Senior_Project

April 19, 2022

[ ]:
```
# Senior Project 2022
# Natalie Rawlings
```

[1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from scipy import stats
import plotly.express as px
```

[2]:
```
import os
```

[3]:
```
pwd
```

[3]:
```
'/Users/chossack/Documents/PMO_Utilities'
```

[2]:
```
# read in excel file (subject data)
df = pd.read_excel(r'/Users/chossack/Documents/PMO_Utilities/Subject Data_rev1.
 ↪xlsx')
```

[99]:
```
df.columns
```

[99]:
```
Index(['Subject_Number', 'Age', 'Height', 'Weight', 'Sex', 'Shoe_size',
       'Gender', 'Orthotics', 'Physical_Activity', 'Sports_highest_level',
       'LESI', 'Reason_for_visit', 'LFA', 'RFA', 'Notes'],
      dtype='object')
```

[3]:
```
# drop Notes column as it will not be used in this analysis
df = df.drop(columns=['Notes','Subject_Number'], axis=1)
```

[32]:
```
# understand and confirm data types
df.dtypes
```

[32]:
```
Age                     int64
Height                  int64
Weight                  int64
```

```
Sex                       object
Shoe_size                float64
Gender                    object
Orthotics                 object
Physical_Activity          int64
Sports_highest_level       int64
LESI                      object
Reason_for_visit           int64
LFA                        int64
RFA                        int64
dtype: object
```

[33]:
```
df.head()
```

[33]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics	Physical_Activity	Sports_highest_level	LESI	Reason_for_visit	LFA	RFA
0	3	2	1	F	8.5	F	N	3	3	N	2	0	0
1	1	3	1	F	9.0	F	N	4	3	Y	4	34	29
2	1	4	4	M	10.0	M	N	5	3	Y	4	28	28
3	4	3	4	M	10.0	M	N	3	4	Y	4	35	36
4	1	5	5	M	13.0	M	N	4	4	Y	1	36	36

[4]:
```
# drop first row since it does not have values for LFA and RFA
df = df.drop(0)
```

[35]:
```
df.head()
```

[35]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics	Physical_Activity	Sports_highest_level	LESI	Reason_for_visit	LFA	RFA
1	1	3	1	F	9.0	F	N	4	3	Y	4	34	29
2	1	4	4	M	10.0	M	N	5	3	Y	4	28	28
3	4	3	4	M	10.0	M	N	3	4	Y	4	35	36
4	1	5	5	M	13.0	M	N	4	4	Y	1	36	36
5	4	3	4	M	10.0	M	Y	5	3	Y	3	31	28

[36]:
```
# check for NA's
df.isna().sum()
```

[36]:
```
Age                    0
Height                 0
Weight                 0
Sex                    0
Shoe_size              0
Gender                 0
Orthotics              0
Physical_Activity      0
Sports_highest_level   0
LESI                   0
Reason_for_visit       0
LFA                    0
RFA                    0
dtype: int64
```

[37]:
```
# correlation coefficient matrix table (default is Pearson but can also run
# Kendall Tau and Spearman)
df.corr()
```

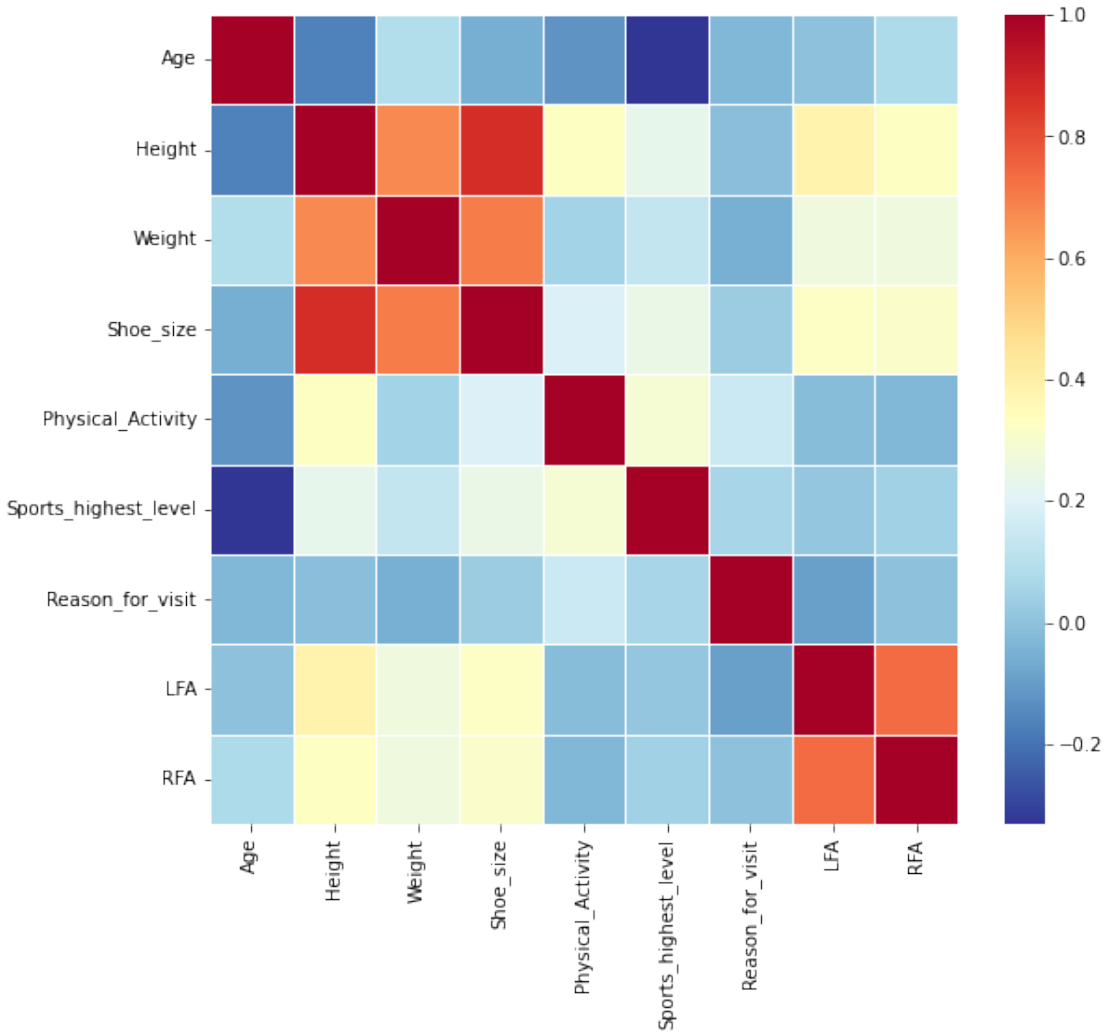
[37]:

	Age	Height	Weight	Shoe_size	Physical_Activity	Sports_highest_level	Reason_for_visit	LFA	RFA
Age	1.000000	-0.169999	0.087631	-0.057526	-0.127895	-0.331936	-0.031546	-0.004108	0.074074
Height	-0.169999	1.000000	0.672558	0.875002	0.327420	0.230546	-0.010484	0.383997	0.325227
Weight	0.087631	0.672558	1.000000	0.703036	0.051007	0.126842	-0.055566	0.261869	0.264853
Shoe_size	-0.057526	0.875002	0.703036	1.000000	0.188768	0.244735	0.027781	0.319753	0.308488
Physical_Activity	-0.127895	0.327420	0.051007	0.188768	1.000000	0.291578	0.150729	-0.018329	-0.032035
Sports_highest_level	-0.331936	0.230546	0.126842	0.244735	0.291578	1.000000	0.060472	0.011567	0.044285
Reason_for_visit	-0.031546	-0.010484	-0.055566	0.027781	0.150729	0.060472	1.000000	-0.018329	-0.032035
LFA	-0.004108	0.383997	0.261869	0.319753	-0.018329	0.011567	-0.018329	1.000000	0.044285
RFA	0.074074	0.325227	0.264853	0.308488	-0.032035	0.044285	-0.032035	0.044285	1.000000

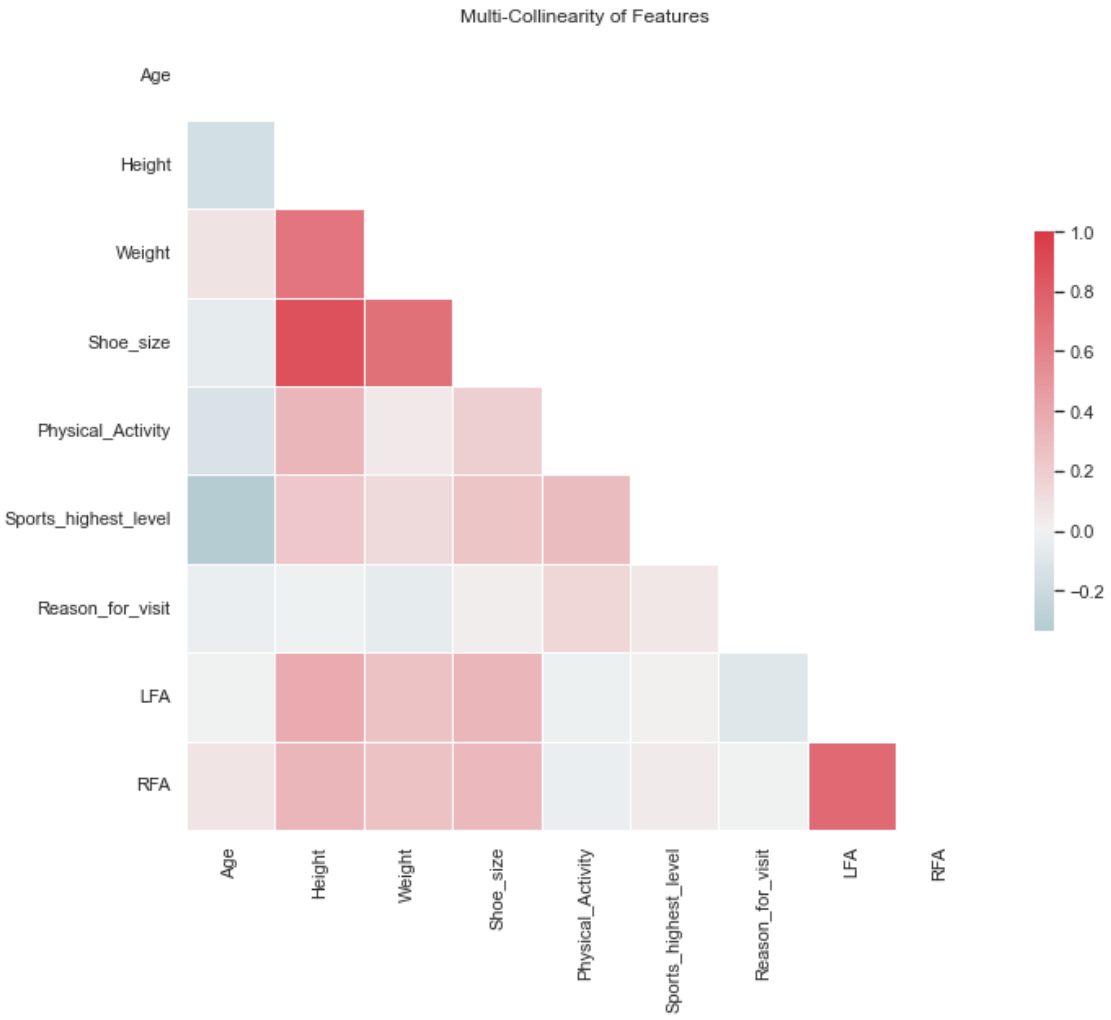
```
Weight                    -0.055566  0.261869  0.264853
Shoe_size                  0.027781  0.319753  0.308488
Physical_Activity          0.150729 -0.018329 -0.032035
Sports_highest_level       0.060472  0.011567  0.044285
Reason_for_visit           1.000000 -0.096921 -0.002009
LFA                       -0.096921  1.000000  0.738120
RFA                       -0.002009  0.738120  1.000000
```

[38]:
```
# correlation matrix heat map
corrmat = df.corr()
f, ax = plt.subplots(figsize =(9, 8))
sns.heatmap(corrmat, ax = ax, cmap = 'RdYlBu_r', linewidths = 0.5)
```

[38]: <AxesSubplot:>



[39]:
```
# another option for correlation matrix
sns.set(style="white")

#create a covariance matrix
corr = df.corr()

# creating a mask the size of our covariance matrix
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom divergin colormap
cmap = sns.diverging_palette(220,10,as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, center=0, square=True,
            linewidth=.5, cbar_kws={'shrink': .5})

ax.set_title("Multi-Collinearity of Features")
plt.savefig("correlation2.png")
```



[86]:
```
df.shape
```

[86]: (74, 13)

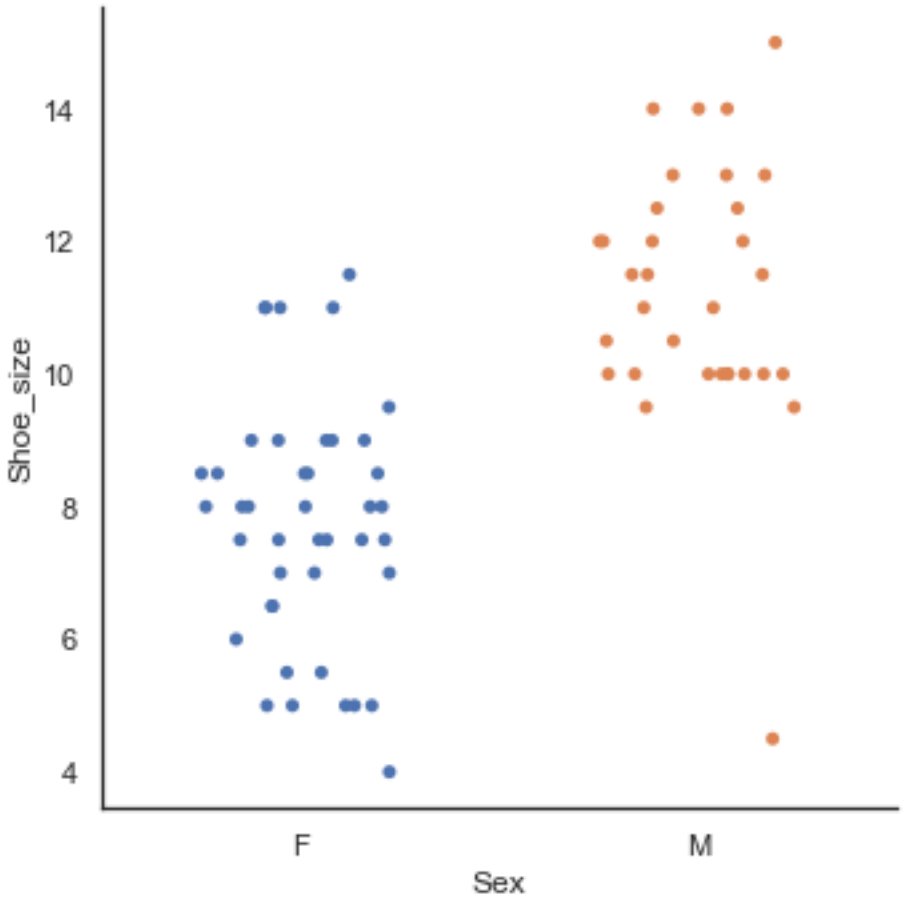
[41]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74 entries, 1 to 74
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Age                   74 non-null     int64
 1   Height                74 non-null     int64
 2   Weight                74 non-null     int64
 3   Sex                   74 non-null     object
 4   Shoe_size             74 non-null     float64
```

```
 5   Gender                 74 non-null     object
 6   Orthotics              74 non-null     object
 7   Physical_Activity      74 non-null     int64
 8   Sports_highest_level   74 non-null     int64
 9   LESI                   74 non-null     object
 10  Reason_for_visit       74 non-null     int64
 11  LFA                    74 non-null     int64
 12  RFA                    74 non-null     int64
dtypes: float64(1), int64(8), object(4)
memory usage: 8.1+ KB
```

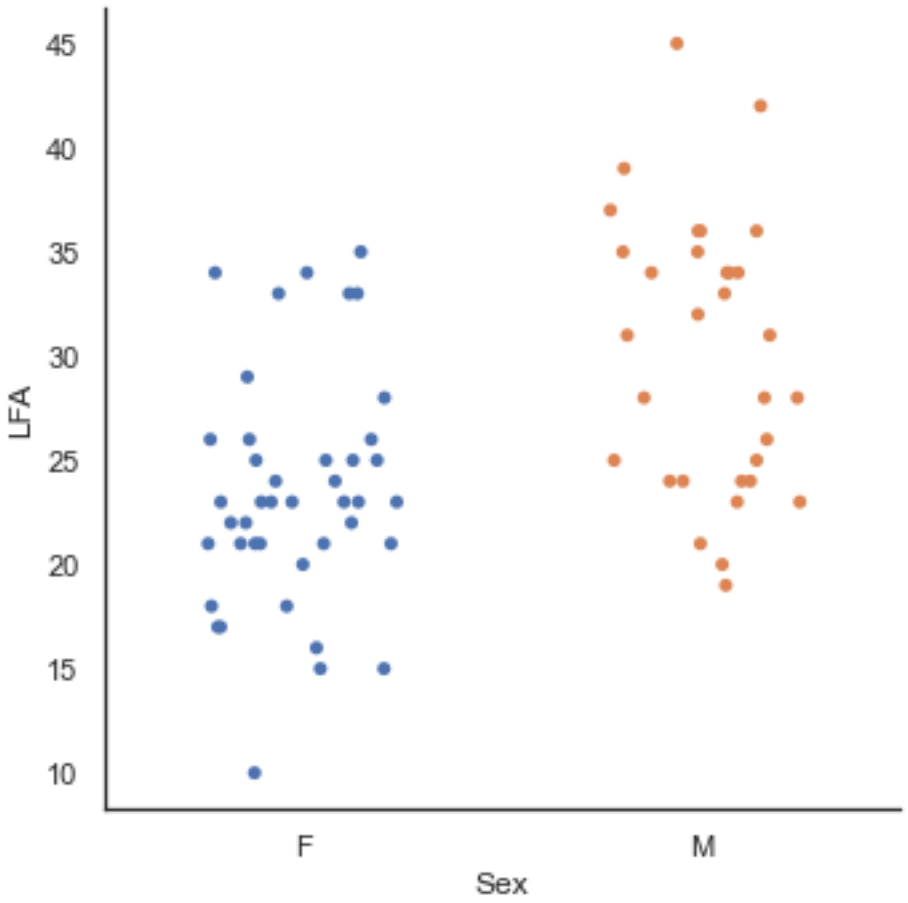
[42]:
```
# eda - gender
sns.catplot(x='Sex',y='Shoe_size',data=df,jitter='0.25')
```

[42]: <seaborn.axisgrid.FacetGrid at 0x7fe9f03d7fd0>



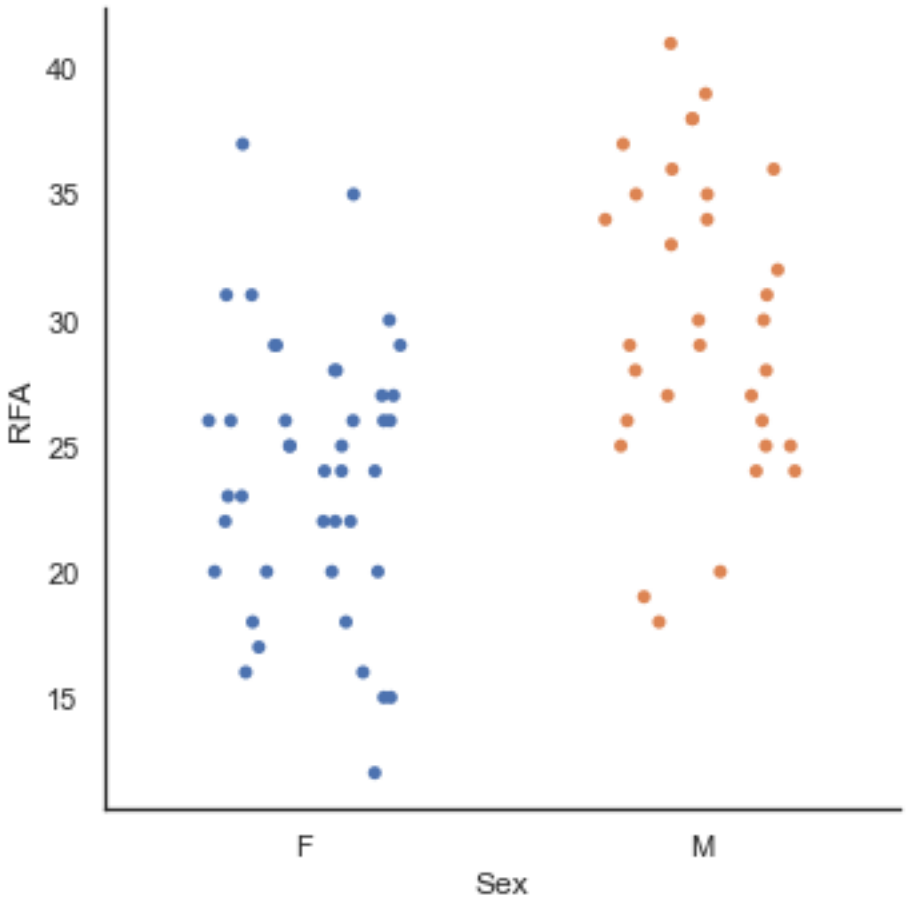
[43]:
```
sns.catplot(x='Sex',y='LFA',data=df,jitter='0.25')
```

[43]: <seaborn.axisgrid.FacetGrid at 0x7fe9980d2fd0>



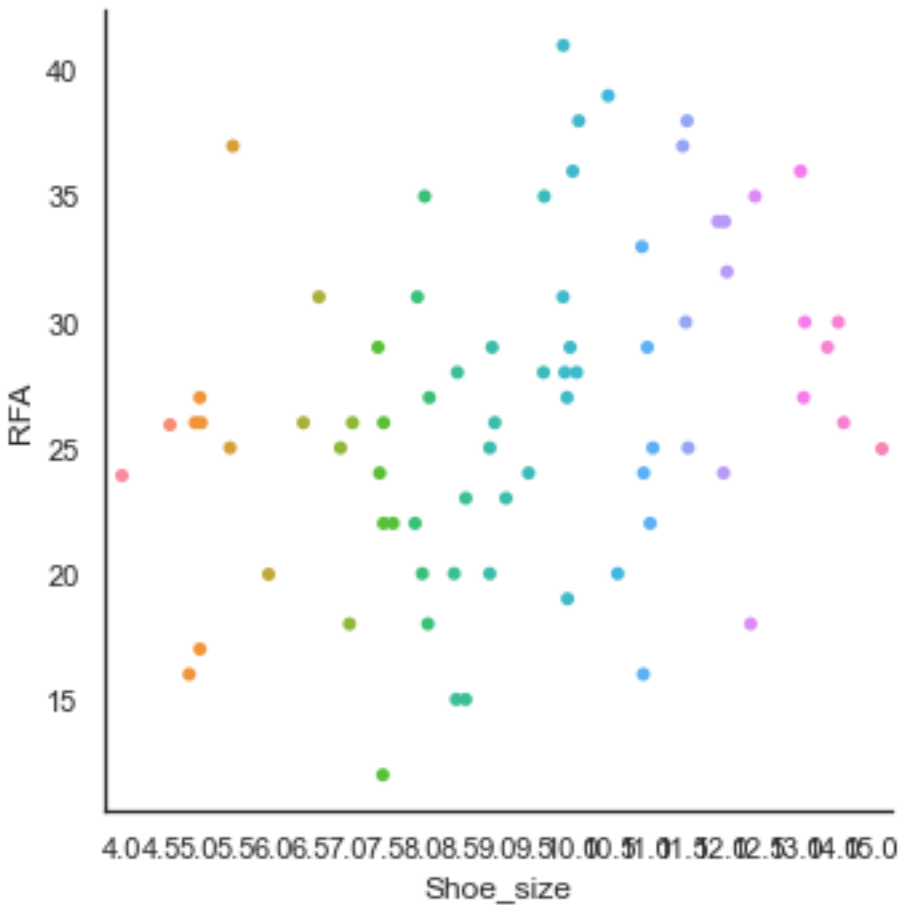
[44]:
```
sns.catplot(x='Sex',y='RFA',data=df,jitter='0.25')
```

[44]: <seaborn.axisgrid.FacetGrid at 0x7fe9a1e44a90>



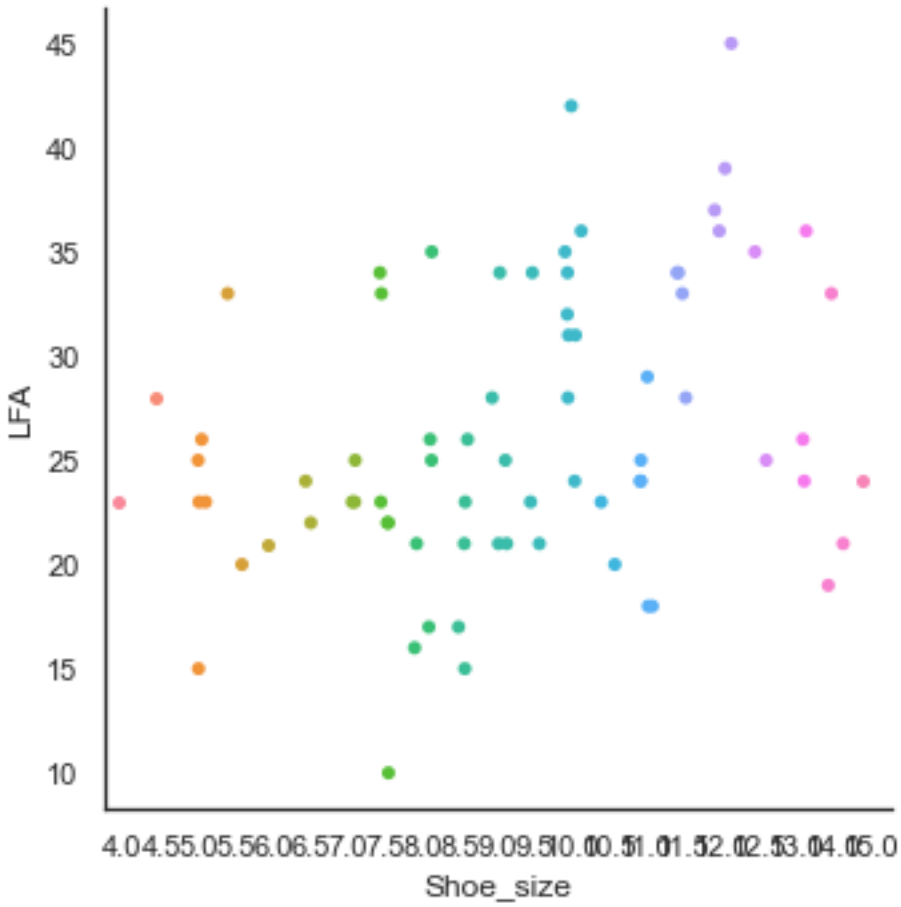
[45]:
```
sns.catplot(x='Shoe_size',y='RFA',data=df,jitter='0.25')
```

[45]:
```
<seaborn.axisgrid.FacetGrid at 0x7fe9e2fcf7f0>
```



[46]:
```
sns.catplot(x='Shoe_size',y='LFA',data=df,jitter='0.25')
```

[46]: <seaborn.axisgrid.FacetGrid at 0x7fe9e320c970>



[102]:
```
# include categorical variables
df.describe(include = 'all')
```

[102]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics
count	74.000000	74.000000	74.000000	74	74.000000	74	74
unique	NaN	NaN	NaN	2	NaN	2	2
top	NaN	NaN	NaN	F	NaN	F	N
freq	NaN	NaN	NaN	42	NaN	41	55
mean	3.310811	3.027027	3.216216	NaN	9.263514	NaN	NaN
std	1.489007	1.170095	1.367632	NaN	2.568544	NaN	NaN
min	1.000000	1.000000	1.000000	NaN	4.000000	NaN	NaN
25%	2.000000	2.000000	2.000000	NaN	7.500000	NaN	NaN
50%	4.000000	3.000000	3.000000	NaN	9.250000	NaN	NaN
75%	5.000000	4.000000	4.000000	NaN	11.000000	NaN	NaN
max	5.000000	5.000000	5.000000	NaN	15.000000	NaN	NaN

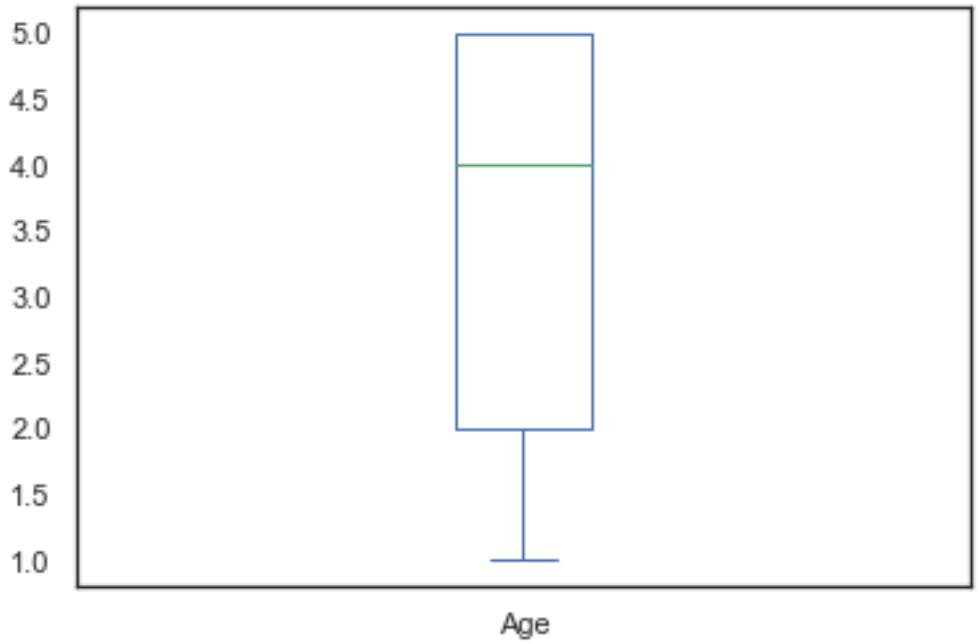
	Physical_Activity	Sports_highest_level	LESI	Reason_for_visit
count	74.000000	74.000000	74	74.000000

unique	NaN	NaN	2	NaN
top	NaN	NaN	Y	NaN
freq	NaN	NaN	43	NaN
mean	4.121622	1.567568	NaN	3.054054
std	0.992474	1.414999	NaN	1.237451
min	1.000000	0.000000	NaN	0.000000
25%	4.000000	0.000000	NaN	2.000000
50%	4.000000	2.000000	NaN	3.000000
75%	5.000000	3.000000	NaN	4.000000
max	5.000000	5.000000	NaN	5.000000

	LFA	RFA
count	74.000000	74.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	26.351351	26.540541
std	6.899340	6.404598
min	10.000000	12.000000
25%	22.000000	22.250000
50%	25.000000	26.000000
75%	33.000000	30.000000
max	45.000000	41.000000

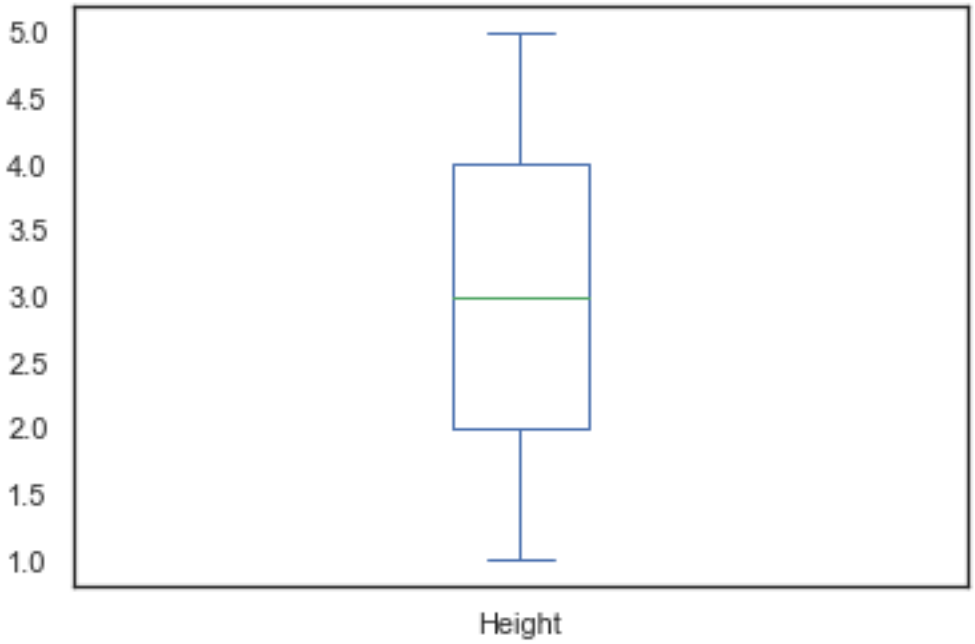
[103]:
```
df.Age.plot(kind='box')
```

[103]: <AxesSubplot:>



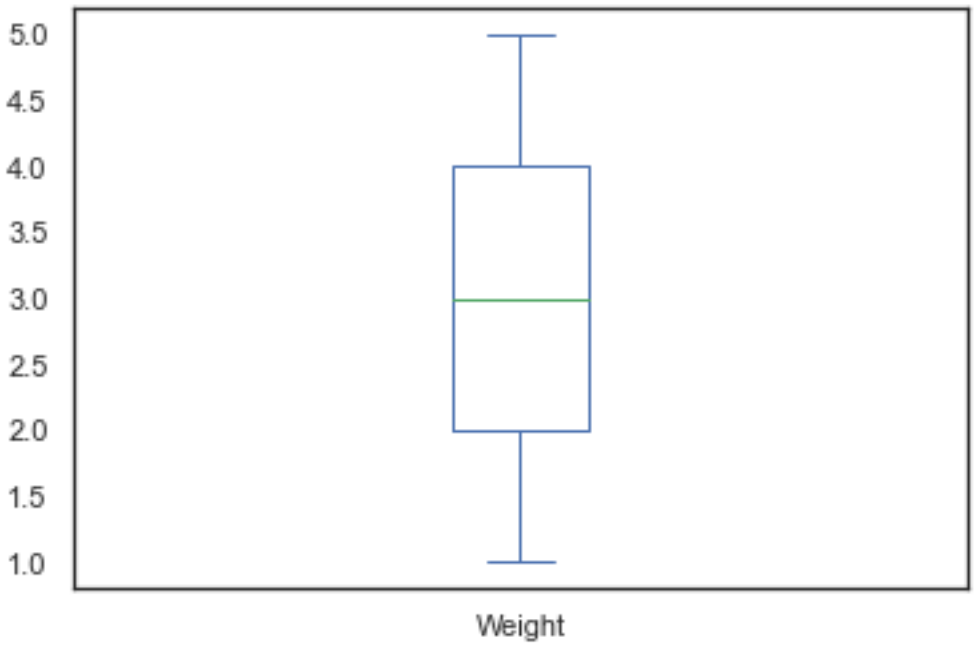
[104]:
```
df.Height.plot(kind='box')
```

[104]: <AxesSubplot:>



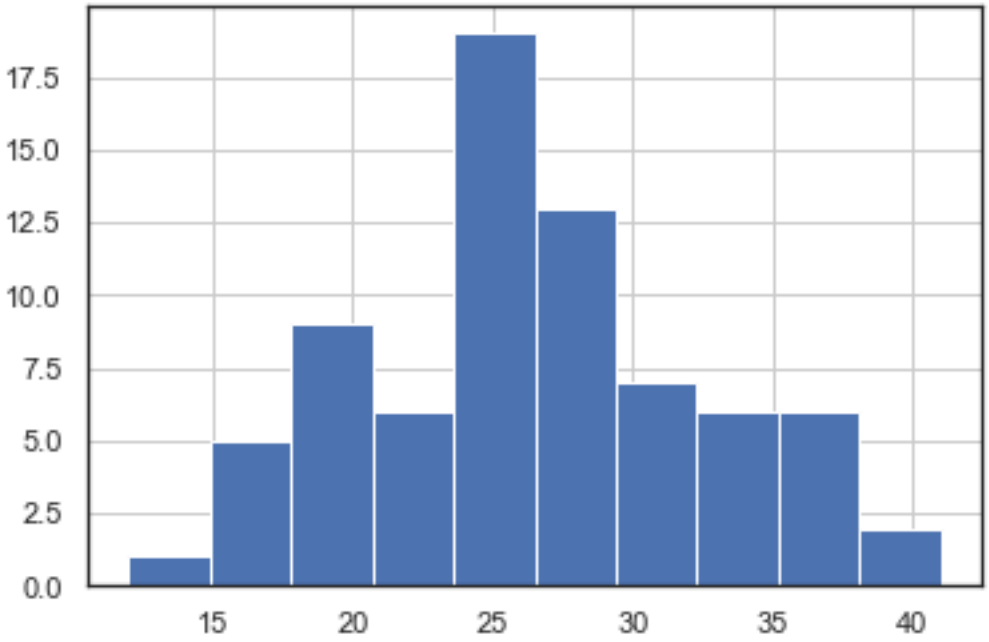
[105]:
```
df.Weight.plot(kind='box')
```

[105]: <AxesSubplot:>



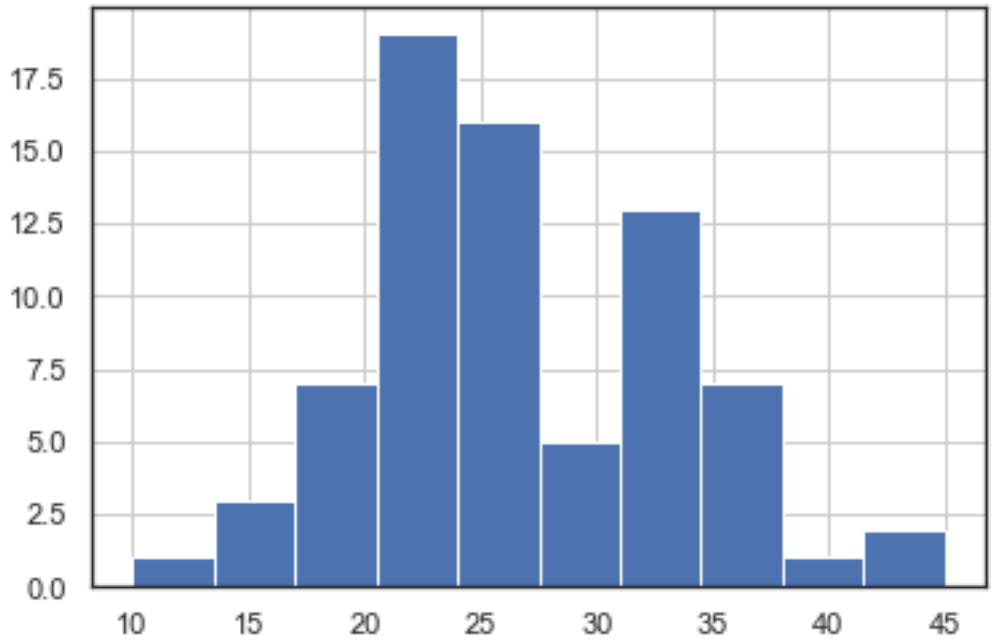
[51]:
```
df.RFA.hist()
```

[51]: <AxesSubplot:>



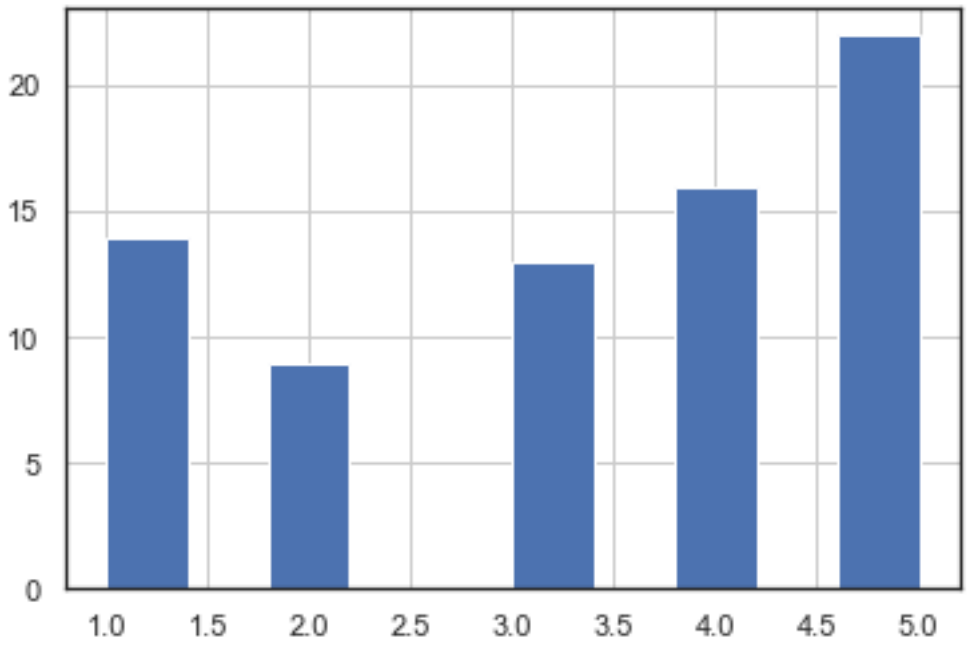
[52]:
```
df.LFA.hist()
```

[52]: <AxesSubplot:>



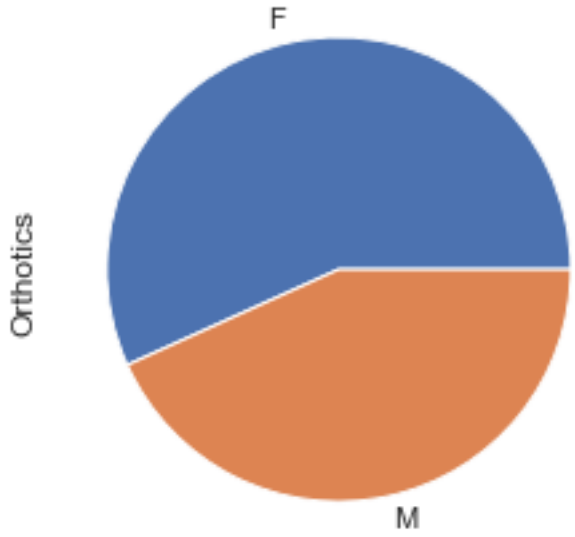
[53]:
```
df.Age.hist()
```

[53]: <AxesSubplot:>



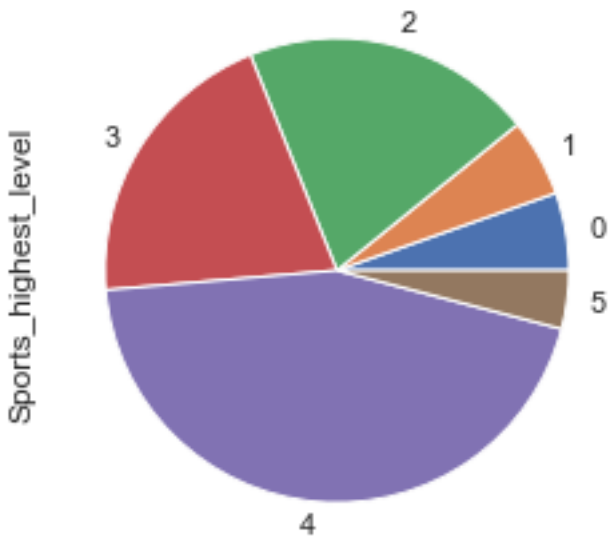
[54]:
```
df.groupby('Sex').Orthotics.count().plot(kind='pie')
```

[54]: <AxesSubplot:ylabel='Orthotics'>



[55]:
```
df.groupby('Reason_for_visit').Sports_highest_level.count().plot(kind='pie')
```

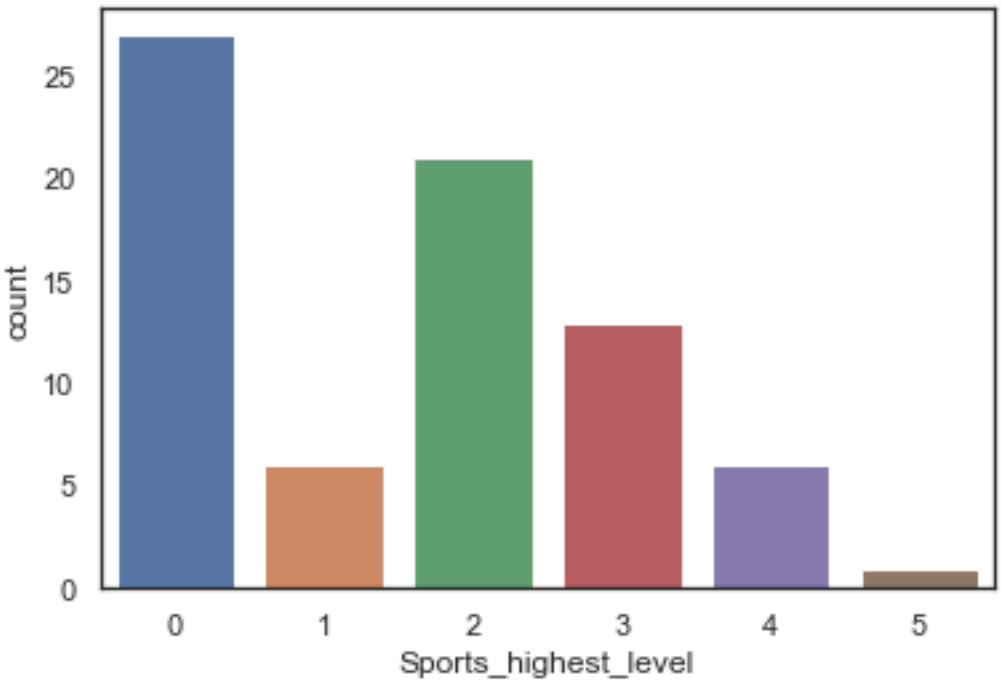
[55]: <AxesSubplot:ylabel='Sports_highest_level'>



[56]:
```
sns.countplot(df.Sports_highest_level)
```

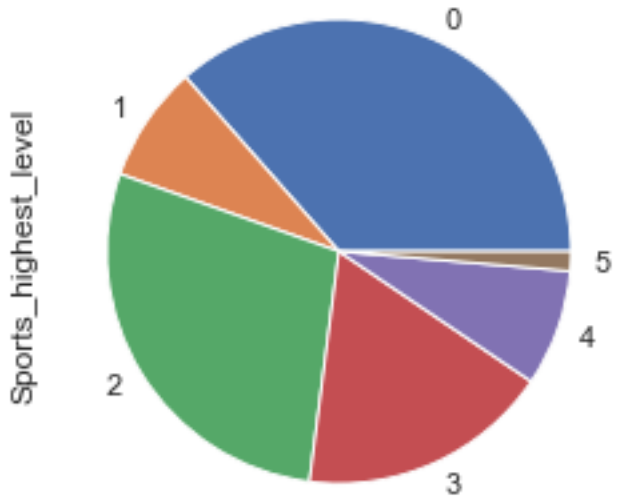
```
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

[56]: <AxesSubplot:xlabel='Sports_highest_level', ylabel='count'>



[57]:
```
df.groupby('Sports_highest_level').Sports_highest_level.count().plot(kind='pie')
```

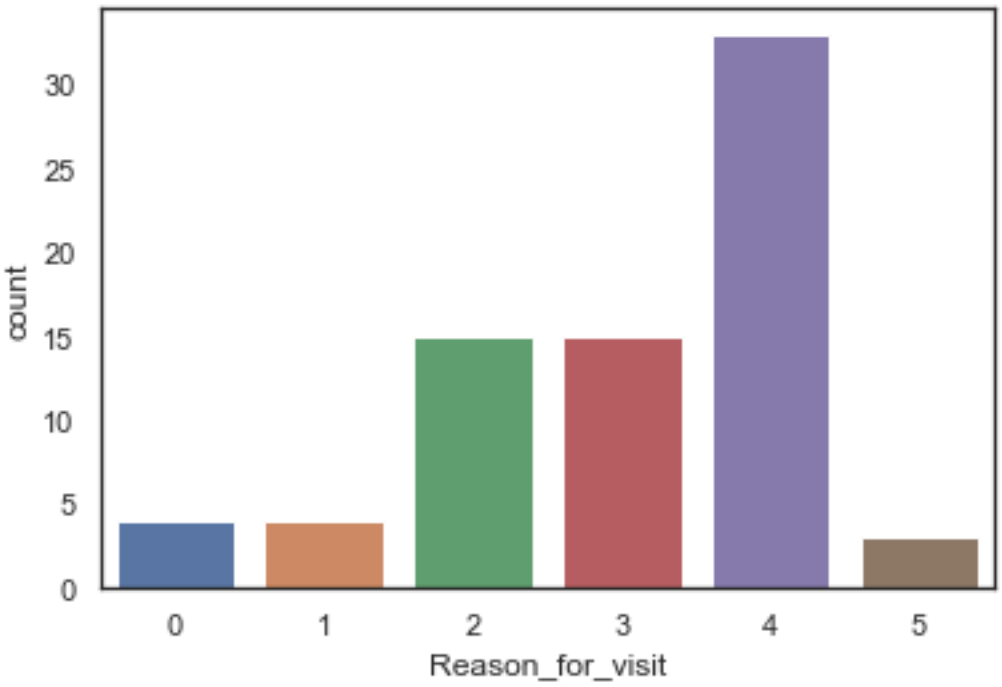
[57]: <AxesSubplot:ylabel='Sports_highest_level'>



[58]:
```
sns.countplot(df.Reason_for_visit)
```

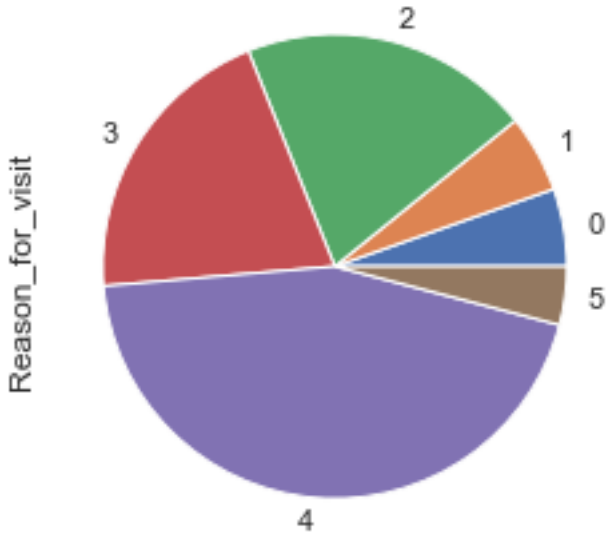
```
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

[58]: <AxesSubplot:xlabel='Reason_for_visit', ylabel='count'>



[59]:
```
df.groupby('Reason_for_visit').Reason_for_visit.count().plot(kind='pie')
```

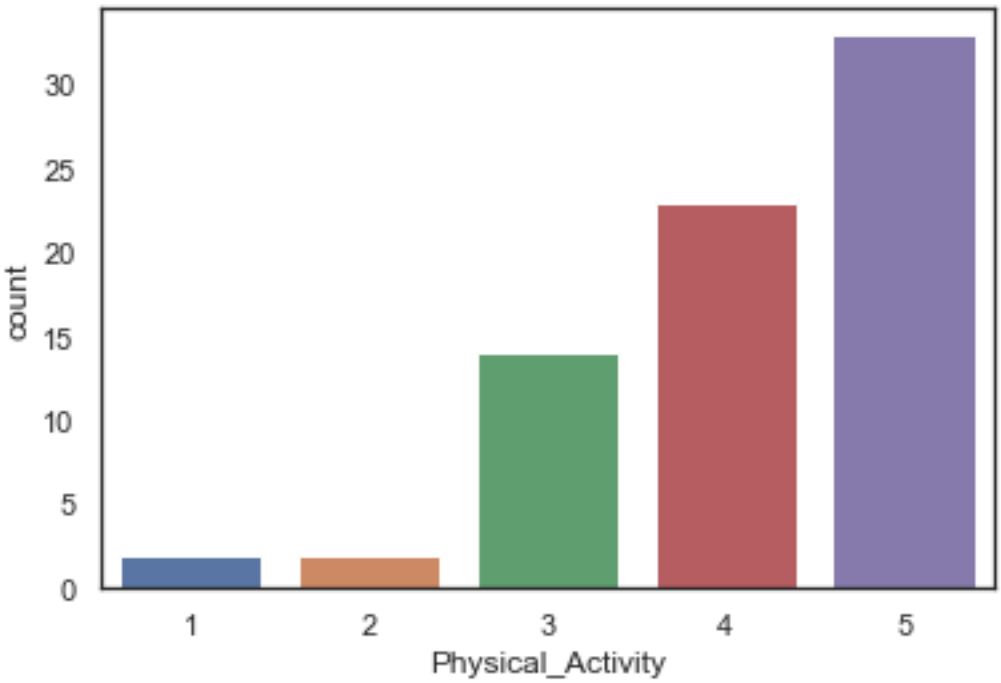
[59]: <AxesSubplot:ylabel='Reason_for_visit'>



[60]:
```
sns.countplot(df.Physical_Activity)
```

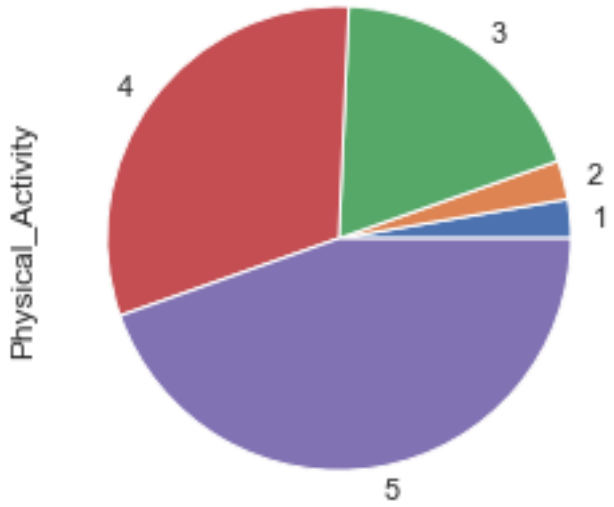
```
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

[60]:
```
<AxesSubplot:xlabel='Physical_Activity', ylabel='count'>
```



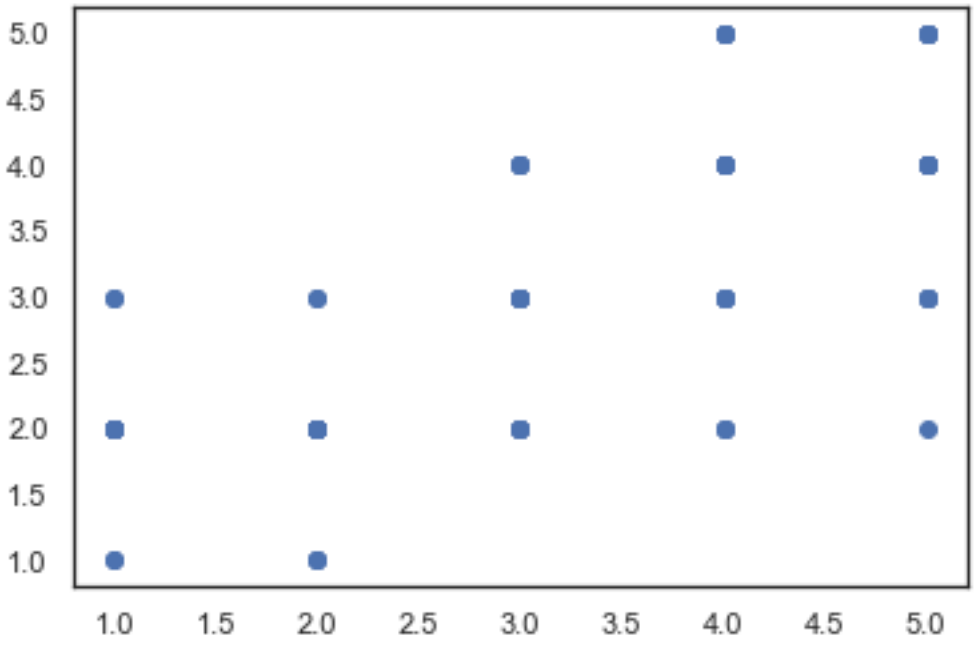
[61]:
```
df.groupby('Physical_Activity').Physical_Activity.count().plot(kind='pie')
```

[61]: <AxesSubplot:ylabel='Physical_Activity'>



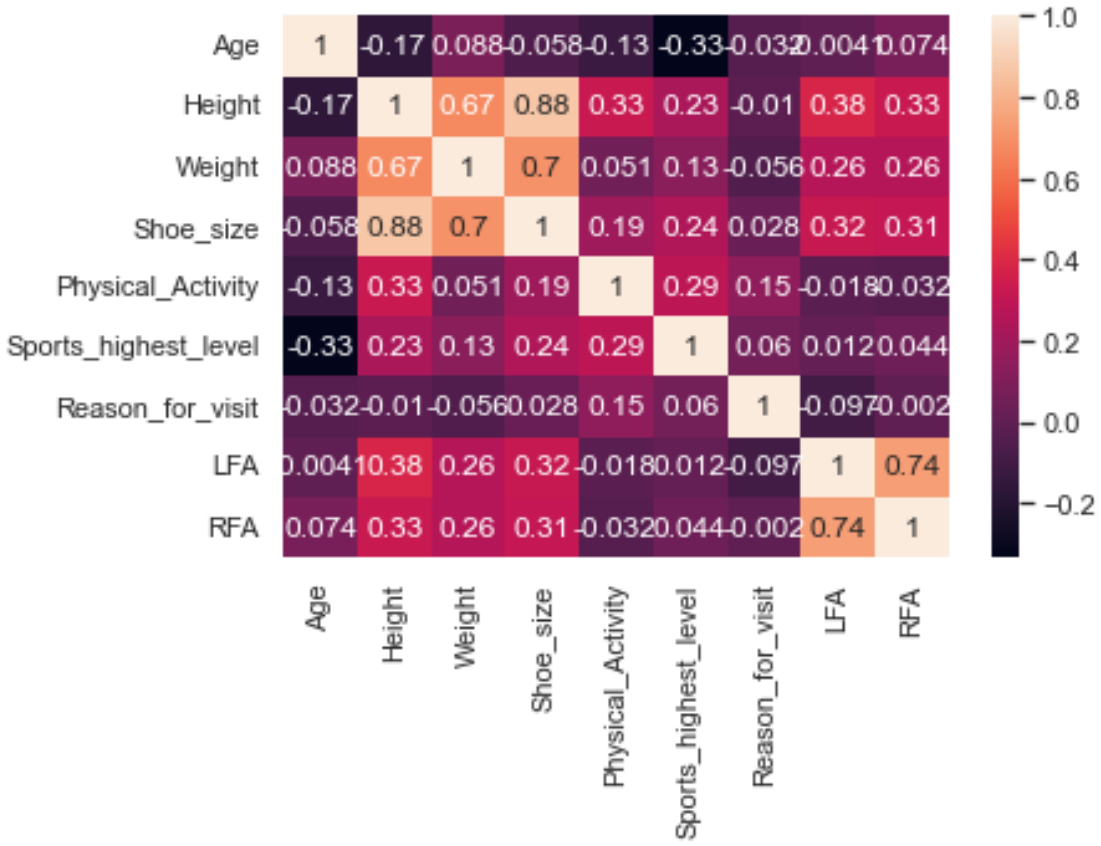
[62]:
```
plt.scatter(x=df.Weight,y=df.Height)
```

[62]:
```
<matplotlib.collections.PathCollection at 0x7fe9a24c6dc0>
```



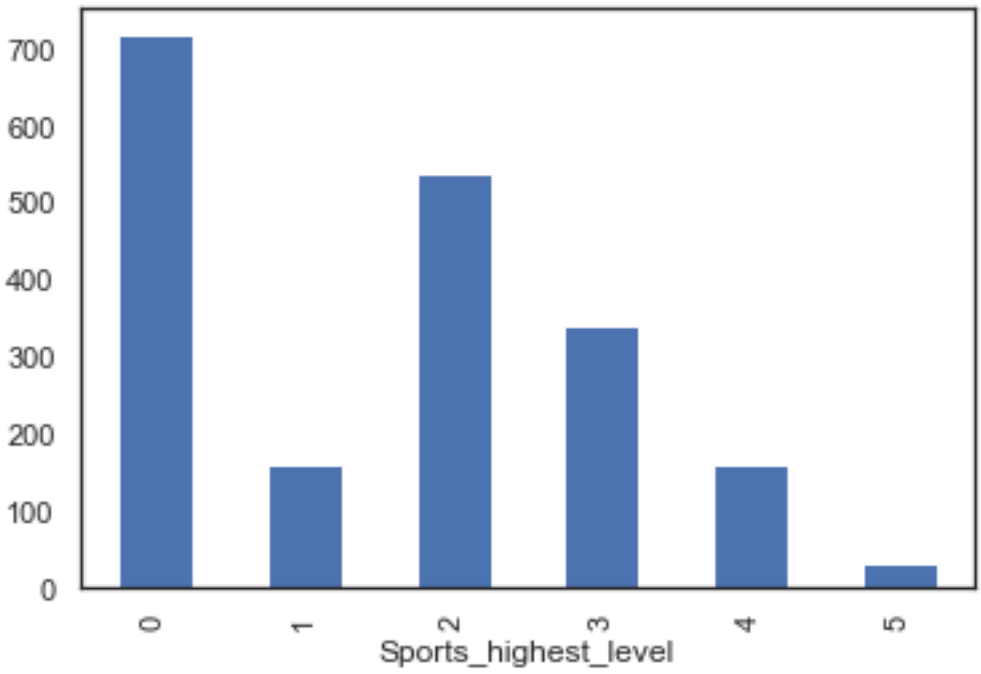
[63]:
```
sns.heatmap(df.select_dtypes(['float64', 'int64']).corr(),annot=True)
```

[63]:
```
<AxesSubplot:>
```



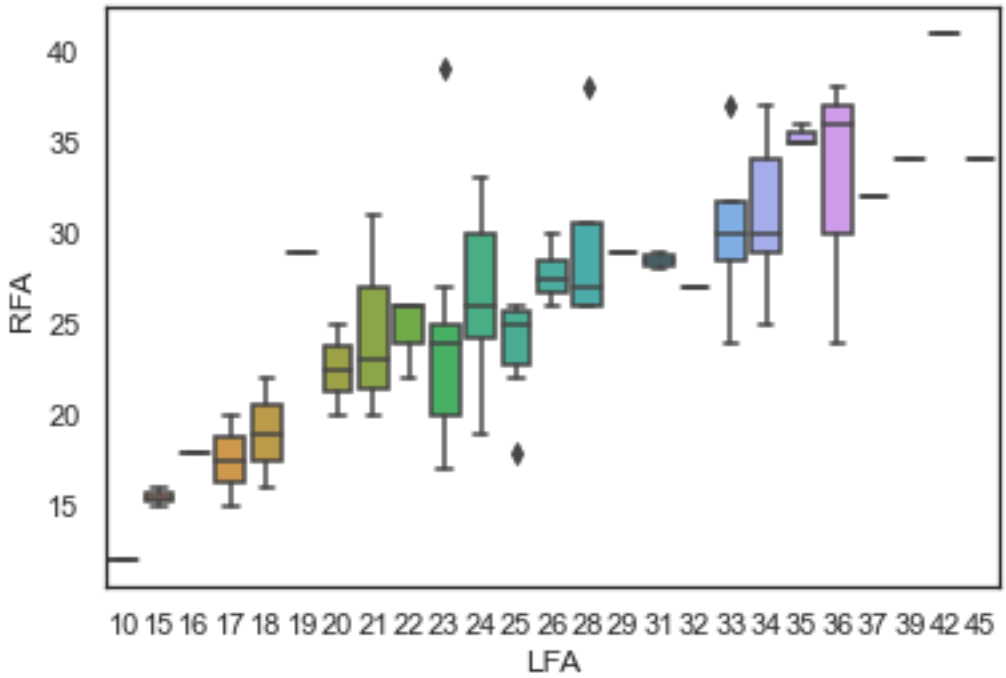
[64]:
```
df.groupby('Sports_highest_level').LFA.sum().plot(kind='bar')
```

[64]: <AxesSubplot:xlabel='Sports_highest_level'>



[65]:
```
sns.boxplot(x='LFA',y='RFA',data=df)
```

[65]: <AxesSubplot:xlabel='LFA', ylabel='RFA'>

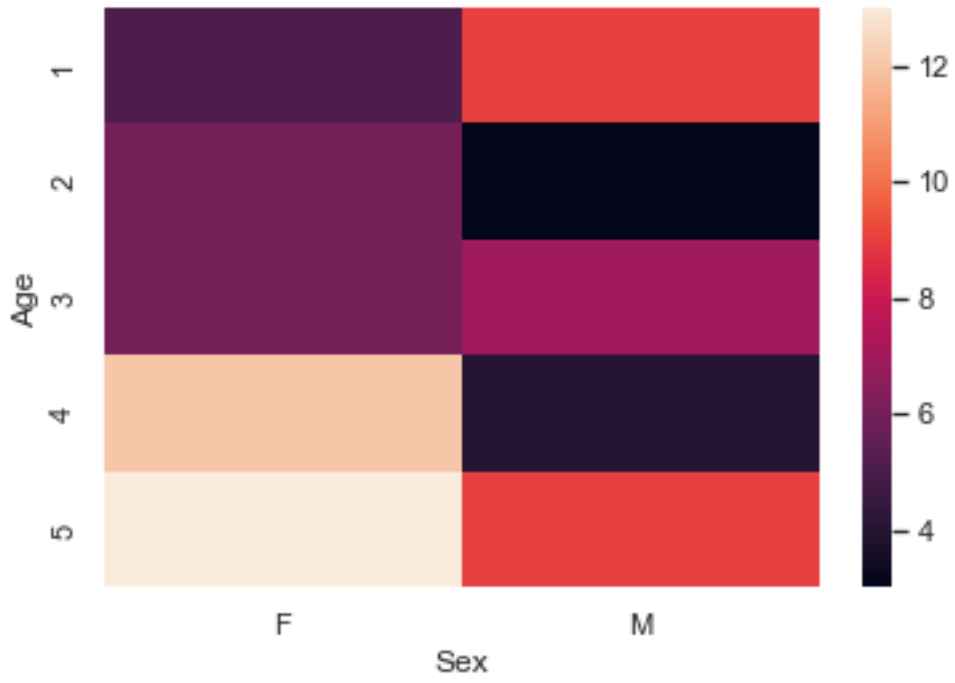


[66]:
```
pd.crosstab(df.LESI,df.Sex)
```

[66]:
```
Sex     F   M
LESI
N      19  12
Y      23  20
```

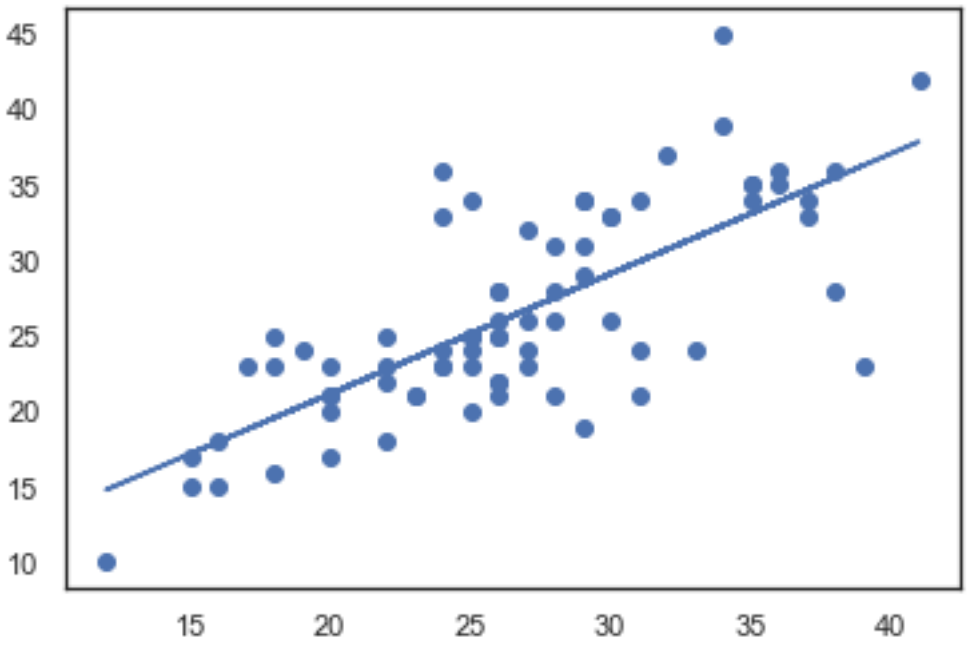
[67]:
```
sns.heatmap(pd.crosstab(df.Age,df.Sex))
```

[67]:
```
<AxesSubplot:xlabel='Sex', ylabel='Age'>
```



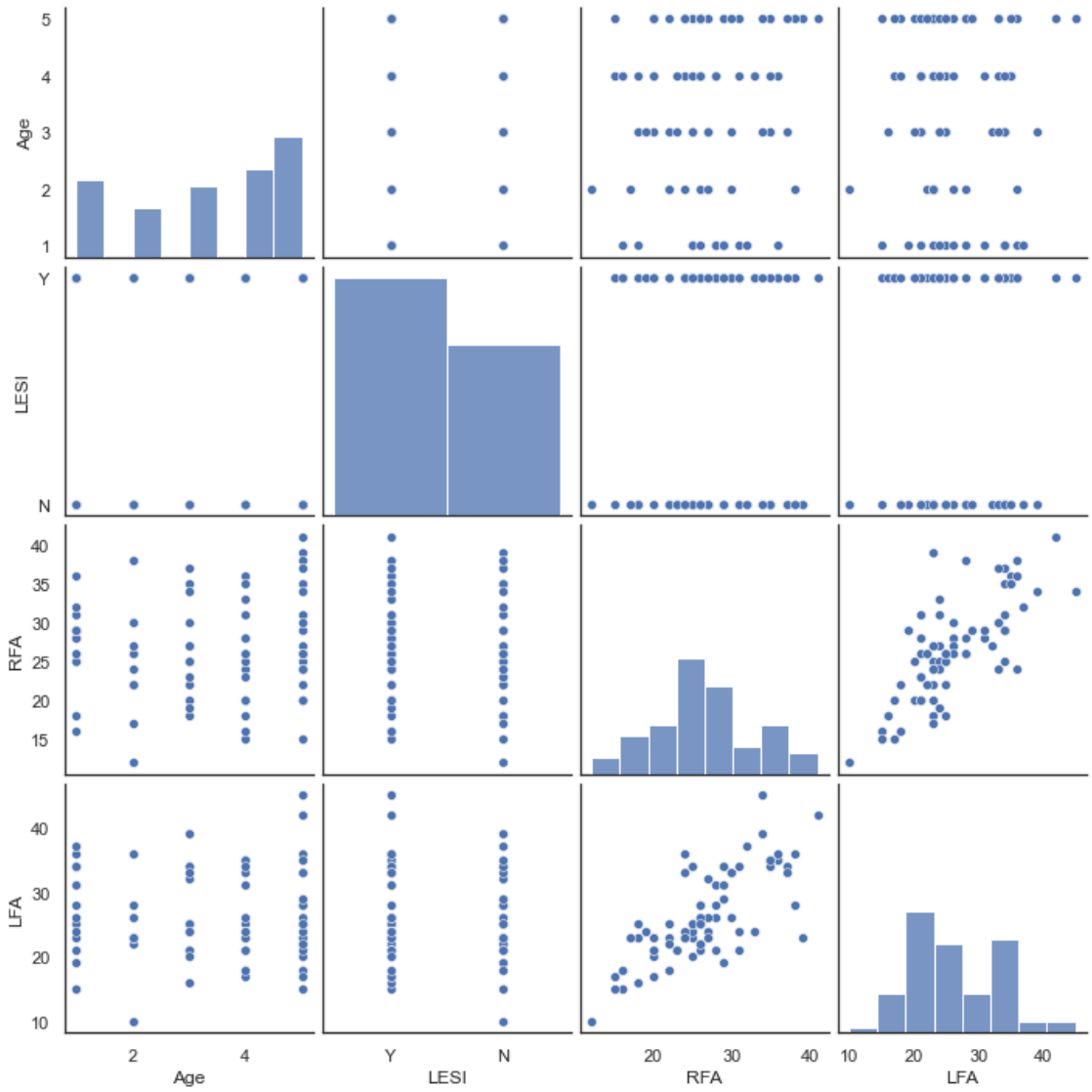
[87]:
```
plt.scatter(df.RFA,df.LFA)

# check numpy version
z = np.polyfit(df.RFA,df.LFA,1)
p = np.poly1d(z)

plt.plot(df.RFA,p(df.RFA))
```

[87]:
```
[<matplotlib.lines.Line2D at 0x7fe9f1ba0940>]
```



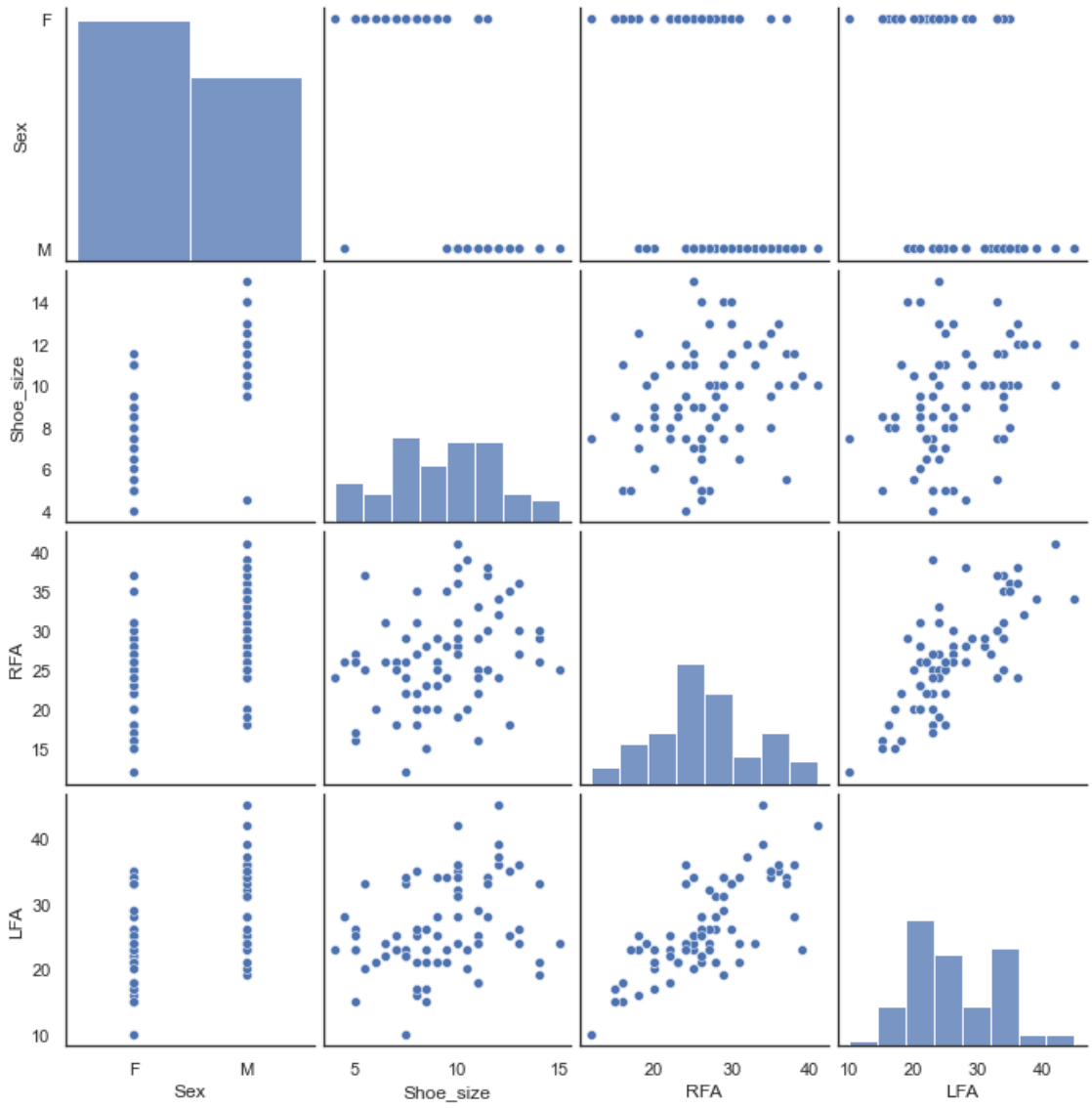
[69]:
```
sns.pairplot(data=df, vars=['Age','LESI','RFA','LFA'])
```

[69]: <seaborn.axisgrid.PairGrid at 0x7fe9d06d37c0>



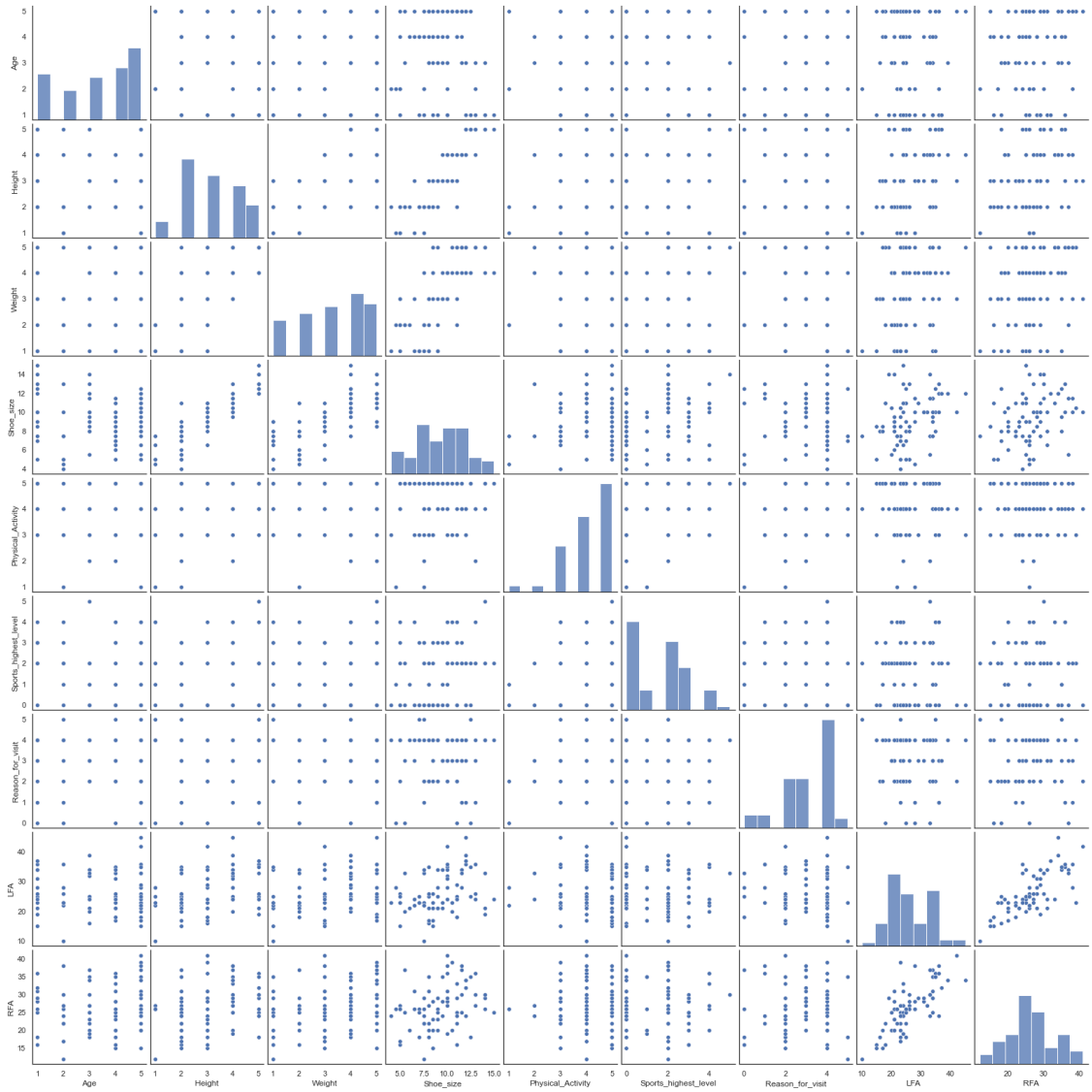
[70]:
```
sns.pairplot(data=df, vars=['Sex','Shoe_size','RFA','LFA'])
```

[70]: <seaborn.axisgrid.PairGrid at 0x7fe9d06cc040>



[71]:
```
sns.pairplot(data=df)
```

[71]: <seaborn.axisgrid.PairGrid at 0x7fe9f0c722e0>



[72]:
```
df['Physical_Activity'].value_counts(normalize=True)
```

[72]:
```
5    0.445946
4    0.310811
3    0.189189
2    0.027027
1    0.027027
Name: Physical_Activity, dtype: float64
```

[73]:
```
df['Sports_highest_level'].value_counts(normalize=True)
```

[73]:
```
0    0.364865
2    0.283784
```

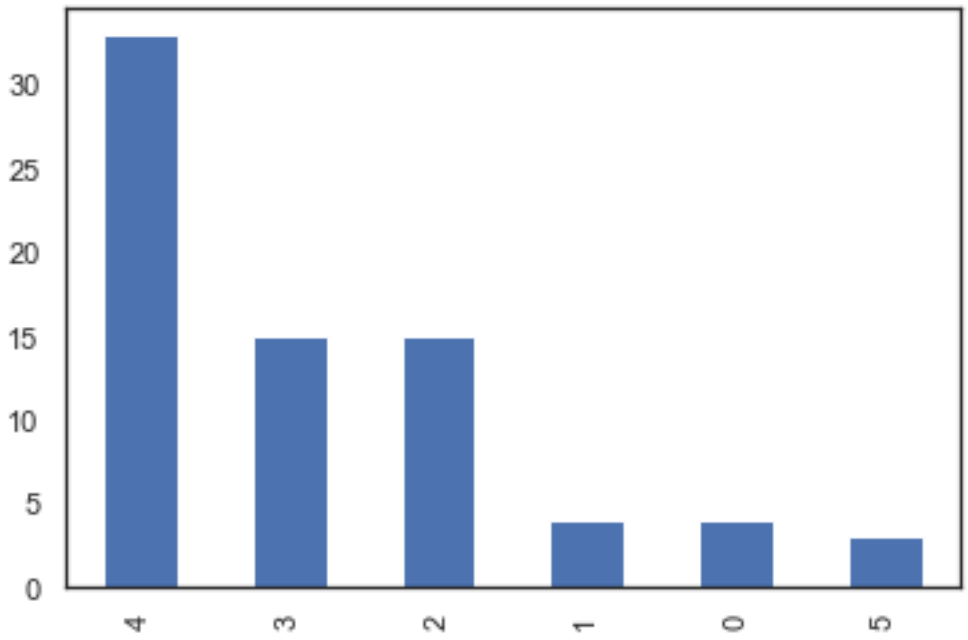
```
3    0.175676
4    0.081081
1    0.081081
5    0.013514
Name: Sports_highest_level, dtype: float64
```

[74]:
```
df['Reason_for_visit'].value_counts(normalize=True)
```

[74]:
```
4    0.445946
3    0.202703
2    0.202703
1    0.054054
0    0.054054
5    0.040541
Name: Reason_for_visit, dtype: float64
```

[75]:
```
df['Reason_for_visit'].value_counts().plot(kind='bar')
```

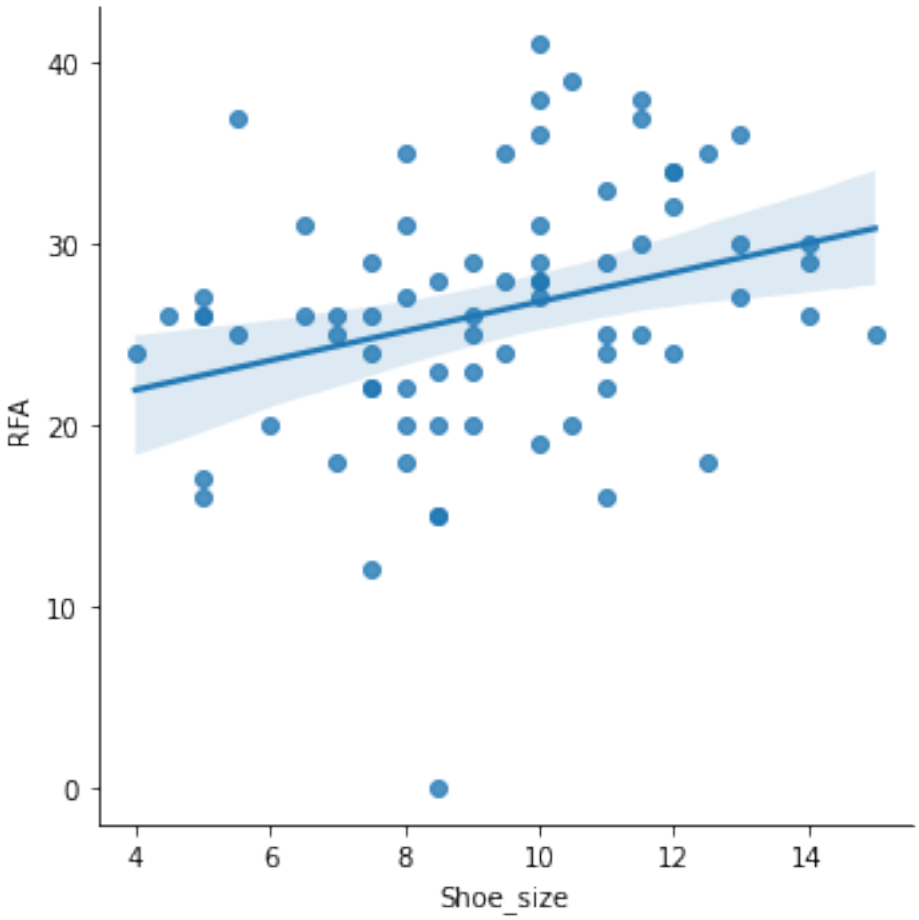
[75]:
```
<AxesSubplot:>
```



[4]:
```
fig = px.scatter(df, x="Shoe_size", y="RFA", trendline="ols",
                 trendline_color_override="red")
fig.show()
```

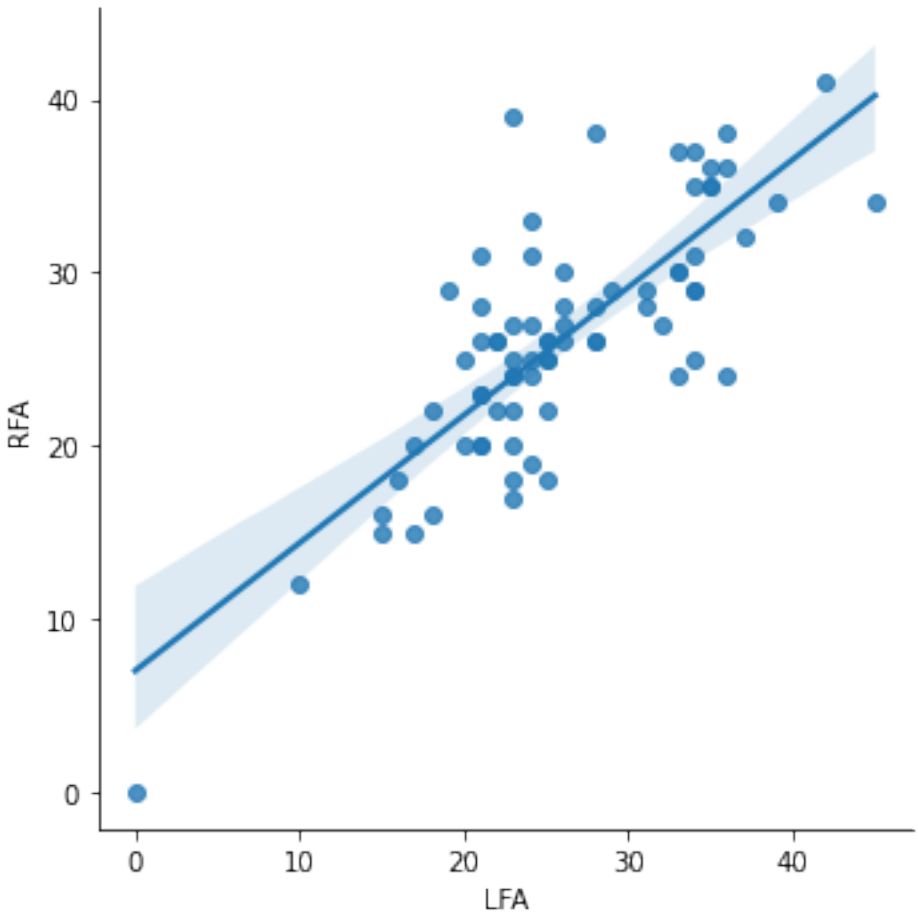
[5]:
```
sns.lmplot(x='Shoe_size',y='RFA', data=df)
```

[5]: <seaborn.axisgrid.FacetGrid at 0x7f98686737f0>



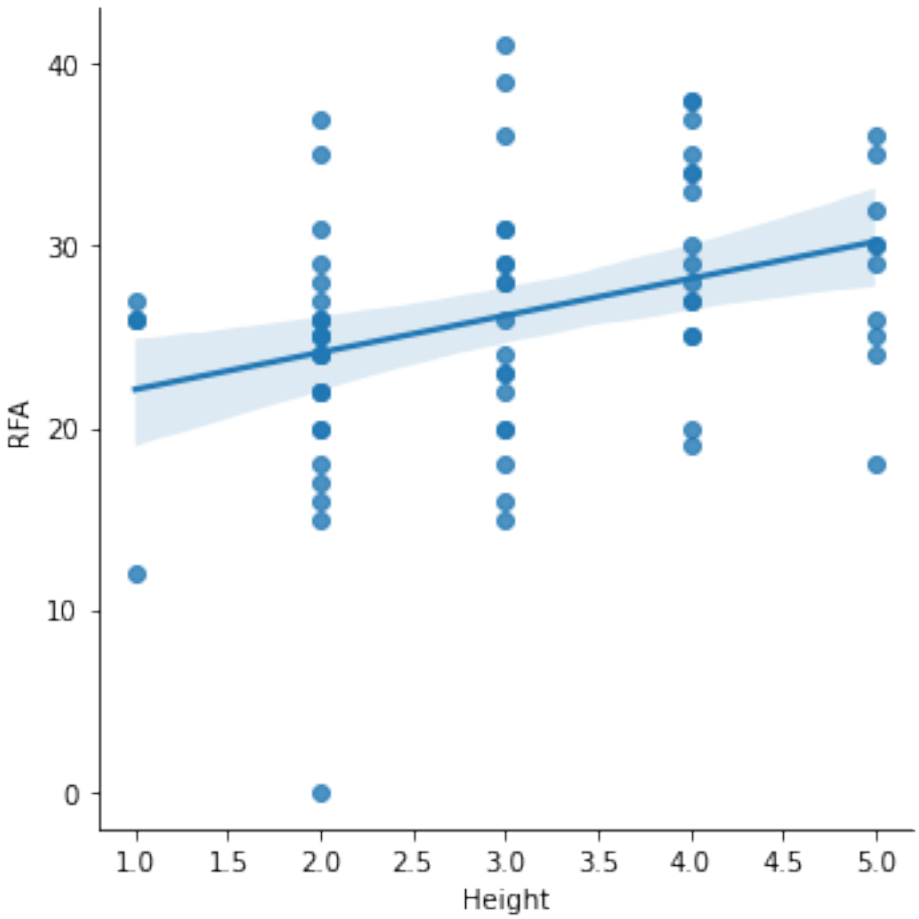
[6]:
```
sns.lmplot(x='LFA',y='RFA', data=df)
```

[6]: <seaborn.axisgrid.FacetGrid at 0x7f9818018b50>



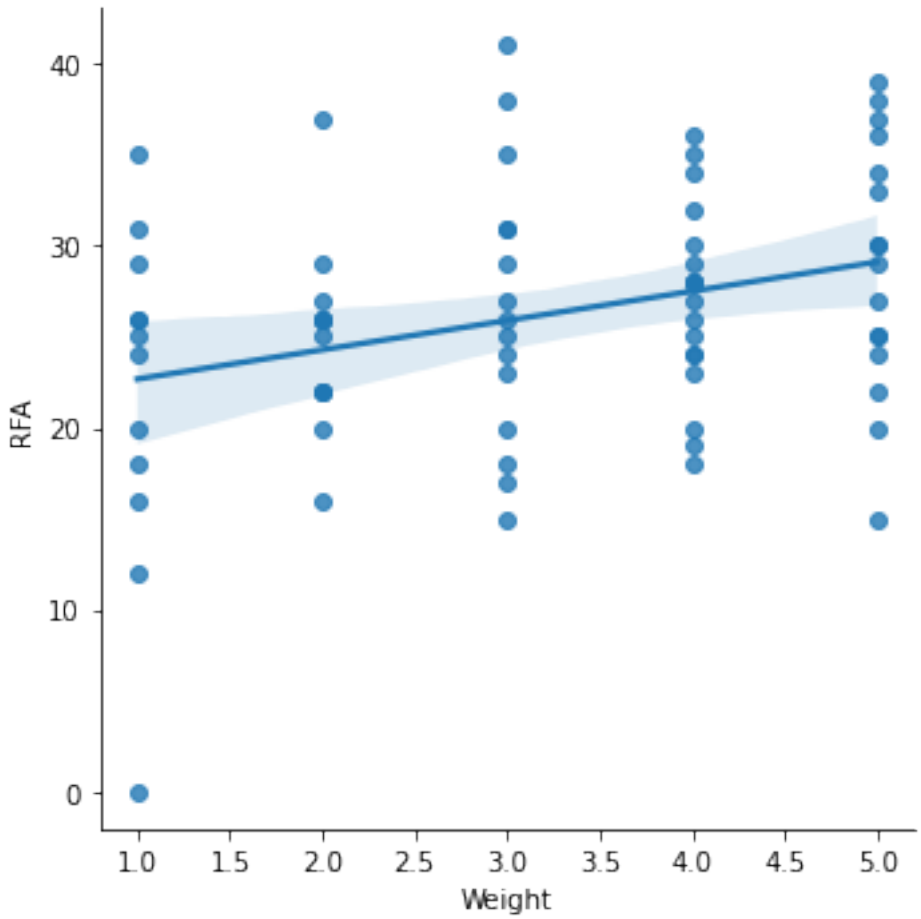
[7]:
```
sns.lmplot(x='Height',y='RFA', data=df)
```

[7]: <seaborn.axisgrid.FacetGrid at 0x7f9818018f40>



[9]:
```
sns.lmplot(x='Weight',y='RFA', data=df)
```

[9]: <seaborn.axisgrid.FacetGrid at 0x7f988ad1d0d0>



[ ]:

[ ]:

1 prepare data for machine learning

[76]:
```
df.Age.value_counts()
```

[76]:
```
5    22
4    16
1    14
3    13
2     9
Name: Age, dtype: int64
```

[5]:
```
# Age:
# 1 = 18-25; 2 = 26-35; 3 = 36-45; 4 = 46-55; 5 = 56-75
```

```
# create initial dataframe
# column index = 1
# isolate age column, save results in new variable, add it back into main df

def age_group(age):

    """Creates an age bucket for each participant using the age variable.
        Meant to be used on a DataFrame with .apply()."""

    # Convert to an int, in case the data is read in as an "object" (aka string)
    age = int(age)

    if age == 1:
        bucket = '18-25'

    # Age 26 to 35
    if age == 2:
        bucket = '26-35'

    if age == 3:
        bucket = '36-45'

    if age == 4:
        bucket = '46-55'

    if age == 5:
        bucket = '56-75'

    return bucket
```

[6]:
```
df['Age_grp'] = df['Age'].apply(age_group)
df['Age_grp'].head()
```

[6]:
```
1    18-25
2    18-25
3    46-55
4    18-25
5    46-55
Name: Age_grp, dtype: object
```

[182]:
```
df.head()
```

[182]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics	Physical_Activity	\
1	1	3	1	F	9.0	F	N	4	
2	1	4	4	M	10.0	M	N	5	
3	4	3	4	M	10.0	M	N	3	
4	1	5	5	M	13.0	M	N	4	

```
5    4       3       4   M       10.0      M         Y                   5

   Sports_highest_level LESI  Reason_for_visit  LFA  RFA Age_grp
1                     3    Y                 4   34   29   18-25
2                     3    Y                 4   28   28   18-25
3                     4    Y                 4   35   36   46-55
4                     4    Y                 1   36   36   18-25
5                     3    Y                 3   31   28   46-55
```

[183]:
```
df.Height.value_counts()
```

[183]:
```
2    24
3    19
4    16
5    10
1     5
Name: Height, dtype: int64
```

[7]:
```
# Height:
# 1 = 4'9-5'1; 2 = 5'2-5'5; 3 = 5'6-5'10; 4 = 5'11-6'1; 5 = 6'2-6'5
# column index = 2

def height_group(height):

    """Creates an age bucket for each participant using the age variable.
        Meant to be used on a DataFrame with .apply()."""

    # Convert to an int, in case the data is read in as an "object" (aka string)
    height = int(height)

    if height == 1:
        bucket = '4-9:5-1'

    if height == 2:
        bucket = '5-2:5-5'

    if height == 3:
        bucket = '5-6:5-10'

    if height == 4:
        bucket = '5-11:6-1'

    if height == 5:
        bucket = '6-2:6-5'

    return bucket
```

[8]:
```
df['Height_grp'] = df['Height'].apply(height_group)
df['Height_grp'].head()
```

[8]:
```
1    5-6:5-10
2    5-11:6-1
3    5-6:5-10
4     6-2:6-5
5    5-6:5-10
Name: Height_grp, dtype: object
```

[186]:
```
df.head()
```

[186]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics	Physical_Activity	Sports_highest_level	LESI	Reason_for_visit	LFA	RFA	Age_grp	Height_grp
1	1	3	1	F	9.0	F	N	4	3	Y	4	34	29	18-25	5-6:5-10
2	1	4	4	M	10.0	M	N	5	3	Y	4	28	28	18-25	5-11:6-1
3	4	3	4	M	10.0	M	N	3	4	Y	4	35	36	46-55	5-6:5-10
4	1	5	5	M	13.0	M	N	4	4	Y	1	36	36	18-25	6-2:6-5
5	4	3	4	M	10.0	M	Y	5	3	Y	3	31	28	46-55	5-6:5-10

[12]:
```
df.Weight.value_counts()
```

[12]:
```
4    18
5    16
3    11
1    10
2     8
Name: Weight, dtype: int64
```

[9]:
```
# Weight:
# 1 = 100-125; 2 = 126-140; 3 = 141-170; 4 = 171-200; 5 = 201-250+
# column index = 3

def weight_group(weight):

    """Creates an age bucket for each participant using the age variable.
        Meant to be used on a DataFrame with .apply()."""

    # Convert to an int, in case the data is read in as an "object" (aka string)
    weight = int(weight)

    if weight == 1:
```

```
        bucket = '100:125'

    if weight == 2:
        bucket = '126:140'

    if weight == 3:
        bucket = '141:170'

    if weight == 4:
        bucket = '171:200'

    if weight == 5:
        bucket = '201:250+'

    return bucket
```

[10]:
```
df['Weight_grp'] = df['Weight'].apply(weight_group)
df['Weight_grp'].head()
```

[10]:
```
1     100:125
2     171:200
3     171:200
4    201:250+
5     171:200
Name: Weight_grp, dtype: object
```

[189]:
```
df.head()
```

[189]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics	Physical_Activity	Sports_highest_level	LESI	Reason_for_visit	LFA	RFA	Age_grp	Height_grp	Weight_grp
1	1	3	1	F	9.0	F	N	4	3	Y	4	34	29	18-25	5-6:5-10	100:125
2	1	4	4	M	10.0	M	N	5	3	Y	4	28	28	18-25	5-11:6-1	171:200
3	4	3	4	M	10.0	M	N	3	4	Y	4	35	36	46-55	5-6:5-10	171:200
4	1	5	5	M	13.0	M	N	4	4	Y	1	36	36	18-25	6-2:6-5	201:250+
5	4	3	4	M	10.0	M	Y	5	3	Y	3	31	28	46-55	5-6:5-10	171:200

```
5    171:200
```

[14]:
```
df.Physical_Activity.value_counts()
```

[14]:
```
5    28
4    18
3    13
2     2
1     2
Name: Physical_Activity, dtype: int64
```

[11]:
```
# Physical Activity:
# 1 = Sedentary; 2 = Mostly Sedentary; 3 = Moderately active; 4 = Active; 5 = Highly active
# column index = 8

def physical_group(physical):

    """Creates an age bucket for each participant using the age variable.
        Meant to be used on a DataFrame with .apply()."""

    # Convert to an int, in case the data is read in as an "object" (aka string)
    physical = int(physical)

    if physical == 1:
        bucket = 'Sedentary'

    if physical == 2:
        bucket = 'Mostly Sedentary'

    if physical == 3:
        bucket = 'Moderately Active'

    if physical == 4:
        bucket = 'Active'

    if physical == 5:
        bucket = 'Highly Active'

    return bucket
```

[12]:
```
df['Physical_grp'] = df['Physical_Activity'].apply(physical_group)
df['Physical_grp'].head()
```

[12]:
```
1               Active
2         Highly Active
3    Moderately Active
```

```
4               Active
5        Highly Active
Name: Physical_grp, dtype: object
```

[116]:
```
df.columns
```

[116]:
```
Index(['Age', 'Height', 'Weight', 'Sex', 'Shoe_size', 'Gender', 'Orthotics',
       'Physical_Activity', 'Sports_highest_level', 'LESI', 'Reason_for_visit',
       'LFA', 'RFA', 'Age_grp', 'Height_grp', 'Weight_grp'],
      dtype='object')
```

[165]:
```
df.head()
```

[165]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics	Physical_Activity	\
1	1	3	1	F	9.0	F	N	4	
2	1	4	4	M	10.0	M	N	5	
3	4	3	4	M	10.0	M	N	3	
4	1	5	5	M	13.0	M	N	4	
5	4	3	4	M	10.0	M	Y	5	

	Sports_highest_level	LESI	Reason_for_visit	LFA	RFA	Age_grp	Height_grp	\
1	3	Y	4	34	29	18-25	5-6:5-10	
2	3	Y	4	28	28	18-25	5-11:6-1	
3	4	Y	4	35	36	46-55	5-6:5-10	
4	4	Y	1	36	36	18-25	6-2:6-5	
5	3	Y	3	31	28	46-55	5-6:5-10	

	Weight_grp	Physical_grp
1	100:125	Active
2	171:200	Highly Active
3	171:200	Moderately Active
4	201:250+	Active
5	171:200	Highly Active

[15]:
```
df.Sports_highest_level.value_counts()
```

[15]:
```
0    21
2    19
3    12
4     5
1     5
5     1
Name: Sports_highest_level, dtype: int64
```

[13]:
```
# Sports:
# 0 = No; 1 = Club; 2 = High School; 3 = College; 4 = Semi-Professional; 5 =␣
 ↪Professional
```

```
# column index = 9

def sports_group(sports):

    """Creates an age bucket for each participant using the age variable.
        Meant to be used on a DataFrame with .apply()."""

    # Convert to an int, in case the data is read in as an "object" (aka string)
    sports = int(sports)

    if sports == 0:
        bucket = 'None'

    if sports == 1:
        bucket = 'Club'

    if sports == 2:
        bucket = 'High School'

    if sports == 3:
        bucket = 'College'

    if sports == 4:
        bucket = 'Semi-Professional'

    if sports == 5:
        bucket = 'Professional'

    return bucket
```

[14]:
```
df['Sports_grp'] = df['Sports_highest_level'].apply(sports_group)
df['Sports_grp'].head()
```

[14]:
```
1              College
2              College
3    Semi-Professional
4    Semi-Professional
5              College
Name: Sports_grp, dtype: object
```

[168]:
```
df.head()
```

[168]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics	Physical_Activity	\
1	1	3	1	F	9.0	F	N	4	
2	1	4	4	M	10.0	M	N	5	
3	4	3	4	M	10.0	M	N	3	
4	1	5	5	M	13.0	M	N	4	

```
5    4       3       4   M       10.0      M         Y                   5

   Sports_highest_level LESI  Reason_for_visit  LFA  RFA Age_grp Height_grp  \
1                     3    Y                 4   34   29   18-25   5-6:5-10
2                     3    Y                 4   28   28   18-25   5-11:6-1
3                     4    Y                 4   35   36   46-55   5-6:5-10
4                     4    Y                 1   36   36   18-25    6-2:6-5
5                     3    Y                 3   31   28   46-55   5-6:5-10

  Weight_grp       Physical_grp         Sports_grp
1    100:125             Active            College
2    171:200      Highly Active            College
3    171:200  Moderately Active  Semi-Professional
4   201:250+             Active  Semi-Professional
5    171:200      Highly Active            College
```

[16]:
```
df.Reason_for_visit.value_counts()
```

[16]:
```
4    26
2    14
3    13
1     4
5     3
0     3
Name: Reason_for_visit, dtype: int64
```

[15]:
```
# Reason for Visit:
# 1 = Instability of lower extremity/gait-related
# 2 = Recent/Acute onset lower extremity injury
# 3 = Recurring/Chronic Ongoing lower extremity injury
# 4 = Foot or Ankle Pain
# 5 = Deformity of lower extremity (arthritis, bunions, unknown mass etc.)
# column index = 11

def visit_group(visit):
    global bucket

    """Creates an age bucket for each participant using the age variable.
        Meant to be used on a DataFrame with .apply()."""

    # Convert to an int, in case the data is read in as an "object" (aka string)
    visit = int(visit)

    if visit == 1:
        bucket = 'Instability_LE_GR'

    if visit == 2:
```

```
        bucket = 'Acute_onset_LEI'

    if visit == 3:
        bucket = 'Chronic_LEI'

    if visit == 4:
        bucket = 'Foot Ankle Pain'

    if visit == 5:
        bucket = 'Deformity_LE'

    return bucket
```

[16]:
```
df['Visit_grp'] = df['Reason_for_visit'].apply(visit_group)
df['Visit_grp'].head()
```

[16]:
```
1      Foot Ankle Pain
2      Foot Ankle Pain
3      Foot Ankle Pain
4    Instability_LE_GR
5          Chronic_LEI
Name: Visit_grp, dtype: object
```

[196]:
```
df.head()
```

[196]:

	Age	Height	Weight	Sex	Shoe_size	Gender	Orthotics	Physical_Activity	\
1	1	3	1	F	9.0	F	N	4	
2	1	4	4	M	10.0	M	N	5	
3	4	3	4	M	10.0	M	N	3	
4	1	5	5	M	13.0	M	N	4	
5	4	3	4	M	10.0	M	Y	5	

	Sports_highest_level	LESI	Reason_for_visit	LFA	RFA	Age_grp	Height_grp	\
1	3	Y	4	34	29	18-25	5-6:5-10	
2	3	Y	4	28	28	18-25	5-11:6-1	
3	4	Y	4	35	36	46-55	5-6:5-10	
4	4	Y	1	36	36	18-25	6-2:6-5	
5	3	Y	3	31	28	46-55	5-6:5-10	

	Weight_grp	Physical_grp	Sports_grp	Visit_grp
1	100:125	Active	College	Foot Ankle Pain
2	171:200	Highly Active	College	Foot Ankle Pain
3	171:200	Moderately Active	Semi-Professional	Foot Ankle Pain
4	201:250+	Active	Semi-Professional	Instability_LE_GR
5	171:200	Highly Active	College	Chronic_LEI

[17]:
```
# drop Notes column as it will not be used in this analysis
df = df.drop(columns=['Age','Height','Weight','Physical_Activity',
                      'Sports_highest_level','Reason_for_visit'], axis=1)
```

[18]:
```
df.head()
```

[18]:

	Sex	Shoe_size	Gender	Orthotics	LESI	LFA	RFA	Age_grp	Height_grp
1	F	9.0	F	N	Y	34	29	18-25	5-6:5-10
2	M	10.0	M	N	Y	28	28	18-25	5-11:6-1
3	M	10.0	M	N	Y	35	36	46-55	5-6:5-10
4	M	13.0	M	N	Y	36	36	18-25	6-2:6-5
5	M	10.0	M	Y	Y	31	28	46-55	5-6:5-10

	Weight_grp	Physical_grp	Sports_grp	Visit_grp
1	100:125	Active	College	Foot Ankle Pain
2	171:200	Highly Active	College	Foot Ankle Pain
3	171:200	Moderately Active	Semi-Professional	Foot Ankle Pain
4	201:250+	Active	Semi-Professional	Instability_LE_GR
5	171:200	Highly Active	College	Chronic_LEI

[ ]:
```
# need to encode categorical variables
# Age, Height, Weight, Sex, Gender, Orthotics, Physical Activity, Sports, LESI,␣
 ↪and
# Reason for Visit
# importing OneHotEncoder
#from sklearn.compose import ColumnTransformer
#from sklearn.preprocessing import OneHotEncoder
```

[ ]:
```
# One-hot encoding: takes a list and returns an upated DF with cols included,␣
 ↪specifying
# a prefix for readability
# Dummy encoding: the base value is encoded by the absence (creates n-1␣
 ↪categories, omitting
# first category); its value is represented by the intercept
```

[19]:
```
# create dummy variables for binary return columns

df = pd.get_dummies(df, columns=['Sex','Gender','Orthotics','LESI'],␣
 ↪drop_first=True)
```

[20]:
```
df.head()
```

[20]:

	Shoe_size	LFA	RFA	Age_grp	Height_grp	Weight_grp	Physical_grp
1	9.0	34	29	18-25	5-6:5-10	100:125	Active
2	10.0	28	28	18-25	5-11:6-1	171:200	Highly Active
3	10.0	35	36	46-55	5-6:5-10	171:200	Moderately Active
4	13.0	36	36	18-25	6-2:6-5	201:250+	Active

```
5       10.0   31   28   46-55   5-6:5-10    171:200      Highly Active

          Sports_grp          Visit_grp  Sex_M  Gender_M  Orthotics_Y  LESI_Y
1            College    Foot Ankle Pain      0         0            0       1
2            College    Foot Ankle Pain      1         1            0       1
3  Semi-Professional    Foot Ankle Pain      1         1            0       1
4  Semi-Professional  Instability_LE_GR      1         1            0       1
5            College        Chronic_LEI      1         1            1       1
```

[ ]:
```
# need to bin 'Age_grp', 'Height_grp', 'Weight', Physical_grp',
# 'Sports_grp', 'Visit_grp'
# will use One Hot Encoding in order to run machine learning models
```

[203]:
```
df.Age_grp.value_counts()
```

[203]:
```
56-75    22
46-55    16
18-25    14
36-45    13
26-35     9
Name: Age_grp, dtype: int64
```

[204]:
```
df.Height_grp.value_counts()
```

[204]:
```
5-2:5-5     24
5-6:5-10    19
5-11:6-1    16
6-2:6-5     10
4-9:5-1      5
Name: Height_grp, dtype: int64
```

[205]:
```
df.Weight_grp.value_counts()
```

[205]:
```
171:200     19
201:250+    16
141:170     15
126:140     13
100:125     11
Name: Weight_grp, dtype: int64
```

[206]:
```
df.Physical_grp.value_counts()
```

[206]:
```
Highly Active        33
Active               23
Moderately Active    14
Mostly Sedentary      2
Sedentary             2
Name: Physical_grp, dtype: int64
```

[207]:
```
df.Sports_grp.value_counts()
```

[207]:
```
None                27
High School         21
College             13
Semi-Professional    6
Club                 6
Professional         1
Name: Sports_grp, dtype: int64
```

[208]:
```
df.Visit_grp.value_counts()
```

[208]:
```
Foot Ankle Pain      35
Chronic_LEI          16
Acute_onset_LEI      16
Instability_LE_GR     4
Deformity_LE          3
Name: Visit_grp, dtype: int64
```

[21]:
```
categorical_cols = ['Age_grp','Height_grp','Weight_grp','Physical_grp',
                    'Sports_grp','Visit_grp']
```

[22]:
```
for column in categorical_cols:
    df_temp = pd.get_dummies(df[column], prefix=column)

    df = pd.merge(left=df, right=df_temp,
                  left_index=True, right_index=True,)

    df = df.drop(columns=column)
```

[251]:
```
#feature_labels = np.array(feature_labels).ravel()
```

[23]:
```
df.head()
```

[23]:

	Shoe_size	LFA	RFA	Sex_M	Gender_M	Orthotics_Y	LESI_Y	Age_grp_18-25	\
1	9.0	34	29	0	0	0	1	1	
2	10.0	28	28	1	1	0	1	1	
3	10.0	35	36	1	1	0	1	0	
4	13.0	36	36	1	1	0	1	1	
5	10.0	31	28	1	1	1	1	0	

	Age_grp_26-35	Age_grp_36-45	…	Sports_grp_College	\
1	0	0	…	1	
2	0	0	…	1	
3	0	0	…	0	
4	0	0	…	0	
5	0	0	…	1	

```
   Sports_grp_High School  Sports_grp_None  Sports_grp_Professional  \
1                       0                0                        0
2                       0                0                        0
3                       0                0                        0
4                       0                0                        0
5                       0                0                        0

   Sports_grp_Semi-Professional  Visit_grp_Acute_onset_LEI  \
1                             0                          0
2                             0                          0
3                             1                          0
4                             1                          0
5                             0                          0

   Visit_grp_Chronic_LEI  Visit_grp_Deformity_LE  Visit_grp_Foot Ankle Pain  \
1                      0                       0                          1
2                      0                       0                          1
3                      0                       0                          1
4                      0                       0                          0
5                      1                       0                          0

   Visit_grp_Instability_LE_GR
1                            0
2                            0
3                            0
4                            1
5                            0

[5 rows x 38 columns]
```

[50]:
```
df.shape
```

[50]:
```
(74, 38)
```

[51]:
```
df.columns
```

[51]:
```
Index(['Shoe_size', 'LFA', 'RFA', 'Sex_M', 'Gender_M', 'Orthotics_Y', 'LESI_Y',
       'Age_grp_18-25', 'Age_grp_26-35', 'Age_grp_36-45', 'Age_grp_46-55',
       'Age_grp_56-75', 'Height_grp_4-9:5-1', 'Height_grp_5-11:6-1',
       'Height_grp_5-2:5-5', 'Height_grp_5-6:5-10', 'Height_grp_6-2:6-5',
       'Weight_grp_100:125', 'Weight_grp_126:140', 'Weight_grp_141:170',
       'Weight_grp_171:200', 'Weight_grp_201:250+', 'Physical_grp_Active',
       'Physical_grp_Highly Active', 'Physical_grp_Moderately Active',
       'Physical_grp_Mostly Sedentary', 'Physical_grp_Sedentary',
       'Sports_grp_Club', 'Sports_grp_College', 'Sports_grp_High School',
       'Sports_grp_None', 'Sports_grp_Professional',
       'Sports_grp_Semi-Professional', 'Visit_grp_Acute_onset_LEI',
```

```
       'Visit_grp_Chronic_LEI', 'Visit_grp_Deformity_LE',
       'Visit_grp_Foot Ankle Pain', 'Visit_grp_Instability_LE_GR'],
      dtype='object')
```

[27]:
```
#df = df.drop(columns=['Sex','Gender','Orthotics','LESI'], axis=1)
```

[28]:
```
#df.head()
```

[28]:

	Shoe_size	LFA	RFA	Age_grp_18-25	Age_grp_26-35	Age_grp_36-45	\
0	8.5	0	0	0	0	1	
1	9.0	34	29	1	0	0	
2	10.0	28	28	1	0	0	
3	10.0	35	36	0	0	0	
4	13.0	36	36	1	0	0	

	Age_grp_46-55	Age_grp_56-75	Height_grp_4-9:5-1	Height_grp_5-11:6-1	…	\
0	0	0	0	0	…	
1	0	0	0	0	…	
2	0	0	0	1	…	
3	1	0	0	0	…	
4	0	0	0	0	…	

	Sports_grp_College	Sports_grp_High School	Sports_grp_None	\
0	1	0	0	
1	1	0	0	
2	1	0	0	
3	0	0	0	
4	0	0	0	

	Sports_grp_Professional	Sports_grp_Semi-Professional	\
0	0	0	
1	0	0	
2	0	0	
3	0	1	
4	0	1	

	Visit_grp_Acute_onset_LEI	Visit_grp_Chronic_LEI	Visit_grp_Deformity_LE	\
0	1	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

	Visit_grp_Foot Ankle Pain	Visit_grp_Instability_LE_GR
0	0	0
1	1	0
2	1	0

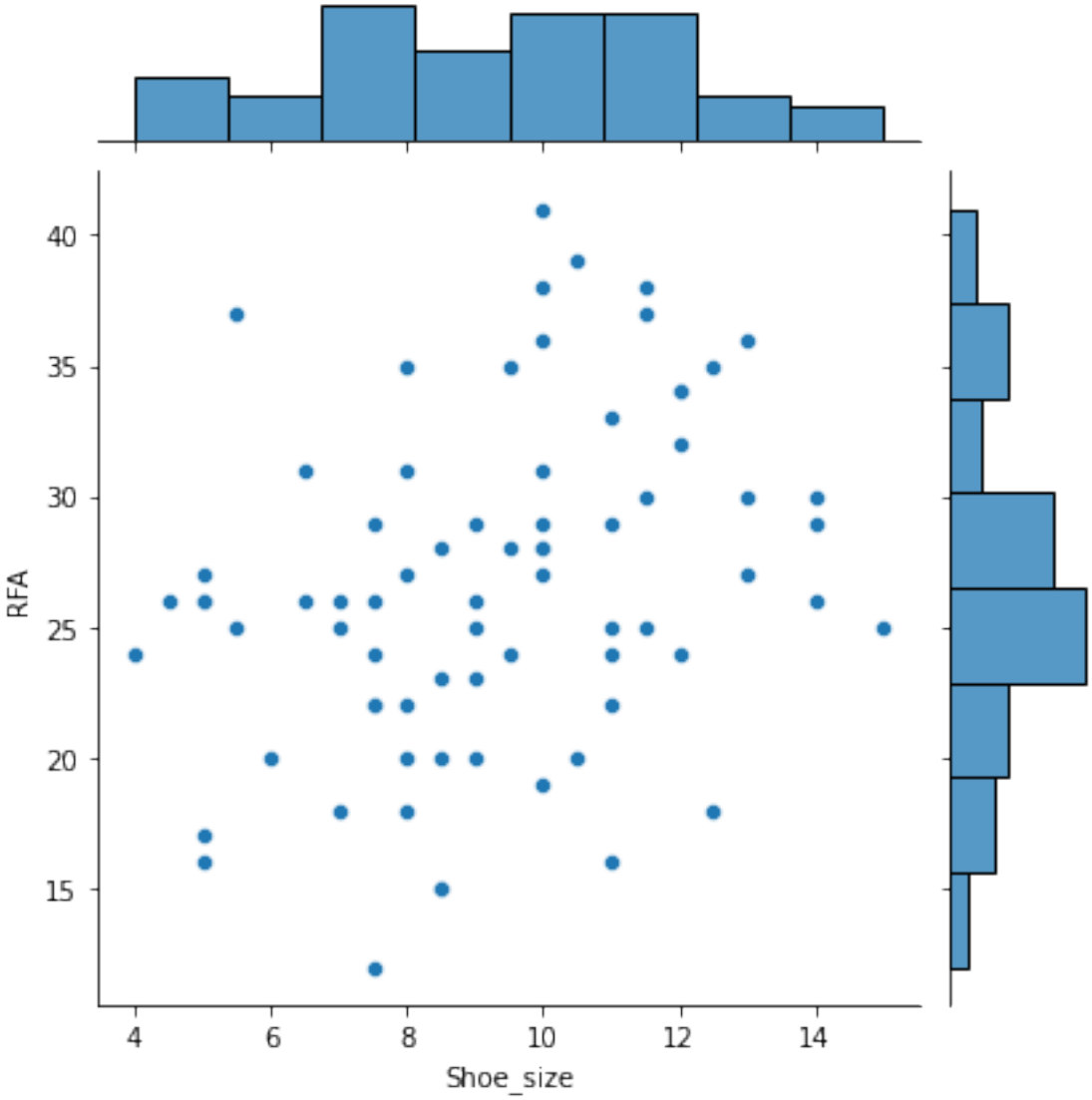
```
3                          1                          0
4                          0                          1

[5 rows x 34 columns]
```

[24]:
```
#df.shape
```

[24]:
```
(74, 34)
```

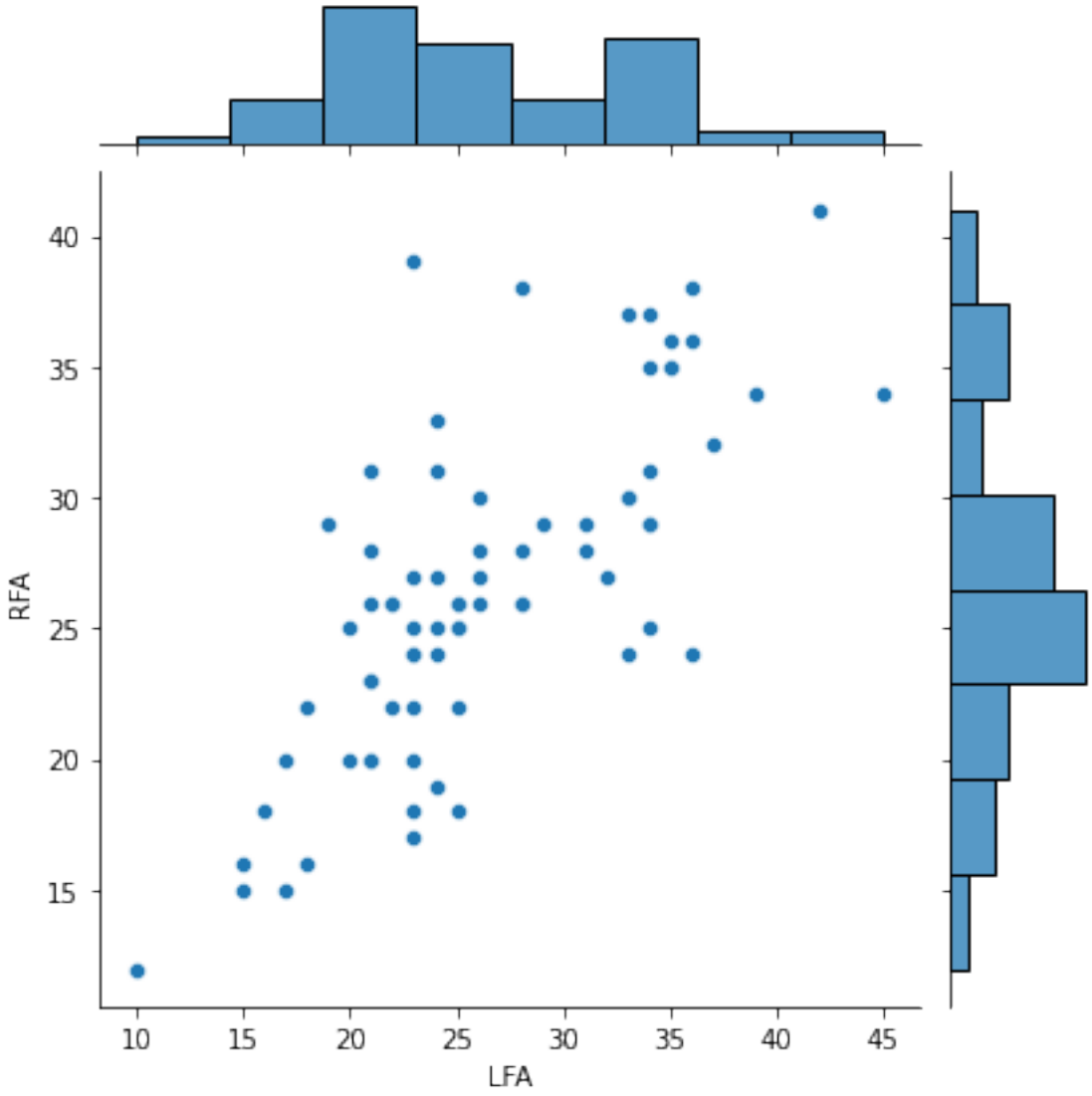
[26]:
```
# additional view

sns.jointplot(x = 'Shoe_size', y = 'RFA', data = df)
```

[26]:
```
<seaborn.axisgrid.JointGrid at 0x7fc7d88e2a30>
```



[27]:
```
# additional view

sns.jointplot(x = 'LFA', y = 'RFA', data = df)
```

[27]: <seaborn.axisgrid.JointGrid at 0x7fc7b807b100>



2 linear regression

3 we are trying to find a function that maps some features or variables to

4 other sufficiently well

5 dependent features: dependent variables, outputs or responses

6 independent features: independent variables, inputs or predictors

[ ]:
```
# we can update notebook:
# Problem Statement
# Step 0: load libraries
# Step 1: import data
# Step 2: visualize, explore data
# Step 3: create testing and training datasets/data cleaning
# Step 4: evaluate models
# Step 5: create summary - discuss findings, recommend any next actions
```

[25]:
```
# multiple linear regression: we will focus on predicting RFA
# MLR examines relationship between more than two variables
# each independent variable has its own corresponding coefficient
# y = b0 + b1 * x1 + b2 * x2 +..+bnxn, where y is dependent var and x are the␣
 ↪ind vars

# import packages and classes (already imported numpy)

#from sklearn.linear_model import LinearRegression
```

7 Model 1

[28]:
```
df.columns
```

[28]:
```
Index(['Shoe_size', 'LFA', 'RFA', 'Age_grp_18-25', 'Age_grp_26-35',
       'Age_grp_36-45', 'Age_grp_46-55', 'Age_grp_56-75', 'Height_grp_4-9:5-1',
       'Height_grp_5-11:6-1', 'Height_grp_5-2:5-5', 'Height_grp_5-6:5-10',
       'Height_grp_6-2:6-5', 'Weight_grp_100:125', 'Weight_grp_126:140',
       'Weight_grp_141:170', 'Weight_grp_171:200', 'Weight_grp_201:250+',
       'Physical_grp_Active', 'Physical_grp_Highly Active',
       'Physical_grp_Moderately Active', 'Physical_grp_Mostly Sedentary',
       'Physical_grp_Sedentary', 'Sports_grp_Club', 'Sports_grp_College',
       'Sports_grp_High School', 'Sports_grp_None', 'Sports_grp_Professional',
       'Sports_grp_Semi-Professional', 'Visit_grp_Acute_onset_LEI',
       'Visit_grp_Chronic_LEI', 'Visit_grp_Deformity_LE',
       'Visit_grp_Foot Ankle Pain', 'Visit_grp_Instability_LE_GR'],
      dtype='object')
```

[52]:
```
# create training and testing data
```

```
y = df['RFA']
X =␣
↪df[['LFA','Shoe_size','Age_grp_18-25','Age_grp_26-35','Age_grp_36-45','Age_grp_46-55',
      'Age_grp_56-75','Height_grp_4-9:5-1','Height_grp_5-2:
↪5-5','Height_grp_5-6:5-10',
      'Height_grp_5-11:6-1','Height_grp_6-2:6-5','Weight_grp_100:
↪125','Weight_grp_126:140',
      'Weight_grp_141:170','Weight_grp_171:200','Weight_grp_201:
↪250+','Physical_grp_Active',
      'Physical_grp_Highly Active','Physical_grp_Moderately␣
↪Active','Physical_grp_Mostly Sedentary',
     ␣
↪'Physical_grp_Sedentary','Sports_grp_Club','Sports_grp_College','Sports_grp_High␣
↪School',
     ␣
↪'Sports_grp_None','Sports_grp_Professional','Sports_grp_Semi-Professional','Visit_grp_Acute
      'Visit_grp_Chronic_LEI','Visit_grp_Deformity_LE','Visit_grp_Foot Ankle␣
↪Pain',
      'Visit_grp_Instability_LE_GR']]
```

[53]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2)
```

[32]:
```
X_train.shape
```

[32]:
```
(59, 33)
```

[33]:
```
X_test.shape
```

[33]:
```
(15, 33)
```

[34]:
```
y_train.shape
```

[34]:
```
(59,)
```

[35]:
```
y_test.shape
```

[35]:
```
(15,)
```

[54]:
```
# train the model (1st attempt)
from sklearn.linear_model import LinearRegression
```

[55]:
```
# instantiate the object out of the class
regressor = LinearRegression(fit_intercept = True)
```

[56]:
```
regressor.fit(X_train, y_train)
```

[56]:
```
LinearRegression()
```

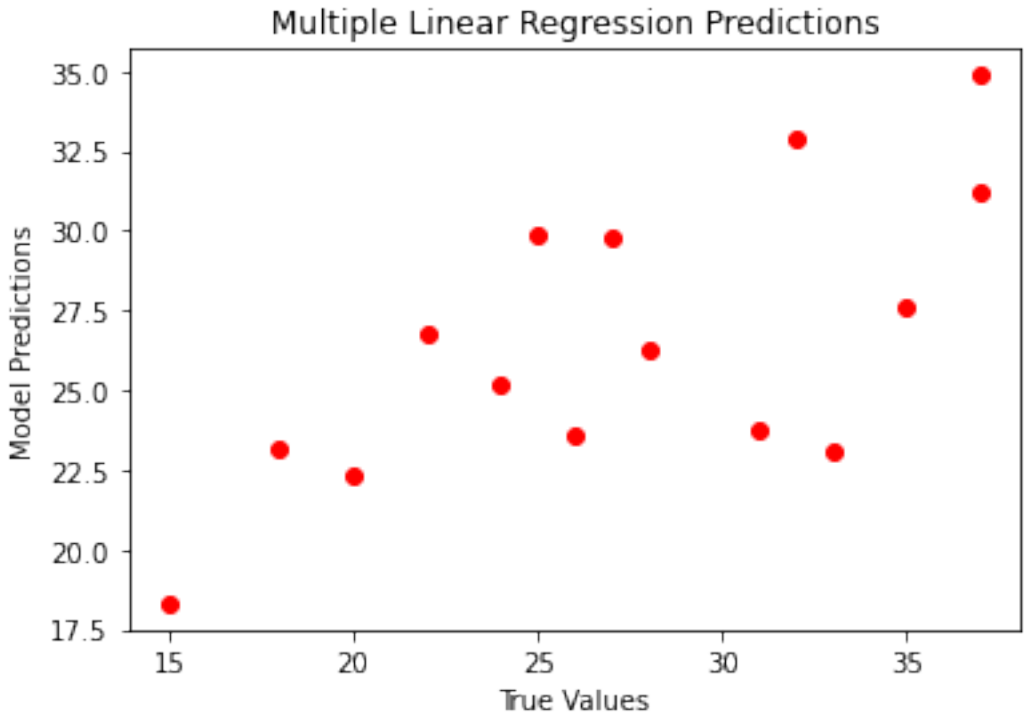
[39]:
```
print('Linear Model Coefficients (m)', regressor.coef_)
print('Linear Model Coefficients (b)', regressor.intercept_)
```

```
Linear Model Coefficients (m) [ 0.79128859  0.07092728 -0.11315172 -2.06949806
1.24242415 -0.88552315
  1.82574878 -0.09865378  1.97368739  2.41294664  0.55444785 -4.84242809
  0.48773157 -0.58904892 -0.45081183  2.35947241 -1.80734323  2.12530466
  0.36787367 -0.91924819 -5.89091195  4.3169818  -1.77853119 -2.86915388
  1.37083757 -1.48262662  3.45694122  1.3025329  -2.89284176 -1.65522913
  0.79058721  0.3027613   3.45472238]
Linear Model Coefficients (b) 4.174265809088386
```

[57]:
```
# evaluate the model: model predictions
y_predict = regressor.predict(X_test)
```

[58]:
```
# plot the model predictions against the true values
plt.scatter(y_test, y_predict, color = 'red')
plt.ylabel('Model Predictions')
plt.xlabel('True Values')
plt.title('Multiple Linear Regression Predictions')
```

[58]:
```
Text(0.5, 1.0, 'Multiple Linear Regression Predictions')
```



[60]:
```
regressor.score(X,y)
```

[60]:
```
0.6302596666555779
```

[61]:
```
r2 = regressor.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[61]:
```
0.3252238916464296
```

8 Model 1 - summary:

r2 = 0.630 adjusted r2 = 0.325 RMSE = 4.855 MSE = 23.572

notes: heavy penalty for having too many variables that are likely not good predictors and is overfitting the line

[62]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from math import sqrt

# root mean square error (RMSE): the std dev of the residuals (prediction␣
 ↪errors); these
# are a measure of how far from the regression line data points are; measures␣
 ↪how spread
# out the residuals are. How concentrated the data is around the line of best␣
 ↪fit.
# has a direct relationship with the correlation coefficient

# mean squared error (MSE): how close a regression line is to a set of points,␣
 ↪the avg of
# a set of errors...the lower the MSE, the better the forecast

# mean absolute error (MAE): the avg of all absolute errors

# r-squared (r2): goodness of fit measure, indicates the percentage of the␣
 ↪variance
# in the dependent variable that the independent variables explain,␣
 ↪collectively.
# measures the strength of the relationship between your model and the␣
 ↪dependent var on a
# 0-100% scale.

# here, the r2 score is negative, indicating the chosen model does not follow␣
 ↪the
```

```
# trend of the data. Likely overfitting.  We may try removing the LFA.  Again,␣
 ↪negative
# r2 typically indicates this is worse than using the simple mean.
# r2 = Var(mean) - Var(line) / Var(mean)

# adjusted r2:

# mean absolute percentage error (MAPE): measure of how accurate a forecast
# system is. Measures accuracy as a percentage and can be calculated as the avg
# absolute percentage error for each time period minus actual values divided by
# actual values. Most common measure used to forecast error, and works best
# if there are no extremes to the data (and no zeros)

RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_predict))))
MSE = mean_squared_error(y_test, y_predict)
#MAE = mean_absolute_error(y_test, y_predict)
#r2 = r2_score(y_test, y_predict)
#adj_r2 = 1-(1-r2) * (n-1)/(n-k-1)
#MAPE = np.mean(np.abs((y_test-y_predict/y_test)))*100
```

[64]:
```
print('RMSE = ', round(RMSE,3))
print('MSE = ', round(MSE,3))
#print('MAE = ', round(MAE,2))
#print('r2 = ', round(r2,2))
#print('Adjusted r2 = ', adj_r2)
#print('MAPE = ', round(MAPE,2))
```

```
RMSE =  4.855
MSE =  23.572
```

9 Model 2

[65]:
```
# tweak model, run again (2nd attempt)

# create training and testing data

y = df['RFA']
X = df[['LFA','Shoe_size']]
```

[66]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2)
```

[67]:
```
# train the model (1st attempt)
from sklearn.linear_model import LinearRegression

# instantiate the object out of the class
regressor = LinearRegression(fit_intercept = True)
```

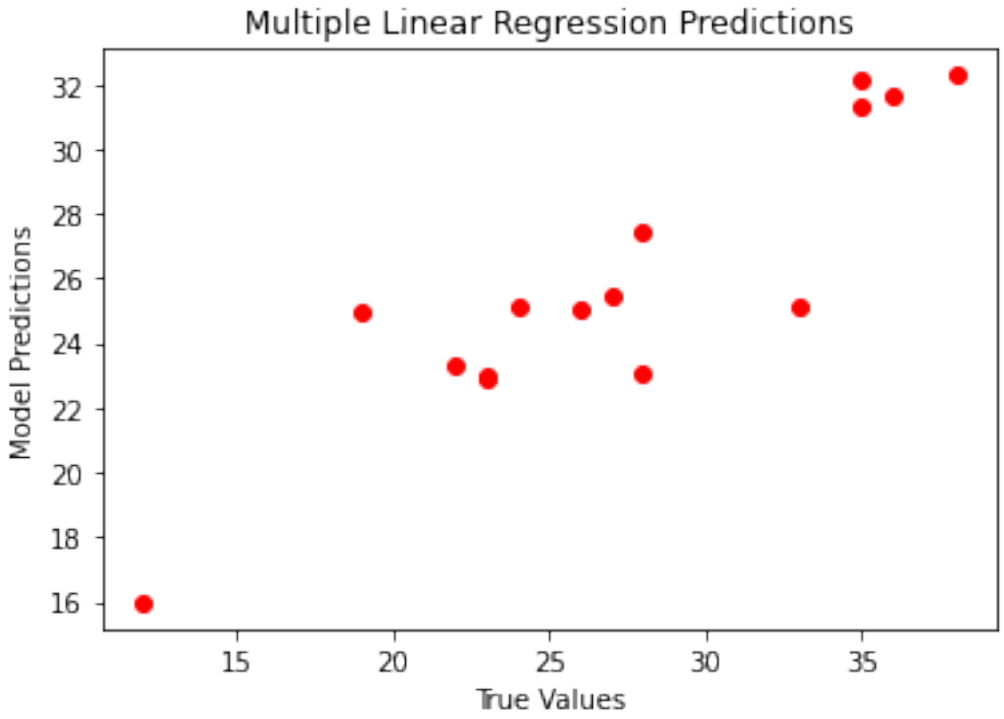
[68]:
```
regressor.fit(X_train, y_train)
```

[68]:
```
LinearRegression()
```

[69]:
```
# evaluate the model: model predictions
y_predict = regressor.predict(X_test)
```

[70]:
```
# plot the model predictions against the true values
plt.scatter(y_test, y_predict, color = 'red')
plt.ylabel('Model Predictions')
plt.xlabel('True Values')
plt.title('Multiple Linear Regression Predictions')
```

[70]:
```
Text(0.5, 1.0, 'Multiple Linear Regression Predictions')
```



[71]:
```
regressor.score(X,y)
```

[71]:
```
0.5453601746061137
```

[72]:
```
r2 = regressor.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
```

```
adjusted_r2
```

[72]: 0.5325534189612156

[73]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from math import sqrt

RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_predict))))
MSE = mean_squared_error(y_test, y_predict)
#MAE = mean_absolute_error(y_test, y_predict)
#r2 = r2_score(y_test, y_predict)
#adj_r2 = 1-(1-r2) * (n-1)/(n-k-1)
#MAPE = np.mean(np.abs((y_test-y_predict/y_test)))*100
```

[74]:
```
# much better results
print('RMSE = ', round(RMSE,2))
print('MSE = ', round(MSE,2))
#print('MAE = ', round(MAE,2))
#print('r2 = ', round(r2,2))
#print('Adjusted r2 = ', round(adj_r2,2))
#print('MAPE = ', round(MAPE,2))
```

```
RMSE =  3.81
MSE =  14.5
```

10 Model 2 Summary:

r2 = 0.545 adjusted r2 = 0.533 RMSE = 3.81 MSE = 14.5

Notes: r2 is lower (less predictors), but adjusted r2 is much higher (less penalty) likely meaning, these predictors are more suitable for the model, based on p-values, indicating they are better predictors of the response

11 Model 3

[75]:
```
# 3rd attempt

# create training and testing data

y = df['RFA']
X =␣
 ↪df[['Shoe_size','Age_grp_18-25','Age_grp_26-35','Age_grp_36-45','Age_grp_46-55',
       'Age_grp_56-75']]
```

[76]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

[77]:
```
# train the model (1st attempt)
from sklearn.linear_model import LinearRegression

# instantiate the object out of the class
regressor = LinearRegression(fit_intercept = True)
```

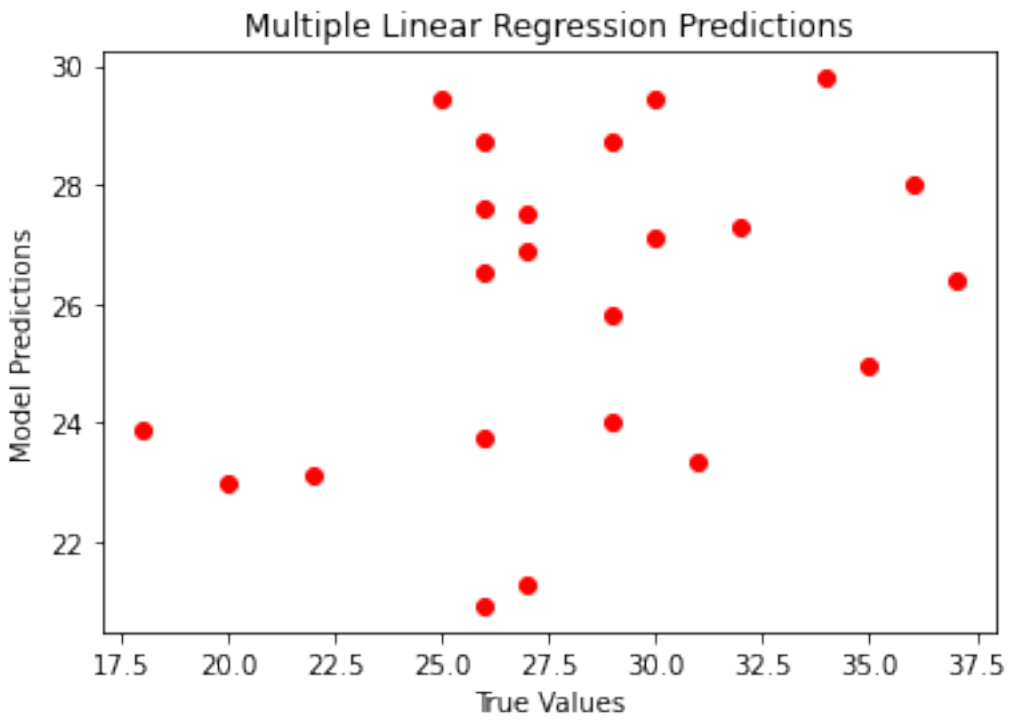
[78]:
```
regressor.fit(X_train, y_train)
```

[78]:
```
LinearRegression()
```

[79]:
```
# evaluate the model: model predictions
y_predict = regressor.predict(X_test)
```

[80]:
```
# plot the model predictions against the true values
plt.scatter(y_test, y_predict, color = 'red')
plt.ylabel('Model Predictions')
plt.xlabel('True Values')
plt.title('Multiple Linear Regression Predictions')
```

[80]:
```
Text(0.5, 1.0, 'Multiple Linear Regression Predictions')
```



[81]:
```
regressor.score(X,y)
```

[81]:
```
0.0990255622885926
```

[82]:
```
r2 = regressor.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[82]:
```
0.018341284284585924
```

[83]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from math import sqrt

RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_predict))))
MSE = mean_squared_error(y_test, y_predict)
#MAE = mean_absolute_error(y_test, y_predict)
#r2 = r2_score(y_test, y_predict)
#adj_r2 = 1-(1-r2) * (n-1)/(n-k-1)
#MAPE = np.mean(np.abs((y_test-y_predict/y_test)))*100
```

[84]:
```
# much better results
print('RMSE = ', round(RMSE,2))
print('MSE = ', round(MSE,2))
#print('MAE = ', round(MAE,2))
#print('r2 = ', round(r2,2))
#print('Adjusted r2 = ', round(adj_r2,2))
#print('MAPE = ', round(MAPE,2))
```

```
RMSE =  4.92
MSE =  24.23
```

12 Model 3 summary:

r2 = 0.099 adjusted r2 = 0.018 RMSE = 4.92 MSE = 24.23

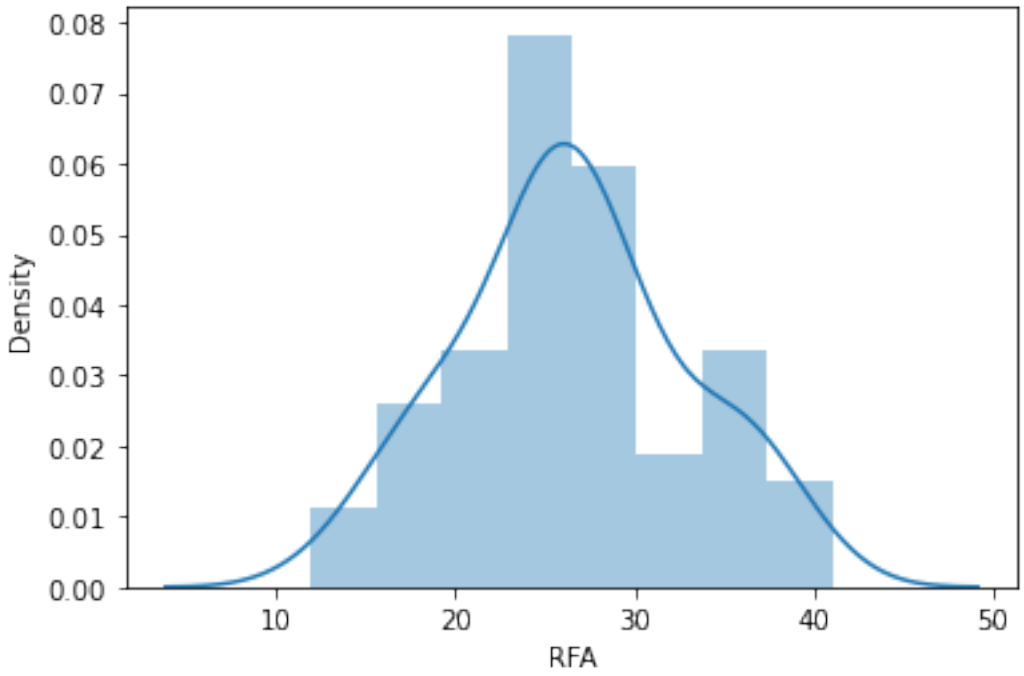
Notes: taking LFA out of the predictors and inserting Age group negatively impacted the model results. Much less accuracy.

[ ]:
```
# model 2 is best so far, meaning our paramaters produced the best forecast
```

[112]:
```
# since we are trying to predict RFA, let's take a closer look at that variable
sns.distplot(df['RFA'])
```

```
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

[112]: <AxesSubplot:xlabel='RFA', ylabel='Density'>



13 Model 4

[85]:
```
# 4th attempt

# create training and testing data

y = df['RFA']
X = df[['LFA','Shoe_size','Physical_grp_Active',
       'Physical_grp_Highly Active','Physical_grp_Moderately␣
↪Active','Physical_grp_Mostly Sedentary',
      ␣
↪'Physical_grp_Sedentary','Sports_grp_Club','Sports_grp_College','Sports_grp_High␣
↪School',
      ␣
↪'Sports_grp_None','Sports_grp_Professional','Sports_grp_Semi-Professional','Visit_grp_Acute
       'Visit_grp_Chronic_LEI','Visit_grp_Deformity_LE','Visit_grp_Foot Ankle␣
↪Pain',
       'Visit_grp_Instability_LE_GR']]
```

[86]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

[87]:
```
# train the model
from sklearn.linear_model import LinearRegression
```

[88]:
```
lm = LinearRegression()
```

[89]:
```
lm.fit(X_train,y_train)
```

[89]:
```
LinearRegression()
```

[118]:
```
print(lm.intercept_)
```

```
7.929096412707086
```

[119]:
```
lm.coef_
```

[119]:
```
array([ 6.76774036e-01, -1.51129809e-01,  2.93412864e+00,  9.32710702e-01,
       -2.40300393e+00, -1.71208384e+00,  2.48248430e-01, -9.02315249e-01,
       -1.49244004e+00,  1.00126871e+00,  8.04762542e-01, -2.22044605e-16,
        5.88724040e-01, -1.37997252e+00,  2.66195715e+00, -4.16875488e+00,
        4.55381535e-01,  2.43138871e+00])
```

[120]:
```
# create dataframe off this
cdf = pd.DataFrame(lm.coef_,X.columns,columns=['Coeff'])
```

[122]:
```
cdf
```

[122]:

	Coeff
LFA	6.767740e-01
Shoe_size	-1.511298e-01
Physical_grp_Active	2.934129e+00
Physical_grp_Highly Active	9.327107e-01
Physical_grp_Moderately Active	-2.403004e+00
Physical_grp_Mostly Sedentary	-1.712084e+00
Physical_grp_Sedentary	2.482484e-01
Sports_grp_Club	-9.023152e-01
Sports_grp_College	-1.492440e+00
Sports_grp_High School	1.001269e+00
Sports_grp_None	8.047625e-01
Sports_grp_Professional	-2.220446e-16
Sports_grp_Semi-Professional	5.887240e-01
Visit_grp_Acute_onset_LEI	-1.379973e+00
Visit_grp_Chronic_LEI	2.661957e+00
Visit_grp_Deformity_LE	-4.168755e+00
Visit_grp_Foot Ankle Pain	4.553815e-01
Visit_grp_Instability_LE_GR	2.431389e+00

[90]:
```
# predictions
predictions = lm.predict(X_test)
```

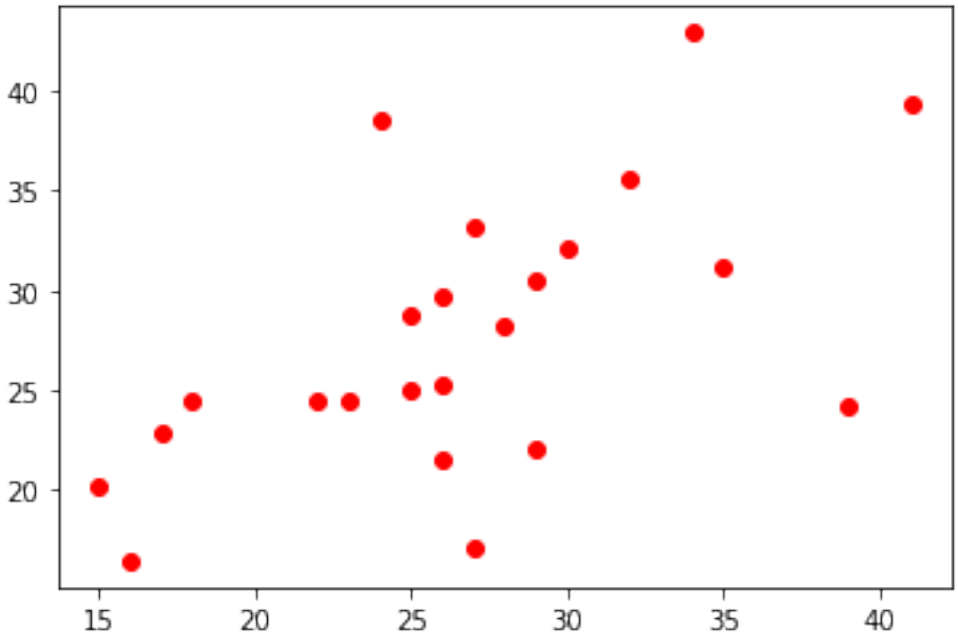
[124]:
```
predictions
```

[124]:
```
array([25.29467503, 21.3576901 , 16.50889931, 33.7973574 , 22.41489479,
       18.70365854, 22.33264278, 28.96384503, 27.29578344, 36.900505  ,
       21.74752016, 27.86623822, 24.06110604, 28.420299  , 19.78945285,
       28.74599123, 20.4768831 , 25.22929975, 20.96168256, 29.53491452,
       22.65070761, 35.37760019, 30.26742226])
```

[ ]:
```
# now the y_test contains the correct RFA's and we want to find out how far off
# the predictions above are off from the correct answers in y_test
```

[91]:
```
# can visually analyze this with scatter plot
# results not good
plt.scatter(y_test, predictions, color = 'red')
```

[91]:
```
<matplotlib.collections.PathCollection at 0x7f988a019670>
```

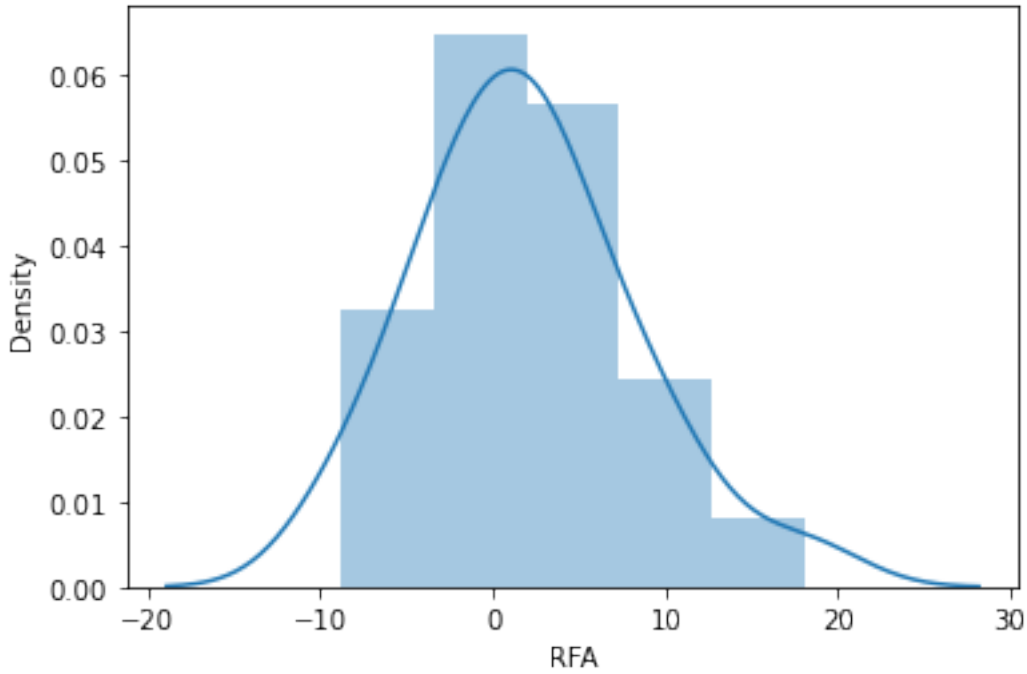


[126]:
```
# histogram of residuals
# these are right skewed, so Linear Model may not be the correct model here
sns.distplot((y_test-predictions))
```

```
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
```

```
  warnings.warn(msg, FutureWarning)
```

[126]:
```
<AxesSubplot:xlabel='RFA', ylabel='Density'>
```



[ ]:
```
# evaluation metrics
# MAE - easiest to understand, it's the average error
# MSE - more popular, it penalizes larger errors, more useful in real world data
# RMSE - even more popular b/c RMSE is interpretable in the 'y' units
# they are all loss functions, b/c we want to minimize them in order to create␣
 ↪best model
```

[92]:
```
from sklearn import metrics
```

[129]:
```
# MAE
#metrics.mean_absolute_error(y_test,predictions)
```

[129]:
```
4.945846427300812
```

[93]:
```
# MSE
metrics.mean_squared_error(y_test,predictions)
```

[93]:
```
38.81846524094573
```

[94]:
```
# RMSE
np.sqrt(metrics.mean_squared_error(y_test,predictions))
```

[94]:
```
6.23044663254134
```

[95]:
```
lm.score(X,y)
```

[95]:
```
0.5285227331696887
```

[96]:
```
r2 = lm.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[96]:
```
0.3742210822070414
```

14 Model 4 summary:

r2 = 0.529 adjusted r2 = 0.374 RMSE = 6.230 MSE = 38.818

15 Model 5 (changing response var to LFA)

[97]:
```
# 5th attempt: trying to predict LFA now, with RFA and Shoe size

# create training and testing data

y = df['LFA']
X = df[['RFA','Shoe_size']]
```

[98]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

[99]:
```
lm = LinearRegression()
```

[100]:
```
lm.fit(X_train,y_train)
```

[100]:
```
LinearRegression()
```

[101]:
```
# create dataframe off this
cdf = pd.DataFrame(lm.coef_,X.columns,columns=['Coeff'])
```

[102]:
```
cdf
```

[102]:

	Coeff
RFA	0.683438
Shoe_size	0.221443

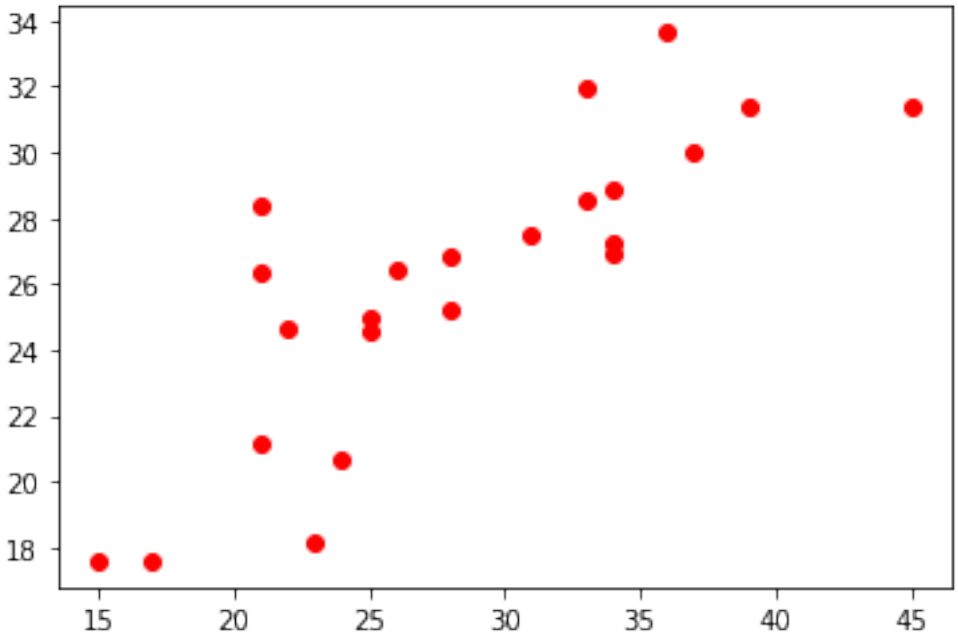
[103]:
```
# predictions
predictions = lm.predict(X_test)
```

[104]:
```
predictions
```

[104]:
```
array([27.49170576, 26.32716578, 26.80826818, 29.98490486, 18.18323892,
       24.66634187, 26.4761034 , 33.64264393, 31.35178001, 28.50730811,
       17.59141493, 24.53651227, 31.35178001, 31.96271201, 25.21994984,
       26.93809779, 28.85858091, 27.27026257, 17.59141493, 21.1193244 ,
       24.97939865, 28.41569453, 20.65733001])
```

[105]:
```
# can visually analyze this with scatter plot
# results: better, but still not great
plt.scatter(y_test, predictions, color = 'red')
```

[105]:
```
<matplotlib.collections.PathCollection at 0x7f98081165b0>
```

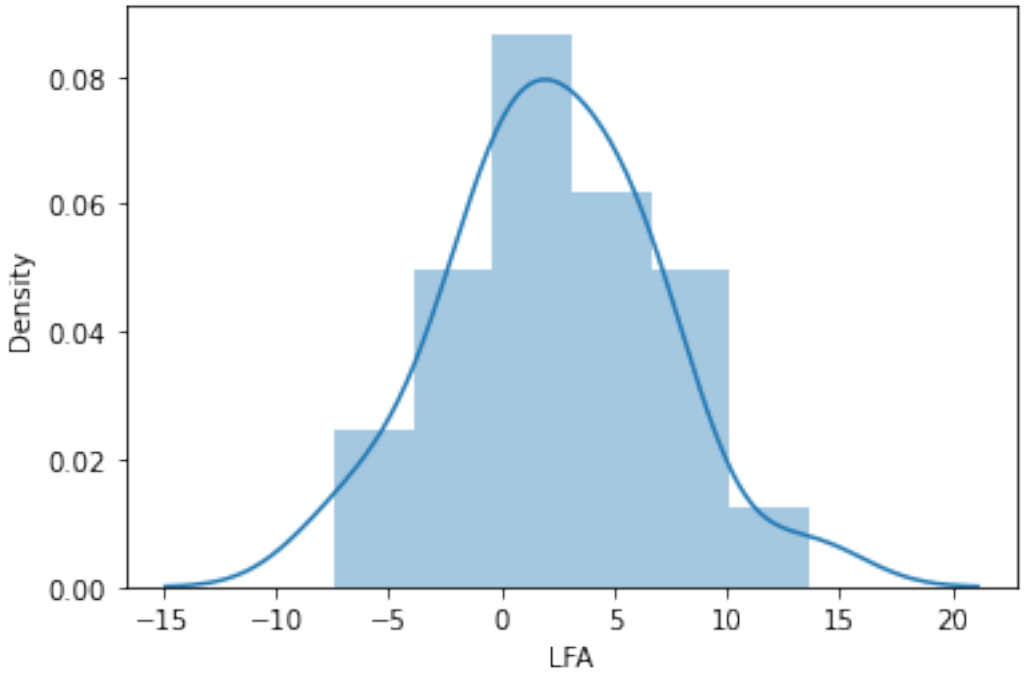


[106]:
```
# histogram of residuals
# small dataset causing some weirdness, so Linear Model may not be the correct␣
 ↪model here
sns.distplot((y_test-predictions))
```

```
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/seaborn/distributions.py:2619: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
```

```
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

[106]: <AxesSubplot:xlabel='LFA', ylabel='Density'>



[142]:
```
# MAE
#metrics.mean_absolute_error(y_test,predictions)
```

[142]: 5.435982124966751

[107]:
```
# MSE
metrics.mean_squared_error(y_test,predictions)
```

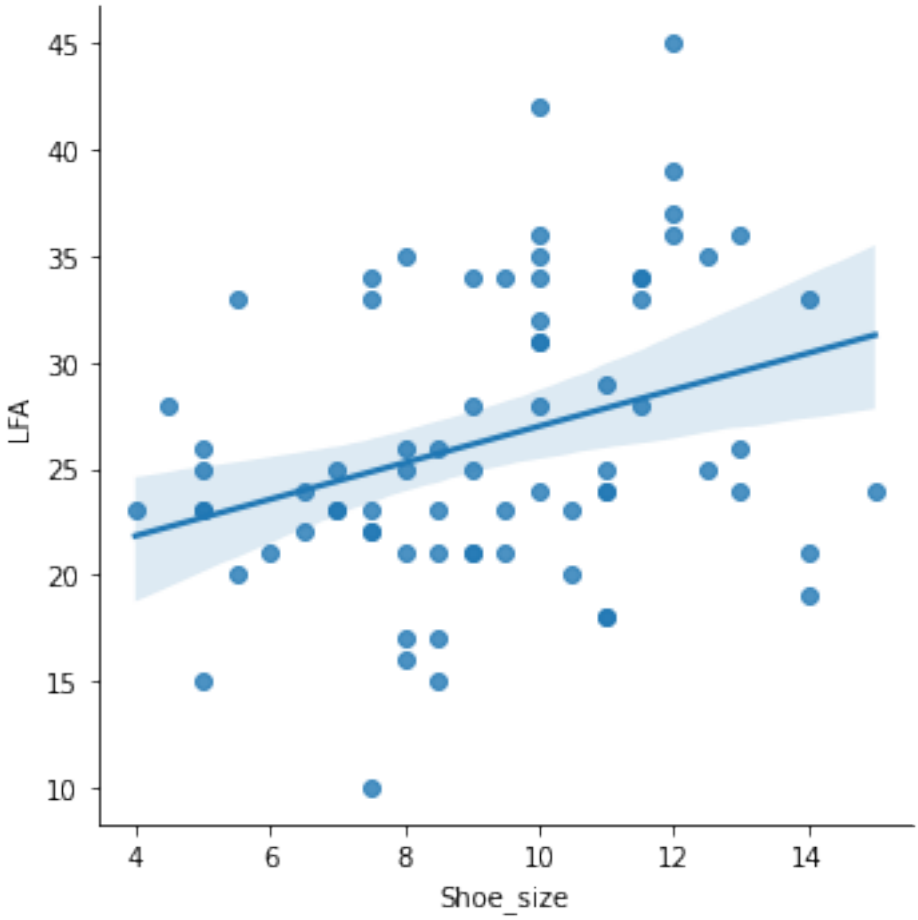
[107]: 25.922858876986904

[108]:
```
# RMSE
np.sqrt(metrics.mean_squared_error(y_test,predictions))
```

[108]: 5.091449585038323

[109]:
```
sns.lmplot(x='Shoe_size',y='LFA', data=df)
```

[109]: <seaborn.axisgrid.FacetGrid at 0x7f986b996730>



[110]:
```
lm.score(X,y)
```

[110]: 0.5371819105819523

[111]:
```
r2 = lm.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[111]: 0.5241447813025706

16 Model 5 summary:

r2 = 0.537 adjusted r2 = 0.524 RMSE = 5.091 MSE = 25.923

Notes: similar results to predicting RFA

[147]:
```
# Decision Trees and Random Forest

from sklearn.model_selection import train_test_split
```

[148]:
```
X = df.drop('RFA', axis=1)
```

[149]:
```
y = df['RFA']
```

[150]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

[151]:
```
from sklearn.tree import DecisionTreeClassifier
```

[152]:
```
dtree = DecisionTreeClassifier()
```

[153]:
```
dtree.fit(X_train,y_train)
```

[153]:
```
DecisionTreeClassifier()
```

[154]:
```
predictions = dtree.predict(X_test)
```

[155]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[156]:
```
print(confusion_matrix(y_test,predictions))
print('\n')
print(classification_report(y_test,predictions))
```

```
[[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 1 0 0 0 2 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]]


              precision    recall  f1-score   support
```

```
           16       0.00      0.00      0.00         1
           18       0.00      0.00      0.00         1
           19       0.00      0.00      0.00         1
           20       0.00      0.00      0.00         2
           23       0.00      0.00      0.00         0
           24       0.00      0.00      0.00         4
           25       0.00      0.00      0.00         0
           26       0.00      0.00      0.00         2
           27       0.00      0.00      0.00         1
           28       0.00      0.00      0.00         2
           29       0.00      0.00      0.00         1
           30       0.00      0.00      0.00         1
           31       0.00      0.00      0.00         2
           32       0.00      0.00      0.00         0
           35       0.33      1.00      0.50         1
           36       0.00      0.00      0.00         1
           37       0.00      0.00      0.00         1
           38       0.00      0.00      0.00         1
           39       0.00      0.00      0.00         1

    accuracy                           0.04        23
   macro avg       0.02      0.05      0.03        23
weighted avg       0.01      0.04      0.02        23


/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
```

```
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[157]:
```
# random forest

from sklearn.ensemble import RandomForestClassifier
```

[162]:
```
rfc = RandomForestClassifier(n_estimators=200)
```

[163]:
```
rfc.fit(X_train,y_train)
```

[163]:
```
RandomForestClassifier(n_estimators=200)
```

[164]:
```
rfc_pred = rfc.predict(X_test)
```

[165]:
```
print(confusion_matrix(y_test,rfc_pred))
print('\n')
print(classification_report(y_test,rfc_pred))
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

```
              precision    recall  f1-score   support

          15       0.00      0.00      0.00       0.0
          16       0.00      0.00      0.00       1.0
          18       0.00      0.00      0.00       1.0
          19       0.00      0.00      0.00       1.0
          20       0.00      0.00      0.00       2.0
          22       0.00      0.00      0.00       0.0
          23       0.00      0.00      0.00       0.0
          24       0.00      0.00      0.00       4.0
          25       0.00      0.00      0.00       0.0
          26       0.00      0.00      0.00       2.0
          27       0.00      0.00      0.00       1.0
          28       0.00      0.00      0.00       2.0
          29       0.00      0.00      0.00       1.0
          30       0.00      0.00      0.00       1.0
          31       0.00      0.00      0.00       2.0
          34       0.00      0.00      0.00       0.0
          35       0.00      0.00      0.00       1.0
          36       0.00      0.00      0.00       1.0
          37       0.00      0.00      0.00       1.0
          38       0.00      0.00      0.00       1.0
          39       0.00      0.00      0.00       1.0

    accuracy                           0.00      23.0
   macro avg       0.00      0.00      0.00      23.0
weighted avg       0.00      0.00      0.00      23.0


/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

17 decision tree and random forest not optimal

[168]:
```
round(df['RFA'].mean(),2)
```

[168]:
```
26.54
```

18 Model 6 - RFA is response var

[ ]:
```
# attempt 6 - MLR
```

[185]:
```
X = df.drop('RFA', axis=1)
y = df['RFA']
```

[186]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

[187]:
```
lm = LinearRegression()
```

[188]:
```
lm.fit(X_train,y_train)
```

[188]:
```
LinearRegression()
```

[189]:
```
coeff_df = pd.DataFrame(lm.coef_,X.columns, columns=['Coefficient'])
```

[190]:
```
coeff_df
```

[190]:

	Coefficient
Shoe_size	-1.312614
LFA	0.636790
Sex_M	-5.159338
Gender_M	5.737025
Orthotics_Y	4.045764
LESI_Y	1.980370
Age_grp_18-25	1.923760
Age_grp_26-35	2.449215

```
Age_grp_36-45                      -2.585351
Age_grp_46-55                      -2.333902
Age_grp_56-75                       0.546278
Height_grp_4-9:5-1                 -3.020938
Height_grp_5-11:6-1                 1.548775
Height_grp_5-2:5-5                 -0.367624
Height_grp_5-6:5-10                 3.326675
Height_grp_6-2:6-5                 -1.486888
Weight_grp_100:125                 -0.409718
Weight_grp_126:140                 -1.707350
Weight_grp_141:170                 -1.975408
Weight_grp_171:200                  0.698296
Weight_grp_201:250+                 3.394180
Physical_grp_Active                 1.130943
Physical_grp_Highly Active         -1.793843
Physical_grp_Moderately Active     -1.232914
Physical_grp_Mostly Sedentary       1.209763
Physical_grp_Sedentary              0.686051
Sports_grp_Club                    -2.458839
Sports_grp_College                 -4.229646
Sports_grp_High School              3.584585
Sports_grp_None                    -0.538943
Sports_grp_Professional             0.000000
Sports_grp_Semi-Professional        3.642843
Visit_grp_Acute_onset_LEI          -0.149508
Visit_grp_Chronic_LEI              -2.252386
Visit_grp_Deformity_LE              5.748273
Visit_grp_Foot Ankle Pain          -1.729666
Visit_grp_Instability_LE_GR        -1.616713
```

[191]:
```
y_pred = lm.predict(X_test)
```

[192]:
```
# check differences btwn predicted and actual
new_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

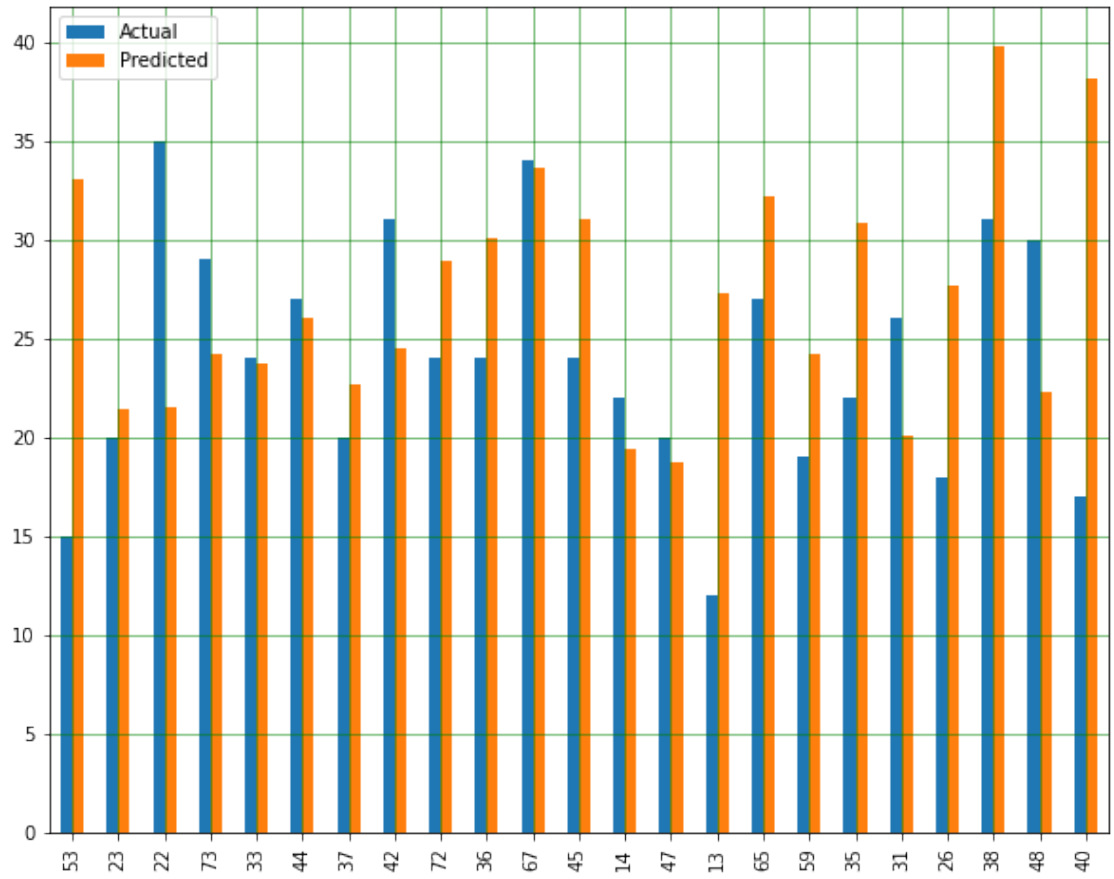
[193]:
```
new_df.head(10)
```

[193]:

	Actual	Predicted
53	15	33.045930
23	20	21.446149
22	35	21.565065
73	29	24.239780
33	24	23.725227
44	27	26.024768
37	20	22.712145
42	31	24.508255
72	24	28.889465

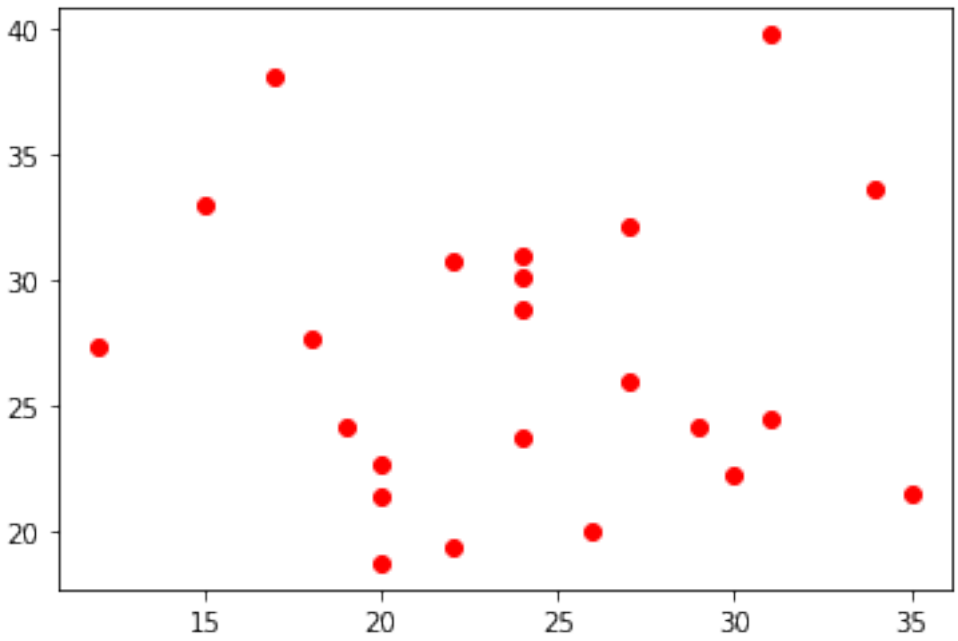
```
36      24  30.119530
```

[194]:
```
# plot actuals vs predicted
new_df.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



[195]:
```
# this is actually better
plt.scatter(y_test, y_pred, color = 'red')
```

[195]:
```
<matplotlib.collections.PathCollection at 0x7f986bf0df70>
```



[135]:
```
# another plot regression against actual data

#plt.figure(figsize=(12,6))
#plt.plot()
```

[196]:
```
# evaluation

RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_pred))))
MSE = mean_squared_error(y_test, y_pred)
#MAE = mean_absolute_error(y_test, y_pred)
#r2 = r2_score(y_test, y_pred)
#adj_r2 = 1-(1-r2) * (n-1)/(n-k-1)
#MAPE = np.mean(np.abs((y_test-y_pred/y_test)))*100
```

[197]:
```
print('RMSE = ', round(RMSE,2))
print('MSE = ', round(MSE,2))
#print('MAE = ', round(MAE,2))
#print('r2 = ', round(r2,2))
#print('Adjusted r2 = ', adj_r2)
#print('MAPE = ', round(MAPE,2))
```

```
RMSE =  8.79
MSE =  77.29
```

[198]:
```
print('Intercept: \n', lm.intercept_)
```

```
Intercept:
```

```
21.323397755913263
```

[199]:
```
lm.score(X,y)
```

[199]:
```
0.2740972366417632
```

[200]:
```
r2 = lm.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[200]:
```
-0.4719694923653135
```

19 Model 6 summary:

r2 = 0.274 adjusted r2 = -0.472 RMSE = 8.790 MSE = 77.290

Notes:

[ ]:
```
# thoughts:  the r2 score is strange, and I need to review each metric more␣
↪closely.
# we need to build an excel table for each attempt or model and list each metric
# this way we can compare them side by side
# i'll also review software for anything new that helps us with this
```

Multiple linear regression formula The formula for a multiple linear regression is:

20 y-hat = B0 + B1X1 + ... + BnXn + E

Multiple linear regression formula y-hat = the predicted value of the dependent variable B0 = the y-intercept (value of y when all other parameters are set to 0) B1X1= the regression coefficient (B1) of the first independent variable (X1) (a.k.a. the effect that increasing the value of the independent variable has on the predicted y value) ... = do the same for however many independent variables you are testing BnXn = the regression coefficient of the last independent variable e = model error (a.k.a. how much variation there is in our estimate of y) To find the best-fit line for each independent variable, multiple linear regression calculates three things:

The regression coefficients that lead to the smallest overall model error. The t-statistic of the overall model. The associated p-value (how likely it is that the t-statistic would have occurred by chance if the null hypothesis of no relationship between the independent and dependent variables was true). It then calculates the t-statistic and p-value for each regression coefficient in the model.

[ ]:
```
# a little tweak, same variables
# this is less about the line and more about the best fitting model
# want the minimum least sum of squared errors
# more variables, better r2 score
# need to use adjusted r2 score so we are forced to use the best variables
```

[192]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn
seaborn.set()
```

21 Model 7

[201]:
```
X = df.drop('RFA', axis=1)
y = df['RFA']
```

[202]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

[203]:
```
reg = LinearRegression()
```

[204]:
```
reg.fit(X,y)
```

[204]:
```
LinearRegression()
```

[205]:
```
reg.coef_
```

[205]:
```
array([ 0.25361105,  0.59429755, -2.87531365,  6.15073479,  1.86447745,
       -0.08210349,  1.35894471, -0.42795406, -0.622067  , -1.10002418,
        0.79110052, -1.58792525,  0.07554798,  1.88747304,  2.17123001,
       -2.54632578,  1.39912118,  1.44317819, -0.87422958, -2.22904157,
        0.26097177,  1.6145545 , -0.6593107 , -0.93950554, -0.8796896 ,
        0.86395135, -0.87942871, -3.25896205, -0.52942622,  0.3857796 ,
        0.52735193,  3.75468545, -1.33893815,  2.02517062, -0.91932994,
        0.84537389, -0.61227642])
```

[206]:
```
reg.intercept_
```

[206]:
```
5.838314383758146
```

[207]:
```
# calculating R-squared: 67%
reg.score(X,y)
```

[207]:
```
0.6972854321199535
```

22 calculating Adj. R-squared

$R^2_{adj.} = 1 - (1 - R^2) * \frac{n-1}{n-p-1}$

[200]:
```
X.shape
```

[200]:
```
(74, 33)
```

[ ]:
```
# above means:
# n = 74 (the number of observations)
# p = 33 (the number of predictors)
```

[208]:
```
# adjusted r2: 33%
r2 = reg.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[208]:
```
0.3861621262432391
```

[209]:
```
RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_pred))))
MSE = mean_squared_error(y_test, y_pred)
```

[210]:
```
print('RMSE = ', round(RMSE,2))
print('MSE = ', round(MSE,2))
```

```
RMSE =  10.01
MSE =  100.26
```

23 Model 7 summary:

r2 = 0.697 adjusted r2 = 0.386 RMSE = 10.010 MSE = 100.260

Notes:

[ ]:
```
# the adjusted r2 is considerably lower than the r2, which tells us, one or more
# of the predictors has little or no explanatory power.  Now we need to use
# other tools to understand which predictors are best and optimize the model
```

[ ]:
```
# feature selection:
# if a variable has a p-value > 0.05, we can disregard it
# feature_selection.f_regression:
# f-regression creates simple linear regressions of each feature and
# the dependent variable
# does many simple regressions, calcs the f-statistic and returns the p-values
```

[147]:
```
# feature selection

from sklearn.feature_selection import f_regression
```

[148]:
```
# returns 2 output arrays:
# 1st are the F-statistics to the regressions
# 2nd are the corresponding p-values
```

```
f_regression(X,y)
```

[148]:
```
(array([7.57253052e+00, 8.61798060e+01, 2.04605256e+01, 2.19824756e+01,
        2.35035060e-01, 1.84976594e-01, 1.88957411e-01, 1.34894486e+00,
        2.05655808e-02, 8.26898394e-01, 1.96954924e+00, 1.29457311e+00,
        6.14771986e+00, 4.41608338e+00, 4.73251923e-02, 1.08352040e+00,
        2.37856868e+00, 1.00592525e+00, 7.19698021e-03, 1.61553428e-01,
        3.09683451e+00, 1.08685064e+00, 6.92511337e-01, 6.82366204e-04,
        5.35544991e-02, 1.44443731e-02, 7.86080687e-02, 1.31907319e+00,
        7.07384624e-02, 4.95724998e-04, 2.92896579e-01, 1.56771694e+00,
        2.52201580e+00, 1.06092893e+00, 1.83137584e+00, 2.23748272e-01,
        1.23776607e+00]),
 array([7.49308466e-03, 6.21434554e-14, 2.34501553e-05, 1.27068511e-05,
        6.29286997e-01, 6.68415431e-01, 6.65085854e-01, 2.49298631e-01,
        8.86369322e-01, 3.66204927e-01, 1.64793885e-01, 2.58979007e-01,
        1.54997248e-02, 3.91018481e-02, 8.28400104e-01, 3.01394150e-01,
        1.27394174e-01, 3.19239884e-01, 9.32628006e-01, 6.88920991e-01,
        8.26922707e-02, 3.00657958e-01, 4.08063170e-01, 9.79232184e-01,
        8.17646144e-01, 9.04671546e-01, 7.79996661e-01, 2.54559860e-01,
        7.91024733e-01, 9.82298251e-01, 5.90040937e-01, 2.14591016e-01,
        1.16649079e-01, 3.06451145e-01, 1.80198825e-01, 6.37629549e-01,
        2.69602870e-01]))
```

[149]:
```
# since we only want the p-values:
# these are written in scientific notation:
# interpreting:  ex: e-11 = *10^(-11) = /10^11
# also, e-1 = *10^(-1) = /10

p_values = f_regression(X,y)[1]
p_values
```

[149]:
```
array([7.49308466e-03, 6.21434554e-14, 2.34501553e-05, 1.27068511e-05,
       6.29286997e-01, 6.68415431e-01, 6.65085854e-01, 2.49298631e-01,
       8.86369322e-01, 3.66204927e-01, 1.64793885e-01, 2.58979007e-01,
       1.54997248e-02, 3.91018481e-02, 8.28400104e-01, 3.01394150e-01,
       1.27394174e-01, 3.19239884e-01, 9.32628006e-01, 6.88920991e-01,
       8.26922707e-02, 3.00657958e-01, 4.08063170e-01, 9.79232184e-01,
       8.17646144e-01, 9.04671546e-01, 7.79996661e-01, 2.54559860e-01,
       7.91024733e-01, 9.82298251e-01, 5.90040937e-01, 2.14591016e-01,
       1.16649079e-01, 3.06451145e-01, 1.80198825e-01, 6.37629549e-01,
       2.69602870e-01])
```

[150]:
```
# since the above is confusing, we will round

p_values.round(3)
```

[150]:
```
array([0.007, 0.   , 0.   , 0.   , 0.629, 0.668, 0.665, 0.249, 0.886,
       0.366, 0.165, 0.259, 0.015, 0.039, 0.828, 0.301, 0.127, 0.319,
       0.933, 0.689, 0.083, 0.301, 0.408, 0.979, 0.818, 0.905, 0.78 ,
       0.255, 0.791, 0.982, 0.59 , 0.215, 0.117, 0.306, 0.18 , 0.638,
       0.27 ])
```

[151]:
```
X.columns
```

[151]:
```
Index(['Shoe_size', 'LFA', 'Sex_M', 'Gender_M', 'Orthotics_Y', 'LESI_Y',
       'Age_grp_18-25', 'Age_grp_26-35', 'Age_grp_36-45', 'Age_grp_46-55',
       'Age_grp_56-75', 'Height_grp_4-9:5-1', 'Height_grp_5-11:6-1',
       'Height_grp_5-2:5-5', 'Height_grp_5-6:5-10', 'Height_grp_6-2:6-5',
       'Weight_grp_100:125', 'Weight_grp_126:140', 'Weight_grp_141:170',
       'Weight_grp_171:200', 'Weight_grp_201:250+', 'Physical_grp_Active',
       'Physical_grp_Highly Active', 'Physical_grp_Moderately Active',
       'Physical_grp_Mostly Sedentary', 'Physical_grp_Sedentary',
       'Sports_grp_Club', 'Sports_grp_College', 'Sports_grp_High School',
       'Sports_grp_None', 'Sports_grp_Professional',
       'Sports_grp_Semi-Professional', 'Visit_grp_Acute_onset_LEI',
       'Visit_grp_Chronic_LEI', 'Visit_grp_Deformity_LE',
       'Visit_grp_Foot Ankle Pain', 'Visit_grp_Instability_LE_GR'],
      dtype='object')
```

[152]:
```
# using the above we can see that most variables are useless in predicting RFA
# variables we should keep, based on p-value:
# Shoe_size (0.007)
# LFA (0.)
# Sex_M (0.)
# Gender_M (0.)
# Height_grp_5-11:6-1 (0.015)
# Height_grp_5-2:5-5 (0.039)
# and that is it, based on p-values
```

24 Model 8 - post feature selection

[211]:
```
# attempt 7

X = df[['Shoe_size','LFA','Sex_M','Gender_M','Height_grp_5-11:
 ↪6-1','Height_grp_5-2:5-5']]
y = df['RFA']
```

[212]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

[213]:
```
reg = LinearRegression()
```

[214]:
```
reg.fit(X,y)
```

[214]:
```
LinearRegression()
```

[215]:
```
reg.coef_
```

[215]:
```
array([-0.03549516,  0.62105261,  1.44396601,  0.66382994, -0.49434478,
       -0.16576863])
```

[216]:
```
# calculating R-squared: 56%
reg.score(X,y)
```

[216]:
```
0.5613797982308107
```

[158]:
```
X.shape
```

[158]:
```
(74, 6)
```

[ ]:
```
# above means:
# n = 74 (the number of observations)
# p = 4 (the number of predictors)
```

[217]:
```
# adjusted r2: 52% -- this is up 20% from model 6, a significant improvement
r2 = reg.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[217]:
```
0.5221003771768535
```

[218]:
```
# evaluation

RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_pred))))
MSE = mean_squared_error(y_test, y_pred)
```

[219]:
```
print('RMSE = ', round(RMSE,2))
print('MSE = ', round(MSE,2))
```

```
RMSE =  8.43
MSE =  71.01
```

25 Model 8 summary:

r2 = 0.561 adjusted r2 = 0.522 RMSE = 8.430 MSE = 71.010

Notes:

[220]:
```
# creating a summary table

reg_summary = pd.DataFrame(data= X.columns.values, columns=['Features'])
reg_summary
```

[220]:

	Features
0	Shoe_size
1	LFA
2	Sex_M
3	Gender_M
4	Height_grp_5-11:6-1
5	Height_grp_5-2:5-5

[221]:
```
p_values = f_regression(X,y)[1]
p_values
```

[221]:
```
array([7.49308466e-03, 6.21434554e-14, 2.34501553e-05, 1.27068511e-05,
       1.54997248e-02, 3.91018481e-02])
```

[222]:
```
reg_summary['Coefficients'] = reg.coef_
reg_summary['p-values'] = p_values.round(3)
```

[223]:
```
reg_summary
```

[223]:

	Features	Coefficients	p-values
0	Shoe_size	-0.035495	0.007
1	LFA	0.621053	0.000
2	Sex_M	1.443966	0.000
3	Gender_M	0.663830	0.000
4	Height_grp_5-11:6-1	-0.494345	0.015
5	Height_grp_5-2:5-5	-0.165769	0.039

[ ]:
```
# stopping here 4/12/2022
# next is trying additional feature scaling techniques
# more to come
```

26 Model 9 - compare last model but change response var

[224]:
```
X = df[['Shoe_size','RFA','Sex_M','Gender_M','Height_grp_5-11:
 ↪6-1','Height_grp_5-2:5-5']]
y = df['LFA']
```

[225]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

[226]:
```
reg = LinearRegression()
```

[227]:
```
reg.fit(X,y)
```

[227]:
```
LinearRegression()
```

[228]:
```
# calculating R-squared: 59%
reg.score(X,y)
```

[228]:
```
0.5859538640612207
```

[229]:
```
# adjusted r2: 55% -- this is up 20% from model 6, a significant improvement
r2 = reg.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[229]:
```
0.548875105618942
```

[230]:
```
# evaluation

RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_pred))))
MSE = mean_squared_error(y_test, y_pred)
```

[231]:
```
print('RMSE = ', round(RMSE,2))
print('MSE = ', round(MSE,2))
```

```
RMSE =  9.48
MSE =  89.97
```

[232]:
```
# create summary table
reg_summary = pd.DataFrame(data= X.columns.values, columns=['Features'])
p_values = f_regression(X,y)[1]
```

[233]:
```
reg_summary['Coefficients'] = reg.coef_
reg_summary['p-values'] = p_values.round(3)
```

[234]:
```
reg_summary
```

[234]:

	Features	Coefficients	p-values
0	Shoe_size	-0.028271	0.005
1	RFA	0.680330	0.000
2	Sex_M	-2.438274	0.000
3	Gender_M	5.824587	0.000
4	Height_grp_5-11:6-1	0.677710	0.004
5	Height_grp_5-2:5-5	1.401073	0.059

27 Model 9 summary:

r2 = 0.586 adjusted r2 = 0.549 RMSE = 9.480 MSE = 89.970

Notes:

28 Stepwise regression: backwards elimination

29 Model 10: y=RFA

[24]:
```
X = df.drop('RFA', axis=1)
y = df['RFA']
```

[25]:
```
# gather model statistics
# use statmodels.api library
# create function which grabs the columns of interest from a list,
# then fits an ordinary least squares (OLS) linear model to it.
```

[26]:
```
import statsmodels.api as sm
```

[27]:
```
def get_stats():
    results = sm.OLS(y,X).fit()
    print(results.summary())
```

[28]:
```
get_stats()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    RFA   R-squared:                       0.697
Model:                            OLS   Adj. R-squared:                  0.474
Method:                 Least Squares   F-statistic:                     3.121
Date:                Tue, 19 Apr 2022   Prob (F-statistic):           0.000338
Time:                        19:25:12   Log-Likelihood:                -197.70
No. Observations:                  74   AIC:                             459.4
Df Residuals:                      42   BIC:                             533.1
Df Model:                          31
Covariance Type:            nonrobust
================================================================================
==================
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
------------------
Shoe_size                        0.2536      0.590      0.430      0.670
-0.937       1.445
LFA                              0.5943      0.117      5.061      0.000
0.357       0.831
Sex_M                           -2.8753      5.450     -0.528      0.601
-13.875       8.124
Gender_M                         6.1507      6.609      0.931      0.357
-7.187      19.489
```

```
Orthotics_Y                        1.8645      1.626      1.146      0.258
-1.418       5.147
LESI_Y                            -0.0821      1.394     -0.059      0.953
-2.896       2.732
Age_grp_18-25                      2.3598      2.188      1.079      0.287
-2.055       6.775
Age_grp_26-35                      0.5729      1.834      0.312      0.756
-3.129       4.275
Age_grp_36-45                      0.3788      1.772      0.214      0.832
-3.196       3.954
Age_grp_46-55                     -0.0992      1.812     -0.055      0.957
-3.757       3.558
Age_grp_56-75                      1.7920      1.945      0.922      0.362
-2.132       5.716
Height_grp_4-9:5-1                -0.5871      2.837     -0.207      0.837
-6.312       5.138
Height_grp_5-11:6-1                1.0764      2.851      0.378      0.708
-4.677       6.830
Height_grp_5-2:5-5                 2.8883      1.740      1.660      0.104
-0.623       6.399
Height_grp_5-6:5-10                3.1721      2.084      1.522      0.135
-1.033       7.377
Height_grp_6-2:6-5                -1.5455      3.729     -0.414      0.681
-9.071       5.980
Weight_grp_100:125                 2.4000      1.879      1.277      0.209
-1.393       6.193
Weight_grp_126:140                 2.4440      2.094      1.167      0.250
-1.783       6.671
Weight_grp_141:170                 0.1266      1.624      0.078      0.938
-3.152       3.405
Weight_grp_171:200                -1.2282      1.950     -0.630      0.532
-5.164       2.707
Weight_grp_201:250+                1.2618      2.037      0.620      0.539
-2.848       5.372
Physical_grp_Active                2.6154      1.860      1.406      0.167
-1.138       6.369
Physical_grp_Highly Active         0.3415      1.654      0.206      0.837
-2.996       3.680
Physical_grp_Moderately Active     0.0613      1.999      0.031      0.976
-3.973       4.096
Physical_grp_Mostly Sedentary      0.1212      3.495      0.035      0.973
-6.933       7.175
Physical_grp_Sedentary             1.8648      3.876      0.481      0.633
-5.957       9.687
Sports_grp_Club                   -0.0454      2.164     -0.021      0.983
-4.412       4.321
Sports_grp_College                -2.4249      2.124     -1.142      0.260
-6.711       1.861
```

```
Sports_grp_High School              0.3046      1.793      0.170      0.866
-3.314       3.923
Sports_grp_None                     1.2198      1.636      0.746      0.460
-2.082       4.522
Sports_grp_Professional             1.3614      5.366      0.254      0.801
-9.468      12.191
Sports_grp_Semi-Professional        4.5887      2.311      1.986      0.054
-0.075       9.252
Visit_grp_Acute_onset_LEI          -0.3381      1.921     -0.176      0.861
-4.215       3.539
Visit_grp_Chronic_LEI               3.0260      1.707      1.772      0.084
-0.419       6.471
Visit_grp_Deformity_LE              0.0815      3.154      0.026      0.979
-6.283       6.446
Visit_grp_Foot Ankle Pain           1.8462      1.443      1.279      0.208
-1.067       4.759
Visit_grp_Instability_LE_GR         0.3886      2.754      0.141      0.888
-5.170       5.947
==============================================================================
Omnibus:                        0.092   Durbin-Watson:                   1.652
Prob(Omnibus):                  0.955   Jarque-Bera (JB):                0.224
Skew:                          -0.073   Prob(JB):                        0.894
Kurtosis:                       2.774   Cond. No.                     1.11e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 5.01e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

[ ]:
```
# read this article and included articles to understand more about the report␣
↪above:
# https://towardsdatascience.com/
↪stepwise-regression-tutorial-in-python-ebf7c782c922
```

[29]:
```
def stepwise_selection(X,y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out=0.05,
                       verbose=True):
    """Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (cols of X)
```

```
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite loop
    """
    included = list(initial_list)
    while True:
        changed = False
        # forward step
        excluded = list(set(X)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.
↪DataFrame(X[included+[new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add {:30} with p-value {:.6}'.format(best_feature,␣
↪best_pval))

        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max()  # null if pvalues empty
        if worst_pval > threshold_out:
            changed = True
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature,␣
↪worst_pval))
        if not changed:
            break
    return included

result = stepwise_selection(X,y)
print('Resulting features:')
print(result)
```

```
/var/folders/n9/llmkvp611qbfrmrpjvyc70t80000gp/T/ipykernel_89114/1068618509.py:2
3: DeprecationWarning: The default dtype for empty Series will be 'object'
```

```
instead of 'float64' in a future version. Specify a dtype explicitly to silence
this warning.
  new_pval = pd.Series(index=excluded)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
```

```
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
```

```
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
```

```
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
```

```
  x = pd.concat(x[::order], 1)
/var/folders/n9/llmkvp611qbfrmrpjvyc70t80000gp/T/ipykernel_89114/1068618509.py:2
3: DeprecationWarning: The default dtype for empty Series will be 'object'
instead of 'float64' in a future version. Specify a dtype explicitly to silence
this warning.
  new_pval = pd.Series(index=excluded)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
```

```
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
```

```
Add LFA                           with p-value 6.21435e-14
Resulting features:
['LFA']
```

```
/Users/chossack/opt/anaconda3/lib/python3.9/site-
```

```
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
```

```
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
/Users/chossack/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
```

```
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
```

30 OLS model # 2

31 Model 11

[ ]:

[30]:
```
X = df['LFA']
y = df['RFA']
```

[31]:
```
def get_stats():
    results = sm.OLS(y,X).fit()
    print(results.summary())
```

[32]:
```
get_stats()
```

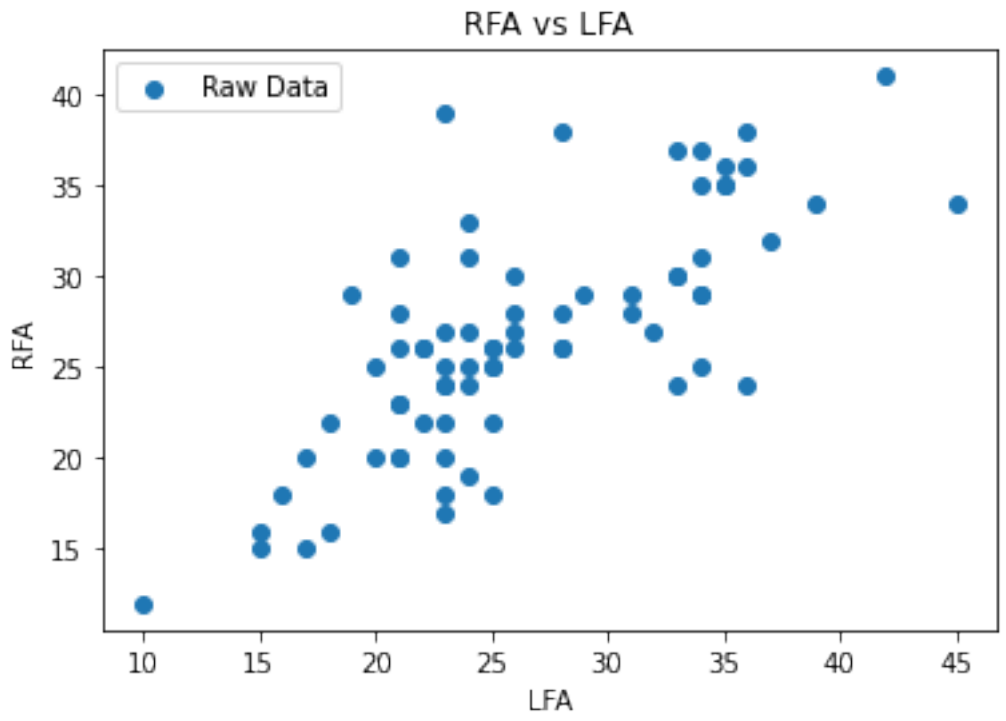
```
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                    RFA   R-squared (uncentered):
0.969
Model:                            OLS   Adj. R-squared (uncentered):
0.969
Method:                 Least Squares   F-statistic:
2293.
Date:                Tue, 19 Apr 2022   Prob (F-statistic):
6.81e-57
Time:                        20:02:49   Log-Likelihood:
-220.98
No. Observations:                  74   AIC:
444.0
Df Residuals:                      73   BIC:
446.3
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
LFA            0.9868      0.021     47.889      0.000       0.946       1.028
================================================================================
Omnibus:                        5.091   Durbin-Watson:                   1.712
Prob(Omnibus):                  0.078   Jarque-Bera (JB):                5.548
Skew:                           0.285   Prob(JB):                       0.0624
Kurtosis:                       4.215   Cond. No.                         1.00
```

```
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

32 Model 12: simple linear regression

[33]:
```
lfa = df['LFA']
rfa = df['RFA']
plt.scatter(lfa,rfa, label='Raw Data')
plt.title('RFA vs LFA')
plt.xlabel('LFA')
plt.ylabel('RFA')
plt.legend()
```

[33]:
```
<matplotlib.legend.Legend at 0x7fa1d99ce760>
```



[34]:
```
df2 = df[['RFA','LFA']]
```

[35]:
```
df2.columns
```

[35]:
```
Index(['RFA', 'LFA'], dtype='object')
```

[36]:
```
# build model and train it
parameters = {'alpha': 40, 'beta': 4}

def y_hat(left, params):
    alpha = params['alpha']
    beta = params['beta']
    return alpha + beta * left
y_hat(5, parameters)
```

[36]:
```
60
```

[37]:
```
# train the model
def learn_parameters(data, params):
    x, y = df2['LFA'], df2['RFA']
    x_bar, y_bar = x.mean(), y.mean()
    x, y = x.to_numpy(), y.to_numpy()
    beta = sum( ((x-x_bar) * (y-y_bar)) / sum( (x-x_bar)**2))
    alpha = y_bar - beta * x_bar
    params['alpha'] = alpha
    params['beta'] = beta
```

[38]:
```
new_parameter = {'alpha' : 0, 'beta' : 0}
learn_parameters(df2, new_parameter)
new_parameter
```

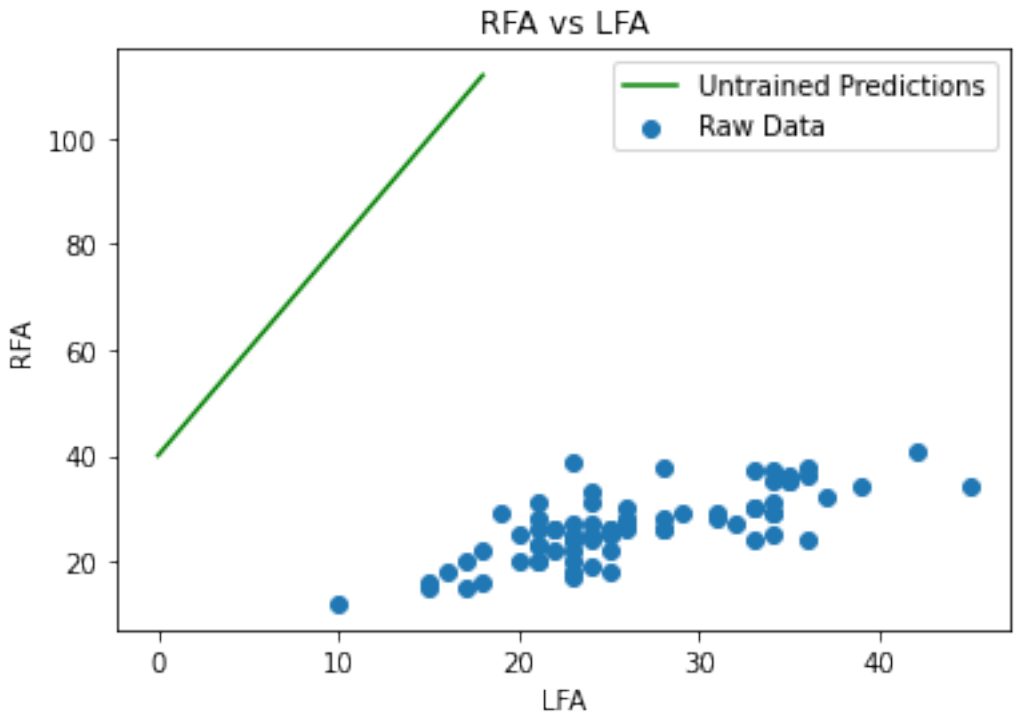
[38]:
```
{'alpha': 8.484833164812944, 'beta': 0.6851909465660729}
```

[39]:
```
spaced_lfas = list(range(19))
spaced_untrained_predictions = [y_hat(x, parameters) for x in spaced_lfas]
print(spaced_untrained_predictions)
```

```
[40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100, 104, 108, 112]
```

[40]:
```
lfa = df2['LFA']
rfa = df2['RFA']
plt.scatter(lfa,rfa, label='Raw Data')
plt.plot(spaced_lfas, spaced_untrained_predictions, label='Untrained␣
 ↪Predictions', color='green')
plt.title('RFA vs LFA')
plt.xlabel('LFA')
plt.ylabel('RFA')
plt.legend()
```

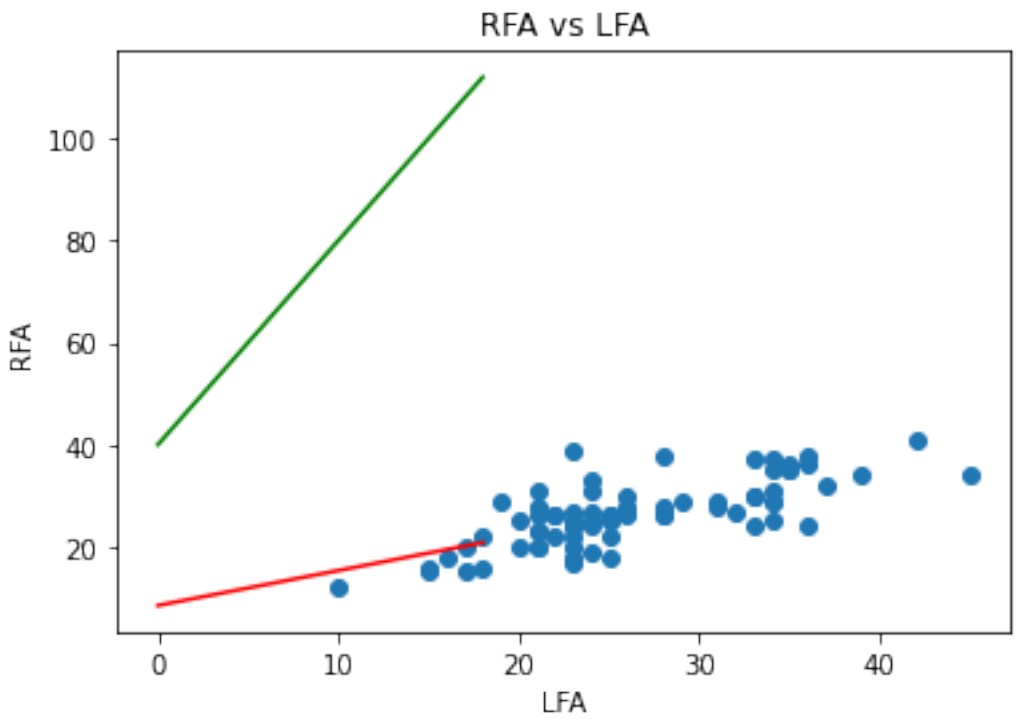
[40]:
```
<matplotlib.legend.Legend at 0x7fa1e9c5c4c0>
```



[41]:
```
# above accuracy is poor as you can see in the graph
# now we train the model
spaced_trained_predictions = [y_hat(x, new_parameter) for x in spaced_lfas]
print(spaced_trained_predictions)
plt.scatter(lfa,rfa, label='Raw Data')
plt.plot(spaced_lfas, spaced_untrained_predictions, label='Untrained␣
 ↪Predictions', color='green')
plt.plot(spaced_lfas, spaced_trained_predictions, label='Trained Predictions',␣
 ↪color='red')
plt.title('RFA vs LFA')
plt.xlabel('LFA')
plt.ylabel('RFA')
```

```
[8.484833164812944, 9.170024111379016, 9.85521505794509, 10.540406004511162,
11.225596951077236, 11.910787897643308, 12.59597884420938, 13.281169790775454,
13.966360737341528, 14.6515516839076, 15.336742630473672, 16.021933577039746,
16.70712452360582, 17.392315470171894, 18.077506416737965, 18.76269736330404,
19.44788830987011, 20.133079256436183, 20.818270203002257]
```

[41]:
```
Text(0, 0.5, 'RFA')
```



[ ]:
```
# above the green is on untrained predictions
# above the red is on trained predictions - much closer to actuals
```

[42]:
```
# make predictions on unseen data:
new_lfas = int(input('Enter LFA to predict RFA: '))
y_hat(new_lfas, new_parameter)
```

```
Enter LFA to predict RFA: 28
```

[42]:
```
27.670179668662986
```

[ ]:
```
# y = mx + b
# b is the intercept
# m is the slope
```

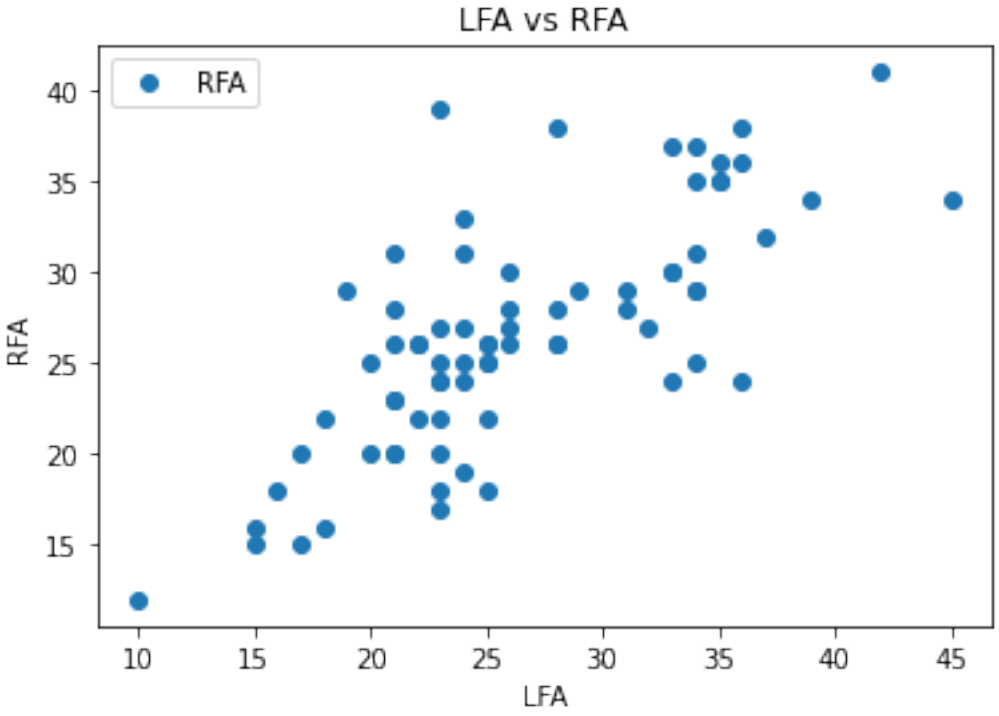
[43]:
```
df2.describe()
```

[43]:

	RFA	LFA
count	74.000000	74.000000
mean	26.540541	26.351351
std	6.404598	6.899340
min	12.000000	10.000000
25%	22.250000	22.000000
50%	26.000000	25.000000

```
75%     30.000000  33.000000
max     41.000000  45.000000
```

[44]:
```
df2.plot(x='LFA', y='RFA', style='o')
plt.title('LFA vs RFA')
plt.xlabel('LFA')
plt.ylabel('RFA')
plt.show()
```



[45]:
```
X = df2.iloc[:, :-1].values
y = df2.iloc[:, 1].values
```

[47]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[48]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

[48]:
```
LinearRegression()
```

[49]:
```
regressor.intercept_
```

[49]:
```
4.691387524951548
```

[51]:
```
regressor.coef_
```

[51]:
```
array([0.82272376])
```

[52]:
```
# this means that for every one unit of change in LFA, the change in RFA is␣
 ↪about 0.82%
# if the LFA is one degree higher, you can expect the RFA to be about 1%␣
 ↪greater than that
# this is not a great way to predict
# you want to predict hours studying and test score achieved, etc but for this␣
 ↪data
# this is how I would interpret
```

[53]:
```
# making predictions
y_pred = regressor.predict(X_test)
```

[54]:
```
df3 = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df3
```

[54]:

	Actual	Predicted
0	29	28.550377
1	15	17.032244
2	23	21.145863
3	26	26.904929
4	25	19.500415
5	18	22.791310
6	33	35.132167
7	21	23.614034
8	22	22.791310
9	24	31.841272
10	26	29.373100
11	45	32.663995
12	28	35.954891
13	24	24.436758
14	25	25.259482

[55]:
```
from sklearn import metrics
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_pred)))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
```

```
Root Mean Squared Error: 4.915114060558089
Mean Squared Error: 24.158346228295827
```

[56]:
```
# calculating R-squared: 54%
regressor.score(X,y)
```

[56]:
```
0.5435096401074273
```

[57]:
```
# adjusted r2:
r2 = regressor.score(X,y)
n = X.shape[0]
p = X.shape[1]

adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
adjusted_r2
```

[57]:
```
0.5371694962200304
```

[ ]:
```
# common way to check for multicollinearity is the Variance Inflation Factor␣
↪(VIF)
# VIF = 1, low
# VIF = <5, moderate
# VIF = >5, high
```

[ ]:
```
# compute VIF scores:
```

[ ]:
```
# the ai university -  youtube
# https://www.youtube.com/watch?v=mS-TFiXoh8E
```

Articles:

https://medium.datadriveninvestor.com/exploratory-data-analysis-in-python-a3b53fadb421

https://datatofish.com/read_excel/

https://inria.github.io/scikit-learn-mooc/python_scripts/03_categorical_pipeline_column_transformer.html

https://medium.com/analytics-vidhya/exploratory-data-analysis-eda-in-python-cf757afa2d2d

https://medium.com/swlh/exploratory-data-analysis-eda-from-scratch-in-python-8c12c2673aa7

https://towardsdatascience.com/an-extensive-guide-to-exploratory-data-analysis-ddd99a03199e

https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html

https://towardsdatascience.com/beware-of-the-dummy-variable-trap-in-pandas-727e8e6b8bde

https://towardsdatascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db

https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/

https://www.askpython.com/python/examples/correlation-regression-analysis

https://www.analyticsvidhya.com/blog/2022/02/exploratory-data-analysis-in-python/

https://towardsdatascience.com/exploratory-data-analysis-eda-python-87178e35b14

https://www.dataquest.io/blog/how-to-analyze-survey-data-python-beginner/

https://www.absentdata.com/pandas/pandas-cut-continuous-to-categorical/

https://www.statology.org/pandas-get-dummies/

https://www.statology.org/dummy-variables-regression/

https://www.youtube.com/watch?v=EXzTeXpQzi4

https://plotly.com/python/linear-fits/

https://vbsreddy1.medium.com/unboundlocalerror-when-the-variable-has-a-value-in-python-e34e097547d6

https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/

https://stackoverflow.com/questions/55229301/one-hot-encoding-multiple-columns-in-sklearn-and-naming-columns

https://stackoverflow.com/questions/60742878/getting-error-shape-of-passed-values-is-8708-27-indices-imply-8708-4-wh

https://realpython.com/linear-regression-in-python/

https://towardsdatascience.com/introduction-to-linear-regression-in-python-c12a072bedf0

https://towardsdatascience.com/what-are-rmse-and-mae-e405ce230383

https://statisticsbyjim.com/regression/interpret-r-squared-regression/

33 some good youtube stats lectures: statquest with josh starmer

https://www.youtube.com/watch?v=2AQKmw14mHM

https://medium.com/analytics-vidhya/multiple-linear-regression-with-python-98f4a7f1c26c

https://medium.com/analytics-vidhya/multiple-linear-regression-with-python-98f4a7f1c26c

https://towardsdatascience.com/introduction-to-linear-regression-in-python-c12a072bedf0

https://www.scribbr.com/statistics/multiple-linear-regression/

https://www.geeksforgeeks.org/interpreting-the-results-of-linear-regression-using-ols-summary/

https://medium.com/swlh/interpreting-linear-regression-through-statsmodels-summary-4796d359035a

https://towardsdatascience.com/stepwise-regression-tutorial-in-python-ebf7c782c922

https://medium.com/mlearning-ai/short-python-code-for-backward-elimination-with-detailed-explanation-52894a9a7880

https://www.analyticsvidhya.com/blog/2021/03/a-practical-tutorial-to-simple-linear-regression-using-python/

https://stackabuse.com/linear-regression-in-python-with-scikit-learn/

[ ]: