Used OLS module of statsmodels library of python to build linear regression models and pandas library for data analysis.

1) For the final project of my Data science course, I worked on the 2015 County Demographics and unemployment rates dataset.
2) The objective of the project is to predict the unemployment rates of the counties for the year 2017 based on the unemployment rates of counties in the year 2015 using linear regression
3) The dataset has the following columns:

```
Index(['CensusId', 'State', 'County_x', 'TotalPop',
       'Percent Population with age between 16 and 44',
       'Percent Population with age between 45 and 74',
       'Percent Population with age between 75 and over', 'Men', 'Women',
       'Hispanic', 'White', 'Black', 'Native', 'Asian', 'Pacific', 'Citizen',
       'Income', 'IncomeErr', 'IncomePerCap', 'IncomePerCapErr', 'Poverty',
       'ChildPoverty', 'Professional', 'Service', 'Office', 'Construction',
       'Production', 'Drive', 'Carpool', 'Transit', 'Walk', 'OtherTransp',
       'WorkAtHome', 'MeanCommute', 'Unemployment', 'Class', 'Id2',
       'Unemployment rate', 'County_y', 'County Unemployment Rate'],
      dtype='object')
```

4) Solution Flow Chart:

   i) First, I processed the datasets to identify any missing values and replaced them with either mean or median

```
Data columns (total 35 columns):
CensusId                                           3220 non-null int64
State                                              3220 non-null object
County                                             3220 non-null object
TotalPop                                           3220 non-null int64
Percent Population with age between 16 and 44      3220 non-null float64
Percent Population with age between 45 and 74      3220 non-null float64
Percent Population with age between 75 and over    3220 non-null float64
Men                                                3220 non-null int64
Women                                              3220 non-null int64
Hispanic                                           3220 non-null float64
White                                              3220 non-null float64
Black                                              3220 non-null float64
Native                                             3220 non-null float64
Asian                                              3220 non-null float64
Pacific                                            3220 non-null float64
Citizen                                            3220 non-null int64
Income                                             3219 non-null float64
IncomeErr                                          3219 non-null float64
IncomePerCap                                       3220 non-null int64
IncomePerCapErr                                    3220 non-null int64
Poverty                                            3220 non-null float64
ChildPoverty                                       3219 non-null float64
Professional                                       3220 non-null float64
Service                                            3220 non-null float64
Office                                             3220 non-null float64
Construction                                       3220 non-null float64
Production                                         3220 non-null float64
Drive                                              3220 non-null float64
Carpool                                            3220 non-null float64
Transit                                            3220 non-null float64
Walk                                               3220 non-null float64
OtherTransp                                        3220 non-null float64
WorkAtHome                                         3220 non-null float64
MeanCommute                                        3220 non-null float64
Unemployment                                       3220 non-null float64
dtypes: float64(26), int64(7), object(2)
memory usage: 880.5+ KB
```

```python
# There are some missing values in the Income column
# Replacing using median
median = data['Income'].median()
data['Income'].fillna(median, inplace=True)
```

```
C:\Users\kalya\Anaconda3\lib\site-packages\pandas\core\gene
A value is trying to be set on a copy of a slice from a Dat

See the caveats in the documentation: http://pandas.pydata.
-versus-copy
  self._update_inplace(new_data)
```

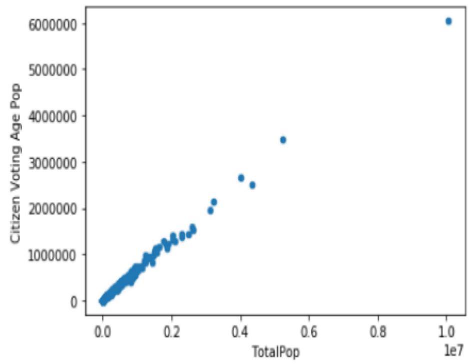
```python
# There are some missing values in the ChildPoverty column
# Replacing using median
median = data['ChildPoverty'].median()
data['ChildPoverty'].fillna(median, inplace=True)
```

   The columns Income and Child poverty have missing values. I replaced them with the median of their other values:

   ii) I removed attributes that cause multicollinearity (Correlation between the predictors itself).

```
data_new.plot.scatter(x= "TotalPop", y = 'Citizen Voting Age Pop')
r = np.corrcoef(data_new[["TotalPop", "Citizen Voting Age Pop"]].transpose())
r[0,1]**2
```

```
0.9927694507810428
```



The variable Citizen Voting age population and total age have a strong correlation(0.99). So, I removed one of them:

```
#removing "Citizen Voting Age Pop"
data_new.drop(['Citizen Voting Age Pop'], axis = 1, inplace = True)
```

iii) I explored the data and removed any irrelevant or redundant variables.

```
X = data_new[["Men"]]
Y = data_new["Unemployment"]
X = sm.add_constant(X)
results3 = sm.OLS(Y,X).fit()
print(results3.summary())
#SSE = round(results3.ssr,3)
#SSE
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Unemployment   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                  0.001
```

The model with the “Men” variable has a very small R2 value. So, I dropped it:

```
#removing "Men"
data_new.drop(["Men"], axis = 1, inplace = True)
```

iv) I partitioned the dataset as training dataset and the test dataset using the hold-out method with 80% percent of data as training and 20% as validation data:

```
X_train, X_val= train_test_split(dataR, test_size = 0.2, random_state = 1)
x_train = X_train.iloc[:,4:28]
x_val =   X_val.iloc[:,4:28]
y_train = X_train.iloc[:,28:29]
y_val = X_val.iloc[:,28:29]
```

v) I standardized both training and validation datasets:

```
scaler = MinMaxScaler(feature_range=(0, 1))
x_train[['TotalPop', 'Percent Population with age between 16 and 44',
       'Percent Population with age between 45 and 74',
       'Percent Population with age between 75 and over', 'Hispanic', 'White',
       'Black', 'Native', 'Asian', 'Pacific', 'Median Household Income',
       'Poverty', 'Service', 'Office', 'Construction', 'Production', 'Drive',
       'Carpool', 'Transit', 'Walk', 'WorkAtHome', 'Mean Commute Time',
       'State Unemployment rate for 2013',
       'County Unemployment Rate for 2013']] = scaler.fit_transform(x_train[['TotalPop', 'Percent Population
 with age between 16 and 44',
       'Percent Population with age between 45 and 74',
       'Percent Population with age between 75 and over', 'Hispanic', 'White',
       'Black', 'Native', 'Asian', 'Pacific', 'Median Household Income',
       'Poverty', 'Service', 'Office', 'Construction', 'Production', 'Drive',
       'Carpool', 'Transit', 'Walk', 'WorkAtHome', 'Mean Commute Time',
       'State Unemployment rate for 2013',
       'County Unemployment Rate for 2013']])

x_val[['TotalPop', 'Percent Population with age between 16 and 44',
       'Percent Population with age between 45 and 74',
       'Percent Population with age between 75 and over', 'Hispanic', 'White',
       'Black', 'Native', 'Asian', 'Pacific', 'Median Household Income',
       'Poverty', 'Service', 'Office', 'Construction', 'Production', 'Drive',
       'Carpool', 'Transit', 'Walk', 'WorkAtHome', 'Mean Commute Time',
       'State Unemployment rate for 2013',
       'County Unemployment Rate for 2013']] = scaler.fit_transform(x_val[['TotalPop', 'Percent Population wi
th age between 16 and 44',
       'Percent Population with age between 45 and 74',
       'Percent Population with age between 75 and over', 'Hispanic', 'White',
       'Black', 'Native', 'Asian', 'Pacific', 'Median Household Income',
       'Poverty', 'Service', 'Office', 'Construction', 'Production', 'Drive',
       'Carpool', 'Transit', 'Walk', 'WorkAtHome', 'Mean Commute Time',
       'State Unemployment rate for 2013',
       'County Unemployment Rate for 2013']])
```

vi) I built linear regression models on the 2015 data set using the forward feature selection method and picked the best model based on the root mean squared errors on the test dataset.

Note: In the forward selection method, I first start with the null model (no variable only the intercept which would be either mean or median of the values of the response variable). Then, I build models with a single feature/variable and pick the best model based on the mean squared errors on the validation test dataset. Then, I build models with each of the remaining q-1 variables and select the best two-variable model. I continue like that until I left with no variables.

Theoretically, the best subset selection (checking every possible subset of feature) should give the best model but there is a computational problem associated with it. It is difficult to check all the 2^p subsets.

```
# Build model with 1 predictor (using statsmodels)
X = x_train["Poverty"]
Y = y_train["Unemployment"]
X = sm.add_constant(X)
results = sm.OLS(Y,X).fit()
print(results.summary())
#SSE = round(results.ssr,3)
#SSE

# Checking the model with y_val
X = sm.add_constant(x_val[["Poverty"]])
y_pred = results.predict(X)

from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score
MSE = mean_squared_error(y_val["Unemployment"], y_pred)
MSE
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Unemployment   R-squared:                       0.503
Model:                            OLS   Adj. R-squared:                  0.503
Method:                 Least Squares   F-statistic:                     2604.
Date:                Mon, 02 Dec 2019   Prob (F-statistic):               0.00
Time:                        19:27:09   Log-Likelihood:                -6362.7
No. Observations:                2576   AIC:                         1.273e+04
Df Residuals:                    2574   BIC:                         1.274e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.5757      0.122     21.147      0.000       2.337       2.815
Poverty       21.6637      0.425     51.031      0.000      20.831      22.496
==============================================================================
Omnibus:                      151.689   Durbin-Watson:                   1.967
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              626.420
Skew:                           0.062   Prob(JB):                    9.43e-137
Kurtosis:                       5.413   Cond. No.                         8.02
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl

C:\Users\kalya\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: Futu
ecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)

8.634216106956256
```

The MSE (Mean squared error) on the test dataset of the above model with a single predictor = 8.63

```
# Build model with 2 predictors (using statsmodels)
X = x_train[["Poverty", 'White']]
Y = y_train["Unemployment"]
X = sm.add_constant(X)
results = sm.OLS(Y,X).fit()
print(results.summary())
#SSE = round(results.ssr,3)
#SSE

# Checking the model with y_val
X = sm.add_constant(x_val[["Poverty", 'White']])
y_pred = results.predict(X)

from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score
MSE = mean_squared_error(y_val["Unemployment"], y_pred)
MSE
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Unemployment   R-squared:                       0.519
Model:                            OLS   Adj. R-squared:                  0.519
Method:                 Least Squares   F-statistic:                     1388.
Date:                Mon, 02 Dec 2019   Prob (F-statistic):               0.00
Time:                        19:27:09   Log-Likelihood:                -6320.4
No. Observations:                2576   AIC:                         1.265e+04
Df Residuals:                    2573   BIC:                         1.266e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.4832      0.336     16.315      0.000       4.824       6.142
Poverty       18.6425      0.530     35.172      0.000      17.603      19.682
White         -2.8321      0.306     -9.259      0.000      -3.432      -2.232
==============================================================================
Omnibus:                      157.886   Durbin-Watson:                   1.960
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              680.088
Skew:                          -0.055   Prob(JB):                    2.09e-148
Kurtosis:                       5.515   Cond. No.                         15.0
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl

C:\Users\kalya\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: Futu
ecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)

: 8.349700624803543
```

The MSE (Mean squared error) on the test dataset of the above model with two predictors = 8.4.
So, I kept the second variable and added the remaining variables one by one and kept the variables whose inclusion reduced the Mse value.

vii) I obtained the best model that has 14 predictors out of 40:

```
# Build model with 14 predictors (using statsmodels)
X = x_train[["Poverty", 'White', "Black","Service", "WorkAtHome", "Hispanic", 'Mean Commute Time',
             "Percent Population with age between 16 and 44", 'Asian', 'Office', 'Construction', 'Drive', 'Tr
ansit',
            "State Unemployment rate for 2013", "County Unemployment Rate for 2013"]]
Y = y_train["Unemployment"]
X = sm.add_constant(X)
results = sm.OLS(Y,X).fit()
print(results.summary())
#SSE = round(results.ssr,3)
#SSE

# Checking the model with y_val
X = sm.add_constant(x_val[["Poverty", 'White', "Black", "Service", "WorkAtHome", "Hispanic", 'Mean Commute Ti
me',
                            "Percent Population with age between 16 and 44", 'Asian','Office', 'Construction',
'Drive', 'Transit',
                           "State Unemployment rate for 2013", "County Unemployment Rate for 2013"]])
y_pred = results.predict(X)

from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score
MSE = mean_squared_error(y_val["Unemployment"], y_pred)
MSE
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Unemployment   R-squared:                       0.840
Model:                            OLS   Adj. R-squared:                  0.839
Method:                 Least Squares   F-statistic:                     893.3
Date:                Mon, 02 Dec 2019   Prob (F-statistic):               0.00
Time:                        19:27:10   Log-Likelihood:                -4905.9
No. Observations:                2576   AIC:                             9844.
Df Residuals:                    2560   BIC:                             9937.
Df Model:                          15
Covariance Type:            nonrobust
```

```
3.0408994085812773
```

viii)I applied the above model on the 2017 dataset, predicted the county unemployment rates. I got the mean squared error of 2.26 from the actual unemployment rates.

```
X = sm.add_constant(dataN2017[["Poverty", 'White', "Black", "Service", "WorkAtHome", "Hispanic", 'Mean Commut
e Time',
                           "Percent Population with age between 16 and 44", 'Asian','Office', 'Construction',
'Drive', 'Transit',
                          "State Unemployment rate for 2015", "County Unemployment Rate for 2015"]])
y = Reg.predict(X)
```

```
C:\Users\kalya\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: FutureWarning: Method .ptp is depr
ecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

```
y.head()
```

```
0     6.365575
1     6.106433
2    13.749840
3     7.012512
4     6.714942
dtype: float64
```

```
MSE = mean_squared_error(dataN2017["Unemployment"], y)
MSE
```

```
2.2657266854821754
```

Better Approaches:

a. Instead of manually removing correlated predictor variables, we can use the Variable Inflation factor of statsmodels library of python which gives the VIF score for each of the variables.

   The Variance Inflation Factor (VIF) is a measure of collinearity among predictor variables within a multiple regression. Steps for Implementing VIF:

   1. Run a multiple regression.
   2. Calculate the VIF factors.

3. Inspect the factors for each predictor variable, if the VIF is between 5-10, multi-collinearity is likely to present and you should consider dropping the variable.

   Ex:

	VIF Factor	features
0	5.1	Intercept
1	1.0	dti
2	678.4	funded_amnt
3	678.4	loan_amnt

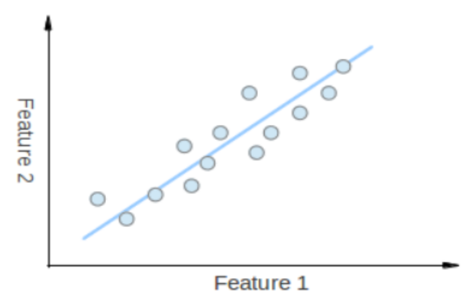
As expected, the total funded amount for the loan and the amount of the loan have a high variance inflation factor because they "explain" the same variance within this dataset. We would need to discard one of these variables before moving on to model building or risk building a model with high multicolinearity.

Ref: https://etav.github.io/python/vif_factor_python.html

b. And instead of using forward selection methods to find the best variables, we can use dimensionality reduction techniques like Principal Component Analysis (PCA). These techniques represent a given dataset in fewer dimensions while retaining the variance of the original dataset.
Principal component analysis (PCA) is a technique for reducing the dimensionality of large datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.

What is PCA?

Principal Component Analysis (PCA) is a statistical procedure that extracts the most important features of a dataset.



Consider that you have a set of 2D points as it is shown in the figure above. Each dimension corresponds to a feature you are interested in. Here some could argue that the points are set in a random order. However, if you have a better look you will see that there is a linear pattern (indicated by the blue line) which is hard to dismiss. A key point of PCA is the Dimensionality Reduction. Dimensionality Reduction is the process of reducing the number of the dimensions of the given dataset. For example, in the above case it is possible to approximate the set of points to a single line and therefore, reduce the dimensionality of the given points from 2D to 1D.

Moreover, you could also see that the points vary the most along the blue line, more than they vary along the Feature 1 or Feature 2 axes. This means that if you know the position of a point along the blue line you have more information about the point than if you only knew where it was on Feature 1 axis or Feature 2 axis.

The blue line above is a new artificial feature found by the PCA technique. That line is good enough to represent all the data points. Here, we converted the 2-dimensional dataset into the 1-dimensional dataset (dimensionality reduction)

Ref: https://towardsdatascience.com/principal-component-analysis-pca-from-scratch-in-python-7f3e2a540c51
https://en.wikipedia.org/wiki/Principal_component_analysis
Video link: https://www.youtube.com/watch?v=FgakZw6K1QQ

Note: For another project of my data science course, I predicted the votes of Republican and Democratic parties (two response variables) for counties of the US in 2018. I built the Linear regression models on the 2018 demographics of the US counties in a similar fashion.