Unemployment rate by County

Teammates:
Dharun Selvan - Data Science
Vallabha Datta Varma Penmetcha - Data Science
Milin Dharmshibhai Desai - Data Science

ABSTRACT

Unemployment rate by county in the United States Unemployment in the United States discusses the causes and measures of U.S. unemployment and strategies for reducing it. Job creation and unemployment are affected by factors such as economic conditions, global competition, education, automation, and demographics. These factors can affect the number of workers, the duration of unemployment, and wage levels.

The objective of our research is to design a predictive algorithm using Python to find the total employment numbers and unemployment numbers of every county in the USA for the next 5 years. the predictive models professionals often use.

Goal -

1. Estimating the total employment numbers and unemployment numbers of every county in the USA for the next 5 years.

2. Using Clustering, finding which state is going to have less unemployment rate in the next 5 years .

Findings

Certain variables within the dataset were found to have a high impact upon unemployment rate. This shows a level of correlation between certain variables in a given observation, allowing us to refine a different model.

Recommendation - given various attributes, the Clustering, regression analysis and Classification model, and the threshold, we could predict with unemployment rate in the next 5 years some confidence.

Our generated model would identify the most significant predictors for unemployment rate.

INTRODUCTION

Unemployment can be measured in several ways. A person is defined as unemployed in the United States if they are jobless but have looked for work in the last four weeks and are available for work.

People who are neither employed nor defined as unemployed are not included in the labor force calculation. For example, as of September 2017, the unemployment rate in the United States was 4.2% or 6.8 million people, while the government's broader U-6 unemployment rate, which includes the part-time underemployed was 8.3%.

Both of these rates were below the November 2007 level that preceded the Great Recession. These figures were calculated with a civilian labor force of approximately 159.6 million people,relative to a U.S. population of approximately 326 million people.The unemployment rates (U-3 and U-6) fell steadily from 2010 to 2018.

Unemployment is one of the big issues in the United States, because without employment youth influence to do unethical activities like drug smuggling, robbery, car snatching and cyber-crime as well. When we talk about Unemployment we define it as person who is fit for job and also willing to do work but because of circumstances unable to find paid employment

METHODOLOGY

Step 1:

- First we will analyze the data set and get to know about the data.
- Do data visualization to check whether data cleaning is necessary or not.

Step 2:

- If necessary, do preprocessing techniques like finding the missing data. and replacing it with the mean, normalizing the data etc.
- Creating Box plots and histograms to analyze the distributions and understanding the relationships among the variables.

Step 3:

- Splitting the data into training dataset and two Test data sets as 80%-10%-10% weightage.
- Finding the correlation between the target variable and other variables .
- Applying various data regression models and finding the best model to estimate the numbers.

Step 4:

- Applying clustering techniques in order to find the numbers for state-wise.

Step 5:

- Testing the codes with test data sets.
- Compare the values of train data output and test data output.

Step 6:

- Do a presentation and video explaining the project.
- Write the project report including codes, outputs, tables, test results etc.
- Create a readme file that has instructions on how to compile/run the program.

ANALYSIS, RESULTS & FINDINGS

Data Exploration:

Data Set link:

https://www.ers.usda.gov/webdocs/DataFiles/48747/Unemployment.xls?v=9115.7

Estimated file size is 1525kb.

There are 56 attributes(Columns) and 3222 observations(Rows) in the data, of which 3 are categorical and rest are numerical attributes. Unemplyment_rate_2018 is our target variable which is numerical. The description of the columns is given below

Variables	Description
FIPS	State_county FIPS Code
State	State Abbreviation
Area_name	State or County name
Rural_urban_continuum_code	Code of rural-urban Continuum
Urban_influence_code	Urban Influence Code
Metro	Metro or Non-metro. It is dummy variable 0 = Non-metro, 1 = Metro

Civilian_labor_force	Civilian labor force annual average (Vary Yearly)
Employed	Annual average of number of people employed (Vary Yearly)
Unemployed	Annual average of number of people unemployed (Vary Yearly)
Unemployment_rate	Unemployment rate (Vary yearly)
Median_Household_Income_2017	Estimation of median household Income, 2017
Med_HH_Income_Percent_of_State_Total_2017	County Household Median Income as a percent of the State Total Median Household Income, 2017

STATISTICS

	Unnamed: 0	Area_name	Rural_urban_continuum_code_2013	Metro_2013	Civilian_labor_force_2007	Unemployment_rate_2007	Civilian_labor_force_2008
count	3214.000000	3214.000000	3214.000000	3214.000000	3.214000e+03	3214.000000	3.214000e+03
mean	1614.286559	1615.286559	4.932172	0.383945	4.778207e+04	5.066957	4.825767e+04
std	928.306880	928.306880	2.721953	0.486421	1.534632e+05	2.124130	1.551947e+05
min	0.000000	1.000000	1.000000	0.000000	4.100000e+01	1.500000	4.300000e+01
25%	811.250000	812.250000	2.000000	0.000000	5.238500e+03	3.700000	5.282500e+03
50%	1614.500000	1615.500000	6.000000	0.000000	1.217400e+04	4.700000	1.223150e+04
75%	2417.750000	2418.750000	7.000000	1.000000	3.145000e+04	5.800000	3.170900e+04
max	3221.000000	3222.000000	9.000000	1.000000	4.864160e+06	20.400000	4.928959e+06

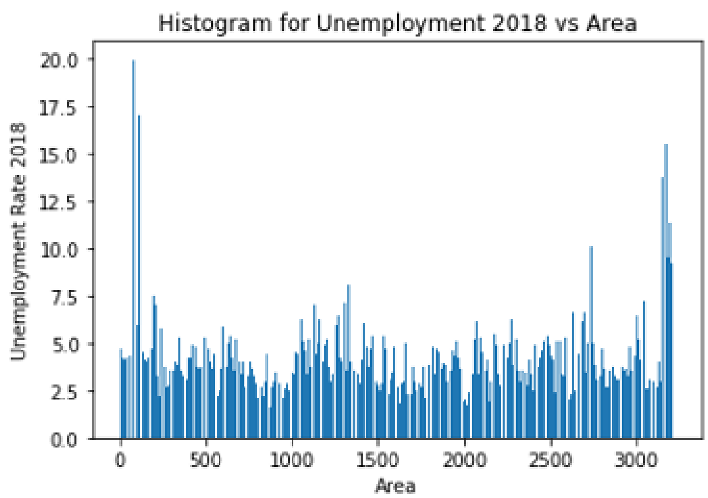
Unemployment_rate_2018	Med_HH_Income_Percent_of_State_Total_2017	State
3214.000000	3214.000000	3214.000000
4.294337	89.104050	27.693528
1.881752	19.818933	14.407254
1.300000	39.900000	1.000000
3.100000	76.225000	16.000000
3.900000	86.700000	27.000000
4.900000	98.400000	42.000000
19.900000	251.400000	52.000000

There are so many columns in the dataset so, we computed the statistics for unique columns after pre-processing the data. From the descriptive statistics we can say that

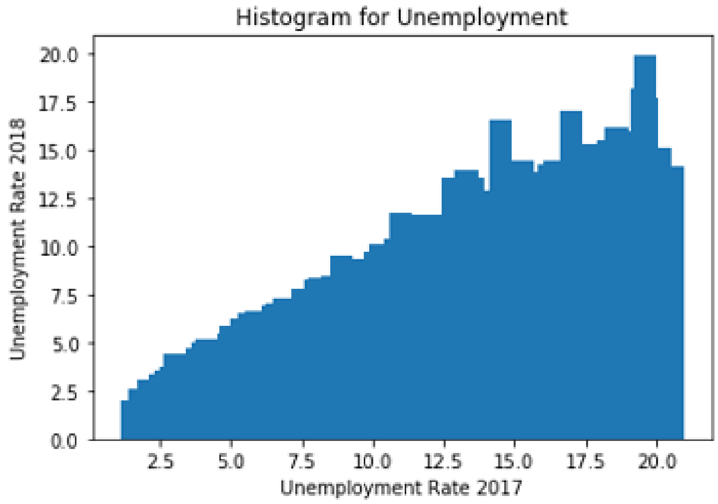
- There are 3222 unique in area_name varying from 1 to 3214 with mean of 1615.
- Metro is binary. If state is metro state then 1, else zero. So max is 1 and min is 0.
- Rural_urban_continuum_code is the rural code of the county with min value of 1 and maximun value of 9
- The min household income to percentage of state is 39% and maximum of 251%. The mean income percentage is 89% for 2017.

Data Visualization

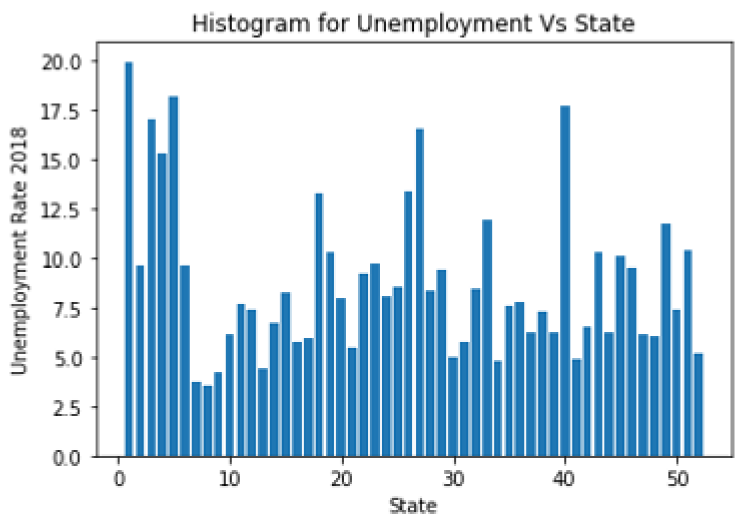
We did visualization only as bivariate since univariate analysis doesn't make any sense with our dataset.



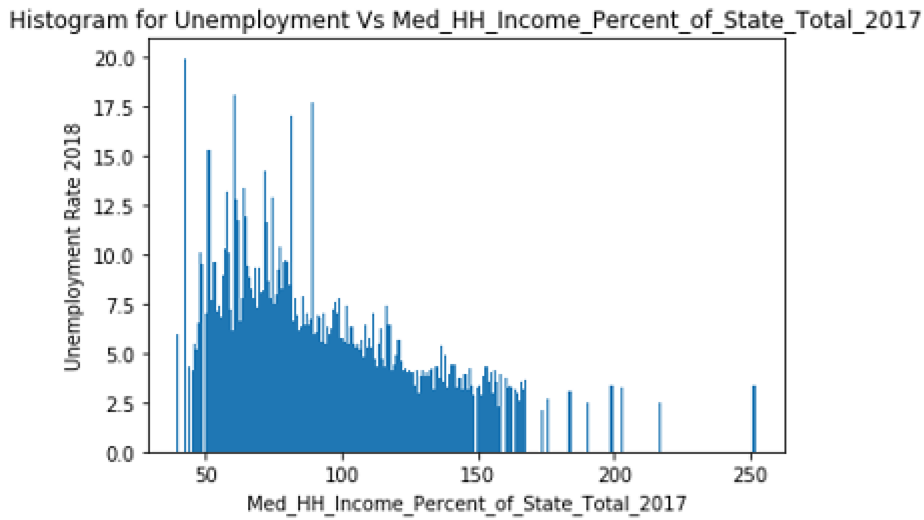
The above histogram shows the unemployment rate in every county. From this we can see that most of the county has less than 7.5 unemployment rate in 2018.



The above histogram shows the unemployment rate 2018 vs 2017 in every county. From this we can see that most of the county's rate decreases from 2017 to 2018.



The above histogram shows the unemployment rate 2018 vs State.



The above histogram shows the unemployment rate 2018 vs Household income in every county. From this we can see that higher the income, lower the unemployment rate.

Data Preprocessing

The shape of employment dataset is (3222,56) i.e, 3222 entries and 56 columns. The columns are shown below

```
Index(['FIPS', 'State', 'Area_name', 'Rural_urban_continuum_code_2013',
       'Urban_influence_code_2013', 'Metro_2013',
       ' Civilian_labor_force_2007 ', ' Employed_2007 ', ' Unemployed_2007 ',
       'Unemployment_rate_2007', ' Civilian_labor_force_2008 ',
       ' Employed_2008 ', ' Unemployed_2008 ', 'Unemployment_rate_2008',
       'Civilian_labor_force_2009', 'Employed_2009', 'Unemployed_2009',
       'Unemployment_rate_2009', ' Civilian_labor_force_2010 ',
       ' Employed_2010 ', ' Unemployed_2010 ', 'Unemployment_rate_2010',
       ' Civilian_labor_force_2011 ', ' Employed_2011 ', ' Unemployed_2011 ',
       'Unemployment_rate_2011', ' Civilian_labor_force_2012 ',
       ' Employed_2012 ', ' Unemployed_2012 ', 'Unemployment_rate_2012',
       ' Civilian_labor_force_2013 ', ' Employed_2013 ', ' Unemployed_2013 ',
       'Unemployment_rate_2013', ' Civilian_labor_force_2014 ',
       ' Employed_2014 ', ' Unemployed_2014 ', 'Unemployment_rate_2014',
       ' Civilian_labor_force_2015 ', ' Employed_2015 ', ' Unemployed_2015 ',
       'Unemployment_rate_2015', ' Civilian_labor_force_2016 ',
       ' Employed_2016 ', ' Unemployed_2016 ', 'Unemployment_rate_2016',
       'Civilian_labor_force_2017', 'Employed_2017', 'Unemployed_2017',
       'Unemployment_rate_2017', 'Civilian_labor_force_2018', 'Employed_2018',
       'Unemployed_2018', 'Unemployment_rate_2018',
       'Median_Household_Income_2017',
       'Med_HH_Income_Percent_of_State_Total_2017'],
      dtype='object')
```

The original data set has variable description along with the data in the same file. Hence, our first is to delete the variable description and extract only the data in .xlsx format.

Checked for the data type format and found that all the columns are in correct format. Two columns are type ‘Object’, two have type ‘int’ and all other columns are type ‘float’.

After extracting, there are 52 rows with a number of total labour force, Employment and Unemployment details per state. Since, we are doing county wise prediction these rows are not useful hence deleted the rows in excel that have state wise numbers.

There are five rows with almost 90% missing values. Since data will not be healthy if we replace all the missing values in those rows, we dropped them.

After dropping five rows, checked for null values and found that there are 3 null values for all the columns from 2007 to 2009. This is because there are three rows with null values, so we deleted those three rows.

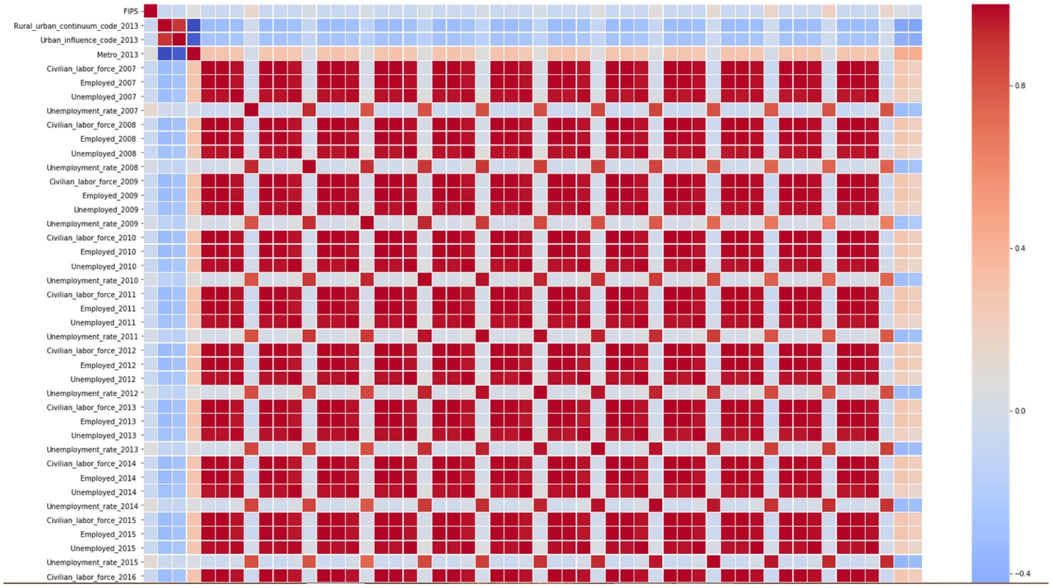
```
 Civilian_labor_force_2007                    3
 Employed_2007                                3
 Unemployed_2007                              3
Unemployment_rate_2007                        3
 Civilian_labor_force_2008                    3
 Employed_2008                                3
 Unemployed_2008                              3
Unemployment_rate_2008                        3
Civilian_labor_force_2009                     3
Employed_2009                                 3
Unemployed_2009                               3
Unemployment_rate_2009                        3
```

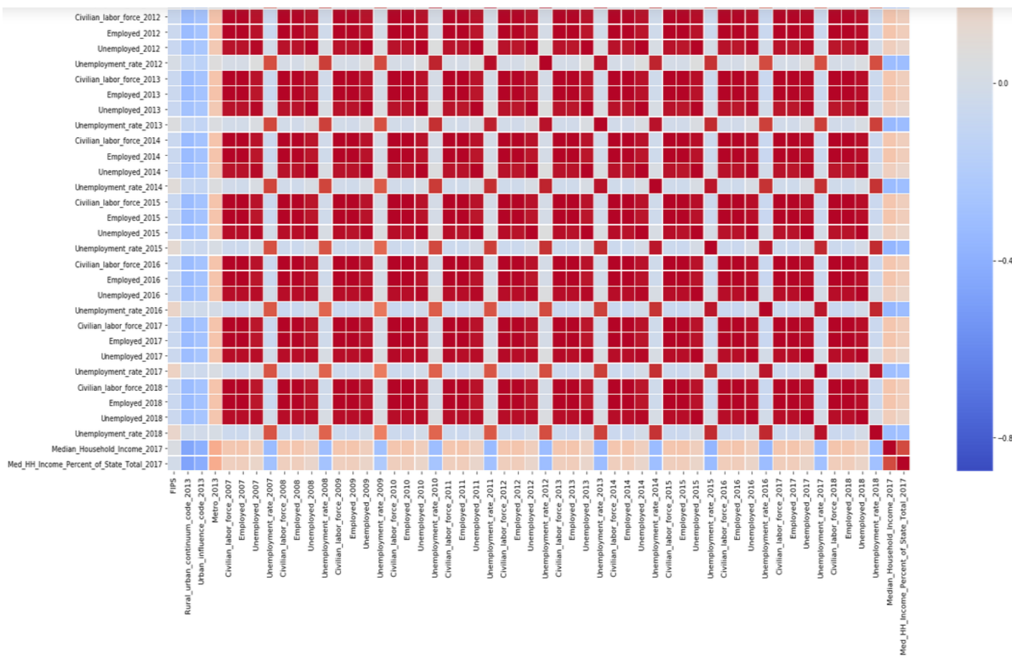
After deleting the above mentioned rows shape of the data set is (3214, 56) . Then we checked for missing values and found that Median_Household_Income_2017 and Med_HH_Income_Percent_of_State_Total_2017 columns have 78 missing values. We replaced them with the mean of the column.

```
Unemployed_2018                               0
Unemployment_rate_2018                        0
Median_Household_Income_2017                 78
Med_HH_Income_Percent_of_State_Total_2017    78
dtype: int64
```

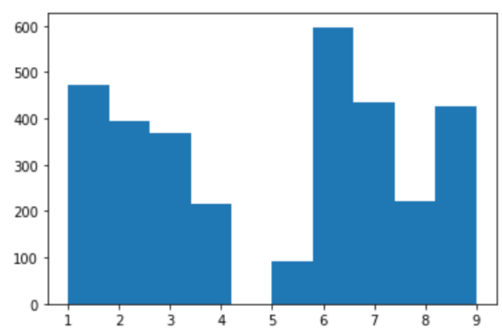
The Median_Household_Income_2017 column has the ‘$’ symbol in the income values so we deleted that symbol in the excel file.

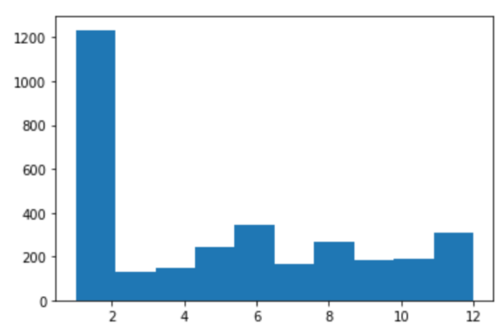
Next we checked for correlation values for all the variables. There is high correlation for employment, unemployment, rates and civilian labour force for all years.





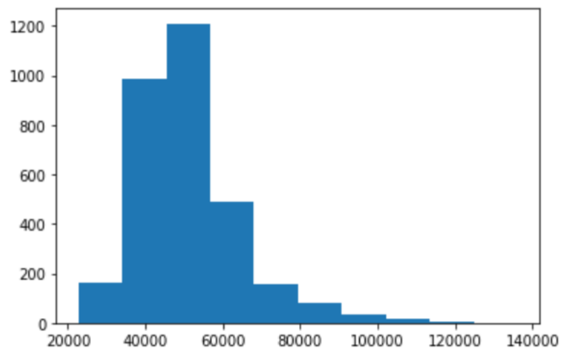
There is also high correlation between Rural_urban_continuum_code_2013 and Urban_influence_code_2013. So, we should delete one of them. We checked the histograms for that.

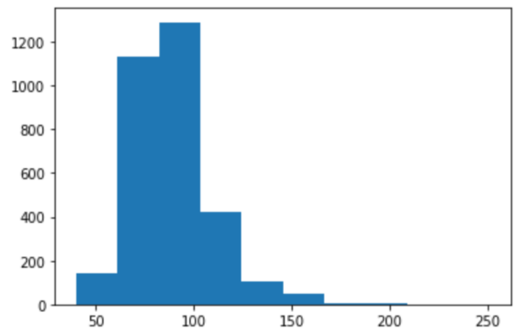




Rural_urban_continuum_code_2013 Urban_influence_code_2013

From the above histograms Urban_influence_code_2013 is highly skewed so we dropped that column. Same way we should drop one column from Median_Household_Income_2017 and Med_HH_Income_Percent_of_State_Total_2017. They have almost the same histogram. It's better to take the percentage of income so we dropped the Median_Household_Income_2017 column.





Median_Household_Income_2017 Med_HH_Income_Percent_of_State_Total_2017

The aim of the project is to predict the employment rate so the number of people unemployed columns for all years are not useful and drop them. Number employed and employment rate columns are also not useful because we already have Civilian_labour_force. So, dropped the number employment and employment rate for all years.

The FIPS column is also not useful and dropped that one too. In the dataset area column (county name for every state) and state columns are categorical and we changed them into numerical. We used state.cat.codes to change the state column. It will give the value in alphabetical order of state. There are 3222 county's and 52 states. After changing into a numerical area it has 3222 values and the state has 52 unique values.

State	Area_name	Rural_urban_continuum_code_2013	Urban_influence_code_2013	Metro_2013
AL	Autauga County, AL	2.0	2.0	1
AL	Baldwin County, AL	3.0	2.0	1
AL	Barbour County, AL	6.0	6.0	0
AL	Bibb County, AL	1.0	1.0	1
AL	Blount County, AL	1.0	1.0	1

	Area_name	Rural_urban_continuum_code_2013	Metro_2013
0	1	2.0	1
1	2	3.0	1
2	3	6.0	0
3	4	1.0	1
4	5	1.0	1
5	6	6.0	0

Med_HH_Income_Percent_of_State_Total_2017	State
121.1	2.0
117.5	2.0
67.4	2.0
95.0	2.0
100.1	2.0

After doing the pre-processing the final shape of the data set is (3214, 29). Total of 3214 entries and 29 columns. Finally we exported the preprocessed data and used it to perform Regression, Classification and Clustering techniques.

CLASSIFICATION:

After data pre-processing, we exported the pre processed data in .xlsx format and we are using that to perform different classification techniques. Firstly, imported data using pandas. Output of first five rows are shown below:

	Unnamed: 0	Area_name	Rural_urban_continuum_code_2013	Metro_2013	Civilian_labor_force_2007	Unemployment_rate_2007	Civilian_labor_force_2008	Uner
0	0	1	2	1	24383	3.3	24687	
1	1	2	3	1	82659	3.1	83223	
2	2	3	6	0	10334	6.3	10161	
3	3	4	1	1	8791	4.1	8749	
4	4	5	1	1	26629	3.2	26698	

Data contains 3214 entries and 30 columns. While exporting the data in excel, an Unnamed column is created with row numbers as values. So, this column should be dropped. Next we checked whether any of the columns contain missing or null values and data type error.

```
Data columns (total 29 columns):
Area_name                                 3214 non-null int64
Rural_urban_continuum_code_2013           3214 non-null int64
Metro_2013                                3214 non-null int64
 Civilian_labor_force_2007                3214 non-null int64
Unemployment_rate_2007                    3214 non-null float64
 Civilian_labor_force_2008                3214 non-null int64
Unemployment_rate_2008                    3214 non-null float64
Civilian_labor_force_2009                 3214 non-null int64
Unemployment_rate_2009                    3214 non-null float64
 Civilian_labor_force_2010                3214 non-null int64
Unemployment_rate_2010                    3214 non-null float64
 Civilian_labor_force_2011                3214 non-null int64
Unemployment_rate_2011                    3214 non-null float64
 Civilian_labor_force_2012                3214 non-null int64
Unemployment_rate_2012                    3214 non-null float64
 Civilian_labor_force_2013                3214 non-null int64
Unemployment_rate_2013                    3214 non-null float64
 Civilian_labor_force_2014                3214 non-null int64
Unemployment_rate_2014                    3214 non-null float64
 Civilian_labor_force_2015                3214 non-null int64
Unemployment_rate_2015                    3214 non-null float64
 Civilian_labor_force_2016                3214 non-null int64
Unemployment_rate_2016                    3214 non-null float64
Civilian_labor_force_2017                 3214 non-null int64
Unemployment_rate_2017                    3214 non-null float64
Civilian_labor_force_2018                 3214 non-null int64
Unemployment_rate_2018                    3214 non-null float64
Med_HH_Income_Percent_of_State_Total_2017 3214 non-null float64
State                                     3206 non-null float64
dtypes: float64(14), int64(15)
memory usage: 728.2 KB
```

All the data types of columns are correct so, no need to change them. In the “State” column we can see that there are 3206 non-null, from this we can infer that there are 8 missing values in the “State” column. This column is numeric since we have replaced the states with cat codes. Now, replacing the null values with the mean of the column. We got mean as 27, where 27 represents the state “Montana”.

Selecting “Unemployment_rate_2018” as target variable. The “Unemployment_rate_2018” has continuous values so we are changing it to categorical with 5 categories (‘Very_low’, ‘low’, ‘Average’, ‘High’, ‘Very_high’) to perform classification techniques.

- If Unemployment_rate is less than 3.1, then it is very low
- If Unemployment_rate is between 3.1 & 3.9[including 3.9] then it is low
- If Unemployment_rate is between 3.9 & 4.9[including 4.9] then it is Average

- If Unemployment_rate is between 4.9 & 10[including 10] then it is High
- If Unemployment_rate is greater than 10, then it is very high

After changing “Unemployment_rate_2018” into categorical, the head of “Unemployment_rate_2018” is shown below:

```
Out[191]: ['Low',
           'Low',
           'High',
           'Average',
           'Low',
           'Average',
           'Average',
           'Average',
           'Low',
           'Low',
           'Low',
           'High',
           'High',
           'Low',
           'Average',
           'Low',
           'Average',
           'High',
           'Average',
```

Unemployment_rate_2018 is our target variable, so separating the target variable from the data set. Performed min-max scaling to the dataset and then splitting the scaled data into 80% training and 20% testing.

Training set consists of 2571 entries and 28 columns. Testing set consists of 643 entries and 28 columns.

Training set:

Out[204]:

	Area_name	Rural_urban_continuum_code_2013	Metro_2013	Civilian_labor_force_2007	Unemployment_rate_2007	Civilian_labor_force_2008	Unemployme
599	0.188451	0.625	0.0	0.003692	0.185185	0.003632	
809	0.253648	1.000	0.0	0.000854	0.137566	0.000858	
1785	0.556659	0.000	1.0	0.036842	0.084656	0.036833	
725	0.227569	0.625	0.0	0.004108	0.169312	0.004085	
2935	0.913691	0.000	1.0	0.001292	0.042328	0.001290	

Testing set:

Out[205]:

	Area_name	Rural_urban_continuum_code_2013	Metro_2013	Civilian_labor_force_2007	Unemployment_rate_2007	Civilian_labor_force_2008	Unemployme
2947	0.917417	0.625	0.0	0.001641	0.269841	0.001638	
2335	0.727414	0.125	1.0	0.026773	0.185185	0.026511	
511	0.161130	0.875	0.0	0.000641	0.259259	0.000639	
2503	0.779572	0.125	1.0	0.015212	0.132275	0.015133	
1794	0.559454	0.500	0.0	0.004301	0.068783	0.004255	

I. KNN CLASSIFIER

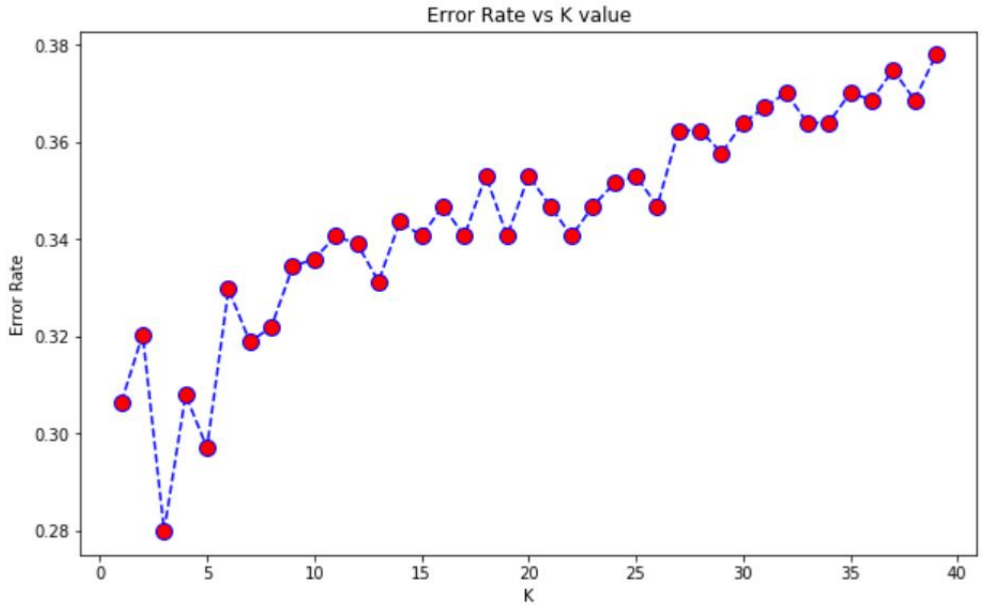
Fitted training and target set using kNeighborsClassifier from sklearn.neighbors with five nearest neighbors. Then performed the prediction with the test set and printed confusion matrix and classification report using sklearn.metrics.

Confusion matrix and Classification report:

```
[[118  16  26   0   0]
 [ 32 100   2   3   0]
 [ 41   2 101   0  24]
 [  0   3   0  11   0]
 [  4   0  38   0 122]]
              precision    recall  f1-score   support

     Average       0.61      0.74      0.66       160
        High       0.83      0.73      0.78       137
         Low       0.60      0.60      0.60       168
   Very High       0.79      0.79      0.79        14
    Very low       0.84      0.74      0.79       164

    accuracy                           0.70       643
   macro avg       0.73      0.72      0.72       643
weighted avg       0.71      0.70      0.71       643
```

From the above table accuracy obtained with n_neighbors=5 is 70%

To improve the accuracy we should the best value of k (number of k-nearest neighbors). So, plotted the variation of error rates with respect to 1 to 40 ‘k’ values.



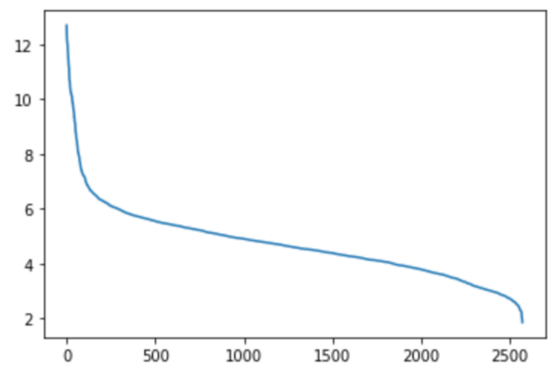
From the above plot, the error rate is less for k=3. So, again fitted training and target set with 3 k-nearest neighbors and computed classification report and confusion matrix.

```
[[118  18  24   0   0]
 [ 26 108   1   2   0]
 [ 43   1 101   0  23]
 [  0   3   0  11   0]
 [  7   0  32   0 125]]
              precision    recall  f1-score   support

     Average       0.61      0.74      0.67       160
        High       0.83      0.79      0.81       137
         Low       0.64      0.60      0.62       168
   Very High       0.85      0.79      0.81        14
    Very low       0.84      0.76      0.80       164

    accuracy                           0.72       643
   macro avg       0.75      0.73      0.74       643
weighted avg       0.73      0.72      0.72       643
```

We can see that by performing prediction with 3 nearest neighbors, the accuracy was improved by 2% i.e., 72%

II) TERM-DOCUMENT CATEGORIZATION

Scaled training and testing sets using term- document categorization. The term frequencies of the training set is shown below:



As the number of observations increases, frequency is decreased. Performed TD*IDF for both training and testing set. The outputs are shown below

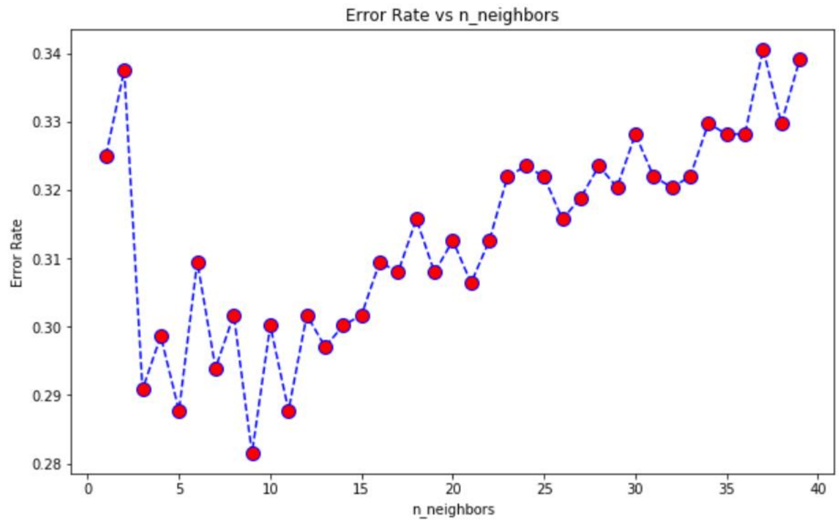
TD*IDF training:

	Area_name	Rural_urban_continuum_code_2013	Metro_2013	Civilian_labor_force_2007	Unemployment_rate_2007	Civilian_labor_force_2008	Unemployme
599	0.009888	0.032792	0.000000	0.000194	0.009716	0.000191	
809	0.013308	0.052467	0.000000	0.000045	0.007218	0.000045	
1785	0.029206	0.000000	0.052467	0.001933	0.004442	0.001933	
725	0.011940	0.032792	0.000000	0.000216	0.008883	0.000214	
2935	0.047939	0.000000	0.052467	0.000068	0.002221	0.000068	

TD*IDF testing:

	Area_name	Rural_urban_continuum_code_2013	Metro_2013	Civilian_labor_force_2007	Unemployment_rate_2007	Civilian_labor_force_2008	Unemployme
2947	0.048135	0.032792	0.0	0.000086	0.014158	0.000086	
2335	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	
511	0.008454	0.045909	0.0	0.000034	0.013603	0.000034	
2503	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	
1794	0.029353	0.026234	0.0	0.000226	0.003609	0.000223	

After transforming the training and testing set using term-document categorization, we plotted the variation of error rates with different ‘k’ values (1 to 40) to find the best k-value. From the below error vs k plot, the error value is less when k=10. So, performed knn classification using td*idf training set and then performed prediction using td*idf test set. The classification report and confusion matrix are shown below.




```
[[121   7  33   0   0]
 [ 60  83   2   1   0]
 [ 64   2  72   0  21]
 [  5   1   0   7   0]
 [ 39   0  16   0 109]]
              precision    recall  f1-score   support

     Average       0.42      0.75      0.54       161
        High       0.89      0.57      0.69       146
         Low       0.59      0.45      0.51       159
   Very High       0.88      0.54      0.67        13
    Very low       0.84      0.66      0.74       164

    accuracy                           0.61       643
   macro avg       0.72      0.60      0.63       643
weighted avg       0.68      0.61      0.62       643
```

The maximum accuracy achieved using term-document categorization is only 61% which is less than the previous one.

III) DECISION TREE CLASSIFIER

Fitted training and target set using DecisionTreeClassifier from sklearn.tree. First we used default values for the decision tree classifier.

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False,
            random_state=None, splitter='best')
```

Then performed the prediction with the test set and printed confusion matrix and classification report using sklearn.metrics.

	precision	recall	f1-score	support
Average	0.60	0.67	0.64	161
High	0.82	0.73	0.78	146
Low	0.64	0.62	0.63	159
Very High	0.63	0.92	0.75	13
Very low	0.84	0.84	0.84	164
accuracy			0.72	643
macro avg	0.71	0.76	0.73	643
weighted avg	0.72	0.72	0.72	643

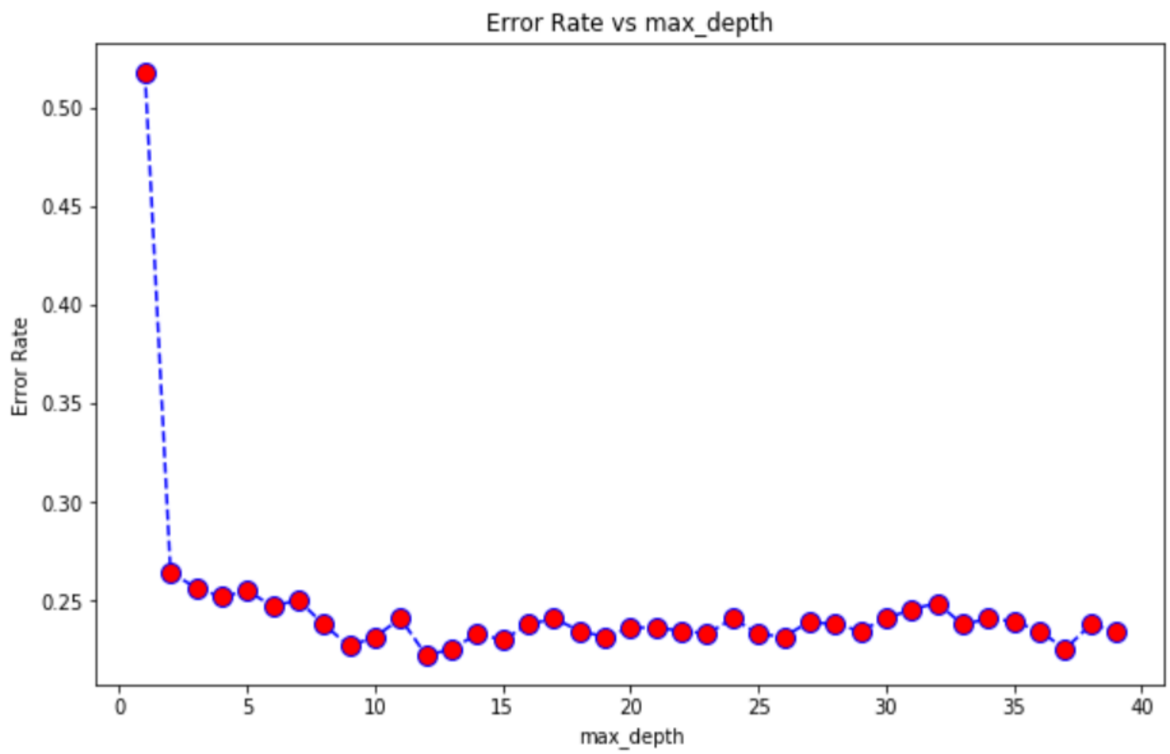
```
In [271]: print(dtree.score(df_test, df_target_test))
0.7107309486780715

In [272]: print(dtree.score(df_train, df_target_train))
1.0
```

Accuracy achieved by using default values is 72%. The tree score for test data is 0.71 and for train set it is 1.0. From this we can infer that the tree performed well for the training set than the testing set. So, the decision tree model is too simple and underfitting.

Next, we fitted the tree and performed prediction while selecting criterion = ‘entropy’. The classification report is shown below

	precision	recall	f1-score	support
Average	0.69	0.72	0.71	161
High	0.83	0.82	0.83	146
Low	0.69	0.64	0.66	159
Very High	0.61	0.85	0.71	13
Very low	0.85	0.87	0.86	164
accuracy			0.76	643
macro avg	0.74	0.78	0.75	643
weighted avg	0.76	0.76	0.76	643

Accuracy achieved is 76%. As the accuracy is increased when criterion = ‘entropy’ we fixed that criterion and then plotted the variation in error values with different values of max_depth (1 to 40).



From the above Error Rate vs max_depth plot, the error rate is less when max_depth is 13. So, while fitting the model using the decision tree classifier we are using criterion = ‘entropy’ and max_depth = ‘13’. Then performed prediction using the testing set.

Classification report:

	precision	recall	f1-score	support
Average	0.72	0.73	0.73	161
High	0.85	0.83	0.84	146
Low	0.71	0.68	0.69	159
Very High	0.61	0.85	0.71	13
Very low	0.86	0.87	0.86	164
accuracy			0.78	643
macro avg	0.75	0.79	0.77	643
weighted avg	0.78	0.78	0.78	643

[1]

The best accuracy achieved using Decision Tree Classifier is 78% which is better than knn and term-document categorized knn.

[1] The picture of the decision tree is too big to fit in this document, so we are attaching that image while submission.

IV) NAIVE BAYES CLASSIFIER

Fitted training and target set using naive_bayes.GaussianNB from sklearn. Then performed the prediction with the test set. The first twenty values of the prediction matrix is shown below.

array(['Average', 'High', 'Low', 'Very low', 'High', 'High', 'Very low', 'Low', 'High', 'Very low', 'Very low',

'Very low', 'High', 'High', 'High', 'Very low', 'Low', 'High', 'High'], dtype='<U9')

Nb_model score for the test set is 0.54 and for the training set it is 0.56. From these values we infer that the model performed moderately for both training and testing sets.

```
In [282]:  print(nb_model.score(df_test, df_target_test))

           0.5443234836702955

In [283]:  print(nb_model.score(df_train, df_target_train))

           0.5593154414624659
```

Confusion matrix and Classification report of the model is shown below.

Confusion Matrix:

```
[[ 12 100  44   0   5]
 [  6 122   5  13   0]
 [ 10  28  72   0  49]
 [  0   0   0  13   0]
 [  3   3  27   0 131]]
```

Classification report:

```
              precision    recall  f1-score   support

     Average       0.39      0.07      0.12       161
        High       0.48      0.84      0.61       146
         Low       0.49      0.45      0.47       159
   Very High       0.50      1.00      0.67        13
    Very low       0.71      0.80      0.75       164

    accuracy                           0.54       643
   macro avg       0.51      0.63      0.52       643
weighted avg       0.52      0.54      0.49       643
```

Accuracy obtained by using Naive Bayes Classifier is just 54% which is too low and is less than all other classifiers.

V) RANDOM FOREST CLASSIFIER

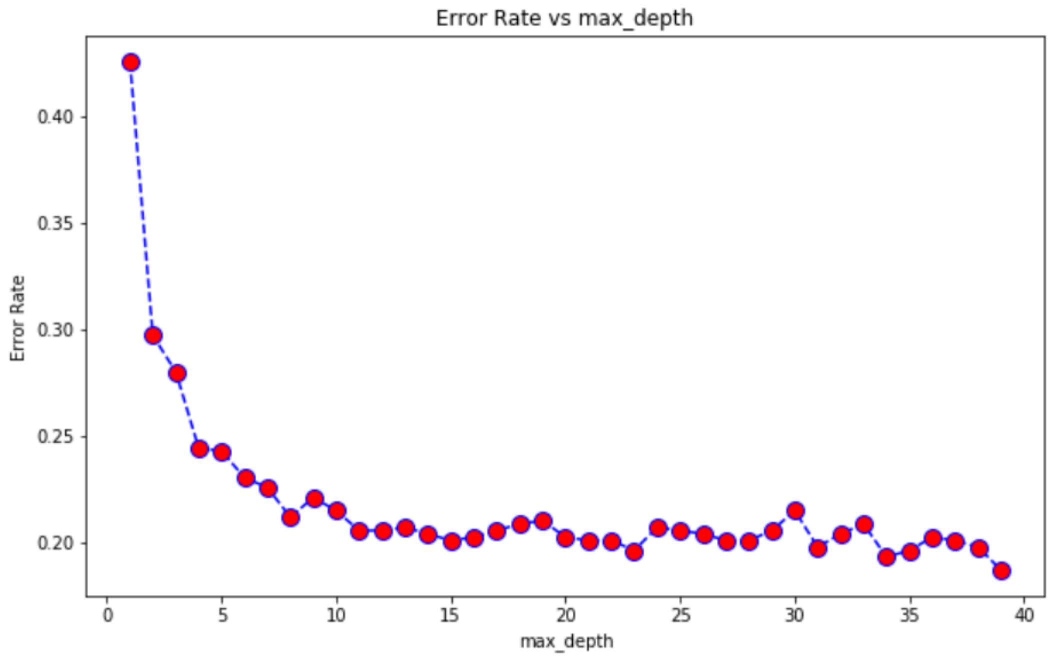
Fitted training and target set using RandomForestClassifier from sklearn.ensemble. First we used default values for decision tree classifier. Then, we used entropy and 100 estimators.

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100,
            n_jobs=None, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

Then performed the prediction with the test set and printed confusion matrix and printed accuracy using sklearn.metrics.

```
In [287]:  print(confusion_matrix(df_target_test,rfc_pred))

[[122  14  25   0   0]
 [ 21 124   0   1   0]
 [ 23   0 114   0  22]
 [  0   1   0  12   0]
 [  1   0  29   0 134]]

In [288]:  # Model Accuracy, how often is the classifier correct?
           print("Accuracy:",metrics.accuracy_score(df_target_test, rfc_pred))

Accuracy: 0.7869362363919129
```

Accuracy achieved using criterion = 'entropy' and n_estimators = 100 is 78.7%. Then we plotted the error values with different max_depth(1,40) values to find the best value of max_depth. The plot of error values vs max_depth is shown below



From the above plot, the error value is less when max_depth is 40. So, using max_depth as 40 while fitting the model with RandomForestClassifier. The metrics for the classifier are:

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
max_depth=40, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
n_jobs=None, oob_score=False, random_state=None,
verbose=0, warm_start=False)

After fitting the model we performed prediction using the test set. The confusion matrix and accuracy are shown below

```
In [351]:  print(confusion_matrix(df_target_test,rfc_pred))
           print("Accuracy:",metrics.accuracy_score(df_target_test, rfc_pred))

[[127  13  21   0   0]
 [ 19 125   0   2   0]
 [ 23   0 118   0  18]
 [  0   1   0  12   0]
 [  2   0  25   0 137]]
Accuracy: 0.807153965785381
```

Accuracy achieved using Random Forest Classifier is 80.7% which is better than all other classifier techniques. So, let's perform prediction using random forest.

CLUSTERING:

After Classification, We are performing clustering. With the help of clustering we can divide population or data in the different groups and data in each group is more likely to be the same with other members of the group. In short, segregate data with similar characteristics and traits and allocate them into clusters.

There are different type of clusterings like soft and hard clustering which includes different types of clusterings algorithms e.g. Knn clustering, hierarchical clustering and there are also some supervised algorithms.

Here, we are using K-means clustering and we divide the whole data in five clusters following, Very low, Low, Average, High, Very High. Where Unemployment_rate is less than 3.1,then it is very low, if Unemployment_rate is between 3.1 & 3.9[including 3.9] then it is low, if Unemployment_rate is between 3.9 & 4.9[including 4.9] then it is Average. If Unemployment_rate is between 4.9 & 10[including 10] then it is High. If Unemployment_rate is greater than 10, then it is very high.

Let's understand everything better with the few results, findings and graphs from our model :

```
0.0:   806
1.0:   836
2.0:   799
3.0:   705
4.0:   68
```

```
size = cluster_sizes(kmeans.labels_)
size
```

```
{0: 458, 1: 751, 2: 820, 3: 481, 4: 61}
```

- Here we have selected the size of cluster is 5. We are splitting data in tests and training.
- You can also find the size of individual clusters as below. We have also performed Silhouette analysis on the clusters that also you can find below.

```
centroids = pd.DataFrame(kmeans_pca.cluster_centers_, columns=df_train.columns)
centroids
```

	Area_name	Rural_urban_continuum_code_2013	Metro_2013	Civilian_labor_force_2007	Unemployment_rate_2007	Civilian_labor_force_2008	Unemployment_ra
0	0.270478	0.722398	-5.329071e-15	0.002278	0.213406	0.002250	
1	0.753787	0.111707	1.000000e+00	0.020987	0.156144	0.020998	
2	0.730927	0.732694	-4.996004e-15	0.002324	0.157800	0.002306	
3	0.929002	0.179276	8.289474e-01	0.002907	0.628028	0.002842	
4	0.231845	0.119685	1.000000e+00	0.024771	0.170500	0.024665	

```
kmeans.cluster_centers_
```

```
array([[ 2.82e-01,  7.37e-01, -8.88e-16,  2.11e-03,  2.14e-01,  2.09e-03,  2.46e-01,  2.09e-03,
         3.00e-01,  2.02e-03,  2.98e-01,  2.01e-03,  2.93e-01,  1.99e-03,  2.82e-01,  1.95e-03,
         2.62e-01,  1.93e-03,  2.27e-01,  1.92e-03,  1.87e-01,  1.90e-03,  1.75e-01,  1.88e-03,
         1.80e-01,  1.87e-03,  1.98e-01,  2.99e-01],
       [ 2.24e-01,  1.37e-01,  1.00e+00,  3.37e-02,  1.67e-01,  3.37e-02,  2.18e-01,  3.36e-02,
         2.84e-01,  3.35e-02,  2.87e-01,  3.36e-02,  2.84e-01,  3.38e-02,  2.71e-01,  3.36e-02,
         2.42e-01,  3.36e-02,  2.09e-01,  3.38e-02,  1.66e-01,  3.39e-02,  1.52e-01,  3.39e-02,
         1.55e-01,  3.40e-02,  2.82e-01,  2.57e-01],
       [ 7.45e-01,  7.37e-01, -6.66e-16,  2.26e-03,  1.64e-01,  2.24e-03,  1.87e-01,  2.26e-03,
         2.44e-01,  2.19e-03,  2.39e-01,  2.18e-03,  2.35e-01,  2.18e-03,  2.28e-01,  2.14e-03,
         2.09e-01,  2.11e-03,  1.76e-01,  2.09e-03,  1.51e-01,  2.06e-03,  1.51e-01,  2.04e-03,
         1.53e-01,  2.03e-03,  2.03e-01,  7.79e-01],
       [ 9.90e-01,  1.70e-01,  8.57e-01,  2.51e-03,  5.93e-01,  2.42e-03,  5.67e-01,  2.36e-03,
         6.16e-01,  2.22e-03,  6.50e-01,  2.13e-03,  6.39e-01,  2.08e-03,  6.28e-01,  2.00e-03,
         6.62e-01,  1.94e-03,  6.59e-01,  1.90e-03,  5.58e-01,  1.88e-03,  5.60e-01,  1.82e-03,
         6.57e-01,  1.78e-03,  2.33e-01,  7.28e-01],
       [ 7.71e-01,  1.03e-01,  1.00e+00,  2.44e-02,  1.61e-01,  2.45e-02,  1.93e-01,  2.46e-02,
         2.63e-01,  2.51e-02,  2.58e-01,  2.52e-02,  2.52e-01,  2.54e-02,  2.45e-01,  2.53e-02,
         2.21e-01,  2.53e-02,  1.86e-01,  2.56e-02,  1.51e-01,  2.58e-02,  1.46e-01,  2.58e-02,
         1.56e-01,  2.59e-02,  2.74e-01,  8.03e-01]])
```

```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(TT2_new,kmeans.labels_))
print(classification_report(TT2_new,kmeans.labels_))
```

```
[[46 24 58  0 36]
 [38 37 51  0 33]
 [56 27 44  0 34]
 [59 19 42  4 22]
 [ 3  0  0 10  0]]
              precision    recall  f1-score   support

         0.0       0.23      0.28      0.25       164
         1.0       0.35      0.23      0.28       159
         2.0       0.23      0.27      0.25       161
         3.0       0.29      0.03      0.05       146
         4.0       0.00      0.00      0.00        13

    accuracy                           0.20       643
   macro avg       0.22      0.16      0.17       643
weighted avg       0.26      0.20      0.21       643
```

We can see that clasification report is almost same for both training and test data

```
size2 = cluster_sizes(clusters)

for c in size2.keys():
    print("Size of Cluster", c, "= ", size2[c])
```

```
Size of Cluster 0 =  1028
Size of Cluster 1 =  583
Size of Cluster 2 =  939
Size of Cluster 3 =  76
Size of Cluster 4 =  588
```

Performing Silhouette analysis on the clusters

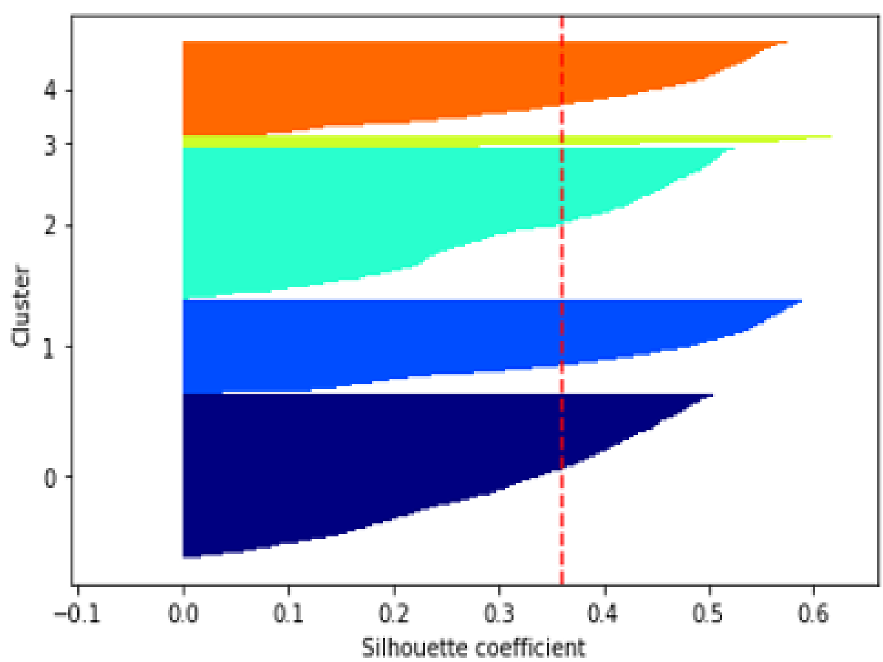
```
silhouettes = metrics.silhouette_samples(df_scaled, clusters)
silhouettes
```

```
array([0.55, 0.54, 0.46, ..., 0.58, 0.58, 0.58])
```

```
print(silhouettes.mean())
```

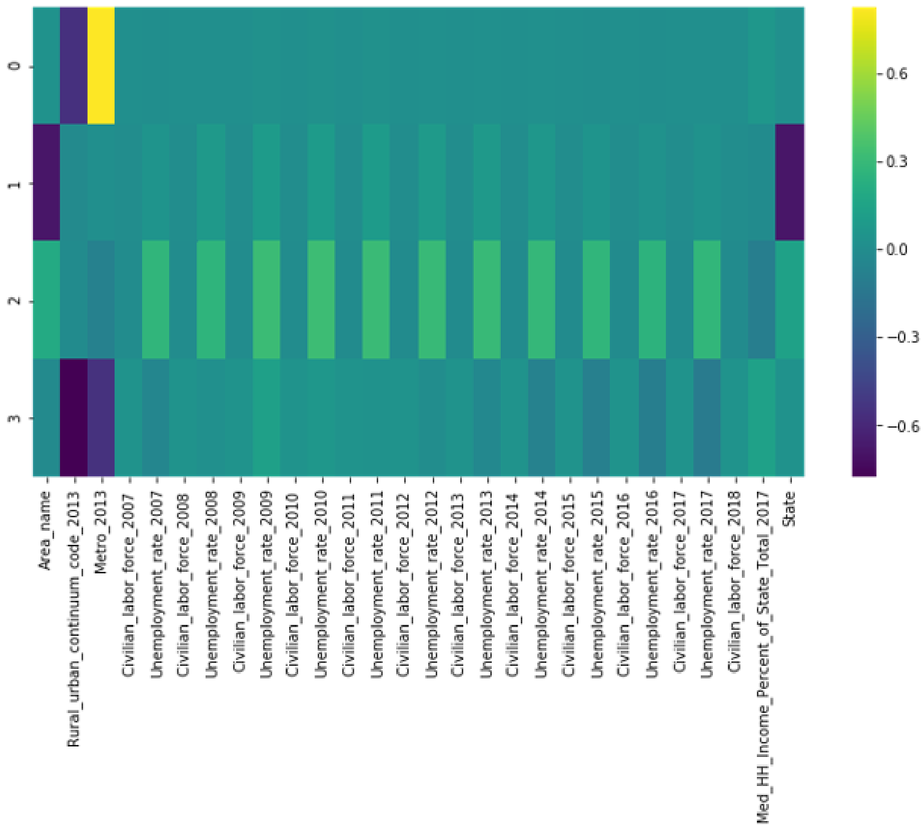
```
0.36047538573358207
```

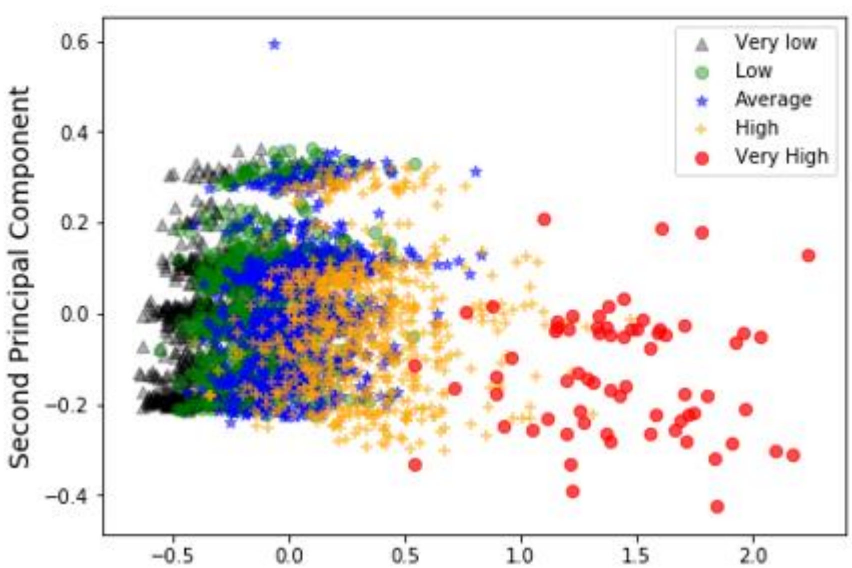
- We are using decomposition so that we can improve accuracy and also we are not splitting data in the PCA. Here in the below graph you can see that we have plotted mean of different clusters with respect to silhouettes co-efficient.



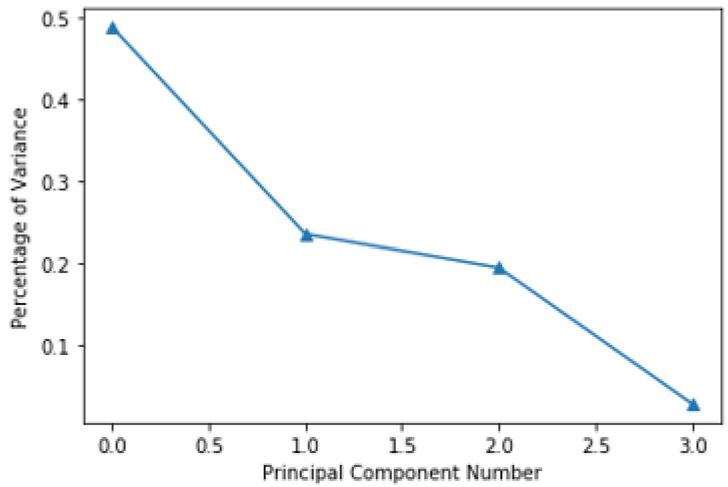
● You can find heat-map for the unemployment rate from 2007 to 2018.

<matplotlib.axes._subplots.AxesSubplot at 0xdc43c10>





- You can see clusters of employment rate Very low, Low, Average, High and Very High. There are four principal components but here third and fourth capture almost 97 percent of the data so we are considering that only.



- Graph for Percentage of Variance and Principle Component Number you can see a downfall in the graph.
- Also the mean of silhouettes is almost 0.4 for dimensional data and also you can see that accuracy is lower in clustering so definitely clustering is not a good option for our data set.

```
silhouettes = metrics.silhouette_samples(DTtrans, clusters)
print(silhouettes.mean())
```

```
0.4013534204637816
```

```
print(confusion_matrix(TT3_new,clusters))
print(classification_report(TT3_new,clusters))
```

```
[[229 134 302 141   0]
 [202 197 234 203   0]
 [266 159 222 151   1]
 [320  98 179  88  20]
 [  9   0   3   0  56]]
              precision    recall  f1-score   support

         0.0       0.22      0.28      0.25       806
         1.0       0.34      0.24      0.28       836
         2.0       0.24      0.28      0.26       799
         3.0       0.15      0.12      0.14       705
         4.0       0.73      0.82      0.77        68

    accuracy                           0.25      3214
   macro avg       0.33      0.35      0.34      3214
weighted avg       0.25      0.25      0.24      3214
```

Accuracy and silhouettes mean increased by using lower dimensional data

From the above reports we can conclude that, clustering is not a good option for predicting for our data set

- However, You can see that after using decomposition and PCA techniques accuracy changed eventually but this accuracy is also not good for the prediction so we are not planning to move forward with the Clustering techniques.

REGRESSION:

After Clustering, With the same preprocessed data we did different types of regression such as Linear Regression, Ridge Regression, Lasso Regression etc. We used the SciKit Learn (sklearn) library to do the regression process in Jupyter notebook. Using Pandas, the preprocessed data was imported as shown below.

Then with the help of data exploration, we dropped the unnecessary columns from the data set using the .drop option and saved the new table in the variable name ‘**new_data**'. The columns we deleted from the preprocessed data set is:

1. Unnamed: 0
2. Civilian_labor_force_2007
3. Metro_2013
4. Civilian_labor_force_2008
5. Civilian_labor_force_2009
6. Civilian_labor_force_2010
7. Civilian_labor_force_2011
8. Civilian_labor_force_2012
9. Civilian_labor_force_2013
10. Civilian_labor_force_2014
11. Civilian_labor_force_2015
12. Civilian_labor_force_2016
13. Civilian_labor_force_2017
14. Civilian_labor_force_2018

Then we splitted the data set into train and test data set with 80%-20% as shown as follows. Once the data set splitted, we created object y with target variable ‘**Unemployment_rate_2018**’ using pd.DataFrame option. And the remaining columns were saved into object name x. Both x and y were converted into numpy arrays.

LINEAR REGRESSION:

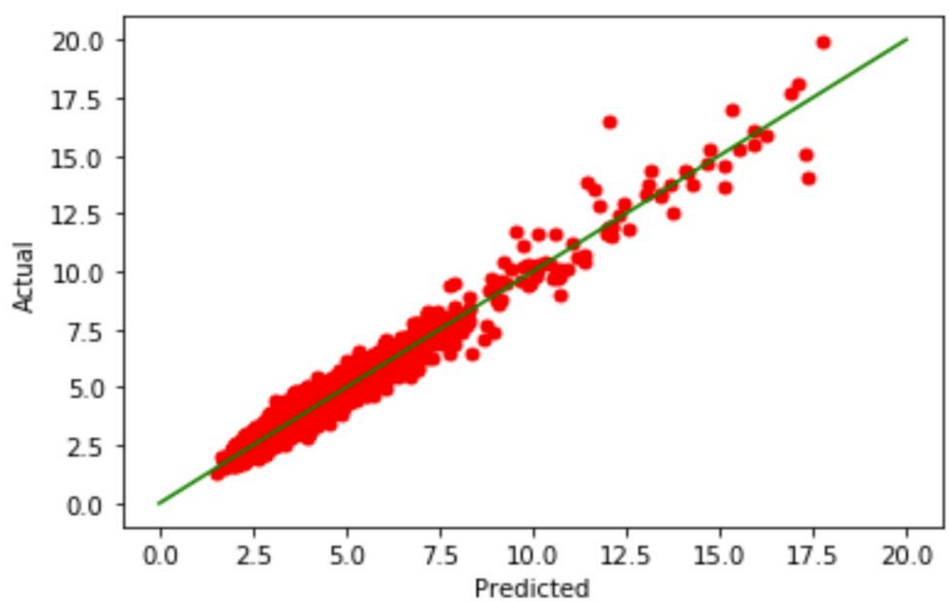
Linear Regression was done first using the LinearRegression() module from sklearn.linear_model. We got RMSE as 0.396.

```
In [104]:  total_error = np.dot(err.T,err)

           # Finally compute RMSE
           rmse_train = np.sqrt(total_error/len(p))
           print("RMSE on Training Data: ", rmse_train)

           RMSE on Training Data:  [[0.396]]
```

On checking the accuracy (R Square Value) , we got 0.95582.

```
In [101]:  linreg.score(x,y)

Out[101]:  0.955822538089255
```

The below scatterplot shows the Regression line with Actual value vs Predicted Value using Linear Regression.



Then we did 10 fold cross validation using linear regression.

```
Fold  1 RMSE: 0.4170
Fold  2 RMSE: 0.4104
Fold  3 RMSE: 0.4383
Fold  4 RMSE: 0.3848
Fold  5 RMSE: 0.3862
Fold  6 RMSE: 0.3709
Fold  7 RMSE: 0.3901
Fold  8 RMSE: 0.4565
Fold  9 RMSE: 0.4141
Fold 10 RMSE: 0.3723
```

RIDGE REGRESSION:

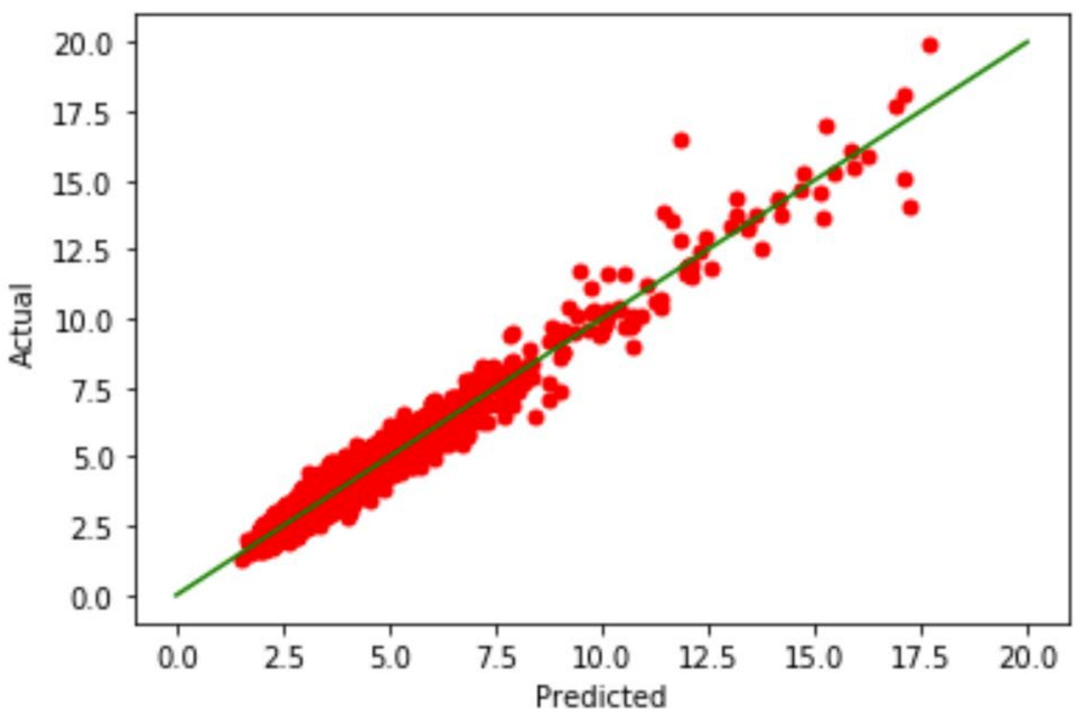
Ridge Regression was done first using the Ridge() module from sklearn.linear_model. We gave alpha value as 20. We got RMSE as 0.396.

```
In [113]:  rmse_r

Out[113]:  array([[0.396]])
```

On checking the accuracy (R Square Value) , we got 0.95574.

```
In [111]:  ridge.score(x,y)

Out[111]:  0.9557376865309639
```

The below scatterplot shows the Regression line with Actual value vs Predicted Value using Ridge Regression.



ELASTIC-NET REGRESSION:

Elastic_Net Regression was done using the ElasticNet() module from sklearn.linear_model. We gave the alpha value as 0.5. We got RMSE as 1.5895.

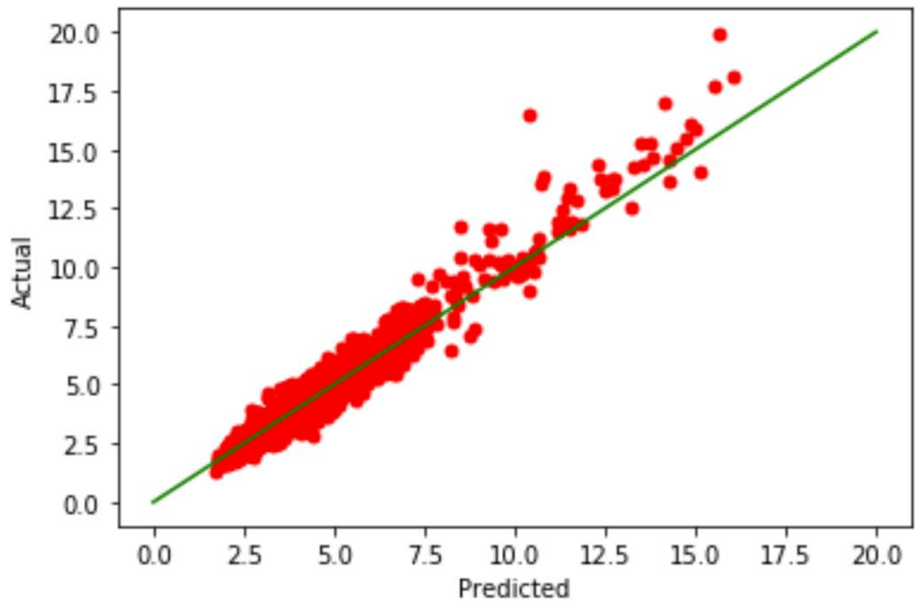
```
In [46]:  rmse_en = np.sqrt(np.linalg.norm(p_en - y)) / np.sqrt(len(y))
          rmse_en

Out[46]:  1.5895040552647781
```

On checking the accuracy (R Square Value) , we got 0.93360.

```
In [44]:  en.score(x,y)
Out[44]:  0.9336034828704546
```

The below scatterplot shows the Regression line with Actual value vs Predicted Value using Elastic-Net Regression

.



LASSO REGRESSION:

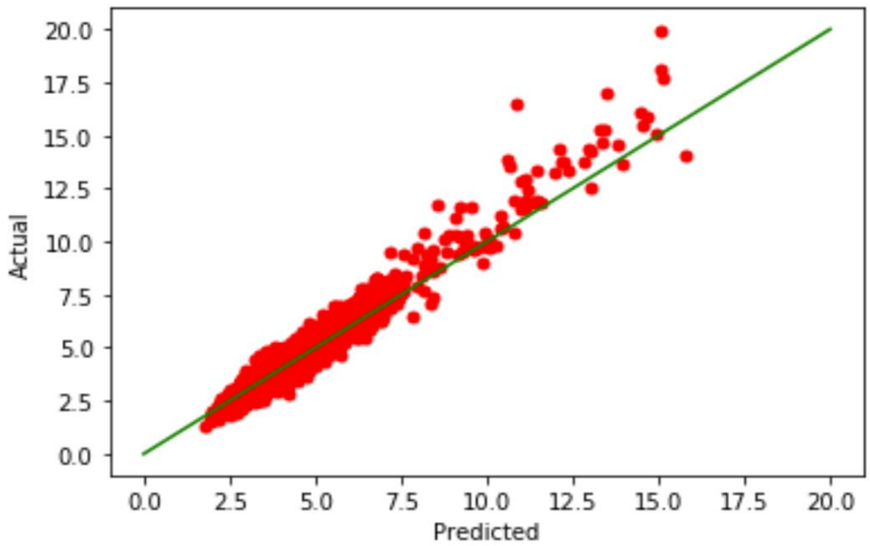
Lasso Regression was done first using the Lasso() module from sklearn.linear_model.module from sklearn.linear_model. We gave the alpha value as 0.5. We got RMSE as 1.57436.

```
In [51]:  rmse_la = np.sqrt(np.linalg.norm(p_la - y)) / np.sqrt(len(y))
           rmse_la
Out[51]:  1.5743621001614931
```

On checking the accuracy (R Square Value) , we got 0.93234.

```
In [55]:  lasso.score(x,y)
Out[55]:  0.9323435587586455
```

The below scatterplot shows the Regression line with Actual value vs Predicted Value using Lasso Regression.



STOCHASTIC GRADIENT DESCENT

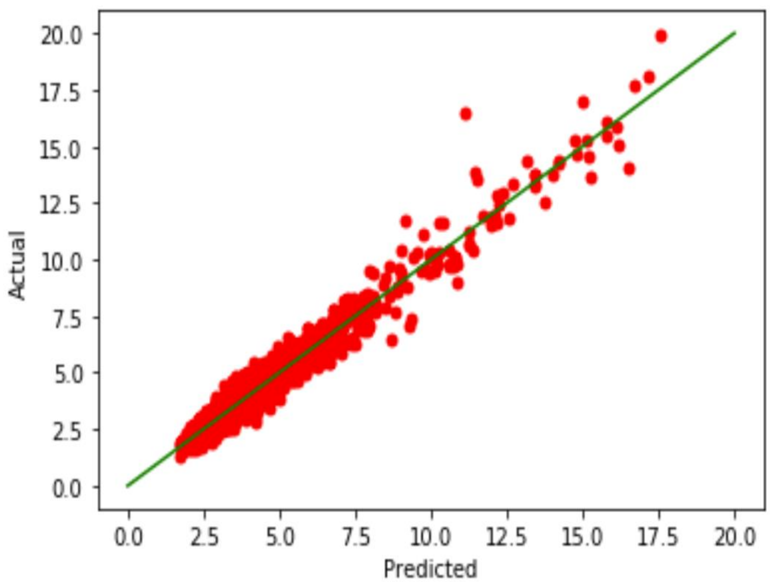
Stochastic Gradient Descent Regression was done first using the SGDRegressor() module from sklearn.linear_model. We gave penalty = l2, alpha = 0.01, max_iter = 300. We got RMSE as 0.396.

```
Out[53]: array([[0.396]])
```

On checking the accuracy (R Square Value) , we got 0.95221.

```
In [54]:  sgdreg.score(x_s,y)
Out[54]: 0.9522165276682497
```

The below scatterplot shows the Regression line with Actual value vs Predicted Value using Stochastic Gradient Descent Regression.



Thus all five different types of regression were done and we did compare the accuracy of different models.

```
Comparing Accuracy:
Accuracy for Linear Regression:  0.953830323005319
Accuracy for Ridge Regression:  0.9557008928914242
Accuracy for Lasso Regression:  0.9323435587586455
Accuracy for Elastic-Net Regression:  0.9336034828704546
Accuracy for Stochastic Gradient Descent Regression:  0.9522165276682497
```

On comparing we came to the conclusion that Ridge Regression has the high and we selected it as the best regression type for our data set and we did find the intercept and coefficient of ridge regression.

```
In [74]:  print('Ridge Regression Intercept',ridge.intercept_)

          Ridge Regression Intercept [0.143]

In [41]:  print('Ridge Regression Intercept Coefficient',ridge.coef_)

          Ridge Regression Intercept Coefficient [[-0.     0.018  0.036  0.044 -0.044 -0.016 -0.036  0.074  0.007 -0.039  0.187 -0.138
           0.799  0.002  0.006  0.    ]]
```

TESTING:

Then we did the testng with Ridge Regression to our test data set. y_t is our target variable here 643 rows. On testing we found that accuracy and rmse value are similar to the train data set.

```
In [70]:  ridge.fit(x_t,y_t)
          ridge.score(x_t,y_t)

Out[70]:  0.9584574594320934

In [72]:  p_t = ridge.predict(x_t) # p is the array of predicted values

          # Now we can constuct an array of errors
          err_t = abs(p_t-y_t)

In [73]:  total_error_t = np.dot(err_t.T,err_t)
          rmse_test = np.sqrt(total_error_t/len(p_t))
          print("RMSE on Test Data: ", rmse_test)

          RMSE on Test Data:  [[0.382]]
```

PREDICTION FILE:

With the intercept value and coefficient value we created a python file where if we give the values of x’s then it will give us the prediction value.

```
31 - New Hampshire
32 - New Jersey
33 - New Mexico
34 - Nevada
35 - New York
36 - Ohio
37 - Oklahoma
38 - Oregon
39 - Pennsylvania
40 - Puerto Rico
41 - Rhode Island
42 - South Carolina
43 - South Dakota
44 - Tennessee
45 - Texas
46 - Utah
47 - Virginia
48 - Vermont
49 - Washington
50 - Wisconsin
51 - West Virginia
52 - Wyoming
Rural_urban_continuum_code_2013 of your County: 4
Unemployment_rate_2007: 4.54
Unemployment_rate_2008: 5.45
Unemployment_rate_2009: 4.55
Unemployment_rate_2010: 6.54
Unemployment_rate_2011: 3.45
Unemployment_rate_2012: 6.765
Unemployment_rate_2013: 5.667
Unemployment_rate_2014: 7.65
Unemployment_rate_2015: 4.44
Unemployment_rate_2016: 5.65
Unemployment_rate_2017: 5.677
Med_HouseHold_Income_Percent_of_State_Total_2017: 78
State (Give the number as seen above) : 15
Predicted Unemplyment Rate is 5.263632
Press any key to continue . . .
```

Above screenshot is the output of that python file.

CONCLUSION:

Classification:

Random Forest Classifier model is the best model with accuracy of 80.7%

Clustering:

None of the models are accurate for the data. Hence, clustering is not a good method for the data. Based on the research, Decomposing the data to 4 principle components gives the best accuracy which is 25%.

Regression:

Ridge Model is the best model. Accuracy is 0.95574 and RMSE is 0.396.

Video Link -
https://www.youtube.com/watch?v=CD3p2QzCays&feature=youtu.be

CODE:

```
import numpy as np
import pandas as pd
# Deleted state wise total values in dataset
# Median_Household_Income_2017 column has "$" symbol. So, deleted dollar symbol in excel sheet
df = pd.read_excel("Unemployment_data.xlsx", delimiter=',')
df.head(5)
df.shape
df.columns
df[89:]
# deleting 89,91,92,95,96 rows because, they have many missing values
df_new = df.drop(df.index[[89,91,92,95,96]])
df_new.shape
# In[8]:
```

df_new.info()

All data types of columns are correct. So, no need to convert

df_new.isnull().sum()

From the above null data, there are 3 more rows with null values. So, we should delete those rows

Deleting rows 76,87,88

df_new = df_new.drop(df_new.index[[76,87,88]])

df_new.shape

again checking for null values

df_new.isnull().sum()

Median_Household_Income_2017, Med_HH_Income_Percent_of_State_Total_2017 have null values. Replacing them with mean of the column

replacing "Med_HH_Income_Percent_of_State_Total_2017" column

HHI_mean = df_new.Med_HH_Income_Percent_of_State_Total_2017.mean()

df_new.Med_HH_Income_Percent_of_State_Total_2017.fillna(HHI_mean, inplace=True)

replacing "Median_Household_Income_2017" column

HI_mean = df_new.Median_Household_Income_2017.mean()

df_new.Median_Household_Income_2017.fillna(HI_mean, inplace=True)

Again checking for null values

df_new.isnull().sum()

PERFORMING REGRESSION

import seaborn as sns

df_new.corr()

import matplotlib.pyplot as plt

plt.figure(figsize = (25,20))

sns.heatmap(df_new.corr(), cmap='coolwarm', linecolor='white',linewidths=0.5,)

plt.hist(df['Rural_urban_continuum_code_2013'])

plt.hist(df['Urban_influence_code_2013'])

plt.hist(df['Median_Household_Income_2017'])

plt.hist(df['Med_HH_Income_Percent_of_State_Total_2017'])

df_reg = df_new.drop(['FIPS', 'State', 'Urban_influence_code_2013', 'Median_Household_Income_2017'], axis = 1)

df_reg

```
area_name = df_reg['Area_name']
area_name_lst = np.array(area_name)
lst = []
count = 1
for i in area_name:
    lst.append(count)
    count +=1
lst[-1]
lst = pd.DataFrame(lst)
lst.shape
area_lst = np.arange(1,3223)
area_lst = pd.DataFrame(area_lst)
df_reg['Area_name'].shape #= df_reg[]
df_reg['Area_name'] = area_lst
df_reg
#df_reg.to_excel('df_reg.xlsx')
df_reg['State'] = df['State']
df_reg.head(5)
df_reg.State = pd.Categorical(df_reg.State)
df_reg['State'] = df_reg.State.cat.codes
state = df_reg['State']
state_arr = np.array(state)
for i in range(0,len(state_arr)):
    state_arr[i] += 1
state = pd.DataFrame(state_arr)
state.head(5)
df_reg['State'] = state
df_reg.head(5)
df_reg.columns
df_reg = df_reg.drop([' Employed_2007 ', ' Unemployed_2007 ',' Employed_2008 ', ' Unemployed_2008 ',
                    'Employed_2009', 'Unemployed_2009',' Employed_2010 ', ' Unemployed_2010 ',
                    ' Employed_2011 ', ' Unemployed_2011 ',' Employed_2012 ', ' Unemployed_2012 ',
```

```
' Employed_2013 ', ' Unemployed_2013 ',' Employed_2014 ', ' Unemployed_2014 ',
' Employed_2015 ', ' Unemployed_2015 ',' Employed_2016 ', ' Unemployed_2016 ',
'Employed_2017', 'Unemployed_2017','Employed_2018','Unemployed_2018'], axis
= 1)
df_reg.head(5)
#df_reg.to_excel('df_reg.xlsx')
```

CLASSIFICATION:

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
import numpy as np
import pandas as pd
from sklearn import metrics
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
# In[183]:
df = pd.read_excel("C:\\Users\\vallabh\\Vallabha Datta\\Project\\df_reg.xlsx", delimiter=',')
df.head(5)
# In[184]:
df = df.drop('Unnamed: 0', axis=1)
df.head(5)
# In[185]:
# Checking for data types and null values
df.info()
# From above info, there are 8 null values in "State" column
# In[186]:
state_mean = int(df.State.mean())
state_mean
# Replacing null value with mean
# For Montana cat code is 27, So, replacing missing state with Montana
# df.State.fillna(state_mean, inplace=True)
# In[187]:
df.State.fillna(state_mean, inplace=True)
```

```
# In[188]:
df.info()
# In[189]:
df.shape
# In[190]:
df['Unemployment_rate_2018'].describe()
# In[191]:
# Changing Unemployment_rate_2018 to category to perform knn
# Dividing into 5 Categories
lst = []
for i in df.Unemployment_rate_2018:
    if i<=3.1:
        lst.append("Very low")  # If Unemployment_rate is less than 3.1, then it is very low
    elif i>3.1 and i<=3.9:
        lst.append("Low")       # If Unemployment_rate is between 3.1 & 3.9[including 3.9] then it is low
    elif i>3.9 and i<=4.9:
        lst.append("Average")   # If Unemployment_rate is between 3.9 & 4.9[including 4.9] then it is Average
    elif i>4.9 and i<=10:
        lst.append("High")      # If Unemployment_rate is between 4.9 & 10[including 10] then it is High
    else:
        lst.append("Very High") # If Unemployment_rate is greater than 10, then it is very high
lst
# In[192]:
df.Unemployment_rate_2018 = lst
# In[193]:
df['Unemployment_rate_2018'].head(5)
# In[194]:
# Selecting Unemployment_rate_2018 as target variable
df_target = df['Unemployment_rate_2018']
df = df.drop('Unemployment_rate_2018', axis =1)
```

```
# In[195]:
from sklearn import preprocessing
# In[196]:
# Performing min-max scaling
min_max_scaler = preprocessing.MinMaxScaler()
min_max_scaler.fit(df)
# In[197]:
df_scaled = min_max_scaler.fit_transform(df)
df_scaled
# In[198]:
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)
# In[199]:
df_scaled.head(5)
# In[200]:
np.set_printoptions(precision=2, linewidth=100)
# In[201]:
# Splitting data into training and testing
from sklearn.model_selection import train_test_split
df_train, df_test, df_target_train, df_target_test = train_test_split(df_scaled, df_target,
test_size=0.2, random_state=478)
# In[202]:
df_target_train.head(5)
# In[203]:
df_target_test.head(5)
# In[204]:
df_train.head(5)
# In[205]:
df_test.head(5)
# In[206]:
print("Shape of Train data: ", df_train.shape)
print("Shape of Test data: ", df_test.shape)
# In[207]:
df_train_arr = np.array(df_train)
```

```
df_test_arr = np.array(df_test)
df_target_train_arr = np.array(df_target_train)
df_target_test_arr = np.array(df_target_test)
# In[208]:
def knn_search(x, D, K, measure):
    """ find K nearest neighbors of an instance x among the instances in D """
    if measure == 0:
        # euclidean distances from the other points
        dists = np.sqrt(((D - x)**2).sum(axis=1))
    elif measure == 1:
        # first find the vector norm for each instance in D as wel as the norm for vector x
        D_norm = np.array([np.linalg.norm(D[i]) for i in range(len(D))])
        x_norm = np.linalg.norm(x)
        # Compute Cosine: divide the dot product o x and each instance in D by the product of the
two norms
        sims = np.dot(D,x)/(D_norm * x_norm)
        # The distance measure will be the inverse of Cosine similarity
        dists = 1 - sims
    idx = np.argsort(dists) # sorting
    # return the indexes of K nearest neighbors
    return idx[:K], dists
# In[209]:
neigh_idx, distances = knn_search(df_test_arr[0], df_train_arr, 5, 0)
# In[210]:
print(neigh_idx)
print("\nNearest Neigbors:")
df_train.iloc[neigh_idx]
# In[211]:
# printing distances of top 5 nearest neighbors
print(distances[neigh_idx])
# In[212]:
neigh_labels = df_target_train_arr[neigh_idx]
print(neigh_labels)
```

Top 5 nearest neighbors are: High, Average, High, Average, Average

In[213]:

from collections import Counter

print(Counter(neigh_labels))

In[214]:

Counter(neigh_labels).most_common(1)

Since, majority of votes are Average for top 5 nearest neighbours, the predicted value of unemployment rate will fall under Average category

In[215]:

X_train, X_test, y_train, y_test = train_test_split(df_scaled, df_target, test_size=0.2, random_state=50)

In[216]:

from sklearn.neighbors import KNeighborsClassifier

Using KNN classifier

In[217]:

knn = KNeighborsClassifier(n_neighbors=5)

In[218]:

knn.fit(X_train, y_train)

In[219]:

Performing prediction

pred = knn.predict(X_test)

In[220]:

from sklearn.metrics import classification_report, confusion_matrix

In[105]:

print(confusion_matrix(y_test, pred))

print(classification_report(y_test, pred))

In[221]:

```
error_rate = []
for i in range(1,40):

    knn= KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
```

```
    error_rate.append(np.mean(pred_i != y_test))

# In[222]:

import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))

plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker = 'o',
markerfacecolor='red', markersize=10)

plt.title('Error Rate vs K value')

plt.xlabel('K')

plt.ylabel('Error Rate')

# From the Error vs K plot, the best value of k is 3. So, performing prediction with 3 neighbors

# In[738]:

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

pred_knn = knn.predict(X_test)

print(confusion_matrix(y_test, pred))

print(classification_report(y_test, pred))

# Accuracy using knn classifier is 72%

# 2) USING TERM DOCUMENT CATEGORIZATION

# In[223]:

df_train.shape

# In[224]:

numTerms=df_train.shape[0]

NDocs = df_train.shape[1]

# In[225]:

termFreqs = df_train.sum(axis=1)

termFreqs.head(5)

# In[226]:

plt.plot(sorted(termFreqs, reverse=True))

plt.show()

# In[227]:

DF = pd.DataFrame([(df_train!=0).sum(1)]).T

DF.head(5)

# In[228]:
```

```
NMatrix=np.ones(np.shape(df_train), dtype=float)*NDocs
np.set_printoptions(precision=2,suppress=True,linewidth=120)
print(NMatrix)
# In[229]:
# Convert each entry into IDF values
# IDF is the log of the inverse of document frequency
# Note that IDF is only a function of the term, so all columns will be identical.
IDF = np.log2(np.divide(NMatrix, np.array(DF)))
# In[230]:
IDF
# In[231]:
TD_tfidf = df_train * IDF
# In[232]:
TD_tfidf.head(10)
# In[233]:
IDF.T[0].shape
# Converting test data using TD*iDF
# In[234]:
numTerms2=df_test.shape[0]
NDocs2 = df_test.shape[1]
# In[235]:
DF2 = pd.DataFrame([(df_test!=0).sum(1)]).T
DF2.head(10)
# In[236]:
NMatrix2=np.ones(np.shape(df_test), dtype=float)*NDocs2
np.set_printoptions(precision=2,suppress=True,linewidth=120)
print(NMatrix2)
# In[237]:
IDF2 = np.log2(np.divide(NMatrix2, np.array(DF2)))
IDF2
# In[238]:
df2_tfidf = df_test * IDF2
# In[239]:
```

```
df2_tfidf.head(5)

# In[240]:

error_rate = []

for i in range(1,40):


    knn= KNeighborsClassifier(n_neighbors = i)
    knn.fit(df_train, df_target_train)
    pred_i = knn.predict(df_test)
    error_rate.append(np.mean(pred_i != df_target_test))

# In[241]:

plt.figure(figsize=(10,6))
plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker = 'o',
markerfacecolor='red', markersize=10)
plt.title('Error Rate vs n_neighbors')
plt.xlabel('n_neighbors')
plt.ylabel('Error Rate')

# In[242]:

# From the above graph the best k value is 10
knn = KNeighborsClassifier(n_neighbors=10)

# In[243]:

knn.fit(TD_tfidf, df_target_train)

# In[244]:

pred = knn.predict(df2_tfidf)

# In[546]:

print(confusion_matrix(df_target_test, pred))
print(classification_report(df_target_test, pred))

# In[732]:

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(df_target_test, pred))
# From the above classification report,
# Wei_Ave using TD*IDF = 0.61,
# Wei_Ave using Knn = 0.73. So, knn model without using TD_IDF is best in this case.
# Predicting the Unemploment_rate for random query
```

```
# In[245]:
import random
x=[]
for i in range(1,29):
    y = random.random()
    x.append(y)
# In[246]:
# Each term in query x must be multiplied by the idf value of the term we computed earlier (the IDF matrix)
x_tfidf = x * IDF[0]  # note that this coordinatewise multiplication of two vectors
print(x_tfidf)
# In[247]:
x_tfidf.shape
# In[248]:
DT_tfidf = TD_tfidf
DT_array = np.array(DT_tfidf)
# In[249]:
DT_array.shape
# In[250]:
df.head(5)
# In[251]:
DT_array
# In[253]:
neigh_idx, distances = knn_search(x_tfidf, DT_array, 5, 0)
# In[261]:
# Distances between query objects and training objects
distances
# In[262]:
distances = pd.Series(distances, index=DT_tfidf.index)
# In[263]:
print("Query:", x)
print("\nNeighbors:")
DT_tfidf.iloc[neigh_idx]
```

```
# In[264]:
df_target.shape
# In[265]:
cat_labels = np.array(df_target_train)
cat_labels = pd.Series(cat_labels, index=DT_tfidf.index)
DT_tfidf["Category"] = cat_labels
# In[266]:
def knn_classify(x, D, K, labels, measure):
    from collections import Counter
    neigh_idx, distances = knn_search(x, D, K, measure)
    neigh_labels = labels[neigh_idx]
    count = Counter(neigh_labels)
    print("Labels for top ", K, "neighbors: ", count)
    return count.most_common(1)[0][0]
# In[267]:
print("Instance to classify:\n", x)
print("Predicted Category for the new instance: ", knn_classify(x_tfidf, DT_array, 5, cat_labels, 0))
# Predicted category for the new instance using TD*IDF is "High"
# 3) USING DECISION TREE CLASSIFIER
# In[268]:
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
# In[269]:
dtree.fit(df_train,df_target_train)
# In[270]:
predictions = dtree.predict(df_test)
# In[271]:
print(dtree.score(df_test, df_target_test))
# In[272]:
print(dtree.score(df_train, df_target_train))
# Tree performed well for training set than testing set. Tree score for training set is 1 which is not possible practically. So, we can infer that the decision tree model is too simple and underfitted
```

```
# In[552]:

print(classification_report(df_target_test,predictions))

# Accuracy using decision tree is 72%

# Setting criterion to entropy

# In[273]:

dtree = DecisionTreeClassifier(criterion = "entropy")

# In[274]:

dtree.fit(df_train,df_target_train)

# In[275]:

predictions = dtree.predict(df_test)

print(classification_report(df_target_test,predictions))

# After setting criterion as "entropy", I got accuracy as 76%. Which is better than "gini" criterion

# Changing default values in decision tree classifier

# In[276]:

error_rate = []

for i in range(1,40):


    dtree = DecisionTreeClassifier(criterion = "entropy", max_depth = i)

    dtree.fit(df_train,df_target_train)

    predictions = dtree.predict(df_test)

    error_rate.append(np.mean(predictions != df_target_test))

# In[277]:

plt.figure(figsize=(10,6))

plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker = 'o',
markerfacecolor='red', markersize=10)

plt.title('Error Rate vs max_depth')

plt.xlabel('max_depth')

plt.ylabel('Error Rate')

# In[278]:

# from the above plot, best value of max_depth =13

dtree = DecisionTreeClassifier(criterion = "entropy",max_depth = 13)

# In[279]:

t = dtree.fit(df_train,df_target_train)
```

```
# In[673]:
predictions = dtree.predict(df_test)
print(classification_report(df_target_test,predictions))
# In[731]:
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(df_target_test, predictions))
# So, by using decision tree classifier the best accuracy achieved is 78%
# In[280]:
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydot
features = list(df.columns[0:])
features
# Printing and exporting decision tree
# In[691]:
dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data,feature_names=features,filled=True,rounded=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
# 4)USING NAIVE BAYES (GAUSSIAN) CLASSIFIER
# In[281]:
from sklearn import naive_bayes
nb_model = naive_bayes.GaussianNB()
nb_model = nb_model.fit(df_train, df_target_train)
nb_pred = nb_model.predict(df_test)
nb_pred[1:20]
# In[282]:
print(nb_model.score(df_test, df_target_test))
# In[283]:
print(nb_model.score(df_train, df_target_train))
# In[284]:
confusion_mat = confusion_matrix(df_target_test, nb_pred)
```

```
print(confusion_mat)

# In[474]:

print(classification_report(df_target_test, nb_pred))

# In[730]:

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(df_target_test, nb_pred))

# Accuracy using Naive Bayes is 54.4%

# 5) USING RANDOM FOREST CLASSIFIER

# In[285]:

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=100,criterion='entropy', random_state=None)

rfc.fit(df_train, df_target_train)

# In[286]:

rfc_pred = rfc.predict(df_test)

# In[287]:

print(confusion_matrix(df_target_test,rfc_pred))

# In[288]:

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(df_target_test, rfc_pred))

# By using default values for RandomForestClassifier we got 78.69% accuracy

# In[289]:

error_rate = []

for i in range(1,40):


    rfc = RandomForestClassifier(n_estimators=100,criterion='entropy', random_state=None,
max_depth=i)

    rfc.fit(df_train, df_target_train)

    rfc_pred = rfc.predict(df_test)

    error_rate.append(np.mean(rfc_pred != df_target_test))

# In[290]:

plt.figure(figsize=(10,6))

plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker = 'o',
markerfacecolor='red', markersize=10)
```

plt.title('Error Rate vs max_depth')

plt.xlabel('max_depth')

plt.ylabel('Error Rate')

In[349]:

best value of max_depth is 40

rfc = RandomForestClassifier(n_estimators=100,criterion='entropy', random_state=None,max_depth=40)

rfc.fit(df_train, df_target_train)

In[350]:

rfc_pred = rfc.predict(df_test)

In[351]:

print(confusion_matrix(df_target_test,rfc_pred))

print("Accuracy:",metrics.accuracy_score(df_target_test, rfc_pred))

By setting n_estimators to 100,criterion to 'entropy', random_state to None and max_depth to 40, the maximum accuracy achieved is 80.7%

Of all of the above classification techniques, Random Forest Classifier has maximum accuracy. So, let's perform prediction using random forest

CLUSTERING:

#!/usr/bin/env python

coding: utf-8

In[1]:

import numpy as np

import pandas as pd

from sklearn import metrics

import matplotlib.pyplot as plt

get_ipython().run_line_magic('matplotlib', 'inline')

In[2]:

df = pd.read_excel("df_reg.xlsx", delimiter=',')

df.head(5)

In[3]:

df = df.drop('Unnamed: 0', axis=1)

df.head(5)

In[4]:

```
state_mean = int(df.State.mean())
state_mean
# In[5]:
df.State.fillna(state_mean, inplace=True)
# In[6]:
# Changing Unemployment_rate_2018 to category to perform knn
# Dividing into 5 Categories
lst = []
for i in df.Unemployment_rate_2018:
    if i<=3.1:
        lst.append("Very low")  # If Unemployment_rate is less than 3.1, then it is very low
    elif i>3.1 and i<=3.9:
        lst.append("Low")       # If Unemployment_rate is between 3.1 & 3.9[including 3.9] then it is low
    elif i>3.9 and i<=4.9:
        lst.append("Average")   # If Unemployment_rate is between 3.9 & 4.9[including 4.9] then it is Average
    elif i>4.9 and i<=10:
        lst.append("High")      # If Unemployment_rate is between 4.9 & 10[including 10] then it is High
    else:
        lst.append("Very High") # If Unemployment_rate is greater than 10, then it is very high
lst
# In[7]:
df.Unemployment_rate_2018 = lst
# In[8]:
df['Unemployment_rate_2018'].head(5)
# In[9]:
df_target = df['Unemployment_rate_2018']
df = df.drop('Unemployment_rate_2018', axis =1)
# In[10]:
from sklearn import preprocessing
# In[11]:
```

```
min_max_scaler = preprocessing.MinMaxScaler()
min_max_scaler.fit(df)
# In[12]:
df_scaled = min_max_scaler.fit_transform(df)
df_scaled
# In[13]:
df_scaled = pd.DataFrame(df_scaled, columns=df.columns)
# In[14]:
np.set_printoptions(precision=2, linewidth=100)
# In[15]:
from sklearn.model_selection import train_test_split
df_train, df_test, df_target_train, df_target_test = train_test_split(df_scaled, df_target, test_size=0.2, random_state=478)
# In[16]:
df_train.head(5)
# In[17]:
df_target_train.head(5)
# In[18]:
df_train_arr = np.array(df_train)
df_test_arr = np.array(df_test)
df_target_train_arr = np.array(df_target_train)
df_target_test_arr = np.array(df_target_test)
# In[19]:
from sklearn.cluster import KMeans
# In[156]:
'''
k Means Clustering for Ch10 of Machine Learning in Action
@author: Peter Harrington
'''
from numpy import *
from numpy import dot
from numpy.linalg import norm
def distEuclid(vecA, vecB):
```

```
    return sqrt(sum(power(vecA - vecB, 2))) #la.norm(vecA-vecB)
def Cosine_dist(vecA, vecB):
    """ find K nearest neighbors of an instance x among the instances in D """
    cos_sim = dot(vecA, vecB)/(norm(vecA)*norm(vecB))
    dists = 1-cos_sim
    return dists
def randCent(dataSet, k):
        n = shape(dataSet)[1]
        centroids = zeros((k,n), dtype=float)
        for j in range(n): #create random cluster centers
            minJ = min(dataSet[:,j])
            rangeJ = float(max(dataSet[:,j]) - minJ)
            centroids[:,j] = minJ + rangeJ * random.rand(k)
        return centroids
def kMeans(dataSet, k, distMeas=distEuclid, createCent=randCent):
    m = shape(dataSet)[0]
    clusterAssment = zeros((m,2)) #create mat to assign data points
                                  #to a centroid, also holds SE of each point
    centroids = createCent(dataSet, k)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m): #for each data point assign it to the closest centroid
            minDist = inf; minIndex = -1
            for j in range(k):
                distJI = distMeas(centroids[j,:],dataSet[i,:])
                if distJI < minDist:
                    minDist = distJI; minIndex = j
            if clusterAssment[i,0] != minIndex: clusterChanged = True
            clusterAssment[i,:] = minIndex,minDist**2
        # print centroids
        for cent in range(k):#recalculate centroids
```

```
            ptsInClust = dataSet[nonzero(clusterAssment[:,0]==cent)[0]] #get all the point in this cluster - Note: this was incorrect in the original distribution.
            if(len(ptsInClust)!=0):
                centroids[cent,:] = mean(ptsInClust, axis=0) #assign centroid to mean - Note condition was added 10/28/2013
    return centroids, clusterAssment
def biKmeans(dataSet, k, distMeas=distEuclid):
    m = shape(dataSet)[0]
    clusterAssment = mat(zeros((m,2)))
    centroid0 = mean(dataSet, axis=0).tolist()[0]
    centList =[centroid0] #create a list with one centroid
    for j in range(m): #calc initial Error
        clusterAssment[j,1] = distMeas(mat(centroid0), dataSet[j,:])**2
    while (len(centList) < k):
        lowestSSE = inf
        for i in range(len(centList)):
            ptsInCurrCluster = dataSet[nonzero(clusterAssment[:,0].A==i)[0],:] #get the data points currently in cluster i
            centroidMat, splitClustAss = kMeans(ptsInCurrCluster, 2, distMeas)
            sseSplit = sum(splitClustAss[:,1]) #compare the SSE to the currrent minimum
            sseNotSplit = sum(clusterAssment[nonzero(clusterAssment[:,0].A!=i)[0],1])
            print("sseSplit, and notSplit: ",sseSplit,sseNotSplit)
            if (sseSplit + sseNotSplit) < lowestSSE:
                bestCentToSplit = i
                bestNewCents = centroidMat
                bestClustAss = splitClustAss.copy()
                lowestSSE = sseSplit + sseNotSplit
        bestClustAss[nonzero(bestClustAss[:,0] == 1)[0],0] = len(centList) #change 1 to 3,4, or whatever
        bestClustAss[nonzero(bestClustAss[:,0] == 0)[0],0] = bestCentToSplit
        print('the bestCentToSplit is: ',bestCentToSplit)
        print('the len of bestClustAss is: ', len(bestClustAss))
```

```
            centList[bestCentToSplit] = bestNewCents[0,:].tolist()[0] #replace a centroid with two best
centroids
            centList.append(bestNewCents[1,:].tolist()[0])
            clusterAssment[nonzero(clusterAssment[:,0].A == bestCentToSplit)[0],:]= bestClustAss
#reassign new clusters, and SSE
        return mat(centList), clusterAssment
# In[167]:
centroids_matrix, clusters_matrix = kMeans(df_train_arr, 5, Cosine_dist, randCent)
# In[168]:
centroids_matrix
# In[169]:
clusters_matrix
# In[170]:
clusters_matrix2 = pd.DataFrame(clusters_matrix, columns=['Cluster','MinDistance**2'])
clusters_matrix2.head(10)
# In[171]:
size = cluster_sizes(clusters_matrix2.Cluster)
size
# In[166]:
size_new = {"Very Low": 481, "Low": 458, "Average": 751, "High": 61, "Very High": 820}
size_new
# In[164]:
for i in size_new.keys():
    print(i, "= ", size_new[i])
# In[131]:
df_target_train.head(5)
# In[132]:
clusters_matrix2.Cluster.head(10)
# In[149]:
TT = np.array(df_target_train)
TT_new = []
for i in TT:
    if i == "Very low":
```

```
        TT_new.append(0.0)
    elif i == "Low":
        TT_new.append(1.0)
    elif i == "Average":
        TT_new.append(2.0)
    elif i == "High":
        TT_new.append(3.0)
    else:
        TT_new.append(4.0)
# In[150]:
TT_new
# In[151]:
TT_df = pd.DataFrame(TT_new)
# In[152]:
from sklearn.metrics import completeness_score, homogeneity_score
print(completeness_score(TT_df[0],clusters_matrix2.Cluster))
# In[153]:
print(homogeneity_score(TT_df[0],clusters_matrix2.Cluster))
# In[138]:
centroids_matrix2 = np.array(centroids_matrix)
centroids_matrix2
# In[139]:
def Cosine_dist(vecA, vecB):
    """ find K nearest neighbors of an instance x among the instances in D """
    cos_sim = dot(vecA, vecB)/(norm(vecA)*norm(vecB))
    dists = 1-cos_sim
    idx = np.argsort(dists) # sorting
    # return the indexes of K nearest neighbors
    return idx[0], max(cos_sim)
# In[140]:
arr = []
for i in range(len(df_test_arr)):
    cluster, simi = Cosine_dist(centroids_matrix2, df_test_arr[i])
```

```
    arr.append(cluster)
    print("%-12i%-12i   %-12f" % (i, cluster, simi))
# In[147]:
pred = np.array(arr)
completeness_score(df_target_test_arr, pred)
# In[148]:
homogeneity_score(df_target_test_arr, pred)
```

REGRESSION:

```
#!/usr/bin/env python
# coding: utf-8
# In[2]:


import pandas as pd
import numpy as np
import matplotlib as plt
import pylab as pl
import math
from sklearn.metrics import accuracy_score


# In[3]:


import sklearn as sk
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet, SGDRegressor


# In[4]:


data=pd.read_excel(r'C:\Users\dharu\Documents\Studies\ML\project\data.xlsx')


# In[5]:


data


# In[6]:
```

```
data.isnull().sum()

# In[7]:

data.describe()

# In[8]:

data.columns

# In[9]:

new_data=data.drop(['Unnamed: 0',' Civilian_labor_force_2007 ','Metro_2013',' Civilian_labor_force_2008 ','Civilian_labor_force_2009',' Civilian_labor_force_2010 ',' Civilian_labor_force_2011 ',' Civilian_labor_force_2012 ',' Civilian_labor_force_2013 ',' Civilian_labor_force_2014 ',' Civilian_labor_force_2015 ',' Civilian_labor_force_2016 ','Civilian_labor_force_2017','Civilian_labor_force_2018'],axis=1)

# In[10]:

new_data

# In[11]:

from sklearn.model_selection import train_test_split
train,test=train_test_split(new_data,test_size=0.2,random_state=33)

# In[12]:

y=pd.DataFrame(train['Unemployment_rate_2018'])

# In[13]:
```

```
y

# In[14]:

x=train

# In[15]:

x.columns

# In[16]:

x=x.drop(['Unemployment_rate_2018'],axis=1)

# In[17]:

x=np.array(x)

# In[18]:

x = np.array([np.concatenate((v,[1])) for v in x])

# In[19]:

x

# In[20]:

y=np.array(y)

# In[21]:
```

```
x

# In[22]:

np.set_printoptions(precision=3, linewidth=120, suppress=True, edgeitems=7)

# In[23]:

linreg=LinearRegression()

# In[24]:

x.shape

# In[25]:

y.shape

# In[26]:

linreg.fit(x,y)

# In[27]:

linreg.score(x,y)

# In[28]:

for i in range(len(y)):
    pred = linreg.predict(np.array([x[i]]))[0]
    print(pred)

# In[29]:
```

```
p = linreg.predict(x) # p is the array of predicted values
# Now we can constuct an array of errors
err = abs(p-y)
# Let's see the error on the first 10 predictions
print(err)


# In[30]:


total_error = np.dot(err.T,err)
# Finally compute RMSE
rmse_train = np.sqrt(total_error/len(p))
print("RMSE on Training Data: ", rmse_train)


# In[31]:


print('Linear Regression Intercept',linreg.intercept_)


# In[32]:


print('Linear Regression Coefficient',linreg.coef_)


# In[33]:


get_ipython().run_line_magic('matplotlib', 'inline')
pl.plot(p, y,'ro', markersize=5)
pl.plot([0,20],[0,20], 'g-')
pl.xlabel('Predicted')
pl.ylabel('Actual')
pl.show()


# In[34]:
```

```
def cross_validate(model,X,y,n,verbose=False):
    from sklearn.model_selection import KFold
    kf = KFold(n_splits=n, random_state=22)
    xval_err = 0
    f = 1
    for train,test in kf.split(x):
        model.fit(X[train],y[train])
        p = model.predict(x[test])
        e = p-y[test]
        rmse = np.sqrt(np.dot(e.T,e)/len(x[test]))
        if verbose:
            print("Fold %2d RMSE: %.4f" % (f, rmse))
        xval_err += rmse
        f += 1
    return xval_err/n

# In[35]:


rmse10=cross_validate(linreg,x,y,10,verbose=True)


# In[36]:


ridge=Ridge(alpha=20)
ridge.fit(x,y)


# In[37]:


ridge.score(x,y)


# In[38]:


p_r = ridge.predict(x)
err_r = p-y
```

```
total_error_r = np.dot(err_r.T,err_r)
rmse_r = np.sqrt(total_error_r/len(p_r))

# In[39]:

rmse_r

# In[40]:

rmse_10cv_ridge = cross_validate(ridge, x, y, 10, verbose=True)

# In[74]:

print('Ridge Regression Intercept',ridge.intercept_)

# In[41]:

print('Ridge Regression Intercept Coefficient',ridge.coef_)

# In[42]:

get_ipython().run_line_magic('matplotlib', 'inline')
pl.plot(p_r, y,'ro', markersize=5)
pl.plot([0,20],[0,20], 'g-')
pl.xlabel('Predicted')
pl.ylabel('Actual')
pl.show()

# In[43]:

en=ElasticNet(alpha=0.5)
en.fit(x,y)
```

```
# In[44]:


en.score(x,y)


# In[45]:


p_en = en.predict(x)


# In[46]:


rmse_en = np.sqrt(np.linalg.norm(p_en - y)) / np.sqrt(len(y))
rmse_en


# In[47]:


print('Elastic Net Regression Intercept Coefficient',en.coef_)


# In[48]:


get_ipython().run_line_magic('matplotlib', 'inline')
pl.plot(p_en, y,'ro', markersize=5)
pl.plot([0,20],[0,20], 'g-')
pl.xlabel('Predicted')
pl.ylabel('Actual')
pl.show()


# In[49]:


lasso=Lasso(alpha=0.5)
lasso.fit(x,y)


# In[50]:
```

```
p_la = lasso.predict(x)


# In[51]:


rmse_la = np.sqrt(np.linalg.norm(p_la - y)) / np.sqrt(len(y))
rmse_la


# In[55]:


lasso.score(x,y)


# In[52]:


get_ipython().run_line_magic('matplotlib', 'inline')
pl.plot(p_la, y,'ro', markersize=5)
pl.plot([0,20],[0,20], 'g-')
pl.xlabel('Predicted')
pl.ylabel('Actual')
pl.show()


# In[53]:


from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x)
x_s = scaler.transform(x)
sgdreg = SGDRegressor(penalty='l2', alpha=0.01, max_iter=300)
# Compute RMSE on training data
sgdreg.fit(x_s,y)
p_s = sgdreg.predict(x_s)
err_s = p_s-y
total_error_s = np.dot(err_s.T,err_s)
rmse_s = np.sqrt(total_error/len(p_s))
```

```
rmse_s

# In[54]:

sgdreg.score(x_s,y)

# In[65]:

get_ipython().run_line_magic('matplotlib', 'inline')
pl.plot(p_s, y,'ro', markersize=5)
pl.plot([0,20],[0,20], 'g-')
pl.xlabel('Predicted')
pl.ylabel('Actual')
pl.show()

# In[69]:

print('Comparing Accuracy:')
print('Accuracy for Linear Regression: ',linreg.score(x,y))
print('Accuracy for Ridge Regression: ',ridge.score(x,y))
print('Accuracy for Lasso Regression: ',lasso.score(x,y))
print('Accuracy for Elastic-Net Regression: ',en.score(x,y))
print('Accuracy for Stochastic Gradient Descent Regression: ',sgdreg.score(x_s,y))

# On Comparing the accuracy, we have higher accuracy in Ridge Regression. So we are going with Ridge regression for our model. Let's do the testing part now.
# In[57]:

y_t=pd.DataFrame(test['Unemployment_rate_2018'])

# In[58]:

x_t=test
```

```
# In[59]:


x_t=x_t.drop(['Unemployment_rate_2018'],axis=1)
x_t=np.array(x_t)
x_t = np.array([np.concatenate((v,[1])) for v in x_t])
y_t=np.array(y_t)

# In[60]:


y_t.shape


# In[70]:


ridge.fit(x_t,y_t)
ridge.score(x_t,y_t)

# In[72]:


p_t = ridge.predict(x_t) # p is the array of predicted values
# Now we can constuct an array of errors
err_t = abs(p_t-y_t)

# In[73]:


total_error_t = np.dot(err_t.T,err_t)
rmse_test = np.sqrt(total_error_t/len(p_t))
print("RMSE on Test Data: ", rmse_test)
```

VALIDATION CODE:

```
def reg():
```

```
print('1 - Alaska\n2 - Alabama\n3 - Arkansas\n4 - Arizona\n5 - California\n6 - Colorado\n7 - Connecticut\n8 - District of Columbia\n9 - Delaware\n10 - Florida\n11 - Georgia\n12 - Hawaii\n13 - Iowa\n14 - Idaho\n15 - Illinois\n16 - Indiana\n17 - Kansas\n18 - Kentucky\n19 - Louisiana\n20 - Massachusetts\n21 - Maryland\n22 - Maine\n23 - Michigan\n24 - Minnesota\n25 - Missouri\n26 - Mississippi\n27 - Montana\n28 - North Carolina\n29 - North Dakota\n30 - Nebraska\n31 - New Hampshire\n32 - New Jersey\n33 - New Mexico\n34 - Nevada\n35 - New York\n36 - Ohio\n37 - Oklahoma\n38 - Oregon\n39 - Pennsylvania\n40 - Puerto Rico\n41 - Rhode Island\n42 - South Carolina\n43 - South Dakota\n44 - Tennessee\n45 - Texas\n46 - Utah\n47 - Virginia\n48 - Vermont\n49 - Washington\n50 - Wisconsin\n51 - West Virginia\n52 - Wyoming')
a=int(input("Rural_urban_continuum_code_2013 of your County: "))
b=float(input("Unemployment_rate_2007: "))
c=float(input("Unemployment_rate_2008: "))
d=float(input("Unemployment_rate_2009: "))
e=float(input("Unemployment_rate_2010: "))
f=float(input("Unemployment_rate_2011: "))
g=float(input("Unemployment_rate_2012: "))
h=float(input("Unemployment_rate_2013: "))
i=float(input("Unemployment_rate_2014: "))
j=float(input("Unemployment_rate_2015: "))
k=float(input("Unemployment_rate_2016: "))
l=float(input("Unemployment_rate_2017: "))
m=float(input("Med_HouseHold_Income_Percent_of_State_Total_2017: "))
n=float(input("State (Give the number as seen above) : "))
if(a>10 or a<1):
   print("Incorrect Rural_urban_continuum_code_2013!! Should be between 1-9 ")
elif(b<0 or b>101):
   print("Incorrect Unemployment_rate_2007. Should be between 0-100")
elif(c<0 or c>101):
   print("Incorrect Unemployment_rate_2008. Should be between 0-100")
elif(d<0 or d>101):
   print("Incorrect Unemployment_rate_2009. Should be between 0-100")
```

```
        elif(e<0 or e>101):
            print("Incorrect Unemployment_rate_2010. Should be between 0-100")
        elif(f<0 or f>101):
            print("Incorrect Unemployment_rate_2011. Should be between 0-100")
        elif(g<0 or g>101):
            print("Incorrect Unemployment_rate_2012. Should be between 0-100")
        elif(h<0 or h>101):
            print("Incorrect Unemployment_rate_2013. Should be between 0-100")
        elif(i<0 or i>101):
            print("Incorrect Unemployment_rate_2014. Should be between 0-100")
        elif(j<0 or j>101):
            print("Incorrect Unemployment_rate_2015. Should be between 0-100")
        elif(k<0 or k>101):
            print("Incorrect Unemployment_rate_2016. Should be between 0-100")
        elif(l<0 or l>101):
            print("Incorrect Unemployment_rate_2017. Should be between 0-100")
        elif(m<0 or m>101):
            print("Incorrect Med_HouseHold_Income_Percent_of_State_Total_2017. Should be between 0-100")
        elif(n<0 or n>101):
            print("Incorrect State Number. Should be between 0-52")
        else:
            ur=0.143+(0.018*a)+(0.036*b)+(0.044*c)-(0.044*d)-(0.016*e)-(0.036*f)+(0.074*g)+(0.007*h)-(0.039*i)+(0.187*j)-(0.138*k)+(0.799*l)+(0.002*m)+(0.006*n)
            print("Predicted Unemplyment Rate is",ur)
        op=int(input("press 1 if you want to repeat and other buttons to exit: "))
        if(op==1):
            reg()
        else:
            exit
    reg()
```