Assignment – 2

Task description (including task 1, 2 and 3)

Steps for pre-processing of data

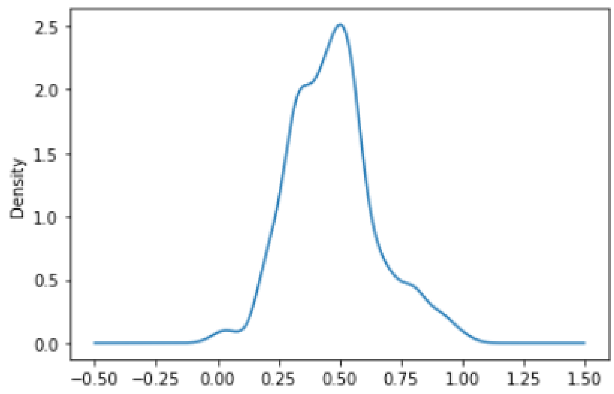
1. Imported Traffic data. Code.pdf cell [2]
2. Imported CO data and grouped CO data for each day and did average per day for CO data. Reference : Code.pdf cell[4]
3. Merged data for both Tables as shown in snapshot below. Reference: Code.pdf cell[5]
4. Normalized data for columns ADT, AADT and Average as shown below. Reference: Code.pdf cell[6]

Descriptive analysis of data

1. Below shows normal distribution for column Average, which shows after normalization data lies between 0 and 1

Below shows Normal-Distribution of column Average after normalization

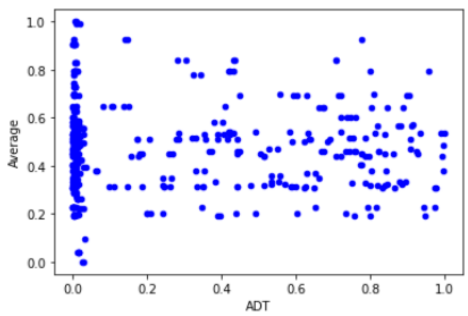
```
In [7]:  MergedData['Average'].plot.kde()
Out[7]: <AxesSubplot:ylabel='Density'>
```



2. Next, I visualized scatter plot between ADT vs Average and AADT vs Average which shows data is non-linear and because of that decision tree is the best model to classify non-linear data.

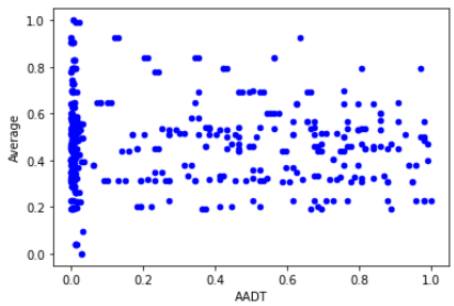
ADT and Average on compare is non-linear

```
In [8]:  MergedData.plot.scatter(x='ADT',
                                 y='Average'
                                 ,c='Blue')
Out[8]: <AxesSubplot:xlabel='ADT', ylabel='Average'>
```



AADT and Average on compare is non-linear

```
In [9]:  MergedData.plot.scatter(x='AADT',
                                 y='Average'
                                 ,c='Blue')
Out[9]: <AxesSubplot:xlabel='AADT', ylabel='Average'>
```

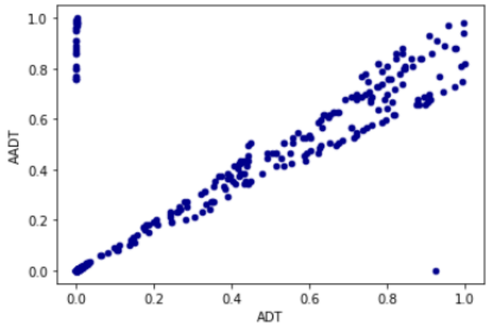


3. Visualized scatter plot for ADT and AADT which shoed and interesting result that both are linear and using both as features will not improve Model accuracy hence, we can skip AADT from features list.

ADT and AADT on compare is linear, so we will consider only ADT into feature and will remove AADT as both are linear and taking both will not improve our model

In [13]:
```
MergedData.plot.scatter(x='ADT',
                        y='AADT',
                        c='DarkBlue')
```

Out[13]: <AxesSubplot:xlabel='ADT', ylabel='AADT'>



Summary visualization of our final Merged data

Below shows Summary Visualization of Data

In [14]:
```
MergedData.drop(['AADT','Date'],axis=1,inplace=False).describe(include = 'all')
```

Out[14]:

	HIGHWAY	SECTION	SECTION LENGTH	ADT	Average
count	586.000000	586.000000	586.000000	586.000000	586.000000
mean	148.576792	52.779863	0.350969	0.197231	0.471287
std	125.552938	56.718809	0.194030	0.306120	0.174435
min	1.000000	1.000000	0.000000	0.000000	0.000000
25%	7.000000	17.000000	0.192495	0.001846	0.335788
50%	104.000000	30.000000	0.343738	0.007201	0.462445
75%	245.000000	60.000000	0.479532	0.382794	0.559647
max	376.000000	270.000000	1.000000	1.000000	1.000000

Reason for removing colums from our Final Data

Date & Time- removed because our model cannot understand date and time values.
Pollutant- removed because in data it is unique as shown below.
Unit- removed because in data it is unique as shown below.

Performing PCA for dimensionality reduction

- What Principal Component Analysis means that we are tyring to do in a linear language is to fit a hyperplane that best cuts a n-dimensional plane in 2 halves.
- Or you can say a best line to represent a data.
- In this we assume that we have data in Gaussian distribution.
- i.e we will build an orthogonal plane based on the assumption that these are normally distributed and will build a line that is orthogonal to it.
- And the line will be the good classifier if the data is normally distributed. Like we will choose a dimension with high variance.

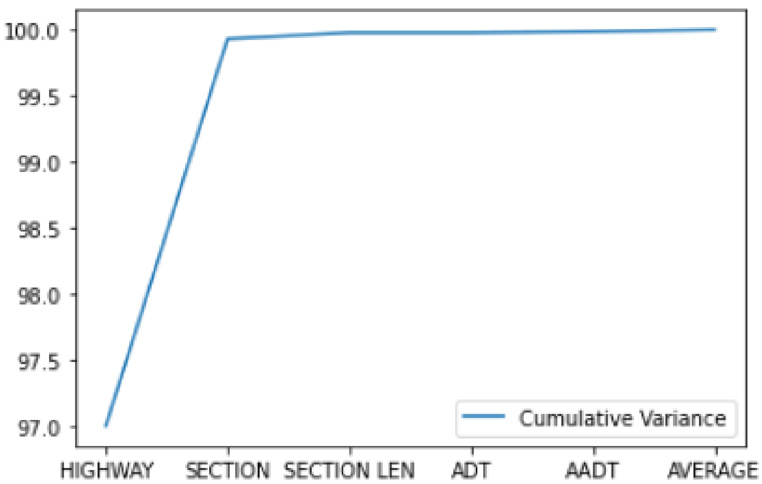
Steps:

1. In Code.pdf Cell[17] we created NumPy arrays of all features and target that we have in our dataset.

2. In Code.pdf Cell[20] we create Standard Scaler so that It transforms the data in such a manner that it has mean as 0 and standard deviation as 1.
3. In Code.pdf Cell[21] we find the covariance matrix for the dataset which did not gave us clear result like how each feature change with Average CO Data. So we cannot comment here that which feature we should take for hypothesis.
4. In Code.pdf Cell [22][23] we found Eigen vectors and Eigen values for the dataset, the characteristics of them is that they do not change their direction when a linear regression is performed on them. These values help in representing matrix in simplified form.
5. Then in Code.pdf Cell[25] [26] we find variance and cumulative variance which helps us understand which dimension will give us highest variance if we perform regression.
6. As from snapshot below we can see that the cumulative variance for "Highway" is 96.99 which means that if we take Highway as only our feature to perform hypothesis it will be able to fit a plane that will pass from 96% of total data.

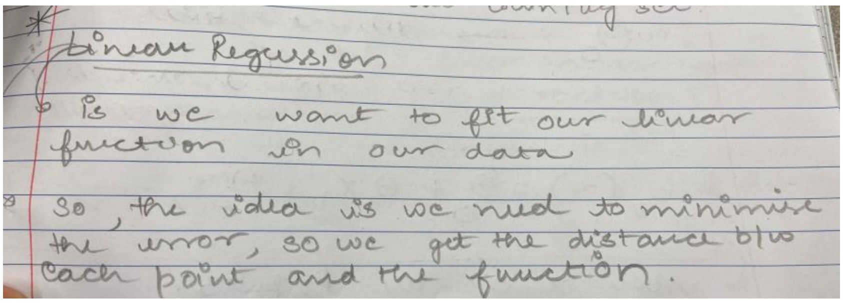
In [25]:
```
cum_variance = np.cumsum(varaiance_explained)
print(cum_variance)
```
```
[ 96.99880574  99.93094741  99.97702633  99.97702633  99.98560738
 100.        ]
```

In [26]:
```
plot_data = pd.DataFrame({
    'Cumulative Variance':cum_variance
}, index = ['HIGHWAY','SECTION','SECTION LEN','ADT','AADT','AVERAGE'])
plot_data.plot.line()
```
Out[26]: <AxesSubplot:>



2(i) How Linear Regression Algorithm Works?

I have included pictures in the explanation because I was having hard time representing formulas in word document.



and then we compute error on top of that and find the minimum error of

So in linear straight line equation used to be

$$y = m x + c$$

$m \rightarrow$ slope

$c \rightarrow$ intercept on y-axis

Same way, we assume that there is a linear relation between the O/P variable and I/P features:

$\theta_1, \theta_2, \theta_3 \text{---} \theta_n$ } defines slope of line

$\theta_0 \longrightarrow$ defines bias

ie intercept on y-axis.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \text{---} \theta_n x_n$$

where x is the inputs.

loss function | error funct

define the cost | loss function as the sum of squared error of predictions on training data

$$\boxed{J(\theta) = \frac{1}{2} \sum_{k=1}^{m} \left(h_\theta(x^k) - y^k \right)^2}$$

when we apply derivative later it will make our calculations easier

hypothesis i.e our linear function

correct data

So what happen when we apply a derivate on a function we get the (tangent (ie direction)

→ slope ie (+) → up

(−) → down

update Rule :-

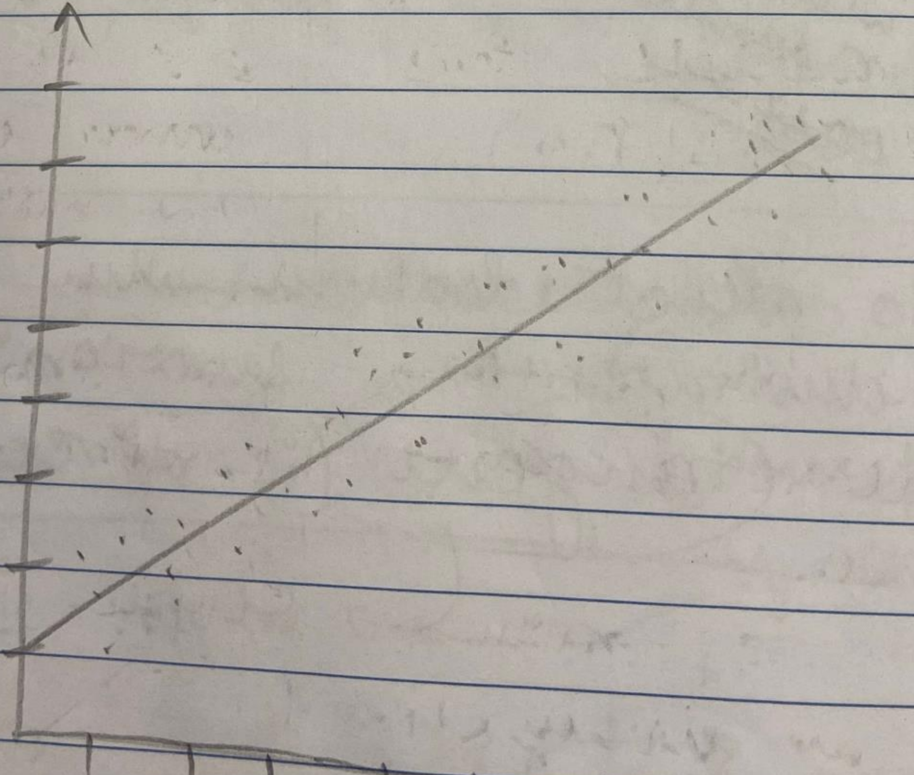
$$\theta_j = \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$$

represent the step to the function.

derivative ie direction

so continously we change the θ (ie slope) to find the minimum error

→ Datapoints regression



as long as we iterate and process the derivative and this function will fit our data to find minimum error

So, we have 2 ways to update our values

Batch Gradient Descent

- when we pass all our input data before we update these θ's (slopes)
- slow, but guarantees convergence to global minimum.
- use entire training set in every step

$$\boxed{\theta_j = \theta_j - \alpha \underbrace{\sum_{k=1}^{m}}_{\text{all data}} (\theta^T x^k - y^k) x_j^k}$$

Stochastic Gradient Descent

- one per time based on each instance.
- ∴ it is easy to converge, but it will not gurrantte converge, but it'll get close to it.

```
for i = 1 to m {
    θ_j = θ_j − α(θ^T x^k − y^k) x_j^k
}
```

2(ii) Most relevant features for prediction.

As after doing PCA which is not used for feature selection but dimensionality reduction though it gave us result like with "Highway" we can linearize 97% of data so Highway can be the hidden feature which did not showed much collinearity with Target (Average CO Level) but can be very good feature for predicting hypothesis.

2(iii) Evaluation matrix:

We choose R2 score for evaluation of linear regression, and it showed like our model fails to fit a line using "Highway" as a feature in contrary to in PCA reduction we found out that "Highway" can be the most relevant feature for prediction. That doesn't mean that the feature is not relevant for predicting result but what happed is our data in Highway is not numerical data rather it is a categorical data. Which is not suitable for regression that is the reason for very less R2 score. Less R2 score doesn't meant that the model is incorrect, but it means that while predicting future values our model will suffer.

```
In [39]:  y_trainpred = model.predict(X_Train)

In [40]:  y_prediction = model.predict(X_Test)

In [41]:  scoretrain = r2_score(Y_Train,y_trainpred)
          scoretrain

Out[41]:  0.0008074024040841676

In [42]:  score = r2_score(Y_Test,y_prediction)
          score

Out[42]:  -0.008672021779141836
```

2(iv) Insights and Conclusion

So as in regression model we found out that data that we have is categorical rather than numerical. What we meant by categorical is that for example: we have highway 1 and highway 374, we cannot say that distance between these two highways is 374 -1 = 373. Because it is possible that these two highways might be neighbours in the n-dimensional space. So, we cannot say that both highways that we took as an example are far. This is called categorical data.

Solution to this is if we increase dimensions of our data by creating this categorical data into dummy or indicator variable that can solve our problem. So, we tried changing categorical data for "Highway" and "Section" to indicator variables and then perform regression for all dimensions.

So, after creating dummy variables we had around 155 column or dimensions as features. As shown in below figure.

```
In [46]: testvar = ['HIGHWAY','SECTION']
         MergedData_Dummies =  pd.get_dummies(MergedData,columns=testvar)
```

```
In [47]: MergedData_Dummies
```

Out[47]:

	Date	SECTION LENGTH	ADT	AADT	Average	HIGHWAY_1	HIGHWAY_2	HIGHWAY_3	HIGHWAY_4	HIGHWAY_6	...	SECTION_210	SECTION_22(
0	09/09/2019	0.209552	0.001571	0.001446	0.542614	1	0	0	0	0	...	0	(
1	06/17/2019	0.360624	0.003282	0.002872	0.698085	1	0	0	0	0	...	0	(
2	06/17/2019	0.360624	0.002948	0.002573	0.698085	1	0	0	0	0	...	0	(
3	06/17/2019	0.360624	0.001932	0.001658	0.698085	1	0	0	0	0	...	0	(
4	09/09/2019	0.360624	0.005191	0.004570	0.542614	1	0	0	0	0	...	0	(
...	...	...	...	...	...	...	...	...	...	...	...	...	..
581	06/27/2019	0.323099	0.241446	0.221436	0.201447	0	0	0	0	0	...	0	(
582	06/27/2019	0.528265	0.489938	0.443883	0.201447	0	0	0	0	0	...	0	(
583	06/04/2019	0.267057	0.000407	0.000324	0.541973	0	0	0	0	0	...	0	(
584	05/28/2019	0.280702	0.001218	0.001092	0.367420	0	0	0	0	0	...	0	(
585	05/28/2019	0.222222	0.003895	0.003620	0.367420	0	0	0	0	0	...	0	(

586 rows × 155 columns

After doing regression again with all these features as we removed the categorical the regression should improve so the results that we got it below:

So, for training set the R2 score was very good as compared to before that we got. R2 score for training set = 71.69 %

```
In [61]: y_trainpred = model.predict(X_Train)
```

```
In [62]: y_prediction = model.predict(X_Test)
```

```
In [63]: scoretrain = r2_score(Y_Train,y_trainpred)
         scoretrain

Out[63]: 0.7169531837632088
```

But we got very less score for testing set because of increasing dimensions of the features we end up overfitting our data and made our model Highly Biased, due to which our model failed while predicting test results.

```
In [64]: score = r2_score(Y_Test,y_prediction)
         score

Out[64]: -3.626032695825652e+23
```

So, we conclude that, the dataset we have for finding the best hypothesis is categorical and because of that Linear regression is not the good hypothesis. As with only one feature as "Highway" model complexity was very less, and it was less Biased and creating dummy variable made our model too complexed and highly Biased.

3(i) How does K-means algorithm works?

Partitioning Clustering – k means

Step -1.

Define k- the number of clusters

Step -2.

Choose k points randomly as cluster centres

Step - 3.

For any instance, assign it to the cluster whose centre is the closest

Step-4. If no cluster gets modified, we have to stop

Step - 5. Make centroids (“instances” created by taking means of all instances in the cluster) new clusters centres

Step - 6.

go to 3

3(ii) What clusters represent in dataset?

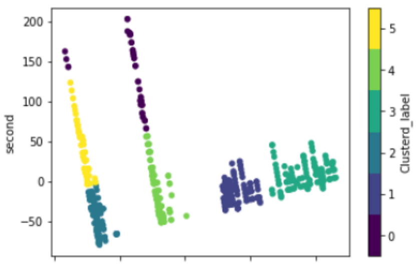
After doing clustering, we found out like we have n_clusters = 6 for maximum silhouette score and found that most clusters were cluttered around “Highway” and then “Section Length” and last with “ADT”.

Clusters	SectionLength Average	ADT Average
Cluster-0	0.36	0.16
Cluster-1	0.4	0.28
Cluster-2	0.22	0.02
Cluster-3	0.35	0.19
Cluster-4	0.42	0.32
Cluster-5	0.27	0.03

All these average values are calculated after exporting dataset as csv after attaching Cluster as one column and calculating the average. From this data we found out that Section Length and ADT along with Highway (we cannot average Highway as we did not normalize it and it’s a categorical data) are the columns on which clusters are formed.

In [84]:
```
#graph showing clusters created for highest silhouette value
df.plot.scatter(x='first', y='second', c = 'Clusterd_label', colormap='viridis')
```

Out[84]: <AxesSubplot:xlabel='first', ylabel='second'>



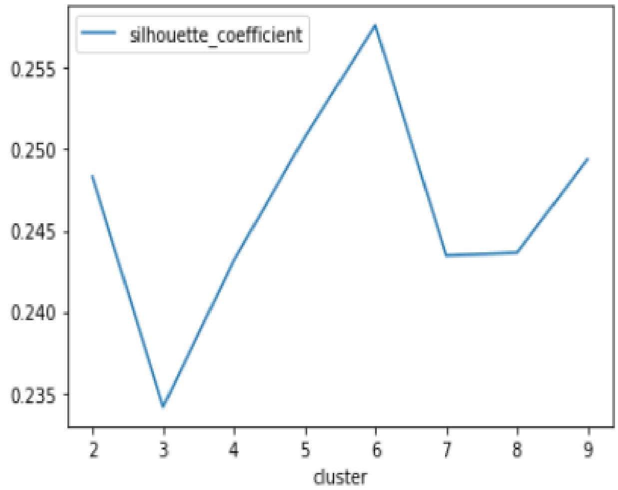
Applying Decision tree classifier after clustering

3(iii) Silhouette Measure and information about dataset?

Silhouette Value is measure of how similar an object is to its own cluster which is called cohesion as compared to other clusters. Silhouette values ranges from -1 to 1 where higher the value means the object is matched to its own cluster and lower the value means it is matched or closed to another cluster.

In [76]:
```
#plotting graph for all silhouette values for and finding for how many clusters it is the greatest
df.plot(x='cluster', y='silhouette_coefficient')
```

Out[76]: <AxesSubplot:xlabel='cluster'>



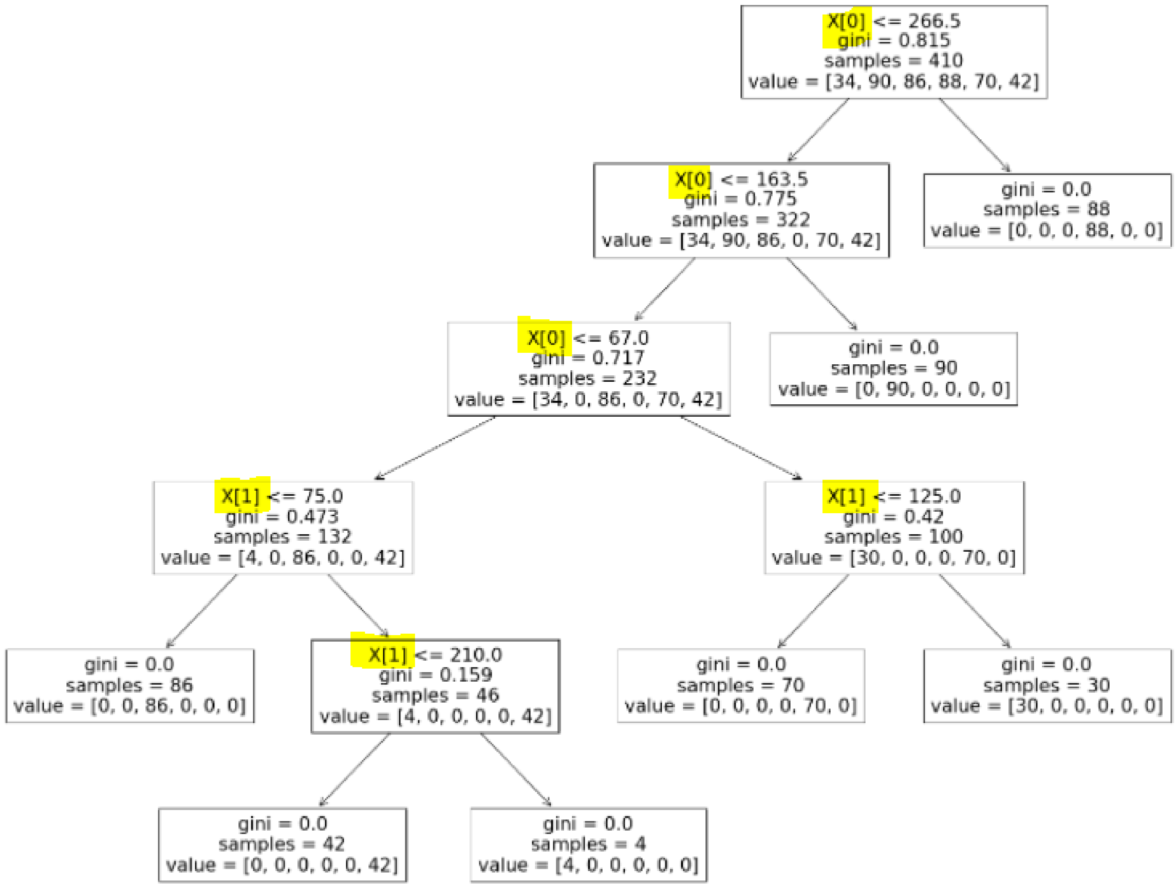
For our dataset we found like if we have six clusters that the silhouette value is maximum as compared to other numbers. The information that it gave for our dataset is that as we know that feature “ADT” and “AADT” are collinear with each other and if we take both while doing clustering, we will end up giving more weight to same datapoints which will make good clusters (for example only 2 clusters) but will not be helpful for our hypothesis and is not a good practice while clustering.

4(i) what information each model can provide about the dataset?

As we performed two models after performing clustering i.e. Decision Tree and Naïve Bayes, we found out like because of clusters that we made our model is able to learn efficient as for both model it performed exceptionally well.

From Decision Tree classifier we can see that most decision node as shown in below picture are from “Highway” which showed that Highway was one of the main features for our dataset and after that Section Length. And earlier with PCA also we found that Highway can be the best feature for the hypothesis of this data which we proved while creating Decision tree as most decision nodes are of Highway.

However, because of 6 clusters, we can easily see that our Hypothesis is overfitted and is Highly Biased. Which is less for Naïve Bayes as compared to Decision Tree



4(ii) which evaluation metric you used and why?

We used Accuracy because for both classifiers we saw that Testing data worked very well. As we can easily see that our data is overfitted as we got 6 clusters hence Accuracy for Both test and train is very high close to 100% in case of Decision Tree. Which shows that model of DT is highly complexed. Though because of clustering it learned well as it was able to predict well on testing set also.

Decision Tree:

```
Out[99]: 1.0

In [100]: print ("Accuracy of testdata: \n",accuracy_score(y_test,y_pred)*100)

Accuracy of testdata: 
 100.0
```

Naïve Bayes:

```
In [110]: y_pred = NBclassifier.predict(X_test)

In [111]: print ("Accuracy of testdata: \n",accuracy_score(y_test,y_pred)*100)

Accuracy of testdata: 
 98.29545454545455
```

4(iii) which model provide high score? Why do you think this happened? Perform statistical significance testing for both NB and DT?

As we saw that, both the models performed very well but DT score was around 100% even for testing test. The reason this happened because we got a lot of clusters i.e. clusters = 6 while clustering, because of that DT was having high depth as it overfitted the data. Whereas in Naïve Bayes we do hypothesis on the basic of probability which is not much affected due to high number of clusters.

Statistical hypothesis testing

We will perform 10 times 10-fold cross validation and then do student t-test on the difference of accuracy receives from both the hypothesis. So as performance of model is measured using test data i.e. why we are doing K-fold cross validation as in that we never know what are testing set would be.

Statistical test are used to compare the performance of Machine Learning models, ie, if null hypothesis is rejected that means difference in scores is significant.

Null – Hypothesis is this test is that there is no difference between two model that we are considering and both models perform same.

Alternative hypothesis assumes that two applied Model will perform differently.

This we can find by finding P-value from after doing student's t-test, and **a P-value smaller than considered significant (5%) rejects the null hypothesis and favours Alternative hypothesis. And if P-value is more than considered significant i.e we fail to reject null hypothesis and both the Models will perform same.**

In our case, we found out as from below screenshot that we get P-value of around 27% which is greater than the considered significant (5%) **hence we fail to reject the null hypothesis, and both models will perform same** as per this data. And for more unseen data might show the same trend.

Statistical test for best hypothesis

Performing 10X10-Fold cross validation and student t-test on accuaracy obtained from cross validations

In [113]:
```
NBAccuracy_List = []
DTAccuracy_List = []
#Length of datapoints for training
n1_train = []
#Length of datapoints for testing
n2_test = []
kf = KFold(n_splits=10)
for i in range(10):
    for train_index, test_index in kf.split(X):
        Xtrain, Xtest = X[train_index], X[test_index]
        ytrain, ytest = Y[train_index], Y[test_index]

        n1_train.append(len(ytrain))
        n2_test.append(len(ytest))

        clf_tree.fit(Xtrain, ytrain)
        NBclassifier.fit(Xtrain, ytrain)

        ypredDT = clf_tree.predict(Xtest)
        ypredNB = NBclassifier.predict(Xtest)

        DTAccuracy = accuracy_score(ytest, ypredDT)
        NBAccuracy = accuracy_score(ytest, ypredNB)
        DTAccuracy_List.append(DTAccuracy)
        NBAccuracy_List .append(NBAccuracy)
```

In [114]:
```
Differences_list = [y -x for y, x in zip(DTAccuracy_List,NBAccuracy_List)]
```

In [115]:
```
#mean of differences
d_bar = np.mean(Differences_list)
d_bar
```

Out[115]: 0.06708357685563995

In [116]:
```
#variance of differences
sigma2 = np.var(Differences_list, ddof=1)
sigma2
```

Out[116]: 0.031042436742399274

In [117]:
```
#no of datapoints used for training
n1 = np.median(n1_train)
n1
```

Out[117]: 527.0

In [118]:
```
#no of datapoints used for testing
n2 = np.median(n2_test)
n2
```
Out[118]: 59.0

In [119]:
```
#compute Total number of datapoints
n = 10 * 10
n
```
Out[119]: 100

In [120]:
```
#computing modified variance
sigma2_mod = sigma2*(1/n + n2/n1)
sigma2_mod
```
Out[120]: 0.0037857635852637595

In [121]:
```
#computing t-static
t_static = d_bar / np.sqrt(sigma2_mod)
t_static
```
Out[121]: 1.0902835664262074

In [122]:
```
#computing p-value
Pvalue = ((1 - t.cdf(np.abs(t_static),n-1))*2)
Pvalue
```
Out[122]: 0.2782348766691636

4(iv) The patterns found by the supervised models are the same as in the one presented in the clustering?

Yes, the patterns found in clustering are same in Hypothesis also as we saw while evaluating our clusters that Highway and Section Length were effective for the clusters. The same way in DT we saw most decision nodes are of Highway and Sectional Length. And In PCA while starting also we predicted that "Highway" can be the best feature for our hypothesis. We can say, yes, the patterns found in clustering and supervised models are same.

Summary And Conclusions.

- After performing the PCA we found that "Highway", can be the most relevant feature but after doing regression we found that data in this feature is categorical and is not well suited for regression and our model failed to learn as R2 score were very less. And we can say our model went underfitting and did not learn anything and Bias was very low.
- As we found categorical data in dataset, we created indicator variables which showed us good R2 score for training set, but our model failed on testing set as model was highly Biased because we included so many dimensions in order to remove categorical data.
- We conclude that Linear regression is not the best Hypothesis for our dataset as most of our data is categorical.
- We found that n_clusters = 6 showed us best silhouette value and our model learned well after clustering.
- We found that DT classifier showed around 100% accuracy on test set because, we got a lot of clusters and our DT overfitted the data. We can say model was highly biased.
- We found that Naïve Bayes language worked well on our dataset and performed well on Testing data.
- Statistical Hypothesis Testing showed us that P-value was greater than considered significant and we failed to reject null hypothesis. And both models will perform same.

References:

1. https://pandas.pydata.org/docs/

2. https://scikit-learn.org/stable/

3. https://numpy.org/doc

4. Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learningalgorithms.Neural computation, 10(7):1895–1923, 1998

https://direct.mit.edu/neco/article-abstract/10/7/1895/6224/Approximate-Statistical-Tests-for-Comparing?redirectedFrom=PDF

In [1]:
```
import pandas as pd
import numpy as np
from numpy import cov
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold
from scipy.stats import t
import datetime
```

In [2]:
```
TraffData = pd.read_csv("cleaned_traffic_data.csv",sep=',',parse_dates=['Date'])
TraffData['Date'] = TraffData['Date'].dt.strftime('%m/%d/%Y')
TraffData
```

Out[2]:

	Date	HIGHWAY	SECTION	SECTION LENGTH	ADT	AADT
0	09/09/2019	1	47	4.50	2.566	2.430
1	06/17/2019	1	50	7.60	4.266	3.840
2	06/17/2019	1	50	7.60	3.934	3.545
3	06/17/2019	1	50	7.60	2.924	2.640
4	09/09/2019	1	50	7.60	6.164	5.520
...	...	...	...	...	...	...
581	06/27/2019	374	28	6.83	241.000	220.000
582	06/27/2019	374	30	11.04	488.000	440.000
583	06/04/2019	376	10	5.68	1.409	1.320
584	05/28/2019	376	20	5.96	2.215	2.080
585	05/28/2019	376	30	4.76	4.876	4.580

586 rows × 6 columns

In [3]:
```
AllCOData = pd.read_csv("Nova_Scotia_Provincial_Ambient_Carbon_Monoxide__CO__Hourly_Data_Halifax_Johnston.csv", sep =',')
AllCOData.head()
```

Out[3]:

	Date & Time	Pollutant	Unit	Station	Average
0	01/01/2019 12:00:00 AM	CO	ppm	Halifax Johnston	0.25
1	01/01/2019 01:00:00 AM	CO	ppm	Halifax Johnston	0.26
2	01/01/2019 02:00:00 AM	CO	ppm	Halifax Johnston	0.20
3	01/01/2019 03:00:00 AM	CO	ppm	Halifax Johnston	0.17
4	01/01/2019 04:00:00 AM	CO	ppm	Halifax Johnston	0.15

In [4]:
```
COData = AllCOData[['Date & Time','Average']]
COData.rename(columns=({'Date & Time':'Date'}),inplace=True,)
COData[['Date','Time','AP']] =  COData.Date.str.split(" ", expand = True)
COData = COData[['Date','Average']]
COData = COData[COData['Date'].str.contains('[\d/]2019')]
COData = COData.groupby('Date').mean()
COData
```

```
C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-packages\pandas\core\frame.py:5039: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().rename(
C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-packages\pandas\core\frame.py:3641: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
```

Out[4]:

	Average
Date	
01/01/2019	0.146250
01/02/2019	0.152917
01/03/2019	0.198333
01/04/2019	0.178333
01/05/2019	0.197083
...	...
12/27/2019	0.127083
12/28/2019	0.116250
12/29/2019	0.106667
12/30/2019	0.096667
12/31/2019	0.071250

365 rows × 1 columns

In [5]:
```
MergedData = pd.merge(TraffData,COData,on='Date',how='left')
MergedData
```

Out[5]:

	Date	HIGHWAY	SECTION	SECTION LENGTH	ADT	AADT	Average
0	09/09/2019	1	47	4.50	2.566	2.430	0.122174
1	06/17/2019	1	50	7.60	4.266	3.840	0.144167
2	06/17/2019	1	50	7.60	3.934	3.545	0.144167
3	06/17/2019	1	50	7.60	2.924	2.640	0.144167
4	09/09/2019	1	50	7.60	6.164	5.520	0.122174
...	...	...	...	...	...	...	...
581	06/27/2019	374	28	6.83	241.000	220.000	0.073913

	Date	HIGHWAY	SECTION	SECTION LENGTH	ADT	AADT	Average
582	06/27/2019	374	30	11.04	488.000	440.000	0.073913
583	06/04/2019	376	10	5.68	1.409	1.320	0.122083
584	05/28/2019	376	20	5.96	2.215	2.080	0.097391
585	05/28/2019	376	30	4.76	4.876	4.580	0.097391

586 rows × 7 columns

In [6]:
```
cols_to_norm = ['Average','ADT','AADT','SECTION LENGTH']
print(cols_to_norm)
MergedData[cols_to_norm] = MergedData[cols_to_norm].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
MergedData
```

```
['Average', 'ADT', 'AADT', 'SECTION LENGTH']
```

Out[6]:

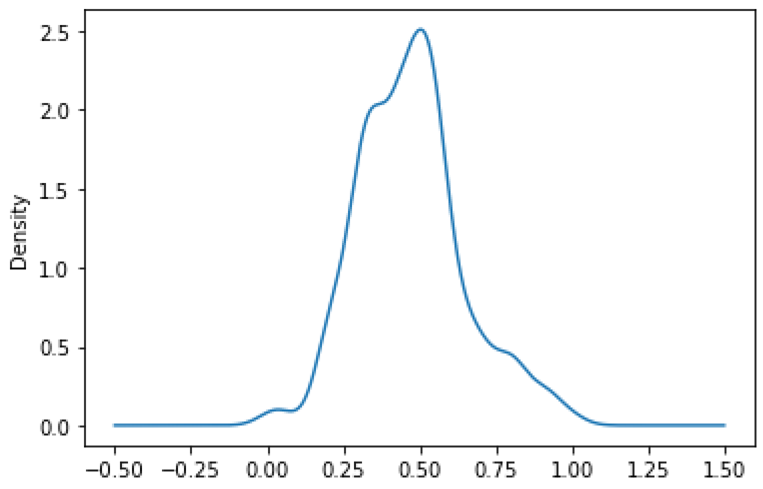
	Date	HIGHWAY	SECTION	SECTION LENGTH	ADT	AADT	Average
0	09/09/2019	1	47	0.209552	0.001571	0.001446	0.542614
1	06/17/2019	1	50	0.360624	0.003282	0.002872	0.698085
2	06/17/2019	1	50	0.360624	0.002948	0.002573	0.698085
3	06/17/2019	1	50	0.360624	0.001932	0.001658	0.698085
4	09/09/2019	1	50	0.360624	0.005191	0.004570	0.542614
...	...	...	...	...	...	...	...
581	06/27/2019	374	28	0.323099	0.241446	0.221436	0.201447
582	06/27/2019	374	30	0.528265	0.489938	0.443883	0.201447
583	06/04/2019	376	10	0.267057	0.000407	0.000324	0.541973
584	05/28/2019	376	20	0.280702	0.001218	0.001092	0.367420
585	05/28/2019	376	30	0.222222	0.003895	0.003620	0.367420

586 rows × 7 columns

Below shows Normal-Distribution of column Average after normalization

In [7]:
```
MergedData['Average'].plot.kde()
```

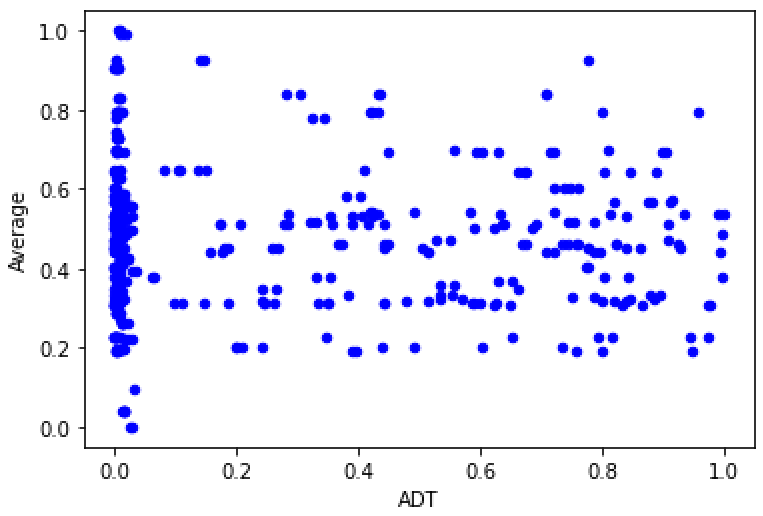
Out[7]: <AxesSubplot:ylabel='Density'>



ADT and Average on compare is non-linear

In [8]:
```
MergedData.plot.scatter(x='ADT',
                        y='Average'
                        ,c='Blue')
```

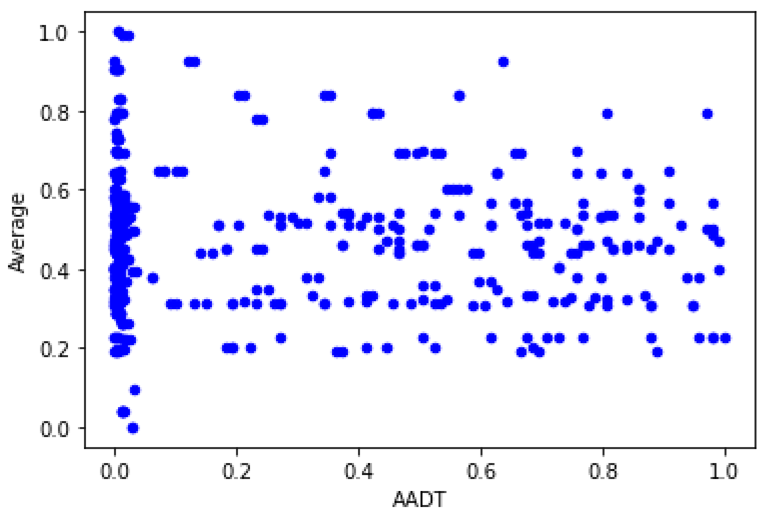
Out[8]: <AxesSubplot:xlabel='ADT', ylabel='Average'>



AADT and Average on compare is non-linear

In [9]:
```
MergedData.plot.scatter(x='AADT',
                        y='Average'
                        ,c='Blue')
```

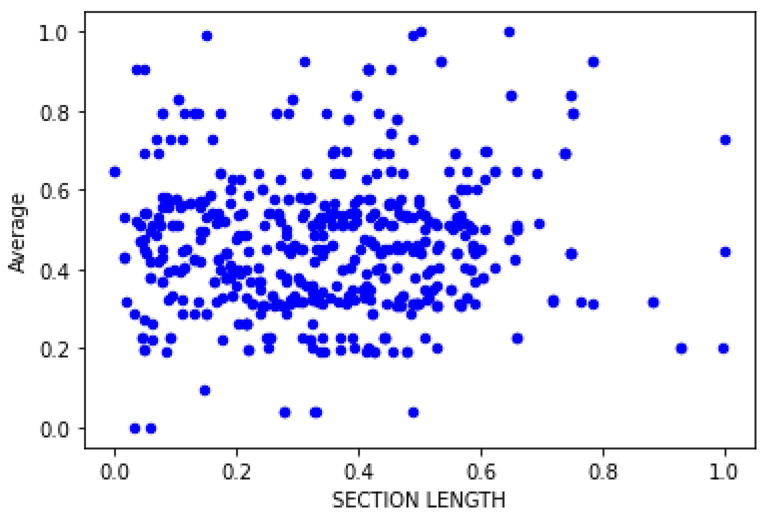
Out[9]: <AxesSubplot:xlabel='AADT', ylabel='Average'>



Section Length and Average on comapre are much better than other features

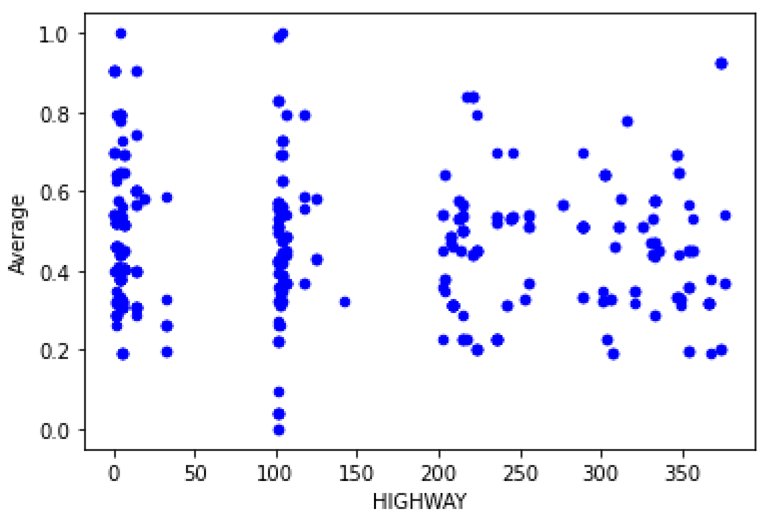
In [10]:
```
MergedData.plot.scatter(x='SECTION LENGTH',
                        y='Average'
                        ,c='Blue')
```

Out[10]: <AxesSubplot:xlabel='SECTION LENGTH', ylabel='Average'>



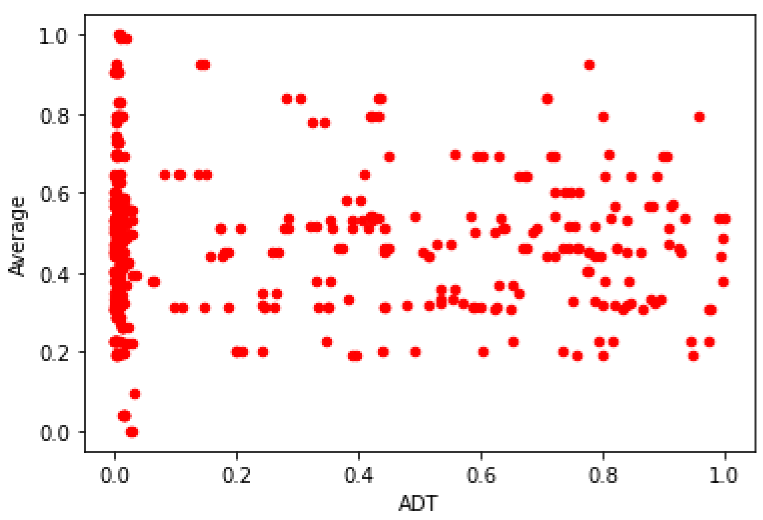
In [11]:
```
MergedData.plot.scatter(x='HIGHWAY',
                        y='Average'
                        ,c='Blue')
```

Out[11]: <AxesSubplot:xlabel='HIGHWAY', ylabel='Average'>



In [12]:
```
MergedData.plot.scatter(x='ADT',
                        y='Average'
                        ,c='Red')
```

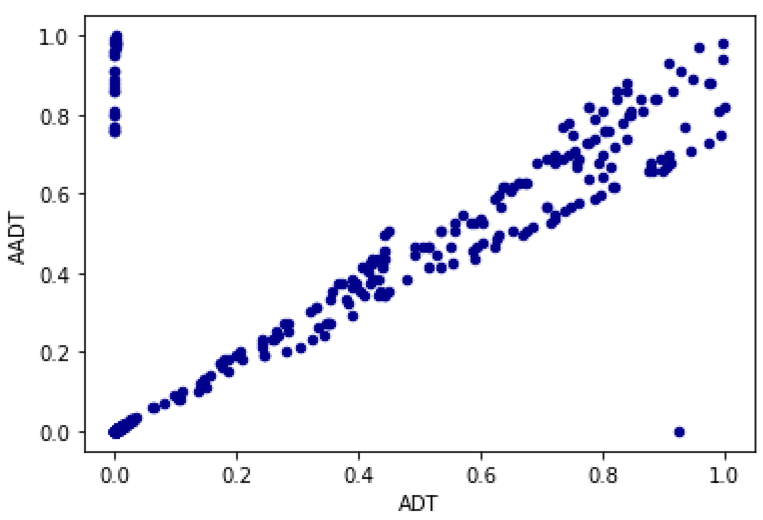
Out[12]: <AxesSubplot:xlabel='ADT', ylabel='Average'>



ADT and AADT on compare is linear, so we will consider only ADT into feature and will remove AADT as both are linear and taking both will not improve our model

In [13]:
```
MergedData.plot.scatter(x='ADT',
                y='AADT',
                c='DarkBlue')
```

Out[13]: <AxesSubplot:xlabel='ADT', ylabel='AADT'>



Below shows Summary Visualization of Data

In [14]:
```
MergedData.drop(['AADT','Date'],axis=1,inplace=False).describe(include = 'all')
```

Out[14]:

	HIGHWAY	SECTION	SECTION LENGTH	ADT	Average
count	586.000000	586.000000	586.000000	586.000000	586.000000
mean	148.576792	52.779863	0.350969	0.197231	0.471287
std	125.552938	56.718809	0.194030	0.306120	0.174435
min	1.000000	1.000000	0.000000	0.000000	0.000000
25%	7.000000	17.000000	0.192495	0.001846	0.335788
50%	104.000000	30.000000	0.343738	0.007201	0.462445
75%	245.000000	60.000000	0.479532	0.382794	0.559647
max	376.000000	270.000000	1.000000	1.000000	1.000000

Reason for removing colums from our Final Data

Date & Time- removed because our model cannot understand date and time values.
Pollutant- removed because in data it is unique as shown below.
Unit- removed because in data it is unique as shown below.

In [15]:
```
AllCOData.describe(include='all')
```

Out[15]:

	Date & Time	Pollutant	Unit	Station	Average
count	236687	236687	236687	236687	205269.000000
unique	236687	1	1	2	NaN
top	01/01/2019 12:00:00 AM	CO	ppm	Halifax	NaN
freq	1	236687	236687	210384	NaN
mean	NaN	NaN	NaN	NaN	0.358628

	Date & Time	Pollutant	Unit	Station	Average
std	NaN	NaN	NaN	NaN	0.322583
min	NaN	NaN	NaN	NaN	0.000000
25%	NaN	NaN	NaN	NaN	0.130000
50%	NaN	NaN	NaN	NaN	0.270000
75%	NaN	NaN	NaN	NaN	0.500000
max	NaN	NaN	NaN	NaN	11.020000

We removed station from final data because for year 2019 its unique as shown below

In [16]:
```
AllCOData[(AllCOData["Date & Time"]>='01/01/2019 12:00:00 AM' )& (AllCOData["Date & Time"]<'01/01/2020 12:00:00 AM')]['Station'].nunique()
```

Out[16]:
```
1
```

Doing PCA to reduce dimensions to find which features might fit a strait line best like if our hypothesis can divide dataset ito two halves

In [17]:
```
A = MergedData['HIGHWAY']
B = MergedData['SECTION']
C = MergedData['SECTION LENGTH']
D = MergedData['ADT']
E = MergedData['AADT']
F = MergedData['Average']
```

In [18]:
```
data = np.array([A,B,C,D,E,F])
```

In [19]:
```
scaler = StandardScaler()
```

In [20]:
```
scaled_data = scaler.fit_transform(data)
```

In [21]:
```
covMatrix = np.cov(scaled_data)
covMatrix
```

Out[21]:
```
array([[ 1.11997266e+00, -1.12222851e+00,  6.86251615e-05,
         2.61277826e-03,  3.25733744e-03, -3.68289482e-03],
       [-1.12222851e+00,  1.21911336e+00, -2.42707901e-02,
        -2.69365950e-02, -2.67250679e-02, -1.89523933e-02],
       [ 6.86251615e-05, -2.42707901e-02,  6.47813361e-03,
         5.98505460e-03,  5.79902082e-03,  5.93995590e-03],
       [ 2.61277826e-03, -2.69365950e-02,  5.98505460e-03,
         6.68165492e-03,  6.13918359e-03,  5.51792363e-03],
       [ 3.25733744e-03, -2.67250679e-02,  5.79902082e-03,
         6.13918359e-03,  6.26397198e-03,  5.26555408e-03],
       [-3.68289482e-03, -1.89523933e-02,  5.93995590e-03,
         5.51792363e-03,  5.26555408e-03,  5.91185454e-03]])
```

In [22]:
```
eigen_values, eigen_vector = np.linalg.eig(covMatrix)
```

In [23]:
```
print(" Eigen Vector \n:", eigen_vector,"\n")
print(" Eigen Values \n:", eigen_values,"\n")
```

```
 Eigen Vector
: [[ 6.91072213e-01 -5.96403293e-01 -6.58452498e-04  4.08248290e-01
  -6.87460198e-03  2.81910577e-03]
 [-7.22604455e-01 -5.57741050e-01 -5.20835482e-03  4.08248290e-01
  -8.35432365e-03  2.03330436e-03]
 [ 7.75080320e-03  2.95459386e-01  3.82518521e-01  4.08248290e-01
  -7.39861897e-01 -2.28606570e-01]
 [ 9.35900838e-03  2.95590540e-01 -4.78524645e-01  4.08248290e-01
  -1.22872189e-01  7.08370358e-01]
 [ 9.48371280e-03  2.84737433e-01 -5.05605149e-01  4.08248290e-01
   2.77767024e-01 -6.47593159e-01]
 [ 4.93871686e-03  2.78356984e-01  6.07478081e-01  4.08248290e-01
   6.00195988e-01  1.62976961e-01]]

 Eigen Values
: [2.29346075e+00 6.93281921e-02 1.08949987e-03 1.45384735e-16
 2.02892280e-04 3.40302237e-04]
```

In [24]:
```
varaiance_explained = []
for j in eigen_values:
    varaiance_explained.append((j/sum(eigen_values))*100)

print(varaiance_explained)
```

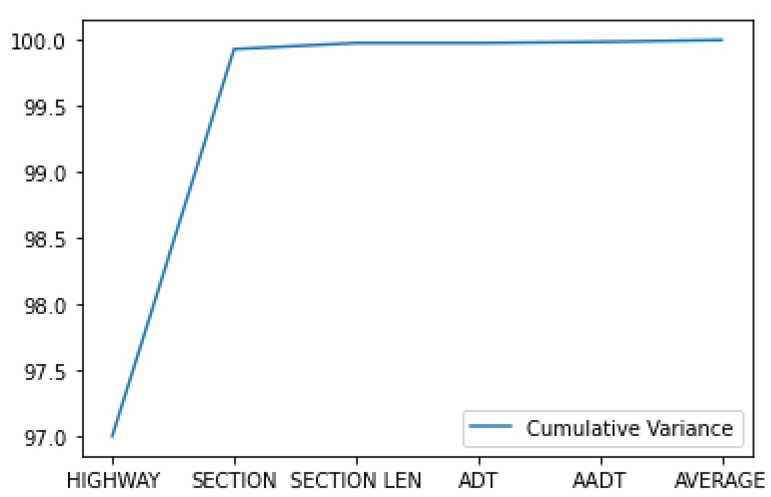
```
[96.99880573541189, 2.932141674303186, 0.046078916361717, 6.1488498048344344e-15, 0.008581053256413732, 0.01439262066677324]
```

In [25]:
```
cum_variance = np.cumsum(varaiance_explained)
print(cum_variance)
```

```
[ 96.99880574  99.93094741  99.97702633  99.97702633  99.98560738
 100.        ]
```

In [26]:
```
plot_data = pd.DataFrame({
    'Cumulative Variance':cum_variance
}, index = ['HIGHWAY','SECTION','SECTION LEN','ADT','AADT','AVERAGE'])
plot_data.plot.line()
```

Out[26]:
```
<AxesSubplot:>
```



In [27]:
```
proj_matrix = (eigen_vector.T[:][:1]).T
print(proj_matrix)
```

```
[[ 0.69107221]
 [-0.72260445]
 [ 0.0077508 ]
 [ 0.00935901]
 [ 0.00948371]
 [ 0.00493872]]
```

In [125…

```
MergedData_PCA = MergedData.iloc[:,1:7].values.dot(proj_matrix)
```

In [29]:

```
Merged_Data_PCA_DF = pd.DataFrame(data = MergedData_PCA, columns=["HIGHWAY"])
```

In [30]:

```
Merged_Data_PCA_DF
```

Out[30]:

	HIGHWAY
0	-33.267005
1	-35.432850
2	-35.432856
3	-35.432874
4	-35.433584
...	...
581	238.235942
582	236.796759
583	252.621861
584	245.395075
585	238.168626

586 rows × 1 columns

In [31]:

```
temp = [MergedData["Date"],Merged_Data_PCA_DF["HIGHWAY"],MergedData["Average"]]
headers = ["Date","HIGHWAY","Average"]
Final_DataFrame = pd.concat(temp, axis=1,keys=headers)
Final_DataFrame
```

Out[31]:

	Date	HIGHWAY	Average
0	09/09/2019	-33.267005	0.542614
1	06/17/2019	-35.432850	0.698085
2	06/17/2019	-35.432856	0.698085
3	06/17/2019	-35.432874	0.698085
4	09/09/2019	-35.433584	0.542614
...	...	...	...
581	06/27/2019	238.235942	0.201447
582	06/27/2019	236.796759	0.201447
583	06/04/2019	252.621861	0.541973
584	05/28/2019	245.395075	0.367420
585	05/28/2019	238.168626	0.367420

586 rows × 3 columns

In [32]:

```
split_date = datetime.datetime(2019,7,31)
split_date = pd.to_datetime(split_date)
```

In [33]:

```
Final_DataFrame["Date"] = pd.to_datetime(Final_DataFrame["Date"])
Final_DataFrame.dtypes
```

Out[33]:

```
Date       datetime64[ns]
HIGHWAY           float64
Average           float64
dtype: object
```

In [34]:

```
Train =  Final_DataFrame[(pd.to_datetime(Final_DataFrame["Date"]) < split_date)]
Train
```

Out[34]:

	Date	HIGHWAY	Average
1	2019-06-17	-35.432850	0.698085
2	2019-06-17	-35.432856	0.698085
3	2019-06-17	-35.432874	0.698085
5	2019-06-17	-35.432873	0.698085
18	2019-07-10	-34.742839	0.285714
...	...	...	...
581	2019-06-27	238.235942	0.201447
582	2019-06-27	236.796759	0.201447
583	2019-06-04	252.621861	0.541973
584	2019-05-28	245.395075	0.367420
585	2019-05-28	238.168626	0.367420

375 rows × 3 columns

In [35]:

```
Test = Final_DataFrame[(pd.to_datetime(Final_DataFrame["Date"]) > split_date)]
Test
```

Out[35]:

	Date	HIGHWAY	Average
0	2019-09-09	-33.267005	0.542614
4	2019-09-09	-35.433584	0.542614
6	2019-09-16	-42.661506	0.397644
7	2019-09-09	-46.271913	0.542614
8	2019-09-09	-49.887195	0.542614
...	...	...	...
556	2019-09-18	238.872792	0.356406
557	2019-10-03	237.422936	0.450663
558	2019-10-03	237.422765	0.450663
559	2019-10-03	233.804993	0.450663
560	2019-10-03	230.191939	0.450663

209 rows × 3 columns

In [36]:
```
X_Train = Train[["HIGHWAY"]]
Y_Train = Train[["Average"]]
X_Test = Test[["HIGHWAY"]]
Y_Test = Test[["Average"]]
```

In [37]:
```
model = linear_model.LinearRegression()
```

In [38]:
```
model.fit(X_Train,Y_Train)
```

Out[38]:
```
LinearRegression()
```

In [39]:
```
y_trainpred = model.predict(X_Train)
```

In [40]:
```
y_prediction = model.predict(X_Test)
```

In [41]:
```
scoretrain = r2_score(Y_Train,y_trainpred)
scoretrain
```

Out[41]:
```
0.0008074024040841676
```

In [42]:
```
score = r2_score(Y_Test,y_prediction)
score
```

Out[42]:
```
-0.008672021779141836
```

In [43]:
```
mean_squared_error(Y_Test,y_prediction)
```

Out[43]:
```
0.04742863137520921
```

Trying linear regression after Dumminsing the HIGHWAY and SECTION as it is a categorical data
It converts categorical data into dummy or indicator variables

In [44]:
```
MergedData['HIGHWAY'] = MergedData['HIGHWAY'].astype('category')
MergedData['SECTION'] = MergedData['SECTION'].astype('category')
```

In [45]:
```
MergedData.dtypes
```

Out[45]:
```
Date               object
HIGHWAY          category
SECTION          category
SECTION LENGTH    float64
ADT               float64
AADT              float64
Average           float64
dtype: object
```

In [46]:
```
testvar = ['HIGHWAY','SECTION']
MergedData_Dummies =  pd.get_dummies(MergedData,columns=testvar)
```

In [47]:
```
MergedData_Dummies
```

Out[47]:

	Date	SECTION LENGTH	ADT	AADT	Average	HIGHWAY_1	HIGHWAY_2	HIGHWAY_3	HIGHWAY_4	HIGHWAY_6	...	SECTION_210	SECTION_220	SECTION_223	SECTION_227	SECTION_230	SECTION_240	SECTION_250
0	09/09/2019	0.209552	0.001571	0.001446	0.542614	1	0	0	0	0	...	0	0	0	0	0	0	0
1	06/17/2019	0.360624	0.003282	0.002872	0.698085	1	0	0	0	0	...	0	0	0	0	0	0	0
2	06/17/2019	0.360624	0.002948	0.002573	0.698085	1	0	0	0	0	...	0	0	0	0	0	0	0
3	06/17/2019	0.360624	0.001932	0.001658	0.698085	1	0	0	0	0	...	0	0	0	0	0	0	0
4	09/09/2019	0.360624	0.005191	0.004570	0.542614	1	0	0	0	0	...	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
581	06/27/2019	0.323099	0.241446	0.221436	0.201447	0	0	0	0	0	...	0	0	0	0	0	0	0
582	06/27/2019	0.528265	0.489938	0.443883	0.201447	0	0	0	0	0	...	0	0	0	0	0	0	0
583	06/04/2019	0.267057	0.000407	0.000324	0.541973	0	0	0	0	0	...	0	0	0	0	0	0	0
584	05/28/2019	0.280702	0.001218	0.001092	0.367420	0	0	0	0	0	...	0	0	0	0	0	0	0
585	05/28/2019	0.222222	0.003895	0.003620	0.367420	0	0	0	0	0	...	0	0	0	0	0	0	0

586 rows × 155 columns

In [48]:
```
MergedDataNew = MergedData.copy()
MergedData_NoDate = MergedData.drop('Date',axis=1, inplace=False)
```

In [49]:
```
MergedData_NoDate
```

Out[49]:

	HIGHWAY	SECTION	SECTION LENGTH	ADT	AADT	Average
0	1	47	0.209552	0.001571	0.001446	0.542614
1	1	50	0.360624	0.003282	0.002872	0.698085
2	1	50	0.360624	0.002948	0.002573	0.698085
3	1	50	0.360624	0.001932	0.001658	0.698085
4	1	50	0.360624	0.005191	0.004570	0.542614
...	...	...	...	...	...	...
581	374	28	0.323099	0.241446	0.221436	0.201447
582	374	30	0.528265	0.489938	0.443883	0.201447
583	376	10	0.267057	0.000407	0.000324	0.541973
584	376	20	0.280702	0.001218	0.001092	0.367420
585	376	30	0.222222	0.003895	0.003620	0.367420

586 rows × 6 columns

In [50]:
```
MergedData_NoDate.corr()
```

Out[50]:

	SECTION LENGTH	ADT	AADT	Average
SECTION LENGTH	1.000000	0.280939	0.285422	0.077435
ADT	0.280939	1.000000	0.780457	-0.008396

	SECTION LENGTH	ADT	AADT	Average
AADT	0.285422	0.780457	1.000000	-0.049260
Average	0.077435	-0.008396	-0.049260	1.000000

In [51]:

```
split_date = datetime.datetime(2019,7,31)
split_date = pd.to_datetime(split_date)
```

In [52]:

```
scaler = StandardScaler()
temp_col = ['SECTION LENGTH','ADT','AADT'];
scaler.fit(MergedData_Dummies[temp_col])
MergedData_Dummies[temp_col] = scaler.transform(MergedData_Dummies[temp_col])
```

In [53]:

```
pd.set_option("max_columns",200)
pd.set_option("max_rows",200)
```

In [54]:

```
Train =  MergedData_Dummies[(pd.to_datetime(MergedData_Dummies["Date"]) <= split_date)]
Train
```

Out[54]:

	Date	SECTION LENGTH	ADT	AADT	Average	HIGHWAY_1	HIGHWAY_2	HIGHWAY_3	HIGHWAY_4	HIGHWAY_6	HIGHWAY_7	HIGHWAY_14	HIGHWAY_19	HIGHWAY_32	HIGHWAY_33	HIGHWAY_101	HIGHWAY_102
1	06/17/2019	0.049800	-0.634112	-0.687993	0.698085	1	0	0	0	0	0	0	0	0	0	0	0
2	06/17/2019	0.049800	-0.635204	-0.688965	0.698085	1	0	0	0	0	0	0	0	0	0	0	0
3	06/17/2019	0.049800	-0.638526	-0.691945	0.698085	1	0	0	0	0	0	0	0	0	0	0	0
5	06/17/2019	0.049800	-0.638329	-0.691781	0.698085	1	0	0	0	0	0	0	0	0	0	0	0
18	07/10/2019	0.693325	-0.631408	-0.687730	0.285714	0	1	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
581	06/27/2019	-0.143760	0.144560	0.023918	0.201447	0	0	0	0	0	0	0	0	0	0	0	0
582	06/27/2019	0.914536	0.957000	0.748475	0.201447	0	0	0	0	0	0	0	0	0	0	0	0
583	06/04/2019	-0.432843	-0.643509	-0.696293	0.541973	0	0	0	0	0	0	0	0	0	0	0	0
584	05/28/2019	-0.362458	-0.640858	-0.693790	0.367420	0	0	0	0	0	0	0	0	0	0	0	0
585	05/28/2019	-0.664110	-0.632105	-0.685556	0.367420	0	0	0	0	0	0	0	0	0	0	0	0

377 rows × 155 columns

In [55]:

```
Test = MergedData_Dummies[(pd.to_datetime(MergedData_Dummies["Date"]) > split_date)]
Test
```

Out[55]:

	Date	SECTION LENGTH	ADT	AADT	Average	HIGHWAY_1	HIGHWAY_2	HIGHWAY_3	HIGHWAY_4	HIGHWAY_6	HIGHWAY_7	HIGHWAY_14	HIGHWAY_19	HIGHWAY_32	HIGHWAY_33	HIGHWAY_101	HIGHWAY_102
0	09/09/2019	-0.729468	-0.639703	-0.692637	0.542614	1	0	0	0	0	0	0	0	0	0	0	0
4	09/09/2019	0.049800	-0.627869	-0.682460	0.542614	1	0	0	0	0	0	0	0	0	0	0	0
6	09/16/2019	-0.701817	-0.633832	-0.687861	0.397644	1	0	0	0	0	0	0	0	0	0	0	0
7	09/09/2019	0.590260	-0.641322	-0.694778	0.542614	1	0	0	0	0	0	0	0	0	0	0	0
8	09/09/2019	-0.938111	-0.634727	-0.688619	0.542614	1	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
556	09/18/2019	-0.133705	1.101726	0.946082	0.356406	0	0	0	0	0	0	0	0	0	0	0	0
557	10/03/2019	-0.216659	0.230080	0.089787	0.450663	0	0	0	0	0	0	0	0	0	0	0	0
558	10/03/2019	-0.216659	0.203766	0.056852	0.450663	0	0	0	0	0	0	0	0	0	0	0	0
559	10/03/2019	-0.309669	-0.641861	-0.694942	0.450663	0	0	0	0	0	0	0	0	0	0	0	0
560	10/03/2019	-0.334806	-0.640815	-0.693987	0.450663	0	0	0	0	0	0	0	0	0	0	0	0

209 rows × 155 columns

In [56]:

```
X_Train = Train.drop(['Date','Average'], axis=1)
Y_Train = Train[["Average"]]
X_Test = Test.drop(['Date','Average'], axis=1)
Y_Test = Test[["Average"]]
```

In [57]:

```
X_Train
```

Out[57]:

	SECTION LENGTH	ADT	AADT	HIGHWAY_1	HIGHWAY_2	HIGHWAY_3	HIGHWAY_4	HIGHWAY_6	HIGHWAY_7	HIGHWAY_14	HIGHWAY_19	HIGHWAY_32	HIGHWAY_33	HIGHWAY_101	HIGHWAY_102	HIGHWAY_103
1	0.049800	-0.634112	-0.687993	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0.049800	-0.635204	-0.688965	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0.049800	-0.638526	-0.691945	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0.049800	-0.638329	-0.691781	1	0	0	0	0	0	0	0	0	0	0	0	0
18	0.693325	-0.631408	-0.687730	0	1	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
581	-0.143760	0.144560	0.023918	0	0	0	0	0	0	0	0	0	0	0	0	0
582	0.914536	0.957000	0.748475	0	0	0	0	0	0	0	0	0	0	0	0	0
583	-0.432843	-0.643509	-0.696293	0	0	0	0	0	0	0	0	0	0	0	0	0
584	-0.362458	-0.640858	-0.693790	0	0	0	0	0	0	0	0	0	0	0	0	0
585	-0.664110	-0.632105	-0.685556	0	0	0	0	0	0	0	0	0	0	0	0	0

377 rows × 153 columns

In [58]:

```
X_Test
```

Out[58]:

	SECTION LENGTH	ADT	AADT	HIGHWAY_1	HIGHWAY_2	HIGHWAY_3	HIGHWAY_4	HIGHWAY_6	HIGHWAY_7	HIGHWAY_14	HIGHWAY_19	HIGHWAY_32	HIGHWAY_33	HIGHWAY_101	HIGHWAY_102	HIGHWAY_103
0	-0.729468	-0.639703	-0.692637	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0.049800	-0.627869	-0.682460	1	0	0	0	0	0	0	0	0	0	0	0	0
6	-0.701817	-0.633832	-0.687861	1	0	0	0	0	0	0	0	0	0	0	0	0
7	0.590260	-0.641322	-0.694778	1	0	0	0	0	0	0	0	0	0	0	0	0

	SECTION LENGTH	ADT	AADT	HIGHWAY_1	HIGHWAY_2	HIGHWAY_3	HIGHWAY_4	HIGHWAY_6	HIGHWAY_7	HIGHWAY_14	HIGHWAY_19	HIGHWAY_32	HIGHWAY_33	HIGHWAY_101	HIGHWAY_102	HIGHWAY_103	HIGHW
8	-0.938111	-0.634727	-0.688619	1	0	0	0	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
556	-0.133705	1.101726	0.946082	0	0	0	0	0	0	0	0	0	0	0	0	0	
557	-0.216659	0.230080	0.089787	0	0	0	0	0	0	0	0	0	0	0	0	0	
558	-0.216659	0.203766	0.056852	0	0	0	0	0	0	0	0	0	0	0	0	0	
559	-0.309669	-0.641861	-0.694942	0	0	0	0	0	0	0	0	0	0	0	0	0	
560	-0.334806	-0.640815	-0.693987	0	0	0	0	0	0	0	0	0	0	0	0	0	

209 rows × 153 columns

In [59]:
```
model = linear_model.LinearRegression()
```

In [60]:
```
model.fit(X_Train,Y_Train)
```

Out[60]:
```
LinearRegression()
```

In [61]:
```
y_trainpred = model.predict(X_Train)
```

In [62]:
```
y_prediction = model.predict(X_Test)
```

In [63]:
```
scoretrain = r2_score(Y_Train,y_trainpred)
scoretrain
```

Out[63]:
```
0.7169531837632088
```

In [64]:
```
score = r2_score(Y_Test,y_prediction)
score
```

Out[64]:
```
-3.626032695825652e+23
```

In [65]:
```
mean_squared_error(Y_Train,y_trainpred)
```

Out[65]:
```
0.005974097022804146
```

In [66]:
```
mean_squared_error(Y_Test,y_prediction)
```

Out[66]:
```
1.7049919534937502e+22
```

Clustering

In [67]:
```
#removing AADT as ADT and AADT are collinear and if we keep both it will give twice weightage to those datapoints.
MergedData_NoDate.drop("AADT", axis=1, inplace=True)
```

In [68]:
```
MergedData_NoDate
```

Out[68]:

	HIGHWAY	SECTION	SECTION LENGTH	ADT	Average
0	1	47	0.209552	0.001571	0.542614
1	1	50	0.360624	0.003282	0.698085
2	1	50	0.360624	0.002948	0.698085
3	1	50	0.360624	0.001932	0.698085
4	1	50	0.360624	0.005191	0.542614
...	...	...	...	...	...
581	374	28	0.323099	0.241446	0.201447
582	374	30	0.528265	0.489938	0.201447
583	376	10	0.267057	0.000407	0.541973
584	376	20	0.280702	0.001218	0.367420
585	376	30	0.222222	0.003895	0.367420

586 rows × 5 columns

In [69]:
```
#scaling the data to normalize all values before doing clustering
scaler.fit(MergedData_NoDate)
Scaled_MergedData = scaler.transform(MergedData_NoDate)
```

In [70]:
```
Scaled_MergedData
```

Out[70]:
```
array([[-1.17641908, -0.10199088, -0.72946801, -0.63970347,  0.40924857],
       [-1.17641908, -0.04905319,  0.04980006, -0.63411178,  1.30129509],
       [-1.17641908, -0.04905319,  0.04980006, -0.63520381,  1.30129509],
       ...,
       [ 1.81292057, -0.75488913, -0.43284339, -0.64350911,  0.40557457],
       [ 1.81292057, -0.57843014, -0.36245789, -0.64085799, -0.59595707],
       [ 1.81292057, -0.40197116, -0.66411004, -0.63210535, -0.59595707]])
```

In [71]:
```
silhouette_coefficients = []
```

In [72]:
```
#storing all silhouette values in array to find the best one
for x in range(2,10):
    Kmeans = KMeans(n_clusters = x)
    Kmeans.fit(Scaled_MergedData)
    score = silhouette_score(Scaled_MergedData, Kmeans.labels_)
    silhouette_coefficients.append(score)
```

In [73]:
```
df = pd.DataFrame({'silhouette_coefficient':silhouette_coefficients})
```

In [74]:
```
df['cluster'] = range(2,10)
```

In [75]:
```
df.set_index('cluster')
```

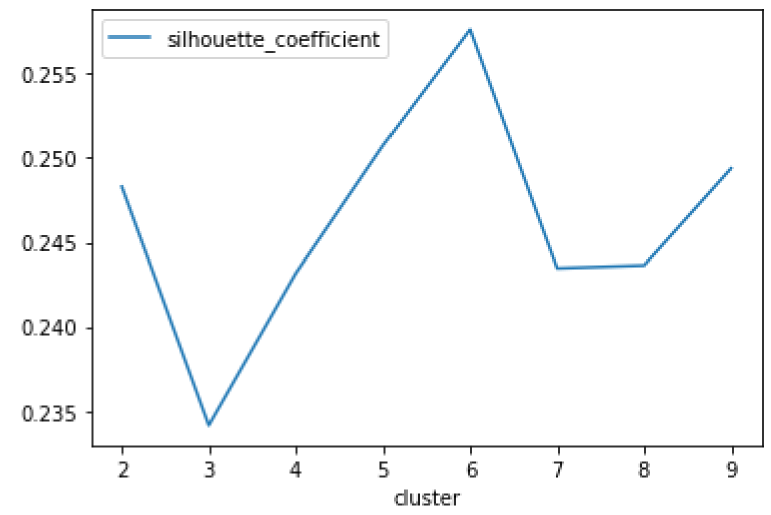
Out[75]:

	silhouette_coefficient
cluster	

	silhouette_coefficient
cluster	
2	0.248295
3	0.234144
4	0.243156
5	0.250728
6	0.257624
7	0.243456
8	0.243627
9	0.249379

In [76]:
```
#plotting graph for all silhouette values for and finding for how many clusters it is the greatest
df.plot(x='cluster', y='silhouette_coefficient')
```

Out[76]:
```
<AxesSubplot:xlabel='cluster'>
```



Applying Clustering for 2 and then doing PCA for Dimensionality reduction and plotting result

In [77]:
```
#as we got highest silhouette score for 6 clusters let us do clustering n_clusters = 6
k_means = KMeans(n_clusters = 6)
```

In [78]:
```
MergedData
```

Out[78]:

	Date	HIGHWAY	SECTION	SECTION LENGTH	ADT	AADT	Average
0	09/09/2019	1	47	0.209552	0.001571	0.001446	0.542614
1	06/17/2019	1	50	0.360624	0.003282	0.002872	0.698085
2	06/17/2019	1	50	0.360624	0.002948	0.002573	0.698085
3	06/17/2019	1	50	0.360624	0.001932	0.001658	0.698085
4	09/09/2019	1	50	0.360624	0.005191	0.004570	0.542614
...	...	...	...	...	...	...	...
581	06/27/2019	374	28	0.323099	0.241446	0.221436	0.201447
582	06/27/2019	374	30	0.528265	0.489938	0.443883	0.201447
583	06/04/2019	376	10	0.267057	0.000407	0.000324	0.541973
584	05/28/2019	376	20	0.280702	0.001218	0.001092	0.367420
585	05/28/2019	376	30	0.222222	0.003895	0.003620	0.367420

586 rows × 7 columns

In [79]:
```
label = k_means.fit_predict(MergedData.drop(['Date'], axis=1))
```

In [80]:
```
MergedData['Clusterd_label'] = label
```

In [81]:
```
pca = PCA(n_components=2)
```

In [82]:
```
df_pca = pca.fit_transform(MergedData.drop(['Date','Clusterd_label'], axis=1))
df_pca.shape
```

Out[82]:
```
(586, 2)
```

In [83]:
```
df = pd.DataFrame(df_pca, columns=['first','second'])
df['Clusterd_label'] = label
df['Clusterd_label'] = df['Clusterd_label'].astype('category')
df
```

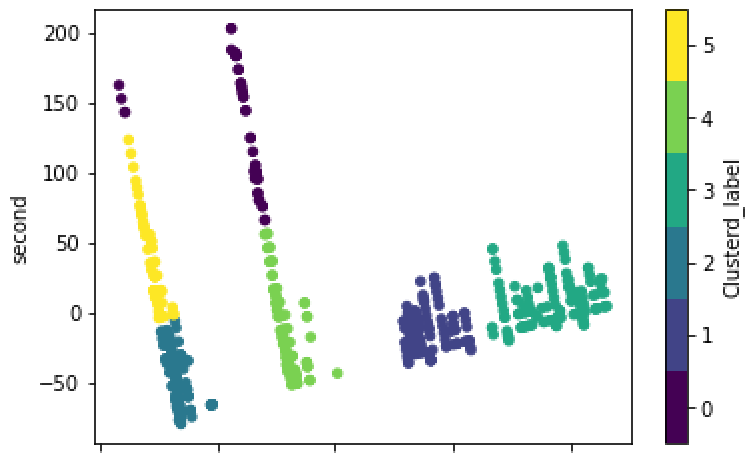
Out[83]:

	first	second	Clusterd_label
0	-143.247890	-35.950088	2
1	-143.863681	-33.013978	2
2	-143.863681	-33.013977	2
3	-143.863682	-33.013976	2
4	-143.863663	-33.013976	2
...	...	...	...
581	225.709379	22.021172	3
582	225.299145	23.978286	3
583	231.361395	4.815326	3
584	229.308691	14.602372	3
585	227.255952	24.389410	3

586 rows × 3 columns

In [84]:
```
#graph showing clusters created for highest silhouette value
df.plot.scatter(x='first', y='second', c = 'Clusterd_label', colormap='viridis')
```

Out[84]:
```
<AxesSubplot:xlabel='first', ylabel='second'>
```



Applying Decision tree classifier after clustering

In [85]:
```
DT_MergedData = MergedData.drop(['Date','AADT'], axis=1)
DT_MergedData
```

Out[85]:

	HIGHWAY	SECTION	SECTION LENGTH	ADT	Average	Clusterd_label
0	1	47	0.209552	0.001571	0.542614	2
1	1	50	0.360624	0.003282	0.698085	2
2	1	50	0.360624	0.002948	0.698085	2
3	1	50	0.360624	0.001932	0.698085	2
4	1	50	0.360624	0.005191	0.542614	2
...	...	...	...	...	...	...
581	374	28	0.323099	0.241446	0.201447	3
582	374	30	0.528265	0.489938	0.201447	3
583	376	10	0.267057	0.000407	0.541973	3
584	376	20	0.280702	0.001218	0.367420	3
585	376	30	0.222222	0.003895	0.367420	3

586 rows × 6 columns

Applying Decision Tree classsifier for new clustured dataset

In [86]:
```
X = DT_MergedData.values[:,0:5]
Y = DT_MergedData.values[:,-1]
```

In [87]:
```
X_train, X_test, y_train, y_test = train_test_split(
X,Y, test_size = 0.3, random_state = 100)
```

In [88]:
```
clf_tree = DecisionTreeClassifier()
clf_tree.fit(X_train,y_train)
```

Out[88]:
```
DecisionTreeClassifier()
```

In [89]:
```
y_trainpred = clf_tree.predict(X_train)
```

In [90]:
```
print("Confusion Matrix fot raining data: \n",confusion_matrix(y_train,y_trainpred))
```

```
Confusion Matrix fot raining data:
 [[34  0  0  0  0  0]
 [ 0 90  0  0  0  0]
 [ 0  0 86  0  0  0]
 [ 0  0  0 88  0  0]
 [ 0  0  0  0 70  0]
 [ 0  0  0  0  0 42]]
```

In [91]:
```
print ("Accuracy of training data: \n",accuracy_score(y_train,y_trainpred)*100)
```

```
Accuracy of training data:
 100.0
```

In [92]:
```
y_pred = clf_tree.predict(X_test)
```

In [93]:
```
print("Confusion Matrix: \n", confusion_matrix(y_test,y_pred))
```

```
Confusion Matrix:
 [[10  0  0  0  0  0]
 [ 0 34  0  0  0  0]
 [ 0  0 36  0  0  0]
 [ 0  0  0 43  0  0]
 [ 0  0  0  0 32  0]
 [ 0  0  0  0  0 21]]
```

In [94]:
```
conmat = confusion_matrix(y_test,y_pred)
```

In [95]:
```
TP = conmat[0][0]
FN = conmat[0][1]
FP = conmat[1][0]
TN = conmat[1][1]
```

In [96]:
```
Accuracy = (TP + TN)/(TP+FN+FP+TN)
Accuracy
```

Out[96]:
```
1.0
```

In [97]:
```
Precision = TP/(TP+FP)
Precision
```

Out[97]:
```
1.0
```

In [98]:
```
 Recall = TP/(TP+FN)
Recall
```

Out[98]:
```
1.0
```

In [99]:
```
F1_measure = (2*(Recall)*(Precision))/(Recall+Precision)
F1_measure
```

Out[99]:
```
1.0
```

In [100…
```
print ("Accuracy of testdata: \n",accuracy_score(y_test,y_pred)*100)
```

```
Accuracy of testdata:
 100.0
```

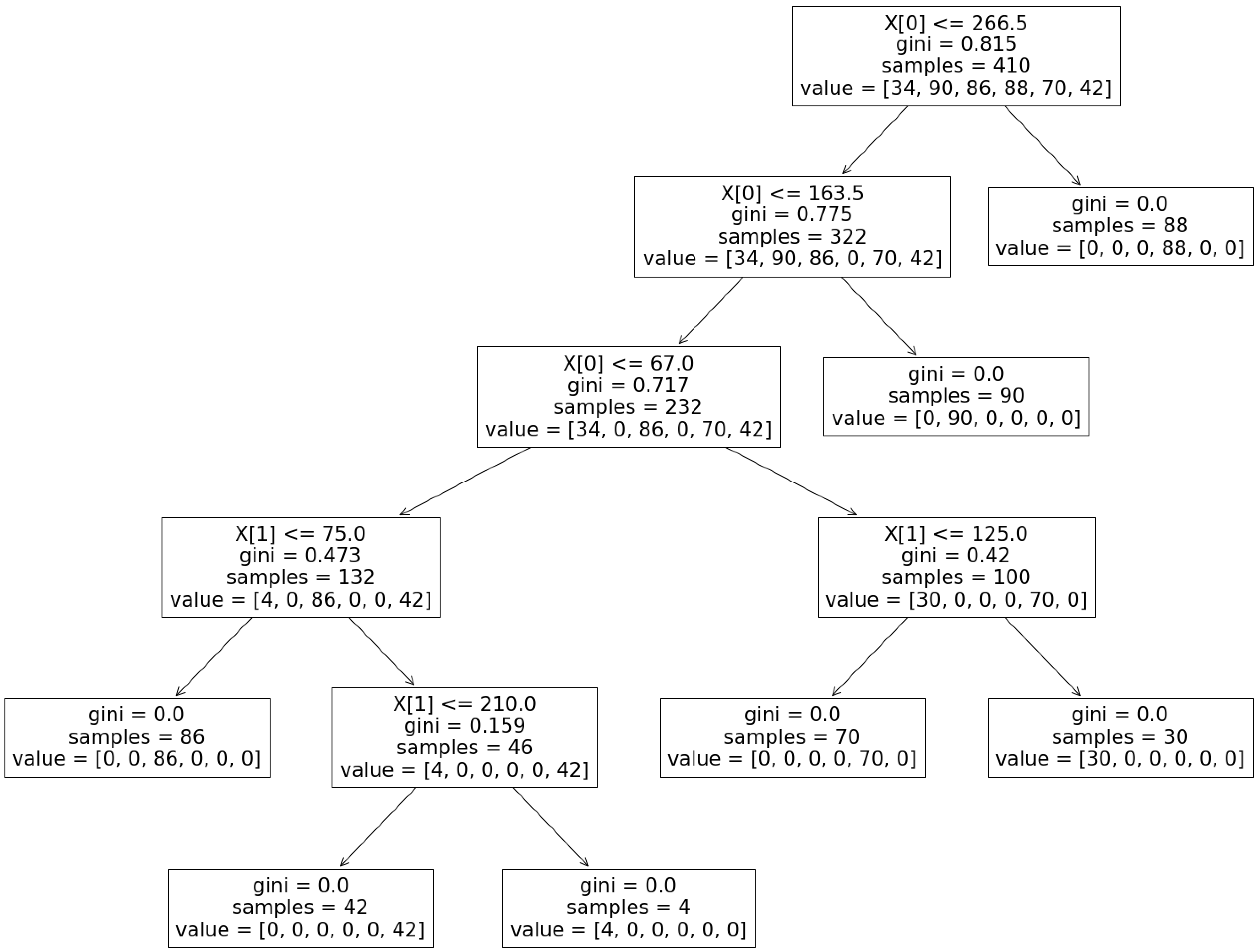
In [101…

```
print("Report : \n",classification_report(y_test, y_pred))
```

```
Report :
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        10
         1.0       1.00      1.00      1.00        34
         2.0       1.00      1.00      1.00        36
         3.0       1.00      1.00      1.00        43
         4.0       1.00      1.00      1.00        32
         5.0       1.00      1.00      1.00        21

    accuracy                           1.00       176
   macro avg       1.00      1.00      1.00       176
weighted avg       1.00      1.00      1.00       176
```

In [102…

```
from sklearn import tree
from matplotlib import pyplot as plt


fig = plt.figure(figsize=(25,20))
tree.plot_tree(clf_tree)
fig.savefig("decistion_tree.png")
```



Applying Navie Bayes after clustering

In [103…

```
X = DT_MergedData.values[:,0:5]
Y = DT_MergedData.values[:,-1]
```

In [104…

```
X_train, X_test, y_train, y_test = train_test_split(
X,Y, test_size = 0.3, random_state = 100)
```

In [105…

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

```
C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-packages\sklearn\base.py:441: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
C:\Users\Mayank\AppData\Local\Programs\Python\Python38\lib\site-packages\sklearn\base.py:441: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [106…

```
NBclassifier = GaussianNB()
NBclassifier.fit(X_train,y_train)
```

Out[106…

```
GaussianNB()
```

In [107…

```
y_trainpred = NBclassifier.predict(X_train)
```

In [108…

```
print("Confusion Matrix for training data: \n",confusion_matrix(y_train,y_trainpred))
```

```
Confusion Matrix for training data:
 [[31  0  0  0  1  2]
 [ 0 90  0  0  0  0]
 [ 0  0 83  0  0  3]
```

```
 [ 0  0  0 88  0  0]
 [ 1  0  0  0 69  0]
 [ 0  0  3  0  0 39]]
```

In [109…

```
print ("Accuracy of training data: \n",accuracy_score(y_train,y_trainpred)*100)
```

```
Accuracy of training data: 
 97.5609756097561
```

In [110…

```
y_pred = NBclassifier.predict(X_test)
```

In [111…

```
print ("Accuracy of testdata: \n",accuracy_score(y_test,y_pred)*100)
```

```
Accuracy of testdata: 
 98.29545454545455
```

In [112…

```
print("Confusion Matrix: \n", confusion_matrix(y_test,y_pred))
```

```
Confusion Matrix: 
 [[ 9  0  0  0  1  0]
 [ 0 34  0  0  0  0]
 [ 0  0 35  0  0  1]
 [ 0  0  0 43  0  0]
 [ 1  0  0  0 31  0]
 [ 0  0  0  0  0 21]]
```

Statistical test for best hypothesis

Performing 10X10-Fold cross validation and student t-test on accuaracy obtained from cross validations

In [113…

```
NBAccuracy_List = []
DTAccuracy_List = []
#length of datapoints for training
n1_train = []
#length of datapoints for testing
n2_test = []
kf = KFold(n_splits=10)
for i in range(10):
    for train_index, test_index in kf.split(X):
        Xtrain, Xtest = X[train_index], X[test_index]
        ytrain, ytest = Y[train_index], Y[test_index]

        n1_train.append(len(ytrain))
        n2_test.append(len(ytest))

        clf_tree.fit(Xtrain, ytrain)
        NBclassifier.fit(Xtrain, ytrain)

        ypredDT = clf_tree.predict(Xtest)
        ypredNB = NBclassifier.predict(Xtest)

        DTAccuracy = accuracy_score(ytest, ypredDT)
        NBAccuracy = accuracy_score(ytest, ypredNB)
        DTAccuracy_List.append(DTAccuracy)
        NBAccuracy_List .append(NBAccuracy)
```

In [114…

```
Differences_list = [y -x for y, x in zip(DTAccuracy_List,NBAccuracy_List)]
```

In [115…

```
#mean of differences
d_bar = np.mean(Differences_list)
d_bar
```

Out[115…

```
0.06708357685563995
```

In [116…

```
#variance of differences
sigma2 = np.var(Differences_list, ddof=1)
sigma2
```

Out[116…

```
0.031042436742399274
```

In [117…

```
#no of datapoints used for training
n1 = np.median(n1_train)
n1
```

Out[117…

```
527.0
```

In [118…

```
#no of datapoints used for testing
n2 = np.median(n2_test)
n2
```

Out[118…

```
59.0
```

In [119…

```
#compute Total number of datapoints
n = 10 * 10
n
```

Out[119…

```
100
```

In [120…

```
#computing modified variance
sigma2_mod = sigma2*(1/n + n2/n1)
sigma2_mod
```

Out[120…

```
0.0037857635852637595
```

In [121…

```
#computing t-static
t_static = d_bar / np.sqrt(sigma2_mod)
t_static
```

Out[121…

```
1.0902835664262074
```

In [122…

```
#computing p-value
Pvalue = ((1 - t.cdf(np.abs(t_static),n-1))*2)
Pvalue
```

Out[122…

```
0.2782348766691636
```