National College of Ireland

Configuration Manual

MSc. Research Project
MSc. In Data Analytics

Shrey Sanjay Shah
Student ID: X18192271

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shrey Sanjay Shah		
Student ID:	X18192271		
Programme:	MSc. In Data Analytics	**Year:**	2019-2020
Module:	MSc. Research Project		
Supervisor:	Mr. Hicham Rifai		
Submission Due Date:	28th September 2020		
Project Title:	Developing promotional model using customer lifetime value score to avoid Customer churn		
Word Count:	863	**Page Count**:	12

Signature: Shrey Sanjay Shah

Date: 28/09/2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shrey Sanjay Shah
x18192271

1 Introduction

This configuration manual gives comprehensive set of procedures required to be followed in order to replicate the proposed research and achieve identical results. The manual includes various sections such as minimum software configuration required, steps to replicate sub parts of this research such as pre-processing, transformation, implementation as well as evaluation.

2 System Requirement

All the required tools and software for this research can be easily installed in any computer system having basic configuration list given below:

Operating System	Windows 10
RAM	8GB+
Hard Disk	256GB SSD
Processor	Intel Core i5 8th gen +

All the basic tools / software requires for implementation of this research are listed below:

a. Microsoft Office Suite
b. Python 3.7
c. Anaconda Jupyter Notebook

From Microsoft Office suite, MS Excel and MS word were used for data viewing, selection and reporting. Python is the core programming language used for the purpose of this research. For the purpose of this research python 3.7 was used which could be downloaded for free from their official website[1]. The platform used for programming in python was Anaconda which is again freely available to download from their website [2]. The program called Jupyter notebook from the Anaconda Navigator was used. The advantages of Jupyter Notebooks are ease of use, fast implementation, portability, etc.

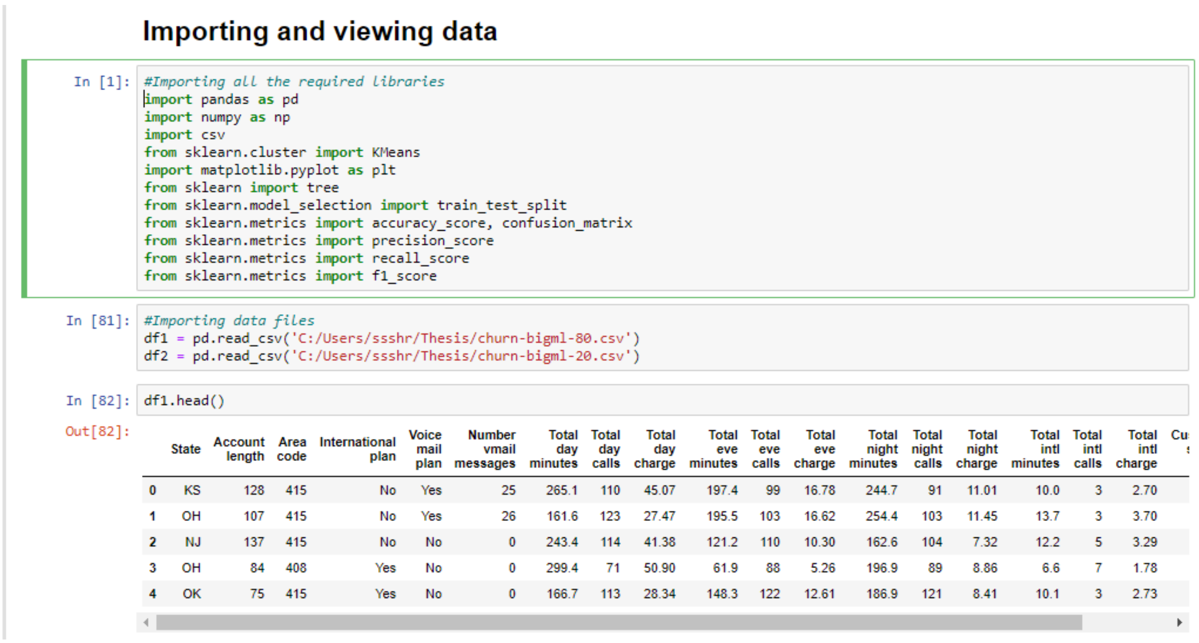
[1] https://www.python.org/downloads/
[2] https://repo.anaconda.com/archive/Anaconda3-2020.07-Windows-x86_64.exe

3 Dataset Selection

The Orange telecom's datasets were made available on KDD championship data [3] but the same data was processed and reuploaded[4][5] with 2 different datasets from the same company. This processed data had all the redundant features eliminated and only the required features for churn prediction were kept.

4 Importing required libraries and datasets

All the required libraries are imported and dataset is imported and viewed as well as shown below.



Importing and viewing data

```
In [1]: #Importing all the required libraries
        import pandas as pd
        import numpy as np
        import csv
        from sklearn.cluster import KMeans
        import matplotlib.pyplot as plt
        from sklearn import tree
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score, confusion_matrix
        from sklearn.metrics import precision_score
        from sklearn.metrics import recall_score
        from sklearn.metrics import f1_score
```

```
In [81]: #Importing data files
         df1 = pd.read_csv('C:/Users/ssshr/Thesis/churn-bigml-80.csv')
         df2 = pd.read_csv('C:/Users/ssshr/Thesis/churn-bigml-20.csv')
```

```
In [82]: df1.head()
```

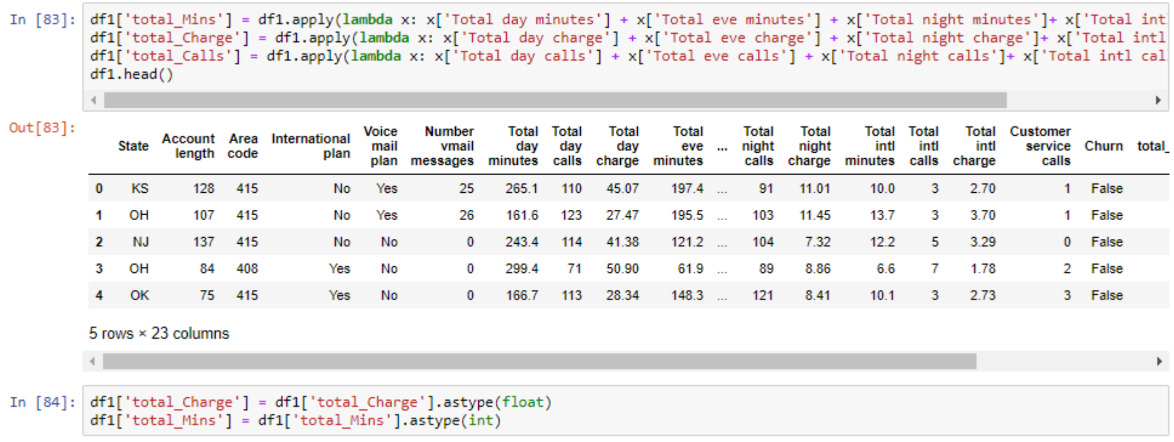
Out[82]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls	Total intl charge	Cu
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	

[3] https://www.kdd.org/kdd-cup/view/kdd-cup-2009/Data
[4] https://bml-data.s3.amazonaws.com/churn-bigml-80.csv
[5] https://bml-data.s3.amazonaws.com/churn-bigml-20.csv

5 Pre-processing

All the redundant features were eliminated, new attributes necessary for future implementation of the model were calculated and some datatypes were changed for ease of use further.



```
In [83]: df1['total_Mins'] = df1.apply(lambda x: x['Total day minutes'] + x['Total eve minutes'] + x['Total night minutes']+ x['Total int
         df1['total_Charge'] = df1.apply(lambda x: x['Total day charge'] + x['Total eve charge'] + x['Total night charge']+ x['Total intl
         df1['total_Calls'] = df1.apply(lambda x: x['Total day calls'] + x['Total eve calls'] + x['Total night calls']+ x['Total intl cal
         df1.head()
```

Out[83]:

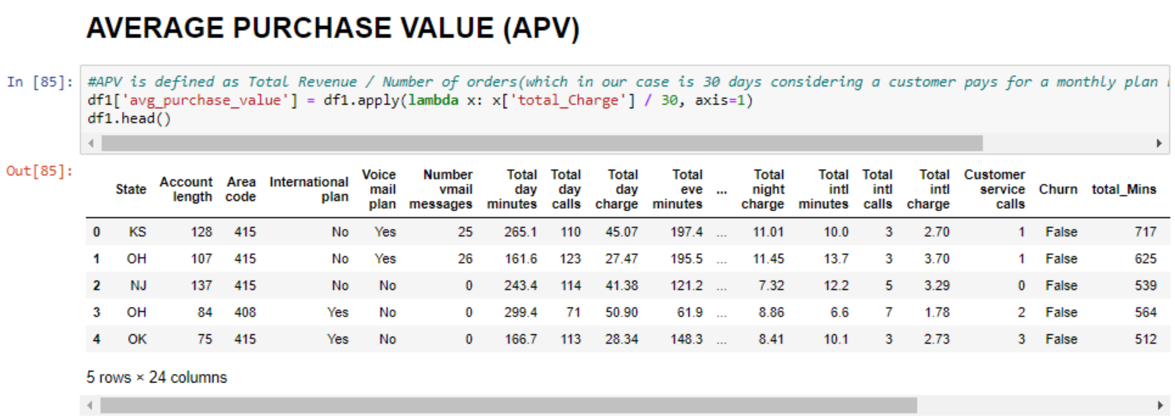
	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Total night calls	Total night charge	Total intl minutes	Total intl calls	Total intl charge	Customer service calls	Churn	total_
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	...	91	11.01	10.0	3	2.70	1	False	
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	...	103	11.45	13.7	3	3.70	1	False	
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	...	104	7.32	12.2	5	3.29	0	False	
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	...	89	8.86	6.6	7	1.78	2	False	
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	...	121	8.41	10.1	3	2.73	3	False	

5 rows × 23 columns

```
In [84]: df1['total_Charge'] = df1['total_Charge'].astype(float)
         df1['total_Mins'] = df1['total_Mins'].astype(int)
```

6 CLV Calculation Model

4 attributes are calculated before getting the final CLV scores. After getting final CLV score the updated dataset is stored in csv format in order to retrieve the updated data whenever necessary.



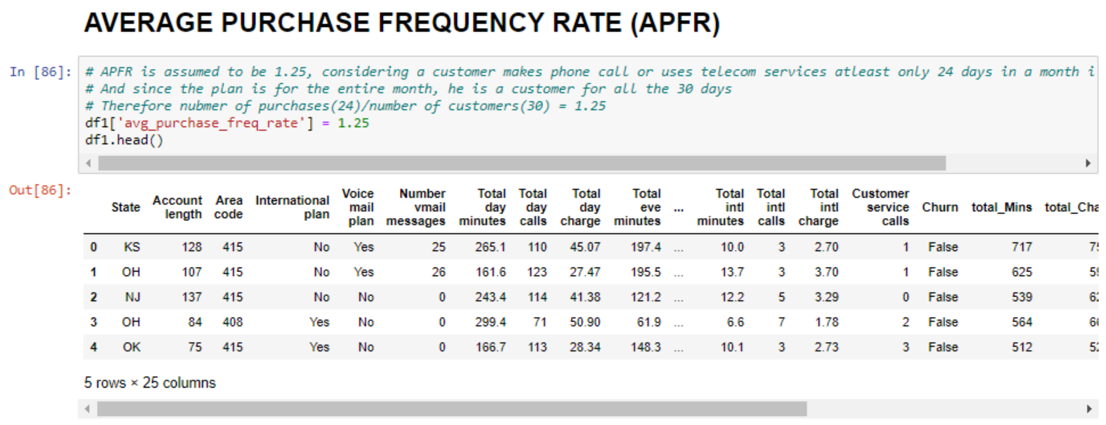
AVERAGE PURCHASE VALUE (APV)

```
In [85]: #APV is defined as Total Revenue / Number of orders(which in our case is 30 days considering a customer pays for a monthly plan
         df1['avg_purchase_value'] = df1.apply(lambda x: x['total_Charge'] / 30, axis=1)
         df1.head()
```

Out[85]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Total night charge	Total intl minutes	Total intl calls	Total intl charge	Customer service calls	Churn	total_Mins
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	...	11.01	10.0	3	2.70	1	False	717
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	...	11.45	13.7	3	3.70	1	False	625
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	...	7.32	12.2	5	3.29	0	False	539
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	...	8.86	6.6	7	1.78	2	False	564
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	...	8.41	10.1	3	2.73	3	False	512

5 rows × 24 columns



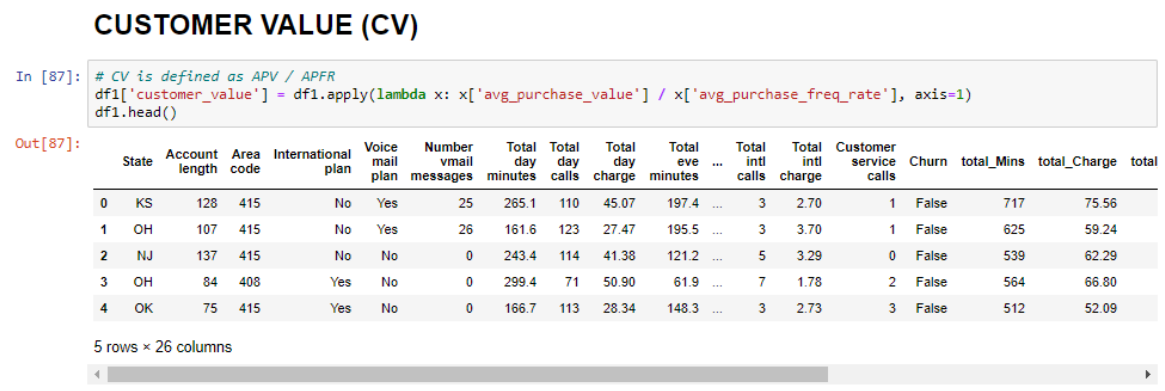
AVERAGE PURCHASE FREQUENCY RATE (APFR)

```
In [86]: # APFR is assumed to be 1.25, considering a customer makes phone call or uses telecom services atleast only 24 days in a month i
         # And since the plan is for the entire month, he is a customer for all the 30 days
         # Therefore nubmer of purchases(24)/number of customers(30) = 1.25
         df1['avg_purchase_freq_rate'] = 1.25
         df1.head()
```

Out[86]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Total intl minutes	Total intl calls	Total intl charge	Customer service calls	Churn	total_Mins	total_Cha
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	...	10.0	3	2.70	1	False	717	7
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	...	13.7	3	3.70	1	False	625	5
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	...	12.2	5	3.29	0	False	539	6
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	...	6.6	7	1.78	2	False	564	6
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	...	10.1	3	2.73	3	False	512	5

5 rows × 25 columns

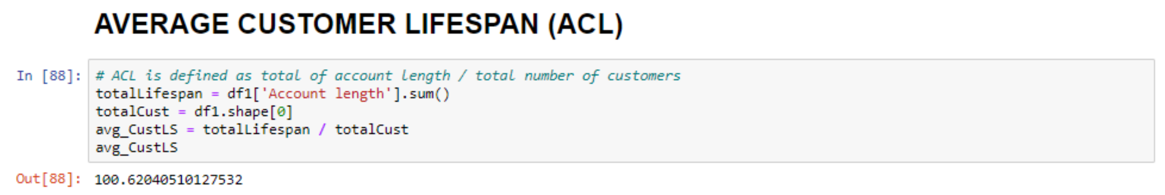

CUSTOMER VALUE (CV)

In [87]:
```
# CV is defined as APV / APFR
df1['customer_value'] = df1.apply(lambda x: x['avg_purchase_value'] / x['avg_purchase_freq_rate'], axis=1)
df1.head()
```

Out[87]:

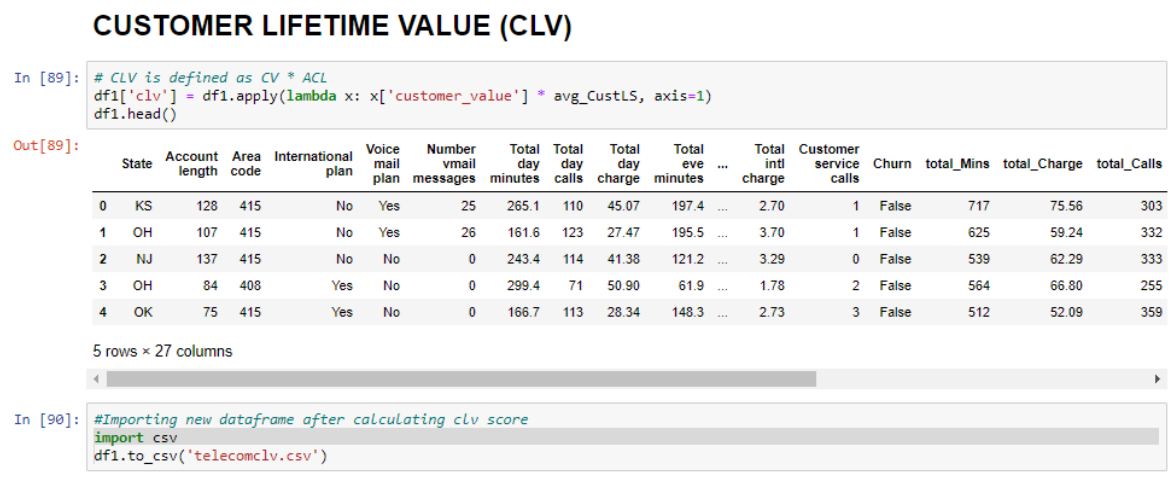
	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Total intl calls	Total intl charge	Customer service calls	Churn	total_Mins	total_Charge	total
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	...	3	2.70	1	False	717	75.56	
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	...	3	3.70	1	False	625	59.24	
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	...	5	3.29	0	False	539	62.29	
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	...	7	1.78	2	False	564	66.80	
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	...	3	2.73	3	False	512	52.09	

5 rows × 26 columns


AVERAGE CUSTOMER LIFESPAN (ACL)

In [88]:
```
# ACL is defined as total of account length / total number of customers
totalLifespan = df1['Account length'].sum()
totalCust = df1.shape[0]
avg_CustLS = totalLifespan / totalCust
avg_CustLS
```

Out[88]:
```
100.62040510127532
```


CUSTOMER LIFETIME VALUE (CLV)

In [89]:
```
# CLV is defined as CV * ACL
df1['clv'] = df1.apply(lambda x: x['customer_value'] * avg_CustLS, axis=1)
df1.head()
```

Out[89]:

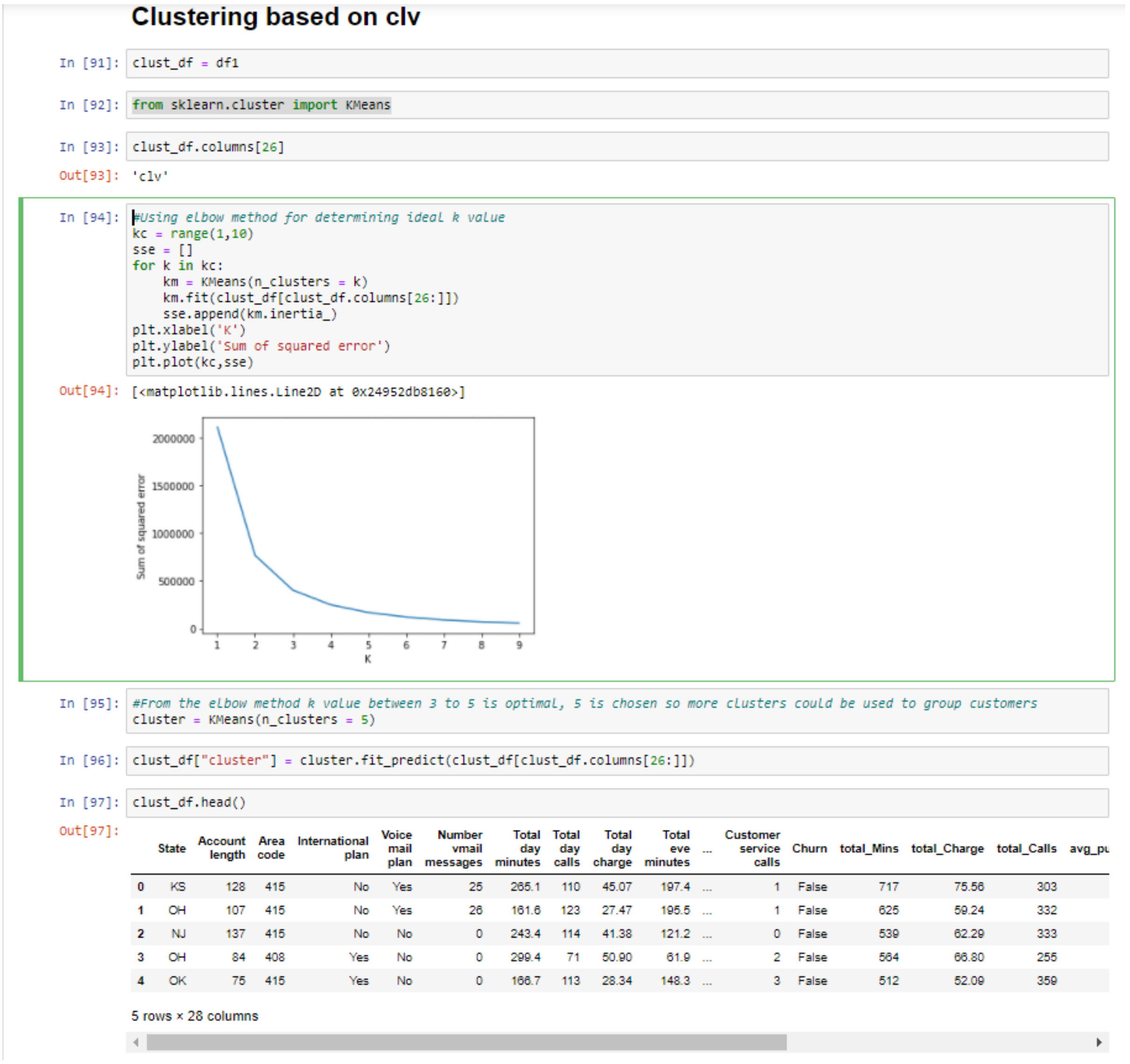
	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Total intl charge	Customer service calls	Churn	total_Mins	total_Charge	total_Calls
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	...	2.70	1	False	717	75.56	303
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	...	3.70	1	False	625	59.24	332
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	...	3.29	0	False	539	62.29	333
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	...	1.78	2	False	564	66.80	255
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	...	2.73	3	False	512	52.09	359

5 rows × 27 columns

In [90]:
```
#Importing new dataframe after calculating clv score
import csv
df1.to_csv('telecomclv.csv')
```

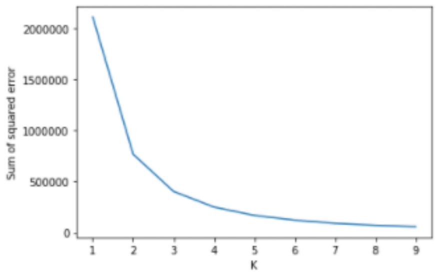
7 CLV Clustering Model

In this model, the customers are grouped in clusters based on their CLV scores. Also the most optimal number of clusters is selected to be 5 using the elbow method. The clusters are arranged in ascending order to ease mapping in further steps.



Clustering based on clv

```
In [91]: clust_df = df1

In [92]: from sklearn.cluster import KMeans

In [93]: clust_df.columns[26]

Out[93]: 'clv'

In [94]: #Using elbow method for determining ideal k value
         kc = range(1,10)
         sse = []
         for k in kc:
             km = KMeans(n_clusters = k)
             km.fit(clust_df[clust_df.columns[26:]])
             sse.append(km.inertia_)
         plt.xlabel('K')
         plt.ylabel('Sum of squared error')
         plt.plot(kc,sse)

Out[94]: [<matplotlib.lines.Line2D at 0x24952db8160>]
```



```
In [95]: #From the elbow method k value between 3 to 5 is optimal, 5 is chosen so more clusters could be used to group customers
         cluster = KMeans(n_clusters = 5)

In [96]: clust_df["cluster"] = cluster.fit_predict(clust_df[clust_df.columns[26:]])

In [97]: clust_df.head()

Out[97]:
```

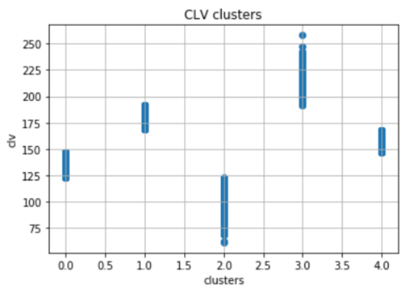
	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Customer service calls	Churn	total_Mins	total_Charge	total_Calls	avg_pu
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	...	1	False	717	75.56	303	
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	...	1	False	625	59.24	332	
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	...	0	False	539	62.29	333	
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	...	2	False	564	66.80	255	
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	...	3	False	512	52.09	359	

5 rows × 28 columns

Resulting cluster shown with the help of scatter plot

```
In [98]: import matplotlib.pyplot as plt
```

```
In [99]: #Evaluation of k means clustering through visualization
         plt.scatter(clust_df["cluster"], clust_df["clv"])
         plt.xlabel('clusters')
         plt.ylabel('clv')
         plt.title('CLV clusters')
         plt.grid()
         plt.show()
```



```
In [100]: #From the above graph, it can be evaluated that the clusters are evenly grouped.
          #But,the clusters are randomly assigned and changes everytime when the project is executed.
          #hence, it becomes difficult to map which cluster is having customers with least CLV scores overall
          # Thus, the Clusters should be arranged in ascending order i.e. the customers in cluster 0 having the least CLV scores
          # and likewise, customers in cluster 4 having the highest CLV scores.
```

Sorting clusters:

Sorting CLV based clusters in ascending order

```
In [101]: #Calculating average CLV scores for individual clusters
          import numpy as np
          arr = np.empty(5)
          i = 0
          while (i < 5):
              select_cluster = clust_df.loc[clust_df['cluster'] == i]
              arr[i] = select_cluster["clv"].mean()
              clust_df.loc[clust_df['cluster'] == i, 'clv_avg'] = arr[i]
              i = i + 1
```

```
In [102]: clust_df.head()
```

Out[102]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	Churn	total_Mins	total_Charge	total_Calls	avg_purchase_valu
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	...	False	717	75.56	303	2.5186
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	...	False	625	59.24	332	1.9746
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	...	False	539	62.29	333	2.0763
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	...	False	564	66.80	255	2.2266
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	...	False	512	52.09	359	1.7363

5 rows × 29 columns

```
In [103]: # Sorting Clusters in ascending order
          arr = np.sort(arr)
          arr

Out[103]: array([108.91429162, 136.48796415, 157.76693214, 178.94277924,
                 205.20101535])
```

```
In [104]: #Replacing new cluster values in dataframe clust_df
          i = 0
          while (i < 5):
              clust_df.loc[clust_df['clv_avg'] == arr[i], 'clust'] = i
              i = i + 1
```

```
In [105]: clust_df.head()
```

Out[105]:

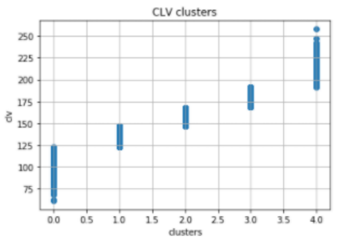
	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	...	total_Mins	total_Charge	total_Calls	avg_purchase_value	avg_
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	...	717	75.56	303	2.518667	
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	...	625	59.24	332	1.974667	
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	...	539	62.29	333	2.076333	
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	...	564	66.80	255	2.226667	
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	...	512	52.09	359	1.736333	

5 rows × 30 columns

```
In [106]: #Removing all the unwanted columns
          del clust_df['clv_avg']
          del clust_df['cluster']
```

Resulting sorted clusters visualization with the help of scatter plot:

```
In [107]: plt.scatter(clust_df["clust"], clust_df["clv"])
          plt.xlabel('clusters')
          plt.ylabel('clv')
          plt.title('CLV clusters')
          plt.grid()
          plt.show()
```



```
In [108]: # Now the clusters are arranged in ascending order i.e. the customer in cluster 0 has lowest clv score and has given
          # least profits to the company
          # and like wise, the customers in cluster 4 has given highest profits to the company
          # and thus higher the cluster number more valuable the customer is to the company and should be given maximum promotional offers
          # so as to prevent the customer from churning out of the company
```

8 Churn Prediction Model

The dataset was viewed, redundant columns for churn prediction were eliminated and correlation matrix was used to eliminate highly correlated columns finally churn prediction model was implemented with the help of decision tree classifier and the results were evaluated based on multiple evaluation metrics.

Reviewing the data for churn prediction

```
In [31]: #Checking range of values in dataframe
         rev_dat = clust_df.describe().transpose()
         rev_dat.head()
```

Out[31]:

	count	mean	std	min	25%	50%	75%	max
Account length	2666.0	100.620405	39.563974	1.0	73.0	100.00	127.0	243.0
Area code	2666.0	437.438860	42.521018	408.0	408.0	415.00	510.0	510.0
Number vmail messages	2666.0	8.021755	13.612277	0.0	0.0	0.00	19.0	50.0
Total day minutes	2666.0	179.481620	54.210350	0.0	143.4	179.95	215.9	350.8
Total day calls	2666.0	100.310203	19.988162	0.0	87.0	101.00	114.0	160.0

```
In [32]: #Display columns
         rev_dat.index.values

Out[32]: array(['Account length', 'Area code', 'Number vmail messages',
                'Total day minutes', 'Total day calls', 'Total day charge',
                'Total eve minutes', 'Total eve calls', 'Total eve charge',
                'Total night minutes', 'Total night calls', 'Total night charge',
                'Total intl minutes', 'Total intl calls', 'Total intl charge',
                'Customer service calls', 'total_Mins', 'total_Charge',
                'total_Calls', 'avg_purchase_value', 'avg_purchase_freq_rate',
                'customer_value', 'clv', 'clust'], dtype=object)
```

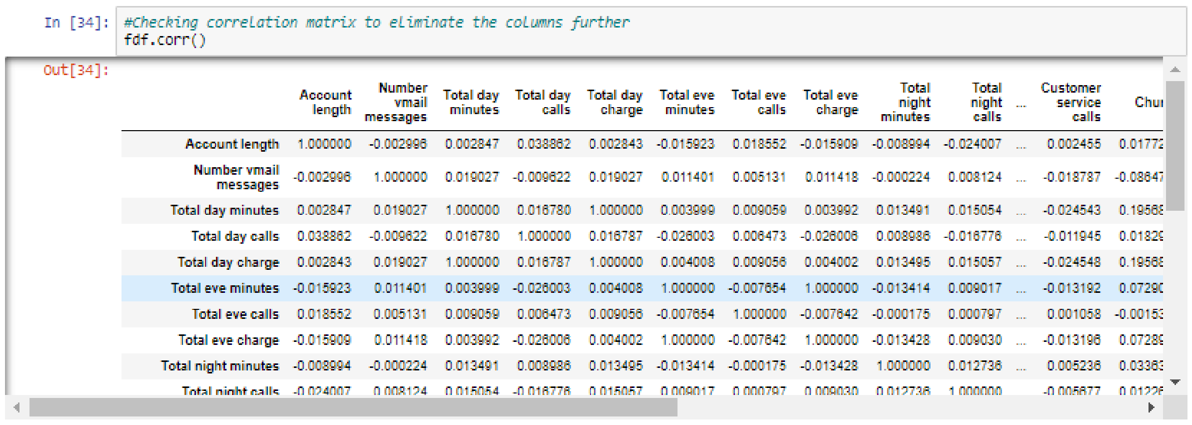
```
In [33]: #Duplicating dataframe to perform further operations
         fdf = clust_df
         #Delete redundant columns
         del fdf['Area code']
         fdf.head()
```

Out[33]:

	State	Account length	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	...	Customer service calls	Churn	total_Mins	total_Charge	total_Calls	avg_pu
0	KS	128	No	Yes	25	265.1	110	45.07	197.4	99	...	1	False	717	75.56	303	
1	OH	107	No	Yes	26	161.6	123	27.47	195.5	103	...	1	False	625	59.24	332	
2	NJ	137	No	No	0	243.4	114	41.38	121.2	110	...	0	False	539	62.29	333	
3	OH	84	Yes	No	0	299.4	71	50.90	61.9	88	...	2	False	564	66.80	255	
4	OK	75	Yes	No	0	166.7	113	28.34	148.3	122	...	3	False	512	52.09	359	

5 rows × 27 columns

Checking Correlation matrix:

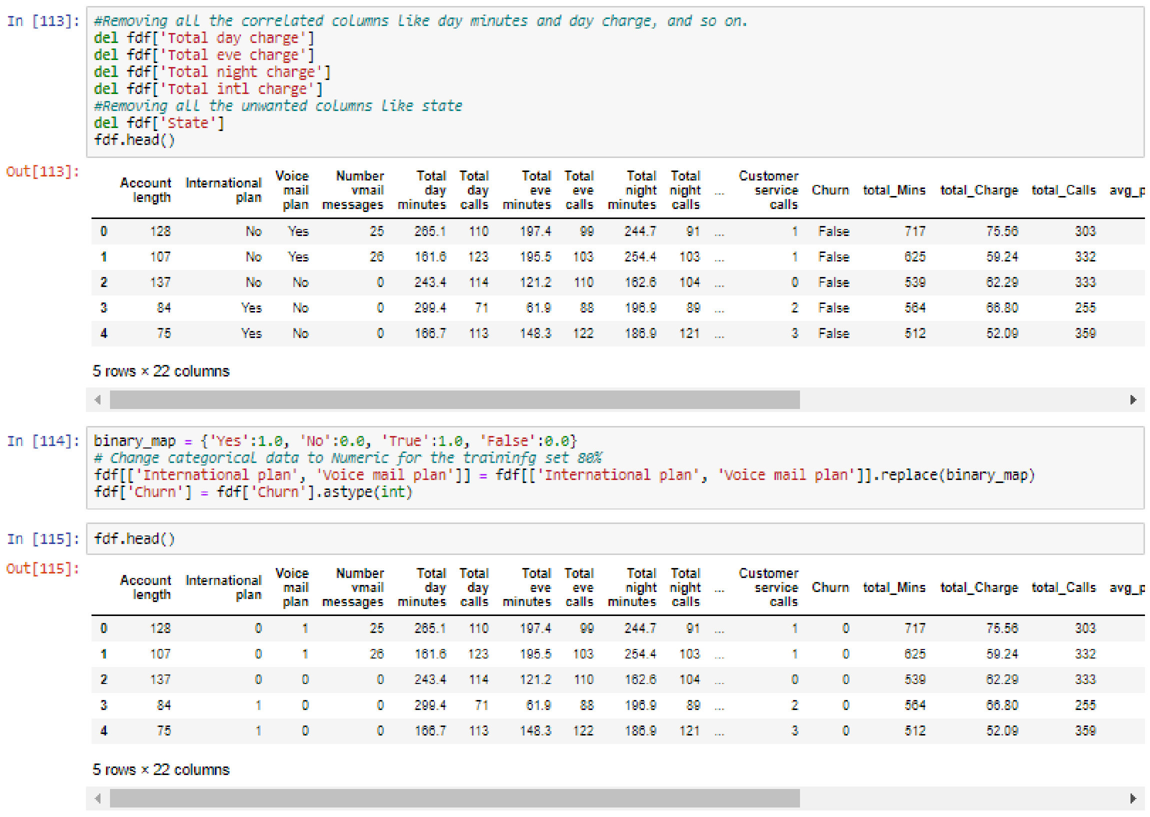


In [34]:
```
#Checking correlation matrix to eliminate the columns further
fdf.corr()
```

Out[34]:

	Account length	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	...	Customer service calls	Chur
Account length	1.000000	-0.002996	0.002847	0.038862	0.002843	-0.015923	0.018552	-0.015909	-0.008994	-0.024007	...	0.002455	0.01772
Number vmail messages	-0.002996	1.000000	0.019027	-0.009622	0.019027	0.011401	0.005131	0.011418	-0.000224	0.008124	...	-0.018787	-0.08647
Total day minutes	0.002847	0.019027	1.000000	0.016780	1.000000	0.003999	0.009059	0.003992	0.013491	0.015054	...	-0.024543	0.19568
Total day calls	0.038862	-0.009622	0.016780	1.000000	0.016787	-0.026003	0.006473	-0.026006	0.008986	-0.016776	...	-0.011945	0.01829
Total day charge	0.002843	0.019027	1.000000	0.016787	1.000000	0.004008	0.009056	0.004002	0.013495	0.015057	...	-0.024548	0.19568
Total eve minutes	-0.015923	0.011401	0.003999	-0.026003	0.004008	1.000000	-0.007654	1.000000	-0.013414	0.009017	...	-0.013192	0.07290
Total eve calls	0.018552	0.005131	0.009059	0.006473	0.009056	-0.007654	1.000000	-0.007642	-0.000175	0.000797	...	0.001058	-0.00153
Total eve charge	-0.015909	0.011418	0.003992	-0.026006	0.004002	1.000000	-0.007642	1.000000	-0.013428	0.009030	...	-0.013196	0.07289
Total night minutes	-0.008994	-0.000224	0.013491	0.008986	0.013495	-0.013414	-0.000175	-0.013428	1.000000	0.012736	...	0.005236	0.03363
Total night calls	-0.024007	0.008124	0.015054	-0.016776	0.015057	0.009017	0.000797	0.009030	0.012736	1.000000		-0.005877	0.01226

Removing highly correlated columns and making some further adjustments.



In [113]:
```
#Removing all the correlated columns like day minutes and day charge, and so on.
del fdf['Total day charge']
del fdf['Total eve charge']
del fdf['Total night charge']
del fdf['Total intl charge']
#Removing all the unwanted columns like state
del fdf['State']
fdf.head()
```

Out[113]:

	Account length	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total eve minutes	Total eve calls	Total night minutes	Total night calls	...	Customer service calls	Churn	total_Mins	total_Charge	total_Calls	avg_p
0	128	No	Yes	25	265.1	110	197.4	99	244.7	91	...	1	False	717	75.56	303	
1	107	No	Yes	26	161.6	123	195.5	103	254.4	103	...	1	False	625	59.24	332	
2	137	No	No	0	243.4	114	121.2	110	162.6	104	...	0	False	539	62.29	333	
3	84	Yes	No	0	299.4	71	61.9	88	196.9	89	...	2	False	564	66.80	255	
4	75	Yes	No	0	166.7	113	148.3	122	186.9	121	...	3	False	512	52.09	359	

5 rows × 22 columns

In [114]:
```
binary_map = {'Yes':1.0, 'No':0.0, 'True':1.0, 'False':0.0}
# Change categorical data to Numeric for the traininfg set 80%
fdf[['International plan', 'Voice mail plan']] = fdf[['International plan', 'Voice mail plan']].replace(binary_map)
fdf['Churn'] = fdf['Churn'].astype(int)
```

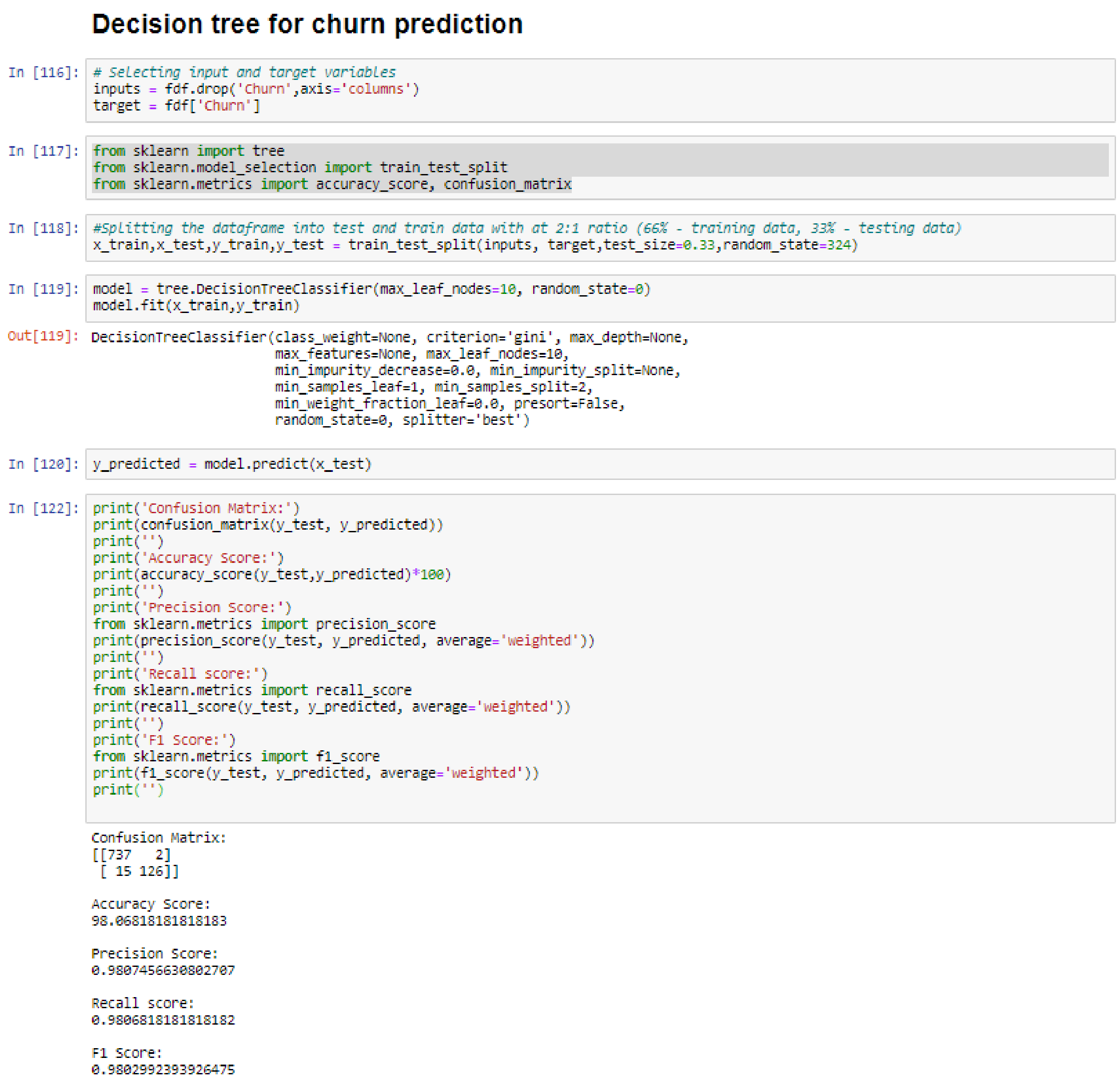
In [115]:
```
fdf.head()
```

Out[115]:

	Account length	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total eve minutes	Total eve calls	Total night minutes	Total night calls	...	Customer service calls	Churn	total_Mins	total_Charge	total_Calls	avg_p
0	128	0	1	25	265.1	110	197.4	99	244.7	91	...	1	0	717	75.56	303	
1	107	0	1	26	161.6	123	195.5	103	254.4	103	...	1	0	625	59.24	332	
2	137	0	0	0	243.4	114	121.2	110	162.6	104	...	0	0	539	62.29	333	
3	84	1	0	0	299.4	71	61.9	88	196.9	89	...	2	0	564	66.80	255	
4	75	1	0	0	166.7	113	148.3	122	186.9	121	...	3	0	512	52.09	359	

5 rows × 22 columns

After all the above adjustments, finally decision tree classifier was implemented for churn prediction and the results were evaluated:



Decision tree for churn prediction

```
In [116]: # Selecting input and target variables
          inputs = fdf.drop('Churn',axis='columns')
          target = fdf['Churn']
```

```
In [117]: from sklearn import tree
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import accuracy_score, confusion_matrix
```

```
In [118]: #Splitting the dataframe into test and train data with at 2:1 ratio (66% - training data, 33% - testing data)
          x_train,x_test,y_train,y_test = train_test_split(inputs, target,test_size=0.33,random_state=324)
```

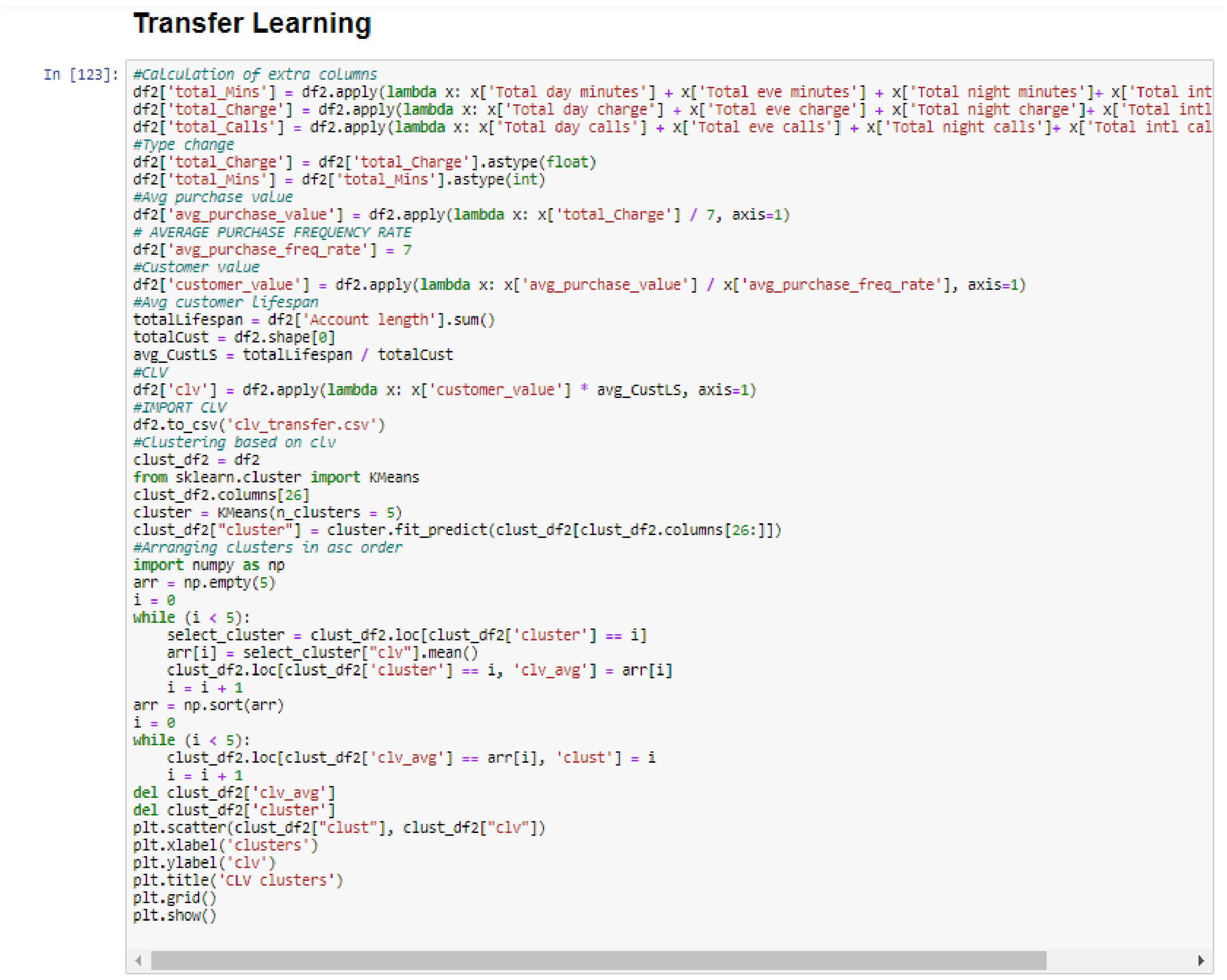
```
In [119]: model = tree.DecisionTreeClassifier(max_leaf_nodes=10, random_state=0)
          model.fit(x_train,y_train)

Out[119]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                                 max_features=None, max_leaf_nodes=10,
                                 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=0, splitter='best')
```

```
In [120]: y_predicted = model.predict(x_test)
```

```
In [122]: print('Confusion Matrix:')
          print(confusion_matrix(y_test, y_predicted))
          print('')
          print('Accuracy Score:')
          print(accuracy_score(y_test,y_predicted)*100)
          print('')
          print('Precision Score:')
          from sklearn.metrics import precision_score
          print(precision_score(y_test, y_predicted, average='weighted'))
          print('')
          print('Recall score:')
          from sklearn.metrics import recall_score
          print(recall_score(y_test, y_predicted, average='weighted'))
          print('')
          print('F1 Score:')
          from sklearn.metrics import f1_score
          print(f1_score(y_test, y_predicted, average='weighted'))
          print('')

          Confusion Matrix:
          [[737   2]
           [ 15 126]]

          Accuracy Score:
          98.06818181818183

          Precision Score:
          0.9807456630802707

          Recall score:
          0.9806818181818182

          F1 Score:
          0.9802992393926475
```

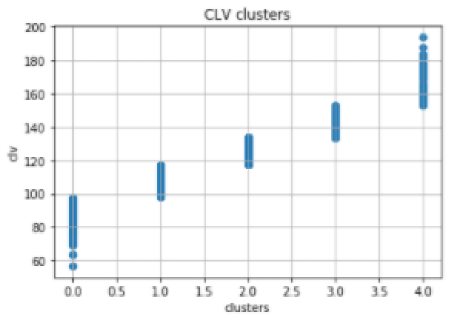
9 Transfer Learning

As done above, same adjustments were made on second dataset in order to perform transfer learning.

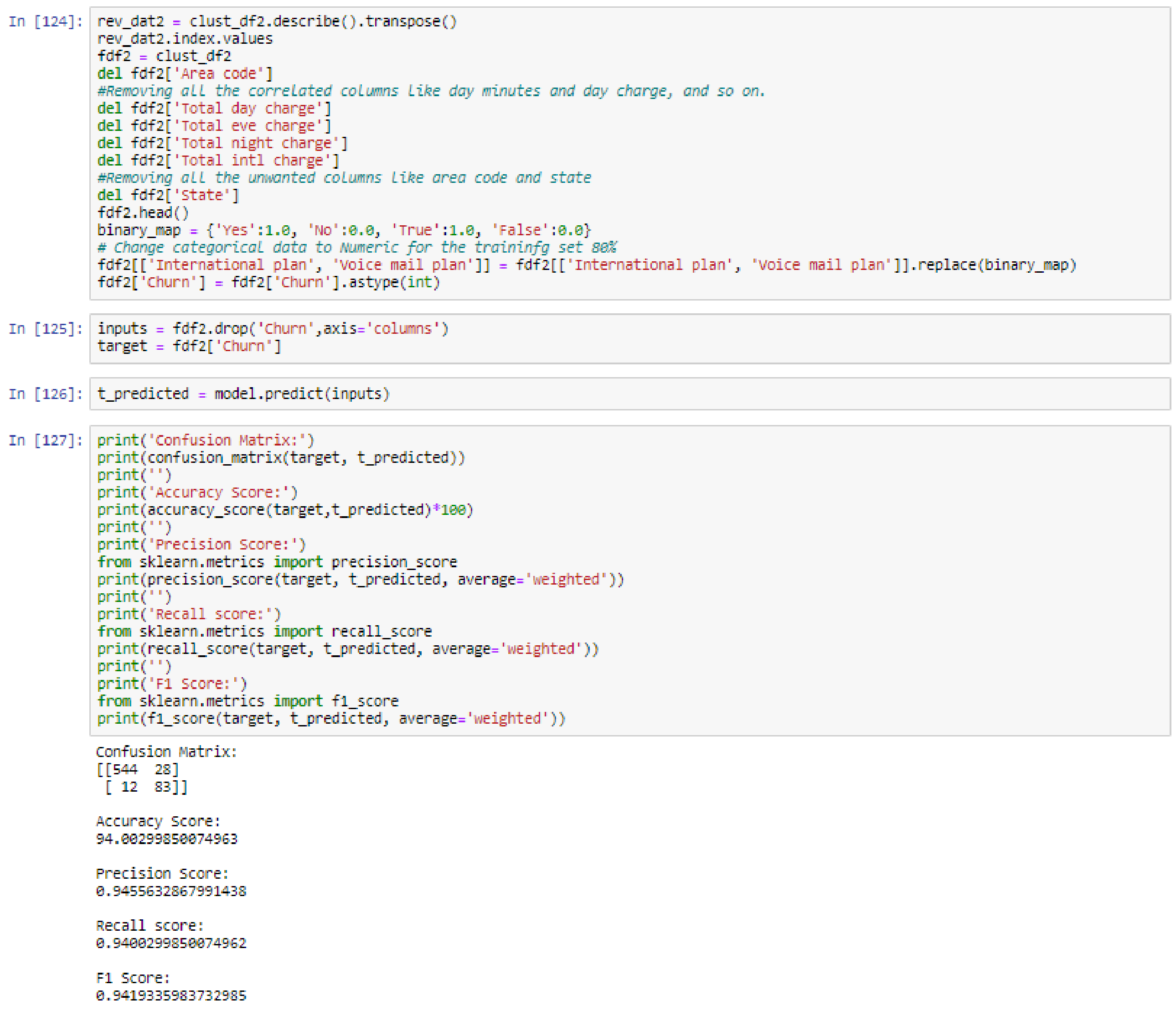


Transfer Learning

In [123]:
```
#Calculation of extra columns
df2['total_Mins'] = df2.apply(lambda x: x['Total day minutes'] + x['Total eve minutes'] + x['Total night minutes']+ x['Total int
df2['total_Charge'] = df2.apply(lambda x: x['Total day charge'] + x['Total eve charge'] + x['Total night charge']+ x['Total intl
df2['total_Calls'] = df2.apply(lambda x: x['Total day calls'] + x['Total eve calls'] + x['Total night calls']+ x['Total intl cal
#Type change
df2['total_Charge'] = df2['total_Charge'].astype(float)
df2['total_Mins'] = df2['total_Mins'].astype(int)
#Avg purchase value
df2['avg_purchase_value'] = df2.apply(lambda x: x['total_Charge'] / 7, axis=1)
# AVERAGE PURCHASE FREQUENCY RATE
df2['avg_purchase_freq_rate'] = 7
#Customer value
df2['customer_value'] = df2.apply(lambda x: x['avg_purchase_value'] / x['avg_purchase_freq_rate'], axis=1)
#Avg customer lifespan
totalLifespan = df2['Account length'].sum()
totalCust = df2.shape[0]
avg_CustLS = totalLifespan / totalCust
#CLV
df2['clv'] = df2.apply(lambda x: x['customer_value'] * avg_CustLS, axis=1)
#IMPORT CLV
df2.to_csv('clv_transfer.csv')
#Clustering based on clv
clust_df2 = df2
from sklearn.cluster import KMeans
clust_df2.columns[26]
cluster = KMeans(n_clusters = 5)
clust_df2["cluster"] = cluster.fit_predict(clust_df2[clust_df2.columns[26:]])
#Arranging clusters in asc order
import numpy as np
arr = np.empty(5)
i = 0
while (i < 5):
    select_cluster = clust_df2.loc[clust_df2['cluster'] == i]
    arr[i] = select_cluster["clv"].mean()
    clust_df2.loc[clust_df2['cluster'] == i, 'clv_avg'] = arr[i]
    i = i + 1
arr = np.sort(arr)
i = 0
while (i < 5):
    clust_df2.loc[clust_df2['clv_avg'] == arr[i], 'clust'] = i
    i = i + 1
del clust_df2['clv_avg']
del clust_df2['cluster']
plt.scatter(clust_df2["clust"], clust_df2["clv"])
plt.xlabel('clusters')
plt.ylabel('clv')
plt.title('CLV clusters')
plt.grid()
plt.show()
```



After the required adjustment is performed, the dataset is in the same format as the original dataset on which the model was trained. Now the same trained model is tested on this dataset.



```
In [124]: rev_dat2 = clust_df2.describe().transpose()
          rev_dat2.index.values
          fdf2 = clust_df2
          del fdf2['Area code']
          #Removing all the correlated columns like day minutes and day charge, and so on.
          del fdf2['Total day charge']
          del fdf2['Total eve charge']
          del fdf2['Total night charge']
          del fdf2['Total intl charge']
          #Removing all the unwanted columns like area code and state
          del fdf2['State']
          fdf2.head()
          binary_map = {'Yes':1.0, 'No':0.0, 'True':1.0, 'False':0.0}
          # Change categorical data to Numeric for the traininfg set 80%
          fdf2[['International plan', 'Voice mail plan']] = fdf2[['International plan', 'Voice mail plan']].replace(binary_map)
          fdf2['Churn'] = fdf2['Churn'].astype(int)
```

```
In [125]: inputs = fdf2.drop('Churn',axis='columns')
          target = fdf2['Churn']
```

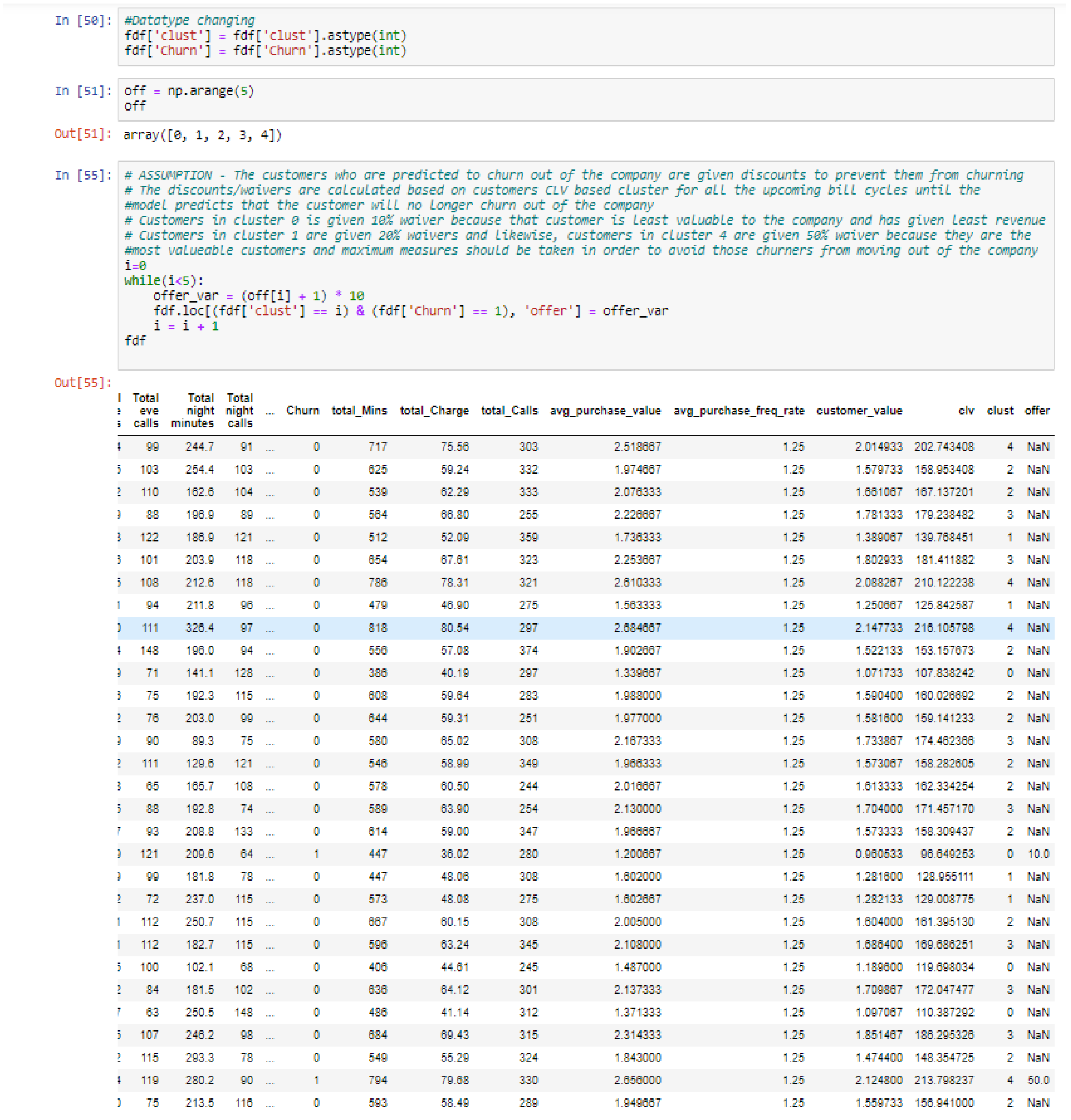
```
In [126]: t_predicted = model.predict(inputs)
```

```
In [127]: print('Confusion Matrix:')
          print(confusion_matrix(target, t_predicted))
          print('')
          print('Accuracy Score:')
          print(accuracy_score(target,t_predicted)*100)
          print('')
          print('Precision Score:')
          from sklearn.metrics import precision_score
          print(precision_score(target, t_predicted, average='weighted'))
          print('')
          print('Recall score:')
          from sklearn.metrics import recall_score
          print(recall_score(target, t_predicted, average='weighted'))
          print('')
          print('F1 Score:')
          from sklearn.metrics import f1_score
          print(f1_score(target, t_predicted, average='weighted'))
```

```
Confusion Matrix:
[[544  28]
 [ 12  83]]

Accuracy Score:
94.00299850074963

Precision Score:
0.9455632867991438

Recall score:
0.9400299850074962

F1 Score:
0.9419335983732985
```

10 Discount Model

Probable churners are mapped to their CLV scores and appropriate offers/ discounts are generated.



```
In [50]: #Datatype changing
         fdf['clust'] = fdf['clust'].astype(int)
         fdf['Churn'] = fdf['Churn'].astype(int)
```

```
In [51]: off = np.arange(5)
         off

Out[51]: array([0, 1, 2, 3, 4])
```

```
In [55]: # ASSUMPTION - The customers who are predicted to churn out of the company are given discounts to prevent them from churning
         # The discounts/waivers are calculated based on customers CLV based cluster for all the upcoming bill cycles until the
         #model predicts that the customer will no longer churn out of the company
         # Customers in cluster 0 is given 10% waiver because that customer is least valuable to the company and has given least revenue
         # Customers in cluster 1 are given 20% waivers and likewise, customers in cluster 4 are given 50% waiver because they are the
         #most valueable customers and maximum measures should be taken in order to avoid those churners from moving out of the company
         i=0
         while(i<5):
             offer_var = (off[i] + 1) * 10
             fdf.loc[(fdf['clust'] == i) & (fdf['Churn'] == 1), 'offer'] = offer_var
             i = i + 1
         fdf
```

Out[55]:

l e s	Total eve calls	Total night minutes	Total night calls	...	Churn	total_Mins	total_Charge	total_Calls	avg_purchase_value	avg_purchase_freq_rate	customer_value	clv	clust	offer
4	99	244.7	91	...	0	717	75.56	303	2.518667	1.25	2.014933	202.743408	4	NaN
5	103	254.4	103	...	0	625	59.24	332	1.974667	1.25	1.579733	158.953408	2	NaN
2	110	162.6	104	...	0	539	62.29	333	2.076333	1.25	1.661067	167.137201	2	NaN
9	88	196.9	89	...	0	564	66.80	255	2.226667	1.25	1.781333	179.238482	3	NaN
3	122	186.9	121	...	0	512	52.09	359	1.736333	1.25	1.389067	139.768451	1	NaN
3	101	203.9	118	...	0	654	67.61	323	2.253667	1.25	1.802933	181.411882	3	NaN
5	108	212.6	118	...	0	786	78.31	321	2.610333	1.25	2.088267	210.122238	4	NaN
1	94	211.8	96	...	0	479	46.90	275	1.563333	1.25	1.250667	125.842587	1	NaN
0	111	326.4	97	...	0	818	80.54	297	2.684667	1.25	2.147733	216.105798	4	NaN
4	148	196.0	94	...	0	556	57.08	374	1.902667	1.25	1.522133	153.157673	2	NaN
9	71	141.1	128	...	0	386	40.19	297	1.339667	1.25	1.071733	107.838242	0	NaN
3	75	192.3	115	...	0	608	59.64	283	1.988000	1.25	1.590400	160.026692	2	NaN
2	76	203.0	99	...	0	644	59.31	251	1.977000	1.25	1.581600	159.141233	2	NaN
9	90	89.3	75	...	0	580	65.02	308	2.167333	1.25	1.733867	174.462366	3	NaN
2	111	129.6	121	...	0	546	58.99	349	1.966333	1.25	1.573067	158.282605	2	NaN
3	65	165.7	108	...	0	578	60.50	244	2.016667	1.25	1.613333	162.334254	2	NaN
5	88	192.8	74	...	0	589	63.90	254	2.130000	1.25	1.704000	171.457170	3	NaN
7	93	208.8	133	...	0	614	59.00	347	1.966667	1.25	1.573333	158.309437	2	NaN
9	121	209.6	64	...	1	447	36.02	280	1.200667	1.25	0.960533	96.649253	0	10.0
9	99	181.8	78	...	0	447	48.06	308	1.602000	1.25	1.281600	128.955111	1	NaN
2	72	237.0	115	...	0	573	48.08	275	1.602667	1.25	1.282133	129.008775	1	NaN
1	112	250.7	115	...	0	667	60.15	308	2.005000	1.25	1.604000	161.395130	2	NaN
1	112	182.7	115	...	0	596	63.24	345	2.108000	1.25	1.686400	169.686251	3	NaN
5	100	102.1	68	...	0	406	44.61	245	1.487000	1.25	1.189600	119.698034	0	NaN
2	84	181.5	102	...	0	636	64.12	301	2.137333	1.25	1.709867	172.047477	3	NaN
7	63	260.5	148	...	0	486	41.14	312	1.371333	1.25	1.097067	110.387292	0	NaN
5	107	246.2	98	...	0	684	69.43	315	2.314333	1.25	1.851467	186.295326	3	NaN
2	115	293.3	78	...	0	549	55.29	324	1.843000	1.25	1.474400	148.354725	2	NaN
4	119	280.2	90	...	1	794	79.68	330	2.656000	1.25	2.124800	213.798237	4	50.0
0	75	213.5	116	...	0	593	58.49	289	1.949667	1.25	1.559733	156.941000	2	NaN