Telecom Customers Churn

By : Mohamed Abdelkader
Omar Sabri Marwan Ashref Farag

The Outlines :

- Import The Libraries
- Load The Dataset
- EDA
- Feature Engineering
- Modeling
- Clustering
- Hierarchical Clustering
- PCA

The Problems we faced :

- The first problem we faced was feature extraction: deciding which columns to use.
- The second problem was encoding the object columns.
- The third problem was improving the accuracy.

1) The Libraries :

```
# 1. to handle the data
import pandas as pd
import numpy as np
from scipy import stats

# to visualize the data
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# To preprocess the data
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder,OneHotEncoder
from sklearn.impute import SimpleImputer, KNNImputer
# import iterative imputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# machine learning
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
#for classification tasks
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, RandomForestRegressor
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import  ExtraTreesClassifier
from sklearn.linear_model import  SGDClassifier
# pipeline
from sklearn.pipeline import Pipeline
# metrics
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, mean_absolute_error,mean_squared_error,r2_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import precision_recall_curve
```

2) Load The Data :

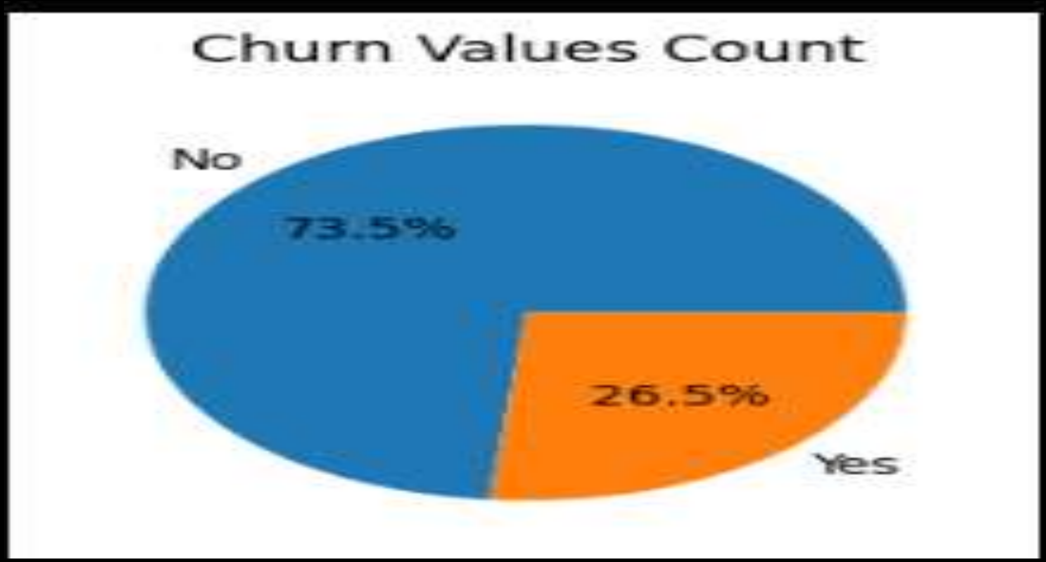
Here we wrote the [info()] to see all the information about our dataset , we noticed that the shape of the data is (7043 , 12) and 1 column float ,2 columns integer and 18 columns is object .

```
telecom.info()
✓ 0.0s
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

3) EDA

- This image shows us two colorful pie charts. The first one is about customer churn - it tells us that most customers (73.5%) are sticking around, while about a quarter (26.5%) are leaving which mean that there are in imbalance in the data.

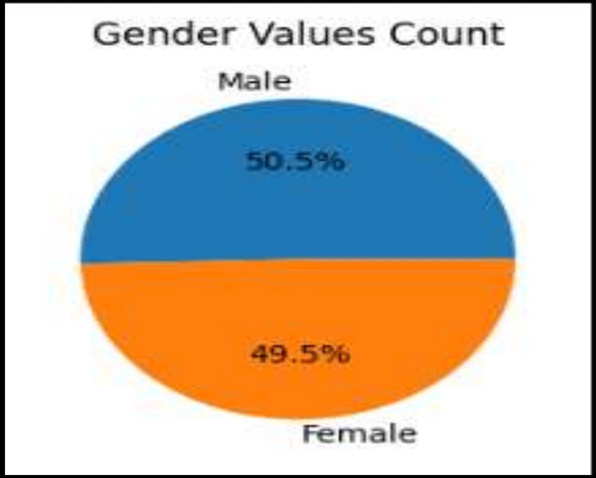


3) EDA

- The second chart is about gender, showing it's pretty much split down the middle between men and women customers which mean that there are a balance in that column

```
gender_values = telecom['gender'].value_counts()
plt.figure(figsize=(3,3))
plt.pie(gender_values,labels=gender_values.index,autopct='%1.1f%%')
plt.title('Gender Values Count')
plt.show()
```

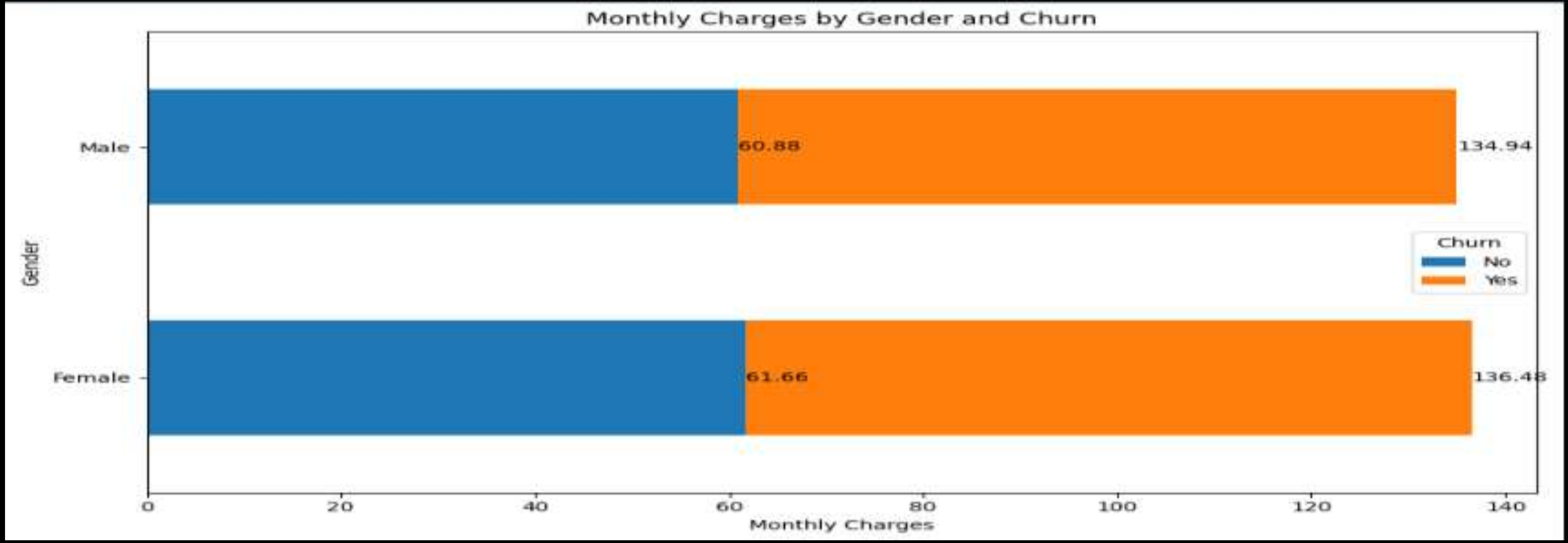
0.0s



3) EDA

- This image is a bar chart comparing how much men and women pay monthly, and whether they're likely to leave (churn) or stay. It's interesting because it shows that customers who leave tend to pay more, regardless of gender. The difference is pretty big too - about twice as much!

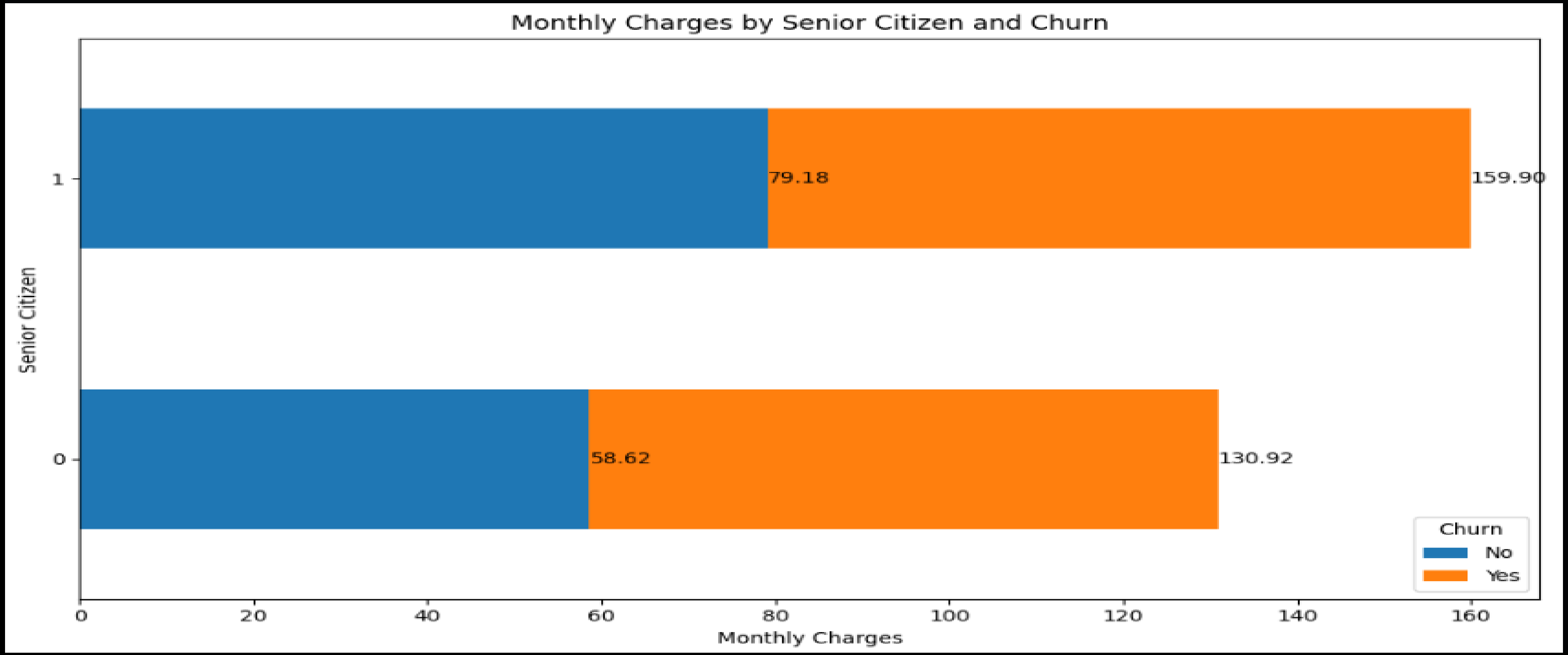
```
grouby_gender_churn=telecom.groupby(['gender','Churn'])['MonthlyCharges'].mean().unstack().fillna(0)
ax = grouby_gender_churn.plot(kind='barh', stacked=True, figsize=(10, 6))
ax.set_xlabel('Monthly Charges')
ax.set_ylabel('Gender')
ax.set_title('Monthly Charges by Gender and Churn')
for i in ax.containers:
    ax.bar_label(i, fmt='%.2f')
plt.tight_layout()
plt.show()
```
0.1s



3) EDA

- This image is similar, but it's comparing senior citizens to everyone else. Again, we see that people who leave are paying a lot more, and this is true for both groups. Senior citizens who leave are paying the most of all.

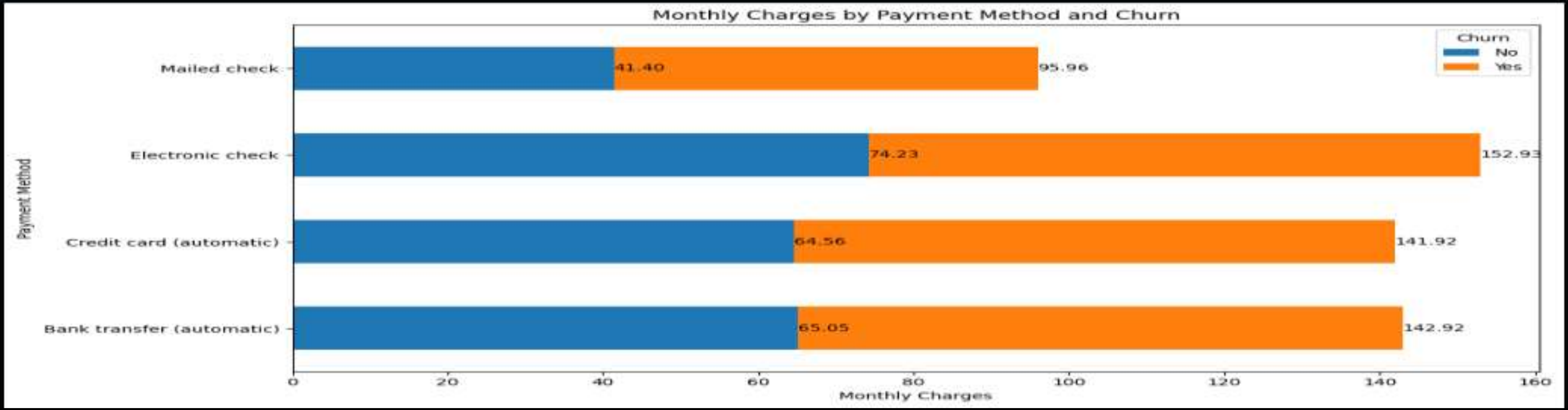
```
groupby_senior_churn=telecom.groupby(['SeniorCitizen','Churn'])['MonthlyCharges'].mean().unstack().fillna(0)
ax = groupby_senior_churn.plot(kind='barh', stacked=True, figsize=(10, 6))
ax.set_xlabel('Monthly Charges')
ax.set_ylabel('Senior Citizen')
ax.set_title('Monthly Charges by Senior Citizen and Churn')
for i in ax.containers:
    ax.bar_label(i, fmt='%.2f')
plt.tight_layout()
plt.show()
```
✓ 0.1s



3)EDA

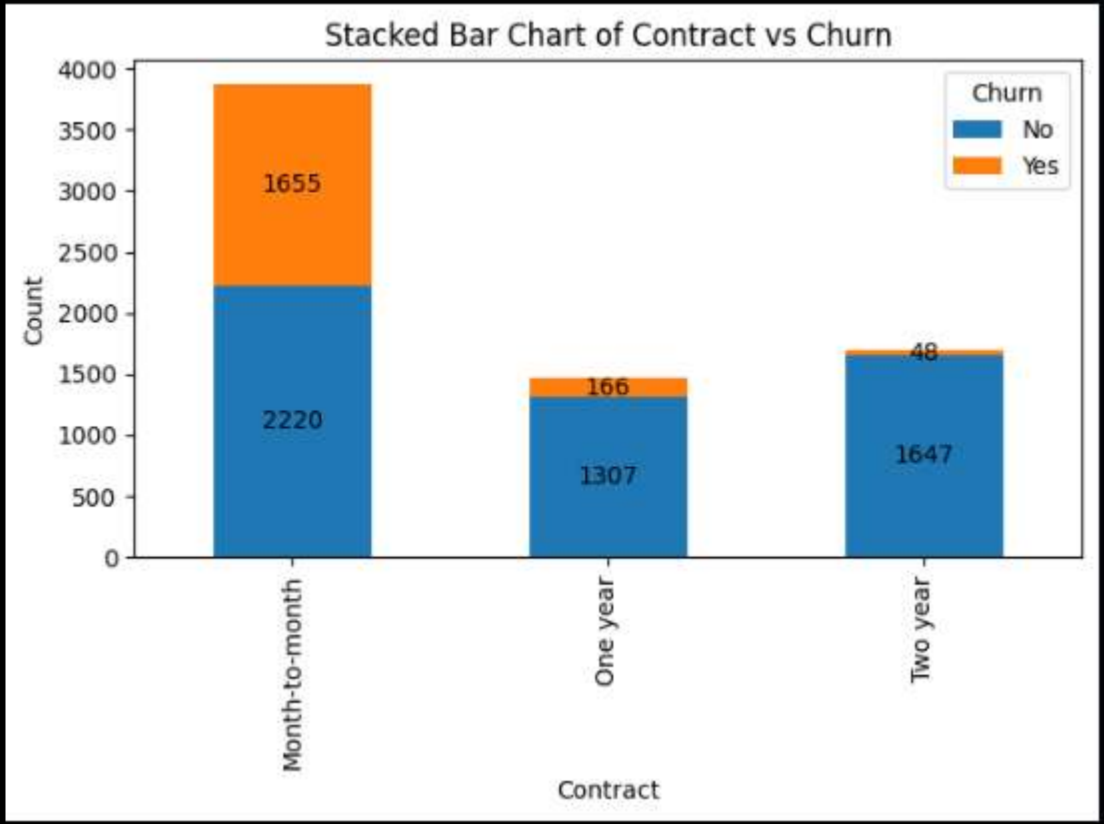
- This visualization helps in understanding how payment methods relate to monthly charges and customer churn, which could be valuable for a telecom company's customer retention strategies.

```python
groupby_payment_churn=telecom.groupby(['PaymentMethod','Churn'])['MonthlyCharges'].mean().unstack().fillna(0)
ax = groupby_payment_churn.plot(kind='barh', stacked=True, figsize=(12, 6))
ax.set_xlabel('Monthly Charges')
ax.set_ylabel('Payment Method')
ax.set_title('Monthly Charges by Payment Method and Churn')
for i in ax.containers:
    ax.bar_label(i, fmt='%.2f')
plt.tight_layout()
plt.show()
```
0.1s



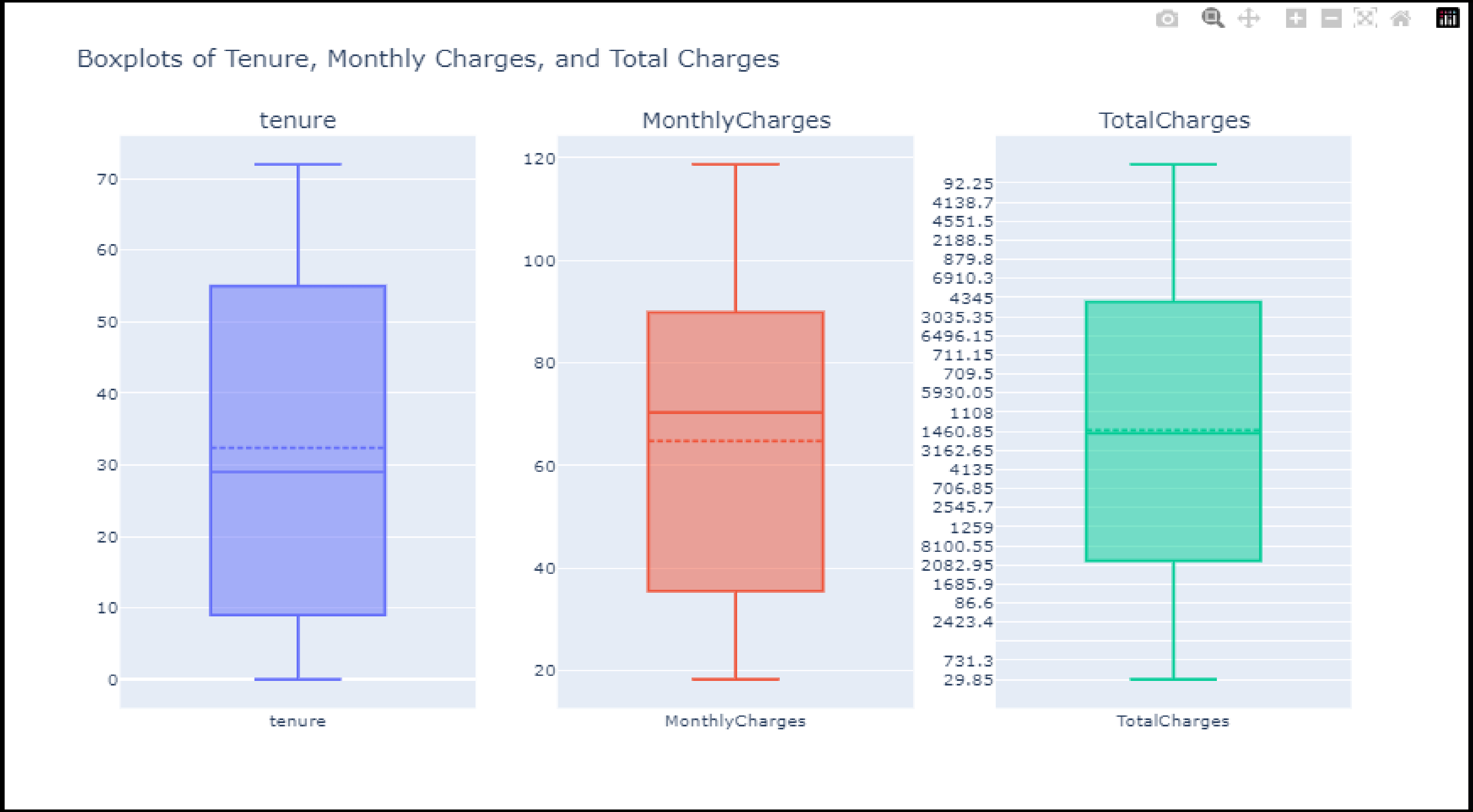
3) EDA

- This visualization suggests that longer-term contracts are more effective at retaining customers, while month-to-month contracts, despite being popular, are more prone to customer churn.



3) EDA

- This visualize aim to detect the outliers by ploty library



Feature Engineering

Part -1

4) Feature Engineering

1) This code aim to encoding the categorical columns to numerical by [Label Encoder]

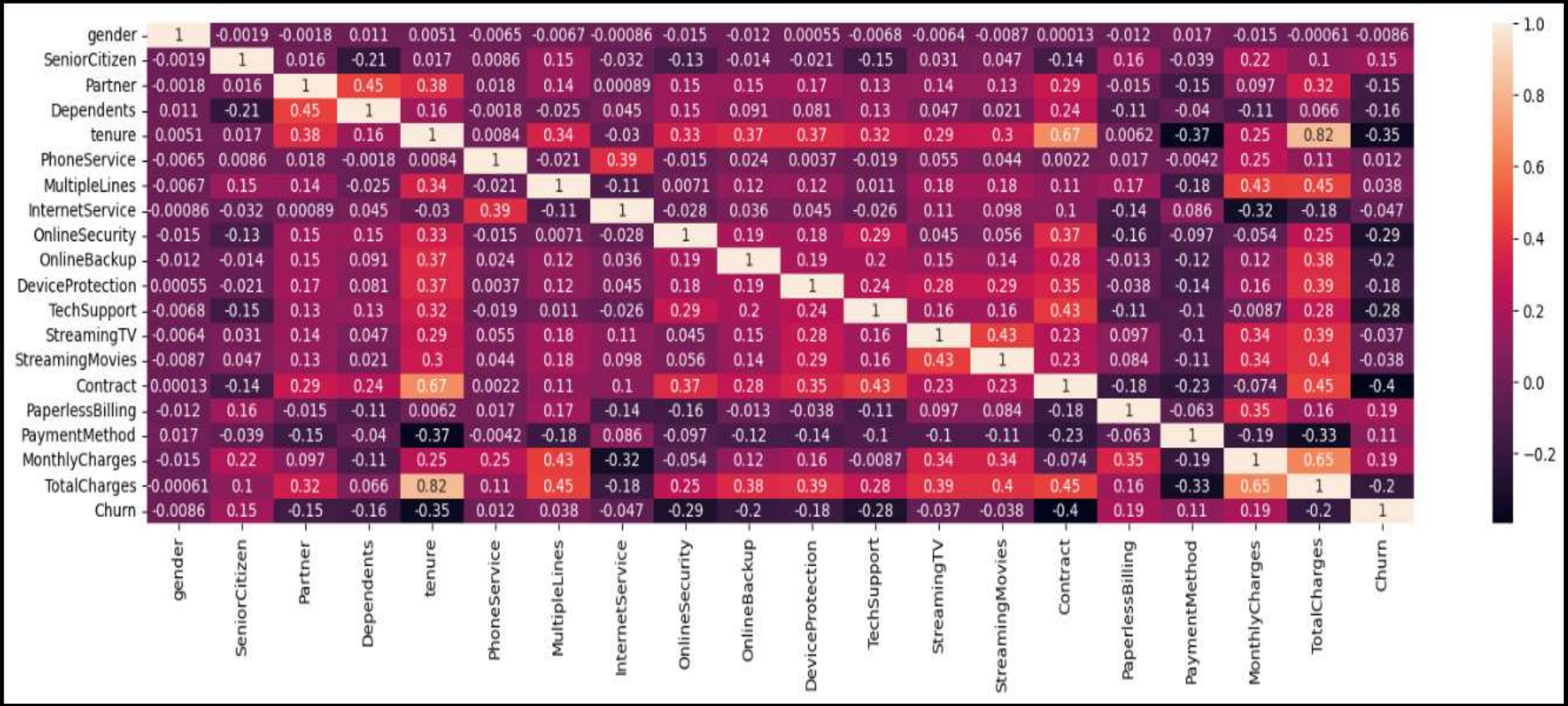
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

ds = telecom.copy(deep = True)
text_data_features = [i for i in list(telecom.columns) if i not in list(telecom.describe().columns)]

print('Label Encoder Transformation')
for i in text_data_features :
    ds[i] = le.fit_transform(ds[i])
    print(i,' : ',ds[i].unique(),' = ',le.inverse_transform(ds[i].unique()))
```

2) This code aim to normalize the numerical columns by [MinMix Scaler]

```
from sklearn.preprocessing import MinMaxScaler,StandardScaler
mms = MinMaxScaler() # Normalization
ss = StandardScaler() # Standardization

ds['tenure'] = mms.fit_transform(ds[['tenure']])
ds['MonthlyCharges'] = mms.fit_transform(ds[['MonthlyCharges']])
ds['TotalCharges'] = mms.fit_transform(ds[['TotalCharges']])
```

4)Feature Engineering

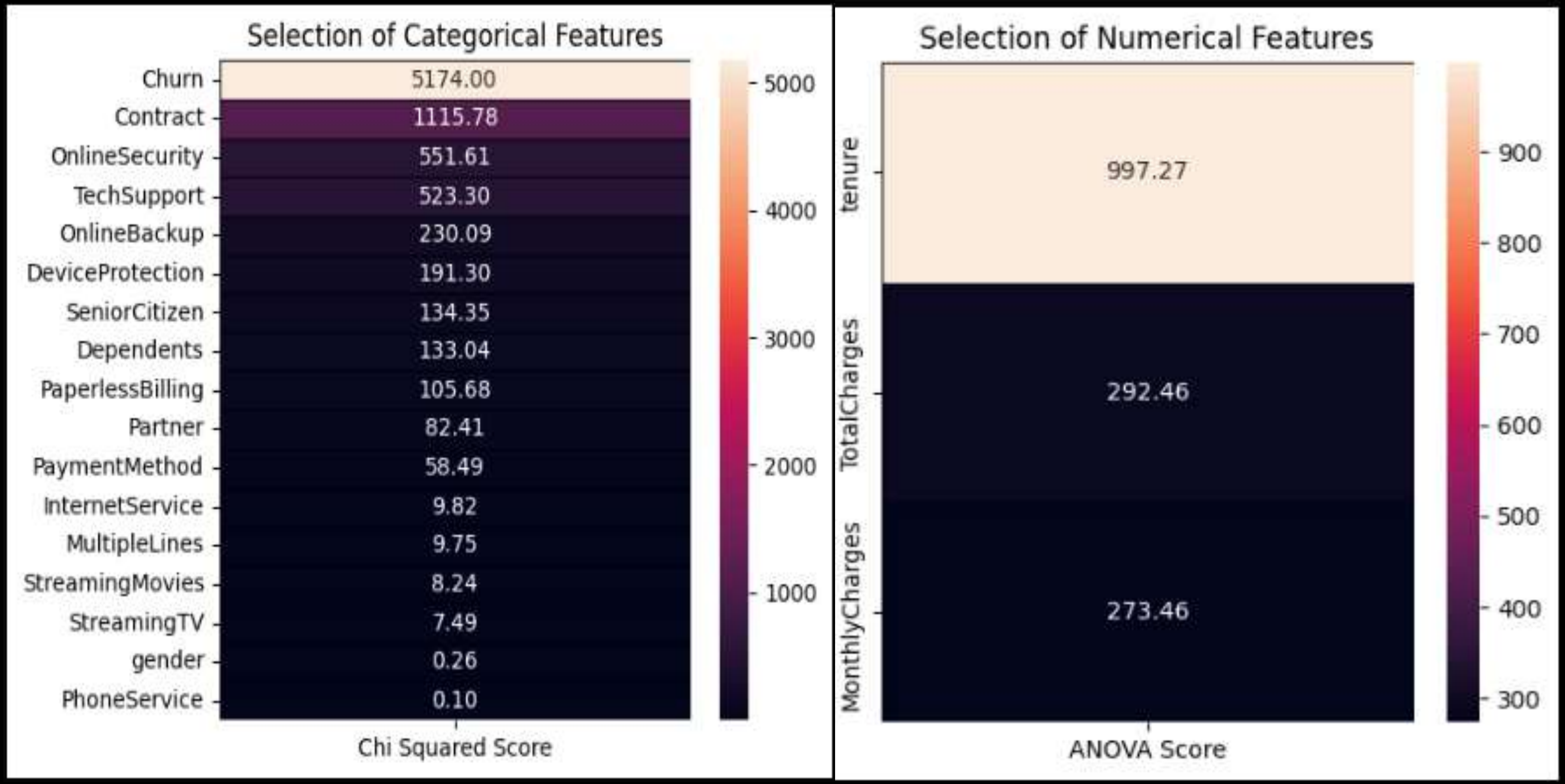
This is the correlation matrix after encoding and normalize the data



4) Feature Engineering

This Code aim to sperate the catigorical columns and the numerical columns to know the importance of each column to the target column

```
col = list(ds.columns)
categorical_features = []
numerical_features = []
for i in col:
    if len(telecom[i].unique()) > 6:
        numerical_features.append(i)
    else:
        categorical_features.append(i)
```



4)Feature Engineering

And here is the results

So we dropped Some columns :

['PhoneService', 'gender','StreamingTV','StreamingMovies','MultipleLines']

4) Feature Engineering

Part -2 : Feature Extraction

4) Feature Engineering

- *1) Monthly Charge With Tenure Feature*
- creates a new feature in the dataset that multiplies the monthly charges (MonthlyCharges) by the number of months the customer has been with the company (tenure).
- By multiplying these two columns, the new feature MonthlyChargeWithTenure provides an estimate of the total amount the customer has been charged over the period of their tenure. This value may help to identify long-term customers who have been paying higher amounts, which could influence their likelihood of churning or staying with the company.

```
ds['MonthlyChargeWithTenure']=ds['MonthlyCharges']*ds['tenure']
```

- *2) Contract Length Feature*
- Convert the Contract type into a numerical value to represent the length of the contract. Longer contracts might correlate with lower churn.

```
def ContractLength(df):
  if df['Contract']==0:
    return 1
  elif df['Contract']==1:
    return 12
  else:
    return 24

ds['ContractLength']=ds.apply(ContractLength,axis=1)
```

```python
    contract = df['Contract']
    tenure = df['tenure']

    if contract == 0:
        if tenure <= 6:
            return 'High'
        elif 7 <= tenure <= 12:
            return 'Medium'
        else:
            return 'Low'
    elif contract == 1:
        if tenure <= 6:
            return 'Medium'
        elif 7 <= tenure <= 12:
            return 'Low'
        else:
            return 'Very Low'
    elif contract == 2:
        if tenure <= 12:
            return 'Low'
        else:
            return 'Very Low'

ds['ContractTenureRisk'] = ds.apply(calculate_contract_tenure_risk, axis=1)
```

4) Feature Extraction

3) Contract Tenure Risk Feature

Create a new feature that reflects whether a customer is more likely to churn based on their contract type and tenure.

Month-to-month with low tenure could indicate high churn risk, while a two-year contract with long tenure might indicate low risk.

4) Feature Engineering

To solve the imbalance column [Chrun] we used [SMOTEENN] from imblearn library.

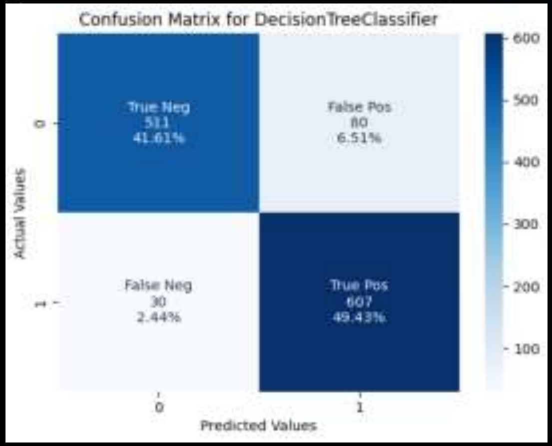
```
from imblearn.combine import SMOTEENN
# UpSampling
X = ds.drop('Churn',axis=1)
y = ds['Churn']

sm = SMOTEENN()
X_res, y_res = sm.fit_resample(X, y)

Xr_train, Xr_test, yr_train, yr_test = train_test_split(X_res, y_res, test_size=0.2)
```

5) Modeling

5) Modeling

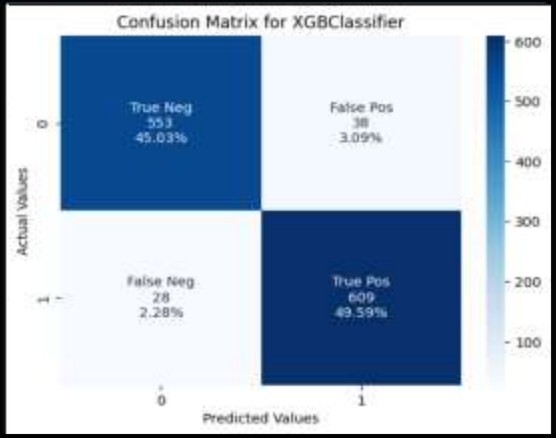
We used to many models to see which one the best for this data , we used Decision Tree :

```
Best parameters for Decision Tree: {'max_depth': 6, 'min_samples_leaf': 4, 'min_samples_split': 10}
Best score for Decision Tree: 96.44%
Cross Validation Score :  96.43%
ROC_AUC Score :  90.95%
```



And XGBoost :

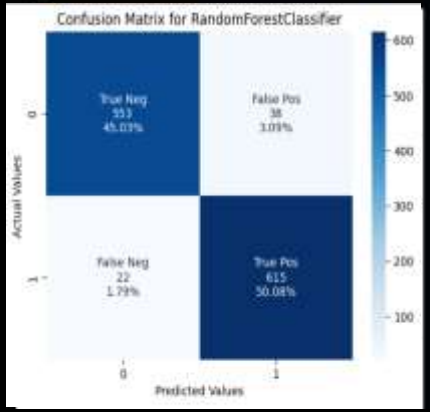
```
Running GridSearchCV for XGBoost classifier...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for XGBoost: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 500}
Best score for XGBoost: 98.61%
Cross Validation Score :  98.70%
ROC_AUC Score :  93.83%
```



5) Modeling

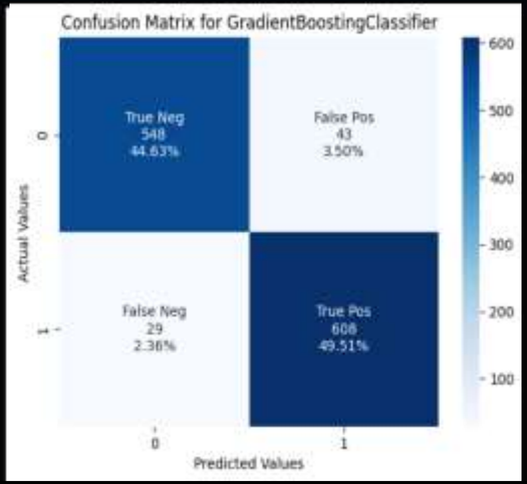
3) Random forest :

```
Running GridSearchCV for Random Forest classifier...
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best parameters for Random Forest: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best score for Random Forest: 98.91%
Cross Validation Score :  98.98%
ROC_AUC Score :  94.65%
```



4) Gradient Boosting :

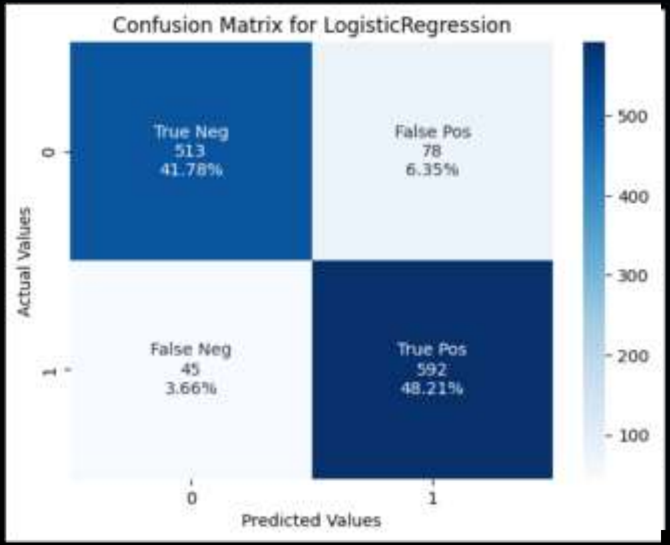
```
Running GridSearchCV for Gradient Boosting classifier...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for Gradient Boosting: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 500}
Best score for Gradient Boosting: 98.65%
Cross Validation Score :  98.75%
ROC_AUC Score :  94.33%
```



5) Modeling

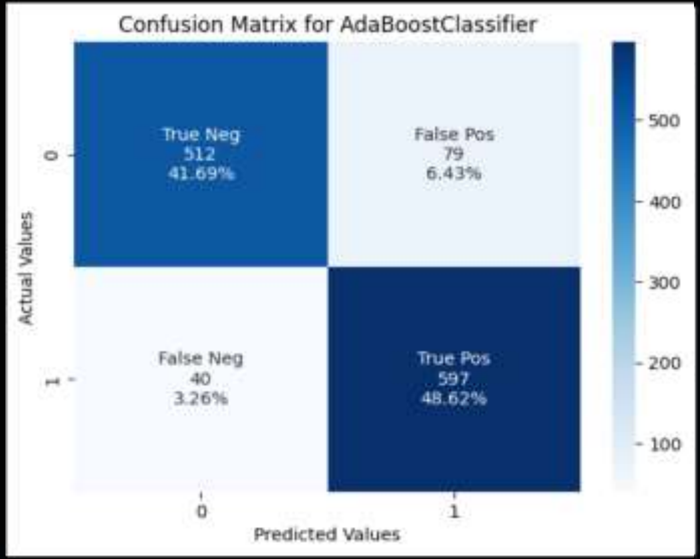
5) Logistic Regression :

```
Running GridSearchCV for Logistic Regression classifier...
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Best parameters for Logistic Regression: {'C': 10, 'penalty': 'l2'}
Best score for Logistic Regression: 96.64%
Cross Validation Score :  96.62%
ROC_AUC Score :  90.13%
```



6) Ada Boost :

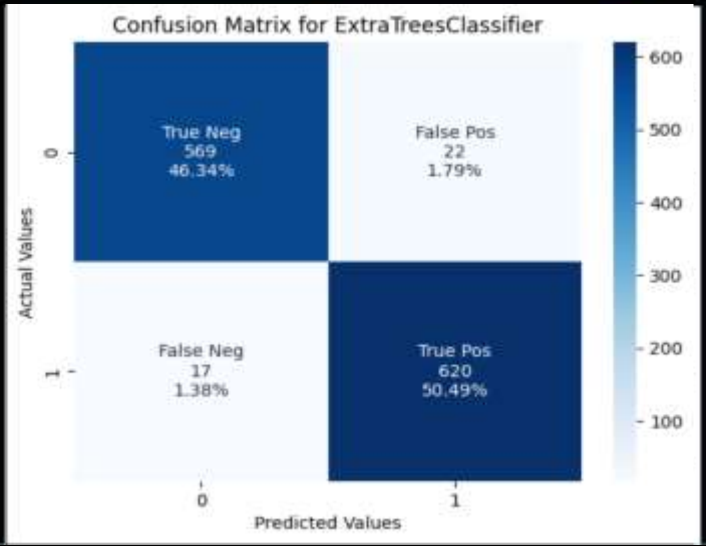
```
Running GridSearchCV for AdaBoost classifier...
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters for AdaBoost: {'learning_rate': 1, 'n_estimators': 50}
Best score for AdaBoost: 96.77%
Cross Validation Score :  96.74%
ROC_AUC Score :  90.78%
```



5) Modeling

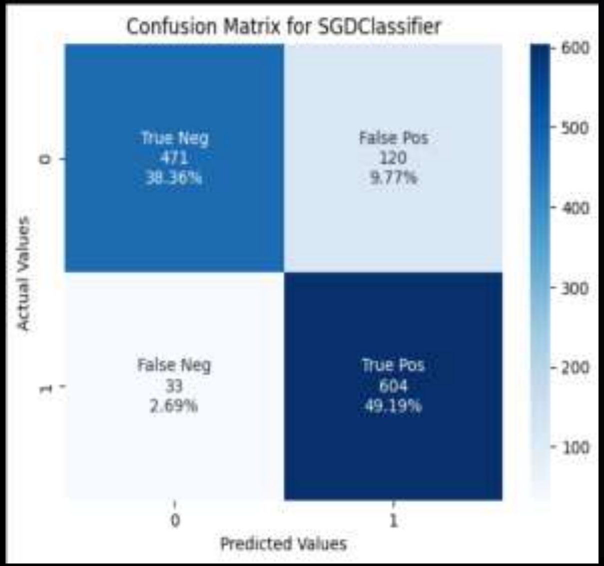
7) Extra Trees classifier :

```
Running GridSearchCV for Extra Trees classifier...
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters for Extra Trees: {'max_depth': None, 'n_estimators': 200}
Best score for Extra Trees: 99.41%
Cross Validation Score :  99.53%
ROC_AUC Score :  96.30%
```



8) SGD Classifier :

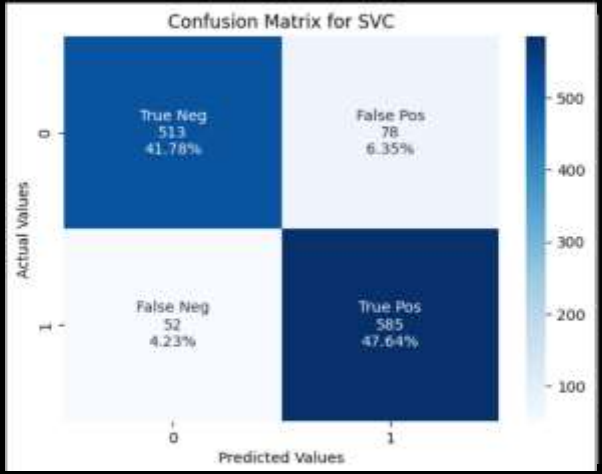
```
Running GridSearchCV for SGD Classifier classifier...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for SGD Classifier: {'alpha': 0.001, 'loss': 'hinge', 'penalty': 'l1'}
Best score for SGD Classifier: 96.46%
Cross Validation Score :  96.37%
ROC_AUC Score :  89.26%
```



5) Modeling

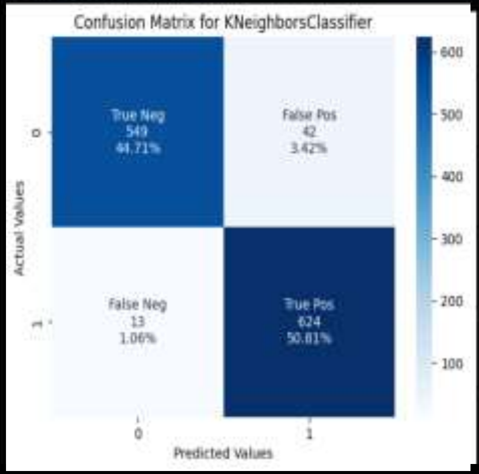
9) SVC :

```
Running GridSearchCV for Support Vector Classifier (SVC) classifier...
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters for Support Vector Classifier (SVC): {'C': 10, 'kernel': 'linear'}
Best score for Support Vector Classifier (SVC): 96.57%
Cross Validation Score :  96.52%
ROC_AUC Score :  90.29%
```



10) KNN :

```
Running GridSearchCV for K-Nearest Neighbors classifier...
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters for K-Nearest Neighbors: {'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'distance'}
Best score for K-Nearest Neighbors: 98.57%
Cross Validation Score :  98.97%
ROC_AUC Score :  96.04%
```



6) Clustering

6) Clustering

Here is the libraries that used in the clustering

Clustering

```python
## Import The important libraries :
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

6) Clustering

- This is the columns that we used it in the clustering

```
X_Cluster=telecom[['SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'InternetService',
        'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
        'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
        'MonthlyChargeWithTenure', 'ContractLength', 'ContractTenureRisk']]
```

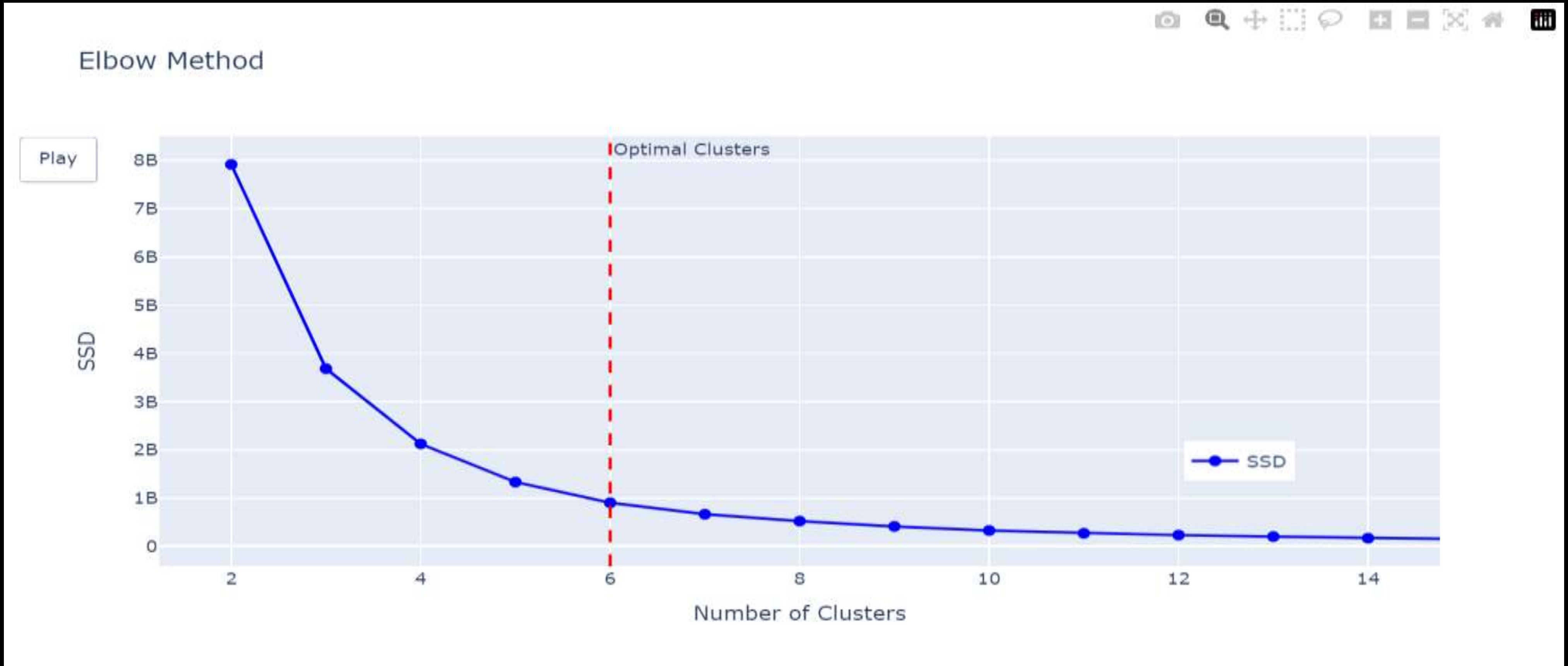
6) Clustering

- This is the elbow method function that we created to get the optimal (n) for clustering

```
## Elbow method :
InertiaDict={}
for i in range(2,16):
    KMeansModel = KMeans(n_clusters=i, init='k-means++', algorithm= 'lloyd', random_state=33) # ,
    KMeansModel.fit(X_Cluster)
    InertiaDict[i]=KMeansModel.inertia_
```

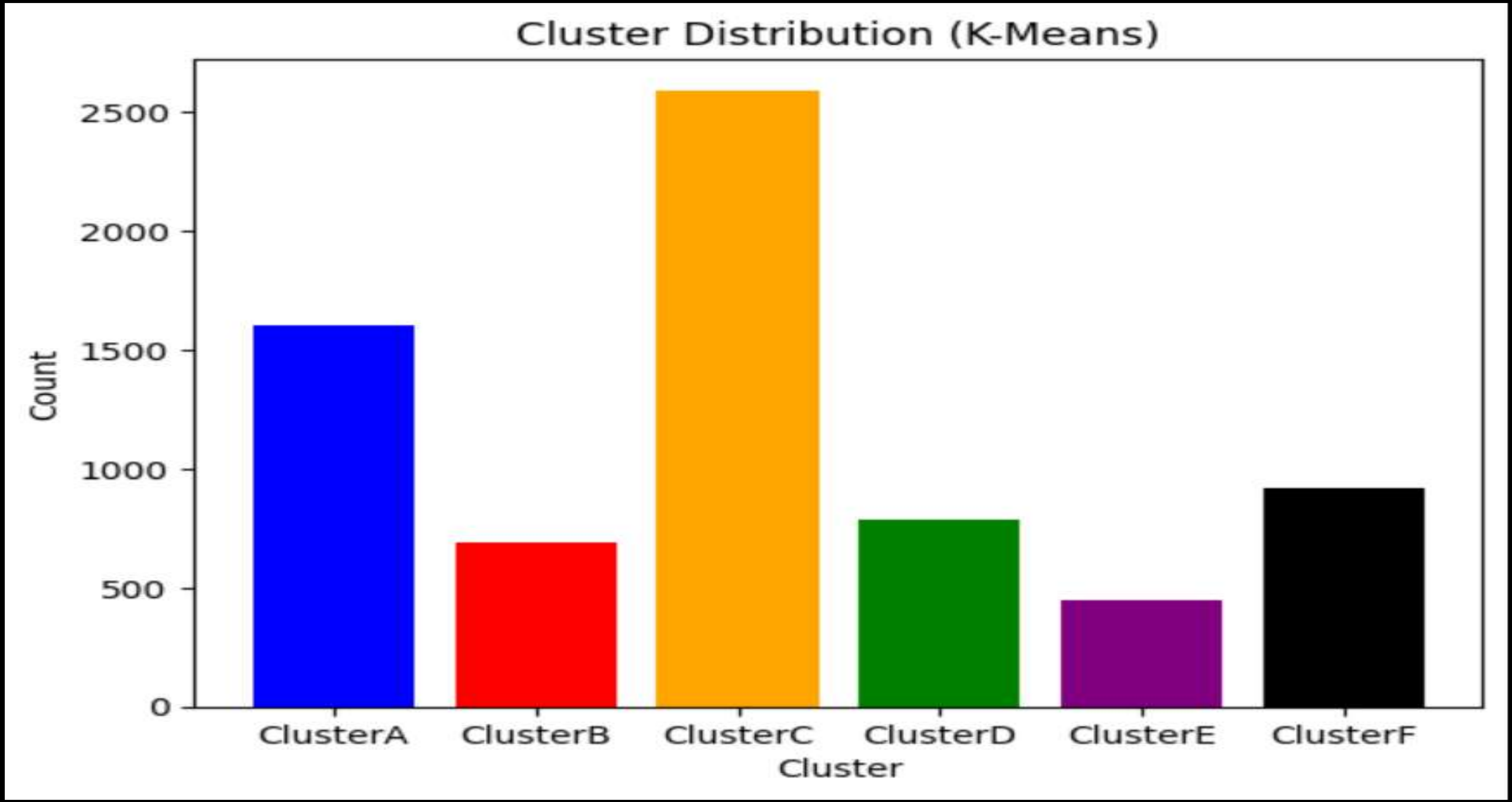
6) Clustering

- And we found that the optimal (n) is 6



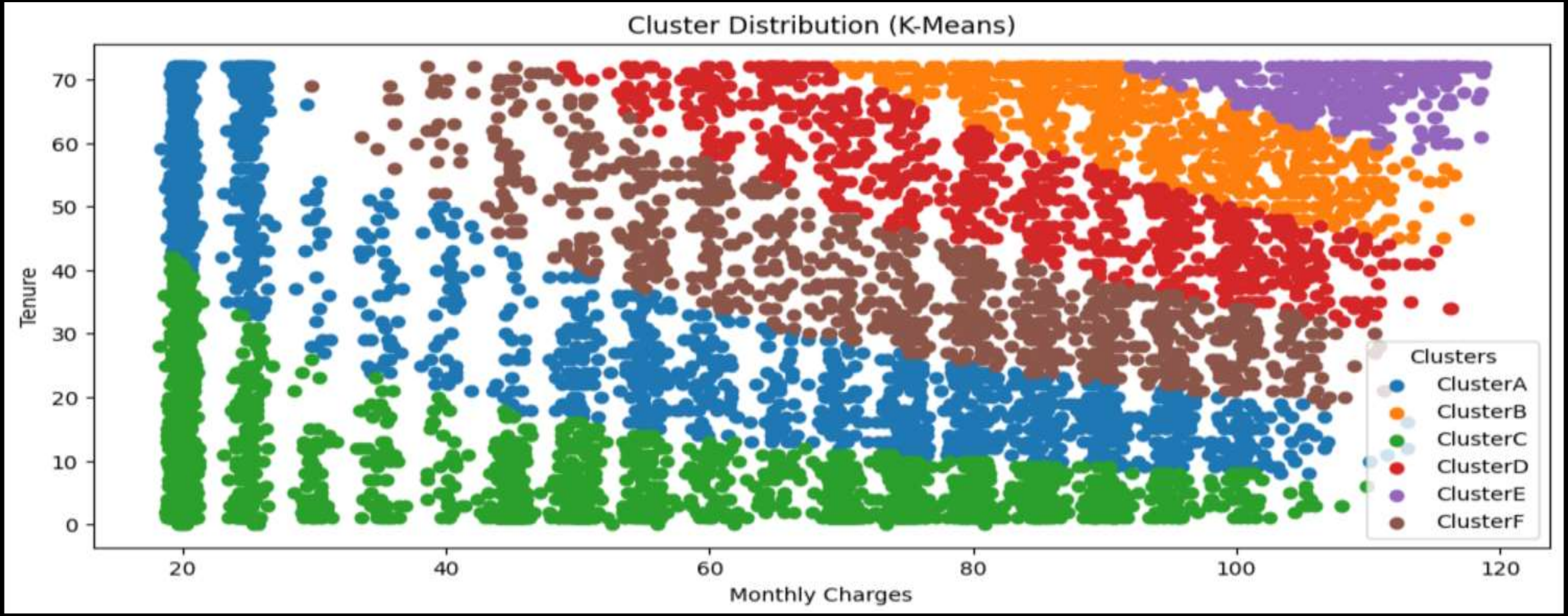
6) Clustering

- This is the distribution for the 6 clusters :



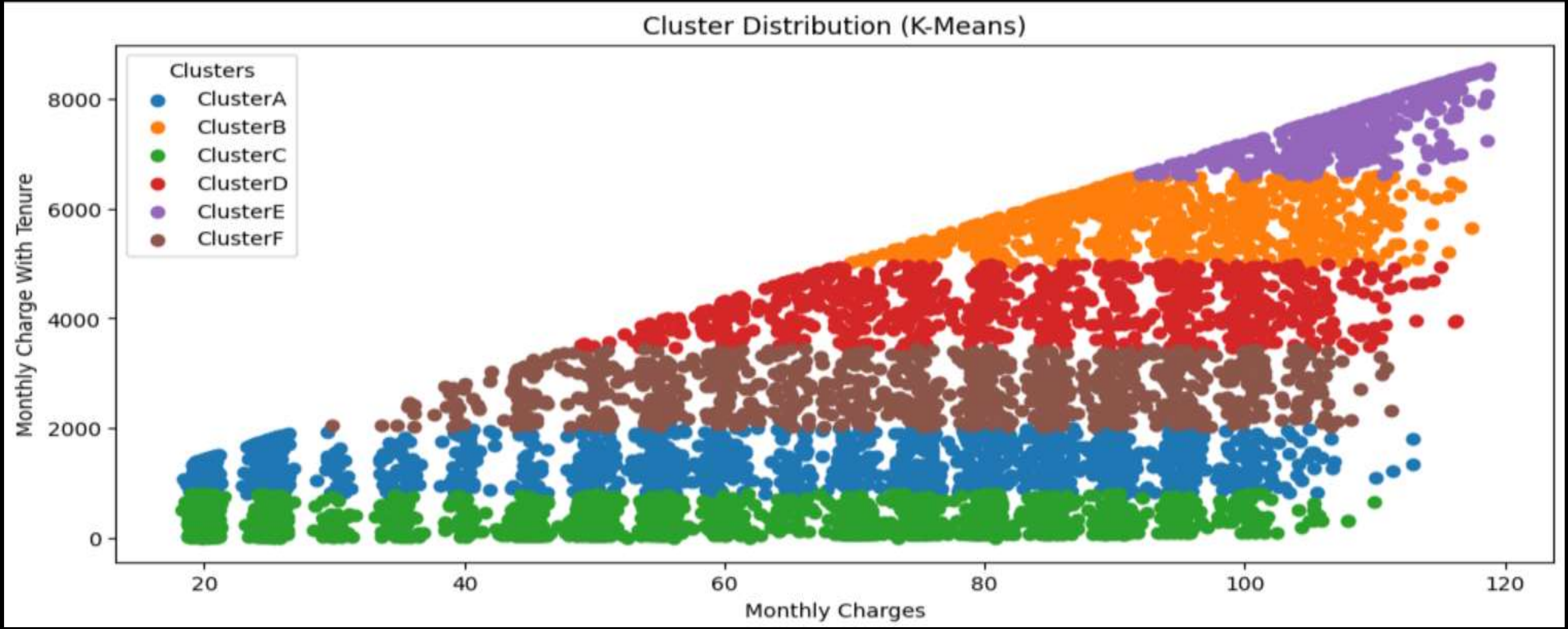
6) Clustering

- And this is the cluster about monthlycharges & tenure which each one in a column :



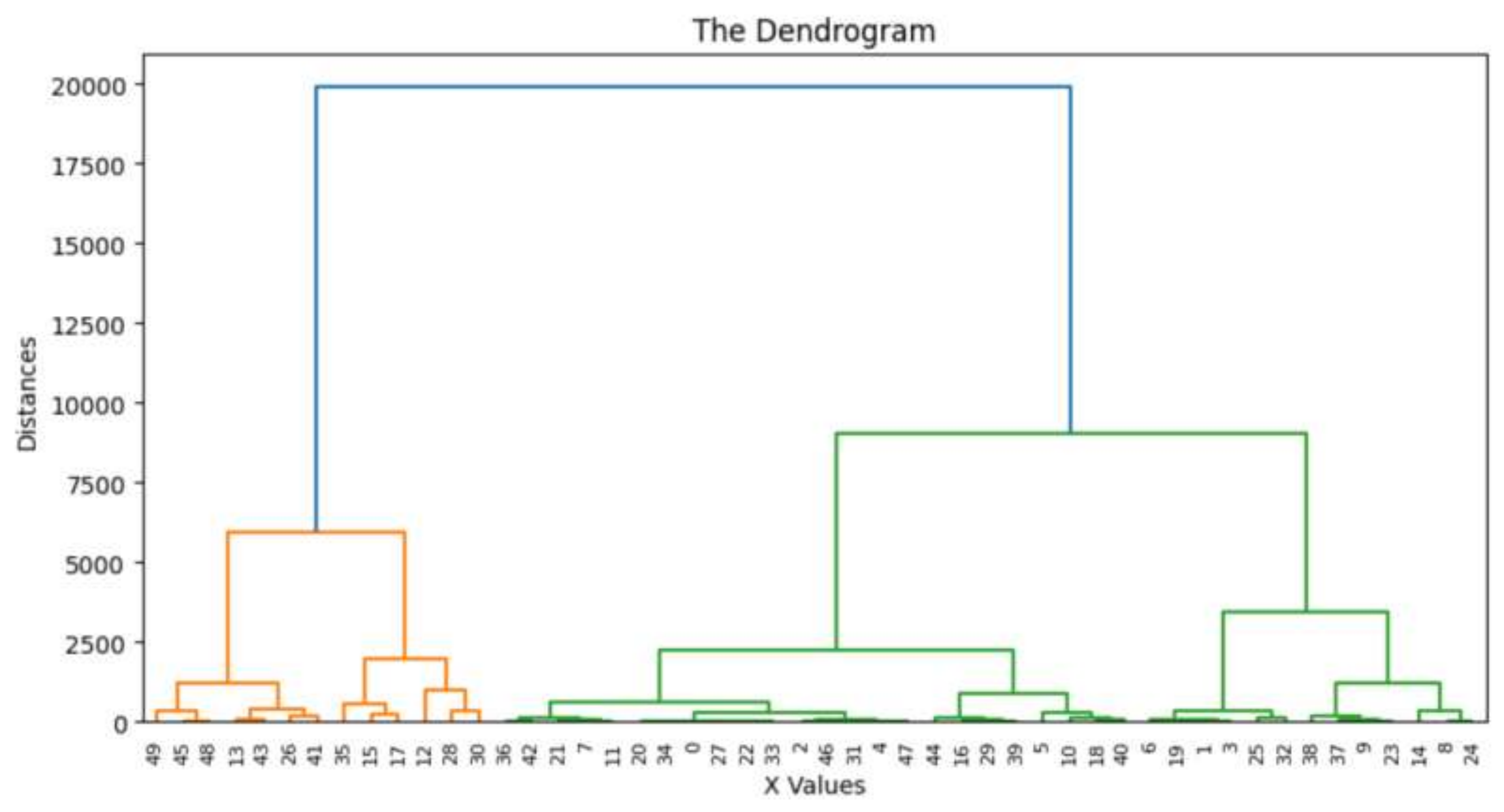
6) Clustering

- And this is the cluster about monthlycharges & tenure in one column :



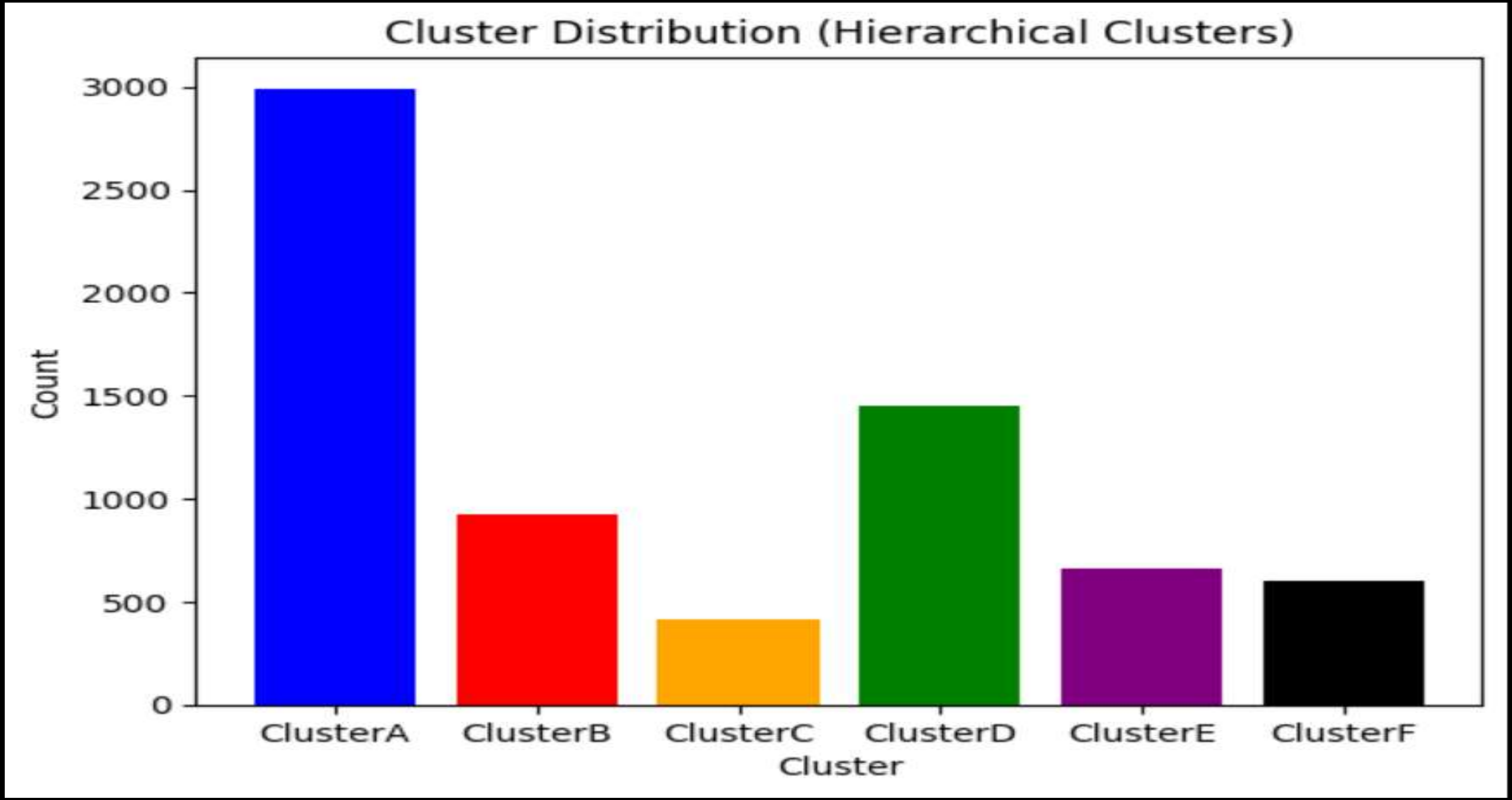
7) Hierarchical Clusters

- This is the Hierarchical Clusters on this optimal number of clusters :



7) Hierarchical Clusters

- And this is the distribution of the 6 clusters :



7) Hierarchical Clusters

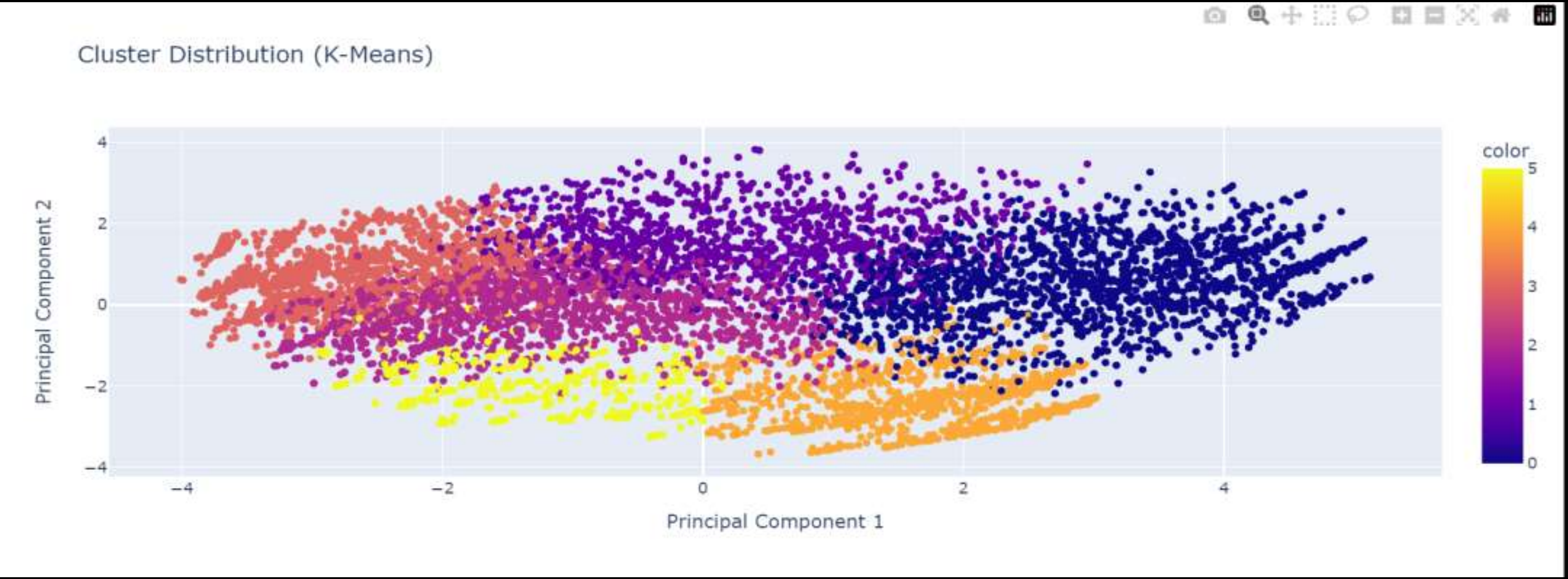
- This is a compression in the score between Kmeans & hierarchical cluster :

```
# Silhouette Score :
from sklearn.metrics import silhouette_score
silhouette_kmeans = silhouette_score(X_Cluster,KMeansModel.labels_)
silhouette_hier = silhouette_score(X_Cluster, AggClusteringModel.fit_predict(X_Cluster))
print(f'Silhouette Score (K-Means): {silhouette_kmeans}')
print(f'Silhouette Score (Hierarchical): {silhouette_hier}')
```

```
Silhouette Score (K-Means): 0.5952399212505846
Silhouette Score (Hierarchical): 0.5800121517407858
```

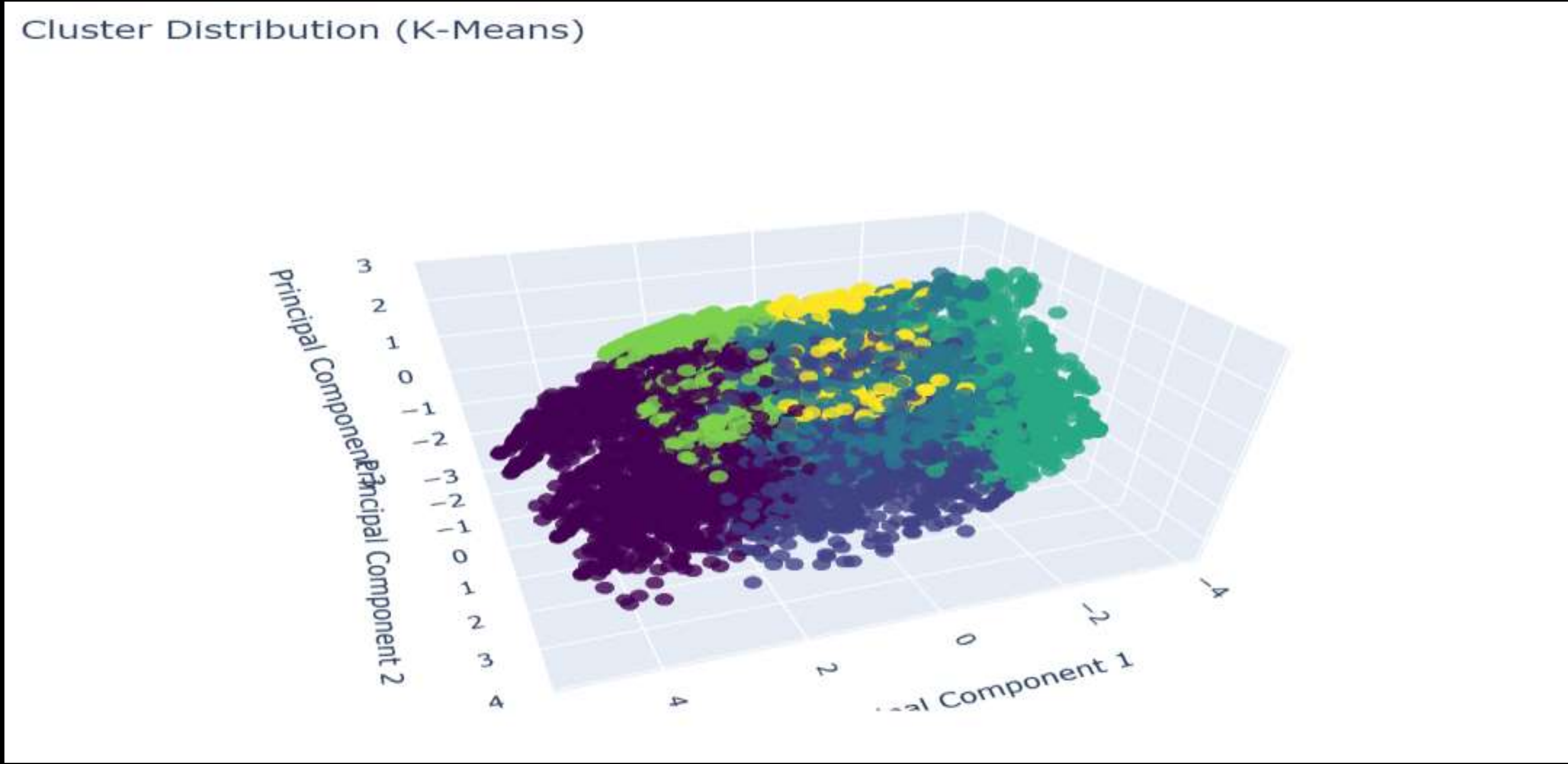
8) After using PCA

- This is the results of k-means after using PCA:



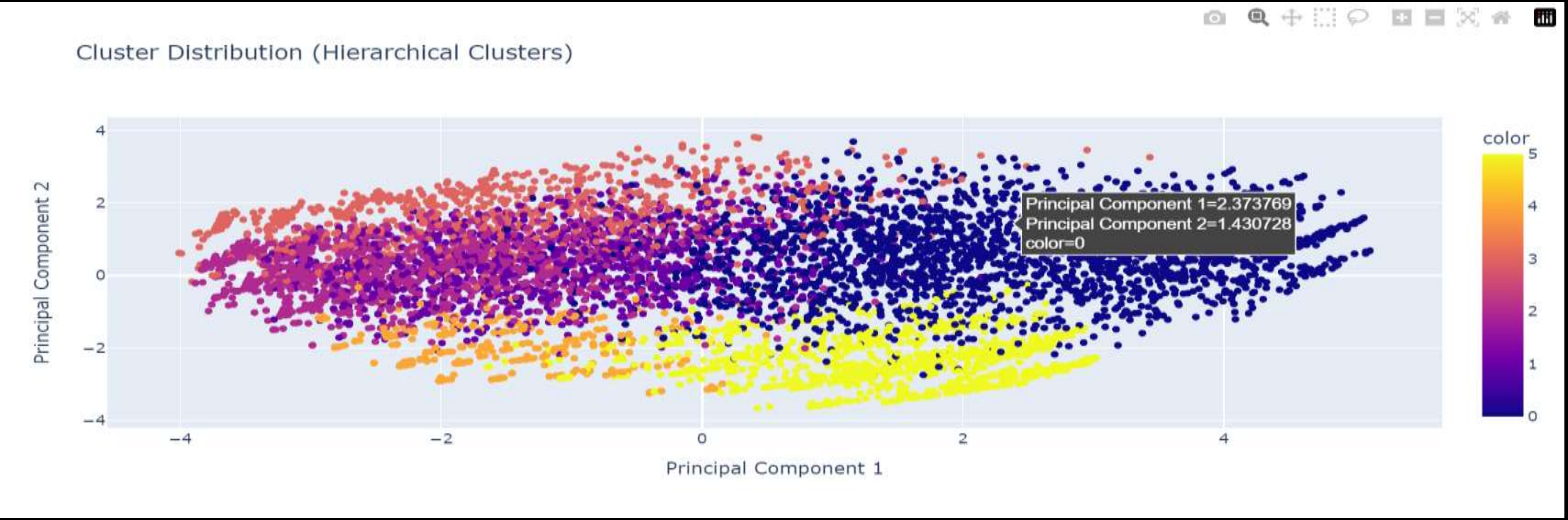
8) After using PCA

- And this in the 3D :



8) After using PCA

- And this is the results of Hierarchical clusters after using PCA :



Ending

- THANK YOU ☺