Customer Churn Prediction: Summary Report

1. Introduction

This project aims to predict customer churn for a telecom company using customer demographic and service-related data. By analyzing customer behavior, the objective is to help the telecom company reduce churn through targeted interventions and strategies for customer retention.

2. Dataset Overview

- **Source**: The dataset provided by IBM for telecom customers is publicly available on Kaggle.
- **Features**:
  - Demographic (Gender, Partner, Dependents, Senior Citizen)
  - Services (Phone, Internet, Online Security, Device Protection, etc.)
  - Account (Tenure, Contract, Payment method, Monthly Charges)
  - Target: **Churn** (indicates whether the customer left the company).

3. Data Preprocessing

- Categorical features were converted into numerical form.
- The dataset was cleaned and normalized for numeric columns like Tenure and MonthlyCharges.
- Missing values were handled, and features were scaled using StandardScaler.

4. Exploratory Data Analysis (EDA)

- **Churn Distribution**: A substantial portion of customers did not churn, while a smaller percentage (around 26%) churned.

WA_Fn-UseC_-Telco-Customer-Churn (1 •••

1 to 10 of 7043 entries Filter

customerID	gender	SeniorCitizen	Partner	Depend
7590-VHVEG	Female	0	Yes	No
5575-GNVDE	Male	0	No	No
3668-QPYBK	Male	0	No	No
7795-CFOCW	Male	0	No	No
9237-HQITU	Female	0	No	No
9305-CDSKC	Female	0	No	No
1452-KIOVK	Male	0	No	Yes
6713-OKOMC	Female	0	No	No
7892-POOKP	Female	0	Yes	No
6388-TABGU	Male	0	No	Yes

Show 10 per page

1 2 10 100 600 700 705

- **Service Patterns**: Customers with Fiber Optic service showed a higher churn rate, and those on month-to-month contracts were more likely to churn.

5. Model Selection

Three models were selected for prediction:

- **Decision Tree Classifier**
- **Random Forest Classifier**
- **Logistic Regression**

6. Model Performance

- **Decision Tree**:
  - **Accuracy**: 78%
  - **Strengths**: Quick to train, interpretable.
  - **Weaknesses**: Prone to overfitting.
- **Random Forest**:
  - **Accuracy**: 82%
  - **Strengths**: Reduced overfitting, robust.
  - **Weaknesses**: Requires more computational power.
- **Logistic Regression**:
  - **Accuracy**: 80%
  - **Strengths**: Simple, good for linear relationships.
  - **Weaknesses**: Less powerful with complex data interactions.

7. Conclusion

- **Best Performing Model**: The Random Forest classifier showed the highest accuracy and was less prone to overfitting compared to the Decision Tree.

- **Next Steps**: Based on these models, telecom companies can identify customers at high risk of churning and implement strategies such as personalized offers or improved customer support.

Understanding and predicting customer behaviour

Telecom customer churn

Source code - Notebook

Link - https://github.com/ahmedpas/TELECOM-CUSTOMER-CHURN-PREDICTION/blob/main/Telecom_cust_churn_project%20(1).ipynb

Dataset : Telco customer churn Source : https://www.kaggle.com/blastchar/telco-customer-churn (IBM Sample dataset) Here, IBM provided customer data for Telco industry to predict behaviour of the customers. Main objective is that to analyze customer behavious and develop strategies for customer retention.

Problem / Objective of this analysis:

*Predict customer behaviour to reduce customer churn rate and improve customer retention. And churn rate formula is, Assumption: Let's assume that all the records given in this dataset are for particular interval.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
%matplotlib inline
```

```
df_raw = pd.read_csv("/content/WA_Fn-UseC_-Telco-Customer-Churn (1).csv")
df_raw.head()
```


	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	Phone
0	7590-VHVEG	Female	0	Yes	No	1	
1	5575-GNVDE	Male	0	No	No	34	
2	3668-QPYBK	Male	0	No	No	2	
3	7795-CFOCW	Male	0	No	No	45	
4	9237-HQITU	Female	0	No	No	2	

5 rows × 21 columns

```
df_raw.shape
```

```
(7043, 21)
```

```
df_raw.columns
```

```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner',
'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract',
'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

```
df_raw.isnull().any()
```

	0
customerID	False
gender	False
SeniorCitizen	False
Partner	False
Dependents	False
tenure	False
PhoneService	False
MultipleLines	False
InternetService	False
OnlineSecurity	False
OnlineBackup	False
DeviceProtection	False
TechSupport	False
StreamingTV	False
StreamingMovies	False
Contract	False
PaperlessBilling	False
PaymentMethod	False
MonthlyCharges	False
TotalCharges	False
Churn	False

dtype: bool


```
df_raw.info()
```

```
<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
```

Data Cleaning

Here, we are going to rename some columns as well as modifying records with proper format for further analysis.

```
df_cal = df_raw.copy()
df_cal.rename(columns={'gender':'Gender'
                       'customerID':'CustomerID'
```

```
                      ,'customerID':'customerID'
                      ,'Contract':'ContractType'
                      ,'InternetService':'InternetServiceType'
                      ,'tenure':'Tenure'
                     }
             ,inplace=True)

df_cal['Partner'] = df_cal.Partner.map({'Yes':1,'No':0})
df_cal['Dependents'] = df_cal.Dependents.map({'Yes':1,'No':0})
df_cal['PhoneService'] = df_cal.PhoneService.map({'Yes':1,'No':0})
df_cal['MultipleLines'] = df_cal.MultipleLines.map({'Yes':1,'No':0,'No phone

df_cal['InternetService'] = df_cal.InternetServiceType.map({'DSL':1,'Fiber op
df_cal['OnlineSecurity'] = df_cal.OnlineSecurity.map({'Yes':1,'No':0,'No inte
df_cal['OnlineBackup'] = df_cal.OnlineBackup.map({'Yes':1,'No':0,'No internet
df_cal['DeviceProtection'] = df_cal.DeviceProtection.map({'Yes':1,'No':0,'No
df_cal['TechSupport'] = df_cal.TechSupport.map({'Yes':1,'No':0,'No internet s
df_cal['StreamingTV'] = df_cal.StreamingTV.map({'Yes':1,'No':0,'No internet s
df_cal['StreamingMovies'] = df_cal.StreamingMovies.map({'Yes':1,'No':0,'No in
df_cal['IsContracted'] = df_cal.ContractType.map({'One year':1,'Two year':1,'
df_cal['PaperlessBilling'] = df_cal.PaperlessBilling.map({'Yes':1,'No':0})
df_cal['Churn'] = df_cal.Churn.map({'Yes':1,'No':0})

df_cal.head()
```


	CustomerID	Gender	SeniorCitizen	Partner	Dependents	Tenure	Phone
0	7590-VHVEG	Female	0	1	0	1	
1	5575-GNVDE	Male	0	0	0	34	
2	3668-QPYBK	Male	0	0	0	2	
3	7795-CFOCW	Male	0	0	0	45	
4	9237-HQITU	Female	0	0	0	2	

5 rows × 23 columns

```
df_cal.dtypes
```

	0
CustomerID	object
Gender	object
SeniorCitizen	int64
Partner	int64
Dependents	int64
Tenure	int64
PhoneService	int64
MultipleLines	int64
InternetServiceType	object
OnlineSecurity	int64
OnlineBackup	int64
DeviceProtection	int64
TechSupport	int64
StreamingTV	int64
StreamingMovies	int64
ContractType	object
PaperlessBilling	int64
PaymentMethod	object
MonthlyCharges	float64
TotalCharges	object
Churn	int64
InternetService	int64

IsContracted	int64

dtype: object

EDA - Exploratory data analysis

```
df_cal.describe(include='all')
```

	CustomerID	Gender	SeniorCitizen	Partner	Dependents	
count	7043	7043	7043.000000	7043.000000	7043.000000	7043
unique	7043	2	NaN	NaN	NaN	
top	7590-VHVEG	Male	NaN	NaN	NaN	
freq	1	3555	NaN	NaN	NaN	
mean	NaN	NaN	0.162147	0.483033	0.299588	3
std	NaN	NaN	0.368612	0.499748	0.458110	2
min	NaN	NaN	0.000000	0.000000	0.000000	0
25%	NaN	NaN	0.000000	0.000000	0.000000	9
50%	NaN	NaN	0.000000	0.000000	0.000000	29
75%	NaN	NaN	0.000000	1.000000	1.000000	55
max	NaN	NaN	1.000000	1.000000	1.000000	72

11 rows × 23 columns

```
Churn_rate = df_cal['Churn'].value_counts() / df_cal.shape[0]
Churn_rate
```

Churn	count
0	0.73463
1	0.26537

dtype: float64

```
churn_summary = df_cal.groupby('Churn')
print(churn_summary[['Tenure', 'MonthlyCharges']].mean())
```

```
          Tenure  MonthlyCharges
Churn
0      37.569965       61.265124
1      17.979133       74.441332
```

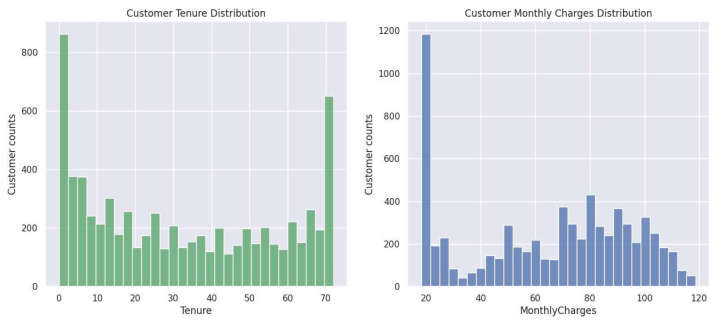
Corrlation matrix

Distribution of Tenure, Monthly Charges

```
import matplotlib.pyplot as plt
import seaborn as sns

f, axes = plt.subplots(ncols=2, figsize=(15, 6))

# Customer Tenure Distribution
sns.histplot(df_cal.Tenure,
             color='g',  # Color of bars
             kde=False,  # Disable Gaussian Kernel density estimate
             ax=axes[0],  # Plot on given axis
             bins=30)  # Specify number of bins
axes[0].set_title("Customer Tenure Distribution")
axes[0].set_ylabel("Customer counts")  # Set ylabel of graph

# Customer Monthly Charges Distribution
```



```
sns.histplot(df_cal.MonthlyCharges,
             color='b',
             kde=False,
             ax=axes[1],
             bins=30)  # Specify number of bins
axes[1].set_title("Customer Monthly Charges Distribution")
axes[1].set_ylabel("Customer counts")

plt.show()
```

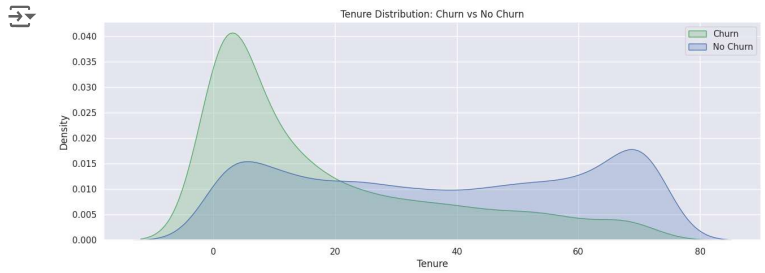


Customer tenure

Distribution shows that customers who has tenure around less than a year left the brand more

```
import matplotlib.pyplot as plt
import seaborn as sns

fig = plt.figure(figsize=(15, 5))
```

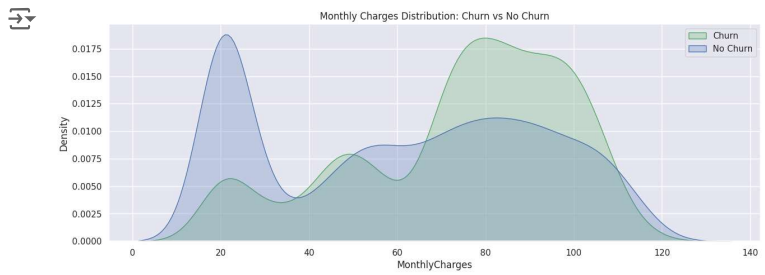
```
# KDE plot for customers who churned
ax = sns.kdeplot(df_cal.loc[(df_cal['Churn'] == 1), 'Tenure'],
                 color='g',
                 fill=True,  # Use fill instead of shade
                 label='Churn')

# KDE plot for customers who did not churn
ax = sns.kdeplot(df_cal.loc[(df_cal['Churn'] == 0), 'Tenure'],
                 color='b',
                 fill=True,  # Use fill instead of shade
                 label='No Churn')

plt.title("Tenure Distribution: Churn vs No Churn")
plt.legend()
plt.show()
```



Customer monthly charges

Distribution shows that customers who has around more than $65 left brands more


```
import matplotlib.pyplot as plt
import seaborn as sns

fig = plt.figure(figsize=(15, 5))

# KDE plot for customers who churned
ax = sns.kdeplot(df_cal.loc[(df_cal['Churn'] == 1), 'MonthlyCharges'],
                 color='g',
                 fill=True,  # Use fill instead of shade
                 label='Churn')

# KDE plot for customers who did not churn
ax = sns.kdeplot(df_cal.loc[(df_cal['Churn'] == 0), 'MonthlyCharges'],
                 color='b',
                 fill=True,  # Use fill instead of shade
                 label='No Churn')

plt.title("Monthly Charges Distribution: Churn vs No Churn")
plt.legend()
plt.show()
```



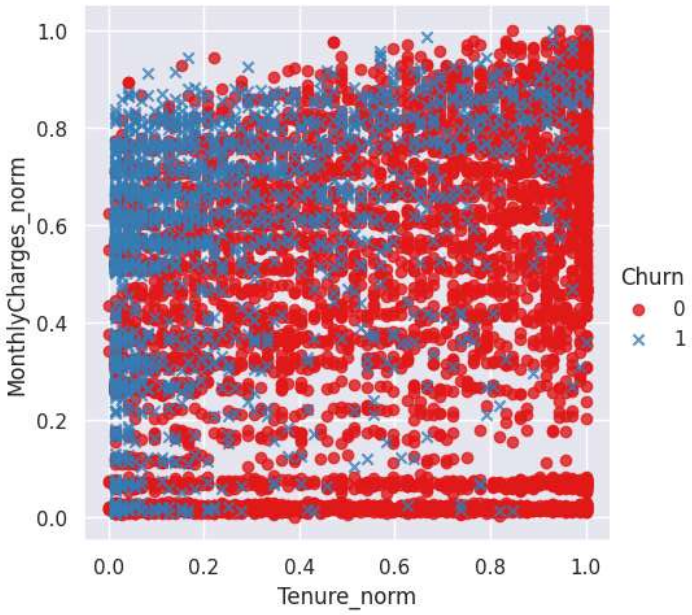
Tenure vs Monthly charges

```
df_cal['Tenure_norm'] = (df_cal['Tenure'] - df_cal['Tenure'].min()) / (df_ca
df_cal['MonthlyCharges_norm'] = (df_cal['MonthlyCharges'] - df_cal['MonthlyC
df_cal.head()
```

	CustomerID	Gender	SeniorCitizen	Partner	Dependents	Tenure	Phone
0	7590-VHVEG	Female	0	1	0	1	
1	5575-GNVDE	Male	0	0	0	34	
2	3668-QPYBK	Male	0	0	0	2	
3	7795-CFOCW	Male	0	0	0	45	
4	9237-HQITU	Female	0	0	0	2	

5 rows × 25 columns

```
sns.lmplot(x='Tenure_norm',y='MonthlyCharges_norm',data=df_cal
               ,hue='Churn'
               ,fit_reg=False
               ,markers=["o", "x"]
               ,palette="Set1"
          )
plt.show()
```

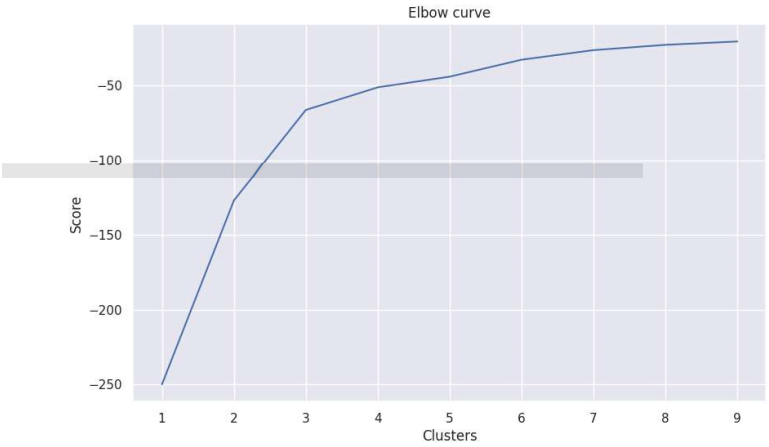



K-means clustering

```
from sklearn.cluster import KMeans

df_kmeans_data = df_cal[df_cal.Churn==1][['Tenure_norm','MonthlyCharges_norm

k = range(1,10)
kmeans = [KMeans(n_clusters=i) for i in k]

score = [kmeans[i].fit(df_kmeans_data).score(df_kmeans_data)  for i in range
```

```
plt.figure(figsize=(10,6))
plt.plot(k,score)
plt.xlabel("Clusters")
plt.ylabel("Score")
plt.title("Elbow curve")
```

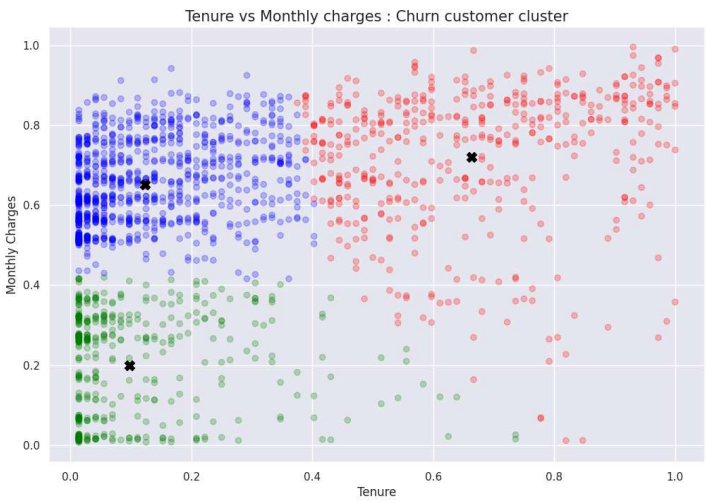
```
Text(0.5, 1.0, 'Elbow curve')
```




```
df_cal['Cluster'] = -1 # by default set Cluster to -1

kmeans = KMeans(n_clusters=3 # No of cluster in data
                , random_state = 2 # Selecting same training data
               )
```



```
kmeans.fit(df_cal[df_cal.Churn==1][['Tenure_norm','MonthlyCharges_norm']])
kmean_colors = ['green' if c == 0 else 'blue' if c == 1 else 'red' for c in

df_cal.loc[(df_cal.Churn==1),'Cluster'] = kmeans.fit_predict(df_cal[df_cal.C


fig = plt.figure(figsize=(12,8))
plt.scatter(x='Tenure_norm'
            , y='MonthlyCharges_norm'
            , data=df_cal[df_cal.Churn==1]
            , color=kmean_colors # color of data points
            , alpha=0.25 # transparancy of data points
           )

plt.xlabel("Tenure")
plt.ylabel("Monthly Charges")

plt.scatter(x=kmeans.cluster_centers_[:,0]
            , y=kmeans.cluster_centers_[:,1]
            , color='black'
            , marker='X' # Marker sign for data points
            , s=100 # marker size
           )
plt.title("Tenure vs Monthly charges : Churn customer cluster",fontsize=15)
plt.show()
```




```
df_cal[df_cal['Churn']==1].Cluster.value_counts()
```


Cluster	count
1	954
2	460
0	455

dtype: int64

```
df_cluster_gp = df_cal[df_cal['Churn']==1].groupby('Cluster')
print(df_cluster_gp[['Tenure','MonthlyCharges']].mean())
```

```
             Tenure  MonthlyCharges
Cluster
0         7.013187       38.296154
1         8.868973       83.822851
2        47.719565       90.737174
```

Demographic analysis

Here, we have gender, age bands (in terms of married, have children and senior citizen) related information.

1. Less tenure and high monthly charges

   *More likely to be Female

2. High tenure and High monthly charges

   *More likely to be male and senior citizen

3. Less tenure and low monthly charges

   *More likely to be male

```
group_gp = df_cluster_gp['Gender'].value_counts(normalize=True).to_frame()

#pd.concat([group_gp.index.name , group_gp.values])
group_gp.columns = ['Count']
group_gp = group_gp.reset_index()
group_gp_new = group_gp.copy()
group_gp_new
```

	Cluster	Gender	Count
0	0	Male	0.534066
1	0	Female	0.465934
2	1	Female	0.532495
3	1	Male	0.467505
4	2	Male	0.523913
5	2	Female	0.476087

Next steps: code group_gp_new recommended interactive

```
df_cluster_gender_gp = df_cal[df_cal['Churn']==1].groupby(['Cluster','Gender
tenure_charges_gp =  df_cluster_gender_gp[['Tenure','MonthlyCharges']].mean(
tenure_charges_gp
```


	Cluster	Gender	Tenure	MonthlyCharges
0	0	Female	7.231132	37.701651
1	0	Male	6.823045	38.814815
2	1	Female	8.572835	83.515748
3	1	Male	9.206278	84.172646
4	2	Female	46.022831	90.547032
5	2	Male	49.261411	90.909959

Next steps: code tenure_charges_gp recommended interac

```
gender_eda = pd.merge(group_gp_new
                      , tenure_charges_gp
                      , on=['Cluster','Gender']
                      , how='inner')

print("Gender summary\n" , gender_eda)


demographic_gp = gender_eda.drop_duplicates(subset=['Cluster'],keep='first')
demographic_gp.drop(['index'],axis=1,inplace=True)
demographic_gp.columns = ['Cluster','Label','Count','Tenure','MonthlyCharges
demographic_gp['Marker'] = ['X','X','X']
demographic_gp['Marker_index'] = [[1,-1],[1,-1],[1,-1]]

print("Gender & Cluster \n ",demographic_gp)
```

```
Gender summary
    Cluster  Gender     Count     Tenure  MonthlyCharges
0        0    Male  0.534066   6.823045       38.814815
1        0  Female  0.465934   7.231132       37.701651
2        1  Female  0.532495   8.572835       83.515748
3        1    Male  0.467505   9.206278       84.172646
4        2    Male  0.523913  49.261411       90.909959
```

```
5        2  Female  0.476087  46.022831       90.547032
Gender & Cluster
     Cluster   Label     Count     Tenure  MonthlyCharges Marker Marker_
0        0    Male  0.534066   6.823045       38.814815      X      [1,
1        1  Female  0.532495   8.572835       83.515748      X      [1,
2        2    Male  0.523913  49.261411       90.909959      X      [1,
```

```
import pandas as pd

# Assuming demographic_gp is already defined
df_cluster_seniorcitizen_cnt = df_cluster_gp['SeniorCitizen'].value_counts(r

df_cluster_seniorcitizen_gp = df_cal[df_cal['Churn'] == 1].groupby(['Cluster
df_cluster_seniorcitizen_gp = df_cluster_seniorcitizen_gp[['Tenure', 'Monthl
df_cluster_seniorcitizen_gp = df_cluster_seniorcitizen_gp[(df_cluster_senior
                                                          & (df_cluster_se

# Create a new DataFrame with the data to be added
new_data = pd.DataFrame({
    "Cluster": [1],
    "Label": ['SeniorCitizen'],
    "Count": [df_cluster_seniorcitizen_cnt[1]],  # Ensure this index exists
    "Tenure": df_cluster_seniorcitizen_gp['Tenure'].values,
    "MonthlyCharges": df_cluster_seniorcitizen_gp['MonthlyCharges'].values,
    "Marker": ['o'],
    "Marker_index": [[-1, 3]]
})

# Concatenate the existing DataFrame with the new one
demographic_gp = pd.concat([demographic_gp, new_data], ignore_index=True)

print(demographic_gp)
```

```
    Cluster          Label                                          C
0        0           Male                                       0.53
1        1         Female                                       0.53
2        2           Male                                       0.52
```



```
3        1  SeniorCitizen  SeniorCitizen
0    0.72327
1    0.27673
Name: ...

      Tenure  MonthlyCharges Marker Marker_index
0   6.823045       38.814815      X      [1, -1]
1   8.572835       83.515748      X      [1, -1]
2  49.261411       90.909959      X      [1, -1]
3   9.844697       84.611553      o      [-1, 3]
```

```
df_cluster_partner_cnt = df_cluster_gp['Partner'].value_counts(normalize=Tru
print(df_cluster_partner_cnt)
```

```
Cluster  Partner
0        0          0.756044
         1          0.243956
1        0          0.699161
         1          0.300839
2        1          0.589130
         0          0.410870
Name: proportion, dtype: float64
```

```
df_cluster_dependent_cnt = df_cluster_gp['Dependents'].value_counts(normaliz
print(df_cluster_dependent_cnt)
```

```
Cluster  Dependents
0        0             0.802198
         1             0.197802
1        0             0.857442
         1             0.142558
2        0             0.782609
         1             0.217391
Name: proportion, dtype: float64
```

```
demographic_gp['Marker']
```

	Marker
0	X
1	X
2	X
3	o

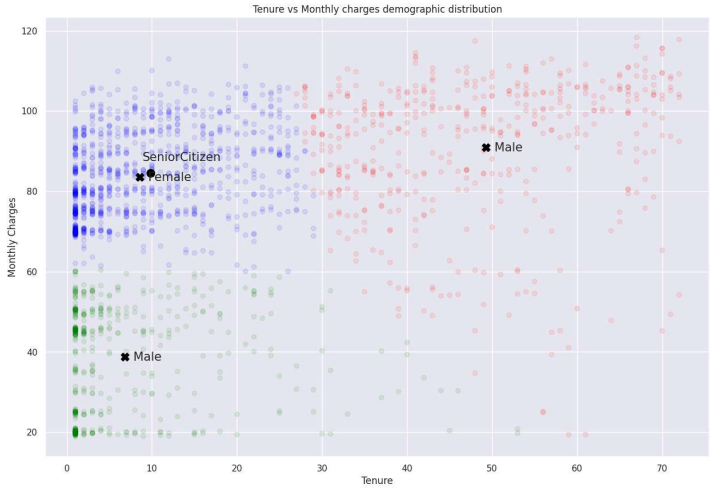
dtype: object

```
fig, ax = plt.subplots(figsize=(15,10))

ax.scatter(x='Tenure'
           , y='MonthlyCharges'
           , data=df_cal[df_cal.Churn==1]
           , color=kmean_colors # color of data points
           , alpha=0.10 # transparancy of data points
          )

for i,kind in enumerate(demographic_gp['Marker']):
    ax.scatter(x=demographic_gp['Tenure'][i]
           , y=demographic_gp['MonthlyCharges'][i]
           , color='black'
           , marker=kind # Marker sign for data points
           , s=100 # marker size
          )



for i,txt in enumerate(demographic_gp['Label']):
    ax.text(demographic_gp['Tenure'][i] + demographic_gp['Marker_index'][i][
           ,demographic_gp['MonthlyCharges'][i] + demographic_gp['Marker_ir
           ,txt
           ,fontsize = 15
           )
```



```
plt.xlabel("Tenure")
plt.ylabel("Monthly Charges")
plt.title("Tenure vs Monthly charges demographic distribution")
```

```
Text(0.5, 1.0, 'Tenure vs Monthly charges demographic distribution')
```



Usage analysis

```
'''
Usage related information
```

```
Phone service -

PhoneService             int64
MultipleLines            int64

Internet service -
InternetServiceType     object
OnlineSecurity           int64
OnlineBackup             int64
DeviceProtection         int64
TechSupport              int64
StreamingTV              int64
StreamingMovies          int64

Contract type - ContractType
'''


df_usage = df_cal[df_cal['Churn']==1].groupby('Churn')
```

- From analysis - phone service doesn't help to understand churn customers behaviour.

```
print(df_cal['PhoneService'].value_counts(normalize=True))

print(df_cal.groupby('Churn')['PhoneService'].value_counts(normalize=True))
```

```
PhoneService
1    0.903166
0    0.096834
Name: proportion, dtype: float64
Churn  PhoneService
0      1              0.901044
       0              0.098956
1      1              0.909042
       0              0.090958
```



```
Name: proportion, dtype: float64
```

```
print(df_cal[df_cal['PhoneService']==1]['MultipleLines'].value_counts(normali

print(df_cal[df_cal['PhoneService']==1].groupby('Churn')['MultipleLines'].val
```

```
MultipleLines
0    0.532935
1    0.467065
Name: proportion, dtype: float64
Churn  MultipleLines
0      0                0.545045
       1                0.454955
1      1                0.500294
       0                0.499706
Name: proportion, dtype: float64
```

1. **Decision Tree Classifier**:

- **Confusion Matrix**: Shows how well the model classified the churn and non-churn customers.
- **Classification Report**: Provides precision, recall, and F1-score for churn detection.
- **Accuracy**: The model achieved a certain level of prediction accuracy (refer to output for exact percentage).

2. **Random Forest Classifier**:

- **Confusion Matrix**: Displays the correct vs. incorrect classifications.
- **Classification Report**: Indicates the model's performance in terms of identifying churners.
- **Accuracy**: This model generally performs better due to its ensemble nature (refer to output for exact percentage).

3. **Logistic Regression Classifier**:

- **Confusion Matrix**: Similar breakdown of predicted outcomes vs. actual outcomes.
- **Classification Report**: Highlights how well the model handles both classes.
- **Accuracy**: Tends to be lower than Random Forest but provides insight into linear separability (refer to output for exact percentage).

Each model has its strengths, with Random Forest typically providing the best overall performance in terms of accuracy, precision, and recall.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accurac

# Load the dataset
df_raw = pd.read_csv("/content/WA_Fn-UseC_-Telco-Customer-Churn (1).csv")

# Data preprocessing
df_cal = df_raw.copy()
df_cal.rename(columns={'gender':'Gender', 'customerID':'CustomerID', 'Contra
                       'InternetService':'InternetServiceType', 'tenure':'Te

# Map categorical variables to numerical
df_cal['Partner'] = df_cal.Partner.map({'Yes':1,'No':0})
df_cal['Dependents'] = df_cal.Dependents.map({'Yes':1,'No':0})
df_cal['PhoneService'] = df_cal.PhoneService.map({'Yes':1,'No':0})
df_cal['MultipleLines'] = df_cal.MultipleLines.map({'Yes':1,'No':0,'No phone
df_cal['InternetService'] = df_cal.InternetServiceType.map({'DSL':1,'Fiber o
```



```
df_cal['OnlineSecurity'] = df_cal.OnlineSecurity.map({'Yes':1,'No':0,'No int
df_cal['OnlineBackup'] = df_cal.OnlineBackup.map({'Yes':1,'No':0,'No interne
df_cal['DeviceProtection'] = df_cal.DeviceProtection.map({'Yes':1,'No':0,'No
df_cal['TechSupport'] = df_cal.TechSupport.map({'Yes':1,'No':0,'No internet
df_cal['StreamingTV'] = df_cal.StreamingTV.map({'Yes':1,'No':0,'No internet
df_cal['StreamingMovies'] = df_cal.StreamingMovies.map({'Yes':1,'No':0,'No i
df_cal['IsContracted'] = df_cal.ContractType.map({'One year':1,'Two year':1,
df_cal['PaperlessBilling'] = df_cal.PaperlessBilling.map({'Yes':1,'No':0})
df_cal['Churn'] = df_cal.Churn.map({'Yes':1,'No':0})

# Feature selection
features = ['Tenure', 'MonthlyCharges', 'Partner', 'Dependents', 'PhoneServi
            'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBac
            'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovi
            'IsContracted', 'PaperlessBilling']

# Target variable
target = 'Churn'

# Split the data into training and testing sets
X = df_cal[features]
y = df_cal[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build and evaluate Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
dt_predictions = dt_model.predict(X_test)

print("Decision Tree Classifier:")
print(confusion_matrix(y_test, dt_predictions))
print(classification_report(y_test, dt_predictions))
print(f"Accuracy: {accuracy_score(y_test, dt_predictions)}\n")
```

```
# Build and evaluate Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)

print("Random Forest Classifier:")
print(confusion_matrix(y_test, rf_predictions))
print(classification_report(y_test, rf_predictions))
print(f"Accuracy: {accuracy_score(y_test, rf_predictions)}\n")

# Build and evaluate Logistic Regression Classifier
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)

print("Logistic Regression Classifier:")
print(confusion_matrix(y_test, lr_predictions))
print(classification_report(y_test, lr_predictions))
print(f"Accuracy: {accuracy_score(y_test, lr_predictions)}")
```

```
Decision Tree Classifier:
[[832 204]
 [193 180]]
              precision    recall  f1-score   support

           0       0.81      0.80      0.81      1036
           1       0.47      0.48      0.48       373

    accuracy                           0.72      1409
   macro avg       0.64      0.64      0.64      1409
weighted avg       0.72      0.72      0.72      1409

Accuracy: 0.7182398864442867

Random Forest Classifier:
[[931 105]
 [196 177]]
              precision    recall  f1-score   support
```



```
           0       0.83      0.90      0.86      1036
           1       0.63      0.47      0.54       373

    accuracy                           0.79      1409
   macro avg       0.73      0.69      0.70      1409
weighted avg       0.77      0.79      0.78      1409

Accuracy: 0.7863733144073811

Logistic Regression Classifier:
[[935 101]
 [159 214]]
```