Tittle : Telecustomer churn prediction

Description : telecustomer churn prediction weather a customer is about to churn.

problem statement : Developing the model to predict which customers are likely to churn.

desired outcome : identifying the customers who are leaving from our subcribstion and the predactive steps can be taken by retain them.

Importing neccessary libraries

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the dataset

In [4]:
```
df = pd.read_csv ("Telco-Customer-Churn_Missing.csv")
```

In [5]:
```
df
```

Out[5]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLin
0	7590-VHVEG	Female	0	Yes	No	NaN	No	No pho servi
1	5575-GNVDE	Male	0	No	No	34.0	Yes	I
2	3668-QPYBK	Male	0	No	No	2.0	Yes	I
3	7795-CFOCW	Male	0	No	No	45.0	No	No pho servi
4	9237-HQITU	Female	0	No	No	2.0	Yes	I
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24.0	Yes	Y
7039	2234-XADUH	Female	0	Yes	Yes	72.0	Yes	Y
7040	4801-JZAZL	Female	0	Yes	Yes	11.0	No	No pho servi
7041	8361-LTMKD	Male	1	Yes	No	4.0	Yes	Y
7042	3186-AJIEK	Male	0	No	No	66.0	Yes	I

7043 rows × 21 columns

Understanding the data structure

In [6]:
```
df.head(5)
```

Out[6]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	7590-VHVEG	Female	0	Yes	No	NaN	No	No phone service
1	5575-GNVDE	Male	0	No	No	34.0	Yes	No
2	3668-QPYBK	Male	0	No	No	2.0	Yes	No
3	7795-CFOCW	Male	0	No	No	45.0	No	No phone service
4	9237-HQITU	Female	0	No	No	2.0	Yes	No

5 rows × 21 columns

In [7]:
```
df.tail(5)
```

Out[7]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLin
7038	6840-RESVB	Male	0	Yes	Yes	24.0	Yes	Y
7039	2234-XADUH	Female	0	Yes	Yes	72.0	Yes	Y
7040	4801-JZAZL	Female	0	Yes	Yes	11.0	No	No pho servi
7041	8361-LTMKD	Male	1	Yes	No	4.0	Yes	Y
7042	3186-AJIEK	Male	0	No	No	66.0	Yes	I

5 rows × 21 columns

In [8]:
```
df.shape
```

Out[8]: (7043, 21)

In [9]:
```
df.columns
```

Out[9]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

In [10]:
```
df.dtypes
```

Out[10]:
```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure              float64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

In [11]:
```
df.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            7041 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7042 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7042 non-null   float64
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7039 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     7040 non-null   object 
 18  MonthlyCharges    7042 non-null   float64
 19  TotalCharges      7042 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(2), int64(1), object(18)
memory usage: 1.1+ MB
```

Handling Missing Values

In [12]:
```
df.isna().sum()
```

Out[12]:
```
customerID          0
gender              2
SeniorCitizen       0
Partner             1
Dependents          0
tenure              1
PhoneService        0
MultipleLines       0
InternetService     4
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       3
MonthlyCharges      1
TotalCharges        1
Churn               0
dtype: int64
```

In [13]:
```
df.dropna(inplace=True)
```

In [14]:
```
df.isna().sum()
```

Out[14]:
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

In [15]:
```
df.drop(columns='customerID',inplace=True)
```

In [16]:
```
df.shape
```

Out[16]:
```
(7030, 20)
```

In [17]:
```
#df.fillna('missing',inplace=True)
```

In [18]:
```
### Handling the missing values
### Numerical data we use median, and categorical data we use mode
```

In [19]:
```
#df['gender'].fillna(df['gender'].mode()[0],inplace=True)
```

In [20]:
```
#df['Partner'].fillna(df['Partner'].mode()[0],inplace=True)
```

In [21]:
```
#df['tenure'].fillna(df['tenure'].median(),inplace=True)
```

In [22]:
```
#df['InternetService'].fillna(df['InternetService'].mode()[0],inplace=True)
```

In [23]:
```
#df['PaymentMethod'].fillna(df['PaymentMethod'].mode()[0],inplace=True)
```

In [24]:
```
#df['MonthlyCharges'].fillna(df['MonthlyCharges'].median(),inplace=True)
```

In [25]:
```
#df['TotalCharges'].fillna(df['TotalCharges'].mode()[0],inplace=True)
```

converting string to float

In [26]:
```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'],errors='coerce')
```

In [27]:
```
df.isna().sum()
```

Out[27]:
```
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

In [28]:
```
df['TotalCharges'].fillna(df['TotalCharges'].median(),inplace=True)
```

In [29]:
```
df.isna().sum()
```

Out[29]:
```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

Categorical encoding

In [30]:
```
from sklearn.preprocessing import LabelEncoder
```

In [31]:
```
le = LabelEncoder()
```

In [32]:
```
df['gender'] = le.fit_transform(df['gender'])
```

In [33]:
```
df['Partner'] = le.fit_transform(df['Partner'])
```

In [34]:
```
df['Dependents'] = le.fit_transform(df['Dependents'])
```

In [35]:
```
df['PhoneService'] = le.fit_transform(df['PhoneService'])
```

In [36]:
```
df['MultipleLines'] = le.fit_transform(df['MultipleLines'])
```

In [37]:
```
df['InternetService'] = le.fit_transform(df['InternetService'])
```

In [38]:
```
df['OnlineSecurity'] = le.fit_transform(df['OnlineSecurity'])
```

In [39]:
```
df['DeviceProtection'] = le.fit_transform(df['DeviceProtection'])
```

In [40]:
```
df['TechSupport'] = le.fit_transform(df['TechSupport'])
```

In [41]:
```
df['StreamingTV'] = le.fit_transform(df['StreamingTV'])
```

In [42]:
```
df['Contract'] = le.fit_transform(df['Contract'])
```

In [43]:
```
df['StreamingMovies'] = le.fit_transform(df['StreamingMovies'])
```

In [44]:
```
df['PaperlessBilling'] = le.fit_transform(df['PaperlessBilling'])
```

In [45]:
```
df['PaymentMethod'] = le.fit_transform(df['PaymentMethod'])
```

In [46]:
```
df['Churn'] = le.fit_transform(df['Churn'])
```

In [47]:
```
df['OnlineBackup'] = le.fit_transform(df['OnlineBackup'])
```

In [48]:
```
df
```

Out[48]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetS
1	1	0	0	0	34.0	1	0	
2	1	0	0	0	2.0	1	0	
8	0	0	1	0	28.0	1	2	
9	1	0	0	1	62.0	1	0	
10	1	0	1	1	13.0	1	0	
...	...	...	...	...	...	...	...	
7038	1	0	1	1	24.0	1	2	
7039	0	0	1	1	72.0	1	2	
7040	0	0	1	1	11.0	0	1	
7041	1	1	1	0	4.0	1	2	
7042	1	0	0	0	66.0	1	0	

7030 rows × 20 columns

Feature scaling

In [49]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [50]:
```
df_scale = MinMaxScaler()
```

In [51]:
```
df_scale = df_scale.fit_transform(df)
```

In [52]:
```
df_scale
```

Out[52]:
```
array([[1.        , 0.        , 0.        , ..., 0.38507463, 0.21586661,
        0.        ],
       [1.        , 0.        , 0.        , ..., 0.35422886, 0.01031041,
        1.        ],
       [0.        , 0.        , 1.        , ..., 0.86119403, 0.34932495,
        1.        ],
       ...,
       [0.        , 0.        , 1.        , ..., 0.11293532, 0.03780868,
        0.        ],
       [1.        , 1.        , 1.        , ..., 0.55870647, 0.03321025,
        1.        ],
       [1.        , 0.        , 0.        , ..., 0.86965174, 0.78764136,
        0.        ]])
```

Splitting data for model training

In [53]:
```
x = df.iloc [:,:-1]
y = df.iloc [:,-1]
```

In [54]:
```
print (x.shape)
print (y.shape)
```

```
(7030, 19)
(7030,)
```

In [55]:
```
df['Churn'].value_counts()
```

Out[55]:
```
0    5166
1    1864
Name: Churn, dtype: int64
```

In [56]:
```
#df.isna().sum()
#df = pd.get_dummies (df,dtype=int)
```

In [57]:
```
#df
```

Oversampling

In [58]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [59]:
```
ros = RandomOverSampler (random_state=0)
```

In [60]:
```
x_res,y_res = ros.fit_resample(x,y)
```

In [61]:
```
x.shape
```

Out[61]:
```
(7030, 19)
```

In [62]:
```
x_res.shape
```

Out[62]:
```
(10332, 19)
```

In [63]:
```
y_res.shape
```

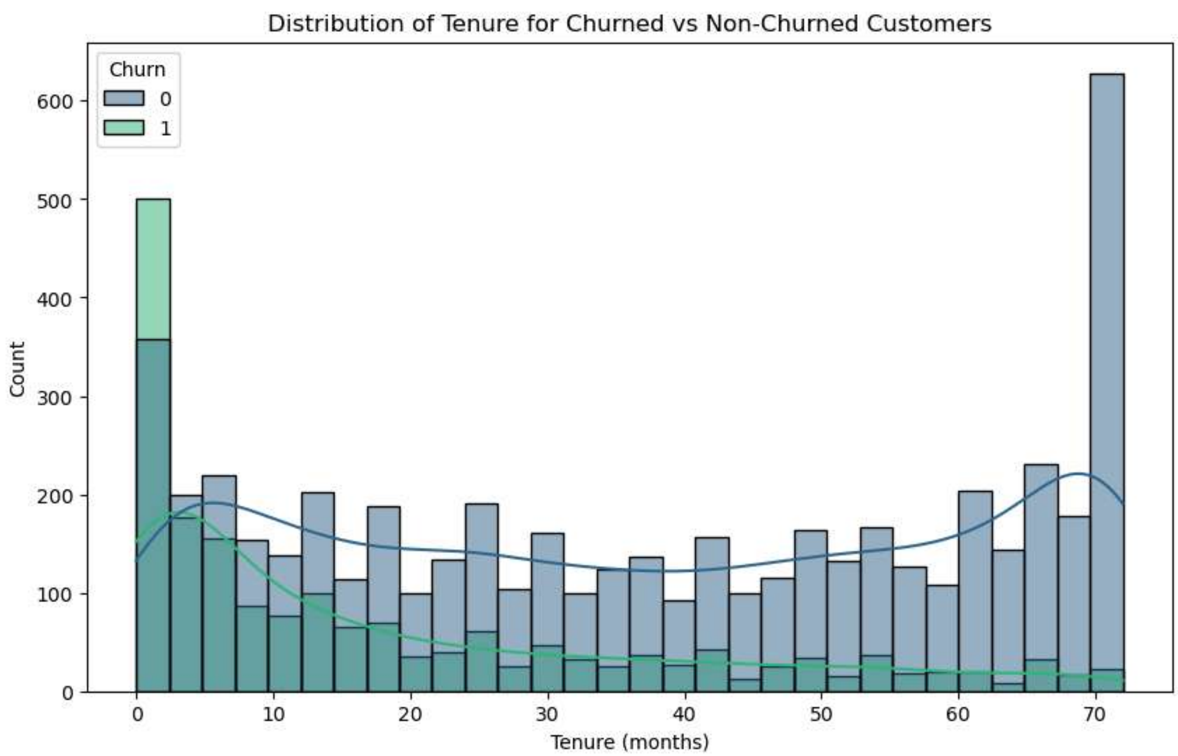
Out[63]:
```
(10332,)
```

In [64]:
```
from collections import  Counter
```

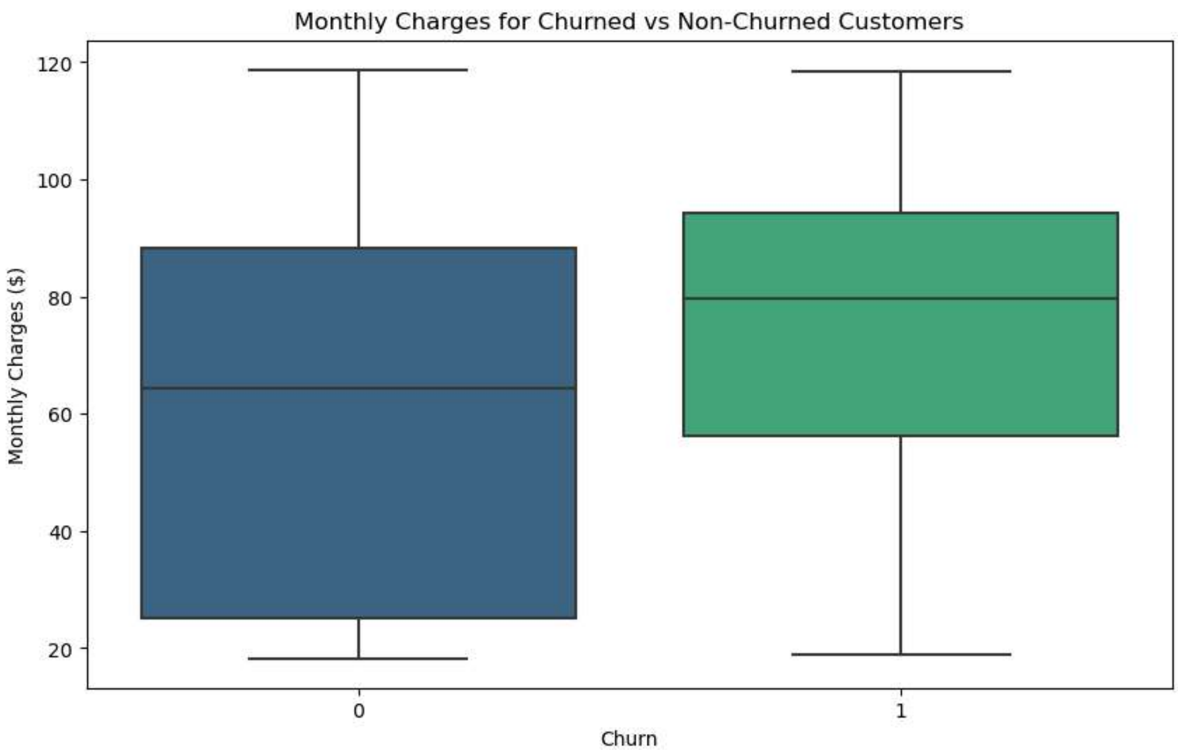
In [65]:
```
print (Counter(y))
print (Counter (y_res))
```

```
Counter({0: 5166, 1: 1864})
Counter({0: 5166, 1: 5166})
```

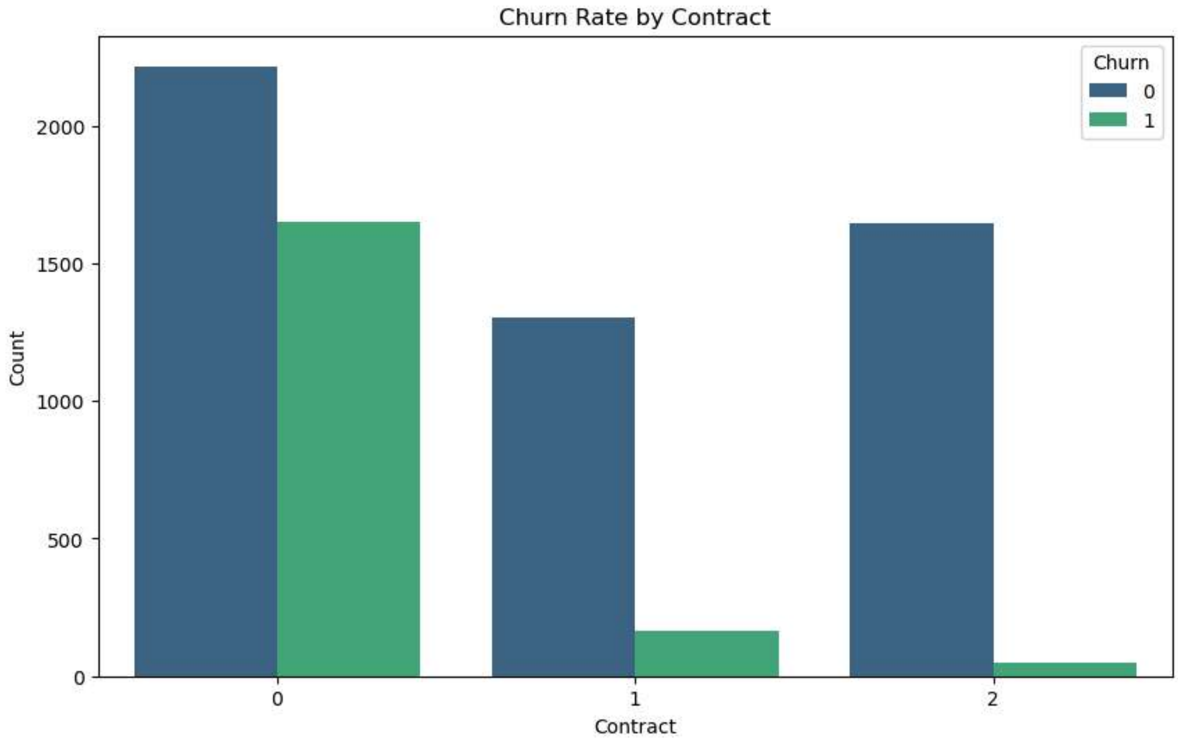
In [66]:
```
# Distribution of tenure for churned vs non-churned customers
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='tenure', hue='Churn', kde=True, bins=30, palette='vir
plt.title('Distribution of Tenure for Churned vs Non-Churned Customers')
plt.xlabel('Tenure (months)')
plt.ylabel('Count')
plt.show()
```



In [67]:
```
# Monthly Charges for churned vs non-churned customers
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Churn', y='MonthlyCharges', palette='viridis')
plt.title('Monthly Charges for Churned vs Non-Churned Customers')
plt.xlabel('Churn')
plt.ylabel('Monthly Charges ($)')
plt.show()
```



In [68]:
```
# Plotting churn rate by contract type
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='Contract', hue='Churn', palette='viridis')
plt.title('Churn Rate by Contract')
plt.xlabel('Contract')
plt.ylabel('Count')
plt.show()
```



Train_test_split

In [69]:
```
from sklearn.model_selection import train_test_split
```

In [70]:
```
x_train,x_test,y_train,y_test = train_test_split (x_res,y_res,test_size = 0.30
```

In [71]:
```
print (x_train.shape)
print (x_test.shape)
print (y_train.shape)
print (y_test.shape)
```

```
(7232, 19)
(3100, 19)
(7232,)
(3100,)
```

Data modeling

In [72]:
```
# y variable Churn is categorical data so we use logistic regression
```

In [73]:
```
from sklearn.linear_model import LogisticRegression
```

In [74]:
```
model = LogisticRegression()
```

In [75]:
```
model
```

Out[75]:

```
▾ LogisticRegression ⓘ ?
(https://scikit-
learn.org/1.4/modules/generated/sklearn.linear_model.LogisticRe
LogisticRegression()
```

In [76]:
```
model = model.fit (x_train,y_train)
```

```
C:\Users\sande\AppData\Roaming\Python\Python311\site-packages\sklearn\linear_
model\_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

In [77]:
```
y_true = y_test
```

In [78]:
```
y_pred1 = model.predict (x_test)
```

In [79]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [80]:
```
print (classification_report(y_true,y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.77      0.73      0.75      1526
           1       0.75      0.79      0.77      1574

    accuracy                           0.76      3100
   macro avg       0.76      0.76      0.76      3100
weighted avg       0.76      0.76      0.76      3100
```

In [81]:
```
print (confusion_matrix(y_true,y_pred1))
```

```
[[1115  411]
 [ 331 1243]]
```

In [82]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [83]:
```
model = DecisionTreeClassifier()
```

In [84]:
```
model = model.fit(x_train,y_train)
```

In [85]:
```
y_pred2 = model.predict(x_test)
```

In [86]:
```
print (classification_report(y_true,y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.90      0.78      0.84      1526
           1       0.81      0.92      0.86      1574

    accuracy                           0.85      3100
   macro avg       0.86      0.85      0.85      3100
weighted avg       0.86      0.85      0.85      3100
```

In [87]:
```
print (confusion_matrix(y_true,y_pred2))
```

```
[[1197  329]
 [ 133 1441]]
```

In [88]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [89]:
```
model = RandomForestClassifier()
```

In [90]:
```python
model
```

Out[90]:

▾ RandomForestClassifier ⓘ ? (https://scikit-learn.org/1.4/modules/generated/sklearn.ensemble.RandomFor

```
RandomForestClassifier()
```

In [91]:
```python
model = model.fit(x_train,y_train)
```

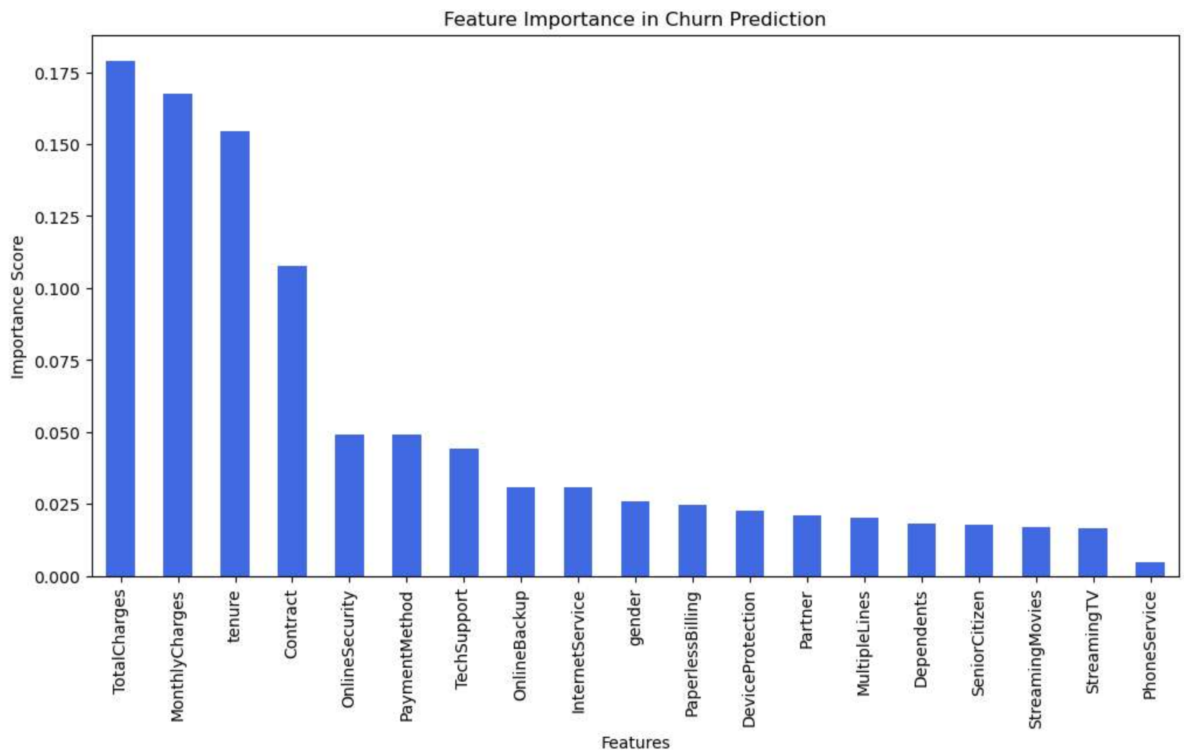
In [92]:
```python
y_predict = model.predict(x_test)
```

In [93]:
```python
y_true = y_test
```

In [94]:
```python
from sklearn.metrics import classification_report
print (classification_report(y_true,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.93      0.83      0.88      1526
           1       0.85      0.94      0.89      1574

    accuracy                           0.89      3100
   macro avg       0.89      0.88      0.88      3100
weighted avg       0.89      0.89      0.89      3100
```

In [97]:
```
#  Feature Importance Analysis

# Get feature importances
feature_importances = pd.Series(model.feature_importances_, index=df.drop(colu

# Sort and plot feature importance
plt.figure(figsize=(12, 6))
feature_importances.sort_values(ascending=False).plot(kind='bar', color='royal
plt.title("Feature Importance in Churn Prediction")
plt.xlabel("Features")
plt.ylabel("Importance Score")
plt.show()
```



Final summary of findings:

Main factors affecting churn are contract type, tenure, and monthly charges. Customers on month-to-month contracts are more likely to churn. Recommendation : Promote long-term contracts and discounts on monthly charges to reduce churn.

In [ ]: