MACHINE LEARNING FINAL PROJECT PART-B

NAME:Gazi Aman Khan BATCH: 15th MAY(Data Science)

Problem Statement The objective of this analysis is to predict customer churn — whether a customer is likely to discontinue the company's services — based on demographic information, account details, and service usage patterns. By building and comparing classification models, the goal is to identify key factors driving churn and develop a predictive model that helps the business take proactive retention actions to reduce customer loss and improve satisfaction.

LIBRARIES USED: pandas, matplotlib, seaborn, scikit-learn

PROJECT EXPLANATION VIDEO LINK:https://www.loom.com/share/637afaa5d04f4bd1a4bd6029c6addd18

Customer Churn Prediction

In [1]:
```
#IMPORTING NECESSARY LIBRARIES
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
df=pd.read_csv("E:\Course datasets\ML Datasets\Customer_data.csv", header=None)
print(df.head(3))
```

```
            0       1              2        3           4       5   \
0  customerID  gender  SeniorCitizen  Partner  Dependents  tenure
1  7590-VHVEG  Female              0      Yes          No       1
2  5575-GNVDE    Male              0       No          No      34

             6                 7                8               9   ...  \
0  PhoneService     MultipleLines  InternetService  OnlineSecurity  ...
1            No  No phone service              DSL              No  ...
2           Yes                No              DSL             Yes  ...

                 11           12           13               14  \
0  DeviceProtection  TechSupport  StreamingTV  StreamingMovies
1                No           No           No               No
2               Yes           No           No               No

               15                16                17              18  \
0        Contract  PaperlessBilling     PaymentMethod  MonthlyCharges
1  Month-to-month               Yes  Electronic check           29.85
2        One year                No      Mailed check           56.95

             19     20
0  TotalCharges  Churn
1         29.85     No
2        1889.5     No

[3 rows x 21 columns]
```

In [3]:
```
df=pd.read_csv("E:\Course datasets\ML Datasets\Customer_data.csv", header=0)
```

TASK-1

In [4]:
```
print(df.columns)
print(df['Churn'].unique()[:5])
```

```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
['No' 'Yes']
```

In [5]:
```
df['Churn'] = df['Churn'].astype(str).str.strip().str.title()
df['Churn'] = df['Churn'].map({'Yes': 1, 'No': 0})
print(df['Churn'].value_counts(dropna=False))
```

```
Churn
0    5174
1    1869
Name: count, dtype: int64
```

In [6]:
```
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            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      7032 non-null   float64
 20  Churn             7043 non-null   int64
dtypes: float64(2), int64(3), object(16)
memory usage: 1.1+ MB
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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        11
Churn                0
dtype: int64
```

In [8]:
```
#HANDLING THE MISSING VALUES
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df['TotalCharges'].fillna(df['TotalCharges'].median())
```

Out[8]:
```
0         29.85
1       1889.50
2        108.15
3       1840.75
4        151.65
         ...
7038    1990.50
7039    7362.90
7040     346.45
7041     306.60
7042    6844.50
Name: TotalCharges, Length: 7043, dtype: float64
```

In [9]:
```
#REMOVING THE UNNECESSARY COLUMNS
df.drop(columns=["customerID"], inplace=True)
```

In [10]:
```
cat_cols= df.select_dtypes(include=['object']).columns       #SELECTS THE COLUMNS WITH OBJECT DATATYPES
df=pd.get_dummies(df, columns=cat_cols, drop_first=True)      #CHANGES STRING VALUES TO NUMERIC DUMMIES FOR MACHINE TO UNDERSTAND
```

In [11]:
```
#SCALING THE DATA
num_cols=['tenure', 'MonthlyCharges', 'TotalCharges']
df[num_cols]= StandardScaler().fit_transform(df[num_cols])
```

TASK-2

MODEL-1: LOGISTIC REGRESSION

In [12]:
```
#SPLITING THE DATA
x=df.drop('Churn', axis=1)
y=df['Churn']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=42, stratify=y)
```

In [13]:
```
#HANDLING NaN VALUES
x_train=x_train.fillna(x_train.median())
x_test=x_test.fillna(x_test.median())
```

In [14]:
```
#INITIALIZE THE MODEL AND TRAINING
lr_model=LogisticRegression()
lr_model.fit(x_train,y_train)
```

Out[14]:
```
▾ LogisticRegression
LogisticRegression()
```

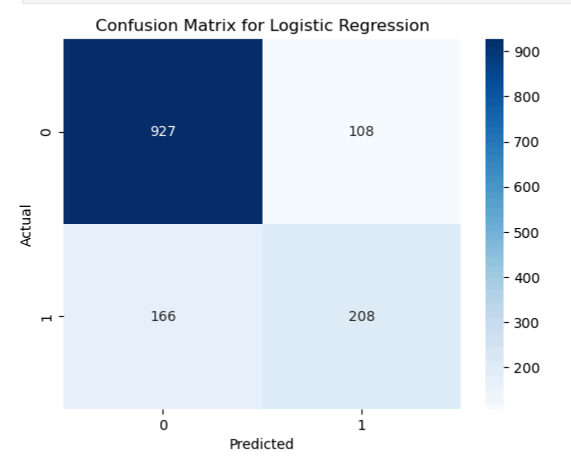
In [15]:
```
#PREDICTIONS
y_pred=lr_model.predict(x_test)
```

TASK-3

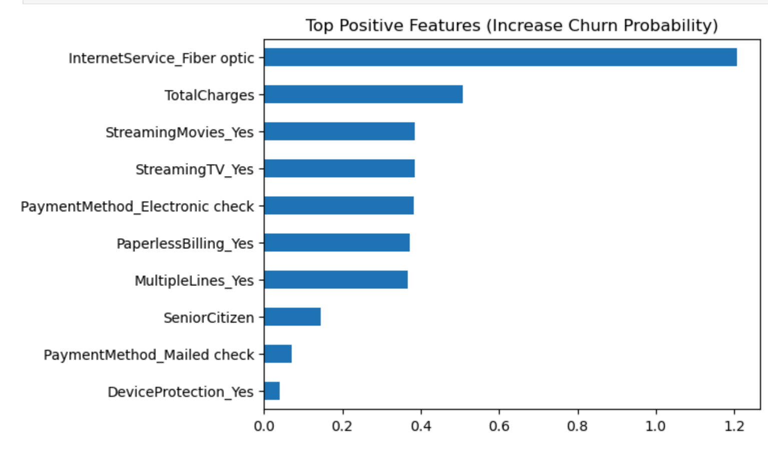
In [16]:
```
accuracy=accuracy_score(y_test,y_pred)
precision=precision_score(y_test,y_pred)
recall=recall_score(y_test, y_pred)
f1=f1_score(y_test, y_pred)

print("Accuracy:", round(accuracy,3))
print("Precision:", round(precision,3))
print("Recall:", round(recall,3))
print("f1 score:", round(f1,3))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.806
Precision: 0.658
Recall: 0.556
f1 score: 0.603
              precision    recall  f1-score   support

           0       0.85      0.90      0.87      1035
           1       0.66      0.56      0.60       374

    accuracy                           0.81      1409
   macro avg       0.75      0.73      0.74      1409
weighted avg       0.80      0.81      0.80      1409
```

In [17]:
```
#CONFUSION MATRIX
cml=confusion_matrix(y_test, y_pred)
sns.heatmap(cml, annot=True, fmt='d', cmap="Blues")
plt.title("Confusion Matrix for Logistic Regression")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [18]:
```
#feature Importance
coef = pd.Series(lr_model.coef_[0], index=x.columns).sort_values()
coef.tail(10).plot(kind='barh', title='Top Positive Features (Increase Churn Probability)')
plt.show()
```



MODEL-2: DECISION TREE CLASSIFIER

In [19]:
```
dt_model=DecisionTreeClassifier(max_depth=6, random_state=42)
dt_model.fit(x_train, y_train)
```

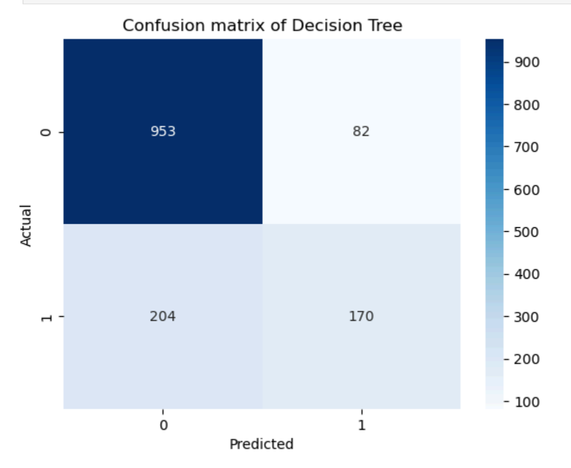
Out[19]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, random_state=42)
```

In [20]:
```
#PREDICTIONS
ydt_pred=dt_model.predict(x_test)
```

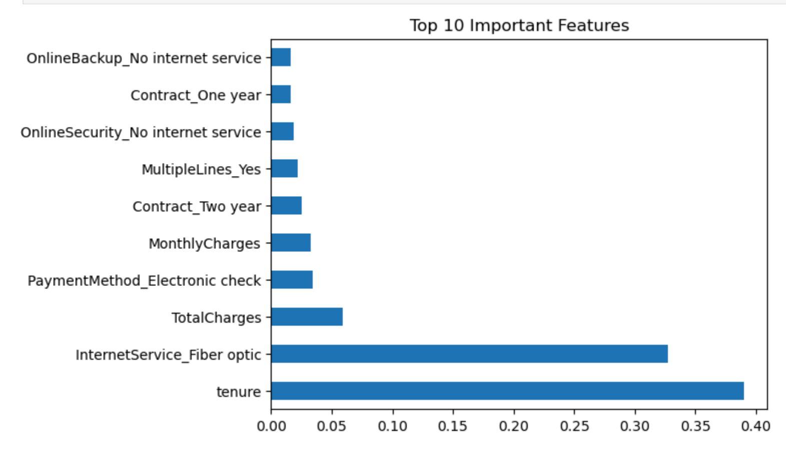
In [21]:
```
#EVALUATIONS
Accuracy=(accuracy_score(y_test,ydt_pred))
Precision=(precision_score(y_test, ydt_pred))
Recall=(recall_score(y_test, ydt_pred))
F1=(f1_score(y_test, ydt_pred))

print("Accuracy:", round(Accuracy, 3))
print("Precision:", round(Precision,3))
print("Recall:", round(Recall,3))
print("F1 score:", round(F1,3))
print(classification_report(y_test, ydt_pred))
```

```
Accuracy: 0.797
Precision: 0.675
Recall: 0.455
F1 score: 0.543
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      1035
           1       0.67      0.45      0.54       374

    accuracy                           0.80      1409
   macro avg       0.75      0.69      0.71      1409
weighted avg       0.78      0.80      0.78      1409
```

In [22]:
```
#CONFUSION MATRIX
cm=confusion_matrix(y_test, ydt_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues")
plt.title("Confusion matrix of Decision Tree")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [23]:
```
#FEATURE IMPORTANCE
importance = pd.Series(dt_model.feature_importances_, index=x.columns)
importance.nlargest(10).plot(kind='barh', title='Top 10 Important Features')
plt.show()
```



MODEL-3: RANDOM FOREST CLASSIFIER

In [24]:
```
rf_model=RandomForestClassifier(n_estimators=200, random_state=42)
rf_model.fit(x_train, y_train)
```

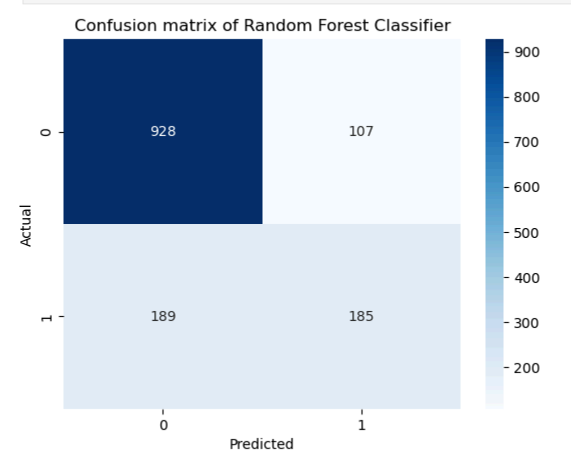
Out[24]:
```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=200, random_state=42)
```

In [25]:
```
#PREDICTIONS
yrf_pred=rf_model.predict(x_test)
```

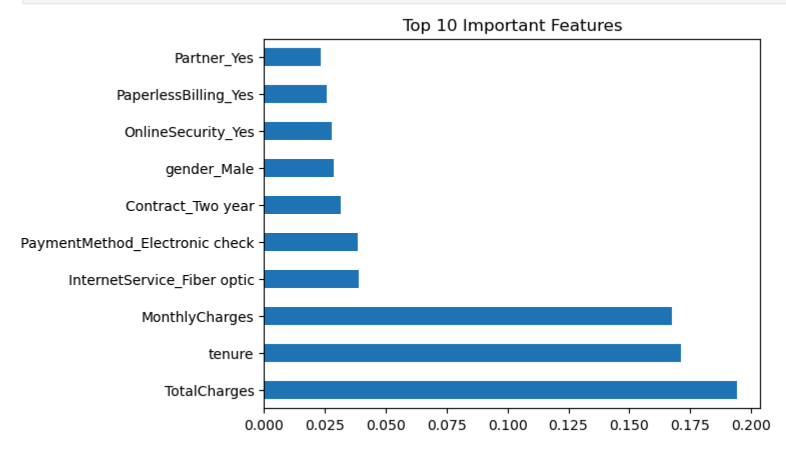
In [27]:
```
#EVALUATIONS
r_accuracy=accuracy_score(y_test,yrf_pred)
r_precision=precision_score(y_test, yrf_pred)
r_recall=recall_score(y_test, yrf_pred)
r_f1=f1_score(y_test, yrf_pred)

print("Accuracy:", round(r_accuracy,3))
print("Precision:", round(r_precision, 3))
print("Recall:", round(r_recall, 3))
print("F1 score:", round(r_f1, 3))
print(classification_report(y_test, yrf_pred))
```

```
Accuracy: 0.79
Precision: 0.634
Recall: 0.495
F1 score: 0.556
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1035
           1       0.63      0.49      0.56       374

    accuracy                           0.79      1409
   macro avg       0.73      0.70      0.71      1409
weighted avg       0.78      0.79      0.78      1409
```

In [28]:
```
#CONFUSION MATRIX
cmr=confusion_matrix(y_test, yrf_pred)
sns.heatmap(cmr, annot=True, fmt='d', cmap="Blues")
plt.title("Confusion matrix of Random Forest Classifier")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [29]:
```
#FEATURE IMPORTANCE
importance = pd.Series(rf_model.feature_importances_, index=x.columns)
importance.nlargest(10).plot(kind='barh', title='Top 10 Important Features')
plt.show()
```



In [30]:
```
#COMPARING THE EVALUATION METRICS OF ALL MODELS
results = pd.DataFrame({
    'Model': ['Logistic Regression', 'Decision Tree', 'Random Forest'],
    'Accuracy': [accuracy, Accuracy, r_accuracy],
    'Precision': [precision, Precision, r_precision],
    'Recall': [recall, Recall, r_recall],
    'F1 Score': [f1, F1, r_f1]})

results = results.round(3)
print(results)
```

```
                 Model  Accuracy  Precision  Recall  F1 Score
0  Logistic Regression     0.806      0.658   0.556     0.603
1        Decision Tree     0.797      0.675   0.455     0.543
2        Random Forest     0.790      0.634   0.495     0.556
```

MODEL EVALUATION INSIGHTS

Logistic Regression

1.Highest Accuracy (0.806) and F1-Score (0.603) among all models. 2.Performs consistently across Precision (0.658) and Recall (0.556).

Insight: A strong, interpretable baseline model with well-balanced performance.

Decision Tree

1.Accuracy 0.797, highest Precision (0.675) but lower Recall (0.455).

Insight: Predicts churn confidently but misses some actual churners — slightly conservative.

Random Forest

1.Accuracy 0.790, Precision 0.634, Recall 0.495, F1 0.556.

Insight: Robust and consistent; tuning hyperparameters could improve recall.

In [ ]: