Project Title: Customer Churn Prediction Using Logistic Regression and Model Comparison on Telco Data

Objective:

To analyze customer behavior, identify factors that contribute to churn, and build predictive models to classify whether a customer is likely to churn or not. The project also compares multiple machine learning algorithms to select the most effective model.

Problem Statement:

Customer churn is a critical issue in the telecom industry. Losing customers can significantly impact revenue. This project aims to use data analysis and machine learning to understand the drivers of churn and provide a predictive solution that helps businesses take proactive measures.

Dataset Overview:

Source: Kaggle - Telco Customer Churn

Rows: 7043

Columns: 21

Target Variable: Churn (Yes/No)

** STEP 1: Import Libraries**

```
# Importing Required Libraries

# Data Handling
import pandas as pd
import numpy as np
```

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

# Machine Learning
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matr
```

Step 2: Load Dataset

```
df = pd.read_csv('/content/drive/MyDrive/Customer Churn Prediction.csv')
df.head()
```

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

5 rows × 21 columns

Step 3: EDA (Exploratory Data Analysis)

```
# Shape of Dataset
print("Shape of Dataset:", df.shape)

# Dataset Info
df.info()

# Summary Statistics
df.describe()
```

```
Shape of Dataset: (7043, 21)
<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
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

Drop customerID → Irrelevant for Prediction

Convert TotalCharges to Numeric

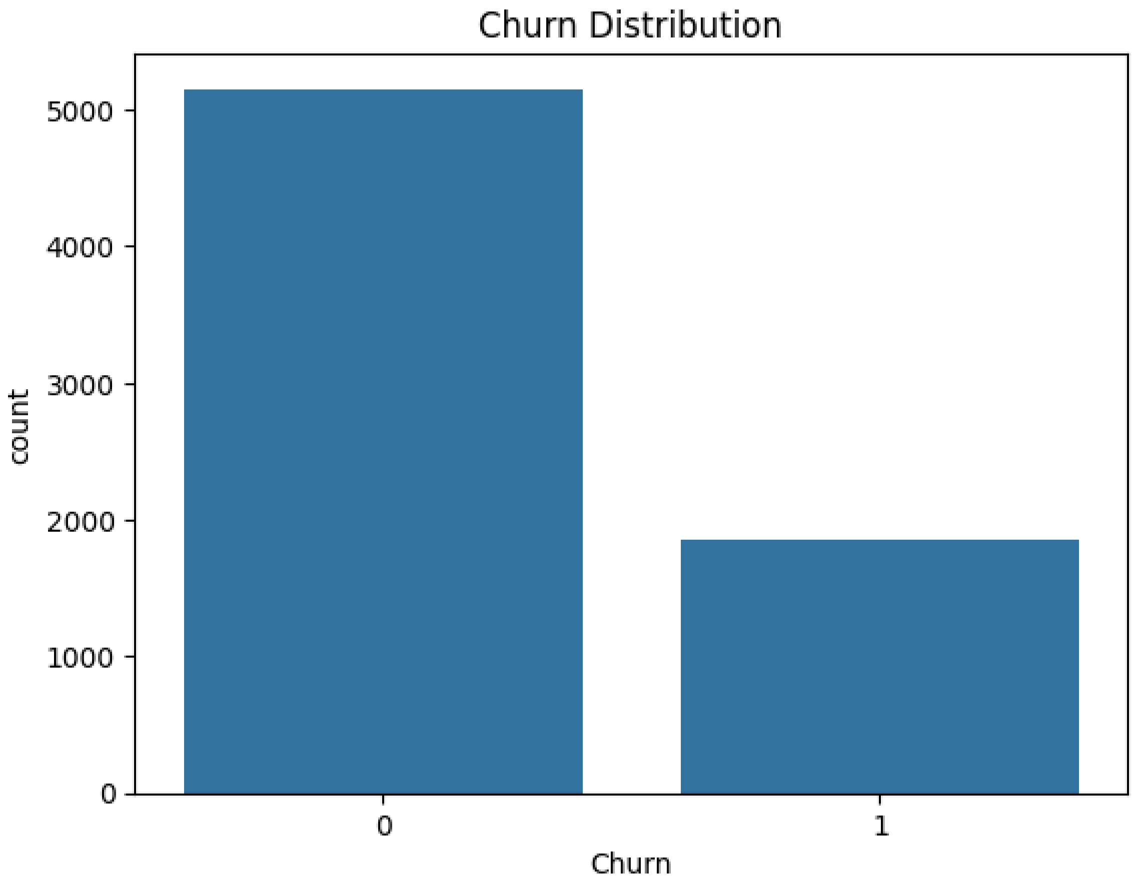
Handle Nulls if any

Step 4: Data Cleaning

```
# Drop Irrelevant Column
df.drop('customerID', axis=1, inplace=True)

# Convert TotalCharges to Numeric
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Drop Rows with Null Values
df.dropna(inplace=True)

# Check Duplicates
df.drop_duplicates(inplace=True)
```

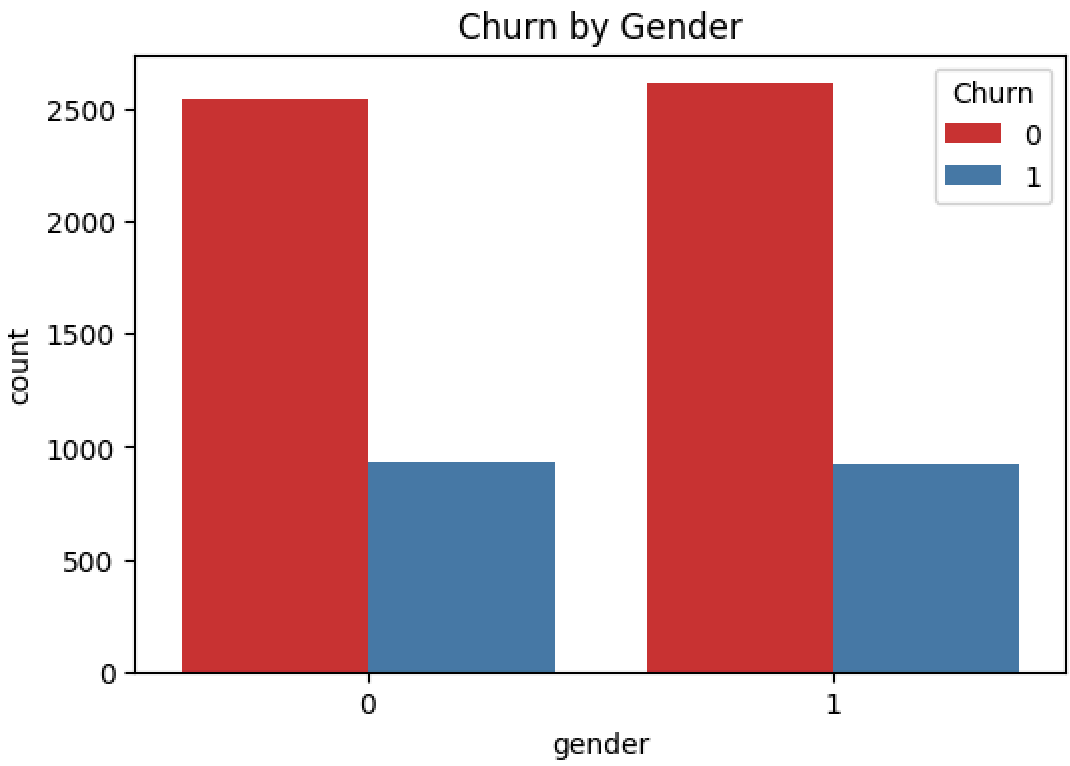
EDA

```
#UNIVARIIANT
## Churn distribution
sns.countplot(x='Churn', data=df)
plt.title("Churn Distribution")
plt.show()
```



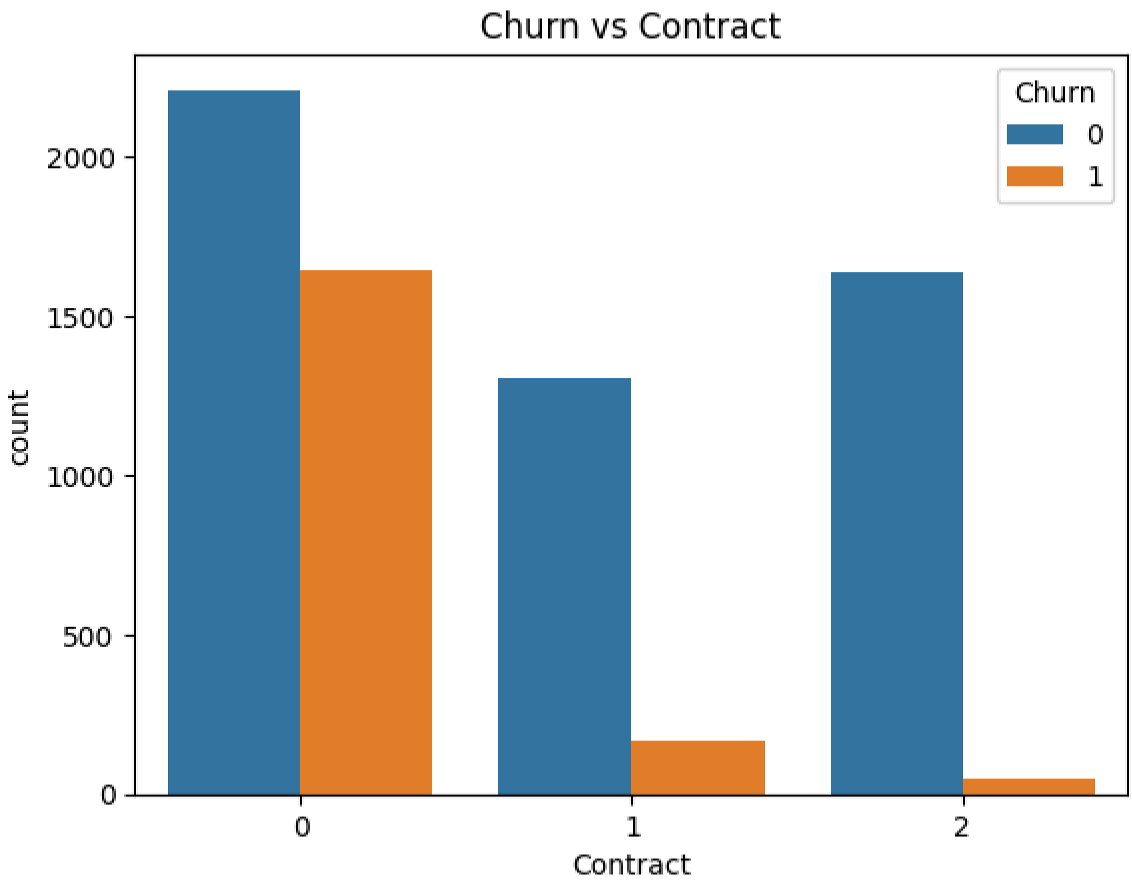

```
# Churn by Gender
plt.figure(figsize=(6,4))
sns.countplot(x='gender', hue='Churn', data=df, palette='Set1')
```

```
plt.title("Churn by Gender")
plt.show()
```

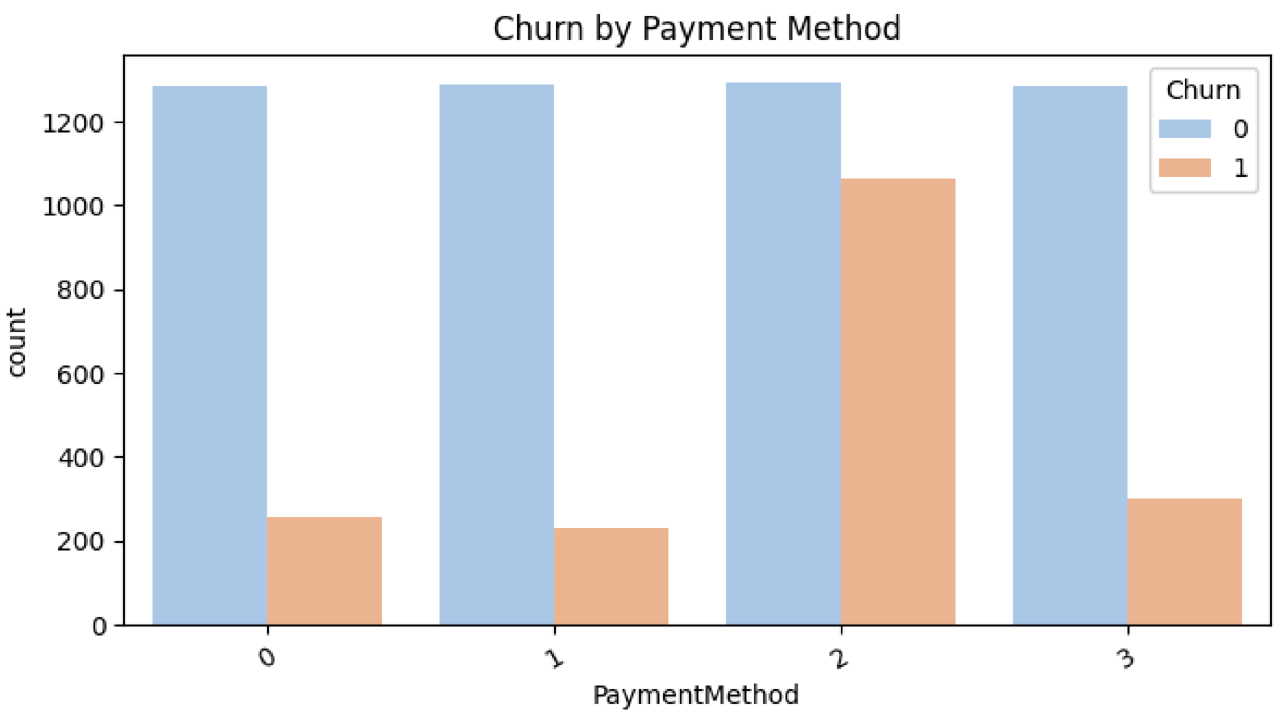


Churn by Contract Type

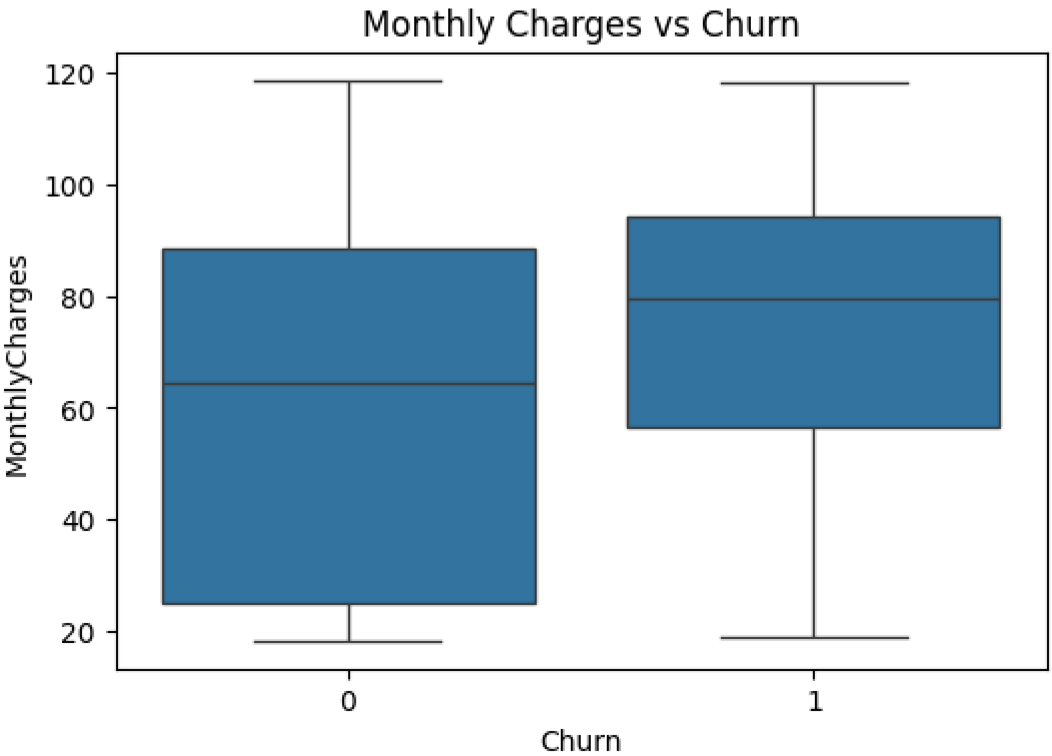
```
#Bivariate Analysis
sns.countplot(x='Contract', hue='Churn', data=df)
plt.title("Churn vs Contract")
plt.show()
```




```
#Churn by Payment Method
plt.figure(figsize=(8,4))
sns.countplot(x='PaymentMethod', hue='Churn', data=df, palette='pastel')
plt.title("Churn by Payment Method")
plt.xticks(rotation=30)
plt.show()
```



```
#Monthly Charges vs Churn (Boxplot)
plt.figure(figsize=(6,4))
sns.boxplot(x='Churn', y='MonthlyCharges', data=df)
plt.title("Monthly Charges vs Churn")
plt.show()
```




```
#Tenure Distribution by Churn
plt.figure(figsize=(6,4))
sns.histplot(data=df, x='tenure', hue='Churn', multiple='stack', bins=30)
plt.title("Tenure Distribution by Churn")
plt.show()
```

Step 5: Encode Categorical Features

```
#ML Models Need Numerical Data
# Label Encoding for Categorical Features
le = LabelEncoder()

for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = le.fit_transform(df[col])
```

Step 6: Feature & Target Split

```
# Define X and y
X = df.drop('Churn', axis=1)
```

```
y = df['Churn']
```

Step 7: Train-Test Split

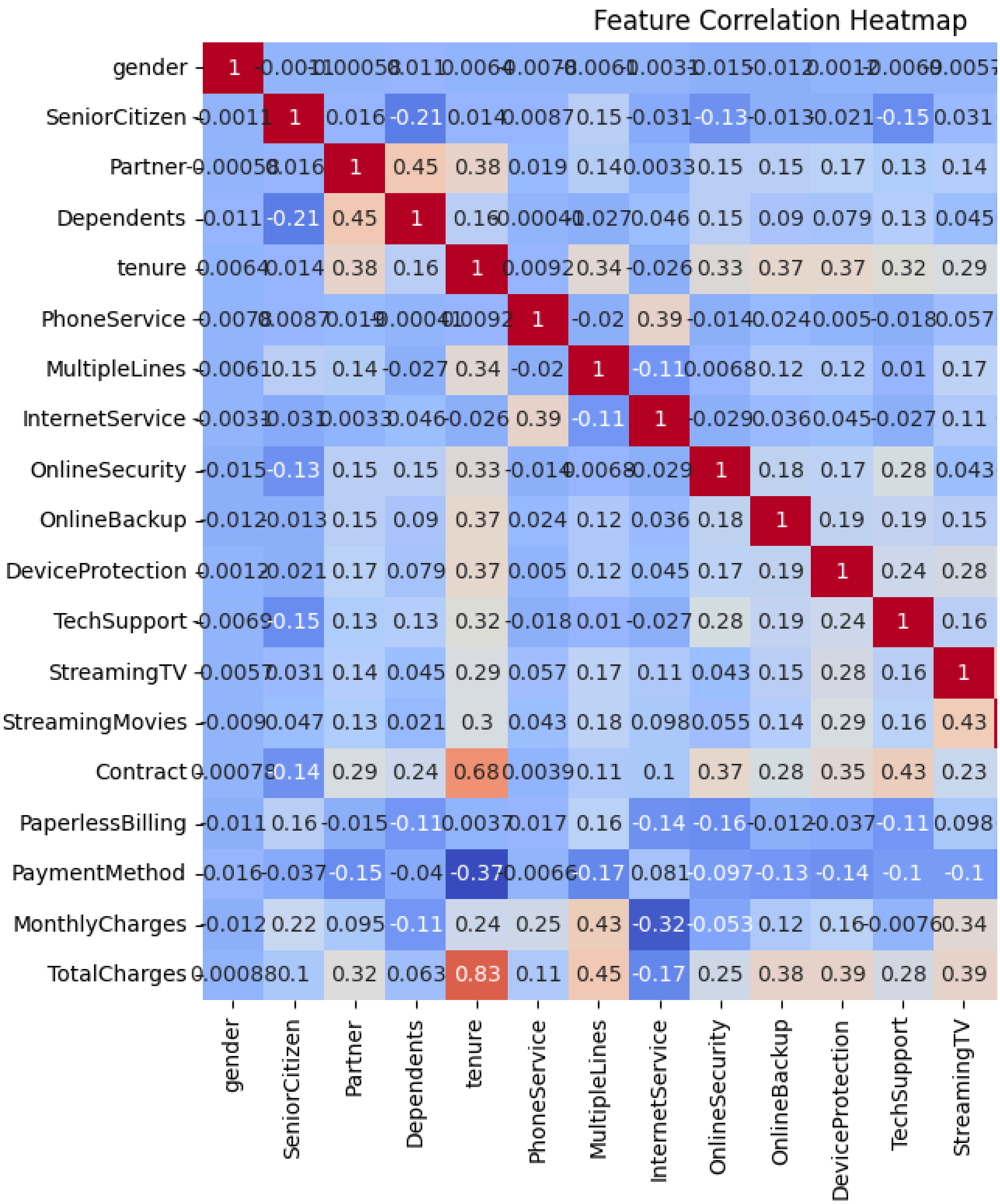
```
# Splitting Data into Train and Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
```

Step 8: Feature Scaling (Important for SVM & KNN)

```
# Scaling Data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 9: Checking Assumptions of Logistic Regression

```
#Assumption 1: No Multicollinearity
#We Check using Correlation & VIF (Variance Inflation Factor)
# Correlation Heatmap
plt.figure(figsize=(12,8))
sns.heatmap(X.corr(), annot=True, cmap='coolwarm')
plt.title("Feature Correlation Heatmap")
plt.show()
```



VIF value

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
```

Step 2: Calculate VIF for Each Feature

```
# Adding Constant Term for VIF Calculation
X_const = sm.add_constant(X)

# Create Dataframe for VIF
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns

# Calculate VIF for Each Feature
vif_data['VIF'] = [variance_inflation_factor(X_const.values, i+1) for i in range(

vif_data
```

	Feature	VIF
0	gender	1.001847
1	SeniorCitizen	1.149441
2	Partner	1.457258
3	Dependents	1.378798
4	tenure	7.487740
5	PhoneService	1.623337
6	MultipleLines	1.392872
7	InternetService	1.819220
8	OnlineSecurity	1.268132
9	OnlineBackup	1.218183
10	DeviceProtection	1.296837
11	TechSupport	1.321031
12	StreamingTV	1.446139
13	StreamingMovies	1.446964
14	Contract	2.492075
15	PaperlessBilling	1.201457
16	PaymentMethod	1.182203
17	MonthlyCharges	4.979241
18	TotalCharges	10.638410

Next steps: View recommended plots New interactive sheet

→ If VIF > 5 → Multicollinearity Problem → It will affect Logistic Regression Model Accuracy.

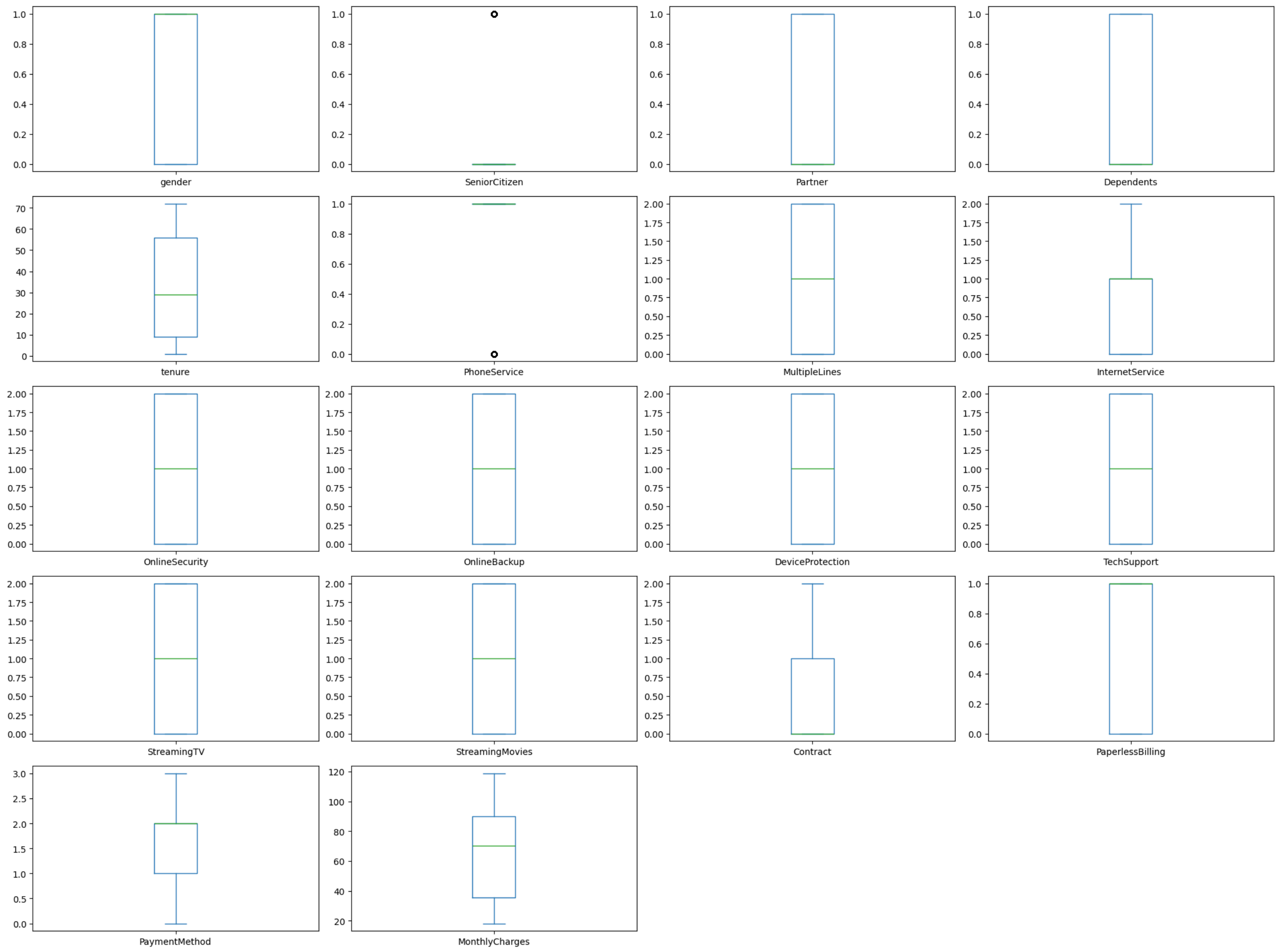
```
# total charge has vip value 10.63 so i remove that feature
# Drop TotalCharges Column Due to High VIF
```

```
X.drop('TotalCharges', axis=1, inplace=True)
```

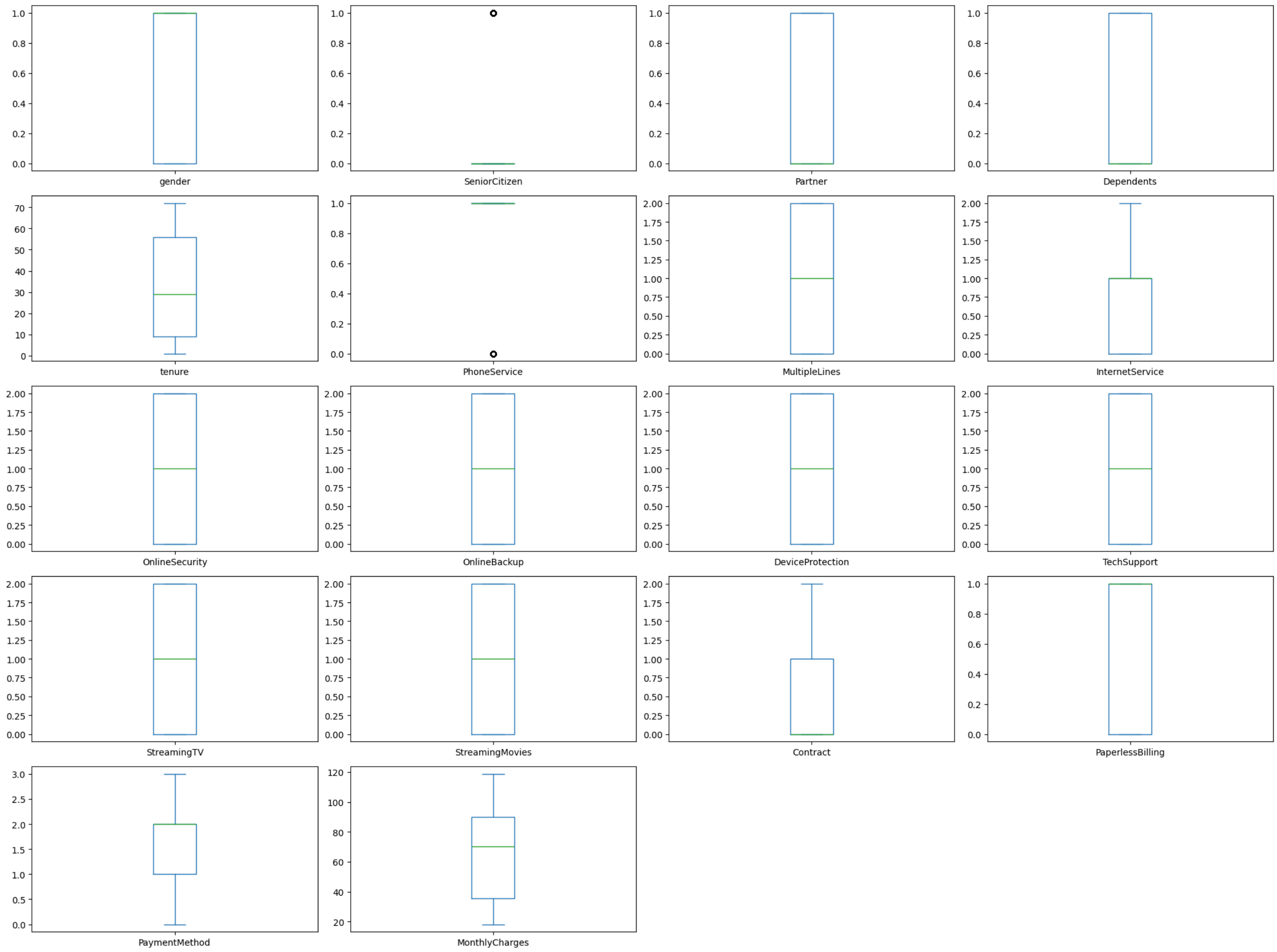
After calculating VIF, I found that the TotalCharges feature had a high VIF of 10.63, indicating multicollinearity. Since TotalCharges is dependent on Tenure and MonthlyCharges, I dropped it to avoid multicollinearity in my Logistic Regression model."

Assumption 2: No Outliers

```
#Check Using Boxplot
# Boxplot For Checking Outliers (2nd Assumption)
X.plot(kind='box', subplots=True, layout=(5,4), figsize=(20,15), sharex=False, sh
plt.tight_layout()
plt.show()
```


gender
SeniorCitizen
Partner
Dependents
tenure
PhoneService
MultipleLines
InternetService
OnlineSecurity
OnlineBackup
DeviceProtection
TechSupport
StreamingTV
StreamingMovies
Contract
PaperlessBilling
PaymentMethod
MonthlyCharges

I used Boxplot to check outliers for all numerical features. Since Logistic Regression is robust to small outliers, I handled only extreme cases if needed



I checked outliers using boxplot for all features. There were no extreme outliers in important numerical features like tenure & MonthlyCharges. Logistic Regression can handle small outliers after scaling, so I did not apply capping or removal.

Since I am applying multiple models in my project, I checked outliers using boxplot. Logistic Regression, SVM, and KNN models are sensitive to outliers, but small outliers were handled using scaling. Tree-based models like Decision Tree & Random Forest are robust to outliers, so no outlier removal was needed."

Assumption 3: Feature Scaling

Since Logistic Regression, SVM, and KNN are distance and weight-based algorithms, they are highly sensitive to the scale of data. Therefore, I applied feature scaling using StandardScaler() from sklearn to standardize all numerical features (mean=0, standard deviation=1).

This ensures that no feature dominates the model due to its higher magnitude and improves the model's performance and stability

Step 10: Logistic Regression Model Building

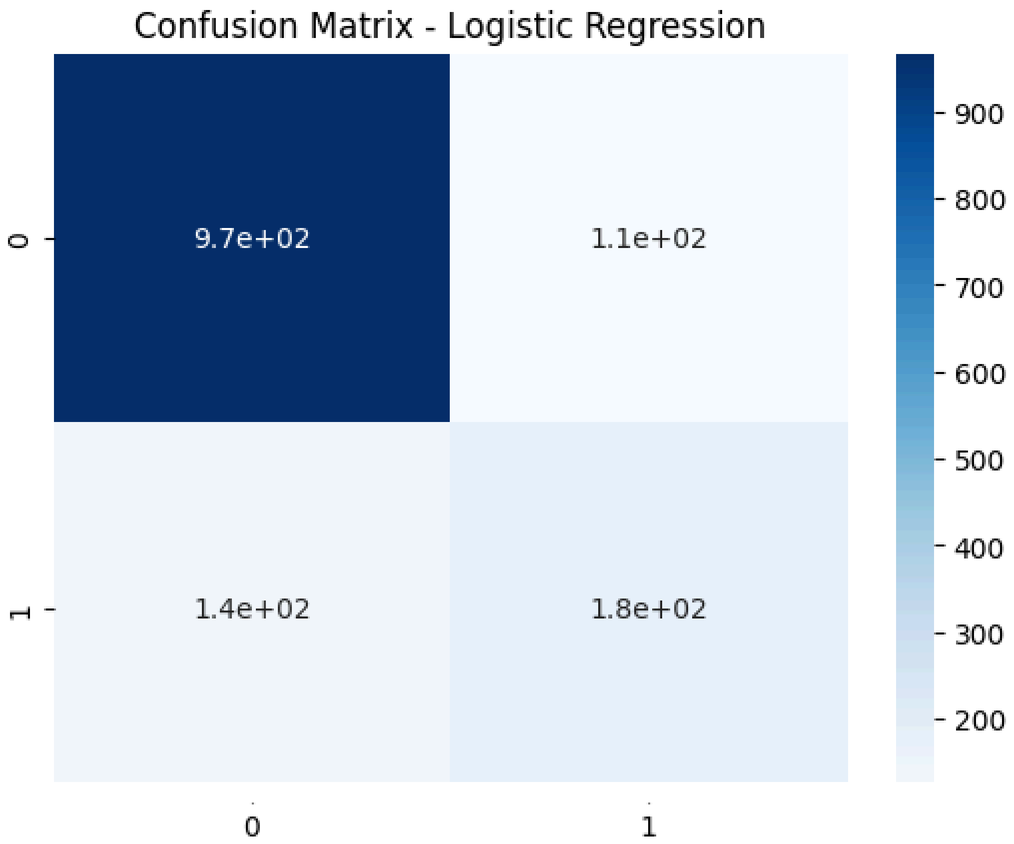
```
# Step 10: Train Logistic Regression Model
lr = LogisticRegression()

# Fit Model
lr.fit(X_train_scaled, y_train)

# Predict on Test Data
y_pred_lr = lr.predict(X_test_scaled)
```

Step 11: Evaluation of Logistic Regression:

```
# Logistic Regression with Cross Validation

# Importing Required Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matr
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Instantiate Model
lr = LogisticRegression()

# Step 2: Apply Cross Validation (5 Fold)
lr_cv_scores = cross_val_score(lr, X_train_scaled, y_train, cv=5)

print("Cross Validation Scores in 5 Splits:", lr_cv_scores)
print("Average CV Accuracy:", lr_cv_scores.mean())

# Step 3: Train Model on Train Data
lr.fit(X_train_scaled, y_train)

# Step 4: Predict on Test Data
y_pred_lr = lr.predict(X_test_scaled)

# Step 5: Evaluate Model
print("Accuracy without CV (On Test Data):", accuracy_score(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))

# Step 6: Confusion Matrix
cm_lr = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cm_lr, annot=True, cmap='Blues')
plt.title("Confusion Matrix - Logistic Regression")
```

```
plt.show()
```

```
Cross Validation Scores in 5 Splits: [0.8057041  0.81729055 0.79679144 0.7591
Average CV Accuracy: 0.7963568624270729
Accuracy without CV (On Test Data): 0.8209700427960057
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.89      0.89      1081
           1       0.62      0.57      0.59       321

    accuracy                           0.82      1402
   macro avg       0.75      0.73      0.74      1402
weighted avg       0.82      0.82      0.82      1402
```

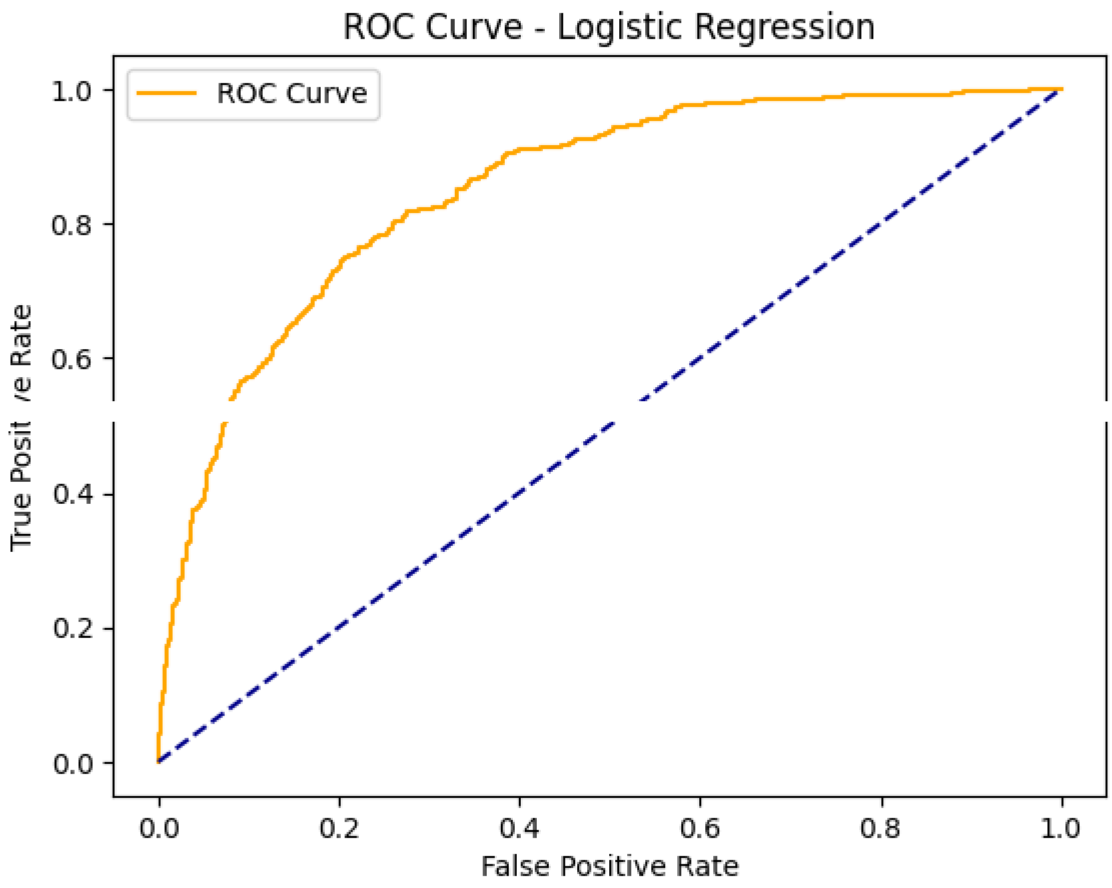


Step 12: ROC-AUC Score:

```
# ROC-AUC Score
print("ROC-AUC Score:", roc_auc_score(y_test, lr.predict_proba(X_test_scaled)[:,1

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, lr.predict_proba(X_test_scaled)[:,1])

plt.plot(fpr, tpr, color='orange', label='ROC Curve')
plt.plot([0,1], [0,1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend()
```

```
plt.show()
```

```
ROC-AUC Score: 0.8511963942467025
```



After checking all assumptions, I applied Logistic Regression as a base model. The model was evaluated using accuracy, precision, recall, F1-score, and ROC-AUC. Logistic Regression gave a good baseline result on the Telco Churn Dataset.

I applied 5-Fold Cross Validation on Logistic Regression to avoid overfitting and check model stability. Accuracy improved from Test Data accuracy to Cross Validation average accuracy. This confirms the robustness of the Logistic Regression model.

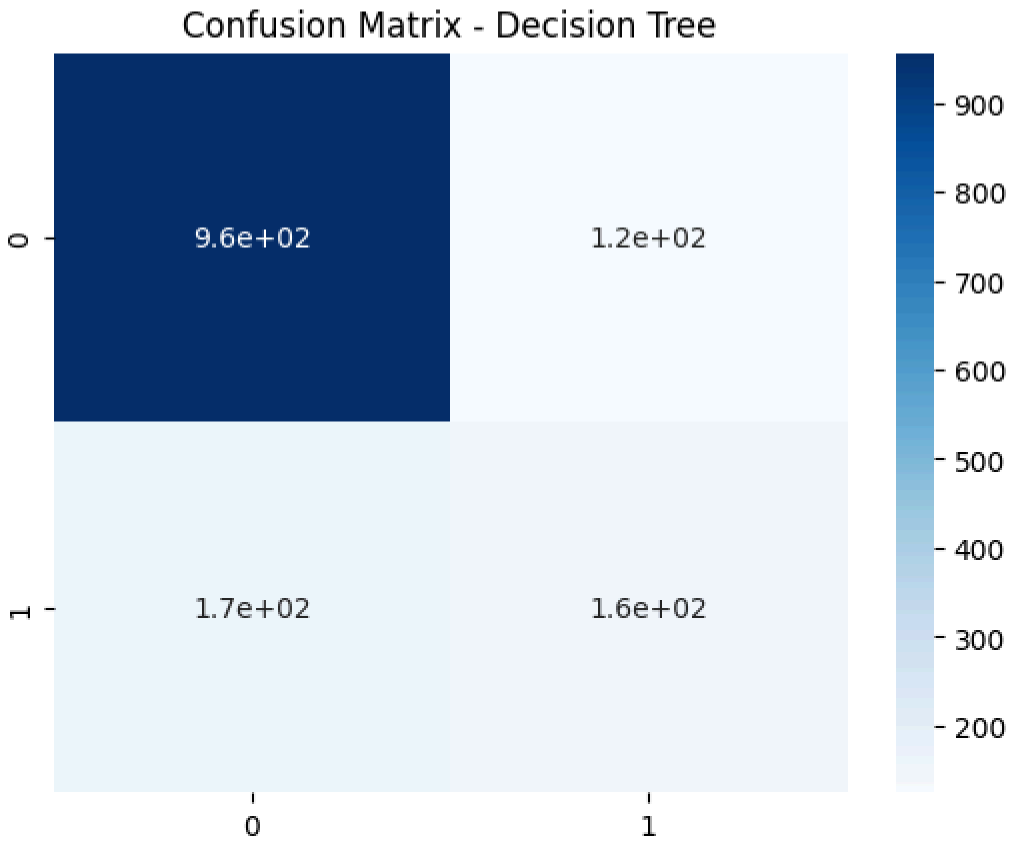
Step 13: Apply Decision Tree Classifier

Decision Tree with Cross Validation + Hyperparameter Tuning (Best Industry Approach)

```
# Decision Tree with Cross Validation + Hyperparameter Tuning

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matr

# Step 1: Define Parameter Grid
```

```
param_dt = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10]
}

# Step 2: Apply GridSearchCV with 5 Fold CV
dt = DecisionTreeClassifier(random_state=42)

grid_dt = GridSearchCV(dt, param_dt, cv=5)
grid_dt.fit(X_train, y_train)

# Step 3: Best Parameters
print("Best Parameters for Decision Tree:", grid_dt.best_params_)

# Step 4: Predict on Test Data
y_pred_dt = grid_dt.predict(X_test)

# Step 5: Evaluation
print("Accuracy without CV (Default DT):", accuracy_score(y_test, DecisionTreeCla
print("Accuracy After CV & Tuning:", accuracy_score(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))

# Step 6: Confusion Matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm_dt, annot=True, cmap='Blues')
plt.title("Confusion Matrix - Decision Tree")
plt.show()
```

```
Best Parameters for Decision Tree: {'max_depth': 7, 'min_samples_split': 10}
Accuracy without CV (Default DT): 0.7310984308131241
Accuracy After CV & Tuning: 0.7924393723252496
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.88      0.87      1081
           1       0.55      0.48      0.52       321

    accuracy                           0.79      1402
   macro avg       0.70      0.68      0.69      1402
weighted avg       0.78      0.79      0.79      1402
```



"I applied Cross Validation with Hyperparameter Tuning on Decision Tree using GridSearchCV. After tuning max_depth and min_samples_split parameters, model performance improved and overfitting was reduced.

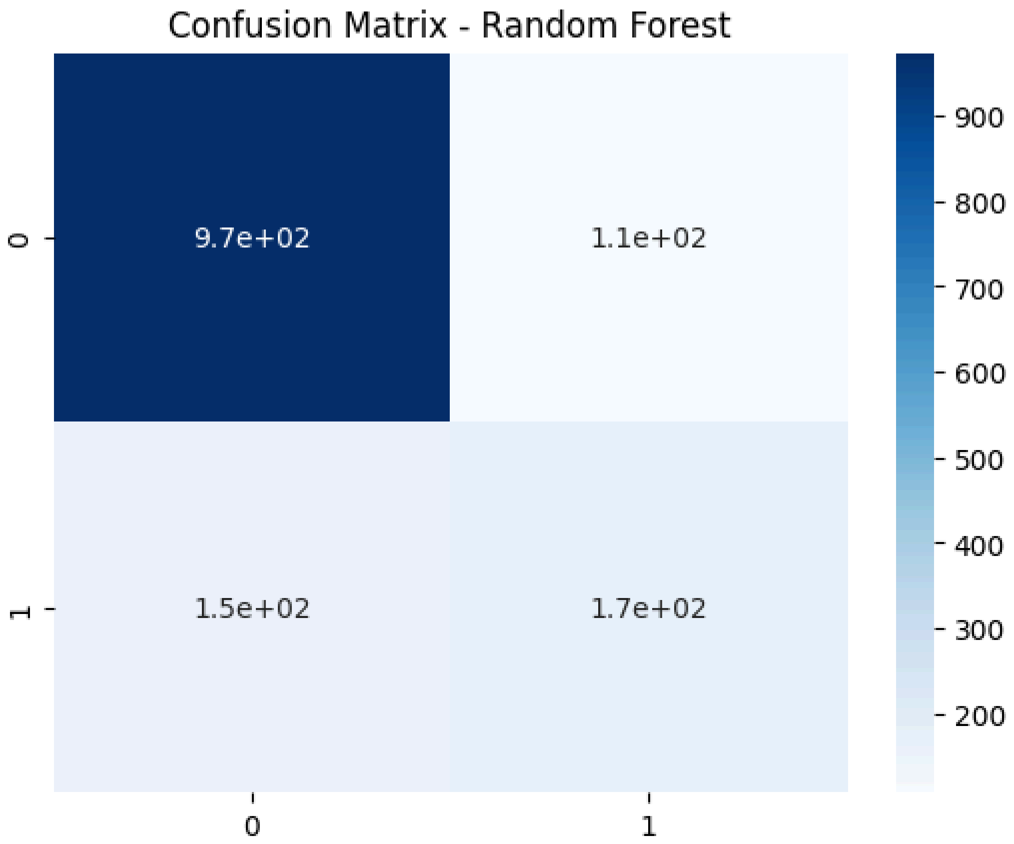
Step 14: Apply Random Forest Classifier (Best Expected Model)

Random Forest with Cross Validation + Hyperparameter Tuning

```
# Random Forest with Cross Validation + Hyperparameter Tuning

from sklearn.ensemble import RandomForestClassifier
```

```
# Step 1: Define Parameter Grid
param_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10]
}

# Step 2: Apply GridSearchCV with 5 Fold CV
rf = RandomForestClassifier(random_state=42)

grid_rf = GridSearchCV(rf, param_rf, cv=5)
grid_rf.fit(X_train, y_train)

# Step 3: Best Parameters
print("Best Parameters for Random Forest:", grid_rf.best_params_)

# Step 4: Predict on Test Data
y_pred_rf = grid_rf.predict(X_test)

# Step 5: Evaluation
print("Accuracy without CV (Default RF):", accuracy_score(y_test, RandomForestCla
print("Accuracy After CV & Tuning:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))

# Step 6: Confusion Matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, cmap='Blues')
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

```
Best Parameters for Random Forest: {'max_depth': 10, 'min_samples_split': 10,
Accuracy without CV (Default RF): 0.7995720399429387
Accuracy After CV & Tuning: 0.8145506419400856
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1081
           1       0.61      0.52      0.56       321

    accuracy                           0.81      1402
   macro avg       0.74      0.71      0.72      1402
weighted avg       0.81      0.81      0.81      1402
```



Random Forest performed the best in my project. After applying Cross Validation and Hyperparameter Tuning using GridSearchCV, I optimized n_estimators, max_depth, and min_samples_split. This improved accuracy and reduced model overfitting.

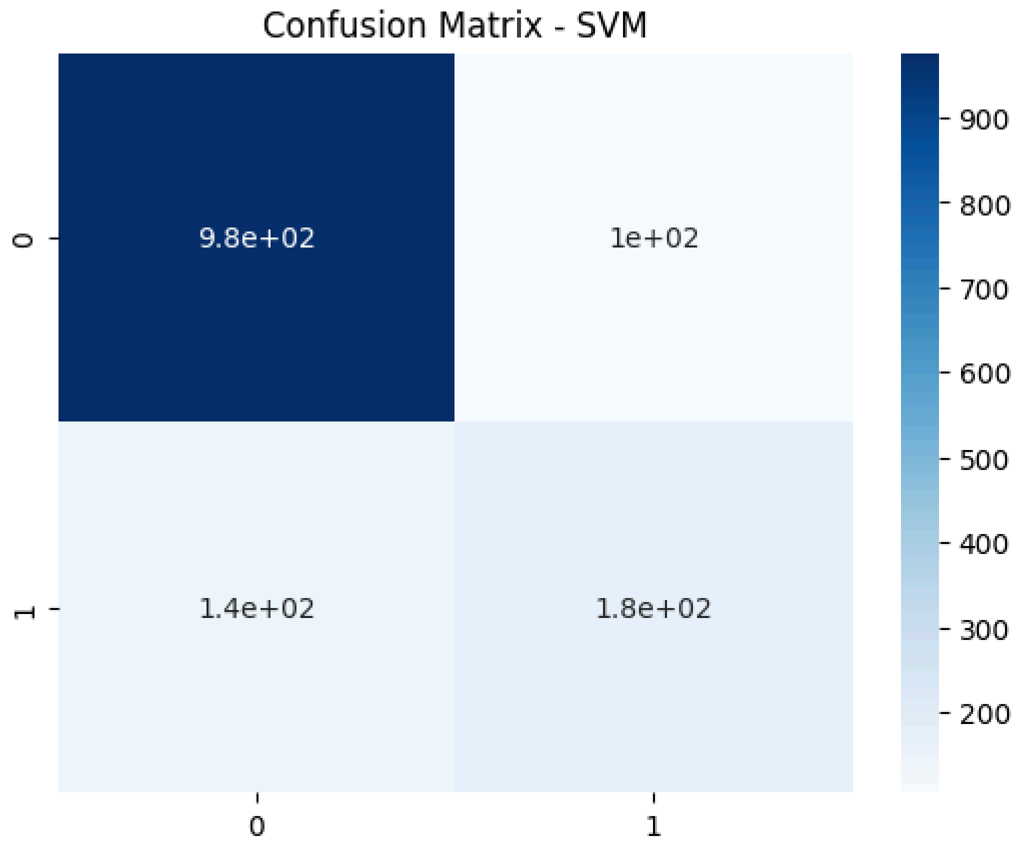
Step 15: Apply SVM Classifier (Scaling Mandatory)

Step 4: SVM (Before CV & After CV with Hyperparameter Tuning)

```
# SVM Before CV
from sklearn.svm import SVC

# Default SVM Model
svm = SVC(probability=True)
```

```
svm.fit(X_train_scaled, y_train)
y_pred_svm = svm.predict(X_test_scaled)

print("SVM Accuracy Before CV & Tuning:", accuracy_score(y_test, y_pred_svm))

# Apply Hyperparameter Tuning with CV
param_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid_svm = GridSearchCV(SVC(probability=True), param_svm, cv=5)
grid_svm.fit(X_train_scaled, y_train)

print("Best Parameters for SVM:", grid_svm.best_params_)

# Predict After CV
y_pred_svm_cv = grid_svm.predict(X_test_scaled)

print("SVM Accuracy After CV & Tuning:", accuracy_score(y_test, y_pred_svm_cv))
print("Classification Report:\n", classification_report(y_test, y_pred_svm_cv))

# Confusion Matrix
cm_svm = confusion_matrix(y_test, y_pred_svm_cv)
sns.heatmap(cm_svm, annot=True, cmap='Blues')
plt.title("Confusion Matrix - SVM")
plt.show()
```

```
SVM Accuracy Before CV & Tuning: 0.81811697574893
Best Parameters for SVM: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
SVM Accuracy After CV & Tuning: 0.8231098430813124
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.90      0.89      1081
           1       0.63      0.55      0.59       321

    accuracy                           0.82      1402
   macro avg       0.75      0.73      0.74      1402
weighted avg       0.82      0.82      0.82      1402
```



Step 16: Apply KNN Classifier (Scaling Mandatory)

KNN (Before CV & After CV with Hyperparameter Tuning)

```
# KNN Before CV
from sklearn.neighbors import KNeighborsClassifier

# Default KNN Model
knn = KNeighborsClassifier()
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)

print("KNN Accuracy Before CV & Tuning:", accuracy_score(y_test, y_pred_knn))

# Apply Hyperparameter Tuning with CV
```

```
param_knn = {
    'n_neighbors': list(range(1, 21))
}

grid_knn = GridSearchCV(KNeighborsClassifier(), param_knn, cv=5)
grid_knn.fit(X_train_scaled, y_train)

print("Best k Value for KNN:", grid_knn.best_params_)

# Predict After CV
y_pred_knn_cv = grid_knn.predict(X_test_scaled)

print("KNN Accuracy After CV & Tuning:", accuracy_score(y_test, y_pred_knn_cv))
print("Classification Report:\n", classification_report(y_test, y_pred_knn_cv))

# Confusion Matrix
cm_knn = confusion_matrix(y_test, y_pred_knn_cv)
sns.heatmap(cm_knn, annot=True, cmap='Blues')
plt.title("Confusion Matrix - KNN")
plt.show()
```

```
KNN Accuracy Before CV & Tuning: 0.7532097004279601
Best k Value for KNN: {'n_neighbors': 16}
KNN Accuracy After CV & Tuning: 0.7902995720399429
Classification Report:
              precision    recall  f1-score   support
```

all model accuracy