Predicting Customer Churn for a Telecommunication Company

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab

NAME: Shyam

REGISTRATION NO.: 12111001

1.Introduction:

- Project Objective:

Predict customer churn using machine learning techniques.

- Data Source:

'Churn.csv'

- Tools and Libraries Used:

Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, XGBoost

2. Data Loading and Preprocessing

- 2.1 Import Libraries:

- **pandas (pd)**

  -Uses:

  1. **Data Loading and Manipulation:** Used to load data from CSV files (e.g., pd.read_csv) and manipulate it using dataframes for cleaning, transformation, and analysis (e.g., selecting rows/columns, filtering data).
  2. **Data Analysis:** Provides functions for descriptive statistics (e.g., mean, median, standard deviation), time series analysis (e.g., resampling data), and creating data visualizations (e.g., plotting dataframes).

- **numpy (np)**

  -Uses:

  1. **Numerical Computations:** Offers efficient arrays for numerical operations (e.g., vectorized calculations, matrix operations) like element-wise addition, multiplication, and linear algebra functions.
  2. **Data Analysis:** Provides functions for array manipulation (e.g., reshaping, indexing), random number generation (e.g., generating random samples from various distributions), and linear algebra operations (e.g., solving systems of equations).

- **seaborn (sns)**

  -Uses:

  1. **Statistical Visualization:** Built on top of matplotlib, it provides high-level functions for creating informative and aesthetically pleasing statistical graphics like boxplots, violin plots, and jointplots to visualize relationships between variables.
  2. **Distribution Plots:** Offers functions for creating histograms, kernel density plots, and cumulative distribution functions to explore the distribution of data.

- **matplotlib.pyplot (plt)**

  -Uses:

1. **Basic Plotting:** Provides functions for creating basic plots like line plots, scatter plots, and bar charts to visualize data trends and relationships.
2. **Customization:** Allows for customization of plot elements like labels, titles, colors, and legends for clear and informative data representation.

- **sklearn.linear_model (LogisticRegression)**

  -Uses:

  1. **Classification:** Implements the Logistic Regression algorithm for binary classification problems, predicting the probability of an outcome based on features (e.g., churn prediction).
  2. **Regression Analysis:** Can also be used for linear regression tasks to model the relationship between a continuous dependent variable and one or more independent variables.

- **sklearn.metrics (accuracy_score, confusion_matrix, classification_report)**

  -Uses:

  1. **Model Evaluation:** Provides functions like accuracy_score to measure overall model performance, confusion_matrix to visualize the distribution of true vs. predicted labels, and classification_report for detailed metrics like precision, recall, and F1-score.

- **sklearn.preprocessing (LabelEncoder)**

  -Uses:

  1. **Categorical Encoding:** Used for converting categorical data (e.g., text labels) into numerical values suitable for machine learning algorithms. LabelEncoder assigns a unique integer to each category.
  2. **Data Scaling:** Other functions like StandardScaler can be used to scale numerical features to a common range for improved model performance.

- **imblearn.over_sampling (SMOTE)**

  -Uses:

  1. **Data Balancing:** Addresses class imbalance problems in datasets, where one class has significantly fewer samples than others. SMOTE (Synthetic Minority Oversampling Technique) creates synthetic samples for the minority class.
  2. **Under-Sampling:** Other techniques like random under-sampling can also be used to reduce the majority class size and achieve a more balanced dataset.

- **collections (Counter)**

  -Uses:

  1. **Frequency Analysis:** Provides the Counter class to count the occurrences of elements in an iterable object (e.g., list, dictionary). Useful for exploring the distribution of categorical data or imbalanced classes.

2. **Set Operations:** Offers functions for working with sets, such as finding the union, intersection, and difference between sets, which can be helpful for data manipulation tasks.

- **sklearn.model_selection (train_test_split)**

  -Uses:

  1. **Data Splitting:** Splits a dataset into training and testing sets for model training and evaluation. train_test_split allows for specifying the desired test size ratio and randomization for robust model assessment.

- **sklearn.tree (DecisionTreeClassifier)**

  -Uses:

  1. **Classification:** Implements the Decision Tree algorithm for classification tasks. It builds a tree-like structure based on feature values to predict class labels.

  2. **Feature Importance:** Can be used to understand the relative importance of features in the model by analyzing the splits made at each node in the tree.

- **sklearn.ensemble (RandomForestClassifier, GradientBoostingClassifier)**

  -Uses:

  1. **Ensemble Learning:** Both Random Forest and Gradient Boosting are ensemble methods that combine multiple weak learners (e.g., decision trees) to create a stronger model.
  2. **Model Regularization:** These techniques help to reduce

- **sklearn.neighbors (KNeighborsClassifier)**

  -Uses:

  1. **Classification:** Used for classifying data points based on the k nearest neighbors in the training data. It predicts the class label that is most frequent among the k nearest neighbors for a new data point.

- **sklearn.svm (SVC)**

  -Uses:

  1. **Classification:** Implements Support Vector Machines (SVM) for classification tasks. It finds a hyperplane that best separates the data points of different classes with the maximum margin.

- **xgboost (XGBClassifier)**

  -Uses:

  1. **Classification (Advanced):** XGBoost is a powerful machine learning library for Gradient Boosting, a technique that builds an ensemble of decision trees sequentially. It's often used for complex classification problems due to its accuracy and efficiency.

- **plotly.express (px)**

-Uses:

1. **Interactive Data Visualization:** Provides a high-level interface for creating interactive visualizations like scatter plots, bar charts, and heatmaps using plotly.

Useful for exploring relationships between variables and understanding model predictions.

- **statsmodels.api (sm)**

  -Uses:

  1. **Statistical Modeling:** Offers a comprehensive library for statistical modeling and econometrics. While it can be used for various tasks, a common use case in machine learning is for fitting statistical models like linear regression and analyzing their properties.

2.2 Load Data:

- Data File: 'Churn.csv'

- Dataframe: churn_data

```
churn_data=pd.read_csv('/content/sample_data/Churn.csv')
```

2.3 Data Cleaning:

Missing Values:

Strategy: Converted empty strings in 'totalcharges' to the mean value.

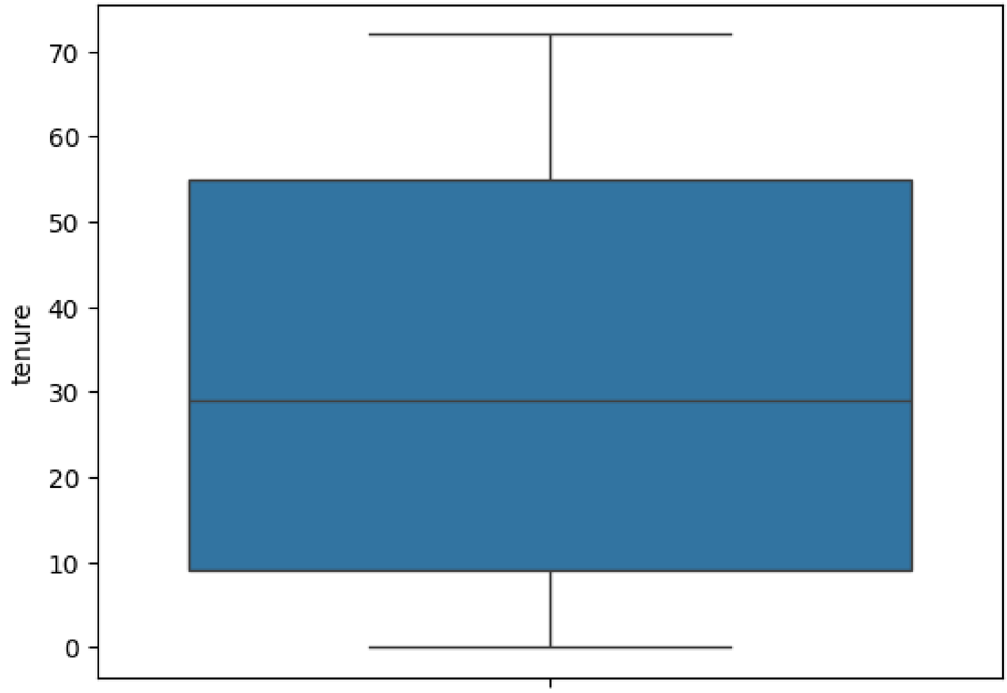
```
[304] churn_data['totalcharges'].describe()

count    7032.000000
mean     2283.300441
std      2266.771362
min        18.800000
25%       401.450000
50%      1397.475000
75%      3794.737500
max      8684.800000
Name: totalcharges, dtype: float64
```

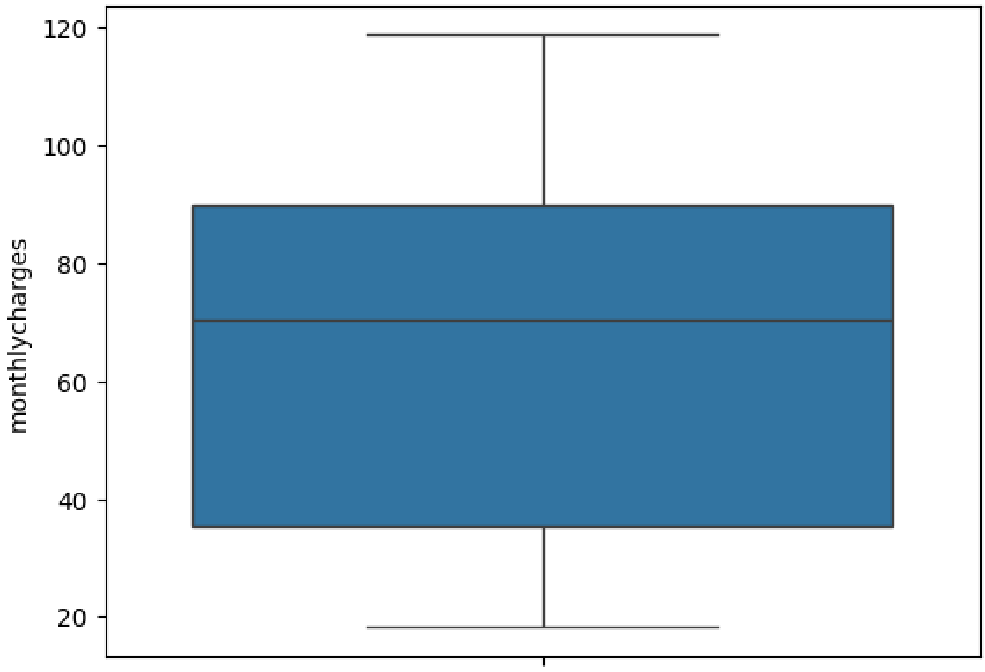
```
[305] churn_data['totalcharges'].fillna(value=2283.300441,inplace=True)
```

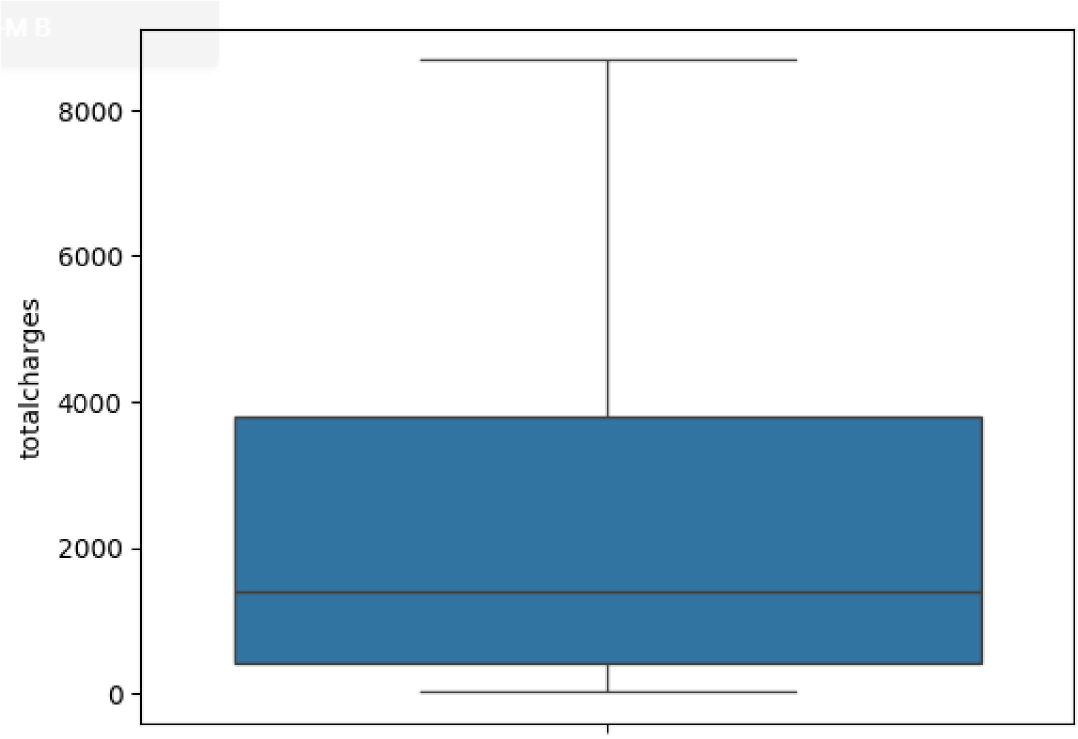
Outliers:

- Identification: Used boxplots.

```python
columns=['tenure','monthlycharges','totalcharges']
for i in columns:
  sns.boxplot(churn_data[i])
  plt.show()
```







- Decision: Kept outliers as they were not extreme.

Duplicates:

- Action: duplicates are not present.

```
churn_data.duplicated().sum()
```

```
0
```

Irrelevant Columns:

- Action: Removed 'customerID'.

customerid is not correlated to the target variable.so I remove the customerid column.

```
[311] churn_data.drop('customerid', axis=1, inplace=True)
      churn_data.head()
```

Categorical Data:

_-Change abnormal categorical values:

```
churn_data['multiplelines'] = churn_data['multiplelines'].replace('No phone service', 'No')
```

```python
data_columns = ['internetservice', 'onlinesecurity', 'onlinebackup', 'deviceprotection', 'techsupport',
                'streamingtv', 'streamingmovies']

for column in data_columns:
  churn_data[column] = churn_data[column].replace('No internet service', 'No')
```

- Strategy: Converted categorical data to numerical values using LabelEncoder.

```python
categorical = list(churn_data.select_dtypes(include=['object']).columns)
categorical
```

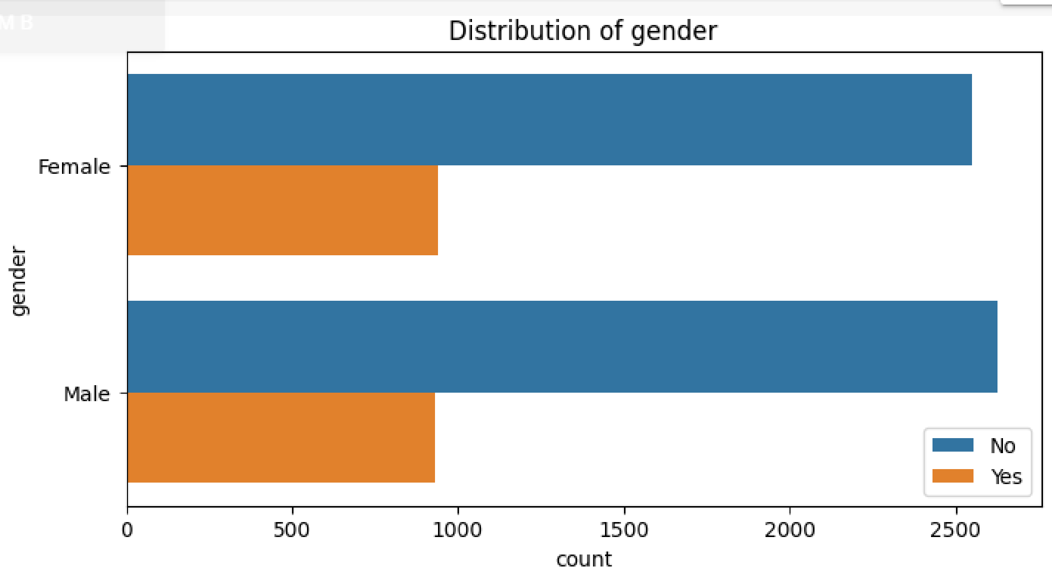
```python
from sklearn.preprocessing import LabelEncoder
LB=LabelEncoder()
for i in categorical:
  churn_data[i]=LB.fit_transform(churn_data[i])
```

3. Exploratory Data Analysis (EDA)

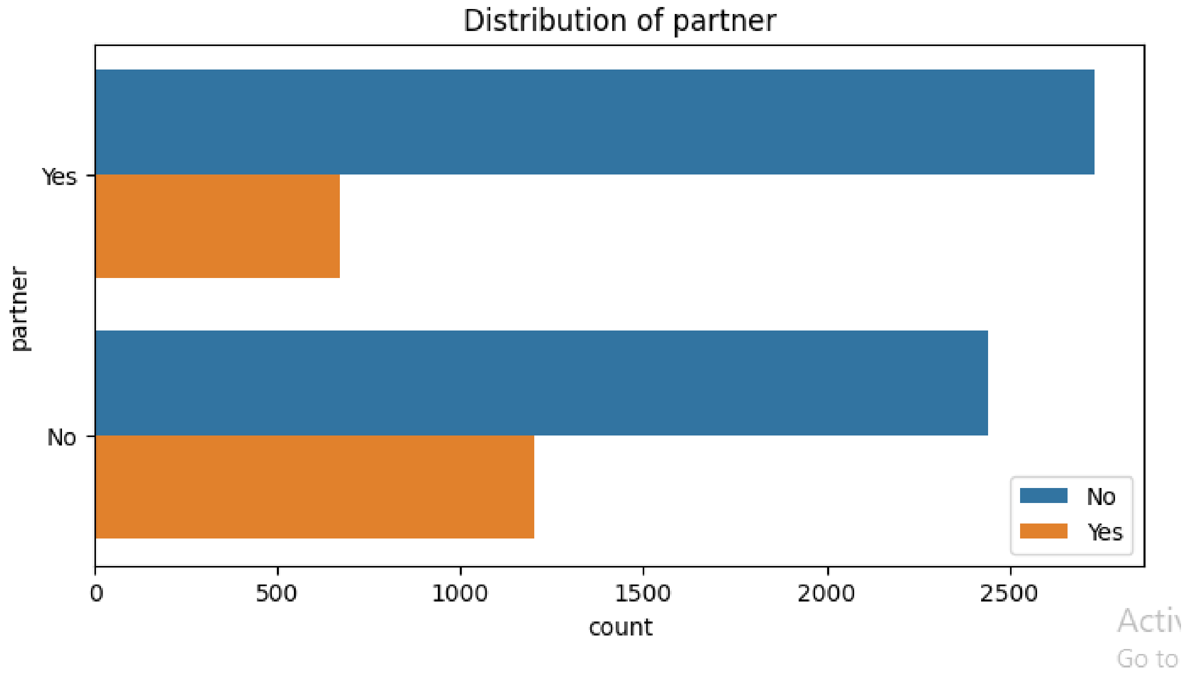
3.1 Visualization :

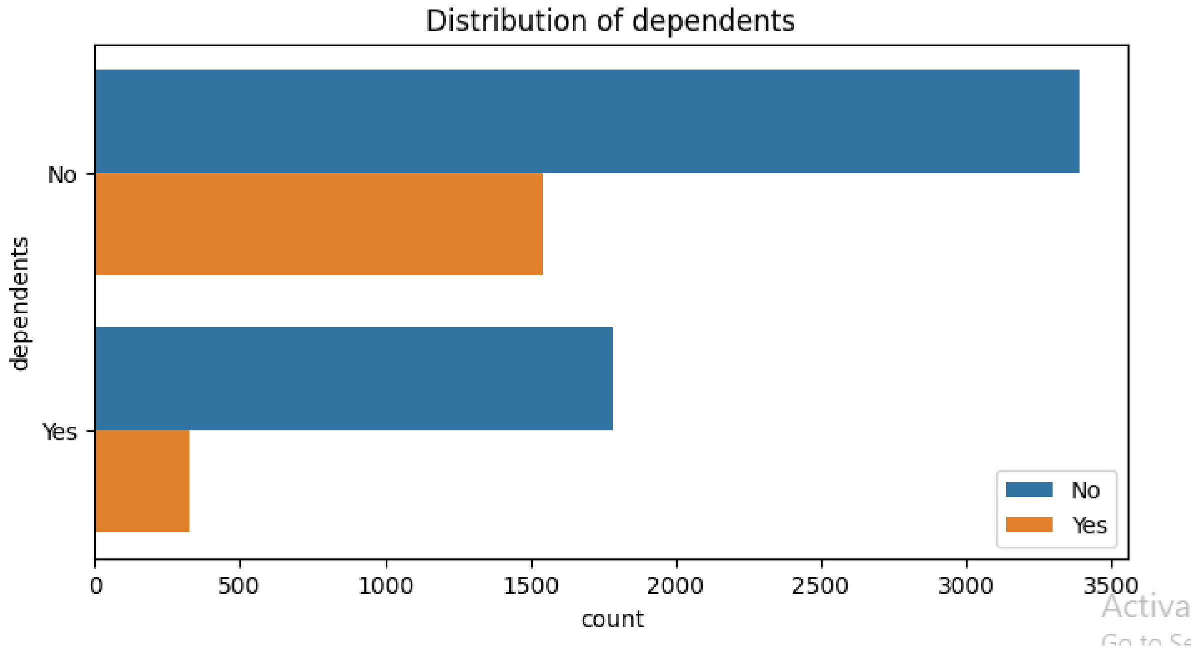
- Categorical Features: Count plots to visualize the distribution and relationship with churn.

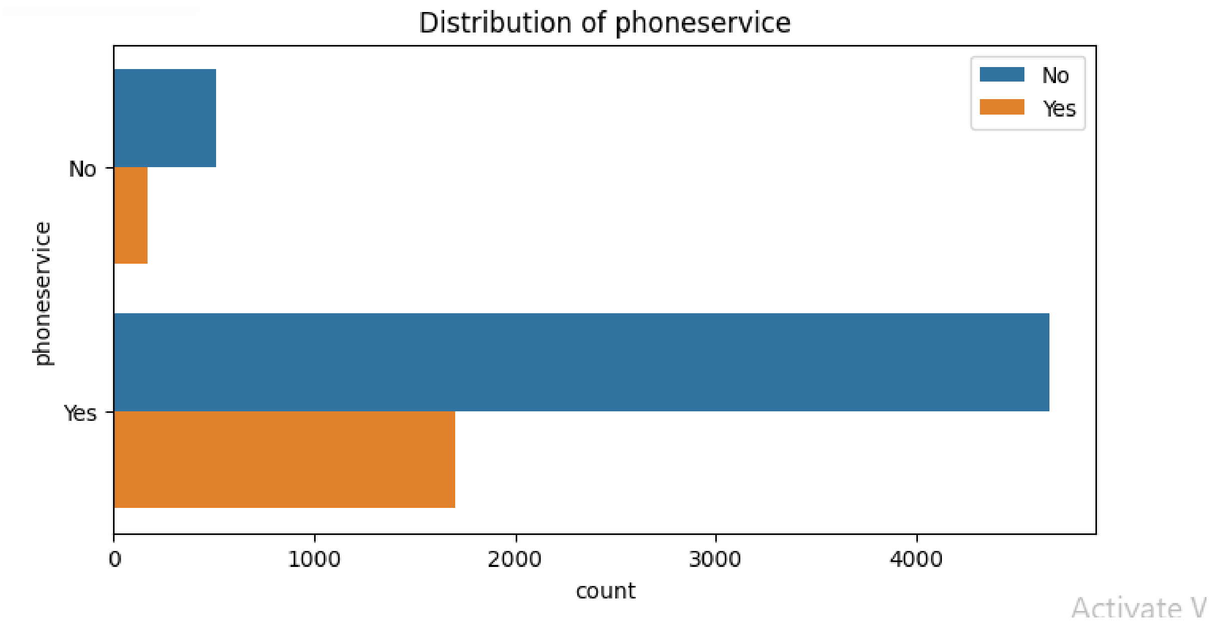
```python
for i in categorical:
    plt.figure(figsize=(8, 4))
    sns.countplot(y=i, data=churn_data, hue='churn')
    plt.title(f'Distribution of {i}')
    plt.legend()
    plt.show()
```

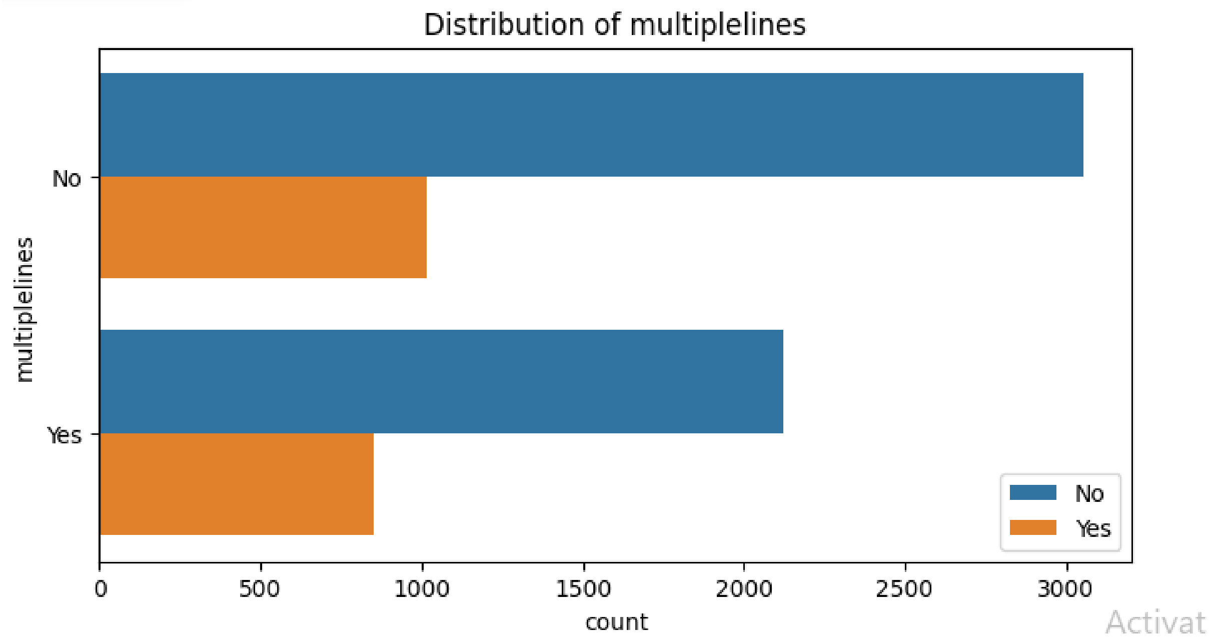


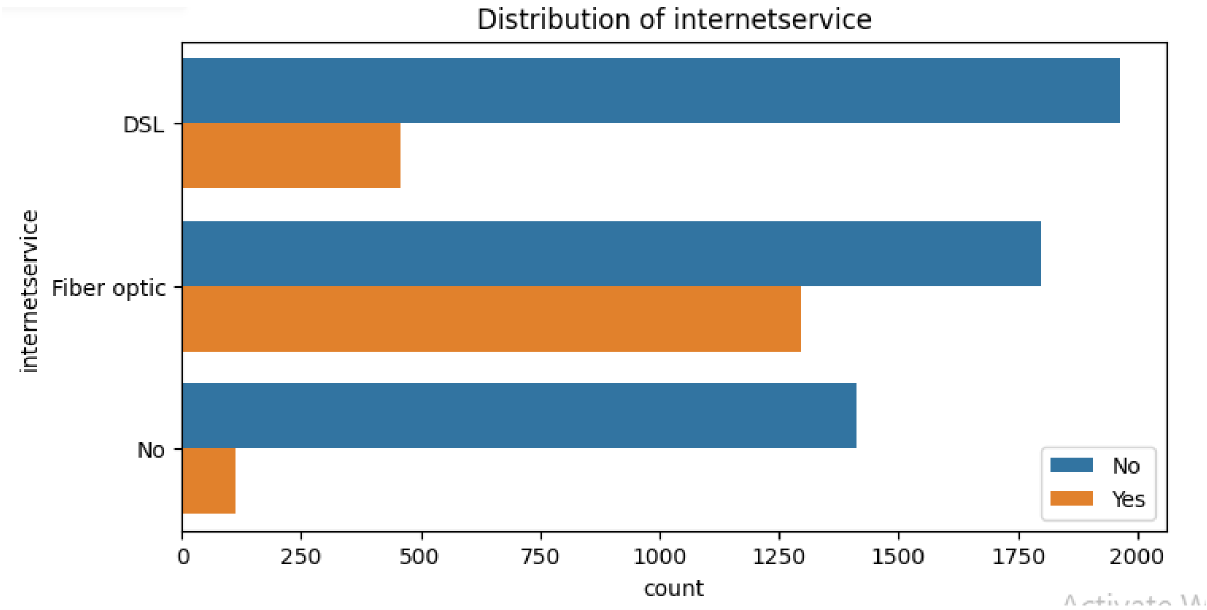
count

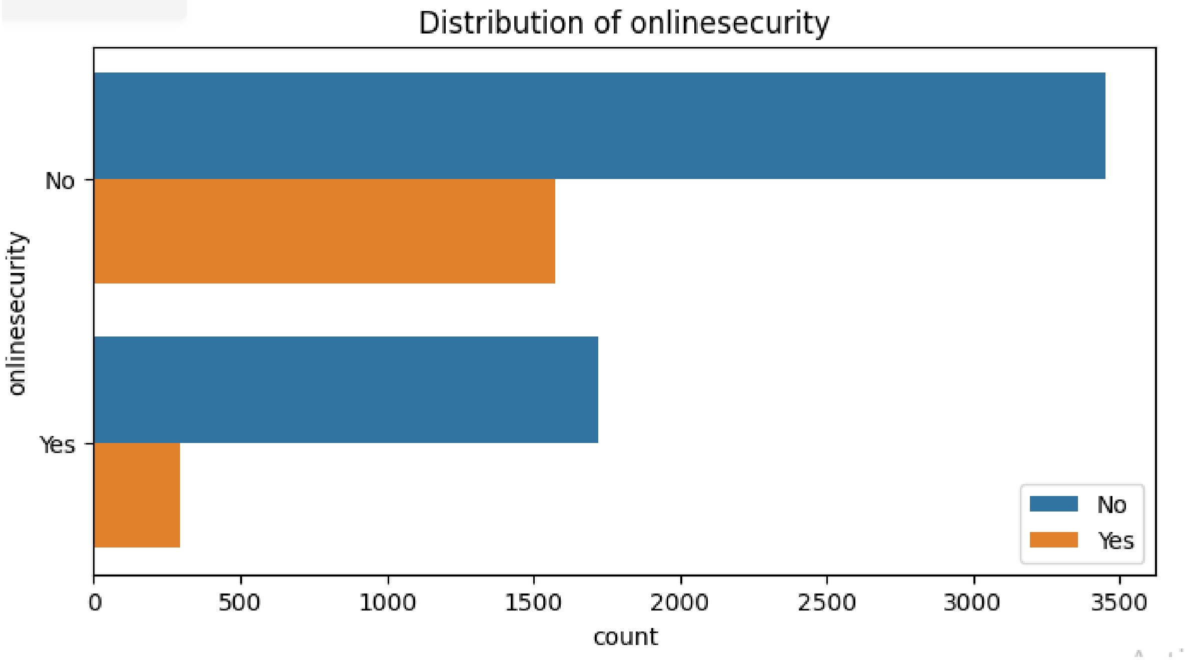

Distribution of partner
partner
Yes
No
0
500
1000
1500
2000
2500
count
No
Yes

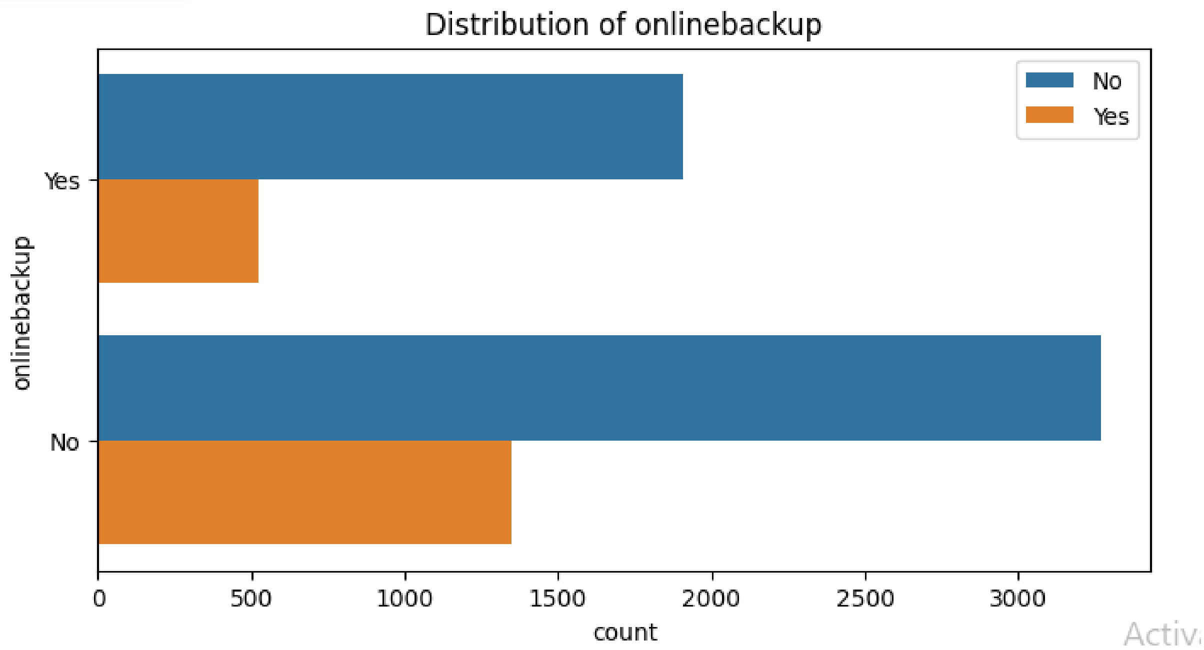

Distribution of dependents
dependents
No
Yes
0
500
1000
1500
2000
2500
3000
3500
count
No
Yes

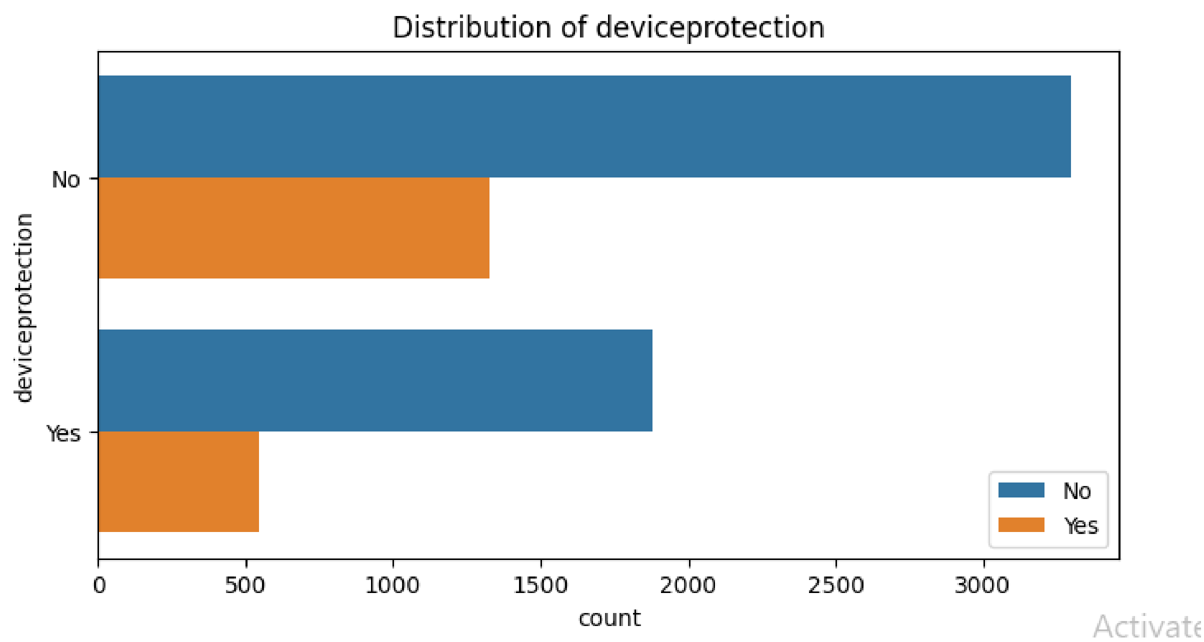

Distribution of phoneservice
No
Yes
phoneservice
No
Yes
0
1000
2000
3000
4000
count
Activate V

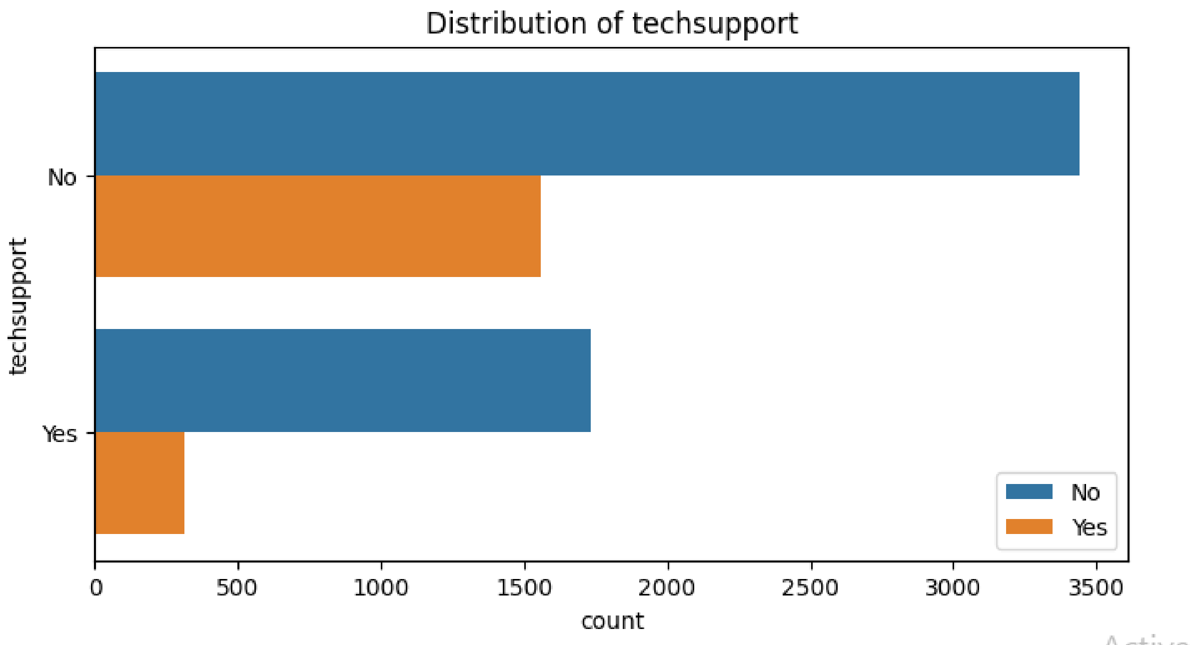

Distribution of multiplelines
multiplelines
No
Yes
No
Yes
0
500
1000
1500
2000
2500
3000
count
Activat

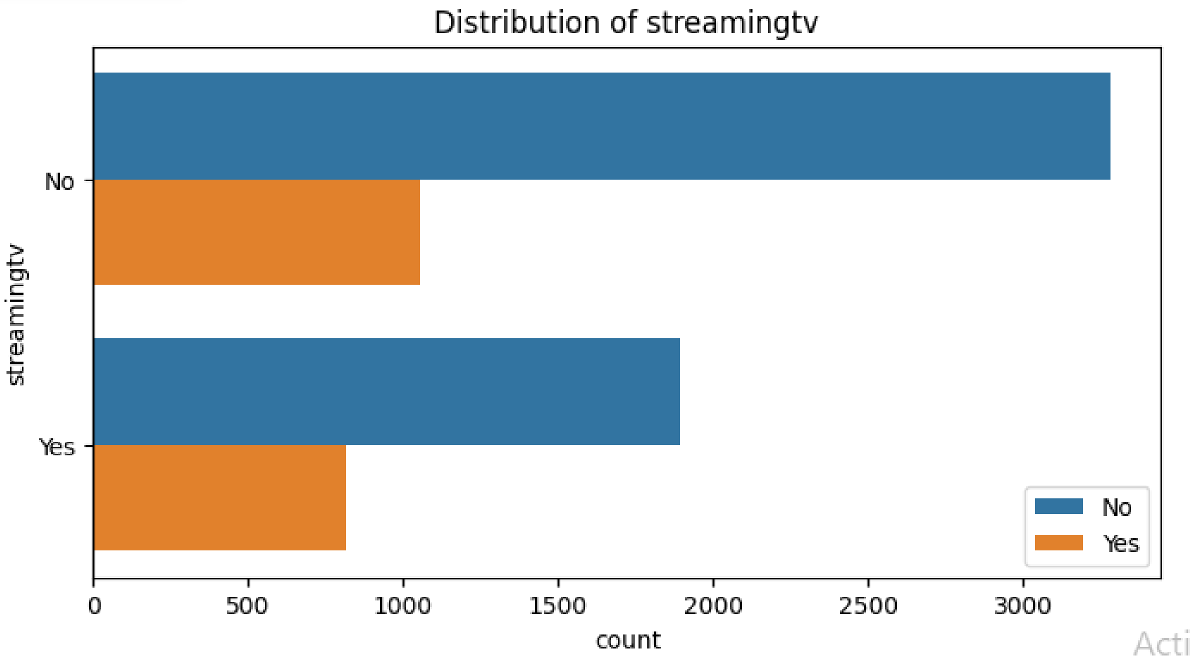

Distribution of internetservice
internetservice
DSL
Fiber optic
No
No
Yes
0
250
500
750
1000
1250
1500
1750
2000
count

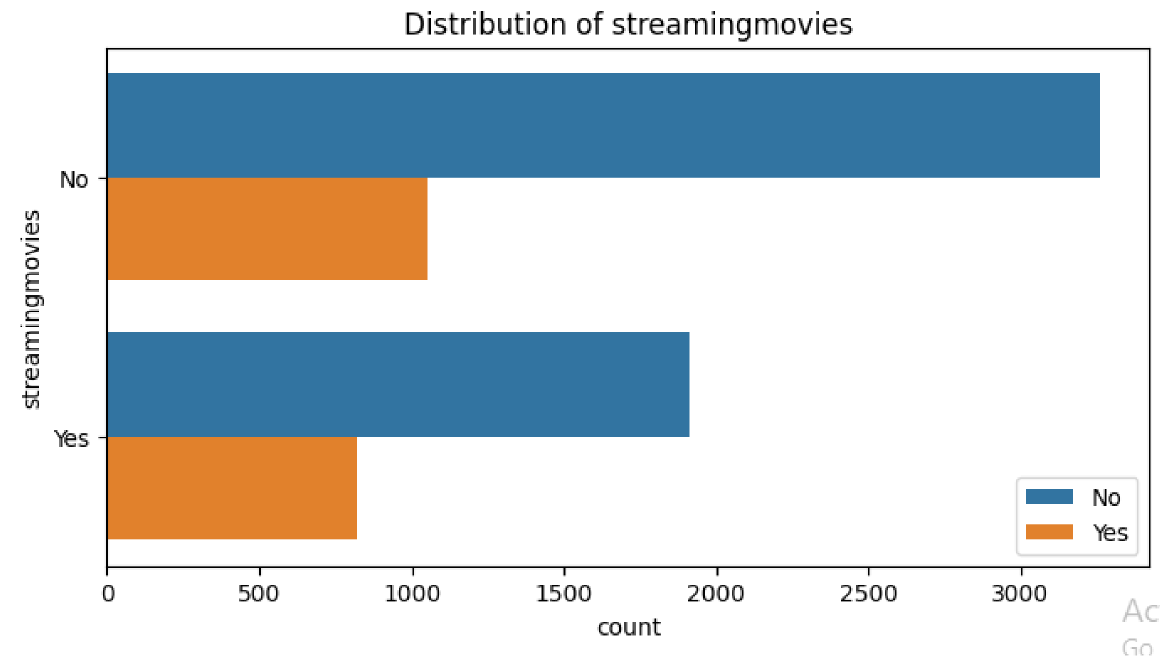

Distribution of onlinesecurity
onlinesecurity
No
Yes
No
Yes
0
500
1000
1500
2000
2500
3000
3500
count

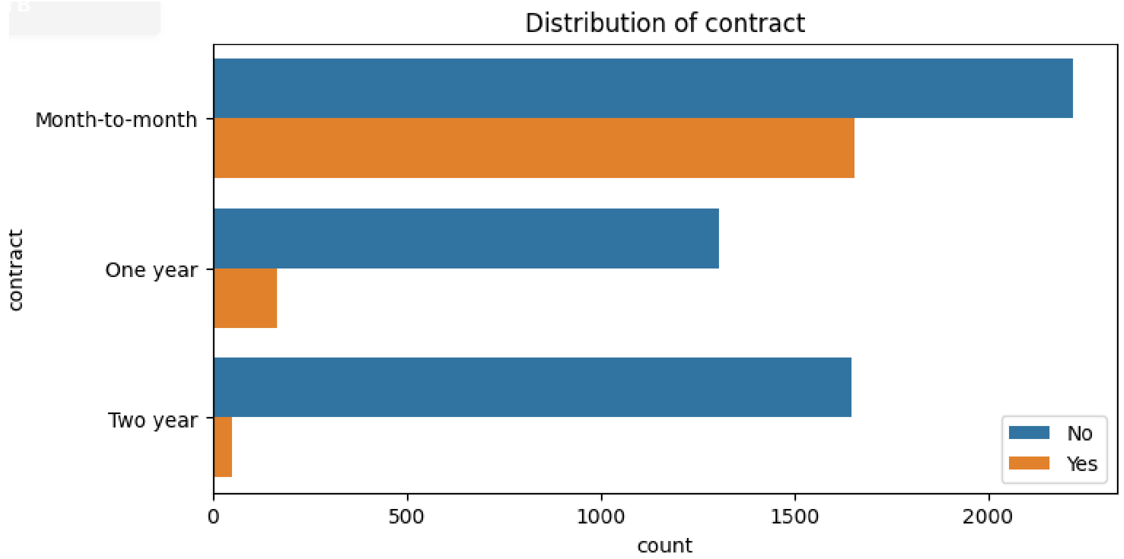

Distribution of onlinebackup
No
Yes
Yes
No
onlinebackup
0
500
1000
1500
2000
2500
3000
count

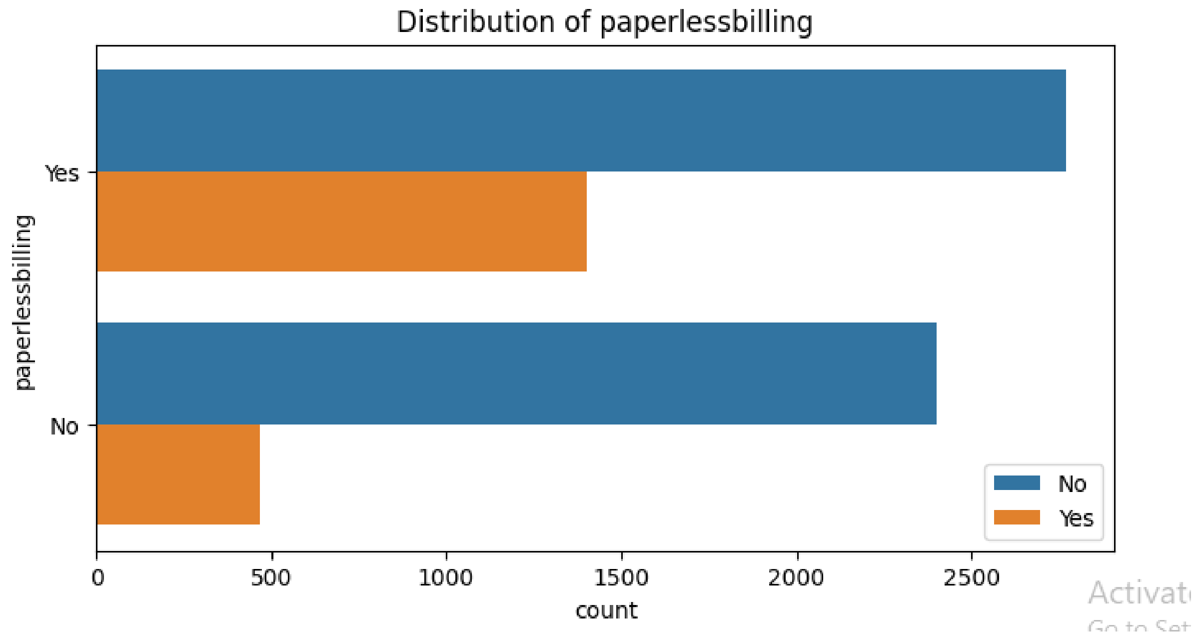

Distribution of deviceprotection
No
Yes
deviceprotection
No
Yes
0
500
1000
1500
2000
2500
3000
count


Distribution of techsupport
techsupport
No
Yes
count
0
500
1000
1500
2000
2500
3000
3500
No
Yes

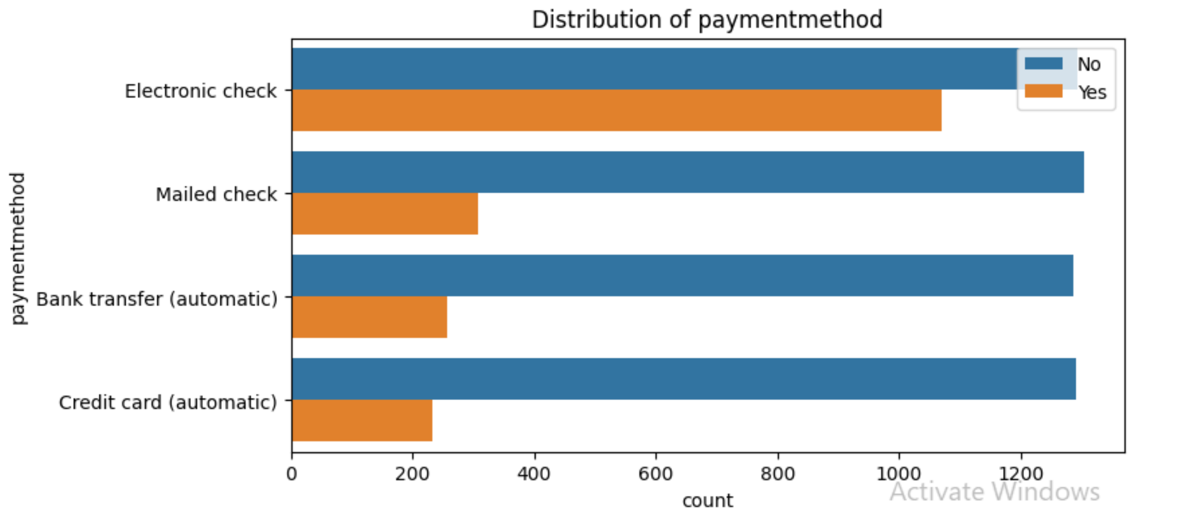

Distribution of streamingtv
streamingtv
No
Yes
count
0
500
1000
1500
2000
2500
3000
No
Yes


Distribution of streamingmovies
streamingmovies
No
Yes
0
500
1000
1500
2000
2500
3000
count
No
Yes

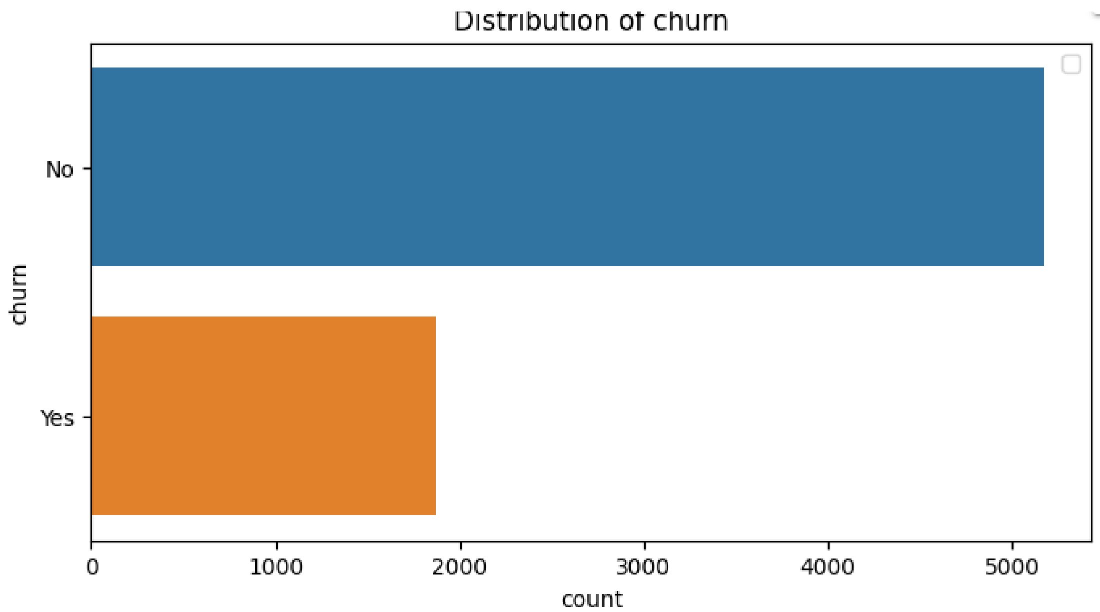

Distribution of contract
contract
Month-to-month
One year
Two year
0
500
1000
1500
2000
count
No
Yes



Distribution of paperlessbilling
paperlessbilling
Yes
No
No
Yes
0
500
1000
1500
2000
2500
count



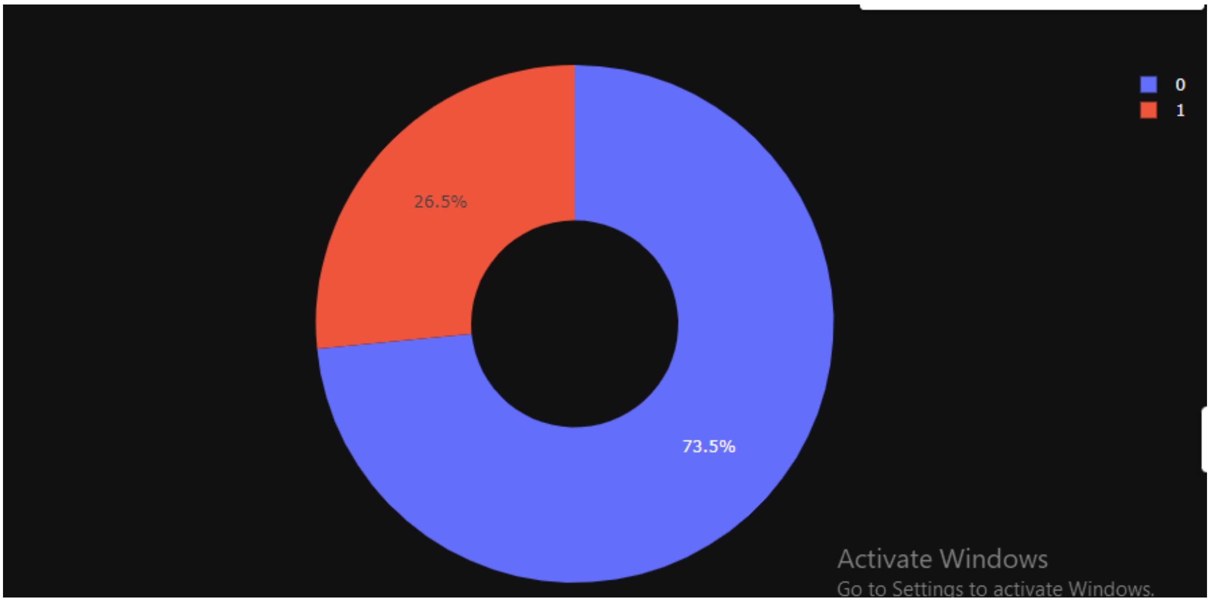
Distribution of paymentmethod
paymentmethod
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
No
Yes
0
200
400
600
800
1000
1200
count



Distribution of churn
churn
No
Yes
0
1000
2000
3000
4000
5000
count

4. Data Balancing

```
import plotly.express as px
fig = px.pie(churn_data,names="churn",hole = 0.4,template = "plotly_dark")
fig.show()
```



- Technique Used: SMOTE (Synthetic Minority Oversampling Technique)

```
x = churn_data.drop("churn",axis=1)
y = churn_data["churn"]
```

```
from imblearn.over_sampling import SMOTE
from collections import Counter
print(Counter(y))
```

```
Counter({0: 5174, 1: 1869})
```

```
smote = SMOTE()
x_resampled, y_resampled = smote.fit_resample(x, y)
```

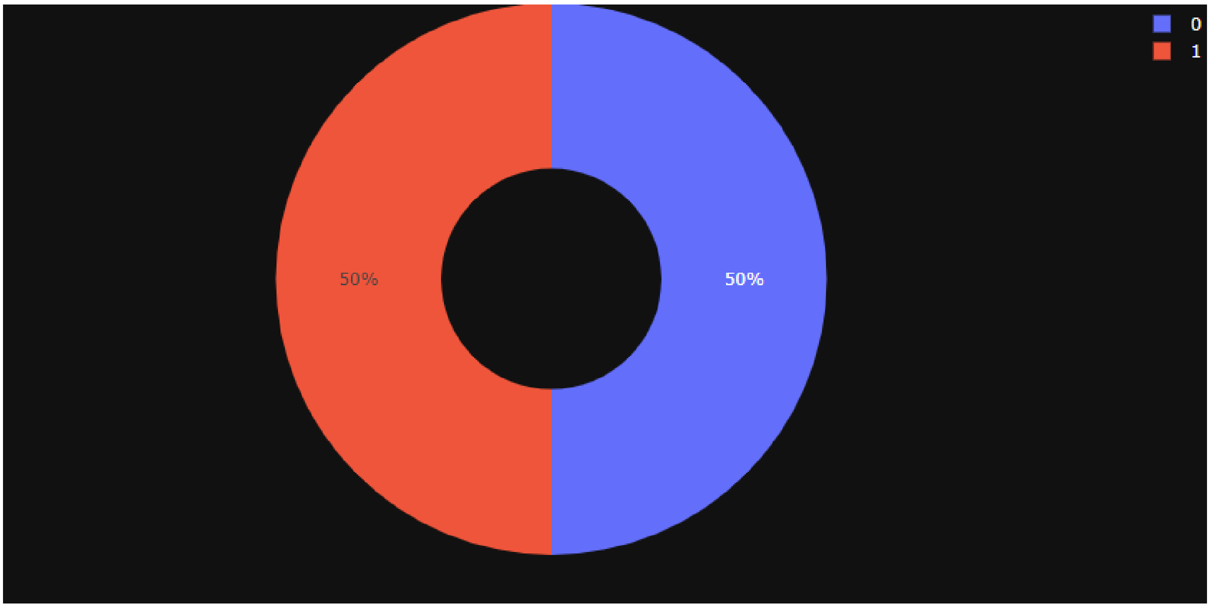
```
x_resampled.shape, y_resampled.shape
```

```
((10348, 19), (10348,))
```

```
print(Counter(y_resampled))
```

```
Counter({0: 5174, 1: 5174})
```

```
data = x_resampled
target = y_resampled
```



- Purpose:Address class imbalance in the churn target variable.

5. Model Training and Evaluation (Without Feature Selection)

5.1 Data Splitting:

- Training and Testing Sets: Description of how the data was split.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,target, test_size=0.2, random_state=42)
```

5.2 Model Training:

- Models Used: Logistic Regression, Decision Tree, Random Forest, KNeighborsClassifier, SVC, XGBoost, GradientBoostingClassifier

LogisticRegression:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

model_lr = LogisticRegression(max_iter=2000)

model_lr.fit(X_train,y_train)

pred_lr = model_lr.predict(X_test)
pred_train=model_lr.predict(X_train)
accuracy_score_train=accuracy_score(y_train,pred_train)
print('training_accuray',accuracy_score_train*100)
accuracy_score_lr = accuracy_score(y_test,pred_lr)
print('testing_accuracy',accuracy_score_lr*100)


sns.heatmap((confusion_matrix(y_test,pred_lr)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');

print("")
print("Classification Report")
print(classification_report(y_test, pred_lr))
```


Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier

for i in range(1,11):
    model_dt = DecisionTreeClassifier(max_depth=i)
    model_dt.fit(X_train,y_train)
    pred_dt = model_dt.predict(X_test)
    accuracy_score_dt = accuracy_score(y_test,pred_dt)
    print(i,accuracy_score_dt)
```

RandomForestClassifier:

```
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier()
```

```
model_rf.fit(X_train,y_train)
pred_rf = model_rf.predict(X_test)
pred_train=model_rf.predict(X_train)
print("Predicted: ",Counter(pred_rf))
print("Actual: ",Counter(y_test))

accuracy_score_train=accuracy_score(y_train,pred_train)
print('training_accuray',accuracy_score_train*100)
accuracy_score_rf = accuracy_score(y_test,pred_rf)
print('testing_accuracy',accuracy_score_rf*100)

sns.heatmap((confusion_matrix(y_test,pred_rf)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');


print(classification_report(y_test, pred_rf))
```

KNN:

```
from sklearn.neighbors import KNeighborsClassifier

model_knn = KNeighborsClassifier()


for i in range(1,30,5):
    model_knn = KNeighborsClassifier(n_neighbors=i)
    model_knn.fit(X_train,y_train)
    pred_knn = model_knn.predict(X_test)
    accuracy_score_knn = accuracy_score(y_test,pred_knn)
    print(i,accuracy_score_knn)
```

Support Vector Classifier:

```
from sklearn.svm import SVC

model_svm = SVC(kernel="rbf")

model_svm.fit(X_train,y_train)

pred_train=model_svm.predict(X_train)
pred_svm = model_svm.predict(X_test)

accuracy_score_train=accuracy_score(y_train,pred_train)
print('training_accuray',accuracy_score_train*100)
accuracy_score_svm = accuracy_score(y_test,pred_svm)
print('testing_accuracy',accuracy_score_svm*100)

sns.heatmap((confusion_matrix(y_test,pred_svm)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');

print(classification_report(y_test, pred_svm))
```

GradientBoostingClassifier:

```
from sklearn.ensemble import GradientBoostingClassifier
model_gbc = GradientBoostingClassifier(n_estimators=100,learning_rate=0.03)

model_gbc.fit(X_train,y_train)
pred_gbc = model_gbc.predict(X_test)
pred_train=model_gbc.predict(X_train)

accuracy_score_train=accuracy_score(y_train,pred_train)
print('training_accuray',accuracy_score_train*100)
accuracy_score_gbc = accuracy_score(y_test,pred_gbc)
print('testing_accuracy',accuracy_score_gbc*100)

sns.heatmap((confusion_matrix(y_test,pred_gbc)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');
```

XGBClassifier:

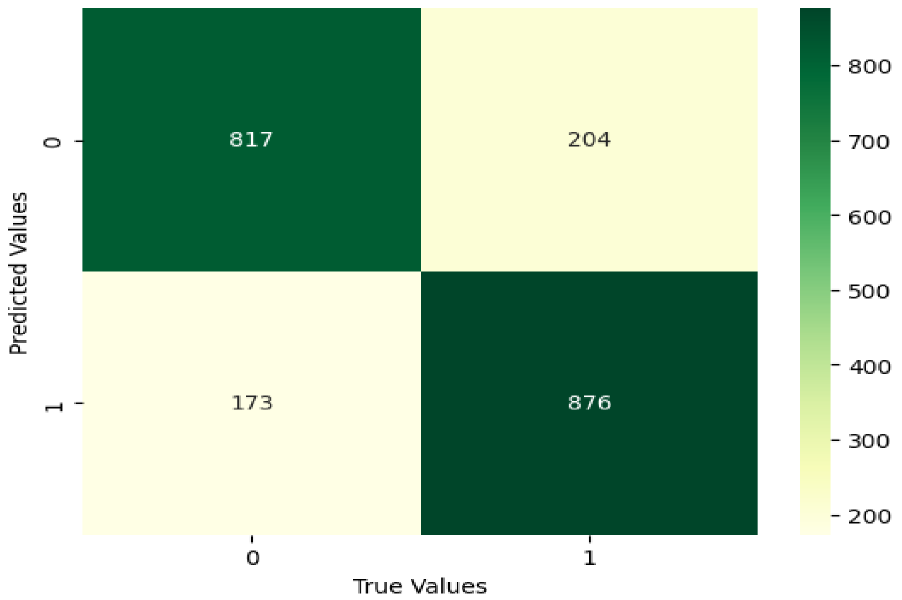
```
from xgboost import XGBClassifier

model_xgb = XGBClassifier(n_estimators=100,learning_rate=0.03)

model_xgb.fit(X_train,y_train)
pred_xgb = model_xgb.predict(X_test)
pred_train=model_xgb.predict(X_train)
accuracy_score_train=accuracy_score(y_train,pred_train)
print('training_accuray',accuracy_score_train*100)
accuracy_score_xgb = accuracy_score(y_test,pred_xgb)
print('testing_accuracy',accuracy_score_xgb*100)


sns.heatmap((confusion_matrix(y_test,pred_xgb)),annot=True,fmt='.5g',cmap="YlGn");
plt.xlabel('True Values')
plt.ylabel('Predicted Values');

print(classification_report(y_test, pred_xgb))
```

5.3 Model Evaluation:

- Metrics: Accuracy score, confusion matrix, classification report.

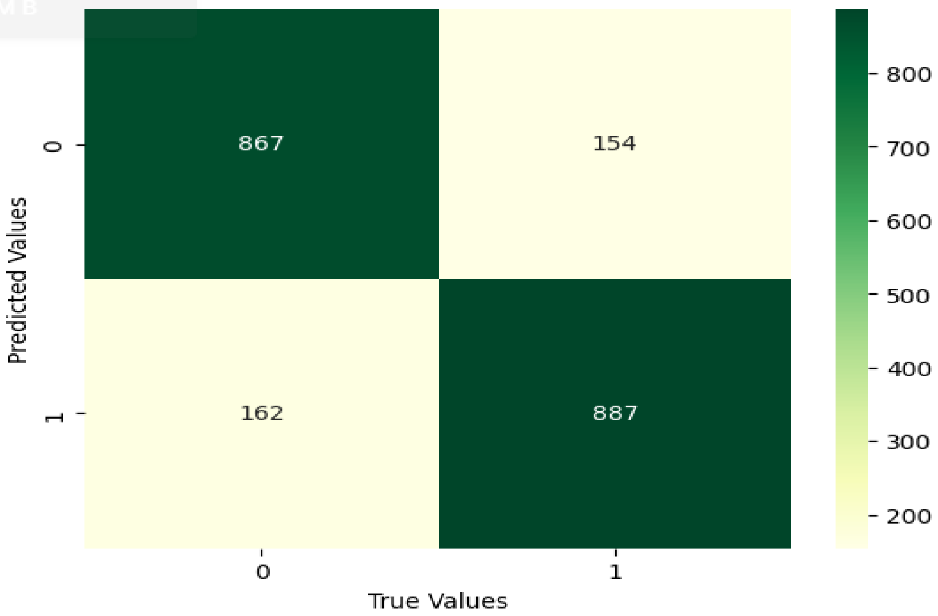
Logistic Regression:

```
training_accuray 81.85552065716357
testing_accuracy 81.78743961352657

Classification Report
              precision    recall  f1-score   support

           0       0.83      0.80      0.81      1021
           1       0.81      0.84      0.82      1049

    accuracy                           0.82      2070
   macro avg       0.82      0.82      0.82      2070
weighted avg       0.82      0.82      0.82      2070
```



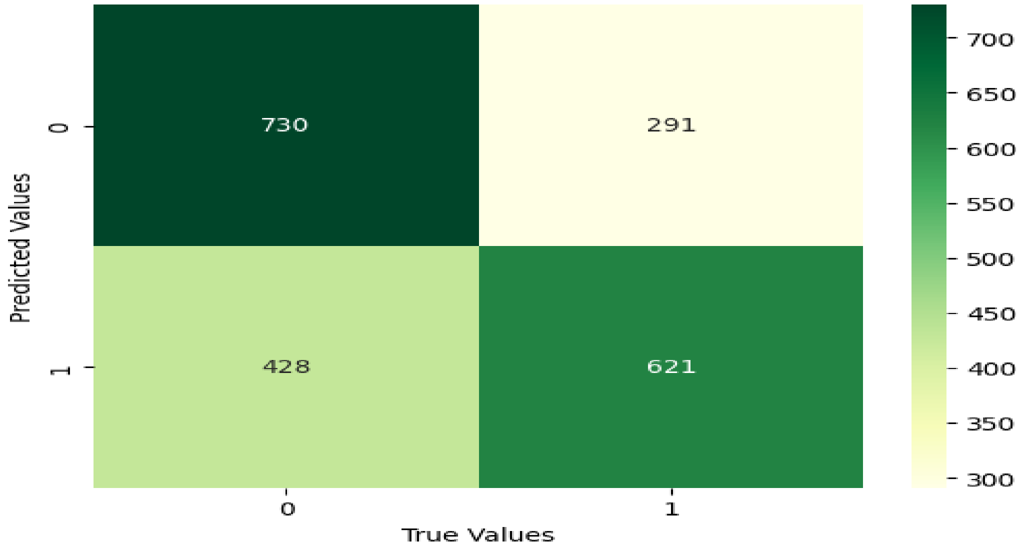
RandomForestClassifier:

```
Predicted:  Counter({1: 1041, 0: 1029})
Actual:  Counter({1: 1049, 0: 1021})
training_accuray 99.86711766127084
testing_accuracy 84.73429951690822
              precision    recall  f1-score   support

           0       0.84      0.85      0.85      1021
           1       0.85      0.85      0.85      1049

    accuracy                           0.85      2070
   macro avg       0.85      0.85      0.85      2070
weighted avg       0.85      0.85      0.85      2070
```



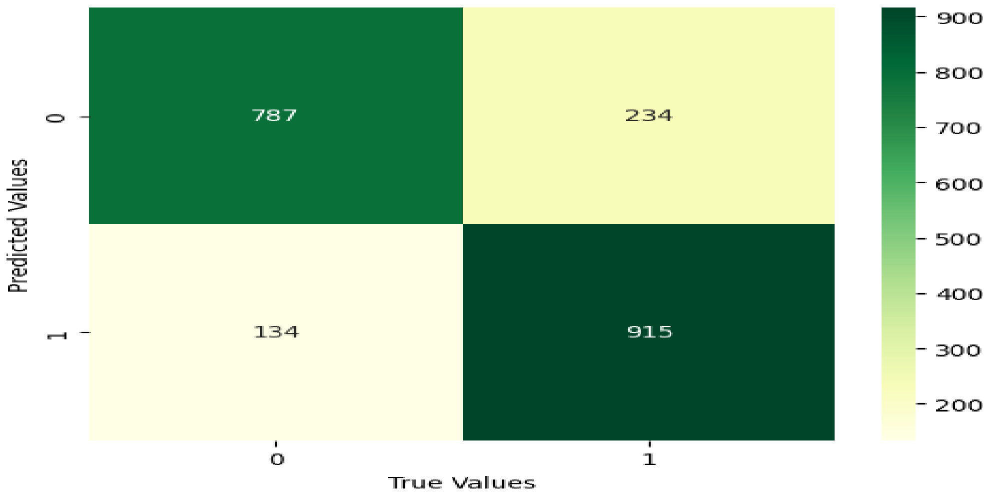
Support Vector Classifier:

```
training_accuray 65.92172022227591
testing_accuracy 65.26570048309178
              precision    recall  f1-score   support

           0       0.63      0.71      0.67      1021
           1       0.68      0.59      0.63      1049

    accuracy                           0.65      2070
   macro avg       0.66      0.65      0.65      2070
weighted avg       0.66      0.65      0.65      2070
```



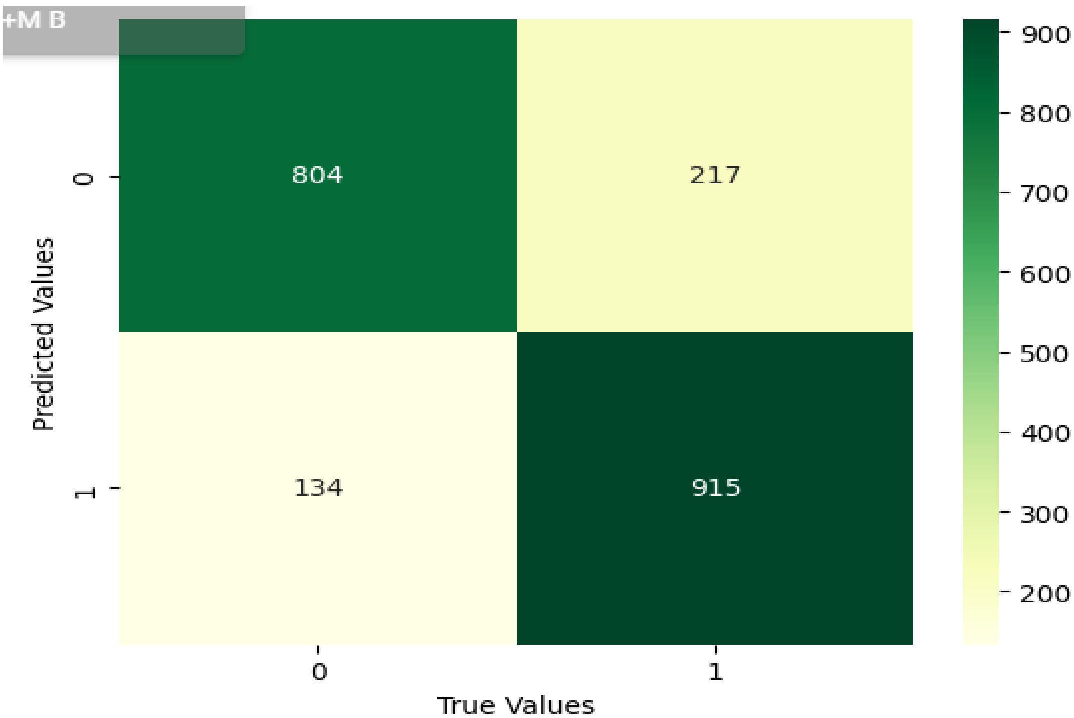
GradientBoostingClassifier:

```
training_accuray 82.08504469678667
testing_accuracy 82.22222222222221
              precision    recall  f1-score   support

           0       0.85      0.77      0.81      1021
           1       0.80      0.87      0.83      1049

    accuracy                           0.82      2070
   macro avg       0.83      0.82      0.82      2070
weighted avg       0.83      0.82      0.82      2070
```



XGBClassifier:

```
training_accuray 85.92655230732062
testing_accuracy 83.04347826086956
              precision    recall  f1-score   support

           0       0.86      0.79      0.82      1021
           1       0.81      0.87      0.84      1049

    accuracy                           0.83      2070
   macro avg       0.83      0.83      0.83      2070
weighted avg       0.83      0.83      0.83      2070
```



```python
models = pd.DataFrame({
    "Model": ["Logistic Regression",
              "Decision Tree",
              "Random Forest",
              "KNN",
              "SVM",
              "XGBoosT",
              "GradientBoosting"],

    "Accuracy Score" : [accuracy_score_lr*100,accuracy_score_dt*100,accuracy_score_rf*100,accuracy_score_knn*100,
                        accuracy_score_svm*100,accuracy_score_xgb*100,accuracy_score_gbc*100]

})
```

```python
print("Model Accuracies without using RFE")
models
```

```
Model Accuracies without using RFE
```

	Model	Accuracy Score
0	Logistic Regression	81.787440
1	Decision Tree	80.966184
2	Random Forest	84.734300
3	KNN	76.183575
4	SVM	65.265700
5	XGBoosT	83.043478
6	GradientBoosting	82.222222

6. Feature Selection using RFE

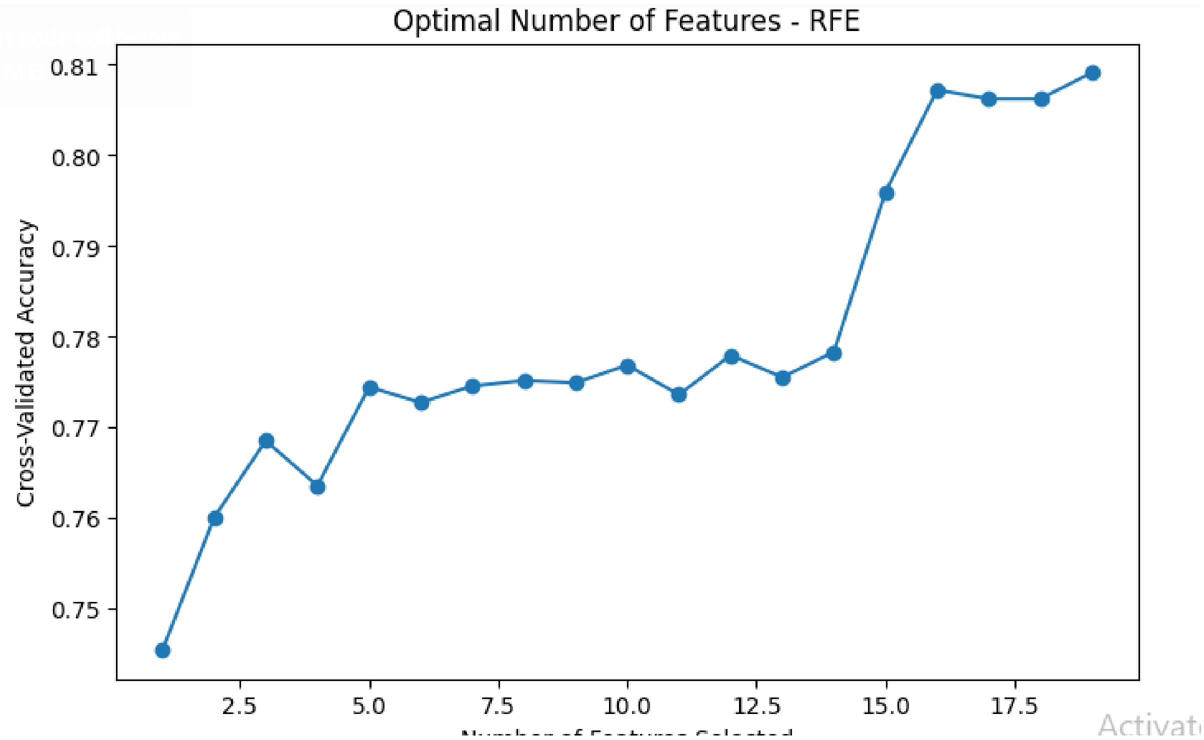
6.1 Recursive Feature Elimination (RFE)

- Model Used: Logistic Regression

```python
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
```

```python
scores = []

# Perform RFE with cross-validation to determine the optimal number of features
for n_features in range(1, X_train.shape[1] + 1):
    rfe = RFE(estimator=model_lr, n_features_to_select=n_features)
    rfe.fit(X_train, y_train)
    score = cross_val_score(rfe, X_train, y_train, cv=5, scoring='accuracy').mean()
    scores.append(score)
```

```
plt.figure(figsize=(10, 6))
plt.plot(range(1, X_train.shape[1] + 1), scores, marker='o')
plt.xlabel('Number of Features Selected')
plt.ylabel('Cross-Validated Accuracy')
plt.title('Optimal Number of Features - RFE')
plt.show()

# Get the optimal number of features
optimal_n_features = np.argmax(scores) + 1
print(f'Optimal number of features: {optimal_n_features}')
```



- Objective: Identify optimal number of features

```
lg=LogisticRegression(max_iter=2000)
lg.fit(X_train,y_train)
rfe=RFE(estimator=lg,n_features_to_select=16)
rfe=rfe.fit(X_train,y_train)
```

```
selected_features = X_train.columns[rfe.support_]
print("Selected features:", selected_features)
```

```
Selected features: Index(['gender', 'seniorcitizen', 'partner', 'dependents', 'phoneservice',
       'multiplelines', 'internetservice', 'onlinesecurity', 'onlinebackup',
       'deviceprotection', 'techsupport', 'streamingtv', 'streamingmovies',
       'contract', 'paperlessbilling', 'monthlycharges'],
      dtype='object')
```

```python
import statsmodels.api as sm
X_train_sm=sm.add_constant(X_train_rfe)
logistic_model=sm.Logit(y_train,X_train_rfe).fit()
logistic_model.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.417370
         Iterations 7
```

Logit Regression Results

Dep. Variable:	churn	**No. Observations:**	8278
Model:	Logit	**Df Residuals:**	8262
Method:	MLE	**Df Model:**	15
Date:	Mon, 10 Jun 2024	**Pseudo R-squ.:**	0.3979
Time:	13:30:15	**Log-Likelihood:**	-3455.0
converged:	True	**LL-Null:**	-5737.8
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
gender	-0.4234	0.059	-7.224	0.000	-0.538	-0.308
seniorcitizen	-0.4733	0.085	-5.547	0.000	-0.641	-0.306
partner	-0.4832	0.073	-6.661	0.000	-0.625	-0.341
dependents	-0.5636	0.089	-6.337	0.000	-0.738	-0.389
phoneservice	-1.5468	0.116	-13.355	0.000	-1.774	-1.320
multiplelines	-0.5331	0.074	-7.191	0.000	-0.678	-0.388
internetservice	-0.0500	0.054	-0.930	0.352	-0.155	0.055
onlinesecurity	-1.2524	0.084	-14.867	0.000	-1.417	-1.087
onlinebackup	-0.9323	0.074	-12.595	0.000	-1.077	-0.787
deviceprotection	-0.6995	0.078	-8.992	0.000	-0.852	-0.547
techsupport	-1.0538	0.085	-12.404	0.000	-1.220	-0.887
streamingtv	-0.5319	0.083	-6.418	0.000	-0.694	-0.369
streamingmovies	-0.4982	0.083	-6.020	0.000	-0.660	-0.336
contract	-1.1368	0.064	-17.855	0.000	-1.262	-1.012
paperlessbilling	-0.1264	0.065	-1.931	0.053	-0.255	0.002
monthlycharges	0.0573	0.002	29.931	0.000	0.054	0.061

```python
X_train_rfe=X_train_rfe.drop(['paperlessbilling','internetservice'],axis=1)
```

```python
X_test_rfe=X_test[X_train_rfe.columns]
```

6.2 Feature Selection

- Best Features: List of selected features based on RFE analysis.

	coef	std err	z	P>|z|	[0.025	0.975]
gender	-0.4339	0.058	-7.435	0.000	-0.548	-0.320
seniorcitizen	-0.4914	0.085	-5.780	0.000	-0.658	-0.325
partner	-0.4892	0.072	-6.756	0.000	-0.631	-0.347
dependents	-0.5655	0.089	-6.364	0.000	-0.740	-0.391
phoneservice	-1.6123	0.096	-16.870	0.000	-1.800	-1.425
multiplelines	-0.5338	0.074	-7.230	0.000	-0.678	-0.389
onlinesecurity	-1.2360	0.084	-14.749	0.000	-1.400	-1.072
onlinebackup	-0.9326	0.074	-12.578	0.000	-1.078	-0.787
deviceprotection	-0.6950	0.078	-8.925	0.000	-0.848	-0.542
techsupport	-1.0451	0.085	-12.360	0.000	-1.211	-0.879
streamingtv	-0.5398	0.083	-6.527	0.000	-0.702	-0.378
streamingmovies	-0.4987	0.083	-6.028	0.000	-0.661	-0.337
contract	-1.1398	0.063	-18.131	0.000	-1.263	-1.017
monthlycharges	0.0566	0.002	31.557	0.000	0.053	0.060

7. Model Training and Evaluation (With Feature Selection)

7.1 Retraining:

- Model Used: Logistic Regression with selected features.

```
logreg = LogisticRegression()

# Train the classifier
logreg.fit(X_train_rfe, y_train)
```

7.2 Model Evaluation:

- Metrics: Accuracy score, precision, recall, F1 score, confusion matrix, classification report.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_rep
y_pred_logreg = logreg.predict(X_test_rfe)
accuracy_logreg = accuracy_score(y_test, y_pred_logreg) *100
conf_matrix_logreg = confusion_matrix(y_test, y_pred_logreg)
class_report_logreg = classification_report(y_test, y_pred_logreg)
precision_logreg = precision_score(y_test, y_pred_logreg)*100
recall_logreg = recall_score(y_test, y_pred_logreg)*100
f1_logreg = f1_score(y_test, y_pred_logreg)*100
```

```
print("Logistic Regression Classifier Metrics:")
print("Accuracy:", accuracy_logreg)
print("Precision:", precision_logreg)
print("Recall:", recall_logreg)
print("F1 Score:", f1_logreg)
```

```
Logistic Regression Classifier Metrics:
Accuracy: 82.41545893719807
Precision: 80.82808280828083
Recall: 85.60533841754051
F1 Score: 83.14814814814815
```

8. Hyperparameter Tuning using GridSearchCV

8.1 Pipelines and Parameter Grids:

- Models Included:Logistic Regression, Random Forest, Gradient Boosting, Decision Tree

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
```

Pipelines:

```
# Logistic Regression Pipeline
pipeline_lr = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(max_iter=2000))
])

# Random Forest Pipeline
pipeline_rf = Pipeline([
    ('classifier', RandomForestClassifier())
])

# Gradient Boosting Pipeline
pipeline_gb = Pipeline([
    ('classifier', GradientBoostingClassifier())
])

# Decision Tree Pipeline
pipeline_dt = Pipeline([
    ('classifier', DecisionTreeClassifier())
])
```

```
# Define parameter grids for each model
param_grid_lr = {
    'classifier__solver': ['liblinear', 'saga'],
    'classifier__C': [0.1, 1, 10],
    'classifier__penalty': ['l2']
}

param_grid_rf = {
    'classifier__n_estimators': [10,100,200,300],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5],
    'classifier__min_samples_leaf': [1, 2]
}

param_grid_gb = {

    'classifier__n_estimators': [10,100,200,300],
    'classifier__learning_rate': [0.1, 0.01],
    'classifier__max_depth':[1,2,3,4,6,8]
}

param_grid_dt = {
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5],
    'classifier__min_samples_leaf': [1, 2]
}
```

```
pipelines = {
    'Logistic Regression': (pipeline_lr, param_grid_lr),
    'Random Forest': (pipeline_rf, param_grid_rf),
    'Gradient Boosting': (pipeline_gb, param_grid_gb),
    'Decision Tree': (pipeline_dt, param_grid_dt)
}
```

- Objective: Tune hyperparameters to maximize accuracy.

8.2 GridSearchCV:

- Process: Description of GridSearchCV process.

```
best_estimators = {}
for name, (pipeline, param_grid) in pipelines.items():
    grid_search = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1, scoring='accuracy')
    grid_search.fit(X_train_rfe, y_train)
    best_estimators[name] = grid_search.best_estimator_
    print(f"Best parameters for {name}: {grid_search.best_params_}")
    print(f"Best cross-validation accuracy for {name}: {grid_search.best_score_}")
```

- Best Hyperparameters: Summary of the best hyperparameter combinations found.

9. Final Model Evaluation

_9.1 Evaluation Metrics:

- Metrics: Accuracy score, confusion matrix, classification report.

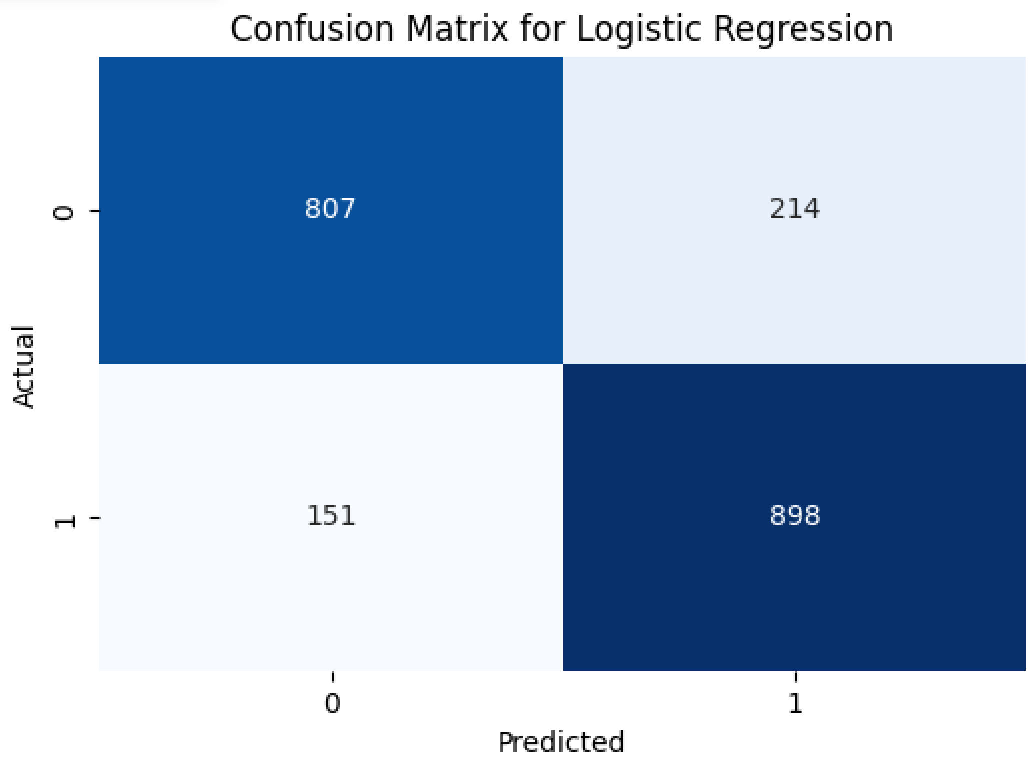
```
Logistic Regression
Train_Accuracy: 0.8061125875815415
Test Accuracy: 0.8236714975845411
Confusion Matrix:
[[807 214]
 [151 898]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.79      0.82      1021
           1       0.81      0.86      0.83      1049

    accuracy                           0.82      2070
   macro avg       0.82      0.82      0.82      2070
weighted avg       0.82      0.82      0.82      2070
```

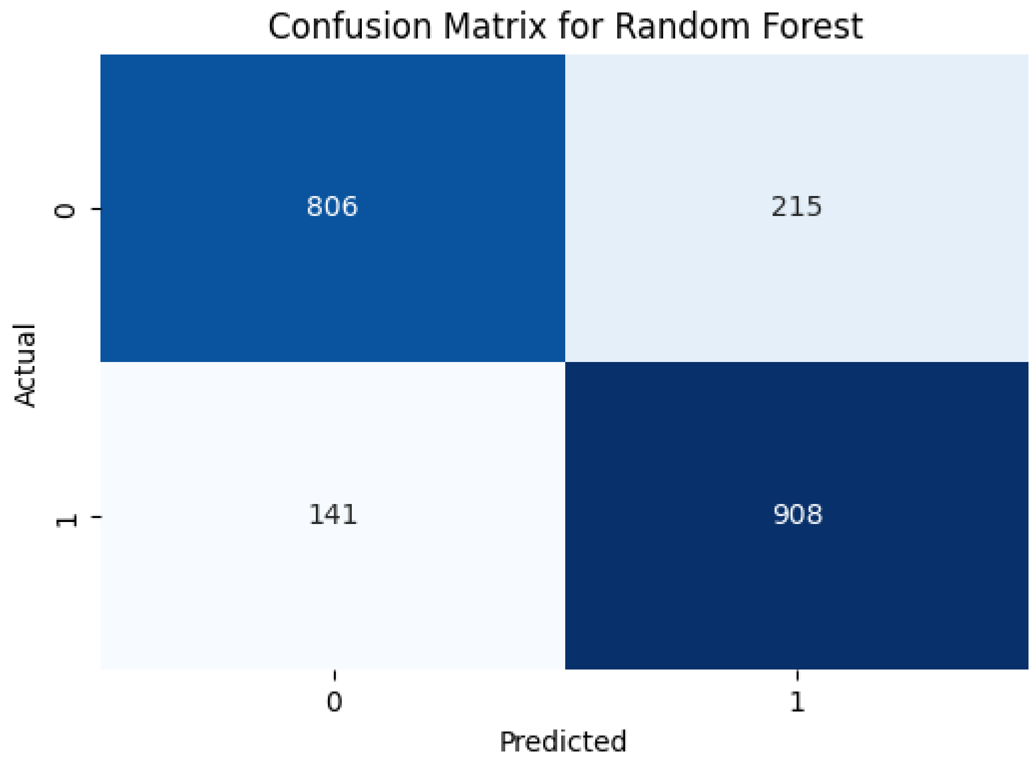
```
Random Forest
Train_Accuracy: 0.858661512442619
Test Accuracy: 0.8280193236714976
Confusion Matrix:
[[806 215]
 [141 908]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82      1021
           1       0.81      0.87      0.84      1049

    accuracy                           0.83      2070
   macro avg       0.83      0.83      0.83      2070
weighted avg       0.83      0.83      0.83      2070
```

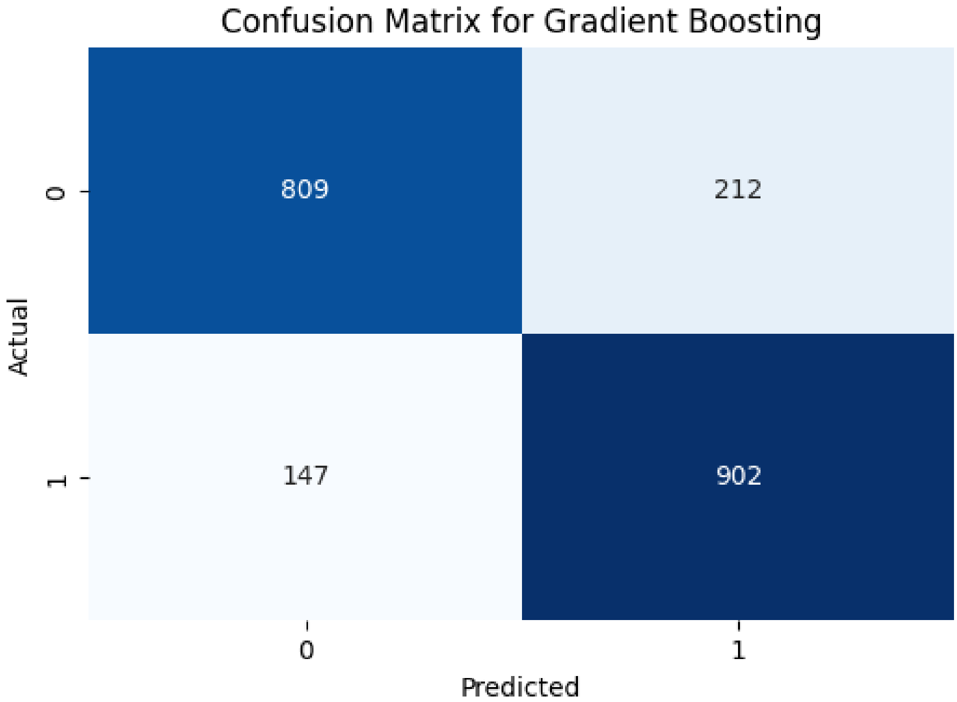
```
Gradient Boosting
Train_Accuracy: 0.8235080937424498
Test Accuracy: 0.8265700483091788
Confusion Matrix:
[[809 212]
 [147 902]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82      1021
           1       0.81      0.86      0.83      1049

    accuracy                           0.83      2070
   macro avg       0.83      0.83      0.83      2070
weighted avg       0.83      0.83      0.83      2070


Decision Tree
Train_Accuracy: 0.8518965933800435
Test Accuracy: 0.8028985507246377
Confusion Matrix:
[[776 245]
 [163 886]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.76      0.79      1021
           1       0.78      0.84      0.81      1049

    accuracy                           0.80      2070
   macro avg       0.80      0.80      0.80      2070
weighted avg       0.80      0.80      0.80      2070
```

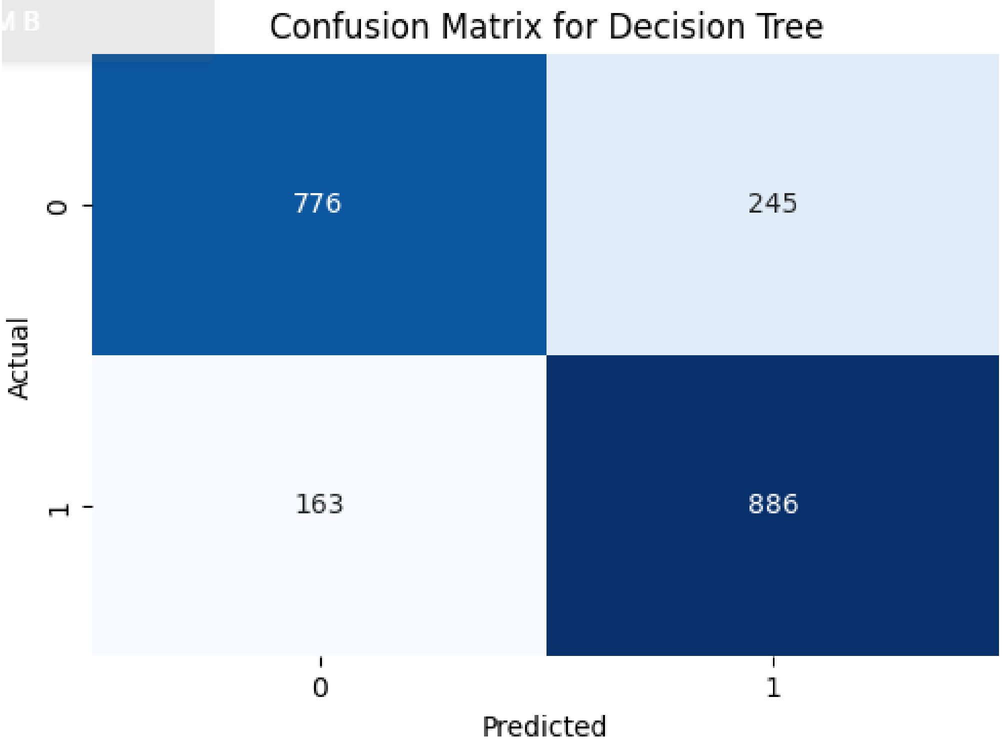
9.2 Confusion Matrix Visualization:

- Purpose:Understand model performance on different classes.






Confusion Matrix for Gradient Boosting
809
212
147
902
Actual
0
1
Predicted
0
1


Confusion Matrix for Decision Tree
776
245
163
886
Actual
0
1
Predicted
0
1

10. Conclusion

Summary: Comprehensive comparison of models with and without feature selection and hyperparameter tuning.

without feature selection:

```
Logistic Regression
Train_Accuracy: 0.8136023194008215
Test Accuracy: 0.8251207729468599
Confusion Matrix:
[[809 212]
 [150 899]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.79      0.82      1021
           1       0.81      0.86      0.83      1049

    accuracy                           0.83      2070
   macro avg       0.83      0.82      0.82      2070
weighted avg       0.83      0.83      0.82      2070
```

```
Random Forest
Train_Accuracy: 0.998550374486591
Test Accuracy: 0.8492753623188406
Confusion Matrix:
[[853 168]
 [144 905]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.84      0.85      1021
           1       0.84      0.86      0.85      1049

    accuracy                           0.85      2070
   macro avg       0.85      0.85      0.85      2070
weighted avg       0.85      0.85      0.85      2070
```

```
Gradient Boosting
Train_Accuracy: 0.9981879681082387
Test Accuracy: 0.8545893719806763
Confusion Matrix:
[[874 147]
 [154 895]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.86      0.85      1021
           1       0.86      0.85      0.86      1049

    accuracy                           0.85      2070
   macro avg       0.85      0.85      0.85      2070
weighted avg       0.85      0.85      0.85      2070
```

```
Decision Tree
Train_Accuracy: 0.8706209229282436
Test Accuracy: 0.8193236714975846
Confusion Matrix:
[[780 241]
 [133 916]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.76      0.81      1021
           1       0.79      0.87      0.83      1049

    accuracy                           0.82      2070
   macro avg       0.82      0.82      0.82      2070
weighted avg       0.82      0.82      0.82      2070
```

Without feature selection Gradient Boosting is better than the other algorithms. But is Overfitting .

With feature selection:

```
Logistic Regression
Train_Accuracy: 0.8061125875815415
Test Accuracy: 0.8236714975845411
Confusion Matrix:
[[807 214]
 [151 898]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.79      0.82      1021
           1       0.81      0.86      0.83      1049

    accuracy                           0.82      2070
   macro avg       0.82      0.82      0.82      2070
weighted avg       0.82      0.82      0.82      2070
```

```
Random Forest
Train_Accuracy: 0.858661512442619
Test Accuracy: 0.8280193236714976
Confusion Matrix:
[[806 215]
 [141 908]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82      1021
           1       0.81      0.87      0.84      1049

    accuracy                           0.83      2070
   macro avg       0.83      0.83      0.83      2070
weighted avg       0.83      0.83      0.83      2070
```

```
Gradient Boosting
Train_Accuracy: 0.8235080937424498
Test Accuracy: 0.8265700483091788
Confusion Matrix:
[[809 212]
 [147 902]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82      1021
           1       0.81      0.86      0.83      1049

    accuracy                           0.83      2070
   macro avg       0.83      0.83      0.83      2070
weighted avg       0.83      0.83      0.83      2070
```

```
Decision Tree
Train_Accuracy: 0.8518965933800435
Test Accuracy: 0.8028985507246377
Confusion Matrix:
[[776 245]
 [163 886]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.76      0.79      1021
           1       0.78      0.84      0.81      1049

    accuracy                           0.80      2070
   macro avg       0.80      0.80      0.80      2070
weighted avg       0.80      0.80      0.80      2070
```

Without feature selection Gradient Boosting and Random Forest is better than the other algorithms. But is Random Forest is less Overfitting when compare with without feature selection .The best model is Gradient Boosting because is not overfitted.

- Best Performing Model:Not explicitly mentioned, but comparative analysis provided.