Churn Prediction System

The goal of this project is to predict customer churn using machine learning techniques and help businesses reduce customer attrition by identifying high-risk segments in advance.
We built, tuned, and evaluated multiple classification models on real-world customer data (from the banking/telecom sector), then visualized the results using charts and dashboards.

Dataset:Bank Customer Churn Dataset(Kaggle)

- Records: ~10,000
- Features:
- Customer tenure
- Credit Score
- Geography
- Gender
- Age
- Balance
- Target: Exited (1 if customer left, 0 otherwise)

Tools Used

- **Python** – Data preprocessing and modeling
- **Pandas, NumPy** – Data handling
- **Scikit-learn, XGBoost, LightGBM** – ML models
- **Matplotlib, Seaborn** – Visualizations
- **GitHub + Streamlit Cloud** – Deployment

Data loading

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

```
[ ] df=pd.read_csv('Churn_Modelling.csv')
```

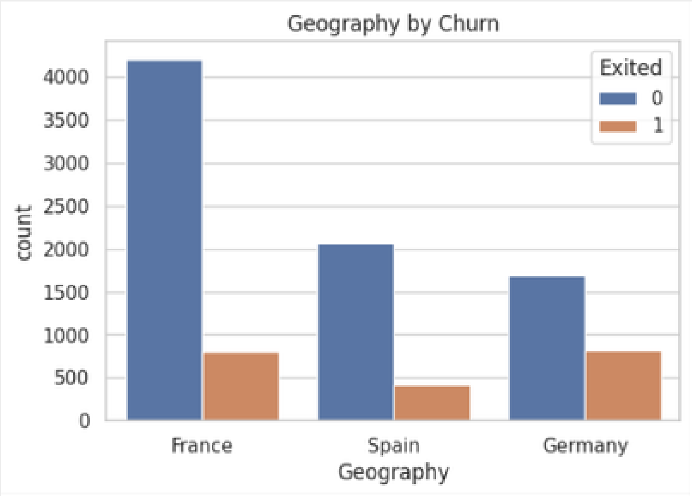
	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

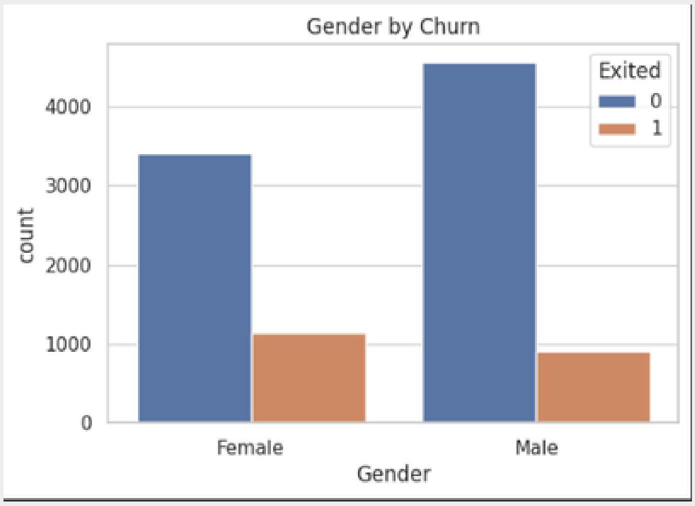
Exploratory Data Analysis(EDA)

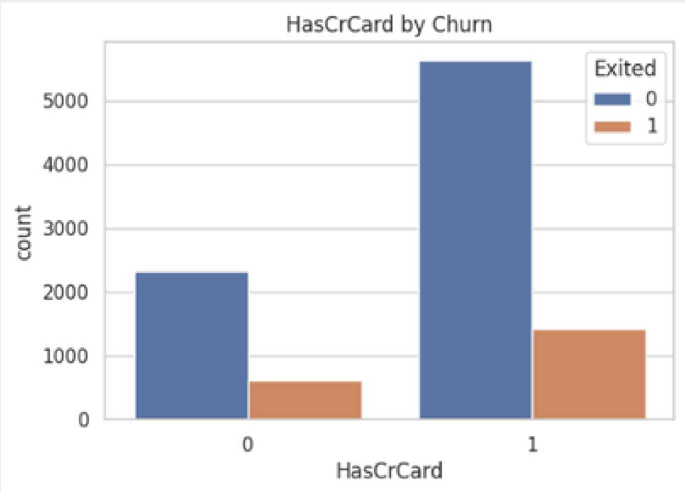
```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

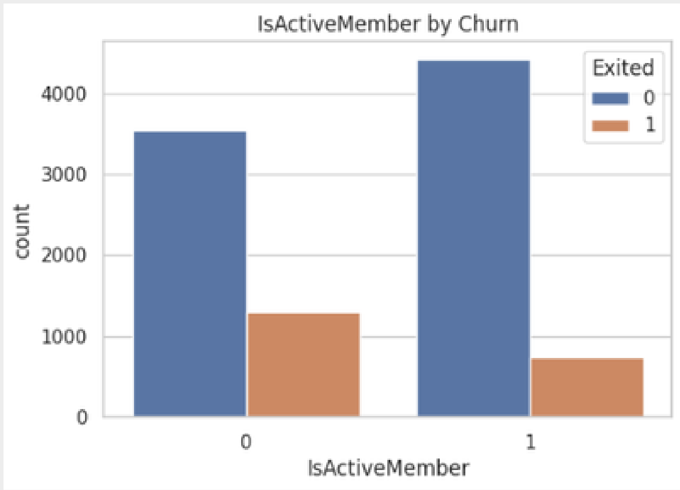
```
[ ] df.describe()
```

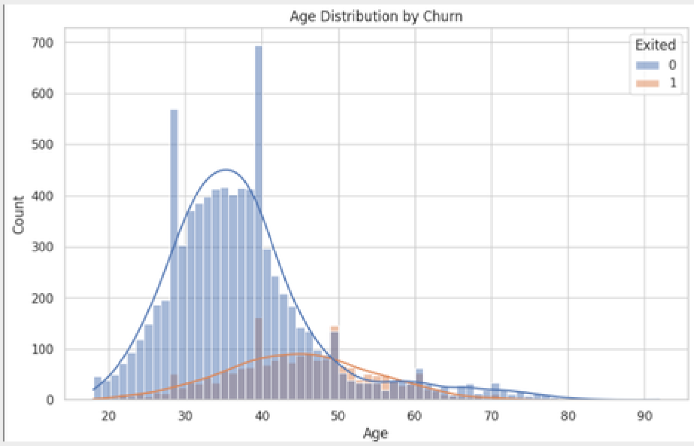
	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

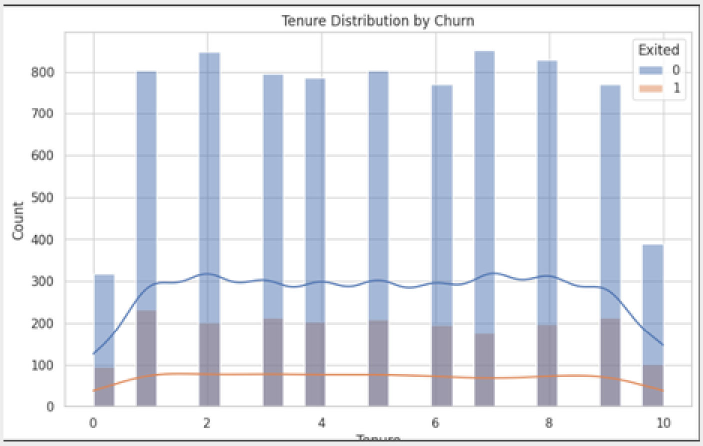

Geography by Churn
count
4000
3500
3000
2500
2000
1500
1000
500
0
France
Spain
Germany
Geography
Exited
0
1

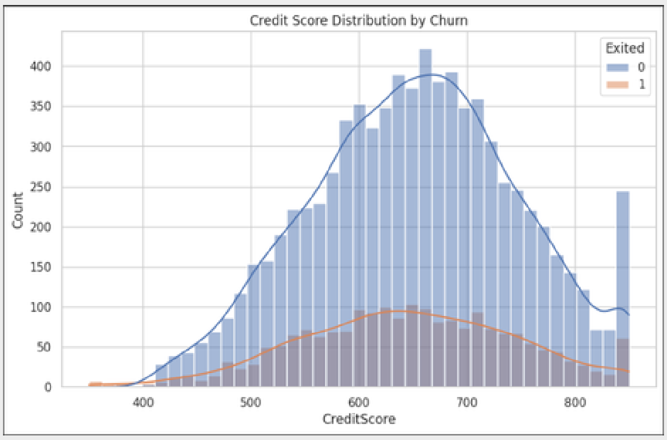

Gender by Churn
count
4000
3000
2000
1000
0
Female
Male
Gender
Exited
0
1

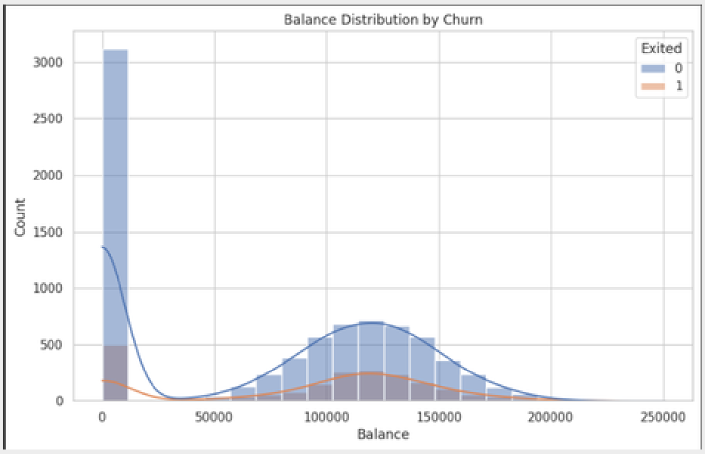

HasCrCard by Churn
count
5000
4000
3000
2000
1000
0
0
1
HasCrCard
Exited
0
1

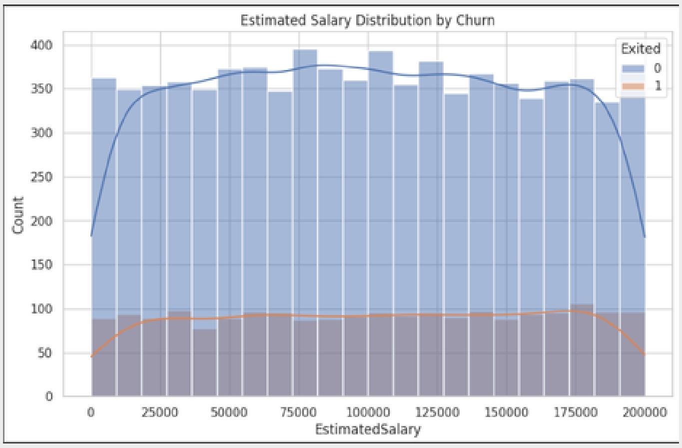

IsActiveMember by Churn
count
4000
3000
2000
1000
0
0
1
IsActiveMember
Exited
0
1

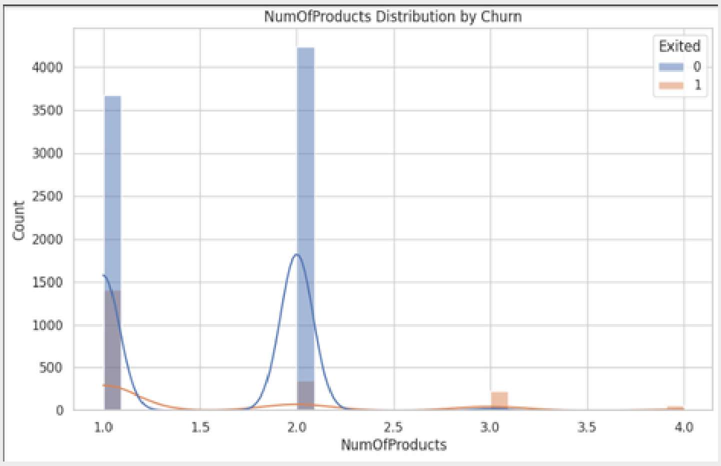

Age Distribution by Churn
Count
700
600
500
400
300
200
100
0
20
30
40
50
60
70
80
90
Age
Exited
0
1


Tenure Distribution by Churn
Count
800
700
600
500
400
300
200
100
0
0
2
4
6
8
10
Exited
0
1


Credit Score Distribution by Churn
Count
400
350
300
250
200
150
100
50
0
400
500
600
700
800
CreditScore
Exited
0
1





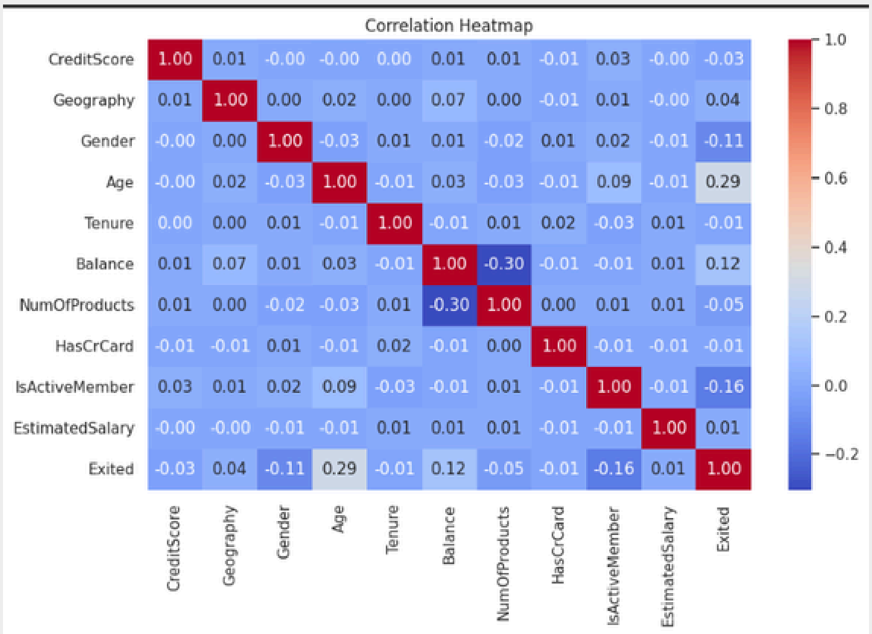


Handling categorical features

```
[ ]  from sklearn.preprocessing import LabelEncoder
     le=LabelEncoder()
     df['Geography']=le.fit_transform(df['Geography'])
     df['Gender']=le.fit_transform(df['Gender'])
```

Handling Multi-Collinearity

Observation: No features are highly correlated

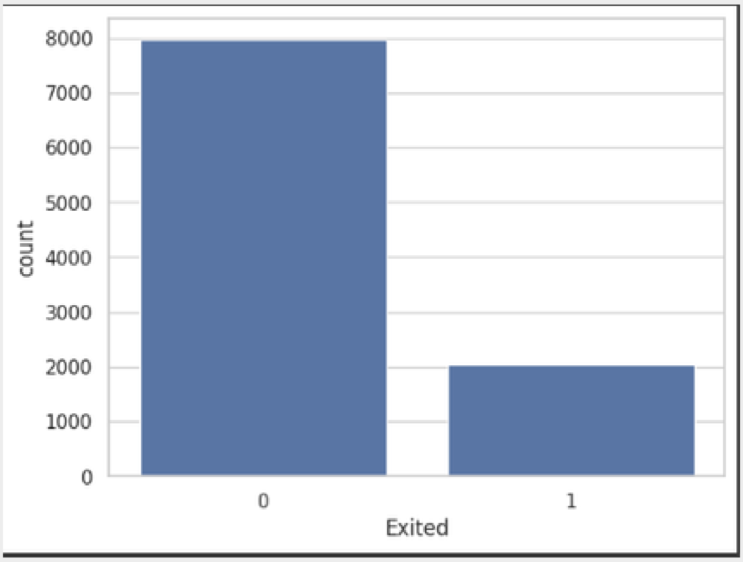


Handling imbalance dataset

```
df['Exited'].value_counts()
```

Exited	count
0	7963
1	2037

dtype: int64



Over-Sampling using SMOTE

```
from imblearn.over_sampling import SMOTE
smote=SMOTE(sampling_strategy='minority',random_state=42)
X=df.drop(columns=['Exited'])
y=df['Exited']
X_res,y_res=smote.fit_resample(X,y)
```

Splitting the data

Train-Test split

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X_res,y_res,test_size=0.2,random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train_sc=scaler.fit_transform(X_train)
X_test_sc=scaler.transform(X_test)
```

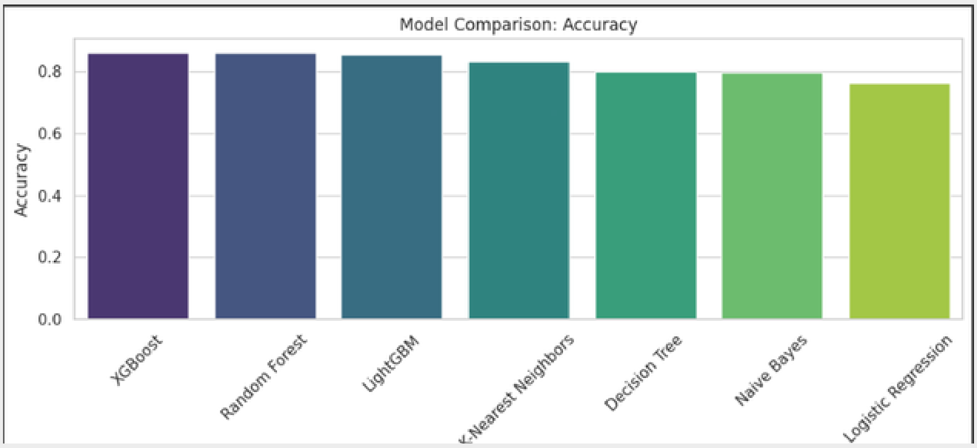
Training Classification Models

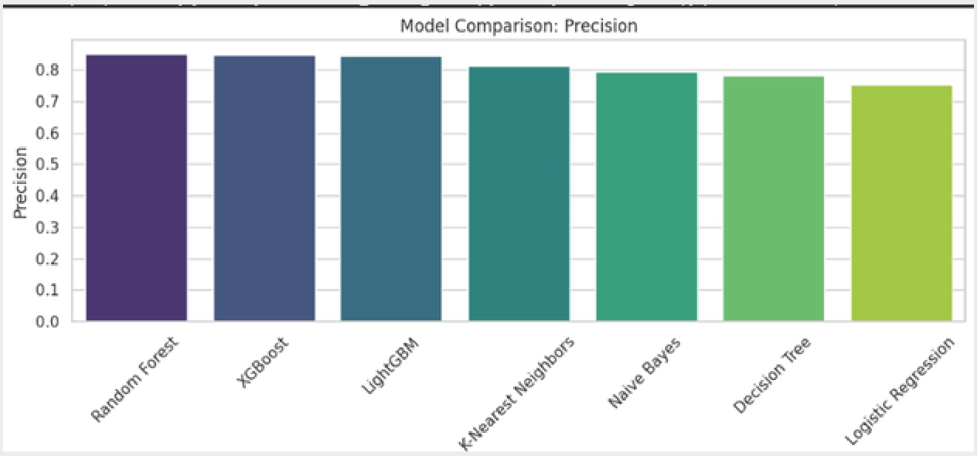
We trained and evaluated the following classification models:

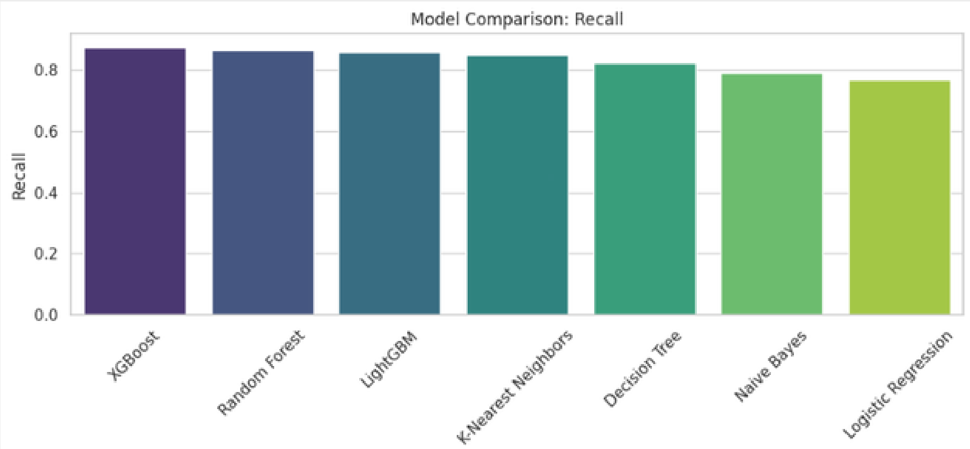
- Logistic Regression
- Decision Tree
- K-Nearest Neighbors (KNN)
- Naive Bayes
- Random Forest
- XGBoost
- LightGBM

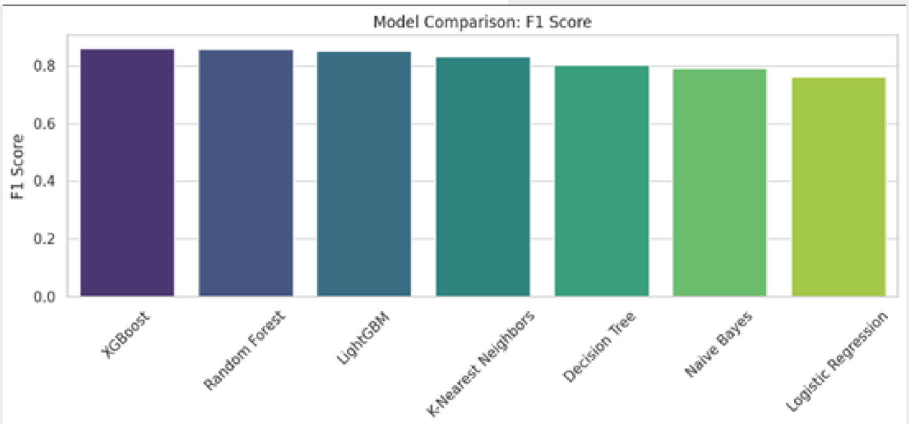
```python
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "Random Forest": RandomForestClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
    "LightGBM": lgb.LGBMClassifier()
}

results = []
confusion_matrices={}
classification_reports={}

for name, model in models.items():
    # Scale data only for models that require it
    if name in ["Logistic Regression", "K-Nearest Neighbors", "Naive Bayes"]:
        model.fit(X_train_sc, y_train)
        y_pred = model.predict(X_test_sc)
        y_proba = model.predict_proba(X_test_sc)[:, 1]
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_proba = model.predict_proba(X_test)[:, 1]

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc = roc_auc_score(y_test, y_proba)
    cm = confusion_matrix(y_test, y_pred)
    cr=classification_report(y_test,y_pred)

    results.append({
        "Model": name,
        "Accuracy": round(acc, 4),
        "Precision": round(prec, 4),
        "Recall": round(rec, 4),
        "F1 Score": round(f1, 4),
        "ROC-AUC": round(roc, 4)
    })

    confusion_matrices[name] = cm
    classification_reports[name]=cr

# Convert results to DataFrame
results_df = pd.DataFrame(results)
```

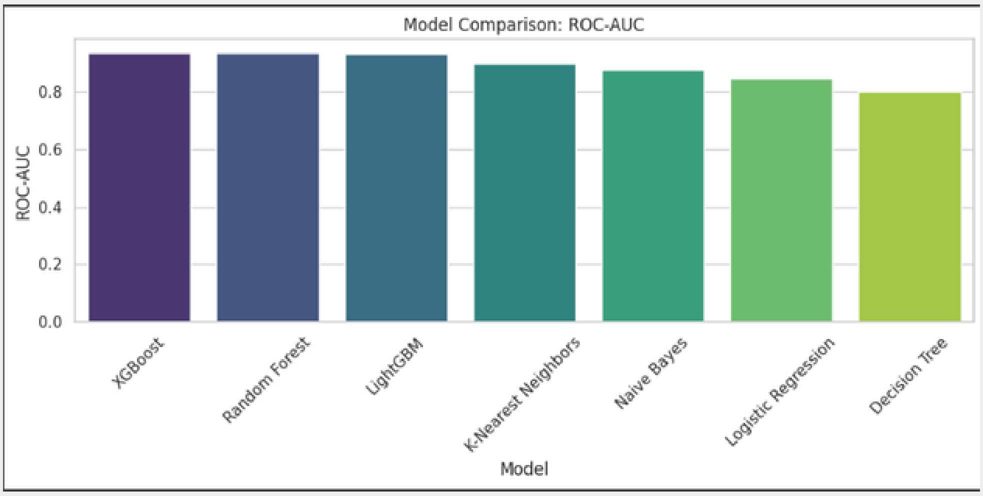
Comparing Metrics



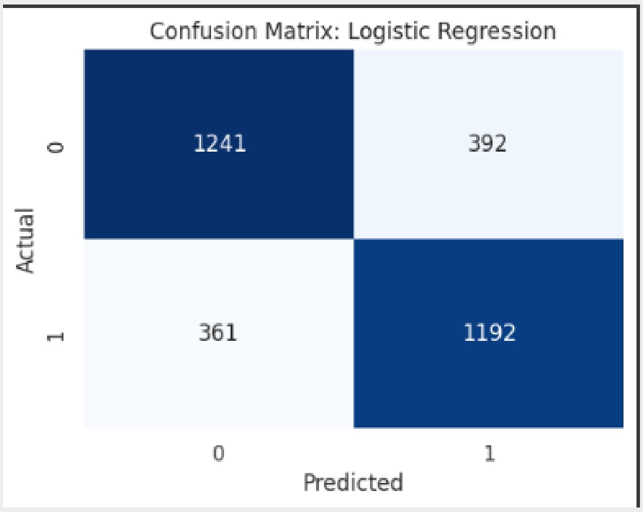


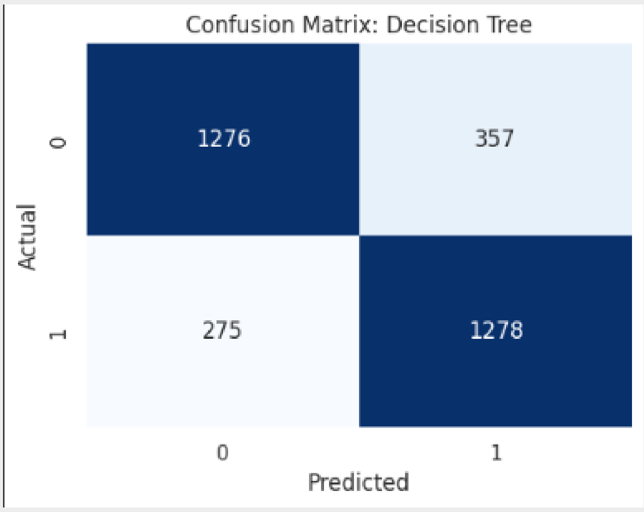


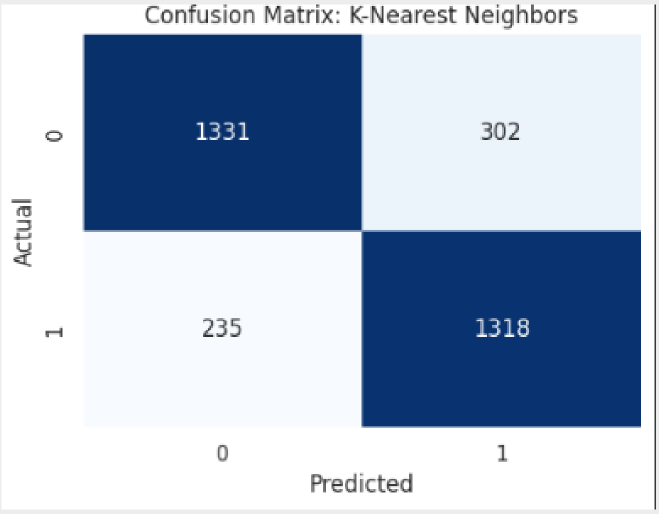


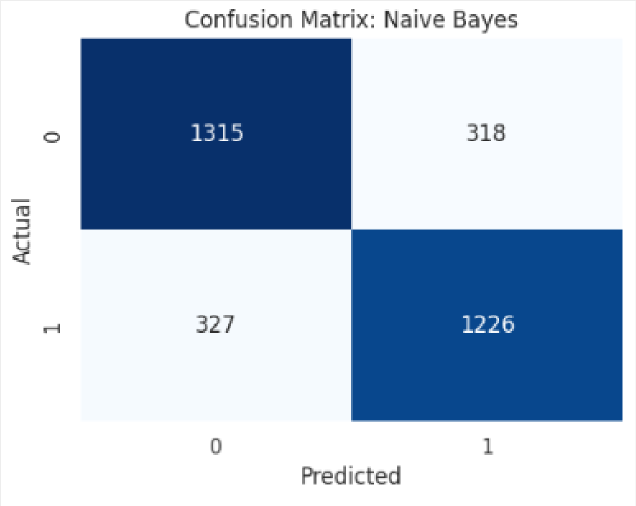


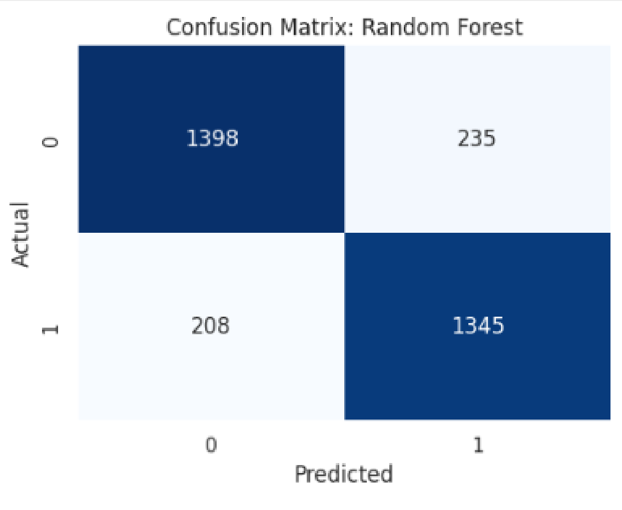
Confusion Matrix comparisons

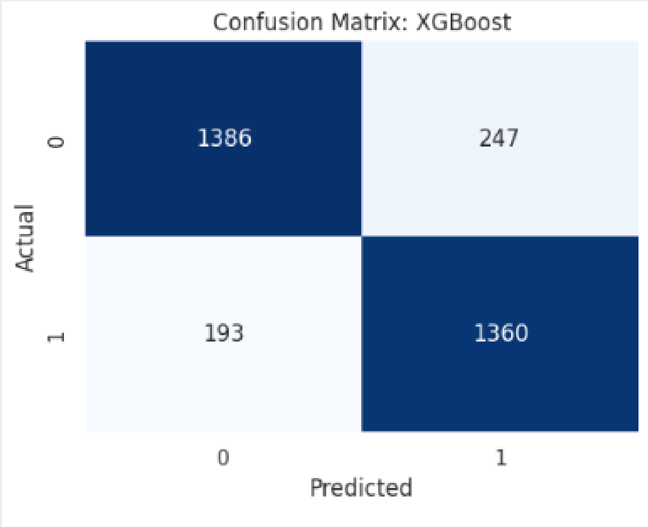


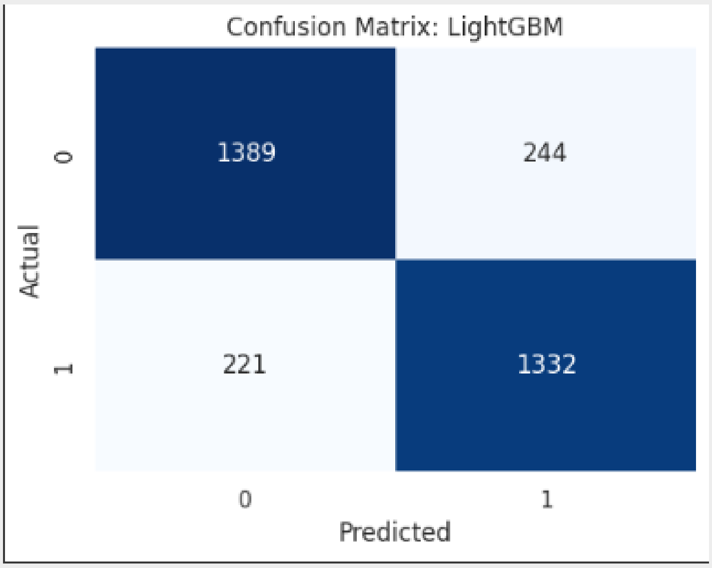












	Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC
0	Logistic Regression	0.7637	0.7525	0.7675	0.7600	0.8470
1	Decision Tree	0.8016	0.7817	0.8229	0.8018	0.8022
2	K-Nearest Neighbors	0.8315	0.8136	0.8487	0.8308	0.8997
3	Naive Bayes	0.7976	0.7940	0.7894	0.7917	0.8765
4	Random Forest	0.8610	0.8513	0.8661	0.8586	0.9369
5	XGBoost	0.8619	0.8463	0.8757	0.8608	0.9371
6	LightGBM	0.8540	0.8452	0.8577	0.8514	0.9333

XGBOOST performs the best!!

Accuracy: 86% , F1-score: 0.86

Hyper-parameter Tuning XGBoost

```python
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
param_dist = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6, 7],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.3, 0.5],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
}

xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

random_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_dist,
    n_iter=50,
    scoring='f1',
    cv=5,
    verbose=1,
    n_jobs=-1
)

random_search.fit(X_train, y_train)

print("Best Parameters:\n", random_search.best_params_)
best_xgb = random_search.best_estimator_
```

	precision	recall	f1-score	support
0	0.89	0.86	0.87	1633
1	0.85	0.89	0.87	1553
accuracy			0.87	3186
macro avg	0.87	0.87	0.87	3186
weighted avg	0.87	0.87	0.87	3186

ROC-AUC Score: 0.9442250524339 24

Final Accuracy: 88%