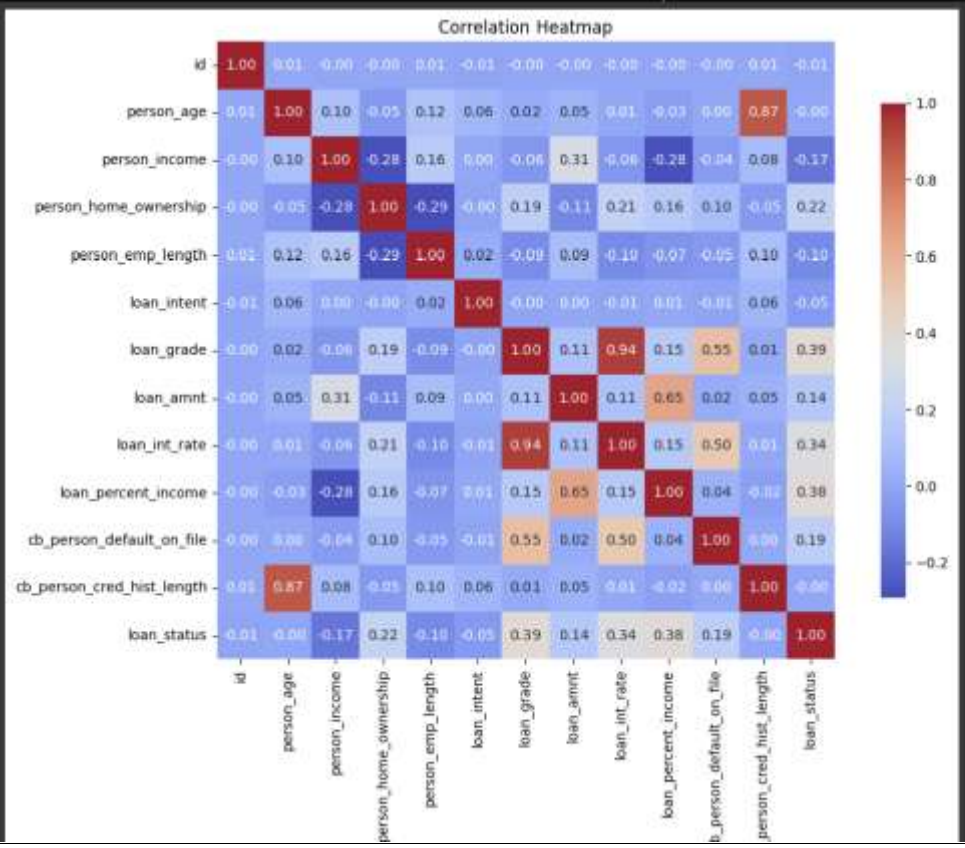
EC 9560 DATA MINING
LAB 04

PRIYATHARSINI S.
2020E122
SEMESTER 7
18 DEC 2024

Feature Selection

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix
correlation_matrix = df.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 8))

# Generate a heatmap
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm',
square=True, cbar_kws={"shrink": .8})

# Set title and labels
plt.title('Correlation Heatmap')
plt.show()
```

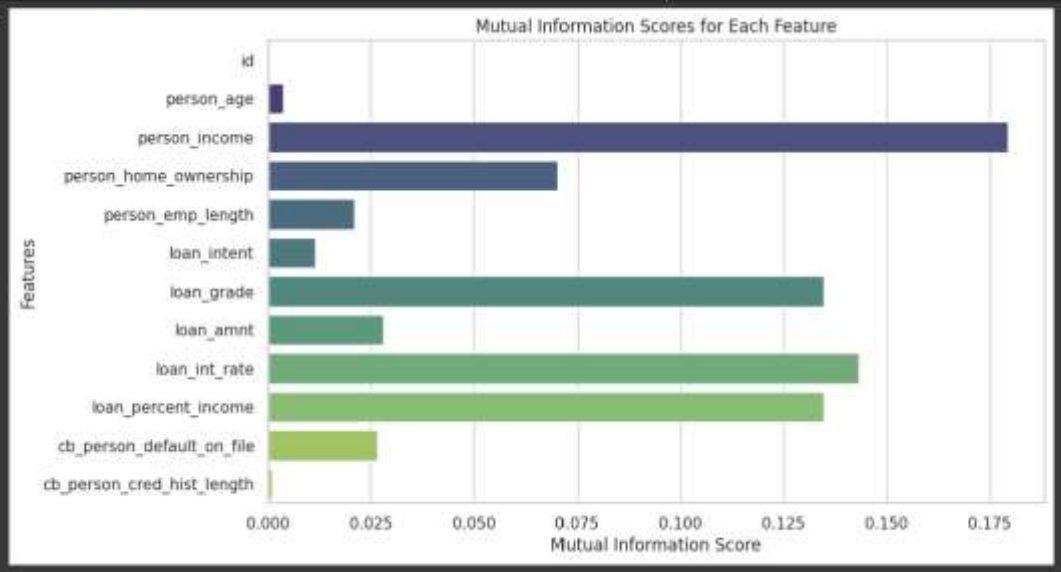


Miscore

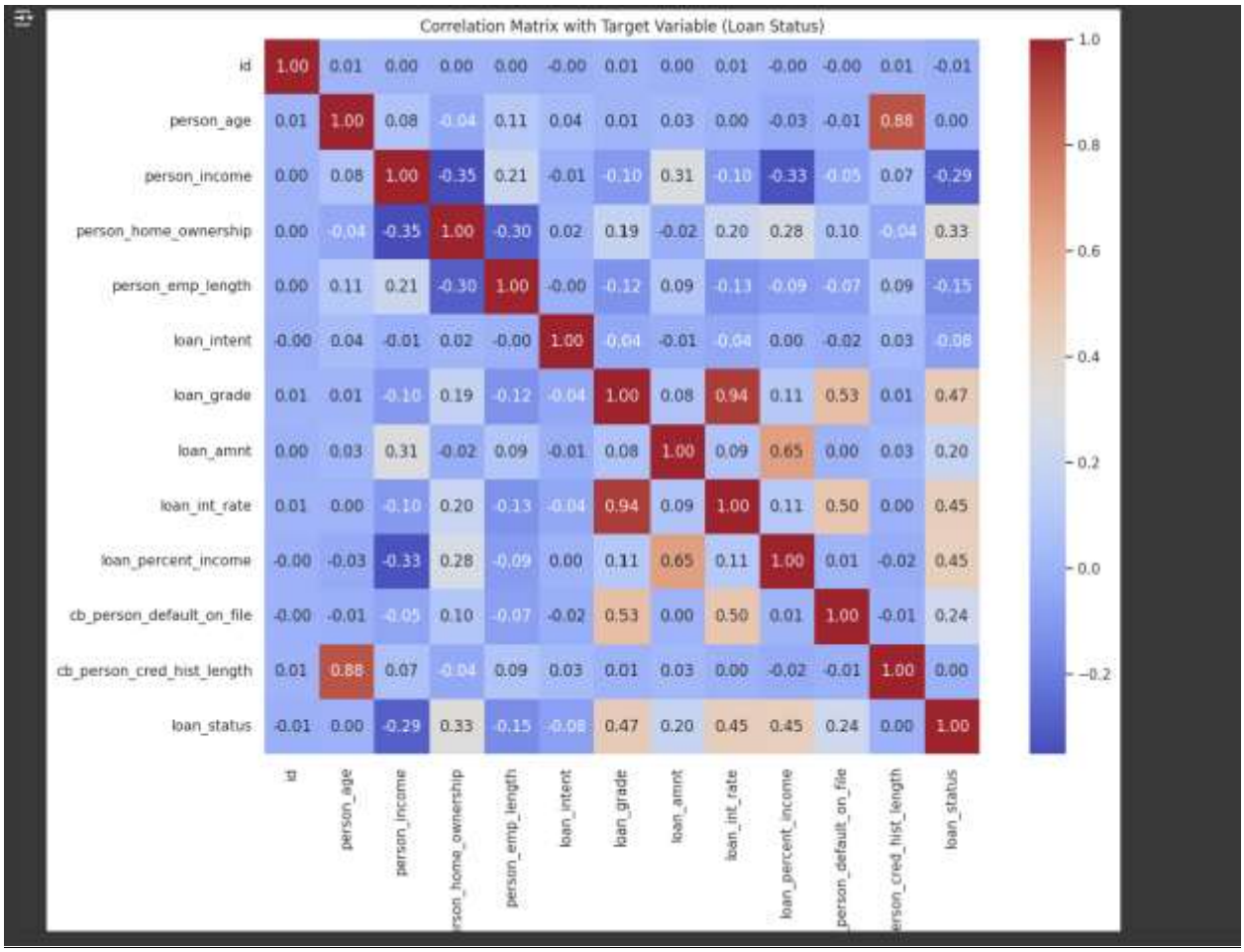
```
from sklearn.feature_selection import mutual_info_classif,
mutual_info_regression

# Use mutual_info_classif for classification tasks or mutual_info_regression
for regression tasks
# Replace X and y with your actual feature matrix and target variable

# For classification tasks
mi_score = mutual_info_classif(X, y, random_state=0)

# For regression tasks
# mi_score = mutual_info_regression(X, y, random_state=0)

# Display MI scores
print(mi_score)

```

```
[0.         0.00371586 0.17936325 0.07044917 0.02101987 0.01156984
 0.13482432 0.02798012 0.14331738 0.13460879 0.02672561 0.00098097]
```

```
[70]
# Assuming X is a DataFrame, get the feature names
feature_names = X.columns if hasattr(X, 'columns') else [f'Feature {i}' for i
in range(len(mi_score))]

# Create a bar plot of MI scores
plt.figure(figsize=(10, 6))
sns.barplot(x=mi_score, y=feature_names, palette="viridis")
plt.xlabel('Mutual Information Score')
plt.ylabel('Features')
plt.title('Mutual Information Scores for Each Feature')
plt.show()
```



Correlation with Target



Feature Removal Analysis

Combining insights from both Mutual Information Scores and the Correlation Matrix, we can identify features that can be safely removed.

1. From the Correlation Matrix:

• Features with very low correlation with `loan_status`:
- `person_age` (0.00 correlation)
- `cb_person_cred_hist_length` (-0.02 correlation)
- `id` (-0.01 correlation)

These features have almost no linear relationship with the target variable.

• Features with high correlation to other features:
- `loan_grade` and `loan_int_rate` have a very high correlation (0.94). Retaining both might introduce multicollinearity.
- `loan_amnt` has a moderately high correlation (0.65) with `loan_percent_income`, which suggests redundancy.

2. From the Mutual Information Scores:

• Features with near-zero mutual information scores:
- `id` (close to 0)
- `person_age` (very low)
- `cb_person_cred_hist_length` (low)
- `loan_intent` (low)

3. Final Analysis - Features to Remove:

Based on the combination of low correlation and low mutual information scores, the following features can be safely removed:
1. `id` - Irrelevant for prediction and has zero contribution.
2. `person_age` - Minimal correlation and mutual information.
3. `cb_person_cred_hist_length` - No significant relationship with `loan_status`.
4. `loan_intent` - Low mutual information and correlation.
5. One of `loan_grade` or `loan_int_rate` - Due to high correlation (0.94), keep only one to avoid redundancy.

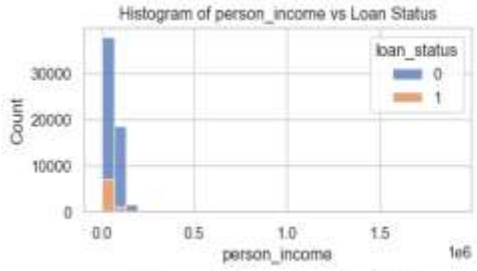
Summary:

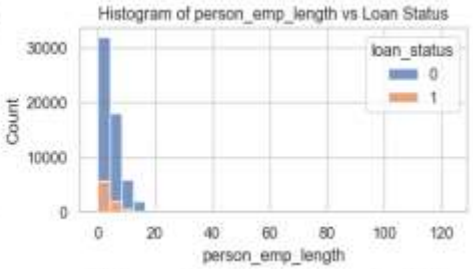
The features for removal are:
- `id`
- `person_age`
- `cb_person_cred_hist_length`
- `loan_intent`
- `loan_int_rate`

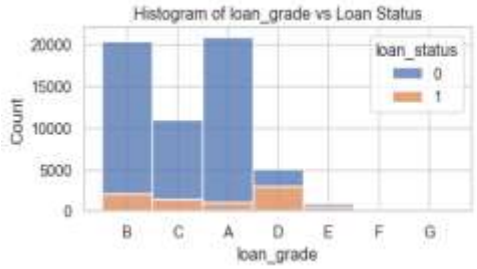
The Features for Selected are:
- `person_income`
- `person_home_ownership`
- `person_emp_length`
- `loan_grade`
- `loan_amnt`
- `loan_percent_income`
- `cb_person_default_on_file`

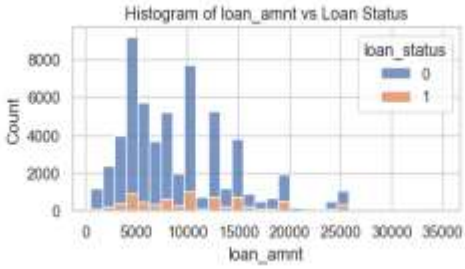
Visualizing Distributions of Selected Features with Loan Status

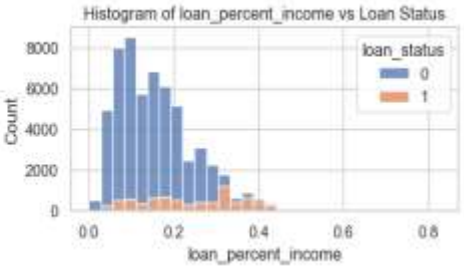
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Ensure 'data' is a DataFrame (replace with your actual DataFrame name)
# cols_to_exclude contains all columns you want to exclude
cols_to_exclude = ['id', 'person_age', 'cb_person_cred_hist_length', 'loan_intent', 'loan_int_rate', 'loan_status']

# Select numerical columns after excluding unnecessary columns
cols_to_plot = data_to_train.drop(columns=cols_to_exclude)

# Create the histograms
plt.figure(figsize=(15, 10))  # Adjust figure size

# Loop through each column and plot
for i, col in enumerate(cols_to_plot, 1):
    plt.subplot(4, 3, i)  # 2 rows, 3 columns layout
    sns.histplot(data=data, x=col, hue="loan_status", multiple="stack", bins=30)
    plt.title(f'Histogram of {col} vs Loan Status')
    plt.xlabel(f'{col}')
    plt.ylabel('Count')
    plt.tight_layout()

plt.show()
```

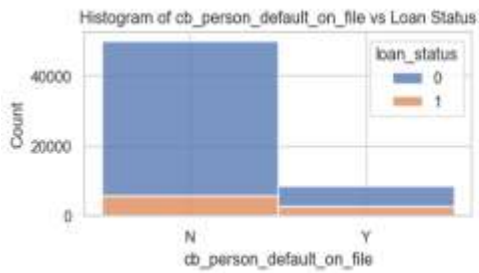












Loan Status Analysis: Histograms of Selected Features

These visualizations are histograms analyzing loan status (default vs non-default) in relation to multiple features. The loan status is divided into two categories:
0 = No default
1= Default

Here are the observations for each histogram:

1. Person Income vs Loan Status

• Observation: Most individuals have lower income (concentrated on the left side).
- Loan defaults (orange bars) occur across all income levels but decrease as income increases.
• Insight: Higher income tends to correlate with fewer defaults.

2. Person Home Ownership vs Loan Status

• Observation: Most borrowers either rent or have a mortgage.
- Defaults are proportionally higher for renters compared to those with a mortgage or owning a home.
• Insight: Renting individuals may have higher financial instability, leading to more defaults.

3. Person Employment Length vs Loan Status

• Observation: Employment lengths are concentrated between 0–20 years.
- Defaults are more frequent among borrowers with shorter employment lengths.
• Insight: Shorter job stability could lead to higher chances of loan default.

4. Loan Grade vs Loan Status

• Observation: Loan grades B, C, and A are the most common.
- Defaults are higher in lower loan grades like D, E, F, and G.
• Insight: Lower loan grades (which represent higher risk) tend to have a greater share of defaults.

5. Loan Amount vs Loan Status

• Observation: Loan amounts are concentrated in the $5,000 to $15,000 range.
- Defaults occur across all loan amounts but are slightly higher for smaller loans.
• Insight: Borrowers with smaller loans may still default, possibly due to tighter financial conditions.

6. Loan Percent Income vs Loan Status

• Observation: Loan amounts relative to income (loan_percent_income) are generally low, with most values under 0.3 (30%).
- Defaults are higher when the loan percent income is relatively high.
• Insight: High loan-to-income ratios indicate affordability issues, leading to defaults.

7. Credit Bureau Default on File vs Loan Status

• Observation: Most borrowers have no prior defaults on their credit history (N).
- However, individuals with a previous default (Y) show a higher likelihood of defaulting again.
• Insight: Borrowers with a history of defaults are at a significantly higher risk of defaulting again.

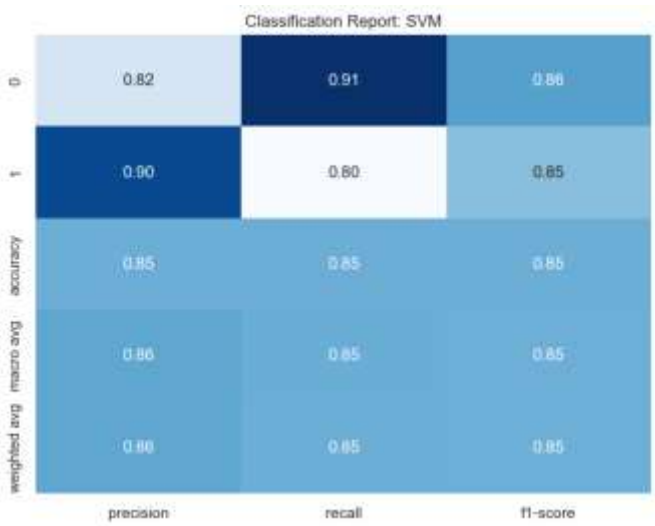
Model Training, Evaluation, and Reporting with Classification Report, Confusion Matrix, and ROC Curve

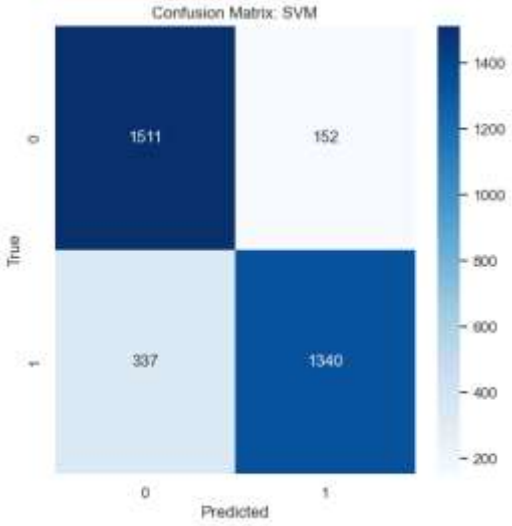
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
import warnings
import seaborn as sns
import matplotlib.pyplot as plt
import os

warnings.filterwarnings("ignore")

# Ensure the directory for saving images exists
if not os.path.exists("model_reports"):
    os.makedirs("model_reports")

# Replace with your actual data
selected_features = ['person_income', 'person_home_ownership', 'person_emp_length',
                     'loan_grade', 'loan_amnt', 'loan_percent_income', 'cb_person_default_on_file']
target_column = 'loan_status'

X = data_to_train[selected_features]
y = data_to_train[target_column]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

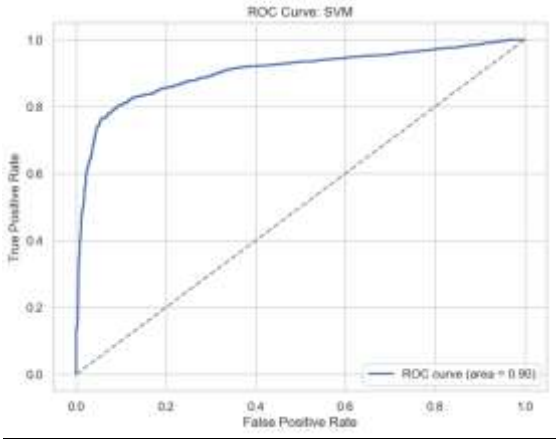
```python
models = {
    "SVM": SVC(probability=True),
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
}

results = {}

for name, model in models.items():
    print(f"\nTraining Model: {name}")

    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    print(f"Train Accuracy: {train_acc:.4f}")
    print(f"Test Accuracy: {test_acc:.4f}")

    print("Classification Report for Test Set:")
    print(classification_report(y_test, y_test_pred))

    results[name] = {"Train Accuracy": train_acc, "Test Accuracy": test_acc}
```

```
        # Classification Report Image
        classification_rep = classification_report(y_test, y_test_pred, output_dict=True)
        fig, ax = plt.subplots(figsize=(8, 6))
        sns.heatmap(pd.DataFrame(classification_rep).iloc[:-1, :].T, annot=True, cmap='Blues', cbar=False, fmt='.2f')
        ax.set_title(f"Classification Report: {name}")
        fig.savefig(f"model_reports/{name}_classification_report.png")

        # Confusion Matrix
        cm = confusion_matrix(y_test, y_test_pred)
        fig, ax = plt.subplots(figsize=(6, 6))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['0', '1'], yticklabels=['0', '1'])
        ax.set_title(f"Confusion Matrix: {name}")
        ax.set_xlabel('Predicted')
        ax.set_ylabel('True')
        fig.savefig(f"model_reports/{name}_confusion_matrix.png")

        # ROC Curve
        fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
        roc_auc = auc(fpr, tpr)
        fig, ax = plt.subplots(figsize=(8, 6))
        ax.plot(fpr, tpr, color='b', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
        ax.plot([0, 1], [0, 1], color='gray', linestyle='--')
        ax.set_title(f"ROC Curve: {name}")
        ax.set_xlabel('False Positive Rate')
        ax.set_ylabel('True Positive Rate')
        ax.legend(loc='lower right')
        fig.savefig(f"model_reports/{name}_roc_curve.png")

print("\nModel Performance Summary:")
for model_name, accuracy in results.items():
    print(f"{model_name}: Train Accuracy = {accuracy['Train Accuracy']:.4f}, Test Accuracy = {accuracy['Test Accuracy']:.4f}")
```

SVM



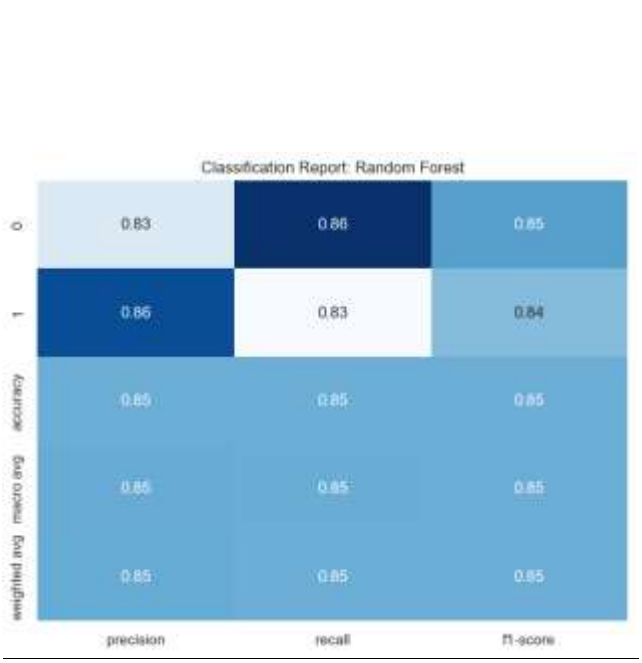


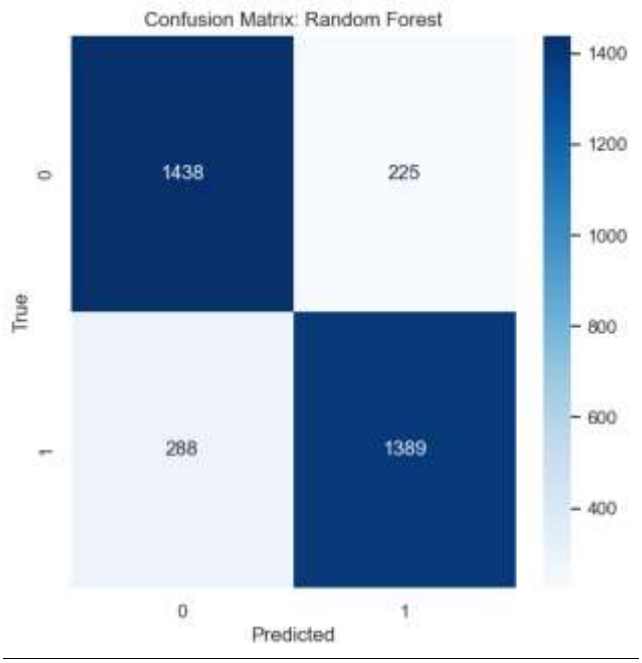


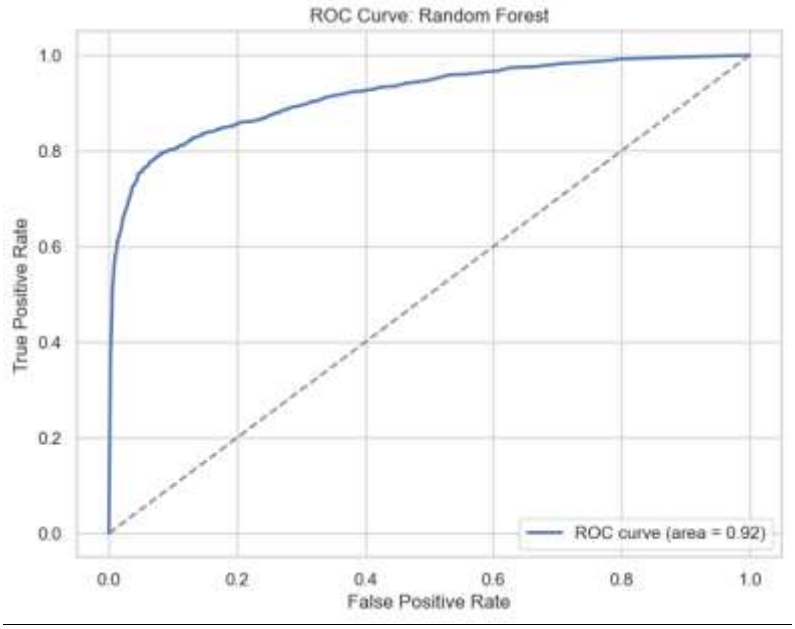
The SVM model performs well with the following insights:

1. Accuracy: 85% — good, but slightly lower than XGBoost.
2. Precision: Higher for class 1 (0.90), but recall for class 1 is only 0.80, showing it struggles with false negatives.
3. AUC Score: 0.89 — reflects strong classification performance

Random Forest





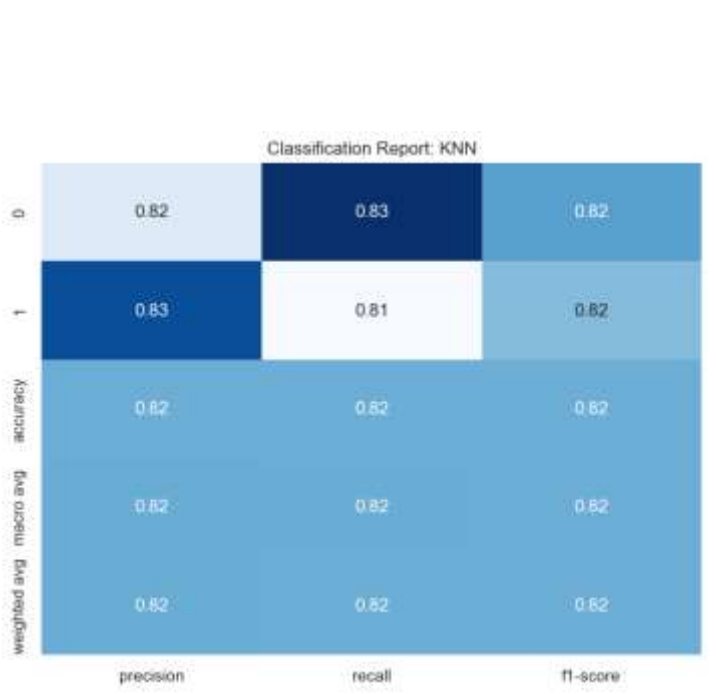


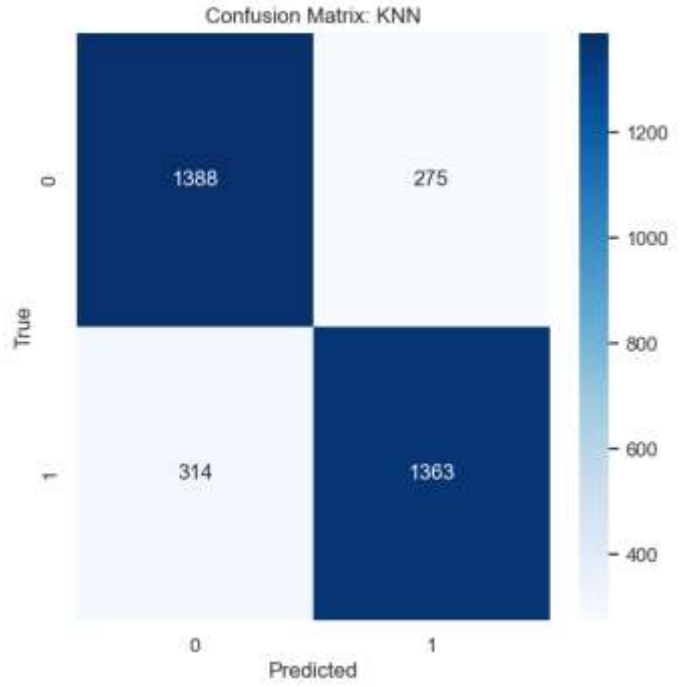
The Random Forest classifier achieves strong performance:

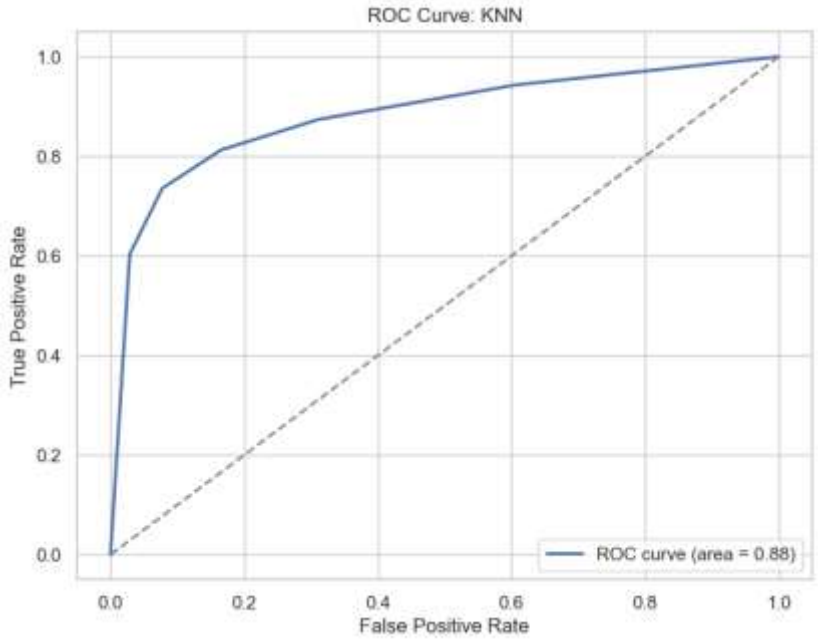
- Balanced precision, recall, and F1-score for both classes.
- A confusion matrix showing low misclassification rates.
- An excellent ROC curve with an AUC of 0.92, indicating strong model performance.

This suggests the model effectively distinguishes between the two classes.

K Nearest Neighbor





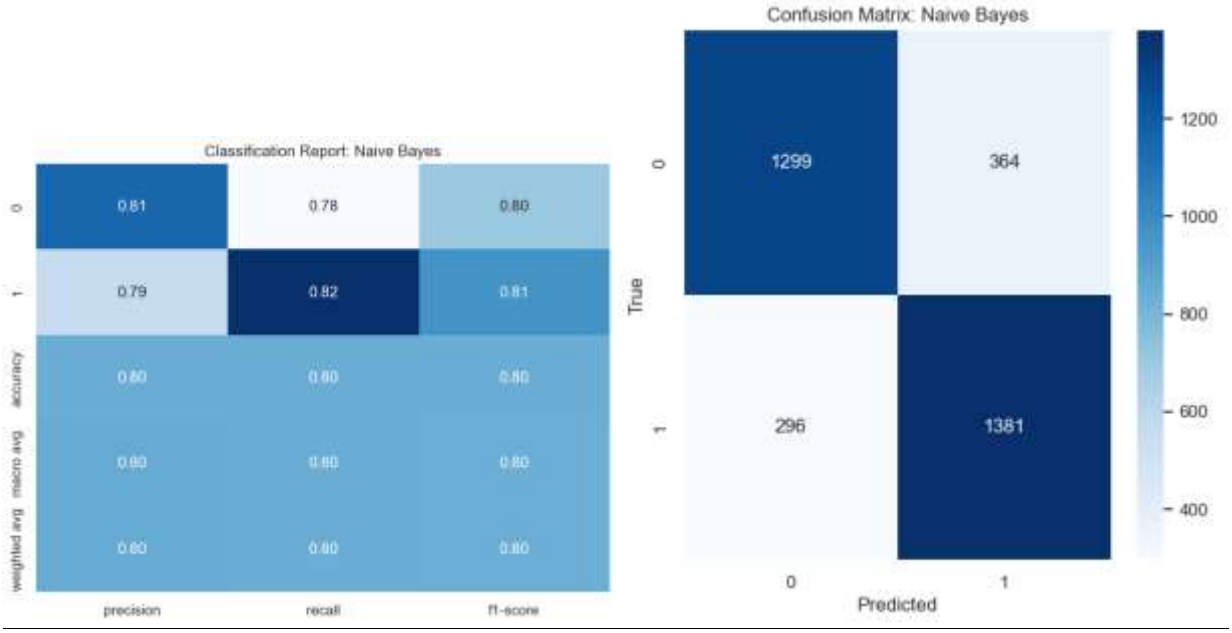


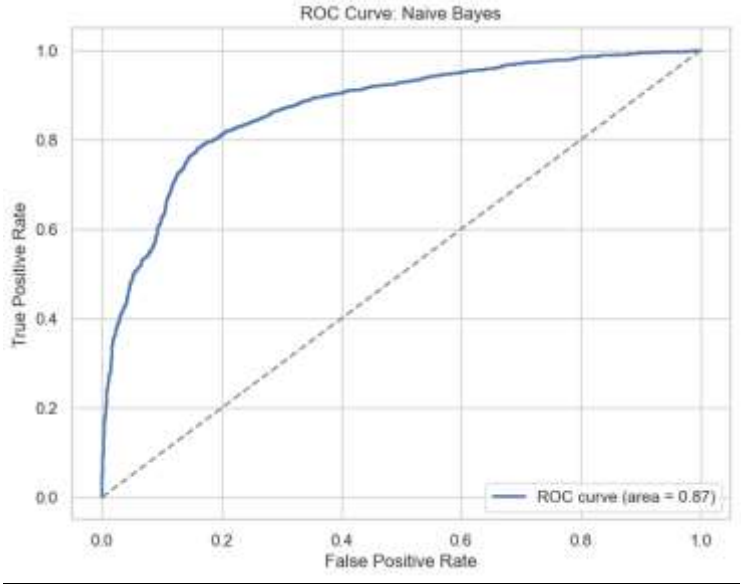
The **KNN classifier** performs well overall but is slightly less effective than the Random Forest model:

- Precision, recall, and F1-scores are balanced at **0.82**.
- The confusion matrix shows a moderate misclassification rate.
- The AUC of **0.88** is good but lower than the Random Forest's **0.92**.

This suggests that KNN is a reasonable model, but Random Forest performs better on this dataset.

Naïve Bayes

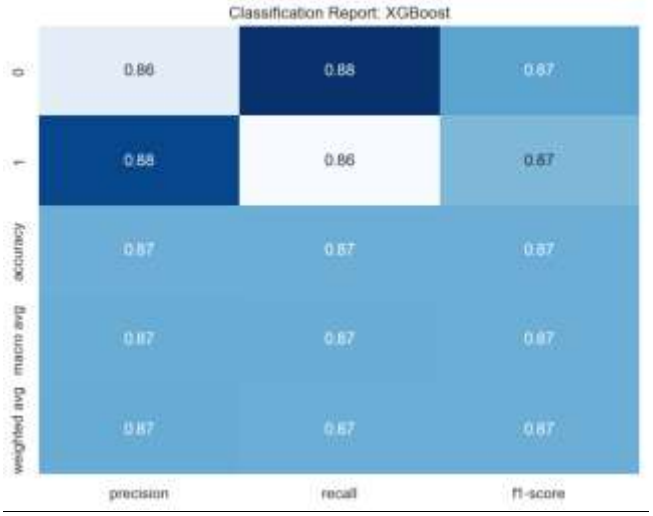


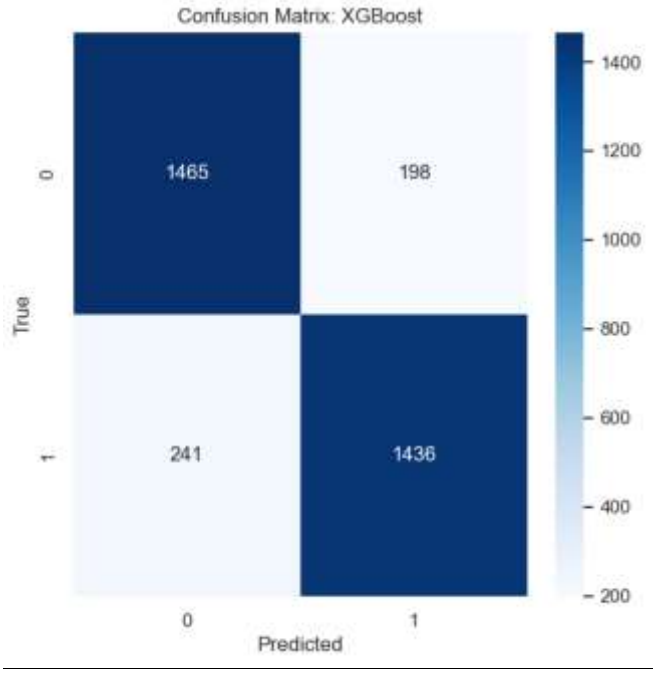


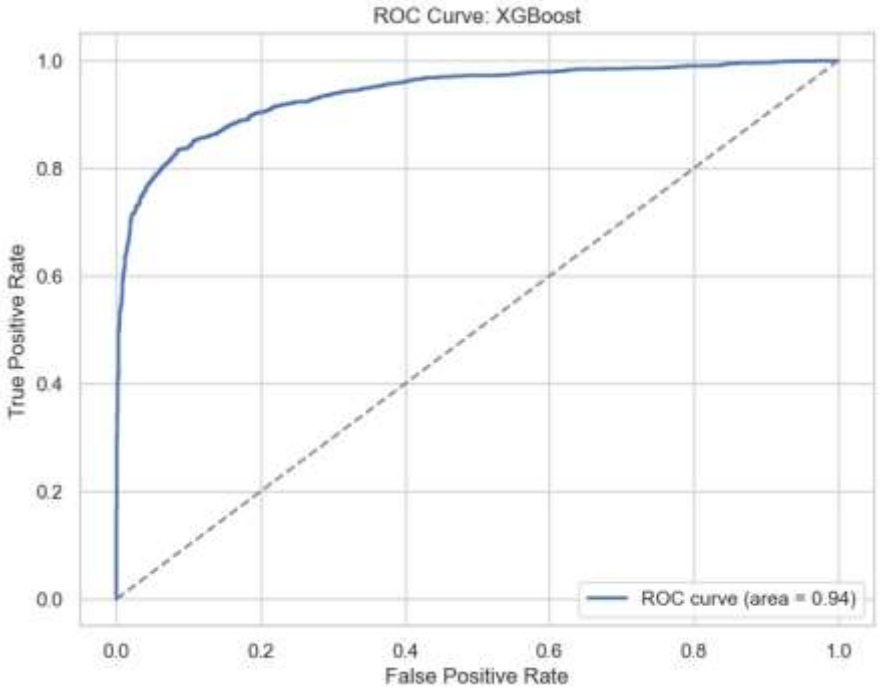
The Naive Bayes model achieves:

- 80% accuracy overall.
- Strong precision, recall, and F1-scores across both classes.
- An AUC score of 0.87, indicating good classification ability.
  While the model performs well, there are opportunities to further reduce false positives and false negatives.

XGBoost







The XGBoost model shows significant improvements compared to Naive Bayes:

- Accuracy increased to 87%.
- Both precision and recall are higher for each class.
- Lower false positives and false negatives, as seen in the confusion matrix.
- The AUC score of 0.94 indicates excellent performance in distinguishing between classes.

Summary of Each Model

Model	Accuracy	Precision	Recall	F1-Score	AUC Score
Random Forest	85%	0.85	0.85	0.85	0.92
KNN	82%	0.82	0.82	0.82	0.88
Naive Bayes	80%	0.80	0.80	0.80	0.87
XGBoost	87%	0.87	0.87	0.87	0.94
SVM	85%	0.85	0.85	0.85	0.89

Detailed Analysis

1. Accuracy
   - XGBoost achieves the highest accuracy (87%), followed by Random Forest and SVM at 85%.
   - KNN lags slightly at 82%, and Naive Bayes has the lowest accuracy (80%).
2. Precision, Recall, and F1-Score
   - XGBoost again performs the best with balanced precision, recall, and F1-scores at 0.87 for both classes.
   - Random Forest and SVM follow closely with average values of 0.85.
   - KNN is slightly weaker at 0.82, and Naive Bayes shows the lowest scores at 0.80.
3. Confusion Matrix Insights
   - XGBoost has the fewest false positives and false negatives, leading to better recall and precision.
   - Random Forest performs better than KNN and Naive Bayes but has more misclassifications than XGBoost.
   - SVM struggles with false negatives, especially for class 1, despite good precision.
   - Naive Bayes shows the most false positives and false negatives, indicating weaker overall performance.
4. ROC-AUC Score
   - XGBoost has the highest AUC score (0.94), indicating excellent class discrimination.
   - Random Forest follows closely at 0.92.
   - SVM achieves a respectable 0.89, while KNN and Naive Bayes score 0.88 and 0.87, respectively.

Conclusion

XGBoost:

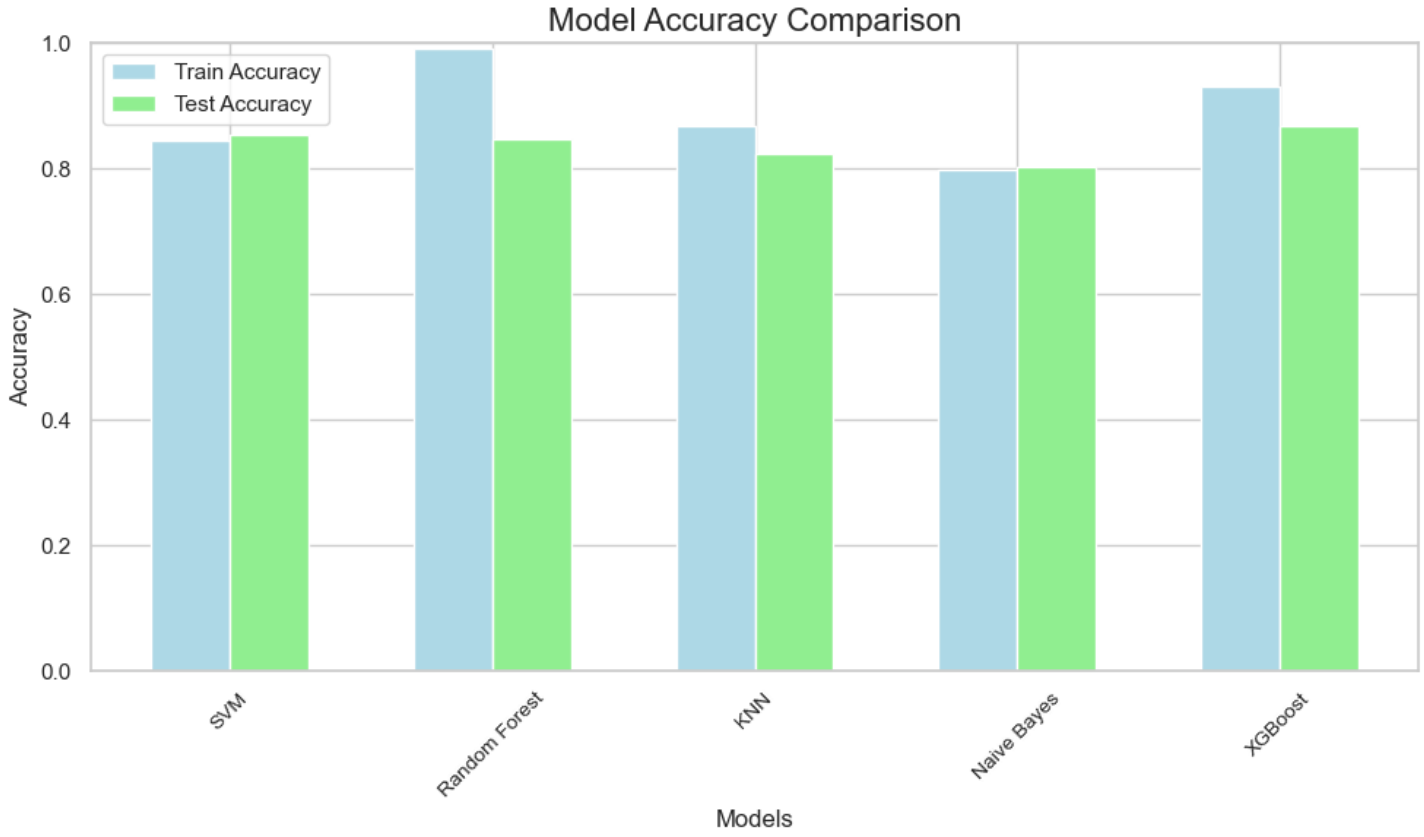
- Highest accuracy (87%).
- Best precision, recall, and F1-scores.
- Lowest false positives and false negatives.
- Best AUC score (0.94).

- XGBoost is the best performing model across all evaluation metrics. It achieves the highest accuracy, precision, recall, and AUC score while minimizing errors.
- Random Forest is a strong second choice, followed by SVM.
- KNN and Naive Bayes are less effective, with Naive Bayes being the weakest performer.

For tasks requiring high classification performance, XGBoost is the clear choice.

Model Accuracy Comparison: Train vs Test Performance

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

model_names = list(results.keys())
train_accuracies = [results[model]['Train Accuracy'] for model in model_names]
test_accuracies = [results[model]['Test Accuracy'] for model in model_names]

x = range(len(model_names))

plt.figure(figsize=(10, 6))
width = 0.3

plt.bar(x, train_accuracies, width=width, label="Train Accuracy", color="lightblue")
plt.bar([i + width for i in x], test_accuracies, width=width, label="Test Accuracy", color="lightgreen")

plt.title("Model Accuracy Comparison", fontsize=16)
plt.xlabel("Models", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.xticks([i + width / 2 for i in x], model_names, rotation=45, fontsize=10)
plt.ylim(0, 1)
plt.tight_layout()

plt.legend()

plt.show()
```



The best model appears to be XGBoost, as it has:

- The highest test accuracy (≈ 0.88).
- A small gap between training and testing accuracy, showing good generalization.