Date: 08-08-2025 **Name:** Harini LV
Experiment: 3 **Roll No:** 3122237001016

Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An autonomous Institution affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	Semester	V
Subject Code & Name	ICS1512 & Machine Learning Algorithms Laboratory		
Academic year	2025-2026 (Odd)	Batch:2023-2028	**Due date:**

Experiment 3: Email Spam or Ham Classification using Naive Bayes, KNN, and SVM

Aim: To predict the loan amount sanctioned to users using Linear Regression on historical data, and analyze model performance using visual and statistical metrics.

Libraries used:

- Pandas - for data handling
- numpy - for numerical operations
- matplotlib.pyplot and seaborn - for visualization
- sklearn - for model building and evaluation

Objective: To build a linear regression model using Scikit-learn to predict the loan amount, perform exploratory data analysis, visualize model performance, and interpret results.

Mathetical/theoritical description: The linear regression model expresses the relationship between the input features and the predicted output as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

Where:

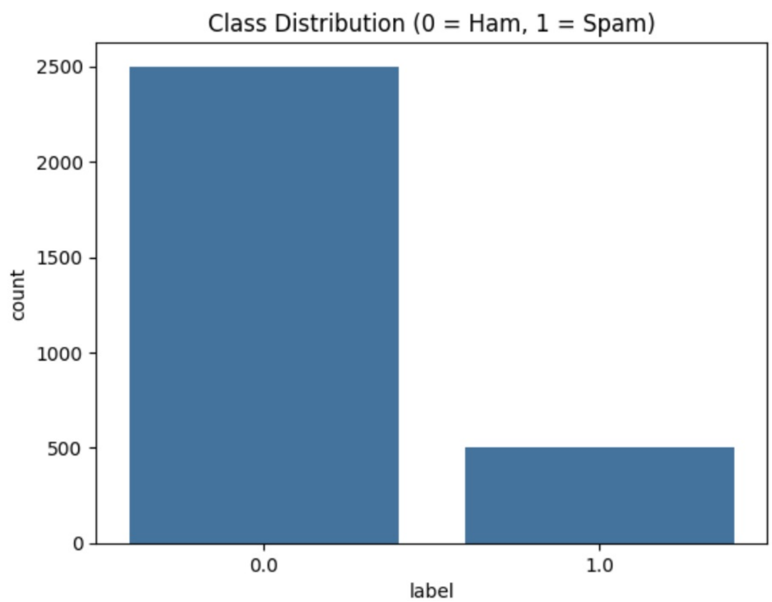
- y is the predicted loan amount,
- x_i are the input features (e.g., income, credit score, etc.),
- β_i are the coefficients (weights) learned by the model,
- ϵ is the error term (residual).

CODE:

```
# ========== 1. Imports ==========
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')

from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, confusion_matrix,
    roc_curve, auc, fbeta_score, matthews_corrcoef
)
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.stats import zscore

# ========== 2. Load Dataset ==========
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ML LAB SEM 5/spam_or_not_spam.c
df.dropna(subset=['email'], inplace=True)

# ========== 3. EDA ==========
# Fill missing numeric values if any
num_cols = df.select_dtypes(include=[np.number]).columns
if len(num_cols) > 0:
    imputer = SimpleImputer(strategy='mean')
    df[num_cols] = imputer.fit_transform(df[num_cols])

# Remove outliers from numeric columns
if len(num_cols) > 0:
    z_scores = np.abs(zscore(df[num_cols]))
    df = df[(z_scores < 3).all(axis=1)]

print(df.info())
print(df['label'].value_counts())
sns.countplot(x='label', data=df)
plt.title("Class Distribution (0 = Ham, 1 = Spam)")
plt.show()
```

OUTPUT

```
<class 'pandas.core.frame.DataFrame'>
Index: 2999 entries, 0 to 2999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   email   2999 non-null   object
 1   label   2999 non-null   float64
dtypes: float64(1), object(1)
memory usage: 70.3+ KB
None
label
0.0    2500
1.0     499
Name: count, dtype: int64
```




```
# ========== 4. Text Preprocessing ==========
tfidf = TfidfVectorizer(stop_words='english', max_features=1000)
X = tfidf.fit_transform(df['email']).toarray()
y = df['label']

# ========== 5. Scaling ==========
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("Shape of TF-IDF matrix before scaling:", X.shape)
print("Shape of TF-IDF matrix after scaling:", X_scaled.shape)

# Print first 5 rows before and after scaling
```

```
print("\nBefore Scaling (TF-IDF values):")
print(pd.DataFrame(X[:5, :10]))

print("\nAfter Scaling:")
print(pd.DataFrame(X_scaled[:5, :10]))
```

OUTPUT

```
Shape of TF-IDF matrix before scaling: (2999, 1000)
Shape of TF-IDF matrix after scaling: (2999, 1000)

Before Scaling (TF-IDF values):
          0    1         2    3    4    5    6    7    8    9
0  0.031672  0.0  0.050609  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.000000  0.0  0.000000  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.000000  0.0  0.000000  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.098441  0.0  0.000000  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.000000  0.0  0.000000  0.0  0.0  0.0  0.0  0.0  0.0  0.0

After Scaling:
          0        1         2         3         4         5         6  \
0  0.435582 -0.13322  1.980391 -0.126859 -0.126388 -0.179581 -0.112595
1 -0.454271 -0.13322 -0.180178 -0.126859 -0.126388 -0.179581 -0.112595
2 -0.454271 -0.13322 -0.180178 -0.126859 -0.126388 -0.179581 -0.112595
3  2.311546 -0.13322 -0.180178 -0.126859 -0.126388 -0.179581 -0.112595
4 -0.454271 -0.13322 -0.180178 -0.126859 -0.126388 -0.179581 -0.112595

          7         8         9
0 -0.120289 -0.137888 -0.125068
1 -0.120289 -0.137888 -0.125068
2 -0.120289 -0.137888 -0.125068
3 -0.120289 -0.137888 -0.125068
4 -0.120289 -0.137888 -0.125068
```

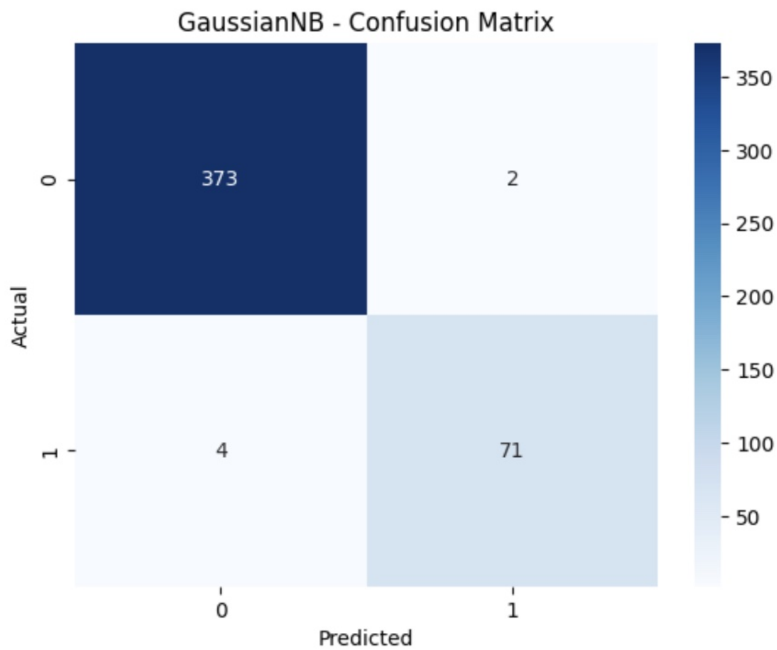
```
# ========== 6. Split Data Separately ==========
# For models that require scaled features (KNN, SVM)
X_train_scaled, X_temp_scaled, y_train_scaled, y_temp_scaled = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)
X_val_scaled, X_test_scaled, y_val_scaled, y_test_scaled = train_test_split(
    X_temp_scaled, y_temp_scaled, test_size=0.5, random_state=42, stratify=y_temp_scaled
)

# For models that require unscaled features (Naive Bayes)
X_train_raw, X_temp_raw, y_train_raw, y_temp_raw = train_test_split(
```

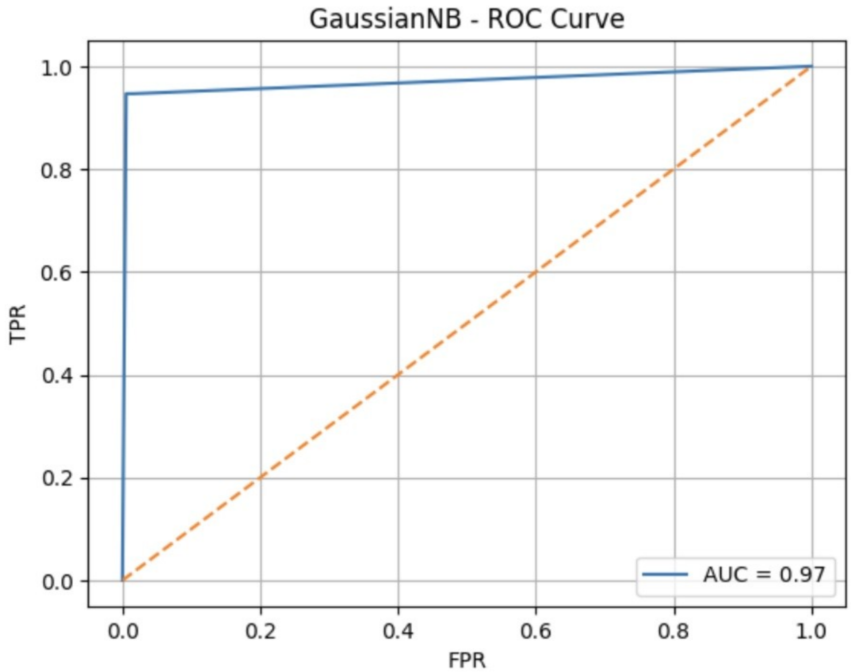
```
    X, y, test_size=0.3, random_state=42, stratify=y
)
X_val_raw, X_test_raw, y_val_raw, y_test_raw = train_test_split(
    X_temp_raw, y_temp_raw, test_size=0.5, random_state=42, stratify=y_temp_raw
)

# ========== 7. Evaluation Helper ==========
def evaluate_model(model, name, X_test, y_test):
    y_pred = model.predict(X_test)
    print(f"\n{name} Evaluation:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))
    print("F-beta Score (β=0.5):", fbeta_score(y_test, y_pred, beta=0.5))
    print("Matthews Corr Coef:", matthews_corrcoef(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"{name} - Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)[:, 1]
    else:
        y_score = model.decision_function(X_test)

    fpr, tpr, _ = roc_curve(y_test, y_score)
    plt.plot(fpr, tpr, label=f"AUC = {auc(fpr, tpr):.2f}")
    plt.plot([0, 1], [0, 1], linestyle="--")
    plt.title(f"{name} - ROC Curve")
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.legend()
    plt.grid()
    plt.show()

# ========== 8. Naïve Bayes ==========
for name, model in {
    "GaussianNB": GaussianNB(),
    "MultinomialNB": MultinomialNB(),
    "BernoulliNB": BernoulliNB()
}.items():
```

```
    model.fit(X_train_raw, y_train_raw)
    evaluate_model(model, name, X_test_raw, y_test_raw)
```

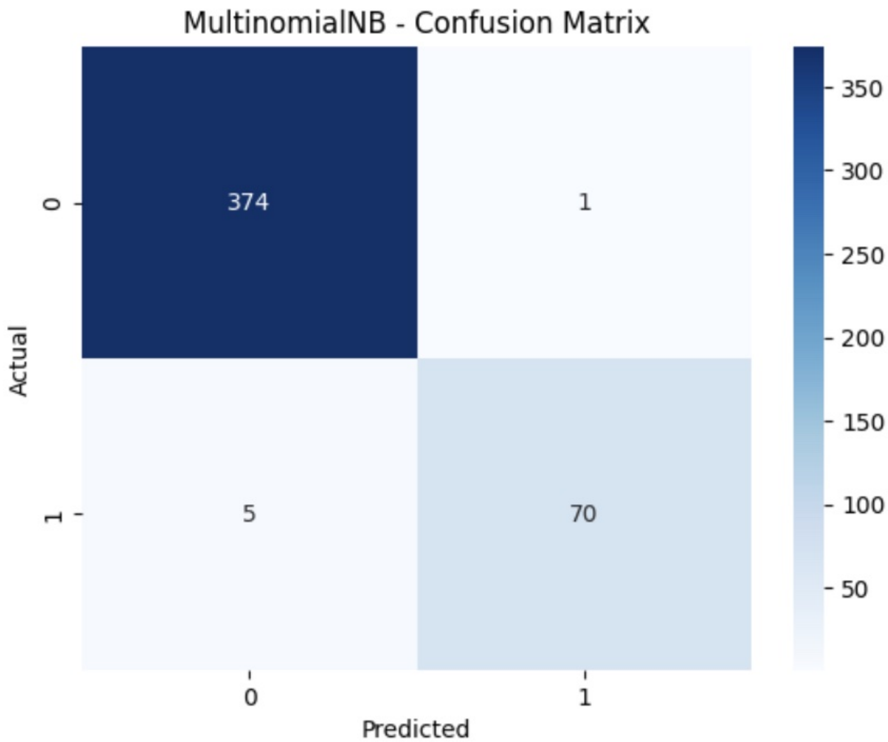
OUTPUT

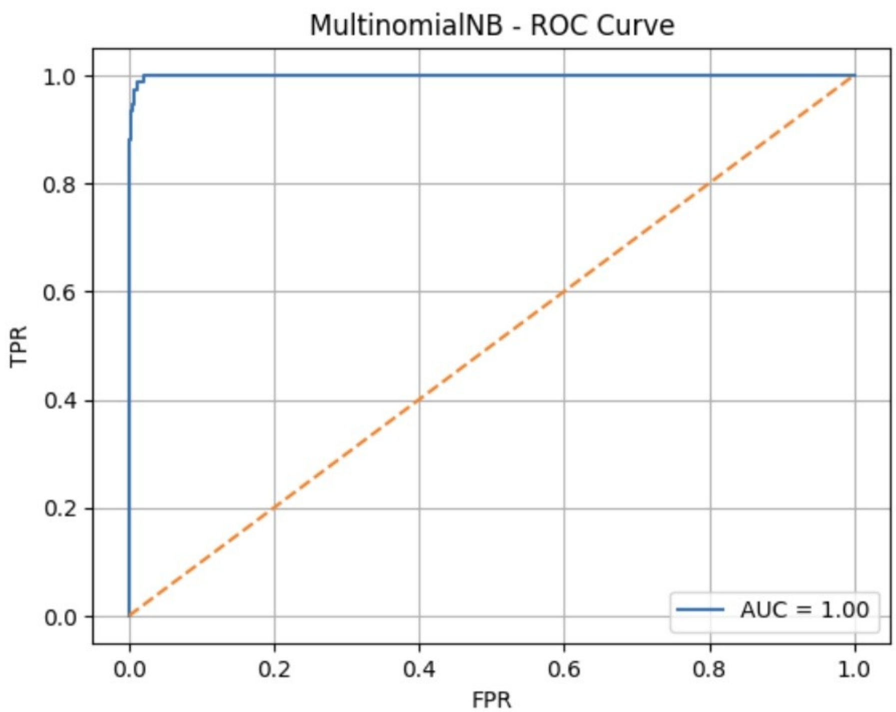
```
GaussianNB Evaluation:
Accuracy: 0.9866666666666667
Precision: 0.9726027397260274
Recall: 0.9466666666666667
F1 Score: 0.9594594594594594
F-beta Score (β=0.5): 0.9673024523160763
Matthews Corr Coef: 0.9516069343072303
```





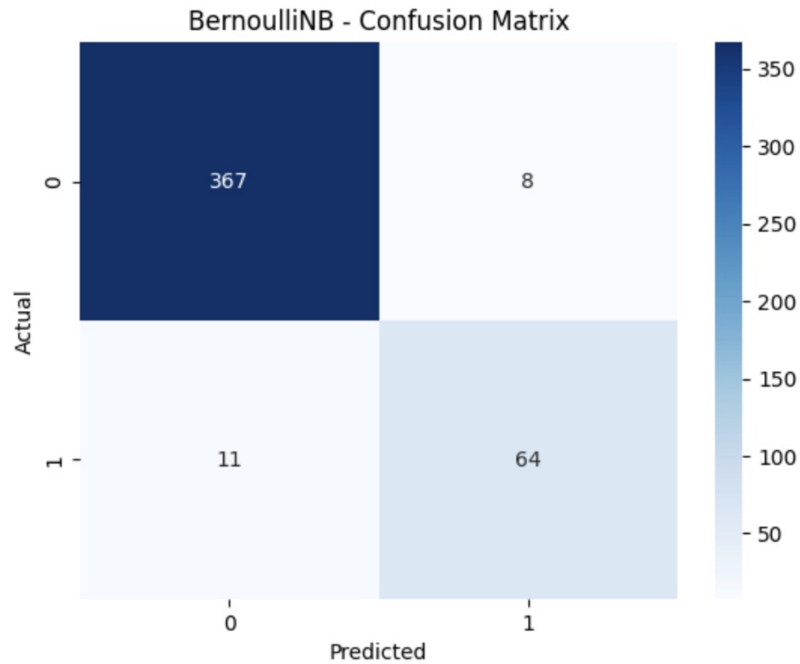

```
MultinomialNB Evaluation:
Accuracy: 0.9866666666666667
Precision: 0.9859154929577465
Recall: 0.9333333333333333
F1 Score: 0.958904109589041
F-beta Score (β=0.5): 0.9749303621169917
Matthews Corr Coef: 0.9514624328283552
```

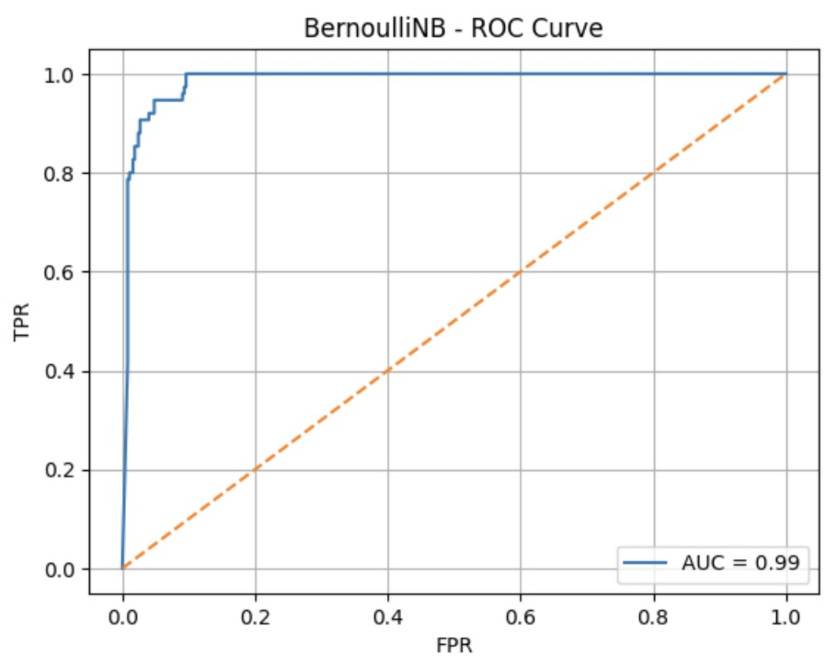





```
BernoulliNB Evaluation:
Accuracy: 0.9577777777777777
Precision: 0.8888888888888888
Recall: 0.8533333333333334
F1 Score: 0.8707482993197279
```

```
F-beta Score (β=0.5): 0.8815426997245179
Matthews Corr Coef: 0.8457800632220621
```



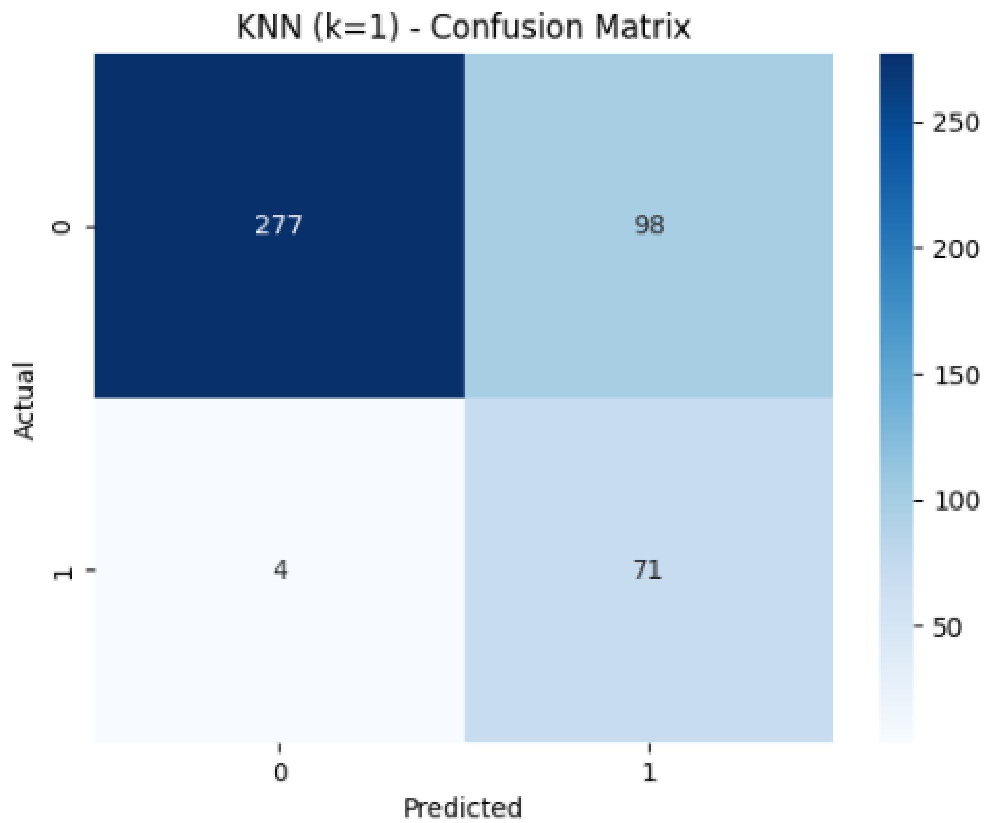


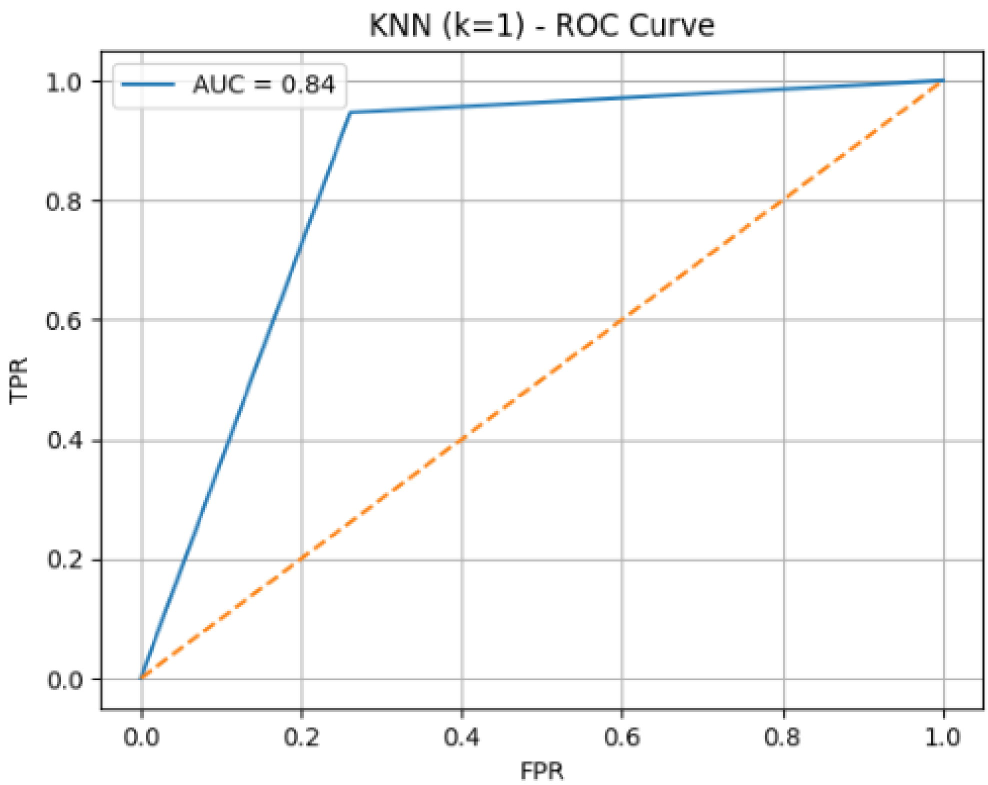

```
# ========== 9. KNN ==========
for k in [1, 3, 5, 7]:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train_scaled, y_train_scaled)
    evaluate_model(model, f"KNN (k={k})", X_test_scaled, y_test_scaled)
```

```
for algo in ["kd_tree", "ball_tree"]:
    model = KNeighborsClassifier(algorithm=algo)
    model.fit(X_train_scaled, y_train_scaled)
    evaluate_model(model, f"KNN ({algo})", X_test_scaled, y_test_scaled)
```

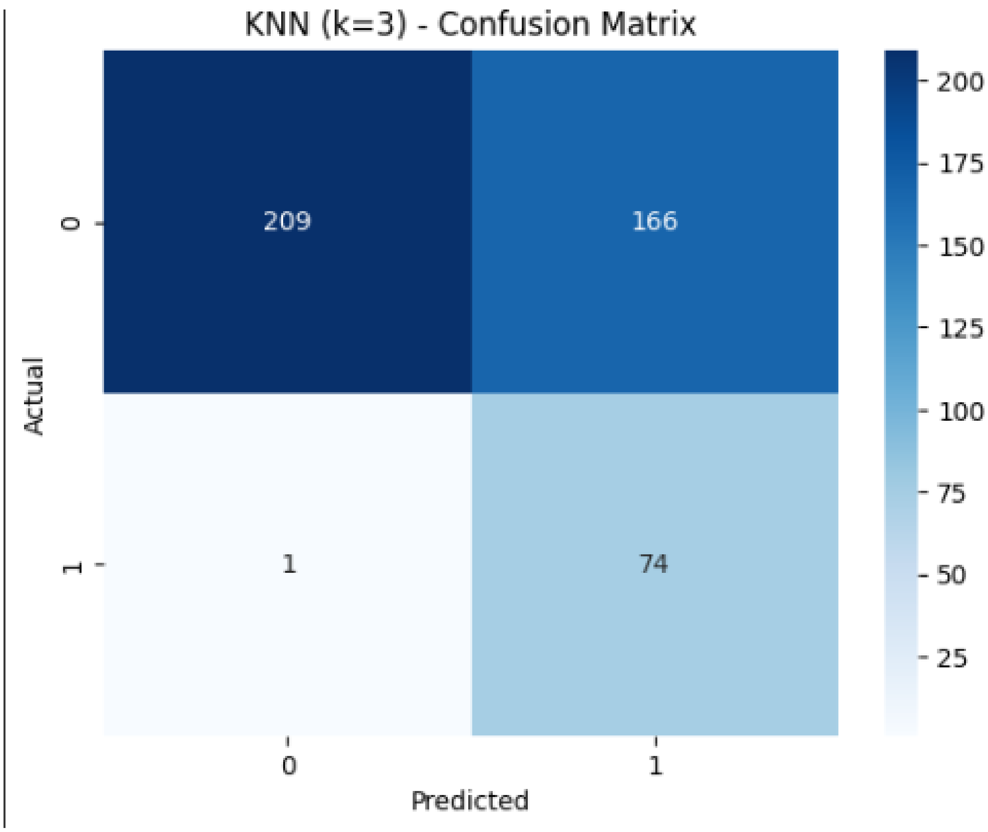
OUTPUT

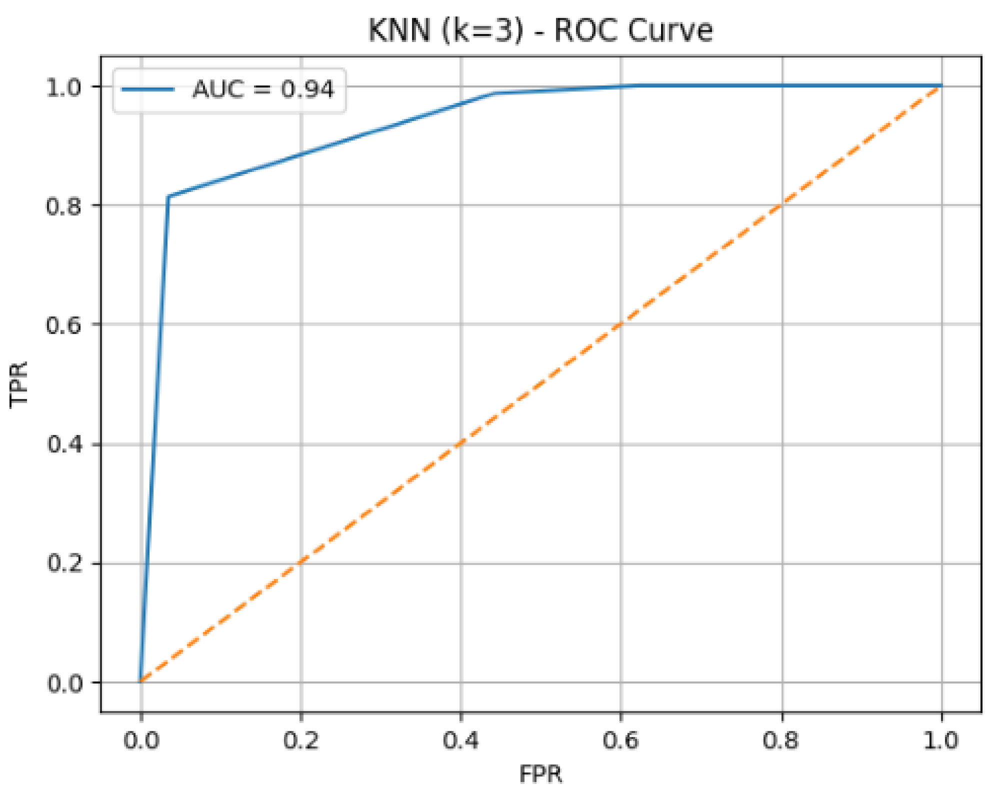
```
KNN (k=1) Evaluation:
Accuracy: 0.7733333333333333
Precision: 0.42011834319526625
Recall: 0.9466666666666667
F1 Score: 0.5819672131147541
F-beta Score (β=0.5): 0.47270306258322237
Matthews Corr Coef: 0.5274139454909135
```





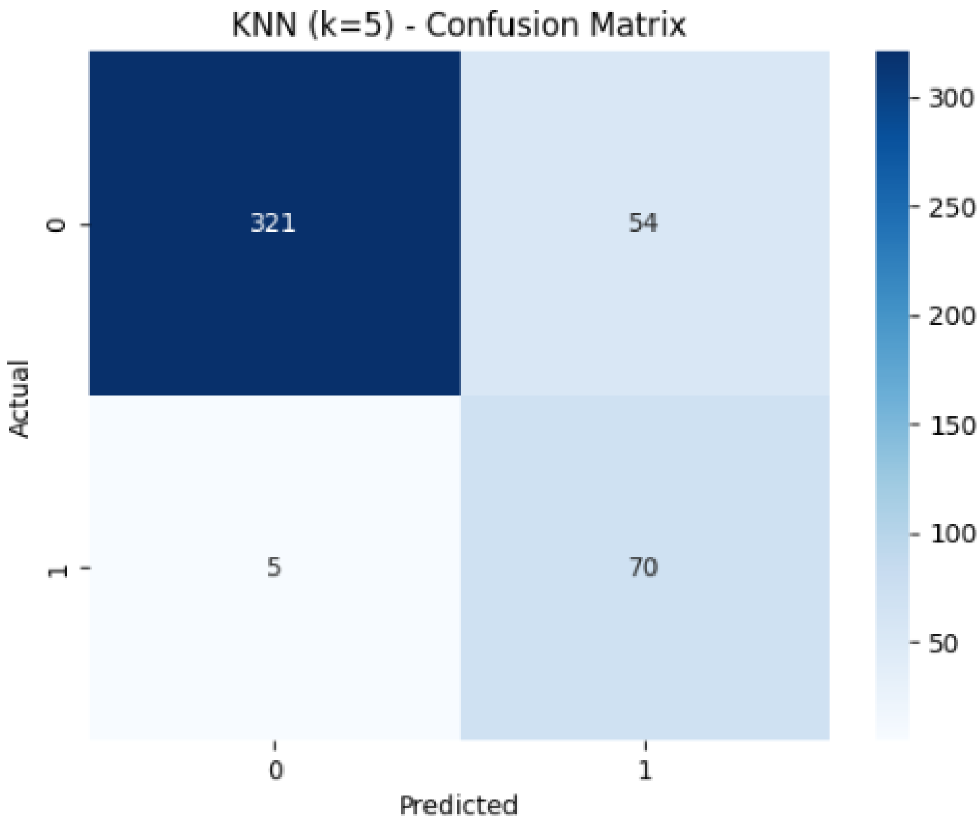

```
KNN (k=3) Evaluation:
Accuracy: 0.6288888888888889
Precision: 0.30833333333333335
Recall: 0.9866666666666667
F1 Score: 0.46984126984126984
F-beta Score (β=0.5): 0.357487922705314
Matthews Corr Coef: 0.40637772717369386
```

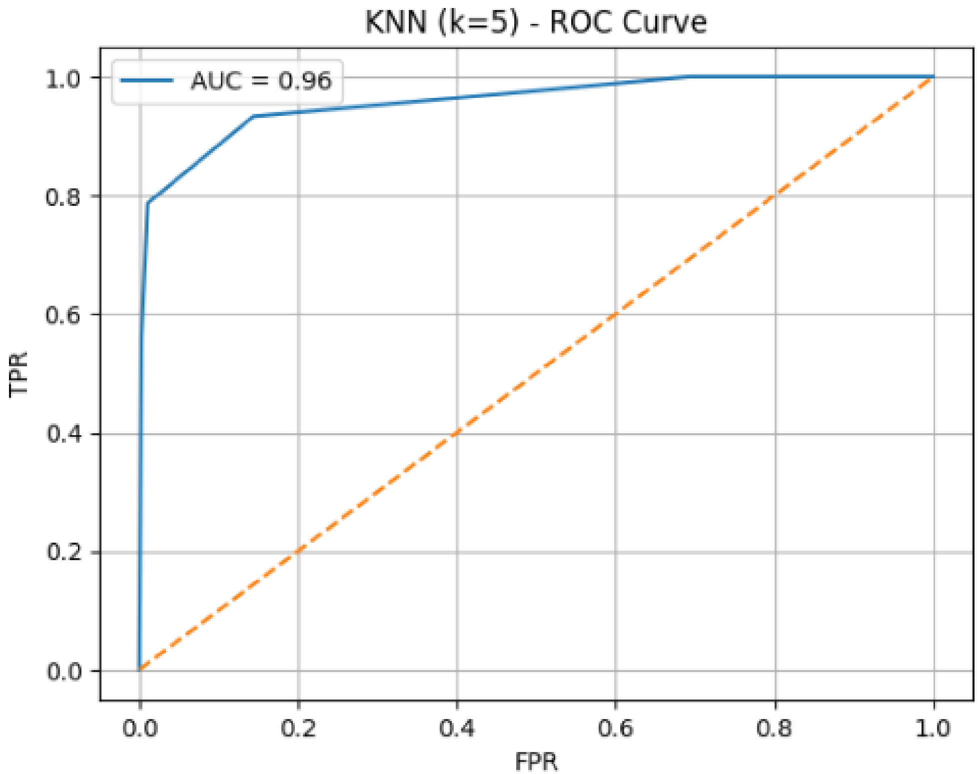




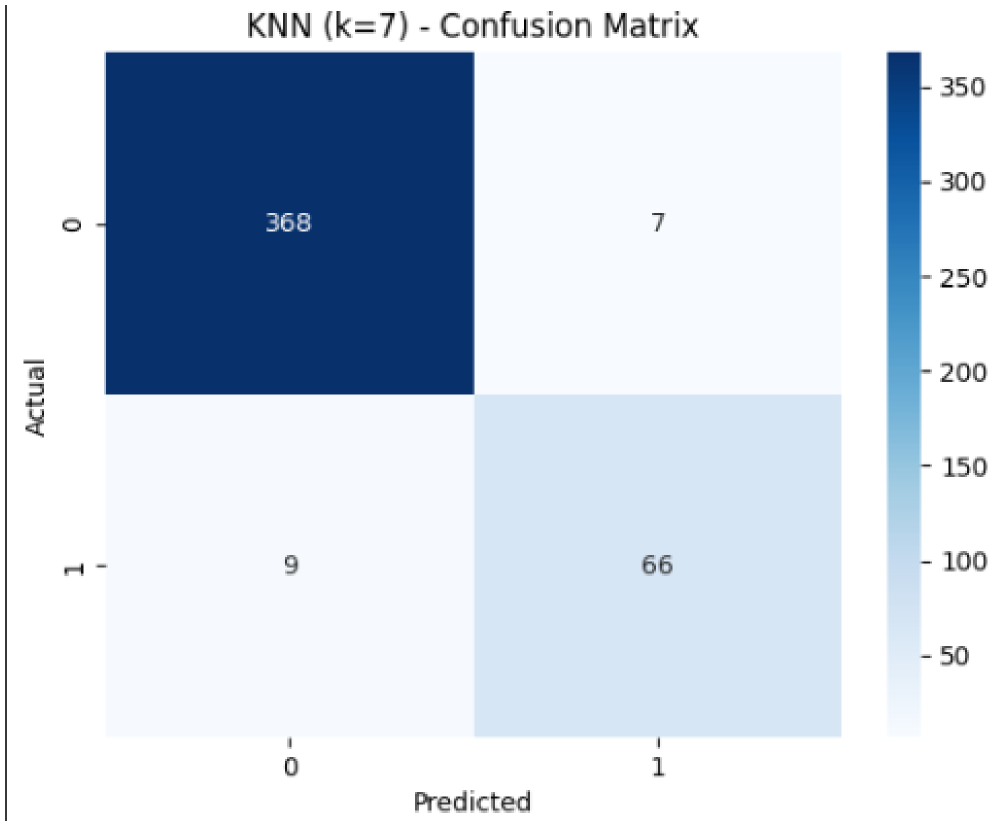

```
KNN (k=5) Evaluation:
```

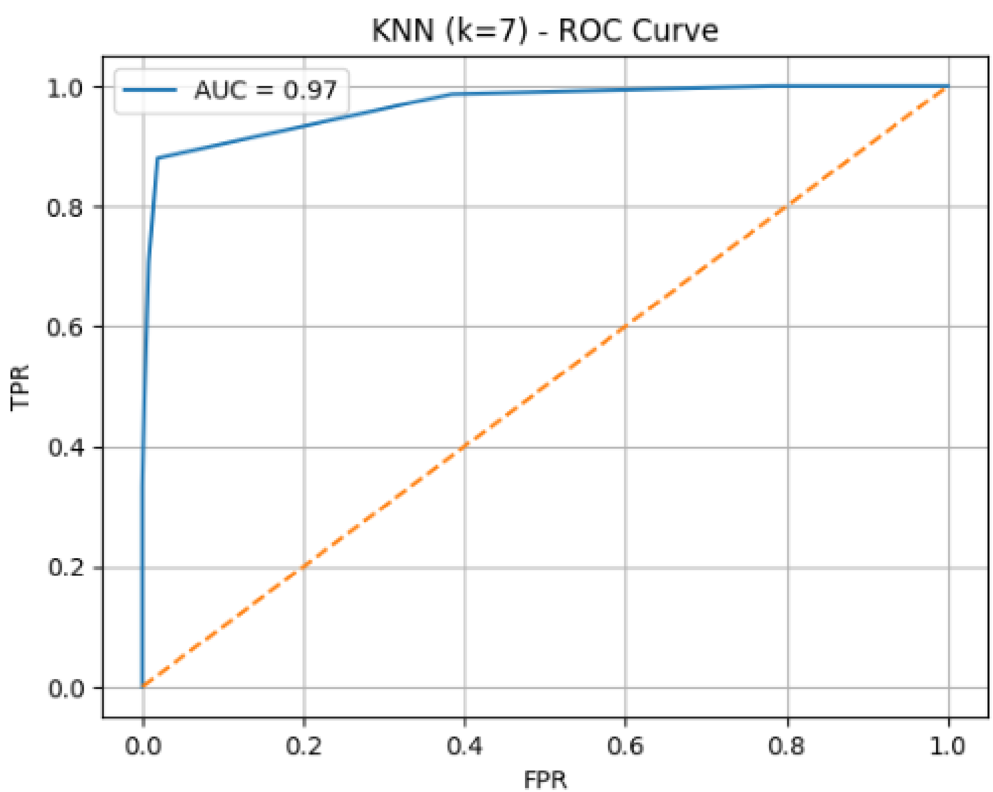
```
Accuracy: 0.8688888888888889
Precision: 0.5645161290322581
Recall: 0.9333333333333333
F1 Score: 0.7035175879396985
F-beta Score (β=0.5): 0.6129597197898424
Matthews Corr Coef: 0.6583958219651456
```





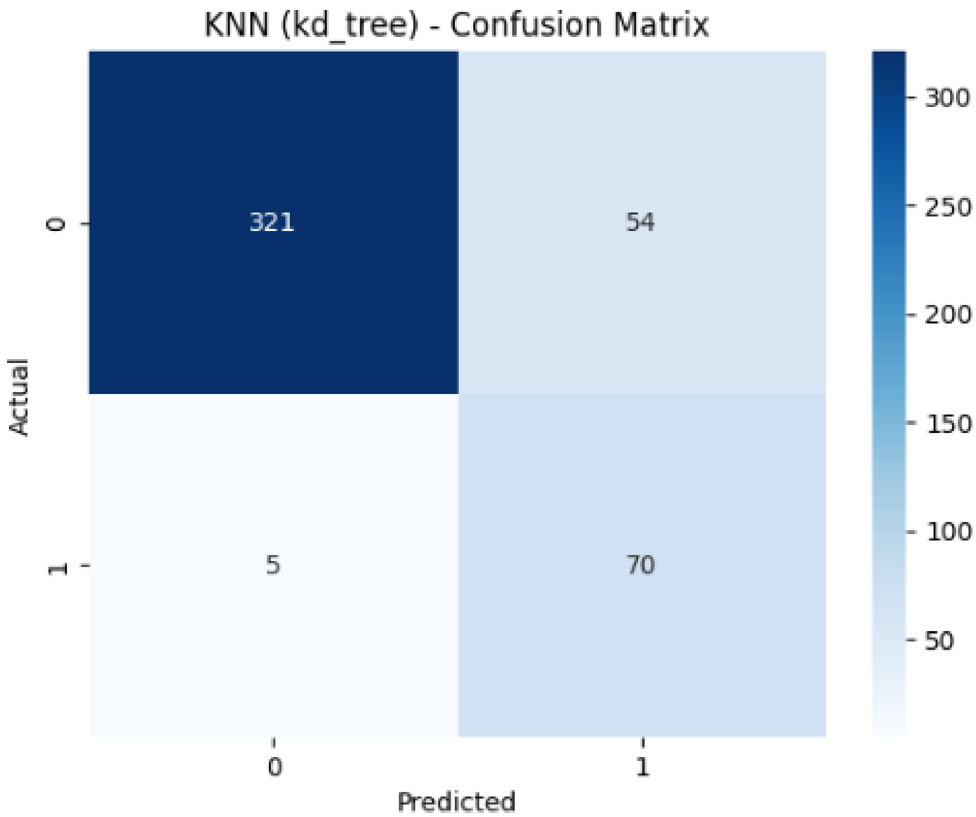

```
KNN (k=7) Evaluation:
Accuracy: 0.9644444444444444
Precision: 0.9041095890410958
Recall: 0.88
F1 Score: 0.8918918918918919
F-beta Score (β=0.5): 0.8991825613079019
Matthews Corr Coef: 0.8707338237428764
```

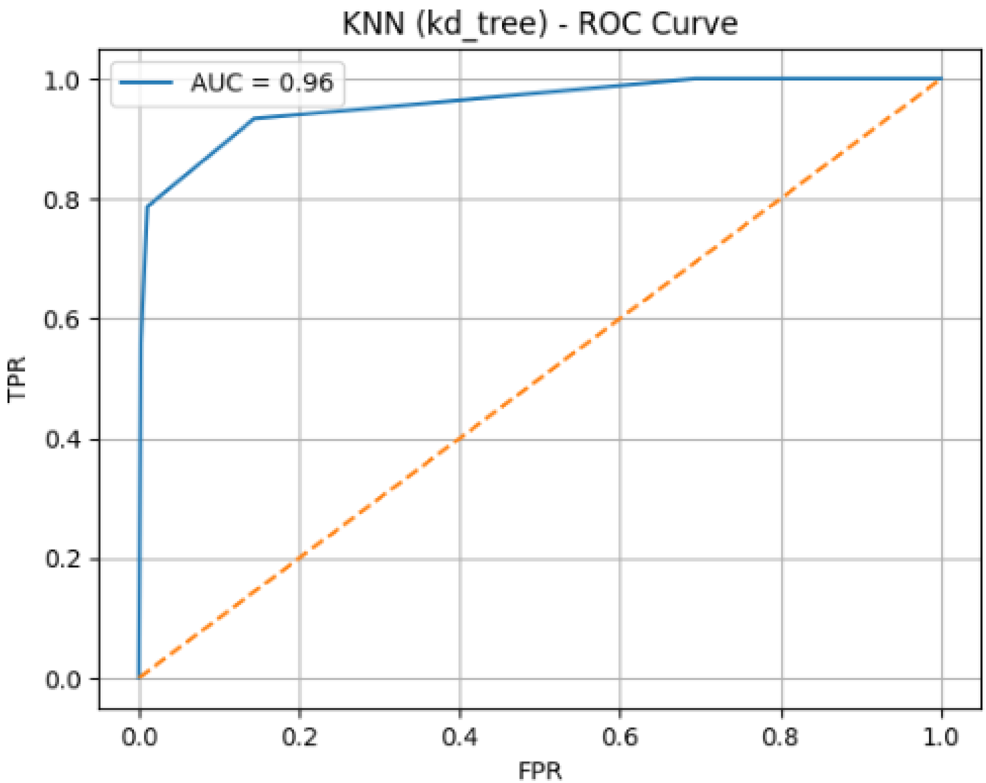




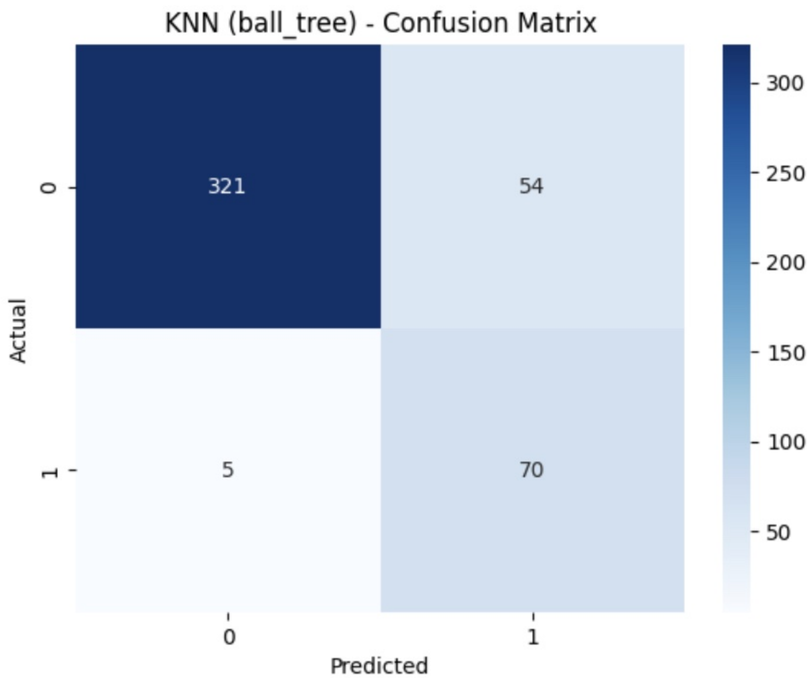

```
KNN (kd_tree) Evaluation:
```

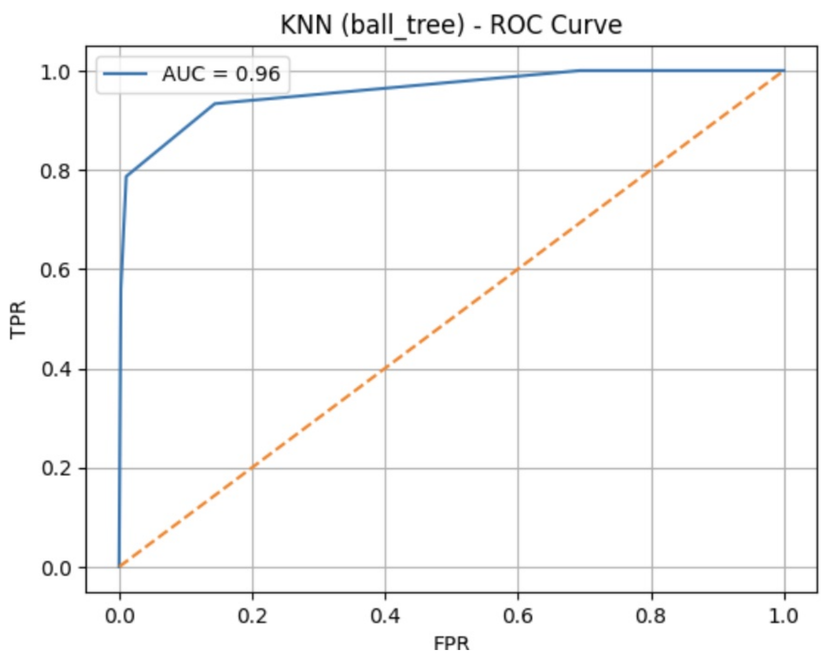
```
Accuracy: 0.8688888888888889
Precision: 0.5645161290322581
Recall: 0.9333333333333333
F1 Score: 0.7035175879396985
F-beta Score (β=0.5): 0.6129597197898424
Matthews Corr Coef: 0.6583958219651456
```






```
KNN (ball_tree) Evaluation:
Accuracy: 0.8688888888888889
Precision: 0.5645161290322581
Recall: 0.9333333333333333
F1 Score: 0.7035175879396985
F-beta Score (β=0.5): 0.6129597197898424
Matthews Corr Coef: 0.6583958219651456
```



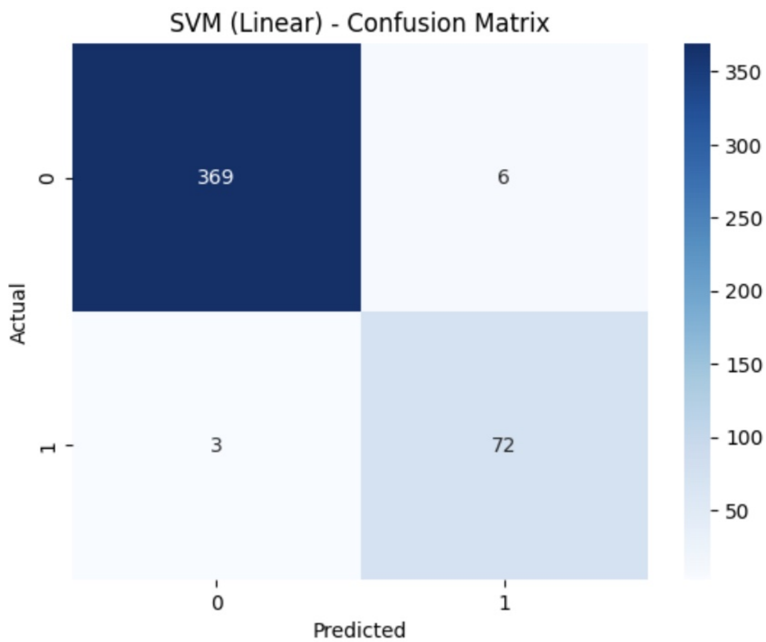


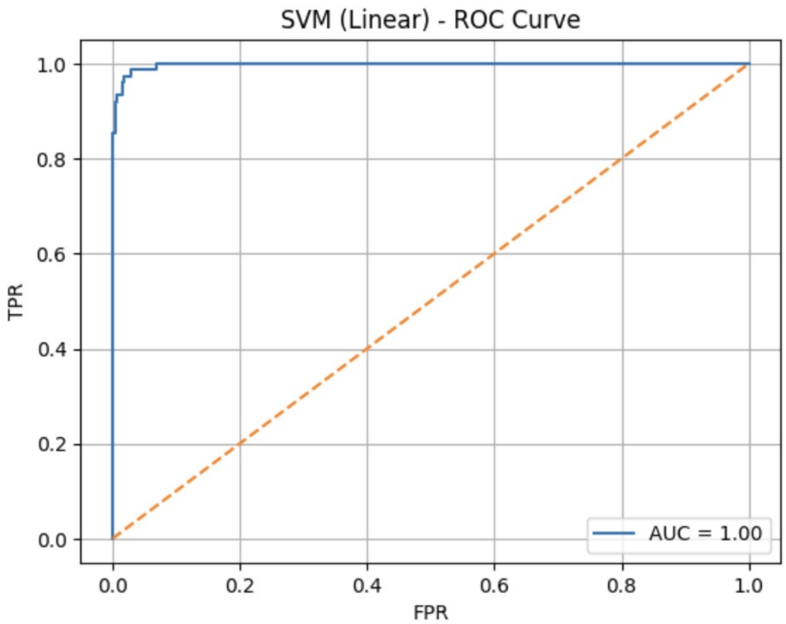

```
# ========== 10. SVM ==========
kernels = {
    "Linear": SVC(kernel='linear', C=1, probability=True),
    "Polynomial": SVC(kernel='poly', C=1, degree=3, gamma='auto', probability=True),
    "RBF": SVC(kernel='rbf', C=1, gamma='scale', probability=True),
    "Sigmoid": SVC(kernel='sigmoid', C=1, gamma='auto', probability=True)
}
for name, model in kernels.items():
```

```
    model.fit(X_train_scaled, y_train_scaled)
    evaluate_model(model, f"SVM ({name})", X_test_scaled, y_test_scaled)
```

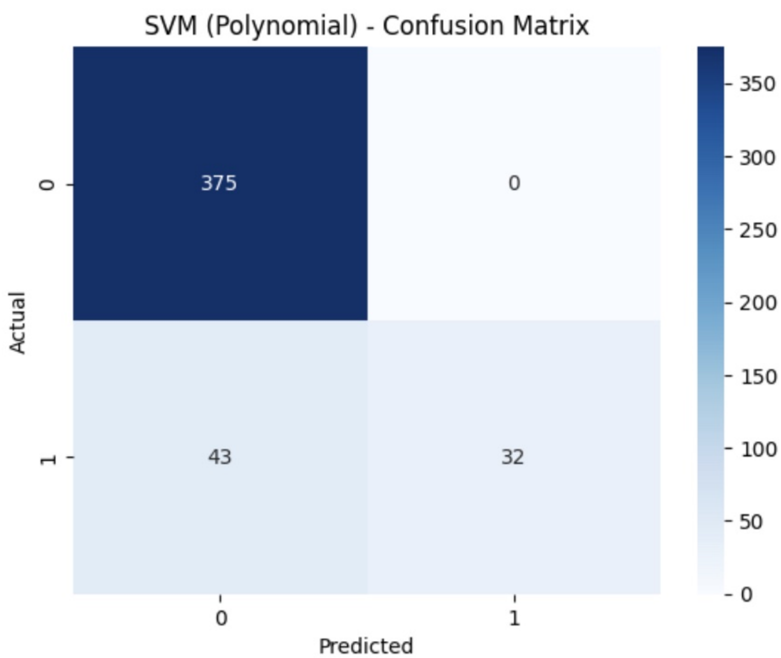
OUTPUT

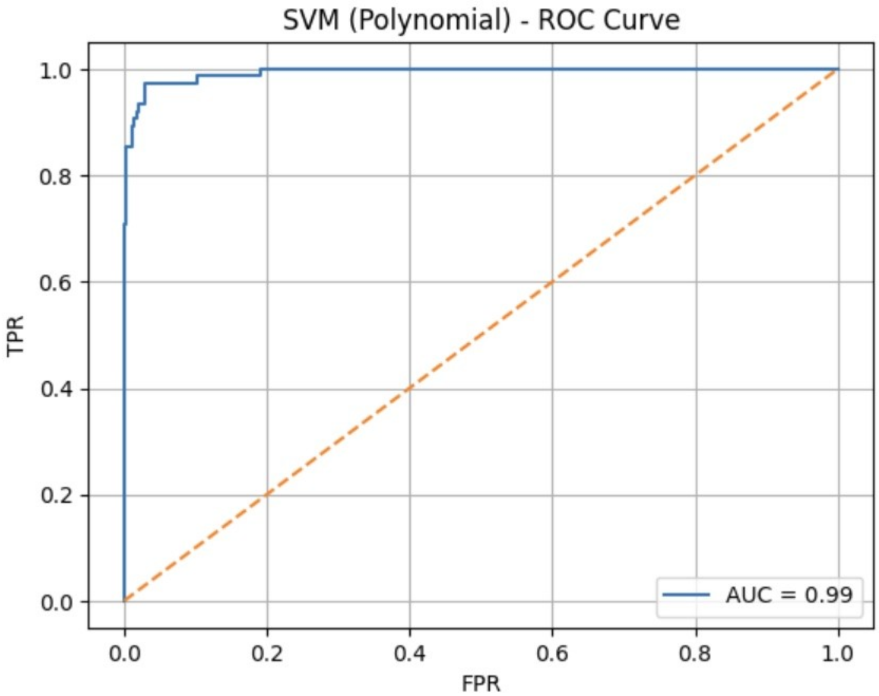
```
SVM (Linear) Evaluation:
Accuracy: 0.98
Precision: 0.9230769230769231
Recall: 0.96
F1 Score: 0.9411764705882353
F-beta Score (β=0.5): 0.9302325581395349
Matthews Corr Coef: 0.9293931956705993
```



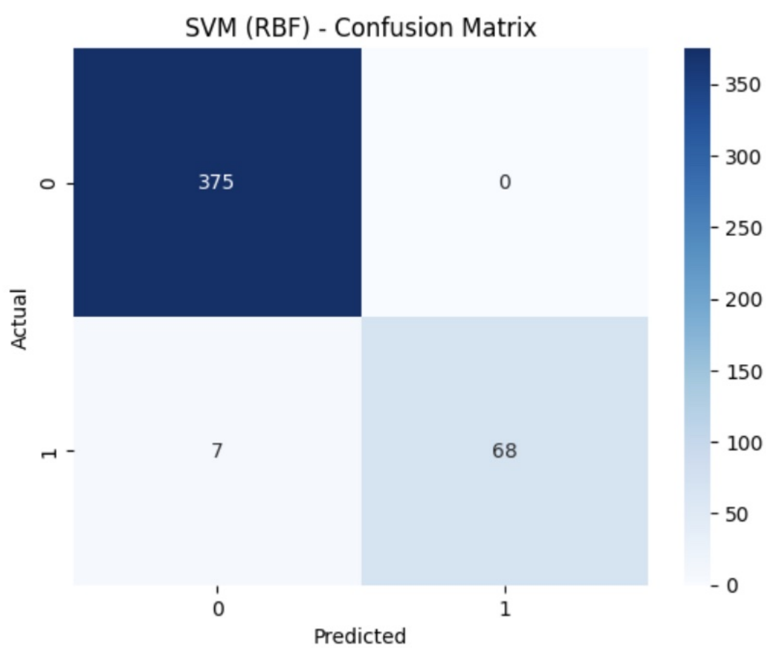


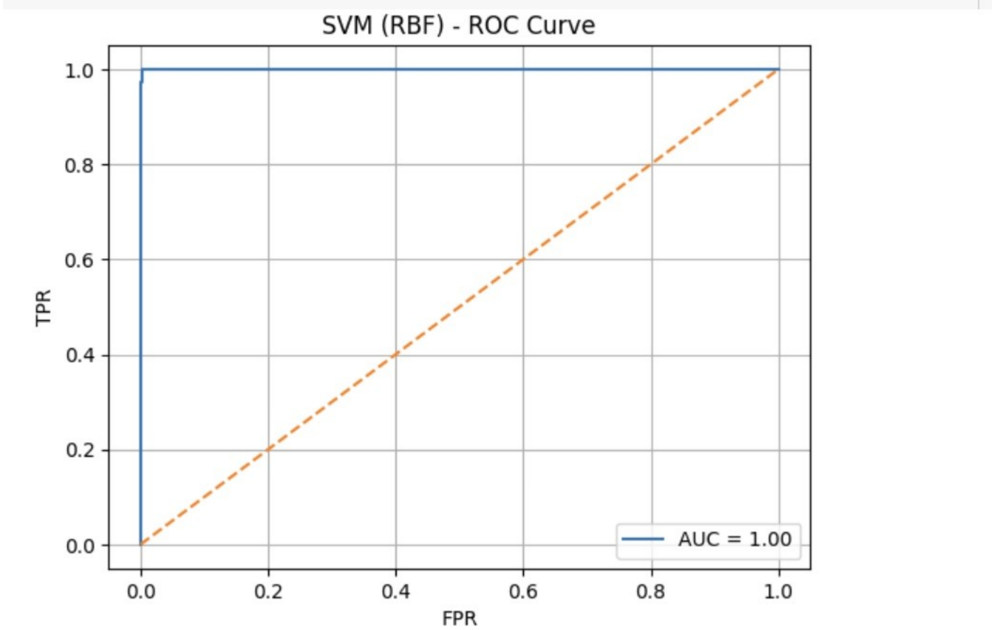

```
SVM (Polynomial) Evaluation:
Accuracy: 0.9044444444444445
Precision: 1.0
Recall: 0.4266666666666667
F1 Score: 0.5981308411214953
F-beta Score (β=0.5): 0.7881773399014779
Matthews Corr Coef: 0.6186882248897461
```



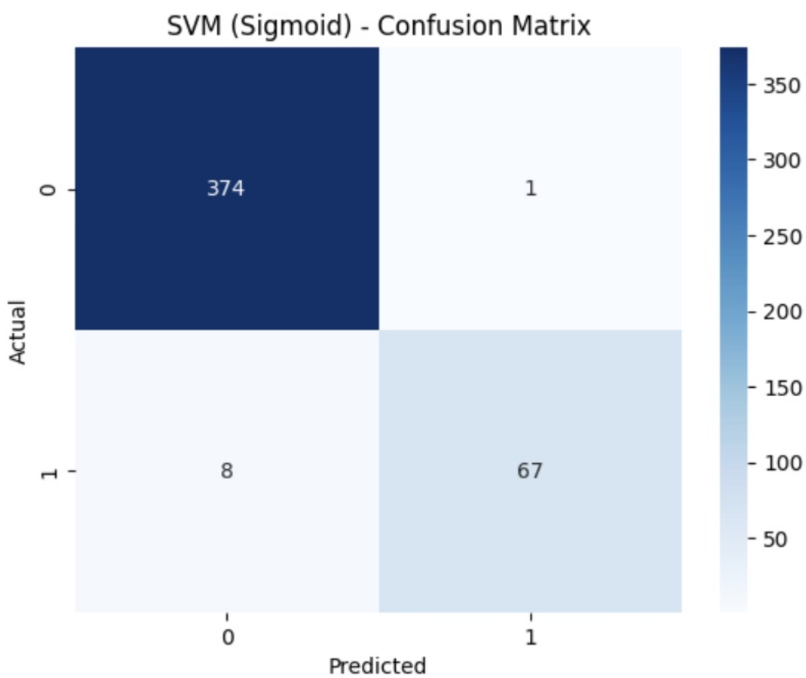


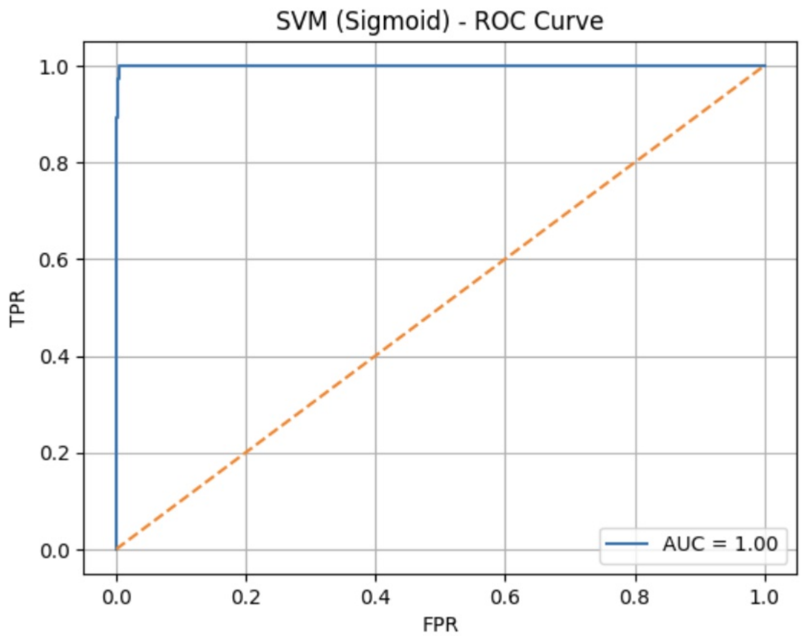

```
SVM (RBF) Evaluation:
Accuracy: 0.9844444444444445
Precision: 1.0
Recall: 0.9066666666666666
F1 Score: 0.951048951048951
F-beta Score (β=0.5): 0.9798270893371758
Matthews Corr Coef: 0.9434258614331825
```






```
SVM (Sigmoid) Evaluation:
Accuracy: 0.98
Precision: 0.9852941176470589
Recall: 0.8933333333333333
F1 Score: 0.9370629370629371
F-beta Score (β=0.5): 0.9654178674351584
Matthews Corr Coef: 0.9267771697608322
```






```
# ========== 11. 5-Fold Cross Validation ==========
# Common CV strategy
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

def evaluate_cv(model, name, X_data, y_data):
    print(f"\n5-Fold Cross Validation: {name}")
    scores = {
        "Accuracy": cross_val_score(model, X_data, y_data, cv=skf, scoring='accuracy').m
        "Precision": cross_val_score(model, X_data, y_data, cv=skf, scoring='precision')
        "Recall": cross_val_score(model, X_data, y_data, cv=skf, scoring='recall').mean(
        "F1 Score": cross_val_score(model, X_data, y_data, cv=skf, scoring='f1').mean()
    }
    for metric, score in scores.items():
        print(f"{metric}: {score:.4f}")

# ===========================
# Naive Bayes
# ===========================
for name, model in {
    "GaussianNB": GaussianNB(),
    "MultinomialNB": MultinomialNB(),
    "BernoulliNB": BernoulliNB()
}.items():
    evaluate_cv(model, name, X, y)

# ===========================
# KNN
```

```
# ===========================
for k in [1, 3, 5, 7]:
    model = KNeighborsClassifier(n_neighbors=k)
    evaluate_cv(model, f"KNN (k={k})", X_scaled, y)

for algo in ["kd_tree", "ball_tree"]:
    model = KNeighborsClassifier(algorithm=algo)
    evaluate_cv(model, f"KNN ({algo})", X_scaled, y)

# ===========================
# SVM Kernels
# ===========================
svm_kernels = {
    "SVM Linear": SVC(kernel='linear', C=1, probability=True),
    "SVM Polynomial": SVC(kernel='poly', C=1, degree=3, gamma='auto', probability=True),
    "SVM RBF": SVC(kernel='rbf', C=1, gamma='scale', probability=True),
    "SVM Sigmoid": SVC(kernel='sigmoid', C=1, gamma='auto', probability=True)
}
for name, model in svm_kernels.items():
    evaluate_cv(model, name, X_scaled, y)
```

OUTPUT

Table 1: Naive Bayes Models (5-Fold Cross Validation)

Model	Accuracy	Precision	Recall	F1 Score
GaussianNB	0.9647	0.9099	0.8758	0.8917
MultinomialNB	0.9770	0.9754	0.8839	0.9270
BernoulliNB	0.9507	0.8618	0.8398	0.8497

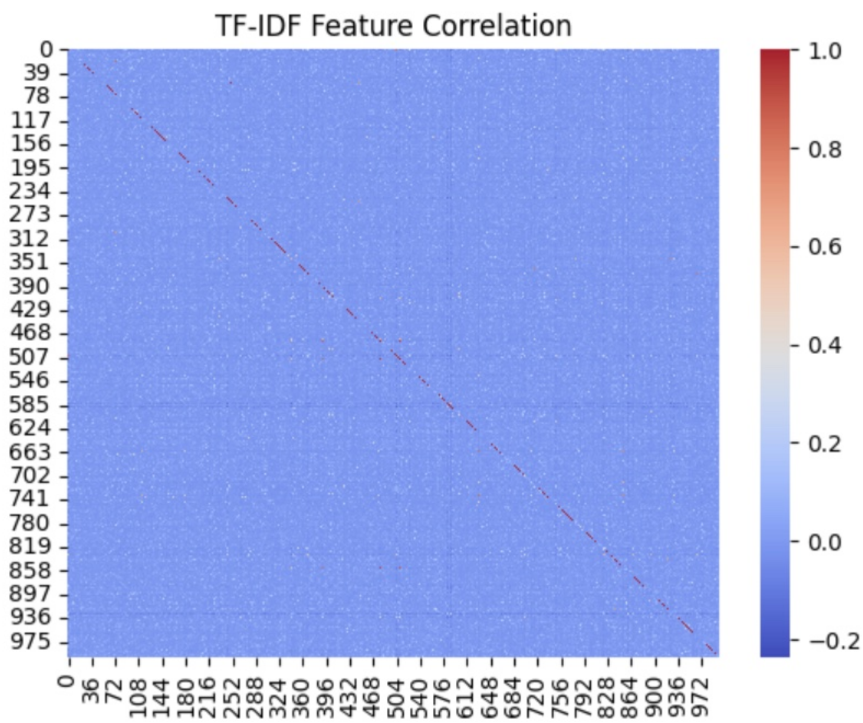
Table 2: KNN Models (5-Fold Cross Validation)

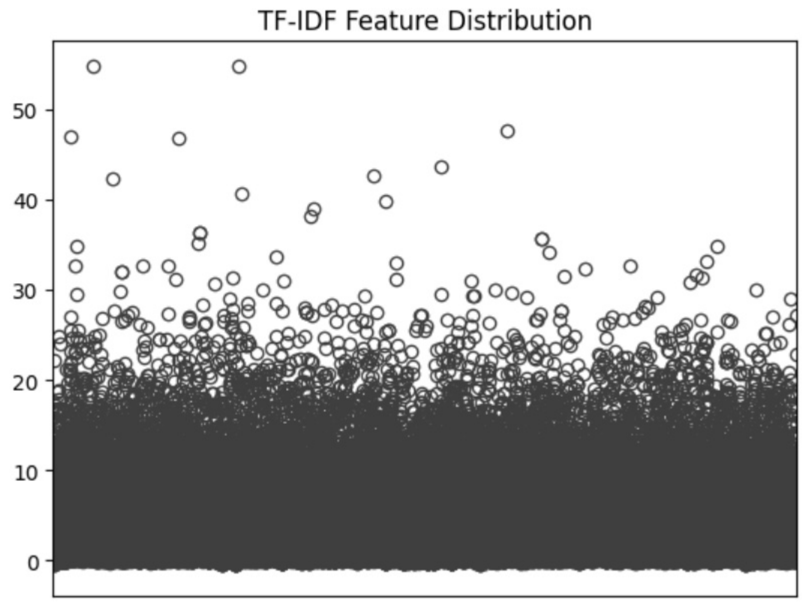
Model	Accuracy	Precision	Recall	F1 Score
KNN (k=1)	0.8056	0.4569	0.8858	0.6025
KNN (k=3)	0.7619	0.4659	0.9139	0.5922
KNN (k=5)	0.9196	0.7654	0.8178	0.7759
KNN (k=7)	0.9423	0.9380	0.7036	0.8001
KNN (kd_tree)	0.9193	0.7646	0.8178	0.7754
KNN (ball_tree)	0.9196	0.7654	0.8178	0.7759

Table 3: SVM Models (5-Fold Cross Validation)

Model	Accuracy	Precision	Recall	F1 Score
SVM Linear	0.9660	0.8714	0.9339	0.9015
SVM Polynomial	0.9053	1.0000	0.4311	0.5996
SVM RBF	0.9733	0.9977	0.8418	0.9125
SVM Sigmoid	0.9753	0.9735	0.8759	0.9215

```
# ========== 12. Visualizations (Scaled TF-IDF) ==========
sns.heatmap(pd.DataFrame(X_scaled).corr(), cmap='coolwarm')
plt.title("TF-IDF Feature Correlation")
plt.show()

sns.boxplot(data=pd.DataFrame(X_scaled))
plt.xticks([])
plt.title("TF-IDF Feature Distribution")
plt.show()
```





Observation:

- The scatter plot of *Actual vs Predicted Loan Amount* shows that the predictions closely follow the ideal $y = x$ line, indicating that the model captures the trend of the data effectively.
- The residual plot indicates that most residuals are centered around zero, with no strong non-linear patterns, validating the suitability of linear regression for this dataset.
- Feature importance analysis (model coefficients) reveals that certain features such as income and credit score have the most significant influence on loan amount prediction.

Inference:

- The linear regression model demonstrates good predictive performance with relatively small residuals, meaning it can reliably estimate sanctioned loan amounts.
- The presence of both positive and negative coefficients suggests that some features increase the sanctioned loan amount while others reduce it.
- Since residuals show random distribution, assumptions of linear regression (linearity, homoscedasticity) are reasonably satisfied.

Learning outcomes:

- Gained hands-on experience in applying `scikit-learn` for building a linear regression model.

- Understood the importance of exploratory data analysis (EDA) and visualization in evaluating model performance.
- Learned to interpret regression coefficients and residual plots to validate model assumptions.
- Acquired skills in comparing predicted vs actual outcomes and deriving insights using statistical and graphical metrics.