IDEAS - Institute of Data Engineering, Analytics and Science Foundation
Autumn Internship Program 2025
Project Notebook

Diabetes Prediction: Classification Comparison + Metrics + Evaluation

Titanic survival Prediction: Classification Comparison + Metrics + Evaluation

Student Name: Bipasha Das

Course: B.Sc Statistics

Institute Name: Sister Nibedita Government General Degree Collage for Girls'

Period of Internship: 25th August 2025- 19th September 2025

Report submitted to: IDEAS-Institute of Data Engineering, Analytics and Science Foundation, ISI Kolkata

Designation:

This notebook is structured as an **assignment**. The goal is to build a machine learning workflow for predicting diabetes, compare models, and evaluate them using metrics.

The structure is provided, but you are expected to fill in the details.

Problem Statement

You are tasked with building a classification model to predict whether a patient has diabetes based on diagnostic measurements.

- Use the **Pima Indian Diabetes Dataset**.
- Compare multiple classification models.
- Evaluate them using accuracy, precision, recall, F1, ROC-AUC.
- Extend the workflow to a new dataset of your choice.

Dataset Introduction

The dataset contains medical predictor variables and one target variable (`Outcome`).

- Pregnancies
- Glucose
- Blood Pressure
- Skin Thickness
- Insulin
- BMI
- DiabetesPedigreeFunction
- Age
- Outcome (0 = No Diabetes, 1 = Diabetes)

```
#Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, roc_auc_score, confusion_matrix,
classification_report
```

Data Loading

```
# Load dataset
url =
'https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv
'
```

```
df = pd.read_csv(url)
df.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Exploratory Data Analysis (EDA)

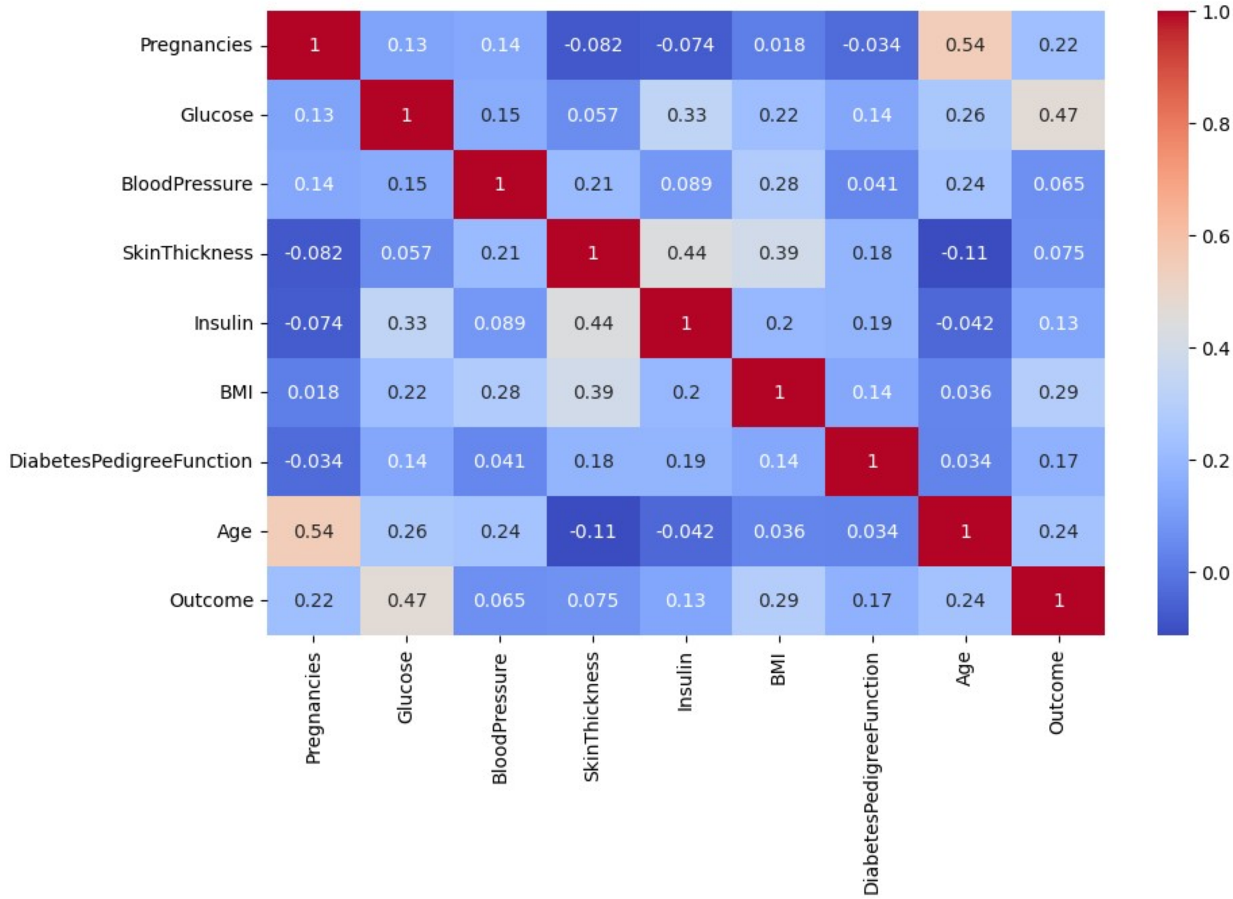
- Check dataset shape
- Missing values
- Basic statistics
- Correlation heatmap
- Distribution plots

```
# Basic EDA
print(df.shape)
print(df.info())
df.describe()
```

```
(768, 9)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
```

```
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None

       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  768.000000     768.000000     768.000000
768.000000
mean      3.845052  120.894531      69.105469      20.536458
79.799479
std       3.369578   31.972618      19.355807      15.952218
115.244002
min       0.000000    0.000000       0.000000       0.000000
0.000000
25%       1.000000   99.000000      62.000000       0.000000
0.000000
50%       3.000000  117.000000      72.000000      23.000000
30.500000
75%       6.000000  140.250000      80.000000      32.000000
127.250000
max      17.000000  199.000000     122.000000      99.000000
846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

```python
# Correlation heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```



Data Preprocessing & Train/Test Split

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
print(X_train.shape, X_test.shape)
```

```
(614, 8) (154, 8)
```

Data Scaling

```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Machine Learning Models

KNN Classifier

```
# KNN Model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)

print("KNN Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

```
KNN Results:
Accuracy: 0.7012987012987013
[[80 20]
 [26 28]]
              precision    recall  f1-score   support

           0       0.75      0.80      0.78       100
           1       0.58      0.52      0.55        54

    accuracy                           0.70       154
   macro avg       0.67      0.66      0.66       154
weighted avg       0.69      0.70      0.70       154
```

Support Vector Machine

```
# SVM Model
svm = SVC(kernel="linear", random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred_svm = svm.predict(X_test_scaled)

print("SVM Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print(confusion_matrix(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

```
SVM Results:
Accuracy: 0.7207792207792207
[[83 17]
 [26 28]]
              precision    recall  f1-score   support
```

```
           0       0.76      0.83      0.79       100
           1       0.62      0.52      0.57        54

    accuracy                           0.72       154
   macro avg       0.69      0.67      0.68       154
weighted avg       0.71      0.72      0.71       154
```

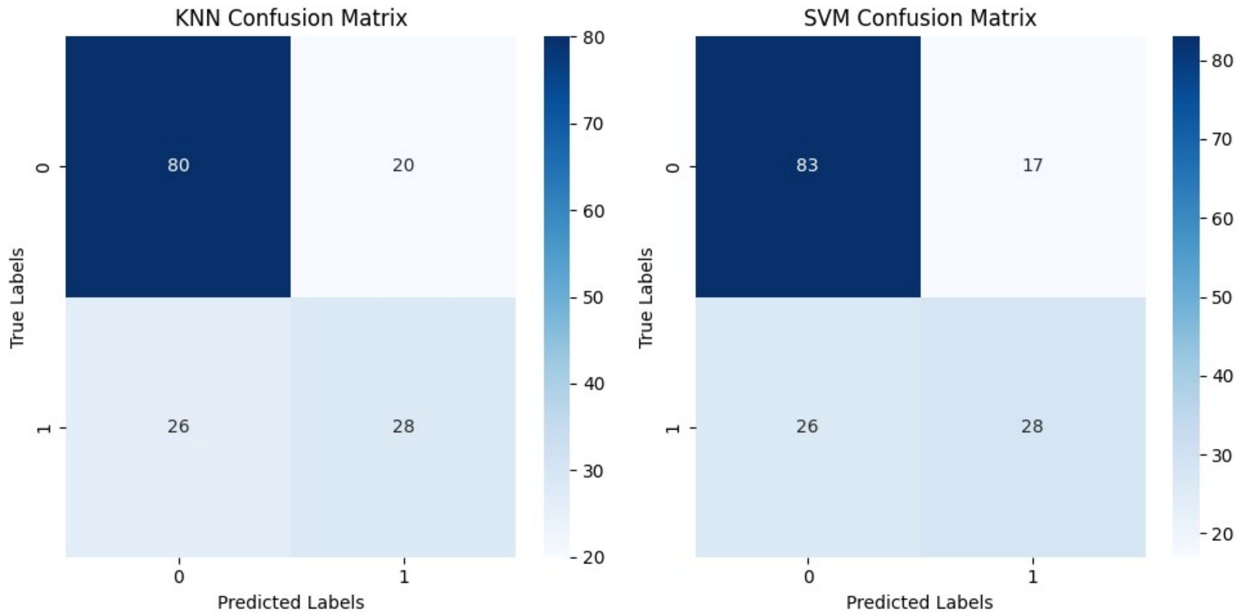
Evaluation

Compare the models using metrics and visualize results:

- Confusion matrices
- ROC curves
- Metric comparison table

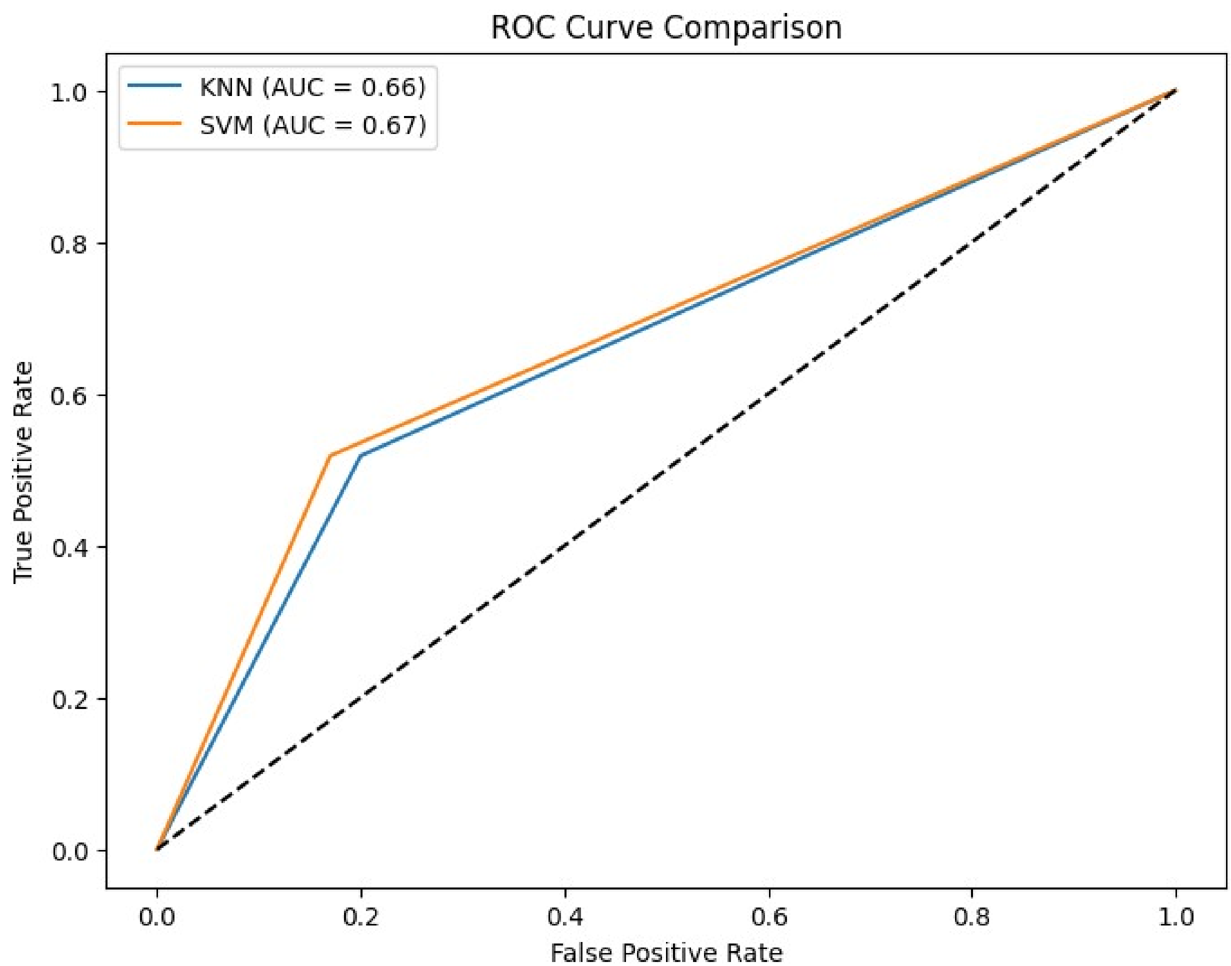
Confusion Matrices

```
#Confusion matrices
cm_knn = confusion_matrix(y_test, y_pred_knn)
cm_svm = confusion_matrix(y_test, y_pred_svm)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
sns.heatmap(cm_knn, annot=True, cmap='Blues')
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')

plt.subplot(1, 2, 2)
sns.heatmap(cm_svm, annot=True, cmap='Blues')
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')

plt.tight_layout()
plt.show()
```



ROC curves

```
#ROC curve
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_pred_knn)
auc_knn = roc_auc_score(y_test, y_pred_knn)
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_pred_svm)
auc_svm = roc_auc_score(y_test, y_pred_svm)
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, label=f'KNN (AUC = {auc_knn:.2f})')
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {auc_svm:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend()
plt.show()
```



Metric comparison table

```
#Metric comparison table
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn)
recall_knn = recall_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)
auc_knn = roc_auc_score(y_test, y_pred_knn)  # Assuming
y_pred_knn_proba is the probability output

# Calculate metrics for SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
auc_svm = roc_auc_score(y_test, y_pred_svm)  # Assuming
y_pred_svm_proba is the probability output

# Create a comparison table
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score', 'AUC-ROC']
```

```
knn_values = [accuracy_knn, precision_knn, recall_knn, f1_knn,
auc_knn]
svm_values = [accuracy_svm, precision_svm, recall_svm, f1_svm,
auc_svm]

comparison_table = pd.DataFrame({
    'Metric': metrics,
    'KNN': knn_values,
    'SVM': svm_values
})

print(comparison_table)
```

```
      Metric       KNN       SVM
0   Accuracy  0.701299  0.720779
1  Precision  0.583333  0.622222
2     Recall  0.518519  0.518519
3   F1-score  0.549020  0.565657
4    AUC-ROC  0.659259  0.674259
```

Apply Workflow on Another Dataset

Repeat the same steps on a dataset of your choice (e.g., Breast Cancer, Titanic, etc.).

Problem Statement

You are tasked with building a classification model to predict whether a passenger has survived based on the measurements.

- Use the **Titanic Dataset**.
- Compare multiple classification models.
- Evaluate them using accuracy, precision, recall, F1, ROC-AUC.

Dataset Introduction

The dataset contains survival predictor variables, one target variable (`survived`) and other details.

- PassengerId
- Survived (0 = not Survived, 1 = Survived)
- Pclass
- Name
- Sex
- Age
- SibSp
- Parch

- Ticket
- Fare
- Cabin
- Embarked

```
#Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, roc_auc_score, confusion_matrix,
classification_report
```

Data Loading

```
# Load dataset
url =
'https://raw.githubusercontent.com/pandas-dev/pandas/master/doc/data/
titanic.csv'
df = pd.read_csv(url)
df.head()

   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
```

```
1      0          PC 17599  71.2833   C85        C
2      0   STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

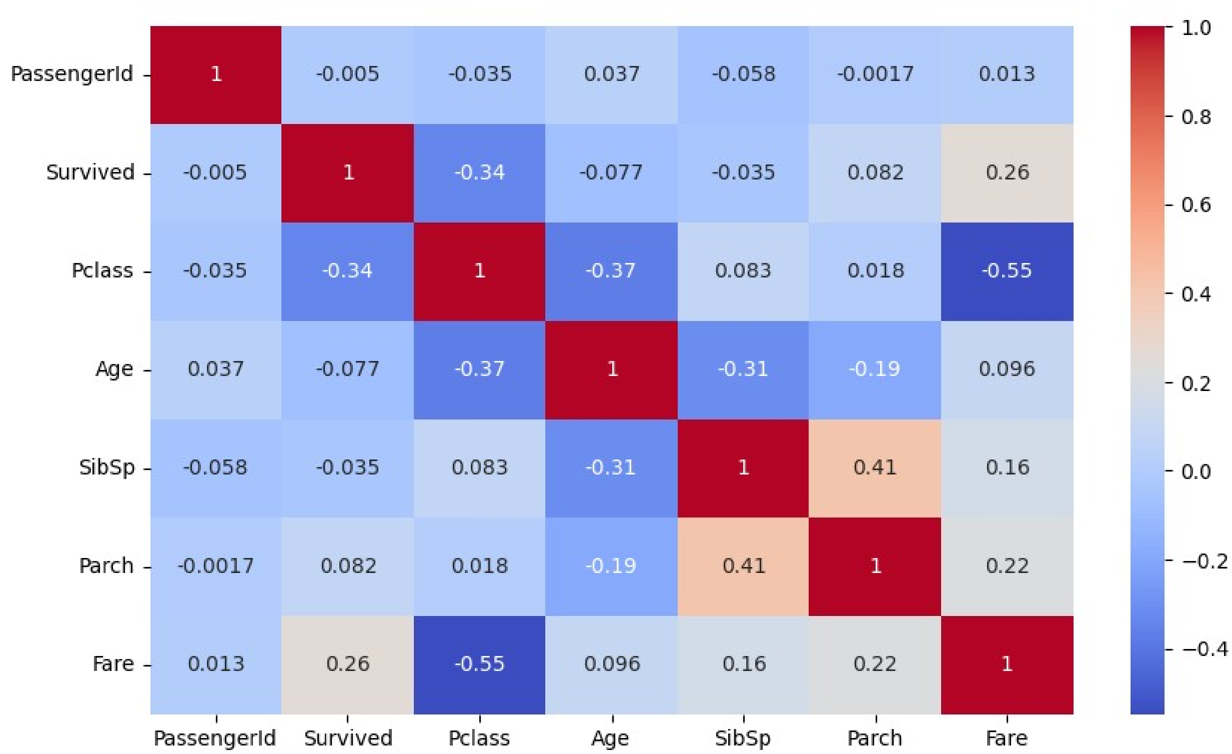
Exploratory Data Analysis (EDA)

- Check dataset shape
- Missing values
- Basic statistics
- Correlation heatmap
- Distribution plots

```python
# Basic EDA
print(df.shape)
print(df.info())
df.describe()
```

```
(891, 12)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000
```

```
       Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200

# Correlation heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df.select_dtypes(include=['float64',
'int64']).corr(),annot=True, cmap='coolwarm')
plt.show()
```



Data Preprocessing & Train/Test Split

```
X = df.drop(['Survived', 'Name','Sex', 'Ticket', 'Cabin', 'Embarked'],
axis= 1).dropna(axis=0)
y = df['Survived'].loc[X.index]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
print(X_train.shape, X_test.shape)

(571, 6) (143, 6)
```

Data Scaling

```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Machine Learning Models

KNN Classifier

```
# KNN Model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)
print("KNN Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

```
KNN Results:
Accuracy: 0.6503496503496503
[[61 24]
 [26 32]]
              precision    recall  f1-score   support

           0       0.70      0.72      0.71        85
           1       0.57      0.55      0.56        58

    accuracy                           0.65       143
   macro avg       0.64      0.63      0.64       143
weighted avg       0.65      0.65      0.65       143
```

Support Vector Machine

```
# SVM Model
svm = SVC(kernel="linear",random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred_svm = svm.predict(X_test_scaled)
print("SVM Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print(confusion_matrix(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_knn))
```

```
SVM Results:
Accuracy: 0.7062937062937062
[[71 14]
 [28 30]]
              precision    recall  f1-score   support
```

```
              0       0.70      0.72      0.71        85
              1       0.57      0.55      0.56        58

       accuracy                           0.65       143
      macro avg       0.64      0.63      0.64       143
   weighted avg       0.65      0.65      0.65       143
```

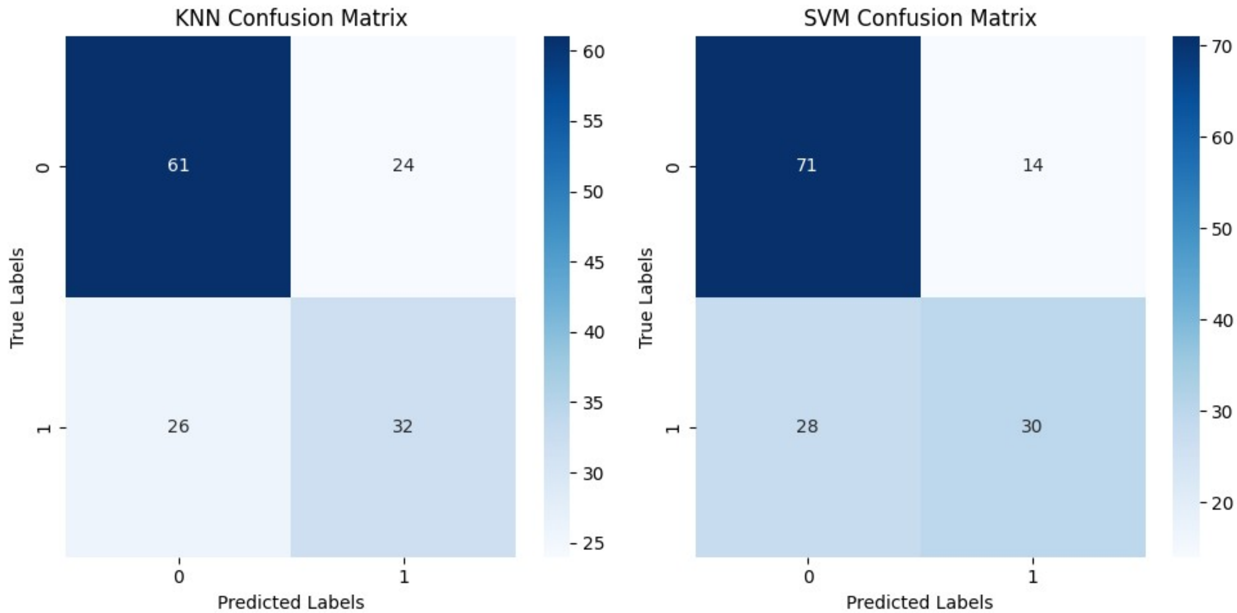
Evaluation

Compare the models using metrics and visualize results:

- Confusion matrices
- ROC curves
- Metric comparison table

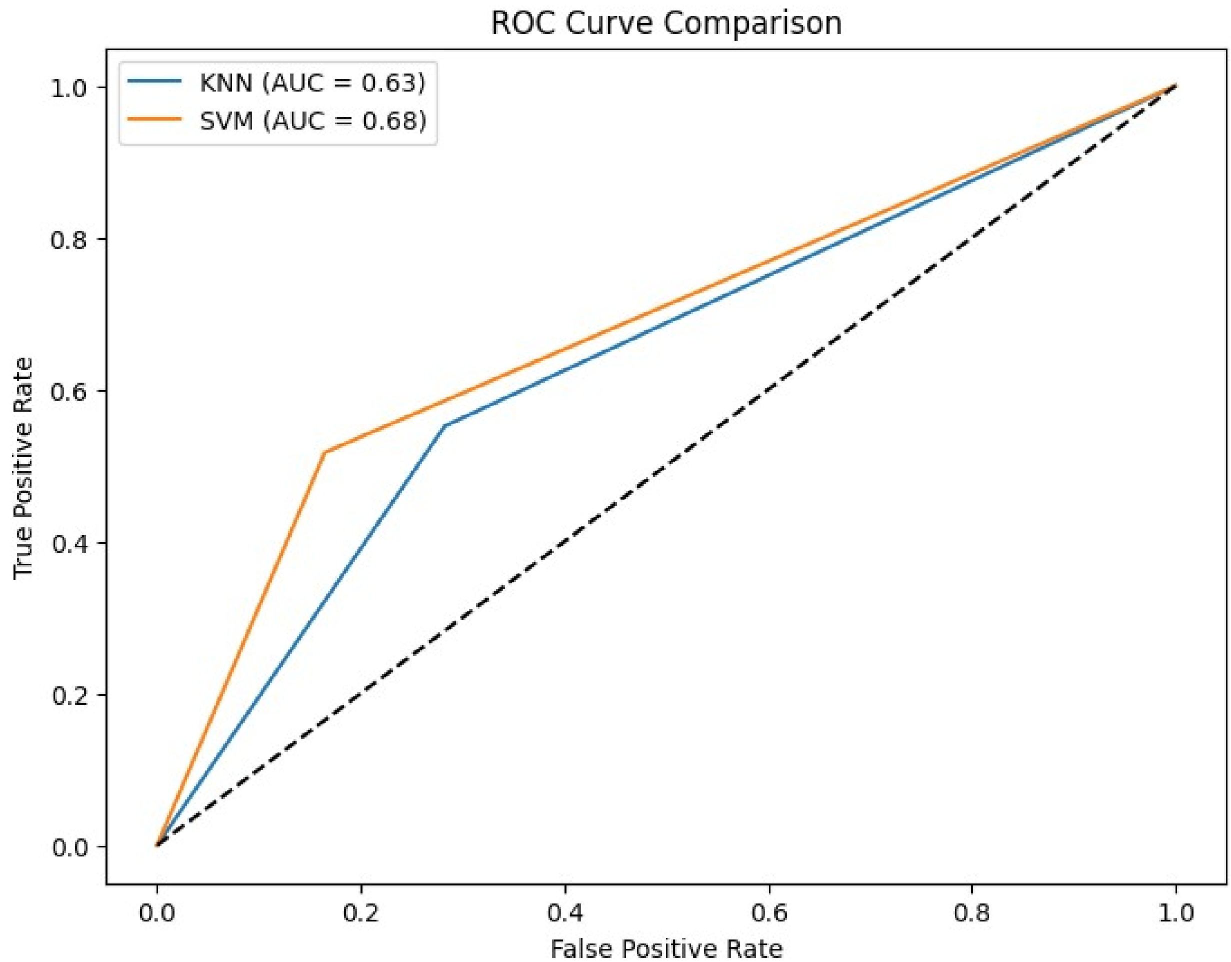
Confusion matrices

```
#Confusion Matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)
cm_svm = confusion_matrix(y_test, y_pred_svm)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
sns.heatmap(cm_knn, annot=True, cmap='Blues')
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')

plt.subplot(1, 2, 2)
sns.heatmap(cm_svm, annot=True, cmap='Blues')
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')

plt.tight_layout()
plt.show()
```



ROC curve

```
#ROC curve
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_pred_knn)
auc_knn = roc_auc_score(y_test, y_pred_knn)
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_pred_svm)
auc_svm = roc_auc_score(y_test, y_pred_svm)
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, label=f'KNN (AUC = {auc_knn:.2f})')
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {auc_svm:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend()
plt.show()
```



Metric comparison table

```
#Metric comparison table
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn)
recall_knn = recall_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)
auc_knn = roc_auc_score(y_test, y_pred_knn)  # Assuming
y_pred_knn_proba is the probability output

# Calculate metrics for SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
auc_svm = roc_auc_score(y_test, y_pred_svm)  # Assuming
y_pred_svm_proba is the probability output

# Create a comparison table
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score', 'AUC-ROC']
```

```python
knn_values = [accuracy_knn, precision_knn, recall_knn, f1_knn,
auc_knn]
svm_values = [accuracy_svm, precision_svm, recall_svm, f1_svm,
auc_svm]

comparison_table = pd.DataFrame({
    'Metric': metrics,
    'KNN': knn_values,
    'SVM': svm_values
})
print(comparison_table)
```

```
      Metric       KNN       SVM
0   Accuracy  0.650350  0.706294
1  Precision  0.571429  0.681818
2     Recall  0.551724  0.517241
3   F1-score  0.561404  0.588235
4    AUC-ROC  0.634686  0.676268
```

Conclusion

Summarize the findings:

- Which model performed best?
- Trade-offs between metrics
- Generalizability of the workflow

Which model performed best?

Considering both prediction (Diabetes and Titatic Survival), we can say that SVM Model performed better than KNN Model overall based on metrics like Accuracy, Precision, Recall, F1-score, AUC-ROC.

Trade-offs between metrics

SVM has higher Accuracy, Precision, F1-score and AUC-ROC ,but in Titanic Survival Prediction SVM has slightly lower Recall compared to KNN. KNN might have more false positive. Overall, we can say that SVM is more precise and overall performs better.

Generalizibility of the Workflow

The workflow of building classification models to predict survival in the Titanic dataset, comparing multiple models, and evaluating them using accuracy, precision, recall, F1-score, and ROC-AUC is generalizable to other classification problems. We can apply this approach to different datasets with a binary target variable and predictor variables. This method helps in selecting the best-performing model based on the metrics that matter most for our specific problem.