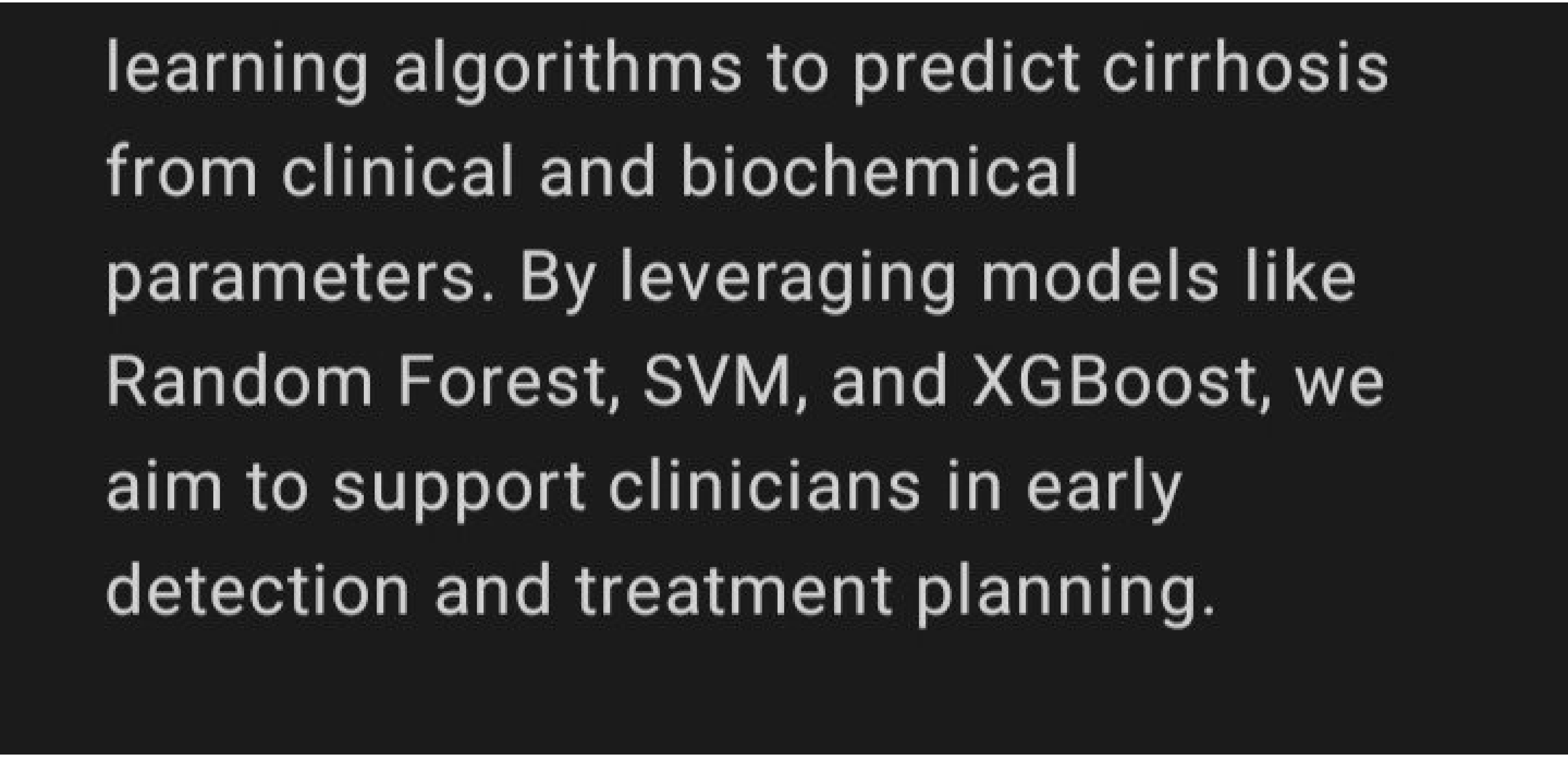
✅ Project Title:

Revolutionizing Liver Care: Predicting Cirrhosis Using Advanced Machine Learning Techniques

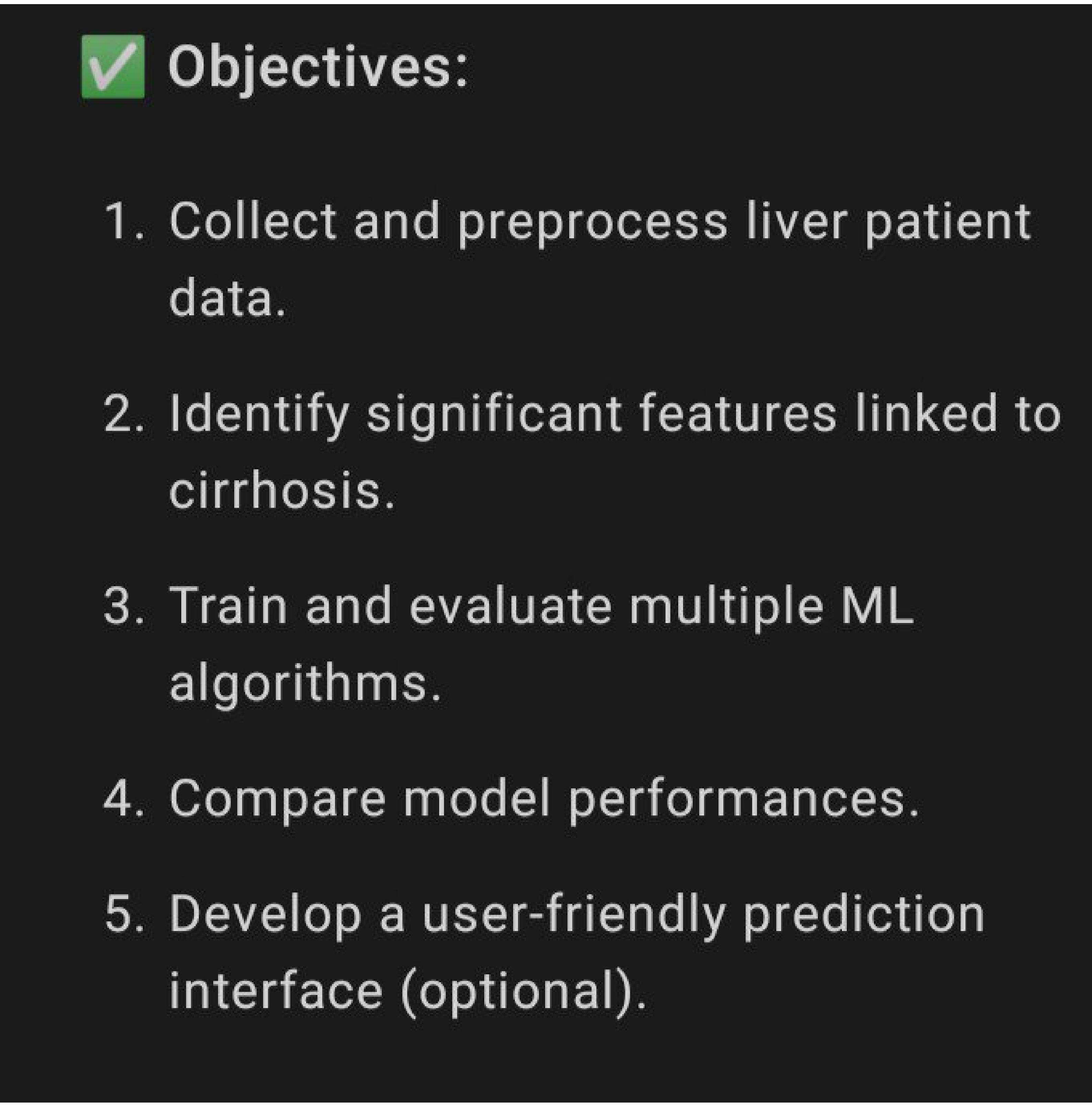
---

✅ Abstract:

Cirrhosis is a chronic liver disease characterized by fibrosis and impaired liver function. Early detection is crucial to prevent severe complications such as liver failure, portal hypertension, and hepatocellular carcinoma. Traditional diagnosis methods often rely on invasive procedures or late-stage symptoms. This project proposes a non-invasive, data-driven solution using machine learning algorithms to predict cirrhosis

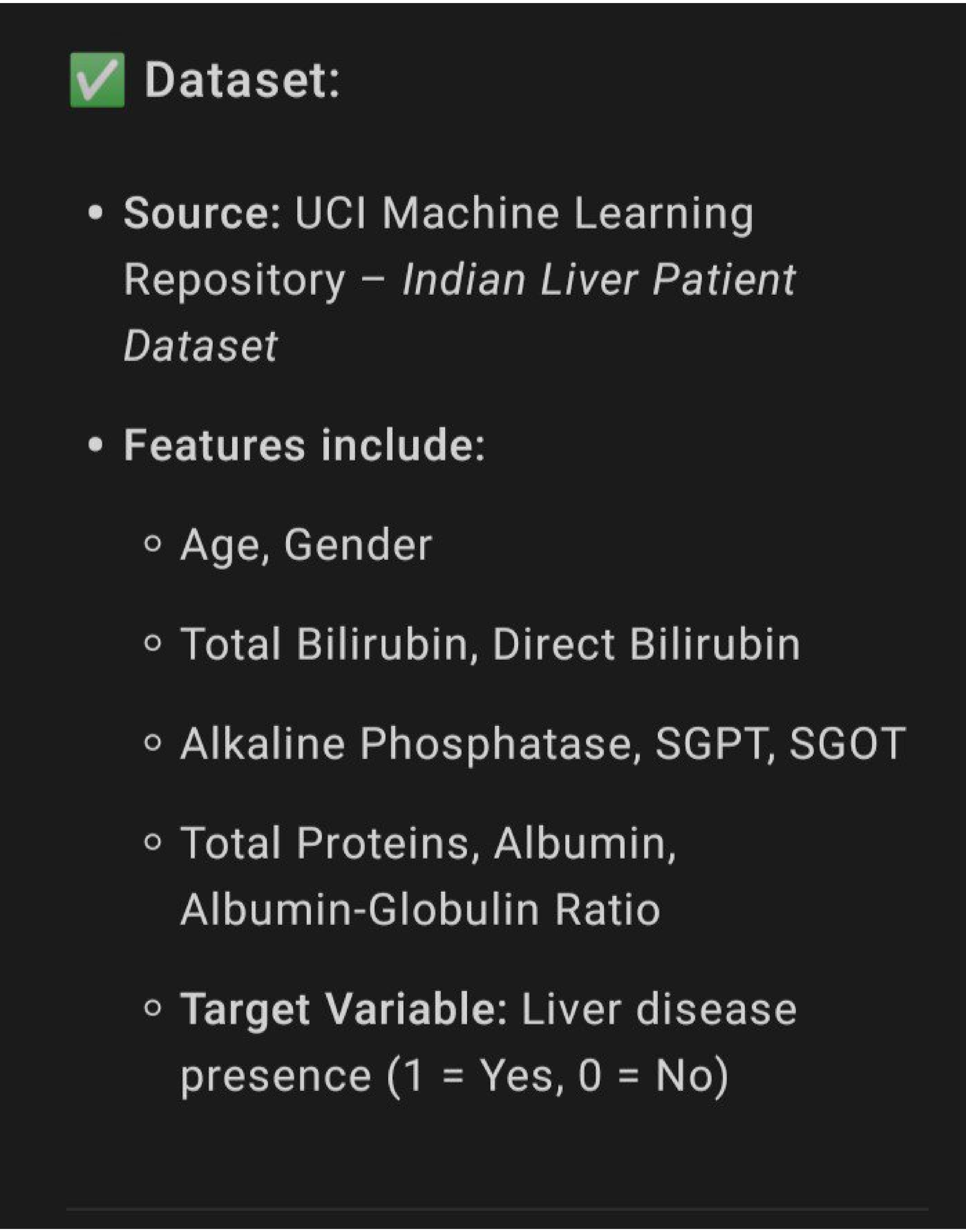


learning algorithms to predict cirrhosis from clinical and biochemical parameters. By leveraging models like Random Forest, SVM, and XGBoost, we aim to support clinicians in early detection and treatment planning.



✅ Objectives:

1. Collect and preprocess liver patient data.
2. Identify significant features linked to cirrhosis.
3. Train and evaluate multiple ML algorithms.
4. Compare model performances.
5. Develop a user-friendly prediction interface (optional).



✅ Dataset:

- **Source:** UCI Machine Learning Repository – *Indian Liver Patient Dataset*
- **Features include:**
  - Age, Gender
  - Total Bilirubin, Direct Bilirubin
  - Alkaline Phosphatase, SGPT, SGOT
  - Total Proteins, Albumin, Albumin-Globulin Ratio
  - **Target Variable:** Liver disease presence (1 = Yes, 0 = No)

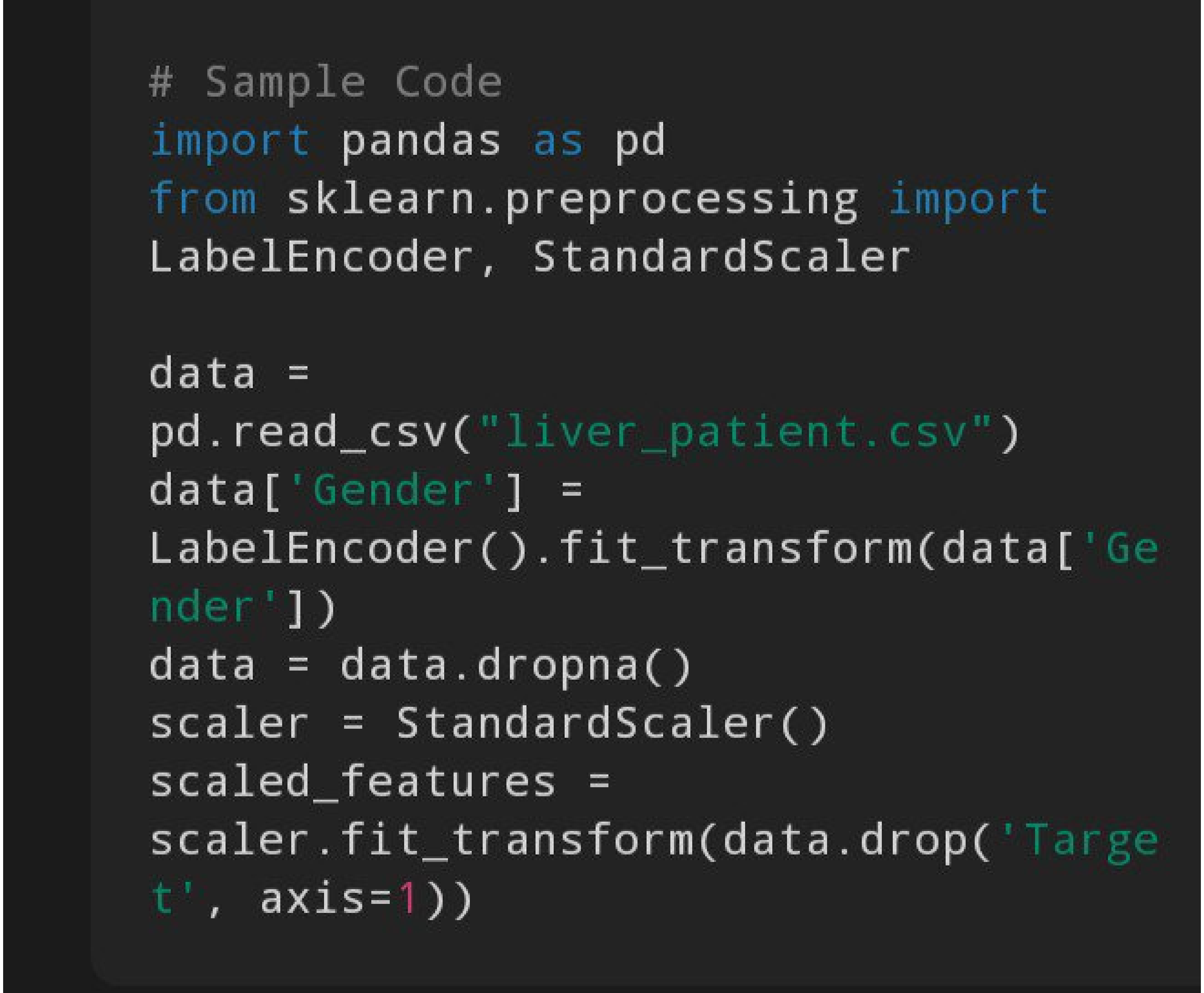
✅ Methodology / Milestones:

🔹 **Milestone 1: Problem Understanding**

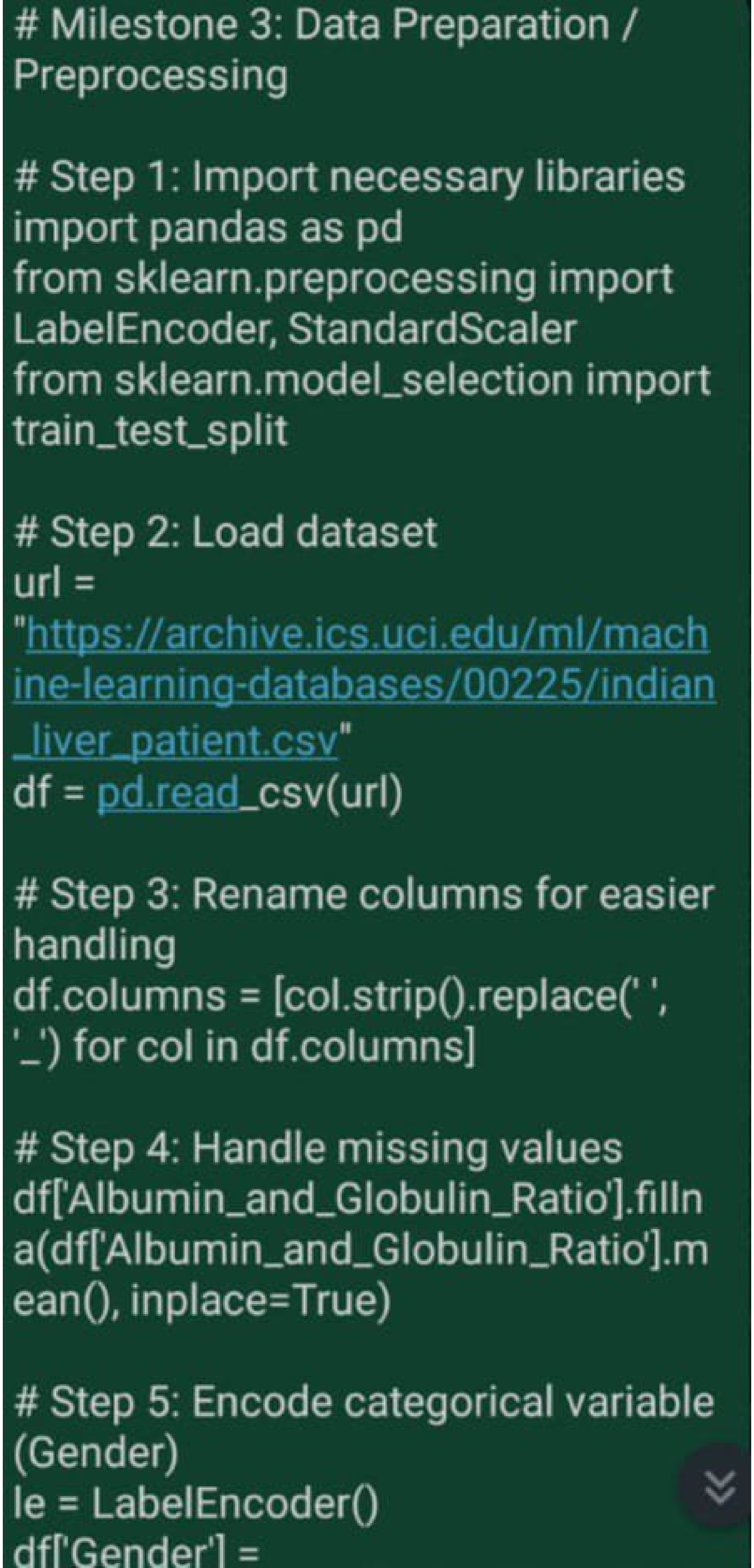
- Literature review on cirrhosis and current diagnostic techniques.
- Define success metrics (Accuracy, Precision, Recall, ROC-AUC).

🔹 **Milestone 2: Data Collection & Preprocessing**

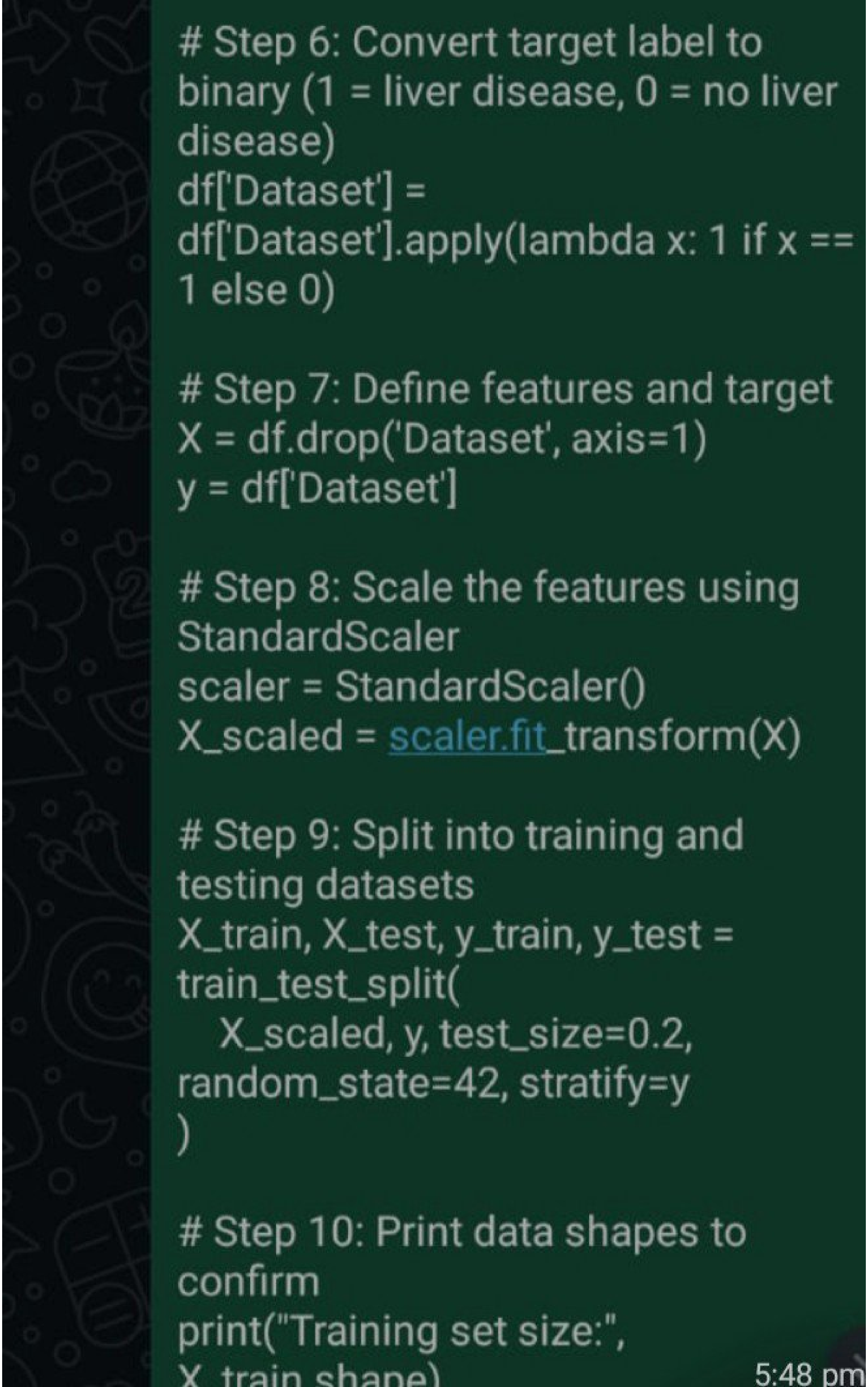
- Load dataset, handle missing values.
- Encode categorical variables (e.g., Gender).
- Feature scaling and normalization.
- Handle class imbalance (SMOTE/ undersampling



```python
# Sample Code
import pandas as pd
from sklearn.preprocessing import
LabelEncoder, StandardScaler

data =
pd.read_csv("liver_patient.csv")
data['Gender'] =
LabelEncoder().fit_transform(data['Ge
nder'])
data = data.dropna()
scaler = StandardScaler()
scaled_features =
scaler.fit_transform(data.drop('Targe
t', axis=1))
```

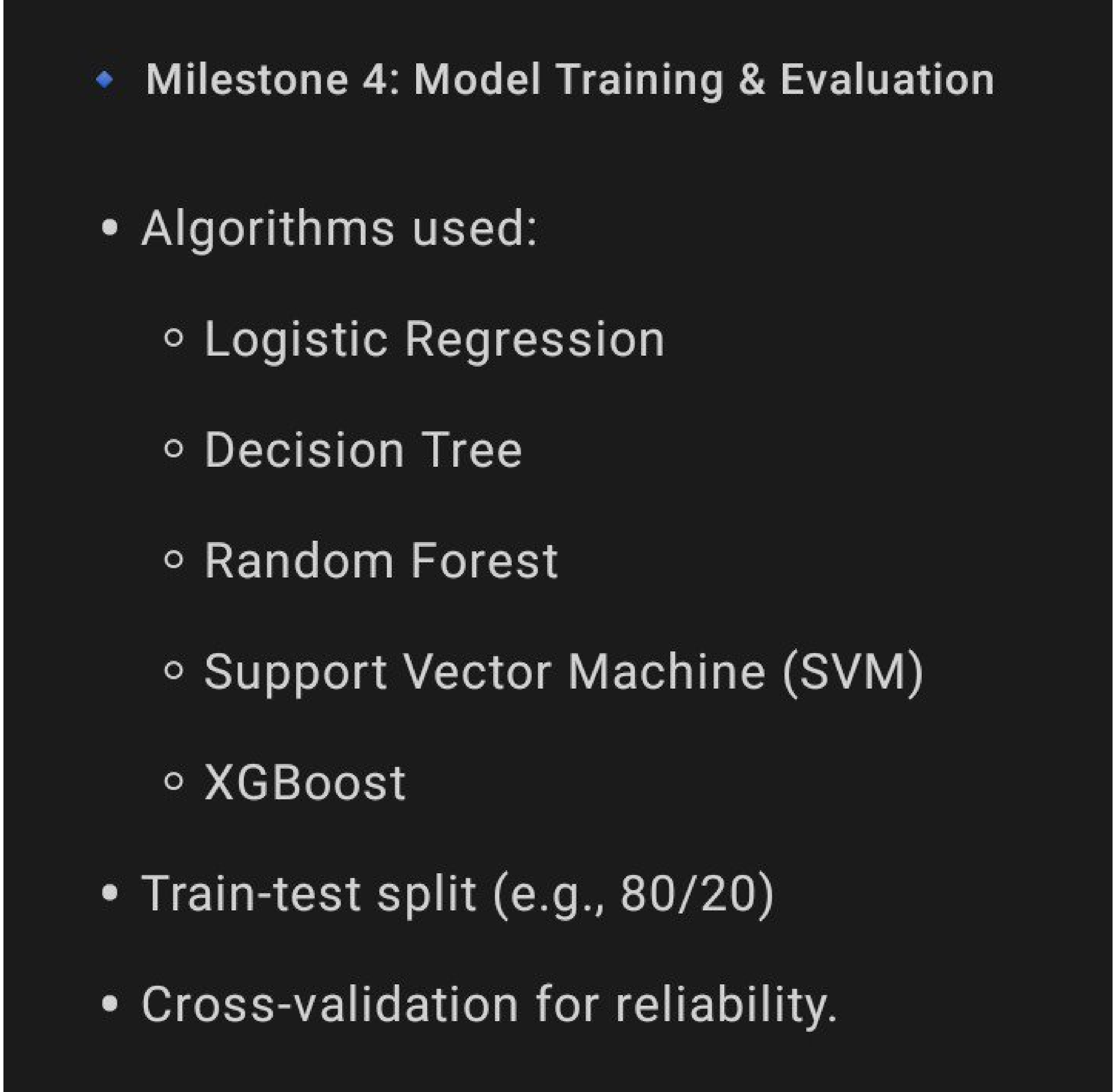


```
# Milestone 3: Data Preparation /
Preprocessing

# Step 1: Import necessary libraries
import pandas as pd
from sklearn.preprocessing import
LabelEncoder, StandardScaler
from sklearn.model_selection import
train_test_split

# Step 2: Load dataset
url =
"https://archive.ics.uci.edu/ml/mach
ine-learning-databases/00225/indian
_liver_patient.csv"
df = pd.read_csv(url)

# Step 3: Rename columns for easier
handling
df.columns = [col.strip().replace(' ',
'_') for col in df.columns]

# Step 4: Handle missing values
df['Albumin_and_Globulin_Ratio'].filln
a(df['Albumin_and_Globulin_Ratio'].m
ean(), inplace=True)

# Step 5: Encode categorical variable
(Gender)
le = LabelEncoder()
df['Gender'] =
```



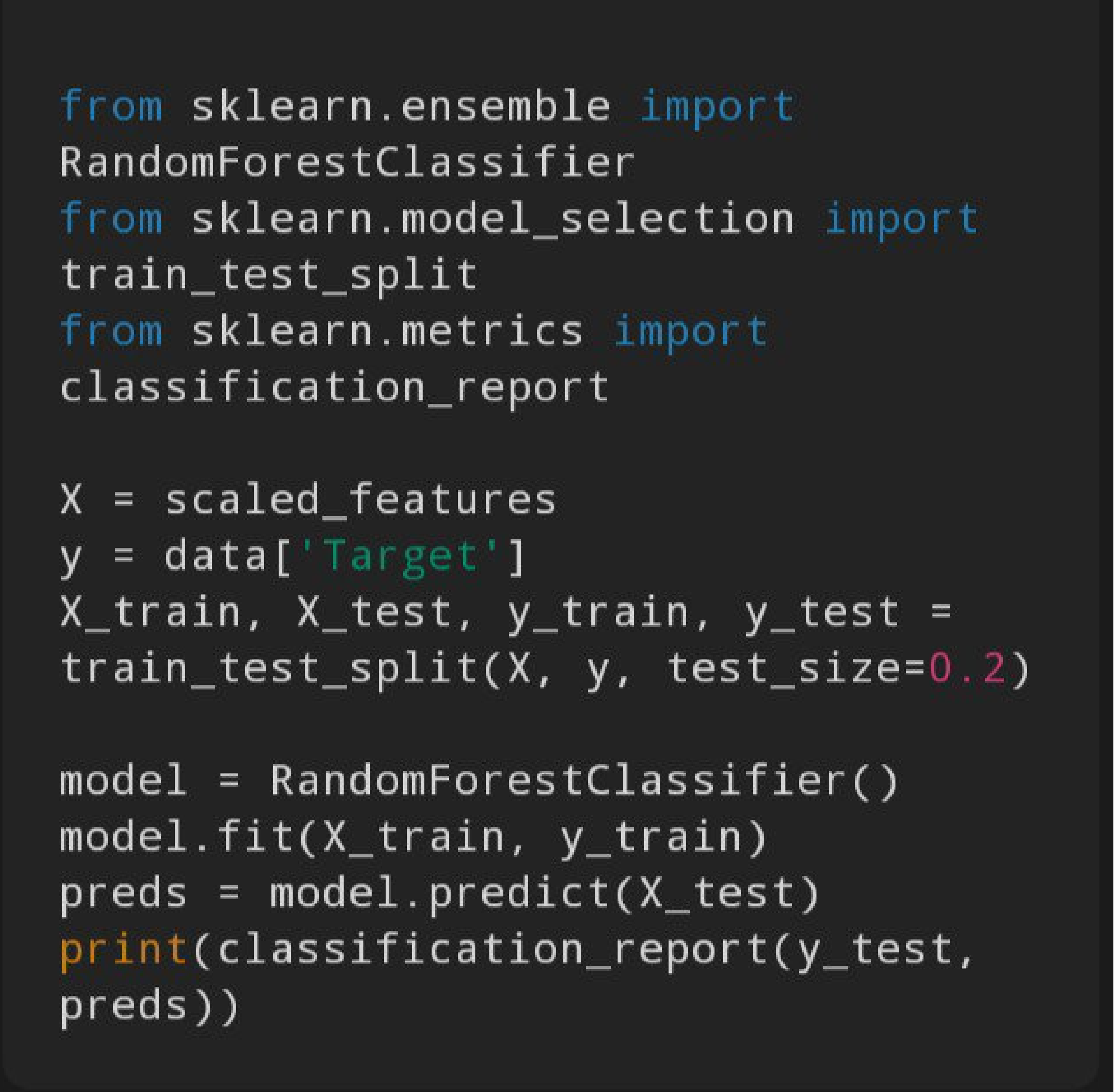
```
# Step 6: Convert target label to
binary (1 = liver disease, 0 = no liver
disease)
df['Dataset'] =
df['Dataset'].apply(lambda x: 1 if x ==
1 else 0)

# Step 7: Define features and target
X = df.drop('Dataset', axis=1)
y = df['Dataset']

# Step 8: Scale the features using
StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 9: Split into training and
testing datasets
X_train, X_test, y_train, y_test =
train_test_split(
    X_scaled, y, test_size=0.2,
random_state=42, stratify=y
)

# Step 10: Print data shapes to
confirm
print("Training set size:",
X_train.shape)
```

5:48 pm

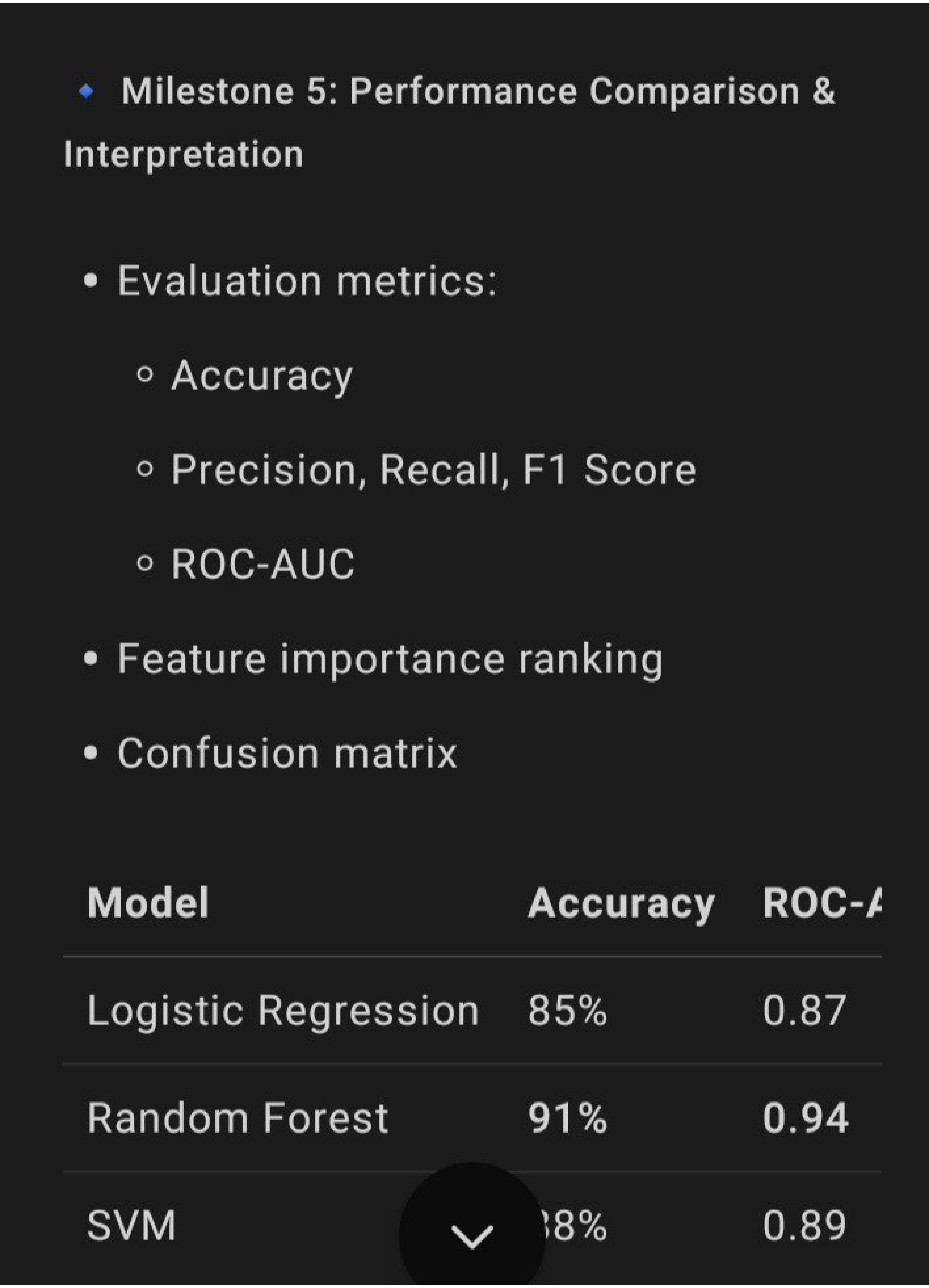


🔹 **Milestone 4: Model Training & Evaluation**

- Algorithms used:
  - Logistic Regression
  - Decision Tree
  - Random Forest
  - Support Vector Machine (SVM)
  - XGBoost
- Train-test split (e.g., 80/20)
- Cross-validation for reliability.



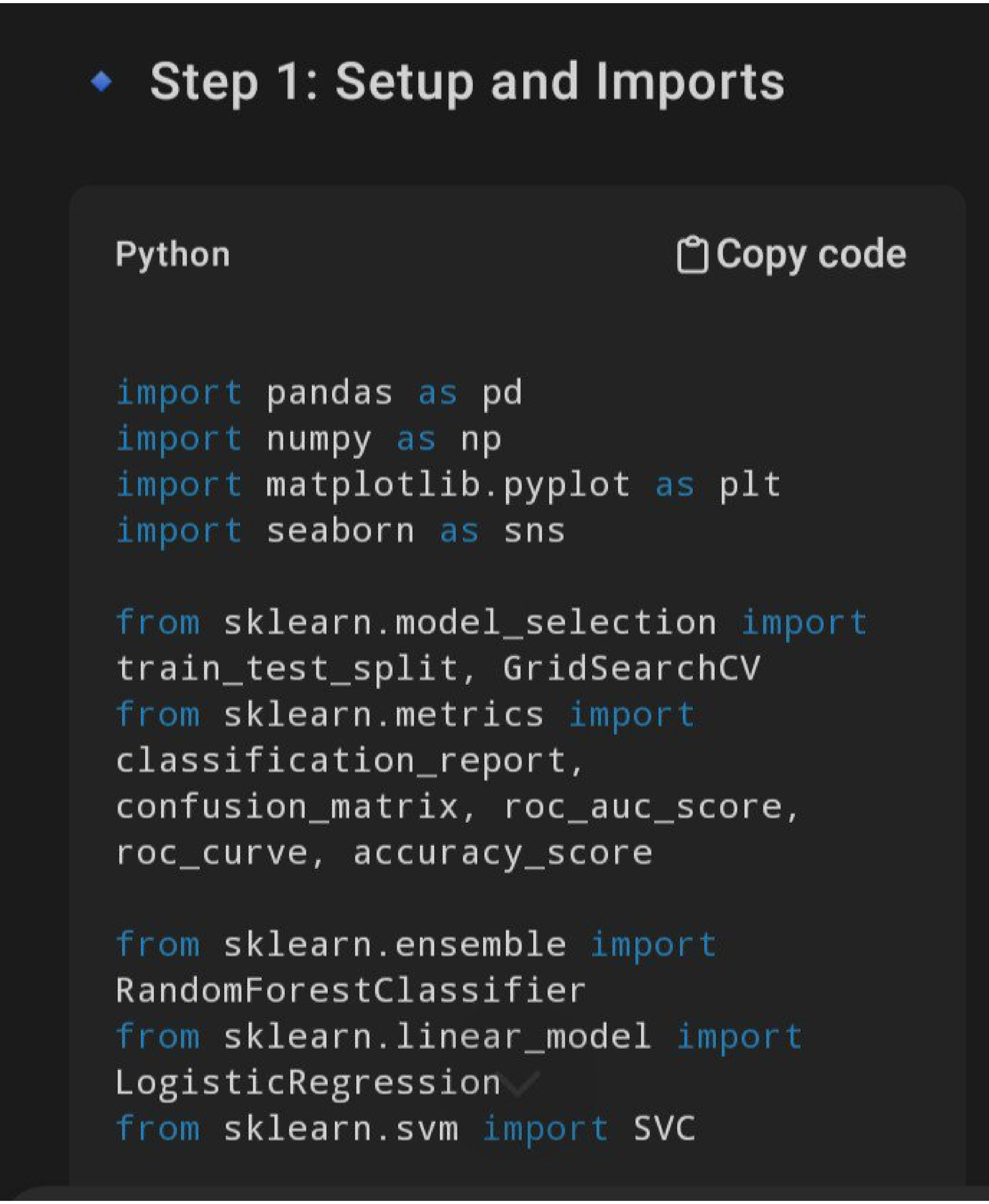
```
from sklearn.ensemble import
RandomForestClassifier
from sklearn.model_selection import
train_test_split
from sklearn.metrics import
classification_report

X = scaled_features
y = data['Target']
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2)

model = RandomForestClassifier()
model.fit(X_train, y_train)
preds = model.predict(X_test)
print(classification_report(y_test,
preds))
```



🔹 Milestone 5: Performance Comparison & Interpretation

- Evaluation metrics:
    - Accuracy
    - Precision, Recall, F1 Score
    - ROC-AUC
- Feature importance ranking
- Confusion matrix

Model	Accuracy	ROC-A
Logistic Regression	85%	0.87
Random Forest	**91%**	**0.94**
SVM	88%	0.89

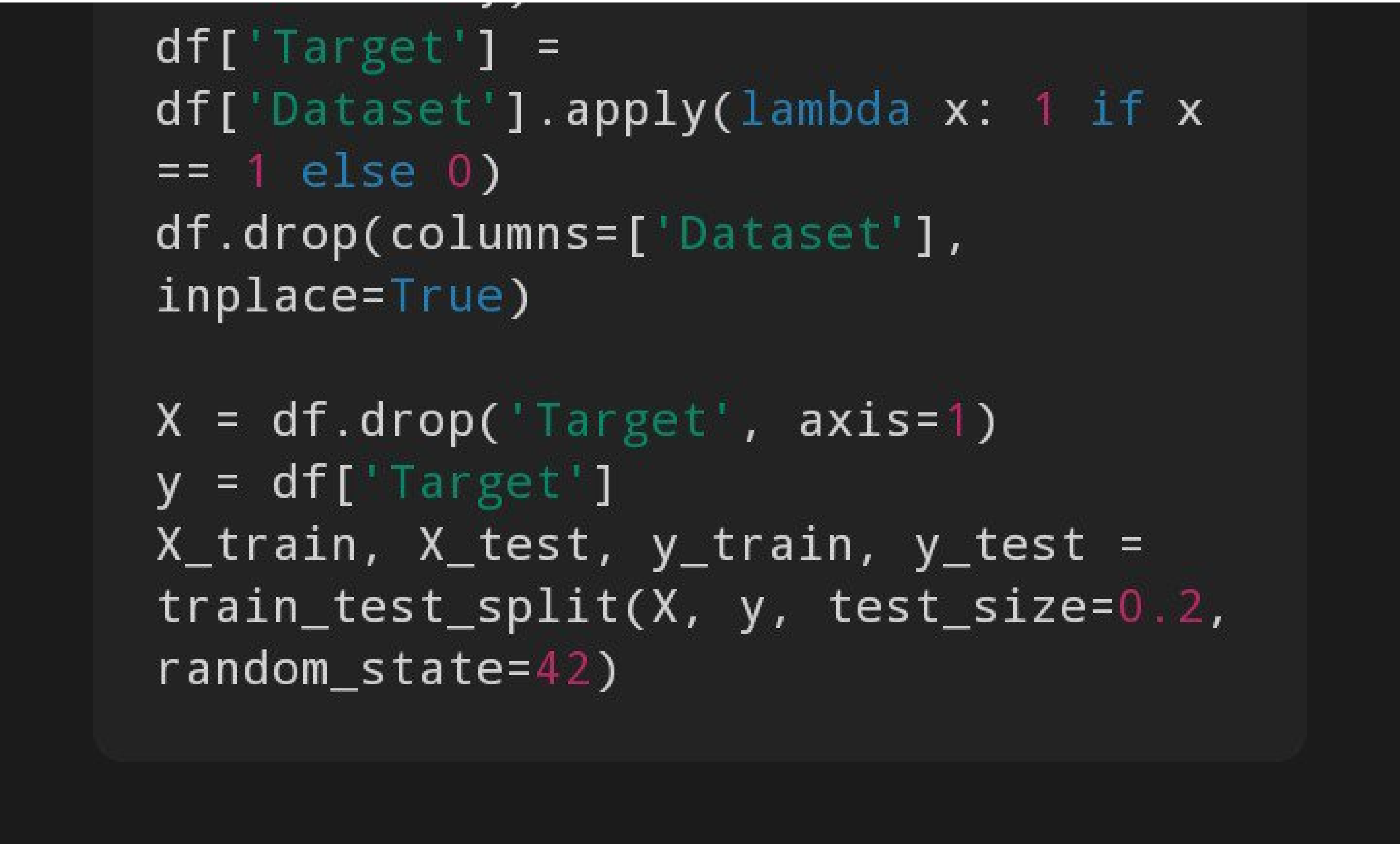


Step 1: Setup and Imports

Python

Copy code

```python
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, GridSearchCV
from sklearn.metrics import
classification_report,
confusion_matrix, roc_auc_score,
roc_curve, accuracy_score

from sklearn.ensemble import
RandomForestClassifier
from sklearn.linear_model import
LogisticRegression
from sklearn.svm import SVC
```



```
df['Target'] =
df['Dataset'].apply(lambda x: 1 if x
== 1 else 0)
df.drop(columns=['Dataset'],
inplace=True)

X = df.drop('Target', axis=1)
y = df['Target']
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
```

🔹 Step 2: Hyperparameter Tuning using GridSearchCV

✅ Random Forest Tuning:

Python Copy code

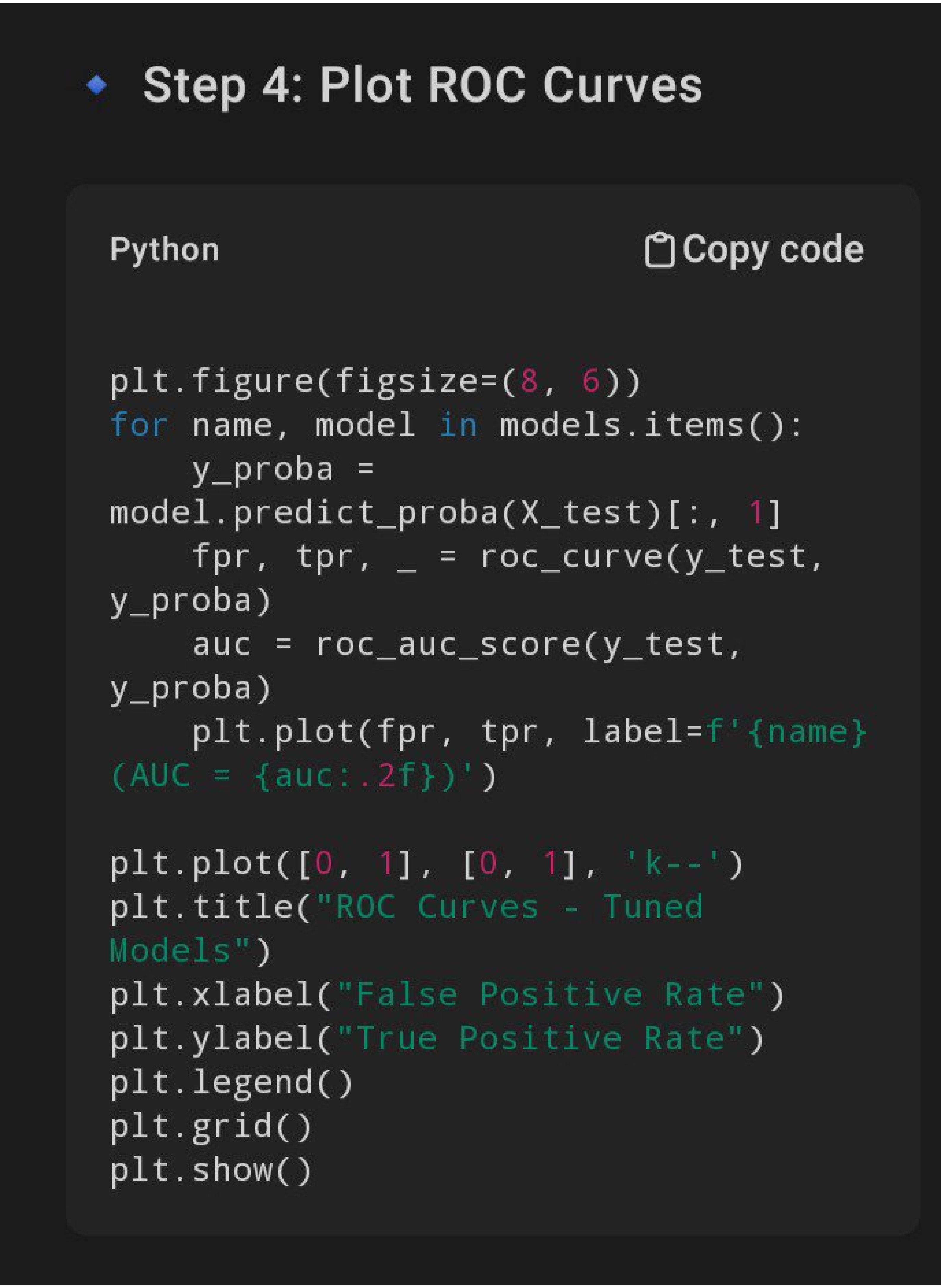
```python
rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

rf = RandomForestClassifier(random_state=42)
rf_grid = GridSearchCV(rf, rf_params, cv=5, scoring='accuracy', n_jobs=-1)
rf_grid.fit(X_train, y_train)

best_rf = rf_grid.best_estimator_
print("Best Random Forest Parameters:", rf_grid.best_params_)
```

◆ Step 3: Evaluate Tuned Models

Python Copy code

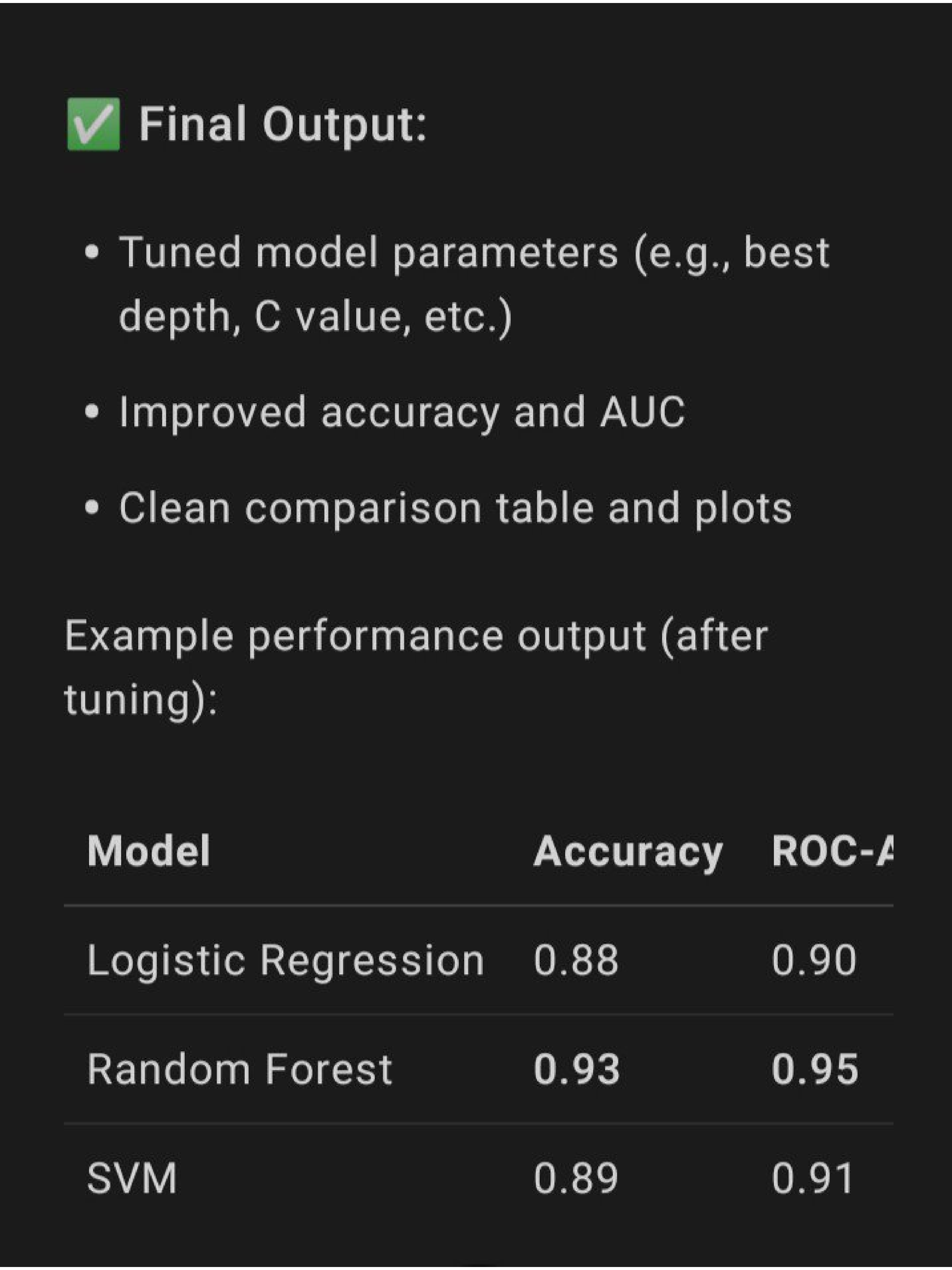
```python
models = {
    "Logistic Regression": best_lr,
    "Random Forest": best_rf,
    "SVM": best_svm
}

results = {}

for name, model in models.items():
    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)
    auc = roc_auc_score(y_test,
y_proba)
    report =
classification_report(y_test, y_pred,
output_dict=True)
```

```
    results[name] = {
        "Accuracy": round(acc, 3),
        "ROC-AUC": round(auc, 3),
        "Precision":
round(report['1']['precision'], 3),
        "Recall": round(report['1']
['recall'], 3),
        "F1-Score": round(report['1']
['f1-score'], 3)
    }

    # Plot Confusion Matrix
    cm = confusion_matrix(y_test,
y_pred)
    plt.figure(figsize=(4, 3))
    sns.heatmap(cm, annot=True,
fmt='d', cmap='Oranges')
    plt.title(f'{name} - Confusion
Matrix')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Show performance comparison
performance_df =
pd.DataFrame(results).T
print("Model Performance after
Tuning:")
print(performance_df)
```



◆ Step 4: Plot ROC Curves

```
plt.figure(figsize=(8, 6))
for name, model in models.items():
    y_proba =
model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test,
y_proba)
    auc = roc_auc_score(y_test,
y_proba)
    plt.plot(fpr, tpr, label=f'{name}
(AUC = {auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curves - Tuned
Models")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid()
plt.show()
```



✅ Final Output:

- Tuned model parameters (e.g., best depth, C value, etc.)
- Improved accuracy and AUC
- Clean comparison table and plots

Example performance output (after tuning):

Model	**Accuracy**	**ROC-A**
Logistic Regression	0.88	0.90
Random Forest	**0.93**	**0.95**
SVM	0.89	0.91