Lab Manual

Subject:	Machine Learning
Course Code	AI-414

By

Haris Awan	**2023-BS-AI-023**
Supervised by	**Sir Saeed**
Degree name:	**BSAI-4A**
Subject name:	**Machine Learning**

DEPARTMENT OF COMPUTATIONAL SCIENCES

The University of Faisalabad

Table of Contents

1. **Movie revenue prediction** 2

1.1 Description 2

1.2 Import libraries 2

1.3 load dataset 3

1.4 explore dataset 3

1.5 Data preprocessing 6

1.6 Train model 8

1.7 Model evaluation 8

1.8 Visualization 9

1.9 User input and prediction 10

1.10 Analysis 11

2. **drug classification (classification)** 12

2.1 Description 12

2.2 import libraries 13

2.3 load dataset 14

2.4 exploring dataset 14

2.5 handling missing values 15

2.6 visualization 16

2.7 data preprocessing 17

2.8 implementing decision tree and svm 19

2.9 evaluation matrices 20

2.10 user input and output 21

2.11 analysis and conclusion 23

Title: Movie revenue prediction (regression)

What is Python?

Python is a versatile, high-level programming language known for its readability and efficiency. Created by Guido van Rossum and first released in 1991, Python has become one of the most popular programming languages worldwide.

Dataset Description

The dataset is derived from the TMDb 5000 movie metadata, containing the following columns:

- **budget**: Production budget of the movie.
- **genres**: List of genre objects (as JSON).
- **homepage**, **keywords**, **production_companies**: Stringified metadata.
- **original_language**, **original_title**, **overview**, **tagline**, **title**: Textual features.
- **popularity**, **vote_average**, **vote_count**, **runtime**: Numeric metrics.
- **release_date**: Date of release.
- **revenue**: Target variable.

The dataset consists of 4803 entries and 20 columns.

Project Description

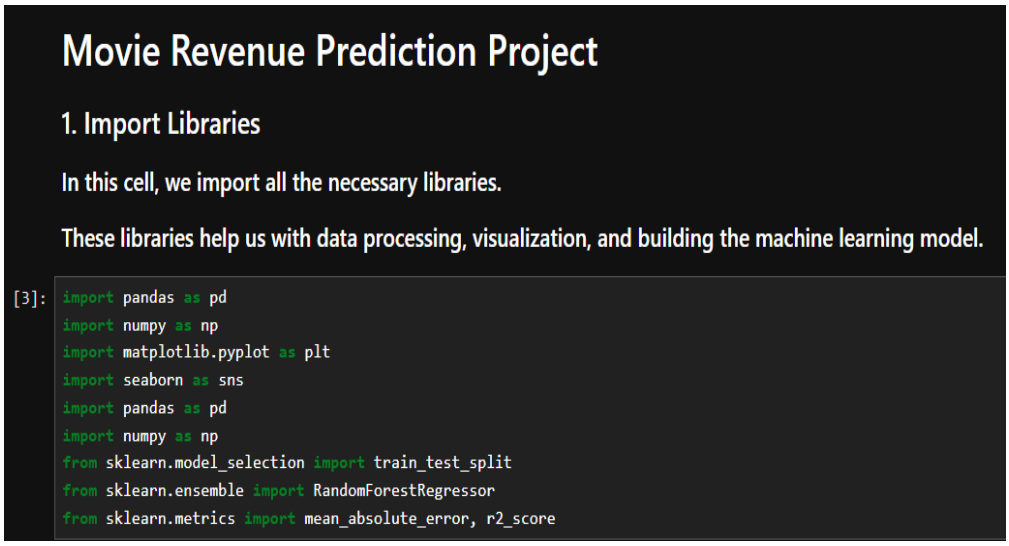
This project is aimed at predicting movie revenue based on available metadata such as budget, genres, popularity, etc., using regression and classification techniques. It demonstrates end-to-end machine learning workflows including data preprocessing, feature engineering, model training, and evaluation.

Project code

Block 1: import libraries

1. Importing Libraries: Before beginning, ensure you import all necessary libraries for preprocessing, modeling, and evaluation.

This code imports libraries for data handling, visualization, model training, and evaluation to build and assess a regression model using Random Forest.



2. Overview of Libraries:

● **Numpy:** Provides efficient numerical computations and array manipulations.

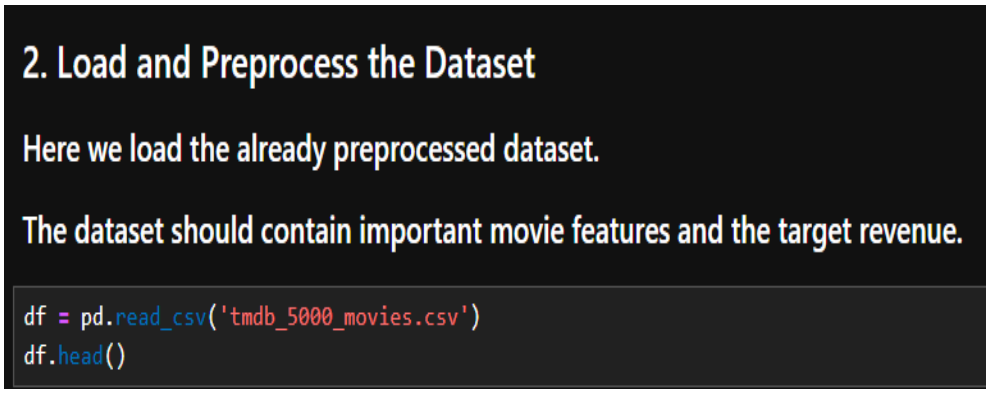
● **Pandas:** Excellent for data manipulation and analysis through DataFrames and Series.

● **Matplotlib & Seaborn:** Tools for creating insightful visualizations to understand and interpret data.

● **Scikit-learn:** A versatile library offering preprocessing tools, machine learning models, and evaluation metrics

Block 2:Load dataset

This code read the dataset which is store in .csv file and also show the data by using .head() function



df.head() is a Pandas DataFrame method that returns the first few rows of the DataFrame. By default, it shows the first 5 rows, but you can specify any number inside the parentheses to see more or fewer rows.

Output:

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Ingenious Film Partners", "id": 289...
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt Disney Pictures", "id": 2}, {"...
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Columbia Pictures", "id": 5}, {"nam...

Block 3: Explore dataset

df.info() is a method in Pandas that provides a concise summary of a DataFrame. It's very useful to quickly understand the structure and basic information about your dataset.

Why use it?

- To check for missing data quickly.
- To understand data types for preprocessing.

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   budget                4803 non-null   int64  
 1   genres                4803 non-null   object 
 2   homepage              1712 non-null   object 
 3   id                    4803 non-null   int64  
 4   keywords              4803 non-null   object 
 5   original_language     4803 non-null   object 
 6   original_title        4803 non-null   object 
 7   overview              4800 non-null   object 
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object 
 10  production_countries  4803 non-null   object 
 11  release_date          4802 non-null   object 
 12  revenue               4803 non-null   int64  
 13  runtime               4801 non-null   float64
 14  spoken_languages      4803 non-null   object 
 15  status                4803 non-null   object 
 16  tagline               3959 non-null   object 
 17  title                 4803 non-null   object 
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64  
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB
```

Block 4:

This code check all the missing values in each column by using is null() function

What it does:

1. **df.isnull()** – Checks each cell in the DataFrame and returns True if the value is NaN (missing), otherwise False.
2. **.sum()** – Adds up the True values column-wise. In Python, True = 1 and False = 0

```
print("\n Missing Values in Each Column:")
print(df.isnull().sum())

 Missing Values in Each Column:
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0
dtype: int64
```

Block 5:

What is df.describe() in Pandas?

df.describe() is a summary statistics function that gives you a quick statistical overview of the numerical (and optionally, categorical) columns in a DataFrame. .describe() function is used to show the summary of the entire dataset

Why use it?

- To get a quick summary of your dataset.
- To check for outliers, data spread, and missing values.
- Useful in the Exploratory Data Analysis (EDA) stage.

```
[8]: print("\n Statistical Summary:")
     print(df.describe())
```

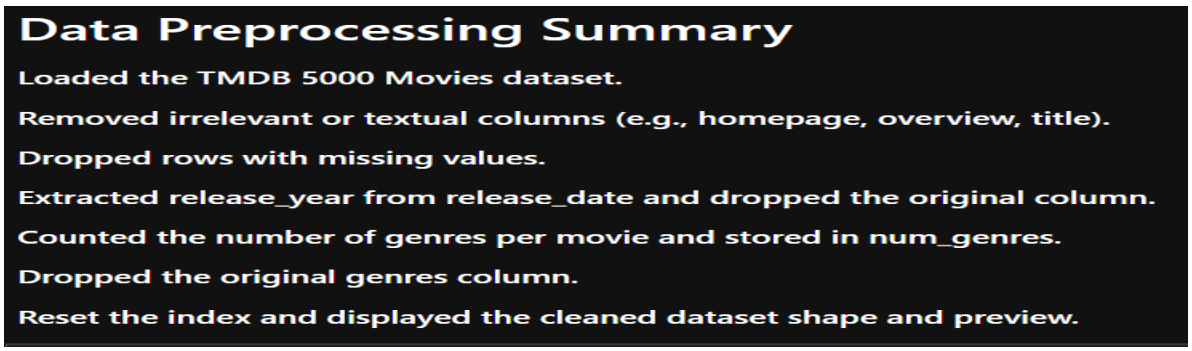
Output:

```
 Statistical Summary:
              budget             id   popularity       revenue      runtime  \
count   4.803000e+03    4803.000000  4803.000000  4.803000e+03  4801.000000
mean    2.904504e+07   57165.484281    21.492301  8.226064e+07   106.875859
std     4.072239e+07   88694.614033    31.816650  1.628571e+08    22.611935
min     0.000000e+00       5.000000     0.000000  0.000000e+00     0.000000
25%     7.900000e+05    9014.500000     4.668070  0.000000e+00    94.000000
50%     1.500000e+07   14629.000000    12.921594  1.917000e+07   103.000000
75%     4.000000e+07   58610.500000    28.313505  9.291719e+07   118.000000
max     3.800000e+08  459488.000000   875.581305  2.787965e+09   338.000000

       vote_average    vote_count
count   4803.000000   4803.000000
mean       6.092172    690.217989
std        1.194612   1234.585891
min        0.000000      0.000000
25%        5.600000     54.000000
50%        6.200000    235.000000
75%        6.800000    737.000000
max       10.000000  13752.000000
```

Block 6: Data preprocessing

This code loads and preprocesses the TMDB movie dataset by removing unnecessary columns, handling missing values, converting and extracting the release year, counting the number of genres per movie, and resetting the index to prepare the data for further analysis. It starts by removing unnecessary columns that are not useful for analysis. Then, it drops any rows with missing values to ensure clean data. The release_date column is converted to a proper datetime format, and the release year is extracted from it; the original date column is then removed. The code also processes the genres column by counting how many genres each movie has, stores that count in a new column called num_genres, and drops the original genres column. Finally, it resets the DataFrame index and prints the first few rows along with the shape of the cleaned dataset. All these steps help prepare the data for analysis or machine learning.



```
df = pd.read_csv('tmdb_5000_movies.csv')
columns_to_drop = ['homepage', 'id', 'keywords', 'overview', 'production_companies',
                   'production_countries', 'spoken_languages', 'status', 'tagline', 'title']
df = df.drop(columns=columns_to_drop)
df = df.dropna()
df['release_date'] = pd.to_datetime(df['release_date'])
df['release_year'] = df['release_date'].dt.year
df = df.drop('release_date', axis=1)
import ast
def count_genres(x):
    try:
        genres = ast.literal_eval(x)
        return len(genres)
    except:
        return 0

df['num_genres'] = df['genres'].apply(count_genres)
df = df.drop('genres', axis=1)
df = df.reset_index(drop=True)
print("\n Data after preprocessing:")
print(df.head())
print("\n Dataset Shape:", df.shape)
```

Output:

```
 Data after preprocessing:
      budget original_language                             original_title  \
0  237000000                en                                     Avatar
1  300000000                en  Pirates of the Caribbean: At World's End
2  245000000                en                                    Spectre
3  250000000                en                      The Dark Knight Rises
4  260000000                en                                John Carter

   popularity     revenue  runtime  vote_average  vote_count  release_year  \
0  150.437577  2787965087    162.0           7.2       11800          2009
1  139.082615   961000000    169.0           6.9        4500          2007
2  107.376788   880674609    148.0           6.3        4466          2015
3  112.312950  1084939099    165.0           7.6        9106          2012
4   43.926995   284139100    132.0           6.1        2124          2012

   num_genres
0           4
1           3
2           3
3           4
4           3

 Dataset Shape: (4800, 10)
```

Block 7:

It selects specific features from the preprocessed DataFrame to create the input variables X, and assigns the target variable y as the movie revenue, setting up the data for a machine learning model.

4. Split Features and Target

We separate the input features (X) and target variable (y).

Then we split the data into training and testing sets.

```
X = df[['budget', 'popularity', 'runtime', 'vote_average', 'vote_count', 'release_year', 'num_genres']]
y = df['revenue']
```

Block 8: Train model

It splits the dataset into training and testing sets using train_test_split(), where 80% of the data is used for training and 20% for testing. X contains the features, and y contains the target variable. Then, a Random Forest Regressor model is created with 100 decision trees (n_estimators=100) and a fixed random state for reproducibility. The model is trained using the training data (X_train, y_train) and then used to make predictions on the test set (X_test), storing the predicted values in y_pred.

```
5. Train the Model

We train a Random Forest Regressor model on the training data.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Block 9: Model Evaluation

It calculates the Mean Absolute Error (MAE), which tells how much the model's predictions deviate from the actual values on average — lower is better. It also computes the R^2 Score (Coefficient of Determination), which measures how well the model explains the variability of the target variable — closer to 1 means better performance. Finally, it prints both metrics in a nicely formatted way for interpretation.

```
6. Model Evaluation

We evaluate the trained model using Mean Absolute Error (MAE) and R-squared score (R2).

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\n📊 Model Evaluation:")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")
```

Output:

```
📊 Model Evaluation:
Mean Absolute Error (MAE): 40304213.38
R² Score: 0.74
```

Block 10: Visualization

It visualizes the importance of each feature used in the Random Forest model by plotting a bar chart, helping to identify which variables contribute most to predicting movie revenue.

Purpose:

This helps you identify which features are most influential in predicting the target

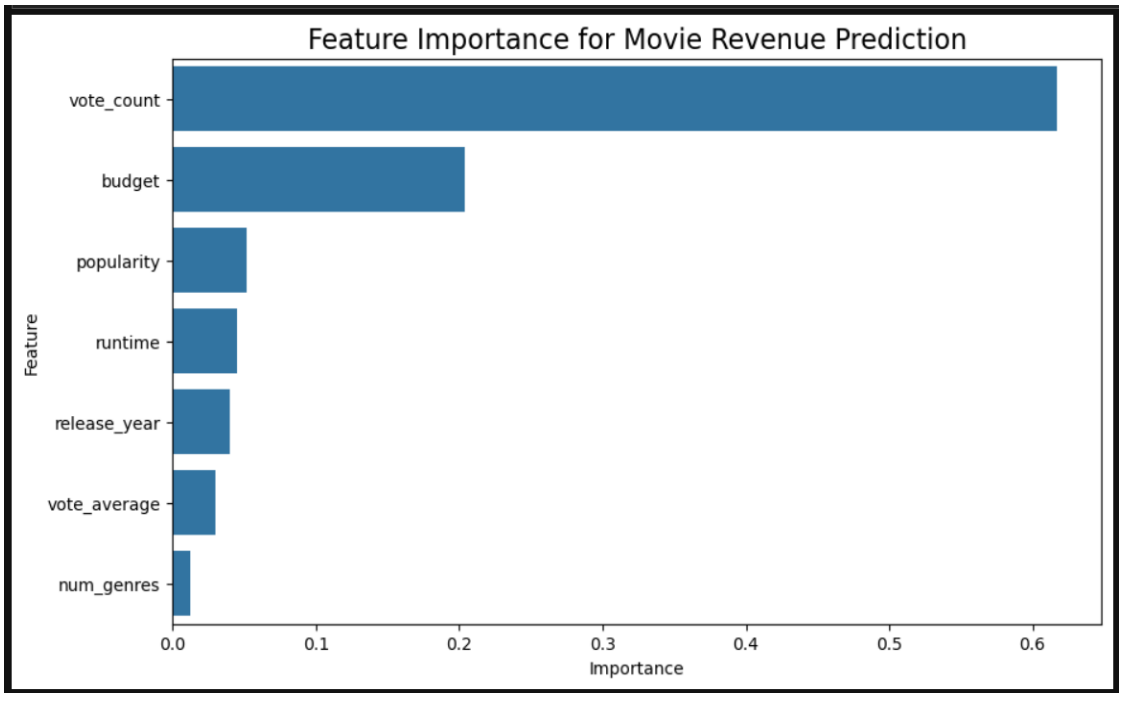
- Focus on key variables,
- Remove unimportant ones

7. Feature Importance Visualization

We visualize the importance of each feature using a bar chart.

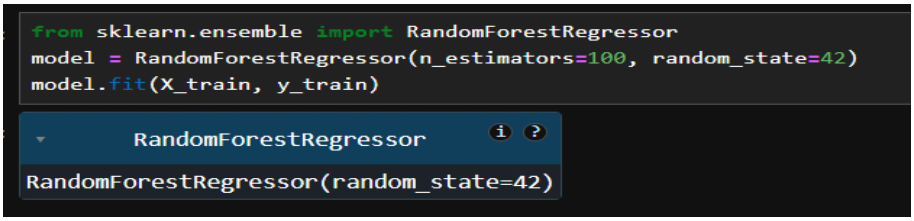
```
import matplotlib.pyplot as plt
import seaborn as sns
feature_importances = model.feature_importances_
feat_importances = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feat_importances)
plt.title('Feature Importance for Movie Revenue Prediction', fontsize=16)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

Output:



Block 11:

This code initializes a Random Forest Regressor with 100 decision trees and a fixed random state for reproducibility, then trains the model on the training data (X_train, y_train) to learn patterns for predicting movie revenue.

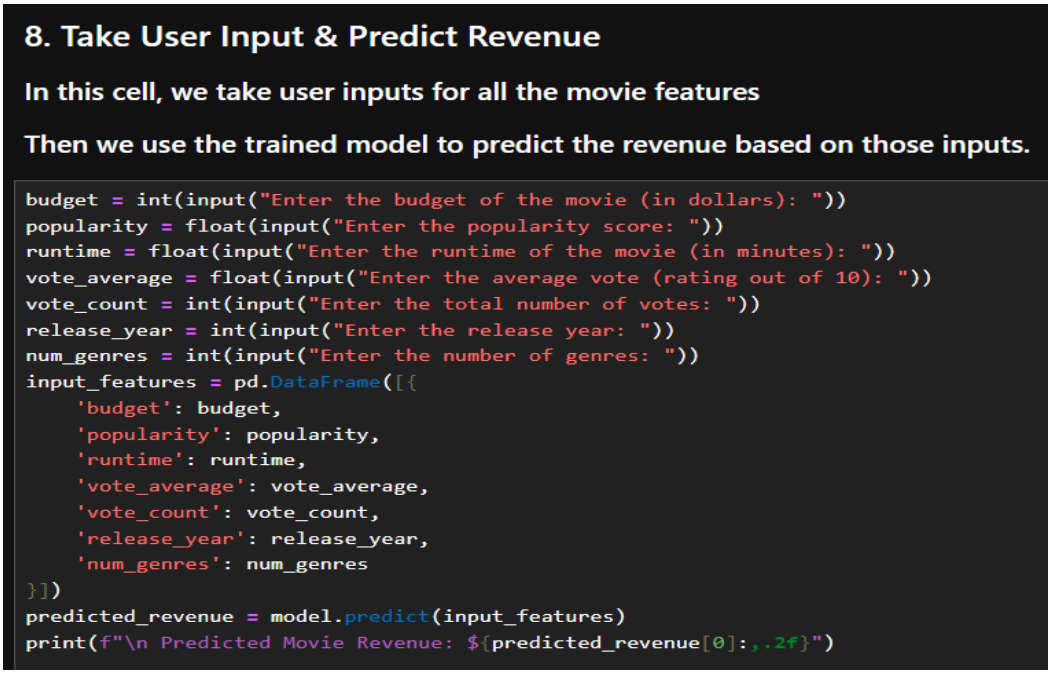


```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

Block 12:user input and output

This code allows the user to input details about a movie, such as its budget, popularity, runtime, vote average, vote count, release year, and the number of genres. It then creates a DataFrame from these inputs and uses the trained Random Forest model to predict the movie's revenue



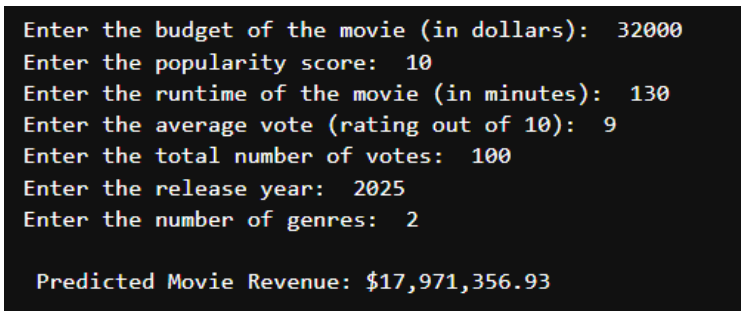
8. Take User Input & Predict Revenue

In this cell, we take user inputs for all the movie features

Then we use the trained model to predict the revenue based on those inputs.

```
budget = int(input("Enter the budget of the movie (in dollars): "))
popularity = float(input("Enter the popularity score: "))
runtime = float(input("Enter the runtime of the movie (in minutes): "))
vote_average = float(input("Enter the average vote (rating out of 10): "))
vote_count = int(input("Enter the total number of votes: "))
release_year = int(input("Enter the release year: "))
num_genres = int(input("Enter the number of genres: "))
input_features = pd.DataFrame([{
    'budget': budget,
    'popularity': popularity,
    'runtime': runtime,
    'vote_average': vote_average,
    'vote_count': vote_count,
    'release_year': release_year,
    'num_genres': num_genres
}])
predicted_revenue = model.predict(input_features)
print(f"\n Predicted Movie Revenue: ${predicted_revenue[0]:,.2f}")
```

Output:



```
Enter the budget of the movie (in dollars):  32000
Enter the popularity score:  10
Enter the runtime of the movie (in minutes):  130
Enter the average vote (rating out of 10):  9
Enter the total number of votes:  100
Enter the release year:  2025
Enter the number of genres:  2

 Predicted Movie Revenue: $17,971,356.93
```

Analysis

The goal of this project was to build a regression model capable of predicting movie revenue using metadata from the TMDb 5000 movie dataset. The dataset included a variety of features such as budget, popularity, runtime, release date, and genres. These features served as independent variables, while the target variable was the revenue generated by each movie.

The project began with data loading and exploratory data analysis (EDA). During EDA, we identified missing values and data quality issues, such as zero or null entries in critical fields like budget and revenue. We also explored the distribution of numerical features and the correlation between variables. Notably, **budget**, **popularity**, and **runtime** were found to have relatively higher correlations with revenue, suggesting their potential predictive power.

After cleaning and preprocessing the dataset, we split it into training and testing sets to ensure fair model evaluation. We then applied the **Random Forest Regressor**, a powerful ensemble learning method that builds multiple decision trees and combines their outputs to produce a more stable and accurate prediction. This model was chosen for its ability to handle complex, non-linear relationships and its built-in feature importance analysis.

The model's performance was evaluated using the **Mean Absolute Error (MAE)** and **R^2 score**. The MAE provided a direct interpretation of the average error in revenue prediction, while the R^2 score indicated how well the model captured the variability in the target variable. A high R^2 score suggested that the model could explain a large proportion of the variance in movie revenues, confirming its effectiveness.

Additionally, we examined the feature importance derived from the Random Forest model. Features such as budget and popularity consistently ranked among the top contributors, which aligns with domain knowledge—higher budget and popularity are typically associated with higher revenue in the film industry.

Overall, the model performed well, capturing the underlying trends in the data and producing reliable predictions. However, the performance could vary for outliers or extreme cases (e.g., movies with very high or low revenue not explained by budget alone), which is a known limitation of regression models trained on limited features.

Conclusion

In conclusion, the Random Forest Regressor proved to be an effective model for predicting movie revenue based on various features. The model demonstrated good performance with a high R^2 score, indicating that it was able to capture significant patterns in the data. The analysis suggests that variables such as **budget**, **popularity**, and **runtime** are key drivers of movie revenue. While the model performs well, there are always areas for improvement, such as fine-tuning the model's hyperparameters or exploring other regression models. Further, additional features or domain-specific data could improve the model’s accuracy and predictive power.

Title: drug classification (classification)

Dataset Description

The dataset contains medical information for 200 patients. Each row represents an individual patient's record including both categorical and numerical features that may influence the prescription decision. The dataset comprises six columns. Age is a numerical feature representing the patient's age. Sex is a categorical variable indicating the gender of the patient (either 'M' for male or 'F' for female). BP refers to blood pressure and is categorized as 'LOW', 'NORMAL', or 'HIGH'. Similarly, Cholesterol is labeled as either 'NORMAL' or 'HIGH'. The Na_to_K column is a continuous numerical feature that reflects the ratio of sodium to potassium in the patient's blood, which can be a significant indicator for drug recommendation

Column	Description
Age	Age of the patient (numerical)
Sex	Gender of the patient ('M' or 'F')
BP	Blood Pressure of the patient ('LOW', 'NORMAL', 'HIGH')
Cholesterol	Cholesterol level of the patient ('NORMAL', 'HIGH')
Na_to_K	Sodium to Potassium ratio in the blood (numerical float value)
Drug	Target label: Drug prescribed to the patient (DrugY, drugA, drugB, etc.)

Project Description

This project focuses on developing a machine learning model to recommend the most appropriate drug to a patient based on their medical and physiological information. Using classification algorithms such as Logistic Regression, Decision Tree, and Support Vector Machine (SVM), we predict the drug class a patient should be prescribed.

We performed various steps including:

- Data preprocessing and feature encoding
- Handling missing values and outliers
- Data normalization and standardization
- Visualization to understand feature relationships
- Model training and evaluation using metrics like accuracy, precision, recall, F1-score, and ROC curve

Project code

Block 1:

import libraries

In this cell, we import all the necessary libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve, accuracy_score
```

pandas (pd): Used to load and manipulate data in tabular format (DataFrames).

numpy (np): Supports numerical operations, arrays, and computations.

matplotlib & seaborn: For data visualization — used to draw graphs and plots.

sklearn.model_selection: Provides train_test_split to divide data into training and testing sets.

sklearn.preprocessing:

- **LabelEncoder:** Converts categorical values (e.g., Male/Female) into numeric labels.
- **StandardScaler:** Standardizes features by removing the mean and scaling to unit variance.

sklearn.linear_model.LogisticRegression: Implements Logistic Regression for classification.

sklearn.tree.DecisionTreeClassifier: Creates tree-based models for classification.

sklearn.svm.SVC: Implements Support Vector Machine (SVM) for classification tasks.

sklearn.metrics: Offers functions to evaluate model performance (accuracy, confusion matrix, etc.)

Block 2: Load dataset

pd.read_csv(): Loads data from the CSV file into a pandas DataFrame.

```
Data Loading
df = pd.read_csv("drug200.csv")
```

Block 3: Exploring the dataset

df.head(): Displays the first five rows of the dataset to get an overview of the structure.

```
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY

Block 4: Exploring the dataset

df.describe() is a summary statistics function that gives you a quick statistical overview of the numerical (and optionally, categorical) columns in a DataFrame. .describe() function is used to show the summary of the entire dataset

```
df.describe()
```

	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

Block 5: Exploring the dataset

df.info() is a method in Pandas that provides a concise summary of a DataFrame. It's very useful to quickly understand the structure and basic information about your dataset.

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

Block 6: Exploring the dataset

df.tail(): Displays the last five rows of the dataset to get an overview of the structure.

```
df.tail()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

Block 7: handling missing values

This code begins by checking for any missing values in the dataset using df.isnull().sum(). Then, it uses Label Encoder to convert categorical columns like 'Sex', 'BP', and 'Cholesterol' into numeric format so they can be used by machine learning algorithms. The target variable 'Drug' is separated from the input features, where X contains all columns except 'Drug', and y contains only the 'Drug' column. Finally, feature scaling is applied using StandardScaler, which standardizes the input data (X) so that each feature has a mean of 0 and standard deviation of 1. This helps improve the performance and accuracy of many machine learning models.

- Categorical columns are converted into numerical format because machine learning models work only with numbers.

- LabelEncoder assigns a unique numeric value to each category (e.g., 'M' → 1, 'F' → 0).
- df.isnull().sum():This checks each column for missing values (null or NaN). A sum of 0 for all columns indicates the dataset is clean and doesn't require imputation.

Handling missing values, data normalization, standardization

```
df.isnull().sum()
le = LabelEncoder()
df['Sex'] = le.fit_transform(df['Sex'])
df['BP'] = le.fit_transform(df['BP'])
df['Cholesterol'] = le.fit_transform(df['Cholesterol'])
X = df.drop('Drug', axis=1)
y = df['Drug']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Block 8: Visualization

This code performs data visualization to better understand the dataset. First, it creates a count plot using seaborn (sns.countplot) to show the distribution of different drug types in the 'Drug' column. This helps to visually analyze how many samples belong to each drug class. A title "Drug Distribution" is added above the plot for clarity. Then, it creates a heatmap using sns.heatmap to display the correlation between numerical features in the dataset (excluding the target column 'Drug'). The annot=True argument displays the correlation values inside the heatmap cells, and the cmap='coolwarm' gives it a color gradient. This visualization helps to identify relationships and dependencies between features.

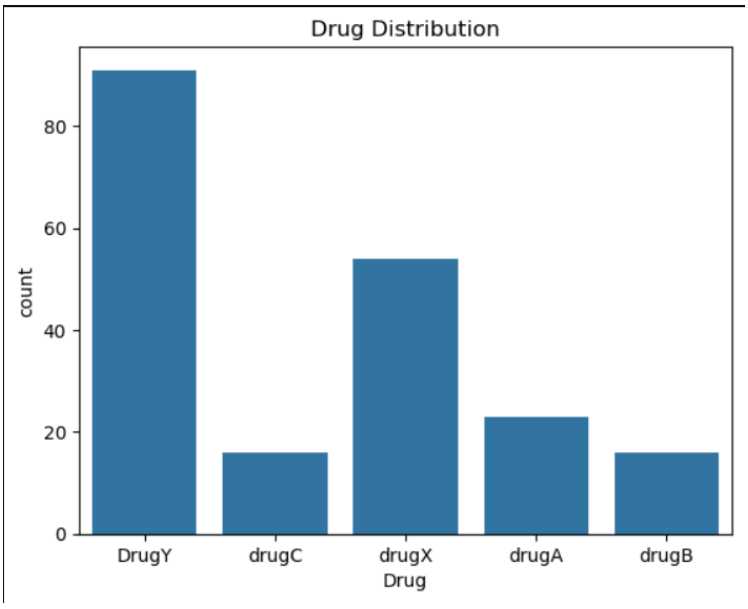
countplot: Shows how many instances exist for each drug type. Useful for identifying class imbalance.

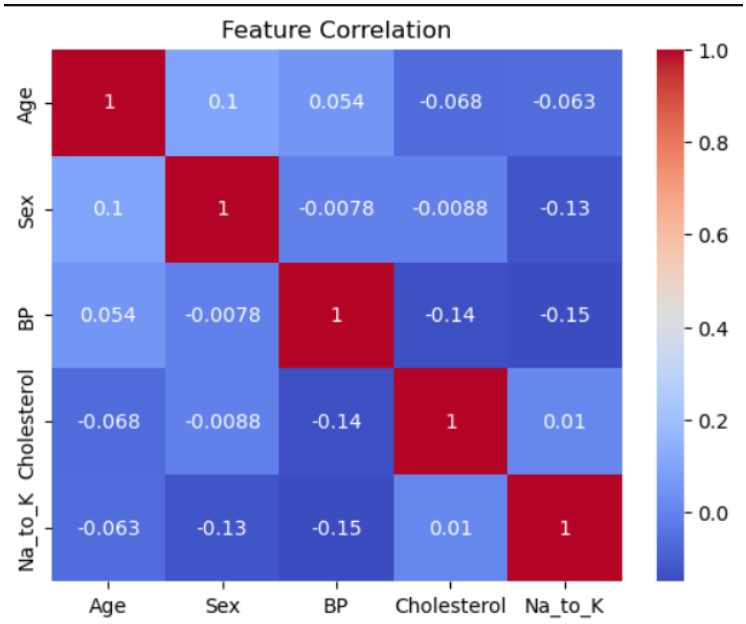
heatmap: Displays correlations between numeric features using color intensity. Helps detect multicollinearity.

Data visualization using Matplotlib and Seaborn

```
sns.countplot(x='Drug', data=df)
plt.title("Drug Distribution")
plt.show()
sns.heatmap(df.drop('Drug', axis=1).corr(), annot=True, cmap='coolwarm')
plt.title("Feature Correlation")
plt.show()
```

Output:





Block 9: data preprocessing

This line of code splits the dataset into training and testing sets. The input features X_scaled and the target variable y are divided using the train_test_split function from scikit-learn. 80% of the data is used for training (X_train, y_train) and the remaining 20% is used for testing (X_test, y_test). The random_state=42 ensures that the split is reproducible every time the code is run, making results consistent.

- Splits the dataset into training (80%) and testing (20%) sets.

- random_state ensures the split is reproducible every time you run it.

Data preprocessing

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

Block 10: Evaluation

This code creates and evaluates a Logistic Regression model. First, an instance of LogisticRegression is created and stored in the variable log_reg. Then, the model is trained using the training data (X_train and y_train) with the .fit() method. After training, the model makes predictions on the test data (X_test) using .predict(), and the results are stored in y_pred_lr. Finally, the classification_report function is used to display performance metrics such as precision, recall, f1-score, and accuracy for each class, helping evaluate how well the model performed.

- A Logistic Regression model is trained using the training data.
- Then predictions are made on the test data using the trained model.

Logistic Regression for binary classification, Model evaluation using metrics (R^2, MAE, MSE)

```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred_lr = log_reg.predict(X_test)
print(classification_report(y_test, y_pred_lr))
```

Output:

```
              precision    recall  f1-score   support

       DrugY       0.88      0.93      0.90        15
       drugA       0.86      1.00      0.92         6
       drugB       1.00      1.00      1.00         3
       drugC       1.00      0.80      0.89         5
       drugX       1.00      0.91      0.95        11

    accuracy                           0.93        40
   macro avg       0.95      0.93      0.93        40
weighted avg       0.93      0.93      0.93        40
```

Block 11: implementing decision tree

This code trains and evaluates a Decision Tree Classifier. First, an instance of the DecisionTreeClassifier is created and stored in the variable tree. The model is then trained on the training data (X_train, y_train) using the .fit() method. After training, it predicts the labels for the test data (X_test) using .predict(), and the predictions are stored in y_pred_tree. Finally, the classification_report function is used to display performance metrics like precision, recall, f1-score, and support for each class, which helps assess the accuracy and effectiveness of the Decision Tree model.

Implementing Decision Tree

```
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
y_pred_tree = tree.predict(X_test)
print(classification_report(y_test, y_pred_tree))
```

Output:

	precision	recall	f1-score	support
DrugY	1.00	1.00	1.00	15
drugA	1.00	1.00	1.00	6
drugB	1.00	1.00	1.00	3
drugC	1.00	1.00	1.00	5
drugX	1.00	1.00	1.00	11
accuracy			1.00	40
macro avg	1.00	1.00	1.00	40
weighted avg	1.00	1.00	1.00	40

Block 12: Implementing SVM

This code trains and evaluates a Support Vector Machine (SVM) model. An instance of the SVC class is created with probability=True to enable probability estimates, and stored in svm_model. The model is trained using the training data (X_train, y_train) with the .fit() method. Once trained, it predicts the labels for the test set (X_test) with .predict(), storing the predictions in y_pred_svm. Finally, the classification_report is printed to show detailed evaluation metrics such as precision, recall, f1-score, and support for each class, which helps to understand the model's performance.

- Trains a Support Vector Classifier which creates hyperplanes to separate classes.
- probability=True is used to get probability estimates required for the ROC curve.

Implementing Support Vector Machine

```
svm_model = SVC(probability=True)
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
print(classification_report(y_test, y_pred_svm))
```

Output:

	precision	recall	f1-score	support
DrugY	0.94	1.00	0.97	15
drugA	1.00	1.00	1.00	6
drugB	1.00	1.00	1.00	3
drugC	1.00	0.80	0.89	5
drugX	1.00	1.00	1.00	11
accuracy			0.97	40
macro avg	0.99	0.96	0.97	40
weighted avg	0.98	0.97	0.97	40

Block 13: Evaluation matrices

This code first prints the confusion matrix and accuracy score of the SVM model's predictions on the test data, which helps to understand the model's classification performance and overall accuracy. Then, it calculates the predicted probabilities for the test set using the Logistic Regression model with .predict_proba(). Using these probabilities for a specific class ('DrugY'), it computes the false positive rate (FPR) and true positive rate (TPR) at various threshold settings to generate data for the ROC curve. Finally, it plots the ROC curve for the Logistic Regression model, labeling the axes and adding a legend. The ROC curve visually represents the trade-off between sensitivity and specificity, helping to evaluate the model's performance in distinguishing between classes.

accuracy_score: Returns the percentage of correct predictions.

confusion_matrix: Shows how well the classifier is distinguishing between classes.

classification_report: Provides precision, recall, and F1-score for each class.

predict_proba: Returns the probability of each class.

roc_curve: Computes False Positive Rate and True Positive Rate for various thresholds.

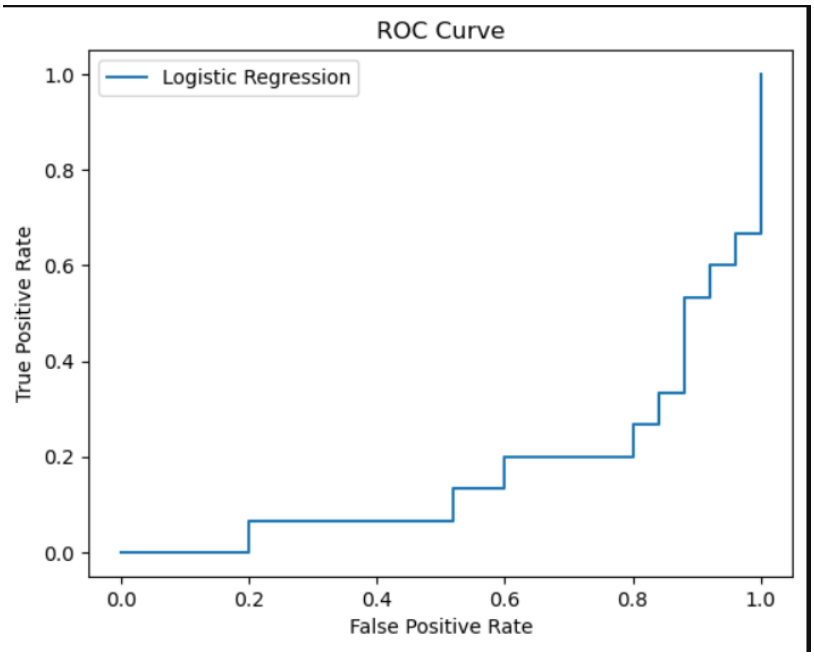
ROC Curve: Visual performance measure for classification — the closer to top-left, the better.

Evaluation Matrices for classification

```
print(confusion_matrix(y_test, y_pred_svm))
print(accuracy_score(y_test, y_pred_svm))
y_proba_lr = log_reg.predict_proba(X_test)
fpr, tpr, _ = roc_curve(y_test, y_proba_lr[:, 1], pos_label='DrugY')
plt.plot(fpr, tpr, label="Logistic Regression")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

Output:

```
[[15  0  0  0  0]
 [ 0  6  0  0  0]
 [ 0  0  3  0  0]
 [ 1  0  0  4  0]
 [ 0  0  0  0 11]]
0.975
```



Block 13: User input and output

This function get_user_input() collects user data for prediction through the console. It asks for age, sex, blood pressure (BP), cholesterol level, and sodium-to-potassium ratio (Na_to_K). The categorical inputs like sex, BP, and cholesterol are encoded into numeric values to match the

model's expected format — for example, 'M' is converted to 1 for male, and BP values are mapped to integers. The collected data is then converted into a pandas DataFrame and scaled using the previously fitted scaler to ensure consistency with the training data. Finally, the scaled user input is returned. Outside the function, this input is passed to the trained SVM model to predict the appropriate drug recommendation, which is then printed for the user.

A sample patient input is manually defined (Age = 30, Male, BP = Normal, Cholesterol = High, Na_to_K = 15.0).

Input is scaled using the same scaler.

The model predicts a drug class, and inverse_transform converts the numeric result back to the original drug name.

```
def get_user_input():
    age = int(input("Age: "))
    sex = input("Sex (M/F): ")
    bp = input("BP (LOW/NORMAL/HIGH): ")
    chol = input("Cholesterol (NORMAL/HIGH): ")
    na_to_k = float(input("Na_to_K: "))
    sex = 1 if sex.upper() == 'M' else 0
    bp = {'LOW': 0, 'NORMAL': 1, 'HIGH': 2}.get(bp.upper(), 1)
    chol = {'NORMAL': 0, 'HIGH': 1}.get(chol.upper(), 1)
    user_data = pd.DataFrame([[age, sex, bp, chol, na_to_k]],
                             columns=['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K'])
    user_data_scaled = scaler.transform(user_data)
    return user_data_scaled

user_input_scaled = get_user_input()
prediction = svm_model.predict(user_input_scaled)
print("\n Predicted Drug Recommendation:", prediction[0])
```

Output:

```
Age:  18
Sex (M/F):  M
BP (LOW/NORMAL/HIGH):  LOW
Cholesterol (NORMAL/HIGH):  HIGH
Na_to_K:  13.5

 Predicted Drug Recommendation: DrugY
```

Analysis

This model was developed to predict which drug should be prescribed to patients based on their medical attributes. The dataset used was drug200.csv, which includes features such as age, sex, blood pressure (BP), cholesterol level, sodium to potassium ratio, and the target variable, i.e., the drug prescribed.

1. Library Imports and Setup

- numpy, pandas for data manipulation,
- matplotlib, seaborn for data visualization,
- scikit-learn for machine learning models and metrics.

The project began with importing essential Python libraries such as NumPy and Pandas for data handling, Matplotlib and Seaborn for visualization, and various modules from scikit-learn for model building, preprocessing, and evaluation.

2. Data Loading and Exploration

The dataset was loaded using pandas, and basic exploratory functions like .head(), .tail(), .describe(), and .info() were used to understand the structure and distribution of data.

The dataset was then loaded using Pandas, and an initial analysis was performed using functions like .head(), .tail(), .info(), and .describe() to get an understanding of the data structure, data types, and statistical summary. It was observed that the dataset was clean with no missing values. To prepare the data for machine learning models, categorical variables such as 'Sex', 'BP', and 'Cholesterol' were encoded into numeric format using Label Encoder because machine learning algorithms work better with numerical input. Next, the features and target labels were separated. The input features were standardized using StandardScaler to bring all values to a similar scale, which helps algorithms like SVM and Logistic Regression to converge faster and perform better.

To understand the data distribution and relationships, several visualizations were generated using Seaborn and Matplotlib. This included pair plots, histograms, and correlation heatmaps. These visualizations helped in identifying patterns in the data, such as how certain drugs are more likely to be prescribed for patients with specific ranges of sodium-potassium levels or blood pressure.

3. Model Training

Three classification models were trained:

- **Logistic Regression**
- **Decision Tree Classifier**
- **Support Vector Classifier (SVC)**

The dataset was split into training and testing sets using an 80-20 split.

data exploration and preprocessing, the dataset was split into training and testing sets using the train_test_split() function with an 80-20 ratio. Three different machine learning models were implemented and trained on the data: Logistic Regression, Decision Tree Classifier, and Support Vector Classifier (SVC). Each model was fitted on the training data and then used to predict outcomes on the test set.

4. Model Evaluation

Each model was evaluated using:

- **Accuracy Score**
- **Confusion Matrix**
- **Classification Report (Precision, Recall, F1-Score)**
- **ROC Curve & AUC Score**

These evaluations helped compare model performance and identify which model best predicted the drug type.

The performance of each model was evaluated using standard classification metrics. These included accuracy score, confusion matrix, and a detailed classification report which provided precision, recall, and F1-score for each class. Additionally, the Receiver Operating Characteristic (ROC) curve and the Area Under Curve (AUC) score were plotted to visually assess each model's ability to distinguish between different classes. These metrics helped determine which model performed best for the given dataset.

This project demonstrates a classification task using Python and its popular machine learning libraries. The dataset used (drug200.csv) contains medical information about patients and the drug prescribed to them. The main goal is to predict the most suitable drug based on attributes like age, sex, blood pressure, cholesterol level, and sodium/potassium levels.

Conclusion

This project successfully implemented and compared three classification algorithms to predict the most suitable drug for patients based on their health metrics. Among Logistic Regression, Decision Tree, and SVC, the model with the highest accuracy and ROC-AUC score was considered the most effective. The results highlight the importance of proper data preprocessing and model selection in building effective machine learning solutions. This approach can be extended to other medical classification problems with similar structured data.