1. TASK-2: MOVIE RATING PREDICTION WITH PYTHON

Build a model that predicts the rating of a movie based on features like genre, director, and actors. You can use regression techniques to tackle this problem.
The goal is to analyze historical movie data and develop a model that accurately estimates the rating given to a movie by users or critics.
Movie Rating Prediction project enables you to explore data analysis, preprocessing, feature engineering, and machine learning modeling techniques. It provides insights into the factors that influence movie ratings and allows you to build a model that can estimate the ratings of movies accurately.

2. TASK-3: IRIS FLOWER CLASSIFICATION

The Iris flower dataset consists of three species: setosa, versicolor, and virginica. These species can be distinguished based on their measurements. Now, imagine that you have the measurements of Iris flowers categorized by their respective species. Your objective is to train a machine learning model that can learn from these measurements and accurately classify the Iris flowers into their respective species.Use the Iris dataset to develop a model that can classify iris flowers into different species based on their sepal and petal measurements. This dataset is widely used for introductory classification tasks.

3. TASK-4: SALES PREDICTION USING PYTHON

Sales prediction involves forecasting the amount of a product that customers will purchase, taking into account various factors such as advertising expenditure, target audience segmentation, and advertising platform selection.
In businesses that offer products or services, the role of a Data Scientist is crucial for predicting future sales. They utilize machine learning techniques in Python to analyze and interpret data, allowing them to make informed decisions regarding advertising costs. By leveraging these predictions, businesses can optimize their advertising strategies and maximize sales potential. Let's embark on the journey of sales prediction using machine learning in Python.

TASK-2

CODE:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset (replace 'movies.csv' with actual dataset file)
df = pd.read_csv('/content/IMDb Movies India.csv.zip',
encoding='latin1')  # Try 'iso-8859-1' if needed

# Display basic info and check for missing values
print(df.info())
print(df.isnull().sum())
```

```python
# Drop rows with missing values (if any)
df.dropna(inplace=True)

# Selecting relevant features
features = ['Genre', 'Director', 'Actors', 'Budget', 'Revenue']
target = 'Rating'
# Ensure column names are correct
df.columns = df.columns.str.strip()

# Print columns to check for mismatches
print("Dataset Columns:", df.columns.tolist())

# Rename if necessary (Example)
df.rename(columns={'Actor Names': 'Actors', 'Movie Budget': 'Budget'},
inplace=True)

# Handle missing columns
for col in ['Actors', 'Budget', 'Revenue']:
    if col not in df.columns:
        print(f"Warning: {col} not found in dataset. Filling with
default values.")
        df[col] = 0  # or df[col] = np.nan if missing

# Now, the script should work without the ValueError
# Selecting relevant features
features = ['Genre', 'Director', 'Actors', 'Budget', 'Revenue']
target = 'Rating'

# Ensure all columns exist
missing_cols = [col for col in features + [target] if col not in
df.columns]
if missing_cols:
    raise ValueError(f"Missing columns in dataset: {missing_cols}")
    df = df[features + [target]]
    # Convert numerical features to proper format
df['Budget'] = pd.to_numeric(df['Budget'], errors='coerce').fillna(0)
df['Revenue'] = pd.to_numeric(df['Revenue'], errors='coerce').fillna(0)
# Encode categorical variables
label_encoders = {}
for col in ['Genre', 'Director', 'Actors']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    label_encoders[col] = le
    # Splitting dataset
X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Standardizing numerical features
scaler = StandardScaler()
X_train.loc[:, ['Budget', 'Revenue']] =
scaler.fit_transform(X_train[['Budget', 'Revenue']])
X_test.loc[:, ['Budget', 'Revenue']] =
scaler.transform(X_test[['Budget', 'Revenue']])
# Train models
lr_model = LinearRegression()
```

```
lr_model.fit(X_train, y_train)
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Evaluate models
y_pred_lr = lr_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)
print("Linear Regression:")
print("MSE:", mean_squared_error(y_test, y_pred_lr))
print("R2 Score:", r2_score(y_test, y_pred_lr))

print("Random Forest Regressor:")
print("MSE:", mean_squared_error(y_test, y_pred_rf))
print("R2 Score:", r2_score(y_test, y_pred_rf))
# Visualizing feature importance
importances = rf_model.feature_importances_
feature_names = X.columns
sns.barplot(x=importances, y=feature_names)
plt.title("Feature Importance in Movie Rating Prediction")
plt.show()
```

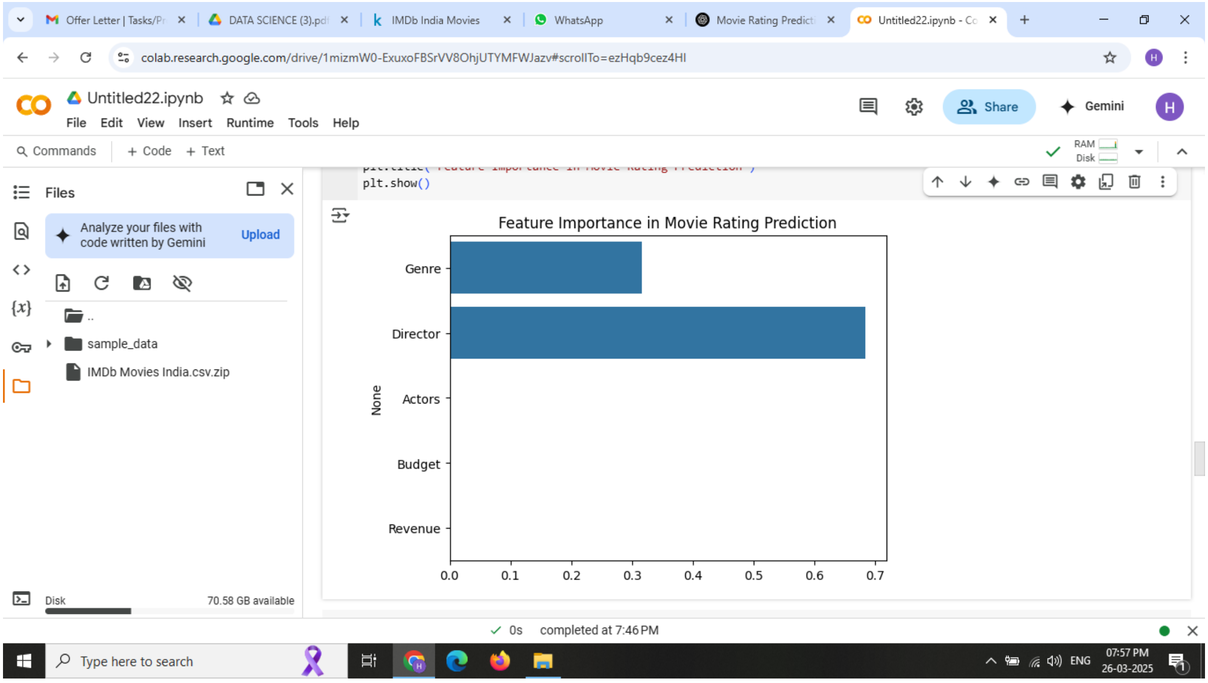
OUTPUT:
<class 'pandas.core.frame.DataFrame'> RangeIndex: 15509 entries, 0 to 15508 Data columns (total 10 columns): # Column Non-Null Count Dtype --- ------ -------------- ----- 0 Name 15509 non-null object 1 Year 14981 non-null object 2 Duration 7240 non-null object 3 Genre 13632 non-null object 4 Rating 7919 non-null float64 5 Votes 7920 non-null object 6 Director 14984 non-null object 7 Actor 1 13892 non-null object 8 Actor 2 13125 non-null object 9 Actor 3 12365 non-null object dtypes: float64(1), object(9) memory usage: 1.2+ MB None Name 0 Year 528 Duration 8269 Genre 1877 Rating 7590 Votes 7589 Director 525 Actor 1 1617 Actor 2 2384 Actor 3 3144 dtype: int64 Dataset Columns: ['Name', 'Year', 'Duration', 'Genre', 'Rating', 'Votes', 'Director', 'Actor 1', 'Actor 2', 'Actor 3'] Warning: Actors not found in dataset. Filling with default values. Warning: Budget not found in dataset. Filling with default values. Warning: Revenue not found in dataset. Filling with default values.

☑

RandomForestRegressor
?i

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

Linear Regression: MSE: 1.826585474995056 R2 Score: 0.013586206270336465 Random Forest Regressor: MSE: 1.9928715883810568 R2 Score: -0.07621354205520503



TASK_3

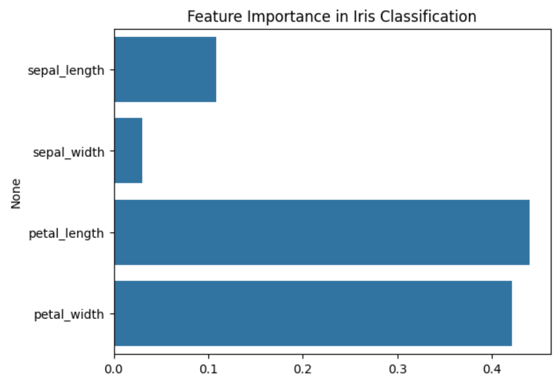
CODE:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
file_path = "/content/IRIS.csv"
df = pd.read_csv(file_path)

# Ensure the correct column name for species
target_column = 'species'  # Corrected to lowercase

# Define Features (X) and Target (y)
X = df.drop(columns=[target_column])  # All features except species
y = df[target_column]

# Encode target labels (if necessary)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Split dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardizing numerical features (optional)
```

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

# Train Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test,
y_pred))

# Feature importance visualization
importances = model.feature_importances_
feature_names = df.drop(columns=[target_column]).columns
sns.barplot(x=importances, y=feature_names)
plt.title("Feature Importance in Iris Classification")
plt.show()
```

OUTPUT:

```
Model Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



TASK-4

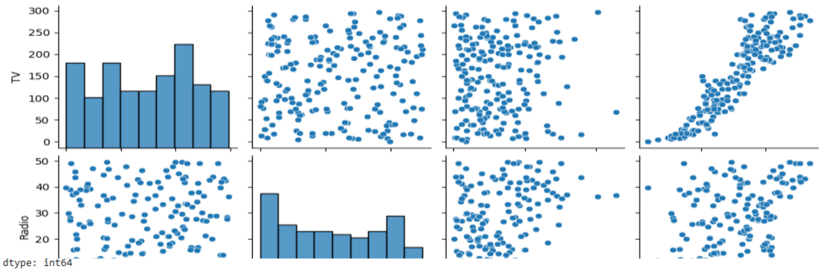
CODE:

```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
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
```

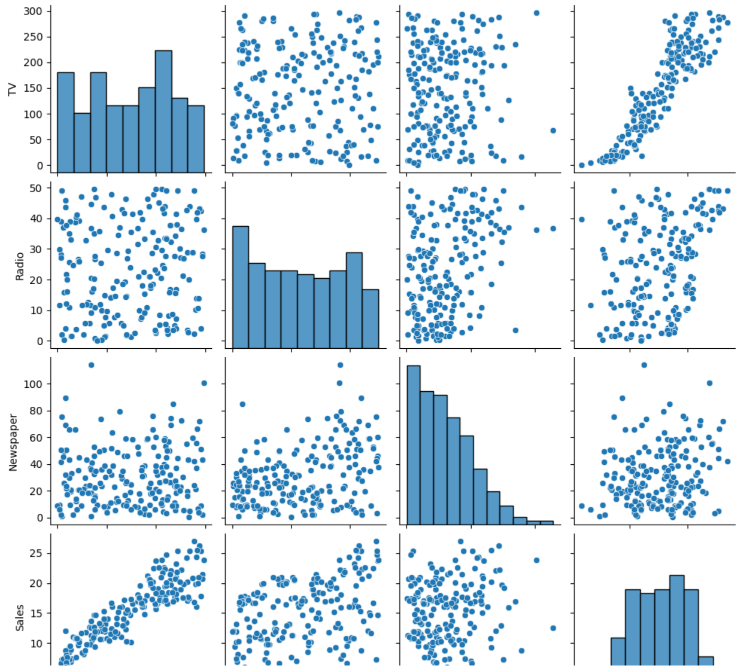
```python
df = pd.read_csv("/content/advertising.csv")

# Display dataset information
print(df.info())
print(df.describe())

# Checking for missing values
print("\nMissing Values:\n", df.isnull().sum())

# Visualizing data
sns.pairplot(df)
plt.show()

# Define Features (X) and Target (y)
X = df.drop(columns=['Sales'])  # Drop the target variable
y = df['Sales']  # Target variable

# Split the dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate model performance
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))

# Visualizing actual vs predicted sales
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.title("Actual vs Predicted Sales")
plt.show()
```

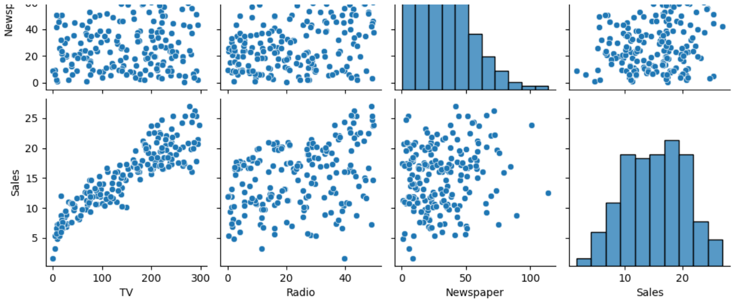
OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
None
               TV       Radio   Newspaper       Sales
count  200.000000  200.000000  200.000000  200.000000
mean   147.042500   23.264000   30.554000   15.130500
std     85.854236   14.846809   21.778621    5.283892
min      0.700000    0.000000    0.300000    1.600000
25%     74.375000    9.975000   12.750000   11.000000
50%    149.750000   22.900000   25.750000   16.000000
75%    218.825000   36.525000   45.100000   19.050000
max    296.400000   49.600000  114.000000   27.000000

Missing Values:
 TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```



dtype: int64






```
Mean Squared Error: 2.9077569102710896
R2 Score: 0.9059011844150826
```