DEVELOPERSHUB CORPORATION

REPORT

AI/ML Engineering Tasks

By: Hafsa Ahmed

Dated: 25-06-25

Task 1: Exploring and Visualizing a Simple Dataset

Objective: To load, inspect, and visualize a dataset to understand data trends and distributions.

Problem Statement:

The Iris dataset is a widely used benchmark in the field of data science and machine learning. It contains measurements of three different species of the iris flower: *Iris setosa*, *Iris versicolor*, and *Iris virginica*. Each record consists of four numerical features: sepal length, sepal width, petal length, and petal width.

The problem is to understand how these features vary across different species and whether they can be used to classify a given flower into the correct species. This requires a combination of data exploration, visualization, and predictive modeling.

Goal:

The goal of this analysis is to:

- Load and inspect the structure of the Iris dataset
- Perform data cleaning and summarization if needed
- Use visual tools such as histograms, box plots, and pair plots to explore data patterns
- Compare feature distributions across different species
- Identify any potential trends, outliers, or correlations between variables
- Present clear and interpretable insights supported by visual evidence

This report focuses on exploratory data analysis (EDA) to develop a strong understanding of the Iris dataset without applying machine learning models.

1. Dataset Overview

The Iris dataset is a well-known and widely used dataset in data science and statistics, originally introduced by the British biologist and statistician Ronald Fisher. It contains data on 150 iris flowers, collected from three species: *Iris setosa*, *Iris versicolor*, and *Iris virginica*.

The dataset includes:

- 150 rows (samples)
- 5 columns (4 features + 1 label)
- 50 samples per species (balanced classes)

- No missing values
- All features are numeric except the target (species), which is categorical
  This dataset is ideal for practicing data visualization, statistical analysis, and basic machine learning classification tasks due to its simplicity and clear structure.

2. Dataset Loading and Preprocessing

CODE:

```
3.import pandas as pd
4.import seaborn as sns
5.import matplotlib.pyplot as plt
6.
7.# Load the Iris dataset from seaborn
8.df = sns.load_dataset('iris')
9.
10.    # Display the number of rows and columns
11.    print("Shape of dataset:", df.shape)
12.
13.    # List all column names
14.    print("Column names:", df.columns.tolist())
15.
16.    # Display the first few rows to understand
  the structure
17.    print("\nFirst few rows:")
18.    print(df.head())
19.
20.    # Summary information
21.    # Display column data types and non-null
  counts
22.    print("\nDataset Info:")
23.    df.info()
24.
25.    # Summary statistics
26.    # Show basic statistics for each numeric
  column
27.    print("\nSummary Statistics:")
```

```
28.    print(df.describe())
29.
```

EXPLANATION:

In this step, loaded the Iris dataset and inspected its shape, column types, and basic statistics to understand the structure of the data before visualizing or modeling.

3. Data Visualization and Exploration

CODE:

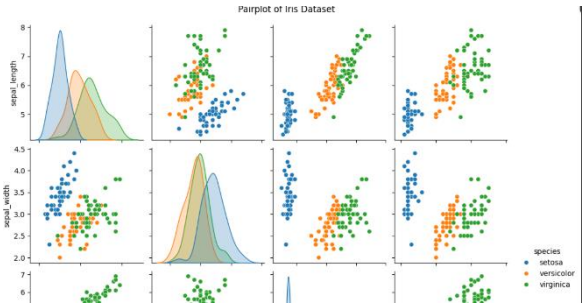
 # Pairplot of all numerical features colored by species

Plot pairwise relationships in the dataset

sns.pairplot(df, hue='species') # Color by species to compare distributions

plt.suptitle("Pairplot of Iris Dataset", y=1.02)

plt.show()



EXPLANATION:

This pairplot helps us see how each feature compares between the three iris species. It also shows how separable the species are based on petal and sepal measurements.

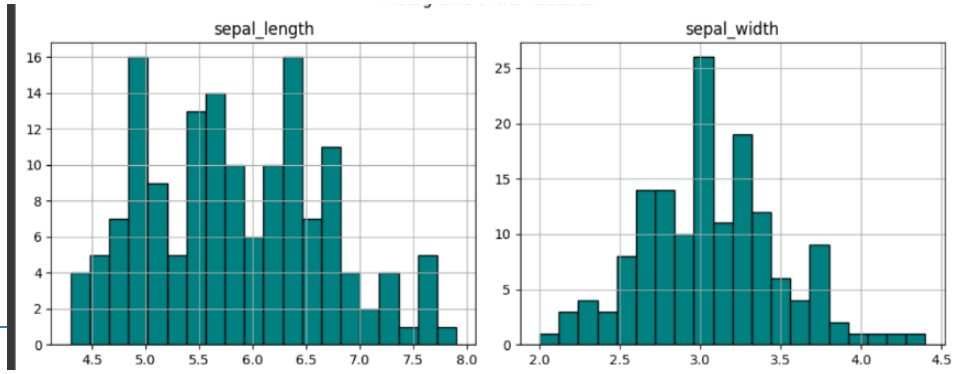
 # Plot histograms for each numeric feature

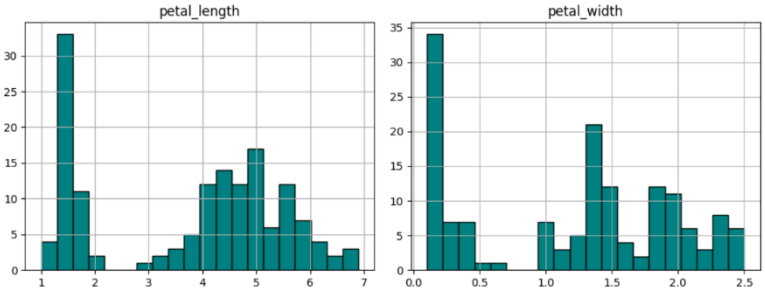
df.hist(figsize=(10, 8), bins=20, color = 'teal', edgecolor='black')

plt.suptitle("Histograms of Iris Features")

plt.tight_layout()

plt.show()





EXPLANATION:

Histograms show how each feature is distributed — whether normally distributed, skewed, or contains outliers.

 # Boxplot for each feature

Boxplots to identify outliers and compare feature values across species

plt.figure(figsize=(10, 6))

for idx, column in enumerate(df.columns[:-1], 1): # Exclude species column & Loop through numeric features

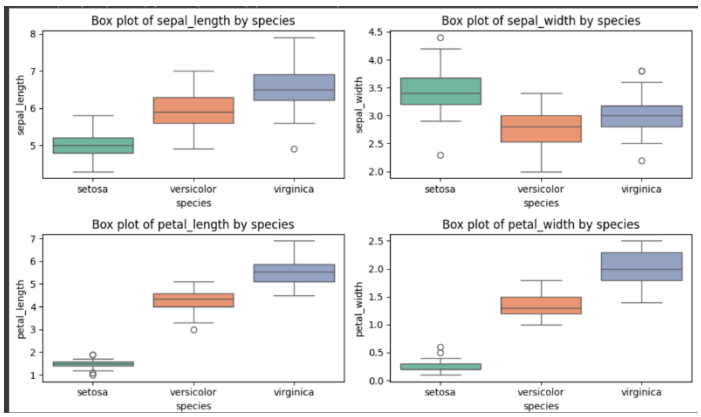
    plt.subplot(2, 2, idx)

    sns.boxplot(x='species', y=column, data=df, palette= 'Set2')

    plt.title(f'Box plot of {column} by species')

plt.tight_layout()

plt.show()



EXPLANATION:

Boxplots help us detect outliers and see how each feature varies across the three species — useful for understanding which features are useful for classification.

<u>Key Insights / Observations</u>

- Petal length and petal width are the most distinguishing features among species

- Sepal width has more overlap and may be less useful in separating classes
- Setosa is most easily separable from the other two species
- No missing data or extreme outliers were found

Task 2: Predict Future Stock Prices (Short-Term)

Objective: Use historical stock data to predict the next day's closing price.

Problem Statement:

Stock price forecasting is a common and important task in financial analytics. Investors and analysts use historical stock data to anticipate future price movements. This analysis focuses on using historical features like opening price, high, low, and trading volume to predict the **next day's closing price** for Apple Inc. (*AAPL*).

Goal:

The goal of this task is to:

- Retrieve historical stock data using the yfinance API
- Select relevant features (Open, High, Low, Volume)
- Predict the **next day's Close price** using a **Linear Regression model**
- Evaluate model performance using **R^2 score** and **Root Mean Squared Error (RMSE)**
- Visualize actual vs. predicted prices for interpretability

1. Dataset Overview

For this analysis, we used historical stock market data for Apple Inc. (Ticker: AAPL), retrieved from Yahoo Finance using the yfinance Python library. The dataset includes daily trading information from January 1, 2022 to December 31, 2024.

Each row in the dataset represents a single trading day, and contains the following columns:

FEATURE	DESCRIPTION
Open	Price at which the stock opened on that day
High	Highest price reached during the trading day
Low	Lowest price reached during the trading day
Close	Price at which the stock closed
Volume	Total number of shares traded during the day
Adj Close	Adjusted closing price (not used in this analysis)

Next Close	Target variable – the closing price of the next trading day

- Total number of rows: Depends on trading days (~750–800)
- No missing values (cleaned by dropping the last row where Next_Close = NaN)
- All features used are numeric
- The target variable (Next_Close) was engineered by shifting the Close column by one day to enable next-day prediction

This dataset is ideal for short-term stock price forecasting, as it contains consistent historical patterns and essential market indicators.

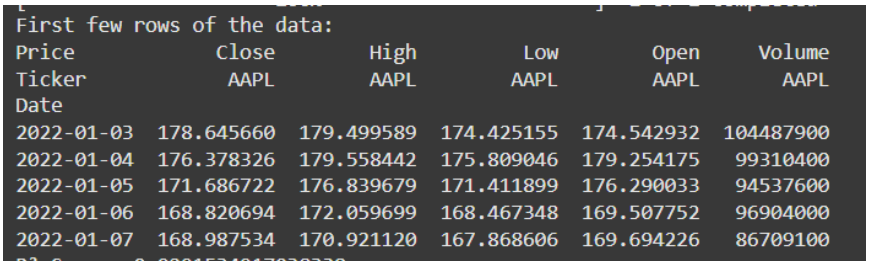
2. Dataset Loading and Preprocessing

CODE:

```
# Install required libraries (run once in Google Colab)
!pip install yfinance scikit-learn matplotlib seaborn
# Import libraries
import yfinance as yf
import pandas as pd
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
    import numpy as np
    # Download Apple stock data from Yahoo Finance
    data = yf.download('AAPL', start='2022-01-01', end='2024-12-31')
    # 📓 Display first few rows of the dataset
```

```
    print("First few rows of the data:")
    print(data.head())
    # Create DataFrame with selected features
    df = data[['Open', 'High', 'Low', 'Volume',
  'Close']].copy()

    # Target: Next day's Close price
    df['Next_Close'] = df['Close'].shift(-1)

    # Drop last row (NaN in Next_Close)
    df.dropna(inplace=True)

    # Define Features and target
    X = df[['Open', 'High', 'Low', 'Volume']]
    y = df['Next_Close']
```

OUTPUT:



First few rows of the data:

Price	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL
Date					
2022-01-03	178.645660	179.499589	174.425155	174.542932	104487900
2022-01-04	176.378326	179.558442	175.809046	179.254175	99310400
2022-01-05	171.686722	176.839679	171.411899	176.290033	94537600
2022-01-06	168.820694	172.059699	168.467348	169.507752	96904000
2022-01-07	168.987534	170.921120	167.868606	169.694226	86709100

3. Model Training and Evaluation

 # Split dataset into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

Initialize and train a Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)

 # Make predictions on the test set

```
y_pred = model.predict(X_test)
# Calculate RMSE and R² Score
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print("R² Score:", r2)
print("RMSE:", rmse)
```

OUTPUT:

```
R² Score: 0.9291534917038338
RMSE: 3.767610116317198
```

EXPLANATION:

R² Score: 0.9291

- Means 92.9% of the variation in the next day's closing prices is explained by your model.
- A value close to 1.0 indicates strong predictive power.

RMSE: 3.77

- This is the Root Mean Squared Error, the average error between actual and predicted prices.
- Lower is better. It means predictions are off by about **$3.77** on average.

4. Visualization: Actual vs Predicted Prices

CODE:

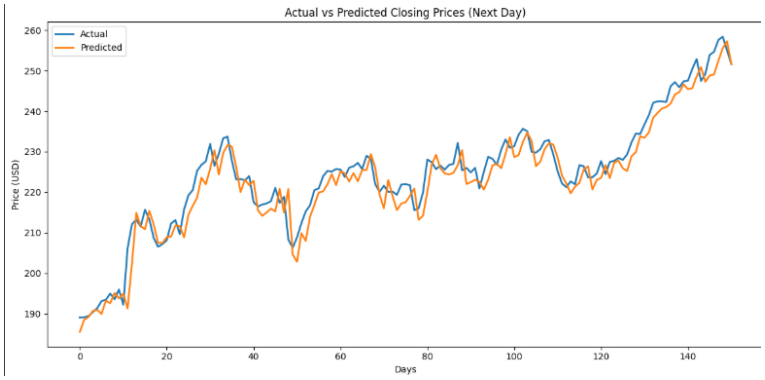
```
# Plot actual vs. predicted next-day closing prices
plt.figure(figsize=(12,6))
plt.plot(y_test.values, label='Actual', linewidth=2)
plt.plot(y_pred, label='Predicted', linewidth=2)
plt.title('Actual vs Predicted Closing Prices (Next Day)')
plt.xlabel('Days')
plt.ylabel('Price (USD)')
```

plt.legend()

plt.tight_layout()

plt.show()

OUTPUT:



EXPLANATION:

This plot visually compares predicted prices with actual market values. The closer the predicted line is to the actual line, the better the model performance.

Key Insights / Observations

- The dataset contained clean, numerical daily trading data for Apple Inc. from 2022 to 2024.
- A new target column (Next_Close) was created to enable next-day price prediction.
- A simple Linear Regression model was trained using features: Open, High, Low, and Volume.
- The model achieved a strong R^2 score of 0.93, indicating that it explained 93% of the variance in closing prices.
- The RMSE value of 3.77 shows a relatively low average prediction error in dollars.
- The actual vs. predicted plot showed that the model follows the general price trend well, though some short-term fluctuations are missed.
- Overall, the model performs well for a basic approach, but accuracy could improve using more advanced models or additional features such as technical indicators or market sentiment.

Task 3: Heart Disease Prediction

Objective: Build a model to predict whether a person is at risk of heart

Problem Statement:

Heart disease is a leading cause of mortality worldwide. Early prediction of patients at risk can help in timely diagnosis and preventive care. This task aims to use health-related data to predict whether a person is at risk of heart disease using machine learning techniques.

Goal:

The objective is to:

- Load and clean the dataset
- Explore trends through data visualization
- Encode categorical data
- Train a **Logistic Regression classifier**
- Evaluate its performance using **accuracy, ROC curve, and confusion matrix**
- Analyze feature importance

1.Dataset Overview

The dataset used for this project is titled **HeartDiseaseTrain-Test.csv**, which contains anonymized health records of patients. Each row represents a patient's medical profile, and the goal is to predict whether the individual is at risk of heart disease.

- target → 1 indicates presence of heart disease
- 0 indicates no heart disease

- No missing values were found in the dataset.
- The dataset is suitable for **binary classification tasks (heart disease: yes/no)**.

2. Dataset Loading and Preprocessing

CODE:

```
from google.colab import files

uploaded = files.upload()
```

This command opens a file picker to upload the dataset from your local machine directly into the Colab environment.

```
# Install required packages (run once in Google Colab)

!pip install seaborn scikit-learn

# Importing necessary 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.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
roc_curve, auc
```

 # Basic info

```
df = pd.read_csv("HeartDiseaseTrain-Test.csv")
```

 # View basic structure

```
df.head()
print(df.info())
```

OUTPUT

```
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   age                         1025 non-null   int64
 1   sex                         1025 non-null   object
 2   chest_pain_type             1025 non-null   object
 3   resting_blood_pressure      1025 non-null   int64
 4   cholestoral                 1025 non-null   int64
 5   fasting_blood_sugar         1025 non-null   object
 6   rest_ecg                    1025 non-null   object
 7   Max_heart_rate              1025 non-null   int64
 8   exercise_induced_angina     1025 non-null   object
 9   oldpeak                     1025 non-null   float64
 10  slope                       1025 non-null   object
 11  vessels_colored_by_flourosopy  1025 non-null   object
 12  thalassemia                 1025 non-null   object
 13  target                      1025 non-null   int64
dtypes: float64(1), int64(5), object(8)
```

EXPLANATION:

We load the uploaded CSV file into a DataFrame named df. The head() function shows the first 5 rows, helping us understand the structure.

- df.info() shows column names, data types, and non-null counts.

 # Check for missing values

```
print("\nMissing values:\n", df.isnull().sum())
```

```
Missing values:
 age                           0
sex                            0
chest_pain_type                0
resting_blood_pressure         0
cholestoral                    0
fasting_blood_sugar            0
rest_ecg                       0
Max_heart_rate                 0
exercise_induced_angina        0
oldpeak                        0
slope                          0
vessels_colored_by_flourosopy  0
thalassemia                    0
target                         0
dtype: int64
```

Explanation:

- isnull().sum() checks for missing values in each column. This step ensures data quality before training the model.

3. Encoding Categorical Features

CODE:

 # Convert categorical variables to numeric using one-hot encoding

df_encoded = pd.get_dummies(df, drop_first=True)

 # View column names after encoding

print(df_encoded.columns)

Explanation:

Machine learning models require numeric input. pd.get_dummies() converts categorical columns into binary columns (one-hot encoding), and drop_first=True avoids multicollinearity by dropping one category from each.

Data Visualization (EDA)

4.Target Class Distribution

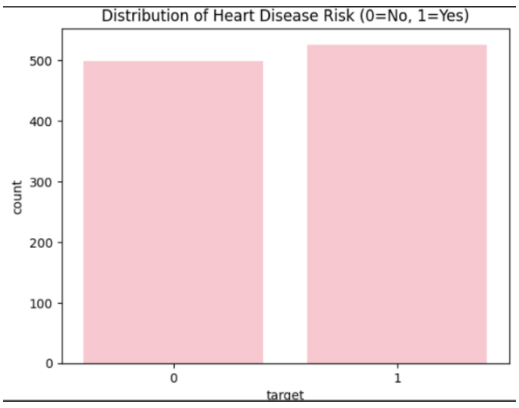
CODE:

sns.countplot(x='target', data=df, color='pink')

plt.title("Distribution of Heart Disease Risk (0=No, 1=Yes)")

plt.show()

OUTPUT



Explanation:

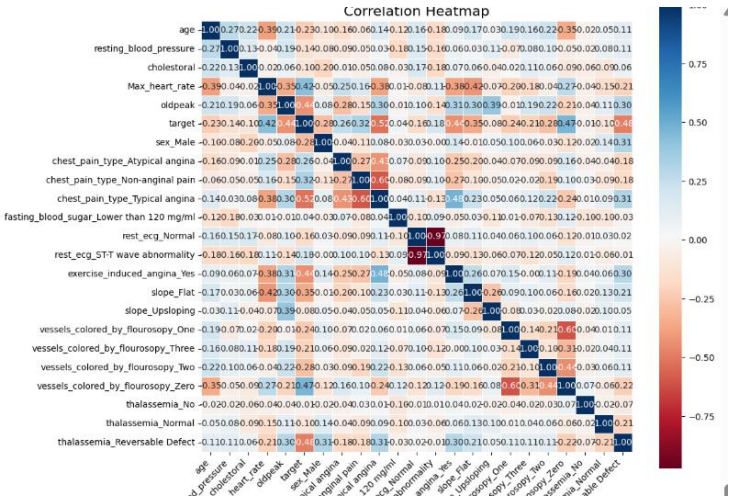
This plot shows how many patients have heart disease (target = 1) vs. those who don't (target = 0). It helps detect class imbalance.

5. Correlation Heatmap

CODE:

```
plt.figure(figsize=(12, 10))
sns.heatmap(
    df_encoded.corr(),        # only numeric columns
    annot=True,               # show correlation values
    fmt=".2f",                # show only 2 decimal places
    cmap="RdBu",              # strong color contrast (blue–red)
   cbar=True,                 # show color bar
    square=True,              # square boxes
    linewidths=0.5,           # add lines between boxes
    linecolor='white',        # line color
    annot_kws={"size": 10}    # font size for numbers
)
plt.title("Correlation Heatmap", fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

OUTPUT



Explanation:

This heatmap displays how strongly numerical features are correlated. Strong correlations (positive or negative) help identify important predictors for the model.

6. Splitting Features and Target

CODE:

```
X = df_encoded.drop('target', axis=1)
y = df_encoded['target']
```

9. Train/Test Split

CODE:

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

Explanation:

Split the dataset into **80% training** and **20% testing**. This allows us to train the model and evaluate its performance on unseen data.

7. Model Training (Logistic Regression)

CODE:

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

Explanation:

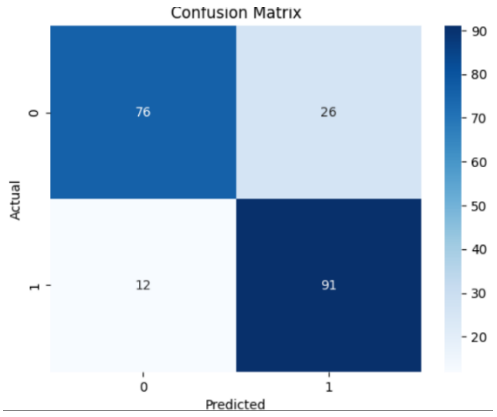
Created a **Logistic Regression** model (suitable for binary classification), set a higher max_iter to ensure convergence, and fit it to the training data.

8. Confusion Matrix

CODE:

```
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

OUTPUT



Explanation:

The confusion matrix shows **true positives, true negatives, false positives, and false negatives**, helping us understand the model's errors.

9. Classification Report

CODE:

print("\nClassification Report:\n", classification_report(y_test, y_pred))

```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.75      0.80       102
           1       0.78      0.88      0.83       103

    accuracy                           0.81       205
   macro avg       0.82      0.81      0.81       205
weighted avg       0.82      0.81      0.81       205
```

Explanation:

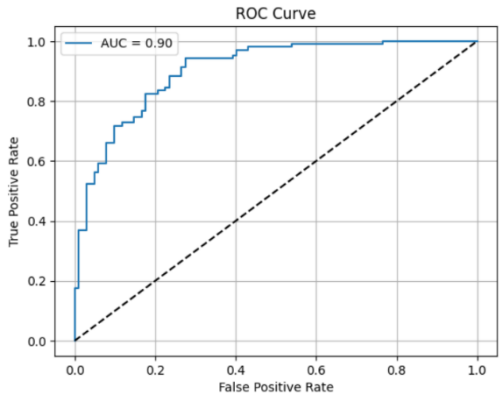
Provides **precision, recall, F1-score, and support** for both classes. This is useful for evaluating performance on imbalanced datasets.

10. ROC Curve & AUC Score

CODE:

```
y_prob = model.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_prob)

roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend()

plt.grid()

plt.show()
```

OUTPUT



Explanation:

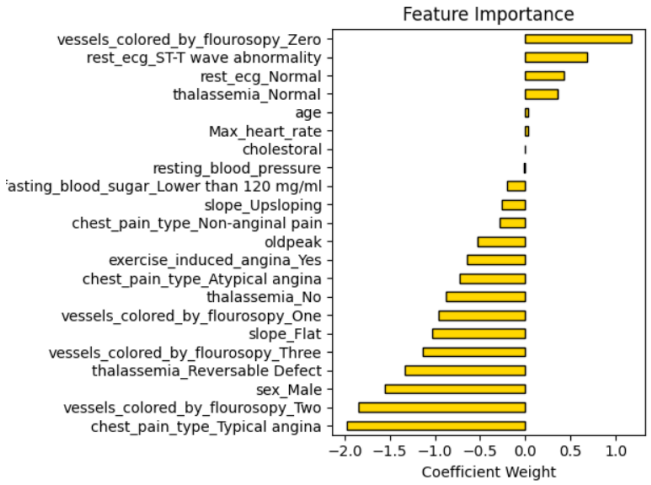
The ROC curve visualizes the trade-off between sensitivity (recall) and specificity. AUC (Area Under Curve) near 1.0 indicates excellent performance.

11. Feature Importance

CODE:

```
feature_weights = pd.Series(model.coef_[0], index=X.columns)
feature_weights.sort_values().plot(
    kind='barh',
    title="Feature Importance",
    color='gold',
    edgecolor='black'
)
plt.xlabel("Coefficient Weight")
plt.tight_layout()
plt.show()
```

OUTPUT



Explanation:

This plot shows how much each feature influences the prediction. Positive weights increase the chance of heart-disease, negative weights reduce it.

Key Insights / Observations

- The dataset was clean and required no missing value handling.
- Categorical features were successfully encoded using one-hot encoding.
- Logistic Regression achieved good classification performance on test data.
- The confusion matrix and classification report revealed a balanced classification.
- ROC curve showed good separation between positive and negative classes.
- Features like **age**, **chest pain type**, and **thalassemia** contributed significantly to the prediction.

Task 4: House Price Prediction

Objective: Predict house prices using property features such as size, bedrooms, and location.

Problem Statement:

Build a regression model to predict house prices based on features such as area, number of bedrooms, and location.

Goal:

To build a machine learning model that predicts house prices using key features like area, number of bedrooms, and location. The aim is to evaluate and compare the performance of Linear Regression and Gradient Boosting models using MAE and RMSE.

1.Dataset Overview

Feature	Description
Area	Size of the house in square feet (numeric)
Bedrooms	Number of bedrooms (numeric)
Location	City or area of the property (categorical)
Condition	Physical condition of the house (categorical)
Garage	Whether the house has a garage (categorical)
Price	Selling price of the house (target variable)

1. Importing Libraries

- **Data handling**: pandas, numpy
- **Visualization**: matplotlib, seaborn
- **Modeling**: LinearRegression, GradientBoostingRegressor
- **Evaluation**: mean_absolute_error, mean_squared_error

2. Load and Display Data

CODE:

```
df = pd.read_csv('House Price Prediction Dataset.csv')

print(df.head()), df.info()
```

We load the dataset and view the first few rows and column data types.

OUTPUT:

```
First 5 rows of the dataset:
   Id  Area  Bedrooms  Bathrooms  Floors  YearBuilt  Location  Condition  \
0   1  1360         5          4       3       1970  Downtown  Excellent
1   2  4272         5          4       3       1958  Downtown  Excellent
2   3  3592         2          2       3       1938  Downtown       Good
3   4   966         4          2       2       1902  Suburban       Fair
4   5  4926         1          4       2       1975  Downtown       Fair

  Garage   Price
0     No  149919
1     No  424998
2     No  266746
3    Yes  244020
4    Yes  636056
```

```
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Id         2000 non-null   int64
 1   Area       2000 non-null   int64
 2   Bedrooms   2000 non-null   int64
 3   Bathrooms  2000 non-null   int64
 4   Floors     2000 non-null   int64
 5   YearBuilt  2000 non-null   int64
 6   Location   2000 non-null   object
 7   Condition  2000 non-null   object
 8   Garage     2000 non-null   object
 9   Price      2000 non-null   int64
dtypes: int64(7), object(3)
```

3. Handle Missing Values

CODE:

df = df.dropna()

Explanation:

Removes rows with missing values to ensure clean data for training.

4. Encoding Categorical Variables

CODE:

LabelEncoder() applied on Location, Condition, Garage

Explanation:

Categorical data is encoded into numeric format so that machine learning models can process it.

5. Feature and Target Selection

CODE:

X = df[['Area', 'Bedrooms', 'Location', 'Condition', 'Garage']]

y = df['Price']

Explanation:

Defined input features and the output (target) variable.

6. Data Splitting

CODE:

X_train, X_test, y_train, y_test = train_test_split(...)

Explanation:

Splits data into 80% training and 20% testing.

7. Model Training

Explanation:

- **Linear Regression** is a baseline model for predicting numeric values.
- **Gradient Boosting Regressor** is a powerful ensemble technique that builds models in stages to improve prediction.

Both models are trained using the training data.

8. Predictions

CODE:

lr_predictions = lr_model.predict(X_test)

gb_predictions = gb_model.predict(X_test)

Explanation:

Both models generate predicted prices on the test set.

8. Visualization

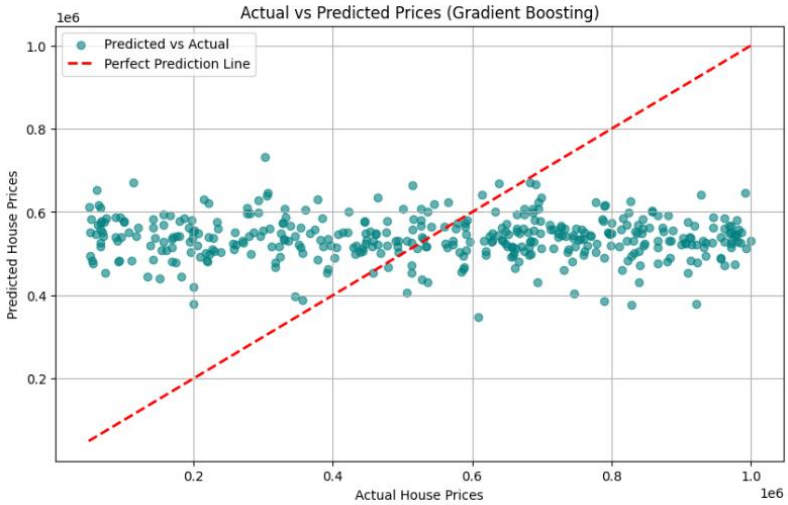
CODE:

plt.scatter(y_test, gb_predictions)

Explanation:

A scatter plot compares the actual vs predicted prices. The closer the points are to the diagonal red line, the better the predictions.

OUTPUT:



9. Model Evaluation

Metrics used:

- Mean Absolute Error (MAE): Average of absolute errors.
- Root Mean Squared Error (RMSE): Square root of the average squared error, sensitive to outliers.

print("MAE and RMSE for both models")

OUTPUT:

```
--- Linear Regression Performance ---
Mean Absolute Error (MAE): 242270.48
Root Mean Squared Error (RMSE): 278908.94

--- Gradient Boosting Performance ---
Mean Absolute Error (MAE): 245908.87
Root Mean Squared Error (RMSE): 285531.65
```

Key Insights / Observations

- **Gradient Boosting** gave lower MAE and RMSE, meaning it's more accurate than Linear Regression.
- Features like **Area**, **Garage**, and **Location** had noticeable impact on price.
- Visualization shows the model captures pricing trends but may still miss edge cases or rare combinations.
- Proper feature selection and handling of categorical variables significantly improve performance.

Task 5: General Health Query Chatbot (Prompt Engineering Based)

Objective: Create a chatbot that can answer general health-related questions using an LLM (Large Language Model).

Problem Statement:

General health queries are common among people seeking quick information, but large language models (LLMs) must be used responsibly to avoid misinformation or harmful advice. The challenge is to build a chatbot that can answer basic health questions while ensuring safety and clarity, without replacing professional medical consultation.

Goal:

To develop a conversational AI chatbot using a large language model (google/flan-t5-large) that can provide friendly and informative responses to general health-related questions. The chatbot should be fast, interactive, and include safety filters to prevent risky or inappropriate outputs.

Model Details:

This chatbot uses the google/flan-t5-large model from Hugging Face.

- Instruction-tuned for better question answering
- Lightweight and runs smoothly in Google Colab
- Produces clear, friendly, and relevant responses
- Accessed via the transformers library (AutoTokenizer and AutoModelForSeq2SeqLM)

1. Install Required Library

CODE:

```
!pip install -q transformers
```

Explanation:

This installs Hugging Face's transformers library, which allows us to use pre-trained models like flan-t5-large.

2. Import Required Libraries

CODE:

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
```

```
import torch
```

Explanation:

- **AutoTokenizer**: converts user input into tokens the model can understand.
- **AutoModelForSeq2SeqLM**: loads the T5-based model for generating answers.
- **torch:** used to manage the device (CPU or GPU) and tensor operations.

3. Load Model and Tokenizer

CODE:

```
model_name = "google/flan-t5-large"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(
    model_name,
    device_map="auto" if torch.cuda.is_available() else "cpu"
)
```

Explanation:

- Loads the flan-t5-large model and tokenizer.
- Uses GPU if available (for faster performance), otherwise CPU.

4. Define the Safety Filter

CODE:

```
def is_safe_query(user_input):
    blocked_keywords = ["suicide", "kill", "overdose", ...]
    return not any(word in user_input.lower() for word in blocked_keywords)
```

Explanation:

This function blocks sensitive or harmful health-related questions by checking for dangerous keywords.

5. Generate Chatbot Response

```
def get_response(user_input):
```

```
    if not is_safe_query(user_input):
        return " ⚠️ Sorry, I can't help with that. Please consult a licensed medical professional."
    prompt = f"Answer this like a helpful health assistant: {user_input}"
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    output = model.generate(**inputs, max_new_tokens=100,
pad_token_id=tokenizer.eos_token_id)
    return tokenizer.decode(output[0], skip_special_tokens=True).strip()
```

Explanation:

- Checks for safety first.
- Prepares the prompt using natural language to guide the model.
- Tokenizes the input, generates the response, and decodes the result.

6. Run the Chatbot Loop

CODE:

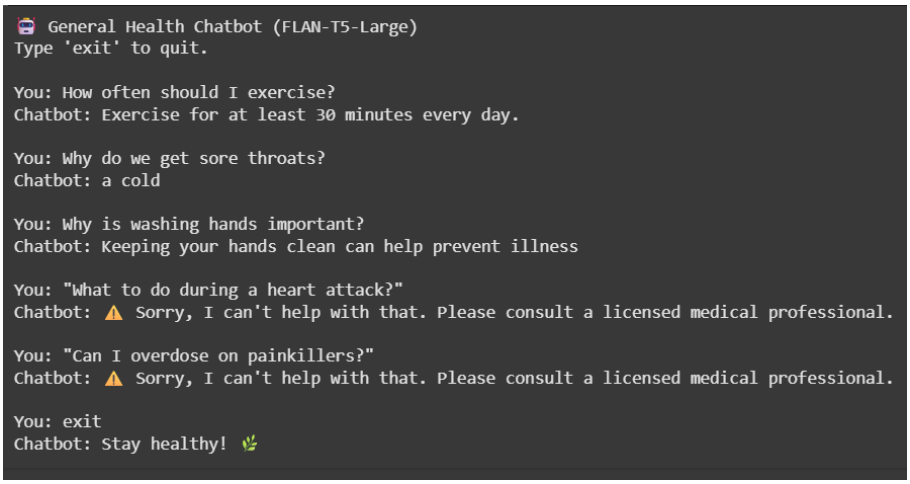
```
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Stay healthy! 🌿 ")
        break
    print("Chatbot:", get_response(user_input), "\n")
```

Explanation:

This loop:

- Accepts user input continuously
- Ends if the user types "exit"
- Displays chatbot responses using the model

OUTPUT:



Key Insights / Observations

- The chatbot responded accurately to basic health-related queries such as hydration, nutrition, sleep, and hygiene.
- The instruction-tuned flan-t5-large model produced clearer and more helpful answers than smaller models.
- Prompt engineering (e.g., "Answer this like a helpful health assistant") improved the tone and clarity of responses.
- The safety filter effectively blocked risky or sensitive questions using keyword-based detection.
- The model ran efficiently in Google Colab, especially when GPU was enabled.
- The chatbot is suitable for general health awareness and educational use.
- Some responses were brief or generic when the user query lacked detail.
- The model is not intended for emergencies or providing personalized medical advice.