This project covers data analysis, stock and health predictions, house price modeling, and building AI chatbots for health and empathy, demonstrating practical machine learning and responsible AI deployment.

Tasks Report

AI/ML Internship Tasks,
Developerhub Corporation

Maria Samreen

Task 1: Exploring and Visualizing a Simple Dataset

1. Objective

The primary objective of this task was to perform an exploratory data analysis (EDA) on the IRIS dataset using Python libraries such as **pandas**, **seaborn**, and **matplotlib**. The aim was to understand the structure, distribution, and relationships of various features in the dataset, particularly in relation to the species classification.

---

2. Dataset Description

- **Dataset Used:** `IRIS.csv`
- **Shape of Dataset:** 150 rows × 5 columns
- **Columns:** `sepal_length`, `sepal_width`, `petal_length`, `petal_width`, `species`
- The dataset consists of 4 numeric features and 1 categorical target variable (`species`) with three distinct classes.

---

3. Methodology

3.1 Data Loading and Inspection

- Loaded the dataset using `pandas.read_csv()`.
- Inspected dataset shape, column names, top 5 entries, data types, and statistical summaries using:
  - `df.shape`
  - `df.columns`
  - `df.head()`
  - `df.info()`
  - `df.describe()`

3.2 Scatter Plot Visualization

- Created a scatter plot for `sepal_length` vs. `sepal_width`, colored by species, using `seaborn.scatterplot()`.
- Added relevant title and axis labels to interpret the visual.

3.3 Histogram Analysis

- Used `df.hist()` to visualize the distributions of all numeric features in a grid layout.
- Supplemented with KDE-enhanced histograms using `seaborn.histplot()` for each numeric column individually to show smoother distribution curves.

3.4 Boxplot Analysis

- Generated boxplots for each numeric feature, grouped by the `species` category, using `seaborn.boxplot()`.
- These visualizations helped identify differences in feature distribution across species and potential outliers.

---

4. Results and Observations

- **Feature Distributions:**
  - Petal features (`petal_length` and `petal_width`) show clear separation between species.
  - Sepal features show more overlap between species.
- **Species Differences:**
  - `Iris-setosa` is clearly separable from other species, particularly in petal-related measurements.
  - `Iris-versicolor` and `Iris-virginica` show some overlap but still present measurable differences.
- **Boxplots:**
  - Highlighted the variance and median differences among species for each feature.
  - Helped identify the range and interquartile spread for each feature within species.

---

5. Conclusion

The exploratory data analysis provided valuable insights into the IRIS dataset. Visualization techniques such as scatter plots, histograms, KDE plots, and boxplots were instrumental in understanding the relationships between features and species classifications. These visual patterns can assist in future tasks such as classification model building or feature selectio

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

1. Objective

The goal of this task was to develop a simple machine learning model to predict short-term stock prices, specifically the **next day's closing price** of a selected stock. The project utilized historical data from Yahoo Finance and applied a **Random Forest Regressor** for prediction.

---

2. Tools and Libraries

The following Python libraries were used:

- `yfinance` for downloading historical stock data
- `pandas` for data manipulation
- `scikit-learn` for machine learning modeling and evaluation
- `matplotlib` for data visualization

---

3. Data Acquisition

- **Stock Selected:** Apple Inc. (`AAPL`)
- **Time Range:** January 1, 2020 – December 31, 2024
- Historical data included daily stock prices and volume:
  - **Features Used:** `Open`, `High`, `Low`, `Volume`
  - **Target Variable:** Next day's `Close` price

---

4. Data Preparation

- Created a new column `Target` by shifting the `Close` price by one day.
- Removed the final row due to the resulting `NaN` in the `Target` column.
- Selected features relevant to short-term price movement.
- Split the dataset into **training (80%)** and **testing (20%)** subsets without shuffling to preserve time-series integrity.

---

5. Model Training

- **Model Used:** `RandomForestRegressor` with 100 estimators and a fixed random seed for reproducibility.
- Trained the model on the historical dataset using the training split.

---

6. Model Evaluation

- **Metric:** Mean Squared Error (MSE) on the test dataset.
- **Result:** The MSE was printed to assess how close the predictions were to actual values. (Exact value depends on the runtime environment.)
- A line plot was generated comparing:
    - Actual closing prices
    - Predicted closing prices using the Random Forest model

---

7. Future Price Prediction

- Used the most recent available data point to predict the **next day's closing price**.
- The predicted price was printed with two decimal places for readability.

---

8. Key Observations

- The Random Forest model provides a smooth approximation of the next-day closing price based on key trading features.
- The approach is straightforward but does not incorporate advanced time series modeling techniques or external market factors.
- Model performance can be sensitive to:
    - Feature selection
    - Market volatility
    - Lookahead bias and overfitting

Task 3: Heart Disease Prediction

1. Objective

The goal of this project was to build a predictive model to identify the presence of heart disease based on patient health attributes. The binary classification task aimed to differentiate patients **with** and **without** heart disease using clinical and demographic data.

---

2. Tools and Technologies Used

- **Programming Language:** Python
- **Libraries:**
  - `pandas`, `numpy` for data handling
  - `matplotlib`, `seaborn` for data visualization
  - `scikit-learn` for preprocessing, model training, and evaluation

---

3. Dataset Overview

- **Source:** `heart_disease_uci.csv`
- **Initial Checks:**
  - Verified dataset structure with `.info()` and `.head()`
  - Checked and handled missing values using median for numeric features and mode for categorical ones
  - Dropped non-informative columns like `id` and `dataset`

---

4. Data Cleaning and Preprocessing

- **Missing Values:**
  - Imputed numerical columns (`trestbps`, `chol`, `thalch`, `oldpeak`, `ca`) with their **median**
  - Imputed categorical columns (`fbs`, `restecg`, `exang`, `slope`, `thal`) with their **mode**
- **Encoding:**
  - Used `LabelEncoder` to convert categorical variables to numeric form
- **Target Variable:**
  - Converted multi-class target `num` into **binary classification**:
    - `0` = No Heart Disease

- `1` = Heart Disease Present (`num` > 0)

---

5. Exploratory Data Analysis (EDA)

- **Target Distribution:**
  - Visualized with `sns.countplot`, showed class imbalance favoring non-disease cases
- **Feature-Target Relationship:**
  - Plotted chest pain type (`cp`) vs target, indicating strong influence of chest pain type on disease presence
- **Correlation Matrix:**
  - Used `seaborn.heatmap` to highlight relationships among numeric variables

---

6. Feature Engineering and Scaling

- Defined features (`X`) and target (`y`)
- Applied **StandardScaler** to normalize the feature set before model training
- Split dataset into **training (80%)** and **testing (20%)** sets using `train_test_split`

---

7. Model Development

Logistic Regression

- Trained a logistic regression model with `max_iter=1000` for convergence
- Model predicted the binary target based on scaled features

---

8. Model Evaluation

- **Accuracy:** Evaluated using `accuracy_score`
- **Confusion Matrix:** Showed counts of TP, FP, TN, and FN
- **ROC-AUC:**
  - Computed AUC score using `roc_auc_score`
  - Plotted ROC curve to visualize trade-off between true and false positive rates
  - Achieved an AUC indicating good model performance (exact score printed during execution)

---

9. Feature Importance

- Derived from **logistic regression coefficients**
- Visualized using `seaborn.barplot`
- Notable important features (high absolute coefficient values):
  - `cp` (chest pain type)
  - `thalach` (maximum heart rate)
  - `oldpeak` (ST depression)
  - `exang` (exercise-induced angina)

---

10. Key Takeaways

- Logistic Regression provided interpretable results and reasonable predictive performance.
- Features related to chest pain, exercise response, and heart rate were the most influential.
- The model is suitable for preliminary screening or risk stratification in clinical settings.

Task 4: General Health Query Chatbot (Prompt Engineering Based)

1. Objective

The objective of this task was to develop a **safe and responsible health chatbot** using the **Mistral-7B-Instruct-v0.1** language model. The chatbot was designed to provide **general health-related information**, while actively avoiding specific **medical advice or prescriptions** to ensure ethical use and user safety.

---

2. Technologies and Tools Used

- **Model:** `mistralai/Mistral-7B-Instruct-v0.1` from Hugging Face
- **Libraries:**
  - `transformers` (for tokenizer and model pipeline)
  - `huggingface_hub` (for authentication and model access)
  - `torch` (for model loading and device management)
- **Hardware Assumption:** GPU-enabled system (due to model size and `device_map="auto"` usage)

---

3. Workflow Overview

Step 1: Hugging Face Authentication

- User logs in using a **Hugging Face token** to access gated model weights.
- Token must be securely stored and not hardcoded in production environments.

Step 2: Load Model and Tokenizer

- Loaded **AutoTokenizer** and **AutoModelForCausalLM** for the `Mistral-7B-Instruct-v0.1` model.
- Model is configured to use **float16 precision** for optimized GPU memory usage and performance.

Step 3: Build Text Generation Pipeline

- Created a text generation pipeline using `transformers.pipeline()` for streamlined inference.

Step 4: Prompt Engineering

- Constructed a tailored prompt:
    - **Persona:** A cautious, friendly medical assistant
    - **Restrictions:** Avoid giving diagnosis or treatment suggestions
    - This prompt helps condition the model to respond ethically within the defined scope.

Step 5: Safety Filtering

- Implemented a **basic keyword-based filter** to detect potentially unsafe or sensitive queries:
    - Filters out words such as "dosage", "pill", "treat", "mg", etc.
    - Responds with a warning and refuses to answer when such terms are detected

Step 6: Chat Interface

- Developed a simple command-line interface allowing users to ask health-related questions.
- Incorporated:
    - Exit condition
    - Real-time model inference
    - Output cleaning to isolate the answer portion from the generated text

---

4. Features

- **Safe Query Filtering:** Basic but effective safeguard against inappropriate medical queries.
- **Contextual Response Generation:** Leveraging Mistral's instruction-tuned design for consistent and helpful outputs.
- **User-Friendly Interface:** Command-line based chatbot suitable for demonstrations or local testing.
- **Modular Design:** Key components (prompt generation, safety checks, pipeline loading) are reusable and adaptable.

---

5. Limitations

- **Safety Filter is Rule-Based:** Relies solely on keyword matching, which may miss nuanced unsafe questions.
- **No Persistence or Logging:** The system does not log conversations, which limits reviewability or improvement tracking.

- **Resource-Intensive:** The Mistral-7B model requires significant GPU memory, which may not be feasible for all users.
- **No Web Interface:** The interface is restricted to the terminal; no deployment to web or app layers is included.

Task 5: Mental Health Support Chatbot (Fine-Tuned)

1. Objective

This project aimed to build an **empathetic chatbot** by fine-tuning a **DistilGPT-2** language model on the **EmpatheticDialogues** dataset. The goal was to generate emotionally aware responses appropriate for mental health support, while avoiding clinical or diagnostic roles.

---

2. Tools and Technologies Used

- **Dataset:** `empathetic_dialogues` from Hugging Face Datasets
- **Model:** `distilgpt2` (a distilled version of GPT-2)
- **Libraries:**
  - `transformers` (by Hugging Face) for model fine-tuning and generation
  - `datasets` for loading and managing the dataset
  - `torch` for tensor operations and inference

---

3. Dataset Overview

- **EmpatheticDialogues** is a benchmark dataset for training models to generate emotionally appropriate responses.
- The dataset consists of over 24,000 one-to-one conversations grounded in emotional situations.

---

4. Model Preparation

Model Selection

- Used `distilgpt2` for efficiency and speed, balancing capability and resource usage.

Tokenizer Initialization

- Used the matching `AutoTokenizer` for `distilgpt2` to tokenize user input and training data.

---

5. Training Setup

Tokenization (Implicit Step)

- Although not shown in the code snippet above, the dataset must be tokenized before training (assumed to be stored as `tokenized_dataset`).

Data Collator

- Used `DataCollatorForLanguageModeling` with `mlm=False` since GPT-2 is a causal language model and does not support masked language modeling.

Training Configuration

- **Epochs:** 2
- **Batch Size:** 4 (train and eval)
- **Evaluation Strategy:** Once per epoch
- **Logging:** Logs written to `./logs`
- **Save Directory:** `./empathetic_chatbot`
- **Reporting:** Disabled external integrations (`report_to=[]`)

Trainer API

- Managed training using Hugging Face's `Trainer` class for streamlined fine-tuning.

---

6. Model Saving

- Post-training, both the model and tokenizer were saved to the local directory `empathetic_distilgpt2` for future use in inference and deployment.

---

7. Chatbot Deployment

Chat Function

- Implemented a simple interactive chatbot using the fine-tuned model.
- Flow:
    - Takes user input from the terminal
    - Tokenizes and sends it to the model

    - Generates a response with `model.generate()`
    - Decodes and returns a cleaned response

Design Philosophy

- The chatbot is designed for **empathy and listening**, not medical advice.
- Responses aim to be supportive and emotionally attuned, leveraging training on emotionally rich dialog data.

---

8. Limitations

- **Tokenization Not Shown:** Tokenization and preprocessing of the dataset (`tokenized_dataset`) must be included explicitly.
- **Short Context Window:** DistilGPT2 has a smaller context window, which limits multi-turn memory.
- **No Safety Filter:** No explicit filtering to detect unsafe content or suicidal ideation. (Critical for real-world mental health apps.)
- **Model Biases:** The model may reflect biases present in the training data or the pre-trained GPT-2 base.

Task 6: House Price Prediction

1. Objective

The objective of this project was to develop a machine learning model to predict **house prices** based on various features of residential properties. The model leverages regression techniques to learn from historical housing data and estimate sale prices of new or unseen properties.

---

2. Tools and Libraries Used

- **Programming Language:** Python
- **Libraries:**
  - `pandas`, `numpy` for data manipulation
  - `matplotlib`, `seaborn` for visualization
  - `scikit-learn` for preprocessing, modeling, and evaluation

---

3. Dataset Overview

- **Source:** `Housing.csv`

- **Initial Inspection:**
  - Loaded and explored data using `.head()`, `.info()`, `.describe()`
  - Identified 12 feature columns and 1 target column (`price`)

Key Features:

- **Numeric:** `area`, `bedrooms`, `bathrooms`, `stories`, `parking`
- **Binary Categorical (yes/no):** `mainroad`, `guestroom`, `basement`, `hotwaterheating`, `airconditioning`, `prefarea`
- **Multi-class Categorical:** `furnishingstatus`
- **Target Variable:** `price` (in local currency units)

---

4. Data Cleaning and Preparation

- Checked for missing values using `.isnull().sum()`
- **Handled Missing Values:**
  - Numeric columns filled with **median**
  - Categorical columns filled with **mode**
- Ensured no missing values remained before model training

---

5. Feature Engineering

- Defined the feature matrix (`X`) and target variable (`y`)
- Constructed a **preprocessing pipeline** using `ColumnTransformer`:
  - Scaled numeric features with `StandardScaler`
  - Converted binary "yes/no" features to 1/0 using a custom function
  - Applied `OneHotEncoder` to `furnishingstatus`

---

6. Model Development

- Split the dataset using `train_test_split` (80% train, 20% test)
- Built a **pipeline** consisting of:
  - Preprocessing steps
  - `LinearRegression` model
- Trained the model using the training data

---

7. Prediction and Evaluation

- Used the trained model to predict prices on the test set
- **Performance Metrics:**
  - **Mean Absolute Error (MAE):** Measures average magnitude of errors
  - **Root Mean Squared Error (RMSE):** Penalizes larger errors more heavily

Evaluation Results:

- **MAE:** Reasonably low, suggesting good average prediction performance
- **RMSE:** Provides a clearer picture of overall model error
  (Exact values are printed during execution and depend on data variability)

---

8. Visualization

- Plotted **Actual vs Predicted Prices** using a scatter plot
- Included a red dashed diagonal line to indicate ideal (perfect) prediction
- The clustering of points around the diagonal line confirmed reasonable accuracy and model fit

---

9. Key Observations

- The linear regression model captured underlying trends in the data well, though:
  - Outliers and non-linear relationships may limit its precision
  - Binary and categorical encoding was handled effectively via pipeline
- Predictive performance was acceptable for initial modeling, but improvement is possible