Crude Oil Price Prediction Documentation

1. Project Overview

1.1 Objective

The objective of this project is to develop a machine learning model capable of predicting crude oil prices based on historical data. This helps in understanding price trends and supporting informed decision-making in industries reliant on oil pricing.

1.2 Scope

- Prediction of crude oil prices for the next day using time-series data.
- Implementation using a Long Short-Term Memory (LSTM) neural network.
- Deployment of the prediction system via a web application using Flask.

1. 3 Problem Statement

Crude oil price fluctuations significantly impact global economies and industries. Predicting these prices accurately is critical for businesses, governments, and investors. This project aims to develop a machine learning-based system to forecast crude oil prices using historical data, facilitating better decision-making and planning.

Key Challenges:

- High volatility in crude oil prices due to economic and geopolitical factors.
- Identifying patterns in time-series data for accurate short-term predictions.
- Building a scalable and user-friendly application for practical deployment.

---

2. Data Description

2.1 Data Source

- Dataset: Crude Oil Prices Daily

- Source: Kaggle
- Format: CSV
- Columns:
  - `Date`: The date of the record.
  - `Closing Price`: The daily closing price of crude oil.

2.2 Data Preprocessing

1. Importing Libraries

Key libraries such as **NumPy**, **Pandas**, **Matplotlib**, and **Seaborn** were imported for handling data, visualization, and statistical analysis.

2. Importing the Dataset

The dataset was loaded using **Pandas**, ensuring the CSV file was correctly read into a DataFrame. The dataset structure was examined using functions like head(), info(), and describe().

3. Analyzing the Data

- **Inspection**: Checked for irregularities such as missing values, duplicates, and outliers.
- **Statistical Summary**: Generated basic statistics to understand the dataset distribution.

4. Handling Missing Data

- Missing entries were identified and addressed by either:
  - Imputing missing values using the previous day's price.
  - Dropping rows with significant data inconsistencies.

5. Feature Scaling

- **Min-Max Scaling**: Normalized the "Closing Price" to a range of 0-1 using the formula:

  $$xscaled=max(x)−min(x)x−min(x)$$

- $max(x)−min(x)x−min(x)$ This step ensured the model performed well during gradient descent.

6. Data Visualization

- Visualized crude oil price trends over time to identify patterns and trends using **Matplotlib** and **Seaborn**:
  - Line plots for overall trends.
  - Box plots to detect outliers.
  - Histograms to view distribution.

7. Splitting Data into Train and Test Sets

- Split the dataset into:
  - **Training Set**: 80% of the data for model learning.
  - **Testing Set**: 20% of the data for evaluating performance.

8. Creating a Dataset with a Sliding Window

- Transformed the data into a supervised learning format using a sliding window of n_steps = 10. This step involved:
  - Creating sequences of 10 days of closing prices as input features.
  - Assigning the next day's price as the corresponding label.

2.3 How the Data Supports Model Building

Each step in **Model Building** utilizes the preprocessed data as follows:

Importing the Model Building Libraries

- The data prepared in the preprocessing stage is passed to machine learning libraries, such as TensorFlow/Keras, for model construction and training.

Initializing the Model

- The model requires data formatted as (samples, n_steps, features) to handle sequential patterns. Here, n_steps = 10 and features = 1 (closing price).

Adding LSTM Layers

- The sliding window dataset structure (10-day sequences as input, next-day price as output) allows the LSTM layers to learn temporal dependencies, capturing patterns in price fluctuations over time.

Adding Output Layer

- The single feature of the dataset (Closing Price) determines that the output layer has one neuron, representing the predicted price.

Configure the Learning Process

- The normalized data (scaled between 0-1) minimizes the loss function (Mean Squared Error) during training, improving optimization.

Training the Model

- The dataset is split into training and validation sets:
  - **Training Set**: Helps the model learn patterns.
  - **Validation Set**: Evaluates performance and ensures the model generalizes well to unseen data.

Model Evaluation

- The test set (remaining 20% of the data) is used to evaluate the model. The preprocessed data ensures predictions can be compared directly with actual prices to compute metrics like MSE and RMSE.

Save the Model

- The trained model is saved to a file format, allowing it to be loaded later for making predictions without re-training. The data's sliding window structure ensures compatibility with the saved model's input format.

Test the Model

- The sliding window format of the test data allows the trained model to make predictions. The original scaling is applied back to the predicted prices to compare them with actual prices for evaluation.

2.4 How the Data Supports Application Building

The processed data plays a critical role in the functionality of the web application:

Create an HTML File

- The **HTML file** forms the frontend interface where users input crude oil prices for the last 10 days.

- Data Description Relevance:
    - Users must provide inputs in a format consistent with the model's requirements: 10 numerical values (representing 10 consecutive days' prices).
    - Input fields in the HTML are designed to validate these requirements.

Build Python Code

- The **Python backend** connects the frontend to the trained LSTM model.
- Data Description Relevance:
    - The backend ensures that the user inputs are formatted correctly and scaled to match the normalized data used during model training.
    - The backend accepts the user input (10 prices), reshapes it to (1, 10, 1) format, and passes it to the trained model for prediction.
    - After prediction, the output is inverse-scaled to return the result in the original price range.

---

3. Model Details

3.1 Architecture

- **Model Type:** LSTM (Long Short-Term Memory)
- **Framework:** TensorFlow/Keras
- **Input Shape:** `(n_steps, 1)` where `n_steps = 10`
- **Output:** A single value representing the predicted next day price.

3.2 Hyperparameters

- Number of LSTM layers: [e.g., 2]
- Number of neurons per layer: [e.g., 50]
- Batch size: [e.g., 32]
- Epochs: 10
- Loss function: Mean Squared Error (MSE)
- Optimizer: Adam

3.3 Model Training

- Training Data: [e.g., 80% of the dataset]
- Validation Data: [e.g., 20% of the dataset]
- Performance Metrics:
  - MSE: [Value]
  - RMSE: [Value]

---

4. Web Application

4.1 Technology Stack

- **Backend:** Flask (Python)
- **Frontend:** HTML, CSS, JavaScript (with TailwindCSS for styling)
- **Deployment:** http://127.0.0.1:5000/

4.2 Functionality

- **Input:**
  Users input crude oil prices for the last 10 days.
- **Output:**
  The application predicts and displays the crude oil price for the next day.

4.3 Workflow

1. User inputs the last 10 days' crude oil prices.
2. Input validation ensures proper data format.
3. The LSTM model processes the data and generates a prediction.
4. The web application displays the prediction.

---

5. Results

5.1 Key Insights

- The model was able to predict crude oil prices with reasonable accuracy (based on MSE/RMSE values).
- The prediction system demonstrated robust handling of noisy or incomplete data.

5.2 Limitations

- Performance is sensitive to the quality of historical data.
- Predictions are limited to short-term trends due to the model's design.

---

6. Future Improvements

- Incorporate external factors (e.g., geopolitical events, weather) to improve accuracy.
- Extend the model to predict for multiple days ahead.
- Deploy the application on a scalable cloud platform.

---

7. Usage Instructions

7.1 Prerequisites

- Python (version 3.8or later)
- Required Libraries: TensorFlow, Flask, NumPy, Pandas, etc.

---

8. Milestones & developer (contribution)

1. **Data Preparation** - developed by - **aryaman singh**
    - Timeframe: Week 1
    - Tasks: Data collection, cleaning, and preprocessing

2. **Model Development** - developed by - **viraj yadav**
    - Timeframe: Week 2–3
    - Tasks: Build, train, and tune the LSTM model.

3. **Application Development** - developed by - **Ansh Jaiswar**
    - Timeframe: Week 4
    - Tasks: Develop and test the Flask web application.

4. **Testing and Deployment** - developed by - **Vinit Pawar**
    - Timeframe: Week 5
    - Tasks: Test the complete system, deploy locally, and refine based on feedback.

-

---

9. Appendix

9.1 References & Pre-requisites

- Link to Dataset Source - https://www.kaggle.com/datasets/rockbottom73/crude-oil-prices
- **Anaconda Navigator - https://www.anaconda.com/download**
- https://www.youtube.com/watch?v=5mDYijMfSzs&embeds_referring_euri=https%3A%2F%2Fskillwallet.smartinternz.com%2F&source_ve_path=OTY3MTQ
- https://www.youtube.com/watch?v=akj3_wTploU&embeds_referring_euri=https%3A%2F%2Fskillwallet.smartinternz.com%2F&source_ve_path=OTY3MTQ
-

➢ **GITHUB REFERNCE**

https://github.com/ANSHJAISWAR/Crude-oil-Price-Prediction

9.2 Contributors

- **ANSH.D. JAISWAR** -[**LEADER**]
- ARYAMAN SINGH -[Team Member]
- VIRAJ YADAV -[Team Member]
- VINIT PAWAR -[Team Member]

9.3 Acknowledgments

Special thanks to resources and communities that supported this project, including online tutorials and forums.

---

10. Conclusion

The Crude Oil Price Prediction project successfully demonstrates the application of machine learning techniques to forecast short-term crude oil prices based on historical data. The use of a Long Short-Term Memory (LSTM) neural network enabled the model to capture temporal dependencies and trends in time-series data, achieving reasonable accuracy in predictions.

The integration of the model into a web application provides a user-friendly interface, allowing stakeholders to input historical prices and obtain next-day predictions effortlessly. This tool can aid decision-making for industries and investors by offering insights into potential price movements.

Key Achievements

The model effectively learned patterns in historical data, enabling accurate short-term price forecasting. The application handled real-time user input and provided quick, reliable predictions. The project highlighted the feasibility of deploying predictive models for practical use cases in economic and industrial contexts.

Limitations

The model's predictions are limited to short-term trends and do not account for external factors like geopolitical events or market shocks.The accuracy of predictions depends on the quality and completeness of the input data.

Future Directions

Incorporating external features, such as global economic indicators or news sentiment, could enhance prediction accuracy. Extending the model to predict long-term trends or multiple days ahead. Deploying the application on a scalable cloud platform to make it accessible to a wider audience.This project showcases the potential of machine learning in addressing real-world challenges, setting a strong foundation for future improvements and applications in financial forecasting.