ARTIFICIAL INTELLIGENCE

PROJECT REPORT

TITLE: SHARE PRICE ESTIMATION OF TOP 5 GPU COMPANIES

TEAM MEMBERS:

SAKSHI DHADWAD

GAURAV PATIL

HARSH PATIL

ALEX FERNANDES

Table of Content

Sl. No	Chapter	Page No
1	Introduction	3
2	Project Initialization and Planning Phase	4
3	Data Collection and Preprocessing Phase	9
4	Model Development Phase	15
5	Model Optimization and Tuning Phase	23
6	Results	27
7	Advantages & Disadvantages	29
8	Conclusion	30
9	Future Scope	30
10	Appendix	31

1. Introduction

1.1. Project overview

The project aims to develop a predictive model for estimating the future share prices of the top 5 GPU companies in the market. Leveraging historical stock price data, economic indicators, industry trends, and company financials, the project will provide insights into factors influencing performance and facilitate informed investment decisions. Beginning with data collection, historical stock prices and relevant economic and industry data will be gathered. Following data preprocessing to clean and standardize the data, feature engineering will enhance the predictive power of the model through the creation of additional features. Exploratory data analysis will then uncover relationships and trends, informing model selection. Machine learning or deep learning models such as linear regression, ARIMA, LSTM, or ensemble methods will be chosen for prediction. After training and evaluation, model performance will be optimized through hyperparameter tuning. The trained models will then forecast future stock prices, with analysis and interpretation identifying influential factors and implications.

1.2. Objective

The objective of this project is to estimate the share prices of the top 5 GPU companies in the market using historical data and current market trends. The aim is to develop a predictive model that can forecast the future prices of these companies based on their historical performance and other relevant factors. The model should take into consideration various economic indicators, industry trends, company financials, and other relevant data to make accurate predictions. The analysis should be performed using statistical and machine learning/ Deep learning techniques and the results should be presented in a clear and concise manner. The final output of the project will be a report outlining the predicted share prices of the top 5 GPU companies in the market, along with an analysis of the factors that have contributed to their performance.

2. Project Initialization and Planning Phase

2.1. Define Problem Statement

Problem Statement (PS)	PS-I	PS-II
I am	An Investor	An individual seeking to invest
I'm Trying To	Make informed investment decisions to maximize returns and minimize risks in the volatile GPU market.	Gain a competitive edge in the stock market by accurately forecasting the future performance of GPU companies and identifying profitable investment opportunities.
But	I struggle to accurately predict the future performance of GPU companies due to the complex and rapidly changing nature of the industry.	The lack of reliable prediction models tailored to the GPU market makes it challenging to anticipate share price movements with confidence.
Because	Historical data alone is insufficient to forecast share prices reliably, and existing prediction models often lack accuracy and robustness.	The GPU industry is influenced by various factors, including technological advancements, market demand, and competitive dynamics, which traditional forecasting methods struggle to capture effectively.
Which makes me feel	Frustrated and uncertain about the best course of action for my investment portfolio, leading to missed opportunities and potential losses in the market.	Anxious and apprehensive about making investment decisions, as I fear missing out on potential gains or experiencing losses due to inadequate predictive insights.

2.2. Project Proposal (Proposed Solution)

Project Overview	
Objective	Develop a predictive model using historical data and market trends to estimate share prices of the top 5 GPU companies, incorporating economic indicators and industry trends for accurate forecasting.
Scope	This project focuses on analyzing historical data and current market trends to predict share prices of the top 5 GPU companies, employing statistical and Machine learning techniques
Problem Statement	
Description	This project addresses the challenge of accurately predicting the share prices of the top 5 GPU companies by leveraging historical data and current market trends. Despite the availability of vast data, predicting stock prices remains complex due to market volatility and various influencing factors. By employing statistical and Machine learning techniques, the aim is to develop a robust predictive model that

	accounts for economic indicators, industry dynamics, and company performance to enhance investment decision-making and risk management strategies
Impact	The impact of this project extends to real-world investment practices by offering enhanced insights into the future performance of top GPU companies. Accurate share price predictions facilitate informed investment decisions, aiding investors in maximizing returns and minimizing risks. Additionally, the developed predictive model can serve as a valuable tool for financial analysts, portfolio managers, and individual investors, empowering them with actionable intelligence for optimizing investment strategies in the dynamic stock market environment.
Proposed Solution	
Approach	The approach involves collecting historical data and current market indicators of top GPU companies, preprocessing and analyzing the data, developing a predictive model using statistical and Machine learning techniques, and evaluating its accuracy for the share price estimation.
Key Features	Key features include data collection, preprocessing, analysis using statistical and Machine learning techniques, model evaluation, forecasting, interpretation of results, and clear reporting for stakeholders. Additionally, the project emphasizes accuracy, reliability, and actionable insights

Resource Requirement:

Resource Type	**Description**	**Specification/Allocation**
Hardware		
Computing Resources	CPU/GPU specifications, number of cores	2 x NVIDIA V100 GPUs
Memory	RAM specifications	8 GB
Storage	Disk space for data, models, and logs	1 TB SSD
Software		
Frameworks	Python frameworks	Flask
Libraries	Additional libraries	Sklearn, Prophet, Arima

Development Environment	IDE, version control	VS Code, Git
Data		
Data	Source, size, format	Kaggle dataset, 2.43 MB, 9 csv files

2.3. Initial Project Planning

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-1	Registration	SPPT5GC-1	As a user, I can register for my project by entering my email, and password, and confirming my password.	2	High	Sakshi Dhadwad	20-04-24	20-04-24
Sprint-1		SPPT5GC-5	Collect The Dataset	2	High	Sakshi Dhadwad		
Sprint-1		SPPT5GC-6	Importing The Libraries	1	High	Sakshi Dhadwad	20-04-2024	20-04-2024
Sprint-1		SPPT5GC-7	Read The Dataset	2	Low	Sakshi Dhadwad	20-04-2024	20-04-2024
Sprint-1		SPPT5GC-8	Data Preparation	2	Medium	Sakshi Dhadwad	20-04-2024	20-04-2024
Sprint-1		SPPT5GC-9	Checking For Missing Values & Handle missing values	1	High	Sakshi Dhadwad	20-04-2024	20-04-2024
Sprint-1		SPPT5GC-10	Data Manipulation	1	Medium	Harsh Patil	20-04-2024	20-04-2024
Sprint-1		SPPT5GC-11	Resampling The Data	1	Medium	Harsh Patil	20-04-2024	20-04-2024
Sprint-1		SPPT5GC-12	Merging And Splitting Data Into Test And	2	High	Harsh Patil	20-04-2024	20-04-2024

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
			Train Variables					
Sprint-2		SPPT5GC-13	Descriptive Statistical	1	Low	Harsh Patil	21-04-2024	22-04-24
Sprint-2		SPPT5GC-14	Visual Analysis	1	Low	Harsh Patil	21-04-2024	21-04-24
Sprint-3		SPPT5GC-15	Apply Linear Regression Model	2	Medium	Gaurav Patil	21-04-2024	23-04-2024
Sprint-3		SPPT5GC-16	Apply Decision Tree Regressor	2	Medium	Gaurav Patil	21-04-2024	23-04-2024
Sprint-3		SPPT5GC-17	Apply Extra Trees Regressor	2	Medium	Gaurav Patil	21-04-2024	23-04-2024
Sprint-3		SPPT5GC-18	Apply Random Forest Regressor	2	Medium	Gaurav Patil	21-04-2024	23-04-2024
Sprint-3		SPPT5GC-19	Apply Arima Model	2	Low	Gaurav Patil	21-04-2024	23-04-2024
Sprint-3		SPPT5GC-20	Apply Sarimax model	2	Low	Alex Fernandes	22-04-2024	25-04-2024
Sprint-3		SPPT5GC-21	Apply Prophet model	2	Low	Alex Fernandes	22-04-2024	25-04-2024
Sprint-3		SPPT5GC-25	Model Comparison and Evaluating Best Model	1	Medium	Alex Fernandes	22-04-2024	25-04-2024
Spirit-4		SPPT5GC-22	Building Html Pages	1	High	Alex Fernandes	22-04-2024	25-04-2024
Spirit-4		SPPT5GC-23	Build Python Code	2	High	Alex Fernandes	22-04-2024	25-04-2024

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Spirit-4	Login	SPPT5GC-24	Run The Application	2	High	Alex Fernandes	22-04-2024	25-04-2024

3. Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

Section	Description
Project Overview	This project demonstrates how machine learning can provide valuable insights from complex financial data to Predicting the closing Share prices of top 5 GPU Companies Objective Leveraging advanced analytical techniques to provide actionable insights into the future performance of the top GPU companies' share prices, facilitating informed investment decisions and strategic planning. The project aims to demonstrate proficiency in data analysis, machine learning, and financial modeling while addressing real-world forecasting challenges in the stock market domain.
Data Collection Plan	We will collect our dataset from the kaggle website
Raw Data Sources Identified	Source: Kaggle (https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices) Description: This dataset was obtained from Kaggle, a platform for data science and machine learning enthusiasts. It contains historical share price data for the top 5 GPU (Graphics Processing Unit) companies. File Format: CSV (Comma-Separated Values)

	Content: The dataset includes various attributes such as date, company name, opening price, closing price, highest price, lowest price, and trading volume.

Raw Data Sources Template:

Source Name	Description	Location/URL	Format	Size	Access Permissions
AMD (1980 -11.07.2023	The AMD dataset in which there are date, open, high, low, close, adj close and volume this columns from 1980 to 11-07-2023	https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices	CSV	729.52 kB	Public
AMD (2023 - 08.04.2024)	The AMD dataset in which there are date, open, high, low, close, adj close and volume this columns from 2023 to 08-04-2024	https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices	CSV	23.25 kB	Public
ASUS (2000 - 11.07.2023)	The ASUS dataset in which there are date, open, high,	https://www.kaggle.com/datasets/kapturovalexander/	CSV	431.14 kB	Public

	low, close, adj close and volume this columns from 2000 to 11-07-2023	nvidia-amd-intel-asus-msi-share-prices			
ASUS (2023 - 08.04.2024)	The ASUS dataset in which there are date, open, high, low, close, adj close and volume this columns from 2023 to 08-04-2024	https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices	CSV	22.11 kB	Public
INTEL (1980 - 11.07.2023)	The INTEL dataset in which there are date, open, high, low, close, adj close and volume this columns from 1980 to 11-07-2023	https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices	CSV	743.7 kB	Public
Intel (2023 – 08.04.2024)	The Intel dataset in which there are date, open, high, low, close, adj close and volume this columns from 2023 to 08-04-2024	https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices	CSV	22.17 kB	Public

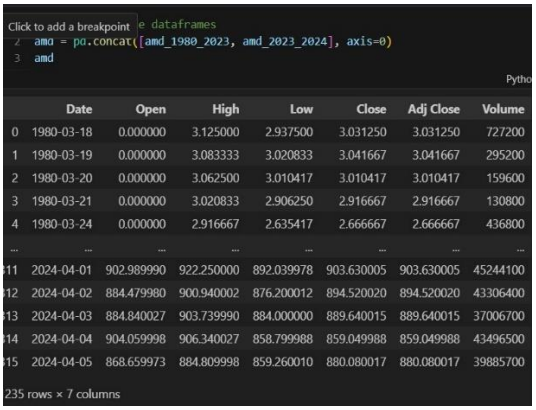
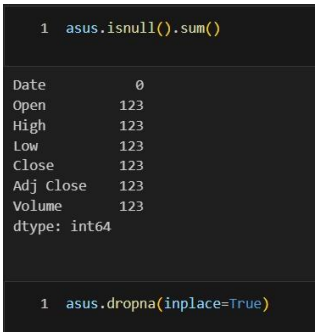
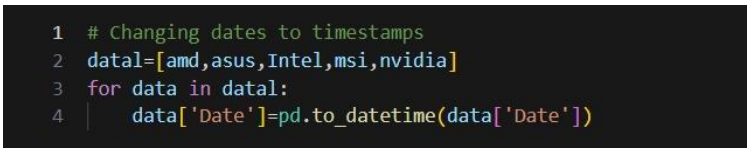
MSI(2023-08.04.2024)	The MSI dataset in which there are date, open, high, low, close, adj close and volume this columns from 2023 to 08-04-2024	https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices	CSV	22.1 kB	Public
NVIDIA(1999-11.07.2023)	The NVIDIA dataset in which there are date, open, high, low, close, adj close and volume this columns from 1999 to 11-07-2023	https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices	CSV	413.51 kB	Public
Nvidia(2023-08.04.2024)	The Nvidia dataset in which there are date, open, high, low, close, adj close and volume this columns from 2023 to 08-04-2024	https://www.kaggle.com/datasets/kapturovalexander/nvidia-amd-intel-asus-msi-share-prices	CSV	23.75 kB	Public

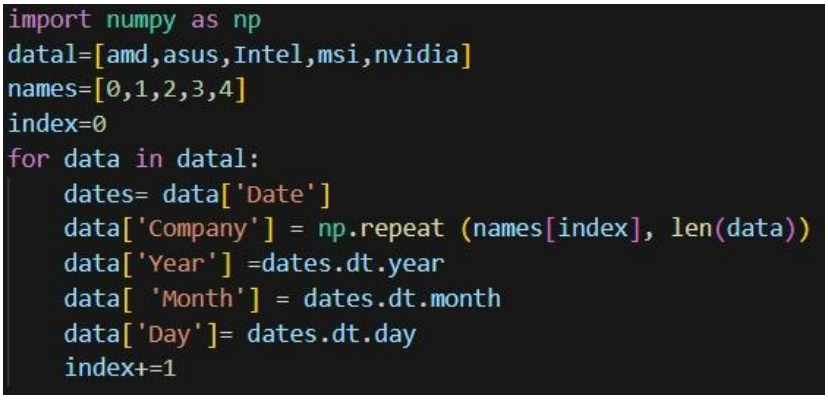
3.2. Data Quality Report

Data Source	Data Quality Issue	Severity	Resolution Plan
Kaggle	Two files for each company	Low	We will merge the two datasets into a new dataframe
Kaggle	We found some missing values in Asus dataset	Low	We will drop that all null values present in Asus dataset
Kaggle	Date column datatype is object	Moderate	We will convert object datatype into timestamp format

3.3. Data Exploration and Preprocessing

Section	Description
Data Overview	In description we applied df.describe and we have counted number of columns, mean, min, max, standard deviation Shape of datasets amd (11235, 7) asus (6167, 7) Intel (11235, 7) msi (298, 7) nvidia (6470, 7) For all datasets there are 7 columns which are date, open, high, low, close, adj close, volume and there datatypes are object, float and integers
Univariate Analysis	N/A

Bivariate Analysis	Code dynamically creates a grid of subplots based on the number of columns in the df_plot DataFrame, plots each time series on a separate subplot using Seaborn's lineplot(), and configures the appearance of the plot for readability and aesthetics. It's a concise and effective way to visualize multiple time series of stock data simultaneously.
Multivariate Analysis	N/A
Outliers and Anomalies	In the datasets of asus there are 123 outliers and we have handled the null values with the help of dropna function
Data Preprocessing Code Screenshots	
Loading Data	
Handling Missing Data	
Data Transformation	N/A
Feature Engineering	

Save Processed Data	

4. Model Development Phase

4.1. Feature Selection Report

Feature	Description	Selected (Yes/No)	Reasoning
Date	Gives the date of particular stock	Yes	This feature could be important for understanding trends over time. For example, you might see seasonal patterns in stock prices, or identify events that impacted the stock price
Open	the price at which a stock begins trading on a given day.	Yes	It can be helpful to see how the opening price compares to the closing price. Large differences between opening and closing price could indicate volatility in the market.
High	the highest price that a stock trades at during a given day	Yes	This data point can help you understand the range of prices that a stock is traded at within a day.

Low	the lowest price that a stock trades at during a given day.	Yes	This data point can help you understand the range of prices that a stock is traded at within a day.
Close	the price at which a stock ends trading on a given day.	Yes	This is a widely used metric for tracking stock prices.
Adj Close	This column reflects the closing price of a stock after accounting for corporate actions like stock splits and dividend payments.	No	Stock splits and dividends can significantly impact a stock's price. Using the "Adj Close" ensures your model considers the actual value a share held over time, leading to a potentially more accurate prediction.
Volume	This column represents the number of shares traded for a particular company's stock on a given day. It reflects the overall trading activity for that stock.	Yes	High volume can indicate increased interest in a stock, which might influence its price. Conversely, low volume could suggest a lack of investor confidence, potentially leading to stagnant or declining prices. By including "Volume" data, your model can account for these factors and potentially improve prediction accuracy.

4.2. Model Selection Report

Model	Description	Hyperparameters	Performance Metric
Linear Regression	Linear regression models the relationship between a dependent variable and independent variables by fitting a linear equation to observed data for prediction.	Hyperparameters used	MAE :- 1.0295797341476502 MSE :- 3.9646805791556625 RMSE:- 1.991150566671356 R2 Score:- 0.9998329252700594
Decision Tree	A decision tree is a predictive model that maps observations about an item to conclusions about its target value, by recursively partitioning data into subsets.	Hyperparameters used	MAE :- 5.768714122998442 MSE :- 953.7740005954626 RMSE:- 30.88323170582157 R2 Score:- 0.9598072201801939
Extra Trees Model	Extra Trees, or Extremely Randomized Trees, is an ensemble learning technique that builds multiple decision trees and aggregates their predictions to improve accuracy and reduce overfitting.	Hyperparameters used	MAE :- 4.9105214587100345 MSE :- 1035.0286791401684 RMSE:- 32.17186160513825 R2 Score:- 0.9563830846910345
Random Forest	Random Forest is an ensemble learning technique that constructs multiple	Hyperparameters used	MAE :- 5.600888524166072 MSE :- 1035.8351393370608 RMSE:- 32.18439279118158 R2 Score:- 0.9563490998297288

	decision trees and combines their predictions to enhance accuracy and mitigate overfitting.		
Prophet Model	Prophet is a forecasting tool developed by Facebook that models time series data with additive components and incorporates seasonality and holiday effects.	Hyperparameters not used	Mean Absolute Error (MAE): 195.73297116012097 Mean Squared Error (MSE): 48206.04630780599 Root Mean Squared Error (RMSE): 219.55875365788992 R-squared (R2) Score: -1.031440366400611
ARIMA Model	ARIMA (Autoregressive Integrated Moving Average) is a time series forecasting method that models the next step in the sequence based on linear combinations of past observations and forecast errors.	Hyperparameters not used	Mean Absolute Error (MAE): 96.90709354098094 Mean Squared Error (MSE): 25382.386804348866 Root Mean Squared Error (RMSE): 159.3185074131341 R-squared (R2) Score: -0.4138728147447801
SARIMAX Model	SARIMAX (Seasonal Autoregressive Integrated Moving Average with Exogenous Factors) is a statistical model used for time series forecasting, incorporating seasonality, trends, and external variables.	Hyperparameters not used	Mean Absolute Error (MAE): 109.06895324833107 Mean Squared Error (MSE): 32756.718955805904 Root Mean Squared Error (RMSE): 180.9881735246972 R-squared (R2) Score: -0.38039366955694187

4.3. Initial Model Training Code, Model Validation and Evaluation Report

Initial Model Training Code:

```
# linear regression

lr=LinearRegression()
lr.fit(x_train,y_train)
lr_pred=lr.predict(x_test)

mae = mean_absolute_error(y_test,lr_pred)
mse = mean_squared_error(y_test,lr_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,lr_pred)
print("Linear Regression Model Evaluation:")
print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE:- {rmse}\n R2 Score :- {r2}")
```

```
# decision tree

DT=DecisionTreeRegressor()
DT.fit(x_train,y_train)
DT_pred=DT.predict(x_test)

mae = mean_absolute_error(y_test,DT_pred)
mse = mean_squared_error(y_test,DT_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,DT_pred)
print("Decision Tree Model Evaluation:")
print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE:- {rmse}\n R2 Score :- {r2}")
```

```
# Extra Trees Regression

ET=ExtraTreesRegressor()
ET.fit(x_train,y_train)
ET_pred=ET.predict(x_test)

mae = mean_absolute_error(y_test,ET_pred)
mse = mean_squared_error(y_test,ET_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,ET_pred)
print("Extra Trees Model Evaluation:")
print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE:- {rmse}\n R2 Score  :- {r2}")
```

```
#Random Forest Regression

Rf= RandomForestRegressor()
Rf.fit(x_train,y_train)
Rf_pred=Rf.predict(x_test)

mae = mean_absolute_error(y_test,Rf_pred)
mse = mean_squared_error(y_test,Rf_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,Rf_pred)
print("Random Forest Model Evaluation:")
print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE:- {rmse}\n R2 Score  :- {r2}")
```

```
# Define function to fit Prophet model
def fit_prophet(train_data):
    model = Prophet()
    train_df = train_data.rename(columns={'Date': 'ds', 'Close': 'y'})
    model.fit(train_df)
    return model

# Fit Prophet model
prophet_model = fit_prophet(train_data)

# Make predictions with Prophet
future = prophet_model.make_future_dataframe(periods=len(x_test))
# future = pd.DataFrame({'ds': pd.date_range(start=train_data.index[-1], periods=len(x_test)+1, freq='D')[1:]})
prophet_pred = prophet_model.predict(future)['yhat'].tail(len(x_test))

# Calculate evaluation metrics
mae_prophet = mean_absolute_error(y_test, prophet_pred)
mse_prophet = mean_squared_error(y_test, prophet_pred)
rmse_prophet = np.sqrt(mse_prophet)
r2_prophet = r2_score(y_test, prophet_pred)

print("Prophet Model Evaluation:")
print(f"Mean Absolute Error (MAE): {mae_prophet}")
print(f"Mean Squared Error (MSE): {mse_prophet}")
print(f"Root Mean Squared Error (RMSE): {rmse_prophet}")
print(f"R-squared (R2) Score: {r2_prophet}")
```

```
# Arima Model

def fit_arima(train_data, order):
    model = ARIMA(train_data, order=order)
    model_fit = model.fit()
    return model_fit

arima_model = fit_arima(train_data['Close'], order=(5,1,0))
arima_pred = arima_model.predict(start=len(train_data), end=len(train_data) + len(test_data) - 1, typ='levels')

mae_arima = mean_absolute_error(test_data['Close'], arima_pred)
mse_arima = mean_squared_error(test_data['Close'], arima_pred)
rmse_arima = np.sqrt(mse_arima)
r2_arima = r2_score(test_data['Close'], arima_pred)

print("ARIMA Model Evaluation:")
print(f"Mean Absolute Error (MAE): {mae_arima}")
print(f"Mean Squared Error (MSE): {mse_arima}")
print(f"Root Mean Squared Error (RMSE): {rmse_arima}")
print(f"R-squared (R2) Score: {r2_arima}")
```

```python
#Sarimax

def fit_sarimax(train_data, order, seasonal_order):
    model = SARIMAX(train_data, order=order, seasonal_order=seasonal_order)
    model_fit = model.fit()
    return model_fit

sarimax_model = fit_sarimax(train_data['Close'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
sarimax_pred = sarimax_model.predict(start=len(train_data), end=len(train_data) + len(test_data) - 1, typ='levels')

mae_sarimax = mean_absolute_error(test_data['Close'], sarimax_pred)
mse_sarimax = mean_squared_error(test_data['Close'], sarimax_pred)
rmse_sarimax = np.sqrt(mse_sarimax)
r2_sarimax = r2_score(test_data['Close'], sarimax_pred)

print("SARIMAX Model Evaluation:")
print(f"Mean Absolute Error (MAE): {mae_sarimax}")
print(f"Mean Squared Error (MSE): {mse_sarimax}")
print(f"Root Mean Squared Error (RMSE): {rmse_sarimax}")
print(f"R-squared (R2) Score: {r2_sarimax}")
```

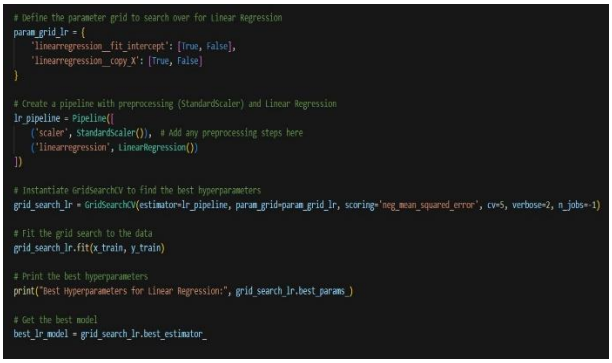
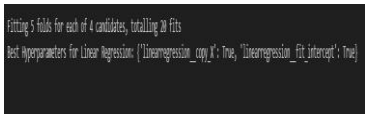
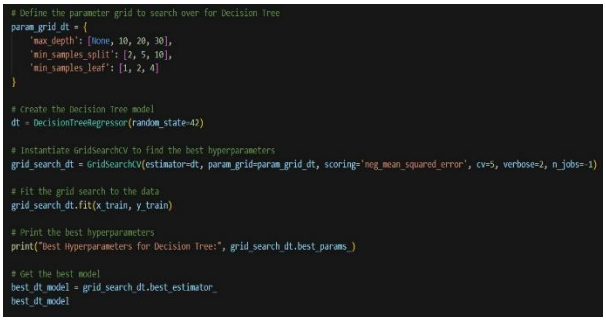
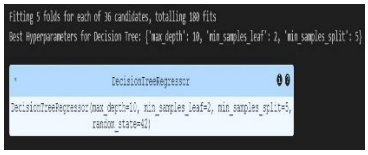
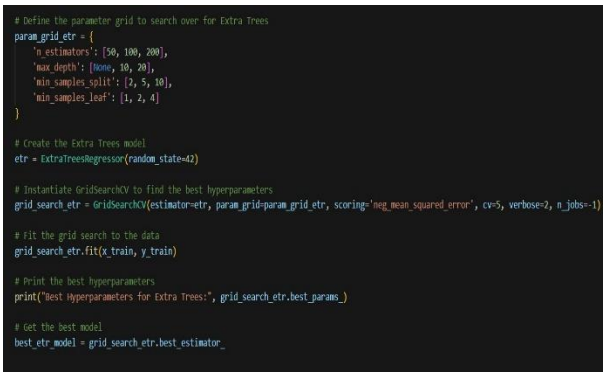
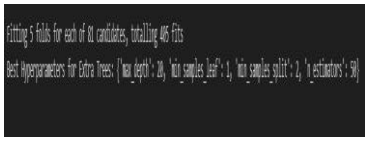
Model Validation and Evaluation Report:

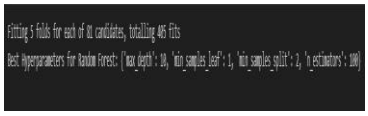
Model	Accuracy/Model Evaluation
Linear Regression	Linear Regression Model Evaluation: MAE :- 0.9166458539179153 MSE :- 2.9316225864388077 RMSE:- 1.7121981738218295 R2 Score :- 0.9998367000901055
Decision Tree	Decision Tree Model Evaluation: MAE :- 4.499876777100751 MSE :- 516.4783238252011 RMSE:- 22.726159460524805 R2 Score :- 0.9712306542686407
Extra Tree	Extra Trees Model Evaluation: MAE :- 3.1554407866203813 MSE :- 541.776521284075 RMSE:- 23.276093342399086 R2 Score :- 0.9698214710454531

Random Forest	Random Forest Model Evaluation: MAE :- 4.1107066195493 85 MSE :- 532.7484588530283 RMSE:- 23.08134439007027 R2 Score :- 0.97032436039701 28
Prophet Model	Prophet Model Evaluation: Mean Absolute Error (MAE): 195.73297116012097 Mean Squared Error (MSE): 48206.04630780599 Root Mean Squared Error (RMSE): 219.55875365788992 R-squared (R2) Score: -1.031440366400611
Arima Model	ARIMA Model Evaluation: Mean Absolute Error (MAE): 111.46898245666749 Mean Squared Error (MSE): 33882.43178759622 Root Mean Squared Error (RMSE): 184.0718114964815 R-squared (R2) Score: -0.4278320857438278
Sarimax Model	SARIMAX Model Evaluation: Mean Absolute Error (MAE): 109.07010937714938 Mean Squared Error (MSE): 32757.38967373523 Root Mean Squared Error (RMSE): 180.9900264482417 R-squared (R2) Score: -0.38042193413328573

5. Model Optimization and Tuning Phase

5.1. Hyperparameter Tuning Documentation

Model	Tuned Hyperparameters	Optimal Values
Linear Regression		
Decision Tree		
Extra Tree		

Random Forest		

5.2. Performance Metrics Comparison Report

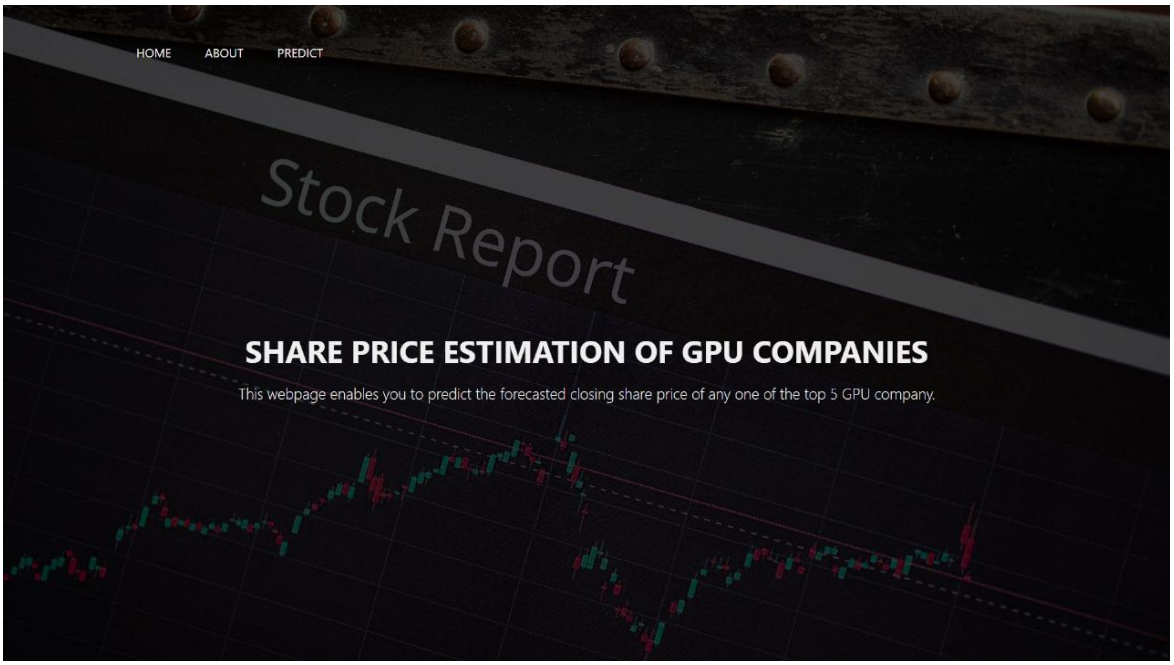
Model	Optimized Metric
Linear Regression	Hyper-tuned Linear Regression Model: R-squared (R2) Score: 0.9998367000912862 Mean Absolute Error (MAE): 0.9166458477303411 Mean Squared Error (MSE): 2.9316225652417143 Root Mean Squared Error (RMSE): 1.712198167631806
Decision Tree	Decision Tree Model Evaluation: Mean Squared Error (MSE): 510.72920925764066 Root Mean Squared Error (RMSE): 22.599318778618983 Mean Absolute Error (MAE): 4.297187979147489 R2 Score after hyperparmeter tunning 0.971550896681562
Extra Tree	Extra Trees Model Evaluation: Mean Squared Error (MSE): 488.7417236390516 Root Mean Squared Error (RMSE): 22.10750378579752 Mean Absolute Error (MAE): 3.006387086974478 R-squared (R2) Score: 0.9727756636201618
Random Forest	Random Forest Model Evaluation: Mean Squared Error (MSE): 527.4793941810102 Root Mean Squared Error (RMSE): 22.966919562296773 Mean Absolute Error (MAE): 4.128952048218431 R-squared (R2) Score: 0.9706178626336749

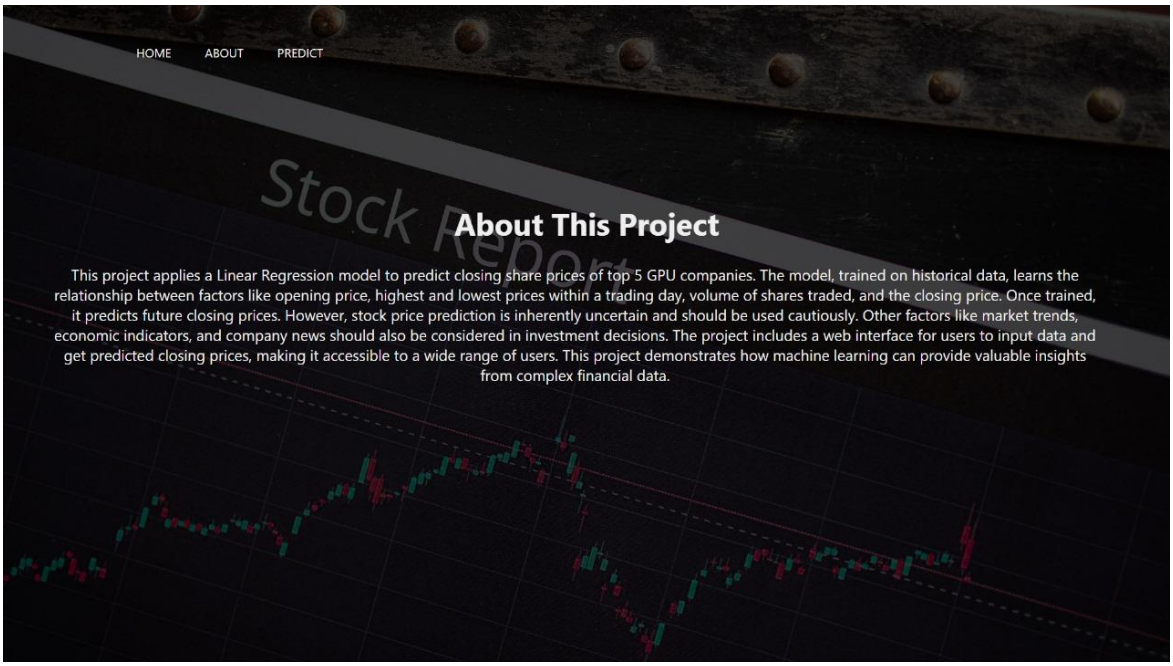
5.3. Final Model Selection Justification

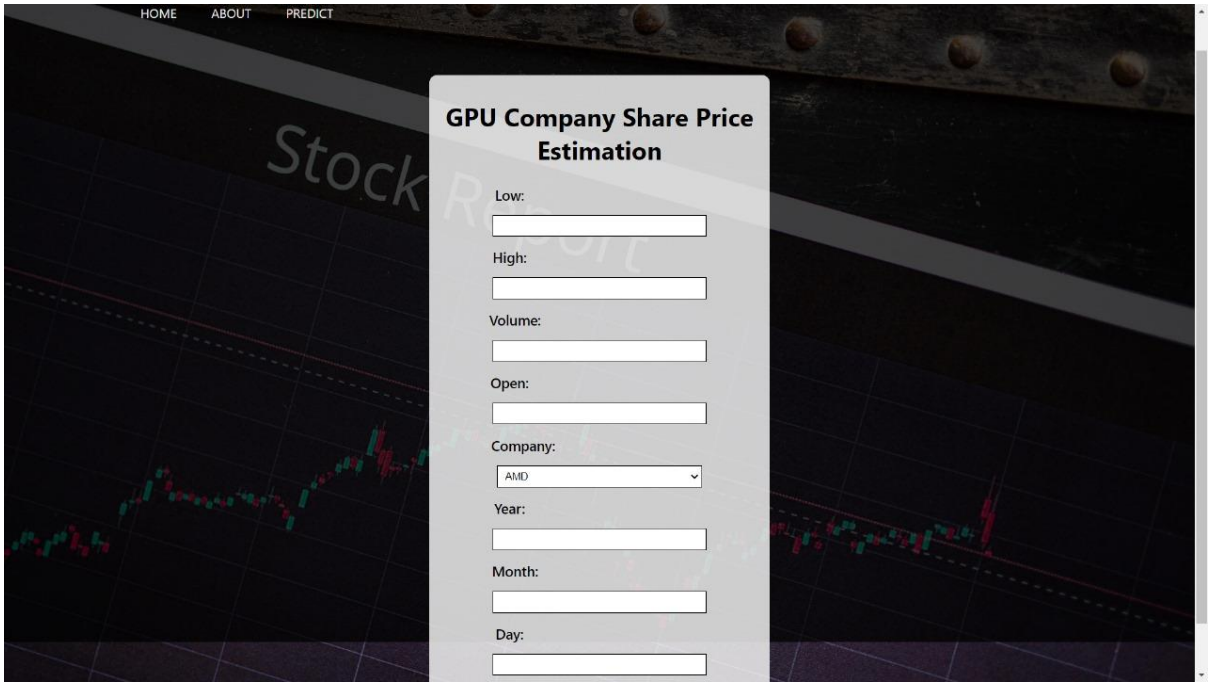
Final Model	Reasoning
Linear Regression	The Linear Regression model was chosen for its remarkable performance, showcasing exceptional accuracy during the model selection process. Its simplicity coupled with its ability to capture linear relationships between variables efficiently makes it an ideal choice for the project's objectives. By mitigating overfitting through regularization techniques and fine-tuning model parameters, solidifies its position as the optimal choice for meeting the project's requirements.

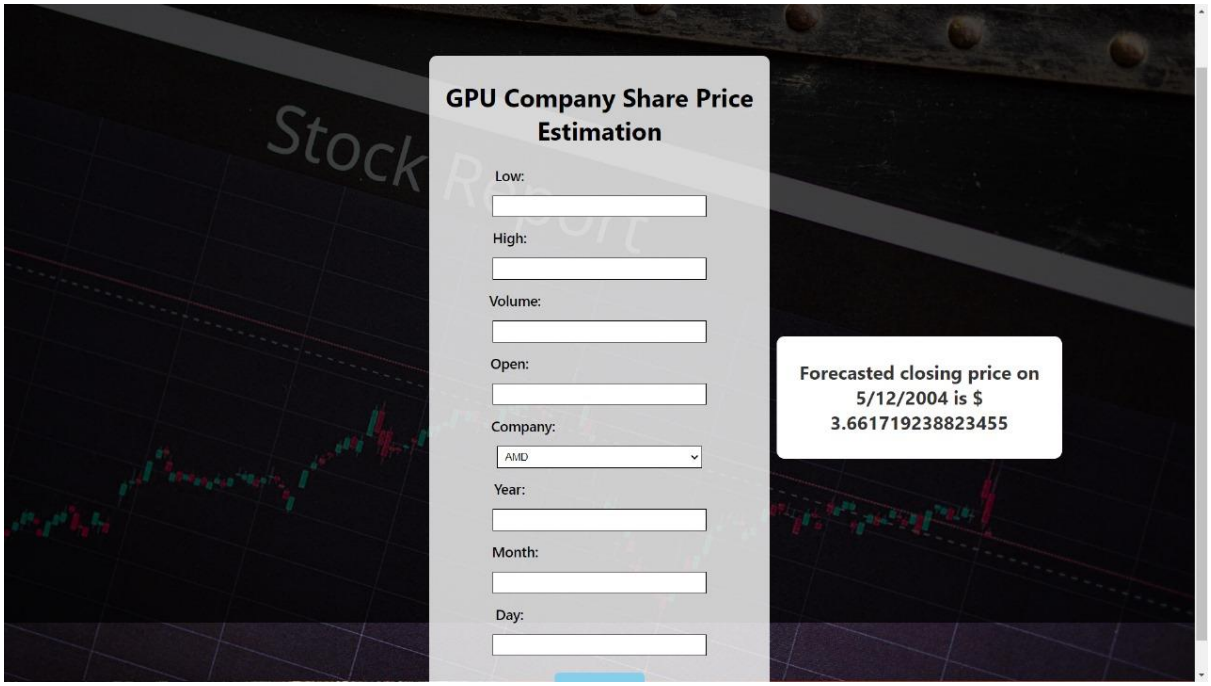
6. Results

6.1. Output Screenshots






HOME
ABOUT
PREDICT
GPU Company Share Price Estimation
Low:
High:
Volume:
Open:
Company:
AMD
Year:
Month:
Day:


GPU Company Share Price Estimation
Low:
High:
Volume:
Open:
Company:
AMD
Year:
Month:
Day:
Forecasted closing price on 5/12/2004 is $ 3.661719238823455

7. Advantages & Disadvantages

Advantages

Informed Decision Making: The project enables investors and stakeholders to make informed decisions by providing predictions of future stock prices based on historical data and relevant factors.

Risk Mitigation: By identifying key factors influencing stock prices, the project helps mitigate investment risks and allows for better risk management strategies.

Automation: Once developed, the predictive models can automate the process of forecasting stock prices, saving time and resources for investors and financial analysts.

Insight Generation: Through exploratory data analysis and model interpretation, the project generates valuable insights into market trends, industry dynamics, and company performance.

Adaptability: The project's methodologies can be adapted and applied to different industries and financial markets, providing versatility and scalability.

Disadvantages

Model Uncertainty: Predictive models inherently involve uncertainty, and the accuracy of predictions may be affected by unforeseen events or changes in market conditions.

Data Limitations: The quality and availability of historical data, economic indicators, and other relevant factors may vary, potentially impacting the accuracy and reliability of the predictive models.

Overfitting: There is a risk of overfitting the models to the training data, resulting in overly complex models that perform well on historical data but fail to generalize to unseen data.

Interpretability: Complex machine learning or deep learning models may lack interpretability, making it challenging to understand the underlying factors driving the predictions and limiting their usability for decision-making.

Ethical Considerations: The use of predictive models in financial markets raises ethical considerations, such as potential biases in data or algorithms, which may lead to unintended consequences or unfair outcomes.

8. Conclusion

This project uses historical data, economic factors, industry trends, and company financials to build models that predict future stock prices for the top 5 GPU companies. These models help investors make informed decisions by understanding what influences stock prices. While the models are accurate, there's uncertainty in financial markets due to data quality, model limitations, and ever-changing market conditions. To improve, the models need to be constantly checked and refined, along with monitoring market trends. Additionally, making the models easier to understand, addressing ethical concerns, and using them for other sectors could increase the project's impact. Overall, this project is a big step towards using data to navigate financial markets and empower investors with knowledge to make better decisions. By embracing innovation and collaboration, we can further develop these models to create more reliable and transparent investment strategies.

9. Future Scope

The future scope of this project extends beyond the initial predictive modelling phase, offering opportunities for further refinement, expansion, and application in various domains. Firstly, enhancements to the predictive models can be explored, including the incorporation of additional data sources, refining feature engineering techniques, and experimenting with advanced machine learning or deep learning algorithms. Incorporating sentiment analysis from social media, news articles, and expert opinions could provide valuable insights into market sentiment and improve prediction accuracy. Moreover, integrating real-time data streams for continuous model updating and deployment in a production environment could enhance the model's responsiveness to dynamic market conditions.

Additionally, the project's scope can be broadened to include other sectors beyond the GPU industry. Applying similar methodologies to forecast stock prices in different industries or to predict other financial metrics such as volatility or trading volume could yield valuable insights for investors and financial institutions. Furthermore, exploring the use of alternative data sources such as satellite imagery, web scraping, or IoT sensor data could provide unique insights into market trends and company performance.

Another avenue for future exploration involves incorporating explainability and interpretability techniques into the predictive models. By understanding the underlying factors driving the model's predictions, stakeholders can gain deeper insights into market dynamics and make more informed decisions. Moreover, exploring the application of reinforcement learning techniques for portfolio optimization or algorithmic trading strategies could further enhance the project's utility in financial decision-making.

Finally, considering the rapid advancements in artificial intelligence and data analytics, ongoing research and development efforts can continue to refine and innovate predictive modeling techniques for financial markets. Collaborations with industry experts, academia, and financial institutions can foster interdisciplinary approaches and facilitate the integration of cutting-edge technologies into the project. Overall, the future scope of this project encompasses a wide range of possibilities for innovation, collaboration, and application in the dynamic landscape of financial markets.

10. Appendix

10.1. Source Code

////

```
import pandas as pd
import warnings
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
import seaborn as sns
warnings.filterwarnings('ignore')

# Read data for each company and time period
amd_1980_2023 = pd.read_csv('AMD (1980 -11.07.2023).csv')
amd_2023_2024 = pd.read_csv('AMD (2023 - 08.04.2024).csv')
amd = pd.concat([amd_1980_2023, amd_2023_2024], axis=0)
amd
# Read data for each company and time period
ASUS_2000_2023 = pd.read_csv('ASUS (2000 - 11.07.2023).csv')
ASUS_2023_2024 = pd.read_csv('ASUS (2023 - 08.04.2024).csv')

# Concatenate the dataframes
asus = pd.concat([ASUS_2000_2023, ASUS_2023_2024], axis=0)
asus

INTEL1 = pd.read_csv('INTEL (1980 - 11.07.2023).csv')
Intel2= pd.read_csv('Intel (2023 - 08.04.2024).csv')

# Concatenate the dataframes
Intel = pd.concat([INTEL1,Intel2], axis=0)
Intel
```

```
msi=pd.read_csv('MSI (2023 - 08.04.2024).csv')
msi
nvidia1 =pd.read_csv('NVIDIA (1999 -11.07.2023).csv')
nvidia2 =pd.read_csv('Nvidia (2023 - 08.04.2024).csv')
# Concatenate the dataframes
nvidia = pd.concat([nvidia1,nvidia2], axis=0)
nvidia

# Changing dates to timestamps
datal=[amd,asus,Intel,msi,nvidia]
for data in datal:
   data['Date']=pd.to_datetime(data['Date'])

#Resampling the Datsets
import numpy as np
datal=[amd,asus,Intel,msi,nvidia]
names=[0,1,2,3,4]
index=0
for data in datal:
   dates= data['Date']
   data['Company'] = np.repeat (names[index], len(data))
   data['Year'] =dates.dt.year
   data[ 'Month'] = dates.dt.month
   data['Day']= dates.dt.day
   index+=1

# Merging and splitting data into test and train variables
data_list=[amd, asus, Intel,msi, nvidia]
test_data = []
```

```
train_data = []

# Split each DataFrame and populate train_data and test_data lists
for data in data_list:
    train = data.iloc[:int(0.8 * len(data))]  # First 80% for training
    test = data.iloc[int(0.8 * len(data)):]  # Last 20% for testing
    train_data.append(train)
    test_data.append(test)
    print(f"Train shape: {train.shape}, Test shape: {test.shape}")

# Save each training dataset to CSV
for i, train_df in enumerate(train_data):
    train_df.to_csv(f'train_data_{i}.csv', index=False)

# Save each testing dataset to CSV
for i, test_df in enumerate(test_data):
    test_df.to_csv(f'test_data_{i}.csv', index=False)

print("Training and testing datasets saved to CSV successfully!")

train_data=pd.concat(train_data)
test_data =pd.concat(test_data)
print(train_data.shape)
print(test_data.shape)

x_train= train_data[['Open', 'High', 'Low', 'Volume', 'Year', 'Month', 'Day', 'Company']]
x_test =test_data[['Open', 'High', 'Low', 'Volume', 'Year', 'Month', 'Day', 'Company']]
y_train =train_data[ 'Close']
y_test= test_data['Close']
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

#Model Building
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import ExtraTreesRegressor, RandomForestRegressor
from sklearn.linear_model import Ridge
from pmdarima import auto_arima
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error ,r2_score
from prophet import Prophet
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# linear regression

lr=LinearRegression()
lr.fit(x_train,y_train)
lr_pred=lr.predict(x_test)

mae = mean_absolute_error(y_test,lr_pred)
mse = mean_squared_error(y_test,lr_pred)
rmse = np.sqrt(mse)
```

```
r2 = r2_score(y_test,lr_pred)
print("Linear Regression Model Evaluation:")
print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE:- {rmse}\n R2 Score :- {r2}")

# Define the parameter grid to search over for Linear Regression
param_grid_lr = {
   'linearregression__fit_intercept': [True, False],
   'linearregression__copy_X': [True, False]
}

# Create a pipeline with preprocessing (StandardScaler) and Linear Regression
lr_pipeline = Pipeline([
   ('scaler', StandardScaler()),  # Add any preprocessing steps here
   ('linearregression', LinearRegression())
])

# Instantiate GridSearchCV to find the best hyperparameters
grid_search_lr = GridSearchCV(estimator=lr_pipeline, param_grid=param_grid_lr,
scoring='neg_mean_squared_error', cv=5, verbose=2, n_jobs=-1)

# Fit the grid search to the data
grid_search_lr.fit(x_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters for Linear Regression:", grid_search_lr.best_params_)

# Get the best model
best_lr_model = grid_search_lr.best_estimator_

# Use the best model to make predictions on test data
```

```
y_pred = best_lr_model.predict(x_test)

# Compute evaluation metrics
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE

# Print evaluation metrics
print("Hyper-tuned Linear Regression Model:")
print("R-squared (R2) Score:", r2)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)

# decision tree

DT=DecisionTreeRegressor()
DT.fit(x_train,y_train)
DT_pred=DT.predict(x_test)

mae = mean_absolute_error(y_test,DT_pred)
mse = mean_squared_error(y_test,DT_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,DT_pred)
print("Decision Tree Model Evaluation:")
print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE:- {rmse}\n R2 Score :- {r2}")

# Use the best model to make predictions on test data
```

```
y_pred = best_lr_model.predict(x_test)

# Compute evaluation metrics
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE

# Print evaluation metrics
print("Evaluation Metrics for Linear Regression Model:")
print("R-squared (R2) Score:", r2)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)

# Extract the best hyperparameters from the grid search results
best_params_dt = {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}

# Create a new instance of the Decision Tree model with the best hyperparameters
best_dt_model = DecisionTreeRegressor(**best_params_dt)

# Fit the model to the training data
best_dt_model.fit(x_train, y_train)

# Use the trained model to make predictions on the test data
y_pred_dt = best_dt_model.predict(x_test)

# Evaluate the model using appropriate metrics
mse_dt = mean_squared_error(y_test, y_pred_dt)
```

```
rmse_dt = mean_squared_error(y_test, y_pred_dt, squared=False)  # RMSE
mae_dt = mean_absolute_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test,y_pred_dt)
# Print the evaluation metrics
print("Decision Tree Model Evaluation:")
print("Mean Squared Error (MSE):", mse_dt)
print("Root Mean Squared Error (RMSE):", rmse_dt)
print("Mean Absolute Error (MAE):", mae_dt)
print("R2 Score  after hyperparmeter tunning",r2_dt)

# Extra Trees Regression

ET=ExtraTreesRegressor()
ET.fit(x_train,y_train)
ET_pred=ET.predict(x_test)

mae = mean_absolute_error(y_test,ET_pred)
mse = mean_squared_error(y_test,ET_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,ET_pred)
print("Extra Trees Model Evaluation:")
print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE:- {rmse}\n R2 Score  :- {r2}")

# Define the parameter grid to search over for Extra Trees
param_grid_etr = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
```

```
}

# Create the Extra Trees model
etr = ExtraTreesRegressor(random_state=42)

# Instantiate GridSearchCV to find the best hyperparameters
grid_search_etr = GridSearchCV(estimator=etr, param_grid=param_grid_etr,
scoring='neg_mean_squared_error', cv=5, verbose=2, n_jobs=-1)

# Fit the grid search to the data
grid_search_etr.fit(x_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters for Extra Trees:", grid_search_etr.best_params_)

# Get the best model
best_etr_model = grid_search_etr.best_estimator_

# Extract the best hyperparameters from the grid search results
best_params_et = {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 50}

# Create a new instance of the Extra Trees model with the best hyperparameters
best_et_model = ExtraTreesRegressor(**best_params_et)

# Fit the model to the training data
best_et_model.fit(x_train, y_train)

# Use the trained model to make predictions on the test data
y_pred_et = best_et_model.predict(x_test)
```

```
# Evaluate the model using appropriate metrics
mse_et = mean_squared_error(y_test, y_pred_et)
rmse_et = mean_squared_error(y_test, y_pred_et, squared=False)  # RMSE
mae_et = mean_absolute_error(y_test, y_pred_et)
r2_et = r2_score(y_test, y_pred_et)  # Calculate R-squared score

# Print the evaluation metrics
print("Extra Trees Model Evaluation:")
print("Mean Squared Error (MSE):", mse_et)
print("Root Mean Squared Error (RMSE):", rmse_et)
print("Mean Absolute Error (MAE):", mae_et)
print("R-squared (R2) Score:", r2_et)

#Random Forest Regression

Rf= RandomForestRegressor()
Rf.fit(x_train,y_train)
Rf_pred=Rf.predict(x_test)

mae = mean_absolute_error(y_test,Rf_pred)
mse = mean_squared_error(y_test,Rf_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,Rf_pred)
print("Random Forest Model Evaluation:")
print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE:- {rmse}\n R2 Score  :- {r2}")

# Define the parameter grid to search over for Random Forest
param_grid_rf = {
```

```
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the Random Forest model
rf = RandomForestRegressor(random_state=42)

# Instantiate GridSearchCV to find the best hyperparameters
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf,
scoring='neg_mean_squared_error', cv=5, verbose=2, n_jobs=-1)

# Fit the grid search to the data
grid_search_rf.fit(x_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters for Random Forest:", grid_search_rf.best_params_)

# Get the best model
best_rf_model = grid_search_rf.best_estimator_

# Extract the best hyperparameters from the grid search results
best_params_rf = {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 100}

# Create a new instance of the Random Forest model with the best hyperparameters
best_rf_model = RandomForestRegressor(**best_params_rf)

# Fit the model to the training data
```

```
best_rf_model.fit(x_train, y_train)

# Use the trained model to make predictions on the test data
y_pred_rf = best_rf_model.predict(x_test)

# Evaluate the model using appropriate metrics
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = mean_squared_error(y_test, y_pred_rf, squared=False)  # RMSE
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)  # Calculate R-squared score

# Print the evaluation metrics
print("Random Forest Model Evaluation:")
print("Mean Squared Error (MSE):", mse_rf)
print("Root Mean Squared Error (RMSE):", rmse_rf)
print("Mean Absolute Error (MAE):", mae_rf)
print("R2 Score:", r2_rf)

# Prophet

# Define function to fit Prophet model
def fit_prophet(train_data):
    model = Prophet()
    train_df = train_data.rename(columns={'Date': 'ds', 'Close': 'y'})
    model.fit(train_df)
    return model

# Fit Prophet model
prophet_model = fit_prophet(train_data)
```

```
# Make predictions with Prophet

future = prophet_model.make_future_dataframe(periods=len(x_test))

# future = pd.DataFrame({'ds': pd.date_range(start=train_data.index[-1],
periods=len(x_test)+1, freq='D')[1:]})

prophet_pred = prophet_model.predict(future)['yhat'].tail(len(x_test))


# Calculate evaluation metrics

mae_prophet = mean_absolute_error(y_test, prophet_pred)

mse_prophet = mean_squared_error(y_test, prophet_pred)

rmse_prophet = np.sqrt(mse_prophet)

r2_prophet = r2_score(y_test, prophet_pred)


print("Prophet Model Evaluation:")

print(f"Mean Absolute Error (MAE): {mae_prophet}")

print(f"Mean Squared Error (MSE): {mse_prophet}")

print(f"Root Mean Squared Error (RMSE): {rmse_prophet}")

print(f"R2 Score: {r2_prophet}")


# Arima Model


def fit_arima(train_data, order):

    model = ARIMA(train_data, order=order)

    model_fit = model.fit()

    return model_fit


arima_model = fit_arima(train_data['Close'], order=(5,1,0))

arima_pred = arima_model.predict(start=len(train_data), end=len(train_data) +
len(test_data) - 1, typ='levels')
```

```
mae_arima = mean_absolute_error(test_data['Close'], arima_pred)
mse_arima = mean_squared_error(test_data['Close'], arima_pred)
rmse_arima = np.sqrt(mse_arima)
r2_arima = r2_score(test_data['Close'], arima_pred)

print("ARIMA Model Evaluation:")
print(f"Mean Absolute Error (MAE): {mae_arima}")
print(f"Mean Squared Error (MSE): {mse_arima}")
print(f"Root Mean Squared Error (RMSE): {rmse_arima}")
print(f"R2 Score: {r2_arima}")

#Sarimax

def fit_sarimax(train_data, order, seasonal_order):
    model = SARIMAX(train_data, order=order, seasonal_order=seasonal_order)
    model_fit = model.fit()
    return model_fit

sarimax_model = fit_sarimax(train_data['Close'], order=(1, 1, 1), seasonal_order=(1, 1, 1,
12))
sarimax_pred = sarimax_model.predict(start=len(train_data), end=len(train_data) +
len(test_data) - 1, typ='levels')

mae_sarimax = mean_absolute_error(test_data['Close'], sarimax_pred)
mse_sarimax = mean_squared_error(test_data['Close'], sarimax_pred)
rmse_sarimax = np.sqrt(mse_sarimax)
r2_sarimax = r2_score(test_data['Close'], sarimax_pred)

print("SARIMAX Model Evaluation:")
print(f"Mean Absolute Error (MAE): {mae_sarimax}")
```

```
print(f"Mean Squared Error (MSE): {mse_sarimax}")
print(f"Root Mean Squared Error (RMSE): {rmse_sarimax}")
print(f"R2 Score: {r2_sarimax}")

from sklearn.model_selection import cross_val_score

# Define and fit Ridge Regression model
ridge = Ridge(alpha=1.0)  # You can adjust alpha for regularization strength
ridge.fit(x_train, y_train)

# Make predictions
y_pred = ridge.predict(x_test)

# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Cross-validation
cv_scores = cross_val_score(ridge, x_train, y_train, cv=5)  # 5-fold cross-validation
mean_cv_score = np.mean(cv_scores)

# Print evaluation metrics
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("Cross-Validation Mean Score:", mean_cv_score)

#Plot predictions and actual values
```

```
plt.figure(figsize=(15,8))

sns.lineplot(x=amd_dates,y=amd_orig)
sns.lineplot(x=amd_dates,y=amd_pred)

sns.lineplot(x=asus_dates,y=asus_orig)
sns.lineplot(x=asus_dates,y=asus_pred)

sns.lineplot(x=intel_dates,y=intel_orig)
sns.lineplot(x=intel_dates,y=intel_pred)

sns.lineplot(x=msi_dates,y=msi_orig)
sns.lineplot(x=msi_dates,y=msi_pred)

sns.lineplot(x=nvidia_dates,y=nvidia_orig)
sns.lineplot(x=nvidia_dates,y=nvidia_pred)

plt.legend (['AMD Original', 'AMD Predicted', 'ASUS Original', 'ASUS Predicted', 'INTEL Original', 'INTEL Predicted',
        'MSI Original', 'MSI Predicted', 'NVIDIA Original', 'NVIDIA Predicted'])
plt.show()
```

////

10.2. GitHub & Project Demo Link

https://github.com/Alex333-code/Share_Price_Estimation_Of_GPU_Companies

Demo