Stock Market

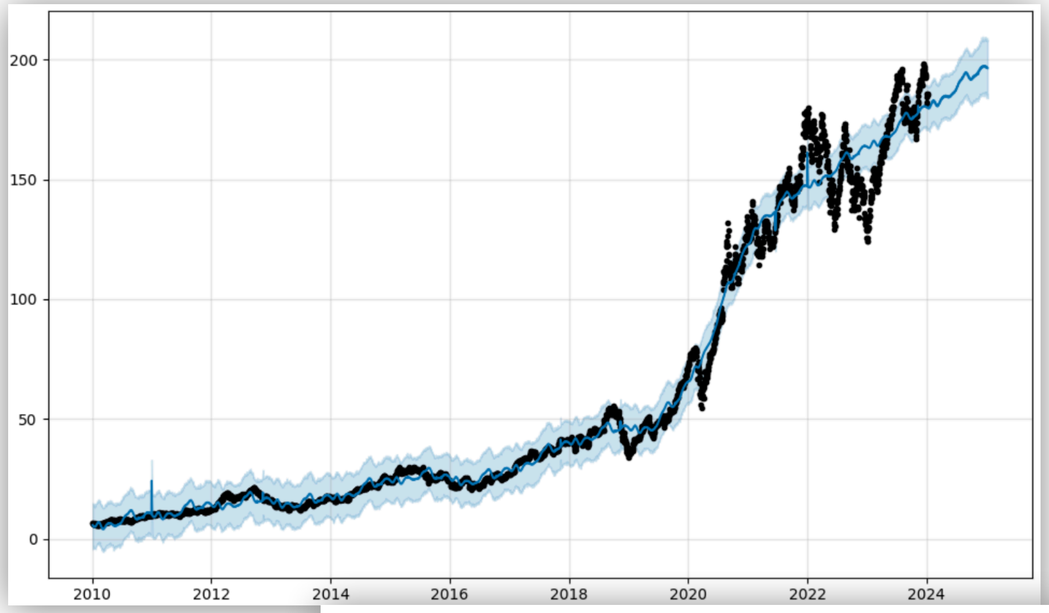
Fear of Loss

Socioeconomic Factors

Psychological Biases

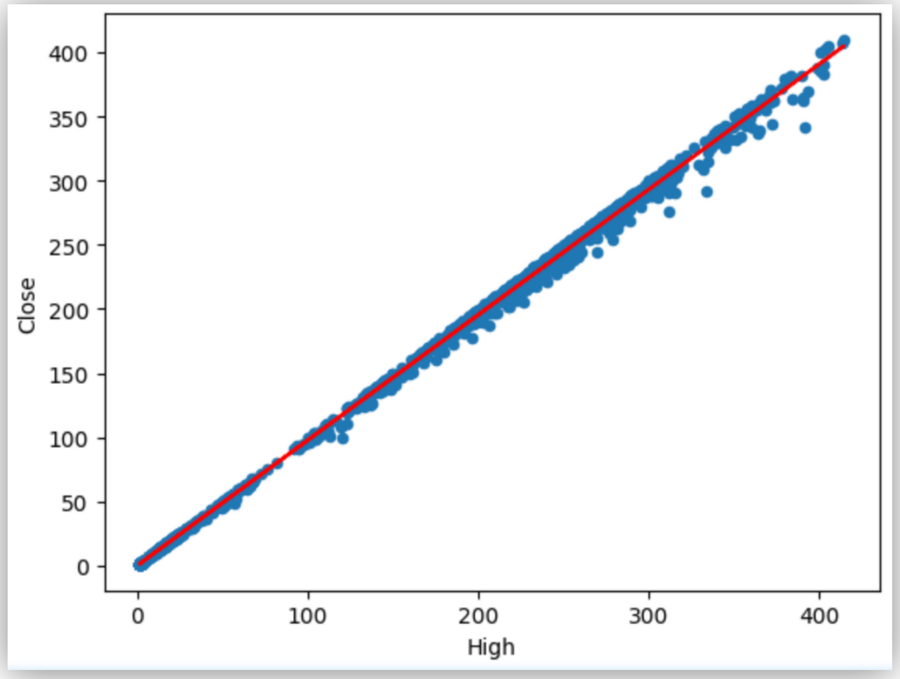
Stress and Anxiety

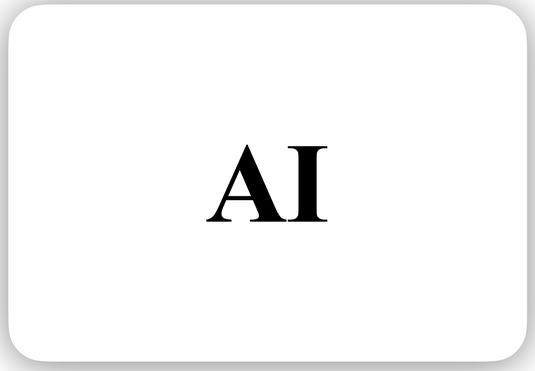
Preferences
- Socioeconomic Factors and Barriers to Stock Market Participation" by Jennifer Baker and Sarah Bloom Raskin (2015)
- The Impact of Media Portrayals on Investor Sentiment and the Stock Market" by David C. Garland and William R. Maurer (2010)



Prophet

Linear regression

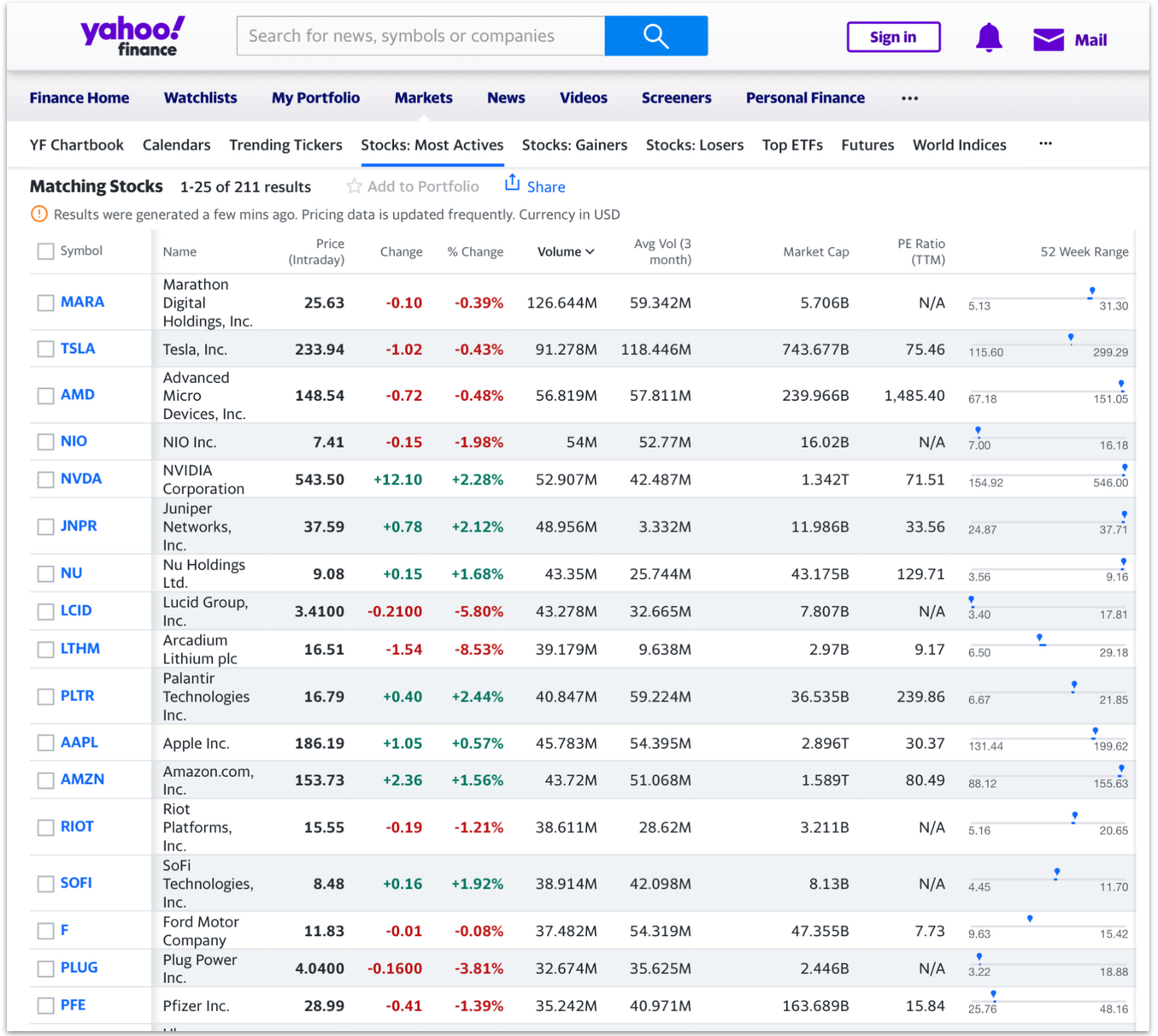



AI

DataSet

DataSet
from
Yahoo Finance

```
!pip install yfinance
import yfinance as yf
data = yf.download('AAPL', start='2010-01-01', end='2025-01-01')
```



Symbol	Name	Price (Intraday)	Change	% Change	Volume	Avg Vol (3 month)	Market Cap	PE Ratio (TTM)	52 Week Range
MARA	Marathon Digital Holdings, Inc.	25.63	-0.10	-0.39%	126.644M	59.342M	5.706B	N/A	5.13 31.30
TSLA	Tesla, Inc.	233.94	-1.02	-0.43%	91.278M	118.446M	743.677B	75.46	115.60 299.29
AMD	Advanced Micro Devices, Inc.	148.54	-0.72	-0.48%	56.819M	57.811M	239.966B	1,485.40	67.18 151.05
NIO	NIO Inc.	7.41	-0.15	-1.98%	54M	52.77M	16.02B	N/A	7.00 16.18
NVDA	NVIDIA Corporation	543.50	+12.10	+2.28%	52.907M	42.487M	1.342T	71.51	154.92 546.00
JNPR	Juniper Networks, Inc.	37.59	+0.78	+2.12%	48.956M	3.332M	11.986B	33.56	24.87 37.71
NU	Nu Holdings Ltd.	9.08	+0.15	+1.68%	43.35M	25.744M	43.175B	129.71	3.56 9.16
LCID	Lucid Group, Inc.	3.4100	-0.2100	-5.80%	43.278M	32.665M	7.807B	N/A	3.40 17.81
LTHM	Arcadium Lithium plc	16.51	-1.54	-8.53%	39.179M	9.638M	2.97B	9.17	6.50 29.18
PLTR	Palantir Technologies Inc.	16.79	+0.40	+2.44%	40.847M	59.224M	36.535B	239.86	6.67 21.85
AAPL	Apple Inc.	186.19	+1.05	+0.57%	45.783M	54.395M	2.896T	30.37	131.44 199.62
AMZN	Amazon.com, Inc.	153.73	+2.36	+1.56%	43.72M	51.068M	1.589T	80.49	88.12 155.63
RIOT	Riot Platforms, Inc.	15.55	-0.19	-1.21%	38.611M	28.62M	3.211B	N/A	5.16 20.65
SOFI	SoFi Technologies, Inc.	8.48	+0.16	+1.92%	38.914M	42.098M	8.13B	N/A	4.45 11.70
F	Ford Motor Company	11.83	-0.01	-0.08%	37.482M	54.319M	47.355B	7.73	9.63 15.42
PLUG	Plug Power Inc.	4.0400	-0.1600	-3.81%	32.674M	35.625M	2.446B	N/A	3.22 18.88
PFE	Pfizer Inc.	28.99	-0.41	-1.39%	35.242M	40.971M	163.689B	15.84	25.76 48.16

Preferences
- https://github.com/ranaroussi/yfinance
- https://finance.yahoo.com/most-active/

DataSet

```
!pip install yfinance
import yfinance as yf
data = yf.download('AAPL', start='2010-01-01', end='2025-01-01')
```

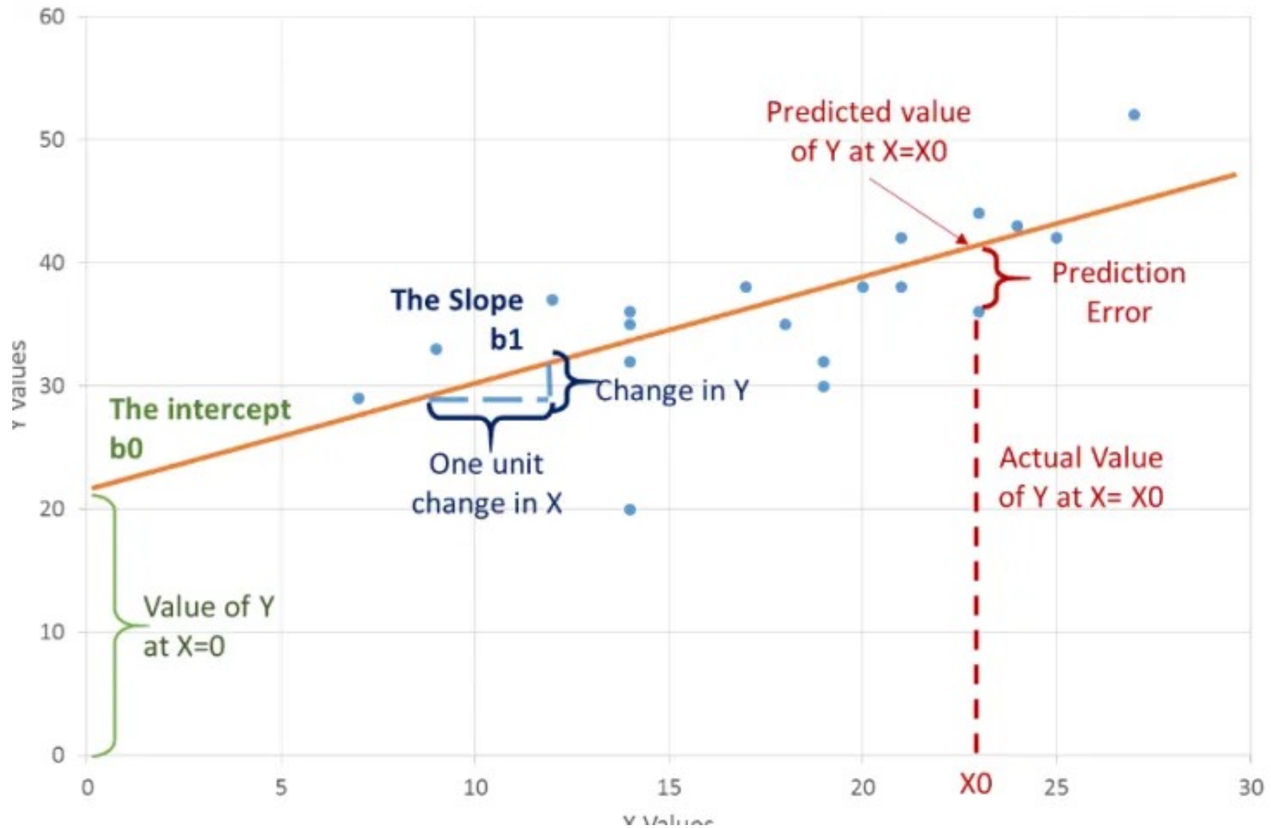
Preferences
- https://github.com/ranaroussi/yfinance
- https://finance.yahoo.com/most-active/

Prophet

Linear Regression

Linear Regression

Regression analysis is a predictive modeling technique that estimates the relationship between two or more variables.



The most common formula for linear regression is:

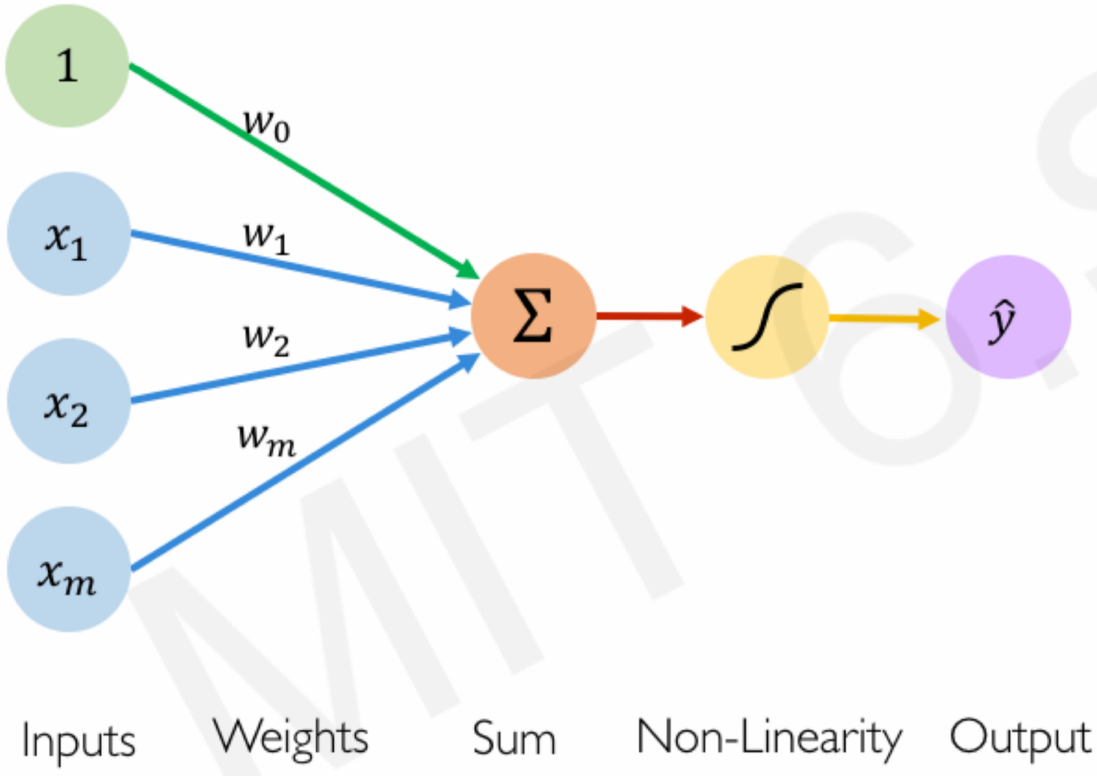
$$y = mx + b$$

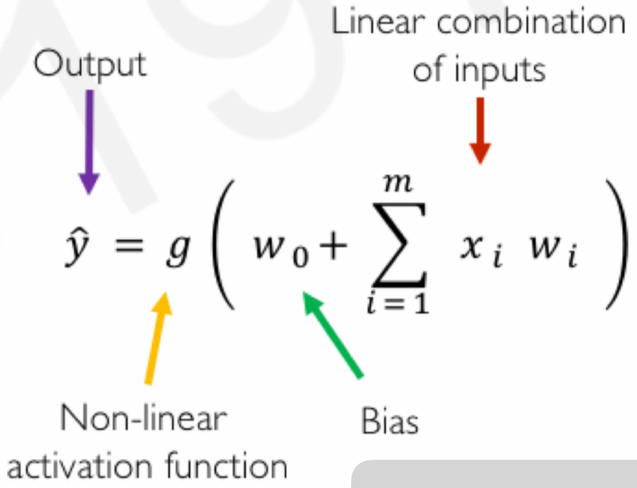
where:

y : the variable you are trying to predict
x : the variable you are using to make the prediction
m : the slope of the regression line
b : the y-intercept of the regression line

https://medium.com/swlh/predictive-modelling-using-linear-regression-e0e399dc4745

Linear Regression





In linear regression analysis, bias refers to the error that is introduced by approximating a real-life problem, which may be complicated, by a much simpler model. Though the linear algorithm can introduce bias, it also makes their output easier to understand

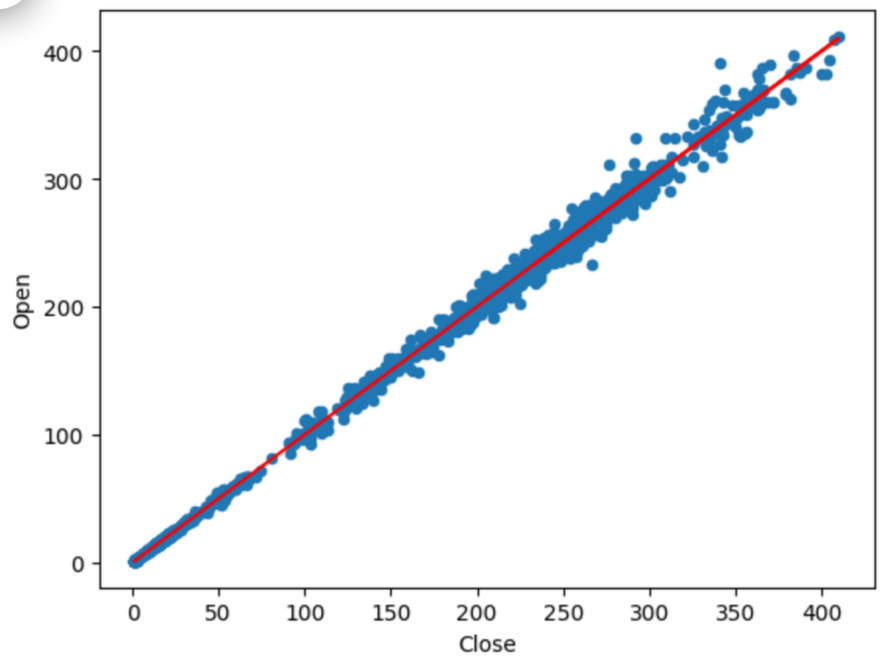
Linear Regression

```
stock = "TSLA"
Begin_data = '2000-1-1'
Today_data = datetime.now().strftime('%Y-%m-%d')
data = yf.download(stock, Begin_data, Today_data)
data.head(5)
```

Output

Date	Open	High	Low	Close	Volume
2010-06-29	1.266667	1.666667	1.169333	1.592667	281494500
2010-06-30	1.719333	2.028000	1.553333	1.588667	257806500
2010-07-01	1.666667	1.728000	1.351333	1.464000	123282000
2010-07-02	1.533333	1.540000	1.247333	1.280000	77097000
2010-07-06	1.333333	1.333333	1.055333	1.074000	103003500

```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
lr = LinearRegression()
lr.fit(data[["Close"]],data["Open"])
data.plot.scatter("Close", "Open")
plt.plot(data["Close"], lr.predict(data[["Close"]]), color="r")
```




```
print(f"Weight: {lr.coef_[0]:.2f}")
print(f"Bias: {lr.intercept_:.2f}")
```

```
Weight: 1.00
Bias: 0.02
```

Prophet

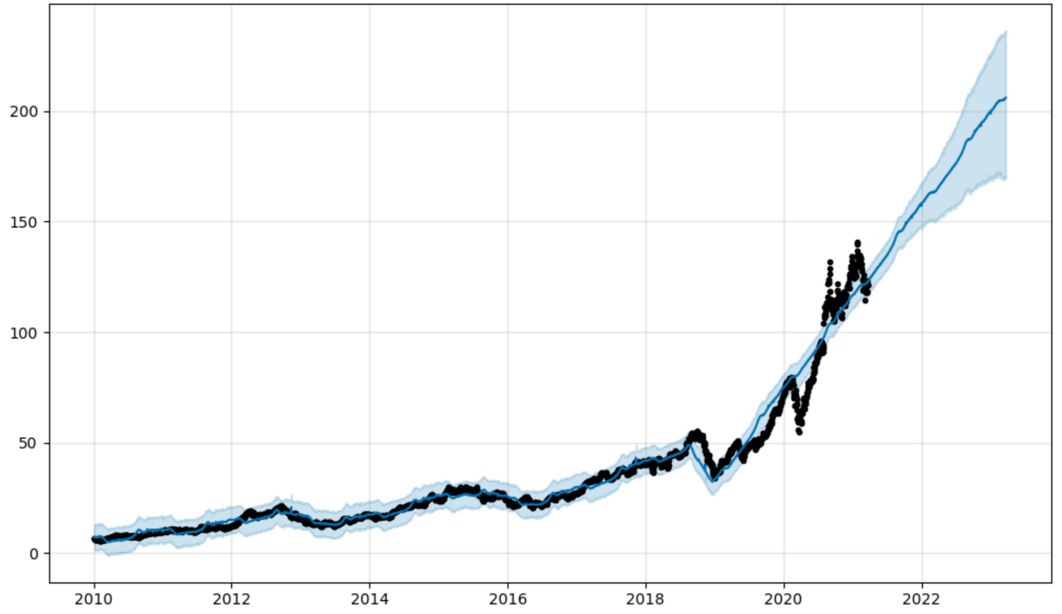
Prophet

Prophet from Meta (Facebook) is a forecasting procedure implemented in Python. We can use it to predict time series data. Since stock prices are time-series data and as we can easily get historic data.

The Prophet model has this form:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon(t)$$

Trend (g(t)): Represents the overall non-periodic changes in the time series. Prophet can model linear or logistic growth trends.
Seasonality (s(t)): Captures periodic patterns (daily, weekly, yearly, etc.).
Holidays (h(t)): Models the effect of holidays or known events.
Error (ε(t)): The unexplained variation in the data.



https://facebook.github.io/prophet/
https://facebook.github.io/prophet/static/prophet_paper_20170113.pdf

Prophet

```
import yfinance as yf
import pandas as pd
from prophet import Prophet
import numpy as np
import matplotlib.pyplot as plt
from prophet.plot import plot_components_plotly
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
```

Prophet

```python
def fetch_data(stock_symbol, start_date, end_date):
    data = yf.download(stock_symbol, start=start_date, end=end_date)
    return data

def preprocess_data(data):
    df = data.reset_index()
    df.rename(columns={'Date': 'ds', 'Adj Close': 'y'}, inplace=True)
    return df[['ds', 'y']]
```

```python
class StockPredictor:
    def __init__(self, stock_symbol, start_date, end_date, periods=365, split_ratio=0.8):
        self.stock_symbol = stock_symbol
        self.start_date = start_date
        self.end_date = end_date
        self.periods = periods
        self.split_ratio = split_ratio
        self.model = None
        self.forecast = None

        self.load_and_preprocess_data()
        self.split_data()
        self.train_prophet_model()
```

```python
import yfinance as yf
import pandas as pd
from prophet import Prophet
import numpy as np
import matplotlib.pyplot as plt
from prophet.plot import plot_components_plotly
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
```

Prophet

```python
import yfinance as yf
import pandas as pd
from prophet import Prophet
import numpy as np
import matplotlib.pyplot as plt
from prophet.plot import plot_components_plotly
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
```

```python
def fetch_data(stock_symbol, start_date, end_date):
    data = yf.download(stock_symbol, start=start_date, end=end_date)
    return data

def preprocess_data(data):
    df = data.reset_index()
    df.rename(columns={'Date': 'ds', 'Adj Close': 'y'}, inplace=True)
    return df[['ds', 'y']]
```

```python
class StockPredictor:
    def __init__(self, stock_symbol, start_date, end_date, periods=365, split_ratio=0.8):
        self.stock_symbol = stock_symbol
        self.start_date = start_date
        self.end_date = end_date
        self.periods = periods
        self.split_ratio = split_ratio
        self.model = None
        self.forecast = None

        self.load_and_preprocess_data()
        self.split_data()
        self.train_prophet_model()
```

```python
    def load_and_preprocess_data(self):
        stock_data = fetch_data(self.stock_symbol, self.start_date, self.end_date)
        self.data = preprocess_data(stock_data)
```

Prophet

```python
import yfinance as yf
import pandas as pd
from prophet import Prophet
import numpy as np
import matplotlib.pyplot as plt
from prophet.plot import plot_components_plotly
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
```

```python
def fetch_data(stock_symbol, start_date, end_date):
    data = yf.download(stock_symbol, start=start_date, end=end_date)
    return data

def preprocess_data(data):
    df = data.reset_index()
    df.rename(columns={'Date': 'ds', 'Adj Close': 'y'}, inplace=True)
    return df[['ds', 'y']]
```

```python
class StockPredictor:
    def __init__(self, stock_symbol, start_date, end_date, periods=365, split_ratio=0.8):
        self.stock_symbol = stock_symbol
        self.start_date = start_date
        self.end_date = end_date
        self.periods = periods
        self.split_ratio = split_ratio
        self.model = None
        self.forecast = None

        self.load_and_preprocess_data()
        self.split_data()
        self.train_prophet_model()
    def load_and_preprocess_data(self):
        stock_data = fetch_data(self.stock_symbol, self.start_date, self.end_date)
        self.data = preprocess_data(stock_data)
    def split_data(self):
        train_size = int(self.split_ratio * len(self.data))
        self.train_data = self.data[:train_size]
        self.test_data = self.data[train_size:]
```

```python
    def train_prophet_model(self):
        self.model = Prophet(
            changepoint_prior_scale=0.05,
            holidays_prior_scale=15,
            seasonality_prior_scale=10,
            weekly_seasonality=True,
            yearly_seasonality=True,
            daily_seasonality=False
        )
        self.model.add_country_holidays(country_name='US')
        self.model.fit(self.train_data)
```

Prophet

```python
def fetch_data(stock_symbol, start_date, end_date):
    data = yf.download(stock_symbol, start=start_date, end=end_date)
    return data

def preprocess_data(data):
    df = data.reset_index()
    df.rename(columns={'Date': 'ds', 'Adj Close': 'y'}, inplace=True)
    return df[['ds', 'y']]
```

```python
class StockPredictor:
    def __init__(self, stock_symbol, start_date, end_date, periods=365, split_ratio=0.8):
        self.stock_symbol = stock_symbol
        self.start_date = start_date
        self.end_date = end_date
        self.periods = periods
        self.split_ratio = split_ratio
        self.model = None
        self.forecast = None

        self.load_and_preprocess_data()
        self.split_data()
        self.train_prophet_model()
    def load_and_preprocess_data(self):
        stock_data = fetch_data(self.stock_symbol, self.start_date, self.end_date)
        self.data = preprocess_data(stock_data)
    def split_data(self):
        train_size = int(self.split_ratio * len(self.data))
        self.train_data = self.data[:train_size]
        self.test_data = self.data[train_size:]
    def train_prophet_model(self):
        self.model = Prophet(
            changepoint_prior_scale=0.05,
            holidays_prior_scale=15,
            seasonality_prior_scale=10,
            weekly_seasonality=True,
            yearly_seasonality=True,
            daily_seasonality=False
        )
        self.model.add_country_holidays(country_name='US')
        self.model.fit(self.train_data)
    def generate_forecast(self, on_test_data=False):
        if on_test_data:
            future = pd.DataFrame(self.test_data['ds'])
        else:
            future = self.model.make_future_dataframe(periods=self.periods)
        self.forecast = self.model.predict(future)
```

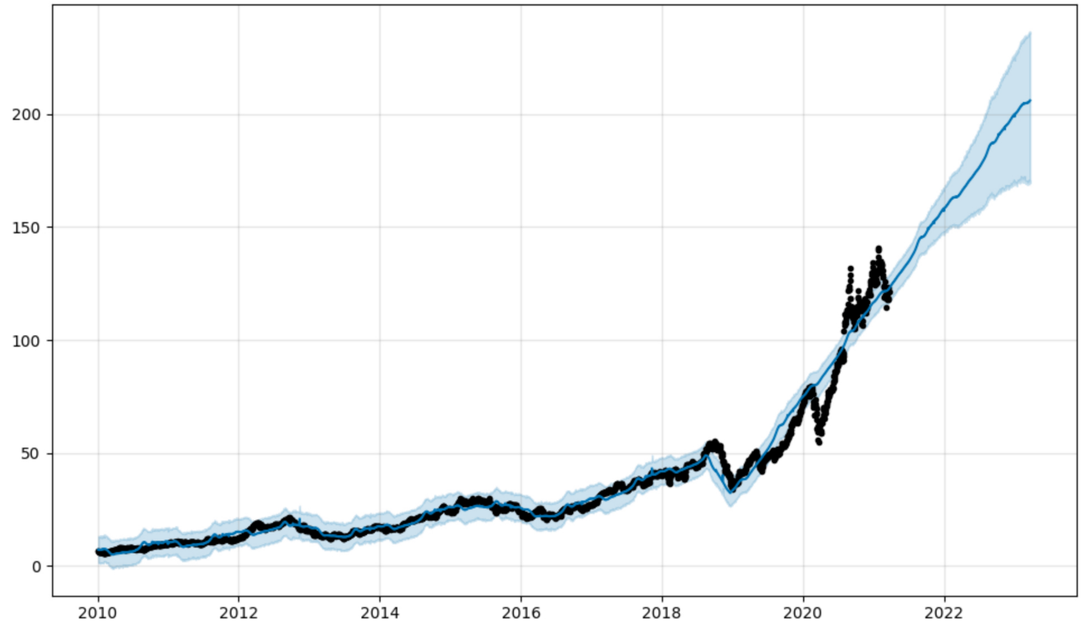
```python
import yfinance as yf
import pandas as pd
from prophet import Prophet
import numpy as np
import matplotlib.pyplot as plt
from prophet.plot import plot_components_plotly
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
```

```python
    def plot_forecast(self):
        fig = self.model.plot(self.forecast)
        plt.show()
    def plot_components(self):
        return plot_components_plotly(self.model, self.forecast)
```

Prophet

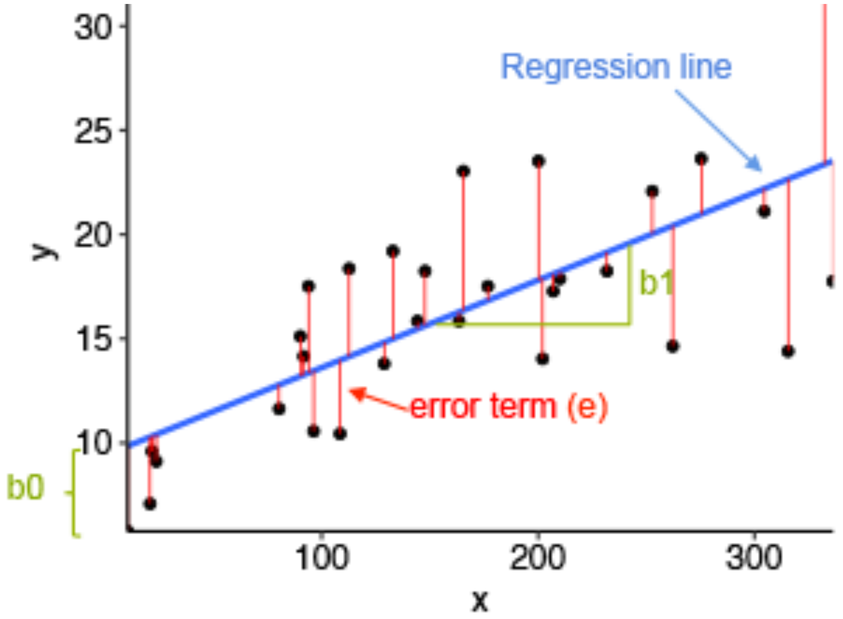
```
stock_symbol = 'AAPL'
start_date = '2010-01-01'
end_date = '2025-01-01'
periods_to_predict = (datetime(2024, 12, 31) - datetime(2021, 4, 9)).days
predictor = StockPredictor(stock_symbol, start_date, end_date, periods=periods_to_predict)
predictor.generate_forecast()
predictor.plot_forecast()
```

```
    self.load_and_preprocess_data()
    self.split_data()
    self.train_prophet_model()
def load_and_preprocess_data(self):
    stock_data = fetch_data(self.stock_symbol, self.start_date, self.end_date)
    self.data = preprocess_data(stock_data)
def split_data(self):
    train_size = int(self.split_ratio * len(self.data))
    self.train_data = self.data[:train_size]
    self.test_data = self.data[train_size:]
def train_prophet_model(self):
    self.model = Prophet(
        changepoint_prior_scale=0.05,
        holidays_prior_scale=15,
        seasonality_prior_scale=10,
        weekly_seasonality=True,
        yearly_seasonality=True,
        daily_seasonality=False
    )
    self.model.add_country_holidays(country_name='US')
    self.model.fit(self.train_data)
def generate_forecast(self, on_test_data=False):
    if on_test_data:
        future = pd.DataFrame(self.test_data['ds'])
    else:
        future = self.model.make_future_dataframe(periods=self.periods)
    self.forecast = self.model.predict(future)
def plot_forecast(self):
    fig = self.model.plot(self.forecast)
    plt.show()
def plot_components(self):
    return plot_components_plotly(self.model, self.forecast)
```



How to Evaluate Prophet model

How to Evaluate Prophet model



Loss Function
Error = prediction — actual

Mean Square Error (MSE)

$$MSE = \frac{1}{n} * \sum (prediction - actual)^2$$

MAE is a valuable tool for evaluating and comparing forecasting models in financial applications due to its robustness, interpretability, and compatibility with optimization algorithms.

Root Mean Square Error (RMSE)

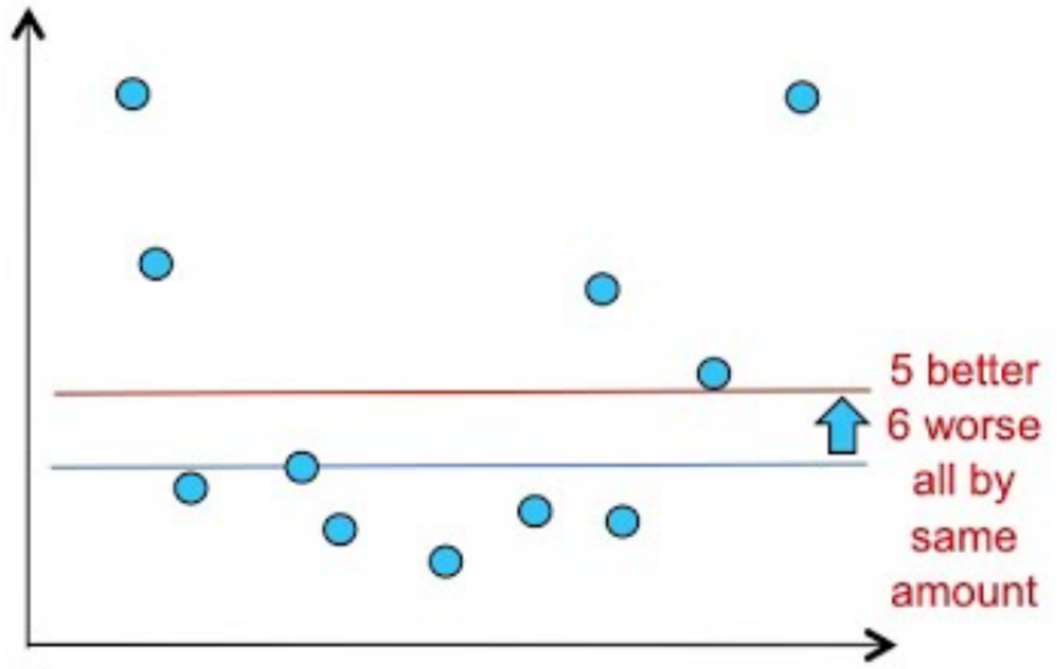
$$RMSE = \sqrt{\frac{1}{n} * \sum (prediction - actual)^2}$$

RMSE, which will measure the tolerance between the actual value, And the estimated value from the model How much is the difference?

If the RMSE value is equal to zero, Will mean the estimated model is equal to the actual value.

https://shorturl.asia/43hZ5

How to Evaluate Prophet model



Mean Absolute Error (MAE):

- less sensitive to outliers
- many small errors = one large error
- best 0^{th} order baseline: median$\{y_i\}$
  - not the mean as for MSE

$$MAE = \frac{1}{n} * \sum |prediction - actual|$$

https://medium.com/@615162020027/metrics-พื้นฐานสำหรับวัดประสิทธิภาพของโมเดล-machine-learning-c00fcc32fa30

```
def evaluate_model(self):
    def calculate_mae(actual, predicted):
        return mean_absolute_error(actual, predicted)
    def calculate_mse(actual, predicted):
        return mean_squared_error(actual, predicted)
    def calculate_rmse(actual, predicted):
        return math.sqrt(mean_squared_error(actual, predicted))
    self.generate_forecast(on_test_data=True)
    predictions = self.forecast.tail(len(self.test_data))['yhat']
    mae = calculate_mae(self.test_data['y'], predictions)
    mse = calculate_mse(self.test_data['y'], predictions)
    rmse = calculate_rmse(self.test_data['y'], predictions)

    return mae, mse, rmse
```

```python
class StockPredictor:
    def __init__(self, stock_symbol, start_date, end_date, periods=365, split_ratio=0.8):
        self.stock_symbol = stock_symbol
        self.start_date = start_date
        self.end_date = end_date
        self.periods = periods
        self.split_ratio = split_ratio
        self.model = None
        self.forecast = None

        self.load_and_preprocess_data()
        self.split_data()
        self.train_prophet_model()
    def load_and_preprocess_data(self):
        stock_data = fetch_data(self.stock_symbol, self.start_date, self.end_date)
        self.data = preprocess_data(stock_data)
    def split_data(self):
        train_size = int(self.split_ratio * len(self.data))
        self.train_data = self.data[:train_size]
        self.test_data = self.data[train_size:]
    def train_prophet_model(self):
        self.model = Prophet(
            changepoint_prior_scale=0.05,
            holidays_prior_scale=15,
            seasonality_prior_scale=10,
            weekly_seasonality=True,
            yearly_seasonality=True,
            daily_seasonality=False
        )
        self.model.add_country_holidays(country_name='US')
        self.model.fit(self.train_data)
    def generate_forecast(self, on_test_data=False):
        if on_test_data:
            future = pd.DataFrame(self.test_data['ds'])
        else:
            future = self.model.make_future_dataframe(periods=self.periods)
        self.forecast = self.model.predict(future)
    def plot_forecast(self):
        fig = self.model.plot(self.forecast)
        plt.show()
    def plot_components(self):
        return plot_components_plotly(self.model, self.forecast)
    def evaluate_model(self):
        def calculate_mae(actual, predicted):
            return mean_absolute_error(actual, predicted)
        def calculate_mse(actual, predicted):
            return mean_squared_error(actual, predicted)
        def calculate_rmse(actual, predicted):
            return math.sqrt(mean_squared_error(actual, predicted))
        self.generate_forecast(on_test_data=True)
        predictions = self.forecast.tail(len(self.test_data))['yhat']
        mae = calculate_mae(self.test_data['y'], predictions)
        mse = calculate_mse(self.test_data['y'], predictions)
        rmse = calculate_rmse(self.test_data['y'], predictions)

        return mae, mse, rmse
```

```python
mae, mse, rmse = predictor.evaluate_model()
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Forecast data range: 2021-04-12 00:00:00 - 2024-02-05 00:00:00
Mean Absolute Error (MAE): 31.625434836366015
Mean Squared Error (MSE): 1513.2944299330786
Root Mean Squared Error (RMSE): 38.90108520251175
```

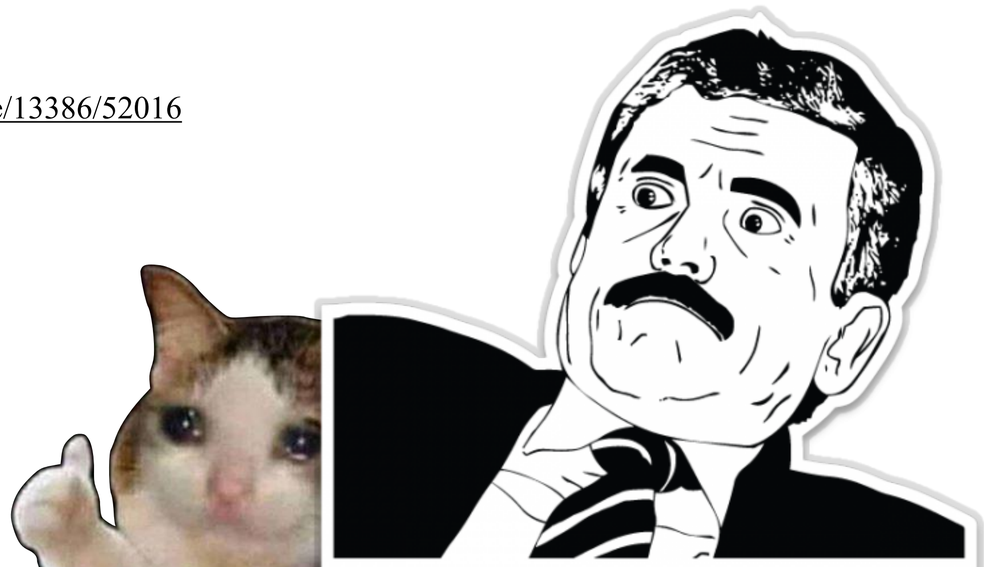
REFERENCE

Prophet

https://medium.com/qunt-i-love-u/ทำนายข้อมูล-time-series-ด้วย-prophet-in-python-d16bd7b5be13
https://medium.com/qunt-i-love-u/การพยากรณ์ข้อมูลอนุกรมเวลาด้วยเทคนิค-arima-ด้วย-python-44809eb8e990
https://facebook.github.io/prophet/
https://github.com/facebook/prophet

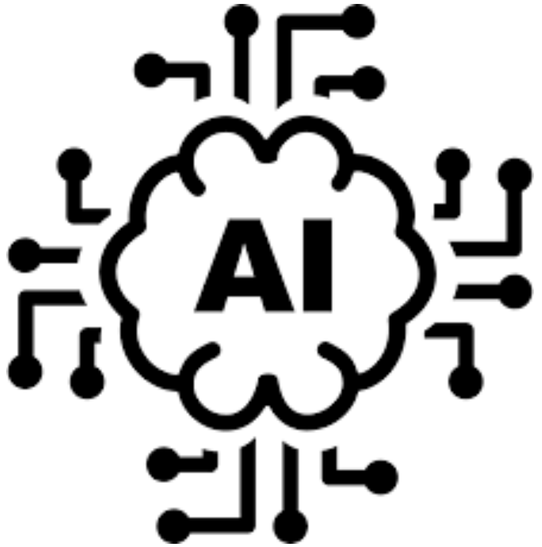
Linear regression

https://medium.com/@phiphatchomchit/คู่มือสำหรับ-linear-regression-แนวคิด-และ-การใช้งาน-273b10b55472
https://th.tradingview.com/scripts/linearregression/
https://dev.to/ketnas/sraang-simple-linear-regression-model-odyaich-python-3ofo
https://archive.cm.mahidol.ac.th/bitstream/123456789/3140/1/TP FM.016 2562.pdf
https://kukr.lib.ku.ac.th/kukr_es/index.php?/BKN/search_detail/dowload_digital_file/13386/52016



Question Time

AI

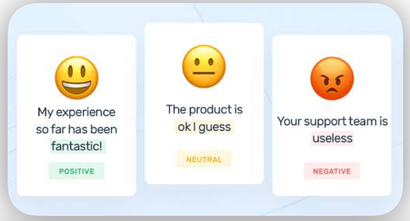


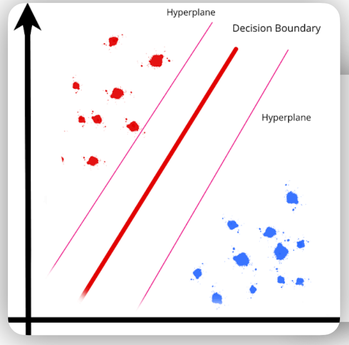
Model

Model

Prophet

Neural Network





SVM
(Support Vector Machines)

Sentiment Analysis

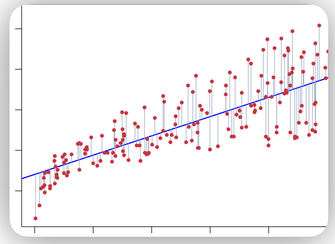
Reinforcement Learning

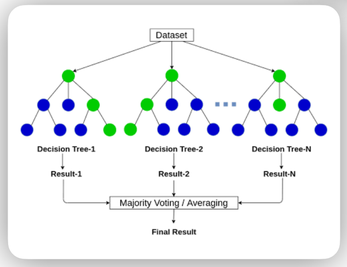
LSTM
(Long Short-Term Memory)

GRU
(Gated Recurrent Unit)

ARIMA
(AutoRegressive Integrated Moving Average)

Linear Regression





Random Forest

GARCH
(Generalized AutoRegressive Conditional Heteroskedasticity)

Evaluate Prophet model

Evaluate Prophet model

```
stock_symbol = 'AAPL'
start_date = '2010-01-01'
end_date = '2025-01-01'
```

```
mae, mse, rmse = predictor.evaluate_model()
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Forecast data range: 2021-04-12 00:00:00 - 2024-02-05 00:00:00
Mean Absolute Error (MAE): 31.625434836366015
Mean Squared Error (MSE): 1513.2944299330786
Root Mean Squared Error (RMSE): 38.90108520251175
```

```
stock_symbol = 'AAPL'
start_date = '2010-01-01'
end_date = '2024-03-01'
```

```
mae, mse, rmse = predictor.evaluate_model()
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Forecast data range: 2021-04-12 00:00:00 - 2024-02-05 00:00:00
Mean Absolute Error (MAE): 31.32888116944116
Mean Squared Error (MSE): 1484.4198343660971
Root Mean Squared Error (RMSE): 38.52816936172931
```

Evaluate Prophet model

```
stock_symbol = 'AAPL'
start_date = '2010-01-01'
end_date = '2025-01-01'
```

```
mae, mse, rmse = predictor.evaluate_model()
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

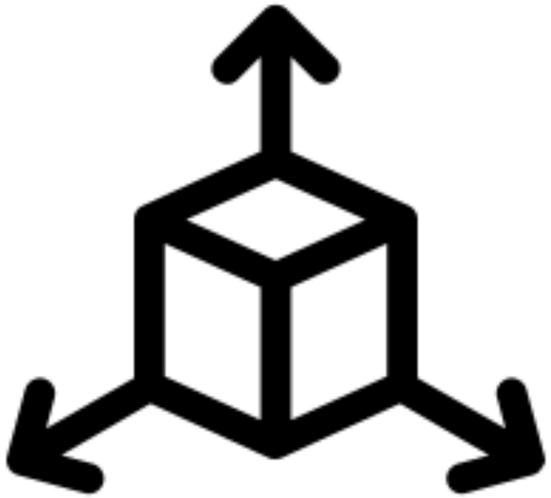
```
Forecast data range: 2021-04-12 00:00:00 - 2024-02-05 00:00:00
Mean Absolute Error (MAE)  31.625434836366015
Mean Squared Error (MSE): 1513.2944299330786
Root Mean Squared Error (RMSE): 38.90108520251175
```

```
stock_symbol = 'AAPL'
start_date = '2010-01-01'
end_date = '2024-03-01'
```

```
mae, mse, rmse = predictor.evaluate_model()
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Forecast data range: 2021-04-12 00:00:00 - 2024-02-05 00:00:00
Mean Absolute Error (MAE): 31.32888116944116
Mean Squared Error (MSE):  1484.4198343660971
Root Mean Squared Error (RMSE): 38.52816936172931
```

DATA Size



DATA Size

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.478998	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656429	6.490200	601904800
2010-01-06	7.656429	7.686786	7.526786	7.534643	6.386964	552160000
2010-01-07	7.562500	7.571429	7.466071	7.520714	6.375157	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.417541	447610800
...	...	...	...	...	...	...
2024-01-30	190.940002	191.800003	187.470001	188.039993	188.039993	55859400
2024-01-31	187.039993	187.100006	184.350006	184.399994	184.399994	55467800
2024-02-01	183.990005	186.949997	183.820007	186.860001	186.860001	64885400
2024-02-02	179.860001	187.330002	179.250000	185.850006	185.850006	102518000
2024-02-05	188.149994	189.250000	185.839996	187.679993	187.679993	69574200

3546 rows × 6 columns

DATA Size

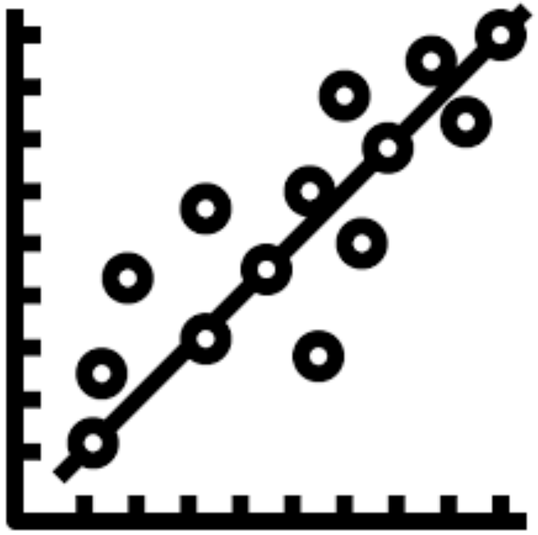
Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.478998	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656429	6.490200	601904800
2010-01-06	7.656429	7.686786	7.526786	7.534643	6.386964	552160000
2010-01-07	7.562500	7.571429	7.466071	7.520714	6.375157	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.417541	447610800
...	...	...	...	...	...	...
2024-01-30	190.940002	191.800003	187.470001	188.039993	188.039993	55859400
2024-01-31	187.039993	187.100006	184.350006	184.399994	184.399994	55467800
2024-02-01	183.990005	186.949997	183.820007	186.860001	186.860001	64885400
2024-02-02	179.860001	187.330002	179.250000	185.850006	185.850006	102518000
2024-02-05	188.149994	189.250000	185.839996	187.679993	187.679993	69574200

3546 rows × 6 columns

```
len(data)
```

```
3547
```

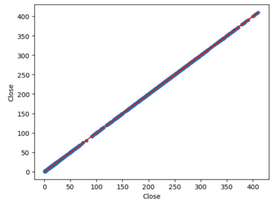
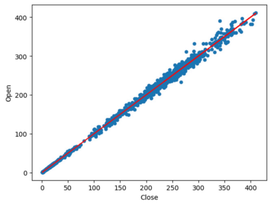
Linear Regression



Linear Regression

Measure

```
for j in Rows:
    for i in Rows:
        if i != j:
            lr.fit(data[[j]], data[i])
            print(f"Linear Regression between {j} and {i}:")
            print(f"Weight (Coefficient): {lr.coef_[0]:.2f}")
            print(f"Bias (Intercept): {lr.intercept_:.2f}\n")
```

between Close and Open:
Weight: 1.00
Bias: 0.02

between Close and High:
Weight: 1.02
Bias: -0.03

between Close and Low:
Weight: 0.98
Bias: 0.04

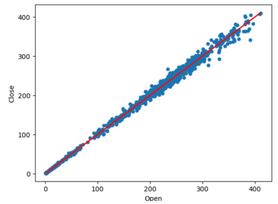
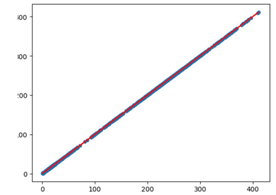
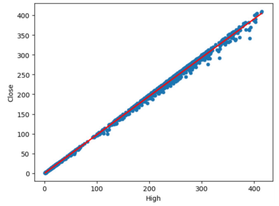
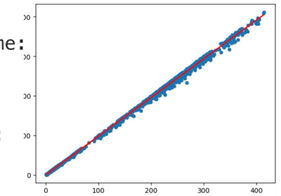
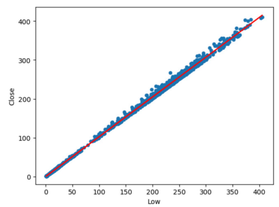
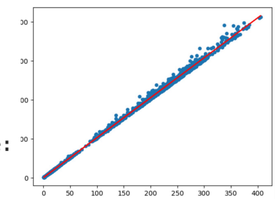
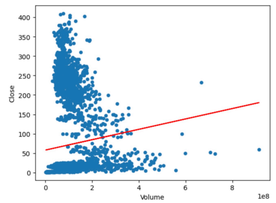
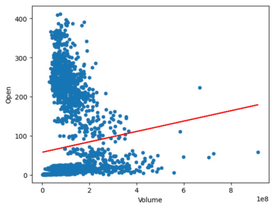
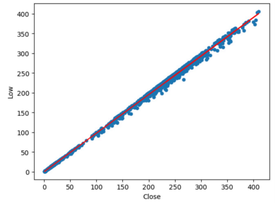
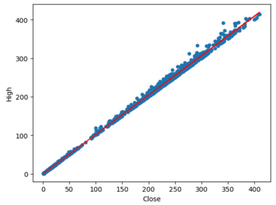
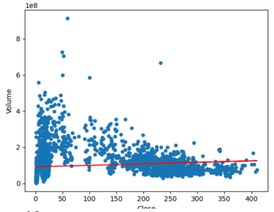
between Close and Volume:
Weight: 82800.45
Bias: 90952548.64

between Open and Close:
Weight: 1.00
Bias: 0.09

between Open and High:
Weight: 1.02
Bias: 0.00

between Open and Low:
Weight: 0.98
Bias: 0.08

between Open and Volume:
Weight: 81999.38
Bias: 91007450.96

between High and Close:
Weight: 0.98
Bias: 0.08

between High and Open:
Weight: 0.98
Bias: 0.04

between High and Low:
Weight: 0.95
Bias: 0.10

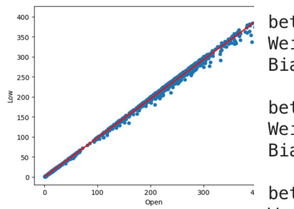
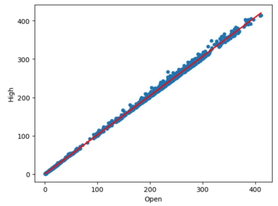
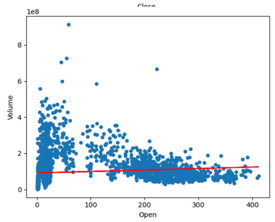
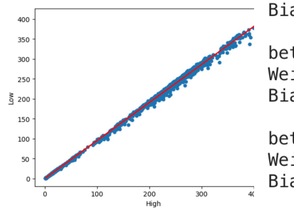
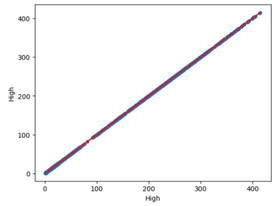
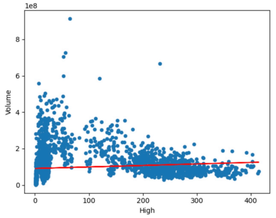
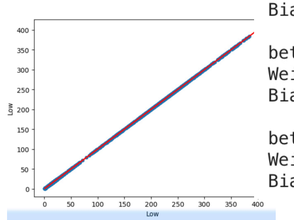
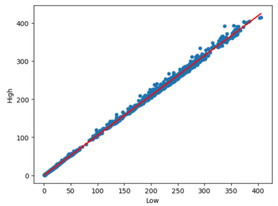
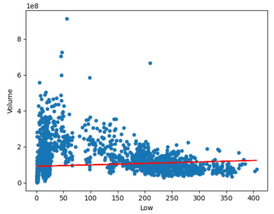
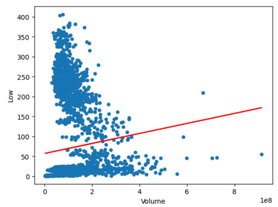
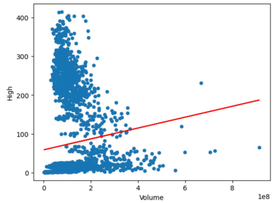
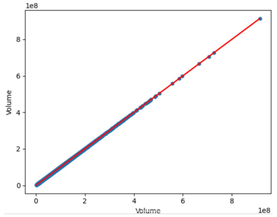
between High and Volume:
Weight: 82925.59
Bias: 90812837.56

between Low and Close:
Weight: 1.02
Bias: 0.00

between Low and Open:
Weight: 1.02
Bias: -0.03

between Low and High:
Weight: 1.05
Bias: -0.05

between Low and Volume:
Weight: 81454.64
Bias: 91182800.22

between Volume and Close:
Weight: 0.00
Bias: 57.86

between Volume and Open:
Weight: 0.00
Bias: 57.99

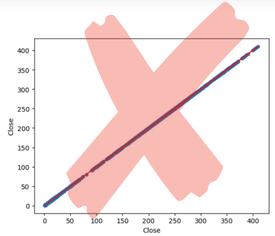
between Volume and High:
Weight: 0.00
Bias: 58.82

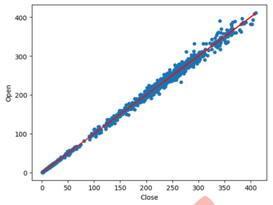
between Volume and Low:
Weight: 0.00
Bias: 57.01

Linear Regression

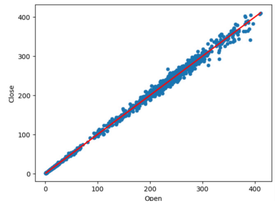
Measure

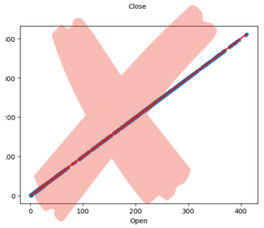
```
for j in Rows:
    for i in Rows:
        if i != j:
            lr.fit(data[[j]], data[i])
            print(f"Linear Regression between {j} and {i}:")
            print(f"Weight (Coefficient): {lr.coef_[0]:.2f}")
            print(f"Bias (Intercept): {lr.intercept_:.2f}\n")
```





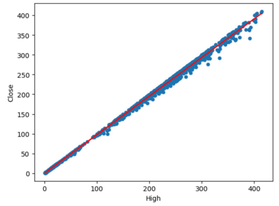
between Close and Open:
Weight: 1.00
Bias: 0.02

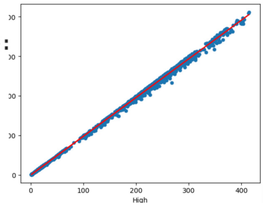




between Close and High:
Weight: 1.02
Bias: -0.03

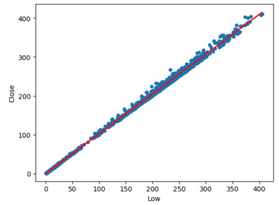
between Close and Low:
Weight: 0.98
Bias: 0.04

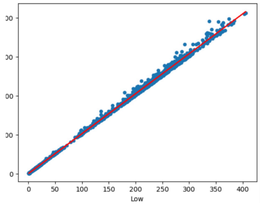




between Close and Volume:
Weight: 82800.45
Bias: 90952548.64

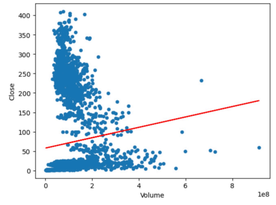
between Open and Close:
Weight: 1.00
Bias: 0.09

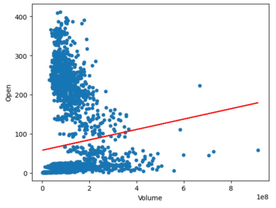




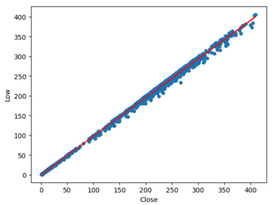
between Open and High:
Weight: 1.02
Bias: 0.00

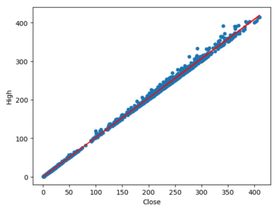
between Open and Low:
Weight: 0.98
Bias: 0.08

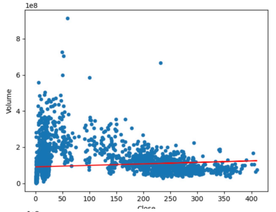


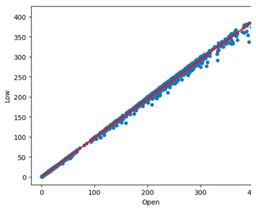


between Open and Volume:
Weight: 81999.38
Bias: 91007450.96



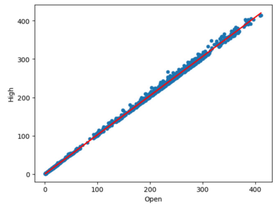


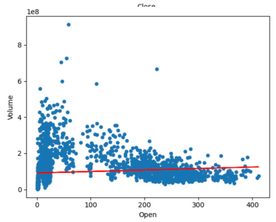




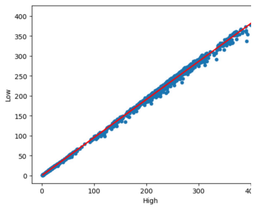
between High and Close:
Weight: 0.98
Bias: 0.08

between High and Open:
Weight: 0.98
Bias: 0.04

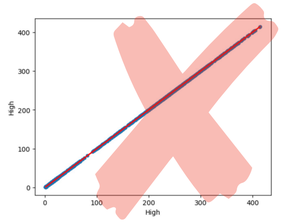


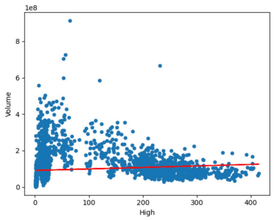


between High and Low:
Weight: 0.95
Bias: 0.10



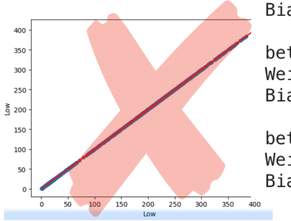
between High and Volume:
Weight: 82925.59
Bias: 90812837.56





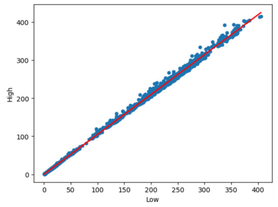
between Low and Close:
Weight: 1.02
Bias: 0.00

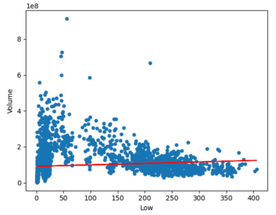
between Low and Open:
Weight: 1.02
Bias: -0.03

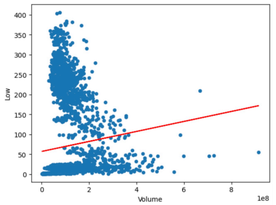


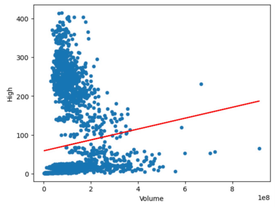
between Low and High:
Weight: 1.05
Bias: -0.05

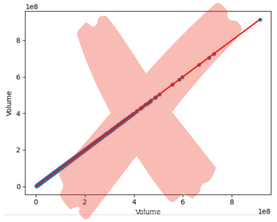
between Low and Volume:
Weight: 81454.64
Bias: 91182800.22











between Volume and Close:
Weight: 0.00
Bias: 57.86

between Volume and Open:
Weight: 0.00
Bias: 57.99

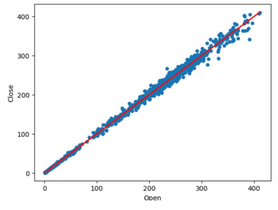
between Volume and High:
Weight: 0.00
Bias: 58.82

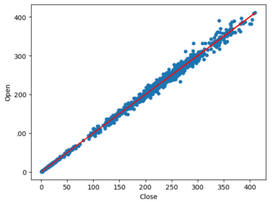
between Volume and Low:
Weight: 0.00
Bias: 57.01

Linear Regression

Measure

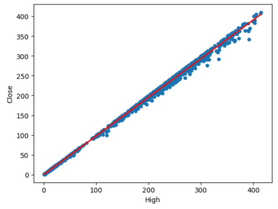
```
for j in Rows:
    for i in Rows:
        if i != j:
            lr.fit(data[[j]], data[i])
            print(f"Linear Regression between {j} and {i}:")
            print(f"Weight (Coefficient): {lr.coef_[0]:.2f}")
            print(f"Bias (Intercept): {lr.intercept_:.2f}\n")
```

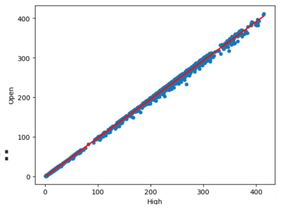




between Close and Open:
Weight: 1.00
Bias: 0.02

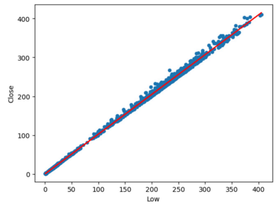
between Close and High:
Weight: 1.02
Bias: -0.03

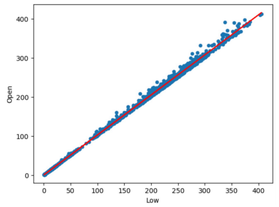




between Close and Low:
Weight: 0.98
Bias: 0.04

between Close and Volume:
Weight: 82800.45
Bias: 90952548.64

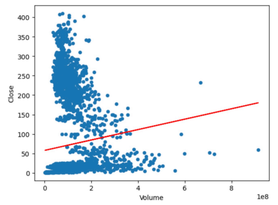


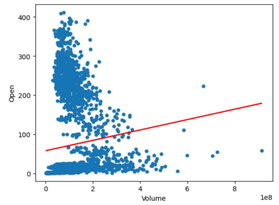


between Open and Close:
Weight: 1.00
Bias: 0.09

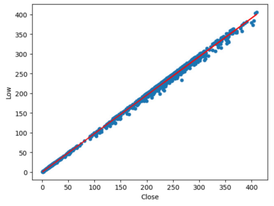
between Open and High:
Weight: 1.02
Bias: 0.00

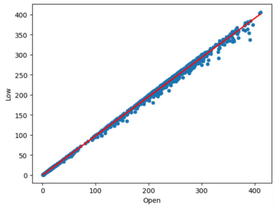
between Open and Low:
Weight: 0.98
Bias: 0.08





between Open and Volume:
Weight: 81999.38
Bias: 91007450.96

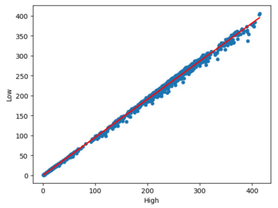


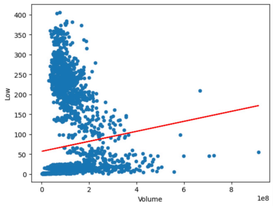


between High and Close:
Weight: 0.98
Bias: 0.08

between High and Open:
Weight: 0.98
Bias: 0.04

between High and Low:
Weight: 0.95
Bias: 0.10





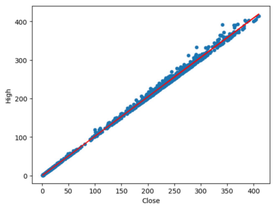
between High and Volume:
Weight: 82925.59
Bias: 90812837.56

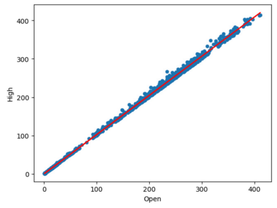
between Low and Close:
Weight: 1.02
Bias: 0.00

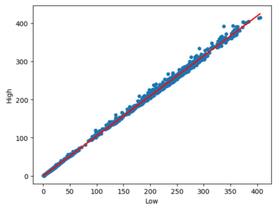
between Low and Open:
Weight: 1.02
Bias: -0.03

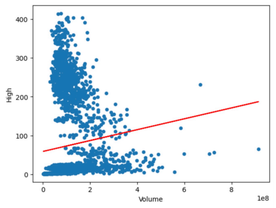
between Low and High:
Weight: 1.05
Bias: -0.05

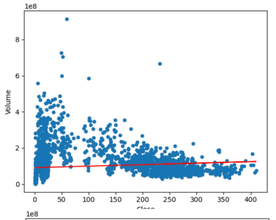
between Low and Volume:
Weight: 81454.64
Bias: 91182800.22

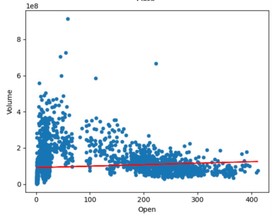


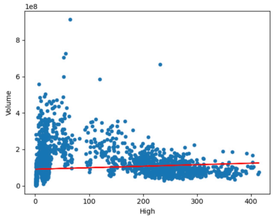


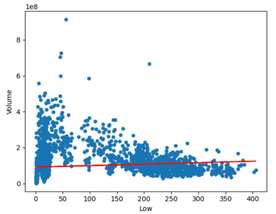












between Volume and Close:
Weight: 0.00
Bias: 57.86

between Volume and Open:
Weight: 0.00
Bias: 57.99

between Volume and High:
Weight: 0.00
Bias: 58.82

between Volume and Low:
Weight: 0.00
Bias: 57.01

TEAM

Chayanapas Kongsamrong 6652300630

Varanpong Suthiponpisal 6652300788