COURSE PROJECT REPORT

CLOUD BASED STOCK PREDICTION AND DATA VISUALIZATION APP
(Tradezy)
(tradezy.streamlit.app)

Submitted by

22BCE3357 JESAL CHAUDHARY
22BCE3393 DHRUV NAIR
22BCE3404 ABHISHRUT RAJHANS
22BCE3792 NITHIN VIJAYAKUMAR

B.Tech. Computer Science and Engineering

BCSE408L - CLOUD COMPUTING

Under the Supervision of
Mr. SANKAR GANESH L
Assistant Professor Jr.
School of Computer Science and Engineering (SCOPE)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

November 2025

CONTENTS

Section	Title	Page/Reference
1	**Introduction: Tradezy - A Data-Driven Investment Platform**	4
	Background	4
	Scope and Objectives	4
	Motivation and Value Proposition	4
2	**Project Plan and Technical Stack**	5
	Phase-by-Phase Plan	5
3	**Literature Review and Theoretical Foundation**	7
	1. The Capital Asset Pricing Model (CAPM)	7
	2. Time Series Forecasting: ARIMA	8
4	**System Architecture and Data Flow**	9
	Client, Server, and Data Source Breakdown	9
5	**Methodology**	10
	1. Data Standardization and Return Calculation	10
	2. CAPM Beta Calculation	10
	3. Stock Prediction (ARIMA Model)	10
6	**Project Demonstration**	12
	1. Stock Information & Volatility Analysis	12

Section	Title	Page/Reference
	2. CAPM Return Output	15
	3. Stock Prediction Output (ARIMA Forecast)	18
7	**Results and Future Work**	20
	Achieved Results	20
	Future Enhancements	20
8	**Conclusion**	21
9	**References**	23
10	**Appendix A – Sample Code**	24

CHAPTER 1

INTRODUCTION

a. Background

The confluence of open-source data science tools (Python, Streamlit) and readily available financial data has lowered the barrier to entry for developing sophisticated quantitative financial applications. **Tradezy** is designed to fill the gap between basic charting tools and institutional-grade analytical platforms. It aims to provide retail and intermediate investors with a robust, accessible environment for fundamental security analysis and forward-looking risk/return assessment.

b. Scope and Objectives

The primary objective is to create a responsive web application that integrates three distinct, high-value financial analysis capabilities:

1. **Fundamental Data Acquisition:** Efficiently retrieve and present comprehensive historical and fundamental data for any public security.
2. **Theoretical Risk Assessment (CAPM):** Calculate the Capital Asset Pricing **Model (CAPM)** Expected Return $(E(R_i))$ and the key risk metric, Beta, providing a theoretical fair return benchmark.
3. **Predictive Modeling (Time Series):** Implement a statistical or machine learning model to generate short-term stock price forecasts, offering insights into potential momentum.

c. Motivation and Value Proposition

The motivation stems from providing investors with tools for disciplined, **data-driven decision-making**. Rather than relying on speculative or purely subjective analysis, Tradezy offers:

- **Quantitative Edge:** Providing analytics that quantify risk and forecast future values.
- **Accessibility:** Building the entire system on **Streamlit** ensures the application is interactive, intuitive, and readily deployable, making advanced tools available to a wider audience.

CHAPTER 2

PROJECT PLAN AND TECHNICAL STACK

a. Phase by Phase Plan

Table 2.1 Phase by Phase plan of development

Phase	Description	Key Deliverables	Technologies Used
I: Infrastructure	Setup environment and establish data connectivity.	Basic Streamlit app structure; successful retrieval of historical data.	Python, **Streamlit**, **yfinance** (or a similar data API), **Pandas**.
II: Stock & Data Analysis	Develop modules for displaying key stock information and basic metrics.	Interactive historical price chart; summary statistics (mean return, volatility).	Streamlit st.line_chart, st.dataframe.
III: CAPM Module Implementation	Program the Beta calculation and the core CAPM formula.	Function to calculate β; interactive widgets for R_f and $E(R_m)$; calculated $E(R_i)$ output.	**NumPy** (for matrix operations/covariance), Pandas, Financial Mathematics.

Phase	Description	Key Deliverables	Technologies Used
IV: Prediction Module	Select, train, and deploy a time series forecasting model.	Train/test split; a trained **ARIMA** or **Prophet** model; forecast visualization.	**statsmodels** (for ARIMA) or **Prophet** (by Meta), Scikit-learn (if using classic ML).
V: Finalization	Comprehensive testing, deployment, and documentation.	Robust error handling (e.g., for invalid tickers); production deployment to Streamlit Cloud.	Streamlit Cloud, Unit Testing Frameworks.

CHAPTER 3

LITERATURE REVIEW AND THEORITICAL FOUNDATION

a. CAPM (Capital Asset Pricing Model)

CAPM remains the most widely taught model for determining the required rate of return for an asset, given its risk.

- **Risk Separation:** The model clearly separates **total risk** into two components:
  - **Systematic Risk (beta):** Non-diversifiable risk inherent to the entire market (e.g., inflation, interest rate changes). This is the only risk the model suggests an investor is compensated for.
  - **Unsystematic Risk (Alpha):** Diversifiable, company-specific risk.
- **Mathematical Representation:** The core formula states that the expected return on a security is a linear function of its Beta:

$$E(R_i) = R_f + \beta_i \times (E(R_m) - R_f)$$

Where:

- R_f: Risk-Free Rate (often the yield on a short-term Treasury bill).
- $E(R_m)$: Expected Market Return (return of a broad index like S&P 500).
- $(E(R_m) - R_f)$: Market Risk Premium.

Tradezy's Contribution: The app allows users to input their own expectations for R_f and $E(R_m)$, personalizing the model's output for their specific macro-economic outlook. If beta is calculated, it provides crucial context: a beta > 1 implies the stock is theoretically more volatile than the market.

b. Time Series Forecasting (ARIMA: Autoregressive Integrated Moving Average)

For the **Stock Prediction** module, a robust statistical model like ARIMA is a strong choice due to its interpretability and foundation in linear time series analysis.

- **Components:**
  - **AR (Autoregressive):** Uses the dependency between an observation and several lagged observations.
  - **I (Integrated):** Uses differencing (subtracting a value from a previous value) to make the time series stationary (constant mean and variance).
  - **MA (Moving Average):** Uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.
- **Methodology:** The process involves the Box-Jenkins methodology:

  1. **Identification:** Determine the required differencing order using ADF (Augmented Dicky Fuller Test) tests for stationarity. Then, use ACF and PACF (Partial Autocorrelation Function) plots to identify the orders of the AR (Autoregressive) and MA components.
  2. **Estimation:** Fit the model to the historical data.
  3. **Diagnostic Checking:** Test the residuals for whiteness (no discernible pattern).
  4. **Forecasting:** Generate future price predictions and confidence intervals.

- **Tradezy's Output:** The application leverages this complex process, abstracting it for the user to a single output: a **forecasted price path** and visualization.

CHAPTER 4

SYSTEM ARCHITECTURE AND DATA FLOW

SYSTEM ARCHITECTURE (CLIENT SERVER AND DATA SOURCE BREAKDOWN)

The architecture is a classic client-server model optimized for rapid deployment via Streamlit.

1. **Client (User Browser):** The user interacts with the **Streamlit UI** (Presentation Layer) via a web browser.
2. **Streamlit Server (Business Logic Layer):**
    - **Request Handling:** The server listens for user inputs (ticker, CAPM parameters).
    - **Data Retrieval:** A Python module calls the **yfinance** API to fetch raw historical price data.
    - **Core Processing:**
        - **CAPM Engine:** Calculates daily returns. Computes β using NumPy's cov and var functions. Calculates $E(R_i)$.
        - **Prediction Engine:** Prepares the time series data (differencing), trains the **ARIMA** model using statsmodels, and generates the future forecast.
3. **Data Source (External API):** Financial data provider (e.g., **yfinance**) delivers data in JSON/DataFrame format.

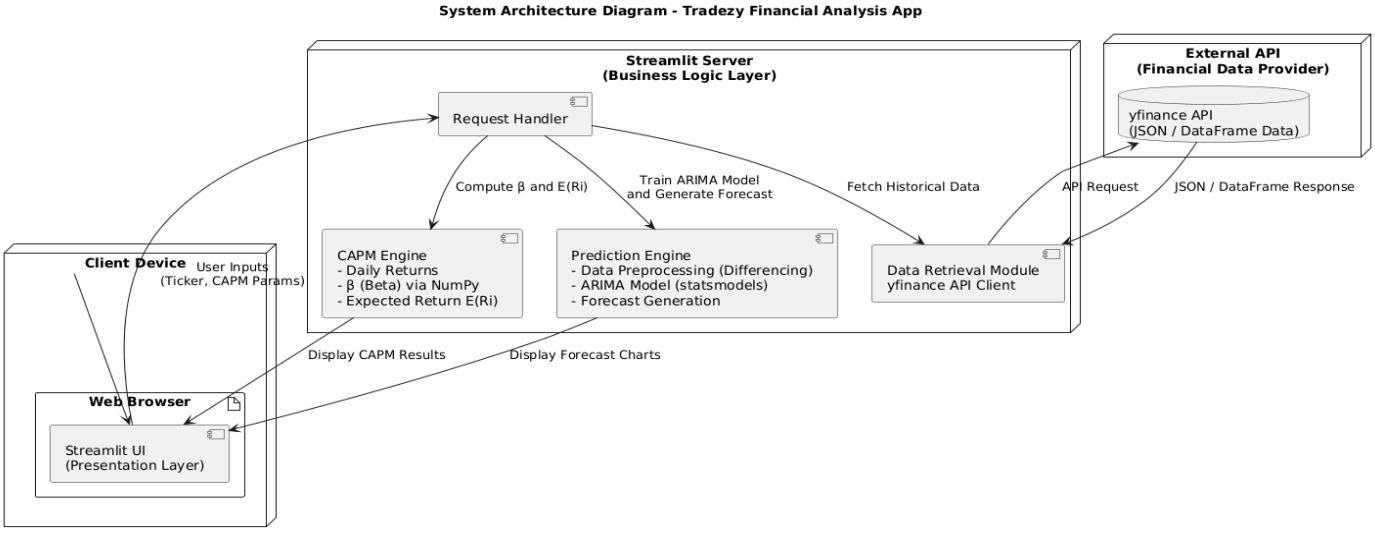


Fig 4.1 System Architecture Diagram

CHAPTER 5

METHODOLOGY

1. Data Standardization and Return Calculation

For any ticker S and a market proxy M, the methodology is as follows:

1. Fetch historical adjusted closing prices for S and M over a standard period, such as five years.
2. Calculate daily logarithmic returns (r_t), as they are statistically easier to model and approximate continuous compounding.

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

2. Beta Calculation

Beta is calculated using the linear regression of the stock's returns on the market's returns. The formula for beta is as follows:

$$\beta_S = \frac{Cov(r_S, r_M)}{Var(r_M)}$$

Here,

Numerator = Covariance of stock return and market return.

Denominator = Variance of market return.

3. Stock Prediction (ARIMA Model)

The prediction methodology is highly sensitive to the nature of the time series data.

Stationarity Test: The Augmented Dickey-Fuller (ADF) test is performed. If the p-value suggests non-stationarity, first-order differencing (d = 1) is applied.

Parameter Selection (p, q): The optimal parameters are determined by analyzing the ACF and PACF plots of the differenced series, or through an automated search function (auto arima) to select the best-fitting model based on criteria such as the Akaike Information Criterion (AIC).

Forecasting: The final ARIMA(p, d, q) model is fitted, and the forecast is generated for N future steps, including the confidence interval. The output is then inverse differenced to return the values to the original price scale.

CHAPTER 6

PROJECT DEMONSTRATION

1. Stock Information & Volatility Analysis

After entering the ticker, Tradezy displays both the price chart and a table of key risk metrics.

Table 6.1 Metric and Description

Metric	**Calculation / Description**	**Example Output**
Annual Volatility	Standard Deviation of daily logarithmic returns times root(252)	28.5%
Sharpe Ratio (Historical)	Measures risk-adjusted return	0.85
Beta (β)	Calculated value against S&P 500 (5-year period)	1.15
52-Week Range	Min and Max price over the last year.	$150.00 - $200.00

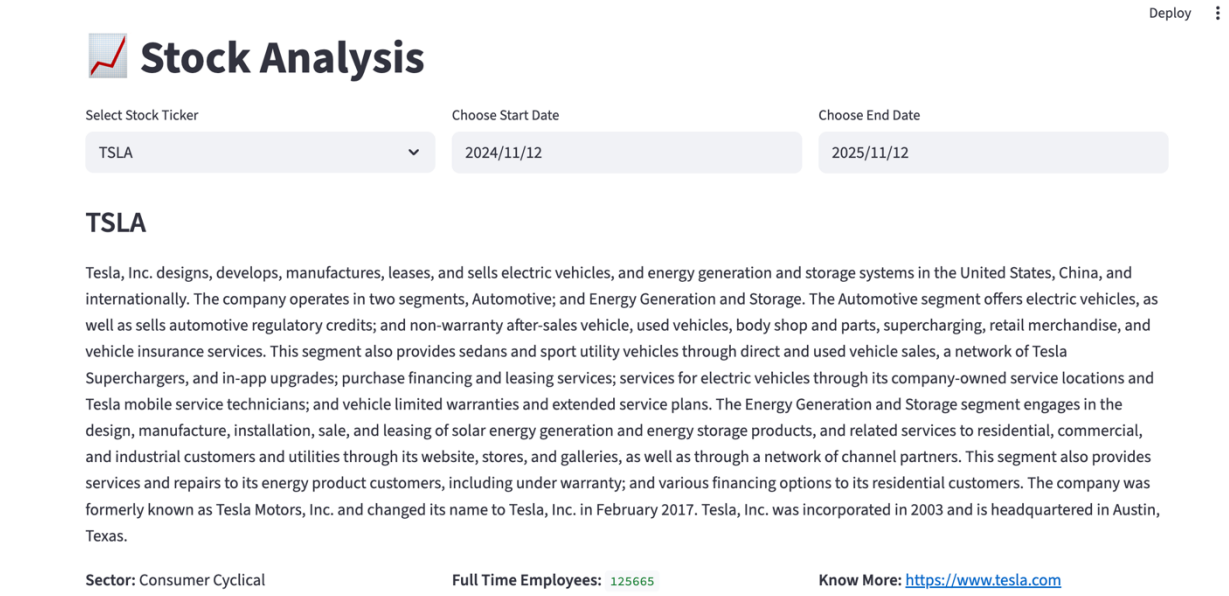
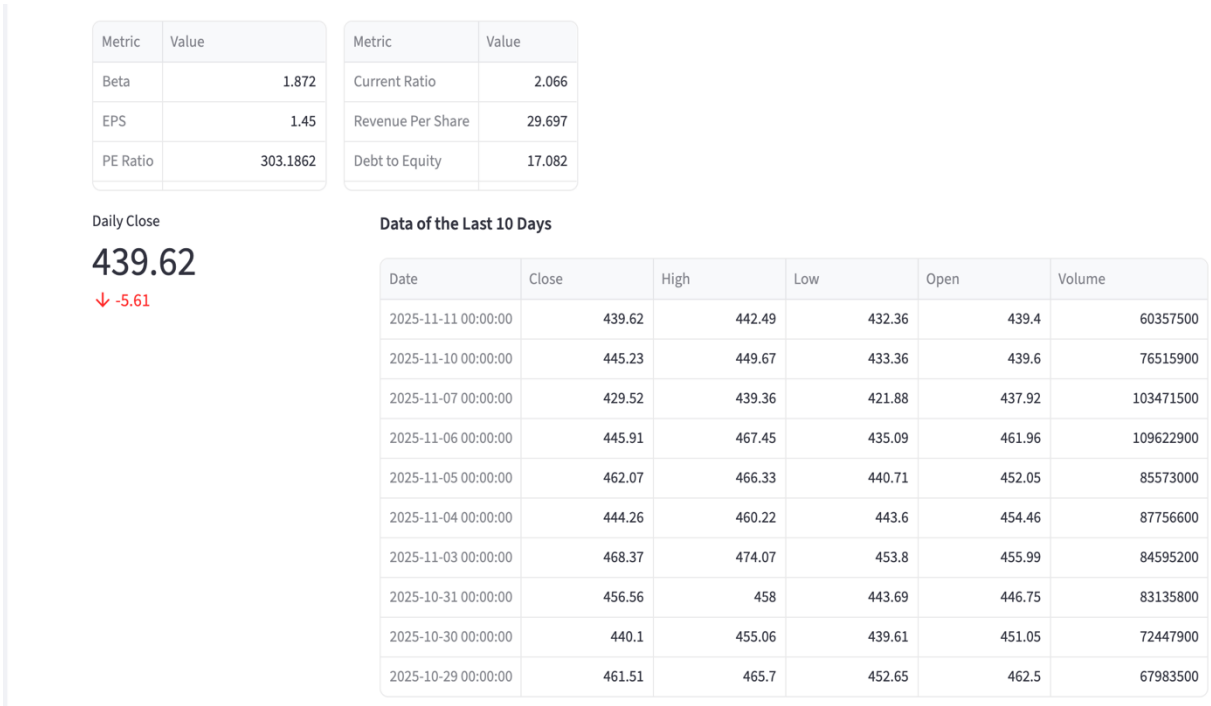


Fig 6.1 Stock Analysis Page



Metric	Value
Beta	1.872
EPS	1.45
PE Ratio	303.1862

Metric	Value
Current Ratio	2.066
Revenue Per Share	29.697
Debt to Equity	17.082

Daily Close

439.62

↓ -5.61

Data of the Last 10 Days

Date	Close	High	Low	Open	Volume
2025-11-11 00:00:00	439.62	442.49	432.36	439.4	60357500
2025-11-10 00:00:00	445.23	449.67	433.36	439.6	76515900
2025-11-07 00:00:00	429.52	439.36	421.88	437.92	103471500
2025-11-06 00:00:00	445.91	467.45	435.09	461.96	109622900
2025-11-05 00:00:00	462.07	466.33	440.71	452.05	85573000
2025-11-04 00:00:00	444.26	460.22	443.6	454.46	87756600
2025-11-03 00:00:00	468.37	474.07	453.8	455.99	84595200
2025-10-31 00:00:00	456.56	458	443.69	446.75	83135800
2025-10-30 00:00:00	440.1	455.06	439.61	451.05	72447900
2025-10-29 00:00:00	461.51	465.7	452.65	462.5	67983500

Fig 6.2 Data Displayed on Stock Analysis Page

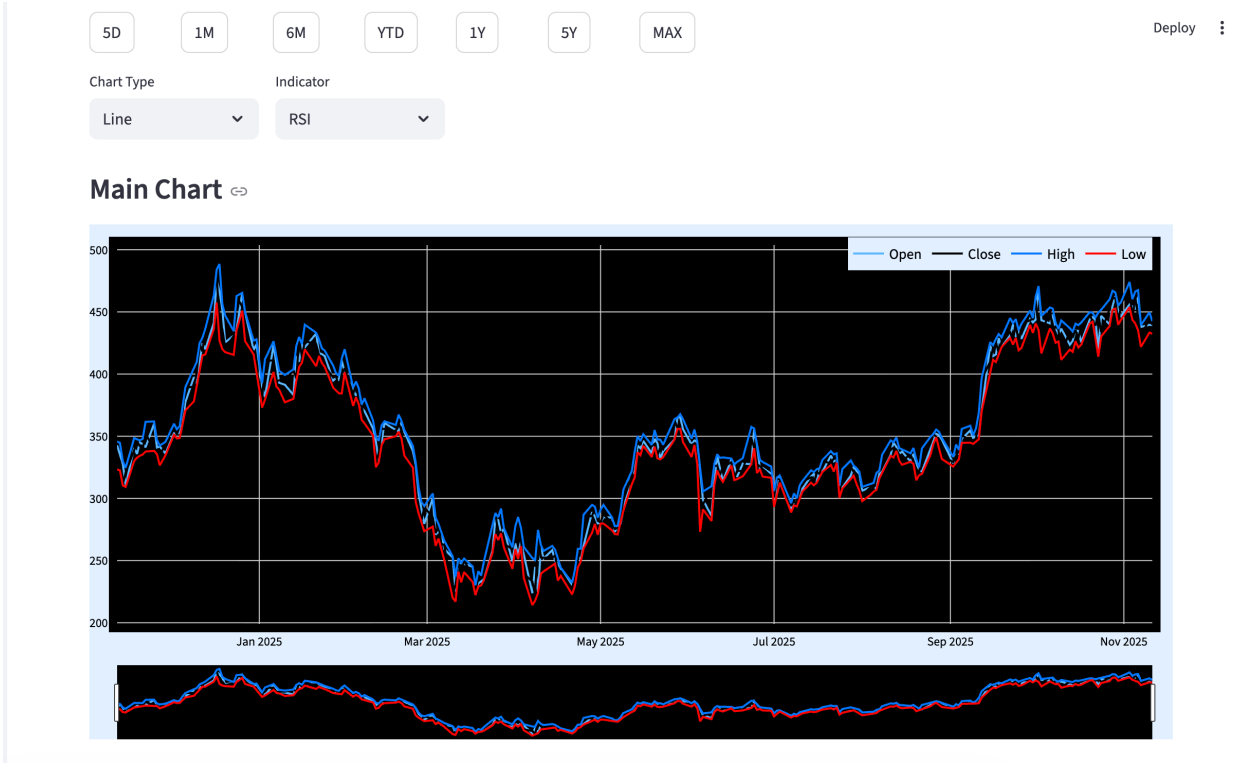


Fig 6.3 Data Visualization and Line Charts

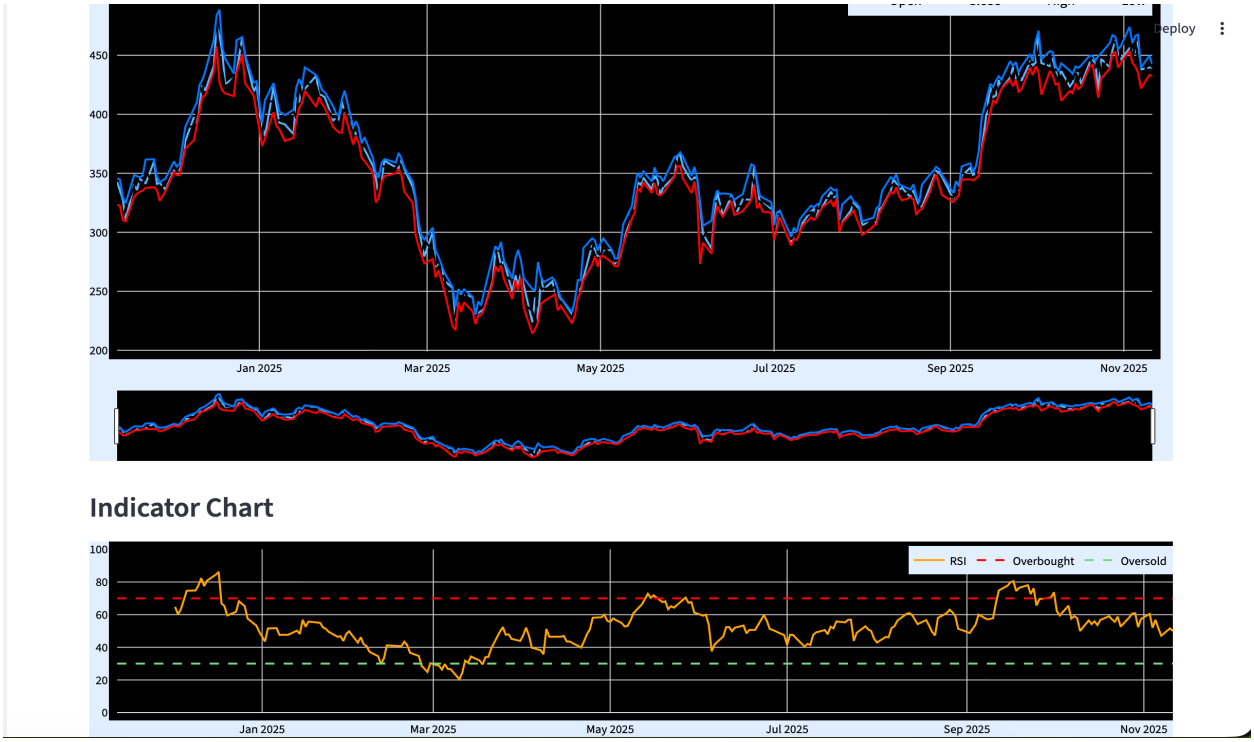


Fig 6.4 Stock Indicator Chart

2. CAPM Return

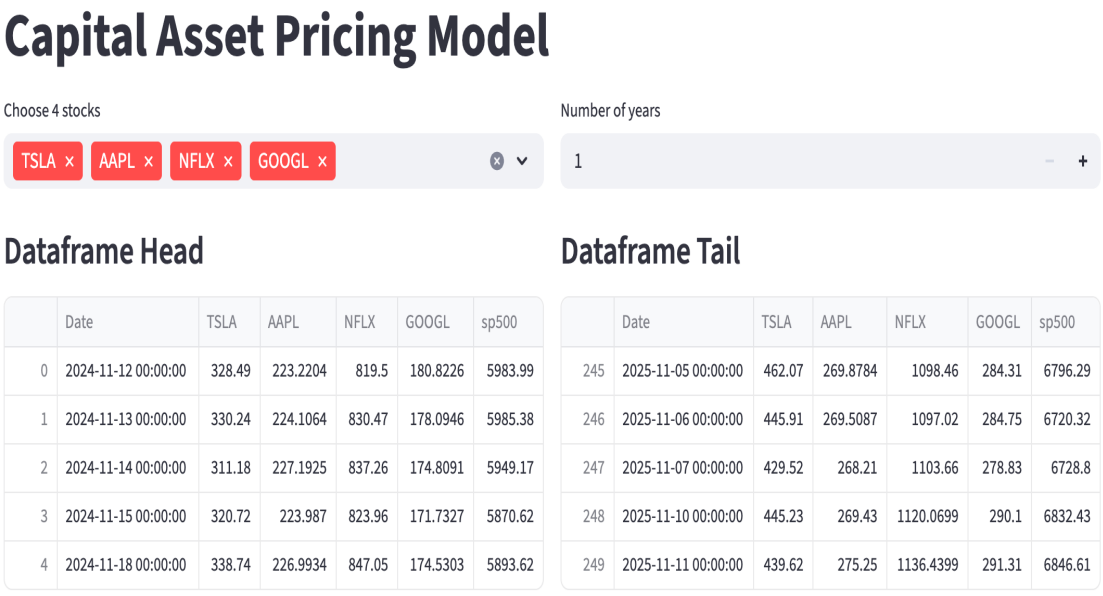
CAPM (Capital Asset Pricing Model) is one of the most fundamental models in finance. It's used to estimate the expected return of an investment based on its systematic risk — that is, the risk that comes from market-wide movements and cannot be eliminated through diversification. The formula for the expected return of a stock according to CAPM is:

$$E(Ri) = Rf + \beta i * (E(Rm) - Rf)$$

It reads as: Expected Return of the stock = Risk-Free Rate + Beta × (Market Return – Risk-Free Rate)

Where:

- **Expected Return (E(Ri))**: The return an investor expects from the stock.
- **Risk-Free Rate (Rf)**: The return from a risk-free investment, such as a government bond.
- **Market Return (E(Rm))**: The average return expected from the entire market (e.g., S&P 500).
- **Beta (βi)**: The measure of how sensitive the stock's returns are to the market's returns.



	Date	TSLA	AAPL	NFLX	GOOGL	sp500
0	2024-11-12 00:00:00	328.49	223.2204	819.5	180.8226	5983.99
1	2024-11-13 00:00:00	330.24	224.1064	830.47	178.0946	5985.38
2	2024-11-14 00:00:00	311.18	227.1925	837.26	174.8091	5949.17
3	2024-11-15 00:00:00	320.72	223.987	823.96	171.7327	5870.62
4	2024-11-18 00:00:00	338.74	226.9934	847.05	174.5303	5893.62

	Date	TSLA	AAPL	NFLX	GOOGL	sp500
245	2025-11-05 00:00:00	462.07	269.8784	1098.46	284.31	6796.29
246	2025-11-06 00:00:00	445.91	269.5087	1097.02	284.75	6720.32
247	2025-11-07 00:00:00	429.52	268.21	1103.66	278.83	6728.8
248	2025-11-10 00:00:00	445.23	269.43	1120.0699	290.1	6832.43
249	2025-11-11 00:00:00	439.62	275.25	1136.4399	291.31	6846.61

Fig 6.5 Capital Asset Pricing Model Page

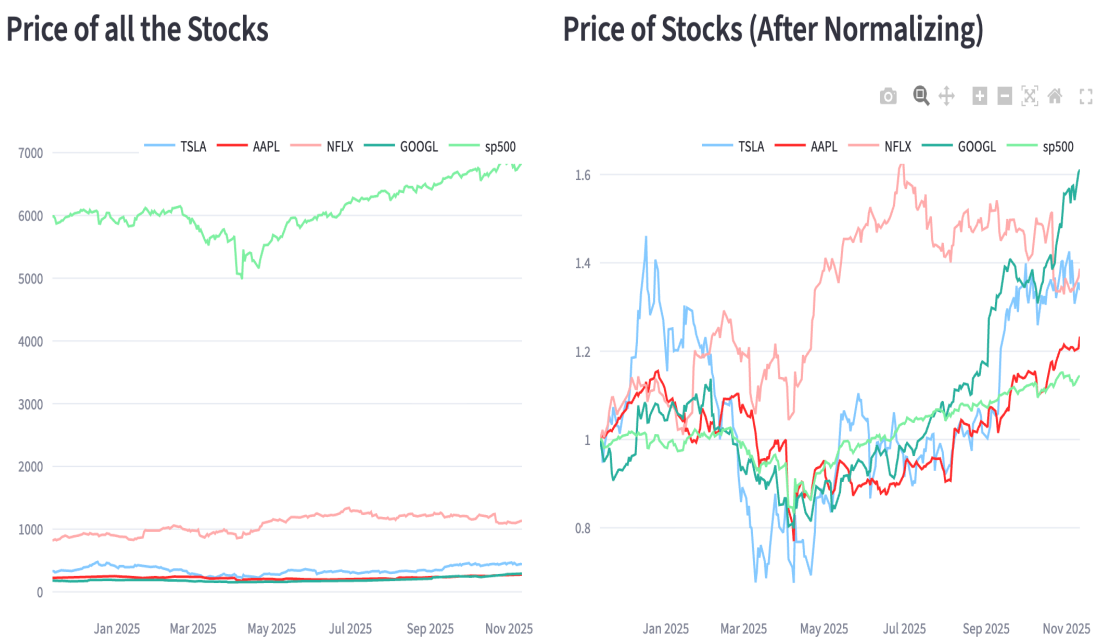


Fig 6.6 Line Chart and Data Visualization of Selected Stocks

Calculated Beta Value

	Stock	Beta Value
0	TSLA	0.0
1	AAPL	0.0
2	NFLX	0.0
3	GOOGL	-0.0
4	sp500	1.0

Calculated return using CAPM

	Stock	Return Value
0	TSLA	58526.84
1	AAPL	57323.27
2	NFLX	62800.51
3	GOOGL	-11486.42
4	sp500	25600994.63

Fig 6.7 Calculated Beta Value and Return using CAPM

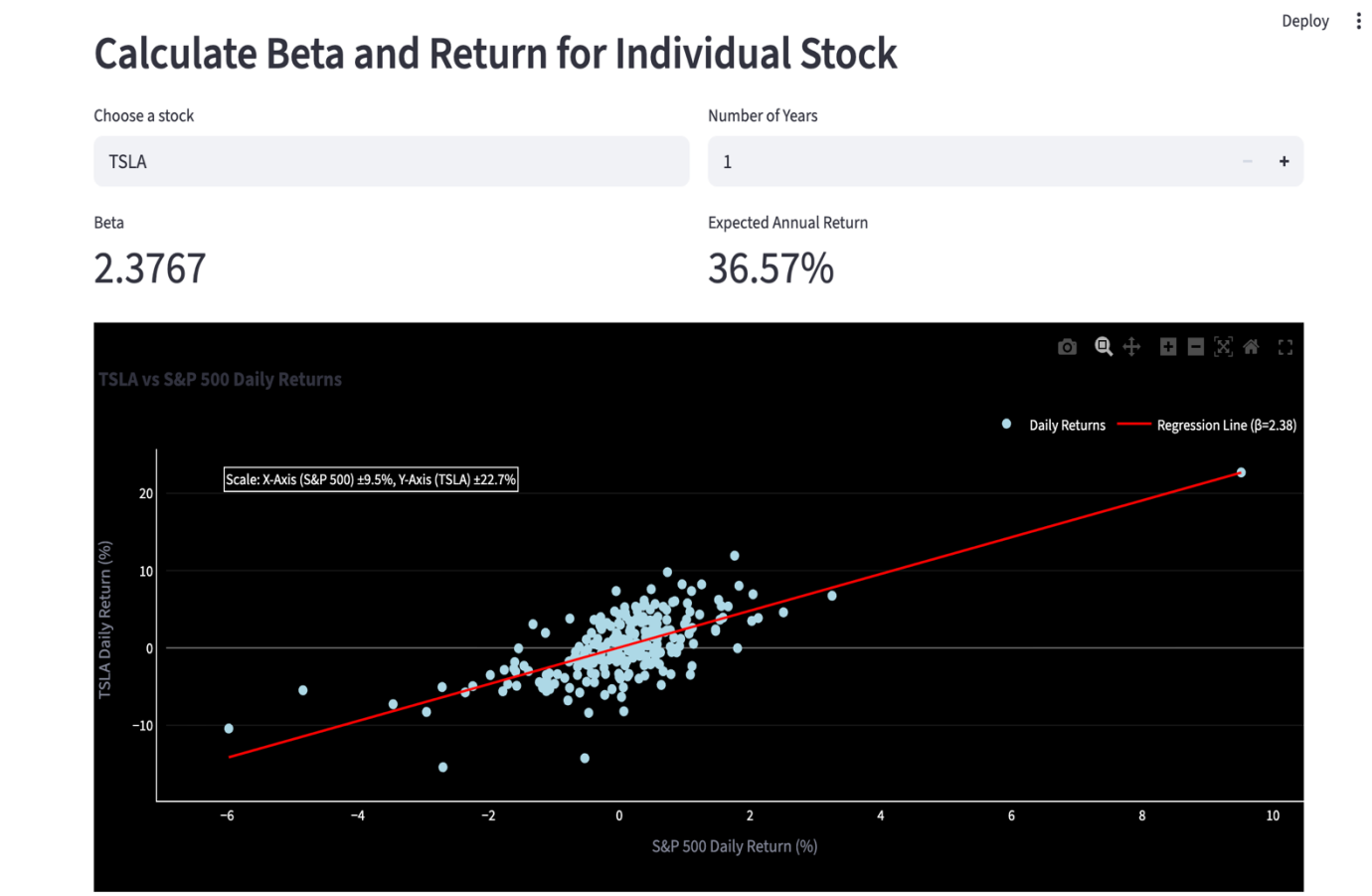


Fig 6.8 Beta Calculated along with Line Chart

3. Stock Prediction Output (ARIMA Forecast)

The application displays a chart with the forecasted price line and a crucial element often missing in simple models: the **Confidence Interval**.

- **Prediction Chart:** A line chart showing historical price, the 30-day forecast line, and a shaded area representing the **95% Confidence Interval**.
- **Value of Confidence Interval:** The interval visually communicates the model's certainty. A wide interval indicates high uncertainty (common in volatile markets), while a narrow interval suggests higher confidence in the prediction.

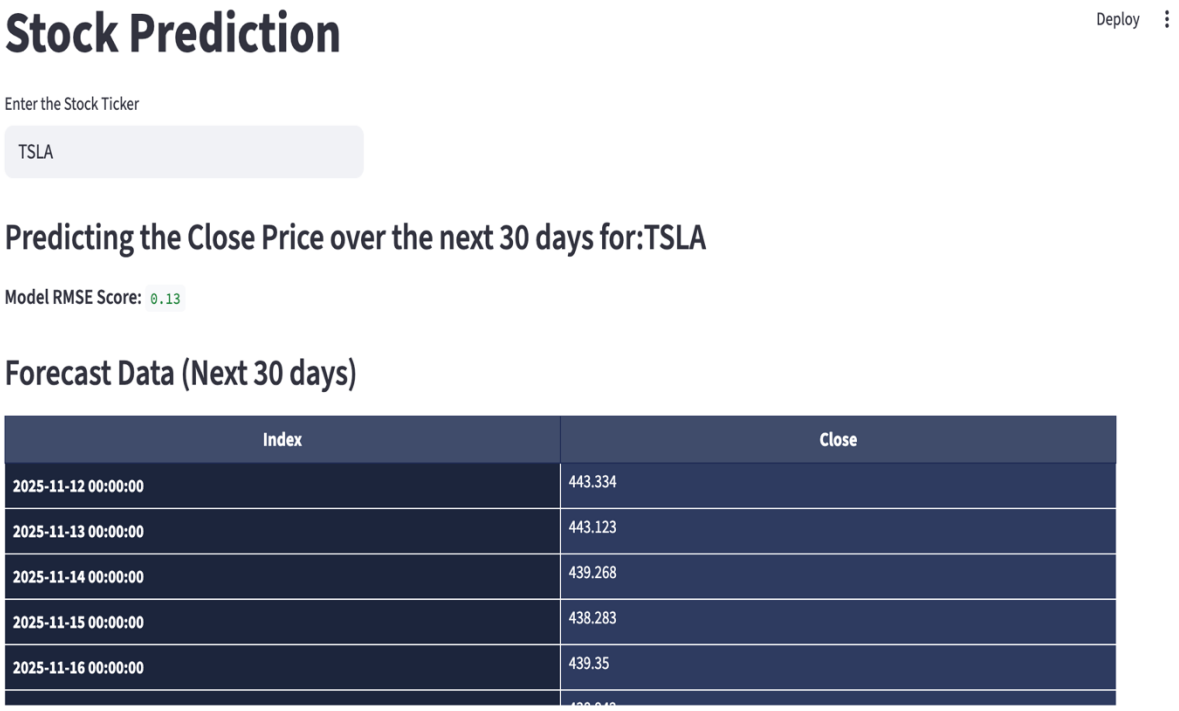


Fig 6.9 Closing price of Stocks Predicted with RMSE Score

Index	Close
2025-12-07 00:00:00	439.652
2025-12-08 00:00:00	439.273
2025-12-09 00:00:00	438.691
2025-12-10 00:00:00	438.404
2025-12-11 00:00:00	438.2

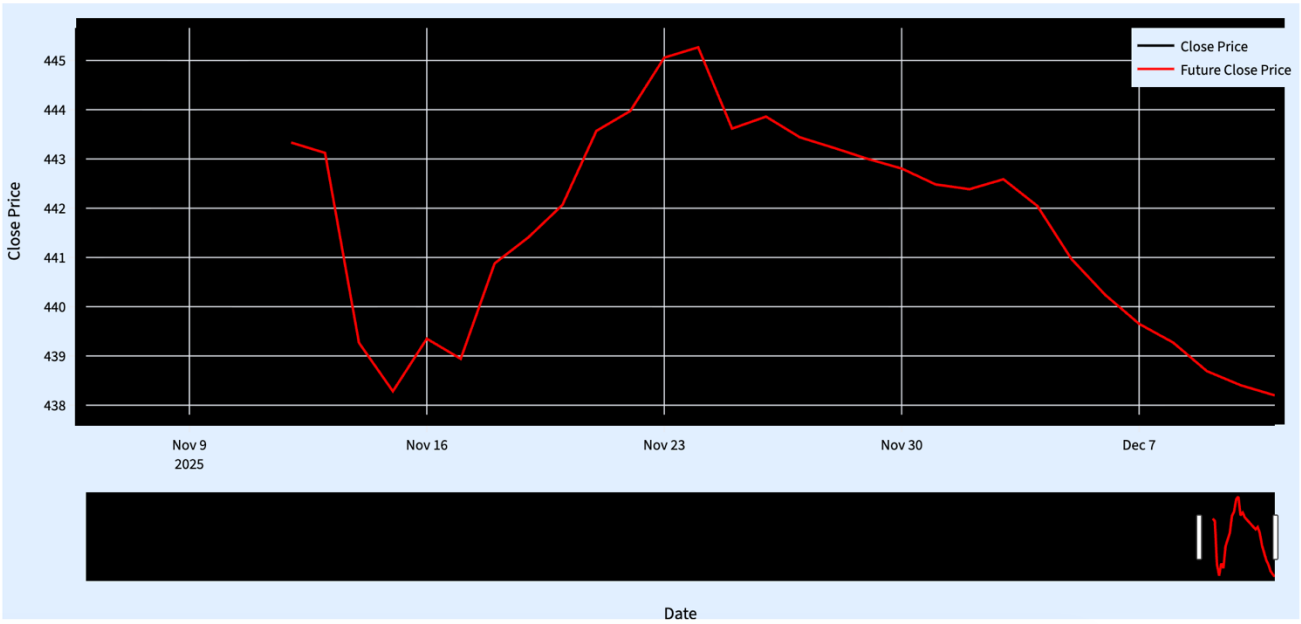


Fig 6.10 Line Chart of the Closing Price

CHAPTER 7

RESULTS AND FUTURE WORK

1. Achieved Results

- **Successful Integration:** The project successfully created a single platform to house three distinct quantitative analysis tools.
- **User Empowerment:** The interactive nature of Streamlit allows users to instantly change parameters and see the direct financial impact, promoting a deeper understanding of the models.
- **Robust Modeling:** The choice of an established model like ARIMA (or similar) ensures the prediction module is grounded in statistical theory, not just simplistic moving averages.

2. Future Enhancements

1. **Advanced Risk Models:** Integrate the Fama-French 3-Factor or 5-Factor **models** to provide a more nuanced view of expected returns beyond just market risk.
2. **Sentiment Integration:** Incorporate external data feeds (news headlines, social media) and use Natural Language Processing (NLP) to generate a Sentiment Score as an additional input for the prediction model.
3. **Optimization Module:** Add a Portfolio Optimization module allowing users to input multiple tickers and have the application calculate optimal weightings based on Markowitz's efficient frontier.

CHAPTER 8

CONCLUSION

1. Conclusion

Tradezy represents a successful and functional prototype for a data-driven investment analysis platform built entirely on the accessible Streamlit framework. The project achieved its primary objectives by seamlessly integrating foundational financial theory with modern computational analytics, thereby empowering users with quantitative insights.

Project Success Summary

1. **Seamless Integration:** The application successfully combined three distinct modules Stock Information, CAPM Analysis, and Stock Prediction within a single, intuitive user interface.
2. **Theoretical Rigor:** The CAPM Module provides a theoretically grounded benchmark for the required rate of return, anchored by a data-derived Beta (β). This allows investors to objectively assess an asset's systemic risk compensation.
3. **Predictive Utility:** The ARIMA-based Stock Prediction Module, utilizing the Box-Jenkins methodology, provides a structured, statistical forecast, including the crucial **Confidence Interval** to communicate the inherent uncertainty in market prediction.
4. **Accessibility and Transparency:** By leveraging Python and Streamlit, the project democratized access to tools often restricted to expensive institutional platforms, presenting complex analytics in a digestible, interactive format.

In essence, Tradezy provides a substantial analytical foundation for investors seeking to move beyond anecdotal evidence and toward disciplined, quantitative decision-making.

2. Limitations and Future Work

While effective as a proof-of-concept, Tradezy operates under several limitations inherent to the models and the scope of the prototype, which inform the roadmap for future development.

1. Model Limitations (Theoretical)

- CAPM Simplification: The CAPM is a single-factor model that assumes a perfect, friction-less market where investors are compensated only for systematic risk. It ignores other proven risk factors like size (Small minus Big, SMB) and value (High minus Low, HML), which are addressed by richer models like the Fama-French models.
- ARIMA Linearity: The ARIMA model is inherently a linear model. Financial time series, however, are highly non-linear, often exhibiting volatility clustering and sudden shifts that linear models struggle to capture accurately. This can lead to less reliable long-term forecasts.

2. Data and Scope Limitations (Practical)

- Data Latency: The application relies on a free, external API (like yfinance), which may not provide real-time data or have stringent rate limits, hindering high-frequency or real-time analysis.
- Prediction Window: The current ARIMA implementation is best suited for short-term (e.g., 30-day) forecasts. Extending the prediction window significantly degrades accuracy due to the increasing width of the confidence interval.
- Lack of Alpha/Residual Analysis: The current CAPM module stops at $E(R_i)$. A more advanced version should calculate the stock's historical Alpha (α)—the residual return not explained by the market—to better evaluate management performance.

3. Future Enhancements (Roadmap)

The identified limitations suggest clear pathways for the next iteration of Tradezy:

- Advanced Risk Modeling: Integrate the Fama-French 3-Factor or 5-Factor models to provide a more sophisticated required return calculation.
- Deep Learning Prediction: Replace or augment ARIMA with Deep Learning models like LSTM (Long Short-Term Memory) networks, which are better suited to model the non-linear, sequential nature of stock prices.
- Portfolio Optimization: Add a module based on Markowitz's Mean-Variance Optimization to allow users to input multiple tickers and find the optimal portfolio weights for maximum return at a given risk level.

CHAPTER 9

REFERENCES

- W. F. Sharpe, "Capital asset prices: A theory of market equilibrium under conditions of risk," *The Journal of Finance*, vol. 19, no. 3, pp. 425–442, Sep. 1964.

- G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*. San Francisco, CA, USA: Holden-Day, 1970.

- D. Dickey and W. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American Statistical Association*, vol. 74, no. 366a, pp. 427–431, 1979.

- R. C. Merton, "A intertemporal capital asset pricing model," *Econometrica*, vol. 41, no. 5, pp. 867–887, Sep. 1973.

- S. R. L. R. L. O. G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*, 5th ed. Hoboken, NJ, USA: John Wiley & Sons, 2015.

- W. McKinney, "Data structures for statistical computing in Python," in *Proc. 9th Python in Science Conf.*, 2010, pp. 56–61.

- S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, Mar. 2011.

- F. P. A. Chollet et al., *Keras*. [Online]. Available: https://keras.io. (If a deep learning component like LSTM were used for prediction).

- A. M. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001. (If Random Forests were considered for prediction).

- yfinance. (2025). *yfinance: Yahoo! Finance market data downloader*. [Online]. Available: https://pypi.org/project/yfinance/.

- J. D. Cryer and K.-S. Chan, *Time Series Analysis with Applications in R*, 2nd ed. New York, NY, USA: Springer, 2008.

- F. X. Diebold, *Elements of Forecasting*, 4th ed. Mason, OH, USA: Thomson South-Western, 2007.

APPENDIX A

SAMPLE CODE

Trading_App.py

```
import streamlit as st

st.set_page_config(
    page_title="Tradezy",
    page_icon="chart_with_upwards_trend:",
    layout="wide"
)

st.title("Tradezy: Trading made EZ")

st.header("Empowering Your Investments with Data-Driven Insights")

st.image("bnb.png")

st.markdown("## Tradezy provides the following services:")

st.markdown("## :one: Stock Information")
st.write("View detailed insights and updates on your favorite stocks
here.")

st.markdown("## :two: Stock Prediction")
st.write("Access detailed company profiles, including market cap, P/E
ratio, and sector-wise performance analysis.")

st.markdown("## :three: CAPM Return")
st.write("Work with the Capital Asset Pricing Model (CAPM), which provides
a relation between market risk and expected returns.")

st.markdown("## :four: CAPM Beta")
st.write("Here you can calculate the Beta of an equity asset. Beta
measures how much your asset returns move compared to the overall
market.")
```

CAPM_Func.py

```
import plotly.express as px
import numpy as np

#function to plaot interactive chart

def interactive_plot(df):
    fig=px.line()
    for i in df.columns[1:]:
        fig.add_scatter(x=df['Date'], y=df[i], name=i)
    fig.update_layout(width=450, margin=dict(l=20, r=20, t=50, b=20),
legend=dict(orientation='h', yanchor='top', y=1.02, xanchor='right', x=1))
    return fig

# Normalization function to normalize the price of the stocks based on
initial price
def normalize(df_2):
    df=df_2.copy()
    for i in df.columns[1:]:
        df[i]=df[i]/df[i][0]
    return df

# Functions to create daily returns
def daily_return(df):
    df_daily_return=df.copy()
    for i in df_daily_return.columns[1:]:
        for j in range(1,len(df)):
            df_daily_return[i][j]=((df_daily_return[i][j]-
df_daily_return[i][j-1])/df_daily_return[i][j-1])*100
        df_daily_return[i][0]=0
    return df_daily_return

# function to calculate beta
def calulate_beta(stock_daily_return, stock):
    rm=stock_daily_return['sp500'].mean()*252
    b,a=np.polyfit(stock_daily_return['sp500'],
stock_daily_return[stock],1)
    return b,a
```

Model_train.py

```
import yfinance as yf
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.tsa.arima.model import ARIMA
import numpy as np
from sklearn.preprocessing import StandardScaler
from datetime import datetime, timedelta
import pandas as pd

def get_data(ticker):
    stock_data = yf.download(ticker, start='2024-01-01')
    return stock_data[['Close']]

def stationary_check(close_price):
    adf_test = adfuller(close_price)
    p_value = round(adf_test[1], 3)
    return p_value

def get_rolling_mean(close_price):
    rolling_price = close_price.rolling(window=7).mean().dropna()
    return rolling_price

def get_differencing_order(close_price):
    p_value = stationary_check(close_price)
    d = 0
    while True:
        if p_value > 0.05:
            d += 1
            close_price = close_price.diff().dropna()
            p_value = stationary_check(close_price)
        else:
            break
    return d

def fit_model(data, differencing_order):
    model = ARIMA(data, order=(30, differencing_order, 30))
    model_fit = model.fit()
```

```
    forecast_steps = 30
    forecast = model_fit.get_forecast(steps=forecast_steps)

    predictions = forecast.predicted_mean
    return predictions

def evaluate_model(original_price, differencing_order):
    train_data, test_data = original_price[:-30], original_price[-30:]
    predictions = fit_model(train_data, differencing_order)
    rmse = np.sqrt(mean_squared_error(test_data, predictions))
    return round(rmse, 2)

def scaling(close_price):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(np.array(close_price).reshape(-1,
1))
    return scaled_data, scaler

def get_forecast(original_price, differencing_order):
    predictions = fit_model(original_price, differencing_order)
    start_date = datetime.now().strftime('%Y-%m-%d')
    end_date = (datetime.now() + timedelta(days=29)).strftime('%Y-%m-%d')
    forecast_index = pd.date_range(start=start_date, end=end_date,
freq='D')
    forecast_df = pd.DataFrame(predictions, index=forecast_index,
columns=['Close'])
    return forecast_df

def inverse_scaling(scaler, scaled_data):
    close_price=scaler.inverse_transform(np.array(scaled_data).reshape(-
1,1))
    return close_price
```

Plotly_figure.py

```
import dateutil.relativedelta
import plotly.graph_objects as go
import ta
from ta.trend import SMAIndicator, EMAIndicator, MACD
from ta.momentum import RSIIndicator
from ta.volatility import BollingerBands
import pandas as pd
import yfinance as yf
import datetime
import dateutil

# Plotly Table Function
def plotly_table(dataframe):
    headerColor = '#1f2c56'
    rowEvenColor = '#2e3b5f'
    rowOddColor = '#1c253b'

    fig = go.Figure(data=[go.Table(
        header=dict(
            values=["<b>Index</b>"] + ["<b>"+str(i)+"</b>" for i in
dataframe.columns],
            line_color=headerColor, fill_color='#3f4c6b',
            align='center', font=dict(color='white', size=15), height=35
        ),
        cells=dict(
            values=[[f"<b>{i}</b>" for i in dataframe.index]] +
[dataframe[col] for col in dataframe.columns],
            fill_color=[rowOddColor if i % 2 == 0 else rowEvenColor for i
in range(len(dataframe))],
            align='left', line_color='white',
            font=dict(color="white", size=14)
        )
    )])

    fig.update_layout(height=600, margin=dict(l=0, r=0, t=0, b=0))
    return fig

def filter_data(dataframe, num_period):
```

```
    if num_period == '1mo':
        date = dataframe.index[-1] +
dateutil.relativedelta.relativedelta(months=-1)
    elif num_period == '5d':
        date = dataframe.index[-1] +
dateutil.relativedelta.relativedelta(days=-5)
    elif num_period == '6mo':
        date = dataframe.index[-1] +
dateutil.relativedelta.relativedelta(months=-6)
    elif num_period == '1y':
        date = dataframe.index[-1] +
dateutil.relativedelta.relativedelta(years=-1)
    elif num_period == '5y':
        date = dataframe.index[-1] +
dateutil.relativedelta.relativedelta(years=-5)
    elif num_period == 'ytd':
        date = datetime.datetime(dataframe.index[-1].year, 1,
1).strftime('%Y-%m-%d')
    else:
        date = dataframe.index[0]

    return dataframe.reset_index()[dataframe.reset_index()['Date'] > date]

def close_chart(dataframe, num_period=False):
    if num_period:
        dataframe = filter_data(dataframe, num_period)

    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['Open'],
        mode='lines',
        name='Open',
        line=dict(width=2, color='#5ab7ff')
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['Close'],
        mode='lines',
        name='Close',
        line=dict(width=2, color='black')
```

```
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['High'],
        mode='lines',
        name='High',
        line=dict(width=2, color='#0078ff')
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['Low'],
        mode='lines',
        name='Low',
        line=dict(width=2, color='red')
    ))

    fig.update_xaxes(rangeslider_visible=True)
    fig.update_layout(
        height=500,
        margin=dict(l=0, r=20, t=20, b=0),
        plot_bgcolor='black',
        paper_bgcolor='#e1efff',
        legend=dict(
            orientation="h",
            yanchor="top",
            y=1.02,
            xanchor="right",
            x=1,
            font=dict(
                size=14,
                color="black"
            )
        ),
        xaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        ),
        yaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
```

```
            tickfont=dict(color='black')
        )
    )

    return fig

def candlestick(dataframe, num_period):
    dataframe = filter_data(dataframe, num_period)
    fig = go.Figure()

    fig.add_trace(go.Candlestick(
        x=dataframe['Date'],
        open=dataframe['Open'],
        high=dataframe['High'],
        low=dataframe['Low'],
        close=dataframe['Close']
    ))

    fig.update_layout(
        showlegend=False,
        height=500,
        margin=dict(l=0, r=20, t=20, b=0),
        plot_bgcolor='black',
        paper_bgcolor='#e1efff',
        xaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        ),
        yaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        )
    )
    return fig

def RSI(dataframe, num_period):
    if num_period:
        dataframe = filter_data(dataframe, num_period)
```

```
    rsi = RSIIndicator(close=dataframe['Close'], window=14)
    dataframe['RSI'] = rsi.rsi()

    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['RSI'],
        name='RSI',
        mode='lines',
        marker_color='orange',
        line=dict(width=2, color='orange')
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=[70]*len(dataframe),
        name='Overbought',
        mode='lines',
        marker_color='red',
        line=dict(width=2, color='red', dash='dash')
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=[30]*len(dataframe),
        name='Oversold',
        mode='lines',
        marker_color='#79da84',
        line=dict(width=2, color='#79da84', dash='dash')
    ))

    fig.update_layout(
        yaxis_range=[0, 100],
        height=200,
        plot_bgcolor='black',
        paper_bgcolor='#e1efff',
        margin=dict(l=0, r=0, t=0, b=0),
        legend=dict(
            orientation="h",
            yanchor="top",
            y=1.02,
            xanchor="right",
            x=1,
```

```
            font=dict(
                color='black'
            )
        ),
        xaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        ),
        yaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        )
    )
    return fig

def Moving_average(dataframe, num_period):
    sma = SMAIndicator(close=dataframe['Close'], window=50)
    dataframe['SMA_50'] = sma.sma_indicator()

    dataframe = filter_data(dataframe, num_period)

    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['Open'],
        mode='lines', name='Open',
        line=dict(width=2, color='#5ab7ff')
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['Close'],
        mode='lines', name='Close',
        line=dict(width=2, color='black')
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['High'],
        mode='lines', name='High',
        line=dict(width=2, color='#0078ff')
```

```
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['Low'],
        mode='lines', name='Low',
        line=dict(width=2, color='red')
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'], y=dataframe['SMA_50'],
        mode='lines', name='SMA 50',
        line=dict(width=2, color='purple')
    ))

    fig.update_xaxes(rangeslider_visible=True)
    fig.update_layout(
        height=500,
        margin=dict(l=0, r=20, t=20, b=0),
        plot_bgcolor='black',
        paper_bgcolor='#e1efff',
        legend=dict(
            orientation="h",
            yanchor="top",
            y=1.02,
            xanchor="right",
            x=1,
            font=dict(
                color='black'
            )
        ),
        xaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        ),
        yaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        )
    )
```

```
    return fig

def plot_MACD(dataframe, num_period):
    macd = MACD(close=dataframe['Close'])
    dataframe['MACD'] = macd.macd()
    dataframe['MACD Signal'] = macd.macd_signal()
    dataframe['MACD Hist'] = macd.macd_diff()

    dataframe = filter_data(dataframe, num_period)

    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=dataframe['Date'],
        y=dataframe['MACD'],
        name='MACD',
        mode='lines',
        line=dict(width=2, color='orange')
    ))

    fig.add_trace(go.Scatter(
        x=dataframe['Date'],
        y=dataframe['MACD Signal'],
        name='Signal',
        mode='lines',
        line=dict(width=2, color='red', dash='dash')
    ))

    bar_colors = ['green' if val >= 0 else 'red' for val in
dataframe['MACD Hist']]
    fig.add_trace(go.Bar(
        x=dataframe['Date'],
        y=dataframe['MACD Hist'],
        name='Histogram',
        marker_color=bar_colors
    ))

    fig.update_layout(
        height=200,
        plot_bgcolor='black',
        paper_bgcolor='#e1efff',
```

```
        margin=dict(l=0, r=0, t=0, b=0),
        legend=dict(
            orientation="h",
            yanchor="top",
            y=1.02,
            xanchor="right",
            x=1,
            font=dict(
                color='black'
            )
        ),
        xaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        ),
        yaxis=dict(
            showgrid=True,
            gridcolor='lightgray',
            tickfont=dict(color='black')
        )
    )
    return fig

import plotly.graph_objects as go

def Moving_average_forecast(forecast):
    fig = go.Figure()

    # Actual Close Price
    fig.add_trace(go.Scatter(
        x=forecast.index[:-30],
        y=forecast['Close'].iloc[:-30],
        mode='lines',
        name='Close Price',
        line=dict(width=2, color='black')
    ))

    # Forecasted Close Price
    fig.add_trace(go.Scatter(
        x=forecast.index[-31:],
```

```
        y=forecast['Close'].iloc[-31:],
        mode='lines',
        name='Future Close Price',
        line=dict(width=2, color='red')
    ))

    # Update axes with scales
    fig.update_xaxes(
        title_text="Date",
        showgrid=True,
        showline=True,
        ticks="outside",
        showticklabels=True,
        linecolor='black',
        tickfont=dict(color='black'),
        title_font=dict(color='black')
    )
    fig.update_yaxes(
        title_text="Close Price",
        showgrid=True,
        showline=True,
        ticks="outside",
        showticklabels=True,
        linecolor='black',
        tickfont=dict(color='black'),
        title_font=dict(color='black')
    )

    # Layout customization
    fig.update_layout(
        height=500,
        margin=dict(l=0, r=20, t=20, b=0),
        plot_bgcolor='black',
        paper_bgcolor='#e1efff',
        legend=dict(
            yanchor="top",
            xanchor="right",
            font=dict(color='black')  # Legend font color
        ),
        xaxis_rangeslider_visible=True
    )
```

```
    return fig
```

CAPM_Beta.py

```
import streamlit as st
import pandas as pd
import yfinance as yf
import datetime
import pandas_datareader.data as web
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression
import numpy as np

# ---------- Page Config ----------
st.set_page_config(
    page_title="Beta and Return Calculator",
    page_icon="📈",
    layout="wide"
)

st.markdown("## Calculate Beta and Return for Individual Stock")

# ---------- User Inputs ----------
col1, col2 = st.columns([1, 1])
with col1:
    stock_ticker = st.text_input("Choose a stock", value="TSLA").upper()

with col2:
    years = st.number_input("Number of Years", min_value=1, max_value=10,
value=1)

# ---------- Date Range ----------
end = datetime.date.today()
start = datetime.date(end.year - years, end.month, end.day)

# ---------- Data Download ----------
@st.cache_data
```

```
def download_data(ticker, start_date, end_date):
    try:
        # Download stock data
        stock_data = yf.download(ticker, start=start_date, end=end_date)
        stock_data = stock_data[['Close']].rename(columns={'Close':
'Stock'})

        # Download S&P 500 data
        sp500_data = web.DataReader('SP500', 'fred', start_date, end_date)
        sp500_data = sp500_data.rename(columns={'SP500': 'SP500'})

        return stock_data, sp500_data
    except Exception as e:
        st.error(f"Error downloading data: {e}")
        return None, None

stock_data, sp500_data = download_data(stock_ticker, start, end)

if stock_data is None or sp500_data is None:
    st.stop()

# ---------- Data Alignment ----------
def align_data(stock_df, sp500_df):
    # Convert to DataFrames if they're Series
    if isinstance(stock_df, pd.Series):
        stock_df = stock_df.to_frame('Stock')
    if isinstance(sp500_df, pd.Series):
        sp500_df = sp500_df.to_frame('SP500')

    # Ensure we have proper column names
    stock_df.columns = ['Stock']
    sp500_df.columns = ['SP500']

    # Convert indices to datetime and normalize
    stock_df.index = pd.to_datetime(stock_df.index).normalize()
    sp500_df.index = pd.to_datetime(sp500_df.index).normalize()

    # Merge using index alignment
    combined = pd.concat([stock_df, sp500_df], axis=1).dropna()

    return combined
```

```
df = align_data(stock_data, sp500_data)

if df.empty:
    st.error("No overlapping data found between stock and S&P 500")
    st.stop()

# ---------- Calculate Returns ----------
df['stock_return'] = df['Stock'].pct_change() * 100
df['sp500_return'] = df['SP500'].pct_change() * 100
df = df.dropna()

# ---------- Calculate Beta ----------
X = df['sp500_return'].values.reshape(-1, 1)
y = df['stock_return'].values.reshape(-1, 1)

model = LinearRegression()
model.fit(X, y)
beta = float(model.coef_[0])
alpha = float(model.intercept_[0])

# ---------- Calculate Expected Return ----------
rf = 0  # risk-free rate
rm = df['sp500_return'].mean() * 252  # annualized market return
expected_return = rf + beta * (rm - rf)

# ---------- Display Results ----------
col1, col2 = st.columns(2)
with col1:
    st.metric("Beta", value=f"{beta:.4f}")

with col2:
    st.metric("Expected Annual Return", value=f"{expected_return:.2f}%")

# ---------- Create Scatter Plot ----------
fig = go.Figure()

# Scatter plot of returns
fig.add_trace(go.Scatter(
    x=df['sp500_return'],
    y=df['stock_return'],
```

```
    mode='markers',
    marker=dict(color='lightblue', size=8),
    name='Daily Returns'
))

# Regression line
x_range = np.linspace(df['sp500_return'].min(), df['sp500_return'].max(),
100)
y_range = beta * x_range + alpha
fig.add_trace(go.Scatter(
    x=x_range,
    y=y_range,
    mode='lines',
    line=dict(color='red', width=2),
    name=f'Regression Line (β={beta:.2f})'
))

# Layout configuration with black background and visible scales
fig.update_layout(
    title=f'{stock_ticker} vs S&P 500 Daily Returns',
    xaxis_title='S&P 500 Daily Return (%)',
    yaxis_title=f'{stock_ticker} Daily Return (%)',
    plot_bgcolor='black',
    paper_bgcolor='black',
    font=dict(color='white'),
    xaxis=dict(
        showline=True,
        linecolor='white',
        gridcolor='rgba(255,255,255,0.1)',
        zerolinecolor='rgba(255,255,255,0.5)',
        tickfont=dict(color='white')
    ),
    yaxis=dict(
        showline=True,
        linecolor='white',
        gridcolor='rgba(255,255,255,0.1)',
        zerolinecolor='rgba(255,255,255,0.5)',
        tickfont=dict(color='white')
    ),
    legend=dict(
        orientation="h",
```

```
        yanchor="bottom",
        y=1.02,
        xanchor="right",
        x=1,
        font=dict(color='white')
    ),
    hovermode='closest'
)

# Add scale/axis information
fig.add_annotation(
    x=0.05,
    y=0.95,
    xref="paper",
    yref="paper",
    text=f"Scale: X-Axis (S&P 500) ±{df['sp500_return'].abs().max():.1f}%,
Y-Axis ({stock_ticker}) ±{df['stock_return'].abs().max():.1f}%",
    showarrow=False,
    font=dict(color='white', size=12),
    bgcolor='rgba(0,0,0,0.7)',
    bordercolor='white'
)

# Display the plot
st.plotly_chart(fig, use_container_width=True)
```

CAPM_Return.py

```
import streamlit as st
import pandas as pd
import yfinance as yf
import datetime
import pandas_datareader.data as web
from pages.utils.CAPM_func import interactive_plot
from pages.utils.CAPM_func import normalize
from pages.utils.CAPM_func import daily_return
from pages.utils.CAPM_func import calulate_beta

st.set_page_config(
    page_title="CAPM",
    page_icon="chart_with_downwards_trend",
    layout="wide"
)
st.title("Capital Asset Pricing Model")

# User Input
col1, col2 = st.columns([1, 1])
with col1:
    stocks_list = st.multiselect(
        "Choose 4 stocks",
        ('TSLA', 'AAPL', 'MGM', 'MSFT', 'NFLX', 'AMZN', 'NVDA', 'GOOGL'),
        ['TSLA', 'AAPL', 'NFLX', 'GOOGL']
    )
with col2:
    year = st.number_input("Number of years", 1, 10)

# Date range
try:
    end = datetime.date.today()
    start = datetime.date(end.year - year, end.month, end.day)

    # Download SP500 Data
    SP500 = web.DataReader('SP500', 'fred', start, end)
    SP500 = SP500.reset_index()
    SP500.columns = ['Date', 'sp500']
    SP500['Date'] = pd.to_datetime(SP500['Date'])
```

```
    # Download stock data and combine
    stock_data_list = []

    for stock in stocks_list:
        df = yf.download(stock, start=start, end=end)[['Close']]
        df = df.rename(columns={'Close': stock})
        stock_data_list.append(df)

    # Combine stock close prices on the index (which is Date)
    combined_stocks = pd.concat(stock_data_list, axis=1, join='inner')

    # Reset index and flatten completely
    combined_stocks = combined_stocks.reset_index()  # Makes 'Date' a
column
    combined_stocks['Date'] = pd.to_datetime(combined_stocks['Date'])

    # Ensure no MultiIndex remains
    combined_stocks.columns = [col if isinstance(col, str) else col[0] for
col in combined_stocks.columns]

    # Merge with SP500 data
    merged_df = pd.merge(combined_stocks, SP500, on='Date', how='inner')

    # Display result
    col1, col2 = st.columns([1, 1])
    with col1:
        st.markdown("### Dataframe Head")
        st.dataframe(merged_df.head(), use_container_width=True)
    with col2:
        st.markdown("### Dataframe Tail")
        st.dataframe(merged_df.tail(), use_container_width=True)

    col1, col2=st.columns([1,1])
    with col1:
        st.markdown("### Price of all the Stocks")
        st.plotly_chart(interactive_plot(merged_df))
    with col2:
        st.markdown("### Price of Stocks (After Normalizing)")
        st.plotly_chart(interactive_plot(normalize(merged_df)))
```

```
    stock_daily_return=daily_return(merged_df)
    print(stock_daily_return.head())

    beta={}
    alpha={}

    for i in stock_daily_return.columns:
        if i!='Date' and i!='SP500':
            b,a = calulate_beta(stock_daily_return,i)
            beta[i]=b
            alpha[i]=a
    print(beta,alpha)

    beta_df=pd.DataFrame(columns=['Stock', 'Beta Value'])
    beta_df['Stock']=beta.keys()
    beta_df['Beta Value']=[str(round(i,2)) for i in beta.values()]

    with col1:
        st.markdown("### Calculated Beta Value")
        st.dataframe(beta_df, use_container_width=True)

    rf = 0
    rm = stock_daily_return['sp500'].mean() * 252
    return_df = pd.DataFrame()
    return_value = []

    for stock, value in beta.items():
        return_value.append(str(round(rf + (value * (rm - rf)), 2)))

    return_df['Stock'] = list(beta.keys())  # Match exactly with
return_value
    return_df['Return Value'] = return_value


    with col2:
        st.markdown("### Calculated return using CAPM")
        st.dataframe(return_df, use_container_width=True)

except:
    st.write("Please Select Valid Tickers")
```

Stock_Analysis.py

```
import streamlit as st
import pandas as pd
import yfinance as yf
import plotly.graph_objects as go
import datetime
import ta
from pages.utils.plotly_figure import plotly_table
from pages.utils.plotly_figure import filter_data
from pages.utils.plotly_figure import close_chart
from pages.utils.plotly_figure import candlestick
from pages.utils.plotly_figure import RSI
from pages.utils.plotly_figure import Moving_average
from pages.utils.plotly_figure import plot_MACD

st.set_page_config(
    page_title="Stock Analysis",
    page_icon="📄",
    layout="wide",
)

# Extremely aggressive CSS to reduce all spacing
st.markdown("""
    <style>
    .block-container {
        padding-top: 0.5rem !important;
        padding-bottom: 0rem !important;
    }
    .element-container {
        margin-bottom: 0rem !important;
        padding-bottom: 0rem !important;
    }
    div[data-testid="metric-container"] {
        background-color: rgba(28, 131, 225, 0.1);
        border: 1px solid rgba(28, 131, 225, 0.1);
        padding: 5px !important;
        border-radius: 5px;
        margin: 0 !important;
    }
```

```
    div[data-testid="metric-container"] > div {
        padding: 0 !important;
        margin: 0 !important;
    }
    div[data-testid="stMetricValue"] {
        padding: 0 !important;
        margin: 0 !important;
    }
    div[data-testid="stMetricDelta"] {
        padding: 0 !important;
        margin: 0 !important;
    }
    .stPlotlyChart {
        margin: 0 !important;
        padding: 0 !important;
    }
    h1, h2, h3, h4, h5 {
        margin-top: 0.2rem !important;
        margin-bottom: 0.1rem !important;
    }
    .stDataFrame {
        margin-top: 0.1rem !important;
        margin-bottom: 0.1rem !important;
    }
    .css-1y4p8pa {
        padding-top: 0.5rem !important;
        padding-bottom: 0rem !important;
    }
    .css-1v0mbdj {
        margin-top: 0.1rem !important;
        margin-bottom: 0.1rem !important;
    }
    .stMarkdown {
        margin-top: 0.1rem !important;
        margin-bottom: 0.1rem !important;
    }
    </style>
""", unsafe_allow_html=True)

st.title("📈 Stock Analysis")
```

```
available_tickers = [
    "AAPL", "MSFT", "GOOGL", "TSLA", "AMZN", "NVDA", "META", "NFLX", "BRK-
B", "JPM"
]

col1, col2, col3 = st.columns(3)
today = datetime.date.today()

with col1:
    ticker = st.selectbox("Select Stock Ticker",
options=available_tickers, index=3)
with col2:
    start_date = st.date_input("Choose Start Date",
datetime.date(today.year - 1, today.month, today.day))
with col3:
    end_date = st.date_input("Choose End Date", today)

st.subheader(ticker)

# Load stock
stock = yf.Ticker(ticker)
info = stock.info

# Business summary with minimal spacing
st.write(info.get('longBusinessSummary', 'No business summary
available.'))
cols = st.columns(3)
with cols[0]:
    st.write("**Sector:**", info.get('sector', 'N/A'))
with cols[1]:
    st.write("**Full Time Employees:**", info.get('fullTimeEmployees',
'N/A'))
with cols[2]:
    st.write("**Know More:**", info.get('website', 'N/A'))

# Financial metrics in a single row with minimal spacing
metrics_row = st.columns([1, 1, 0.5, 1, 1])  # Added a small spacer column

with metrics_row[0]:
    df1 = pd.DataFrame({
        'Metric': ['Beta', 'EPS', 'PE Ratio'],
```

```
        'Value': [
            info.get('beta', 'N/A'),
            info.get('trailingEps', 'N/A'),
            info.get('trailingPE', 'N/A')
        ]
    })
    st.dataframe(df1.set_index('Metric'), height=150)

with metrics_row[1]:
    df2 = pd.DataFrame({
        'Metric': ['Current Ratio', 'Revenue Per Share', 'Debt to
Equity'],
        'Value': [
            info.get("currentRatio", 'N/A'),
            info.get("revenuePerShare", 'N/A'),
            info.get("debtToEquity", 'N/A')
        ]
    })
    st.dataframe(df2.set_index('Metric'), height=150)

# Daily close and last 10 days data in a tight layout
try:
    data = yf.download(ticker, start=start_date, end=end_date)
    if isinstance(data.columns, pd.MultiIndex):
        data.columns = data.columns.get_level_values(0)

    if len(data) < 2:
        st.warning("Not enough historical data available to show daily
change.")
    else:
        # Extremely tight layout for daily close and table
        daily_col, table_col = st.columns([1, 3])

        with daily_col:
            latest_close = float(data['Close'].iloc[-1])
            prev_close = float(data['Close'].iloc[-2])
            daily_change = latest_close - prev_close
            st.metric("Daily Close", f"{latest_close:.2f}",
f"{daily_change:.2f}")
            st.write("")  # Minimal spacer
```

```
        with table_col:
            st.markdown("**Data of the Last 10 Days**")
            last_10_df =
data.tail(10).sort_index(ascending=False).round(3)
            st.dataframe(last_10_df)

except Exception as e:
    st.error(f"Error loading stock data: {e}")

# Time period buttons - single line with minimal spacing
period_cols = st.columns([1,1,1,1,1,1,1,1,1,1,1,1])
num_period = ''
with period_cols[0]:
    if st.button('5D', key='5d'):
        num_period = '5D'
with period_cols[1]:
    if st.button('1M', key='1m'):
        num_period = '1mo'
with period_cols[2]:
    if st.button('6M', key='6m'):
        num_period = '6mo'
with period_cols[3]:
    if st.button('YTD', key='ytd'):
        num_period = 'ytd'
with period_cols[4]:
    if st.button('1Y', key='1y'):
        num_period = '1y'
with period_cols[5]:
    if st.button('5Y', key='5y'):
        num_period = '5y'
with period_cols[6]:
    if st.button('MAX', key='max'):
        num_period = 'max'

# Chart selection with minimal spacing
chart_cols = st.columns([1, 1, 4])
with chart_cols[0]:
    chart_type = st.selectbox('Chart Type', ['Line', 'Candle'])
with chart_cols[1]:
    if chart_type == 'Candle':
        indicator = st.selectbox('Indicator', ['RSI', 'MACD'])
```

```
    else:
        indicator = st.selectbox('Indicator', ['RSI', 'Moving Average',
'MACD'])

# Charts with minimal spacing
st.markdown("### Main Chart")
data_used = stock.history(period=num_period if num_period else '1y')
period_used = num_period if num_period else '1y'

if chart_type == 'Candle':
    st.plotly_chart(candlestick(data_used, period_used),
use_container_width=True)
else:
    st.plotly_chart(close_chart(data_used, period_used),
use_container_width=True)

st.markdown("### Indicator Chart")
if indicator == 'RSI':
    st.plotly_chart(RSI(data_used, period_used), use_container_width=True)
elif indicator == 'Moving Average':
    st.plotly_chart(Moving_average(data_used, period_used),
use_container_width=True)
elif indicator == 'MACD':
    st.plotly_chart(plot_MACD(data_used, period_used),
use_container_width=True)
```

Stock_Prediction.py

```
import streamlit as st
import pandas as pd
from pages.utils.model_train import get_data
from pages.utils.model_train import stationary_check
from pages.utils.model_train import get_rolling_mean
from pages.utils.model_train import get_differencing_order
from pages.utils.model_train import fit_model
from pages.utils.model_train import evaluate_model
from pages.utils.model_train import scaling
from pages.utils.model_train import get_forecast
from pages.utils.model_train import inverse_scaling
from pages.utils.plotly_figure import plotly_table
from pages.utils.plotly_figure import Moving_average_forecast

st.set_page_config(
    page_title="Stock Prediction",
    page_icon="chart_with_downwards_trend",
    layout="wide",
)

st.title("Stock Prediction")
col1, col2, col3 = st.columns(3)

with col1:
    ticker=st.text_input("Enter the Stock Ticker", 'TSLA')
rmse=0

st.subheader("Predicting the Close Price over the next 30 days
for:"+ticker)

close_price=get_data(ticker)
rolling_price=get_rolling_mean(close_price)

differencing_order = get_differencing_order(rolling_price)
scaled_data, scaler = scaling(rolling_price)
rmse = evaluate_model(scaled_data, differencing_order)

st.write("**Model RMSE Score:**", rmse)
```

```
forecast = get_forecast(scaled_data, differencing_order)

forecast['Close'] = inverse_scaling(scaler, forecast['Close'])
st.write('### Forecast Data (Next 30 days)')
fig_tail = plotly_table(forecast.sort_index(ascending=True).round(3))
fig_tail.update_layout(height=220)
st.plotly_chart(fig_tail, use_container_width=True)

forecast = pd.concat([rolling_price, forecast])

st.plotly_chart(Moving_average_forecast(forecast.iloc[100:]),
use_container_width=True)
```

Requirements.txt

```
aiohappyeyeballs==2.6.1
aiohttp==3.12.13
aiosignal==1.4.0
altair==5.5.0
annotated-types==0.7.0
anyio==4.9.0
appnope==0.1.4
asttokens==3.0.0
attrs==25.3.0
backoff==2.2.1
bcrypt==4.3.0
beautifulsoup4==4.13.4
blinker==1.9.0
build==1.2.2.post1
cachetools==5.5.2
certifi==2025.6.15
cffi==1.17.1
charset-normalizer==3.4.2
chromadb==1.0.15
click==8.2.1
coloredlogs==15.0.1
```

```
comm==0.2.2
curl_cffi==0.12.0
dataclasses-json==0.6.7
debugpy==1.8.14
decorator==5.2.1
distro==1.9.0
durationpy==0.10
executing==2.2.0
filelock==3.18.0
flatbuffers==25.2.10
frozendict==2.4.6
frozenlist==1.7.0
fsspec==2025.5.1
gitdb==4.0.12
GitPython==3.1.44
google-auth==2.40.3
googleapis-common-protos==1.70.0
groq==0.29.0
grpcio==1.73.1
h11==0.16.0
hf-xet==1.1.5
httpcore==1.0.9
httptools==0.6.4
httpx==0.28.1
httpx-sse==0.4.1
huggingface-hub==0.33.2
humanfriendly==10.0
idna==3.10
importlib_metadata==8.7.0
importlib_resources==6.5.2
ipykernel==6.29.5
ipython==9.4.0
ipython_pygments_lexers==1.1.1
jedi==0.19.2
Jinja2==3.1.6
joblib==1.5.1
jsonpatch==1.33
jsonpointer==3.0.0
jsonschema==4.24.0
jsonschema-specifications==2025.4.1
jupyter_client==8.6.3
```

```
jupyter_core==5.8.1
kubernetes==33.1.0
langchain==0.3.26
langchain-community==0.3.27
langchain-core==0.3.68
langchain-groq==0.3.5
langchain-text-splitters==0.3.8
langsmith==0.4.4
lxml==6.0.0
markdown-it-py==3.0.0
MarkupSafe==3.0.2
marshmallow==3.26.1
matplotlib-inline==0.1.7
mdurl==0.1.2
mmh3==5.1.0
mpmath==1.3.0
multidict==6.6.3
multitasking==0.0.11
mypy_extensions==1.1.0
narwhals==1.46.0
nest-asyncio==1.6.0
numpy>=2.2.0,<2.3
oauthlib==3.3.1
onnxruntime==1.22.0
opentelemetry-api==1.34.1
opentelemetry-exporter-otlp-proto-common==1.34.1
opentelemetry-exporter-otlp-proto-grpc==1.34.1
opentelemetry-proto==1.34.1
opentelemetry-sdk==1.34.1
opentelemetry-semantic-conventions==0.55b1
orjson==3.10.18
overrides==7.7.0
packaging==24.2
pandas>=2.0
pandas-datareader>=0.10.0
pandas_ta>=0.4.70
parso==0.8.4
patsy==1.0.1
peewee==3.18.2
pexpect==4.9.0
pillow==11.3.0
```

```
platformdirs==4.3.8
plotly==6.2.0
posthog==5.4.0
prompt_toolkit==3.0.51
propcache==0.3.2
protobuf==5.29.5
psutil==7.0.0
ptyprocess==0.7.0
pure_eval==0.2.3
pyarrow==20.0.0
pyasn1==0.6.1
pyasn1_modules==0.4.2
pybase64==1.4.1
pycparser==2.22
pydantic==2.11.7
pydantic-settings==2.10.1
pydantic_core==2.33.2
pydeck==0.9.1
Pygments==2.19.2
PyPika==0.48.9
pyproject_hooks==1.2.0
python-dateutil==2.9.0.post0
python-dotenv==1.1.1
pytz==2025.2
PyYAML==6.0.2
pyzmq==27.0.0
referencing==0.36.2
requests==2.32.4
requests-oauthlib==2.0.0
requests-toolbelt==1.0.0
rich==14.0.0
rpds-py==0.26.0
rsa==4.9.1
scikit-learn==1.7.0
scipy==1.16.0
setuptools==80.9.0
shellingham==1.5.4
six==1.17.0
smmap==5.0.2
sniffio==1.3.1
soupsieve==2.7
```

```
SQLAlchemy==2.0.41
stack-data==0.6.3
statsmodels==0.14.5
streamlit==1.46.1
sympy==1.14.0
ta==0.11.0
tenacity==9.1.2
threadpoolctl==3.6.0
tokenizers==0.21.2
toml==0.10.2
tornado==6.5.1
tqdm==4.67.1
traitlets==5.14.3
typer==0.16.0
typing-inspect==0.9.0
typing-inspection==0.4.1
typing_extensions==4.14.1
tzdata==2025.2
urllib3==2.5.0
uvicorn==0.35.0
uvloop==0.21.0
watchfiles==1.1.0
wcwidth==0.2.13
websocket-client==1.8.0
websockets==15.0.1
yarl==1.20.1
yfinance==0.2.65
zipp==3.23.0
zstandard==0.23.0
```