**PROJECT SYNOPSIS**

**Project Title: STOCK PRICE ANALYSIS AND PREDICTION**

**Introduction:**   
In today's financial markets, the ability to predict stock price movements is highly valuable. This project focuses on analyzing and forecasting stock prices for major tech companies, including Apple (AAPL), Google (GOOG), Microsoft (MSFT), and Amazon (AMZN). Utilizing historical stock data, the project employs various statistical and machine learning techniques such as moving averages, daily return analysis, and an LSTM (Long Short-Term Memory) model to predict future stock prices. The project also explores relationships between stocks through correlation analysis and evaluates the risk-return profile of each stock, providing valuable insights for investors.

**Objectives:**

1. To perform exploratory data analysis (EDA) on historical stock prices of AAPL, GOOG, MSFT, and AMZN.
2. To calculate moving averages and analyze daily returns for each stock.
3. To visualize stock price trends and identify patterns using advanced plotting techniques.
4. To build an LSTM-based model for predicting future stock prices based on past performance.
5. To assess the performance of the LSTM model using RMSE and compare predictions with actual stock prices.
6. To evaluate the correlation between the stock prices of the selected companies.
7. To analyze the risk-return tradeoff for the selected stocks using statistical techniques.

**Scope of work:**

**Data Exploration**

* Understanding the stock data, including features such as open, high, low, close, adjusted close, and volume.
* Examining the behavior of key metrics like closing prices and daily returns over time.

**Data Preprocessing**

* Cleaning the dataset by handling missing values using forward fill and removing duplicate entries.
* Scaling stock prices with Min-Max normalization to prepare the data for analysis and modeling.
* Calculating moving averages (10, 20, 50 days) and daily returns to enhance the dataset.

**Feature Selection**

* Identifying and focusing on critical metrics like closing prices, moving averages, and daily returns to predict stock trends effectively.

**Data Visualization**

* Plotting closing prices and moving averages to analyze long-term trends and fluctuations.
* Visualizing daily returns to understand the volatility of stocks.
* Using joint plots, pair plots, and heatmaps to explore correlations and relationships between different stocks.

**Model Building**

* Implementing and evaluating an LSTM model for stock price forecasting.
* Training the model on historical stock data and tuning it to minimize forecasting errors.

**Interpretation of Results**

* Analyzing the LSTM model’s predictions to assess performance and forecasting accuracy.
* Interpreting visual patterns and correlations to gain insights into stock behavior.

**Reporting**

* Documenting the analysis, methodology, visualizations, and model results.
* Preparing a final report that summarizes key findings, challenges, and future recommendations.

**Methodology**

The project will follow a structured approach:

**Data Collection**

* The dataset will be sourced using APIs like YFinance to retrieve historical stock data of AAPL, GOOG, MSFT, and AMZN.

import yfinance as yf

import pandas as pd

tickers = ['msft', 'aapl', 'goog', 'amzn']

stock\_data = {}

for symbol in tickers:

stock\_data[symbol] = yf.download(symbol, period="1y")

# 1. Ensure 'Date' is the index and is of datetime type

stock\_data[symbol] = stock\_data[symbol].set\_index(pd.to\_datetime(stock\_data[symbol].index))

# 2. Concatenate with explicit outer join and handle index carefully

df = pd.concat(stock\_data, axis=1, keys=tickers, join='outer')

print(df)

**Data Preprocessing**

* Handle missing data using forward fill techniques.
* Detect and remove duplicate entries and outliers.
* Normalize stock prices using Min-Max scaling for LSTM modeling.
* Calculate additional features like moving averages and daily returns.

df.isnull().sum() # To check if there are any missing values in the data

# Forward-fill any missing values in the data

df.ffill(inplace=True)

df.isnull().sum()

df.shape

df.duplicated().sum() # To check if there are any duplicate values

# Remove duplicates if any

df.drop\_duplicates(inplace=True)

df.duplicated().sum()

df.shape

from sklearn.preprocessing import MinMaxScaler

import numpy as np

# Prepare the close price data for scaling

adj\_close\_data = df.xs('Adj Close', axis=1, level=1)

print(adj\_close\_data.head())

dataset=adj\_close\_data.values

print(dataset)

# Scale data to range [0, 1]

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(dataset)

**Exploratory Data Analysis (EDA)**

* Use descriptive statistics to summarize stock price behaviors.
* Visualize trends using line plots for closing prices, moving averages, and daily returns.
* Generate heatmaps and pair plots to explore correlations between stocks.

# prompt: print summary statistics of apple stock. multilevel indexing is here

print(df.describe())

df.info() # Data types and non-null counts

df.dtypes # Data types for columns

Plotting Closing Prices

# plot the adjusting close prices of 4 companies in subplots

import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 6))

fig.suptitle('Adjusted Closing Prices of 4 Companies')

# Iterate through the companies and plot each subplot

for i, company in enumerate(adj\_close\_data.columns):

row = i // 2 # Determine the row for the subplot

col = i % 2 # Determine the column for the subplot

axes[row, col].plot(adj\_close\_data[company])

axes[row, col].set\_title(company)

axes[row, col].set\_xlabel('Date')

axes[row, col].set\_ylabel('Adjusted Close Price')

plt.tight\_layout()

plt.show()

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 6))

fig.suptitle('Volume of Sales of 4 Companies')

# Access the 'Volume' data from the original DataFrame 'df'

volume\_data = df.xs('Volume', axis=1, level=1)

# Iterate through the companies and plot each subplot

for i, company in enumerate(volume\_data.columns):

row = i // 2 # Determine the row for the subplot

col = i % 2 # Determine the column for the subplot

axes[row, col].plot(volume\_data[company])

axes[row, col].set\_title(company)

axes[row, col].set\_xlabel('Date')

axes[row, col].set\_ylabel('Volume')

plt.tight\_layout()

plt.show()

**Feature Selection**

* Use correlation analysis to identify key features for forecasting.
* Evaluate relationships between stock features like closing prices and volume to enhance predictions.

import pandas as pd

# Calculate moving averages and daily returns for each ticker

for ticker, data in stock\_data.items():

data[f'{ticker}\_MA\_10'] = data['Adj Close'].rolling(window=10).mean()

data[f'{ticker}\_MA\_20'] = data['Adj Close'].rolling(window=20).mean()

data[f'{ticker}\_MA\_50'] = data['Adj Close'].rolling(window=50).mean()

data[f'{ticker}\_Daily\_Return'] = data['Adj Close'].pct\_change()

# Concatenate each ticker's DataFrame on common dates (inner join)

df = pd.concat(stock\_data.values(), axis=1, join='inner')

# Flatten the multi-level columns

df.columns = ['\_'.join(col).strip() for col in df.columns.values]

# Display the final DataFrame without NaNs on non-overlapping dates

print(df.tail())

df.columns

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

# Define a list of tickers for subplots

tickers = list(stock\_data.keys())

# Set up a 2x2 grid of subplots

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10), sharex=True)

fig.suptitle("Moving Averages for Each Company", fontsize=20)

# Flatten axes array for easier indexing when iterating over tickers

axes = axes.flatten()

# Plot moving averages for each ticker in a 2x2 grid

for i, ticker in enumerate(tickers):

data = stock\_data[ticker]

ax = axes[i] # Get the specific subplot axis for each ticker

# Plot each moving average line

ax.plot(data.index, data[f'{ticker}\_MA\_10'], label=f'{ticker} 10-Day MA', color='blue', linestyle='--', linewidth=1)

ax.plot(data.index, data[f'{ticker}\_MA\_20'], label=f'{ticker} 20-Day MA', color='orange', linestyle='-', linewidth=1)

ax.plot(data.index, data[f'{ticker}\_MA\_50'], label=f'{ticker} 50-Day MA', color='green', linestyle='-', linewidth=1)

# Adding title, legend, and labels

ax.set\_title(f"{ticker} Moving Averages", fontsize=14)

ax.legend(loc='upper left')

ax.set\_ylabel("Price")

ax.grid(True, linestyle='--', alpha=0.7)

# Set date formatting for x-axis

ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m-%d'))

ax.xaxis.set\_major\_locator(mdates.MonthLocator())

# Hide any extra subplots (if there are fewer than 4 tickers)

for j in range(len(tickers), 4):

fig.delaxes(axes[j])

# General plot formatting

plt.xlabel("Date")

plt.xticks(rotation=45)

plt.tight\_layout(rect=[0, 0.03, 1, 0.95]) # Adjust layout to fit title

plt.show()

# prompt: as above plot daily return

# Plot daily returns for each ticker

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10), sharex=True)

fig.suptitle("Daily Returns for Each Company", fontsize=20)

axes = axes.flatten()

for i, ticker in enumerate(tickers):

data = stock\_data[ticker]

ax = axes[i]

ax.plot(data.index, data[f'{ticker}\_Daily\_Return'], label=f'{ticker} Daily Return', linestyle='--', marker='o')

ax.set\_title(f"{ticker} Daily Returns", fontsize=14)

ax.legend(loc='upper left')

ax.set\_ylabel("Daily Return")

ax.grid(True, linestyle='--', alpha=0.7)

ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m-%d'))

ax.xaxis.set\_major\_locator(mdates.MonthLocator())

for j in range(len(tickers), 4):

fig.delaxes(axes[j])

plt.xlabel("Date")

plt.xticks(rotation=45)

plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

plt.show()

**Correlation Analysis**

# Initialize an empty DataFrame to store adjusted close prices

closing\_df = pd.DataFrame()

# Collect adjusted close prices for each company in stock\_data

for ticker in stock\_data.keys():

closing\_df[ticker] = stock\_data[ticker]['Adj Close']

# Calculate daily percentage returns

tech\_rets = closing\_df.pct\_change()

tech\_rets.head()

print(tech\_rets.columns)

import seaborn as sns

sns.jointplot(x='goog', y='goog', data=tech\_rets, kind='scatter', color='seagreen')

sns.jointplot(x='goog', y='msft', data=tech\_rets, kind='scatter')

# We can simply call pairplot on our DataFrame for an automatic visual analysis of all the comparisons

sns.pairplot(tech\_rets, kind='reg')

import matplotlib.pyplot as plt

# Set up our figure by naming it returns\_fig, call PairPLot on the DataFrame

return\_fig = sns.PairGrid(tech\_rets.dropna())

# Using map\_upper we can specify what the upper triangle will look like.

return\_fig.map\_upper(plt.scatter, color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde) or the color map (BluePurple)

return\_fig.map\_lower(sns.kdeplot, cmap='cool\_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return

return\_fig.map\_diag(plt.hist, bins=30)

# Set up our figure by naming it returns\_fig, call PairPLot on the DataFrame

returns\_fig = sns.PairGrid(closing\_df)

# Using map\_upper we can specify what the upper triangle will look like.

returns\_fig.map\_upper(plt.scatter,color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde) or the color map (BluePurple)

returns\_fig.map\_lower(sns.kdeplot,cmap='cool\_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return

returns\_fig.map\_diag(plt.hist,bins=30)

import numpy as np

rets = tech\_rets.dropna()

area = np.pi \* 20

plt.figure(figsize=(10, 8))

plt.scatter(rets.mean(), rets.std(), s=area)

plt.xlabel('Expected return')

plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):

plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points', ha='right', va='bottom',

arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.3'))

plt.figure(figsize=(10, 6))

plt.subplot(2, 2, 1)

sns.heatmap(tech\_rets.corr(), annot=True, cmap='summer')

plt.title('Correlation of stock return')

plt.subplot(2, 2, 2)

sns.heatmap(closing\_df.corr(), annot=True, cmap='summer')

plt.title('Correlation of stock closing price')

**Model building**

To build a reliable LSTM model, the dataset was split into training and testing sets. 95% of the data was used for training the model, while the remaining 5% was reserved for testing and evaluation.

import yfinance as yf

# Get the stock data

df = yf.download('AAPL',period="10y")

Df

import matplotlib.pyplot as plt

plt.figure(figsize=(16,6))

plt.title('Close Price History')

plt.plot(df['Close']['AAPL'])

plt.xlabel('Date', fontsize=18)

plt.ylabel('Close Price USD ($)', fontsize=18)

plt.show()

import numpy as np

# Access the 'Close' column for 'AAPL' using multilevel indexing

data = df.loc[:, ('Close', 'AAPL')]

# Convert the Series to a DataFrame

data = data.to\_frame()

# Rename the column to 'Close' (optional, for consistency)

data.columns = ['Close']

# Convert the dataframe to a numpy array

dataset = data.values

# Get the number of rows to train the model on

training\_data\_len = int(np.ceil(len(dataset) \* .95))

print(dataset.shape)

training\_data\_len

# Scale the data

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(dataset)

scaled\_data

# Create the training data set

train\_data = scaled\_data[0:int(training\_data\_len),:]

# Create sequences for training the LSTM model

x\_train = []

y\_train = []

# Use slicing instead of iloc for numpy arrays

for i in range(60, len(train\_data)):

x\_train.append(train\_data[i-60:i,0]) # Append past 60 days' scaled data

y\_train.append(train\_data[i,0]) # Append the target value (scaled Adj Close)

if i<= 61:

print(x\_train)

print(y\_train)

print()

# Convert to numpy arrays and reshape for LSTM input

x\_train, y\_train = np.array(x\_train), np.array(y\_train)

#Reshape the data

x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))

# Check the shapes of the arrays

print(f"x\_train shape: {x\_train.shape}")

print(f"y\_train shape: {y\_train.shape}")

**Building LSTM Model**

from keras.models import Sequential

from keras.layers import Dense, LSTM

# Build the LSTM model

model = Sequential()

model.add(LSTM(128, return\_sequences=True, input\_shape=(x\_train.shape[1], 1))) # 60 time steps, 1 feature

model.add(LSTM(64, return\_sequences=False)) # Output of the last LSTM layer

model.add(Dense(25)) # Dense layer with 25 units

model.add(Dense(1)) # Final output layer with 1 unit

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(x\_train, y\_train, batch\_size=32, epochs=30)

from keras.models import Sequential

from keras.layers import Dense, LSTM

# Build the LSTM model

model = Sequential()

model.add(LSTM(128, return\_sequences=True, input\_shape=(x\_train.shape[1], 1))) # 60 time steps, 1 feature

model.add(LSTM(64, return\_sequences=False)) # Output of the last LSTM layer

model.add(Dense(25)) # Dense layer with 25 units

model.add(Dense(1)) # Final output layer with 1 unit

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(x\_train, y\_train, batch\_size=32, epochs=30)

**Result**

# Plot the data

train = data[:training\_data\_len]

valid = data[training\_data\_len:]

valid['Predictions'] = predictions

# Visualize the data

plt.figure(figsize=(16,6))

plt.title('Model')

plt.xlabel('Date', fontsize=18)

plt.ylabel('Close Price USD ($)', fontsize=18)

plt.plot(train['Close'])

plt.plot(valid[['Close', 'Predictions']])

plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')

plt.show()

valid

**Tools and Technologies**

* **Programming Language:** Python
* **Libraries:** Pandas, NumPy, Matplotlib, Seaborn, TensorFlow/Keras, YFinance
* **IDE:** Google Colab
* **Data Source:** YFinance API for stock price data
* .

### **Timeline**

Completed:

* **Week 1:** Data Collection and Preprocessing
* **Week 2:** Exploratory Data Analysis and Feature Selection
* **Week 3:** Model Building and Evaluation
* **Week 4:** Visualization, Reporting, and Final Submission

### **Conclusion**

This project effectively analyzed and predicted stock prices of major tech companies using techniques like exploratory data analysis (EDA), moving averages, and an LSTM model for forecasting. The model provided reasonable predictions, while the risk-return analysis offered insights into stock volatility and performance. Though there is room for improvement in accuracy, the project demonstrates a practical approach to stock price prediction, offering valuable insights for investors.