In [ ]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# for the scatter plot render
import seaborn as sns


URL = './NASDAQ_100_Data_From_2010.csv/NASDAQ_100_Data_From_2010.csv'
stock_data_raw = pd.read_csv(URL)
stock_data_raw.head()
```

Out[ ]:

	Date\tOpen\tHigh\tLow\tClose\tAdj Close\tVolume\tName
0	2010-01-04\t7.622499942779541\t7.6607141494750...
1	2010-01-05\t7.664286136627197\t7.6996431350708...
2	2010-01-06\t7.656428813934326\t7.6867861747741...
3	2010-01-07\t7.5625\t7.5714287757873535\t7.4660...
4	2010-01-08\t7.510714054107666\t7.5714287757873...

In [ ]:

```python
# Because the raw data is not splitted by , need to handle the dataset first
def generate_dataset():
    URL = './NASDAQ_100_Data_From_2010.csv/NASDAQ_100_Data_From_2010.csv'
    # URL = './google stock/GOOG.csv'
    NASDAQ_stock_data = pd.read_csv(URL)
    NASDAQ_stock_data.head()
    # df_All_stock = pd.DataFrame(columns="Date\tOpen\tHigh\tLow\tClose\tAdj Close\tVolume\tName".split('\t'))

    raw_list = []
    # df_All_stock.iloc[:0] = ['2010-01-05', '7.664286136627197', '7.699643135070801', '7.6160712242126465', '7.65642881
    for i in range(0, len(NASDAQ_stock_data.index)):
        row_str_list = NASDAQ_stock_data.iloc[i,0].split('\t')
        raw_list.append(row_str_list)
    df_All_stock = pd.DataFrame(raw_list, columns="Date\tOpen\tHigh\tLow\tClose\tAdj Close\tVolume\tName".split('\t'))

    df_All_stock.head()
    print("======df_All_stock.shape == ", df_All_stock.shape)
    df_All_stock['Date'] = df_All_stock['Date'].astype("datetime64")
    df_All_stock['Open'] = df_All_stock['Open'].astype("float")
    df_All_stock['High'] = df_All_stock['High'].astype("float")
    df_All_stock['Low'] = df_All_stock['Low'].astype("float")
    df_All_stock['Close'] = df_All_stock['Close'].astype("float")
    df_All_stock['Adj Close'] = df_All_stock['Adj Close'].astype("float")
    df_All_stock['Volume'] = df_All_stock['Volume'].astype("int")
    df_All_stock['Name'] = df_All_stock['Name'].astype("string")
    df_All_stock.dtypes

    df_All_stock.to_csv('NASDAQ_stock_data_df.csv')
# generate_dataset()
```

In [ ]:

```python
URL = './NASDAQ_stock_data_df.csv'
stock_data = pd.read_csv(URL)
# stock_data.set_index("Date", inplace=True)
stock_data.drop(["Unnamed: 0"], axis=1, inplace=True)
print("stock_data.shape>>>", stock_data.shape)
print("stock_data.columns>>>", stock_data.columns)
print(stock_data.tail(1))
stock_data.head()
```

```
stock_data.shape>>> (271680, 8)
stock_data.columns>>> Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'Name'], dtype='object')
              Date        Open     High         Low  Close  Adj Close  \
271679  2021-09-10  296.910004  306.263  296.809998  301.5       301.5

        Volume Name
271679  6089600   ZM
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	Name
0	2010-01-04	7.622500	7.660714	7.585000	7.643214	6.562591	493729600	AAPL
1	2010-01-05	7.664286	7.699643	7.616071	7.656429	6.573935	601904800	AAPL
2	2010-01-06	7.656429	7.686786	7.526786	7.534643	6.469369	552160000	AAPL

	Date	Open	High	Low	Close	Adj Close	Volume	Name
3	2010-01-07	7.562500	7.571429	7.466071	7.520714	6.457407	477131200	AAPL
4	2010-01-08	7.510714	7.571429	7.466429	7.570714	6.500339	447610800	AAPL

Explotary Data Anaylsis

The first step is to do some EDA to see different charactertics of the data set.

1. Choose the desired symbol of different companies, for analysis, using one company= `AAPL`
2. Describe the data columns and shapes (features and observations)
3. Draw the scatter plot of different features and see the relationship among them
4. Draw the scatter plot for 4 companies to see the correlation among them
5. Decide what to predict

In [ ]:

```
"""
===1. Choose the Companies Symbol

"""
print(">>> RAW data shape", stock_data.shape)
stock_data.info()
# Unique data obervation of each column
print(stock_data.nunique())
# See the symbols list
stock_data_symbols = stock_data[["Name"]]
symbol_list = stock_data_symbols["Name"].unique()
print("====== Companies Symbol List ======\n")
print(symbol_list)
```

```
>>> RAW data shape (271680, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271680 entries, 0 to 271679
Data columns (total 8 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   Date       271680 non-null  object
 1   Open       271680 non-null  float64
 2   High       271680 non-null  float64
 3   Low        271680 non-null  float64
 4   Close      271680 non-null  float64
 5   Adj Close  271680 non-null  float64
 6   Volume     271680 non-null  int64
 7   Name       271680 non-null  object
dtypes: float64(5), int64(1), object(2)
memory usage: 16.6+ MB
Date           2943
Open          68178
High          69492
Low           68973
Close         70096
Adj Close    188419
Volume       124591
Name            102
dtype: int64
====== Companies Symbol List ======

['AAPL' 'ADBE' 'ADI' 'ADP' 'ADSK' 'AEP' 'ALGN' 'AMAT' 'AMD' 'AMGN' 'AMZN'
 'ANSS' 'ASML' 'ATVI' 'AVGO' 'BIDU' 'BIIB' 'BKNG' 'CDNS' 'CDW' 'CERN'
 'CHKP' 'CHTR' 'CMCSA' 'COST' 'CPRT' 'CRWD' 'CSCO' 'CSX' 'CTAS' 'CTSH'
 'DLTR' 'DOCU' 'DXCM' 'EA' 'EBAY' 'EXC' 'FAST' 'FB' 'FISV' 'FOX' 'FOXA'
 'GILD' 'GOOG' 'GOOGL' 'HON' 'IDXX' 'ILMN' 'INCY' 'INTC' 'INTU' 'ISRG'
 'JD' 'KDP' 'KHC' 'KLAC' 'LRCX' 'LULU' 'MAR' 'MCHP' 'MDLZ' 'MELI' 'MNST'
 'MRNA' 'MRVL' 'MSFT' 'MTCH' 'MU' 'NFLX' 'NTES' 'NVDA' 'NXPI' 'OKTA'
 'ORLY' 'PAYX' 'PCAR' 'PDD' 'PEP' 'PTON' 'PYPL' 'QCOM' 'REGN' 'ROST'
 'SBUX' 'SGEN' 'SIRI' 'SNPS' 'SPLK' 'SWKS' 'TCOM' 'TEAM' 'TMUS' 'TSLA'
 'TXN' 'VRSK' 'VRSN' 'VRTX' 'WBA' 'WDAY' 'XEL' 'XLNX' 'ZM']
```

In [ ]:

```
# Create the interested symbol list
stock_dict = {
    'AAPL': 'Apple Inc.',
    'AMD': 'Advanced Micro Devices, Inc.',
    'AMZN': 'Amazon.com, Inc.',
    'BIDU': 'Baidu, Inc.ADS',
```

```
    'CSCO': 'Cisco Systems, Inc.(DE)',
    'FB': 'Facebook, Inc.',
    'GOOG': 'Alphabet Inc.Class C Capital Stock',
    'GOOGL': 'Alphabet Inc.',
    'JD': 'JD.com, Inc.',
    'NFLX': 'Netflix, Inc.',
    'NVDA': 'NVIDIA Corporation',
    'PYPL': 'PayPal Holdings, Inc.',
    'QCOM': 'QUALCOMM Incorporated',
    'TSLA': 'Tesla, Inc.',
    'ZM': 'Zoom Video Communications, Inc.'
}

stock_list = ['AAPL','AMZN','GOOG','FB']

# Create the interested symbol list
stock_dict = {
    'AAPL': 'Apple Inc.',
    'AMD': 'Advanced Micro Devices, Inc.',
    'AMZN': 'Amazon.com, Inc.',
    'BIDU': 'Baidu, Inc.ADS',
    'CSCO': 'Cisco Systems, Inc.(DE)',
    'FB': 'Facebook, Inc.',
    'GOOG': 'Alphabet Inc.Class C Capital Stock',
    'GOOGL': 'Alphabet Inc.',
}

stock_list = ['AAPL','AMZN','GOOG','FB']


def stock_show(stock_name):
    # print(stock_name, end=" ")
    # print(" stock is %s" %stock_dict[stock_name])
    if stock_name in list(stock_dict.keys()):
        print("Asking stock is %s" %stock_dict[stock_name])
        ans = "The company full name: "+ stock_dict[stock_name]
        return ans
    return ""

ans = stock_show("FBI")
print(ans)
```

Apple Company Stock EDA

In [ ]:

```
"""
google_stock_data = google_stock_data[["date", "open", "close"]]
google_stock_data["date"] = pd.to_datetime(google_stock_data["date"].apply(lambda x: x.split()[0]))
google_stock_data.set_index('date', inplace=True)
google_stock_data.head()
"""
# Choose the data feature we want to predict ==== 'date', 'open', 'close'
def generate_company_set(com_symbol):
    row_list = []
    # df_com = pd.DataFrame(columns=stock_data.columns)
    for i in range(0, len(stock_data.index)):
        if stock_data.loc[i,"Name"] == com_symbol:
            tmp_df = stock_data.loc[i,:]
            row_list.append(tmp_df)

    df_com = pd.DataFrame(row_list, columns=stock_data.columns)
    return df_com

apple = generate_company_set("AAPL")
# google = generate_company_set("GOOG")
apple["Date"] = pd.to_datetime(apple["Date"].apply(lambda x: x.split()[0]))
apple.set_index('Date', inplace=True)
print("======Data Shape======")
print(apple.shape)
print("======Data Dtypes======")
print(apple.dtypes)
apple.head()
```

```
======Data Shape======
(2943, 7)
======Data Dtypes======
Open          float64
```

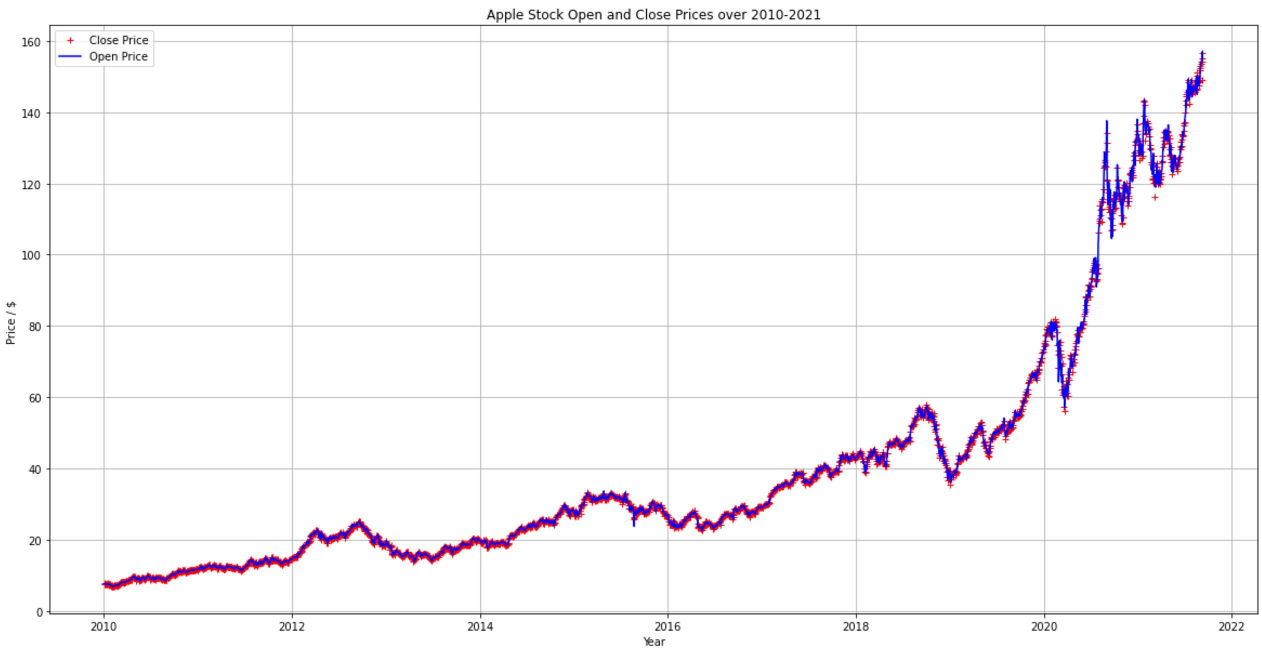
```
High          float64
Low           float64
Close         float64
Adj Close     float64
Volume          int64
Name           object
dtype: object
```

Out[ ]:

Date	Open	High	Low	Close	Adj Close	Volume	Name
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.562591	493729600	AAPL
2010-01-05	7.664286	7.699643	7.616071	7.656429	6.573935	601904800	AAPL
2010-01-06	7.656429	7.686786	7.526786	7.534643	6.469369	552160000	AAPL
2010-01-07	7.562500	7.571429	7.466071	7.520714	6.457407	477131200	AAPL
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.500339	447610800	AAPL

In [ ]:

```
"""
fg,  ax =plt.subplots(1,2,figsize=(20,7))
ax[0].plot(google_stock_data['open'],label='Open',color='green')
ax[0].set_xlabel('Date',size=15)
ax[0].set_ylabel('Price',size=15)
ax[0].legend()

ax[1].plot(google_stock_data['close'],label='Close',color='red')
ax[1].set_xlabel('Date',size=15)
ax[1].set_ylabel('Price',size=15)
ax[1].legend()

fg.show()
"""

plt.figure(figsize=(20,10))
plt.plot(apple['Close'], '+r',label='Close Price')
plt.plot(apple['Open'], '-b',label='Open Price')
plt.legend()
plt.grid()
plt.xlabel("Year")
plt.ylabel("Price / $ ")
plt.title("Apple Stock Open and Close Prices over 2010-2021")
plt.show()
```



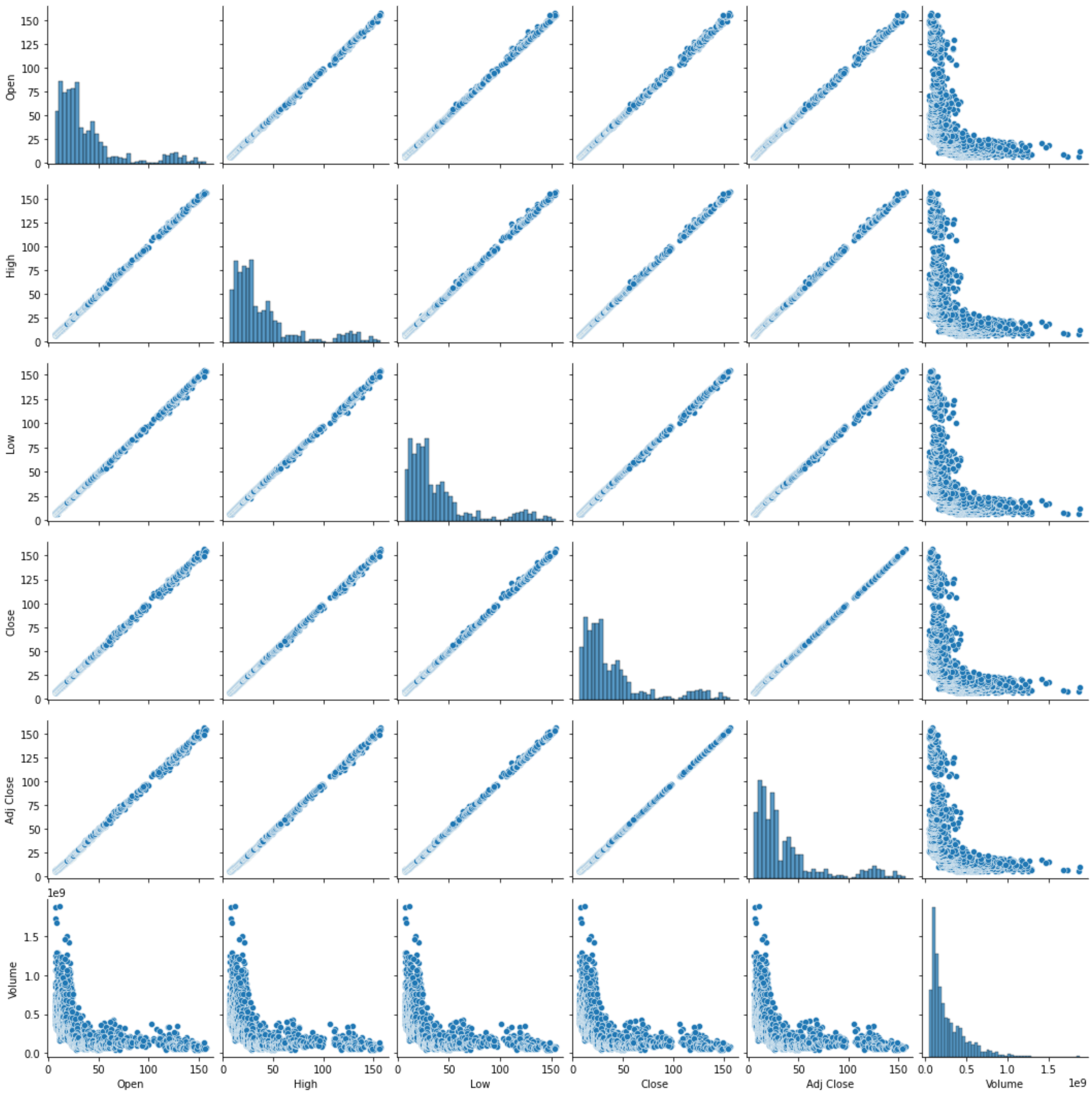
In [ ]:

```
# 3. Draw the scatter plot of different features and see the relationship among them
# print(apple.head())
plt.figure(figsize=(10,10))
sns.pairplot(apple)

#=== 4. Draw the scatter plot for 4 companies to see the correlation among them
```

```
# 5. Decide what to predict
"""
1. For Stock data, the most valuable data is the closing price.
From the Scatter plot, we can see there are many features are linear to close price and we may use them to predict furtu
"""
```

Out[ ]:

```
'\n1. For Stock data, the most valuable data is the closing price. \nFrom the Scatter plot, we can see there are many features are linear to close price and we may use them to predict furture close price.\n'
<Figure size 720x720 with 0 Axes>
```



In [ ]:

```
four_companies_closing = pd.DataFrame(index={"Date": range(0, 2944)})
# stock_list = ['GOOG', 'AMZN']

for company in stock_list:
    tmp = generate_company_set(company)
    # print(tmp.shape)
    tmp[company] = tmp.loc[:, "Close"]
    tmp = tmp[[company]]
    tmp.reset_index(drop=True)
    print(tmp.head())
    # four_companies_closing.loc[:, company] = tmp["Close"]
    four_companies_closing = four_companies_closing.join(tmp)
    # four_companies_closing = pd.concat([four_companies_closing, tmp[["Close"]]], axis=1, ignore_index=True)
    # four_companies_closing["Date"] = stock_data["Date"]
four_companies_closing.tail()
```

```
       AAPL
0  7.643214
1  7.656429
```

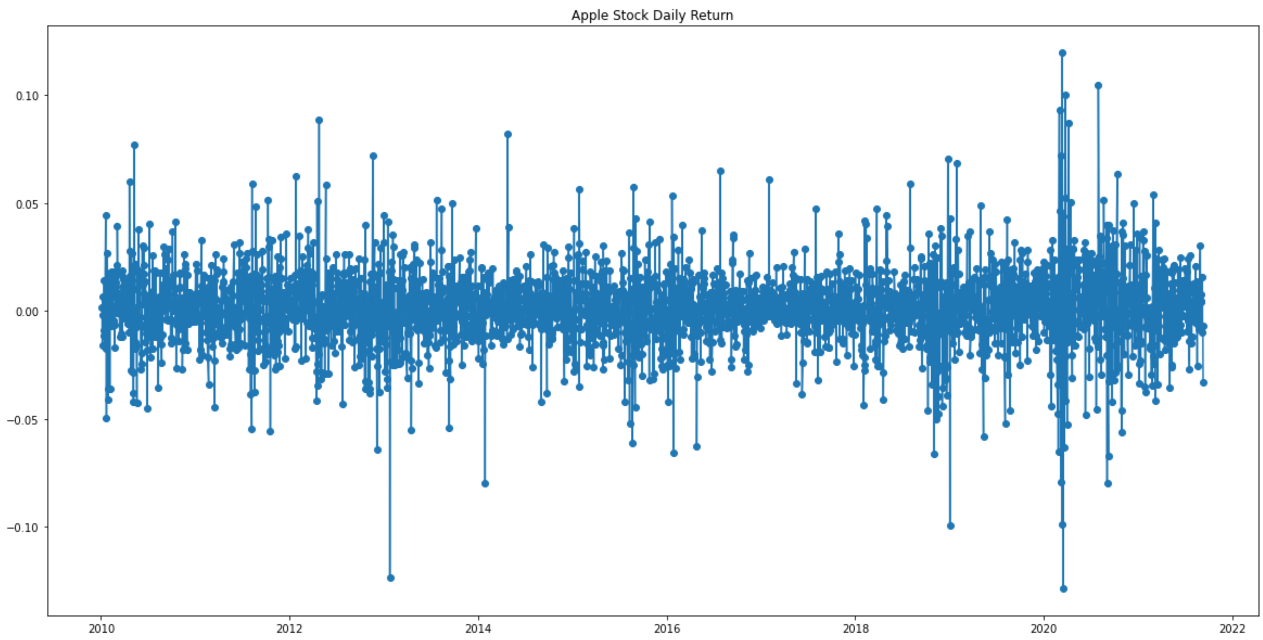
```
2   7.534643
3   7.520714
4   7.570714
              AMZN
29430   133.899994
29431   134.690002
29432   132.250000
29433   130.000000
29434   133.520004
               GOOG
115976   312.204773
115977   310.829926
115978   302.994293
115979   295.940735
115980   299.885956
               FB
106483   38.230000
106484   34.029999
106485   31.000000
106486   32.000000
106487   33.029999
```

Out[ ]:

	AAPL	AMZN	GOOG	FB
Date	NaN	NaN	NaN	NaN

In [ ]:

```
apple['Daily Return'] = apple['Adj Close'].pct_change()
plt.figure(figsize=(20,10))
plt.plot(apple['Daily Return'], marker='o',label='Daily Return')
plt.title("Apple Stock Daily Return")
plt.show()
```



In [ ]:

```
# Check to missing value and handle it
apple.isnull().sum()
```

Out[ ]:

```
Open            0
High            0
Low             0
Close           0
Adj Close       0
Volume          0
Name            0
Daily Return    1
dtype: int64
```

Choose Prediction

Choose the 'close' price to predict.

Data Pre-processing

Dataset Desgin

From above EDA analysis, we choose the "Close" price of different companies to predict. For the training dataset, we need to do as follow:

1. Change the dataset into numeric because DNN only takes numeric values as input.
   - Since the data already in float, no need to vectorize such as "one-hot" coding.
2. Generate the data-set for train and test
   - In python, there are many ways to generate the dataset, one is put all the data into one dataframe. The other is put the data into a Generator.

Define some general use function

```
# Collect Data Function using the company Symbol
def generate_company_set_date(com_symbol, year=2010, month=1, day=1):
# Split dataset into train and test using the percentage
def generate_MinMAX_dataset(df, train_split_per):

def split_sequence(sequence_x, sequence_y, n_steps_in, n_steps_out):
# Give stock and related cols. Plot different plt
### Default is plot the Close Price
def plot_df_val(df, column, stock, title=' Price History For ', ylabel=" Price USD($) for "):
def draw_train_from_history(his):


# Evalution Metrics
def calculate_rmse(y_true, y_pred):
def calculate_mape(y_true, y_pred):
```

In [ ]:

```
# 1. Collect Data Function using the company Symbol
import datetime
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split

def generate_company_set_date(com_symbol, year=2010, month=1, day=1):
    row_list = []
    for i in range(0, len(stock_data.index)):
        if stock_data.loc[i,"Name"] == com_symbol:
            tmp_df = stock_data.loc[i,:]
            row_list.append(tmp_df)

    df_com = pd.DataFrame(row_list, columns=stock_data.columns)
    # df_com["Date"] = df_com["Date"].astype("datetime64")
    df_com["Date"] = pd.to_datetime(df_com["Date"])
    # print(df_com.dtypes)
    # Filter for specific date
    df_com = df_com[df_com["Date"] > pd.Timestamp(datetime.date(year, month, day))]
    # df_com["Date"] = pd.to_datetime(df_com["Date"].apply(lambda x: x.split()[0]))
    # df_com.set_index('Date', inplace=True)
    # df_com.sort_index(axis=0)
    return df_com

"""
=========== For Draw the image ===========

"""
# For Regression Problem
def draw_train_from_history(his):
    plt.figure(figsize=(10,10))
    # fig, (ax1, ax2) = plt.subplots(2,1)
    plt.plot(his.history['loss'],label='Train_Loss')
    plt.plot(his.history['val_loss'],label='Validation_Loss')
    plt.legend()
    # ax2.plot(history.history['mean_squared_error'],label='Train_ACC')
    # ax2.plot(history.history['val_mean_squared_error'],label='Validation_ACC')
    # ax2.legend()


def plot_df_val(df, column, stock, title=' Price History For ', ylabel="USD($) for "):
    plt.clf()
    plt.figure(figsize=(16,6))
```

```python
    plt.title(str(column)+title+stock)
    plt.plot(df[column], label="column")
    plt.xlabel('Date', fontsize=18)
    plt.ylabel(ylabel +stock, fontsize=18)
    plt.grid()
    plt.legend(column)
    plt.show()


"""
=========== Calculate the metrics RMSE and MAPE ===========
"""
def calculate_rmse(y_true, y_pred):
    """
    Calculate the Root Mean Squared Error (RMSE)
    """
    rmse = np.sqrt(np.mean((y_true-y_pred)**2))
    return rmse

def calculate_mape(y_true, y_pred):
    """
    Calculate the Mean Absolute Percentage Error (MAPE) %
    """
    y_pred, y_true = np.array(y_pred), np.array(y_true)
    y_pred = np.nan_to_num(y_pred)
    y_true = np.nan_to_num(y_true)
    mape = np.mean(np.abs((y_true-y_pred) / y_true))*100
    return mape



"""
============ For DataSet Prepration ============
"""
# Use for Scale
# scaler = MinMaxScaler(feature_range=(0, 1))
scaler = StandardScaler()


# StandardScaler
def std_data(x):
    scaler_data = scaler.fit_transform(x)
    return scaler_data


# split data into samples
# n_steps_in= sample input_train
# n_steps_out= sample output_train
def split_sequence(sequence_x, sequence_y, n_steps_in, n_steps_out):
    X = []
    y = []
    for i in range(len(sequence_x)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the sequence
        if out_end_ix > len(sequence_x):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence_x[i:end_ix], sequence_x[end_ix:out_end_ix,0]
        X.append(seq_x)
        y.append(seq_y)
    print(">>>>>>>>>>In split_sequence\\n",np.array(X).shape)
    return np.array(X), np.array(y)


def split_dataset_test_train(X, y, split_factor):
    train_data_len = int(np.ceil( len(X[:,0]) * split_factor))
    # Split the data into train and test
    X_train = X[0:train_data_len]
    X_test = X[train_data_len:]
    y_train = y[0:train_data_len]
    y_test = y[train_data_len:]

    return X_train, X_test, y_train, y_test

def filter_x_y(df):
    predict_dataset = df.filter(['Date', 'Close', 'Open', 'Low', 'High', 'Adj Close'])
    X = predict_dataset.filter(['Close', 'Open', 'Low', 'High', 'Adj Close'])
    # X = predict_dataset.filter(['Close'])
```

```
    y = predict_dataset.filter(['Close'])

    return X, y



def generate_from_data(X, y, n_steps_in, n_steps_out):
    # Set the time step for both train data set and test
    # print(X)
    # X_train, Y_train = split_sequence(X, y, n_steps_in, n_steps_out)

    X_, Y_ = split_sequence(X, y, n_steps_in, n_steps_out)
    n_feature = 5
    # reshape from [samples, timesteps] into [samples, timesteps, features]
    X_ = X_.reshape(X_.shape[0], X_.shape[1], n_feature)
    # X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], n_feature)
    print("====Generate X_ shape====", X_.shape)    #  (2348, 7, 5)
    print("====Generate Y_ shape====", Y_.shape)    # (2348, 1, 5)
    # print("====Generate X_test shape====", X_test.shape)    # (581, 7, 5)
    # print("====Generate Y_test shape====", Y_test.shape)    # (581, 1, 5)
    return X_, Y_

"""
====== Run main function to get the normalized data for train
1. get the raw datafram
2. Scale the data
3. Split into X_train adn X_test
"""
apple = generate_company_set_date("AAPL", 2010,1,1)
# Get the data for X, y
X, y= filter_x_y(apple)

# Use for Scale
scaler = MinMaxScaler()
X = scaler.fit_transform(np.array(X))

X_, y_ = generate_from_data(X, np.array(y), 50, 1)

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle=False)
X_train, X_test, y_train, y_test = split_dataset_test_train(X_, y_, 0.8)

train_size = len(X_train)
print(y_test[10])
```

```
>>>>>>>>>>In split_sequence\n (2893, 50, 5)
====Generate X_ shape==== (2893, 50, 5)
====Generate Y_ shape==== (2893, 1)
[0.27823715]
```

In [ ]:

```
X_train[len(X_train)-1,0,0]
```

Out[ ]:

```
0.264771977788475465
```

Building a Basic Model to predict

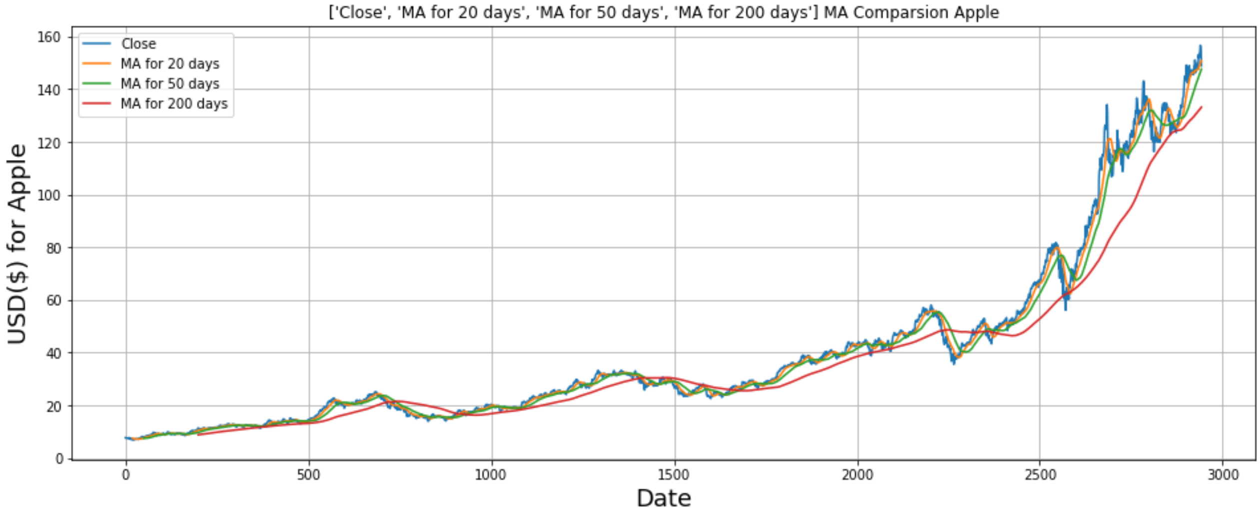
Moving Average MA method

- MA is an average that moves along the timeseries data such as stock data.
- One important feature is older data points get dropped as newer data are added.
- Commonly used periods are 20-days, 50-days, and 200 days for short-time, medium-time, long-time ### Types of MA
- SMA
- Exponential MA

In [ ]:

```
# For SMA calculation

MA_day = [20, 50, 200]
# apple = generate_company_set("AAPL",2016)
for ma in MA_day:
    column_name = f"MA for {ma} days"
    apple[column_name] = apple["Close"].rolling(ma).mean()
print(apple.columns)
plot_df_val(apple, ["Close", 'MA for 20 days', 'MA for 50 days', 'MA for 200 days'], "Apple", " MA Comparsion ")
apple.head(30)
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'Name',
       'MA for 20 days', 'MA for 50 days', 'MA for 200 days'],
      dtype='object')
<Figure size 432x288 with 0 Axes>
```



Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume	Name	MA for 20 days	MA for 50 days	MA for 200 days
0	2010-01-04	7.622500	7.660714	7.585000	7.643214	6.562591	493729600	AAPL	NaN	NaN	NaN
1	2010-01-05	7.664286	7.699643	7.616071	7.656429	6.573935	601904800	AAPL	NaN	NaN	NaN
2	2010-01-06	7.656429	7.686786	7.526786	7.534643	6.469369	552160000	AAPL	NaN	NaN	NaN
3	2010-01-07	7.562500	7.571429	7.466071	7.520714	6.457407	477131200	AAPL	NaN	NaN	NaN
4	2010-01-08	7.510714	7.571429	7.466429	7.570714	6.500339	447610800	AAPL	NaN	NaN	NaN
5	2010-01-11	7.600000	7.607143	7.444643	7.503929	6.442997	462229600	AAPL	NaN	NaN	NaN
6	2010-01-12	7.471071	7.491786	7.372143	7.418571	6.369709	594459600	AAPL	NaN	NaN	NaN
7	2010-01-13	7.423929	7.533214	7.289286	7.523214	6.459555	605892000	AAPL	NaN	NaN	NaN
8	2010-01-14	7.503929	7.516429	7.465000	7.479643	6.422143	432894000	AAPL	NaN	NaN	NaN
9	2010-01-15	7.533214	7.557143	7.352500	7.354643	6.314816	594067600	AAPL	NaN	NaN	NaN
10	2010-01-19	7.440357	7.685357	7.401429	7.680000	6.594175	730007600	AAPL	NaN	NaN	NaN
11	2010-01-20	7.675357	7.698214	7.482143	7.561786	6.492675	612152800	AAPL	NaN	NaN	NaN
12	2010-01-21	7.574286	7.618214	7.400357	7.431071	6.380439	608154400	AAPL	NaN	NaN	NaN
13	2010-01-22	7.385000	7.410714	7.041429	7.062500	6.063978	881767600	AAPL	NaN	NaN	NaN
14	2010-01-25	7.232500	7.310714	7.149643	7.252500	6.227116	1065699600	AAPL	NaN	NaN	NaN
15	2010-01-26	7.355357	7.632500	7.235000	7.355000	6.315125	1867110000	AAPL	NaN	NaN	NaN
16	2010-01-27	7.387500	7.520714	7.126071	7.424286	6.374613	1722568400	AAPL	NaN	NaN	NaN
17	2010-01-28	7.318929	7.339286	7.096429	7.117500	6.111203	1173502400	AAPL	NaN	NaN	NaN
18	2010-01-29	7.181429	7.221429	6.794643	6.859286	5.889495	1245952400	AAPL	NaN	NaN	NaN

	Date	Open	High	Low	Close	Adj Close	Volume	Name	MA for 20 days	MA for 50 days	MA for 200 days
19	2010-02-01	6.870357	7.000000	6.832143	6.954643	5.971371	749876400	AAPL	7.395214	NaN	NaN
20	2010-02-02	6.996786	7.011429	6.906429	6.995000	6.006022	698342400	AAPL	7.362804	NaN	NaN
21	2010-02-03	6.970357	7.150000	6.943571	7.115357	6.109362	615328000	AAPL	7.335750	NaN	NaN
22	2010-02-04	7.026071	7.084643	6.841786	6.858929	5.889189	757652000	AAPL	7.301964	NaN	NaN
23	2010-02-05	6.879643	7.000000	6.816071	6.980714	5.993757	850306800	AAPL	7.274964	NaN	NaN
24	2010-02-08	6.988929	7.067143	6.928571	6.932857	5.952664	478270800	AAPL	7.243071	NaN	NaN
25	2010-02-09	7.015000	7.053571	6.955357	7.006786	6.016141	632886800	AAPL	7.218214	NaN	NaN
26	2010-02-10	6.996071	7.021429	6.937857	6.968571	5.983329	370361600	AAPL	7.195714	NaN	NaN
27	2010-02-11	6.960000	7.133929	6.930714	7.095357	6.092191	550345600	AAPL	7.174321	NaN	NaN
28	2010-02-12	7.075357	7.201429	6.982143	7.156429	6.144627	655468800	AAPL	7.158161	NaN	NaN
29	2010-02-16	7.212143	7.274643	7.197143	7.264286	6.237236	543737600	AAPL	7.153643	NaN	NaN

In [ ]:

```
ma_predicted_values_rmse = []
ma_predicted_values_mape = []

for ma in MA_day:
    column_name = f"MA for {ma} days"
    rmse = calculate_rmse(apple.loc[train_size:, "Close"], apple.loc[train_size:,column_name])
    mape = calculate_mape(apple.loc[train_size:, "Close"], apple.loc[train_size:,column_name])
    ma_predicted_values_rmse.append(rmse)
    ma_predicted_values_mape.append(mape)
    print("RMSE for %(ma_day)s is %(value).4f " %{'ma_day':column_name, 'value':rmse})
    print("MAPE for %(ma_day)s is %(value).4f %%" %{'ma_day':column_name, 'value':mape})
```

```
RMSE for MA for 20 days is 4.7529
MAPE for MA for 20 days is 4.1027 %
RMSE for MA for 50 days is 8.0100
MAPE for MA for 50 days is 7.1363 %
RMSE for MA for 200 days is 18.1478
MAPE for MA for 200 days is 15.0209 %
```

Building DNN network to predict

In [ ]:

```
"""
General DNN lib
"""

# import keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU, LSTM, SimpleRNN,Flatten, Dropout
from tensorflow.keras.optimizers import Adam, RMSprop
# from keras import regularizers
from tensorflow.keras.utils import plot_model
from tensorflow.keras.metrics import categorical_crossentropy
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import datetime
import csv
```

LSTM

In [ ]:

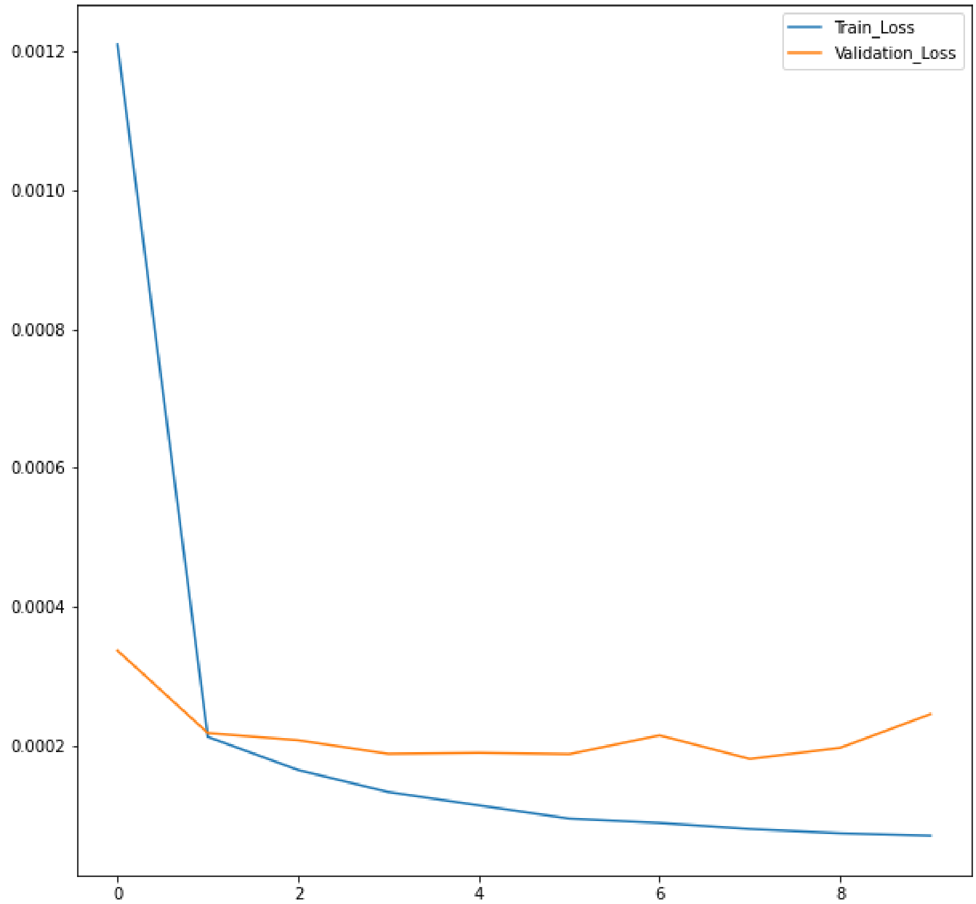
```
epochs = 50
verbose=2
validation_split = 0.2
batch_size=32

"""
callback_list = [
                 keras.callbacks.EarlyStopping(monitor='val_loss', patience=2),
                 keras.callbacks.ModelCheckpoint(filepath='best_model.h5',
                                                 monitor='val_loss',
                                                 save_best_only=True),
                tf.keras.callbacks.TensorBoard(log_dir='./log_dir')
]
"""
callback_list = [
                keras.callbacks.EarlyStopping(monitor='val_loss', patience=2),
                keras.callbacks.ModelCheckpoint(filepath='best_model-20.h5',
                                                 monitor='val_loss',
                                                 save_best_only=True),
                tf.keras.callbacks.TensorBoard(log_dir='./log_dir')
]


def get_LSTM_GRU_model(X_train_step, feature):
    # model
    model = Sequential()
    model.add(GRU(128, return_sequences=True, recurrent_dropout=0.1, input_shape=(X_train_step, feature)))
    model.add(Dropout(0.1))
    model.add(GRU(64))
    model.add(Dropout(0.1))
    model.add(Dense(16))
    model.add(Dense(1))     # Dense for 1, predict furture 1 days.
    model.compile(optimizer=Adam(learning_rate=0.0008), loss='mse')
    model.summary()
    return model
"""

def get_LSTM_GRU_model(X_train_step, feature):
    # model
    model = Sequential()
    model.add(GRU(128, return_sequences=True,recurrent_dropout=0.1, activation='relu', input_shape=(X_train_step, featur
    model.add(Dropout(0.1))
    model.add(GRU(64,recurrent_dropout=0.1))
    model.add(Dense(16))
    model.add(Dense(1))     # Dense for 1, predict furture 1 days.
    model.compile(optimizer=Adam(learning_rate=0.008), loss='mse')
    model.summary()
    return model
"""

print("X_train shape", X_train.shape)
print("Y_train shape", y_train.shape)

model = get_LSTM_GRU_model(50 ,5)
history = model.fit(X_train,
                    y_train,
                    epochs=epochs,
                    callbacks=callback_list,
                    batch_size=32,
                    verbose=verbose, validation_split=validation_split)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_20416/3228302032.py in <module>
     14 """
     15 callback_list = [
---> 16                 keras.callbacks.EarlyStopping(monitor='val_loss', patience=2),
     17                 keras.callbacks.ModelCheckpoint(filepath='best_model-20.h5',
     18                                                  monitor='val_loss',

NameError: name 'keras' is not defined
```

In [ ]:

```
draw_train_from_history(history)
```

Out[ ]: 111



In [ ]:

```
"""
============demonstrate prediction============
"""

y_predict = model.predict(X_test, verbose=2)
y_predict = np.mean(y_predict, axis=1)

y_predict_pad = np.zeros((y_predict.shape[0],5))
y_predict_pad[:,0] = y_predict[:]

y_test_pad = np.zeros((y_test.shape[0],5))
y_test_pad[:,0] = y_test[:,0]
y_test_pad = scaler.inverse_transform(y_test_pad)


print(y_predict.shape)
y_predict_pad = scaler.inverse_transform(y_predict_pad)

# test_mean = np.mean(y_predict, axis=1)
# print("Test Mean is", test_mean[0:10])
# test_mean = scaler.inverse_transform(y_predict)
print("y_predict is", y_predict_pad.shape)
y_predict_pad[-10:,0]
```

```
18/18 - 1s - 824ms/epoch - 46ms/step
(576,)
y_predict is (576, 5)
```

Out[ ]:

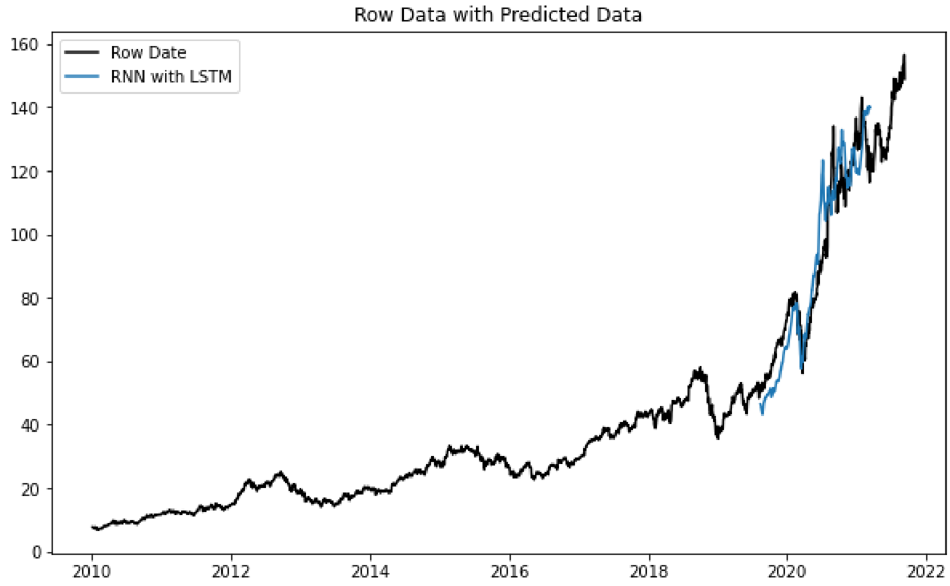
```
array([138.94667558, 139.65163256, 140.3519724 , 140.33016381,
       139.75017311, 139.47713702, 139.6677702 , 140.09386832,
       140.2944145 , 140.18694333])
```

In [ ]:

```
plt.figure(figsize=(10,6))
plt.title("Row Data with Predicted Data")
plt.plot(apple.Date, apple.Close, 'k', label='Row Date')
plt.plot(pd.date_range(end='2021-3-10', periods=576, freq='D'), y_predict_pad[:,0], label='RNN with LSTM')
plt.legend()
```

Out[ ]:

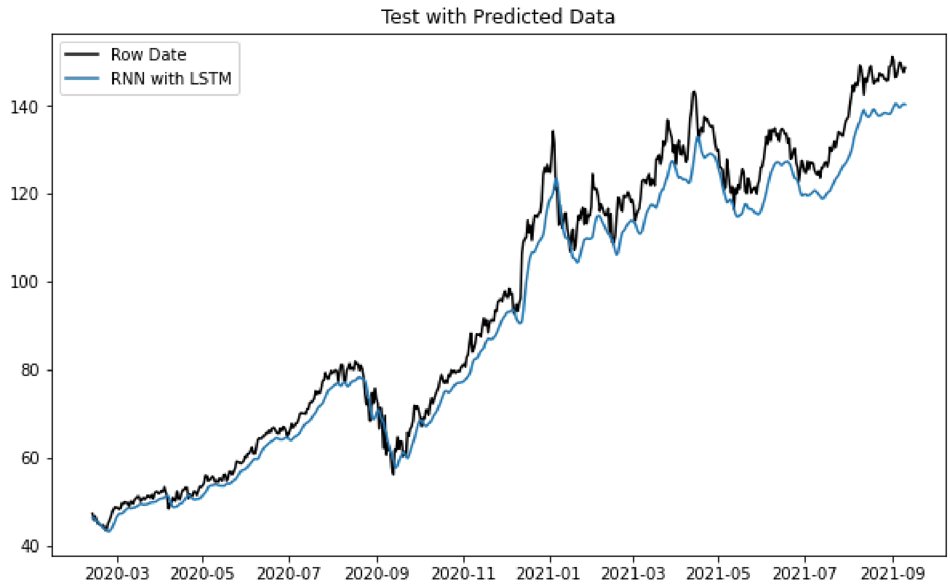
```
<matplotlib.legend.Legend at 0x1f727782550>
```



In [ ]:

```
plt.figure(figsize=(10,6))
plt.title("Test with Predicted Data")
plt.plot(pd.date_range(end='2021-09-10', periods=576, freq='D'), y_test_pad[:,0], 'k', label='Row Date')
plt.plot(pd.date_range(end='2021-09-10', periods=576, freq='D'), y_predict_pad[:,0], label='RNN with LSTM')
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x1f72780c5e0>



In [ ]:

```
rmse = calculate_rmse(y_test_pad[:,0], y_predict_pad[:,0])
print("Current ", rmse)
```

```
Current  5.746011865277723
```

In [ ]:

```
plot_model(model, show_shapes=True)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for plot_model/model_to_dot to work.
```

Train for specified Company List

Save all the model name as "mode*[companyName]*[timeDays].h5"

In [ ]:

```
# Create the interested symbol list
stock_dict = {
    'AAPL': 'Apple Inc.',
    'AMD': 'Advanced Micro Devices, Inc.',
    'AMZN': 'Amazon.com, Inc.',
```

```python
    'BIDU': 'Baidu, Inc.ADS',
    'CSCO': 'Cisco Systems, Inc.(DE)',
    'FB': 'Facebook, Inc.',
    'GOOG': 'Alphabet Inc.Class C Capital Stock',
    'GOOGL': 'Alphabet Inc.',
    'JD': 'JD.com, Inc.',
    'NFLX': 'Netflix, Inc.',
    'NVDA': 'NVIDIA Corporation',
    'PYPL': 'PayPal Holdings, Inc.',
    'QCOM': 'QUALCOMM Incorporated',
    'TSLA': 'Tesla, Inc.',
    'ZM': 'Zoom Video Communications, Inc.'
}

# stock_list = ['AAPL','GOOG','AMZN','FB','TSLA','JD','AMD','NVDA','ZM','CSCO']
stock_list = ['AAPL','AMZN','GOOG','FB','JD','AMD']

def stock_show(stock_name):
    print(stock_name, end=" ")
    print(" stock is %s" %stock_dict[stock_name])

stock_show("AAPL")

# 1. Get all the dateframe
def generate_company_model():
    epochs = 50
    verbose=0
    validation_split = 0.2
    rmse_list_all = []
    stock_list = ['AAPL','TSLA','GOOG','FB','JD','CSCO']

    for company in stock_list:
        rmse_list_tmp = []
        print("=======Generate Model for>>>", company, "<<<<<=======")
        com_df = generate_company_set_date(company, 2010, 1, 1)
        # 1. Prepare the data frame
        X, y = filter_x_y(com_df)
        scaler = MinMaxScaler()
        X = scaler.fit_transform(np.array(X))
        X_, y_ = generate_from_data(X, np.array(y), 50, 1)
        X_train, X_test, y_train, y_test = split_dataset_test_train(X_, y_, 0.8)
        train_size = len(X_train)

        # 2. Evaluate the data for SMA and save
        print("=======Calculating the SMA as baseline >>>", company, " <<<<<=======")
        com_df["SMA_50"] = com_df["Close"].rolling(50).mean()
        rmse_SMA = calculate_rmse(com_df.loc[train_size:, "Close"], com_df.loc[train_size:,"SMA_50"])
        print("=======SMA for  >>>", company, " is ", rmse_SMA,"<<<<<=======")
        rmse_list_tmp.append(rmse_SMA)

        # 3. Train the different company and save the model
        print("=======Train the Designed Model for >>>", company, " <<<<<=======")
        # Pre-defined parameter
        file_path_save = "./trained_model/" + company+"_best_model-50D.h5"
        callback_list = [
                keras.callbacks.EarlyStopping(monitor='val_loss', patience=2),
                keras.callbacks.ModelCheckpoint(filepath=file_path_save,
                                                monitor='val_loss',
                                                save_best_only=True),
                tf.keras.callbacks.TensorBoard(log_dir='./log_dir')
        ]
        X_train_step = 50
        feature = 5
        print("======= X_train shape", X_train.shape, "=======")
        print("======= Y_train shape", y_train.shape, "=======")
        model = get_LSTM_GRU_model(X_train_step ,feature)
        history = model.fit(X_train,
                            y_train,
                            epochs=epochs,
                            callbacks=callback_list,
                            batch_size=32,
                            verbose=verbose, validation_split=validation_split)
        # 4. Draw the plot
        # draw_train_from_history(history)
        # Because we use the EarlyStopping callback, the history is the optimizal
        print("=======Prediction Test for Model with >>>", company, " <<<<<=======")
        # Demo the prediction
        y_predict = model.predict(X_test, verbose=2)
        y_predict_pad = np.zeros((y_predict.shape[0],5))
        y_predict_pad[:,0] = y_predict[:,0]
```

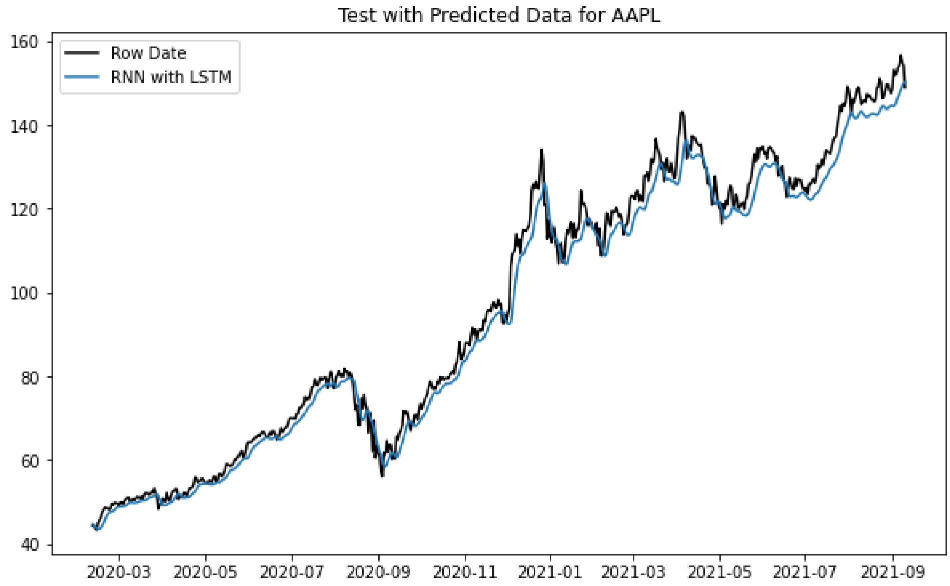
```
        y_test_pad = np.zeros((y_test.shape[0],5))
        y_test_pad[:,0] = y_test[:,0]
        y_test_pad = scaler.inverse_transform(y_test_pad)
        y_predict_pad = scaler.inverse_transform(y_predict_pad)
        rmse_model = calculate_rmse(y_test_pad[:,0], y_predict_pad[:,0])

        print("=======Current MSE for trained model is", rmse_model)
        success = rmse_model < rmse_SMA
        rmse_list_tmp.append(rmse_model)
        rmse_list_tmp.append(success)
        print("=======Current Model wins ? >>> ", success)

        plt.figure(figsize=(10,6))
        testSize = len(X_test)
        plt.title("Test with Predicted Data for "+ company)
        plt.plot(pd.date_range(end='2021-09-10', periods=testSize, freq='D'), y_test_pad[:,0], 'k', label='Row Date')
        plt.plot(pd.date_range(end='2021-09-10', periods=testSize, freq='D'), y_predict_pad[:,0], label='RNN with LSTM')
        plt.legend()
        rmse_list_all.append(rmse_list_tmp)


    # Write the result in a CSV file
    with open('company_list_result.csv', 'w') as file:
        writer = csv.writer(file)
        writer.writerows(rmse_list_all)


# generate_company_model()
```

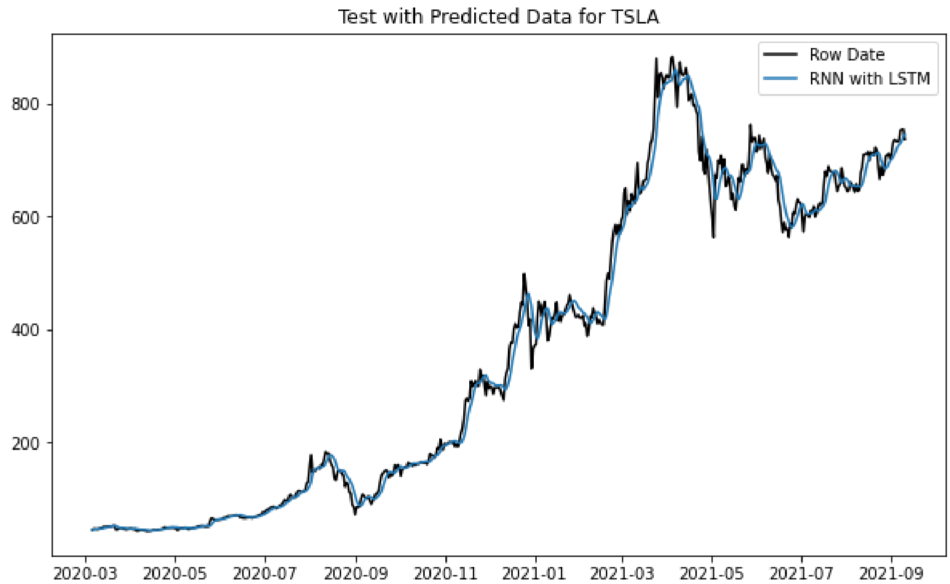
```
AAPL  stock is Apple Inc.
=======Generate Model for>>> AAPL <<<<<=======
>>>>>>>>>>In split_sequence\n (2893, 50, 5)
====Generate X_ shape==== (2893, 50, 5)
====Generate Y_ shape==== (2893, 1)
=======Calculating the SMA as baseline >>> AAPL  <<<<<=======
=======SMA for  >>> AAPL  is  8.009989523698845 <<<<<=======
=======Train the Designed Model for >>> AAPL  <<<<<=======
======= X_train shape (2315, 50, 5) =======
======= Y_train shape (2315, 1) =======
WARNING:tensorflow:Layer gru_43 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Model: "sequential_41"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_43 (GRU)                (None, 50, 128)           51840

 dropout_29 (Dropout)        (None, 50, 128)           0

 gru_44 (GRU)                (None, 64)                37248

 dropout_30 (Dropout)        (None, 64)                0

 dense_63 (Dense)            (None, 16)                1040

 dense_64 (Dense)            (None, 1)                 17

=================================================================
Total params: 90,145
Trainable params: 90,145
Non-trainable params: 0
_________________________________________________________________
=======Prediction Test for Model with >>> AAPL  <<<<<=======
19/19 - 2s - 2s/epoch - 82ms/step
=======Current MSE for trained model is 3.9865294257531096
=======Current Model wins ? >>>  True
=======Generate Model for>>> TSLA <<<<<=======
>>>>>>>>>>In split_sequence\n (2771, 50, 5)
====Generate X_ shape==== (2771, 50, 5)
====Generate Y_ shape==== (2771, 1)
=======Calculating the SMA as baseline >>> TSLA  <<<<<=======
=======SMA for  >>> TSLA  is  33.109758319691785 <<<<<=======
=======Train the Designed Model for >>> TSLA  <<<<<=======
======= X_train shape (2217, 50, 5) =======
======= Y_train shape (2217, 1) =======
WARNING:tensorflow:Layer gru_45 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Model: "sequential_42"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

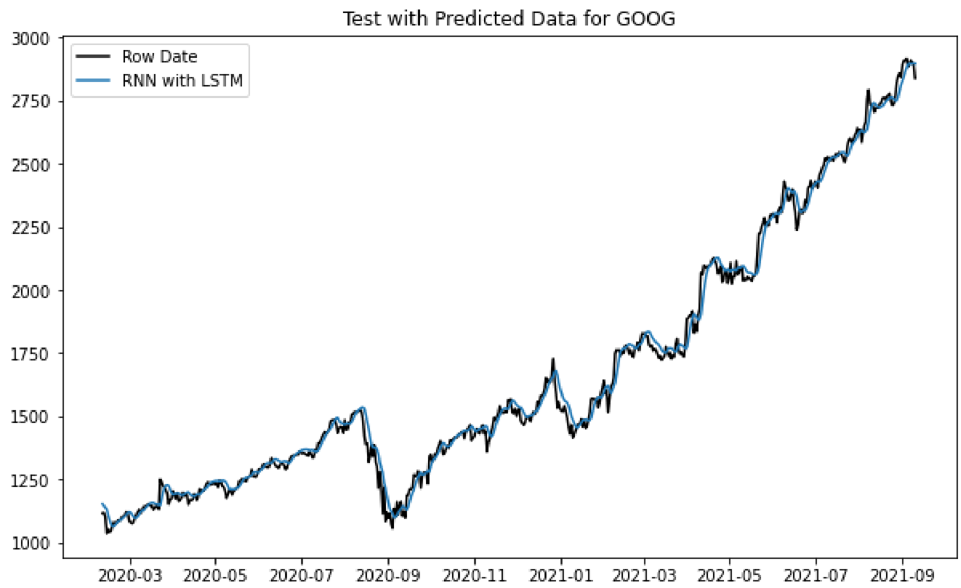
```
=================================================================
 gru_45 (GRU)                (None, 50, 128)           51840

 dropout_31 (Dropout)        (None, 50, 128)           0

 gru_46 (GRU)                (None, 64)                37248

 dropout_32 (Dropout)        (None, 64)                0

 dense_65 (Dense)            (None, 16)                1040

 dense_66 (Dense)            (None, 1)                 17

=================================================================
Total params: 90,145
Trainable params: 90,145
Non-trainable params: 0
_________________________________________________________________
=======Prediction Test for Model with >>> TSLA  <<<<<=======
18/18 - 1s - 1s/epoch - 75ms/step
=======Current MSE for trained model is 23.90907994535828
=======Current Model wins ? >>>  True
=======Generate Model for>>> GOOG <<<<<=======
>>>>>>>>>>In split_sequence\n (2893, 50, 5)
====Generate X_ shape==== (2893, 50, 5)
====Generate Y_ shape==== (2893, 1)
=======Calculating the SMA as baseline >>> GOOG  <<<<<=======
=======SMA for  >>> GOOG  is  63.23436154248993 <<<<<=======
=======Train the Designed Model for >>> GOOG  <<<<<=======
======= X_train shape (2315, 50, 5) =======
======= Y_train shape (2315, 1) =======
WARNING:tensorflow:Layer gru_47 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU
kernel as fallback when running on GPU.
Model: "sequential_43"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_47 (GRU)                (None, 50, 128)           51840

 dropout_33 (Dropout)        (None, 50, 128)           0

 gru_48 (GRU)                (None, 64)                37248

 dropout_34 (Dropout)        (None, 64)                0

 dense_67 (Dense)            (None, 16)                1040

 dense_68 (Dense)            (None, 1)                 17

=================================================================
Total params: 90,145
Trainable params: 90,145
Non-trainable params: 0
_________________________________________________________________
=======Prediction Test for Model with >>> GOOG  <<<<<=======
19/19 - 1s - 1s/epoch - 76ms/step
=======Current MSE for trained model is 38.743310527697304
=======Current Model wins ? >>>  True
=======Generate Model for>>> FB <<<<<=======
>>>>>>>>>>In split_sequence\n (2294, 50, 5)
====Generate X_ shape==== (2294, 50, 5)
====Generate Y_ shape==== (2294, 1)
=======Calculating the SMA as baseline >>> FB  <<<<<=======
=======SMA for  >>> FB  is  11.51772971115935 <<<<<=======
=======Train the Designed Model for >>> FB  <<<<<=======
======= X_train shape (1836, 50, 5) =======
======= Y_train shape (1836, 1) =======
WARNING:tensorflow:Layer gru_49 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU
kernel as fallback when running on GPU.
Model: "sequential_44"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_49 (GRU)                (None, 50, 128)           51840

 dropout_35 (Dropout)        (None, 50, 128)           0

 gru_50 (GRU)                (None, 64)                37248

 dropout_36 (Dropout)        (None, 64)                0
```

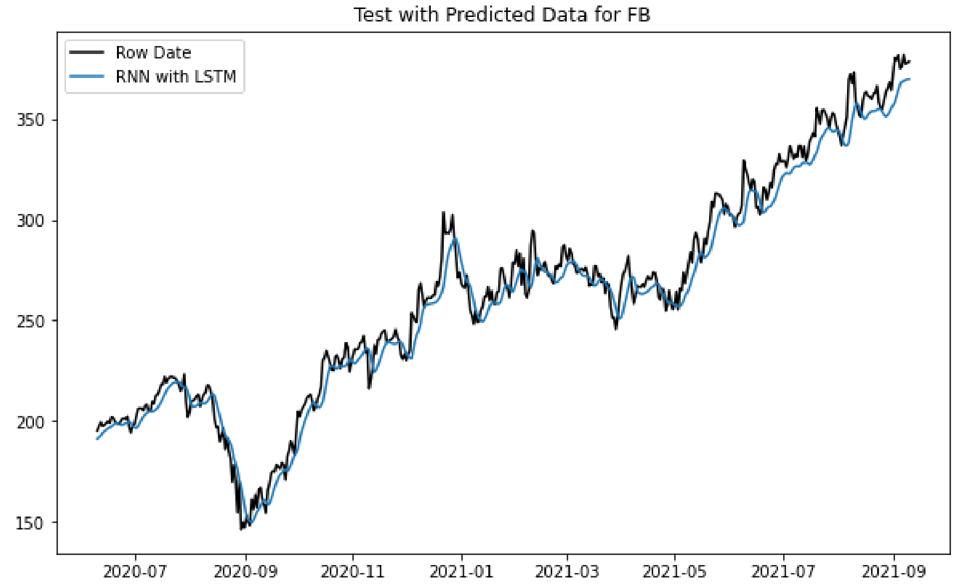
```
 dense_69 (Dense)            (None, 16)                1040

 dense_70 (Dense)            (None, 1)                 17

=================================================================
Total params: 90,145
Trainable params: 90,145
Non-trainable params: 0
_________________________________________________________________
=======Prediction Test for Model with >>> FB  <<<<<=======
15/15 - 1s - 1s/epoch - 91ms/step
=======Current MSE for trained model is 9.236320220675012
=======Current Model wins ? >>>  True
=======Generate Model for>>> JD <<<<<=======
>>>>>>>>>>In split_sequence\n (1790, 50, 5)
====Generate X_ shape==== (1790, 50, 5)
====Generate Y_ shape==== (1790, 1)
=======Calculating the SMA as baseline >>> JD  <<<<<=======
=======SMA for  >>> JD  is  4.176060620893943 <<<<<=======
=======Train the Designed Model for >>> JD  <<<<<=======
======= X_train shape (1432, 50, 5) =======
======= Y_train shape (1432, 1) =======
WARNING:tensorflow:Layer gru_51 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Model: "sequential_45"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_51 (GRU)                (None, 50, 128)           51840

 dropout_37 (Dropout)        (None, 50, 128)           0

 gru_52 (GRU)                (None, 64)                37248

 dropout_38 (Dropout)        (None, 64)                0

 dense_71 (Dense)            (None, 16)                1040

 dense_72 (Dense)            (None, 1)                 17

=================================================================
Total params: 90,145
Trainable params: 90,145
Non-trainable params: 0
_________________________________________________________________
=======Prediction Test for Model with >>> JD  <<<<<=======
12/12 - 1s - 1s/epoch - 93ms/step
=======Current MSE for trained model is 2.9785180923004035
=======Current Model wins ? >>>  True
=======Generate Model for>>> CSCO <<<<<=======
>>>>>>>>>>In split_sequence\n (2893, 50, 5)
====Generate X_ shape==== (2893, 50, 5)
====Generate Y_ shape==== (2893, 1)
=======Calculating the SMA as baseline >>> CSCO  <<<<<=======
=======SMA for  >>> CSCO  is  1.9939036919424655 <<<<<=======
=======Train the Designed Model for >>> CSCO  <<<<<=======
======= X_train shape (2315, 50, 5) =======
======= Y_train shape (2315, 1) =======
WARNING:tensorflow:Layer gru_53 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Model: "sequential_46"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_53 (GRU)                (None, 50, 128)           51840

 dropout_39 (Dropout)        (None, 50, 128)           0

 gru_54 (GRU)                (None, 64)                37248

 dropout_40 (Dropout)        (None, 64)                0

 dense_73 (Dense)            (None, 16)                1040

 dense_74 (Dense)            (None, 1)                 17

=================================================================
Total params: 90,145
Trainable params: 90,145
```

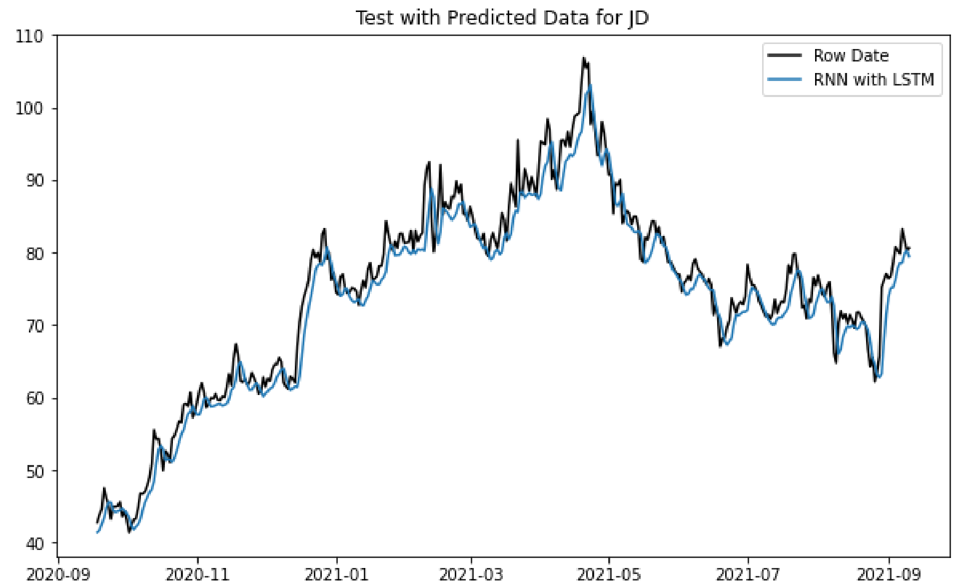
```
Non-trainable params: 0
_________________________________________________________________
=======Prediction Test for Model with >>> CSCO  <<<<<=======
19/19 - 1s - 1s/epoch - 76ms/step
=======Current MSE for trained model is 1.6597573500885865
=======Current Model wins ? >>>  True
```

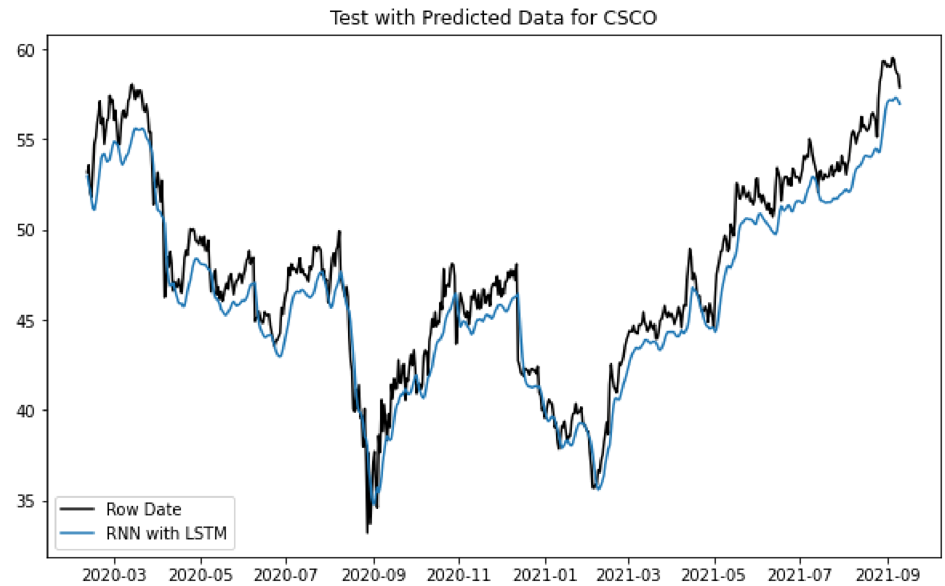







Test with Predicted Data for FB
Row Date
RNN with LSTM
350
300
250
200
150
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09


Test with Predicted Data for JD
Row Date
RNN with LSTM
110
100
90
80
70
60
50
40
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09


Test with Predicted Data for CSCO
Row Date
RNN with LSTM
60
55
50
45
40
35
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09