This program uses an artificial recurrent neural network called Long Short Term Memory (LSTM)

To predict the closing stock price of a corporation (Facebook, Apple, Amazon, Netflix, Google) using the 1 years ago stock price.

Import the Libraries

In [1]:

```
# Import the Libraries
import pandas as pd
import numpy as np

# To draw a diagram
import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")
%matplotlib inline

# To draw a hit map
import seaborn as sns
sns.set_style('whitegrid')

# To get information from Yahoo
import yfinance as yf
from yahoo_fin import stock_info as si #To get an instant price


# For time stamps
from datetime import datetime

# For the LSTM
import math
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

Download stock data from yahoo then export as CSV

In [2]:

```
# Set up End and Start times for data grab
end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

# Download stock data then export as CSV
df_fb = yf.download("FB", start, end)
df_fb.to_csv('facebook.csv')

df_aapl = yf.download("AAPL", start, end)
df_aapl.to_csv('apple.csv')

df_amzn = yf.download("AMZN", start, end)
df_amzn.to_csv('amazon.csv')

df_nflx = yf.download("NFLX", start, end)
df_nflx.to_csv('netflix.csv')

df_goog = yf.download("GOOG", start, end)
df_goog.to_csv('google.csv')
```

```
[*********************100%***********************]  1 of 1 compl
eted
[*********************100%***********************]  1 of 1 compl
eted
[*********************100%***********************]  1 of 1 compl
eted
[*********************100%***********************]  1 of 1 compl
eted
[*********************100%***********************]  1 of 1 compl
eted
```

Read dataset

In [3]:

```
# read google dataset
df = pd.read_csv('google.csv')
df.head()
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Vol
0	2021-06-21	2514.800049	2540.735107	2502.685059	2529.100098	2529.100098	131:
1	2021-06-22	2529.000000	2545.399902	2520.530029	2539.989990	2539.989990	104!
2	2021-06-23	2531.000000	2555.919922	2525.040039	2529.229980	2529.229980	98
3	2021-06-24	2541.070068	2550.709961	2539.199951	2545.639893	2545.639893	94
4	2021-06-25	2539.139893	2550.100098	2528.879883	2539.899902	2539.899902	167

Analyze the closing prices from dataset:

In [4]:

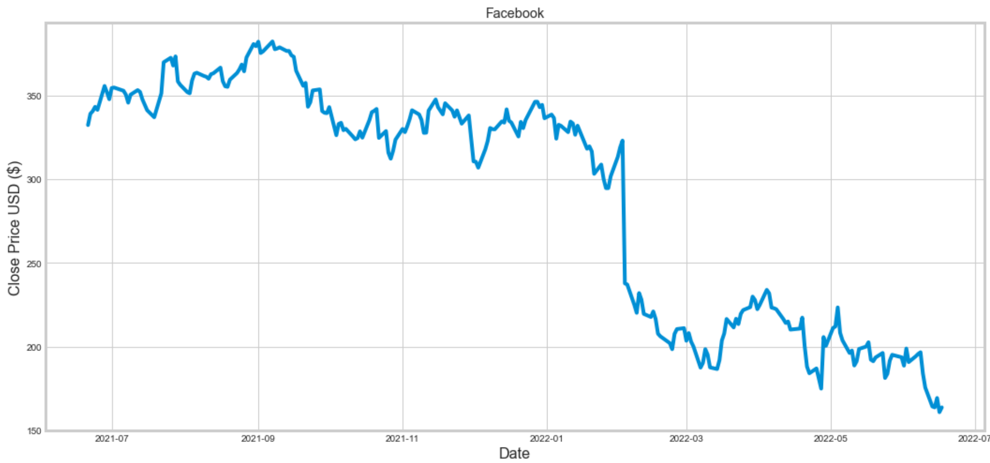
```
# Plot stock company['Close']
def figure_close(stockname , name):
    df = pd.read_csv(stockname)
    df["Date"] = pd.to_datetime(df.Date,format="%Y-%m-%d")
    df.index = df['Date']

    fig = plt.figure(figsize=(16,8))
    ax = fig.add_subplot()
    ax.set_title(name)
    ax.set_xlabel('Date', fontsize=16)
    ax.set_ylabel('Close Price USD ($)', fontsize=16)
    plt.plot(df["Close"],label='Close Price history')
```

Facebook

In [5]:

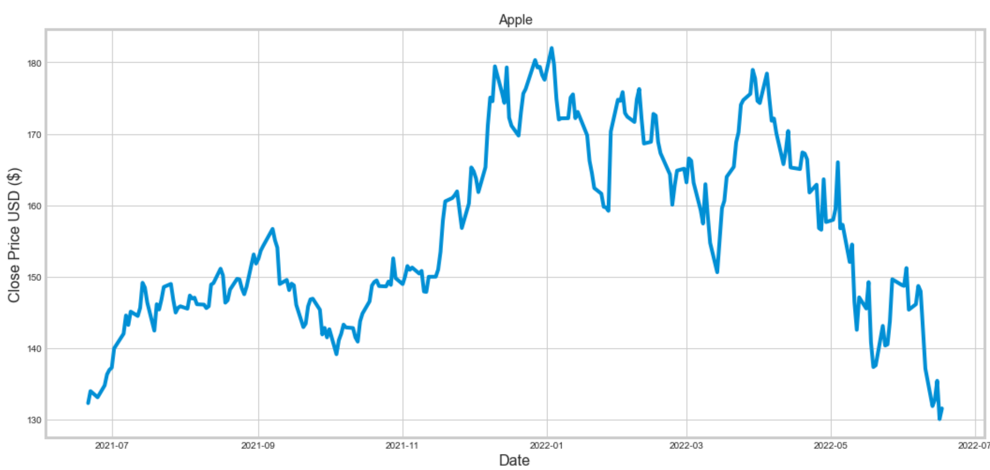
```
figure_close('facebook.csv' , 'Facebook')
```



Apple

In [6]:

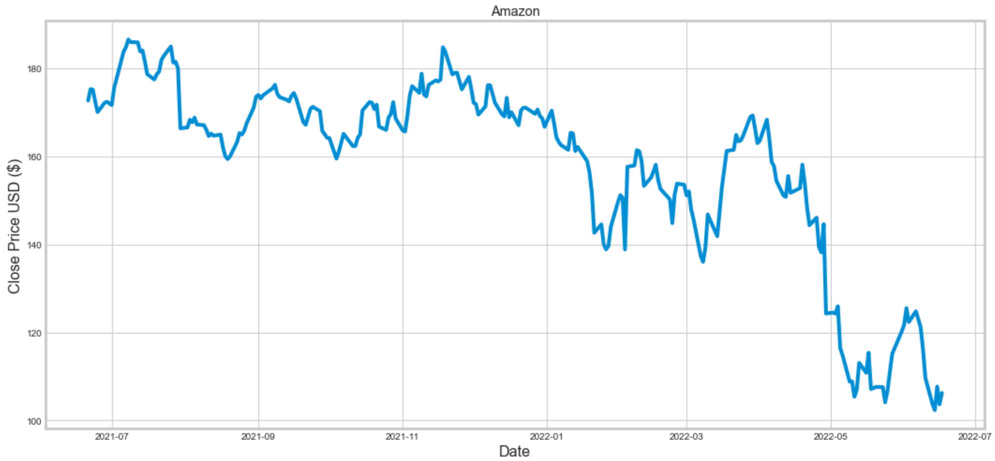
```
figure_close('apple.csv' , 'Apple')
```



Amazon

In [7]:

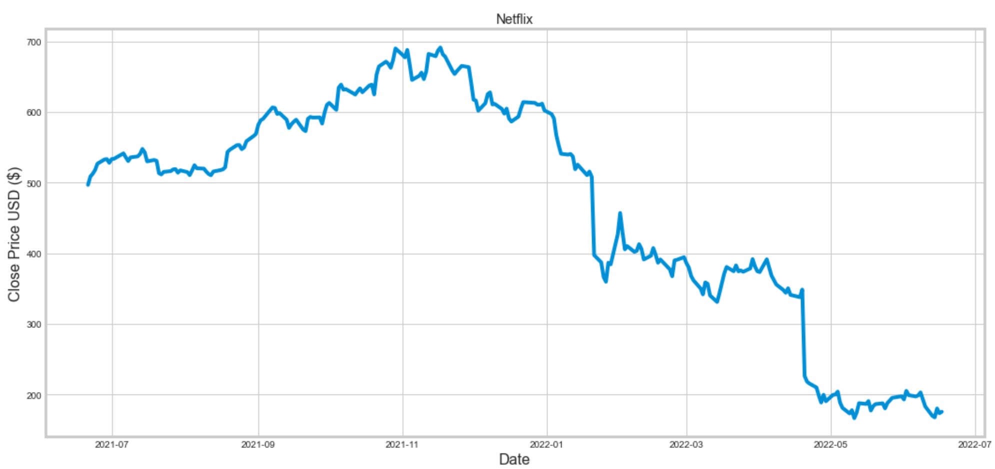
```
figure_close('amazon.csv' , 'Amazon')
```



Netflix

In [8]:

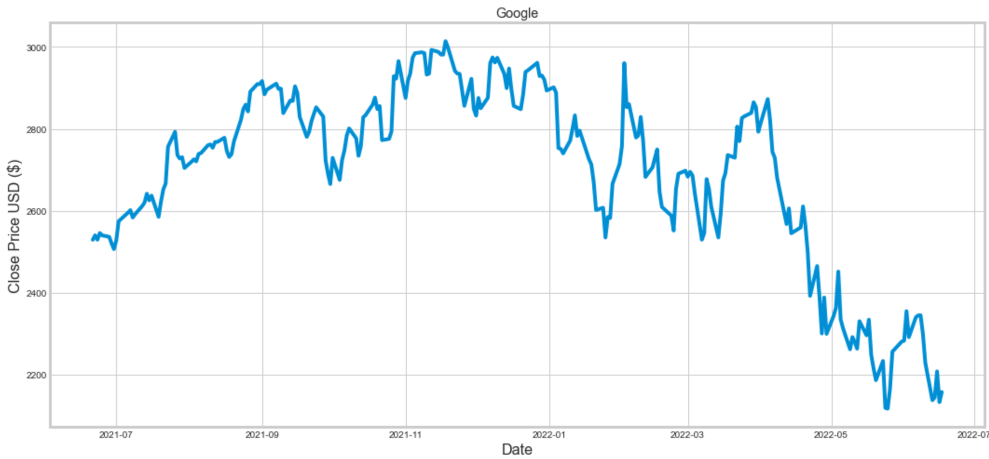
```
figure_close('netflix.csv' , 'Netflix')
```



Google

In [9]:

```
figure_close('google.csv' , 'Google')
```



The total volume of stock being traded each day:

In [10]:

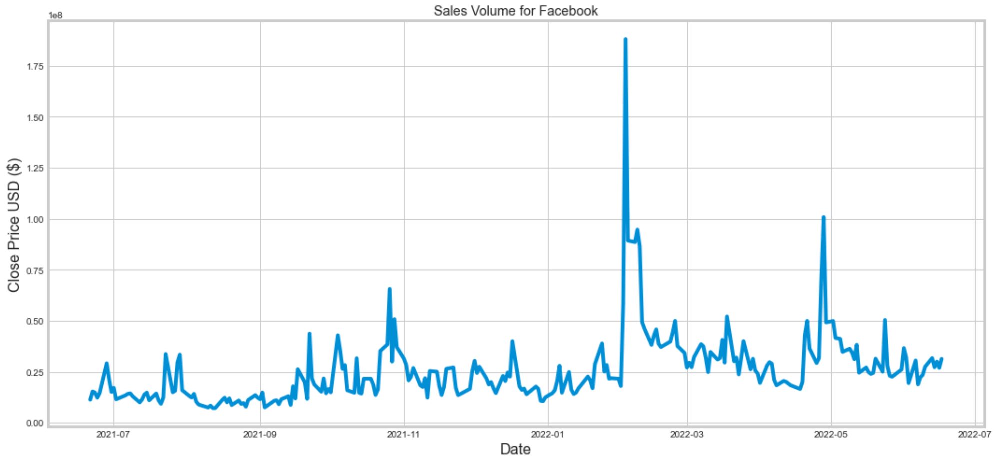
```
# Plot stock compony['Volume']
def figure_Sales(stockname , name):
    df = pd.read_csv(stockname)
    df["Date"] = pd.to_datetime(df.Date,format="%Y-%m-%d")
    df.index = df['Date']

    fig = plt.figure(figsize=(16,8))
    ax = fig.add_subplot()
    ax.set_title(name)
    ax.set_xlabel('Date', fontsize=16)
    ax.set_ylabel('Close Price USD ($)', fontsize=16)
    plt.plot(df["Volume"],label='Close Price history')
```

Facebook

In [11]:

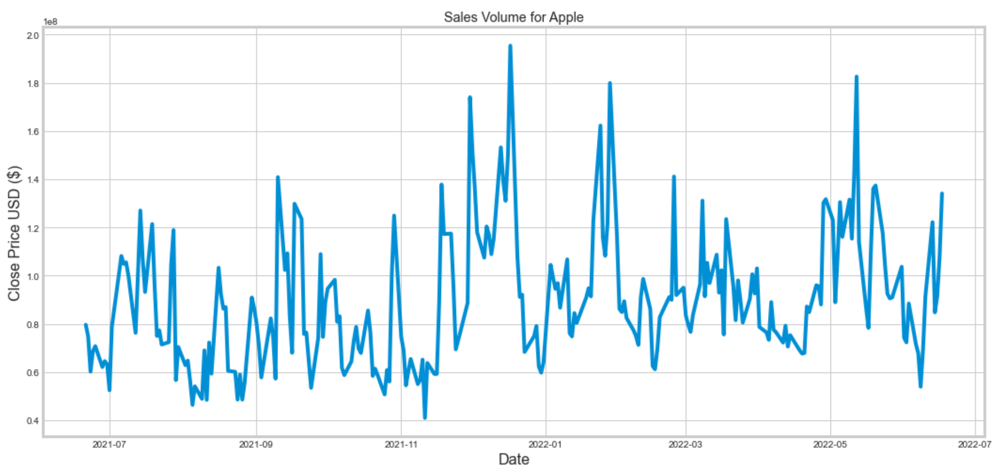
```
figure_Sales('facebook.csv' , 'Sales Volume for Facebook')
```



Apple

In [12]:

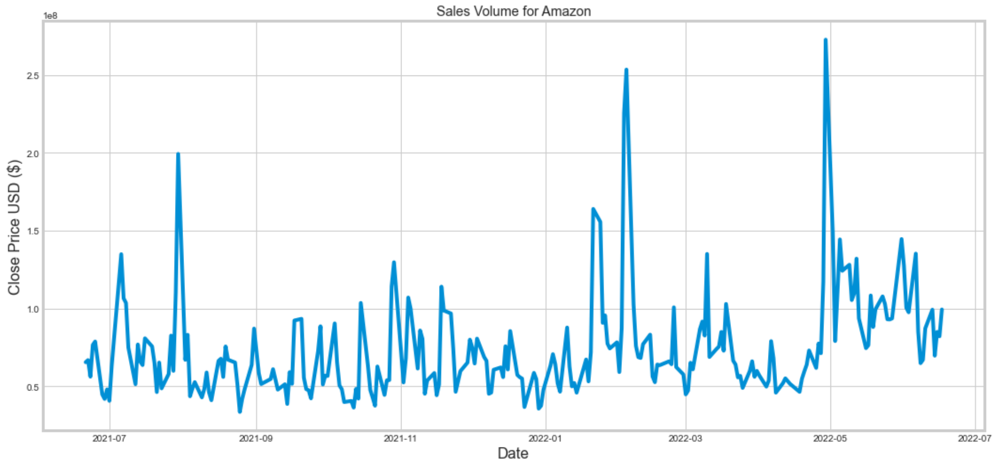
```
figure_Sales('apple.csv' , 'Sales Volume for Apple')
```



Amazon

In [13]:

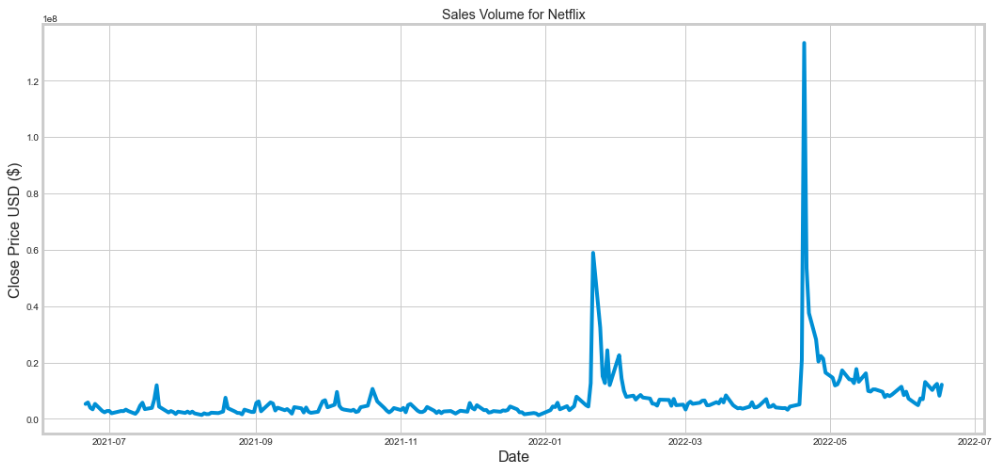
```
figure_Sales('amazon.csv' , 'Sales Volume for Amazon')
```



Netflix

In [14]:

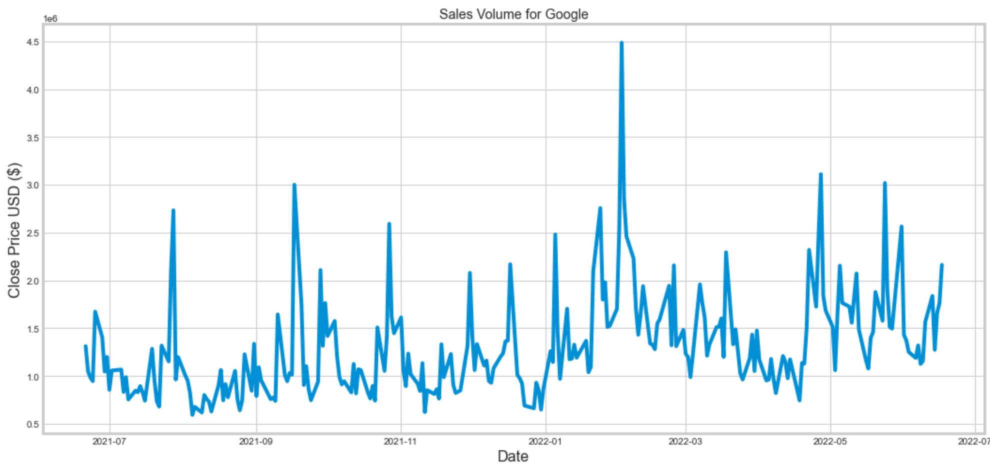
```
figure_Sales('netflix.csv' , 'Sales Volume for Netflix')
```



Google

In [15]:

```
figure_Sales('google.csv' , 'Sales Volume for Google')
```



Use sebron for a quick correlation plot for the daily returns

In [16]:

```
# The tech stocks we'll use for this analysis
tech_list = ['FB', 'AAPL', 'AMZN', 'NFLX', 'GOOG']

# Download stock data then export as CSV
df_close = yf.download(tech_list, start, end)['Adj Close']
df_close.to_csv('closing.csv')
```

```
[*********************100%***********************]  5 of 5 compl
eted
```

In [17]:

```
print(df_close.head())
```

```
                  AAPL        AMZN          FB         GOOG
NFLX
Date
2021-06-21  131.548447  172.697998  332.290009  2529.100098  49
7.000000
2021-06-22  133.218887  175.272003  339.029999  2539.989990  50
8.820007
2021-06-23  132.940475  175.190994  340.589996  2529.229980  51
2.739990
2021-06-24  132.652130  172.453995  343.179993  2545.639893  51
8.059998
2021-06-25  132.353851  170.072998  341.369995  2539.899902  52
7.070007
```

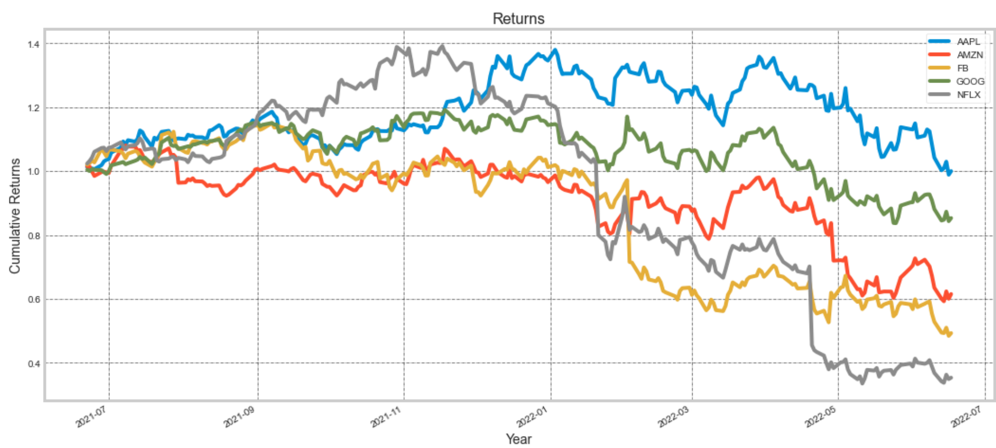
In [18]:

```
# Plot all the close prices
((df_close.pct_change()+1).cumprod()).plot(figsize=(16,8))

# Show the legend
plt.legend()

# Define the label for the title of the figure
plt.title("Returns", fontsize=16)
plt.ylabel('Cumulative Returns', fontsize=14)
plt.xlabel('Year', fontsize=14)

# Plot the grid lines
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



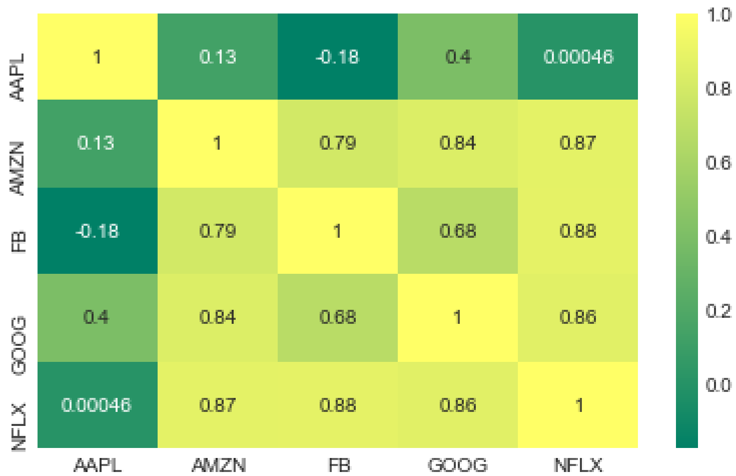
Heatmap

In [19]:

```
sns.heatmap(df_close.corr(), annot=True, cmap='summer')
```

Out[19]:

<AxesSubplot:>



Real-Time stock price

Last Price of Apple Inc. & Google Inc. & Microsoft Inc. & Tesla Inc.

In [20]:

```
# import stock_info module from yahoo_fin
def get_Price(stock , name):
    price = si.get_live_price(stock)
    print(name,"live stock price: " , price)


print(datetime.now())
print()
get_Price('fb' , 'Facebook Inc.')
get_Price('aapl' , 'Apple Inc.')
get_Price('amzn' , 'Amazon Inc.')
get_Price('nflx' , 'Netflix Inc.')
get_Price('goog' , 'Google Inc.')
```

```
2022-06-21 11:12:50.228410

Facebook Inc. live stock price:  163.74000549316406
Apple Inc. live stock price:  131.55999755859375
Amazon Inc. live stock price:  106.22000122070312
Netflix Inc. live stock price:  175.50999450683594
Google Inc. live stock price:  2157.31005859375
```

Predicting the closing price stock price with LSTM

Build and train the LSTM model

In [21]:

```
# For Warning
pd.options.mode.chained_assignment = None  # default='warn'

def LSTM_Model(stockname):
    # read the stock Price
    df = pd.read_csv(stockname)
    df["Date"] = pd.to_datetime(df.Date,format="%Y-%m-%d")
    df.index = df['Date']

    # Create a new dataframe with only the 'Close' column
    data = df.filter(['Close'])

    # Converting the dataframe to a numpy array
    dataset = data.values

    # Get /Compute the number of rows to train the model on
    training_data_len = math.ceil( len(dataset) *.8)

    # Scale the all of the data to be values between 0 and 1
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(dataset)

    # Create the scaled training data set
    train_data = scaled_data[0:training_data_len  , : ]

    # Split the data into x_train and y_train data sets
    x_train = []
    y_train = []

    for i in range(60,len(train_data)):
        x_train.append(train_data[i-60:i,0])
        y_train.append(train_data[i,0])

    # Convert x_train and y_train to numpy arrays
    x_train, y_train = np.array(x_train), np.array(y_train)

    # Reshape the data into the shape accepted by the LSTM
    x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))

    # Build the LSTM network model
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True,input_shape=(x_train.shape[1
    model.add(LSTM(units=50, return_sequences=False))
    model.add(Dense(units=25))
    model.add(Dense(units=1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)

# Test data set
test_data = scaled_data[training_data_len - 60: , : ]

# Create the x_test and y_test data sets
x_test = []


y_test =  dataset[training_data_len : , : ]

for i in range(60,len(test_data)):
    x_test.append(test_data[i-60:i,0])

# Convert x_test to a numpy array
x_test = np.array(x_test)

# Reshape the data into the shape accepted by the LSTM
x_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))

# Getting the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)#Undo scaling

# Calculate/Get the value of RMSE
rmse = np.sqrt(np.mean(((predictions- y_test)**2)))
print("rmse: ",rmse)

# Plot/Create the data for the graph
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions

#Visualize the data
plt.figure(figsize=(16,8))
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()
```

Facebook Stock Forecast

In [22]:

```
LSTM_Model('facebook.csv')
```

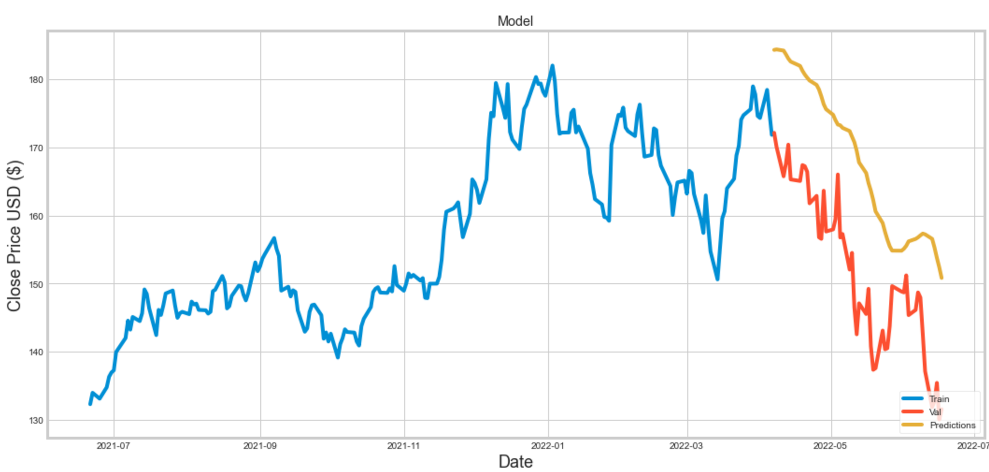
```
142/142 [==============================] - 18s 17ms/step - loss:
0.0196
rmse:  15.219372471843315
```

Apple Stock Forecast

In [23]:

```
LSTM_Model('apple.csv')
```

```
142/142 [==============================] - 5s 18ms/step - loss:
0.0290
rmse:  17.008633386448626
```

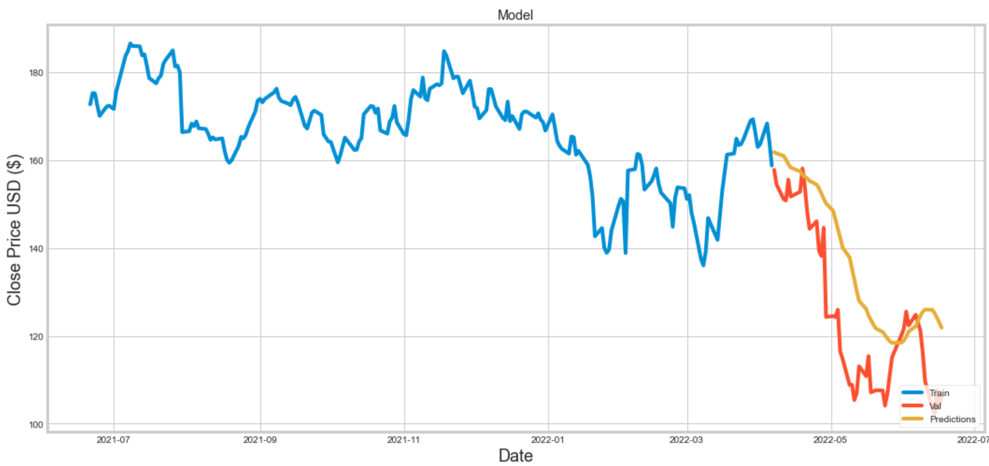


Amazon Stock Forecast

In [24]:

```
LSTM_Model('amazon.csv')
```

```
142/142 [==============================] - 5s 17ms/step - loss:
0.0376
rmse:  15.164336531756668
```

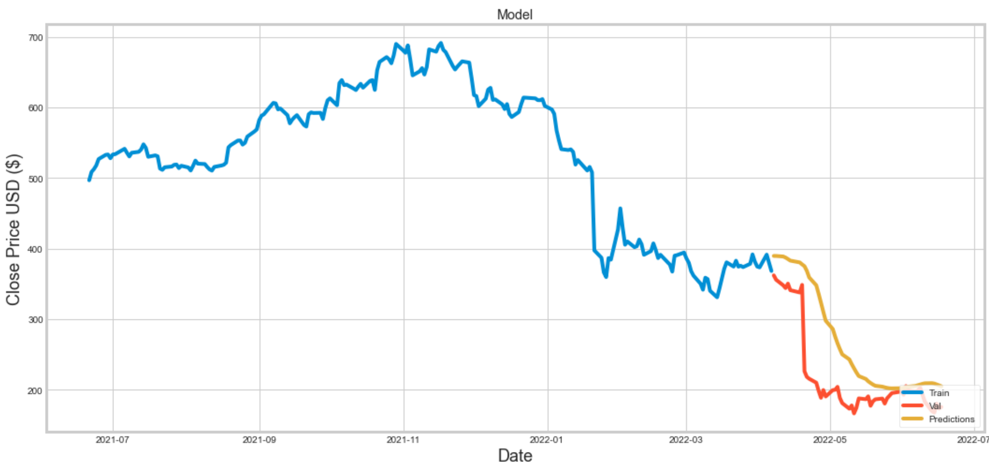


Netflix Stock Forecast

In [25]:

```
LSTM_Model('netflix.csv')
```

```
142/142 [==============================] - 5s 17ms/step - loss:
0.0633
rmse:  64.90316481646279
```



Google Stock Forecast

In [26]:

```
LSTM_Model('google.csv')
```

```
142/142 [==============================] - 5s 18ms/step - loss:
0.0697
WARNING:tensorflow:5 out of the last 9 calls to <function Model.
make_predict_function.<locals>.predict_function at 0x0000025365F
1B550> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @t
f.function repeatedly in a loop, (2) passing tensors with differ
ent shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For
(2), @tf.function has experimental_relax_shapes=True option that
relaxes argument shapes that can avoid unnecessary retracing. Fo
r (3), please refer to https://www.tensorflow.org/guide/function
#controlling_retracing (https://www.tensorflow.org/guide/functio
n#controlling_retracing) and https://www.tensorflow.org/api_doc
s/python/tf/function (https://www.tensorflow.org/api_docs/pytho
n/tf/function) for  more details.
rmse:  184.92513518135425
```