In [2]:

```
# Importing libraries
import math
import pandas as pd
import pandas_datareader as web
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

In [3]:

```
# Importing ML libraries
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

```
Using TensorFlow backend.
```

In [4]:

```
# Getting APPLE stock quote
apple_df = web.DataReader('AAPL', data_source = 'yahoo', start = '2010-01-01', end = '20
21-11-11')
apple_df
```

Out[4]:

	High	Low	Open	Close	Volume	Adj Close
Date						
2009-12-31	7.619643	7.520000	7.611786	7.526071	352410800.0	6.452590
2010-01-04	7.660714	7.585000	7.622500	7.643214	493729600.0	6.553025
2010-01-05	7.699643	7.616071	7.664286	7.656429	601904800.0	6.564354
2010-01-06	7.686786	7.526786	7.656429	7.534643	552160000.0	6.459940
2010-01-07	7.571429	7.466071	7.562500	7.520714	477131200.0	6.447996
...	...	...	...	...	...	...
2021-11-05	152.199997	150.059998	151.889999	151.279999	65414600.0	151.279999
2021-11-08	151.570007	150.160004	151.410004	150.440002	55020900.0	150.440002
2021-11-09	151.429993	150.059998	150.199997	150.809998	56787900.0	150.809998
2021-11-10	150.130005	147.850006	150.020004	147.919998	65187100.0	147.919998
2021-11-11	149.429993	147.679993	148.960007	147.869995	41000000.0	147.869995

2988 rows × 6 columns

In [5]:

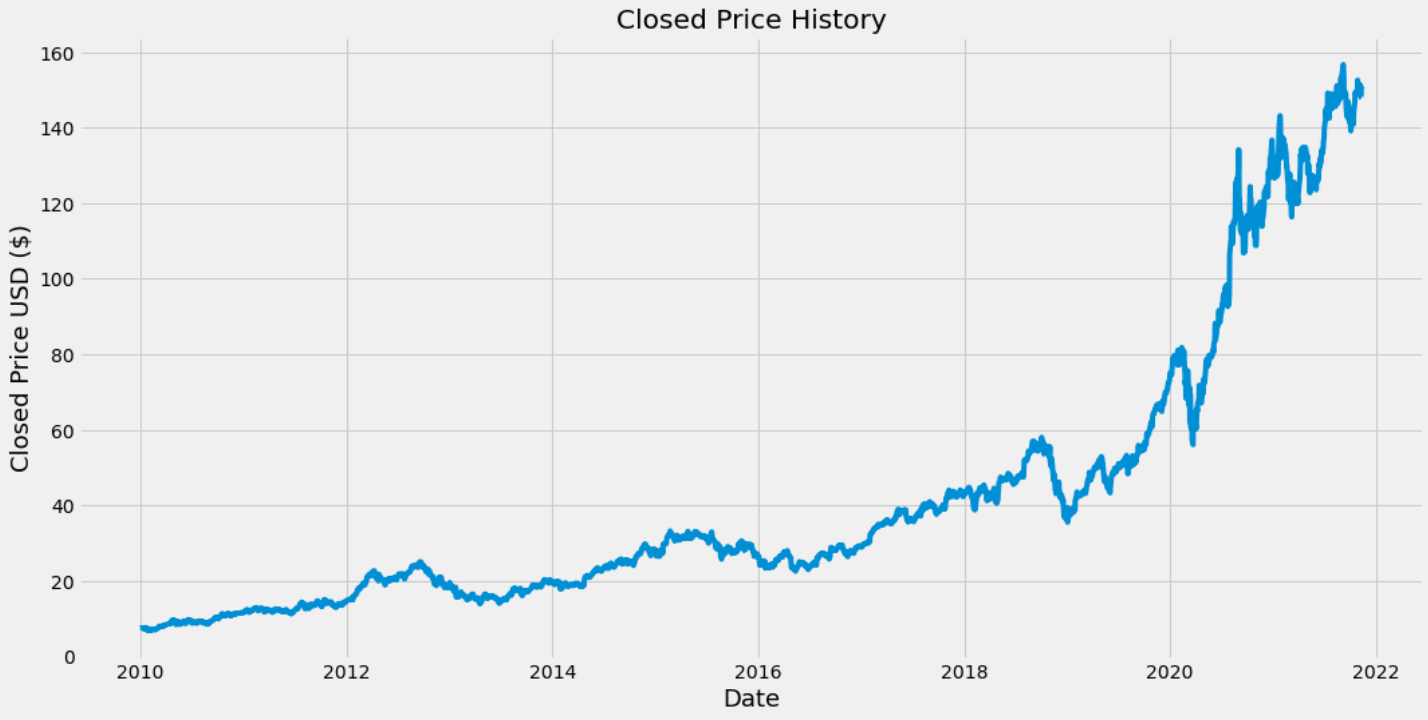
```
apple_df.shape
```

Out[5]:

```
(2988, 6)
```

In [6]:

```
# Closing Price History
plt.figure(figsize = (16, 8))
plt.title("Closed Price History")
plt.plot(apple_df['Close'])
plt.xlabel('Date', fontsize = 18)
plt.ylabel('Closed Price USD ($)', fontsize = 18)
plt.show()
```



In [8]:

```
# Create Closed Price dataframe
apple_closed = apple_df.filter(['Close'])
# Convert DataFrame to npArray
apple_closed_dataset = apple_closed.values
# Compute number of rows to train
training_data_length = math.ceil(len(apple_closed_dataset) * .8)

training_data_length
```

Out[8]:

```
2391
```

In [9]:

```
# Scaling the data
scaler = MinMaxScaler(feature_range = (0, 1))
scaled_apple_data = scaler.fit_transform(apple_closed_dataset)

scaled_apple_data
```

Out[9]:

```
array([[0.00445263],
       [0.00523446],
       [0.00532266],
       ...,
       [0.96075577],
       [0.94146739],
       [0.94113366]])
```

In [12]:

```
# Create Training DataSet
train_data = scaled_apple_data[0: training_data_length, :]
# Split the data: x_train and y_train
x_train = [] #Independent variables
y_train = [] #Target variables

for i in range(60, len(train_data)):
    x_train.append(train_data[i - 60: i, 0])
    y_train.append(train_data[i, 0])
    if i <= 61:
        print(x_train)
        print()
```

```
        print(y_train)
```

```
[array([4.45262721e-03, 5.23446206e-03, 5.32265864e-03, 4.50983899e-03,
       4.41687184e-03, 4.75058227e-03, 4.30484790e-03, 3.73515203e-03,
       4.43355768e-03, 4.14275689e-03, 3.30848401e-03, 5.47997590e-03,
       4.69099635e-03, 3.81857805e-03, 1.35866905e-03, 2.62676420e-03,
       3.31086771e-03, 3.77329428e-03, 1.72574797e-03, 2.38050884e-06,
       6.38810183e-04, 9.08160940e-04, 1.71144583e-03, 0.00000000e+00,
       8.12813286e-04, 4.93408194e-04, 9.86822754e-04, 7.31770963e-04,
       1.57796229e-03, 1.98556715e-03, 2.70542602e-03, 2.50281862e-03,
       2.59339253e-03, 2.29305378e-03, 1.99509873e-03, 1.19419754e-03,
       2.05231050e-03, 2.37171560e-03, 2.99622681e-03, 4.03788083e-03,
       4.00450915e-03, 4.11891679e-03, 4.44785983e-03, 6.41198012e-03,
       6.44296811e-03, 7.38212022e-03, 7.81593931e-03, 7.97325657e-03,
       8.23546262e-03, 7.57757972e-03, 7.72297853e-03, 7.64431990e-03,
       7.77065236e-03, 7.19857917e-03, 7.79448927e-03, 8.65497957e-03,
       8.89572921e-03, 8.24737790e-03, 9.26043080e-03, 9.61558490e-03])]

[0.010440323006406994]
[array([4.45262721e-03, 5.23446206e-03, 5.32265864e-03, 4.50983899e-03,
       4.41687184e-03, 4.75058227e-03, 4.30484790e-03, 3.73515203e-03,
       4.43355768e-03, 4.14275689e-03, 3.30848401e-03, 5.47997590e-03,
       4.69099635e-03, 3.81857805e-03, 1.35866905e-03, 2.62676420e-03,
       3.31086771e-03, 3.77329428e-03, 1.72574797e-03, 2.38050884e-06,
       6.38810183e-04, 9.08160940e-04, 1.71144583e-03, 0.00000000e+00,
       8.12813286e-04, 4.93408194e-04, 9.86822754e-04, 7.31770963e-04,
       1.57796229e-03, 1.98556715e-03, 2.70542602e-03, 2.50281862e-03,
       2.59339253e-03, 2.29305378e-03, 1.99509873e-03, 1.19419754e-03,
       2.05231050e-03, 2.37171560e-03, 2.99622681e-03, 4.03788083e-03,
       4.00450915e-03, 4.11891679e-03, 4.44785983e-03, 6.41198012e-03,
       6.44296811e-03, 7.38212022e-03, 7.81593931e-03, 7.97325657e-03,
       8.23546262e-03, 7.57757972e-03, 7.72297853e-03, 7.64431990e-03,
       7.77065236e-03, 7.19857917e-03, 7.79448927e-03, 8.65497957e-03,
       8.89572921e-03, 8.24737790e-03, 9.26043080e-03, 9.61558490e-03]), array([5.2344620
6e-03, 5.32265864e-03, 4.50983899e-03, 4.41687184e-03,
       4.75058227e-03, 4.30484790e-03, 3.73515203e-03, 4.43355768e-03,
       4.14275689e-03, 3.30848401e-03, 5.47997590e-03, 4.69099635e-03,
       3.81857805e-03, 1.35866905e-03, 2.62676420e-03, 3.31086771e-03,
       3.77329428e-03, 1.72574797e-03, 2.38050884e-06, 6.38810183e-04,
       9.08160940e-04, 1.71144583e-03, 0.00000000e+00, 8.12813286e-04,
       4.93408194e-04, 9.86822754e-04, 7.31770963e-04, 1.57796229e-03,
       1.98556715e-03, 2.70542602e-03, 2.50281862e-03, 2.59339253e-03,
       2.29305378e-03, 1.99509873e-03, 1.19419754e-03, 2.05231050e-03,
       2.37171560e-03, 2.99622681e-03, 4.03788083e-03, 4.00450915e-03,
       4.11891679e-03, 4.44785983e-03, 6.41198012e-03, 6.44296811e-03,
       7.38212022e-03, 7.81593931e-03, 7.97325657e-03, 8.23546262e-03,
       7.57757972e-03, 7.72297853e-03, 7.64431990e-03, 7.77065236e-03,
       7.19857917e-03, 7.79448927e-03, 8.65497957e-03, 8.89572921e-03,
       8.24737790e-03, 9.26043080e-03, 9.61558490e-03, 1.04403230e-02])]

[0.010440323006406994, 0.010237712425207511]
```

In [13]:

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

In [15]:

```
# ReShape x_train dataset since LSTM model expects a 3-D dataset
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_train.shape
```

Out[15]:

```
(2331, 60, 1)
```

In [17]:

```
# Build LSTM Model
model = Sequential()
model.add(LSTM(50, return_sequences = True, input_shape = (x_train.shape[1], x_train.sha
```

```
pe[2])))
model.add(LSTM(50, return_sequences = False))
model.add(Dense(25))
model.add(Dense(1))
```

```
2021-11-19 18:24:22.647007: I tensorflow/core/platform/cpu_feature_guard.cc:145] This Ten
sorFlow binary is optimized with Intel(R) MKL-DNN to use the following CPU instructions i
n performance critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compile
r flags.
2021-11-19 18:24:22.652841: I tensorflow/core/common_runtime/process_util.cc:115] Creatin
g new thread pool with default inter op setting: 4. Tune using inter_op_parallelism_threa
ds for best performance.
```

In [18]:

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

In [19]:

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

```
Epoch 1/1
2331/2331 [==============================] - 499s 214ms/step - loss: 2.1830e-04
```

Out[19]:

```
<keras.callbacks.callbacks.History at 0x7fb7606250d0>
```

In [21]:

```
# Create Testing Dataset

# Create New Array containing scaled values from 2331 to  end of our dataset
test_data = scaled_apple_data[training_data_length - 60: , :]

# Create the testing dataset
x_test = []
y_test = apple_closed_dataset[training_data_length:, :] # All values that we want our mod
el to predict
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])
```

In [23]:

```
# Convert data to np array
x_test = np.array(x_test)
```

In [24]:

```
# Reshape our data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

In [25]:

```
# Get predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

In [26]:

```
# Evaluate our model

# Getting the Root Mean Squared Error:
rmse = np.sqrt(np.mean( predictions - y_test )**2)

rmse
```

Out[26]:

```
5.281946581612081
```

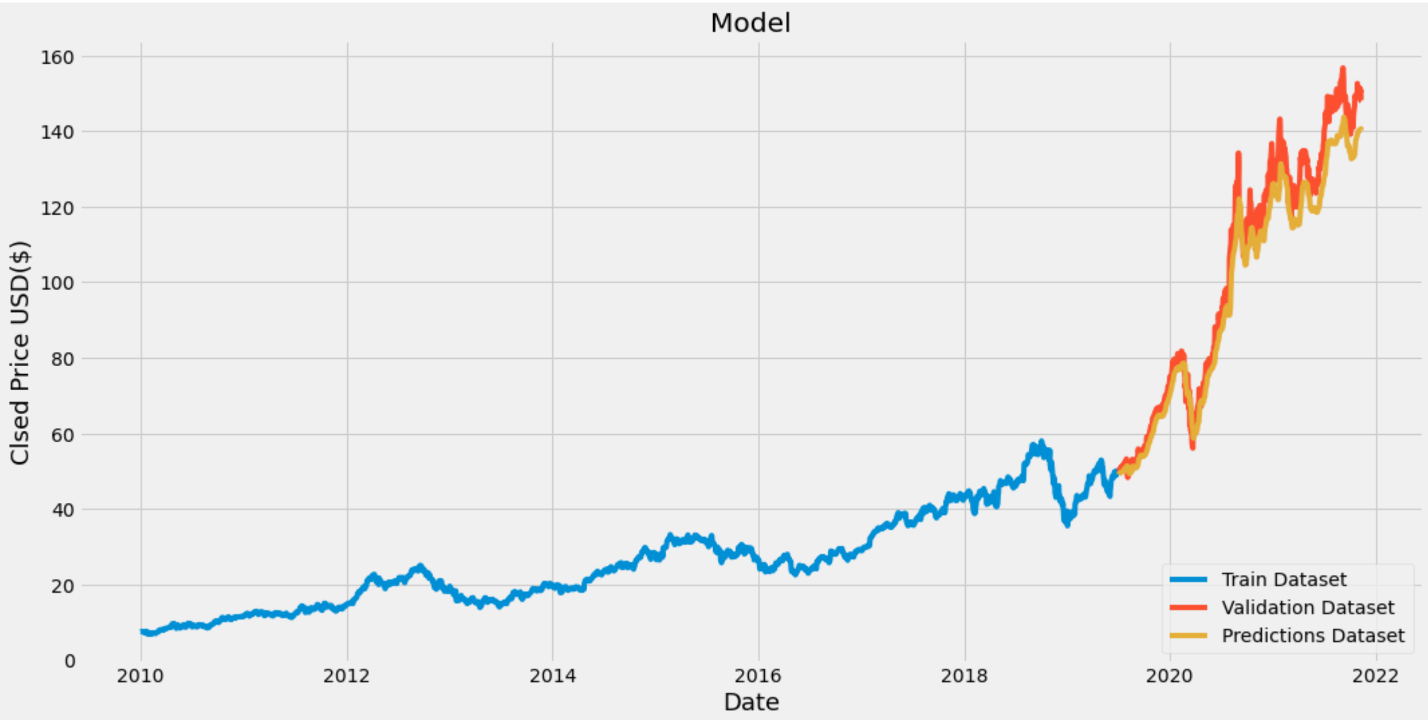
In [29]:

```
# Plot the data
train = apple_closed[0:training_data_length]
validation = apple_closed[training_data_length:]
validation['Predictions'] = predictions

# Visualizing the data
plt.figure(figsize = (16, 8))
plt.title('Model')
plt.xlabel('Date', fontsize = 18)
plt.ylabel('Clsed Price USD($)', fontsize = 18)
plt.plot(train['Close'])
plt.plot(validation[['Close', 'Predictions']])
plt.legend(['Train Dataset', 'Validation Dataset', 'Predictions Dataset'], loc = 'lower right')
plt.show()
```

```
/Users/rishabburman/opt/anaconda3/envs/Python37/lib/python3.7/site-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```



In [32]:

```
# Show the valid price vs predicted prices
validation
```

Out[32]:

Date	Close	Predictions
2019-07-03	51.102501	49.092457
2019-07-05	51.057499	49.324913
2019-07-08	50.005001	49.553692
2019-07-09	50.310001	49.616692
2019-07-10	50.807499	49.629822

Date	Close	Predictions
...		
2021-11-05	151.279999	140.306854
2021-11-08	150.440002	140.488739
2021-11-09	150.809998	140.530212
2021-11-10	147.919998	140.558578
2021-11-11	147.869995	140.178055

597 rows × 2 columns

In [35]:

```
# Predict closing prices for Apple on 18-11-2021
new_apple = web.DataReader('AAPL', data_source = 'yahoo', start = '2010-01-01', end = '2021-11-18')

new_apple_df = new_apple.filter(['Close'])

# Get last 60 days closing values and convert df to array
last_60_days = new_apple_df[-60:].values

# Scale data to values in range (0, 1)
last_60_days_scaled = scaler.transform(last_60_days)

# Create New Test List
X_test = []

# Append values to our list
X_test.append(last_60_days_scaled)

# Convert X_test dataset to numpy array
X_test = np.array(X_test)

# Reshape the data
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Get predicted scaled price
pred_price = model.predict(X_test)

# Undo the scaling
pred_price = scaler.inverse_transform(pred_price)

print(pred_price)
```

```
[[141.42715]]
```

In [44]:

```
# Quoted closing Price for Apple on 18-11-2021
new_apple_2 = web.DataReader('AAPL', data_source = 'yahoo', start = '2021-11-18', end = '2021-11-18')
print(new_apple_2['Close'][0])
```

```
153.49000549316406
```

In [43]:

```
new_apple_2['Close'][0]
```

Out[43]:

```
153.49000549316406
```

In [ ]:

```

```