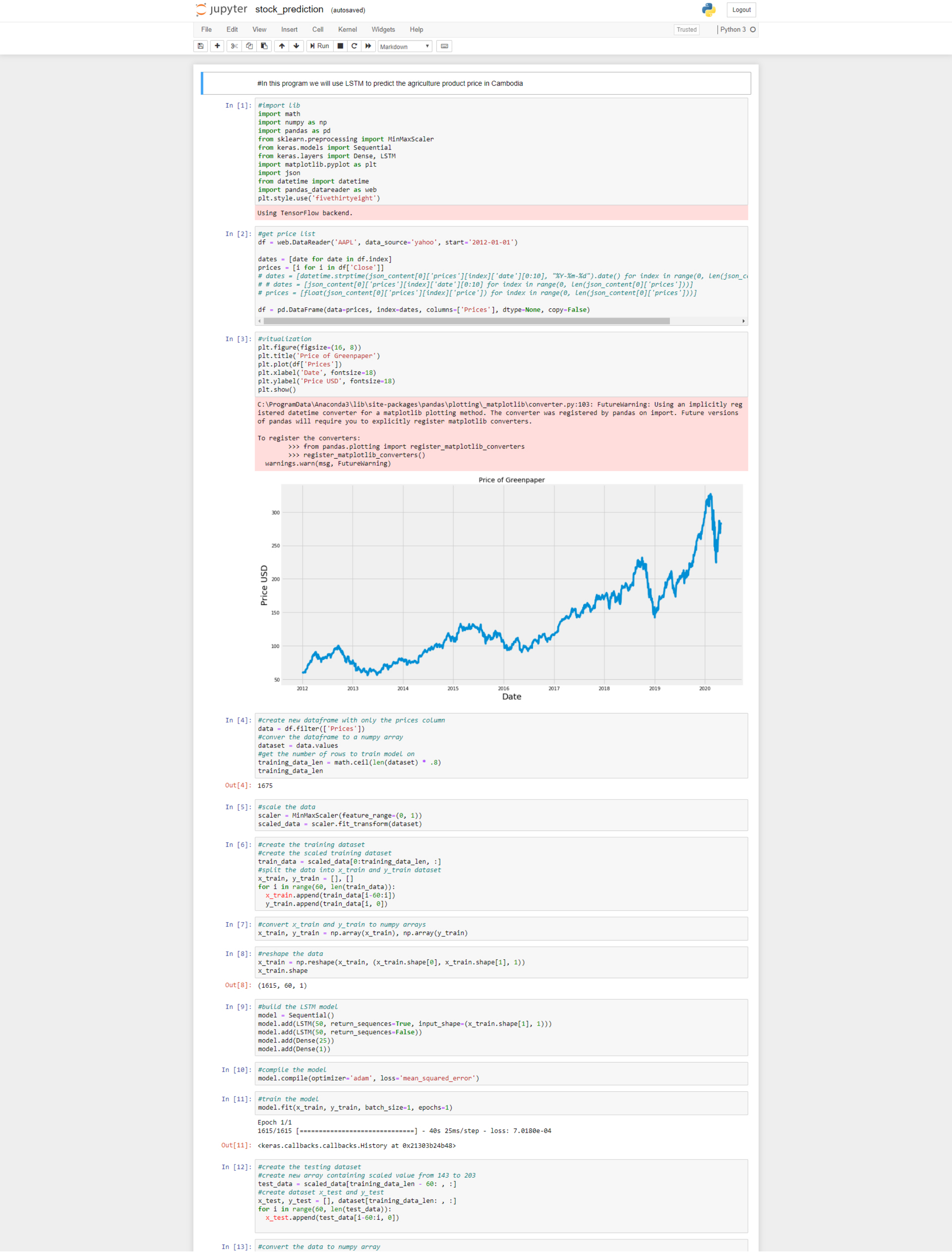


#In this program we will use LSTM to predict the agriculture product price in Cambodia

In [1]:
```
#import lib
import math
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
import json
from datetime import datetime
import pandas_datareader as web
plt.style.use('fivethirtyeight')
```

```
Using TensorFlow backend.
```

In [2]:
```
#get price list
df = web.DataReader('AAPL', data_source='yahoo', start='2012-01-01')

dates = [date for date in df.index]
prices = [i for i in df['Close']]
# dates = [datetime.strptime(json_content[0]['prices'][index]['date'][0:10], "%Y-%m-%d").date() for index in range(0, len(json_c
# # dates = [json_content[0]['prices'][index]['date'][0:10] for index in range(0, len(json_content[0]['prices']))]
# prices = [float(json_content[0]['prices'][index]['price']) for index in range(0, len(json_content[0]['prices']))]

df = pd.DataFrame(data=prices, index=dates, columns=['Prices'], dtype=None, copy=False)
```

In [3]:
```
#vitualization
plt.figure(figsize=(16, 8))
plt.title('Price of Greenpaper')
plt.plot(df['Prices'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Price USD', fontsize=18)
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\plotting\_matplotlib\converter.py:103: FutureWarning: Using an implicitly reg
istered datetime converter for a matplotlib plotting method. The converter was registered by pandas on import. Future versions
of pandas will require you to explicitly register matplotlib converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```

In [4]:
```
#create new dataframe with only the prices column
data = df.filter(['Prices'])
#conver the dataframe to a numpy array
dataset = data.values
#get the number of rows to train model on
training_data_len = math.ceil(len(dataset) * .8)
training_data_len
```

Out[4]:
```
1675
```

In [5]:
```
#scale the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)
```

In [6]:
```
#create the training dataset
#create the scaled training dataset
train_data = scaled_data[0:training_data_len, :]
#split the data into x_train and y_train dataset
x_train, y_train = [], []
for i in range(60, len(train_data)):
  x_train.append(train_data[i-60:i])
  y_train.append(train_data[i, 0])
```

In [7]:
```
#convert x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)
```

In [8]:
```
#reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_train.shape
```

Out[8]:
```
(1615, 60, 1)
```

In [9]:
```
#build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

In [10]:
```
#compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

In [11]:
```
#train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)
```

```
Epoch 1/1
1615/1615 [==============================] - 40s 25ms/step - loss: 7.0180e-04
```

Out[11]:
```
<keras.callbacks.callbacks.History at 0x21303b24b48>
```

In [12]:
```
#create the testing dataset
#create new array containing scaled value from 143 to 203
test_data = scaled_data[training_data_len - 60: , :]
#create dataset x_test and y_test
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])
```

In [13]:
```
#convert the data to numpy array
```

```
x_test = np.array(x_test)
```

In [14]:
```
#reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

In [15]:
```
#get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

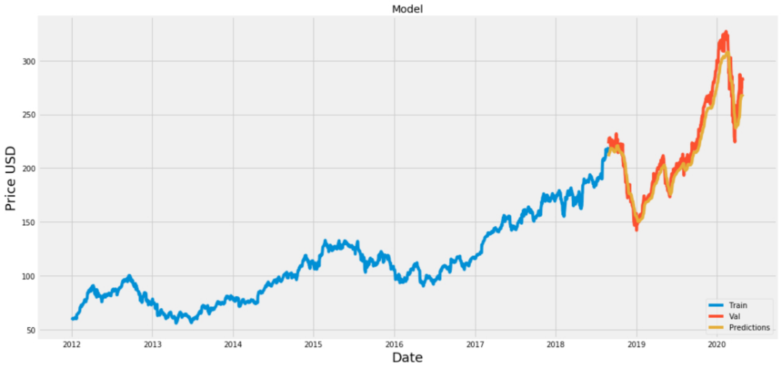
In [16]:
```
#get the root mean square error (RMSE)
rmse = np.sqrt(np.mean(((predictions - y_test)**2)))
rmse
```

Out[16]:
```
11.012116049137362
```

In [17]:
```
#plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
#vitualize the data
plt.figure(figsize=(16, 8))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Price USD', fontsize=18)
plt.plot(train['Prices'])
plt.plot(valid[['Prices', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\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: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy
  after removing the cwd from sys.path.
```



In [19]:
```
apple_quote = web.DataReader('AAPL', data_source='yahoo', start='2012-01-01')
#create new dataframe
new_df = apple_quote.filter(['Close'])
#get the last 60day closing price values and convert the dataframe to array
last_60_days = new_df[-60:].values
#scale the data to be values between 0 and 1
last_60_days_scaled = scaler.transform(last_60_days)
#create an empty list
X_test = []
#append the pass 60days
X_test.append(last_60_days_scaled)
#convert the X_test dataset to a 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 the predicted scaled price
pred_price = model.predict(X_test)
#undo the scaling
pred_price = scaler.inverse_transform(pred_price)
print(pred_price)
```

```
[[269.1614]]
```

In [20]:
```
valid
```

Out[20]:

	Prices	Predictions
2018-08-29	222.979996	211.020905
2018-08-30	225.029999	212.063187
2018-08-31	227.630005	213.344055
2018-09-04	228.360001	214.866028
2018-09-05	226.869995	216.431122
...	...	...
2020-04-22	276.100006	267.602081
2020-04-23	275.029999	267.622559
2020-04-24	282.970001	267.418945
2020-04-27	283.170013	267.859650
2020-04-28	281.459991	268.563995

418 rows × 2 columns