Time Series Modeling

Yousof Rahimian

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import statsmodels.api as sm
import itertools
```

Outline

- Plot the data with proper labeling and make some observations on the graph.
- Split this data into a training and test set. Use the last year of data (July 2020 – June 2021) of data as your test set and the rest as your training set.
- Use the training set to build a predictive model for the monthly retail sales.
- Use the model to predict the monthly retail sales on the last year of data.
- Report the RMSE of the model predictions on the test set.

Import and Clean Data

In [2]:
```
import pandas as pd
df = pd.read_csv('us_retail_sales.csv')
df.head()
```

Out[2]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	
0	1992	146925	147223	146805	148032	149010	149800	150761.0	151067.0	152588.0	15
1	1993	157555	156266	154752	158979	160605	160127	162816.0	162506.0	163258.0	16
2	1994	167518	169649	172766	173106	172329	174241	174781.0	177295.0	178787.0	18
3	1995	182413	179488	181013	181686	183536	186081	185431.0	186806.0	187366.0	18
4	1996	189135	192266	194029	194744	196205	196136	196187.0	196218.0	198859.0	20

In [3]:
```
df.columns.to_list()
```

Out[3]:
```
['YEAR',
 'JAN',
 'FEB',
 'MAR',
 'APR',
 'MAY',
 'JUN',
 'JUL',
 'AUG',
 'SEP',
 'OCT',
 'NOV',
 'DEC']
```

In [4]:

```
#Identify variables and it's attributes in the dataset
df.shape
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   YEAR    30 non-null     int64  
 1   JAN     30 non-null     int64  
 2   FEB     30 non-null     int64  
 3   MAR     30 non-null     int64  
 4   APR     30 non-null     int64  
 5   MAY     30 non-null     int64  
 6   JUN     30 non-null     int64  
 7   JUL     29 non-null     float64
 8   AUG     29 non-null     float64
 9   SEP     29 non-null     float64
 10  OCT     29 non-null     float64
 11  NOV     29 non-null     float64
 12  DEC     29 non-null     float64
dtypes: float64(6), int64(7)
memory usage: 3.2 KB
```

Out[4]:

	YEAR	JAN	FEB	MAR	APR	MA
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.0000
mean	2006.500000	304803.833333	305200.900000	307533.566667	306719.600000	309205.6333
std	8.803408	97687.399232	96682.043053	100002.422696	98207.161171	99541.0100
min	1992.000000	146925.000000	147223.000000	146805.000000	148032.000000	149010.0000
25%	1999.250000	228856.750000	231470.750000	233019.000000	233235.500000	234976.5000
50%	2006.500000	303486.000000	304592.500000	308655.500000	311233.500000	308690.0000
75%	2013.750000	371527.000000	377008.500000	379221.000000	376797.500000	382698.2500
max	2021.000000	520162.000000	504458.000000	559871.000000	562269.000000	548987.0000

In [5]:
```
# Check any missing values accros columns
df.isnull().any()
```

Out[5]:
```
YEAR     False
JAN      False
FEB      False
MAR      False
APR      False
MAY      False
JUN      False
JUL       True
AUG       True
SEP       True
OCT       True
NOV       True
DEC       True
dtype: bool
```

Reshaping the Data with Pandas' Melt Method '

In [6]:
```
# Reshape long data to a wide format
df = pd.melt(df, id_vars = 'YEAR', value_vars = df.iloc[:, 1:], var_name = 'M
df.head()
```

Out[6]:

	YEAR	Month	Sales
0	1992	JAN	146925.0
1	1993	JAN	157555.0
2	1994	JAN	167518.0
3	1995	JAN	182413.0
4	1996	JAN	189135.0

In [7]:
```
df.tail()
```

Out[7]:

	YEAR	Month	Sales
355	2017	DEC	433282.0
356	2018	DEC	434803.0
357	2019	DEC	458055.0
358	2020	DEC	484782.0
359	2021	DEC	NaN

In [8]:

```python
# Convert Month and Year to a date
df['Date'] = pd.to_datetime(dict(year = df.YEAR, month = pd.to_datetime(df.Mo
df.head()
```

Out[8]:

	YEAR	Month	Sales	Date
0	1992	JAN	146925.0	1992-01-01
1	1993	JAN	157555.0	1993-01-01
2	1994	JAN	167518.0	1994-01-01
3	1995	JAN	182413.0	1995-01-01
4	1996	JAN	189135.0	1996-01-01

In [9]:

```python
df.tail()
```

Out[9]:

	YEAR	Month	Sales	Date
355	2017	DEC	433282.0	2017-12-01
356	2018	DEC	434803.0	2018-12-01
357	2019	DEC	458055.0	2019-12-01
358	2020	DEC	484782.0	2020-12-01
359	2021	DEC	NaN	2021-12-01

In [10]:

```python
# Check any missing values accros columns
df.isnull().any()
```

Out[10]:

```
YEAR      False
Month     False
Sales      True
Date      False
dtype: bool
```

In [11]:

```
# Group Sales by Date
sales_df = df.groupby('Date')['Sales'].sum().to_frame("Sales").reset_index()
sales_df = sales_df[sales_df['Sales'] > 0]
sales_df.head()
```

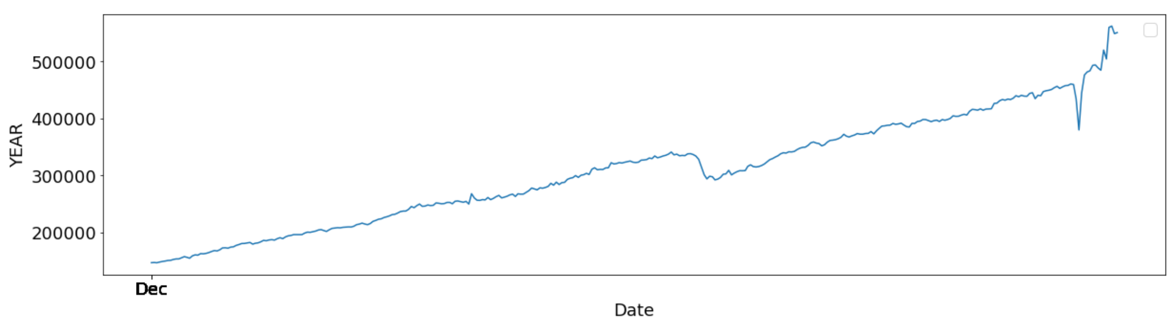
Out[11]:

	Date	Sales
0	1992-01-01	146925.0
1	1992-02-01	147223.0
2	1992-03-01	146805.0
3	1992-04-01	148032.0
4	1992-05-01	149010.0

In [12]:

```
plt.rcParams['font.size'] = 18
fig, ax = plt.subplots(figsize = (20,5))

plt.plot(sales_df['Date'], sales_df['Sales'])

#ax.plot(df['YEAR'], label = 'Sales per Year: Jan, 1992 - June, 2021')
#plt.xlim(("1992","2021"))
plt.xticks(["1992-01-01".format(x) for x in range(1,13,1)],["Jan","Feb","Mar"
                                                            "Jul","Aug","Sep"

plt.legend()
ax.legend()
ax.set_ylabel('YEAR')
ax.set_xlabel('Date')

plt.show()
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```



In [16]:
```
# I got the bellow code from "https://towardsdatascience.com/predicting-sales

from datetime import datetime, timedelta,date
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from __future__ import division

import warnings
warnings.filterwarnings("ignore")

import plotly.plotly as py
import plotly.offline as pyoff
import plotly.graph_objs as go

#import Keras
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.utils import np_utils
from keras.layers import LSTM
from sklearn.model_selection import KFold, cross_val_score, train_test_split

#initiate plotly
pyoff.init_notebook_mode()
```

In [17]:
```
#represent month in date field as its first day
df['Date'] = df['Date'].dt.year.astype('str') + '-' + df['Date'].dt.month.ast
df['Date'] = pd.to_datetime(df['Date'])
#groupby date and sum the sales
df = df.groupby('Date').Sales.sum().reset_index()
```

In [18]:
```
#plot monthly sales
plot_data = [
    go.Scatter(
        x=df['Date'],
        y=df['Sales'],
    )
]
plot_layout = go.Layout(
        title='Montly Sales'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

In [19]:

```
#create a new dataframe to model the difference
df_diff = df.copy()
#add previous sales to the next row
df_diff['prev_sales'] = df_diff['Sales'].shift(1)
#drop the null values and calculate the difference
df_diff = df_diff.dropna()
df_diff['diff'] = (df_diff['Sales'] - df_diff['prev_sales'])
df_diff.head(10)
```

Out[19]:

	Date	Sales	prev_sales	diff
1	1992-02-01	147223.0	146925.0	298.0
2	1992-03-01	146805.0	147223.0	-418.0
3	1992-04-01	148032.0	146805.0	1227.0
4	1992-05-01	149010.0	148032.0	978.0
5	1992-06-01	149800.0	149010.0	790.0
6	1992-07-01	150761.0	149800.0	961.0
7	1992-08-01	151067.0	150761.0	306.0
8	1992-09-01	152588.0	151067.0	1521.0
9	1992-10-01	153521.0	152588.0	933.0
10	1992-11-01	153583.0	153521.0	62.0

In [20]:

```
df_diff.tail(10)
```

Out[20]:

	Date	Sales	prev_sales	diff
350	2021-03-01	559871.0	504458.0	55413.0
351	2021-04-01	562269.0	559871.0	2398.0
352	2021-05-01	548987.0	562269.0	-13282.0
353	2021-06-01	550782.0	548987.0	1795.0
354	2021-07-01	0.0	550782.0	-550782.0
355	2021-08-01	0.0	0.0	0.0
356	2021-09-01	0.0	0.0	0.0
357	2021-10-01	0.0	0.0	0.0
358	2021-11-01	0.0	0.0	0.0
359	2021-12-01	0.0	0.0	0.0

In [21]:

```
#plot sales diff
plot_data = [
    go.Scatter(
        x=df_diff['Date'],
        y=df_diff['diff'],
    )
]
plot_layout = go.Layout(
        title='Montly Sales Diff'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

In [22]:

```
#create dataframe for transformation from time series to supervised
df_supervised = df_diff.drop(['prev_sales'],axis=1)
#adding lags
for inc in range(1,13):
    field_name = 'lag_' + str(inc)
    df_supervised[field_name] = df_supervised['diff'].shift(inc)
#drop null values
df_supervised = df_supervised.dropna().reset_index(drop=True)
```

In [23]:

```
df_supervised
```

Out[23]:

	Date	Sales	diff	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6
0	1993-02-01	156266.0	-1289.0	1941.0	2031.0	62.0	933.0	1521.0	306.0
1	1993-03-01	154752.0	-1514.0	-1289.0	1941.0	2031.0	62.0	933.0	1521.0
2	1993-04-01	158979.0	4227.0	-1514.0	-1289.0	1941.0	2031.0	62.0	933.0
3	1993-05-01	160605.0	1626.0	4227.0	-1514.0	-1289.0	1941.0	2031.0	62.0
4	1993-06-01	160127.0	-478.0	1626.0	4227.0	-1514.0	-1289.0	1941.0	2031.0
...	...	...	...	...	...	...	...	...	...
342	2021-08-01	0.0	0.0	-550782.0	1795.0	-13282.0	2398.0	55413.0	-15704.0
343	2021-09-01	0.0	0.0	0.0	-550782.0	1795.0	-13282.0	2398.0	55413.0
344	2021-10-01	0.0	0.0	0.0	0.0	-550782.0	1795.0	-13282.0	2398.0
345	2021-11-01	0.0	0.0	0.0	0.0	0.0	-550782.0	1795.0	-13282.0
346	2021-12-01	0.0	0.0	0.0	0.0	0.0	0.0	-550782.0	1795.0

347 rows × 15 columns

In [24]:

```
# Import statsmodels.formula.api
import statsmodels.formula.api as smf
# Define the regression formula
model = smf.ols(formula='diff ~ lag_1', data=df_supervised)
# Fit the regression
model_fit = model.fit()
# Extract the adjusted r-squared
regression_adj_rsq = model_fit.rsquared_adj
print(regression_adj_rsq)
```

```
-0.002846828618318309
```

lag_1 explains .2% of the variation. Let's check out others:

In [25]:

```
# Import statsmodels.formula.api
import statsmodels.formula.api as smf
# Define the regression formula
model = smf.ols(formula='diff ~ lag_1+lag_2+lag_3+lag_4+lag_5', data=df_super
# Fit the regression
model_fit = model.fit()
# Extract the adjusted r-squared
regression_adj_rsq = model_fit.rsquared_adj
print(regression_adj_rsq)
```

```
-0.004554188189075159
```

Adding four more features dicreased the score from .2% to .4%.

In [26]:

```
# Import statsmodels.formula.api
import statsmodels.formula.api as smf
# Define the regression formula
model = smf.ols(formula='diff ~ lag_1+lag_2+lag_3+lag_4+lag_5+lag_6+lag_7+lag
# Fit the regression
model_fit = model.fit()
# Extract the adjusted r-squared
regression_adj_rsq = model_fit.rsquared_adj
print(regression_adj_rsq)
```

```
0.0642838078337633
```

Adding more features dicreased the score from .4% to 6%.

In [27]:

```
#import MinMaxScaler and create a new dataframe for LSTM model
from sklearn.preprocessing import MinMaxScaler
df_model = df_supervised.drop(['Sales','Date'],axis=1)
#split train and test set
train_set, test_set = df_model[0:-1].values, df_model[-1:].values
```

In [ ]:

In [28]:

```
#apply Min Max Scaler
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(train_set)
# reshape training set
train_set = train_set.reshape(train_set.shape[0], train_set.shape[1])
train_set_scaled = scaler.transform(train_set)
# reshape test set
test_set = test_set.reshape(test_set.shape[0], test_set.shape[1])
test_set_scaled = scaler.transform(test_set)
```

In [ ]:

```
```

In [29]:

```
#X_train, y_train = train_set_scaled[:, 1:], train_set_scaled[:, 0:1]
#X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
#X_test, y_test = test_set_scaled[:, 1:], test_set_scaled[:, 0:1]
#X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
```

In [30]:

```
X_train, y_train = train_set_scaled[:, 1:], train_set_scaled[:, 0:1]
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test, y_test = test_set_scaled[:, 1:], test_set_scaled[:, 0:1]
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
```

In [31]:

```
#apply Min Max Scaler
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(train_set)
# reshape training set
train_set = train_set.reshape(train_set.shape[0], train_set.shape[1])
train_set_scaled = scaler.transform(train_set)
# reshape test set
test_set = test_set.reshape(test_set.shape[0], test_set.shape[1])
test_set_scaled = scaler.transform(test_set)
```

In [ ]:

```
```

In [32]:

```
model = Sequential()
model.add(LSTM(4, batch_input_shape=(1, X_train.shape[1], X_train.shape[2]),
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=100, batch_size=1, verbose=1, shuffle=Fals
```

```
Epoch 1/100
346/346 [==============================] - 3s 2ms/step - loss: 0.0158
Epoch 2/100
346/346 [==============================] - 1s 2ms/step - loss: 0.0118
Epoch 3/100
346/346 [==============================] - 1s 2ms/step - loss: 0.0112
Epoch 4/100
346/346 [==============================] - 1s 2ms/step - loss: 0.0107
Epoch 5/100
346/346 [==============================] - 1s 2ms/step - loss: 0.0103
Epoch 6/100
346/346 [==============================] - 1s 2ms/step - loss: 0.0100
Epoch 7/100
346/346 [==============================] - 1s 2ms/step - loss: 0.0098
Epoch 8/100
346/346 [==============================] - 1s 2ms/step - loss: 0.0095
Epoch 9/100
346/346 [==============================] - 1s 2ms/step - loss: 0.0094
Epoch 10/100
```

In [33]:

```
y_pred = model.predict(X_test,batch_size=1)
#for multistep prediction, you need to replace X_test values with the predict
```

```
1/1 [==============================] - 1s 621ms/step
```

In [34]:

```
#reshape y_pred
y_pred = y_pred.reshape(y_pred.shape[0], 1, y_pred.shape[1])
#rebuild test set for inverse transform
pred_test_set = []
for index in range(0,len(y_pred)):
     np.concatenate([y_pred[index],X_test[index]],axis=1)

pred_test_set.append(np.concatenate([y_pred[index],X_test[index]],axis=1))

#reshape pred_test_set
pred_test_set = np.array(pred_test_set)
pred_test_set = pred_test_set.reshape(pred_test_set.shape[0], pred_test_set.s
#inverse transform
pred_test_set_inverted = scaler.inverse_transform(pred_test_set)
```

In [35]:
```
#create dataframe that shows the predicted sales
result_list = []
sales_dates = list(df[-7:].Date)
act_sales = list(df[-7:].Sales)
for index in range(0,len(pred_test_set_inverted)):
    result_dict = {}
    result_dict['pred_value'] = int(pred_test_set_inverted[index][0] + act_sa
    result_dict['Date'] = sales_dates[index+1]
    result_list.append(result_dict)
df_result = pd.DataFrame(result_list)
#for multistep prediction, replace act_sales with the predicted sales
```

In [36]:
```
df_result
```

Out[36]:

	pred_value	Date
0	-227722	2021-07-01

In [37]:
```
#merge with actual sales dataframe
df_pred = pd.merge(df, df_result, on='Date', how='left')
#plot actual and predicted
plot_Data = [
    go.Scatter(
        x=df_pred['Date'],
        y=df_pred['Sales'],
        name='actual'
    ),
        go.Scatter(
        x=df_pred['Date'],
        y=df_pred['pred_value'],
        name='predicted'
    )

]
plot_layout = go.Layout(
        title='Sales Prediction'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

Summary

- It looks like the seal is an overall increasing, except drop between 2008 - 2009 and in 2020.

- The data is not stationary

- The result is impressive as the score is 60%

- I should drop row# 354 to get the better result

- Most of the code copied from "

```
In [ ]:
```