Exercise 8.2: Time Series Modeling

In [1]:

```
# First I will import some needed libraries
import pandas as pd
from importlib import reload
import sys
import numpy as np
from imp import reload
import nltk
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib as mpl
%matplotlib inline
import seaborn as sns
```

Importing the Data

In [235…

```
# I will use pandas to pull the data to create a data frame to work from
Retail_sales_Data = pd.read_csv('us_retail_sales.csv')
Retail_sales_Data
```

Out[235…

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	
0	1992	146925	147223	146805	148032	149010	149800	150761.0	151067.0	152588.0	153
1	1993	157555	156266	154752	158979	160605	160127	162816.0	162506.0	163258.0	164
2	1994	167518	169649	172766	173106	172329	174241	174781.0	177295.0	178787.0	180
3	1995	182413	179488	181013	181686	183536	186081	185431.0	186806.0	187366.0	186
4	1996	189135	192266	194029	194744	196205	196136	196187.0	196218.0	198859.0	200
5	1997	202371	204286	204990	203399	201699	204675	207014.0	207635.0	208326.0	208
6	1998	209666	209552	210832	213633	214639	216337	214841.0	213636.0	215720.0	219
7	1999	223997	226250	227417	229037	231235	231903	233948.0	236566.0	237481.0	237
8	2000	243436	247133	249825	245831	246201	248160	247176.0	247576.0	251837.0	251
9	2001	252654	252704	250328	254763	255218	254022	252997.0	254560.0	249845.0	267
10	2002	256307	257670	257059	261333	257573	259786	262769.0	265043.0	260626.0	261
11	2003	267230	263188	267820	267197	267362	270396	273352.0	277965.0	276430.0	274
12	2004	278913	280932	286209	282952	288252	284133	287358.0	287941.0	293139.0	295
13	2005	296696	300557	301308	303760	301776	310989	313520.0	310046.0	310673.0	310
14	2006	322348	320171	320869	322561	321794	323184	324204.0	325324.0	323236.0	322
15	2007	327181	327953	330579	329560	334202	331076	332342.0	334169.0	335442.0	337
16	2008	337412	334584	335193	334843	337947	338311	336771.0	334045.0	328343.0	314
17	2009	298673	297631	292300	293614	296501	302169	302802.0	309023.0	301033.0	304

18	2010	308299	308628	316003	318707	315604	314925	315632.0	317408.0	320080.0	32
19	2011	332357	334710	338007	339884	339303	341600	341373.0	342288.0	345496.0	34
20	2012	352862	357379	358719	356849	356018	352043	353891.0	358450.0	361470.0	36
21	2013	367009	372291	369081	367514	369493	371041	373554.0	372489.0	372505.0	37
22	2014	373033	378581	382601	386689	387100	388106	388359.0	391305.0	389860.0	39
23	2015	385648	385157	391420	391356	394718	395464	398193.0	398105.0	396248.0	39
24	2016	394749	398105	396911	398190	400143	404756	403730.0	403968.0	405958.0	40
25	2017	416081	415503	414620	416889	414540	416505	416744.0	417179.0	426501.0	42
26	2018	432148	434106	433232	435610	439996	438191	440703.0	439278.0	438985.0	44
27	2019	440751	439996	447167	448709	449552	450927	454012.0	456500.0	452849.0	45
28	2020	460586	459610	434281	379892	444631	476343	481627.0	483716.0	493327.0	49
29	2021	520162	504458	559871	562269	548987	550782	NaN	NaN	NaN	

Reviewing the Data Set

In [236…

```
# Next I will display the dimensions of the Retail sales dataframe
Retail_sales_Data.shape
```

Out[236… (30, 13)

Above we see the Retail sales Data frame has 13 columns for the year and month and 20 rows that are for the sales of each year that is broken out by month.

In [237…

```
# Next I will use info to see further information on each of my variables.
Retail_sales_Data.info()
```

```
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
```

Above we can see that July through December are float64 due to the fact that we have NaNs for those months in 2022

In [238…
```
# Next I will check and see what variables have NA values and what percentage o
Retail_sales_Data.isnull().sum()/len(Retail_sales_Data)
```

Out[238…
```
YEAR    0.000000
JAN     0.000000
FEB     0.000000
MAR     0.000000
APR     0.000000
MAY     0.000000
JUN     0.000000
JUL     0.033333
AUG     0.033333
SEP     0.033333
OCT     0.033333
NOV     0.033333
DEC     0.033333
dtype: float64
```

As suspected we see that July through December have a 0.033333 percent of NaNs that come from 2021. We will create a model to predict the monthly retail sales on the last year of data so I will not adjust these NANs at the moment.

In [239…
```
# Next I will use melt to transform my data frame
Retail_Data = pd.melt(Retail_sales_Data, id_vars=["YEAR"], var_name="Month", va
Retail_Data
```

Out[239…

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

360 rows × 3 columns

In [240…

```
# I will now use info to view my Dtypes
Retail_Data.info()
```

```
RangeIndex: 360 entries, 0 to 359
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   YEAR    360 non-null    int64  
 1   Month   360 non-null    object 
 2   Sales   354 non-null    float64
dtypes: float64(1), int64(1), object(1)
memory usage: 8.6+ KB
```

In [241…

```
# I will now use replace to change my months from float to an int
Retail_Data["Month"].replace({'JAN': 1, 'FEB': 2, 'MAR': 3, 'APR': 4, 'MAY': 5,
                'SEP': 9, 'OCT': 10, 'NOV': 11, 'DEC': 12}, inplace=True)
Retail_Data
```

Out[241…

	YEAR	Month	Sales
0	1992	1	146925.0
1	1993	1	157555.0
2	1994	1	167518.0
3	1995	1	182413.0
4	1996	1	189135.0
…	…	…	…
355	2017	12	433282.0
356	2018	12	434803.0
357	2019	12	458055.0
358	2020	12	484782.0
359	2021	12	NaN

360 rows × 3 columns

In [242…

```
# I will now use to_datetime to combine year and month and create a date variab
Retail_Data['DATE'] = pd.to_datetime(Retail_Data[['YEAR', 'Month']].assign(DAY=
Retail_Data
```

Out[242…

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

...	...	...	...	...
355	2017	12	433282.0	2017-12-01
356	2018	12	434803.0	2018-12-01
357	2019	12	458055.0	2019-12-01
358	2020	12	484782.0	2020-12-01
359	2021	12	NaN	2021-12-01

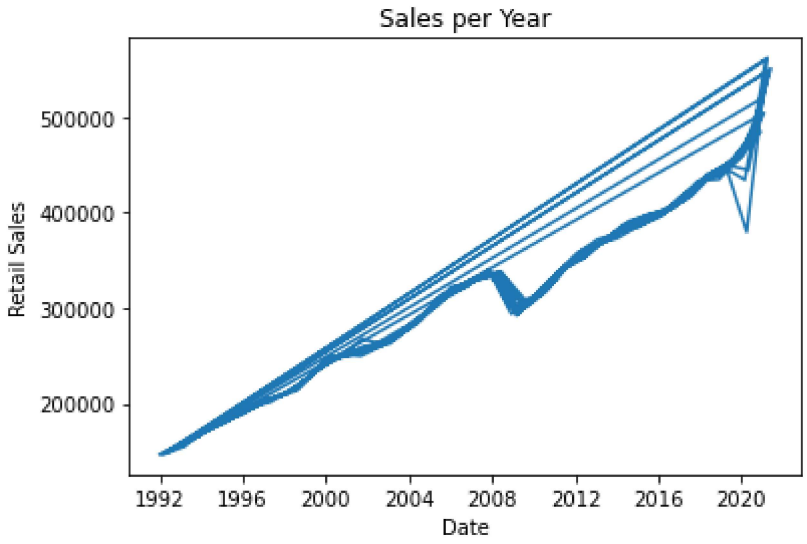
360 rows × 4 columns

Step 1: Plot the data with proper labeling and make some observations on the graph.

In [243…

```
# I will create a plot that shows the sales per year
x = Retail_Data.DATE
y = Retail_Data.Sales
plt.plot(x, y)
plt.xlabel("Date")
plt.ylabel("Retail Sales")
plt.title("Sales per Year")
```

Out[243… `Text(0.5, 1.0, 'Sales per Year')`



Above we see that the retial sales rise up until 2008 and the we see adip due to the market crash and we see another dip in 2020 and a sharp rise right after. This is most likely due to the pandemic as when everyone got furlowed or even lost their job they tried to spend less.

Step 2: 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

training set.

In [244…
```
# First I will split the data into training and test set for July 2020 - June 2
# and the rest as the training set
train_df = Retail_Data[Retail_Data['DATE'] < '2020-07-01']
test_df = Retail_Data[Retail_Data['DATE'] >= '2020-07-01']
```

In [245…
```
# Next I will split the target variable from the feature
X_train = test_df[["DATE"]]
Y_train = test_df[["Sales"]]
```

Step 3: Use the training set to build a predictive model for the monthly retail sales.

In [246…
```
# I will pull my sales data
Sale = train_df[['Sales']]
```

In [249…
```
Sale.index = pd.DatetimeIndex(Sale.index).to_period('M')
```

In [250…
```
# I will now fit my model with the use of SARIMAX
from statsmodels.tsa.statespace.sarimax import SARIMAX
ARMA_model = SARIMAX(Sale, order = (1, 0, 1))
```

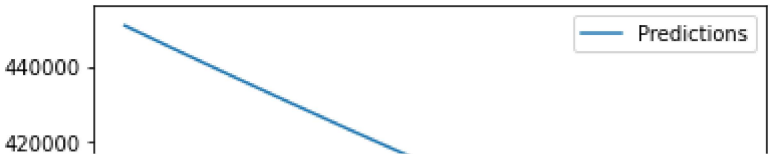
In [251…
```
ARMA_model = ARMA_model.fit()
```

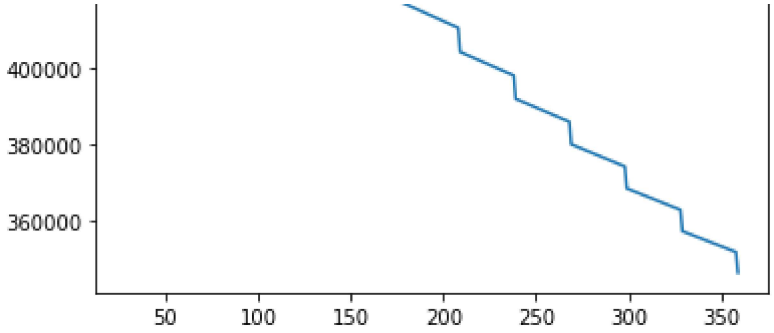
Step 4: Use the model to predict the monthly retail sales on the last year of data.

In [252…
```
# I will use my ARMA_model to predict my monthly retail sales
y_pred = ARMA_model.get_forecast(len(test_df.index))
y_pred_df = y_pred.conf_int(alpha = 0.05)
y_pred_df["Predictions"] = ARMA_model.predict(start = y_pred_df.index[0], end =
y_pred_df.index = test_df.index
y_pred_out = y_pred_df["Predictions"]
```

In [258…
```
# I will now plot my prediction for my monthly sales
plt.plot(y_pred_out, label = 'Predictions')
plt.legend()
```

Out[258…





above we see as tim goes on we see a decrease in sales

Step 5: Report the RMSE of the model predictions on the test set.

In [257…

```
# I will now calculate the RMSE with the use of mean_squared_error
from sklearn.metrics import mean_squared_error
test_df = test_df.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
arma_rmse = np.sqrt(mean_squared_error(test_df["Sales"].values, y_pred_df["Pred
print("RMSE: ",arma_rmse)
```

```
RMSE:  234669.78586317564
```

Above we see we have a pretty large RMSE of 234669.78