Time Series Modeling

In [2]:

```
## import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

## library versions
print('pandas version:', pd.__version__)
print('numpy version:', np.__version__)
print('seaborn version:', sns.__version__)
```

```
pandas version: 1.4.2
numpy version: 1.21.5
seaborn version: 0.11.2
```

In [28]:

```
## load dataset
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
df = pd.read_csv("us_retail_sales.csv")
print(df)
```

```
    YEAR     JAN     FEB     MAR     APR     MAY     JUN       JUL  
AUG  \
0   1992  146925  147223  146805  148032  149010  149800  150761.0  1
51067.0
1   1993  157555  156266  154752  158979  160605  160127  162816.0  1
62506.0
2   1994  167518  169649  172766  173106  172329  174241  174781.0  1
77295.0
3   1995  182413  179488  181013  181686  183536  186081  185431.0  1
86806.0
4   1996  189135  192266  194029  194744  196205  196136  196187.0  1
96218.0
5   1997  202371  204286  204990  203399  201699  204675  207014.0  2
07635.0
6   1998  209666  209552  210832  213633  214639  216337  214841.0  2
13636.0
7   1999  223997  226250  227417  229037  231235  231903  233948.0  2
36566.0
8   2000  243436  247133  249825  245831  246201  248160  247176.0  2
47576.0
9   2001  252654  252704  250328  254763  255218  254022  252997.0  2
54560.0
10  2002  256307  257670  257059  261333  257573  259786  262769.0  2
65043.0
11  2003  267230  263188  267820  267197  267362  270396  273352.0  2
77965.0
12  2004  278913  280932  286209  282952  288252  284133  287358.0  2
87941.0
13  2005  296696  300557  301308  303760  301776  310989  313520.0  3
10046.0
14  2006  322348  320171  320869  322561  321794  323184  324204.0  3
25324.0
15  2007  327181  327953  330579  329560  334202  331076  332342.0  3
34169.0
16  2008  337412  334584  335193  334843  337947  338311  336771.0  3
34045.0
17  2009  298673  297631  292300  293614  296501  302169  302802.0  3
09023.0
18  2010  308299  308628  316003  318707  315604  314925  315632.0  3
17408.0
19  2011  332357  334710  338007  339884  339303  341600  341373.0  3
42288.0
20  2012  352862  357379  358719  356849  356018  352043  353891.0  3
58450.0
21  2013  367009  372291  369081  367514  369493  371041  373554.0  3
72489.0
22  2014  373033  378581  382601  386689  387100  388106  388359.0  3
91305.0
23  2015  385648  385157  391420  391356  394718  395464  398193.0  3
98105.0
24  2016  394749  398105  396911  398190  400143  404756  403730.0  4
03968.0
25  2017  416081  415503  414620  416889  414540  416505  416744.0  4
17179.0
26  2018  432148  434106  433232  435610  439996  438191  440703.0  4
39278.0
27  2019  440751  439996  447167  448709  449552  450927  454012.0  4
```

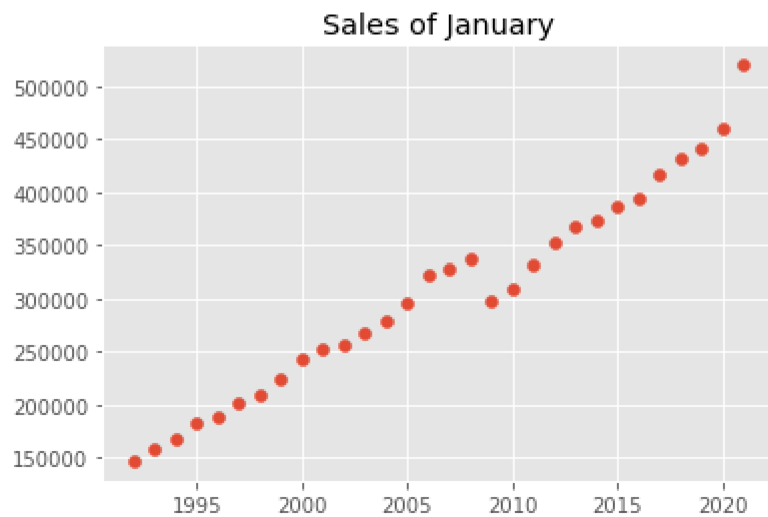
```
56500.0
28  2020  460586  459610  434281  379892  444631  476343  481627.0  4
83716.0
29  2021  520162  504458  559871  562269  548987  550782       NaN
NaN
```

	SEP	OCT	NOV	DEC
0	152588.0	153521.0	153583.0	155614.0
1	163258.0	164685.0	166594.0	168161.0
2	178787.0	180561.0	180703.0	181524.0
3	187366.0	186565.0	189055.0	190774.0
4	198859.0	200509.0	200174.0	201284.0
5	208326.0	208078.0	208936.0	209363.0
6	215720.0	219483.0	221134.0	223179.0
7	237481.0	237553.0	240544.0	245485.0
8	251837.0	251221.0	250331.0	250658.0
9	249845.0	267999.0	260514.0	256549.0
10	260626.0	261953.0	263568.0	265930.0
11	276430.0	274764.0	278298.0	277612.0
12	293139.0	295115.0	296177.0	299763.0
13	310673.0	310479.0	313303.0	313473.0
14	323236.0	322678.0	323343.0	326849.0
15	335442.0	337530.0	341133.0	336189.0
16	328343.0	314830.0	301332.0	294025.0
17	301033.0	304154.0	306675.0	308413.0
18	320080.0	323900.0	327745.0	329627.0
19	345496.0	347924.0	349304.0	349744.0
20	361470.0	361991.0	362876.0	364488.0
21	372505.0	373663.0	373914.0	377032.0
22	389860.0	390506.0	391805.0	388569.0
23	396248.0	394503.0	396240.0	397052.0
24	405958.0	407395.0	406061.0	412610.0
25	426501.0	426933.0	431158.0	433282.0
26	438985.0	444038.0	445242.0	434803.0
27	452849.0	455486.0	457658.0	458055.0
28	493327.0	493991.0	488652.0	484782.0
29	NaN	NaN	NaN	NaN

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

In [12]:

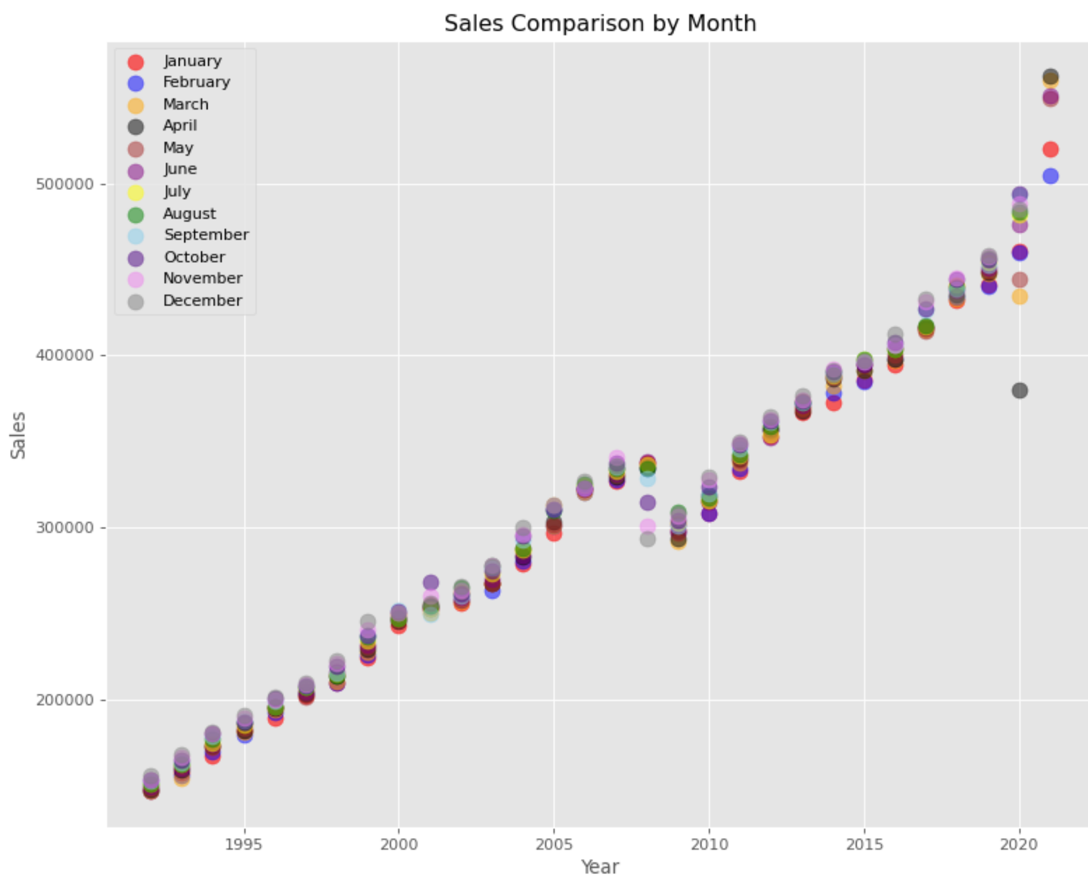
```
## scatterplot
plt.scatter(df["YEAR"], df["JAN"])
plt.title('Sales of January')
plt.show()
```



Observation: I was originally going to plot one month at a time to see if any months had downward trends, but after plotting, January/February/March, I noticed that this quarter all had similar growth. I decided to plot the year against one another to get an overall idea of the sales month over month. Looking at the chart below, 2020 had the most volatility, while the years leading up to it all stayed within the same general sales range. Earlier, 2008 also had more variety with the general year showing a downward trend January to December.

In [11]:

```
## scatterplot of months
## import library
from matplotlib.pyplot import figure

figure(figsize=(10, 8), dpi=80)

plt.style.use('ggplot')

plt.title('Sales Comparison by Month')
plt.xlabel('Year')
plt.ylabel('Sales')

plt.scatter(x = df['YEAR'],y = df['JAN'],s = 100,c = 'red',alpha = 0.6
plt.scatter( x= df['YEAR'],y = df['FEB'],s = 100,c = 'blue',alpha = 0.
plt.scatter( x= df['YEAR'],y = df['MAR'],s = 100,c = 'orange',alpha =
plt.scatter( x= df['YEAR'],y = df['APR'],s = 100,c = 'black',alpha = 0
plt.scatter( x= df['YEAR'],y = df['MAY'],s = 100,c = 'brown',alpha = 0
plt.scatter( x= df['YEAR'],y = df['JUN'],s = 100,c = 'purple',alpha =
plt.scatter( x= df['YEAR'],y = df['JUL'],s = 100,c = 'yellow',alpha =
plt.scatter( x= df['YEAR'],y = df['AUG'],s = 100,c = 'green',alpha = 0
plt.scatter( x= df['YEAR'],y = df['SEP'],s = 100,c = 'skyblue',alpha =
plt.scatter( x= df['YEAR'],y = df['OCT'],s = 100,c = 'indigo',alpha =
plt.scatter( x= df['YEAR'],y = df['NOV'],s = 100,c = 'violet',alpha =
plt.scatter( x= df['YEAR'],y = df['DEC'],s = 100,c = 'gray',alpha = 0.

plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



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.

In [110]:

```
## change column names to numeric
df2 = df.set_axis(['YEAR', '1', '2', '3', '4', '5', '6', '7', '8', '9'
df2.head()
```

Out[110]:

	YEAR	1	2	3	4	5	6	7	8	
0	1992	146925	147223	146805	148032	149010	149800	150761.0	151067.0	152588
1	1993	157555	156266	154752	158979	160605	160127	162816.0	162506.0	163258
2	1994	167518	169649	172766	173106	172329	174241	174781.0	177295.0	178787
3	1995	182413	179488	181013	181686	183536	186081	185431.0	186806.0	187366
4	1996	189135	192266	194029	194744	196205	196136	196187.0	196218.0	198859

In [214]:

```
## melt df
df_melt = pd.melt(df2, id_vars = ['YEAR'], value_vars = ['1', '2', '3'
df_melt.head()
```

Out[214]:

	YEAR	variable	value
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

In [215]:

```
## add day value
df_melt['day']='1'
df_melt.head()
```

Out[215]:

	YEAR	variable	value	day
0	1992	1	146925.0	1
1	1993	1	157555.0	1
2	1994	1	167518.0	1
3	1995	1	182413.0	1
4	1996	1	189135.0	1

In [216]:

```python
## rename melt df columns
df_melt.columns = ['Year', 'Month', 'Sales', 'Day']
df_melt.head()
```

Out[216]:

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

In [219]:

```python
## consolidate year + month
df_melt['Date'] = pd.to_datetime(df_melt[['Day','Month','Year']], dayf
df_melt.head()
```

Out[219]:

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

In [220]:

```python
## drop columns
df3 = df_melt
df3 = df3.drop('Year', axis=1)
df3 = df3.drop('Month', axis=1)
df3 = df3.drop('Day', axis=1)

## change order
df3 = df3[['Date', 'Sales']]
df3.head()
```

Out[220]:

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

In [221]:
```
print(df3.dtypes)
```

```
Date     datetime64[ns]
Sales           float64
dtype: object
```

In [222]:
```
## drop rows as later troubleshoot attempt
df3 = df3.dropna()
```

In [223]:
```
df3 = df3.set_index(df3['Date'])
df3 = df3.sort_index()
```

In [227]:
```
## import library
from datetime import datetime

## split data
split_date = datetime(2020, 6, 1)

train = df3.loc[df3['Date'] <= split_date]
test = df3.loc[df3['Date'] > split_date]

print('Train Dataset:',train.shape)
print('Test Dataset:',test.shape)
```

```
Train Dataset: (342, 2)
Test Dataset: (12, 2)
```

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

In [246]:
```
## import library
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [259]:

```
y = train['Sales'] ## input
ARMAmodel = SARIMAX(y, order = (1, 0, 1)) ## define model
ARMAmodel = ARMAmodel.fit() ## fit model

## predictions
y_pred = ARMAmodel.get_forecast(len(test.index))
y_pred_df = y_pred.conf_int(alpha = 0.05)
y_pred_df["Predictions"] = ARMAmodel.predict(start = y_pred_df.index[0
y_pred_df.index = test.index
y_pred_out = y_pred_df["Predictions"]

## challenges troubleshooting no frequency information output
```

```
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so
inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so
inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\statespace
\sarimax.py:966: UserWarning: Non-stationary starting autoregressive
parameters found. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
```

In [248]:

```
## troubleshooting errors
## code above to drop July 2021 columns on
df3.isnull().sum().sum()
```

Out[248]:

```
0
```

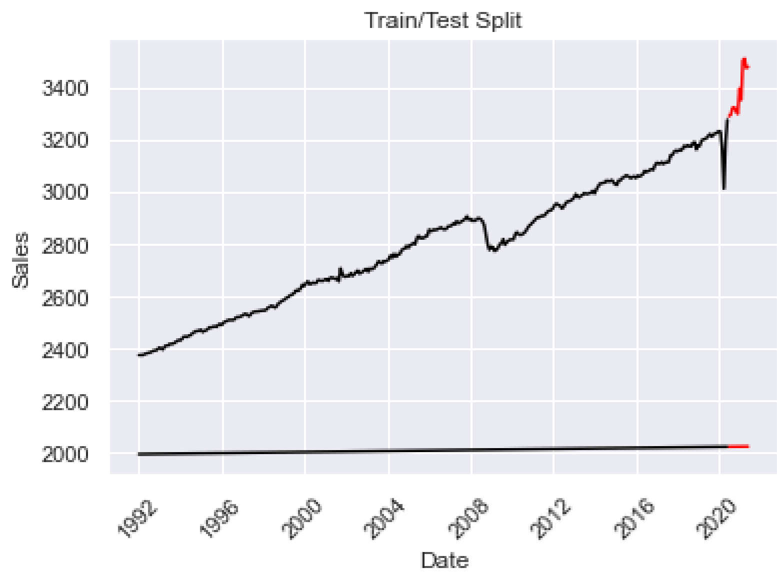
In [231]:

```
## troubleshooting errors
## looking at secondary line
## possibly due to set index?
print(df3.shape)
```

```
(354, 2)
```

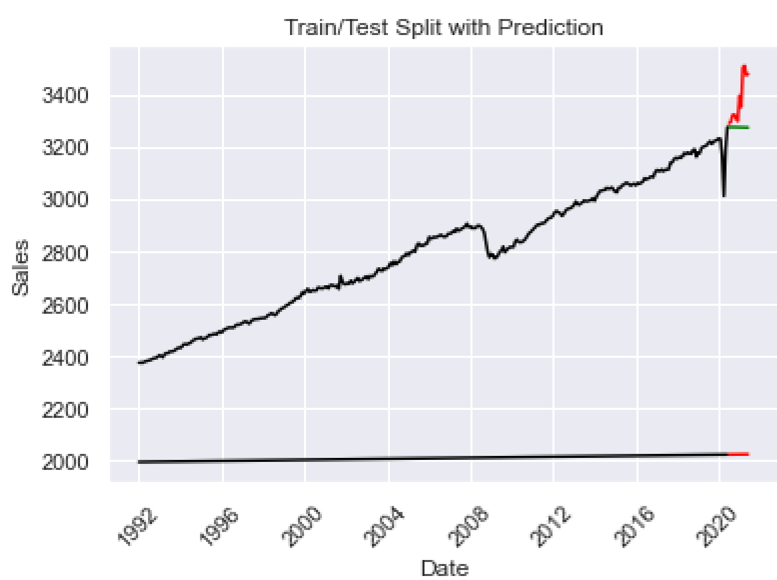
In [249]:

```
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.ylabel('Sales')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title("Train/Test Split")
plt.show()
```



In [250]:

```
plt.plot(y_pred_out, color='green')
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.ylabel('Sales')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title("Train/Test Split with Prediction")
plt.show()
```



In [251]:

```
## RMSE of training data
## import library
from sklearn.metrics import mean_squared_error

train_rmse = np.sqrt(mean_squared_error(test['Sales'].values, y_pred_d
print("RMSE: ",train_rmse)
```

```
RMSE:  49316.86825188333
```

Observation: High RMSE in training set. Model performance is low based on historical trends. Visually, it looks like there is a stagnation or decrease whereas reality shows an increase.

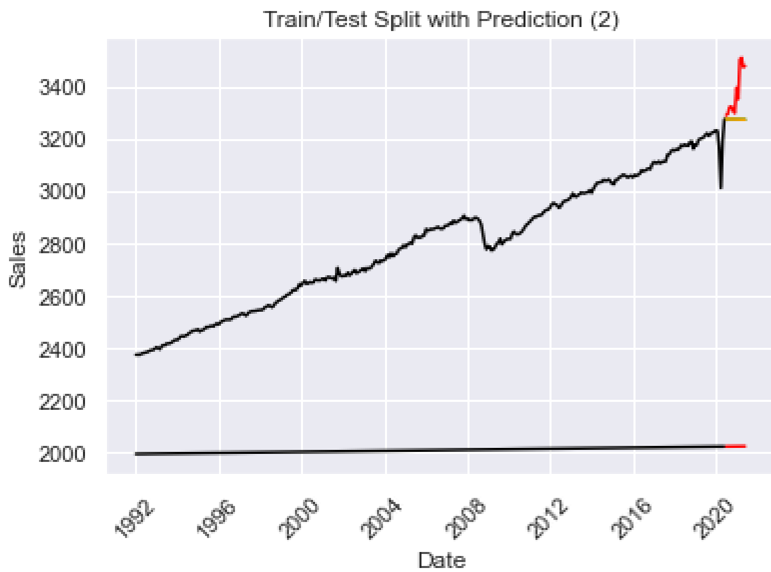
In [254]:

```
## finetuning model
## import library
from statsmodels.tsa.arima.model import ARIMA

ARIMAmodel = ARIMA(y, order = (1, 1, 1))
ARIMAmodel = ARIMAmodel.fit()

## predictions
y_pred = ARMAmodel.get_forecast(len(test.index))
y_pred_df = y_pred.conf_int(alpha = 0.05)
y_pred_df["Predictions"] = ARMAmodel.predict(start = y_pred_df.index[0
y_pred_df.index = test.index
y_pred_out2 = y_pred_df["Predictions"]
```

```
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so
inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so
inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so
inferred frequency MS will be used.
  self._init_dates(dates, freq)
```

In [255]:

```
plt.plot(y_pred_out, color='green')
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.plot(y_pred_out2, color='orange')
plt.ylabel('Sales')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title("Train/Test Split with Prediction (2)")
plt.show()
```



In [256]:

```
train_2_rmse = np.sqrt(mean_squared_error(test['Sales'].values, y_pred
print("RMSE: ",train_2_rmse)
```

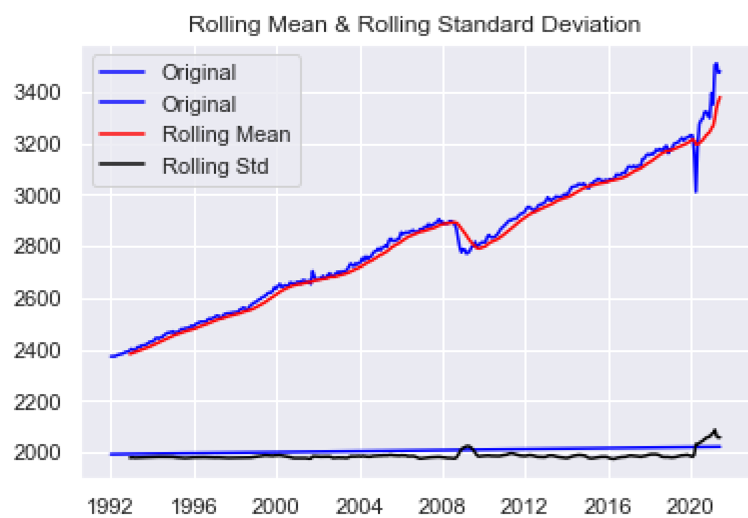
```
RMSE:  49316.86825188333
```

Observation: unsure if second prediction is overlaid upon first or if there is an additional error, but there was no improvement to the RMSE. I'm wondering if introducing seasonality would improve the model's performance. Troubleshooting hasn't led to any fixes for the model's automated frequency choice. I went back to the actual datetime logic and was unable to update to "datetime" or add "freq"

In [258]:

```
rolling_mean = df3.rolling(window = 12).mean()
rolling_std = df3.rolling(window = 12).std()
plt.plot(df3, color = 'blue', label = 'Original')
plt.plot(rolling_mean, color = 'red', label = 'Rolling Mean')
plt.plot(rolling_std, color = 'black', label = 'Rolling Std')
plt.legend(loc = 'best')
plt.title('Rolling Mean & Rolling Standard Deviation')
plt.show()

## data is not stationary
```

```
C:\Users\alexi\AppData\Local\Temp\ipykernel_10032\1929232825.py:1: Fu
tureWarning: Dropping of nuisance columns in rolling operations is de
precated; in a future version this will raise TypeError. Select only
valid columns before calling the operation. Dropped columns were Inde
x(['Date'], dtype='object')
  rolling_mean = df3.rolling(window = 12).mean()
C:\Users\alexi\AppData\Local\Temp\ipykernel_10032\1929232825.py:2: Fu
tureWarning: Dropping of nuisance columns in rolling operations is de
precated; in a future version this will raise TypeError. Select only
valid columns before calling the operation. Dropped columns were Inde
x(['Date'], dtype='object')
  rolling_std = df3.rolling(window = 12).std()
```



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

In [269]:

```
y = test['Sales'] ## input
ARMAmodel = SARIMAX(y, order = (1, 0, 1)) ## define model
ARMAmodel = ARMAmodel.fit() ## fit model

## predictions
y_pred = ARMAmodel.get_forecast(len(test.index))
y_pred_df = y_pred.conf_int(alpha = 0.05)
y_pred_df["Predictions"] = ARMAmodel.predict(start = y_pred_df.index[0
y_pred_df.index = test.index
y_pred_out = y_pred_df["Predictions"]
```

```
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so
inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_m
odel.py:471: ValueWarning: No frequency information was provided, so
inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\alexi\anaconda3\lib\site-packages\statsmodels\tsa\statespace
\sarimax.py:966: UserWarning: Non-stationary starting autoregressive
parameters found. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
```

In [270]:

```
test_rmse = np.sqrt(mean_squared_error(test['Sales'].values, y_pred_df
print("RMSE: ",test_rmse)
```

```
RMSE:  46868.4980834692
```