# Time Series Analysis - Time_Series_Analysis_Daily_Website_Visitors

Introduction to Time-Series Analysis
• A time-series data is a series of data points or observations recorded at different or regular time intervals. In general, a time series is a sequence of data points taken at equally spaced time intervals. The frequency of recorded data points may be hourly, daily, weekly, monthly, quarterly or annually.
• Time-Series Forecasting is the process of using a statistical model to predict future values of a time-series based on past results. • A time series analysis encompasses statistical methods for analyzing time series data. These methods enable us to extract meaningful statistics, patterns and other characteristics of the data. Time series are visualized with the help of line charts. So, time series analysis involves understanding inherent aspects of the time series data so that we can create meaningful and accurate forecasts.
• Applications of time series are used in statistics, finance or business applications. A very common example of time series data is the daily closing value of the stock index like NASDAQ or Dow Jones. Other common applications of time series are sales and demand forecasting, weather forecasting, econometrics, signal processing, pattern recognition and earthquake prediction

## Components of a Time-Series

• Trend - The trend shows a general direction of the time series data over a long period of time. A trend can be increasing(upward), decreasing(downward), or horizontal(stationary).
• Seasonality - The seasonality component exhibits a trend that repeats with respect to timing, direction, and magnitude. Some examples include an increase in water consumption in summer due to hot weather conditions.
• Cyclical Component - These are the trends with no set repetition over a particular period of time. A cycle refers to the period of ups and downs, booms and slums of a time series, mostly observed in business cycles. These cycles do not exhibit a seasonal variation but generally occur over a time period of 3 to 12 years depending on the nature of the time series.
• Irregular Variation - These are the fluctuations in the time series data which become evident when trend and cyclical variations are removed. These variations are unpredictable, erratic, and may or may not be random.
• ETS Decomposition - ETS Decomposition is used to separate different components of a time series. The term ETS stands for Error, Trend and Seasonality

In [ ]:
```
##!mkdir ~/.kaggle
```

In [4]:
```
##!cp /kaggle.json ~/.kaggle/
```

In [5]:
```
##!chmod 600 ~/.kaggle/kaggle.json
```

In [ ]:
```
####! pip install kaggle
```

In [ ]:
```
####!pip install keras-tuner
```

In [ ]:
```
###! kaggle datasets download -d bobnau/daily-website-visitors
```

In [10]:
```
###! unzip ./daily-website-visitors.zip
```

In [11]:
```
###! pip install tensorflow
```

In [12]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn import preprocessing
import matplotlib.pyplot as plt
tf.random.set_seed(123)
np.random.seed(123)
```

In [13]:
```
import pandas as pd
#import fbprophet
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

In [14]:
```
df=pd.read_csv('/content/daily-website-visitors.csv', parse_dates=['Date'], index_col ="Date")
df.head(3)
```

Out[14]:

| Date | Row | Day | Day.Of.Week | Page.Loads | Unique.Visits | First.Time.Visits | Returning.Visits |
|---|---|---|---|---|---|---|---|
| **2014-09-14** | 1 | Sunday | 1 | 2,146 | 1,582 | 1,430 | 152 |
| **2014-09-15** | 2 | Monday | 2 | 3,621 | 2,528 | 2,297 | 231 |
| **2014-09-16** | 3 | Tuesday | 3 | 3,698 | 2,630 | 2,352 | 278 |

In [15]:
```
df.columns
```

Out[15]:
```
Index(['Row', 'Day', 'Day.Of.Week', 'Page.Loads', 'Unique.Visits',
       'First.Time.Visits', 'Returning.Visits'],
      dtype='object')
```

In [16]:
```
df2 = df [['Page.Loads', 'Unique.Visits',
       'First.Time.Visits', 'Returning.Visits']]
```

In [17]:
```
df2.describe()
```

Out[17]:

| | Page.Loads | Unique.Visits | First.Time.Visits | Returning.Visits |
|---|---|---|---|---|
| **count** | 2167 | 2167 | 2167 | 2167 |
| **unique** | 1756 | 1658 | 1587 | 663 |
| **top** | 2,948 | 2,780 | 3,146 | 552 |
| **freq** | 5 | 5 | 5 | 12 |

In [18]:
```
df2['Page.Loads'] = df2['Page.Loads'].str.replace(',', '').astype(int)
```

```
<ipython-input-18-10a780dcfd9e>:1: 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#returnin
g-a-view-versus-a-copy
  df2['Page.Loads'] = df2['Page.Loads'].str.replace(',', '').astype(int)
```

In [19]:
```
df2['Unique.Visits'] = df2['Unique.Visits'].str.replace(',', '').astype(int)
```

```
<ipython-input-19-749f4711fe3b>:1: 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#returnin
g-a-view-versus-a-copy
  df2['Unique.Visits'] = df2['Unique.Visits'].str.replace(',', '').astype(int)
```

In [20]:
```
df2['First.Time.Visits'] = df2['First.Time.Visits'].str.replace(',', '').astype(int)
```

```
<ipython-input-20-d715d5f15c0d>:1: 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#returnin
g-a-view-versus-a-copy
  df2['First.Time.Visits'] = df2['First.Time.Visits'].str.replace(',', '').astype(int)
```

In [21]:
```
df2['Returning.Visits'] = df2['Returning.Visits'].str.replace(',', '').astype(int)
```

```
<ipython-input-21-7e6c0f8bfde5>:1: 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#returnin
g-a-view-versus-a-copy
  df2['Returning.Visits'] = df2['Returning.Visits'].str.replace(',', '').astype(int)
```

In [22]:
```
df2.head(3)
```

Out[22]:

| Date | Page.Loads | Unique.Visits | First.Time.Visits | Returning.Visits |
|---|---|---|---|---|
| **2014-09-14** | 2146 | 1582 | 1430 | 152 |
| **2014-09-15** | 3621 | 2528 | 2297 | 231 |
| **2014-09-16** | 3698 | 2630 | 2352 | 278 |

In [23]:
```
# Basic packages
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import random as rd # generating random numbers
import datetime # manipulating date formats
# Viz
import matplotlib.pyplot as plt # basic plotting
import seaborn as sns # for prettier plots
# TIME SERIES
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.stattools import adfuller, acf, pacf,arma_order_select_ic
import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs
# settings
import warnings
warnings.filterwarnings("ignore")
```

In [24]:
```
monthly_page_loads = df2["Page.Loads"].resample('M').sum()
```

In [25]:
```
monthly_unique_visitors = df2["Unique.Visits"].resample('M').sum()
```

In [26]:
```
monthly_First_Time_Visits = df2["First.Time.Visits"].resample('M').sum()
```

In [27]:
```
monthly_Returning_Visits = df2["Returning.Visits"].resample('M').sum()
```

In [28]:
```
monthly_page_loads.head(3)
```

Out[28]:
```
Date
2014-09-30     56052
2014-10-31    121983
2014-11-30    114190
Freq: M, Name: Page.Loads, dtype: int64
```

In [29]:
```
monthly_page_loads = pd.DataFrame(monthly_page_loads)
```

In [30]:
```
monthly_page_loads.head(2)
```

Out[30]:

| | Page.Loads |
|---|---|
| **Date** | |
| **2014-09-30** | 56052 |
| **2014-10-31** | 121983 |

In [31]:
```
monthly_page_loads = monthly_page_loads.reset_index()
```

In [32]:
```
monthly_page_loads['weekday'] = monthly_page_loads['Date'].apply(lambda x: x.weekday())
monthly_page_loads.head()
```

Out[32]:

| | Date | Page.Loads | weekday |
|---|---|---|---|
| **0** | 2014-09-30 | 56052 | 1 |
| **1** | 2014-10-31 | 121983 | 4 |
| **2** | 2014-11-30 | 114190 | 6 |
| **3** | 2014-12-31 | 105617 | 2 |
| **4** | 2015-01-31 | 96077 | 5 |

In [33]:
```
monthly_page_loads['month']=monthly_page_loads['Date'].dt.month
monthly_page_loads.head()
```

Out[33]:

| | Date | Page.Loads | weekday | month |
|---|---|---|---|---|
| **0** | 2014-09-30 | 56052 | 1 | 9 |
| **1** | 2014-10-31 | 121983 | 4 | 10 |
| **2** | 2014-11-30 | 114190 | 6 | 11 |
| **3** | 2014-12-31 | 105617 | 2 | 12 |
| **4** | 2015-01-31 | 96077 | 5 | 1 |

In [34]:

```
monthly_page_loads['day']=monthly_page_loads['Date'].dt.day
monthly_page_loads.head()
```

Out[34]:

| | Date | Page.Loads | weekday | month | day |
|---|---|---|---|---|---|
| **0** | 2014-09-30 | 56052 | 1 | 9 | 30 |
| **1** | 2014-10-31 | 121983 | 4 | 10 | 31 |
| **2** | 2014-11-30 | 114190 | 6 | 11 | 30 |
| **3** | 2014-12-31 | 105617 | 2 | 12 | 31 |
| **4** | 2015-01-31 | 96077 | 5 | 1 | 31 |

In [35]:

```
train_month = monthly_page_loads.groupby(["month", "weekday"])['Page.Loads'].mean().reset_index()
train_month = train_month.pivot('weekday','month','Page.Loads')
train_month.sort_index(inplace=True)
```

In [36]:

```
train_month.head()
```

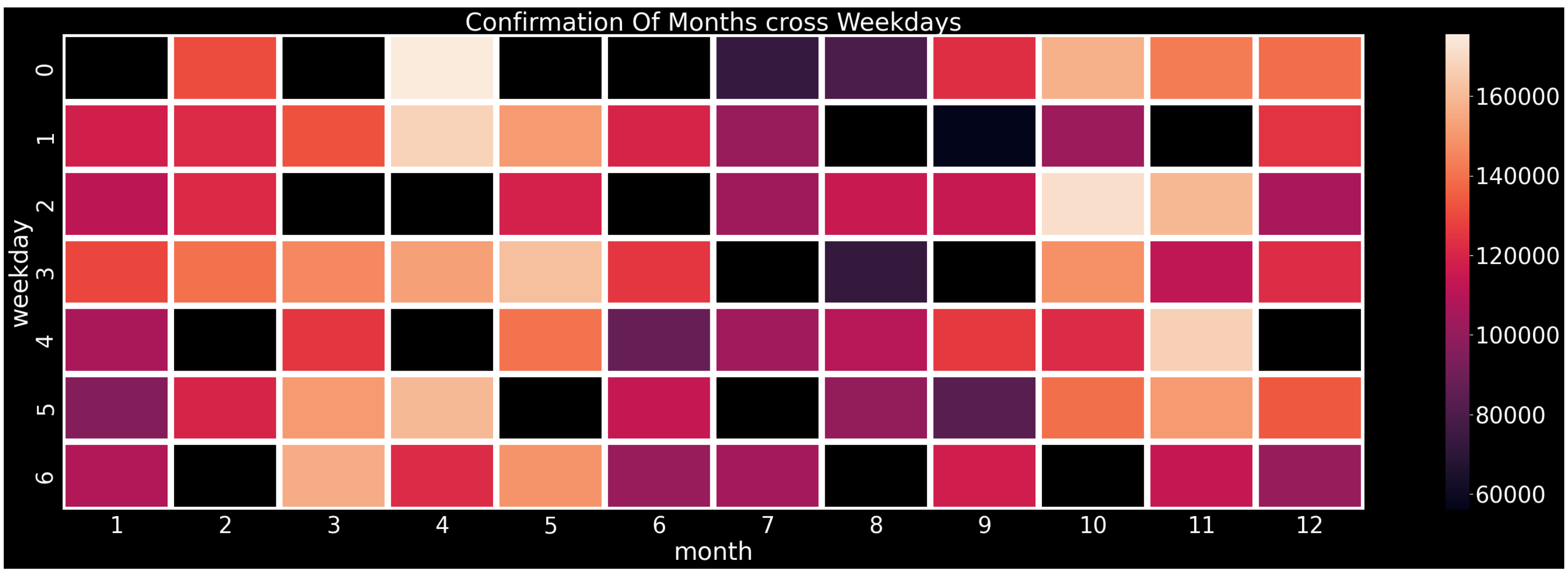
Out[36]:

| month<br>weekday | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | 131171.0 | NaN | 175572.0 | NaN | NaN | 73308.0 | 79732.0 | 123213.0 | 157604.0 | 142650.0 | 139140.0 |
| **1** | 118114.0 | 122170.0 | 132481.5 | 167945.0 | 151279.0 | 119990.0 | 101669.0 | NaN | 56052.0 | 102451.0 | NaN | 124419.0 |
| **2** | 111716.0 | 121740.0 | NaN | NaN | 119004.0 | NaN | 103522.0 | 114891.0 | 114716.0 | 171488.0 | 159651.0 | 105617.0 |
| **3** | 129723.0 | 139744.0 | 145948.0 | 152386.5 | 161708.0 | 125190.0 | NaN | 72854.0 | NaN | 148176.0 | 112155.0 | 122397.0 |
| **4** | 106492.0 | NaN | 125290.0 | NaN | 140413.0 | 87766.0 | 104132.0 | 109901.0 | 126234.0 | 121983.0 | 167131.0 | NaN |

In [37]:

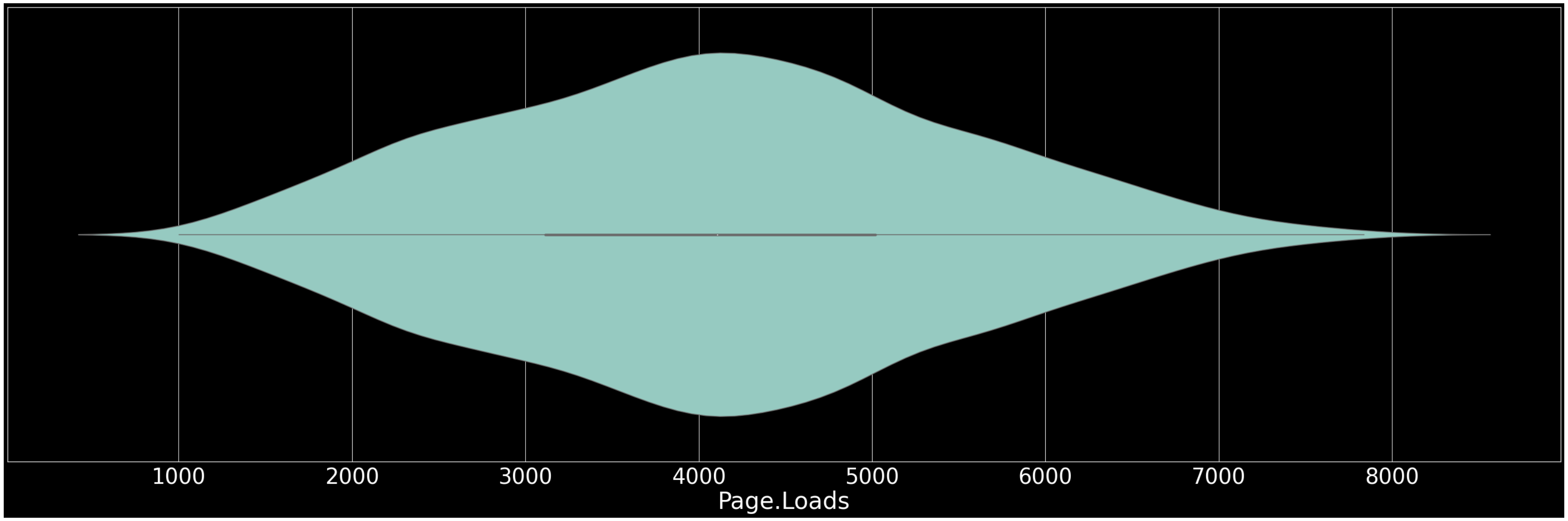
```python
import seaborn as sns


sns.set(font_scale=3.5)
plt.style.use('dark_background')
# Draw a heatmap with the numeric values in each cell
f, ax = plt.subplots(figsize=(50, 15))
sns.heatmap(train_month, annot=False, ax=ax, fmt="d", linewidths=10)
plt.title('Confirmation Of Months cross Weekdays')
plt.show()
```



In [38]:
```
plt.figure(figsize=(50,15))
plt.style.use('dark_background')
sns.violinplot(df2['Page.Loads'])
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2cb34c1c10>




In [39]:
```
monthly_page_loads.head(10)
```

Out[39]:

| | Date | Page.Loads | weekday | month | day |
|---|---|---|---|---|---|
| **0** | 2014-09-30 | 56052 | 1 | 9 | 30 |
| **1** | 2014-10-31 | 121983 | 4 | 10 | 31 |
| **2** | 2014-11-30 | 114190 | 6 | 11 | 30 |
| **3** | 2014-12-31 | 105617 | 2 | 12 | 31 |
| **4** | 2015-01-31 | 96077 | 5 | 1 | 31 |
| **5** | 2015-02-28 | 118876 | 5 | 2 | 28 |
| **6** | 2015-03-31 | 143990 | 1 | 3 | 31 |
| **7** | 2015-04-30 | 153331 | 3 | 4 | 30 |
| **8** | 2015-05-31 | 142113 | 6 | 5 | 31 |
| **9** | 2015-06-30 | 115297 | 1 | 6 | 30 |

In [40]:
```
train_days = monthly_page_loads.groupby(["month", "day"])['Page.Loads'].mean().reset_index()
train_days = train_days.pivot('day','month','Page.Loads')
train_days.sort_index(inplace=True)
train_days.dropna(inplace=True)
```

In [41]:
```
df2.columns
```

Out[41]:
```
Index(['Page.Loads', 'Unique.Visits', 'First.Time.Visits', 'Returning.Visits'], dtype='object')
```

In [42]:
```
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [43]:
```
#####! pip install plotly
```

In [44]:
```
import plotly
# plotly.tools.set_credentials_file()
```

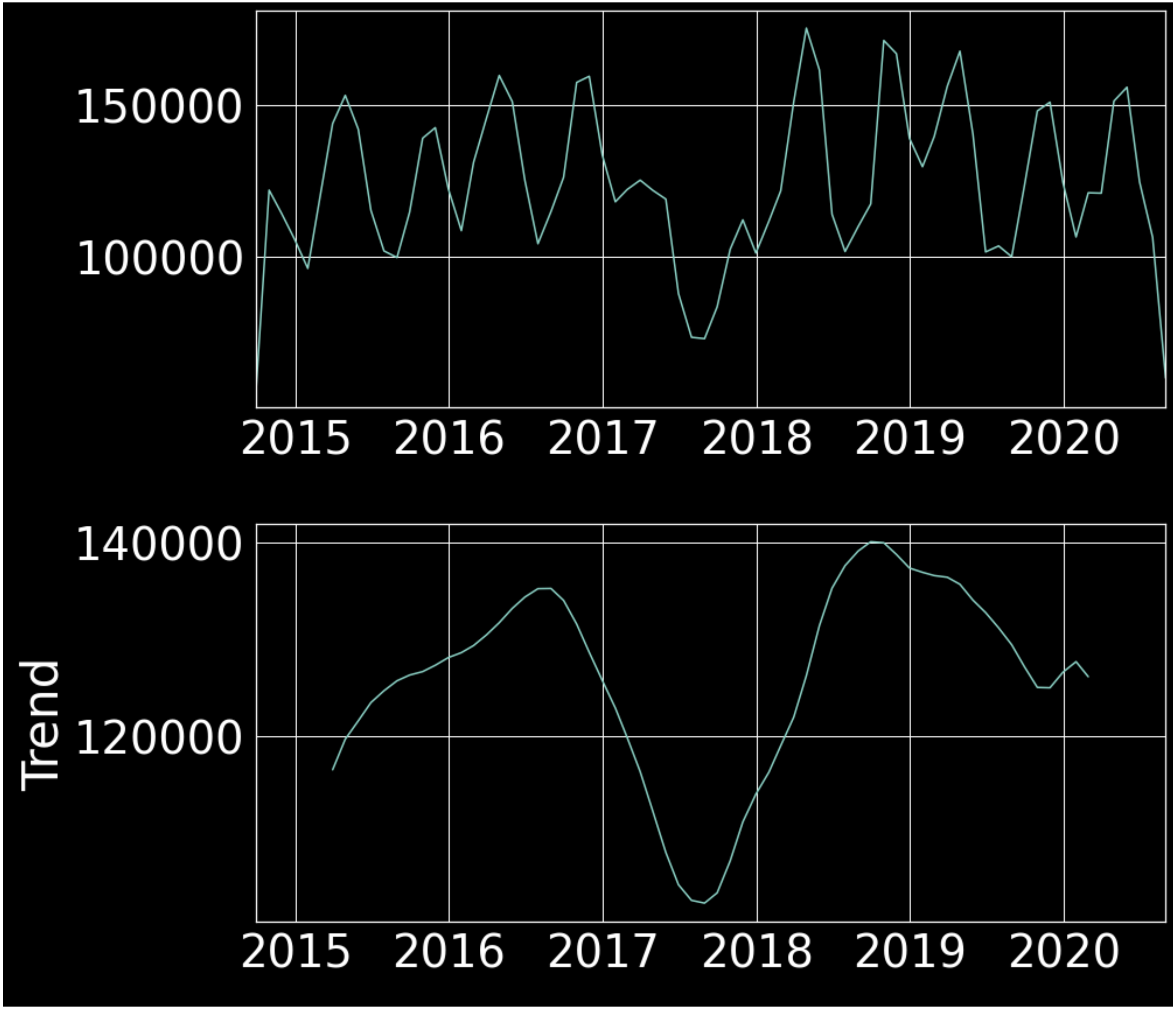
In [45]:
```
monthly_page_loads.rename(columns = {"Page.Loads" : "Page_Loads"}, inplace=True)
```

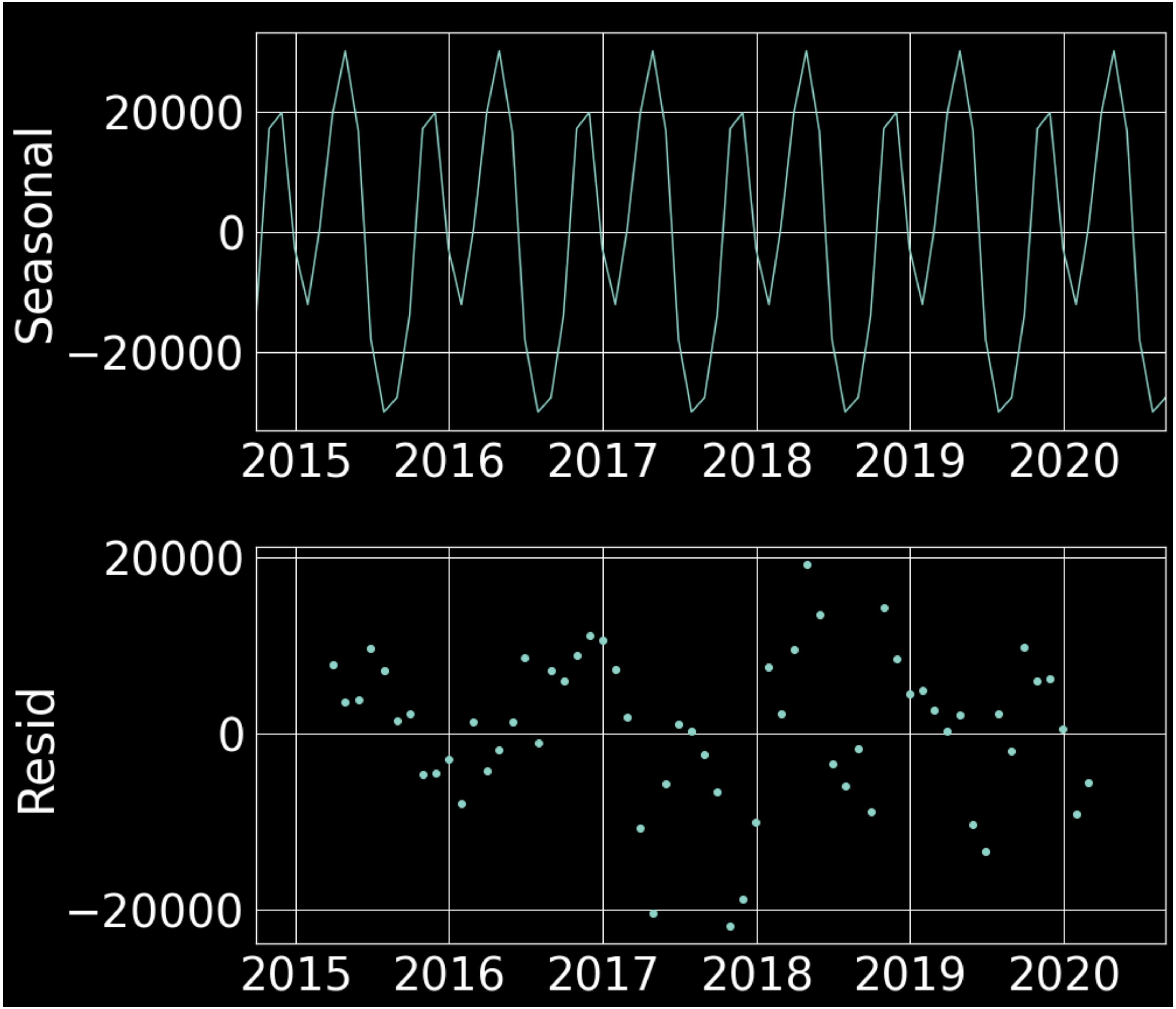
In [46]:
```
# Show Rolling mean, Rolling Std and Test for the stationnarity
df_date_index = monthly_page_loads[['Date','Page_Loads']].set_index('Date')
df_date_index.head()
```

Out[46]:

| | Page_Loads |
|---|---|
| **Date** | |
| **2014-09-30** | 56052 |
| **2014-10-31** | 121983 |
| **2014-11-30** | 114190 |
| **2014-12-31** | 105617 |
| **2015-01-31** | 96077 |

In [47]:
```
from pylab import rcParams
rcParams['figure.figsize'] = 15, 25
decomposition = sm.tsa.seasonal_decompose(df_date_index, model='additive')
fig = decomposition.plot()
plt.show()
```


150000
100000
2015
2016
2017
2018
2019
2020
140000
Trend
120000
2015
2016
2017
2018
2019
2020

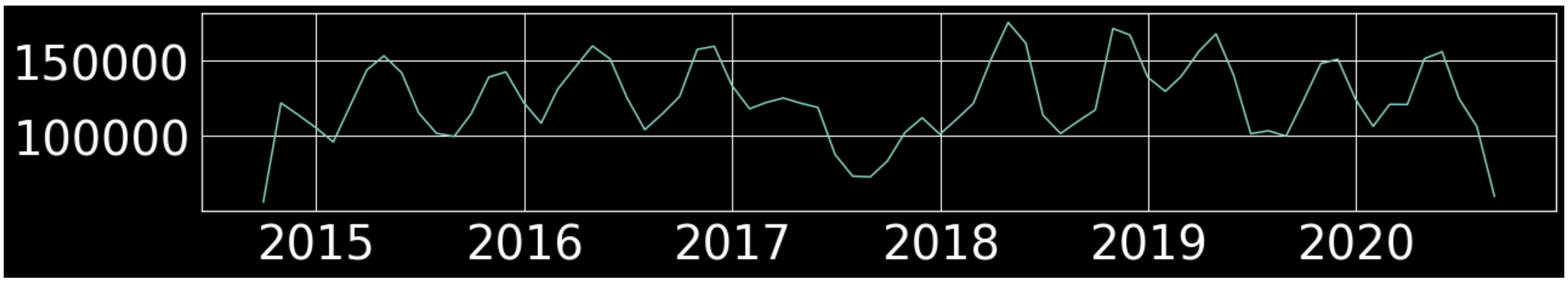

Seasonal
20000
0
−20000
2015
2016
2017
2018
2019
2020
Resid
20000
0
−20000
2015
2016
2017
2018
2019
2020

## Stationarity

A Time Series is said to be stationary if its statistical properties such as mean, variance remain constant over time. Most of the Time Series models work on the assumption that the TS is stationary. Major reason for this is that there are many ways in which a series can be non-stationary, but only one way for stationarity.
Intuitively, we can say that if a Time Series has a particular behaviour over time, there is a very high probability that it will follow the same in the future.
Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series.

In [48]:
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20,3)
plt.plot(df_date_index)
```

Out[48]:
```
[<matplotlib.lines.Line2D at 0x7f2cacdf0f40>]
```




Testing For Stationarity

In [49]:
```
### Testing For Stationarity

from statsmodels.tsa.stattools import adfuller

def adf_test(dataset):
  dftest = adfuller(dataset, autolag = 'AIC')
  print("1. ADF : ",dftest[0])
  print("2. P-Value : ", dftest[1])
  print("3. Num Of Lags : ", dftest[2])
  print("4. Num Of Observations Used For ADF Regression and Critical Values Calculation :", dftest[3])
  print("5. Critical Values :")
  for key, val in dftest[4].items():
      print("\t",key, ": ", val)
```

# AD-FULLER-TEST

In [50]:
```
df_date_index.columns
```

Out[50]:
```
Index(['Page_Loads'], dtype='object')
```

In [51]:
```
adf_test(df_date_index['Page_Loads'])
```

```
1. ADF :  -2.410024672338927
2. P-Value :  0.13892698645568835
3. Num Of Lags :  12
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 59
5. Critical Values :
	 1% :  -3.5463945337644063
	 5% :  -2.911939409384601
	 10% :  -2.5936515282964665
```

## DIFFERENCING

In [52]:
```
df_date_index['Page_Loads_First_Order_Differencing'] = df_date_index['Page_Loads'] - df_date_index['Page_Loads'].shi
```

In [53]:
```
adf_test(df_date_index['Page_Loads_First_Order_Differencing'].dropna())
```

```
1. ADF :  -1.9067763211534854
2. P-Value :  0.32888006117831814
3. Num Of Lags :  11
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 59
5. Critical Values :
         1% :  -3.5463945337644063
         5% :  -2.911939409384601
         10% :  -2.5936515282964665
```

In [54]:
```
df_date_index['Page_Loads_Second_Order_Differencing'] = df_date_index['Page_Loads_First_Order_Differencing'] - df_dat
```

In [55]:
```
adf_test(df_date_index['Page_Loads_Second_Order_Differencing'].dropna())
```

```
1. ADF :  -5.5736290225934155
2. P-Value :  1.4486118892546306e-06
3. Num Of Lags :  10
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 59
5. Critical Values :
         1% :  -3.5463945337644063
         5% :  -2.911939409384601
         10% :  -2.5936515282964665
```

In [56]:
```
def adfuller_test(confirmed):

    result=adfuller(confirmed)

    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']

    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:

        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary ")
```

In [57]:
```python
adfuller_test(df_date_index['Page_Loads_Second_Order_Differencing'].dropna())
```

```
ADF Test Statistic : -5.5736290225934155
p-value : 1.4486118892546306e-06
#Lags Used : 10
Number of Observations Used : 59
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary
```

## Plotting ACF and PACF

Autocorrelation and Partial Autocorrelation Functions

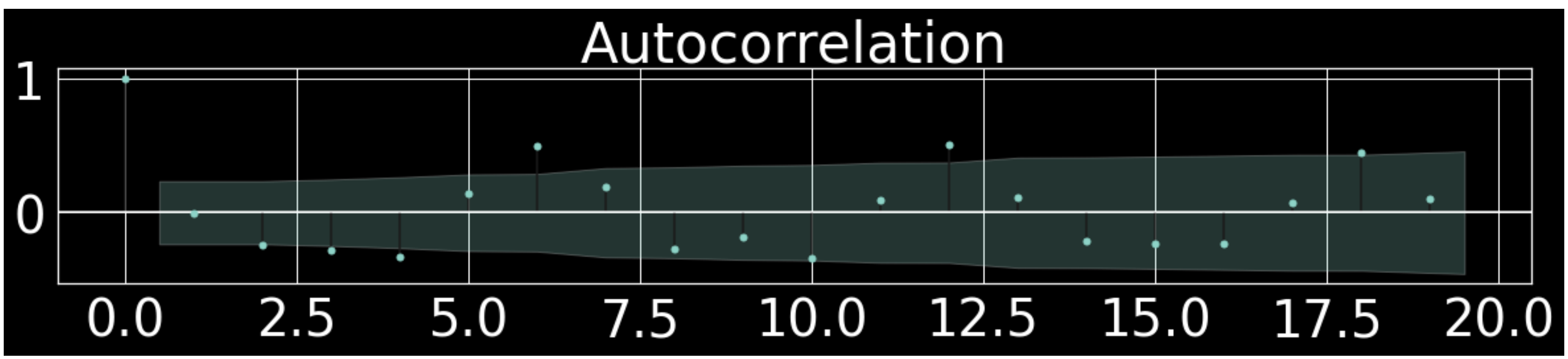
• Autocorrelation is simply the correlation of a series with its own lags. If a series is significantly autocorrelated, that means, the previous values of the series (lags) may be helpful in predicting the current value.
• Partial Autocorrelation also conveys similar information but it conveys the pure correlation of a series and its lag, excluding the correlation contributions from the intermediate lags.

In [58]:
```python
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

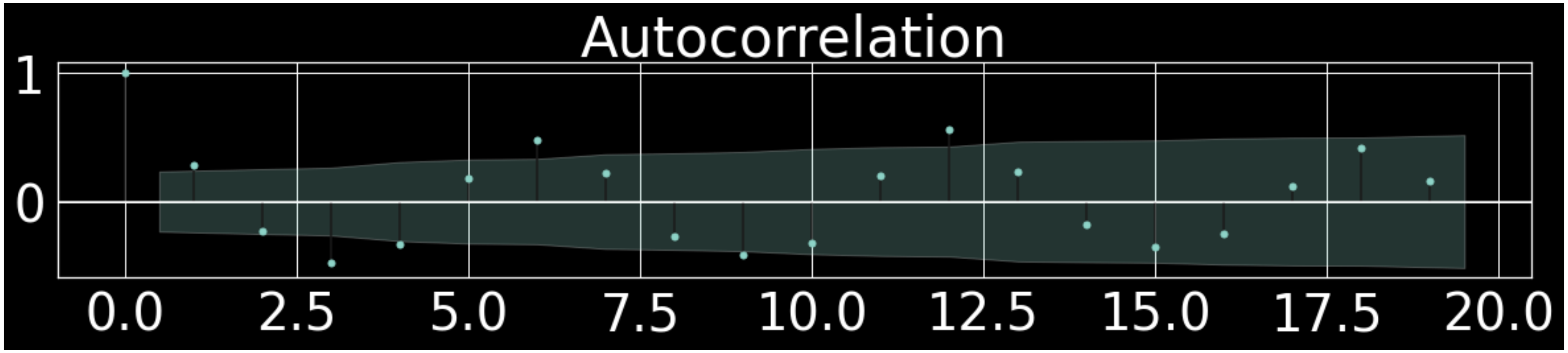
In [59]:
```python
df_date_index.columns
```

Out[59]:
```
Index(['Page_Loads', 'Page_Loads_First_Order_Differencing',
       'Page_Loads_Second_Order_Differencing'],
      dtype='object')
```

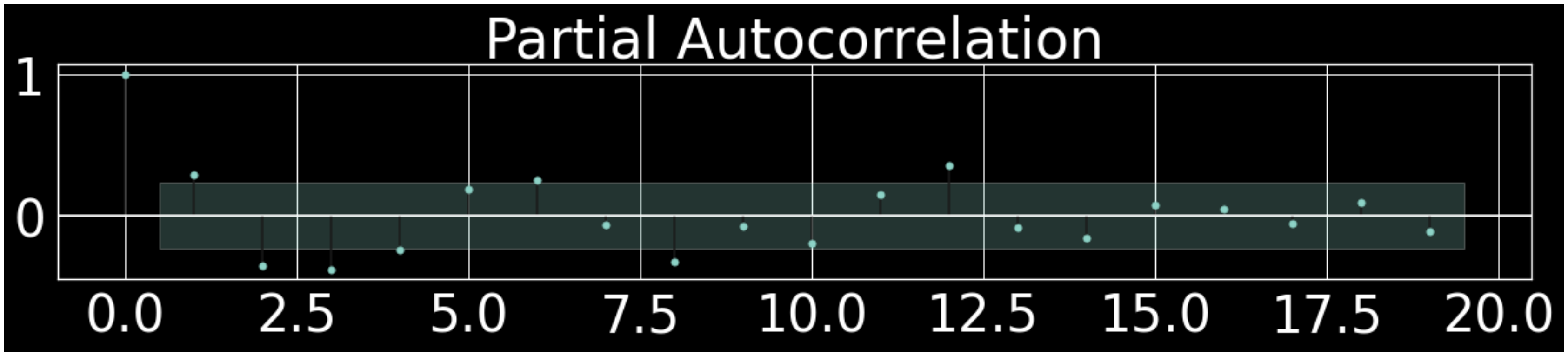
In [60]:
```python
acf2 = plot_acf(df_date_index['Page_Loads_Second_Order_Differencing'].dropna())
```



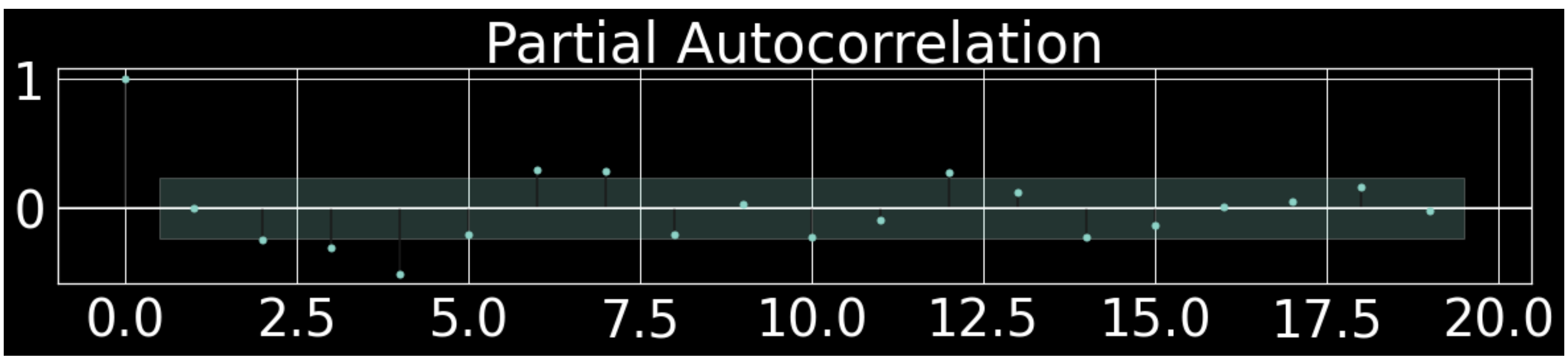

In [61]:
```python
acf1 = plot_acf(df_date_index["Page_Loads_First_Order_Differencing"].dropna())
```




In [62]:
```
result1 = plot_pacf(df_date_index["Page_Loads_First_Order_Differencing"].dropna())
```




In [63]:
```
result2 = plot_pacf(df_date_index['Page_Loads_Second_Order_Differencing'].dropna())
```




Split Data into `Training` and `Testing`

In [64]:
```
print(df_date_index.shape)
train=df_date_index.iloc[:-30]
test=df_date_index.iloc[-30:]
print(train.shape,test.shape)
### print(test.iloc[0],test.iloc[-1])
```

```
(72, 3)
(42, 3) (30, 3)
```

In [65]:
```
train.columns
```

Out[65]:
```
Index(['Page_Loads', 'Page_Loads_First_Order_Differencing',
       'Page_Loads_Second_Order_Differencing'],
      dtype='object')
```

In [66]:
```
from statsmodels.tsa.arima.model   import ARIMA
model_ARIMA=ARIMA(train['Page_Loads_Second_Order_Differencing'],order=(0,2,0))
```

In [67]:
```
model_Arima_fit=model_ARIMA.fit()
```

In [68]:
```
model_Arima_fit.summary()
```

Out[68]:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Page_Loads_Second_Order_Differencing | **No. Observations:** | 42 |
| **Model:** | ARIMA(0, 2, 0) | **Log Likelihood** | -483.086 |
| **Date:** | Sun, 11 Dec 2022 | **AIC** | 968.171 |
| **Time:** | 16:08:09 | **BIC** | 969.860 |
| **Sample:** | 09-30-2014 | **HQIC** | 968.782 |
| | - 02-28-2018 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **sigma2** | 1.381e+09 | 3.09e+08 | 4.465 | 0.000 | 7.75e+08 | 1.99e+09 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 13.79 | **Jarque-Bera (JB):** | 0.70 |
| **Prob(Q):** | 0.00 | **Prob(JB):** | 0.70 |
| **Heteroskedasticity (H):** | 0.87 | **Skew:** | -0.06 |
| **Prob(H) (two-sided):** | 0.80 | **Kurtosis:** | 2.36 |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [69]:

```
##prediction
pred_start_date=test.index[0]
pred_end_date=test.index[-1]
print(pred_start_date)
print(pred_end_date)
```

```
2018-03-31 00:00:00
2020-08-31 00:00:00
```

In [70]:

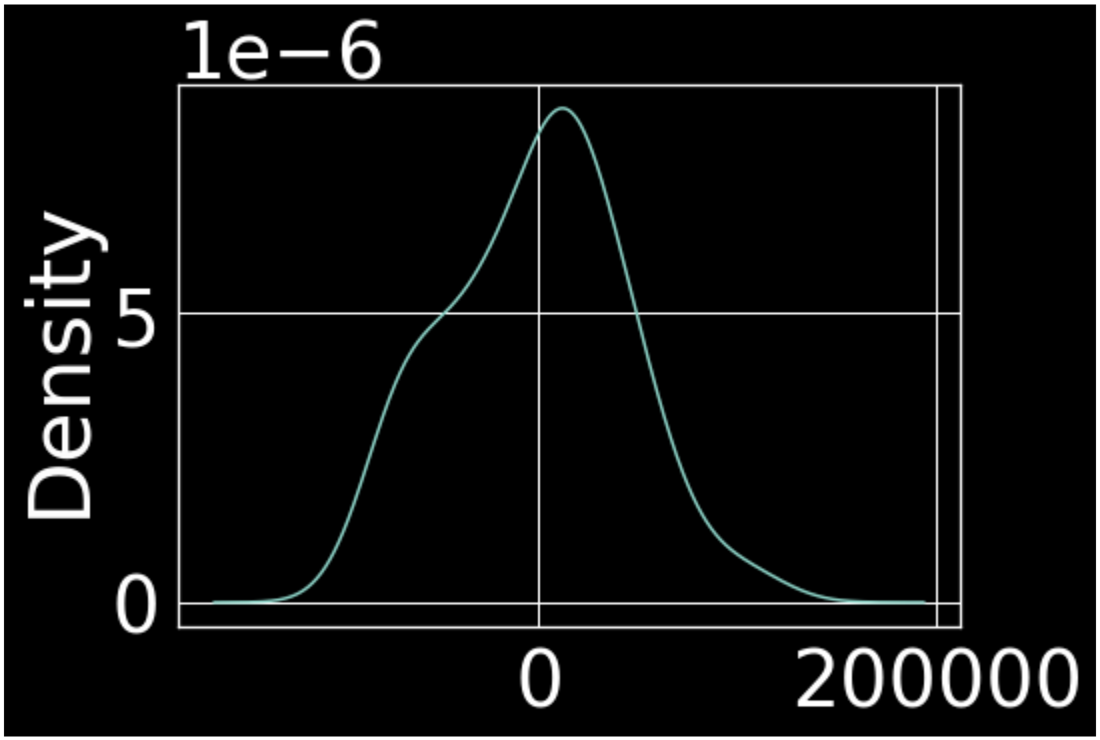
```
pred=model_Arima_fit.predict(start=pred_start_date,end=pred_end_date)
```

In [71]:
```
residuals=test['Page_Loads']- pred
```

In [72]:
```
residuals.head(3)
```

Out[72]:
```
Date
2018-03-31    173775.0
2018-04-30    220193.0
2018-05-31    228379.0
dtype: float64
```

In [73]:
```
plt.figure(figsize=[7,5])
model_Arima_fit.resid.plot(kind='kde')
plt.show()
```




In [74]:
```
test['Predicted_ARIMA']=pred
```

In [75]:
```
test.columns
```

Out[75]:
```
Index(['Page_Loads', 'Page_Loads_First_Order_Differencing',
       'Page_Loads_Second_Order_Differencing', 'Predicted_ARIMA'],
      dtype='object')
```

In [79]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, median_absolute_error, mean_squared_lo
```

In [80]:
```
from math import sqrt
```

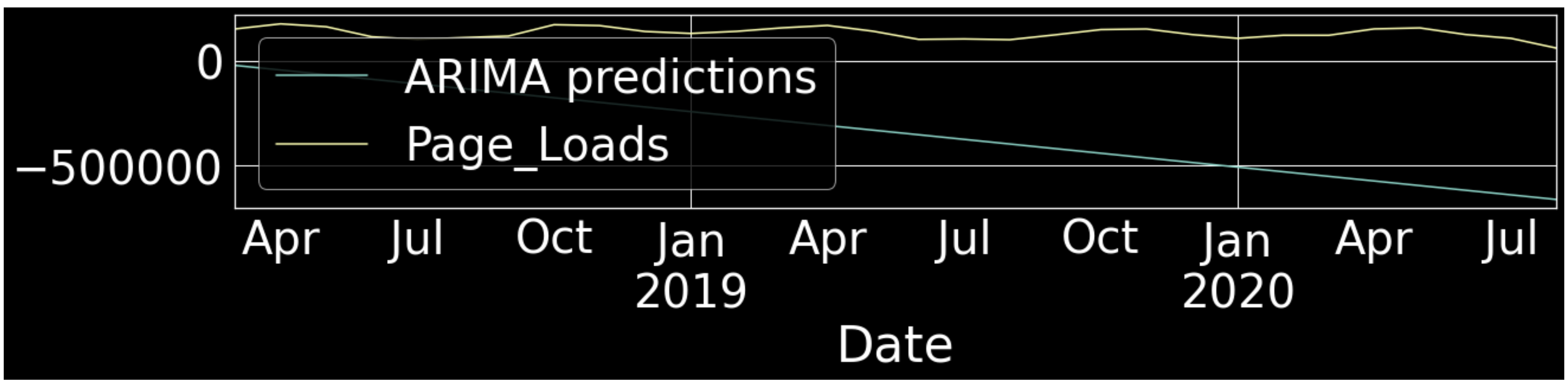
In [81]:
```
# report performance
mse = mean_squared_error(test["Page_Loads"], test["Predicted_ARIMA"])
rmse = sqrt(mse)
print('ARIMA RMSE: {}, MSE:{}'.format(rmse,mse))

##plt.title('RMSE: %.4f'% rmse)
```

```
ARIMA RMSE: 507774.5966582679, MSE:257835041011.46667
```

In [82]:
```
start=len(train)
end=len(train)+len(test)-1
#if the predicted values dont have date values as index, you will have to uncomment the following two commented lines
#index_future_dates=pd.date_range(start='2018-12-01',end='2018-12-30')
pred=model_Arima_fit.predict(start=start,end=end,typ='levels').rename('ARIMA predictions')
#pred.index=index_future_dates
pred.plot(legend=True)
test['Page_Loads'].plot(legend=True)
```

Out[82]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2cabdd1610>
```




In [83]:
```
df2.columns
```

Out[83]:
```
Index(['Page.Loads', 'Unique.Visits', 'First.Time.Visits', 'Returning.Visits'], dtype='object')
```

In [84]:
```
df2.head(3)
```

Out[84]:

| | Page.Loads | Unique.Visits | First.Time.Visits | Returning.Visits |
|---|---|---|---|---|
| **Date** | | | | |
| **2014-09-14** | 2146 | 1582 | 1430 | 152 |
| **2014-09-15** | 3621 | 2528 | 2297 | 231 |
| **2014-09-16** | 3698 | 2630 | 2352 | 278 |

In [85]:
```
df2 = df2.reset_index()
```

In [86]:
```
df2.head(3)
```

Out[86]:

| | Date | Page.Loads | Unique.Visits | First.Time.Visits | Returning.Visits |
|---|---|---|---|---|---|
| **0** | 2014-09-14 | 2146 | 1582 | 1430 | 152 |
| **1** | 2014-09-15 | 3621 | 2528 | 2297 | 231 |
| **2** | 2014-09-16 | 3698 | 2630 | 2352 | 278 |

In [87]:
```
final = df2[["Date","Page.Loads"]]
```

In [88]:
```
final.columns
```

Out[88]:
```
Index(['Date', 'Page.Loads'], dtype='object')
```

In [89]:
```
y = pd.Series(data=final['Page.Loads'].values, index=final['Date'])
```

In [90]:
```
y.head(3)
```

Out[90]:
```
Date
2014-09-14    2146
2014-09-15    3621
2014-09-16    3698
dtype: int64
```

## Grid search the `p, d, q parameters`

In [91]:
```
import itertools
# Define the p, d and q parameters to take any value between 0 and 3
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [92]:
```
warnings.filterwarnings("ignore") # specify to ignore warning messages

best_result = [0, 0, 1000]
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)

            results = mod.fit()

            print('ARIMA{} x {} - AIC: {}'.format(param, param_seasonal, results.aic))

            if results.aic < best_result[2]:
                best_result = [param, param_seasonal, results.aic]
        except:
            continue

print('\nBest Result:', best_result)
```

```
ARIMA(0, 0, 0) x (0, 0, 0, 12) - AIC: 42425.717184898036
ARIMA(0, 0, 0) x (0, 0, 1, 12) - AIC: 40851.64824104455
ARIMA(0, 0, 0) x (0, 1, 0, 12) - AIC: 38145.78754098146
ARIMA(0, 0, 0) x (0, 1, 1, 12) - AIC: 37044.05664493689
ARIMA(0, 0, 0) x (1, 0, 0, 12) - AIC: 38080.43136062496
ARIMA(0, 0, 0) x (1, 0, 1, 12) - AIC: 37277.31040369803
ARIMA(0, 0, 0) x (1, 1, 0, 12) - AIC: 37749.32223654218
ARIMA(0, 0, 0) x (1, 1, 1, 12) - AIC: 36956.07859514652
ARIMA(0, 0, 1) x (0, 0, 0, 12) - AIC: 40710.94920795676
ARIMA(0, 0, 1) x (0, 0, 1, 12) - AIC: 37771.581807463284
ARIMA(0, 0, 1) x (0, 1, 0, 12) - AIC: 36401.431809061185
ARIMA(0, 0, 1) x (0, 1, 1, 12) - AIC: 35111.40389091437
ARIMA(0, 0, 1) x (1, 0, 0, 12) - AIC: 36250.642693204965
ARIMA(0, 0, 1) x (1, 0, 1, 12) - AIC: 35391.14440267109
ARIMA(0, 0, 1) x (1, 1, 0, 12) - AIC: 35884.466502963856
ARIMA(0, 0, 1) x (1, 1, 1, 12) - AIC: 35066.32474097704
ARIMA(0, 1, 0) x (0, 0, 0, 12) - AIC: 35890.78174577265
ARIMA(0, 1, 0) x (0, 0, 1, 12) - AIC: 34885.16639554426
ARIMA(0, 1, 0) x (0, 1, 0, 12) - AIC: 37860.90404717403
ARIMA(0, 1, 0) x (0, 1, 1, 12) - AIC: 36294.68822314573
ARIMA(0, 1, 0) x (1, 0, 0, 12) - AIC: 35402.969076336805
ARIMA(0, 1, 0) x (1, 0, 1, 12) - AIC: 34884.424872536256
ARIMA(0, 1, 0) x (1, 1, 0, 12) - AIC: 37051.814098113595
ARIMA(0, 1, 0) x (1, 1, 1, 12) - AIC: 35249.199104267274
ARIMA(0, 1, 1) x (0, 0, 0, 12) - AIC: 35304.602680482
ARIMA(0, 1, 1) x (0, 0, 1, 12) - AIC: 34670.40680000931
ARIMA(0, 1, 1) x (0, 1, 0, 12) - AIC: 37118.317716272315
ARIMA(0, 1, 1) x (0, 1, 1, 12) - AIC: 34972.45745761369
ARIMA(0, 1, 1) x (1, 0, 0, 12) - AIC: 34910.486235664896
ARIMA(0, 1, 1) x (1, 0, 1, 12) - AIC: 34672.38508734617
ARIMA(0, 1, 1) x (1, 1, 0, 12) - AIC: 36157.23287649274
ARIMA(0, 1, 1) x (1, 1, 1, 12) - AIC: 34740.74008456864
ARIMA(1, 0, 0) x (0, 0, 0, 12) - AIC: 35883.807465184334
ARIMA(1, 0, 0) x (0, 0, 1, 12) - AIC: 34902.16793390615
ARIMA(1, 0, 0) x (0, 1, 0, 12) - AIC: 37345.03899148312
ARIMA(1, 0, 0) x (0, 1, 1, 12) - AIC: 35288.052190881994
ARIMA(1, 0, 0) x (1, 0, 0, 12) - AIC: 35411.4277048134
ARIMA(1, 0, 0) x (1, 0, 1, 12) - AIC: 35029.9320734105
ARIMA(1, 0, 0) x (1, 1, 0, 12) - AIC: 36639.60274432837
ARIMA(1, 0, 0) x (1, 1, 1, 12) - AIC: 35078.57653277343
ARIMA(1, 0, 1) x (0, 0, 0, 12) - AIC: 35272.22925242842
ARIMA(1, 0, 1) x (0, 0, 1, 12) - AIC: 34745.1460108378
ARIMA(1, 0, 1) x (0, 1, 0, 12) - AIC: 36229.57075185914
ARIMA(1, 0, 1) x (0, 1, 1, 12) - AIC: 34484.598122057796
```

```
ARIMA(1, 0, 1) x (1, 0, 0, 12) - AIC: 34888.19402088795
ARIMA(1, 0, 1) x (1, 0, 1, 12) - AIC: 34747.13227962806
ARIMA(1, 0, 1) x (1, 1, 0, 12) - AIC: 35519.09094533223
ARIMA(1, 0, 1) x (1, 1, 1, 12) - AIC: 34412.95268709655
ARIMA(1, 1, 0) x (0, 0, 0, 12) - AIC: 35708.171394599536
ARIMA(1, 1, 0) x (0, 0, 1, 12) - AIC: 34787.27113789217
ARIMA(1, 1, 0) x (0, 1, 0, 12) - AIC: 37738.08264166268
ARIMA(1, 1, 0) x (0, 1, 1, 12) - AIC: 35373.812247399575
ARIMA(1, 1, 0) x (1, 0, 0, 12) - AIC: 35115.198460961175
ARIMA(1, 1, 0) x (1, 0, 1, 12) - AIC: 34785.742504817164
ARIMA(1, 1, 0) x (1, 1, 0, 12) - AIC: 36627.89952571115
ARIMA(1, 1, 0) x (1, 1, 1, 12) - AIC: 34976.89381286328
ARIMA(1, 1, 1) x (0, 0, 0, 12) - AIC: 35276.82667338982
ARIMA(1, 1, 1) x (0, 0, 1, 12) - AIC: 34651.79421762709
ARIMA(1, 1, 1) x (0, 1, 0, 12) - AIC: 37067.90551756926
ARIMA(1, 1, 1) x (0, 1, 1, 12) - AIC: 34945.14396482198
ARIMA(1, 1, 1) x (1, 0, 0, 12) - AIC: 34891.79033229842
ARIMA(1, 1, 1) x (1, 0, 1, 12) - AIC: 34652.11602235254
ARIMA(1, 1, 1) x (1, 1, 0, 12) - AIC: 36140.28795494583
ARIMA(1, 1, 1) x (1, 1, 1, 12) - AIC: 34738.81469452181

Best Result: [0, 0, 1000]
```

In [93]:

```
train.head(3)
```

Out[93]:

| Date | Page_Loads | Page_Loads_First_Order_Differencing | Page_Loads_Second_Order_Differencing |
|---|---|---|---|
| **2014-09-30** | 56052 | NaN | NaN |
| **2014-10-31** | 121983 | 65931.0 | NaN |
| **2014-11-30** | 114190 | -7793.0 | -73724.0 |

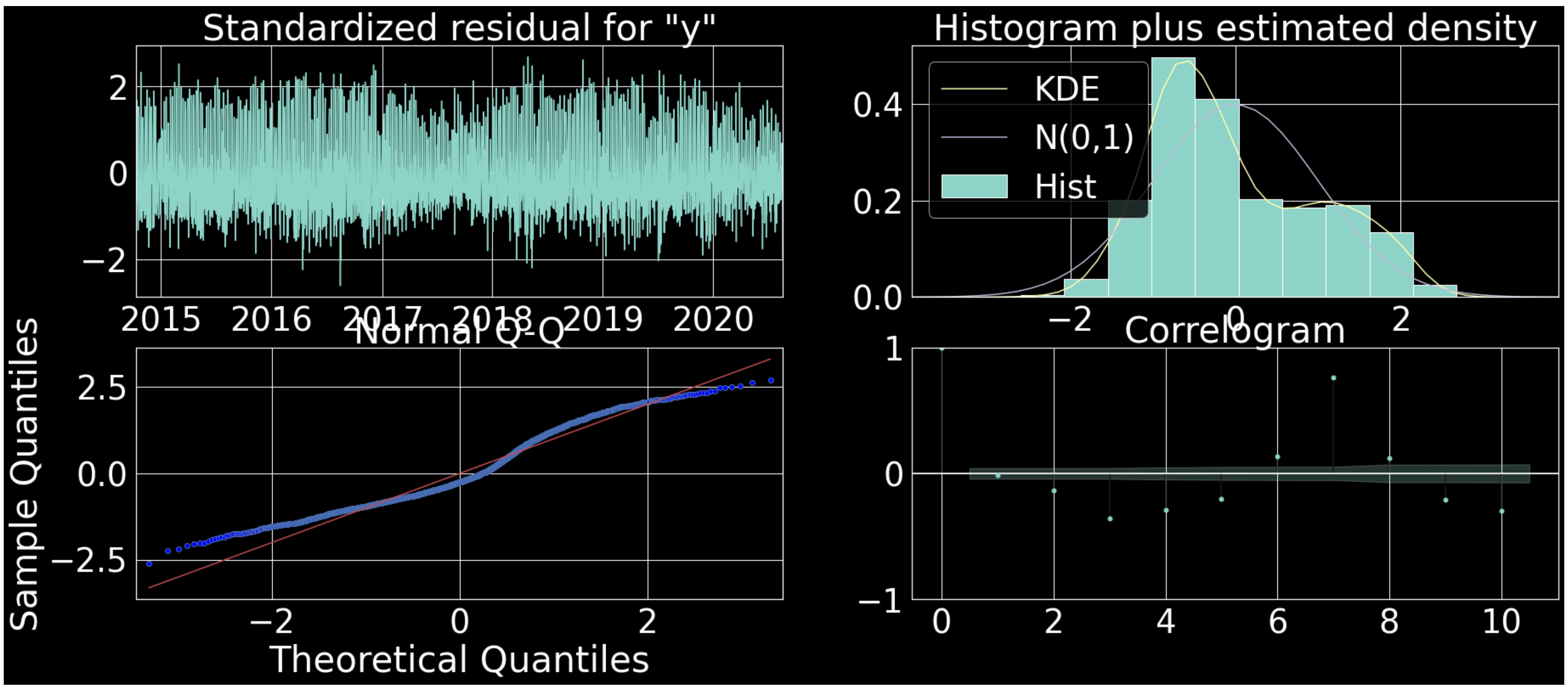
## SARIMAX

The implementation is called SARIMAX instead of SARIMA because the "X" addition to the method name means that the implementation also supports exogenous variables. Exogenous variables are optional can be specified via the "exog" argument.
model = SARIMAX(data, exog=other_data, ...)
Examples of exogenous variables: Population, holidays, number of airline companies, major events

In [94]:
```
# SARIMA example
from statsmodels.tsa.statespace.sarimax import SARIMAX

# fit model
model = SARIMAX(train["Page_Loads"], order=(1, 0, 1), seasonal_order=(0, 1, 1, 12))
model_fit = model.fit(disp=False)
```

In [95]:
```
results.plot_diagnostics(figsize=(30, 12))
plt.show()
```



In [96]:
```
print(train.shape, test.shape)
```

```
(42, 3) (30, 4)
```

In [97]:
```
start_index = test.index.min()
end_index = test.index.max()

#Predictions
predictions = model_fit.predict(start=start_index, end=end_index)
```

In [98]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, median_absolute_error, mean_squared_lo
```

In [99]:
```
from math import sqrt
```

In [100…
```
predictions.head(3)
```

Out[100]:
```
2018-03-31    128713.686417
2018-04-30    130532.723844
2018-05-31    124506.155304
Freq: M, Name: predicted_mean, dtype: float64
```

In [101…
```
test2 = test["Page_Loads"]
```

In [102…
```
test2.head(3)
```

Out[102]:
```
Date
2018-03-31    151204
2018-04-30    175572
2018-05-31    161708
Name: Page_Loads, dtype: int64
```

In [103…
```
train2 = train["Page_Loads"]
```

In [104…
```
train2.head(3)
```

Out[104]:
```
Date
2014-09-30     56052
2014-10-31    121983
2014-11-30    114190
Name: Page_Loads, dtype: int64
```

In [105…
```
# report performance
mse = mean_squared_error(test2[start_index:end_index], predictions)
rmse = sqrt(mse)
print('RMSE: {}, MSE:{}'.format(rmse,mse))
```

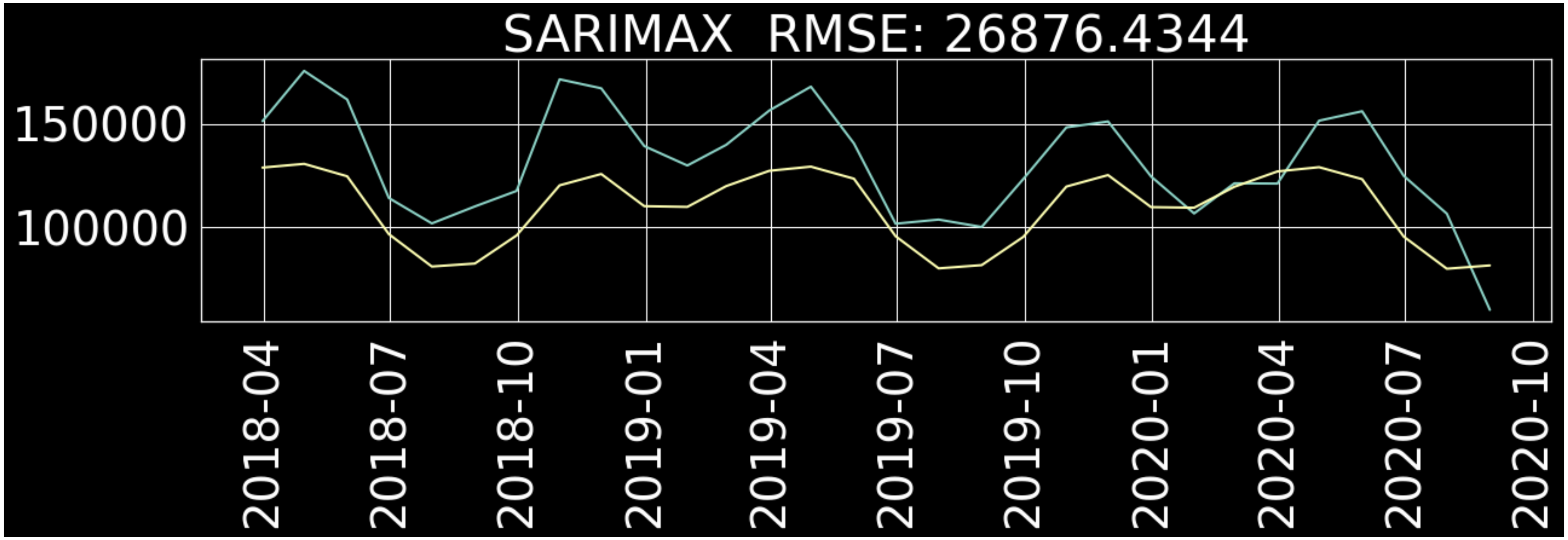
```
RMSE: 26876.43436509159, MSE:722342724.1810763
```

In [106…
```
###plt.xticks(x, labels, rotation ='vertical')
```

In [110…
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20,4)

plt.plot(test2, linewidth=2)
plt.plot(predictions, linewidth=2)
plt.xticks(rotation ='vertical')
plt.title('SARIMAX  RMSE: %.4f'% rmse)
```

Out[110]:
```
Text(0.5, 1.0, 'SARIMAX  RMSE: 26876.4344')
```




In [ ]:
```
####!pip install pmdarima
```

In [118…
```
test.head(3)
```

Out[118]:

| Date | Page_Loads | Page_Loads_First_Order_Differencing | Page_Loads_Second_Order_Differencing | Predicted_ARIMA |
|---|---|---|---|---|
| **2018-03-31** | 151204 | 29464.0 | 19440.0 | -22571.0 |
| **2018-04-30** | 175572 | 24368.0 | -5096.0 | -44621.0 |
| **2018-05-31** | 161708 | -13864.0 | -38232.0 | -66671.0 |

In [119…

```
train.head(3)
```

Out[119]:

| Date | Page_Loads | Page_Loads_First_Order_Differencing | Page_Loads_Second_Order_Differencing |
|---|---|---|---|
| **2014-09-30** | 56052 | NaN | NaN |
| **2014-10-31** | 121983 | 65931.0 | NaN |
| **2014-11-30** | 114190 | -7793.0 | -73724.0 |

In [113…

```
from pmdarima.arima import auto_arima
from pmdarima.arima import ADFTest
```

## Hyper Parameter Tuning Using AUTO-ARIMA

In [120…

```
model=auto_arima(train["Page_Loads"],start_p=0,d=1,start_q=0,
          max_p=5,max_d=5,max_q=5, start_P=0,
          D=1, start_Q=0, max_P=5,max_D=5,
          max_Q=5, m=12, seasonal=True,
          error_action='warn',trace=True,
          supress_warnings=True,stepwise=True,
          random_state=20,n_fits=50)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=632.135, Time=0.13 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=627.435, Time=0.27 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=623.573, Time=0.25 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=634.064, Time=0.05 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=623.847, Time=0.38 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=624.012, Time=0.55 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=628.068, Time=0.23 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=625.842, Time=1.18 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=623.990, Time=0.20 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=624.205, Time=0.67 sec
 ARIMA(0,1,2)(0,1,1)[12]             : AIC=625.096, Time=0.45 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=623.597, Time=0.22 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=625.334, Time=2.75 sec
 ARIMA(0,1,1)(0,1,1)[12] intercept   : AIC=624.010, Time=0.32 sec

Best model:  ARIMA(0,1,1)(0,1,1)[12]
Total fit time: 7.716 seconds
```

In [121…

```python
# SARIMA example
from statsmodels.tsa.statespace.sarimax import SARIMAX

# fit model
model_auto = SARIMAX(train["Page_Loads"], order=(0, 1, 1), seasonal_order=(0, 1, 1, 12))
model_auto_fit = model_auto.fit(disp=False)
```

In [122…

```python
model_auto_fit.summary()
```

Out[122]:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Page_Loads | **No. Observations:** | 42 |
| **Model:** | SARIMAX(0, 1, 1)x(0, 1, 1, 12) | **Log Likelihood** | -308.786 |
| **Date:** | Sun, 11 Dec 2022 | **AIC** | 623.573 |
| **Time:** | 16:31:28 | **BIC** | 627.675 |
| **Sample:** | 09-30-2014 | **HQIC** | 624.858 |
| | - 02-28-2018 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ma.L1** | 0.0660 | 0.213 | 0.310 | 0.757 | -0.351 | 0.483 |
| **ma.S.L12** | -0.6855 | 0.298 | -2.302 | 0.021 | -1.269 | -0.102 |
| **sigma2** | 9.773e+07 | 1.18e-09 | 8.29e+16 | 0.000 | 9.77e+07 | 9.77e+07 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.21 | **Jarque-Bera (JB):** | 2.21 |
| **Prob(Q):** | 0.65 | **Prob(JB):** | 0.33 |
| **Heteroskedasticity (H):** | 1.11 | **Skew:** | 0.63 |
| **Prob(H) (two-sided):** | 0.87 | **Kurtosis:** | 2.49 |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

[2] Covariance matrix is singular or near-singular, with condition number 6.68e+32. Standard errors may be unstable.

In [123…

```
print(train.shape, test.shape)
```

```
(42, 3) (30, 4)
```

In [125…

```
start_index = test.index.min()
end_index = test.index.max()

#Predictions
## pred = model_auto_fit.get_prediction(start=start_index,end=end_index, dynamic=False)
```

In [127…

```
print(start_index)
print(end_index)
```

```
2018-03-31 00:00:00
2020-08-31 00:00:00
```

In [128…

```
predictions = model_auto_fit.predict(start=start_index, end=end_index)
```

In [132…

```
print(predictions.shape, test.shape)
```

```
(30,) (30, 4)
```

In [133…

```
test.columns
```

Out[133]:

```
Index(['Page_Loads', 'Page_Loads_First_Order_Differencing',
       'Page_Loads_Second_Order_Differencing', 'Predicted_ARIMA'],
      dtype='object')
```

In [134…

```
test3 = test["Page_Loads"]
```

In [135…

```
# report performance
mse = mean_squared_error(test3[start_index:end_index], predictions)
rmse = sqrt(mse)
print('RMSE: {}, MSE:{}'.format(rmse,mse))
```

```
RMSE: 21696.720653403383, MSE:470747687.11182094
```

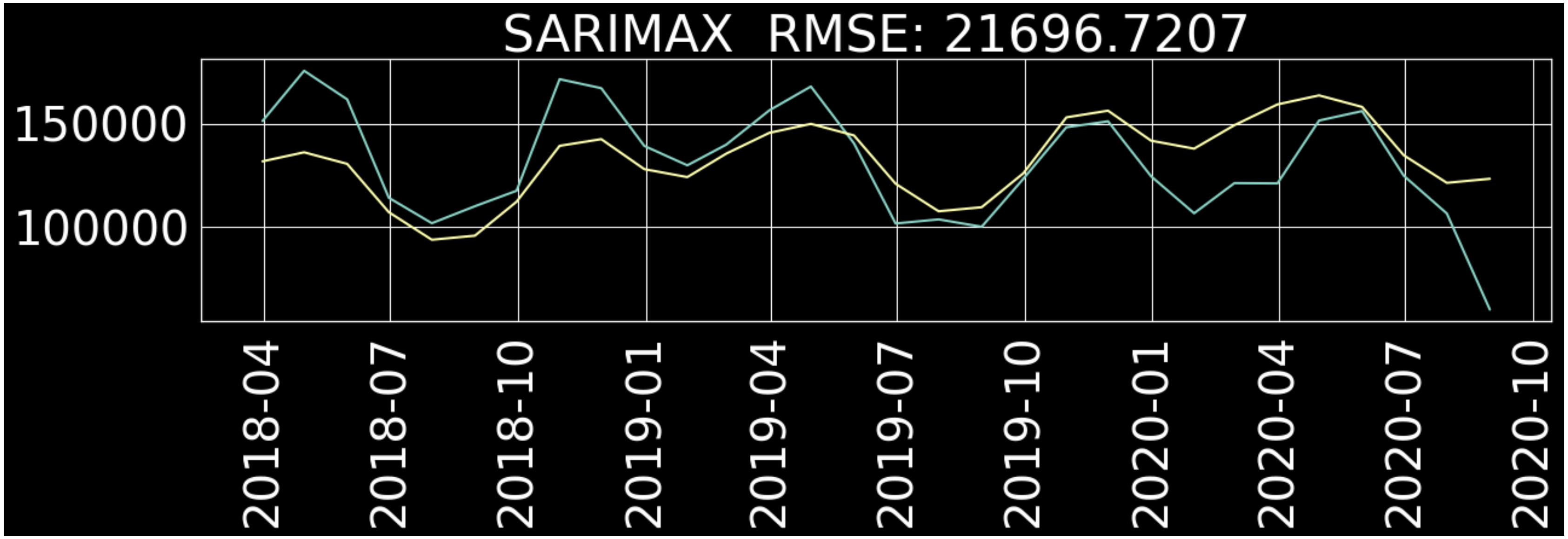
In [136…

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20,4)

plt.plot(test3, linewidth=2)
plt.plot(predictions, linewidth=2)
plt.xticks(rotation ='vertical')
plt.title('SARIMAX  RMSE: %.4f'% rmse)
```

Out[136]:

```
Text(0.5, 1.0, 'SARIMAX  RMSE: 21696.7207')
```


SARIMAX RMSE: 21696.7207
150000
100000
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
2020-01
2020-04
2020-07
2020-10

In [ ]: