whenever a data is collected over a period of Time, that is basically known as Time Series

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
f_birth=pd.read_csv('F:\Time_Series Data Analysis\data science Tutorials/daily-total-female-births-CA.csv',parse_dates=[0],index_col=0)
```

In [3]:
```
f_birth.head()
```

Out[3]:

date	births
1959-01-01	35
1959-01-02	32
1959-01-03	30
1959-01-04	31
1959-01-05	44

In [4]:
```
bb=f_birth.values
bb
```

Out[4]:
```
array([[35],
       [32],
       [30],
       [31],
       [44],
       [29],
       [45],
       [43],
       [38],
       [27],
       [38],
       [33],
       [55],
       [47],
       [45],
       [37],
       [50],
       [43],
       [41],
       [52],
       [34],
       [53],
       [39],
       [32],
       [37],
       [43],
       [39],
       [35],
       [44],
       [38],
       [24],
       [23],
       [31],
       [44],
       [38],
       [50],
       [38],
       [51],
       [31],
       [31],
       [51],
       [36],
       [45],
       [51],
       [34],
       [52],
       [47],
       [45],
       [46],
       [39],
       [48],
       [37],
       [35],
       [52],
       [42],
       [45],
       [39],
       [37],
       [30],
       [35],
       [28],
       [45],
       [34],
       [36],
       [50],
       [44],
       [39],
       [32],
       [39],
       [45],
       [43],
       [39],
       [31],
       [27],
       [30],
       [42],
       [46],
       [41],
       [36],
       [45],
       [46],
       [43],
       [38],
       [34],
       [35],
       [56],
       [36],
       [32],
       [50],
       [41],
       [39],
       [41],
       [47],
       [34],
       [36],
       [33],
       [35],
       [38],
       [38],
       [34],
       [53],
       [34],
       [34],
       [38],
       [35],
       [32],
       [42],
       [34],
       [46],
       [30],
       [46],
       [45],
       [54],
       [34],
       [37],
       [35],
       [40],
       [42],
       [58],
       [51],
       [32],
       [35],
       [38],
       [33],
       [39],
       [47],
       [38],
       [52],
       [30],
       [34],
       [40],
       [35],
       [42],
       [41],
       [42],
       [38],
       [24],
       [34],
       [43],
       [36],
       [55],
       [41],
       [45],
       [41],
       [37],
       [43],
       [39],
       [33],
       [43],
       [40],
       [38],
       [45],
       [46],
       [34],
       [35],
       [48],
       [51],
       [36],
       [33],
       [46],
       [42],
       [48],
       [34],
       [41],
       [35],
       [40],
       [34],
       [30],
       [36],
       [40],
       [39],
       [45],
       [38],
       [47],
       [33],
       [30],
       [42],
       [43],
       [41],
       [41],
       [59],
       [43],
       [45],
       [38],
       [37],
       [45],
       [42],
       [57],
       [46],
       [51],
       [41],
       [47],
       [26],
       [35],
       [44],
       [41],
       [42],
       [36],
       [45],
       [45],
       [45],
       [47],
       [38],
       [42],
       [35],
       [36],
       [39],
       [45],
       [43],
       [47],
       [36],
       [41],
       [50],
       [39],
       [41],
       [46],
       [64],
       [45],
       [34],
       [38],
       [44],
       [48],
       [46],
       [44],
       [37],
       [39],
       [44],
       [45],
       [33],
       [44],
       [38],
       [46],
       [46],
       [40],
       [39],
       [44],
       [48],
       [50],
       [41],
       [42],
       [51],
       [41],
       [44],
       [38],
       [68],
       [40],
       [42],
       [51],
       [44],
       [45],
       [36],
       [57],
       [44],
       [42],
       [53],
       [42],
       [34],
       [40],
       [56],
       [44],
       [53],
       [55],
       [39],
       [59],
       [55],
       [73],
       [55],
       [44],
       [43],
       [40],
       [47],
       [51],
       [56],
       [49],
       [54],
       [56],
       [47],
       [44],
       [43],
       [42],
       [45],
       [50],
       [48],
       [43],
       [40],
       [59],
       [41],
       [42],
       [51],
       [49],
       [45],
       [43],
       [42],
       [38],
       [47],
       [38],
       [36],
       [42],
       [35],
       [28],
       [44],
       [36],
       [45],
       [46],
       [48],
       [49],
       [43],
       [42],
       [59],
       [45],
       [52],
       [46],
       [42],
       [40],
       [40],
       [45],
       [35],
       [35],
       [40],
       [39],
       [33],
       [42],
       [47],
       [51],
       [44],
       [40],
       [57],
       [49],
       [45],
       [49],
       [51],
       [46],
       [44],
       [52],
       [45],
       [32],
       [46],
       [41],
       [34],
       [33],
       [36],
       [49],
       [43],
       [43],
       [34],
       [39],
       [35],
       [52],
       [47],
       [52],
       [39],
       [40],
       [42],
       [42],
       [53],
       [39],
       [40],
       [38],
       [44],
       [34],
       [37],
       [52],
       [48],
       [55],
       [50]], dtype=int64)
```

In [5]:
```
type(bb)
```

Out[5]: numpy.ndarray

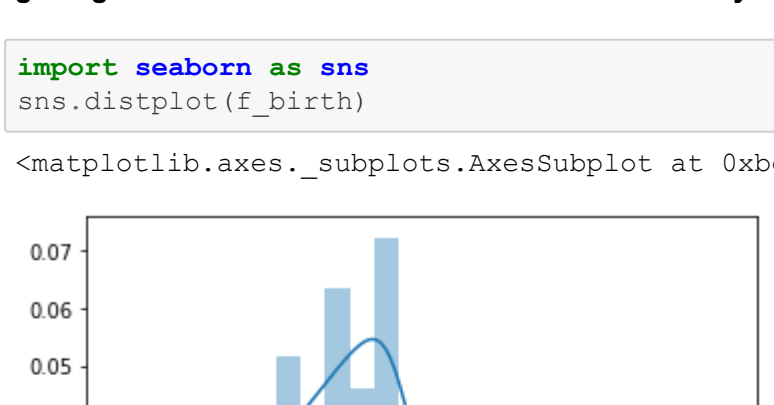
In [6]:
```
### to get size of dataframe
f_birth.size
```

Out[6]: 365

getting distribution of data to check whether I have any outlier or not

In [7]:
```
import seaborn as sns
sns.distplot(f_birth)
```

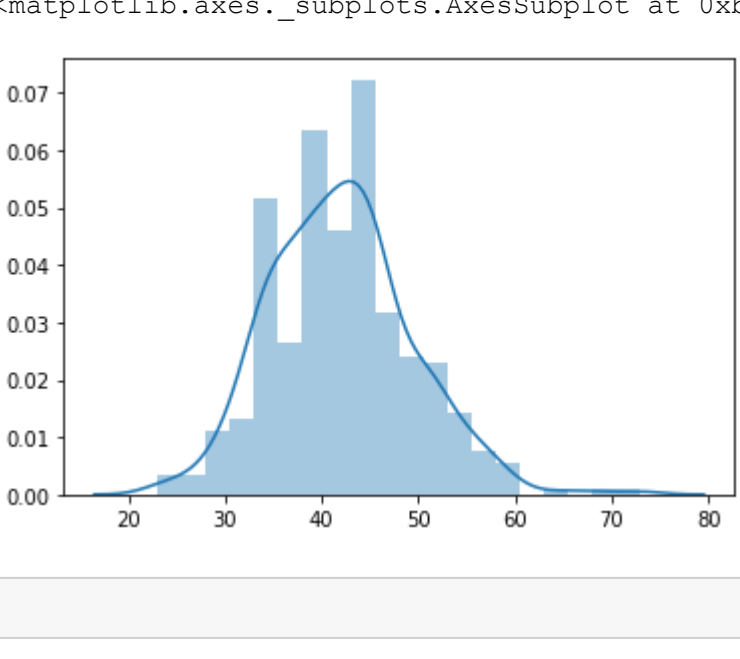
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0xbd61e77f48>



In [ ]:

In [8]:
```
### if we want to plot with respect to time or index , we can use df.plot()
f_birth.plot()
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0xbd63fede88>



this somehow a very stationery series data, what exactly is a stationery data, so data that has no trend no that has constant mean & std dev throughout data

In [ ]:

In [ ]:

In [9]:
```
f_birth.describe()
```

Out[9]:

	births
count	365.000000
mean	41.980822
std	7.348257
min	23.000000
25%	37.000000
50%	42.000000
75%	46.000000
max	73.000000

In [ ]:

In [15]:
```
f_birth.tail()
```

Out[15]:

date	births
1959-12-27	37
1959-12-28	52
1959-12-29	48
1959-12-30	55
1959-12-31	50

Smoothening your series using moving average Rolling is just like a rolling 5 days, rolling 6 days, rolling 7 days window & I can say its a window that we have considered for Moving average

In [10]:
```
f_birth_mean=f_birth.rolling(window=20).mean()
```

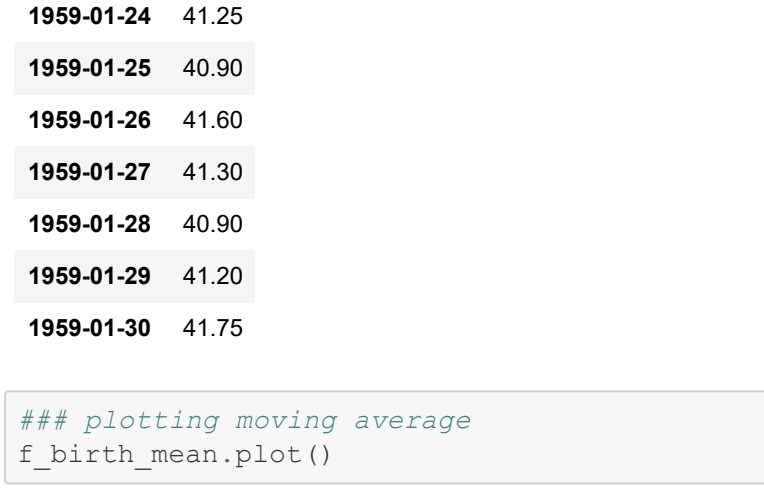
In [11]:
```
f_birth_mean.head(30)
```

Out[11]:

date	births
1959-01-01	NaN
1959-01-02	NaN
1959-01-03	NaN
1959-01-04	NaN
1959-01-05	NaN
1959-01-06	NaN
1959-01-07	NaN
1959-01-08	NaN
1959-01-09	NaN
1959-01-10	NaN
1959-01-11	NaN
1959-01-12	NaN
1959-01-13	NaN
1959-01-14	NaN
1959-01-15	NaN
1959-01-16	NaN
1959-01-17	NaN
1959-01-18	NaN
1959-01-19	NaN
1959-01-20	39.75
1959-01-21	39.70
1959-01-22	40.75
1959-01-23	41.20
1959-01-24	41.25
1959-01-25	40.90
1959-01-26	41.60
1959-01-27	41.30
1959-01-28	40.90
1959-01-29	41.20
1959-01-30	41.75

In [12]:
```
### plotting moving average
f_birth_mean.plot()
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0xbd6459ae88>



in this plot, we will see we have a spike over here in OCT & then it comes down, if we will see there is a very less trend here at all

in a similar way, i can play with window perimeter as well as window=5 or window=10

In [13]:
```
f_birth.rolling(window=5).mean().plot()
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0xbd646a3a88>

this technique has already used lots of time in Stock Market & stock Analyst used basically Moving average to remove such noise in data

In [ ]:

types of models in Time-Series :

```
1.Base line model
2.Exponential model
3.AR model
4.MA model
5.ARIMA model
6.SARIMA model(Seasonal ARIMA)
```

baseline model=naive model in which , we assume as lets say today no of births is somewhere around 41,so we assume as tommorow also we get 41 births

```
summary for baseline:--> the recent history is a best refection of the future
```

In [ ]:

In [17]:
```
value=f_birth['births']
value
```

Out[17]:
```
date
1959-01-01    35
1959-01-02    32
1959-01-03    30
1959-01-04    31
1959-01-05    44
              ..
1959-12-27    37
1959-12-28    52
1959-12-29    48
1959-12-30    55
1959-12-31    50
Name: births, Length: 365, dtype: int64
```

In [18]:
```
forecast=f_birth['births'].shift(1)
forecast
```

Out[18]:
```
date
1959-01-01     NaN
1959-01-02    35.0
1959-01-03    32.0
1959-01-04    30.0
1959-01-05    31.0
              ...
1959-12-27    34.0
1959-12-28    37.0
1959-12-29    52.0
1959-12-30    48.0
1959-12-31    55.0
Name: births, Length: 365, dtype: float64
```

In [ ]:

In [ ]:

In [ ]:

In [19]:
```
birth_df=pd.concat([value,forecast],axis=1)
birth_df.head()
```

Out[19]:

date	births	births
1959-01-01	35	NaN
1959-01-02	32	35.0
1959-01-03	30	32.0
1959-01-04	31	30.0
1959-01-05	44	31.0

In [ ]:

In [20]:
```
birth_df.columns=['actual_birth','forecast_birth']
```

In [21]:
```
birth_df.head(20)
```

Out[21]:

date	actual_birth	forecast_birth
1959-01-01	35	NaN
1959-01-02	32	35.0
1959-01-03	30	32.0
1959-01-04	31	30.0
1959-01-05	44	31.0
1959-01-06	29	44.0
1959-01-07	45	29.0
1959-01-08	43	45.0
1959-01-09	38	43.0
1959-01-10	27	38.0
1959-01-11	38	27.0
1959-01-12	33	38.0
1959-01-13	55	33.0
1959-01-14	47	55.0
1959-01-15	45	47.0
1959-01-16	37	45.0
1959-01-17	50	37.0
1959-01-18	43	50.0
1959-01-19	41	43.0
1959-01-20	52	41.0

In [ ]:

In [ ]:

Once we have done this, now we have to identify errors so there are 2 ways to figure out the error

```
1.Either u can figure it out on complete dataset
2.or we can use train_test_split
```

but in this train_test_split doesnt makes sense bcz this is a naive model,previous value is used for current value I will both the approches, and its upto what we exacty want.. if data is small, like we have 300 observations we can do it on entire data, but if it is having lets say 30000 entries , in such, it is not advisable we just take a partial data by doing some sampling or directly taking it & then go ahead & use the error..

we can computer error by importing mean_squared_error module

In [22]:
```
from sklearn.metrics import mean_squared_error
```

In [23]:
```
### then check accuracy
np.sqrt(mean_squared_error(birth_df['forecast_birth'][1:],birth_df['actual_birth'][1:]))
```

Out[23]: 9.177283229394606

why [1:] bcz we have a missing value at 1st index,thats why [1:]

why np.sqrt, bcz mean_squared_error gives square of error , and then by calling np.sqrt we get actual error

if we have outlier in data, This error will increase, dats why we have to deal with outlier

using Base-line, we can say using this basic technique we are getting error of 9 births every day, it means error of +-9 is going to happen in your prediction

In [ ]:

for this use-case we will come to ARIMA

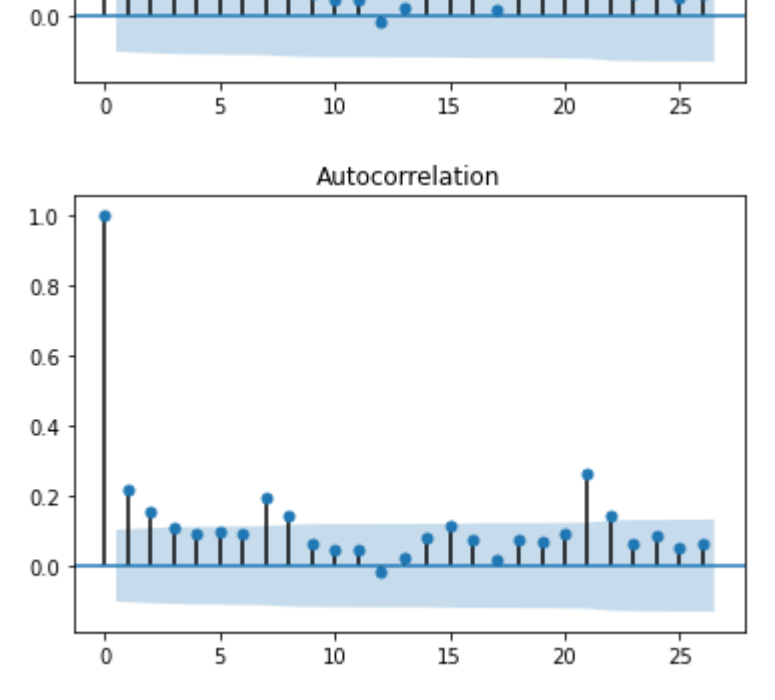
ACF,PACF

In [25]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.graphics.api import qqplot
```

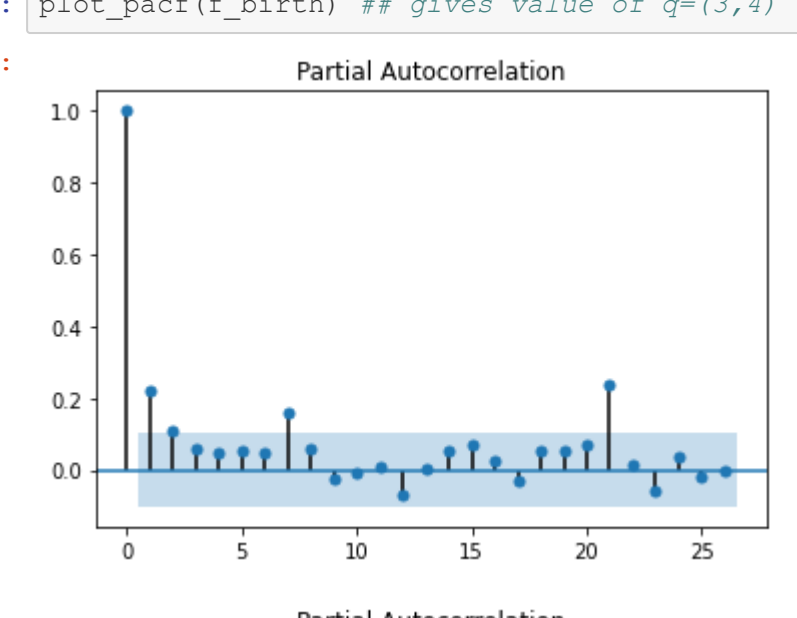
to get your q value for your ARIMA ,i am going to visualise Auto-corelation chart

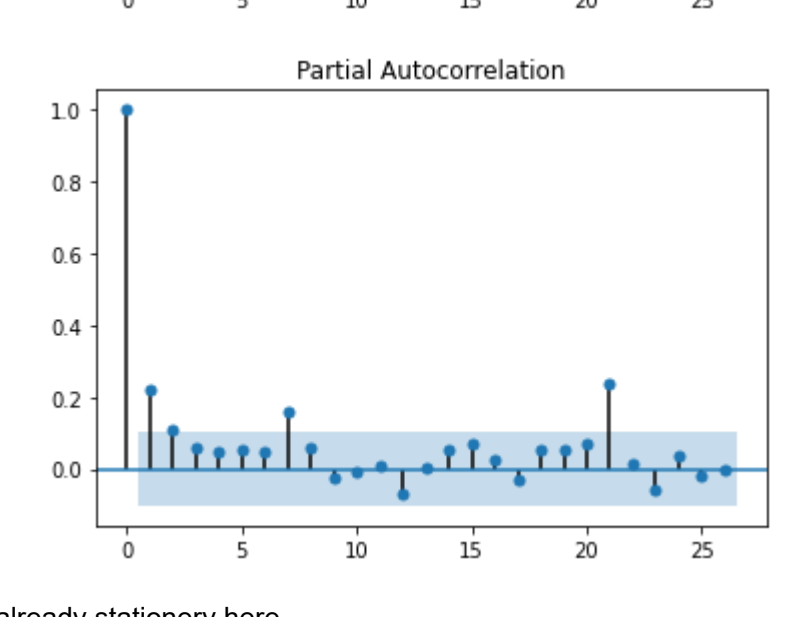
In [26]:
```
plot_acf(f_birth['births']) ## p=2,3
```

Out[26]:



In [27]:
```
plot_pacf(f_birth) ## gives value of q=(3,4)
```
Out[27]:





d=0 as already stationery here

In [56]:
```
training_data=f_birth[0:320]
test_data=f_birth[320:]
### create test that is basically unseen to our model, or which is basically is used for evalaution
```
In [57]:
```
training_data.size
```
Out[57]: 320

In [58]:
```
test_data.size
```
Out[58]: 45

In [31]:
```
from statsmodels.tsa.arima_model import ARIMA
```
In [61]:
```
### arima= ARIMA(training_data,order=(p,d,q))

arima= ARIMA(training_data,order=(2,1,3))
```
```
C:\Users\mcr\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarning: No freq
uency information was provided, so inferred frequency D will be used.
  % freq, ValueWarning)
C:\Users\mcr\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarning: No freq
uency information was provided, so inferred frequency D will be used.
  % freq, ValueWarning)
```
In [62]:
```
### then doing model.fit ie fitting our model to data
model=arima.fit()
```
In [34]:
```
model.aic
```
Out[34]: 2164.700025508478

In [35]:
```
pred= model.forecast(steps=45)[0]
```
In [36]:
```
pred
```
Out[36]:
```
array([43.11819215, 44.26392629, 43.64786207, 43.03747563, 44.0806814 ,
       43.97670631, 43.10917132, 43.84816621, 44.19618218, 43.30274198,
       43.63438843, 44.28805863, 43.56679434, 43.49198806, 44.2643378 ,
       43.84351657, 43.44959365, 44.1596409 , 44.0820681 , 43.50973978,
       44.02004897, 44.24824448, 43.65269193, 43.89136884, 44.32898152,
       43.84437578, 43.80942045, 44.33160219, 44.04603501, 43.7940763 ,
       44.27884637, 44.22327109, 43.84769545, 44.20144747, 44.35264253,
       43.9575505 , 44.13024792, 44.42482605, 44.10106258, 44.08961034,
       44.44424113, 44.25225942, 44.09331119, 44.42580973, 44.3878761 ])
```
In [37]:
```
test_data
```
Out[37]:

date	births
1959-11-17	33
1959-11-18	42
1959-11-19	47
1959-11-20	51
1959-11-21	44
1959-11-22	40
1959-11-23	57
1959-11-24	49
1959-11-25	45
1959-11-26	49
1959-11-27	51
1959-11-28	46
1959-11-29	44
1959-11-30	52
1959-12-01	45
1959-12-02	32
1959-12-03	46
1959-12-04	41
1959-12-05	34
1959-12-06	33
1959-12-07	36
1959-12-08	49
1959-12-09	43
1959-12-10	43
1959-12-11	34
1959-12-12	39
1959-12-13	35
1959-12-14	52
1959-12-15	47
1959-12-16	52
1959-12-17	39
1959-12-18	40
1959-12-19	42
1959-12-20	42
1959-12-21	53
1959-12-22	39
1959-12-23	40
1959-12-24	38
1959-12-25	44
1959-12-26	34
1959-12-27	37
1959-12-28	52
1959-12-29	48
1959-12-30	55
1959-12-31	50

In [39]:
```
#### now we can compare our prediction in some cases, we are doing very well whereas in some cases, we
 are not doing good prediction
```
In [38]:
```
np.sqrt(mean_squared_error(test_data,pred))
```
Out[38]: 6.613037458913119

In [ ]:

now u will see it is better than the model that we have created as Error decreases from 9 to 6.6

and in the future we are going to play with diff pairs of (P,d,q) or we can choose best pair by just automating it using Hyperparameter tuning

In [ ]:

In [ ]:

another way of decreasing more error is by just normalizing & standarizing data

In [39]:
```
f_birth.head()
```
Out[39]:

date	births
1959-01-01	35
1959-01-02	32
1959-01-03	30
1959-01-04	31
1959-01-05	44

Normalization- useful in 2 cases-

```
 1. if ur data is on different scale, Lets say height of 100 students are on different scale th
en in such scenario, we can bring our data to same scale,after normalize our data comes in a ran
ge of [0,1]

 2.bcz some algo works better work better if ur data is normalized lets say Linear Regression,
Logistic Regression
```

normalization= (xi-min) / (max-min), we can create our own function to perform normalization but scikit provides us in-built func to perform normalization

In [40]:
```
f_birth.min()
```
Out[40]:
```
births    23
dtype: int64
```
In [41]:
```
f_birth.max()
```
Out[41]:
```
births    73
dtype: int64
```
In [42]:
```
f_birth
```
Out[42]:

date	births
1959-01-01	35
1959-01-02	32
1959-01-03	30
1959-01-04	31
1959-01-05	44
...	...
1959-12-27	37
1959-12-28	52
1959-12-29	48
1959-12-30	55
1959-12-31	50

365 rows × 1 columns

In [43]:
```
(35-23)/(72-23)
```
Out[43]: 0.24489795918367346

35 getting normalized to 0.24489

In [44]:
```
from sklearn.preprocessing import MinMaxScaler
```
In [45]:
```
data=f_birth.values
```

as MinMaxScaler receives Data in the form of array

In [46]:
```
scaler=MinMaxScaler()
```
In [47]:
```
birth_normalize=scaler.fit_transform(f_birth)
```
In [48]:
```
birth_normalize
```
Out[48]:
```
array([[0.24],
       [0.18],
       [0.14],
       [0.16],
       [0.42],
       [0.12],
       [0.44],
       [0.4 ],
       [0.3 ],
       [0.08],
       [0.3 ],
       [0.2 ],
       [0.64],
       [0.48],
       [0.44],
       [0.28],
       [0.54],
       [0.4 ],
       [0.36],
       [0.58],
       [0.22],
       [0.6 ],
       [0.32],
       [0.18],
       [0.28],
       [0.4 ],
       [0.32],
       [0.24],
       [0.42],
       [0.3 ],
       [0.02],
       [0.  ],
       [0.16],
       [0.42],
       [0.3 ],
       [0.54],
       [0.3 ],
       [0.56],
       [0.16],
       [0.16],
       [0.56],
       [0.26],
       [0.44],
       [0.56],
       [0.22],
       [0.58],
       [0.48],
       [0.44],
       [0.46],
       [0.32],
       [0.5 ],
       [0.28],
       [0.24],
       [0.58],
       [0.38],
       [0.44],
       [0.32],
       [0.28],
       [0.14],
       [0.24],
       [0.1 ],
       [0.44],
       [0.22],
       [0.26],
       [0.54],
       [0.42],
       [0.32],
       [0.18],
       [0.32],
       [0.44],
       [0.4 ],
       [0.32],
       [0.16],
       [0.08],
       [0.14],
       [0.38],
       [0.46],
       [0.36],
       [0.26],
       [0.44],
       [0.46],
       [0.4 ],
       [0.3 ],
       [0.22],
       [0.24],
       [0.66],
       [0.26],
       [0.18],
       [0.54],
       [0.36],
       [0.32],
       [0.36],
       [0.48],
       [0.22],
       [0.26],
       [0.2 ],
       [0.24],
       [0.3 ],
       [0.3 ],
       [0.22],
       [0.6 ],
       [0.22],
       [0.22],
       [0.3 ],
       [0.24],
       [0.18],
       [0.38],
       [0.22],
       [0.46],
       [0.14],
       [0.46],
       [0.44],
       [0.62],
       [0.22],
       [0.28],
       [0.24],
       [0.34],
       [0.38],
       [0.7 ],
       [0.56],
       [0.18],
       [0.24],
       [0.3 ],
       [0.2 ],
       [0.32],
       [0.48],
       [0.3 ],
       [0.58],
       [0.14],
       [0.22],
       [0.34],
       [0.24],
       [0.38],
       [0.36],
       [0.38],
       [0.3 ],
       [0.02],
       [0.22],
       [0.4 ],
       [0.26],
       [0.64],
       [0.36],
       [0.44],
       [0.36],
       [0.28],
       [0.4 ],
       [0.32],
       [0.2 ],
       [0.4 ],
       [0.34],
       [0.3 ],
       [0.44],
       [0.46],
       [0.22],
       [0.24],
       [0.5 ],
       [0.56],
       [0.26],
       [0.2 ],
       [0.46],
       [0.38],
       [0.5 ],
       [0.22],
       [0.36],
       [0.24],
       [0.34],
       [0.22],
       [0.14],
       [0.26],
       [0.34],
       [0.32],
       [0.44],
       [0.3 ],
       [0.48],
       [0.2 ],
       [0.14],
       [0.38],
       [0.4 ],
       [0.36],
       [0.36],
       [0.72],
       [0.4 ],
       [0.44],
       [0.3 ],
       [0.28],
       [0.44],
       [0.38],
       [0.68],
       [0.46],
       [0.56],
       [0.36],
       [0.48],
       [0.06],
       [0.24],
       [0.42],
       [0.36],
       [0.38],
       [0.26],
       [0.44],
       [0.44],
       [0.44],
       [0.48],
       [0.3 ],
       [0.38],
       [0.24],
       [0.26],
       [0.32],
       [0.44],
       [0.4 ],
       [0.48],
       [0.26],
       [0.36],
       [0.54],
       [0.32],
       [0.36],
       [0.46],
       [0.82],
       [0.44],
       [0.22],
       [0.3 ],
       [0.42],
       [0.5 ],
       [0.46],
       [0.42],
       [0.28],
       [0.32],
       [0.42],
       [0.44],
       [0.2 ],
       [0.42],
       [0.3 ],
       [0.46],
       [0.46],
       [0.34],
       [0.32],
       [0.42],
       [0.5 ],
       [0.54],
       [0.36],
       [0.38],
       [0.56],
       [0.36],
       [0.42],
       [0.3 ],
       [0.9 ],
       [0.34],
       [0.38],
       [0.56],
       [0.42],
       [0.44],
       [0.26],
       [0.68],
       [0.42],
       [0.38],
       [0.6 ],
       [0.38],
       [0.22],
       [0.34],
       [0.66],
       [0.42],
       [0.6 ],
       [0.64],
       [0.32],
       [0.72],
       [0.64],
       [1.  ],
       [0.64],
       [0.42],
       [0.4 ],
       [0.34],
       [0.48],
       [0.56],
       [0.66],
       [0.52],
       [0.62],
       [0.66],
       [0.48],
       [0.42],
       [0.4 ],
       [0.38],
       [0.44],
       [0.54],
       [0.5 ],
       [0.4 ],
       [0.34],
       [0.72],
       [0.36],
       [0.38],
       [0.56],
       [0.52],
       [0.44],
       [0.4 ],
       [0.38],
       [0.3 ],
       [0.48],
       [0.3 ],
       [0.26],
       [0.38],
       [0.24],
       [0.1 ],
       [0.42],
       [0.26],
       [0.44],
       [0.46],
       [0.5 ],
       [0.52],
       [0.4 ],
       [0.38],
       [0.72],
       [0.44],
       [0.58],
       [0.46],
       [0.38],
       [0.34],
       [0.34],
       [0.44],
       [0.24],
       [0.24],
       [0.34],
       [0.32],
       [0.2 ],
       [0.38],
       [0.48],
       [0.56],
       [0.42],
       [0.34],
       [0.68],
       [0.52],
       [0.44],
       [0.52],
       [0.56],
       [0.46],
       [0.42],
       [0.58],
       [0.44],
       [0.18],
       [0.46],
       [0.36],
       [0.22],
       [0.2 ],
       [0.26],
       [0.52],
       [0.4 ],
       [0.4 ],
       [0.22],
       [0.32],
       [0.24],
       [0.58],
       [0.48],
       [0.58],
       [0.32],
       [0.34],
       [0.38],
       [0.38],
       [0.6 ],
       [0.32],
       [0.34],
       [0.3 ],
       [0.42],
       [0.22],
       [0.28],
       [0.58],
       [0.5 ],
       [0.64],
       [0.54]])
```
In [49]:
```
birth_normalize[0]
```
Out[49]: array([0.24])

In [ ]:

so now my data has been normalized so now we are ready to insert this data into algorithm

In [66]:
```
training_data=birth_normalize[0:320]
test_data=birth_normalize[320:]
```
In [70]:
```
training_data[0:5]
```
Out[70]:
```
array([[0.24],
       [0.18],
       [0.14],
       [0.16],
       [0.42]])
```
In [67]:
```
training_data.size
```
Out[67]: 320

In [68]:
```
test_data.size
```
Out[68]: 45

In [53]:
```
from statsmodels.tsa.arima_model import ARIMA
```
In [71]:
```
### arima= ARIMA(training_data,order=(p,d,q))

arima= ARIMA(training_data,order=(2,1,3))
```
In [72]:
```
model=arima.fit()
```
```
C:\Users\mcr\Anaconda3\lib\site-packages\statsmodels\base\model.py:548: HessianInversionWarning: Inve
rting hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
```
In [73]:
```
model.aic
```
Out[73]: -325.955725078043

now we will see -ve aic, earlier we have positive aic

In [74]:
```
pred= model.forecast(steps=45)[0]
```
In [75]:
```
pred
```
Out[75]:
```
array([0.39993182, 0.41298825, 0.41447656, 0.40199793, 0.40499743,
       0.41700022, 0.41119326, 0.40192801, 0.4110888 , 0.41810889,
       0.40777774, 0.40473263, 0.41644852, 0.41662979, 0.40585153,
       0.40961813, 0.41973087, 0.41374084, 0.40637596, 0.41514647,
       0.42042779, 0.4109984 , 0.40938109, 0.41978891, 0.41896768,
       0.4097669 , 0.41404003, 0.42246075, 0.41648278, 0.41076648,
       0.41903154, 0.42285445, 0.41435531, 0.41388353, 0.42303997,
       0.4214749 , 0.41371878, 0.41828564, 0.42520776, 0.41939172,
       0.41509025, 0.42277353, 0.42538913, 0.4178196 , 0.41824835])
```
In [76]:
```
from sklearn.metrics import mean_squared_error
```
In [77]:
```
np.sqrt(mean_squared_error(test_data,pred))
```
Out[77]: 0.13100596695951017

very less error

In [ ]:

now to get your actual forecast, u have to perform inverse transformation

In [78]:
```
actual_forecast=scaler.inverse_transform(pred.reshape(-1,1))
```
In [79]:
```
actual_forecast
```
Out[79]:
```
array([[42.99659081],
       [43.64941247],
       [43.72382794],
       [43.09989665],
       [43.24987149],
       [43.85001076],
       [43.55966322],
       [43.09640055],
       [43.55443998],
       [43.90544458],
       [43.38888685],
       [43.23663138],
       [43.82242615],
       [43.83148939],
       [43.29257647],
       [43.48090675],
       [43.98654375],
       [43.68704194],
       [43.31879803],
       [43.75732332],
       [44.02138945],
       [43.54991975],
       [43.46905472],
       [43.98944539],
       [43.94838376],
       [43.48834488],
       [43.70200159],
       [44.12303747],
       [43.82413884],
       [43.5383238 ],
       [43.95157721],
       [44.14272249],
       [43.71776552],
       [43.69417667],
       [44.15199852],
       [44.07374498],
       [43.6859389 ],
       [43.91428221],
       [44.26038793],
       [43.9695859 ],
       [43.75451258],
       [44.13867674],
       [44.26945629],
       [43.89097998],
       [43.91241738]])
```

to get your actual error

In [80]:
```
np.sqrt(mean_squared_error(scaler.inverse_transform(test_data),actual_forecast))
```
Out[80]: 6.550298347975507

now we will see whether after Data transformation our error reduces just a little bit

In [ ]:

In [ ]:

In [ ]:

feature engineering on Time SEries Data

In [81]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```
In [82]:
```
f_birth=pd.read_csv('F:\Time_Series Data Analysis\data science Tutorials/daily-total-female-births-CA.c
sv',parse_dates=[0],index_col=0)
```
In [83]:
```
f_birth.head()
```
Out[83]:

date	births
1959-01-01	35
1959-01-02	32
1959-01-03	30
1959-01-04	31
1959-01-05	44

In [84]:
```
f_birth['lag1']=f_birth.shift(1)
```
In [86]:
```
f_birth.head()
```
Out[86]:

date	births	lag1
1959-01-01	35	NaN
1959-01-02	32	35.0
1959-01-03	30	32.0
1959-01-04	31	30.0
1959-01-05	44	31.0

In [87]:
```
f_birth['lag2']=f_birth['births'].shift(2)
```
In [88]:
```
f_birth['lag3']=f_birth['births'].shift(3)
```

In [89]:
```
f_birth.head()
```
Out[89]:

date	births	lag1	lag2	lag3
1959-01-01	35	NaN	NaN	NaN
1959-01-02	32	35.0	NaN	NaN
1959-01-03	30	32.0	35.0	NaN
1959-01-04	31	30.0	32.0	35.0
1959-01-05	44	31.0	30.0	32.0

moving average

In [90]:
```
f_birth['MA3']=f_birth['births'].rolling(window=3).mean()

# window: This is the number of observations used for calculating the statistic.
```
In [ ]:

In [91]:
```
f_birth.head()
```
Out[91]:

date	births	lag1	lag2	lag3	MA3
1959-01-01	35	NaN	NaN	NaN	NaN
1959-01-02	32	35.0	NaN	NaN	NaN
1959-01-03	30	32.0	35.0	NaN	32.333333
1959-01-04	31	30.0	32.0	35.0	31.000000
1959-01-05	44	31.0	30.0	32.0	35.000000

In [92]:
```
f_birth['MA4']=f_birth['births'].rolling(window=4).mean()
```
In [93]:
```
f_birth.head()
```
Out[93]:

date	births	lag1	lag2	lag3	MA3	MA4
1959-01-01	35	NaN	NaN	NaN	NaN	NaN
1959-01-02	32	35.0	NaN	NaN	NaN	NaN
1959-01-03	30	32.0	35.0	NaN	32.333333	NaN
1959-01-04	31	30.0	32.0	35.0	31.000000	32.00
1959-01-05	44	31.0	30.0	32.0	35.000000	34.25

In [ ]:

In [98]:
```
##### lets day u r expertise in any area & u kmow within 7 days, the maximum value is the true reflecto
r of our business as we are in high growth era
##### & minimum value is a true reflector of our baseline, so in such case instead of mean , u can call
 min & max function
```
In [ ]:

In [94]:
```
#### lets say 5 is the window size or 5 is the business period
f_birth['MAX_5']=f_birth['births'].rolling(window=5).max()
f_birth.head()
```
Out[94]:

date	births	lag1	lag2	lag3	MA3	MA4	MAX_5
1959-01-01	35	NaN	NaN	NaN	NaN	NaN	NaN
1959-01-02	32	35.0	NaN	NaN	NaN	NaN	NaN
1959-01-03	30	32.0	35.0	NaN	32.333333	NaN	NaN
1959-01-04	31	30.0	32.0	35.0	31.000000	32.00	NaN
1959-01-05	44	31.0	30.0	32.0	35.000000	34.25	44.0

In [95]:
```
f_birth['MIN_5']=f_birth['births'].rolling(window=5).min()
f_birth.head()
```
Out[95]:

date	births	lag1	lag2	lag3	MA3	MA4	MAX_5	MIN_5
1959-01-01	35	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1959-01-02	32	35.0	NaN	NaN	NaN	NaN	NaN	NaN
1959-01-03	30	32.0	35.0	NaN	32.333333	NaN	NaN	NaN
1959-01-04	31	30.0	32.0	35.0	31.000000	32.00	NaN	NaN
1959-01-05	44	31.0	30.0	32.0	35.000000	34.25	44.0	30.0

In [1]:
```
### now we will see just having a basic knowledge of Lag and some subject matter expertise, we can crea
te so many variables
```
In [ ]:
```
###### to more u will dig in data & business,more u will figure it about feature enginering
###### u can say ,feature engineering is a most creative part apart from intrepreting the model
```
In [101]:
```
### lets say to capture day from date
```
In [96]:
```
f_birth['day']=f_birth.index.day
```
In [97]:
```
f_birth.head()
```
Out[97]:

date	births	lag1	lag2	lag3	MA3	MA4	MAX_5	MIN_5	day
1959-01-01	35	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1
1959-01-02	32	35.0	NaN	NaN	NaN	NaN	NaN	NaN	2
1959-01-03	30	32.0	35.0	NaN	32.333333	NaN	NaN	NaN	3
1959-01-04	31	30.0	32.0	35.0	31.000000	32.00	NaN	NaN	4
1959-01-05	44	31.0	30.0	32.0	35.000000	34.25	44.0	30.0	5

In [98]:
```
f_birth['month']=f_birth.index.month
```
In [99]:
```
f_birth.head()
```
Out[99]:

date	births	lag1	lag2	lag3	MA3	MA4	MAX_5	MIN_5	day	month
1959-01-01	35	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1	1
1959-01-02	32	35.0	NaN	NaN	NaN	NaN	NaN	NaN	2	1
1959-01-03	30	32.0	35.0	NaN	32.333333	NaN	NaN	NaN	3	1
1959-01-04	31	30.0	32.0	35.0	31.000000	32.00	NaN	NaN	4	1
1959-01-05	44	31.0	30.0	32.0	35.000000	34.25	44.0	30.0	5	1

we can also create dummy variable if we want to explore our data more

In [ ]:

In [100]:
```
f_birth['year']=f_birth.index.year
```
In [101]:
```
f_birth.head()
```
Out[101]:

date	births	lag1	lag2	lag3	MA3	MA4	MAX_5	MIN_5	day	month	year
1959-01-01	35	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1	1	1959
1959-01-02	32	35.0	NaN	NaN	NaN	NaN	NaN	NaN	2	1	1959
1959-01-03	30	32.0	35.0	NaN	32.333333	NaN	NaN	NaN	3	1	1959
1959-01-04	31	30.0	32.0	35.0	31.000000	32.00	NaN	NaN	4	1	1959
1959-01-05	44	31.0	30.0	32.0	35.000000	34.25	44.0	30.0	5	1	1959

In [108]:
```
#### remove na values
```
In [103]:
```
f_birth.dropna(inplace=True)
```
In [ ]:

4. Stationery Time series Testing

test whether your time-series has been stationery or not

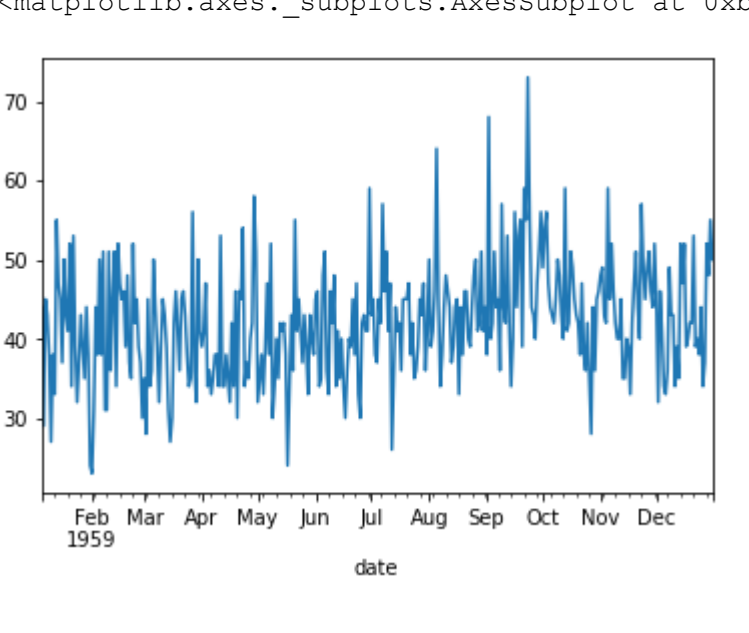
bcz stationery time-series helps us to make prediction better

stationery is all about when ur mean & variance is constant over a period of time

1.visualizing Time series 2.Dickey Fuller Test(Statistical Test) 3.Constant mean & variance

getting trend

In [104]:
```
f_birth['births'].plot()
```
Out[104]: <matplotlib.axes._subplots.AxesSubplot at 0xbd66839288>

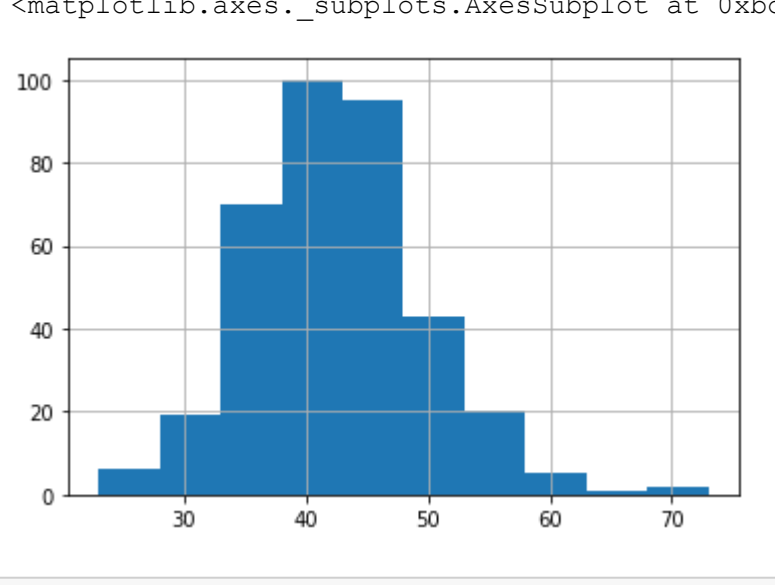


now here we will see there is a bit of trend or seasonality here, as variance is not constant here

from this we will visualise it doesnt have a srong seasonlity but a bit of seasonality

In [ ]:

In [107]:
```
f_birth['births'].hist()
```
Out[107]: <matplotlib.axes._subplots.AxesSubplot at 0xbd67acdc08>



In [ ]:

looks to normal distribution but little bit skewes to right side

In [108]:
```
f_birth_1=f_birth[0:201]
f_birth_2=f_birth[201:]
```
In [109]:
```
f_birth_1.head()
```
Out[109]:

date	births	lag1	lag2	lag3	MA3	MA4	MAX_5	MIN_5	day	month	year
1959-01-05	44	31.0	30.0	32.0	35.000000	34.25	44.0	30.0	5	1	1959
1959-01-06	29	44.0	31.0	30.0	34.666667	33.50	44.0	29.0	6	1	1959
1959-01-07	45	29.0	44.0	31.0	39.333333	37.25	45.0	29.0	7	1	1959
1959-01-08	43	45.0	29.0	44.0	39.000000	40.25	45.0	29.0	8	1	1959
1959-01-09	38	43.0	45.0	29.0	42.000000	38.75	45.0	29.0	9	1	1959

In [110]:
```
f_birth_2.head()
```
Out[110]:

date	births	lag1	lag2	lag3	MA3	MA4	MAX_5	MIN_5	day	month	year
1959-07-25	36	35.0	42.0	38.0	37.666667	37.75	47.0	35.0	25	7	1959
1959-07-26	39	36.0	35.0	42.0	36.666667	38.00	42.0	35.0	26	7	1959
1959-07-27	45	39.0	36.0	35.0	40.000000	38.75	45.0	35.0	27	7	1959
1959-07-28	43	45.0	39.0	36.0	42.333333	40.75	45.0	35.0	28	7	1959
1959-07-29	47	43.0	45.0	39.0	45.000000	43.50	47.0	36.0	29	7	1959

In [ ]:

In [111]:
```
f_birth_1['births'].mean()
```
Out[111]: 40.19402985074627

In [112]:
```
f_birth_2['births'].mean()
```
Out[112]: 44.475

we know our assumption for our stationery time-series is that mean should be same, but here mean is variating

so we can say it is not stationery lets run another test to check whether it is stationery or not

In [113]:
```
f_birth_1['births'].var()
```
Out[113]: 48.33716417910449

In [114]:
```
f_birth_2['births'].var()
```
Out[114]: 49.92389937106918

In [ ]:

hence by visualising graph we can conclude yahh it has a bit of seasonality with a constant variance

In [ ]:

Next is Dickey Fuller Test

```
##### dickery fuller test or Augumented Dickey Fuller(adfuller) test, so this test basically tel
ls us how strongly Time-series is defined by  a trend
```

In [ ]:
```
#### adfuller gives 5 values ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Use
d']
#most imp is p-value
#this is almost similar to Hypothesis testing,whereas Null hypo which says Data is not Stationery
#whereas Alternate hypo says data is stationery

#lags are those which are our previous Data
```
In [115]:
```
from statsmodels.tsa.stattools import adfuller
```
In [ ]:
```
#### apply adfuller on dataframe
```
In [ ]:

In [119]:
```
def adfuller_test(column):
    result=adfuller(column)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(' {}: {}'.format(label,value))
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has n
o unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is no
n-stationary ")
      #Ho: ie null hypoIt is non stationary
#H1: ie Alternate Hypo ,It is stationary
#zip basically combines result,labels
```
In [120]:
```
adfuller_test(f_birth_1['births'])
```
```
 ADF Test Statistic: -13.198304128015673
 p-value: 1.1076885620359273e-24
 #Lags Used: 0
 Number of Observations Used: 200
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root an
d is stationary
```
In [ ]:
```
#adfuller gives 5 values ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
#most imp is p-value
#this is almost similar to Hypothesis testing,whereas Null hypo which says Data is not Stationery
#whereas Alternate hypo says data is stationery
```

```
if Pvalue<0.05 ,we reject Null hypo ie alternate hypo is true,ie data is stationery
```

In [ ]:

In [ ]:

In [ ]:

tune your parameter of time-series (Hyper-parameter Tuning)

all about playing with params (P,d,q) so whichever pair will give less error select those pair

In [121]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```
In [122]:
```
from statsmodels.tsa.arima_model import ARIMA
```
In [123]:
```
from sklearn.metrics import mean_squared_error
```
In [128]:
```
f_birth=pd.read_csv('F:\Time_Series Data Analysis\data science Tutorials/daily-total-female-births-CA.
csv',parse_dates=[0],index_col=0)
```
In [124]:
```
#### now define hyper-para=meters
p_values=range(0,8)
d_values=range(0,3)
q_values=range(0,5)
```
In [125]:
```
for p in p_values:
    for d in d_values:
        for q in q_values:
            order=(p,d,q)
            print((order))
```
```
(0, 0, 0)
(0, 0, 1)
(0, 0, 2)
(0, 0, 3)
(0, 0, 4)
(0, 1, 0)
(0, 1, 1)
(0, 1, 2)
(0, 1, 3)
(0, 1, 4)
(0, 2, 0)
(0, 2, 1)
(0, 2, 2)
(0, 2, 3)
(0, 2, 4)
(1, 0, 0)
(1, 0, 1)
(1, 0, 2)
(1, 0, 3)
(1, 0, 4)
(1, 1, 0)
(1, 1, 1)
(1, 1, 2)
(1, 1, 3)
(1, 1, 4)
(1, 2, 0)
(1, 2, 1)
(1, 2, 2)
(1, 2, 3)
(1, 2, 4)
(2, 0, 0)
(2, 0, 1)
(2, 0, 2)
(2, 0, 3)
(2, 0, 4)
(2, 1, 0)
(2, 1, 1)
(2, 1, 2)
(2, 1, 3)
(2, 1, 4)
(2, 2, 0)
(2, 2, 1)
(2, 2, 2)
(2, 2, 3)
(2, 2, 4)
(3, 0, 0)
(3, 0, 1)
(3, 0, 2)
(3, 0, 3)
(3, 0, 4)
(3, 1, 0)
(3, 1, 1)
(3, 1, 2)
(3, 1, 3)
(3, 1, 4)
(3, 2, 0)
(3, 2, 1)
(3, 2, 2)
(3, 2, 3)
(3, 2, 4)
(4, 0, 0)
(4, 0, 1)
(4, 0, 2)
(4, 0, 3)
(4, 0, 4)
(4, 1, 0)
(4, 1, 1)
(4, 1, 2)
(4, 1, 3)
(4, 1, 4)
(4, 2, 0)
(4, 2, 1)
(4, 2, 2)
(4, 2, 3)
(4, 2, 4)
(5, 0, 0)
(5, 0, 1)
(5, 0, 2)
(5, 0, 3)
(5, 0, 4)
(5, 1, 0)
(5, 1, 1)
(5, 1, 2)
(5, 1, 3)
(5, 1, 4)
(5, 2, 0)
(5, 2, 1)
(5, 2, 2)
(5, 2, 3)
(5, 2, 4)
(6, 0, 0)
(6, 0, 1)
(6, 0, 2)
(6, 0, 3)
(6, 0, 4)
(6, 1, 0)
(6, 1, 1)
(6, 1, 2)
(6, 1, 3)
(6, 1, 4)
(6, 2, 0)
(6, 2, 1)
(6, 2, 2)
(6, 2, 3)
(6, 2, 4)
(7, 0, 0)
(7, 0, 1)
(7, 0, 2)
(7, 0, 3)
(7, 0, 4)
(7, 1, 0)
(7, 1, 1)
(7, 1, 2)
(7, 1, 3)
(7, 1, 4)
(7, 2, 0)
(7, 2, 1)
(7, 2, 2)
(7, 2, 3)
(7, 2, 4)
```
In [9]:
```
p_values
```
Out[9]: range(0, 8)

In [129]:
```
f_birth.shape
```
Out[129]: (365, 1)

In [6]:
```
import warnings
warnings.filterwarnings('ignore')
```
In [ ]:
```
### so we are just trying to provide pairs of (p,d,q) & whichever pair will give me least error i will
 consider that pair,
for p in p_values:
    for d in d_values:
        for q in q_values:
            order=(p,d,q)
            train=f_birth[0:300]
            test=f_birth[300:]
            ### now we have train & test data that we can pass to our algorithm
            predictions=[]
            for i in range(len(test)):
                try:
                    arima=ARIMA(train,order)
                    ##displacement=0
                    model=arima.fit(disp=0)
                    pred_y=model.forecast()[0]
                    predictions.append(pred_y)
                    error=mean_squared_error(test,predictions)
                    print('MSE is {} with order {}'.format(error,order))
                except:
                    continue
```
```
MSE is 56.52293920759617 with order (0, 0, 1)
MSE is 63.848571300147526 with order (0, 0, 2)
MSE is 62.33416783975561 with order (0, 0, 3)
MSE is 64.77092579774711 with order (0, 0, 4)
MSE is 40.91062176216822 with order (0, 1, 1)
MSE is 39.77996582323159 with order (0, 1, 2)
MSE is 39.82369065486819 with order (0, 1, 3)
MSE is 40.18808708659703 with order (0, 1, 4)
MSE is 290.01457940015314 with order (0, 2, 1)
MSE is 41.84436774518044 with order (0, 2, 2)
MSE is 40.5256416517787 with order (0, 2, 3)
MSE is 40.52658806905503 with order (0, 2, 4)
MSE is 62.738509490275845 with order (1, 0, 0)
MSE is 47.07166038789473 with order (1, 0, 1)
MSE is 192.99862301899228 with order (1, 1, 0)
MSE is 39.71362064325994 with order (1, 1, 1)
MSE is 42.798809072682566 with order (1, 1, 2)
MSE is 40.23552880790164 with order (1, 1, 3)
MSE is 39.71375436121507 with order (1, 1, 4)
MSE is 552.2760596545423 with order (1, 2, 0)
MSE is 67.29451447898408 with order (2, 0, 0)
MSE is 46.642984165183115 with order (2, 0, 1)
MSE is 43.85843085865771 with order (2, 0, 2)
MSE is 127.63815488309805 with order (2, 1, 0)
MSE is 40.28862015285816 with order (2, 1, 1)
MSE is 41.537644338362966 with order (2, 1, 2)
```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: