Loading Libraries

In [43]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
sns.set()
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

import warnings
warnings.filterwarnings('ignore')
```

Load Data

In [2]:

```
f_birth = pd.read_csv(r"E:\Data Science Projects\Project Files\Timeseries\Dail
                      sep=",", encoding='cp1252')
```

In [3]:

```
f_birth.head()
```

Out[3]:

	Date	Daily total female births in California, 1959
0	01-01-1959	35
1	02-01-1959	32
2	03-01-1959	30
3	04-01-1959	31
4	05-01-1959	44

In [4]:

```
f_birth.tail()
```

Out[4]:

	Date	Daily total female births in California, 1959
360	27-12-1959	37
361	28-12-1959	52
362	29-12-1959	48
363	30-12-1959	55
364	31-12-1959	50

Data Preparation

In [5]:
```
f_birth.isnull().sum()
```

Out[5]:
```
Date                                              0
Daily total female births in California, 1959    0
dtype: int64
```

In [6]:
```
f_birth.describe()
```

Out[6]:

	Daily total female births in California, 1959
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 [7]:
```
f_birth.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 2 columns):
 #   Column                                         Non-Null Count  Dtype
---  ------                                         --------------  -----
 0   Date                                           365 non-null    object
 1   Daily total female births in California, 1959  365 non-null    int64
dtypes: int64(1), object(1)
memory usage: 5.8+ KB
```

In [8]:
```
f_birth = f_birth.groupby('Date')['Daily total female births in California, 19
```

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

Out[9]:

	Date	Daily total female births in California, 1959
0	01-01-1959	35
1	01-02-1959	23
2	01-03-1959	35
3	01-04-1959	39
4	01-05-1959	32

In [10]:
```
f_birth['Date'] = pd.to_datetime(f_birth['Date'],format='%d-%m-%Y')
```

In [11]:
```
f_birth.rename(columns={'Daily total female births in California, 1959':'femal
```

In [12]:
```
f_birth.head()
```

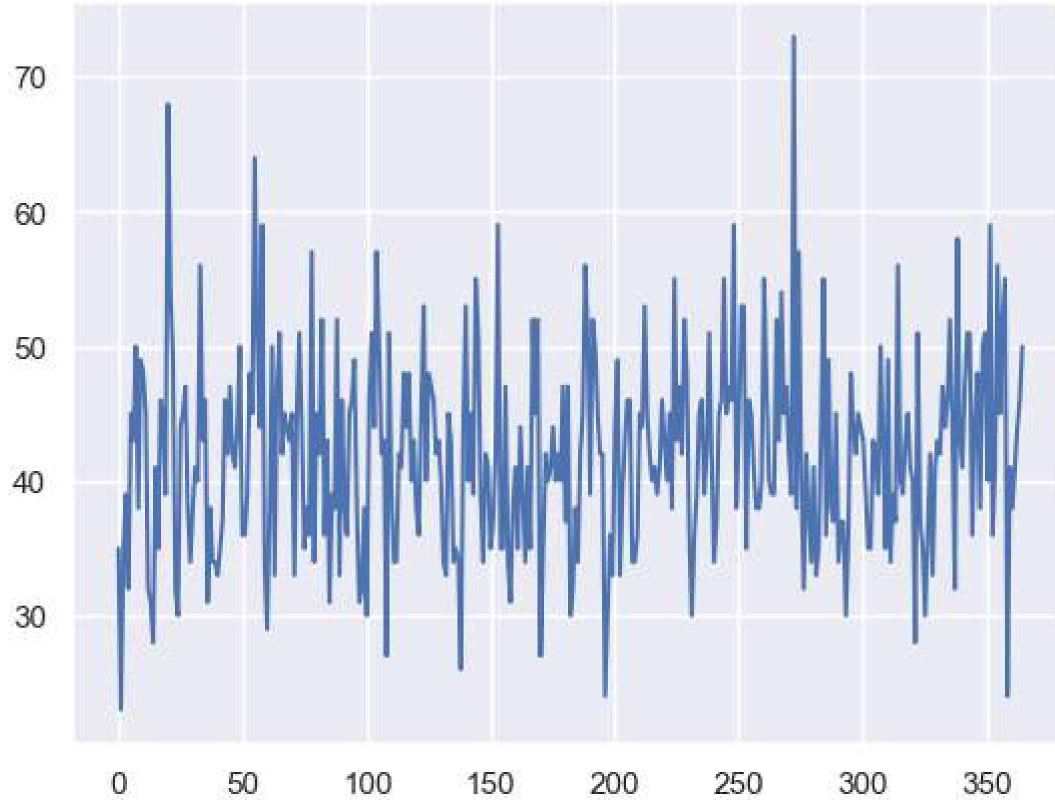
Out[12]:

	Date	female_births
0	1959-01-01	35
1	1959-02-01	23
2	1959-03-01	35
3	1959-04-01	39
4	1959-05-01	32

In [13]:
```
f_birth['female_births']
```

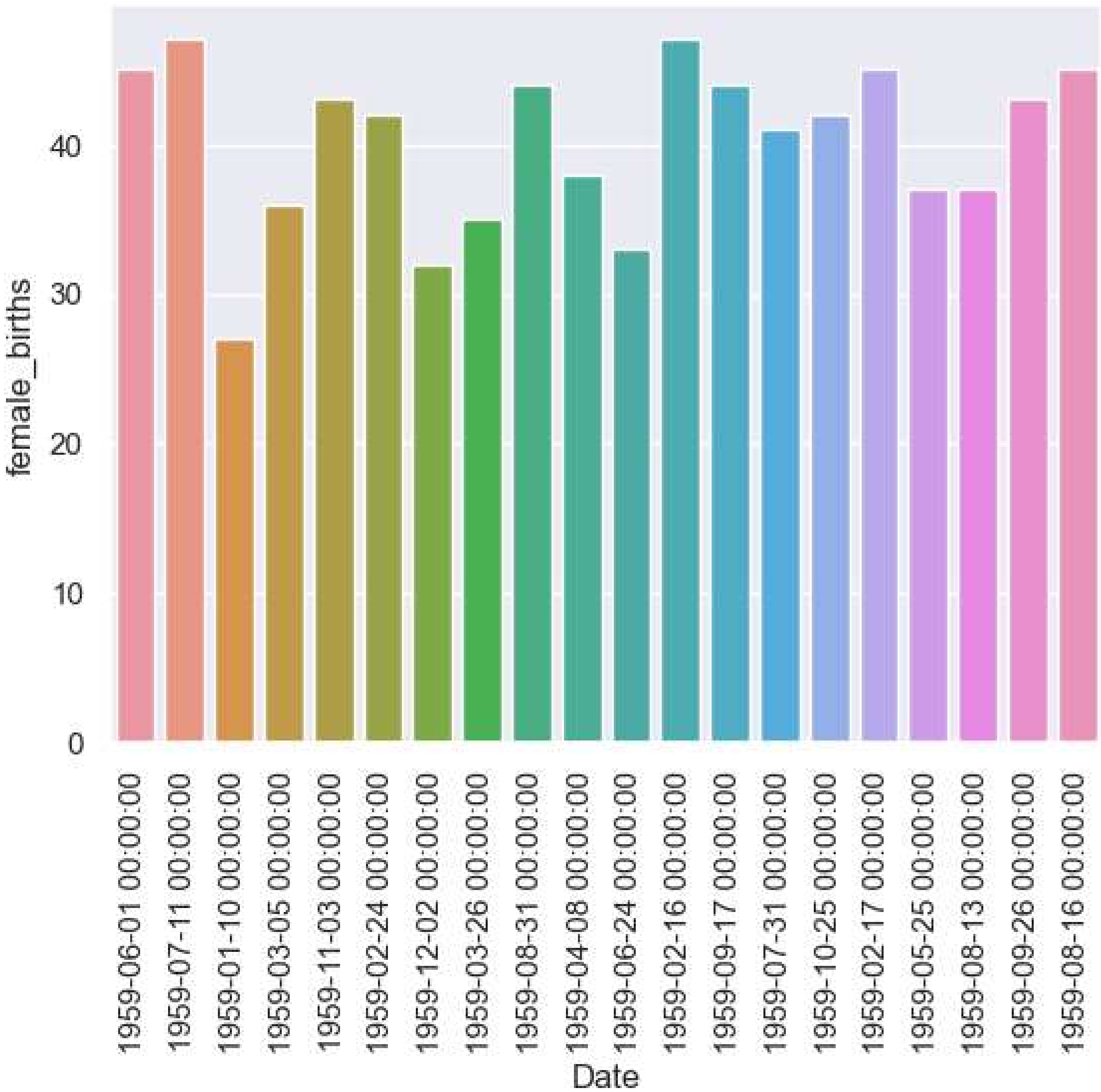
Out[13]:
```
0      35
1      23
2      35
3      39
4      32
       ..
360    38
361    41
362    44
363    46
364    50
Name: female_births, Length: 365, dtype: int64
```

In [14]:
```
f_birth['female_births'].plot()
plt.show()
```

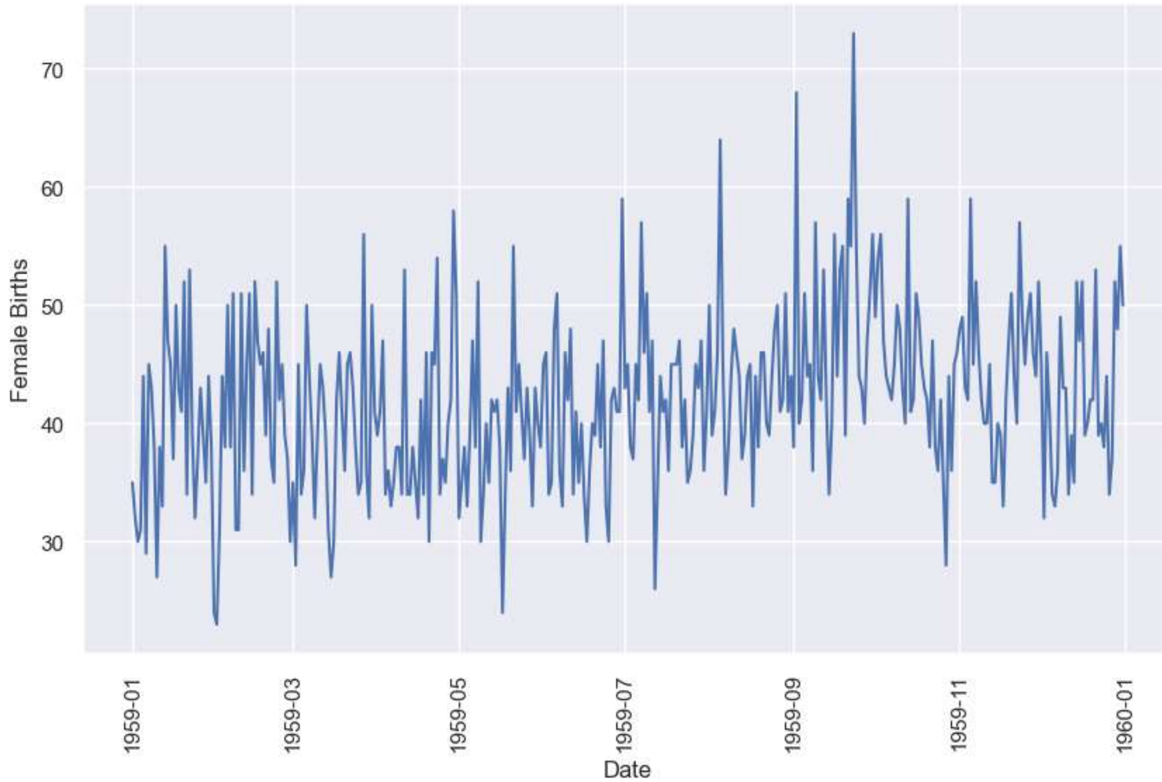


In [15]:
```
a= f_birth.sample(20)
```

In [16]:
```
sns.barplot(x='Date',y='female_births',data=a)
plt.xticks(rotation = 90)
plt.show()
```



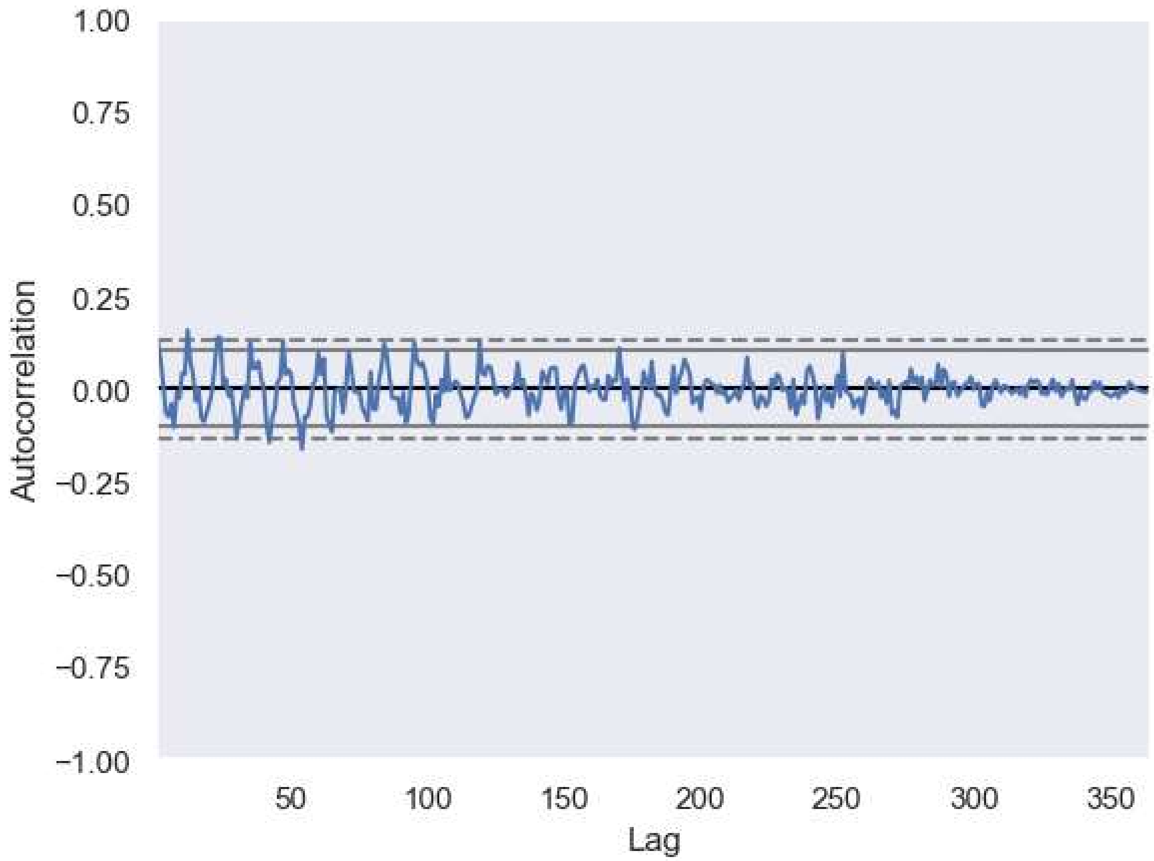
In [17]:
```
plt.figure(figsize=(10, 6))
sns.lineplot(x='Date', y='female_births', data=f_birth)
plt.xlabel('Date')
plt.ylabel('Female Births')
plt.xticks(rotation=90)
plt.show()
```



AutoCorrelation Plots

In [18]:
```
from pandas.plotting import autocorrelation_plot as aplt
```

In [19]:
```
aplt(f_birth['female_births'])
plt.show()
```



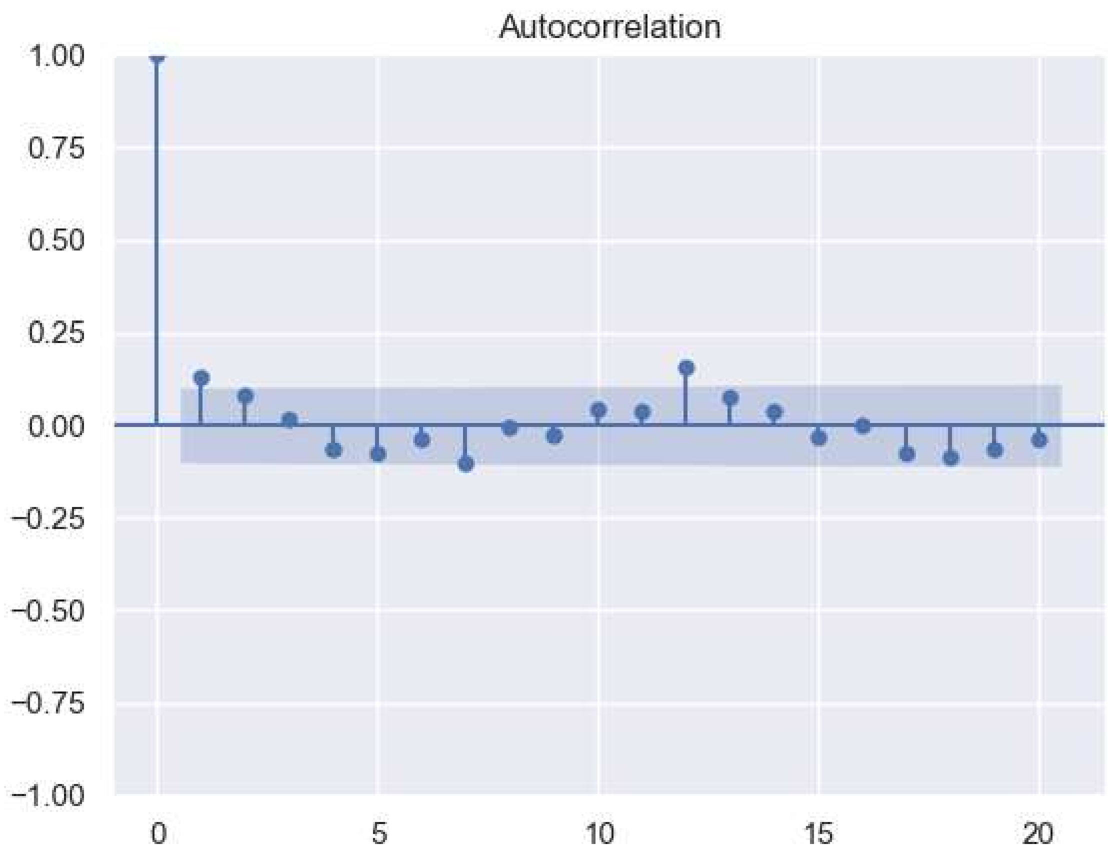
stationarity and plot ACF and PACF

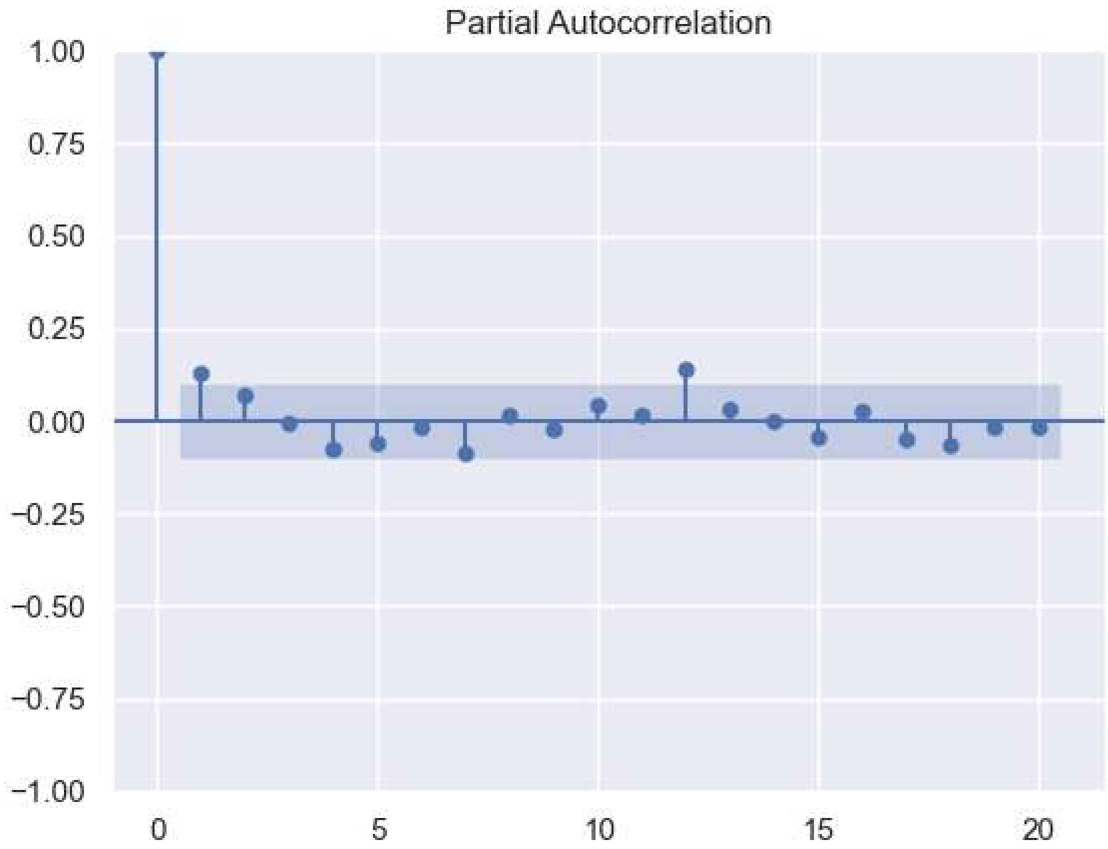
In [27]:

```
# Function to test stationarity and plot ACF and PACF
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.ar_model import AutoReg
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# Dickey-Fuller test
from statsmodels.tsa.stattools import adfuller
def test_stationarity(ts):


    result = adfuller(ts, autolag='AIC')
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    print('Critical Values:')
    for key, value in result[4].items():
        print(f'   {key}: {value}')

    # Plot ACF and PACF
    plot_acf(ts, lags=20)
    plot_pacf(ts, lags=20)
    plt.show()
```

In [28]:
```
# Test stationarity
test_stationarity(f_birth['female_births'])
```

```
ADF Statistic: -16.680635544776198
p-value: 1.5174514677228693e-29
Critical Values:
   1%: -3.4484434475193777
   5%: -2.869513170510808
   10%: -2.571017574266393
```





In [29]:
```
# Split the data into training and testing sets
train_size = int(len(f_birth) * 0.8)
train, test = f_birth['female_births'][:train_size],f_birth['female_births'][t
```

In [30]:
```
train_size
```

Out[30]: 292

In [31]:
```
train
```

Out[31]:
```
0      35
1      23
2      35
3      39
4      32
       ..
287    38
288    37
289    45
290    34
291    37
Name: female_births, Length: 292, dtype: int64
```

In [32]:
```
test
```

Out[32]:
```
292    37
293    30
294    36
295    48
296    44
       ..
360    38
361    41
362    44
363    46
364    50
Name: female_births, Length: 73, dtype: int64
```

In [33]:
```
# AR Model
ar_model = AutoReg(train, lags=1).fit()
ar_pred = ar_model.predict(start=len(train), end=len(train) + len(test) - 1)
```

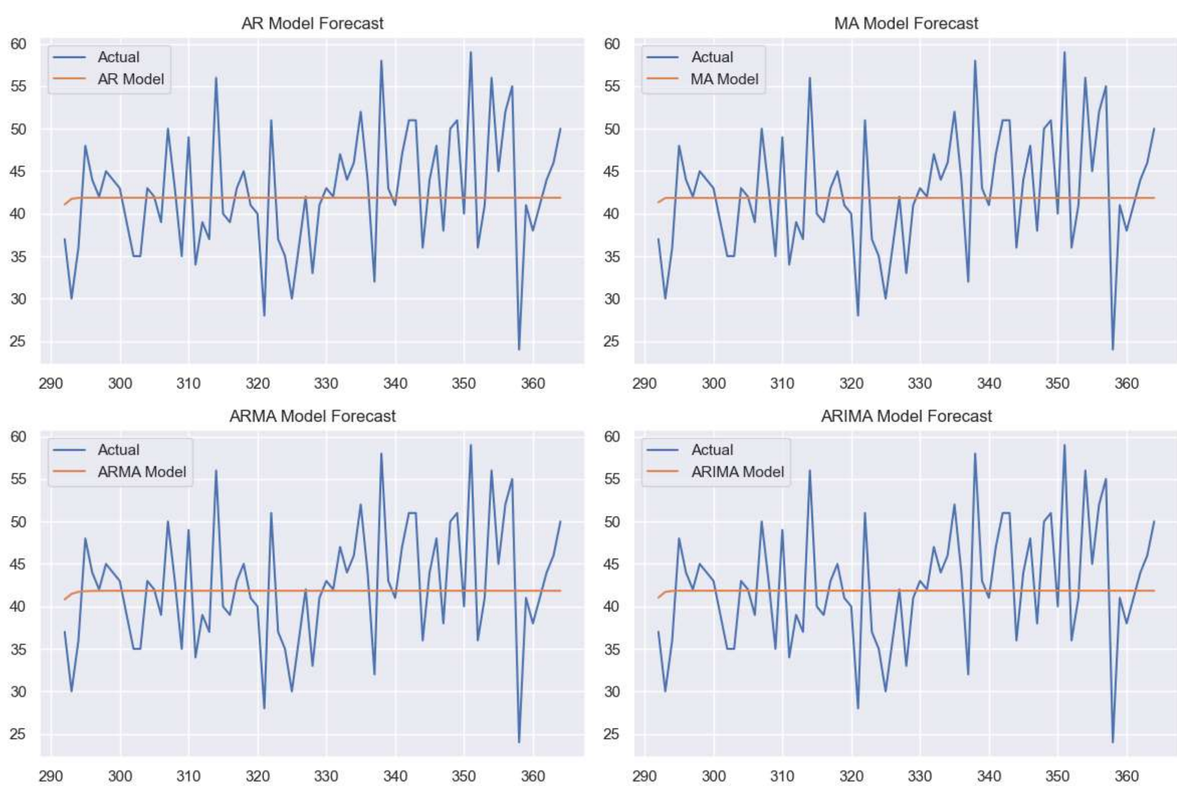
In [34]:
```
# MA Model
ma_model = ARIMA(train, order=(0, 0, 1)).fit()
ma_pred = ma_model.predict(start=len(train), end=len(train) + len(test) - 1)
```

In [35]:
```
# ARMA Model
arma_model = ARIMA(train, order=(1, 0, 1)).fit()
arma_pred = arma_model.predict(start=len(train), end=len(train) + len(test) -
```

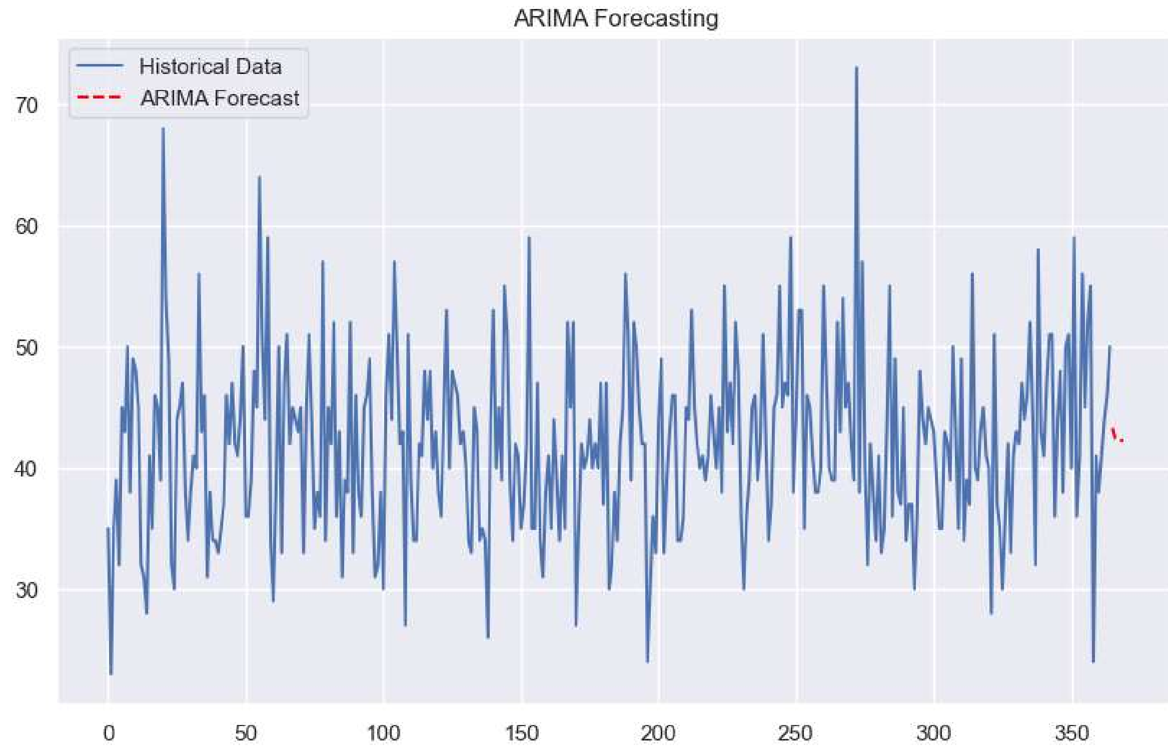
In [36]:
```
# ARIMA Model
arima_model = ARIMA(train, order=(1, 1, 1)).fit()
arima_pred = arima_model.predict(start=len(train), end=len(train) + len(test)
```

In [37]:
```
# Forecasting with ARIMA
model = ARIMA(f_birth['female_births'], order=(1, 1, 1)).fit()
forecast_steps = 5  # Change this according to your needs
forecast = model.forecast(steps=forecast_steps, typ='levels')
```

In [38]:
```
# Plotting results
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(test, label='Actual')
plt.plot(ar_pred, label='AR Model')
plt.title('AR Model Forecast')
plt.legend()

plt.subplot(2, 2, 2)
plt.plot(test, label='Actual')
plt.plot(ma_pred, label='MA Model')
plt.title('MA Model Forecast')
plt.legend()

plt.subplot(2, 2, 3)
plt.plot(test, label='Actual')
plt.plot(arma_pred, label='ARMA Model')
plt.title('ARMA Model Forecast')
plt.legend()

plt.subplot(2, 2, 4)
plt.plot(test, label='Actual')
plt.plot(arima_pred, label='ARIMA Model')
plt.title('ARIMA Model Forecast')
plt.legend()

plt.tight_layout()
plt.show()
```



In [39]:
```
# Plot Forecasting with ARIMA
plt.figure(figsize=(10, 6))
plt.plot(f_birth['female_births'], label='Historical Data')
plt.plot(forecast, label='ARIMA Forecast', linestyle='dashed', color='red')
plt.title('ARIMA Forecasting')
plt.legend(loc ="upper left")
plt.show()
```



Seasonal Decomposition

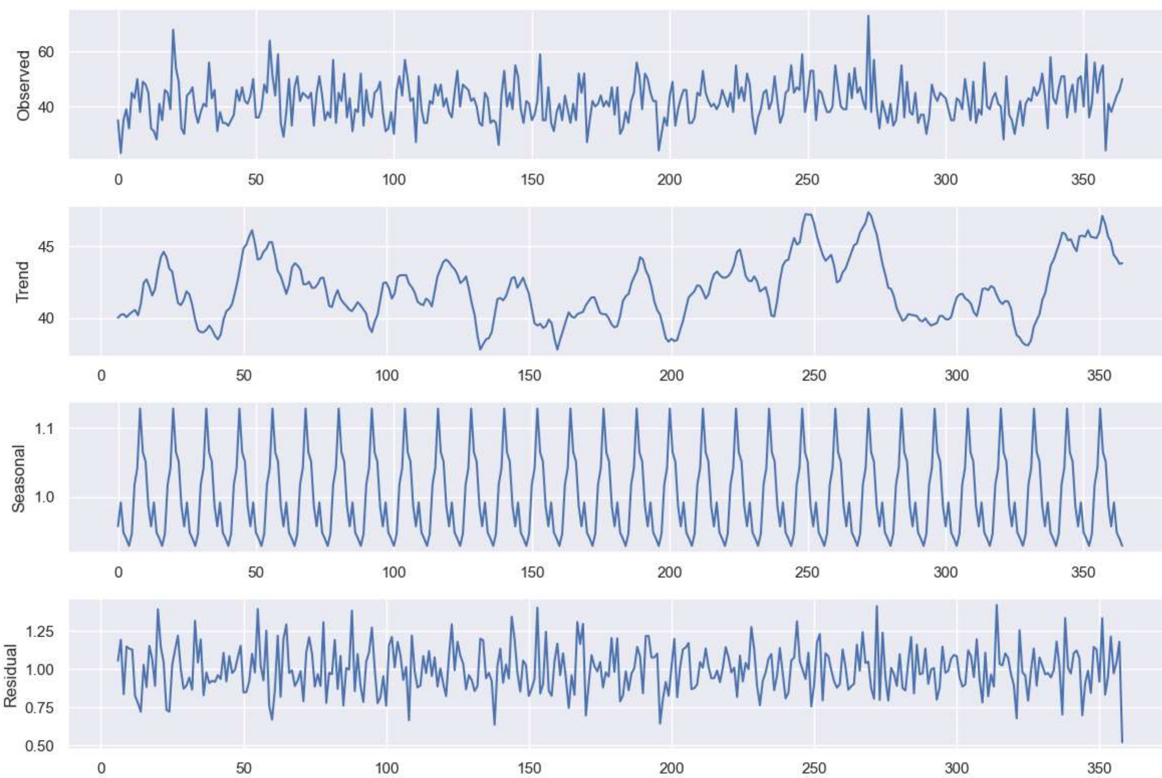
In [40]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Load specific forecasting tools
from statsmodels.tsa.statespace.sarimax import SARIMAX

from statsmodels.tsa.seasonal import seasonal_decompose, DecomposeResult
from pmdarima import auto_arima                              # for determining

# Load specific evaluation tools
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
```

In [41]:
```
def plot_seasonal_decompose(result:DecomposeResult, dates:pd.Series=None, titl
    x_values = dates if dates is not None else np.arange(len(result.observed))
    cols = px.colors.qualitative.Pastel
    return (
        make_subplots(
            rows=4,
            cols=1,
            subplot_titles=["Observed", "Trend", "Seasonal", "Residuals"],
        )
        .add_trace(
            go.Scatter(x=x_values, y=result.observed, mode="lines", name='Obse
            row=1,
            col=1,
        )
        .add_trace(
            go.Scatter(x=x_values, y=result.trend, mode="lines", name='Trend',
            row=2,
            col=1,
        )
        .add_trace(
            go.Scatter(x=x_values, y=result.seasonal, mode="lines", name='Seas
            row=3,
            col=1,
        )
        .add_trace(
            go.Scatter(x=x_values, y=result.resid, mode="lines", name='Residua
            row=4,
            col=1,
        )
        .update_layout(
            height=900, title=f'<b>{title}</b>', margin={'t':100}, title_x=0.5
        )
    )
```

In [44]:
```
seasonal_df = pd.DataFrame(f_birth.groupby(['Date'])['female_births'].apply("m

decomposition = seasonal_decompose(seasonal_df['female_births'], model='additi
fig = plot_seasonal_decompose(decomposition, dates=seasonal_df.index)

fig.update_layout(plot_bgcolor='white')
fig.update_yaxes(
    mirror=True,
    ticks='outside',
    showline=True,
    linecolor='black',
    gridcolor='lightgrey',
    title=''
)

fig.show()
```

In [45]:
```
results = seasonal_decompose(f_birth["female_births"], model="multiplicative",
```

In [46]:
```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 8))

# Plot the seasonal decomposition components on respective subplots
results.observed.plot(ax=ax1)
ax1.set_ylabel('Observed')

results.trend.plot(ax=ax2)
ax2.set_ylabel('Trend')

results.seasonal.plot(ax=ax3)
ax3.set_ylabel('Seasonal')

results.resid.plot(ax=ax4)
ax4.set_ylabel('Residual')

# Adjust layout for better presentation
fig.tight_layout()

# Display the plot
plt.show()
```



In [47]:
```
#forecasting for Female_births
auto_arima(seasonal_df['female_births'], seasonal=True, m=12).summary()
```

Out[47]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	365
Model:	SARIMAX(1, 1, 1)x(1, 0, [], 12)	**Log Likelihood**	-1224.165
Date:	Fri, 22 Dec 2023	**AIC**	2456.330
Time:	15:16:57	**BIC**	2471.919
Sample:	01-01-1959	**HQIC**	2462.526
	- 12-31-1959		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	0.1043	0.060	1.736	0.083	-0.013	0.222
ma.L1	-0.9544	0.019	-50.429	0.000	-0.991	-0.917
ar.S.L12	-0.1192	0.051	-2.327	0.020	-0.220	-0.019
sigma2	48.4909	3.247	14.935	0.000	42.127	54.855

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	25.08
Prob(Q):	0.88	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.94	**Skew:**	0.58
Prob(H) (two-sided):	0.72	**Kurtosis:**	3.54

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

In [48]:
```
train = seasonal_df.iloc[:len(seasonal_df)-12]
test = seasonal_df.iloc[len(seasonal_df)-12:]
```

In [49]:
```
model = SARIMAX(train['female_births'], order=(2, 1, 2), seasonal_order=(2, 0,
results = model.fit()
results.summary()
```

Out[49]:

SARIMAX Results

Dep. Variable:	female_births	**No. Observations:**	353
Model:	SARIMAX(2, 1, 2)x(2, 0, [1], 12)	**Log Likelihood**	-1184.019
Date:	Fri, 22 Dec 2023	**AIC**	2384.039
Time:	15:17:00	**BIC**	2414.948
Sample:	01-01-1959	**HQIC**	2396.339
	- 12-19-1959		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	0.2802	1.910	0.147	0.883	-3.464	4.024
ar.L2	0.0108	0.210	0.052	0.959	-0.401	0.423
ma.L1	-1.1350	1.912	-0.594	0.553	-4.883	2.613
ma.L2	0.1691	1.828	0.093	0.926	-3.413	3.752
ar.S.L12	0.5794	1.584	0.366	0.714	-2.524	3.683
ar.S.L24	0.0969	0.157	0.618	0.536	-0.210	0.404
ma.S.L12	-0.6950	1.591	-0.437	0.662	-3.813	2.423
sigma2	48.4428	3.342	14.493	0.000	41.892	54.994

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	23.96
Prob(Q):	0.90	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.96	**Skew:**	0.57
Prob(H) (two-sided):	0.84	**Kurtosis:**	3.58

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

In [50]:
```
start = len(train)
end = len(train) + len(test) - 1

predictions = results.predict(start, end, typ='levels').rename('SARIMA Test Pr
```

In [51]:
```
cols = px.colors.qualitative.Pastel
fig = go.Figure()
fig.add_trace(go.Scatter(y=test['female_births'],
                    mode='lines',
                    name='Real Values',
                    marker=dict(color=cols[0])
                        ))
fig.add_trace(go.Scatter(y=predictions,
                    mode='lines',
                    name='SARIMA Test Predictions',
                    marker=dict(color=cols[1])
                        ))
fig.update_layout(plot_bgcolor='white')
fig.update_yaxes(
    mirror=True,
    ticks='outside',
    showline=True,
    linecolor='black',
    gridcolor='lightgrey',
    title=''
)
fig.update_xaxes(
    dtick = 1
)
fig.show()
```

In [52]:
```
error = rmse(test['female_births'], predictions)
print(f'RMSE Error: {error}')
```

```
RMSE Error: 6.874284881909711
```

In [53]:
```
seasonal_df.describe().T[['mean','std']]
```

Out[53]:

	mean	std
female_births	41.980822	7.348257

In [54]:
```
#Retraining the model on the entire dataset and forecast several years ahead
model = SARIMAX(seasonal_df['female_births'], order=(2, 1, 2), seasonal_order=
results = model.fit()
forecast = results.predict(start=len(seasonal_df), end=len(seasonal_df)+22, ty
```

In [55]:
```
cols = px.colors.qualitative.Pastel
fig = go.Figure()
fig.add_trace(go.Scatter(x=seasonal_df.index, y=seasonal_df['female_births'],
                    mode='lines',
                    name='Real Values',
                    marker=dict(color=cols[0])
                        ))
fig.add_trace(go.Scatter(x=forecast.index, y=forecast,
                    mode='lines',
                    name='SARIMA Test Predictions',
                    marker=dict(color=cols[1])
                        ))
fig.update_layout(plot_bgcolor='white')
fig.update_yaxes(
    mirror=True,
    ticks='outside',
    showline=True,
    linecolor='black',
    gridcolor='lightgrey',
    title=''
)

fig.show()
```

Persistance Model

In [57]:
```
f_birth['t']=f_birth['female_births'].shift(1)
```

In [58]:
```
f_birth.head()
```

Out[58]:

	Date	female_births	t
0	1959-01-01	35	NaN
1	1959-02-01	23	35.0
2	1959-03-01	35	23.0
3	1959-04-01	39	35.0
4	1959-05-01	32	39.0

In [59]:
```
train,test= f_birth[1:f_birth.shape[0]-7],f_birth[f_birth.shape[0]-7:]
```

In [60]:
```
train.shape
```

Out[60]:
```
(357, 3)
```

In [61]:
```
test.shape
```

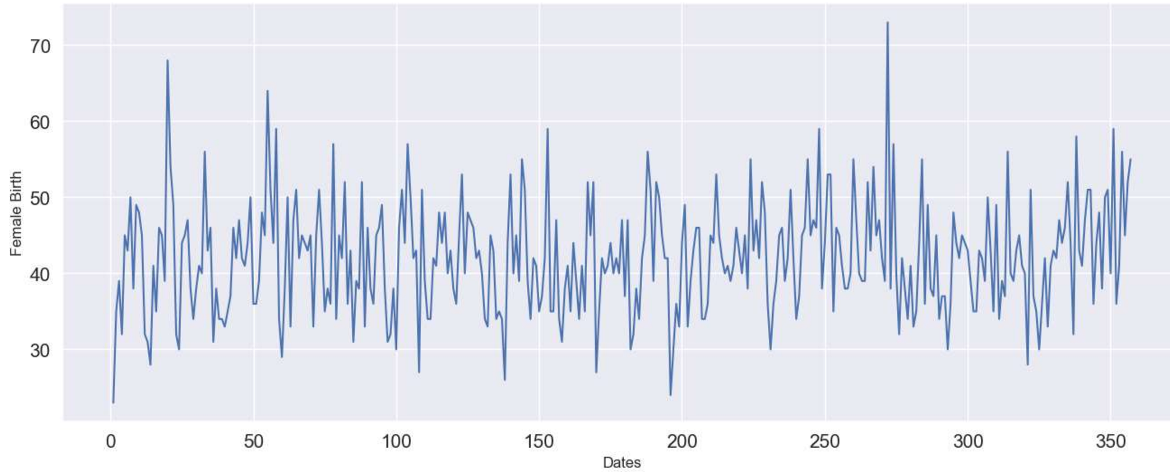
Out[61]:
```
(7, 3)
```

In [62]:
```
train.head()
```

Out[62]:

	Date	female_births	t
1	1959-02-01	23	35.0
2	1959-03-01	35	23.0
3	1959-04-01	39	35.0
4	1959-05-01	32	39.0
5	1959-06-01	45	32.0

In [63]:
```
train['female_births'].plot(figsize=(16,6),fontsize=15)
plt.xlabel("Dates")
plt.ylabel('Female Birth')
plt.show()
```



Walk Forward Validation

In [64]:
```
train_x,train_y = train['t'],train['female_births']
test_x,test_y = test['t'],test['female_births']
```

In [65]:
```
predictions= test_x.copy()
```

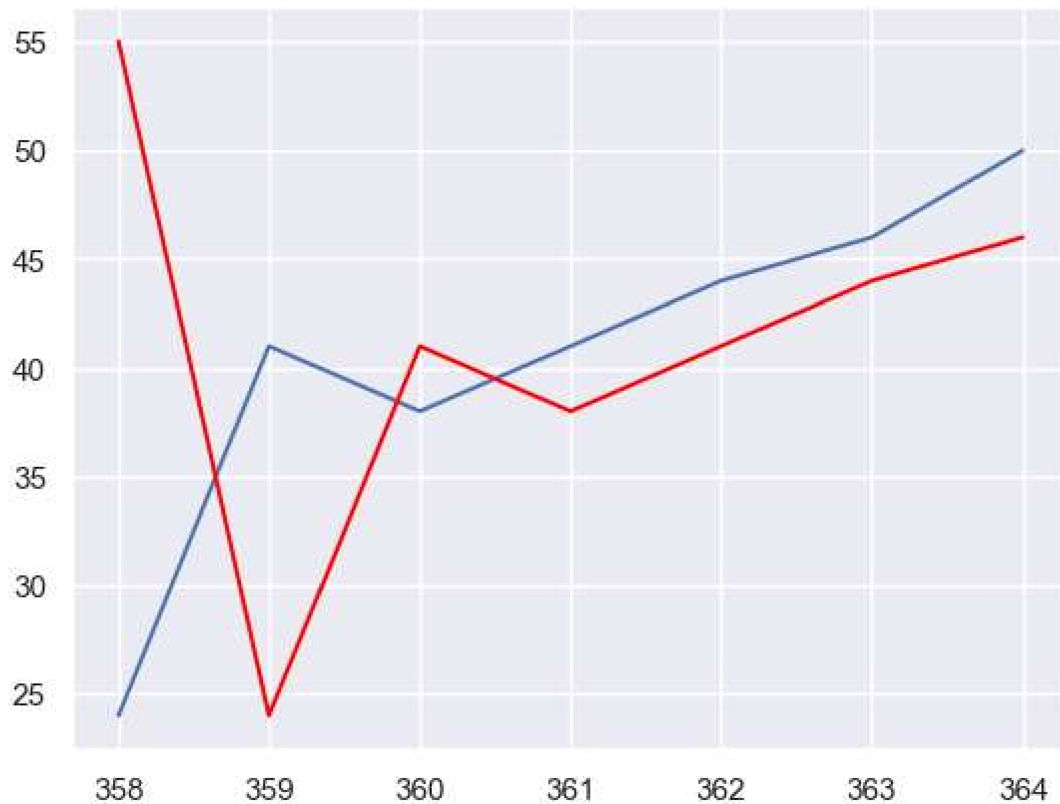
In [66]:
```
print(predictions)
print(test_y)
```

```
358    55.0
359    24.0
360    41.0
361    38.0
362    41.0
363    44.0
364    46.0
Name: t, dtype: float64
358    24
359    41
360    38
361    41
362    44
363    46
364    50
Name: female_births, dtype: int64
```

In [67]:
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(test_y,predictions)
mse
```

Out[67]: 185.28571428571428

In [68]:
```
plt.plot(test_y)
plt.plot(predictions,color='red')
plt.show()
```



Random Forest Time Series Forecasting

In [69]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
```

In [70]:
```
#transfer a timeseries data into a supervised learning dataset
def series_to_supervised(data,n_in=1,n_out=1,dropnan=True):
    n_vars= 1 if type(data) is list else data.shape[1]
    df=pd.DataFrame(data)
    cols=list()
    #input sequence (t-n,...,t-1)
    for i in range(n_in,0,-1):
        cols.append(df.shift(i))
    #forecast sequence(t,t+1,...,t+n)

    for i in range(0,n_out):
        cols.append(df.shift(-i))
    #put it all together
    agg= pd.concat(cols,axis=1)
    #Drop rows with NAN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg.values
```

In [71]:
```
#Split univariate dataset into train/test sets -
def train_test_split(data, n_test):
    return data[:-n_test,:],data[-n_test:,:]
```

In [72]:
```
#fit a Random Forest model and make a one step prediction
def random_forest_forecast(train, testX):
    #transform list into array
    train = np.asarray(train)
    #split into input and output columns
    trainX,trainY = train[:,:-1],train[:,-1]
    #fit model
    model = RandomForestRegressor(n_estimators=1000)
    model.fit(trainX,trainY)
    #make one step prediction
    y_pred = model.predict([testX])
    return y_pred[0]
```

In [73]:

```
#Walk Forward validation for univariate data
def walk_forward_validation(data,n_test):
    predictions  = list()
    #split dataset
    train, test = train_test_split(data,n_test)
    #seed history with training dataset
    history = [x for x in train]
    #step over each time step in the test set
    for i in range(len(test)):
        #split test row into input output columns
        testX, testY = test[i,:-1],test[i,-1]
        #fit model on history and make a prediction
        y_pred = random_forest_forecast(history, testX)
        #store forecatse in list of predictions
        predictions.append(y_pred)
        #add actual observation to history for the next loop
        history.append(test[i])
        #summarize progress
        print('>expected=%.1f, predicted = %.1f'%(testY,y_pred))
    #estimate prediction error
    error = mean_absolute_error(test[:,-1],predictions)
    return error, test[:,-1],predictions
```

In [74]:

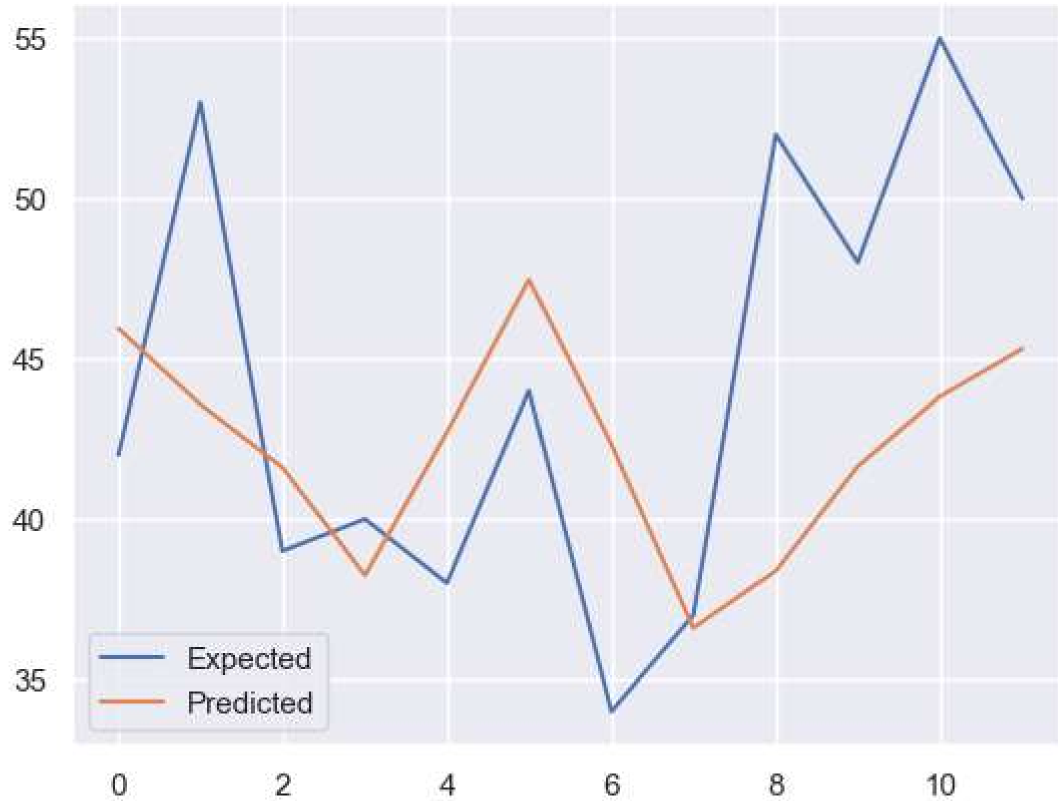
```
#Load the dataset
series = pd.read_csv(r"E:\Data Science Projects\Project Files\Timeseries\Daily
                      header=0,index_col=0)
values = series.values
#transform the timeseries data set into supervised learning dataset
data= series_to_supervised(values,n_in=6)
```

In [75]:

```
#evaluate
mae, y , y_pred = walk_forward_validation(data,12)
print("MAE:%3f"%mae)
```

```
>expected=42.0, predicted = 45.9
>expected=53.0, predicted = 43.6
>expected=39.0, predicted = 41.6
>expected=40.0, predicted = 38.2
>expected=38.0, predicted = 42.7
>expected=44.0, predicted = 47.5
>expected=34.0, predicted = 42.3
>expected=37.0, predicted = 36.6
>expected=52.0, predicted = 38.4
>expected=48.0, predicted = 41.6
>expected=55.0, predicted = 43.8
>expected=50.0, predicted = 45.3
MAE:5.873583
```

In [76]:
```
#plot expected vs predicted
plt.plot(y,label='Expected')
plt.plot(y_pred,label = 'Predicted')
plt.legend()
plt.show()
```



In [ ]: