In [1]:
```
import datetime
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [2]:
```
climate=pd.read_csv("C:\\Users\\Jordy\\Desktop\\DailyDelhiClimateTrain.csv")
```

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

Out[3]:

	date	meantemp	humidity	wind_speed	meanpressure
0	2013-01-01	10.000000	84.500000	0.000000	1015.666667
1	2013-01-02	7.400000	92.000000	2.980000	1017.800000
2	2013-01-03	7.166667	87.000000	4.633333	1018.666667
3	2013-01-04	8.666667	71.333333	1.233333	1017.166667
4	2013-01-05	6.000000	86.833333	3.700000	1016.500000

Understanding our data/checking for null values

In [5]:
```
climate.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1462 entries, 0 to 1461
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   date          1462 non-null   object 
 1   meantemp      1462 non-null   float64
 2   humidity      1462 non-null   float64
 3   wind_speed    1462 non-null   float64
 4   meanpressure  1462 non-null   float64
dtypes: float64(4), object(1)
memory usage: 57.2+ KB
```

In [6]:
```
climate.isnull().sum()
```

Out[6]:
```
date            0
meantemp        0
humidity        0
wind_speed      0
meanpressure    0
dtype: int64
```

We have 5 different columns in our data. Date is for the date we are observing. Meantemp is the mean temperature on a given day. Humidity is for the observed humidy on a given day.

Wind speed measures the wind speed on given day. Meanpressure is the mean of the air pressure observed on that given day. we will monitor all these features to see how they interact with each other and develop a model to be able to predict the temperature on "x" days after the last day observed.

Formatting our date

In [9]:
```
#Changing the data type for date
climate['date'] = pd.to_datetime(climate['date'])
climate.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1462 entries, 0 to 1461
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   date          1462 non-null   datetime64[ns]
 1   meantemp      1462 non-null   float64       
 2   humidity      1462 non-null   float64       
 3   wind_speed    1462 non-null   float64       
 4   meanpressure  1462 non-null   float64       
dtypes: datetime64[ns](1), float64(4)
memory usage: 57.2 KB
```

In [10]:
```
#Indexing date
climate.set_index('date', inplace=True)
climate.info()
```

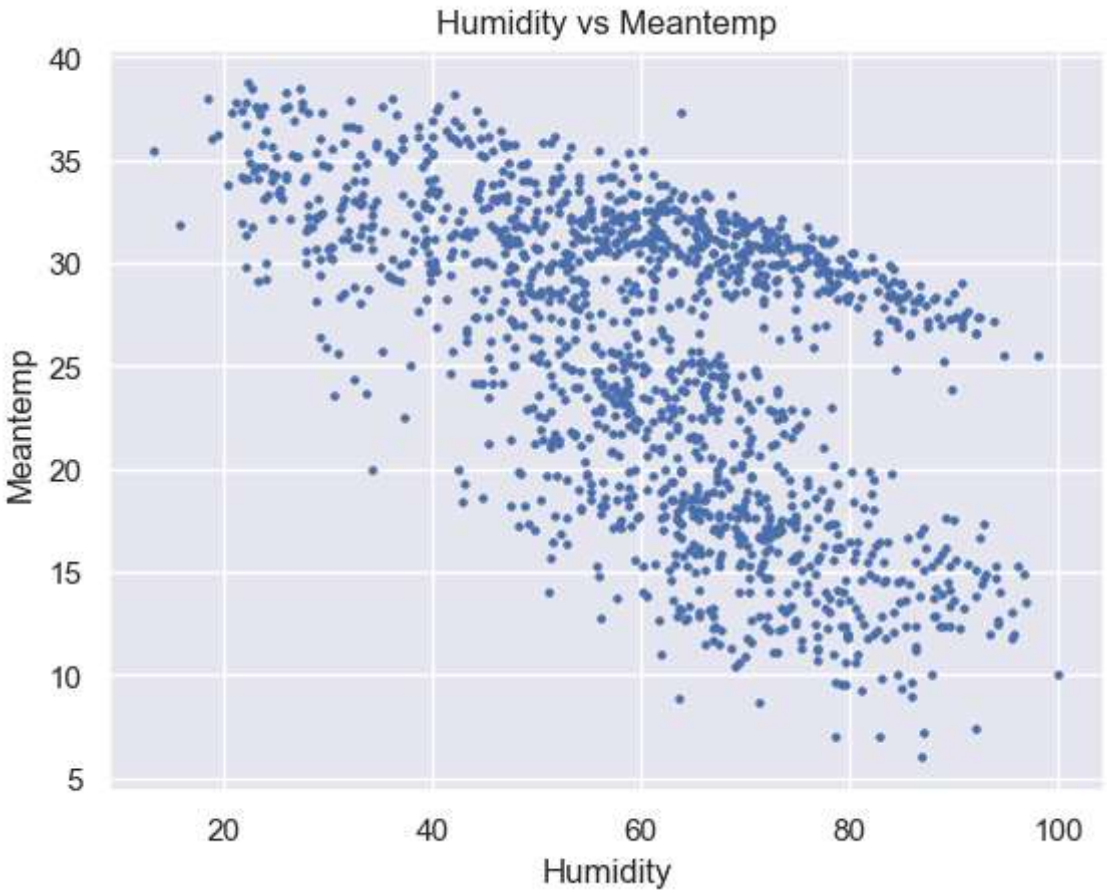
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1462 entries, 2013-01-01 to 2017-01-01
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   meantemp      1462 non-null   float64
 1   humidity      1462 non-null   float64
 2   wind_speed    1462 non-null   float64
 3   meanpressure  1462 non-null   float64
dtypes: float64(4)
memory usage: 57.1 KB
```

EDA of our features

In [13]:
```
climate.plot.scatter(x='humidity',y='meantemp',s=5)
plt.title('Humidity vs Meantemp')
plt.xlabel('Humidity')
plt.ylabel('Meantemp')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
d as value-mapping will have precedence in case its length matches with *x* & *y*.  P
lease use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```

Out[13]:
```
Text(0, 0.5, 'Meantemp')
```

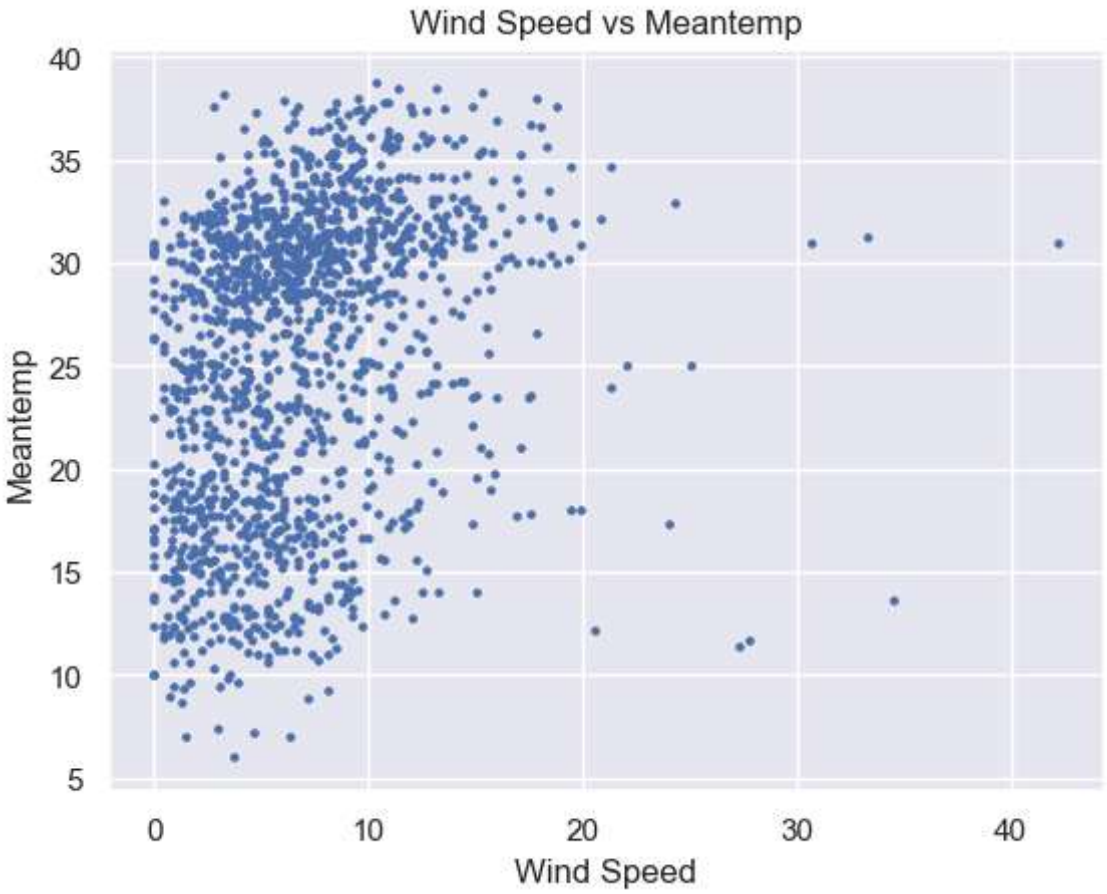


Seems to be a relatively strong negative correlation between humidity and temperature. It looks like the higher the temperature the lower the humidity is.

In [14]:
```
climate.plot.scatter(x='wind_speed',y='meantemp',s=5)
plt.title('Wind Speed vs Meantemp')
plt.xlabel('Wind Speed')
plt.ylabel('Meantemp')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
d as value-mapping will have precedence in case its length matches with *x* & *y*.  P
lease use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```

Out[14]:
```
Text(0, 0.5, 'Meantemp')
```

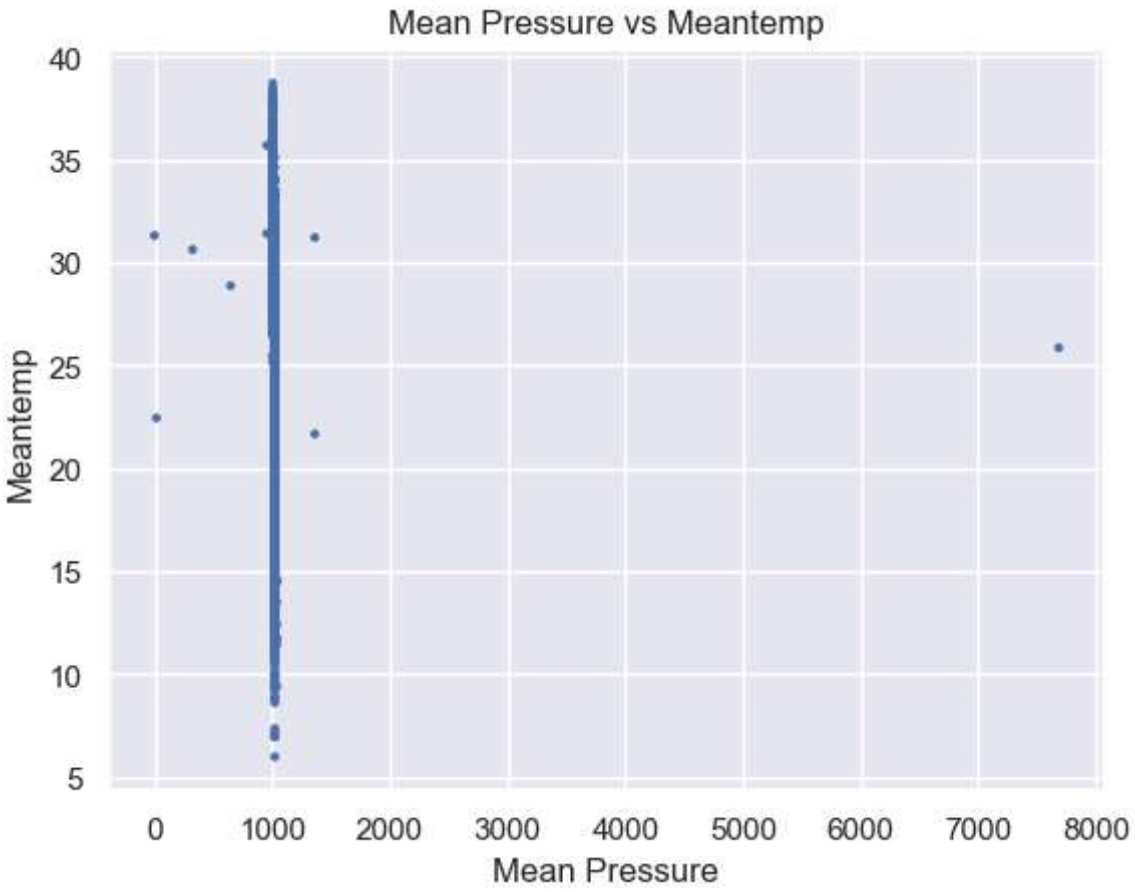


There doesn't seem to be much of a correlation here. Nothing to necessarily note.

In [15]:
```
climate.plot.scatter(x='meanpressure',y='meantemp',s=5)
plt.title('Mean Pressure vs Meantemp')
plt.xlabel('Mean Pressure')
plt.ylabel('Meantemp')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
d as value-mapping will have precedence in case its length matches with *x* & *y*.  P
lease use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```

Out[15]:
```
Text(0, 0.5, 'Meantemp')
```



Again I don't see any glaring correlation to write about. Seems to be that the mean air pressure stays around 1000.

In [16]:
```
corr=climate.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[16]:

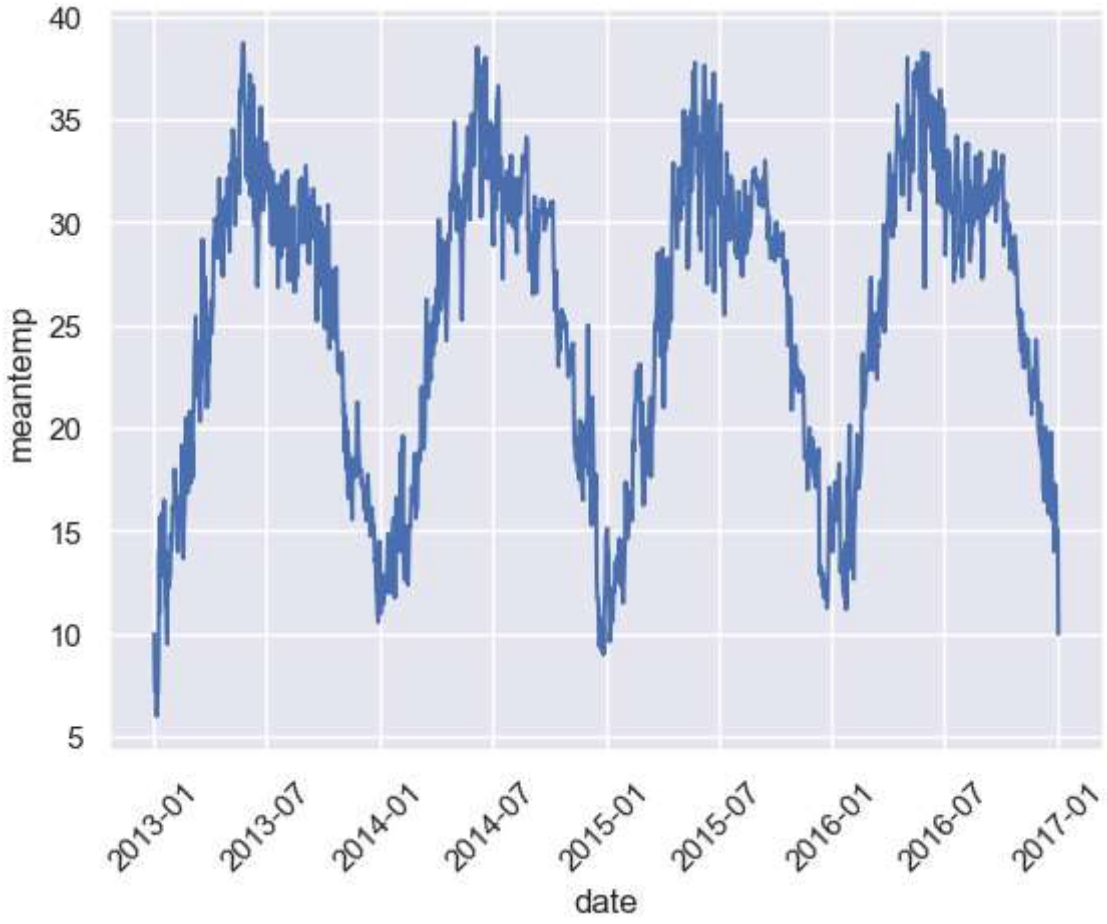
	meantemp	humidity	wind_speed	meanpressure
meantemp	1.000000	-0.571951	0.306468	-0.038818
humidity	-0.571951	1.000000	-0.373972	0.001734
wind_speed	0.306468	-0.373972	1.000000	-0.020670
meanpressure	-0.038818	0.001734	-0.020670	1.000000

Here we can see what we have already observed in that humidy has a decently strong negative correlation with meantemp. It is also important to note the correlation that wind speed has with mean temp. It is a slightly positive correlation. Although it isn't ver strong, it is important to note.

Plotting our data for visualization

In [12]:
```
sns.set()
plt.ylabel('meantemp')
plt.xlabel('date')
plt.xticks(rotation=45)
plt.plot(climate.index, climate['meantemp'], )
```

Out[12]: [<matplotlib.lines.Line2D at 0x20daf7790d0>]



Here, it is evident that we have some sort of seasonality, we can see that by the regular and almost predictable lows and highs when we visualized our data. In terms of a trend, I don't necessarily see any, or a strong sign of one but we will further explore this when we decompose our data.

Decomposing our data

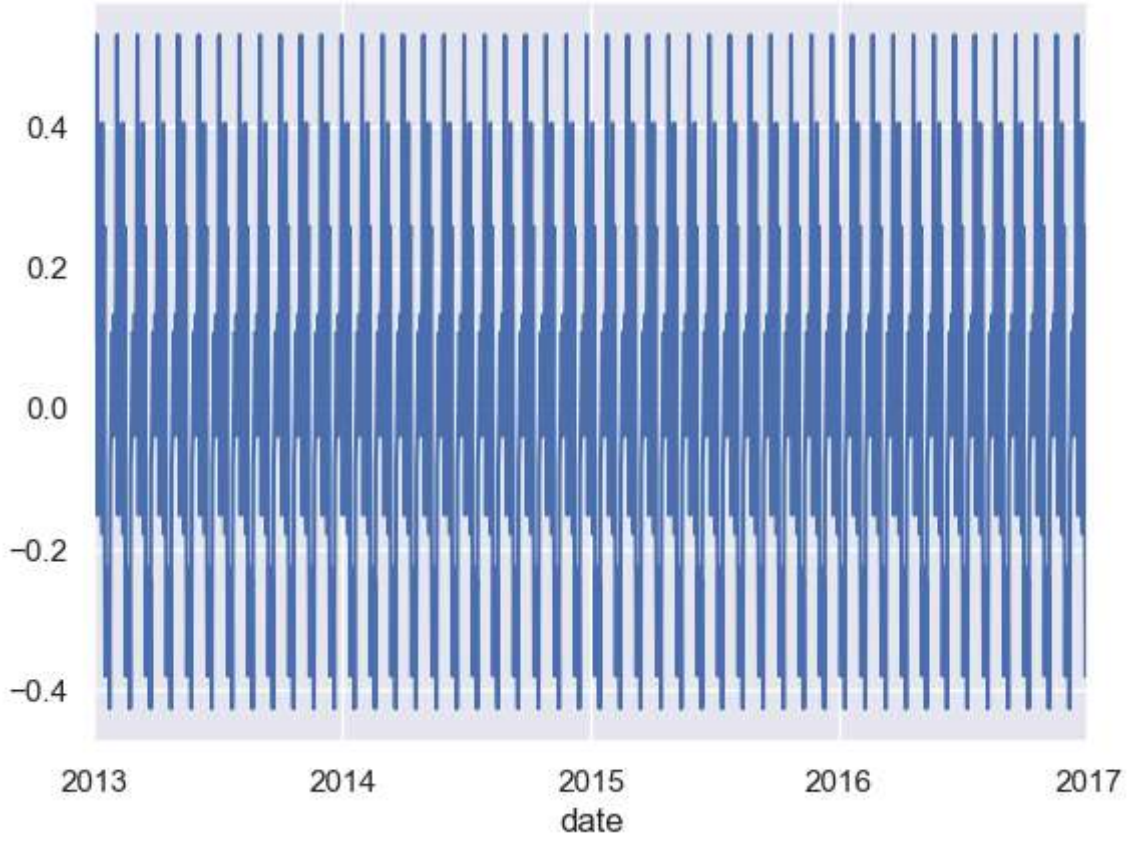
We will decompose our data to further understand the seasonality and trends observed in meantemp over time.

When we decide whether to choose from additive or multplicative model, we will select additive because just going based off our time series chart, we can see that our seasonality does not vary in frequency and magnitude over time. The amount of seasonality in how long, the difference, and the frequency tends to stay the same throughout time.

In [17]:
```
df = climate[['meantemp']]
decompose_data = seasonal_decompose(df, model="additive", period=30)
```

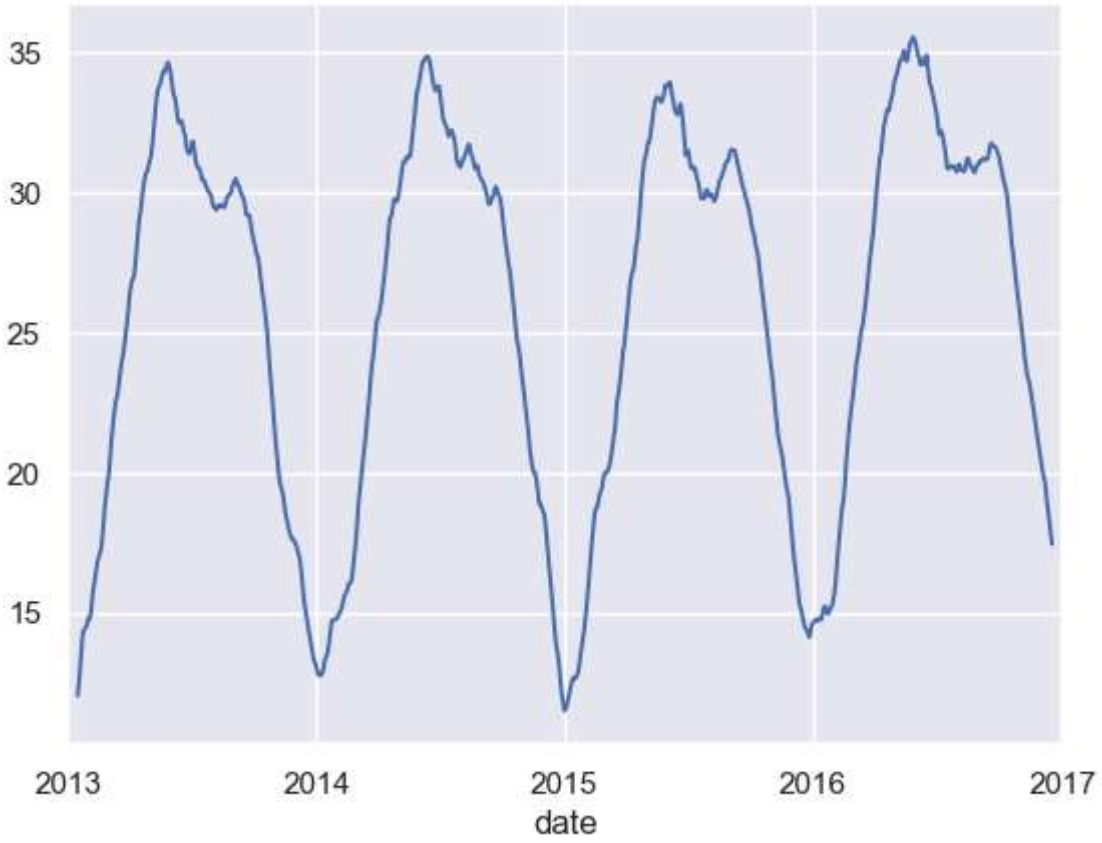
In [18]:
```
decompose_data.seasonal.plot()
```

Out[18]: <AxesSubplot:xlabel='date'>



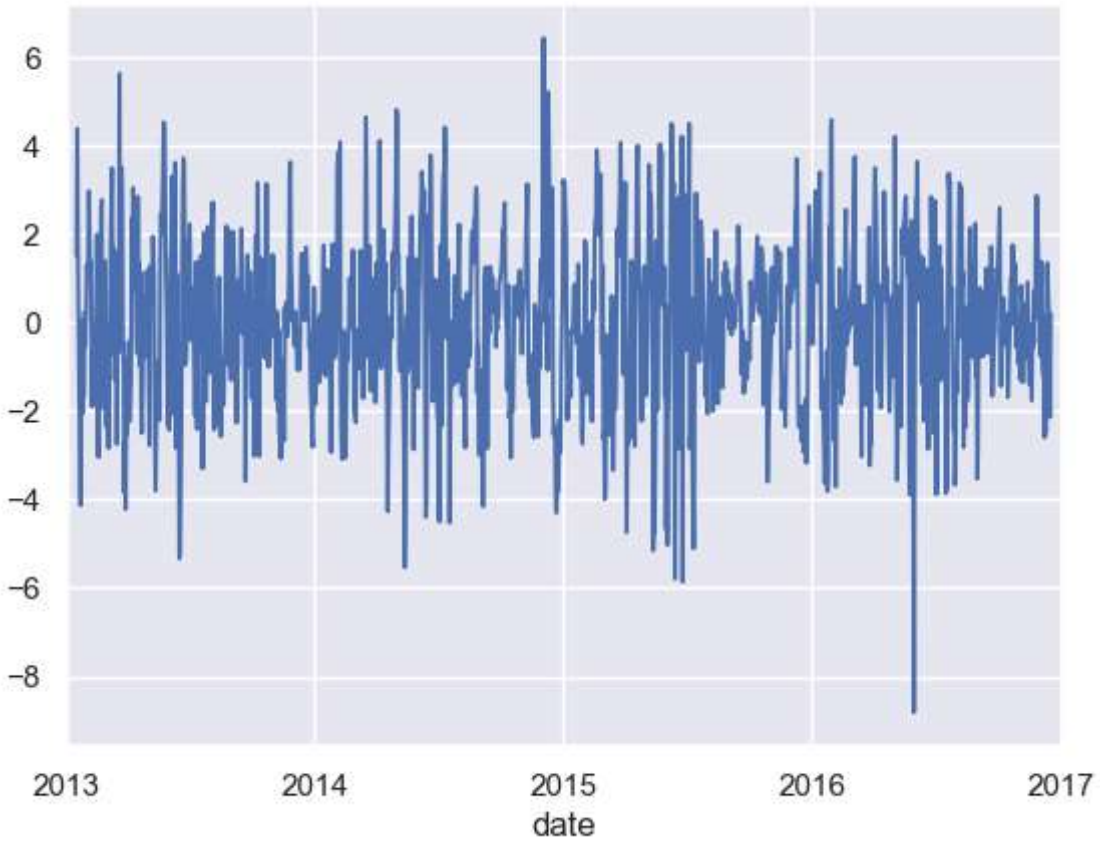
In [19]:
```
decompose_data.trend.plot()
```

Out[19]:
```
<AxesSubplot:xlabel='date'>
```



In [20]:
```
decompose_data.resid.plot()
```

Out[20]:
```
<AxesSubplot:xlabel='date'>
```

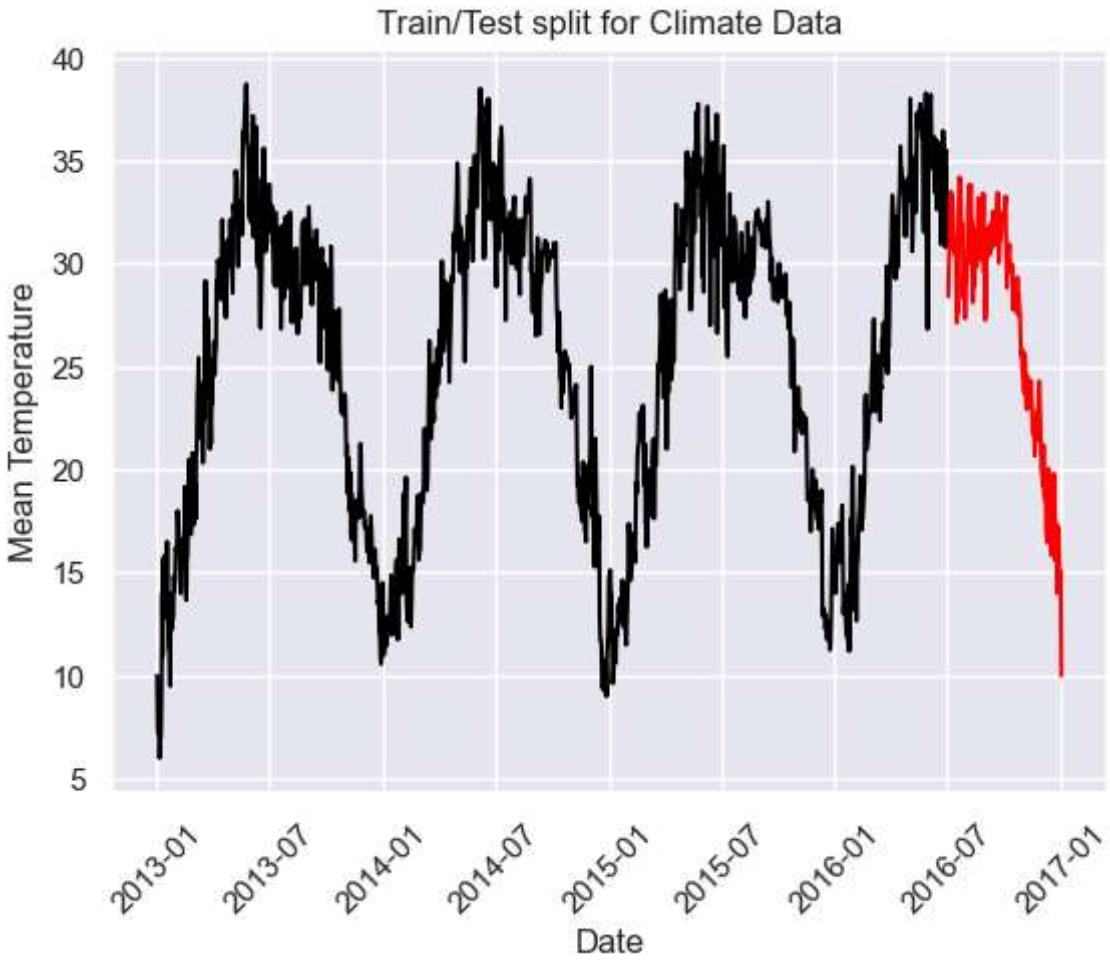


Its important to note here, that our decomposed trend plot looks very similar to our data plotted, ad we an see the obvious and almost predictable dips and rises in temperature every half a year it looks like. Our residual line is decently static but still has a decent amount of noise and we can see our peaks and troves when we look at our seasonal plot.

Splitting data to begin building models

In [21]:
```
train = climate[climate.index < pd.to_datetime("2016-07-01", format='%Y-%m-%d')]
test = climate[climate.index > pd.to_datetime("2016-07-01", format='%Y-%m-%d')]

#plt.plot(train, color = "black")
#plt.plot(test, color = "red")
plt.plot(train.index, train['meantemp'], color = "black")
plt.plot(test.index, test['meantemp'], color = "red")
plt.ylabel('Mean Temperature')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title("Train/Test split for Climate Data")
plt.show()
```



We split the data so that anything before July 2016 is our training data and everything after July 2016 is what we will use to test the model. it is important to differentiate what the test line looks like before we apply a model to it so that we understand how accurarate our model is post test. We will compare it.

Autocorrelation to help with our model building

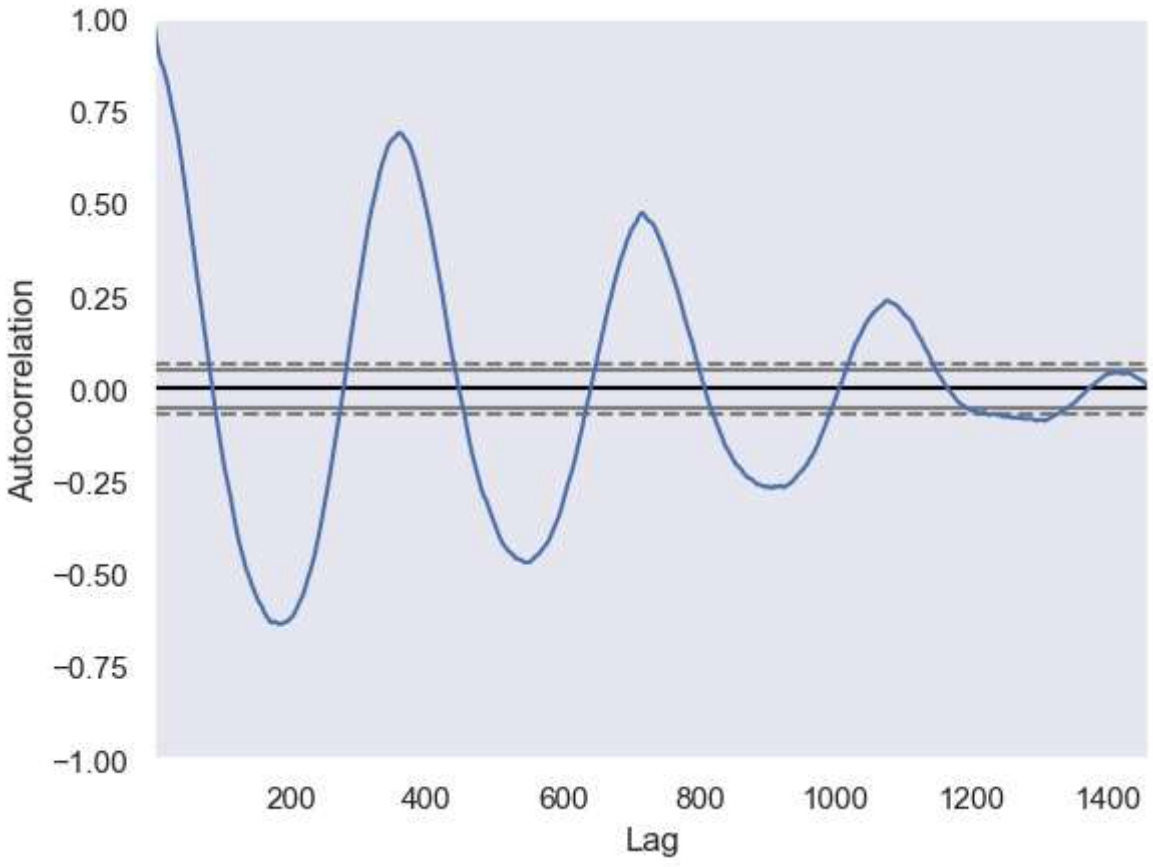
In [27]:
```
climate2=climate[['meantemp']]
print(climate2)
```

```
             meantemp
date
2013-01-01  10.000000
2013-01-02   7.400000
2013-01-03   7.166667
2013-01-04   8.666667
2013-01-05   6.000000
...               ...
2016-12-28  17.217391
2016-12-29  15.238095
2016-12-30  14.095238
2016-12-31  15.052632
2017-01-01  10.000000

[1462 rows x 1 columns]
```

In [30]:
```
from pandas.plotting import autocorrelation_plot
from matplotlib import pyplot
```

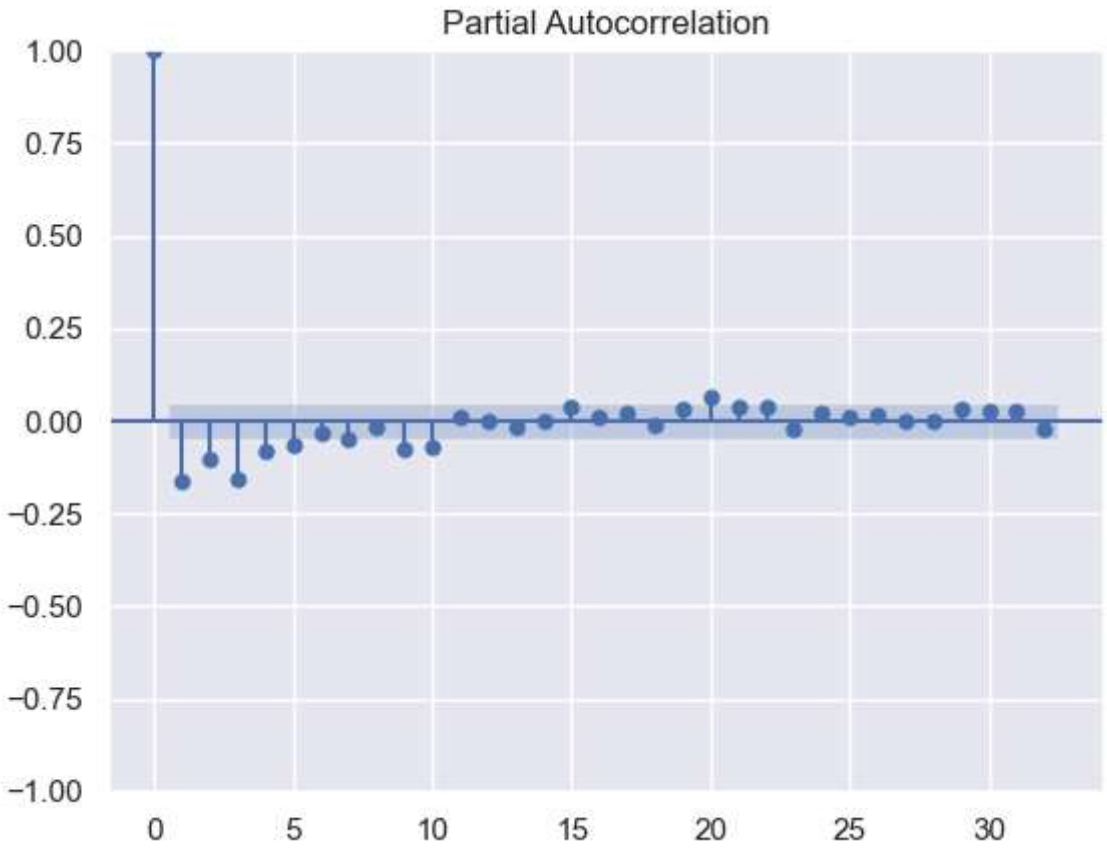
In [31]:
```
autocorrelation_plot(climate2)
pyplot.show()
```

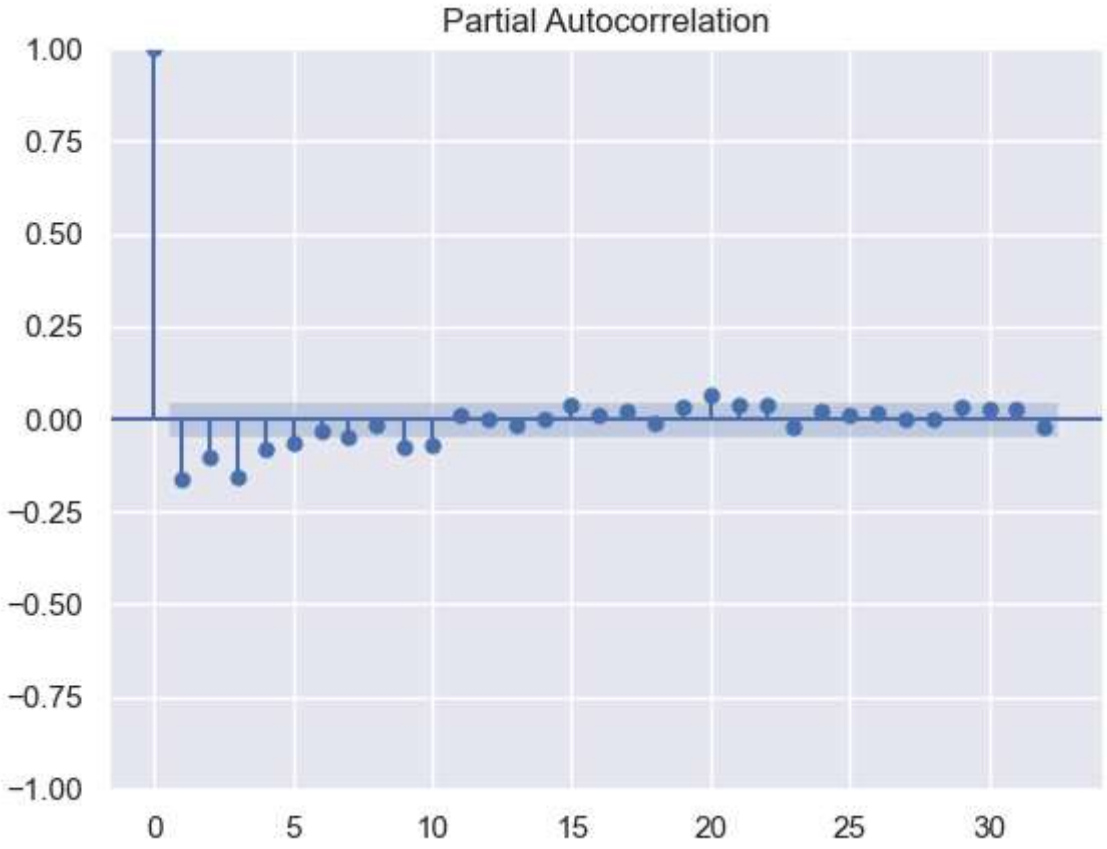


In [32]:
```
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(climate.meantemp.diff().dropna())
```

```
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348: Futu
reWarning: The default method 'yw' can produce PACF values outside of the [-1,1] inte
rval. After 0.13, the default will change tounadjusted Yule-Walker ('ywm'). You can u
se this method now by setting method='ywm'.
  warnings.warn(
```

Out[32]:

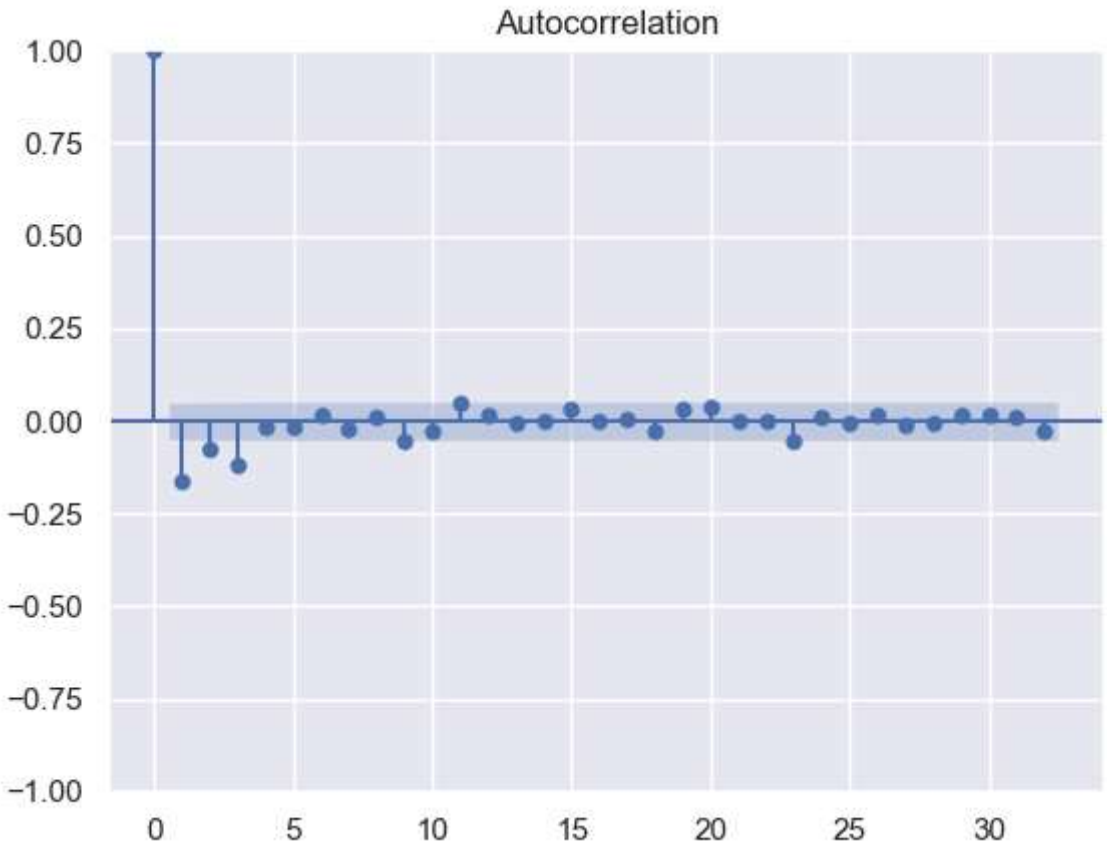


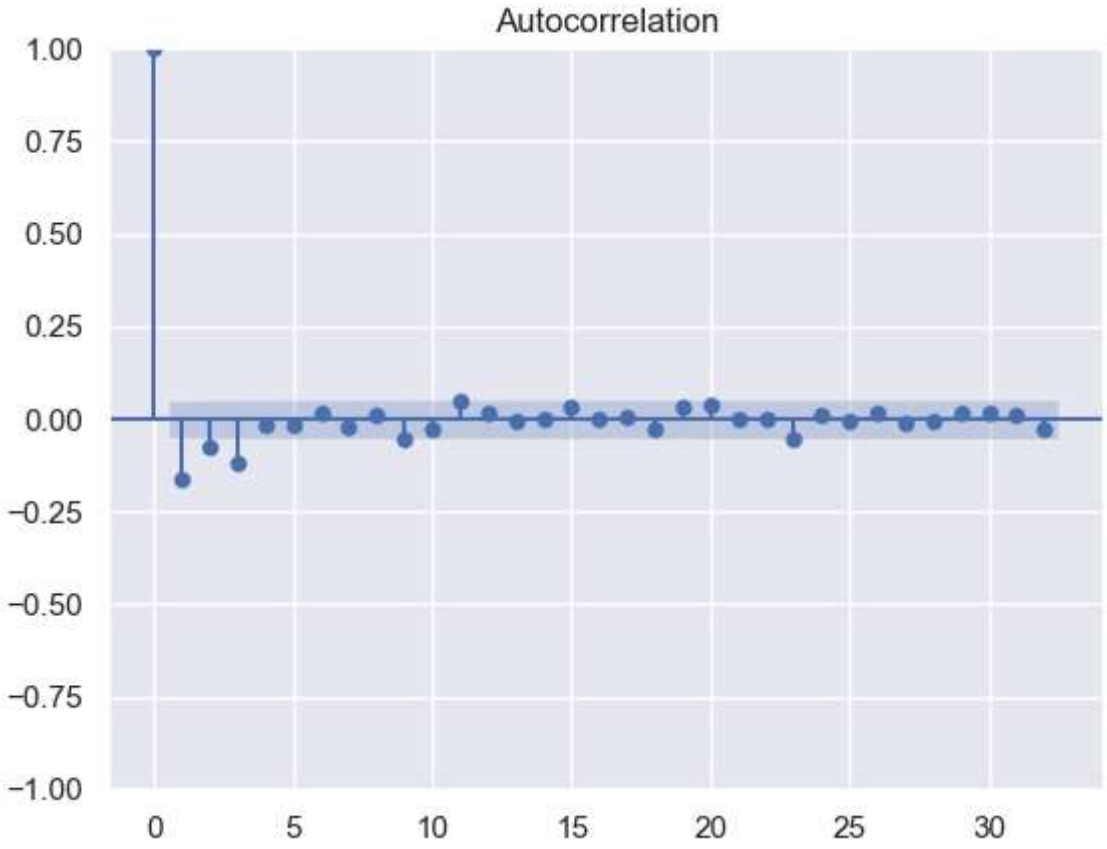


Here, I want to make it understood that I am using this partial autocorrelation to identify what I want to use as my P value in the Arima model. Here I see that our first lag is way out of the limit, so we will ignore it but I can see that although the second lag is out of the limit as well, it is much closer. With that being said we will test the value of either 1 or 2 for our P.

In [40]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(climate.meantemp.diff().dropna())
```

Out[40]:





Using my autocorrelation, I can only see one major instance of the lags being out of the significance limit, so here I will experiment with my q being 1. I can also try 2 or 3 and see how my results look because I do see 2 more out of the significance limit but not by much.

Just looking at the visualization of our data, I have come to the conclusion that not much differencing is needed with our data, So I will attempt using a value of 1 for our d in pdq.

ARIMA Model

In [41]:
```
y = train['meantemp']

ARIMAmodel = ARIMA(y, order = (1, 1, 1))
# The first parameter corresponds to the lagging (past values),
# the second corresponds to differencing (this is what makes
# non-stationary data stationary), and the last parameter corresponds
# to the white noise (for modeling shock events)
ARIMAmodel = ARIMAmodel.fit()

y_pred_2 = ARIMAmodel.get_forecast(len(test.index))
y_pred_df_2 = y_pred_2.conf_int(alpha = 0.05) #--higher alpha assigns greater weight t
y_pred_df_2["Predictions"] = ARIMAmodel.predict(start = y_pred_df_2.index[0], end = y_
y_pred_df_2.index = test.index
y_pred_out_2 = y_pred_df_2["Predictions"]
```

```
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
```

SARIMA Model

In [42]:
```
# Lets define our model input
y = train['meantemp']

SARIMAXmodel = SARIMAX(y, order = (1, 1, 1), seasonal_order=(1,1,1,13))
# The first 3 parameters for seasonal_order are defined the same as when treating non-
# The fourth indicates the seasonal periods to apply the parameters to
# In other words, how many observations of seasonal spikes/dips per year
SARIMAXmodel = SARIMAXmodel.fit()

y_pred_3 = SARIMAXmodel.get_forecast(len(test.index))
y_pred_df_3 = y_pred_3.conf_int(alpha = 0.05)
y_pred_df_3["Predictions"] = SARIMAXmodel.predict(start = y_pred_df_3.index[0], end =
y_pred_df_3.index = test.index
y_pred_out_3 = y_pred_df_3["Predictions"]
```

```
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978:
UserWarning: Non-invertible starting MA parameters found. Using zeros as starting par
ameters.
  warn('Non-invertible starting MA parameters found.'
```

Performance

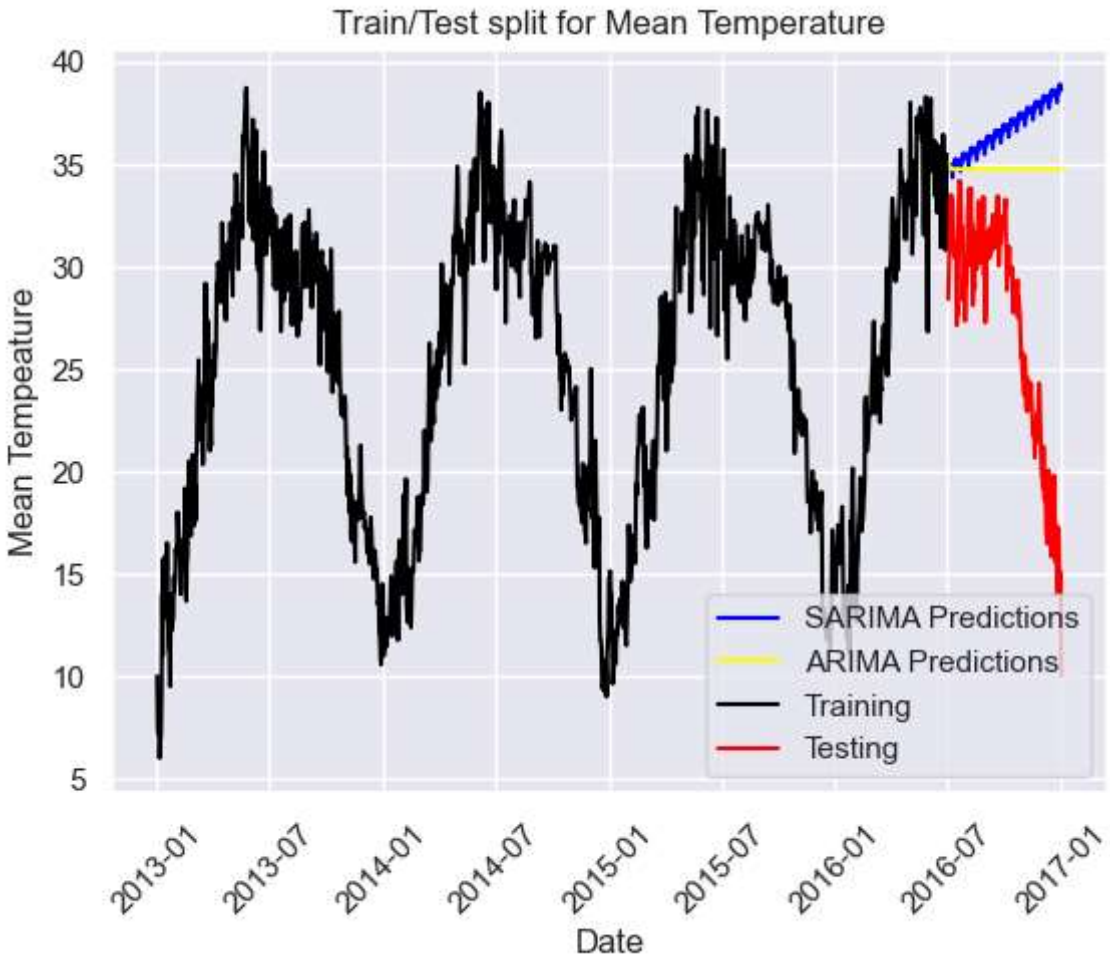
In [43]:

```
plt.plot(y_pred_out_3, color='Blue', label = 'SARIMA Predictions')
plt.plot(y_pred_out_2, color='Yellow', label = 'ARIMA Predictions')
plt.plot(train.index, train['meantemp'], color = "black", label = 'Training')
plt.plot(test.index, test['meantemp'], color = "red", label = 'Testing')
plt.ylabel('Mean Tempeature')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title("Train/Test split for Mean Temperature")
plt.legend()

sarima_rmse = np.sqrt(mean_squared_error(test["meantemp"].values, y_pred_df_3["Predict
print("SARIMA RMSE: ",sarima_rmse)

arima_rmse = np.sqrt(mean_squared_error(test["meantemp"].values, y_pred_df_2["Predicti
print("ARIMA RMSE: ",arima_rmse)
```

```
SARIMA RMSE:  11.65329822866356
ARIMA RMSE:  9.490129237099728
```



I don't like my results, my RMSE (basically a measure of how far your actual values are from the models predicted values on average) is too high, for both models, I have an RMSE of 9.4 and 11.6, when some of my max values for mean temp is around 38. Also looking at the graph, my model prediction graph does not fit or comes close to fitting my original graph. I will mess around with the parameters and see if I can get better results.

Tweaking ARIMA and SARIMA models

In [53]:
```
y = train['meantemp']

ARIMAmodel = ARIMA(y, order = (5, 1, 3))

ARIMAmodel = ARIMAmodel.fit()

y_pred_2 = ARIMAmodel.get_forecast(len(test.index))
y_pred_df_2 = y_pred_2.conf_int(alpha = 0.05)
y_pred_df_2["Predictions"] = ARIMAmodel.predict(start = y_pred_df_2.index[0], end = y_
y_pred_df_2.index = test.index
y_pred_out_2 = y_pred_df_2["Predictions"]
```

```
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\base\model.py:604: Convergence
Warning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [54]:
```
# Lets define our model input
y = train['meantemp']

SARIMAXmodel = SARIMAX(y, order = (5, 1, 3), seasonal_order=(5,1,3,13))

SARIMAXmodel = SARIMAXmodel.fit()

y_pred_3 = SARIMAXmodel.get_forecast(len(test.index))
y_pred_df_3 = y_pred_3.conf_int(alpha = 0.05)
y_pred_df_3["Predictions"] = SARIMAXmodel.predict(start = y_pred_df_3.index[0], end =
y_pred_df_3.index = test.index
y_pred_out_3 = y_pred_df_3["Predictions"]
```

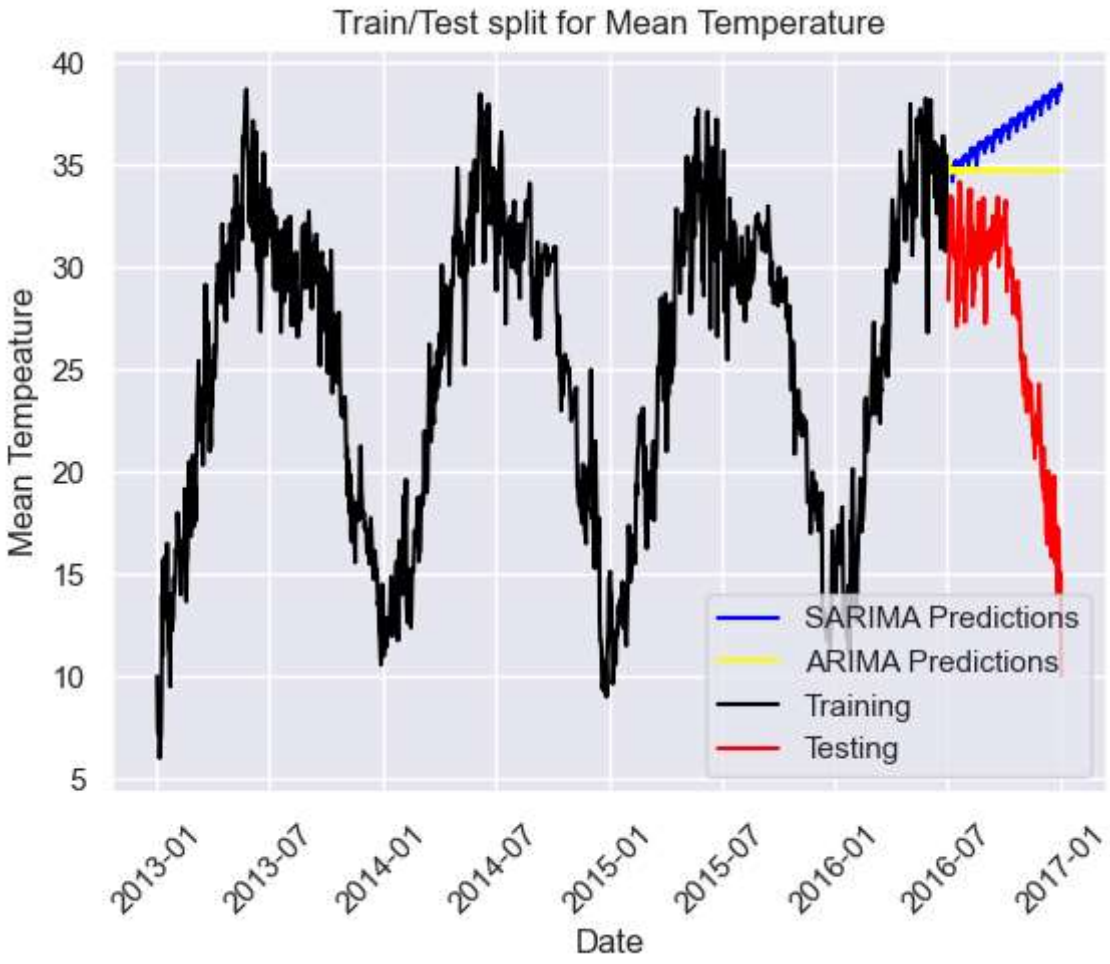
```
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\base\model.py:604: Convergence
Warning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [55]:
```
plt.plot(y_pred_out_3, color='Blue', label = 'SARIMA Predictions')
plt.plot(y_pred_out_2, color='Yellow', label = 'ARIMA Predictions')
plt.plot(train.index, train['meantemp'], color = "black", label = 'Training')
plt.plot(test.index, test['meantemp'], color = "red", label = 'Testing')
plt.ylabel('Mean Tempeature')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title("Train/Test split for Mean Temperature")
plt.legend()

sarima_rmse = np.sqrt(mean_squared_error(test["meantemp"].values, y_pred_df_3["Predict
print("SARIMA RMSE: ",sarima_rmse)

arima_rmse = np.sqrt(mean_squared_error(test["meantemp"].values, y_pred_df_2["Predicti
print("ARIMA RMSE: ",arima_rmse)
```

```
SARIMA RMSE:  11.69474801034673
ARIMA RMSE:  9.505709508435476
```

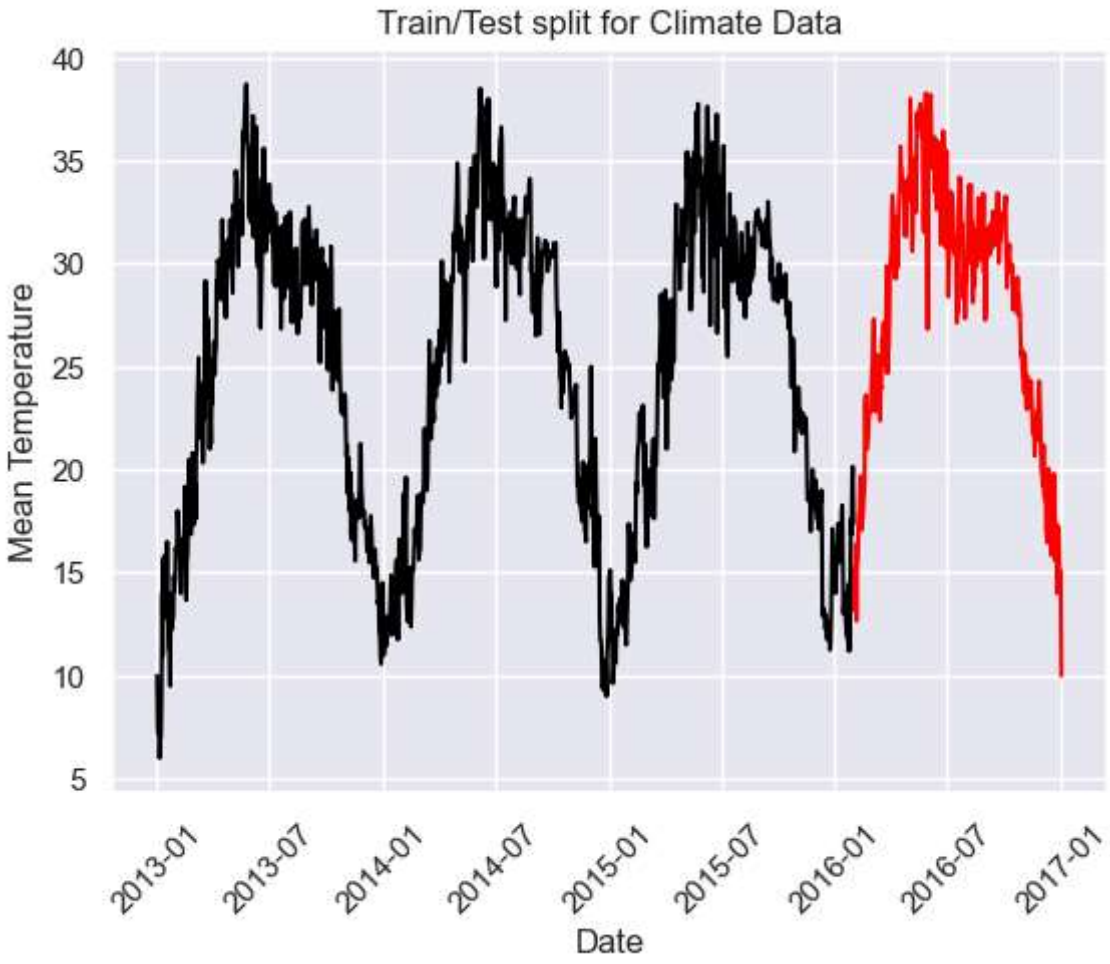


I am getting similar results but it isnt caputring the data and prediciting well. I will either keep tweaking the parameters or try another model to see what works best.

Trying different train/test split

In [72]:
```
train2 = climate[climate.index < pd.to_datetime("2016-02-01", format='%Y-%m-%d')]
test2 = climate[climate.index > pd.to_datetime("2016-02-01", format='%Y-%m-%d')]

#plt.plot(train, color = "black")
#plt.plot(test, color = "red")
plt.plot(train2.index, train2['meantemp'], color = "black")
plt.plot(test2.index, test2['meantemp'], color = "red")
plt.ylabel('Mean Temperature')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title("Train/Test split for Climate Data")
plt.show()
```



Rebuilding ARIMA and SARIMA Model With New Splits

In [73]:
```
y = train2['meantemp']

ARIMAmodel = ARIMA(y, order = (5, 1, 5))

ARIMAmodel = ARIMAmodel.fit()

y_pred_2 = ARIMAmodel.get_forecast(len(test2.index))
y_pred_df_2 = y_pred_2.conf_int(alpha = 0.05)
y_pred_df_2["Predictions"] = ARIMAmodel.predict(start = y_pred_df_2.index[0], end = y_
y_pred_df_2.index = test2.index
y_pred_out_2 = y_pred_df_2["Predictions"]
```

```
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\base\model.py:604: Convergence
Warning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [74]:

```
# Lets define our model input
y = train2['meantemp']

SARIMAXmodel = SARIMAX(y, order = (5, 1, 5), seasonal_order=(5,1,5,13))

SARIMAXmodel = SARIMAXmodel.fit()

y_pred_3 = SARIMAXmodel.get_forecast(len(test2.index))
y_pred_df_3 = y_pred_3.conf_int(alpha = 0.05)
y_pred_df_3["Predictions"] = SARIMAXmodel.predict(start = y_pred_df_3.index[0], end =
y_pred_df_3.index = test2.index
y_pred_out_3 = y_pred_df_3["Predictions"]
```

```
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:966:
UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as
starting parameters.
  warn('Non-stationary starting autoregressive parameters'
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978:
UserWarning: Non-invertible starting MA parameters found. Using zeros as starting par
ameters.
  warn('Non-invertible starting MA parameters found.'
C:\Users\Jordy\anaconda3\lib\site-packages\statsmodels\base\model.py:604: Convergence
Warning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [75]:

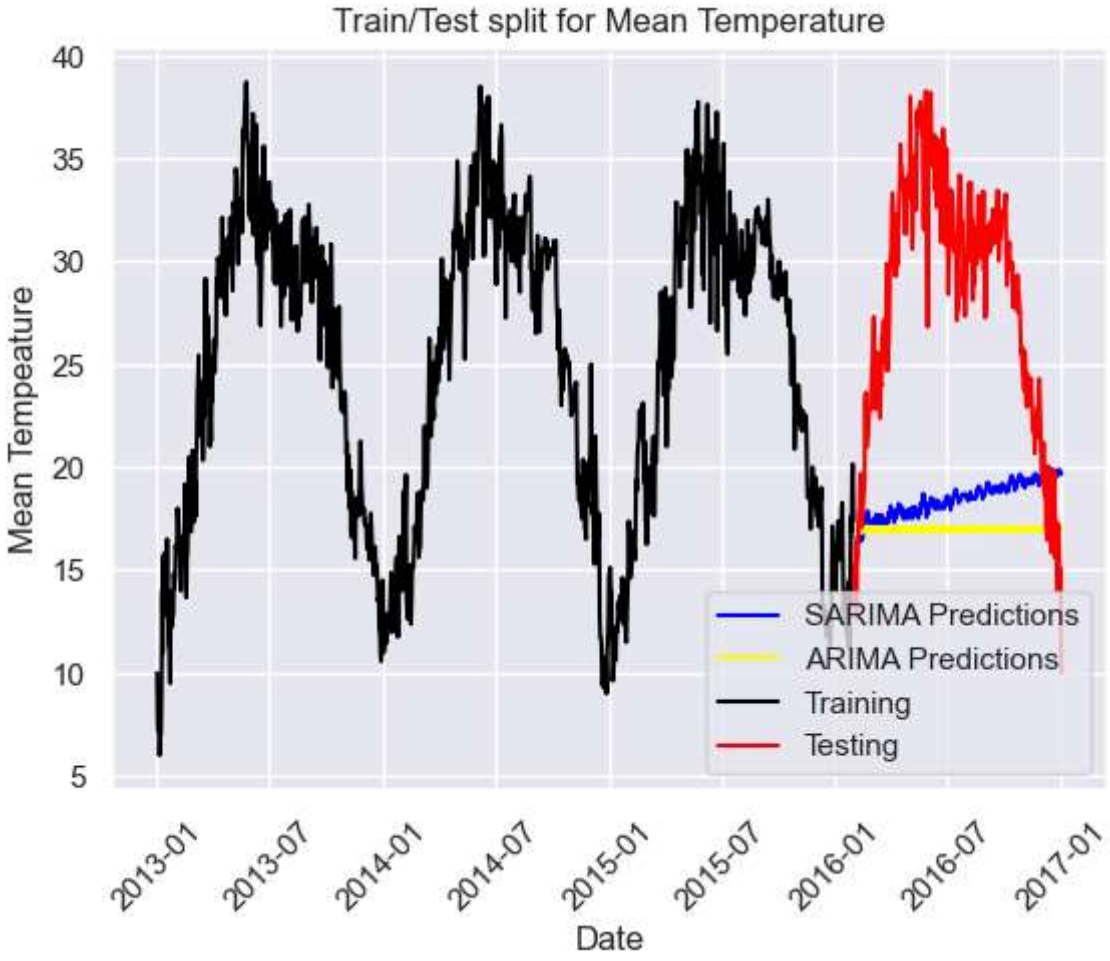
```
plt.plot(y_pred_out_3, color='Blue', label = 'SARIMA Predictions')
plt.plot(y_pred_out_2, color='Yellow', label = 'ARIMA Predictions')
plt.plot(train2.index, train2['meantemp'], color = "black", label = 'Training')
plt.plot(test2.index, test2['meantemp'], color = "red", label = 'Testing')
plt.ylabel('Mean Tempeature')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.title("Train/Test split for Mean Temperature")
plt.legend()

sarima_rmse = np.sqrt(mean_squared_error(test2["meantemp"].values, y_pred_df_3["Predic
```

```
print("SARIMA RMSE: ",sarima_rmse)

arima_rmse = np.sqrt(mean_squared_error(test2["meantemp"].values, y_pred_df_2["Predict
print("ARIMA RMSE: ",arima_rmse)
```

```
SARIMA RMSE:  11.588246557333134
ARIMA RMSE:  12.774350424101097
```



After trying many different parameter, I am starting to realize that this ARIMA and SARIMA model is not good for seasonal time series data like the one I have here. It just does not do the job in capturing the seasonalities. I will look for better models to use and try them.

FB Prophet Model

After some research, i wwas able to identify the fact that a prophet model might be best for my datat because of the amount of seasonality in this data. I will attempt to implement it here and test out my results compared to the ARIMA and SARIMA models.

In [77]:
```
!pip install prophet
```

```
Collecting prophet
  Downloading prophet-1.1.2-py3-none-win_amd64.whl (12.1 MB)
     ---------------------------------------- 12.1/12.1 MB 12.1 MB/s eta 0:00:00
Requirement already satisfied: matplotlib>=2.0.0 in c:\users\jordy\anaconda3\lib\site
-packages (from prophet) (3.5.2)
Requirement already satisfied: python-dateutil>=2.8.0 in c:\users\jordy\anaconda3\lib
\site-packages (from prophet) (2.8.2)
Collecting cmdstanpy>=1.0.4
  Downloading cmdstanpy-1.1.0-py3-none-any.whl (83 kB)
     ---------------------------------------- 83.2/83.2 kB 4.6 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.36.1 in c:\users\jordy\anaconda3\lib\site-pack
ages (from prophet) (4.64.1)
Requirement already satisfied: numpy>=1.15.4 in c:\users\jordy\anaconda3\lib\site-pac
kages (from prophet) (1.21.5)
Requirement already satisfied: pandas>=1.0.4 in c:\users\jordy\anaconda3\lib\site-pac
kages (from prophet) (1.4.4)
Collecting convertdate>=2.1.2
  Downloading convertdate-2.4.0-py3-none-any.whl (47 kB)
     ---------------------------------------- 47.9/47.9 kB ? eta 0:00:00
Collecting LunarCalendar>=0.0.9
  Downloading LunarCalendar-0.0.9-py2.py3-none-any.whl (18 kB)
Collecting holidays>=0.14.2
  Downloading holidays-0.24-py3-none-any.whl (499 kB)
     ---------------------------------------- 499.9/499.9 kB 30.6 MB/s eta 0:00:00
Collecting pymeeus<=1,>=0.3.13
  Downloading PyMeeus-0.5.12.tar.gz (5.8 MB)
     ---------------------------------------- 5.8/5.8 MB 12.2 MB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting korean-lunar-calendar
  Downloading korean_lunar_calendar-0.3.1-py3-none-any.whl (9.0 kB)
Collecting hijri-converter
  Downloading hijri_converter-2.3.1-py3-none-any.whl (13 kB)
Collecting ephem>=3.7.5.3
  Downloading ephem-4.1.4-cp39-cp39-win_amd64.whl (1.4 MB)
     ---------------------------------------- 1.4/1.4 MB 18.2 MB/s eta 0:00:00
Requirement already satisfied: pytz in c:\users\jordy\anaconda3\lib\site-packages (fr
om LunarCalendar>=0.0.9->prophet) (2022.1)
Requirement already satisfied: packaging>=20.0 in c:\users\jordy\anaconda3\lib\site-p
ackages (from matplotlib>=2.0.0->prophet) (21.3)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\jordy\anaconda3\lib\site
-packages (from matplotlib>=2.0.0->prophet) (1.4.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\jordy\anaconda3\lib\site
-packages (from matplotlib>=2.0.0->prophet) (4.25.0)
Requirement already satisfied: cycler>=0.10 in c:\users\jordy\anaconda3\lib\site-pack
ages (from matplotlib>=2.0.0->prophet) (0.11.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\jordy\anaconda3\lib\site-pac
kages (from matplotlib>=2.0.0->prophet) (9.2.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\jordy\anaconda3\lib\site-
packages (from matplotlib>=2.0.0->prophet) (3.0.9)
Requirement already satisfied: six>=1.5 in c:\users\jordy\anaconda3\lib\site-packages
(from python-dateutil>=2.8.0->prophet) (1.16.0)
Requirement already satisfied: colorama in c:\users\jordy\anaconda3\lib\site-packages
(from tqdm>=4.36.1->prophet) (0.4.5)
Building wheels for collected packages: pymeeus
  Building wheel for pymeeus (setup.py): started
  Building wheel for pymeeus (setup.py): finished with status 'done'
  Created wheel for pymeeus: filename=PyMeeus-0.5.12-py3-none-any.whl size=732001 sha
256=ec64b24006ad4aa2a7d23a89ba0914728d32f7f4e7c362a961ccda4cd91031b6
  Stored in directory: c:\users\jordy\appdata\local\pip\cache\wheels\04\1f\e5\8dd0c66
```

```
1cd8d252817655dc14a84f7ae045d6616594145aa81
Successfully built pymeeus
Installing collected packages: pymeeus, korean-lunar-calendar, ephem, hijri-converte
r, convertdate, LunarCalendar, holidays, cmdstanpy, prophet
Successfully installed LunarCalendar-0.0.9 cmdstanpy-1.1.0 convertdate-2.4.0 ephem-4.
1.4 hijri-converter-2.3.1 holidays-0.24 korean-lunar-calendar-0.3.1 prophet-1.1.2 pym
eeus-0.5.12
```

In [82]:
```
from prophet import Prophet
from prophet.plot import plot_plotly, plot_components_plotly
```

In [95]:
```
#Pulling up my data again to be stored under a new dictionary to be easier to manipula
prophet_climate=pd.read_csv("C:\\Users\\Jordy\\Desktop\\DailyDelhiClimateTrain.csv")
prophet_climate.head()
```

Out[95]:

	date	meantemp	humidity	wind_speed	meanpressure
0	2013-01-01	10.000000	84.500000	0.000000	1015.666667
1	2013-01-02	7.400000	92.000000	2.980000	1017.800000
2	2013-01-03	7.166667	87.000000	4.633333	1018.666667
3	2013-01-04	8.666667	71.333333	1.233333	1017.166667
4	2013-01-05	6.000000	86.833333	3.700000	1016.500000

In [97]:
```
#To create my model, I am only interested in the date and the meantemp so I am definin
prophet_climate2=prophet_climate[['date','meantemp']]
prophet_climate2.head()
```

Out[97]:

	date	meantemp
0	2013-01-01	10.000000
1	2013-01-02	7.400000
2	2013-01-03	7.166667
3	2013-01-04	8.666667
4	2013-01-05	6.000000

In [98]:
```
#Renaming date and meantemp columns because that is the labels that the prophet model
prophet_climate2.columns = ['ds','y']
```

In [100…
```
prophet_climate2.head()
```

Out[100]:

	ds	y
0	2013-01-01	10.000000
1	2013-01-02	7.400000
2	2013-01-03	7.166667
3	2013-01-04	8.666667
4	2013-01-05	6.000000

In [101…
```
#Making sure that my date(ds) is converted to date time format to be able to be read b
prophet_climate2['ds']=pd.to_datetime(prophet_climate2['ds'])
```

```
C:\Users\Jordy\AppData\Local\Temp\ipykernel_214960\1095441077.py:1: SettingWithCopyWa
rning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  prophet_climate2['ds']=pd.to_datetime(prophet_climate2['ds'])
```

In [103…
```
prophet_climate2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1462 entries, 0 to 1461
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ds      1462 non-null   datetime64[ns]
 1   y       1462 non-null   float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 23.0 KB
```

Train/Test Split

In [118…
```
#Setting up my training and testing splits so that my test data is for the last 365 da
trainp = prophet_climate2.iloc[:len(prophet_climate2)-365]
testp = prophet_climate2.iloc[len(prophet_climate2)-365:]
```

Model Building

In [119…
```
#Setting up the dictionaries for my model and running the model
m=Prophet()
m.fit(trainp)
future=m.make_future_dataframe(periods=365)
forecast=m.predict(future)
```

```
03:47:28 - cmdstanpy - INFO - Chain [1] start processing
03:47:28 - cmdstanpy - INFO - Chain [1] done processing
```

In [126…
```
forecast.tail(5)
```

Out[126]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_te
1457	2016-12-28	25.502582	10.157044	15.104973	25.216136	25.795188	-12.992934	
1458	2016-12-29	25.503282	9.790280	14.929746	25.215006	25.797102	-13.229643	
1459	2016-12-30	25.503981	9.512914	14.426487	25.214009	25.799015	-13.360205	
1460	2016-12-31	25.504681	9.583960	14.653934	25.213698	25.801471	-13.452698	
1461	2017-01-01	25.505381	9.346984	14.237332	25.213387	25.804093	-13.579498	

In [127…

```
forecast[['ds','yhat']].tail()
```

Out[127]:

	ds	yhat
1457	2016-12-28	12.509648
1458	2016-12-29	12.273639
1459	2016-12-30	12.143777
1460	2016-12-31	12.051983
1461	2017-01-01	11.925883

As you can see here, all that we are really worried about is the yhat value and the date. The yhat value tells us essentially what the models predictions are for the given date.

In [107…

```
prophet_climate2.tail()
```

Out[107]:

	ds	y
1457	2016-12-28	17.217391
1458	2016-12-29	15.238095
1459	2016-12-30	14.095238
1460	2016-12-31	15.052632
1461	2017-01-01	10.000000

You can see here that if you compare the different days and their meantemp to the forecast yhat(essentially the models predictions), they seem to be very close and able to predict better than our ARIMA and SARIMA models. We will further explore how much better we did when we calculate our RMSE

Visualizing Results From Prophet Model

In [121…

```
plot_plotly(m, forecast)
```

Here we see that the solid blue line is the predictions of my model and the black dots are my actual values. As you can see, the model did pretty well in predicting the values. It did much better than using ARIMA and SARIMA.

In [122…
```
from statsmodels.tools.eval_measures import rmse
```

In [123…
```
predictions= forecast.iloc[-365:]['yhat']
```

In [125…
```
print('Root Mean Squared Error Is: ',rmse(predictions,testp['y']))
```

```
Root Mean Squared Error Is:  2.8156775368652815
```

Hear we have calculated our RMSE for the prophet model which is SIGNIFICANTLY lower than what we were getting for the ARIMA and SARIMA models, so we can easily say that this model is probably best for forecasting our data.

In [ ]:
```
```