Rodriguez_Felipe_DSC680_Milestone3_Code

May 5, 2024

[140]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[141]:
```
df = pd.read_csv('london_weather.csv')
```

[142]:
```
df.head()
```

[142]:
```
       date  cloud_cover  sunshine  global_radiation  max_temp  mean_temp  \
0  19790101          2.0       7.0              52.0       2.3       -4.1
1  19790102          6.0       1.7              27.0       1.6       -2.6
2  19790103          5.0       0.0              13.0       1.3       -2.8
3  19790104          8.0       0.0              13.0      -0.3       -2.6
4  19790105          6.0       2.0              29.0       5.6       -0.8

   min_temp  precipitation  pressure  snow_depth
0      -7.5            0.4  101900.0         9.0
1      -7.5            0.0  102530.0         8.0
2      -7.2            0.0  102050.0         4.0
3      -6.5            0.0  100840.0         2.0
4      -1.4            0.0  102250.0         1.0
```

[5]:
```
df.dtypes
```

[5]:
```
date                  int64
cloud_cover         float64
sunshine            float64
global_radiation    float64
max_temp            float64
mean_temp           float64
min_temp            float64
precipitation       float64
pressure            float64
snow_depth          float64
dtype: object
```

[132]:
```
description = {
    'date': 'Date of record in YYYYMMDD Format',
```

```
    'cloud_cover': 'Cloud cover measurement in oktas',
    'sunshine': 'Sunshine measure in hours',
    'global_radiation': 'Irradiance measurement in Watt per square meter (W/
 ↪m2)',
    'max_temp': 'Maximum temperature recorded in degrees Celsius (°C)',
    'mean_temp': 'Mean temperature recorded in degrees Celsius (°C)',
    'min_temp': 'Min temperature recorded in degrees Celsius (°C)',
    'precipitation': 'Precipitation measurement in millimeters (mm)',
    'pressure': 'Pressure measurement in Pascals (Pa)',
    'snow_depth': 'Snow depth measurement in centimeters (cm)'
              }
# Initialize an empty dictionary to store data types
dtype_dict = {}

# Iterate through each column and store its data type in the dictionary
for col in df.columns:
    dtype_dict[col] = str(df[col].dtype)

series1 = pd.Series(description, name='description')
series1 = series1.rename_axis('column')
series2 = pd.Series(dtype_dict, name='data_type')
series2 = series2.rename_axis('column')



# Combining the Series into a DataFrame using pd.merge()
data_dictionary = pd.merge(series1, series2, left_index=True, right_index=True)
print('Data Dictionary\n')
print(data_dictionary.to_markdown())
```

```
Data Dictionary

| column           | description                                             |
data_type   |
|:-----------------|:--------------------------------------------------------|:--
----------|
| date             | Date of record in YYYYMMDD Format                       |
object      |
| cloud_cover      | Cloud cover measurement in oktas                        |
float64     |
| sunshine         | Sunshine measure in hours                               |
float64     |
| global_radiation | Irradiance measurement in Watt per square meter (W/m2) |
float64     |
| max_temp         | Maximum temperature recorded in degrees Celsius (°C)    |
float64     |
| mean_temp        | Mean temperature recorded in degrees Celsius (°C)       |
```

```
float64      |
| min_temp          | Min temperature recorded in degrees Celsius (°C)        |
float64      |
| precipitation     | Precipitation measurement in millimeters (mm)           |
float64      |
| pressure          | Pressure measurement in Pascals (Pa)                    |
float64      |
| snow_depth        | Snow depth measurement in centimeters (cm)              |
float64      |
```

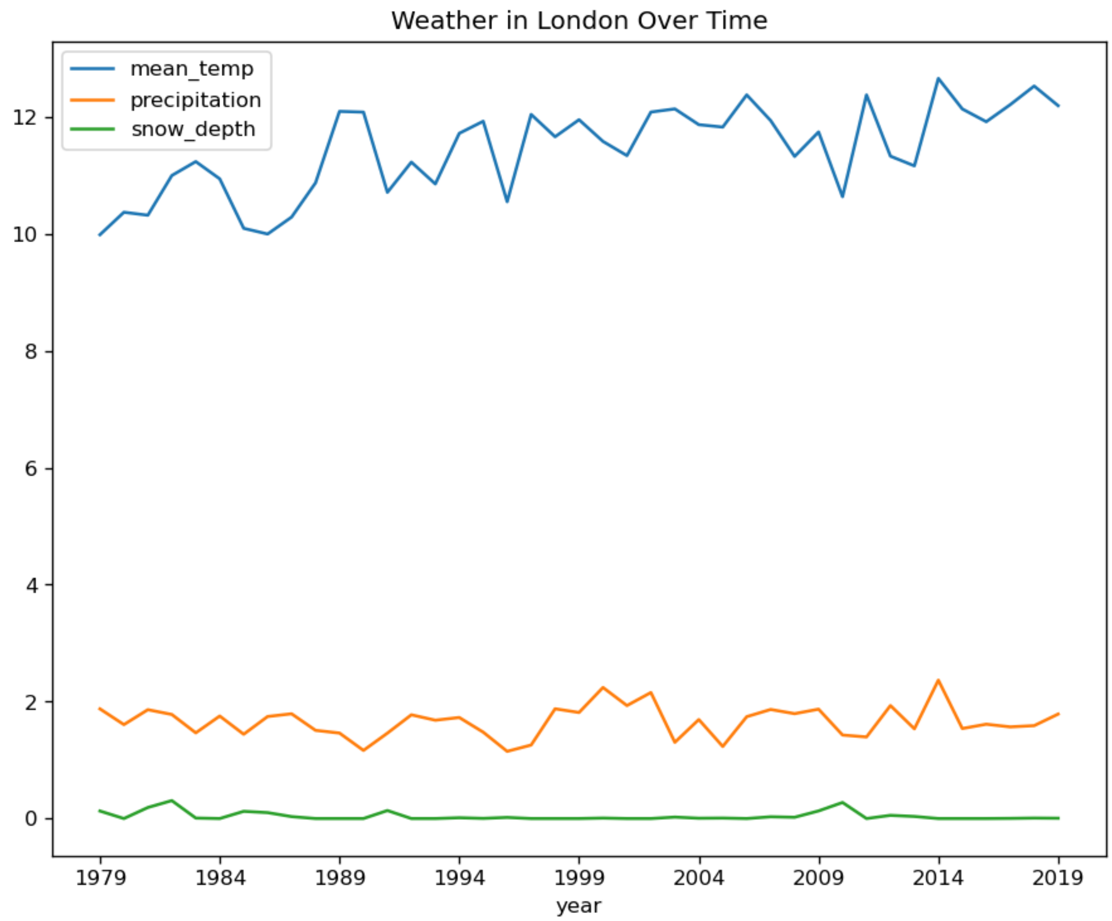
[149]:
```
df['date'] = df['date'].astype(str)
df['year'] = df['date'].str[0:4]
df['month'] = df['date'].str[4:6]
df['day'] = df['date'].str[6:]
df.head()
```

[149]:
```
       date  cloud_cover  sunshine  global_radiation  max_temp  mean_temp  \
0  19790101          2.0       7.0              52.0       2.3       -4.1
1  19790102          6.0       1.7              27.0       1.6       -2.6
2  19790103          5.0       0.0              13.0       1.3       -2.8
3  19790104          8.0       0.0              13.0      -0.3       -2.6
4  19790105          6.0       2.0              29.0       5.6       -0.8

   min_temp  precipitation  pressure  snow_depth  year month day
0      -7.5            0.4  101900.0         9.0  1979    01  01
1      -7.5            0.0  102530.0         8.0  1979    01  02
2      -7.2            0.0  102050.0         4.0  1979    01  03
3      -6.5            0.0  100840.0         2.0  1979    01  04
4      -1.4            0.0  102250.0         1.0  1979    01  05
```

[16]:
```
averages = df.groupby('year')[['mean_temp', 'precipitation', 'snow_depth']].
  ↪mean()
averages.head()
```

[16]:
```
      mean_temp  precipitation  snow_depth
year
1979   9.986575       1.875890    0.128767
1980  10.370492       1.606831    0.000000
1981  10.320000       1.861918    0.189041
1982  10.998904       1.780274    0.306849
1983  11.237260       1.465753    0.008219
```

[133]:
```
averages.plot()
plt.title('Weather in London Over Time')
plt.show()
```



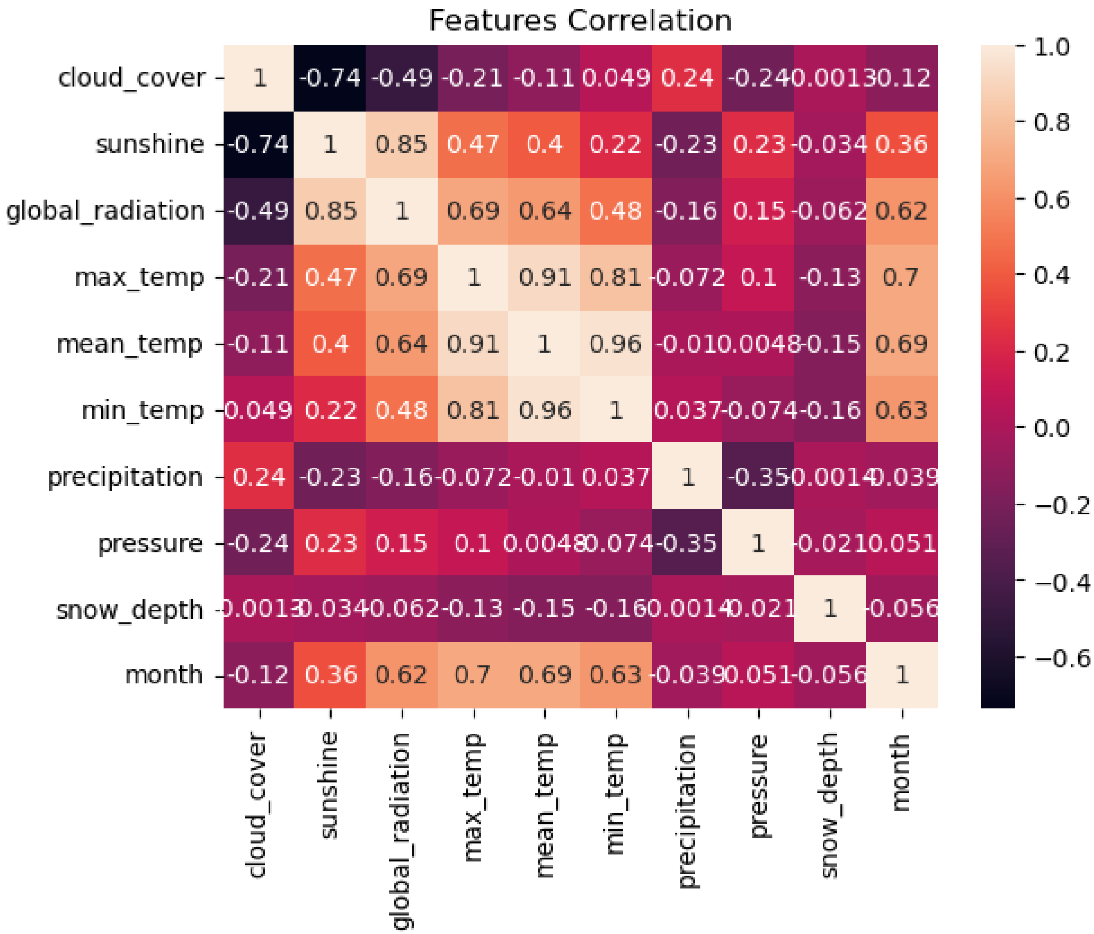
[28]:
```
import seaborn as sns
```

[36]:
```
df_num = df.drop(columns=['date', 'year', 'day'])
```

[37]:
```
corr = df_num.corr()
```

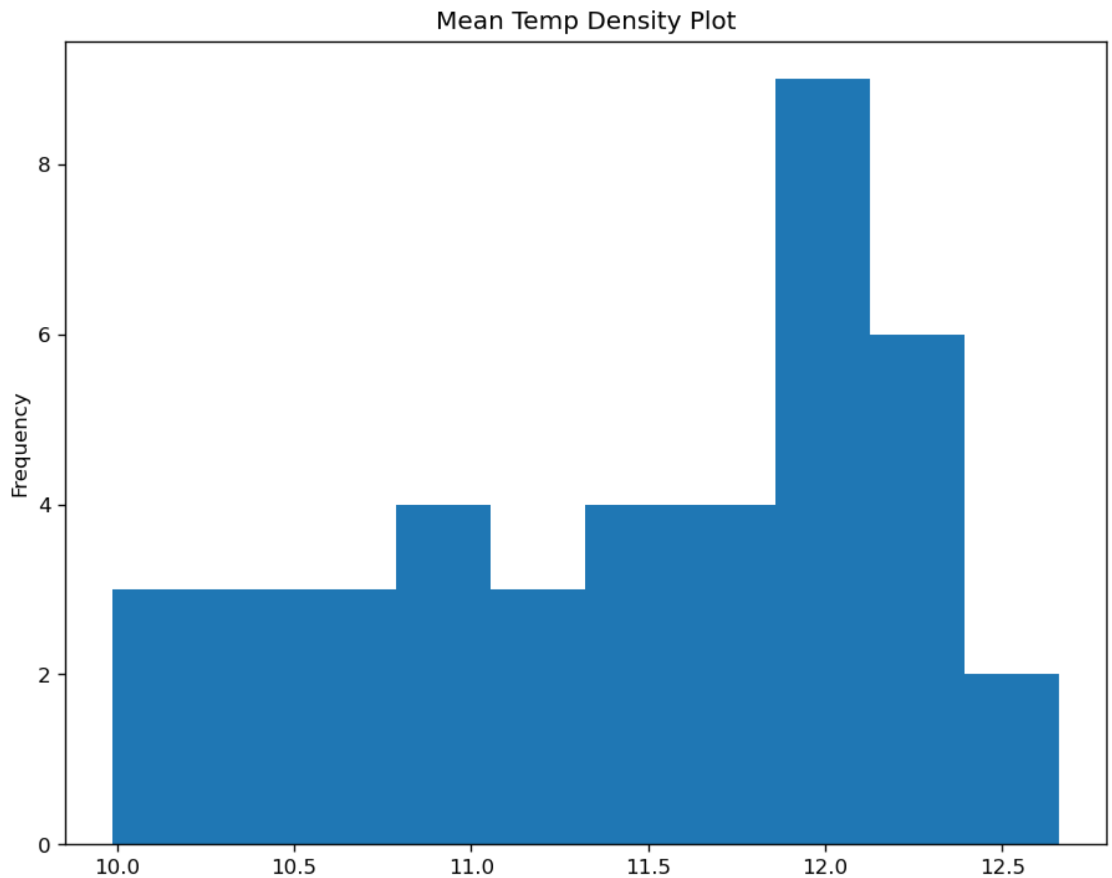
[38]:
```
sns.heatmap(corr, annot=True)
plt.title('Features Correlation')
```

[38]: Text(0.5, 1.0, 'Features Correlation')



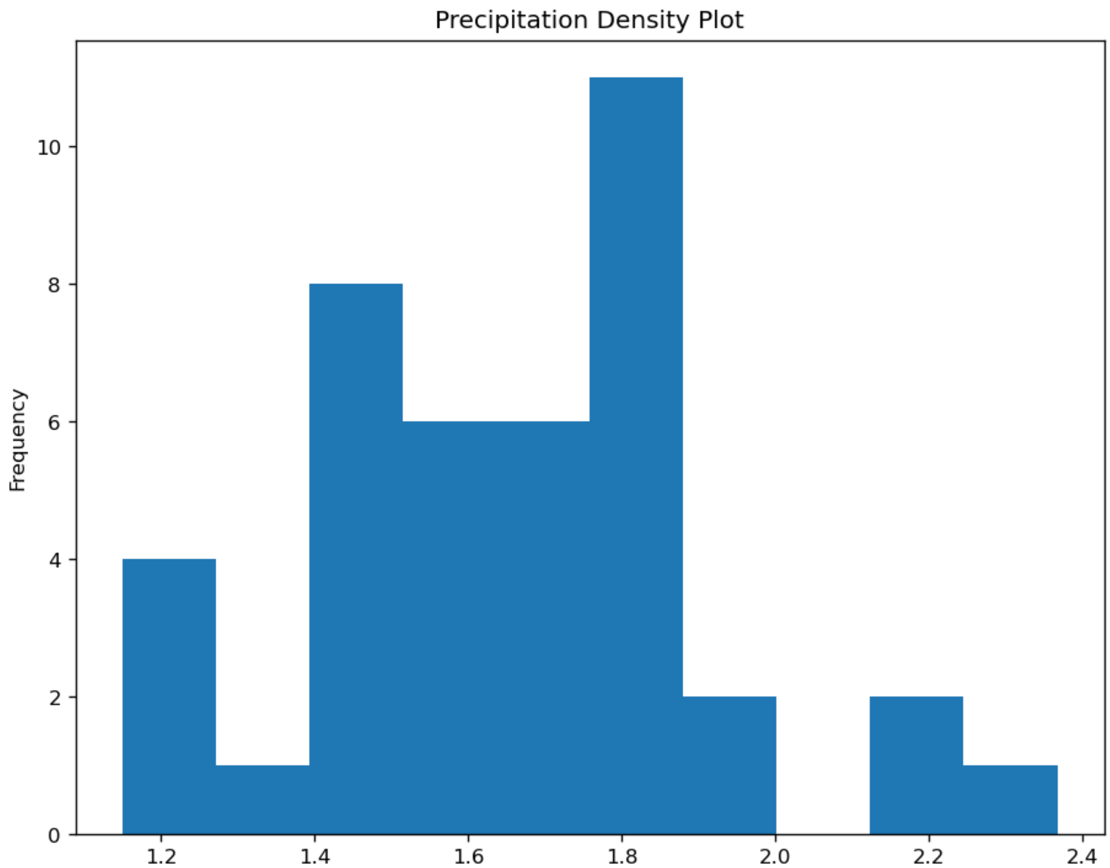
[134]:
```
averages['mean_temp'].plot(kind='hist')
plt.title('Mean Temp Density Plot')
```

[134]: Text(0.5, 1.0, 'Mean Temp Density Plot')



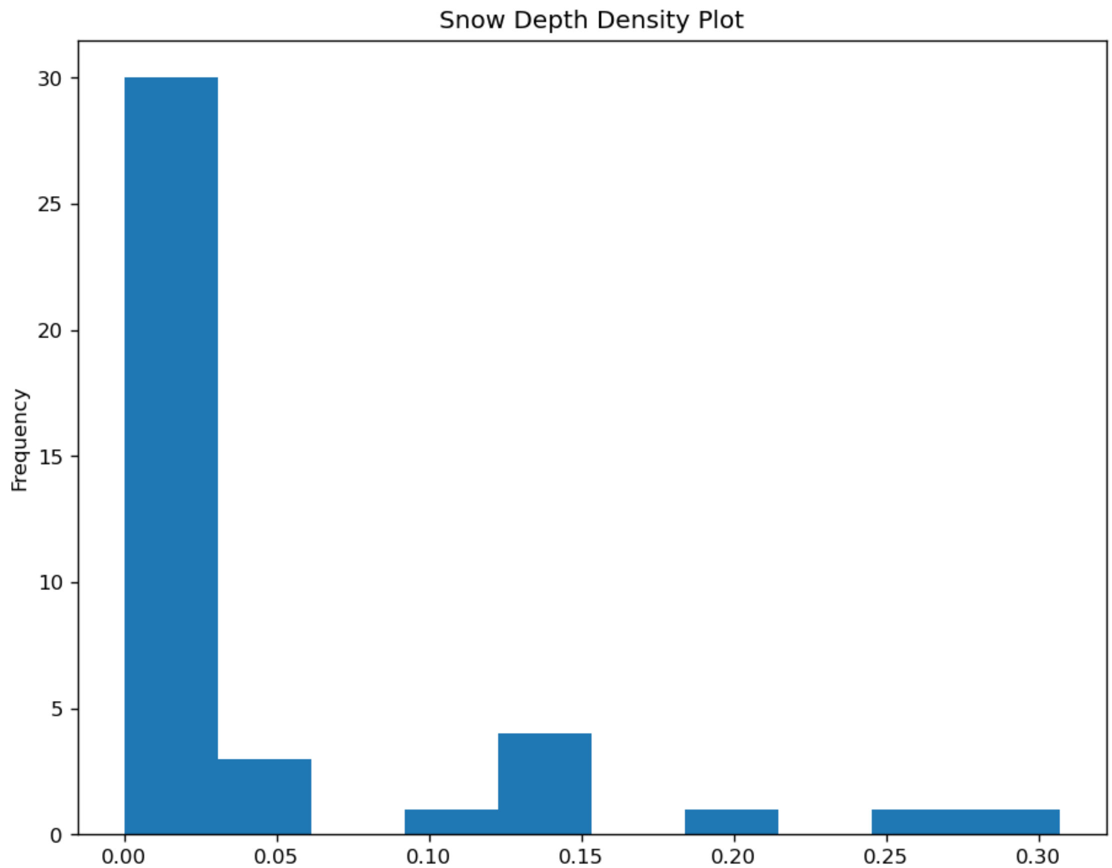
[135]:
```
averages['precipitation'].plot(kind='hist')
plt.title('Precipitation Density Plot')
```

[135]: Text(0.5, 1.0, 'Precipitation Density Plot')



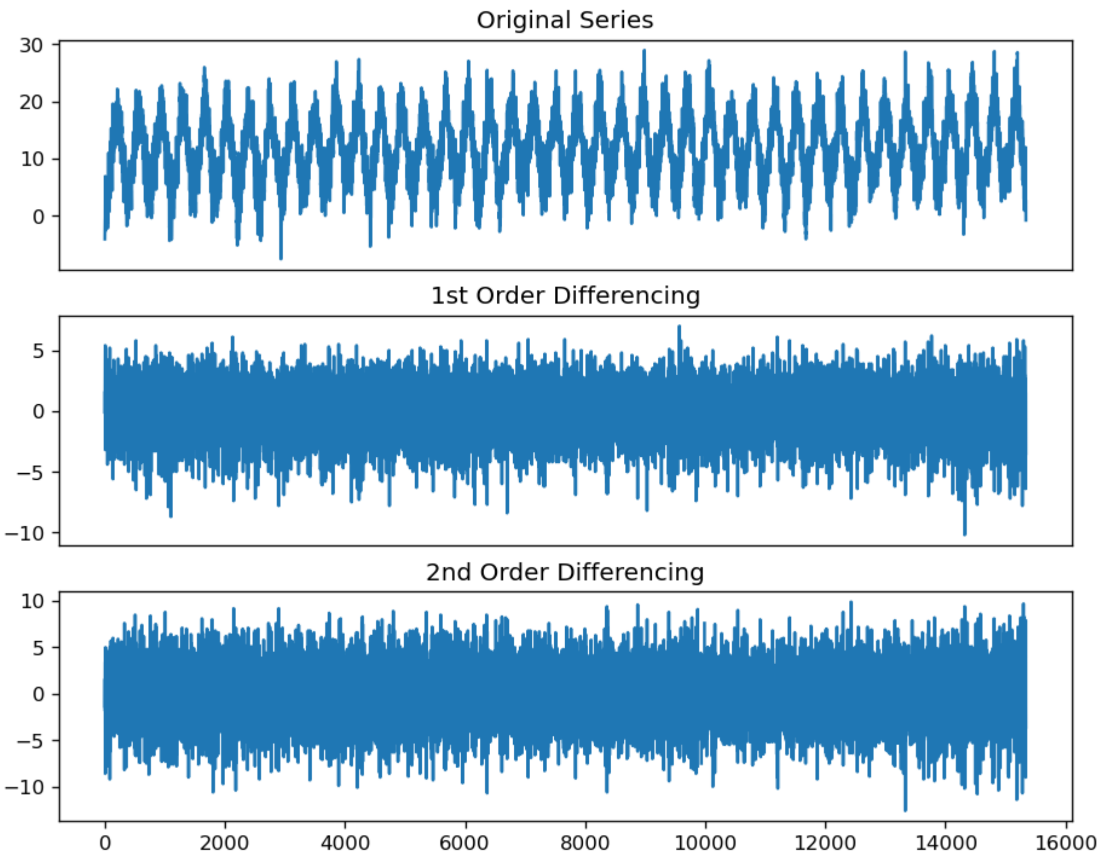
[136]:
```
averages['snow_depth'].plot(kind='hist')
plt.title('Snow Depth Density Plot')
```

[136]:
```
Text(0.5, 1.0, 'Snow Depth Density Plot')
```



[84]:

```
import numpy as np
plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})

# Original Series
fig, (ax1, ax2, ax3) = plt.subplots(3)
ax1.plot(df['mean_temp']); ax1.set_title('Original Series'); ax1.axes.xaxis.
 ↪set_visible(False)
# 1st Differencing
ax2.plot(df['mean_temp'].diff()); ax2.set_title('1st Order Differencing'); ax2.
 ↪axes.xaxis.set_visible(False)
# 2nd Differencing
ax3.plot(df['mean_temp'].diff().diff()); ax3.set_title('2nd Order Differencing')
plt.show()
```



[66]:
```
from statsmodels.tsa.stattools import adfuller
```

[68]:
```
adf_test = adfuller(averages['mean_temp'])
# Output the results
print('ADF Statistic: %f' % adf_test[0])
print('p-value: %f' % adf_test[1])
```

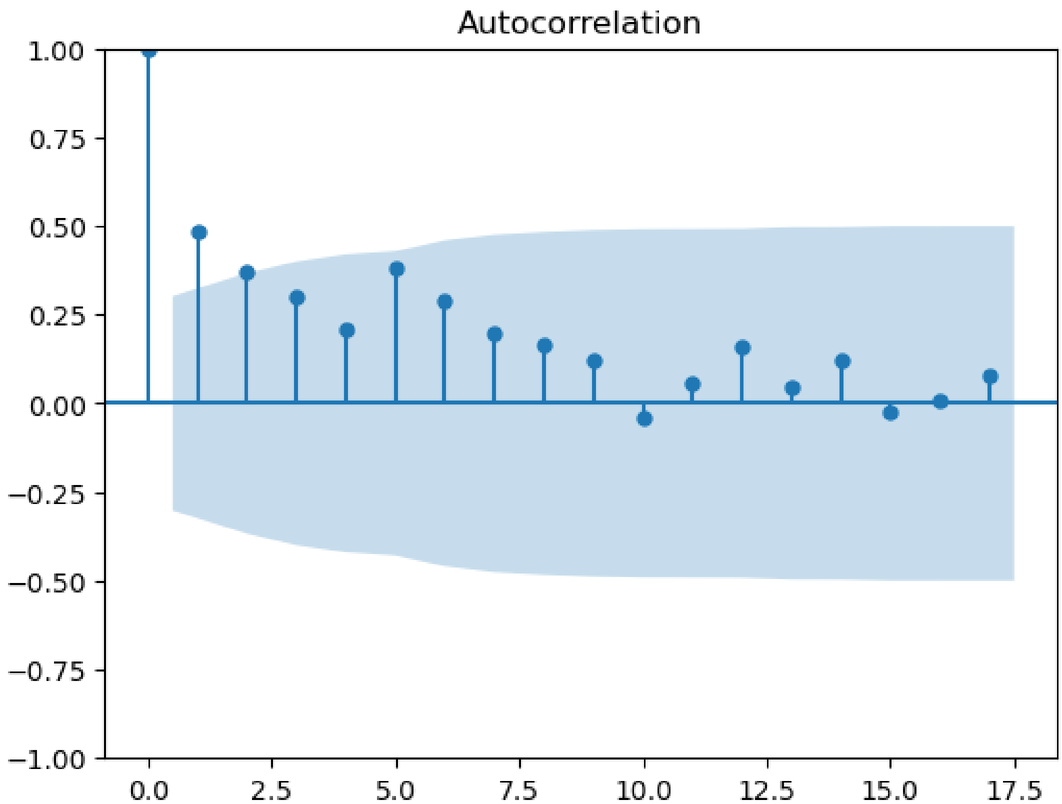
```
ADF Statistic: -3.584279
p-value: 0.006069
```

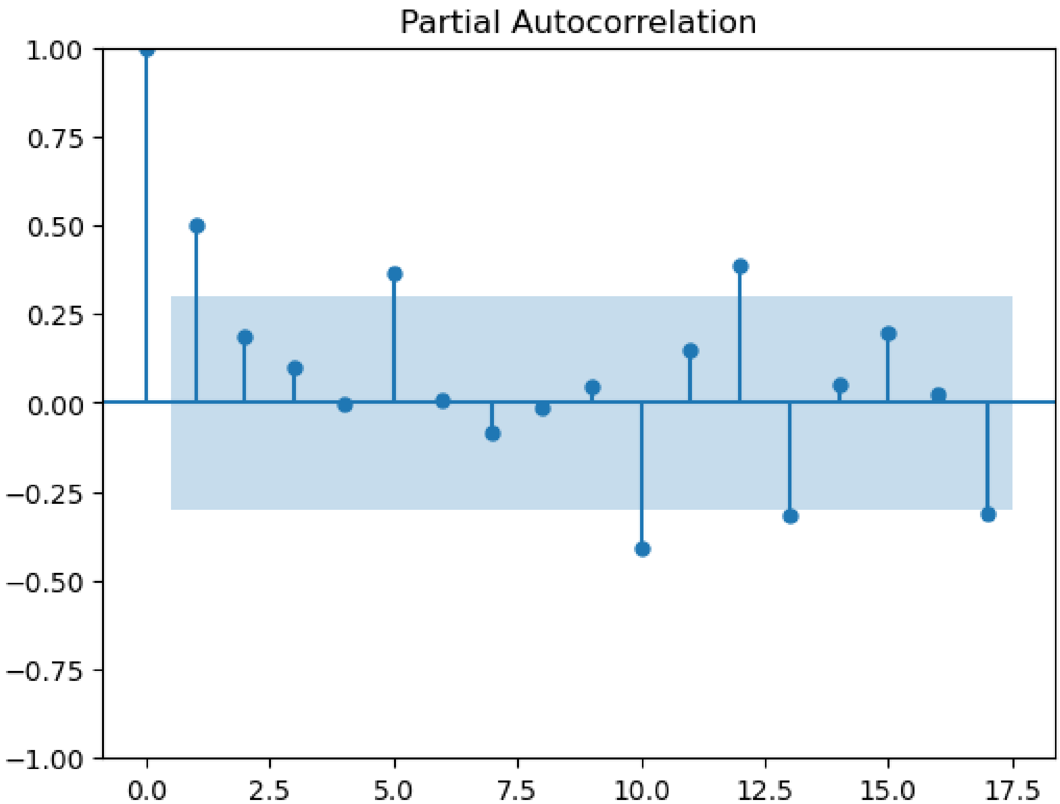
[55]:
```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

[65]:
```
plot_acf(averages['mean_temp'])
plot_pacf(averages['mean_temp'])
plt.show()
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method
'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the
```

```
default will change tounadjusted Yule-Walker ('ywm'). You can use this method
now by setting method='ywm'.
  warnings.warn(
```





[75]:
```
model = ARIMA(averages['mean_temp'], order = (1,1,1))
model_fit = model.fit()
model_fit.summary()
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
```

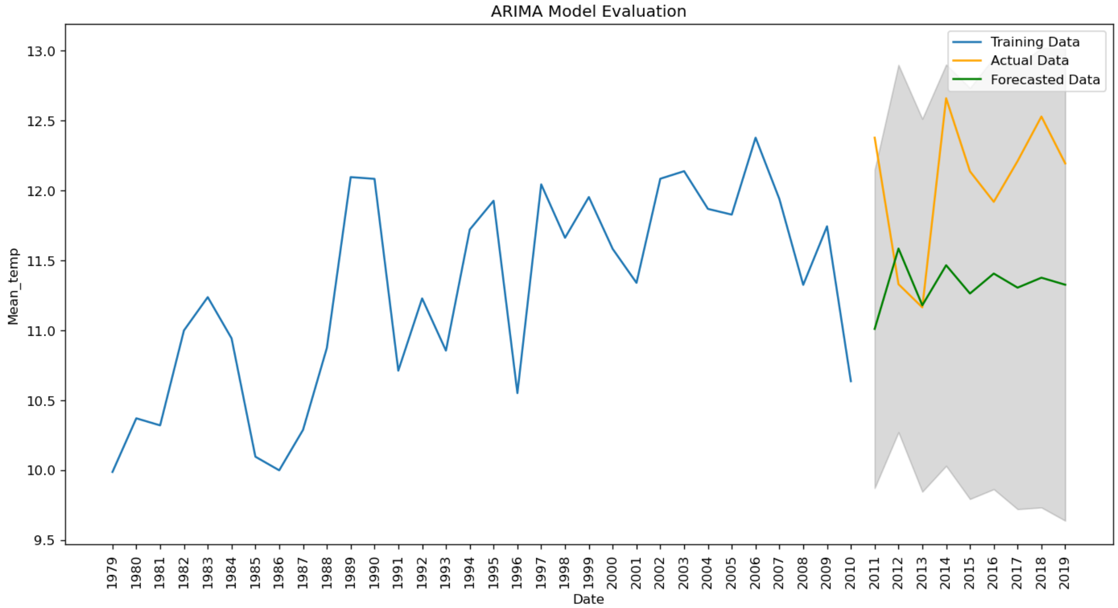
[75]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
```

```
Dep. Variable:              mean_temp   No. Observations:                   42
Model:                 ARIMA(1, 1, 1)   Log Likelihood                 -37.917
Date:                Sun, 21 Apr 2024   AIC                             81.833
Time:                        11:39:51   BIC                             86.974
Sample:                    01-01-1979   HQIC                            83.705
                         - 01-01-2020
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0961      0.243      0.396      0.692      -0.380       0.573
ma.L1         -0.7352      0.204     -3.598      0.000      -1.136      -0.335
sigma2         0.3664      0.090      4.077      0.000       0.190       0.543
================================================================================
===
Ljung-Box (L1) (Q):                   0.31   Jarque-Bera (JB):
0.12
Prob(Q):                              0.58   Prob(JB):
0.94
Heteroskedasticity (H):               0.88   Skew:
-0.06
Prob(H) (two-sided):                  0.82   Kurtosis:
2.76
================================================================================
===

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

[104]:
```python
from sklearn.metrics import mean_squared_error

# Split the data into train and test
train_size = int(len(averages) * 0.8)
train, test = averages[0:train_size], averages[train_size:len(averages)]

# Fit the ARIMA model on the training dataset
model_train = ARIMA(train['mean_temp'], order=(1, 1, 2))
model_train_fit = model_train.fit()

# Forecast on the test dataset
test_forecast = model_train_fit.get_forecast(steps=len(test))
test_forecast_series = pd.Series(test_forecast.predicted_mean, index=test.index)

# Calculate the mean squared error
mse = mean_squared_error(test['mean_temp'], test_forecast_series)
```

```python
rmse = mse**0.5

# Create a plot to compare the forecast with the actual test data
plt.figure(figsize=(14,7))
plt.plot(train['mean_temp'], label='Training Data')
plt.plot(test['mean_temp'], label='Actual Data', color='orange')
plt.plot(test_forecast_series, label='Forecasted Data', color='green')
plt.fill_between(test.index,
                 test_forecast.conf_int().iloc[:, 0],
                 test_forecast.conf_int().iloc[:, 1],
                 color='k', alpha=.15)
plt.title('Mean Temp ARIMA Model Evaluation')
plt.xlabel('Date')
plt.xticks(rotation='vertical')
plt.ylabel('Mean_temp')
plt.legend()
plt.show()

print('RMSE:', rmse)
```

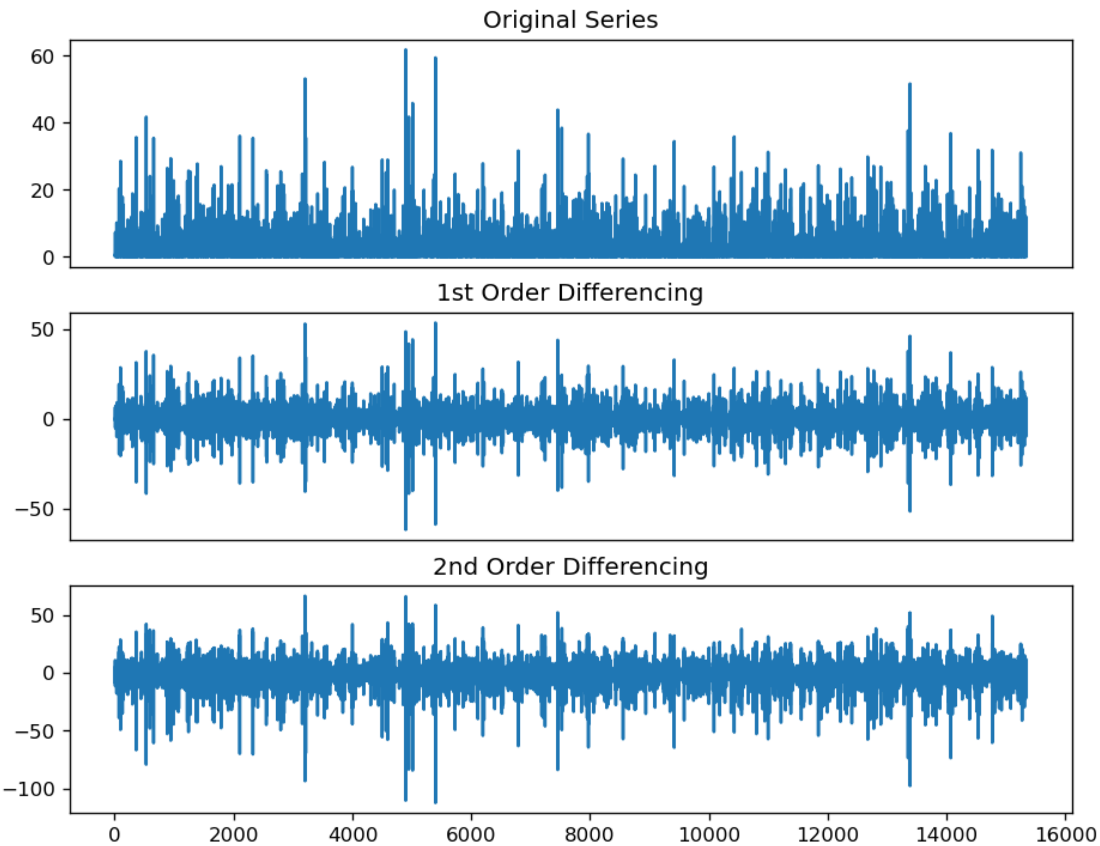
```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
```




```
RMSE: 0.9002148303139442
```

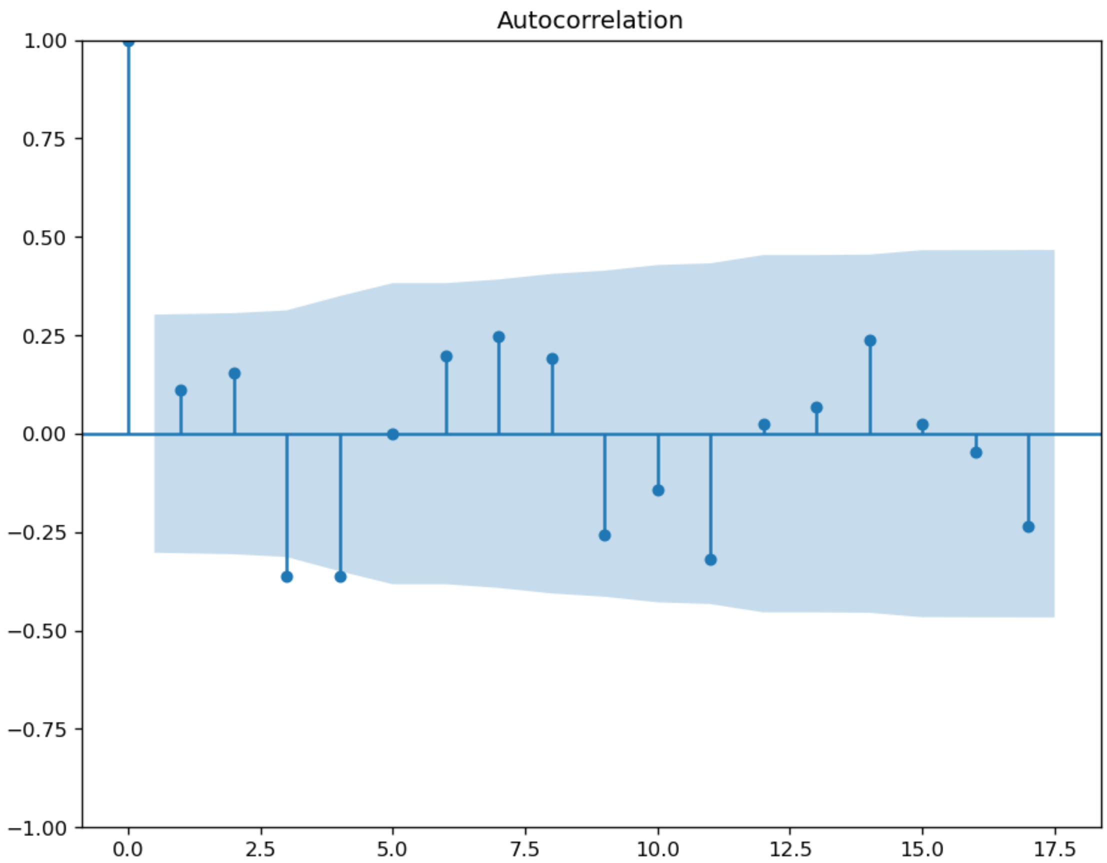
Precipitation

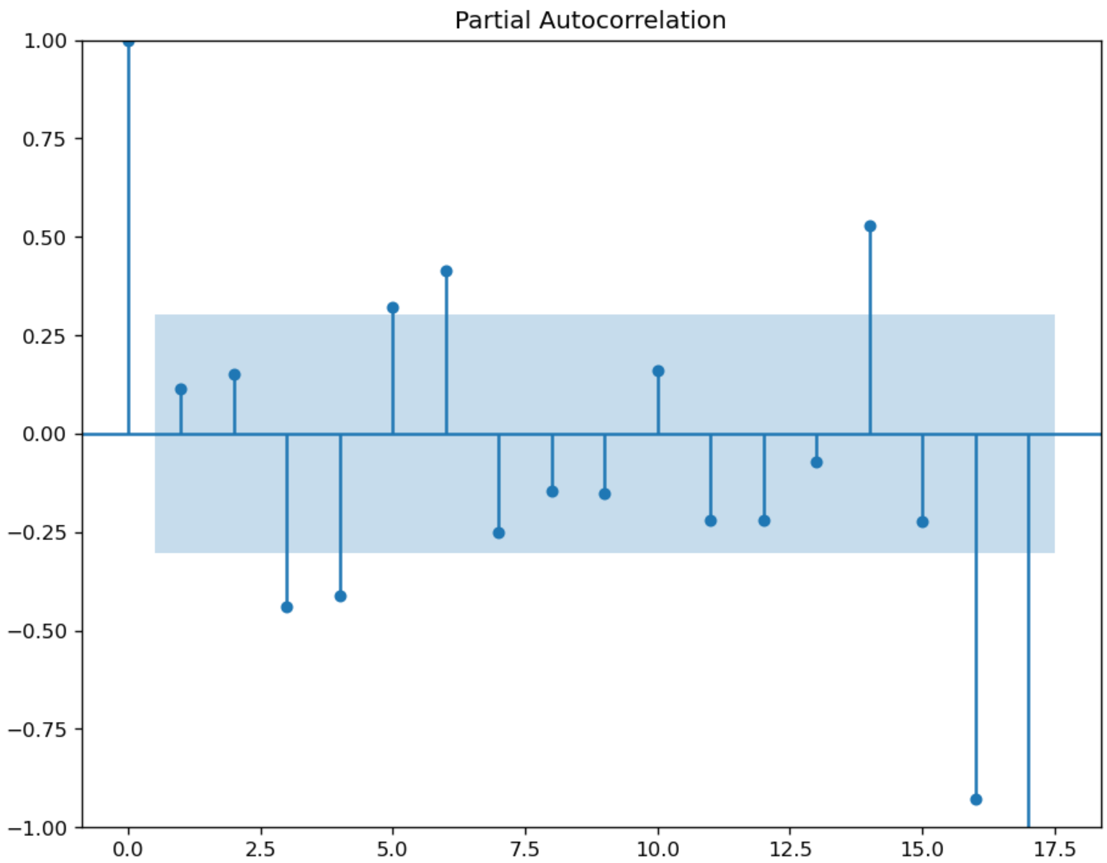
[85]:
```
plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})

# Original Series
fig, (ax1, ax2, ax3) = plt.subplots(3)
ax1.plot(df['precipitation']); ax1.set_title('Original Series'); ax1.axes.xaxis.
  ↪set_visible(False)
# 1st Differencing
ax2.plot(df['precipitation'].diff()); ax2.set_title('1st Order Differencing');␣
  ↪ax2.axes.xaxis.set_visible(False)
# 2nd Differencing
ax3.plot(df['precipitation'].diff().diff()); ax3.set_title('2nd Order␣
  ↪Differencing')
plt.show()
```



[86]:
```
plot_acf(averages['precipitation'])
plot_pacf(averages['precipitation'])
plt.show()
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method
'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the
default will change tounadjusted Yule-Walker ('ywm'). You can use this method
now by setting method='ywm'.
  warnings.warn(
```

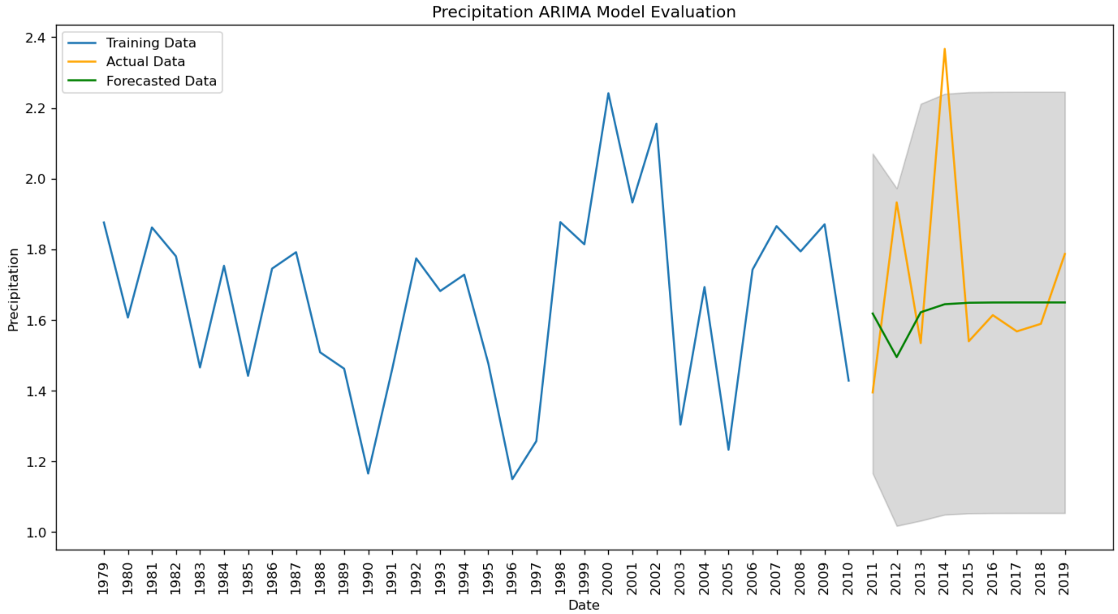

Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



[138]:
```
# Split the data into train and test
train_size = int(len(averages) * 0.8)
train, test = averages[0:train_size], averages[train_size:len(averages)]

# Fit the ARIMA model on the training dataset
model_train = ARIMA(train['precipitation'], order=(1, 1, 3))
model_train_fit = model_train.fit()

# Forecast on the test dataset
test_forecast = model_train_fit.get_forecast(steps=len(test))
test_forecast_series = pd.Series(test_forecast.predicted_mean, index=test.index)

# Calculate the mean squared error
mse = mean_squared_error(test['precipitation'], test_forecast_series)
rmse = mse**0.5

# Create a plot to compare the forecast with the actual test data
plt.figure(figsize=(14,7))
plt.plot(train['precipitation'], label='Training Data')
plt.plot(test['precipitation'], label='Actual Data', color='orange')
```

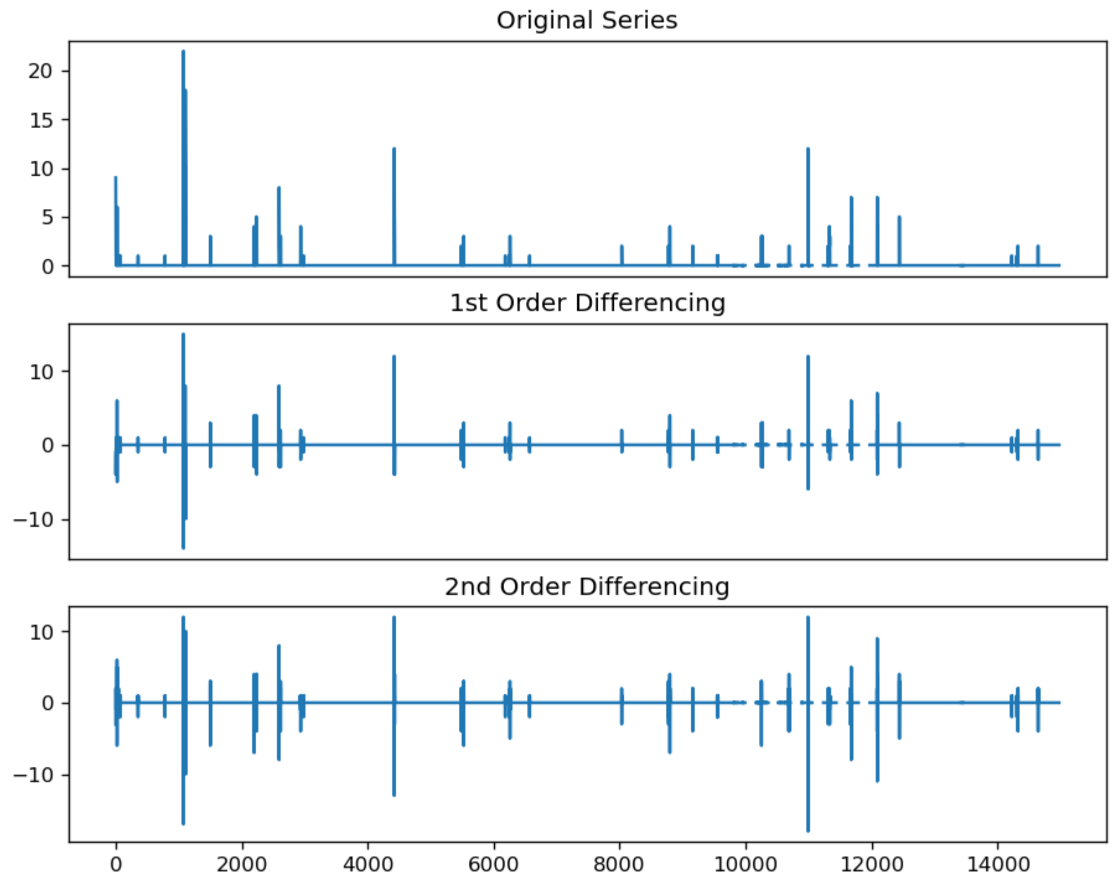
```
plt.plot(test_forecast_series, label='Forecasted Data', color='green')
plt.fill_between(test.index,
                 test_forecast.conf_int().iloc[:, 0],
                 test_forecast.conf_int().iloc[:, 1],
                 color='k', alpha=.15)
plt.title('Precipitation ARIMA Model Evaluation')
plt.xlabel('Date')
plt.xticks(rotation='vertical')
plt.ylabel('Precipitation')
plt.legend()
plt.show()

print('RMSE:', rmse)
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible
starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood
optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

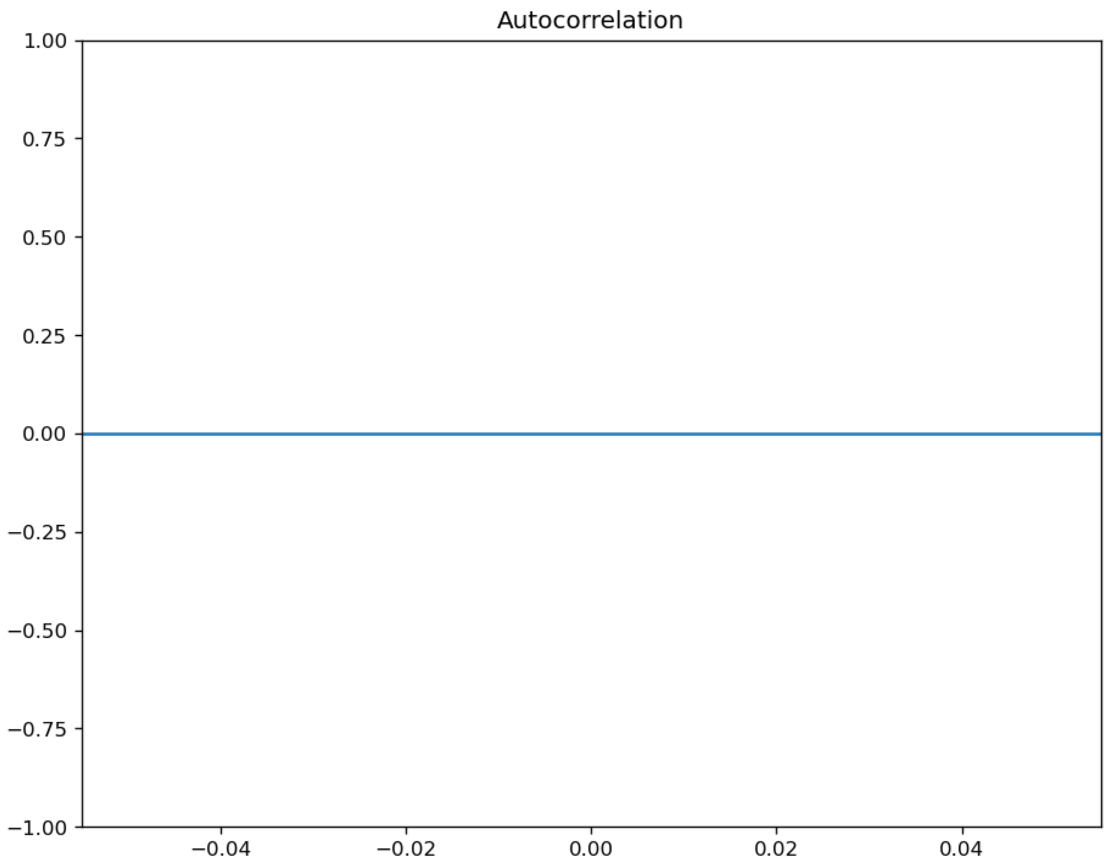


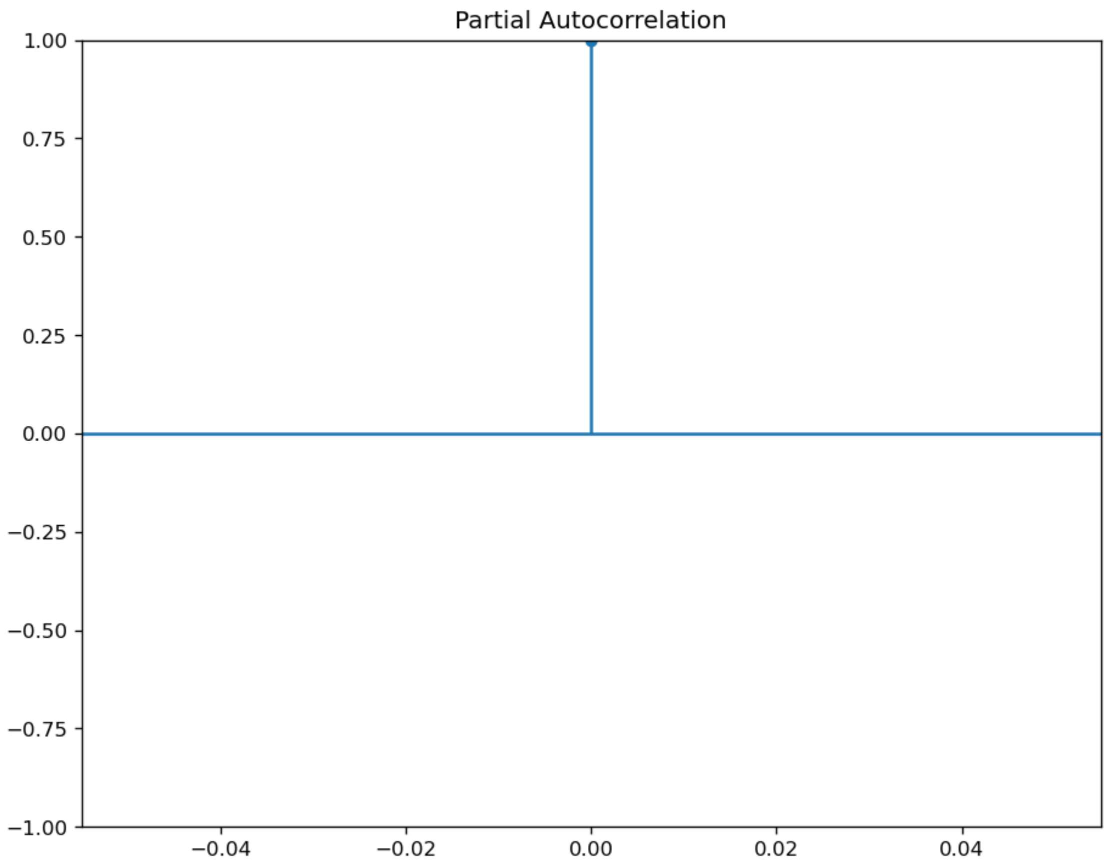
RMSE: 0.300663266736998

[90]:
```
plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})

# Original Series
fig, (ax1, ax2, ax3) = plt.subplots(3)
ax1.plot(df['snow_depth']); ax1.set_title('Original Series'); ax1.axes.xaxis.
↪set_visible(False)
# 1st Differencing
ax2.plot(df['snow_depth'].diff()); ax2.set_title('1st Order Differencing'); ax2.
↪axes.xaxis.set_visible(False)
# 2nd Differencing
ax3.plot(df['snow_depth'].diff().diff()); ax3.set_title('2nd Order␣
↪Differencing')
plt.show()
```



[95]:
```
plot_acf(averages['snow_depth'])
plot_pacf(averages['snow_depth'])
plt.show()
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.04
−0.02
0.00
0.02
0.04

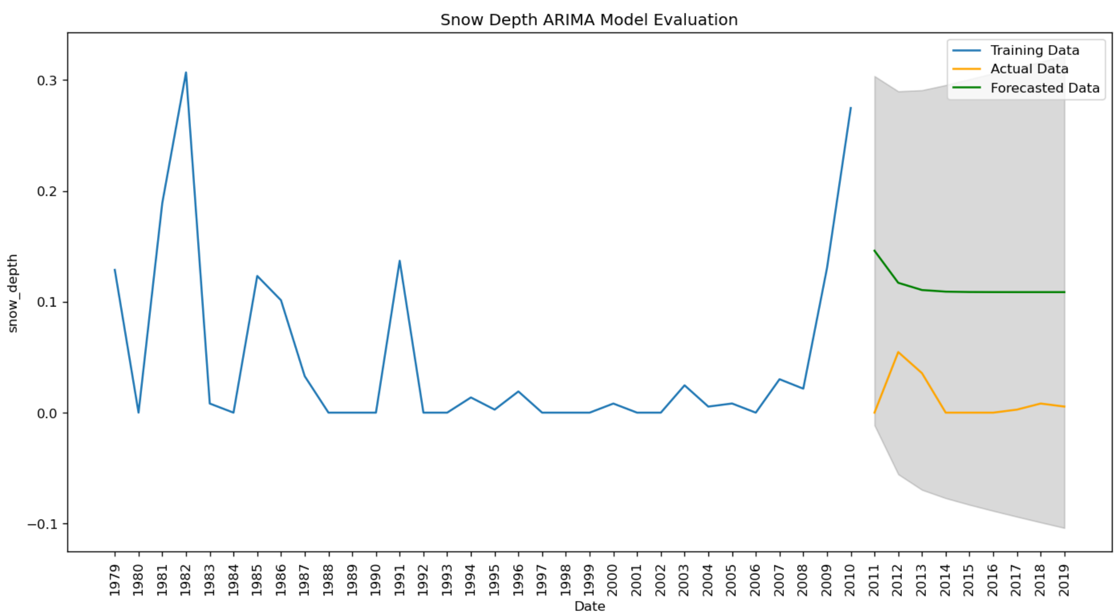


[97]:
```
averages = averages.dropna()
```

[139]:
```
# Split the data into train and test
train_size = int(len(averages) * 0.8)
train, test = averages[0:train_size], averages[train_size:len(averages)]

# Fit the ARIMA model on the training dataset
model_train = ARIMA(train['snow_depth'], order=(1, 1, 1))
model_train_fit = model_train.fit()

# Forecast on the test dataset
test_forecast = model_train_fit.get_forecast(steps=len(test))
test_forecast_series = pd.Series(test_forecast.predicted_mean, index=test.index)

# Calculate the mean squared error
mse = mean_squared_error(test['snow_depth'], test_forecast_series)
rmse = mse**0.5

# Create a plot to compare the forecast with the actual test data
plt.figure(figsize=(14,7))
```

```
plt.plot(train['snow_depth'], label='Training Data')
plt.plot(test['snow_depth'], label='Actual Data', color='orange')
plt.plot(test_forecast_series, label='Forecasted Data', color='green')
plt.fill_between(test.index,
                 test_forecast.conf_int().iloc[:, 0],
                 test_forecast.conf_int().iloc[:, 1],
                 color='k', alpha=.15)
plt.title('Snow Depth ARIMA Model Evaluation')
plt.xlabel('Date')
plt.xticks(rotation='vertical')
plt.ylabel('snow_depth')
plt.legend()
plt.show()

print('RMSE:', rmse)
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
```



```
RMSE: 0.10456593039572244
```

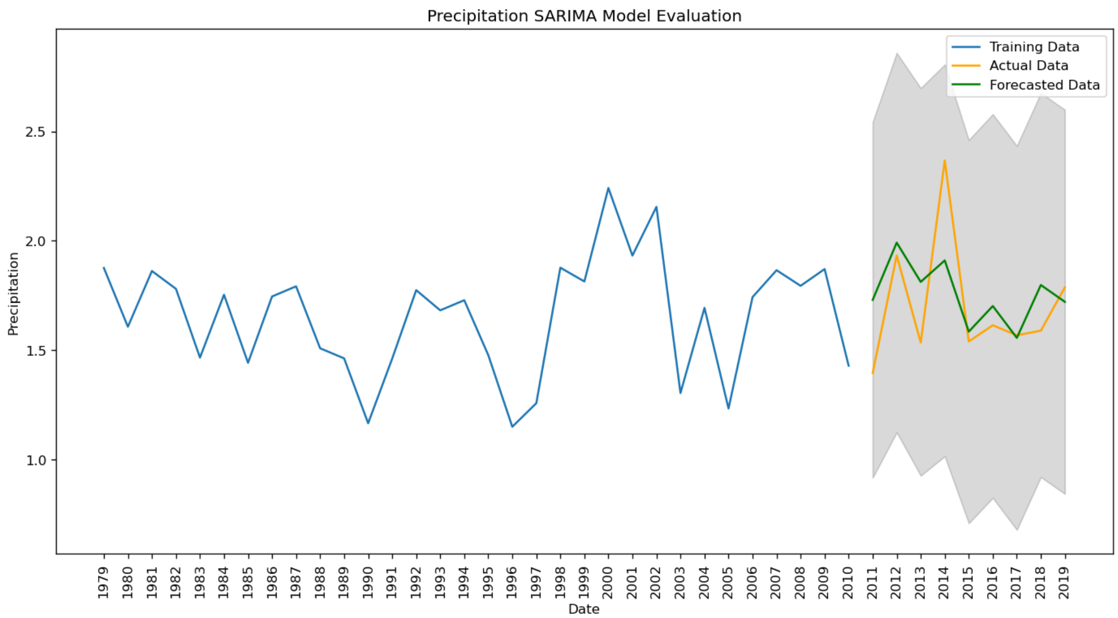
SARIMA Model

```
[137]: from statsmodels.tsa.statespace.sarimax import SARIMAX

# Split the data into train and test
train_size = int(len(averages) * 0.8)
train, test = averages[0:train_size], averages[train_size:len(averages)]

# Fit the SARIMA model on the training dataset
model_train = SARIMAX(train['precipitation'], order=(1, 1, 2),␣
 ↪seasonal_order=(1, 1, 2, 12))
model_train_fit = model_train.fit()

# Forecast on the test dataset
test_forecast = model_train_fit.get_forecast(steps=len(test))
test_forecast_series = pd.Series(test_forecast.predicted_mean, index=test.index)

# Calculate the mean squared error
mse = mean_squared_error(test['precipitation'], test_forecast_series)
rmse = mse**0.5

# Create a plot to compare the forecast with the actual test data
plt.figure(figsize=(14,7))
plt.plot(train['precipitation'], label='Training Data')
plt.plot(test['precipitation'], label='Actual Data', color='orange')
plt.plot(test_forecast_series, label='Forecasted Data', color='green')
plt.fill_between(test.index,
                 test_forecast.conf_int().iloc[:, 0],
                 test_forecast.conf_int().iloc[:, 1],
                 color='k', alpha=.15)
plt.title('Precipitation SARIMA Model Evaluation')
plt.xlabel('Date')
plt.xticks(rotation='vertical')
plt.ylabel('Precipitation')
plt.legend()
plt.show()

print('RMSE:', rmse)
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
```

```
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few
observations to estimate starting parameters for seasonal ARMA. All parameters
except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            7     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.19743D-01    |proj g|=  2.56820D-01

At iterate    5    f=  3.80875D-01    |proj g|=  6.07192D-02

At iterate   10    f=  3.75655D-01    |proj g|=  2.63465D-02

At iterate   15    f=  3.68063D-01    |proj g|=  2.14258D-02

At iterate   20    f=  3.62500D-01    |proj g|=  1.76635D-03

At iterate   25    f=  3.62407D-01    |proj g|=  3.86149D-03

At iterate   30    f=  3.62359D-01    |proj g|=  1.27176D-03

At iterate   35    f=  3.62347D-01    |proj g|=  8.71139D-04

At iterate   40    f=  3.62346D-01    |proj g|=  3.71540D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *
```

```
   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    7     44     59      1     0     0   4.092D-05   3.623D-01
  F =  0.36234568679425577

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```




```
RMSE: 0.22582083286525856
```

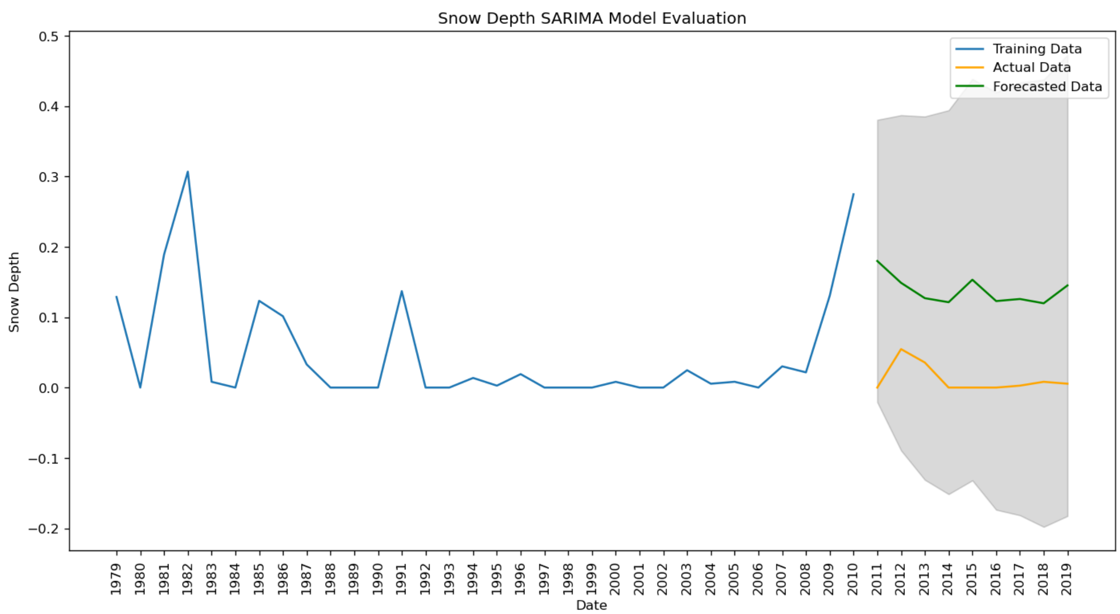
[128]:

```python
# Split the data into train and test
train_size = int(len(averages) * 0.8)
train, test = averages[0:train_size], averages[train_size:len(averages)]

# Fit the SARIMA model on the training dataset
model_train = SARIMAX(train['snow_depth'], order=(1, 1, 1), seasonal_order=(1,␣
 ↪1, 1, 12))
model_train_fit = model_train.fit()

# Forecast on the test dataset
test_forecast = model_train_fit.get_forecast(steps=len(test))
test_forecast_series = pd.Series(test_forecast.predicted_mean, index=test.index)

# Calculate the mean squared error
mse = mean_squared_error(test['snow_depth'], test_forecast_series)
rmse = mse**0.5

# Create a plot to compare the forecast with the actual test data
plt.figure(figsize=(14,7))
```

```python
plt.plot(train['snow_depth'], label='Training Data')
plt.plot(test['snow_depth'], label='Actual Data', color='orange')
plt.plot(test_forecast_series, label='Forecasted Data', color='green')
plt.fill_between(test.index,
                 test_forecast.conf_int().iloc[:, 0],
                 test_forecast.conf_int().iloc[:, 1],
                 color='k', alpha=.15)
plt.title('Snow Depth SARIMA Model Evaluation')
plt.xlabel('Date')
plt.xticks(rotation='vertical')
plt.ylabel('Snow Depth')
plt.legend()
plt.show()

print('RMSE:', rmse)
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few
observations to estimate starting parameters for seasonal ARMA. All parameters
except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f= -4.34994D-01    |proj g|=  4.65547D+00

At iterate    5    f= -4.80322D-01    |proj g|=  4.85228D-02

At iterate   10    f= -4.85501D-01    |proj g|=  1.62895D-01

At iterate   15    f= -5.04503D-01    |proj g|=  6.74680D-02
```

```
At iterate   20    f= -5.04561D-01    |proj g|=  3.95180D-03

At iterate   25    f= -5.04564D-01    |proj g|=  2.54990D-03

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     28     37      1     0     0   6.758D-04  -5.046D-01
  F = -0.50456421413981323

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```




```
RMSE: 0.12910089674446118
```