Experiment 3

TIME SERIES ANALYSIS

B.Tech in Computer Science and Engineering **(CSE)**, **Winter** Semester **2020-21**

Name:	Swaranjana Nayak
Registration Number:	19BCE0977
Slot:	B2
Date:	21/03/2021

Aim:
To perform time series analysis on Climate Change Dataset (Project dataset)

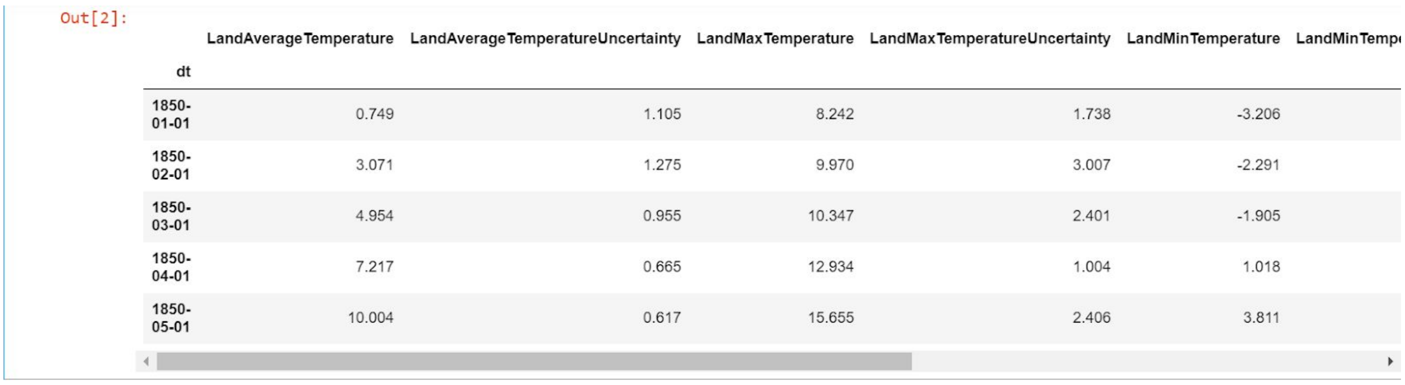
1. **Importing Libraries**

   Code:
```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import copy
%matplotlib inline
```

2. **Reading dataset CSV file and NaN values handling**

   Code:
```
gt = pd.read_csv('GlobalTemperatures.csv', header=0, index_col=0,
parse_dates=True, squeeze=True)
gt.dropna(inplace = True)
gt.head()
```

   Output:



Out[2]:

dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandMinTempe
1850-01-01	0.749	1.105	8.242	1.738	-3.206	
1850-02-01	3.071	1.275	9.970	3.007	-2.291	
1850-03-01	4.954	0.955	10.347	2.401	-1.905	
1850-04-01	7.217	0.665	12.934	1.004	1.018	
1850-05-01	10.004	0.617	15.655	2.406	3.811	

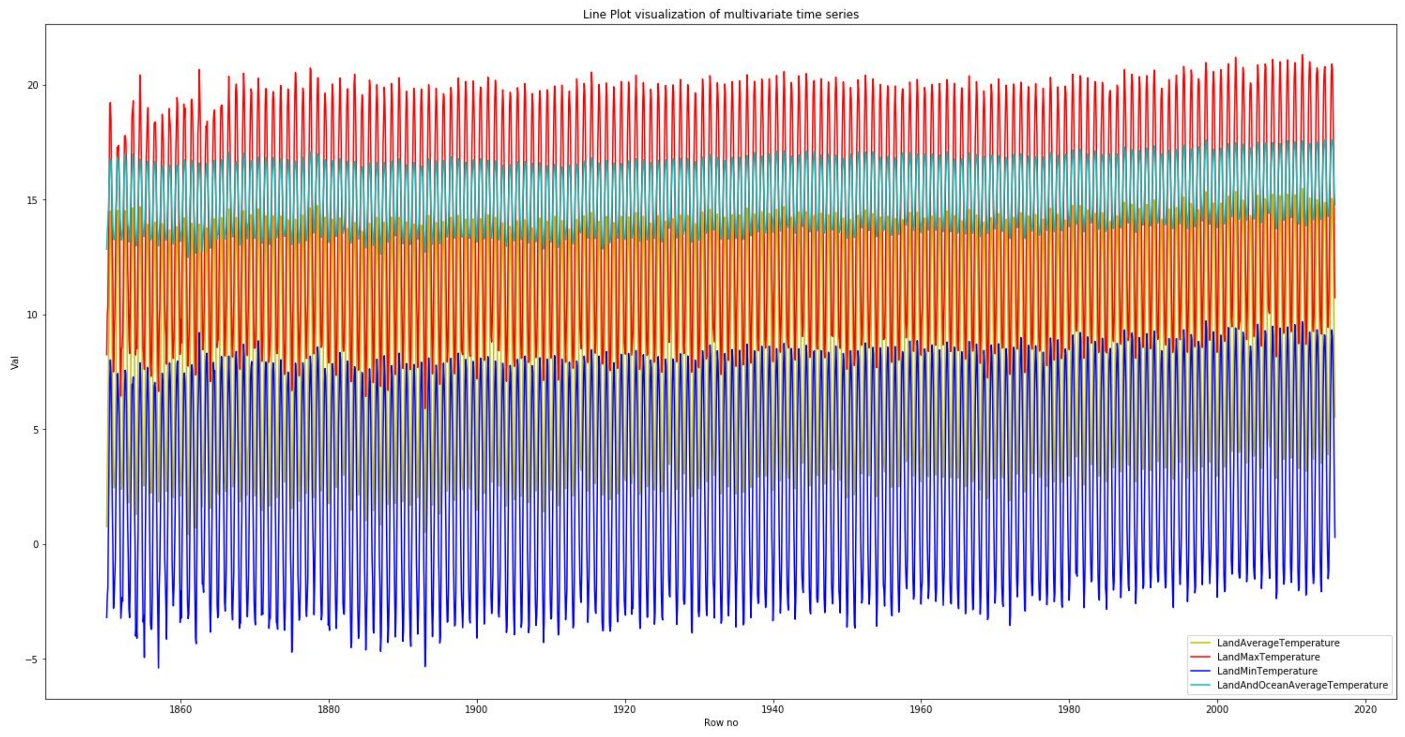
3. **Visualizations**

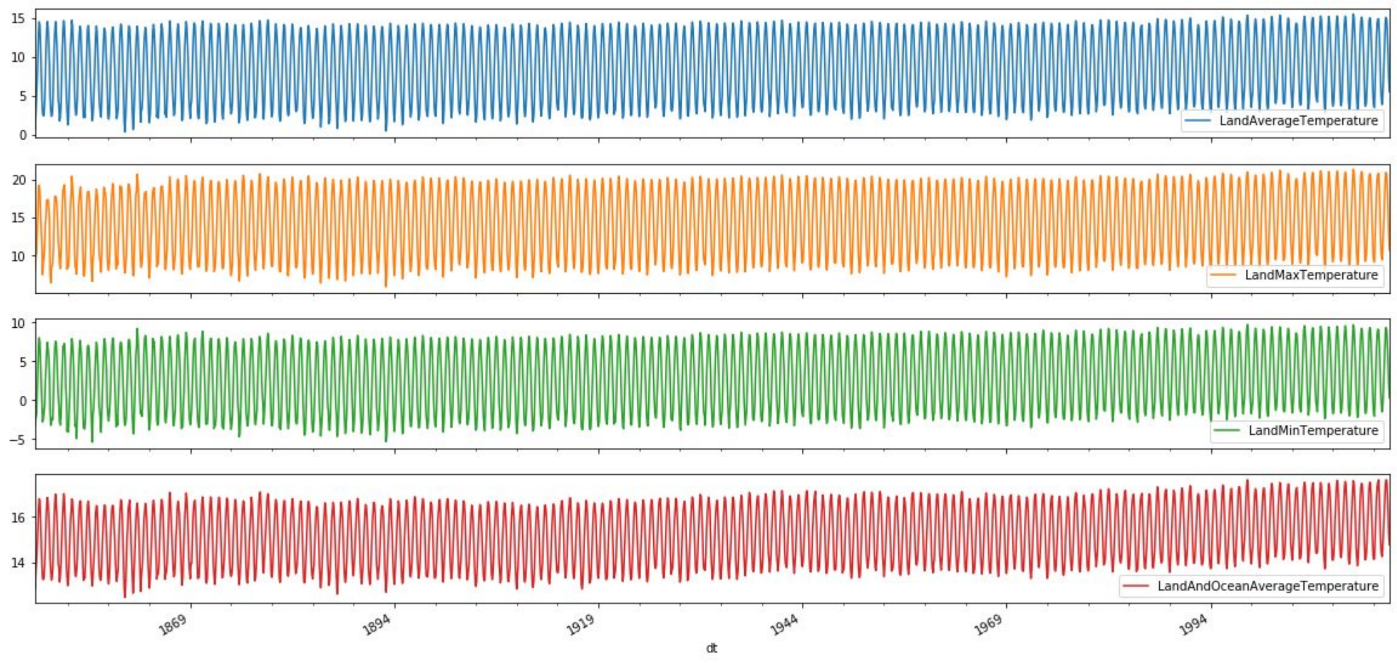
   - **Line Plot of attributes**
     - Not considering uncertainty attributes.
     - The first one has lines on the same plot
     - We can see that max is at the top, min is in the bottom area and the average is in middle
     - The second plot is using subplots because overlapping lines create confusion

Code:

```
            col = [gt.columns[0], gt.columns[2], gt.columns[4],
gt.columns[6]]

            fig = plt.figure(figsize = (20, 10))
            axes = fig.add_axes([0, 0, 1, 1])

            axes.plot(col[0], data = gt, color = 'y')
            axes.plot(col[1], data = gt, color = 'r')
            axes.plot(col[2], data = gt, color = 'b')
            axes.plot(col[3], data = gt, color = 'c')

            axes.set_title('Line Plot visualization of multivariate time
            series')
            axes.set_xlabel('Row no')
            axes.set_ylabel('Val')
            axes.legend()
            fig.savefig('lineplot.png', bbox_inches = 'tight')

            gt[col].plot(subplots=True, figsize=(20, 10))
            plt.savefig('Linesubplots.png', bbox_inches = 'tight')
```

Output:



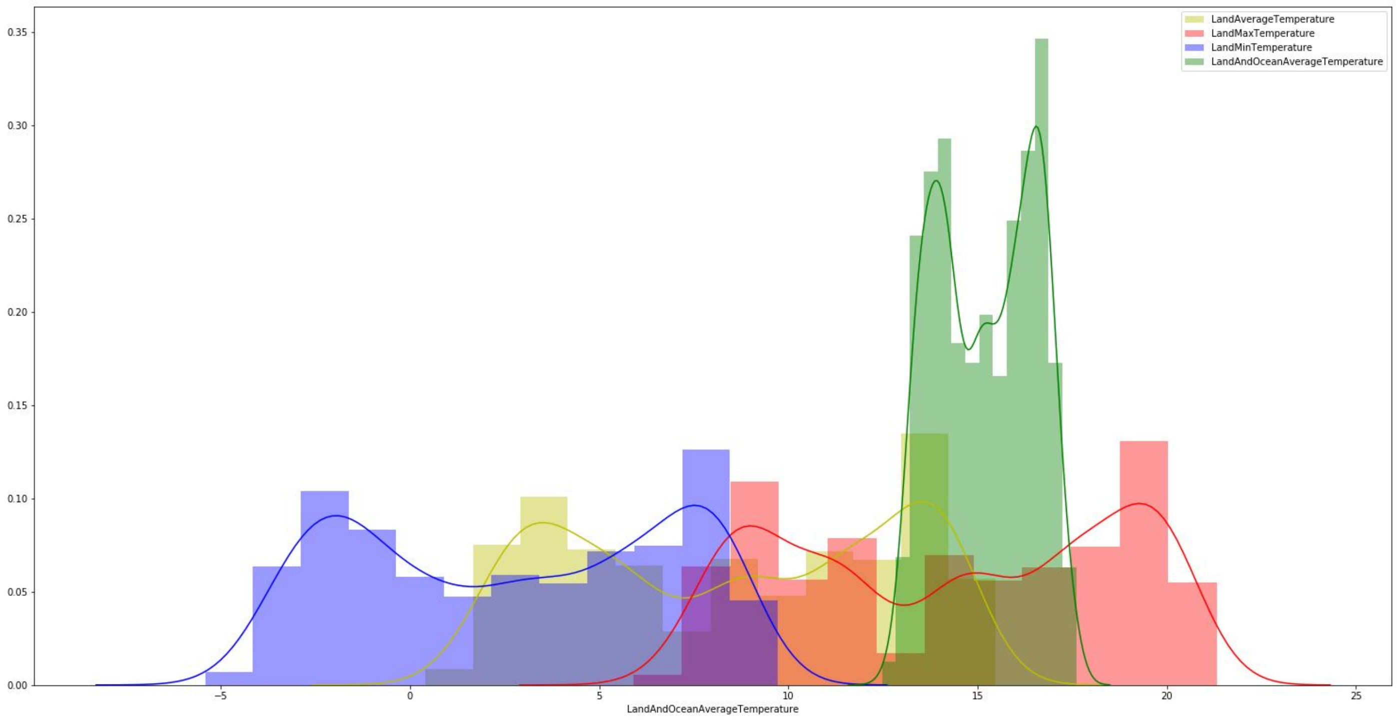


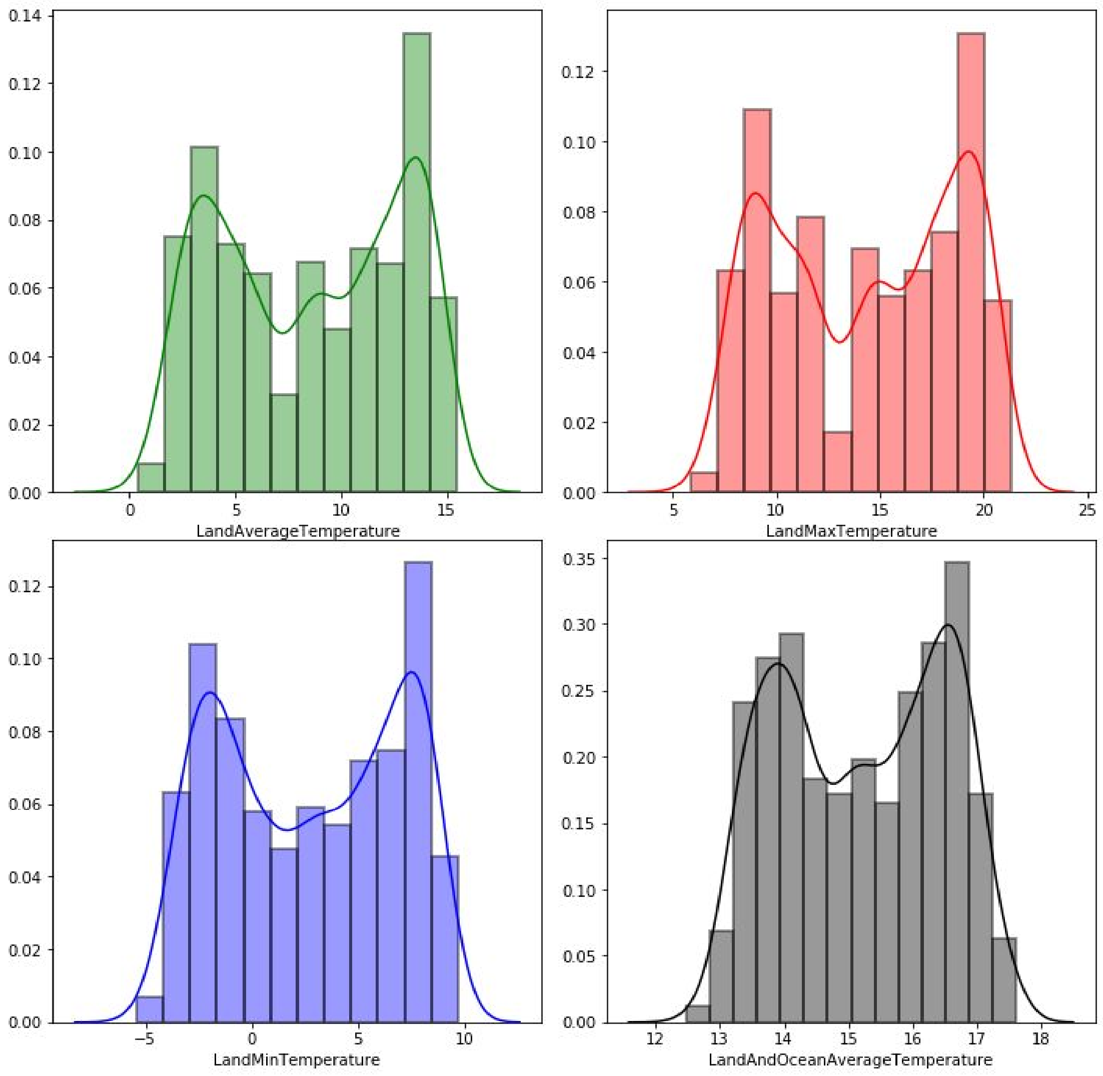
- **<u>Distribution plot</u>**
    - Distribution plot of same 4 attributes considered above
    - The first one has all four distributions in same plot
    - The second one is done using subplots.

<u>Code:</u>

```
fig = plt.figure(figsize = (20, 10))
axes = fig.add_axes([0, 0, 1, 1])
sns.distplot(gt[col[0]], ax = axes, color = 'y')
sns.distplot(gt[col[1]], ax = axes, color = 'r')
sns.distplot(gt[col[2]], ax = axes, color = 'b')
sns.distplot(gt[col[3]], ax = axes, color = 'g')
axes.legend(col)
fig.savefig('distplot.png', bbox_inches = 'tight')

colors = [['g', 'r'], ['b', 'k']]
fig, axes = plt.subplots(nrows = 2, ncols = 2, figsize = (10,
10))
plt.tight_layout()
data = np.reshape(col, (2, 2))

for i in range(2):
    for j in range(2):
        sns.distplot(gt[data[i][j]], ax = axes[i][j],
hist_kws=dict(edgecolor= 'k', linewidth=2), color =
colors[i][j])

fig.savefig('distsubplot.png', bbox_inches = 'tight')
```

<u>Output:</u>


LandAverageTemperature
LandMaxTemperature
LandMinTemperature
LandAndOceanAverageTemperature
LandAndOceanAverageTemperature

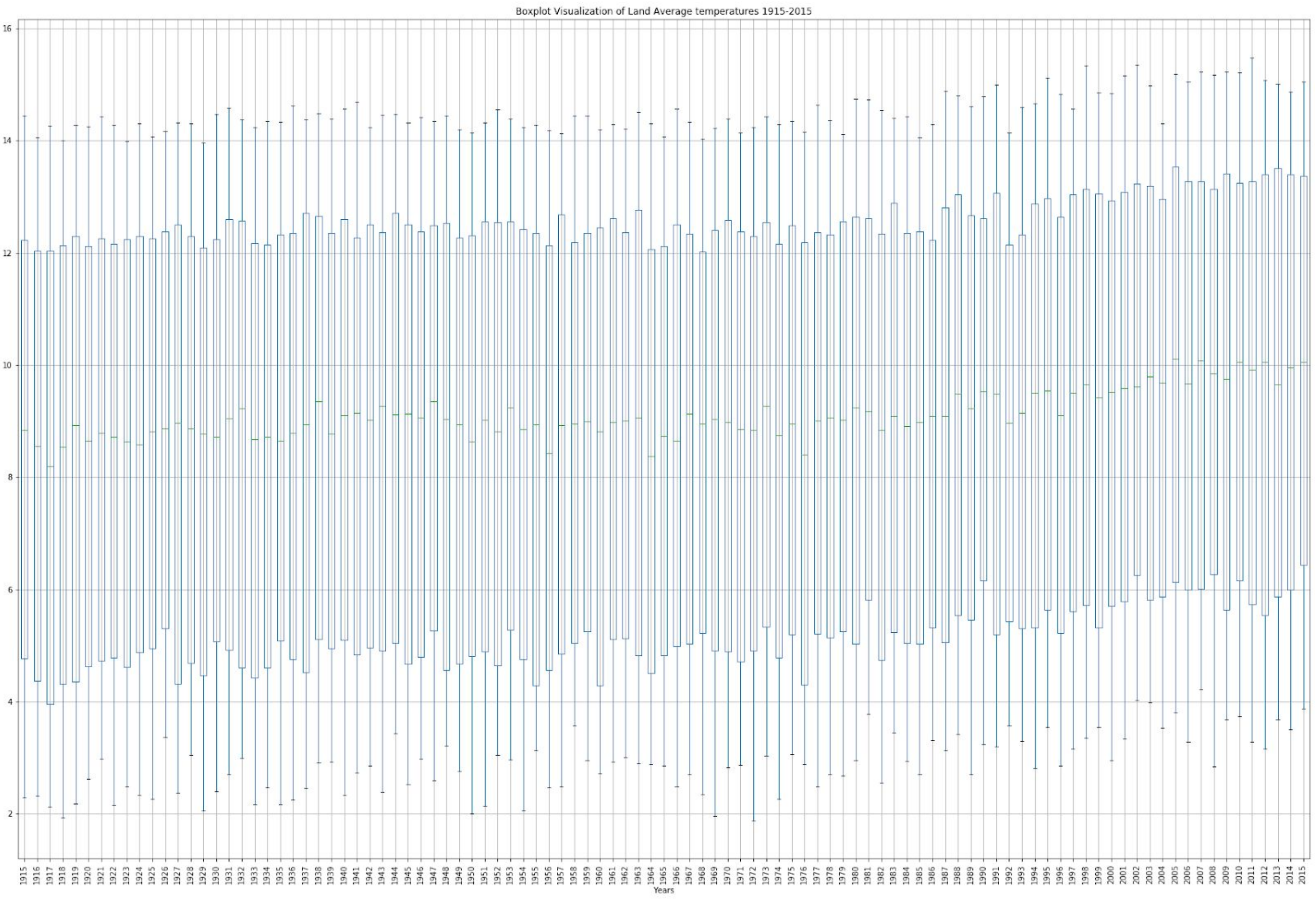

LandAverageTemperature
LandMaxTemperature
LandMinTemperature
LandAndOceanAverageTemperature

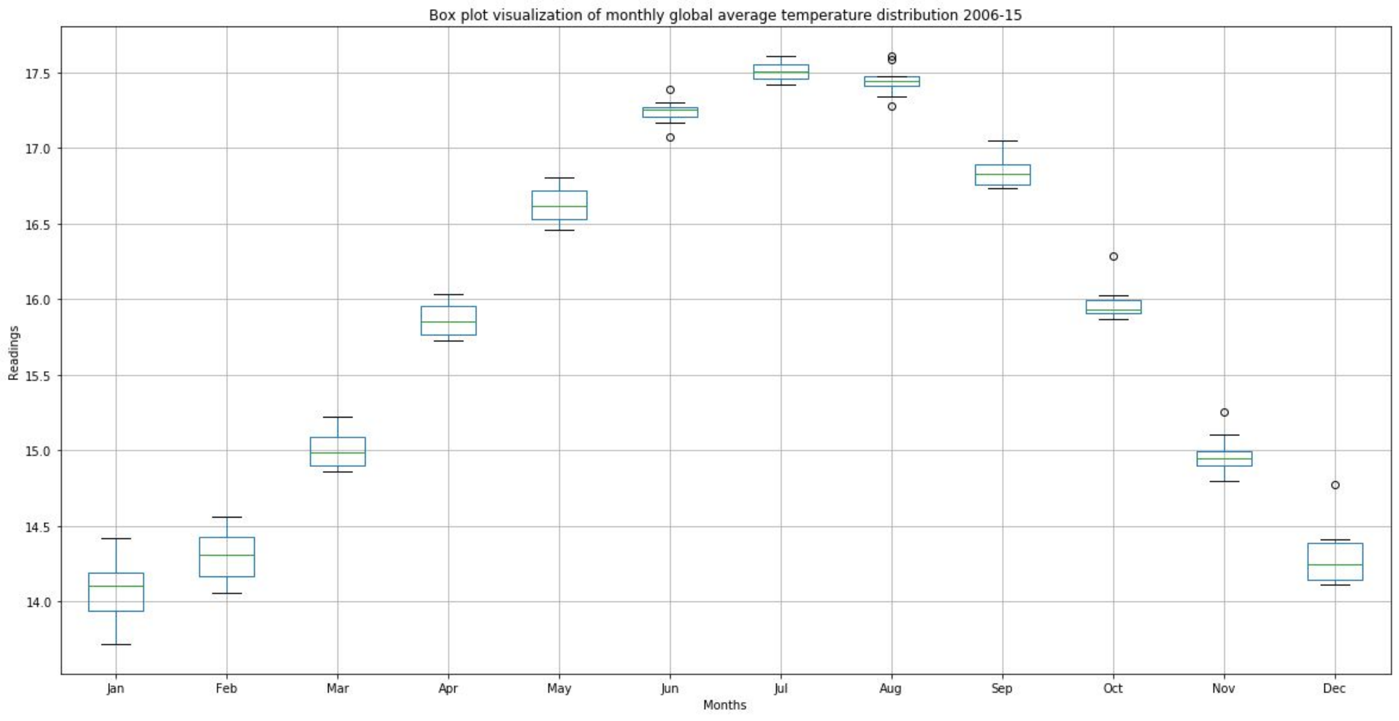
- **Box and Whisper plot**
    - LandAverageTemperature is grouped year-wise and plotted from 1915 - 2015
    - The monthly distribution of LandAndOceanAverageTemperature is checked for 2006-15

Code:

```
groups = gt[col[0]].groupby(pd.Grouper(freq='A'))
LandAverageTemperature = pd.DataFrame()
for name, group in groups:
    LandAverageTemperature[name.year] = group.values

LandAverageTemperature[LandAverageTemperature.columns[65:]].bo
xplot(figsize = (30, 20))
plt.xlabel('Years')
plt.title('Boxplot Visualization of Land Average temperatures
1915-2015')
plt.xticks(rotation = 90)
plt.savefig('boxplot.png', bbox_inches = 'tight')

groups = gt[col[3]].groupby(pd.Grouper(freq='A'))
LandAndOceanAverageTemperature = pd.DataFrame()
for name, group in groups:
    if(name.year > 2005):
        LandAndOceanAverageTemperature[name.year] =
group.values

# LandAndOceanAverageTemperature.columns
LandAndOceanAverageTemperature =
LandAndOceanAverageTemperature.transpose()
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
LandAndOceanAverageTemperature.columns = months

LandAndOceanAverageTemperature.boxplot(figsize = (20, 10))
plt.xlabel('Months')
plt.ylabel('Readings')
plt.title('Box plot visualization of monthly global average
temperature distribution 2006-15')
plt.savefig('monthlyboxplot.png', bbox_inches = 'tight')
```

Output:


Boxplot Visualization of Land Average temperatures 1915-2015
Years

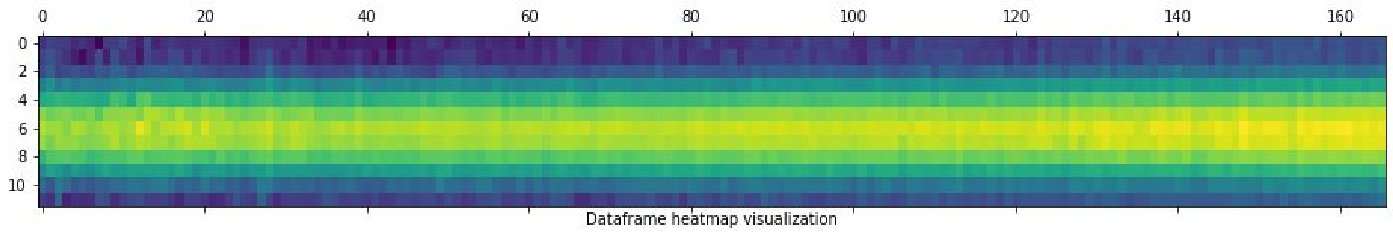

Box plot visualization of monthly global average temperature distribution 2006-15
Readings
Months
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec

- **Heatmap visualization**
  - A matrix of numbers can be plotted as a surface, where the values in each cell of the matrix are assigned a unique color.
  - This is called a heatmap, as larger values can be drawn with warmer colors (yellows and reds) and smaller values can be drawn with cooler colors (blues and greens).
  - LandMinTemperature heatmap visualization.
  - Years v/s months
  - 166 years v/s 12 months
  - We can see that the minimum temperatures are higher in the middle of the year

Code:

```
groups = gt[col[2]].groupby(pd.Grouper(freq='A'))
LandMinTemperature = pd.DataFrame()
for name, group in groups:
    LandMinTemperature[name.year] = group.values
# years = years.T
plt.matshow(LandMinTemperature,          interpolation=None,
aspect='auto')
plt.xlabel('Dataframe heatmap visualization')
plt.savefig('matviz.png', bbox_inches = 'tight')
```

Output:

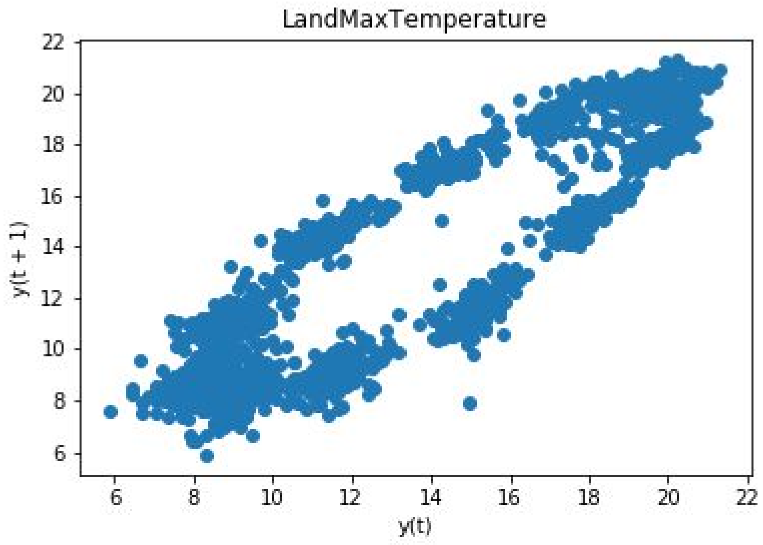


- **Lagplot**
  - Time series modeling assumes a relationship between an observation and the previous observation.
  - Previous observations in a time series are called lags, with the observation at the previous time step called lag1, the observation at two-time steps ago lag2, and so on.
  - A useful type of plot to explore the relationship between each observation and a lag of that observation is called the scatter plot.
  - Pandas has a built-in function for exactly this called the lag plot. It plots the observation at time t on the x-axis and the lag1 observation (t-1) on the y-axis.
  - If the points cluster along a diagonal line from the bottom-left to the top-right of the plot, it suggests a positive correlation relationship.
  - If the points cluster along a diagonal line from the top-left to the bottom-right, it suggests a negative correlation relationship.
  - Either relationship is good as they can be modeled.
  - More points tighter into the diagonal line suggests a stronger relationship and more spread from the line suggests a weaker relationship.
  - A ball in the middle of a spread across the plot suggests a weak or no relationship.

Code:

```
pd.plotting.lag_plot(gt[col[1]])
plt.title('LandMaxTemperature')
plt.savefig('lagplot.png')
```

Output:

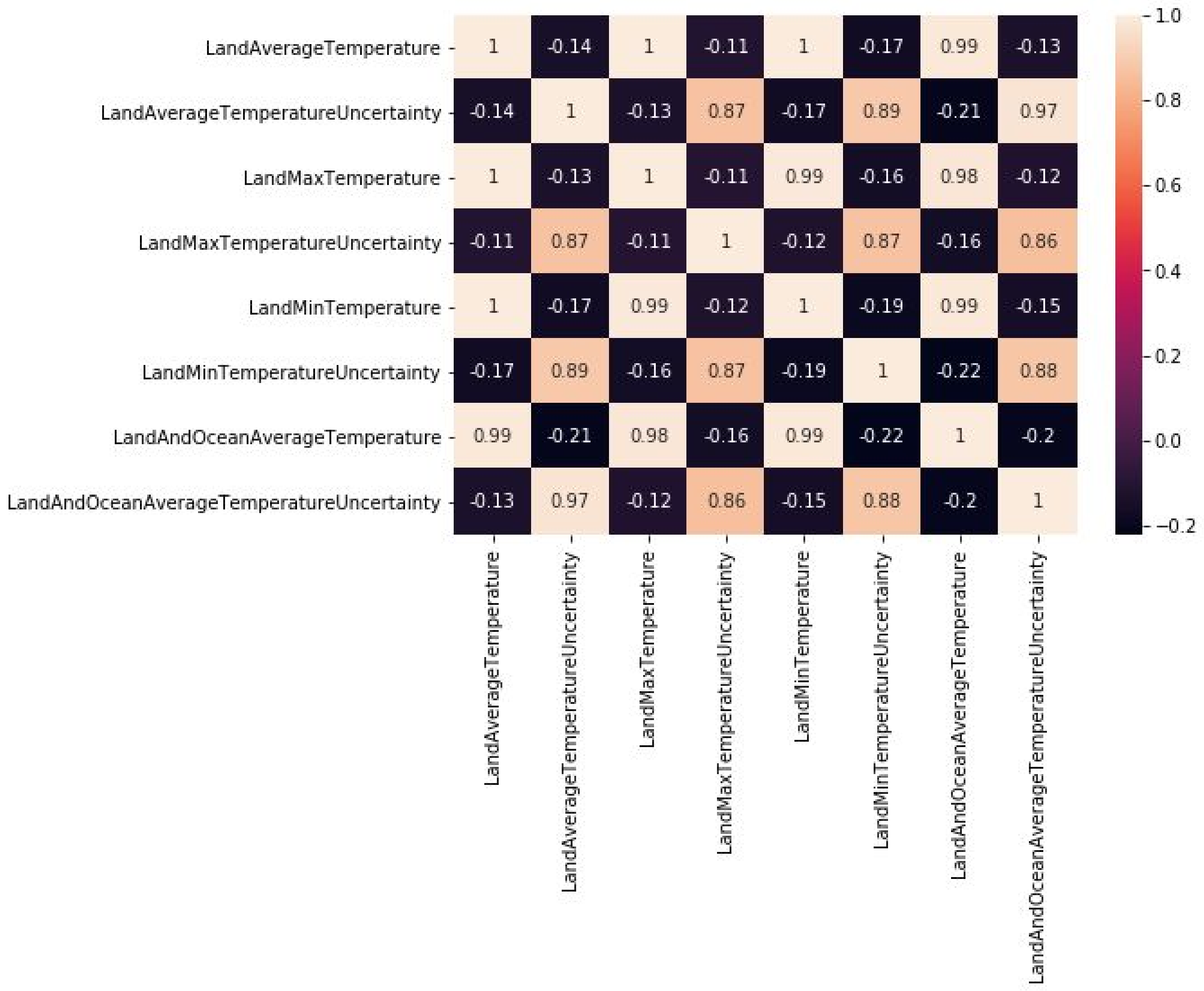


- **Correlation Heatmap**
  - Correlation matrices are an essential tool of exploratory data analysis.
  - Correlation heatmaps contain the same information in a visually appealing way.
  - They show in a glance which variables are correlated, to what degree, in which direction, and alerts us to potential multicollinearity problems.
  - Here we can see that the four attributes we considered earlier have a correlation of almost 1 with each other.
  - That is a very high correlation.

Code:

```
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
sns.heatmap(gt.corr(), annot=True)
fig.savefig('correlation_heatmap.png', bbox_inches = 'tight')
```

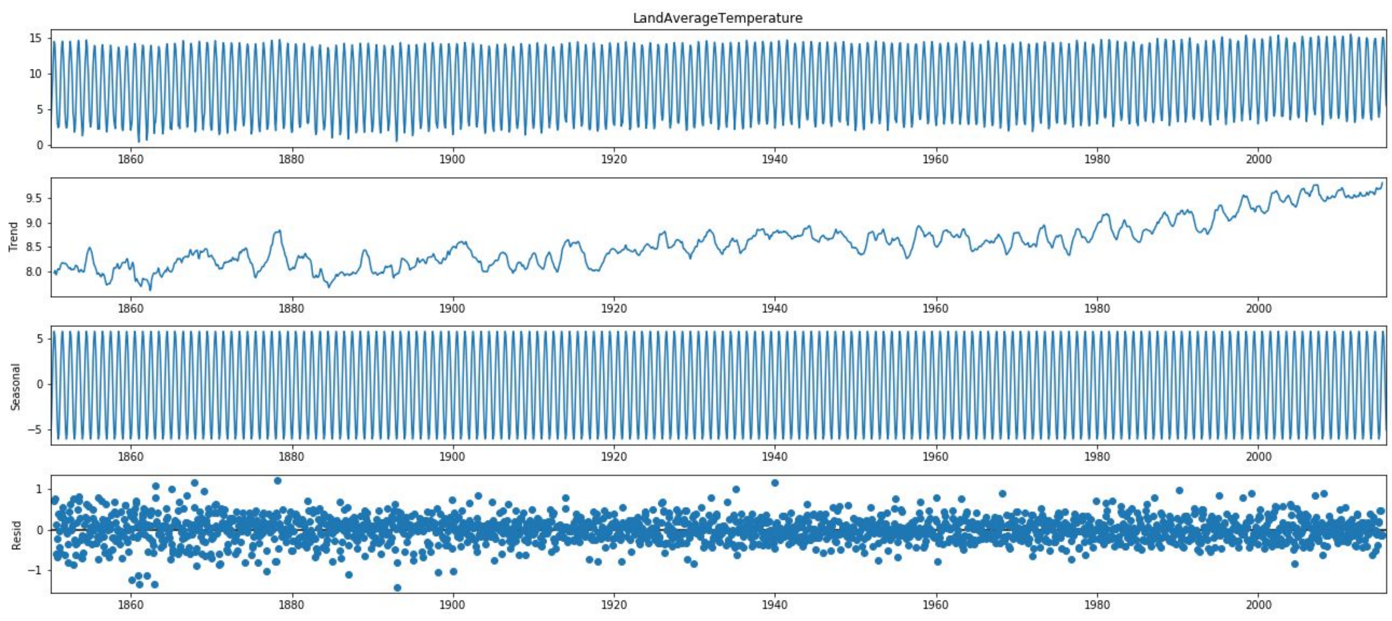
Output:



- **Visualization after decomposing the matrix into trend, seasonality, and noise**

Code:
```
y = gt[col[0]]
import statsmodels.api as sm
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
# tsa = time series analysis
fig = decomposition.plot()
fig.savefig('decomposition.png')
```

Output:

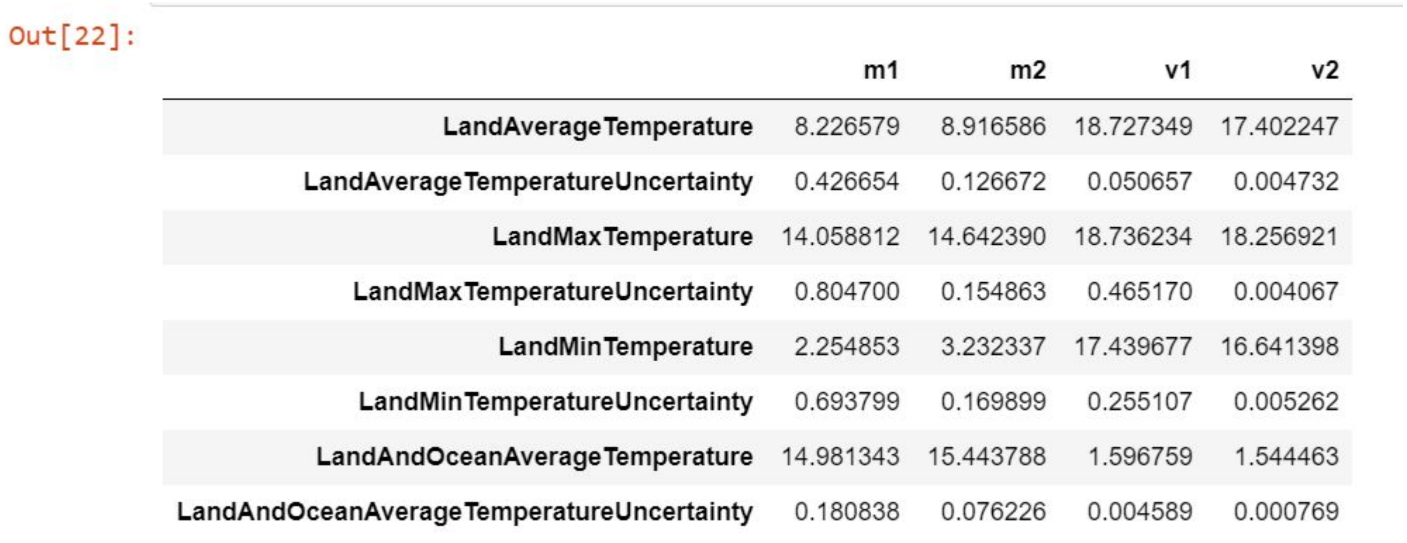


4. **<u>Checking for stationarity</u>**
    - A stationary series is one in which the properties – mean, variance, and covariance, do not vary with time.
    - For a series to be classified as stationary, it should not exhibit a trend.
    - It can be tested visually or using statistical tests
    - **ADF (Augmented Dickey Fuller) Test**
        - The Dickey Fuller test is one of the most popular statistical tests. It can be used to determine the presence of unit root in the series, and hence help us understand if the series is stationary or not. The null and alternate hypothesis of this test are:
            1. Null Hypothesis: The series has a unit root (value of a =1)
            2. Alternate Hypothesis: The series has no unit root.
        - If we fail to reject the null hypothesis, we can say that the series is non-stationary. This means that the series can be linear or difference stationary (we will understand more about difference stationery in the next section).
        - Test for stationarity: If the test statistic is less than the critical value, we can reject the null hypothesis (aka the series is stationary). When the test statistic is greater than the critical value, we fail to reject the null hypothesis (which means the series is not stationary).
    - **KPSS (Kwiatkowski-Phillips-Schmidt-Shin) Test**
        - KPSS is another test for checking the stationarity of a time series (slightly less popular than the Dickey Fuller test). The null and alternate hypotheses for the KPSS test are opposite that of the ADF test, which often creates confusion.
        - The authors of the KPSS test have defined the null hypothesis as the process is trend stationary, to an alternate hypothesis of a unit root series. We will understand the trend stationarity in detail in the next section. For now, let's focus on the implementation and see the results of the KPSS test.
            1. Null Hypothesis: The process is trend stationary.
            2. Alternate Hypothesis: The series has a unit root (series is not stationary).
        - Test for stationarity: If the test statistic is greater than the critical value, we reject the null hypothesis (series is not stationary). If the test statistic is less than the critical

value, it fails to reject the null hypothesis (series is stationary). For the air passenger data, the value of the test statistic is greater than the critical value at all confidence intervals, and hence we can say that the series is not stationary.

Code:
```
df1, df2 = gt[0:996], gt[996:]
m1, m2 = df1.mean(), df2.mean()
v1, v2 = df1.var(), df2.var()
mv = pd.DataFrame([m1, m2, v1, v2])
mv = mv.T
mv.columns = ['m1', 'm2', 'v1', 'v2']
mv
```

Output:



Out[22]:

	m1	m2	v1	v2
LandAverageTemperature	8.226579	8.916586	18.727349	17.402247
LandAverageTemperatureUncertainty	0.426654	0.126672	0.050657	0.004732
LandMaxTemperature	14.058812	14.642390	18.736234	18.256921
LandMaxTemperatureUncertainty	0.804700	0.154863	0.465170	0.004067
LandMinTemperature	2.254853	3.232337	17.439677	16.641398
LandMinTemperatureUncertainty	0.693799	0.169899	0.255107	0.005262
LandAndOceanAverageTemperature	14.981343	15.443788	1.596759	1.544463
LandAndOceanAverageTemperatureUncertainty	0.180838	0.076226	0.004589	0.000769

Code:
```
# Null Hypothesis (H0): If failed to be rejected, it suggests
the time series has a unit root,
#     meaning it is non-stationary. It has some time-dependent
structure.
# Alternate Hypothesis (H1): The null hypothesis is rejected;
it suggests the time series does not have a unit root,
#     meaning it is stationary. It does not have a
time-dependent structure.
# p-value > 0.05: Fail to reject the null hypothesis (H0), the
data has a unit root and is non-stationary.
# p-value <= 0.05: Reject the null hypothesis (H0), the data
does not have a unit root and is stationary.

from statsmodels.tsa.stattools import adfuller
X = gt[col[0]].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

# time series is non-stationary
```

Output:

```
ADF Statistic: -1.455328
p-value: 0.555483
Critical Values:
     1%: -3.434
     5%: -2.863
     10%: -2.568
```

Code:

```
# Null Hypothesis: The process is trend stationary.
# Alternate Hypothesis: The series has a unit root (series is not stationary).
# Test for stationarity: If the test statistic is greater than the critical value,
#         we reject the null hypothesis (series is not stationary). If the test statistic is
#     less than the critical value it fails to reject the null hypothesis (series is stationary).

from statsmodels.tsa.stattools import kpss
result = kpss(X)
print('KPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[3].items():
    print('\t%s: %.3f' % (key, value))

# series is non-stationary
```

Output:

```
KPSS Statistic: 3.207645
p-value: 0.010000
Critical Values:
     10%: 0.347
     5%: 0.463
     2.5%: 0.574
     1%: 0.739
```

5. ARIMA model fitting

Code:

```
import itertools

p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in
list(itertools.product(p, d, q))]

for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y, order=param,
seasonal_order=param_seasonal, enforce_stationarity=False,
enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal,
results.aic))
        except:
            continue
```

Output:

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:14648.232239910616
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:12071.555258261513
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:2711.969242167406
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:1718.884346276585
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:2726.5248355473327
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1710.9350330084148
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:2159.4236375924247
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:1718.1731216864578
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:12086.072037326963
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:9796.850612655458
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:2498.3039225464736
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:1455.4123293790185
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:2517.9985894053343
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:1449.2470058287436
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:1923.5460531601395
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:1456.2365410271439
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:8834.845000426783
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:6933.155112353311
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:3224.2980128173585
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:1967.3596825925506
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:3211.7060867559753
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:1983.0594633048736
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:2589.399610444788
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:1963.6957235339676
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:7263.481479760837
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:6178.990791909218
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:2637.3231386220014
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:1360.7066601443917
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:2653.5551180265306
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:1364.5075694168652
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:1985.19149939806
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:1356.7767595785617
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:8814.437284610009
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:6905.517188648168
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:2458.1443668298634
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:1363.1300227453323
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:2459.1144852815687
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:1370.4570786904717
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:1857.1875945636307
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:1359.3074046157933
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:7233.540266174019
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:6141.041212666039
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:2440.976568803814
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:1299.058838125167
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:2452.3938331516156
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:1317.6917146593255
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:1834.6532058786756
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:1292.6119509187179
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:6581.372948003664
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:5653.659601689609
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:2935.766543360469
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:1668.7586816669768
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:2921.4367207364567
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:1683.1759106901304
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:2292.620305246856
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:1660.8501214256962
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:6197.001674349018
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:5582.759681225403
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:2457.4655600778133
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:1259.8893719781759
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:2457.7702593776594
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:1267.7019778536528
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:1862.990292265445
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:1258.1375234236712
```

This value for the parameter and seasonal parameter gives the least (optimal) value of AIC.

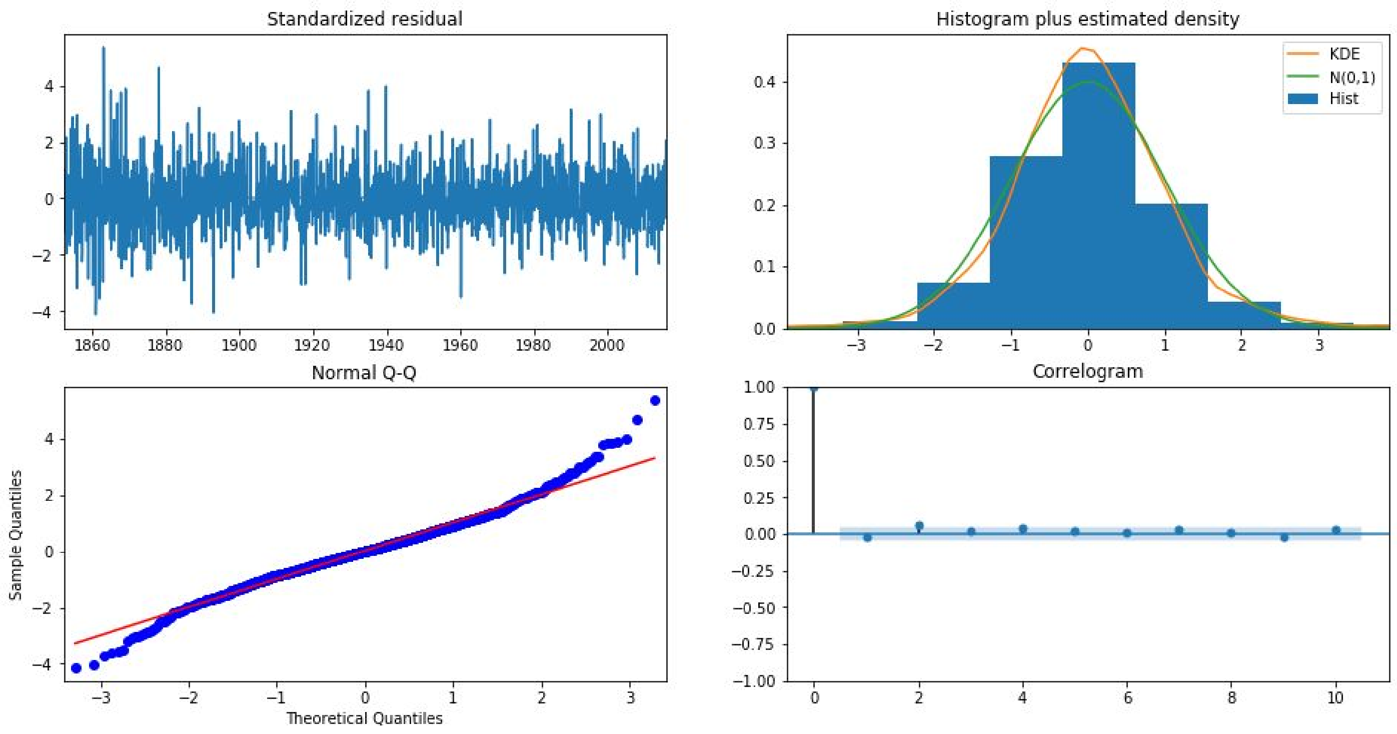
Code:

```
mod = sm.tsa.statespace.SARIMAX(y,order=(1, 1, 1), seasonal_order=(1, 1,
1, 12), enforce_stationarity=True, enforce_invertibility=False)
results = mod.fit()
```

```
print(results.summary().tables[1])
results.plot_diagnostics(figsize=(16, 8))
plt.savefig('arima-stationtrue.png', bbox_inches = 'tight')
```

Output:

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3231      0.020     16.347      0.000       0.284       0.362
ma.L1         -0.9534      0.008   -116.561      0.000      -0.969      -0.937
ar.S.L12       0.0023      0.019      0.117      0.907      -0.036       0.040
ma.S.L12      -0.9568      0.008   -118.114      0.000      -0.973      -0.941
sigma2         0.1106      0.003     42.930      0.000       0.106       0.116
==============================================================================
```

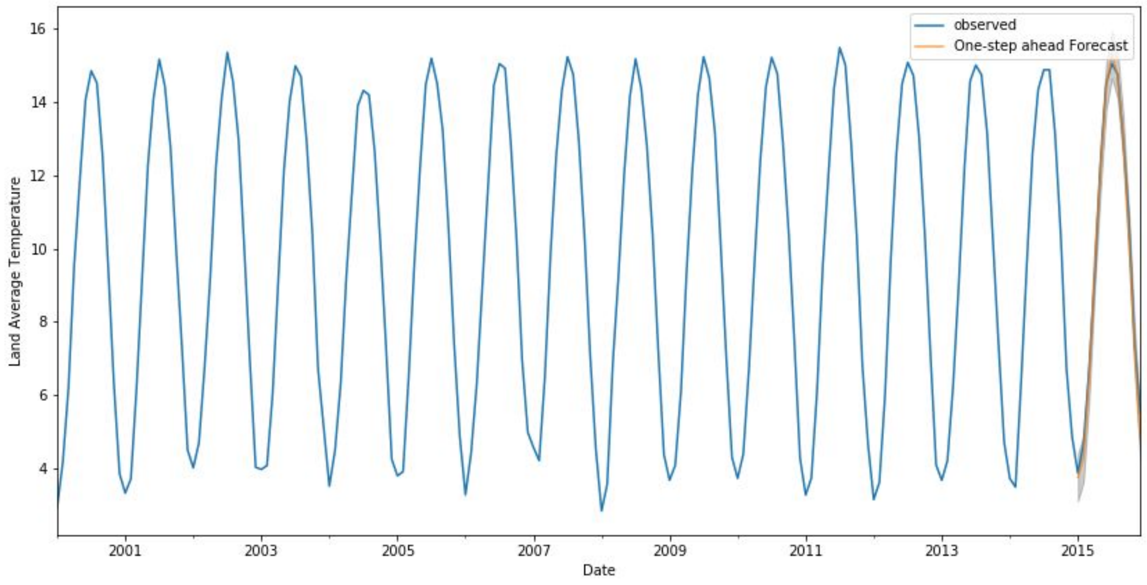


Forecasting using the model:

Code:

```
pred = results.get_prediction(start=pd.to_datetime('2015-01-01'),
dynamic=False)
pred_ci = pred.conf_int()
ax = y['2000':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast',
alpha=.7, figsize=(14, 7))
ax.fill_between(pred_ci.index, pred_ci.iloc[:, 0], pred_ci.iloc[:, 1],
color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Land Average Temperature')
plt.legend()

plt.savefig('forecast.png')
```

Output



Error Calculation:

Code:

```
y_forecasted = pred.predicted_mean
y_truth = y['2015-01-01':]
mse = ((y_forecasted - y_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(mse,
2)))
```

Output:

```
The Mean Squared Error of our forecasts is 0.09
```

Which is an excellent prediction!