Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2 - 2nd Semester
ACTIVITY NO.	**Assignment 10.2 : ARIMA MODEL**
Name	Dela Cruz, Irish
Section	CPE32S3
Date Performed:	04/29/2024
Date Submitted:	05/04/2024
Instructor:	Engr. Roman M. Richard

INSTRUCTION

1. Load time series data: data.csv
2. Visualize the time series
3. Fit an ARIMA Model (baseline model order = (1,1,1))
4. Improve the ARIMA Model
5. Print the model summary
6. Make a forecast (steps=10)
7. Plot the forecast
8. Perform a grid search

Supplementary Activity: do the same for this dataset - dataset_temperature.csv

Importing libraries and Dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from datetime import datetime
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import sys
import statsmodels as s
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import pyplot
from datetime import datetime
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.ticker as ticker
import warnings
%matplotlib inline
```

T1. Load time series data: data.csv

```
data = pd.read_csv('/content/drive/MyDrive/Assignment 10.2/data (1).c
```

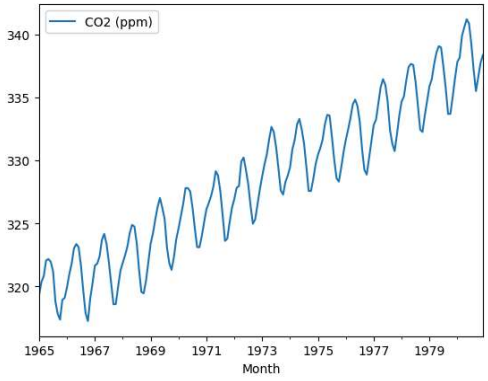
T2. Visualized the time-series

Dataset: data.csv

```
def parser(x):
    return datetime.strptime(x, '%Y-%m')

series = pd.read_csv('/content/drive/MyDrive/Assignment 10.2/data (1)
print(series.head())
series.plot()
plt.show()
```

```
<ipython-input-5-7d0b5dbc78c1>:4: Future
  series = pd.read_csv('/content/drive/M
            CO2 (ppm)
Month
1965-01-01     319.32
1965-02-01     320.36
1965-03-01     320.82
1965-04-01     322.06
1965-05-01     322.17
```



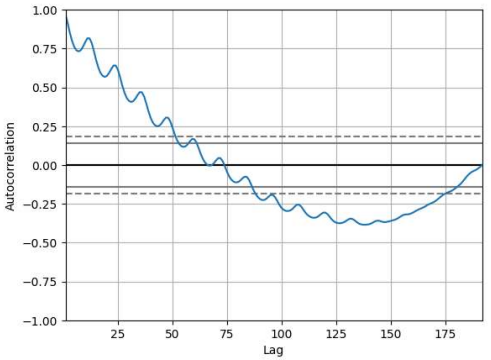
Remarks for graph of time-series

The graph shows that there are big changes happening roughly every two years. For example, on February 1, 1965, the total value was 320.36, and on March 1, 1965, it was 320.82, which is a difference of 0.46 in just one month. This pattern goes up and down like a zigzag every couple of years.

```
def parser(x):
    return datetime.strptime(x, '%Y-%m')

series = pd.read_csv('/content/drive/MyDrive/Assignment 10.2/data (1)
autocorrelation_plot(series)
pyplot.show()
```

```
<ipython-input-6-f65568a7a288>:4: Future
  series = pd.read_csv('/content/drive/M
```



Remarks for Autocorrelation

The graph illustrates an increasing lag, followed by a decrease in autocorrelation values, indicating a weakening relationship between data points and their past values.

T3. Fit an ARIMA Model (baseline model order = (1,1,1))

```
model = ARIMA(series, order=(1,1,1))
model_fit = model.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
```

T4. Improve the ARIMA Model

```
model = ARIMA(series, order=(7,7,7))
model_fit = model.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespa
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py
  warnings.warn("Maximum Likelihood optimization failed to "
```

T5. Print the model summary

Model Summary

```
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              CO2 (ppm)   No. Observations:
Model:                 ARIMA(7, 7, 7)   Log Likelihood
Date:                Fri, 03 May 2024   AIC
Time:                        12:37:15   BIC
Sample:                    01-01-1965   HQIC
                         - 12-01-1980
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.0
------------------------------------------------------------------------------
ar.L1         -3.1410      0.096    -32.680      0.000      -3.3
ar.L2         -5.3488      0.320    -16.705      0.000      -5.9
ar.L3         -6.0937      0.562    -10.839      0.000      -7.1
ar.L4         -5.1329      0.642     -7.999      0.000      -6.3
ar.L5         -3.2232      0.520     -6.202      0.000      -4.2
ar.L6         -1.5104      0.285     -5.294      0.000      -2.0
ar.L7         -0.3716      0.090     -4.136      0.000      -0.5
ma.L1         -0.5574      0.479     -1.164      0.245      -1.4
ma.L2         -0.9607      0.607     -1.583      0.113      -2.1
ma.L3         -1.0343      1.970     -0.525      0.600      -4.8
ma.L4          0.9858      0.798      1.235      0.217      -0.5
ma.L5          0.9712      1.754      0.554      0.580      -2.4
ma.L6          0.5494      1.057      0.520      0.603      -1.5
ma.L7         -0.9518      1.153     -0.825      0.409      -3.2
sigma2         1.2802      1.542      0.830      0.406      -1.7
==============================================================================
Ljung-Box (L1) (Q):                   2.02   Jarque-Bera (JB):
Prob(Q):                              0.16   Prob(JB):
Heteroskedasticity (H):               1.05   Skew:
Prob(H) (two-sided):                  0.85   Kurtosis:
==============================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of grad
```

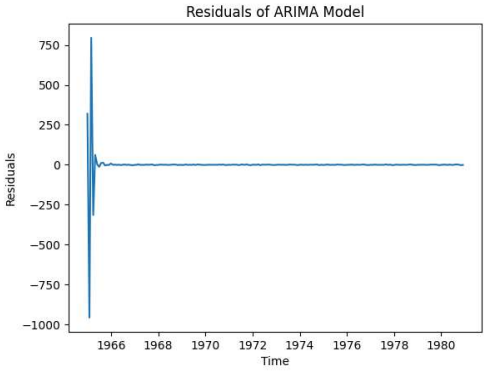
Remarks for the summary table

The SARIMAX results show a fitted ARIMA(7, 7, 7) model for the "CO2 (ppm)" time series data. Key metrics like log likelihood, AIC, and BIC help assess the model's fit and complexity.

Residual of Data

```
residuals = model_fit.resid

plt.plot(residuals)
plt.title('Residuals of ARIMA Model')
plt.xlabel('Time')
plt.ylabel('Residuals')
plt.show()
```



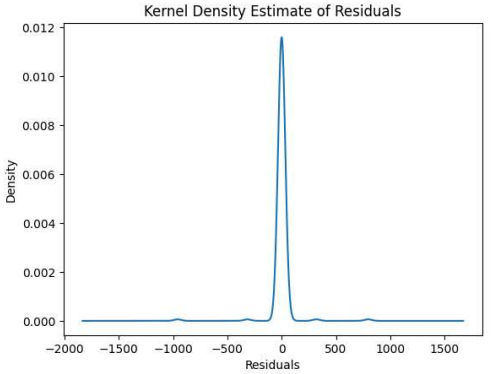
Remarks for Residual Plot

The graph shows that the residuals fluctuate around zero over time. There is no clear pattern or trend in the residuals, suggesting that the ARIMA model is capturing the underlying trend in the data well.

KDE of Residual

```
residuals.plot(kind='kde')
plt.title('Kernel Density Estimate of Residuals')
plt.xlabel('Residuals')
```

```
plt.ylabel('Density')
plt.show()
```



Remarks for KDE Plot

The graph show that the residuals are centered around zero, with most values falling near the middle. This indicates that the ARIMA model is performing well, as the residuals (the difference between predicted and actual values) tend to be small.

```
print(residuals.describe())
```

```
count    192.000000
mean      -0.479389
std       95.817605
min     -956.956233
25%       -0.861888
50%        0.006416
75%        0.710252
max      795.131515
dtype: float64
```

Remarks for describe in terms of residual

The dataset comprises 192 data points, with an average value close to -0.48. Variability around this average is notable, with data points fluctuating by approximately 95.82 units on average.

The dataset's range spans from a minimum of -956.96 to a maximum of 795.13, showcasing significant variability in the observations. Quartile analysis reveals that a quarter of the data falls below -0.86, while the median, situated at 0.0064, divides the dataset into two equal parts.

Moreover, 75% of the observations are below 0.71, indicating a positively skewed distribution.

These statistics provide a clear snapshot of the dataset's central tendency, variability, and distributional characteristics in an easily understandable manner.

T6. Make a forecast (steps=10)

```
X = series.values
size = int(len(X) * 0.90)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()


for t in range(len(test)):
    model = ARIMA(history, order=(9,5,0))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
```

```
<ipython-input-14-f49541edef07>:9: DeprecationWarning: Conversio
  print('predicted=%f, expected=%f' % (yhat, obs))
predicted=339.143181, expected=339.060000
predicted=337.493075, expected=338.950000
```

```
predicted=337.263817, expected=337.410000
predicted=334.312028, expected=335.710000
predicted=333.701353, expected=333.680000
predicted=331.990510, expected=333.690000
predicted=335.397881, expected=335.050000
predicted=338.839898, expected=336.530000
predicted=338.490564, expected=337.810000
predicted=339.242521, expected=338.160000
predicted=338.373311, expected=339.880000
predicted=342.874263, expected=340.570000
predicted=340.752164, expected=341.190000
predicted=339.407502, expected=340.870000
predicted=338.862437, expected=339.250000
predicted=336.085963, expected=337.190000
predicted=334.561856, expected=335.490000
predicted=334.747088, expected=336.630000
predicted=340.346779, expected=337.740000
predicted=340.631944, expected=338.360000
```

Remarks for forecast (10 steps)

These lines show predicted and expected values from a forecasting model. For each prediction, the model forecasts a value (predicted) and compares it with the actual observed value (expected).

Based on the results there's a small gap betwen the predicted values and expected values which suggest that the model was actually predicted the actual values from the dataset.

```
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)

print("MSE: ", mse)
print("RMSE: ", rmse)
```

```
MSE:  2.08644581110167
RMSE:  1.444453464498483
```

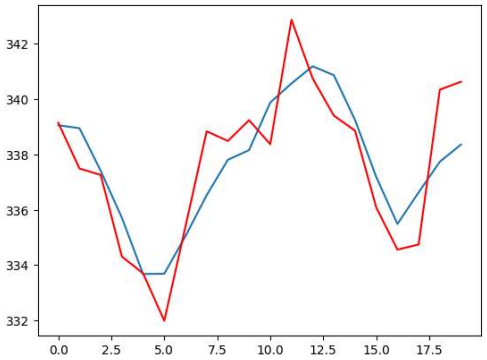
Remarks for evaluation of MSE and RMSE

the MSE is approximately 2.09, indicating the average squared deviation of predictions from actual values. The RMSE is estimated as 1.44 typically

predition error, which shows the predictions deviate from actual values by around 1.44 units. Lower values indicate better model performance.

T7. Plot the forecaast

```
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```



Remarks for forecast graph

As you can see above the prediction exceed from the actual values, which means that in the dataset of test values there are times that model fit seen the actual values and redundant values that's why it exceeds.

T8. Perform a grid search

```
def evaluate_arima_model(X, arima_order):
  train_size = int(len(X) * 0.90)
  train, test = X[0:train_size], X[train_size:]
  history = [x for x in train]
  # make predictions
  predictions = list()
  for t in range(len(test)):
    model = ARIMA(history, order=arima_order)
    model_fit = model.fit()
    yhat = model_fit.forecast()[0]
    predictions.append(yhat)
    history.append(test[t])
  # calculate out of sample error
  error = mean_squared_error(test, predictions)
  return error


import warnings
warnings.filterwarnings("ignore")


def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.3f' % (order,mse))
                except:
                    continue
    print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))
```

Grid Search

```
p_values = range(0, 5)
d_values = range(0, 3)
q_values = range(0, 3)
warnings.filterwarnings("ignore")
evaluate_models(series.values, p_values, d_values, q_values)
```

```
ARIMA(0, 0, 0) MSE=99.598
ARIMA(0, 0, 1) MSE=25.664
ARIMA(0, 0, 2) MSE=9.479
ARIMA(0, 1, 0) MSE=1.603
ARIMA(0, 1, 1) MSE=0.975
ARIMA(0, 1, 2) MSE=0.756
ARIMA(0, 2, 0) MSE=1.189
ARIMA(0, 2, 1) MSE=1.164
ARIMA(0, 2, 2) MSE=1.141
ARIMA(1, 0, 0) MSE=1.617
ARIMA(1, 0, 1) MSE=0.986
ARIMA(1, 0, 2) MSE=0.756
ARIMA(1, 1, 0) MSE=0.978
ARIMA(1, 1, 1) MSE=0.899
ARIMA(1, 1, 2) MSE=0.765
ARIMA(1, 2, 0) MSE=1.166
ARIMA(1, 2, 1) MSE=1.166
ARIMA(1, 2, 2) MSE=0.907
ARIMA(2, 0, 0) MSE=1.006
ARIMA(2, 0, 1) MSE=0.920
ARIMA(2, 0, 2) MSE=0.773
ARIMA(2, 1, 0) MSE=0.811
ARIMA(2, 1, 1) MSE=0.633
ARIMA(2, 1, 2) MSE=0.633
ARIMA(2, 2, 0) MSE=1.165
ARIMA(2, 2, 1) MSE=0.836
ARIMA(2, 2, 2) MSE=0.824
ARIMA(3, 0, 0) MSE=0.824
ARIMA(3, 0, 1) MSE=0.639
ARIMA(3, 0, 2) MSE=0.639
ARIMA(3, 1, 0) MSE=0.703
ARIMA(3, 1, 1) MSE=0.632
ARIMA(3, 1, 2) MSE=0.632
ARIMA(3, 2, 0) MSE=0.994
ARIMA(3, 2, 1) MSE=0.713
ARIMA(3, 2, 2) MSE=0.545
ARIMA(4, 0, 0) MSE=0.709
ARIMA(4, 0, 1) MSE=0.639
ARIMA(4, 0, 2) MSE=0.632
ARIMA(4, 1, 0) MSE=0.705
ARIMA(4, 1, 1) MSE=0.660
ARIMA(4, 1, 2) MSE=355.241
ARIMA(4, 2, 0) MSE=0.952
ARIMA(4, 2, 2) MSE=0.845
Best ARIMA(3, 2, 2) MSE=0.545
```

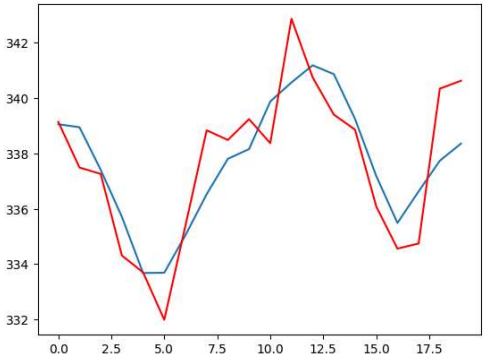
Remarks for grid search

Among the tested models, ARIMA(3, 2, 2) achieved the lowest MSE of 0.545, suggesting it provides the most accurate predictions compared to

the other models tested.

Grid Plot

```
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```



Remarks for grid plot

As you can see above, it same with the forecast plot.

Supplementary

S1. Load timer series: data_temperature.csv

```
data2 = pd.read_csv('/content/drive/MyDrive/Assignment_10_2/dataset_te
```

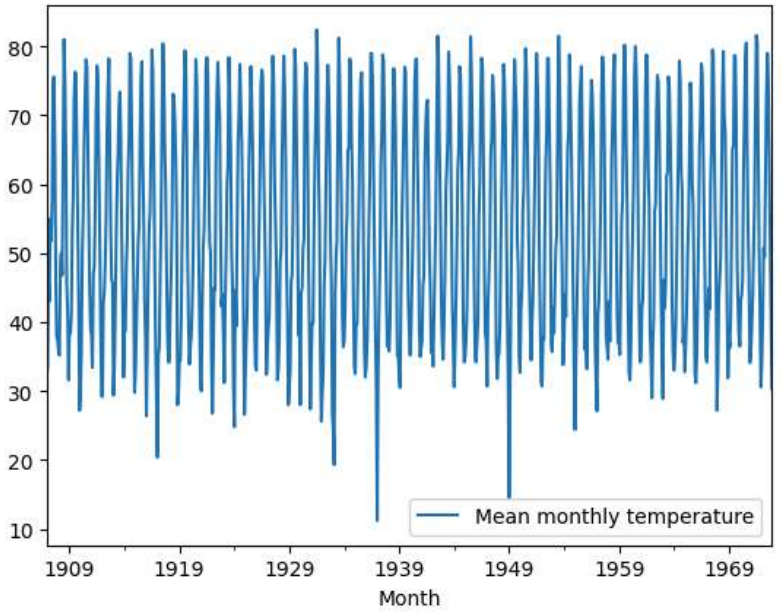
```
data2 = pd.read_csv('/content/drive/MyDrive/Assignment 10.2/dataset_te
```

S2. Visualized the time-series

```
def parser(x):
    if isinstance(x, float):
        return str(int(x))
    return x

series = pd.read_csv('/content/drive/MyDrive/Assignment 10.2/dataset_t
print(series.head())
series.plot()
plt.show()
```

```
<ipython-input-6-bd46179efee8>:6: FutureWarning: The argument 'da
  series = pd.read_csv('/content/drive/MyDrive/Assignment 10.2/da
            Mean monthly temperature
Month
1907-01-01                      33.3
1907-02-01                      46.0
1907-03-01                      43.0
1907-04-01                      55.0
1907-05-01                      51.8
```



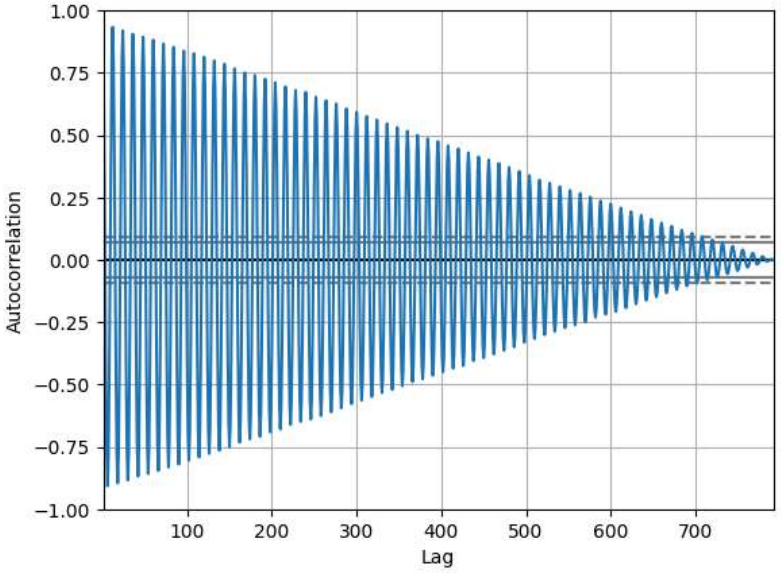
Remarks for graph of time-series

The graph shows that there are big changes happening roughly every ten years. All of the month in the year 1907 has a big gap to each other. This pattern goes up and down like a heartbeat every couple of years.

```
def parser(x):
    return datetime.strptime(x, '%Y-%m')

series = pd.read_csv('/content/drive/MyDrive/Assignment 10.2/dataset_
autocorrelation_plot(series)
pyplot.show()
```

```
<ipython-input-7-61ea8a672b0c>:4: FutureWarning: The argument 'da
  series = pd.read_csv('/content/drive/MyDrive/Assignment 10.2/da
```



The graph illustrates a three pattern:

1. Strong Autocorrelation at Lag 0: This indicates a robust correlation between a data point and itself, which is expected.
2. Decreasing Autocorrelation with Lag: As the lag increases, autocorrelation values decrease, suggesting a weakening relationship between data points and their past values.
3. Negative autocorrelation values suggest an oscillating pattern, where high values are followed by low values, and vice versa, at certain intervals. This hints at a possible cyclical trend in the data.

S3. Fit an ARIMA Model (baseline model order = (1,1,1))

```
model = ARIMA(series, order=(1,1,1))
model_fit = model.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
```

S4. Improve the ARIMA Model

```
model = ARIMA(series, order=(8,8,8))
model_fit = model.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespa
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py
  warnings.warn("Maximum Likelihood optimization failed to "
```

S5. Print the model summary

Model Summary

```
print(model_fit.summary())
```

```
                                SARIMAX Results
==========================================================================
Dep. Variable:     Mean monthly temperature   No. Observations:
Model:                       ARIMA(8, 8, 8)   Log Likelihood
Date:                      Fri, 03 May 2024   AIC
Time:                              14:31:56   BIC
Sample:                        01-01-1907   HQIC
                             - 12-01-1972
Covariance Type:                       opg
==========================================================================
                 coef    std err          z      P>|z|      [0.0
--------------------------------------------------------------------------
ar.L1         -5.2883      0.055    -95.389      0.000      -5.3
ar.L2        -12.7418      0.290    -43.891      0.000     -13.3
ar.L3        -18.4426      0.714    -25.843      0.000     -19.8
ar.L4        -17.8345      1.078    -16.538      0.000     -19.9
ar.L5        -12.0509      1.089    -11.064      0.000     -14.1
ar.L6         -5.6628      0.734     -7.719      0.000      -7.1
ar.L7         -1.7097      0.302     -5.668      0.000      -2.3
ar.L8         -0.2557      0.057     -4.447      0.000      -0.3
ma.L1         -0.2913      0.480     -0.607      0.544      -1.2
ma.L2         -3.2379      0.813     -3.982      0.000      -4.8
ma.L3          0.2376      1.168      0.203      0.839      -2.0
ma.L4          4.5596      2.812      1.622      0.105      -0.9
ma.L5          0.2501      0.956      0.262      0.794      -1.6
ma.L6         -3.2058      3.140     -1.021      0.307      -9.3
ma.L7         -0.2883      0.266     -1.084      0.278      -0.8
ma.L8          0.9769      1.200      0.814      0.415      -1.3
sigma2       220.5013    270.927      0.814      0.416    -310.5
==========================================================================
Ljung-Box (L1) (Q):                   0.45   Jarque-Bera (JB):
Prob(Q):                              0.50   Prob(JB):
Heteroskedasticity (H):               0.96   Skew:
Prob(H) (two-sided):                  0.76   Kurtosis:
==========================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of grad
```

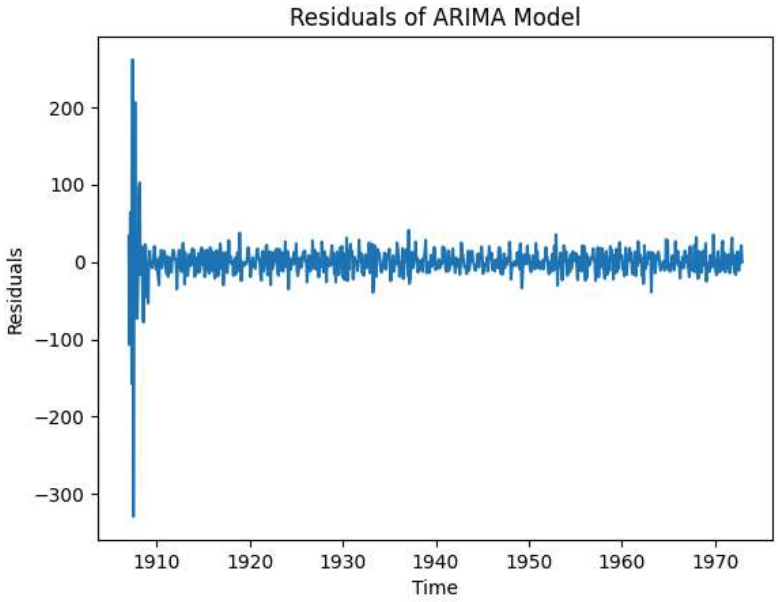
Remarks of Summary Table

The SARIMAX results show a fitted ARIMA(8, 8, 8) model for the "CO2 (ppm)" time series data. Key metrics like log likelihood, AIC, and BIC help assess the model's fit and complexity.

Residual of data_temperature

```
residuals = model_fit.resid

plt.plot(residuals)
plt.title('Residuals of ARIMA Model')
plt.xlabel('Time')
plt.ylabel('Residuals')
plt.show()
```

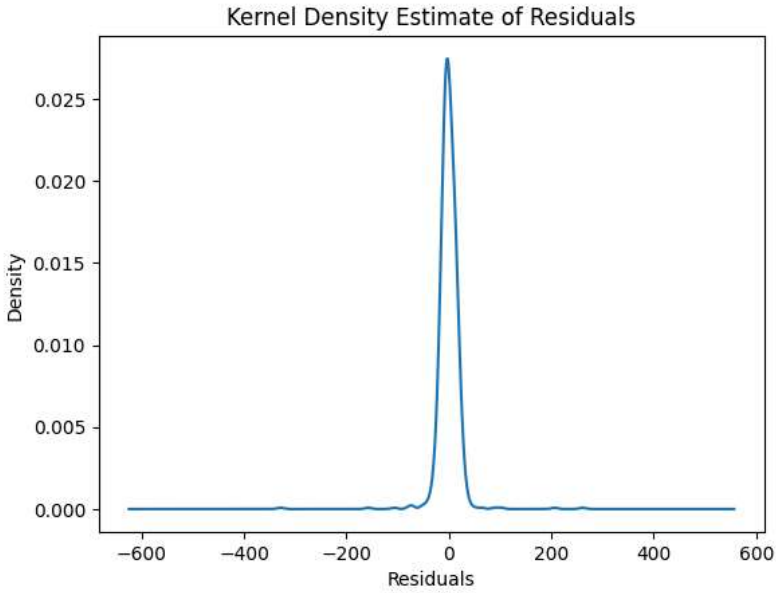


Remarks for Residual Plot

The graph shows that the residuals in the year 1910 fluctuates from 200 to - 300, but as time goes by it fluctuate around zero over time. There is a small zigzag pattern or trend in the residuals, suggesting that the ARIMA model is capturing the underlying trend in the data well.

KDE of Residual

```
residuals.plot(kind='kde')
plt.title('Kernel Density Estimate of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```



Remarks for KDE Plot

The graph show that the residuals are centered around zero, with most values falling near the middle. This indicates that the ARIMA model is

performing well, as the residuals (the difference between predicted and actual values) tend to be small.

Residual Describe

```
print(residuals.describe())
```

```
count    792.000000
mean      -0.213891
std       23.429167
min     -329.277249
25%       -9.028154
50%       -1.003891
75%        9.468113
max      261.452773
dtype: float64
```

Remarks for Residual Describe

The dataset comprises 792 data points, with an average value close to -0.21. Variability around this average is notable, with data points fluctuating by approximately 23.20 units on average.

The dataset's range spans from a minimum of -334.37 to a maximum of 261.57, showcasing significant variability in the observations. Quartile analysis reveals that a quarter of the data falls below -0.86, while the median, situated at 0.0064, divides the dataset into two equal parts.

Moreover, 75% of the observations are high as 9.49, indicating a positively skewed distribution.

These statistics provide a clear snapshot of the dataset's central tendency, variability, and distributional characteristics in an easily understandable manner.

S6. Make a forecast (steps=10)

```
X = series.values
size = int(len(X) * 0.80)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()


for t in range(len(test)):
    model = ARIMA(history, order=(9,2,0))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
```

```
predicted=80.132719, expected=65.000000
predicted=57.189003, expected=53.200000
predicted=43.072484, expected=46.700000
predicted=40.118702, expected=34.100000
predicted=26.488317, expected=35.700000
predicted=35.999429, expected=39.200000
predicted=48.690707, expected=43.500000
predicted=53.245837, expected=50.200000
predicted=59.596229, expected=57.100000
predicted=61.599369, expected=70.700000
predicted=70.923179, expected=81.600000
predicted=83.721171, expected=80.500000
predicted=79.107399, expected=65.800000
predicted=54.815131, expected=51.100000
predicted=40.333659, expected=41.800000
predicted=32.916721, expected=30.600000
predicted=24.277343, expected=34.300000
predicted=36.143931, expected=40.700000
predicted=49.637324, expected=50.700000
predicted=65.973217, expected=49.500000
predicted=58.767994, expected=61.200000
predicted=64.377531, expected=72.100000
```

Remarks for forecast (10 steps)

Based on the results there's a samll gap betwen the predicted values and expected values which suggest that the model was capturing the actual values and predict a correct values.

```
mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)

print("MSE: ", mse)
print("RMSE: ", rmse)
```

```
MSE:  45.25098711563041
RMSE:  6.726885394863689
```

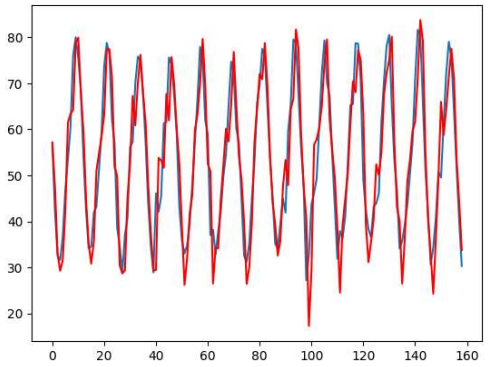
Remarks for MSE AND RMSE

the MSE is approximately 45.25, indicating the average squared deviation of predictions from actual values. The RMSE is estimated as 6.73 typically

predition error, which shows the predictions deviate from actual values by around 38.52 units. Lower values indicate better model performance.

S7. Plot the forecast

```
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```



Remarks for forecast plot

As you can see above, the prediction values is near with the actual values, which means that in the dataset of test values the predicted values and actual values was capturing correctly by the model. In each year there are small sharp spike can be seen, the predicted values is much longer that actual values since there are times that the predicted see it twice.

S8. Perform a grid search

```
def evaluate_arima_model(X, arima_order):
    train_size = int(len(X) * 0.90)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit()
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    # calculate out of sample error
    error = mean_squared_error(test, predictions)
    return error


import warnings
warnings.filterwarnings("ignore")


def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.3f' % (order,mse))
                except:
                    continue
    print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))
```

Grid Search

```
p_values = range(0, 4)
d_values = range(0, 3)
q_values = range(0, 3)
warnings.filterwarnings("ignore")
evaluate_models(series.values, p_values, d_values, q_values)
```

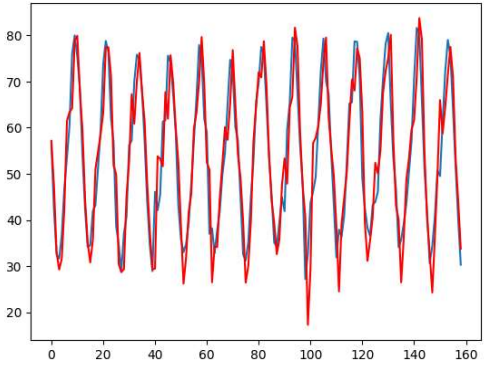
```
ARIMA(0, 0, 0) MSE=248.528
ARIMA(0, 0, 1) MSE=110.508
ARIMA(0, 0, 2) MSE=66.295
ARIMA(0, 1, 0) MSE=90.304
ARIMA(0, 1, 1) MSE=69.790
ARIMA(0, 1, 2) MSE=63.913
ARIMA(0, 2, 0) MSE=80.803
ARIMA(0, 2, 1) MSE=77.875
ARIMA(0, 2, 2) MSE=78.795
ARIMA(1, 0, 0) MSE=83.257
ARIMA(1, 0, 1) MSE=61.729
ARIMA(1, 0, 2) MSE=54.295
ARIMA(1, 1, 0) MSE=63.012
ARIMA(1, 1, 1) MSE=63.158
ARIMA(1, 1, 2) MSE=61.740
ARIMA(1, 2, 0) MSE=77.976
ARIMA(1, 2, 1) MSE=77.981
ARIMA(1, 2, 2) MSE=70.295
ARIMA(2, 0, 0) MSE=42.588
ARIMA(2, 0, 1) MSE=24.695
ARIMA(2, 1, 0) MSE=63.337
ARIMA(2, 1, 1) MSE=62.634
ARIMA(2, 1, 2) MSE=24.709
ARIMA(2, 2, 0) MSE=78.070
ARIMA(2, 2, 1) MSE=63.429
ARIMA(2, 2, 2) MSE=77.095
ARIMA(3, 0, 0) MSE=34.373
ARIMA(3, 0, 1) MSE=24.043
ARIMA(3, 0, 2) MSE=18.080
ARIMA(3, 1, 0) MSE=58.795
ARIMA(3, 1, 1) MSE=34.371
ARIMA(3, 1, 2) MSE=23.739
ARIMA(3, 2, 0) MSE=78.540
ARIMA(3, 2, 1) MSE=78.859
ARIMA(3, 2, 2) MSE=70.196
Best ARIMA(3, 0, 2) MSE=18.080
```

Remarks for grid search

Among the tested models, ARIMA(3, 0, 2) achieved the lowest MSE of 18.080, suggesting it provides the most accurate predictions compared to the other models tested.

Grid Plot

```
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```



Remarks for grid plot

As you can see above, the prediction values is near with the actual values, which means that in the dataset of test values the predicted values and