**Module 10**

**Time Series Analysis and Forecasting**

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3167413?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3167413/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3167412?wrap=1)
* [Solution Files for Codio Activity 10.1](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_1_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.2](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_2_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.3](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_3_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.4](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_4_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.5](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_5_solution.zip)(ZIP)
* [Solution Files for Codio Activity 10.6](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/codio_10_6_solution.zip)(ZIP)
* [Lecture Datasets and Jupyter Notebooks](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_10/video_assets_starter.zip)(ZIP)

**Notes:**

**ARMA**

An acronym that stands for autoregressive integrated moving average; a forecasting model for a stationary time series

**Cycle**

Random, low-frequency variations in time series data

**Decomposition**

A technique to break time series data into trend, cycle, seasonality, and remainder

**Differencing Time Series**

A technique to convert a non-stationary time series into a stationary one

**Non-Stationary Time Series**

A series of data that shows seasonal effects, trends, and other structures related to the time index

**Remainder**

Any data that is not part of the trend, cycle, or seasonality in a time series dataset; also known as ‘residue’

**Seasonality**

Predictable, periodic variations in time series data known to the modeler; also known as periodicity

**Stationary Time Series**

A series of data that has no trend or seasonal effects; stationarity is assumed or required for many statistical models

**Trend**

Long-term behavior of time series data

Trends can cause a change in the mean over time, while seasonality can change the variance over time, both of which define a time series as *non-stationary*. Conversely, we make assumptions about times series datasets that do not have these same trends or seasonality, and they are referred to as *stationary*.

There are two main decomposition methods: Multiplicative and additive decomposition.

Additive decomposition states that time series data results from the sum of its components. Thus,

**Y = T + S + R**

where **Y** is the time-series data, **T** is the trend-cycle component, **S** is the seasonal component, and **R** is the remainder.

Multiplicative decomposition states that time-series data results from the product of its components. Thus,

**Y = T × S × R**

ARMA (sometimes notated as ARIMA) stands for autoregressive integrated moving average.

* **Autoregression**: A model based on observations that are correlated with lagged observations
* **Integrated**: A term that indicates that raw observations have been differentiated to make the time series stationary
* **Moving average**: A model based on the dependence between an observation and a residual error after applying a moving average model to lagged observations

**Augmented Dickey-Fuller test**

<https://machinelearningmastery.com/time-series-data-stationary-python/>

# check if stationary

# adf\_results = adfuller(tsla['Adj Close'].diff().dropna()) # check alternative if below is not stationary

adf\_results = adfuller(tsla['Adj Close'].dropna())

print('ADF Statistic: %f' % adf\_results[0])

print('p-value: %f' % adf\_results[1], 'stationary' if (adf\_results[0] <= 0.05) else 'not stationary')

print('Critical Values:')

for key, value in adf\_results[4].items():

print('\t%s: %.3f' % (key, value), 'stationary' if (adf\_results[0] < value) else 'not stationary')

adf\_results

ADF Statistic: -23.460407

p-value: 0.000000

Critical Values:

1%: -3.443 stationary

5%: -2.867 stationary

10%: -2.570 stationary

Means -23.460407 < -2.867 it is *stationary*!

adf\_results[1] is p-value!:

* **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.

fig, ax = plt.subplots(1, 4, figsize = (20, 5))

plot\_acf(df[(df['store'] == 1) & (df['item'] == 1)]['sales'], ax = ax[0])

ax[0].set\_title('Original Series Autocorrelation')

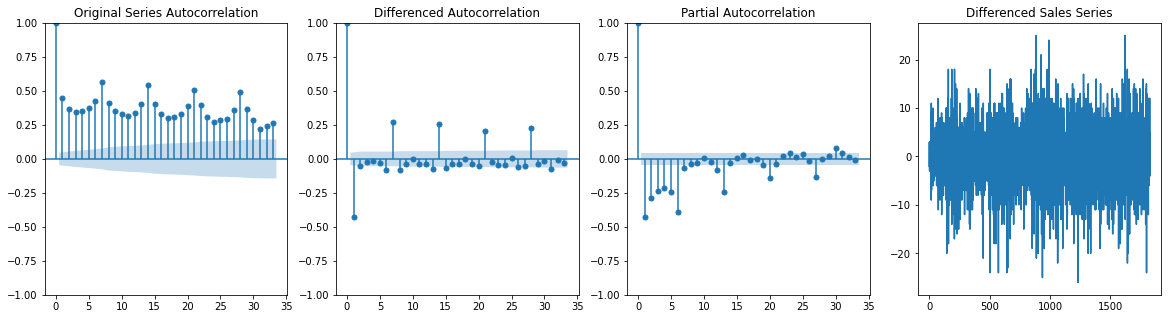
plot\_acf(df[(df['store'] == 1) & (df['item'] == 1)]['sales'].diff().dropna(), ax = ax[1])

ax[1].set\_title('Differenced Autocorrelation')

plot\_pacf(df[(df['store'] == 1) & (df['item'] == 1)]['sales'].diff().dropna(), ax = ax[2], method = 'ywm')

ax[3].plot(df[(df['store'] == 1) & (df['item'] == 1)]['sales'].diff().dropna())

ax[3].set\_title('Differenced Sales Series')



# Log & diff if needed

y = np.log(gnp).diff().dropna()

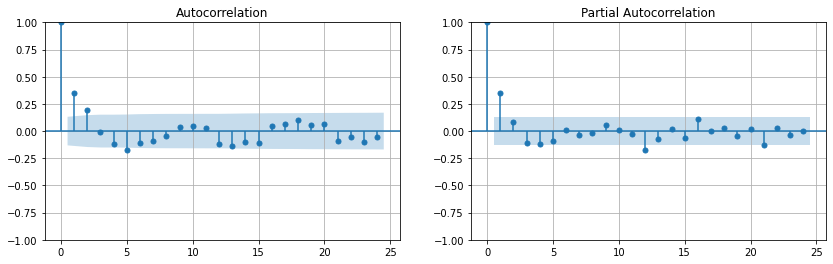
fig, ax = plt.subplots(1, 2, figsize = (14, 4))

plot\_acf(y, ax = ax[0])

ax[0].grid()

plot\_pacf(y, ax = ax[1])

ax[1].grid()



**Module Issues:**

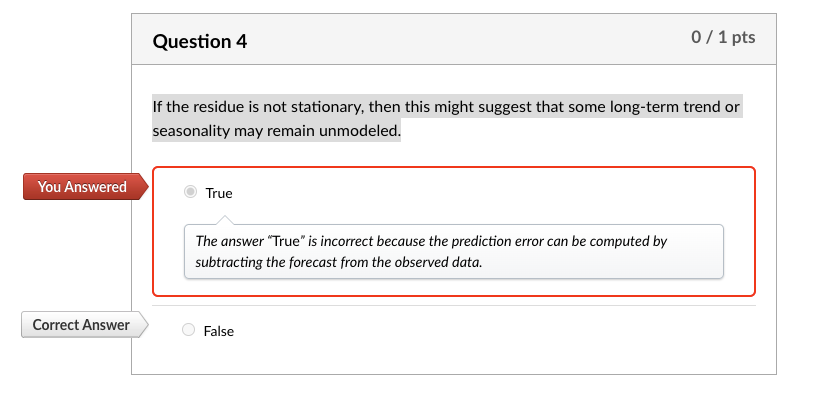
Codio 10.2 Problem 2: variable *sample* was not given in the cell upfront!

Codio 10.5 Problem 3: Fit an ARIMA model with p = 1 and q = **0**!

Codio 10.6 Problem 5: Create model with these options: “ARIMA(X\_train.diff(), order = (6, 0, 6), freq = 'D')”

Codio 10.6: Auto-grading failed to grade, upload *nbgrader\_config.py* per instructions!

In the module-end quiz the answer is wrong, pointing to something else:



**Quizes:**

Which of the following models does not represent time series data? : Autocorrelation

*You are correct! The answer “*Autocorrelation*” is correct because it is not the name of a model at all; it is an analytical tool.*

Given some historical data along a time series, predicting the future over some time period is referred to as a ‘forecasting problem.’ : True

*You are correct! The answer “*True*” is correct because a forecasting problem simply asks, “Given some historical data, what will happen in the future?”*

Prediction error is the sum of the predicted output and the actual data. : False

*You are correct! The answer “*False*” is correct because prediction error is the difference between the prediction and the actual data.*

Which of the following techniques is not commonly used to calculate the error of a model for time series data? : Mean squared error (MSE)

*You are correct! The answer “*Mean squared error (MSE)*” is correct because the most common techniques for calculating the error of a time series model are MAE and RMSE.*

What is a defining characteristic of stationary time series data? : The data has no trend or seasonal effects

*You are correct! “*The data has no trend or seasonal effects*” is correct because time series data is said to be stationary when there are no seasonal effects or clear trends in the data.*

A stochastic process is an unordered sequence of random variables. : False

*You are correct! The answer “*False*” is correct because a stochastic process is an ordered sequence of random variables.*

The stochastic process is represented as (Yt)1:T. What does the symbol “T” represent? : Length of stochastic process

*You are correct! The answer “*Length of stochastic process*” is correct because the symbol “T” represents the length of the stochastic process.*

A single sample from a stochastic process is known as (blank). : A time series

*You are correct! The answer “*A time series*” is correct because a single sample from a stochastic process is known as a time series.*

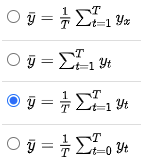
Independent and identically distributed (IID) processes are stationary. : False

*You are correct! The answer “*False*” is correct because IID processes are both stationary and independent.*

In the autocorrelation matrix, which value represents all values in the diagonal? : 1

*You are correct! The answer “*1*” is correct because in the diagonals, each value is the correlation of (Yt) with itself.*

What is the formula for the mean “y¯” of a time series? :



*You are correct! The answer "*y¯=1T∑t=1Tyt*" is correct because this is the formula for the mean of a time series.*

The Python library scikit-learn is used to build time series models. : False

*You are correct! The answer “*False*” is correct because the Python library statsmodels is used to build time series models.*

The ArmaProcess() object takes two constructors. What are these constructors? *(Check all that apply.)* : ar, ma

*You are correct! The answers “*ar*” and “*ma*” are correct because these are the two constructors used to create*ArmaProcess()*objects.*

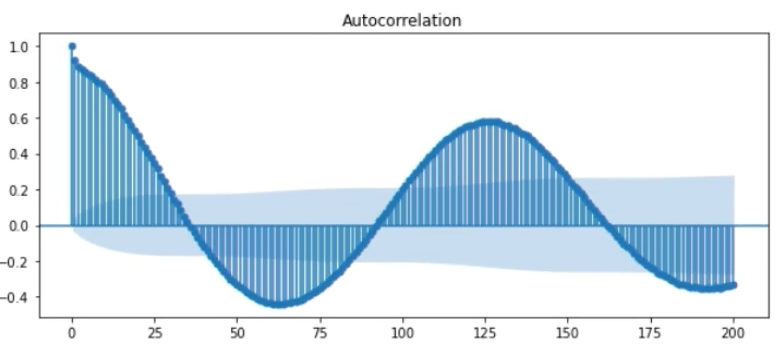
Which function is called on an ArmaProcess() object to give 20 autocorrelation values? : acf(lags=20)

*You are correct! The answer*“acf(lags=20)”*is correct because this is the function called on an*ArmaProcess()*object to get autocorrelation values.*

The function tsaplots.plot\_acf()is used to plot the autocorrelations of a sample of an ArmaProcess() object. : True

*You are correct! The answer “*True*” is correct because the function is used to plot the autocorrelations of a sample of an*ArmaProcess()*object.*

The plot below of the autocorrelation of a data sample shows that it is stationary. : False



*You are correct! The answer “*False*” is correct because for the data to be stationary, the autocorrelations should decay to zero.*

Any behavior in time series data that is not described by long-term behavior, random low-frequency variations, or known periodicity is considered a (blank). : Residual

*You are correct! The answer “*Residual*” is correct because any behavior in time series data that is not described by long-term behavior, random low-frequency variations, or known periodicity is considered a residual.*

To extract a trend from the time series data, a filter “f” is applied, which removes the components that are (blank). : Seasonal

*You are correct! The answer “*Seasonal*” is correct because the filter “f” is chosen as an array of positive numbers that sums up to 1. It will remove seasonal components for the period of the time series, leaving trends and cycles.*

If the residue is not stationary, then this might suggest that some long-term trend or seasonality may remain unmodeled. : True

*You are correct! The answer “*True*” is correct because if the residue is not stationary, it might suggest that trends or seasonality remain unmodeled.*

You can compute the time series prediction error by subtracting the trend from the observed data. : False

*You are correct! The answer “*False*” is correct because the prediction error can be computed by subtracting the forecast from the observed data.*

Which function is used in Python to extract the trend from time series data given the “filter” and the “data” as constructors? : convolution\_filter(data,filter)

*You are correct! The answer*“convolution\_filter(data,filter)”*is correct because this is the function used to extract trends from time series data given the “data” and “filter” as constructors.*

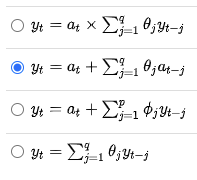
Which snippet of Python code allows you to get the mean absolute error of a data named “pred\_error”? : np.abs(pred\_error).mean()

*You are correct! The answer*“np.abs(pred\_error).mean()”*is correct because to get the mean absolute error, absolute values of the data are taken, and then their mean is computed.*

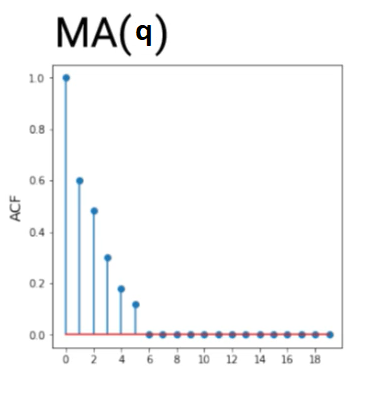
The ARMA family of models captures the time-variant structure exhibited in stationary time series. : False

*You are correct! The answer “*False*” is correct because the ARMA family of models captures the time-invariant structure exhibited in stationary time series.*

What is the mathematical formula for summarizing the moving average (MA) process with order “q”? : yt = at + sum of theta x at



*You are correct! The answer “*yt=at+∑j=1qθjat−j*” is correct because this is the formula for MA(q).*



Consider the above plot of ACF values for a moving average process with order “q”. What can you deduce as the value of q from the graph? : 5

*You are correct! The answer “*5*” is correct because the direct method for identifying the order of a moving average process is to look at the lag of the largest non-zero entry in the ACF.*

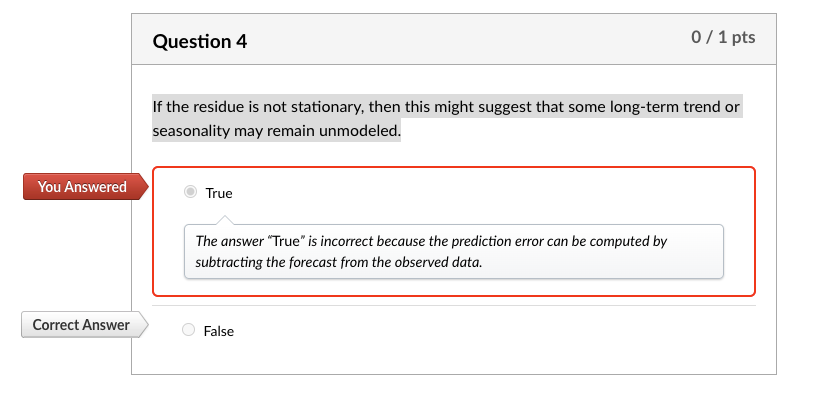
To get only the AR of an ARMA model, the parameter q is set to zero. : True

*You are correct! The answer “*True*” is correct because in the ARMA model, the parameter q is for MA. When q is set to zero, the model that remains is AR.*

What is the second step in building an ARMA model? : Use SACF and SPACF to select p and q

*You are correct! The answer “*Use SACF and SPACF to select p and q*” is correct because this is the second step in building an ARMA model.*

In the module-end quiz the answer is wrong, pointing to something else:



<https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/>

A given time series is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

These components are defined as follows:

* **Level**: The average value in the series.
* **Trend**: The increasing or decreasing value in the series.
* **Seasonality**: The repeating short-term cycle in the series.
* **Noise**: The random variation in the series.

**Try it 10.1: Decomposing Time Series**

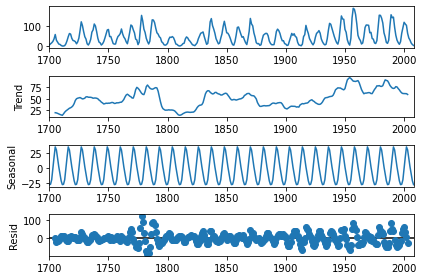
I studied sunspots and Air Passenger datasets for this exercise.

I directly set YEAR as index on sunspots, and transformed Month to timestamp and set it as index on air dataset, no cleanup or any other transformation. I set the period to 11 years (~128 month from the lecture video.):

ss\_results = seasonal\_decompose(ss, model = 'additive', period = 11)

ss\_results.plot()

plt.show



Reconstruct and overlay:

# check if aditive model match seasonal pattern when reconstructed

plt.plot(ss\_results.seasonal+ss\_results.trend, label = 'seasonal + trend')

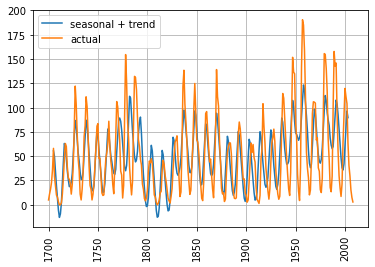
plt.plot(ss, label = 'actual')

plt.grid()

plt.legend()

plt.xticks(rotation = 90)

plt.show



It looks overlap with the actual data.

I analyzed two given datasets by seasonal\_decompose, it required a parameter to specify whether the model is *additive* or *multiplicative*. **sunspots** dataset has 0 values in SUNACTIVITY column so it went by **additive** model by default.

I did an *additive* model first on Air Passenger dataset, however, when I plotted *'seasonal + trend'* it did not overlap with the actual data as below:

# check if aditive model match seasonal pattern when reconstructed

plt.plot(air\_results.seasonal + air\_results.trend, label = 'seasonal + trend')

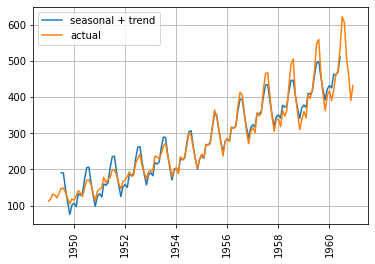
plt.plot(air, label = 'actual')

plt.grid()

plt.legend()

plt.xticks(rotation = 90)

plt.show

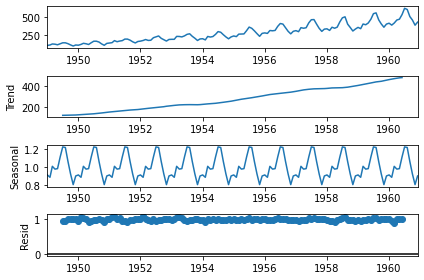


So, I went with a **multiplicative** model:

air\_results = seasonal\_decompose(air, model = 'multiplicative')

air\_results.plot()

plt.show



The **multiplicative** model plot overlapped well with the actual data comparing to the ‘additive’ model:

plt.plot(air\_results.seasonal \* air\_results.trend, label = 'seasonal \* trend')

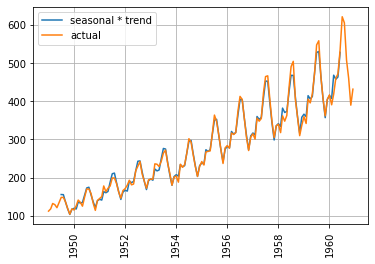
plt.plot(air, label = 'actual')

plt.grid()

plt.legend()

plt.xticks(rotation = 90)

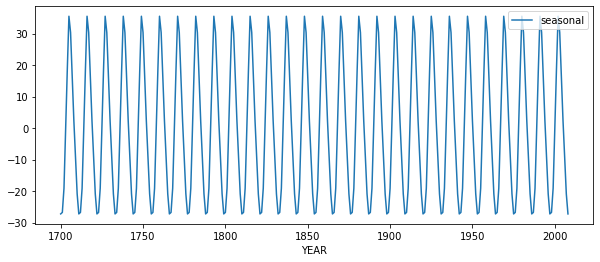
plt.show



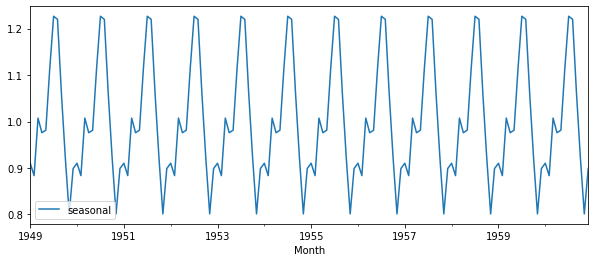
Just observe the height of each cycle for comparing these 2 plots, in the additive model they are the same heights of cycles whereas in the multiplicative model, the heights notably vary.

**Seasonal Components**

Sunspots seasonal component:



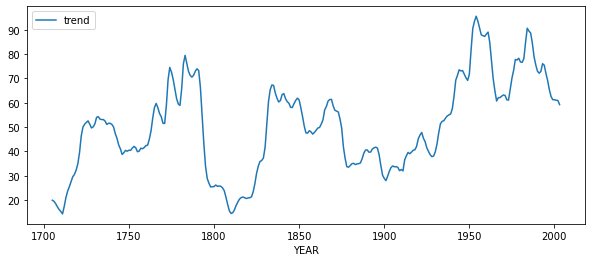
Air Passengers seasonal component:



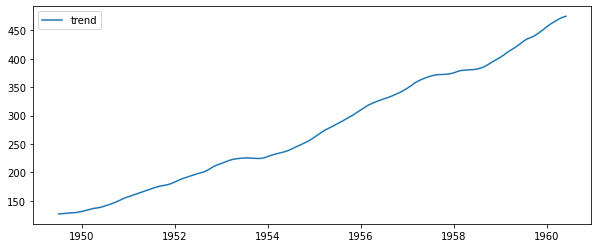
Both seasonal components on datasets showing similar behavior in periodical variability.

**Trend Components**

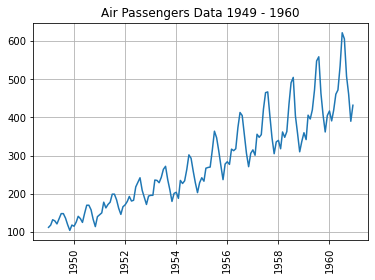
Sunspots trend component:



Air Passengers trend component:



*Air Passengers trend* shows gradual increase over time which differs from *Sunspots trend* shows erratic upward or downward movement.



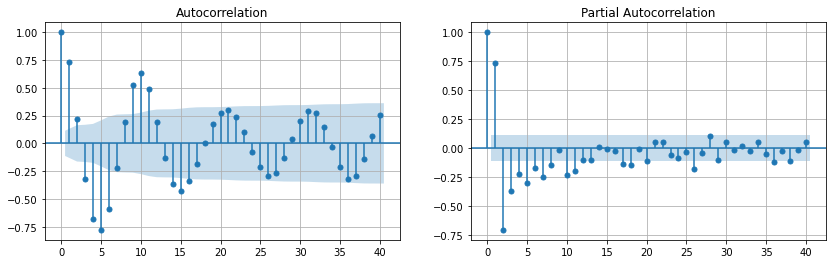
fig, ax = plt.subplots(1, 2, figsize = (14, 4))

plot\_acf(ss\_results.resid.dropna(), ax = ax[0], lags=40)

ax[0].grid()

plot\_pacf(ss\_results.resid.dropna(), ax = ax[1], lags=40)

ax[1].grid()



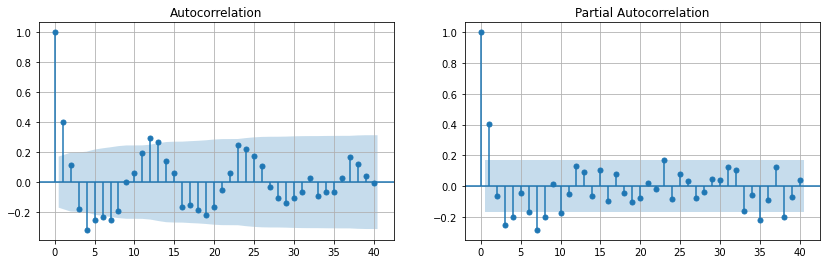
fig, ax = plt.subplots(1, 2, figsize = (14, 4))

plot\_acf(air\_results.resid.dropna(), ax = ax[0], lags=40)

ax[0].grid()

plot\_pacf(air\_results.resid.dropna(), ax = ax[1], lags=40)

ax[1].grid()



————— o —————

**Try-It Activity 10.2: Forecasting with Decomposition Models**

In this activity, your goal is to identify a new (to you) time series dataset and build a forecast using a seasonal and trend additive or multiplicative model using statsmodels.

Summarize your findings in an executive brief that explores the following:

* **Data description:** Provide a high-level overview of your data and its timeframe along with general information on your dataset
* **Forecast:** Give a description of the forecast. Describe the period that was projected and what the forecast implies about your data. Be sure to include presentation-ready plots with appropriate labels and titles.
* **Uncertainty:** Discuss the evaluation of your model on testing data, and explore the residuals. Discuss the consequence of this error for your model and forecasts. Is there still structure to uncover?

**ARIMA freq parameter**

freq : str {'B','D','W','M','A', 'Q'}

'B' - business day, ie., Mon. - Fri.

'D' - daily

'W' - weekly

'M' - monthly

'A' - annual

'Q' - quarterly

Additive / Multiplicative

Univariate Time Series

Detrended data: mean is at 0!

——

**Data Introduction and Cleanup**

I found a dataset contains 9357 records of air pollutants with hourly averaged responses from 5 sensors, the measurement was taken for one year at road level in an Italian town: <https://archive.ics.uci.edu/ml/datasets/Air+Quality>. I only used CO reading, I did some cleanup as follows:

1. Made CO column float after replacing decimal symbol to ‘.’ from ‘,’: airq['CO'] = airq['CO(GT)'].str.replace(',','.').astype('float')
2. Missing measurement marked as -200, replaced them with mean() value: airq[airq['CO'] == -200] = 2.15275
3. Transformed date and time columns to airq['datetime'] = pd.to\_datetime(aq['Date'] + ' ' + aq['Time'], format='%d/%m/%Y %H.%M.%S') and set index

**Build historical and future datasets**

Split last 4 days (96 hours) for future and rest in hist:

# split data into hist and future

y\_hist = airq[:-96]

y\_future = airq[-96:]

**Extracting the trend**

**10 Points**

I created an instance of the STL estimator and passed y\_hist, period = 24 for one day.

stl = STL(y\_hist, period=24)

results = stl.fit()

# Plot trend with original series

plt.plot(y\_hist)

plt.plot(results.trend)

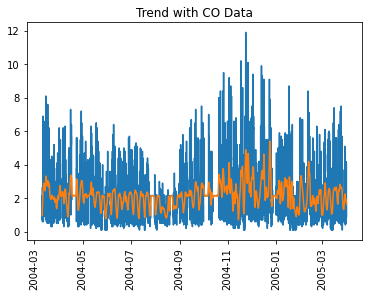
#plt.plot(results.seasonal)

plt.title('Trend with CO Data')

#plt.legend()

plt.xticks(rotation = 90)

plt.show



**Model Historical Series and Seasonal+Trend**

I checked the seasonal+trend plot if it is a good model, looked good:

# additive model fits well! no need for multiplicative model…

season\_and\_trend = results.seasonal + results.trend

# Plot

#plt.plot(season\_and\_trend, label = 'seasonal + trend')

plt.plot(y\_hist, label = 'actual')

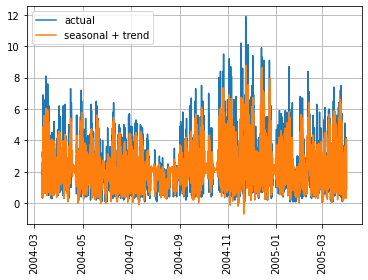
plt.plot(season\_and\_trend, label = 'seasonal + trend')

plt.grid()

plt.legend()

plt.xticks(rotation = 90)

plt.show



**Examining the residuals, Examining Error in Forecast**

Then, I checked both residual plot and Dickey Fuller test below:

# plot residual on actual series

plt.plot(y\_hist, label = 'actual')

plt.plot(results.resid, label = 'residue')

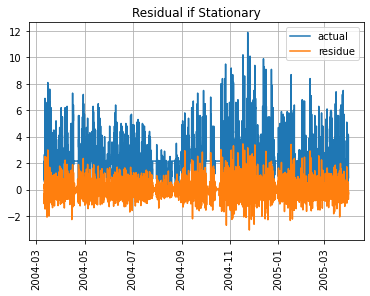
plt.grid()

plt.title('Residual if Stationary')

plt.legend()

plt.xticks(rotation = 90)

plt.show



# check stationarity by Dickey Fuller at different thresholds

adf\_results = adfuller(results.resid)

print('ADF Statistic: %f' % adf\_results[0])

print('p-value: %f' % adf\_results[1], 'stationary' if (adf\_results[1] <= 0.05) else 'not stationary')

print('Critical Values:')

for key, value in adf\_results[4].items():

print('\t%s: %.3f' % (key, value), 'stationary' if (adf\_results[0] < value) else 'not stationary')

adf\_results

ADF Statistic: -21.211974

p-value: 0.000000 stationary

Critical Values:

1%: -3.431 stationary

5%: -2.862 stationary

10%: -2.567 stationary

(-21.211974346936564,

0.0,

38,

9222,

{'1%': -3.431059295288066,

'5%': -2.8618534634125252,

'10%': -2.566936851507016},

9094.238191568482)

Passed! Stationary!

Forecast future values to compare, I played with p,d,q parameters, below produced better results in terms of MAE and RMSE values:

#instantiate

stlf = STLForecast(y\_hist, ARIMA, model\_kwargs={'order':(2, 0, 2), 'trend':"t", 'enforce\_stationarity': True})

#fit model using historical data

stlf\_results = stlf.fit()

#produce forecast for future data

forecast = stlf\_results.forecast(len(y\_future))

Plot entire series

plt.subplots(figsize=(16,6))

plt.plot(forecast, label = 'forecast')

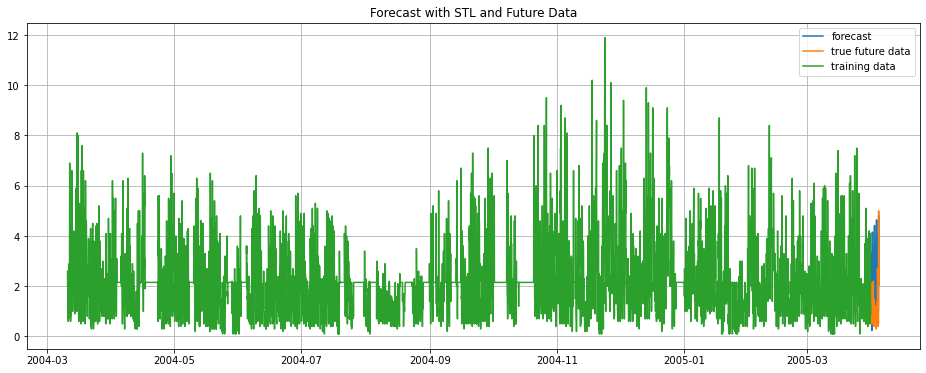
plt.plot(y\_future, label = 'true future data')

plt.plot(y\_hist, label = 'training data')

plt.legend()

plt.title('Forecast with STL and Future Data')

plt.grid()



I compared the forecast with future data if matched:

plt.subplots(figsize=(10,4))

plt.plot(forecast, label = 'Forecast')

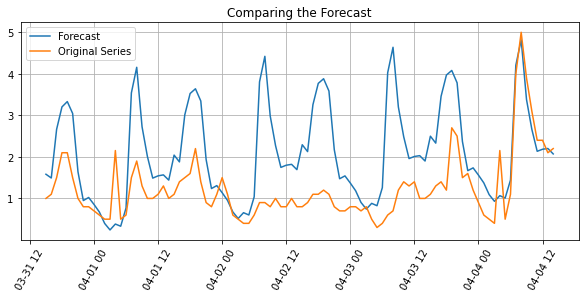
plt.plot(y\_future, label = 'Original Series')

plt.title('Comparing the Forecast')

plt.legend()

plt.xticks(rotation = 60)

plt.grid()



Prediction errors:

pred\_error = y\_future['CO'] - forecast

mae = np.abs(pred\_error).mean()

rmse = np.sqrt(np.square(pred\_error).mean())

rmse2 = np.sqrt(mean\_squared\_error(y\_future['CO'], forecast))

# Results

print(f'MAE : {mae}')

print(f'RMSE : {rmse}')

print(f'RMSE2 : {rmse2}')

MAE : 0.9724595068578156

RMSE : 1.2966376019704622

RMSE2 : 1.2966376019704622

**Conclusion**

The dataset has only one year of data, may not be enough for showing seasonal pattern, trend, by looking at the forecast and actual plot above, there is some room for improvement perhaps with different modeling techniques.

——

Do not include:…..

I plotted Autocorrelation and Partial autocorrelation charts:

# plot ACF and PACF with plain series

fig, ax = plt.subplots(1, 2, figsize = (14, 4))

# ACF

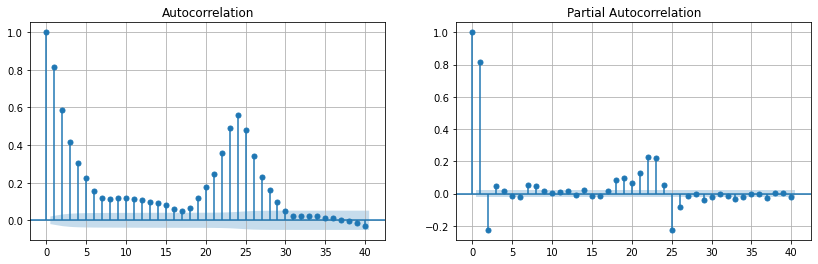
plot\_acf(airq, ax = ax[0])

ax[0].grid()

# PACF

plot\_pacf(airq, ax = ax[1])

ax[1].grid()



——

————— o —————

**Try-It Activity 10.3: Building and Evaluating ARMA Models**

First, you learned about the importance of transforming your data into a stationary series. You tested for stationarity using the adfuller function and interpreted the value of the hypothesis test. If the data was not stationary, you learned how to apply strategies, such as differencing and logarithmic transformations, to achieve stationarity.

Once the series was stationary, building an ARMA model involved using autocorrelation and partial autocorrelation plots to determine the appropriate *p* and *q* parameters of the model.

This activity asks you to identify a time series of interest to you and build an ARMA model to construct a basic forecast for the series and analyze the error. You might also consider building models with different *p* and *q* parameters because, while ACF and PACF plots are helpful, they provide rough ideas of the appropriate parameters, and it is usually good practice to perform a simple grid search on these.

* **AR**: *Autoregression*. A model that uses the dependent relationship between an observation and some number of lagged observations.
* **I**: *Integrated*. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
* **MA**: *Moving Average*. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

The parameters of the ARIMA model are defined as follows:

* **p**: The number of lag observations included in the model, also called the lag order, is the number of autoregressive terms.
* **d**: The number of times that the raw observations are differenced, also called the degree of differencing, is the number of nonseasonal differences needed for stationarity.
* **q**: The size of the moving average window, also called the order of moving average, is the number of lagged forecast errors in the prediction equation.

a suite of lag values (*p*) and just a few difference iterations (*d*) and residual error lag values (*q*).

*autocorrelations* (correlations with its own prior deviations from the mean) remain constant over time

[How to Grid Search ARIMA Model Hyperparameters with Python](https://machinelearningmastery.com/grid-search-arima-hyperparameters-with-python/)

**Dataset Introduction**

I found a time series dataset for price of gold from 1970 to 2020 on Kaggle: <https://www.kaggle.com/datasets/arashnic/learn-time-series-forecasting-from-gold-price>, there are 10787 records, it is ready to use once Date is transferred to timestamp data type and set as index:

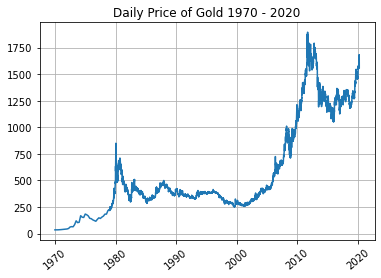
gold=gd.set\_index(pd.to\_datetime(gd['Date'])).drop('Date', axis=1)

plt.plot(gold['Value'])

plt.grid()

plt.xticks(rotation = 40)

plt.title('Daily Price of Gold 1970 - 2020')



I checked ACF and PACF also Dickey Fuller test on gold dataset:

# plot ACF and PACF with plain series

fig, ax = plt.subplots(1, 2, figsize = (14, 4))

# ACF

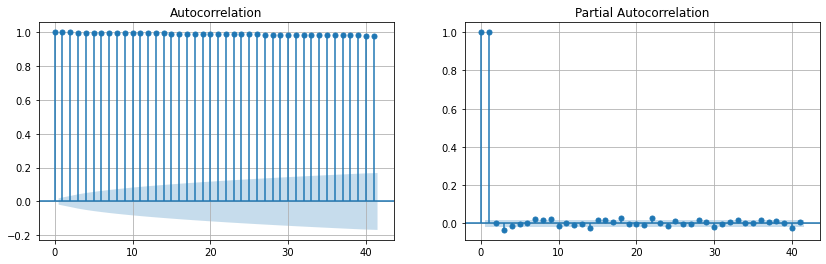
plot\_acf(gold, ax = ax[0])

ax[0].grid()

# PACF

plot\_pacf(gold, ax = ax[1])

ax[1].grid()



It is not stationary data, Dickey Fuller test confirms that too:

# check stationarity by Dickey Fuller at different thresholds

adf\_results = adfuller(gold)

print('ADF Statistic: %f' % adf\_results[0])

print('p-value: %f' % adf\_results[1], 'stationary' if (adf\_results[1] <= 0.05) else 'not stationary')

print('Critical Values:')

for key, value in adf\_results[4].items():

print('\t%s: %.3f' % (key, value), 'stationary' if (adf\_results[0] < value) else 'not stationary')

adf\_results

ADF Statistic: -0.071121

p-value: 0.952240 not stationary

Critical Values:

1%: -3.431 not stationary

5%: -2.862 not stationary

10%: -2.567 not stationary

Out[98]:

(-0.07112124453813447,

0.9522400769147394,

39,

10747,

{'1%': -3.4309586221840513,

'5%': -2.861808976860248,

'10%': -2.566913171245489},

77621.49868017703)

I created a differenced dataset from this series and plotted ACF, PACF again, Dickey Fuller test:

# do diff and dropna

gold\_df = gold.diff().dropna()

# plot charts

fig, ax = plt.subplots(1, 3, figsize = (20, 5))

plot\_acf(gold\_df, lags=20, ax = ax[0])

ax[0].set\_title('Gold Series Autocorrelation')

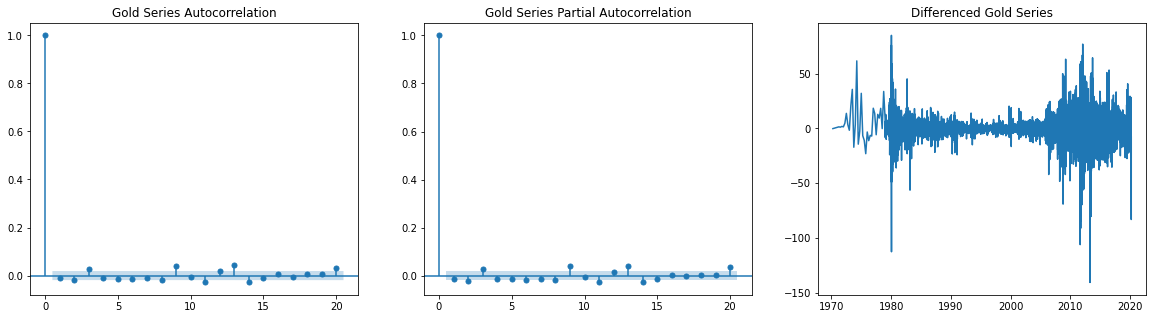
plot\_pacf(gold\_df, lags=20, ax = ax[1])

ax[1].set\_title('Gold Series Partial Autocorrelation')

ax[2].plot(gold\_df)

ax[2].set\_title('Differenced Gold Series')

plt.show



It is stationary!

I created a simple ARMA model and a grid search for it, I split dataset into historical and last 30 days for forecast in the function:

def arma\_grid\_search(data, p\_values, d\_values, q\_values):

# split data

y\_hist, y\_future = data[:-30], data[-30:]

best\_rmse, best\_order = float("inf"), None

for p in p\_values:

for d in d\_values:

for q in q\_values:

arima\_order = (p, d, q)

# build ARMA model & fit

arma = ARIMA(y\_hist, order = arima\_order).fit()

hist\_preds = arma.predict()

# forecast

forecast = arma.forecast(len(y\_future))

# set index same as y\_future!

forecast.index=y\_future.index

pred\_error = y\_future['Value'] - forecast

mae = np.abs(pred\_error).mean()

rmse = np.sqrt(np.square(pred\_error).mean())

print('ARIMA Order=%s MSE=%.3f' % (arima\_order, rmse))

if rmse < best\_rmse:

best\_rmse, best\_config = rmse, arima\_order

# Results

print('Best ARIMA Order=%s MSE=%.3f' % (best\_config, best\_rmse))

# seed values from PACF plot!

p\_values = [0, 1, 2, 3, 9, 13]

d\_values = [0]

q\_values = range(0, 3)

arma\_grid\_search(gold\_df, p\_values, d\_values, q\_values)

The best order option came as (13,0,2), when I fed it into model and predict:

arma = ARIMA(y\_hist, order = (13, 0, 2)).fit()

hist\_preds = arma.predict()

# Results

print(hist\_preds.tail())

plt.figure(figsize = (12, 4))

plt.plot(y\_hist, label = 'Historical')

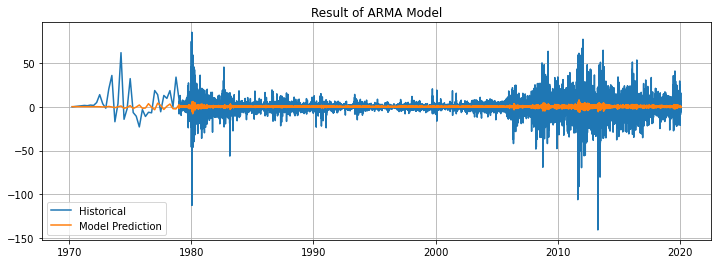
plt.plot(hist\_preds, label = 'Model Prediction')

plt.legend()

plt.grid()

plt.title('Result of ARMA Model')

plt.show



# forecast

forecast = arma.forecast(len(y\_future))

# set index same as y\_future!

forecast.index=y\_future.index

pred\_error = y\_future['Value'] - forecast

mae = np.abs(pred\_error).mean()

rmse = np.sqrt(np.square(pred\_error).mean())

# Results

print(f'MAE : {mae}')

print(f'RMSE : {rmse}')

MAE : 15.945910619864206

RMSE : 22.13985346444464

plt.subplots(figsize=(10,4))

plt.plot(y\_future, label = 'Differenced Gold Series')

plt.plot(forecast, label = 'Forecast')

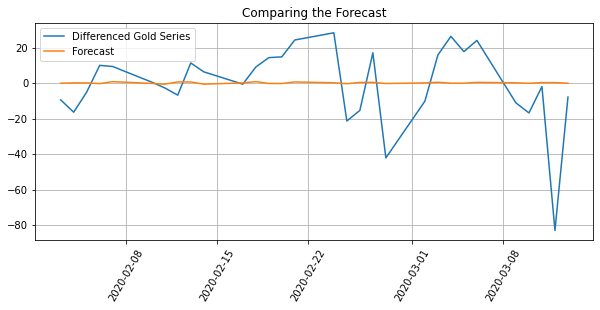
plt.title('Comparing the Forecast')

plt.legend()

plt.xticks(rotation = 60)

plt.grid()

plt.show



I could only come up a simple grid search by utilizing for loop, although, it vigorously iterating all given possibilities, the forecast fluctuates at zero line, however, it does not seem follow trend as its root mean squared error considerably remained high.

 ————— o —————