




DO NOT DISTRIBUTE Proceedings of the REU2022








Cybertraining




EARLY DRAFT, laszewski@gmail.com

04 June, 2022

Contents

1	USEFUL LINKS	5
1.1	REU 2022 	5
1.1.1	Books	5
1.1.2	The repository for our report(s)	5
1.1.3	The many UVA technical help ticket systems	5
1.1.4	Our technical slack	5
1.1.5	The administrative slack	5
1.1.6	Rivanna Information	6
1.1.7	GitBash (Windows only)	6
1.1.8	Pycharm	6
1.1.9	Bibliography Management	6
1.1.10	GitHub repo	7
1.1.11	Only important for Gregor (do not use)	7
2	TIMESERIES WITH PYTHON	7
2.1	Time Series Prediction 	7
2.1.1	Advantages of Time series prediction	7
2.1.2	Time series algorithm	7
2.1.3	References	8
2.2	ANOVA Time Series Prediction 	9
2.2.1	One Way ANOVA	9
2.2.1.1	Output	10
2.2.1.1.1	Output Explanation	10
2.2.2	Two Way ANOVA	10
2.2.2.1	Output	11
2.2.2.2	Output Explanation	11
2.2.3	ANOVA Time series prediction	11
2.2.4	Reference	12
2.3	ARIMA Time Series Prediction 	12
2.3.1	So, how do you make a series stationary?	12
2.3.2	What exactly are AR and MA models?	13
2.3.2.1	ARIMA Python Implementation	13
2.3.3	Rolling Forecast ARIMA Model	15
2.3.4	Dataset	17
2.3.5	References	18

2.4	Autocorrelation in images/Time Series 	18
2.4.1	Partial Autocorrelation Function	19
2.4.2	Reference	20
2.5	Detrend a Time Series 	21
2.5.1	Detrend by Differencing	21
2.5.2	Detrend by Model Fitting	21
2.5.3	Reference	22
2.6	Granger casualty test 	22
2.6.1	Output	23
2.6.2	Reference	23
2.7	Lag Plot 	23
2.7.1	Model Distribution	24
2.7.2	Outliers	24
2.7.3	Randomness in data	24
2.7.4	Seasonality	24
2.7.5	Autocorrelation	24
2.7.6	Reference	26
2.8	Sample Entropy 	26
2.8.1	Python Example	27
2.8.2	Output	27
2.8.2.1	Output Breakdown	27
2.8.3	Reference	28
2.8.3.1	Dataset:	28
2.9	Seasonal decomposition from stats model 	28
2.9.1	Additive Decomposition	28
2.9.2	Multiplicative Decomposition	30
2.9.3	Desasonalising of Time Series	31
2.9.4	References	33
2.10	Stationary and Non-Stationary Time Series 	33
2.10.1	Stationary Time series	33
2.10.1.1	How to make a time series stationary?	34
2.10.1.1.1	Differencing the Series (once or more)	34
2.10.2	How to test for stationary?	34
2.10.2.1	Visual Test	35
2.10.2.2	Statistical test	36
2.10.2.3	Augmented Dickey Fuller test (ADH Test)	37
2.10.2.4	Kwiatkowski-Phillips-Schmidt-Shin – KPSS test	38

2.10.3	Types of Stationary Time series	38
2.10.3.1	Strict Stationary	38
2.10.3.2	Trend Stationary	39
2.10.3.3	Difference Stationary	39
2.10.4	References	39
2.11	Smoothen 	39
2.11.1	Take a Moving Average	40
2.11.2	LOESS smoothing (Localized Regression)	40
2.11.3	Reference	43
2.12	Temporal Fusion Transformer(TFT) 	43
2.12.1	Advantages of Temporal Fusion Transformer	43
2.12.1.1	Rich features	43
2.12.1.2	Heterogeneous time series	43
2.12.1.3	Multi-horizon forecasting	43
2.12.1.4	Interpretability	43
2.12.1.5	High Performance	44
2.12.1.6	Documentation	44
2.12.1.7	References	44
2.13	Time series in python 	45
2.13.1	Time Series prediction	45
2.13.2	Anova Time series	45
2.13.3	TFT	45
2.13.4	ARIMA Time series	45
2.13.5	Seasonal Decomposition from StatsModel	46
2.13.6	Stationary and Non-Stationary Time Series	46
2.13.7	AutoCorrelation	46
2.13.8	Lag Plot	46
2.13.9	Smoothen	46
2.13.10	Granger casualty test	47

3 REFERENCES

47

1 USEFUL LINKS

1.1 REU 2022

1.1.1 Books

- <https://cloudmesh-community.github.io/pub/vonLaszewski-python.pdf>
- <https://cloudmesh-community.github.io/pub/vonLaszewski-python.epub>
- <https://cloudmesh-community.github.io/pub/vonLaszewski-linux.pdf>
- <https://cloudmesh-community.github.io/pub/vonLaszewski-linux.epub>
- <https://cloudmesh.github.io/cloudmesh-mpi/report-mpi.pdf>
- <https://cloudmesh-community.github.io/pub/vonLaszewski-writing.pdf>
- <https://cloudmesh-community.github.io/pub/vonLaszewski-communicate.pdf>
- <https://cloudmesh-community.github.io/pub/vonLaszewski-reu2022.pdf>

1.1.2 The repository for our report(s)

- <https://github.com/cybertraining-dsc/reu2022>

1.1.3 The many UVA technical help ticket systems

- ITS: <https://virginia.edusupportcenter.com/shp/uva/helpcenter>
- Rivanna: <https://varesearchhelp.atlassian.net/servicedesk/customer/user/requests?page=1&reporter=all>
- BII: <https://uva-biocomplexity.atlassian.net/servicedesk/customer/user/requests?page=1&reporter=all&statuses=open&statuses=closed>

1.1.4 Our technical slack

- <https://cloudmesh-reu2022.slack.com>

1.1.5 The administrative slack

- <https://biocomplexity-eoo5671.slack.com/archives/C031CR0B2QG>

1.1.6 Rivanna Information

- Research Computing: <https://www.rc.virginia.edu/>
- Rivanna Overview: <https://www.rc.virginia.edu/userinfo/rivanna/overview/>
- Research Computing Support Center: <https://www.rc.virginia.edu/support/#office-hours>
- access through ssh: <https://www.rc.virginia.edu/userinfo/rivanna/login/>
- access through Web Browser: <https://shibidp.its.virginia.edu/idp/profile/SAML2/Redirect/SSO?execution=e2s1>
- Rivanna Slide Show: <https://learning.rc.virginia.edu/notes/rivanna-intro/>
- Globus Data Transfer (not using for reu2022) <https://www.rc.virginia.edu/userinfo/globus/>
- Slurm Job Manager Documentation <https://www.rc.virginia.edu/userinfo/rivanna/slurm/>
- Rivanna Allocations: <https://www.rc.virginia.edu/userinfo/rivanna/allocations/>

1.1.7 GitBash (Windows only)

- <https://git-scm.com/downloads>

1.1.8 Pycharm

- pycharm community edition: <https://www.jetbrains.com/pycharm/download>
- md: <https://www.jetbrains.com/help/pycharm/markdown.html>
- (optional) extension makefile: <https://plugins.jetbrains.com/plugin/9333-makefile-language>
- (optional) rst: <https://www.jetbrains.com/help/pycharm/restructured-text.html>
- 80 char: <https://intellij-support.jetbrains.com/hc/en-us/community/posts/206070859-How-do-I-enable-the-80-column-guideline-change>
- background to white as i can not read white on black when in meetings with me I will not support anyone that has black background: <https://www.jetbrains.com/help/pycharm/user-interface-themes.html> use IntelliJ Light. After the meeting you can set it to whatever.

1.1.9 Bibliography Management

- jabref: <https://www.jabref.org/>
- bibtex: <https://en.wikipedia.org/wiki/BibTeX>
- draft bibtex from urls: <https://addons.mozilla.org/en-US/firefox/addon/bibitnow/> (has to be modified once copied. Not everything will work.

1.1.10 GitHub repo

- <https://github.com/cybertraining-dsc/reu2022>
- Sandra, JP: <https://github.com/cloudmesh/cloudmesh-mpi>

1.1.11 Only important for Gregor (do not use)

- request storage: <https://www.rc.virginia.edu/form/storage/>
- rivanna partition/account: <https://www.rc.virginia.edu/userinfo/rivanna/allocations/>
- managing groups: <https://mygroups.virginia.edu/>

2 TIMESERIES WITH PYTHON

2.1 Time Series Prediction

An extensive document on time series prediction is located in

- <https://www.tessellationtech.io/3-advantages-to-time-series-analysis-and-forecasting/>
- <https://www.tableau.com/learn/articles/time-series-forecasting#:~:text=Time%20series%20forecasting%20occurs%20when,drive%20future%20strategic%20decision%2Dmaking.>

Time series prediction is the modeling for Time Series data (years, days, hours, etc.) for predicting future values using Time Series modeling. This helps if your data is serially correlated.

Time series can only be performed dependent on the set of time series data provided.

2.1.1 Advantages of Time series prediction

- Time Series Analysis Helps You Identify Patterns. Memories are fragile and prone to error
- Time Series Analysis Creates the Opportunity to Clean Your Data.
- Time Series Forecasting Can Predict the Future.

2.1.2 Time series algorithm

Time series can be done in different form be it the AI or non AI methods the table below classify each of the time series predicting algorithm in their method.

```
* Time-series-prediction
* AI
  * Temporal Fusion Transformer
  * ARIMA
    * Smoothen
    * Stationary and Non-stationary
    * Seasonal Model
      * Detrend
  * ANOVA
  * StatModel
  * Sample Entropy
* NON AI
  * Regression
  * Augmented Dickey-Fuller test
  * Granger Casualty Test
  * Lag plot
```

This talks about the different forms of time series prediction. such as the ANOVA model used to determine whether a survey or The experiment results are significant. The ARIMA model, which is a class statistical model for time series analysis and forecasting, and the Temporal fusion transformer, provide additional insight on feature importance by utilizing self-attention when forecasting. We will get more into them as we proceed. It also It discusses various aspects of time series, such as autocorrelation. of the time series which checks the relationship between a variable and current value and the previous one. Stationary and non-stationary time series which checks if a time series changes over time or not. seasonal decomposition, which entails viewing a time series as a deseasonalizing a time, trend, seasonality, and noise component which is removing seasonality from the time series, detrending time series, which is removing trends from a time series. Granger-casualty-test which is used to check if a date can be predicted or not. lag-plot is a type of scatter plot in which the x and y-axis are lagged, SampleEntropy, which is a technique for quantifying the amount of regularity and unpredictability of small time series, and Smoothen To remove fine-grained variation between time steps, time series techniques are used.

2.1.3 References

- <https://www.tessellationtech.io/3-advantages-to-time-series-analysis-and-forecasting/>
- <https://www.tableau.com/learn/articles/time-series-forecasting#:~:text=Time%20series%20forecasting%20occurs%20when,drive%20future%20strategic%20decision%2Dmaking.>

2.2 ANOVA Time Series Prediction

An extensive documentation on ANOVA Time series prediction are Available at

- <https://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/anova/>
- <https://medium.com/@stallonejacob/time-series-forecast-a-basic-introduction-using-python-414fcb963000>
- <https://www.geeksforgeeks.org/how-to-perform-a-two-way-anova-in-python/>
- <https://www.geeksforgeeks.org/how-to-perform-a-one-way-anova-in-python/>

An ANOVA test is used to determine whether survey or experiment results are significant. In other words, they assist you in determining whether you should reject the null hypothesis or accept the alternative hypothesis. it's categories into two namely:

- One Way ANOVA
- Two Way ANOVA

2.2.1 One Way ANOVA

A one way ANOVA is used to compare two means from two independent groups using the F-distribution. The null hypothesis for the test is that the two means are equal. Therefore, a significant result means that the two means are unequal. a situation when these can be used is when a group of individuals randomly split into smaller groups and completing different tasks. For example, you might be studying the amount of task for all the individual and given them a score per how efficient they performed the task as shown below.

Example formatted from [4]

```
from scipy.stats import f_oneway

# rated scores
score1 = [84, 86, 68, 98, 49]
score2 = [83, 29, 64, 89, 88]
score3 = [69, 88, 69, 73, 40]
score4 = [81, 70, 81, 92, 82]

scores_ave= f_oneway(score1, score2, score3, score4)

print(scores_ave)
```

2.2.1.1 Output

```
F_onewayResult(statistic=0.5410224469358347, pvalue=0.6610639388335927)
```

2.2.1.1.1 Output Explanation The statistic and p-value turn out to be equal to 0.5410 and 0.661063. Since the p-value is greater than 0.05 hence we would accept the null hypothesis. This implies that we have sufficient proof to say that the performance of the students are similar

2.2.2 Two Way ANOVA

A Two Way ANOVA is an extension of the One Way ANOVA. With a One Way ANOVA, you have one independent variable affecting a dependent variable. With a Two Way ANOVA, there are two independents. Use a Two way ANOVA when you have one dependent variable and two independent variables. In other words, if your experiment has a quantitative outcome, and you have two independent variables, a two-way ANOVA is appropriate. To manage a Two Way ANOVA data set two python libraries are required which are the numpy and panda are needed. to perform the Two Way ANOVA the statistical model is needed, known as the statsmodel. statsmodels is a Python package that provides descriptive statistics and estimation and inference for statistical models. to install the python libraries using `pip` below

Dataset Management Libraries

```
pip3 install numpy pandas
```

Statistical models Library

```
pip install statsmodels
```

Example Formatted from [3]

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Create a dataframe
dataframe = pd.DataFrame({'shared_post': np.repeat(['daily', 'weekly'], 15),
                          'comments_likes': np.repeat(['daily', 'weekly'], 15),
                          'SocialMedia_growth': [10, 18, 16, 14, 19, 16, 18, 14,
                                                    17, 18, 17, 17, 18, 17, 19, 11,
                                                    16, 15, 15, 16, 17, 15, 16, 15,
                                                    19, 11, 18, 15, 15, 12]})

# Performing two-way ANOVA
```

```

result = sm.stats.anova_lm(ols('SocialMedia_growth ~ C(shared_post) + C(
    ↪ comments_likes) +\
C(shared_post):C(comments_likes)', data=dataframe).fit(), type=2)

# Print the result
print(result)

```

2.2.2.1 Output

	df	sum_sq	...	F	PR(>F)
C(shared_post)	1.0	16.133333	...	2.998230	0.094362
C(comments_likes)	1.0	0.049552	...	0.009209	0.924234
C(shared_post):C(comments_likes)	1.0	0.012326	...	0.002291	0.962168
Residual	28.0	150.666667	...	NaN	NaN

2.2.2.2 Output Explanation

Following are the p-values for each of the factors in the output:

The shared_post p-value is equal to 0.94362 The comments_likes p-value is equal to 0.924234 The shared_post * comments_likes: p-value is equal to 0.962168

The p-values for shared_post and comments_likes turn out to be greater than 0.05 which implies that the means of both the factors possess a similar effect on the user SocialMedia_growth. The p-value for the interaction effect (0.962168) is greater than 0.05 which shows the interaction effect between shared_post frequency and comments_likes frequency.

Test the following codes below:

- One Way ANOVA <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/anova/anova-test.py>
- Two Way ANOVA <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/anova/anovatest2.py>

2.2.3 ANOVA Time series prediction

The ANOVA time series prediction deals with using the grand mean of the occurrence of past to predict the output of what the probable output will be in the future. the example in the TWO Way ANOVA section that predicts a user social media growth is a perfect example of the predicting time series with ANOVA.

2.2.4 Reference

- [1] <https://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/anova/>
- [2] <https://medium.com/@stallonejacob/time-series-forecast-a-basic-introduction-using-python-414fcb963000>
- [3] <https://www.geeksforgeeks.org/how-to-perform-a-two-way-anova-in-python/>
- [4] <https://www.geeksforgeeks.org/how-to-perform-a-one-way-anova-in-python/>

2.3 ARIMA Time Series Prediction

Extensive documents on ARIMA Time series prediction are available at

- <https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/>
- <https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/#:~:text=ARIMA%2C%20short%20for%20'AutoRegressive%20Integrated,to%20predict%20the%20future%20values.>

ARIMA is a forecasting algorithm that stands for 'AutoRegressive Integrated Moving Average'. It is a class of statistical models for analyzing and forecasting time series data. The ARIMA model is defined by three values namely: p , d , and q .

where:

The order of the AR term is denoted by p .

The order of the MA term is denoted by q .

The differencing required to make the time series stationary is denoted by d .

2.3.1 So, how do you make a series stationary?

The most common method is to differentiate it. To put it another way, subtract the previous value from the current value. Depending on the complexity of the series, more than one differencing may be required at times.

As a result, the value of d is the smallest number of differences required to make the series stationary. And $d = 0$ if the time series is already stationary.

2.3.2 What exactly are AR and MA models?

A pure Auto Regressive (AR only) model is one in which Y_t is solely determined by its lags. That is, Y_t is a function of the ' Y_t lags.'

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \dots + \beta_p Y_{t-p} + \epsilon_t$$

'where $Y_t - 1$ is the series' lag1, β_1 is the lag1 coefficient estimated by the model, and α is the intercept term estimated by the model.'

Similarly, a pure Moving Average (MA only) model is one in which Y_t is determined solely by the lagged forecast errors.

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} \dots + \phi_q \epsilon_{t-q}$$

An ARIMA model is one in which the time series is different at least once to make it stationary and the AR and MA terms are combined. As a result, the equation is:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \dots + \beta_p Y_{t-p} + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \dots + \phi_q \epsilon_{t-q}$$

Predicted $Y_t = \text{Constant} + \text{Linear combination of } Y(\text{upto lags}) + \text{Linear Combination of Lagged forecast errors}$

2.3.2.1 ARIMA Python Implementation Example is Formatted from [1]

```
from datetime import datetime
from pandas import read_csv
from pandas import DataFrame
from statsmodels.tsa.arima.model import ARIMA
from matplotlib import pyplot

# load dataset
def parser(x):
    return datetime.strptime('200' + x, '%Y-%m')

series = read_csv('https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/
    ↪ main/time-series-prediction/temperature2.csv',
    header=0, index_col=0, parse_dates=True, squeeze=True, date_parser
    ↪ =parser)
```

```
series.index = series.index.to_period('M')
# fit model
model = ARIMA(series, order=(5, 1, 0))
model_fit = model.fit()
# summary of fit model
print(model_fit.summary())
# line plot of residuals
residuals = DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
# density plot of residuals
residuals.plot(kind='kde')
pyplot.show()
# summary stats of residuals
print(residuals.describe())
```

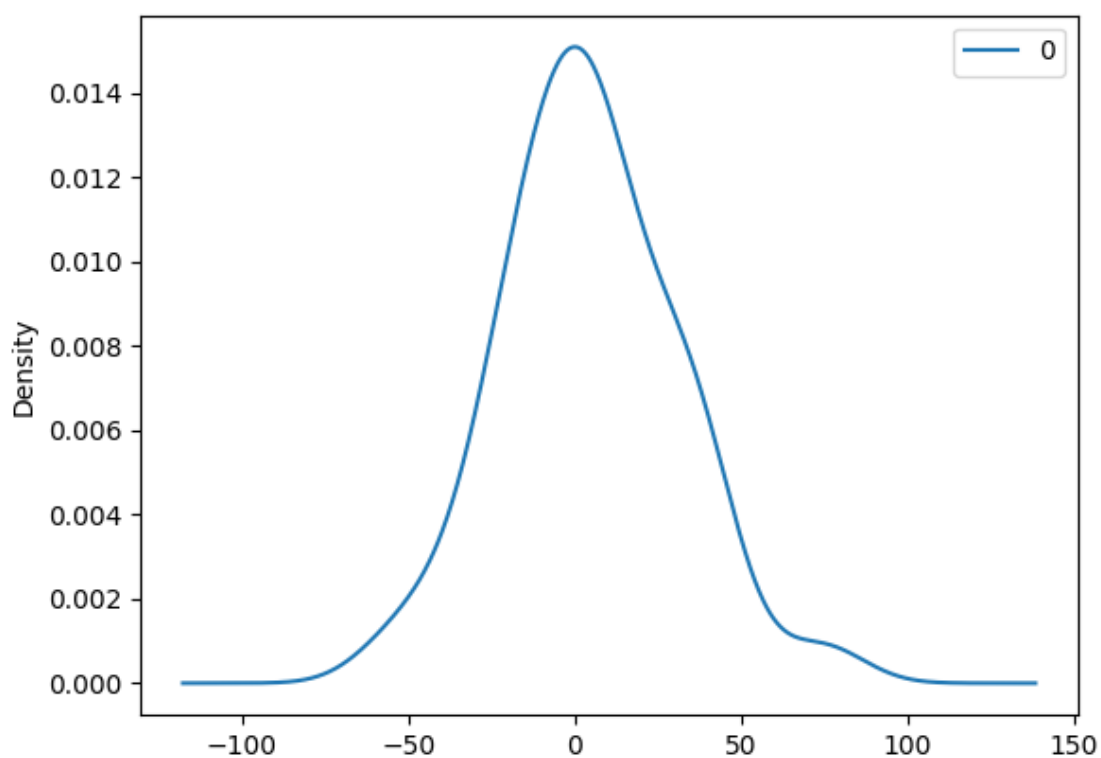


Figure 1: img2.png

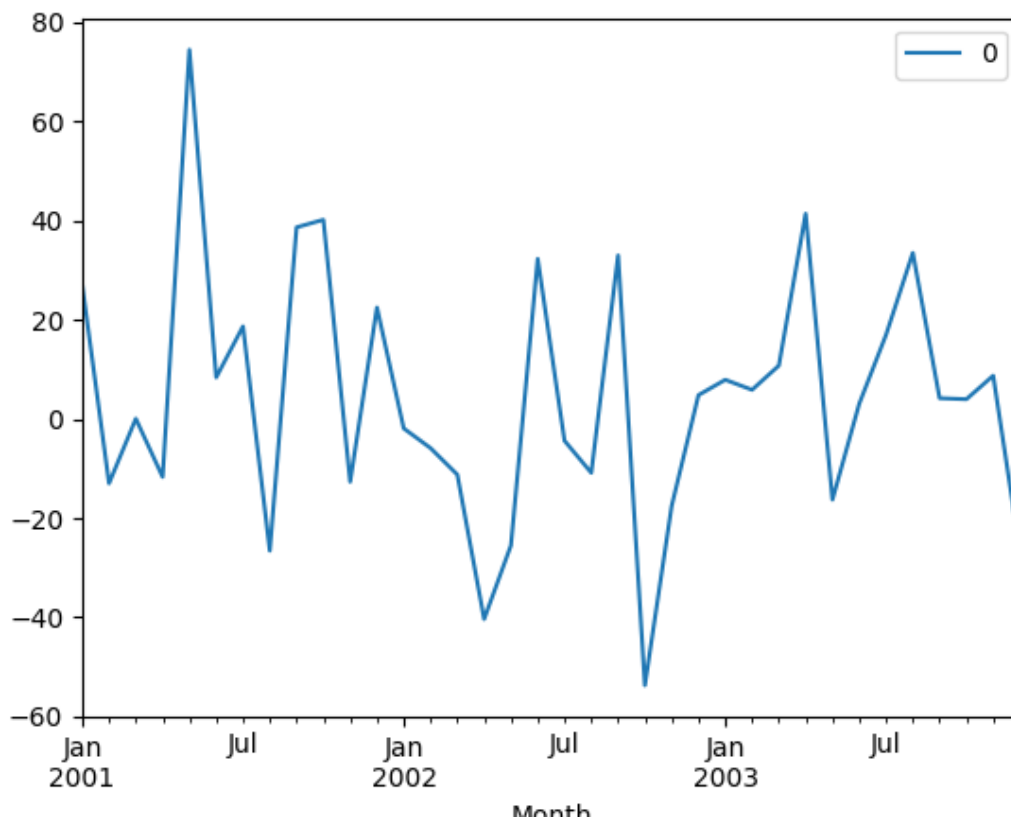


Figure 2: img3.png

2.3.3 Rolling Forecast ARIMA Model

Given the reliance on observations in previous time steps for differencing and the AR model, a rolling forecast is required. The ARIMA model is re-created after each new observation, which is a crude way to perform this rolling forecast.

Each iteration, we manually keep track of all observations in a list called history, which is seeded with the training data and to which new observations are appended.

Putting it all together, here's an example of a rolling forecast in Python using the ARIMA model.

Example is Formatted from [1]

```
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
from statsmodels.tsa.arima.model import ARIMA
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt

# load dataset
def parser(x):
    return datetime.strptime('190' + x, '%Y-%m')

series = read_csv('https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/
    ↪ main/time-series-prediction/temperature2.csv',
                  header=0, index_col=0, parse_dates=True, squeeze=True, date_parser
    ↪ =parser)
series.index = series.index.to_period('M')
# split into train and test sets
X = series.values
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
# walk-forward validation
for tes in range(len(test)):
    model = ARIMA(history, order=(6, 2, 0))
    model_fit = model.fit()
    output = model_fit.forecast()
    predictions.append(output[0])
    obs = test[tes]
    history.append(obs)
    print('predicted=%f, expected=%f' % (output[0], obs))
# evaluate forecasts
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
# plot forecasts against actual outcomes
pyplot.plot(test)
pyplot.plot(predictions, color='purple')
pyplot.show()
```

above is a line plot is created showing the expected values (blue) compared to the rolling forecast predictions (purple)

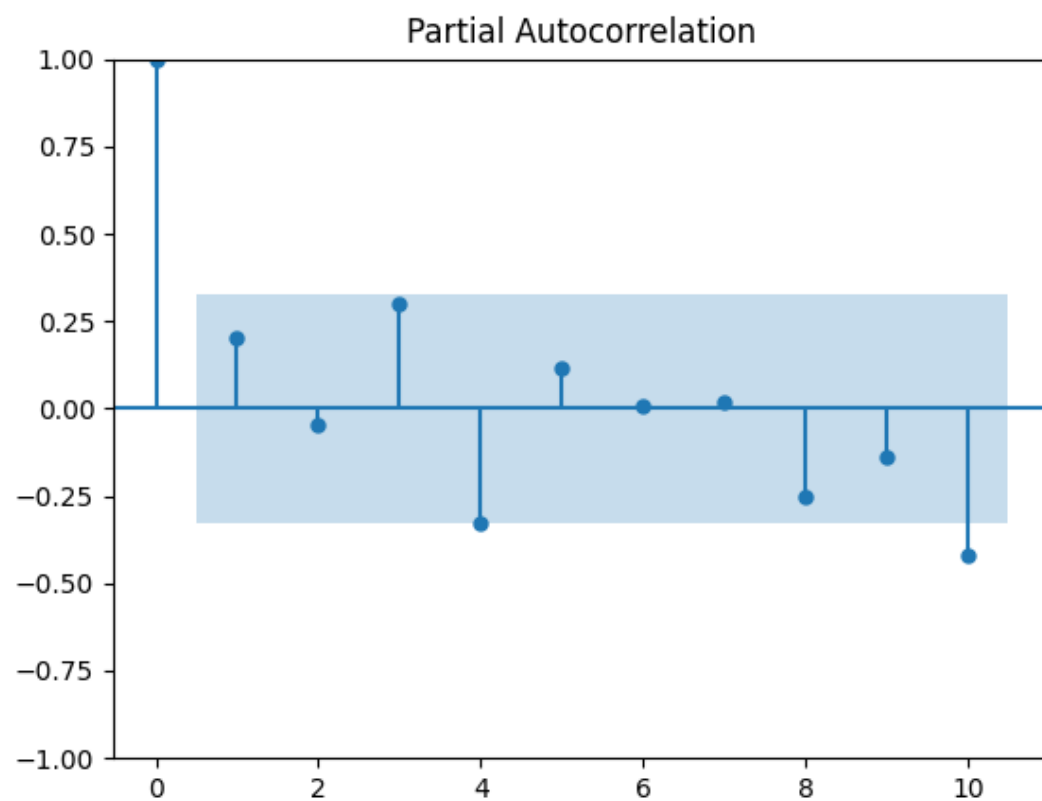


Figure 3: img1.png

Test the following codes below:

- <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/arma/arma-prediction.py>
- <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/arma/rolling-forecast-arma.py>

2.3.4 Dataset

- <https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/main/time-series-prediction/temperature2.csv>

2.3.5 References

- [1] Create an ARIMA Model for Time Series <https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/>
- [2] ARIMA Model – Complete Guide to Time Series Forecasting in Python <https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/#:~:text=ARIMA%2C%20short%20for%20'AutoRegressive%20Integrated,to%20predict%20the%20future%20values.>

2.4 Autocorrelation in images/Time Series

Extensive documents on autocorrelation of Time Series are available at

- <https://www.influxdata.com/blog/autocorrelation-in-time-series-data/#:~:text=The%20term%20autocorrelation%20refers%20to,you%20may%20have%20access%20to.>
- <https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/>

The degree of similarity between a given time series and a lagged version of itself over successive time intervals is referred to as autocorrelation. In other words, autocorrelation is used to assess the relationship between a variable's current value and any previous values to which you have access.

Example Formatted from [2]

```
from pandas import read_csv
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_acf
series = read_csv('https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/
↪ main/time-series-prediction/temperature2.csv', header=0, index_col=0)
plot_acf(series)
pyplot.show()
```

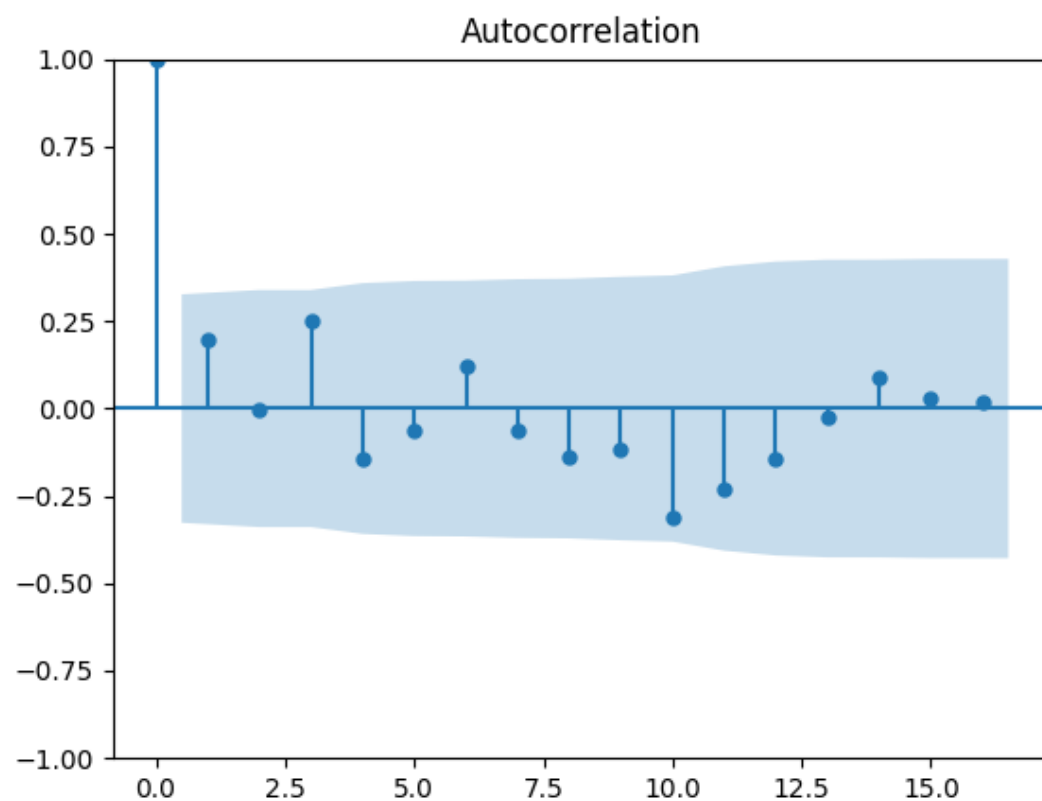


Figure 4: img.png

2.4.1 Partial Autocorrelation Function

partial correlation unlike autocorrelation the partial auto correlation deals with the relationship between an observation presently and a previous observation not taking into consideration the middle observation. the middle observation depends on the lag value it the value tell us the amount of time period to be delayed.

Example Formatted from [2]

```
from pandas import read_csv
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_pacf
temp = read_csv('https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/
    ↪ main/time-series-prediction/temperature2.csv', header=0, index_col=0)
plot_pacf(temp, lags=10)
pyplot.show()
```

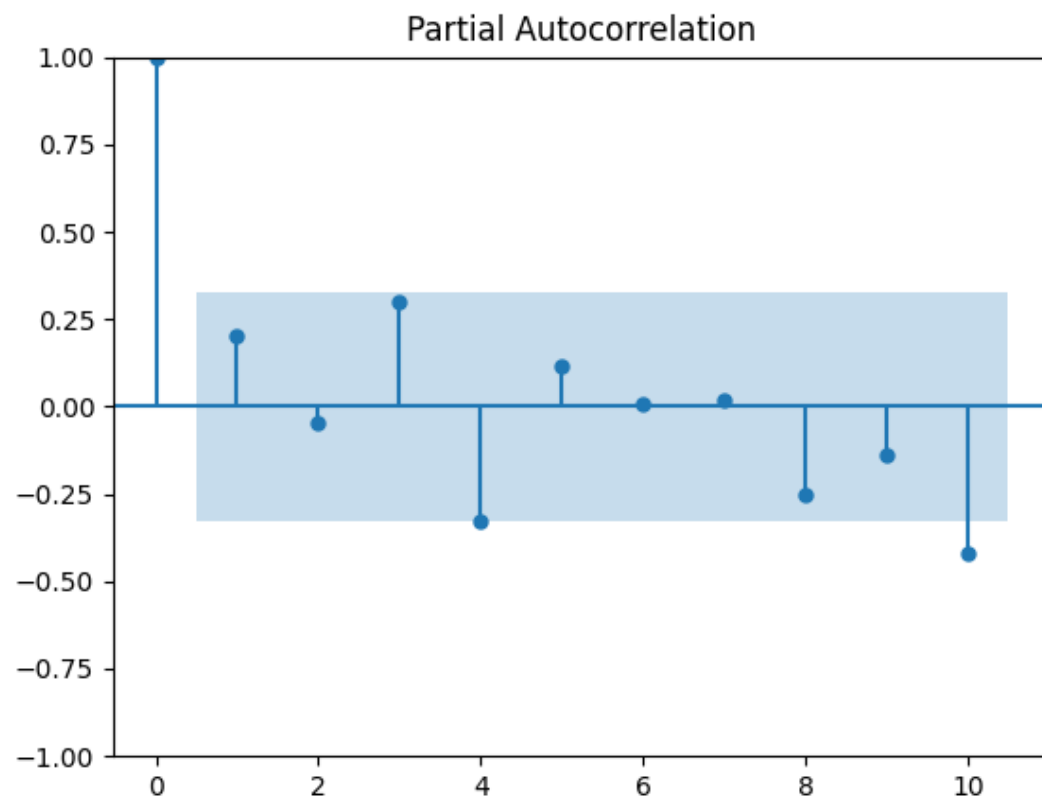


Figure 5: img1.png

Test the following codes below:

- AutoCorrelation <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/autocorrelation/autocorrelation.py>
- Partial AutoCorrelation <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/autocorrelation/partialautocorrelation.py>

2.4.2 Reference

- [1] <https://www.influxdata.com/blog/autocorrelation-in-time-series-data/#:~:text=The%20term%20autocorrelation%20refers%20to,you%20may%20have%20access%20to.>
- [2] <https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/>

2.5 Detrend a Time Series

Extensive documents on Detrend of Time Series are available at

- <https://www.statology.org/detrend-data/>

Detrending time series data means removing an underlying trend from the data. The main reason for doing so is to make seasonal or cyclical sub trends in the data more visible.

Time series data can be detrended using two methods:

- Detrend by Differencing
- Detrend by Model Fitting

2.5.1 Detrend by Differencing

This method of Detrend time series deals with finding the difference between the observations the inputting it into a new data set.

Example:

```
observation = [10, 16, 18, 21, 28, 34]
```

To get the detrend data set values you will do :

```
detrend_dataset = [16-10, 18-16, 21-18, 28-21, 34-28]
```

```
detrend_dataset = [6, 2, 3, 7, 6]
```

2.5.2 Detrend by Model Fitting

The detrend by model Fitting is Fitting a regression model to the data and then calculating the difference between the observed and predicted values from the model. Example:

```
observation = [10, 16, 18, 21, 28, 34]
```

```
predicted_observation = [9, 17, 20, 26, 32, 38]
```

```
Detrend_dataset = observation - predicted_observation
```

```
detrend_dataset = [10-9, 16-17, 18-20, 21-26, 28-32, 34-38]
```

```
detrend_dataset = [1, -1, -2, -5, -4, -4]
```

2.5.3 Reference

- [1] <https://www.statology.org/detrend-data/>

2.6 Granger casualty test

Extensive documents on granger casualty test are available at

- <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- http://www.scholarpedia.org/article/Granger_causality#:~:text=Granger%20causality%20is%20a%20statistical,values%20of%20X2%20alone.

The Granger causality test is used to determine whether one time series can be used to predict another.”Granger causality is a statistical concept of causality that is based on prediction. According to Granger causality, if a signal

$$X_1$$

“Granger-causes” (or “G-causes”) a signal

$$X_2$$

, then past values of

$$X_1$$

should contain information that helps predict

$$X_2$$

above and beyond the information contained in past values of

$$X_2$$

alone.”

Example formatted From [1]

```
# Import Libraries
import pandas as pd
from statsmodels.tsa.stattools import grangercausalitytests

# Read the CSV file
df = pd.read_csv('https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/
    ↪ main/time-series-prediction/temp.csv', parse_dates=['date'])
df['month'] = df.date.dt.month
```

```
grangercausalitytests(df[['temperature', 'month']], maxlag=2)
```

The Max lag value give the granger causality test for the time period apart which in these case its (2). so the code above check if the previous time period which is (1) then use it to predict the next time period which is (2) it keeps going like that depending on the max lag value

2.6.1 Output

```
Granger Causality
number of lags (no zero) 1
ssr based F test:      F=2.7089 , p=0.1237 , df_denom=13, df_num=1
ssr based chi2 test:   chi2=3.3340 , p=0.0679 , df=1
likelihood ratio test: chi2=3.0285 , p=0.0818 , df=1
parameter F test:      F=2.7089 , p=0.1237 , df_denom=13, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:      F=1.4529 , p=0.2793 , df_denom=10, df_num=2
ssr based chi2 test:   chi2=4.3588 , p=0.1131 , df=2
likelihood ratio test: chi2=3.8265 , p=0.1476 , df=2
parameter F test:      F=1.4529 , p=0.2793 , df_denom=10, df_num=2
```

Test the following codes below:

- <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/granger/granger.py>

2.6.2 Reference

- [1] <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- [2] http://www.scholarpedia.org/article/Granger_causality#:~:text=Granger%20causality%20is%20a%20statistical,values%20of%20X%20alone.

2.7 Lag Plot

Extensive Documents on lag plot can be found

- <https://www.geeksforgeeks.org/lag-plots>

A lag plot is a type of scatter plot in which the X-axis represents the dataset behind or ahead of the Y-axis by some time units. The difference in time units is known as lag or lagged, and it is represented by the symbol k .

The following axes are included in the lag plot:

Vertical axis:

$$y_i$$

for all i

Horizontal axis:

$$y_{i-k}$$

for all i where k represents the lag value.

The lag plot can be used in answering the following questions:

2.7.1 Model Distribution

Model Distribution means deciding the shape of the data on the lag plot.

2.7.2 Outliers

Outliers are data points that can be regarded as the extreme value in the distribution of the lag plot

2.7.3 Randomness in data

lag plot also shows the randomness of data if the plot seems random it will reflect in the lag plot.

2.7.4 Seasonality

If there is seasonality present in the lag-plot, it will give a periodic lag plot.

2.7.5 Autocorrelation

When the lag plot produces a linear plot, it indicates that autocorrelation exists in the data; whether it is positive or negative depends on the slope of the dataset's line.

Example Formatted from [1]


```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats as sc

# Sine graph and lag plot
time = np.arange(0, 10, 0.1);
amplitude = np.sin(time)
fig, ax = plt.subplots(1, 2, figsize=(12, 7))
ax[0].plot(time, amplitude)
ax[0].set_xlabel('Time')
ax[0].set_ylabel('Amplitude')
ax[0].axhline(y=0, color='k')
amplitude_series = pd.Series(amplitude)
pd.plotting.lag_plot(amplitude_series, lag=3, ax=ax[1])
plt.show()

# Random and Lag Plot
sample_size = 100
fig, ax = plt.subplots(1, 2, figsize=(12, 7))
random_series = pd.Series(np.random.normal(size=sample_size))
random = random_series.reset_index(inplace=True)
ax[0].plot(random['index'], random[0])
pd.plotting.lag_plot(random[0], lag=1)
plt.show()

temp_data = pd.read_csv('https://raw.githubusercontent.com/cybertraining-dsc/su22-
    ↪ reu-385/main/time-series-prediction/temperature2.csv')
temp_data.reset_index(inplace=True)
fig, ax = plt.subplots(1, 2, figsize=(12, 7))
ax[0].plot(temp_data['Adj Close'], temp_data['index'])
pd.plotting.lag_plot(temp_data['Adj Close'], lag=1, ax=ax[1])
plt.show()
```

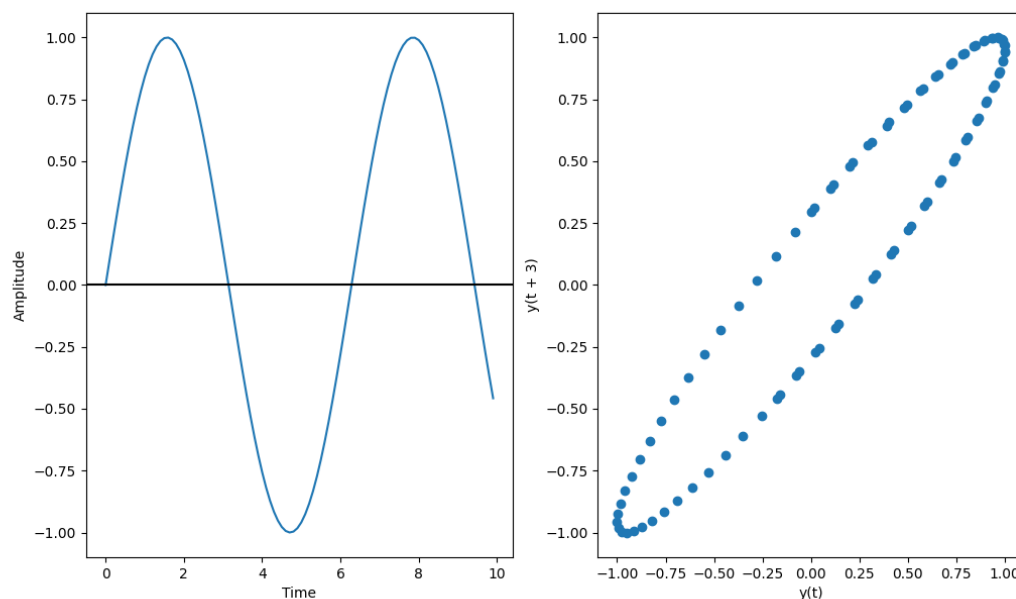


Figure 6: CAPTION MISSING

Figure Figure 6 shows ... A SENTENCE IS MISSING THAT REFERS TO THIS IMAGE AND EXPLAINS WHT IT SHOWS. ALL IMAGES INCLUDED IN PAPER MUST HAVE SUCH A SENTENCE.

To test the following code:

- <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/lag-plot/lag-plot.py>

2.7.6 Reference

- [1] <https://www.geeksforgeeks.org/lag-plots>

2.8 Sample Entropy

Extensive documents on Sample Entropy are available at

- <https://pythonmana.com/2022/130/202205101057060253.html>
- <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- <https://sampen.readthedocs.io/en/stable/>

Approximate entropy a technique for quantifying the amount of regularity and unpredictability in time-series data. so what is sample Entropy? “Sample Entropy is similar to approximate entropy but is more consistent in estimating the complexity even for smaller time series. For example, a random time series with fewer data points can have a lower ‘approximate entropy’ than a more ‘regular’ time series, whereas, a longer random time series will have a higher ‘approximate entropy’”

to use the library you will install using `pip`

```
$ pip install sampen
```

2.8.1 Python Example

Example Formatted from [3]

```
from sampen import sampen2, normalize_data

# initialize a list
series_data = []

# open the file and read each line into the list
with open('temp3.txt', 'r') as file:
    for row in file:
        series_data.append(float(row.strip(' \t\n\r')))

# calculate the sample entropy
sampen_of_series = sampen2(normalize_data(series_data))
print(sampen_of_series)
```

2.8.2 Output

```
[(0, 2.196817997610929, 0.002684778756853663),
 (1, 2.2248168592127824, 0.004639787747652105),
 (2, 2.1972245773362196, 0.007540128072706757)]
```

2.8.2.1 Output Breakdown

```
# Epoch length for max epoch
2,

# SampEn
```

```
2.1972245773362196
# Standard Deviation
0.007540128072706757
```

To test the following code:

- <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/sample-entropy/sample-entropy.py>

2.8.3 Reference

- [1] <https://pythonmana.com/2022/130/202205101057060253.html>
- [2] <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- [3] <https://sampen.readthedocs.io/en/stable/>

2.8.3.1 Dataset:

- <https://www.physionet.org/content/sampen/1.0.0/c/sampentest.txt>

2.9 Seasonal decomposition from stats model

An extensive documentation on seasonal decomposition and deseasonalising of stat model is available at

- <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- <https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/#:~:text=The%20statsmodels%20library%20provides%20an,model%20is%20additive%20or%20multiplicative.>

Decomposing a time series entails viewing it as a collection of level, trend, seasonality, and noise components. it uses a function called `seasonal_decompose()`, the statsmodels library implements the naive, or classical, decomposition method. You must specify whether the model is additive or multiplicative.

2.9.1 Additive Decomposition

An additive model deals with components being added together as follows:

$$y(t) = Level + Trend + Seasonality + Noise$$

We can decompose a time series that are made up of a linearly increasing trend from 1 to 99, and random noise as an additive model.

Example Formatted from [1]

```
from random import randrange
from matplotlib import pyplot
from statsmodels.tsa.seasonal import seasonal_decompose
data = [i+randrange(15) for i in range(5,90)]
result = seasonal_decompose(data, model='additive', period=3)
result.plot()
pyplot.show()
```

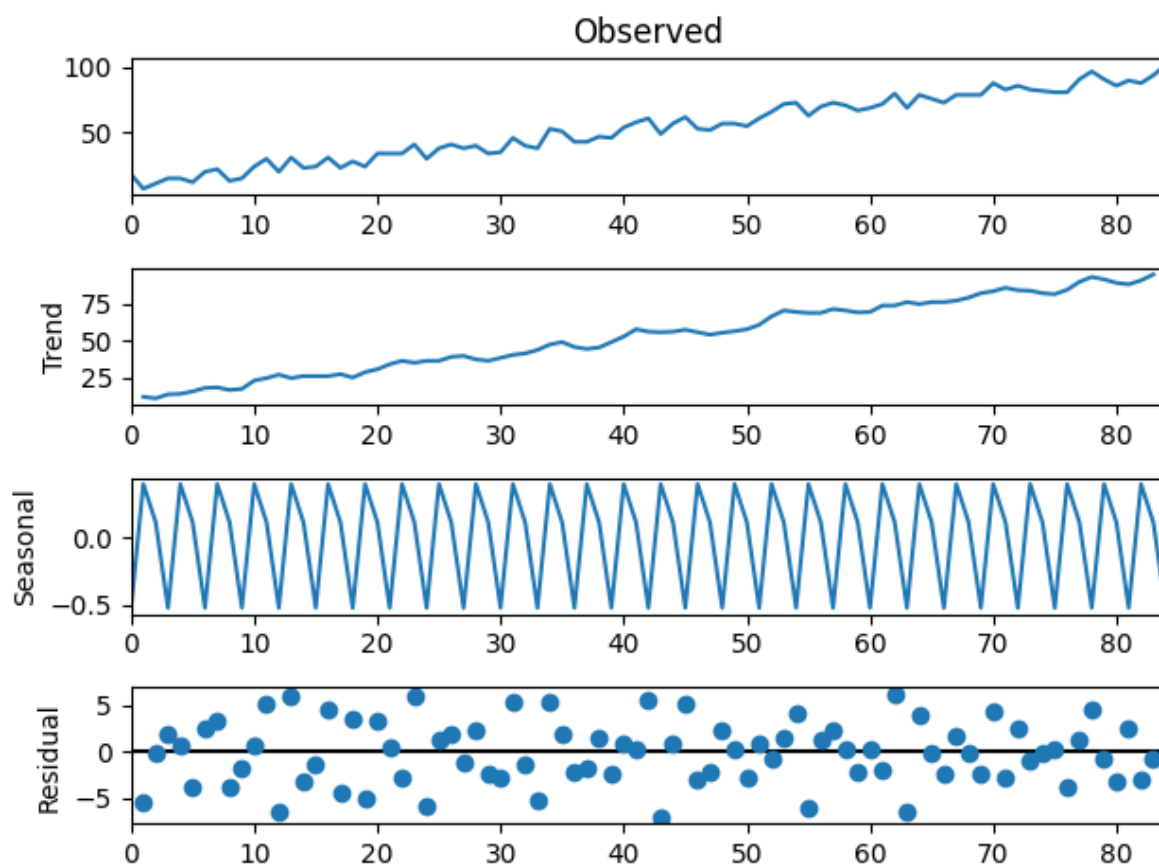


Figure 7: TPODP: CAPTION MISSING

2.9.2 Multiplicative Decomposition

A multiplicative model deals with components being multiplied together as follows:

$$y(t) = Level * Trend * Seasonality * Noise$$

Nonlinear models, include are quadratic or exponential, are multiplicative. Changes grow or shrink over time. A curved line represents a nonlinear trend. The frequency and/or amplitude of a non-linear seasonality increase or decrease over time.

We can create a quadratic time series by using the square of the time step from 1 to 99 and decomposing it using a multiplicative model.

Example Formatted from [1]

```
from matplotlib import pyplot
from statsmodels.tsa.seasonal import seasonal_decompose

data = [i ** 4.0 for i in range(10, 150)]
result = seasonal_decompose(data, model='multiplicative', period=2)
result.plot()
pyplot.show()
```

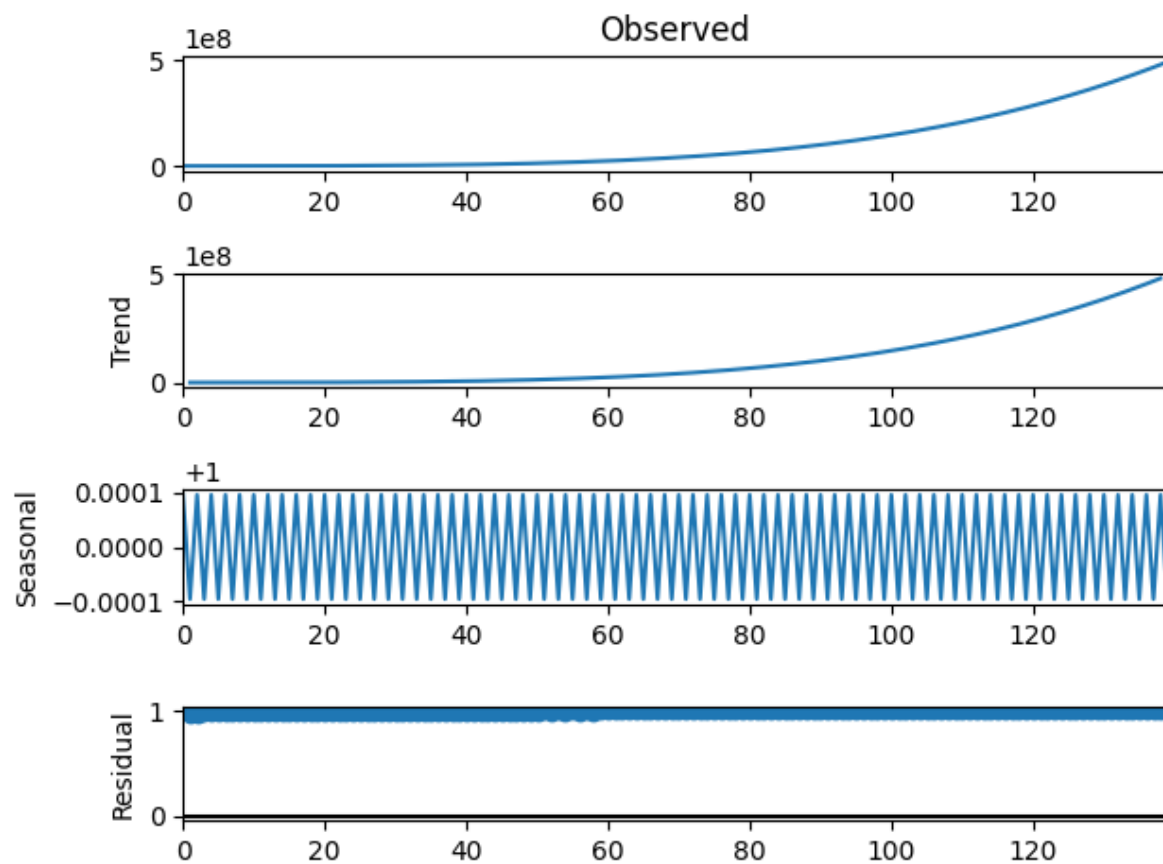


Figure 8: TPODP: CAPTION MISSING

2.9.3 Desasonalising of Time Series

This is process of removing seasonality from time series

There are several methods for deseasonalising a time series. Here are a few :

- Take a moving average with length as the seasonal window. This will smoothen in series in the process.
- Seasonal difference the series (subtract the value of previous season from the current value)
- Divide the series by the seasonal index obtained from STL decomposition

Example Formatted from [1],[2]

```
from matplotlib import pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
df = [i ** 4.0 for i in range(10, 1000)]

# Time Series Decomposition
result_mul = seasonal_decompose(df, model='multiplicative', period=60)

# Deseasonalize
deseasonalized = df / result_mul.seasonal

# Plot
plt.plot(deseasonalized)
plt.title('Data Deseasonalized', fontsize=15)
plt.show()
```

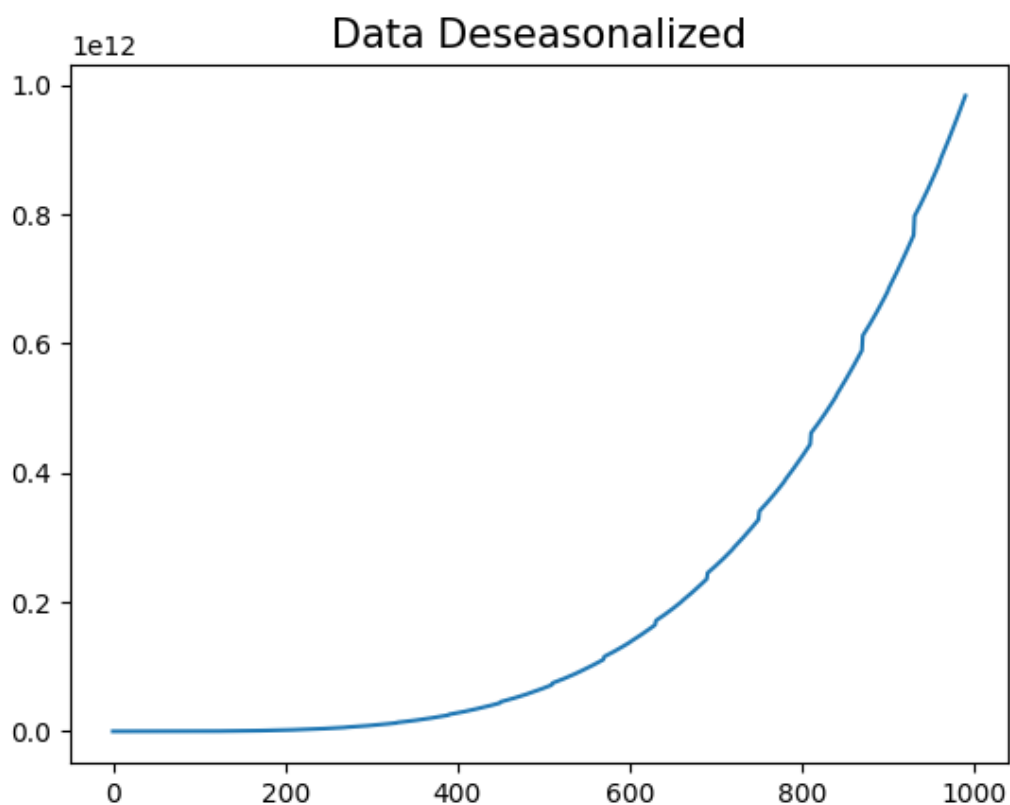


Figure 9: TPODP: CAPTION MISSING

Test the following codes below:

- Additive Decomposition <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/seasonal-decomposition/additive-decomposition.py>
- Multiplicative Decomposition <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/seasonal-decomposition/multiplicative-decomposition.py>
- Deseasonalize <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/seasonal-decomposition/deseasonalize.py>

2.9.4 References

- [1] How to Decompose Time Series Data into Trend and Seasonality <https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/#:~:text=The%20statsmodels%20library%20provides%20an,model%20is%20additive%20or%20multiplicative.>
- [2] <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>

2.10 Stationary and Non-Stationary Time Series

Extensive documents on stationary and Non-Stationary Time Series are available at

- <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- <https://www.investopedia.com/articles/trading/07/stationary.asp#:~:text=Non%2DStationary%20Processes-,Non%2DStationary%20Time%20Series%20Data,cannot%20be%20modeled%20or%20forecasted.>
- <https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/>
- <https://www.geeksforgeeks.org/how-to-check-if-time-series-data-is-stationary-with-python/>
- https://www.statsmodels.org/devel/examples/notebooks/generated/stationarity_detrending_adf_kpss.html

2.10.1 Stationary Time series

A stationary time series has statistical properties or moments that do not change over time (for example, mean and variance).

The series' statistical properties, such as mean, variance, and auto correlation, remain constant over time. The auto correlation of a series is simply the correlation of the series with its previous values.

2.10.1.1 How to make a time series stationary? You can make series stationary by:

- Differencing the Series (once or more)
- Take the log of the series
- Take the nth root of the series
- Combination of the above

The most common and convenient method is Differencing the series

2.10.1.1.1 Differencing the Series (once or more) If

$$y_t$$

is the value at time ‘

$$t$$

,’ the first difference of

$$y$$

equals

$$y_t - y_{t-1}$$

. To put it simply, differencing the series is simply subtracting the next value from the current value.

If the first difference does not make a series stationary, the second differencing can be used. And so forth.

- Consider the following sequence: [8, 6, 4, 10, 25] .
- First differencing gives: [6-8, 4-6, 10-4, 25-10] = [-2, -2, 6, 15]
- Second differencing gives: [-2-(-2), 6-(-2), 15-6] = [0, 8, 9] ## Non-Stationary Time series

Data points are frequently non-stationary or have changing means, variances, and covariances. Trends, cycles, random walks, and combinations of the three are examples of non-stationary behaviors.

While forecasting, it is recommended to convert a non-stationary series to a stationary series because autoregressive forecasting models are essentially linear regression models that use the lag(s) of the series itself as predictors.

2.10.2 How to test for stationary?

A method for determining whether a given series is stationary. This is possible with statistical tests known as ‘Unit Root Tests.’ There are several variants of this in which the tests determine whether a

time series is non-stationary and has a unit root.

Unit Root tests are implemented in a variety of ways, which includes:

- Visual Test
- Statistical test
- Kwiatkowski-Phillips-Schmidt-Shin – KPSS test (trend stationary)

2.10.2.1 Visual Test simply by looking at each plot, being able to identify the series in which the mean and variance changed over time Similarly, we can plot the data to see if the properties of the series change over time.

```
from pandas import read_csv
from matplotlib import pyplot
data = read_csv('https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/
    ↪ main/time-series-prediction/temperature2.csv', header=0, index_col=0)
data.plot()

pyplot.show()
```

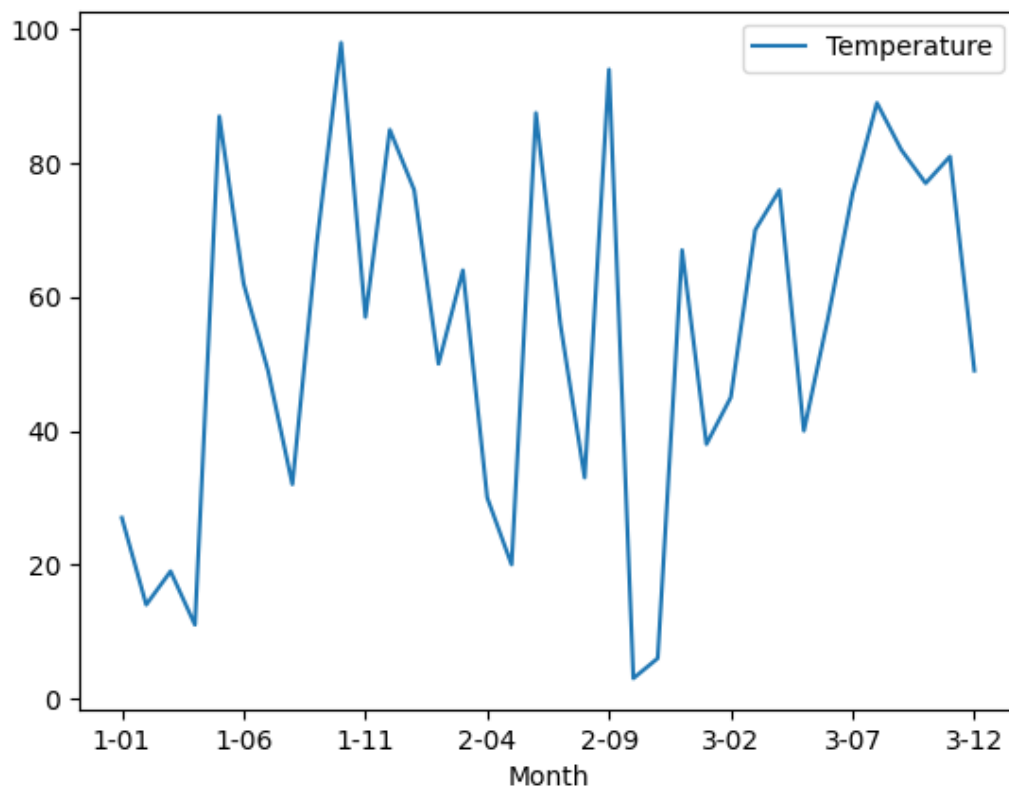


Figure 10: TODO CAPTION MISSING

2.10.2.2 Statistical test Statistical tests such as the unit root stationary tests can be used. The presence of a unit root indicates that the statistical properties of a given series are not constant over time, which is required for stationary time series. The following is a mathematical explanation:

Suppose we have a time series

$$y_t = a * y_{t-1} + \epsilon_t$$

where y_t is the value at the time instant

t

and

ϵ_t

is the error term. In order to calculate y_t we need the value of

$$y_{t-1}$$

, which is

$$y_{t-1} = a * y_{t-2} + \epsilon_{t-1}$$

If we do that for all observations, the value of y_t will come out to be:

$$y_t = a^n \times y_{t-n} + \sum \epsilon_t - i \times a^i$$

2.10.2.3 Augmented Dickey Fuller test (ADH Test) The Dickey-Fuller test is a well-known statistical test. It can be used to determine whether a series contains a unit root, and thus whether the series is stationary. This test's null and alternate hypotheses are as follows:

Example Formatted from [4]

```
# import python pandas package
import pandas as pd

# import the adfuller function from statsmodel
# package to perform ADF test
from statsmodels.tsa.stattools import adfuller

# read the dataset using pandas read_csv() function
data = pd.read_csv(
    "https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/main/time-
    ↪ series-prediction/temperature2.csv",
    header=0, index_col=0)

# extracting only the passengers count using values function
value = data.values

# passing the extracted passengers count to adfuller function.
# result of adfuller function is stored in a res variable
res = adfuller(value)

# Printing the statistical result of the adfuller test
print('Augmented Dickey-Fuller Statistic: %f' % res[0])
print('p-value: %f' % res[1])

# printing the critical values at different alpha levels.
```

```
print('critical values at different levels:')
for l, m in res[4].items():
    print('\t%s: %.3f' % (l, m))
```

- Null Hypothesis: The series has a unit root (value of `a=1`)
- Alternate Hypothesis: The series has no unit root.

2.10.2.4 Kwiatkowski-Phillips-Schmidt-Shin – KPSS test KPSS is yet another test for determining a time series' stationarity (slightly less popular than the Dickey-Fuller test). The null and alternate hypotheses for the KPSS test are the inverse of those for the ADF test, which frequently leads to confusion. The KPSS test's authors defined the null hypothesis as the process being trend stationary, as opposed to an alternate hypothesis of a unit root series.

Example Formatted from [5]

```
import pandas as pd
from statsmodels.tsa.stattools import kpss

data = pd.read_csv(
    "https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/main/time-
    ↪ series-prediction/temperature2.csv",
    header=0, index_col=0)
print("Results of KPSS Test:")
kpsstest = kpss(data, regression="c", nlags="auto")
kpss_output = pd.Series(
    kpsstest[0:3], index=["Test Statistic", "p-value", "Lags Used"]
)
for num, test in kpsstest[3].items():
    kpss_output["Critical Value (%s)" % num] = test
print(kpss_output)
```

- Null Hypothesis: The process is trend stationary.
- Alternate Hypothesis: The series has a unit root (series is not stationary).

2.10.3 Types of Stationary Time series

2.10.3.1 Strict Stationary A series that has no unit root but exhibits a trend is referred to as a trend stationary series. Once the trend is removed, the resulting series will be strict stationary. The KPSS test classifies a series as stationary on the absence of unit root. This means that the series can be strict stationary or trend stationary.

2.10.3.2 Trend Stationary A trend stationary series is one that does not have a unit root but exhibits a trend. The resulting series will be strictly stationary once the trend is removed. The KPSS test determines whether a series is stationary based on the absence of a unit root. As a result, the series can be either strict stationary or trend stationary.

2.10.3.3 Difference Stationary A difference stationary time series is one that can be made strict stationary by differencing. ADF tests are also referred to as difference Stationary tests.

Test the following code below:

- Augmented Dickey Fuller test (ADH Test) <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/stationary-and-nonstationary-timeseries/adh-test.py>
- Visual Test <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/stationary-and-nonstationary-timeseries/visual.py>
- Kwiatkowski-Phillips-Schmidt-Shin – KPSS test <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/stationary-and-nonstationary-timeseries/kpss.py>

2.10.4 References

- [1] Time Series Analysis in Python <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- [2] Intro to Stationary and Non-Stationary Processes <https://www.investopedia.com/articles/trading/07/stationary.asp#:~:text=Non%2DStationary%20Processes-,Non%2DStationary%20Time%20Series%20Data,cannot%20be%20modeled%20or%20forecasted.>
- [3] Methods to Check Stationary <https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/>
- [4] How to Check if Time Series Data is Stationary with Python? <https://www.geeksforgeeks.org/how-to-check-if-time-series-data-is-stationary-with-python/>
- [5] Stationarity and detrending (ADF/KPSS) https://www.statsmodels.org/devel/examples/notebooks/generated/stationarity_detrending_adf_kpss.html

2.11 Smoothen

Extensive documents on Smoothen are available at

- <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>

- <https://machinelearningmastery.com/moving-average-smoothing-for-time-series-forecasting-python/#:~:text=Smoothing%20is%20a%20technique%20applied,of%20the%20underlying%20causal%20processes.>

Smoothing is a technique used to remove fine-grained variation between time steps in time series. So, how do you smooth out a series? Let us now look at the following method.

- Take a Moving Average
- Do a LOESS smoothing (Localized Regression)
- Do a LOWESS smoothing (Locally Weighted Regression)

2.11.1 Take a Moving Average

A moving average is simply the average of a fixed-width rolling window. However, you must choose the window width carefully because a large window will over-smooth the series.

2.11.2 LOESS smoothing (Localized Regression)

LOESS, which stands for 'LocalizeD regrESSion,' fits multiple regressions in each point's local neighborhood. It is implemented in the `statsmodels` package, and the degree of smoothing can be controlled using the `frac` argument.

Example Formatted from [2]

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.nonparametric.smoothers_lowess import lowess

plt.rcParams.update({'xtick.bottom': False, 'axes.titlepad': 5})

# Import
df_orig = pd.read_csv(
    'https://raw.githubusercontent.com/cybertraining-dsc/su22-reu-385/main/time-
    ↪ series-prediction/temperature2.csv',
    parse_dates=['Month'], index_col='Month')

# 1. Moving Average
df_ma = df_orig.Temperature.rolling(3, center=True, closed='both').mean()

# 2. Loess Smoothing (5%)
```



```
df_loess_5 = pd.DataFrame(lowess(df_orig.Temperature, np.arange(len(df_orig.  
    ↪ Temperature))), frac=0.05)[: , 1],  
                           index=df_orig.index, columns=['Temperature'])  
  
# Plot  
fig, axes = plt.subplots(3, 1, figsize=(7, 7), sharex=True, dpi=120)  
df_orig['Temperature'].plot(ax=axes[0], color='k', title='Original Series')  
df_loess_5['Temperature'].plot(ax=axes[1], title='Loess Smoothed 5%')  
df_ma.plot(ax=axes[2], title='Moving Average (3)')  
fig.suptitle('How to Smoothen a Time Series', y=0.95, fontsize=14)  
plt.show()
```



Figure 11: TODO CAPTION MISSING

Test the following code below:

- <https://github.com/cybertraining-dsc/su22-reu-385/blob/main/time-series-prediction/smoothen/smoothen.py>

2.11.3 Reference

- [1] <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- [2] <https://machinelearningmastery.com/moving-average-smoothing-for-time-series-forecasting-python/#:~:text=Smoothing%20is%20a%20technique%20applied,of%20the%20underlying%20causal%20processes.>

2.12 Temporal Fusion Transformer(TFT)

An extensive documentation on Temporal Fusion Transformer are Available at

- <https://arxiv.org/abs/1912.09363>
- <https://towardsdatascience.com/temporal-fusion-transformer-googles-model-for-interpretable-time-series-forecasting-5aa17beb621>

TFT is an attention-based Deep Neural Network that has been optimized for performance and interpretability. To learn temporal dependencies, the TFT employs recurrent layers for local processing and interpretable self-attention layers. relationships at different scales TFT also employs specialized components.

2.12.1 Advantages of Temporal Fusion Transformer

2.12.1.1 Rich features The Temporal Fusion Transformer support 3 feature which includes: temporal data with known inputs into the future,temporal data known only up to the present, and the time-invariant features

2.12.1.2 Heterogeneous time series The TFT architecture split processing into two part: Local processing focuses on specific event characteristics, whereas global processing captures the collective characteristics of all time series.

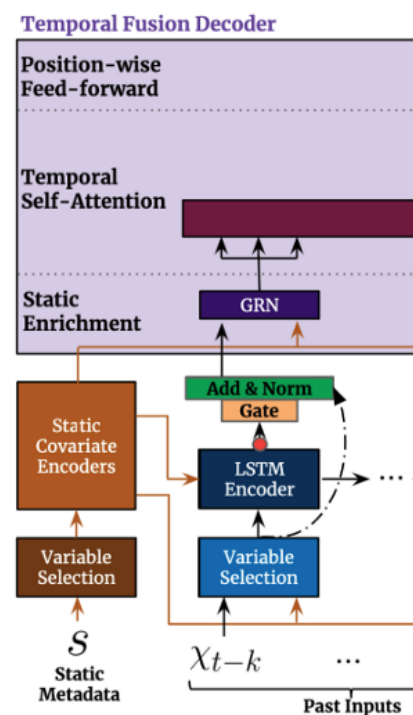
2.12.1.3 Multi-horizon forecasting TFT produces prediction intervals in addition to the actual prediction by utilizing the quantile loss function.

2.12.1.4 Interpretability The TFT a transformers based architecture. This model presents a novel Muti Head attention mechanism that, when analyzed, provides additional insight on feature importance by utilizing self-attention.

2.12.1.5 High Performance TFT outperformed traditional statistical models (ARIMA) as well as DeepAR, MQRNN, and Deep Space-State Models (DSSM)

2.12.1.6 Documentation There are already open source TFT implementations in Tensorflow and Python.

Example from [2]



The Figure below shows the Top level Architecture of Temporal Fusion Transformer

Understanding the figure above:

TODO: verify the equations and use latex to write them

“For a given timestep t , a look back window k , and a τ_{max} step ahead window, where $t \in [t - k..t + \tau_{max}]$, the model takes as input: i) Observed past inputs x in the time period $[t - k..t]$, future known inputs x in the time period $[t + 1..t + \tau_{max}]$ and a set of static variables s (if exist). The target variable y also spans the time window $[t + 1..t + \tau_{max}]$ ”.

2.12.1.7 References

- [1] Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting <https://arxiv.org/abs/1912.09363>
- [2] <https://towardsdatascience.com/temporal-fusion-transformer-googles-model-for-interpretable-time-series-forecasting-5aa17beb621>

2.13 Time series in python

A large number of work had been done on time series in python. These are extensive set of links

2.13.1 Time Series prediction

- <https://www.tessellationtech.io/3-advantages-to-time-series-analysis-and-forecasting/>
- <https://www.tableau.com/learn/articles/time-series-forecasting#:~:text=Time%20series%20forecasting%20occurs%20when,drive%20future%20strategic%20decision%20making.>

2.13.2 Anova Time series

- <https://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/anova/>
- <https://medium.com/@stallonejacob/time-series-forecast-a-basic-introduction-using-pyt>
- <https://www.geeksforgeeks.org/how-to-perform-a-two-way-anova-in-python/>
- <https://www.geeksforgeeks.org/how-to-perform-a-one-way-anova-in-python/>

2.13.3 TFT

- <https://arxiv.org/abs/1912.09363>
- <https://towardsdatascience.com/temporal-fusion-transformer-googles-model-for-interpretable-time-series-forecasting-5aa17beb621>

2.13.4 ARIMA Time series

- <https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/>
- <https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/#:~:text=ARIMA%2C%20short%20for%20'AutoRegressive%20Integrated,to%20predict%20the%20future%20values.>

2.13.5 Seasonal Decomposition from StatsModel

- <https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/#:~:text=The%20statsmodels%20library%20provides%20an,model%20is%20additive%20or%20multiplicative.>

2.13.6 Stationary and Non-Stationary Time Series

- <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- <https://www.investopedia.com/articles/trading/07/stationary.asp#:~:text=Non%2DStationary%20Processes-,Non%2DStationary%20Time%20Series%20Data,cannot%20be%20modeled%20or%20forecasted.>
- <https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/>
- <https://www.geeksforgeeks.org/how-to-check-if-time-series-data-is-stationary-with-python/>
- https://www.statsmodels.org/devel/examples/notebooks/generated/stationarity_detrending_adf_kpss.html

2.13.7 AutoCorrelation

- <https://www.influxdata.com/blog/autocorrelation-in-time-series-data/#:~:text=The%20term%20autocorrelation%20refers%20to,you%20may%20have%20access%20to.>
- <https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/>

2.13.8 Lag Plot

- <https://www.geeksforgeeks.org/lag-plots/>

2.13.9 Smoothen

- <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
- <https://machinelearningmastery.com/moving-average-smoothing-for-time-series-forecasting-python/#:~:text=Smoothing%20is%20a%20technique%20applied,of%20the%20underlying%20causal%20processes.>

2.13.10 Granger casualty test

- <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>

3 REFERENCES

