Contents

1. Process Outline (Flowchart)

1. **Load the data.**

Part 1

2. **Pre-processing:** Creating timestamps, converting the datatype of date/time column, making the series univariate, etc.
3. **Check and make the series stationary:** Most time series methods assume the data is stationary. Therefore, we need to check that using tests like ADF and perform transformations like differencing to make the series stationary.
4. **Differencing (If needed):** The number of times a difference operation needs to be performed to make the series stationary is the **d** value.

Part 2

5. Univariate or Multivariate?

6. **Create ACF and PACF plots:** ACF and PACF plots are used to determine the input parameters **p and q** for our ARIMA model
7. **Determine the p and q values:** Read the values of p and q from the plots in the previous step.
8. **Fit the model of choice:** Using the processed data and parameter values we calculated from the previous steps to fit the ARIMA model.

Part 3

9. **Predict values on validation set:** Predict the future values.
10. **Calculate RMSE:** To check the performance of the model, check the RMSE value using the predictions and actual values on the validation set.

2. Preprocessing (Part 1)

Check Nulls

Check for null values in the dataset. The below command will show us how many null values are in each column of the dataset.

```
perthTemp.isnull().sum()
```

```
Year                              1
Month                             1
Day                               1
Minimum temperature (Degree C)    76
Maximum temperature (Degree C)    31
Rainfall amount (millimetres)     0
dtype: int64
```

Missing values

One of the simplest ways to fill missing values is to use the previous or next value. This method works well enough in most of the cases as time series data usually changes gradually. As long as there aren't multiple NULLs in a row, we can use backward or forward fills

```
perthTemp.fillna(method='ffill', inplace = True)
```

method*{'backfill', 'bfill', 'pad', 'ffill', None}, default None*

Method to use for filling holes in reindexed Series
pad / ffill: propagate last valid observation forward to next valid
backfill / bfill: use next valid observation to fill gap.

[1]

Timestamps

Parse Dates

Datasets come with dates in many different formats. It could be in a single column like in the first image below or it could be split into multiple columns.

date	Year	Month	Day
1944-06-03	1944.0	6.0	3.0
1944-06-04	1944.0	6.0	4.0
1944-06-05	1944.0	6.0	5.0
1944-06-06	1944.0	6.0	6.0

We can combine the 3 columns into a single column while reading the CSV.

```
test = pd.read_csv("data/PerthTemperatures.csv",
                   parse_dates=[['Year', 'Month', 'Day']])
```

This will combine the output

Year_Month_Day
1944 6 3
1944 6 4
1944 6 5
1944 6 6

Pandas usually reads date column as strings when reading the CSV file. That needs to be converted to datetime datatype. This can be done using the following code

```
perthTemp['Date'] = pd.to_datetime(perthTemp[['Year', 'Month',
                                              'Day']])
```

We can check if the date column is of the right data type using the code below

```
dataframe.dtypes
```

```
Minimum temperature (Degree C)          float64
Maximum temperature (Degree C)          float64
Rainfall amount (millimetres)           float64
Date                                datetime64[ns]
dtype: object
```

Date index

Setting the date time column as the index for the data frame makes it very convenient to handle.

We can split the dataset into train test sets based on the date and also performing any transformations or operations on the entire dataframe becomes easy when the date column in out of the way. We can do that using the code below.

```
perthTemp_2 = perthTemp.set_index('Date')
```

Date	Minimum temperature (Degree C)
1944-06-03	11.0
1944-06-04	12.2
1944-06-05	12.0
1944-06-06	7.4

Resample

Time series data can be hourly, daily, monthly, or even once every second or higher frequency.

```
perthTemp_Train_MS = perthTemp_Train.resample('MS').mean()
```

Train/Test Split

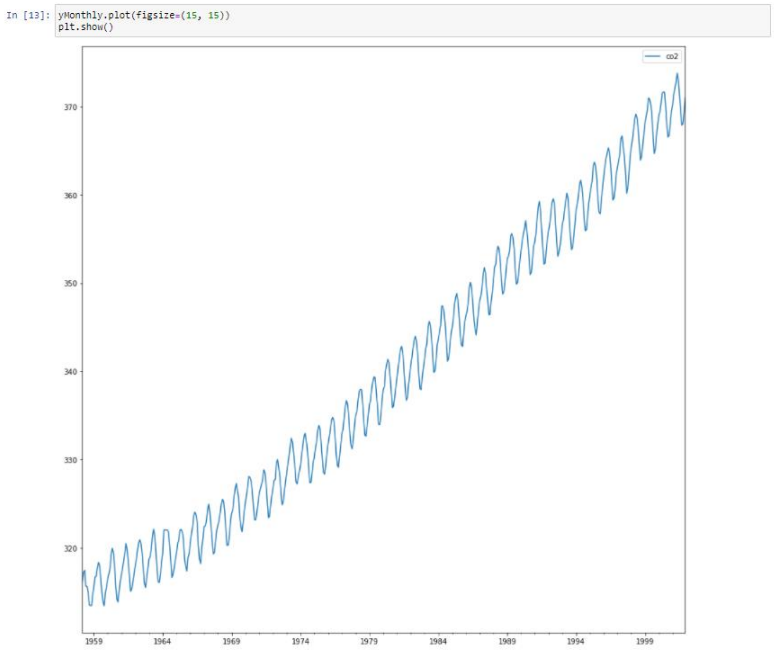
We can use the time index to easily split the dataset.

```
perthTemp_Train = perthTemp_2[:'2010-12-31']
perthTemp_Test = perthTemp_2['2011-01-01':]
```

3. Visualizing Data

The first step in Time Series Analysis is visualizing the data. A simple plot can give us the basic information.

```
dataframe.plot(figsize = (15, 15))
plt.show()
```



Stationarity Tests

A stationary time series is one whose statistical properties like mean and variance are constant over time. Most statistical forecasting methods assume that the time series data is stationary. Therefore, it is important to check if the data is non-stationary.

One of the most popular methods for checking stationarity is **Augmented Dickey-Fuller test.**

Null Hypothesis (H0): If failed to be rejected, it suggests the time series is non-stationary. It has some time dependent structure.

Alternate Hypothesis (H1): The null hypothesis is rejected; it suggests the time series is stationary. It does not have time-dependent structure.

We interpret this result using the **p-value** from the test. A p-value below a threshold (such as 5% or 1%) suggests we reject the null hypothesis (stationary), otherwise a p-value above the threshold suggests we fail to reject the null hypothesis (non-stationary).

Below code shows a function that accepts a time series and does a stationarity test

```python
def adf_test(ts, signif=0.05):
    dftest = adfuller(ts, autolag='AIC')
    adf = pd.Series(dftest[0:4], index=['Test Statistic','p-value','# Lags','# Observations'])

    for key,value in dftest[4].items():
       adf['Critical Value (%s)'%key] = value
    print (adf)

    p = adf['p-value']
    if p <= signif:
        print(f" Series is Stationary")
    else:
        print(f" Series is Non-Stationary")
```

Output of the test

```
Test Statistic            2.359810
p-value                   0.998990
# Lags                   14.000000
# Observations          511.000000
Critical Value (1%)      -3.443212
Critical Value (5%)      -2.867213
Critical Value (10%)     -2.569791
dtype: float64
 Series is Non-Stationary
```

How to make a time series stationary?

[2]

A time series can be thought of as a combination of **Level, Trend, Seasonality and Noise**

Level – The average value of the series

Trend - The increasing or decreasing value in the series

Seasonality – Repeating cycles in the series

Noise – Random variation in the series

Trend and Seasonality are what make a time series stationary. There are two common ways to deal with trend and stationarity and make the time series stationary.

Differencing and Decomposition

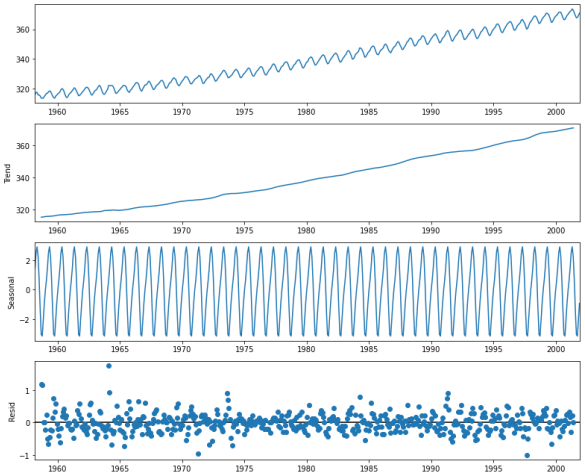
Decomposing a Timeseries

We can decompose the given time series in as shown below.

```
decomposition = sm.tsa.seasonal_decompose(yMonthly,
model='additive')
fig = decomposition.plot()
plt.show()
```

The individual components can be accessed from the returned object. They can simply be removed from the original time series to make it stationary.

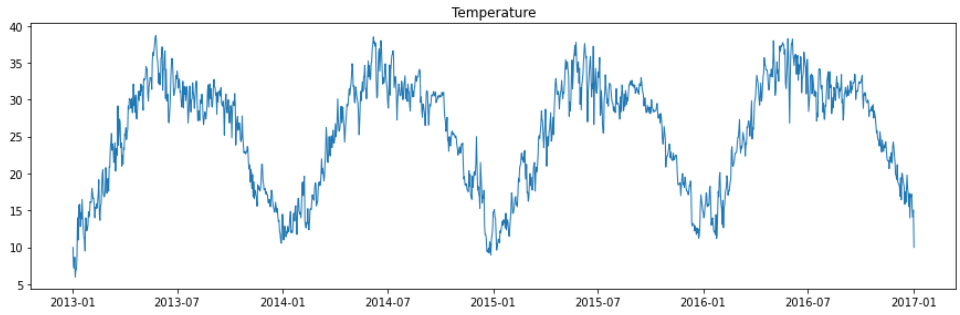
```
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```



Differencing

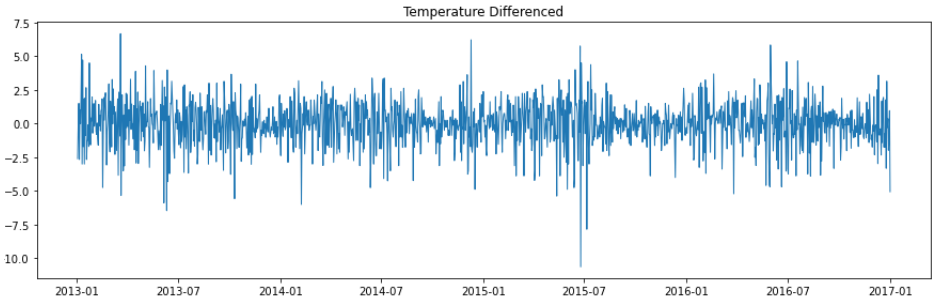
The first difference of a time series is the series of changes from one period to the next.

The image below shows the original values of a time series.




```
differencedTrainMS = dfTrainMS.diff()
```

The image below shows the first difference of the above time series.



However, if we are only dealing with univariate time series analysis we need not do the differencing manually. That will be handled by the forecasting models discussed in the next section.

For Multivariate analysis we will have to do the differencing manually and invert the process after the forecasting to get the correct values.

Part 2 – Picking the model

If there is only one column to data for which we are trying to forecast the future values, it is called univariate time series analysis. One of the most popular models for univariate analysis is **ARIMA.**

If there are multiple columns of data and we are trying to forecast all of them and they all influence each other , we call it multivariate analysis. **Vector Auto Regression** is one of the popular multivariate analysis models.

If we are only interested in forecasting one target variable but we want to use multiple input or predictor variables we can use **Facebook's Prophet.**

4. Univariate Analysis

ARIMA Models

ARIMA stands for Auto Regressive Integrated Moving Average. It has 3 main parameters. **p, q, d**

p = Periods to lag (AR Terms)
if P = 3 then we will use the three previous periods of our time series in the autoregressive portion of the calculation

q = This is the number of MA terms. These are lagged forecast errors. If q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at ith instant and actual value. This variable denotes the lag of the error component, where error component is a part of the time series not explained by trend or seasonality

d = d refers to the number of differencing transformations required by the time series to get stationarity.

Determining the Parameters

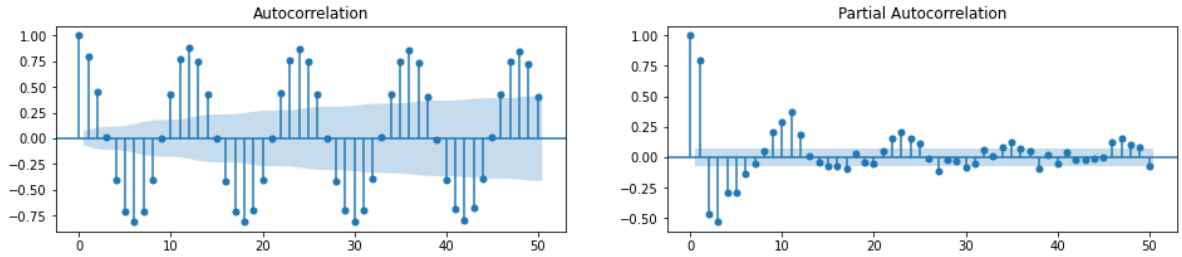
We use ACF and PACF plots to determine the values for p, q and d.

We must use a stationary time series for ACF and PACF plots

These plots show the relationship between a TS value in time and lagged values. An autocorrelation of lag 1 will measure the relationship between Y(t) and Y(t-1)

Autocorrelation and Partial Autocorrelation Plots

```
fig, axes = plt.subplots(1,2,figsize=(16,3))
plot_acf(yMonthly.co2.tolist(), lags=50, ax=axes[0])
plot_pacf(yMonthly.co2.tolist(), lags=50, ax=axes[1])
```



Differencing Term (d)

If the series has positive autocorrelations out to a high number of lags, then it probably needs a higher order of differencing.

If the lag-1 autocorrelation is zero or negative, or the autocorrelations are all small and patternless, then the series does not need a higher order of differencing. If the lag-1 autocorrelation is -0.5 or more negative, the series may be over differenced.

AR Term

If the PACF of the differenced series displays a sharp cutoff and/or the lag-1 autocorrelation is positive--i.e., if the series appears slightly "under differenced"--then consider adding an AR term to the model. The lag at which the PACF cuts off is the indicated number of AR terms.

p – The lag value where the PACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case p=2.

MA Term

If the ACF of the differenced series displays a sharp cutoff and/or the lag-1 autocorrelation is negative--i.e., if the series appears slightly "over differenced"--then consider adding an MA term to the model. The lag at which the ACF cuts off is the indicated number of MA terms.

q – The lag value where the ACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case q=3.

[3]

[4]

Auto ARIMA and Seasonal ARIMA

[5]

Determining the parameters from ACF and PACF plots is time taking and confusing. We can use python packages for doing that automatically. The code below uses auto_arima to determine the best parameters.

```
# Seasonal - fit stepwise auto-ARIMA
smodel = None
smodel = pm.auto_arima(perthTemp_Train_MS[['mintemp']], start_p=1, start_q=1,
                       test='adf',
                       max_p=9, max_q=9, m=12,
                       start_P=0, seasonal=True,
                       d=None, D=1, trace=True,
                       error_action='ignore',
                       suppress_warnings=True,
                       stepwise=True)

smodel.summary()
```

Output

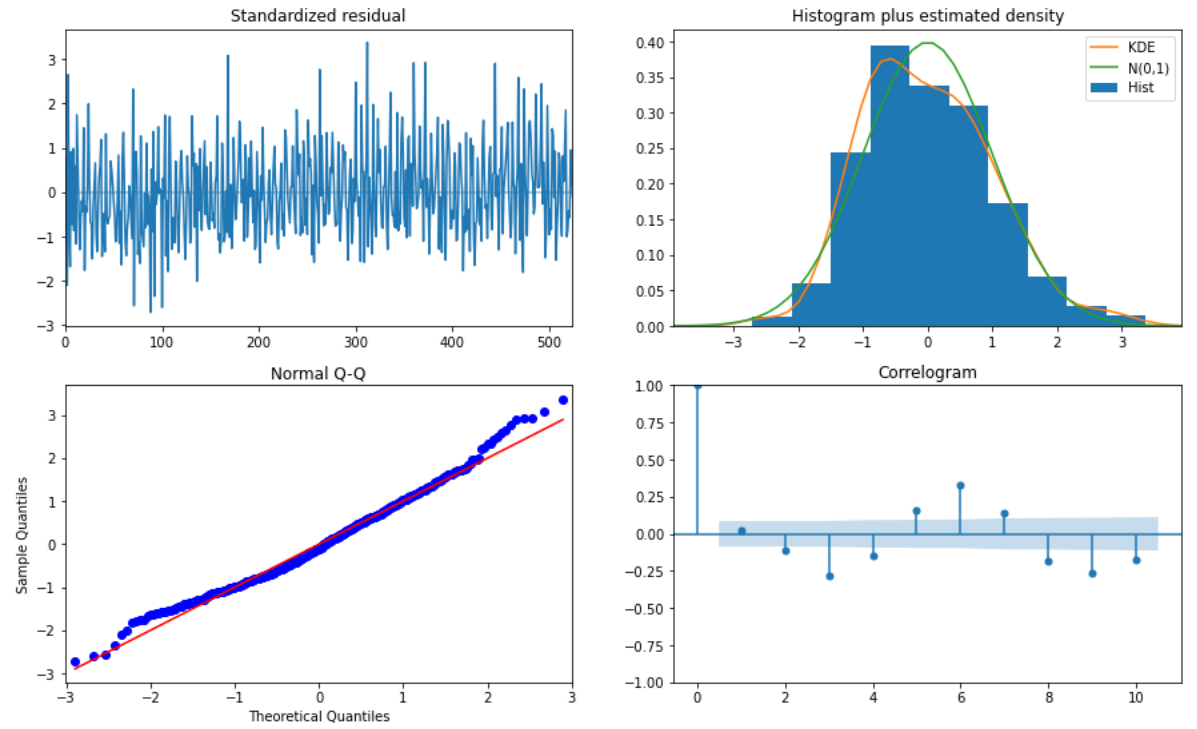
```
ARIMA(3,0,4)(2,1,0)[12] intercept   : AIC=inf, Time=9.70 sec
ARIMA(2,0,3)(2,1,0)[12]             : AIC=2665.060, Time=1.22 sec
ARIMA(2,0,3)(1,1,0)[12]             : AIC=2746.145, Time=0.64 sec
ARIMA(2,0,3)(2,1,1)[12]             : AIC=inf, Time=5.97 sec
ARIMA(2,0,3)(1,1,1)[12]             : AIC=inf, Time=2.84 sec
ARIMA(1,0,3)(2,1,0)[12]             : AIC=2668.655, Time=1.08 sec
ARIMA(2,0,2)(2,1,0)[12]             : AIC=2666.399, Time=1.09 sec
ARIMA(3,0,3)(2,1,0)[12]             : AIC=inf, Time=5.42 sec
ARIMA(2,0,4)(2,1,0)[12]             : AIC=2665.168, Time=1.30 sec
ARIMA(1,0,2)(2,1,0)[12]             : AIC=2668.620, Time=0.78 sec
ARIMA(1,0,4)(2,1,0)[12]             : AIC=2665.581, Time=1.29 sec
ARIMA(3,0,2)(2,1,0)[12]             : AIC=2665.991, Time=1.37 sec
ARIMA(3,0,4)(2,1,0)[12]             : AIC=inf, Time=5.87 sec

Best model:  ARIMA(2,0,3)(2,1,0)[12]
Total fit time: 142.876 seconds
```

The best values for p, d, q are: 2, 0, 3

Analyzing Residuals

Analyzing the results of ARIMA. Residual plot diagnostics.

```
model.plot_diagnostics(figsize=(15,9))
plt.show()
```



5. Multivariate Analysis

VAR

[6]

Vector Autoregression (VAR) is a multivariate forecasting algorithm that is used when two or more time series influence each other.

In the VAR model, each variable is modeled as a linear combination of past values of itself and the past values of other variables in the system.

```
testModel = VAR(perthTemp_Train_MS)

results = testModel.fit(ic='aic')

prediction = results.forecast(results.endog, steps =
                                              len(perthTemp_Test_MS))
```

The first step before fitting the model is to check the relationship among the variables. We need to check that the variables influence each other. A popular method to check this is Granger's Causality Test

Granger Causality

```
def grangers_causality_matrix(X_train, variables, test = 'ssr_chi2test', verbose=False):

    dataset = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, index=variables)

    for c in dataset.columns:
        for r in dataset.index:

            test_result = grangercausalitytests(X_train[[r,c]], maxlag=maxlag, verbose=False)
            p_values = [round(test_result[i+1][0][test][1],4) for i in range(maxlag)]
            if verbose: print(f'Y = {r}, X = {c}, P Values = {p_values}')
            max_p_value = np.max(p_values)
            dataset.loc[r,c] = max_p_value

    dataset.columns = [var + '_x' for var in variables]
    dataset.index = [var + '_y' for var in variables]

    return dataset
```

```
grangers_causality_matrix(perthTemp_Train_MS, variables = perthTemp_Train_MS.columns, verbose = True)
```

```
Y = mintemp, X = mintemp, P Values = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Y = maxtemp, X = mintemp, P Values = [0.0, 0.0, 0.0, 0.0004, 0.0001, 0.0001, 0.0001, 0.0002, 0.0005, 0.0022, 0.0014, 0.0007]
Y = rainfall, X = mintemp, P Values = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Y = mintemp, X = maxtemp, P Values = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Y = maxtemp, X = maxtemp, P Values = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Y = rainfall, X = maxtemp, P Values = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Y = mintemp, X = rainfall, P Values = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Y = maxtemp, X = rainfall, P Values = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0001, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Y = rainfall, X = rainfall, P Values = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

	mintemp_x	maxtemp_x	rainfall_x
mintemp_y	1.0000	0.0	0.0000
maxtemp_y	0.0022	1.0	0.0001
rainfall_y	0.0000	0.0	1.0000

Granger causality test is used to determine if one time series will be useful to forecast another variable by investigating causality between two variables in a time series.

It is based on the idea that if X causes Y, then the forecast of Y based on previous values of Y AND the previous values of X should best result in the forecast of Y based on previous values of Y alone.

[8]

[9]

It accepts a 2D array with 2 columns as the main argument. The values are in the first column and the predictor (X) is in the second column.

The Null hypothesis is: the series in the second column, does not Granger cause the series in the first. If the P-Values are less than a significance level (0.05) then you reject the null hypothesis and conclude that the said lag of X is indeed useful.

Manual Differencing and Inverting

Unlike ARIMA, VAR does not perform a differencing operation by itself. This means that we must check the stationarity of each of the variables independently and then difference those columns that are not stationary.

Since the differencing operation is done manually, after generating the forecast values they must be transformed back into actual values for the forecast. This inverse differencing also needs to be done manually. The process is shown below.

Initial differencing operation

```
differencedTrainMS = dfTrainMS.diff()
```

Generating forecasts

```
prediction = results.forecast(results.endog, steps =
                                                len(perthTemp_Test_MS))
```

Inverse differencing operation

```
# predDF['meantemp'] = np.r_[dfTrainMS[-1:]['meantemp'][0],
                             diffPredDF['meantemp']].cumsum()[1:]
```

np.r_ is a function that just concatenates its arguments and returns an array. In this case we give it two arguments. The first argument is the last values of the variable from the train dataset. The second argument is the differenced forecast values from the VAR model. The numpy function returns a concatenated array. This array is then given as an input to the cumsum() function.

This function returns a cumulative sum array. For example, if we give an array like [1,2,3,4,5] to cumsum() we get this array in return [1, 3, 6, 10, 15]

Each value in the returned array is the cumulative sum of the value in that index position and all the values before it. This in effect inverse differences the values we get from the forecast.

[7]

6. Prophet (By Facebook)

Univariate

To use Prophet for forecasting, first, a Prophet() object is defined and configured, then it is fit on the dataset by calling the fit() function and passing the data.

The Prophet() object takes arguments to configure the type of model you want, such as the type of growth, the type of seasonality, and more. By default, the model will work hard to figure out almost everything automatically.

The fit() function takes a DataFrame of time series data. The DataFrame must have a specific format. The first column must have the name 'ds' and contain the date-times. The second column must have the name 'y' and contain the observations.

This means we change the column names in the dataset. It also requires that the first column be converted to date-time objects, if they are not already (e.g. this can be down as part of loading the dataset with the right arguments to read_csv).

```
prophet_perthMinTemp = fbprophet.Prophet()

prophet_perthMinTemp.fit(perthMinTemp)

forecastDates =
pd.DataFrame(perthTempMS_Test.index).rename(columns={'Date':'ds'})

forecast_minTemp = prophet_perthMinTemp.predict(forecastDates)

forecast_minTemp_1 = forecast_minTemp[['ds', 'yhat']]
```

Official documentation

[10]

[11]

Multivariate (Additional Regressors)

Additional regressors can be added to the linear part of the model using the add_regressor method or function. A column with the regressor value will need to be present in both the fitting and prediction dataframes.

Prophet can only predict/forecast one column per model. To additional predictors we first need to forecast the future values of the predictors and then use them in the final model to forecast the final target.

[12]

7. LSTM

8. DTW

9. Evaluation

10. References

[1] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html

[2] [3] https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/

[4] http://people.duke.edu/~rnau/411arim3.htm

[5] https://www.analyticsvidhya.com/blog/2018/08/auto-arima-time-series-modeling-python-r/

[6] https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/

[7] https://numpy.org/doc/stable/reference/generated/numpy.r_.html

[8] https://towardsdatascience.com/granger-causality-and-vector-auto-regressive-model-for-time-series-forecasting-3226a64889a6

[9] https://www.machinelearningplus.com/time-series/time-series-analysis-python/

[10] https://facebook.github.io/prophet/docs/quick_start.html#python-api

[11] https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/#:~:text=The%20Prophet%20library%20is%20an,and%20seasonal%20structure%20by%20default.

[12] https://facebook.github.io/prophet/docs/seasonality,_holiday_effects,_and_regressors.html#additional-regressors