

INFO6105 DSEM Final Project

Project Name - An Automated Time Series Forecasting Package

Presented by

Parth Shah | 001006181

Ankana Asit Baran Samanta | 001007431

Abstract

Time series forecasting is the process of analysing time series data using statistics and modelling to make predictions and inform strategic decision-making. A general Time series forecasting analysis involves running multiple algorithms, hypertuning the parameters and finding the best algorithm out there to predict the output.

We aim to implement a Python package which automates the process of running each Time Series algorithm individually. The package will take any Time series dataset, run all the Time series algorithms and give the output for the best algorithm out there to predict the results. The estimators considered are MAPE and MSE. For our implementation, we will be considering a Store sales time series dataset for a period of 3 years to build the models.

Applications of Time Series Data

The applications for Time Series data can range from a varied range of areas where the prediction depends on past observations/behaviours of data in certain time periods (Yearly, monthly or daily). It has many applications including :

1. Weather forecasting
2. Stock Market Analysis
3. Health Monitoring
4. Sale Forecasting
5. Process and Quality Control

Components of Time Series

Time series analysis provides a body of techniques to better understand a dataset. Perhaps the most useful of these is the decomposition of a time series into 4 constituent parts:

1. **Level** - The baseline value for the series if it were a straight line.
2. **Trend** - The optional and often linear increasing or decreasing behaviour of the series over time.
3. **Seasonality** - The optional repeating patterns or cycles of behaviour over time.
4. **Noise** - The optional variability in the observations that cannot be explained by the model.

All time series have a level, most have noise, and the trend and seasonality are optional.

Loss Functions

The error calculation for the Time series Forecasting Algorithms will be MAPE and MSE.

1. **MAPE (Mean Absolute Percentage Error)** is an estimator used to measure Forecast Accuracy for Time Series Data. It is the sum of the individual absolute error divided by the demand.
2. **MSE (Mean Squared Error)** is calculated as the average of the squared forecast error values. The error values are in squared units of the predicted values. A mean squared error of zero indicates perfect skill, or no error.

Implementation

To build the Python Package we have implemented each Time series Forecasting algorithm in separate python files first.

Requirements:

To run this package we should have 2 files sales_data and flags_data (exogenous variables)

The column structure for sales data has to be: **DATE,ID,SALES**

Where ID is the level on which we want to forecast the sales(Such as store names for store level prediction or branch name for store branch level predictions).

Data format : DD-MM-YYYY

Python Package Structure:

-Config.py

-Starting.py

- Forecast01.py
- transform.py
- traintest.py
- AlgoARIMA.py
- AlgoSARIMA.py
- AlgoProphet.py
- HoltWinters.py

Input: Sales data , Exogenous variable data

Output: Best Algorithm CSV

1. ARIMA Model

Arima stands as an acronym for AutoRegressive Integrated Moving Average a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.

File Name: **AlgoArima.py**

We are running the Arima Algorithm from the file.

Arima only takes Univariate data.

```
# function for forecasting values using ARIMA model
def arimax(data,n,pval=None,qval=None,trainFlags=None,forecastFlags=None,
           only_error=False,testdata=None):

    if pval!=None:
        p = pval
    else:
        p = pparameter(data)

    if qval!=None:
        q = qval
    else:
        q = qparameter(data)
    model = ARIMA(data,order=(p,0,q),exog=trainFlags)
    results = model.fit(dis=0)
    forecast = results.predict(start=len(data),end=len(data)+n-1,exog=forecastFlags)
    if only_error==False:
        return forecast
    else:
        error = mean_squared_error(forecast,testdata)
        return error
```

As Arima requires 2 parameters(p and q) we calculate this on the basis of the data inserted in the code.

p - The number of lag observations included in the model, also called the lag order.

q - The size of the moving average window, also called the order of moving average.

2. ARIMAX Model

Arimax is an extension of Arima. The X in the ARIMAX stands for “exogenous”. In our dataset, we have a flag for holidays which acts as an exogenous variable. It is a class of models that captures a suite of different standard temporal structures in time series data.

File Name: AlgoArima.py

```
# function for forecasting values using ARIMA model
def arimax(data,n,pval=None,qval=None,trainFlags=None,forecastFlags=None,
           only_error=False,testdata=None):

    if pval!=None:
        p = pval
    else:
        p = pparameter(data)

    if qval!=None:
        q = qval
    else:
        q = qparameter(data)
    model = ARIMA(data,order=(p,0,q),exog=trainFlags)
    results = model.fit(dis=0)
    forecast = results.predict(start=len(data),end=len(data)+n-1,exog=forecastFlags)
    if only_error==False:
        return forecast
    else:
        error = mean_squared_error(forecast,testdata)
        return error
```

Contrast to ARIMA in ARIMAX we pass exogenous variables as forecastFlags parameters as TRUE.

3. SARIMA Model

A seasonal autoregressive integrated moving average (SARIMA) model is one step different from an ARIMA model based on the concept of seasonal components. ARIMA does not support seasonal data so the seasonal data is adjusted using seasonal differencing.

File Name: AlgoSARIMA.py

```
1 from pmdarima.arima import auto_arima
2 from transform import difference, transform
3
4 #Full data(in form of pd.Series) should be passed. Not train/test data
5 def sarimax(data, length, f, n=1, Flags=None, only_error=False, exog=None):
6     """
7     data(pd.Series): data in form of pandas series
8     f-frequency(string): "D" = day level data ; "W" = weekly level data ; "M" = Monthly level data
9     length(int): No.of data points to be used for testing
10    n(int): no.of periods ahead to forecast
11    Flags(pd.DataFrame): flags data corresponding to dates in 'data '
12    exog(pd.DataFrame): flags data corresponding to dates for forecast period
13    only_error(True/False): Returns only MSE and test data predictions if True ; Returns future forecasts if False
14    """
15    if f=="W":
16        frequency = 52
17    elif f=="M":
18        frequency = 12
19    model = auto_arima(data, exogenous=Flags, m=frequency, information_criterion='oob',
20                      out_of_sample_size=length, scoring='mse', seasonal=True)
21
22    if only_error==True:
23        error = model.oob_
24        pred = model.oob_preds_
25        return error ,pred
26
27    if only_error==False:
28        preds = model.predict(n_periods=n, exogenous=exog)
29        return preds
30
31
32
```

This python file helps us to run seasonal Arima models. We use the pmdarima package where the seasonal parameter is passed as true. From the user we take frequency parameters for this algorithm.

SARIMA will only run when the sales data is at the *monthly level*.

4. SARIMAX Model

An extension of SARIMA. The X in the SARIMAX stands for “exogenous”. It includes seasonal effects and exogenous factors with the autoregressive and moving average component in the model.

File Name: AlgoSARIMA.py

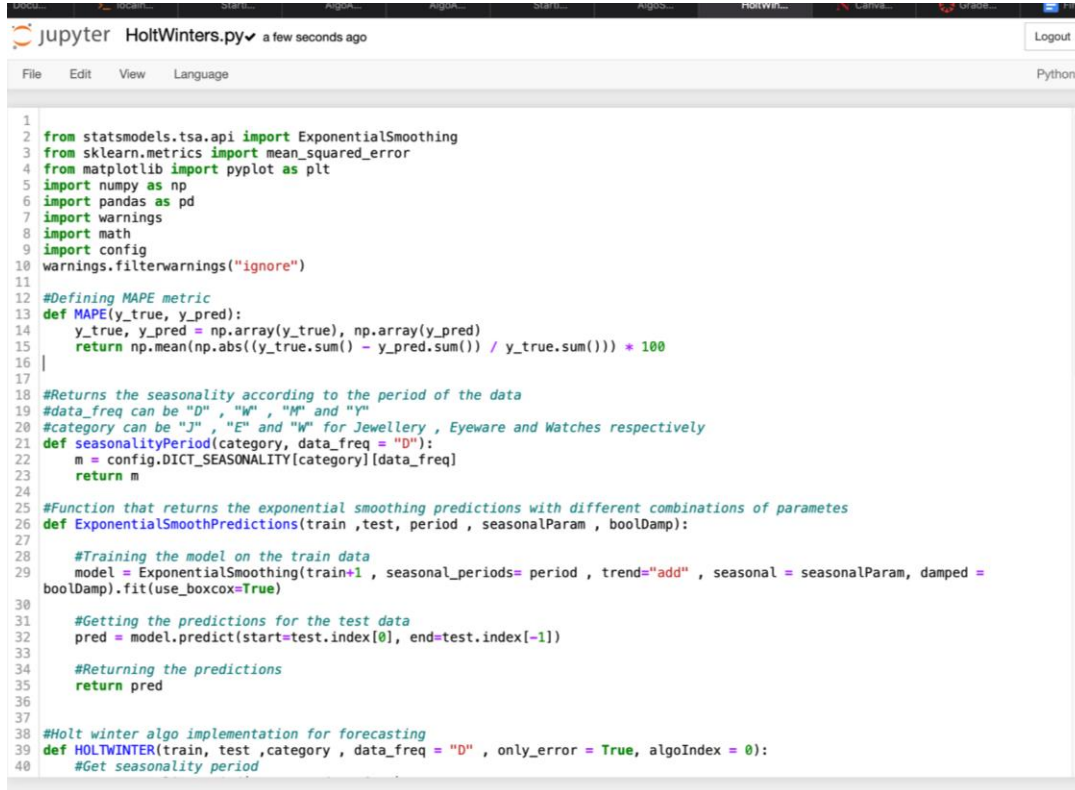
```
1
2 from pmdarima.arma import auto_arma
3 from transform import difference, transform
4
5 #Full data(in form of pd.Series) should be passed. Not train/test data
6 def sarimax(data, length, f, n=1, Flags=None, only_error=False, exog=None):
7     """
8     data(pd.Series): data in form of pandas series
9     f--frequency(string): "D" = day level data ; "W" = weekly level data ; "M" = Monthly level data
10    length(int): No. of data points to be used for testing
11    n(int): no. of periods ahead to forecast
12    Flags(pd.DataFrame): flags data corresponding to dates in 'data '
13    exog(pd.DataFrame): flags data corresponding to dates for forecast period
14    only_error(True/False): Returns only MSE and test data predictions if True ; Returns future forecasts if False
15    """
16    if f=="W":
17        frequency = 52
18    elif f=="M":
19        frequency = 12
20    model = auto_arma(data, exogenous=Flags, m=frequency, information_criterion='oob',
21                     out_of_sample_size=length, scoring='mse', seasonal=True)
22
23    if only_error==True:
24        error = model.oob_
25        pred = model.oob_preds_
26        return error ,pred
27
28    if only_error==False:
29        preds = model.predict(n_periods=n, exogenous=exog)
30        return preds
31
32
```

```
error, pred = AlgoSARIMA.sarimax(salesdata, len(testsales), f=frequency,
                                Flags=flagsdata, only_error=True)
```

We pass Flags data in the same function as used while doing sarima.

5. HOLT- Winters Model

The Holt-Winters method is a very common time series forecasting procedure capable of including both trend and seasonality.



```
1 from statsmodels.tsa.api import ExponentialSmoothing
2 from sklearn.metrics import mean_squared_error
3 from matplotlib import pyplot as plt
4 import numpy as np
5 import pandas as pd
6 import warnings
7 import math
8 import config
9 warnings.filterwarnings("ignore")
10
11 #Defining MAPE metric
12 def MAPE(y_true, y_pred):
13     y_true, y_pred = np.array(y_true), np.array(y_pred)
14     return np.mean(np.abs((y_true.sum() - y_pred.sum()) / y_true.sum())) * 100
15
16 #Returns the seasonality according to the period of the data
17 #data_freq can be "D", "W", "M" and "Y"
18 #category can be "J", "E" and "W" for Jewellery, Eyeware and Watches respectively
19 def seasonalityPeriod(category, data_freq = "D"):
20     m = config.DICT_SEASONALITY[category][data_freq]
21     return m
22
23 #Function that returns the exponential smoothing predictions with different combinations of parameters
24 def ExponentialSmoothPredictions(train, test, period, seasonalParam, boolDamp):
25     #Training the model on the train data
26     model = ExponentialSmoothing(train+1, seasonal_periods= period, trend="add", seasonal = seasonalParam, damped =
27     boolDamp).fit(use_boxcox=True)
28
29     #Getting the predictions for the test data
30     pred = model.predict(start=test.index[0], end=test.index[-1])
31
32     #Returning the predictions
33     return pred
34
35 #Holt winter algo implementation for forecasting
36 def HOLTWINTER(train, test, category, data_freq = "D", only_error = True, algoIndex = 0):
37     #Get seasonality period
```

File Name: HoltWinters.py

In Holt winters we have considered four different Cases of generating a prediction

Case 1 : Additive trend and additive seasonality ; No Damp

Case 2 : Additive trend and multiplicative seasonality ; No Damp

Case 3 : Additive trend and additive seasonality ; With Damp

Case 4 : Additive trend and multiplicative seasonality ; With Damp

The lowest MAPE value among all four is selected as the prediction from HoltWinter algorithm.

6. Time Series Linear Model

tslm is used to fit linear models to time series including trend and seasonality components.

File Name: AlgoArima.py

TSLM models run arima model with trend and seasonality as a component

```
def allcombTSLM(trainsales, testsales, frequency, n, trainflags, testflags):  
    #creating dataframe for storing MSE for models  
    errortable = pd.DataFrame()  
  
    ##### TSLM WITHOUT FLAGS (MODEL - ADDITIVE) #####  
    # AM - ADDITIVE MOEDL  
    ### RESIDUAL + TREND + SEASONALITY ###  
    try:  
        tslmForecast = TSLM(trainsales, f=frequency, n=len(testsales))  
        error = MSE(tslmForecast, testsales)  
        error2 = MAPE(testsales, tslmForecast)  
        errortable = errortable.append({'Algorithm': '(AM)TSLM', 'MSE': error,  
                                         'MAPE': error2},  
                                       ignore_index=True)  
    except:  
        errortable = errortable.append({'Algorithm': '(AM)TSLM', 'MSE': np.nan},  
                                       ignore_index=True)  
  
    ### RESIDUAL + SEASONALITY ###  
    try:  
        tslmForecast = TSLM(trainsales, f=frequency, n=len(testsales),  
                             remove_trend=True)  
        error = MSE(tslmForecast, testsales)  
        error2 = MAPE(testsales, tslmForecast)  
        errortable = errortable.append({'Algorithm': '(AM)TSLM(Trend)',  
                                         'MSE': error, 'MAPE': error2},  
                                       ignore_index=True)  
    except:  
        errortable = errortable.append({'Algorithm': '(AM)TSLM(Trend)',  
                                         'MSE': np.nan}, ignore_index=True)  
  
    ### RESIDUAL + TREND ###  
    try:  
        tslmForecast = TSLM(trainsales, f=frequency, n=len(testsales),  
                             remove_seasonality=True)  
        error = MSE(tslmForecast, testsales)  
        error2 = MAPE(testsales, tslmForecast)  
        errortable = errortable.append({'Algorithm': '(AM)TSLM(Seas)',  
                                         'MSE': error, 'MAPE': error2},  
                                       ignore_index=True)
```

In our allcombTSLM function we consider following types of predictions

1) Additive

- RESIDUAL + TREND + SEASONALITY
- RESIDUAL + TREND
- RESIDUAL + SEASONALITY

2) Multiplicative

- RESIDUAL + TREND + SEASONALITY
- RESIDUAL + TREND
- RESIDUAL + SEASONALITY

There is a best method approach inside TSLM which compares all the above methods and gives out the best prediction for TSLM.

7. Prophet(Additive Seasonality) Model

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It is easy to use and designed to automatically find a good set of hyperparameters for the model in an effort to make skillful forecasts for data with trends and seasonal structure by default.

File Name: **AlgoArima.py**

```
import pandas as pd
from fbprophet import Prophet
from sklearn.metrics import mean_squared_error

##Expected sales format- pandas series with dates in index
##Expected format for flags - pd.DataFrame with all dates(Do not split into train and test)
def prophet(sales,n,flags=None,seasonality='additive',f="D",only_error=False,testdata=None):
    """
    sales(pd.Series): sales data with dates in index
    n(int): No.of periods into future to forecast
    flags(pd.DataFrame): flags data for data corresponding to sales data AND future forecast periods
    Seasonality(string("additive"/"multiplicative")): "additive" - to be used for constant variance in seasonality ;
    "multiplicative" - to be used for increasing variance in seasonality
    f(string("D","W","M")): frequency of data - daily/weekly/monthly
    only_error(True/ False): Returns only MSE if True ; Returns future forecasts if False
    """
    try:
        #Converting Sales and flags data to format required by Prophet
        if type(sales)==pd.Series:
            #converting pd.Series to pd.DataFrame
            prophdf = pd.Series.to_frame(sales)
        else:
            prophdf = sales
            prophdf["ds"] = prophdf.index
            prophdf.columns = ["y","ds"]

        #checking if a df has been passed to flags argument
        if isinstance(flags,pd.DataFrame):
            #Converting flags data to format required by Prophet
            holiday=pd.DataFrame({'holiday':[],'ds': []})
            for i in flags.columns.values[1:]:
                dates=flags[(flags[i]==1)].index.values
                holidays = pd.DataFrame({'holiday':i,'ds': dates})
                holiday=pd.concat([holiday,holidays])
        else:
            holiday=None
```

Our prophet function in the timeseries helps us in first converting the data frame to prophet understandable pandas series. Where date has to be passed as an index.

We have kept seasonality = "additive". And passed exogenous variables as holidays to the function.

Building the Main Package

Now that all the algorithms have been implemented, let us now see how to build a package that automates the entire process above. Here the package integrates the working of all the algorithms.

Following are files which we have made to run all algorithms smoothly and integrate it on a superior level.

- 1) Config.py
— Here the user can pass it sales data and flag data paths. Which will be consumed by the package.
- 2) Starting.py
— This is the main wrap up file which imports all other functions from different files. To run the package we have run the **forecast function** in this file.
- 3) Terminal.py
— This file is made to call the forecast function from Starting.py
- 4) Transform.py
— The transformation of data such as standardisation, normalisation, differencing is done in this file. Before we feed the data to algorithms we transform it using this file.
- 5) Trainandtest.py
— File used to divide the dataset into train and test sets
- 6) runstreamlit.py
— The streamlit application which we have made for UI is coded in this application!
- 7) Neededpackage.txt: List of all packages which needs to be installed before running this package!

Following are the algorithms files which are mentioned above

- 1) AlgoARIMA.py 2) AlgoSarima.py 3) HoltWinters.py 4) AlgoProphet.py

The final Step:

Once all the algorithms are run we store them in a dataframe to see which algorithm gave the minimum MAPE for a particular level and push out the final winner algorithm file.

Steps to run the Package

There are two ways we can run this package. First is through a web app on Streamlit and second on Command Prompt Terminal.

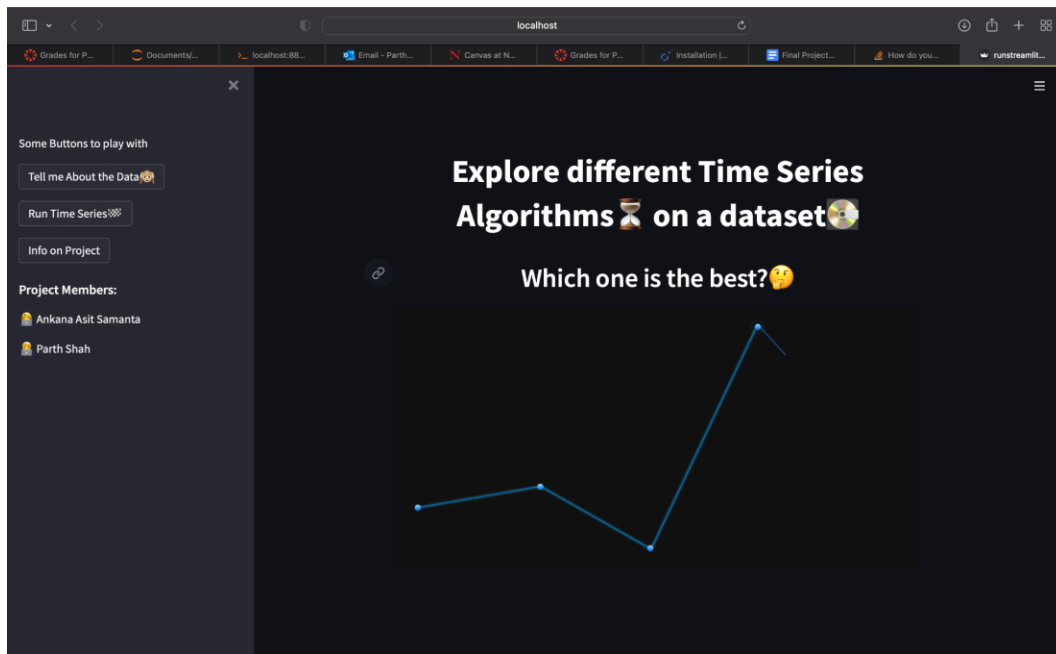
1. GUI implementation using Streamlit

We have implemented a user friendly interface to run the Time series Package using Streamlit Python Package. Streamlit is an open-source Python library that creates web apps for machine learning and data science.

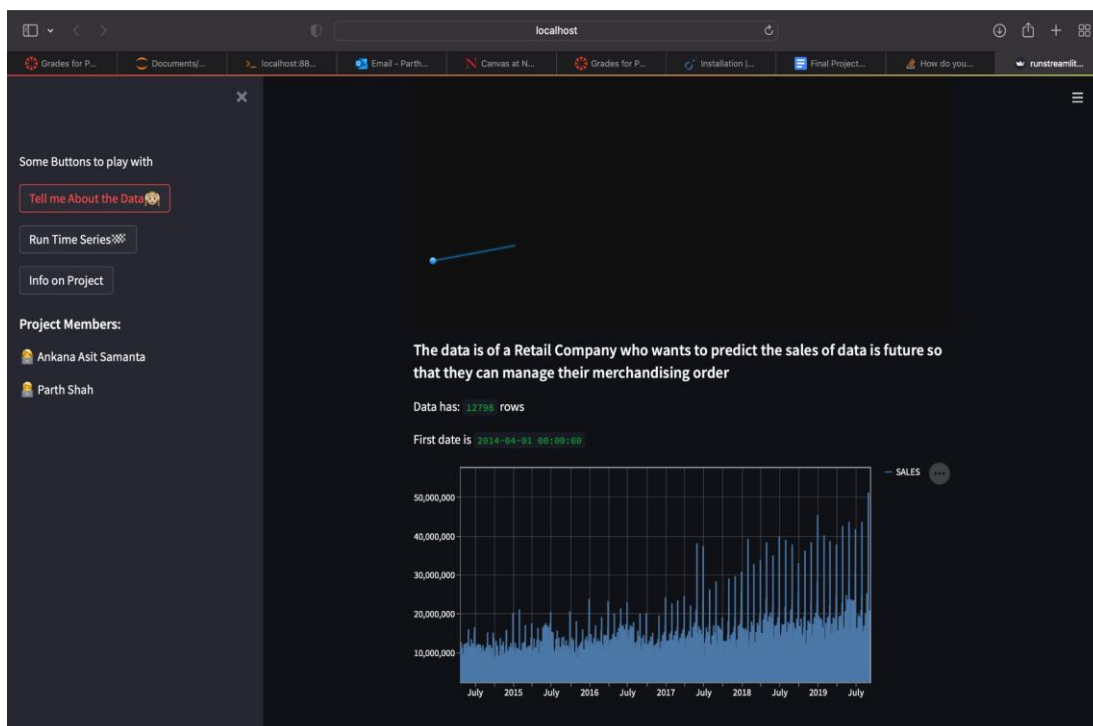
Steps

1. Unzip the TimeSeriesPackage folder
2. Make sure to have the packages installed from neededpackage.py
3. From terminal go to the TimeSeriesPackage folder and run following command :
`streamlit run runstreamlit.py`
4. This will open a good looking UI to run our package!

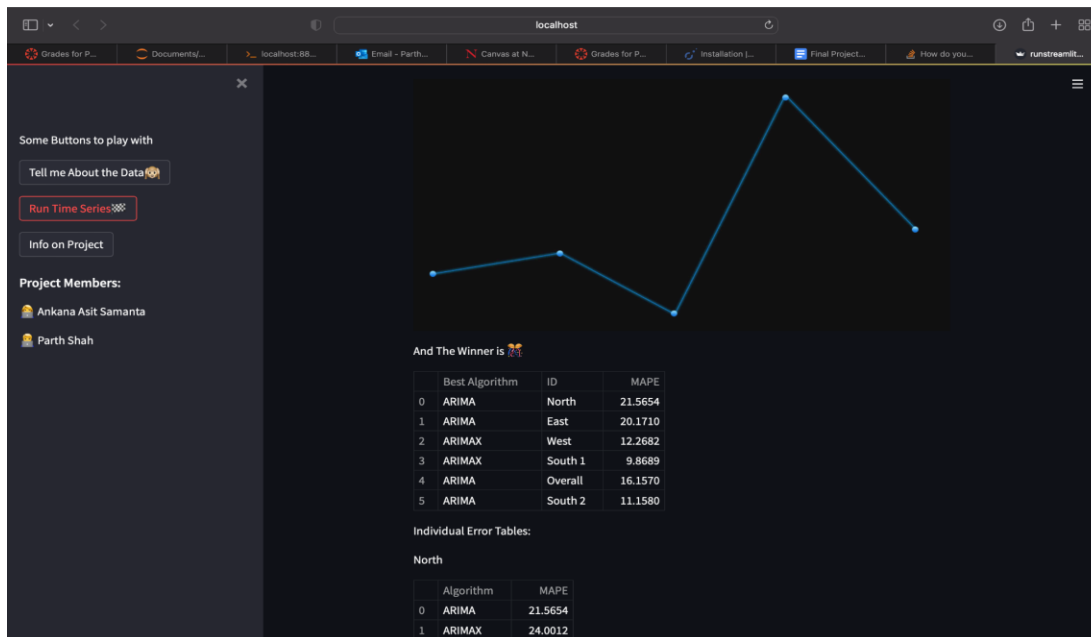
Main Page:



EDA of Data Uploaded:



Run the Time Series Package Button: This page runs the package on data provided in the config file and gives the best algorithms for every branch of the store. It also shows all the estimator values of MAPE for all algorithms in each branch of the store.



Info on Project Button:

The screenshot shows the same web application interface as the previous one, but with the 'Info on Project' button selected. The main content area now displays project information. It starts with a paragraph: 'Now a days ML is running towards becoming Auto-ML and in fast changing world companies want quick results from multiple methodologies. And one such important domain in ML is Timeseries . Our future goal is to build a website completely running all available time series model out there and giving out the results on any given dataset. What all can our package do right now?'. This is followed by a bulleted list of features: 'It runs 5 Different Algorithms on a data Provided', 'We can tune in at any level we want our predction to be weekly, monthly!', 'Package gives precition on basis of level provided by the user! It can be on store level or store item level or region etc', and 'After Running all algorithms it checks the MAPE value and stores the lowest MAPE value algorithm for future predictions'. The section ends with a link 'How to run the Project'.

2. Steps to Run the function on terminal:

- 1) Unzip the TimeSeriesPackage folder
- 2) Make sure to have the packages installed from neededpackage.txt
- 3) From terminal go to the TimeSeriesPackage folder and run following command:

```
python Terminal.py
```

- 4) This will download Final_Algotable.csv in your current directory with the results

Example:

Best Algorithm	ID	MAPE
SARIMAX	South 1	0.008608445991076060
SARIMAX	South 2	0.43461518010785800
HOLT-WINTERS-AMNd	East	0.4241007409747720
SARIMA	North	1.192970019715950
ARIMAX	West	0.46708978835002200
SARIMA	Overall	0.15467145250016300

Conclusion

Different Time series algorithms were executed and evaluated based on the MAPE and MSE loss functions. The Time Series Package was built to integrate all the models and the best algorithms were estimated for each branch of the store.

References

- <https://machinelearningmastery.com/time-series-forecasting-performance-measures-with-python/>
- <https://www.analyticsvidhya.com/blog/2020/10/how-to-create-an-arima-model-for-time-series-forecasting-in-python/>
- <https://docs.streamlit.io/>
- <https://github.com/paulozi/arauto>
- <https://365datascience.com/tutorials/python-tutorials/arimax/>
- <https://www.wisdomgeek.com/development/machine-learning/sarima-forecast-seasonal-data-using-python/>
- <https://facebook.github.io/prophet/>
- <https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/>
- <https://otexts.com/fpp2/transformations.html>
- <https://otexts.com/fpp2/expsmooth.html>
- <https://www.rdocumentation.org/packages/forecast/versions/8.15/topics/tslm>