Flask RESTful API – For AD Capstone Project (Link Here)

A dataset of inventory data on >80 items, monthly data spanning 10 years, was examined, with the goal in mind being to forecast future demand. This API uses the Seasonal ARIMA method in Python to meet this goal.

Table of Contents

API Reference - Resources

Method	HTTP Request	Description
URIs relative to default setting of http://localhost:5000, and localhost is 127.0.0.1 by default		
put	PUT?data={*sendingData*}&id=*requestId*	1) Sends latest month's inventory demand *data* from WebApp to this API. 2) Then runs *GenerateModel* on a separate thread. 3) Returns a string message "Put completed.". Note: • The *requestId* generated by the WebApp is the date on which the request is made, in *ddmmYY* format. (Example) • This is a **scheduled Request** that the WebApp sends every 28th of the month at 0000hrs
get	GET?id= *requestId*	Retrieves latest set of forecasts from *savedPredictions.txt* file, and returns this data as a JSON string. (Example) **Note:** This Request is sent when GetOrder -> GetRecommendedQuantity is selected in the WebApp

API Reference – By Class

Method	Description
Main	
put()	As above.
get()	As above.
GenerateModel()	1) Executes *ProcessData()* with the data from *put()*. 2) Executes *Prediction.GenerateModel()* with the processed data from #1; returns next time period's forecast. 3) Stores next time period's forecast in *savedPredictions.txt* file (overwrites). 4) Returns a string message "Model Generation completed.".
Prediction	
GenerateModel()	1) Calculate the model parameter *d* using *differenceCount()*. 2) Generate list of parameter combinations using *paramList()*. 3) Execute an instance of *GridSearchAndSave()* for each inventory item, in parallel. 4) Save all generated SARIMA models into a single *joblib* file. **Note:** • #3 executes in *Parallel* using the *multiprocessing* library. The number of parallel jobs executed concurrently depends on the number of CPUs the host device has. On a 12-core i7 laptop, this entire process takes 20-30mins. Since this process will only run at midnight, there is sufficient buffer time to run when there would (usually) not be any User using this WebApp, let alone calling specifically for forecast data. • For debugging purposes, *multiprocessing* does not work well on Spyder IDE. However, there is no issue if the API is run externally, or in any other environment.
GetPrediction()	1) Load last forecasted data from saved *joblib* file. 2) Round off the forecasted figures to the nearest whole number, via *UpOrDown()*. 3) Reverse the Differencing effect on the forecasted data, by adding the previous month's figure to the forecasted figure. 4) Convert the forecasted data format to a JSON string, and return this data. 5) The Sender of this *Get* request receives this data.
ProcessData	
ProcessData()	1) Read all historical data previously passed to the API (stored internally as a CSV file) 2) Append the newly-received data to historical dataset. 3) Save this updated dataset to CSV (Overwrite existing file). 4) Add DateStamp to this DataFrame (Monthly time interval). 5) Returns this DataFrame.

DifferenceCount	
differenceCount()	Takes in a DataFrame as a parameter; returns how many rounds of Differencing on the data is required for it to achieve stationarity. Uses the *Augmented Dickey-Fuller Test*.
ParamList	
paramList()	Takes in variable calculated by *differenceCount()* and returns a list of parameter combination strings in form [(p,d,q), (P,D,Q,m),t] for the SARIMA model.
GridSearchAndSave	
GridSearchAndSave()	Runs *bestModel()* and saves the output model. **Note:** This function is timed and is supposed to print the time taken for performing a Grid Search for one model. However, similar to Error messages, printed lines do not always print when running *multiprocessing*.
BestModel	
bestModel()	For each parameter combination: 1) Builds SARIMA model and returns this model's RMSE term, via *modelEvaluated()*. 2) Evaluates RMSE terms, searching for the model with the lowest RMSE score. 3) Trains and returns model with the parameters associated with the best (i.e. lowest) RMSE score. (Using *SARIMAX*)
UpOrDown	
UpOrDown()	Specifically for handling the rounding of forecast figures in *GetPrediction()* before sending them back to the WebApp.
ADFuller_Test	
adfuller_test()	Tests for stationarity among the inventory items' demand records, with significance level set at 5%. **Note:** Items with 0 demand will always return as non-stationary. Based on historical dataset for the past 10 years, there are 9 such item records.
ModelEvaluated	
modelEvaluated()	1) Train-Test-Split on the data (Set to 80% for training) 2) Using *SARIMAX*, Train model, Fit model and forecast a number of steps equal to the length of the Test dataset 3) Calculate the Mean Squared Error, and return its square-root, also known as Root Mean-Squared Error (RMSE).

Historical Data

The historical dataset of monthly demand for inventory, spanning Jan 2010 to Jan 2020 (10 years 1 month; 121 data points), was consolidated, arranged and had the Date-Stamp included.

```
In [3]: len(demand)
Out[3]: 120
```

```
In [4]: demand.head()
Out[4]:
```

	C001	C002	C004	C005	C006	E001	E002	E003	E004	E005	...	T001	T002	T003	T020
0	12	8	6	14	3	1830	510	450	250	2230	...	3	2	1	0
1	13	9	7	15	4	1927	535	469	264	2349	...	4	3	2	0
2	15	10	8	17	4	2180	602	531	296	2627	...	4	3	2	0
3	16	11	9	18	5	2317	644	561	315	2826	...	5	4	3	0
4	11	7	6	12	3	1516	418	367	206	1813	...	3	3	2	0

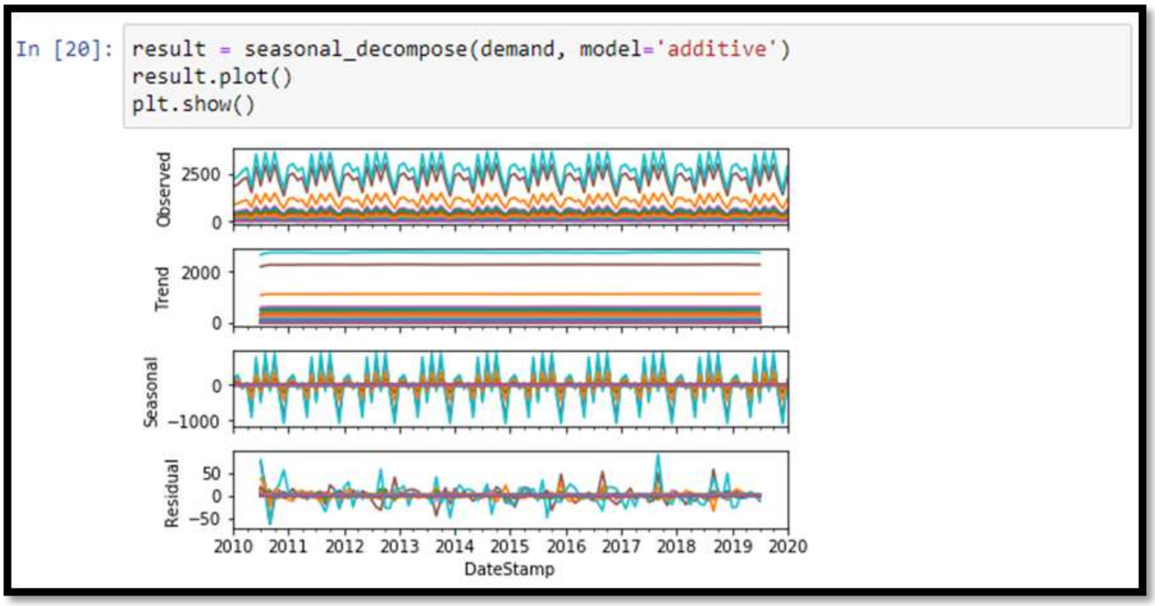
5 rows × 85 columns

```
In [5]: demand.tail()
Out[5]:
```

	C001	C002	C004	C005	C006	E001	E002	E003	E004	E005	...	T001	T002	T003	T020
115	21	15	13	24	8	2971	829	718	411	3665	...	8	6	5	0
116	16	11	10	18	5	2141	601	513	289	2514	...	5	5	4	0
117	22	16	14	25	9	2989	841	736	413	3681	...	9	7	6	0
118	16	11	10	17	5	2061	577	531	293	2467	...	5	5	4	0
119	10	8	7	12	5	1399	389	342	186	1608	...	5	4	3	0

5 rows × 85 columns

When visually examining the dataset, a possibly consistent pattern in monthly fluctuations in demand was observed overall for all items, suggesting an annual seasonal trend.



A breakdown of graphs by item is available at Appendix A.

Methodology

As observed, the nature of the data is time series seasonal. In other words, the method to be deployed for this Use Case is to have the following characteristics:

- As inventory management (i.e. cost) is crucial, the method's accuracy is essential. In other words, the method utilized must strive for the perfect balance between as low an error term as possible, while not overfitting;
- As there is no data on what drives changes in inventory demand to determine a causal effect of any kind; the method used shall be making use of only monthly records of inventory demand; and
- The method deployed is to have strong (i.e. accurate) predictive power using only 120 data points.

Therefore, a judgement call was made to utilize simpler statistical methods; namely by constructing a Seasonal Auto-Regressive Integrated Moving Average ('SARIMA') model for each inventory item in order to forecast. A SARIMA model is implemented via the following steps:

1) Data Transformation to achieve stationarity in the dataset (i.e. consistent Mean and Standard Deviation through time)

Differencing, or taking the changes from one time period to the next, is the simplest and among the most effective methods of doing so. This is done in

2) A Grid Search method shall be used to determine the optimal model.

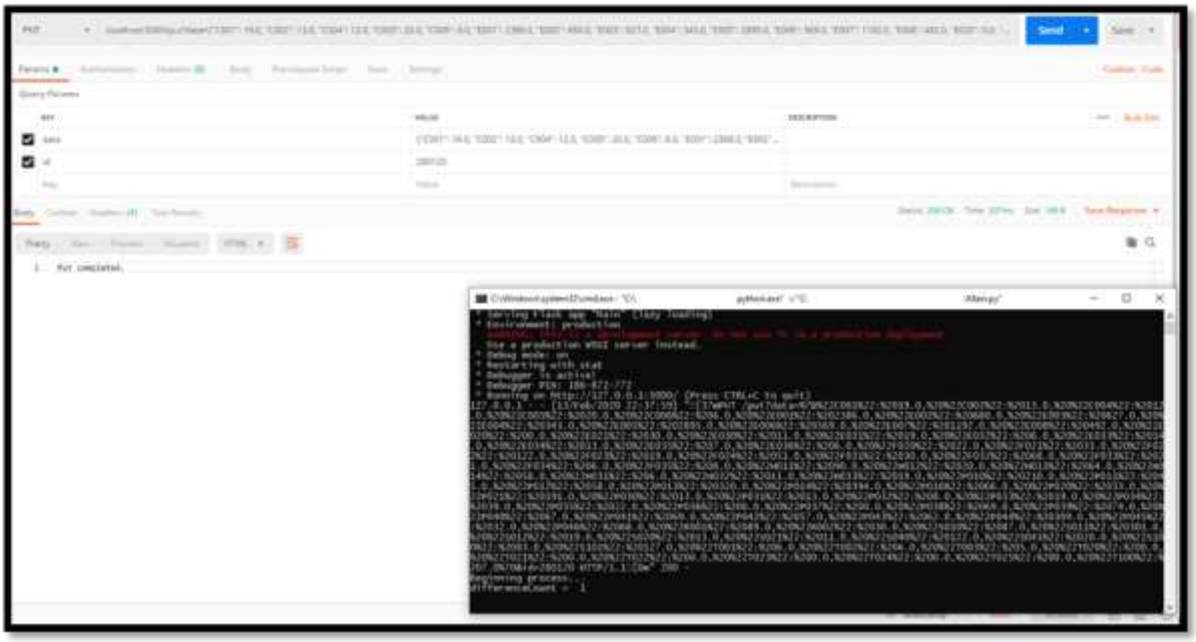
Models shall be constructed using a range of possible input variables and measured by RMSE (root mean-square error), which is the Standard Deviation of the average prediction error. For each inventory item's Grid Search, the model yielding the lowest RMSE shall be considered the best model. This is done in *GridSearchAndSave()*.

3) Using the optimal model, the expected demand to occur in the next period of the time series shall be forecasted.

4) Due to the Data Transformation in Step #1, the forecasted value is the Difference in demand from the last time period. As such, the previous month's demand value has to be added to this forecasted value in order to determine the forecasted demand quantity.

Example Responses

An example of a response to a Put request:



An example of a response to a Get request: