

Low Level Document

BACK-ORDER PREDICTIONS APPLICATION

LLD History :-

Date	Version	Author
08/09/2021	V1.1	Vikas, Chaudhary Shubham R., Bijayalaxmi Kar

Approval Status:

Version	Review Date	Reviewed By	Approved By	Comments

Contents

LLD History

Abstract

1. Introduction.....	6
1.1. What is Low-Level design document?.....	6
1.2. Scope.....	6
1.3. Constraints.....	6
2. Architecture.....	7
3. Architecture Description.....	8
3.1. Data Description.....	8
3.2. Data Transformation.....	8
3.3. Exploratory Data Analysis.....	9
3.4. Event Log.....	9
3.5. Data Insertion into Database.....	10
3.6. Export Data from Database.....	11
3.7. Data Pre-processing.....	13
3.8. Model Building	13

3.9. Hyper Parameter Tuning.....	14
3.10. Model Dump.....	14
3.11. Data from User.....	15
3.12. Data Validation.....	15
3.13. Model Call for Specific Inputs.....	15
3.14. Saving Output in .csv File.....	15
3.15. User Interface.....	16
3.16. Deployment.....	17
4. Technology Stack.....	18
6. Directory Tree Structures.....	19
5. Unit Test Cases	23

Abstract

Increasing demand of products is a common cause of out of product inventory, and the adoption of backordering to satisfy outstanding customer orders after its occurrence cannot be undermined. However, wrong management of backorders incurs several issues such as delay in product delivery, low customer satisfaction, and many more. Therefore, it is necessary to ascertain products with high tendencies of shortage beforehand in order to undertake proactive measures and potentially mitigate both tangible and intangible costs. Hence, this paper proposes a backorder predictive model using Machine learning techniques on large and imbalanced inventory dataset. The data was pre-processed using Robust Scaler, while data balancing method was applied on the imbalanced data simultaneously and their output were fed into Random Forest Classifier / XgBoost Classifier to predict which item goes on backorder. The evaluation of the result obtained showed had performed better with 0.99 precision, 0.96 recall, and 0.989 F1-Score. The proposed model when compared with other machine learning algorithms shows significant impact on prediction of product backorder.

1. Introduction

1.1. What is Low-Level design document?

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Back Order Prediction Model. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

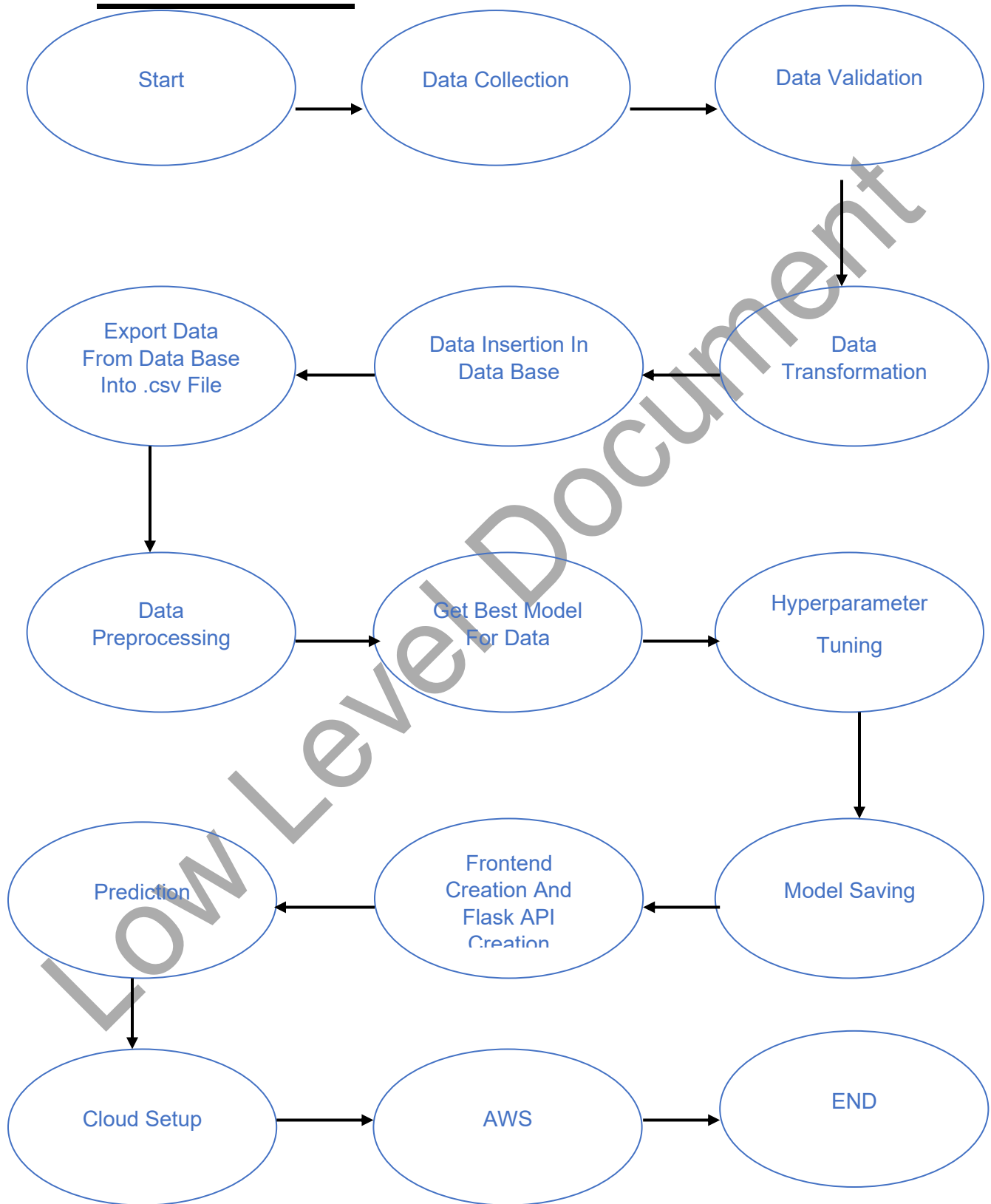
1.2. Scope

Low-level design (LLD) is a component-level design process that follows a step-by step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

1.3. Constraints

Internet connection is a constraint for the application. Since the application fetched the data from the database, it is crucial that there is an Internet connection for the application to function. Since the model can make multiple requests at same time, it may be forced to queue incoming requests and therefore increase the time it takes to provide the response.

2.Architecture



3. Architecture Description

3.1. Data Description

We have 2 lakh Dataset row columnar data includes sku , national_inv , lead_time , in_transit_qty , forecast_3_month , forecast_6_month , forecast_9_month , sales_3_month , sales_6_month , sales_9_month , min_bank , potential_issue , pieces_past_due , perf_6_month_avg , perf_12_month_avg , local_bo_qty , deck_risk , oe_constraint , ppap_risk , stop_auto_buy , rev_stop , went_on_backorder. These is given in the comma separated value format (.csv).These data is Given By Company Team which contains both the test data and train data.

3.2. Data Cleaning / Data Transformation

In the Cleaning process, we have cleaned up all the data because data is present in very bad format which was can not recognized by machine .So data Cleaning is done very first by data validation methods. In which we are create a json file in which name of file , numbers of columns etc information present. File name is given in “BackOrder_08012020_120000” Format.

3.3. Exploratory Data Analysis

In EDA we have seen various insights from the data so we have selected which column is most important and dropped some of the columns by observing them spearman rank co-relation and plotting their heatmap from seaborn library also we done null value managed in an efficient manner and also implemented categorical to numerical transfer of column method here.

3.4. Event Log

The system should log every event so that the user will know what process is running internally. Logging is implemented using python's standard logging library. Initial step-by-step description:-

- The system should be able to log each and every system flow.
- System must be able to handle logging at greater scale because it helps debugging the issue and hence it is mandatory to do.

3.4. Data Insertion into Database

- ❖ Database Creation and connection - Create a database with name passed. If the database is already created, open the connection to the database.
- ❖ Table creation in the database.
- ❖ Insertion of files in the table

The screenshot shows the DataStax Astra web interface. The top navigation bar includes the Astra logo, a live chat button, and the user's name 'Shubham Chaudhari'. The left sidebar contains navigation links for Dashboard, Databases, Streaming, and Sample App Gallery. The main content area is titled 'Dashboard / Backorder_Prediction' and includes tabs for Overview, Health, Connect, CQL Console, and Settings. The 'Overview' tab is active, showing usage metrics for the current billing period. Below the metrics, there is a 'Regions' section with a table listing the available regions. At the bottom, there is a 'Keyspaces' section with a table listing the existing keyspaces.

Read Requests	Write Requests	Storage Consumed	Data Transfer
96.3k	40.0k	409.7 KB	0.00

Provider	Region	Capacity Units	Cluster ID
AWS	us-east-1	1	fee9b492-b712-4557-9b49-12a4b209c555

Keyspace
Good_training_data

```

File Edit Selection View Go Run Terminal Help
Database_handler.py - BackOrderPrediction-Internship - Visual Studio Code

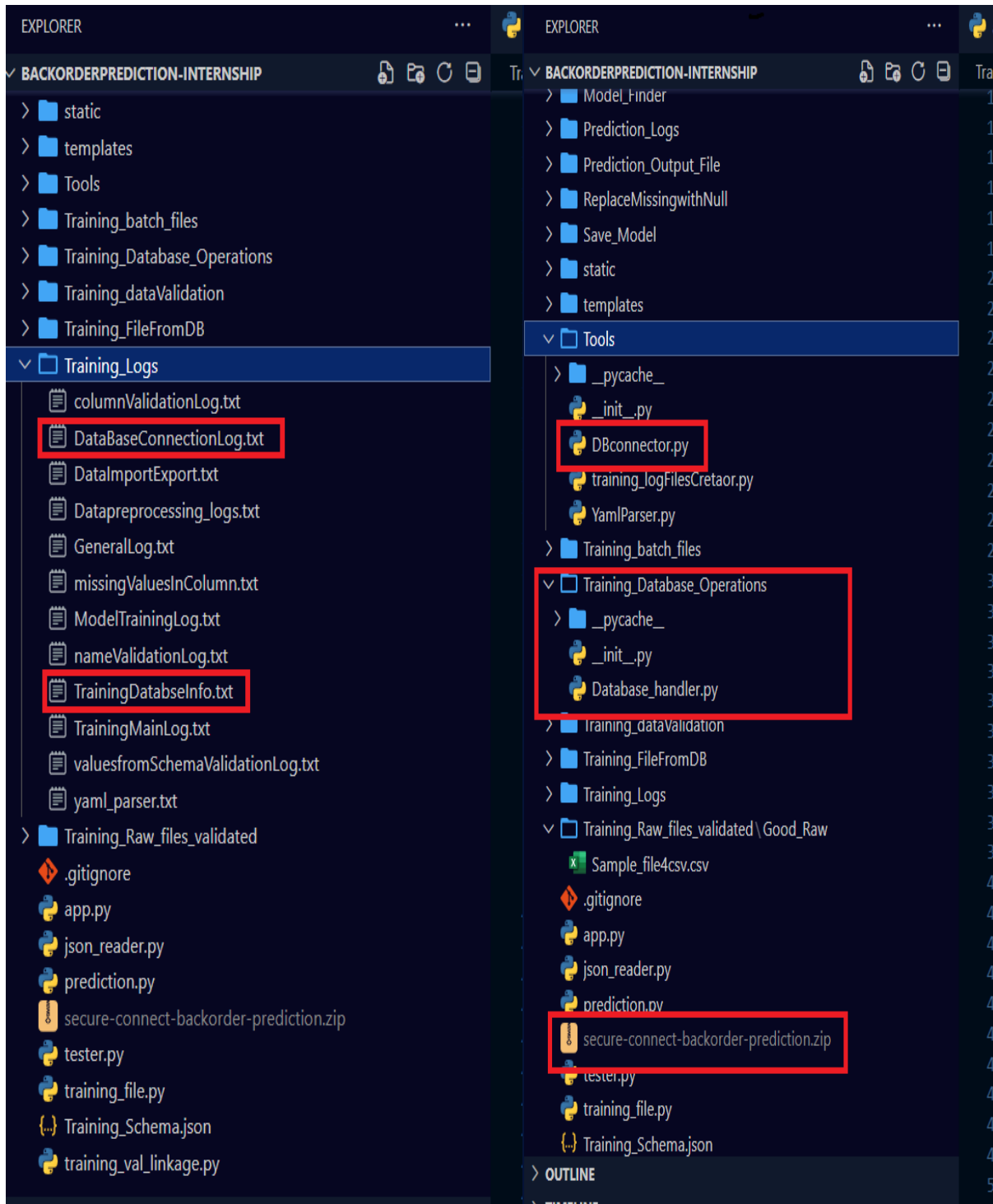
EXPLORER
BACKORDERPREDICTION-INTERNSHIP
  _pyscache_
  idea
  application_logging
  Controllers
  Data_Information
  data_ingestion
  data_perprocessing
  EDA
  Model
  Model_Finder
  Prediction_Logs
  Prediction_Output_File
  ReplaceMissingwithNull
  Save_Model
  static
  templates
  Tools
  Training_batch_files
  Training_Database_Operations
    _pyscache_
    Database_handler.py
    Training_dataValidation
    Training_FileFromDB
    Training_Logs
    Training_Raw_files_validated \ Good_Raw
      Sample_file4csv.csv
      .gitignore
      app.py
      json_reader.py
  OUTLINE
  TIMELINE

Database_handler.py
14 self.goodfilepath = "Training_Raw_files_validated/Good_Raw"
15 self.yaml_path = "Controllers/DBconnection_info.yaml"
16 self.logger = logger_app()
17 self.session = DBconnector().connect()
18 self.key_space = YamlParser(self.yaml_path).yaml_parser()[0]['Good_training_tables_info']['keyspace_name']
19 self.Good_training_TableName = YamlParser(self.yaml_path).yaml_parser()[0]['Good_training_tables_info']['table_name']
20
21 def CreateGoodTraining_table(self):
22     """ Task : this method will create the GoodTraining Table to store the Validated data.
23     Inputs : Receives the Inputs from "DBconnection_info.yaml" files
24     Ip/1 : database connector pointer
25     Ip/2 : Good-Training-TableName
26     Ip/3 : particular keyspace
27     return : None
28     by : shubham chau
29     """
30     try :
31         self.f_ = open('Training_logs/TrainingDatabaseInfo.txt', 'a')
32         self.logger.log(self.f_, "Entered into the class : CreateGoodTraining_table ....")
33         self.session.execute(f"USE {self.key_space}")
34         self.session.execute(f"CREATE TABLE IF NOT EXISTS {self.Good_training_TableName} ( sku INT PRIMARY KEY,")
35         self.logger.log(self.f_, f"{self.Good_training_TableName} created Successfully !....")
36         self.f_.close()
37     except Exception as e :
38         self.logger.log(file_object= self.f_ , log_message= "Exception : " + str(e))
39         self.f_.close()
40         raise e
41
42 def DataInsertion(self):
43     """
44     Method Name: DataInsertion
45     Description: This method inserts the Good data files from the Good_Raw folder into the
46                 above created table.
47     Output: None
48     On Failure: Raise Exception
49     by : shubham chau.
50
51 """

```

3.5. Export Data from Database

- ❖ Data Export from Database - The data in a stored database is exported as a CSV file to be used for Data Pre-processing and Model Training.



3.6. Data Pre-processing

Data Pre-processing steps we could use are Null value handling, Categorical to Numerical Transformation of columns , Splitting Data into Dependent and Independent Features , Robust Scaling , Remove those columns which are does not participate in model building Processes , Imbalanced data set handling, Handling columns with standard deviation zero or below a threshold, etc.

3.7. Model Creation / Model Building

After cleaning the data and completing the feature Engineering/ data Per processing. we have done splitted data in the train data and test data using method build in pre-processing file and implemented various Classification Algorithm like RandomForestClassifier and XgBoostClassifier also calculated their accuracies on test data and train data.

	Logistic Regression	XGBoost Classifier	Support Vector Classifier	Random Forest	KNearst Classifier	Gaussian Naive Bayes	Best Score
Accuracy	0.941960	0.972819	0.945435	0.983600	0.978644	0.935421	Random Forest
Precision	0.989575	0.985004	0.991883	0.994982	0.992221	0.983982	Random Forest
Recall	0.893252	0.960225	0.898160	0.972086	0.964826	0.885174	Random Forest
F1 Score	0.938937	0.972445	0.942686	0.983391	0.978320	0.931943	Random Forest

3.8. Hyperparameter Tuning

In hyperparameter tuning we have implemented randomized search cv or grid search cv and from that we also implemented cross validation techniques for that. From that we have choose best parameters according to hyperparameter tuning and best score from their accuracies so we got 99% accuracy in our random forest classifier after hyper parameter tuning.

3.9. Model Dump

After comparing all accuracies and checked all ROC, AUC curve accuracy we have choose hyper parameterized random forest classifier and xgboost classifiers as our best model by their results so we have dumped this model in a pickle file format with the help of `load_model` and `find_best_model` method build in `save_model.py` python module.

3.10. Data from User

Here we will collect user's requirement to predict whether a product in backorder or not their forecast_3_month, forecast_6_month, forecast_9_month, sales_3_month, sales_6_month, sales_9_month, perf_6_month_avg, perf_12_month_avg .

3.11. Data Validation

Here Data Validation will be done, given by the user.

3.12. Model Call for specific input

Based on the User input will be throwing to the backend in the variable format then it converted into pandas data frame then we are loading our pickle file in the backend and predicting whether product come in backorder or not as an output and sending to our html page.


3.13. Saving Output in .csv file

After calling model Recipe/Output will be recommended, this output will be saved in .csv file and it will be used to show the Output.

3.14. User Interface

In Frontend creation we have made a user interactive page where user can enter their input values to our application. In these frontend page we have made a form which has beautiful styling with CSS and bootstrap. These HTML user input data is transferred in variable format to backend. Made these html fully in a decoupled format.

Input page :-



Project: Back Order Classification
Using Machine Learning, Python, Flask And Cassandra
Made By Vikas And its Team Member

- Vikas
- Chaudhary Shubham R.
- Bijayalaxmi Kar
- Hitesh Singh
- Pallapothu Bhargavam
- Vivek Rathore

Machine Learning App with Flask

Enter Your Details

Enter Sales Quantity For The Prior 1 Month Time Period

Enter Sales Quantity For The Prior 3 Month Time Period

Enter Sales Quantity For The Prior 6 Month Time Period

Enter Sales Quantity For The Prior 9 Month Time Period

Enter Forecast Sales For The Next 3 Months

Enter Forecast Sales For The Next 3 Months

Enter Forecast Sales For The Next 6 Months

Enter Forecast Sales For The Next 9 Months

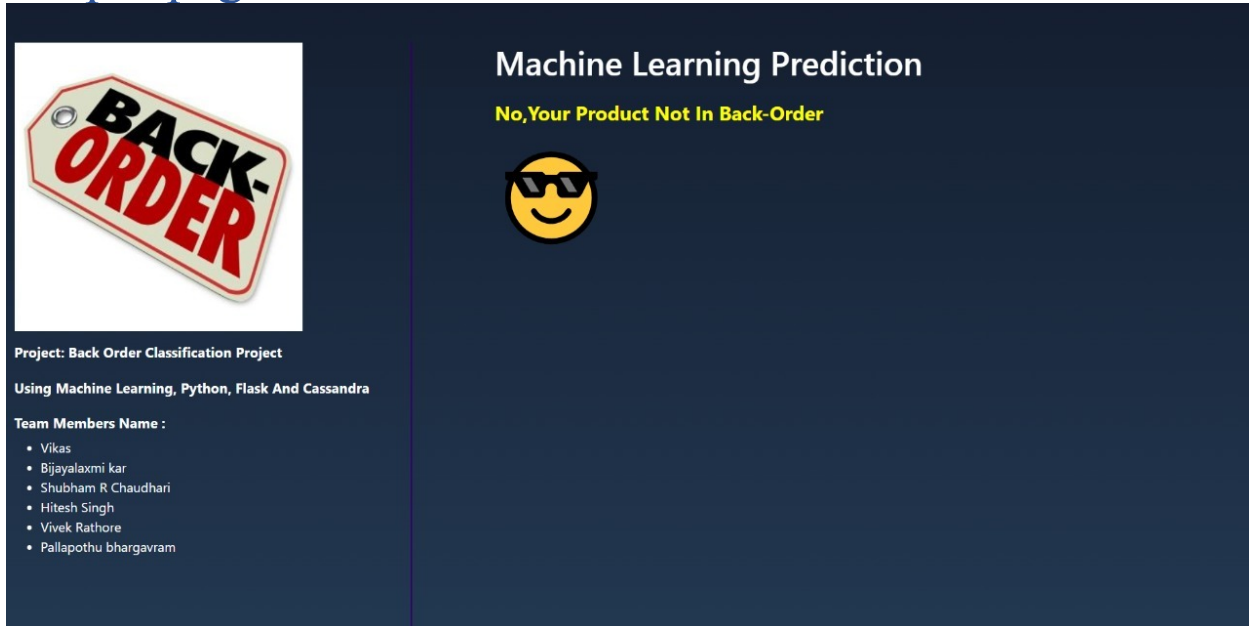
Enter Source Performance For Prior 6 Month Period

Enter Source Performance For Prior 9 Month Period

Predict

Created by VIKAS © 2021

Output page :-



Machine Learning Prediction

No,Your Product Not In Back-Order

Project: Back Order Classification Project

Using Machine Learning, Python, Flask And Cassandra

Team Members Name :

- Vikas
- Bijayalaxmi kar
- Shubham R Chaudhari
- Hitesh Singh
- Vivek Rathore
- Pallapothu bhargavram

3.15. Deployment

We will be deploying the model with the help of AWS cloud platforms.

This is a workflow diagram for the Back-Order Prediction Application



4. Technology Stack

Front End	HTML, CSS, Bootstrap File
Back End	Flask, Pandas, NumPy, scikit-learn etc
Database	Cassandra
Deployment	AWS

5.Directory Tree Structure

```
└── Backorder-Prediction
    ├── application_logging
    │   └── logging.py
    ├── Controllers
    │   └── DBconnection_info.yaml
    ├── Data_Information
    │   └── Null_Values.csv
    ├── data_ingestion
    │   ├── __pycache__
    │   └── data_loader.py
    ├── data_preprocessing
    │   ├── __pycache__
    │   └── preprocessing.py
    ├── EDA
    │   ├── EDA-part1.ipynb
    │   └── EDA-part2.ipynb
    ├── model
    │   └── Random Forest
    │       └── Random Forest.sav
    ├── model_finder
    │   ├── __pycache__
    │   └── Model.py
    ├── Prediction_logs
    │   └── predictionlog.txt
```

- |— Prediction_Output_File
 - | — Predictions.csv
- |— ReplaceMissingwithNull
 - | — __pycache__
 - | — __init__.py
 - | — transformer.py
- |— Save_Models
 - | — __pycache__
 - | — save_methods.py
- |— static
 - | — backorder.jpg
 - | — Style1.css
 - | — Style2.css
- |— Templates
 - | — index.html
 - | — predict.html
- |— Tools
 - | — __pycache__
 - | — __init__.py
 - | — DBconnector.py
 - | — training_logFilescreator.py
 - | — YamlParser.py
- |— Traning_batch_Files
 - | — BackOrder_08012020_120000.csv
- |— Training_Database_operations

- | | — __init__.py
- | | — __pycache__
- | | — Database_handler.py
- | — Traininig_dataValidation
- | | — __pycache__
- | | — RawtrainingValidation.py
- | — Training_FilesfromDB
- | | — InputFile.csv
- | — Training_Logs
- | | — columnValidationLog.txt
- | | — DataBaseConnectionLog.txt
- | | — Datapreprocessing_logs.txt
- | | — GeneralLog.txt
- | | — TmissingValuesInColumn.txt
- | | — ModelTrainingLog.txt
- | | — nameValidationLog.txt
- | | — TrainingDatabseInfo.txt
- | | — TrainingMainLog.txt
- | | — valuesfromSchemaValidationLog.txt
- | | — yaml_parser.txt
- | — Training_Raw_Files_Validated
- | | — BackOrder_08012020_120000.csv
- | — training_val_linkage.py
- | — training_file.py
- | — app.py

- |— prediction.py
- |— requierements.txt
- |— Training_Schema.json
- |— secure-connect-backorder-prediction.zip

Low Level Document

6.Unit Test Case

Test Case Description	Pre – Requisite	Expected Resulted
Verify whether the Application URL is accessible to the user	1. Application URL should be defined	Application URL should be accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed	1. Application URL is accessible 2. Application is deployed	The Application should load completely for the user when the URL is accessed
Verify Response time of URL from backend model.	1. Application is accessible	The latency and accessibility of application is very faster we got in AWS ec2 service.
Verify whether user is giving standard input.	1. Handled test cases at backends.	User should be able to see successfully valid results.
Verify whether user is able to see input fields on logging in	1. Application is accessible 2. User is logged in to the application	User should be able to see input fields on logging in
Verify whether user is able to edit all input fields	1. Application is accessible 2. User is logged in to the application	User should be able to edit all input fields
Verify whether user gets Predict Back-Order button to submit the inputs	1. Application is accessible 2. User is logged in to the application	User should get Submit button to submit the inputs

Verify whether user is presented with recommended results on clicking submit	1. Application is accessible 2. User is logged in to the application	User should be presented with recommended results on clicking submit
Verify whether the recommended results are in accordance to the selections user made	1. Application is accessible 2. User is logged in to the application and database	The recommended results should be in accordance to the selections user made
Verify whether user has options to filter the recommended results as well	1. Application is accessible 3. User is logged in to the application	User should have options to filter the recommended results as well
Verify whether the KPIs indicate details Of the correct inputs	1. Application is accessible 2. User is logged in to the application	The KPIs should indicate details of the suggested inputs to users.