**Project Report Format**

**TEAM ID** : PNT2022TMID52458

**INTRODUCTION**

1.1 **Project Overview**

Machine learning algorithms can be used by businesses to as accurately predict changes in consumer demand as feasible. These algorithms are capable of automatically recognising patterns, locating intricate links in big datasets, and picking up indications for changing demand. A food delivery service has to deal with a lot of perishable raw materials which makes it all, the most important factor for such a company is to accurately forecast daily and weekly demand. Too much inventory in the warehouse means more risk of wastage, and not enough could lead to out-ofstocks - and push customers to seek solutions from your competitors. The replenishment of majority of raw materials is done on weekly basis and since the raw material is perishable, the procurement planning is of utmost importance, the task is to predict the demand for the next 10 weeks

1.2 **Purpose**

.The main aim of this project is to create an appropriate machine learning model to forecast the number of orders to gather raw materials for next ten weeks. To achieve this, we should know the information about of fulfilment center like area, city etc., and meal information like category of food sub category of food price of the food or discount in particular week. By using this data, we can use any classification algorithm to forecast the quantity for 10 weeks. A web application is built which is integrated with the model built.

1. **LITERATURE SURVEY**
   1. Existing problem

There are lot more problems on ordering food over network and there is no proper demand for all the individual as well for the deployment, Consistent evaluation is also eradicated.

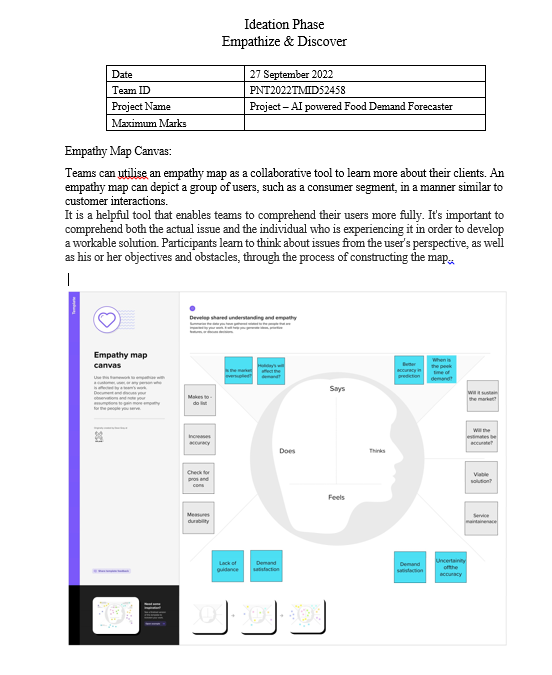
* 1. References

* + - AQUAREL
    - 09Solution
    - Kaggle

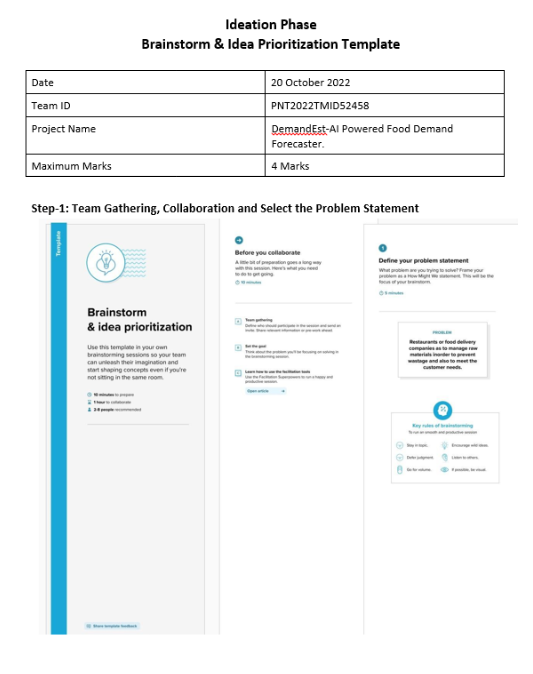
* 1. Problem Statement Definition

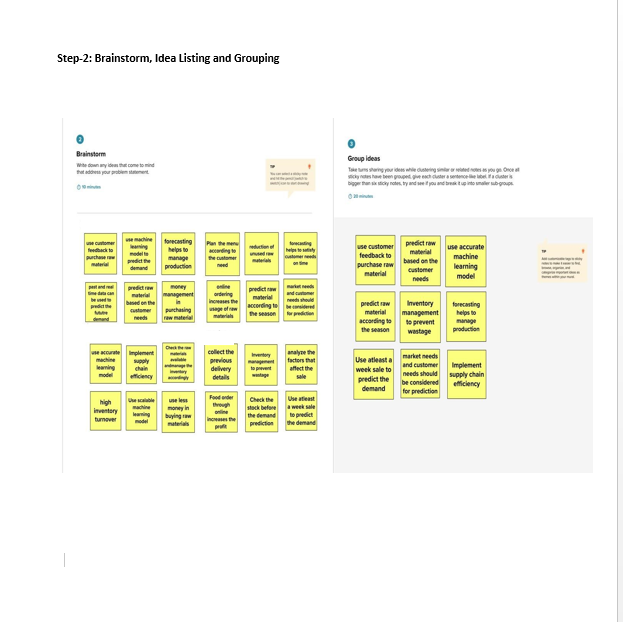
* + - The data set relates to a food delivery service that has operations throughout several cities. For delivering meal orders to clients, they have a number of fulfilment sites in these cities. The required raw materials are stocked appropriately at the fulfilment centers.

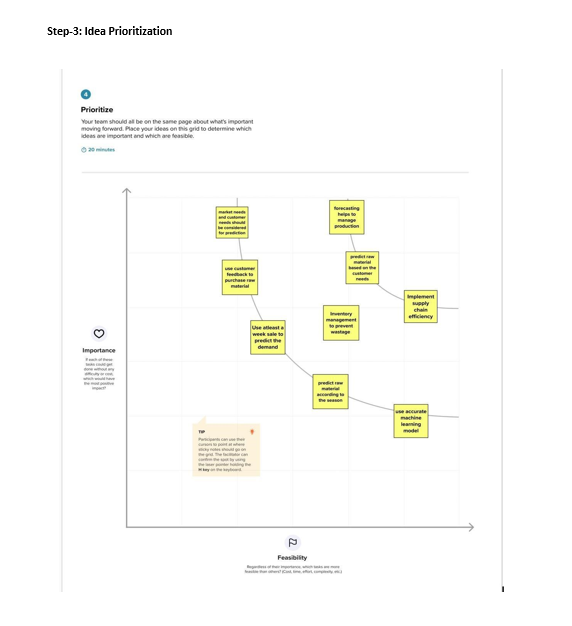
1. **IDEATION & PROPOSED SOLUTION**
   1. Empathy Map Canvas



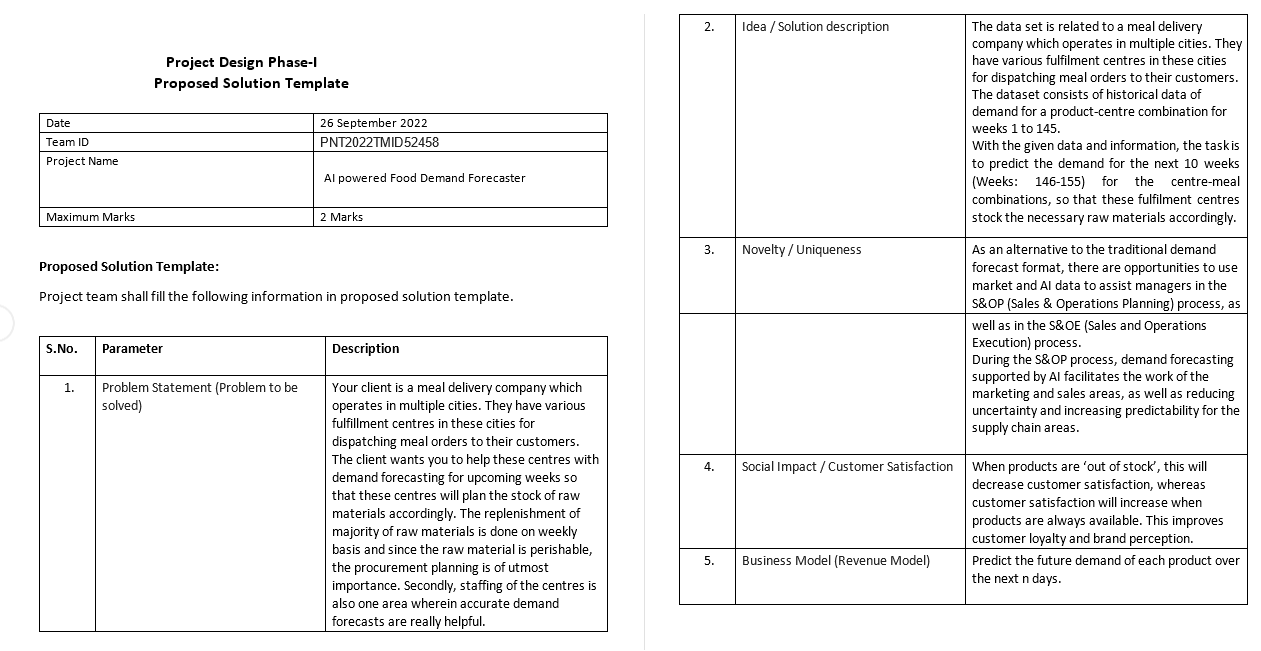
* 1. Ideation & Brainstorming



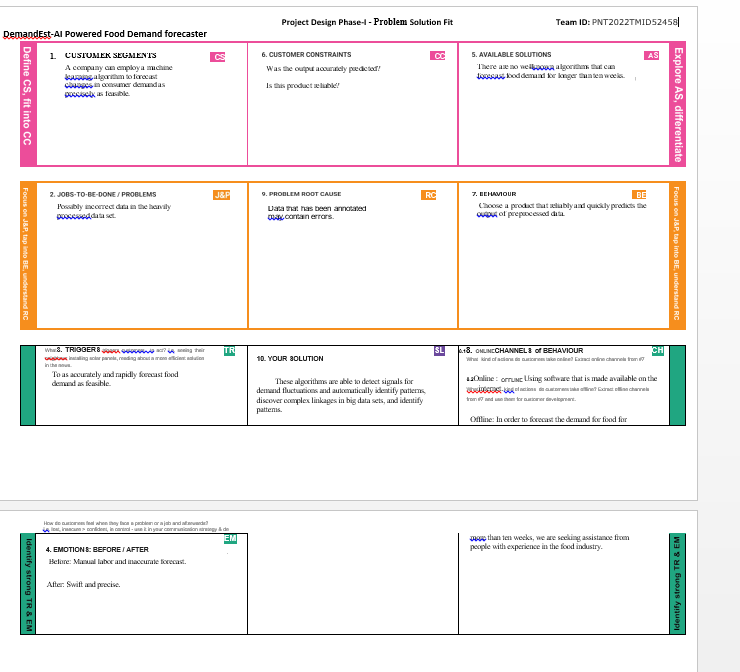




3.3Proposed Solution

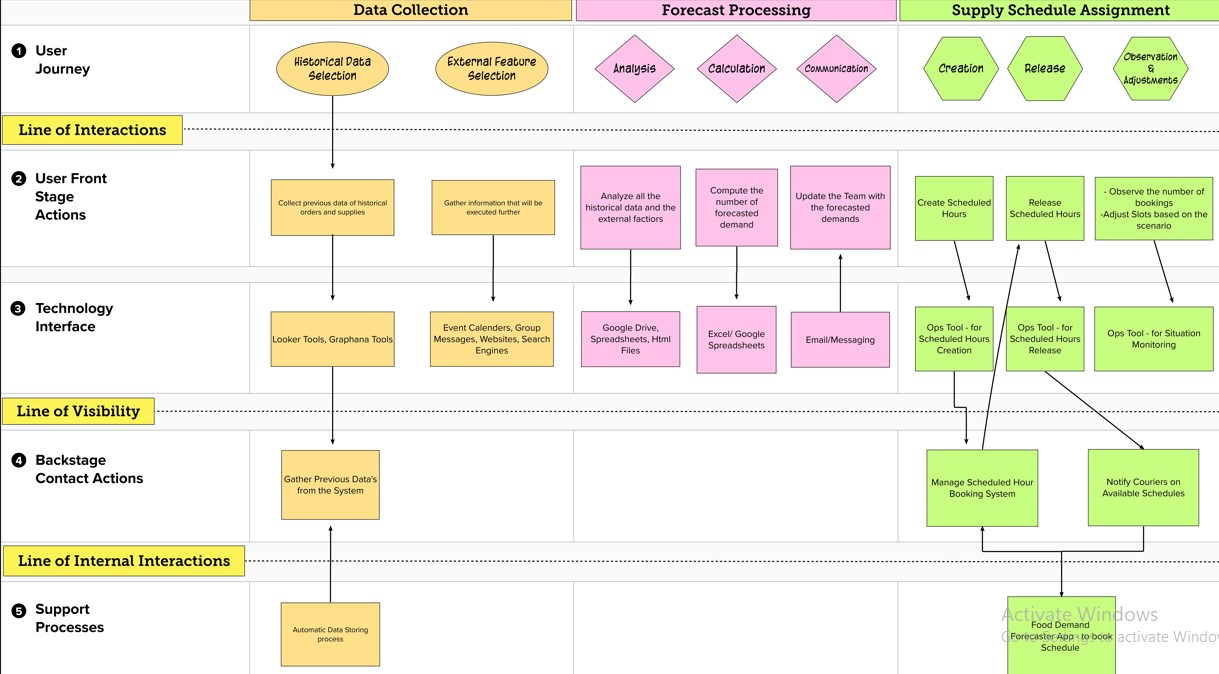


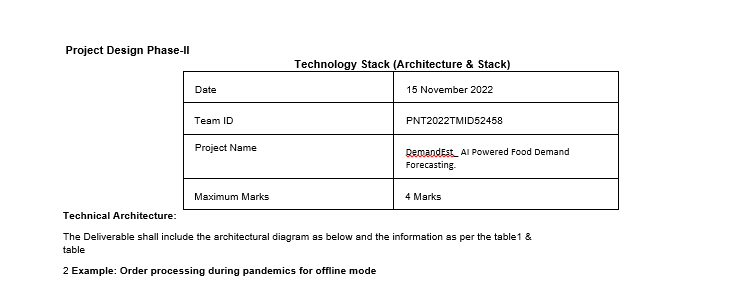
3.4 Problem Solution fit

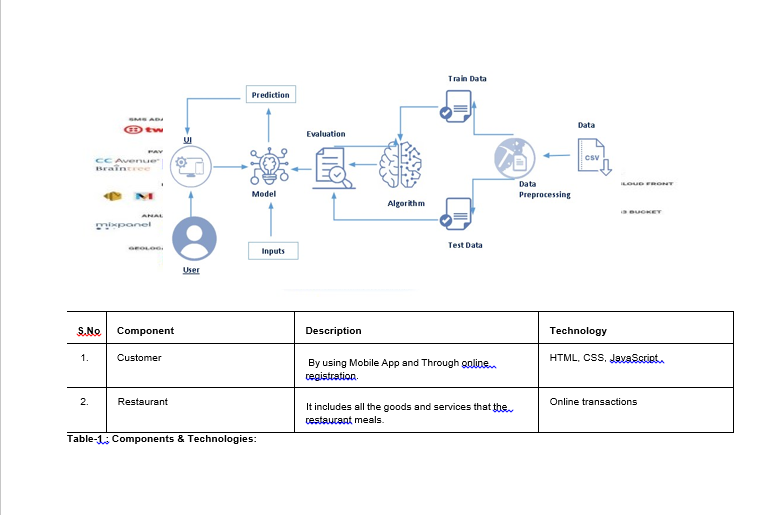


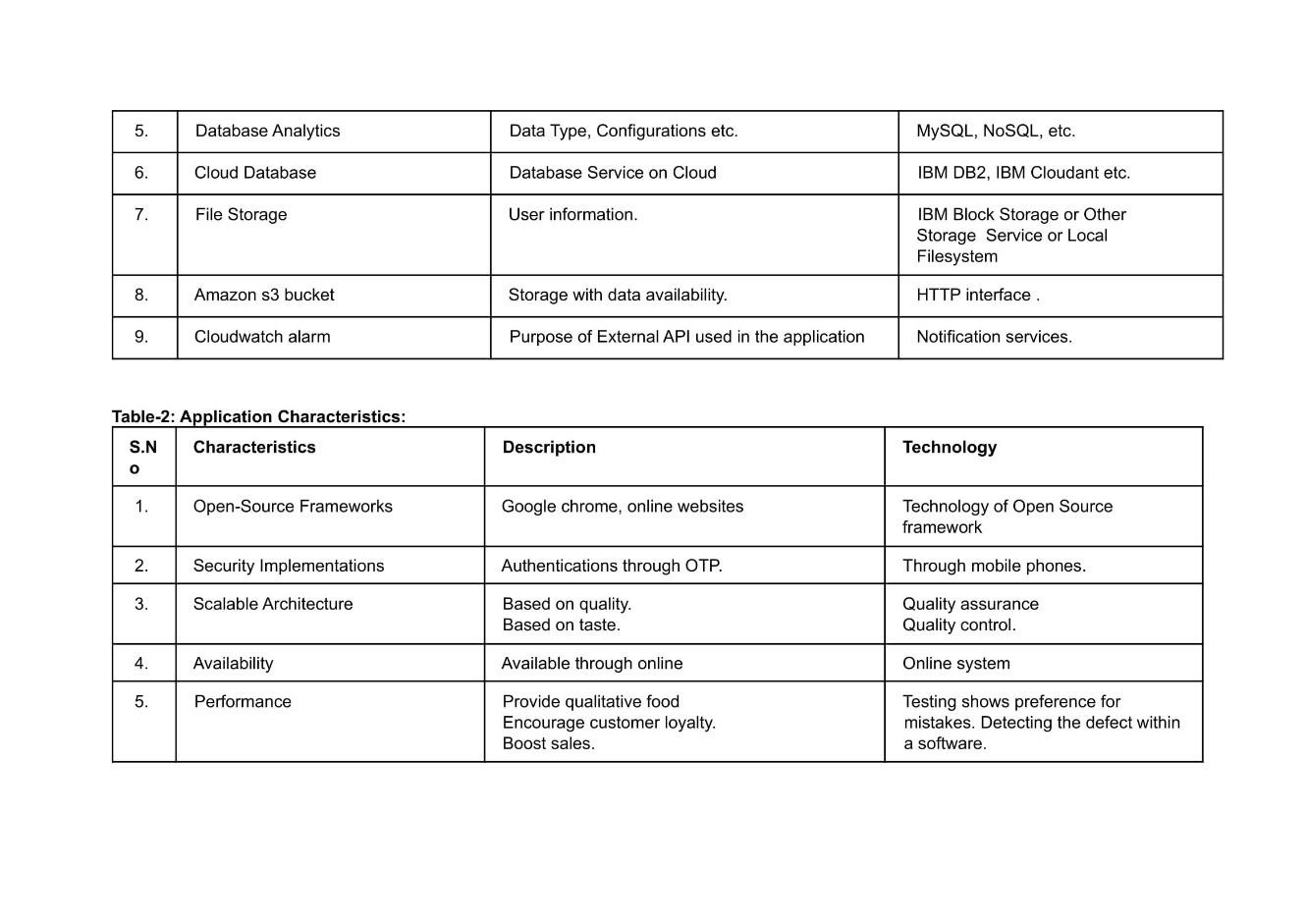
1. **PROJECT DESIGN** 
   1. Data Flow Diagrams

* 1. Solution & Technical Architecture

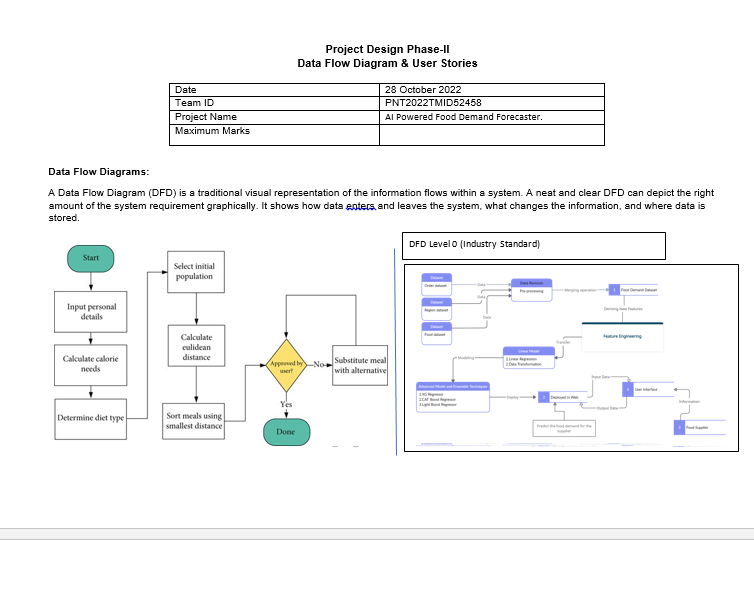


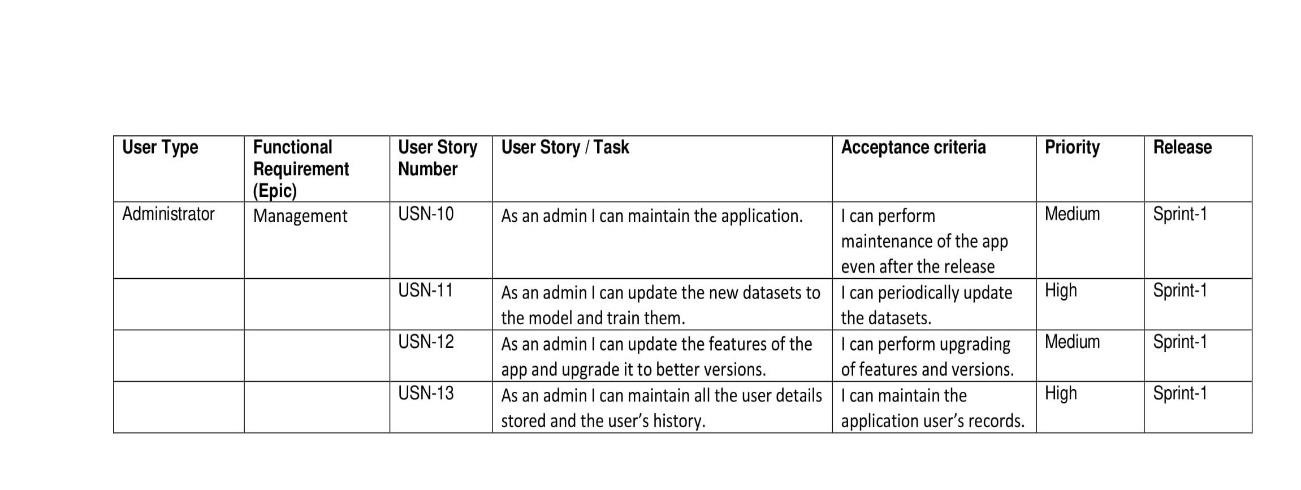
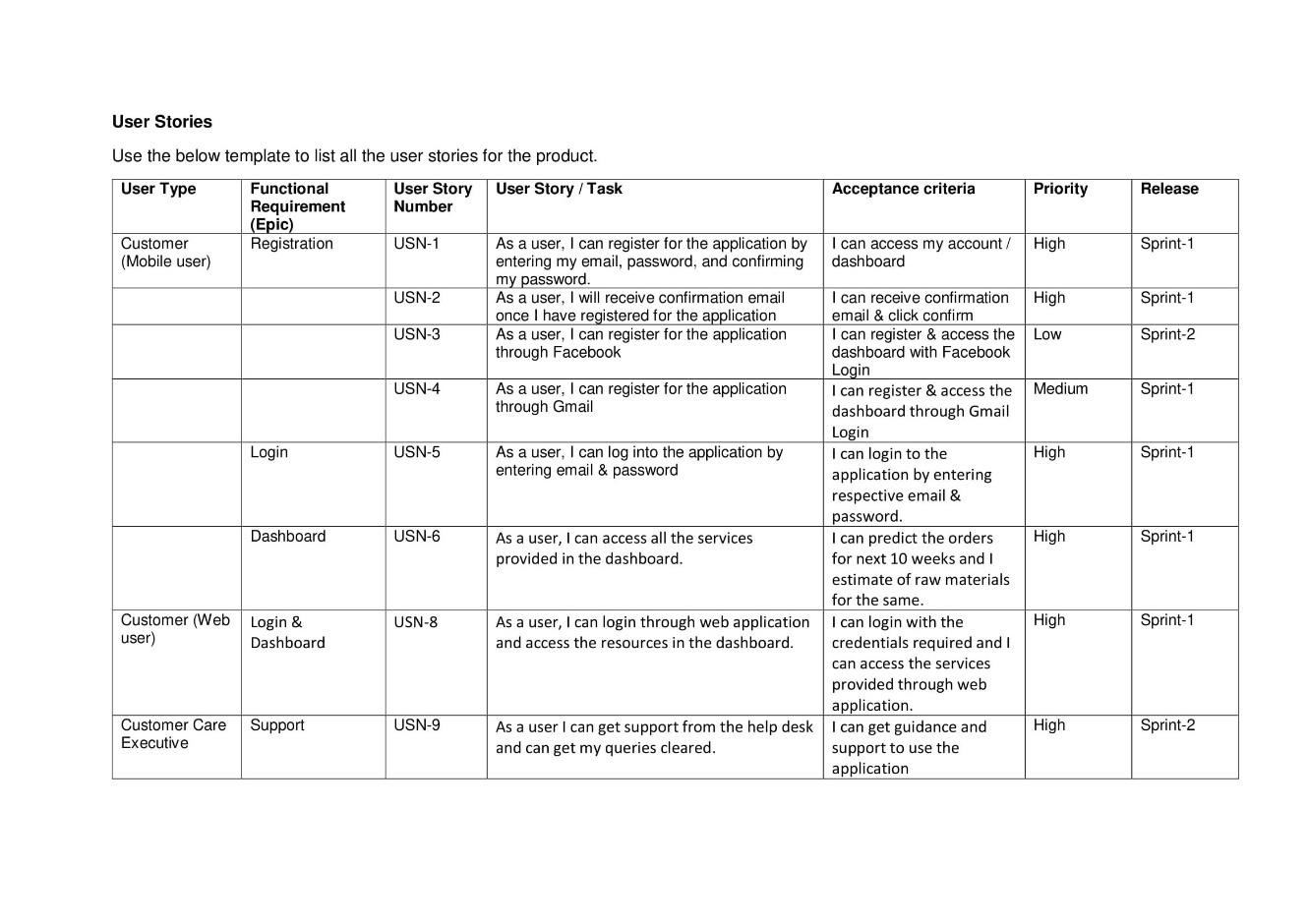






* 1. User Stories





1. **PROJECT PLANNING & SCHEDULING** 
   1. Sprint Planning & Estimation

SPRINT 1:

<!DOCTYP

E html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home</title>

<link type="text/css" rel="stylesheet" href="/Flask/static/style.css">

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link

href="https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;600;800&display=s wap" rel="stylesheet">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0beta2/css/all.min.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0beta2/css/v4-shims.min.css">

<style>

\*{

margin: 0; padding: 0;

font-family: 'Poppins', sans-serif;

}

.bar

{

margin: 0px; padding: 15px;

background-color:rgb(64, 100, 246); font-family:'Poppins',sans-serif;

font-size:25px;

} a{

color:#fff; float:right; text-decoration:none; padding-right:20px;

}

a:hover{

padding: 3.5px; background: #FAAE42;

}

.text-box{

width: 90%;

color:rgba(51, 210, 249, 0.905); text-shadow: #0c0d0e; position:absolute; top: 45%; left: 50%;

transform: translate(-50%,-50%); text-align: center;

}

.text-box h1{

font-size: 70px; text-shadow: 2px 2px 40px #ffffff;

}

.text-box p{

margin: 10px 0 40px; font-size: 25px; color: rgba(0, 0, 0, 0.946);

}

</style>

</head>

<body>

<section class="header">

<div class="bar">

<a href="/pred">Predict</a>

<a href="/home">Home</a>

<br>

</div>

<div class="text-box">

<h1>

DemandEst - AI powered Food Demand Forecaster</h1>

<p> The concept of a balance point between supply and demand is used to explain various situations in our

daily lives, from bread in the neighborhood bakery, which can be sold at the equilibrium price, which

equals the quantities desired by buyers and sellers, to the negotiation of securities of companies in the stock market.

On the supply side, a definition of the correct price to be practiced and mainly the quantity are common issues in the planning and execution of the strategy of several companies.</p>

</div>

</section>

</body>

</html>

ii)

<html lang="en"

>

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Predict</title>

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link

href="https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;600;800&display=swa p" rel="stylesheet">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0beta2/css/all.min.css">

<style>

.bar

{

margin: 0px; padding: 15px;

background-color:rgb(100, 5, 29); /\* opacity:0.6; \*/ font-family:'Poppins',sans-serif; font-size:25px;

} a {

color:#fff; float:right; text-decoration:none;

padding-right:20px;

}

a:hover{

padding: 3.5px;

background: #FAAE42;

}

h1{

color:rgb(100, 5, 29); font-family:Poppins; font-size:30

}

h2{

color:rgb(100, 5, 29); font-family: Poppins; font-size:60; margin-bottom: 10px;

}

.my-cta-button{

font-size: 20px; color: rgb(15, 15, 15); border: 1px solid #0e0e0ccf; padding: 3.5px;

cursor: pointer;

}

.my-cta-button:hover{ border: 2px solid #faae42; padding: 3.5px; background: #FAAE42;

} p {

color:white;

font-family: Poppins; font-size:30px;

}

</style>

</head>

<body>

<div class="bar">

<a href="/pred">Predict</a>

<a href="/home">Home</a>

<br>

</div>

<div class="container">

<center> <div id="content" style="margin-top:2em">

<h2><center>Food Demand Forecasting</center></h2>

<form action="{{ url\_for('predict') }}" method="POST">

<select id="homepage\_featured" name="homepage\_featured">

<option value="">homepage\_featured</option>

<option value="0">No</option>

<option value="1">Yes</option>

</select><br><br>

<select id="emailer\_for\_promotion" name="emailer\_for\_promotion">

<option value="">emailer\_for\_promotion</option>

<option value="0">No</option>

<option value="1">Yes</option>

</select><br><br>

<input class="form-input" type="text" name="op\_area" placeholder="Enter the op\_area(27)"><br><br>

<select id="cuisine" name="cuisine">

<option value="">Cuisine</option>

<option value="0">Continental</option>

<option value="1">Indian</option>

<option value="2">Italian</option>

<option value="3">Thai</option>

</select><br><br>

<input class="form-input" type="text" name="city\_code" placeholder="Enter city\_code"><br><br>

<input class="form-input" type="text" name="region\_code" placeholder="Enter region\_code"><br><br>

<select id="category" name="category">

<option value="">Category</option>

<option value="0">Beverages</option>

<option value="1">Biryani</option>

<option value="2">Desert</option>

<option value="3">Extras</option>

<option value="4">Fish</option>

<option value="5">Other Snacks</option>

<option value="6">Pasta</option>

<option value="7">Pizza</option>

<option value="8">Rice Bowl</option>

<option value="9">Salad</option>

<option value="10">Sandwich</option>

<option value="11">Seafood</option>

<option value="12">Soup</option>

<option value="13">Starters</option>

</select><br><br>

<input type="submit" class="my-cta-button" value="Predict">

</form>

<br>

<h1 class="predict">Number of orders: {{ prediction\_text }}</h1>

</div></center>

</div>

</body>

</body>

5.2 Sprint Delivery Schedule

SPRINT 2:-

import pandas as pd

import numpy as np import pickle import os

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_, template\_folder="templates")

@app.route('/', methods=['GET'])

def index():

return render\_template('home.html')

@app.route('/home', methods=['GET']) def about():

return render\_template('home.html')

@app.route('/pred', methods=['GET']) def page():

return render\_template('upload.html')

@app.route('/predict', methods=['GET', 'POST']) def predict():

print("[INFO] loading model...") model = pickle.load(open('foodDemand.pkl', 'rb')) input\_features = [float(x) for x in request.form.values()] features\_value = [np.array(input\_features)] print(features\_value)

features\_name = ['homepage\_featured', 'emailer\_for\_promotion', 'op\_area', 'cuisine',

'city\_code', 'region\_code', 'category'] prediction = model.predict(features\_value) output = prediction[0] print(output)

return render\_template('upload.html', prediction\_text=output)

if \_\_name\_\_ == '\_\_main\_\_': app.run(debug=False)

ii) ibmapp:

|  |  |
| --- | --- |
|  |  |
| # import the necessary packages |  |
|  | import pandas as pd |
|  | import numpy as np |
|  | import pickle |
|  | import os |
|  | import requests |
|  |  |
|  | # NOTE: you must manually set API\_KEY below using information retrieved from your IBM Cloud account. |
|  | API\_KEY = "68w9XBNJLBQFtHM2rG\_aouV4LmlF-EtecYrhIQBQbt\_K" |
|  | token\_response = requests.post('https://iam.cloud.ibm.com/identity/token', |
|  | data={"apikey": API\_KEY, "grant\_type": 'urn:ibm:params:oauth:granttype:apikey'}) |
|  | mltoken = token\_response.json()["access\_token"] |
|  |  |
|  | header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken} |
|  |  |
|  | from flask import Flask, request, render\_template |
|  |  |
|  | app = Flask(\_\_name\_\_, template\_folder="templates") |
|  |  |
|  |  |
|  | @app.route('/', methods=['GET']) |
|  | def index(): |
|  | return render\_template('home.html') |
|  |  |
|  |  |
|  | @app.route('/home', methods=['GET']) |

def about():

return render\_template('home.html')

@app.route('/pred', methods=['GET']) def page():

return render\_template('upload.html')

@app.route('/predict', methods=['GET', 'POST']) def predict():

print("[INFO] loading model...")

# model = pickle.load(open('fdemand.pkl', 'rb')) input\_features = [int(x) for x in request.form.values()] print(input\_features) features\_value = [[np.array(input\_features)]] print(features\_value)

payload\_scoring = {"input\_data": [{"field": [['homepage\_featured', 'emailer\_for\_promotion',

'op\_area', 'cuisine',

'city\_code', 'region\_code', 'category']],

"values": [input\_features]}]}

response\_scoring = requests.post(

'https://us-south.ml.cloud.ibm.com/ml/v4/deployments/80afcaad-591d-4869-bf54-

17bbb8c70ea3/predictions?version=2022-11-14',

json=payload\_scoring, headers={'Authorization': 'Bearer ' + mltoken}) print("Scoring response") print(response\_scoring.json()) predictions = response\_scoring.json() print(predictions)

print('Final Prediction Result', predictions['predictions'][0]['values'][0][0]) pred = predictions['predictions'][0]['values'][0][0]

# prediction = model.predict(features\_value)

# output=prediction[0]

# print(output) print(pred) return render\_template('upload.html', prediction\_text=pred)

if \_\_name\_\_ == '\_\_main\_\_': app.run(debug=False)

iii) main.py:-

import numpy

as np

import pandas as pd import plotly.express as px import matplotlib.pyplot as plt import seaborn as sns

from sklearn.preprocessing import OneHotEncoder, StandardScaler from sklearn.model\_selection import train\_test\_split from sklearn import metrics from sklearn.pipeline import make\_pipeline from sklearn.ensemble import RandomForestRegressor

import warnings

warnings.filterwarnings('ignore')

# Importing Raw Files train\_raw = pd.read\_csv('train.csv') test\_raw = pd.read\_csv('test.csv') meal = pd.read\_csv('meal\_info.csv') centerinfo = pd.read\_csv('fulfilment\_center\_info.csv')

# Analyzing Data

print("The Shape of Demand dataset :", train\_raw.shape) print("The Shape of Fulfillment Center Information dataset :", centerinfo.shape) print("The Shape of Meal information dataset :", meal.shape) print("The Shape of Test dataset :", test\_raw.shape) train\_raw.head() centerinfo.head() meal.head() test\_raw.head()

# Check for missing values train\_raw.isnull().sum().sum() test\_raw.isnull().sum().sum()

# Analysis report

print("The company has", centerinfo["center\_id"].nunique(), " warehouse ", "spreed into ", centerinfo["city\_code"].nunique(), "City and ", centerinfo["region\_code"].nunique(), "Regions") print("The products of the company are ", meal["meal\_id"].nunique(), "unique meals , divided into ", meal["category"].nunique(), "category and ", meal["cuisine"].nunique(), "cuisine")

# Merge meal,center-info data with train and test data train = pd.merge(train\_raw, meal, on="meal\_id", how="left") train = pd.merge(train, centerinfo, on="center\_id", how="left") print("Shape of train data : ", train.shape)

train.head()

# Merge test data with meal and center info test = pd.merge(test\_raw, meal, on="meal\_id", how="outer") test = pd.merge(test, centerinfo, on="center\_id", how="outer") print("Shape of test data : ", test.shape)

test.head()

# Typecasting to assign appropriate data type to variables col\_names = ['center\_id', 'meal\_id', 'category', 'cuisine', 'city\_code', 'region\_code', 'center\_type'] train[col\_names] = train[col\_names].astype('category') test[col\_names] = test[col\_names].astype('category') print("Train Datatype\n", train.dtypes) print("Test Datatype\n", test.dtypes) # Orders by centers

center\_orders = train.groupby("center\_id", as\_index=False).sum() center\_orders = center\_orders[["center\_id", "num\_orders"]].sort\_values(by="num\_orders", ascending=False).head(10) fig = px.bar(x=center\_orders["center\_id"].astype("str"), y=center\_orders["num\_orders"], title="Top 10

Centers by Order",

labels={"x": "center\_id", "y": "num\_orders"})

fig.show()

# Pie chart on food category fig = px.pie(values=train["category"].value\_counts(), names=train["category"].unique(),

title="Most popular food category") fig.show()

# Orders by Cuisine types cuisine\_orders = train.groupby(["cuisine"], as\_index=False).sum() cuisine\_orders = cuisine\_orders[["cuisine", "num\_orders"]].sort\_values(by="num\_orders", ascending=False)

fig = px.bar(cuisine\_orders, x="cuisine", y="num\_orders", title="orders by cuisine") fig.show()

# Impact of check-out price on order train\_sample = train.sample(frac=0.2) fig = px.scatter(train\_sample, x="checkout\_price", y="num\_orders", title="number of order change with checkout price")

fig.show()

sns.boxplot(train["checkout\_price"]) # Orders weekly trend

week\_orders = train.groupby(["week"], as\_index=False).sum() week\_orders = week\_orders[["week", "num\_orders"]] fig = px.line(week\_orders, x="week", y="num\_orders", markers=True, title="Order weekly trend") fig.show()

# Deriving discount percent and discount y/n train['discount percent'] = ((train['base\_price'] - train['checkout\_price']) / train['base\_price']) \* 100

# Discount Y/N

train['discount y/n'] = [1 if x > 0 else 0 for x in (train['base\_price'] - train['checkout\_price'])]

# Creating same feature in test dataset test['discount percent'] = ((test['base\_price'] - test['checkout\_price']) / test['base\_price']) \* 100 test['discount y/n'] = [1 if x > 0 else 0 for x in (test['base\_price'] - test['checkout\_price'])] train.head(2)

# Check for correlation between numeric features plt.figure(figsize=(13, 13)) sns.heatmap(train.corr(), linewidths=.1, cmap='Reds', annot=True) plt.title('Correlation Matrix')

plt.show()

# Define One hot encoding function def one\_hot\_encode(features\_to\_encode, dataset):

encoder = OneHotEncoder(sparse=False) encoder.fit(dataset[features\_to\_encode]) encoded\_cols = pd.DataFrame(encoder.transform(dataset[features\_to\_encode]),

columns=encoder.get\_feature\_names()) dataset = dataset.drop(columns=features\_to\_encode) for cols in encoded\_cols.columns:

dataset[cols] = encoded\_cols[cols] return dataset

# get list of categorical variables in data set ls = train.select\_dtypes(include='category').columns.values.tolist()

# Run one-hot encoding on all categorical variables features\_to\_encode = ls data = one\_hot\_encode(features\_to\_encode, train) data = data.reset\_index(drop=True) # Train-Validation Data Split y = data[["num\_orders"]]

X = data.drop(["num\_orders", "id", "base\_price", "discount y/n"], axis=1)

X = X.replace((np.inf, -np.inf, np.nan), 0) # replace nan and infinity values with 0

# 20% of train data is used for validation

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.20, random\_state=100)

# Prepare test data post applying onehot encoding OH\_test = one\_hot\_encode(features\_to\_encode, test) test\_final = OH\_test.drop(["id", "base\_price", "discount y/n"], axis=1)

# Create pipeline for scaling and modeling

RF\_pipe = make\_pipeline(StandardScaler(), RandomForestRegressor(n\_estimators=100, max\_depth=7))

# Build Model

RF\_pipe.fit(X\_train, y\_train)

# Predict Value

RF\_train\_y\_pred = RF\_pipe.predict(X\_val)

# Model Evaluation-

print('R Square:', RF\_pipe.score(X\_val, y\_val)) print('RMSLE:', 100 \* np.sqrt(metrics.mean\_squared\_log\_error(y\_val, RF\_train\_y\_pred)))

# Applying algorithm to predict orders test\_y\_pred = RF\_pipe.predict(test\_final) Result = pd.DataFrame(test\_y\_pred) print(Result.values)

Result = pd.DataFrame(test\_y\_pred)

Submission = pd.DataFrame(columns=['id', 'num\_orders'])

Submission['id'] = test['id']

Submission['num\_orders'] = Result.values

Submission.to\_csv('My submission.csv', index=False) print(Submission.shape)

print(Submission.head())

iv) ibm.py:-

import array as arr

import numpy as np import json

import requests from json import JSONEncoder

class NumpyEncoder(JSONEncoder): def default(self, obj): if isinstance(obj, np.ndarray):

return obj.tolist() return JSONEncoder.default(self, obj)

# NOTE: you must manually set API\_KEY below using information retrieved from your IBM Cloud account.

API\_KEY = "68w9XBNJLBQFtHM2rG\_aouV4LmlF-EtecYrhIQBQbt\_K"

token\_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey": API\_KEY, "grant\_type": 'urn:ibm:params:oauth:grant-type:apikey'}) mltoken = token\_response.json()["access\_token"]

header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}

values = np.ndarray([0, 0, 3, 1, 647, 56, 11]) print(values.shape)

# NOTE: manually define and pass the array(s) of values to be scored in the next line payload\_scoring = json.dumps({"input\_data": [{"field": [['homepage\_featured', 'emailer\_for\_promotion',

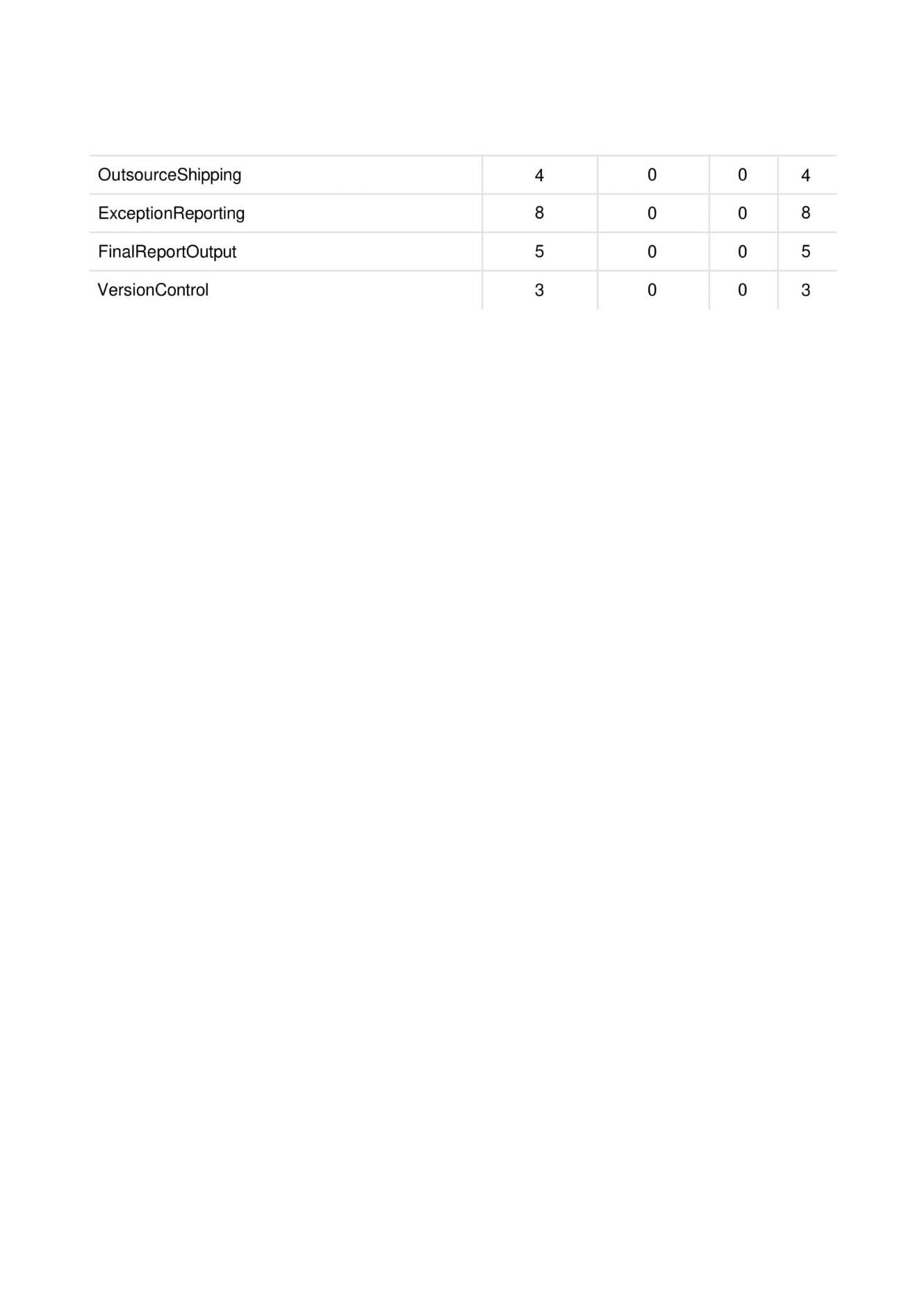
'op\_area', 'cuisine', 'city\_code', 'region\_code', 'category']], "values": [[0, 0, 3, 1, 647, 56, 11], [1, 1, 2, 3, 600, 46, 19]]}]},cls=NumpyEncoder)

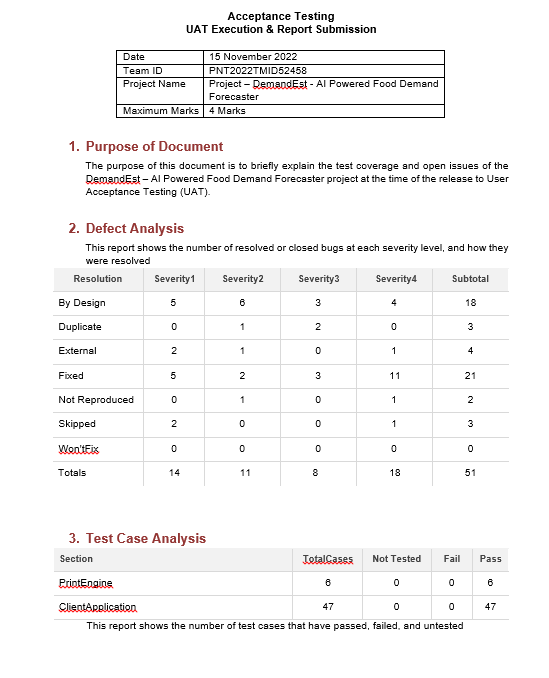
response\_scoring = requests.post(

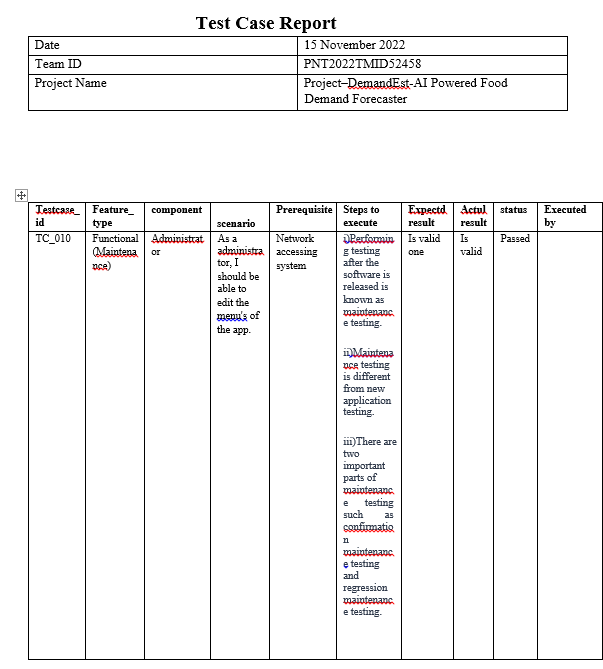
'https://us-south.ml.cloud.ibm.com/ml/v4/deployments/80afcaad-591d-4869-bf54-

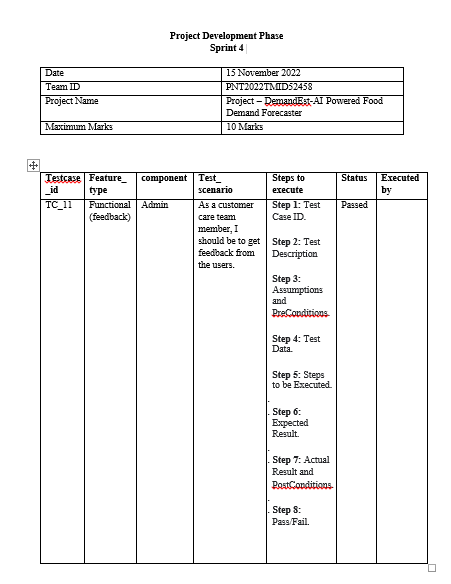
17bbb8c70ea3/predictions?version=2022-11-14', json=payload\_scoring, headers={'Authorization': 'Bearer ' + mltoken}) print("Scoring response") predictions = response\_scoring.json() for i in predictions: print(i, predictions[i])

5.3 Reports from JIRA

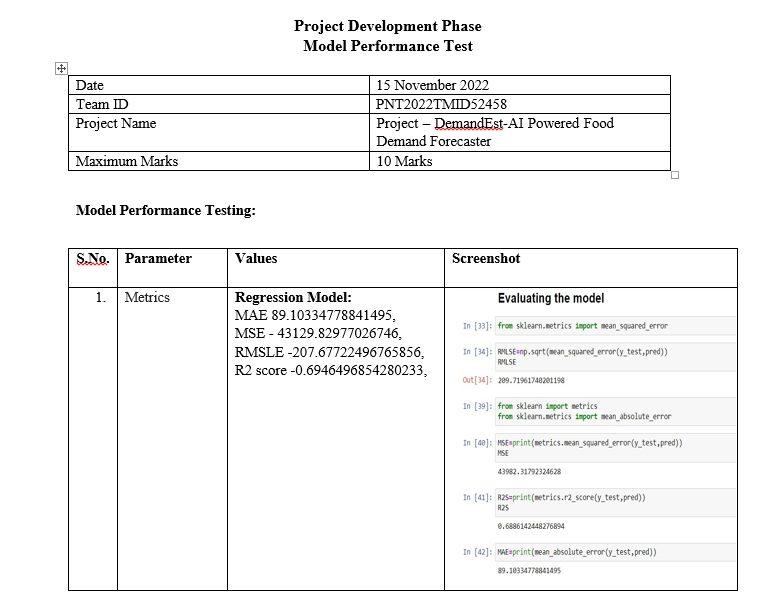


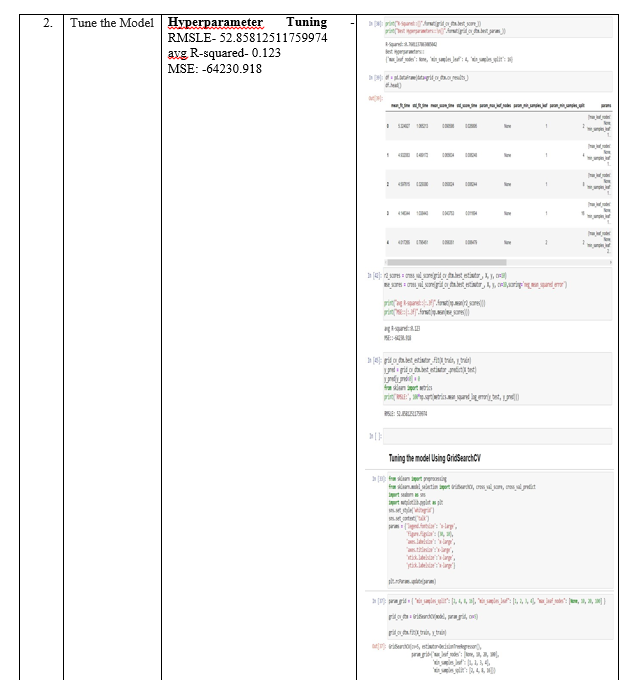






1. **TESTING:**





1. **CODING & SOLUTIONING (Explain the features added in the project along with code)**

a. Feature 1

**Home.html:**

<!DOCTYP

E html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home</title>

<link type="text/css" rel="stylesheet" href="/Flask/static/style.css">

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link

href="https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;600;800&display=s wap" rel="stylesheet">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0beta2/css/all.min.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0beta2/css/v4-shims.min.css">

<style>

\*{

margin: 0; padding: 0;

font-family: 'Poppins', sans-serif;

}

.bar

{

margin: 0px; padding: 15px; background-color:rgb(64, 100, 246); font-family:'Poppins',sans-serif; font-size:25px;

} a{ color:#fff; float:right; text-decoration:none; padding-right:20px;

}

a:hover{

padding: 3.5px; background: #FAAE42;

}

.text-box{

width: 90%;

color:rgba(51, 210, 249, 0.905); text-shadow: #0c0d0e; position:absolute; top: 45%; left: 50%;

transform: translate(-50%,-50%); text-align: center;

}

.text-box h1{

font-size: 70px; text-shadow: 2px 2px 40px #ffffff;

}

.text-box p{

margin: 10px 0 40px; font-size: 25px; color: rgba(0, 0, 0, 0.946);

}

</style>

</head>

<body>

<section class="header">

<div class="bar">

<a href="/pred">Predict</a>

<a href="/home">Home</a>

<br>

</div>

<div class="text-box">

<h1>

DemandEst - AI powered Food Demand Forecaster</h1>

<p> The concept of a balance point between supply and demand is used to explain various situations in our daily lives, from bread in the neighborhood bakery, which can be sold at the equilibrium price, which equals the quantities desired by buyers and sellers, to the negotiation of securities of companies in the stock market.

|  |  |  |
| --- | --- | --- |
|  | On the supply side, a definition of the correct price to be practiced and mainly the quantity are common issues in the planning and execution of the strategy of several companies.</p> | |
|  |  | |
|  | </div> | |
|  | </section> | |
|  | </body> | |
| <html lang="en"  > | </html>      **Upload.html:-** | |
|  |  |
|  |  | <head> |
|  |  | <meta charset="UTF-8"> |
|  |  | <meta http-equiv="X-UA-Compatible" content="IE=edge"> |
|  |  | <meta name="viewport" content="width=device-width, initial-scale=1.0"> |
|  |  | <title>Predict</title> |
|  |  | <link rel="preconnect" href="https://fonts.googleapis.com"> |
|  |  | <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin> |
|  |  | <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;600;800&display=swa p" rel="stylesheet"> |
|  |  | <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0beta2/css/all.min.css"> |
|  |  |  |
|  |  | <style> |

.bar { margin: 0px; padding: 15px; background-color:rgb(100, 5, 29); /\* opacity:0.6; \*/ font-family:'Poppins',sans-serif; font-size:25px;

} a { color:#fff; float:right; text-decoration:none; padding-right:20px;

}

a:hover{

padding: 3.5px; background: #FAAE42;

}

h1{

color:rgb(100, 5, 29); font-family:Poppins; font-size:30

}

h2{

color:rgb(100, 5, 29); font-family: Poppins; font-size:60; margin-bottom: 10px;

}

.my-cta-button{

font-size: 20px; color: rgb(15, 15, 15); border: 1px solid #0e0e0ccf; padding: 3.5px;

cursor: pointer;

}

.my-cta-button:hover{ border: 2px solid #faae42; padding: 3.5px; background: #FAAE42;

} p { color:white; font-family: Poppins; font-size:30px;

}

</style>

</head>

<body>

<div class="bar">

<a href="/pred">Predict</a>

<a href="/home">Home</a>

<br>

</div>

<div class="container">

<center> <div id="content" style="margin-top:2em">

<h2><center>Food Demand Forecasting</center></h2>

<form action="{{ url\_for('predict') }}" method="POST">

<select id="homepage\_featured" name="homepage\_featured">

<option value="">homepage\_featured</option>

<option value="0">No</option>

<option value="1">Yes</option>

</select><br><br>

<select id="emailer\_for\_promotion" name="emailer\_for\_promotion">

<option value="">emailer\_for\_promotion</option>

<option value="0">No</option>

<option value="1">Yes</option>

</select><br><br>

<input class="form-input" type="text" name="op\_area" placeholder="Enter the op\_area(2-

7)"><br><br>

<select id="cuisine" name="cuisine">

<option value="">Cuisine</option>

<option value="0">Continental</option>

<option value="1">Indian</option>

<option value="2">Italian</option>

<option value="3">Thai</option>

</select><br><br>

<input class="form-input" type="text" name="city\_code" placeholder="Enter city\_code"><br><br>

<input class="form-input" type="text" name="region\_code" placeholder="Enter region\_code"><br><br>

<select id="category" name="category">

<option value="">Category</option>

<option value="0">Beverages</option>

<option value="1">Biryani</option>

<option value="2">Desert</option>

<option value="3">Extras</option>

<option value="4">Fish</option>

<option value="5">Other Snacks</option>

<option value="6">Pasta</option>

<option value="7">Pizza</option>

<option value="8">Rice Bowl</option>

<option value="9">Salad</option>

<option value="10">Sandwich</option>

<option value="11">Seafood</option>

<option value="12">Soup</option>

<option value="13">Starters</option>

</select><br><br>

<input type="submit" class="my-cta-button" value="Predict">

</form>

<br>

<h1 class="predict">Number of orders: {{ prediction\_text }}</h1>

</div></center>

</div>

</body>

</body>

**App.py:-**

import pandas as pd import numpy as np

import pickle import os from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_, template\_folder="templates")

@app.route('/', methods=['GET']) def index():

return render\_template('home.html')

@app.route('/home', methods=['GET']) def about():

return render\_template('home.html')

@app.route('/pred', methods=['GET']) def page():

return render\_template('upload.html')

@app.route('/predict', methods=['GET', 'POST']) def predict():

print("[INFO] loading model...") model = pickle.load(open('foodDemand.pkl', 'rb')) input\_features = [float(x) for x in request.form.values()] features\_value = [np.array(input\_features)] print(features\_value)

features\_name = ['homepage\_featured', 'emailer\_for\_promotion', 'op\_area', 'cuisine',

'city\_code', 'region\_code', 'category']

prediction = model.predict(features\_value) output = prediction[0] print(output) return render\_template('upload.html', prediction\_text=output)

if \_\_name\_\_ == '\_\_main\_\_': app.run(debug=False)

**Ibmapp.py:**

import pandas as pd

import numpy as np import pickle import os import requests

# NOTE: you must manually set API\_KEY below using information retrieved from your IBM Cloud account.

API\_KEY = "68w9XBNJLBQFtHM2rG\_aouV4LmlF-EtecYrhIQBQbt\_K"

token\_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey": API\_KEY, "grant\_type": 'urn:ibm:params:oauth:grant-type:apikey'}) mltoken = token\_response.json()["access\_token"]

header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_, template\_folder="templates")

@app.route('/', methods=['GET']) def index():

return render\_template('home.html')

@app.route('/home', methods=['GET']) def about():

return render\_template('home.html')

@app.route('/pred', methods=['GET']) def page():

return render\_template('upload.html')

@app.route('/predict', methods=['GET', 'POST']) def predict():

print("[INFO] loading model...")

# model = pickle.load(open('fdemand.pkl', 'rb')) input\_features = [int(x) for x in request.form.values()] print(input\_features) features\_value = [[np.array(input\_features)]] print(features\_value)

payload\_scoring = {"input\_data": [{"field": [['homepage\_featured', 'emailer\_for\_promotion', 'op\_area',

'cuisine',

'city\_code', 'region\_code', 'category']],

"values": [input\_features]}]}

response\_scoring = requests.post(

'https://us-south.ml.cloud.ibm.com/ml/v4/deployments/80afcaad-591d-4869-bf54-

17bbb8c70ea3/predictions?version=2022-11-14', json=payload\_scoring, headers={'Authorization': 'Bearer ' + mltoken}) print("Scoring response") print(response\_scoring.json()) predictions = response\_scoring.json()

print(predictions) print('Final Prediction Result', predictions['predictions'][0]['values'][0][0])

pred = predictions['predictions'][0]['values'][0][0]

# prediction = model.predict(features\_value)

# output=prediction[0] # print(output) print(pred)

return render\_template('upload.html', prediction\_text=pred)

if \_\_name\_\_ == '\_\_main\_\_': app.run(debug=False)

b. Feature 2

**main.py:-**

import numpy

as np

import pandas as pd import plotly.express as px import matplotlib.pyplot as plt import seaborn as sns

from sklearn.preprocessing import OneHotEncoder, StandardScaler from sklearn.model\_selection import train\_test\_split from sklearn import metrics from sklearn.pipeline import make\_pipeline from sklearn.ensemble import RandomForestRegressor

import warnings

warnings.filterwarnings('ignore')

# Importing Raw Files train\_raw = pd.read\_csv('train.csv') test\_raw = pd.read\_csv('test.csv') meal = pd.read\_csv('meal\_info.csv') centerinfo = pd.read\_csv('fulfilment\_center\_info.csv')

# Analyzing Data

print("The Shape of Demand dataset :", train\_raw.shape) print("The Shape of Fulfillment Center Information dataset :", centerinfo.shape) print("The Shape of Meal information dataset :", meal.shape) print("The Shape of Test dataset :", test\_raw.shape) train\_raw.head() centerinfo.head() meal.head() test\_raw.head()

# Check for missing values train\_raw.isnull().sum().sum() test\_raw.isnull().sum().sum()

# Analysis report

print("The company has", centerinfo["center\_id"].nunique(), " warehouse ", "spreed into ", centerinfo["city\_code"].nunique(), "City and ", centerinfo["region\_code"].nunique(), "Regions") print("The products of the company are ", meal["meal\_id"].nunique(), "unique meals , divided into ", meal["category"].nunique(), "category and ", meal["cuisine"].nunique(), "cuisine")

# Merge meal,center-info data with train and test data train = pd.merge(train\_raw, meal, on="meal\_id", how="left") train = pd.merge(train, centerinfo, on="center\_id", how="left") print("Shape of train data : ", train.shape)

train.head()

# Merge test data with meal and center info test = pd.merge(test\_raw, meal, on="meal\_id", how="outer") test = pd.merge(test, centerinfo, on="center\_id", how="outer") print("Shape of test data : ", test.shape)

test.head()

# Typecasting to assign appropriate data type to variables col\_names = ['center\_id', 'meal\_id', 'category', 'cuisine', 'city\_code', 'region\_code', 'center\_type'] train[col\_names] = train[col\_names].astype('category') test[col\_names] = test[col\_names].astype('category') print("Train Datatype\n", train.dtypes) print("Test Datatype\n", test.dtypes)

# Orders by centers

center\_orders = train.groupby("center\_id", as\_index=False).sum() center\_orders = center\_orders[["center\_id", "num\_orders"]].sort\_values(by="num\_orders", ascending=False).head(10)

fig = px.bar(x=center\_orders["center\_id"].astype("str"), y=center\_orders["num\_orders"], title="Top 10

Centers by Order",

labels={"x": "center\_id", "y": "num\_orders"})

fig.show()

# Pie chart on food category fig = px.pie(values=train["category"].value\_counts(), names=train["category"].unique(), title="Most popular food category")

fig.show()

# Orders by Cuisine types cuisine\_orders = train.groupby(["cuisine"], as\_index=False).sum() cuisine\_orders = cuisine\_orders[["cuisine", "num\_orders"]].sort\_values(by="num\_orders", ascending=False)

fig = px.bar(cuisine\_orders, x="cuisine", y="num\_orders", title="orders by cuisine") fig.show()

# Impact of check-out price on order train\_sample = train.sample(frac=0.2) fig = px.scatter(train\_sample, x="checkout\_price", y="num\_orders", title="number of order change with checkout price")

fig.show()

sns.boxplot(train["checkout\_price"]) # Orders weekly trend

week\_orders = train.groupby(["week"], as\_index=False).sum() week\_orders = week\_orders[["week", "num\_orders"]] fig = px.line(week\_orders, x="week", y="num\_orders", markers=True, title="Order weekly trend") fig.show()

# Deriving discount percent and discount y/n train['discount percent'] = ((train['base\_price'] - train['checkout\_price']) / train['base\_price']) \* 100

# Discount Y/N

train['discount y/n'] = [1 if x > 0 else 0 for x in (train['base\_price'] - train['checkout\_price'])]

# Creating same feature in test dataset test['discount percent'] = ((test['base\_price'] - test['checkout\_price']) / test['base\_price']) \* 100 test['discount y/n'] = [1 if x > 0 else 0 for x in (test['base\_price'] - test['checkout\_price'])]

train.head(2)

# Check for correlation between numeric features plt.figure(figsize=(13, 13)) sns.heatmap(train.corr(), linewidths=.1, cmap='Reds', annot=True) plt.title('Correlation Matrix') plt.show()

# Define One hot encoding function def one\_hot\_encode(features\_to\_encode, dataset):

encoder = OneHotEncoder(sparse=False) encoder.fit(dataset[features\_to\_encode]) encoded\_cols = pd.DataFrame(encoder.transform(dataset[features\_to\_encode]), columns=encoder.get\_feature\_names())

dataset = dataset.drop(columns=features\_to\_encode) for cols in encoded\_cols.columns: dataset[cols] = encoded\_cols[cols] return dataset

# get list of categorical variables in data set ls = train.select\_dtypes(include='category').columns.values.tolist()

# Run one-hot encoding on all categorical variables features\_to\_encode = ls data = one\_hot\_encode(features\_to\_encode, train) data = data.reset\_index(drop=True) # Train-Validation Data Split y = data[["num\_orders"]]

X = data.drop(["num\_orders", "id", "base\_price", "discount y/n"], axis=1)

X = X.replace((np.inf, -np.inf, np.nan), 0) # replace nan and infinity values with 0

# 20% of train data is used for validation

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.20, random\_state=100)

# Prepare test data post applying onehot encoding OH\_test = one\_hot\_encode(features\_to\_encode, test) test\_final = OH\_test.drop(["id", "base\_price", "discount y/n"], axis=1)

# Create pipeline for scaling and modeling

RF\_pipe = make\_pipeline(StandardScaler(), RandomForestRegressor(n\_estimators=100, max\_depth=7))

# Build Model

RF\_pipe.fit(X\_train, y\_train)

# Predict Value

RF\_train\_y\_pred = RF\_pipe.predict(X\_val)

# Model Evaluation- print('R Square:', RF\_pipe.score(X\_val, y\_val))

print('RMSLE:', 100 \* np.sqrt(metrics.mean\_squared\_log\_error(y\_val, RF\_train\_y\_pred)))

# Applying algorithm to predict orders

test\_y\_pred = RF\_pipe.predict(test\_final)

Result = pd.DataFrame(test\_y\_pred)

print(Result.values)

Result = pd.DataFrame(test\_y\_pred)

Submission = pd.DataFrame(columns=['id', 'num\_orders'])

Submission['id'] = test['id']

Submission['num\_orders'] = Result.values

Submission.to\_csv('My submission.csv', index=False) print(Submission.shape)

print(Submission.head())

**RESULTS**

c. Performance Metrics – he evaluation metric for this competition is 100\*RMSLE where RMSLE is Root of Mean Squared Logarithmic Error across all entries in the test set where our accuracy 92% , rsme – 0.8934\

1. **ADVANTAGES & DISADVANTAGES**

**ADVANTAGE:**

* + In supply chain networks, demand forecasting with the aid of AI-based techniques can cut errors by 30 to 50 percent. By implementing these approaches, organisations may be able to forecast accurately at all levels.

**DIS-ADVANTAGE:**

* + Not every situation can be predicted

1. **CONCLUSION**

Therefore, this complete representation shows the progress on the topic in an systematically view .This implementation along with several code has separate topics to evolve around for the best outome as a report.

1. **FUTURE SCOPE**

Predictions , availability, Scalability , Demand , everything will be followed on a correct procedure .

1. **APPENDIX** :

https://github.com/IBM-EPBL/IBM-Project-47554-1660800169