Report For Phase 3

Hotel Booking Cancellation Prediction Web Application

Objective:

Our project is a booking cancellation predictor which predicts whether a booking will be cancelled or not based on the booking details.

This will help hotel management to hold the rooms according to the chances of them being fulfilled.

This predictor would be especially useful when the number of refundable bookings is high.

Allowing the management to get insights into the occupancy of the hotel.

Why did we choose this model?

We have developed an application which supports all the 6 machine learning models we used in phase 2.

Logistic Regression

KneighborsClassifier

SVM (Support Vector Machine)

Gaussian Naïve Bayes

Random Forest Classifier

Decision Tree Classifier

You can choose any model while predicting. As we got the accuracy same for all the models we had to go with this option. We have tuned our model on selecting only required columns which our model trained.

Data used in the prediction model:

Target Variable: Our model will predict whether the booking is cancelled or not.

Canceled or Not: A String indicating if the booking was canceled or not.

Booking Information:

Hotel: Resort or city type

Lead time: Number of days between booking entry and arrival.

Stays in weekend, stays in weeks: Number of nights the guest stays on weekends and weekdays.

Guest Information:

Family, adults, children: Families and Number of adults and children.

Meal: Type of meal booked (BB: Bed & Breakfast, HB: Half board, FB: Full board, SC: No meal package).

Market segment: Market segment designation (e.g., Travel Agents, Tour Operators).

Distribution channel: Booking distribution channel.

Booking History:

Is repeated guest: Binary value indicating if the booking is from a repeated guest.

Previous cancellations, bookings canceled: Number of previous cancellations and non-canceled bookings.

Room Information:

Reserved room type: Codes for reserved.

Deposit and Payment:

Deposit type: Type of deposit made by the customer (No Deposit, Non-Refund, Refundable).

Customer Type:

Customer type: Type of booking (Contract, Group, Transient, Transient party).

ADR: Average Daily Rate (sum of lodging transactions divided by the total number of staying nights).

No parking spaces: Number of car parking spaces required.

No Special Request: Number of special requests made by the customer.

Total Nights: Total nights stayed by the customers.

We also have a selection field to select the model you need for your predictions.

Components and Features:

Libraries:

The application leverages Flask, a micro web framework for Python, to handle HTTP requests and responses.

NumPy is used for numerical operations on input data.

Joblib is utilized for loading pre-trained machine learning models.

Various scikit-learn classifiers, including KNeighborsClassifier, LogisticRegression, GaussianNB, RandomForestClassifier, SVC, and DecisionTreeClassifier, are employed for predictions.

Initialization and Model Loading:

The Flask application is initialized, and pre-trained models are loaded outside the route function to avoid unnecessary reloads on each request.

Models are stored in the 'models' directory and loaded using joblib.

Prediction Route (/predict):

The main route '/predict' is defined to handle both POST and OPTIONS methods.

OPTIONS method is utilized for CORS pre-flight checks, ensuring compatibility with cross-origin requests.

Handling OPTIONS Request:

If the incoming request is an OPTIONS request, the server responds with default options to indicate allowed methods and headers.

Handling Prediction Request:

For POST requests, the application extracts JSON data containing input data and the selected machine learning model.

Input data is processed and converted into a format suitable for model prediction.

Performing Predictions:

Based on the selected model, predictions are performed using the corresponding pre-trained machine learning model.

The application supports various models, including k-Nearest Neighbors, Logistic Regression, Naive Bayes, Random Forest, Support Vector Classifier, and Decision Tree Classifier.

Returning Predictions as JSON:

Predictions are converted to JSON format and included in the response.

The response also contains CORS headers to allow cross-origin requests.

Error Handling:

In case of any exceptions during the prediction process, an error message is included in the response to inform the client.

CORS Headers:

CORS headers are set to allow requests from any origin ('*').

Specific headers ('Content-Type') and methods ('POST', 'OPTIONS') are declared to enhance security.

Debugging and Development:

The application includes a debug mode to assist in development, providing detailed error messages and automatic reloading of the server upon code changes.

Execution:

The Flask application is executed when the script is run.

How to Run the Application:

Install Python, Flask, npm manager, Node Js

To run the application, you need to up backend (Flask) and frontend (React) services.

For the backend you need to install Python.

After installing python, you need to run the below command to install Flask and scikit-learn to run the Flask app.

pip install Flask scikit-learn

After running the above command, you need to navigate to the project folder where you will find the folder called Flask. Enter Flask folder and open the command prompt in that folder and run.

python app.py

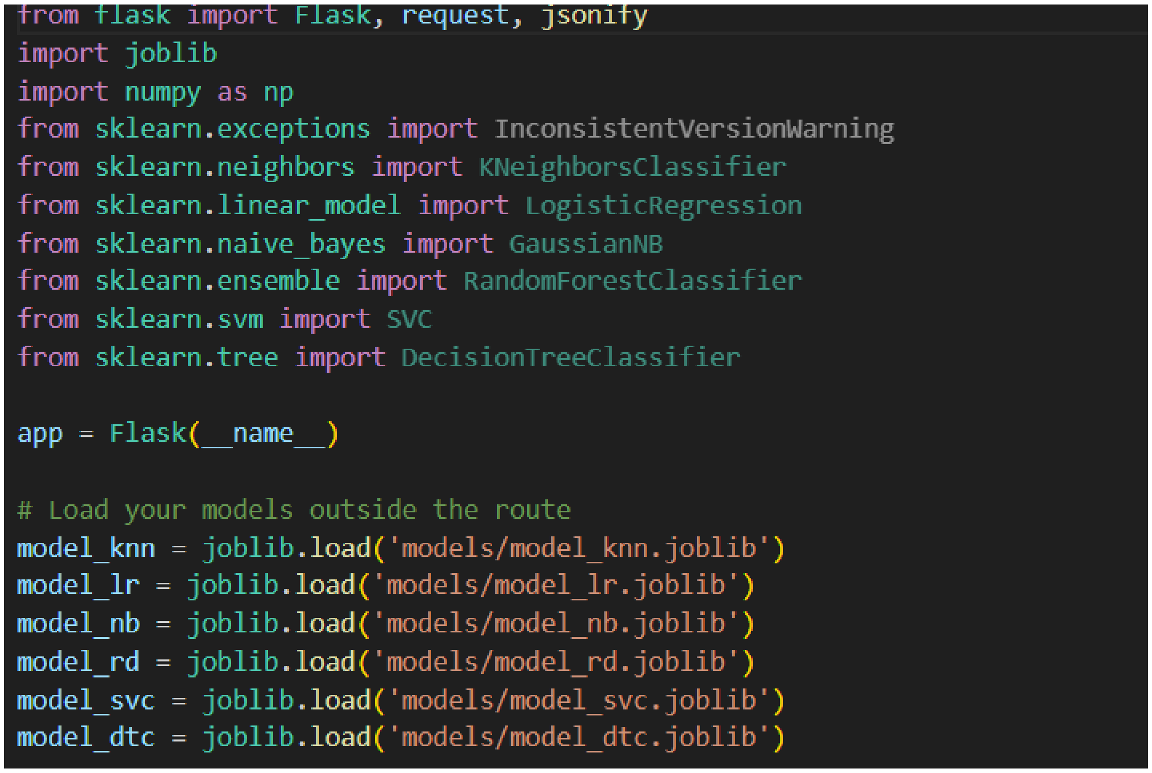
which starts your backend server.

We have provided two folders in the **src** directory.

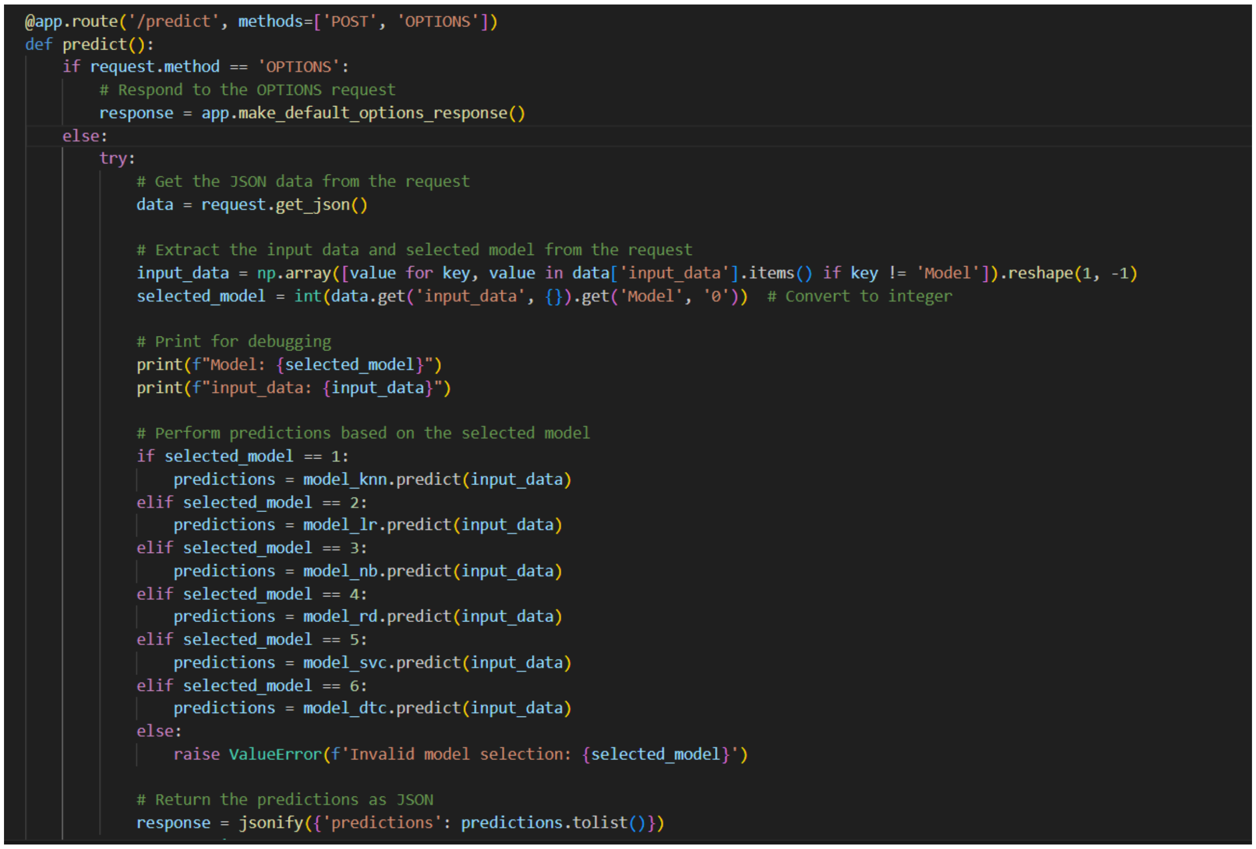
Frontend: in this folder you will find react app folder enter it and then open command prompt in the folder and run **npm start** command which will start the frontend server and shows the UI.

Backend: go into this folder and open the folder the folder in command prompt and run this command **python app.py** this will run the backend flask server.

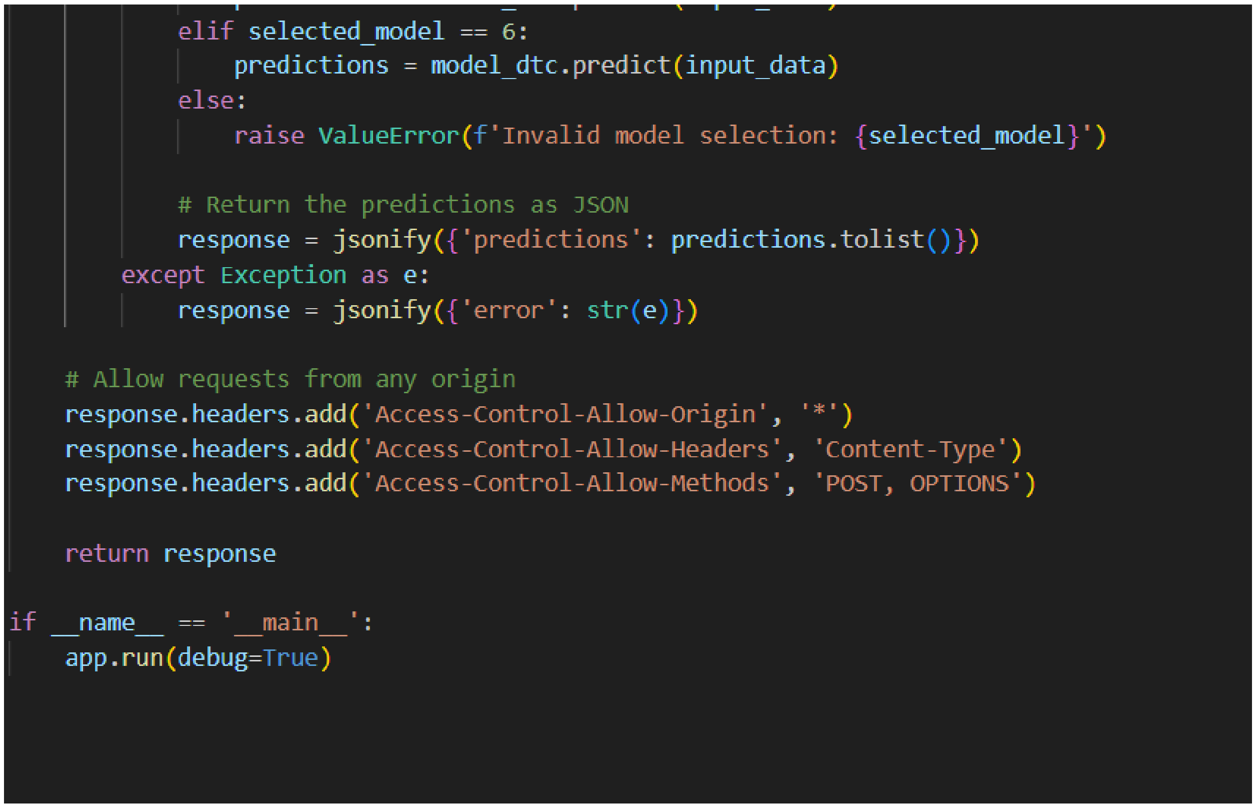
Backend Code



```
from flask import Flask, request, jsonify
import joblib
import numpy as np
from sklearn.exceptions import InconsistentVersionWarning
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

app = Flask(__name__)

# Load your models outside the route
model_knn = joblib.load('models/model_knn.joblib')
model_lr = joblib.load('models/model_lr.joblib')
model_nb = joblib.load('models/model_nb.joblib')
model_rd = joblib.load('models/model_rd.joblib')
model_svc = joblib.load('models/model_svc.joblib')
model_dtc = joblib.load('models/model_dtc.joblib')
```



```python
@app.route('/predict', methods=['POST', 'OPTIONS'])
def predict():
    if request.method == 'OPTIONS':
        # Respond to the OPTIONS request
        response = app.make_default_options_response()
    else:
        try:
            # Get the JSON data from the request
            data = request.get_json()

            # Extract the input data and selected model from the request
            input_data = np.array([value for key, value in data['input_data'].items() if key != 'Model']).reshape(1, -1)
            selected_model = int(data.get('input_data', {}).get('Model', '0'))  # Convert to integer

            # Print for debugging
            print(f"Model: {selected_model}")
            print(f"input_data: {input_data}")

            # Perform predictions based on the selected model
            if selected_model == 1:
                predictions = model_knn.predict(input_data)
            elif selected_model == 2:
                predictions = model_lr.predict(input_data)
            elif selected_model == 3:
                predictions = model_nb.predict(input_data)
            elif selected_model == 4:
                predictions = model_rd.predict(input_data)
            elif selected_model == 5:
                predictions = model_svc.predict(input_data)
            elif selected_model == 6:
                predictions = model_dtc.predict(input_data)
            else:
                raise ValueError(f'Invalid model selection: {selected_model}')

            # Return the predictions as JSON
            response = jsonify({'predictions': predictions.tolist()})
```



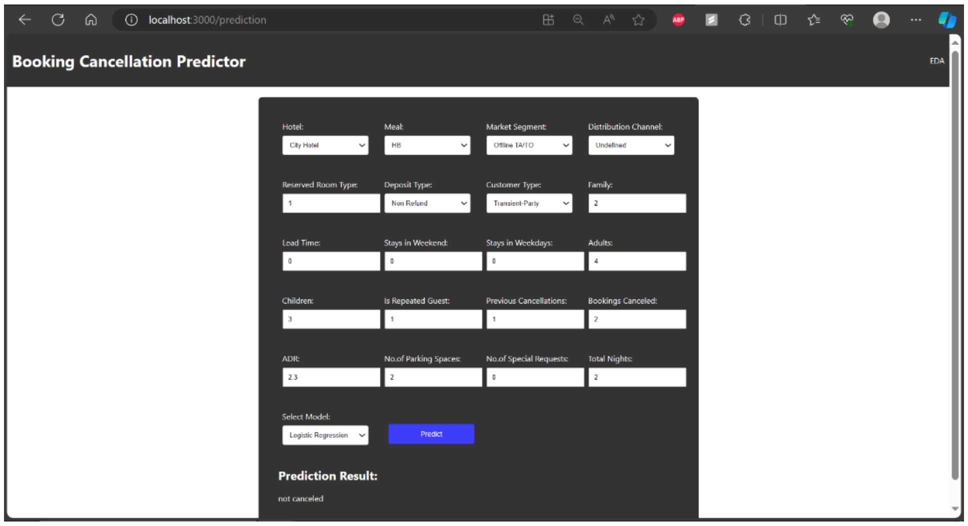
```python
            elif selected_model == 6:
                predictions = model_dtc.predict(input_data)
            else:
                raise ValueError(f'Invalid model selection: {selected_model}')

            # Return the predictions as JSON
            response = jsonify({'predictions': predictions.tolist()})
```

```python
        except Exception as e:
            response = jsonify({'error': str(e)})

    # Allow requests from any origin
    response.headers.add('Access-Control-Allow-Origin', '*')
    response.headers.add('Access-Control-Allow-Headers', 'Content-Type')
    response.headers.add('Access-Control-Allow-Methods', 'POST, OPTIONS')

    return response

if __name__ == '__main__':
    app.run(debug=True)
```

For Frontend, we have used React and CSS for styling.

We created a form which consisted of 20 columns (we choose based on the correlation matrix obtained)

And a Predict button which shows the result and the tab to show the visualization.



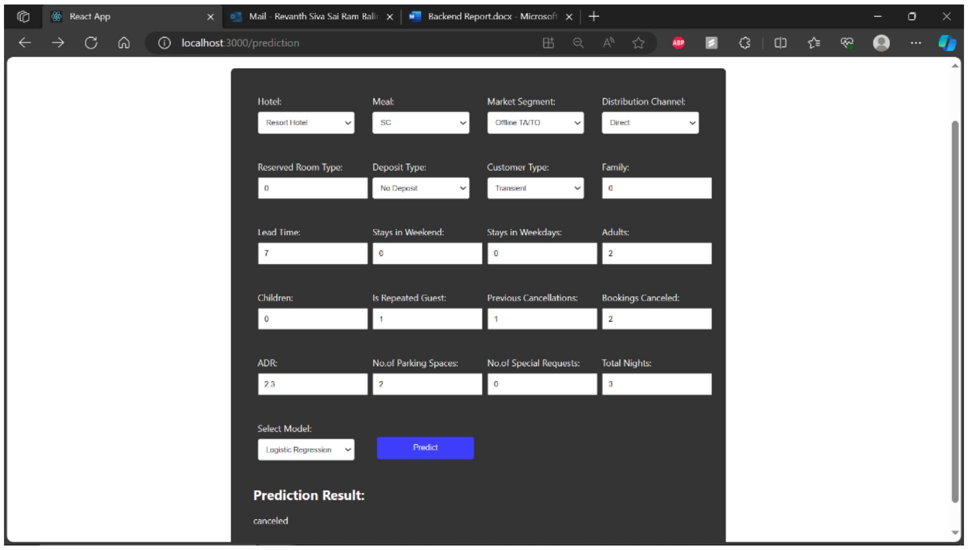
Booking Cancellation Predictor

Hotel: City Hotel	Meal: HB	Market Segment: Offline TA/TO	Distribution Channel: Undefined
Reserved Room Type: 1	Deposit Type: Non Refund	Customer Type: Transient-Party	Family: 2
Lead Time: 0	Stays in Weekend: 0	Stays in Weekdays: 0	Adults: 4
Children: 3	Is Repeated Guest: 1	Previous Cancellations: 1	Bookings Canceled: 2
ADR: 2.3	No.of Parking Spaces: 2	No.of Special Requests: 0	Total Nights: 2

Select Model: Logistic Regression

Prediction Result:

not canceled

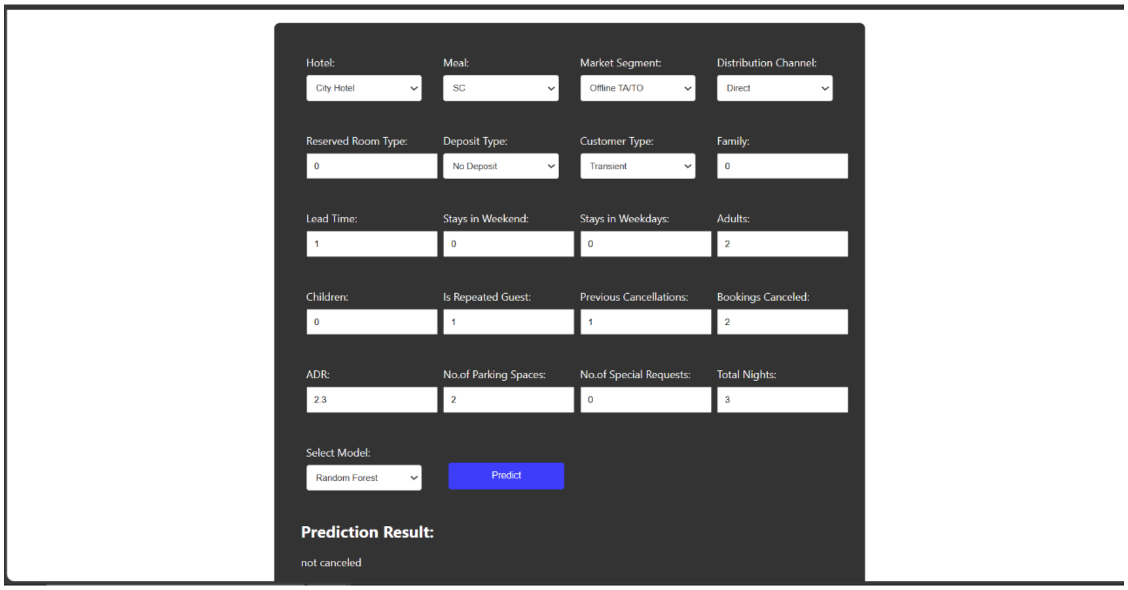

React App
Mail - Revanth Siva Sai Ram Balin
Backend Report.docx - Microsoft
localhost:3000/prediction
Predict

Hotel: Resort Hotel	Meal: SC	Market Segment: Offline TA/TO	Distribution Channel: Direct
Reserved Room Type: 0	Deposit Type: No Deposit	Customer Type: Transient	Family: 0
Lead Time: 7	Stays in Weekend: 0	Stays in Weekdays: 0	Adults: 2
Children: 0	Is Repeated Guest: 1	Previous Cancellations: 1	Bookings Canceled: 2
ADR: 2.3	No.of Parking Spaces: 2	No.of Special Requests: 0	Total Nights: 3

Select Model: Logistic Regression

Prediction Result:

canceled



Hotel: City Hotel	Meal: SC	Market Segment: Offline TA/TO	Distribution Channel: Direct
Reserved Room Type: 0	Deposit Type: No Deposit	Customer Type: Transient	Family: 0
Lead Time: 1	Stays in Weekend: 0	Stays in Weekdays: 0	Adults: 2
Children: 0	Is Repeated Guest: 1	Previous Cancellations: 1	Bookings Canceled: 2
ADR: 2.3	No.of Parking Spaces: 2	No.of Special Requests: 0	Total Nights: 3

Select Model: Random Forest

Prediction Result:

not canceled

FrontEnd Code:

This code defines a React component named PredictionForm, which represents a form for making predictions using a machine learning model. Here's a brief overview of the code:

State Management:

The component uses the useState hook to manage state variables, including inputData (to store form input values) and prediction (to store the result of the prediction).

Form Input Handling:

The form includes various input fields for different features related to hotel bookings.

handleInputChange function updates the inputData state when users enter values in the form.

Model Selection:

Users can select a machine learning model from a dropdown list (selectedModel state).

The selected model is sent along with the input data to the server for prediction.

HTTP Request to Server:

The form has a submit event handler (handleSubmit) that sends a POST request to the server endpoint (http://127.0.0.1:5000/predict) this is a flask server url.

The request includes numeric input data and the selected machine learning model.

Options for Categorical Features:

For categorical features like hotel type, meal, market segment, etc., the code defines options in JavaScript objects (e.g., optionsHotel, optionsMeal).

Prediction Result Display:

After making a prediction, the result is displayed below the form (<div className="prediction-result">).

Axios for HTTP Requests:

The axios library is used for making asynchronous HTTP requests to the server.

Component Structure:

The component is structured in a way that ensures a clear separation of concerns, making it easy to understand and maintain.

Model Selection Dropdown:

Users can select one of the pre-trained machine learning models (Logistic Regression, KNN Neighbors, SVM Classifier, Naive Bayes, Random Forest, Decision Tree) for making predictions.

Result Display:

If a prediction is available (prediction !== null), it is displayed below the form.

Numeric Conversion:

The input data values are converted to numeric format before sending them to the server for prediction.

How to run the application:

To run the application, we need to up the backend and frontend servers.

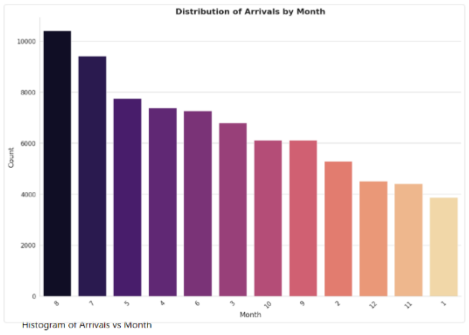
To up the backend server, head to the flask folder and open the command prompt then type **python app.py** which starts the backend server.

To up the frontend server, head to the frontend folder and open the command prompt in the current folder then run **npm start** command, which starts the frontend server and UI shows up like this.

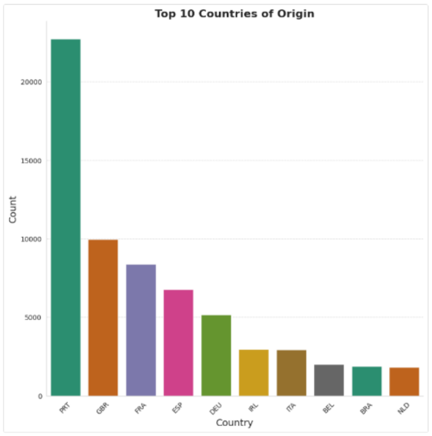
EDA Page:

The EDA Page can be navigated to by clicking the EDA hyperlink on the top bar

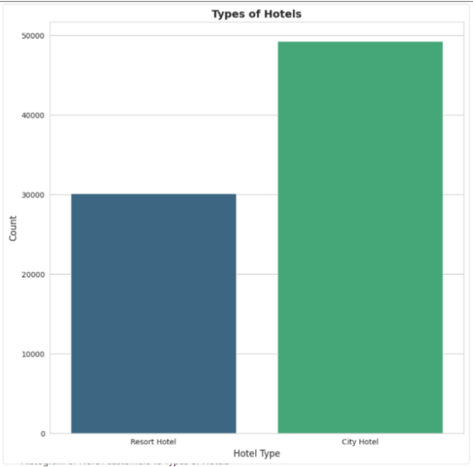
Exploratory Data Analysis (EDA) Page



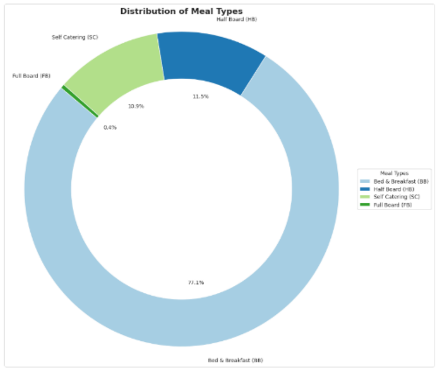
Histogram of Arrivals vs Month



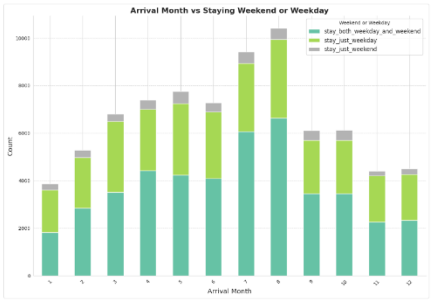
Histogram of No.Of customers vs Origin Country



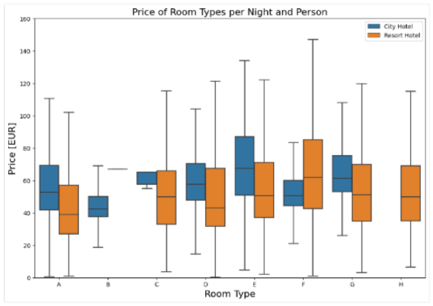
Histogram of Hotel customers vs types of Hotels



Pie Chart of Meal Types

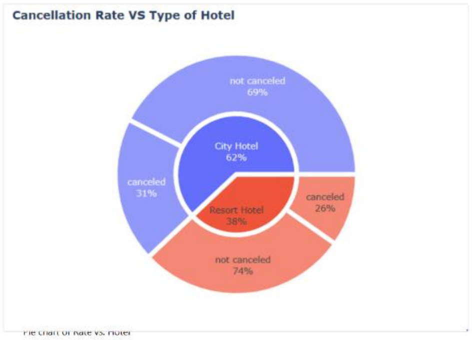


Arrival Month vs. Stay Type

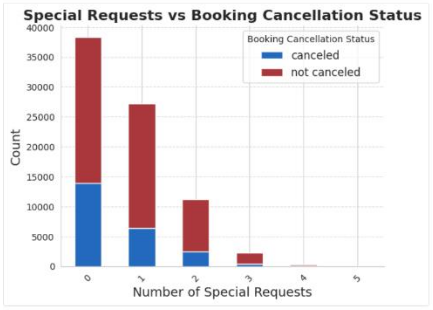


Price vs. Room Type

Price for Room Type



Pie chart of Rate vs. Hotel



Histogram of Special Requests vs. Cancellations