Waiter tips prediction system

A Project Report

Submitted in partial fulfillment of the
Requirements for the award of the Degree of

BACHELOR OF SCIENCE
(COMPUTER SCIENCE)

By

Name: Mohammed Talha Khan
Roll Number: 424

Under the esteemed guidance of

Prof. Madhav Mishra

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE

GURU NANAK KHALSA COLLEGE
OF
ARTS, SCIENCE & COMMERCE

(***Autonomous)***

MATUNGA, MUMBAI - 400 019
AY 2023-2024

GURU NANAK KHALSA COLLEGE
OF
ARTS, SCIENCE & COMMERCE

(*Autonomous)*

MATUNGA, MUMBAI, MAHARASHTRA – 400 019

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the entitled, **"Waiter tips prediction system",** is bonafide work of **Mohammed Talha Khan** bearing Seat No. **424** submitted in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY** from University of Mumbai.

Internal Guide **Coordinator**

External Examiner

Date: **College Seal**

ABSTRACT

In the hospitality industry, the income of waitstaff largely depends on the tips they receive from customers. Predicting the amount of tips a waiter is likely to receive during a given shift is a complex task influenced by various factors such as customer behavior, service quality, and environmental conditions. Developing a Waiter Tips Prediction System aims to address this challenge by leveraging data-driven approaches to provide accurate estimations of tips, ultimately helping waiters and restaurant management optimize their service strategies.

This system aims to provide accurate estimations of tipping amounts by considering a myriad of factors, including customer behavior, service quality, meal cost, and external variables like holidays or special occasions.

The expected outcome is a sophisticated model that not only accurately predicts tips but also offers actionable insights for waitstaff and restaurant management. These insights can be leveraged to optimize service strategies, enhance customer satisfaction, and ultimately improve the financial outcomes for both waiters and restaurant owners.

The dynamic nature of the restaurant industry introduces further complexity. Tipping patterns can vary during peak hours, different seasons, and in response to specific events. The model needs to be agile and capable of adapting in real-time to these fluctuations, providing waitstaff and management with timely and relevant predictions.

Privacy and ethical considerations add another layer to the problem. Balancing the need for accurate predictions with the responsibility of handling sensitive customer data is crucial.

APPROVAL OF PROJECT PROPOSAL

PRN No: _____ Roll No: 424

- **Name of the Student**
  - Mohammed Talha Khan
- **Title of the Project**
  - "Waiter tips prediction system"
- **Name of the Project Guide**
  - Prof. Madhav Mishra

Signature of the Student **Signature of the Guide**

Date: **Date:**

Signature of the Head of Dept. **Signature of the Examiner**

Date: **Date:**

ACKNOWLEDGEMENT

I would like to express my thanks to the people who have helped me most throughout my project. I am grateful to my **Prof. Madhav Mishra** for nonstop support for the project. I cannot say thank you enough for him tremendous support and help.

I owe my deep gratitude to our HOD of Computer Science Department
Mrs. Jasmeet Kaur Ghai who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

At last, but not the least I want to thank all of my friends who helped/treasured me out in completing the project, where they all exchanged their own interesting ideas, thoughts and made this possible to complete my project with all accurate information. I wish to thank my parents for their personal support or attention who inspired/encouraged me to go my own way

DECLARATION

I hereby declare that the project entitled, "**Waiter Tips Prediction System.**" done at **Guru Nanak Khalsa College**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE (Computer Science)** to be submitted as final semester project as part of our curriculum.

Mohammed Talha Khan

Table of Contents

Project

1.TITLE OF THE PROJECT: *Waiter Tips prediction System.*

PROBLEM STATEMENT:

In the hospitality industry, the income of waitstaff largely depends on the tips they receive from customers. Predicting the amount of tips a waiter is likely to receive during a given shift is a complex task influenced by various factors such as customer behavior, service quality, and environmental conditions. Developing a Waiter Tips Prediction System aims to address this challenge by leveraging data-driven approaches to provide accurate estimations of tips, ultimately helping waiters and restaurant management optimize their service strategies.

This system aims to provide accurate estimations of tipping amounts by considering a myriad of factors, including customer behavior, service quality, meal cost, and external variables like holidays or special occasions.

The expected outcome is a sophisticated model that not only accurately predicts tips but also offers actionable insights for waitstaff and restaurant management. These insights can be leveraged to optimize service strategies, enhance customer satisfaction, and ultimately improve the financial outcomes for both waiters and restaurant owners.

The dynamic nature of the restaurant industry introduces further complexity. Tipping patterns can vary during peak hours, different seasons, and in response to specific events. The model needs to be agile and capable of adapting in real-time to these fluctuations, providing waitstaff and management with timely and relevant predictions.

Privacy and ethical considerations add another layer to the problem. Balancing the need for accurate predictions with the responsibility of handling sensitive customer data is crucial.

Expected Outcomes:

Waiter Tips Prediction model would be a system that accurately predicts the amount of tips a waiter is likely to receive during a given shift based on various influencing factors. The model should provide actionable insights for waitstaff and restaurant management to optimize service strategies, improve customer satisfaction, and enhance overall financial outcomes.

Abstract:

In the hospitality industry, the income of waitstaff largely depends on the tips they receive from customers. Predicting the amount of tips a waiter is likely to receive during a given shift is a complex task influenced by various factors such as customer behavior, service quality, and environmental conditions.

Overall, the Waiter Tips Prediction System aims to revolutionize the hospitality industry by providing actionable insights, enhancing service quality, and optimizing financial outcomes for both waiters and restaurant owners.

Key Challenges:

Multifactorial Nature: The tipping amount is affected by a multitude of factors, including customer satisfaction, meal cost, waiter's performance, and external factors like holidays or special occasions. Developing a system that accurately considers these variables is essential.

Dynamic Environment: The restaurant industry is dynamic, with factors like peak hours, seasons, and events impacting tipping patterns. The system needs to adapt to these changes and provide real-time predictions to be practical and effective.

Privacy and Ethical Considerations: Handling customer data requires a robust framework that respects privacy regulations and ensures ethical data usage. The system must be designed to protect sensitive information while still providing valuable predictions.

Model Interpretability: For wider acceptance and trust, the prediction model should be interpretable. Waiters and restaurant management need to understand why a particular prediction was made, allowing them to make informed decisions based on the insights provided.

Integration with Existing Systems: Seamless integration with existing restaurant management systems is crucial for practical implementation. The solution should be compatible with point-of-sale (POS) systems, reservation systems, and other relevant tools.

User-Friendly Interface: The system should offer a user-friendly interface accessible to both waitstaff and management. Visualization tools and easy-to-understand reports can aid in making informed decisions and improving service strategies.

Adaptability to Different Restaurant Settings: Restaurants vary in terms of size, cuisine, and clientele. The system should be adaptable to different settings, ensuring its effectiveness across a diverse range of establishments.

2.Data Description:

The dataset is provided by Kaggle website.
Reference: https://WWW.kaggle.com/datasets/jsphyg/tipping?resource=download

This dataset have 244 rows and 7 columns.

Column 1: [total_bill]
This column shows the amount of bill payed by the customer's.(in Dollars)

total_bill
16.99
10.34
21.01
23.68
24.59

Column2: [tip]
This column shows the amount of tips payed to the waiter by customer's.(in Dollars)

tip
1.01
1.66
3.50
3.31
3.61

Column 3: [sex]
This column provides the gender of the bill payer.

sex
Female
Male
Male
Male
Female

Column 4: [smoker]
This column gives a boolean if whether the bill payer do smoke or he/she does'nt.

smoker
No
No
No
No
No

Column 5: [day]
This column denotes the day on which the customer visited the resturant or he/she payed the bill.

day
Sun
Sun
Sun
Sun
Sun

Column 6: [time]
This column provides with the timing of customer's visiting the resturant. It has only two values. I.e. whether the customer had came for dinner or lunch.

time
Dinner
Dinner
Dinner
Lunch
Lunch

Column 7: [size]
This column provides or shows the size of the people visited in one party. Means the total members along with the bill payer.

size
2
3
3
2
4

Exploratory Data Analysis (EDA):

head() : (It displays the first few rows of dataset).

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

shape : (It gives the number of rows and column in the dataset).

```
(434, 7)
```

info() : (It displays the non-null count and DataType of each column).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 434 entries, 0 to 433
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  434 non-null    float64
 1   tip         434 non-null    float64
 2   sex         434 non-null    object
 3   smoker      434 non-null    object
 4   day         434 non-null    object
 5   time        434 non-null    object
 6   size        434 non-null    int64
dtypes: float64(2), int64(1), object(4)
memory usage: 23.9+ KB
```

isnull().sum() : (It diplay the total missing values in each column).

```
total_bill    0
tip           0
sex           0
smoker        0
day           0
time          0
size          0
dtype: int64
```

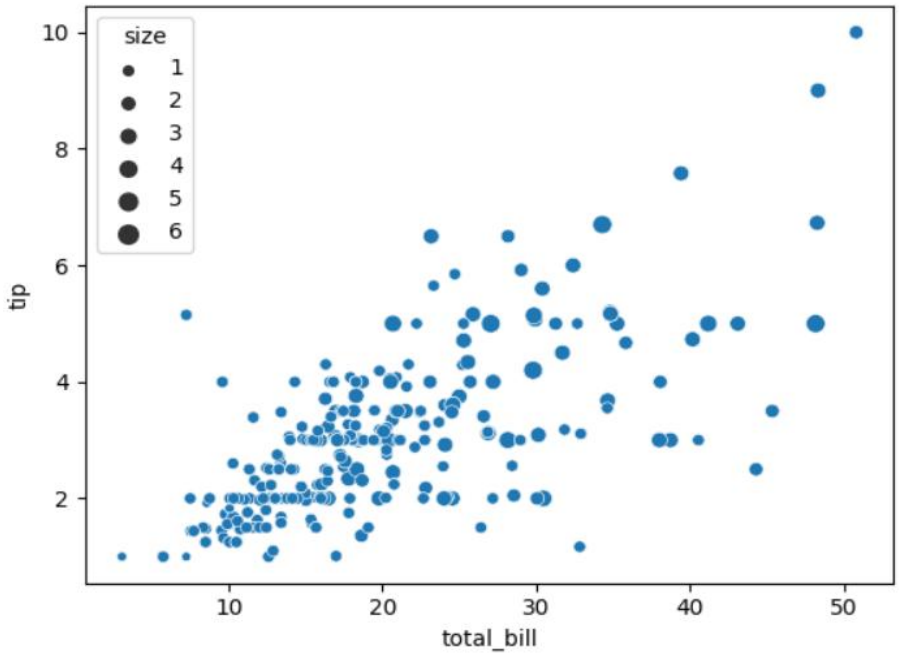
Conclusion : The provided dataset is clean and contains no missing values.

describe() : (It display the statistical values of each continous column).

	total_bill	tip	size
count	434.000000	434.000000	434.000000
mean	20.066429	3.061636	2.516129
std	9.132889	1.475769	0.917355
min	3.070000	1.000000	1.000000
25%	13.270000	2.000000	2.000000
50%	17.905000	3.000000	2.000000
75%	25.157500	3.607500	3.000000
max	50.810000	10.000000	6.000000

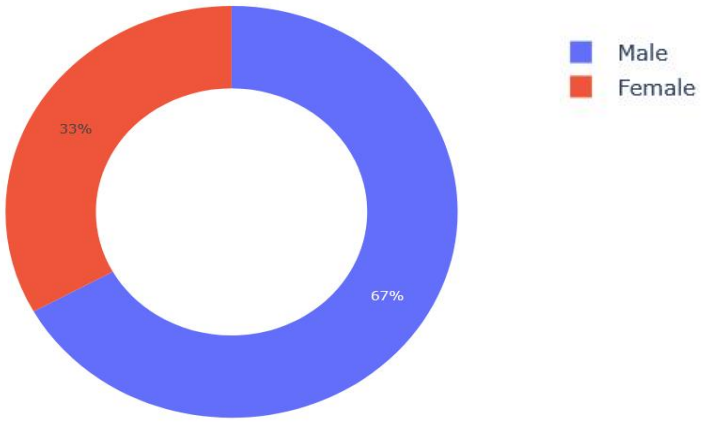
Visual presentation :

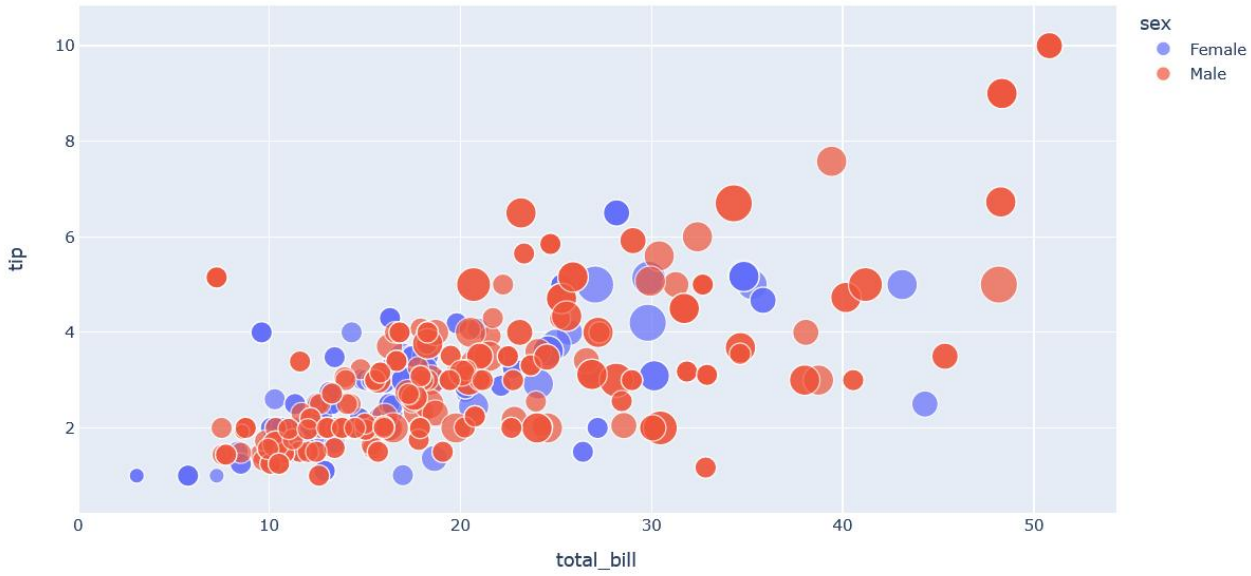
Below scatterPlot shows the **tips** given by the customer with respect to the amount of Bill.



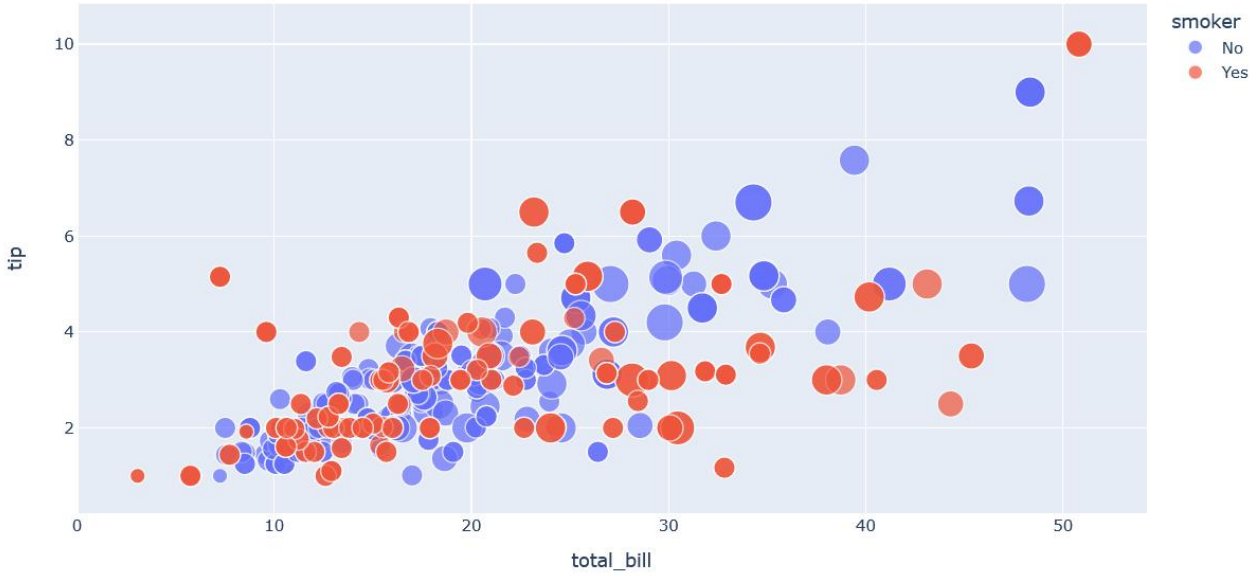
The tip given by the customer with respect to the amount of bill is categorised based on different features. Each feature is display through the scatter plot and donut plot.

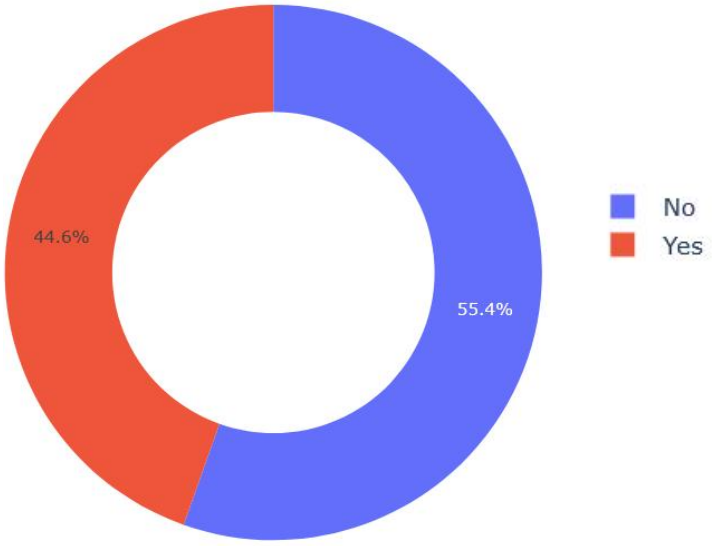
Gender of bill payer :



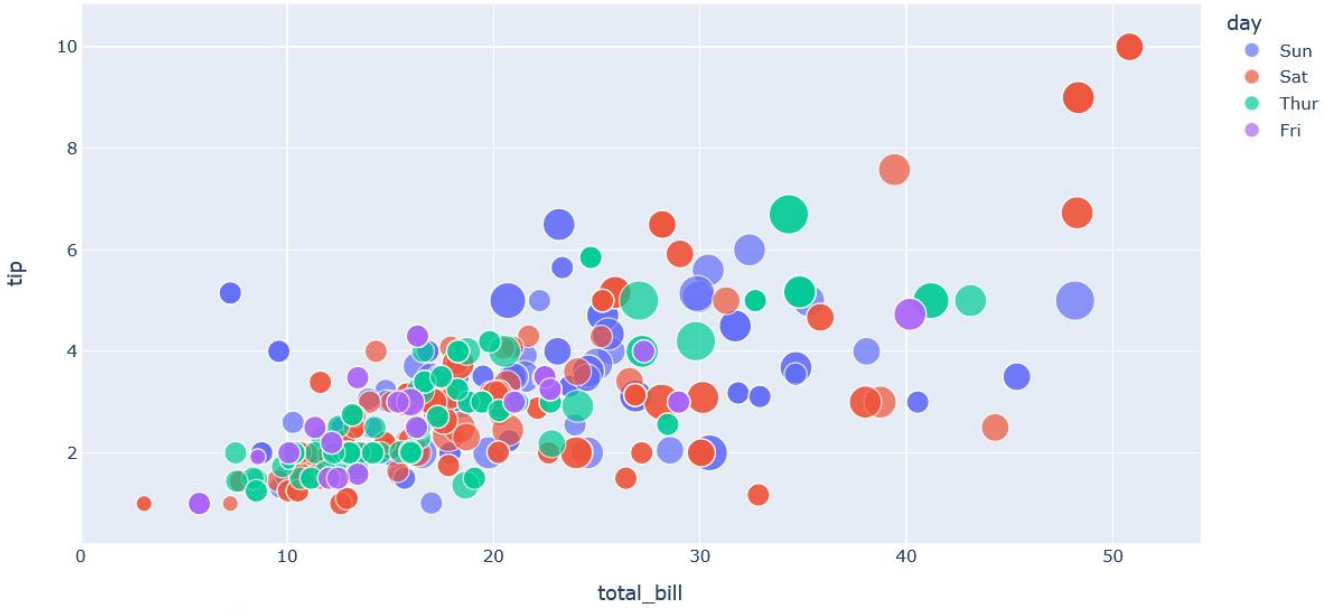

sex
Female
Male
tip
total_bill
0
10
20
30
40
50
2
4
6
8
10

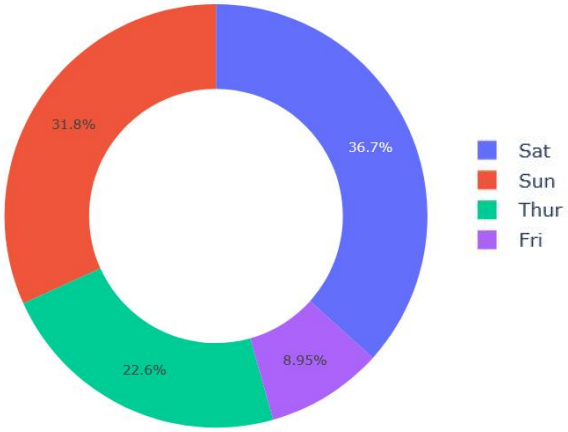
Smoker :


smoker
No
Yes
tip
total_bill
0
10
20
30
40
50
2
4
6
8
10

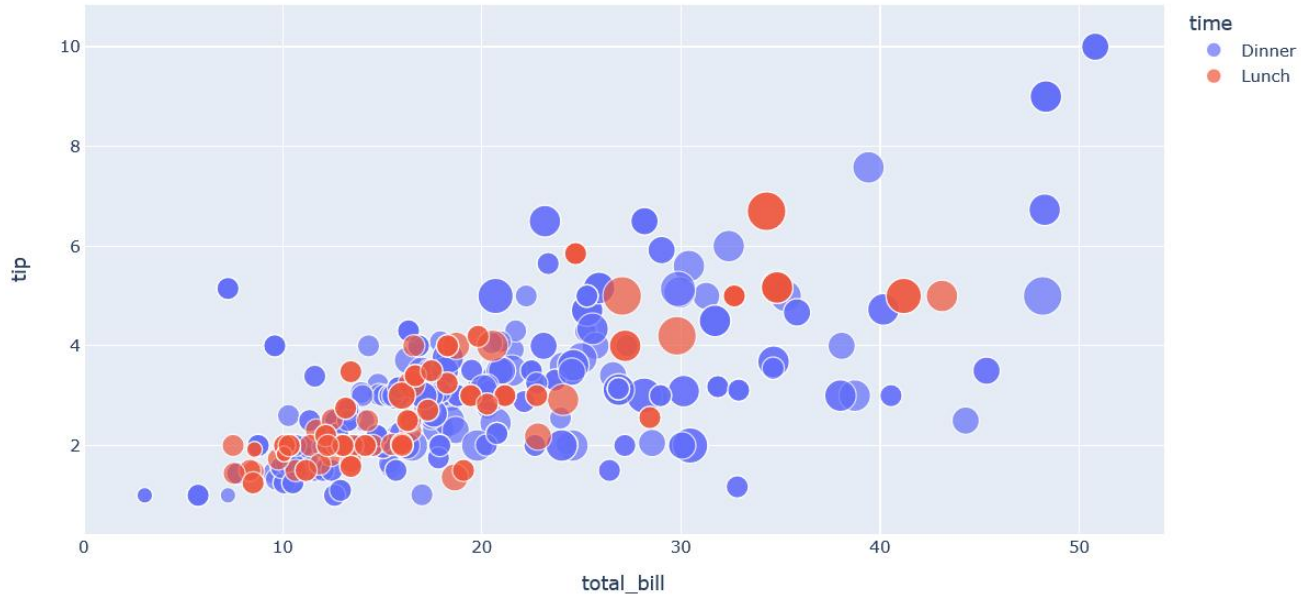

44.6%
55.4%
No
Yes

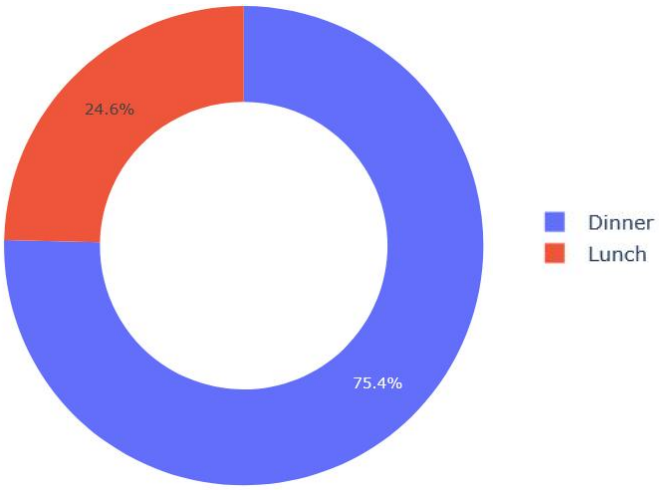
Days of the week:





Dinner or Lunch:





Feature Engineering:

Since, the four column (sex, smoker, day and Time) are continous data type.
So to make it more understandable for model training we will convert this column in encoded form.
This will help for Training a model and making it understand pattern in an efficient way.

sex: `['Female' 'Male']` => 0 , 1

smoker: `['No' 'Yes']` => 0 , 1

days: `['Sun' 'Sat' 'Thur' 'Fri']`

Time: `['Dinner' 'Lunch']` => 0 , 1

A new dataset is form after encoding the above columns.

	total_bill	tip	sex	smoker	time	size	Sat	Sun	Thur
0	16.99	1.01	0	0	0	2	False	True	False
1	10.34	1.66	1	0	0	3	False	True	False
2	21.01	3.50	1	0	0	3	False	True	False
3	23.68	3.31	1	0	0	2	False	True	False
4	24.59	3.61	0	0	0	4	False	True	False

The **days** column is removed and three new columns are formed. (Saturday, Sunday, Thursday).

The boolean value represents the specific day. If all the three days are False that means the customer was visited on fourth day. i.e Friday .

3. Libraries.(dependencies):

1- **pandas**: Pandas is a Python library used for working with data sets.
It has functions for analyzing, cleaning, exploring, and manipulating data
Pandas allows us to analyze big data and make conclusions based on statistical theories.
Pandas can clean messy data sets, and make them readable and relevant.
Relevant data is very important in data science.

2- **seaborn**:Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.
Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

3- **matplotlib***:* Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python
Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
Matplotlib is open source and we can use it freely.
Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

a. **Pyplot**: It is a library consisting of a collection of functions/methods used for plotting simple 2D graphs using Python. Pyplot can be imported using import matplotlib.
Pyplot is a Matplotlib module that provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python and the advantage of being free and open-source. Each pyplot function makes some changes to a figure

4- **sklearn**: sklearn or scikit-learn provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, etc.features of sk-learn:
- Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Accessible to everybody and reusable in various contexts.
- Built on the top of NumPy, SciPy, and matplotlib.
- Open source, commercially usable – BSD license.

a. model_selection : Sklearn's model selection module provides various functions to cross-validate our model**.**
is the task of selecting a model from among various candidates on the basis of performance criterion to choose the best one.
model_selection is used to split our data into train and test sets where feature variables are given as input in the method.

i. **train_test_split**:The train_test_split function splits arrays or matrices into random subsets for train and test data, respectively.
The train_test_split() method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train, X_test, y_train, and y_test. X_train and y_train sets are used for training and fitting the model. The X_test and y_test sets are used for testing the model if it's predicting

the right outputs/labels. we can explicitly test the size of the train and test sets. It is suggested to keep our train sets larger than the test sets.

b. **linear_model**: It is a class which contain different functions for performing machine learning with linear models. The term linear model implies that the model is specified as a linear combination of features. Based on training data, the learning process computes one weight for each feature to form a model that can predict or estimate the target value.

i. **LinearRegression**: Linear regression is a fundamental and widely used statistical technique for modeling the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables, meaning that changes in the dependent variable are assumed to be a constant times the changes in the independent variable(s). The simplest form of linear regression with one independent variable is often referred to as simple linear regression, while the case with multiple independent variables is called multiple linear regression.

c. **ensamble**:Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote.
It may help to improve the accuracy of the model. But, also it is difficult to understand an ensemble of classifiers.

i.**RandomForestRegressor**: Random Forest is one of the most popular and commonly used algorithms by Data Scientists Random forest is a Supervised Machine learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

Random forest is a versatile machine learning algorithm developed by Leo Breiman and Adele Cutler. Its widespread popularity stems from its user-friendly nature and adaptability, enabling it to tackle both classification and regression problems effectively.

ii.**GradientBoostingRegressor**: Gradient Boosting Regressor is a supervised machine learning algorithm that is part of the ensemble learning family. It belongs to the class of boosting algorithms, which combines the predictions of multiple weak learners (typically decision trees) to create a strong predictive model. Specifically, Gradient Boosting Regressor focuses on solving regression problems, where the goal is to predict a continuous target variable.

The algorithm minimizes the residuals (differences between predicted and actual values) using a specified loss function. It iteratively adds weak learners, adjusting their weights based on their contribution to reducing the overall loss. The learning process continues until a predefined number of learners or a specified accuracy level is reached. The algorithm often includes a learning rate parameter to control the contribution of each learner.

iii.**StackingRegressor**: StackingRegressor is an ensemble learning technique used for regression tasks. It combines the predictions of multiple base regressors by training a meta-regressor on their outputs. The base regressors can be diverse algorithms, such as linear regression, support vector machines, or decision trees. During training, the predictions from these base models are used as input features for the meta-regressor. Stacking helps capture diverse patterns in the data and can potentially improve predictive performance compared to individual models. StackingRegressor is known for its flexibility and effectiveness in handling complex relationships within the data, making it a powerful tool in regression tasks.

d. metrics: metrics are used to monitor and measure the performance of a model (during training and testing), and don't need to be differentiable.Metrics are numerical measures that quantify some aspect of your algorithm's behavior, such as accuracy, speed, complexity, or robustness. Metrics help you compare different algorithms, evaluate their strengths and weaknesses, and optimize their parameters.

i.**mean_absolute_error(MAE):** absolute error refers to the magnitude of difference between the prediction of an observation and the true value of that observation. MAE takes the average of absolute errors for a group of predictions and observations as a measurement of the magnitude of errors for the entire group. MAE can also be referred as L1 loss function.MAE helps users to formulate learning problems into optimization problems. It also serves as an easy-to-understand quantifiable measurement of errors for regression problems.

ii.**mean_squared_error:** The Mean Squared Error measures how close a regression line is to a set of data points. It is a risk function corresponding to the expected value of the squared error loss.

A larger MSE indicates that the data points are dispersed widely around its central moment (mean). A smaller MSE is preferred because it indicates that your data points are dispersed closely around its central moment (mean).

MSE = (1/n) * Σ(actual – forecast)2

- Σ – a symbol that means “sum”
- n – sample size
- actual – the actual data value
- forecast – the predicted data value

iii. **r2_score**: Coefficient of determination also called as R 2 score is used to evaluate the performance of a linear regression model. It is the amount of the variation in the output dependent attribute which is predictable from the input independent variable(s). It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model.

R 2= 1- SSres / SStot

Where,

SSres is the sum of squares of the residual errors.

SStot is the total sum of the errors.

e. preprocessing: sklearn.preprocessing is a module in the scikit-learn library, a popular machine learning library in Python. This module provides a collection of functions and classes for data preprocessing tasks. The purpose of these tools is to prepare and clean the input data before it is fed into a machine learning model. Some of the key functionalities provided by sklearn.preprocessing include scaling, normalization, encoding categorical variables, and handling missing values.

i.LabelEncoder: LabelEncoder is a class provided by the scikit-learn library (specifically in the sklearn.preprocessing module) designed to handle the encoding of categorical labels into numerical representations. It is a fundamental tool in the data preprocessing pipeline, particularly when dealing with machine learning algorithms that require numerical input.

f. svm: Support Vector Machines (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. It belongs to the family of discriminative models, aiming to find the optimal hyperplane that separates different classes in the feature space. SVM is particularly effective in high-dimensional spaces and is widely used in various fields, including image classification, text classification, and bioinformatics.

Key concept:

- In a binary classification, svm finds best seperateline called hyperplane.
- Support vectors are the data points that lie closest to the decision boundary (hyperplane).
- The margin is the distance between the decision boundary (hyperplane) and the nearest data point from either class.

i. **SVR (support vector regressor):** Support Vector Regression (SVR) is a variant of Support Vector Machines (SVM) that is used for regression tasks rather than classification. SVR is a powerful machine learning algorithm that is particularly effective in situations where linear regression models may not perform well, especially when dealing with non-linear relationships between features and the target variable.

g. neighbors: neighbors is a module in the scikit-learn library, which is a popular machine learning library in Python. This module provides implementations of algorithms related to nearest neighbors. Nearest Neighbors methods are used for classification, regression, and density estimation tasks.

Furthermore, the neighbors module allows users to fine-tune the behavior of nearest neighbors algorithms by specifying parameters such as the number of neighbors, the type of distance metric, and the algorithm used for computation. This flexibility makes it adaptable to diverse datasets and problem domains. Whether for simple tasks like nearest centroid classification or more complex density estimation using kernel methods, the module serves as a valuable resource for practitioners seeking efficient and effective solutions to proximity-based machine learning challenges.

ii.**KNeighborsRegressor**: K-Nearest Neighbors (KNN) is a simple and versatile machine learning algorithm used for both classification and regression tasks. In KNN, predictions are made based on the majority class or average of the k-nearest data points in the feature space. The algorithm operates on the principle that similar data points tend to have similar outcomes. It doesn't involve explicit training; instead, it classifies or predicts new instances by comparing them to the labeled instances in the training set. The choice of the number of neighbors (k) and the distance metric significantly influences KNN's performance, allowing users to tailor the algorithm to the characteristics of their data.

5- **pickle**: In Python, we sometimes need to save the object on the disk for later use. This can be done by using Python pickle. Pickle in Python is primarily used in serializing and deserializing a Python object structure. In other words, it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network.

6- **math:** math is a built-in module in the Python 3 standard library that provides standard mathematical constants and functions. You can use the math module to perform various mathematical calculations, such as numeric, trigonometric, logarithmic, and exponential calculations.
Math module provides value of various constants like pi, tau, Euler's, etc. Having such constants saves the time of writing the value of each constant every time we want to use it and that too with great precision.

7- **plotly:** Plotly is a versatile and interactive data visualization library that allows users to create a wide range of static and interactive plots and charts. It is available in various programming languages, including Python, JavaScript, and R. Plotly supports a multitude of chart types and is known for its ease of use, customization options, and interactivity features.

a. express : Plotly Express is a high-level interface within the Plotly library designed for creating a variety of interactive visualizations with minimal code. It simplifies the process of

generating complex plots and charts, making it an ideal choice for users who want to quickly create compelling visuals without delving into intricate customization details.

- It is known for its straightforward syntax, allowing users to create expressive visualizations with minimal lines of code.
- Express simplifies color and size mapping for data points, making it easy to represent addi5tional dimensions in the visualizations.

8- **flask:**

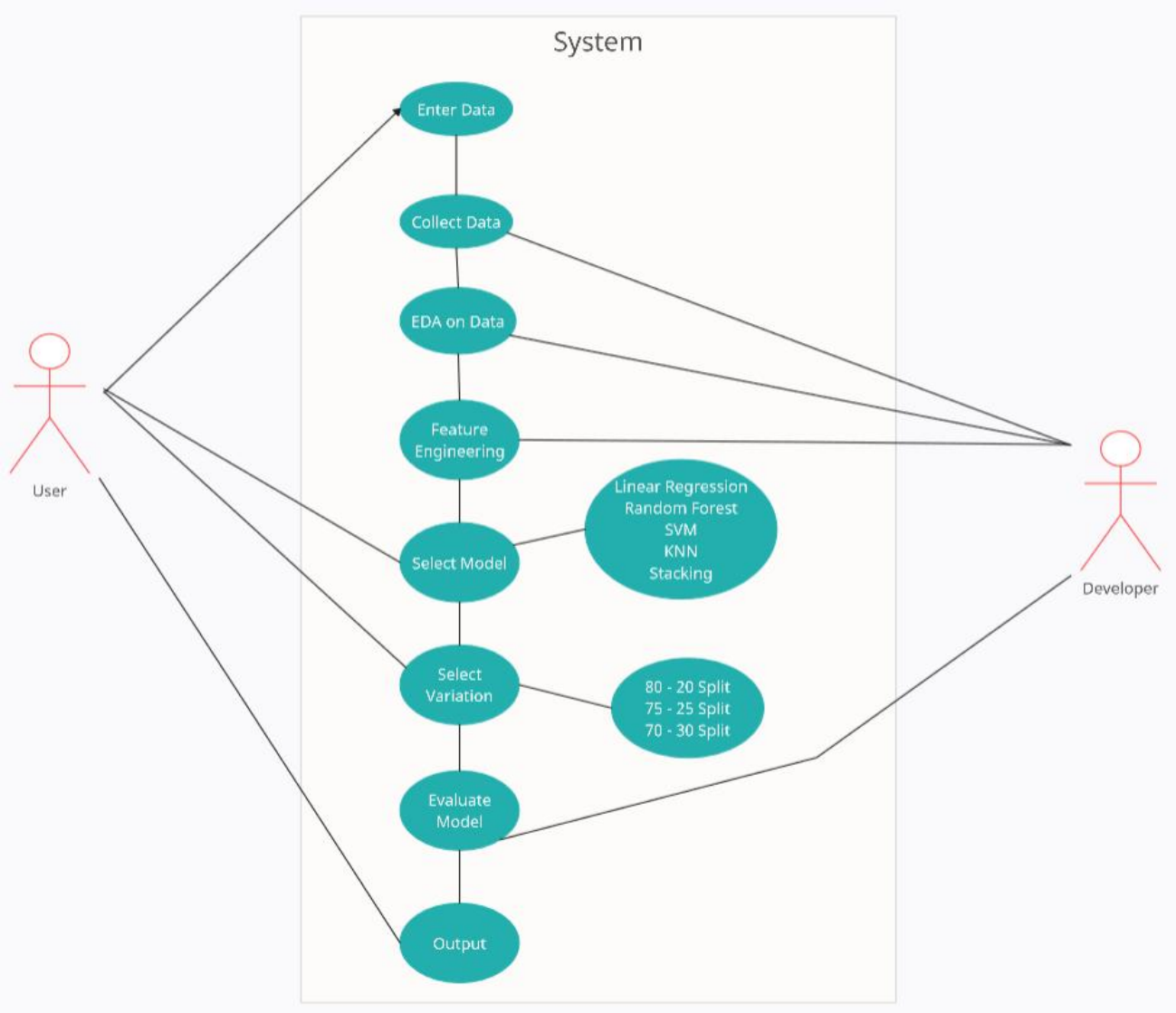
a. Flask: Flask is a lightweight and versatile web framework for Python, designed to make web development simple and scalable. Developed by Armin Ronacher, Flask follows the WSGI (Web Server Gateway Interface) standard and provides the essentials for building web applications. One of its key features is its simplicity, allowing developers to get started quickly and only use what they need. Flask embraces the concept of "micro-framework," offering a core set of functionalities for handling routing, request handling, and templating. Its modular design encourages the use of extensions for additional features, enabling developers to customize and extend the framework according to their project requirements. Flask is well-suited for small to medium-sized projects and serves as an excellent choice for those who prefer flexibility and a minimalistic approach to web development.

b. render_template: In Flask, render_template is a function that plays a crucial role in generating dynamic HTML pages for web applications. This function is part of Flask's templating engine, which allows developers to create templates (HTML files) with placeholders for dynamic content. When a route in a Flask application renders a template using the render_template function, it replaces placeholders with actual data, making the content dynamic. This enhances the separation of concerns in web development, enabling developers to focus on both the application's logic in Python and the presentation in HTML. By utilizing render_template, Flask facilitates the creation of dynamic and maintainable web applications, where the backend and frontend components are cleanly separated, promoting a more organized and scalable development process.

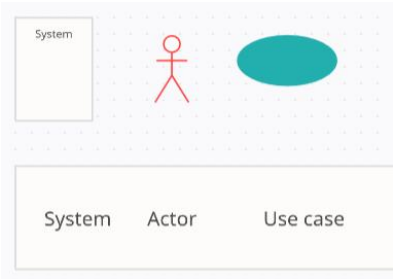
c. Request: In Flask, the request object is a crucial component that provides access to incoming HTTP request data within a web application. It allows developers to access parameters, form data, cookies, and other information sent by a client's browser during an HTTP request. The request object is part of the Flask library and simplifies the process of extracting and handling data from client requests. Developers can utilize the request object to retrieve data from query parameters, access form submissions, or manage cookie information, enabling dynamic and interactive behavior in Flask applications. Its versatility makes it an essential tool for processing user input and building responsive web applications that can adapt to various client interactions.

4. UML DIAGRAM

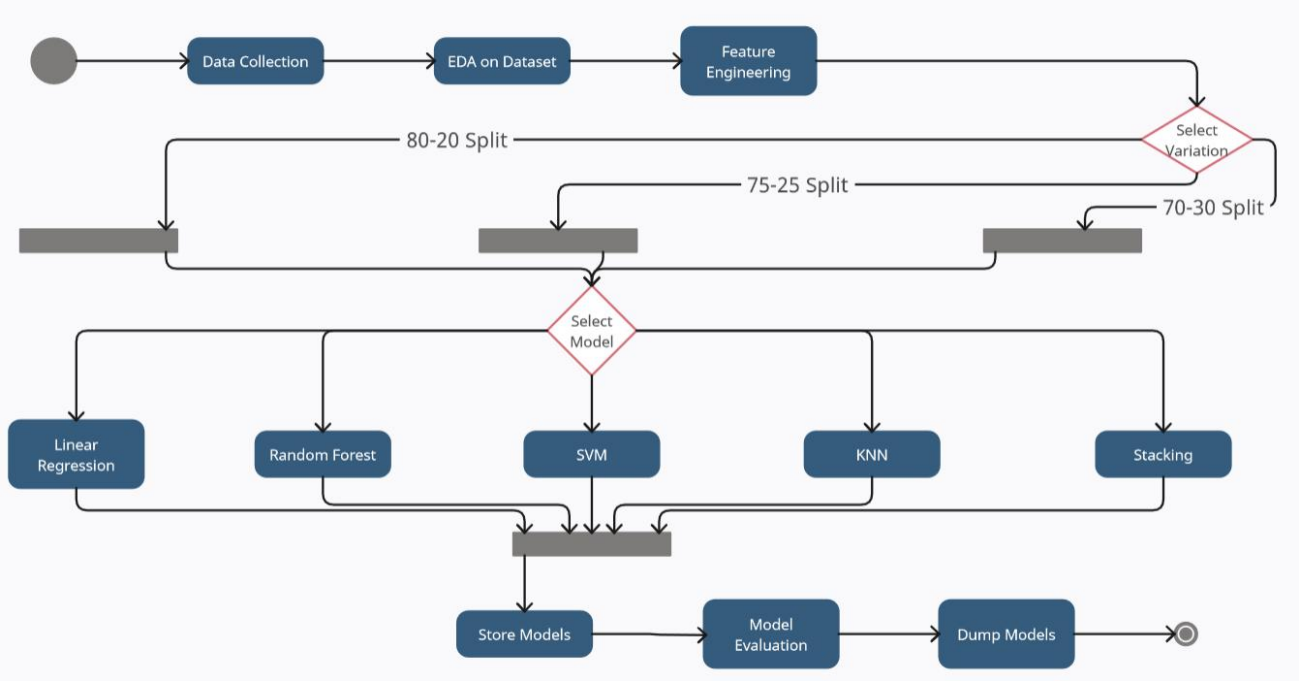
A. **USE Case:** A use case is a graphical representation of actions that describes the behavior of a system to do a particular task.



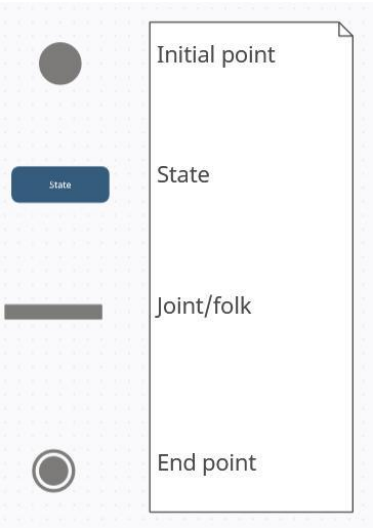
Use case Notation:



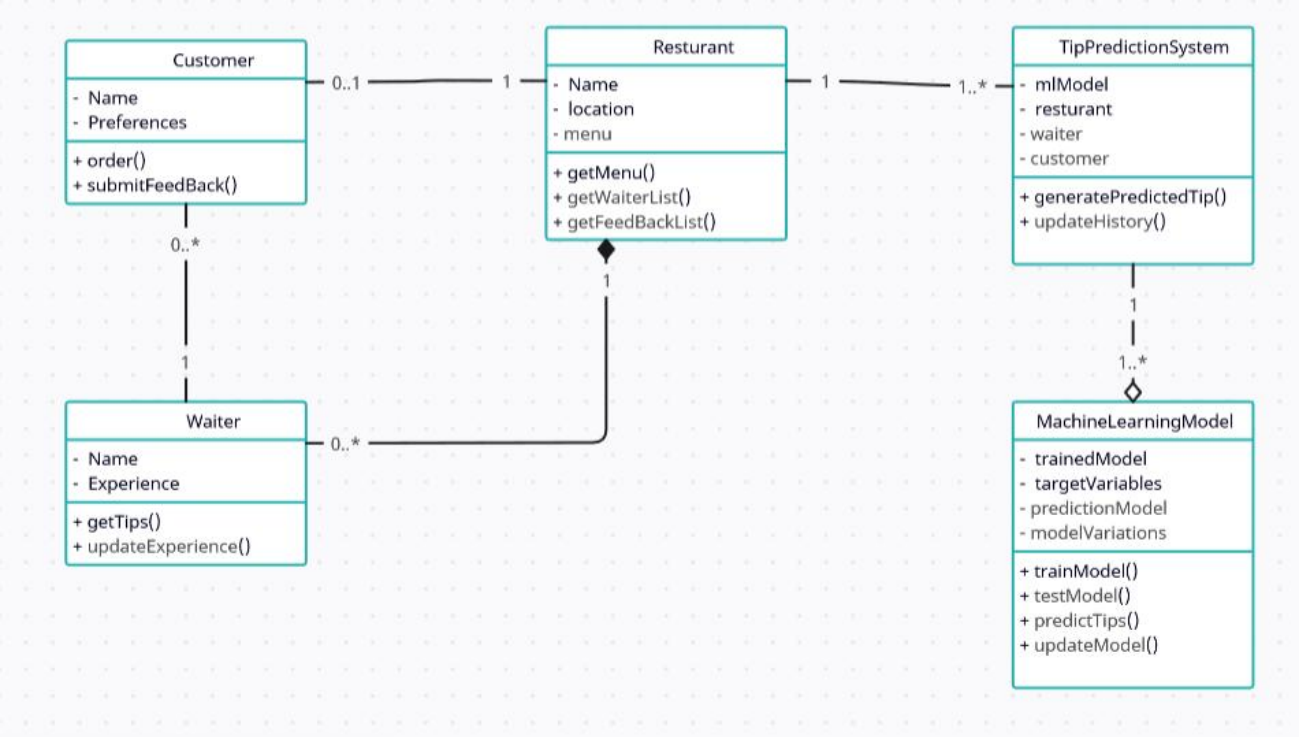
B. State Chart diagram: A state diagram is an illustration of all the possible behavioral states a software system component may exhibit. state diagrams are specifically designed to focus on the state of an object.



State Chart Notation:



C. Class diagram: A class diagram visually represents the structure and relationships of classes within a system or software application. It provides a high-level view of the static structure of the system, showing classes, their attributes, methods, and the associations between classes.



5.MODEL DESCRIPTION:

i. *Linear regression model*:

How to import linear regression from sci-kit learn(sklearn) module.

```
from sklearn.linear_model import LinearRegression
```

Supervised learning has two types:

- **Classification**: It predicts the class of the dataset based on the independent input variable. Class is the categorical or discrete values. like the image of an animal is a cat or dog?
- **Regression**: It predicts the continuous output variables based on the independent input variable. like the prediction of house prices based on different parameters like house age, distance from the main road, location, area, etc.

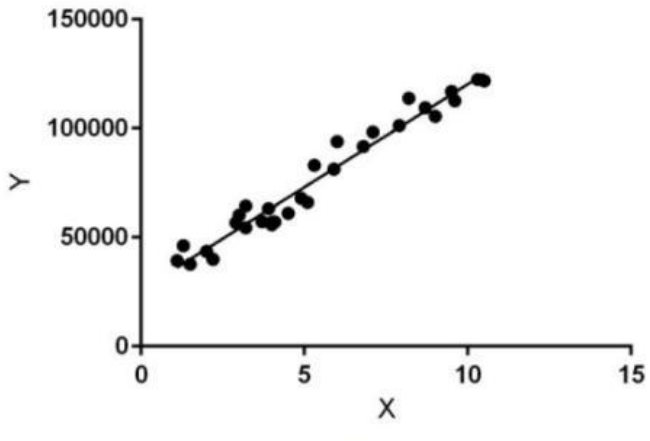
What is Linear Regression:

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression. The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).

One of the most important supervised learning tasks is regression. In regression set of records are present with X and Y values and these values are used to learn a function so if you want to predict Y from an unknown X this learned function can be used. In regression we have to find the value of Y, So, a function is required that predicts continuous Y in the case of regression given X as independent features.

Here Y is called a dependent or target variable and X is called an independent variable also known as the predictor of Y. There are many types of functions or modules that can be used for regression. A linear function is the simplest type of function. Here, X may be a single feature or multiple features representing the problem.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x)). Hence, the name is Linear Regression. In the figure above,X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best-fit line for our model.



Linear Regression

Assumption for Linear Regression Model
Linear regression is a powerful tool for understanding and predicting the behavior of a variable, however, it needs to meet a few conditions in order to be accurate and dependable solutions.
1. **Linearity**: The independent and dependent variables have a linear relationship with one another. This implies that changes in the dependent variable follow those in the independent variable(s) in a linear fashion.
2. **Independence**: The observations in the dataset are independent of each other. This means that the value of the dependent variable for one observation does not depend on the value of the dependent variable for another observation.
3. **Homoscedasticity**: Across all levels of the independent variable(s), the variance of the errors is constant. This indicates that the amount of the independent variable(s) has no impact on the variance of the errors.
4. **Normality**: The errors in the model are normally distributed.
5. **No multicollinearity**: There is no high correlation between the independent variables. This indicates that there is little or no correlation between the independent variables.

Hypothesis function for Linear Regression :
As we have assumed earlier that our independent feature is the experience i.e X and the respective salary Y is the dependent variable. Let's assume there is a linear relationship between X and Y then the salary can be predicted using:

$$\hat{Y} = \theta_1 + \theta_2 X$$

OR

$$\hat{y}_i = \theta_1 + \theta_2 x_i$$

Here,

- $y_i \epsilon Y \quad (i = 1, 2, \cdots, n)$ are labels to data (Supervised learning)
- $x_i \epsilon X \quad (i = 1, 2, \cdots, n)$ are the input independent training data (univariate – one input variable(parameter))
- $\hat{y}_i \epsilon \hat{Y} \quad (i = 1, 2, \cdots, n)$ are the predicted values.

The model gets the best regression fit line by finding the best θ1 and θ2 values.
- **θ1:** intercept
- **θ2:** coefficient of x

Once we find the best θ1 and θ2 values, we get the best-fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

ii. *Random Forest (Regressor) model:*

How to import random forest from sci-kit learn(sklearn) module.

```
from sklearn.ensemble import RandomForestRegressor
```

What is Random Forest *Regressor*:
Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
Note:(***The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.***)

Assumptions for Random Forest
the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output.
- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Why random forest is better then other models?
- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.
- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the over fitting issue.

Working of random forest algorithm. Random Forest works in two-phase first is to create the random forest by combining N
decision tree, and second is to make predictions for each tree created in the first phase.
The Working process can be explained in the below steps and diagram:
Step-1: Select random K data points from the training set.
Step-2: Build the decision trees associated with the selected data points (Subsets).
Step-3: Choose the number N for decision trees that you want to build.
Step-4: Repeat Step 1 & 2.
Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

iii. *SVR model:*
How to import support vector regression fro sci-kit learn (sklearn) module.

```
from sklearn.svm import SVR as svr
```

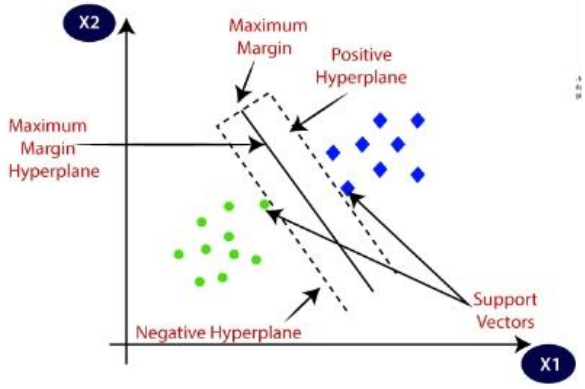
What is Support Vector Machine.

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider

the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane classified using a decision boundary or hyperplane:



Support Vector Machine (SVM) is a powerful machine learning algorithm used for linear or nonlinear classification, regression, and even outlier detection tasks. SVMs can be used for a variety of tasks, such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection, and anomaly detection. SVMs are adaptable and efficient in a variety of applications because they can manage high-dimensional data and nonlinear relationships.

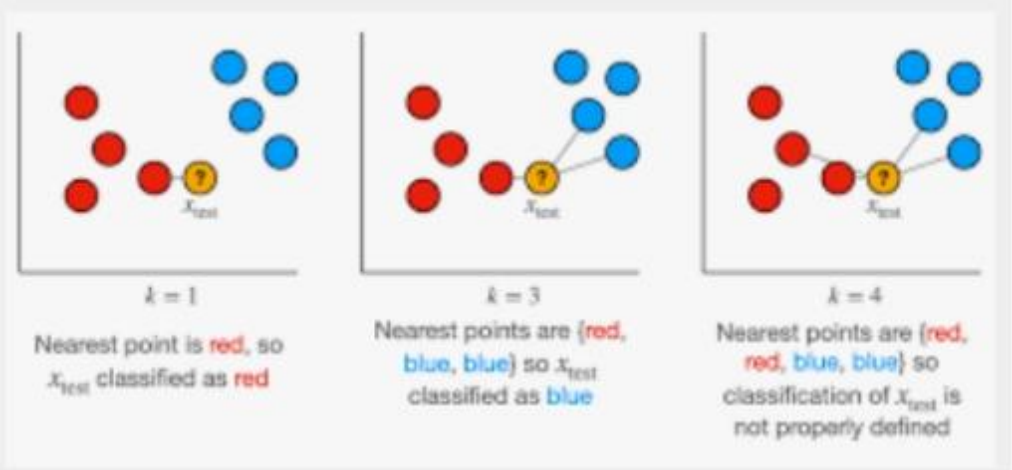
SVM algorithms are very effective as we try to find the maximum separating hyperplane between the different classes available in the target feature.

iv. K-Nearest Neighbors (KNN) model :

How to import k-nearest neighbors fro sci-kit learn (sklearn) module.

```
from sklearn.neighbors import KNeighborsRegressor
```

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.



Regression problems use a similar concept as classification problem, but in this case, the average the k nearest neighbors is taken to make a prediction about a classification. The main distinction

here is that classification is used for discrete values, whereas regression is used with continuous ones. However, before a classification can be made, the distance must be defined. Euclidean distance is most commonly used, which we'll delve into more below.

It's also worth noting that the KNN algorithm is also part of a family of "lazy learning" models, meaning that it only stores a training dataset versus undergoing a training stage. This also means that all the computation occurs when a classification or prediction is being made. Since it heavily relies on memory to store all its training data, it is also referred to as an instance-based or memory-based learning method.

Here's a step-by-step explanation of how the KNN algorithm works:

Training Phase: Store all the training examples in memory. Each example consists of a set of features and a corresponding label or output.

Input Data: When a new, unlabeled data point is to be classified (in the case of classification) or predicted (in the case of regression), the algorithm takes this data point as input.

Calculate Distances: Calculate the distance between the input data point and every point in the training dataset. Common distance metrics include Euclidean distance, Manhattan distance, or other distance measures depending on the problem.

Identify Nearest Neighbors: Identify the k training examples with the smallest distances to the input data point. These k instances are the "nearest neighbors."

Majority Voting (Classification) or Weighted Average (Regression): For classification tasks, assign the class label that is most common among the k nearest neighbors. This is often done through a majority voting mechanism. For regression tasks, predict the output for the new data point as the average (weighted or unweighted) of the outputs of its k nearest neighbors.

Output: The predicted class label (for classification) or predicted value (for regression) is assigned to the new data point.

Model Evaluation: The performance of the KNN model can be evaluated using metrics such as accuracy, precision, recall, F1 score (for classification), or mean squared error (for regression) on a separate test dataset.

Key Considerations:

- The choice of the distance metric and the value of k can significantly impact the performance of the KNN algorithm.
- KNN is sensitive to the scale of the features, so it's often a good practice to normalize or standardize the data.
- The computational cost of predicting a new instance can be relatively high, especially with large datasets, as it requires calculating distances to all training instances.

v. Stacking Regressor model:

How to import Stacking Regressor from sci-kit learn(sklearn) module.

```
from sklearn.ensemble import StackingRegressor
```

What is Stacking *Regressor*:
Stacking is an ensemble learning technique that combines the predictions of multiple base regressors to improve overall performance. In the context of regression, a Stacking Regressor involves training a set of diverse base regression models and then using a meta-regressor to blend their predictions.
The base regressors can be different algorithms or variations of the same algorithm with different hyperparameters. The meta-regressor takes the predictions of the base models as input and learns to weigh or combine them effectively to make a final prediction. This allows the stacking regressor to capture complex patterns in the data and potentially outperform individual models. Stacking is a powerful approach in machine learning, providing a way to leverage the strengths of various models and enhance predictive accuracy. Properly configured, a stacking regressor can often achieve better performance than its constituent base models, making it a valuable tool in predictive modeling.

The key idea behind stacking is that different base regressors capture different aspects of the underlying patterns in the data. By combining their predictions, the stacking regressor aims to leverage the strengths of each base model, potentially improving overall predictive performance. In this specific example, the Random Forest and XGBoost models bring the power of ensemble learning and non-linearity, while the Linear Regression meta-model provides a linear combination of their outputs. This combination allows for flexibility in capturing both linear and non-linear relationships in the data. The choice of models and their hyperparameters should be carefully tuned to achieve optimal performance.

In a stacking regressor with Random Forest, XGBoost, and Linear Regression, the workflow involves training multiple base regressors and then combining their predictions using a meta-regressor. Here's an overview of the process:

1. **Base Regressors Training:**
    - Train individual base regressors, such as Random Forest Regressor and XGBoost Regressor, on the training data. Each base regressor learns patterns in the data and makes predictions.
2. **Base Regressor Predictions:**
    - Use the trained base regressors to make predictions on the validation or test set. The predictions from each base regressor serve as input features for the meta-regressor.
3. **Meta Regressor Training:**
    - Combine the predictions of the base regressors and use them as input features for the meta-regressor, which is typically a Linear Regression model in this case. Train the meta-regressor on the same training set, with the true target values as the target variable.
4. **Final Prediction:**
    - Once the meta-regressor is trained, it can be used to make predictions on new, unseen data. The predictions from the base regressors are fed into the meta-regressor, which then combines them to produce the final prediction.
5. **Model Evaluation:**
    - Evaluate the performance of the stacking regressor on a separate validation or test set. Common regression metrics such as Mean Squared Error (MSE) or R-squared can be used to assess the model's accuracy.

6. EVALUATION OF THE MODELS :

Making Report of models in three different variation:

i. 80-20 split.
ii. 75-25 split.
iii. 70-30 split.

The Report of each model in each variations include:

i. r2-score.
ii. mean_square_error.
iii. root_mean_sqare_error.
iv. mean_absolute_error.

a) Linear Regression:

Train Test split at 80-20
r2 score of Linear regression model : 0.4876761949501752
mean square error of Linear regression model : 1.3275615473369886
Root mean square error of Linear regression model : 1.15219857113997
mean absolute error of Linear regression model : 0.8051900010738701

Train Test split at 75-25
r2 score of Linear regression model : 0.4791884649778383
mean square error of Linear regression model : 1.261659241118019
Root mean square error of Linear regression model : 1.1232360576112304
mean absolute error of Linear regression model : 0.7899047320740296

Train Test split at 70-30
r2 score of Linear regression model : 0.481926514987397
mean square error of Linear regression model : 1.1626302053060065
Root mean square error of Linear regression model : 1.0782533122165712
mean absolute error of Linear regression model : 0.7572458989767376

b) Random Forest Regressor:

Train Test split at 80-20
r2 score of Random Forest model : 0.9215427955105558
mean square error of Random Forest model : 0.20330261206896583
Root mean square error of Random Forest model : 0.4508909092773615
mean absolute error of Random Forest model : 0.30964482758620715

Train Test split at 75-25
r2 score of Random Forest model : 0.8784971868914334
mean square error of Random Forest model : 0.29433900110091754
Root mean square error of Random Forest model : 0.5425301845067402
mean absolute error of Random Forest model : 0.3544458715596331

Train Test split at 70-30
r2 score of Random Forest model : 0.8711417708644489
mean square error of Random Forest model : 0.28917609900763364
Root mean square error of Random Forest model : 0.5377509637440305
mean absolute error of Random Forest model : 0.3766801526717559

c) SVM (*SVR*):

Train Test split at 80-20
r2 score of SVM model : 0.4748289607443996
mean square error of SVM model : 1.3608520053502784
Root mean square error of SVM model : 1.1665556160553505
mean absolute error of SVM model : 0.8063947371968888

Train Test split at 75-25
r2 score of SVM model : 0.46261920634161546
mean square error of SVM model : 1.3017980569297225
Root mean square error of SVM model : 1.1409636527645053
mean absolute error of SVM model : 0.7868806209696599

Train Test split at 70-30
r2 score of SVM model : 0.46981154679543646
mean square error of SVM model : 1.1898179081392295
Root mean square error of SVM model : 1.0907877466029903
mean absolute error of SVM model : 0.7581554984958091

d) KNN:

Train Test split at 80-20
r2 score of KNN model : 0.7414542791775829
mean square error of KNN model : 0.6699578544061303
Root mean square error of KNN model : 0.8185095322634491
mean absolute error of KNN model : 0.6199233716475094

Train Test split at 75-25
r2 score of KNN model : 0.6833551839677601
mean square error of KNN model : 0.7670679918450559
Root mean square error of KNN model : 0.875824178614096
mean absolute error of KNN model : 0.6525688073394496

Train Test split at 70-30
r2 score of KNN model : 0.6933741517152743
mean square error of KNN model : 0.6881117896522476
Root mean square error of KNN model : 0.8295250385927164
mean absolute error of KNN model : 0.6195928753180662

e) Stacking Regressor:

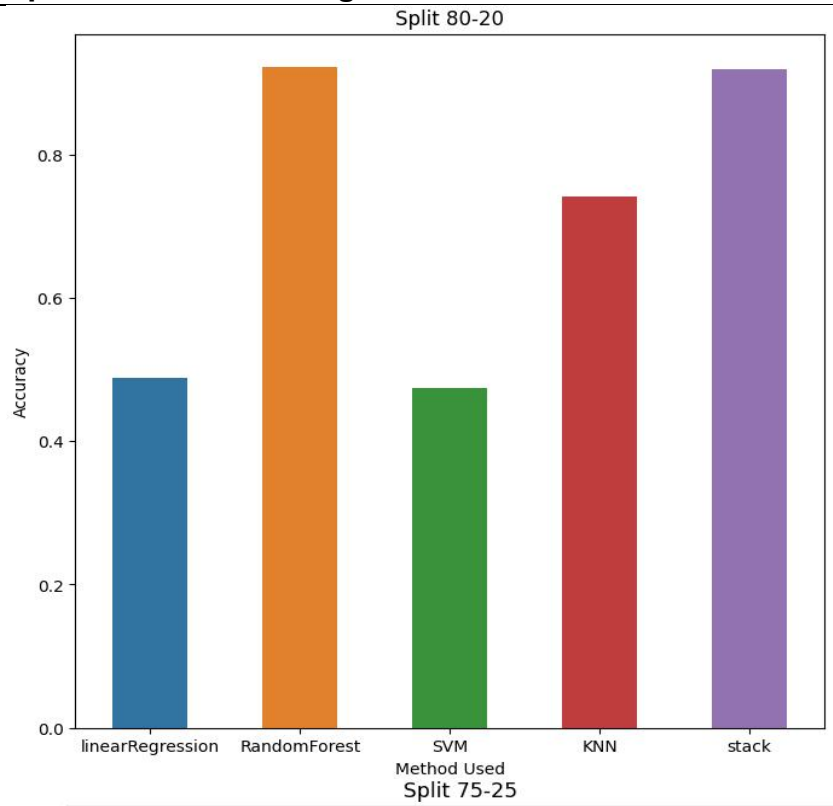
Train Test split at 80-20
r2 score of stack model : 0.9276981488865018
mean square error of stack model : 0.18735252274726796
Root mean square error of stack model : 0.432842376330308
mean absolute error of stack model : 0.29549924826312635

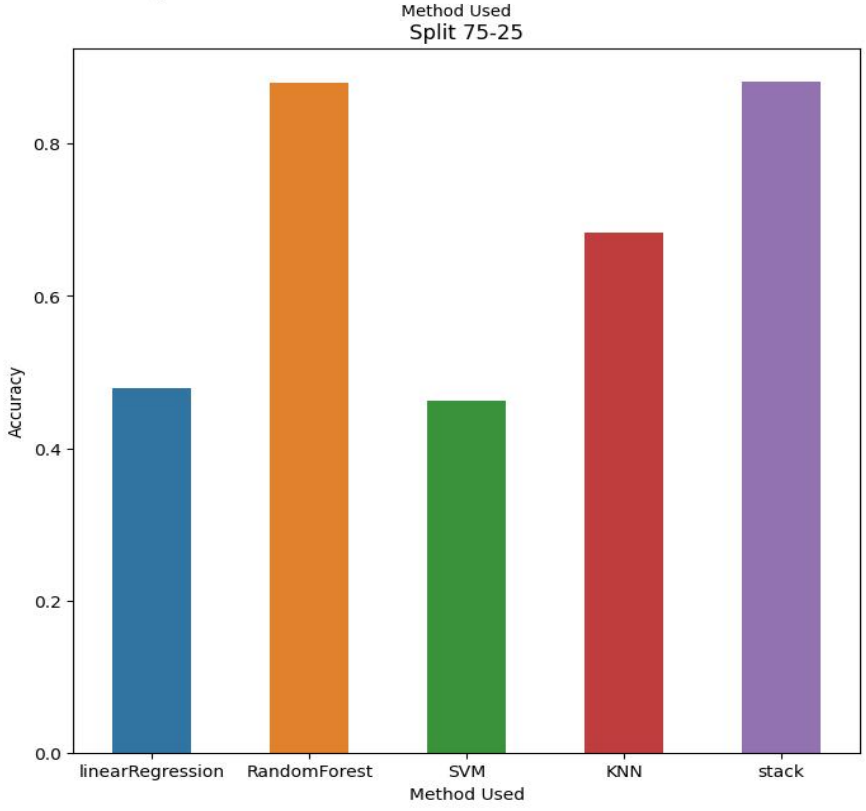
Train Test split at 75-25
r2 score of stack model : 0.8833835333956899
mean square error of stack model : 0.28250188957815237
Root mean square error of stack model : 0.5315090681993604
mean absolute error of stack model : 0.34086296256674975

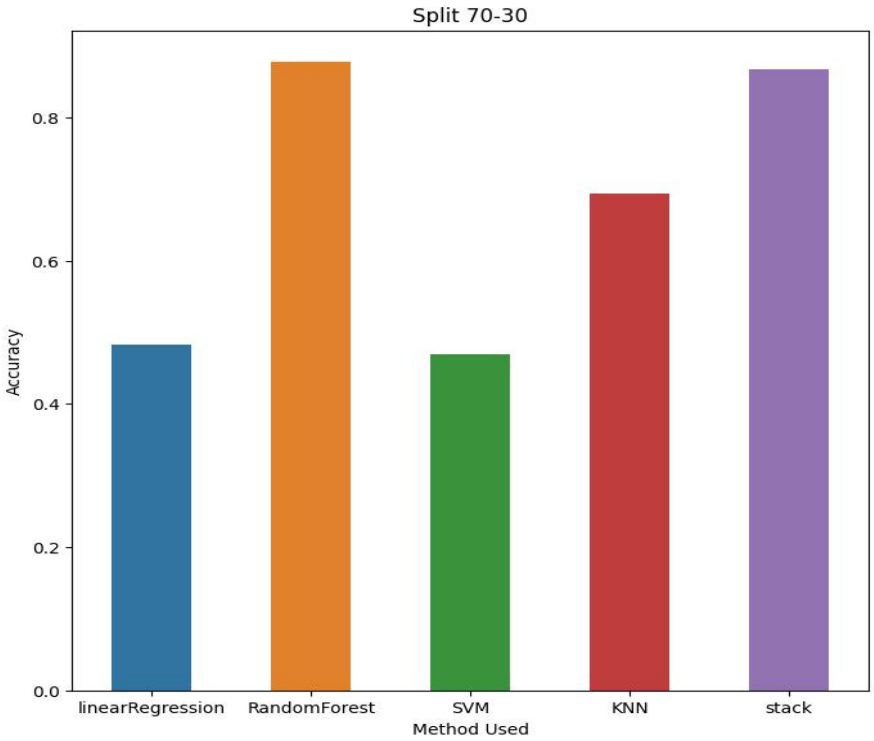
Train Test split at 70-30

r2 score of stack model : 0.8767987532828145
mean square error of stack model : 0.2764810300246745
Root mean square error of stack model : 0.5258146346619448
mean absolute error of stack model : 0.3731317268155549

Now we will compare each models together in different variations through barplot.

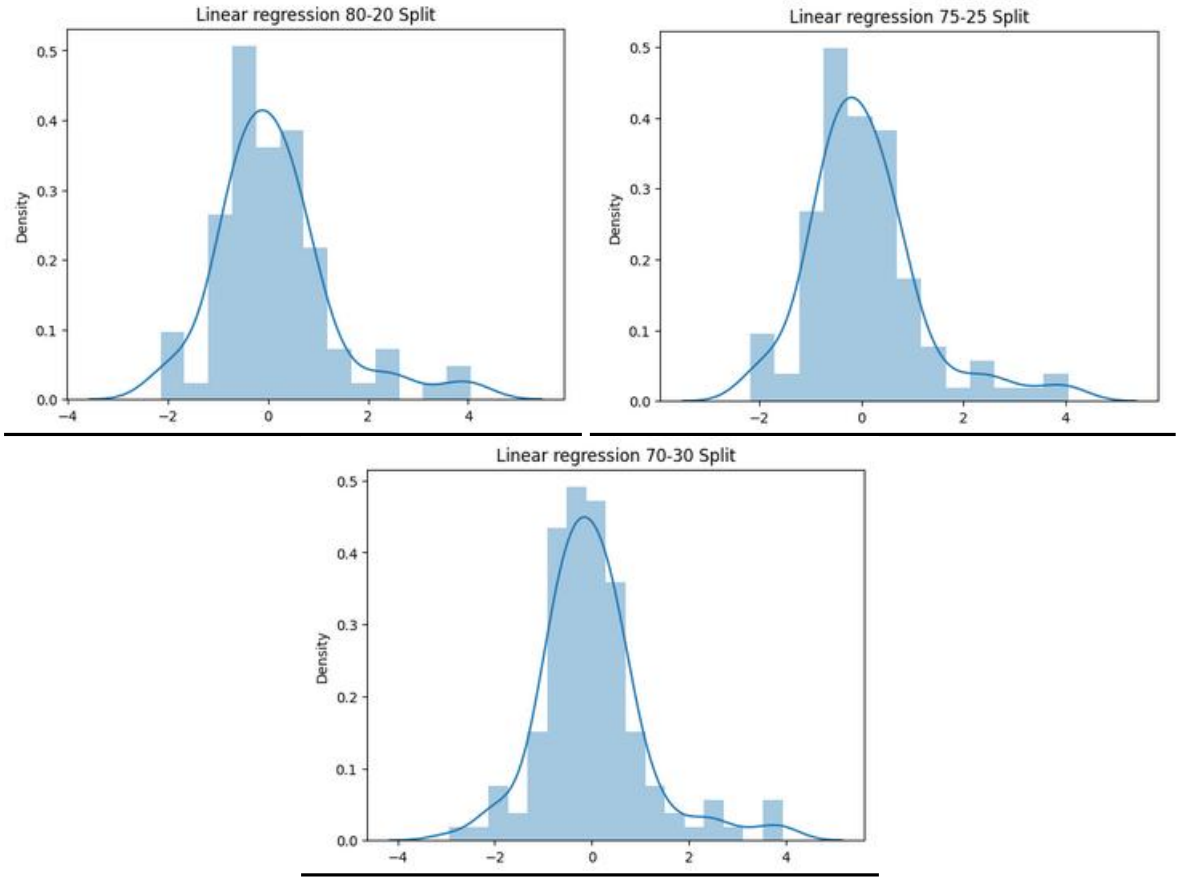




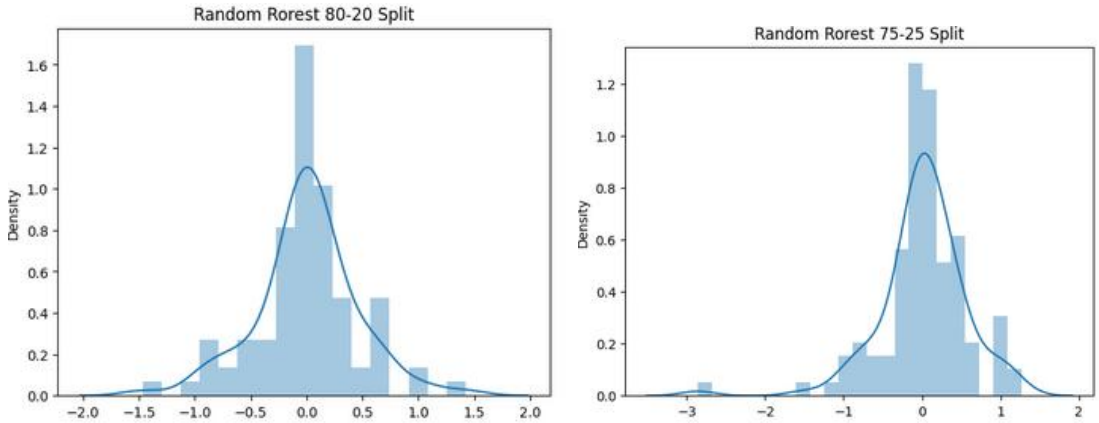


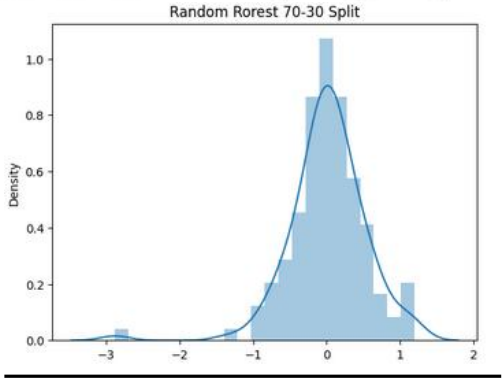
Now we'll examine the variance of each model in different variations using Distplot.

a) Linear Regression:

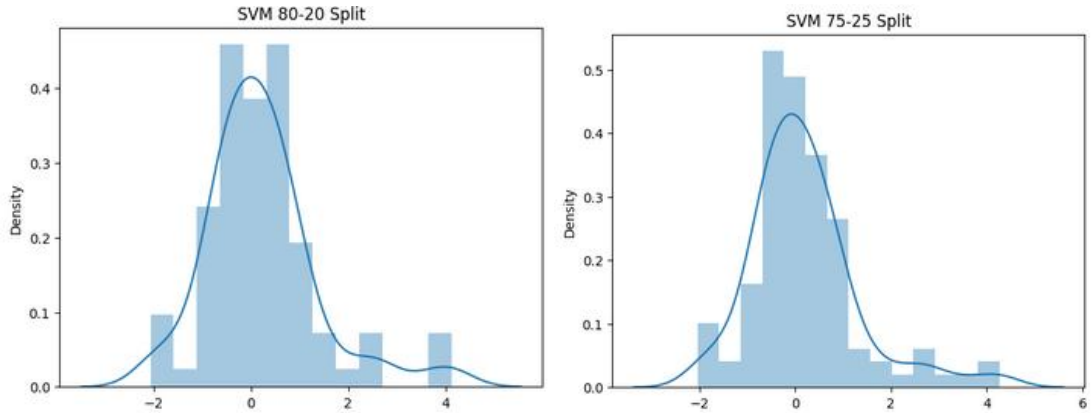


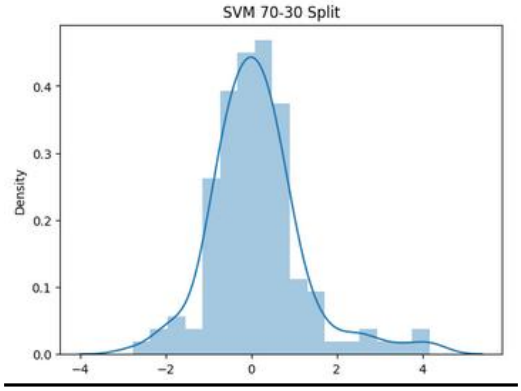
b) Random Forest Regressor:



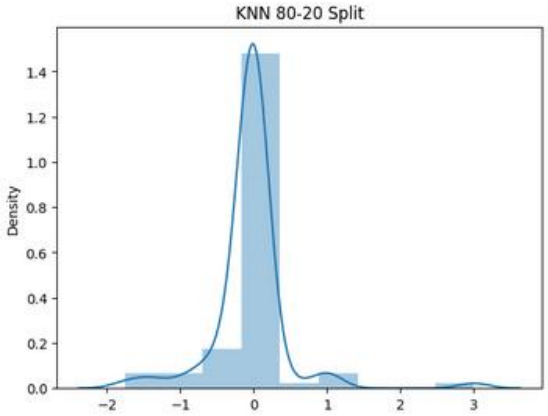


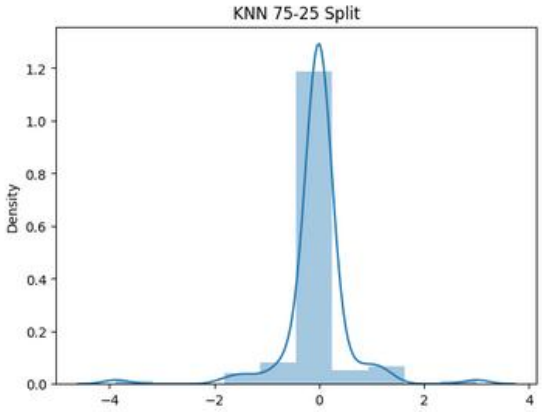
c) SVM (*SVR)*:

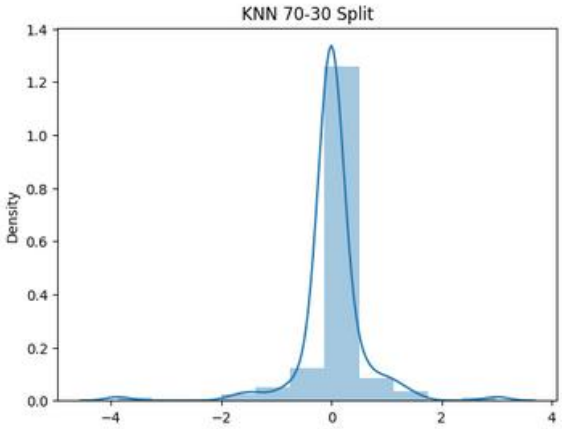




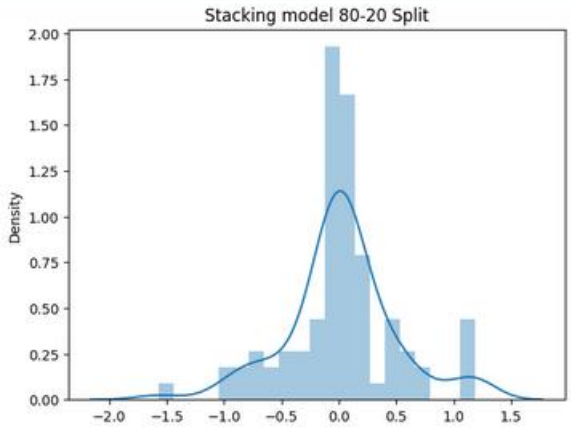
d) KNN:

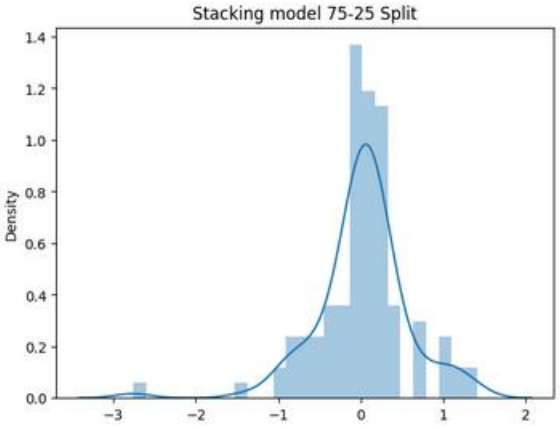


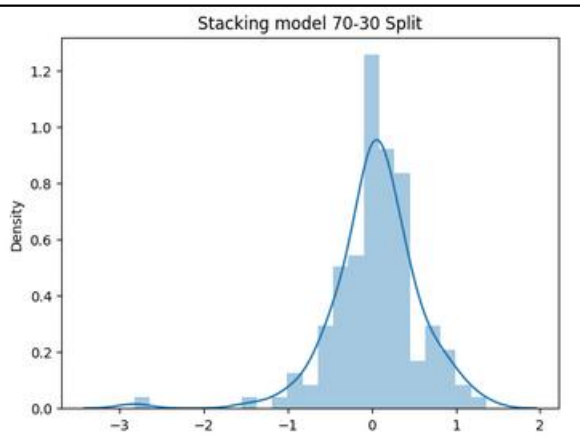




e) Stacking Regressor:







7. CODE :

Waiter_tip.py

```
# import dependencies.
import pandas as pd
import plotly.express as px
import seaborn as sns
import math
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
StackingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR as svr
from sklearn.metrics import (
    r2_score,
    mean_absolute_error as mae,
    mean_squared_error as mse,
)
import pickle
# load the dataset
df = pd.read_csv('tips.csv')

# EDA(exploratory data analysis).
print(df.head())
print(df.tail())
print(df.shape)
print(df.info())
print(df.isnull().sum())
print(df.describe())

scatter = sns.scatterplot(x="total_bill",y="tip",data = df,size="size")
plt.show()
# showing the four continous attribute through scatter plot and donut chart.
sex_scatter = px.scatter(data_frame = df, x = "total_bill", y = "tip", size = "size", color = "sex")
sex_pie = px.pie(df, values = "tip", names = "sex", hole = 0.6)
sex_scatter.show()
sex_pie.show()

smoker_scatter = px.scatter(data_frame = df, x = "total_bill", y = "tip", size = "size", color =
"smoker")
smoker_pie = px.pie(df, values = "tip", names = "smoker", hole = 0.6)
smoker_scatter.show()
smoker_pie.show()
```

```
day_scatter = px.scatter(data_frame = df, x = "total_bill", y = "tip", size = "size", color =
"day")
day_pie = px.pie(df, values = "tip", names = "day", hole = 0.6)
day_scatter.show()
day_pie.show()

time_scatter = px.scatter(data_frame = df, x = "total_bill", y = "tip", size = "size", color =
"time")
time_pie = px.pie(df, values = "tip", names = "time", hole = 0.6)
time_scatter.show()
time_pie.show()

# feature engineering
le = LabelEncoder()
# handling sex feature.
print(df.sex.unique())
df["sex"] = le.fit_transform(df["sex"])
# handling somking feature.
print(df.smoker.unique())
df["smoker"] = le.fit_transform(df["smoker"])
# handling day feature.
print(df.day.unique())
days = pd.get_dummies(df.day, drop_first = True)
df = pd.concat([df,days],axis=1)
df.drop(["day"],axis = 1, inplace = True)
# handling time feature.
print(df.time.unique())
df["time"] = le.fit_transform(df["time"])

# Correlation
cor_relation = df.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(cor_relation,annot = True,cmap="coolwarm")
plt.show()
# Creating dependent and Independent variables.
X = df.drop(['tip'], axis = 1)
Y = df["tip"]
xData = X.values
yData = Y.values
print(X.shape)
print(Y.shape)

# We will create a dictionary for each different models.
# Each dictionary will have keys for model, its prediction value,
# r2-score, mean squre error, root mean square error, and mean absolute error.
lr_dict = {
    "model":[],
    "prediction":[],
    "r2_score":[],
    "mse":[],
```

```
    "rmse":[],
    "mae":[]
}
rfr_dict = {
    "model":[],
    "prediction":[],
    "r2_score":[],
    "mse":[],
    "rmse":[],
    "mae":[]
}
svm_dict = {
    "model":[],
    "prediction":[],
    "r2_score":[],
    "mse":[],
    "rmse":[],
    "mae":[]
}
knn_dict = {
    "model":[],
    "prediction":[],
    "r2_score":[],
    "mse":[],
    "rmse":[],
    "mae":[]
}
stack_dict = {
    "model":[],
    "prediction":[],
    "r2_score":[],
    "mse":[],
    "rmse":[],
    "mae":[]
}
# we have to store our testing set of each variation to compare with predictions of the
models.
# so we will create a list to store this.
yTest_list = []

#Train and test data on different variation
# varition 1 : 80-20 split
# varition 2 : 75-25 split
# varition 3 : 70-30 split
for i in range(3):
    # spliting data into training and testing columns.
    xTrain, xTest, yTrain, yTest = train_test_split(xData, yData, test_size = 0.20 +
(i*0.05),random_state = 31)
    # storing yTest in the list (yTest_list).
    yTest_list.append(yTest)
```

```
# making use of Linear regression model.
lr = LinearRegression()
lr.fit(xTrain, yTrain)
lr_dict["prediction"].append(lr.predict(xTest))
lr_dict["model"].append(lr)
# storing r2-score,mse,rmse and mae.
lr_dict["r2_score"].append(r2_score(yTest,lr_dict["prediction"][i]))
lr_dict["mse"].append(mse(yTest,lr_dict["prediction"][i]))
lr_dict["mae"].append(mae(yTest,lr_dict["prediction"][i]))
# since, rmse is simply the square root of mse. so, we will just find square root
# of mse using math function.
lr_dict["rmse"].append(math.sqrt(mse(yTest_list[i],lr_dict["prediction"][i])))

# making use of Random Forest regressor model.
rfr = RandomForestRegressor()
rfr.fit(xTrain, yTrain)
rfr_dict["prediction"].append(rfr.predict(xTest))
rfr_dict["model"].append(rfr)
# storing r2-score,mse,rmse and mae.
rfr_dict["r2_score"].append(r2_score(yTest,rfr_dict["prediction"][i]))
rfr_dict["mse"].append(mse(yTest,rfr_dict["prediction"][i]))
rfr_dict["mae"].append(mae(yTest,rfr_dict["prediction"][i]))
# since, rmse is simply the square root of mse. so, we will just find square root
# of mse using math function.
rfr_dict["rmse"].append(math.sqrt(mse(yTest,rfr_dict["prediction"][i])))

# making use of SVM model.
svm = svr(kernel="linear",C=1,gamma="auto")
svm.fit(xTrain, yTrain)
svm_dict["prediction"].append(svm.predict(xTest))
svm_dict["model"].append(svm)
# storing r2-score,mse,rmse and mae.
svm_dict["r2_score"].append(r2_score(yTest,svm_dict["prediction"][i]))
svm_dict["mse"].append(mse(yTest,svm_dict["prediction"][i]))
svm_dict["mae"].append(mae(yTest,svm_dict["prediction"][i]))
# since, rmse is simply the square root of mse. so, we will just find square root
# of mse using math function.
svm_dict["rmse"].append(math.sqrt(mse(yTest,svm_dict["prediction"][i])))

# making use of K-Nearest Neighbour(KNN) model.
knn = KNeighborsRegressor(n_neighbors=1)
knn.fit(xTrain, yTrain)
knn_dict["prediction"].append(knn.predict(xTest))
knn_dict["model"].append(knn)
# storing r2-score,mse,rmse and mae.
knn_dict["r2_score"].append(r2_score(yTest,knn_dict["prediction"][i]))
knn_dict["mse"].append(mse(yTest,knn_dict["prediction"][i]))
knn_dict["mae"].append(mae(yTest,knn_dict["prediction"][i]))
# since, rmse is simply the square root of mse. so, we will just find square root
```

```
    # of mse using math function.
    knn_dict["rmse"].append(math.sqrt(mse(yTest,knn_dict["prediction"][i])))

#Creating Base model for stacking
base_models = [
    ('rf', RandomForestRegressor()),
    ('gb', GradientBoostingRegressor())
]
# Creating meta model for stacking
meta_model = LinearRegression()
# Create the stacking regressor
stacking_model = StackingRegressor(estimators=base_models,
final_estimator=meta_model)
for i in range(3):
    # spliting data into training and testing columns.
    xTrain, xTest, yTrain, yTest = train_test_split(xData, yData, test_size = 0.20 +
(i*0.05),random_state = 31)

    # making use of Stacking model.
    stacking_model.fit(xTrain, yTrain)
    stack_dict["prediction"].append(stacking_model.predict(xTest))
    stack_dict["model"].append(stacking_model)
    # storing r2-score,mse,rmse and mae.
    stack_dict["r2_score"].append(r2_score(yTest,stack_dict["prediction"][i]))
    stack_dict["mse"].append(mse(yTest,stack_dict["prediction"][i]))
    stack_dict["mae"].append(mae(yTest,stack_dict["prediction"][i]))
    # since, rmse is simply the square root of mse. so, we will just find square root
    # of mse using math function.
    stack_dict["rmse"].append(math.sqrt(mse(yTest,stack_dict["prediction"][i])))

# Evaluation of the models.

# printing Evaluation report of Linear regression (r2-score, mean_square_error,
# root_mean_sqare_error and mean_absolute_error.)
for i in range(3):
    print(f"\nTrain Test split at {80-(i*5)}-{20+(i*5)}")
    print("r2 score of Linear regression model              :",lr_dict["r2_score"][i])
    print("mean square error of Linear regression model     :",lr_dict["mse"][i])
    print("Root mean square error of Linear regression model:",lr_dict["rmse"][i])
    print("mean absolute error of Linear regression model   :",lr_dict["mae"][i])
# printing Evaluation report of Random Forest Regressor (r2-score, mean_square_error,
# root_mean_sqare_error and mean_absolute_error.)
for i in range(3):
    print(f"\nTrain Test split at {80-(i*5)}-{20+(i*5)}")
    print("r2 score of Random Forest model              :",rfr_dict["r2_score"][i])
    print("mean square error of Random Forest model     :",rfr_dict["mse"][i])
    print("Root mean square error of Random Forest model:",rfr_dict["rmse"][i])
    print("mean absolute error of Random Forest model   :",rfr_dict["mae"][i])
# printing Evaluation report of SVM (r2-score, mean_square_error,
# root_mean_sqare_error and mean_absolute_error.)
```

```
for i in range(3):
    print(f"\nTrain Test split at {80-(i*5)}-{20+(i*5)}")
    print("r2 score of SVM model             :",svm_dict["r2_score"][i])
    print("mean square error of SVM model    :",svm_dict["mse"][i])
    print("Root mean square error of SVM model:",svm_dict["rmse"][i])
    print("mean absolute error of SVM model  :",svm_dict["mae"][i])
# printing Evaluation report of K-Nearest Neighbour(KNN) (r2-score, mean_square_error,
# root_mean_sqare_error and mean_absolute_error.)
for i in range(3):
    print(f"\nTrain Test split at {80-(i*5)}-{20+(i*5)}")
    print("r2 score of KNN model             :",knn_dict["r2_score"][i])
    print("mean square error of KNN model    :",knn_dict["mse"][i])
    print("Root mean square error of KNN model:",knn_dict["rmse"][i])
    print("mean absolute error of KNN model  :",knn_dict["mae"][i])
# printing Evaluation report of Stacking model (r2-score, mean_square_error,
# root_mean_sqare_error and mean_absolute_error.)
for i in range(3):
    print(f"\nTrain Test split at {80-(i*5)}-{20+(i*5)}")
    print("r2 score of stack model           :",stack_dict["r2_score"][i])
    print("mean square error of stack model  :",stack_dict["mse"][i])
    print("Root mean square error of stack model:",stack_dict["rmse"][i])
    print("mean absolute error of stack model  :",stack_dict["mae"][i])
#comparing all variatins of different models through bar plot
for i in range(3):
    list1=['linearRegression','RandomForest','SVM','KNN','stack']
    list2=[lr_dict["r2_score"][i],rfr_dict["r2_score"][i],
           svm_dict["r2_score"][i],knn_dict["r2_score"][i],stack_dict["r2_score"][i]]
    df_Accuracy=pd.DataFrame({"Method Used":list1,"Accuracy":list2})
    chart=sns.barplot(x='Method Used',y='Accuracy',data=df_Accuracy,width = 0.5)
    # title vary depending on variation.
    plt.title(f"Split {80-(i*5)}-{20+(i*5)}")
    plt.show()
# Showing The variance of different through distplot.
# Linear Regression
for i in range(3):
    plt.title(f"Linear regression {80-(i*5)}-{20+(i*5)} Split")
    sns.distplot(yTest_list[i] - lr_dict["prediction"][i])
    plt.show()
# Random Forest.
for i in range(3):
    plt.title(f"Random Rorest {80-(i*5)}-{20+(i*5)} Split")
    sns.distplot(yTest_list[i] - rfr_dict["prediction"][i])
    plt.show()
# SVM.
for i in range(3):
    plt.title(f"SVM {80-(i*5)}-{20+(i*5)} Split")
    sns.distplot(yTest_list[i] - svm_dict["prediction"][i])
    plt.show()
# KNN
for i in range(3):
```

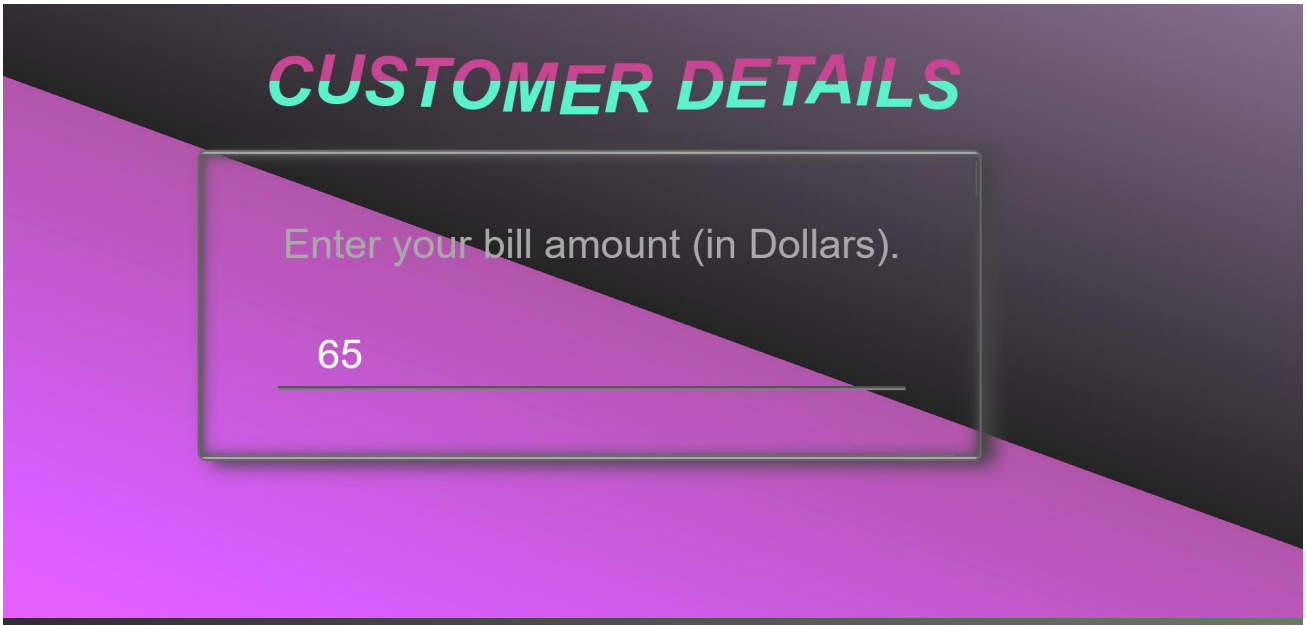
```
    plt.title(f"KNN {80-(i*5)}-{20+(i*5)} Split")
    sns.distplot(yTest_list[i] - knn_dict["prediction"][i])
    plt.show()
# Stacking.
for i in range(3):
    plt.title(f"Stacking model {80-(i*5)}-{20+(i*5)} Split")
    sns.distplot(yTest_list[i] - knn_dict["prediction"][i])
    plt.show()

# Dumping the models in the pickle file

with open('lr_model.pkl', 'wb') as f:
    pickle.dump(lr_dict["model"], f)
with open('rfr_model.pkl', 'wb') as f:
    pickle.dump(rfr_dict["model"], f)
with open('svm_model.pkl', 'wb') as f:
    pickle.dump(svm_dict["model"], f)
with open('knn_model.pkl', 'wb') as f:
    pickle.dump(knn_dict["model"], f)
with open('stack_model.pkl', 'wb') as f:
    pickle.dump(stack_dict["model"], f)
```

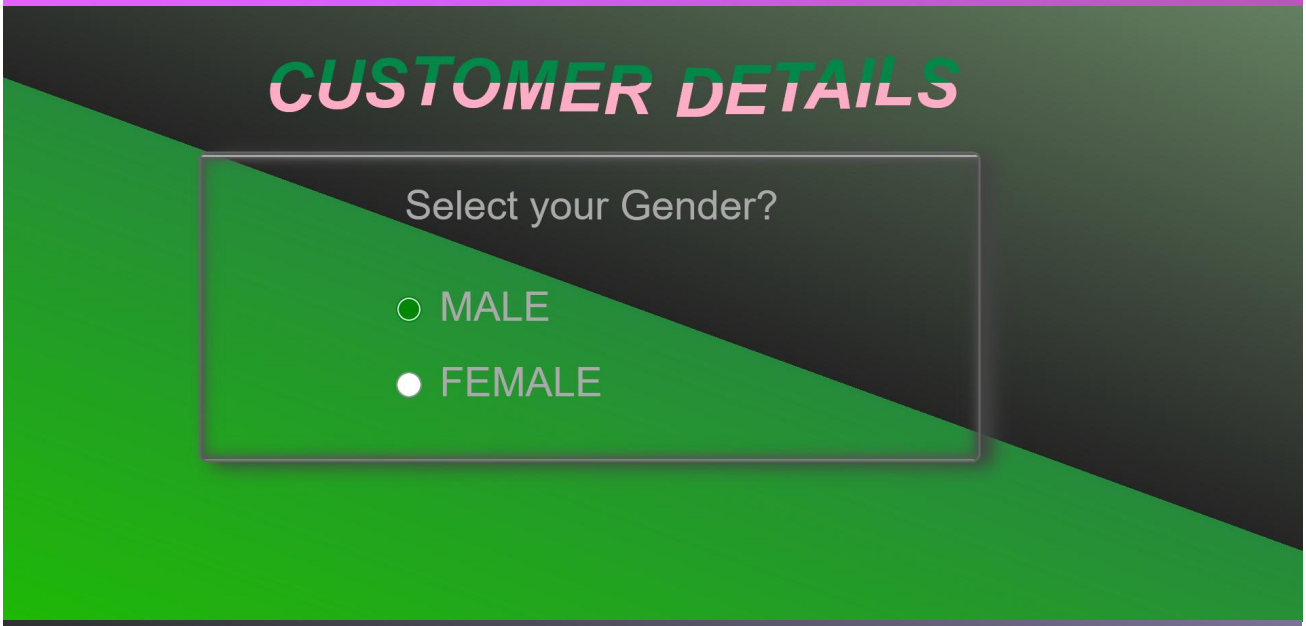
App.py

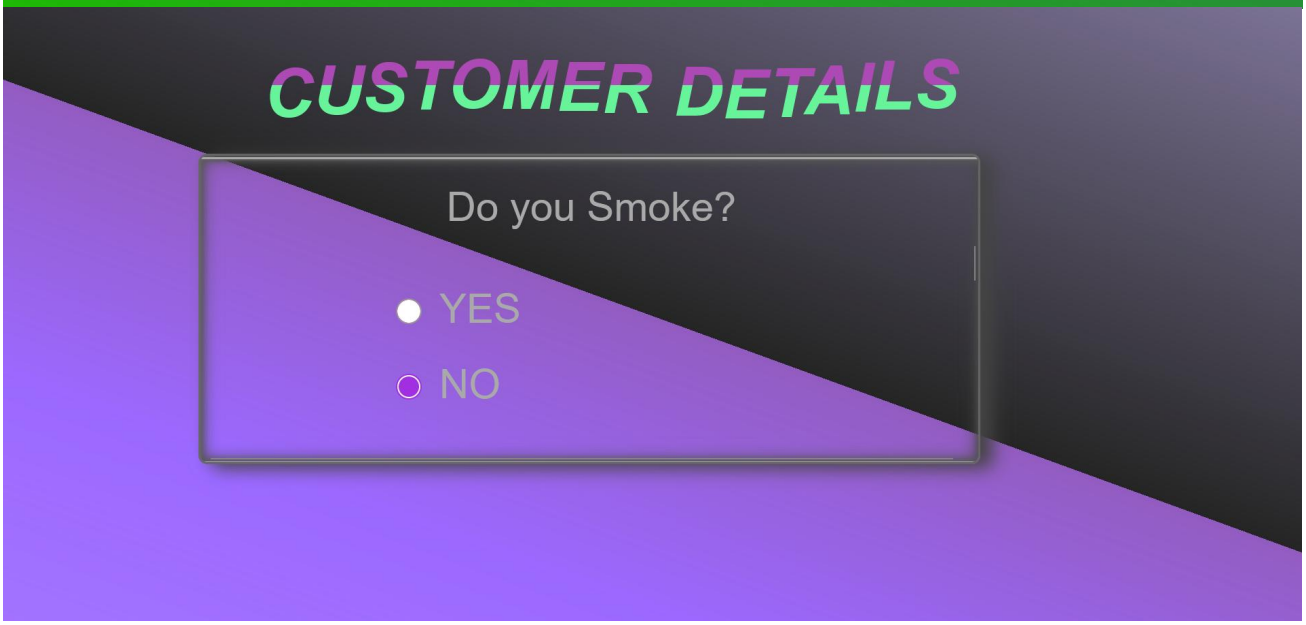
```
# import dependencies.
import pickle
from flask import Flask,render_template,request
import numpy as np
app = Flask(__name__)

# loading all the models with different variation from pikle file.
with open('lr_model.pkl', 'rb') as f:
    lr_model = pickle.load(f)
with open('rfr_model.pkl', 'rb') as f:
    rfr_model = pickle.load(f)
with open('svm_model.pkl', 'rb') as f:
    svm_model = pickle.load(f)
with open('knn_model.pkl', 'rb') as f:
    knn_model = pickle.load(f)
with open('stack_model.pkl', 'rb') as f:
    stack_model = pickle.load(f)

# starting main page.
@app.route('/')
def main():
    return render_template('index.html')

# getting input from user and predicting output using
# model and variation specified.
@app.route('/get_tip',methods=['POST'])
def predict():
```

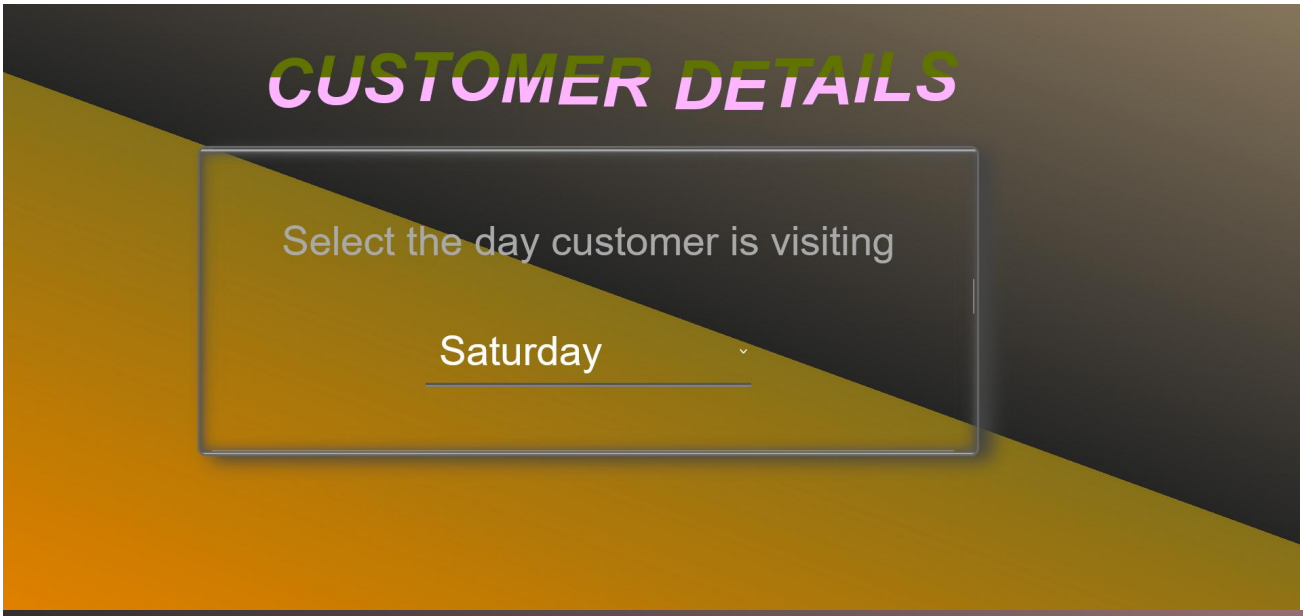
```
    bill = float(request.form['bill'])
    sex = int(request.form['sex'])
    smoker = int(request.form['smoker'])
    time = int(request.form['time'])
    size = int(request.form['size'])
    day = [False,False,False]
    get_day = int(request.form['day'])

    if (get_day == 1):
        day[0] = True
    elif (get_day == 2):
        day[1] = True
    elif (get_day == 3):
        day[2] = True

    values = [bill,sex,smoker,time,size,day[0],day[1],day[2]]

    final = np.array([values])

    # model and variations.
    model = str(request.form['model'])
    variation = int(request.form['variation'])

    # predict_tip = []
    if model == 'linear regression':
        predict_tip = lr_model[variation].predict(final)
    elif model == 'random forest':
        predict_tip = rfr_model[variation].predict(final)
    elif model == 'svm':
        predict_tip = svm_model[variation].predict(final)
    elif model == 'knn':
        predict_tip = knn_model[variation].predict(final)
    elif model == 'stacking model':
        predict_tip = stack_model[variation].predict(final)

    return render_template('get_tip.html',predict=round(predict_tip[0],2))

if __name__ == '__main__':
    # running app on port 8000
    app.run(debug=True,port=8000)
```

USER INTERFACE:

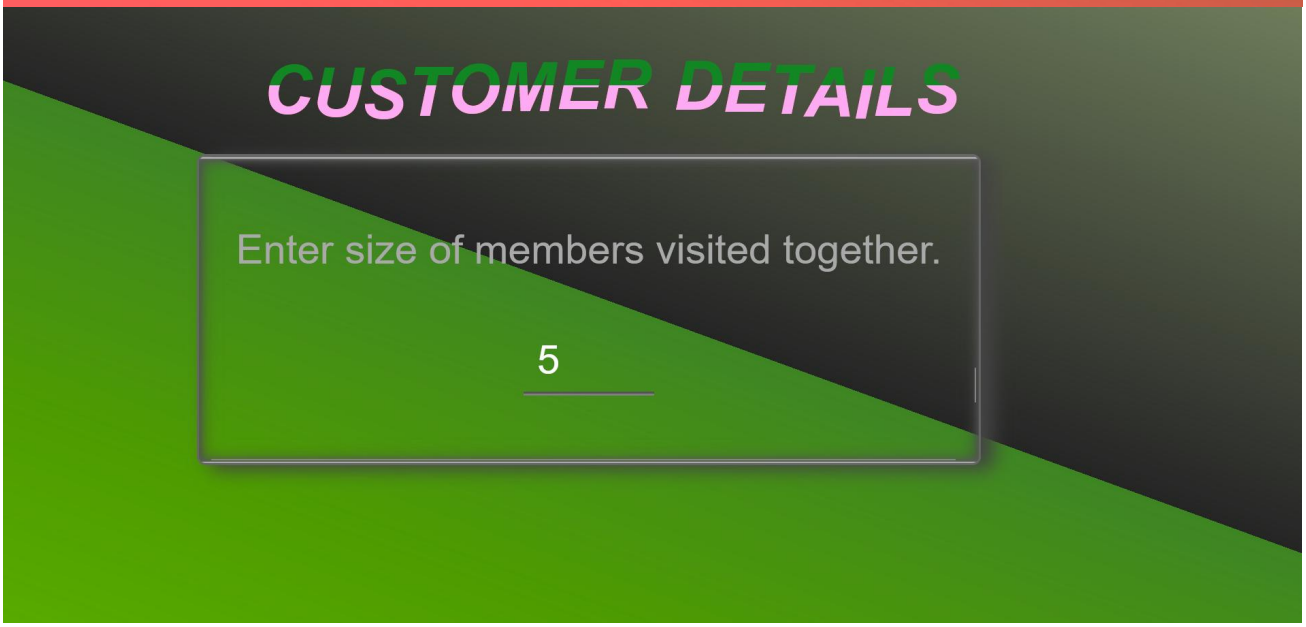

CUSTOMER DETAILS
Enter your bill amount (in Dollars).
65

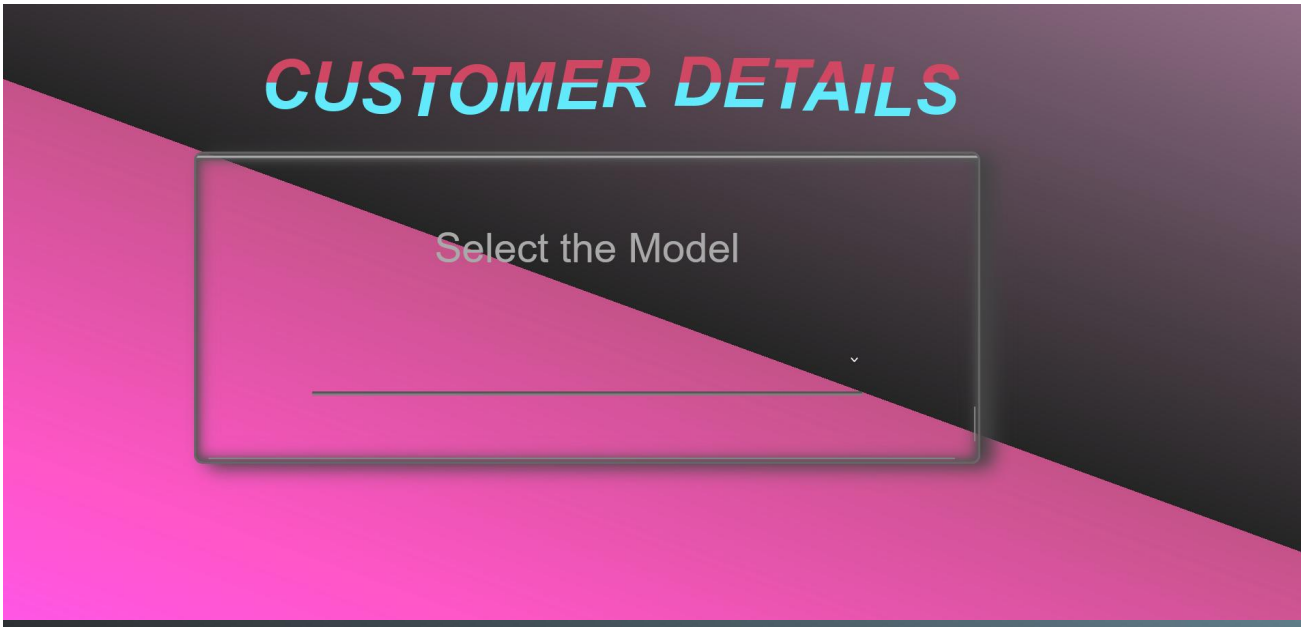

CUSTOMER DETAILS
Select your Gender?
MALE
FEMALE

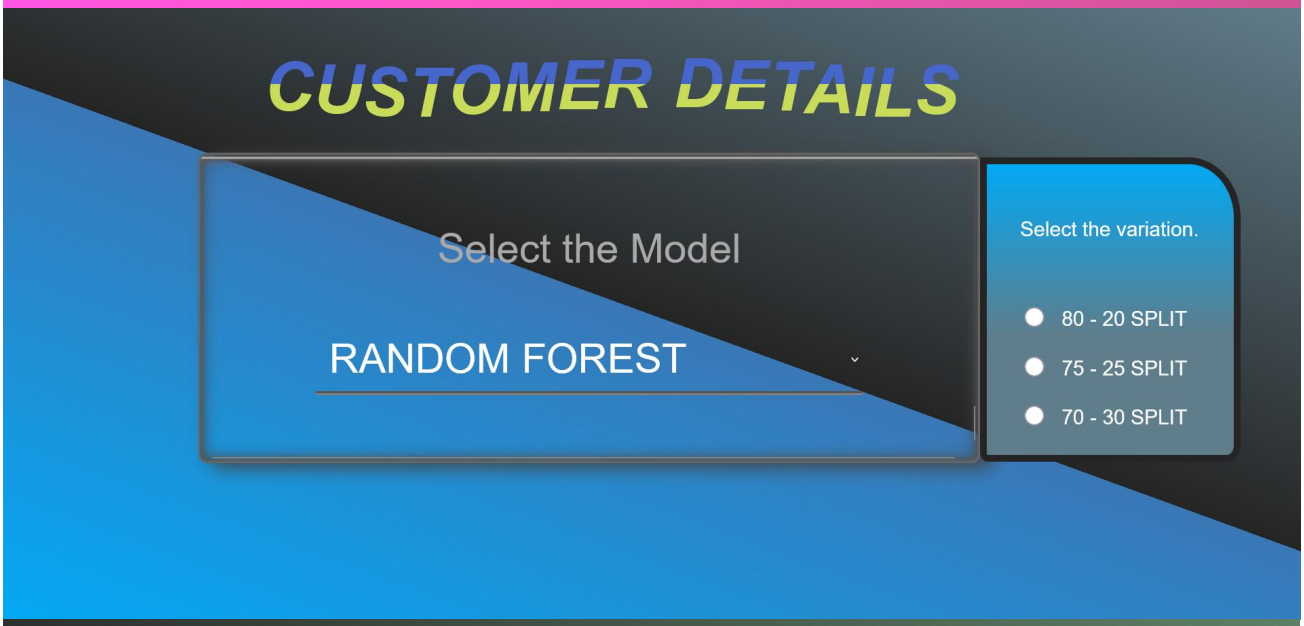

CUSTOMER DETAILS
Do you Smoke?
YES
NO

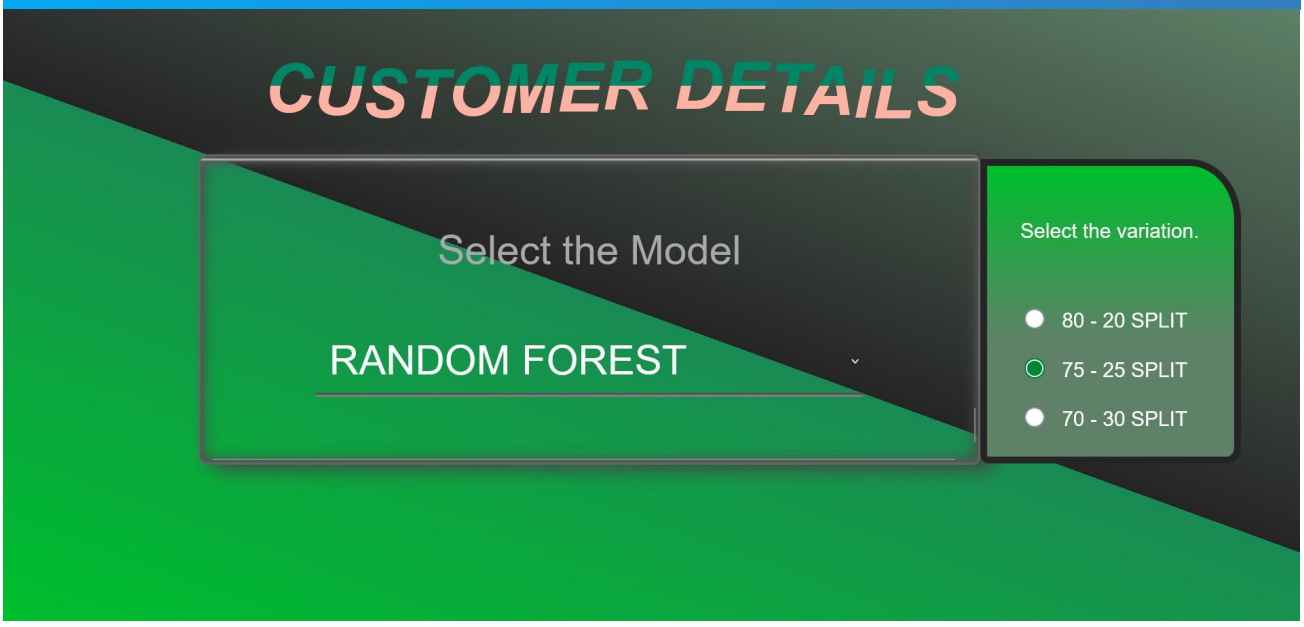

CUSTOMER DETAILS
Select the day customer is visiting
Saturday

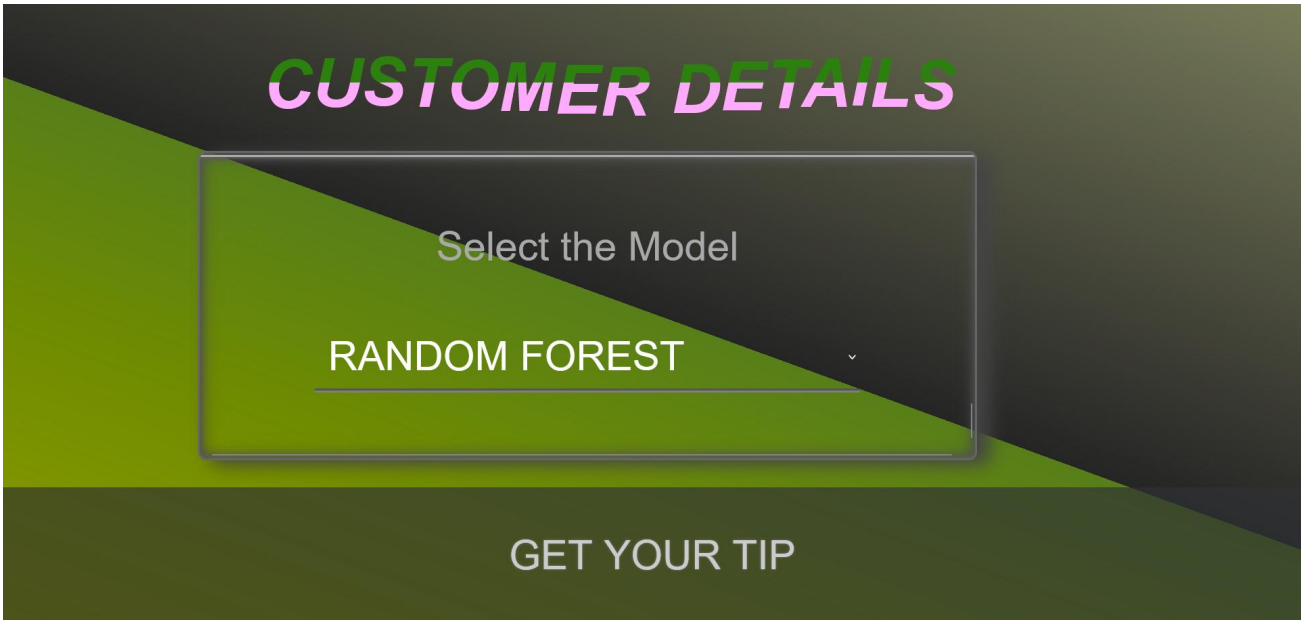

CUSTOMER DETAILS
Select the time (Lunch / Dinner)?
LUNCH
DINNER


CUSTOMER DETAILS
Enter size of members visited together.
5

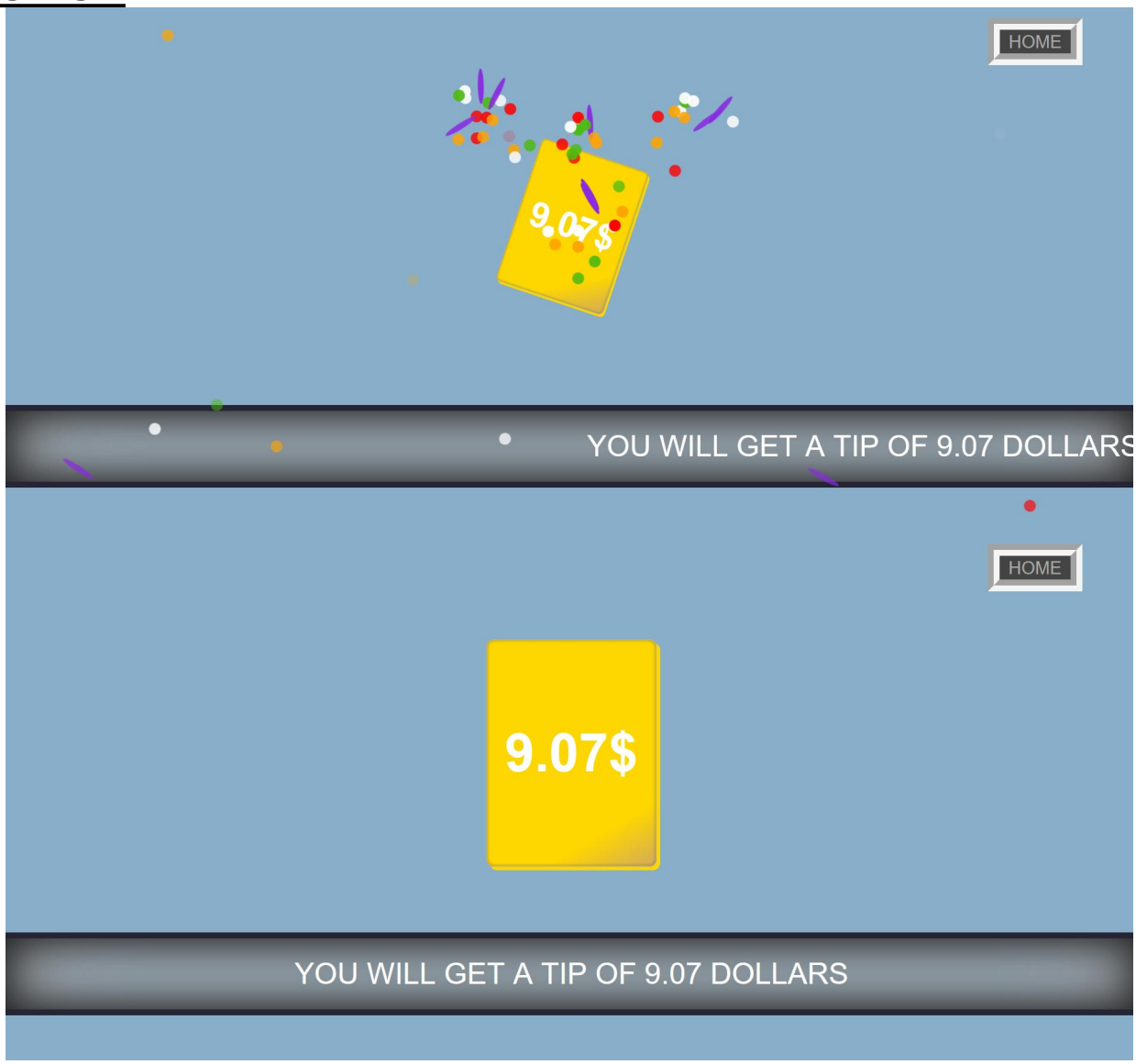

CUSTOMER DETAILS
Select the Model


CUSTOMER DETAILS
Select the Model
RANDOM FOREST
Select the variation.
80 - 20 SPLIT
75 - 25 SPLIT
70 - 30 SPLIT


CUSTOMER DETAILS
Select the Model
RANDOM FOREST
Select the variation.
80 - 20 SPLIT
75 - 25 SPLIT
70 - 30 SPLIT


CUSTOMER DETAILS
Select the Model
RANDOM FOREST
GET YOUR TIP

OUTPUT:


HOME
9.07$
YOU WILL GET A TIP OF 9.07 DOLLARS
HOME
9.07$
YOU WILL GET A TIP OF 9.07 DOLLARS

8. FEATURE ENCHANCEMENT :

1. **Personalized Customer Profiles:**

- Develop a system that creates personalized profiles for regular customers based on their preferences, ordering history, and tipping patterns. This can help waitstaff provide more tailored service, leading to higher tips.

2 **Integration with Reservation Systems:**

- Integrate with reservation systems to access information about special occasions (birthdays, anniversaries) or dietary preferences. This data can be used to provide a more personalized and memorable dining experience, increasing the likelihood of a generous tip.

3 **Customer Engagement Surveys:**

- Conduct periodic customer satisfaction surveys to gather feedback on the dining experience. Use this data to identify areas for improvement and adjust service strategies accordingly.

4 **Mobile App Integration:**

- Develop a mobile app that allows customers to provide instant feedback and rate their dining experience. This data can be used to continuously improve service quality.