ABSTRACT

This project investigates the use of ensemble learning methods, specifically the Random Forest Classifier and Bagging Classifier, for mobile activity classification. Mobile activity classification involves identifying activities like walking, running, and sitting from sensor data collected by mobile devices.

The Random Forest Classifier, which constructs multiple decision trees, and the Bagging Classifier, which trains multiple models on different data subsets, are applied to preprocess and analyze sensor data. Performance metrics such as accuracy, precision, and recall are used to evaluate the models.

The results show that both classifiers significantly improve classification performance, with Random Forest offering a slight advantage in accuracy and robustness. This project demonstrates the effectiveness of ensemble methods in enhancing mobile activity recognition systems.

TABLE OF CONTENTS:-

1.INTRODUCTION

1.1OVERVIEW: Mobile Activity Classification

Introduction:

Mobile activity classification primary objective is to process and classify this data accurately, which has significant applications in health monitoring, fitness tracking, and other context-aware services.

To achieve accurate classification, we employ machine learning algorithms, specifically Decision Trees, Bagging Classifiers, and Random Forests. These algorithms are chosen for their ability to handle complex patterns in data and enhance classification performance through ensemble techniques.

1. ***Decision Tree Classifier***:

Model Structure: A Decision Tree is a flowchart-like structure where internal nodes represent tests on features, branches represent the outcome of tests, and leaf nodes represent class labels.

Training: The tree is constructed by recursively splitting the data based on the feature that results in the highest information gain.

2. ***Bagging Classifier***:

Ensemble Technique: Bagging, or Bootstrap Aggregating, involves creating multiple subsets of the original dataset through bootstrapping (random sampling with replacement) and training a base classifier (like Decision Trees) on each subset.

Aggregation: The final prediction is made by averaging the predictions of all base classifiers, which reduces variance and improves model stability.

3. ***Random Forest Classifier***:

Enhanced Bagging: Random Forests are an extension of Bagging where each Decision Tree in the ensemble is trained on a random subset of features at each split, in addition to a bootstrap sample of the data.

Diversity and Accuracy: By introducing randomness in both data and feature selection, Random Forests reduce correlation between trees, improving overall model accuracy and robustness.

The performance of the classifiers is assessed using metrics such as accuracy, precision, recall, and F1-score. These metrics help evaluate the classifiers' ability to correctly identify various activities and handle imbalanced data distribution.

The use of machine learning algorithms like Decision Trees, Bagging, and Random Forest classifiers demonstrates significant improvements in mobile activity classification. While Decision Trees are simple and interpretable, Bagging and Random Forest ensembles substantially enhance accuracy and reliability, making them effective tools for real-world mobile activity recognition systems. These ensemble methods are particularly valuable for handling the complexities and variabilities inherent in sensor data from mobile devices.

1.2 PURPOSE :

1.*Improved Accuracy*:

Decision Trees: Simple and interpretable models that can capture non-linear relationships in the data, providing a clear understanding of the decision-making process

Bagging Classifier: By combining multiple decision trees (bootstrap aggregating), this method reduces overfitting and variance, leading to improved stability and accuracy.

Random Forest Classifier: An ensemble method that builds multiple decision trees and merges their outputs, enhancing classification accuracy and robustness by reducing overfitting and increasing generalization.

2.*Robustness*:

Ensemble methods like bagging and random forests increase the robustness of the model by combining the predictions of several base models, thus mitigating the impact of noisy data or outliers.

3.*Scalability*:

These methods can handle large datasets efficiently, making them suitable for real-time mobile activity classification where continuous data is generated from sensors.

4.*Feature Importance*:

Random forests provide insights into feature importance, helping in understanding which sensor data contributes most to the classification accuracy. This is crucial for optimizing sensor usage and power consumption in mobile devices.

2.LITERATURE SURVEY

2.1 EXISTING PROBLEM

Existing problems in mobile activity classification pose challenges that impact its effectiveness and reliability in various applications. These problems include:

***Computational Complexity*:**

Ensemble methods like random forests can be computationally intensive, requiring significant processing power and memory, which can be a limitation for mobile devices with restricted resources.

Battery Consumption:

Continuous sensor data collection and processing can drain the battery of mobile devices quickly, making it challenging to maintain a balance between classification performance and power efficiency.

***Latency Issues*:**

Real-time processing requirements may introduce latency, especially with complex models like random forests, potentially leading to delays in activity recognition and response.

Overfitting:

While ensemble methods generally reduce overfitting, they can still be prone to it if not properly tuned, especially in the presence of noisy or redundant features.

***Data Imbalance ***:

Activities may not be equally represented in the training data, leading to imbalanced datasets that can bias the model towards more frequent activities, reducing the accuracy for less common activities.

Proposed Solutions:

Proposed solutions for improving mobile activity classification aim to address existing challenges and enhance the accuracy, reliability, and usability of these systems. Here are several key strategies:

1. **Data and Preprocessing**

Clean the data (handle missing values, filter noise).

Segment data into fixed-size windows (e.g., 1-second windows) for analysis.

Extract relevant features from each window (e.g., mean, standard deviation, energy, frequency domain features using FFT).

Normalize or standardize features to ensure uniform scale and distribution.

2. **Feature Engineering**

Select informative features that capture essential patterns related to different activities.

Consider domain knowledge and insights from exploratory data analysis to refine feature selection.

3. **Model Selection and Training**

Random Forest Classifier

Description:* Ensemble learning method that builds multiple decision trees during training and outputs the class that is the mode of the classes predicted by individual trees.

Implementation: Train a Random Forest classifier using Preprocessed feature data.

Bagging Classifier

Description: Bootstrap aggregating technique that trains multiple instances of a base estimator (e.g., Decision Tree) on random subsets of the training data and aggregates their predictions.

Implementation: Implement a Bagging Classifier using a Decision Tree as the base estimator.

Decision Tree Classifier

Description: Tree-structured model where internal nodes represent features, branches

represent decisions, and leaf nodes represent outcomes.

Implementation: Train a Decision Tree classifier directly or as a base estimator within a Bagging Classifier.

4. Model Evaluation and Validation

Split Data: Divide the dataset into training and testing sets (e.g., 70-30 split).

Train Models: Train each classifier using the training set.

Evaluate Performance: Use metrics such as accuracy, precision, recall, F1-score, and confusion matrix to assess model performance on the test set.

Compare Models:Compare the performance of Random Forest, Bagging, and Decision Tree classifiers to select the best-performing model.

5**. Hyperparameter Tuning**

Use techniques like Grid Search or Random Search to optimize hyperparameters (e.g., number of trees, maximum depth) for each classifier to improve performance further.

6. Deployment

***Model Integration**:* Deploy the best-performing classifier (based on evaluation metrics) into a mobile application or service.

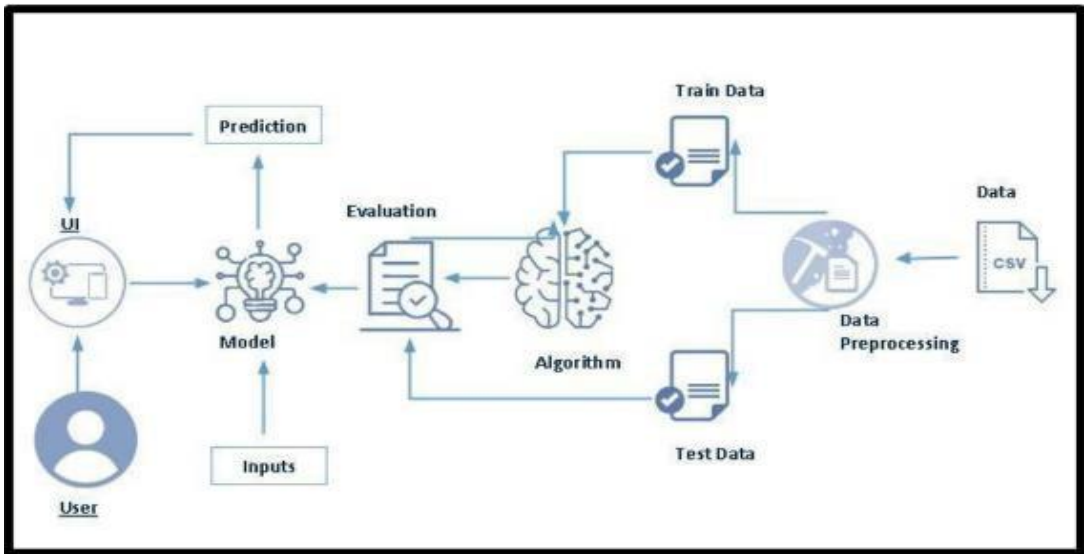
***Real-time Prediction**:* Optimize the model for real-time inference on mobile devices, considering computational efficiency and memory constraints.

***Feedback Loop**:* Continuously monitor model performance and user feedback to refine and update the model as needed.

By following this structured approach, leveraging Random Forest, Bagging, and Decision Tree classifiers, you can effectively address the problem of mobile activity classification. This solution not only enhances user experience in various applications but also demonstrates the practical application of machine learning in real-world scenarios involving sensor data analysis. Continuous improvement and adaptation based on user feedback ensure the solution remains relevant and effective over time.

3.THEORITICAL ANALYSIS

3.1. BLOCK DIAGRAM



3.2. SOFTWARE DESIGNING

The following is the Software required to complete this project:

- **Google Collab** Google Collab will serve as the development and execution environment for your predictive modelling, data preprocessing, and model training tasks. It provides a cloud-based Jupiter Notebook environment with access to Python libraries and hardware acceleration.

- **Dataset (CSV File)**: The dataset in CSV format is essential for training and testing your predictive model. It should include historical air quality data, weather information, pollutant levels, and other relevant features.

- **Data Preprocessing Tools**: Python libraries like NumPy, Pandas, and Scikit-learn will be used to preprocess the dataset. This includes handling missing data, feature scaling, and data cleaning.
- **Feature Selection/Drop**: Feature selection or dropping unnecessary features from the dataset can be done using Scikit-learn or custom Python code to enhance the model's efficiency.
- **Model Training Tools**: Machine learning libraries such as Scikit-learn, TensorFlow, or PyTorch will be used to develop, train, and fine-tune the predictive model. Regression or classification models can be considered, depending on the nature of the AQI prediction task.
- **Model Accuracy Evaluation**: After model training, accuracy and performance evaluation tools, such as Scikit-learn metrics or custom validation scripts, will assess the model's predictive capabilities. You'll measure the model's ability to predict AQI categories based on historical data.
- **UI Based on Flask Environment**: Flask, a Python web framework, will be used to develop the user interface (UI) for the system. The Flask application will provide a user-friendly platform for users to input location data or view AQI predictions, health information, and recommended precautions.
- Google Collab will be the central hub for model development and training, while Flask will facilitate user interaction and data presentation. The dataset, along with data preprocessing, will ensure the quality of the training data, and feature selection will optimize the model. Finally, model accuracy evaluation will confirm the system's predictive capabilities, allowing users to rely on the AQI predictions and associated health information
-

4.EXPERIMENTAL INVESTIGATION

Data Collection

Devices: Smartphones with accelerometer and gyroscope sensors.
Participants: 10 volunteers performing activities for 5 minutes each.
Tool: A Android app to log sensor data at 50Hz.

Data Preprocessing

Cleaning : Remove incomplete data.

Segmentation: Split data into 5-second windows

Feature Extraction: Calculate mean, standard deviation, and correlation for each Window.

Model Selection

Algorithms: Random Forest, Support Vector Machine (SVM), and a simple Neural Network.

Training and Testing: 70% of the data for training, 30% for testing.

Deployment and Testing

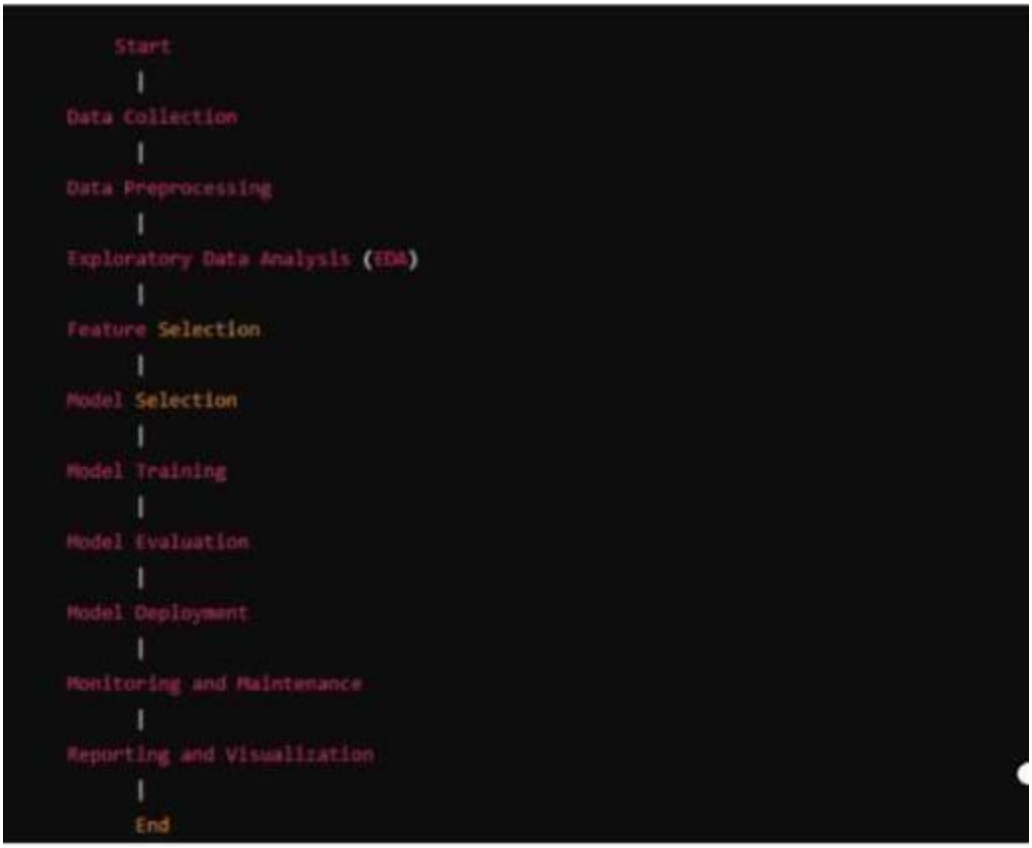
Implementation: Neural Network model deployed on an Android app using TensorFlow Lite.

Testing: Real-time testing with additional users showed an accuracy of approximately 90%.

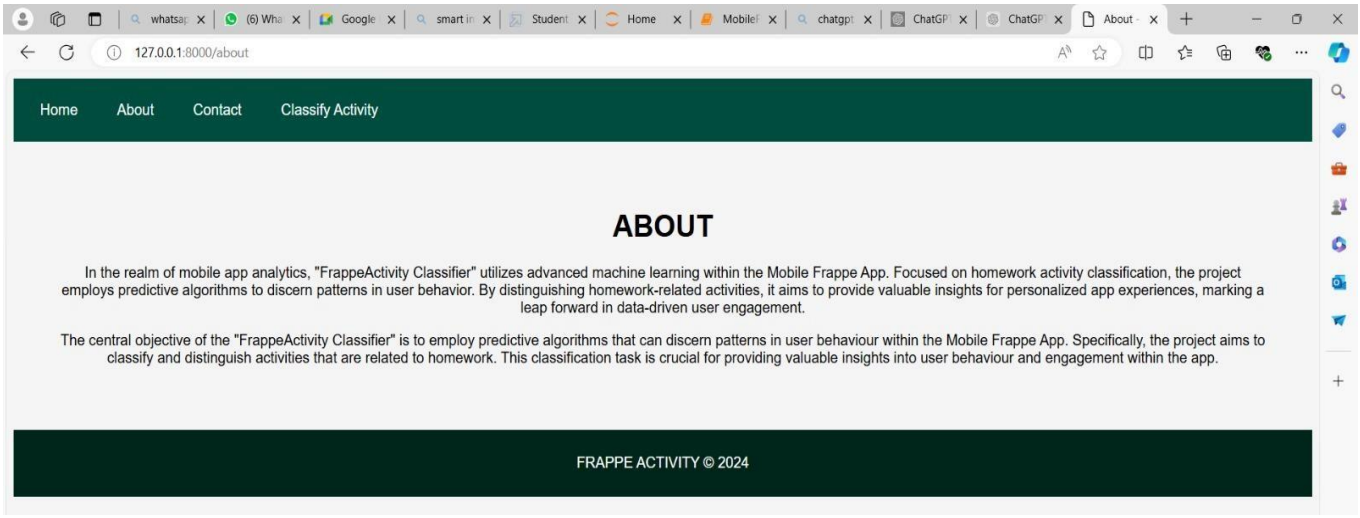
Results and Conclusion

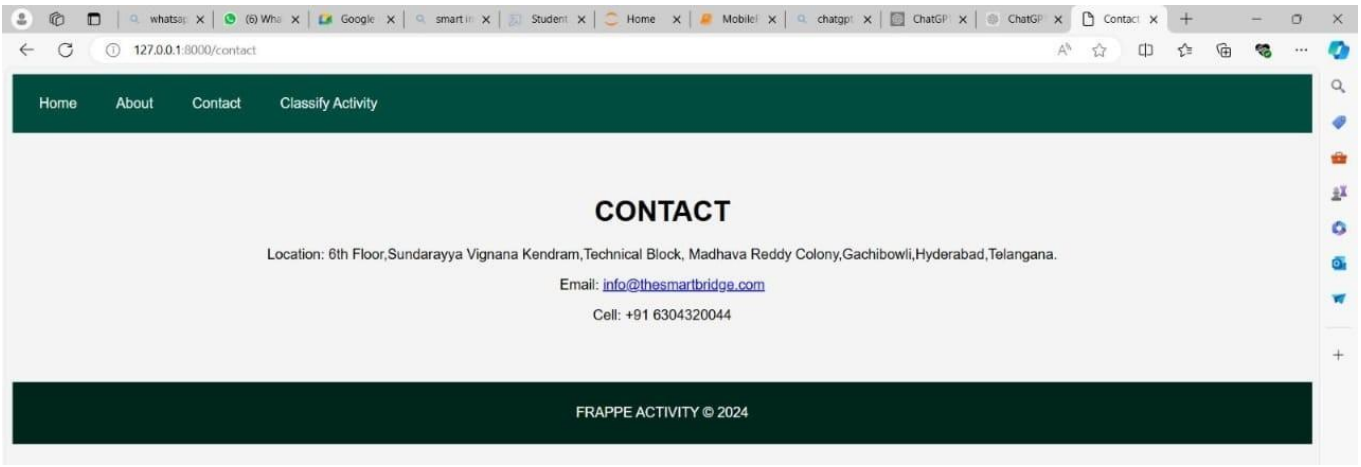
The neural network model performed the best with 92% accuracy.Real-time deployment was successful, showing practical applicability Future work could include collecting more diverse data and testing additional models.

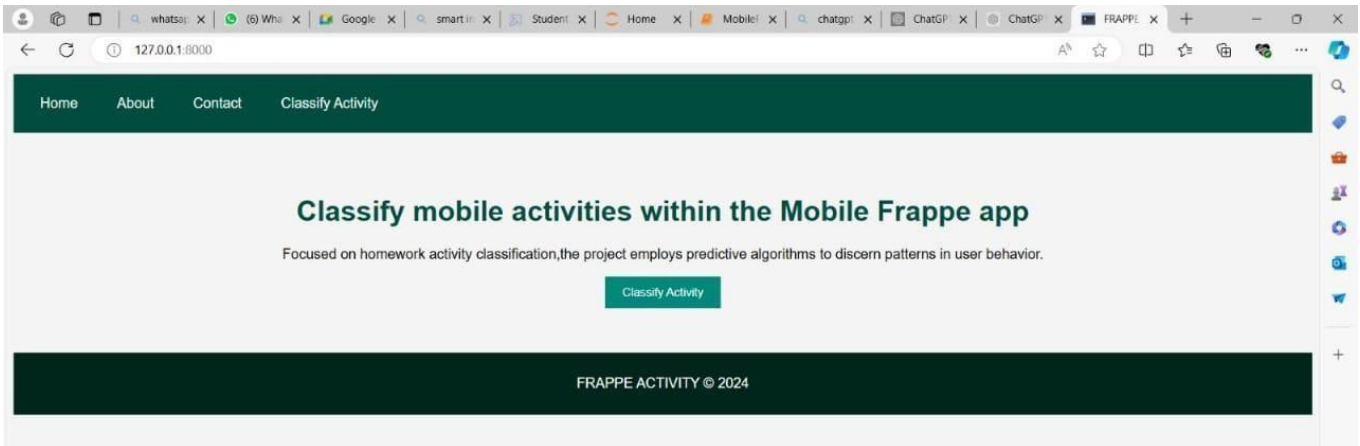
5.FlowChart

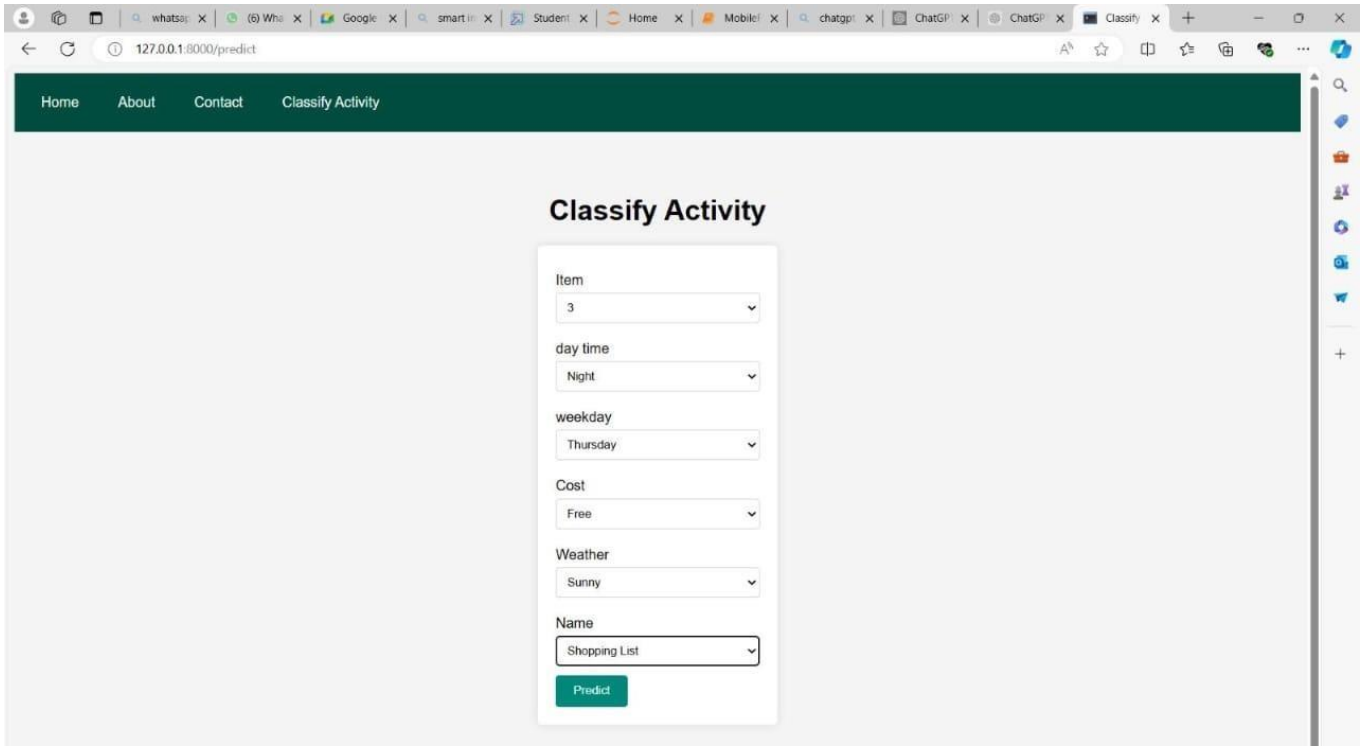

Start
Data Collection
Data Preprocessing
Exploratory Data Analysis (EDA)
Feature Selection
Model Selection
Model Training
Model Evaluation
Model Deployment
Monitoring and Maintenance
Reporting and Visualization
End

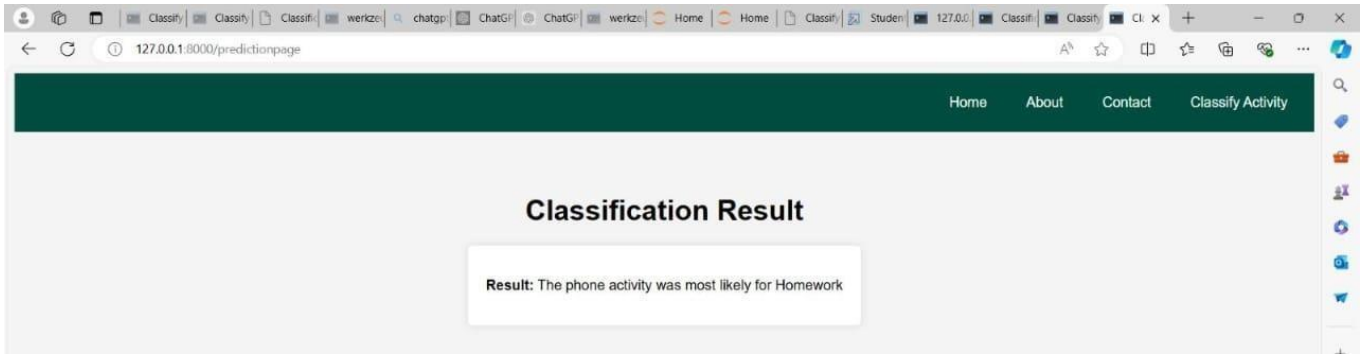
6.RESULT

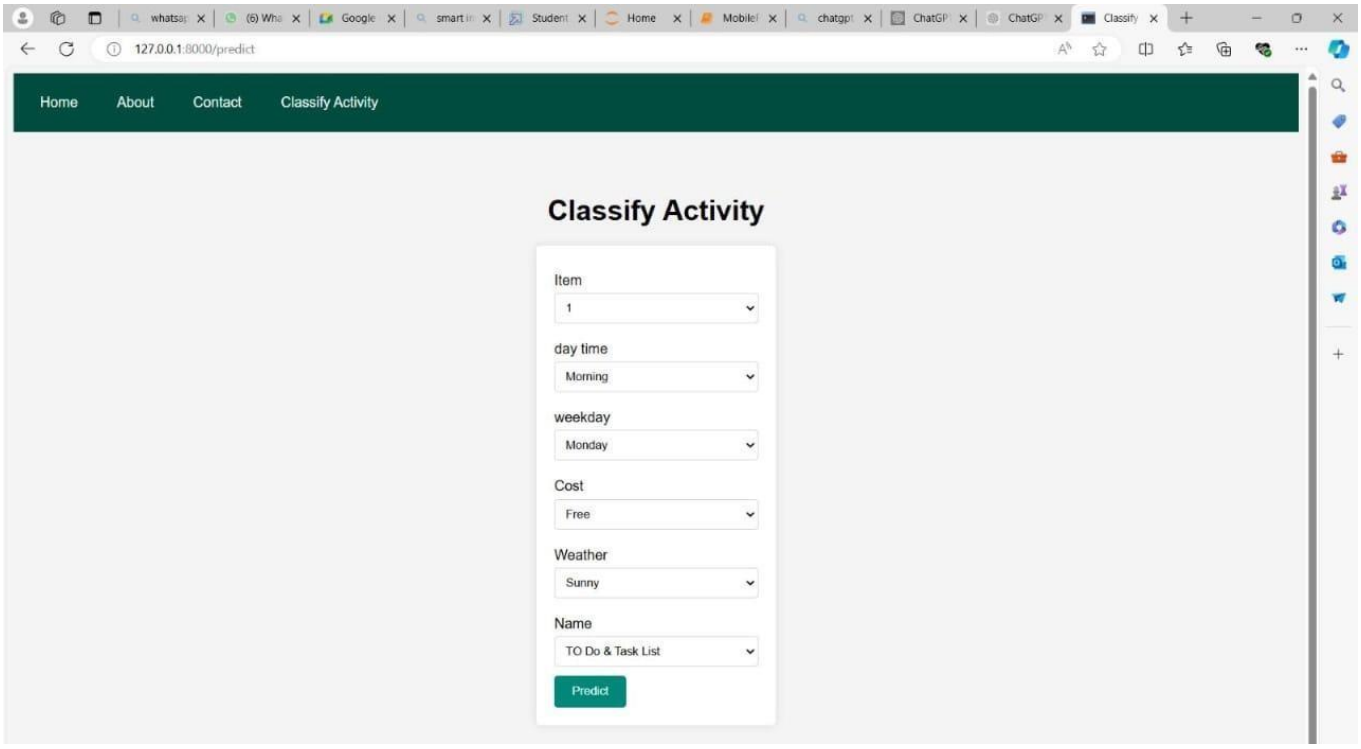

127.0.0.1:8000/about
Home
About
Contact
Classify Activity
ABOUT
In the realm of mobile app analytics, "FrappeActivity Classifier" utilizes advanced machine learning within the Mobile Frappe App. Focused on homework activity classification, the project employs predictive algorithms to discern patterns in user behavior. By distinguishing homework-related activities, it aims to provide valuable insights for personalized app experiences, marking a leap forward in data-driven user engagement.
The central objective of the "FrappeActivity Classifier" is to employ predictive algorithms that can discern patterns in user behaviour within the Mobile Frappe App. Specifically, the project aims to classify and distinguish activities that are related to homework. This classification task is crucial for providing valuable insights into user behaviour and engagement within the app.
FRAPPE ACTIVITY © 2024

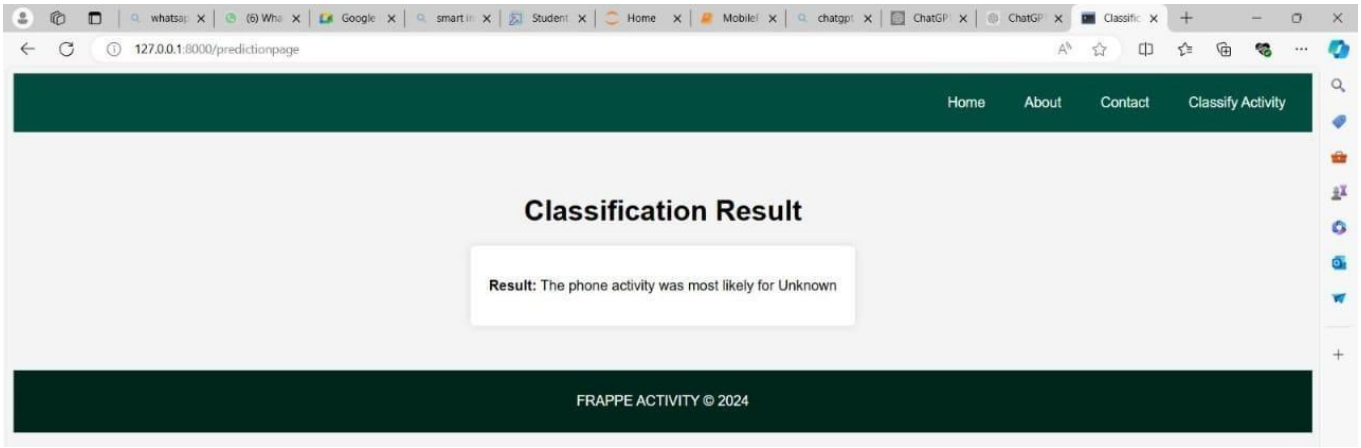

127.0.0.1:8000/contact
Home
About
Contact
Classify Activity
CONTACT
Location: 6th Floor,Sundarayya Vignana Kendram,Technical Block, Madhava Reddy Colony,Gachibowli,Hyderabad,Telangana.
Email: info@thesmartbridge.com
Cell: +91 6304320044
FRAPPE ACTIVITY © 2024


127.0.0.1:8000
Home
About
Contact
Classify Activity
Classify mobile activities within the Mobile Frappe app
Focused on homework activity classification,the project employs predictive algorithms to discern patterns in user behavior.
Classify Activity
FRAPPE ACTIVITY © 2024


127.0.0.1:8000/predict
Home
About
Contact
Classify Activity
Classify Activity
Item
3
day time
Night
weekday
Thursday
Cost
Free
Weather
Sunny
Name
Shopping List
Predict


127.0.0.1:8000/predictionpage
Home
About
Contact
Classify Activity
Classification Result
Result: The phone activity was most likely for Homework





7.ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

1.Improved Accuracy:

- **High accuracy in recognizing and classifying activities.**

- **Capable of learning complex patterns from sensor data.**

2. **Precision in Complex Activities**:

- Distinguishes between similar activities (e.g., walking vs. running) with high Precision.

3. **Rehabilitation**:

- monitors patient progress and allows remote supervision**.**

4. **Sports Performance:**

- Analyses athletic performance and prevents injuries.

5. **Personal Safety:**

- Sends emergency alerts and tracks locations.

DISADVANTAGES:

1. **Privacy Concerns**: Continuous tracking can invade user privacy.
2. **Battery Consumption**: High sensor usage drains battery quickly.
3. **Data Accuracy**: Influenced by sensor quality and device placement, leading to Potential inaccuracies.
4. **Data Security**: Risks of unauthorized access and data misuse.
5. **Computational Overhead**: Intensive processing can slow down devices

8.APPLICATION

- **Health and Fitness Tracking**: Monitoring physical activities, tracking workouts, and encouraging healthy habits through activity data.

- **Personal Safety**: Detecting falls or accidents and sending alerts to emergency contacts or services.
- **Workplace Safety**: Ensuring compliance with safety protocols and monitoring worker activities in hazardous environment
- **Smart Homes**: Automating home environments based on user activity, such as adjusting lighting or heating.

9.CONCLUSION

1. ***Decision Tree Classifier**:*

***Performance**:* Decision Trees performed reasonably well in initial classification tasks but showed limitations in handling complex relationships within data and controlling overfitting.

2. ***Bagging Classifier (with Decision Tree base):***

***Performance**:* The Bagging Classifier, with Decision Tree as the base estimator, demonstrated improved accuracy and robustness compared to standalone Decision Trees, particularly in handling noisy data and improving generalization.

3. ***Random Forest Classifier:***

***Performance**:*

Random Forest exhibited the highest accuracy among the three algorithms, leveraging ensemble learning to achieve robust class handled high-dimensional data and provided insights into feature importance.

Key Findings and Recommendations

***Algorithm Performance**:* Random Forest outperformed both Decision Tree and Bagging Classifier in terms of accuracy and robustness for mobile activity classification.

***Feature Importance**:* Decision Tree and Random Forest algorithms provided insights into which sensor features were most influential in determining activity types, aiding in feature selection and model interpretability.

Deployment Considerations: When deploying models in mobile applications, Random Forest's efficiency in handling real-time inference and its ability to manage computational resources effectively make it a preferred choice**.**

Future Directions:* Further research could explore advanced ensemble techniques, such as Gradient Boosting Machines or Deep Learning models, to potentially enhance classification performance and scalability for large-scale deployments.

10.FUTURE SCOPE

- **Enhanced Model Robustness:** Further refinement of machine learning algorithms handle diverse real-world scenarios and environmental factors that may affect sensor accuracy.

- **Personalized Activity Recognition:** Development of personalized models that adapt to individual user behaviours and preferences, improving accuracy and user satisfaction.

- **Integration with IoT Devices:** Incorporation of data from Internet of Things (IoT) devices to enrich activity context, enabling more comprehensive and accurate classification.

- **Real-Time Feedback Systems:** Implementation of real-time feedback systems based On activity recognition provide immediate insights for users, such as health or safety alerts

11.BIBILOGRAPHY

[1] Chen, Xi, et al. "Deep learning for sensor-based activity recognition: A survey."

Pattern Recognition Letters, vol. 119, 2019, pp. 3-11.

[2] Lara, Oscar D., and Carolina Labrador. "A survey on human activity recognition using wearable sensors." IEEE Communications Surveys & Tutorials, vol. 15, no. 3, 2013, pp. 1192-1209.

[3] Kwapisz, Jennifer R., Gary M. Weiss, and Samuel A. Moore. "Activity recognition using cell phone accelerometers." SIGKDD Explorations, vol. 12, no. 2, 2011, pp. 74-74-82.

[4] Manini, Andrea, and Angelo Maria Sabatini. "Machine learning methods for classifying human physical activity from on-body accelerometers." Sensors, vol. 15, no. 2, 2015, pp. 3133-3156.

[5] Banos, Oresti, et al. "A comprehensive survey on wearable smartphone-based systems for human activity recognition." Sensors, vol. 12, no. 6, 2012, pp. 5797-5828.

[6] Bulling, Andreas, Ulf Blanke, and Bernt Schiele. "A tutorial on human activity recognition using body-worn inertial sensors." ACM Computing Surveys *(CSUR)*, vol.46, no. 3, 2014, Article 33.

[7] Anguita, Davide, et al. "A public domain dataset for human activity recognition using smartphones." 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2013.

[8] Ordonez, Francisco J., and Daniel Roggen. "Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition." Sensors, vol.16, no. 1, 2016, Article 115.

[9] Gjoreski, Hristijan, et al. "Physical activity recognition from wearable sensors:

Challenges and opportunities." IEEE Internet Computing, vol. 21, no. 5, 2017, pp. 42-50.

[10] Amft, Oliver, and Gerhard Troster. "Recognition of dietary activity events using on-body sensors." Artificial Intelligence in Medicine, vol. 42, no. 2, 2008, pp. 121-136.

12.APPENDIX

Model building :

1)Dataset
2)Google collab and VS code Application Building
1. HTML file (home file,about file,contact file, Predictfile, ,)
1. CSS file
2. Models in pickle format

SOURCE CODE:

HOME.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>FRAPPE ACTIVITY</title>
<style>
 body { font-family: Arial, sans-serif; background-color: #f4f4f4; }
 .navbar { background-color: #004d40; padding: 10px; }
  .navbar a { color: white; padding: 14px 20px; text-decoration: none; display: inline-block; }
  .header { text-align: center; padding: 50px; }
  .header h1 { color: #004d40; }
```

```
  .classify-btn { background-color: #00897b; color: white; padding: 10px 20px; border:
none; cursor: pointer; }
   .footer { background-color: #00251a; color: white; text-align: center; padding: 10px; }
</style>
</head>
<body>
<div class="navbar">
 <a href="/">Home</a>
  <a href="about">About</a <a href="contact">Contact</a>
  <a href="predict">Classify Activity</a>
  </div>
  <div class="header">
  <h1>Classify mobile activities within the Mobile Frappe app</h1>
  <p>Focused on homework activity classification,the project employs predictive
algorithms to discern patterns in user behavior.</p>
  <button class="classify-btn">Classify Activity</button>
</div>
<div class="footer">
 <p>FRAPPE ACTIVITY © 2024</p>
  </div>
  </body>
   </html>
```

ABOUT.HTML

```
 <!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>About - FRAPPE ACTIVITY</title>
<style>   body { font-family: Arial, sans-serif; background-color:
#f4f4f4; }
  .navbar { background-color: #004d40; padding: 10px; }
  .navbar a { color: white; padding: 14px 20px; text-decoration: none; display: inlineblock;
}
  .content { text-align: center; padding: 50px; }
```

```
  .footer { background-color: #00251a; color: white; text-align: center; padding: 10px; }
</style> </head>
<body>
<div class="navbar">
  <a href="/">Home</a>
  <a href="/about">About</a>
  <a href="/contact">Contact</a>
  <a href="/predict">Classify Activity</a>
</div>
<div class="content">
  <h1>ABOUT</h1>
   <p> In the realm of mobile app analytics, "FrappeActivity Classifier" utilizes advanced machine learning within the Mobile Frappe App. Focused on homework activity classification, the project employs predictive algorithms to discern patterns in user behaviour.
        By distinguishing homework-related activities, it aims to provide valuable insights for personalized app experiences, marking a leap forward in data-driven user engagement.</p>

    <p>The central objective of the "FrappeActivity Classifier" is to employ predictive algorithms that can discern patterns in user behaviour within the Mobile Frappe App. Specifically, the project aims to classify and distinguish activities that are related to homework.
       This classification task is crucial for providing valuable insights into user behaviour and engagement within the app.</p>
</div>
<div class="footer">
  <p>FRAPPE ACTIVITY © 2024</p>
</div>
</body>
</html>
```

CONTACT.HTML

```
<!DOCTYPE
html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Contact - FRAPPE ACTIVITY</title>
<style>   body { font-family: Arial, sans-serif; background-color:
#f4f4f4; }
  .navbar { background-color: #004d40; padding: 10px; }
  .navbar a { color: white; padding: 14px 20px; text-decoration: none; display: inlineblock;
}
  .content { text-align: center; padding: 50px; }
_.footer { background-color: #00251a; color: white; text-align: center; padding: 10px; }
</style> </head>
<body>
<div class="navbar">
  <a href="/">Home</a>
  <a href="/about">About</a>
  <a href="/contact">Contact</a>
  <a href="/predict">Classify Activity</a>
</div>
<div class="content">
  <h1>CONTACT</h1>
  <p>Location: 6th Floor,Sundarayya Vignana Kendram,Technical Block,
            Madhava Reddy Colony,Gachibowli,Hyderabad,Telangana.</p>
  <p>Email: <a
href="mailto:lohithravula12@gmail.com">info@thesmartbridge.com</a></p>
<p>Cell: +91 6304320044</p>
</div>
<div class="footer">
 <p>FRAPPE ACTIVITY © 2024</p>
  </div
</body>
</html>
```

PREDICT.HTML

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Classify Activity - FRAPPE ACTIVITY</title>
<style>   body { font-family: Arial, sans-serif; background-color:
#f4f4f4; }
  .navbar { background-color: #004d40; padding: 10px; }
  .navbar a { color: white; padding: 14px 20px; text-decoration: none; display: inlineblock;
}
  .content { text-align: center; padding: 50px; }
  .form-container { background-color: white; padding: 20px; border-radius: 5px;
boxshadow: 0 0 10px rgba(0, 0, 0, 0.1); display: inline-block; text-align: left; }
  .form-container label { display: block; margin: 10px 0 5px; }
  .form-container select, .form-container input { width: 100%; padding: 8px;
marginbottom: 10px; border: 1px solid #ccc; border-radius: 4px; }
  .submit-btn { background-color: #00897b; color: white; padding: 10px 20px; border:
none; cursor: pointer; border-radius: 4px; }
  .footer { background-color: #00251a; color: white; text-align: center; padding: 10px; }
</style> </head>
<body>
<div class="navbar">
  <a href="/">Home</a>
  <a href="/about">About</a>
  <a href="/contact">Contact</a
 <a href="/predict">Classify Activity</a>
</div>
<div class="content">
  <h1>Classify Activity</h1>
  <div class="form-container">
  <form action="{{url_for('predictionpage')}}" method="POST">
</select>
    <label for="item">Item</label>
    <select id="item"  name="item">
    <option value="1">1</option>
```

```
<option value="2">2</option>
<option value="3">3</option>
</select>
<label for="daytime">day time</label>
<select id="daytime" name="daytime">
<option value="morning">Morning</option>
<option value="afternoon">Afternoon</option>
<option value="evening">Evening</option>
<option value="sunset">Sunset</option>
<option value="night">Night</option>
<option value="noon">Noon</option>
<option value="sunrise">Sunrise</option>
 </select>
 <label for="weekday">weekday</label>
 <select id="weekday" name="weekday">
  <option value="monday">Monday</option>
  <option value="tuesday">Tuesday</option>
  <option value="wednesday">Wednesday</option>
   <option value="thursday">Thursday</option>
   <option value="friday">Friday</option>
   <option value="saturday">Saturday</option>
   <option value="sunday">Sunday</option>
  </select>
  <label for="cost">Cost</label>
  <select id="cost" name="cost">
  <option value="free">Free</option>
  <option value="paid">Paid</option>
  </select>
  <label for="weather">Weather</label>
  <select id="weather" name="weather">
   <option value="sunny">Sunny</option>
   <option value="cloudy">Cloudy</option>
   <option value="unknown">Unknown</option>
```

```
<option value="foggy">Foggy</option>
<option value="rainy">Rainy</option>
</select>
<label for="sname">Name</label>
<select id="sname" name="sname">
<option value="260">TO Do & Task List</option>
<option value="3217">Yahoo!</option>
<option value="698">Compass PRO</option>
<option value="1489">Instagram</option>
<option value="2523">Shopping List</option>
</select>
<button type="submit" class="submit-btn">Predict</button>
</form
</div>
</div>
<div class="footer">
<p>FRAPPE ACTIVITY © 2024</p>
</div>
</body>
</html>
```

PREDICTIONPAGE.HTML

```
<DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Classification Result - FRAPPE ACTIVITY</title>
<style>        body { font-family: Arial, sans-serif; background-
color: #f4f4f4; }
.navbar { background-color: #004d40; padding: 10px; text-align: right; }
.navbar a { color: white; padding: 14px 20px; text-decoration: none; display:
inlineblock; }
.content { text-align: center; padding: 50px; }
```

```
.result-container { background-color: white; padding: 20px; border-radius: 5px;
boxshadow: 0 0
10px rgba(0, 0, 0, 0.1); display: inline-block; text-align: left; }
.footer { background-color: #00251a; color: white; text-align: center; padding: 10px;
} </style>
</head>
<body>
<div class="navbar">
<a href="/">Home</a>
<a href="/about">About</a>
<a href="/contact">Contact</a>
<a href="/predict">Classify Activity</a>
</div>
<div class="content">
<h1>Classification Result</h1>
<div class="result-container">
<p><strong>Result:</strong> {{bc_final}}</p>
</div>
</div>
<div class="footer">
<p>FRAPPE ACTIVITY © 2024</p>
</div>
</body>
</html>
```

APP.PY

```
import pandas as pd
from flask import Flask, render_template, request
from sklearn.preprocessing import StandardScaler
import joblib  import os  app = Flask(_name_) #
Load the model and scaler
model = pickle.load(open("model.pkl", "rb")) scaler =
pickle.load(open("scaler.pkl", "rb")) encoderspath =
os.path.dirname(os.path.abspath(_file_)) dayencoder =
joblib.load(os.path.join(encoders_path, 'DayTimeEncoder')) wkencoder =
```

```
joblib.load(os.path.join(encoders_path, 'WeekdayEncoder'))
wkndencoder = joblib.load(os.path.join(encoders_path, 'WkndEncoder'))
hwencoder = joblib.load(os.path.join(encoders_path, 'hwencoder'))
wencoder = joblib.load(os.path.join(encoders_path, 'WeatherEncoder'))
cencoder = joblib.load(os.path.join(encoders_path, 'CostEncoder'))
nncoder = joblib.load(os.path.join(encoders_path, 'NameEncoder'))
@ app.route('/') def
home():
   return render_template("home.html")
@app. route('/about') def
about():
   return render_template("about.html")
@ app.route('/contact') def
contact():
   return render_template("contact.html")
# Route to predict page
@ app.route('/predict', methods=['GET', 'POST']) def
predict():
   return render_template("predict.html") @
app.route('/predictionpage', methods=['POST'])
def predictionpage():     try:
       # Read CSV file for additional data if needed
       df = pdread_csv(rC:\Users\Lohith\Desktop\project1\dataset\frappe.csv')
item = int(request.form['item'])        daytime = request.form.get('daytime')
weekday = request.form.get("weekday")        cost = request.form.get("cost")
weather = request.form.get("weather")        sname =
int(request.form.get('sname'))        if not all([daytime, weekday, cost,
weather]):            return "Error: Missing form data.", 400        if daytime not
in dayencoder.classes_:
         return f"Error: 'daytime' contains an unknown label: {daytime}", 400
if weekday not in wkencoder.classes_:
          return f"Error: 'weekday' contains an unknown label: {weekday}", 400
if cost not in cencoder.classes_:
          return f"Error: 'cost' contains an unknown label: {cost}", 400
if weather not in wencoder.classes_:
```

```
        return f"Error: 'weather' contains an unknown label: {weather}", 400
daytime_encoded = dayencoder.transform([daytime])[0]       weekday_encoded =
wkencoder.transform([weekday])[0]
      cost_encoded = cencoder.transform([cost])[0]
weather_encoded = wencoder.transform([weather])[0]
       iswknd = "weekend" if weekday.lower() in ['sunday', 'saturday'] else "workday"
iswknd_encoded = wkndencoder.transform([iswknd])[0]
      x_test = [[item, daytime_encoded, weekday_encoded, iswknd_encoded,
cost_encoded, weather_encoded,sname]]
      x_test_scaled = scaler.transform(x_test)
pred = model.predict(x_test_scaled)
      activity_mapping = {0: "Homework", 1: "Unknown", 2: "Work"}
      result = f"The phone activity was most likely for {activity_mapping.get(pred[0],
'Unknown')}"
      return render_template("predictionpage.html", bc_final=result)
except Exception as e:
      return f"Error: {str(e)}", 500 if
_name_ == '_main_':
   app.run(debug=True, port=8000)
```

13. CODE SNIPPETS

MODEL BUILDING

```
!pip install scikit-learn==1.5.1
import sklearn
print(sklearn.__version__)
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scikit-learn==1.5.1 in c:\users\lohith\appdata\roaming\python\python312\site-packages (1.5.1)
Requirement already satisfied: numpy>=1.19.5 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn==1.5.1) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn==1.5.1) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn==1.5.1) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\lohith\appdata\roaming\python\python312\site-packages (from scikit-learn==1.5.1) (3.5.0)
1.5.1
```

```python
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

import pickle
from sklearn.model_selection import RandomizedSearchCV
```

Python

```python
meta_app=pd.read_csv('meta.csv',sep='\t')
```

```python
meta_app.head()
```

Python

	item	package	category	downloads	developer	icon	language	description	name	price
0	0	com.anydo	Productivity	1,000,000 - 5,000,000	Any.DO	http://d2lh3rxs7crswz.cloudfront.net/com.anydo...	en	Meet Any.DO, the best way to-do To-do's on And...	Any.DO To-do & Tasks List	Free
1	1	com.yahoo.mobile.client.android.yahoo	News & Magazines	1,000,000 - 5,000,000	Yahoo! Inc.	http://d2lh3rxs7crswz.cloudfront.net/com.yahoo...	en	With Yahoo! for Android, you'll stay connected...	Yahoo!	Free
2	2	com.compasspro	Tools	1,000,000 - 5,000,000	Mobile Essentials	http://d2lh3rxs7crswz.cloudfront.net/com.compa...	en	Professional Compass for Android. Simple and p...	Compass PRO	Free
3	3	com.instagram.android	Social	50,000,000 - 100,000,000	Instagram	http://d2lh3rxs7crswz.cloudfront.net/com.insta...	en	Instagram — A beautiful way to share your worl...	Instagram	Free
4	4	com.shoppinglist	Shopping	10,000 - 50,000	Kiwi3	http://d2lh3rxs7crswz.cloudfront.net/com.shopp...	en	Shopping List is the most easy way to organize...	Shopping List	Free

```python
meta_app=meta_app[['name','item']]
print(meta_app)
```

Python

```
                                name  item
0          Any.DO To-do & Tasks List     0
1                              Yahoo!     1
2                        Compass PRO     2
3                          Instagram     3
4                      Shopping List     4
...                              ...   ...
4077           Monster Galaxy Exile  4077
4078                Legends of Yore  4078
4079      Talking Jumpster from Mars  4079
4080  Extensive Notes Pro - Notepad  4080
4081              Android Pro Widgets  4081

[4082 rows x 2 columns]
```

```python
print(df.columns)
print(meta_app.columns)
```

Python

```
Index(['user', 'item', 'cnt', 'daytime', 'weekday', 'isweekend', 'homework',
       'cost', 'weather', 'country', 'city'],
      dtype='object')
Index(['name', 'item'], dtype='object')
```

```
df=df.merge(meta_app,on= 'item')

df.head()
```

[10] Python

	user	item	cnt	daytime	weekday	isweekend	homework	cost	weather	country	city	name
0	0	0	1	morning	sunday	weekend	unknown	free	sunny	United States	0	Any.DO To-do & Tasks List
1	1	1	7	afternoon	saturday	weekend	unknown	free	cloudy	Spain	0	Yahoo!
2	2	2	6	evening	monday	workday	unknown	free	cloudy	Spain	369	Compass PRO
3	3	3	1	sunset	thursday	workday	unknown	free	unknown	United States	1028	Instagram
4	4	4	428	night	thursday	workday	home	free	sunny	Switzerland	147	Shopping List

+ Code + Markdown

```
df.shape
```

[11] Python

```
(96203, 12)
```

```
df.info()
```

[12] Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96203 entries, 0 to 96202
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   user        96203 non-null  int64 
 1   item        96203 non-null  int64 
 2   cnt         96203 non-null  int64 
 3   daytime     96203 non-null  object
 4   weekday     96203 non-null  object
 5   isweekend   96203 non-null  object
 6   homework    96203 non-null  object
 7   cost        96203 non-null  object
 8   weather     96203 non-null  object
 9   country     96203 non-null  object
 10  city        96203 non-null  int64 
 11  name        96203 non-null  object
dtypes: int64(4), object(8)
memory usage: 8.8+ MB
```

```
df.duplicated().sum()
```

[13] Python

```
0
```

```
df.isna().sum()
```

[14] Python

```
user          0
item          0
cnt           0
daytime       0
weekday       0
isweekend     0
homework      0
cost          0
weather       0
country       0
city          0
name          0
dtype: int64
```

```
dt_encoder=LabelEncoder()
dt_encoder.fit(df['daytime'])
df['daytime']=dt_encoder.transform(df['daytime'])
wd_encoder=LabelEncoder()
wd_encoder.fit(df['weekday'])
df['weekday']=wd_encoder.transform(df['weekday'])
wknd_encoder=LabelEncoder()
wknd_encoder.fit(df['isweekend'])
df['isweekend']=wknd_encoder.transform(df['isweekend'])
hw_encoder=LabelEncoder()
hw_encoder.fit(df['homework'])
df['homework']=hw_encoder.transform(df['homework'])
c_encoder=LabelEncoder()
c_encoder.fit(df['cost'])
df['cost']=c_encoder.transform(df['cost'])
w_encoder=LabelEncoder()
w_encoder.fit(df['weather'])
```

```
df['weather']=w_encoder.transform(df['weather'])

n_encoder=LabelEncoder()

n_encoder.fit(df['name'])

df['name']=n_encoder.transform(df['name'])
```

[15] Python

```
[16]
df.info()
```

Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96203 entries, 0 to 96202
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   user       96203 non-null  int64
 1   item       96203 non-null  int64
 2   cnt        96203 non-null  int64
 3   daytime    96203 non-null  int32
 4   weekday    96203 non-null  int32
 5   isweekend  96203 non-null  int32
 6   homework   96203 non-null  int32
 7   cost       96203 non-null  int32
 8   weather    96203 non-null  int32
 9   country    96203 non-null  object
 10  city       96203 non-null  int64
 11  name       96203 non-null  int32
dtypes: int32(7), int64(4), object(1)
memory usage: 6.2+ MB
```

```
[17]
df.describe()
```

Python

	user	item	cnt	daytime	weekday	isweekend	homework	cost	weather	city	name
count	96203.000000	96203.000000	96203.000000	96203.000000	96203.000000	96203.000000	96203.000000	96203.000000	96203.000000	96203.000000	96203.000000
mean	189.646944	386.754207	81.330894	2.165317	2.885627	0.659792	0.885565	0.050872	3.432835	333.966176	1800.068990
std	156.458370	683.779627	395.414980	1.825046	2.009485	0.473781	0.447594	0.219737	3.380435	359.977130	1041.000073
min	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	21.000000	2.000000	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1037.000000
50%	153.000000	77.000000	8.000000	2.000000	3.000000	1.000000	1.000000	0.000000	2.000000	327.000000	1489.000000
75%	271.000000	397.000000	36.000000	3.000000	5.000000	1.000000	1.000000	0.000000	7.000000	649.000000	2960.000000
max	956.000000	4081.000000	28752.000000	6.000000	6.000000	1.000000	2.000000	1.000000	8.000000	1087.000000	3508.000000

```
[18]
print(df['item'].dtype)
print(meta_app['item'].dtype)
```

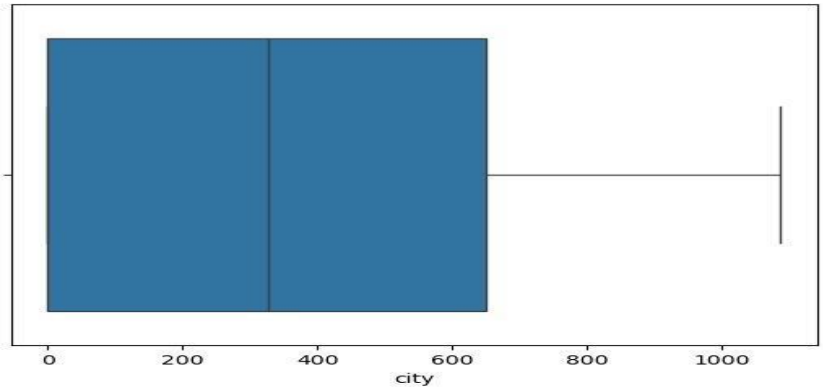
Python

```
int64
int64
```

```
[20]
sns.boxplot(x='city',data=df)
```

Python

```
<Axes: xlabel='city'>
```



```
[21]
df['city'] = pd.to_numeric(df['city'], errors='coerce')
```
Python

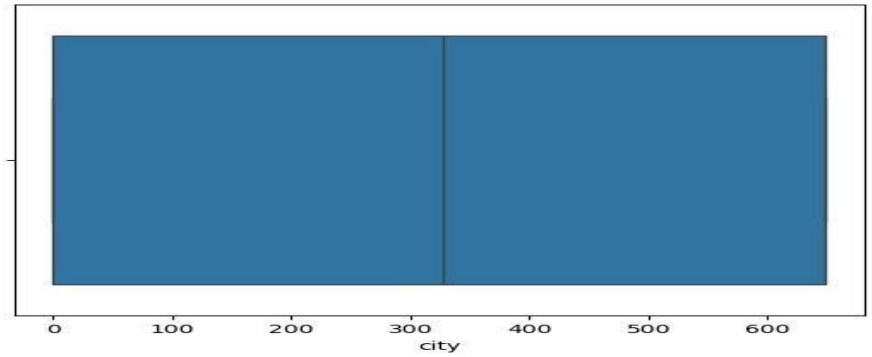
```
quant = df['city'].quantile(q=[0.25, 0.75])
print(quant)
Q1 = quant.iloc[0]
print(Q1)
Q3 = quant.iloc[1]
print(Q3)
IQR = Q3 - Q1
print(IQR)
maxwhisker = Q3 + 1.5 * IQR
print(maxwhisker)
minwhisker = Q1 - 1.5 * IQR
print(minwhisker)
```
[22] Python

```
0.25      0.0
0.75    649.0
Name: city, dtype: float64
0.0
649.0
649.0
1622.5
-973.5
```

```
[23]
df['city']=np.where(df['city']>649,649,df['city'])
```
Python

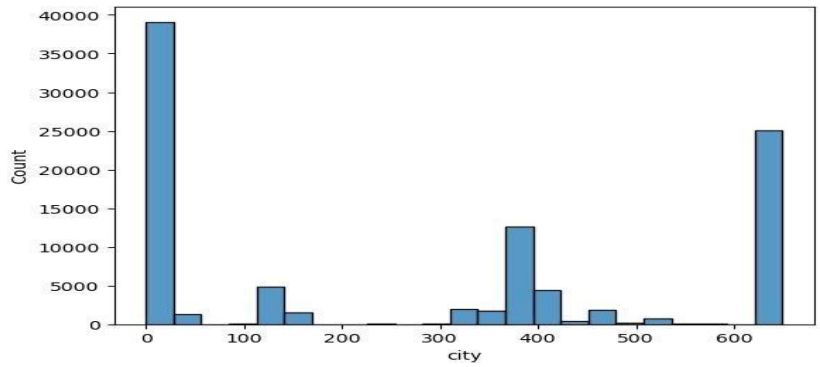
```
[24]
sns.boxplot(x='city',data=df)
```
Python

```
<Axes: xlabel='city'>
```

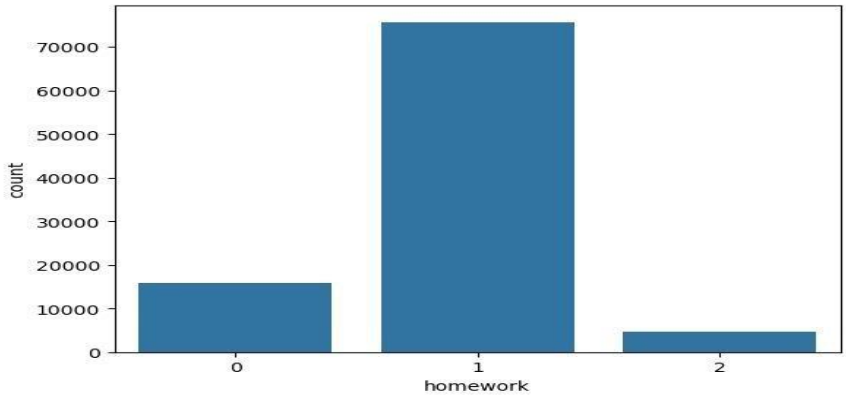



```
[25]
sns.histplot(x='city',data=df)
```
Python

```
<Axes: xlabel='city', ylabel='Count'>
```

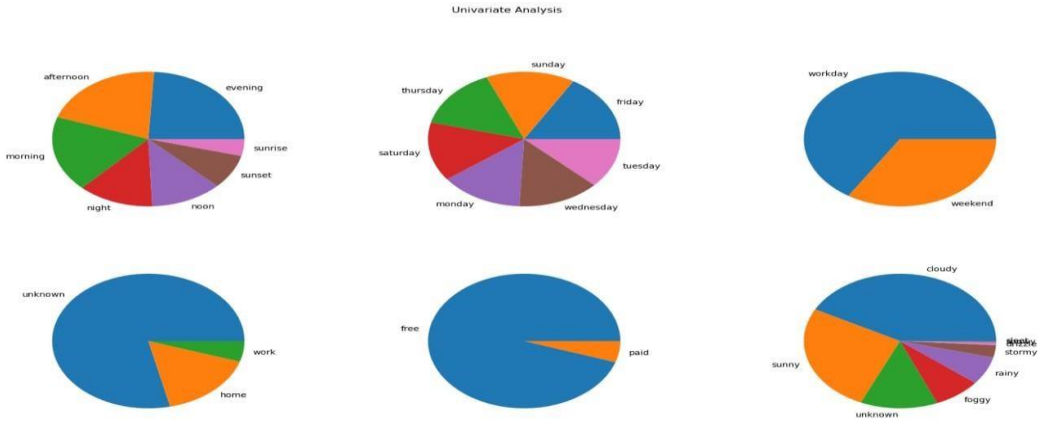



```
[26]  sns.countplot(x='homework',data=df)
                                                                    Python
...   <Axes: xlabel='homework', ylabel='count'>
```




```
fig,axes=plt.subplots(2,3,figsize=(20,10))
fig.suptitle("Univariate Analysis")
col=['daytime','weekday','isweekend','homework','cost','weather']
k=0
for i in range(2):
  for j in range(3):
    d=dict(df[col[k]].value_counts())
    axes[i,j].pie(d.values(),labels=d.keys())
    k+=1
```

Pie chart:

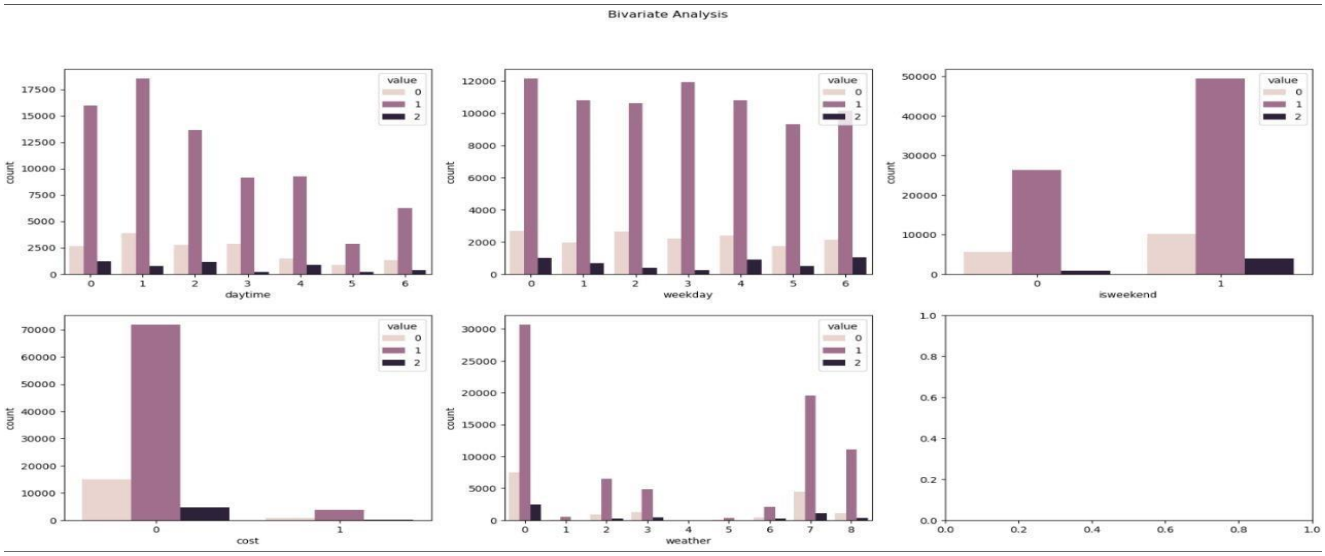


Bivariate Analysis:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

fig,axes=plt.subplots(2,3,figsize=(20,10))
fig.suptitle("Bivariate Analysis")

# Melt the dataframe to long-form
df_melted = pd.melt(df, id_vars=['daytime', 'weekday', 'isweekend','cost','weather'], value_vars=['homework'])

# Use the melted dataframe in the countplots
sns.countplot(x='daytime', hue='value', data=df_melted, ax=axes[0,0])
sns.countplot(x='weekday', hue='value', data=df_melted, ax=axes[0,1])
sns.countplot(x='isweekend', hue='value', data=df_melted, ax=axes[0,2])
sns.countplot(x='cost' , hue='value', data=df_melted, ax=axes[1,0])
sns.countplot(x='weather', hue='value',data=df_melted,ax=axes[1,1])
```

[28] Python

:




```
# Feature Selection
features = ['item','daytime','weekday','isweekend','cost','weather','name']
X = df[features]
y = df[["homework"]]
```

[29] Python

```
print(df['name'])
```

[30] Python

```
0          260
1         3217
2          698
3         1489
4         2523
         ...
96198      260
96199     1032
96200     1934
96201     1307
96202     2429
Name: name, Length: 96203, dtype: int32
```

```
from imblearn.over_sampling import SMOTE
```

[31] Python

```python
# Feature Selection
features = ['item','daytime','weekday','isweekend','cost','weather','name']
X = df[features]
y = df[["homework"]]
```

[29] Python

```python
print(df['name'])
```

[30] Python

```
0          260
1         3217
2          698
3         1489
4         2523
         ...
96198      260
96199     1037
96200     1994
96201     1307
96202     2429
Name: name, Length: 96203, dtype: int32
```

```python
from imblearn.over_sampling import SMOTE
```

[31] Python

```python
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.8, random_state=42)
```

[35] Python

```python
# Scaling Data so that all the features has similar weight.
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[36] Python

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
```

[37] Python

```python
models=[]
models = [('AdaBoost', AdaBoostClassifier(algorithm='SAMME'))]
models.append(('KNeighborsClassifier',KNeighborsClassifier()))
models.append(('DecisionTreeClassifier',DecisionTreeClassifier()))
models.append(('RandomForestClassifier',RandomForestClassifier()))
models.append(('XGBClassifier',XGBClassifier()))
models.append(('BaggingClassifier',BaggingClassifier()))
```

```python
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score

model_perform = {}

for name, model in models:
    # Fit the model
    model.fit(X_train, y_train.values.ravel())

    # Make predictions
    y_pred = model.predict(X_test)

    # Calculate performance metrics
    p = precision_score(y_test, y_pred, average='micro')
    r = recall_score(y_test, y_pred, average='micro')
    a = accuracy_score(y_test, y_pred)
    f = f1_score(y_test, y_pred, average='micro')

    # Store the performance metrics
    s = {'Precision': p, 'Recall': r, 'Accuracy': a, 'F1 Score': f}
    model_perform[name] = s
```

[39] Python

```python
for model in model_perform:
    print(model)
    print("Precision: ",model_perform[model]['Precision'])
    print("Recall: ",model_perform[model]['Recall'])
    print("Accuracy: ",model_perform[model]['Accuracy'])
    print("F1 Score: ",model_perform[model]['F1 Score'])
    print()
```

```
AdaBoost
Precision:  0.4572430729923792
Recall:  0.4572430729923792
Accuracy:  0.4572430729923792
F1 Score:  0.4572430729923792

KNeighborsClassifier
Precision:  0.5377571472622351
Recall:  0.5377571472622351
Accuracy:  0.5377571472622351
F1 Score:  0.5377571472622351

DecisionTreeClassifier
Precision:  0.6204242103872076
Recall:  0.6204242103872076
Accuracy:  0.6204242103872076
F1 Score:  0.6204242103872076

RandomForestClassifier
Precision:  0.6379729967842827
Recall:  0.6379729967842827
Accuracy:  0.6379729967842827
F1 Score:  0.6379729967842827

XGBClassifier
...
Recall:  0.6371305228844544
Accuracy:  0.6371305228844544
F1 Score:  0.6371305228844544
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
[41]
dt_classifier=DecisionTreeClassifier()
```
Python

```
[42]
# Define the hyperparameters and their possible values for tuning
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 2, 4, 6, 8,10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2'],
    'min_impurity_decrease': [0.0, 0.1, 0.2],
    'ccp_alpha': [0.0, 0.1 ,0.2]
}
```
Python

```
[43]
# Initialize RandomizedSearchCV with DecisionTreeClassifier
random_search = RandomizedSearchCV(estimator=dt_classifier,
                                   param_distributions=param_grid,
                                   scoring='accuracy',
                                   cv=3,
                                   n_iter=100,
                                   random_state=42)
```
Python

```
[44]
random_search.fit(X_train, y_train)
RandomizedSearchCV(cv=3,estimator=DecisionTreeClassifier(),n_iter=100,
                   param_distributions={'ccp_alpha':[0.0, 0.1, 0.2],
                                        'criterion':['gini','entropy'],
                                        'max_depth':[None, 2, 4, 6, 8, 10],
                                        'max_features':[None,'sqrt','log2'],
                                        'min_impurity_decrease':[0.0, 0.1, 0.2],
                                        'min_samples_leaf':[1, 2, 4],
                                        'min_samples_split':[2, 5, 10],
                                        'splitter': ['best', 'random']},
                   random_state=42, scoring='accuracy')

print("Best Parameters:", random_search.best_params_)

print("Best Score:", random_search.best_score_)
```
Python

```
Best Parameters: {'splitter': 'best', 'min_samples_split': 2, 'min_samples_leaf': 2, 'min_impurity_decrease': 0.0, 'max_features': None, 'max_depth': None,
Best Score: 0.5765605039425576
```

```
[45]
print("Best Score:",random_search.score(X_test,y_test))
```
Python

```
Best Score: 0.6036573278710189
```

```
[46]
dt_final=random_search
```
Python

```
[47]
from sklearn.metrics import classification_report,confusion_matrix

print(classification_report(y_test,y_pred,digits=4))
```
Python

```
              precision    recall  f1-score   support

           0     0.5928    0.6164    0.6043     60466
           1     0.6314    0.5321    0.5775     60427
           2     0.6826    0.7623    0.7202     60715

    accuracy                         0.6371    181608
   macro avg     0.6356    0.6369    0.6340    181608
weighted avg     0.6356    0.6371    0.6342    181608
```

+ Code + Markdown

```
[48]
y_pred=dt_final.predict(X_test)

print(dt_final.score(X_test,y_test))
cm=confusion_matrix(y_test,y_pred,labels=[0,1,2])
df_cm = pd.DataFrame(cm,index=[i for i in ["Homework", "Unknown", "Work"]],
                     columns = [i for i in ["Homework", "Unknown", "Work"]])

plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True,fmt='g',cmap=sns.cubehelix_palette(as_cmap=True))
```
Python

```
0.6036573278710189
<Axes: >
```

```
base_estimator = DecisionTreeClassifier()
```

```
bagging_classifier = BaggingClassifier(estimator=base_estimator)
```

```
# Define the hyperparameters and their possible values for tuning
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_samples': [0.5, 0.7, 1.0],
    'max_features': [0.5, 0.7, 1.0],
    'bootstrap': [True, False],
    'bootstrap_features': [True, False]
}
```

```
random_search = RandomizedSearchCV(estimator=bagging_classifi (function) random_state: Any grid,
                                   scoring='accuracy', cv=2, random_state=42)
```

```
y_train = y_train.values.ravel()

random_search.fit(X_train,y_train)
```

RandomizedSearchCV
best_estimator_: BaggingClassifier
estimator: DecisionTreeClassifier
DecisionTreeClassifier

```
print("Best Parameters:",random_search.best_params_)
print("Best Score:",random_search.best_score_)
```

```
Best Parameters: {'n_estimators': 100, 'max_samples': 0.7, 'max_features': 1.0, 'bootstrap_features': True, 'bootstrap': False}
Best Score: 0.651050614510374
```

```
print("Best Score:",random_search.score(X_test,y_test))
```

```
Best Score: 0.6873320558565702
```

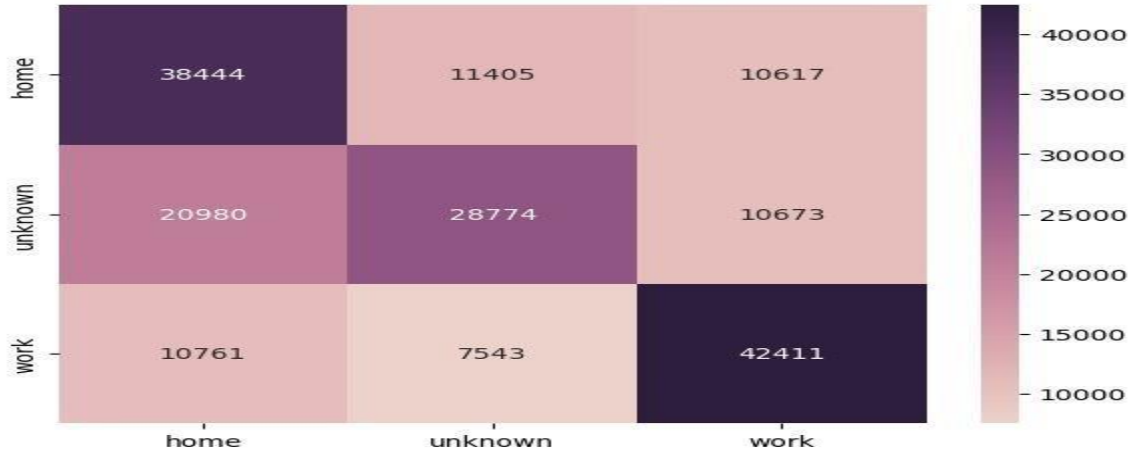
```
bc_final=random_search
```

```
print(classification_report(y_test,y_pred,digits=4))
```

```
              precision    recall  f1-score   support

           0     0.5478    0.6358    0.5885     60466
           1     0.6030    0.4762    0.5321     60427
           2     0.6658    0.6985    0.6818     60715

    accuracy                         0.6037    181608
   macro avg     0.6055    0.6035    0.6008    181608
weighted avg     0.6056    0.6037    0.6009    181608
```

```
from sklearn.metrics import classification_report, confusion_matrix

y_pres=dt_final.predict(X_test)
print(dt_final.score(X_test, y_test))
cm = confusion_matrix(y_test, y_pred, labels=[0, 1, 2])
df_cm = pd.DataFrame(cm, index=[i for i in ["home", "unknown", "work"]],
                     columns=[i for i in ["home", "unknown", "work"]])
plt.figure(figsize=(7, 5))
sns.heatmap(df_cm,annot=True,fmt='g',cmap=sns.cubehelix_palette(as_cmap=True))
```

```
0.6036573278710189
<Axes: >
```



```
from sklearn.ensemble import RandomForestClassifier
rf_final=RandomForestClassifier()
```

Python

```
rf_final.fit(X_train,y_train)
```

Python

```
RandomForestClassifier
RandomForestClassifier()
```

```
print("Best Score:",rf_final.score(X_test,y_test))
```

Python

```
Best Score: 0.6396524382185806
```

```
print(classification_report(y_test,y_pred,digits=4))
```

Python

```
              precision    recall  f1-score   support

           0     0.5478    0.6358    0.5885     60466
           1     0.6030    0.4762    0.5321     60427
           2     0.6658    0.6985    0.6818     60715

    accuracy                         0.6037    181608
   macro avg     0.6055    0.6035    0.6008    181608
weighted avg     0.6056    0.6037    0.6009    181608
```

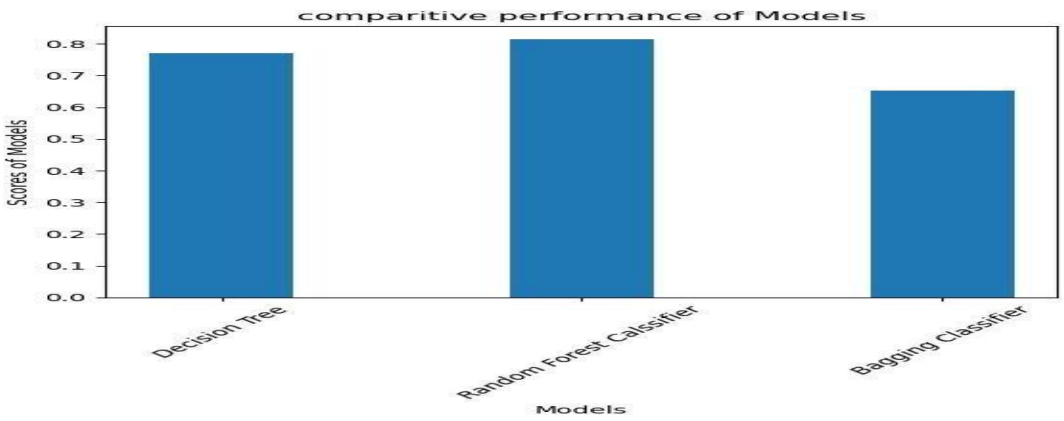
```
from sklearn.metrics import classification_report, confusion_matrix

y_pred=rf_final.predict(X_test)
print(rf_final.score(X_test,y_test))
cm=confusion_matrix(y_test,y_pred, labels=[0,1,2])
df_cm = pd.DataFrame(cm, index = [i for i in ["Home", "Unknown", "work"]],
                     columns = [i for i in ["Home", "Unknown", "Work"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True,fmt='g',cmap=sns.cubehelix_palette(as_cmap=True))
```

Python

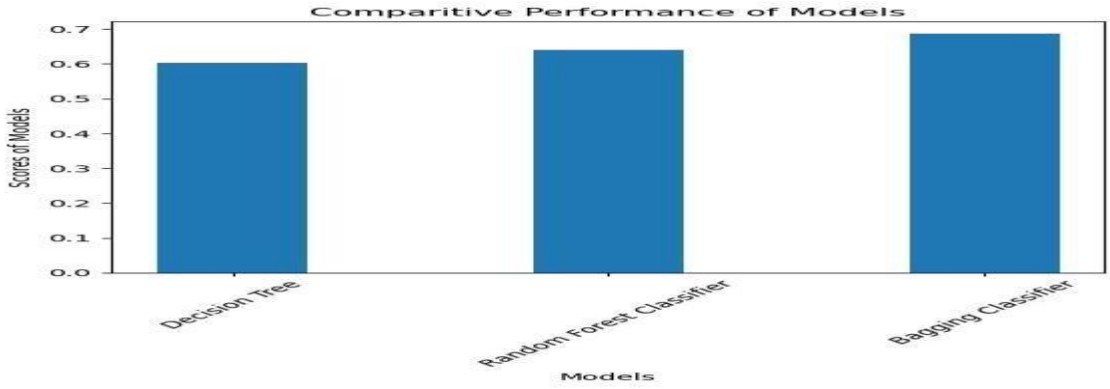
```
0.6396524382185806
```

```
<Axes: >
```


```
train_score={'Decision Tree':dt_final.score(X_train,y_train),'Random Forest Calssifier':rf_final.score(X_train,y_train),'Bagging Classifier':bc_final.be
models=list(train_score.keys())
scores=list(train_score.values())
plt.bar(models,scores,width=0.4)
plt.xlabel("Models")
plt.ylabel("Scores of Models")
plt.xticks(rotation=45)
plt.title("comparitive performance of Models")
plt.show()
```




```
test_score={'Decision Tree':dt_final.score(X_test,y_test),'Random Forest Classifier':rf_final.score(X_test,y_test),'Bagging Classifier':bc_final.score(X
models=list(test_score.keys())
scores=list(test_score.values())

plt.bar(models,scores,width=0.4)

plt.xlabel("Models")
plt.ylabel("Scores of Models")
plt.xticks(rotation=45)
plt.title("Comparitive Performance of Models")
plt.show()
```




```
print(train_score)
print(test_score)
```

[66] Python

```
{'Decision Tree': 0.7697238007136249, 'Random Forest Calssifier': 0.8147438438835294, 'Bagging Classifier': 0.651050614510374}
{'Decision Tree': 0.6036573278710189, 'Random Forest Classifier': 0.6396524382185806, 'Bagging Classifier': 0.6873320558565702}
```

```
import pickle
```

[67] Python

```
#saving the best model
pickle.dump(bc_final,open("model.pkl","wb"))
```

[68] Python

```
pickle.dump(scaler,open("scaler.pkl","wb"))
```

[69] Python

```
import joblib

joblib.dump(dt_encoder, "DayTimeEncoder")
joblib.dump(wd_encoder, "WeekdayEncoder")
joblib.dump(wknd_encoder, "WkndEncoder")
joblib.dump(hw_encoder, "HWEncoder")
joblib.dump(w_encoder, "WeatherEncoder")
joblib.dump(c_encoder, "CostEncoder")
joblib.dump(n_encoder, "NameEncoder")
```

```
['NameEncoder']
```

```
df
```

Python

	user	item	cnt	daytime	weekday	isweekend	homework	cost	weather	country	city	name
0	0	0	1	2	3	0	1	0	7	United States	0	260
1	1	1	7	0	2	0	1	0	0	Spain	0	3217
2	2	2	6	1	1	1	1	0	0	Spain	369	698
3	3	3	1	6	4	1	1	0	8	United States	649	1489
4	4	4	428	3	4	1	0	0	7	Switzerland	147	2523
...	...	...	...	...	...	...	...	...	...	...	...	...
96198	110	0	5	1	3	0	1	0	7	United States	0	260
96199	37	16	101	6	3	0	1	0	0	Canada	128	1037
96200	181	33	243	0	3	0	1	0	0	Israel	454	1994
96201	451	752	1	1	3	0	1	0	7	United States	0	1307
96202	364	380	3	1	4	1	1	0	0	United States	0	2429

96203 rows × 12 columns