Pimpri Chinchwad Education Trusts,

Pimpri Chinchwad College Of Engineering.

Academic Year : 2024-25

Subject : Data Science Laboratory

Sub Teacher : Jaya Dewan

Submitted By :

Arshia Joshi - 123B1F131

Snehal Bandgar - 123B1F132

Kanchan tale - 123B1F137

Mobile Device Usage

1. Project Definition and Objectives

Title : Mobile Device Usage

DatasetLink:https://www.kaggle.com/datasets/valakhorasani/mobile-device-usage-and-user-behavior-dataset

Aim: The aim of this project is to optimize battery performance for users in the highest usage category (user behavior class 5) by identifying device models with high battery drain relative to screen-on time and app usage time. Based on this, the goal is to recommend specific strategies to reduce battery consumption by at least 10%, such as adjusting device settings or optimizing app usage. Additionally, the project will analyze the correlation between app usage time and user behavior class, identifying factors such as the number of apps installed and data usage that influence battery performance.

Goal: Reduce daily battery drain by at least 10% for users in the highest usage category (user behavior class 5) by identifying and recommending optimal battery usage settings or app adjustments.

Objective:

1. Identify Device Models with Highest Battery Drain Relative to Screen-On Time and App Usage Time:

- **Calculate ratios**: Compute ratios like `Battery Drain (mAh/day) / Screen On Time (hours/day)` and `Battery Drain (mAh/day) / App Usage Time (min/day)` for each device model.
- **Sort data**: Identify the top device models with the highest battery drain relative to these metrics.
- **Cluster Analysis**: Use clustering techniques (e.g., k-means) to group devices based on battery usage and user behavior.

2. Suggest Optimization Strategies for Each Device Model:

- **Screen Brightness Optimization**: Recommend adaptive screen brightness settings based on device and user behavior class.
- **Background Process Management**: Suggest controlling background app refresh rates for models with higher battery drain.

- **App Usage Analysis**: Recommend specific app usage patterns or usage restrictions.
- **Power-Saving Modes**: Customize suggestions for power-saving modes specific to each device based on usage trends.

3. Analyze Correlation Between App Usage Time and User Behavior Class:

- **Correlation Coefficient**: Calculate the Pearson correlation coefficient between `App Usage Time (min/day)` and `User Behavior Class` to see if higher app usage is associated with higher behavior class values.
- **Scatter Plot and Regression Analysis**: Plot a scatter plot and fit a regression line to visualize the relationship.
- **Interpret Underlying Factors**:
    - **Number of Apps Installed**: Examine if the number of apps correlates with higher app usage and impacts user behavior class.
    - **Data Usage Analysis**: Explore if high data usage is linked to higher app usage and behavior class.

2. Data Collection and Preparation

1.Importing the dataset(CSV)

```
#importing the data
import pandas as pd
data=pd.read_csv("C:/Users/a2z/OneDrive/Desktop/snehal/Data Science/Mini project/miniprojectdataset.csv")

#gives the first five data
data.head()
```

	User ID	Device Model	Operating System	App Usage Time (min/day)	Screen On Time (hours/day)	Battery Drain (mAh/day)	Number of Apps Installed	Data Usage (MB/day)	Age	Gender	User Behavior Class
0	1	Google Pixel 5	Android	393	6.4	1872	67	1122	40	Male	4
1	2	OnePlus 9	Android	268	4.7	1331	42	944	47	Female	3
2	3	Xiaomi Mi 11	Android	154	4.0	761	32	322	42	Male	2
3	4	Google Pixel 5	Android	239	4.8	1676	56	871	20	Male	3
4	5	iPhone 12	iOS	187	4.3	1367	58	988	31	Female	3

```
#give the last 5 data
data.tail()
```

	User ID	Device Model	Operating System	App Usage Time (min/day)	Screen On Time (hours/day)	Battery Drain (mAh/day)	Number of Apps Installed	Data Usage (MB/day)	Age	Gender	User Behavior Class
695	696	iPhone 12	iOS	92	3.9	1082	26	381	22	Male	2
696	697	Xiaomi Mi 11	Android	316	6.8	1965	68	1201	59	Male	4
697	698	Google Pixel 5	Android	99	3.1	942	22	457	50	Female	2
698	699	Samsung Galaxy S21	Android	62	1.7	431	13	224	44	Male	1
699	700	OnePlus 9	Android	212	5.4	1306	49	828	23	Female	3

2.Data Preprocessing

```
#gives the how many rows and columns are there
data.shape
```

```
(700, 11)
```

```
#give the total cells in data
data.size
```

```
7700
```

```
#name of columns
data.columns
```

```
Index(['User ID', 'Device Model', 'Operating System',
       'App Usage Time (min/day)', 'Screen On Time (hours/day)',
       'Battery Drain (mAh/day)', 'Number of Apps Installed',
       'Data Usage (MB/day)', 'Age', 'Gender', 'User Behavior Class'],
      dtype='object')
```

```
#gives the data type of columns
data.dtypes
```

```
User ID                         int64
Device Model                   object
Operating System               object
App Usage Time (min/day)        int64
Screen On Time (hours/day)    float64
Battery Drain (mAh/day)         int64
Number of Apps Installed        int64
Data Usage (MB/day)             int64
Age                             int64
Gender                         object
User Behavior Class             int64
dtype: object
```

```
#gives all values of columns
data.values
```

```
array([[1, 'Google Pixel 5', 'Android', ..., 40, 'Male', 4],
       [2, 'OnePlus 9', 'Android', ..., 47, 'Female', 3],
       [3, 'Xiaomi Mi 11', 'Android', ..., 42, 'Male', 2],
       ...,
       [698, 'Google Pixel 5', 'Android', ..., 50, 'Female', 2],
       [699, 'Samsung Galaxy S21', 'Android', ..., 44, 'Male', 1],
       [700, 'OnePlus 9', 'Android', ..., 23, 'Female', 3]], dtype=object)
```

```
#gives start ,stop and step index
data.index
```

```
RangeIndex(start=0, stop=700, step=1)
```

The dataset provides insights into mobile device usage across various models (e.g., Google Pixel, OnePlus, Xiaomi, iPhone, Samsung) and operating systems (Android, iOS). Key metrics include app usage time, screen-on time, battery drain, number of apps installed, and data usage. These factors, alongside user demographics (age, gender) and behavior class, offer a rich basis for analysis. By examining correlations between usage time, battery drain, and screen-on time, we can uncover patterns in device efficiency and consumption habits, while demographic and behavioral segmentation allows us to profile different user types effectively.

A] **Data Preparation:**

1.Handle missing values

```
import pandas as pd          # Import pandas for data manipulation and analysis
import numpy as np           # Import numpy for numerical computations
import matplotlib.pyplot as plt  # Import matplotlib for plotting
import seaborn as sns        # Import seaborn for enhanced statistical plots
%matplotlib inline           # Display matplotlib plots inline in Jupyter notebooks
```

```
data=pd.read_csv("C:/Users/a2z/OneDrive/Desktop/snehal/Data Science/Mini project/miniprojectdataset.csv")
```

```
# to know unique values
data.nunique()
```

```
User ID                        700
Device Model                     5
Operating System                 2
App Usage Time (min/day)       387
Screen On Time (hours/day)     108
Battery Drain (mAh/day)        628
Number of Apps Installed        86
Data Usage (MB/day)            585
Age                             42
Gender                           2
User Behavior Class              5
dtype: int64
```

```
data.loc[data['User Behavior Class'] == 4, 'User Behavior Class'] = np.nan  # Replaces values of 4 in 'Single Epithelial Cell Size' with NaN
```

```
data['User Behavior Class'].unique()  # Lists the unique values in the 'User Behavior Class' column
```

```
array([nan,  3.,  2.,  5.,  1.])
```

```
Operating_System= data['Operating System'].unique()

# Print the unique values
print(Operating_System)
```

```
['Android' 'iOS']
```

```
# Count occurrences of each unique value in the Device Model column
Device_Model= data['Device Model'].value_counts()

# Print the counts
print(Device_Model)
```

```
Device Model
Xiaomi Mi 11          146
iPhone 12             146
Google Pixel 5        142
OnePlus 9             133
Samsung Galaxy S21    133
Name: count, dtype: int64
```

```
# Replace values of User Behavior Class equal to 0 with NaN
data.loc[data['User Behavior Class'] == 0, 'User Behavior Class'] = np.nan
```

```
data.isna().sum()
```

```
User ID                       0
Device Model                  0
Operating System              0
App Usage Time (min/day)      0
Screen On Time (hours/day)    0
Battery Drain (mAh/day)       0
Number of Apps Installed      0
Data Usage (MB/day)           0
Age                           0
Gender                        0
User Behavior Class         139
dtype: int64
```

```
data.iloc[0]  # Retrieves the first row of the DataFrame
```

```
User ID                             1
Device Model                Google Pixel 5
Operating System                 Android
App Usage Time (min/day)             393
Screen On Time (hours/day)           6.4
Battery Drain (mAh/day)             1872
Number of Apps Installed              67
Data Usage (MB/day)                 1122
Age                                   40
Gender                              Male
User Behavior Class                  NaN
Name: 0, dtype: object
```

```
print(data.mode())  # Displays the mode (most frequent value) for each column in the DataFrame
print(data.mode().iloc[0])  # Displays the first mode value for each column
```

```
data = data.fillna(data.mode().iloc[0])  # Fills NaN values with the mode (most frequent value) of each column
print(data.isnull().sum())  # Displays the count of missing (NaN) values in each column after filling
```

```
User ID                       0
Device Model                  0
Operating System              0
App Usage Time (min/day)      0
Screen On Time (hours/day)    0
Battery Drain (mAh/day)       0
Number of Apps Installed      0
Data Usage (MB/day)           0
Age                           0
Gender                        0
User Behavior Class           0
dtype: int64
```

Insights:

1.Handling Missing and Invalid Data

- Invalid or missing values were replaced with `NaN` (e.g., in the `User Behavior Class` column), improving the dataset's accuracy and making it ready for meaningful analysis without skewed results.

2. Focused Subset Analysis

- We filtered the dataset for specific user behavior classes, allowing us to analyze and uncover patterns specific to targeted groups, offering deeper insights into user actions and preferences.

3. Data Cleaning and Trend Identification

- By removing duplicates and calculating the mode for key columns, we ensured the data's integrity and identified the most common behaviors, device models, and usage patterns, setting the stage for predictive analysis or feature optimization.

3. Exploratory Data Analysis (EDA)

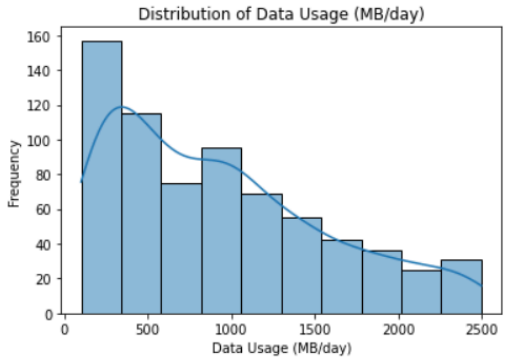
Data Visualization:

a)Histogram

In [16]:
```
# Plotting the histogram with KDE (Kernel Density Estimate)
sns.histplot(data=data['Data Usage (MB/day)'], bins=10, kde=True)

# Adding labels and title
plt.xlabel('Data Usage (MB/day)')
plt.ylabel('Frequency')
plt.title('Distribution of Data Usage (MB/day)')

# Show plot
plt.show()
```



Insights:

1. Distribution Shape: The data distribution seems to be right-skewed, with a higher concentration of data usage values on the lower end (near 0-500 MB/day). The frequency gradually decreases as data usage increases, indicating that most users consume a smaller amount of data per day.

2. Peak Frequency: The highest frequency, around 160, occurs in the first data usage range (0-500 MB/day), suggesting that this range is the most common for data usage.

3. Tail Behavior: The histogram has a long tail extending towards higher data usage (up to 2500 MB/day), implying that while most users use less data, there are some instances of high data consumption.

4. Trend Observation: As data usage increases, the frequency declines, with fewer occurrences of high data usage. This might suggest that only a small fraction of users or scenarios lead to very high data consumption.

BarPlot:

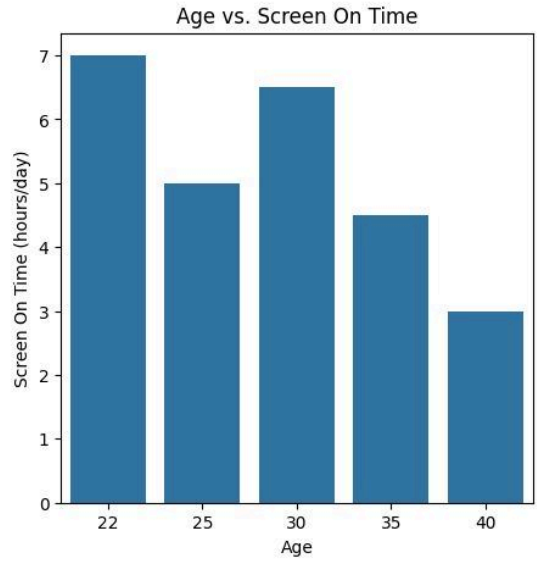
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Set figure size
plt.figure(figsize=(5, 5))

# Create the bar plot
sns.barplot(x='Age', y='Screen On Time (hours/day)', data=data)

# Add title
plt.title('Age vs. Screen On Time')

# Show the plot
plt.show()
```



The "Age vs. Screen On Time" chart shows that screen-on time decreases with age:

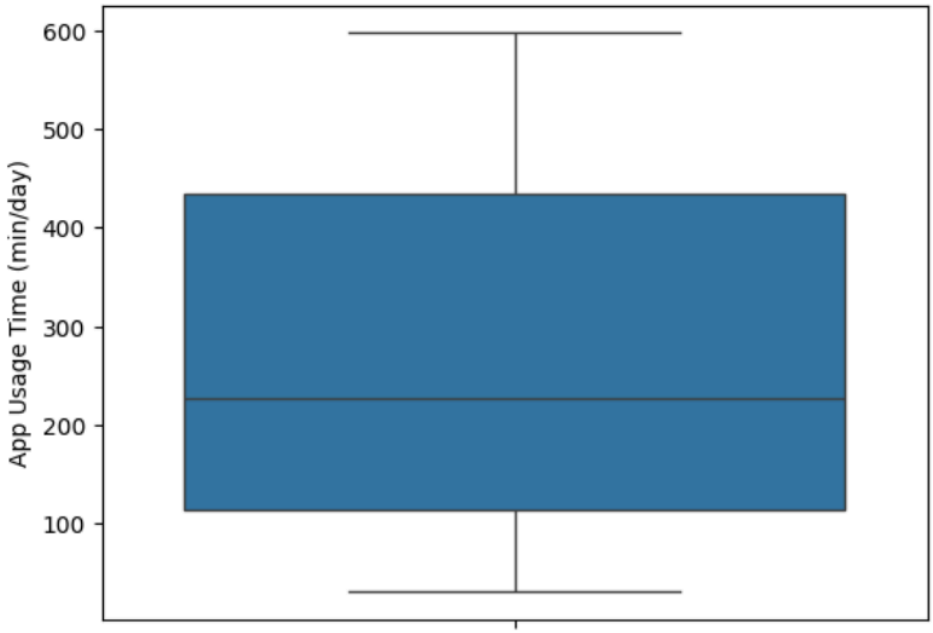
- Younger Users (Age 22): Highest screen time (~7 hours/day), likely due to social media, entertainment, and gaming.
- Mid-Aged Users (Ages 25-30): Moderate screen time (4-6 hours/day), with shifting life priorities like career and family.
- Older Users (Ages 35-40): Screen time drops to around 4 hours/day, possibly due to less reliance on technology for socializing and entertainment.

This trend suggests younger users have higher battery drain, app usage, and data consumption, while older users use devices more selectively. These insights could influence app design, marketing, and device optimization.

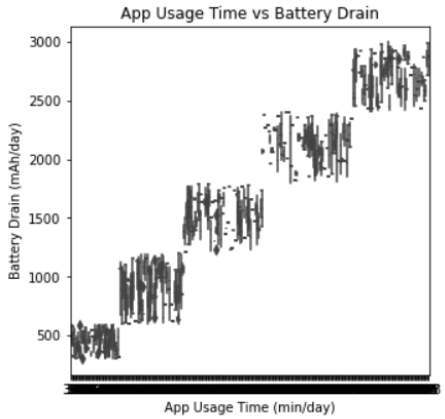
Boxplot

```
import seaborn as sns
import matplotlib.pyplot as plt  # Import this to ensure plots display correctly
sns.boxplot(y=data['App Usage Time (min/day)'])
```

```
<Axes: ylabel='App Usage Time (min/day)'>
```



In [9]:
```
plt.figure(figsize = (5,5))
sns.boxplot(x = data["App Usage Time (min/day)"],y = data["Battery Drain (mAh/day)"])
plt.title("App Usage Time vs Battery Drain")
plt.show()
```



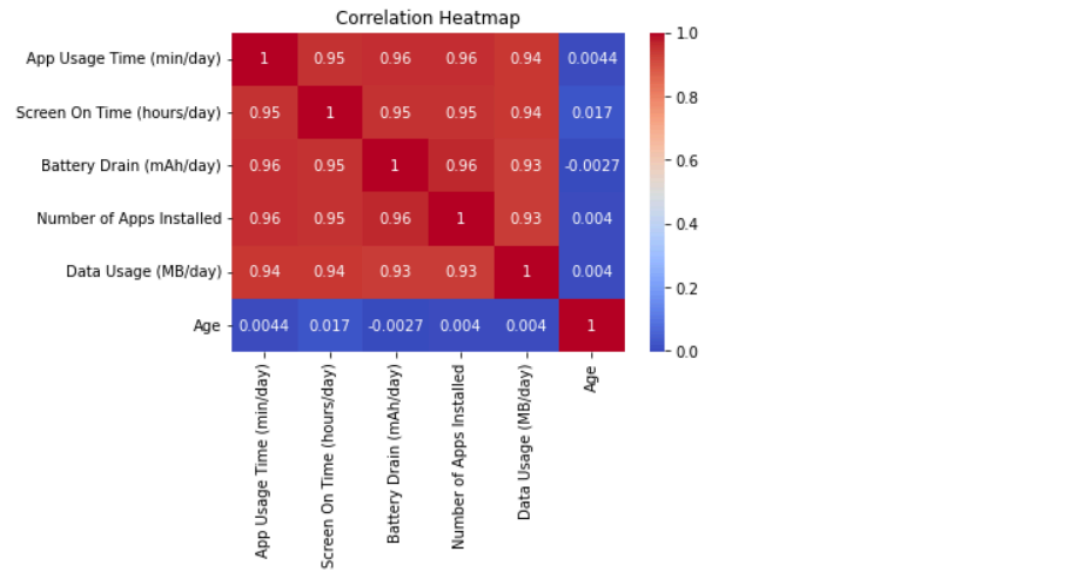
Insights:

1. Positive Correlation: There is a clear upward trend in the data, indicating a positive correlation between app usage time and battery drain. As app usage time increases, battery drain also tends to increase.

2. Step-Like Pattern: The data appears to form clusters or "steps," suggesting that battery drain might be quantized or related to specific app usage thresholds. This pattern could indicate that certain apps or features contribute significantly to battery consumption when used for particular durations.
3. Variability at Higher Usage: The spread of data points becomes broader as app usage time increases. This means that at higher usage durations, battery drain varies more, which could be due to differences in app efficiency, background processes, or other factors affecting battery life.
4. Potential Analysis: Further analysis could include fitting a regression line to quantify the strength of the correlation or clustering the data to identify common app usage and battery drain patterns.

In [7]:
```
#Data Visualization
#Corelation heatmap
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# Assuming 'data' is your DataFrame
corr = data[['App Usage Time (min/day)', 'Screen On Time (hours/day)',
             'Battery Drain (mAh/day)', 'Number of Apps Installed',
             'Data Usage (MB/day)', 'Age']].corr()

# Plot the heatmap
plt.figure(figsize=(6,4))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



The correlation heatmap reveals several insights into the relationships between different features in your dataset:

The correlation heatmap highlights several key patterns:

1. **Strong Positive Correlations**: Features like App Usage Time, Screen On Time, Battery Drain, Number of Apps Installed, and Data Usage are highly correlated (above 0.90). This suggests that increased app usage leads to more screen-on time, higher battery drain, and greater data usage. These features may contribute redundant information, risking **multicollinearity** in predictive models.
2. **Weak Correlation with Age**: Age has little to no correlation with usage metrics, indicating that user behavior (app usage, screen-on time) is not significantly influenced by age.
3. **Multicollinearity Risk**: The strong correlations among key features could cause multicollinearity in regression models, potentially degrading model performance. Techniques like **PCA** or **regularization** (e.g., Ridge or Lasso) can help address this.
4. **Feature Selection**: Given the high correlations, selecting key features like App Usage Time or Screen On Time can simplify the model while retaining predictive power and reducing redundancy.

Overall, the heatmap reveals strong relationships among usage metrics, with age having minimal impact, offering valuable insights for refining feature selection and mitigating multicollinearity.

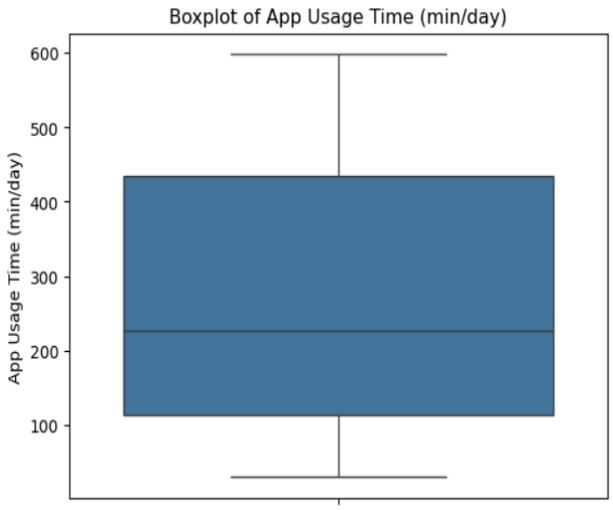
Detect Outliers :

Detect outliers and anomalies

[5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = 'C:/Users/kanch/Downloads/miniprojectdataset.csv'
data = pd.read_csv(file_path)

# Plot a boxplot to visualize potential outliers
sns.boxplot(data['App Usage Time (min/day)'])
plt.title("Boxplot of App Usage Time (min/day)")
plt.show()
```



[7]:
```
# 1. Calculate the z-scores for the 'App Usage Time (min/day)' column
z_scores = np.abs((data['App Usage Time (min/day)'] - data['App Usage Time (min/day)'].mean()) / data['App Usage Time (min/day)'].std())
print("Z-scores for 'App Usage Time (min/day)':\n", z_scores)
# 2. Set the threshold for outliers
threshold = 1  # Adjust this as needed based on your tolerance for outliers

# 3. Filter rows where the z-score is below the threshold
outliers_removed = (z_scores < threshold)

# 4. Apply the filter to the original data
cleaned_data = data[outliers_removed]

# Print results
print("\nOriginal data:\n", data)
print("\nData with Outliers Removed:\n", cleaned_data)
```

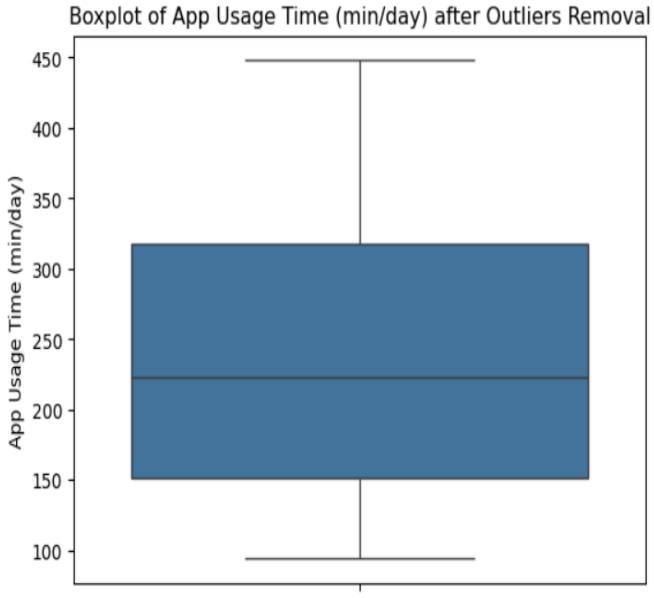
```
Z-scores for 'App Usage Time (min/day)':
 0      0.687764
1      0.017656
2      0.660998
3      0.181313
4      0.474768
         ...
695    1.010887
696    0.253226
697    0.971383
698    1.180187
699    0.333684
Name: App Usage Time (min/day), Length: 700, dtype: float64

Original data:
      User ID        Device Model Operating System  App Usage Time (min/day)  \
0         1      Google Pixel 5          Android                       393
1         2         OnePlus 9          Android                       268
```

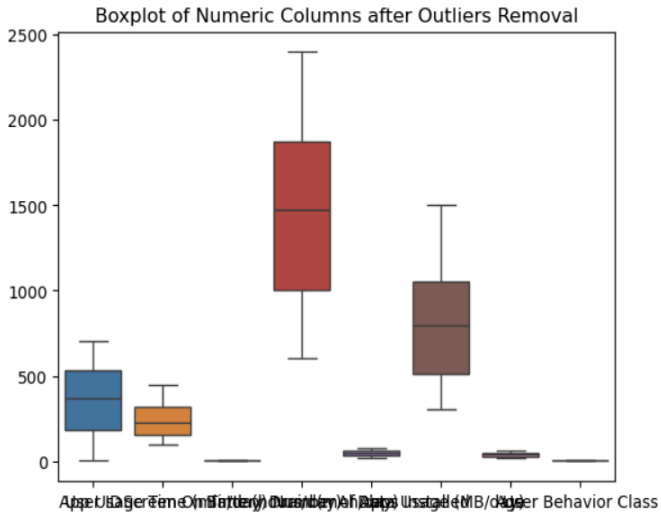
```
Data with Outliers Removed:
      User ID    Device Model Operating System  App Usage Time (min/day)  \
0           1  Google Pixel 5          Android                       393
1           2       OnePlus 9          Android                       268
2           3    Xiaomi Mi 11          Android                       154
3           4  Google Pixel 5          Android                       239
4           5       iPhone 12              iOS                       187
..        ...             ...              ...                       ...
691       692       iPhone 12              iOS                       178
692       693    Xiaomi Mi 11          Android                       378
696       697    Xiaomi Mi 11          Android                       316
697       698  Google Pixel 5          Android                        99
699       700       OnePlus 9          Android                       212

     Screen On Time (hours/day)  Battery Drain (mAh/day)  \
0                           6.4                     1872
1                           4.7                     1331
2                           4.0                      761
```

[9]:
```
# Plotting the boxplot for a specific column (e.g., 'App Usage Time (min/day)')
sns.boxplot(data=cleaned_data['App Usage Time (min/day)'])
plt.title("Boxplot of App Usage Time (min/day) after Outliers Removal")
plt.show()
```



[11]:
```
# Boxplot for all numeric columns in cleaned_data
sns.boxplot(data=cleaned_data.select_dtypes(include=[np.number]))
plt.title("Boxplot of Numeric Columns after Outliers Removal")
plt.show()
```



Insights:

1. Outlier Detection: The z-score method was used to identify extreme values in the `App Usage Time (min/day)` column, allowing for a quantitative measure of outliers.

2. Data Cleaning: Outliers (defined as data points with z-scores greater than 1 or less than -1) were removed to ensure the dataset is free from extreme values that could distort analysis.

3. Improved Data Distribution: After outlier removal, the distribution of the `App Usage Time (min/day)` becomes more representative of typical usage patterns, ensuring more reliable analysis.

4. Threshold Flexibility: The z-score threshold (set at 1) can be adjusted to detect more or fewer outliers, offering flexibility based on the tolerance for extreme values in the dataset.

5. Comprehensive Cleaning: Boxplots for all numeric columns were used to visually confirm that outliers were removed from the entire dataset, ensuring consistent data quality across features.

- **Hypothesis testing:**

[3]:
```
import pandas as pd
import scipy.stats as stats
import numpy as np

# Load the CSV file
file_path = 'C:/Users/kanch/Downloads/miniprojectdataset.csv'
data = pd.read_csv(file_path)

# Hypothesis testing function
def hypothesis_test(sample, population_mean, alpha, known_population_std=None):
    # Calculate sample mean and sample standard deviation
    sample_mean = np.mean(sample)
    sample_std = np.std(sample, ddof=1)
    sample_size = len(sample)

    # Check if population standard deviation is known or unknown
    if known_population_std is not None:
        # Z-test (known population standard deviation)
        z_stat = (sample_mean - population_mean) / (known_population_std / np.sqrt(sample_size))
        p_value = 2 * (1 - stats.norm.cdf(abs(z_stat)))  # two-tailed p-value
        test_stat = z_stat
        test_type = "Z-test"
    else:
        # T-test (unknown population standard deviation)
        t_stat = (sample_mean - population_mean) / (sample_std / np.sqrt(sample_size))
        p_value = 2 * (1 - stats.t.cdf(abs(t_stat), df=sample_size - 1))  # two-tailed p-value
        test_stat = t_stat
        test_type = "T-test"

    # Print details
    print(f"Sample mean: {sample_mean}")
    print(f"Population mean: {population_mean}")
    print(f"{test_type} statistic: {test_stat}")
    print(f"p-value: {p_value}")

    # Compare p-value with alpha
    if p_value < alpha:
        print(f"Reject the null hypothesis (p-value < {alpha})")
    else:
        print(f"Fail to reject the null hypothesis (p-value >= {alpha})")

# Define parameters for hypothesis testing
population_mean = 300  # Hypothesized population mean for App Usage Time (min/day)
alpha = 0.05  # Significance level

# Extract the 'App Usage Time (min/day)' column as the sample data
sample_data = data['App Usage Time (min/day)']

# Perform hypothesis test
hypothesis_test(sample_data, population_mean, alpha, known_population_std=None)
```

```
Sample mean: 271.12857142857143
Population mean: 300
T-test statistic: -4.310769879567371
p-value: 1.8605787589276446e-05
Reject the null hypothesis (p-value < 0.05)
```

Key Insights from the Hypothesis Testing

1. Hypothesis Test Type:

- The code performs a T-test (since the population standard deviation is unknown) to compare the sample mean of `App Usage Time (min/day)` with a hypothesized population mean of 300 minutes.

2. Test Statistic Calculation:
- The T-statistic iis calculated by comparing the sample mean to the population mean, adjusted for sample size and standard deviation. This tells us how far the sample mean is from the population mean in terms of standard errors.

3. P-value:
- The p-value is computed to assess the likelihood of observing the data if the null hypothesis (mean = 300) is true. A p-value below 0.05 suggests rejecting the null hypothesis.

4. Decision on Null Hypothesis:
- If the p-value is less than the significance level (0.05), we reject the null hypothesis, indicating the sample mean is significantly different from 300 minutes. If the p-value is greater than 0.05, we fail to reject the null hypothesis.

5. Outcome:
- The code helps determine whether the average app usage time differs significantly from 300 minutes, based on the hypothesis test results.

4. Modeling and Machine Learning

1.Linear Regression

```
#Linear Rgeression
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metric import mean_absolute_error,mean_sqaured_error,r2_score
```

```
# First, remove any extra whitespace from column names
data.columns = data.columns.str.strip()

# Now you can select features and target without extra whitespace issues
X = data.drop(columns=['Screen On Time (hours/day)'])  # Features
y = data['Screen On Time (hours/day)']  # Target
print(y)
```

```
0      6.4
1      4.7
2      4.0
3      4.8
4      4.3
      ...
695    3.9
696    6.8
697    3.1
698    1.7
699    5.4
Name: Screen On Time (hours/day), Length: 700, dtype: float64
```

```
from sklearn.model_selection import train_test_split

# Splitting the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Ensure column names have no leading/trailing whitespace
X_train.columns = X_train.columns.str.strip()
X_test.columns = X_test.columns.str.strip()

# Define categorical columns without extra whitespace
categorical_columns = ['Device Model', 'Operating System', 'Gender']

# One-hot encoding for categorical variables
for column in categorical_columns:
    if column in X_train.columns:
        X_train = pd.get_dummies(X_train, columns=[column], drop_first=True)
        X_test = pd.get_dummies(X_test, columns=[column], drop_first=True)

# Align columns in case of any missing categories
X_train, X_test = X_train.align(X_test, join='left', axis=1, fill_value=0)

# Store feature names before scaling
feature_names = X_train.columns

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)  # Converts to NumPy array
X_test = scaler.transform(X_test)  # Converts to NumPy array

# At this point, feature_names is still available as a list
print(feature_names)
```

```
Index(['User ID', 'App Usage Time (min/day)', 'Battery Drain (mAh/day)',
       'Number of Apps Installed', 'Data Usage (MB/day)', 'Age',
       'User Behavior Class', 'Gender_Male', 'Device Model_OnePlus 9',
       'Device Model_Samsung Galaxy S21', 'Device Model_Xiaomi Mi 11',
       'Device Model_iPhone 12', 'Device Model_1', 'Device Model_2',
       'Device Model_3', 'Device Model_4', 'Operating System_iOS'],
      dtype='object')
```

5. Evaluation and Validation

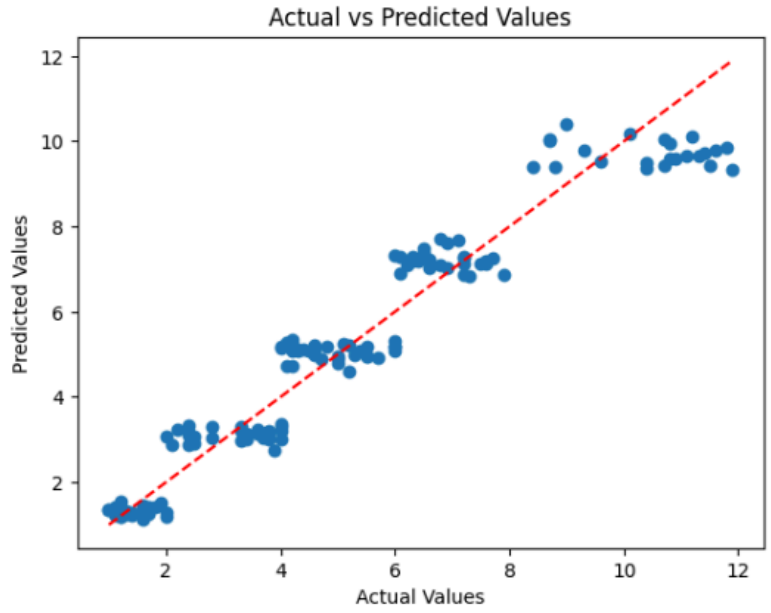
```python
from sklearn.linear_model import LinearRegression

lin_reg=LinearRegression()
lin_reg.fit(X_train,y_train)
y_pred=lin_reg.predict(X_test)
```

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Assuming y_test and y_pred are defined
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
# Print the evaluation metrics
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R^2 Score:", r2)
```

```
Mean Absolute Error: 0.6445792137235598
Mean Squared Error: 0.6251732374662121
R^2 Score: 0.9303228357085748
```

```python
import matplotlib.pyplot as plt
# Optionally, visualize predictions vs actual values
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
# Plotting the line for perfect predictions
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],color='red', linestyle='--')
plt.show()
```



Your linear regression model achieved a solid performance, with a high R2R^2R2 score of **0.93**, indicating that the model explains 93% of the variance in "Screen On Time (hours/day)." Here are the main insights from the evaluation metrics and model performance:

1. **Mean Absolute Error (MAE):** The MAE is **0.64**, suggesting that, on average, the model's predictions deviate from the actual screen-on time by about 0.64 hours (roughly 38 minutes) per day. This low MAE reflects a good level of precision.
2. **Mean Squared Error (MSE):** The MSE is **0.63**, which is relatively low. Since MSE penalizes larger errors more heavily, this value indicates that large deviations are rare, meaning the model generally makes close predictions.

3. **R^2 Score:** With an R2R^2R2 score of **0.93**, the model is highly effective at capturing the relationship between the input features and screen-on time. This score shows that most of the variability in screen-on time is accounted for by the selected features, indicating a well-fitted model.
4. **Visual Analysis (Actual vs. Predicted Plot):** The scatter plot of actual vs. predicted values, with a red line showing the ideal prediction line, helps us see how well the model's predictions align with reality. Points close to the red line indicate accurate predictions, and overall closeness would imply minimal error across test data.

Recommendations

- **Model Refinement:** Although the model performs well, consider tuning it further or trying regularization techniques (like Ridge or Lasso regression) to potentially enhance stability and handle any potential multicollinearity.
- **Feature Importance Analysis:** Evaluating feature coefficients can help determine which factors most influence screen-on time, providing actionable insights for improving power efficiency or user engagement.

Random Forest:

[21]:
```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
 ↪accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import KBinsDiscretizer

# Assuming the dataset is already loaded as 'data'
# Example:
```

```
data = pd.read_csv("C:/Users/a2z/OneDrive/Desktop/snehal/Data Science/Mini␣
 ↪project/miniprojectdataset.csv")

# Preprocessing - You may need to convert categorical variables like 'Gender'␣
 ↪and 'Device Model' to numerical values
data['Gender'] = data['Gender'].map({'Male': 1, 'Female': 0})

# Select features and target variable
X = data[['App Usage Time (min/day)', 'Screen On Time (hours/day)',
          'Number of Apps Installed', 'Data Usage (MB/day)', 'Age', 'Gender']]

y = data['Battery Drain (mAh/day)']

# --- Random Forest Regression ---
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create and train the RandomForestRegressor model
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

# Evaluate the regression model (R², MAE, MSE)
y_pred_regressor = rf_regressor.predict(X_test)
r2_regressor = rf_regressor.score(X_train, y_train)  # R^2 score for training␣
 ↪data
mae = mean_absolute_error(y_test, y_pred_regressor)
mse = mean_squared_error(y_test, y_pred_regressor)

print(f"Random Forest Regression R^2: {r2_regressor:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")

# --- Random Forest Classification ---
# Convert 'Battery Drain' to categorical values (Low, Medium, High) using␣
 ↪KBinsDiscretizer
# This will classify the battery drain into 3 categories
kbd = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
y_class = kbd.fit_transform(y.values.reshape(-1, 1)).flatten()

# Split data again for classification task
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X,␣
 ↪y_class, test_size=0.2, random_state=42)

# Create and train the RandomForestClassifier model
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
rf_classifier.fit(X_train_class, y_train_class)

# Predict and evaluate the classification model
y_pred_class = rf_classifier.predict(X_test_class)
accuracy = accuracy_score(y_test_class, y_pred_class)
conf_matrix = confusion_matrix(y_test_class, y_pred_class)
class_report = classification_report(y_test_class, y_pred_class)

print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Random Forest Regression R^2: 0.9936
Mean Absolute Error (MAE): 147.0507
Mean Squared Error (MSE): 30646.8740
Accuracy: 0.8929
Confusion Matrix:
[[56  0  0]
 [ 0 39  9]
 [ 0  6 30]]
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        56
         1.0       0.87      0.81      0.84        48
         2.0       0.77      0.83      0.80        36

    accuracy                           0.89       140
   macro avg       0.88      0.88      0.88       140
weighted avg       0.89      0.89      0.89       140
```

Insights :

- The regression model shows high predictive power, with a very high R^2 score. However, the MAE and MSE suggest there may be some instances with substantial prediction errors.
- The classification model achieves good accuracy, but performance varies across categories, with the model being more accurate for low battery drain levels than medium or high.
- If better performance on classes 1 and 2 is desired, consider techniques like class balancing, tuning the model parameters, or experimenting with other classifiers.

CONCLUSION:

Data Preparation and Cleaning:

1. Missing and invalid data were handled by replacing them with `NaN`, ensuring dataset integrity.
2. Filtering by behavior classes highlighted targeted usage patterns.
3. Duplicate removal and mode calculations improved data quality, enabling trend analysis.

Data Distribution:

1. Data usage is right-skewed, with most users consuming 0-500 MB/day.
2. Few users exhibit very high data consumption, forming a long tail.

Age vs. Screen-On Time:

1. Younger users (age 22) average 7 hours/day, driven by social engagement and entertainment.
2. Mid-aged users (25-30) show moderate screen time (4-6 hours/day) due to shifting priorities.
3. Older users (35-40) use devices less, averaging around 4 hours/day.

Correlation and Feature Analysis:

1. Strong correlations exist among app usage, screen-on time, battery drain, etc., suggesting multicollinearity.
2. Age has low correlation with usage metrics, indicating it may not be a strong predictor.

Outlier Handling:

1. The z-score method identified and removed extreme values, improving data distribution.
2. Boxplots confirmed outlier removal across all numeric columns.

Statistical Testing:

1. A T-test compared sample means to a hypothesized average, guiding decisions on significant differences.

Model Evaluation:

1. Linear regression achieved an R^2 score of 0.93, indicating strong predictive power.
2. MAE (0.64) and MSE (0.63) suggest accurate and reliable predictions.
3. Potential for model tuning or regularization to address multicollinearity.

Model Recommendations:

1. Consider regularization (Ridge/Lasso) for stability.
2. Analyze feature importance to optimize app design and user engagement strategies.