Student Name: Aseel Alwesabi

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

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (    confusion_matrix,    accuracy_score,    precision_score,    recall_score,    f1_score,    classification_report )

from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
```

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving user_behavior_dataset.csv to user_behavior_dataset.csv
User uploaded file "user_behavior_dataset.csv" with length 38878 bytes
```

```
df = pd.read_csv("user_behavior_dataset.csv")
df.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

```
df['Device Model'].value_counts()
```

Device Model	count
Xiaomi Mi 11	146
iPhone 12	146
Google Pixel 5	142
OnePlus 9	133
Samsung Galaxy S21	133

dtype: int64

Getting the count of missing values

```
df.isnull().sum()
```

	0
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

Deleting columns that have 30% missing values

```
missing_percentage =df.isnull().mean()*100
Threshold =30


columns_to_drop = missing_percentage[missing_percentage > Threshold].index

df=df.drop(columns=columns_to_drop)

print(f"Columns dropped:{list(columns_to_drop)}")
print (f"Remaining columns: {df.columns}")
```

```
Columns dropped:[]
Remaining 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')
```

Detect Outliers

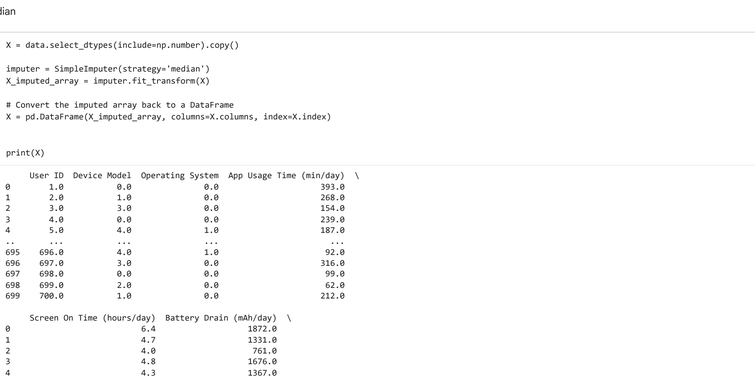
```
data=df.copy()
def detect_outliers_iqr(column):
  Q1 = column.quantile(0.25)
  Q3 = column.quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
# Return a boolean series indicating if the value is an outlier
  return (column < lower_bound) | (column > upper_bound)

# Apply the IQR method to each numeric feature in the DataFrame
numeric_cols = data.select_dtypes(include=np.number).columns
for col in numeric_cols:
  data[f'{col}_Outlier'] = detect_outliers_iqr(data[col])

# Filter the DataFrame to display rows where any outlier flag is True
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]
```

```
# Display the filtered DataFrame with outliers
outliers_only
```

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	User ID_Outlier	App Usage Time (min/day)_Outlier	Screen On Time (hours/day)_Outlier	Battery Drain (mAh/day)_Outlier	Number of Apps Installed_Outlier	Data Usage (MB/day)_Outlier	Age_Outlier	User Behavior Class_Outlier

Visualize Outliers

```
plt.figure(figsize=(12, 6))
for i, col in enumerate(df.columns[:3]): # Only plotting the first 3 features
  plt.subplot(1, 3, i+1) # Create subplots for each feature
  sns.boxplot(x=df[col])
  plt.title(f"Boxplot for {col}")
plt.tight_layout()
plt.show()
```



Handling Outliers

```
data=df.copy()
def replace_outliers_with_nan(column):
  Q1 = column.quantile(0.25)
  Q3 = column.quantile(0.75)
  IQR = Q3 - Q1
  lower_bound = Q1 - 1.5 * IQR
  upper_bound = Q3 + 1.5 * IQR
  # Replace outliers with NaN
  return column.where((column >= lower_bound) & (column <=
  upper_bound), np.nan)
# Replace outliers with NaN for each feature
numeric_cols = data.select_dtypes(include=np.number).columns
for col in numeric_cols:
  data[col] = replace_outliers_with_nan(data[col])
null_counts = data.isnull().sum()
print("Sum of null values in each column:")
print(null_counts)
```

```
Sum of null values in each column:
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
```

Convert categorical features to numerical data

```
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
  le = LabelEncoder()
  data[column] = le.fit_transform(data[column])
  label_encoders[column] = le
```

Median

```
X = data.select_dtypes(include=np.number).copy()

imputer = SimpleImputer(strategy='median')
X_imputed_array = imputer.fit_transform(X)

# Convert the imputed array back to a DataFrame
X = pd.DataFrame(X_imputed_array, columns=X.columns, index=X.index)


print(X)
```

```
     User ID  Device Model  Operating System  App Usage Time (min/day)  \
0        1.0           0.0               0.0                     393.0
1        2.0           1.0               0.0                     268.0
2        3.0           3.0               0.0                     154.0
3        4.0           0.0               0.0                     239.0
4        5.0           4.0               1.0                     187.0
..       ...           ...               ...                       ...
695    696.0           4.0               1.0                      92.0
696    697.0           3.0               0.0                     316.0
697    698.0           0.0               0.0                      99.0
698    699.0           2.0               0.0                      62.0
699    700.0           1.0               0.0                     212.0

     Screen On Time (hours/day)  Battery Drain (mAh/day)  \
0                           6.4                   1872.0
1                           4.7                   1331.0
2                           4.0                    761.0
3                           4.8                   1676.0
4                           4.3                   1367.0
..                          ...                      ...
695                         3.9                   1082.0
696                         6.8                   1965.0
697                         3.1                    942.0
698                         1.7                    431.0
699                         5.4                   1306.0

     Number of Apps Installed  Data Usage (MB/day)   Age  Gender  \
0                        67.0               1122.0  40.0     1.0
1                        42.0                944.0  47.0     0.0
2                        32.0                322.0  42.0     1.0
3                        56.0                871.0  20.0     1.0
4                        58.0                988.0  31.0     0.0
..                        ...                  ...   ...     ...
695                      26.0                381.0  22.0     1.0
696                      68.0               1201.0  59.0     1.0
697                      22.0                457.0  50.0     0.0
698                      13.0                224.0  44.0     1.0
699                      49.0                828.0  23.0     0.0

     User Behavior Class
0                    4.0
1                    3.0
```

```
2                   2.0
3                   3.0
4                   3.0
..                  ...
695                 2.0
696                 4.0
697                 2.0
698                 1.0
699                 3.0

[700 rows x 11 columns]
```

```
df.describe()
```

	User ID	App Usage Time (min/day)	Screen On Time (hours/day)	Battery Drain (mAh/day)	Number of Apps Installed	Data Usage (MB/day)	Age	User Behavior Class
count	700.00000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000
mean	350.50000	271.128571	5.272714	1525.158571	50.681429	929.742857	38.482857	2.990000
std	202.21688	177.199484	3.068584	819.136414	26.943324	640.451729	12.012916	1.401476
min	1.00000	30.000000	1.000000	302.000000	10.000000	102.000000	18.000000	1.000000
25%	175.75000	113.250000	2.500000	722.250000	26.000000	373.000000	28.000000	2.000000
50%	350.50000	227.500000	4.900000	1502.500000	49.000000	823.500000	38.000000	3.000000
75%	525.25000	434.250000	7.400000	2229.500000	74.000000	1341.000000	49.000000	4.000000
max	700.00000	598.000000	12.000000	2993.000000	99.000000	2497.000000	59.000000	5.000000

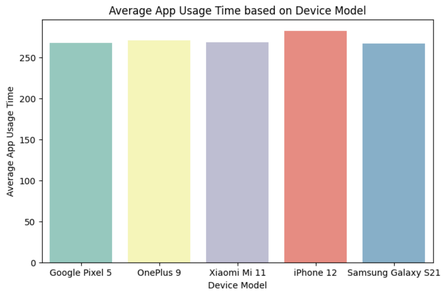
Split data ito training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.30 , random_state= 42 , stratify= y)
```

```
def evaluate_model(model_name, y_true, y_pred): # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average= 'weighted')
    f1 = f1_score(y_true, y_pred, average= 'weighted')
    cm = confusion_matrix(y_true, y_pred)
    # Create a report
    report = classification_report(y_true, y_pred)
    # Output results
    metrics = {
        'Model Name' : model_name,
        'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall': recall,
        'F1 Score': f1,
        'Classification Report': report    }
    # Plot Confusion Matrix
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=np.unique(y_true), yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
    return metrics
```

Average App Usage Time based on Device Model

```
plt.figure(figsize=(8,5))
sns.barplot(
    data=df, x='Device Model',
    y='App Usage Time (min/day)',
    estimator='mean', # default, but explicit is better
    palette='Set3', errorbar=None # removes confidence intervals (optional)
    )
plt.title("Average App Usage Time based on Device Model")
plt.ylabel("Average App Usage Time")
plt.show()
```

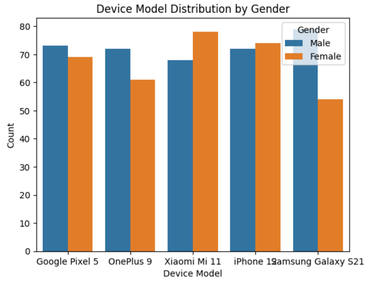


Distribution of Users by Device Model

```
Device_Model = df['Device Model'].value_counts()
plt.figure(figsize=(6,6))
plt.pie( Device_Model, labels=Device_Model.index, autopct='%1.1f%%', startangle=90)
plt.title("Distribution of Users by Device Model")
plt.show()
```

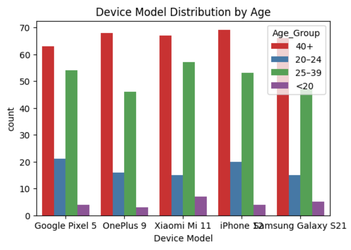

Device Model Distribution by Gender

```
sns.countplot(x='Device Model', hue='Gender', data=df)
plt.title('Device Model Distribution by Gender')
plt.xlabel('Device Model')
plt.ylabel('Count')
plt.show()
```



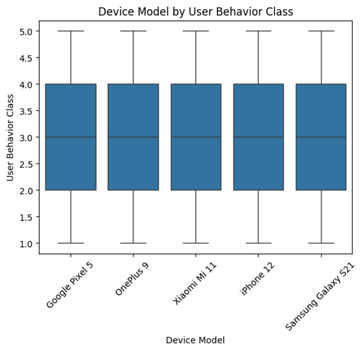
Device Model Distribution by Age

```
def age_group(age):
    if age < 20:
        return "<20"
    elif age < 25:
        return "20-24"
    elif age < 40:
        return "25-39"
    else:
        return "40+"

df["Age_Group"] = df["Age"].apply(age_group)
plt.figure(figsize=(6,4))
sns.countplot(data=df, x='Device Model', hue='Age_Group', palette='Set1')
plt.title("Device Model Distribution by Age")
plt.show()
```



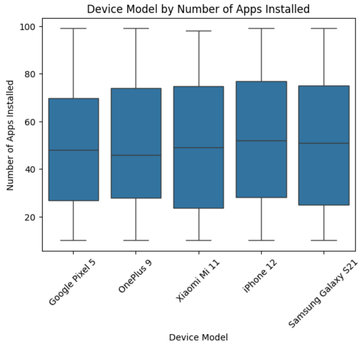
Device Model by User Behavior Class

```
sns.boxplot(x='Device Model', y='User Behavior Class', data=df)
plt.title('Device Model by User Behavior Class')
plt.xticks(rotation=45)
plt.xlabel('Device Model')
plt.ylabel('User Behavior Class')
plt.show()
```



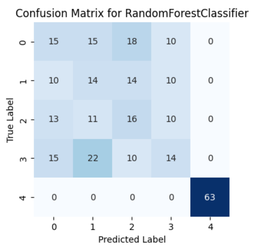
Device Model by Number of Apps Installed

```
sns.boxplot(x='Device Model', y='Number of Apps Installed', data=df)
plt.title('Device Model by Number of Apps Installed')
plt.xticks(rotation=45)
plt.xlabel('Device Model')
plt.ylabel('Number of Apps Installed')
plt.show()
```



Random Forest Classifier

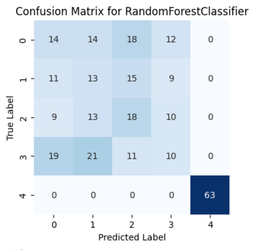
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
# Train the model
rf_classifier.fit(X_train, y_train)
# Make predictions
y_pred = rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
  if key == 'Classification Report':
    print(value)  # Print report separately for better readability
  else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: \n{value}")
```




```
Model Name:
RandomForestClassifier
Accuracy: 0.4357
Precision: 0.4409
Recall: 0.4357
F1 Score: 0.4356
              precision    recall  f1-score   support

           0       0.28      0.26      0.27        58
           1       0.23      0.29      0.25        48
           2       0.28      0.32      0.30        50
           3       0.32      0.23      0.27        61
           4       1.00      1.00      1.00        63

    accuracy                           0.44       280
   macro avg       0.42      0.42      0.42       280
weighted avg       0.44      0.44      0.44       280
```

Random Forest Classifier with grid search

```
from sklearn.model_selection import GridSearchCV
# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4] }
    # Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
  if key == 'Classification Report':
    print(value)
  else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")
# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



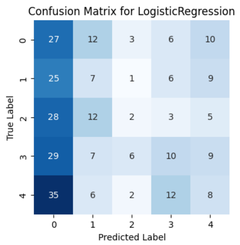

```
Model Name:
RandomForestClassifier
Accuracy: 0.4214
Precision: 0.4212
Recall: 0.4214
F1 Score: 0.4183
              precision    recall  f1-score   support

           0       0.26      0.24      0.25        58
           1       0.21      0.27      0.24        48
           2       0.29      0.36      0.32        50
           3       0.24      0.16      0.20        61
           4       1.00      1.00      1.00        63

    accuracy                           0.42       280
   macro avg       0.40      0.41      0.40       280
weighted avg       0.42      0.42      0.42       280


Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 200}
```

There isn't a huge difference between the Random Forest Classifier with and without Grid

Logistic Regression

```
LR = LogisticRegression()
# Train the model
LR.fit(X_train, y_train)
# Make predictions
y_pred = LR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
  if key == 'Classification Report':
    print(value)
```

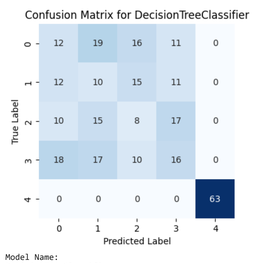
```
  else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")
```




```
Model Name:
LogisticRegression
Accuracy: 0.1929
Precision: 0.1944
Recall: 0.1929
F1 Score: 0.1717
              precision    recall  f1-score   support

           0       0.19      0.47      0.27        58
           1       0.16      0.15      0.15        48
           2       0.14      0.04      0.06        50
           3       0.27      0.16      0.20        61
           4       0.20      0.13      0.15        63

    accuracy                           0.19       280
   macro avg       0.19      0.19      0.17       280
weighted avg       0.19      0.19      0.17       280
```

Decision TreeClassifier

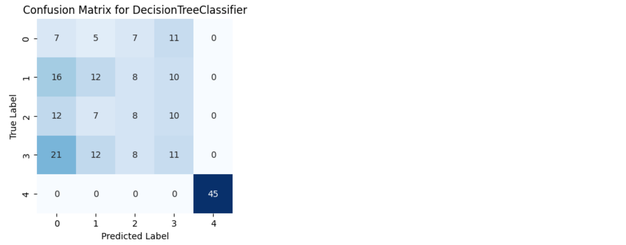
```
DT = DecisionTreeClassifier()
# Train the model
DT.fit(X_train, y_train)
# Make predictions
y_pred = DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test, y_pred) # Print the evaluation results
for key, value in evaluation_results.items():
  if key == 'Classification Report':
    print(value)
  else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")
```




```
Model Name:
DecisionTreeClassifier
Accuracy: 0.3893
Precision: 0.3934
Recall: 0.3893
F1 Score: 0.3906
              precision    recall  f1-score   support

           0       0.23      0.21      0.22        58
           1       0.16      0.21      0.18        48
           2       0.16      0.16      0.16        50
           3       0.29      0.26      0.28        61
           4       1.00      1.00      1.00        63

    accuracy                           0.39       280
   macro avg       0.37      0.37      0.37       280
weighted avg       0.39      0.39      0.39       280
```

Decision TreeClassifier with grid search

```
# Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)
# Define the hyperparameters for grid search
param_grid = {    'criterion': ['gini', 'entropy'],           # The function to measure the quality of a split
                  'max_depth': [None, 10, 20, 30],            # The maximum depth of the tree
                  'min_samples_split': [2, 5, 10],            # The minimum number of samples required to split an internal node
                  'min_samples_leaf': [1, 2, 4],              # The minimum number of samples required to be at a leaf node
                  'max_features': [None, 'sqrt', 'log2']      # The number of features to consider when looking for the best split
                  }
                  # Initialize GridSearchCV
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_DT = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test, y_pred)
                  # Print the evaluation results
for key, value in evaluation_results.items():
  if key == 'Classification Report':
    print(value)
  else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")
    # Print the best hyperparameters found by grid search
    print("\nBest hyperparameters found by GridSearchCV:")
    print(grid_search.best_params_)
```

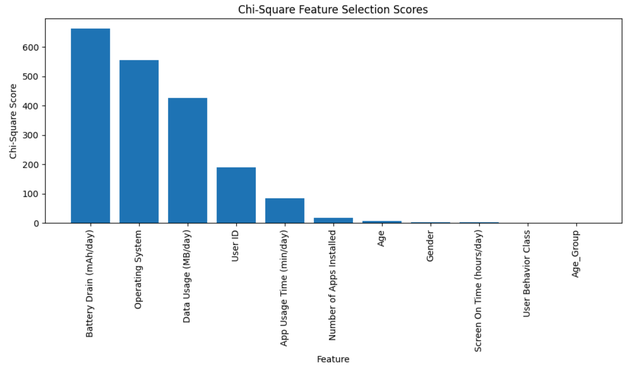



```
Model Name:
DecisionTreeClassifier

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 10}
Accuracy: 0.3952

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 10}
Precision: 0.4155

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 10}
Recall: 0.3952

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 10}
F1 Score: 0.4011

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 10}
              precision    recall  f1-score   support

           0       0.12      0.23      0.16        30
           1       0.33      0.26      0.29        46
           2       0.26      0.22      0.24        37
           3       0.26      0.21      0.23        52
           4       1.00      1.00      1.00        45

    accuracy                           0.40       210
   macro avg       0.40      0.38      0.38       210
weighted avg       0.42      0.40      0.40       210
```

Chi-Square

```
X = df.drop(['Device Model'], axis=1)
y = df['Device Model']
```

```
from sklearn.feature_selection import SelectKBest, chi2
selector = SelectKBest(score_func=chi2, k=10)
selector.fit(X, y)
data = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
data = data.sort_values(by='Score', ascending=False)
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("Chi-Square Score")
plt.title("Chi-Square Feature Selection Scores")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Comparison of models using features selected by the Chi-Square test

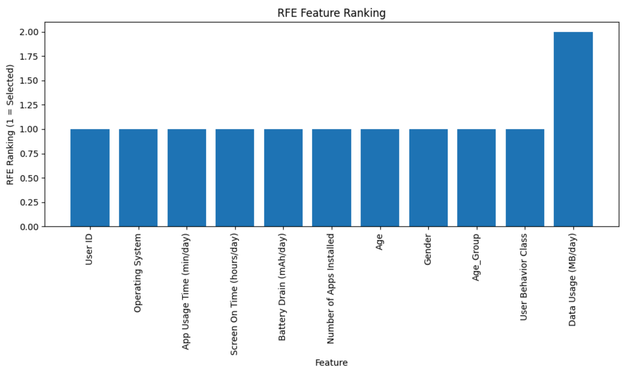
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.40, random_state=40)
k = 10
selector = SelectKBest(score_func=chi2, k=k)
X_train_k = selector.fit_transform(X_train, y_train)
X_test_k = selector.transform(X_test)
selected_features = X.columns[selector.get_support(indices=True)]
print("Selected Features:", list(selected_features))

models = { "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
          "Decision Tree": DecisionTreeClassifier(random_state=42),
           "SVM": SVC(kernel='rbf', random_state=42)}
metrics = []

for name, model in models.items():
  model.fit(X_train_k, y_train)
  y_pred = model.predict(X_test_k)
  print(f"\n==== Classification Report: {name} =====")
  print(classification_report(y_test, y_pred))

  metrics.append({
      "Model": name,
      "Accuracy": accuracy_score(y_test, y_pred),
      "Precision": precision_score(y_test, y_pred,
                                   average='weighted'),
      "Recall": recall_score(y_test, y_pred, average='weighted') })


  results_df = pd.DataFrame(metrics)
  print("\nModel Performance:")
  print(results_df)
```

```
Selected Features: ['User ID', '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']

===== Classification Report: Random Forest =====
              precision    recall  f1-score   support

           0       0.27      0.26      0.26        58
           1       0.23      0.29      0.26        48
           2       0.30      0.30      0.30        50
           3       0.26      0.21      0.23        61
```

```
           4       1.00      1.00      1.00        63

    accuracy                           0.43       280
   macro avg       0.41      0.41      0.41       280
weighted avg       0.43      0.43      0.43       280


Model Performance:
           Model  Accuracy  Precision    Recall
0  Random Forest  0.428571   0.430043  0.428571

===== Classification Report: Decision Tree =====
              precision    recall  f1-score   support

           0       0.24      0.21      0.22        58
           1       0.16      0.21      0.18        48
           2       0.15      0.14      0.14        50
           3       0.29      0.28      0.28        61
           4       1.00      1.00      1.00        63

    accuracy                           0.39       280
   macro avg       0.37      0.37      0.37       280
weighted avg       0.39      0.39      0.39       280


Model Performance:
           Model  Accuracy  Precision    Recall
0  Random Forest  0.428571   0.430043  0.428571
1  Decision Tree  0.389286   0.392747  0.389286

===== Classification Report: SVM =====
              precision    recall  f1-score   support

           0       0.19      0.55      0.28        58
           1       0.00      0.00      0.00        48
           2       0.00      0.00      0.00        50
           3       0.18      0.11      0.14        61
           4       0.21      0.24      0.22        63

    accuracy                           0.19       280
   macro avg       0.12      0.18      0.13       280
weighted avg       0.12      0.19      0.14       280


Model Performance:
           Model  Accuracy  Precision    Recall
0  Random Forest  0.428571   0.430043  0.428571
1  Decision Tree  0.389286   0.392747  0.389286
2            SVM  0.192857   0.124791  0.192857
```

Recursive Feature Elimination (RFE) with Logistic Regression

```
model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=10)
rfe.fit(X, y)

feature_ranking = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_})
feature_ranking = feature_ranking.sort_values(by='Ranking', ascending=True)
plt.figure(figsize=(10, 6))
plt.bar(feature_ranking['Feature'], feature_ranking['Ranking'])
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("RFE Feature Ranking")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



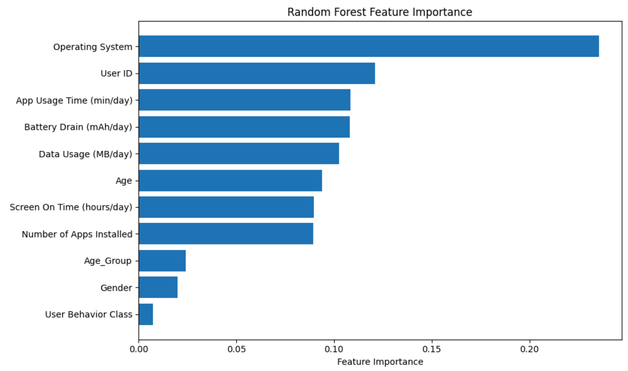
Comparison of models using features selected by the REF

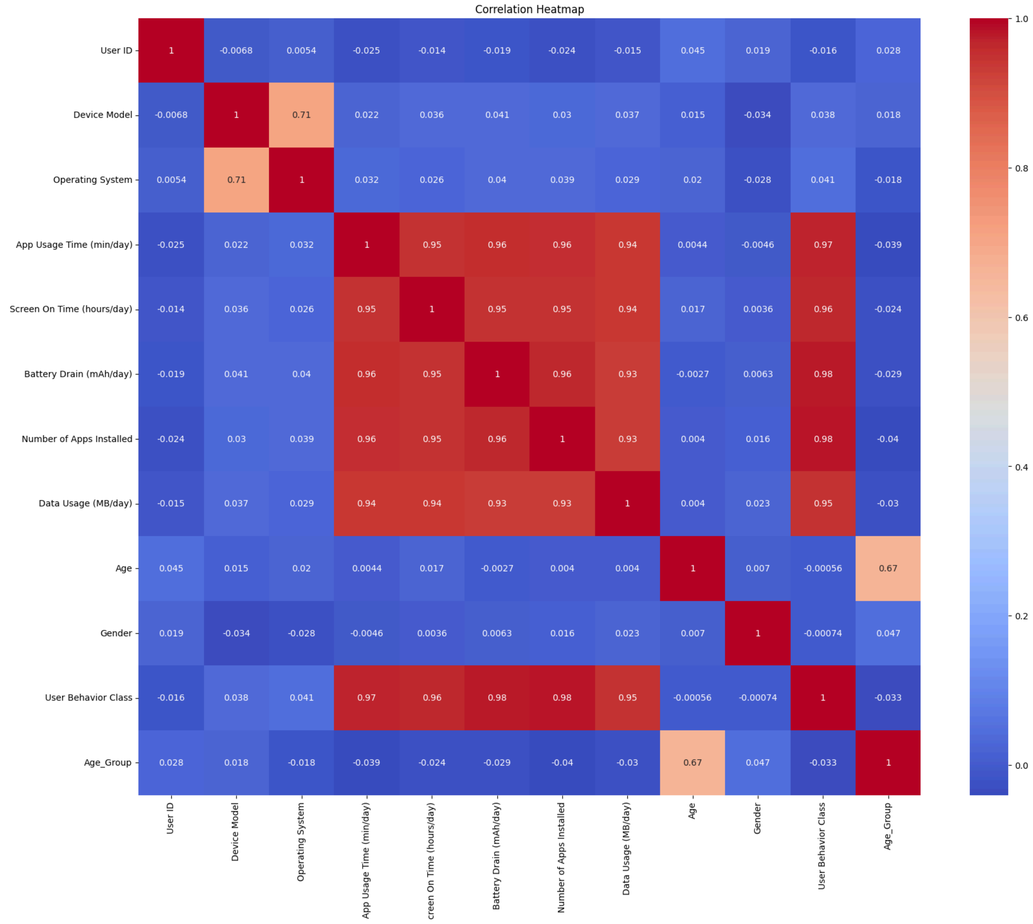
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.30, random_state=42)
model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=10)
rfe.fit(X, y)
X_train_k = rfe.fit_transform(X_train, y_train)
X_test_k = rfe.transform(X_test)
selected_features = X.columns[rfe.get_support(indices=True)]
print("Selected Features:", list(selected_features))


models = { "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
          "Decision Tree": DecisionTreeClassifier(random_state=42),
           "SVM": SVC(kernel='rbf', random_state=42)}
metrics = []

for name, model in models.items():
  model.fit(X_train_k, y_train)
y_pred = model.predict(X_test_k)
print(f"\n===== Classification Report: {name} =====")
print(classification_report(y_test, y_pred))
metrics.append({ "Model": name, "Accuracy": accuracy_score(y_test, y_pred),
                "Precision": precision_score(y_test, y_pred, average='weighted'),
                 "Recall": recall_score(y_test, y_pred, average='weighted') })

results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)
```

```
Selected Features: ['User ID', 'Operating System', 'App Usage Time (min/day)', 'Screen On Time (hours/day)', 'Battery Drain (mAh/day)', 'Number of Apps Installed', 'Age', 'Gender', 'User Behavior Class', 'Age_Group']

===== Classification Report: SVM =====
              precision    recall  f1-score   support

           0       0.14      1.00      0.25        30
           1       0.00      0.00      0.00        46
           2       0.00      0.00      0.00        37
           3       0.00      0.00      0.00        52
           4       0.00      0.00      0.00        45

    accuracy                           0.14       210
   macro avg       0.03      0.20      0.05       210
weighted avg       0.02      0.14      0.04       210


Model Performance:
  Model  Accuracy  Precision    Recall
0   SVM  0.142857   0.020408  0.142857
```

Train Random Forest

```
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)

importances = rf.feature_importances_

feature_names = X.columns if hasattr(X, "columns") else [f"Feature {i}"
                                                         for i in range(X.shape[1])]


df_importance = pd.DataFrame({ 'Feature': feature_names, 'Importance': importances})

df_importance = df_importance.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(df_importance['Feature'][::-1],
         df_importance['Importance'][::-1])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()
```




```
from sklearn.preprocessing import LabelEncoder
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
  le = LabelEncoder()
  df[column] = le.fit_transform(df[column])
  label_encoders[column] = le
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```