Customer Churn Prediction Project

The objective of this project is to develop a machine learning model that accurately predicts monthly user churn for Taxiride navigation app.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline

df = pd.read_csv('dataset.csv')
df.head(5)
```

```
   ID     label  sessions  drives  total_sessions
n_days_after_onboarding  \
0   0  retained       283     226      296.748273
2276
1   1  retained       133     107      326.896596
1225
2   2  retained       114      95      135.522926
2651
3   3  retained        49      40       67.589221
15
4   4  retained        84      68      168.247020
1562

   total_navigations_fav1  total_navigations_fav2  driven_km_drives  \
0                     208                       0       2628.845068
1                      19                      64      13715.920550
2                       0                       0       3059.148818
3                     322                       7        913.591123
4                     166                       5       3950.202008

   duration_minutes_drives  activity_days  driving_days   device
0              1985.775061             28            19  Android
1              3160.472914             13            11   iPhone
2              1610.735904             14             8  Android
3               587.196542              7             3   iPhone
4              1219.555924             27            18  Android
```

```
df.shape
```

```
(14999, 13)
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ID                       14999 non-null  int64  
 1   label                    14299 non-null  object 
 2   sessions                 14999 non-null  int64  
 3   drives                   14999 non-null  int64  
 4   total_sessions           14999 non-null  float64
 5   n_days_after_onboarding  14999 non-null  int64  
 6   total_navigations_fav1   14999 non-null  int64  
 7   total_navigations_fav2   14999 non-null  int64  
 8   driven_km_drives         14999 non-null  float64
 9   duration_minutes_drives  14999 non-null  float64
 10  activity_days            14999 non-null  int64  
 11  driving_days             14999 non-null  int64  
 12  device                   14999 non-null  object 
dtypes: float64(3), int64(8), object(2)
memory usage: 1.5+ MB
```

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

```
ID                           0
label                      700
sessions                     0
drives                       0
total_sessions               0
n_days_after_onboarding      0
total_navigations_fav1       0
total_navigations_fav2       0
driven_km_drives             0
duration_minutes_drives      0
activity_days                0
driving_days                 0
device                       0
dtype: int64
```

```python
df = df.dropna(subset=['label'])
```

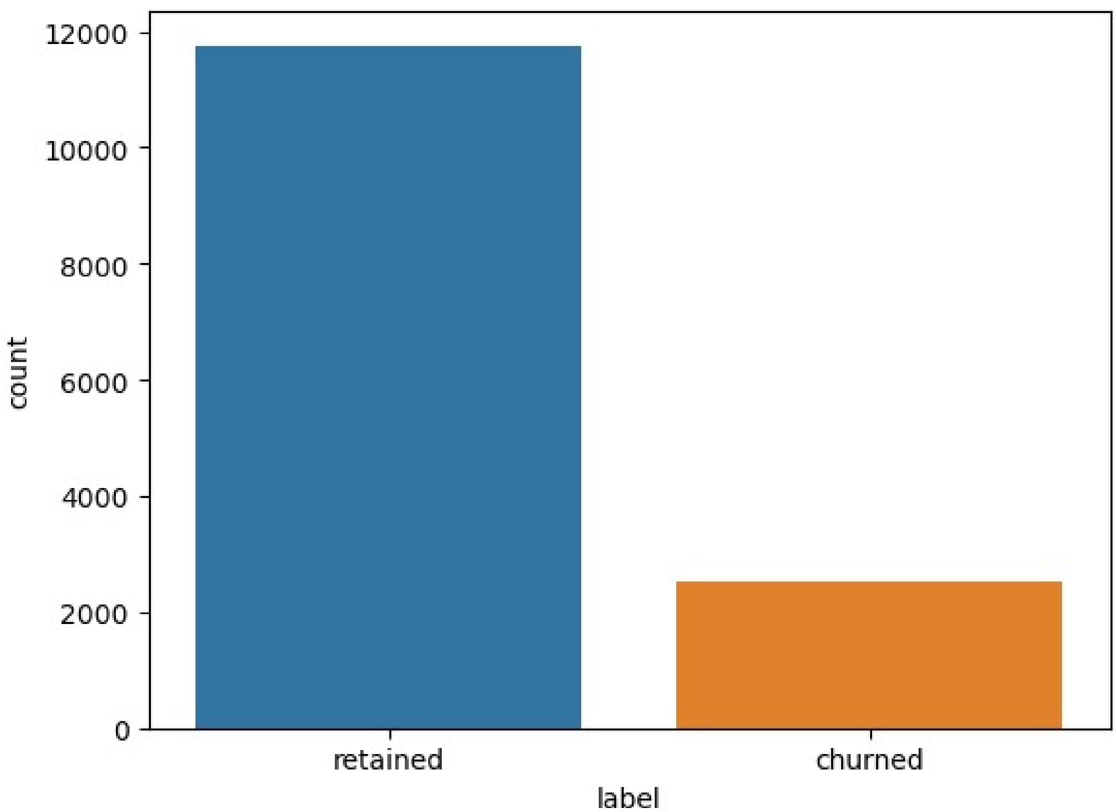
```python
df.isnull().sum()
```

```
ID                         0
label                      0
sessions                   0
drives                     0
total_sessions             0
n_days_after_onboarding    0
total_navigations_fav1     0
```

```
total_navigations_fav2     0
driven_km_drives           0
duration_minutes_drives    0
activity_days              0
driving_days               0
device                     0
dtype: int64
```

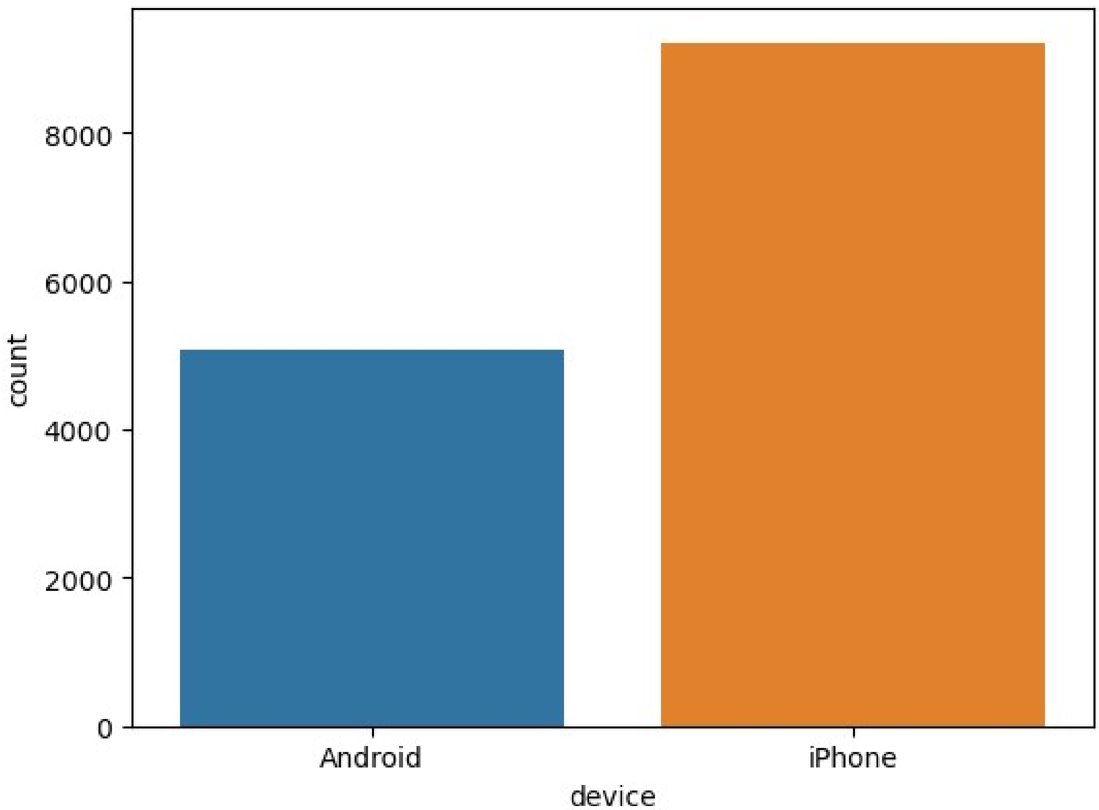
```
sns.countplot(x='label',data = df)
```

```
<Axes: xlabel='label', ylabel='count'>
```




```
sns.countplot(x='device',data = df)
```

```
<Axes: xlabel='device', ylabel='count'>
```




```
df.columns

Index(['ID', 'label', 'sessions', 'drives', 'total_sessions',
       'n_days_after_onboarding', 'total_navigations_fav1',
       'total_navigations_fav2', 'driven_km_drives',
'duration_minutes_drives',
       'activity_days', 'driving_days', 'device'],
      dtype='object')

num_col = df.select_dtypes(include=['number']).columns.tolist()
print(num_col)

['ID', 'sessions', 'drives', 'total_sessions',
'n_days_after_onboarding', 'total_navigations_fav1',
'total_navigations_fav2', 'driven_km_drives',
'duration_minutes_drives', 'activity_days', 'driving_days']

df.set_index('ID', inplace=True)

num_col

['ID',
 'sessions',
 'drives',
 'total_sessions',
```

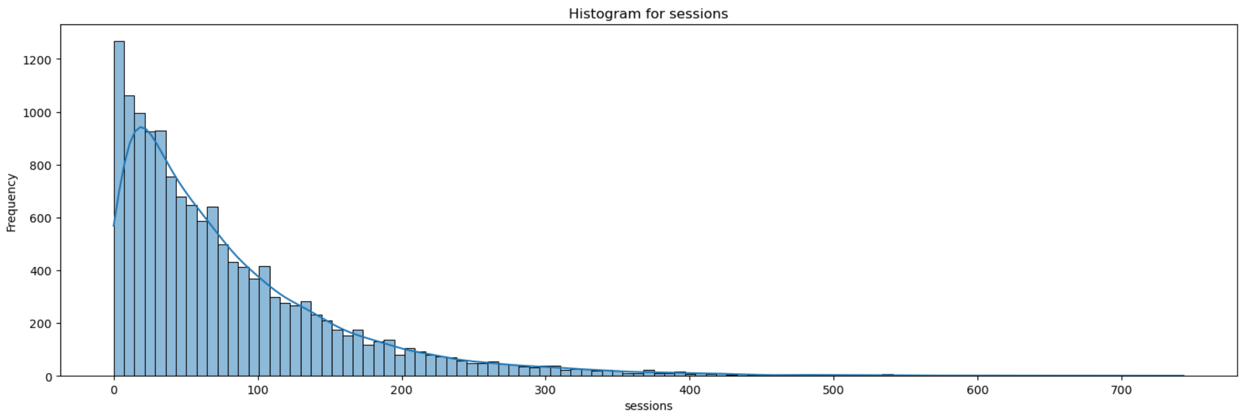
```
 'n_days_after_onboarding',
 'total_navigations_fav1',
 'total_navigations_fav2',
 'driven_km_drives',
 'duration_minutes_drives',
 'activity_days',
 'driving_days']
```

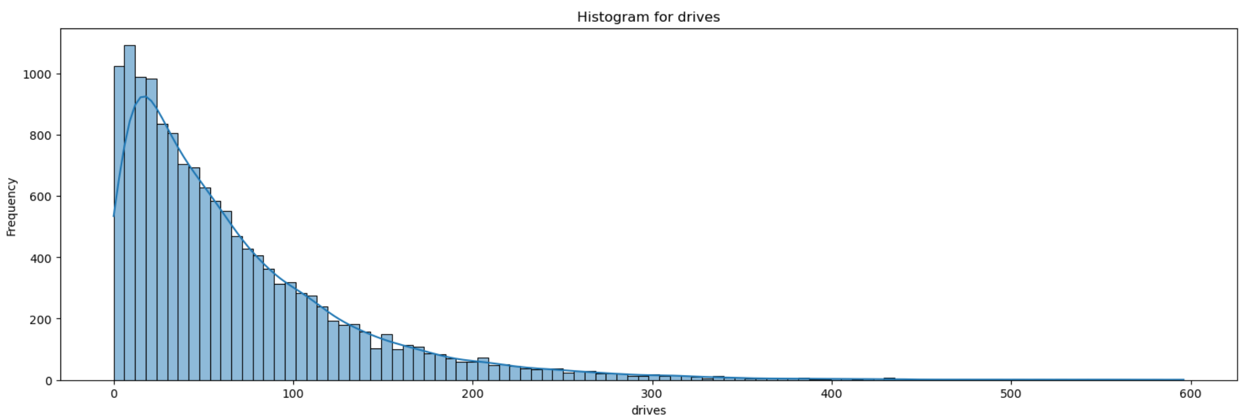
```
num_col.remove('ID')
```

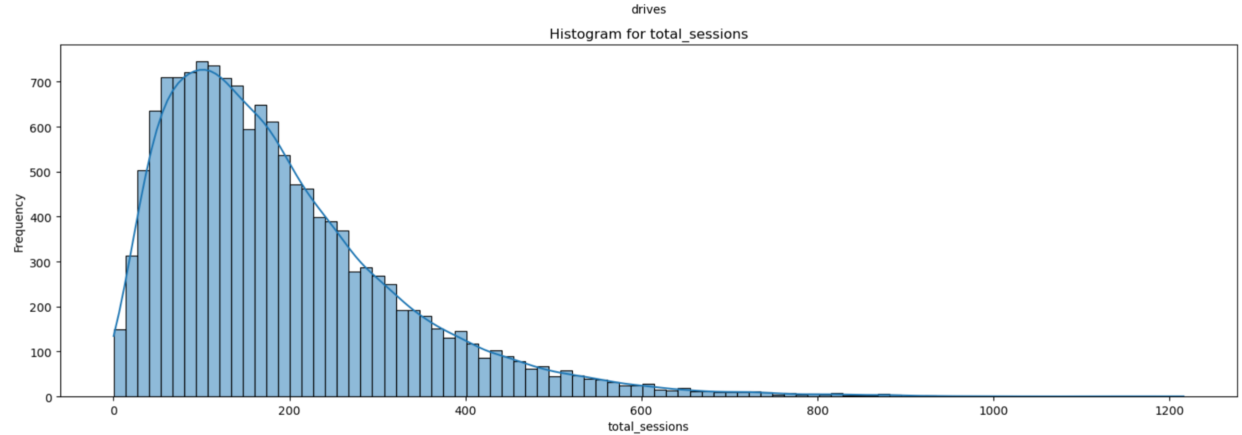
```
num_col
```

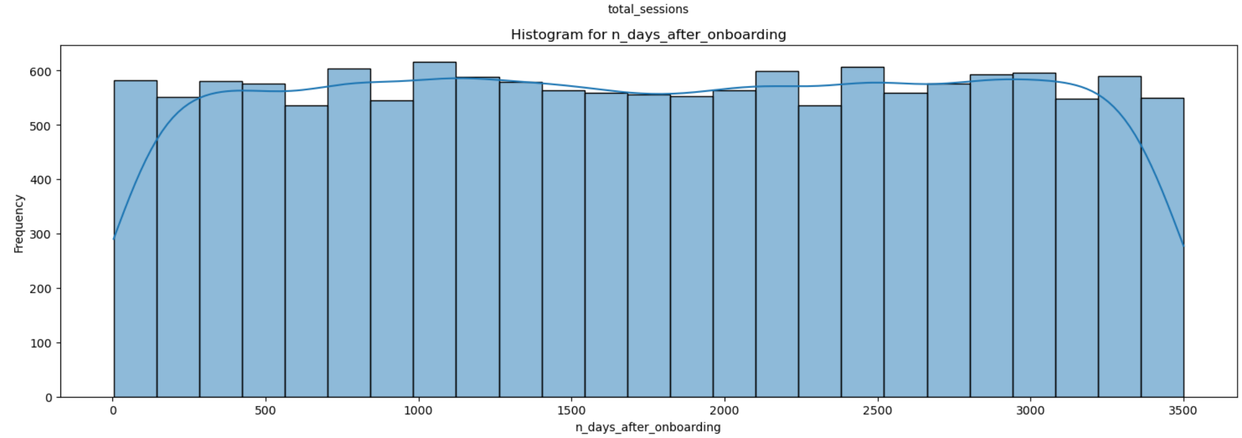
```
['sessions',
 'drives',
 'total_sessions',
 'n_days_after_onboarding',
 'total_navigations_fav1',
 'total_navigations_fav2',
 'driven_km_drives',
 'duration_minutes_drives',
 'activity_days',
 'driving_days']
```

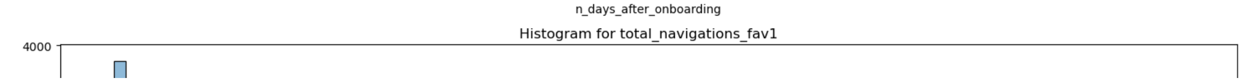
```
plt.figure(figsize=(15,5*len(num_col)))
plot_index = 1

for column in num_col:
    plt.subplot(len(num_col),1,plot_index)
    sns.histplot(data=df, x=column, kde=True)
    plt.title(f'Histogram for {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plot_index += 1

plt.tight_layout()
plt.show()
```


Histogram for sessions
Frequency
sessions
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Histogram for drives
Frequency
drives
0
100
200
300
400
500
600
0
200
400
600
800
1000


Histogram for total_sessions
Frequency
total_sessions
0
200
400
600
800
1000
1200
0
100
200
300
400
500
600
700

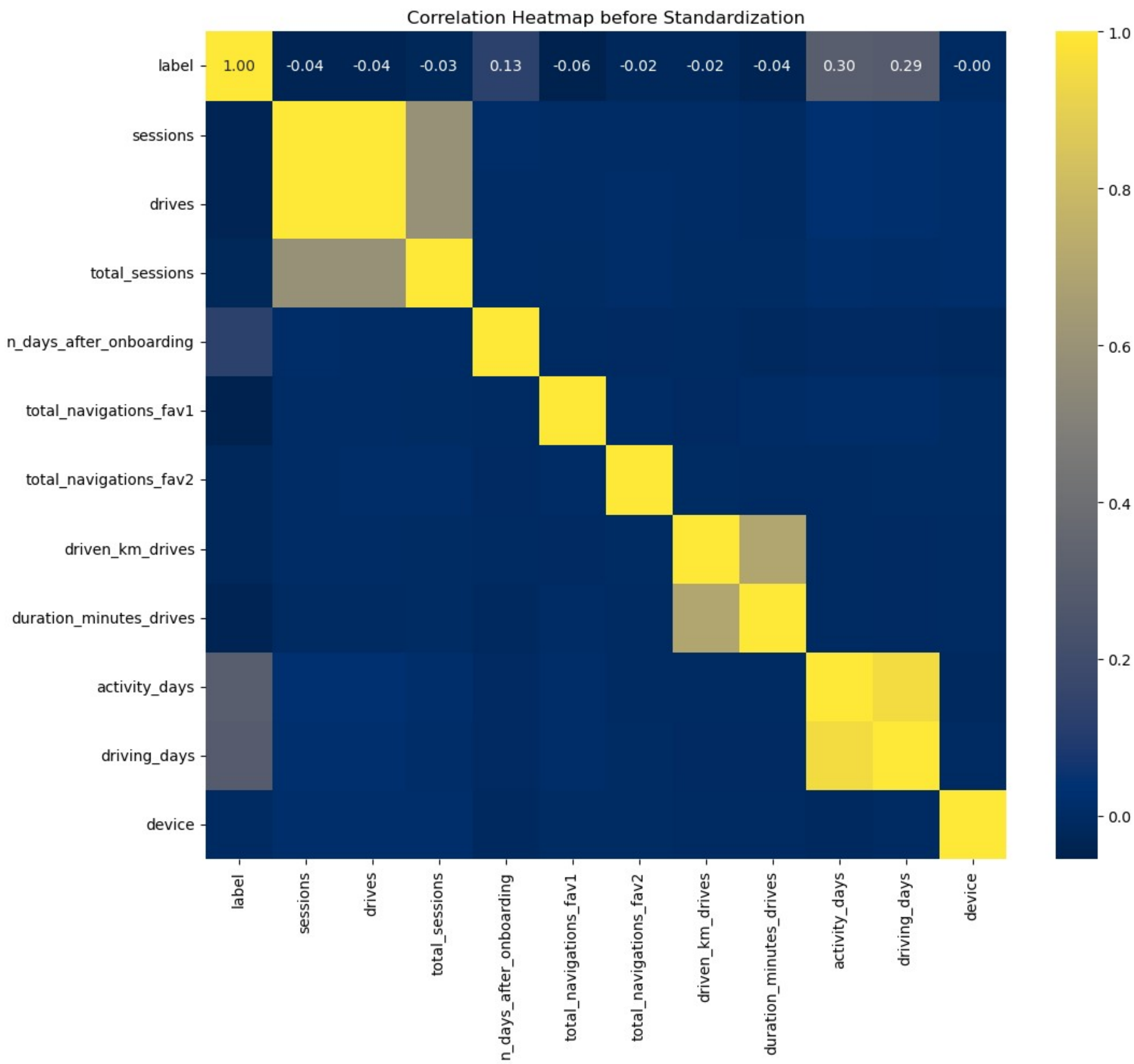

Histogram for n_days_after_onboarding
Frequency
n_days_after_onboarding
0
500
1000
1500
2000
2500
3000
3500
0
100
200
300
400
500
600


Histogram for total_navigations_fav1
4000

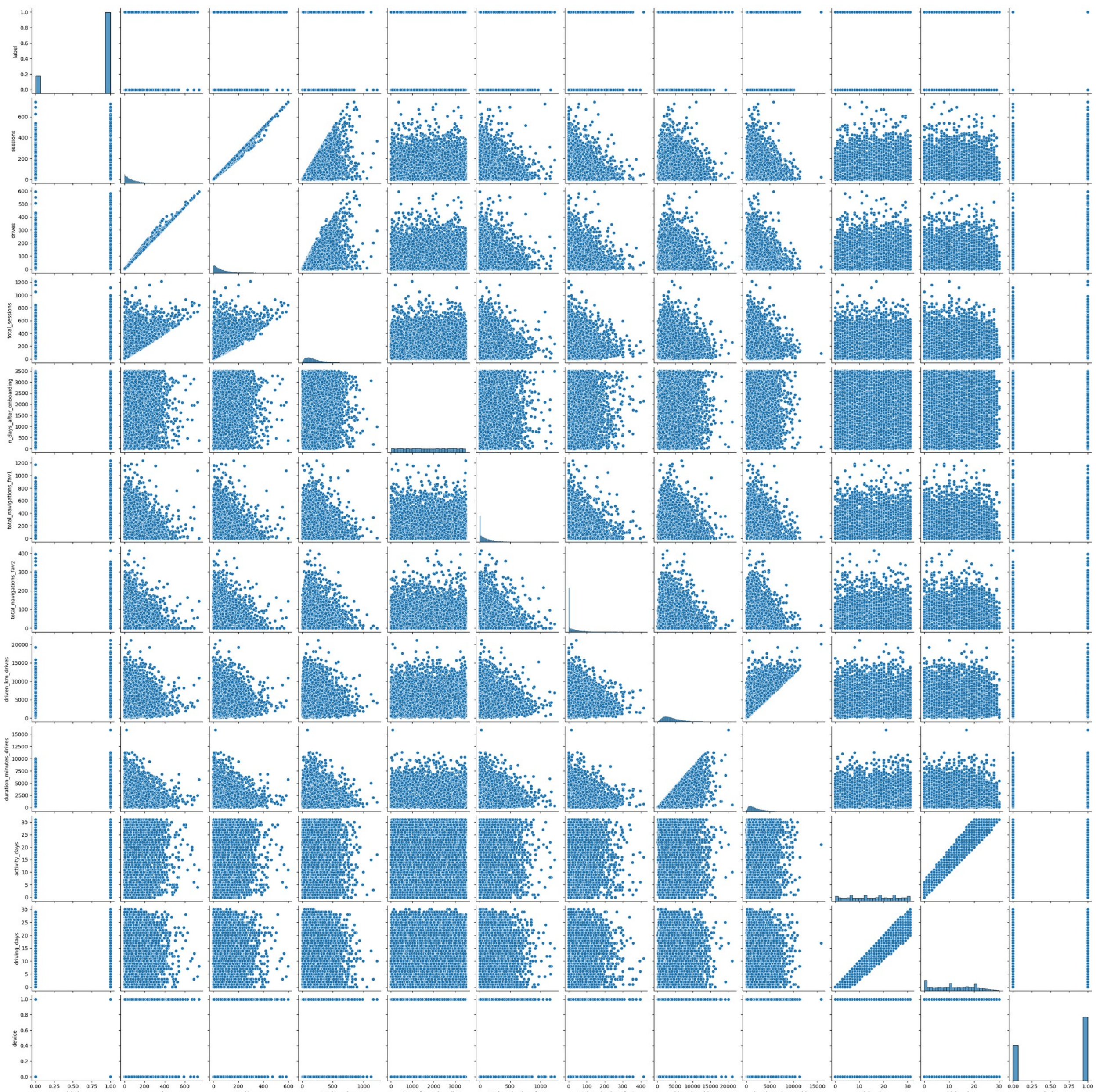
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
cat_col = ['label','device']
df[cat_col] = df[cat_col].apply(label_encoder.fit_transform)
df.head()
```

```
    label  sessions  drives  total_sessions  
n_days_after_onboarding  \
ID

0       1       283     226      296.748273                     2276

1       1       133     107      326.896596                     1225

2       1       114      95      135.522926                     2651

3       1        49      40       67.589221                       15

4       1        84      68      168.247020                     1562


    total_navigations_fav1  total_navigations_fav2  
driven_km_drives  \
ID

0                      208                       0       2628.845068

1                       19                      64      13715.920550

2                        0                       0       3059.148818

3                      322                       7        913.591123

4                      166                       5       3950.202008


    duration_minutes_drives  activity_days  driving_days  device
ID
0                1985.775061             28            19       0
1                3160.472914             13            11       1
2                1610.735904             14             8       0
3                 587.196542              7             3       1
4                1219.555924             27            18       0
```

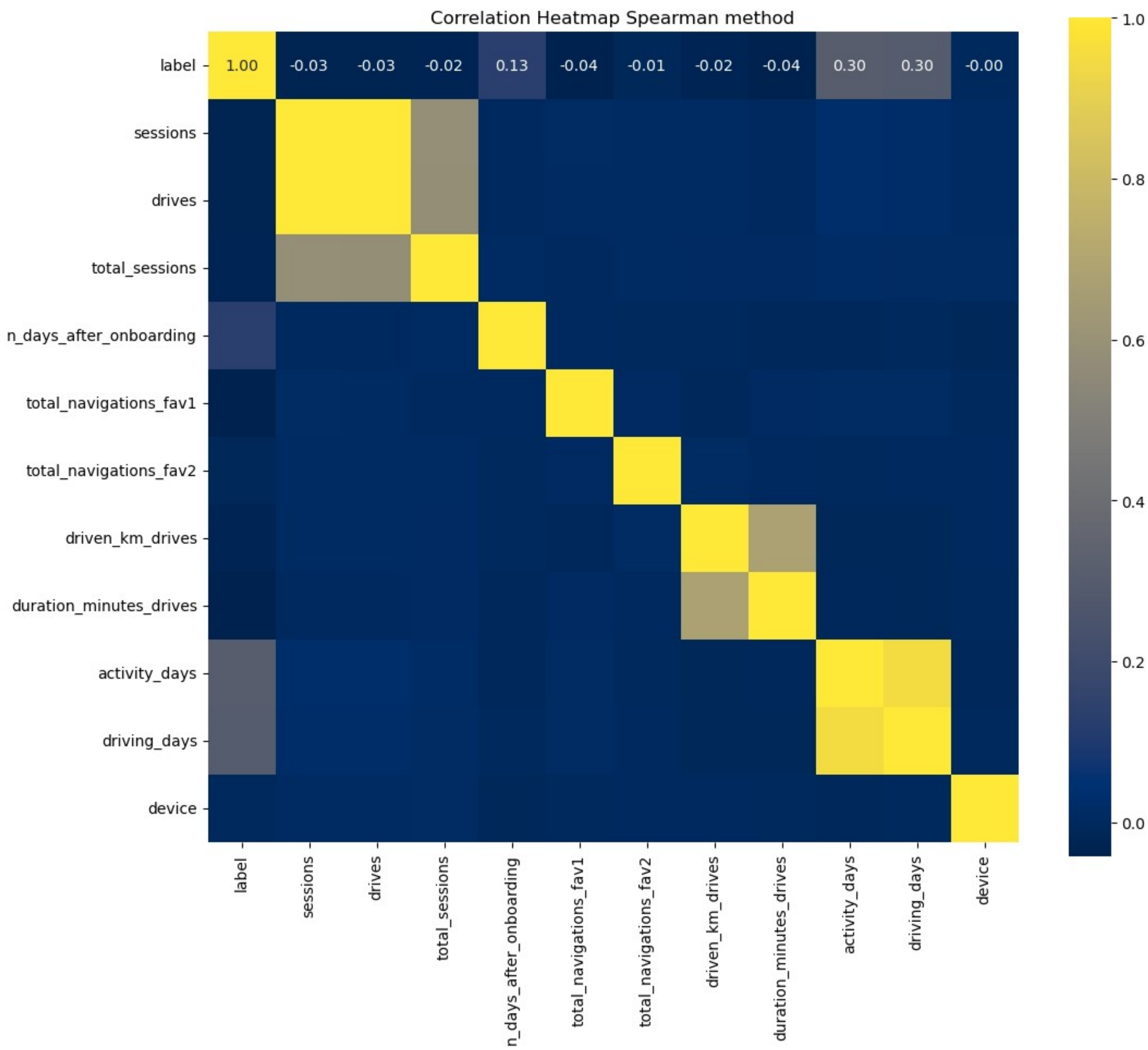
```
corr_matrix = df.corr()
plt.figure(figsize=(12,10))
sns.heatmap(corr_matrix, annot=True, cmap='cividis', fmt='.2f')
plt.title('Correlation Heatmap before Standardization')
plt.show()
```




```
sns.pairplot(df)
plt.show()
```



```python
corr_spearman = df.corr(method='spearman')
plt.figure(figsize=(12,10))
sns.heatmap(corr_spearman, annot=True, cmap='cividis',
fmt='.2f',cbar=True, square=True)
plt.title('Correlation Heatmap Spearman method')
plt.show()
```



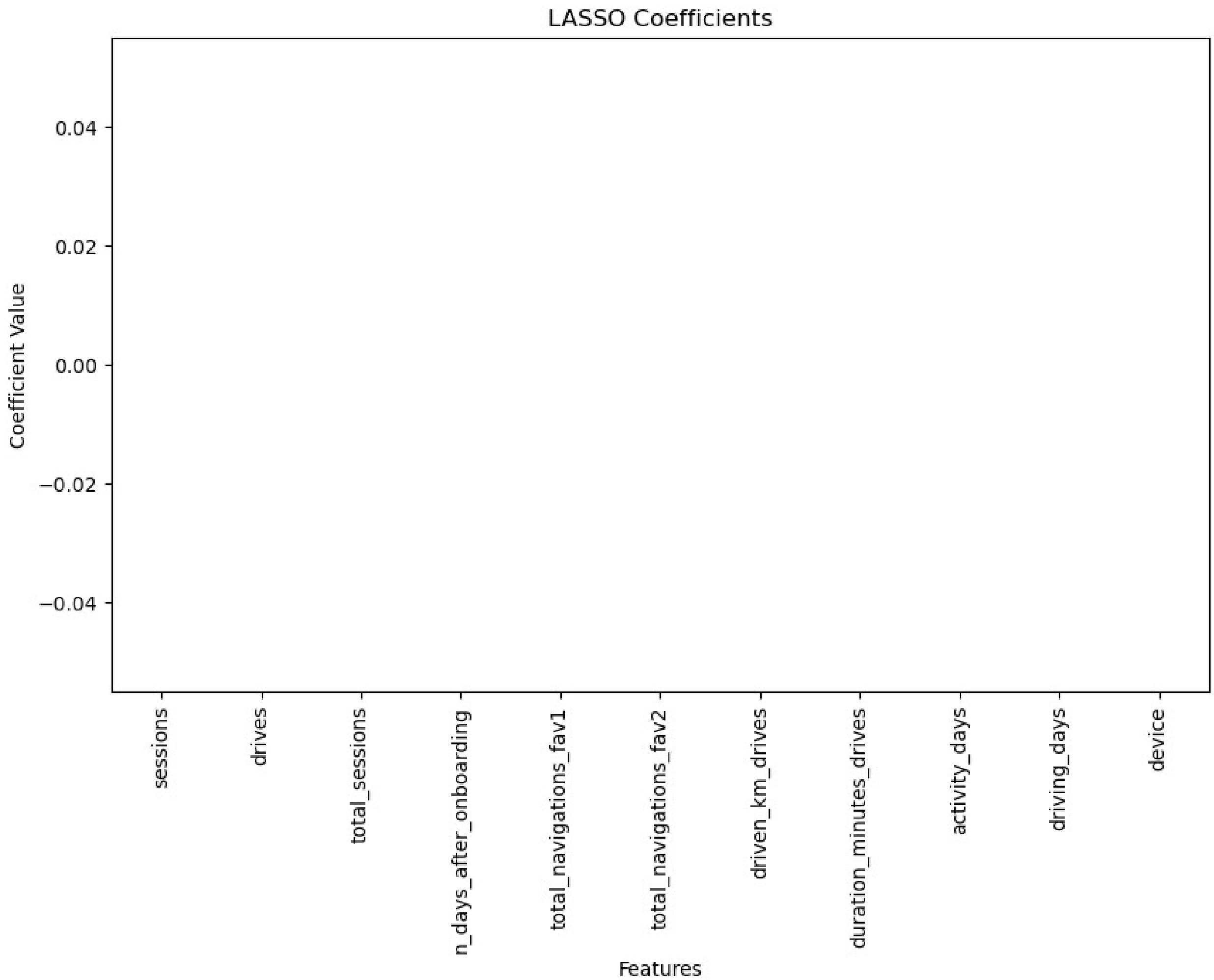

```
# Applying scaling technique for diff numerical columns
# log transformation = sessions,drives,total_sessions, fav1, fav2,
driven_km_drives, duration_minutes
# standardtransformation = activity_days, driving_days,
n_days_after_onboarding

# after that lasso will be applied to figure out less imp feature
from sklearn.preprocessing import RobustScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

X = df.drop(columns=['label'])
y = df['label']

X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.3,
```

```
random_state=1)


columns_for_robust_scale =
['activity_days','driving_days','n_days_after_onboarding']
columns_for_log =
['sessions','drives','total_sessions','total_navigations_fav1','total_
navigations_fav2','driven_km_drives','duration_minutes_drives']

robust_scaler = RobustScaler()
X_train[columns_for_robust_scale]=robust_scaler.fit_transform(X_train[
columns_for_robust_scale])
X_test[columns_for_robust_scale]=robust_scaler.transform(X_test[column
s_for_robust_scale])

X_train[columns_for_log]= np.log1p(X_train[columns_for_log])
X_test[columns_for_log]= np.log1p(X_test[columns_for_log])

print('Transformed X_train : \n', X_train.head())
print('\nTransformed X_train : \n', X_test.head())
```

```
Transformed X_train :
        sessions    drives  total_sessions  n_days_after_onboarding  \
ID
13832  4.912655  4.736198        4.916879                 0.703259
7463   5.187386  4.962845        5.370140                -0.579760
14967  5.433722  5.241747        6.334882                -0.201830
588    4.276666  4.143135        5.166819                 0.486564
8297   4.709530  4.532599        5.286255                 0.785020

       total_navigations_fav1  total_navigations_fav2
driven_km_drives  \
ID

13832                6.102559                4.356709
8.704022
7463                 3.970292                0.000000
8.432454
14967                1.945910                4.770685
7.938738
588                  3.583519                3.555348
9.140724
8297                 0.000000                0.000000
8.722623

       duration_minutes_drives  activity_days  driving_days  device
ID
13832                 8.024135         -0.875     -0.714286       1
7463                  7.381038          0.875      1.000000       1
```

```
14967                      6.588099            0.875      0.714286       1
588                        8.797199            0.125      0.214286       1
8297                       7.832262           -0.500     -0.285714       1

Transformed X_train :
        sessions    drives  total_sessions  n_days_after_onboarding  \
ID
7475   4.564348  4.343805        5.645627                 0.118353
7387   3.258097  3.258097        4.770404                 0.280732
12121  3.218876  3.218876        5.066297                 0.788451
11451  2.944439  2.833213        4.913304                -0.296169
14149  4.219508  4.219508        5.841180                -0.204117

       total_navigations_fav1  total_navigations_fav2
driven_km_drives  \
ID

7475                     6.373320                5.370638
8.552705
7387                     5.823046                2.708050
7.659630
12121                    5.198497                0.000000
8.232976
11451                    4.709530                4.488636
8.445051
14149                    4.941642                0.000000
7.930120

       duration_minutes_drives  activity_days  driving_days  device
ID
7475                  6.787305         0.1250     -0.071429       1
7387                  6.866728        -0.6250     -0.714286       1
12121                 7.685811        -0.3125     -0.071429       1
11451                 8.097410        -0.7500     -0.857143       1
14149                 6.875838        -0.3750     -0.142857       0
```

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.1)
lasso.fit(X_train,y_train)

lasso_coef = lasso.coef_
feature_importance = pd.Series(lasso_coef, index=X.columns)

plt.figure(figsize=(10, 6))
feature_importance.plot(kind='bar')
plt.title('LASSO Coefficients')
plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.show()
```




```
selected_features = feature_importance[feature_importance != 0]
print("Selected Features:\n", selected_features)
```

```
Selected Features:
 Series([], dtype: float64)
```

```
X_train =
X_train.drop(columns=['sessions','total_navigations_fav1','total_navig
ations_fav2','driven_km_drives','device','activity_days'])
X_test =
X_test.drop(columns=['sessions','total_navigations_fav1','total_naviga
tions_fav2','driven_km_drives','device','activity_days'])

X_train.head()
```

```
         drives  total_sessions  n_days_after_onboarding  \
ID
13832  4.736198        4.916879                 0.703259
7463   4.962845        5.370140                -0.579760
14967  5.241747        6.334882                -0.201830
```

```
588    4.143135        5.166819                 0.486564
8297   4.532599        5.286255                 0.785020

       duration_minutes_drives  driving_days
ID
13832                 8.024135     -0.714286
7463                  7.381038      1.000000
14967                 6.588099      0.714286
588                   8.797199      0.214286
8297                  7.832262     -0.285714
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, 
confusion_matrix, f1_score, precision_score, recall_score
```

Model Training

```
!pip install xgboost
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
models = {
    'Logistic Regression': LogisticRegression(max_iter=200),
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5),
    'Random Forest': RandomForestClassifier(n_estimators=100),
    'Support Vector Machine': SVC(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost' : XGBClassifier()
}

results = {}

for model_name, model in models.items():
    cv_scores = cross_val_score(model, X_train, y_train, cv=5)  # 5-
fold cross-validation
    results[model_name] = {
        'CV Scores': cv_scores,
        'Mean CV Score': np.mean(cv_scores)
    }

for model_name, result in results.items():
    print(f"{model_name}:\n  Cross-Validation Scores: {result['CV 
Scores']}\n  Mean CV Score: {result['Mean CV Score']:.4f}\n")


Requirement already satisfied: xgboost in c:\users\bansa\anaconda3\
lib\site-packages (2.1.2)
```

```
Requirement already satisfied: numpy in c:\users\bansa\anaconda3\lib\
site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\bansa\anaconda3\lib\
site-packages (from xgboost) (1.11.4)
Logistic Regression:
  Cross-Validation Scores: [0.82267732 0.82817183 0.82967033
0.82317682 0.82808596]
  Mean CV Score: 0.8264

K-Nearest Neighbors:
  Cross-Validation Scores: [0.81618382 0.8006993  0.7962038
0.81118881 0.8045977 ]
  Mean CV Score: 0.8058

Random Forest:
  Cross-Validation Scores: [0.82117882 0.81918082 0.82217782
0.82567433 0.82008996]
  Mean CV Score: 0.8217

Support Vector Machine:
  Cross-Validation Scores: [0.82417582 0.82417582 0.82417582
0.82467532 0.82458771]
  Mean CV Score: 0.8244

Gradient Boosting:
  Cross-Validation Scores: [0.82417582 0.82717283 0.82817183
0.83066933 0.82058971]
  Mean CV Score: 0.8262

XGBoost:
  Cross-Validation Scores: [0.80669331 0.8036963  0.80619381
0.81868132 0.81209395]
  Mean CV Score: 0.8095
```

Logistic Regression and Gradient Boosting classifiers have high CV score among others.

```
## Hyper parameter tuning of Logistic regression and Gradient Boosting

Logistic_param_grid =  {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'C': [0.01, 0.1, 1.0, 10, 100],
    'solver': ['liblinear', 'lbfgs', 'saga'],
    'class_weight': [None, 'balanced']
}
```

```
logistic_model = LogisticRegression()

logi_grid_search = GridSearchCV(estimator=logistic_model,
param_grid=Logistic_param_grid, cv=5, scoring='accuracy', n_jobs=-1 )

logi_grid_search.fit(X_train, y_train)

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

```
Best Parameters: {'C': 1.0, 'class_weight': None, 'penalty': 'l2',
'solver': 'liblinear'}
Best CV Score: 0.8271558027180216
```

```
final_logistic_model = logi_grid_search.best_estimator_
logi_y_pred = final_logistic_model.predict(X_test)

logi_accuracy = accuracy_score(y_test, logi_y_pred)

print("Test Set Accuracy:", logi_accuracy)
print("Classification Report:\n", classification_report(y_test,
logi_y_pred))
```

```
Test Set Accuracy: 0.8200466200466201
Classification Report:
               precision    recall  f1-score   support

           0       0.54      0.05      0.09       778
           1       0.82      0.99      0.90      3512

    accuracy                           0.82      4290
   macro avg       0.68      0.52      0.50      4290
weighted avg       0.77      0.82      0.75      4290
```