```
import zipfile
import os

zip_path = '/content/drive/My Drive/data.zip'
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

```
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall('/content/data')

import pandas as pd
import numpy as np

# Çıkartılan dosyaları oku
acc_data = pd.read_csv('/content/data/data/all_accelerometer_data_pids_13.csv')
print("Accelerometer Data:")
print(acc_data.head())
```

```
Accelerometer Data:
            time     pid       x       y       z
0              0  JB3156  0.0000  0.0000  0.0000
1              0  CC6740  0.0000  0.0000  0.0000
2  1493733882409  SA0297  0.0758  0.0273 -0.0102
3  1493733882455  SA0297 -0.0359  0.0794  0.0037
4  1493733882500  SA0297 -0.2427 -0.0861 -0.0163
```

```
clean_tac_dir = '/content/data/data/clean_tac'  # Doğru yolu kullandığınızdan e
clean_tac_files = os.listdir(clean_tac_dir)

print("Files in clean_tac directory:", clean_tac_files)
```

```
Files in clean_tac directory: ['SA0297_clean_TAC.csv', 'BU4707_clean_TAC.cs
```

```
clean_tac_files = [f for f in os.listdir(clean_tac_dir) if f.endswith('_clean_T

print("Filtered TAC files:", clean_tac_files)
```

```
Filtered TAC files: ['SA0297_clean_TAC.csv', 'BU4707_clean_TAC.csv', 'BK761
```

```
all_tac_data = []

for file in clean_tac_files:
    file_path = os.path.join(clean_tac_dir, file)
    print(f"Reading file: {file_path}")  # Hangi dosyanın okunduğunu kontrol et
    tac_data = pd.read_csv(file_path)
    all_tac_data.append(tac_data)

if all_tac_data:
    combined_tac_data = pd.concat(all_tac_data, ignore_index=True)
```

```
    print("Combined TAC Data:")
    print(combined_tac_data.head())
else:
    print("No TAC data files found or failed to read.")
```

```
Reading file: /content/data/data/clean_tac/SA0297_clean_TAC.csv
Reading file: /content/data/data/clean_tac/BU4707_clean_TAC.csv
Reading file: /content/data/data/clean_tac/BK7610_clean_TAC.csv
Reading file: /content/data/data/clean_tac/PC6771_clean_TAC.csv
Reading file: /content/data/data/clean_tac/MJ8002_clean_TAC.csv
Reading file: /content/data/data/clean_tac/SF3079_clean_TAC.csv
Reading file: /content/data/data/clean_tac/JB3156_clean_TAC.csv
Reading file: /content/data/data/clean_tac/HV0618_clean_TAC.csv
Reading file: /content/data/data/clean_tac/JR8022_clean_TAC.csv
Reading file: /content/data/data/clean_tac/MC7070_clean_TAC.csv
Reading file: /content/data/data/clean_tac/CC6740_clean_TAC.csv
Reading file: /content/data/data/clean_tac/DK3500_clean_TAC.csv
Reading file: /content/data/data/clean_tac/DC6359_clean_TAC.csv
Combined TAC Data:
    timestamp  TAC_Reading
0  1493716723    -0.010229
1  1493718546    -0.002512
2  1493718863     0.003249
3  1493719179     0.005404
4  1493719495     0.003377
```

```
combined_tac_data = pd.concat(all_tac_data, ignore_index=True)
print("Combined TAC Data:")
print(combined_tac_data.head())
```

```
Combined TAC Data:
    timestamp  TAC_Reading
0  1493716723    -0.010229
1  1493718546    -0.002512
2  1493718863     0.003249
3  1493719179     0.005404
4  1493719495     0.003377
```

```
!ls /content/data/data
```

```
all_accelerometer_data_pids_13.csv  clean_tac  phone_types.csv  pids.txt  r
```

```
phone_data = pd.read_csv('/content/data/data/phone_types.csv')
print("Phone Types Data:")
print(phone_data.head())
```

```
Phone Types Data:
      pid phonetype
0  BK7610    iPhone
1  BU4707    iPhone
2  CC6740   Android
3  DC6359    iPhone
4  DK3500    iPhone
```

```
print(acc_data.head())
```

```
             time     pid       x       y       z
```

```
0              0  JB3156  0.0000  0.0000  0.0000
1              0  CC6740  0.0000  0.0000  0.0000
2  1493733882409  SA0297  0.0758  0.0273 -0.0102
3  1493733882455  SA0297 -0.0359  0.0794  0.0037
4  1493733882500  SA0297 -0.2427 -0.0861 -0.0163
```

```
print(combined_tac_data.head())
```

```
    timestamp  TAC_Reading
0  1493716723    -0.010229
1  1493718546    -0.002512
2  1493718863     0.003249
3  1493719179     0.005404
4  1493719495     0.003377
```

```
print(phone_data.head())
```

```
      pid phonetype
0  BK7610    iPhone
1  BU4707    iPhone
2  CC6740   Android
3  DC6359    iPhone
4  DK3500    iPhone
```

```
acc_data = acc_data.rename(columns={'time': 'timestamp'})
print(acc_data.head())  # Değişiklikleri kontrol et
```

```
       timestamp     pid       x       y       z
0              0  JB3156  0.0000  0.0000  0.0000
1              0  CC6740  0.0000  0.0000  0.0000
2  1493733882409  SA0297  0.0758  0.0273 -0.0102
3  1493733882455  SA0297 -0.0359  0.0794  0.0037
4  1493733882500  SA0297 -0.2427 -0.0861 -0.0163
```

```
# Verileri birleştir
merged_data = pd.merge(acc_data, combined_tac_data, on='timestamp', how='inner'
merged_data = pd.merge(merged_data, phone_data, on='pid', how='inner')

# Birleştirilmiş veriyi görüntüle
print("Merged Data:")
print(merged_data.head())
```

```
Merged Data:
Empty DataFrame
Columns: [timestamp, x, y, z, TAC_Reading, pid, phonetype]
Index: []
```

```
# Eksik verileri kontrol et
print(merged_data.isnull().sum())

# Gerekirse eksik verileri temizle
merged_data = merged_data.dropna()
print(merged_data.info())
# Verinin temel istatistiklerini kontrol et
print(merged_data.describe())
```

```
timestamp      0.0
x              0.0
y              0.0
z              0.0
TAC_Reading    0.0
pid            0.0
phonetype      0.0
dtype: float64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 0 entries
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   timestamp    0 non-null      int64  
 1   x            0 non-null      float64
 2   y            0 non-null      float64
 3   z            0 non-null      float64
 4   TAC_Reading  0 non-null      float64
 5   pid          0 non-null      object 
 6   phonetype    0 non-null      object 
dtypes: float64(4), int64(1), object(2)
memory usage: 124.0+ bytes
None
       timestamp    x    y    z  TAC_Reading
count        0.0  0.0  0.0  0.0          0.0
mean         NaN  NaN  NaN  NaN          NaN
std          NaN  NaN  NaN  NaN          NaN
min          NaN  NaN  NaN  NaN          NaN
25%          NaN  NaN  NaN  NaN          NaN
50%          NaN  NaN  NaN  NaN          NaN
75%          NaN  NaN  NaN  NaN          NaN
max          NaN  NaN  NaN  NaN          NaN
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
# merged_data DataFrame'inin ilk birkaç satırını görüntüle
print(merged_data.head(50))
```

```
Empty DataFrame
Columns: [timestamp, x, y, z, TAC_Reading, pid, phonetype]
Index: []
```

```
print(acc_data.head())
print(acc_data.describe())
```

```
       timestamp     pid       x       y       z
0              0  JB3156  0.0000  0.0000  0.0000
1              0  CC6740  0.0000  0.0000  0.0000
2  1493733882409  SA0297  0.0758  0.0273 -0.0102
3  1493733882455  SA0297 -0.0359  0.0794  0.0037
4  1493733882500  SA0297 -0.2427 -0.0861 -0.0163
          timestamp             x             y             z
count  1.405757e+07  1.405757e+07  1.405757e+07  1.405757e+07
mean   1.493778e+12 -9.269848e-03 -7.168398e+06  7.168398e+06
```

```
std    3.640455e+08  9.540696e-01  3.528998e+07  3.528998e+07
min    0.000000e+00 -4.333507e+01 -1.809008e+08 -4.902300e+01
25%    1.493755e+12 -5.700000e-03 -4.500000e-03 -4.200000e-03
50%    1.493779e+12 -2.000000e-04 -2.000000e-04  6.100000e-03
75%    1.493801e+12  7.600000e-03  4.500000e-03  4.605889e-02
max    1.493829e+12  3.922540e+01  2.731123e+01  1.809008e+08
```

```
print(combined_tac_data.head())
print(combined_tac_data.describe())
```

```
    timestamp  TAC_Reading
0  1493716723    -0.010229
1  1493718546    -0.002512
2  1493718863     0.003249
3  1493719179     0.005404
4  1493719495     0.003377
          timestamp  TAC_Reading
count  7.150000e+02   715.000000
mean   1.493758e+09     0.046124
std    2.707992e+04     0.056917
min    1.493717e+09    -0.028308
25%    1.493732e+09     0.001872
50%    1.493757e+09     0.020045
75%    1.493782e+09     0.076462
max    1.493811e+09     0.244715
```

```
print(phone_data[['pid']].head())
print(phone_data[['pid']].describe())
```

```
      pid
0  BK7610
1  BU4707
2  CC6740
3  DC6359
4  DK3500
           pid
count       13
unique      13
top     BK7610
freq         1
```

```
# combined_tac_data'daki timestamp değerlerini milisaniyeye çevir
combined_tac_data['timestamp'] = combined_tac_data['timestamp'] * 1000
print(combined_tac_data[['timestamp']].describe())
```

```
          timestamp
count  7.150000e+02
mean   1.493758e+12
std    2.707992e+07
min    1.493717e+12
25%    1.493732e+12
50%    1.493757e+12
75%    1.493782e+12
max    1.493811e+12
```

```
# Verileri tekrar birleştir
merged_data = pd.merge(acc_data, combined_tac_data, on='timestamp', how='inner'
```

```
merged_data = pd.merge(merged_data, phone_data, on='pid', how='inner')

# Birleştirilmiş veriyi görüntüle
print("Merged Data:")
print(merged_data.head())
print(merged_data.describe())
```

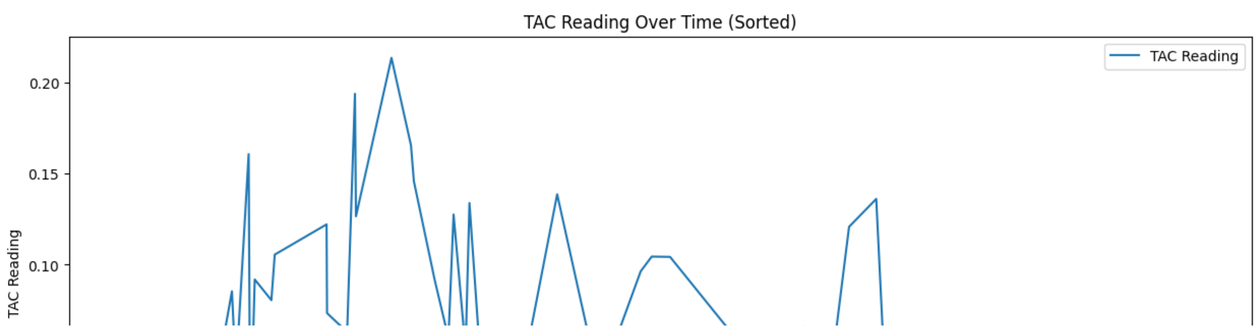
```
Merged Data:
       timestamp    pid       x       y       z  TAC_Reading phonetype
0  1493737519000  BK7610 -0.0002 -0.0007  0.0069     0.009506    iPhone
1  1493745545000  BK7610 -0.3062  0.0052  0.0759     0.003018    iPhone
2  1493752155000  BK7610  0.0420  0.0220 -0.0031     0.193617    iPhone
3  1493752219000  BK7610 -0.2588  0.0658 -0.0047     0.126545    iPhone
4  1493755873000  BK7610  0.1038  0.2266  0.0113     0.145695    iPhone
          timestamp          x             y             z  TAC_Reading
count  6.700000e+01  67.000000  6.700000e+01  6.700000e+01    67.000000
mean   1.493773e+12   0.247356 -1.350006e+07  1.350006e+07     0.057695
std    2.062243e+07   1.143971  4.789740e+07  4.789739e+07     0.057597
min    1.493738e+12  -1.528698 -1.809008e+08 -3.420135e+00    -0.021409
25%    1.493753e+12  -0.004200 -1.100000e-03 -2.900000e-03     0.010466
50%    1.493772e+12   0.000000  3.000000e-04  6.400000e-03     0.039832
75%    1.493792e+12   0.005200  4.700000e-03  1.780000e-02     0.104308
max    1.493805e+12   6.275203  1.056016e+00  1.809008e+08     0.213314
```

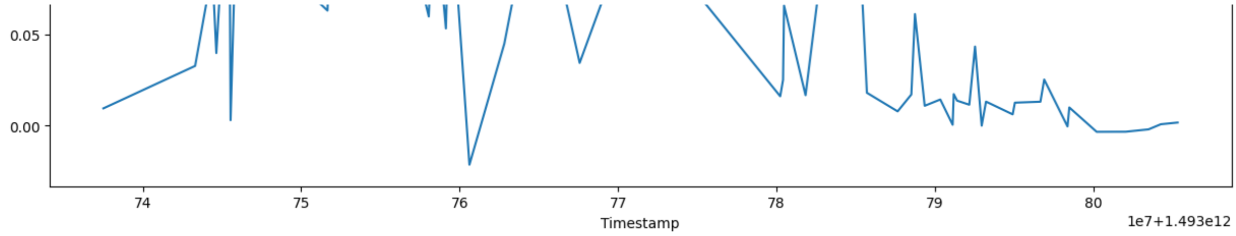
```
# Eksik verileri kontrol et
print(merged_data.isnull().sum())
```

```
timestamp      0
pid            0
x              0
y              0
z              0
TAC_Reading    0
phonetype      0
dtype: int64
```

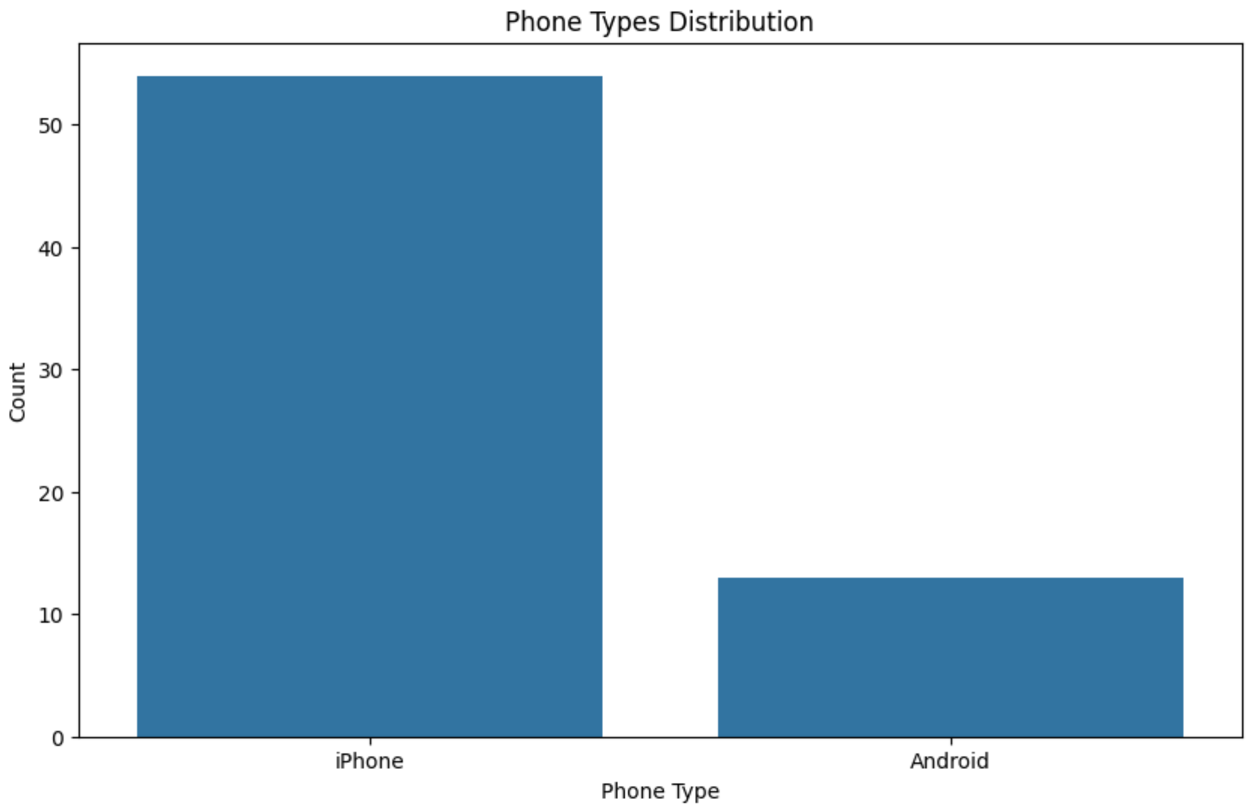
```
merged_data_sorted = merged_data.sort_values(by='timestamp')
```

```
plt.figure(figsize=(15, 6))
plt.plot(merged_data_sorted['timestamp'], merged_data_sorted['TAC_Reading'], la
plt.xlabel('Timestamp')
plt.ylabel('TAC Reading')
plt.title('TAC Reading Over Time (Sorted)')
plt.legend()
plt.show()
```

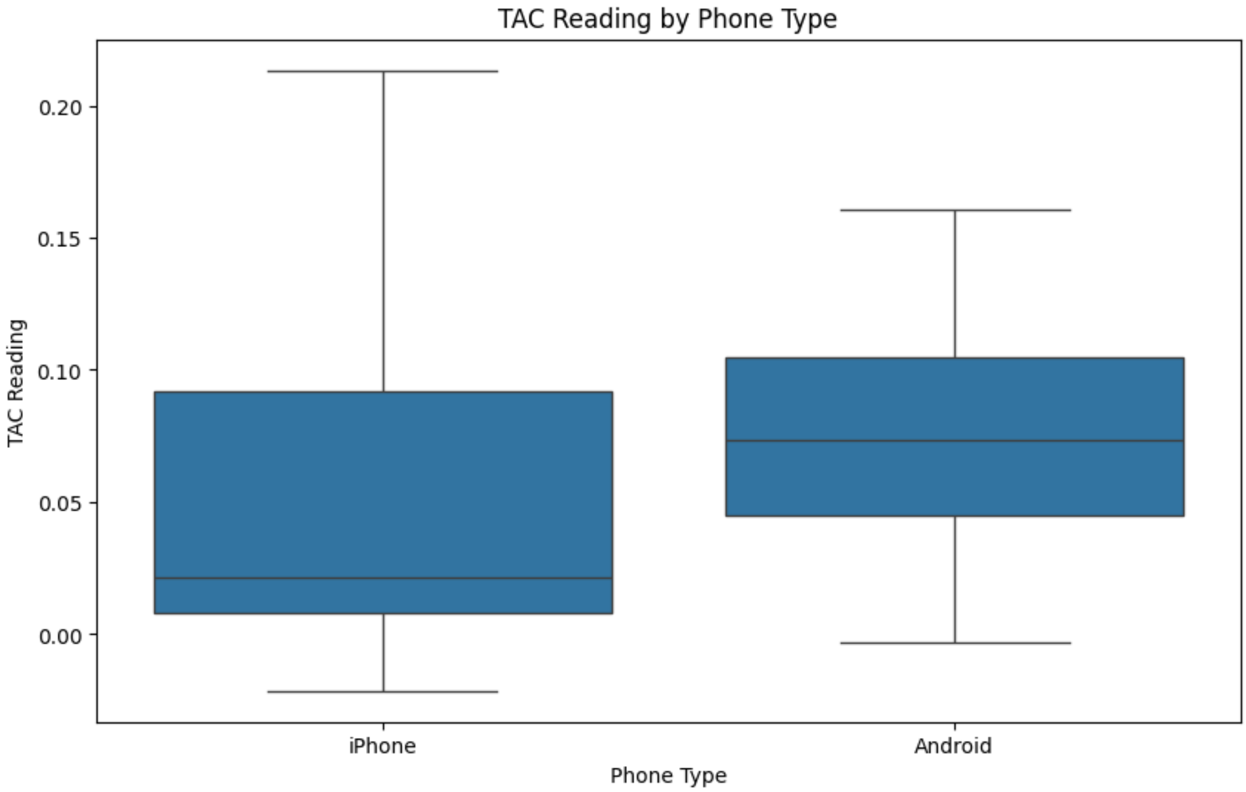





```
phone_type_counts = merged_data['phonetype'].value_counts()

plt.figure(figsize=(10, 6))
sns.barplot(x=phone_type_counts.index, y=phone_type_counts.values)
plt.title('Phone Types Distribution')
plt.xlabel('Phone Type')
plt.ylabel('Count')
plt.show()
```



```
plt.figure(figsize=(10, 6))
sns.boxplot(x='phonetype', y='TAC_Reading', data=merged_data)
plt.title('TAC Reading by Phone Type')
plt.xlabel('Phone Type')
plt.ylabel('TAC Reading')
plt.show()
```

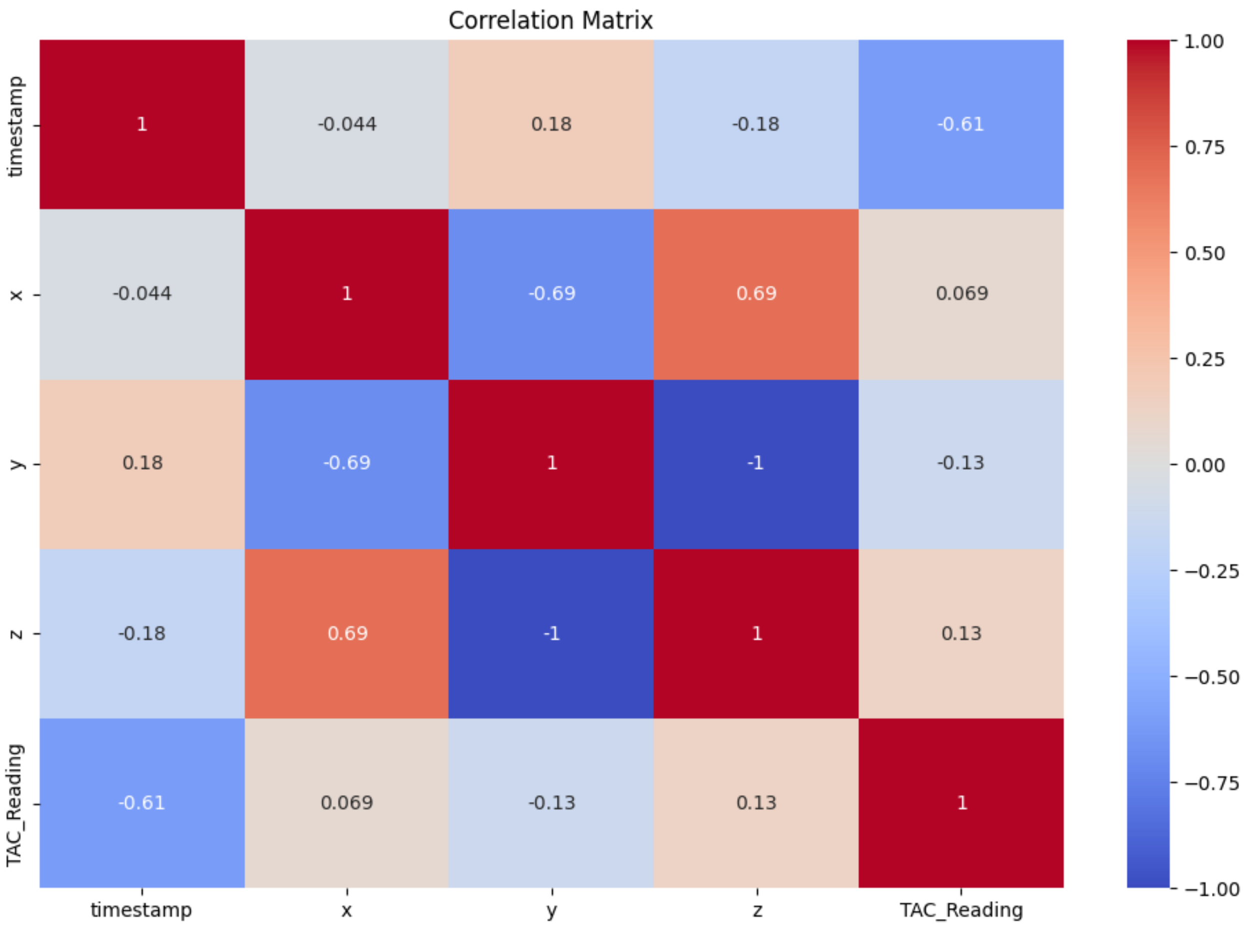



```
numeric_data = merged_data.select_dtypes(include=['float64', 'int64'])
print(numeric_data.head())
```

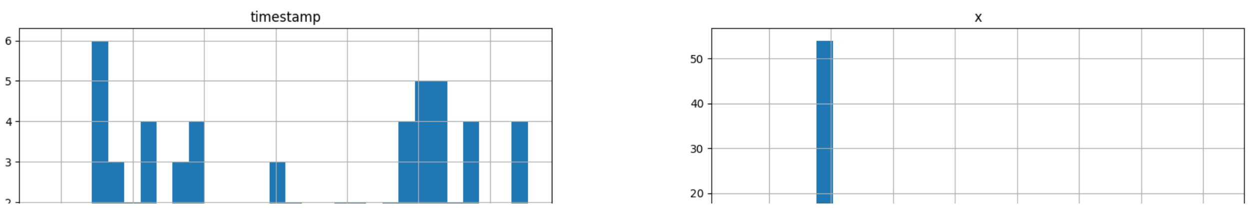
```
       timestamp       x       y       z  TAC_Reading
0  1493737519000 -0.0002 -0.0007  0.0069     0.009506
1  1493745545000 -0.3062  0.0052  0.0759     0.003018
2  1493752155000  0.0420  0.0220 -0.0031     0.193617
3  1493752219000 -0.2588  0.0658 -0.0047     0.126545
4  1493755873000  0.1038  0.2266  0.0113     0.145695
```

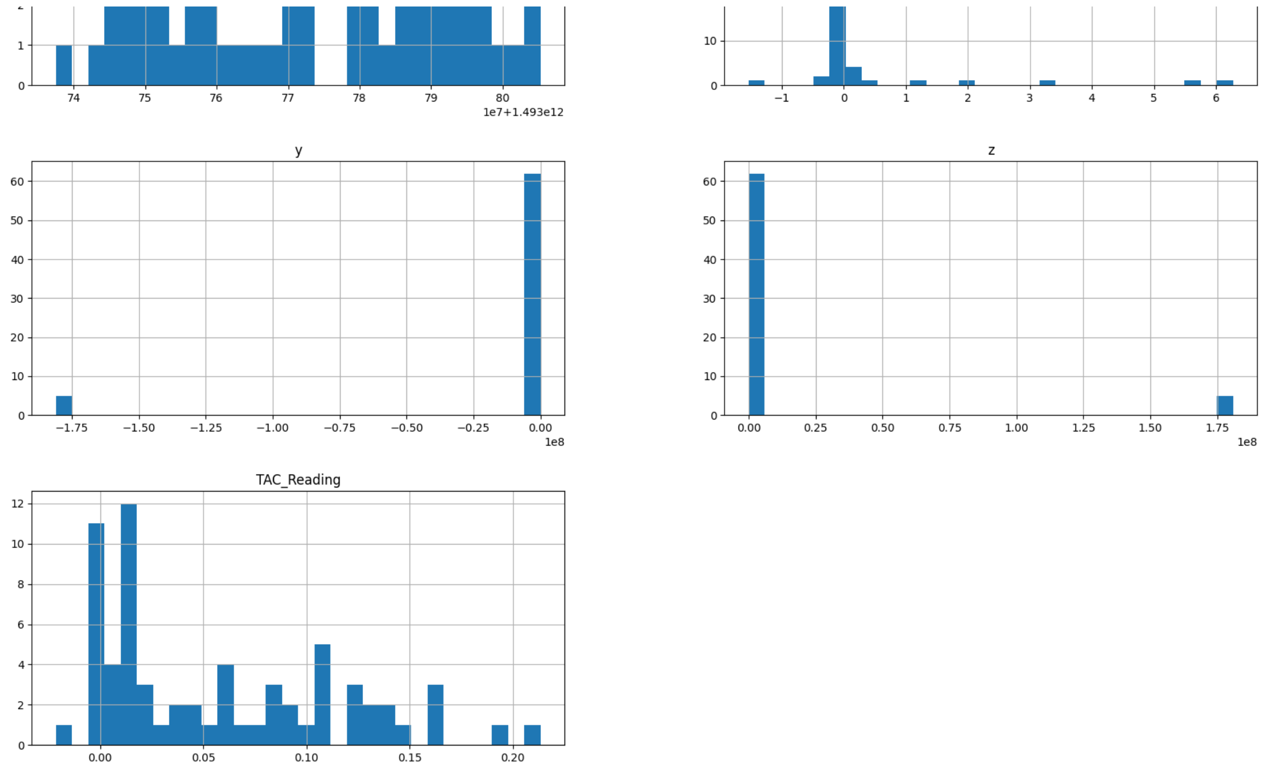
```
# Korelasyon matrisi
```

```
# Korelasyon matrisi
corr_matrix = numeric_data.corr()

# Isı haritası
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



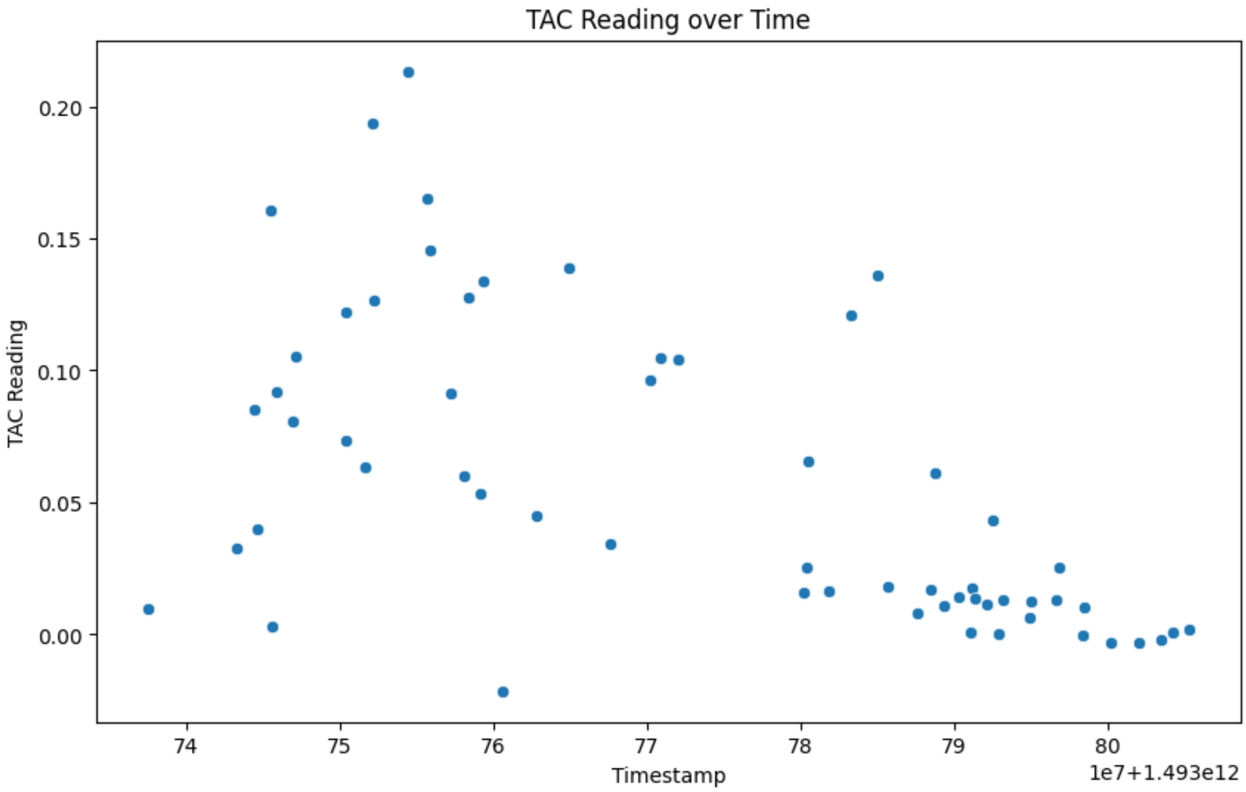

```
# Histogramlar
numeric_data.hist(bins=30, figsize=(20, 15))
plt.show()
```

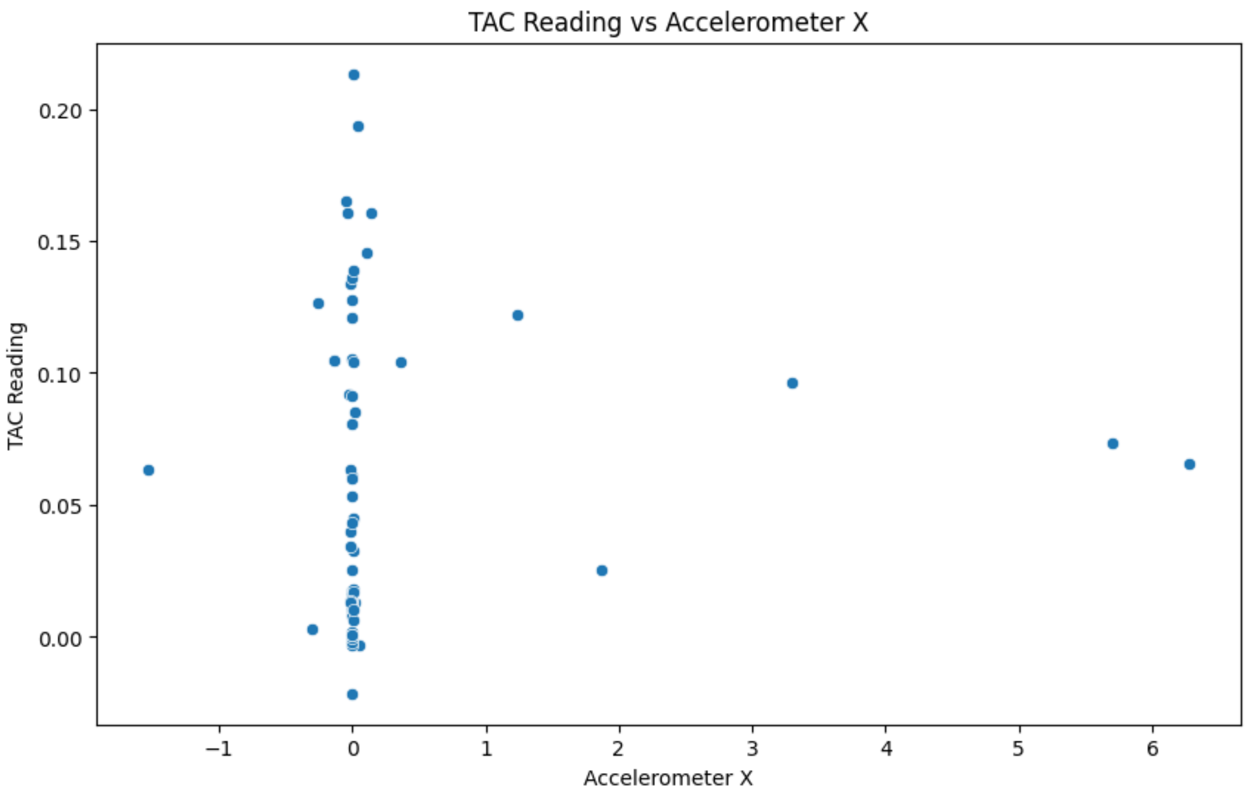





```
# Scatter plotlar
plt.figure(figsize=(10, 6))
sns.scatterplot(x='timestamp', y='TAC_Reading', data=merged_data)
plt.title('TAC Reading over Time')
plt.xlabel('Timestamp')
plt.ylabel('TAC Reading')
plt.show()
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='x', y='TAC_Reading', data=merged_data)
plt.title('TAC Reading vs Accelerometer X')
plt.xlabel('Accelerometer X')
plt.ylabel('TAC Reading')
plt.show()
```





Phase 2: Predictive Analytics

For automate analysis to all excel files

```
directory = 'raw_tac'

# Dizin içindeki tüm dosyaları bir listeye ekle
file_list = [f for f in os.listdir(directory) if os.path.isfile(os.path.join(directory,


for file in file_list:
    print()
    print("Anaylsis Of Person: ",file)


    # Read the Excel file
    tac_data = pd.read_excel("raw_tac/"+file, sheet_name=0, skiprows=1)
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
import itertools
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
import os

ls /content/data/data/raw_tac/
```

```
ls '/content/data/data/raw_tac/'
```

```
'BK7610 CAM Results.xlsx'  'HV0618 CAM Results.xlsx'  'PC6771 CAM Results.x
'BU4707 CAM results.xlsx'  'JB3156 CAM Results.xlsx'  'SA0297 CAM Results.x
'CC6740 CAM Results.xlsx'  'JR8022 CAM results.xlsx'  'SF3079 CAM Results.x
'DC6359 CAM Results.xlsx'  'MC7070 CAM Results.xlsx'
'DK3500 CAM Results.xlsx'  'MJ8002 CAM Results.xlsx'
```

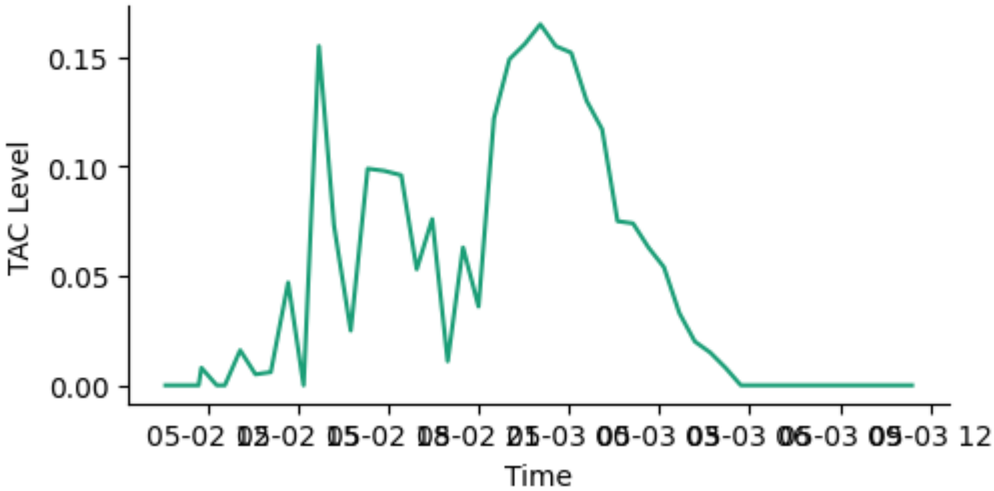
Read the Excel file and Select the necessary columns

```
acc_data = pd.read_csv('/content/data/data/all_accelerometer_data_pids_13.csv')
tac_data = pd.read_excel('/content/data/data/raw_tac/BK7610 CAM Results.xlsx',
tac_data = tac_data[['TAC Level', 'IR Voltage', 'Temperature', 'Time', 'Date']]
tac_data.head(2)
tac_data.info()
tac_data.describe()
tac_data.isnull().sum()
tac_data.nunique()

acc_data.head(2)
acc_data.info()
acc_data.describe()
acc_data.isnull().sum()
acc_data.nunique()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57 entries, 0 to 56
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   TAC Level    57 non-null     float64
 1   IR Voltage   57 non-null     float64
 2   Temperature  57 non-null     float64
 3   Time         57 non-null     datetime64[ns]
 4   Date         57 non-null     datetime64[ns]
dtypes: datetime64[ns](2), float64(3)
memory usage: 2.4 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14057567 entries, 0 to 14057566
Data columns (total 5 columns):
 #   Column  Dtype
---  ------  -----
 0   time    int64
 1   pid     object
 2   x       float64
 3   y       float64
 4   z       float64
dtypes: float64(3), int64(1), object(1)
memory usage: 536.3+ MB
time    12907011
pid           13
x        2940714
y        2729281
z        1328512
dtype: int64
```

Time vs TAC Level

```
# @title Time vs TAC Level

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Time']
  ys = series['TAC Level']

  plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)

fig, ax = plt.subplots(figsize=(5, 2.5), layout='constrained')
df_sorted = tac_data.sort_values('Time', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('TAC Level')
```



Time vs Temperature

```
# @title Time vs Temperature

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['Time']
  ys = series['Temperature']

  plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)

fig, ax = plt.subplots(figsize=(5, 2.5), layout='constrained')
df_sorted = tac_data.sort_values('Time', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
```

```
sns.despine(fig=fig, ax=ax)
plt.xlabel('Time')
_ = plt.ylabel('Temperature')
```

Split features and target variables

```
# İlgili sütunları seçin
accelerometer_features = acc_data[acc_data['pid'] == "BK7610"][['x', 'y', 'z']]
X = tac_data[['IR Voltage', 'Temperature']]
y = tac_data['TAC Level']


# Ölçekleme faktörü
scale_factor = len(accelerometer_features) // len(X)

# Accelerometer features veri setini ölçekleme (downsample)
accelerometer_features_scaled = accelerometer_features.iloc[::scale_factor][:le

combined_data = pd.concat([accelerometer_features_scaled.reset_index(drop=True)

X=combined_data
```

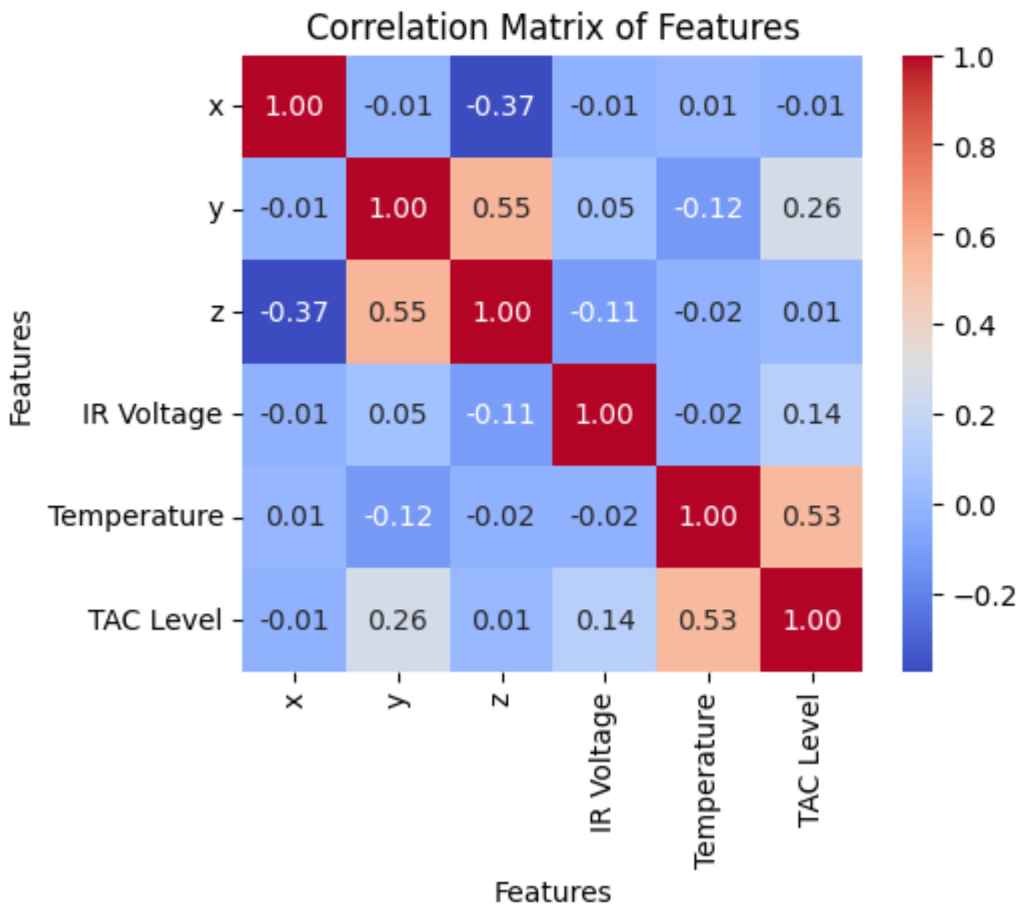
Apply Min-max normalization

```
scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

Correlation matrix

```
corrinput = pd.concat([X.reset_index(drop=True), y.reset_index(drop=True)], axi
corrinput_scaled = pd.DataFrame(scaler.fit_transform(corrinput), columns=corrir
corrinput_scaled_cm = corrinput_scaled.corr()
```

```
# Visualize the correlation matrix
plt.figure(figsize=(5, 4))
sns.heatmap(corrinput_scaled_cm, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Features')
plt.xlabel('Features')
plt.ylabel('Features')
plt.show()
```



Split the data into training and testing sets

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

Linear Regression

```
# Initialize the Linear Regression model
lr_model = LinearRegression()

# Train the model using the training data
lr_model.fit(X_train, y_train)

# Make predictions on the test data
lr_y_pred = lr_model.predict(X_test)
```

actual and predicted values for the test set

```
print("Test Values:", y_test.values)
print("Predicted Values:", lr_y_pred)
```

```
Test Values: [0.    0.    0.149 0.016 0.152 0.    0.063 0.156 0.    0.    0
 0.015 0.011 0.008 0.    0.    0.   ]
Predicted Values: [ 0.00484109  0.0223011   0.07345804  0.05340935  0.10178
  0.03604552  0.08839977 -0.02174259  0.0088466   0.03083502  0.01788691
  0.06902706  0.00103674  0.03263421  0.02699429  0.02207084  0.02185232]
```

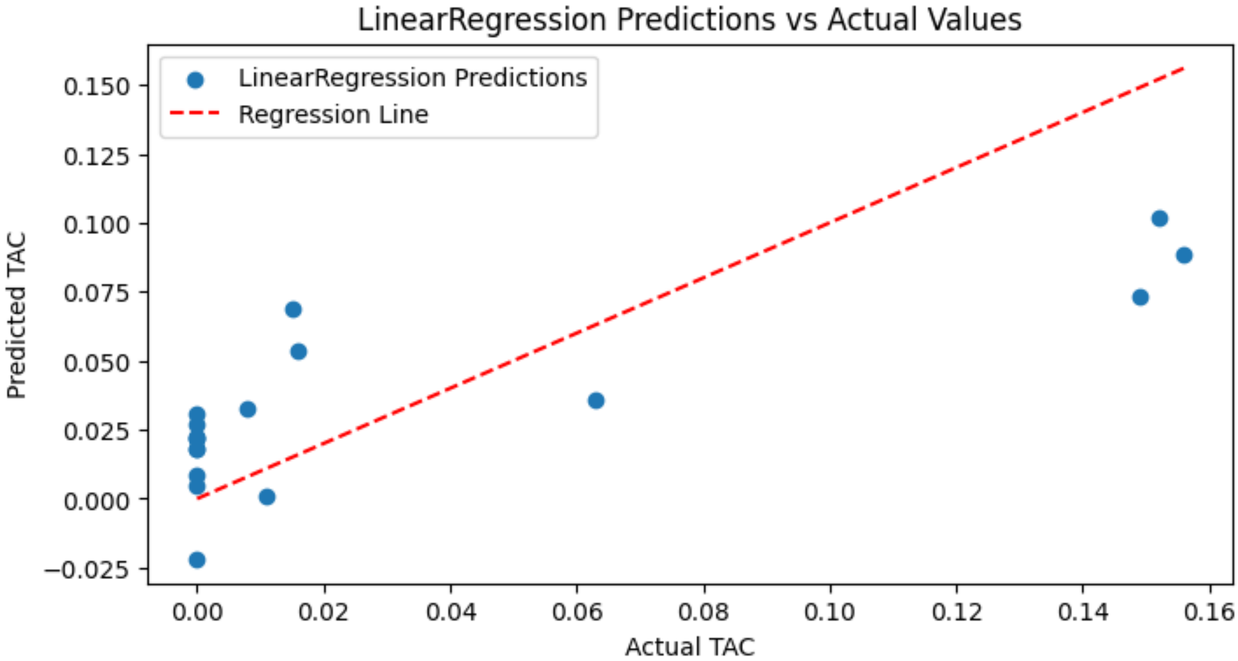
evaluation metrics for the Linear Regression model

```
# Calculate the Mean Squared Error (MSE) for the Linear Regression model
lr_mse = mean_squared_error(y_test, lr_y_pred)

# Calculate the accuracy (R^2 score) for the Linear Regression model
lr_accuracy = lr_model.score(X_test, y_test)

# Calculate the Mean Absolute Error (MAE) for the Linear Regression model
lr_mae = mean_absolute_error(y_test, lr_y_pred)

# Calculate the Root Mean Squared Error (RMSE) for the Linear Regression model
lr_rmse = mean_squared_error(y_test, lr_y_pred, squared=False)

# Print the evaluation metrics for the Linear Regression model
print("Linear Regression Mean Squared Error:", lr_mse)
print("Linear Regression Accuracy:", lr_accuracy)
print("Linear Regression Mean Absolute Error:", lr_mae)
print("Linear Regression Root Mean Squared Error:", lr_rmse)
```

```
Linear Regression Mean Squared Error: 0.0012722441371439222
Linear Regression Accuracy: 0.5927080982358588
Linear Regression Mean Absolute Error: 0.03008211672611925
Linear Regression Root Mean Squared Error: 0.035668531468844106
```

Plotting the actual vs predicted values for Linear Regression

```
plt.figure(figsize=(8, 4))
plt.scatter(y_test, lr_y_pred, label="LinearRegression Predictions")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='rec
plt.xlabel('Actual TAC')
plt.ylabel('Predicted TAC')
plt.title('LinearRegression Predictions vs Actual Values')
plt.legend()
plt.show()
```



Random Forest Regressor

Define the parameter grid for Random Forest to select best parameter

```
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
# Initialize variables to keep track of the best model and its MSE
best_rf_mse = float('inf')
best_rf_model = None
```

Iterate through all combinations of parameters in the parameter grid

```
for rf_params in itertools.product(*rf_param_grid.values()):
    # Initialize the Random Forest Regressor with the current parameters
    rf_regressor = RandomForestRegressor(n_estimators=rf_params[0], max_depth=r
                                         min_samples_split=rf_params[2], min_sam
    # Train the model using the training data
    rf_regressor.fit(X_train, y_train)

    # Make predictions on the test data
    rf_y_pred = rf_regressor.predict(X_test)

    # Calculate the Mean Squared Error (MSE) for the current Random Forest mode
```

```
    rf_mse = mean_squared_error(y_test, rf_y_pred)

    # If the current MSE is the best (lowest) found so far, update the best moc
    if rf_mse < best_rf_mse:
        best_rf_mse = rf_mse
        best_rf_model = rf_regressor
```

actual and predicted values for the test set for the best Random Forest model

```
print("Test Values:", y_test.values)
print("Predicted Values:", best_rf_model.predict(X_test))
```

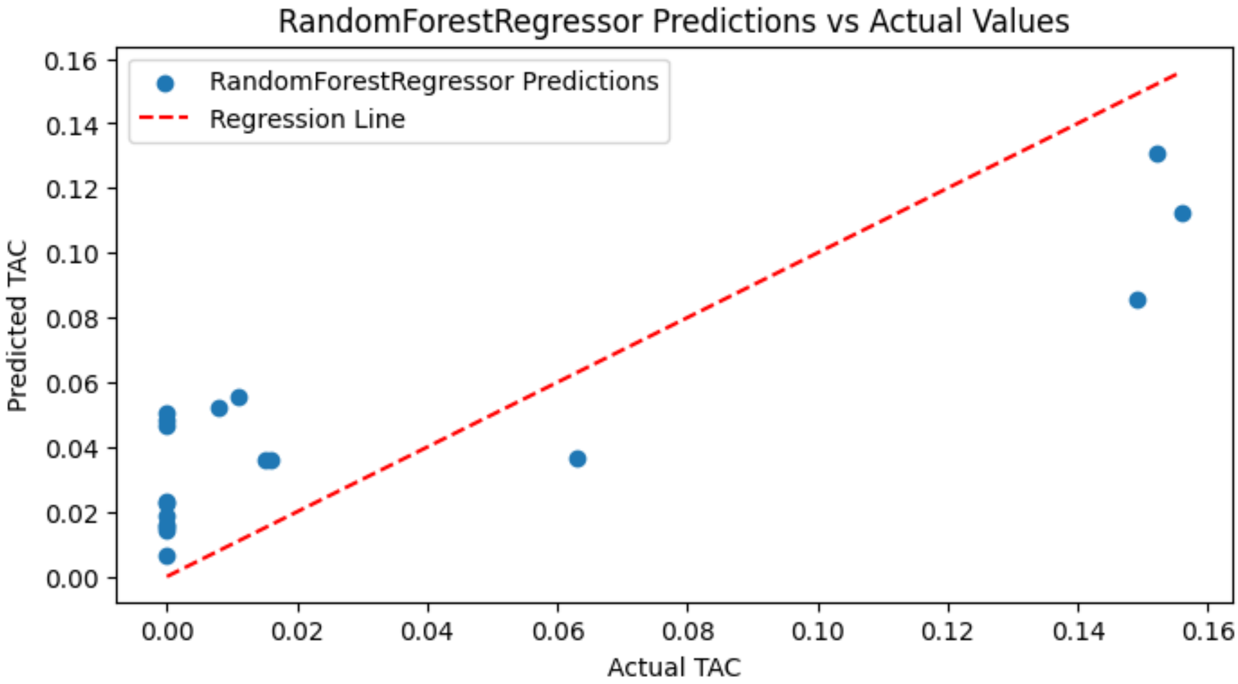
```
Test Values: [0.    0.    0.149 0.016 0.152 0.    0.063 0.156 0.    0.    0
 0.015 0.011 0.008 0.    0.    0.   ]
Predicted Values: [0.01896743 0.0141273  0.08575453 0.03605564 0.13089043 0
 0.03642184 0.1125596  0.02314497 0.04805771 0.02243782 0.0159873
 0.03621318 0.05547886 0.05205798 0.0504429  0.01536705 0.0063745 ]
```

```
print("RandomForestRegressor Parameters:", rf_params)
print("Mean Squared Error:", rf_mse)

# Calculate the accuracy (R^2 score) for the best Random Forest model
rf_accuracy = best_rf_model.score(X_test, y_test)

# Calculate the R-squared value for the best Random Forest model (redundant wit
rf_r_squared = best_rf_model.score(X_test, y_test)

# Calculate the Mean Absolute Error (MAE) for the best Random Forest model
rf_mae = mean_absolute_error(y_test, best_rf_model.predict(X_test))

# Calculate the Root Mean Squared Error (RMSE) for the best Random Forest model
rf_rmse = mean_squared_error(y_test, best_rf_model.predict(X_test), squared=Fal

# Print the evaluation metrics for the best Random Forest model
print("RandomForestRegressor Accuracy:", rf_accuracy)
print("RandomForestRegressor R-squared:", rf_r_squared)
print("RandomForestRegressor Mean Absolute Error:", rf_mae)
print("RandomForestRegressor Root Mean Squared Error:", rf_rmse)
```

```
RandomForestRegressor Parameters: (200, 20, 10, 4)
Mean Squared Error: 0.001706909229621004
RandomForestRegressor Accuracy: 0.6268043103371035
RandomForestRegressor R-squared: 0.6268043103371035
RandomForestRegressor Mean Absolute Error: 0.030325466095542066
RandomForestRegressor Root Mean Squared Error: 0.03414291926510778
```

Plotting the actual vs predicted values for the best Random Forest model

```
plt.figure(figsize=(8, 4))
plt.scatter(y_test, best_rf_model.predict(X_test), label="RandomForestRegressor
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='rec
plt.xlabel('Actual TAC')
plt.ylabel('Predicted TAC')
plt.title('RandomForestRegressor Predictions vs Actual Values')
plt.legend()
plt.show()
```



SVR (Support Vector Regressor)

Define the parameter grid for SVR to select best parameter

```
#
param_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto'],
    'epsilon': [0.1, 0.01, 0.001]
}

# Initialize variables to keep track of the best model and its MSE
best_mse = float('inf')
best_model = None
```

Iterate through all combinations of parameters in the parameter grid

```
for params in itertools.product(*param_grid.values()):
    # Initialize the SVR with the current parameters
    svr = SVR(kernel=params[0], C=params[1], gamma=params[2], epsilon=params[3]

    # Train the model using the training data
    svr.fit(X_train, y_train)

    # Make predictions on the test data
    svr_y_pred = svr.predict(X_test)

    # Calculate the Mean Squared Error (MSE) for the current SVR model
    mse = mean_squared_error(y_test, svr_y_pred)

    # If the current MSE is the best (lowest) found so far, update the best moc
    if mse < best_mse:
        best_mse = mse
        best_model = svr
```

actual and predicted values for the test set for the best SVR model

```
print("Test Values:", y_test.values)
print("Predicted Values:", best_model.predict(X_test))
```

```
Test Values: [0.    0.    0.149 0.016 0.152 0.    0.063 0.156 0.    0.    0
 0.015 0.011 0.008 0.    0.    0.   ]
Predicted Values: [ 0.00243757  0.00199656  0.08012489  0.02560676  0.11462
  0.0189109   0.11425469 -0.01072824 -0.00884745 -0.004068    0.001054
  0.07065531  0.00294499  0.01680592  0.01528225  0.00318726  0.00571527]
```

evaluation metrics for the best SVR model

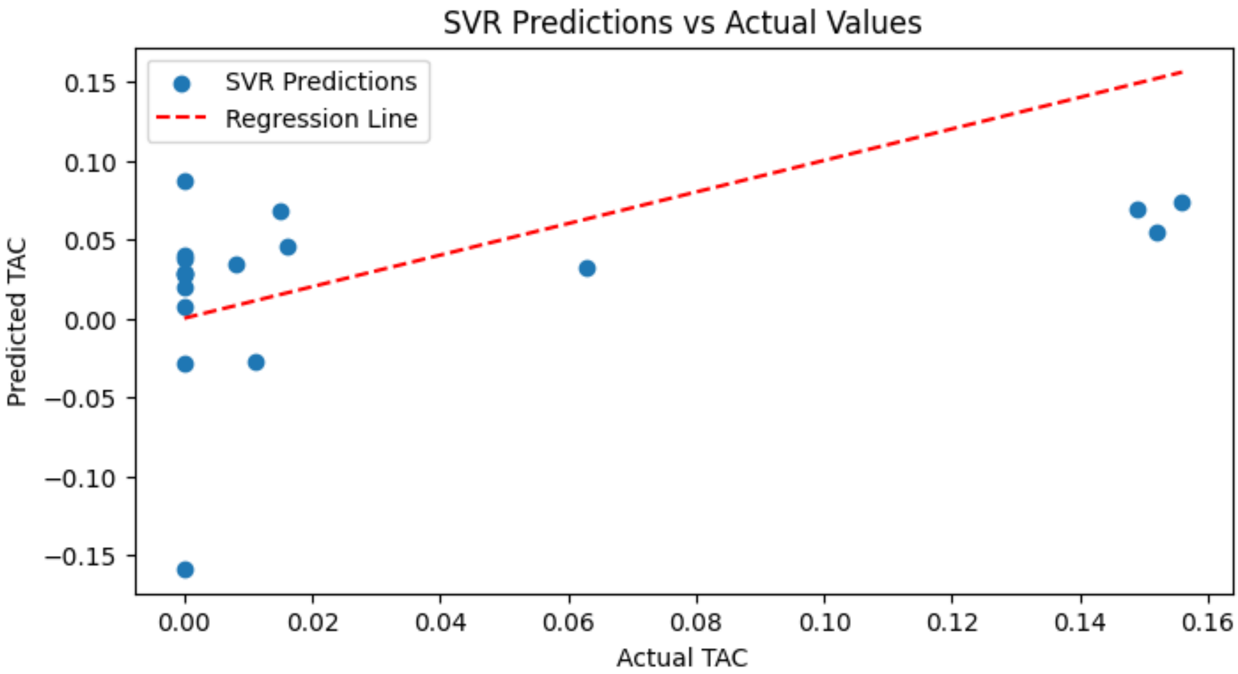
```
print("Parameters:", params)
print("Mean Squared Error:", mse)

# Calculate the accuracy (R^2 score) for the best SVR model
svr_accuracy = best_model.score(X_test, y_test)

# Calculate the R-squared value for the best SVR model (redundant with accuracy
svr_r_squared = best_model.score(X_test, y_test)

# Calculate the Mean Absolute Error (MAE) for the best SVR model
svr_mae = mean_absolute_error(y_test, best_model.predict(X_test))

# Calculate the Root Mean Squared Error (RMSE) for the best SVR model
svr_rmse = mean_squared_error(y_test, best_model.predict(X_test), squared=False

print("SVR Accuracy:", svr_accuracy)
```

```
print("SVR R-squared:", svr_r_squared)
print("SVR Mean Absolute Error:", svr_mae)
print("SVR Root Mean Squared Error:", svr_rmse)
```

```
Parameters: ('sigmoid', 10, 'auto', 0.001)
Mean Squared Error: 0.003832648769707266
SVR Accuracy: 0.7559258038671751
SVR R-squared: 0.7559258038671751
SVR Mean Absolute Error: 0.018648296255527424
SVR Root Mean Squared Error: 0.027611708216870017
```

Plotting the actual vs predicted values for the best SVR model

```
plt.figure(figsize=(8, 4))
plt.scatter(y_test, svr_y_pred, label="SVR Predictions")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='rec
plt.xlabel('Actual TAC')
plt.ylabel('Predicted TAC')
plt.title('SVR Predictions vs Actual Values')
plt.legend()
plt.show()
```



RESULT COmparison

Define classification thresholds based on provided metrics

```
legal_limit = 0.08
mean_tac = 0.065
max_tac = 0.443
inner_quartiles = [0.002, 0.029, 0.092]
```

Function to classify TAC levels based on thresholds

```
def classify_tac(tac_level):
    if tac_level < inner_quartiles[0]:
        return "Less Than Legal Limit"
    elif inner_quartiles[0] <= tac_level <= inner_quartiles[2]:
        return "About Legal Limit"
    else:
        return "Illegal: Heavy Alcohol"
```

Apply the classification function to the predictions of each model

```
Start coding or generate with AI.
```

```
lr_classified_tac = [classify_tac(tac_level) for tac_level in lr_y_pred]
rf_classified_tac = [classify_tac(tac_level) for tac_level in best_rf_model.pre
svr_classified_tac = [classify_tac(tac_level) for tac_level in best_model.predi
```

Create a DataFrame to store the results

```
results = pd.DataFrame({
    'Time': tac_data.iloc[X_test.index]['Time'],
    'Actual_TAC': y_test,
    'LR_Predicted_TAC': lr_y_pred,
    'RF_Predicted_TAC': best_rf_model.predict(X_test),
    'SVR_Predicted_TAC': best_model.predict(X_test),
    'LR_Classified_TAC': lr_classified_tac,
    'RF_Classified_TAC': rf_classified_tac,
    'SVR_Classified_TAC': svr_classified_tac
})
```

Sort the results by time

```
results.sort_values(by='Time', inplace=True)
print("\nResults DataFrame for Person", ":\n", results)
```

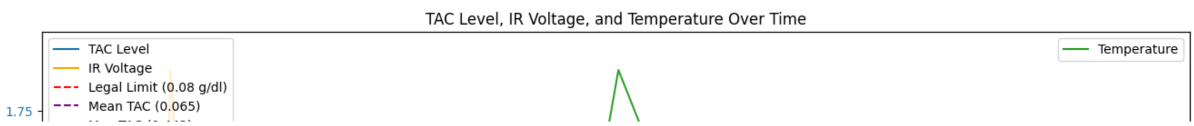
```
Results DataFrame for Person :
                    Time  Actual_TAC  LR_Predicted_TAC  RF_Predicted_TAC  \
0  2017-05-02 10:36:54       0.000          0.004841          0.018967
3  2017-05-02 11:20:57       0.000          0.017887          0.015987
4  2017-05-02 11:26:26       0.000          0.021852          0.006375
```

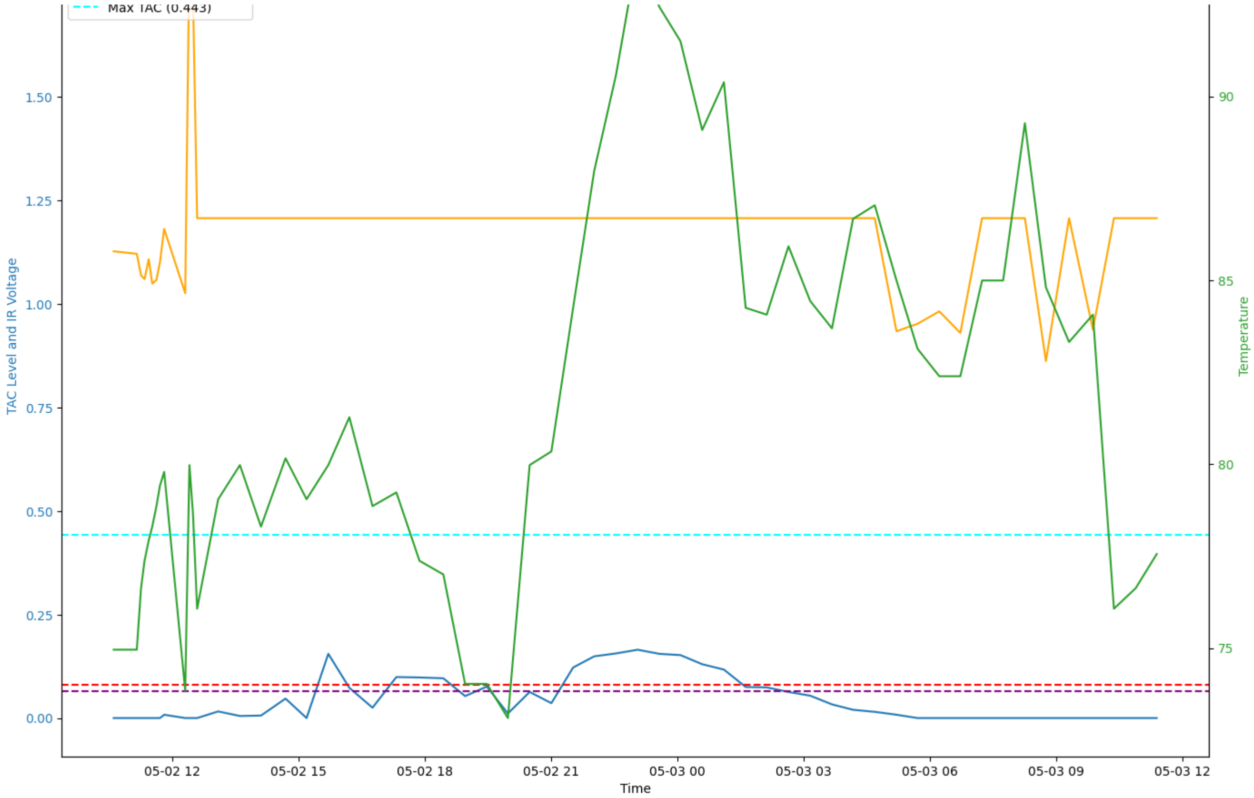
```
5  2017-05-02 11:31:56       0.000          0.022301          0.014127
6  2017-05-02 11:37:25       0.000          0.022071          0.015367
8  2017-05-02 11:48:23       0.008          0.032634          0.052058
12 2017-05-02 12:35:04       0.000          0.008847          0.048058
13 2017-05-02 13:05:36       0.016          0.053409          0.036056
17 2017-05-02 15:11:57       0.000          0.026994          0.050443
26 2017-05-02 19:58:50       0.011          0.001037          0.055479
27 2017-05-02 20:29:25       0.063          0.036046          0.036422
30 2017-05-02 22:01:47       0.149          0.073458          0.085755
31 2017-05-02 22:32:32       0.156          0.088400          0.112560
34 2017-05-03 00:04:46       0.152          0.101784          0.130890
43 2017-05-03 04:41:28       0.015          0.069027          0.036213
45 2017-05-03 05:42:33       0.000         -0.021743          0.023145
47 2017-05-03 06:43:37       0.000          0.030835          0.022438
55 2017-05-03 10:53:04       0.000          0.017760          0.046772

    SVR_Predicted_TAC       LR_Classified_TAC       RF_Classified_TAC  \
0            0.002438       About Legal Limit       About Legal Limit
3            0.001054       About Legal Limit       About Legal Limit
4            0.005715       About Legal Limit       About Legal Limit
5            0.001997       About Legal Limit       About Legal Limit
6            0.003187       About Legal Limit       About Legal Limit
8            0.016806       About Legal Limit       About Legal Limit
12          -0.008847       About Legal Limit       About Legal Limit
13           0.025607       About Legal Limit       About Legal Limit
17           0.015282       About Legal Limit       About Legal Limit
26           0.002945   Less Than Legal Limit       About Legal Limit
27           0.018911       About Legal Limit       About Legal Limit
30           0.080125       About Legal Limit       About Legal Limit
31           0.114255       About Legal Limit  Illegal: Heavy Alcohol
34           0.114621  Illegal: Heavy Alcohol  Illegal: Heavy Alcohol
43           0.070655       About Legal Limit       About Legal Limit
45          -0.010728   Less Than Legal Limit       About Legal Limit
47          -0.004068       About Legal Limit       About Legal Limit
55           0.008141       About Legal Limit       About Legal Limit

         SVR_Classified_TAC
0         About Legal Limit
3     Less Than Legal Limit
4         About Legal Limit
5     Less Than Legal Limit
6         About Legal Limit
8         About Legal Limit
12    Less Than Legal Limit
13        About Legal Limit
17        About Legal Limit
26        About Legal Limit
27        About Legal Limit
30        About Legal Limit
31   Illegal: Heavy Alcohol
34   Illegal: Heavy Alcohol
43        About Legal Limit
```

Graphical Virtualize of The Results

```
# Function to convert time from 'AM/PM' format to 24-hour format
```

```
from datetime import datetime

def convert_to_24_hour_format(time_string):
    return datetime.strptime(time_string, '%I:%M %p').strftime('%H:%M')

# Convert 'Time' and 'Date' columns to string format
tac_data['Time'] = tac_data['Time'].dt.strftime('%I:%M %p')
tac_data['Time'] = tac_data['Time'].apply(convert_to_24_hour_format)

# Convert 'Date' column to string format
tac_data['Date'] = tac_data['Date'].dt.strftime('%Y-%m-%d')

# Create 'Datetime' column by combining 'Date' and 'Time' columns
tac_data['Datetime'] = pd.to_datetime(tac_data['Date'] + ' ' + tac_data['Time'])

# Set 'Datetime' column as the index
tac_data.set_index('Datetime', inplace=True)

# Drop unnecessary 'Time' and 'Date' columns
tac_data.drop(columns=['Time', 'Date'], inplace=True)

# Plot TAC Level and IR Voltage on one y-axis, and Temperature on another y-axis
fig, ax1 = plt.subplots(figsize=(14, 10))

color = 'tab:blue'
ax1.set_xlabel('Time')
ax1.set_ylabel('TAC Level and IR Voltage', color=color)
ax1.plot(tac_data.index.to_numpy(), tac_data['TAC Level'].to_numpy(), label='TAC
ax1.plot(tac_data.index.to_numpy(), tac_data['IR Voltage'].to_numpy(), label='IR
ax1.tick_params(axis='y', labelcolor=color)

# Create a second y-axis for Temperature
ax2 = ax1.twinx()
color = 'tab:green'
ax2.set_ylabel('Temperature', color=color)
ax2.plot(tac_data.index.to_numpy(), tac_data['Temperature'].to_numpy(), label='T
ax2.tick_params(axis='y', labelcolor=color)

# Add horizontal lines to indicate legal limit, mean TAC, and max TAC
ax1.axhline(y=legal_limit, color='red', linestyle='--', label='Legal Limit (0.08
ax1.axhline(y=mean_tac, color='purple', linestyle='--', label=f'Mean TAC ({mean_
ax1.axhline(y=max_tac, color='cyan', linestyle='--', label=f'Max TAC ({max_tac})

# Add legends for the plots
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

fig.tight_layout()
plt.title('TAC Level, IR Voltage, and Temperature Over Time')
plt.show()
```




Max TAC (0.443)
TAC Level and IR Voltage
Temperature
Time
1.50
1.25
1.00
0.75
0.50
0.25
0.00
90
85
80
75
05-02 12
05-02 15
05-02 18
05-02 21
05-03 00
05-03 03
05-03 06
05-03 09
05-03 12