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

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Importing the Dataset

```
data_test = pd.read_csv('/content/drive/MyDrive/project file/test.csv')
data_train = pd.read_csv('/content/drive/MyDrive/project file/train.csv')
```

```
data_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2947 entries, 0 to 2946
Columns: 563 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), int64(1), object(1)
memory usage: 12.7+ MB
```

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Columns: 563 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), int64(1), object(1)
memory usage: 31.6+ MB
```

Display Top 5 Rows of The Dataset

```
data_train.head()
```

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc mad()·
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.98318
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.9749
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.96366
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.98275
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.97967

5 rows × 563 columns

Check Last 5 Rows of The Dataset

```
data_train.tail()
```

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBody ma
7347	0.299665	-0.057193	-0.181233	-0.195387	0.039905	0.077078	-0.282301	0.0
7348	0.273853	-0.007749	-0.147468	-0.235309	0.004816	0.059280	-0.322552	-0.0
7349	0.273387	-0.017011	-0.045022	-0.218218	-0.103822	0.274533	-0.304515	-0.0
7350	0.289654	-0.018843	-0.158281	-0.219139	-0.111412	0.268893	-0.310487	-0.0
7351	0.351503	-0.012423	-0.203867	-0.269270	-0.087212	0.177404	-0.377404	-0.0

5 rows × 563 columns

Find Shape of Our Dataset (Number of Rows And Number of Columns)

```
data_train.shape
```

```
(7352, 563)
```

```
print("Number of Rows",data_train.shape[0])
print("Number of columns",data_train.shape[1])
```

```
Number of Rows 7352
Number of columns 563
```

Taking Care of Duplicate Values

```
data_train.duplicated().any()
```

```
False
```

```
duplicated_columns = data_train.columns[data_train.T.duplicated()].tolist()
```

```
len(duplicated_columns)
```

```
21
```

```
data_train = data_train.drop(duplicated_columns,axis=1)
```

```
data_train.shape
```

```
(7352, 542)
```

Taking Care of Missing Values

```
data_train.isnull().sum()
```

```
tBodyAcc-mean()-X       0
tBodyAcc-mean()-Y       0
tBodyAcc-mean()-Z       0
tBodyAcc-std()-X        0
tBodyAcc-std()-Y        0
                       ..
angle(X,gravityMean)    0
angle(Y,gravityMean)    0
angle(Z,gravityMean)    0
subject                 0
Activity                0
Length: 542, dtype: int64
```

```
data_train['Activity'] = data_train['Activity'].astype('category')
```

Store Feature Matrix In X and Response(Target) In Vector y

```
X = data_train.drop('Activity',axis=1)
y= data_train['Activity']
```

```
y
```

```
0               STANDING
1               STANDING
2               STANDING
3               STANDING
4               STANDING
              ...
7347    WALKING_UPSTAIRS
```

```
7348    WALKING_UPSTAIRS
7349    WALKING_UPSTAIRS
7350    WALKING_UPSTAIRS
7351    WALKING_UPSTAIRS
Name: Activity, Length: 7352, dtype: category
Categories (6, object): ['LAYING', 'SITTING', 'STANDING', 'WALKING', 'WALKING_DOWNSTAIRS',
                         'WALKING_UPSTAIRS']
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

```
y
```

```
array([2, 2, 2, ..., 5, 5, 5])
```

Splitting The Dataset Into The Training Set And Test Set

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,
                                               random_state=42)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
log  = LogisticRegression()
log.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

```
y_pred1 = log.predict(X_test)
accuracy_score(y_test,y_pred1)
```

```
0.9809653297076818
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(X_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_pred2 = rf.predict(X_test)
accuracy_score(y_test,y_pred2)
```

```
0.9836845683208701
```

Feature Selection

Filter Method

```
from sklearn.feature_selection import SelectKBest,f_classif
```

```
k=200
selector = SelectKBest(f_classif,k=k)
X_train_selected = selector.fit_transform(X_train,y_train)
X_test_selected = selector.transform(X_test)
```

```
selected_indices=selector.get_support(indices=True)
selected_features = X_train.columns[selected_indices]
print(len(selected_features))
```

```
    200
```

Wrapper Method

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
```

```
estimator = RandomForestClassifier()
```

```
k=100
rfe_selector = RFE(estimator,n_features_to_select=k)
X_train_selected_rfe = rfe_selector.fit_transform(X_train_selected,y_train)
X_test_selected_rfe = rfe_selector.transform(X_test_selected)

selected_indices_rfe = rfe_selector.get_support(indices=True)
selected_features_rfe = selected_features[selected_indices_rfe]
print(selected_features_rfe)
```

```
    Index(['tBodyAcc-std()-X', 'tBodyAcc-mad()-X', 'tBodyAcc-max()-X',
           'tBodyAcc-sma()', 'tBodyAcc-energy()-X', 'tBodyAcc-iqr()-X',
           'tBodyAcc-entropy()-X', 'tGravityAcc-mean()-X', 'tGravityAcc-mean()-Y',
           'tGravityAcc-max()-X', 'tGravityAcc-max()-Y', 'tGravityAcc-min()-X',
           'tGravityAcc-min()-Y', 'tGravityAcc-energy()-X',
           'tGravityAcc-energy()-Y', 'tBodyAccJerk-std()-X',
           'tBodyAccJerk-std()-Y', 'tBodyAccJerk-std()-Z', 'tBodyAccJerk-mad()-X',
           'tBodyAccJerk-mad()-Y', 'tBodyAccJerk-mad()-Z', 'tBodyAccJerk-max()-X',
           'tBodyAccJerk-max()-Z', 'tBodyAccJerk-sma()', 'tBodyAccJerk-energy()-X',
           'tBodyAccJerk-energy()-Y', 'tBodyAccJerk-iqr()-Z',
           'tBodyAccJerk-entropy()-X', 'tBodyAccJerk-entropy()-Z',
           'tBodyGyro-std()-X', 'tBodyGyro-std()-Y', 'tBodyGyro-std()-Z',
           'tBodyGyro-mad()-X', 'tBodyGyro-mad()-Y', 'tBodyGyro-mad()-Z',
           'tBodyGyro-max()-X', 'tBodyGyro-min()-X', 'tBodyGyro-iqr()-X',
           'tBodyGyro-iqr()-Y', 'tBodyGyro-iqr()-Z', 'tBodyGyroJerk-std()-X',
           'tBodyGyroJerk-std()-Z', 'tBodyGyroJerk-mad()-X',
           'tBodyGyroJerk-mad()-Z', 'tBodyGyroJerk-max()-X',
           'tBodyGyroJerk-min()-X', 'tBodyGyroJerk-sma()', 'tBodyGyroJerk-iqr()-X',
           'tBodyGyroJerk-iqr()-Z', 'tBodyAccMag-std()', 'tBodyAccMag-mad()',
           'tBodyAccMag-energy()', 'tBodyAccJerkMag-mean()',
           'tBodyAccJerkMag-mad()', 'tBodyAccJerkMag-energy()',
           'tBodyAccJerkMag-iqr()', 'tBodyAccJerkMag-entropy()',
           'tBodyGyroJerkMag-mean()', 'tBodyGyroJerkMag-iqr()',
           'fBodyAcc-mean()-X', 'fBodyAcc-std()-X', 'fBodyAcc-mad()-X',
           'fBodyAcc-max()-X', 'fBodyAcc-max()-Y', 'fBodyAcc-energy()-X',
           'fBodyAcc-bandsEnergy()-1,8', 'fBodyAcc-bandsEnergy()-1,16',
           'fBodyAcc-bandsEnergy()-1,24', 'fBodyAcc-bandsEnergy()-1,8.1',
           'fBodyAccJerk-mean()-Z', 'fBodyAccJerk-std()-X', 'fBodyAccJerk-std()-Y',
           'fBodyAccJerk-std()-Z', 'fBodyAccJerk-mad()-Z', 'fBodyAccJerk-max()-Y',
           'fBodyAccJerk-sma()', 'fBodyAccJerk-energy()-X',
           'fBodyAccJerk-energy()-Y', 'fBodyAccJerk-bandsEnergy()-1,8',
           'fBodyAccJerk-bandsEnergy()-1,16', 'fBodyAccJerk-bandsEnergy()-1,24',
           'fBodyAccJerk-bandsEnergy()-1,24.1', 'fBodyGyro-mean()-X',
           'fBodyGyro-std()-X', 'fBodyGyro-std()-Y', 'fBodyGyro-std()-Z',
           'fBodyGyro-mad()-X', 'fBodyGyro-mad()-Y', 'fBodyGyro-max()-X',
           'fBodyGyro-max()-Z', 'fBodyGyro-entropy()-X', 'fBodyAccMag-mean()',
           'fBodyAccMag-std()', 'fBodyAccMag-mad()', 'fBodyAccMag-max()',
       'fBodyAccMag-energy()', 'fBodyBodyAccJerkMag-std()',
       'fBodyBodyAccJerkMag-max()', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)'],
      dtype='object')
```

```
print(len(selected_features_rfe))
```

```
100
```

```
rf = RandomForestClassifier()
```

```
rf.fit(X_train_selected_rfe,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_pred_rf = rf.predict(X_test_selected_rfe)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test,y_pred_rf)
```

```
0.9782460910944936
```

```
import joblib
```

```
joblib.dump(rf,"model_rfe")
```

```
['model_rfe']
```

```
joblib.dump(selector,"k_best_selector")
```

```
['k_best_selector']
```

```
joblib.dump(rfe_selector,"rfe_selector")
```

```
['rfe_selector']
```

```
data_test=data_test.drop("Activity",axis=1)
```

```
duplicated_columns = data_test.columns[data_test.T.duplicated()].to_list()
```

```
data_test = data_test.drop(duplicated_columns,axis=1)
```

```
model = joblib.load('model_rfe')
```

```
selector = joblib.load('k_best_selector')
```

```
rfe_selector = joblib.load('rfe_selector')
```

```
selector=selector.transform(data_test)
```

```
X_test_selected_rfe = rfe_selector.transform(selector)
```

```
model.predict(X_test_selected_rfe)
```

```
array([2, 2, 2, ..., 5, 5, 5])
```

GUI

```
import tkinter as tk
from tkinter import filedialog
```

```
import pandas as pd
import joblib
from tkinter import messagebox

def open_file():
    filepath=filedialog.askopenfile(filetypes=[("CSV Files",".csv")])
    if filepath:
        try:
            data_train=pd.read_csv(filepath)
            process_data_train(data_train)
        except Exception as e:
            messagebox.showerror("Error",f"Failed to open file {e}")

def process_data(data_train):
    # Find columns with the same values
    #data= data.drop("Activity",axis=1)
    duplicated_columns = data_train.columns[data_train.T.duplicated()].tolist()
    # Remove columns with the same values

    data_test = data_train.drop(duplicated_columns, axis=1)

    model = joblib.load("model_rfe")
    # Load the SelectKBest object from the file
    selector = joblib.load('k_best_selector')
    rfe_selector = joblib.load('rfe_selector')

    # Transform the new data using the loaded SelectKBest object
    X_test_selected = selector.transform(data_test)

    # Transform the new data using the loaded RFE object
    X_test_selected_rfe = rfe_selector.transform(X_test_selected)
    y_pred=model.predict(X_test_selected_rfe)
    # standing : 0, sitting : 1,laying : 2, WALKING_DOWNSTAIRS: 3,
    # walking_upstairs:4,walking : 5
    y_pred = pd.Series(y_pred)
    y_pred = y_pred.map({0: 'Standing',1:'Sitting',2:'Laying',
                        3: 'Walking_downstairs',4: 'Walking_upstairs',
                        5:"Walking"})
    data_train['Predicted_target']=y_pred
    save_file(data_train)

def save_file(data_train):
    savepath=filedialog.asksaveasfilename(defaultextension=".csv",
                              filetypes=[("CSV Files",".csv")])
    if savepath:
        try:
            data.to_csv(savepath)
            messagebox.showinfo("Success","File Saved Successfully")
        except Exception as e:
            messagebox.showerror("Error",f"Failed to save file:{e}")


# Create a Tkinter GUI

root = tk.Tk()

root.title("Classification")

root.geometry("200x200")

button1 = tk.Button(root,text="Open CSV File",
                   width=15,
                   height=2,
                   background="lightgreen",
                   activebackground="lightblue",
                   font=("Arial",11,"bold"),
                   command=open_file)

button1.pack(pady=50)

root.mainloop()
```