LOADING AND PREPROCESSING THE DATA

In [89]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
#------------------------------LOADING DATA-----------------------------------------------------
-----------
os.getcwd()
os.chdir('C:\\Users\\Serving Minds\\Desktop\\Exa-mobility')
df=pd.read_csv('steady position with single stand.csv')
df.head()
```

Out[89]:

	raw_ax	raw_ay	raw_az	cal_ax	cal_ay	cal_az	raw_gx	raw_gy	raw_gz	cal_gx	...	filtered_my	filtered_mz	time_sec	
0	491.0	1047.0	16818.0	0.029968	0.063904	1.026489	181.0	348.0	57.0	0.011047	...	-0.324943	-0.930326	1	6
1	450.0	1000.0	16739.0	0.027466	0.061035	1.021667	177.0	341.0	73.0	0.010803	...	-0.342173	-0.925590	1	8
2	508.0	1022.0	16803.0	0.031006	0.062378	1.025574	179.0	346.0	56.0	0.010925	...	-0.358125	-0.921371	2	1
3	466.0	1013.0	16760.0	0.028442	0.061829	1.022949	170.0	350.0	47.0	0.010376	...	-0.348576	-0.923586	2	1
4	456.0	969.0	16747.0	0.027832	0.059143	1.022156	189.0	355.0	48.0	0.011536	...	-0.298063	-0.938176	2	1

5 rows × 35 columns

In [90]:

```
df['State'] = pd.Series(['steady']*1818)
df.head()
```

Out[90]:

	raw_ax	raw_ay	raw_az	cal_ax	cal_ay	cal_az	raw_gx	raw_gy	raw_gz	cal_gx	...	filtered_mz	time_sec	yaw	pi
0	491.0	1047.0	16818.0	0.029968	0.063904	1.026489	181.0	348.0	57.0	0.011047	...	-0.930326	1	-63.7197	41.7
1	450.0	1000.0	16739.0	0.027466	0.061035	1.021667	177.0	341.0	73.0	0.010803	...	-0.925590	1	-86.9113	68.2
2	508.0	1022.0	16803.0	0.031006	0.062378	1.025574	179.0	346.0	56.0	0.010925	...	-0.921371	2	-12.0316	13.1
3	466.0	1013.0	16760.0	0.028442	0.061829	1.022949	170.0	350.0	47.0	0.010376	...	-0.923586	2	-14.2523	72.3
4	456.0	969.0	16747.0	0.027832	0.059143	1.022156	189.0	355.0	48.0	0.011536	...	-0.938176	2	-14.0744	13.1

5 rows × 36 columns

In [91]:

```
df['raw_ax'].count
```

Out[91]:

```
<bound method Series.count of 0        491.0
1         450.0
2         508.0
3         466.0
4         456.0
          ...
```

```
         ...
1813   -4386.0
1814   -4411.0
1815   -4402.0
1816   -4416.0
1817   -4369.0
Name: raw_ax, Length: 1818, dtype: float64>
```

In [92]:

```
df['State'] = pd.Series(['steady']*1818)
```

In [93]:

```
df1=pd.read_csv('steady pos with double stand.csv')
```

In [94]:

```
df1['raw_ax'].count
```

Out[94]:

```
<bound method Series.count of 0       473.0
1       505.0
2       494.0
3       537.0
4       459.0
        ...
1780    461.0
1781    441.0
1782    452.0
1783    446.0
1784    461.0
Name: raw_ax, Length: 1785, dtype: float64>
```

In [95]:

```
df1['State'] = pd.Series(['steady']*1785)
```

In [96]:

```
df2=pd.read_csv('fast tilt.csv')
```

In [97]:

```
df2['raw_ax'].count
```

Out[97]:

```
<bound method Series.count of 0      498.0
1      517.0
2      521.0
3      520.0
4      488.0
       ...
851   -785.0
852   -360.0
853   -693.0
854   -888.0
855   -917.0
Name: raw_ax, Length: 856, dtype: float64>
```

In [98]:

```
df2['State'] = pd.Series(['moving']*856)
```

In [99]:

```
df3=pd.read_csv('slow tilt.csv')
```

In [100]:

```
df3['State'] = pd.Series(['moving']*816)
```

In [101]:

```
df5=pd.read_csv('movement with bump.csv')
```

In [102]:

```
df5['State'] = pd.Series(['moving']*553)
```

In [103]:

```
pieces = (df,df1,df2,df3,df5)
```

In [104]:

```
df_final = pd.concat(pieces, ignore_index = True)
df_final.head(5)
```

Out[104]:

	raw_ax	raw_ay	raw_az	cal_ax	cal_ay	cal_az	raw_gx	raw_gy	raw_gz	cal_gx	...	filtered_mz	time_sec	yaw	pi
0	491.0	1047.0	16818.0	0.029968	0.063904	1.026489	181.0	348.0	57.0	0.011047	...	-0.930326	1	-63.7197	41.7
1	450.0	1000.0	16739.0	0.027466	0.061035	1.021667	177.0	341.0	73.0	0.010803	...	-0.925590	1	-86.9113	68.2
2	508.0	1022.0	16803.0	0.031006	0.062378	1.025574	179.0	346.0	56.0	0.010925	...	-0.921371	2	-12.0316	13.1
3	466.0	1013.0	16760.0	0.028442	0.061829	1.022949	170.0	350.0	47.0	0.010376	...	-0.923586	2	-14.2523	72.3
4	456.0	969.0	16747.0	0.027832	0.059143	1.022156	189.0	355.0	48.0	0.011536	...	-0.938176	2	-14.0744	13.1

5 rows × 36 columns

In [105]:

```
df_final['raw_ax'].count
```

Out[105]:

```
<bound method Series.count of 0        491.0
1        450.0
2        508.0
3        466.0
4        456.0
         ...
5823    -740.0
5824    -743.0
5825    -822.0
5826    -864.0
5827    -862.0
Name: raw_ax, Length: 5828, dtype: float64>
```

In [106]:

```
df_final = df_final.sample(frac=1).reset_index(drop=True)
```

In [107]:

```
df_final.head(10)
```

Out[107]:

	raw_ax	raw_ay	raw_az	cal_ax	cal_ay	cal_az	raw_gx	raw_gy	raw_gz	cal_gx	...	filtered_mz	time_sec	yaw	p
0	584.0	1058.0	16780.0	0.035645	0.064575	1.024170	167.0	355.0	58.0	0.010193	...	-0.930420	84	43.8966	-5.
1	-6672.0	4373.0	14859.0	-0.407227	0.266907	0.906921	1217.0	2203.0	26.0	0.074280	...	-0.945092	20	80.7579	52.
2	222.0	654.0	16782.0	0.013550	0.039917	1.024292	123.0	255.0	67.0	0.007507	...	-0.919936	135	81.0230	50.
3	485.0	1073.0	16849.0	0.029602	0.065491	1.028381	192.0	331.0	-8.0	0.011719	...	-0.925479	100	103.8908	41.
4	3316.0	-928.0	16182.0	0.202393	-0.056641	0.987671	-1595.0	-2870.0	-7.0	-0.097351	...	-0.942973	27	46.9087	34.
5	9179.0	-4494.0	13250.0	0.560242	-0.274292	0.808716	229.0	388.0	185.0	0.013977	...	-0.966161	74	71.9153	77.
6	-53.0	826.0	16808.0	-0.003235	0.050415	1.025879	134.0	238.0	56.0	0.008179	...	-0.925879	7	29.2741	67.
7	498.0	973.0	16806.0	0.030396	0.059387	1.025757	173.0	356.0	66.0	0.010559	...	-0.925581	302	13.9791	18.
8	457.0	984.0	16825.0	0.027893	0.060059	1.026917	171.0	343.0	42.0	0.010437	...	-0.940492	153	96.1946	33.
9	-4377.0	3067.0	16034.0	-0.267151	0.187195	0.978638	178.0	351.0	62.0	0.010864	...	-0.928123	61	155.4320	74.

10 rows × 36 columns

In [108]:

```
x=df_final.iloc[:,18:27].values
y=df_final.iloc[:,-1].values
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
y=le.fit_transform(y)
print(y)
```

```
[1 0 0 ... 1 1 1]
```

SPLITTING INTO TEST AND TRAIN SET

In [109]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In [110]:

```
# from sklearn.model_selection import GridSearchCV
# from sklearn.ensemble import RandomForestClassifier
# # Create the parameter grid based on the results of random search
# param_grid = {
#     'bootstrap': [True],
#     'max_depth': [80, 90, 100, 110],
#     'max_features': [2, 3],
#     'min_samples_leaf': [3, 4, 5],
#     'min_samples_split': [8, 10, 12],
#     'n_estimators': [100, 200, 300, 1000]
# }
# # Create a based model
# rf = RandomForestClassifier()
# # Instantiate the grid search model
# grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
#                           cv = 3, n_jobs = -1, verbose = 2)
```

In [111]:

```
# grid_search.fit(X_train,Y_train)
```

```
# print(grid_search.best_params_)
```

HYPERPARAMETER TUNING

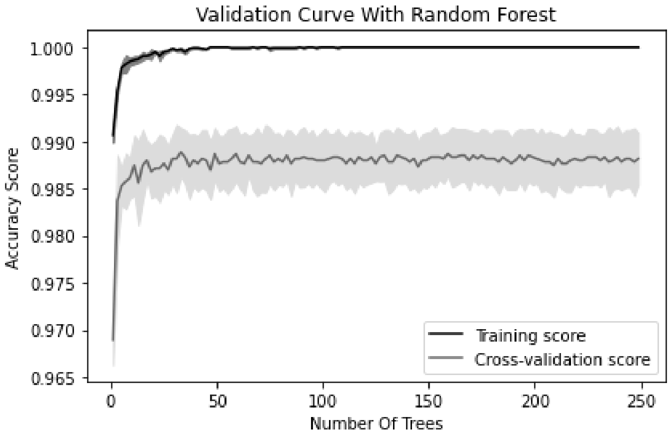
In [112]:

```
# Load libraries
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import validation_curve
```

In [113]:

```
param_range = np.arange(1, 250, 2)
train_scores, test_scores = validation_curve(RandomForestClassifier(),
                                             x,
                                             y,
                                             param_name="n_estimators",
                                             param_range=param_range,
                                             cv=3,
                                             scoring="accuracy",
                                             n_jobs=-1)
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(param_range, train_mean, label="Training score", color="black")
plt.plot(param_range, test_mean, label="Cross-validation score", color="dimgrey")
plt.fill_between(param_range, train_mean - train_std, train_mean + train_std, color="gray")
plt.fill_between(param_range, test_mean - test_std, test_mean + test_std, color="gainsboro")
plt.title("Validation Curve With Random Forest")
plt.xlabel("Number Of Trees")
plt.ylabel("Accuracy Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```



RANDOMISED SEARCH CV

In [114]:

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 100, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
```

```
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100], 'max_features': ['auto', 'sqrt'], 'max
_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'm
in_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

In [115]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv
= 3, verbose=2, random_state=42, n_jobs = -1)
#Fit the random search model
rf_random.fit(X_train, Y_train)
rf_random.best_params_
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   26.8s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  3.7min finished
```

Out[115]:

```
{'n_estimators': 30,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 60,
 'bootstrap': False}
```

Building new classsifier with tuned hyperparameters

In [116]:

```
model = RandomForestClassifier(n_estimators=30,
                               bootstrap = True,
                               max_features = 'auto',
                               min_samples_split=5,
                               min_samples_leaf=1,
                               max_depth=80)

# Fit on training data
model.fit(X_train,Y_train)
```

Out[116]:

```
RandomForestClassifier(max_depth=80, min_samples_split=5, n_estimators=30)
```

In [117]:

```
predictions = model.predict(X_test)
print(predictions)
```

```
[0 0 1 ... 1 1 1]
```

In [118]:

```
x = pd.DataFrame({'True values': Y_test,
                  'Predicted values': predictions})
print(x)
```

```
      True values  Predicted values
0               0                 0
1               0                 0
2               1                 1
3               1                 1
4               0                 0
...           ...               ...
1161            1                 1
1162            1                 1
1163            1                 1
1164            1                 1
1165            1                 1

[1166 rows x 2 columns]
```

Evaluating the model

In [119]:

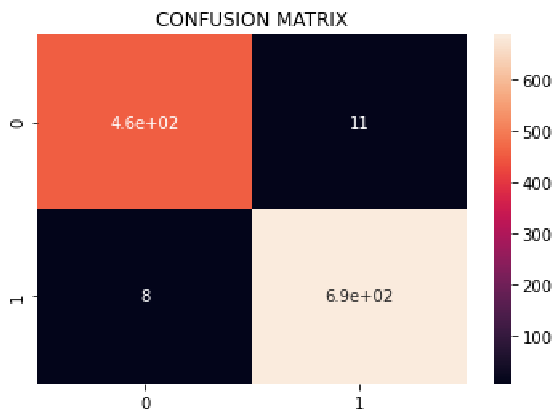
```
from sklearn.metrics import confusion_matrix
cf_matrix=confusion_matrix(Y_test, predictions)
```

In [120]:

```
import seaborn as sns
ax = plt.axes()
sns.heatmap(cf_matrix, annot=True)
ax.set_title('CONFUSION MATRIX')
```

Out[120]:

```
Text(0.5, 1.0, 'CONFUSION MATRIX')
```



ACCURACY OF THE MODEL

In [121]:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,predictions))
```

```
0.983704974271012
```