Mini Project

Contents

- Exploratory Data Analysis
- Data preparation
- Model Helper Functions
- XGBoost Classifier
  - Feature Importance
- Multilayer Perceptron
- Accuracy vs Number of Features
- PCA

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
import shap
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.decomposition import PCA
from sklearn import preprocessing
from matplotlib import cm
```

In [2]:
```
data = pd.read_csv("../Data/DataSet_HAR.csv")
subjects = data['subject'].drop_duplicates()
data.set_index('subject', inplace = True)
```

EDA

In [3]:
```
data.shape
```

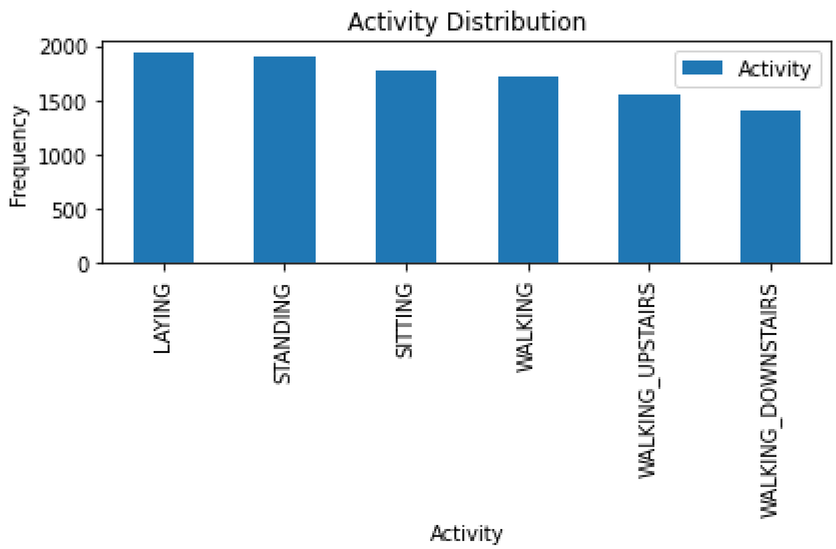
Out[3]: (10299, 562)

In [4]:
```
data.isnull().sum().sum()
```

Out[4]: 0

In [5]:
```
fig = plt.figure(figsize=(15,8))
data[['Activity']].apply(pd.value_counts).plot(kind='bar')
plt.xlabel('Activity')
plt.ylabel('Frequency')
plt.title('Activity Distribution')
#plt.show()
plt.tight_layout()
plt.savefig("Activity Distribution")
```

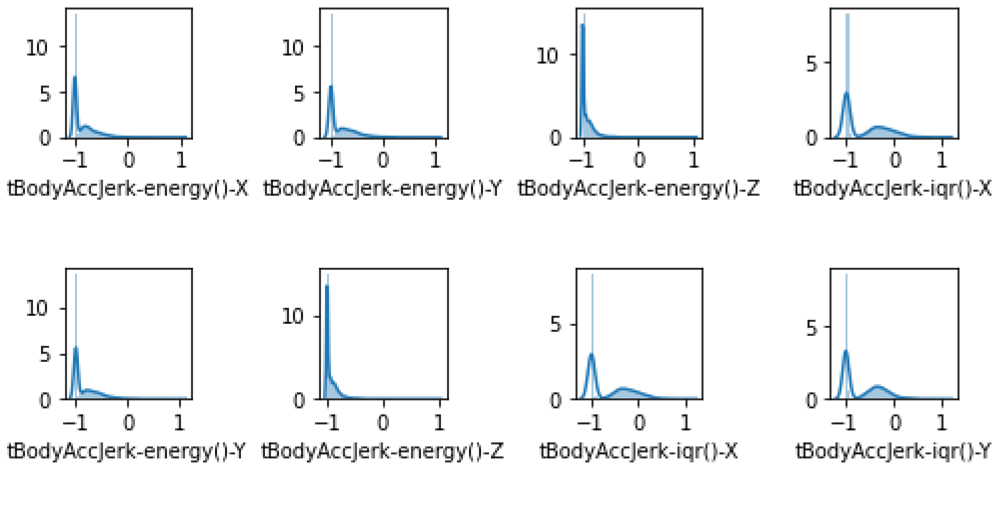
```
<Figure size 1080x576 with 0 Axes>
```



In [6]:
```
columns = data.columns
columns = columns[0:561]
```

In [7]:

```
for i in range(47):
    fig, axs = plt.subplots(3,4)
    fig.subplots_adjust(hspace=1,wspace = 1)
    fig.set_size_inches(8,6)
    for j in range(3):
        for k in range(4):
            sns.distplot(data[columns[i*12 + j + k]], ax = axs[j,k])
    #fig.savefig('../Plots/dist_' + str(i) + '.png', dpi=100)
```



In [8]:

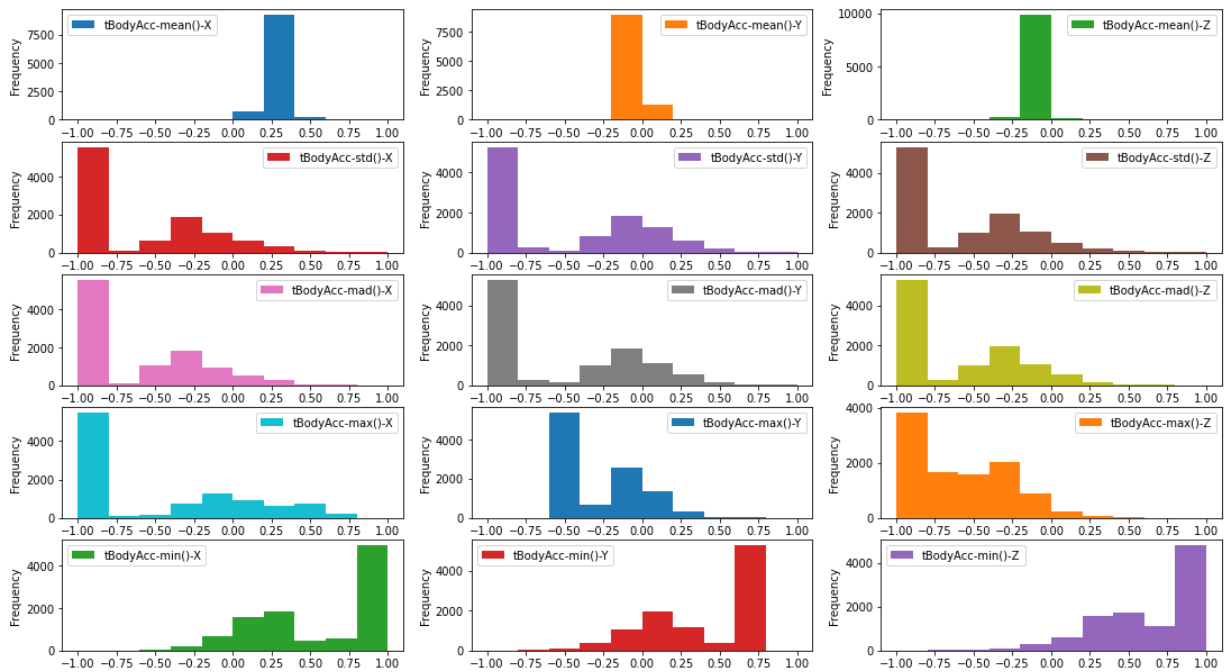
```
for i in range(len(columns[0:47])):
    fig, axs = plt.subplots(3,4)
    fig.subplots_adjust(hspace=1,wspace = 1)
    fig.set_size_inches(8,6)
    for j in range(3):
        for k in range(4):
            ax = axs[j,k]
            data.groupby('Activity')[columns[i]].hist(alpha=0.4, ax=ax)
            ax.set_title(columns[i] + '_hist')
            ax.legend()

   # fig.savefig('../Plots/classwise_dist_' + str(i) + '.png', dpi=100)
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.

No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```

In [9]:

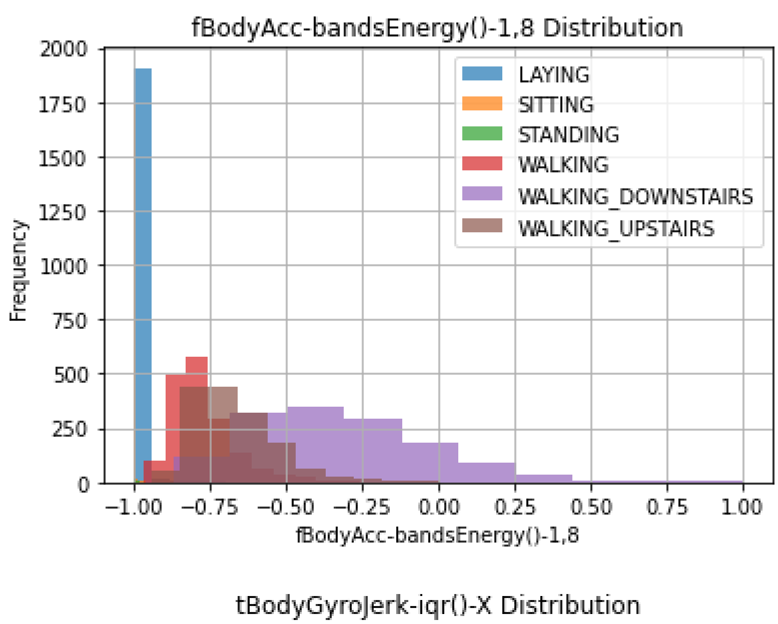
```
fig, axes = plt.subplots(nrows=5, ncols=3)
data[data.columns[0:15]].plot(subplots = True, ax = axes, kind = 'hist', bins = 1
fig.set_size_inches(18.5, 10.5)
plt.savefig('body_Acc.png')
```

```
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two m
inor releases later. Use ax.get_subplotspec().rowspan.start instead.

The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two m
inor releases later. Use ax.get_subplotspec().colspan.start instead.

The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two m
inor releases later. Use ax.get_subplotspec().rowspan.start instead.

The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two m
inor releases later. Use ax.get_subplotspec().colspan.start instead.
```

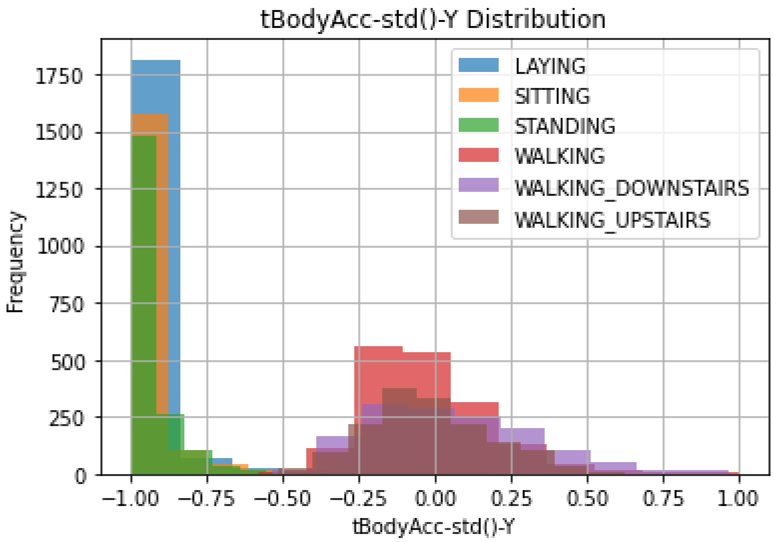


In [10]:
```
def save_plot(column):
    fig,ax = plt.subplots()
    g = data.groupby('Activity')

    num_groups = g.ngroups

    for i, group in g:
        group[column].hist(alpha=0.7, ax =ax , label=i)

    ax.legend()
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.title(column + " Distribution")
    plt.savefig(column + ".png")
```

In [11]:
```
l = ['tGravityAcc-min()-X', 'tBodyAcc-max()-X',
       'fBodyAcc-bandsEnergy()-1,8', 'tBodyGyroJerk-iqr()-X',
       'tGravityAcc-max()-Y', 'tBodyGyro-max()-X', 'tBodyAccJerk-max()-X',
       'tGravityAcc-mean()-X', 'tBodyAcc-correlation()-X,Y',
       'fBodyGyro-bandsEnergy()-9,16']
for i in l:
    save_plot(i)
```



In [12]:
```
save_plot(data.columns[4])
```



Data Prep For model

In [13]:
```
def data_prep(train_set, test_set, columns):

    train_set['Activity'] = pd.factorize(train_set['Activity'], sort = True)[0]
    test_set['Activity'] = pd.factorize(test_set['Activity'], sort = True)[0]
    X_train = train_set[columns]
    X_test = test_set[columns]
    y_train = train_set['Activity']
    y_test = test_set['Activity']

    return (X_train, X_test, y_train, y_test)
```

In [14]:
```
def train_test(X, Y):
    test_size = 0.3
    seed = 123
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_siz
    return (X_train, X_test, y_train, y_test)
```

In [15]:
```
data_org = data.copy()
```

In [16]:
```
X =  data[list(data.columns)[0:561]]
```

In [17]:
```
Y = data[list(data.columns)[561]]

encoder = preprocessing.LabelEncoder()

# encoding train labels
encoder.fit(Y)
```

Out[17]:
```
LabelEncoder()
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test(X, encoder.transform(Y))
```

In [19]:
```
res = {}
for cl in encoder.classes_:
    res.update({cl:encoder.transform([cl])[0]})
res
```

Out[19]:
```
{'LAYING': 0,
 'SITTING': 1,
 'STANDING': 2,
 'WALKING': 3,
 'WALKING_DOWNSTAIRS': 4,
 'WALKING_UPSTAIRS': 5}
```

Model Helper Functions

In [20]:
```
def train_model(X, Y, model):

    model.fit(X, Y)

    return model
```

In [21]:
```
def get_confusion_matrix(y_pred, y_true):

    accuracy = accuracy_score(y_true, y_pred)
    cm = confusion_matrix(y_true, y_pred, labels = [0, 1, 2, 3, 4, 5])

    return (accuracy, cm)
```

XGboost Classifier

In [22]:
```
xgb_model = XGBClassifier()
```

In [23]:
```
model = train_model(X_train, y_train, xgb_model)
y_test_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
```

In [24]:
```
acc_train, cm = get_confusion_matrix(y_train_pred, y_train)
acc_test, cm = get_confusion_matrix(y_test_pred, y_test)
```

In [25]:
```
print("Training Accuracy", acc_train)
print("Testing Accuracy", acc_test)
```

```
Training Accuracy 0.999445138021917
Testing Accuracy 0.9828478964401295
```

In [26]:
```
print(cm)
```

```
[[543   0   0   0   0   0]
 [  0 564  23   0   0   0]
 [  0  14 544   0   0   0]
 [  0   0   0 503   0   3]
 [  0   0   0   1 438   9]
 [  0   0   0   2   1 445]]
```

In [27]:
```
print(classification_report(list(encoder.inverse_transform(y_test_pred)),list(en
```

```
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       543
           SITTING       0.96      0.98      0.97       578
          STANDING       0.97      0.96      0.97       567
           WALKING       0.99      0.99      0.99       506
WALKING_DOWNSTAIRS       0.98      1.00      0.99       439
  WALKING_UPSTAIRS       0.99      0.97      0.98       457

          accuracy                           0.98      3090
         macro avg       0.98      0.98      0.98      3090
      weighted avg       0.98      0.98      0.98      3090
```

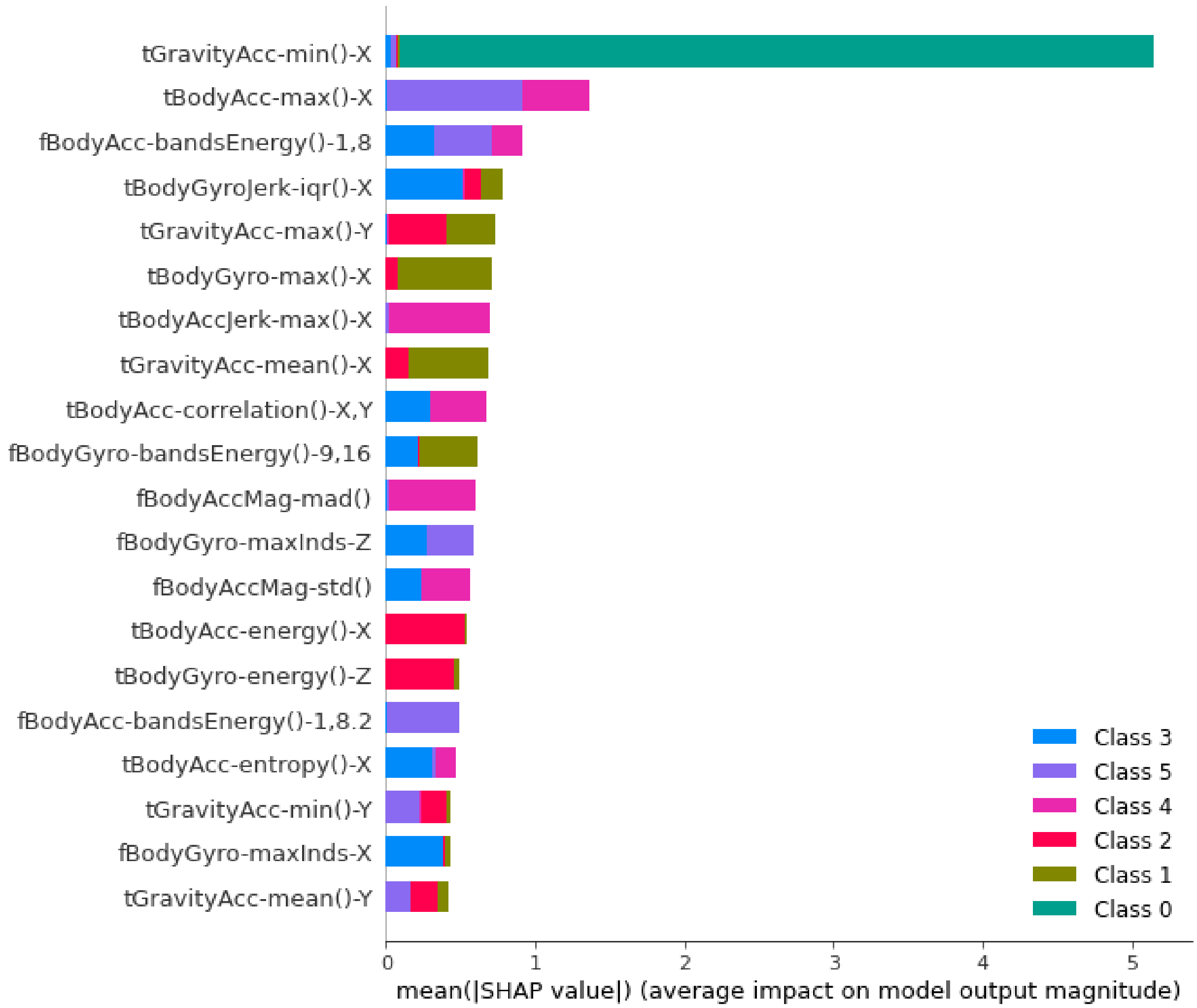
Feature importance

In [28]:
```
shap.initjs()
```

In [29]:
```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
```

```
Setting feature_perturbation = "tree_path_dependent" because no background data
was given.
```

In [30]:
```
# Plot summary_plot
shap.summary_plot(shap_values, X_test)
plt.savefig("Feature Importance.png")
```




```
<Figure size 432x288 with 0 Axes>
```

In [31]:
```
vals= np.abs(shap_values).mean(0)
feature_importance = pd.DataFrame(list(zip(X_test.columns, sum(vals))), columns=
feature_importance.sort_values(by=['feature_importance_vals'], ascending=False,i
feature_importance.to_csv('feature importance XgBoost.csv')
```

In [32]:
```
feature_importance = feature_importance['col_name'].values
```

In [33]:
```
feature_importance[:10]
```

Out[33]:
```
array(['tGravityAcc-min()-X', 'tBodyAcc-max()-X',
       'fBodyAcc-bandsEnergy()-1,8', 'tBodyGyroJerk-iqr()-X',
       'tGravityAcc-max()-Y', 'tBodyGyro-max()-X', 'tBodyAccJerk-max()-X',
       'tGravityAcc-mean()-X', 'tBodyAcc-correlation()-X,Y',
       'fBodyGyro-bandsEnergy()-9,16'], dtype=object)
```

MLP Classifier

In [34]:
```
mlp_model = MLPClassifier(random_state=1, max_iter=300)
```

In [35]:
```
model = train_model(X_train, y_train, mlp_model)
y_test_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
```

In [36]:
```
acc_train, cm = get_confusion_matrix(y_train_pred, y_train)
acc_test, cm = get_confusion_matrix(y_test_pred, y_test)
```

In [37]:
```
print("Training Accuracy", acc_train)
print("Testing Accuracy", acc_test)
```

```
Training Accuracy 0.9933416562630045
Testing Accuracy 0.983495145631068
```

In [38]:
```
print(cm)
```

```
[[543   0   0   0   0   0]
 [  0 551  36   0   0   0]
 [  0  15 543   0   0   0]
 [  0   0   0 506   0   0]
 [  0   0   0   0 448   0]
 [  0   0   0   0   0 448]]
```

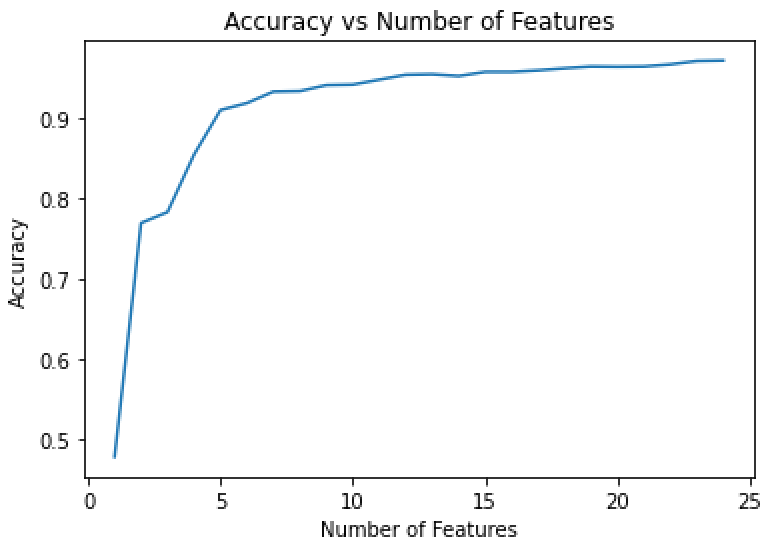
In [39]:
```
print(classification_report(list(encoder.inverse_transform(y_test_pred)),list(en
```

```
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       543
           SITTING       0.94      0.97      0.96       566
          STANDING       0.97      0.94      0.96       579
           WALKING       1.00      1.00      1.00       506
WALKING_DOWNSTAIRS       1.00      1.00      1.00       448
  WALKING_UPSTAIRS       1.00      1.00      1.00       448

          accuracy                           0.98      3090
         macro avg       0.99      0.99      0.99      3090
      weighted avg       0.98      0.98      0.98      3090
```

Identifying Accuracy vs Number of Columns required

In [40]:
```
accuracy = []

for i in range(1,50):
    columns = feature_importance[0:i]
    columns = list(columns)
    xgb_model = XGBClassifier()
    model = train_model(X_train[columns], y_train, xgb_model)
    y_pred = model.predict(X_test[columns])
    acc, cm = get_confusion_matrix(y_pred, y_test)
    accuracy.append(acc)
```

In [41]:
```
plt.plot( list(range(1,25)), accuracy[0:24])
plt.xlabel("Number of Features")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Number of Features")
#plt.show()
plt.savefig("Accuracy vs Numbrt of Features")
```



In [42]:
```
for i,v  in enumerate(accuracy):
    print(i,v)
```

```
0 0.47896440129449835
1 0.7692556634304207
2 0.7828478964401294
3 0.8543689320388349
4 0.9100323624595469
5 0.9187702265372168
6 0.933009708737864
7 0.9336569579288025
8 0.9411003236245955
9 0.941747572815534
10 0.9478964401294498
11 0.9540453074433657
12 0.9546925566343042
13 0.9524271844660194
14 0.9576051779935275
15 0.9576051779935275
16 0.959546925566343
17 0.962135922330097
18 0.9644012944983819
19 0.9640776699029127
20 0.9644012944983819
21 0.9669902912621359
22 0.9711974110032362
23 0.9718446601941747
24 0.9724919093851133
25 0.9744336569579288
26 0.974757281553398
27 0.9744336569579288
28 0.9766990291262136
29 0.976051779935275
30 0.9766990291262136
31 0.9783171521035599
32 0.9770226537216828
33 0.977669902912621З
34 0.9783171521035599
35 0.9786407766990292
36 0.9773462783171522
37 0.9770226537216828
38 0.9773462783171522
39 0.9792880258899677
40 0.9799352750809062
41 0.9796116504854369
42 0.9799352750809062
43 0.9802588996763754
44 0.9818770226537217
45 0.9802588996763754
46 0.9809061488673139
47 0.9796116504854369
48 0.9783171521035599
```

PCA

In [43]:
```
pca = PCA( svd_solver='full')
```

In [44]:
```
pca.fit(X_train)
```

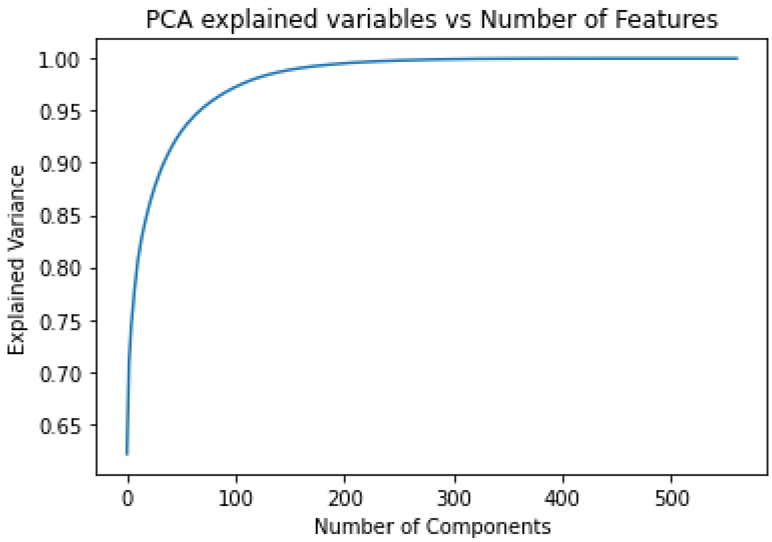
Out[44]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='full', tol=0.0, whiten=False)
```

In [45]:
```
pca_sum = pca.explained_variance_ratio_.cumsum()
```

In [46]:
```
for i,val in enumerate(pca_sum):
    print(i,val)
```

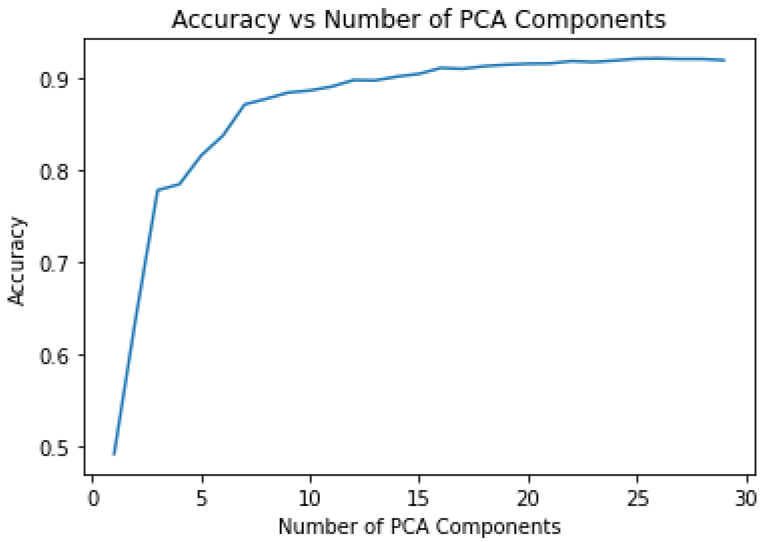
```
69 0.9518033138539366
70 0.9526946136025539
71 0.9535716205535273
72 0.9544081457748887
73 0.9552360403290069
74 0.9560545172060196
75 0.9568663939722262
76 0.9576504098500372
77 0.9584195689389133
78 0.9591849146513712
79 0.9599253780408631
80 0.9606512129230305
81 0.9613672119202841
82 0.9620600125353189
83 0.9627485283168615
84 0.9634250107566441
85 0.9640967379498053
86 0.9647457067666348
87 0.9653833265162237
88 0.9660159329763657
```

In [47]:
```
plt.plot(pca_sum)
plt.xlabel("Number of Components")
plt.ylabel("Explained Variance")
plt.title("PCA explained variables vs Number of Features")
#plt.show()
plt.savefig("Explained Variance.png")
```



In [48]:
```
accuracy = []

for i in range(1,30):
    xgb_model = XGBClassifier()
    model = train_model(pca.transform(X_train)[:,0:i], y_train, xgb_model)
    y_pred = model.predict(pca.transform(X_test)[:,0:i])
    acc, cm = get_confusion_matrix(y_pred, y_test)
    accuracy.append(acc)
```

In [52]:
```
plt.plot( list(range(1,30)), accuracy[0:29])
plt.xlabel("Number of PCA Components")
plt.ylabel("Accuracy")
plt.title("Accuracy vs Number of PCA Components")
#plt.show()
plt.savefig("Accuracy vs Number of PCA Components")
```



In [50]:
```
for i,val in enumerate(accuracy):
    print(i,val)
```

```
0 0.49093851132686084
1 0.6401294498381876
2 0.7779935275080906
3 0.7844660194174757
4 0.8158576051779936
5 0.8375404530744337
6 0.8715210355987055
7 0.8776699029126214
8 0.8844660194174757
9 0.8867313915857605
10 0.8909385113268609
11 0.8980582524271845
12 0.8977346278317152
13 0.9019417475728155
14 0.9048543689320389
15 0.911326860841424
16 0.9103559870550162
17 0.9132686084142395
18 0.9148867313915857
19 0.9158576051779935
20 0.9161812297734628
21 0.9187702265372168
22 0.917799352750809
23 0.9194174757281554
24 0.9213592233009709
25 0.9216828478964402
26 0.9210355987055017
27 0.9210355987055017
28 0.9197411003236245
```