In [71]:
```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
```

In [72]:
```python
df=pd.read_csv("train_har.csv")
```

In [3]:
```python
df.describe()
```

Out[3]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-skewness()	fBodyBodyGyroJerkMag-kurtosis()	angle(tBodyAccMean,gravity)
count	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	...	7352.000000	7352.000000	7352.000000
mean	0.274488	-0.017695	-0.109141	-0.605438	-0.510938	-0.604754	-0.630512	-0.526907	-0.606150	-0.468604	...	-0.307009	-0.625294	0.008684
std	0.070261	0.040811	0.056635	0.448734	0.502645	0.418687	0.424073	0.485942	0.414122	0.544547	...	0.321011	0.307584	0.336787
min	-1.000000	-1.000000	-1.000000	-1.000000	-0.999873	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	...	-0.995357	-0.999765	-0.976580
25%	0.262975	-0.024863	-0.120993	-0.992754	-0.978129	-0.980233	-0.993591	-0.978162	-0.980251	-0.936219	...	-0.542602	-0.845573	-0.121527
50%	0.277193	-0.017219	-0.108676	-0.946196	-0.851897	-0.859365	-0.950709	-0.857328	-0.857143	-0.881637	...	-0.343685	-0.711692	0.009509
75%	0.288461	-0.010783	-0.097794	-0.242813	-0.034231	-0.262415	-0.292680	-0.066701	-0.265671	-0.017129	...	-0.126979	-0.503878	0.150865
max	1.000000	1.000000	1.000000	1.000000	0.916238	1.000000	1.000000	0.967664	1.000000	1.000000	...	0.989538	0.956845	1.000000

8 rows × 562 columns

In [20]:
```python
df['subject']
```

Out[20]:
```
0       1
1       1
2       1
3       1
4       1
       ..
7347   30
7348   30
7349   30
7350   30
7351   30
Name: subject, Length: 7352, dtype: int64
```

In [4]:
```python
missing_value = ["N/a","na",np.nan]
df=pd.read_csv("train_har.csv",na_values=missing_value)
```

In [7]:
```python
df.isnull()
```

Out[7]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-kurtosis()	angle(tBodyAccMean,gravity)	angle(tBodyAccJerkMean),gravityMean)	an
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7347	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
7348	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
7349	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
7350	False	False	False	False	False	False	False	False	False	False	...	False	False	False	
7351	False	False	False	False	False	False	False	False	False	False	...	False	False	False	

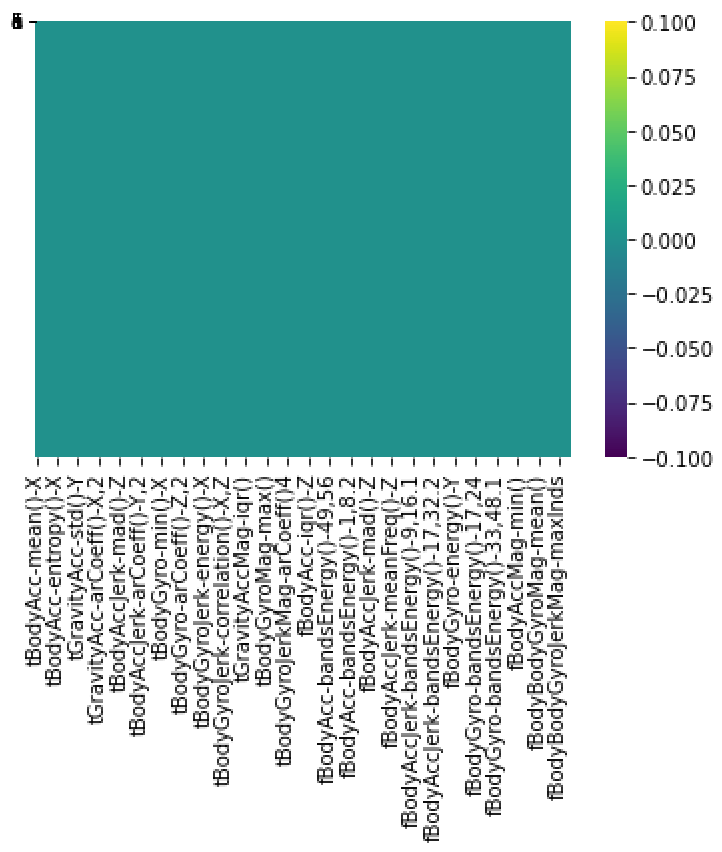
7352 rows × 563 columns

In [8]:
```python
df.isnull().sum()
```

Out[8]:
```
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: 563, dtype: int64
```

In [5]:
```python
sns.heatmap(df.isnull(),yticklabels="False",cmap='viridis')
```

Out[5]:
```
<AxesSubplot:>
```



No null values present in the data

In [ ]:

In [11]:
```python
df.head()
```

Out[11]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-kurtosis()	angle(tBodyAccMean,gravity)	angle(tBodyAccJerkMean),gravityMean)	angle(
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.710304	-0.112754	0.030400	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	-0.861499	0.053477	-0.007435	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.760104	-0.118559	0.177899	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	-0.482845	-0.036788	-0.012892	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	-0.699205	0.123320	0.122542	

5 rows × 563 columns

In [12]:
```python
df.tail()
```

Out[12]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-kurtosis()	angle(tBodyAccMean,gravity)	angle(tBodyAccJerkMean),gravityMean)	an
7347	0.299665	-0.057193	-0.181233	-0.195387	0.039905	0.077078	-0.282301	0.043616	0.060410	0.210795	...	-0.880324	-0.190437	0.829718	
7348	0.273853	-0.007749	-0.147468	-0.235309	0.004816	0.059280	-0.322552	-0.029456	0.080585	0.117440	...	-0.680744	0.064907	0.875679	
7349	0.273387	-0.017011	-0.045022	-0.218218	-0.103822	0.274533	-0.304515	-0.098913	0.332584	0.043999	...	-0.304029	0.052806	-0.266724	
7350	0.289654	-0.018843	-0.158281	-0.219139	-0.111412	0.268893	-0.310487	-0.068200	0.319473	0.101702	...	-0.344314	-0.101360	0.700740	
7351	0.351503	-0.012423	-0.203867	-0.269270	-0.087212	0.177404	-0.377404	-0.038678	0.229430	0.269013	...	-0.740738	-0.280088	-0.007739	

5 rows × 563 columns

In [14]:
```python
df.shape
```

Out[14]: (7352, 563)

In [6]:
```python
#checking for duplicates

print('Number of duplicate entries in the dataset {}'.format(sum(df.duplicated())))
```

```
Number of duplicate entries in the dataset 0
```

In [ ]:

In [ ]:
```python
# Class distribution
```

In [16]:
```python
df['Activity'].unique()
```

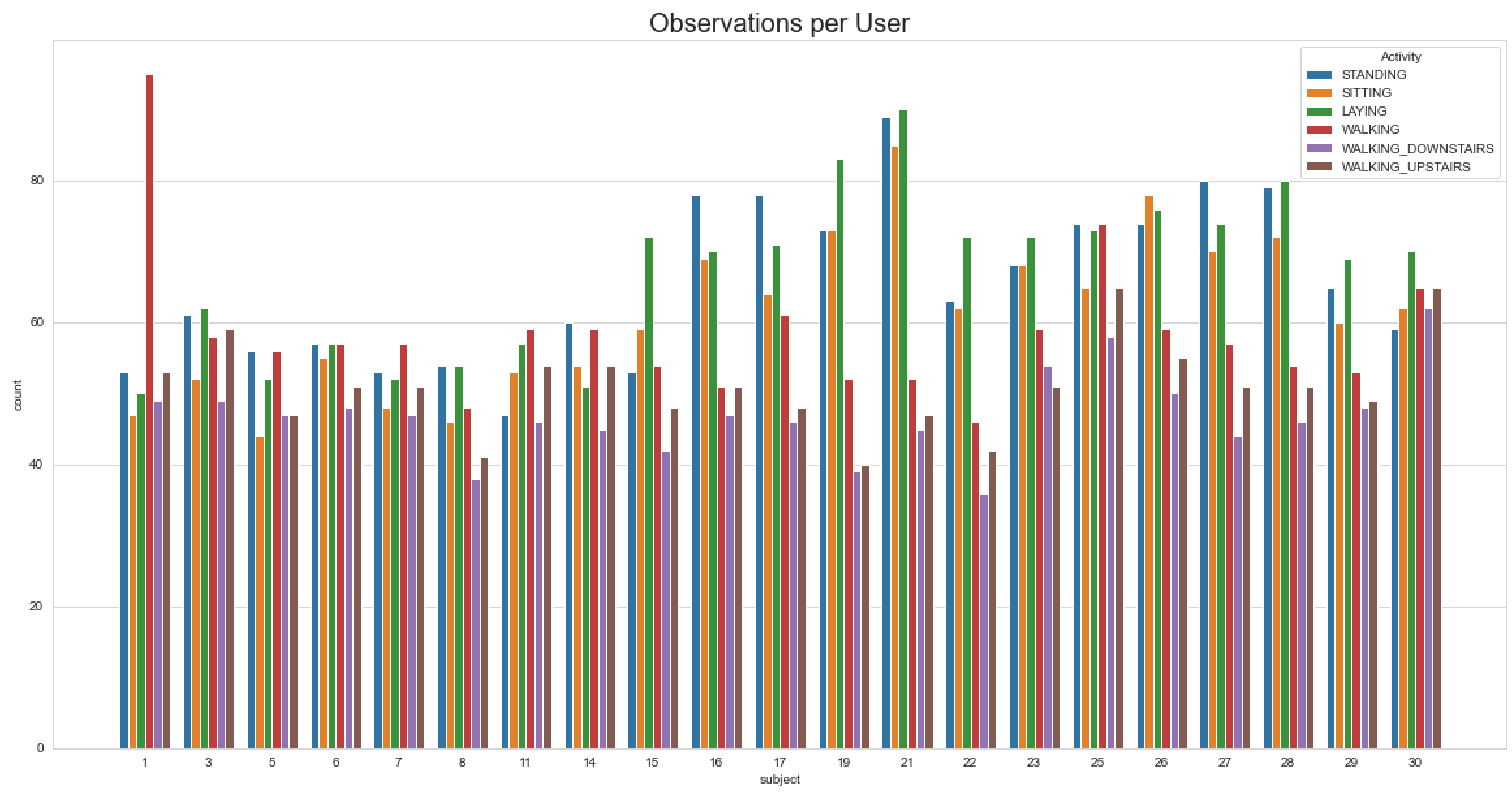
Out[16]:
```
array(['STANDING', 'SITTING', 'LAYING', 'WALKING', 'WALKING_DOWNSTAIRS',
       'WALKING_UPSTAIRS'], dtype=object)
```

In [ ]:

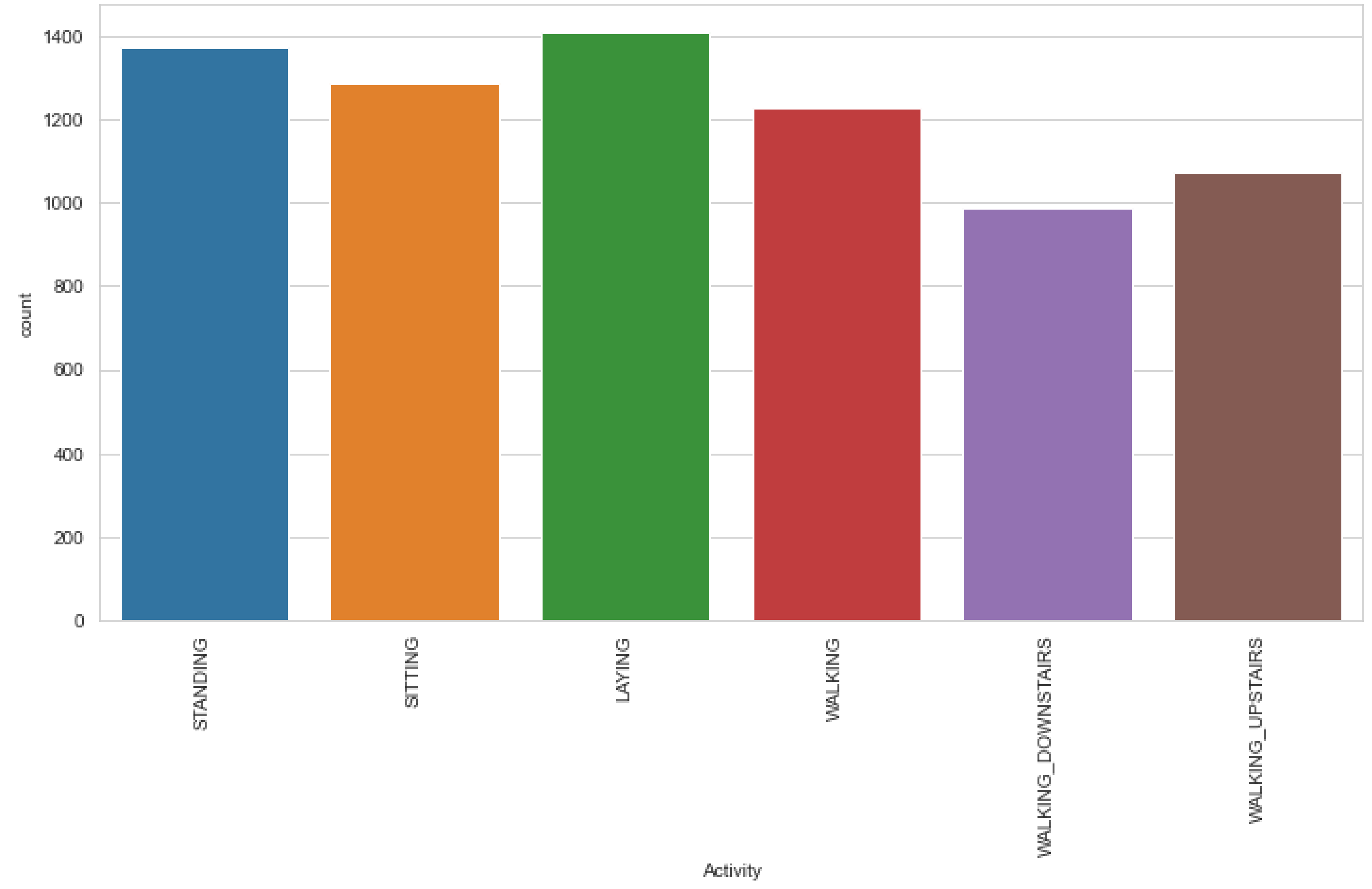
Now Visualize the class Distribution¶

In [4]:
```python
# Plotting data with respect to subject
sns.set_style('whitegrid')
plt.figure(figsize=(20,10))
plt.title('Observations per User', fontsize=20)
sns.countplot(x='subject', hue='Activity', data=df)
plt.plot()
```

Out[4]: []



In [15]:
```python
plt.figure(figsize=(12,6))
axis=sns.countplot(x="Activity",data=df)
plt.xticks(x=df['Activity'],rotation='vertical')
plt.show()
```

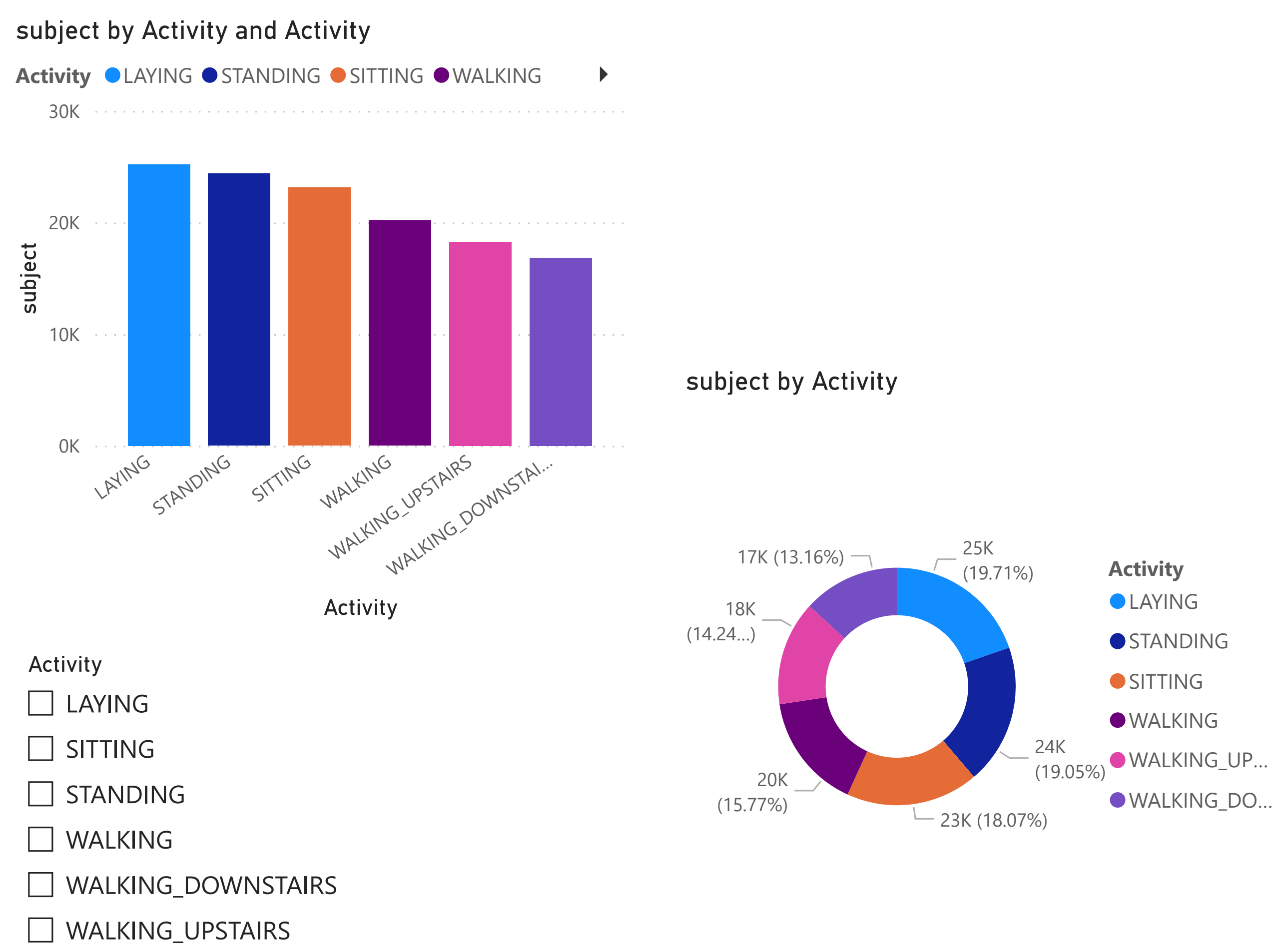


In [ ]:

Connecting Data visualization using power bi

In [3]:
```python
from IPython.display import IFrame
powerBiEmbed = 'https://app.powerbi.com/reportEmbed?reportId=13c711a2-eb57-4fcb-9f44-63c2786d0449&autoAuth=true&ctid=6071e59a-0b4e-4894-b7ea-b4e51a072a6a&config=eyJjbHVzdGVyVXJsIjoi
IFrame(powerBiEmbed, width=1200,height=600)
```

Out[3]:



In [ ]:

In [8]:
```python
df['subject'].unique()
```

Out[8]:
```
array([ 1,  3,  5,  6,  7,  8, 11, 14, 15, 16, 17, 19, 21, 22, 23, 25, 26,
       27, 28, 29, 30], dtype=int64)
```

In [73]:
```python
X=pd.DataFrame(df.drop(['Activity','subject'],axis=1))
y= df.Activity.values.astype(object)
```

In [5]:
```python
X.shape , y.shape
```

Out[5]:
```
((7352, 561), (7352,))
```

In [6]:
```python
print(X)
```

```
      tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  \
0              0.288585          -0.020294          -0.132905
1              0.278419          -0.016411          -0.123520
2              0.279653          -0.019467          -0.113462
3              0.279174          -0.026201          -0.123283
4              0.276629          -0.016570          -0.115362
...                 ...                ...                ...
7347           0.299665          -0.057193          -0.181233
7348           0.273853          -0.007749          -0.147468
7349           0.273387          -0.017011          -0.045022
7350           0.289654          -0.018843          -0.158281
7351           0.351503          -0.012423          -0.203867

      tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  \
0            -0.995279         -0.983111         -0.913526         -0.995112
1            -0.998245         -0.975300         -0.960322         -0.998807
2            -0.995380         -0.967187         -0.978944         -0.996520
3            -0.996091         -0.983403         -0.990675         -0.997099
4            -0.998139         -0.980817         -0.990482         -0.998321
...                ...               ...               ...               ...
7347         -0.195387          0.039905          0.077078         -0.282301
7348         -0.235309          0.004816          0.059280         -0.322552
7349         -0.218218         -0.103822          0.274533         -0.304515
7350         -0.219139         -0.111412          0.268893         -0.310487
7351         -0.269270         -0.087212          0.177404         -0.377404

      tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  ...  \
0            -0.983185         -0.923527         -0.934724  ...
1            -0.974914         -0.957686         -0.943068  ...
2            -0.963668         -0.977469         -0.938692  ...
3            -0.982750         -0.989302         -0.938692  ...
4            -0.979672         -0.990441         -0.942469  ...
...                ...               ...               ...  ...
7347          0.043616          0.060410          0.210795  ...
7348         -0.029456          0.080585          0.117440  ...
7349         -0.098913          0.332584          0.043999  ...
7350         -0.068200          0.319473          0.101702  ...
7351         -0.038678          0.229430          0.269013  ...

      fBodyBodyGyroJerkMag-meanFreq()  fBodyBodyGyroJerkMag-skewness()  \
0                           -0.074323                        -0.298676
1                            0.158075                        -0.595051
2                            0.414503                        -0.390748
3                            0.404573                        -0.117290
4                            0.087753                        -0.351471
...                               ...                              ...
7347                        -0.070157                        -0.588433
7348                         0.165259                        -0.390738
7349                         0.195034                         0.025145
7350                         0.013865                         0.063907
7351                        -0.058402                        -0.387052

      fBodyBodyGyroJerkMag-kurtosis()  angle(tBodyAccMean,gravity)  \
0                           -0.710304                    -0.112754
1                           -0.861499                     0.053477
2                           -0.760104                    -0.118559
3                           -0.482845                    -0.036788
4                           -0.699205                     0.123320
...                               ...                          ...
7347                        -0.880324                    -0.190437
7348                        -0.680744                     0.064907
7349                        -0.304029                     0.052806
7350                        -0.344314                    -0.101360
7351                        -0.740738                    -0.280088

      angle(tBodyAccJerkMean),gravityMean)  angle(tBodyGyroMean,gravityMean)  \
0                                 0.030400                         -0.464761
1                                -0.007435                         -0.732626
2                                 0.177899                          0.100699
3                                -0.012892                          0.640011
4                                 0.122542                          0.693578
...                                    ...                               ...
7347                              0.829718                          0.206972
7348                              0.875679                         -0.879033
7349                             -0.266724                          0.864404
7350                              0.700740                          0.936674
7351                             -0.007739                         -0.056088

      angle(tBodyGyroJerkMean,gravityMean)  angle(X,gravityMean)  \
0                                -0.018446             -0.841247
1                                 0.703511             -0.844788
2                                 0.808529             -0.848933
3                                -0.485366             -0.848649
4                                -0.615971             -0.847865
...                                    ...                   ...
7347                             -0.425619             -0.791883
7348                              0.400219             -0.771840
7349                              0.701169             -0.779133
7350                             -0.589479             -0.785181
7351                             -0.616956             -0.783267

      angle(Y,gravityMean)  angle(Z,gravityMean)
0                 0.179941             -0.058627
1                 0.180289             -0.054317
2                 0.180637             -0.049118
3                 0.181935             -0.047663
4                 0.185151             -0.043892
...                    ...                   ...
7347              0.238604              0.049819
7348              0.252676              0.050053
7349              0.249145              0.040811
7350              0.246432              0.025339
7351              0.246809              0.036695

[7352 rows x 561 columns]
```

In [5]:
```python
print(y)
```

```
['STANDING' 'STANDING' 'STANDING' ... 'WALKING_UPSTAIRS'
 'WALKING_UPSTAIRS' 'WALKING_UPSTAIRS']
```

In [ ]:

Transforming Non numerical Labels into numerical labels¶

In [74]:
```python
from sklearn import preprocessing
```

In [75]:
```python
encoder=preprocessing.LabelEncoder()
```

In [76]:
```python
encoder.fit(y)
y=encoder.transform(y)
y.shape
```
Out[76]:
```
(7352,)
```

In [10]:
```python
encoder.classes_
```
Out[10]:
```
array(['LAYING', 'SITTING', 'STANDING', 'WALKING', 'WALKING_DOWNSTAIRS',
       'WALKING_UPSTAIRS'], dtype=object)
```

In [ ]:

Standard scalar

In [77]:
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [78]:
```python
X=scaler.fit_transform(X)
```

In [79]:
```python
print(X)
```
```
[[ 0.20064157 -0.0636826  -0.41962845 ... -0.68721921  0.40794614
  -0.00756789]
 [ 0.05594788  0.03148567 -0.25390836 ... -0.694138    0.40911698
   0.00787517]
 [ 0.07351535 -0.04341648 -0.07629468 ... -0.702239    0.4102883
   0.02650234]
 ...
 [-0.01566765  0.0167814   1.13222107 ... -0.56584847  0.64059683
   0.34870928]
 [ 0.21586648 -0.02812252 -0.86770988 ... -0.57766781  0.63147758
   0.29327564]
 [ 1.09620157  0.12919873 -1.67268082 ... -0.57392691  0.63274259
   0.33396081]]
```

In [ ]:

Splitting the data into training and test data

In [80]:
```python
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.3,random_state = 100)
```

In [81]:
```python
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)
```
```
X_train (5146, 561)
X_test (2206, 561)
y_train (5146,)
y_test (2206,)
```

In [ ]:

Linear Regression

In [82]:
```python
from sklearn.linear_model import LinearRegression
```

In [83]:
```python
model = LinearRegression()
```

In [22]:
```python
model.fit(X_train, y_train)
LinearRegression()
```
Out[22]:
```
LinearRegression()
```

In [84]:
```python
model = LinearRegression().fit(X_train, y_train)
```

In [195]:
```python
r_sq = model.score(X_test, y_test)
print(f"coefficient of determination: {r_sq}")
```
```
coefficient of determination: 0.9832275611967362
```

In [86]:
```python
print(f"intercept: {model.intercept_}")
```
```
intercept: 2.316036200621678
```

In [26]:
```python
print(f"slope: {model.coef_}")
```
```
 -1.18911597e-01 -2.73928515e-02  5.87718399e-02  1.22032424e+00
  1.35338848e+05  7.24847722e+04  6.46971261e+04 -5.31718117e-02
 -2.01502312e-01  5.99824346e-02 -4.49157854e-02 -1.27917696e-01
 -1.60950285e-01  1.03880213e-02  2.03053286e-02 -1.32510850e-02
 -3.10795606e-02  2.96573821e-02  4.55298015e-02  1.70641247e-02
  7.10578386e-03 -8.50088987e-03  1.75892608e-02  4.21347142e-02
  2.34083001e-02 -2.45780467e-02 -6.27598906e-02  1.56740970e-03
  3.93235620e-02  1.45024054e-02 -8.23726047e-03  4.81237923e+00
  3.44415521e-01  5.64028059e-01  6.58542206e-01 -2.86467539e-01
  4.67453563e-01  7.17372765e-02  2.18021957e-02  7.69152468e-02
 -1.40440322e-01 -3.93887578e-02 -4.10580724e-02 -6.17920701e-01
 -2.60889871e+05 -2.58821418e+05 -1.07497369e+05 -1.28125236e-01

  8.02615757e-02  8.32534702e-02 -6.08944082e-03 -1.85565853e-03
  1.60532757e-02 -7.76050685e-03  5.11619031e-03 -5.93707156e-03
 -2.79045095e-02  3.85463791e-02  2.84751735e-02  2.77277361e-02
 -2.54211181e-03  2.25070537e-01  1.01721988e-01 -4.51394784e-02
 -1.18390852e-01  3.66696798e-02 -4.61704117e-03 -2.52464832e-02
 -1.29514526e-02 -5.17084968e-03 -8.23915500e-03 -1.00635010e+00
 -4.71567039e-01 -4.51652863e-01  5.52382696e-01 -1.23894777e-01
```

In [87]:
```python
# Making predictions using the predict() and xTest data
predictions = model.predict(X_test)
```

In [88]:
```python
comparison = pd.DataFrame({'Predicted Values':predictions,'Actual Values':y_test})
```

In [89]:
```
print(comparison.head(10))
```

```
   Predicted Values  Actual Values
0         -0.266657              0
1          3.613355              4
2          4.276231              4
3          4.061033              4
4          3.167156              3
5          1.762185              2
6         -0.027905              0
7          3.024659              3
8          4.700019              5
9          4.735600              5
```

In [ ]:

Logistic regression

In [90]:
```
from sklearn.linear_model import LogisticRegression
```

In [91]:
```
logmodel=LogisticRegression()
```

In [92]:
```
logmodel.fit(X_train,y_train)
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: 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 (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[92]:
```
LogisticRegression()
```

In [93]:
```
predictions=logmodel.predict(X_test)
```

In [94]:
```
from sklearn.metrics import classification_report
```

In [95]:
```
classification_report(y_test,predictions)
```

Out[95]:
```
'              precision    recall  f1-score   support\n\n           0       1.00      1.00      1.00       419\n           1       0.96      0.96      0.96       386\n           2
0.96      0.97      0.96       410\n           3       1.00      1.00      1.00       356\n           4       1.00      1.00      1.00       316\n           5       0.99      1.00
1.00       319\n\n    accuracy                           0.99      2206\n   macro avg       0.99      0.99      0.99      2206\nweighted avg       0.99      0.99      0.99      220
6\n'
```

In [96]:
```
from sklearn.metrics import confusion_matrix
```

In [97]:
```
confusion_matrix(y_test,predictions)
```

Out[97]:
```
array([[419,   0,   0,   0,   0,   0],
       [  0, 370,  15,   0,   0,   1],
       [  0,  14, 396,   0,   0,   0],
       [  0,   0,   0, 356,   0,   0],
       [  0,   0,   0,   0, 315,   1],
       [  0,   0,   0,   0,   1, 318]], dtype=int64)
```

In [98]:
```
from sklearn.metrics import accuracy_score
```

In [196]:
```
Lr=accuracy_score(y_test,predictions)
```

In [197]:
```
Lr
```

Out[197]:
```
0.985494106980961
```

In [ ]:

SVM

In [168]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report,accuracy_score
```

In [169]:
```
from sklearn import svm
model = svm.SVC(C = 1,kernel = 'linear',gamma = 'auto')
fit_model = model.fit(X_train,y_train)
```

In [170]:
```
sv=model.score(X_test, y_test)
print('Test set\n Accuracy: {:0.2f}'.format(model.score(X_test, y_test))) #the accuracy of the model on test data is given below
```

```
Test set
 Accuracy: 0.98
```

In [ ]:

Decision tree

In [171]:
```
from sklearn.tree import DecisionTreeClassifier
dtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

Out[171]:
```
DecisionTreeClassifier()
```

In [172]:
```
# Predicting the values of test data
y_pred = dtree.predict(X_test)
print("Classification report - \n", classification_report(y_test,y_pred))
```

```
Classification report - 
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       419
           1       0.91      0.87      0.89       386
           2       0.89      0.92      0.90       410
           3       0.95      0.94      0.95       356
           4       0.96      0.94      0.95       316
           5       0.91      0.95      0.93       319

    accuracy                           0.94      2206
   macro avg       0.94      0.94      0.94      2206
weighted avg       0.94      0.94      0.94      2206
```

In [184]:
```
dt=dtree.score(X_test, y_test)
print('Test set\n Accuracy: {:0.2f}'.format(dtree.score(X_test, y_test))) #the accuracy of the model on test data is given below
```

```
Test set
 Accuracy: 0.94
```

In [ ]:

In [ ]:

Random forest

In [174]:
```
from sklearn.ensemble import RandomForestClassifier
Rn = RandomForestClassifier(n_estimators=4,criterion='entropy',random_state=0)
Rn=Rn.fit(X_train,y_train)
```

In [175]:
```
rn=Rn.score(X_test, y_test)
print('Test set\n Accuracy: {:0.2f}'.format(Rn.score(X_test, y_test))) #the accuracy of the model on test data is given below
```

```
Test set
 Accuracy: 0.94
```

In [ ]:

Bagging Classifier Model

In [43]:
```
from sklearn.ensemble import BaggingClassifier
```

In [44]:
```
BC=BaggingClassifier()
BC= BC.fit(X_train , y_train)
BC
```

Out[44]:
```
BaggingClassifier()
```

In [147]:
```
bc=BC.score(X_test, y_test)
print('Test set\n Accuracy: {:0.2f}'.format(BC.score(X_test, y_test))) #the accuracy of the model on test data is given below
```

```
Test set
 Accuracy: 0.96
```

In [ ]:

XGB Classifierr Model

In [140]:
```

from xgboost import XGBClassifier
```

In [141]:
```
XG=XGBClassifier(verbosity = 0)
XG= XG.fit(X_train , y_train)
XG
```

Out[141]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=0, reg_alpha=0, ...)
```

In [142]:
```
xg=XG.score(X_test, y_test)
print('Test set\n Accuracy: {:0.2f}'.format(XG.score(X_test, y_test))) #the accuracy of the model on test data is given below
```

```
Test set
 Accuracy: 0.99
```

In [ ]:

AdaBoost Classifier Model

In [122]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [123]:
```
AD=AdaBoostClassifier()
AD= AD.fit(X_train , y_train)
AD
```

Out[123]:
```
AdaBoostClassifier()
```

In [124]:
```
ad=AD.score(X_test, y_test)
print('Test set\n Accuracy: {:0.2f}'.format(AD.score(X_test, y_test))) #the accuracy of the model on test data is given below
```

```
Test set
 Accuracy: 0.54
```

In [ ]:

Gradient Boosting Classifier Model

In [15]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [16]:
```
GB=GradientBoostingClassifier()
GB= GB.fit(X_train , y_train)
GB
```

Out[16]:
```
GradientBoostingClassifier()
```

In [17]:
```
gb=GB.score(X_test, y_test)
print('Test set\n Accuracy: {:0.2f}'.format(GB.score(X_test, y_test))) #the accuracy of the model on test data is given below
```

```
Test set
 Accuracy: 0.98
```

In [ ]:

Comparison of Bagging and Boosting Models

In [207]:
```
from prettytable import PrettyTable
```

In [208]:
```
x = PrettyTable()
print('\n')
```

In [209]:
```
x.field_names = ["Model", "Accuracy"]
x.add_row(["Linear Regression Model", round(r_sq,2)])
x.add_row(["Logistic Regression model", round(Lr,2)])
x.add_row(["Support Vector Machine", round(sv,2)])
x.add_row(["Decision Tree Model", round(dt,2)])
x.add_row(["Random Forest Classifier Model",round(rn,2)])
x.add_row(["Bagging Classifier Model", round(bc,2)])
x.add_row(["XGB Classifierr Model", round(xg,2)])
x.add_row(["AdaBoost Classifier Model", round(ad,2)])
x.add_row(["Gradient Boosting Classifier Model", round(gb,2)])
```

In [210]:
```
print(x)
print('\n')
```

```
+------------------------------------+----------+
|               Model                | Accuracy |
+------------------------------------+----------+
|      Linear Regression Model       |   0.98   |
|     Logistic Regression model      |   0.99   |
|       Support Vector Machine       |   0.98   |
|        Decision Tree Model         |   0.94   |
|   Random Forest Classifier Model   |   0.94   |
|      Bagging Classifier Model      |   0.96   |
|       XGB Classifierr Model        |   0.99   |
|     AdaBoost Classifier Model      |   0.54   |
| Gradient Boosting Classifier Model |   0.98   |
+------------------------------------+----------+
```

XG Boosting and Logistic regression have given highest accuracy of 99% and while Ada Boosting had given the low accuracy of 54%

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: