# HUMAN ACTIVITY RECOGNITION CLASSIFICATION USING MACHINE LEARNING

The following is the data source link of the selected dataset https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones (https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones)

Importing the required packages

In [1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.decomposition import  PCA
from sklearn import preprocessing
from sklearn import tree
from sklearn.manifold import TSNE
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import svm
from sklearn import metrics
from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import f_classif
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report, precision_score
from sklearn.model_selection import cross_val_score, GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv("C:/Users/dkond/Downloads/archive/test.csv")
df.head()
```

Out[2]:

| | 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(t |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.257178 | -0.023285 | -0.014654 | -0.938404 | -0.920091 | -0.667683 | -0.952501 | -0.925249 | -0.674302 | -0.894088 | ... | -0.705974 | |
| **1** | 0.286027 | -0.013163 | -0.119083 | -0.975415 | -0.967458 | -0.944958 | -0.986799 | -0.968401 | -0.945823 | -0.894088 | ... | -0.594944 | |
| **2** | 0.275485 | -0.026050 | -0.118152 | -0.993819 | -0.969926 | -0.962748 | -0.994403 | -0.970735 | -0.963483 | -0.939260 | ... | -0.640736 | |
| **3** | 0.270298 | -0.032614 | -0.117520 | -0.994743 | -0.973268 | -0.967091 | -0.995274 | -0.974471 | -0.968897 | -0.938610 | ... | -0.736124 | |
| **4** | 0.274833 | -0.027848 | -0.129527 | -0.993852 | -0.967445 | -0.978295 | -0.994111 | -0.965953 | -0.977346 | -0.938610 | ... | -0.846595 | |

5 rows × 563 columns

## Attribute Information

```
For each record in the dataset it is provided:

  Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
  Triaxial Angular velocity from the gyroscope.
  A 561-feature vector with time and frequency domain variables.
  Its activity label.
  An identifier of the subject who carried out the experiment.


  Main features:


 '-XYZ' is used to denote 3-axial signals in the X, Y and Z directions.



  1.tBodyAcc-XYZ
  2.tGravityAcc-XYZ
```

```
3.tBodyAccJerk-XYZ
4.tBodyGyro-XYZ
5.tBodyGyroJerk-XYZ
6.tBodyAccMag
7.tGravityAccMag
8.tBodyAccJerkMag
9.tBodyGyroMag
10.tBodyGyroJerkMag
11.fBodyAcc-XYZ
12.fBodyAccJerk-XYZ
13.fBodyGyro-XYZ
14.fBodyAccMag
15.fBodyAccJerkMag
16.fBodyGyroMag
17.fBodyGyroJerkMag
```

```
//The set of variables that were estimated from these signals are://

mean(): Mean value
std(): Standard deviation
mad(): Median absolute deviation
max(): Largest value in array
min(): Smallest value in array
sma(): Signal magnitude area
energy(): Energy measure. Sum of the squares divided by the number of values.
iqr(): Interquartile range
entropy(): Signal entropy
arCoeff(): Autorregresion coefficients with Burg order equal to 4
correlation(): correlation coefficient between two signals
maxInds(): index of the frequency component with largest magnitude
meanFreq(): Weighted average of the frequency components to obtain a mean frequency
skewness(): skewness of the frequency domain signal
kurtosis(): kurtosis of the frequency domain signal
bandsEnergy(): Energy of a frequency interval within the 64 bins of the FFT of each window.
angle(): Angle between to vectors.
```

In [3]:
```
#using the shape method for defining the number of rows and columns
df.shape
```

Out[3]: (2947, 563)

## Checking for the duplicate values in the dataset

In [4]:
```
print("number of duplicates = " + str(df.duplicated().sum()))
```

```
number of duplicates = 0
```

## Checking if there are any null values present in the dataset

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
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
```

The dataset contains zero null values and no duplicate values hence there we do not need to further clean the dataset.

In [6]:
```
df.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
```

In [7]:
```
list(df.columns.values)
```

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',
 'tBodyAcc-max()-Y',
 'tBodyAcc-max()-Z',
 'tBodyAcc-min()-X',
 'tBodyAcc-min()-Y',
 'tBodyAcc-min()-Z',
 'tBodyAcc-sma()',
 'tBodyAcc-energy()-X',
 'tBodyAcc-energy()-Y',
 'tBodyAcc-energy()-Z',
```

In [8]:
```
#len() defines the number of rows present in the dataset.
len(df)
```

Out[8]:
```
2947
```

In [9]:
```
#nunique() function is used to describe the number of unique elements present in the dataset
df.nunique()
```

Out[9]:
```
tBodyAcc-mean()-X       2947
tBodyAcc-mean()-Y       2947
tBodyAcc-mean()-Z       2947
tBodyAcc-std()-X        2947
tBodyAcc-std()-Y        2947
                        ...
angle(X,gravityMean)    2947
angle(Y,gravityMean)    2947
angle(Z,gravityMean)    2947
subject                    9
Activity                   6
Length: 563, dtype: int64
```

Replacing the unwanted characters in the column names to avoid the confusion.

In [10]:
```
columns = df.columns
columns = columns.str.replace('[()]','')
columns = columns.str.replace('[-]', '')
columns = columns.str.replace('[,]','')
df.columns = columns
```

In [11]:
```
list(df.columns.values)
```

Out[11]:
```
['tBodyAccmeanX',
 'tBodyAccmeanY',
 'tBodyAccmeanZ',
 'tBodyAccstdX',
 'tBodyAccstdY',
 'tBodyAccstdZ',
 'tBodyAccmadX',
 'tBodyAccmadY',
 'tBodyAccmadZ',
 'tBodyAccmaxX',
 'tBodyAccmaxY',
 'tBodyAccmaxZ',
 'tBodyAccminX',
 'tBodyAccminY',
 'tBodyAccminZ',
 'tBodyAccsma',
 'tBodyAccenergyX',
 'tBodyAccenergyY',
 'tBodyAccenergyZ',
```

In [12]:
```
df_des=df.describe()
df_des
```

Out[12]:

| | tBodyAccmeanX | tBodyAccmeanY | tBodyAccmeanZ | tBodyAccstdX | tBodyAccstdY | tBodyAccstdZ | tBodyAccmadX | tBodyAccmadY | tBodyAccmadZ | tI |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2947.000000 | 2947.000000 | 2947.000000 | 2947.000000 | 2947.000000 | 2947.000000 | 2947.000000 | 2947.000000 | 2947.000000 | |
| **mean** | 0.273996 | -0.017863 | -0.108386 | -0.613635 | -0.508330 | -0.633797 | -0.641278 | -0.522676 | -0.637038 | |
| **std** | 0.060570 | 0.025745 | 0.042747 | 0.412597 | 0.494269 | 0.362699 | 0.385199 | 0.479899 | 0.357753 | |
| **min** | -0.592004 | -0.362884 | -0.576184 | -0.999606 | -1.000000 | -0.998955 | -0.999417 | -0.999914 | -0.998899 | |
| **25%** | 0.262075 | -0.024961 | -0.121162 | -0.990914 | -0.973664 | -0.976122 | -0.992333 | -0.974131 | -0.975352 | |
| **50%** | 0.277113 | -0.016967 | -0.108458 | -0.931214 | -0.790972 | -0.827534 | -0.937664 | -0.799907 | -0.817005 | |
| **75%** | 0.288097 | -0.010143 | -0.097123 | -0.267395 | -0.105919 | -0.311432 | -0.321719 | -0.133488 | -0.322771 | |
| **max** | 0.671887 | 0.246106 | 0.494114 | 0.465299 | 1.000000 | 0.489703 | 0.439657 | 1.000000 | 0.427958 | |

8 rows × 562 columns

In [13]:
```
df_corr= df.corr()
df_corr
```

Out[13]:

| | tBodyAccmeanX | tBodyAccmeanY | tBodyAccmeanZ | tBodyAccstdX | tBodyAccstdY | tBodyAccstdZ | tBodyAccmadX | tBo |
|---|---|---|---|---|---|---|---|---|
| **tBodyAccmeanX** | 1.000000 | 0.041274 | -0.129645 | 0.016984 | -0.001799 | -0.008065 | 0.022942 | |
| **tBodyAccmeanY** | 0.041274 | 1.000000 | 0.225980 | -0.054264 | -0.059066 | -0.077051 | -0.052501 | |
| **tBodyAccmeanZ** | -0.129645 | 0.225980 | 1.000000 | -0.038578 | -0.048340 | -0.042342 | -0.037851 | |
| **tBodyAccstdX** | 0.016984 | -0.054264 | -0.038578 | 1.000000 | 0.910636 | 0.896031 | 0.998828 | |
| **tBodyAccstdY** | -0.001799 | -0.059066 | -0.048340 | 0.910636 | 1.000000 | 0.874501 | 0.909197 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **angletBodyGyroJerkMeangravityMean** | 0.049701 | 0.092905 | -0.021375 | -0.033609 | -0.018611 | -0.029401 | -0.033386 | |
| **angleXgravityMean** | -0.058421 | -0.017138 | -0.013933 | -0.382696 | -0.383742 | -0.383490 | -0.381896 | |
| **angleYgravityMean** | 0.034220 | -0.030253 | -0.007318 | 0.401433 | 0.467572 | 0.427469 | 0.397630 | |
| **angleZgravityMean** | 0.038936 | -0.027410 | -0.051057 | 0.388747 | 0.405681 | 0.488541 | 0.386152 | |
| **subject** | 0.005077 | 0.003163 | 0.021476 | -0.068487 | -0.036466 | -0.028508 | -0.064529 | |

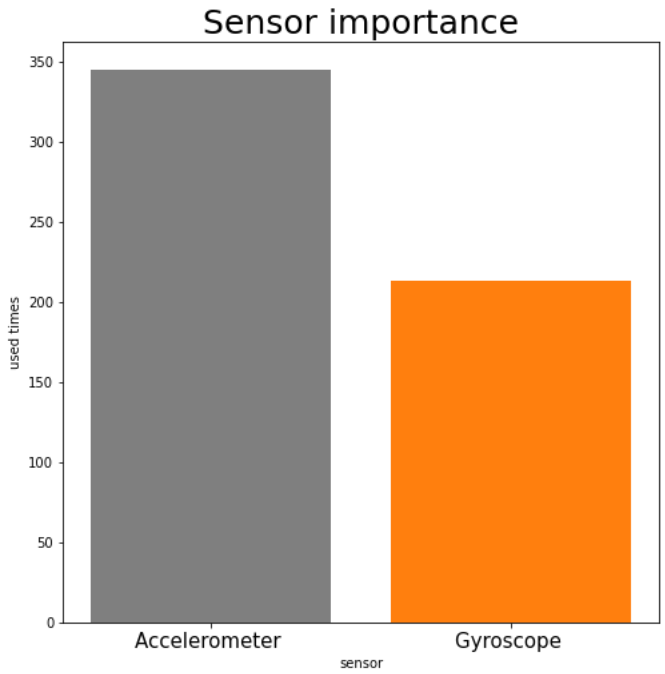
562 rows × 562 columns

# Exploratory Data Analysis

Finding the right sensor essential for categorizing participants based on their walking style.

In [14]:
```
Acc=0
Gyro=0
other=0
for i in df.columns:
    if'Acc'in i:
        Acc += 1
    elif'Gyro'in i:
        Gyro += 1
    else:
        other += 1

plt.figure(figsize=(8,8))
plt.bar(x = ['Accelerometer ','Gyroscope '], height = [Acc,Gyro], color=['tab:gray', 'tab:orange'])
plt.title("Sensor importance", fontsize = 25)
plt.xlabel("sensor")
plt.ylabel("used times")
plt.xticks(fontsize=15)
plt.show()
```



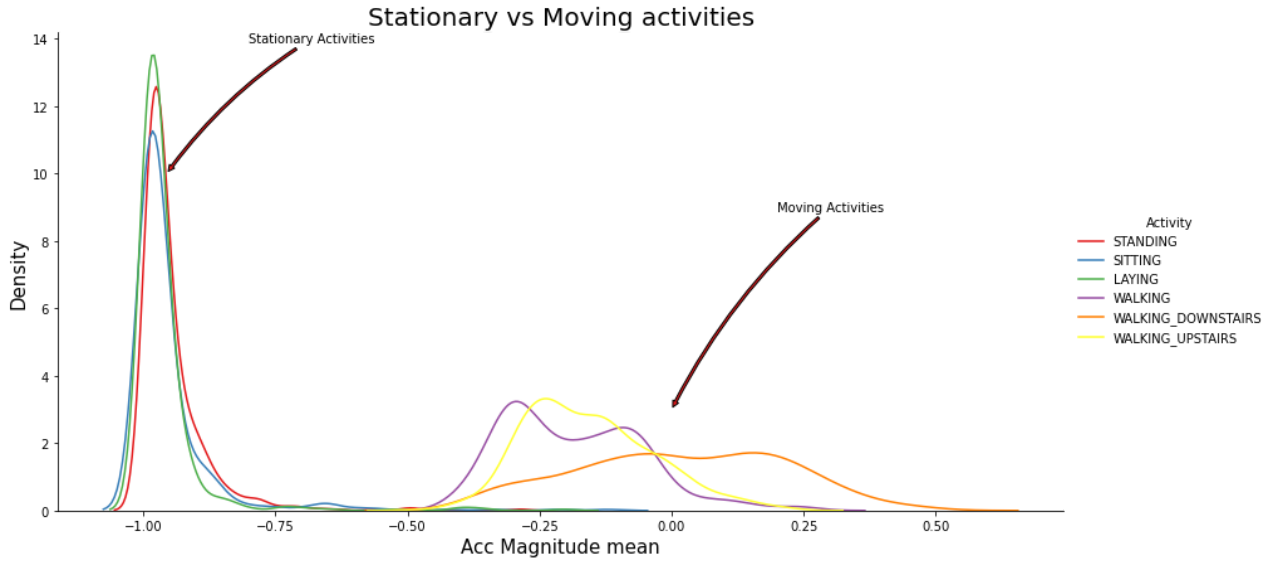

From the above analysis we can define that the most widely used sesor is accelerometer.

Type *Markdown* and LaTeX: $\alpha^2$

Displaying the distribution of moving and stationary activities.

In [15]:
```
sns.set_palette("Set1", desat=1)
facetgrid = sns.FacetGrid(df, hue='Activity', size=6,aspect=2)
facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities", xy=(-0.956,10), xytext=(-0.8, 14), size=10,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))

plt.annotate("Moving Activities", xy=(0,3), xytext=(0.2, 9), size=10,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))
plt.title('Stationary vs Moving activities', fontsize=20)
plt.xlabel("Acc Magnitude mean", size=15)
plt.ylabel('Density', size=15)
plt.show()
```

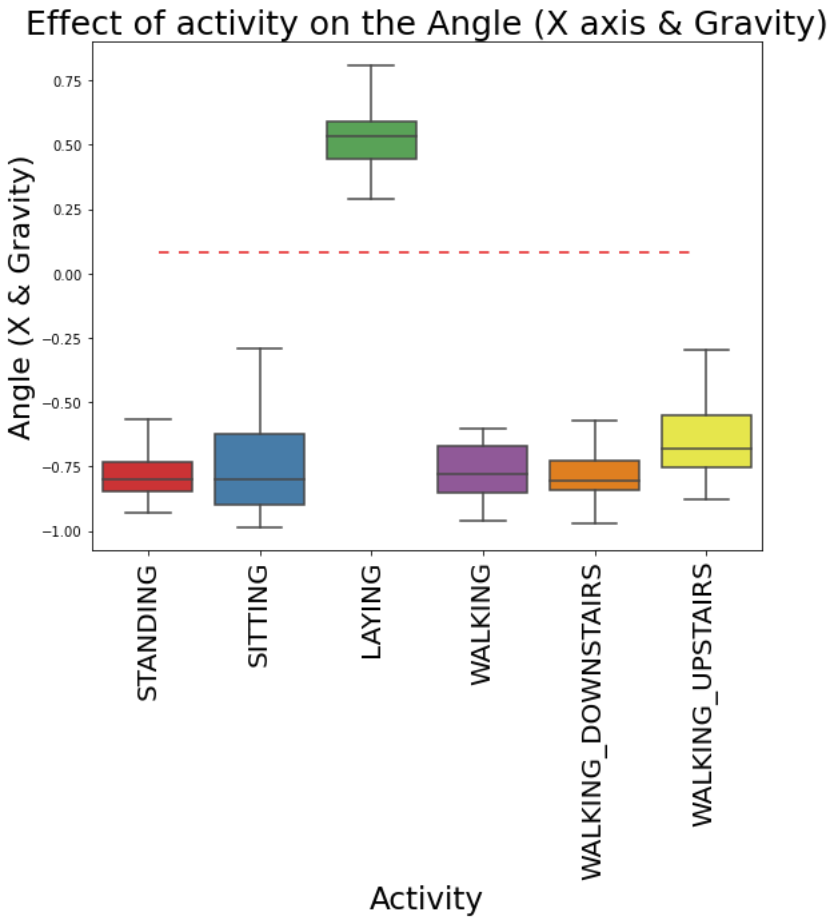



It is clear that the distribution of stationary acitivities is greater than that of the moving activities.

Type *Markdown* and LaTeX: $\alpha^2$

Examining the relationship between x and gravity

In [16]:
```
plt.figure(figsize=(9,7))
sns.boxplot(x='Activity', y= 'angleXgravityMean', data=df, showfliers=False)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,dashes=(5,5))
plt.title('Effect of activity on the Angle (X axis & Gravity)', fontsize=25)
plt.xlabel("Activity", size=23)
plt.ylabel('Angle (X & Gravity)', size=22)
plt.xticks(rotation = 90, fontsize = 20)
plt.show()
```

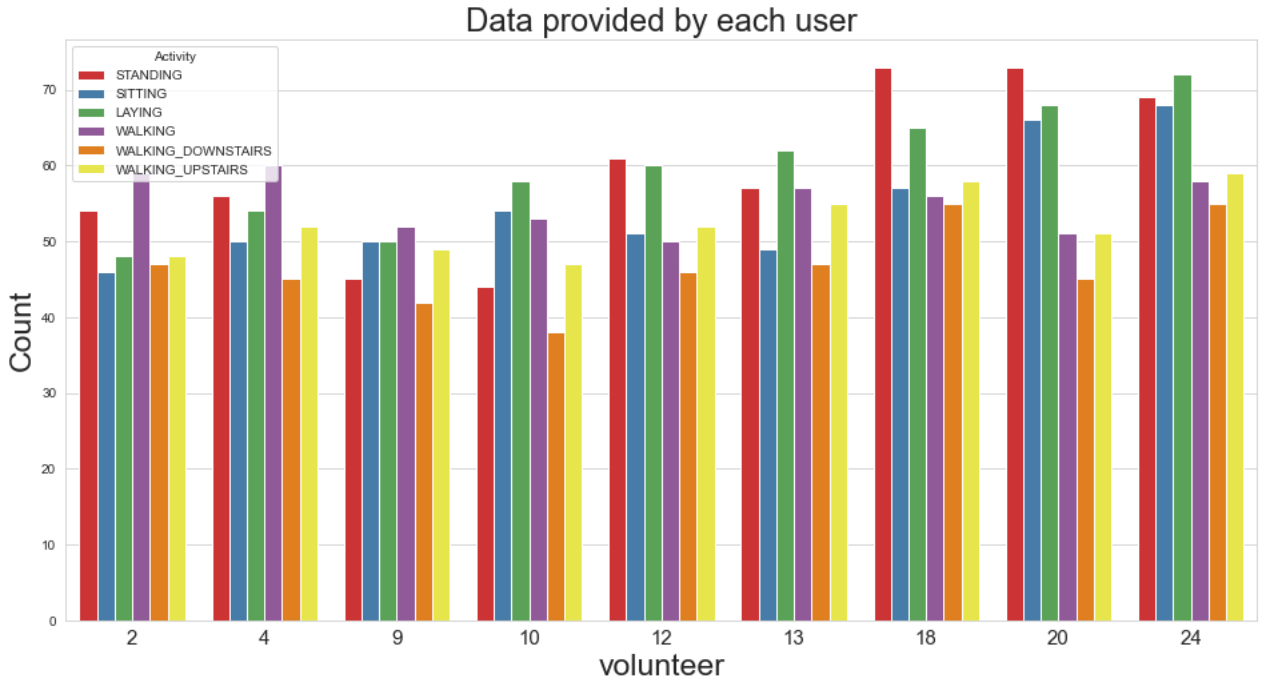



From the observation it is clear that the, Activity is laying if angle(X,gravityMean) > 0

Type *Markdown* and LaTeX: $\alpha^2$

Plotting the data provided by each user.

In [17]:
```
sns.set_style('whitegrid')
plt.figure(figsize=(16,8))
sns.color_palette("tab10")
plt.title('Data provided by each user', fontsize=24)
sns.countplot(x='subject',hue='Activity', data = df)
plt.xlabel("volunteer", size=23)
plt.ylabel("Count", size=23)
plt.xticks(size=15)
plt.show()
```

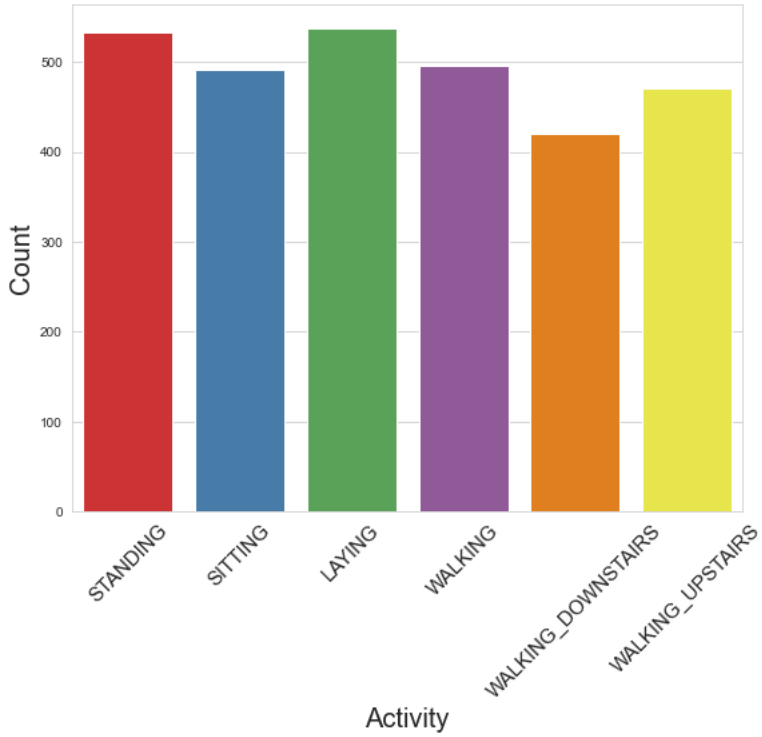



Checking for the imbalance in the data

In [40]:
```
fig = px.pie(df,names='Activity', width=700)
fig.update_layout(
    title={
        'text': "Activities distribution in the data",
        'y':0.95,
        'x':0.40,
        'xanchor': 'center',
        'yanchor': 'top'},
         legend_title ="Activities Involved",
         font=dict(family="Arial",size=12))
fig.show()
```

We will not conduct resampling because the data is balanced.

In [19]:
```
#plotting a countplot for the number of activities performed
plt.figure(figsize=(9,7))
axis=sns.countplot(x="Activity",data=df)
plt.xticks(x=df['Activity'],rotation=45, size= 15)
plt.xlabel('Activity', size = 20)
plt.ylabel('Count', size= 20)
plt.show()
```




Type *Markdown* and LaTeX: $\alpha^2$

# Implementation Using Machine Learning

# 1.Unsupervised Learning

## *Using T-SNE*

```
Data visualization following T-SNE dimension reduction:
```

```
1. With the use of a two- or three-dimensional map, the statistical technique known as T-distributed stochastic neighbor
embedding (t-SNE) makes it possible to see high-dimensional data.
2. It is a nonlinear dimensionality reduction method that works well for visualizing high-dimensional data embedded in a low-
dimensional space with two or three dimensions.
3. In particular, it models each high-dimensional object by a two- or three-dimensional point in a way that, with a high
probability, models similar objects by neighboring points and dissimilar objects by distant points.
4. There are two primary steps in the t-SNE algorithm. First, t-SNE builds a probability distribution across pairs of high-
dimensional objects, assigning a higher probability to comparable items and a lower likelihood to dissimilar points.
5. Second, t-SNE minimizes the Kullback-Leibler divergence (KL divergence) between the two distributions with regard to the
positions of the points in the map by defining a similar probability distribution over the points in the low-dimensional map.
Although the original approach bases its similarity metric on the Euclidean distance between objects, this can be modified as
necessary.
```

In [20]:

```
from sklearn.manifold import TSNE
def perform_tsne(x_data, y_data, perplexities, n_iter=1000, img_name_prefix='t-sne'):

    for index,perplexity in enumerate(perplexities):
        # perform t-sne
        x_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(x_data)

        # prepare the data for seaborn
        print('Creating plot for this t-sne visualization..')
        df = pd.DataFrame({'x':x_reduced[:,0], 'y':x_reduced[:,1] ,'label':y_data})

        # draw the plot in appropriate place in the grid
        sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,\
                   palette="Set1",markers=['^','v','s','o', '1','2'])
        plt.title("perplexity : {} and max_iter : {}".format(perplexity, n_iter))
        img_name = img_name_prefix + '_perp_{}_iter_{}.png'.format(perplexity, n_iter)
        #print('saving this plot as image in present working directory...')
        plt.savefig(img_name)
        plt.show()
```

In [21]:
```
x_pre_tsne = df.drop(['subject', 'Activity'], axis=1)
y_pre_tsne = df['Activity']
perform_tsne(x_data = x_pre_tsne,y_data=y_pre_tsne, perplexities =[20])
```

```
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 2947 samples in 0.008s...
[t-SNE] Computed neighbors for 2947 samples in 0.290s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2947
[t-SNE] Computed conditional probabilities for sample 2000 / 2947
[t-SNE] Computed conditional probabilities for sample 2947 / 2947
[t-SNE] Mean sigma: 1.411132
[t-SNE] Computed conditional probabilities in 0.032s
[t-SNE] Iteration 50: error = 83.8818512, gradient norm = 0.1058002 (50 iterations in 0.476s)
[t-SNE] Iteration 100: error = 74.4007645, gradient norm = 0.0386177 (50 iterations in 0.312s)
[t-SNE] Iteration 150: error = 73.2140350, gradient norm = 0.0161277 (50 iterations in 0.306s)
[t-SNE] Iteration 200: error = 72.7728653, gradient norm = 0.0105243 (50 iterations in 0.278s)
[t-SNE] Iteration 250: error = 72.5276642, gradient norm = 0.0099841 (50 iterations in 0.320s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 72.527664
[t-SNE] Iteration 300: error = 1.7113024, gradient norm = 0.0011812 (50 iterations in 0.283s)
[t-SNE] Iteration 350: error = 1.3653015, gradient norm = 0.0005107 (50 iterations in 0.308s)
[t-SNE] Iteration 400: error = 1.2375126, gradient norm = 0.0003222 (50 iterations in 0.265s)
[t-SNE] Iteration 450: error = 1.1726220, gradient norm = 0.0002397 (50 iterations in 0.359s)
[t-SNE] Iteration 500: error = 1.1333523, gradient norm = 0.0001896 (50 iterations in 0.443s)
[t-SNE] Iteration 550: error = 1.1070287, gradient norm = 0.0001670 (50 iterations in 0.463s)
[t-SNE] Iteration 600: error = 1.0887699, gradient norm = 0.0001384 (50 iterations in 0.473s)
[t-SNE] Iteration 650: error = 1.0752211, gradient norm = 0.0001364 (50 iterations in 0.454s)
[t-SNE] Iteration 700: error = 1.0655441, gradient norm = 0.0001263 (50 iterations in 0.459s)
[t-SNE] Iteration 750: error = 1.0574886, gradient norm = 0.0001195 (50 iterations in 0.448s)
[t-SNE] Iteration 800: error = 1.0504501, gradient norm = 0.0001042 (50 iterations in 0.478s)
[t-SNE] Iteration 850: error = 1.0440609, gradient norm = 0.0001200 (50 iterations in 0.478s)
[t-SNE] Iteration 900: error = 1.0393206, gradient norm = 0.0000987 (50 iterations in 0.462s)
[t-SNE] Iteration 950: error = 1.0352232, gradient norm = 0.0000938 (50 iterations in 0.453s)
[t-SNE] Iteration 1000: error = 1.0309396, gradient norm = 0.0000929 (50 iterations in 0.428s)
[t-SNE] KL divergence after 1000 iterations: 1.030940
Creating plot for this t-sne visualization..
```


Type *Markdown* and LaTeX: $\alpha^2$

Conversion of non-numerical labels into numbers

In [22]:
```
df['subject'].unique()
```

Out[22]:
```
array([ 2,  4,  9, 10, 12, 13, 18, 20, 24], dtype=int64)
```

In [23]:
```
x=pd.DataFrame(df.drop(['Activity','subject'],axis=1))
y=df.Activity.values.astype(object)
print(x.shape)
y.shape
```

```
(2947, 561)
```

Out[23]:
```
(2947,)
```

In [24]:
```
encoder=preprocessing.LabelEncoder()
encoder.fit(y)
y=encoder.transform(y)
y.shape
```

Out[24]:
```
(2947,)
```

In [25]:
```
encoder.classes_
```

Out[25]:
```
array(['LAYING', 'SITTING', 'STANDING', 'WALKING', 'WALKING_DOWNSTAIRS',
       'WALKING_UPSTAIRS'], dtype=object)
```

Type *Markdown* and LaTeX: $\alpha^2$

Aspect Scaling

In [26]:
```
scaler=StandardScaler()
x=scaler.fit_transform(x)
```

In [27]:
```
x[2]
```

Out[27]:
```
array([ 0.02458676, -0.31805537, -0.22848456, -0.92159817, -0.93405419,
       -0.90710686, -0.91689082, -0.93381265, -0.91263854, -0.91098199,
       -0.9688179 , -0.884576  ,  0.94144134,  0.83579326,  0.79277451,
       -0.9524716 , -0.75960247, -0.75774762, -0.77596707, -0.89676576,
       -0.91546768, -0.90468645, -1.17738137, -1.39137457, -1.09630939,
        1.18131711, -0.96032404,  0.24505718,  0.38854293,  0.10668822,
       -0.51735937,  1.57591019, -2.26950304,  0.70527362, -1.31655781,
        2.53050174, -2.02564594,  1.20561283, -0.37581515,  0.52499347,
        0.48422632, -0.69493716,  0.19591306, -0.37703175, -0.47517815,
       -0.44074117, -0.37759935, -0.47681312, -0.41589412,  0.46432355,
       -0.72683216,  0.16496468,  0.50086116, -0.66752007,  0.22851917,
        0.52186933,  0.49424693, -0.32505496, -0.38616303, -0.38450172,
       -0.47681341, -0.36722565, -0.37281956, -0.49532517,  0.26326505,
        0.8196707 , -0.99789412,  1.15538907, -1.27875734,  0.36551654,
       -0.53416477,  0.71284273, -0.88665711, -0.03420078, -0.02245298,
        0.08093258, -0.13913004,  0.57645336, -0.31549319,  0.74256772,
       -0.05566783, -0.06810805, -0.06206911, -0.90036707, -0.85904441,
       -0.80829999, -0.89667692, -0.87470556, -0.82696703, -0.8570322 ,
       -0.783537  , -0.67301446,  0.89956633,  0.82853965,  0.78826102,
       -0.89638094, -0.73775921, -0.69355895, -0.68057901, -0.88054387,
       -0.89589941, -0.82863277, -1.01531101, -0.99724205, -0.79824   ,
        1.00093468,  0.05196759,  0.03182175,  1.15852611,  0.64817929,
       -0.43202512,  1.03932065,  1.32951224,  1.09437257, -1.47213485,
        1.15019839,  2.35439251,  0.2213994 ,  0.63423422,  2.41464615,
       -0.04182236, -0.68477001,  0.49912549, -0.87675684, -0.87901207,
       -0.89864041, -0.8726957 , -0.87458714, -0.89216295, -0.85527387,
       -0.85106909, -0.78496791,  0.81229012,  0.78349702,  0.89731415,
       -0.8508152 , -0.72053445, -0.64833644, -0.7137331 , -0.86729085,
       -0.86061246, -0.86068116, -1.1047084 , -2.41322282,  0.60939641,
        0.07051036, -0.60248446,  0.71284155, -0.32257992, -0.48710734,
       -0.86500649,  2.78110776, -2.96240781, -0.1421193 , -0.53604452,
        1.94313789, -2.76277868, -0.11221126,  0.72463315, -1.69107781,
       -0.03111467,  0.07226393, -0.03135221, -0.88083474, -0.79752655,
       -0.81683918, -0.87956868, -0.80758212, -0.82893466, -0.84757721,
       -0.77127483, -0.74739618,  0.84306564,  0.72008385,  0.75802479,
       -0.86627176, -0.69552243, -0.55783251, -0.63483843, -0.86628904,
       -0.80343857, -0.83256162, -0.79007362, -0.83805611, -0.6269532 ,
        0.23639453, -0.99993873,  0.1170049 , -0.0032827 ,  0.3952807 ,
       -1.81405306,  1.2606062 ,  1.23247903,  0.44175424, -0.75523975,
        0.44055309,  2.34729516,  0.48634668, -0.82929374, -2.92497974,
       -0.94930532, -0.93487533, -0.94080469, -0.9213348 , -0.80292191,
       -0.94930532, -0.82963968, -0.92997233, -1.02958477,  0.73196351,
       -0.31366557, -0.18623464, -0.38664838, -0.94930532, -0.93487533,
       -0.94080469, -0.9213348 , -0.80292191, -0.94930532, -0.82963968,
       -0.92997233, -1.02958477,  0.73196351, -0.31366557, -0.18623464,
       -0.38664838, -0.89580055, -0.86182627, -0.86950089, -0.84630054,
       -0.80483197, -0.89580055, -0.77156262, -0.87569534, -0.84428897,
        1.40041846,  0.24967747, -0.91498292, -1.31788374, -0.86610745,
       -0.93361976, -0.9346229 , -0.91214546, -0.72954732, -0.86610745,
       -0.77306088, -0.92529629, -0.4746301 , -0.74346489,  0.15429178,
        1.37991023, -1.47007538, -0.86649491, -0.81796403, -0.83489464,
       -0.80014116, -0.72695839, -0.86649491, -0.66107614, -0.8597695 ,
       -0.91595872,  0.83103078,  0.11013176,  1.37248782, -2.62232764,
       -0.91790166, -0.90513257, -0.88297362, -0.91752287, -0.94062919,
       -0.90525406, -0.92087547, -0.91254277, -0.88195259, -0.88772335,
       -0.92688243, -0.87477222, -0.6190666 , -0.63231197, -0.53310518,
       -0.93085083, -0.75809395, -0.75373012, -0.77613181, -0.87960314,
       -0.83945092, -0.8200634 , -1.07699699, -0.88515771, -0.73280904,
       -0.91534474, -0.85934683, -0.672708  ,  1.46889873,  0.46647877,
        0.48319269, -0.58671422, -0.60131468, -1.1002718 , -0.88764104,
       -0.32682442, -0.39129471, -0.72331697, -0.68665287, -0.64841097,
       -0.59791168, -0.61558049, -0.6269943 , -0.5568108 , -0.38313713,
       -0.74919641, -0.66223632, -0.64259298, -0.51605892, -0.7568711 ,
       -0.63549732, -0.74913863, -0.63736443, -0.63010466, -0.58181251,
       -0.60194605, -0.6306736 , -0.55521368, -0.35026826, -0.74531646,
       -0.64035557, -0.63325494, -0.50192029, -0.75020973, -0.62579655,
       -0.73984809, -0.6061399 , -0.60022504, -0.58763445, -0.57479186,
       -0.6148681 , -0.57808805, -0.39491076, -0.75825242, -0.62250417,
       -0.61103745, -0.55362977, -0.77116006, -0.61736907, -0.89754166,
       -0.86575152, -0.82052729, -0.89705425, -0.84630046, -0.7872724 ,
       -0.89416686, -0.84877673, -0.80154061, -0.87664338, -0.83520932,
       -0.74364937, -0.64836055, -0.67003591, -0.62604771, -0.89370925,
       -0.73769272, -0.69349071, -0.68047566, -0.86710837, -0.84259173,
       -0.80169203, -1.02612333, -0.95476766, -0.81993434, -0.80014968,
       -0.03063189, -0.46174458,  1.07744977, -0.20829873,  0.16581538,
       -1.14134842, -0.90500401,  0.12928417,  0.00685378,  1.12082393,
        1.19863159, -0.67199276, -0.68850033, -0.64166203, -0.58582543,
       -0.59970159, -0.63254603, -0.57651796, -0.3519661 , -0.72987338,
       -0.65325239, -0.63924031, -0.58024791, -0.7407632 , -0.63292003,
       -0.65912331, -0.64035793, -0.62425519, -0.58252607, -0.60294114,
       -0.62722481, -0.60482683, -0.34607142, -0.66529798, -0.6342617 ,
       -0.63917525, -0.60758137, -0.68302156, -0.63642235, -0.66267947,
       -0.59698092, -0.60006279, -0.57630038, -0.55566323, -0.59308733,
       -0.5596273 , -0.33536768, -0.65208544, -0.62318029, -0.5926455 ,
       -0.56377863, -0.66882918, -0.60607008, -0.90106559, -0.8670249 ,
       -0.89112857, -0.85927492, -0.87009699, -0.88959775, -0.88734691,
```

```
-0.86947296, -0.88351249, -0.80647601, -0.81007552, -0.85330375,
-0.60278667, -0.5904336 , -0.63987041, -0.91337934, -0.700833  ,
-0.64978753, -0.70377096, -0.86214645, -0.81097053, -0.82504689,
-0.8157727 , -0.85576617, -0.82113889, -0.61580937, -0.52166842,
-0.85217575, -1.53061982, -0.16117802,  0.40525828,  0.36063493,
 0.16852714, -0.23550045, -0.40862435, -1.08895015, -1.00091662,
-0.6525072 , -0.67357092, -0.62101203, -0.58823204, -0.53252892,
-0.60387833, -0.50851957, -0.38226289, -0.68638468, -0.64145578,
-0.57512007, -0.46070342, -0.69675856, -0.61411814, -0.58763397,
-0.52460174, -0.45459512, -0.50639736, -0.51845149, -0.53019536,
-0.48973406, -0.31346766, -0.63610416, -0.4791213 , -0.54339975,
-0.45443891, -0.64449052, -0.53195039, -0.66733605, -0.54468788,
-0.54025034, -0.50792706, -0.44674493, -0.52639945, -0.46859531,
-0.33722183, -0.6867986 , -0.57085319, -0.48676414, -0.43438959,
-0.69619092, -0.51747116, -0.91135809, -0.94061763, -0.91627059,
-0.9514143 , -0.64614059, -0.91135809, -0.75295456, -0.85751993,
-0.9693423 , -0.98684755,  1.01809005, -1.35244847, -1.00353379,
-0.86416982, -0.84970156, -0.86029575, -0.83093481, -0.6668091 ,
-0.86416982, -0.72827996, -0.86951573, -0.88503422, -0.63841902,
-0.08062071,  0.15031686, -0.19482307, -0.91143131, -0.92768004,
-0.9083104 , -0.92225327, -0.74015314, -0.91143131, -0.69963197,
-0.83108085, -0.88727012, -0.09989342, -0.51067686, -1.19581979,
-1.0734949 , -0.83273403, -0.789049  , -0.80539166, -0.77703469,
-0.66784324, -0.83273403, -0.58152403, -0.82815561, -0.92521135,
-0.25673473, -0.29156328,  0.27482118, -0.13498919, -0.11967085,
 0.44607161,  0.03791916,  0.32393752, -0.37093228,  0.63284514,
-0.12685534])
```

Type *Markdown* and LaTeX: $\alpha^2$

Dividing the Training and the Validation sets

In [28]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=100)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[28]: ((2357, 561), (590, 561), (2357,), (590,))

Type *Markdown* and LaTeX: $\alpha^2$

## 2. Supervised Learning Models Using:

1. SVM
2. RANDOM FOREST CLASSIFIER
3. CONFUSION MATRIX
4. DECISION TREE CLASSIFIER

## *1.Support Vector Machine (SVM)*

Support Vector Machine, sometimes known as SVM, is a linear model for classification and regression issues. It can solve linear and non-linear problems and is useful for a wide range of practical applications. SVM's basic principle is as follows: The algorithm produces a line or a hyperplane that divides the data into classes.

In [29]:
```
svc=SVC()
svc.fit(x_train,y_train)
```

Out[29]: SVC()

In [30]:
```
y_pred=svc.predict(x_test)
print('Score for model correctness using default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Score for model correctness using default hyperparameters: 0.9814
```

several hyperparameters where kernel use rbf and C=100.0

In [31]:
```
svc_2=SVC(kernel='rbf',C=100.0)


# fitting the classifier to the training data
svc_2.fit(x_train,y_train)

# making predictions on test set
y_pred2 = svc_2.predict(x_test)

# computing and printing the accuracy score
print('Score of model correctness using rbf kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred2)))
```

```
Score of model correctness using rbf kernel and C=100.0 : 0.9881
```

Type *Markdown* and LaTeX: $\alpha^2$

## 2. Random Forest Classifier

The random forest classifier can tackle regression or classification issues. The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is made up of a data sample selected from a training set with replacement, known as the bootstrap sample.

In [32]:
```
random_clf=RandomForestClassifier(random_state=5)
random_clf.fit(x_train,y_train)
```

Out[32]:
```
RandomForestClassifier(random_state=5)
```

In [33]:
```
# computing and printing the accuracy score
random_clf.score(x_test,y_test)
```

Out[33]:
```
0.9796610169491525
```

Type *Markdown* and LaTeX: $\alpha^2$

## 3. Confusion Matrix

A Confusion matrix is a N x N matrix used to evaluate the performance of a classification model, where N is the number of target classes. In the matrix, the actual goal values are contrasted with those that the machine learning model anticipated. This provides a comprehensive picture of how well the classification model is working and the kind of errors it is making.

In [34]:
```
confusion_matrix(y_test, y_pred)
```

Out[34]:
```
array([[113,   0,   0,   0,   1,   0],
       [  0,  88,   3,   0,   0,   0],
       [  0,   6, 105,   0,   0,   0],
       [  0,   0,   0,  86,   0,   0],
       [  0,   0,   0,   1,  94,   0],
       [  0,   0,   0,   0,   0,  93]], dtype=int64)
```

In [35]:
```
#calculating the accuracy Score
accuracy_score(y_test, y_pred)
```

Out[35]:
```
0.9813559322033898
```

In [36]:
```
#calculating the precision Score
precision_score(y_test, y_pred, average=None)
```

Out[36]:
```
array([1.        , 0.93617021, 0.97222222, 0.98850575, 0.98947368,
       1.        ])
```

In [37]:
```
#calculating the F1 Score
f1_score(y_test, y_pred, average=None)
```

Out[37]:
```
array([0.99559471, 0.95135135, 0.95890411, 0.99421965, 0.98947368,
       1.        ])
```

Type *Markdown* and LaTeX: $\alpha^2$

## 4. Decision Tree Classifier

The non-parametric supervised learning approach used for classification and regression applications is the decision tree. It has a tree-like structure with a root node, branches, internal nodes, and leaf nodes.

In [38]:
```
dtree=DecisionTreeClassifier()
dtree.fit(x_train,y_train)
```

Out[38]:
```
DecisionTreeClassifier()
```

In [39]:
```
# 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       114
           1       0.85      0.89      0.87        91
           2       0.91      0.87      0.89       111
           3       0.92      0.97      0.94        86
           4       0.95      0.93      0.94        95
           5       0.96      0.94      0.95        93

    accuracy                           0.93       590
   macro avg       0.93      0.93      0.93       590
weighted avg       0.93      0.93      0.93       590
```

Reference link: Kaggle: https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones (https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones)

In [ ]:
```
```