# Data Science Nigeria: Introductory Machine Learning Training

# WEEK 4: INTRODUCTION TO CLASSIFICATION

Binary/ Multi-class classification

Classification Algorithms

Evaluation of Classification Models Performance

Error/ Cost function

Confusion matrix

Precision

AUC

In Classification, we predict the category a data belongs to ie. Classification algorithms are used to predict labels

- Spam Detection
- Churn Prediction
- Sentiment Analysis
- Dog Breed Detection

## TYPES OF CLASSIFICATION TASK

- Binary classification eg. e-mail spam detection (1 ->spam; or 0→not spam), biometric identification, whether a customer will default or Not
- Multi-class classification eg. digit recognition (where classes go from 0 to 9), predicting a party that wins the election,

  <img src= './class.png', alt = "Data Science Nigeria" width= 600, height = 300/>

Classification Algorithms

1. Logistic Regression
2. Naive Bayes Classifier
3. Nearest Neighbor
4. Support Vector Machines
5. Decision Trees
6. Boosted Trees
7. Random Forest

# Import Modules

In [148]:

```
#import modules

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

%matplotlib inline
```

In [149]:

```
data= pd.read_csv("Social_Network_Ads.csv", delimiter= ",")
data.head()
```

Out[149]:

| | User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|---|
| **0** | 15624510 | Male | 19 | 19000 | 0 |
| **1** | 15810944 | Male | 35 | 20000 | 0 |
| **2** | 15668575 | Female | 26 | 43000 | 0 |
| **3** | 15603246 | Female | 27 | 57000 | 0 |
| **4** | 15804002 | Male | 19 | 76000 | 0 |

In [150]:

```
data['Gender'].value_counts()
```

Out[150]:

```
Female    204
Male      196
Name: Gender, dtype: int64
```

In [ ]:

```

```

# Pre-processing Data

In [151]:

```
#creating a LabelEncoder object
from sklearn.preprocessing import LabelEncoder

le= LabelEncoder()
#invoking fit_transform method on object
data['Gender']=le.fit_transform(data['Gender'])
```

In [152]:

```
data['Gender'].value_counts()
```

Out[152]:

```
0    204
1    196
Name: Gender, dtype: int64
```

## Exploratory analysis

In [153]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
User ID            400 non-null int64
Gender             400 non-null int32
Age                400 non-null int64
EstimatedSalary    400 non-null int64
Purchased          400 non-null int64
dtypes: int32(1), int64(4)
memory usage: 14.1 KB
```

In [154]:

```
# employees that did not buy and those that bought

left= data.groupby('Purchased')
left.mean()
```

Out[154]:

| | User ID | Gender | Age | EstimatedSalary |
|---|---|---|---|---|
| **Purchased** | | | | |
| **0** | 1.569116e+07 | 0.505837 | 32.793774 | 60544.747082 |
| **1** | 1.569222e+07 | 0.461538 | 46.391608 | 86272.727273 |

In [155]:

```
data.describe()
```

Out[155]:

| | User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|---|
| **count** | 4.000000e+02 | 400.000000 | 400.000000 | 400.000000 | 400.000000 |
| **mean** | 1.569154e+07 | 0.490000 | 37.655000 | 69742.500000 | 0.357500 |
| **std** | 7.165832e+04 | 0.500526 | 10.482877 | 34096.960282 | 0.479864 |
| **min** | 1.556669e+07 | 0.000000 | 18.000000 | 15000.000000 | 0.000000 |
| **25%** | 1.562676e+07 | 0.000000 | 29.750000 | 43000.000000 | 0.000000 |
| **50%** | 1.569434e+07 | 0.000000 | 37.000000 | 70000.000000 | 0.000000 |
| **75%** | 1.575036e+07 | 1.000000 | 46.000000 | 88000.000000 | 1.000000 |
| **max** | 1.581524e+07 | 1.000000 | 60.000000 | 150000.000000 | 1.000000 |

# Data Visualization

Users that purchased these Ads

In [188]:

```
left_count = left.count()
```

In [189]:

```
left_count
```

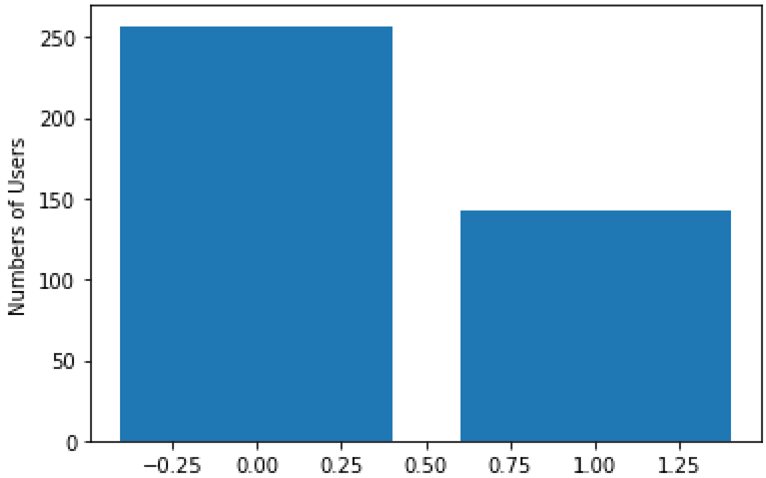
Out[189]:

| | User ID | Gender | Age | EstimatedSalary |
|---|---|---|---|---|
| **Purchased** | | | | |
| **0** | 257 | 257 | 257 | 257 |
| **1** | 143 | 143 | 143 | 143 |

In [156]:

```python
#how many users were in each category ?

plt.bar(left_count.index.values, left_count['User ID'])
plt.ylabel("Numbers of Users")
plt.show()
```




In [161]:

```python
# ratio of users that did bought the Ads were only 36%

data.Purchased.value_counts()
float(data.Purchased.value_counts()[1])/len(data) * 100
```

Out[161]:

```
35.75
```

## A little bit of feature engineering !

1) Using an Age category

IGen[1-24], Millenials[24-39], GenX[40-54], BabyBoomers[55-73]

In [162]:

```
data['Age Category'] = ['iGen' if 0<age<25 else 'Millenials' if 24<age<40 else 'GenX' i
 for age in list(data['Age'].values)]
```

In [163]:

```
data.head()
```

Out[163]:

| | User ID | Gender | Age | EstimatedSalary | Purchased | Age Category |
|---|---|---|---|---|---|---|
| **0** | 15624510 | 1 | 19 | 19000 | 0 | iGen |
| **1** | 15810944 | 1 | 35 | 20000 | 0 | Millenials |
| **2** | 15668575 | 0 | 26 | 43000 | 0 | Millenials |
| **3** | 15603246 | 0 | 27 | 57000 | 0 | Millenials |
| **4** | 15804002 | 1 | 19 | 76000 | 0 | iGen |

2) Using an Income category [Inter-quartile Ranges]

In [164]:

```
data['Income Category'] = pd.qcut(data['EstimatedSalary'],3,labels=['Low','Medium','High
```

In [165]:

```
data.head()
```

Out[165]:

| | User ID | Gender | Age | EstimatedSalary | Purchased | Age Category | Income Category |
|---|---|---|---|---|---|---|---|
| **0** | 15624510 | 1 | 19 | 19000 | 0 | iGen | Low |
| **1** | 15810944 | 1 | 35 | 20000 | 0 | Millenials | Low |
| **2** | 15668575 | 0 | 26 | 43000 | 0 | Millenials | Low |
| **3** | 15603246 | 0 | 27 | 57000 | 0 | Millenials | Medium |
| **4** | 15804002 | 1 | 19 | 76000 | 0 | iGen | Medium |

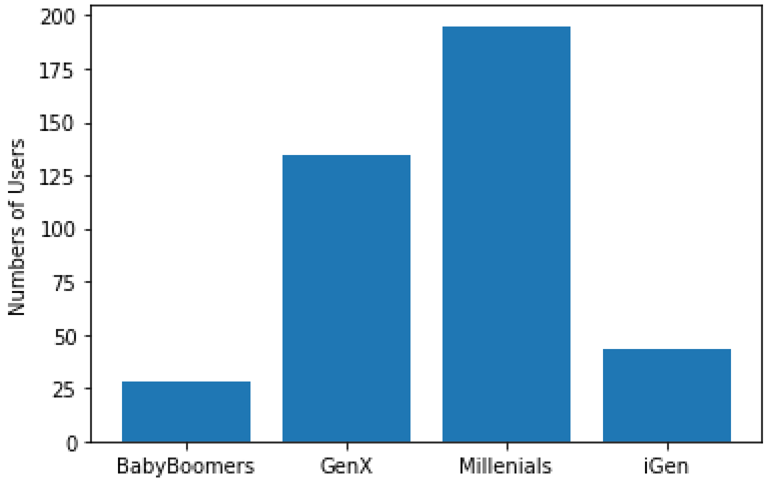
Lets see the Visuals

In [166]:

```
age_cat = data.groupby('Age Category').count()
inc_cat = data.groupby('Income Category').count()
```

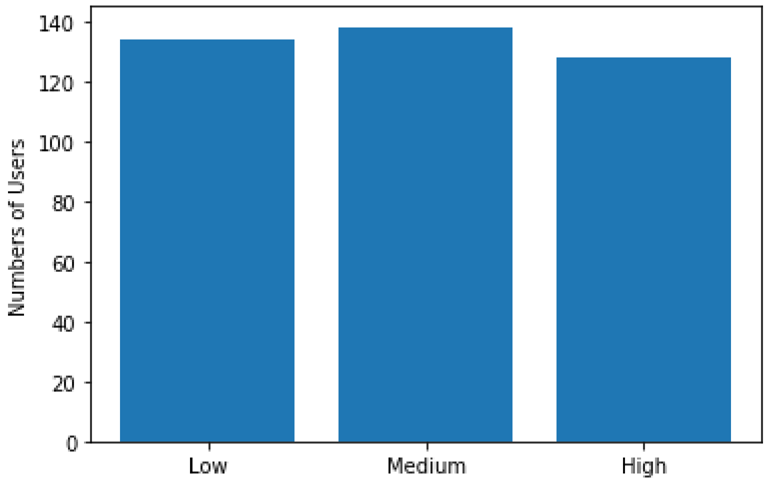
In [167]:

```
plt.bar(age_cat.index.values, age_cat['Age'])
plt.ylabel("Numbers of Users")
plt.show()
```




In [168]:

```
plt.bar(inc_cat.index.values, inc_cat['Age'])
plt.ylabel("Numbers of Users")
plt.show()
```




In [ ]:

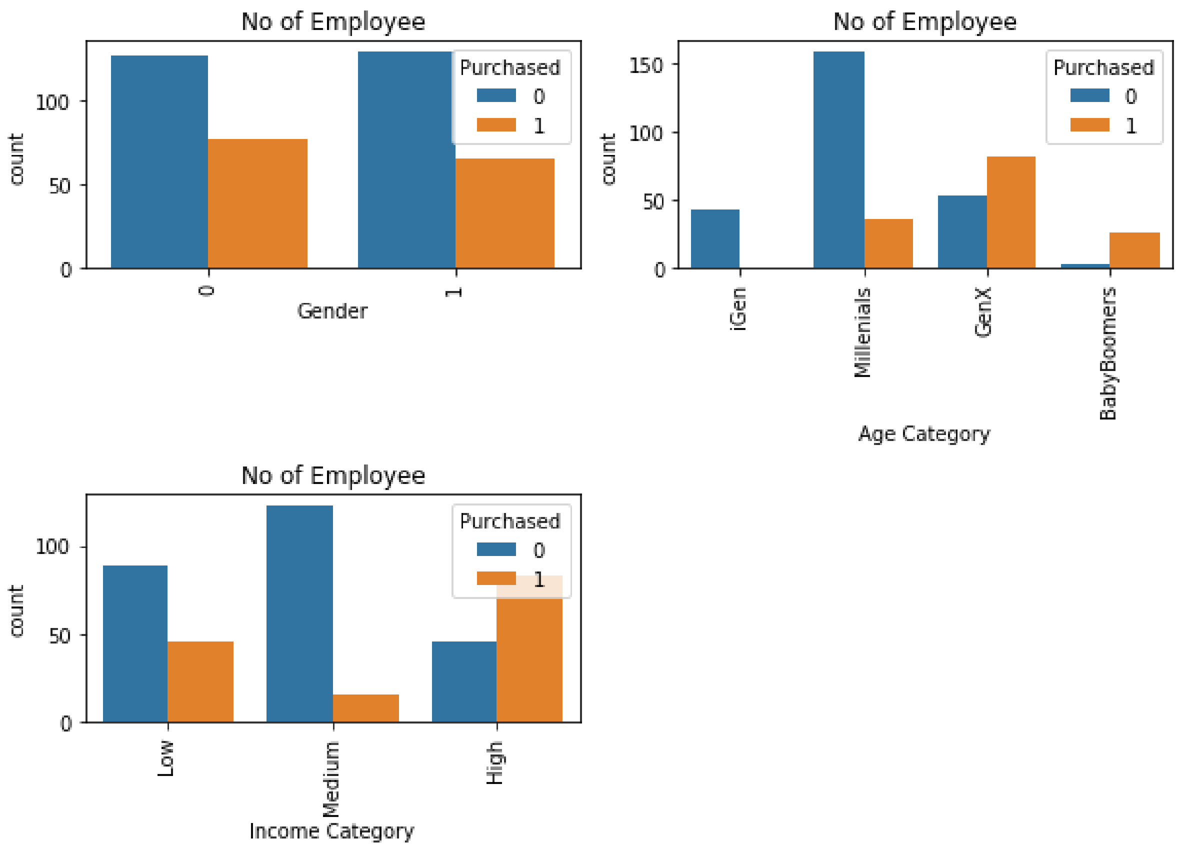
```
# data[data['Income Category'] == 'High'].max()
```

In [169]:

```
#whats the statistics based on those who did not make purchase

features= ['Gender', 'Age Category', 'Income Category']
fig= plt.subplots(figsize= (10,15))

for i,j in enumerate(features):
    plt.subplot(4,2, i+1)
    plt.subplots_adjust(hspace=1.0)
    sns.countplot(x=j, data=data , hue= "Purchased")
    plt.xticks(rotation= 90)
    plt.title("No of Employee")
```




# Model building

In [171]:

```
data.columns
```

Out[171]:

```
Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased',
       'Age Category', 'Income Category'],
      dtype='object')
```

In [172]:

```
# split data into features and target

x = data[['Gender', 'Age', 'EstimatedSalary']]
y=data['Purchased']

#train_test_split
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=42)
```

**Assignment: Use engineered features i.e new columns in training your model. Confirm if it improves our models significantly or not ?**

## Algorithm/ Model 1 : Naive Bayes

In [173]:

```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
```

Out[173]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [174]:

```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

In [175]:

```
# evaluating performance : Accuracy, Precision, Recall
from sklearn import metrics

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
print("Precision: ", metrics.precision_score(y_test, y_pred))
print("Recall: ", metrics.recall_score(y_test, y_pred))
```

```
Accuracy:  0.925
Precision:  0.975
Recall:  0.8297872340425532
```

In [176]:

```
# calculate ROC Curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve (y_test, y_pred)
roc_auc= auc (fpr, tpr)
print  ("ROC AUC", roc_auc)
```

```
ROC AUC 0.9080443019527833
```

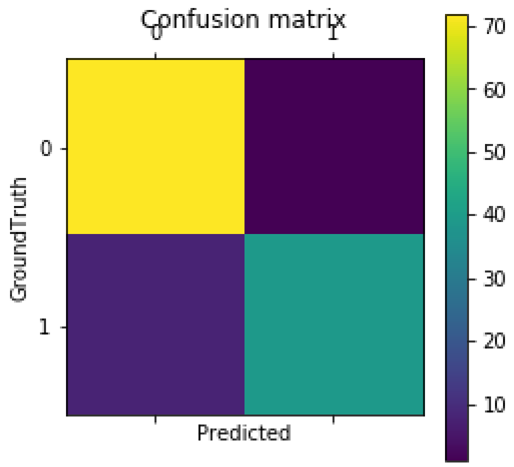
In [177]:

```
# Plot confusion Matrix
def conftable(test,pred, imagename):
    confmatrix= metrics.confusion_matrix(y_test, y_pred)
    plt.matshow(confmatrix)
    plt.title('Confusion matrix')
    plt.colorbar()
    plt.ylabel('GroundTruth')
    plt.xlabel('Predicted')
    plt.savefig(imagename)

    plt.show()
    print(confmatrix)
```

In [178]:

```
conftable(y_test,y_pred,"conf")
```




```
[[72  1]
 [ 8 39]]
```

In [179]:

```
# Ground Truth
pd.Series(y_test).value_counts()
```

Out[179]:

```
0    73
1    47
Name: Purchased, dtype: int64
```

## Algorithm/ Model 2 : Logistic Regression

In [180]:

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(x_train, y_train)
```

```
C:\Users\training.NG-1NW8PX1\Anaconda3\lib\site-packages\sklearn\linear_mo
del\logistic.py:433: FutureWarning: Default solver will be changed to 'lbf
gs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[180]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=0, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [181]:

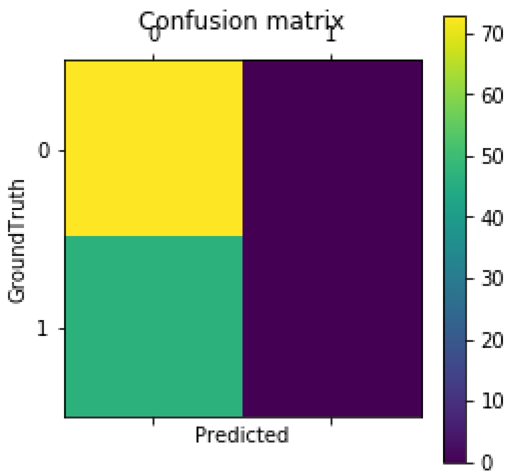
```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

In [182]:

```
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
print("Precision: ", metrics.precision_score(y_test, y_pred))
print("Recall: ", metrics.recall_score(y_test, y_pred))
fpr, tpr, thresholds = roc_curve (y_test, y_pred)
roc_auc= auc (fpr, tpr)
print  ("ROC AUC", roc_auc)
conftable(y_test,y_pred,"conf")
```

```
Accuracy:  0.6083333333333333
Precision:  0.0
Recall:  0.0
ROC AUC 0.5
```

```
C:\Users\training.NG-1NW8PX1\Anaconda3\lib\site-packages\sklearn\metrics\c
lassification.py:1143: UndefinedMetricWarning: Precision is ill-defined an
d being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```




```
[[73  0]
 [47  0]]
```

## Algorithm/ Model 3 : Random Forest

In [183]:

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_sta
classifier.fit(x_train, y_train)
```

Out[183]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entro
py',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

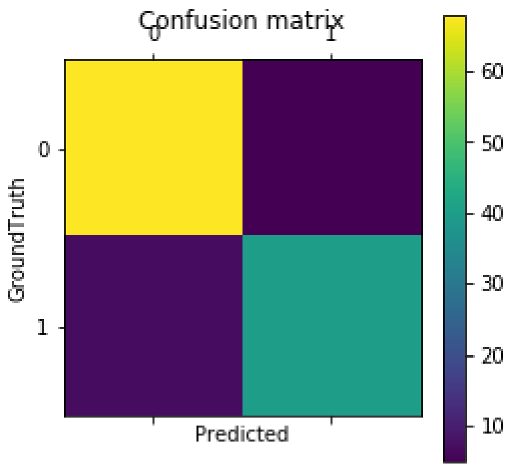
In [184]:

```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

In [185]:

```
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
print("Precision: ", metrics.precision_score(y_test, y_pred))
print("Recall: ", metrics.recall_score(y_test, y_pred))
fpr, tpr, thresholds = roc_curve (y_test, y_pred)
roc_auc= auc (fpr, tpr)
print  ("ROC AUC", roc_auc)
conftable(y_test,y_pred,"conf")
```

```
Accuracy:  0.9
Precision:  0.8888888888888888
Recall:  0.851063829787234
ROC AUC 0.8912853395511511
```




```
[[68  5]
 [ 7 40]]
```

In [ ]:

## Extra Algorithm/ Model 4 : XGBoost

In [186]:

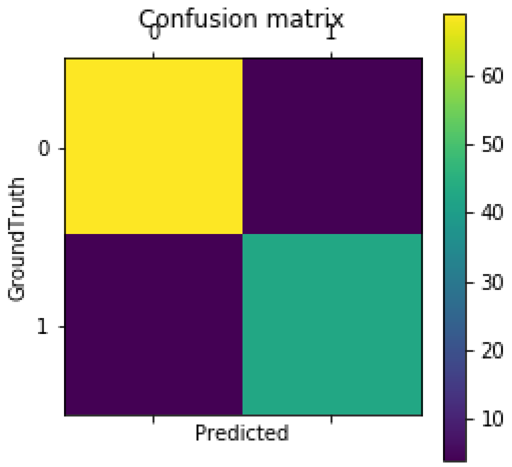
```
import xgboost as xgb
from xgboost import XGBClassifier

xgboost = XGBClassifier()
xgb = xgboost.fit( x_train, y_train)
y_pred = xgb.predict(x_test)
```

In [187]:

```
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
print("Precision: ", metrics.precision_score(y_test, y_pred))
print("Recall: ", metrics.recall_score(y_test, y_pred))
fpr, tpr, thresholds = roc_curve (y_test, y_pred)
roc_auc= auc (fpr, tpr)
print  ("ROC AUC", roc_auc)
conftable(y_test,y_pred,"conf")
```

```
Accuracy:  0.9333333333333333
Precision:  0.9148936170212766
Recall:  0.9148936170212766
ROC AUC 0.9300495482366656
```




```
[[69  4]
 [ 4 43]]
```

# Assignment

```
Extract other features and re-train the classification models to note their accuracy
metrics
```