In [1]:

```
#Importing Libraries
import pandas as pd
import numpy as np
import random as rnd

# Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Machine Learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC, SVC
from sklearn import tree
from sklearn.svm import SVC
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umat
h_tests is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [2]:

```
columns = ['age','workclass','fnlwgt','education','education-num','maritalStatus','occupation',
           'relationship','race','sex','capital-gain','capital-loss','hours-per-week','nativeCountry','Label']
data=pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data",names=columns)
```

In [3]:
```
data.head()
```
Out[3]:

	age	workclass	fnlwgt	education	education-num	maritalStatus	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	nativeCountry	l
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<

In [4]:
```
data.describe()
```
Out[4]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

In [5]:
```
type(data)
```
Out[5]:
```
pandas.core.frame.DataFrame
```

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
maritalStatus     0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
nativeCountry     0
Label             0
dtype: int64
```

In [7]:
```
#copying to the CSV file
data.to_csv(path_or_buf="data.csv",index=True)
```

In [8]:
```
# Creating Duplicate/Dta copy data frame as df
df=data.copy(deep=True)
```

In [9]:
```
#Changing the target column from categorical to numerical
```

In [10]:
```
df.loc[df['Label'].str.contains(">50K"),'target'] = 1
df.loc[df['Label'].str.contains("<=50K"),'target'] = 0
```

In [11]:
```
df.head(10)
```

2	38	Private	215646	HS-grad		Divorced	cleaners	Not-in-family	White	Male				United-States
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States
6	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica
7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45	United-States
8	31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-family	White	Female	14084	0	50	United-States
9	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	5178	0	40	United-States

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 16 columns):
age               32561 non-null int64
workclass         32561 non-null object
fnlwgt            32561 non-null int64
education         32561 non-null object
education-num     32561 non-null int64
maritalStatus     32561 non-null object
occupation        32561 non-null object
relationship      32561 non-null object
race              32561 non-null object
sex               32561 non-null object
capital-gain      32561 non-null int64
capital-loss      32561 non-null int64
hours-per-week    32561 non-null int64
nativeCountry     32561 non-null object
Label             32561 non-null object
target            32561 non-null float64
dtypes: float64(1), int64(6), object(9)
memory usage: 4.0+ MB
```

Data split

In [13]:
```
from sklearn.model_selection import train_test_split
y = df.pop('target')
X = df.iloc[:,0:-1]
```

In [14]:
```
y.shape
```

Out[14]:
```
(32561,)
```

In [15]:
```
X.shape
```

Out[15]:
```
(32561, 14)
```

In [16]:
```
type(y)
```
Out[16]:
```
pandas.core.series.Series
```

In [17]:
```
type(X)
```
Out[17]:
```
pandas.core.frame.DataFrame
```

In [18]:
```
X.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 columns):
age               32561 non-null int64
workclass         32561 non-null object
fnlwgt            32561 non-null int64
education         32561 non-null object
education-num     32561 non-null int64
maritalStatus     32561 non-null object
occupation        32561 non-null object
relationship      32561 non-null object
race              32561 non-null object
sex               32561 non-null object
capital-gain      32561 non-null int64
capital-loss      32561 non-null int64
hours-per-week    32561 non-null int64
nativeCountry     32561 non-null object
dtypes: int64(6), object(8)
memory usage: 3.5+ MB
```

In [19]:
```
X1=pd.get_dummies(X)
```

In [20]:
```
X1.shape
```
Out[20]:
```
(32561, 108)
```

In [21]:
```
# follow the usual sklearn pattern: import, instantiate, fit
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X1, y)

# print intercept and coefficients
print(lm.intercept_)
print(lm.coef_)
```

```
-0.3476853528514131
[ 2.50256242e-03  7.27984762e-08  2.80160839e-02  7.80143490e-06
  9.21109231e-05  2.88282815e-03 -2.04641327e-02  9.48900024e-02
 -6.17049517e-03  2.25084944e-02  2.33021566e-02  8.30342258e-02
 -4.12649245e-02 -2.11014016e-02 -1.34733925e-01 -1.48944955e-02
 -1.90172738e-02 -2.49235619e-02  4.77308623e-02  2.98185741e-02
 -3.11146580e-02 -2.15844205e-02 -8.44584561e-02 -5.48730419e-02
 -1.77601019e-02  9.99604448e-02 -6.50970540e-02  4.17334244e-02
  8.56760330e-02  7.47734321e-02 -4.59697069e-02 -4.91761931e-02
  1.30271832e-01  6.50124289e-02 -2.25067233e-02 -5.77063055e-02
 -3.65099252e-02 -2.93851137e-02  2.04436171e-03  2.63685480e-03
 -1.31329068e-01 -1.29494481e-02  1.30873959e-01 -1.01397613e-01
 -5.44415238e-02 -4.60133911e-02 -2.22364141e-02  1.18726342e-02
  6.93666453e-02  7.49730446e-02  4.24803276e-02  7.34881847e-02
 -3.93685531e-02  7.64661313e-02 -8.40502332e-02 -5.43030497e-02
 -5.83956713e-02 -7.03717449e-02  1.90654568e-01 -2.29106502e-02
  1.47624511e-02  4.75581915e-03 -1.70520643e-02  2.04444444e-02
 -2.92839438e-02  2.92839438e-02 -1.67659468e-02  1.67093651e-01
  4.91016459e-02 -6.68968975e-02 -7.66228332e-02  2.46898253e-02
 -2.70359409e-02 -2.46003071e-03  2.25992264e-02  6.24561831e-02
  9.87434871e-02  6.48164643e-02 -7.80335579e-02  4.81486153e-02
  1.65120208e-03 -1.43475166e-01 -5.81761872e-03 -6.96090518e-03
 -7.70511312e-03 -2.82655218e-02  2.92598599e-02  7.68084206e-02
  9.37897672e-02  1.99001778e-02  8.03265642e-02 -6.56426781e-02
 -2.12531349e-02 -5.06830337e-02 -1.32513927e-01 -3.56411307e-02
  5.40139106e-02 -9.94398869e-03 -2.15365851e-02 -6.65210444e-03
 -1.93679018e-03 -7.02737762e-02 -2.37203347e-02 -3.26140707e-02
 -2.27539641e-02  3.59026376e-02 -5.70340701e-02  8.29374822e-02]
```

In [22]:
```
#Splitting the data into test and train
X_train, X_test, y_train, y_test = train_test_split(X1, y, test_size=0.25, random_state=42)
```

In [23]:
```
def classifyWithLogisticRegression ( trainingData, results, testData ):
    clf_logreg = LogisticRegression()
    clf_logreg.fit(trainingData, results)
    return clf_logreg.predict(testData)

def classifyWithDecisionTree ( trainingData, results, testData ):
    clf_tree = tree.DecisionTreeClassifier()
    clf_tree.fit(trainingData, results)
    return clf_tree.predict(testData)

def classifyWithSVM ( trainingData, results, testData ):
    clf_svm = SVC()
    clf_svm.fit(trainingData,results)
    return clf_svm.predict(testData)

def classifyWithPerceptron ( trainingData, results, testData ):
    clf_perceptron = Perceptron()
    clf_perceptron.fit(trainingData,results)
    return clf_perceptron.predict(testData)

def classifyWithKNeighbors ( trainingData, results, testData ):
    clf_KNN = KNeighborsClassifier()
    clf_KNN.fit(trainingData,results)
    return clf_KNN.predict(testData)

def classifyWithGaussianNaiveBayes ( trainingData, results, testData ):
    clf_GaussianNB = GaussianNB()
    clf_GaussianNB.fit(trainingData,results)
    return clf_GaussianNB.predict(testData)

def classifyWithStochasticGradientDescent ( trainingData, results, testData ):
    sgd = SGDClassifier()
    sgd.fit(trainingData, results)
    return sgd.predict(testData)

def classifyWithLinearSVC ( trainingData, results, testData ):
    linear_svc = LinearSVC()
    linear_svc.fit(trainingData, results)
    return linear_svc.predict(testData)

def classifyWithRandomForest ( trainingData, results, testData ):
```

```
    random_forest = RandomForestClassifier(n_estimators=100)
    random_forest.fit(trainingData, results)
    return random_forest.predict(testData)
```

In [28]:
```
from sklearn import metrics
```

In [25]:
```
LR_prediction = classifyWithLogisticRegression(X_train, y_train, X_test)
DT_prediction = classifyWithDecisionTree(X_train, y_train, X_test)
SVM_prediction = classifyWithSVM(X_train, y_train, X_test)
KN_prediction = classifyWithKNeighbors(X_train, y_train, X_test)
LRSVC_prediction = classifyWithLinearSVC(X_train, y_train, X_test)
RF_prediction = classifyWithRandomForest(X_train, y_train, X_test)
print("Logistic regressor accuracy is",metrics.accuracy_score(y_test,LR_prediction))
print("Decision Tree regressor accuracy is",metrics.accuracy_score(y_test,DT_prediction))
print("SVM regressor accuracy is",metrics.accuracy_score(y_test,SVM_prediction))
print("KNeighbors regressor accuracy is",metrics.accuracy_score(y_test,KN_prediction))
print("LinearSVC regressor accuracy is",metrics.accuracy_score(y_test,LRSVC_prediction))
print("RandomForest regressor accuracy is",metrics.accuracy_score(y_test,RF_prediction))
```

```
Logistic regressor accuracy is 0.8018670924947795
Decision Tree regressor accuracy is 0.8232403881587028
SVM regressor accuracy is 0.7582606559390738
KNeighbors regressor accuracy is 0.7770544159194203
LinearSVC regressor accuracy is 0.7978135364205872
RandomForest regressor accuracy is 0.8580027023707161
```

RandomForest regressor accuracy is 0.8575113622405109 so it is the best one on this data set.

In [30]:

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train,y_train)
Column_Name = list(range(X_train.shape[1]))
Column_Importance_Data = pd.DataFrame({'Column_Number':Column_Name,'Importance':random_forest.feature_importances_})
Column_Importance_Data.sort_values(['Importance'],ascending=False)
```

Out[30]:

	Column_Number	Importance
1	1	0.156954
0	0	0.148559
3	3	0.093401
5	5	0.083554
33	33	0.063517
2	2	0.062776
53	53	0.043808
4	4	0.030246
35	35	0.021175
42	42	0.019066
48	48	0.015342
24	24	0.013108
54	54	0.011845
58	58	0.010866
10	10	0.010479
64	64	0.009921
65	65	0.009084
56	56	0.008907
12	12	0.008451
27	27	0.008344
26	26	0.008284

	Column_Number	Importance
41	41	0.007133
50	50	0.007070
63	63	0.006721
11	11	0.006507
105	105	0.006249
46	46	0.006133
8	8	0.006078
31	31	0.005873
7	7	0.005812
...	...	...
107	107	0.000341
102	102	0.000327
78	78	0.000322
32	32	0.000310
67	67	0.000306
18	18	0.000269
70	70	0.000261
74	74	0.000238
87	87	0.000216
98	98	0.000194
73	73	0.000187
72	72	0.000179
80	80	0.000171
83	83	0.000152
79	79	0.000141
93	93	0.000135

	Column_Number	Importance
104	104	0.000126
47	47	0.000106
84	84	0.000095
14	14	0.000082
95	95	0.000068
100	100	0.000061
91	91	0.000051
103	103	0.000039
94	94	0.000036
28	28	0.000022
9	9	0.000005
40	40	0.000005
82	82	0.000003
81	81	0.000000

108 rows × 2 columns

```
In [ ]:
```