In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
%matplotlib inline
from sklearn import svm
from sklearn import datasets
import statsmodels.api as sm
```

Importing Data test/train

In [2]:
```
# trimming all the whitespaces we use the separator ‘, ’. The test dataset has a w
```

In [3]:
```
columns_ = ['Age','Workclass','FinalWeight','Education','EduNumber','MaritalStatus
```

In [*]:
```
train_data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-database
```

In [*]:
```
test_data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases
```

Data Exploration

In [6]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
Age              32561 non-null int64
Workclass        32561 non-null object
FinalWeight      32561 non-null int64
Education        32561 non-null object
EduNumber        32561 non-null int64
MaritalStatus    32561 non-null object
Job              32561 non-null object
Family           32561 non-null object
Race             32561 non-null object
Gender           32561 non-null object
CapitalGain      32561 non-null int64
CapitalLoss      32561 non-null int64
HrsWeek          32561 non-null int64
NativeCountry    32561 non-null object
Salary           32561 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [7]:
```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16282 entries, 0 to 16281
Data columns (total 15 columns):
Age              16282 non-null object
Workclass        16281 non-null object
FinalWeight      16281 non-null float64
Education        16281 non-null object
EduNumber        16281 non-null float64
MaritalStatus    16281 non-null object
Job              16281 non-null object
Family           16281 non-null object
Race             16281 non-null object
Gender           16281 non-null object
CapitalGain      16281 non-null float64
CapitalLoss      16281 non-null float64
HrsWeek          16281 non-null float64
NativeCountry    16281 non-null object
Salary           16281 non-null object
dtypes: float64(5), object(10)
memory usage: 1.9+ MB
```

In [8]:
```
test_data.head()
```

Out[8]:

	Age	Workclass	FinalWeight	Education	EduNumber	MaritalStatus	Job	Family	Ra
0	|1x3 Cross validator	None	NaN	None	NaN	None	None	None	Nc
1	25	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Bla
2	38	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	Wh
3	28	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	Wh
4	44	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Bla

In [9]:
```
test_data.isnull().sum()
```

Out[9]:
```
Age              0
Workclass        1
FinalWeight      1
Education        1
EduNumber        1
MaritalStatus    1
Job              1
Family           1
Race             1
Gender           1
CapitalGain      1
CapitalLoss      1
HrsWeek          1
NativeCountry    1
Salary           1
dtype: int64
```

In [10]:
```
# There are 16281 samples in the test dataset
#There are both categorical and numerical columns in the dataset
#The columns workClass, occupation, native-country have missing values
```

In [11]:
```
# removing NaN from first row of test_data
test_data.drop(0,inplace=True)
test_data.reset_index(drop=True,inplace=True)
```

In [12]:
```
train_data.head()
```

Out[12]:

	Age	Workclass	FinalWeight	Education	EduNumber	MaritalStatus	Job	Family	Race
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black

In [13]:
```
train_data.isnull().sum()
```

Out[13]:
```
Age              0
Workclass        0
FinalWeight      0
Education        0
EduNumber        0
MaritalStatus    0
Job              0
Family           0
Race             0
Gender           0
CapitalGain      0
CapitalLoss      0
HrsWeek          0
NativeCountry    0
Salary           0
dtype: int64
```

#There are 32561 samples in the training dataset

#There are both categorical and numerical columns in the dataset

#The columns workClass, occupation, native-country have missing values

In [14]:
```
Categorical_cols = [i for i in train_data.columns if type(train_data[i][0])==str]
Numeric_cols = list(set(train_data.columns.tolist())- set(Categorical_cols))
train_data[Numeric_cols].head()
```

Out[14]:

	CapitalLoss	EduNumber	HrsWeek	FinalWeight	Age	CapitalGain
0	0	13	40	77516	39	2174
1	0	13	13	83311	50	0
2	0	9	40	215646	38	0
3	0	7	40	234721	53	0
4	0	13	40	338409	28	0

In [15]:
```
# Age, final weight, EducationNum, CapitalLoss and HoursPerWeek are  numerical_val
```

In [16]:
```
#Next let's dummifying Categorical data
dmCtg_train_data = pd.get_dummies(data=train_data[Categorical_cols])
dmCtg_train_data.astype('int32',copy=True)
dmCtg_test_data = pd.get_dummies(data=test_data[Categorical_cols])
dmCtg_test_data.astype('int32',copy=True)
dmCtg_train_data.tail()
```

Out[16]:

	Workclass_?	Workclass_Federal-gov	Workclass_Local-gov	Workclass_Never-worked	Workclass_Private	W
32556	0	0	0	0	1	
32557	0	0	0	0	1	
32558	0	0	0	0	1	
32559	0	0	0	0	1	
32560	0	0	0	0	0	

5 rows × 104 columns

In [17]:
```
#Let's Normalise numerical values
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler() # default=(0, 1)
Numeric_cols2 =scaler.fit_transform(train_data[Numeric_cols])
df_Numeric_cols = pd.DataFrame(Numeric_cols2,columns=train_data[Numeric_cols].colu
df_Numeric_cols.head()
```

Out[17]:

	CapitalLoss	EduNumber	HrsWeek	FinalWeight	Age	CapitalGain
0	0.0	0.800000	0.397959	0.044302	0.301370	0.02174
1	0.0	0.800000	0.122449	0.048238	0.452055	0.00000
2	0.0	0.533333	0.397959	0.138113	0.287671	0.00000
3	0.0	0.400000	0.397959	0.151068	0.493151	0.00000
4	0.0	0.800000	0.397959	0.221488	0.150685	0.00000

In [18]:
```
#Let's Join numerical and categorical values to get the normalised train_data
new_train_data= pd.concat([dmCtg_train_data,df_Numeric_cols],axis=1)
```

In [19]:
```
new_train_data.loc[new_train_data["Workclass_?"]!=1,:].shape[0]
```

Out[19]: 30725

In [20]:
```
new_train_data.loc[new_train_data["Job_?"]!=1,:].shape[0]
```

Out[20]: 30718

In [21]:
```
new_train_data.loc[new_train_data["NativeCountry_?"]!=1,:].shape[0]
```

Out[21]: 31978

In [22]:
```
new_train_data.rename(columns={"Workclass_?":"Workclass_NoInfo","NativeCountry_?":
```

In [23]:
```
# Let's split Target and INput fearuters
new_train_data.drop("Salary_<=50K",axis=1,inplace=True)
input_cols = [i for i in new_train_data.columns if i!= 'Salary_>50K']
Target_ = 'Salary_>50K'
```

In [24]:
```
#Lets impliment classicication algorith for prediction
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
```

```
C:\Users\mgirm\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.p
y:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy module a
nd should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [25]:
```
m1=DecisionTreeClassifier()
m2=GaussianNB()
m3=RandomForestClassifier()
m4=LogisticRegression()
```

In [26]:
```
#Lets Split Train and Test Data
X = new_train_data[input_cols].values
y = new_train_data[Target_].values
X_Train,X_Test,Y_train,Y_Test = train_test_split(X,y,test_size = 0.3,random_state
```

In [27]:
```
#Implimenting Decision Tree Algorith
m1.fit(X_Train,Y_train)
print(recall_score(Y_Test,m1.predict(X_Test)))
print(accuracy_score(Y_Test,m1.predict(X_Test)))
```

```
0.6386054421768708
0.8171767837035521
```

In [ ]:
```
#Which factors are importance using Decision Tree Algorith
#m1.fit(X_Train,Y_train)
X_Train.columns[np.where(m1.feature_importances_!=0)]
#print(recall_score(Y_Test,m1.predict(X_Test)))
#print(accuracy_score(Y_Test,m1.predict(X_Test)))
```

In [28]:
```
#Implimenting KNN Algorith
m2.fit(X_Train,Y_train)
print(recall_score(Y_Test,m2.predict(X_Test)))
print(accuracy_score(Y_Test,m2.predict(X_Test)))
```

```
0.9596088435374149
0.5427372300133074
```

In [29]:
```
#Implimenting Random Forest Algorith
m3.fit(X_Train,Y_train)
print(recall_score(Y_Test,m3.predict(X_Test)))
print(accuracy_score(Y_Test,m3.predict(X_Test)))
```

```
0.6016156462585034
0.8515712969597707
```

In [30]:
```
#Implimenting Ensemble Algorith
Ensemble_DF_Train = pd.DataFrame({'DT':list(m1.predict(X_Train)),'NB':list(m2.pred
X_Ensemble = Ensemble_DF_Train.iloc[:,:2].values
Y_Ensemble = Ensemble_DF_Train.iloc[:,3].values
Ensemble_DF_Test = pd.DataFrame({'DT':list(m1.predict(X_Test)),'NB':list(m2.predic
Ensemble_X_Test = Ensemble_DF_Test.iloc[:,:2].values
Ensemble_Y_Test = Ensemble_DF_Test.iloc[:,3].values
m4.fit(X_Ensemble,Y_Ensemble)
print(accuracy_score(Ensemble_Y_Test,m4.predict(Ensemble_X_Test)))
print(recall_score(Ensemble_Y_Test,m4.predict(Ensemble_X_Test)))
```

```
0.8171767837035521
0.6386054421768708
```

Optimization

In [31]:
```
#Let's add a booster
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score,fbeta_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV

clf = AdaBoostClassifier(base_estimator=DecisionTreeClassifier())
parameters={'n_estimators':[120],
              'learning_rate':[0.1],
              'base_estimator__min_samples_split' : np.arange(2, 8, 2),
              'base_estimator__max_depth' : np.arange(1, 4, 1)}

scorer = make_scorer(fbeta_score, beta=0.5)
grid_obj = GridSearchCV(clf, parameters, scoring=scorer)
grid_fit = grid_obj.fit(X_Train, Y_train)
best_clf = grid_fit.best_estimator_
predictions = (clf.fit(X_Train, Y_train)).predict(X_Test)
best_predictions = best_clf.predict(X_Test)

print("Unoptimized model\n------")
print("Accuracy score on testing data: {:.4f}".format(accuracy_score(Y_Test, predi
print("F-score on testing data: {:.4f}".format(fbeta_score(Y_Test, predictions, be
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(Y_T
print("Final F-score on the testing data: {:.4f}".format(fbeta_score(Y_Test, best_
```

```
Unoptimized model
------
Accuracy score on testing data: 0.8341
F-score on testing data: 0.6560

Optimized Model
------
Final accuracy score on the testing data: 0.8748
Final F-score on the testing data: 0.7606
```

Problem 3: Which algorithms are best for this datase

AnswertoQuestion#3 AdaBoostClassifier done on top of ensemble method gave improved accuracy score of 0.834 hence the reason it is the base algorithm.

In [ ]:
```
```