# Project3 classifier model

February 19, 2019

# 1 PROJECT -3 APPLICATION OF CLASSIFICATION MODELS

## 1.1 About Data Set

This data was extracted from the census bureau database found at http://www.census.gov/ftp/pub/DES/www/welcome.html Donor: Ronny Kohavi and Barry Becker, Data Mining and Visualization Silicon Graphics. e-mail: ronnyk@sgi.com for questions. Split into train-test using MLC++ GenCVFiles (2/3, 1/3 random). 48842 instances, mix of continuous and discrete (train=32561, test=16281) 45222 if instances with unknown values are removed (train=30162, test=15060) Duplicate or conflicting instances : 6 Class probabilities for adult.all file Probability for the label '>50K' : 23.93% / 24.78% (without unknowns) Probability for the label '<=50K' : 76.07% / 75.22% (without unknowns) Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0)) Prediction task is to determine whether a person makes over 50K a year. Conversion of original data as follows: 1. Discretized a gross income into two ranges with threshold 50,000. 2. Convert U.S. to US to avoid periods. 3. Convert Unknown to "?" 4. Run MLC++ GenCVFiles to generate data,test. Description of fnlwgt (final weight) The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are: 1. A single cell estimate of the population 16+ for each state. 2. Controls for Hispanic Origin by age and sex. 3. Controls by Race, age and sex. We use all three sets of controls in our weighting program and "rake" through them 6 times so that by the end we come back to all the controls we used. The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population. People with similar demographic characteristics should have similar weights. There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state. Dataset Link https://archive.ics.uci.edu/ml/machine-learning-databases/adult/

# 2 Load libraries

```
In [0]: # Core Libraries - Data manipulation and analysis
        import pandas as pd
        import numpy as np
        import math
```

```
from math import sqrt
import matplotlib.pyplot as plt
import seaborn as sns
import csv
import urllib.request

# Core Libraries - Machine Learning
import sklearn
import xgboost as xgb


# Importing Classifiers - Modelling
from sklearn.linear_model import LogisticRegression
from xgboost.sklearn import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

## Importing train_test_split,cross_val_score,GridSearchCV,KFold - Validation and Opti
from sklearn.model_selection import  train_test_split, cross_val_score, GridSearchCV, K


# Importing Metrics - Performance Evaluation
from sklearn import metrics

# Warnings Library - Ignore warnings
import warnings
warnings.filterwarnings('ignore')

import pickle
```

# 3 Load Data

```
In [0]: train_set = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adul
        test_set =  pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adul
                        header = None)
        col_labels = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_sta
                          'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'na
        train_set.columns = col_labels
        test_set.columns = col_labels
```

# 4 Understand the Dataset and Data

```
In [12]: train_set.shape,test_set.shape

Out[12]: ((32561, 15), (16281, 15))
```

In [13]: train_set.columns

Out[13]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'wage_class'],
      dtype='object')
```

In [14]: train_set.head()

Out[14]:
```
   age          workclass  fnlwgt   education  education_num  \
0   39          State-gov   77516   Bachelors             13
1   50   Self-emp-not-inc   83311   Bachelors             13
2   38            Private  215646     HS-grad              9
3   53            Private  234721        11th              7
4   28            Private  338409   Bachelors             13

        marital_status          occupation    relationship   race      sex  \
0        Never-married        Adm-clerical   Not-in-family  White     Male
1   Married-civ-spouse     Exec-managerial         Husband  White     Male
2             Divorced   Handlers-cleaners   Not-in-family  White     Male
3   Married-civ-spouse   Handlers-cleaners         Husband  Black     Male
4   Married-civ-spouse      Prof-specialty            Wife  Black   Female

   capital_gain  capital_loss  hours_per_week  native_country wage_class
0          2174             0              40   United-States      <=50K
1             0             0              13   United-States      <=50K
2             0             0              40   United-States      <=50K
3             0             0              40   United-States      <=50K
4             0             0              40            Cuba      <=50K
```

In [15]: test_set.columns

Out[15]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'wage_class'],
      dtype='object')
```

In [16]: test_set.head()

Out[16]:
```
   age   workclass  fnlwgt      education  education_num       marital_status  \
0   25     Private  226802           11th              7        Never-married
1   38     Private   89814        HS-grad              9   Married-civ-spouse
2   28   Local-gov  336951     Assoc-acdm             12   Married-civ-spouse
3   44     Private  160323   Some-college             10   Married-civ-spouse
4   18           ?  103497   Some-college             10        Never-married

            occupation relationship    race      sex  capital_gain  \
```

```
0   Machine-op-inspct    Own-child   Black     Male             0
1     Farming-fishing      Husband   White     Male             0
2     Protective-serv      Husband   White     Male             0
3   Machine-op-inspct      Husband   Black     Male          7688
4                   ?    Own-child   White   Female             0

   capital_loss  hours_per_week  native_country wage_class
0             0              40   United-States     <=50K.
1             0              50   United-States     <=50K.
2             0              40   United-States      >50K.
3             0              40   United-States      >50K.
4             0              30   United-States     <=50K.
```

In [17]: train_set.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
fnlwgt            32561 non-null int64
education         32561 non-null object
education_num     32561 non-null int64
marital_status    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
native_country    32561 non-null object
wage_class        32561 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [18]: test_set.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16281 entries, 0 to 16280
Data columns (total 15 columns):
age               16281 non-null int64
workclass         16281 non-null object
fnlwgt            16281 non-null int64
education         16281 non-null object
education_num     16281 non-null int64
marital_status    16281 non-null object
occupation        16281 non-null object
```

```
relationship      16281 non-null object
race              16281 non-null object
sex               16281 non-null object
capital_gain      16281 non-null int64
capital_loss      16281 non-null int64
hours_per_week    16281 non-null int64
native_country    16281 non-null object
wage_class        16281 non-null object
dtypes: int64(6), object(9)
memory usage: 1.9+ MB
```

In [19]:
```
train_set.get_dtype_counts()
```

Out[19]:
```
int64     6
object    9
dtype: int64
```

In [20]:
```
test_set.get_dtype_counts()
```

Out[20]:
```
int64     6
object    9
dtype: int64
```

# 5 Clean the data

## 5.1 Clean Column Names

In [21]:
```
train_set.columns
```

Out[21]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'wage_class'],
      dtype='object')
```

In [22]:
```
test_set.columns
```

Out[22]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'wage_class'],
      dtype='object')
```

***The columns don't have any nonsensical values, therefore there is no need to clean or change column names***

## 5.2 Clean Numerical Columns

### 5.2.1 Null values

```
In [0]: num_cols = train_set.select_dtypes(include="int64").columns.values
        # num_cols = test_set.select_dtypes(include="int64").columns.values can also be used b
```

```
In [24]: train_set[num_cols].isna().sum()
```

```
Out[24]: age               0
         fnlwgt            0
         education_num     0
         capital_gain      0
         capital_loss      0
         hours_per_week    0
         dtype: int64
```

```
In [25]: test_set[num_cols].isna().sum()
```

```
Out[25]: age               0
         fnlwgt            0
         education_num     0
         capital_gain      0
         capital_loss      0
         hours_per_week    0
         dtype: int64
```

*No null values in the numerical columns of both the train_set and test_set*

### 5.2.2 Zeros

**Check if there are any rows with all row values = zero that need our consideration so that we can decide to study those rows**

```
In [26]: train_set.loc[(train_set==0).all(axis=1),num_cols].shape
```

```
Out[26]: (0, 6)
```

```
In [27]: test_set.loc[(train_set==0).all(axis=1),num_cols].shape
```

```
Out[27]: (0, 6)
```

*There are no rows which have all row values == 0*

**Check if there are any rows with any row values = zero that need our consideration so that we can decide to study those rows**

```
In [28]: train_set.loc[(train_set==0).any(axis=1),num_cols].shape
```

```
Out[28]: (32561, 6)
```

```
In [29]: train_set.loc[(train_set==0).any(axis=1),num_cols].head()
```

```
Out[29]:    age  fnlwgt  education_num  capital_gain  capital_loss  hours_per_week
         0   39   77516             13          2174             0              40
         1   50   83311             13             0             0              13
         2   38  215646              9             0             0              40
         3   53  234721              7             0             0              40
         4   28  338409             13             0             0              40
```

```
In [30]: train_set.loc[(train_set.drop(["capital_gain", "capital_loss"],axis=1)==0).any(axis=1
```

```
Out[30]: (0, 6)
```

```
In [31]: test_set.loc[(train_set==0).any(axis=1),num_cols].shape
```

```
Out[31]: (16281, 6)
```

```
In [32]: test_set.loc[(test_set.drop(["capital_gain", "capital_loss"],axis=1)==0).any(axis=1),n
```

```
Out[32]: (0, 6)
```

***There are no rows which have any row values == 0, except in captital_gain, capital_loss columns(where 0 is a valid value)***

#### 5.2.3 Nonsensical values

***There are no nonsensical values in the Numerical Columns***

### 5.3 Clean Categorical Columns

#### 5.3.1 Null values

```
In [33]: cat_cols = train_set.select_dtypes(include="object").columns.values
         cat_cols
```

```
Out[33]: array(['workclass', 'education', 'marital_status', 'occupation',
                'relationship', 'race', 'sex', 'native_country', 'wage_class'],
               dtype=object)
```

```
In [34]: train_set[cat_cols].isna().sum()
```

```
Out[34]: workclass         0
         education         0
         marital_status    0
         occupation        0
         relationship      0
         race              0
         sex               0
         native_country    0
         wage_class        0
         dtype: int64
```

```
In [35]: test_set[cat_cols].isna().sum()
```

```
Out[35]: workclass         0
         education         0
         marital_status    0
         occupation        0
         relationship      0
         race              0
         sex               0
         native_country    0
         wage_class        0
         dtype: int64
```

### 5.3.2 Empty Values

```
In [36]: train_set.loc[(train_set=="").any(axis=1),cat_cols].shape
```

```
Out[36]: (0, 9)
```

```
In [37]: test_set.loc[(train_set=="").any(axis=1),cat_cols].shape
```

```
Out[37]: (0, 9)
```

***There are no empty strings in any of the rows***

### 5.3.3 Nonsensical values

```
In [38]: train_set[cat_cols].nunique()
```

```
Out[38]: workclass          9
         education         16
         marital_status     7
         occupation        15
         relationship       6
         race               5
         sex                2
         native_country    42
         wage_class         2
         dtype: int64
```

```
In [39]: for col in cat_cols:
             print(train_set[col].unique(),"\n")
```

```
[' State-gov' ' Self-emp-not-inc' ' Private' ' Federal-gov' ' Local-gov'
 ' ?' ' Self-emp-inc' ' Without-pay' ' Never-worked']

[' Bachelors' ' HS-grad' ' 11th' ' Masters' ' 9th' ' Some-college'
 ' Assoc-acdm' ' Assoc-voc' ' 7th-8th' ' Doctorate' ' Prof-school'
 ' 5th-6th' ' 10th' ' 1st-4th' ' Preschool' ' 12th']

[' Never-married' ' Married-civ-spouse' ' Divorced'
```

```
 ' Married-spouse-absent' ' Separated' ' Married-AF-spouse' ' Widowed']

[' Adm-clerical' ' Exec-managerial' ' Handlers-cleaners' ' Prof-specialty'
 ' Other-service' ' Sales' ' Craft-repair' ' Transport-moving'
 ' Farming-fishing' ' Machine-op-inspct' ' Tech-support' ' ?'
 ' Protective-serv' ' Armed-Forces' ' Priv-house-serv']

[' Not-in-family' ' Husband' ' Wife' ' Own-child' ' Unmarried'
 ' Other-relative']

[' White' ' Black' ' Asian-Pac-Islander' ' Amer-Indian-Eskimo' ' Other']

[' Male' ' Female']

[' United-States' ' Cuba' ' Jamaica' ' India' ' ?' ' Mexico' ' South'
 ' Puerto-Rico' ' Honduras' ' England' ' Canada' ' Germany' ' Iran'
 ' Philippines' ' Italy' ' Poland' ' Columbia' ' Cambodia' ' Thailand'
 ' Ecuador' ' Laos' ' Taiwan' ' Haiti' ' Portugal' ' Dominican-Republic'
 ' El-Salvador' ' France' ' Guatemala' ' China' ' Japan' ' Yugoslavia'
 ' Peru' ' Outlying-US(Guam-USVI-etc)' ' Scotland' ' Trinadad&Tobago'
 ' Greece' ' Nicaragua' ' Vietnam' ' Hong' ' Ireland' ' Hungary'
 ' Holand-Netherlands']

[' <=50K' ' >50K']
```

***The columns workclass, occupation and native_country have rows that have garbage values which need to be imputed or dropped in the train_set***

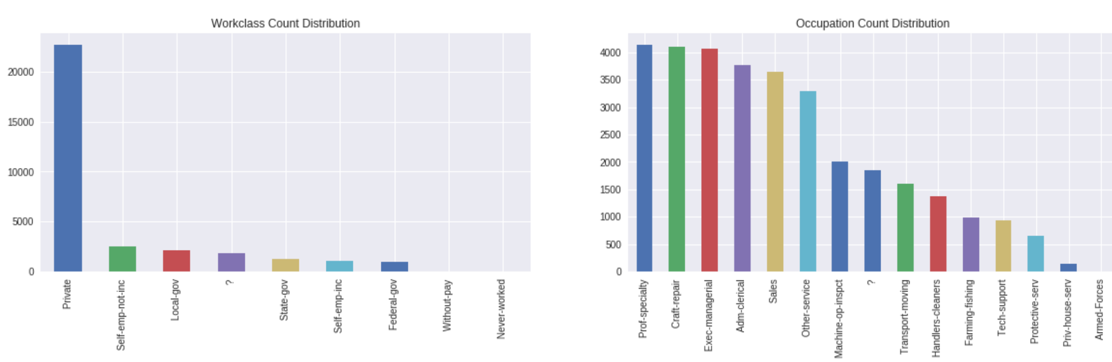
```
In [40]: test_set['workclass'].unique()

Out[40]: array([' Private', ' Local-gov', ' ?', ' Self-emp-not-inc',
                ' Federal-gov', ' State-gov', ' Self-emp-inc', ' Without-pay',
                ' Never-worked'], dtype=object)

In [41]: for col in cat_cols:
             print(test_set[col].unique(),"\n")

[' Private' ' Local-gov' ' ?' ' Self-emp-not-inc' ' Federal-gov'
 ' State-gov' ' Self-emp-inc' ' Without-pay' ' Never-worked']

[' 11th' ' HS-grad' ' Assoc-acdm' ' Some-college' ' 10th' ' Prof-school'
 ' 7th-8th' ' Bachelors' ' Masters' ' Doctorate' ' 5th-6th' ' Assoc-voc'
 ' 9th' ' 12th' ' 1st-4th' ' Preschool']

[' Never-married' ' Married-civ-spouse' ' Widowed' ' Divorced'
 ' Separated' ' Married-spouse-absent' ' Married-AF-spouse']
```

```
[' Machine-op-inspct' ' Farming-fishing' ' Protective-serv' ' ?'
 ' Other-service' ' Prof-specialty' ' Craft-repair' ' Adm-clerical'
 ' Exec-managerial' ' Tech-support' ' Sales' ' Priv-house-serv'
 ' Transport-moving' ' Handlers-cleaners' ' Armed-Forces']

[' Own-child' ' Husband' ' Not-in-family' ' Unmarried' ' Wife'
 ' Other-relative']

[' Black' ' White' ' Asian-Pac-Islander' ' Other' ' Amer-Indian-Eskimo']

[' Male' ' Female']

[' United-States' ' ?' ' Peru' ' Guatemala' ' Mexico'
 ' Dominican-Republic' ' Ireland' ' Germany' ' Philippines' ' Thailand'
 ' Haiti' ' El-Salvador' ' Puerto-Rico' ' Vietnam' ' South' ' Columbia'
 ' Japan' ' India' ' Cambodia' ' Poland' ' Laos' ' England' ' Cuba'
 ' Taiwan' ' Italy' ' Canada' ' Portugal' ' China' ' Nicaragua'
 ' Honduras' ' Iran' ' Scotland' ' Jamaica' ' Ecuador' ' Yugoslavia'
 ' Hungary' ' Hong' ' Greece' ' Trinadad&Tobago'
 ' Outlying-US(Guam-USVI-etc)' ' France']

[' <=50K.' ' >50K.']
```

*The columns workclass, occupation and native_country have rows that have garbage values which need to be imputed or dropped in the test_set*

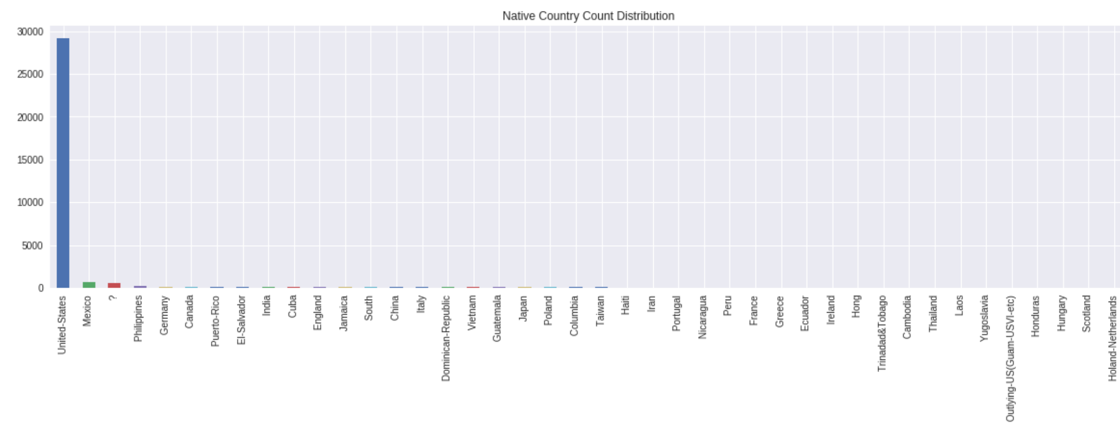
In [42]:
```python
plt.figure(figsize=(20,10))
plt.subplot(2,2,1)
plt.title("Workclass Count Distribution")
train_set['workclass'].value_counts().plot.bar()

plt.subplot(2,2,2)
plt.title("Occupation Count Distribution")
train_set['occupation'].value_counts().plot.bar()
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f901252cf60>
```



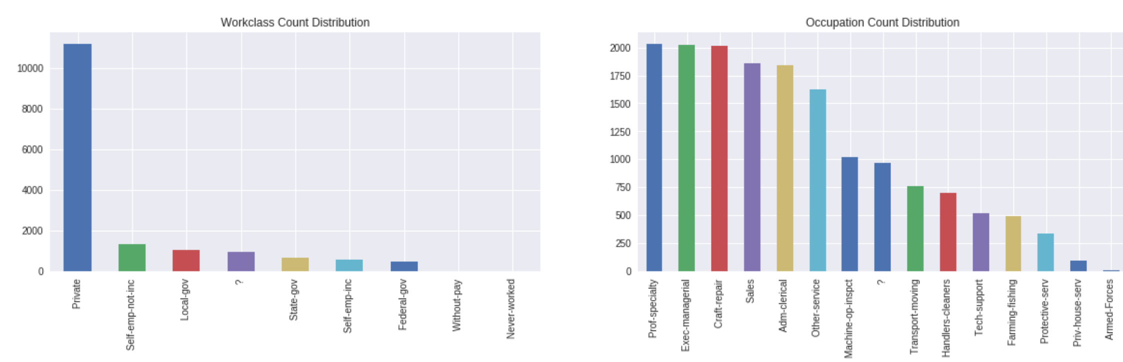
```
In [43]: plt.figure(figsize=(20,5))
         plt.subplot(1,1,1)
         plt.title("Native Country Count Distribution")
         train_set['native_country'].value_counts().plot.bar()
```

```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7f901252c278>
```



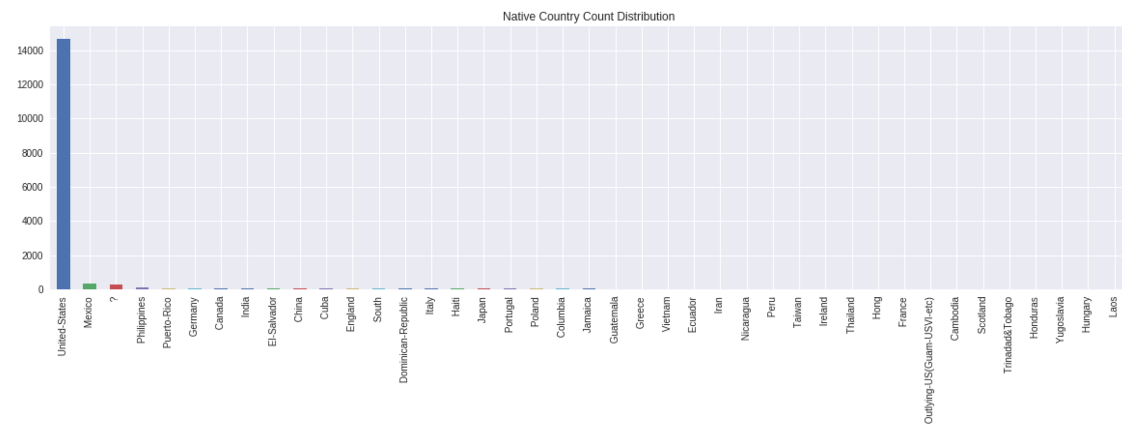

```
In [44]: plt.figure(figsize=(20,10))
         plt.subplot(2,2,1)
         plt.title("Workclass Count Distribution")
         test_set['workclass'].value_counts().plot.bar()

         plt.subplot(2,2,2)
         plt.title("Occupation Count Distribution")
         test_set['occupation'].value_counts().plot.bar()
```

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x7f900fb32a58>
```



In [45]:
```python
plt.figure(figsize=(20,5))
plt.subplot(1,1,1)
plt.title("Native Country Count Distribution")
test_set['native_country'].value_counts().plot.bar()
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f900fbab240>
```




In [46]:
```python
train_set[train_set.workclass.str.contains("\?")].head()
```

Out[46]:
```
     age workclass  fnlwgt     education  education_num  \
27    54         ?  180211  Some-college             10
61    32         ?  293936       7th-8th              4
69    25         ?  200681  Some-college             10
77    67         ?  212759          10th              6
106   17         ?  304873          10th              6

            marital_status occupation   relationship                race  \
27      Married-civ-spouse          ?        Husband  Asian-Pac-Islander
61   Married-spouse-absent          ?  Not-in-family               White
69           Never-married          ?      Own-child               White
77      Married-civ-spouse          ?        Husband               White
106          Never-married          ?      Own-child               White

        sex  capital_gain  capital_loss  hours_per_week  native_country  \
27     Male             0             0              60           South
61     Male             0             0              40               ?
69     Male             0             0              40   United-States
77     Male             0             0               2   United-States
106  Female         34095             0              32   United-States

    wage_class
27        >50K
61       <=50K
```

```
69        <=50K
77        <=50K
106       <=50K
```

In [47]:
```
test_set[test_set.workclass.str.contains("\?")].head()
```

Out[47]:
```
    age workclass  fnlwgt     education  education_num      marital_status  \
4    18         ?  103497  Some-college             10       Never-married
6    29         ?  227026       HS-grad              9       Never-married
13   58         ?  299831       HS-grad              9  Married-civ-spouse
22   72         ?  132015       7th-8th              4            Divorced
35   65         ?  191846       HS-grad              9  Married-civ-spouse

   occupation   relationship   race     sex  capital_gain  capital_loss  \
4           ?      Own-child  White  Female             0             0
6           ?      Unmarried  Black    Male             0             0
13          ?        Husband  White    Male             0             0
22          ?  Not-in-family  White  Female             0             0
35          ?        Husband  White    Male             0             0

    hours_per_week  native_country wage_class
4               30   United-States     <=50K.
6               40   United-States     <=50K.
13              35   United-States     <=50K.
22               6   United-States     <=50K.
35              40   United-States     <=50K.
```

In [48]:
```
(train_set.loc[(train_set==" ?").any(axis=1),cat_cols].shape[0]/train_set.shape[0])*10
```

Out[48]: 7.367709836921471

In [49]:
```
(test_set.loc[(test_set==" ?").any(axis=1),cat_cols].shape[0]/test_set.shape[0])*100
```

Out[49]: 7.499539340335361

***If we drop the rows containing ? values, we incur a data loss of approximately 7.5% data loss in the train_set and the test_set. Therefore we choose to drop it***

In [50]:
```
train_set.drop(train_set.loc[(train_set==" ?").any(axis=1)].index, inplace= True)
train_set.shape[0]
```

Out[50]: 30162

In [51]:
```
test_set.drop(test_set.loc[(test_set==" ?").any(axis=1)].index, inplace= True)
test_set.shape[0]
```

Out[51]: 15060

In [52]:
```
test_set.loc[(test_set==" ?").any(axis=1),cat_cols].shape[0]/test_set.shape[0]
```

Out[52]: 0.0

# 6 Get Basic Statistical Information

In [53]:
```
train_set.describe()
```

Out[53]:

| | age | fnlwgt | education_num | capital_gain | capital_loss |
|---|---|---|---|---|---|
| count | 30162.000000 | 3.016200e+04 | 30162.000000 | 30162.000000 | 30162.000000 |
| mean | 38.437902 | 1.897938e+05 | 10.121312 | 1092.007858 | 88.372489 |
| std | 13.134665 | 1.056530e+05 | 2.549995 | 7406.346497 | 404.298370 |
| min | 17.000000 | 1.376900e+04 | 1.000000 | 0.000000 | 0.000000 |
| 25% | 28.000000 | 1.176272e+05 | 9.000000 | 0.000000 | 0.000000 |
| 50% | 37.000000 | 1.784250e+05 | 10.000000 | 0.000000 | 0.000000 |
| 75% | 47.000000 | 2.376285e+05 | 13.000000 | 0.000000 | 0.000000 |
| max | 90.000000 | 1.484705e+06 | 16.000000 | 99999.000000 | 4356.000000 |

| | hours_per_week |
|---|---|
| count | 30162.000000 |
| mean | 40.931238 |
| std | 11.979984 |
| min | 1.000000 |
| 25% | 40.000000 |
| 50% | 40.000000 |
| 75% | 45.000000 |
| max | 99.000000 |

In [54]:
```
train_set.describe(include='object')
```

Out[54]:

| | workclass | education | marital_status | occupation | relationship |
|---|---|---|---|---|---|
| count | 30162 | 30162 | 30162 | 30162 | 30162 |
| unique | 7 | 16 | 7 | 14 | 6 |
| top | Private | HS-grad | Married-civ-spouse | Prof-specialty | Husband |
| freq | 22286 | 9840 | 14065 | 4038 | 12463 |

| | race | sex | native_country | wage_class |
|---|---|---|---|---|
| count | 30162 | 30162 | 30162 | 30162 |
| unique | 5 | 2 | 41 | 2 |
| top | White | Male | United-States | <=50K |
| freq | 25933 | 20380 | 27504 | 22654 |

In [55]:
```
test_set.describe()
```

Out[55]:

| | age | fnlwgt | education_num | capital_gain | capital_loss |
|---|---|---|---|---|---|
| count | 15060.000000 | 1.506000e+04 | 15060.000000 | 15060.000000 | 15060.000000 |
| mean | 38.768327 | 1.896164e+05 | 10.112749 | 1120.301594 | 89.041899 |
| std | 13.380676 | 1.056150e+05 | 2.558727 | 7703.181842 | 406.283245 |
| min | 17.000000 | 1.349200e+04 | 1.000000 | 0.000000 | 0.000000 |
| 25% | 28.000000 | 1.166550e+05 | 9.000000 | 0.000000 | 0.000000 |
| 50% | 37.000000 | 1.779550e+05 | 10.000000 | 0.000000 | 0.000000 |
| 75% | 48.000000 | 2.385888e+05 | 13.000000 | 0.000000 | 0.000000 |
| max | 90.000000 | 1.490400e+06 | 16.000000 | 99999.000000 | 3770.000000 |

```
       hours_per_week
count    15060.000000
mean        40.951594
std         12.062831
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
```

In [56]:
```
test_set.describe(include='object')
```

Out[56]:
```
       workclass education       marital_status        occupation  \
count      15060     15060                15060             15060
unique         7        16                    7                14
top      Private   HS-grad   Married-civ-spouse   Exec-managerial
freq       11021      4943                 6990              1992

       relationship    race     sex  native_country wage_class
count         15060   15060   15060           15060      15060
unique            6       5       2              40          2
top         Husband   White    Male   United-States     <=50K.
freq           6203   12970   10147           13788      11360
```

In [57]:
```
train_set.corr()
```

Out[57]:
```
                     age    fnlwgt  education_num  capital_gain  capital_loss  \
age             1.000000 -0.076511       0.043526      0.080154      0.060165
fnlwgt         -0.076511  1.000000      -0.044992      0.000422     -0.009750
education_num   0.043526 -0.044992       1.000000      0.124416      0.079646
capital_gain    0.080154  0.000422       0.124416      1.000000     -0.032229
capital_loss    0.060165 -0.009750       0.079646     -0.032229      1.000000
hours_per_week  0.101599 -0.022886       0.152522      0.080432      0.052417

                hours_per_week
age                   0.101599
fnlwgt               -0.022886
education_num         0.152522
capital_gain          0.080432
capital_loss          0.052417
hours_per_week        1.000000
```

In [58]:
```
test_set.corr()
```

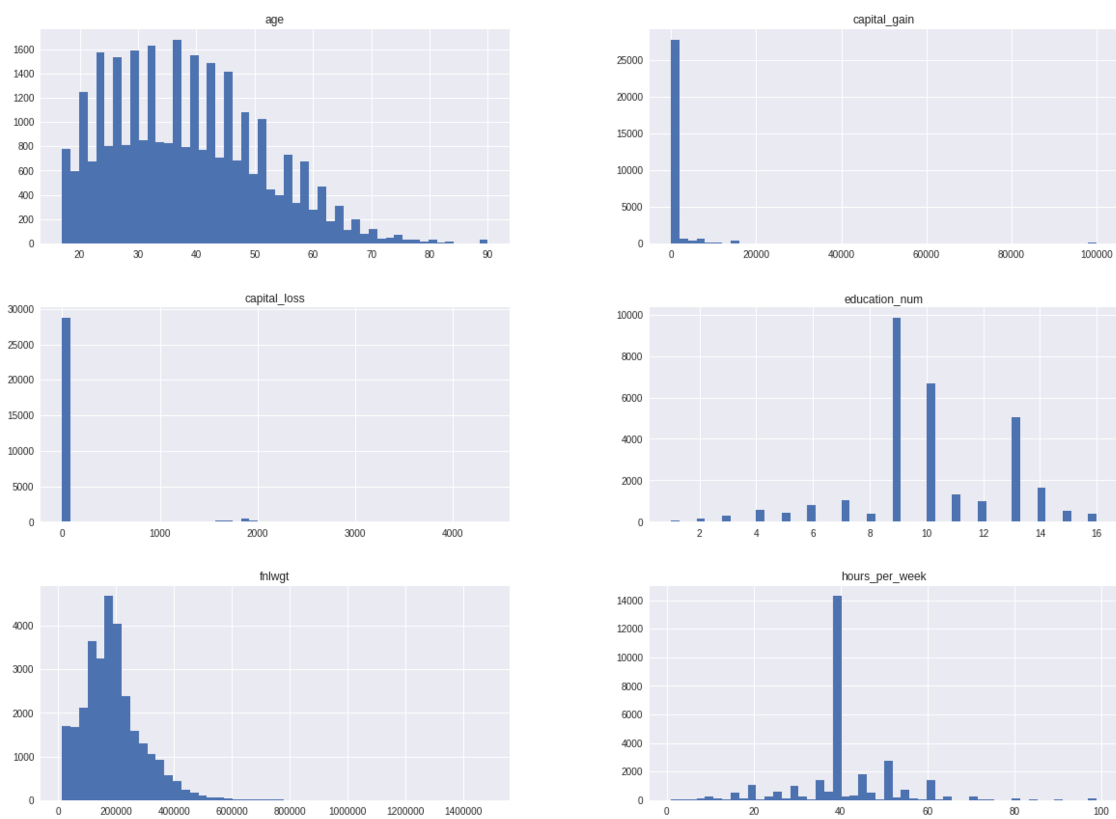
Out[58]:
```
                     age    fnlwgt  education_num  capital_gain  capital_loss  \
age             1.000000 -0.074375       0.026123      0.078760      0.057745
fnlwgt         -0.074375  1.000000      -0.036010     -0.012839      0.006421
education_num   0.026123 -0.036010       1.000000      0.131750      0.085817
```

```
capital_gain    0.078760 -0.012839        0.131750      1.000000      -0.031876
capital_loss    0.057745  0.006421        0.085817     -0.031876       1.000000
hours_per_week  0.102758 -0.010306        0.133691      0.090501       0.057712

                hours_per_week
age                   0.102758
fnlwgt               -0.010306
education_num         0.133691
capital_gain          0.090501
capital_loss          0.057712
hours_per_week        1.000000
```

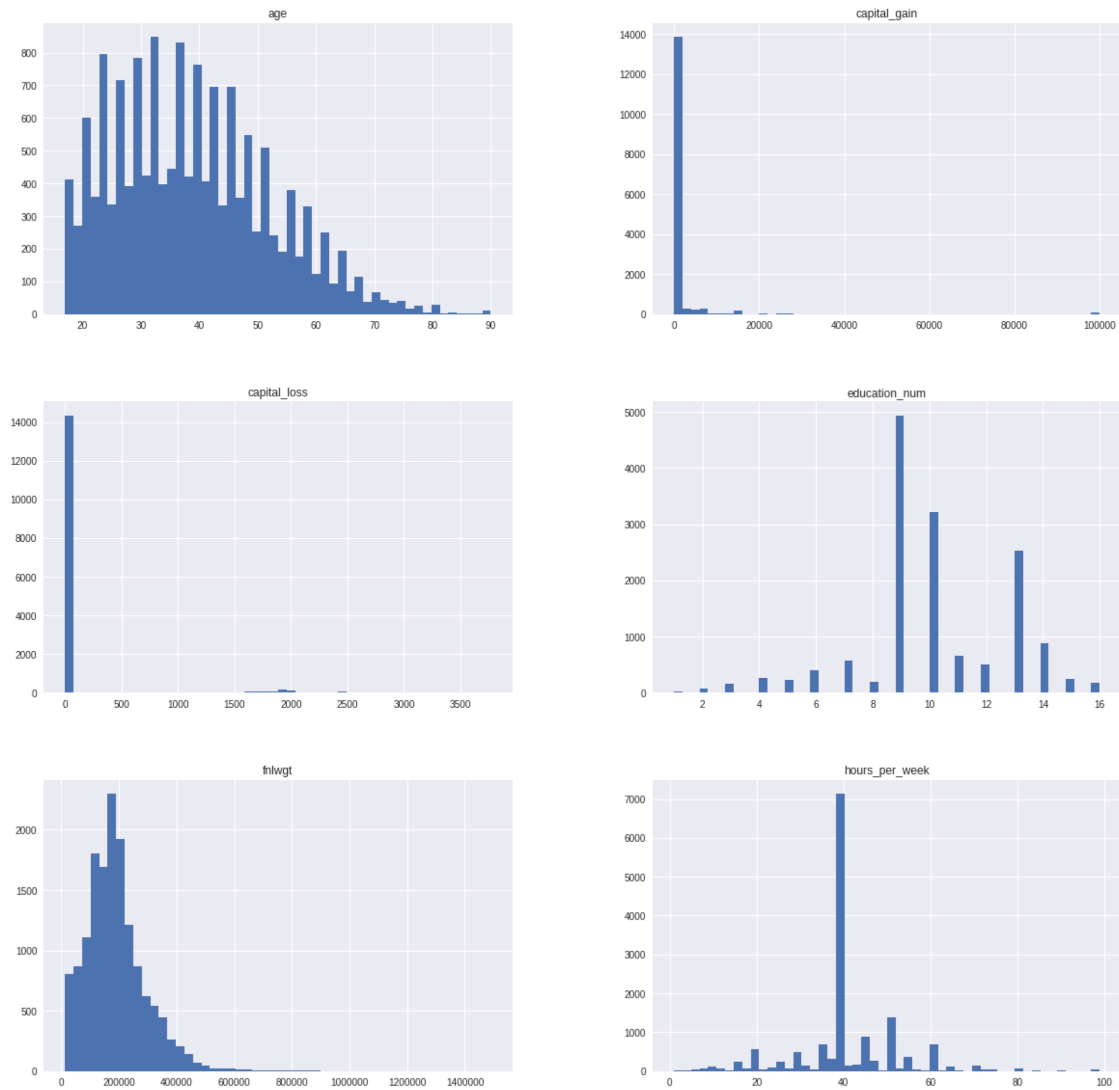
# 7 Explore Data

## 7.1 Uni-variate

In [59]:
```python
train_set[num_cols].hist(bins=50, figsize=(20,20), layout=(4,2))
plt.show()
```

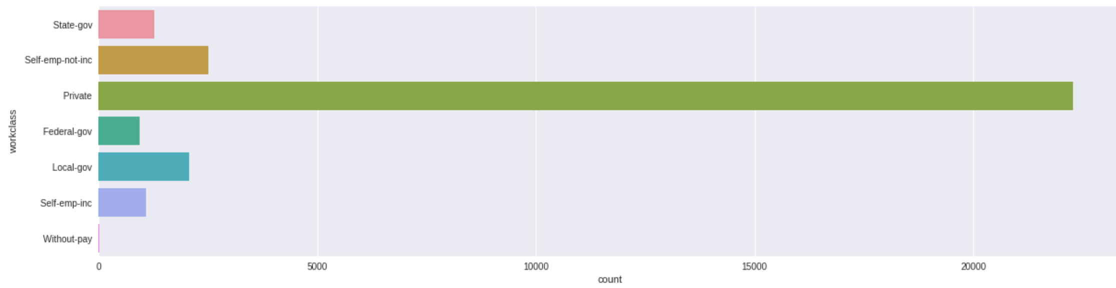


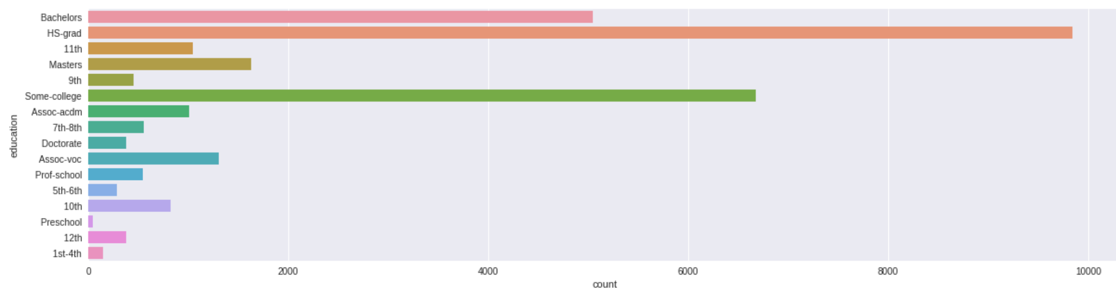

In [60]:
```python
test_set[num_cols].hist(bins=50, figsize=(20,20), layout=(3,2))
plt.show()
```

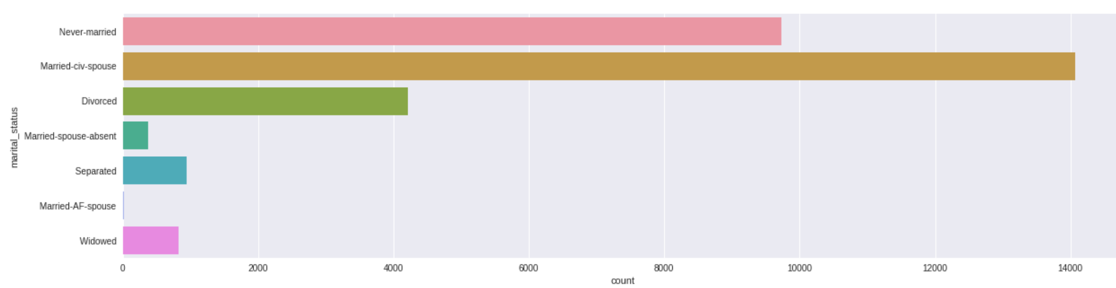


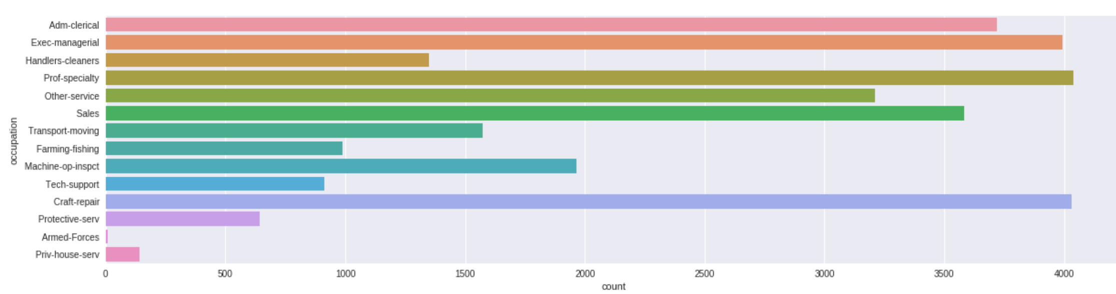

### 7.1.1 Categorical Columns

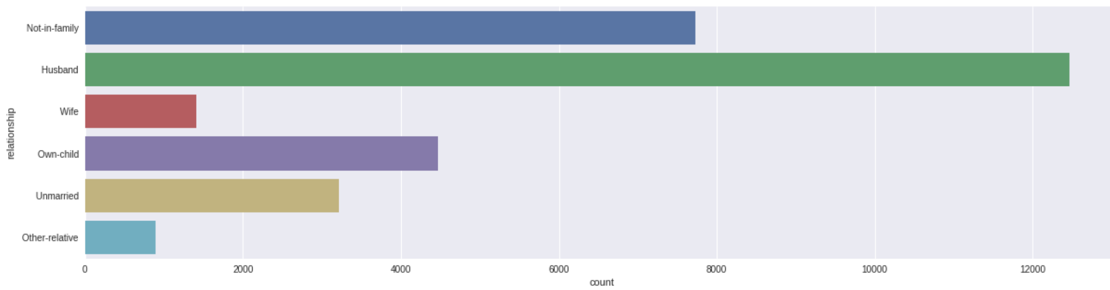
```
In [61]: for i, col in enumerate(cat_cols):
             if(col!='native_country'):
                 plt.figure(i,figsize = (20,5))
                 sns.countplot(y=col, data=train_set,)
             else:
                 plt.figure(i,figsize = (20,10))
                 sns.countplot(y=col, data=train_set)
```

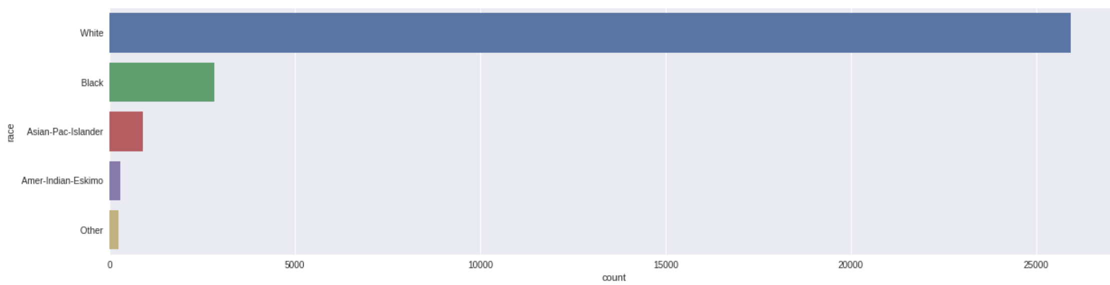

State-gov
Self-emp-not-inc
Private
Federal-gov
Local-gov
Self-emp-inc
Without-pay
workclass
0
5000
10000
15000
20000
count

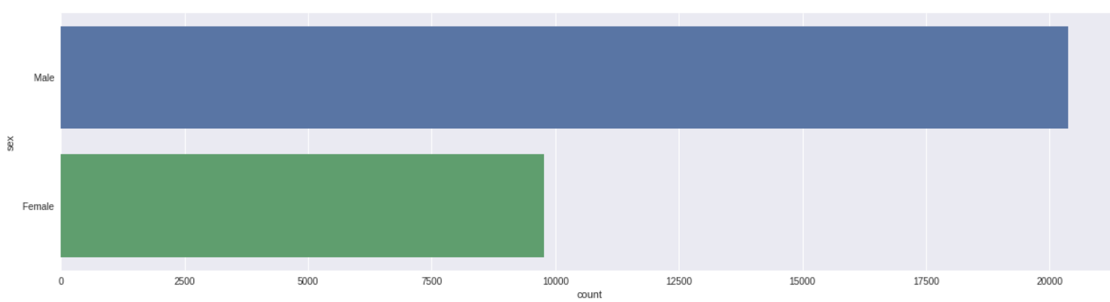

Bachelors
HS-grad
11th
Masters
9th
Some-college
Assoc-acdm
7th-8th
Doctorate
Assoc-voc
Prof-school
5th-6th
10th
Preschool
12th
1st-4th
education
0
2000
4000
6000
8000
10000
count

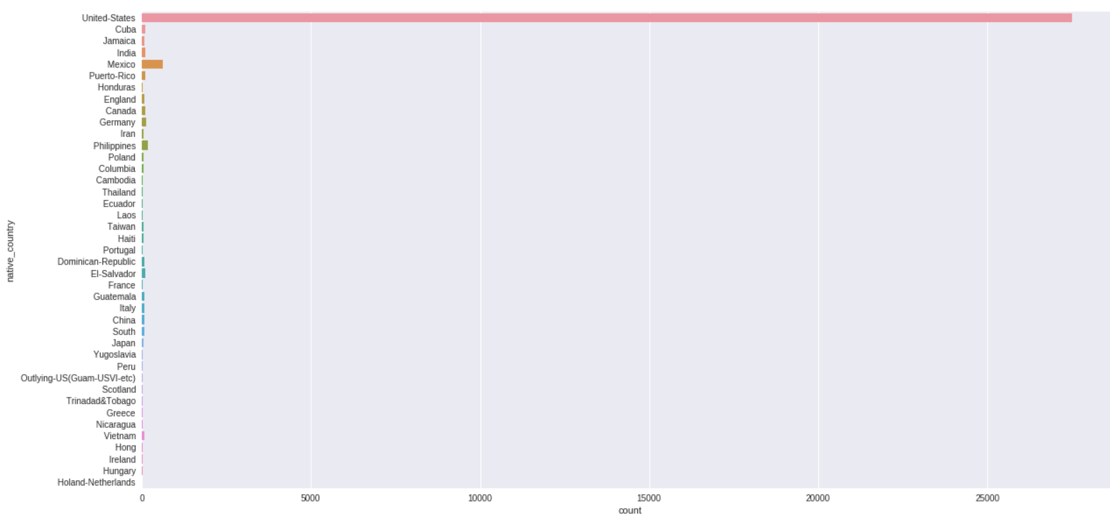

Never-married
Married-civ-spouse
Divorced
Married-spouse-absent
Separated
Married-AF-spouse
Widowed
marital_status
0
2000
4000
6000
8000
10000
12000
14000
count

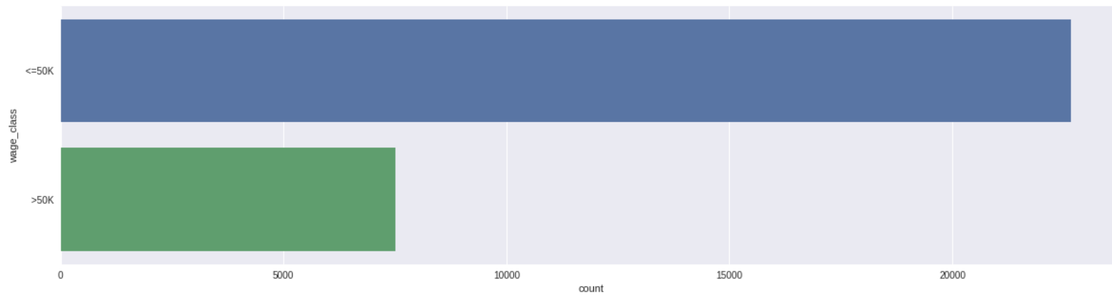

Adm-clerical
Exec-managerial
Handlers-cleaners
Prof-specialty
Other-service
Sales
Transport-moving
Farming-fishing
Machine-op-inspct
Tech-support
Craft-repair
Protective-serv
Armed-Forces
Priv-house-serv
occupation
0
500
1000
1500
2000
2500
3000
3500
4000
count


relationship
Not-in-family
Husband
Wife
Own-child
Unmarried
Other-relative
0
2000
4000
6000
8000
10000
12000
count

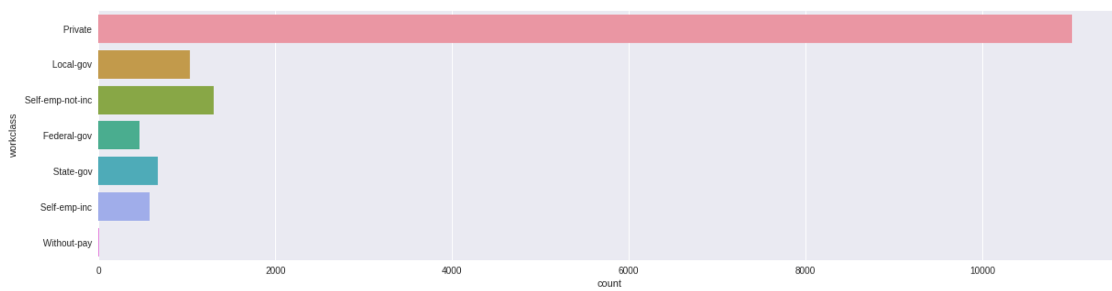

race
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
0
5000
10000
15000
20000
25000
count

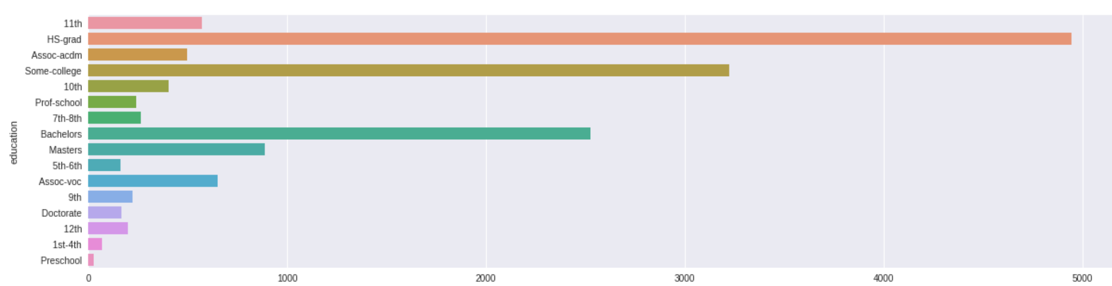

sex
Male
Female
0
2500
5000
7500
10000
12500
15000
17500
20000
count

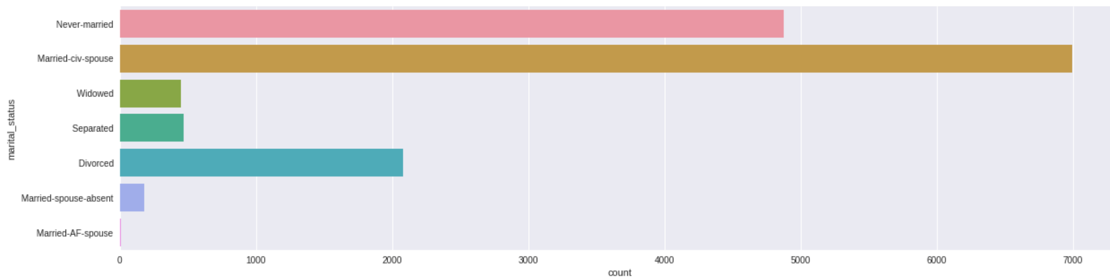

native_country
United-States
Cuba
Jamaica
India
Mexico
Puerto-Rico
Honduras
England
Canada
Germany
Iran
Philippines
Poland
Columbia
Cambodia
Thailand
Ecuador
Laos
Taiwan
Haiti
Portugal
Dominican-Republic
El-Salvador
France
Guatemala
Italy
China
South
Japan
Yugoslavia
Peru
Outlying-US(Guam-USVI-etc)
Scotland
Trinadad&Tobago
Greece
Nicaragua
Vietnam
Hong
Ireland
Hungary
Holand-Netherlands
0
5000
10000
15000
20000
25000
count

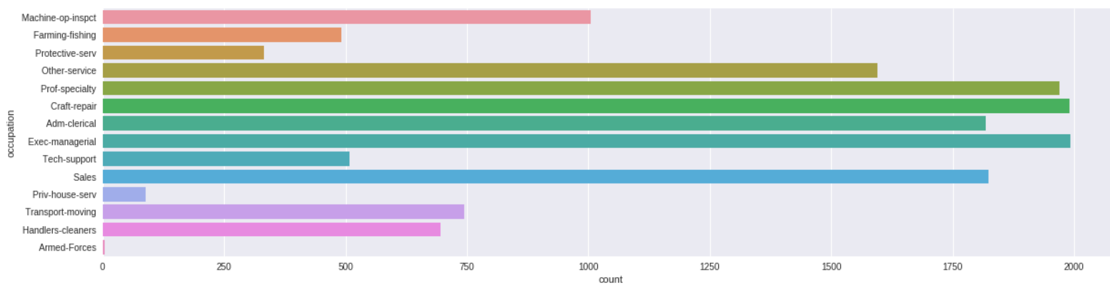


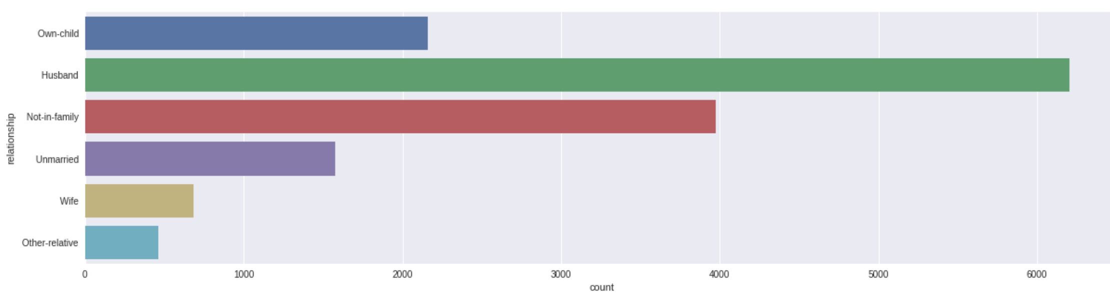

```
In [62]: for i, col in enumerate(cat_cols):
             if(col!='native_country'):
                 plt.figure(i,figsize = (20,5))
                 sns.countplot(y=col, data=test_set)
             else:
                 plt.figure(i,figsize = (20,10))
                 sns.countplot(y=col, data=test_set)
```

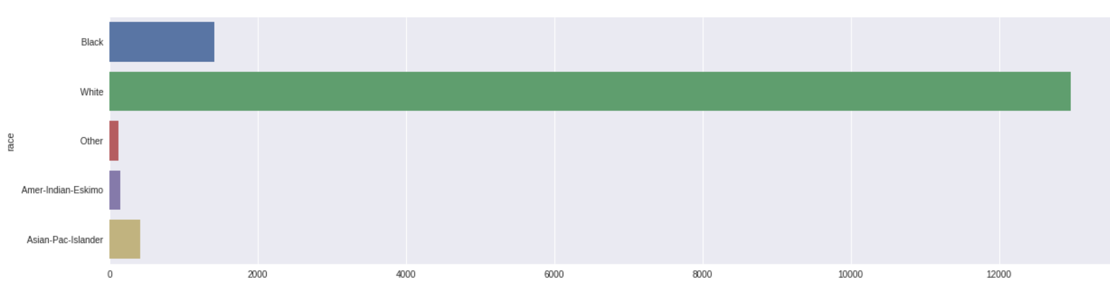


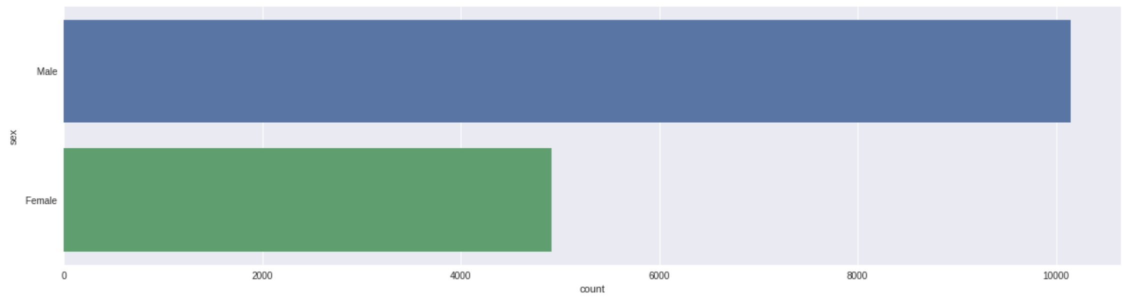



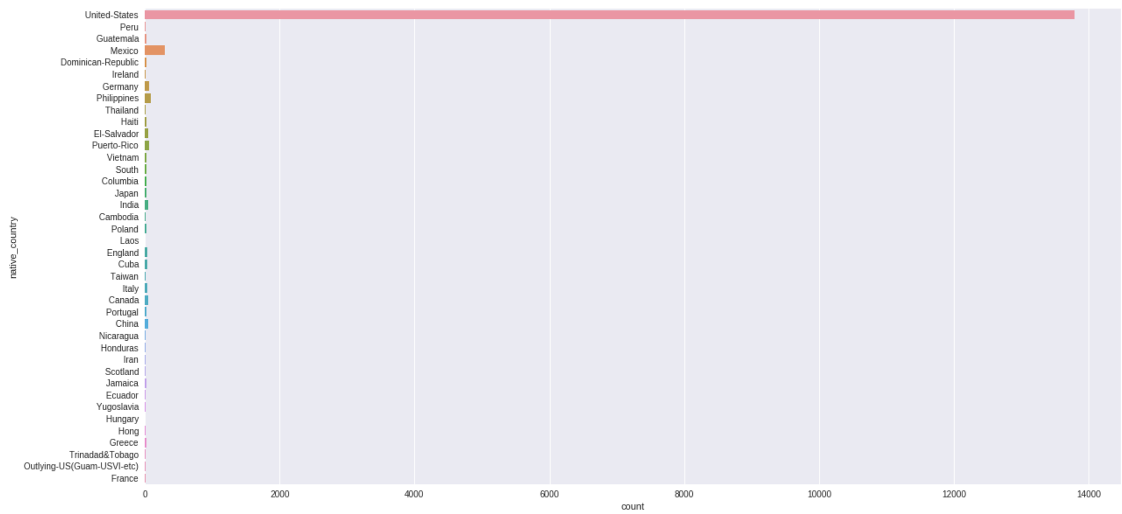

Never-married
Married-civ-spouse
Widowed
Separated
Divorced
Married-spouse-absent
Married-AF-spouse
marital_status
0
1000
2000
3000
4000
5000
6000
7000
count

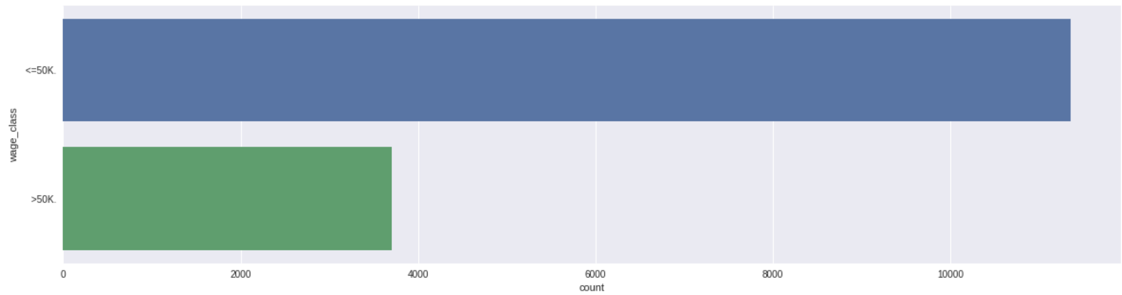

Machine-op-inspct
Farming-fishing
Protective-serv
Other-service
Prof-specialty
Craft-repair
Adm-clerical
Exec-managerial
Tech-support
Sales
Priv-house-serv
Transport-moving
Handlers-cleaners
Armed-Forces
occupation
0
250
500
750
1000
1250
1500
1750
2000
count


Own-child
Husband
Not-in-family
Unmarried
Wife
Other-relative
relationship
0
1000
2000
3000
4000
5000
6000
count


Black
White
Other
Amer-Indian-Eskimo
Asian-Pac-Islander
race
0
2000
4000
6000
8000
10000
12000
count



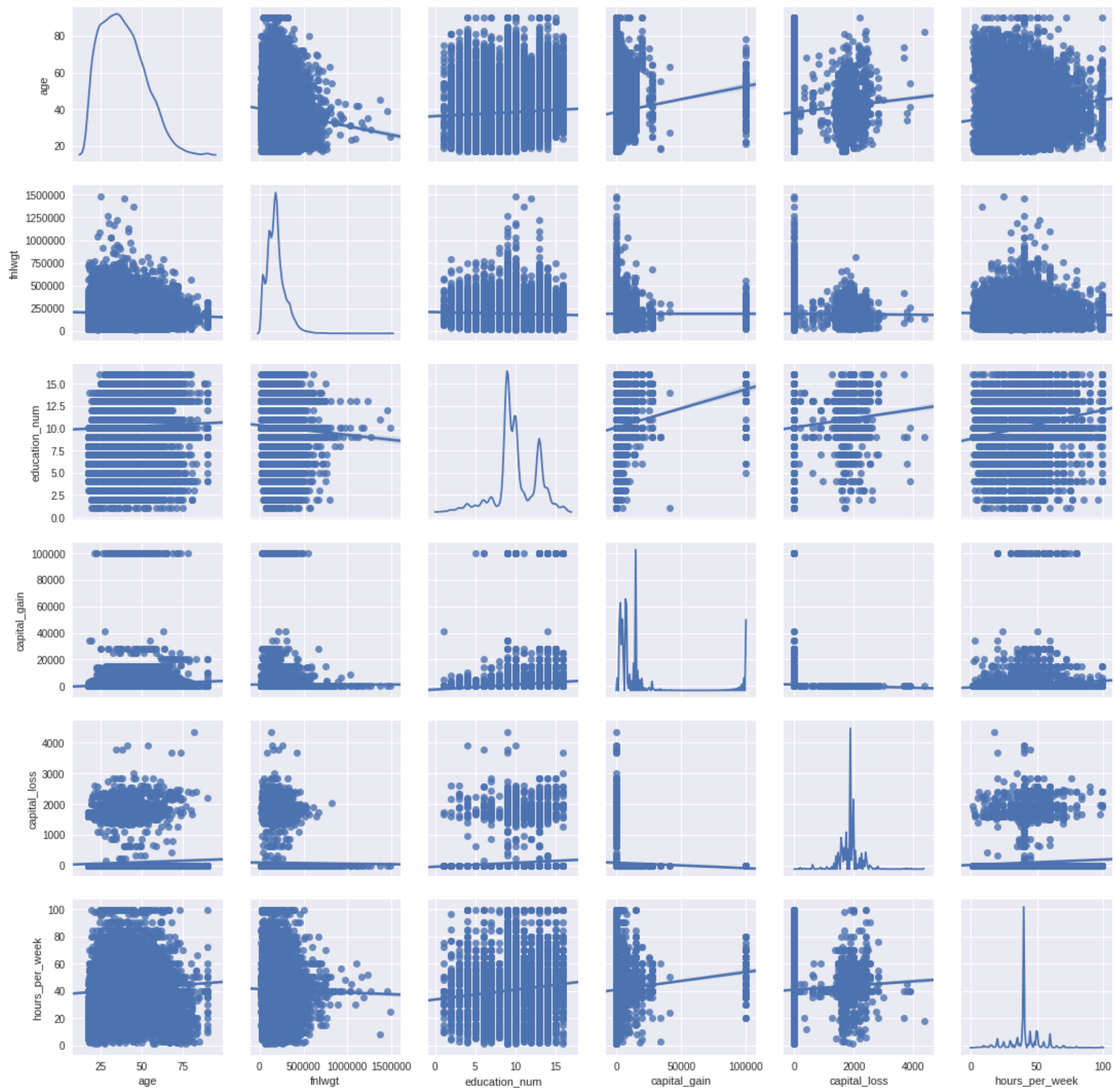




## 7.2 Bi-variate

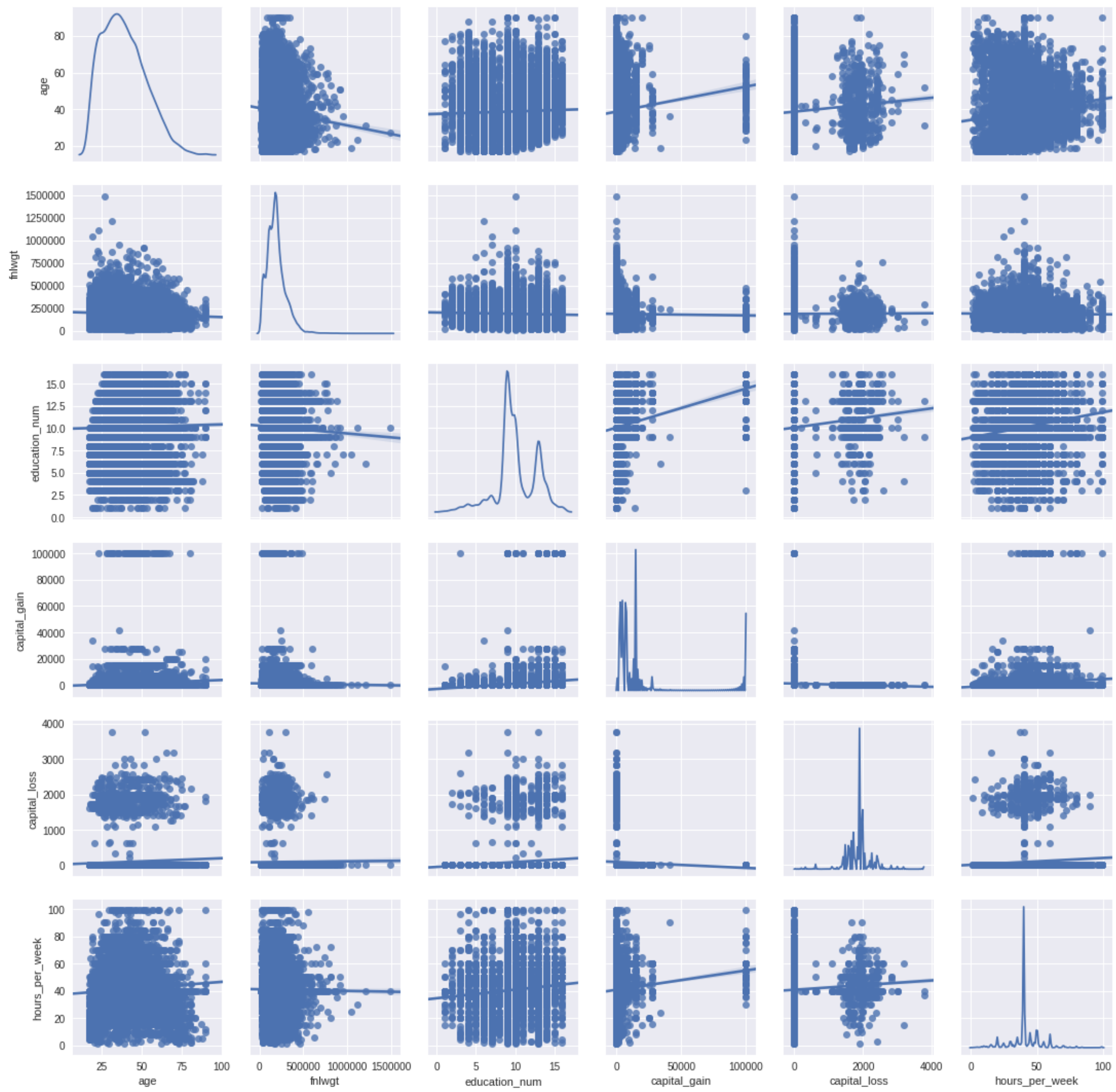
```
In [63]: sns.pairplot(train_set[num_cols],kind ='reg',diag_kind='kde')
```

```
Out[63]: <seaborn.axisgrid.PairGrid at 0x7f900f7057f0>
```



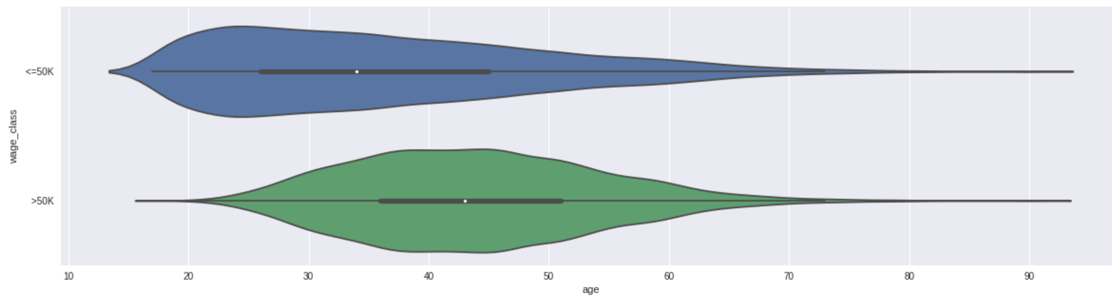

```
In [64]: sns.pairplot(test_set[num_cols],kind ='reg',diag_kind='kde')
```

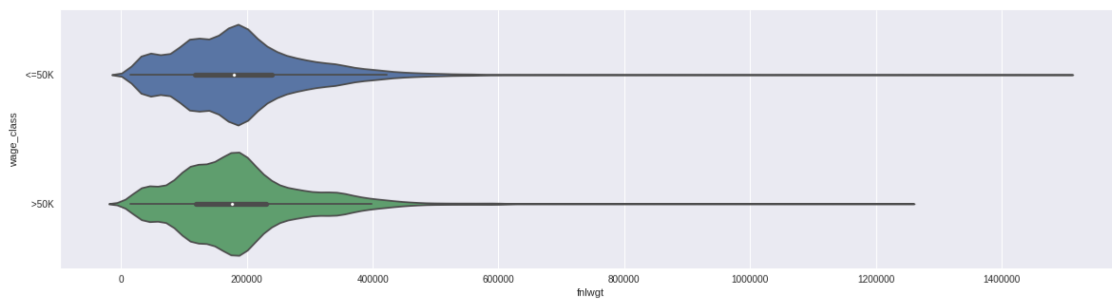
```
Out[64]: <seaborn.axisgrid.PairGrid at 0x7f900f586278>
```

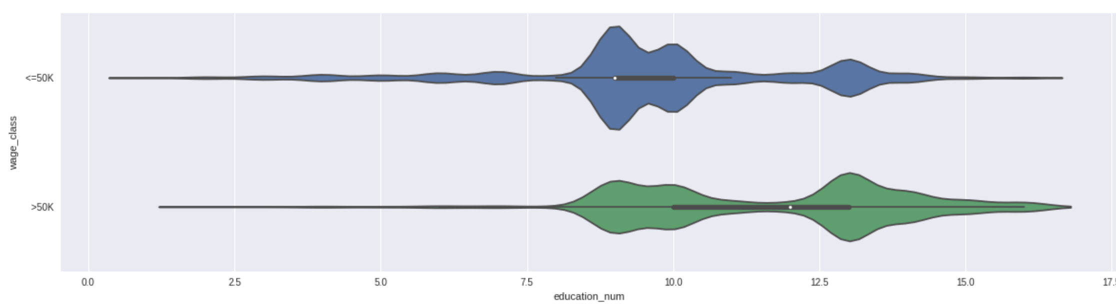


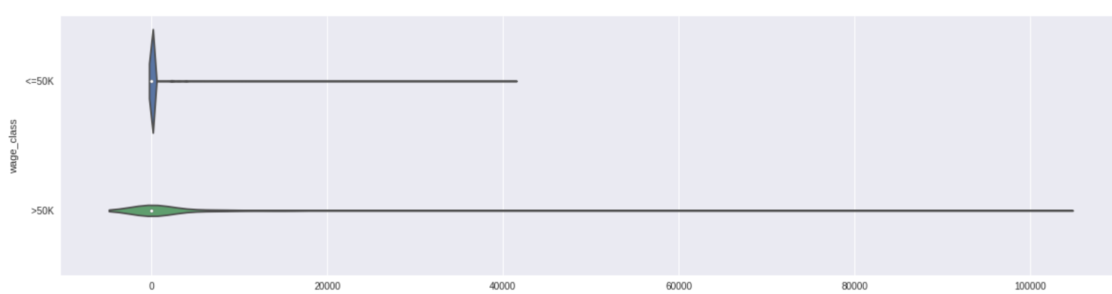

*None of the numerical columns are strongly correlated with each other, either in train_set or test_set. However, it is interesting to note that education_num is more correlated with capital_gain than capital_loss*

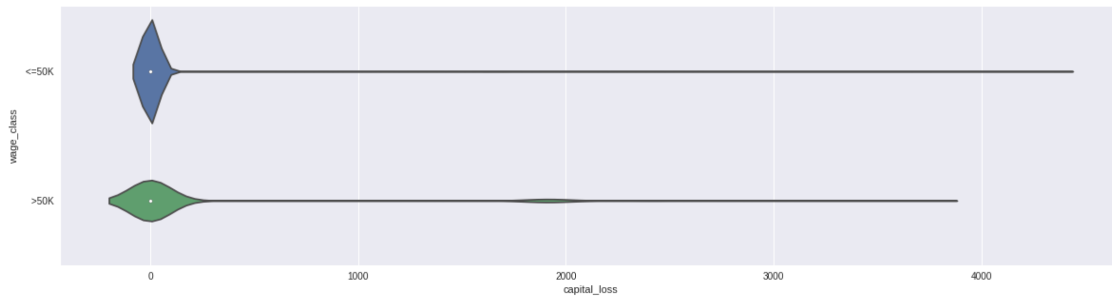
```
In [65]: for i, col in enumerate(num_cols):
             plt.figure(i,figsize = (20,5))
             sns.violinplot(x=col,y='wage_class', data=train_set)
```

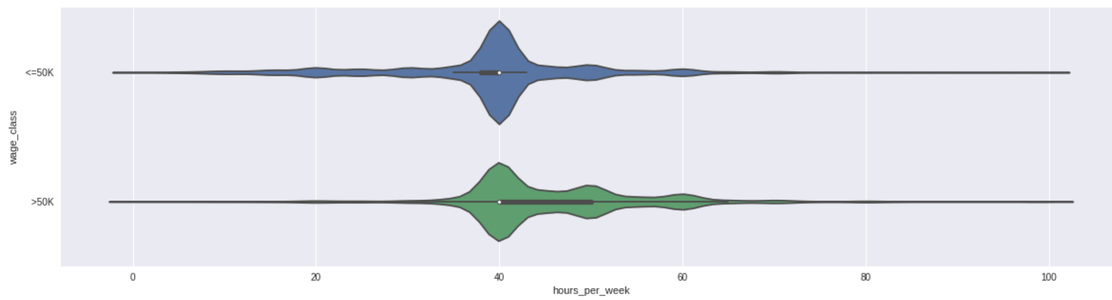

wage_class
<=50K
>50K
10 20 30 40 50 60 70 80 90
age


wage_class
<=50K
>50K
0 200000 400000 600000 800000 1000000 1200000 1400000
fnlwgt

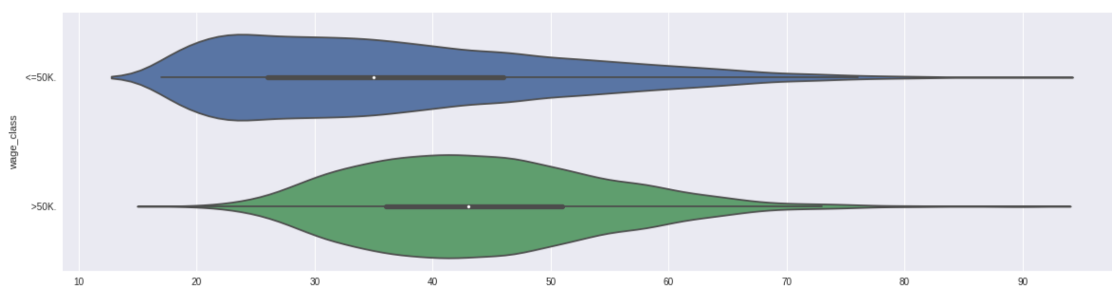

wage_class
<=50K
>50K
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
education_num

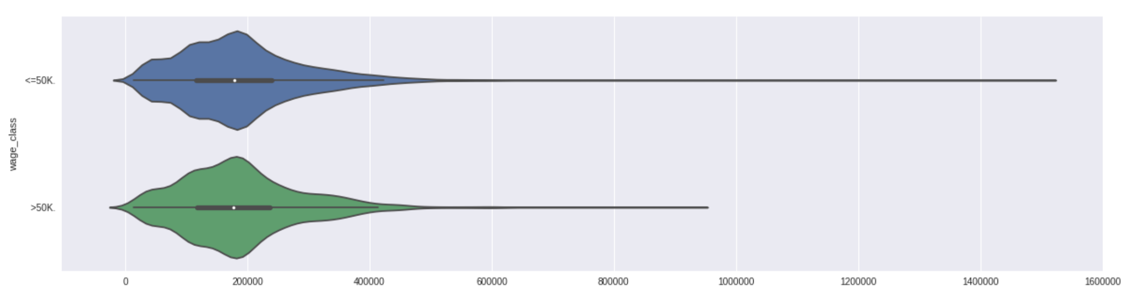

wage_class
<=50K
>50K
0 20000 40000 60000 80000 100000
capital_gain

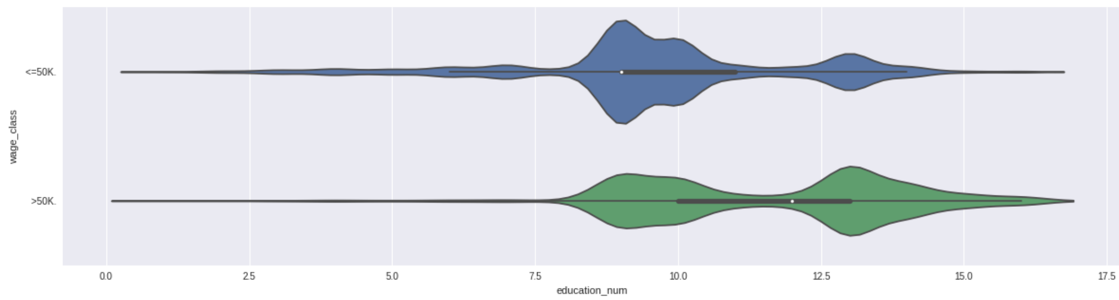


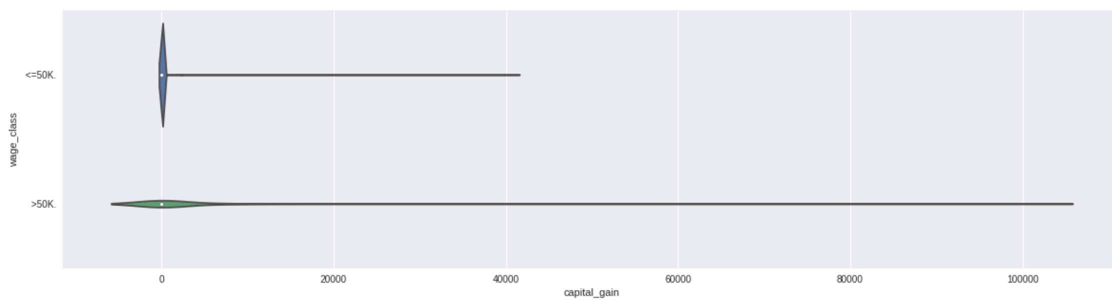

In [66]:
```python
for i, col in enumerate(num_cols):
    plt.figure(i,figsize = (20,5))
    sns.violinplot(x=col,y='wage_class', data=test_set)
```



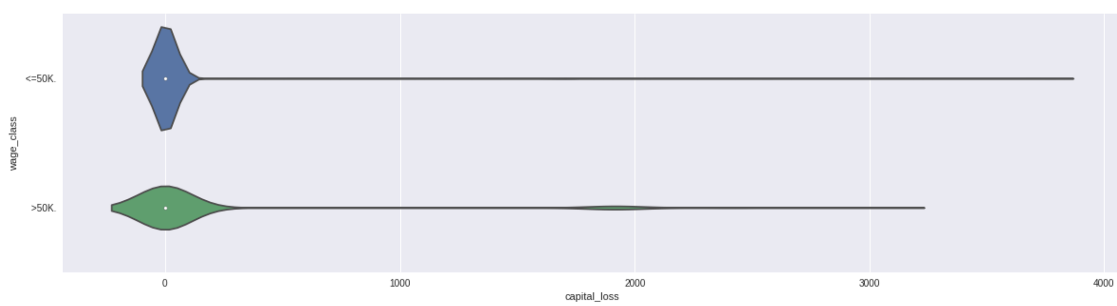





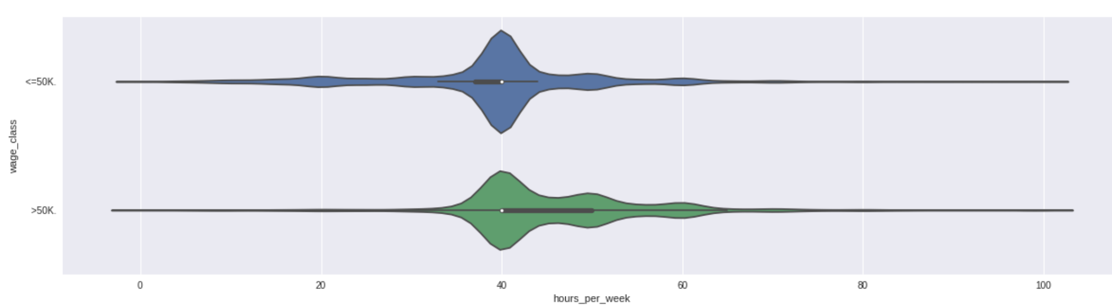
<=50K.
>50K.
wage_class
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
education_num

<=50K.
>50K.
wage_class
0
20000
40000
60000
80000
100000
capital_gain

<=50K.
>50K.
wage_class
0
1000
2000
3000
4000
capital_loss

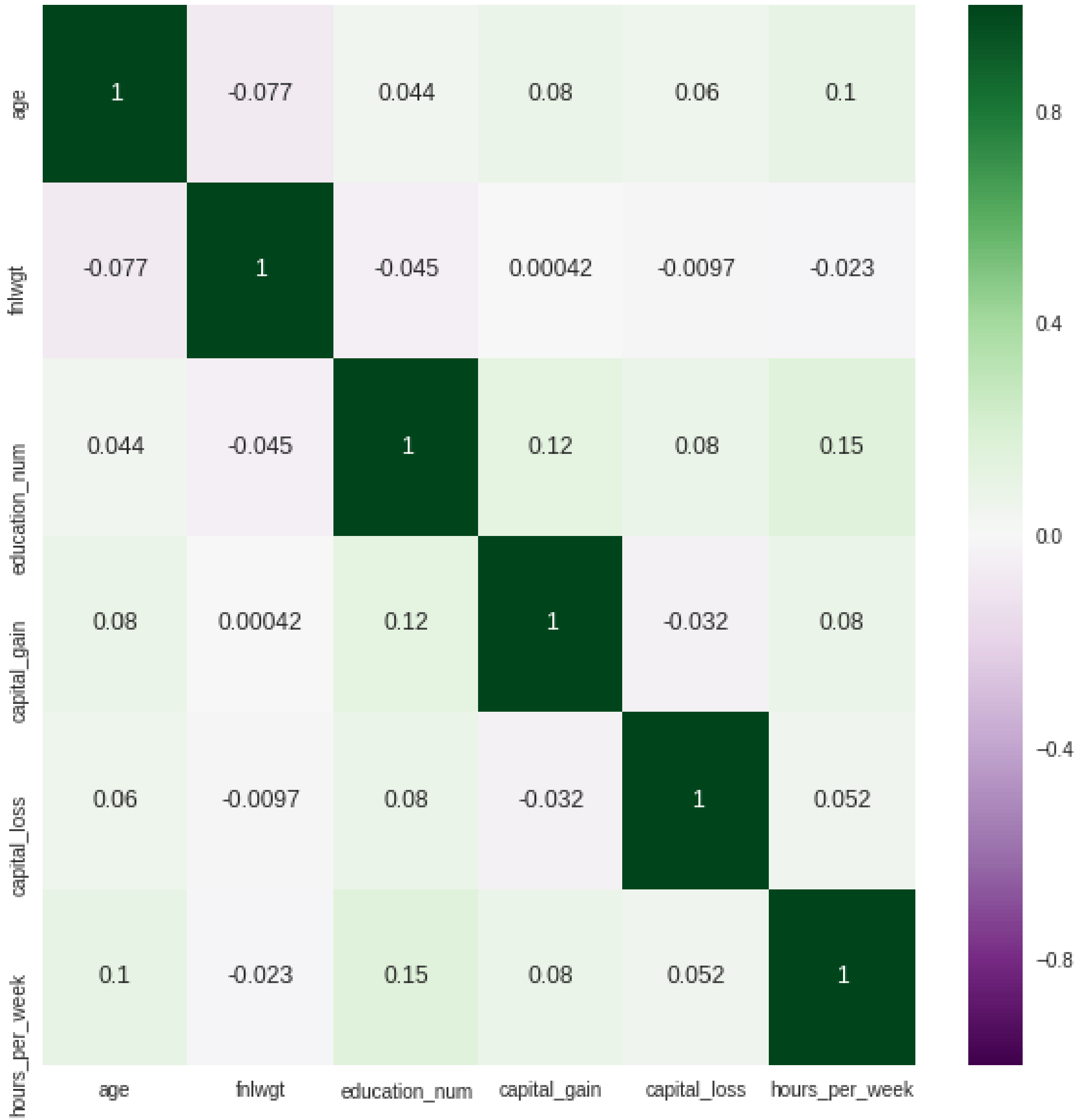



<=50K.
>50K.
wage_class
0
20
40
60
80
100
hours_per_week

### 7.3 Multi-variate

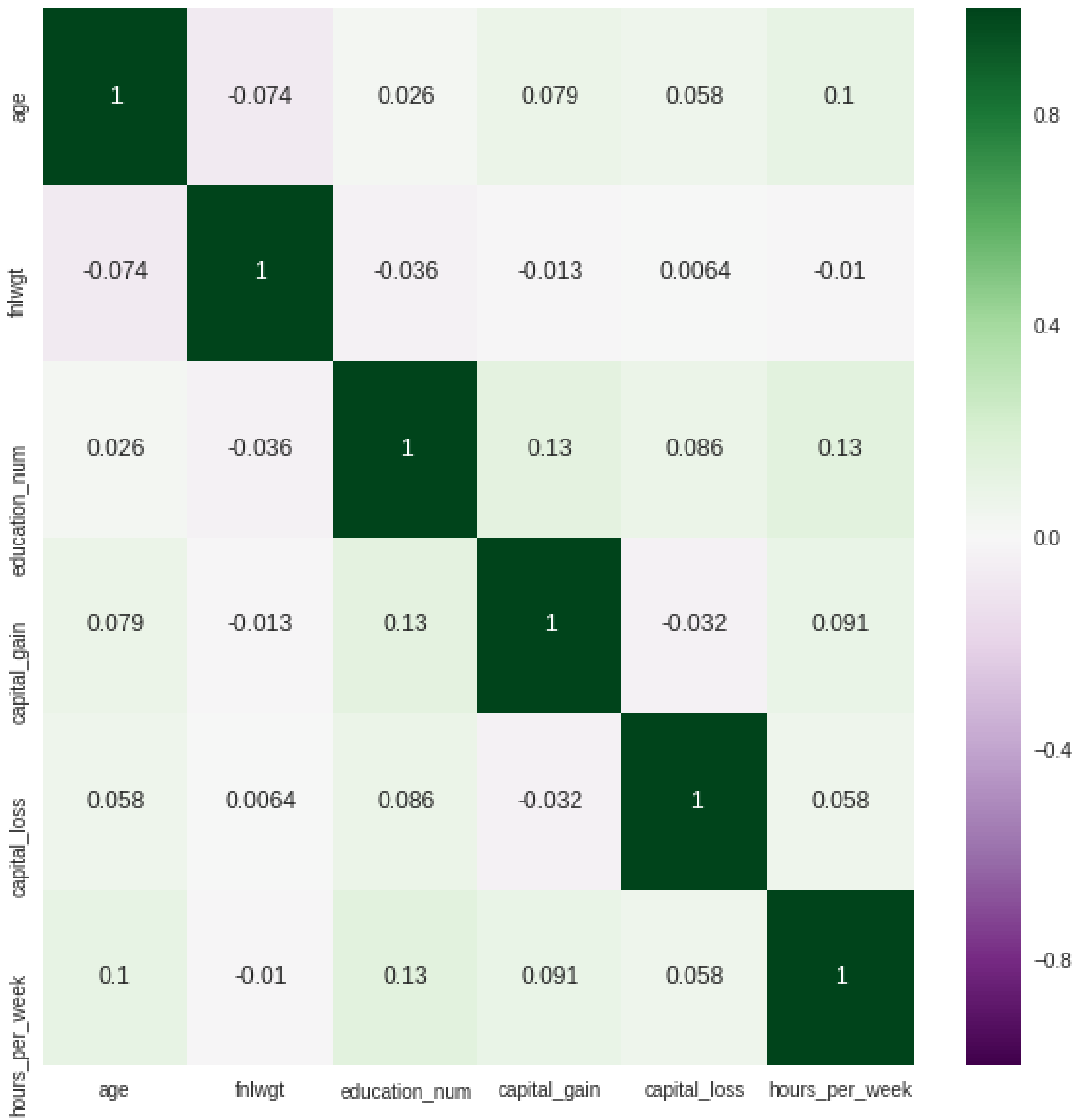
```
In [67]: plt.figure(figsize=(10,10))
         sns.heatmap(train_set.corr(), annot = True,cmap= "PRGn")
```

```
Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x7f900bff6470>
```




```
In [68]: plt.figure(figsize=(10,10))
         sns.heatmap(test_set.corr(), annot = True,cmap= "PRGn")
```

```
Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x7f900acc2390>
```




# 8 Engineer Features

## 8.1 Encode Categorical Columns

```
In [0]: for col in train_set.columns: # Loop through all columns in the dataframe
            if train_set[col].dtype == 'object': # Only apply for columns with categorical str
                train_set[col] = pd.Categorical(train_set[col]).codes # Replace strings with a
```

```
In [0]: for col in test_set.columns: # Loop through all columns in the dataframe
            if test_set[col].dtype == 'object': # Only apply for columns with categorical stri
                test_set[col] = pd.Categorical(test_set[col]).codes # Replace strings with an
```

## 9 Generate Input Vector X and Output Y, and Split the Data for Training and Testing

In [0]:
```
x_train = train_set.drop('wage_class', axis =1)
y_train = train_set['wage_class']
x_test = test_set.drop('wage_class', axis =1)
y_test = test_set['wage_class']
```

In [72]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[72]:
```
((30162, 14), (30162,), (15060, 14), (15060,))
```

## 10 Fit the Base Models and Collect the Metrics

### 10.1 Logistic Regression

In [73]:
```
log_res = LogisticRegression()
model_lr = log_res.fit(x_train, y_train)

y_test_pred = model_lr.predict(x_test)

y_test_pred_prob = model_lr.predict_proba(x_test)

# Generate model evaluation metrics for the Logistic Regression
print("Performance metrics of the model for the Logistic Regression")
print("-"*100)
print("Accuracy: ", metrics.accuracy_score(y_test, y_test_pred))
print("Precision Score: ",metrics.precision_score(y_test, y_test_pred))
print("Recall Score: ",metrics.recall_score(y_test, y_test_pred))
print("AUROC Score: ",metrics.roc_auc_score(y_test,  y_test_pred_prob[:,1]))
print()
print("Confusion Matrix:  \n ",metrics.confusion_matrix(y_test, y_test_pred))
print()
print("Classification Report:\n ",metrics.classification_report(y_test, y_test_pred))
```

```
Performance metrics of the model for the Logistic Regression
----------------------------------------------------------------------------------------------------
Accuracy:  0.7847941567065073
Precision Score:  0.6284275321768327
Recall Score:  0.3035135135135135
AUROC Score:  0.7570339622192616

Confusion Matrix:
  [[10696   664]
 [ 2577  1123]]

Classification Report:
                precision    recall  f1-score   support
```

```
           0       0.81      0.94      0.87     11360
           1       0.63      0.30      0.41      3700

   micro avg       0.78      0.78      0.78     15060
   macro avg       0.72      0.62      0.64     15060
weighted avg       0.76      0.78      0.76     15060
```

### 10.2 Other Classifiers

```
In [0]: classifiers = [
                    ("Logistic Regression - ", LogisticRegression()),
                    ("K-Nearest Neighbors - ",  KNeighborsClassifier(2)),
                    ("Naive Bayes - ",  GaussianNB()),
                    ("Decision Tree - ",  DecisionTreeClassifier(max_depth=5)),
                    ("Random Forest - ",  RandomForestClassifier(n_estimators=100)),
                    ("AdaBoost - ",  AdaBoostClassifier(n_estimators=100)),
                    ("XGBoost - ", XGBClassifier(n_estimators=100,objective='binary:logistic')
```

```
In [75]: # Generate model evaluation metrics
         for clf in classifiers:
             clf[1].fit(x_train, y_train)
             y_test_pred= clf[1].predict(x_test)
             y_test_pred_prob= clf[1].predict_proba(x_test)
             print(clf[0],
                   "\n\t Accuracy: ", metrics.accuracy_score(y_test, y_test_pred),
                   "\n\t Precision Score: ",metrics.precision_score(y_test, y_test_pred),
                   "\n\t Recall Score: ",metrics.recall_score(y_test, y_test_pred),
                   "\n\t AUROC Score: ",metrics.roc_auc_score(y_test,  y_test_pred_prob[:,1]),
                   "\n\t Confusion Matrix:  \n ",metrics.confusion_matrix(y_test, y_test_pred)
                   "\n\t Classification Report:\n ",metrics.classification_report(y_test, y_tes
```

```
Logistic Regression -
         Accuracy:  0.7847941567065073
         Precision Score:  0.6284275321768327
         Recall Score:  0.3035135135135135
         AUROC Score:  0.7570339622192616
         Confusion Matrix:
  [[10696   664]
 [ 2577  1123]]
         Classification Report:
                precision    recall  f1-score   support

           0       0.81      0.94      0.87     11360
           1       0.63      0.30      0.41      3700
```

```
   micro avg       0.78      0.78      0.78     15060
   macro avg       0.72      0.62      0.64     15060
weighted avg       0.76      0.78      0.76     15060


K-Nearest Neighbors -
         Accuracy:  0.7768924302788844
         Precision Score:  0.6010701545778835
         Recall Score:  0.2732432432432432
         AUROC Score:  0.6567672249714503
         Confusion Matrix:
  [[10689   671]
 [ 2689  1011]]
         Classification Report:
                precision    recall  f1-score   support

           0       0.80      0.94      0.86     11360
           1       0.60      0.27      0.38      3700

   micro avg       0.78      0.78      0.78     15060
   macro avg       0.70      0.61      0.62     15060
weighted avg       0.75      0.78      0.74     15060


Naive Bayes -
         Accuracy:  0.7885790172642763
         Precision Score:  0.6469248291571754
         Recall Score:  0.307027027027027
         AUROC Score:  0.8221595807955843
         Confusion Matrix:
  [[10740   620]
 [ 2564  1136]]
         Classification Report:
                precision    recall  f1-score   support

           0       0.81      0.95      0.87     11360
           1       0.65      0.31      0.42      3700

   micro avg       0.79      0.79      0.79     15060
   macro avg       0.73      0.63      0.64     15060
weighted avg       0.77      0.79      0.76     15060


Decision Tree -
         Accuracy:  0.8416334661354582
         Precision Score:  0.7689161554192229
         Recall Score:  0.5081081081081081
         AUROC Score:  0.8738502926341835
```

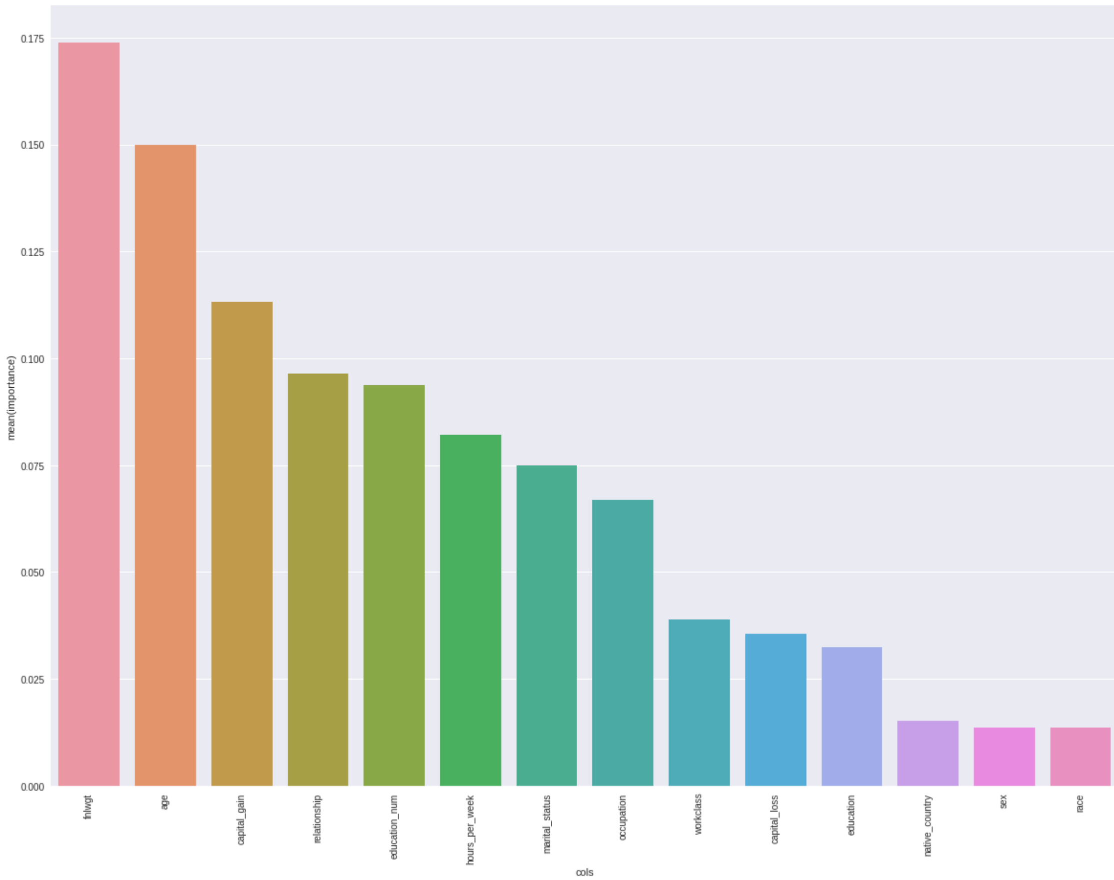
```
        Confusion Matrix:
  [[10795   565]
 [ 1820  1880]]
        Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.95      0.90     11360
           1       0.77      0.51      0.61      3700

   micro avg       0.84      0.84      0.84     15060
   macro avg       0.81      0.73      0.76     15060
weighted avg       0.83      0.84      0.83     15060


Random Forest -
        Accuracy:  0.8518592297476759
        Precision Score:  0.7382419721050925
        Recall Score:  0.6151351351351352
        AUROC Score:  0.9041659687856871
        Confusion Matrix:
  [[10553   807]
 [ 1424  2276]]
        Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.93      0.90     11360
           1       0.74      0.62      0.67      3700

   micro avg       0.85      0.85      0.85     15060
   macro avg       0.81      0.77      0.79     15060
weighted avg       0.85      0.85      0.85     15060


AdaBoost -
        Accuracy:  0.8594289508632138
        Precision Score:  0.7609627431585888
        Recall Score:  0.6237837837837837
        AUROC Score:  0.9162531285687096
        Confusion Matrix:
  [[10635   725]
 [ 1392  2308]]
        Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.94      0.91     11360
           1       0.76      0.62      0.69      3700

   micro avg       0.86      0.86      0.86     15060
```

```
   macro avg       0.82      0.78      0.80     15060
weighted avg       0.85      0.86      0.85     15060


XGBoost -
         Accuracy:  0.8615537848605578
         Precision Score:  0.7920433996383364
         Recall Score:  0.5918918918918918
         AUROC Score:  0.9181394175866008
         Confusion Matrix:
  [[10785   575]
 [ 1510  2190]]
         Classification Report:
                precision    recall  f1-score   support

           0       0.88      0.95      0.91     11360
           1       0.79      0.59      0.68      3700

   micro avg       0.86      0.86      0.86     15060
   macro avg       0.83      0.77      0.79     15060
weighted avg       0.86      0.86      0.85     15060
```

## 11 Select Features

In [76]:
```
rndf = RandomForestClassifier(n_estimators=150)
rndf.fit(x_train, y_train)
importance = pd.DataFrame.from_dict({'cols':x_train.columns, 'importance': rndf.featu
importance = importance.sort_values(by='importance', ascending=False)
plt.figure(figsize=(20,15))
sns.barplot(importance.cols, importance.importance)
plt.xticks(rotation=90)
```

Out[76]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13]),
 <a list of 14 Text xticklabel objects>)
```




```
In [77]: imp_cols = importance[importance.importance >= 0.03].cols.values
         imp_cols
```

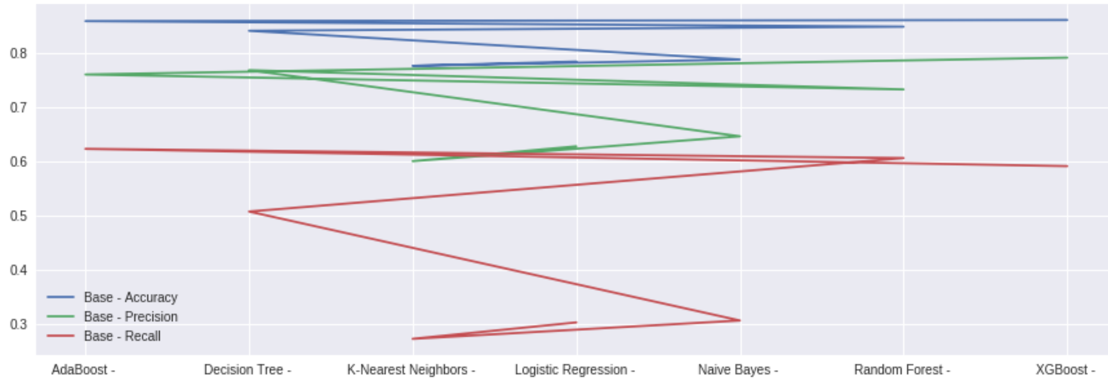
```
Out[77]: array(['fnlwgt', 'age', 'capital_gain', 'relationship', 'education_num',
                'hours_per_week', 'marital_status', 'occupation', 'workclass',
                'capital_loss', 'education'], dtype=object)
```

```
In [78]: # Generate model evaluation metrics
         print("Base Models")
         print('-'*60)
         accuracy_base = []
         precision_base =[]
         recall_base = []
         model_names = [i[0] for i in classifiers]
         for clf in classifiers:
             clf[1].fit(x_train, y_train)
             y_test_pred= clf[1].predict(x_test)
             accuracy_base.append(metrics.accuracy_score(y_test, y_test_pred))
             precision_base.append(metrics.precision_score(y_test, y_test_pred))
             recall_base.append(metrics.recall_score(y_test, y_test_pred))
```

```
            print(clf[0],
                  "\n\t Accuracy: ", metrics.accuracy_score(y_test, y_test_pred),
                  "\n\t Precision Score: ",metrics.precision_score(y_test, y_test_pred),
                  "\n\t Recall Score: ",metrics.recall_score(y_test, y_test_pred))
```

```
Base Models
------------------------------------------------------------
Logistic Regression -
	 Accuracy:  0.7847941567065073
	 Precision Score:  0.6284275321768327
	 Recall Score:  0.3035135135135135
K-Nearest Neighbors -
	 Accuracy:  0.7768924302788844
	 Precision Score:  0.6010701545778835
	 Recall Score:  0.27324324324324323
Naive Bayes -
	 Accuracy:  0.7885790172642763
	 Precision Score:  0.6469248291571754
	 Recall Score:  0.307027027027027
Decision Tree -
	 Accuracy:  0.8416334661354582
	 Precision Score:  0.7689161554192229
	 Recall Score:  0.5081081081081081
Random Forest -
	 Accuracy:  0.849203187250996
	 Precision Score:  0.7334204508330611
	 Recall Score:  0.6067567567567568
AdaBoost -
	 Accuracy:  0.8594289508632138
	 Precision Score:  0.7609627431585888
	 Recall Score:  0.6237837837837837
XGBoost -
	 Accuracy:  0.8615537848605578
	 Precision Score:  0.7920433996383364
	 Recall Score:  0.5918918918918918
```

In [79]:
```
# Plotting the classification metrics for all the base models
plt.figure(figsize=(15,5))
plt.plot(model_names , accuracy_base, label = "Base - Accuracy")
plt.plot(model_names , precision_base, label = "Base - Precision")
plt.plot(model_names , recall_base, label = "Base - Recall")
plt.legend()
plt.show()
```

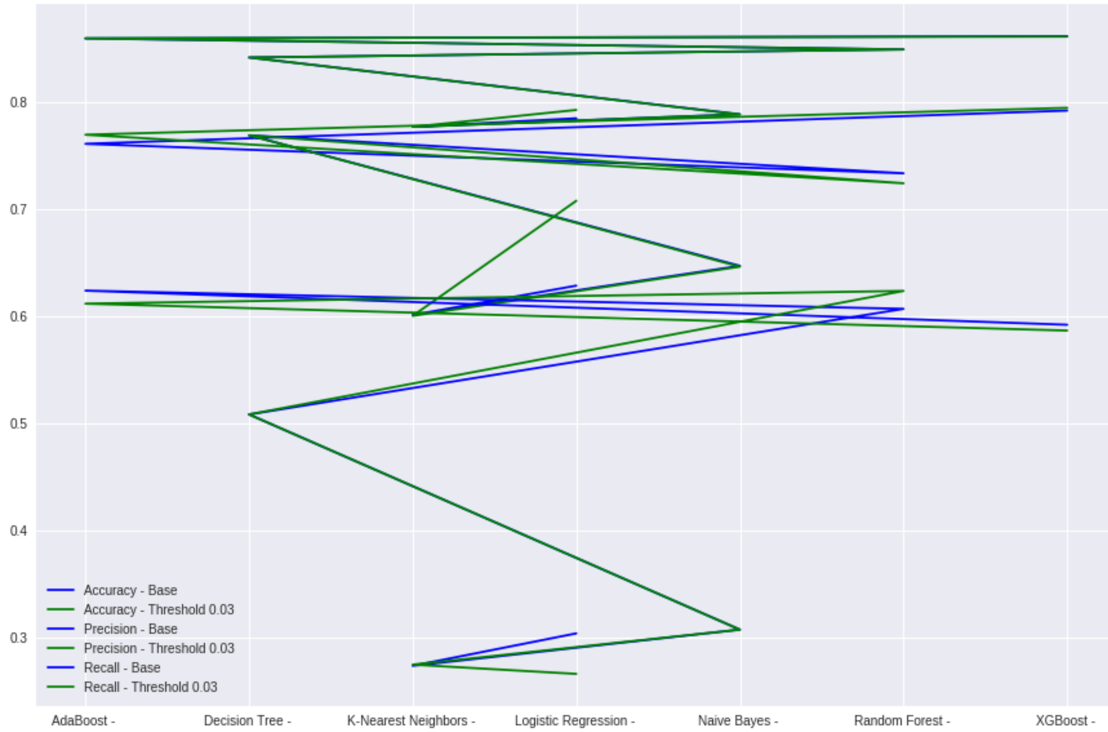



In [80]:
```
# Generate model evaluation metrics
print("Models generated with features having feature importances threshold >= 0.03")
print('-'*60)
accuracy_thresh_03 = []
precision_thresh_03 =[]
recall_thresh_03 = []
for clf in classifiers:
    clf[1].fit(x_train[imp_cols], y_train)
    y_test_pred= clf[1].predict(x_test[imp_cols])
    accuracy_thresh_03.append(metrics.accuracy_score(y_test, y_test_pred))
    precision_thresh_03.append(metrics.precision_score(y_test, y_test_pred))
    recall_thresh_03.append(metrics.recall_score(y_test, y_test_pred))
    print(clf[0],
          "\n\t Accuracy: ", metrics.accuracy_score(y_test, y_test_pred),
          "\n\t Precision Score: ",metrics.precision_score(y_test, y_test_pred),
          "\n\t Recall Score: ",metrics.recall_score(y_test, y_test_pred))
```

```
Models generated with features having feature importances threshold >= 0.03
------------------------------------------------------------
Logistic Regression -
         Accuracy:  0.7926294820717131
         Precision Score:  0.7077033837293016
         Recall Score:  0.2656756756756757
K-Nearest Neighbors -
         Accuracy:  0.7768260292164675
         Precision Score:  0.6002365464222353
         Recall Score:  0.2743243243243243
Naive Bayes -
         Accuracy:  0.7884462151394422
         Precision Score:  0.646188850967008
         Recall Score:  0.307027027027027
Decision Tree -
         Accuracy:  0.8416334661354582
```

```
        Precision Score:  0.7689161554192229
        Recall Score:  0.5081081081081081
Random Forest -
        Accuracy:  0.849136786188579
        Precision Score:  0.724105461393597
        Recall Score:  0.6235135135135135
AdaBoost -
        Accuracy:  0.8595617529880478
        Precision Score:  0.7694661679700782
        Recall Score:  0.6116216216216216
XGBoost -
        Accuracy:  0.8611553784860557
        Precision Score:  0.7945807396558038
        Recall Score:  0.5864864864864865
```

In [81]:
```python
# Plotting the classification metrics for all the base models and models generated fr

plt.figure(figsize=(15,10))
plt.plot(model_names , accuracy_base, label = "Accuracy - Base",c = 'blue')
plt.plot(model_names , accuracy_thresh_03, label = "Accuracy - Threshold 0.03", c = 'g

plt.plot(model_names , precision_base, label = "Precision - Base",c = 'blue')
plt.plot(model_names , precision_thresh_03, label = "Precision - Threshold 0.03", c =


plt.plot(model_names , recall_base, label = "Recall - Base",c = 'blue')
plt.plot(model_names , recall_thresh_03, label = "Recall - Threshold 0.03", c = 'green

plt.legend()
plt.show()
```




In [82]:
```
imp_cols = importance[importance.importance >= 0.014 ].cols.values
imp_cols
```

Out[82]:
```
array(['fnlwgt', 'age', 'capital_gain', 'relationship', 'education_num',
       'hours_per_week', 'marital_status', 'occupation', 'workclass',
       'capital_loss', 'education', 'native_country'], dtype=object)
```

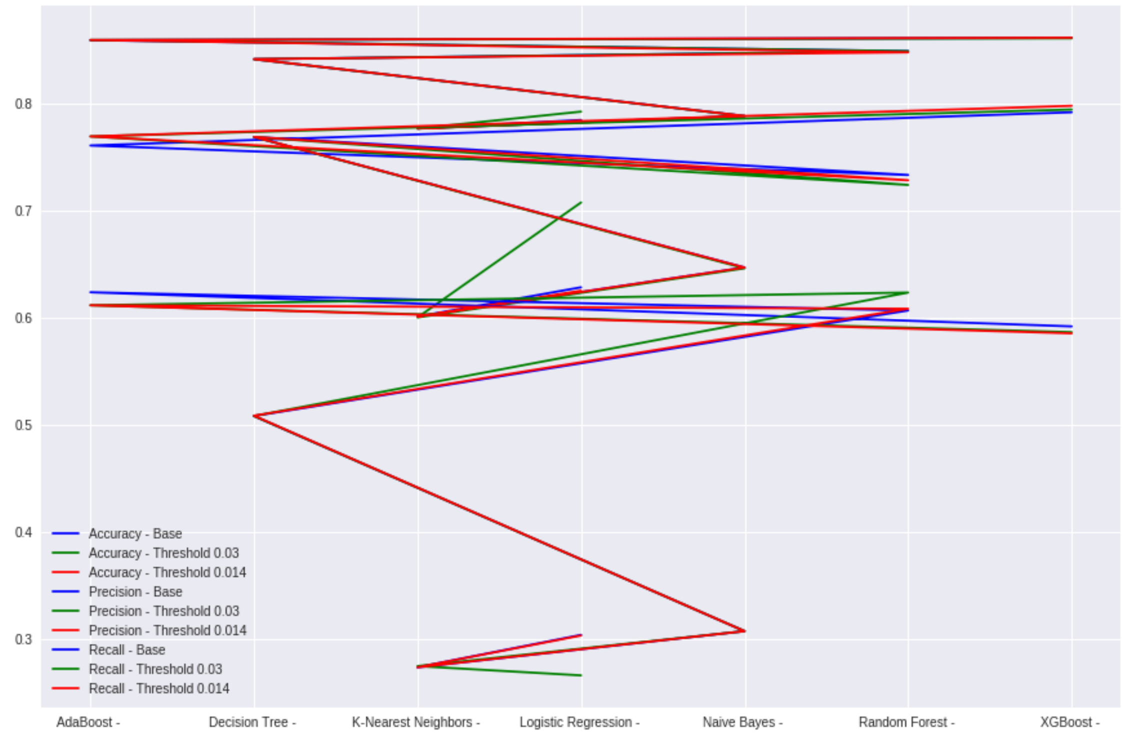
In [83]:
```
# Generate model evaluation metrics
print("Models generated with features having feature importances threshold >= 0.014")
print('-'*60)
accuracy_thresh_014 = []
precision_thresh_014 =[]
recall_thresh_014 = []

for clf in classifiers:
    clf[1].fit(x_train[imp_cols], y_train)
    y_test_pred= clf[1].predict(x_test[imp_cols])
    y_test_pred_prob= clf[1].predict_proba(x_test[imp_cols])
    accuracy_thresh_014.append(metrics.accuracy_score(y_test, y_test_pred))
    precision_thresh_014.append(metrics.precision_score(y_test, y_test_pred))
    recall_thresh_014.append(metrics.recall_score(y_test, y_test_pred))
    print(clf[0],
          "\n\t Accuracy: ", metrics.accuracy_score(y_test, y_test_pred),
          "\n\t Precision Score: ",metrics.precision_score(y_test, y_test_pred),
```

```
                    "\n\t Recall Score: ",metrics.recall_score(y_test, y_test_pred))
```

```
Models generated with features having feature importances threshold >= 0.014
------------------------------------------------------------
Logistic Regression -
         Accuracy:  0.7841301460823373
         Precision Score:  0.6252091466815394
         Recall Score:  0.302972972972973
K-Nearest Neighbors -
         Accuracy:  0.7768924302788844
         Precision Score:  0.6010701545778835
         Recall Score:  0.27324324324324323
Naive Bayes -
         Accuracy:  0.7885790172642763
         Precision Score:  0.6469248291571754
         Recall Score:  0.307027027027027
Decision Tree -
         Accuracy:  0.8416998671978752
         Precision Score:  0.7690106295993459
         Recall Score:  0.5083783783783784
Random Forest -
         Accuracy:  0.8480743691899071
         Precision Score:  0.7284789644012944
         Recall Score:  0.6083783783783784
AdaBoost -
         Accuracy:  0.8595617529880478
         Precision Score:  0.7694661679700782
         Recall Score:  0.6116216216216216
XGBoost -
         Accuracy:  0.8617529880478088
         Precision Score:  0.7980840088430361
         Recall Score:  0.5854054054054054
```

In [84]:
```
# Plotting the classification metrics for all the base models and models generated fr

plt.figure(figsize=(15,10))
plt.plot(model_names , accuracy_base, label = "Accuracy - Base",c = 'blue')
plt.plot(model_names , accuracy_thresh_03, label = "Accuracy - Threshold 0.03", c = 'g
plt.plot(model_names , accuracy_thresh_014, label = "Accuracy - Threshold 0.014", c=


plt.plot(model_names , precision_base, label = "Precision - Base",c = 'blue')
plt.plot(model_names , precision_thresh_03, label = "Precision - Threshold 0.03", c =
plt.plot(model_names , precision_thresh_014, label = "Precision - Threshold 0.014", c=
```

```
plt.plot(model_names , recall_base, label = "Recall - Base",c = 'blue')
plt.plot(model_names , recall_thresh_03, label = "Recall - Threshold 0.03", c = 'green
plt.plot(model_names , recall_thresh_014, label = "Recall - Threshold 0.014",c= 'red')

plt.legend()
plt.show()
```




***Our base model with all the features performs as good as the models for which features were removed with a feature importance threshold of 0.03, 0.014.The difference recall and precision metrics along with accuracy are also too small to notice in models where the features are removed.So we stick with the models with all the features***

***However, we choose Decision Tree, Random Forest, Adaboost and XGBoost classifiers for further optimization.***

## 12 Validate Model

```
In [85]: scoring = 'accuracy'
         results=[]
         names=[]
         for classifier_name, model in classifiers:
             kfold = KFold(n_splits=10, random_state=100)
             cv_results = cross_val_score(model, x_train,y_train, cv=kfold, scoring=scoring)
             results.append(cv_results)
             names.append(classifier_name)
```

```
            print(classifier_name,
                                 "\n\t CV-Mean:", cv_results.mean(),
                               "\n\t CV-Std. Dev:",  cv_results.std(),"\n")
```

```
Logistic Regression -
         CV-Mean: 0.7875471136592027
         CV-Std. Dev: 0.005263509662642161

K-Nearest Neighbors -
         CV-Mean: 0.7798556302086584
         CV-Std. Dev: 0.005827837382234451

Naive Bayes -
         CV-Mean: 0.7885088169690938
         CV-Std. Dev: 0.006169737575906988

Decision Tree -
         CV-Mean: 0.8440757018803262
         CV-Std. Dev: 0.008732608457640546

Random Forest -
         CV-Mean: 0.8526294301346307
         CV-Std. Dev: 0.00474243311176176

AdaBoost -
         CV-Mean: 0.8620121917445702
         CV-Std. Dev: 0.006018891855664965

XGBoost -
         CV-Mean: 0.8598571621993495
         CV-Std. Dev: 0.005960042174289529
```

In [86]:
```
scoring = 'f1'
results=[]
names=[]
for classifier_name, model in classifiers:
    kfold = KFold(n_splits=10, random_state=100)
    cv_results = cross_val_score(model, x_train,y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(classifier_name)
    print(classifier_name,
                         "\n\t CV-Mean:", cv_results.mean(),
                       "\n\t CV-Std. Dev:",  cv_results.std(),"\n")
```

```
Logistic Regression -
         CV-Mean: 0.4104019125355709
```

```
        CV-Std. Dev: 0.015056938505725355

K-Nearest Neighbors -
        CV-Mean: 0.3884925024581391
        CV-Std. Dev: 0.01004596352177543

Naive Bayes -
        CV-Mean: 0.4213398074876366
        CV-Std. Dev: 0.018451912575148527

Decision Tree -
        CV-Mean: 0.6219456901958755
        CV-Std. Dev: 0.02094070650418019

Random Forest -
        CV-Mean: 0.6784702964495954
        CV-Std. Dev: 0.016231454489888105

AdaBoost -
        CV-Mean: 0.6924632840004245
        CV-Std. Dev: 0.014821128220726322

XGBoost -
        CV-Mean: 0.6772151069014127
        CV-Std. Dev: 0.012789766416978925
```

***We have better CV mean and Std deviation scores for Decision Tree, Random Forest, Adaboost and XGBoost classifiers than other classifiers. So these models are robust and in addition have good accuracy. I chose f1 as the CV parameter in addition to accuracy because precision and recall as metrics are just as important as accuracy in classification models.***

***We however, still need to optimize the hyper-parameters on these models.***

# 13 Optimize or Tune Model for better Performance

## 13.1 Decision Tree

```
In [0]: param_grid = {'criterion':['gini','entropy'],
                      'max_depth':[2, 3, 4,5, 6, 7, 8, 9],
                      'random_state':[100],
                      'splitter':['best']}

        DT_grid = GridSearchCV(DecisionTreeClassifier(), param_grid=param_grid, cv = 5, verbos
```

```
In [88]: DT_grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  80 out of  80 | elapsed:    6.3s finished
```

```
Out[88]: GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_deptl
                     max_features=None, 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, presort=False, random_state=None,
                     splitter='best'),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'criterion': ['gini', 'entropy'], 'max_depth': [2, 3, 4, 5, 6, 7, 8
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=1)
```

```
In [89]: DT_grid.best_params_
```

```
Out[89]: {'criterion': 'entropy',
          'max_depth': 7,
          'random_state': 100,
          'splitter': 'best'}
```

```
In [0]: model = DT_grid.best_estimator_
        model.fit(x_train, y_train)
        y_test_pred = model.predict(x_test)
```

```
In [91]: model.score(x_test, y_test)
```

```
Out[91]: 0.849203187250996
```

```
In [92]: # Generate model evaluation metrics for the Decision Tree Classifier - Hyperparameter
         print("Performance metrics of the model for the Decision Tree Classifier - Hyperparame
         print("-"*100)
         print("Accuracy: ", metrics.accuracy_score(y_test, y_test_pred))
         print("Precision Score: ",metrics.precision_score(y_test, y_test_pred))
         print("Recall Score: ",metrics.recall_score(y_test, y_test_pred))
         print("AUROC Score: ",metrics.roc_auc_score(y_test,  y_test_pred_prob[:,1]))
         print()
         print("Confusion Matrix:  \n ",metrics.confusion_matrix(y_test, y_test_pred))
         print()
         print("Classification Report:\n ",metrics.classification_report(y_test, y_test_pred))
```

```
Performance metrics of the model for the Decision Tree Classifier - Hyperparameter Tuned
-----------------------------------------------------------------------------------------------
Accuracy:  0.849203187250996
Precision Score:  0.7975843398583924
Recall Score:  0.5175675675675676
AUROC Score:  0.9184356680624287
```

```
Confusion Matrix:
  [[10874   486]
 [ 1785  1915]]

Classification Report:
                precision    recall  f1-score   support

           0       0.86      0.96      0.91     11360
           1       0.80      0.52      0.63      3700

   micro avg       0.85      0.85      0.85     15060
   macro avg       0.83      0.74      0.77     15060
weighted avg       0.84      0.85      0.84     15060
```

In [0]:
```
DT_best = pickle.dumps(DT_grid.best_estimator_)
```

### 13.2 Random Forest

In [0]:
```
param_grid = {'criterion':['gini','entropy'],
              'max_depth':[2, 3, 4, 5, 6, 7, 8, 9],
              'random_state':[100],
              'n_estimators':[200,400,600],
              'n_jobs':[-1],
              'random_state':[100],
              'verbose': [0]}

RF_grid = GridSearchCV(RandomForestClassifier(), param_grid=param_grid, cv = 5, verbose
```

In [95]:
```
RF_grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 240 out of 240 | elapsed: 20.3min finished
```

Out[95]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
                     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='warn', n_jobs=None,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False),
                fit_params=None, iid='warn', n_jobs=None,
```

```
             param_grid={'criterion': ['gini', 'entropy'], 'max_depth': [2, 3, 4, 5, 6, 7, 8
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=1)
```

In [96]:
```
RF_grid.best_params_
```

Out[96]:
```
{'criterion': 'gini',
 'max_depth': 9,
 'n_estimators': 400,
 'n_jobs': -1,
 'random_state': 100,
 'verbose': 0}
```

In [0]:
```
model = RF_grid.best_estimator_
model.fit(x_train, y_train)
y_test_pred = model.predict(x_test)
```

In [98]:
```
model.score(x_test, y_test)
```

Out[98]:
```
0.8523240371845949
```

In [99]:
```
# Generate model evaluation metrics for the RandomForest Classifier - Hyperparameter
print("Performance metrics of the model for the RandomForest Classifier - Hyperparamet
print("-"*100)
print("Accuracy: ", metrics.accuracy_score(y_test, y_test_pred))
print("Precision Score: ",metrics.precision_score(y_test, y_test_pred))
print("Recall Score: ",metrics.recall_score(y_test, y_test_pred))
print("AUROC Score: ",metrics.roc_auc_score(y_test,  y_test_pred_prob[:,1]))
print()
print("Confusion Matrix:  \n ",metrics.confusion_matrix(y_test, y_test_pred))
print()
print("Classification Report:\n ",metrics.classification_report(y_test, y_test_pred))
```

```
Performance metrics of the model for the RandomForest Classifier - Hyperparameter Tuned
----------------------------------------------------------------------------------------------------
Accuracy:  0.8523240371845949
Precision Score:  0.7973408541498791
Recall Score:  0.5348648648648648
AUROC Score:  0.9184356680624287

Confusion Matrix:
  [[10857   503]
 [ 1721  1979]]

Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.96      0.91     11360
           1       0.80      0.53      0.64      3700
```

```
   micro avg       0.85      0.85      0.85     15060
   macro avg       0.83      0.75      0.77     15060
weighted avg       0.85      0.85      0.84     15060
```

```
In [0]: RF_best = pickle.dumps(RF_grid.best_estimator_)
```

### 13.3 Adaboost

```
In [101]: AdaBoostClassifier()

Out[101]: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
                    learning_rate=1.0, n_estimators=50, random_state=None)

In [0]: param_grid = {'algorithm':['SAMME.R'],
                      'learning_rate':[0.1, 0.2, 0.3],
                      'n_estimators':[200,400,600],
                      'random_state':[100]}

        AB_grid = GridSearchCV(AdaBoostClassifier(), param_grid=param_grid, cv = 5, verbose=1)

In [103]: AB_grid.fit(x_train, y_train)

Fitting 5 folds for each of 9 candidates, totalling 45 fits


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:  7.0min finished


Out[103]: GridSearchCV(cv=5, error_score='raise-deprecating',
                 estimator=AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
                    learning_rate=1.0, n_estimators=50, random_state=None),
                 fit_params=None, iid='warn', n_jobs=None,
                 param_grid={'algorithm': ['SAMME.R'], 'learning_rate': [0.1, 0.2, 0.3], 'n_es
                 pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                 scoring=None, verbose=1)

In [104]: AB_grid.best_params_

Out[104]: {'algorithm': 'SAMME.R',
           'learning_rate': 0.3,
           'n_estimators': 600,
           'random_state': 100}

In [0]: model = AB_grid.best_estimator_
        model.fit(x_train, y_train)
        y_test_pred = model.predict(x_test)
```

In [106]: model.score(x_test, y_test)

Out[106]: 0.8598937583001328

```
In [107]: # Generate model evaluation metrics for the AdaBoost Classifier - Hyperparameter Tun
          print("Performance metrics of the model for the AdaBoost Classifier - Hyperparameter
          print("-"*100)
          print("Accuracy: ", metrics.accuracy_score(y_test, y_test_pred))
          print("Precision Score: ",metrics.precision_score(y_test, y_test_pred))
          print("Recall Score: ",metrics.recall_score(y_test, y_test_pred))
          print("AUROC Score: ",metrics.roc_auc_score(y_test,  y_test_pred_prob[:,1]))
          print()
          print("Confusion Matrix:  \n ",metrics.confusion_matrix(y_test, y_test_pred))
          print()
          print("Classification Report:\n ",metrics.classification_report(y_test, y_test_pred)
```

```
Performance metrics of the model for the AdaBoost Classifier - Hyperparameter Tuned
----------------------------------------------------------------------------------------------------
Accuracy:  0.8598937583001328
Precision Score:  0.7693089430894309
Recall Score:  0.6137837837837837
AUROC Score:  0.9184356680624287

Confusion Matrix:
  [[10679   681]
 [ 1429  2271]]

Classification Report:
                precision    recall  f1-score   support

           0       0.88      0.94      0.91     11360
           1       0.77      0.61      0.68      3700

   micro avg       0.86      0.86      0.86     15060
   macro avg       0.83      0.78      0.80     15060
weighted avg       0.85      0.86      0.85     15060
```

```
In [0]: AB_best = pickle.dumps(AB_grid.best_estimator_)
```

### 13.4 XGBoost

```
In [0]: param_grid = {'learning_rate':[0.1, 0.2, 0.3],
                      'max_depth':[2, 4, 7],
                      'n_estimators':[200,400,600],
                      'n_jobs':[-1],
                      'objective':['binary:logistic'],
                      'random_state':[100],
```

```
                  'reg_alpha':[0.1, 1, 10],
                  'scale_pos_weight':[1],
                  'silent':[True]}

        XGB_grid = GridSearchCV(XGBClassifier(), param_grid=param_grid, cv = 5, verbose=1)
```

In [110]:
```
XGB_grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 405 out of 405 | elapsed: 45.0min finished
```

Out[110]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'learning_rate': [0.1, 0.2, 0.3], 'max_depth': [2, 4, 7], 'n_estir
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=1)
```

In [111]:
```
XGB_grid.best_params_
```

Out[111]:
```
{'learning_rate': 0.2,
 'max_depth': 4,
 'n_estimators': 200,
 'n_jobs': -1,
 'objective': 'binary:logistic',
 'random_state': 100,
 'reg_alpha': 0.1,
 'scale_pos_weight': 1,
 'silent': True}
```

In [0]:
```
model = XGB_grid.best_estimator_
model.fit(x_train, y_train)
y_test_pred = model.predict(x_test)
```

In [113]:
```
model.score(x_test, y_test)
```

Out[113]:
```
0.8699867197875166
```

In [114]:
```
# Generate model evaluation metrics for the XGBOOST - Hyperparameter Tuned
print("Performance metrics of the model for the XGBOOST Classifier - Hyperparameter T
```

```
print("-"*100)
print("Accuracy: ", metrics.accuracy_score(y_test, y_test_pred))
print("Precision Score: ",metrics.precision_score(y_test, y_test_pred))
print("Recall Score: ",metrics.recall_score(y_test, y_test_pred))
print("AUROC Score: ",metrics.roc_auc_score(y_test,  y_test_pred_prob[:,1]))
print()
print("Confusion Matrix:  \n ",metrics.confusion_matrix(y_test, y_test_pred))
print()
print("Classification Report:\n ",metrics.classification_report(y_test, y_test_pred)
```

```
Performance metrics of the model for the XGBOOST Classifier - Hyperparameter Tuned
----------------------------------------------------------------------------------------------------
Accuracy:  0.8699867197875166
Precision Score:  0.8108493932905068
Recall Score:  0.614054054054054
AUROC Score:  0.9184356680624287

Confusion Matrix:
  [[10830   530]
 [ 1428  2272]]

Classification Report:
                 precision    recall  f1-score   support

           0       0.88      0.95      0.92     11360
           1       0.81      0.61      0.70      3700

   micro avg       0.87      0.87      0.87     15060
   macro avg       0.85      0.78      0.81     15060
weighted avg       0.87      0.87      0.86     15060
```

```
In [0]: XGB_best = pickle.dumps(XGB_grid.best_estimator_)
```

# 14 Choose the model for deployment

***We choose the hyperparameter tuned models because they have the better accuracy score even though all other average metrics(from classification report) are the same.***

```
In [116]: best_classifiers = [
                              ("Decision Tree - ",  DT_grid.best_estimator_),
                              ("Random Forest - ",  RF_grid.best_estimator_),
                              ("AdaBoost - ",  AB_grid.best_estimator_),
                              ("XGBoost - ", XGB_grid.best_estimator_)]
          accuracy_best = []
          precision_best = []
          recall_best = []
```

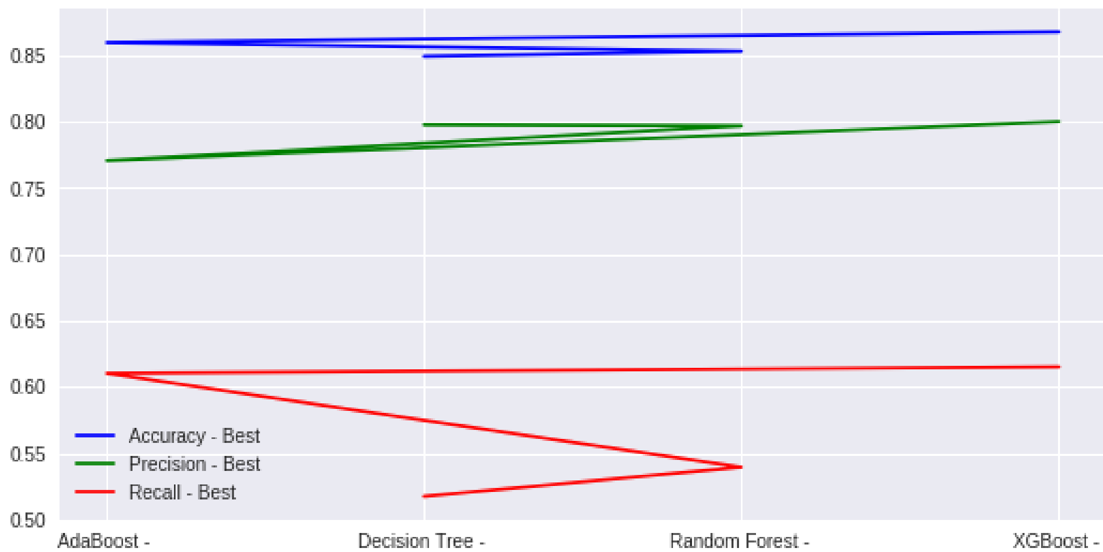
```
best_model_names = [i[0] for i in best_classifiers]

for clf in best_classifiers:
    clf[1].fit(x_train[imp_cols], y_train)
    y_test_pred= clf[1].predict(x_test[imp_cols])
    accuracy_best.append(metrics.accuracy_score(y_test, y_test_pred))
    precision_best.append(metrics.precision_score(y_test, y_test_pred))
    recall_best.append(metrics.recall_score(y_test, y_test_pred))
    print(clf[0])
    print("-"*100)
    print("Accuracy: ", metrics.accuracy_score(y_test, y_test_pred))
    print("Precision Score: ",metrics.precision_score(y_test, y_test_pred))
    print("Recall Score: ",metrics.recall_score(y_test, y_test_pred))
    print("AUROC Score: ",metrics.roc_auc_score(y_test,  y_test_pred_prob[:,1]))
    print()
    print("Confusion Matrix:  \n ",metrics.confusion_matrix(y_test, y_test_pred))
    print()
    print("Classification Report:\n ",metrics.classification_report(y_test, y_test_p
```

```
Decision Tree -
----------------------------------------------------------------------------------------------------
Accuracy:  0.849203187250996
Precision Score:  0.7975843398583924
Recall Score:  0.5175675675675676
AUROC Score:  0.9184356680624287

Confusion Matrix:
  [[10874   486]
 [ 1785  1915]]

Classification Report:
                precision    recall  f1-score   support

           0       0.86      0.96      0.91     11360
           1       0.80      0.52      0.63      3700

   micro avg       0.85      0.85      0.85     15060
   macro avg       0.83      0.74      0.77     15060
weighted avg       0.84      0.85      0.84     15060

Random Forest -
----------------------------------------------------------------------------------------------------
Accuracy:  0.853054448871182
Precision Score:  0.7968063872255489
Recall Score:  0.5394594594594595
AUROC Score:  0.9184356680624287

Confusion Matrix:
```

```
 [[10851   509]
 [ 1704  1996]]

Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.96      0.91     11360
           1       0.80      0.54      0.64      3700

   micro avg       0.85      0.85      0.85     15060
   macro avg       0.83      0.75      0.78     15060
weighted avg       0.85      0.85      0.84     15060

AdaBoost -
--------------------------------------------------------------------------------------------
Accuracy:  0.8596281540504648
Precision Score:  0.7706484641638225
Recall Score:  0.6102702702702703
AUROC Score:  0.9184356680624287

Confusion Matrix:
 [[10688   672]
 [ 1442  2258]]

Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.94      0.91     11360
           1       0.77      0.61      0.68      3700

   micro avg       0.86      0.86      0.86     15060
   macro avg       0.83      0.78      0.80     15060
weighted avg       0.85      0.86      0.85     15060

XGBoost -
--------------------------------------------------------------------------------------------
Accuracy:  0.8676626826029217
Precision Score:  0.8
Recall Score:  0.6151351351351352
AUROC Score:  0.9184356680624287

Confusion Matrix:
 [[10791   569]
 [ 1424  2276]]

Classification Report:
               precision    recall  f1-score   support
```

```
           0       0.88      0.95      0.92     11360
           1       0.80      0.62      0.70      3700

   micro avg       0.87      0.87      0.87     15060
   macro avg       0.84      0.78      0.81     15060
weighted avg       0.86      0.87      0.86     15060
```

In [117]:
```
plt.figure(figsize=(10,5))
plt.plot(best_model_names , accuracy_best, label = "Accuracy - Best",c = 'blue')
plt.plot(best_model_names , precision_best, label = "Precision - Best", c = 'green')
plt.plot(best_model_names , recall_best, label = "Recall - Best",c= 'red')
plt.legend()
plt.show()
```




***Clearly XGBoost offers better Accuracy, Precision and Recall when compared to the other Classifiers. Therefore, we choose it as our model. The following are the hyper-parameters of the model:***

In [118]:
```
XGB_grid.best_estimator_
```

Out[118]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.2, max_delta_step=0,
       max_depth=4, min_child_weight=1, missing=None, n_estimators=200,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=100, reg_alpha=0.1, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
```

In [0]:
```
# Saving the the chosen model in the pickle object
chosen_model = pickle.dumps(XGB_grid.best_estimator_)
```

In [120]: #To Load:

```
pickle.loads(chosen_model)
```

Out[120]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.2, max_delta_step=0,
       max_depth=4, min_child_weight=1, missing=nan, n_estimators=200,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=100, reg_alpha=0.1, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
```