Stat 517 Project #1

Name - **ABHISHEK KUMAR THAKUR**

Major - Materials Science & Engineering

Question 2: Predicting Income >$50k or not

In [31]:

#Loading and displaying the data  
import pandas as pd  
adult = pd.read\_csv("http://www.webpages.uidaho.edu/~stevel/Datasets/adult.csv")  
#adult

Solution

We will start solving this problem by import python tools for various data operations. Numpy, Pandas and random for data analysis. Seaborn and Matplotlib for data visualization. Various models like logistic regression, support vector machines etc for data modelling.

In [32]:

#importing data analysis packages  
import numpy as np  
import pandas as pd  
import random as rnd  
#importing data visualization packages  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline  
#importing machine learning packages  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC, LinearSVC  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.linear\_model import Perceptron  
from sklearn.linear\_model import SGDClassifier  
from sklearn.tree import DecisionTreeClassifier

Target variable

The target variable in the given dataset is "Salary". In the following cell, we are just seeing that how many person has Salary <=50K or Salary >50K

Cleaning the dataset (Imputation)

Now before modelling the data, we have to do cleaning operations such as finding any null value in the dataset, missing/misplaced data, data with '?', '\*' etc characters and so on. This step is very important because the analysis wil be good only when the data is clean.

In [33]:

adult.isnull().sum()

Out[33]:

Checking which columns has " ?" value

In [34]:

for value in ['workclass', 'education', 'marital\_status', 'occupation', 'relationship', 'race', 'sex', 'native\_country', 'salary']:  
 print (value,":", sum(adult[value] == ' ?'))

('workclass', ':', 1836)

('education', ':', 0)

('marital\_status', ':', 0)

('occupation', ':', 1843)

('relationship', ':', 0)

('race', ':', 0)

('sex', ':', 0)

('native\_country', ':', 583)

('salary', ':', 0)

We found that the columns "workclass", "occupation" and "native\_country" has " ?" value which we dont want. So we replace " ?" value with the most frequent occuring value in that column.

In [35]:

adult["workclass"].value\_counts() #This shows "Private" value has highest frequency in "workclass" column.  
adult["occupation"].value\_counts() #This shows "Prof-Speciality" value has highest frequency in "occupation" column.  
adult["native\_country"].value\_counts() #This shows "United-States" value has highest frequency in "native\_country" column.  
replace\_tool = {"workclass": {" ?": "Private"}, "occupation": {" ?": "Prof-specialty"}, "native\_country": {" ?": "United-States"}}  
adult.replace(replace\_tool, inplace = True)

Accounting for Categorical variables

The above analysis was done by taking continuous variables in account. It is more accurate to include categorical data into the analysis as well. For using the categorical data into our analysis, we first first transform the categorical data into numbers representing them. This is done by preprocessing the data using "get\_dummies" from pandas and then transforming it.

Below is the transformation of categorical data into continuous for further data analysis.

Preprocessing using pd.get\_dummies()

In [36]:

adult = pd.get\_dummies(adult, columns=["workclass", "education", "marital\_status", "occupation", "relationship", "race", "sex", "native\_country"])  
print (adult.shape)  
#adult.head(5)

(32561, 109)

Out[36]:

Pandas DataFrame

We are making 2 Pandas DataFrame: x\_adult and y\_adult. x\_adult consists of all the continuous columns from the given dataset. The continuous columns given are: age, fnlwgt, education\_num, capital\_gain, capital\_loss and hours\_per\_week. Our target column is: salary.

In [42]:

import pandas as pd  
y\_adult = pd.DataFrame([adult.salary]).T  
x\_adult = pd.DataFrame(adult)  
x\_adult = adult.drop(['salary'], axis = 1)

Training and Testing Datasets

In [43]:

from sklearn.cross\_validation import train\_test\_split  
xtrain, xtest, ytrain, ytest = train\_test\_split(x\_adult, y\_adult, random\_state = 1, test\_size = 0.25)

Gaussian Naive Bayes

In [44]:

from sklearn.naive\_bayes import GaussianNB  
from sklearn.metrics import accuracy\_score  
model = GaussianNB()  
model.fit(xtrain, ytrain)  
ymodel = model.predict(xtest)  
#The training score is reported 79.2%.  
acc\_gauss1 = round(model.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_gauss1))  
#The accuracy score is found by comparing ymodel and ytest (which is our ground truth).   
#The accuracy is around 80.2%.  
acc\_gauss = round(accuracy\_score(ytest, ymodel) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_gauss))

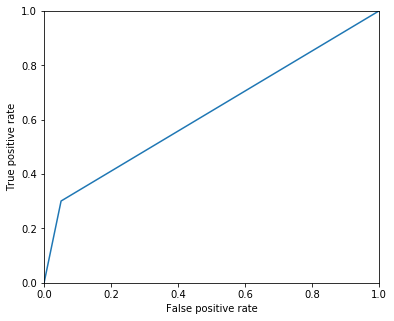
Training accuracy = 79.19

Testing accuracy = 80.28

In [45]:

#Area under ROC curve for Gaussian Naive Bayes model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_gnb = roc\_auc\_score(y1, y2)  
print (area\_gnb)

0.6250505182861089



K - Nearest Neighbor

In [46]:

from sklearn.neighbors import KNeighborsClassifier  
model = KNeighborsClassifier(n\_neighbors = 1)  
model.fit(xtrain, ytrain)  
#Evaluating the model on the test data set  
ymodel = model.predict(xtest)  
#The training score is reported 99.9%.  
acc\_knn1 = round(model.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_knn1))  
#The accuracy score is found by comparing ymodel and ytest (which is our ground truth).   
#The accuracy is around 73.84%.  
acc\_knn = round(accuracy\_score(ytest, ymodel) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_knn))

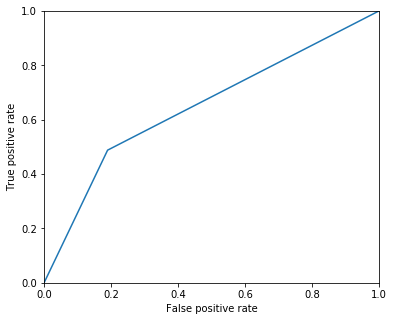
Training accuracy = 100.0

Testing accuracy = 73.75

In [47]:

#Area under ROC curve for K-Nearest Neighbor model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
from sklearn import preprocessing  
area\_knn = roc\_auc\_score(y1, y2)  
print (area\_knn)

0.649139777817945



Logistic Regression

In [48]:

logreg = LogisticRegression()  
logreg.fit(xtrain, ytrain)  
y\_pred = logreg.predict(xtest)  
#The training score is reported 79.39%.  
acc\_log1 = round(logreg.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_log1))  
#The accuracy is around 80.74%.  
acc\_log = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_log))

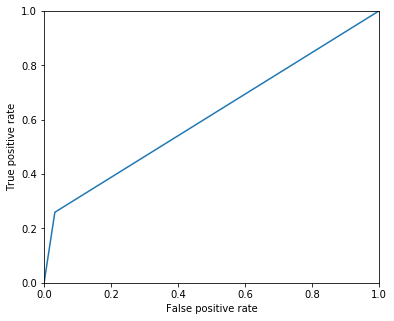
Training accuracy = 79.39

Testing accuracy = 80.79

In [49]:

#Area under ROC curve for Logistic Regression model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn import preprocessing  
area\_lr = roc\_auc\_score(y1, y2)  
print (area\_lr)

0.6136571665529844



Support Vector Machines

In [50]:

svc = SVC()  
svc.fit(xtrain, ytrain)  
y\_pred = svc.predict(xtest)  
#The training score is reported 99.2%.  
acc\_svc1 = round(svc.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_svc1))  
#The accuracy is around 77.26%.  
acc\_svc = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_svc))

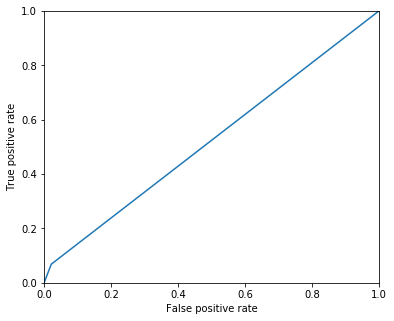
Training accuracy = 96.56

Testing accuracy = 77.3

In [51]:

#Area under ROC curve for Support Vector Machines model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn import preprocessing  
area\_svm = roc\_auc\_score(y1, y2)  
print (area\_svm)

0.5234878739712697



Stochastic Gradient Descent

In [52]:

sgd = SGDClassifier()  
sgd.fit(xtrain, ytrain)  
y\_pred = sgd.predict(xtest)  
#The training score is reported 78.5%.  
acc\_sgd1 = round(sgd.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_sgd1))  
#The accuracy is around 80.41%.  
acc\_sgd = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_sgd))

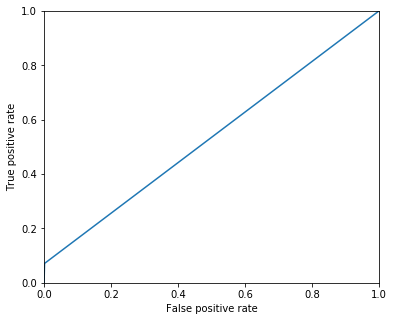
Training accuracy = 76.98

Testing accuracy = 78.92

In [53]:

#Area under ROC curve for Stochastic Gradient Descent model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
from sklearn import preprocessing  
area\_sgd = roc\_auc\_score(y1, y2)  
print (area\_sgd)

0.5349226157397456



Decision Tree

In [54]:

decision\_tree = DecisionTreeClassifier()  
decision\_tree.fit(xtrain, ytrain)  
y\_pred = decision\_tree.predict(xtest)  
#The training score is reported 99.9%.  
acc\_decision\_tree1 = round(decision\_tree.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_decision\_tree1))  
#The accuracy is around 77.44%.  
acc\_decision\_tree = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_decision\_tree))

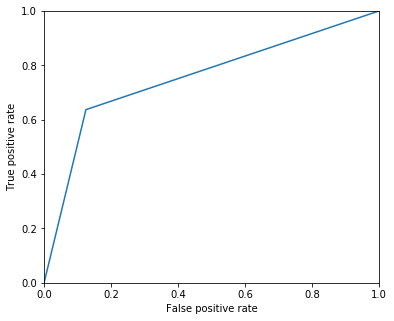
Training accuracy = 100.0

Testing accuracy = 82.13

In [55]:

#Area under ROC curve for Support Vector Machines model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_decision\_tree = roc\_auc\_score(y1, y2)  
print (area\_decision\_tree)

0.7560428588421113



Random Forest

In [56]:

forest = RandomForestClassifier(n\_estimators = 100, random\_state = 0)  
forest.fit(xtrain, ytrain)  
y\_pred = forest.predict(xtest)  
acc\_random\_forest1 = round(forest.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_random\_forest1))  
acc\_random\_forest = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_random\_forest))

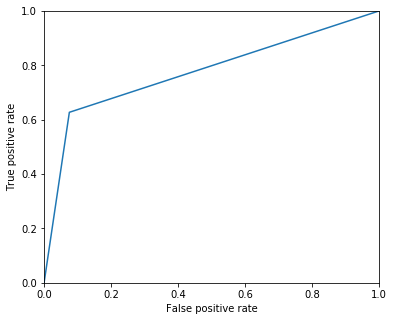
Training accuracy = 100.0

Testing accuracy = 85.74

In [57]:

#Area under ROC curve for Support Vector Machines model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_random\_forest = roc\_auc\_score(y1, y2)  
print (area\_random\_forest)

0.7758963510648417

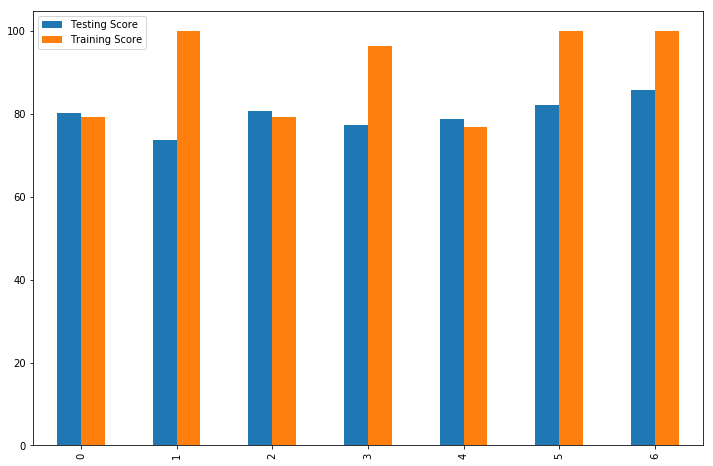


Summarizing Results

Out[61]:

|  | **Area under ROC curve** | **Model** | **Testing Score** | **Training Score** |
| --- | --- | --- | --- | --- |
| **0** | 0.625051 | Gaussian Naive Bayes | 80.28 | 79.19 |
| **1** | 0.649140 | KNN | 73.75 | 100.00 |
| **2** | 0.613657 | Logistic regression | 80.79 | 79.39 |
| **3** | 0.523488 | SVM | 77.30 | 96.56 |
| **4** | 0.534923 | Stochastic Gradient Decent | 78.92 | 76.98 |
| **5** | 0.756043 | Decision Tree | 82.13 | 100.00 |
| **6** | 0.775896 | Random Forest | 85.74 | 100.00 |

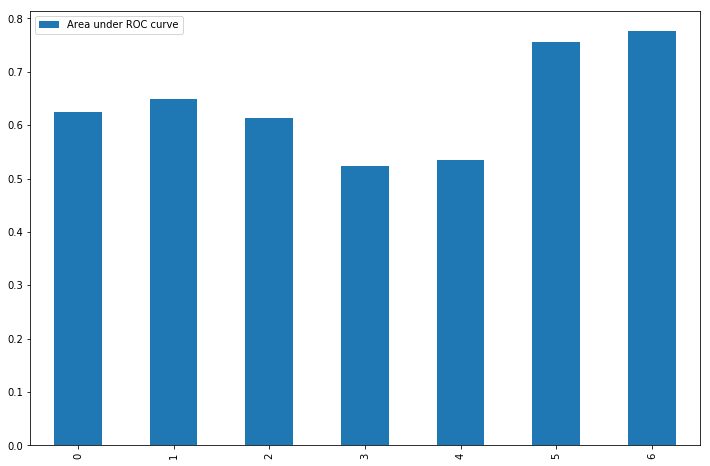
Reporting Scores



In [62]:

models\_print.plot(kind = 'bar', figsize = (12, 8))  
roc\_area.plot(kind = 'bar', figsize = (12, 8))

Out[62]:



Discussion

From the ROC curve presented for each model, we can see that the Random Forest model has the highest area under the ROC curve. The Random Forest model has also 85.74% and 100% testing and training scores respectively. So, for this problem, the Random Forest model is the best suited model and should be used to make future predictions. Second best model is the Decision Tree having ROC area 75.6%.

Question 3: Purchasing Insurance

In [63]:

import pandas as pd  
Caravan\_train = pd.read\_csv("http://www.webpages.uidaho.edu/~stevel/Datasets/Caravan\_train.csv")  
#Caravan\_train.head(5)

In [64]:

Caravan\_unk = pd.read\_csv("http://www.webpages.uidaho.edu/~stevel/Datasets/Caravan\_unk.csv")  
#Caravan\_unk

SOLUTION

Training and Testing Datasets

We identify the target and non-target attributes in our dataset. 'Purchase' is the target attribute (y\_caravan) while all the other attributes are non-target (x\_caravan). Further we split the whole dataset into training (xtrain and ytrain) and testing (xtest and ytest). We split 75% and 25% between training and testing datasets.

In [65]:

import pandas as pd  
x\_caravan = pd.DataFrame(Caravan\_train).T  
x\_caravan = x\_caravan.drop(['Purchase'])  
x\_caravan = x\_caravan.T  
print (x\_caravan.shape)  
y\_caravan = pd.DataFrame([Caravan\_train.Purchase]).T  
print (y\_caravan.shape)  
from sklearn.cross\_validation import train\_test\_split  
xtrain, xtest, ytrain, ytest = train\_test\_split(x\_caravan, y\_caravan, random\_state = 1, test\_size = 0.25)

Importing Data Modelling Tools

Now we import tools for further data operations. For data analysis, we used pandas, numpy and random. For visualization, we used matplotlib and seaborn. For modelling data, we used logistic regression, support vector machines and many more.

In [66]:

# data analysis and wrangling  
import pandas as pd  
import numpy as np  
import random as rnd  
# visualization  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline  
# machine learning  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC, LinearSVC  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.linear\_model import Perceptron  
from sklearn.linear\_model import SGDClassifier  
from sklearn.tree import DecisionTreeClassifier

Logistic Regression

Now since we have already splitted our dataset into training and testing, we will apply various models one by one over the dataset and record their training and testing accuracy.

In [67]:

#LOGISTIC REGRESSION  
from sklearn.metrics import accuracy\_score  
logreg = LogisticRegression()  
logreg.fit(xtrain, ytrain)  
y\_pred = logreg.predict(xtest)  
acc\_log1 = round(logreg.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_log1))  
acc\_log = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_log))

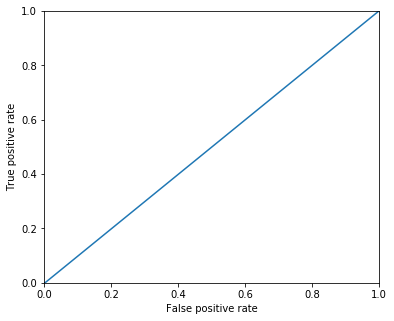
Training accuracy = 94.15

Testing accuracy = 92.92

In [68]:

#Area under ROC curve for Logistic Regression model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_lr = roc\_auc\_score(y1, y2)  
print (area\_lr)

0.4986098239110287



Support Vector Machines

In [69]:

#SUPPORT VECTOR MACHINES  
svc = SVC()  
svc.fit(xtrain, ytrain)  
y\_pred = svc.predict(xtest)  
acc\_svc1 = round(svc.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_svc1))  
acc\_svc = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_svc))

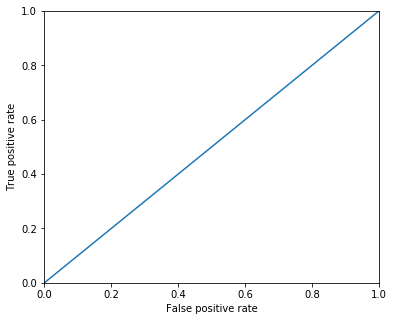
Training accuracy = 94.27

Testing accuracy = 93.09

In [70]:

#Area under ROC curve for Support Vector Machines model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_svm = roc\_auc\_score(y1, y2)  
print (area\_svm)

0.4995366079703429



K - Nearest Neighbor

In [71]:

#K-NEAREST NEIGHBOR  
knn = KNeighborsClassifier(n\_neighbors = 1)  
knn.fit(xtrain, ytrain)  
y\_pred = knn.predict(xtest)  
acc\_knn1 = round(knn.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_knn1))  
acc\_knn = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_knn))

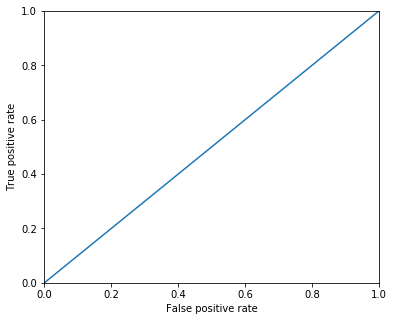
Training accuracy = 99.28

Testing accuracy = 89.12

In [72]:

#Area under ROC curve for KNN model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_knn = roc\_auc\_score(y1, y2)  
print (area\_knn)

0.5427435154444458



Perceptron

In [73]:

#PERCEPTRON  
perceptron = Perceptron()  
perceptron.fit(xtrain, ytrain)  
y\_pred = perceptron.predict(xtest)  
acc\_perceptron1 = round(perceptron.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_perceptron1))  
acc\_perceptron = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_perceptron))

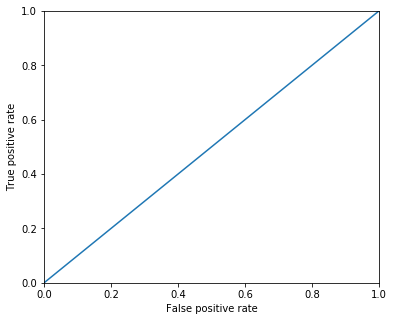
Training accuracy = 94.07

Testing accuracy = 93.18

In [74]:

#Area under ROC curve for Perceptron model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_perceptron = roc\_auc\_score(y1, y2)  
print (area\_perceptron)

0.5



Stochastic Gradient Descent

In [75]:

#STOCHASTIC GRADIENT DESCENT  
sgd = SGDClassifier()  
sgd.fit(xtrain, ytrain)  
y\_pred = sgd.predict(xtest)  
acc\_sgd1 = round(sgd.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_sgd1))  
acc\_sgd = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_sgd))

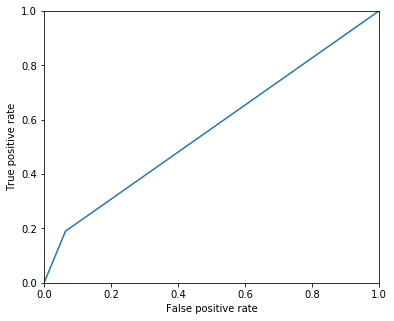
Training accuracy = 90.81

Testing accuracy = 88.51

In [76]:

#Area under ROC curve for SGD model  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_sgd = roc\_auc\_score(y1, y2)  
print (area\_sgd)

0.5629626588144204



Decision Tree

In [77]:

#DECISION TREE  
decision\_tree = DecisionTreeClassifier()  
decision\_tree.fit(xtrain, ytrain)  
y\_pred = decision\_tree.predict(xtest)  
acc\_decision\_tree1 = round(decision\_tree.score(xtrain, ytrain) \* 100, 2)  
print ('Training accuracy = {}'.format(acc\_decision\_tree1))  
acc\_decision\_tree = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_decision\_tree))

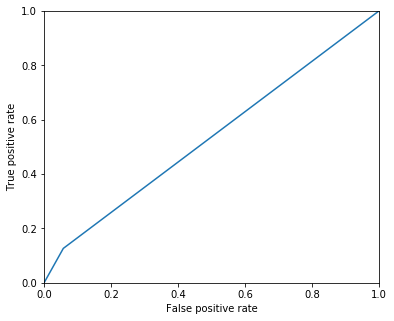
Training accuracy = 99.31

Testing accuracy = 88.69

In [78]:

#Area under ROC curve for decision tree model  
from sklearn import metrics  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_decision = roc\_auc\_score(y1, y2)  
print (area\_decision)

0.5345608334017667



Random Forest

In [79]:

forest = RandomForestClassifier(n\_estimators = 100, random\_state = 0)  
forest.fit(xtrain, ytrain)  
y\_pred = forest.predict(xtest)  
acc\_random\_forest1 = round(forest.score(xtrain, ytrain) \* 100, 2)print ('Training accuracy = {}'.format(acc\_random\_forest1))  
acc\_random\_forest = round(accuracy\_score(ytest, y\_pred) \* 100, 2)  
print ('Testing accuracy = {}'.format(acc\_random\_forest))

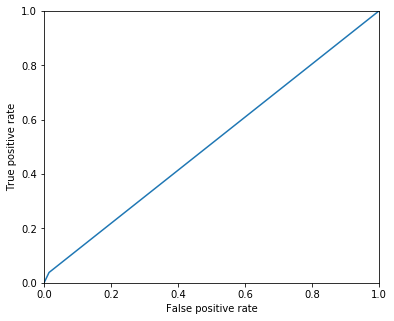
Training accuracy = 99.31

Testing accuracy = 92.06

In [80]:

#Area under ROC curve for Random Forest model  
from sklearn import metric  
plt.figure(figsize = (6, 5))  
fpr, tpr, thresholds = metrics.roc\_curve(y1, y2)  
plt.plot(fpr, tpr)  
from sklearn.metrics import roc\_auc\_score  
area\_random = roc\_auc\_score(y1, y2)  
print (area\_random)

0.5115730692976385



Summarizing Results

Now framing the various models used in the modelling process according to its training and testing accuracy and reporting the results in a tabulated form.

In [81]:

Out[81]:

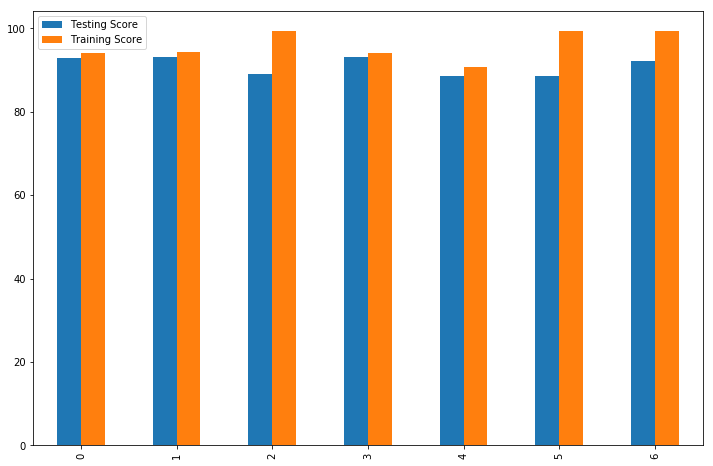
|  | **Area under ROC** | **Model** | **Testing Score** | **Training Score** |
| --- | --- | --- | --- | --- |
| **0** | 0.498610 | Logistic Regression | 92.92 | 94.15 |
| **1** | 0.499537 | Support Vector Machines | 93.09 | 94.27 |
| **2** | 0.542744 | KNN | 89.12 | 99.28 |
| **3** | 0.500000 | Perceptron | 93.18 | 94.07 |
| **4** | 0.562963 | Stochastic Gradient Decent | 88.51 | 90.81 |
| **5** | 0.534561 | Decision Tree | 88.69 | 99.31 |
| **6** | 0.511573 | Random Forest | 92.06 | 99.31 |

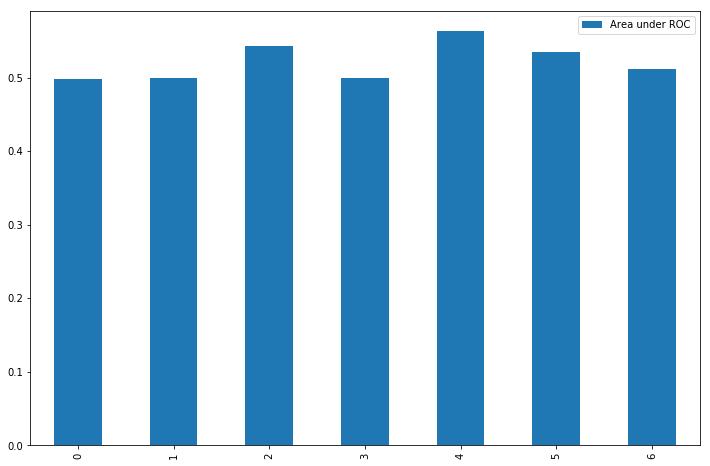
Presenting Scores and ROC areas

In [82]:

#Plotting the training and testing scores obtained using different models in a bar graph.  
models.plot(kind = 'bar', figsize = (12, 8))  
roc\_table.plot(kind = 'bar', figsize = (12, 8))

Out[82]:





Predicting probabilities

We now predict the probabilities of customers to show how likely they can buy the caravan insurance depending upon the Random forest model

In [84]:

from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import cross\_val\_predict  
probs = forest.predict\_proba(Caravan\_unk)[:,1]  
#print (probs[0:50])

Inserting probabilities in the dataset

In [85]:

probs = pd.DataFrame(probs)  
probs.columns = probs.columns +1  
probs.index = probs.index

In [86]:

new\_table = pd.merge(Caravan\_unk, probs, left\_index=True, right\_index=True)

In [87]:

sorted\_table = new\_table.sort\_values(by=[1L], ascending=False) #sort by most likely  
sorted\_pblty = sorted\_table[1L] #only the probabilities

Top 50 customers who are likely to buy insurance.

The table below shows the probablity of different customers (identified using their ID number) in decending order who are likely to buy the caravan insurance.

In [88]:

sorted\_pblty.head(n=50) #Just the top 50

Out[88]:

420 0.880000

136 0.730000

12 0.730000

1004 0.720000

454 0.710000

490 0.690000

162 0.650000

403 0.630000

236 0.630000

912 0.615000

624 0.590000

639 0.583000

319 0.550000

29 0.550000

678 0.545000

703 0.523000

738 0.511667

785 0.500000

1060 0.466976

108 0.460000

76 0.453333

2 0.450000

133 0.440000

805 0.440000

967 0.431333

868 0.430000

644 0.406667

520 0.400000

428 0.398333

651 0.396667

1 0.391000

120 0.380333

458 0.375333

332 0.375000

471 0.370000

123 0.370000

952 0.366667

1015 0.360000

852 0.345000

944 0.340000

811 0.339167

638 0.331667

416 0.330000

921 0.320000

1035 0.320000

277 0.320000

347 0.310000

389 0.310000

285 0.310000

329 0.300000

Name: 1, dtype: float64

**Thus, we predicted the probabilities of all the persons given in the dataset using the Random Forest model. We further sorted the whole database in descending order according to the predicted probabilities. Moreover, we have shown the top 50 persons (identified using their ID number or Row number) with high probabilities that the company should approach for selling the Caravan Policy. We can predict the same probabilities using other methods too which are being considered in the analysis above.**

**\*\*\***Thanks**\*\*\***