# Loading Package

# Customer Churn Prediction:

A Bank wants to take care of customer retention for its product: savings accounts. The bank wants you to identify customers likely to churn balances below the minimum balance. You have the customers information such as age, gender, demographics along with their transactions with the bank. Your task as a data scientist would be to predict the propensity to churn for each customer. Data Dictionary There are multiple variables in the dataset which can be cleanly divided into 3 categories:

# I. Demographic information about customers

• customer_id - Customer id • vintage - Vintage of the customer with the bank in a number of days • age - Age of customer • gender - Gender of customer • dependents - Number of dependents • occupation - Occupation of the customer • city - City of the customer (anonymized)

# II. Customer Bank Relationship

• customer_nw_category - Net worth of customer (3: Low 2: Medium 1: High) • branch_code - Branch Code for a customer account • days_since_last_transaction - No of Days Since Last Credit in Last 1 year

# III. Transactional Information

• current_balance - Balance as of today • previous_month_end_balance - End of Month Balance of previous month • average_monthly_balance_prevQ - Average monthly balances (AMB) in Previous Quarter • average_monthly_balance_prevQ2 - Average monthly balances (AMB) in previous to the previous quarter • current_month_credit - Total Credit Amount current month • previous_month_credit - Total Credit Amount previous month • current_month_debit - Total Debit Amount current month • previous_month_debit - Total Debit Amount previous month • current_month_balance - Average Balance of current month • previous_month_balance - Average Balance of previous month • churn - Average balance of customer falls below minimum balance in the next quarter (1/0)

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix, roc_curve, precision_score, recall_score, p
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

# Loading Data

In [3]:
```
\\SIGMA\Desktop\\Final project problem statement and dataset\\Final project problem statement and dataset\\churn_predicti
```

# Missing Values

In [4]:
```
pd.isnull(df).sum()
```

Out[4]:
```
customer_id                       0
vintage                           0
age                               0
gender                          525
dependents                     2463
occupation                       80
city                            803
customer_nw_category              0
branch_code                       0
days_since_last_transaction    3223
current_balance                   0
previous_month_end_balance        0
average_monthly_balance_prevQ     0
average_monthly_balance_prevQ2    0
current_month_credit              0
previous_month_credit             0
current_month_debit               0
previous_month_debit              0
current_month_balance             0
previous_month_balance            0
churn                             0
dtype: int64
```

Gender

In [5]:
```
df['gender'].value_counts()
```

Out[5]:
```
Male      16548
Female    11309
Name: gender, dtype: int64
```

In [6]:
```
#Convert Gender
dict_gender = {'Male': 1, 'Female':0}
df.replace({'gender': dict_gender}, inplace = True)

df['gender'] = df['gender'].fillna(-1)
```

In [7]:
```python
df['dependents'].value_counts()
```

Out[7]:
```
0.0     21435
2.0      2150
1.0      1395
3.0       701
4.0       179
5.0        41
6.0         8
7.0         3
36.0        1
52.0        1
25.0        1
9.0         1
50.0        1
32.0        1
8.0         1
Name: dependents, dtype: int64
```

In [8]:
```python
df['occupation'].value_counts()
```

Out[8]:
```
self_employed    17476
salaried          6704
student           2058
retired           2024
company             40
Name: occupation, dtype: int64
```

In [9]:
```python
df['dependents'] = df['dependents'].fillna(0)
df['occupation'] = df['occupation'].fillna('self_employed')
```

In [10]:
```python
df['city'] = df['city'].fillna(1020)
```

In [11]:
```python
df['days_since_last_transaction'] = df['days_since_last_transaction'].fillna(999)
```

In [12]:
```python
# Convert occupation to one hot encoded features
df = pd.concat([df,pd.get_dummies(df['occupation'],prefix = str('occupation'),prefix_sep='_')],axis = 1)
```

In [13]:
```
num_cols = ['customer_nw_category', 'current_balance',
            'previous_month_end_balance', 'average_monthly_balance_prevQ2', 'average_monthly_balance_prevQ',
            'current_month_credit','previous_month_credit', 'current_month_debit',
            'previous_month_debit','current_month_balance', 'previous_month_balance']
for i in num_cols:
    df[i] = np.log(df[i] + 17000)

std = StandardScaler()
scaled = std.fit_transform(df[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)
```

In [14]:
```
df_df_og = df.copy()
df = df.drop(columns = num_cols,axis = 1)
df = df.merge(scaled,left_index=True,right_index=True,how = "left")
```

In [15]:
```
y_all = df.churn
df = df.drop(['churn','customer_id','occupation'],axis = 1)
```

# baseline columns

In [16]:
```
baseline_cols = ['current_month_debit', 'previous_month_debit','current_balance','previous_month_end_balance','vintage'
                 ,'occupation_retired', 'occupation_salaried','occupation_self_employed', 'occupation_student']
```

In [17]:
```
df_baseline = df[baseline_cols]
```

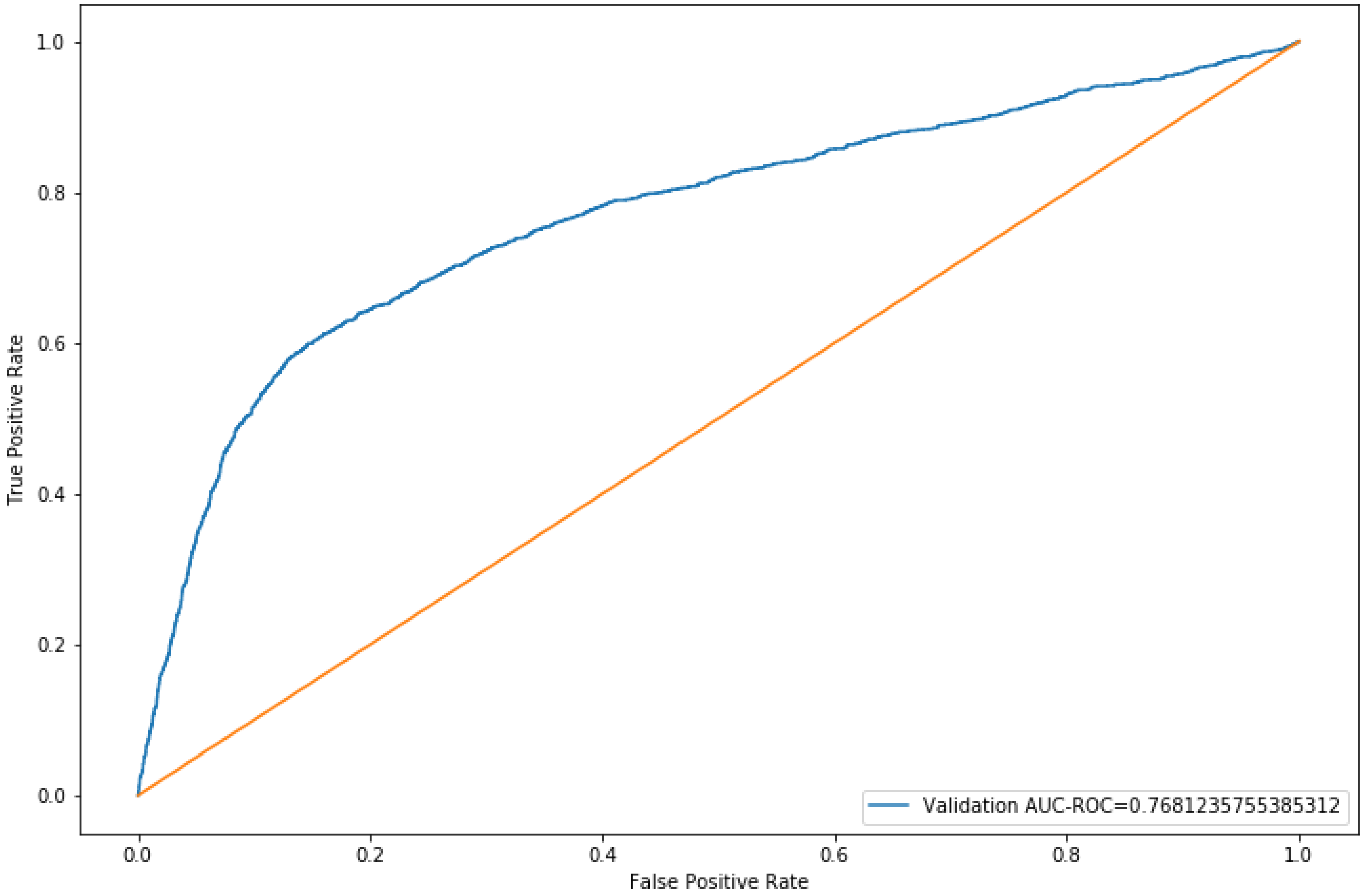
# Train Test Split to create a validation set

In [18]:
```
# Splitting the data into Train and Validation set
xtrain, xtest, ytrain, ytest = train_test_split(df_baseline,y_all,test_size=1/3, random_state=11, stratify = y_all)
```

In [19]:
```
model = LogisticRegression()
model.fit(xtrain,ytrain)
pred = model.predict_proba(xtest)[:,1]
```

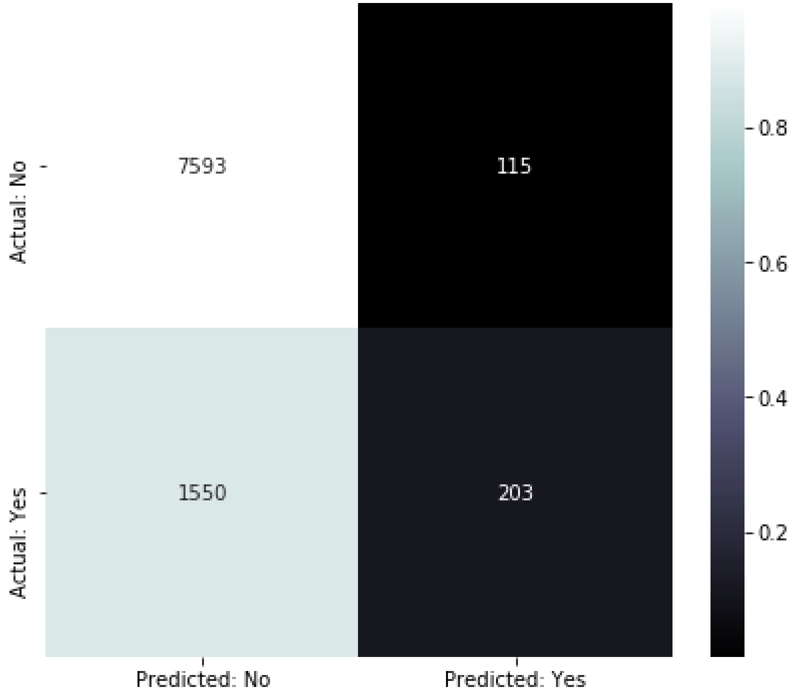
# AUC ROC Curve & Confusion Matrix

In [20]:
```
from sklearn.metrics import roc_curve
fpr, tpr, _ = roc_curve(ytest,pred)
auc = roc_auc_score(ytest, pred)
plt.figure(figsize=(12,8))
plt.plot(fpr,tpr,label="Validation AUC-ROC="+str(auc))
x = np.linspace(0, 1, 1000)
plt.plot(x, x, linestyle='-')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```



In [21]:
```python
# Confusion Matrix
pred_val = model.predict(xtest)
```

In [22]:
```python
label_preds = pred_val

cm = confusion_matrix(ytest,label_preds)


def plot_confusion_matrix(cm, normalized=True, cmap='bone'):
    plt.figure(figsize=[7, 6])
    norm_cm = cm
    if normalized:
        norm_cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        sns.heatmap(norm_cm, annot=cm, fmt='g', xticklabels=['Predicted: No','Predicted: Yes'], yticklabels=['Actual: No

plot_confusion_matrix(cm, ['No', 'Yes'])
```



In [23]:
```
# Recall Score
recall_score(ytest,pred_val)
```

Out[23]:
```
0.11580148317170565
```

# Cross Validation

In [24]:
```
def cv_score(ml_model, rstate = 12, thres = 0.5, cols = df.columns):
    i = 1
    cv_scores = []
    df1 = df.copy()
    df1 = df[cols]

    # 5 Fold cross validation stratified on the basis of target
    kf = StratifiedKFold(n_splits=5,random_state=rstate,shuffle=True)
    for df_index,test_index in kf.split(df1,y_all):
        print('\n{} of kfold {}'.format(i,kf.n_splits))
        xtr,xvl = df1.loc[df_index],df1.loc[test_index]
        ytr,yvl = y_all.loc[df_index],y_all.loc[test_index]

        # Define model for fitting on the training set for each fold
        model = ml_model
        model.fit(xtr, ytr)
        pred_probs = model.predict_proba(xvl)
        pp = []

        # Use threshold to define the classes based on probability values
        for j in pred_probs[:,1]:
            if j>thres:
                pp.append(1)
            else:
                pp.append(0)

        # Calculate scores for each fold and print
        pred_val = pp
        roc_score = roc_auc_score(yvl,pred_probs[:,1])
        recall = recall_score(yvl,pred_val)
        precision = precision_score(yvl,pred_val)
        sufix = ""
        msg = ""
        msg += "ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f} ".format(roc_score, recall,precision)
        print("{}".format(msg))

         # Save scores
        cv_scores.append(roc_score)
        i+=1
    return cv_scores
```

In [25]:
```
baseline_scores = cv_score(LogisticRegression(), cols = baseline_cols)
```

```
1 of kfold 5
ROC AUC Score: 0.7644836090843695, Recall Score: 0.0751, Precision Score: 0.5766

2 of kfold 5
ROC AUC Score: 0.7785366354948104, Recall Score: 0.0770, Precision Score: 0.6532

3 of kfold 5
ROC AUC Score: 0.7552602062967885, Recall Score: 0.1350, Precision Score: 0.6425

4 of kfold 5
ROC AUC Score: 0.758209770152749, Recall Score: 0.1169, Precision Score: 0.6508

5 of kfold 5
ROC AUC Score: 0.7653189015485415, Recall Score: 0.1131, Precision Score: 0.5640
```

In [26]:
```
all_feat_scores = cv_score(LogisticRegression())
```

```
1 of kfold 5
ROC AUC Score: 0.7329374165039565, Recall Score: 0.1093, Precision Score: 0.5066

2 of kfold 5
ROC AUC Score: 0.7680156201829207, Recall Score: 0.1968, Precision Score: 0.6809

3 of kfold 5
ROC AUC Score: 0.7393130731380004, Recall Score: 0.1683, Precision Score: 0.5728

4 of kfold 5
ROC AUC Score: 0.7328447133158789, Recall Score: 0.1207, Precision Score: 0.6019

5 of kfold 5
ROC AUC Score: 0.758855886628863, Recall Score: 0.1730, Precision Score: 0.5987
```

In [27]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [28]:
```
rf_all_features = cv_score(RandomForestClassifier(n_estimators=100, max_depth=8))
```

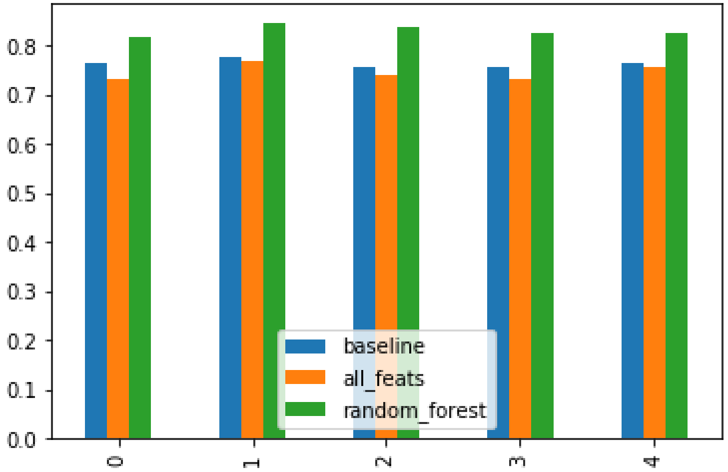
```
1 of kfold 5
ROC AUC Score: 0.8171341074915219, Recall Score: 0.3479, Precision Score: 0.7135

2 of kfold 5
ROC AUC Score: 0.8453017161648342, Recall Score: 0.3631, Precision Score: 0.7764

3 of kfold 5
ROC AUC Score: 0.8373638694462351, Recall Score: 0.3517, Precision Score: 0.7385

4 of kfold 5
ROC AUC Score: 0.828020157682845, Recall Score: 0.3593, Precision Score: 0.7354

5 of kfold 5
ROC AUC Score: 0.8242763618811424, Recall Score: 0.3527, Precision Score: 0.7289
```

# Comparison of Different model fold wise

In [29]:
```
results_df = pd.DataFrame({'baseline':baseline_scores, 'all_feats': all_feat_scores, 'random_forest': rf_all_features})
```

In [30]:
```
results_df.plot(y=["baseline", "all_feats", "random_forest"], kind="bar")
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x2706a8ef0c8>




In [ ]: