In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
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,precision_recall_curve
import warnings
warnings.simplefilter(action='ignore',category=FutureWarning)
warnings.simplefilter(action='ignore',category=UserWarning)
#pip install --upgrade sklearn
```

In [2]:
```
data = pd.read_csv('churn_prediction.csv')
```

In [3]:
```
data
```

Out[3]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category	branch_code	days_since_last_transaction	...	previous_month_end_balance	average_monthly_balance_prevQ	average_monthly_balance_prevQ2	current_month_credit	previous_month_credit	current_month_debit	previous_month_debit	current_month_balance	previous_month_bala
0	1	3135	66	Male	0.0	self_employed	187.0	2	755	224.0	...	1458.71	1458.71	1449.07	0.20	0.20	0.20	0.20	1458.71	145
1	2	310	35	Male	0.0	self_employed	NaN	2	3214	60.0	...	8704.66	7799.26	12419.41	0.56	0.56	5486.27	100.56	6496.78	878
2	4	2356	31	Male	0.0	salaried	146.0	2	41	NaN	...	5815.29	4910.17	2815.94	0.61	0.61	6046.73	259.23	5006.28	507
3	5	478	90	NaN	NaN	self_employed	1020.0	2	582	147.0	...	2291.91	2084.54	1006.54	0.47	0.47	0.47	2143.33	2291.91	166
4	6	2531	42	Male	2.0	self_employed	1494.0	3	388	58.0	...	1401.72	1643.31	1871.12	0.33	714.61	588.62	1538.06	1157.15	167
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28377	30297	1845	10	Female	0.0	student	1020.0	2	1207	70.0	...	1076.43	2282.19	2787.70	0.30	0.30	0.30	0.30	1076.43	107
28378	30298	4919	34	Female	0.0	self_employed	1046.0	2	223	14.0	...	4069.21	3668.83	3865.55	1.71	2.29	901.00	1014.07	3738.54	369
28379	30299	297	47	Male	0.0	salaried	1096.0	2	588	0.0	...	61017.55	53444.81	21925.81	4666.84	3883.06	168.23	71.80	61078.50	5756
28380	30300	2585	50	Male	3.0	self_employed	1219.0	3	274	NaN	...	1625.55	1683.20	1857.42	0.20	0.20	0.20	0.20	1625.55	162
28381	30301	2349	18	Male	0.0	student	1232.0	2	474	59.0	...	2821.34	3213.44	4447.45	0.11	7.44	714.40	1094.09	2402.62	326

28382 rows × 21 columns

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28382 entries, 0 to 28381
Data columns (total 21 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   customer_id                   28382 non-null  int64  
 1   vintage                       28382 non-null  int64  
 2   age                           28382 non-null  int64  
 3   gender                        27857 non-null  object 
 4   dependents                    25919 non-null  float64
 5   occupation                    28302 non-null  object 
 6   city                          27579 non-null  float64
 7   customer_nw_category          28382 non-null  int64  
 8   branch_code                   28382 non-null  int64  
 9   days_since_last_transaction   25159 non-null  float64
 10  current_balance               28382 non-null  float64
 11  previous_month_end_balance    28382 non-null  float64
 12  average_monthly_balance_prevQ  28382 non-null  float64
 13  average_monthly_balance_prevQ2  28382 non-null  float64
 14  current_month_credit          28382 non-null  float64
 15  previous_month_credit         28382 non-null  float64
 16  current_month_debit           28382 non-null  float64
 17  previous_month_debit          28382 non-null  float64
 18  current_month_balance         28382 non-null  float64
 19  previous_month_balance        28382 non-null  float64
 20  churn                         28382 non-null  int64  
dtypes: float64(13), int64(6), object(2)
memory usage: 4.5+ MB
```

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
data['gender'].value_counts()
```

Out[6]:
```
gender
Male      16548
Female    11309
Name: count, dtype: int64
```

In [7]:
```
dg = {'Male':1,'Female':0}
data.replace({'gender':dg},inplace=True)
data
```

Out[7]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category	branch_code	days_since_last_transaction	...	previous_month_end_balance	average_monthly_balance_prevQ	average_monthly_balance_prevQ2	current_month_credit	previous_month_credit	current_month_debit	previous_month_debit	current_month_balance	previous_month_bala
0	1	3135	66	1.0	0.0	self_employed	187.0	2	755	224.0	...	1458.71	1458.71	1449.07	0.20	0.20	0.20	0.20	1458.71	145
1	2	310	35	1.0	0.0	self_employed	NaN	2	3214	60.0	...	8704.66	7799.26	12419.41	0.56	0.56	5486.27	100.56	6496.78	878
2	4	2356	31	1.0	0.0	salaried	146.0	2	41	NaN	...	5815.29	4910.17	2815.94	0.61	0.61	6046.73	259.23	5006.28	507
3	5	478	90	NaN	NaN	self_employed	1020.0	2	582	147.0	...	2291.91	2084.54	1006.54	0.47	0.47	0.47	2143.33	2291.91	166
4	6	2531	42	1.0	2.0	self_employed	1494.0	3	388	58.0	...	1401.72	1643.31	1871.12	0.33	714.61	588.62	1538.06	1157.15	167
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28377	30297	1845	10	0.0	0.0	student	1020.0	2	1207	70.0	...	1076.43	2282.19	2787.70	0.30	0.30	0.30	0.30	1076.43	107
28378	30298	4919	34	0.0	0.0	self_employed	1046.0	2	223	14.0	...	4069.21	3668.83	3865.55	1.71	2.29	901.00	1014.07	3738.54	369
28379	30299	297	47	1.0	0.0	salaried	1096.0	2	588	0.0	...	61017.55	53444.81	21925.81	4666.84	3883.06	168.23	71.80	61078.50	5756
28380	30300	2585	50	1.0	3.0	self_employed	1219.0	3	274	NaN	...	1625.55	1683.20	1857.42	0.20	0.20	0.20	0.20	1625.55	162
28381	30301	2349	18	1.0	0.0	student	1232.0	2	474	59.0	...	2821.34	3213.44	4447.45	0.11	7.44	714.40	1094.09	2402.62	326

28382 rows × 21 columns

In [8]:
```
data['gender']=data['gender'].fillna(-1)
data['dependents'] = data['dependents'].fillna(0)
data['occupation'] = data['occupation'].fillna('self_employed')
data['city'] = data['city'].fillna(1020)
data['days_since_last_transaction'] = data['days_since_last_transaction'].fillna(999)
```

In [9]:
```
pd.isnull(data).sum()
```

Out[9]:
```
customer_id                     0
vintage                         0
age                             0
gender                          0
dependents                      0
occupation                      0
city                            0
customer_nw_category            0
branch_code                     0
days_since_last_transaction     0
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
```

In [10]:
```
data['occupation'].unique()
```

Out[10]:
```
array(['self_employed', 'salaried', 'retired', 'student', 'company'],
      dtype=object)
```

In [11]:
```
data = pd.concat([data,pd.get_dummies(data['occupation'],prefix=str('occupation'),prefix_sep='_')],axis=1)
data
```

Out[11]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category	branch_code	days_since_last_transaction	...	current_month_debit	previous_month_debit	current_month_balance	previous_month_balance	churn	occupation_company	occupation_retired	occupation_salaried	occupation_self_employed	occupation_student
0	1	3135	66	1.0	0.0	self_employed	187.0	2	755	224.0	...	0.20	0.20	1458.71	1458.71	0	False	False	False	True	False
1	2	310	35	1.0	0.0	self_employed	1020.0	2	3214	60.0	...	5486.27	100.56	6496.78	8787.61	0	False	False	False	True	False
2	4	2356	31	1.0	0.0	salaried	146.0	2	41	999.0	...	6046.73	259.23	5006.28	5070.14	0	False	False	True	False	False
3	5	478	90	-1.0	0.0	self_employed	1020.0	2	582	147.0	...	0.47	2143.33	2291.91	1669.79	1	False	False	False	True	False
4	6	2531	42	1.0	2.0	self_employed	1494.0	3	388	58.0	...	588.62	1538.06	1157.15	1677.16	1	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28377	30297	1845	10	0.0	0.0	student	1020.0	2	1207	70.0	...	0.30	0.30	1076.43	1076.43	0	False	False	False	False	True
28378	30298	4919	34	0.0	0.0	self_employed	1046.0	2	223	14.0	...	901.00	1014.07	3738.54	3690.32	0	False	False	False	True	False
28379	30299	297	47	1.0	0.0	salaried	1096.0	2	588	0.0	...	168.23	71.80	61078.50	57564.24	1	False	False	True	False	False
28380	30300	2585	50	1.0	3.0	self_employed	1219.0	3	274	999.0	...	0.20	0.20	1625.55	1625.55	0	False	False	False	True	False
28381	30301	2349	18	1.0	0.0	student	1232.0	2	474	59.0	...	714.40	1094.09	2402.62	3260.58	1	False	False	False	False	True

28382 rows × 26 columns

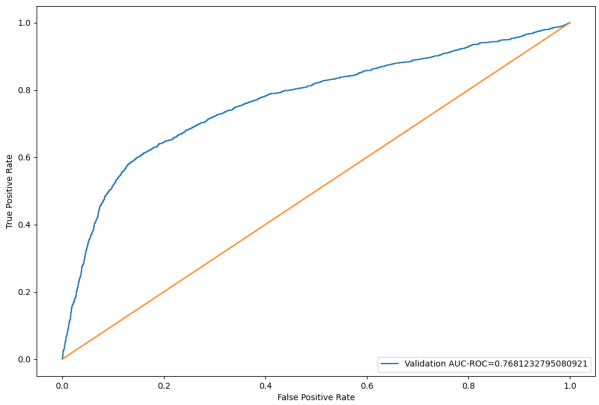
In [12]:
```
nc = ['customer_nw_category','current_balance','previous_month_end_balance','average_monthly_balance_prevQ2','average_monthly_balance_prevQ','current_month_credit','previous_month_credit','current_month_debit','previous_month_debit','current_month_balance', 'previous_month_balance']
for i in nc:
 data[i] = np.log(data[i] + 17000)
std = StandardScaler()
scaled = std.fit_transform(data[nc])
scaled = pd.DataFrame(scaled,columns = nc)
```

In [13]:
```
data_og = data.copy()
data = data.drop(columns = nc , axis =1)
data = data.merge(scaled, left_index =True, right_index =True, how='left')
```

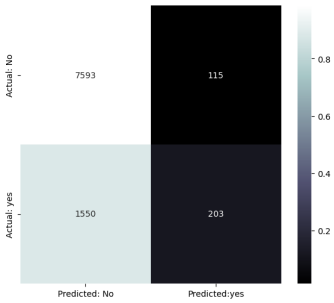
In [14]:
```
y_all = data.churn
data = data.drop(['churn','customer_id','occupation'],axis = 1)
```

In [15]:
```
baseline_cols =['current_month_debit', 'previous_month_debit','current_balance','previous_month_end_balance','vintage','occupation_retired','occupation_salaried','occupation_self_employed','occupation_student']
a_baseline = data[baseline_cols]
xtrain, xtest, ytrain, ytest =train_test_split(a_baseline,y_all,test_size = 1/3 , random_state=11,stratify = y_all)
model = LogisticRegression()
model.fit(xtrain,ytrain)
pred = model.predict_proba(xtest)[:,1]
```

In [16]:
```
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 [17]:
```
pred_val = model.predict(xtest)
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','Actual: yes'],cmap=cmap)
plot_confusion_matrix(cm,['No','Yes'])
```



In [18]:
```
recall_score(ytest,pred_val)
```

Out[18]:
```
0.11580148317170565
```

In [19]:
```
def cv_score(ml_model, rstate = 12, thres = 0.5, cols = data.columns):
    i = 1
    cv_scores = []
    b = data.copy()
    b = data[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(b,y_all):
        print('\n{} of kfold {}'.format(i,kf.n_splits))
        xtr,xvl = b.loc[df_index],b.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} "
        print("{}".format(msg))
# Save scores
        cv_scores.append(roc_score)
        i+=1
    return cv_scores
```

In [20]:
```
baseline_scores = cv_score(LogisticRegression(), cols = baseline_cols)
```

```
1 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

2 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

3 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

4 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

5 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}
```

In [21]:
```
all_feat_scores = cv_score(LogisticRegression())
```

```
1 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

2 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

3 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

4 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

5 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}
```

In [22]:
```
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt
# Create the RFE object and rank each feature
model = LogisticRegression()
rfe = RFE(estimator=model, n_features_to_select=1, step=1)
rfe.fit(data, y_all)
```

Out[22]:
```
RFE
estimator: LogisticRegression
LogisticRegression
```

In [23]:
```
ranking_a = pd.DataFrame()
ranking_a['Feature_name'] = data.columns
ranking_a['Rank'] = rfe.ranking_
```

In [24]:
```
ranked = ranking_a.sort_values(by=['Rank'])
```

In [25]:
```
ranked
```

Out[25]:

	Feature_name	Rank
13	current_balance	1
16	average_monthly_balance_prevQ	2
7	occupation_company	3
15	average_monthly_balance_prevQ2	4
21	current_month_balance	5
22	previous_month_balance	6
19	current_month_debit	7
10	occupation_self_employed	8
9	occupation_salaried	9
11	occupation_student	10
20	previous_month_debit	11
17	current_month_credit	12
12	customer_nw_category	13
2	gender	14
8	occupation_retired	15
3	dependents	16
14	previous_month_end_balance	17
1	age	18
18	previous_month_credit	19
6	days_since_last_transaction	20
4	city	21
0	vintage	22
5	branch_code	23

In [26]:
```
rfe_top_10_scores = cv_score(LogisticRegression(), cols = ranked['Feature_name'][:10].values)
```

```
1 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

2 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

3 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

4 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

5 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}
```

In [27]:
```
cv_score(LogisticRegression(), cols = ranked['Feature_name'][:10].values,thres=0.14)
```

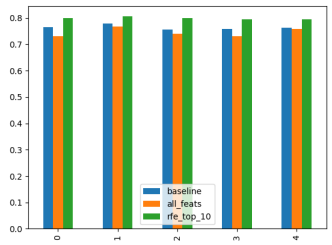
```
1 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

2 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

3 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

4 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}

5 of kfold 5
ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f}
```

Out[27]:
```
[0.7986881101633954,
 0.8050442914397288,
 0.7985130070256687,
 0.7935095616193245,
 0.7942222838028076]
```

In [28]:
```
results_a = pd.DataFrame({'baseline':baseline_scores, 'all_feats': all_feat_scores, 'rfe_top_10': rfe_top_10_scores})
```

In [29]:
```
results_a.plot(y=["baseline", "all_feats", "rfe_top_10"], kind="bar")
```

Out[29]:
```
<Axes: >
```



In [ ]: