Credit Evaluation Notebook

Import All required modules

In [1]:

```
import pandas as pd
import sklearn as sk
import seaborn as sns
from scipy import stats
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, log_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoosti
ngClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.model_selection import train_test_split
```

Load tarining data and read file

In [2]:

```
crdit_raw = pd.read_csv('training.csv')
```

In [3]:

```
crdit_raw.describe()
```

Out[3]:

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	496819.831400	65853.355259	20.962621	0.329419	662.548800
std	287391.314157	31093.369592	18.841121	0.223299	90.549985
min	244.000000	2.054543	-2.140206	0.001781	500.000000
25%	245172.500000	39165.786086	6.150577	0.156965	600.000000
50%	495734.000000	76903.628763	11.913366	0.272263	655.000000
75%	745475.250000	91032.514900	32.238914	0.459890	727.000000
max	999870.000000	150538.809704	81.759632	1.144357	849.000000

In [4]:

```
crdit_raw.head()
```

Out[4]:

	customer_id	demographic_slice	country_reg	ad_exp	est_income	hold_bal	p
0	713782	AX03efs	W	N	33407.901749	3.000000	0
1	515901	AX03efs	E	N	19927.533533	20.257927	0
2	95166	AX03efs	W	Y	51222.470997	4.000000	0
3	425557	AX03efs	E	Y	67211.587467	18.653631	0
4	624581	AX03efs	W	N	20093.342158	4.000000	0

Now that we have an overview, dealing with each each column data and null values

In [5]:

```
crdit_raw.isnull().any()
```

Out[5]:

```
customer_id          False
demographic_slice    False
country_reg          False
ad_exp               False
est_income           False
hold_bal             False
pref_cust_prob       False
imp_cscore           False
RiskScore            False
imp_crediteval       False
axio_score           False
card_offer           False
dtype: bool
```

In [6]:

```
crdit_raw.columns[crdit_raw.isnull().any()]
```

Out[6]:

```
Index([], dtype='object')
```

There are no missing values, so no need to compute aor deal with missing values

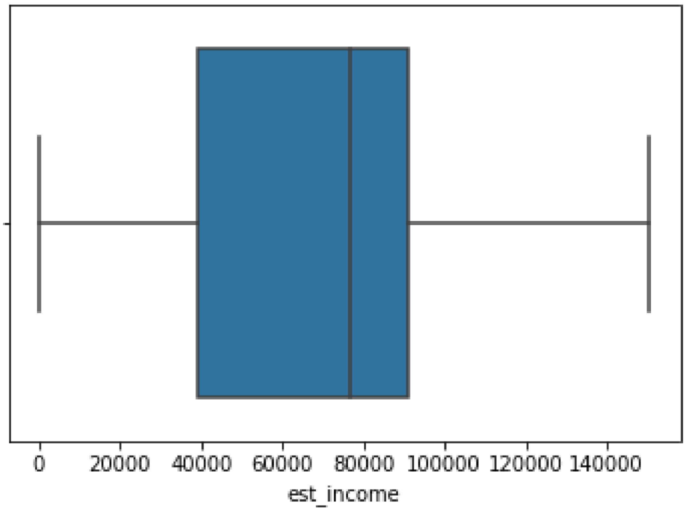
We need to remove outliers if any from each column. This will ensure the results are not skewed

In [7]:

```
sns.boxplot(x=crdit_raw['est_income'])
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f72acef390>
```

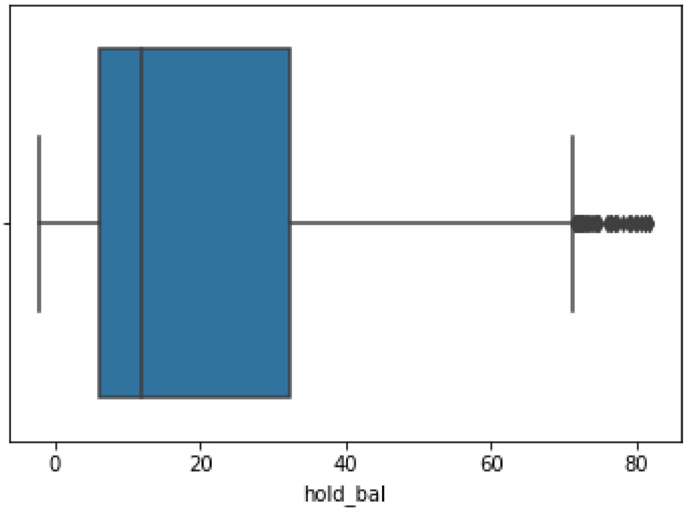


In [8]:

```
sns.boxplot(x=crdit_raw['hold_bal'])
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f72ad93f98>
```

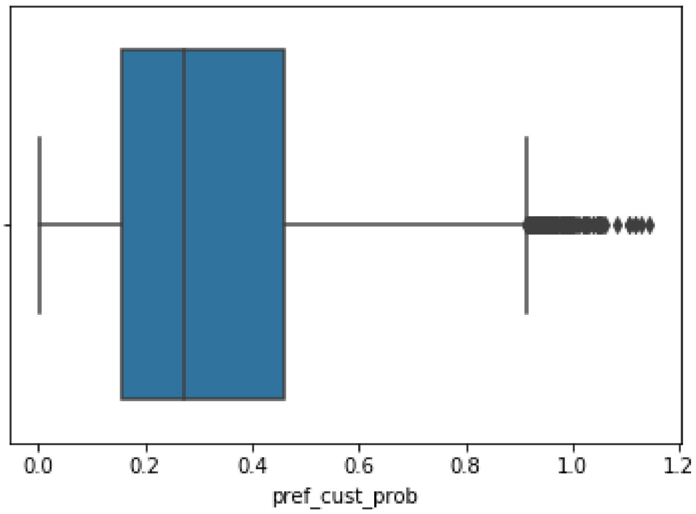


In [9]:

```
sns.boxplot(x=crdit_raw['pref_cust_prob'])
```

Out[9]:

<matplotlib.axes._subplots.AxesSubplot at 0x1f72b08a940>

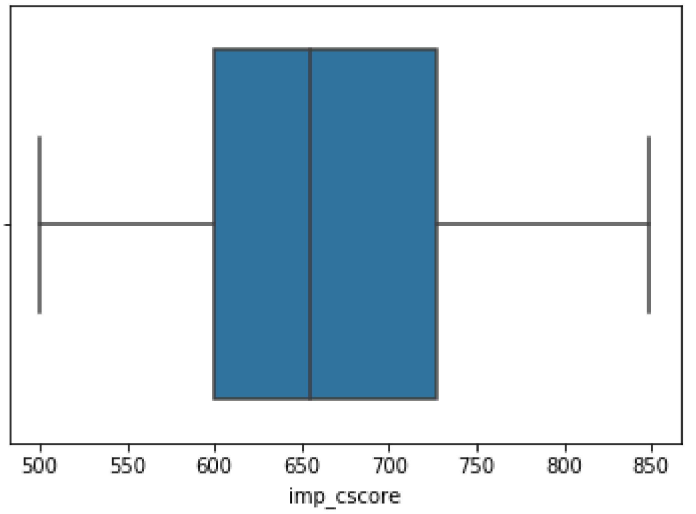


In [10]:

```
sns.boxplot(x=crdit_raw['imp_cscore'])
```

Out[10]:

<matplotlib.axes._subplots.AxesSubplot at 0x1f72b0db358>

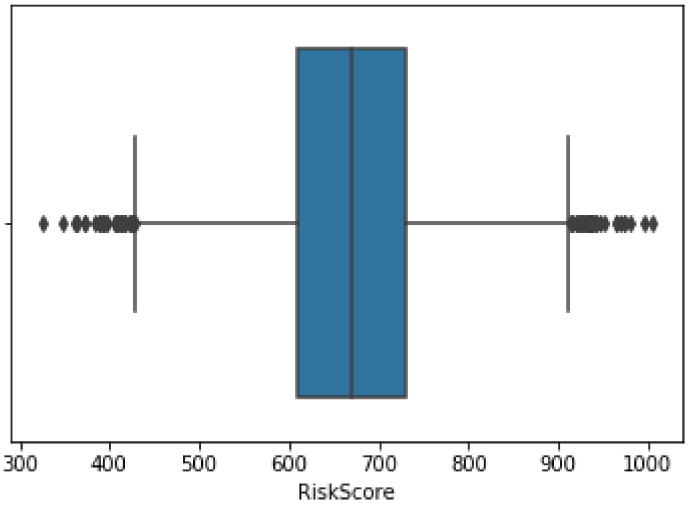


In [11]:

```
sns.boxplot(x=crdit_raw['RiskScore'])
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f72b141860>
```

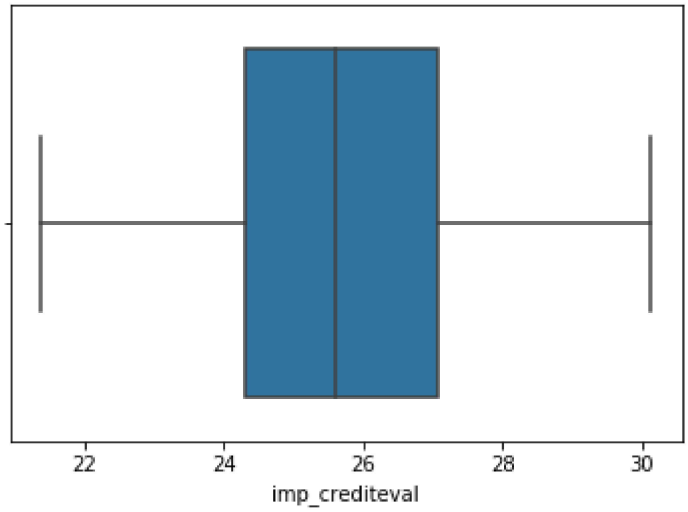


In [12]:

```
sns.boxplot(x=crdit_raw['imp_crediteval'])
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f72b192e10>
```

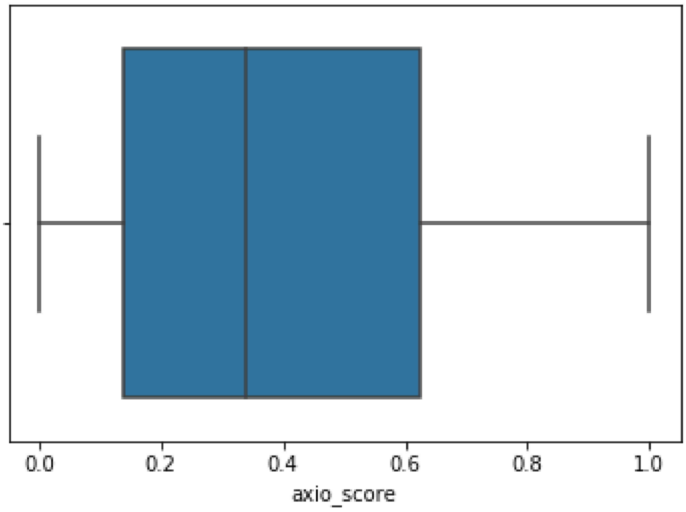


In [13]:

```
sns.boxplot(x=crdit_raw['axio_score'])
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f72b0fb6a0>
```



We see there are outliers for the columns RiskScore, imp_cscore and pref_cust_prob . So we will need to remove those from our training data

Box plot use the IQR method to display data and outliers(shape of the data) but in order to be get a list of identified outlier, we will need to use the mathematical formula and retrieve the outlier data.

In [14]:

```
risk_score_Q1 = crdit_raw['RiskScore'].quantile(0.25)
risk_score_Q3 = crdit_raw['RiskScore'].quantile(0.75)
risk_score_IQR = risk_score_Q3 - risk_score_Q1
risk_score_min=risk_score_Q1 - (risk_score_IQR*1.5)
risk_score_max=risk_score_Q3 + (risk_score_IQR*1.5)
print("risk_score_IQR = "+str(risk_score_IQR))


imp_cscore_Q1 = crdit_raw['imp_cscore'].quantile(0.25)
imp_cscore_Q3 = crdit_raw['imp_cscore'].quantile(0.75)
imp_cscore_IQR = imp_cscore_Q3 - imp_cscore_Q1
imp_cscore_min=imp_cscore_Q1 - (imp_cscore_IQR*1.5)
imp_cscore_max=imp_cscore_Q3 + (imp_cscore_IQR*1.5)
print("imp_cscore_IQR = "+str(imp_cscore_IQR))


pref_cust_prob_Q1 = crdit_raw['pref_cust_prob'].quantile(0.25)
pref_cust_prob_Q3 = crdit_raw['pref_cust_prob'].quantile(0.75)
pref_cust_prob_IQR = pref_cust_prob_Q3 - pref_cust_prob_Q1
pref_cust_prob_min=pref_cust_prob_Q1 - (pref_cust_prob_IQR*1.5)
pref_cust_prob_max=pref_cust_prob_Q3 + (pref_cust_prob_IQR*1.5)
print("pref_cust_prob_IQR = "+str(pref_cust_prob_IQR))
print()
print ("risk_score_min ="+str(risk_score_min) +" ::::: "+"risk_score_max ="+str(risk_sc
ore_max))
print ("imp_cscore_min ="+str(imp_cscore_min) +" ::::: "+"imp_cscore_max ="+str(imp_csc
ore_max))
print ("pref_cust_prob_min ="+str(pref_cust_prob_min) +" ::::: "+"pref_cust_prob_max ="
+str(pref_cust_prob_max))
```

```
risk_score_IQR = 121.25380374369252
imp_cscore_IQR = 127.0
pref_cust_prob_IQR = 0.30292516581097373

risk_score_min =427.35047536172647 ::::: risk_score_max =912.3656903364965
imp_cscore_min =409.5 ::::: imp_cscore_max =917.5
pref_cust_prob_min =-0.2974231051514481 ::::: pref_cust_prob_max =0.914277
5580924468
```

The above gives range of each IQR. Then we remove all th outliers from all the 3 columsn

In [15]:

```
credit_no_outliers = crdit_raw.loc[(crdit_raw['RiskScore'] > risk_score_min) & (crdit_r
aw['RiskScore'] < risk_score_max)]
```

In [16]:

```
credit_no_outliers.describe()
```

Out[16]:

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	R
count	9926.000000	9926.000000	9926.000000	9926.000000	9926.000000	992
mean	497043.859057	65870.077277	20.980726	0.329679	662.437236	669
std	287413.244562	31106.451817	18.845011	0.223421	90.507851	87.
min	244.000000	2.054543	-2.140206	0.001781	500.000000	428
25%	245155.500000	39148.645696	6.154399	0.157192	600.000000	609
50%	496893.500000	76926.365168	11.968306	0.273050	655.000000	669
75%	745531.500000	91082.947495	32.258555	0.459951	727.000000	729
max	999870.000000	150538.809704	81.759632	1.144357	849.000000	911

In [17]:

```
credit_no_outliers = credit_no_outliers.loc[(credit_no_outliers['imp_cscore'] > imp_csc
ore_min) & (credit_no_outliers['imp_cscore'] < imp_cscore_max)]
```

In [18]:

```
credit_no_outliers.describe()
```

Out[18]:

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	
count	9926.000000	9926.000000	9926.000000	9926.000000	9926.000000	9
mean	497043.859057	65870.077277	20.980726	0.329679	662.437236	6
std	287413.244562	31106.451817	18.845011	0.223421	90.507851	8
min	244.000000	2.054543	-2.140206	0.001781	500.000000	4
25%	245155.500000	39148.645696	6.154399	0.157192	600.000000	6
50%	496893.500000	76926.365168	11.968306	0.273050	655.000000	6
75%	745531.500000	91082.947495	32.258555	0.459951	727.000000	7
max	999870.000000	150538.809704	81.759632	1.144357	849.000000	9

In [19]:

```
credit_no_outliers = credit_no_outliers.loc[(crdit_raw['pref_cust_prob'] > pref_cust_pr
ob_min) & (credit_no_outliers['pref_cust_prob'] < pref_cust_prob_max)]
```

In [20]:

```
credit_no_outliers.describe()
```

Out[20]:

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	R
count	9784.000000	9784.000000	9784.000000	9784.000000	9784.000000	978
mean	497751.719338	65885.609009	20.978336	0.320319	662.473835	669
std	287420.840698	31106.356834	18.848666	0.210902	90.561729	87.
min	244.000000	2.054543	-2.140206	0.001781	500.000000	428
25%	245930.250000	39181.333217	6.150577	0.155504	600.000000	609
50%	497614.000000	76955.172160	11.922362	0.268817	655.000000	669
75%	745984.500000	91137.966126	32.287644	0.449736	727.000000	729
max	999870.000000	150538.809704	81.759632	0.914256	849.000000	911

Removed roughly 280 outliers from the dataset

In [21]:

```
credit_no_outliers.head()
```

Out[21]:

	customer_id	demographic_slice	country_reg	ad_exp	est_income	hold_bal
0	713782	AX03efs	W	N	33407.901749	3.000000
1	515901	AX03efs	E	N	19927.533533	20.257927
2	95166	AX03efs	W	Y	51222.470997	4.000000
3	425557	AX03efs	E	Y	67211.587467	18.653631
4	624581	AX03efs	W	N	20093.342158	4.000000

Now we have 3 columns with string values., demographic_slice, country_reg, ad_exp. We will drop columns which are irrelevant or convert them into numerical discrete values.

Checking for dsitinct demographic_slice

In [22]:

```
credit_no_outliers.demographic_slice.unique()
```

Out[22]:

```
array(['AX03efs', 'BWEsk45', 'CARDIF2', 'DERS3w5'], dtype=object)
```

In [23]:

```
credit_no_outliers.country_reg.unique()
```

Out[23]:

```
array(['W', 'E'], dtype=object)
```

In [24]:

```
credit_no_outliers.ad_exp.unique()
credit_no_outliers['ad_exp'] = credit_no_outliers['ad_exp'].map({'Y': 1, 'N': 0})
credit_no_outliers.head()
```

Out[24]:

	customer_id	demographic_slice	country_reg	ad_exp	est_income	hold_bal
0	713782	AX03efs	W	0	33407.901749	3.000000
1	515901	AX03efs	E	0	19927.533533	20.257927
2	95166	AX03efs	W	1	51222.470997	4.000000
3	425557	AX03efs	E	1	67211.587467	18.653631
4	624581	AX03efs	W	0	20093.342158	4.000000

In [25]:

```
credit_sklearn = credit_no_outliers.copy()
lb_make = LabelEncoder()
credit_sklearn['demographic_slice_encoded'] = lb_make.fit_transform(credit_no_outliers[
'demographic_slice'])
credit_sklearn['country_reg_encoded'] = lb_make.fit_transform(credit_no_outliers['count
ry_reg'])
credit_sklearn=credit_sklearn.drop(['demographic_slice', 'country_reg'], axis=1)
credit_sklearn.head()
```

Out[25]:

	customer_id	ad_exp	est_income	hold_bal	pref_cust_prob	imp_cscore	Ri
0	713782	0	33407.901749	3.000000	0.531112	619	50
1	515901	0	19927.533533	20.257927	0.297439	527	82
2	95166	1	51222.470997	4.000000	0.018463	606	58
3	425557	1	67211.587467	18.653631	0.089344	585	63
4	624581	0	20093.342158	4.000000	0.094948	567	63

Dropped the string column after encoding them to numerical values.

Data preprocessing complete. Now we split the data into training and testing data and run various classifer algorithm to get the best model

In [26]:

```
X=credit_sklearn.drop(['card_offer'],axis=1)
y=credit_sklearn['card_offer']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=
42)
```

In [27]:

```
classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="rbf", C=0.025, probability=True),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    GradientBoostingClassifier(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    QuadraticDiscriminantAnalysis()]

# Logging for Visual Comparison
log_cols=["Classifier", "Accuracy", "Log Loss"]
log = pd.DataFrame(columns=log_cols)

for clf in classifiers:
    clf.fit(X_train, y_train)
    name = clf.__class__.__name__

    print("="*30)
    print(name)

    print('****Results****')
    train_predictions = clf.predict(X_test)
    acc = accuracy_score(y_test, train_predictions)
    print("Accuracy: {:.4%}".format(acc))

    train_predictions = clf.predict_proba(X_test)
    ll = log_loss(y_test, train_predictions)
    print("Log Loss: {}".format(ll))

    log_entry = pd.DataFrame([[name, acc*100, ll]], columns=log_cols)
    log = log.append(log_entry)

print("="*30)
```

```
==============================
KNeighborsClassifier
****Results****
Accuracy: 82.0378%
Log Loss: 2.775744407843222
==============================
SVC
****Results****
Accuracy: 86.0948%
Log Loss: 0.4033340696906456
==============================
DecisionTreeClassifier
****Results****
Accuracy: 96.5934%
Log Loss: 1.176607433707085
==============================
RandomForestClassifier
****Results****
Accuracy: 97.0889%
Log Loss: 0.11013802545089481
==============================
AdaBoostClassifier
****Results****
Accuracy: 98.4825%
Log Loss: 0.5510582747398198
==============================
GradientBoostingClassifier
****Results****
Accuracy: 98.1418%
Log Loss: 0.055039685247836506
==============================
GaussianNB
****Results****
Accuracy: 86.0948%
Log Loss: 0.36517707423865137
==============================
LinearDiscriminantAnalysis
****Results****
Accuracy: 95.2307%
Log Loss: 0.11363859589961241
==============================
QuadraticDiscriminantAnalysis
****Results****
Accuracy: 96.1288%
Log Loss: 0.08619483375426466
==============================
```

We have a winner here with AdaBoost having the highest accuracy and lowest Loss. So we will use that model to predict the future details.

Before doing that we need to apply each and every transformation on our predication data

In [28]:

```
pred = pd.read_csv('test.csv')
```

In [29]:

```
pred.head()
```

Out[29]:

	customer_id	demographic_slice	country_reg	ad_exp	est_income	hold_bal
0	596723	AX03efs	W	N	26323.092380	3.000000
1	841834	AX03efs	E	Y	67374.621650	17.861095
2	402401	AX03efs	E	N	1728.369713	21.604489
3	734431	AX03efs	E	Y	15814.210260	22.058403
4	739547	AX03efs	W	Y	45233.588190	1.000000

In [30]:

```
pred.isnull().any()
```

Out[30]:

```
customer_id          False
demographic_slice    False
country_reg          False
ad_exp               False
est_income           False
hold_bal             False
pref_cust_prob       False
imp_cscore           False
RiskScore            False
imp_crediteval       False
axio_score           False
card_offer            True
dtype: bool
```

In [31]:

```
pred= pred.drop(['card_offer'],axis=1)
```

In [32]:

```
pred.demographic_slice.unique()
```

Out[32]:

```
array(['AX03efs', 'BWEsk45', 'CARDIF2', 'DERS3w5'], dtype=object)
```

In [33]:

```
pred.country_reg.unique()
```

Out[33]:

```
array(['W', 'E'], dtype=object)
```

In [34]:

```
pred.ad_exp.unique()
```

Out[34]:

```
array(['N', 'Y'], dtype=object)
```

In [35]:

```
pred['ad_exp'] = pred['ad_exp'].map({'Y': 1, 'N': 0})
pred.head()
```

Out[35]:

	customer_id	demographic_slice	country_reg	ad_exp	est_income	hold_bal
0	596723	AX03efs	W	0	26323.092380	3.000000
1	841834	AX03efs	E	1	67374.621650	17.861095
2	402401	AX03efs	E	0	1728.369713	21.604489
3	734431	AX03efs	E	1	15814.210260	22.058403
4	739547	AX03efs	W	1	45233.588190	1.000000

In [36]:

```
pred_sklearn = pred.copy()
lb_make = LabelEncoder()
pred_sklearn['demographic_slice_encoded'] = lb_make.fit_transform(pred['demographic_sli
ce'])
pred_sklearn['country_reg_encoded'] = lb_make.fit_transform(pred['country_reg'])
pred_sklearn=pred_sklearn.drop(['demographic_slice', 'country_reg'], axis=1)
pred_sklearn.head()
```

Out[36]:

	customer_id	ad_exp	est_income	hold_bal	pref_cust_prob	imp_cscore	Ri
0	596723	0	26323.092380	3.000000	0.461364	603	505
1	841834	1	67374.621650	17.861095	0.473517	650	466
2	402401	0	1728.369713	21.604489	0.486220	606	603
3	734431	1	15814.210260	22.058403	0.462249	530	747
4	739547	1	45233.588190	1.000000	0.541660	640	704

Now the above data is ready to be applied to our learning model

In [37]:

```
ada = AdaBoostClassifier();
model = ada.fit(X_train, y_train)
```

In [38]:

```
y_pred=model.predict(pred_sklearn)
y_pred
```

Out[38]:

```
array([False, False, False, ...,  True,  True, False])
```

Now that we have the prediction ready. Writing the prediction to a csv file named ds4.csv

In [39]:

```
def write_to_csv(predction):
    submission = pd.DataFrame()
    submission['Id'] = pred_sklearn.customer_id
    submission['Card_Offer']=predction
    submission.to_csv('ds4.csv', index=False)
```

In [40]:

```
write_to_csv(y_pred)
```

End of NoteBook. Refer ds4.csv to find predictions along with customer id