In [150]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [151]:

```
df = pd.read_csv('C:/TreeVista/Test 1.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
customer_id          10000 non-null int64
demographic_slice    10000 non-null object
country_reg          10000 non-null object
ad_exp               10000 non-null object
est_income           10000 non-null float64
hold_bal             10000 non-null float64
pref_cust_prob       10000 non-null float64
imp_cscore           10000 non-null int64
RiskScore            10000 non-null float64
imp_crediteval       10000 non-null float64
axio_score           10000 non-null float64
card_offer           10000 non-null bool
dtypes: bool(1), float64(6), int64(2), object(3)
memory usage: 869.2+ KB
```

In [152]:

```
df.columns
```

Out[152]:

```
Index(['customer_id', 'demographic_slice', 'country_reg', 'ad_exp',
       'est_income', 'hold_bal', 'pref_cust_prob', 'imp_cscore', 'RiskScore',
       'imp_crediteval', 'axio_score', 'card_offer'],
      dtype='object')
```

In [153]:

```
df.describe()
```

Out[153]:

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	RiskScore	imp_crediteval	axio_s
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00
mean	496819.831400	65853.355259	20.962621	0.329419	662.548800	670.042869	25.692162	0.393211
std	287391.314157	31093.369592	18.841121	0.223299	90.549985	89.965854	1.889274	0.288243
min	244.000000	2.054543	-2.140206	0.001781	500.000000	324.436647	21.363123	-0.00005
25%	245172.500000	39165.786086	6.150577	0.156965	600.000000	609.231181	24.295435	0.139424
50%	495734.000000	76903.628763	11.913366	0.272263	655.000000	669.493442	25.611903	0.337841
75%	745475.250000	91032.514900	32.238914	0.459890	727.000000	730.484985	27.062519	0.624886
max	999870.000000	150538.809704	81.759632	1.144357	849.000000	1004.497869	30.131214	1.000000

Cleaning the data

In [154]:

```
df.isna().any()
```

Out[154]:

```
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 [155]:

```
# Create a correlation matrix
corr_metrics = df.corr()
corr_metrics.style.background_gradient()
```

Out[155]:

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	RiskScore	imp_crediteval	axio_sc
customer_id	1	0.00492495	0.00685613	-0.00571561	0.00910654	-0.00420723	0.0056651	0.00076
est_income	0.00492495	1	0.0103311	0.00868931	0.00351446	0.00453002	-0.000173806	-0.0046
hold_bal	0.00685613	0.0103311	1	0.00182511	0.269361	0.0139309	0.256165	0.00015
pref_cust_prob	-0.00571561	0.00868931	0.00182511	1	-0.0114992	-0.0125698	-0.013733	-0.0203
imp_cscore	0.00910654	0.00351446	0.269361	-0.0114992	1	-0.00480859	0.926908	0.00536
RiskScore	-0.00420723	0.00453002	0.0139309	-0.0125698	-0.00480859	1	-0.00435891	0.00065
imp_crediteval	0.0056651	-0.000173806	0.256165	-0.013733	0.926908	-0.00435891	1	0.00672
axio_score	0.000761587	-0.00467278	0.000158183	-0.0203884	0.00536186	0.000654177	0.00672548	1
card_offer	-0.00680401	0.27753	0.0561434	0.630311	0.0228125	-0.00360271	0.0170552	-0.0153

In [158]:

```
# it seems feature 'imp_cscore' is correlated to 'imp_crediteval' , it's good to drop one of them
for the sake of curse of DIm..

df_updated= df.drop(columns=['imp_crediteval'])
```

In [159]:

```
df_updated.describe()
```

Out[159]:

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	RiskScore	axio_score
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	496819.831400	65853.355259	20.962621	0.329419	662.548800	670.042869	0.393211
std	287391.314157	31093.369592	18.841121	0.223299	90.549985	89.965854	0.288243
min	244.000000	2.054543	-2.140206	0.001781	500.000000	324.436647	-0.000052
25%	245172.500000	39165.786086	6.150577	0.156965	600.000000	609.231181	0.139424
50%	495734.000000	76903.628763	11.913366	0.272263	655.000000	669.493442	0.337841

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	RiskScore	axio_score
75%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
max	999870.000000	150538.809704	81.759632	1.144357	849.000000	1004.497869	1.000000

In [160]:

```
cat_df = df_updated.select_dtypes(include=['object']).copy()
```

In [162]:

```
cat_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
demographic_slice    10000 non-null object
country_reg          10000 non-null object
ad_exp               10000 non-null object
dtypes: object(3)
memory usage: 234.5+ KB
```

In [163]:

```
cat_df.head()
```

Out[163]:

	demographic_slice	country_reg	ad_exp
0	AX03efs	W	N
1	AX03efs	E	N
2	AX03efs	W	Y
3	AX03efs	E	Y
4	AX03efs	W	N

In [164]:

```
df_updated['demographic_slice']= df_updated['demographic_slice'].astype('category')
```

In [165]:

```
#Label Encoding
df_updated['demographic_slice']= df_updated['demographic_slice'].cat.codes
```

In [ ]:

```
#similary for rest of the other
```

In [166]:

```
df_updated['country_reg']= df_updated['country_reg'].astype('category')
df_updated['ad_exp']= df_updated['ad_exp'].astype('category')
df_updated['card_offer']= df_updated['card_offer'].astype('category')
```

In [167]:

```
#Label Encoding
df_updated['country_reg']= df_updated['country_reg'].cat.codes
df_updated['ad_exp']= df_updated['ad_exp'].cat.codes
df_updated['card_offer']= df_updated['card_offer'].cat.codes
```

In [168]:

```
df_updated.head(10)
```

Out[168]:

	customer_id	demographic_slice	country_reg	ad_exp	est_income	hold_bal	pref_cust_prob	imp_cscore	RiskScor
0	713782	0	1	0	33407.901749	3.000000	0.531112	619	503.24902
1	515901	0	0	0	19927.533533	20.257927	0.297439	527	820.10814
2	95166	0	1	1	51222.470997	4.000000	0.018463	606	586.60579
3	425557	0	0	1	67211.587467	18.653631	0.089344	585	634.70198
4	624581	0	1	0	20093.342158	4.000000	0.094948	567	631.94997
5	721691	0	0	0	73896.096129	12.906641	0.656848	560	809.33396
6	269858	0	1	0	73609.404135	3.000000	0.137818	620	697.30816
7	219196	0	1	0	57619.668582	9.000000	0.367879	658	668.07547
8	413020	0	1	1	49282.620299	0.000000	0.182079	519	656.11159
9	174424	0	1	0	57173.061392	6.000000	0.288242	645	547.76711

In [169]:

```
df_updated.columns
```

Out[169]:

```
Index(['customer_id', 'demographic_slice', 'country_reg', 'ad_exp',
       'est_income', 'hold_bal', 'pref_cust_prob', 'imp_cscore', 'RiskScore',
       'axio_score', 'card_offer'],
      dtype='object')
```

In [170]:

```
offer= df_updated['card_offer']
users= df_updated['customer_id']

df_final= df_updated.drop(columns=['card_offer','customer_id'])
```

In [172]:

```
#splitting the data in training and test DataSets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train,  y_test = train_test_split(df_final, offer, test_size=0.2, random_state=0
)
```

In [173]:

```
#feature scaling

from sklearn.preprocessing  import StandardScaler
sc_X = StandardScaler()
X_train2 = pd.DataFrame(sc_X.fit_transform(X_train))
X_test2 = pd.DataFrame(sc_X.fit_transform(X_test))

X_train2.columns= X_train.columns.values
X_test2.columns= X_test.columns.values
X_train2.index= X_train.index.values
X_test2.index= X_test.index.values

X_train= X_train2
X_test= X_test2

#X_test= np.squeeze(sc_X.transform(X_test.values.reshape(-1, 1)))
```

In [ ]:

```
#ready for modelling

#Logistic Reg
```

In [174]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

Out[174]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [175]:

```
#predicting Test Set

y_pred_LR= classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score , recall_sc
ore
acc= accuracy_score (y_test, y_pred_LR)
prec= precision_score(y_test,y_pred_LR)
rec= recall_score (y_test, y_pred_LR)
f1= f1_score(y_test,y_pred_LR)
```

In [176]:

```
pd.DataFrame([['Log Regr', acc, prec, rec, f1]], columns= ['Model','Accuracy','Precision','Recall',
'F1'])
```

Out[176]:

	Model	Accuracy	Precision	Recall	F1
0	Log Regr	0.9665	0.905724	0.873377	0.889256

In [ ]:

```
#lets check with different model
```

In [177]:

```
from sklearn.svm import SVC
classifier = SVC(random_state=0, kernel='linear')
classifier.fit(X_train, y_train)

#predicting Test Set

y_pred_SVC= classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score , recall_sc
ore
acc= accuracy_score (y_test, y_pred_SVC)
prec= precision_score(y_test,y_pred_SVC)
rec= recall_score (y_test, y_pred_SVC)
f1= f1_score(y_test,y_pred_SVC)

pd.DataFrame([['SVM', acc, prec, rec, f1]], columns= ['Model','Accuracy','Precision','Recall','F1']
)
```

Out[177]:

	Model	Accuracy	Precision	Recall	F1
0	SVM	0.966	0.902685	0.873377	0.887789

Conculsuiion from above results:

1. RBF kernel worked better than linear kernel but not good than that of Log Regr

In [178]:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state=0, n_estimators=100, criterion = 'entropy')
classifier.fit(X_train, y_train)

#predicting Test Set

y_pred_RC= classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score , recall_sc
ore
acc= accuracy_score (y_test, y_pred_RC)
prec= precision_score(y_test,y_pred_RC)
rec= recall_score (y_test, y_pred_RC)
f1= f1_score(y_test,y_pred_RC)

pd.DataFrame([['Random Forest with 100 trees', acc, prec, rec, f1]], columns= ['Model','Accuracy',
'Precision','Recall','F1'])
#print(svm_results)
```

Out[178]:

	Model	Accuracy	Precision	Recall	F1
0	Random Forest with 100 trees	0.9755	0.94198	0.896104	0.918469

Final Conclusion

RFC performed best among the rest of the classifiers with an accuracy of 97.55 percent and F1 score 0.918 on the training

first dataset named "Test1.csv" , please note , no dataset ds1 & ds4 was provided in mail communication"

I will proceed with the RFC model for the test2.csv data