Credit Card Application Approval Model Assignment EDA

Questions & Answers

What is the proportion of females in the applicant customer base ?

In [10]:
```
count_gender = credit_app.groupBy('CODE_GENDER').count().orderBy('count')
count_gender = count_gender.withColumn('proportion %', fun.col('count')*100/fun.sum('count').over(Window.partitionBy()))
count_gender.show()
```

▸ Spark Job Progress

```
+-----------+------+-----------------+
|CODE_GENDER| count|     proportion %|
+-----------+------+-----------------+
|          M|144117|32.86163486160294|
|          F|294440|67.13836513839706|
+-----------+------+-----------------+
```

Based on above result 67.13 % is the ans.

Is house ownership higher among male applicants vs female applicants

In [11]:
```
credit_app_pivoted = credit_app.crosstab('NAME_HOUSING_TYPE','CODE_GENDER')
credit_app_pivoted.show()

credit_app_pivoted = credit_app_pivoted.withColumn('prop_F', fun.col('F')/fun.sum('F').over(Window.partitionBy()))
credit_app_pivoted = credit_app_pivoted.withColumn('prop_M', fun.col('M')/fun.sum('M').over(Window.partitionBy()))

credit_app_pivoted.show()
```

▸ Spark Job Progress

```
+-----------------------------+------+------+
|NAME_HOUSING_TYPE_CODE_GENDER|     F|     M|
+-----------------------------+------+------+
|             Rented apartment|  3284|  2690|
|          Municipal apartment| 10019|  4195|
|             Office apartment|  2122|  1800|
|            House / apartment|267607|126224|
|              Co-op apartment|   862|   677|
|                 With parents| 10546|  8531|
+-----------------------------+------+------+
```

```
+-----------------------------+------+------+--------------------+--------------------+
|NAME_HOUSING_TYPE_CODE_GENDER|     F|     M|              prop_F|              prop_M|
+-----------------------------+------+------+--------------------+--------------------+
|             Rented apartment|  3284|  2690|0.011153375900013585|0.018665389926240485|
|          Municipal apartment| 10019|  4195| 0.03402730607254449|0.029108293955605515|
|             Office apartment|  2122|  1800|0.007206901236245...|0.012489851995253856|
|            House / apartment|267607|126224|  0.9088676810216003|  0.8758439323605126|
|              Co-op apartment|   862|   677|0.002927591359869583|0.004697572111548...|
|                 With parents| 10546|  8531| 0.03581714440972694| 0.05919495965083925|
+-----------------------------+------+------+--------------------+--------------------+
```

We observe that 267607 females owns house as compared to male its 126224. The data shows house ownership is higher in Females.

Is there any correlation between income levels and education level?

In [12]:
```
# Lower secondary = 0, secondary / secondary spl = 1, Incomplete higher = 2, Higher education = 3, Academic degree = 4
credit_app = credit_app.withColumn("NAME_EDUCATION_TYPE_cat", when(col("NAME_EDUCATION_TYPE")=='Lower secondary', 0).when(col("NA

corl_data = credit_app.select('AMT_INCOME_TOTAL', 'NAME_EDUCATION_TYPE_cat')

col_names = corl_data.columns
features = corl_data.rdd.map(lambda row: row[0:])
corl_mat=Statistics.corr(features, method="pearson")
corl_d = pd.DataFrame(corl_mat)
corl_d.index, corl_d.columns = col_names, col_names
print(corl_d.to_string())
```

▸ Spark Job Progress

```
                         AMT_INCOME_TOTAL  NAME_EDUCATION_TYPE_cat
AMT_INCOME_TOTAL                 1.000000                 0.221403
NAME_EDUCATION_TYPE_cat          0.221403                 1.000000
```

Data shows positive corelation between income levels and education level

What is the average and median salary of the applicant base?

In [13]:
```
# Computing the mean
credit_app.select('AMT_INCOME_TOTAL').describe().show()
# Computing the median
credit_app.approxQuantile("AMT_INCOME_TOTAL", [0.5], 0)
```

▸ Spark Job Progress

```
+-------+------------------+
|summary|  AMT_INCOME_TOTAL|
+-------+------------------+
|  count|            438557|
|   mean|187524.28600950394|
| stddev|110086.85306622952|
|    min|           26100.0|
|    max|         6750000.0|
+-------+------------------+

[160780.5]
```

Average mean salary of applicant base is 160780.5

Do people owning cars have higher probability of bad customers ?

In [14]:
```
target_car_customers = merged_Credit_app_records.groupby('FLAG_OWN_CAR', 'target').count().orderBy('FLAG_OWN_CAR')

pivoted = merged_Credit_app_records.groupby('FLAG_OWN_CAR').pivot("target", ['1', '0']).count()

pivoted.show()

row_count = sum(col(x) for x in ['1', '0'])

adjusted = [(col(y) / row_count).alias(y) for y in ['1', '0']]

pivoted.select(col("FLAG_OWN_CAR"), *adjusted).show()
```

▸ Spark Job Progress

```
+------------+---+-----+
|FLAG_OWN_CAR|  1|    0|
+------------+---+-----+
|           Y|177|10339|
|           N|245|14373|
+------------+---+-----+

+------------+--------------------+------------------+
|FLAG_OWN_CAR|                   1|                 0|
+------------+--------------------+------------------+
|           Y| 0.01683149486496767|0.9831685051350323|
|           N|0.016760158708441646|0.9832398412915584|
+------------+--------------------+------------------+
```

We observe that people owning card has higher number of defaulters although the proportion wise when we compare to entire space the difference is marginal.

Do people living on rent have a higher proportion of bad customers as compared to the rest of the population

n [15]:
```
merged_Credit_app_records.select("NAME_HOUSING_TYPE").distinct().show()
```

▸ Spark Job Progress

```
+-------------------+
|  NAME_HOUSING_TYPE|
+-------------------+
|Municipal apartment|
|   Rented apartment|
|  House / apartment|
|    Co-op apartment|
|   Office apartment|
|       With parents|
+-------------------+
```

n [16]:
```
merged_Credit_app_records = merged_Credit_app_records.withColumn('RENT_HOUSING_TYPE', fun.when(fun.col('NAME_HOUSING_TYPE') == 'R

target_rent_customers = merged_Credit_app_records.groupby('RENT_HOUSING_TYPE', 'target').count().orderBy('RENT_HOUSING_TYPE')

pivoted_data = merged_Credit_app_records.groupby('RENT_HOUSING_TYPE').pivot("target", ['1', '0']).count()

pivoted_data.show()

pivoted_data.select(col("RENT_HOUSING_TYPE"), *adjusted).show()
```

▸ Spark Job Progress

```
+-----------------+---+-----+
|RENT_HOUSING_TYPE|  1|    0|
+-----------------+---+-----+
|         Not Rent|416|24279|
|             Rent|  6|  433|
+-----------------+---+-----+

+-----------------+--------------------+------------------+
|RENT_HOUSING_TYPE|                   1|                 0|
+-----------------+--------------------+------------------+
|         Not Rent|0.016845515286495243|0.9831544847135048|
|             Rent| 0.01366742596810934|0.9863325740318907|
+-----------------+--------------------+------------------+
```

Considering the above data people living on rate are less likely to default as compared to non-rental person.

Do single people have a high proportion of bad customers as compared to married customers ?

In [17]:
```
merged_Credit_app_records.select("NAME_FAMILY_STATUS").distinct().show()
```

▸ Spark Job Progress

```
+--------------------+
|  NAME_FAMILY_STATUS|
+--------------------+
|           Separated|
|             Married|
|Single / not married|
|      Civil marriage|
|               Widow|
+--------------------+
```

In [18]:
```
merged_Credit_app_records = merged_Credit_app_records.withColumn('MARRIED_STATUS', fun.when(fun.col('NAME_FAMILY_STATUS') == 'Mar
```

In [19]:
```
target_married_customers = merged_Credit_app_records.groupby('MARRIED_STATUS', 'target').count().orderBy('MARRIED_STATUS')

pivoted_data = merged_Credit_app_records.groupby('MARRIED_STATUS').pivot("target", ['1', '0']).count()

pivoted_data.show()

pivoted_data.select(col("MARRIED_STATUS"), *adjusted).show()
```

▸ Spark Job Progress

```
+--------------+---+-----+
|MARRIED_STATUS|  1|    0|
+--------------+---+-----+
|       Married|277|17232|
|     Unmarried|145| 7480|
+--------------+---+-----+

+--------------+-------------------+------------------+
|MARRIED_STATUS|                  1|                 0|
+--------------+-------------------+------------------+
|       Married|0.01582043520475184|0.9841795647952482|
|     Unmarried|0.01901639344262295| 0.980983606557377|
+--------------+-------------------+------------------+
```

As per above data Married people are more likely to default as compared to unmarried.

Model Evaluation 1st Iteration

In [21]:
```
# bins for continous variables

spike_columns = ['CNT_CHILDREN','AMT_INCOME_TOTAL','DAYS_BIRTH','DAYS_EMPLOYED','CNT_FAM_MEMBERS']

max_bucket_size = 5

for x in spike_columns:
    rt = 0
    num = max_bucket_size
    while ((np.abs(rt) < 1) & (num>1)):
        discretizer = QuantileDiscretizer(numBuckets=num, inputCol=x, outputCol=x + "bucket")
        d = discretizer.fit(merged_Credit_app_records).transform(merged_Credit_app_records)
        d1 = d.groupby(x+ "bucket").agg(F.mean(x).alias("X1"),F.mean("target").alias("Y1")).orderBy("X1")
        a1 = d1.select("X1").rdd.flatMap(lambda x: x).collect()
        b1 = d1.select("Y1").rdd.flatMap(lambda x: x).collect()
        rt, p = stats.spearmanr(a1, b1)
        if(rt == 1):
            print(x+"bucket")
            merged_Credit_app_records = discretizer.fit(merged_Credit_app_records).transform(merged_Credit_app_records)
        num = num-1

print(merged_Credit_app_records.columns)
```

▸ Spark Job Progress

```
AMT_INCOME_TOTALbucket
DAYS_EMPLOYEDbucket
['ID', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'NAME_INCOME_TYPE', 'NAME_EDUCATIO
N_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'FLAG_MOBIL', 'FLAG_WORK_PHONE', 'FLAG_PHON
E', 'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'target', 'RENT_HOUSING_TYPE', 'MARRIED_STATUS', 'AMT_INCOME_TOTALbucke
t', 'DAYS_EMPLOYEDbucket']
```

In [27]:
```
#splitting in test and train data
train, test = encoded_df_saved.randomSplit([0.7, 0.3], seed = 2018)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))

#Specifying feature list to be used for logistic regression
features_list = [ 'CODE_GENDER_woe','FLAG_OWN_CAR_woe','FLAG_OWN_REALTY_woe','NAME_EDUCATION_TYPE_woe','NAME_FAMILY_STATUS_woe',
'NAME_HOUSING_TYPE_woe','FLAG_MOBIL_woe','FLAG_WORK_PHONE_woe','FLAG_PHONE_woe','FLAG_EMAIL_woe','OCCUPATION_TYPE_woe',
'AMT_INCOME_TOTALbucket_woe','DAYS_EMPLOYEDbucket_woe']

#Creating feature vector column with all the input variables
assembler = VectorAssembler(inputCols=features_list, outputCol="features")
train = assembler.transform(train)
test = assembler.transform(test)

#Building logistic regression model and predict on test & train dataset using the same model
lr = LogisticRegression(labelCol="target", featuresCol= "features",maxIter=10)
model=lr.fit(train)
predict_train=model.transform(train)
predict_test=model.transform(test)
predict_test.select("target","prediction","probability").show(truncate=False)

#Creating a probability column for target probability
secondelement=udf(lambda v:float(v[1]),FloatType())
predict_train = predict_train.withColumn('prob_int', secondelement('probability'))
predict_test = predict_test.withColumn('prob_int', secondelement('probability'))
predict_test.select("target","prediction","probability","prob_int").show(truncate=False)
```

```
Training Dataset Count: 17541
Test Dataset Count: 7593
+------+----------+----------------------------------------+
|target|prediction|probability                             |
+------+----------+----------------------------------------+
|0     |0.0       |[0.9846176278381278,0.015382372161872151]|
|0     |0.0       |[0.9846176278381278,0.015382372161872151]|
|0     |0.0       |[0.986773310087667,0.01322668991233313]  |
|0     |0.0       |[0.9863673224127191,0.013632677587280992]|
|0     |0.0       |[0.9863673224127191,0.013632677587280992]|
|0     |0.0       |[0.9872500834417228,0.012749916558277178]|
|0     |0.0       |[0.9872500834417228,0.012749916558277178]|
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |
|0     |0.0       |[0.984954912566530 1,0.015045087433469898]|
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|
|0     |0.0       |[0.9855085313022302,0.014491468697769775]|
+------+----------+----------------------------------------+
only showing top 20 rows
```

```
+------+----------+----------------------------------------+------------+
|target|prediction|probability                             |prob_int    |
+------+----------+----------------------------------------+------------+
|0     |0.0       |[0.9846176278381278,0.015382372161872151]|0.015382372 |
|0     |0.0       |[0.9846176278381278,0.015382372161872151]|0.015382372 |
|0     |0.0       |[0.986773310087667,0.01322668991233313]  |0.01322669  |
|0     |0.0       |[0.9863673224127191,0.013632677587280992]|0.0136326775|
|0     |0.0       |[0.9863673224127191,0.013632677587280992]|0.0136326775|
|0     |0.0       |[0.9872500834417228,0.012749916558277178]|0.012749917 |
|0     |0.0       |[0.9872500834417228,0.012749916558277178]|0.012749917 |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |0.009385393 |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |0.009385393 |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |0.009385393 |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |0.009385393 |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |0.009385393 |
|0     |0.0       |[0.990614606918368,0.00938539308163209]  |0.009385393 |
|0     |0.0       |[0.9849549125665301,0.015045087433469898]|0.015045088 |
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|0.009633797 |
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|0.009633797 |
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|0.009633797 |
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|0.009633797 |
|0     |0.0       |[0.9903662030769003,0.009633796923099743]|0.009633797 |
|0     |0.0       |[0.9855085313022302,0.014491468697769775]|0.014491469 |
+------+----------+----------------------------------------+------------+
only showing top 20 rows
```

```
In [28]: # AUC ROC for training set
trainingSummary = model.summary
roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))


# AUC ROC for test set
preds = predict_test.select('target','prob_int').rdd.map(lambda row: (float(row['prob_int']), float(row['target']))).collect()
lr_probs, y_test = zip(*preds)
fpr, tpr, thresholds = roc_curve(y_test, lr_probs, pos_label = 1)

ns_probs = [0 for _ in range(len(y_test))]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)

# summarize scores
print('Testing set areaUnderROC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

▸ Spark Job Progress

```
Training set areaUnderROC: 0.6454687677330996
Testing set areaUnderROC=0.621
```

```
In [29]: #calculating KS

#Creating deciles using the predcited probability column
discretizer = QuantileDiscretizer(numBuckets=10, inputCol="prob_int", outputCol="PROB_BUCKET")
predict_train = discretizer.fit(predict_train).transform(predict_train)
predict_test = discretizer.fit(predict_test).transform(predict_test)

#Creating #Good & # bad for each decile
ks_train = predict_train.groupby('PROB_BUCKET').agg(F.count("ID").alias("Total"),
                                                     F.max("prob_int").alias("max_prob"),
                                                     F.min("prob_int").alias("min_prob"),
                                                     F.sum("target").alias("bads") ,
                                                     F.count(F.when(col("target") == 0,1)).alias("goods")).sort(col('PROB_BUCKET')

ks_test = predict_test.groupby('PROB_BUCKET').agg(F.count("ID").alias("Total"),
                                                   F.max("prob_int").alias("max_prob"),
                                                   F.min("prob_int").alias("min_prob"),
                                                   F.sum("target").alias("bads") ,
                                                   F.count(F.when(col("target") == 0,1)).alias("goods")).sort(col('PROB_BUCKET').c

#Creating %cumulative good,%cumulative bad & KS
ks_train = ks_train.withColumn('%cumuBad', F.sum('bads').over(Window.orderBy(F.col('PROB_BUCKET').desc()).rowsBetween(Window.unbo
)/ F.sum('bads').over(Window.partitionBy()))
ks_train = ks_train.withColumn('%cumuGood', F.sum('goods').over(Window.orderBy(F.col('PROB_BUCKET').desc()).rowsBetween(Window.ur
)/ F.sum('goods').over(Window.partitionBy()))
ks_train = ks_train.withColumn('%diff', F.col('%cumuBad') -  F.col('%cumuGood'))
ks_train = ks_train.withColumn('max_diff', F.when(F.col('%diff') ==F.max('%diff').over(Window.partitionBy()) ,'<----').otherwise(

ks_test = ks_test.withColumn('%cumuBad', F.sum('bads').over(Window.orderBy(F.col('PROB_BUCKET').desc()).rowsBetween(Window.unbour
)/ F.sum('bads').over(Window.partitionBy()))
ks_test = ks_test.withColumn('%cumuGood', F.sum('goods').over(Window.orderBy(F.col('PROB_BUCKET').desc()).rowsBetween(Window.unbo
)/ F.sum('goods').over(Window.partitionBy()))
ks_test = ks_test.withColumn('%diff', F.col('%cumuBad') -  F.col('%cumuGood'))
ks_test = ks_test.withColumn('max_diff', F.when(F.col('%diff') ==F.max('%diff').over(Window.partitionBy()) ,'<----').otherwise('
```

```
ks_train.sort(col('PROB_BUCKET').desc()).show()
ks_test.sort(col('PROB_BUCKET').desc()).show()
```

Spark Job Progress

PROB_BUCKET	Total	max_prob	min_prob	bads	goods	%cumuBad	%cumuGood	%diff	max_diff
9.0	1765	0.12203759	0.028323418	64	1701	0.2229965156794425	0.09858583516865654	0.12441068051078596	
8.0	1748	0.028291991	0.021746911	49	1699	0.39372822299651566	0.19705575518720297	0.1966724678093127	
7.0	1762	0.021745775	0.018268326	27	1735	0.4878048780487805	0.29761214790773155	0.19019273014104893	
6.0	1738	0.018267384	0.015872356	34	1704	0.6062717770034843	0.39637185580155326	0.20989992120193107	<----
5.0	1763	0.015871823	0.014023574	25	1738	0.6933797909407665	0.497102121247247	0.1962776696935195	
4.0	1748	0.014022303	0.012338519	19	1729	0.759581881533101	0.5973107685174452	0.1622711301565578	
3.0	1752	0.012337526	0.010942443	22	1730	0.8362369337979094	0.6975773733626985	0.13865956043521088	
2.0	1749	0.010939022	0.0095387865	22	1727	0.9128919860627178	0.7976701054827866	0.11522188057993121	
1.0	1768	0.00953231	0.0075515183	16	1752	0.9686411149825784	0.8992117769792511	0.06942933800332729	
0.0	1748	0.007547002	0.0020103783	9	1739	1.0	1.0	0.0	

PROB_BUCKET	Total	max_prob	min_prob	bads	goods	%cumuBad	%cumuGood	%diff	max_diff
9.0	764	0.12203759	0.028026735	32	732	0.23703703703703705	0.09814963797264682	0.13888739906439024	
8.0	760	0.028010188	0.021881435	16	744	0.35555555555555557	0.19790828640386163	0.15764726915169394	
7.0	752	0.02187429	0.018427372	15	737	0.4666666666666667	0.29672834540091175	0.16993832126575492	
6.0	766	0.018425252	0.0160226	21	745	0.6222222222222222	0.39662107803700725	0.22560114418521499	<----
5.0	761	0.01601351	0.013969402	12	749	0.7111111111111111	0.4970501474926254	0.21406096361848576	
4.0	758	0.013969201	0.012325135	2	756	0.725925925925926	0.5984178063824082	0.12750811954351782	
3.0	760	0.012320538	0.010847587	11	749	0.8074074074074075	0.6988468758380263	0.10856053156938117	
2.0	750	0.010843293	0.00939306	6	744	0.8518518518518519	0.798605524269241	0.05324632758261083	
1.0	764	0.009385393	0.00736083	11	753	0.9333333333333333	0.8995709305443819	0.033762402788951484	
0.0	758	0.007354714	0.0020103783	9	749	1.0	1.0	0.0	

Model Evaluation 2^{nd} Iteration

In [30]:

```
#splitting in test and train data
train, test = encoded_df_saved.randomSplit([0.7, 0.3], seed = 2018)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))

#Specifying feature list to be used for logistic regression
features_list = ['AMT_INCOME_TOTALbucket_woe','CODE_GENDER_woe','DAYS_EMPLOYEDbucket_woe','FLAG_OWN_REALTY_woe',
                 'NAME_EDUCATION_TYPE_woe','NAME_FAMILY_STATUS_woe','NAME_HOUSING_TYPE_woe','OCCUPATION_TYPE_woe',
                 'NAME_INCOME_TYPE_woe']

#Creating feature vector column with all the input variables
assembler = VectorAssembler(inputCols=features_list, outputCol="features")
train = assembler.transform(train)
test = assembler.transform(test)

#Building logistic regression model and predict on test & train dataset using the same model
lr = LogisticRegression(labelCol="target", featuresCol= "features",maxIter=10)
model=lr.fit(train)
predict_train=model.transform(train)
predict_test=model.transform(test)
#predict_test.select("target","prediction","probability").show(truncate=False)

#Creating a probability column for target probability
secondelement=udf(lambda v:float(v[1]),FloatType())
predict_train = predict_train.withColumn('prob_int', secondelement('probability'))
predict_test = predict_test.withColumn('prob_int', secondelement('probability'))

# AUC ROC for training set
trainingSummary = model.summary
roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```

```
# AUC ROC for test set
preds = predict_test.select('target','prob_int').rdd.map(lambda row: (float(row['prob_int']), float(row['target']))).collect()
lr_probs, y_test = zip(*preds)
fpr, tpr, thresholds = roc_curve(y_test, lr_probs, pos_label = 1)

ns_probs = [0 for _ in range(len(y_test))]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)

# summarize scores
print('Testing set areaUnderROC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

```
#calculating KS

#Creating deciles using the predcited probability column
discretizer = QuantileDiscretizer(numBuckets=10, inputCol="prob_int", outputCol="PROB_BUCKET")
predict_train = discretizer.fit(predict_train).transform(predict_train)
predict_test = discretizer.fit(predict_test).transform(predict_test)

#Creating #Good & # bad for each decile
ks_train = predict_train.groupby('PROB_BUCKET').agg(F.count("ID").alias("Total"),
                                                     F.max("prob_int").alias("max_prob"),
                                                     F.min("prob_int").alias("min_prob"),
                                                     F.sum("target").alias("bads") ,
                                                     F.count(F.when(col("target") == 0,1)).alias("goods")).sort(col('PROB_BUCKET')

ks_test = predict_test.groupby('PROB_BUCKET').agg(F.count("ID").alias("Total"),
                                                   F.max("prob_int").alias("max_prob"),
                                                   F.min("prob_int").alias("min_prob"),
                                                   F.sum("target").alias("bads") ,
                                                   F.count(F.when(col("target") == 0,1)).alias("goods")).sort(col('PROB_BUCKET').d

#Creating %cumulative good,%cumulative bad & KS
ks_train = ks_train.withColumn('%cumuBad', F.sum('bads').over(Window.orderBy(F.col('PROB_BUCKET').desc()).rowsBetween(Window.unbo
)/ F.sum('bads').over(Window.partitionBy()))
ks_train = ks_train.withColumn('%cumuGood', F.sum('goods').over(Window.orderBy(F.col('PROB_BUCKET').desc()).rowsBetween(Window.un
)/ F.sum('goods').over(Window.partitionBy()))
ks_train = ks_train.withColumn('%diff', F.col('%cumuBad') -  F.col('%cumuGood'))
ks_train = ks_train.withColumn('max_diff', F.when(F.col('%diff') ==F.max('%diff').over(Window.partitionBy()) ,'<----').otherwise(

ks_test = ks_test.withColumn('%cumuBad', F.sum('bads').over(Window.orderBy(F.col('PROB_BUCKET').desc()).rowsBetween(Window.unboun
)/ F.sum('bads').over(Window.partitionBy()))
ks_test = ks_test.withColumn('%cumuGood', F.sum('goods').over(Window.orderBy(F.col('PROB_BUCKET').desc()).rowsBetween(Window.unbo
)/ F.sum('goods').over(Window.partitionBy()))
ks_test = ks_test.withColumn('%diff', F.col('%cumuBad') -  F.col('%cumuGood'))
ks_test = ks_test.withColumn('max_diff', F.when(F.col('%diff') ==F.max('%diff').over(Window.partitionBy()) ,'<----').otherwise('

ks_train.sort(col('PROB_BUCKET').desc()).show()
ks_test.sort(col('PROB_BUCKET').desc()).show()
```

```
Training Dataset Count: 17541
Test Dataset Count: 7593
Training set areaUnderROC: 0.6521478229155769
Testing set areaUnderROC=0.649
```

PROB_BUCKET	Total	max_prob	min_prob	bads	goods	%cumuBad	%cumuGood	%diff	max_diff
9.0	1766	0.9828491	0.027494008	63	1703	0.21951219512195122	0.09870175031876666	0.12081044480318456	
8.0	1739	0.027488615	0.021537388	46	1693	0.3797909407665505	0.19682392488698272	0.1829670158795678	
7.0	1770	0.021510385	0.01797418	37	1733	0.5087108013937283	0.29726440245740116	0.2114463989363271	
6.0	1735	0.017968645	0.015870653	37	1698	0.6376306620209059	0.39567636490089253	0.24195429712001337	<----
5.0	1758	0.015847946	0.01369995	27	1731	0.7317073170731707	0.4960009273212009	0.23570638975196984	
4.0	1755	0.013699586	0.012265156	13	1742	0.7770034843205574	0.5969630230671149	0.18004046125344253	
3.0	1760	0.012264656	0.010990919	18	1742	0.8397212543554007	0.6979251188130289	0.1417961355423718	
2.0	1855	0.010988842	0.00923702	16	1839	0.8954703832752613	0.8045090993392836	0.09096128393597769	
1.0	1640	0.009227373	0.0076757837	17	1623	0.9547038327526133	0.8985742436536456	0.0561295890989677	
0.0	1763	0.0076742047	0.0017910189	13	1750	1.0	1.0	0.0	

PROB_BUCKET	Total	max_prob	min_prob	bads	goods	%cumuBad	%cumuGood	%diff	max_diff
9.0	766	0.99120146	0.02712887	34	732	0.2518518518518518	0.09814963797264682	0.15370221387920502	
8.0	755	0.027115295	0.021611314	18	737	0.3851851851851852	0.19696969696969696	0.18821548821548822	
7.0	761	0.021602597	0.018170366	18	743	0.5185185185185185	0.2965942611960311	0.22192425732248738	
6.0	760	0.018163167	0.015905177	20	740	0.6666666666666666	0.39581657280772325	0.2708500938589434	<----
5.0	758	0.015900783	0.013623883	7	751	0.7185185185185186	0.4965138106731027	0.22200470784541587	
4.0	756	0.013618922	0.012186022	6	750	0.762962962962963	0.5970769643336015	0.1658859986293615	
3.0	759	0.012183561	0.0109289065	7	752	0.8148148148148148	0.6979082864038616	0.11690652841095317	
2.0	754	0.010906844	0.009193793	8	746	0.8740740740740741	0.7979351032448377	0.07613897082923637	
1.0	768	0.009193477	0.0073632	9	759	0.9407407407407408	0.8997050147492626	0.0410357259914782	
0.0	756	0.007326404	0.0017910189	8	748	1.0	1.0	0.0	

In [32]:
```
#Creating prediction classs based on the prob cut off by KS value

min_prob_max_ks = ks_test.where(col("max_diff") == '<----').select(col("min_prob")).collect()[0]
predict_test = predict_test.withColumn('pred_class', F.when(F.col('prob_int') > round(min_prob_max_ks[0],4) ,1).otherwise(0))
predict_test.groupBy('pred_class','target').count().show()

#Generating evaluation metrics
results = predict_test.select(['pred_class', 'target'])
results = results.withColumn("target", results["target"].cast(DoubleType()))
results = results.withColumn("pred_class", results["pred_class"].cast(DoubleType()))

predictionAndLabels=results.rdd
metrics = MulticlassMetrics(predictionAndLabels)
metrics1 = BinaryClassificationMetrics(predictionAndLabels)

cm=metrics.confusionMatrix().toArray()
print(cm)
# Overall statistics
precision = metrics.precision(1)
recall = metrics.recall(1)
f1Score = metrics.fMeasure()
print("Summary Stats")
print("Precision = %s" % precision)
print("Recall = %s" % recall)
print("F1 Score = %s" % f1Score)
# Area under precision-recall curve
print("Area under PR = %s" % metrics1.areaUnderPR)
# Area under ROC curve
print("Area under ROC = %s" % metrics1.areaUnderROC)
```

```
+----------+------+-----+
|pred_class|target|count|
+----------+------+-----+
|         1|     1|   90|
|         0|     0| 4497|
|         0|     1|   45|
|         1|     0| 2961|
+----------+------+-----+

[[4497. 2961.]
 [  45.   90.]]
Summary Stats
Precision = 0.029498525073746312
Recall = 0.6666666666666666
F1 Score = 0.6041090478071909
Area under PR = 0.027545359858307625
Area under ROC = 0.6348216680075086
```