databricks ctr_Prediction

```
> ctr_dataframe = sqlContext.sql("SELECT * FROM ctr_df2")
```

```
> ctr_dataframe.columns
```

```
Out[2]:
['id',
 'label',
 'hour',
 'C1',
 'banner_pos',
 'site_id',
 'site_domain',
 'site_category',
 'app_id',
 'app_domain',
 'app_category',
 'device_id',
 'device_ip',
 'device_model',
 'device_type',
 'device_conn_type',
 'C14',
 'C15',
 'C16',
 'C17',
 'C18',
 'C19',
 'C20',
 'C21']
```

```
> ctr_dataframe.take(5)
```

```
Out[3]:
[Row(id=u'1000009418151094273', label=0.0, hour=14102100, C1=u'1005', banner_pos=u'0', site_id=u'1fbe01fe', site_domain=u'f
3845767', site_category=u'28905ebd', app_id=u'ecad2386', app_domain=u'7801e8d9', app_category=u'07d7df22', device_id=u'a99f
214a', device_ip=u'ddd2926e', device_model=u'44956a24', device_type=u'1', device_conn_type=u'2', C14=u'15706', C15=u'320',
C16=u'50', C17=u'1722', C18=u'0', C19=u'35', C20=u'-1', C21=u'79'),
 Row(id=u'10000169349117863715', label=0.0, hour=14102100, C1=u'1005', banner_pos=u'0', site_id=u'1fbe01fe', site_domain=
u'f3845767', site_category=u'28905ebd', app_id=u'ecad2386', app_domain=u'7801e8d9', app_category=u'07d7df22', device_id=u'a
99f214a', device_ip=u'96809ac8', device_model=u'711ee120', device_type=u'1', device_conn_type=u'0', C14=u'15704', C15=u'32
0', C16=u'50', C17=u'1722', C18=u'0', C19=u'35', C20=u'100084', C21=u'79'),
 Row(id=u'10000371904215119486', label=0.0, hour=14102100, C1=u'1005', banner_pos=u'0', site_id=u'1fbe01fe', site_domain=
u'f3845767', site_category=u'28905ebd', app_id=u'ecad2386', app_domain=u'7801e8d9', app_category=u'07d7df22', device_id=u'a
99f214a', device_ip=u'b3cf8def', device_model=u'8a4875bd', device_type=u'1', device_conn_type=u'0', C14=u'15704', C15=u'32
0', C16=u'50', C17=u'1722', C18=u'0', C19=u'35', C20=u'100084', C21=u'79'),
 Row(id=u'10000640724480838376', label=0.0, hour=14102100, C1=u'1005', banner_pos=u'0', site_id=u'1fbe01fe', site_domain=
u'f3845767', site_category=u'28905ebd', app_id=u'ecad2386', app_domain=u'7801e8d9', app_category=u'07d7df22', device_id=u'a
99f214a', device_ip=u'e8275b8f', device_model=u'6332421a', device_type=u'1', device_conn_type=u'0', C14=u'15706', C15=u'32
0', C16=u'50', C17=u'1722', C18=u'0', C19=u'35', C20=u'100084', C21=u'79'),
 Row(id=u'10000679056417042096', label=0.0, hour=14102100, C1=u'1005', banner_pos=u'1', site_id=u'fe8cc448', site_domain=
u'9166c161', site_category=u'0569f928', app_id=u'ecad2386', app_domain=u'7801e8d9', app_category=u'07d7df22', device_id=u'a
99f214a', device_ip=u'9644d0bf', device_model=u'779d90c2', device_type=u'1', device_conn_type=u'0', C14=u'18993', C15=u'32
0', C16=u'50', C17=u'2161', C18=u'0', C19=u'35', C20=u'-1', C21=u'157')]
```

```
> ctr_dataframe.count()
```

```
Out[4]: 5000000
```

```
> ctr_dataframe.printSchema
```

```
Out[5]: <bound method DataFrame.printSchema of DataFrame[id: string, label: double, hour: int, C1: string, banner_pos: stri
ng, site_id: string, site_domain: string, site_category: string, app_id: string, app_domain: string, app_category: string,
device_id: string, device_ip: string, device_model: string, device_type: string, device_conn_type: string, C14: string, C1
5: string, C16: string, C17: string, C18: string, C19: string, C20: string, C21: string]>
```

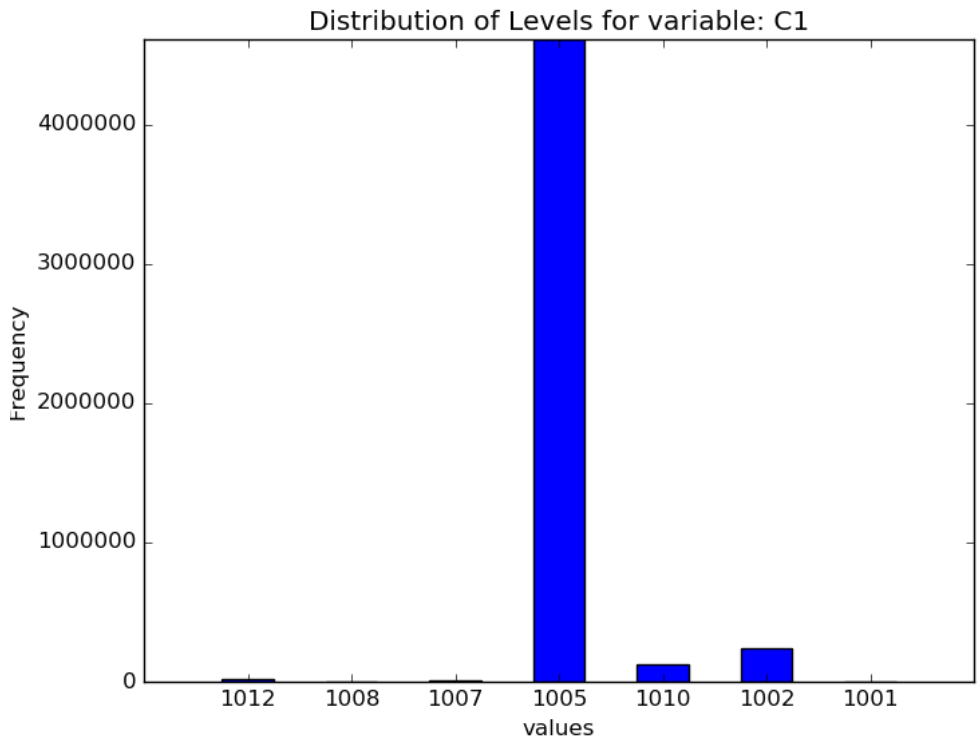
```
############# Collect categorical variable levels
categories = {}
idxCategories = [3,4,5,6,7,8,9,10,11,13,14,15,16,17,18,19,20,21,22,23]
for i in idxCategories: ##idxCategories contains indexes of rows that contains categorical data
    distinctValues = ctr_dataframe.map(lambda x : x[i]).countByValue()
    categories[i] =  distinctValues
```

```
categories[10]
```

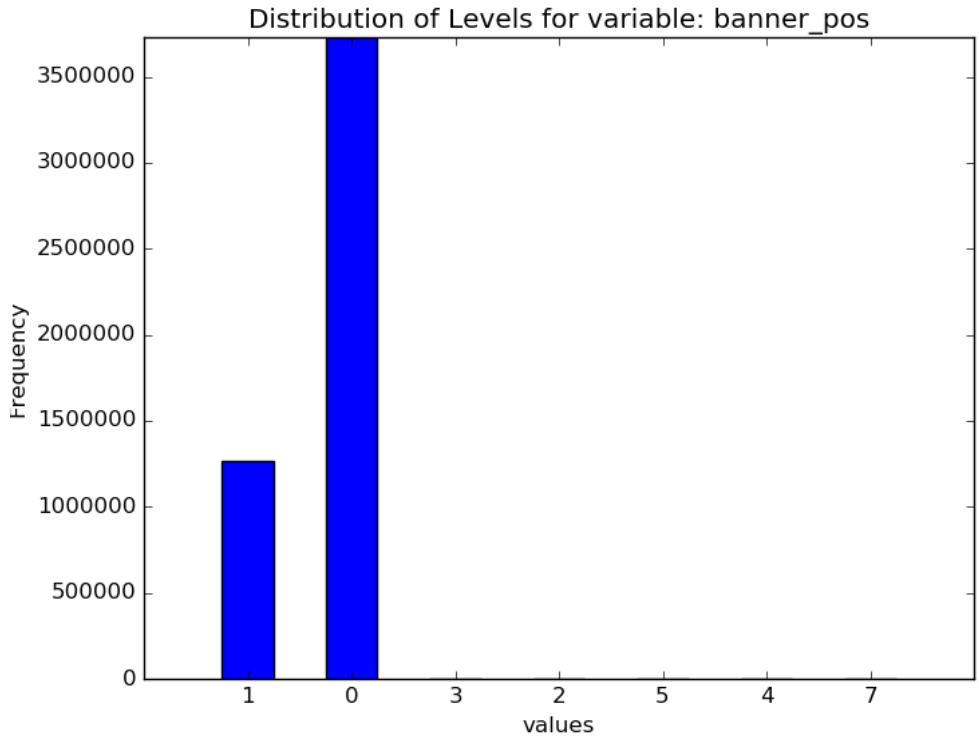
```
Out[6]: defaultdict(<type 'int'>, {u'cef3e649': 263397, u'ef03ae90': 1, u'a3c42688': 1699, u'8df2e842': 183, u'2fc4f2aa':
2, u'86c1a5a3': 2, u'52de74cf': 1, u'd1327cf5': 25510, u'5326cf99': 4, u'fc6fa53d': 3989, u'09481d60': 17778, u'4b7ade46':
3, u'879c24eb': 1478, u'8ded1f7a': 139241, u'0f2161f8': 1151278, u'18b1e0be': 26, u'f95efa07': 159706, u'4681bb9d': 514,
u'07d7df22': 3221761, u'7113d72a': 28, u'71af18ce': 1, u'dc97ec06': 2825, u'75d80bbe': 5252, u'0f9a328c': 838, u'a86a3e89':
538, u'0bfbc358': 128, u'4ce2e9fc': 3291, u'79f0b860': 143, u'bd41f328': 1, u'a7fd01ec': 34, u'2281a340': 348})
```

```
########### visualization of the caregorical variables using matplotlib
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
```

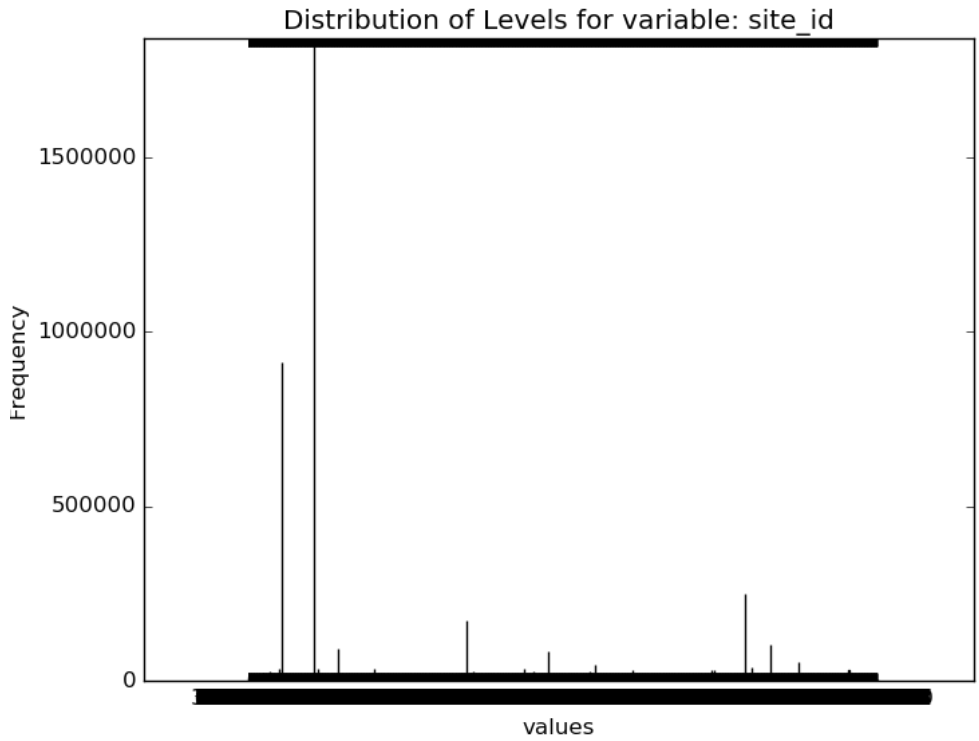
```
X = np.arange(len(categories[3]))
f = plt.figure(3)
plt.bar(X, categories[3].values(), align='center', width=0.5)
plt.xticks(X, categories[3].keys())
ymax = max(categories[3].values()) + 1
plt.ylim(0, ymax)
plt.xlabel('values')
plt.ylabel('Frequency')
plt.title('Distribution of Levels for variable: %s'%(ctr_dataframe.columns[3]))
display(f)
```



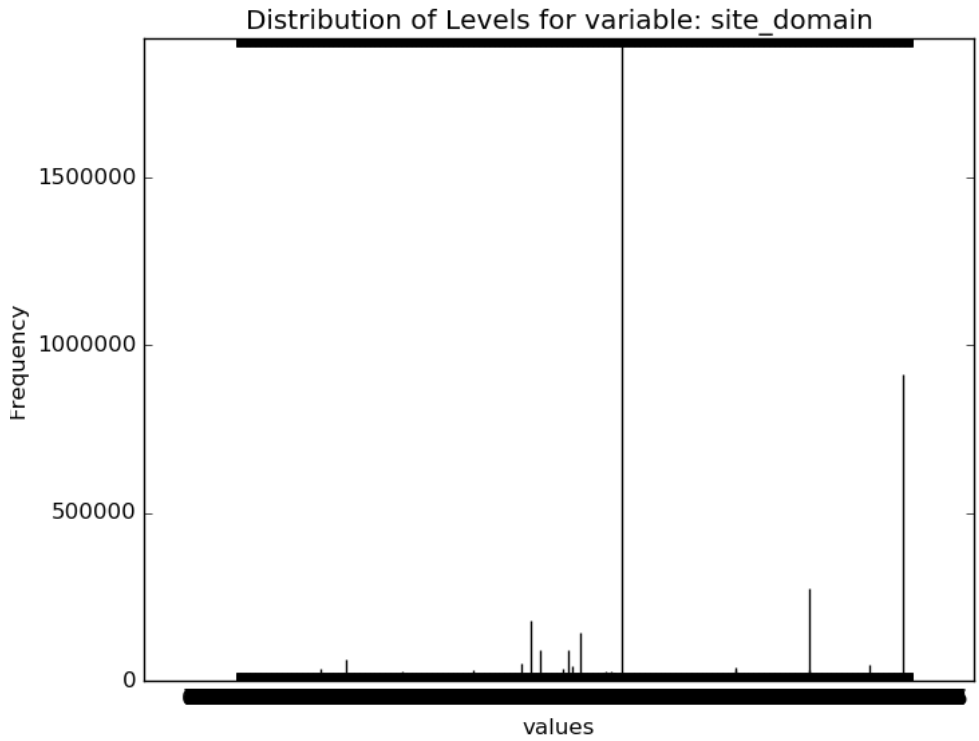
```
> X = np.arange(len(categories[4]))
  f = plt.figure(4)
  plt.bar(X, categories[4].values(), align='center', width=0.5)
  plt.xticks(X, categories[4].keys())
  ymax = max(categories[4].values()) + 1
  plt.ylim(0, ymax)
  plt.xlabel('values')
  plt.ylabel('Frequency')
  plt.title('Distribution of Levels for variable: %s'%(ctr_dataframe.columns[4]))
  display(f)
```




```
> X = np.arange(len(categories[5]))
  f = plt.figure(5)
  plt.bar(X, categories[5].values(), align='center', width=0.5)
  plt.xticks(X, categories[5].keys())
  ymax = max(categories[5].values()) + 1
  plt.ylim(0, ymax)
  plt.xlabel('values')
  plt.ylabel('Frequency')
  plt.title('Distribution of Levels for variable: %s'%(ctr_dataframe.columns[5]))
  display(f)
```




```
> X = np.arange(len(categories[6]))
  f = plt.figure(6)
  plt.bar(X, categories[6].values(), align='center', width=0.5)
  plt.xticks(X, categories[6].keys())
  ymax = max(categories[6].values()) + 1
  plt.ylim(0, ymax)
  plt.xlabel('values')
  plt.ylabel('Frequency')
  plt.title('Distribution of Levels for variable: %s'%(ctr_dataframe.columns[6]))
  display(f)
```




```
> ############## Use randomSplit with weights and seed to get training, test data sets
  weights = [.8, .2]
  seed = 22

  ctr_Train, ctr_Test = ctr_dataframe.randomSplit(weights, seed)
```

```
> ctr_Train.count()
```

```
Out[13]: 4000057
```

```
> ctr_Test.count()
```

```
Out[14]: 999943
```

```
> ################### Use StringIndexer and OneHotEncoder to create transformers
  from pyspark.ml.feature import OneHotEncoder, StringIndexer,VectorAssembler

  C1_Indexer = StringIndexer(inputCol="C1", outputCol="indexed_C1", handleInvalid="skip")
  C1_Encoder = OneHotEncoder(inputCol="indexed_C1", outputCol="C1_Vector")

  banner_pos_Indexer = StringIndexer(inputCol="banner_pos", outputCol="indexed_banner_pos", handleInvalid="skip")
  banner_pos_Encoder = OneHotEncoder(inputCol="indexed_banner_pos", outputCol="banner_pos_Vector")

  site_id_Indexer = StringIndexer(inputCol="site_id", outputCol="indexed_site_id", handleInvalid="skip")
  site_id_Encoder = OneHotEncoder(inputCol="indexed_site_id", outputCol="site_id_Vector")

  site_domain_Indexer = StringIndexer(inputCol="site_domain", outputCol="indexed_site_domain", handleInvalid="skip")
  site_domain_Encoder = OneHotEncoder(inputCol="indexed_site_domain", outputCol="site_domain_Vector")

  site_category_Indexer = StringIndexer(inputCol="site_category", outputCol="indexed_site_category", handleInvalid="skip")
  site_category_Encoder = OneHotEncoder(inputCol="indexed_site_category", outputCol="site_category_Vector")

  app_id_Indexer = StringIndexer(inputCol="app_id", outputCol="indexed_app_id", handleInvalid="skip")
  app_id_Encoder = OneHotEncoder(inputCol="indexed_app_id", outputCol="app_id_Vector")

  app_domain_Indexer = StringIndexer(inputCol="app_domain", outputCol="indexed_app_domain", handleInvalid="skip")
  app_domain_Encoder = OneHotEncoder(inputCol="indexed_app_domain", outputCol="app_domain_Vector")

  app_category_Indexer = StringIndexer(inputCol="app_category", outputCol="indexed_app_category", handleInvalid="skip")
  app_category_Encoder = OneHotEncoder(inputCol="indexed_app_category", outputCol="app_category_Vector")

  device_id_Indexer = StringIndexer(inputCol="device_id", outputCol="indexed_device_id", handleInvalid="skip")
  device_id_Encoder = OneHotEncoder(inputCol="indexed_device_id", outputCol="device_id_Vector")

  device_model_Indexer = StringIndexer(inputCol="device_model", outputCol="indexed_device_model", handleInvalid="skip")
  device_model_Encoder = OneHotEncoder(inputCol="indexed_device_model", outputCol="device_model_Vector")

  device_type_Indexer = StringIndexer(inputCol="device_type", outputCol="indexed_device_type", handleInvalid="skip")
  device_type_Encoder = OneHotEncoder(inputCol="indexed_device_type", outputCol="device_type_Vector")

  device_conn_type_Indexer = StringIndexer(inputCol="device_conn_type", outputCol="indexed_device_conn_type",
  handleInvalid="skip")
  device_conn_type_Encoder = OneHotEncoder(inputCol="indexed_device_conn_type", outputCol="device_conn_type_Vector")

  C14_Indexer = StringIndexer(inputCol="C14", outputCol="indexed_C14", handleInvalid="skip")
  C14_Encoder = OneHotEncoder(inputCol="indexed_C14", outputCol="C14_Vector")

  C15_Indexer = StringIndexer(inputCol="C15", outputCol="indexed_C15", handleInvalid="skip")
  C15_Encoder = OneHotEncoder(inputCol="indexed_C15", outputCol="C15_Vector")

  C16_Indexer = StringIndexer(inputCol="C16", outputCol="indexed_C16", handleInvalid="skip")
  C16_Encoder = OneHotEncoder(inputCol="indexed_C16", outputCol="C16_Vector")

  C17_Indexer = StringIndexer(inputCol="C17", outputCol="indexed_C17", handleInvalid="skip")
  C17_Encoder = OneHotEncoder(inputCol="indexed_C17", outputCol="C17_Vector")

  C18_Indexer = StringIndexer(inputCol="C18", outputCol="indexed_C18", handleInvalid="skip")
  C18_Encoder = OneHotEncoder(inputCol="indexed_C18", outputCol="C18_Vector")

  C19_Indexer = StringIndexer(inputCol="C19", outputCol="indexed_C19", handleInvalid="skip")
  C19_Encoder = OneHotEncoder(inputCol="indexed_C19", outputCol="C19_Vector")

  C20_Indexer = StringIndexer(inputCol="C20", outputCol="indexed_C20", handleInvalid="skip")
  C20_Encoder = OneHotEncoder(inputCol="indexed_C20", outputCol="C20_Vector")

  C21_Indexer = StringIndexer(inputCol="C21", outputCol="indexed_C21", handleInvalid="skip")
  C21_Encoder = OneHotEncoder(inputCol="indexed_C21", outputCol="C21_Vector")
```

```
> ########## Create the assembler stage for the pipeline
  from pyspark.ml import Pipeline
  feat_Assembler = VectorAssembler(
      inputCols=
  ["C1_Vector","banner_pos_Vector","site_id_Vector","site_domain_Vector","site_category_Vector","app_id_Vector",
                "app_domain_Vector","app_category_Vector","device_id_Vector","device_model_Vector","device_type_Vector",
                "device_conn_type_Vector","C14_Vector","C15_Vector","C16_Vector","C17_Vector","C18_Vector","C19_Vector",
                "C20_Vector","C21_Vector","hour"],
      outputCol="features")


> ############### Define the Logistic Regression stage for the pipeline
  from pyspark.ml.classification import LogisticRegression

  lr = LogisticRegression(maxIter=150, regParam=0.01)


> ################## Construct the machine learning pipeline
  from pyspark.ml import Pipeline

  pipeline = Pipeline(stages=
  [C1_Indexer,banner_pos_Indexer,site_id_Indexer,site_domain_Indexer,site_category_Indexer,app_id_Indexer,
                       app_domain_Indexer,app_category_Indexer,device_id_Indexer,device_model_Indexer,device_type_Indexer,

  device_conn_type_Indexer,C14_Indexer,C15_Indexer,C16_Indexer,C17_Indexer,C18_Indexer,C19_Indexer,C20_Indexer,
                       C21_Indexer,C1_Encoder,banner_pos_Encoder,site_id_Encoder,site_domain_Encoder,site_category_Encoder,

  app_id_Encoder,app_domain_Encoder,app_category_Encoder,device_id_Encoder,device_model_Encoder,device_type_Encoder,

  device_conn_type_Encoder,C14_Encoder,C15_Encoder,C16_Encoder,C17_Encoder,C18_Encoder,C19_Encoder,C20_Encoder,
                       C21_Encoder,feat_Assembler, lr])


> ############### Create a pipeline model to train
  model = pipeline.fit(ctr_Train)


> ########### Transform the test data set to produce predictions and compare predictions to labels to determine accuracy of
  the model
  output = model.transform(ctr_Test).select("features", "label", "prediction", "rawPrediction", "probability")

/databricks/spark/python/pyspark/ml/classification.py:207: UserWarning: weights is deprecated. Use coefficients instead.
  warnings.warn("weights is deprecated. Use coefficients instead.")

> Prediction = output.select("prediction","label")


> Prediction.take(10)

Out[22]:
[Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=0.0),
 Row(prediction=0.0, label=1.0)]

> accuracy = Prediction.filter(Prediction['label'] == Prediction['prediction']).count() / float(Prediction.count())



> accuracy

Out[24]: 0.8259478719950977
```

```
> from math import log
  ############Calculates the value of log loss for a given probabilty and label.
  def computeLogLoss(p, y):
      epsilon = 1e-15
      if y == 1:
          return -log(epsilon + p) if p == 0 else -log(p)
      elif y == 0:
          return -log(1 - p + epsilon) if p == 1 else -log(1 - p)
```

```
> Predict_Prob = output.select("label","probability","prediction")
```

```
> Predict_Prob.map(lambda l: (l.label,l.probability[1])).take(5)
```

```
Out[27]:
[(0.0, 0.17459477217379066),
 (0.0, 0.17460492798464505),
 (0.0, 0.17459632762531141),
 (0.0, 0.17459698793512421),
 (0.0, 0.17459254646510092)]
```

```
> Predict_Prob.take(5)
```

```
Out[28]:
[Row(label=0.0, probability=DenseVector([0.8254, 0.1746]), prediction=0.0),
 Row(label=0.0, probability=DenseVector([0.8254, 0.1746]), prediction=0.0),
 Row(label=0.0, probability=DenseVector([0.8254, 0.1746]), prediction=0.0),
 Row(label=0.0, probability=DenseVector([0.8254, 0.1746]), prediction=0.0),
 Row(label=0.0, probability=DenseVector([0.8254, 0.1746]), prediction=0.0)]
```

```
> ####Calculates the log loss for the data given the model.
  def evaluateResults(Predictions, Test_Data):
      return Predictions.map(lambda l: computeLogLoss(l.probability[1],l.label)).sum() / Test_Data.count()
```

```
0.431505297684
```