The dataset was downloaded from UCI Machine Learning Repository: [Data set information](#).

Thanks to the analyticsvidhyateam, we have two more links, where we can find the datasets enriched with headers.

[Download Train Data](#)
[Download Test Data](#)

I called my dataset 'salarypredtrain.csv'. Below are my first actions

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Python version 3.6.3 (default, Oct 15 2017 03:27:45)
SparkSession available as 'spark'.
>>> data = spark.read.csv('C:/salarypredtrain.csv' ,inferSchema = True, header = True)
>>>
>>>
>>> data.printSchema()
root
 |-- age: integer (nullable = true)
 |-- class_of_worker: string (nullable = true)
 |-- industry_code: integer (nullable = true)
 |-- occupation_code: integer (nullable = true)
 |-- education: string (nullable = true)
 |-- wage_per_hour: integer (nullable = true)
 |-- enrolled_in_edu_inst_lastwk: string (nullable = true)
 |-- marital_status: string (nullable = true)
 |-- major_industry_code: string (nullable = true)
 |-- major_occupation_code: string (nullable = true)
 |-- race: string (nullable = true)
 |-- hispanic_origin: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- member_of_labor_union: string (nullable = true)
 |-- reason_for_unemployment: string (nullable = true)
 |-- full_parttime_employment_stat: string (nullable = true)
 |-- capital_gains: integer (nullable = true)
 |-- capital_losses: integer (nullable = true)
 |-- dividend_from_Stocks: integer (nullable = true)
 |-- tax_filer_status: string (nullable = true)
 |-- region_of_previous_residence: string (nullable = true)
 |-- state_of_previous_residence: string (nullable = true)
 |-- d_household_family_stat: string (nullable = true)
 |-- d_household_summary: string (nullable = true)
 |-- migration_msa: string (nullable = true)
 |-- migration_reg: string (nullable = true)
 |-- migration_within_reg: string (nullable = true)
 |-- live_1_year_ago: string (nullable = true)
 |-- migration_sunbelt: string (nullable = true)
 |-- num_person_Worked_employer: integer (nullable = true)
 |-- family_members_under_18: string (nullable = true)
 |-- country_father: string (nullable = true)
 |-- country_mother: string (nullable = true)
 |-- country_self: string (nullable = true)
 |-- citizenship: string (nullable = true)
 |-- business_or_self_employed: integer (nullable = true)
 |-- fill_questionnaire_veteran_admin: string (nullable = true)
 |-- veterans_benefits: integer (nullable = true)
 |-- weeks_worked_in_year: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- income_level: integer (nullable = true)
```

I would like at first to to take a glimpse at the income_level column, which is in fact our target column.

```
>>> data.groupby('income_level').count().show()
+------------+------+
|income_level| count|
+------------+------+
|      -50000|187141|
|       50000| 12382|
+------------+------+
```

We can see that the document has chosen to use the -50000 and 50000 number, and that of course the vast majority, are individuals with less than 50000 dollars per year. I will therefore have to change that through a udf (user defined function), applying a simple lambda function on my income_level column, and create a new column, named 'income'.

```
>>>
>>>
>>> from pyspark.sql.functions import udf
>>>
>>> myfunc = udf(lambda x: 0 if x == -50000 else 1)
>>>
>>> data= data.withColumn('income', myfunc('income_level'))
>>>
>>>
>>> data.describe('income').show()
+-------+-------------------+
|summary|             income|
+-------+-------------------+
|  count|             199523|
|   mean|0.06205800834991455|
| stddev|0.24126148403931813|
|    min|                  0|
|    max|                  1|
+-------+-------------------+
```

I would like to investigate the income in relation to the education level. So

```
>>> data.groupby('education').avg('income').show()
Traceback (most recent call last):
  File "C:\spark\python\pyspark\sql\utils.py", line 63, in deco
    return f(*a, **kw)
  File "C:\spark\python\lib\py4j-0.10.4-src.zip\py4j\protocol.py", line 319, in get_return_value
py4j.protocol.Py4JJavaError: An error occurred while calling o184.avg.
: org.apache.spark.sql.AnalysisException: "income" is not a numeric column. Aggregation function can only be applied on a numeric column.;
        at org.apache.spark.sql.RelationalGroupedDataset$$anonfun$3.apply(RelationalGroupedDataset.scala:99)
        at org.apache.spark.sql.RelationalGroupedDataset$$anonfun$3.apply(RelationalGroupedDataset.scala:96)
        at scala.collection.TraversableLike$$anonfun$map$1.apply(TraversableLike.scala:234)
        at scala.collection.TraversableLike$$anonfun$map$1.apply(TraversableLike.scala:234)
        at scala.collection.Iterator$class.foreach(Iterator.scala:893)
        at scala.collection.AbstractIterator.foreach(Iterator.scala:1336)
        at scala.collection.IterableLike$class.foreach(IterableLike.scala:72)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:54)
        at scala.collection.TraversableLike$class.map(TraversableLike.scala:234)
        at scala.collection.AbstractTraversable.map(Traversable.scala:104)
        at org.apache.spark.sql.RelationalGroupedDataset.aggregateNumericColumns(RelationalGroupedDataset.scala:96)
        at org.apache.spark.sql.RelationalGroupedDataset.avg(RelationalGroupedDataset.scala:270)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
        at py4j.Gateway.invoke(Gateway.java:280)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:214)
        at java.lang.Thread.run(Thread.java:748)


During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\spark\python\pyspark\sql\group.py", line 40, in _api
    jdf = getattr(self._jgd, name)(_to_seq(self.sql_ctx._sc, cols))
  File "C:\spark\python\lib\py4j-0.10.4-src.zip\py4j\java_gateway.py", line 1133, in __call__
  File "C:\spark\python\pyspark\sql\utils.py", line 69, in deco
    raise AnalysisException(s.split(': ', 1)[1], stackTrace)
pyspark.sql.utils.AnalysisException: '"income" is not a numeric column. Aggregation function can only be applied on a numeric column.;'
>>>
```

There is a problem. At the end of the snippet, it states that my 'income' column, is not a numeric type. So I must change the type of my variable to double:

```
>>> from pyspark.sql.types import DoubleType
>>> data = data.withColumn("income2", data["income"].cast("double"))
>>> data.printSchema()
root
 |-- age: integer (nullable = true)
 |-- class_of_worker: string (nullable = true)
 |-- industry_code: integer (nullable = true)
 |-- occupation_code: integer (nullable = true)
 |-- education: string (nullable = true)
 |-- wage_per_hour: integer (nullable = true)
 |-- enrolled_in_edu_inst_lastwk: string (nullable = true)
 |-- marital_status: string (nullable = true)
 |-- major_industry_code: string (nullable = true)
 |-- major_occupation_code: string (nullable = true)
 |-- race: string (nullable = true)
 |-- hispanic_origin: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- member_of_labor_union: string (nullable = true)
 |-- reason_for_unemployment: string (nullable = true)
 |-- full_parttime_employment_stat: string (nullable = true)
 |-- capital_gains: integer (nullable = true)
 |-- capital_losses: integer (nullable = true)
 |-- dividend_from_Stocks: integer (nullable = true)
 |-- tax_filer_status: string (nullable = true)
 |-- region_of_previous_residence: string (nullable = true)
 |-- state_of_previous_residence: string (nullable = true)
 |-- d_household_family_stat: string (nullable = true)
 |-- d_household_summary: string (nullable = true)
 |-- migration_msa: string (nullable = true)
 |-- migration_reg: string (nullable = true)
 |-- migration_within_reg: string (nullable = true)
 |-- live_1_year_ago: string (nullable = true)
 |-- migration_sunbelt: string (nullable = true)
 |-- num_person_Worked_employer: integer (nullable = true)
 |-- family_members_under_18: string (nullable = true)
 |-- country_father: string (nullable = true)
 |-- country_mother: string (nullable = true)
 |-- country_self: string (nullable = true)
 |-- citizenship: string (nullable = true)
 |-- business_or_self_employed: integer (nullable = true)
 |-- fill_questionnaire_veteran_admin: string (nullable = true)
 |-- veterans_benefits: integer (nullable = true)
 |-- weeks_worked_in_year: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- income_level: integer (nullable = true)
 |-- neasthlh: string (nullable = true)
 |-- income: string (nullable = true)
 |-- income2: double (nullable = true)
```

Now we are ready

```
>>>
>>> data.groupby('education').avg('income2').orderBy('avg(income2)').show()
+--------------------+--------------------+
|           education|        avg(income2)|
+--------------------+--------------------+
|            Children|                 0.0|
| Less than 1st grade|0.001221001221001221|
|           9th grade|0.006099518459069021|
|     5th or 6th grade|0.006713457430576747|
|1st 2nd 3rd or 4t...|0.007226236798221234|
|          10th grade|0.008204313881169776|
|    7th and 8th grade|0.008992131884600974|
|          11th grade|0.010180337405468295|
|12th grade no dip...| 0.01599247412982126|
|High school graduate|0.038816700064040324|
|Some college but ...| 0.06423436376707405|
|Associates degree...|  0.0770810003732736|
|Associates degree...| 0.09443043777217511|
|Bachelors degree(...| 0.19708029197080293|
|Masters degree(MA...|  0.3115731539519951|
|Doctorate degree(...|  0.5201900237529691|
|Prof school degre...|  0.5404350250976018|
+--------------------+--------------------+
```

As we would suspect, prof school degree, phds and masters degrees, hold the top 3 positions. The lower the education level the less is the probability of someone having an income more than 50.000. But, I just saw that in our last position are Children!!!! There are 47422 children in our dataset, as we can see from the following code:

```
>>> data.groupBy('education').count().orderBy('count').show()
+--------------------+-----+
|           education|count|
+--------------------+-----+
| Less than 1st grade|  819|
|Doctorate degree(...| 1263|
|Prof school degre...| 1793|
|1st 2nd 3rd or 4t...| 1799|
|12th grade no dip...| 2126|
|    5th or 6th grade| 3277|
|Associates degree...| 4363|
|Associates degree...| 5358|
|           9th grade| 6230|
|Masters degree(MA...| 6541|
|          11th grade| 6876|
|          10th grade| 7557|
|   7th and 8th grade| 8007|
|Bachelors degree(...|19865|
|Some college but ...|27820|
|            Children|47422|
|High school graduate|48407|
+--------------------+-----+
```

I think I will make a subset of my data, called newdata, filtering my dataset not to contain children

```
>>>
>>>
>>> newdata = data.filter('education != "Children"')
>>>
>>>
>>> newdata.count()
152101
>>>
>>>
>>> newdata.groupBy('education').count().show()
+--------------------+-----+
|           education|count|
+--------------------+-----+
|Some college but ...|27820|
|High school graduate|48407|
|Prof school degre...| 1793|
|Associates degree...| 4363|
|Masters degree(MA...| 6541|
|    5th or 6th grade| 3277|
|1st 2nd 3rd or 4t...| 1799|
|   7th and 8th grade| 8007|
|Doctorate degree(...| 1263|
|Associates degree...| 5358|
|           9th grade| 6230|
|          11th grade| 6876|
|          10th grade| 7557|
|Bachelors degree(...|19865|
|12th grade no dip...| 2126|
| Less than 1st grade|  819|
+--------------------+-----+

>>>
```

As we can verify, the number of our rows has dropped to 152 k, from 195k, and also, the 'Children' category does not appear in our groupby command.

Next on I would like to investigate on 'race' and 'sex' variables

```
>>>
>>> newdata.groupBy('race').count().show()
+--------------------+------+
|                race| count|
+--------------------+------+
|Amer Indian Aleut...|  1502|
|               Other|  2361|
|               White|129560|
|               Black| 14296|
|Asian or Pacific ...|  4382|
+--------------------+------+

>>>
>>>
>>> newdata.groupBy('race').avg('income2').orderBy('avg(income2)').show()
+--------------------+--------------------+
|                race|        avg(income2)|
+--------------------+--------------------+
|Amer Indian Aleut...|0.032623169107856l9|
|               Black|0.03777280358142138|
|               Other| 0.0385429902583651|
|               White|0.08700216116085212|
|Asian or Pacific ...|0.09812870835235052|
+--------------------+--------------------+

>>>
>>>
>>>
>>>
>>> newdata.groupBy('sex').count().show()
+------+-----+
|   sex|count|
+------+-----+
|Female|80639|
|  Male|71462|
+------+-----+

>>>
>>> newdata.groupBy('sex').avg('income2').orderBy('avg(income2)').show()
+------+--------------------+
|   sex|        avg(income2)|
+------+--------------------+
|Female|0.03302372301243815|
|  Male| 0.1360023508997789|
+------+--------------------+

>>>
```

Our dataset is dominated by whites, and 2nd most common race are the blacks. We can see that the avg income (meaning the avg possibility that someone has above 50k) of whites is 0.087 , and that of blacks is 0.037, meaning Whites have 0.087/0.037 = 2.35 times probability of having income more than 50k.

About 'sex', the dataset is equally divided, but men are four times more likely to have an income above 50k

MISSING VALUES

```
>>>
>>> from pyspark.sql.functions import col,sum
>>> newdata.select(*(sum(col(c).isNull().cast("int")).alias(c) for c in newdata.columns)).show()
+---+---------------+-------------+---------------+---------+-------------+--------------------------+--------------+-------------------+---------------------+----+---
|age|class_of_worker|industry_code|occupation_code|education|wage_per_hour|enrolled_in_edu_inst_lastwk|marital_status|major_industry_code|major_occupation_code|race|his
+---+---------------+-------------+---------------+---------+-------------+--------------------------+--------------+-------------------+---------------------+----+---
|  0|              0|            0|              0|        0|            0|                         0|             0|                  0|                    0|   0|
+---+---------------+-------------+---------------+---------+-------------+--------------------------+--------------+-------------------+---------------------+----+---
```

Thankfully there are no missing values

CLUSTERING

I would like to take a look at a possible "grouping", meaning clustering of our individuals, regarding their sex, age, race, education. We will have again to import the appropriate libraries, and perform the necessary manipulations. To perform clustering, we must ensure that our variables are in numeric form. So we know that age is ok, but we should transform sex, race, and education.

```
>> from pyspark.ml.feature import StringIndexer
>>
>>
>> stringIndexer = StringIndexer(inputCol = 'education', outputCol = 'test' )
>>
>>
>> newdata2 = stringIndexer.fit(newdata).transform(newdata)
>>
>>
```

```
>>>
>>> stringIndexer = StringIndexer(inputCol = 'education', outputCol = 'education2' )
>>>
>>> newdata2 = stringIndexer.fit(newdata2).transform(newdata2)
>>>
```

```
>>> stringIndexer = StringIndexer(inputCol = 'race', outputCol = 'race2' )
>>>
>>> newdata2 = stringIndexer.fit(newdata2).transform(newdata2)
```

Up next, I will proceed to clustering, meaning that I first must create a vector of my variables to be considered for clustering.

```
>>> from pyspark.ml.feature import VectorAssembler
>>>
>>> vectorassembler = VectorAssembler(inputCols = ['sex2' , 'education2' , 'race2' , 'age' ] , outputCol = 'clusterfeatures')
>>>
>>>
>>>
>>> datavectorized = vectorassembler.transform(newdata2)
>>>
```

```
+----+-----+----------+------------------+
|sex2|race2|education2|   clusterfeatures|
+----+-----+----------+------------------+
| 0.0|  0.0|       0.0|    (4,[3],[73.0])|
| 1.0|  0.0|       1.0|[1.0,1.0,0.0,58.0]|
| 0.0|  2.0|       4.0|[0.0,4.0,2.0,18.0]|
| 0.0|  4.0|       1.0|[0.0,1.0,4.0,48.0]|
| 1.0|  0.0|       2.0|[1.0,2.0,0.0,42.0]|
| 0.0|  0.0|       0.0|    (4,[3],[28.0])|
| 0.0|  0.0|       1.0|[0.0,1.0,0.0,47.0]|
| 1.0|  0.0|       1.0|[1.0,1.0,0.0,34.0]|
| 0.0|  1.0|       0.0|[0.0,0.0,1.0,32.0]|
| 1.0|  0.0|       1.0|[1.0,1.0,0.0,51.0]|
+----+-----+----------+------------------+
```

Next on I will experiment on the number of clusters I should use. I will start with 3 and we will see

```
>>> from pyspark.ml.clustering  import KMeans
>>> kmeans = KMeans().setK(3)
>>> kmeans = kmeans.setSeed(1)
>>> kmodel = kmeans.fit(datavectorized)
18/09/18 16:22:05 WARN KMeans: The input data is not directly cached, which may hurt performance if its parent RDDs are also uncached.
18/09/18 16:22:29 WARN KMeans: The input data was not directly cached, which may hurt performance if its parent RDDs are also uncached.
>>> predictions = kmodel.transform(datavectorized)
```

Let us see now our results:

```
>>> predictions.groupBy('prediction').count().show()
+----------+-----+
|prediction|count|
+----------+-----+
|         1|64086|
|         2|33537|
|         0|54478|
+----------+-----+
```

As we had set, there are 3 clusters, 0,1,2 . I am curious about two things:

1. Whether those clusters have significant difference in the average probability of income > 50000
2. What are the centers of the clusters

As we recall, the parameters are ['sex2' , 'education2' , 'race2' , 'age']

```
>>> predictions.groupBy('prediction').avg('income2').show()
+----------+-------------------+
|prediction|       avg(income2)|
+----------+-------------------+
|         1|0.04181880597946509|
|         2|0.05423860214091898|
|         0|0.14470061309152318|
+----------+-------------------+

>>> centers = kmodel.clusterCenters()
>>>
>>> print(centers)
[array([  0.47867029,   2.93303719,   0.22717427,  46.04566981]), array([  0.48447399,   2.89153637,   0.29362107,  26.10226883]), array([  0.42749799,   3.19327906,
>>>
```

We can directly see that cluster no 0 has almost three times more probability of having income > 50000, than the other two clusters. It would be interesting to train classifiers for each cluster individually and see if we get a better result.

Also I want to see what happens for 4 or 5 clusters:

4 CLUSTERS

```
>>> from pyspark.ml.clustering  import KMeans
>>> kmeans = KMeans().setK(4)
>>> kmeans = kmeans.setSeed(1)
>>> kmodel = kmeans.fit(datavectorized)
18/09/19 12:03:02 WARN KMeans: The input data is not directly cached, which may hurt performance if its parent RDDs are also uncached.
18/09/19 12:03:23 WARN KMeans: The input data was not directly cached, which may hurt performance if its parent RDDs are also uncached.
>>> predictions = kmodel.transform(datavectorized)
>>>
>>> predictions.groupBy('prediction').avg('income2').show()
+----------+--------------------+
|prediction|        avg(income2)|
+----------+--------------------+
|         1|0.012054590834262652|
|         3| 0.10459019790695265|
|         2| 0.04322408564434214|
|         0| 0.14679891794409378|
+----------+--------------------+

>>>
```

Groups 3 and 0 have a substantial difference, compared to the other two.

5 CLUSTERS

```
>>> from pyspark.ml.clustering  import KMeans
>>> kmeans = KMeans().setK(5)  # num of clusters
>>> kmeans = kmeans.setSeed(1)
>>> kmodel = kmeans.fit(datavectorized)
18/09/19 12:09:17 WARN KMeans: The input data is not directly cached, which may hurt performance if its parent RDDs are also uncached.
18/09/19 12:09:46 WARN KMeans: The input data was not directly cached, which may hurt performance if its parent RDDs are also uncached.
>>> predictions = kmodel.transform(datavectorized)
>>>
>>>
>>> predictions.groupBy('prediction').avg('income2').show()
+----------+--------------------+
|prediction|        avg(income2)|
+----------+--------------------+
|         1|0.009681044169764024|
|         3| 0.09340176943337611|
|         4| 0.03081971432248102|
|         2|  0.0857504252627906|
|         0| 0.15801066543009507|
+----------+--------------------+

>>> predictions.groupBy('prediction').count().show()
+----------+-----+
|prediction|count|
+----------+-----+
|         1|34707|
|         3|44421|
|         4|15542|
|         2|22927|
|         0|34504|
+----------+-----+

>>>
```

Cluster no 0 has 34504 individuals and also has the highest percentage of high income. Further investigation of the features of these clusters, could lead to better results.

It is time now to train a random forest model, and see how precise we can be, predicting the minority class. At first we should label the categorical features that we choose to use for our model.

```
>>>
>>> indexer = StringIndexer(inputCol = 'marital_status' , outputCol = 'maritalstatus1')
>>> newdata = indexer.fit(newdata).transform(newdata)
>>>
>>> indexer = StringIndexer(inputCol = 'major_industry_code' , outputCol = 'majorindustrycode1')
>>> newdata = indexer.fit(newdata).transform(newdata)
>>>
>>> indexer = StringIndexer(inputCol = 'race' , outputCol = 'race1')
>>> newdata = indexer.fit(newdata).transform(newdata)
>>>
>>>
>>> indexer = StringIndexer(inputCol = 'major_occupation_code' , outputCol = 'majoroccupationcode1')
>>> newdata = indexer.fit(newdata).transform(newdata)
>>>
>>> indexer = StringIndexer(inputCol = 'sex' , outputCol = 'sex1')
>>> newdata = indexer.fit(newdata).transform(newdata)
>>>
>>>
```

This is how they look after the transformation

classofworker1	education1	maritalstatus1	majorindustrycode1	race1	majoroccupationcode1	sex1
1.0	0.0	3.0	0.0	0.0	0.0	0.0
2.0	1.0	2.0	6.0	0.0	6.0	1.0
1.0	4.0	1.0	0.0	2.0	0.0	0.0
0.0	1.0	0.0	17.0	4.0	2.0	0.0
0.0	2.0	0.0	5.0	0.0	3.0	1.0
0.0	0.0	1.0	6.0	0.0	8.0	0.0
3.0	1.0	0.0	3.0	0.0	1.0	0.0
0.0	1.0	0.0	6.0	0.0	7.0	1.0
1.0	0.0	1.0	0.0	1.0	0.0	0.0
0.0	1.0	0.0	6.0	0.0	6.0	1.0

Now we must proceed to preparing our model. At first I will create the vector of features I will use.

```
>>>
>>> from pyspark.ml.feature import VectorAssembler
>>>
>>>
>>> assembler = VectorAssembler(inputCols = ['age' , 'classofworker1', 'education1', 'maritalstatus1', 'majorindustrycode1', 'race1', 'majoroccupationcode1', 'sex1']
>>>
>>>
>>> newdata = assembler.transform(newdata)
>>>
>>> newdata.show(10)
```

```
+--------------------+
|            features|
+--------------------+
|(8,[0,1,3],[73.0,...|
|[58.0,2.0,1.0,2.0...|
|[18.0,1.0,4.0,1.0...|
|[48.0,0.0,1.0,0.0...|
|[42.0,0.0,2.0,0.0...|
|(8,[0,3,4,6],[28....|
|[47.0,3.0,1.0,0.0...|
|[34.0,0.0,1.0,0.0...|
|(8,[0,1,3,5],[32....|
|[51.0,0.0,1.0,0.0...|
+--------------------+
```

Next on, I will label my target column, using again StringIndexer

```
>>> targetindexer  = StringIndexer( inputCol = 'income2' , outputCol = 'label')
>>>
>>> newdata = targetindexer.fit(newdata).transform(newdata)
>>>
>>>
>>> newdata.show(10)
```

Finally let us deploy all the necessary steps:

```
>>> ########Importing the classifier#########
...
>>> from pyspark.ml.classification import RandomForestClassifier
>>>
>>> classifier = RandomForestClassifier()
>>>
>>> # splitting my dataset at a ratio of 80-20
...
>>> splits = newdata.randomSplit([0.7,0.25] ,12)
>>> train = splits[0]
>>> test = splits[1]
>>>
>>>
>>>
>>>
>>> ######## Splitting my dataset at a ratio of 0.75 - 0.25  #################
...
>>> splits = newdata.randomSplit([0.75,0.25] ,12)
>>> train = splits[0]
>>> test = splits[1]
>>>
>>>
>>>
>>> ###### Fitting the classifier to the train set  ##########
... model = classifier.fit(train)
>>>
>>>
>>> ########## Making predictions on the test set    ##########
... predictionsdf = model.transform(test)
>>>
```

FEATURE IMPORTANCES

```
>> model.featureImportances
parseVector(8, {0: 0.1103, 1: 0.0242, 2: 0.2405, 3: 0.0302, 4: 0.0404, 5: 0.0003, 6: 0.3801, 7: 0.1741})
>>
>>
```

6, 2, and 7 are the top3 features in our dataset, meaning major occupation code, education and sex. To my personal opinion, it makes quite sense.

EVALUATION OF MODEL

We will use the built in roc evaluation

```
>>>
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>>
>>> evaluator = MulticlassClassificationEvaluator()
>>>
>>> roc = evaluator.evaluate(predictionsdf)
>>>
>>> print(roc)
0.8972865098248474
>>>
>>>
>>> #########   Manual Calculation of Precision ########
... predictionsdf.groupBy('prediction' , 'label').count().show()
+----------+-----+-----+
|prediction|label|count|
+----------+-----+-----+
|       1.0|  1.0|  325|
|       0.0|  1.0| 2759|
|       1.0|  0.0|  101|
|       0.0|  0.0|34690|
+----------+-----+-----+

>>>
```

Let us calculate the precision manually

precision = 325/(325+101) = 0.763 , very decent for such an imbalanced dataset.

Next on we will move further with the evaluation of our classifier, applying grid search. I am going to select two hyperparameters that are usually the most influencial, number of estimators and max depth of the tree

```
>>>
>>> ########## GRID SEARCH  ##############
...
>>> from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
>>>
>>> paramgrid = (ParamGridBuilder().addGrid(classifier.maxDepth, [3,5,7,9]).addGrid(classifier.numTrees ,[30, 100, 200]).build())
>>>
>>> cv = CrossValidator(estimator=classifier, estimatorParamMaps=paramgrid, evaluator=evaluator, numFolds=5)
>>>
>>> cvmodel = cv.fit(train)
>>>
>>> gridpredictions = cvmodel.transform(test)
>>>
>>>
>>> roc = evaluator.evaluate(gridpredictions)
>>>
>>>
>>> print(roc)
0.9165743997374877
>>>
>>> cvmodel.bestModel
RandomForestClassificationModel (uid=rfc_d41f443d1336) with 200 trees
>>>
>>>
>>>
>>>
```

Our roc score, indeed improved from 89.7 % to 91.7 %, and we also have the information that the best number of trees was 200.