Homework #2 **Due: turned in by Monday 01/29/2018 before class**

__Xi (Athena) Li__

(put your name above)

Total grade: _____ out of ___100___ points

There are 2 parts in this homework assignment with 4 numbered questions in total. Please answer them all and submit your assignment as a single PDF file by uploading it to the HW2 drop-box on the course website.

If you use others' work as part of your answer, please properly cite your source.

Part I. Short Answers (45 points; 15 points each)

1. **What is a Spark Dataframe? How one should choose between Spark Dataframes and Spark RDDs (or the relative advantages of each)?**
    a. **What is a Spark Dataframe?**
        i. Spark is an open-source, cluster computing framework originally developed as a research project UC Berkeley's AMPLab
        ii. Developed with a focus on interactive, iterative computations
        iii. Utilizes in-memory processing
        iv. Makes it ideal for data science applications (data mining, machine learning)
        v. Source written in Scala
        vi. Functional programming language that runs in a JVM
        vii. Extensive API support for Java, Scala and Python
        viii. Interactive shell for Scala and Python
    b. choose between Spark Dataframes and Spark RDDs
        i. RDDs are immutable
        ii. RDDs can be created from objects in memory, files on disk and other RDDs
        iii. A pipeline typically consists of consecutive operations that create new RDDs from previous ones
        iv. A collection of elements partitioned across the nodes of the cluster
        v. Operated on in parallel
        vi. Elements can be of any type
        vii. The word resilient refers to the fault tolerance that is built in to RDDs

2. **What are the main differences between pyspark.ml and pyspark.mllib? Name at least three differences.**
    a. **Spark MLlib is Spark's machine learning library**
        i. **Makes practical machine learning scalable and easy**
        ii. **Includes many common machine learning algorithms**
        iii. **Includes base data types for efficient calculations at scale**
        iv. **Supports scalable statistics and data transformations**
    b. **Spark ML is a new higher-level API for machine learning pipelines**
        i. **Built on top of Spark's DataFrames API**
        ii. **Simple and clean interface for running series of complex tasks**
        iii. **Supports most functionality included in Spark MLlib**

3. **Compare Scikit-learn and Spark MLlib and discuss their strengths and use cases.**
    a. Scikit-Learn has fantastic performance if your data fits into RAM. Python and Scikit-Learn do in-memory processing and in a non-distributed fashion.

    b. Spark's ML Lib is suitable when you're doing relatively simple ML on a large data set. ML Lib is not computationally efficient for small data sets, and you're better off using scikit-learn for small and medium sized data sets

    c. Scikit-learn has support for Pandas and Matplotlib, which makes the process of developing machine learning models very iterative and efficient. You can visualize results, verify assumptions and use many scipy functions (such as normality tests or distribution fits) where required, as part of your machine learning workflow

Part II. Hands on (55 points)

1. Analyze the Titanic Dataset using pyspark.ml (55 points)

This is a popular dataset for machine learning. A description of the dataset can be found at https://www.kaggle.com/c/titanic/data (our dataset corresponds to the train.csv file from Kaggle). You should train a logistic regression model using this dataset and the pyspark.ml package. The goal is to predict for each passenger whether he/she survive the Titanic tragedy as well as to use the pipeline and feature functionality of pyspark.ml.

Step 1 (6 points): The goal of this step is to read the data from the local folder. You should infer the schema (e.g., columns) from the csv file. Tip: explore the spark_csv package from databricks.

Step 2 (5 points): The goal of this step is to familiarize yourself with the dataset. This step is useful in detecting data problems, informing the data engineering steps, and informing the feature selection processes. You should:

i) Print the dataset and verify that the schema contains all the variables.
ii) Print the first 10 rows from the dataset.
iii) Obtain summary statistics for all variables in the dataframe. Pay attention to whether there are missing data as well as whether the field appears to be continuous or discrete.
iv) For each of the string columns (except name and ticket), print the count of the 10 most frequent values ordered by descending order of frequency.
v) Based on the above, which columns would you keep as features and which would you drop? Justify your answer.
    i. I will drop Cabin column, because it has the most null values and its values are too sparse.

Step 3 (6 points): The goal of this step is to engineer the necessary features for the machine learning model. You should:

i) Select all feature columns you plan to use in addition to the target variable (i.e., 'Survived') and covert all numerical columns into double data type. Tip: you can use the .cast() from pyspark.sql.functions.
ii) Replace the missing values in the Age column with the mean value. Create also a new variable (e.g., 'AgeNA') indicating whether the value of age was missing or not.
iii) Print the revised dataframe and recalculate the summary statistics.

Step 4 (6 points): The goal of this step is to encode all string and categorical variables in order to use them in the pipeline afterwards. You should:

i) Import all necessary pyspark functions.
ii) Create indexers and encoders for categorical string variables. Call them [field]_indexer and [field]_encoder, respectively. For instance, gender_indexer and gender_encoder.

Step 5 (4 points): The goal of this step is to assemble all feature columns into a feature vector in order to be used in the pipeline. Tip: you can use the VectorAssembler to do this.

Step 6 (4 points): The goal of this step is to create the logistic regression model to be used in the pipeline.

Step 7 (4 points): The goal of this step is to assemble the pipeline.

Step 8 (6 points): The goal of this step is to prepare the training and test datasets. You should:

i) Use a 70-30 random split for the training and test sets, respectively.
ii) Verify the size of each dataset after the split.

Step 9 (8 points): The goal of this step is to fit the model and then use it on the test set to generate predictions. You should:

i) Fit the model using the predefined pipeline on the training set.
ii) Use the fitted model for prediction on the test set.
iii) Report the logistic regression coefficients.
iv) Interpret the obtained coefficients.

Step 10 (6 points): The goal of this step is to evaluate the model performance. You should:

i) Print the first 5 rows of the results.
ii) Report the AUC for this model.

Detailed steps are at the next pages

Step 1: import packages, read the data from the local folder

In [1]:
```
import time
import pyspark
import pandas as pd
import numpy as np
from pyspark import SparkContext, SparkConf
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.feature import StandardScaler
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.sql import Row
from pyspark.ml.feature import StringIndexer, VectorAssembler, OneHotEncoder
from pyspark.ml.classification import LogisticRegression
```

In [19]:
```
sqlContext = SQLContext(sc)
```

In [39]:
```
df = sqlContext.read.format('com.databricks.spark.csv').options(header='true
#rawdata = sc.textFile("hdfs:///user/training/mldata/titanic_train.csv")
```

Step 2: familiarize yourself with the dataset

In [41]:
```
df.printSchema()
```

```
root
 |-- PassengerId: string (nullable = true)
 |-- Survived: string (nullable = true)
 |-- Pclass: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- SibSp: string (nullable = true)
 |-- Parch: string (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: string (nullable = true)
 |-- Cabin: string (nullable = true)
 |-- Embarked: string (nullable = true)
```

In [42]:
```
df.head(10)
```

Out[42]:
```
[Row(PassengerId=u'1', Survived=u'0', Pclass=u'3', Name=u'Braund, Mr. Owe
n Harris', Sex=u'male', Age=u'22', SibSp=u'1', Parch=u'0', Ticket=u'A/5 2
1171', Fare=u'7.25', Cabin=None, Embarked=u'S'),
 Row(PassengerId=u'2', Survived=u'1', Pclass=u'1', Name=u'Cumings, Mrs. J
ohn Bradley (Florence Briggs Thayer)', Sex=u'female', Age=u'38', SibSp=
u'1', Parch=u'0', Ticket=u'PC 17599', Fare=u'71.2833', Cabin=u'C85', Emba
rked=u'C'),
 Row(PassengerId=u'3', Survived=u'1', Pclass=u'3', Name=u'Heikkinen, Mis
s. Laina', Sex=u'female', Age=u'26', SibSp=u'0', Parch=u'0', Ticket=u'STO
N/O2. 3101282', Fare=u'7.925', Cabin=None, Embarked=u'S'),
 Row(PassengerId=u'4', Survived=u'1', Pclass=u'1', Name=u'Futrelle, Mrs.
Jacques Heath (Lily May Peel)', Sex=u'female', Age=u'35', SibSp=u'1', Par
ch=u'0', Ticket=u'113803', Fare=u'53.1', Cabin=u'C123', Embarked=u'S'),
 Row(PassengerId=u'5', Survived=u'0', Pclass=u'3', Name=u'Allen, Mr. Will
iam Henry', Sex=u'male', Age=u'35', SibSp=u'0', Parch=u'0', Ticket=u'3734
50', Fare=u'8.05', Cabin=None, Embarked=u'S'),
 Row(PassengerId=u'6', Survived=u'0', Pclass=u'3', Name=u'Moran, Mr. Jame
s', Sex=u'male', Age=None, SibSp=u'0', Parch=u'0', Ticket=u'330877', Fare
=u'8.4583', Cabin=None, Embarked=u'Q'),
 Row(PassengerId=u'7', Survived=u'0', Pclass=u'1', Name=u'McCarthy, Mr. T
imothy J', Sex=u'male', Age=u'54', SibSp=u'0', Parch=u'0', Ticket=u'1746
3', Fare=u'51.8625', Cabin=u'E46', Embarked=u'S'),
 Row(PassengerId=u'8', Survived=u'0', Pclass=u'3', Name=u'Palsson, Maste
r. Gosta Leonard', Sex=u'male', Age=u'2', SibSp=u'3', Parch=u'1', Ticket=
u'349909', Fare=u'21.075', Cabin=None, Embarked=u'S'),
 Row(PassengerId=u'9', Survived=u'1', Pclass=u'3', Name=u'Johnson, Mrs. O
scar W (Elisabeth Vilhelmina Berg)', Sex=u'female', Age=u'27', SibSp=
u'0', Parch=u'2', Ticket=u'347742', Fare=u'11.1333', Cabin=None, Embarked
=u'S'),
 Row(PassengerId=u'10', Survived=u'1', Pclass=u'2', Name=u'Nasser, Mrs. N
icholas (Adele Achem)', Sex=u'female', Age=u'14', SibSp=u'1', Parch=u'0',
Ticket=u'237736', Fare=u'30.0708', Cabin=None, Embarked=u'C')]
```

In [46]:
```
df.describe().show()
```

```
+-------+-----------------+-------------------+-----------------+-------
-------------+------+-----------------+-----------------+-------------
-----+-----------------+-----------------+-----+--------+
|summary|      PassengerId|           Survived|           Pclass|
        Name|   Sex|              Age|            SibSp|
Parch|           Ticket|             Fare|Cabin|Embarked|
+-------+-----------------+-------------------+-----------------+-------
-------------+------+-----------------+-----------------+-------------
-----+-----------------+-----------------+-----+--------+
|  count|              891|                891|              891|
         891|   891|              714|              891|
  891|              891|              891|  204|     889|
|   mean|            446.0| 0.3838383838383838| 2.308641975308642|
        null|  null| 29.69911764705882|0.5230078563411896|0.381593714927
04824|260318.54916792738| 32.2042079685746| null|    null|
| stddev|257.3538420152301|0.48659245426485753|0.8360712409770491|
        null|  null|14.526497332334035|1.1027434322934315| 0.80605722112
99488|471609.26868834975|49.69342859718089| null|    null|
|    min|                1|                  0|                1|"Anders
son, Mr. A...|female|             0.42|                0|
    0|            110152|                0|  A10|       C|
|    max|               99|                  1|                3|van Mel
kebeke, Mr...|  male|                9|                8|
    6|         WE/P 5735|             93.5|    T|       S|
+-------+-----------------+-------------------+-----------------+-------
-------------+------+-----------------+-----------------+-------------
-----+-----------------+-----------------+-----+--------+
```

In [56]:
```
import pandas as pd
dfp = df.toPandas()
```

In [58]:
```
dfp.head(10)
```

Out[58]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	Non
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C8
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925	Non
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C12
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05	Non
5	6	0	3	Moran, Mr. James	male	None	0	0	330877	8.4583	Non
6	7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E4
7	8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075	Non
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333	Non
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708	Non

In [60]:
```
#summary of the dataframe
summary = dfp.describe()
summary.transpose()
```

Out[60]:

	count	unique	top	freq
PassengerId	891	891	823	1
Survived	891	2	0	549
Pclass	891	3	3	491
Name	891	891	Graham, Mr. George Edward	1
Sex	891	2	male	577
Age	714	88	24	30
SibSp	891	7	0	608
Parch	891	7	0	678
Ticket	891	681	CA. 2343	7
Fare	891	248	8.05	43
Cabin	204	147	C23 C25 C27	4
Embarked	889	3	S	644

In [61]:
```
#only age, cabin, and embarked has null values
dfp.isnull().sum()
```

Out[61]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [75]:
```
ctsex = dfp['Sex'].value_counts()
ctcabin = dfp['Cabin'].value_counts()
ctembark = dfp['Embarked'].value_counts()
ctsurvived = dfp['Survived'].value_counts()
ctPclass = dfp['Pclass'].value_counts()
ctage = dfp['Age'].value_counts()
ctsibsp = dfp['SibSp'].value_counts()
ctparch = dfp['Parch'].value_counts()
ctsex.head(10)
```

Out[75]:
```
male      577
female    314
Name: Sex, dtype: int64
```

In [63]:
```
ctcabin.head(10)
#I will drop Cabin column, because it has the most null values and its value
```

Out[63]:
```
C23 C25 C27        4
G6                 4
B96 B98            4
D                  3
C22 C26            3
E101               3
F2                 3
F33                3
B57 B59 B63 B66    2
C68                2
Name: Cabin, dtype: int64
```

In [64]:
```
ctembark.head(10)
```

Out[64]:
```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

In [76]:
```
ctsurvived.head(10)
```

Out[76]:
```
0    549
1    342
Name: Survived, dtype: int64
```

In [77]:
```
ctPclass.head(10)
```

Out[77]:
```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

In [79]:
```
ctage.head(10)
```

Out[79]:
```
24    30
22    27
18    26
30    25
28    25
19    25
21    24
25    23
36    22
29    20
Name: Age, dtype: int64
```

In [80]:
```
ctsibsp.head(10)
```

Out[80]:
```
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

In [81]:
```
ctparch.head(10)
```

Out[81]:
```
0    678
1    118
2     80
5      5
3      5
4      4
6      1
Name: Parch, dtype: int64
```

In [66]:
```
dfsl = sqlContext.createDataFrame(dfp.iloc[:,[1,2,4,5,6,7,9,11]]).show()
```

```
+--------+------+------+----+-----+-----+-------+--------+
|Survived|Pclass|   Sex| Age|SibSp|Parch|   Fare|Embarked|
+--------+------+------+----+-----+-----+-------+--------+
|       0|     3|  male|  22|    1|    0|   7.25|       S|
|       1|     1|female|  38|    1|    0|71.2833|       C|
|       1|     3|female|  26|    0|    0|  7.925|       S|
|       1|     1|female|  35|    1|    0|   53.1|       S|
|       0|     3|  male|  35|    0|    0|   8.05|       S|
|       0|     3|  male|null|    0|    0| 8.4583|       Q|
|       0|     1|  male|  54|    0|    0|51.8625|       S|
|       0|     3|  male|   2|    3|    1| 21.075|       S|
|       1|     3|female|  27|    0|    2|11.1333|       S|
|       1|     2|female|  14|    1|    0|30.0708|       C|
|       1|     3|female|   4|    1|    1|   16.7|       S|
|       1|     1|female|  58|    0|    0|  26.55|       S|
|       0|     3|  male|  20|    0|    0|   8.05|       S|
|       0|     3|  male|  39|    1|    5| 31.275|       S|
|       0|     3|female|  14|    0|    0| 7.8542|       S|
|       1|     2|female|  55|    0|    0|     16|       S|
|       0|     3|  male|   2|    4|    1| 29.125|       Q|
|       1|     2|  male|null|    0|    0|     13|       S|
|       0|     3|female|  31|    1|    0|     18|       S|
|       1|     3|female|null|    0|    0|  7.225|       C|
+--------+------+------+----+-----+-----+-------+--------+
only showing top 20 rows
```

Step 3:engineer the necessary features for the machine learning model

In [120]:
```
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

In [143]:
```
# String to double on some columns of the dataset : creates a new dataset
dfls = df.select(col("Survived").cast("double"),col("Sex"),col("Embarked"),c
```

In [144]:
```
#now we take a look at the new dataframe with selected columns converted to
dfls.printSchema()
```

```
root
 |-- Survived: double (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Embarked: string (nullable = true)
 |-- Parch: double (nullable = true)
 |-- Pclass: double (nullable = true)
 |-- Age: double (nullable = true)
 |-- SibSp: double (nullable = true)
 |-- Fare: double (nullable = true)
 |-- AgeNA: double (nullable = true)
```

In [145]:
```
avgage = dfls.select([mean('Age')]).show()
```

```
+-----------------+
|         avg(Age)|
+-----------------+
|29.69911764705882|
+-----------------+
```

In [159]:
```
#Replace the missing values in the Age column with the mean value
dfillna = dfls.na.fill({'Age': 29.7,'AgeNA': 1})
dfillna.show()
```

```
+--------+------+--------+-----+------+----+-----+-------+-----+
|Survived|   Sex|Embarked|Parch|Pclass| Age|SibSp|   Fare|AgeNA|
+--------+------+--------+-----+------+----+-----+-------+-----+
|     0.0|  male|       S|  0.0|   3.0|22.0|  1.0|   7.25| 22.0|
|     1.0|female|       C|  0.0|   1.0|38.0|  1.0|71.2833| 38.0|
|     1.0|female|       S|  0.0|   3.0|26.0|  0.0|  7.925| 26.0|
|     1.0|female|       S|  0.0|   1.0|35.0|  1.0|   53.1| 35.0|
|     0.0|  male|       S|  0.0|   3.0|35.0|  0.0|   8.05| 35.0|
|     0.0|  male|       Q|  0.0|   3.0|29.7|  0.0| 8.4583|  1.0|
|     0.0|  male|       S|  0.0|   1.0|54.0|  0.0|51.8625| 54.0|
|     0.0|  male|       S|  1.0|   3.0| 2.0|  3.0| 21.075|  2.0|
|     1.0|female|       S|  2.0|   3.0|27.0|  0.0|11.1333| 27.0|
|     1.0|female|       C|  0.0|   2.0|14.0|  1.0|30.0708| 14.0|
|     1.0|female|       S|  1.0|   3.0| 4.0|  1.0|   16.7|  4.0|
|     1.0|female|       S|  0.0|   1.0|58.0|  0.0|  26.55| 58.0|
|     0.0|  male|       S|  0.0|   3.0|20.0|  0.0|   8.05| 20.0|
|     0.0|  male|       S|  5.0|   3.0|39.0|  1.0| 31.275| 39.0|
|     0.0|female|       S|  0.0|   3.0|14.0|  0.0| 7.8542| 14.0|
|     1.0|female|       S|  0.0|   2.0|55.0|  0.0|   16.0| 55.0|
|     0.0|  male|       Q|  1.0|   3.0| 2.0|  4.0| 29.125|  2.0|
|     1.0|  male|       S|  0.0|   2.0|29.7|  0.0|   13.0|  1.0|
|     0.0|female|       S|  0.0|   3.0|31.0|  1.0|   18.0| 31.0|
|     1.0|female|       C|  0.0|   3.0|29.7|  0.0|  7.225|  1.0|
+--------+------+--------+-----+------+----+-----+-------+-----+
only showing top 20 rows
```

In [168]:
```
newdf = dfillna.select(col("Survived"),col("Sex"),col("Embarked"),col("Parch
when(dfillna["Age"] != 29.7, 0).otherwise(1).alias("AgeNA"))
newdf.show()
```

```
+--------+------+--------+-----+------+----+-----+-------+-----+
|Survived|   Sex|Embarked|Parch|Pclass| Age|SibSp|   Fare|AgeNA|
+--------+------+--------+-----+------+----+-----+-------+-----+
|     0.0|  male|       S|  0.0|   3.0|22.0|  1.0|   7.25|    0|
|     1.0|female|       C|  0.0|   1.0|38.0|  1.0|71.2833|    0|
|     1.0|female|       S|  0.0|   3.0|26.0|  0.0|  7.925|    0|
|     1.0|female|       S|  0.0|   1.0|35.0|  1.0|   53.1|    0|
|     0.0|  male|       S|  0.0|   3.0|35.0|  0.0|   8.05|    0|
|     0.0|  male|       Q|  0.0|   3.0|29.7|  0.0| 8.4583|    1|
|     0.0|  male|       S|  0.0|   1.0|54.0|  0.0|51.8625|    0|
|     0.0|  male|       S|  1.0|   3.0| 2.0|  3.0| 21.075|    0|
|     1.0|female|       S|  2.0|   3.0|27.0|  0.0|11.1333|    0|
|     1.0|female|       C|  0.0|   2.0|14.0|  1.0|30.0708|    0|
|     1.0|female|       S|  1.0|   3.0| 4.0|  1.0|   16.7|    0|
|     1.0|female|       S|  0.0|   1.0|58.0|  0.0|  26.55|    0|
|     0.0|  male|       S|  0.0|   3.0|20.0|  0.0|   8.05|    0|
|     0.0|  male|       S|  5.0|   3.0|39.0|  1.0| 31.275|    0|
|     0.0|female|       S|  0.0|   3.0|14.0|  0.0| 7.8542|    0|
|     1.0|female|       S|  0.0|   2.0|55.0|  0.0|   16.0|    0|
|     0.0|  male|       Q|  1.0|   3.0| 2.0|  4.0| 29.125|    0|
|     1.0|  male|       S|  0.0|   2.0|29.7|  0.0|   13.0|    1|
|     0.0|female|       S|  0.0|   3.0|31.0|  1.0|   18.0|    0|
|     1.0|female|       C|  0.0|   3.0|29.7|  0.0|  7.225|    1|
+--------+------+--------+-----+------+----+-----+-------+-----+
only showing top 20 rows
```

In [170]:
```
#recalculate the summary statistics.
summarynew = newdf.toPandas().describe()
summarynew.transpose()
```

Out[170]:

	count	mean	std	min	25%	50%	75%	max
Survived	891.00	0.38	0.49	0.00	0.00	0.00	1.00	1.00
Parch	891.00	0.38	0.81	0.00	0.00	0.00	0.00	6.00
Pclass	891.00	2.31	0.84	1.00	2.00	3.00	3.00	3.00
Age	891.00	29.70	13.00	0.42	22.00	29.70	35.00	80.00
SibSp	891.00	0.52	1.10	0.00	0.00	0.00	1.00	8.00
Fare	891.00	32.20	49.69	0.00	7.91	14.45	31.00	512.33
AgeNA	891.00	0.20	0.40	0.00	0.00	0.00	0.00	1.00

Step 4: Encode all string and categorical variables in order to use them in the pipeline afterwards

In [198]:
```
#Create StringIndexer transformers; StringIndexer encodes a string column o
indexedcols = ["sexVec","embarkedVec","Parch", "Pclass", "Age","SibSp","Fare
sex_indexer = StringIndexer(inputCol="Sex",outputCol="indexedSex").fit(newdf
embarked_indexer = StringIndexer(inputCol="Embarked",outputCol="indexedEmbar
surviveIndexer = StringIndexer(inputCol="Survived", outputCol="indexedSurviv
```

In [199]:
```
# One Hot Encoder on indexed features
sex_encoder = OneHotEncoder(inputCol="indexedSex", outputCol="sexVec")
embarked_encoder = OneHotEncoder(inputCol="indexedEmbarked", outputCol="emba
```

Step 5: Assemble all feature columns into a feature vector in order to be used in the pipeline

In [200]:
```
va = VectorAssembler(inputCols = indexedcols, outputCol = 'features')
```

Step 6: Create the logistic regression model to be used in the pipeline

In [208]:
```
# Train a Logistic regression model.
lgm = LogisticRegression(labelCol="indexedSurvived", featuresCol="features",
```

Step 7: assemble the pipeline

In [209]:
```
# Chain indexers and logistic regression actions in a Pipeline
pipeline = Pipeline(stages=[surviveIndexer, sex_indexer, embarked_indexer, s
```

Step 8: prepare the training and test datasets

In [210]:
```
# splitting dataset into train and test set
(trainData, testData) = newdf.randomSplit([0.7, 0.3])
```

In [211]:
```
#0.7 of the original data set
trainData.count()
```

Out[211]: 617

In [212]:
```
#0.3 of the data set
testData.count()
```

Out[212]: 274

In [207]:
```
trainData.show()
```

```
+--------+------+--------+-----+------+----+-----+-------+-----+
|Survived|   Sex|Embarked|Parch|Pclass| Age|SibSp|   Fare|AgeNA|
+--------+------+--------+-----+------+----+-----+-------+-----+
|     0.0|female|       C|  0.0|   1.0|50.0|  0.0|28.7125|    0|
|     0.0|female|       C|  0.0|   3.0|14.5|  1.0|14.4542|    0|
|     0.0|female|       C|  0.0|   3.0|29.7|  1.0|14.4542|    1|
|     0.0|female|       C|  1.0|   3.0|18.0|  0.0|14.4542|    0|
|     0.0|female|       C|  2.0|   3.0|29.7|  0.0|15.2458|    1|
|     0.0|female|       Q|  0.0|   3.0|18.0|  0.0|   6.75|    0|
|     0.0|female|       Q|  0.0|   3.0|21.0|  0.0|   7.75|    0|
|     0.0|female|       Q|  0.0|   3.0|29.7|  0.0| 7.6292|    1|
|     0.0|female|       Q|  0.0|   3.0|29.7|  0.0|   7.75|    1|
|     0.0|female|       Q|  0.0|   3.0|30.5|  0.0|   7.75|    0|
|     0.0|female|       Q|  1.0|   3.0|32.0|  1.0|   15.5|    0|
|     0.0|female|       Q|  2.0|   3.0|29.7|  0.0|   7.75|    1|
|     0.0|female|       Q|  5.0|   3.0|39.0|  0.0| 29.125|    0|
|     0.0|female|       S|  0.0|   2.0|24.0|  0.0|   13.0|    0|
|     0.0|female|       S|  0.0|   2.0|38.0|  0.0|   13.0|    0|
|     0.0|female|       S|  0.0|   2.0|57.0|  0.0|   10.5|    0|
|     0.0|female|       S|  0.0|   3.0|18.0|  0.0|  7.775|    0|
|     0.0|female|       S|  0.0|   3.0|18.0|  1.0|   17.8|    0|
|     0.0|female|       S|  0.0|   3.0|18.0|  2.0|   18.0|    0|
|     0.0|female|       S|  0.0|   3.0|20.0|  1.0|  9.825|    0|
+--------+------+--------+-----+------+----+-----+-------+-----+
only showing top 20 rows
```

Step 9: Fit the model and then use it on the test set to generate predictions

In [267]:
```
model = pipeline.fit(trainData)
```

In [284]:
```
#logistic regression coefficients
#the odds of your outcome for survived are exp(-2.7938) = 1.01 times that o
[stage.coefficients for stage in model.stages if hasattr(stage, "coefficient
```

Out[284]:
```
[DenseVector([-2.7938, 1.1055, 1.2704, -0.0512, -0.4872, -0.0395, -0.466
2, 0.0144, 0.1406])]
```

In [268]:
```
predictions = model.transform(testData)
```

In [269]:
```
predictions.printSchema()
```

```
root
 |-- Survived: double (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Embarked: string (nullable = true)
 |-- Parch: double (nullable = true)
 |-- Pclass: double (nullable = true)
 |-- Age: double (nullable = false)
 |-- SibSp: double (nullable = true)
 |-- Fare: double (nullable = true)
 |-- AgeNA: integer (nullable = false)
 |-- indexedSurvived: double (nullable = true)
 |-- indexedSex: double (nullable = true)
 |-- indexedEmbarked: double (nullable = true)
 |-- sexVec: vector (nullable = true)
 |-- embarkedVec: vector (nullable = true)
 |-- features: vector (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = true)
```

In [271]:
```
selected = predictions.select(col("Survived").cast("double"),col("predictior
```

Step 10: evaluate the model performance

In [286]:
```
predictions.show(5)
```

```
+--------+------+--------+-----+------+----+-----+-------+-----+---------
------+----------+---------------+---------+-------------+--------------
-----+--------------------+--------------------+----------+
|Survived|   Sex|Embarked|Parch|Pclass| Age|SibSp|   Fare|AgeNA|indexedSu
rvived|indexedSex|indexedEmbarked|   sexVec|  embarkedVec|            fea
tures|       rawPrediction|         probability|prediction|
+--------+------+--------+-----+------+----+-----+-------+-----+---------
------+----------+---------------+---------+-------------+--------------
-----+--------------------+--------------------+----------+
|     0.0|female|       C|  0.0|   3.0|29.7|  1.0|14.4542|    1|
   0.0|       1.0|            1.0|(1,[],[])|(2,[1],[1.0])|[0.0,0.0,1.0,0.
0,...|[-0.7732442944415...|[0.31577771844406...|       1.0|
|     0.0|female|       C|  0.0|   3.0|29.7|  1.0|14.4583|    1|
   0.0|       1.0|            1.0|(1,[],[])|(2,[1],[1.0])|[0.0,0.0,1.0,0.
0,...|[-0.7733033914297...|[0.31576494996067...|       1.0|
|     0.0|female|       C|  1.0|   3.0|18.0|  0.0|14.4542|    0|
   0.0|       1.0|            1.0|(1,[],[])|(2,[1],[1.0])|[0.0,0.0,1.0,1.
0,...|[-1.5102891888670...|[0.18089593933195...|       1.0|
|     0.0|female|       C|  1.0|   3.0|45.0|  0.0|14.4542|    0|
   0.0|       1.0|            1.0|(1,[],[])|(2,[1],[1.0])|[0.0,0.0,1.0,1.
0,...|[-0.4426642224319...|[0.39110632180397...|       1.0|
|     0.0|female|       C|  2.0|   3.0|29.7|  0.0|15.2458|    1|
   0.0|       1.0|            1.0|(1,[],[])|(2,[1],[1.0])|[0.0,0.0,1.0,2.
0,...|[-1.1484582288282...|[0.24077080651554...|       1.0|
+--------+------+--------+-----+------+----+-----+-------+-----+---------
------+----------+---------------+---------+-------------+--------------
-----+--------------------+--------------------+----------+
only showing top 5 rows
```

In [277]:
```
evaluator = BinaryClassificationEvaluator().setLabelCol("indexedSurvived").s
```

In [278]:
```
#View model's predictions and probabilities of each prediction class
#Get the AUC
evaluator.evaluate(predictions)
```

```
     70         else:
     71             raise ValueError("Params must be a param map but got
 %s." % type(params))

/usr/lib/spark/python/pyspark/ml/evaluation.py in _evaluate(self, datase
t)
     97         """
     98         self._transfer_params_to_java()
---> 99         return self._java_obj.evaluate(dataset._jdf)
    100
    101     def isLargerBetter(self):

/usr/lib/spark/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py in __c
all__(self, *args)
   1131         answer = self.gateway_client.send_command(command)
   1132         return_value = get_return_value(
-> 1133             answer, self.gateway_client, self.target_id, self.nam
e)
   1134
   1135         for temp_arg in temp_args:
```