|  |  |
| --- | --- |
|  | **Machine Learning II** |

|  |  |
| --- | --- |
| **Homework #2** | **Due: turned in by Monday 02/11/2019 before class** |

\_\_\_Philipp Scherbel\_\_\_

(put your name above)

Total grade: \_\_\_\_\_\_\_ out of \_\_\_100\_\_\_ points

***There are 2 parts in this homework assignment with 3 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 (30 points; 15 points each)**

1. **What are the main differences between pyspark.ml and pyspark.mllib? Name at least three differences.**

**Data format:**

* pyspark.mllib: source data has to be in RDD format
* pyspark.ml source data has to be in dataframe format

**Modules:**

* pyspark.mllib: includes packages for dist, random and tree
* pyspark.ml includes moduls for tuning, image and param

**Usability:**

* pyspark.mllib: difficult to develop a machine learning pipeline
* pyspark.ml better for machine learning pipelines

1. **Compare Scikit-learn and Spark MLlib and discuss their strengths and use cases.**

* Scikit-learn has great performance, if enough RAM is available and plenty of ML tools
* Sciki-learn is very good for data visualization and on smaller datasets (matplotlib!!!)
* If datasets gets bigger, spark should be used due to better performance on large scale to take advantage of parallel computing
* Sci Kit learn is more user friendly

# Bibliography

Classslides

obstkel. (2018, August 24). *obstkel*. Retrieved February 9, 2019, from What is Pyspark: https://www.obstkel.com/blog/what-is-pyspark

# **Part II. Hands on (70 points)**

**1. Analyze the Titanic Dataset using pyspark.ml (70 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.

from pyspark.sql import SQLContext

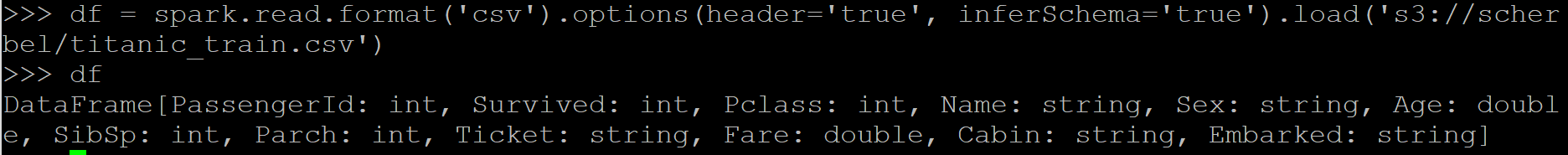
sqlContext = SQLContext(sc)

df = spark.read.format('csv').options(header='true', inferSchema='true').load('s3://scherbel/titanic\_train.csv')

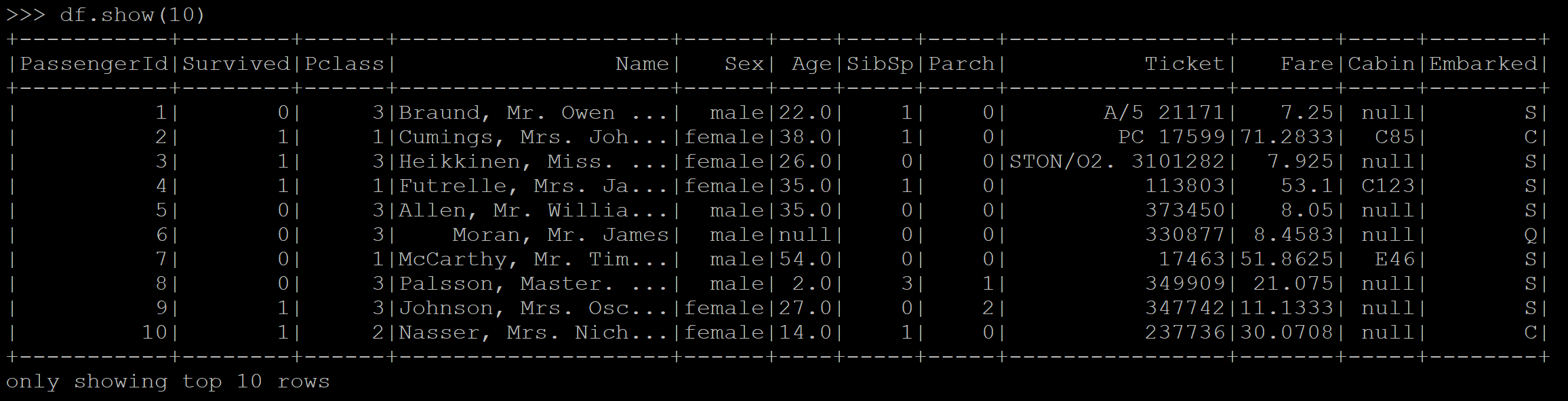
**Step 2** (10 points): The goal of this step is to familiarize yourself with the dataset. This step is useful in detecting data problems, informing the data engineering steps, and informing the feature selection processes. You should:

1. Print the dataset and verify that the schema contains all the variables.

df

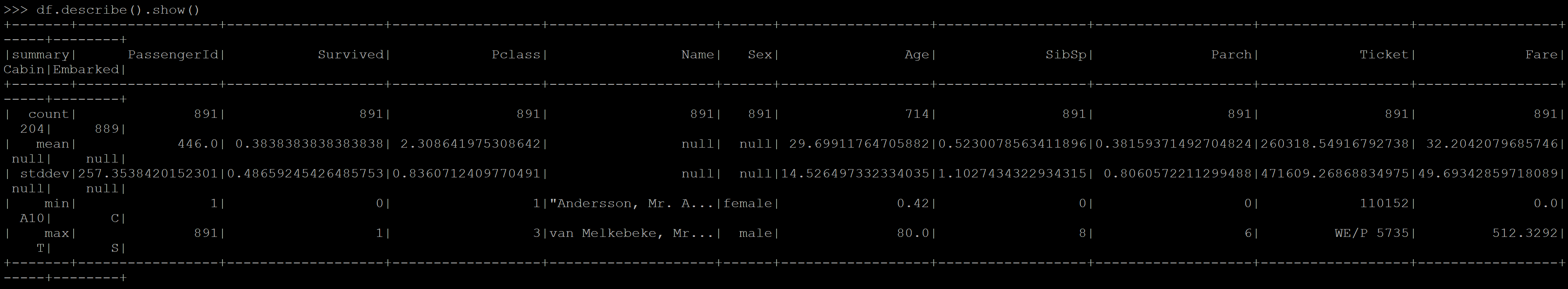


1. Print the first 10 rows from the dataset.

df.show(10) 

1. 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.

df.describe().show()



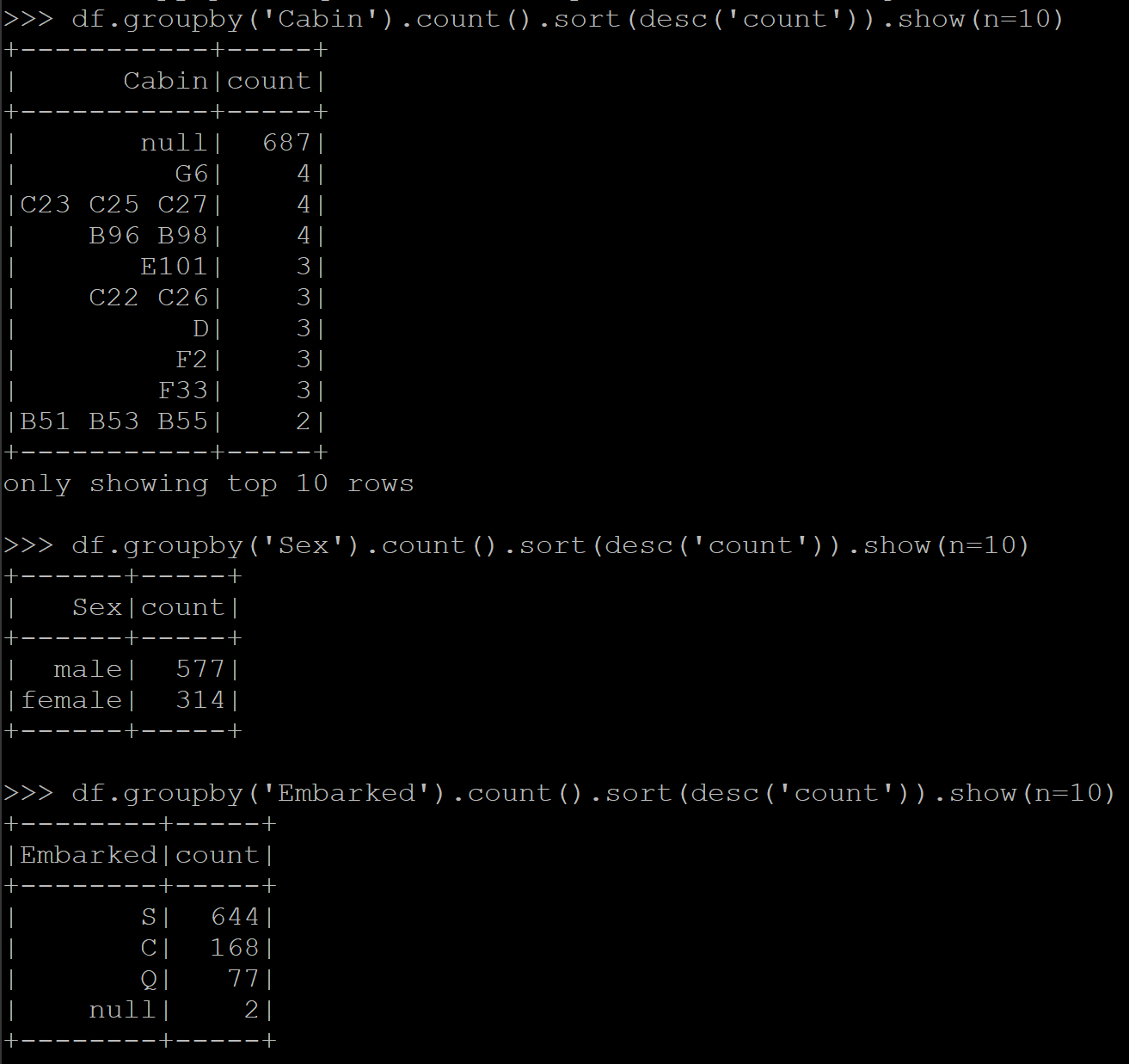
1. For each of the string columns (except name and ticket), print the count of the 10 most frequent values ordered by descending order of frequency.

from pyspark.sql.functions import desc, col, avg, when

df.groupby('Cabin').count().sort(desc('count')).show(n=10)

df.groupby('Sex').count().sort(desc('count')).show(n=10)

df.groupby('Embarked').count().sort(desc('count')).show(n=10)



1. Based on the above, which columns would you keep as features and which would you drop? Justify your answer.

df = df.drop("PassengerId","Name","Ticket","Cabin","Sex") – in case we want to delete

DataFrame[PassengerId: int, Survived: int, Pclass: int, Name: string, Sex: string, Age: double, SibSp: int, Parch: int, Ticket: string, Fare: double, Cabin: string, Embarked: string]

Keep: Survived, Pclass, Sex, Age, SibSp, Parch, Ticket, Fare, Embarked

Remaining columns are not important for the prediction like passengerid, other have a lot of null values (cabin) or don’t provide value like Name.

**Step 3** (9 points): The goal of this step is to engineer the necessary features for the machine learning model. You should:

1. 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.

dfnew = df.select("Survived", "Pclass", "Sex", "Age", "SibSp", 'Parch', 'Ticket', 'Fare', 'Embarked')

from pyspark.sql.types import DoubleType

dfnew = dfnew.withColumn("Survived", dfnew["Survived"].cast(DoubleType()))

dfnew = dfnew.withColumn("SibSp", dfnew["SibSp"].cast(DoubleType()))

dfnew = dfnew.withColumn("Parch", dfnew["Parch"].cast(DoubleType()))

dfnew = dfnew.withColumn("Pclass", dfnew["Pclass"].cast(DoubleType()))

dfnew = dfnew.withColumn("Fare", dfnew["Fare"].cast(DoubleType()))

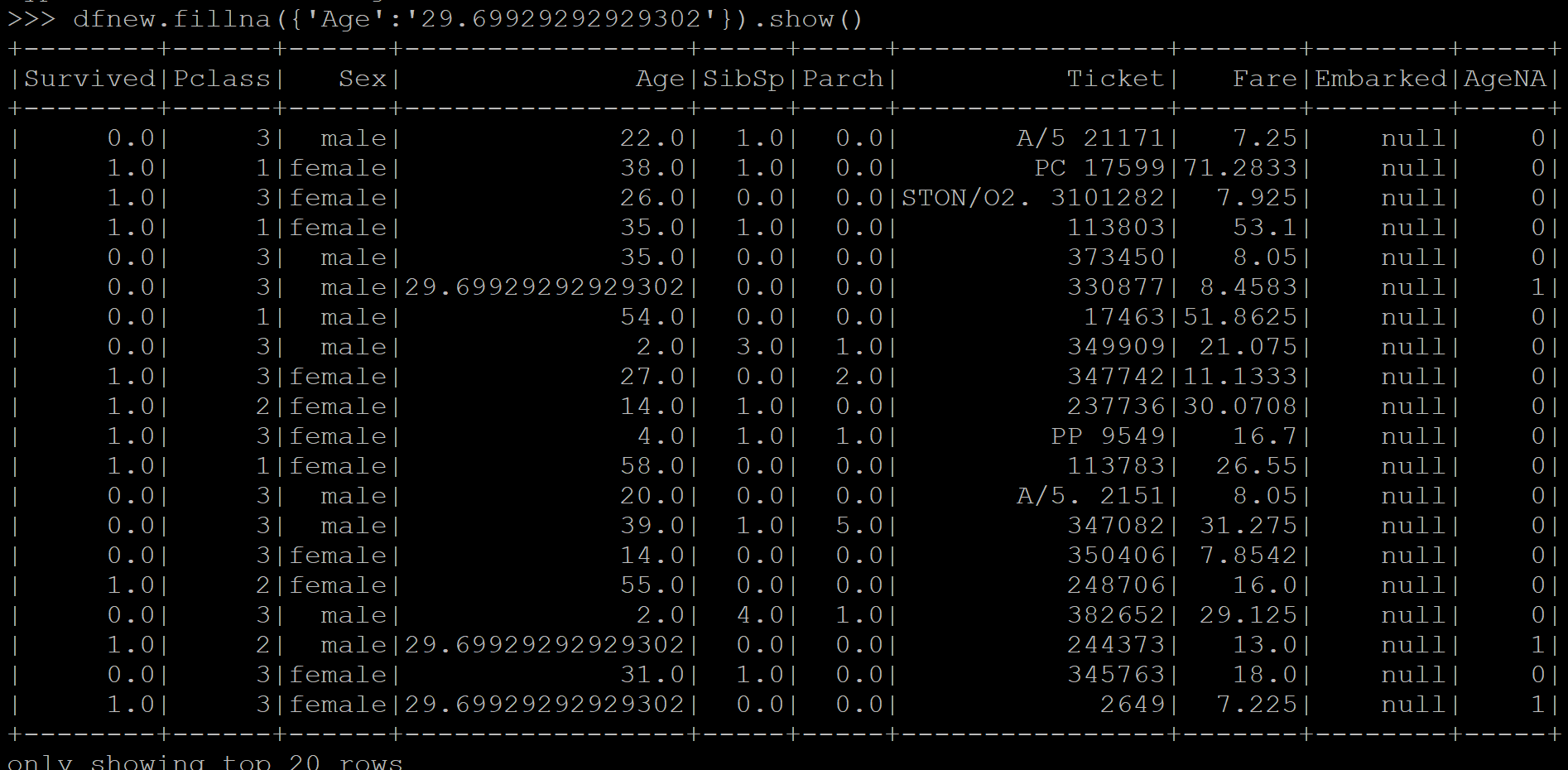
1. Replace the missing values in the Age column with the mean value. Create also a new variable (e.g., ‘AgeNA’) indicating whether the value of age was missing or not.

meanage = dfnew.select(avg("Age")).show()

dfnew = dfnew.withColumn("AgeNA",when(col("Age").isNotNull(),0).otherwise(1))

dfnew.fillna({'Age':'29.69929292929302'}).show()

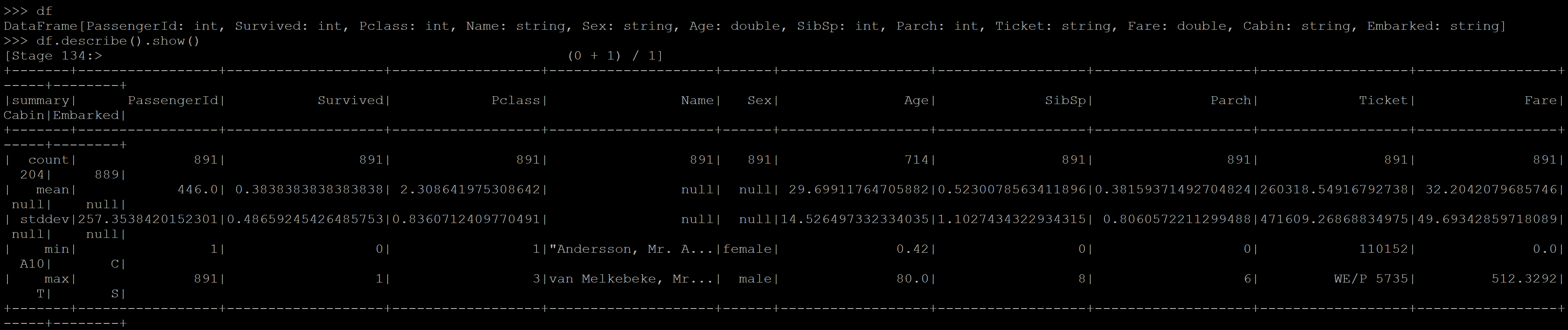
# I am sure there is a more elegant way



1. Print the revised dataframe and recalculate the summary statistics.

df

df.describe().show()



**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:

1. Import all necessary pyspark functions.

from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler

from pyspark.ml.linalg import Vectors

from pyspark.ml.classification import LogisticRegression

from pyspark.ml import Pipeline

from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit

from pyspark.ml.evaluation import BinaryClassificationEvaluator

#from pyspark.mllib.evaluation import BinaryClassificationMetrics

1. Create indexers and encoders for categorical string variables. Call them [field]\_indexer and [field]\_encoder, respectively. For instance, gender\_indexer and gender\_encoder.

categoricalColumns = ["Sex", "Ticket", "Embarked","Sex","AgeNA"]

stages = [] # stages in our Pipeline

for categoricalCol in categoricalColumns:

stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + "Index")

encoder = OneHotEncoderEstimator(inputCol=categoricalCol + "Index", outputCol=categoricalCol + "classVec")

encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])

stages += [stringIndexer, encoder]

label\_stringIdx = StringIndexer(inputCol="Survived", outputCol="label")

stages += [label\_stringIdx]

**Step 5** (5 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.

numericCols = ["Age", "Survived", "Pclass", "SibSp", "Parch", "Fare"]

assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols

assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")

stages += [assembler]

**Step 6** (5 points): The goal of this step is to create the logistic regression model to be used in the pipeline.

from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(labelCol="label", featuresCol=("features"), maxIter = 10)

**Step 7** (5 points): The goal of this step is to assemble the pipeline.

pipe = Pipeline().setStages(stages)

pipeModel = pipe.fit(dfnew)

transform = pipeModel.transform(dfnew)

#from pyspark.ml import Pipeline

#steps = [Survived, Pclass, Age, SibSp, Parch, Ticket, Fare, SexIndex: double, SexclassVec: vector, #TicketIndex, TicketclassVec:, EmbarkedIndex: double, EmbarkedclassVec:, label, features]

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

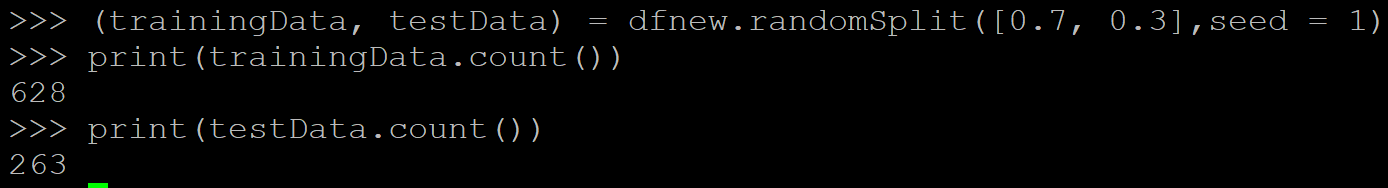
1. Use a 70-30 random split for the training and test sets, respectively.

(trainingData, testData) = data.randomSplit([0.7, 0.3],seed = 1)

1. Verify the size of each dataset after the split.

print(trainingData.count())

print(testData.count())



**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:

1. Fit the model using the predefined pipeline on the training set.

Model = lr.fit(trainingData)

1. Use the fitted model for prediction on the test set.

lr\_prediction = Model.transform(testData)

1. Report the logistic regression coefficients.

coeffs = Model.coefficients

coeffs = [(float(w),) for w in coeffs]

weightsDF = sqlContext.createDataFrame(coeffs, ["Feature Weight"])

display(weightsDF)

1. Interpret the obtained coefficients.

Coefficients are less important if their (absolute) value is little compared to the remaining coefficients. The higher the coeffecients value, the more important their impact on the target variable. Pclass has a high impact on the possible survival of a passenger for example, as these passengers were favoured during the rescue.

**Step 10** (8 points): The goal of this step is to evaluate the model performance. You should:

1. Print the first 5 rows of the results.

lr\_predictions.show(5)

# I get an error during model building process but code should work that way ?!

1. Report the AUC for this model.

from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Evaluate model

evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")

evaluator.evaluate(lr\_predictions)

Source: https://docs.databricks.com/spark/latest/mllib/binary-classification-mllib-pipelines.html