

Hands-On Exercise 6: Outlier Detection with Titanic dataset

Instructor: Dr. Peng Kang

This hands-on exercise is based on an ongoing machine learning competition, “Titanic: Machine Learning from Disaster” from Kaggle. See the below links for the Kaggle competition sites:

- Competition: <https://www.kaggle.com/c/titanic>
- Competition video: <https://www.youtube.com/watch?v=8yZMXCaFshs>

In this Hands-on exercise, you will learn.

- How to use quantiles to detect the outliers in data (the Titanic Training dataset)

Related DM Book Chapters/Sections:

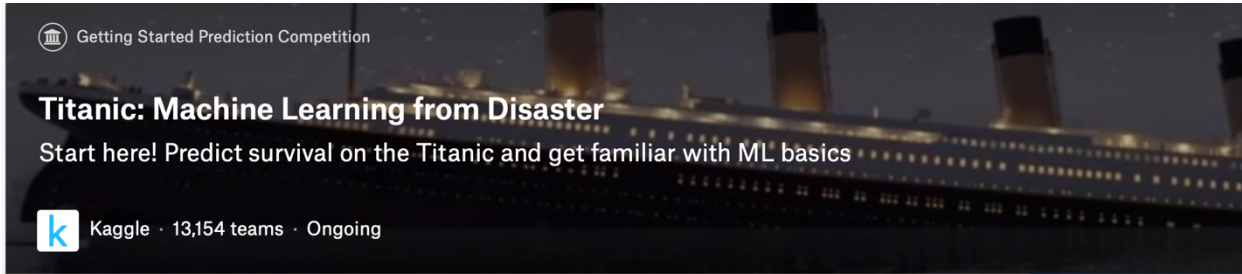
- Section 2.2.2 Measuring the Dispersion of Data: Range, Quartiles, Variance, Standard Deviation, and Interquartile Range

Related Hands-on Exercises:

- Exercise 1-2 Apache Spark and Basic Statistics

Kaggle Prediction Competition

Machine Learning from Disaster



Getting Started Prediction Competition

Titanic: Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

Kaggle · 13,154 teams · Ongoing

[Overview](#) [Data](#) [Notebooks](#) [Discussion](#) [Leaderboard](#) [Rules](#)

Join Competition

Overview

Description

Evaluation

Tutorials


Frequently Asked Questions

👋🚢 **Ahoy, welcome to Kaggle! You're in the right place.**

This is the legendary Titanic ML competition – the best, first challenge for you to dive into ML competitions and familiarize yourself with how the Kaggle platform works.

The competition is simple: use machine learning to create a model that predicts which passengers survived the Titanic shipwreck.

Read on or watch the video below to explore more details. Once you're ready to start competing, click on the ["Join Competition button"](#) to create an account and gain access to the [competition data](#). Then check out [Alexis Cook's Titanic Tutorial](#) that walks you through step by step how to make your first submission!



kaggle

How to Get Started with Kaggle's Titanic Machine Learning Competition

1. Titanic Dataset

1.1. Loading data (into PySpark)

Please make sure that the data is uploaded to your desired notebook via a file upload, or by connecting your notebook to your google drive, as we did in previous examples.

Let's start to load the data. Start your spark session (as we did in previous examples).

```
#Install the required libraries
!pip install pyspark
!pip install spark
from pyspark.sql import SparkSession
# Initialize a Spark session
spark = SparkSession.builder.appName("CSC533").getOrCreate()
```

Load the data using spark.read command from HDFS. Note: Keep in mind that Spark stores all the data or DataFrame in the memory. If you restart your session, you should load data or create DataFrames again. Let's first see what's there in the dataset.

```
PassengerId,Survived,Pclass,Name,Gender,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked
1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S
2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.2833,C85,C
3,1,3,"Heikkinen, Miss. Laina",female,26,0,0,STON/O2. 3101282,7.925,,S
4,1,1,"Futrelle, Mrs. Jacques Heath (Lily May Peel)",female,35,1,0,113803,53.1,C123,S
5,0,3,"Allen, Mr. William Henry",male,35,0,0,373450,8.05,,S
6,0,3,"Moran, Mr. James",male,,0,0,330877,8.4583,,Q
7,0,1,"McCarthy, Mr. Timothy J",male,54,0,0,17463,51.8625,E46,S
```

The dataset contains 12 columns described as follows:

Variable	Definition	Key
PassengerId	Passenger's ID	
Survival	Survival	0 = No, 1 = Yes
Pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
Name	Passenger's Name	
Gender	Passenger's Gender	Male, Female
Age	Age in years	
SibSp	Number of siblings / spouses aboard the Titanic	
Parch	Number of parents / children aboard the Titanic	
Ticket	Ticket number	
Fare	Passenger fare	
Cabin	Cabin number	
Embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

The dataset has a header, so we will use the header option. We will also use 'inferSchema' to infer the input schema automatically from the dataset. It requires one extra pass over the dataset. For a detailed description, see the below link:

<https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrameReader.csv.html?highlight=inferSchema> (search inferSchema). One example is that the 'Age' value will be loaded as a floating number instead of a string in this dataset by adding the 'inferSchema' option.

```
raw_df = spark.read.option("header", "true").option("inferSchema",
"true").csv([PATH_TO_FILE])
```

```
raw_df.show(2)
```

PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen ...	male	22.0	1	0	A/5 21171	7.25	null	S
2	1	1	Cumings, Mrs. Joh...	female	38.0	1	0	PC 17599	71.2833	C85	C

only showing top 2 rows

Original data are shown below: Values of Age for the first two rows were 22 and 38 in the original CSV file.

```
PassengerId,Survived,Pclass,Name,Gender,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked
1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S
2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.2833,C85,C
```

There is another way to give some options. You will see the same results. See below.

```
raw_df = spark.read.csv([PATH_TO_FILE], header=True, inferSchema=True)
```

```
raw_df.show(2)
```

PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen ...	male	22.0	1	0	A/5 21171	7.25	null	S
2	1	1	Cumings, Mrs. Joh...	female	38.0	1	0	PC 17599	71.2833	C85	C

only showing top 2 rows

Also, check the schema (data types)

```
# Also, check the schema (data types)

raw_df.printSchema()
root
|-- PassengerId: integer (nullable = true)
|-- Survived: integer (nullable = true)
|-- Pclass: integer (nullable = true)
|-- Name: string (nullable = true)
|-- Gender: string (nullable = true)
|-- Age: double (nullable = true)
|-- SibSp: integer (nullable = true)
|-- Parch: integer (nullable = true)
|-- Ticket: string (nullable = true)
|-- Fare: double (nullable = true)
|-- Cabin: string (nullable = true)
|-- Embarked: string (nullable = true)
```

1.2. Total number of records: count()

The count () function shows the total number of records below.

```
raw_df.count()
891
```

Displaying the dataset without truncating outputs is shown below.

```
raw_df.show(2, truncate=False)
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| PassengerId | Survived | Pclass | Name | Gender | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.25 | null | S |
| 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Thayer) | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 2 rows
```

1.3. Basic statistics: describe()

Another useful function is ‘describe()’ to get basic statistics for numeric and string columns. It includes count, mean, stddev, min, and max. This function computes statistics for all numerical or string columns if no columns are given. For details, see

<https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/> (search describe)

As you can see, it shows basic statistics for all the columns/features.

summary	PassengerId	Survived	Pclass	Name	Gender	Age
count	891	891	891	891	891	714
mean	446.0	0.3838383838383838	2.308641975308642	null	null	29.69911764705882
stddev	257.3538420152301	0.48659245426485753	0.8360712409770491	null	null	14.526497332334035
min	1	0	1	"Andersson, Mr. A..."	female	0.42
max	99	1	3	van Melkebeke, Mr...	male	9

SibSp	Parch	Ticket	Fare	Cabin	Embarked
891	891	891	891	204	889
0.5230078563411896	0.38159371492704824	260318.54916792738	32.2042079685746	null	null
1.1027434322934315	0.8060572211299488	471609.26868834975	49.69342859718089	null	null
0	0	110152	0.0	A10	C
8	6	WE/P 5735	512.3292	T	S

Here is an example of how someone may select/update his features by analyzing the above tables:

- It does not make sense to include some features such as PassengerID, Name, and Ticket → we will drop them
- The cabin has a lot of null values → we will drop it as well
- Maybe the Embarked column has nothing to do with survival → let us remove it
- We are missing 177 values from the Age column → Age is important; we [need to find a way to deal with the missing values](#)
- Gender has nominal values → [need to encode them](#)

1.4. Filtering: select ()

Let us filter out the unneeded columns first using the select () function

```
filtered_df = raw_df.select(['Survived', 'Pclass', 'Gender', 'Age', 'SibSp', 'Parch', 'Fare'])
```

```
filtered_df.show()
```

Survived	Pclass	Gender	Age	SibSp	Parch	Fare
0	3	male	22.0	1	0	7.25
1	1	female	38.0	1	0	71.2833
1	3	female	26.0	0	0	7.925
1	1	female	35.0	1	0	53.1
0	3	male	35.0	0	0	8.05
0	3	male	null	0	0	8.4583
0	1	male	54.0	0	0	51.8625
0	3	male	2.0	3	1	21.075
1	3	female	27.0	0	2	11.1333
1	2	female	14.0	1	0	30.0708
1	3	female	4.0	1	1	16.7
1	1	female	58.0	0	0	26.55
0	3	male	20.0	0	0	8.05
0	3	male	39.0	1	5	31.275
0	3	female	14.0	0	0	7.8542
1	2	female	55.0	0	0	16.0
0	3	male	2.0	4	1	29.125
1	2	male	null	0	0	13.0
0	3	female	31.0	1	0	18.0
1	3	female	null	0	0	7.225

only showing top 20 rows

Detecting Outliers

There are many different techniques to find outliers. Some of them are shown below.

This hands-on exercise is based on tutorials in References, especially “How to detect outliers with Spark” by Siavashi [1]

1.5. Four outlier detection techniques

1.5.1. Numeric Outlier

This is the simplest, nonparametric outlier detection method in a one-dimensional feature space. Here outliers are calculated using the *IQR* (InterQuartile Range).

1.5.2. Z-Score

Z-score is a parametric outlier detection method in a one or low-dimensional feature space. This technique assumes a Gaussian distribution of the data. The outliers are the data points that are in the tails of the distribution and, therefore, far from the mean.

1.5.3. Density Based Spatial Clustering of Applications with Noise (DBSCAN)

This technique is based on the DBSCAN clustering method. DBSCAN is a nonparametric, density-based outlier detection method in a one or multi-dimensional feature space.

In the DBSCAN clustering technique, all data points are defined either as *Core Points*, *Border Points*, or *Noise Points*.

- **Core Points** are data points that have at least *MinPts* neighboring data points within a distance ϵ .
- **Border Points** are neighbors of a *Core Point* within the distance ϵ but with less than *MinPts* neighbors within the distance ϵ .
- All other data points are **Noise Points**, also identified as outliers.

Outlier detection thus depends on the required number of neighbors *MinPts*, the distance ϵ , and the selected distance measure, like Euclidean or Manhattan.

1.5.4. Isolation Forest

This is a nonparametric method for large datasets in a one or multi-dimensional feature space. An essential concept in this method is the isolation number.

The isolation number is the number of splits needed to isolate a data point. This number of splits is ascertained by following these steps:

- A point “a” to isolate is selected randomly.
- A random data point “b” is set between the minimum and maximum values and differs from “a”.
- If the value of “b” is lower than the value of “a”, the value of “b” becomes the new lower limit.
- If the value of “b” is greater than the value of “a”, the value of “b” becomes the new upper limit.
- This procedure is repeated as long as there are data points other than “a” between the upper and the lower limit.

It requires fewer splits to isolate an outlier than it does to isolate a nonoutlier, i.e., an outlier has a lower isolation number in comparison to a nonoutlier point. A data point is therefore defined as an outlier if its isolation number is lower than the threshold.

The threshold is defined based on the estimated percentage of outliers in the data, which is the starting point of this outlier detection algorithm.

A summary of the four outlier detection techniques' characteristics is shown below.

Outlier Detection Technique	Normality Assumption	Dimension	Big Data	Required Pre-Processing	Parameters
Numeric Outlier	✗	1	✗		IQR multiplier k
Z-Score	✓	1 → low	✗	Normalize	Threshold z
DBSCAN	✗	1 → multi	✗	Normalize, calculate distance matrix	Required number of neighbors $MinPts$, Distance ϵ , distance measure
Isolation Forest	✗	1 → multi	✓		Estimated percentage of outliers

Table source: [6]

1.6. Pros and Cons

1.6.1. Z-Score

1.6.1.1. Pros:

- It is a very effective method if you can describe the values in the feature space with a Gaussian distribution. (Parametric)

1.6.1.2. Cons:

- It is only convenient to use in low-dimensional feature space in a small to medium-sized dataset.
- It is not recommended when distributions cannot be assumed to be parametric.

1.6.2. DBSCAN

1.6.2.1. Pros:

- It is a super effective method when the distribution of values in the feature space cannot be assumed.
- Works well if the feature space for searching outliers is multidimensional (i.e., 3 or more dimensions)
- Visualizing the results is easy, and the method itself is very intuitive.

1.6.2.2. Cons:

- The values in the feature space need to be scaled accordingly.
- Selecting the optimal parameters ϵ , $MinPts$ and metric can be difficult since it is very sensitive to any of the three params.
- It is an unsupervised model and needs to be re-calibrated each time a new batch of data is analyzed.
- It can predict once calibrated but is strongly not recommended.

1.6.3. Isolation Forest

1.6.3.1. Pros:

- There is no need to scale the values in the feature space.

- It is an effective method when value distributions cannot be assumed.
- It has few parameters; this makes this method relatively robust and easy to optimize.
- [1.6.3.2.](#) *Cons:*
- Visualizing results is complicated.
- Training time can be very long and computationally expensive if not correctly optimized.

Please look at this tutorial if you want more details [7]

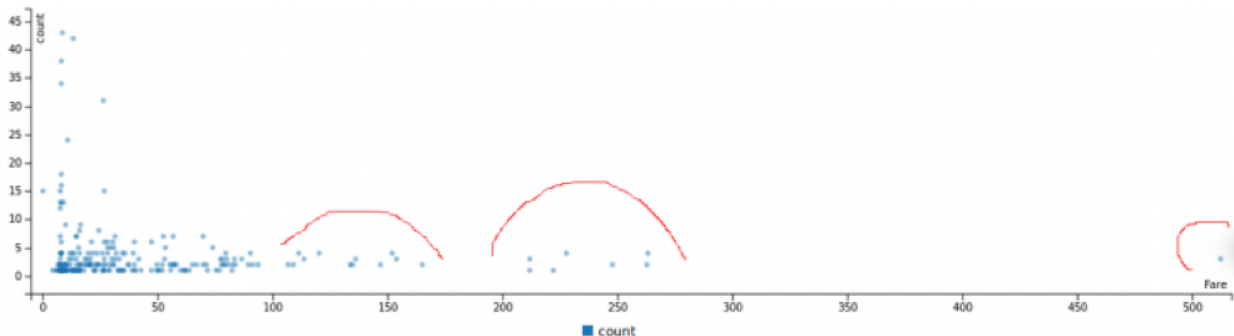
1.7. Possible Outliers in Fare Column

Let's look at the basic statistics of the 'Fare' column.

```
filtered_df.select('Fare').describe().show()
```

summary	Fare
count	891
mean	32.2042079685746
stddev	49.69342859718089
min	0.0
max	512.3292

Despite the huge variation indicated by the range from 0 to 512.3292, its meaning is just 32.20. If you sort them on an axis, most will gather on the left side. It may be showing up like the below:



As you can see, more than 90% of the records are less than 100, and the outliers are exposed on the right side. Let's bucketize the records to measure the distribution.

1.8. Bucketizer

Bucketizer transforms a column of continuous features into a column of feature buckets, where the buckets are specified by users. It takes a parameter:

- **Splits:** Parameter for mapping continuous features into buckets. With $n+1$ splits, there are n buckets. A bucket defined by splits x,y holds values in the range $[x,y)$ except for the last bucket, which also includes y . Splits should be strictly increasing. Values at $-\infty, \infty$ must be explicitly provided to cover all Double values; Otherwise, values outside the splits specified will be treated as errors.

See for detail: <https://spark.apache.org/docs/latest/ml-features - bucketizer>

Let's divide the records into 5 bins in the range of 0 to 100, 100 to 200, 200 to 300, 300 to 400, and bigger than 400:

```
# import Bucketize
from pyspark.ml.feature import Bucketizer

# Define Splits
splits = [0.0, 100.0, 200.0, 300.0, 400.0, float("inf")]

# Define bucketizer with splits, input and output columns
bucketizer = Bucketizer(splits=splits, inputCol="Fare", outputCol="bucketedFare")

# Transform the data to get a bucketed DataFrame
bucketed_df = bucketizer.transform(filtered_df)
```

Let's check the bucketed DataFrame by showing only the Fare and 'bucketedFare' columns. The first 20 rows all belong to bucket 0.

```
bucketed_df.select('Fare', 'bucketedFare').show()
+-----+-----+
|   Fare|bucketedFare|
+-----+-----+
```

```

| 7.25| 0.0|
|71.2833| 0.0|
| 7.925| 0.0|
| 53.1| 0.0|
| 8.05| 0.0|
| 8.4583| 0.0|
|51.8625| 0.0|
| 21.075| 0.0|
|11.1333| 0.0|
|30.0708| 0.0|
| 16.7| 0.0|
| 26.55| 0.0|
| 8.05| 0.0|
| 31.275| 0.0|
| 7.8542| 0.0|
| 16.0| 0.0|
| 29.125| 0.0|
| 13.0| 0.0|
| 18.0| 0.0|
| 7.225| 0.0|
+-----+
only showing top 20 rows

```

Now let's calculate the magnitude of each bucket and observe the distribution:

```

bucketed_df.groupBy('bucketedFare').count().orderBy('bucketedFare').show()
+-----+-----+
|bucketedFare|count|
+-----+-----+
|          0.0| 838|
|          1.0|  33|
|          2.0|  17|
|          4.0|   3|
+-----+-----+

```

Around 90% of the records lie in bucket 0.

Now we've found out that outliers infect our dataset. But what data points must be considered outliers?

1.9. Simple Outlier detection using Box-and-Whisker Plot (or Box Plot)

There are many methods to identify outliers in statistics. We will discuss one of the easiest methods, which uses quantiles.

The quartiles indicate a distribution's center, spread, and shape. The **first quartile**, denoted by Q_1 , is the 25th percentile. It cuts off the lowest 25% of the data. The **third quartile**, denoted by Q_3 , is the 75th percentile—it cuts off the lowest 75% (or highest 25%) of the data. The second quartile is the 50th percentile. As the median, it gives the center of the data distribution.

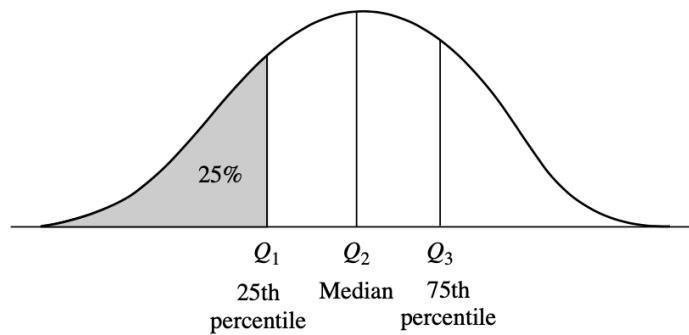


Figure 2.2 A plot of the data distribution for some attribute X. The quantiles plotted are quartiles. The three quartiles divide the distribution into four equal-size consecutive subsets. The second quartile corresponds to the median.

Source: DM book: Section 2.2.2 Measuring the Dispersion of Data: Range, Quartiles, Variance, Standard Deviation, and Interquartile Range

The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the **interquartile range (IQR)** and is defined as

$$IQR = Q3 - Q1$$

Using IQR, we can identify the outliers. This method is known as **Box and Whisker method**.

In this method, any value smaller than $Q1 - 1.5 * IQR$ or any value greater than $Q3 + 1.5 * IQR$ will be categorized as the **outlier**.

1.10. Calculate Quantiles and IQR in PySpark

Once we understand the method, we can implement it in spark. The following are the steps for implementing the same.

We are using the *approxQuantile* method to compute the quantiles needed with the following parameters:

- col: String: the names of the numerical columns
- [Double]: a list of quantile probabilities. Each number must belong to [0, 1]. For example, 0 is the minimum, 0.25 is the first quartile, 0.5 is the median, 0.75 is the third quartile, and 1 is the maximum
- relativeError: The relative target precision to achieve (greater than or equal to 0). The exact quantiles are computed if set to zero, which could be very expensive. Note that values greater than 1 are accepted but give the same result as 1.

```

# Calculate quantiles
quantiles = filtered_df.approxQuantile("Fare", [0.25, 0.75], 0.0)

# Show first quantile (25%)
print(quantiles[0])
# 7.8958

# Show third quantile (75%)
print(quantiles[1])
# 31.0

```

Once we have quantiles, we can calculate IQR.

```

# Quantiles
Q1 = quantiles[0]
Q3 = quantiles[1]

# Calculate IQR
IQR = Q3-Q1

```

We then calculate the ranges. We are using 1.5 as an interquartile multiplier value; the range limits are a box plot's typical upper and lower whiskers.

```

# Calculate Lower Range using Q1 and IQR
lowerRange = Q1 - 1.5 * IQR
print(lowerRange)
# -26.7605

# Calculate Upper Range using Q3 and IQR
upperRange = Q3 + 1.5 * IQR
print(upperRange)
# 65.6563

```

Then we filter the data using DataFrame filters.

This data's 'Fare' column has no outliers in the lower range.

```

# Calculate lower Outliers
outliers_low = filtered_df.filter(filtered_df.Fare < lowerRange)
print(outliers_low.count())
# 0
outliers_low.show()

```

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|PassengerId|Survived|Pclass|Name|Gender|Age|SibSp|Parch|Ticket|Fare|Cabin|Embarked|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

```
# Calculate Upper Outliers
outliers_upper = filtered_df.filter(filtered_df.Fare > upperRange)
print(outliers_upper.count())
# 116

outliers_upper.show(15)
```

PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	1	Cumings, Mrs. Joh...	female	38.0	1	0	PC 17599	71.2833	C85	C
28	0	1	Fortune, Mr. Char...	male	19.0	3	2	19950	263.0	C23 C25 C27	S
32	1	1	Spencer, Mrs. Wil...	female	null	1	0	PC 17569	146.5208	B78	C
35	0	1	Meyer, Mr. Edgar ...	male	28.0	1	0	PC 17604	82.1708	null	C
53	1	1	Harper, Mrs. Henr...	female	49.0	1	0	PC 17572	76.7292	D33	C
62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0	B28	null
63	0	1	Harris, Mr. Henry...	male	45.0	1	0	36973	83.475	C83	S
73	0	2	Hood, Mr. Ambrose Jr	male	21.0	0	0	S.O.C. 14879	73.5	null	S
89	1	1	Fortune, Miss. Ma...	female	23.0	3	2	19950	263.0	C23 C25 C27	S
103	0	1	White, Mr. Richar...	male	21.0	0	1	35281	77.2875	D26	S
119	0	1	Baxter, Mr. Quigg...	male	24.0	0	1	PC 17558	247.5208	B58 B60	C
121	0	2	Hickman, Mr. Stan...	male	21.0	2	0	S.O.C. 14879	73.5	null	S
125	0	1	White, Mr. Perciv...	male	54.0	0	1	35281	77.2875	D26	S
140	0	1	Giglio, Mr. Victor	male	24.0	0	0	PC 17593	79.2	B86	C
152	1	1	Pears, Mrs. Thoma...	female	22.0	1	0	113776	66.6	C2	S

only showing top 15 rows

Submit

Finish the assignments shown below. Submit a python notebook with answers/ explanations and your queries/codes/scripts and outputs.

Most common causes of outliers on a data set:

- Data entry errors (human errors)
- Measurement errors (instrument errors)
- Experimental errors (data extraction or experiment planning/executing errors)
- Intentional (dummy outliers made to test detection methods)
- Data processing errors (data manipulation or data set unintended mutations)
- Sampling errors (extracting or mixing data from wrong or various sources)
- Natural (not an error, novelties in data)

To improve the performances of classification or clustering models, you should remove outliers from the dataset caused by human data entry errors or common causes mentioned above.

Assignment #1 - 4 (0.6pts per each column you choose, 2.4pts in total)

Remove outliers from the 4 columns you choose (You may include 'Fare') in the dataset using the Box and Whisker method. Choose *at least three* different columns that have (possible) outliers. For each column you choose,

- Analyze the column to see whether it has outliers or not. Need outlier removal or not? Why?
- Show how did you find outliers using the IQR method in this exercise.

Assignment #5 (3.6pts)

After you removed all outliers from the columns you chose, redo Hands-on Ex 4-2 and evaluate the model you trained with the no-outliers dataset. Explain the differences you found, e.g., AUROC and AUPR, between the model trained with outliers and the model trained with a no-outliers. Better or not?

Note: You may not be able to see a dramatic increase or changes.

Note: It's up to you to remove outliers first and then process missing values or vice versa.

References

1. How to detect outliers with Spark
<https://jrviton.wordpress.com/2017/09/20/how-to-detect-outliers-with-spark/>
2. Statistical Data Exploration using Spark 2.0 - Part 3: Outlier Detection using Quantiles
<http://blog.madhukaraphatak.com/statistical-data-exploration-spark-part-3/>
3. Anomaly Detection Using PySpark, Hive, and Hue on Amazon EMR
<https://aws.amazon.com/blogs/big-data/anomaly-detection-using-pyspark-hive-and-hue-on-amazon-emr/>
4. Introduction to Anomaly Detection
<https://blogs.oracle.com/datascience/introduction-to-anomaly-detection>
5. An Introduction to Outlier Detection Techniques
<https://www.digitalvidya.com/blog/outlier-detection/>
6. Four Techniques for Outlier Detection
<https://www.knime.com/blog/four-techniques-for-outlier-detection>
7. My secret recipe for Kaggle competition for top 8% on leaderboard
<https://confusedcoders.com/data-science/machine-learning/my-secret-recipe-for-kaggle-competition-for-top-8-on-leaderboard>
8. Interquartile Ranges & Outliers
<https://www.purplemath.com/modules/boxwhisk3.htm>

Copy the links if the above hyperlinks do not work