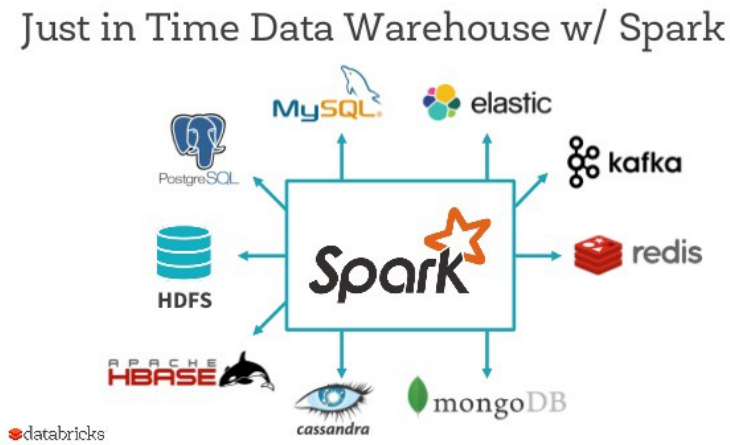
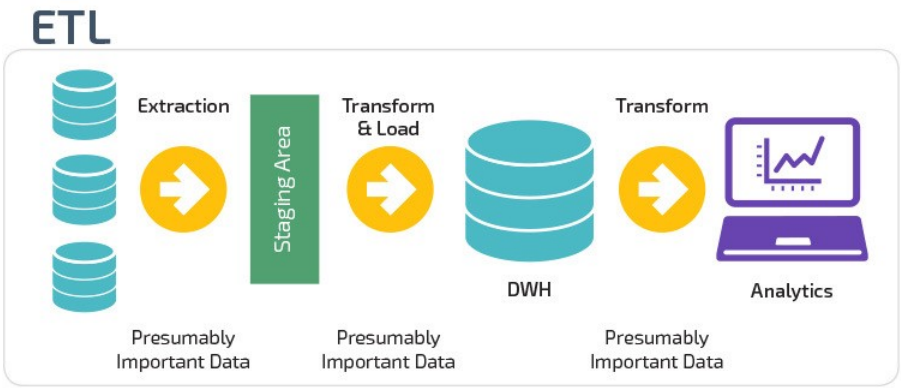
Exploratory Data Analysis using Pyspark Dataframe in Python

Ayesha Shafique Follow
Apr 4 · 14 min read

> *In this post, we will do the exploratory data analysis using PySpark dataframe in python unlike the traditional machine learning pipeline, in which we practice pandas dataframe* ***(no doubt pandas is awesome)****. But let's shed a light on PySpark, which drives our data (big data) even crazier.*



This article will give you the comprehensive overview of PySpark dataframe like as a data scientist or machine learning engineer, how you can employ PySpark dataframe in your machine learning pipeline from (Extract, Transform and Load) ETL process to Data Pre-processing step.



Note: For this tutorial, I used the IBM Watson free account to utilize Spark service with python notebook 3.5 version. The code remains the same.

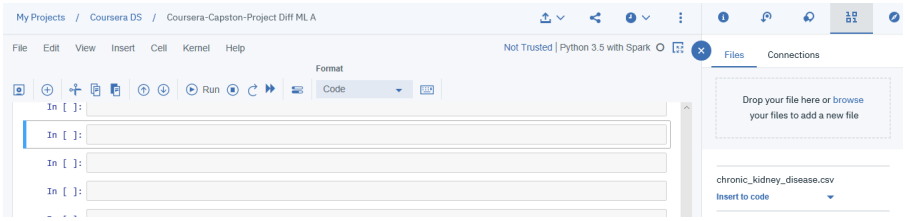
Before getting our hands dirty with pyspark, let's define our datasource first, that will eventually be the first step of our machine learning pipeline.

PySpark Data Source: Chronic Kidney Disease Dataset

The dataset on which we are ready to play is of chronic kidney disease. After exploratory analysis, we can also use this dataset to predict the chronic kidney disease (in part 2) to build a complete end to end machine learning pipeline.

Note: You can download this data from here.

For people who are using IBM Watson free account, first, go the project, then create a notebook with environment Python 3.5 with Spark. Now upload this dataset in CSV format to the IBM Watson.



Upload CSV file to IBM Watson

Now click on the uploaded file exploration menu and select the option “Insert SparkSession DataFrame”. All the code to load “chronic_kidney_disease.csv” to this notebook is automatically drafted to the notebook cell.

```
import ibmos2spark
# @hidden_cell
credentials = {
    'endpoint': 'https://s3-api.us-geo.objectstorage.service.networklayer.com',
    'service_id': 'iam-ServiceId-05ab914c-2647-4655-a774-046065e40648',
    'iam_service_endpoint': 'https://iam.bluemix.net/oidc/token',
    'api_key': 'PGXBGSbZAGDQ0FTVUVX99kEaENnOj_hfxNqSpPo6d1MJ'
}

configuration_name = 'os_d5e255d53c97413bad14470512fa886c_configs'
cos = ibmos2spark.CloudObjectStorage(sc, credentials, configuration_name, 'bluemix_cos')

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
df = spark.read\
  .format('org.apache.spark.sql.execution.datasources.csv.CSVFileFormat')\
  .option('header', 'true')\
  .load(cos.url('chronic_kidney_disease_dataset.csv', 'courserads-donotdelete-pr-y8rkc0wxhk
df.take(5)
```

lib2spark.py hosted with ♡ by GitHub view raw

Now this dataset is loaded as a spark dataframe using spark.read method. All the steps from onwards will be equivalent no matter which platform you are using (cloud or local) for spark service.

If we talk about machine learning pipeline, you have completed your ETL step. Roughly but it’s your nice attempt :-).

> *Let’s deep dive into this exploratory pyspark blog*

Tables Of Content

1. schema of pyspark dataframe

2.Show your PySpark Dataframe

3. Count function of PySpark Dataframe

4. Statistical Properties of PySpark Dataframe

5. Remove Column from the PySpark Dataframe

6. Find unique values of a categorical column

7. Filter PySpark Dataframe based on the Condition

8.Count the missing values in a column of PySpark Dataframe

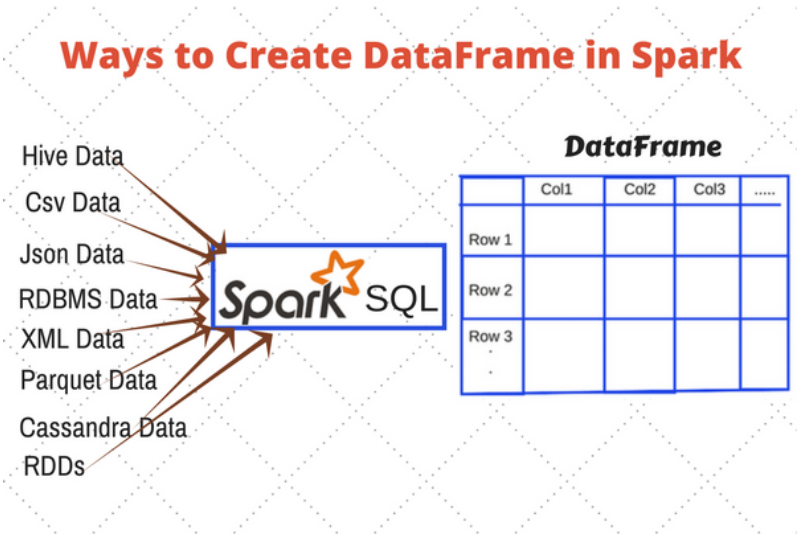
9. Replace values in PySpark Dataframe

10. How to fill missing values using mean of the column of PySpark Dataframe

11. How to fill missing values using mode of the column of PySpark Dataframe

1. Schema of PySpark Dataframe

In an exploratory analysis, the first step is to look into your schema. A schema is a big picture of your dataset. What your dataset actually narrates. Columns name of dataframe, with their datatype.



df.printSchema()

```
Output:

root

| — age: string (nullable = true)

| — bp: string (nullable = true)
```

```
| — sg: string (nullable = true)

| — al: string (nullable = true)

| — su: string (nullable = true)

| — rbc: string (nullable = true)

| — pc: string (nullable = true)

| — pcc: string (nullable = true)

| — ba: string (nullable = true)

| — bgr: string (nullable = true)

| — bu: string (nullable = true)

| — sc: string (nullable = true)

| — sod: string (nullable = true)

| — pot: string (nullable = true)

| — hemo: string (nullable = true)

| — pcv: string (nullable = true)

| — wbcc: string (nullable = true)

| — rbcc: string (nullable = true)

| — htn: string (nullable = true)

| — dm: string (nullable = true)

| — cad: string (nullable = true)

| — appet: string (nullable = true)

| — pe: string (nullable = true)

| — ane: string (nullable = true)

| — class: string (nullable = true)
```

2. Show your PySpark Dataframe

Just like Pandas **head**, you can use **show** and **head** functions to display the first N rows of the dataframe.

df.show(5)

```
Output:

+ — -+ — -+ — — -+ — -+ — -+ — — — — + — — — — + — — — — — — + — — —
— — + — -+ — -+ — -+ — -+ — -+ — — + — -+ — — -+ — — + — -+ — -+ —
-+ — — -+ — -+ — -+ — — -+ — — +

|age| bp| sg| al| su| rbc| pc| pcc| ba|bgr| bu|
sc|sod|pot|hemo|pcv| wbcc|rbcc|htn| dm|cad|appet| pe|ane|class|

+ — -+ — -+ — — -+ — -+ — -+ — — — — + — — — — + — — — — — — + — — —
— — + — -+ — -+ — -+ — -+ — -+ — — + — -+ — — -+ — — + — -+ — -+ —
-+ — — -+ — -+ — -+ — — -+ — — +

| 48| 80| 1.02| 1| 0| ?| normal|notpresent|notpresent|121| 36|1.2|
?| ?|15.4| 44| 7800| 5.2|yes|yes| no| good| no| no| ckd|
```

```
| 7| 50| 1.02| 4| 0| ?| normal|notpresent|notpresent| ?| 18|0.8| ?
| ?|11.3| 38| 6000| ?| no| no| no| good| no| no| ckd|

| 62| 80| 1.01| 2| 3| normal| normal|notpresent|notpresent|423|
53|1.8| ?| ?| 9.6| 31| 7500| ?| no|yes| no| poor| no|yes| ckd|

| 48| 70|1.005| 4| 0| normal|abnormal| present|notpresent|117|
56|3.8|111|2.5|11.2| 32| 6700| 3.9|yes| no| no| poor|yes|yes| ckd|

| 51| 80| 1.01| 2| 0| normal| normal|notpresent|notpresent|106|
26|1.4| ?| ?|11.6| 35| 7300| 4.6| no| no| no| good| no| no| ckd|

+ — -+ — -+ — — -+ — -+ — -+ — — — — + — — — — + — — — — — — + — — —
— — + — -+ — -+ — -+ — -+ — -+ — — + — -+ — — -+ — — + — -+ — -+ —
-+ — — -+ — -+ — -+ — — -+ — — +
```

only showing top 5 rows

df.head(5)

```
Output:

[Row(age='48', bp='80', sg='1.02', al='1', su='0', rbc='?',
pc='normal', pcc='notpresent', ba='notpresent', bgr='121',
bu='36', sc='1.2', sod='?', pot='?', hemo='15.4', pcv='44',
wbcc='7800', rbcc='5.2', htn='yes', dm='yes', cad='no',
appet='good', pe='no', ane='no', class='ckd'),

Row(age='7', bp='50', sg='1.02', al='4', su='0', rbc='?',
pc='normal', pcc='notpresent', ba='notpresent', bgr='?', bu='18',
sc='0.8', sod='?', pot='?', hemo='11.3', pcv='38', wbcc='6000',
rbcc='?', htn='no', dm='no', cad='no', appet='good', pe='no',
ane='no', class='ckd'),

Row(age='62', bp='80', sg='1.01', al='2', su='3', rbc='normal',
pc='normal', pcc='notpresent', ba='notpresent', bgr='423',
bu='53', sc='1.8', sod='?', pot='?', hemo='9.6', pcv='31',
wbcc='7500', rbcc='?', htn='no', dm='yes', cad='no', appet='poor',
pe='no', ane='yes', class='ckd'),

Row(age='48', bp='70', sg='1.005', al='4', su='0', rbc='normal',
pc='abnormal', pcc='present', ba='notpresent', bgr='117', bu='56',
sc='3.8', sod='111', pot='2.5', hemo='11.2', pcv='32',
wbcc='6700', rbcc='3.9', htn='yes', dm='no', cad='no',
appet='poor', pe='yes', ane='yes', class='ckd'),

Row(age='51', bp='80', sg='1.01', al='2', su='0', rbc='normal',
pc='normal', pcc='notpresent', ba='notpresent', bgr='106',
bu='26', sc='1.4', sod='?', pot='?', hemo='11.6', pcv='35',
wbcc='7300', rbcc='4.6', htn='no', dm='no', cad='no',
appet='good', pe='no', ane='no', class='ckd')
```

3. Count function of PySpark Dataframe

If you want to know the count of the dataframe, just write this:

df.count()

Output:400

4. Statistical Properties of PySpark Dataframe

If you are interested in more statistics of the dataframe like the total count of the rows in particular column, its mean, standard deviation, min and max of the column. It is very beneficial if someone wants to know the count of null values in the column with its statistical properties.

df.describe([“age”]).show()

```
Output:

+ – – – -+ – – – – – – – – -+

|summary| age|

+ – – – -+ – – – – – – – – -+

| count| 400|

| mean|51.48337595907928|

| stddev|17.16971408926224|

| min| 11|

| max| ?|

+ – – – -+ – – – – – – – – -+
```

For the newbie, this **show()** is just to show the output. Here you can see that max is ‘?’, it means that there is some missing value in this column that apparently seems to be this ‘?’.

> ***Tips for newbie:***
>
> *To know the statistical properties of all the columns at once, just write this.*

for col in df.columns:

df.describe([col]).show()

5. Remove Column from the PySpark Dataframe

To remove any column from the pyspark dataframe, use the drop function.

df = df.drop(“col_name”)

6. Find unique values of a categorical column

This is very helpful when you want to know the discrete classes of a categorical column. Like in this dataset, we have a column “class”, and we want to know its categories, which should be 2 (‘notckd’, ‘ckd’) in this classes because we only want to predict chronic not chronic kidney disease.

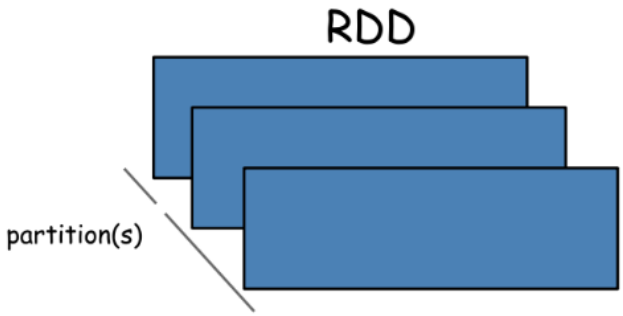
```
df.select('class').distinct().rdd.map(lambda r: r[0]).collect()

Output:

['notckd', 'ckd', 'no', 'ckd\t']
```

As you can see from the output that there are 4 unique classes. Since these classes are missing value, so we have to provide them with their right match

in next steps.

Tips for newbie:

> *If you are a newbie to this pyspark dataframe, I can explain to you in more simple terms, what is going on?* ***Select*** *is the function from which you can select one or more columns and can perform a simple and complex operation on it.* ***Distinct()*** *is the function which you can use on a pyspark column to tell the unique values of the column.* ***RDD*** *tells us that we are using pyspark dataframe as* ***Resilient Distributed Dataset (RDD)****, the basic abstraction in Spark. Represents an immutable, partitioned collection of elements that can be operated on in parallel. A map is used to transform your rdd to the form you desired using a lambda function.*



From PySpark-Pictures by Jeffrey Thompson.

pyspark package — PySpark 2.1.0 documentation

Read a directory of binary files from HDFS, a local file system (available on all nodes), or any Hadoop-supported...

spark.apache.org

For more information, you can read this above documentation.

7. Filter PySpark Dataframe based on the Condition

If you want to filter out those rows in which 'class' columns have this value "ckd\t", as shown above.

df.filter(df['class']=="ckd\t").show()

```
Output:

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|age| bp| sg| al| su| rbc| pc| pcc| ba|bgr| bu|
sc|sod|pot|hemo|pcv|wbcc|rbcc|htn| dm|cad|appet| pe|ane|class|

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

```
| 72| 80| ?| ?| ?| ?|notpresent|notpresent|137| 65|3.4|141|4.7|
9.7| 28|6900| 2.5|yes|yes| no| poor| no|yes| ckd |

| 65| 60|1.01| 2| 0|normal|abnormal| present|notpresent|192|
17|1.7|130|4.3| ?| ?|9500| ?|yes|yes| no| poor| no| no| ckd |

+ – -+ – -+ – – + – -+ – -+ – – – + – – – – + – – – – – + – – – –
– + – -+ – -+ – -+ – -+ – -+ – – + – -+ – – + – – + – -+ – -+ – -+
– – -+ – -+ – -+ – – -+
```

8. Count the missing values in a column of PySpark Dataframe

To know the missing values, we first count the null values in a dataframe.

> ***df.filter(df['col_name'].isNull()).count()***

To know the count of every column at once, write this:

#Count the value of null in every column

for col in df.columns:

print(col, "\t", "with null values: ", df.filter(df[col].isNull()).count())

```
Output:

age with null values: 0

bp with null values: 0

sg with null values: 0

al with null values: 0

su with null values: 0

rbc with null values: 0

pc with null values: 0

pcc with null values: 0

ba with null values: 0

bgr with null values: 0

bu with null values: 0

sc with null values: 0

sod with null values: 0

pot with null values: 0

hemo with null values: 0

pcv with null values: 0

wbcc with null values: 0

rbcc with null values: 0

htn with null values: 0
```

```
dm with null values: 1

cad with null values: 0

appet with null values: 0

pe with null values: 0

ane with null values: 0

class with null values: 0
```

This output tells that only 'dm' has a null value. But remember that we have some **'?'** values in dataframe that show that there are many missing values in our dataset and after identifying them, we have to cater them.

So let's filter out those rows, in which we have '?' values.

#Count the value of '?' in every column

for col in df.columns:

print(col, "\t", "with '?' values: ", df.filter(df[col]=="?").count())

```
Output:

age with '?' values: 9

bp with '?' values: 12

sg with '?' values: 47

al with '?' values: 46

su with '?' values: 49

rbc with '?' values: 152

pc with '?' values: 65

pcc with '?' values: 4

ba with '?' values: 4

bgr with '?' values: 44

bu with '?' values: 19

sc with '?' values: 17

sod with '?' values: 87

pot with '?' values: 88

hemo with '?' values: 52

pcv with '?' values: 70

wbcc with '?' values: 105

rbcc with '?' values: 130

htn with '?' values: 2
```

```
dm with '?' values: 2

cad with '?' values: 2

appet with '?' values: 1

pe with '?' values: 1

ane with '?' values: 1

class with '?' values: 0
```

Oh my goodness, you can see there are lots of missing values in this dataframe labeled as '?'.

9. Replace values in PySpark Dataframe

If you want to replace any value in pyspark dataframe, without selecting particular column, just use pyspark replace function.

#since in our dataset, column value is represented through '?' and None, so we will first replace the '? with None values.'

df = df.replace('?',None)

df=df.replace('ckd\t', 'ckd')

Now for validation count the value of '?' in every column

for col in df.columns:

print(col, "\t", "with '?' values: ", df.filter(df[col]=="?").count())

```
age with '?' values: 0

bp with '?' values: 0

sg with '?' values: 0

al with '?' values: 0

su with '?' values: 0

rbc with '?' values: 0

pc with '?' values: 0

pcc with '?' values: 0

ba with '?' values: 0

bgr with '?' values: 0

bu with '?' values: 0

sc with '?' values: 0

sod with '?' values: 0

pot with '?' values: 0
```

```
hemo with '?' values: 0

pcv with '?' values: 0

wbcc with '?' values: 0

rbcc with '?' values: 0

htn with '?' values: 0

dm with '?' values: 0

cad with '?' values: 0

appet with '?' values: 0

pe with '?' values: 0

ane with '?' values: 0

class with '?' values: 0
```

But if you want to replace the values only in particular column, you can't simply use replace function. You have to use pyspark.sql.functions.

#replace no with notckd of class column

```
from pyspark.sql.functions import when, lit

df = df.withColumn('class', when(df['class']=='no',

lit("notckd")).otherwise(df['class']))
```

In above line of code, we first select our column, and tell pyspark.sql.functions that when df['class']=='no', replace it with literal "notckd", otherwise unchanged.

```
#validate now unique values for class

df.select('class').distinct().rdd.map(lambda r: r[0]).collect()

['notckd', 'ckd']
```

Tips for newbie:

> ***People who are thinking that why I didn't opt first method for second case also, to replace no with 'notckd' because there could be a column who has 2 classes in it like yes or no, we can't replace that 'no' with our 'notckd'.***

10. How to fill missing values using mean of the column of PySpark Dataframe

Like in pandas we can just find the mean of the columns of dataframe just by df.mean() but in pyspark it is not so easy. You don't have any readymade

function available to do so. You have to define your custom function for the mean of the numeric column of the pyspark dataframe.

. . .

```python
#Find the avg of all numeric columns
from pyspark.sql.functions import avg

def mean_of_pyspark_columns(df, numeric_cols, verbose=False):
    col_with_mean=[]
    for col in numeric_cols:
        mean_value = df.select(avg(df[col]))
        avg_col = mean_value.columns[0]
        res = mean_value.rdd.map(lambda row : row[avg_col]).collect()

        if (verbose==True): print(mean_value.columns[0], "\t", res[0])
        col_with_mean.append([col, res[0]])
    return col_with_mean
```

pyspark.py hosted with ♡ by GitHub view raw

avg of all numeric columns

This is the function you can apply as it is in your code to find the mean of the numeric columns. You just have to pass the dataframe with your list of numeric columns. It will return the 2D list of the column name, with their mean, and then you can utilize this list to fill in the missing values of the numeric columns.

To use the code in an optimal fashion make an extra function that will make use of this mean_of_pyspark_columns function and will automatically fill the missing values with the mean of the numeric columns.

```python
#Fill missing values for mean
from pyspark.sql.functions import when, lit

def fill_missing_with_mean(df, numeric_cols):
    col_with_mean = mean_of_pyspark_columns(df, numeric_cols)

    for col, mean in col_with_mean:
        df = df.withColumn(col, when(df[col].isNull()==True,
        lit(mean)).otherwise(df[col]))

    return df
```

pyspark1.py hosted with ♡ by GitHub view raw

fill missing values for mean

Again, this is the piece of the code you can apply as it is in your program. It will need dataframe and the list of the numeric column as an input and will return the transformed data frame without impurities or missing values in numeric columns as they are filled by the mean of the column.

#Now this is the time to literally consume the fill_missing_with_mean

```
numeric_cols=['age', 'bp', 'sg','bgr', 'bu', 'sc', 'sod', 'pot',
'hemo', 'pcv', 'wbcc', 'rbcc']

df = fill_missing_with_mean(df, numeric_cols)
```

11. How to fill missing values using mode of the column of PySpark Dataframe

Like in pandas we can just find the mode of the columns of dataframe just by df.mode() but in pyspark it is not so easy. You have to define your custom function for the mode of the categorical column of the pyspark dataframe.

```
def mode_of_pyspark_columns(df, cat_col_list, verbose=False):
    col_with_mode=[]
    for col in cat_col_list:
        #Filter null
        df = df.filter(df[col].isNull()==False)
        #Find unique_values_with_count
        unique_classes = df.select(col).distinct().rdd.map(lambda x: x[0]).collect()
        unique_values_with_count=[]
        for uc in unique_classes:
             unique_values_with_count.append([uc, df.filter(df[col]==uc).count()])
        #sort unique values w.r.t their count values
        sorted_unique_values_with_count= sorted(unique_values_with_count, key = lambda x: x

        if (verbose==True): print(col, sorted_unique_values_with_count, " and mode is ", sc
        col_with_mode.append([col, sorted_unique_values_with_count[0][0]])
    return col_with_mode
```

pyspark2.py hosted with ♡ by GitHub view raw

This is the function you can use as it is in your code to find the mode of the categorical columns. You just have to give the dataframe with your list of categorical columns. It will return the 2D list of the column name, with their mode, and then you can use this list to fill in the missing values of the categorical columns.

To use the code in an optimal fashion make an extra function that will make use of this mode_of_pyspark_columns function and will automatically fill the missing values with the mode of the categorical columns.

```
#Fill missing values for mode
from pyspark.sql.functions import when, lit

def fill_missing_with_mode(df, cat_col_list):
    col_with_mode =mode_of_pyspark_columns(df, cat_col_list)

    for col, mode in col_with_mode:
        df = df.withColumn(col, when(df[col].isNull()==True,
        lit(mode)).otherwise(df[col]))

    return df
```

pyspark3.py hosted with ♡ by GitHub view raw

Again, this is the piece of the code you can use as it is in your program. It will take dataframe and the list of the categorical column as an input and will return the transformed data frame without impurities or missing values in categorical columns as they are filled by the mode of the column.

#Now this is the time to literally consume the fill_missing_with_mode

cat_col_list=[‘al’, ‘su’, ‘rbc’, ‘pc’, ‘pcc’, ‘ba’, ‘htn’, ‘dm’, ‘cad’, ‘appet’, ‘pe’, ‘ane’]

df = fill_missing_with_mode(df, cat_col_list)

To validate our function count the value of missing values in every column

for col in df.columns:

print(col, “\t”, “with null values: “, df.filter(df[col].isNull()).count())

Output:

```
age with null values: 0

bp with null values: 0

sg with null values: 0

al with null values: 0

su with null values: 0

rbc with null values: 0

pc with null values: 0

pcc with null values: 0

ba with null values: 0

bgr with null values: 0

bu with null values: 0

sc with null values: 0

sod with null values: 0

pot with null values: 0

hemo with null values: 0

pcv with null values: 0

wbcc with null values: 0

rbcc with null values: 0

htn with null values: 0

dm with null values: 0

cad with null values: 0

appet with null values: 0

pe with null values: 0

ane with null values: 0

class with null values: 0
```

> *Hurray, these custom functions are awesome, blossom :-)*

So, this is the end of my detailed session on pyspark dataframe that not only includes the exploratory data analysis alone but also shows readers that how can they use in it machine learning pipeline on some real dataset and covers almost all the cases that are in needed data preprocessing step. Here is the link to complete exploratory github repository.

I tried my best to deliver all the knowledge that is in my brain regarding pyspark dataframe exploratory analysis. If you enjoyed this blog (that I hope so: P), hit the like button ❤.

Python Machine Learning Spark Big Data Data Science