# CSP-554(Big Data Technologies)
# Project Draft

**Name:** Tirumalesh Nagothi
**Date:** 04-16-2023

## Overview
To enable encoding and imputing missing values using the Pandas-API at the most accessible level within Spark, PySpark will be enhanced with new features and methodologies as part of my project to contribute to Apache Spark. Before training machine learning models, I also intend to evaluate the integration of third-party libraries like boto3 and TensorFlow.

I have cloned Spark's complete source code and the pertinent packages from GitHub to accomplish these goals. Next, choose the best route for integrating the new feature and package. Then, to ensure appropriate execution, I built the new features in Python code and tested them against numerous small and large datasets. I conducted two notable test cases covering the complete code better to use the new capabilities, including exception cases.

Integrating my feature can improve PySpark's capabilities, especially in data preparation for machine learning models.

## About the Project area
Apache Spark is an open-source distributed computing framework for big data processing and analytics. It offers Python, R, and SQL APIs and is written in Scala and Java.
At UC Berkeley's AMP Lab, Spark was created in 2009 and released as open-source software in 2010. Since then, it has become one of the most well-liked big data processing frameworks, with a significant and vibrant contributor community.

Spark provides several high-level APIs, including Spark SQL for processing structured data, MLlib for machine learning, GraphX for processing graphs, and streaming for processing real-time data. These APIs can be integrated into a single platform to handle a variety of significant data processing activities.

The ease with which data scientists can use their preferred Python tools with Spark is made available by PySpark's seamless integration with well-known Python libraries like NumPy, Pandas, and SciPy.

## Main problem
If the PySpark user wants to encode the object(string) values before giving them to the machine learning model, then the entire dataset should be converted to numeric values, which needs to be done from scratch. Also, we cannot transform the object type(string) to numeric if the column has one or more null values.

## Solution
The entire work is done using Python by using pandas API in PySpark. The dataset is read into a data frame, and the user can call the functions to make operations such as encoding the values of a particular column with the datatype as an object or string and encoding the entire dataset if it has any object type columns. Also, the null values cannot be encoded, while the non-null values can. So, in addition to the above functions, there is an optional method to run the invalid value replacement for each column or the entire. This works by taking the simple string value of the user's choice as a parameter for string values imputation and the mode of imputation, such as mode, mean or median for numerical values.

## Other integrations
There are two more libraries to integrate, such as TensorFlow and boto3. Even though Spark has libraries like MLlib, integrating with various high-end libraries makes it even broader. TensorFlow lets the users train the Machine Learning and deep learning models. So, running on spark as a part of integration is better as it follows parallel distributed computing. Coming to the boto3, we can call this package inside the PySpark environment to communicate with AWS services such as s3, DynamoDB etc., right from the local machine, which makes it even more flexible to work.

## Objectives
- Define a package to encode the values, replacing the null values with various options in a data frame.
- Testing the above features using various datasets to tune the methods where required.
- Integrating and testing TensorFlow and boto3 to make a part of Spark.

## Design
1. Create a package in PySpark/pandas/spark 'transform.py'.
2. Define the first method with the parameters fill_null_str with the parameters such as data_frame, which is the data frame, col_name, which is the column's name and replace_str_with='Missing', which is the new value to replace with.

2.1 If the user enters the value for replace_str_with that already exists in the column, there will be an intimation that the value already exists, and an input prompt will be raised to proceed with the given existing value or to enter a new value with options(y/n)

2.2 If the input is given as y, the replacement is done with the given existing value. If n, then the user needs to enter a new value.

2.3 This repeats till the user gives the new value or proceeds with the existing value.

2.4 In the end, this method returns the column with the replacement of null values.

3. Define the second method fill_null_num, with the parameters data_frame, col_name, impute_type='mode'.

3.1 This replaces the null values in a numerical column with the impute_type, which should be either mode, mean or median.

3.2 This returns the values with the null values being replaced.

4. Define the third method, fill_null_data, with the parameters data_frame, which is the data frame, fill_null_sc='Missing', which is the value(default value is 'Missing') to replace the null values in string columns and nc_impute_type='Mode' is the type of imputation whether the null values in the numerical columns should be replaced by its mean, median or mode value.

4.1 Iterates through all the columns and checks whether the column type is object or numerical.

4.2 There will be a try exception case where the try will execute for the types such as an object or not(numerical), and an exception is raised if there is any data type that is not object or numerical.

4.3 If the column type is an object, it will use the fill_null_str method to pass the fill_null_sc value and store the returned values in the data frame's column.

4.4 Else, it will call the fill_null_num method, passing the nc_impute_type and storing the returned value in the data frame's column.

4.5 After iterating and imputing the column values, this will return the data frame with no null values.

5. Define the fourth method with the parameters strc_to_numc (data,col_name), where data is the data frame, and col_name is the column's name.

5.1 Firstly, it checks whether the column has null values or not.

5.2 If the column has no null values, the method encodes the values using unique labels for each value. It returns the encoded values and the dictionary, which contains the unique value as the key and its label as the value.

5.3 If the column has null values, it tells the user about them and asks whether to replace them (y/n) using a prompt.

5.4 If y, it asks for the new value to replace the null values.

5.5 Then the method fill_null_str is called, taking the values such as data frame, column name and new value to replace. Then returns the column with the non-null values and a dictionary object with the unique value as the key and its label as the value.

5.6 If n, it doesn't change or replace any null value and returns the initial column and empty dictionary.

6. Define the fifth method, strd_to_numd, with the parameter data_frame, which is the data frame. This will iterate through all the columns in the data frame. And this executes only for the object(string) columns.

6.1 For each object column, the strc_to_numc method is called, taking the values such as data which is the data frame and col_name, which is the column.

6.2 The values returned from the strc_to_numc method will be stored in the column's value in the data frame and the dictionary in the overall dictionary(dictionary in a dictionary).

6.3 At the end of the strd_to_numd, it returns the data frame with no object(string) columns and the dictionary object, which contains the column value as the key and the stored dictionary object as the value.

## Working with the software

- Launch the pyspark
- We can do various tasks using the existing libraries.

**But, for the task, we are aiming:**

- Import the pandas library from pyspark
- Import transform package from the spark in pyspark/pandas
- Read the CSV using pandas API and store it in a variable as a data frame.
- Use the methods inside the transform to perform operations of your choice.
- In case of existing from the pyspark. Use ctrl+d(windows) or cmd+d(mac)

## Client-Server Architecture

### Client-end

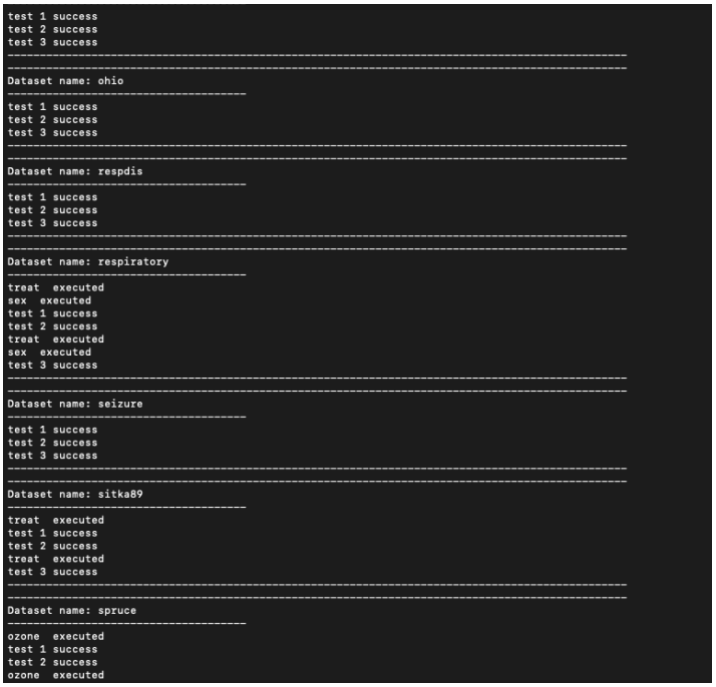
- Client launches the pyspark
- Import modules such as pandas and transform.
- Reads the dataset as a data frame.
- Performs the transformations by calling the methods in the transform library.
- Interacting with the command line interface
- Execution of test files

### Server-end

- Stores the transform package in pyspark.pandas.spark
- Data files are sent
- Execution of methods
- Takes the input, such as calling the method inside the transform package and executes that.
- Stores the test files

## Unit Testing

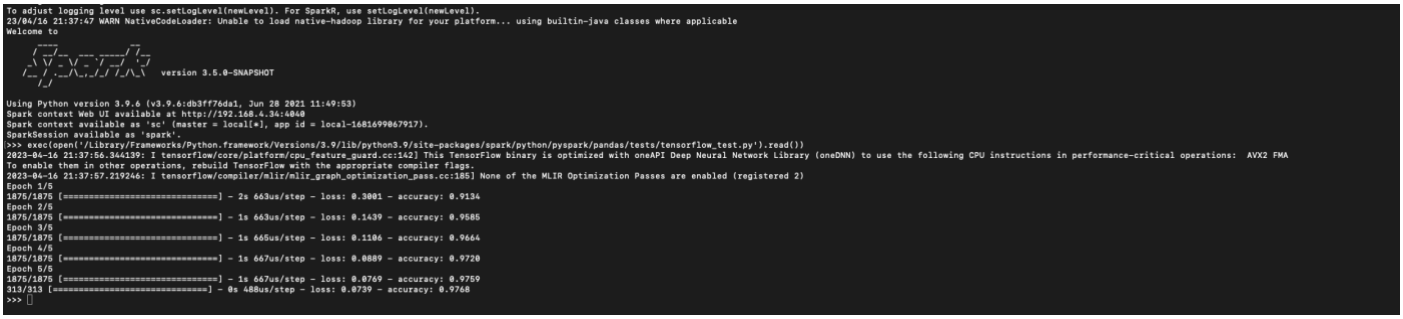
- Unit testing is done by calling the datasets from the local machine, i.e. from a specific folder and the inbuilt modules.
- For testing with the local files, the execution will take the folder path and then appends all the file names with the extension as .csv to a list.
- There will be the execution of three methods: encoding, which runs the strd_to_numd method, null_replace runs the fill_null_data method; and mani runs the null_replace and then returns the data frame after calling inside encoding, which runs the strd_to_numd. All the methods in the unit testing take the test_data as input.
- All the above methods are called inside a method called test_case with the file as the parameter, which is the file name.
- Then the data frame will be called using the file name, performing each method, such as encoding, null_replace and mani, and saving the output data frames as CSV to the path.
- Iter the file list and call the test_case by passing each file name.




Unit testing

## Integration Testing

- We need to run the sample codes to check whether the execution of the boto3 and TensorFlow is within PySpark.
- For boto3, I will get the credentials and execute it by calling the boto3
- I will run the sample dataset for TensorFlow, probably the inbuilt dataset.




Integration testing

## Conclusion

Developing and testing the new feature transform is almost done. I need to test the execution of TensorFlow and boto3. I will finish the proposed statement and the things mentioned in this draft as the result of this project. Since this new feature development is an initial one, there will be more and more implementation in the future.

## References

1. My GitHub repository branch(Latest branch on 04-16-2023): https://github.com/tirumaleshn2458/spark.git

[Note: Please check the latest branch for the latest code commit]

2. Getting started with spark: https://spark.apache.org
3. Pandas API on spark:https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_ps.html
4. My GitHub repository: https://github.com/tirumaleshn2458/spark.git
5. Spark GitHub repository: https://github.com/apache/spark.git
6. Boto3 GitHub repository: https://github.com/boto/boto3
7. TensorFlow GitHub repository: https://github.com/tensorflow/tensorflow