Big Data technologies (CSP 554)

Final project Report

Ramchandra Reddy Sathu (A20526126)

Title: Extending the features of encoding and imputing missing values in PySpark.

Development Summary

To enable encoding and imputing the missing values using Pandas-API within Spark, PySpark will be added with new features and methodologies as a part of this project. Before training machine learning models, I also intend to evaluate the integration of third-party libraries like boto3 and TensorFlow. In order to do this the entire spark code is cloned from the GitHub. Next I choose the better route to integrate the new feature code and package. After that to validate the working of the code I choose to test the code against small and large datasets. Integrating my feature can improve PySpark's capabilities, especially in data preparation for machine learning models.

To begin, we will clone the entire source code of Spark and relevant packages from GitHub and identify the appropriate path to add new features and integrate new packages. The next step involves testing the new features against simple datasets to assess the execution engine's response to the integration. This will help identify any issues that must be resolved before trying with larger datasets. Once the features are stable, the developer will run multiple test cases, including exception cases, to ensure the features are robust and improve their usage. To make it easy for others to understand, I've written inline documentation for each feature, including method and function descriptions.

This development plan demonstrates a structured and thorough approach to contributing to Apache Spark. The project covers testing, documentation, and integration with other libraries, ensuring the new features are reliable, efficient, and easily understood and adopted by other developers.

Objectives

The main aim is to prepare the comprehensive data by converting it into numerical before training the machine learning model. So, to do that, the basic things such as:

- The dataset's null values should all be imputed, or replaced with values by a variety of techniques.
- The object type of values is encoded with numeric values such as labelling with serial numbers, count of each unique value, and one-hot encoding (currently for a Boolean feature).
- Ensure that every value in the dataset is set for training the machine learning model.
  The next one is to test the running of the third-party libraries, such as boto3 and TensorFlow's implementation.

The project aims to produce clean, comprehensive data ready to train a machine learning model. The objectives prioritize data pre-processing and cleaning, followed by testing the compatibility of third-party libraries with the project's goals. This approach ensures that the final output is reliable, efficient, and easily used by other developers.

Next Steps:

In future, there will be many more techniques to come, and will make this feature and integrations work dynamically so that every PySpark user can suitably access them.

Solution outline

The solution outline for the given problem can be summarized as follows:

1. Read the dataset into a PySpark Data Frame using the pandas API.
2. Define functions that the user can call to perform encoding operations. These functions should accept parameters such as the column's name to encode and any other relevant parameters.
3. Implement a function to encode the entire dataset if it contains object type columns.
4. Create an optional method to replace invalid values for each column or the entire dataset. This method should accept a simple string value from the user for string values imputation and the mode of imputation (mode, mean, or median) for numerical values.
5. Handle null values for numerical columns by imputing them using various methods such as mean, median, or mode.
6. Ensure the final output is a clean, comprehensive dataset ready for machine learning model training.
7. Test the implementation using sample data to validate that the encoding and imputation functions work as expected.
8. The solution outline proposes a comprehensive and efficient way to encode and pre-process data using PySpark's pandas API in Python. It ensures that the final output is reliable, efficient, and easily used by other developers.

Relevant literature

1. "Encoding Categorical Variables" by Max Halford (2018) - This article overviews various encoding techniques, including label encoding, and discusses their advantages and disadvantages.

2. "Categorical Encoding Using Label Encoding and One-Hot-Encoder" by GeeksforGeeks (2021) - This tutorial provides a step-by-step guide to implementing label encoding in Python and an explanation of how it works and when to use it.

3. "Label Encoding in Machine Learning" by Arvind N (2019) - This article explains label encoding, including its advantages and disadvantages. It provides examples of how it can be used in machine learning.

In machine learning (ML) and deep learning (DL), distributed systems have become more and more crucial since large-scale data sets and intricate models demand more processing power than a single computer can offer. A well-liked framework for creating and honing ML and DL models, TensorFlow has strong support for distributed training. Here is a list of relevant list of resources than can be referred on using distributed computing with TensorFlow:

4. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems" by Martín Abadi et al. (2016) - This paper introduces TensorFlow distributed computing capabilities, including the use of parameter servers to coordinate model updates across multiple machines.

"Horovod: fast and easy distributed deep learning in TensorFlow" by Alex Sergeev et al. (2018) - This paper describes Horovod. This framework simplifies distributed training of deep learning models in TensorFlow by providing a simple API and efficient communication primitives.

Proposed system

1. The proposed system for achieving the objectives of adding new features and integrating new packages to PySpark can be broken down into several steps:
2. Clone the entire source code of Spark and other relevant packages from GitHub to a local development environment.
3. Identify the appropriate path within the source code where the new feature or package integration should be added.
4. Write the new feature or package integration using Python code compatible with PySpark.
5. Test the new feature or package integration against simple datasets to ensure the execution engine reacts as expected.
6. Test the new feature or package integration against larger datasets to identify any alternative modes of implementation that may be necessary for scalability.
7. Run multiple test cases, including exception cases, to improve the usage and identify potential bugs or errors.
8. Write inline documentation for each feature or function to make it easy to understand and use.
9. Submit the new feature or package integration as a pull request to the Spark GitHub repository for review and inclusion in future releases.

By following this proposed system, it is possible to add new features and integrate new packages into PySpark while ensuring that the execution engine remains stable and scalable. The testing and documentation steps help to ensure that the new functionality is practical and accessible to other developers while also minimizing the potential for errors or bugs.

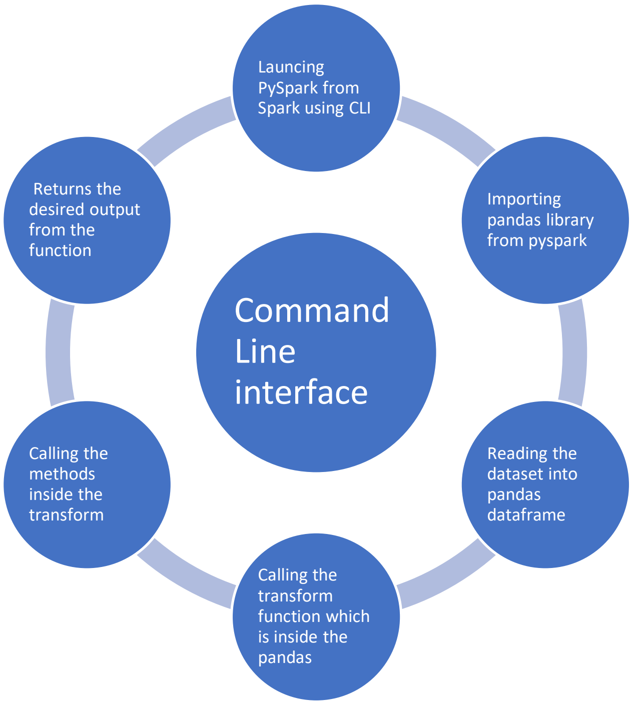
Architecture:

The system architecture would be made up of several parts that cooperate to provide the required functionality. The input module, which reads the dataset into a data frame, is the first part. Through the user interface module, which allows them to choose the desired dataset actions, the user interacts with the system. These actions include encoding the whole dataset, if it contains any object-type columns, and encoding the values of a specific column with the datatype as an object or string.

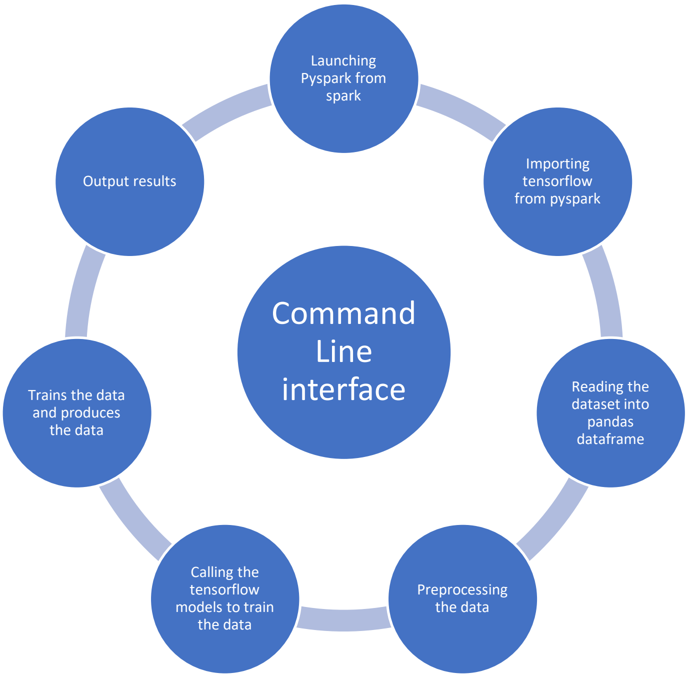
The encoding module, which is the second part, actually encodes the dataset using the input from the user. This module makes sure that although non-null values may be encoded, null values cannot. Using the invalid value replacement for all columns or just some of them is an optional strategy. In order for this to function, the user-selected simple string value and the imputation mode—such as the mode, mean, or median for numerical values—must be provided.

Lastly, it is the output module's responsibility to provide the user with a format that is easy to grasp when viewing the findings. The data encoding and imputation procedure is covered in depth in the reports generated by this module's reporting module. These reports contain information on the methods used for encoding and imputation, the size of the dataset, and the time required to complete the operations.

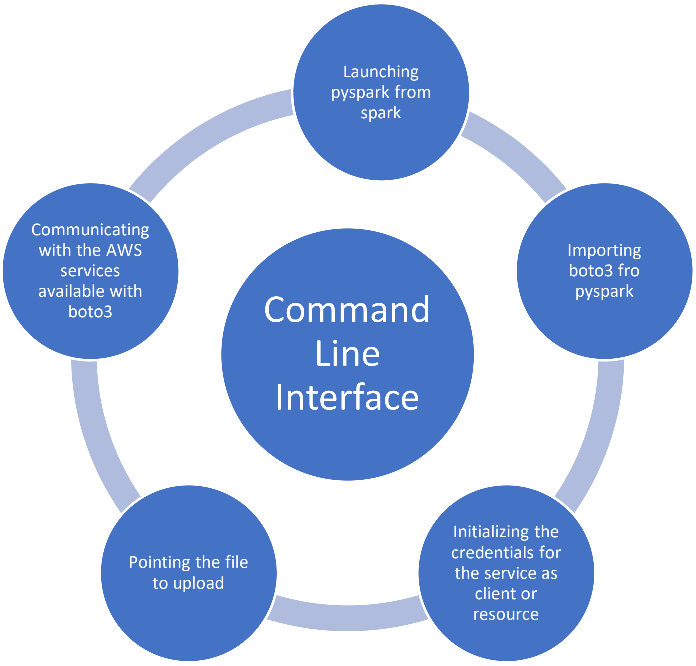
All things considered, the system architecture is made to be both adaptable and scalable, enabling the addition of new features and techniques as needed. It is a useful tool for academics and data scientists since it offers an intuitive user interface for imputation and data encoding.


Launcing PySpark from Spark using CLI
Importing pandas library from pyspark
Reading the dataset into pandas dataframe
Calling the transform function which is inside the pandas
Calling the methods inside the transform
Returns the desired output from the function
Command Line interface

Transform Method


Launching Pyspark from spark
Importing tensorflow from pyspark
Reading the dataset into pandas dataframe
Preprocessing the data
Calling the tensorflow models to train the data
Trains the data and produces the data
Output results
Command Line interface

Running Tensorflow on pyspark


Launching pyspark from spark
Importing boto3 fro pyspark
Initializing the credentials for the service as client or resource
Pointing the file to upload
Communicating with the AWS services available with boto3
Command Line Interface

Running boto3 on pyspark

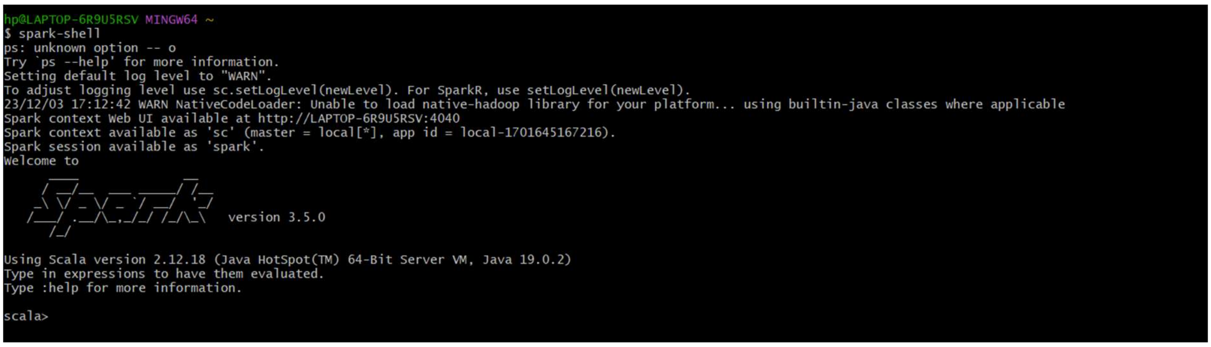
Software components

- Spark (PySpark)
- Pandas
- Python 3.9
- Command Shell
- GitHub
- Jupyter Notebook
- Visual Studio Code IDE

Installation of PySpark is done in two ways:

- By cloning the source code from the spark GitHub repository .
- By installing using pip command, i.e. pip install pyspark
-

To run Spark on the command prompt, we can simply type "spark-shell" to launch Spark. However, in order to implement any features in Spark, we need to use the Scala programming language. If we want to use Python instead of Scala, we can use the "pyspark" command to write our code in Python. This allows us to use the Python programming language to leverage the capabilities of the Spark engine.



```
hp@LAPTOP-6R9U5RSV MINGW64 ~
$ spark-shell
ps: unknown option -- o
Try `ps --help' for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/12/03 17:12:42 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://LAPTOP-6R9U5RSV:4040
Spark context available as 'sc' (master = local[*], app id = local-1701645167216).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Scala version 2.12.18 (Java HotSpot(TM) 64-Bit Server VM, Java 19.0.2)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Installation of TensorFlow and boto3 and copying into PySpark:

```
#installation for boto3
import os
def run():
    try:
        import pyspark.boto3
    except:
        print('Installing boto3...')
        os.system('pip install boto3 -q -q -q')
        print('Installed boto3')
    import boto3
    boto3_path = os.path.dirname(boto3.__file__)
    import pyspark
    source_path = os.path.dirname(pyspark.__file__)
    boto3_folder = "boto3"
    try:
        boto3_source_path=os.path.join(source_path,boto3_folder)
    except:
        pass
    import shutil
    try:
        shutil.copytree(boto3_path,boto3_source_path)
    except:
        pass

#installation for tensorflow
    #import os
    try:
        import pyspark.tensorflow
    except:
        print('Installing tensorflow...')
        os.system('pip install tensorflow -q -q -q')
        print('Installed tensorflow')
    import tensorflow
    tensorflow_path = os.path.dirname(tensorflow.__file__)
    import pyspark
    source_path = os.path.dirname(pyspark.__file__)
    ts_folder = "tensorflow"

    try:

        tensorflow_source_path=os.path.join(source_path,ts_folder)

    except:

        pass

    import shutil

    #copying files in the tensorflow directory to the pyspark's tensorflow
directory

    try:

        shutil.copytree(tensorflow_path,tensorflow_source_path)

    except:

        pass
```

Design and Implementation:

As a part of this new feature implementation we create a new file called 'transform.py' in the location python/pyspark/spark in the spark codebase.

The following method are written in the file which perform the necessary operations.

fill_null_str

- **fill_null_str** is used to easily replace the null values for a string(object) column in the data frame. It can be considered the single line code for filling the null values with the user's string choice, making it more interactive.

    Parameters:

    data_frame: Data frame object, Pandas data frame object, which is 2-dimensional.

    col_name: String, Column name of the data_frame to get the null values filled.

    replace_str_with: String, Default 'Missing'.

    The value which should get replaced in place of the null value.

    Returns:

    After replacing the null values, the series contains all the values in the column.

fill_null_num

- **fill_null_num** replaces the null values in the numerical column of the data frame. Filling the null values using a single line of code with the choice of imputation type.

  Parameters:

  data_frame: Data frame object, Pandas data frame object, which is 2-dimensional.

  col_name : String, Numerical column name of the data_frame to fill the null values.

  impute_type: String, Default 'mode'

Type of imputation, i.e. whether the null values in the column should be replaced with the mode or mean or median value of the column.

Returns:

Series

which contains all the values of the column after replacing the null values.

fill_null_data

- **fill_null_data** is used to replace the null values for both the object and numerical columns in the data frame.

  This is based on the fill_null_str and fill_null_num methods.

  Parameters:

  data_frame: Data frame object, pandas data frame object which is 2-Dimensional.

  fill_null_sc: String value, default 'Missing'

  It takes the value(any) to replace with the null values in

  string columns of the data_frame.

  nc_impute_type: String value, default 'mode.'

  Takes the value(mode, mean, median) to apply the kind of imputation

  to replace the null values in numerical columns of the data_frame.

  Returns:

  A pandas data frame object with no null values in the object and numerical columns.

strc_to_numc

- **strc_to_numc** method encodes (labels) the string values in a column to the numerical values, i.e. Converts the entire object column's dtype to the numerical column.

  This is also partially reliable on fill_null_str to replace the null values in the column since it cannot encode the columns with the null values.

  Parameters

  data: DataFrame object, which is the 2-dimensional pandas data frame object.

  col_name: String value

  Specific column's name you wish to encode the values in.

  Returns

  A Series object

  Contains the encoded(numerical) values.

  A dictionary object

  Contains the key-value pairs that reference the numerical value(value) for each the string value(key) in the column.

strd_to_numd

- **strd_to_numd** is used to encode all the string values in the columns to numerical values by applying labels for string values in columns.

  This relies on the strc_to_numc method to encode a single column value in the data frame.

  Parameters:

  data_frame: DataFrame object, which is the 2-dimensional pandas data frame object.

  Returns:

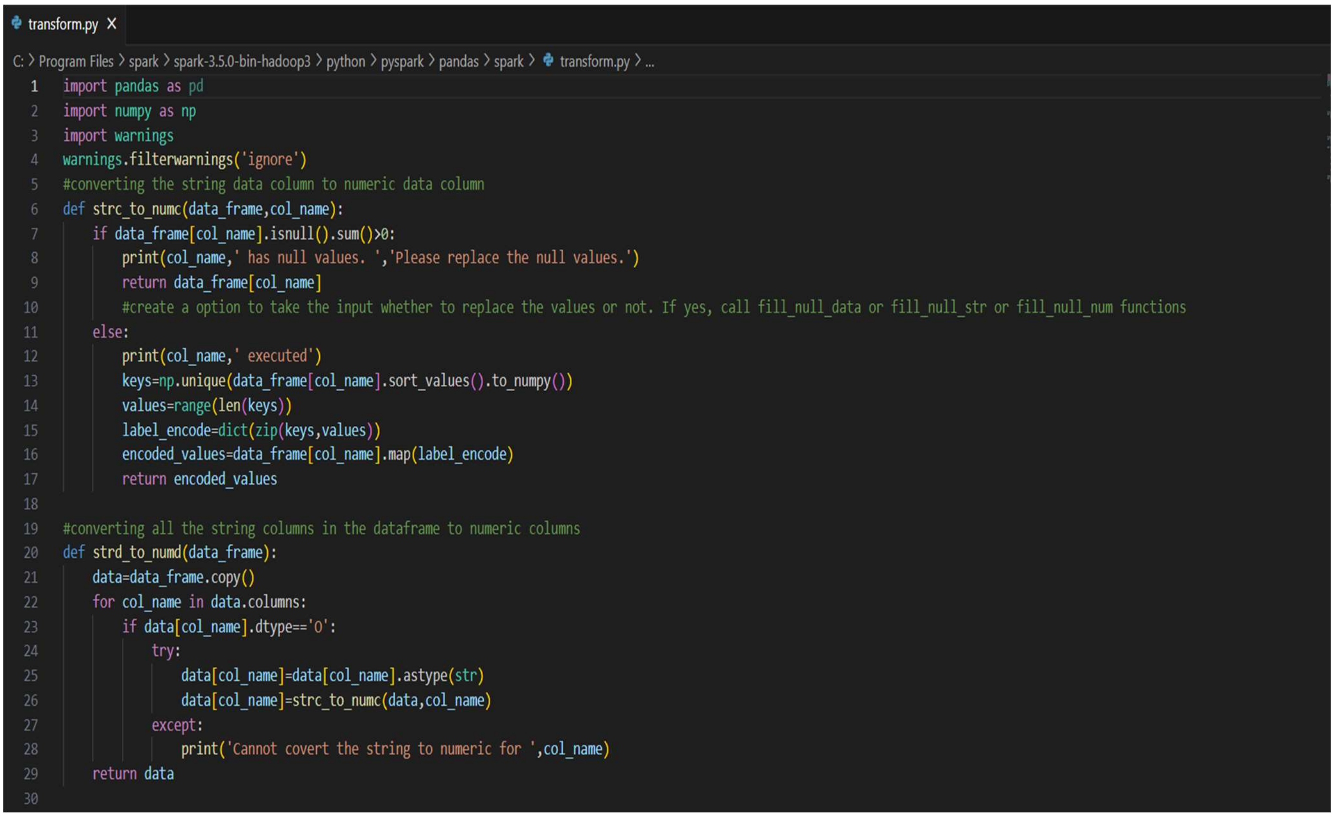
  A data frame object -- contains all the initial columns where the string(object) columns are encoded to numerical values.

  A dictionary object -- contains the dictionaries of each column's values and their labels returned from strc_to_numc.

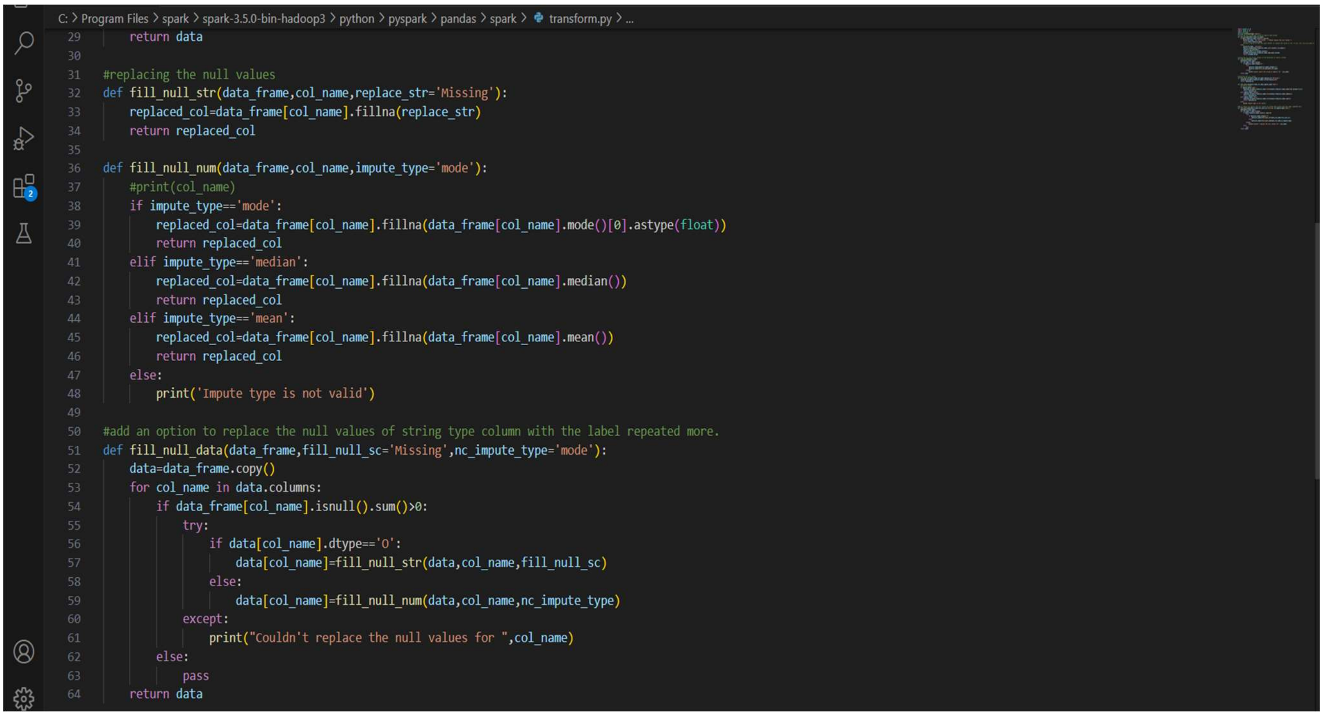
Coding and Testing:

First we test the newly introduced ‘transform.py’ file which contains the newly introduced methods for filling the missing values.

Transform.py


transform.py X
C: > Program Files > spark > spark-3.5.0-bin-hadoop3 > python > pyspark > pandas > spark > transform.py > ...

```python
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
#converting the string data column to numeric data column
def strc_to_numc(data_frame,col_name):
    if data_frame[col_name].isnull().sum()>0:
        print(col_name,' has null values. ','Please replace the null values.')
        return data_frame[col_name]
        #create a option to take the input whether to replace the values or not. If yes, call fill_null_data or fill_null_str or fill_null_num functions
    else:
        print(col_name,' executed')
        keys=np.unique(data_frame[col_name].sort_values().to_numpy())
        values=range(len(keys))
        label_encode=dict(zip(keys,values))
        encoded_values=data_frame[col_name].map(label_encode)
        return encoded_values

#converting all the string columns in the dataframe to numeric columns
def strd_to_numd(data_frame):
    data=data_frame.copy()
    for col_name in data.columns:
        if data[col_name].dtype=='O':
            try:
                data[col_name]=data[col_name].astype(str)
                data[col_name]=strc_to_numc(data,col_name)
            except:
                print('Cannot covert the string to numeric for ',col_name)
    return data

```

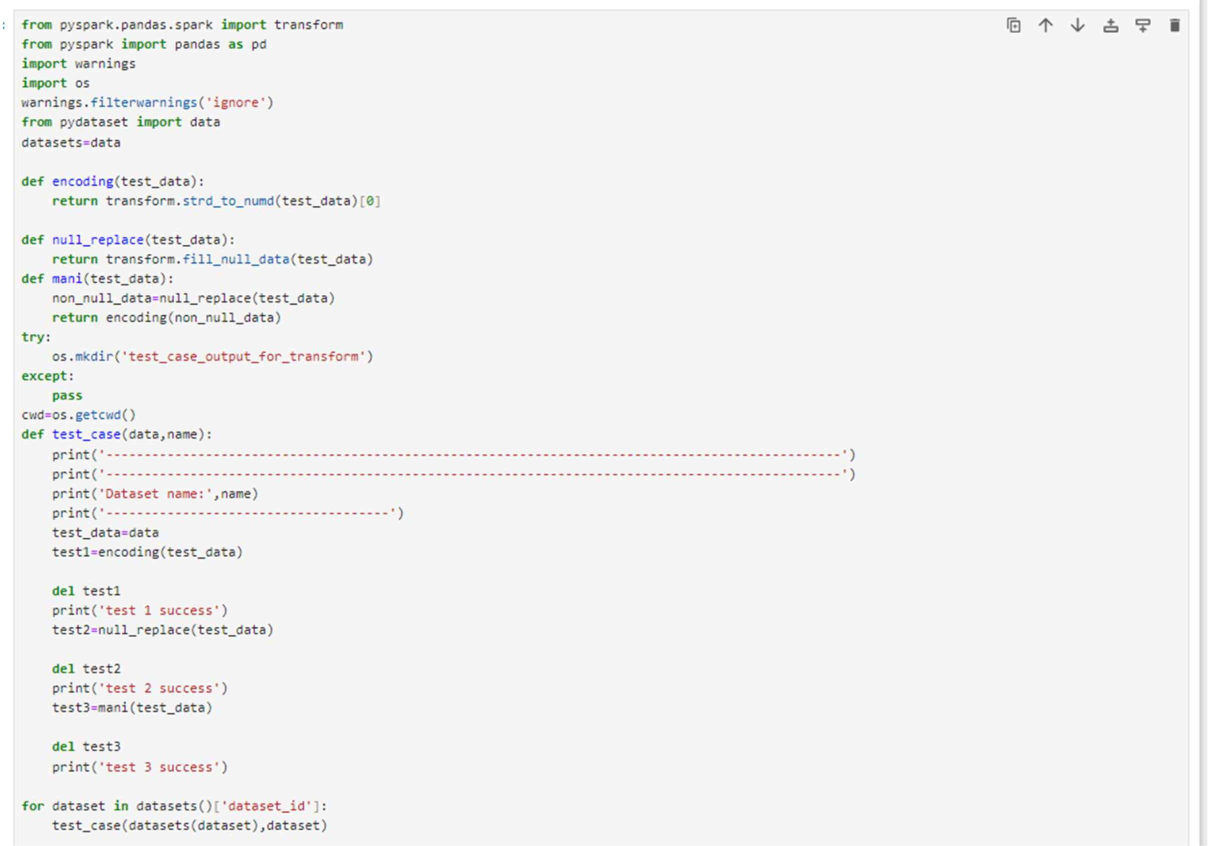


C: > Program Files > spark > spark-3.5.0-bin-hadoop3 > python > pyspark > pandas > spark > transform.py > ...

```python
    return data

#replacing the null values
def fill_null_str(data_frame,col_name,replace_str='Missing'):
    replaced_col=data_frame[col_name].fillna(replace_str)
    return replaced_col

def fill_null_num(data_frame,col_name,impute_type='mode'):
    #print(col_name)
    if impute_type=='mode':
        replaced_col=data_frame[col_name].fillna(data_frame[col_name].mode()[0].astype(float))
        return replaced_col
    elif impute_type=='median':
        replaced_col=data_frame[col_name].fillna(data_frame[col_name].median())
        return replaced_col
    elif impute_type=='mean':
        replaced_col=data_frame[col_name].fillna(data_frame[col_name].mean())
        return replaced_col
    else:
        print('Impute type is not valid')

#add an option to replace the null values of string type column with the label repeated more.
def fill_null_data(data_frame,fill_null_sc='Missing',nc_impute_type='mode'):
    data=data_frame.copy()
    for col_name in data.columns:
        if data_frame[col_name].isnull().sum()>0:
            try:
                if data[col_name].dtype=='O':
                    data[col_name]=fill_null_str(data,col_name,fill_null_sc)
                else:
                    data[col_name]=fill_null_num(data,col_name,nc_impute_type)
            except:
                print("Couldn't replace the null values for ",col_name)
        else:
            pass
    return data
```

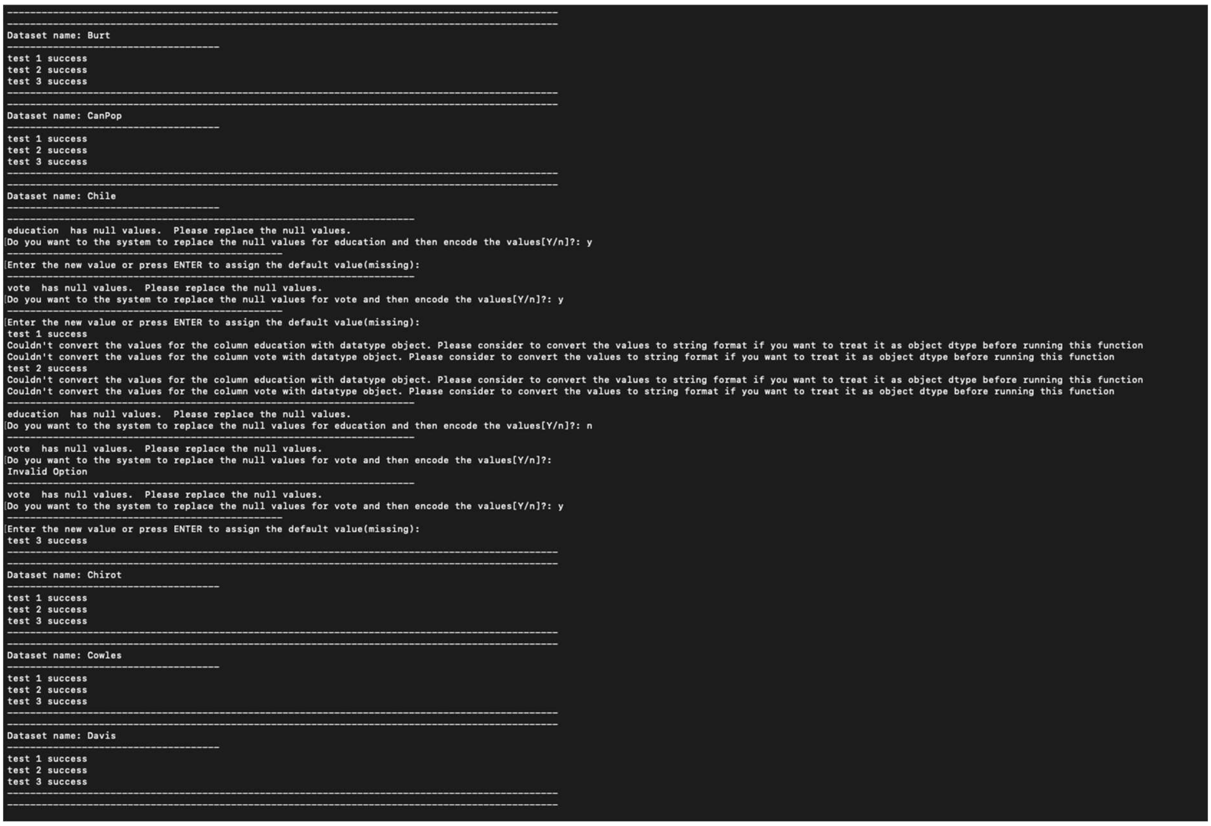
Testing for transform.py

TestCase1:



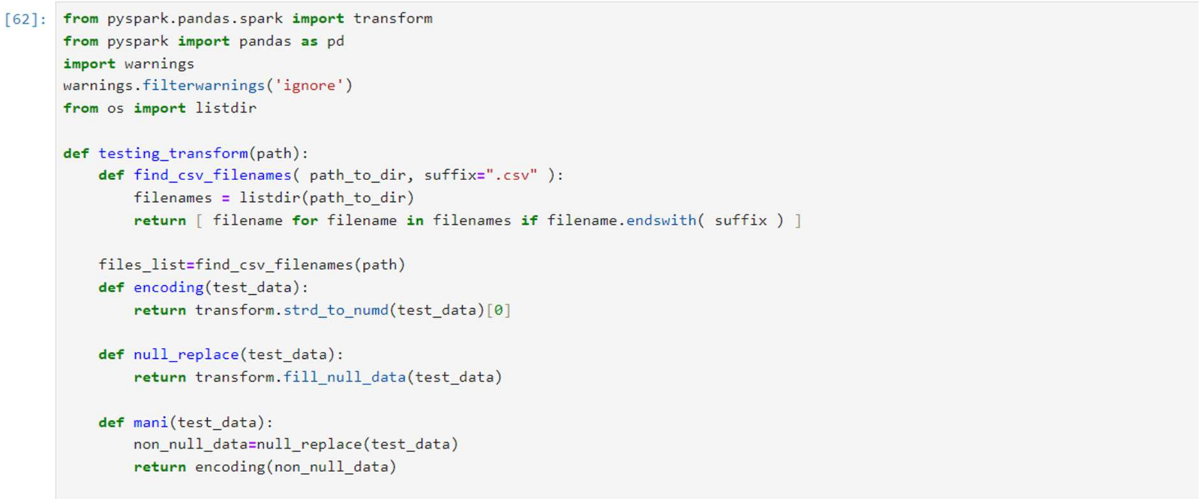
```python
from pyspark.pandas.spark import transform
from pyspark import pandas as pd
import warnings
import os
warnings.filterwarnings('ignore')
from pydataset import data
datasets=data

def encoding(test_data):
    return transform.strd_to_numd(test_data)[0]

def null_replace(test_data):
    return transform.fill_null_data(test_data)
def mani(test_data):
    non_null_data=null_replace(test_data)
    return encoding(non_null_data)
try:
    os.mkdir('test_case_output_for_transform')
except:
    pass
cwd=os.getcwd()
def test_case(data,name):
    print('------------------------------------------------------------------------------------------------')
    print('------------------------------------------------------------------------------------------------')
    print('Dataset name:',name)
    print('--------------------------------------')
    test_data=data
    test1=encoding(test_data)

    del test1
    print('test 1 success')
    test2=null_replace(test_data)

    del test2
    print('test 2 success')
    test3=mani(test_data)

    del test3
    print('test 3 success')

for dataset in datasets()['dataset_id']:
    test_case(datasets(dataset),dataset)
```

Output:



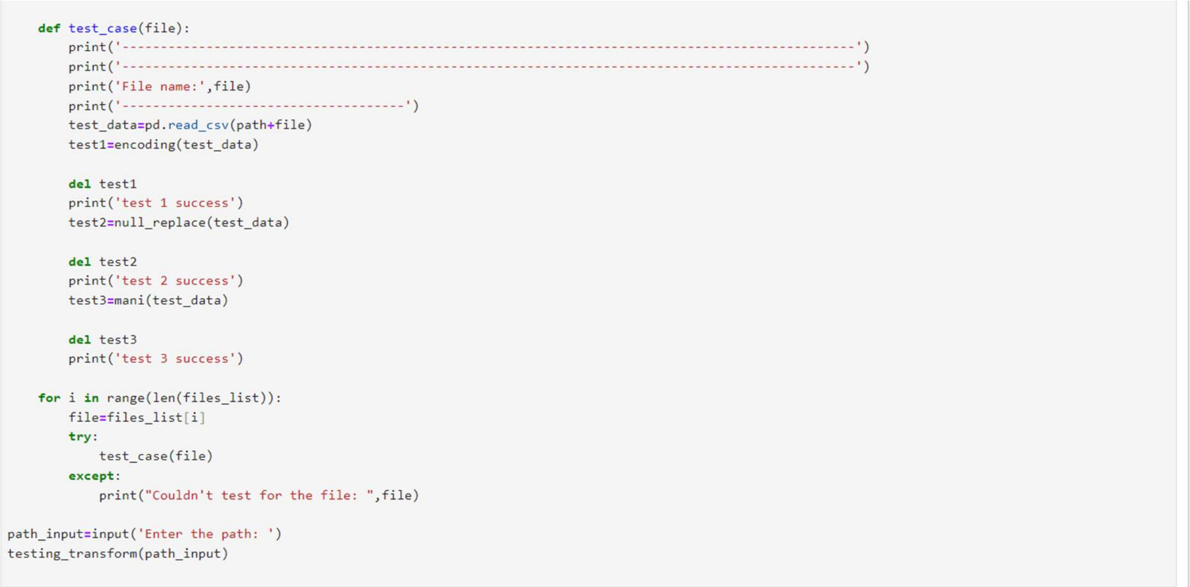
```
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
Dataset name: Burt
-------------------------------------
test 1 success
test 2 success
test 3 success
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
Dataset name: CanPop
-------------------------------------
test 1 success
test 2 success
test 3 success
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
Dataset name: Chile
-------------------------------------
-----------------------------------------------------------------------
education  has null values.  Please replace the null values.
Do you want to the system to replace the null values for education and then encode the values[Y/n]?: y
------------------------------------------------
Enter the new value or press ENTER to assign the default value(missing):
-----------------------------------------------------------------------
vote  has null values.  Please replace the null values.
Do you want to the system to replace the null values for vote and then encode the values[Y/n]?: y
------------------------------------------------
Enter the new value or press ENTER to assign the default value(missing):
test 1 success
Couldn't convert the values for the column education with datatype object. Please consider to convert the values to string format if you want to treat it as object dtype before running this function
Couldn't convert the values for the column vote with datatype object. Please consider to convert the values to string format if you want to treat it as object dtype before running this function
test 2 success
Couldn't convert the values for the column education with datatype object. Please consider to convert the values to string format if you want to treat it as object dtype before running this function
Couldn't convert the values for the column vote with datatype object. Please consider to convert the values to string format if you want to treat it as object dtype before running this function
-----------------------------------------------------------------------
education  has null values.  Please replace the null values.
Do you want to the system to replace the null values for education and then encode the values[Y/n]?: n
-----------------------------------------------------------------------
vote  has null values.  Please replace the null values.
Do you want to the system to replace the null values for vote and then encode the values[Y/n]?:
Invalid Option
-----------------------------------------------------------------------
vote  has null values.  Please replace the null values.
Do you want to the system to replace the null values for vote and then encode the values[Y/n]?: y
------------------------------------------------
Enter the new value or press ENTER to assign the default value(missing):
test 3 success
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
Dataset name: Chirot
-------------------------------------
test 1 success
test 2 success
test 3 success
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
Dataset name: Cowles
-------------------------------------
test 1 success
test 2 success
test 3 success
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
Dataset name: Davis
-------------------------------------
test 1 success
test 2 success
test 3 success
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
```

TestCase 2 for testing Transform.py:



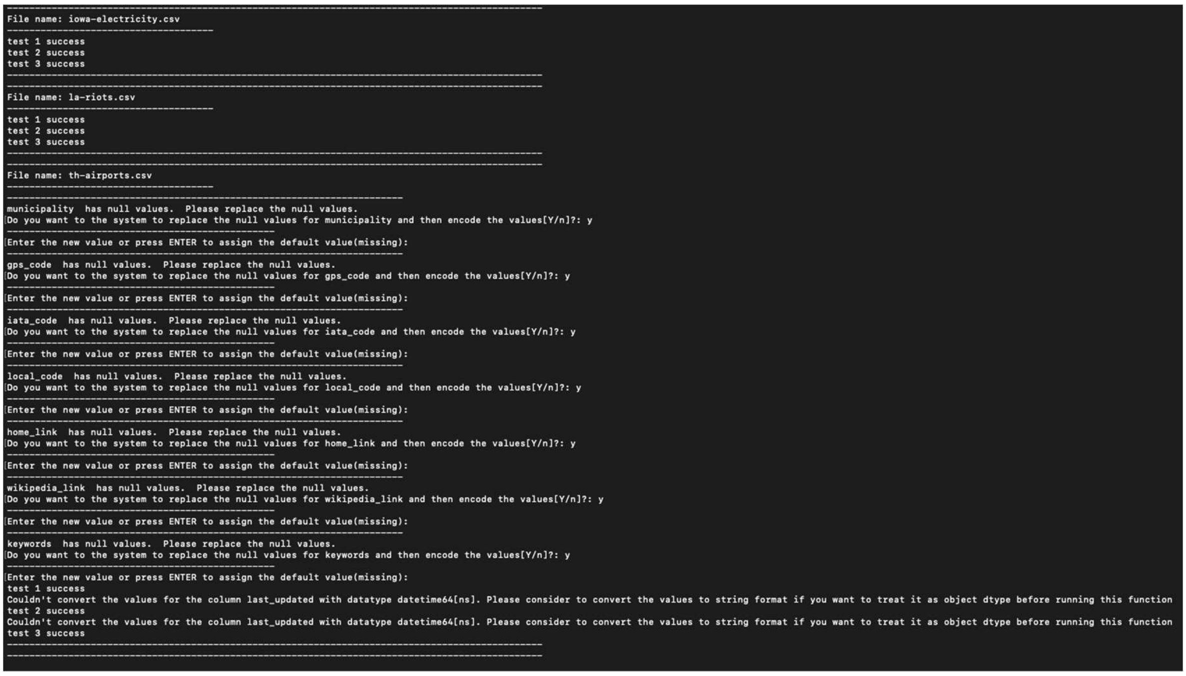
[62]:

```python
from pyspark.pandas.spark import transform
from pyspark import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from os import listdir

def testing_transform(path):
    def find_csv_filenames( path_to_dir, suffix=".csv" ):
        filenames = listdir(path_to_dir)
        return [ filename for filename in filenames if filename.endswith( suffix ) ]

    files_list=find_csv_filenames(path)
    def encoding(test_data):
        return transform.strd_to_numd(test_data)[0]

    def null_replace(test_data):
        return transform.fill_null_data(test_data)

    def mani(test_data):
        non_null_data=null_replace(test_data)
        return encoding(non_null_data)
```



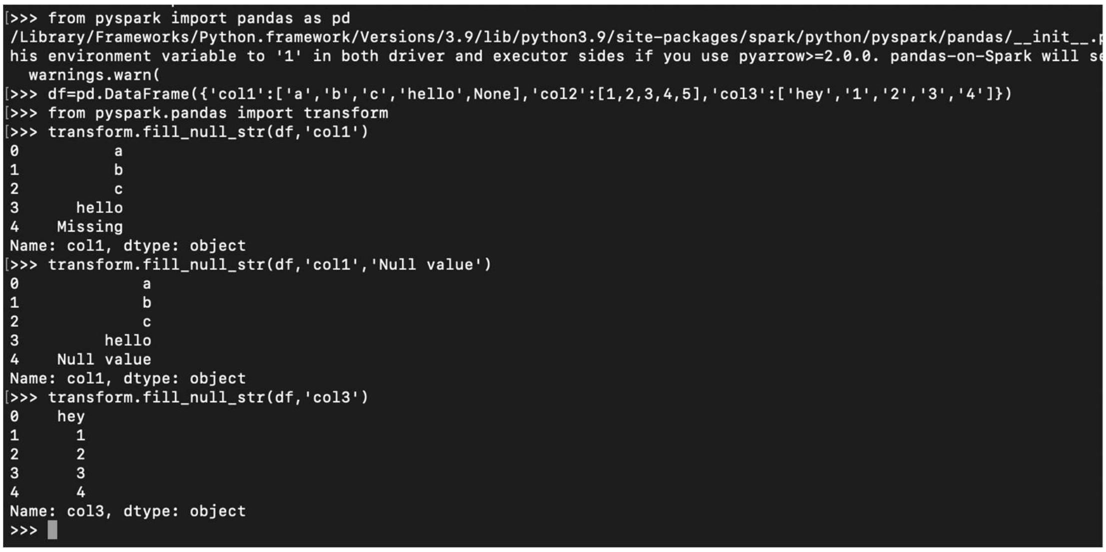
```
    def test_case(file):
        print('----------------------------------------------------------------------------------------------------')
        print('----------------------------------------------------------------------------------------------------')
        print('File name:',file)
        print('-------------------------------------')
        test_data=pd.read_csv(path+file)
        test1=encoding(test_data)

        del test1
        print('test 1 success')
        test2=null_replace(test_data)

        del test2
        print('test 2 success')
        test3=mani(test_data)

        del test3
        print('test 3 success')

    for i in range(len(files_list)):
        file=files_list[i]
        try:
            test_case(file)
        except:
            print("Couldn't test for the file: ",file)

path_input=input('Enter the path: ')
testing_transform(path_input)
```

Output:



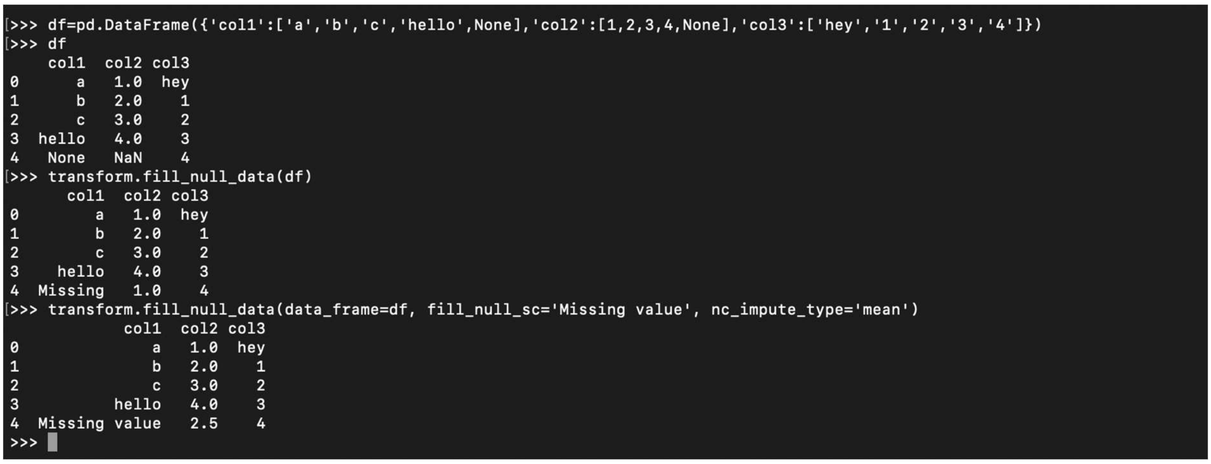
```
----------------------------------------------------------------------------------------------------
File name: iowa-electricity.csv
-------------------------------------
test 1 success
test 2 success
test 3 success
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
File name: la-riots.csv
-------------------------------------
test 1 success
test 2 success
test 3 success
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
File name: th-airports.csv
-------------------------------------
-----------------------------------------------------------------------
municipality  has null values.  Please replace the null values.
[Do you want to the system to replace the null values for municipality and then encode the values[Y/n]?: y
------------------------------------------------
[Enter the new value or press ENTER to assign the default value(missing):
-----------------------------------------------------------------------
gps_code  has null values.  Please replace the null values.
[Do you want to the system to replace the null values for gps_code and then encode the values[Y/n]?: y
------------------------------------------------
[Enter the new value or press ENTER to assign the default value(missing):
-----------------------------------------------------------------------
iata_code  has null values.  Please replace the null values.
[Do you want to the system to replace the null values for iata_code and then encode the values[Y/n]?: y
------------------------------------------------
[Enter the new value or press ENTER to assign the default value(missing):
-----------------------------------------------------------------------
local_code  has null values.  Please replace the null values.
[Do you want to the system to replace the null values for local_code and then encode the values[Y/n]?: y
------------------------------------------------
[Enter the new value or press ENTER to assign the default value(missing):
-----------------------------------------------------------------------
home_link  has null values.  Please replace the null values.
[Do you want to the system to replace the null values for home_link and then encode the values[Y/n]?: y
------------------------------------------------
[Enter the new value or press ENTER to assign the default value(missing):
-----------------------------------------------------------------------
wikipedia_link  has null values.  Please replace the null values.
[Do you want to the system to replace the null values for wikipedia_link and then encode the values[Y/n]?: y
------------------------------------------------
[Enter the new value or press ENTER to assign the default value(missing):
-----------------------------------------------------------------------
keywords  has null values.  Please replace the null values.
[Do you want to the system to replace the null values for keywords and then encode the values[Y/n]?: y
------------------------------------------------
[Enter the new value or press ENTER to assign the default value(missing):
test 1 success
Couldn't convert the values for the column last_updated with datatype datetime64[ns]. Please consider to convert the values to string format if you want to treat it as object dtype before running this function
test 2 success
Couldn't convert the values for the column last_updated with datatype datetime64[ns]. Please consider to convert the values to string format if you want to treat it as object dtype before running this function
test 3 success
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
```

Output for fill_null_str:



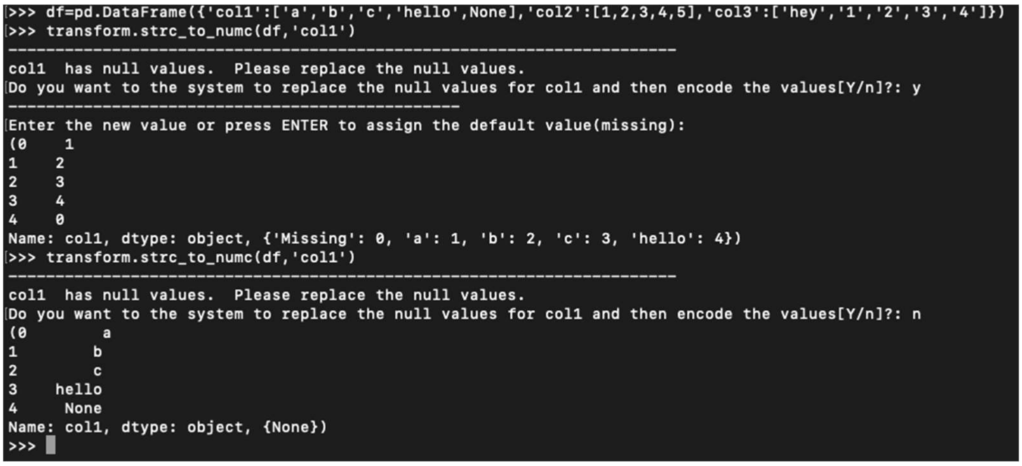
```
[>>> from pyspark import pandas as pd
 /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/spark/python/pyspark/pandas/__init__.p
 his environment variable to '1' in both driver and executor sides if you use pyarrow>=2.0.0. pandas-on-Spark will se
   warnings.warn(
[>>> df=pd.DataFrame({'col1':['a','b','c','hello',None],'col2':[1,2,3,4,5],'col3':['hey','1','2','3','4']})
[>>> from pyspark.pandas import transform
[>>> transform.fill_null_str(df,'col1')
 0          a
 1          b
 2          c
 3      hello
 4    Missing
 Name: col1, dtype: object
[>>> transform.fill_null_str(df,'col1','Null value')
 0             a
 1             b
 2             c
 3         hello
 4    Null value
 Name: col1, dtype: object
[>>> transform.fill_null_str(df,'col3')
 0    hey
 1      1
 2      2
 3      3
 4      4
 Name: col3, dtype: object
 >>> 
```

Output for fill_null_data:

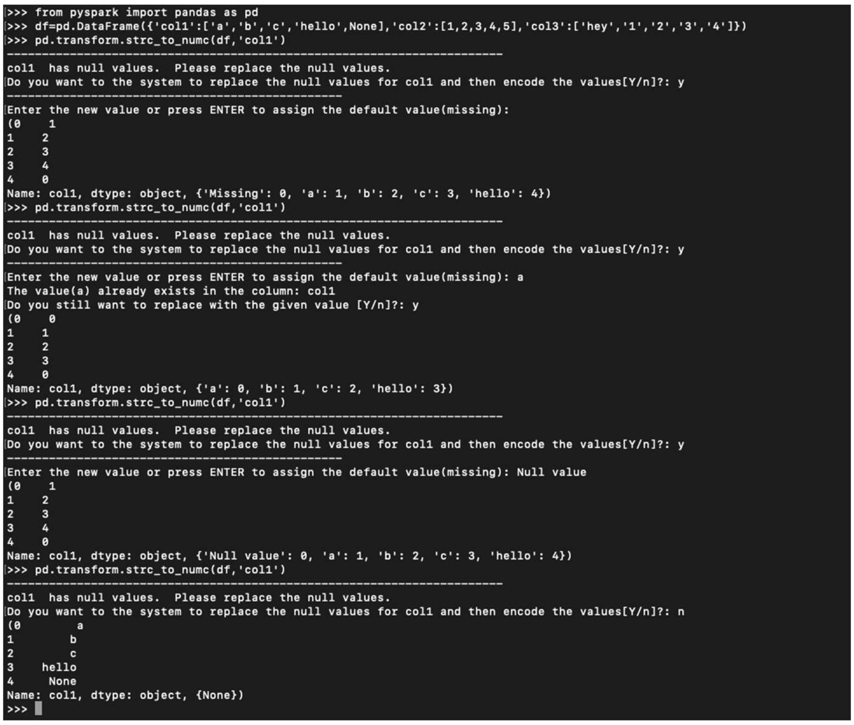


```
>>> df=pd.DataFrame({'col1':['a','b','c','hello',None],'col2':[1,2,3,4,None],'col3':['hey','1','2','3','4']})
>>> df
    col1  col2 col3
0      a   1.0  hey
1      b   2.0    1
2      c   3.0    2
3  hello   4.0    3
4   None   NaN    4
>>> transform.fill_null_data(df)
      col1  col2 col3
0        a   1.0  hey
1        b   2.0    1
2        c   3.0    2
3    hello   4.0    3
4  Missing   1.0    4
>>> transform.fill_null_data(data_frame=df, fill_null_sc='Missing value', nc_impute_type='mean')
            col1  col2 col3
0              a   1.0  hey
1              b   2.0    1
2              c   3.0    2
3          hello   4.0    3
4  Missing value   2.5    4
>>> 
```

Output for strc_to_numc:

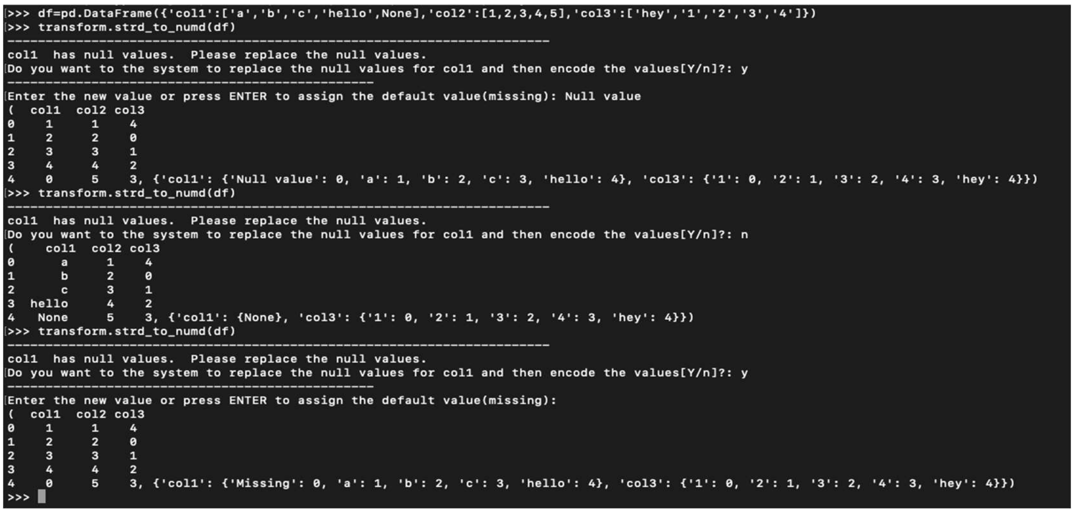


```
>>> df=pd.DataFrame({'col1':['a','b','c','hello',None],'col2':[1,2,3,4,5],'col3':['hey','1','2','3','4']})
>>> transform.strc_to_numc(df,'col1')
-----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: y
----------------------------------------------
Enter the new value or press ENTER to assign the default value(missing):
(0    1
1    2
2    3
3    4
4    0
Name: col1, dtype: object, {'Missing': 0, 'a': 1, 'b': 2, 'c': 3, 'hello': 4})
>>> transform.strc_to_numc(df,'col1')
-----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: n
(0        a
1        b
2        c
3    hello
4     None
Name: col1, dtype: object, {None})
>>> 
```

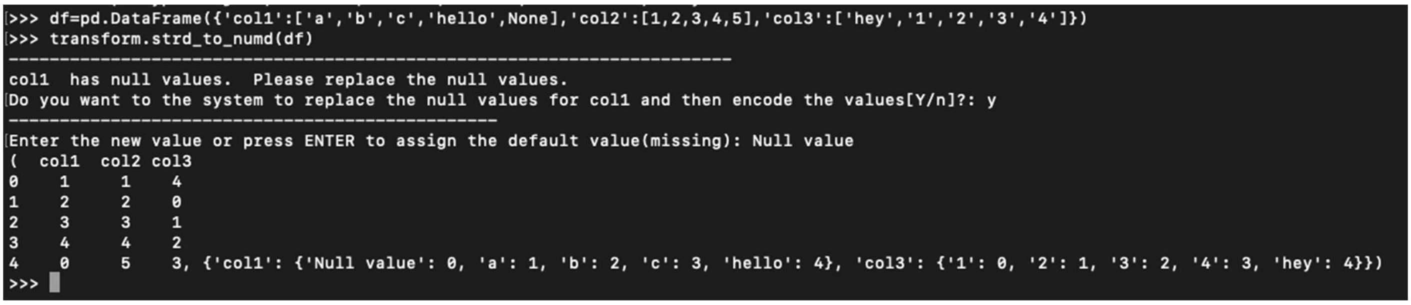


```
>>> from pyspark import pandas as pd
>>> df=pd.DataFrame({'col1':['a','b','c','hello',None],'col2':[1,2,3,4,5],'col3':['hey','1','2','3','4']})
>>> pd.transform.strc_to_numc(df,'col1')
-----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: y
----------------------------------------------
Enter the new value or press ENTER to assign the default value(missing):
(0    1
1    2
2    3
3    4
4    0
Name: col1, dtype: object, {'Missing': 0, 'a': 1, 'b': 2, 'c': 3, 'hello': 4})
>>> pd.transform.strc_to_numc(df,'col1')
-----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: y
----------------------------------------------
Enter the new value or press ENTER to assign the default value(missing): a
The value(a) already exists in the column: col1
Do you still want to replace with the given value [Y/n]?: y
(0    0
1    1
2    2
3    3
4    0
Name: col1, dtype: object, {'a': 0, 'b': 1, 'c': 2, 'hello': 3})
>>> pd.transform.strc_to_numc(df,'col1')
-----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: y
----------------------------------------------
Enter the new value or press ENTER to assign the default value(missing): Null value
(0    1
1    2
2    3
3    4
4    0
Name: col1, dtype: object, {'Null value': 0, 'a': 1, 'b': 2, 'c': 3, 'hello': 4})
>>> pd.transform.strc_to_numc(df,'col1')
-----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: n
(0        a
1        b
2        c
3    hello
4     None
Name: col1, dtype: object, {None})
>>> 
```

Output for str_to_numd:



```
>>> df=pd.DataFrame({'col1':['a','b','c','hello',None],'col2':[1,2,3,4,5],'col3':['hey','1','2','3','4']})
>>> transform.strd_to_numd(df)
----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: y
------------------------------------------------
Enter the new value or press ENTER to assign the default value(missing): Null value
(  col1  col2 col3
0    1     1    4
1    2     2    0
2    3     3    1
3    4     4    2
4    0     5    3, {'col1': {'Null value': 0, 'a': 1, 'b': 2, 'c': 3, 'hello': 4}, 'col3': {'1': 0, '2': 1, '3': 2, '4': 3, 'hey': 4}})
>>> transform.strd_to_numd(df)
----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: n
(    col1  col2 col3
0      a     1    4
1      b     2    0
2      c     3    1
3  hello     4    2
4   None     5    3, {'col1': {None}, 'col3': {'1': 0, '2': 1, '3': 2, '4': 3, 'hey': 4}})
>>> transform.strd_to_numd(df)
----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: y
------------------------------------------------
Enter the new value or press ENTER to assign the default value(missing):
(  col1  col2 col3
0    1     1    4
1    2     2    0
2    3     3    1
3    4     4    2
4    0     5    3, {'col1': {'Missing': 0, 'a': 1, 'b': 2, 'c': 3, 'hello': 4}, 'col3': {'1': 0, '2': 1, '3': 2, '4': 3, 'hey': 4}})
>>> 
```

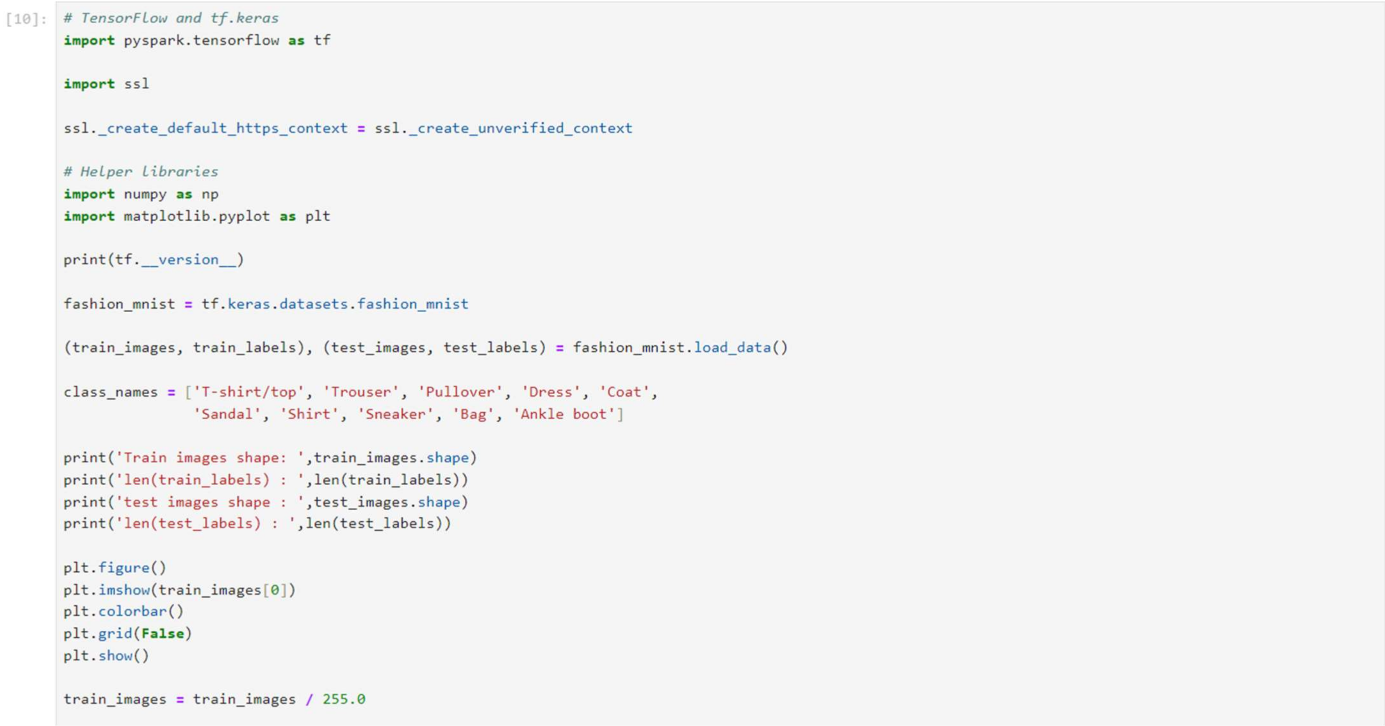


```
>>> df=pd.DataFrame({'col1':['a','b','c','hello',None],'col2':[1,2,3,4,5],'col3':['hey','1','2','3','4']})
>>> transform.strd_to_numd(df)
----------------------------------------------------------------------
col1  has null values.  Please replace the null values.
Do you want to the system to replace the null values for col1 and then encode the values[Y/n]?: y
------------------------------------------------
Enter the new value or press ENTER to assign the default value(missing): Null value
(  col1  col2 col3
0    1     1    4
1    2     2    0
2    3     3    1
3    4     4    2
4    0     5    3, {'col1': {'Null value': 0, 'a': 1, 'b': 2, 'c': 3, 'hello': 4}, 'col3': {'1': 0, '2': 1, '3': 2, '4': 3, 'hey': 4}})
>>> 
```

From the above testing results of the newly written methods in transform.py function we can say that we are able to replace the null values using pyspark. So we an say that the program is working as expected.

TensorFlow Testing:

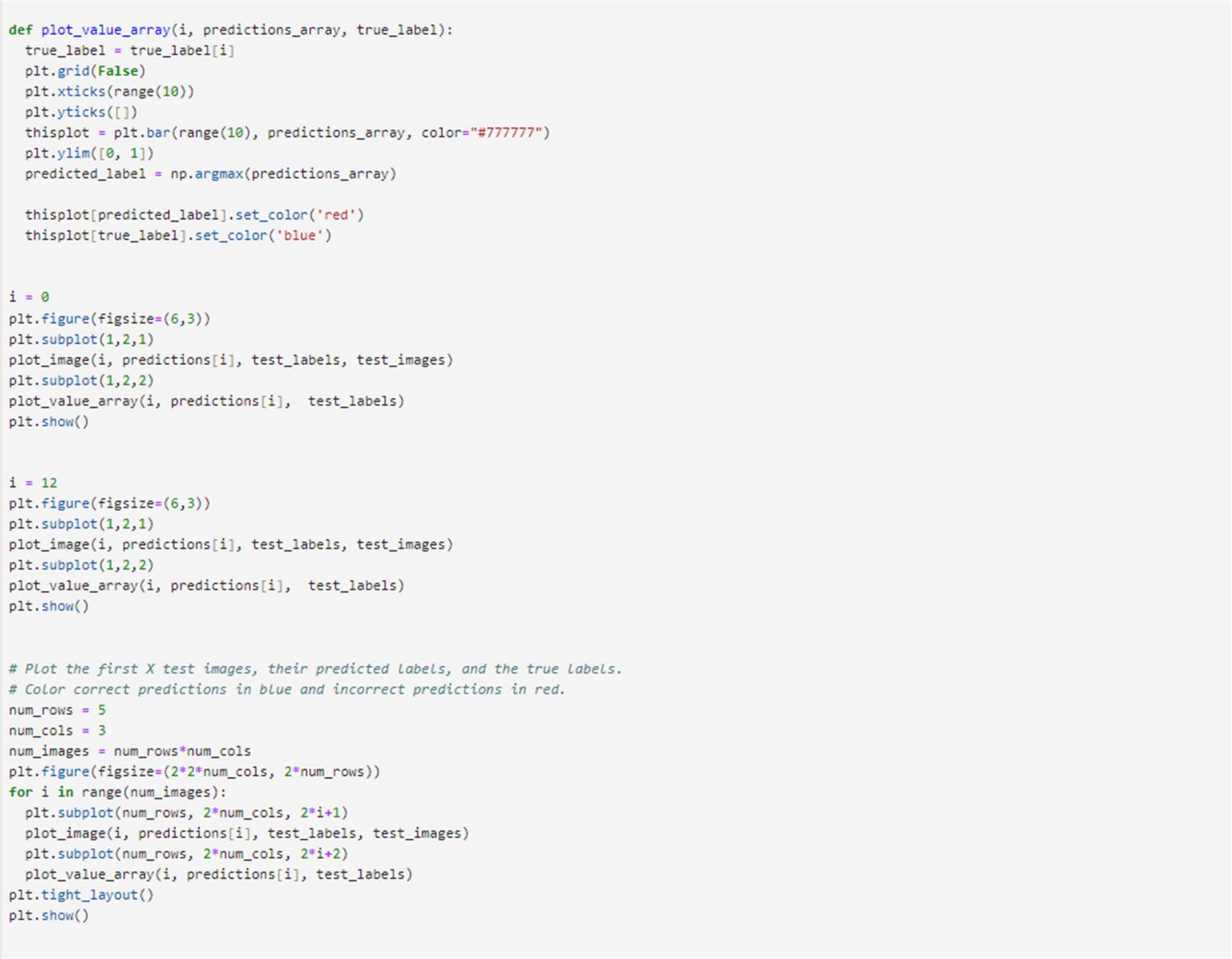
Tensorflow TestCase1:



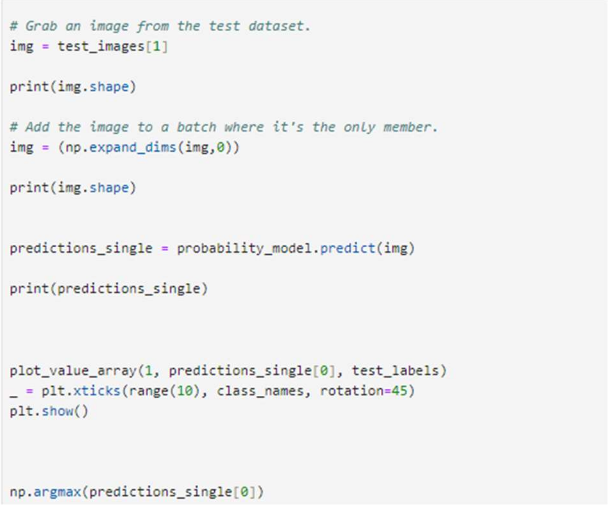
```
[10]: # TensorFlow and tf.keras
      import pyspark.tensorflow as tf

      import ssl

      ssl._create_default_https_context = ssl._create_unverified_context

      # Helper libraries
      import numpy as np
      import matplotlib.pyplot as plt

      print(tf.__version__)

      fashion_mnist = tf.keras.datasets.fashion_mnist

      (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

      class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                     'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

      print('Train images shape: ',train_images.shape)
      print('len(train_labels) : ',len(train_labels))
      print('test images shape : ',test_images.shape)
      print('len(test_labels) : ',len(test_labels))

      plt.figure()
      plt.imshow(train_images[0])
      plt.colorbar()
      plt.grid(False)
      plt.show()

      train_images = train_images / 255.0
```

```
test_images = test_images / 255.0

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])



model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])


model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)


probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

```
predictions = probability_model.predict(test_images)

print(predictions[0])


np.argmax(predictions[0])


test_labels[0]

def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)
```

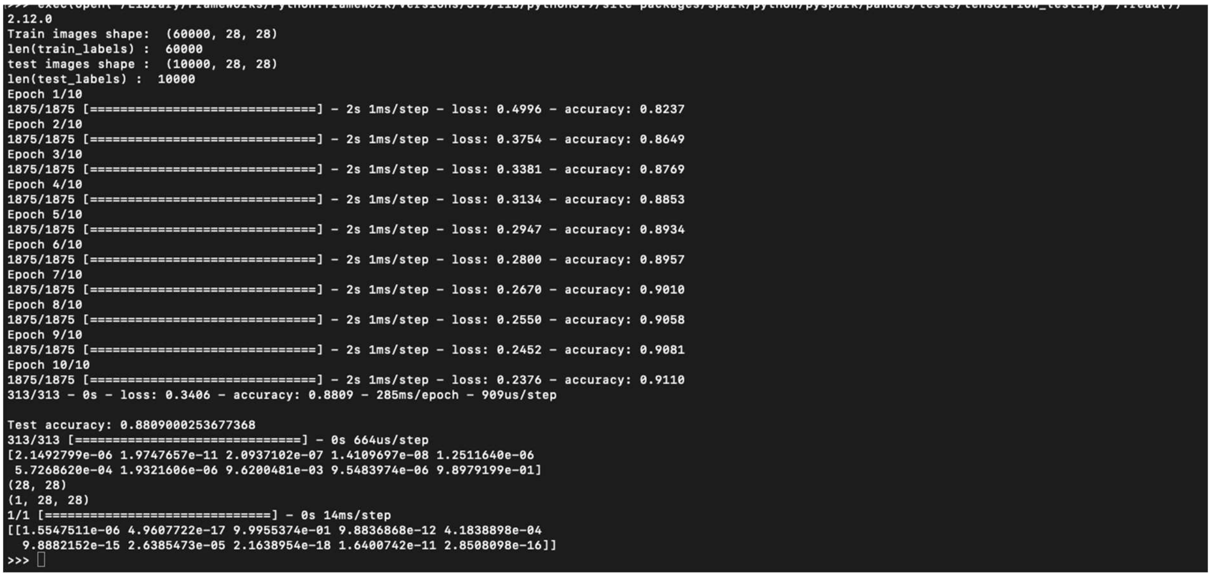


```
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')


i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()


i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()


# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

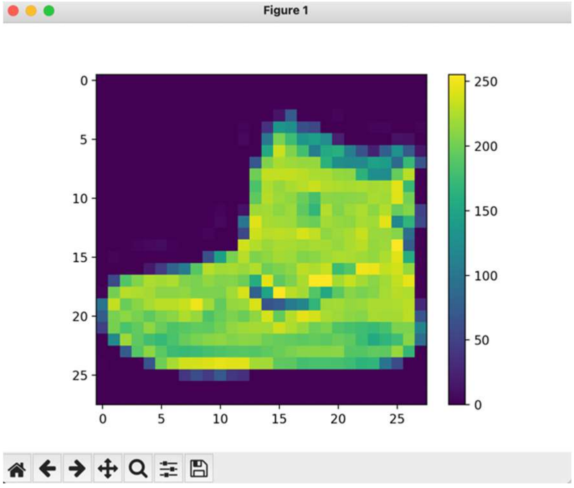


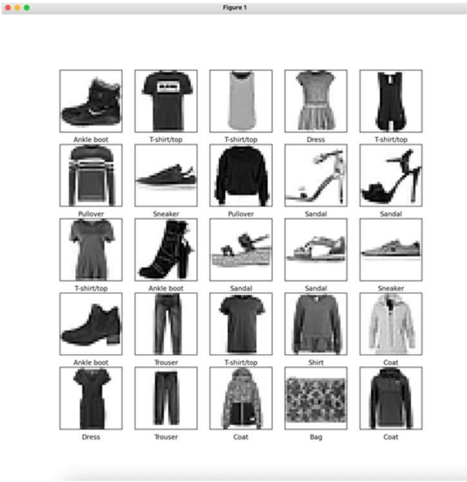
```
# Grab an image from the test dataset.
img = test_images[1]

print(img.shape)

# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))

print(img.shape)


predictions_single = probability_model.predict(img)

print(predictions_single)



plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
plt.show()



np.argmax(predictions_single[0])
```

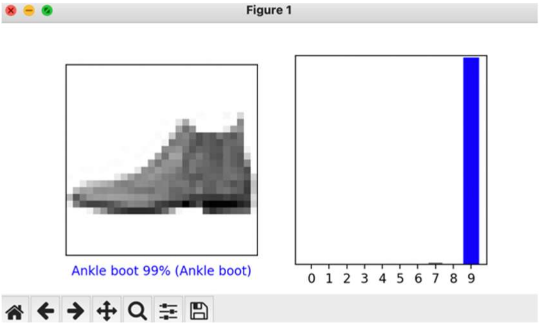
Output:

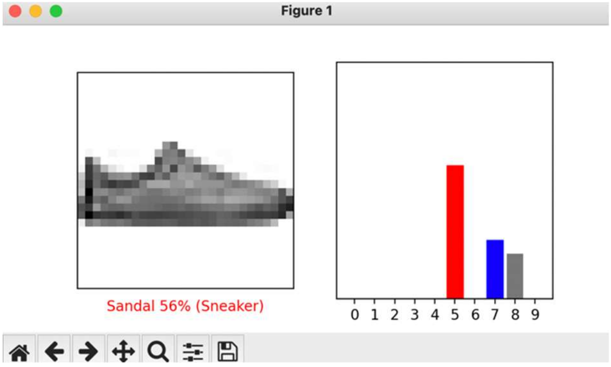


```
>>> exec(open('/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/spark/python/pyspark/pandas/tests/tensorflow_test1.py').read())
2.12.0
Train images shape:  (60000, 28, 28)
len(train_labels) :  60000
test images shape :  (10000, 28, 28)
len(test_labels) :  10000
Epoch 1/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.4996 - accuracy: 0.8237
Epoch 2/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3754 - accuracy: 0.8649
Epoch 3/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3381 - accuracy: 0.8769
Epoch 4/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3134 - accuracy: 0.8853
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2947 - accuracy: 0.8934
Epoch 6/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2800 - accuracy: 0.8957
Epoch 7/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2670 - accuracy: 0.9010
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2550 - accuracy: 0.9058
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2452 - accuracy: 0.9081
Epoch 10/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2376 - accuracy: 0.9110
313/313 - 0s - loss: 0.3406 - accuracy: 0.8809 - 285ms/epoch - 909us/step

Test accuracy: 0.8809000253677368
313/313 [==============================] - 0s 664us/step
[2.1492799e-06 1.9747657e-11 2.0937102e-07 1.4109697e-08 1.2511640e-06
 5.7268620e-04 1.9321606e-06 9.6200481e-03 9.5483974e-06 9.8979199e-01]
(28, 28)
(1, 28, 28)
1/1 [==============================] - 0s 14ms/step
[[1.5547511e-06 4.9607722e-17 9.9955374e-01 9.8836868e-12 4.1838898e-04
  9.8882152e-15 2.6385473e-05 2.1638954e-18 1.6400742e-11 2.8508098e-16]]
>>> 
```

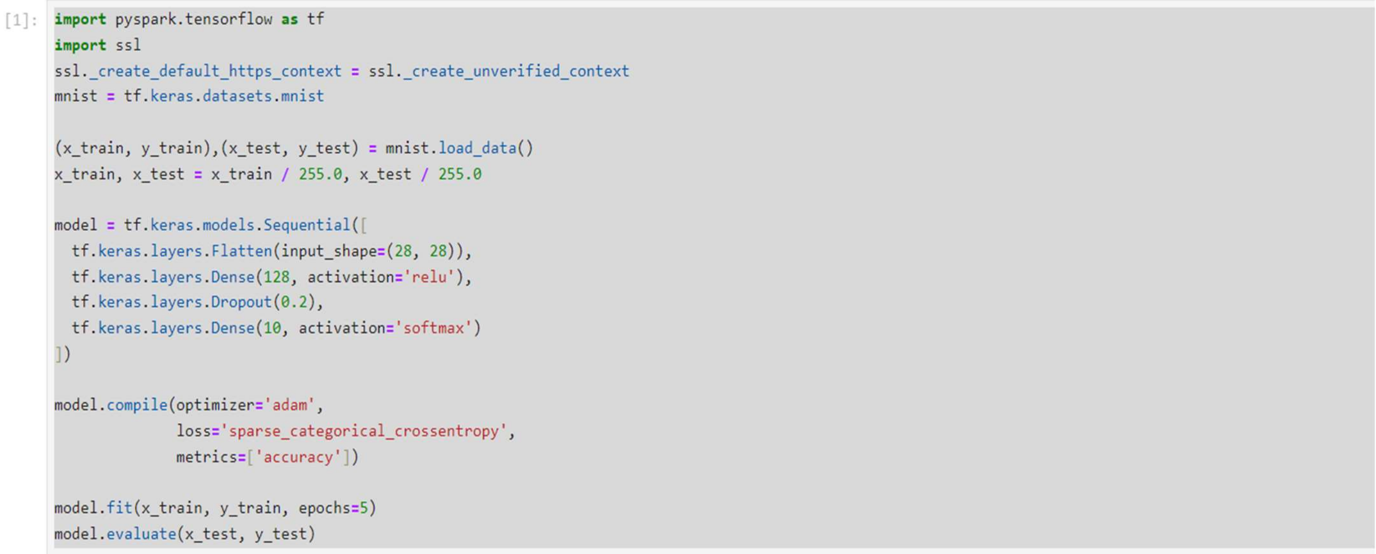






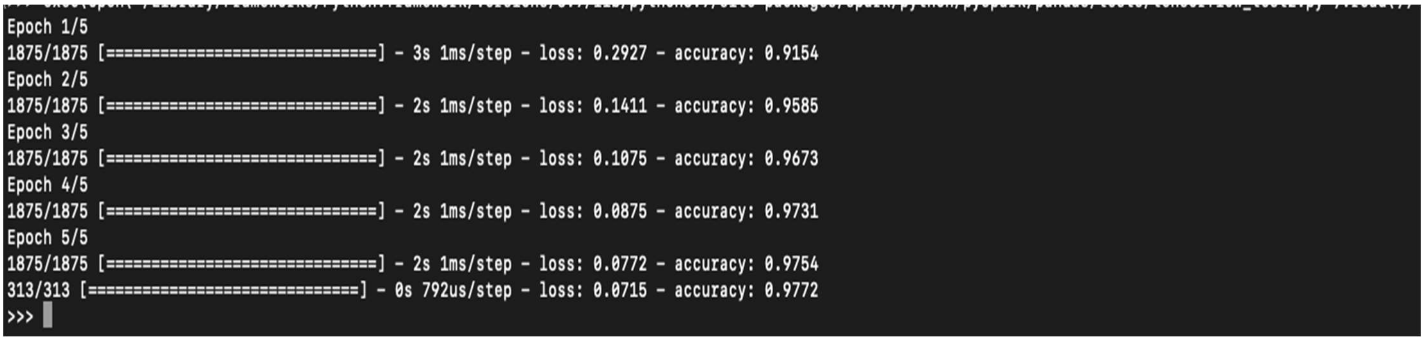


Testing for TensorFlow TestCase2:



```
[1]: import pyspark.tensorflow as tf
     import ssl
     ssl._create_default_https_context = ssl._create_unverified_context
     mnist = tf.keras.datasets.mnist

     (x_train, y_train),(x_test, y_test) = mnist.load_data()
     x_train, x_test = x_train / 255.0, x_test / 255.0

     model = tf.keras.models.Sequential([
       tf.keras.layers.Flatten(input_shape=(28, 28)),
       tf.keras.layers.Dense(128, activation='relu'),
       tf.keras.layers.Dropout(0.2),
       tf.keras.layers.Dense(10, activation='softmax')
     ])

     model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

     model.fit(x_train, y_train, epochs=5)
     model.evaluate(x_test, y_test)
```

Output:



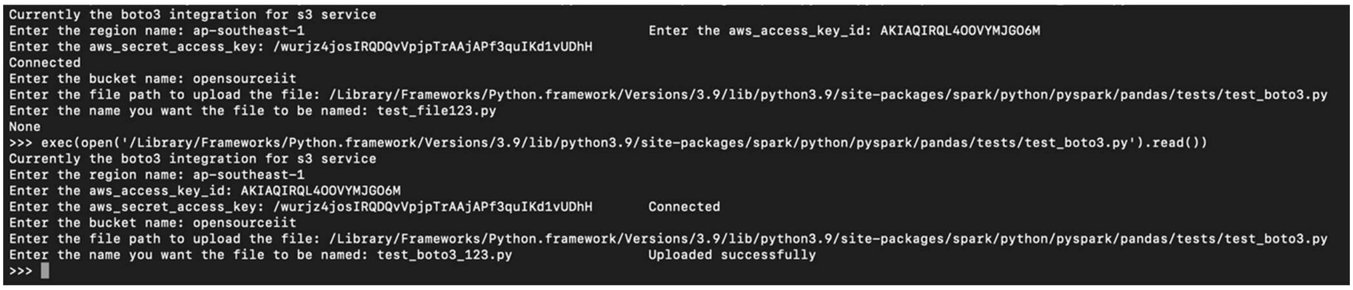
```
Epoch 1/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2927 - accuracy: 0.9154
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1411 - accuracy: 0.9585
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1075 - accuracy: 0.9673
Epoch 4/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0875 - accuracy: 0.9731
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0772 - accuracy: 0.9754
313/313 [==============================] - 0s 792us/step - loss: 0.0715 - accuracy: 0.9772
>>> 
```

Test for boto3:

#Code

```
from pyspark import boto3

print('Currently the boto3 integration for s3 service')

#taking the region anme, access keys from the user.

region_name=input('Enter the region name: ')

aws_access_key_id=input('Enter the aws_access_key_id: ')

aws_secret_access_key=input('Enter the aws_secret_access_key: ')

#intializing the connection as None

connection=None

try:

    #connecting the s3 service the given credentials to show the status as connected if the conneciton is made.

    s3_client = boto3.resource(service_name='s3', region_name=region_name,

                               aws_access_key_id=aws_access_key_id,
                               aws_secret_access_key=aws_secret_access_key)
    print('Connected')
    connection='Success'
#in exception case, it will produce the error to show what is wrong in the given credentials
except:
    s3_client = boto3.resource(service_name='s3', region_name=region_name,
aws_access_key_id=aws_access_key_id,               aws_secret_access_key=aws_secret_access_key)
#checking whether connection equal to 'Success'
if connection=='Success':
    #taking the bucket name, file path to upload and naming the file.
    bucket_name=input('Enter the bucket name: ')
    file_path=input('Enter the file path to upload the file: ')
    name_file=input('Enter the name you want the file to be named: ')
    result=s3_client.Bucket(bucket_name).upload_file(file_path,name_file)
    #if the result it None then the file got uploaded successfully
    if result==None:
      print('Uploaded successfully')
    #else there will be error produced.
```

Output:



```
Currently the boto3 integration for s3 service
Enter the region name: ap-southeast-1                                   Enter the aws_access_key_id: AKIAQIRQL4OOVYMJGO6M
Enter the aws_secret_access_key: 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
Connected
Enter the bucket name: opensourceiit
Enter the file path to upload the file: /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/spark/python/pyspark/pandas/tests/test_boto3.py
Enter the name you want the file to be named: test_file123.py
None
>>> exec(open('/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/spark/python/pyspark/pandas/tests/test_boto3.py').read())
Currently the boto3 integration for s3 service
Enter the region name: ap-southeast-1
Enter the aws_access_key_id: AKIAQIRQL4OOVYMJGO6M
Enter the aws_secret_access_key: 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       Connected
Enter the bucket name: opensourceiit
Enter the file path to upload the file: /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/spark/python/pyspark/pandas/tests/test_boto3.py
Enter the name you want the file to be named: test_boto3_123.py                 Uploaded successfully
>>> █
```

Testing framework

Cases

For new feature testing: Two test cases are done on the in-built datasets and different .csv files downloaded from the internet. Two scripts have been written for two other test cases.

For TensorFlow testing: Two test cases are done on the scripts from the official Tensorflow website.

Source code

Dependencies(open-source)

- Pandas, Numpy
- TensorFlow
- Boto3

Bibiliography

My github repository master branch: https://github.com/RamchandraReddy07/Extending-the-features-in-PySpark-Final-Project-

TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems: **https://arxiv.org/abs/1603.04467**

Horovod: fast and easy distributed deep learning in TensorFlow: https://arxiv.org/abs/1802.05799

Target encoding done the right way: https://maxhalford.github.io/blog/target-encoding/

Feature encoding techniques – Machine learning: https://www.geeksforgeeks.org/feature-encoding-techniques-machine-learning/

Guide to Encoding Categorical Values in Python: https://www.niit.com/india/knowledge-centre/python-categorical-values

Getting started with spark: https://spark.apache.org/

Pandas API on sparkhttps://spark.apache.org/docs/latest/api/python/getting_started/quickstart_ps.html

Spark GitHub repository: https://github.com/apache/spark.git

Boto3 GitHub repositoryhttps://github.com/boto/boto3

TensorFlow GitHub repository: https://github.com/tensorflow/tensorflow

Building spark and launch pyspark from the source code on the local machine: https://spark.apache.org/docs/latest/building-spark.html#building-submodules-individually

Guide to contributing to Open-Source projects: https://dev.to/codesphere/how-to-start-contributing-to-open-source-projects-on-github-534n

Integration with Cloud Infrastructures: https://spark.apache.org/docs/latest/cloud-integration.html