Gender Recognition using Open Images dataset V5

Background –

One of the basic fields of research is the identification of gender from face pictures. Automated gender recognition is critical in many areas of application such as interaction of human computers, biometrics, surveillance, demographic statistics, etc.

Goal–

Here, we are proposing this dataset to evaluate a model using deep learning techniques to detect human faces in images and then predict the image-based gender. To that end, the special pre -trained algorithm from source - https://github.com/arunponnusamy/gender-detection-keras will be trained on a dataset of men and women (Human Face) photos. The algorithm would predict a person's gender after training just by examining his / her face.

For exploratory data analysis, we will be examining on –

1. Total number of different categories with the highest number of labels available in them.
2. Count of Good images and Bad images of Human Face for Male and Female individually.

Data Source-

-This data source link - https://storage.googleapis.com/openimages/web/visualizer/index.html?set=train&type=detection&c=%2Fm%2F0dzct

This dataset contains around 36.5M images with 19,969 classes with three types- 'Segmentation', 'Detection' and 'Relationships' with multiple categories. For Gender recognition from face images, we will be using 'Detection' type with 'Human Face' category from this dataset for implementation.

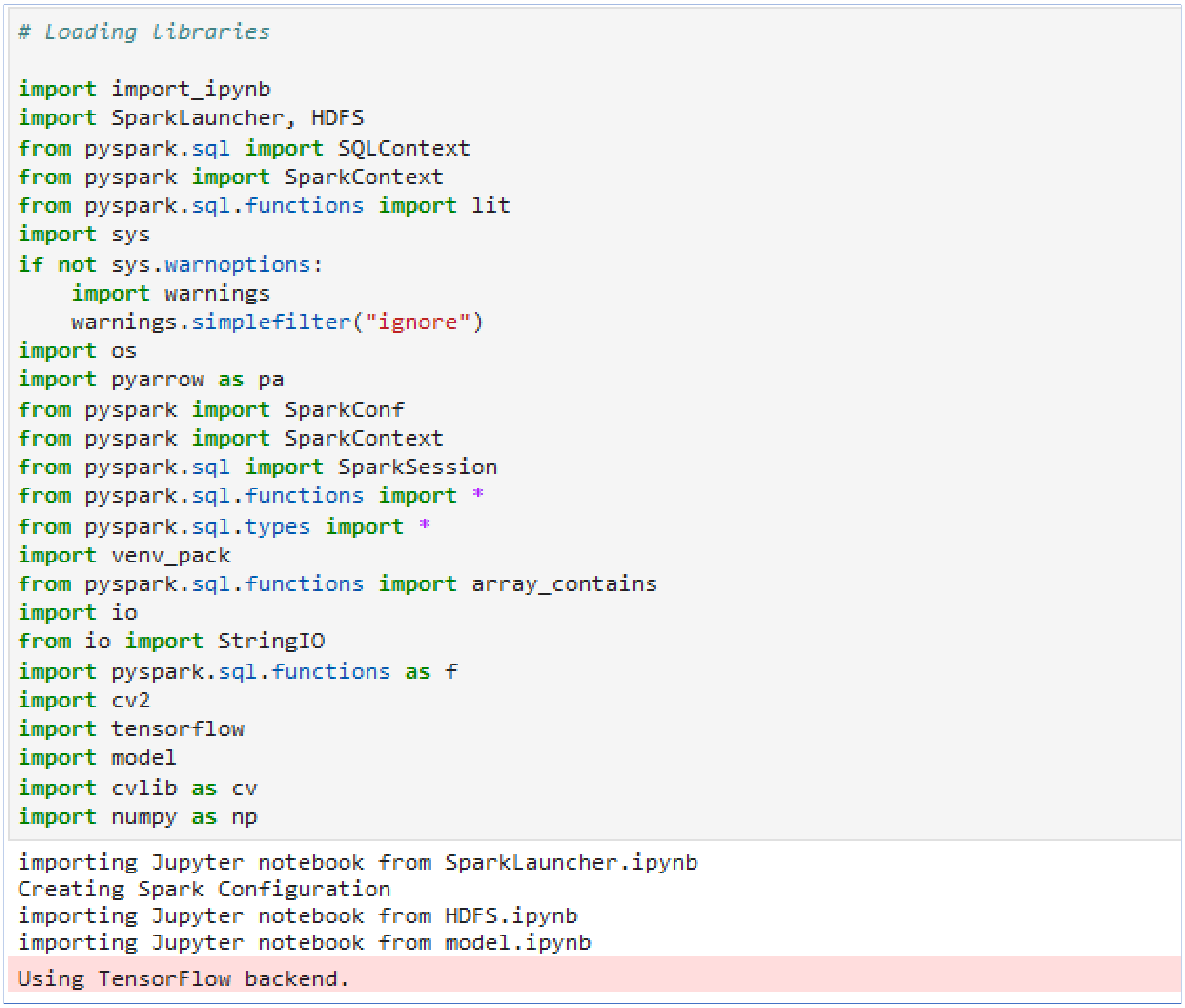
Execution-

For identify genders from face images, we will need to break this project into the following 5 parts;

- Part1 – It covers data extraction and data loading and data cleaning part
- Part2 – It covers data exploratory data analysis.
- Part3 – This part contains the Feature extraction.
- Part4 – This part contains model training.
- Part5 – This part carries out the evaluation and predictions.

Below are the Relevant Libraries we used to support our project and our spark session creation.

Relevant libraries to import –

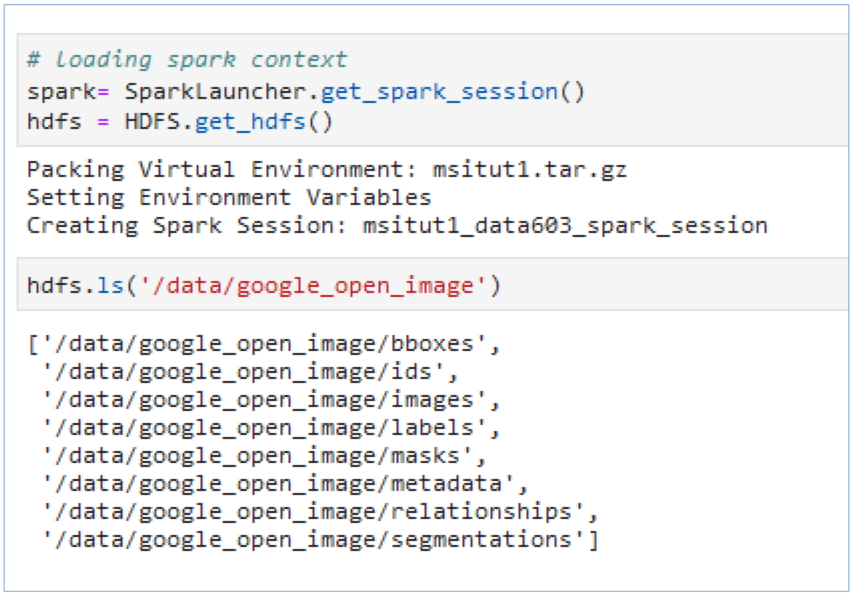


```
# Loading libraries

import import_ipynb
import SparkLauncher, HDFS
from pyspark.sql import SQLContext
from pyspark import SparkContext
from pyspark.sql.functions import lit
import sys
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
import os
import pyarrow as pa
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
import venv_pack
from pyspark.sql.functions import array_contains
import io
from io import StringIO
import pyspark.sql.functions as f
import cv2
import tensorflow
import model
import cvlib as cv
import numpy as np
```

```
importing Jupyter notebook from SparkLauncher.ipynb
Creating Spark Configuration
importing Jupyter notebook from HDFS.ipynb
importing Jupyter notebook from model.ipynb
Using TensorFlow backend.
```

Spark Session Creation-

We are creating spark session and hdfs connection using files – ‘SparkLauncher.ipynb’ and ‘HDFS.ipynb’ as shown below.



```
# Loading spark context
spark= SparkLauncher.get_spark_session()
hdfs = HDFS.get_hdfs()
```

```
Packing Virtual Environment: msitut1.tar.gz
Setting Environment Variables
Creating Spark Session: msitut1_data603_spark_session
```

```
hdfs.ls('/data/google_open_image')
```

```
['/data/google_open_image/bboxes',
 '/data/google_open_image/ids',
 '/data/google_open_image/images',
 '/data/google_open_image/labels',
 '/data/google_open_image/masks',
 '/data/google_open_image/metadata',
 '/data/google_open_image/relationships',
 '/data/google_open_image/segmentations']
```

<u>Part 1: Data Extraction and Data cleaning</u>

Here we used 4 data files downloaded from the data source into the hdfs location-

The first file we used was the '/data/google_open_image/metadata/class-descriptions-boxable.csv'. This file is used to retrieve the specific label names for description 'Human face', 'Woman', 'Man', 'Boy', 'Girl'

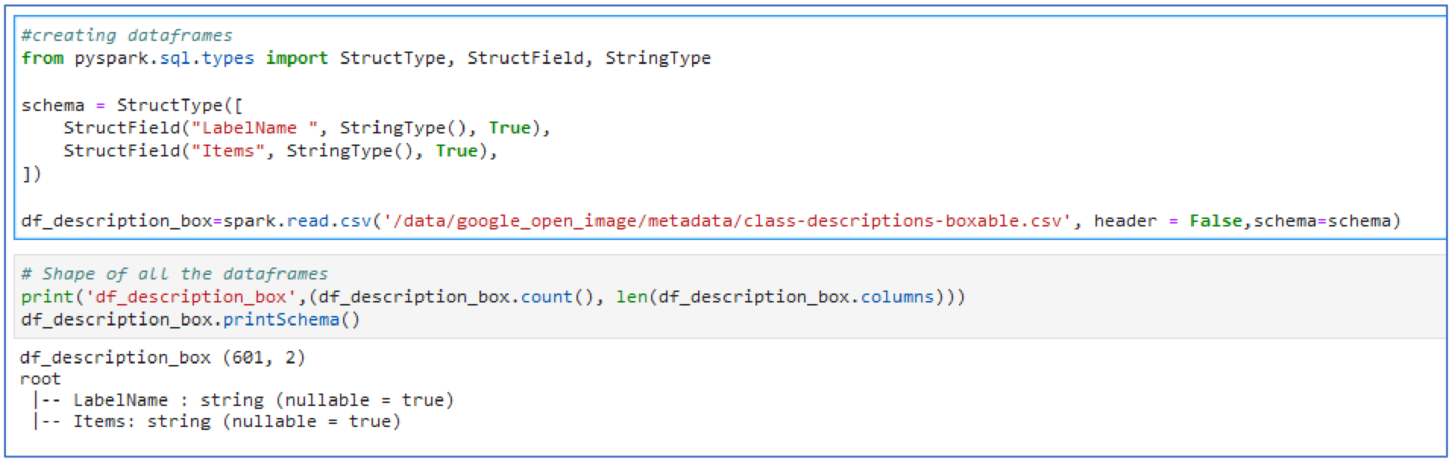
The following file"/data/google_open_image/labels/*.csv", was used to get the ImageID for specific label names mentioned above.

The third file, '/etl/google_open_image/images.parquet', file contains the data point of all the images ids from file 2 that we created from above.

The last files files used were('/data/google_open_image/bboxes/test-annotations-bbox.csv')
('/data/google_open_image/bboxes/train-annotations-bbox.csv')
('/data/google_open_image/bboxes/validation-annotations-bbox.csv'). These files were used to fetch the bounding boxes details for the ImageId mentioned in files 3.

<u>DataFrame creation and description-</u>

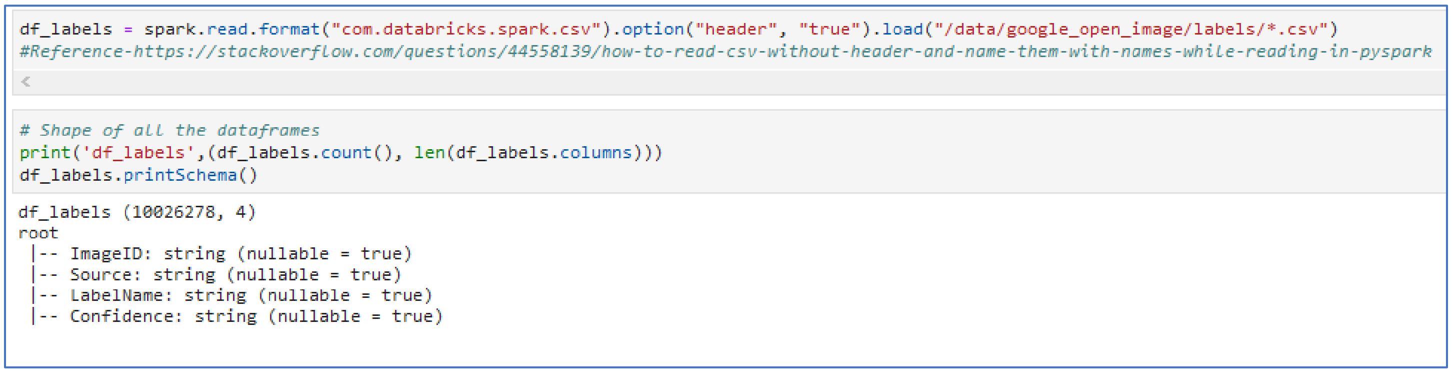
The below section details our DataFrame creation and description.



```
#creating dataframes
from pyspark.sql.types import StructType, StructField, StringType

schema = StructType([
    StructField("LabelName ", StringType(), True),
    StructField("Items", StringType(), True),
])

df_description_box=spark.read.csv('/data/google_open_image/metadata/class-descriptions-boxable.csv', header = False,schema=schema)
```

```
# Shape of all the dataframes
print('df_description_box',(df_description_box.count(), len(df_description_box.columns)))
df_description_box.printSchema()
```

```
df_description_box (601, 2)
root
 |-- LabelName : string (nullable = true)
 |-- Items: string (nullable = true)
```

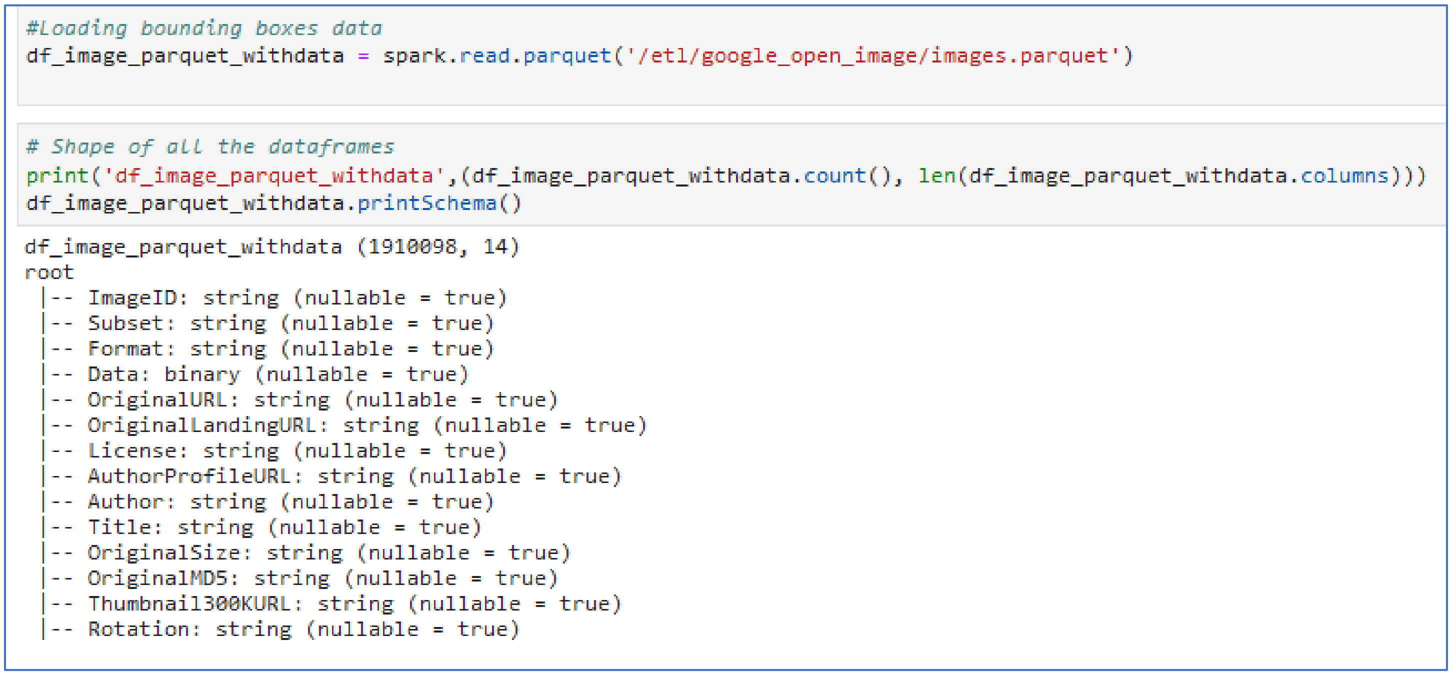


```
df_labels = spark.read.format("com.databricks.spark.csv").option("header", "true").load("/data/google_open_image/labels/*.csv")
#Reference-https://stackoverflow.com/questions/44558139/how-to-read-csv-without-header-and-name-them-with-names-while-reading-in-pyspark
```

```
# Shape of all the dataframes
print('df_labels',(df_labels.count(), len(df_labels.columns)))
df_labels.printSchema()
```

```
df_labels (10026278, 4)
root
 |-- ImageID: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- LabelName: string (nullable = true)
 |-- Confidence: string (nullable = true)
```

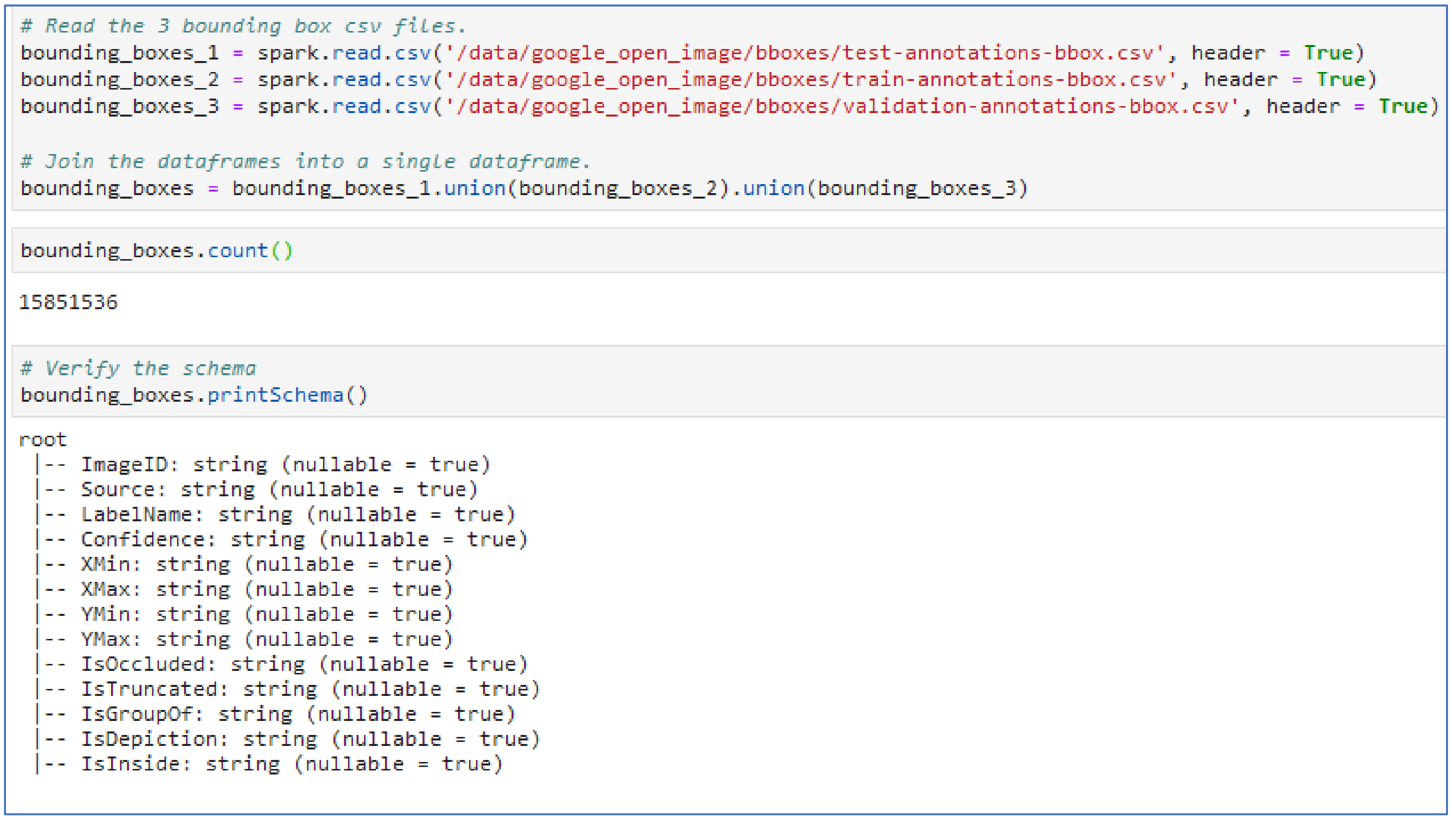
For this project 150 images IDs of each Male and Female category were used for performance reasons.



```
#Loading bounding boxes data
df_image_parquet_withdata = spark.read.parquet('/etl/google_open_image/images.parquet')
```

```
# Shape of all the dataframes
print('df_image_parquet_withdata',(df_image_parquet_withdata.count(), len(df_image_parquet_withdata.columns)))
df_image_parquet_withdata.printSchema()
```

```
df_image_parquet_withdata (1910098, 14)
root
 |-- ImageID: string (nullable = true)
 |-- Subset: string (nullable = true)
 |-- Format: string (nullable = true)
 |-- Data: binary (nullable = true)
 |-- OriginalURL: string (nullable = true)
 |-- OriginalLandingURL: string (nullable = true)
 |-- License: string (nullable = true)
 |-- AuthorProfileURL: string (nullable = true)
 |-- Author: string (nullable = true)
 |-- Title: string (nullable = true)
 |-- OriginalSize: string (nullable = true)
 |-- OriginalMD5: string (nullable = true)
 |-- Thumbnail300KURL: string (nullable = true)
 |-- Rotation: string (nullable = true)
```



```
# Read the 3 bounding box csv files.
bounding_boxes_1 = spark.read.csv('/data/google_open_image/bboxes/test-annotations-bbox.csv', header = True)
bounding_boxes_2 = spark.read.csv('/data/google_open_image/bboxes/train-annotations-bbox.csv', header = True)
bounding_boxes_3 = spark.read.csv('/data/google_open_image/bboxes/validation-annotations-bbox.csv', header = True)

# Join the dataframes into a single dataframe.
bounding_boxes = bounding_boxes_1.union(bounding_boxes_2).union(bounding_boxes_3)
```

```
bounding_boxes.count()
```

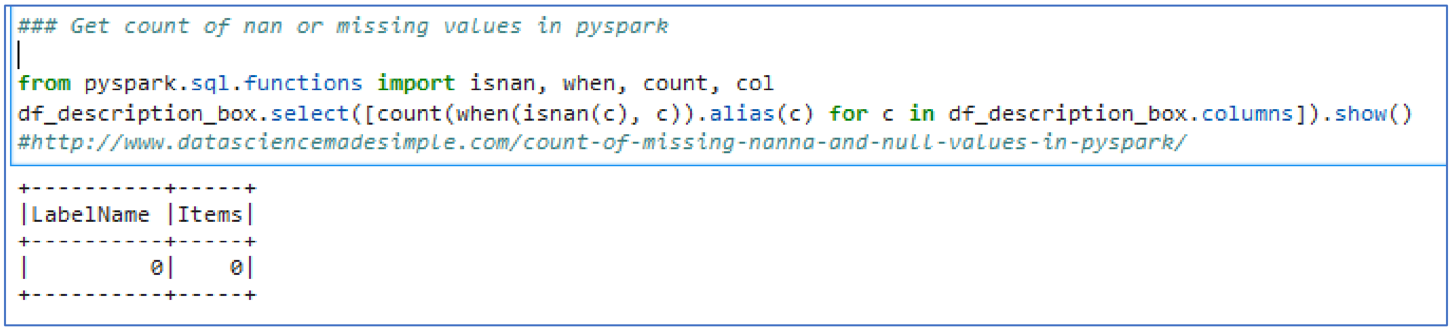
```
15851536
```

```
# Verify the schema
bounding_boxes.printSchema()
```

```
root
 |-- ImageID: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- LabelName: string (nullable = true)
 |-- Confidence: string (nullable = true)
 |-- XMin: string (nullable = true)
 |-- XMax: string (nullable = true)
 |-- YMin: string (nullable = true)
 |-- YMax: string (nullable = true)
 |-- IsOccluded: string (nullable = true)
 |-- IsTruncated: string (nullable = true)
 |-- IsGroupOf: string (nullable = true)
 |-- IsDepiction: string (nullable = true)
 |-- IsInside: string (nullable = true)
```

It is extremely important to make sure all data frames used in this project do NOT contain Null Values.

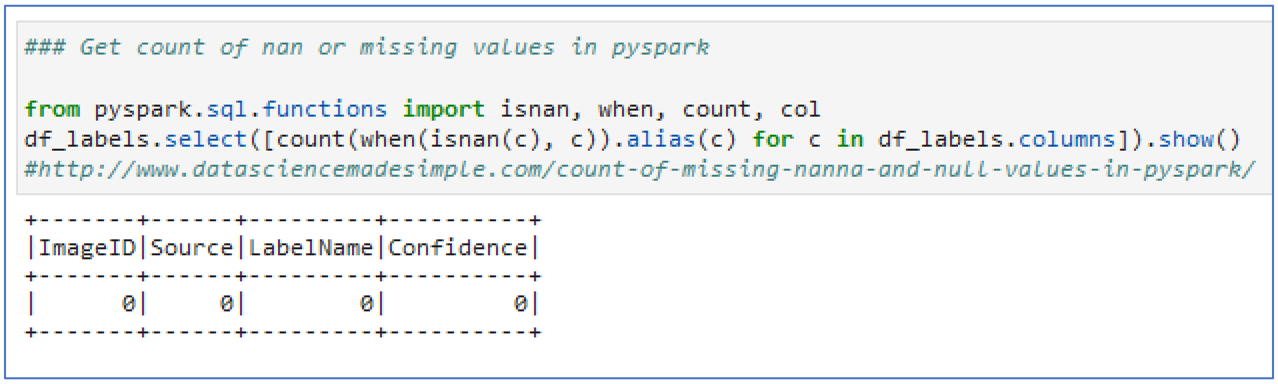
df_description_box-



```
### Get count of nan or missing values in pyspark

from pyspark.sql.functions import isnan, when, count, col
df_description_box.select([count(when(isnan(c), c)).alias(c) for c in df_description_box.columns]).show()
#http://www.datasciencemadesimple.com/count-of-missing-nanna-and-null-values-in-pyspark/
```

```
+----------+-----+
|LabelName |Items|
+----------+-----+
|         0|    0|
+----------+-----+
```

df_labels -



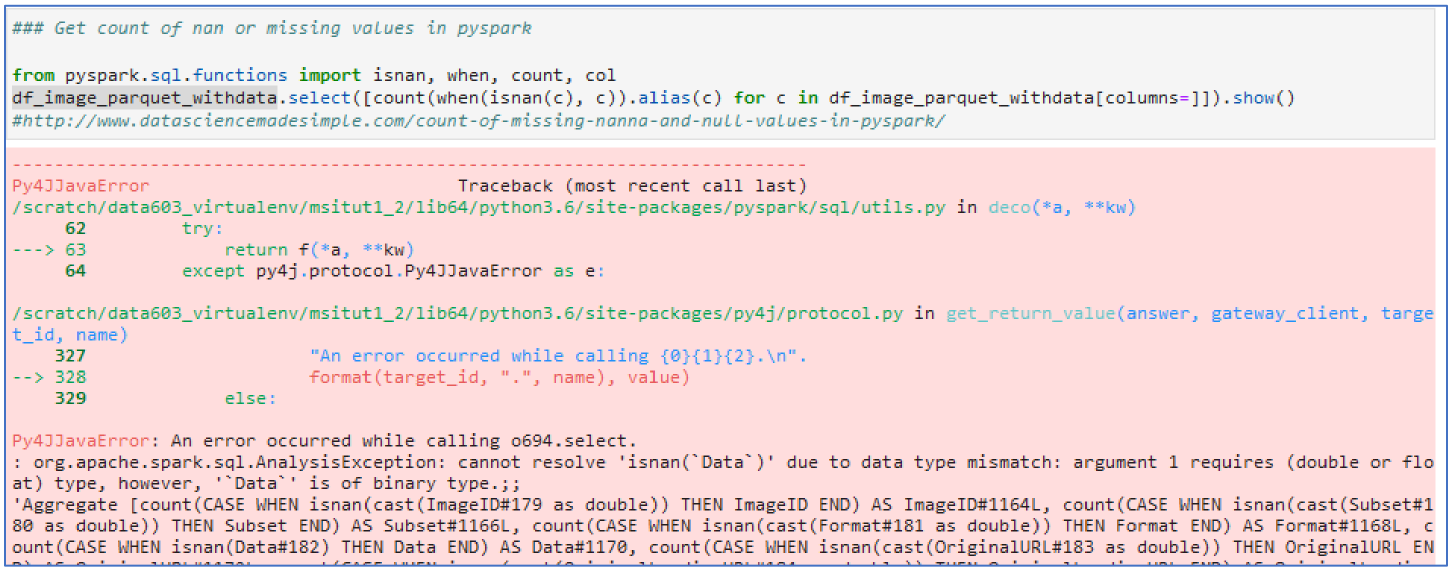
```
### Get count of nan or missing values in pyspark

from pyspark.sql.functions import isnan, when, count, col
df_labels.select([count(when(isnan(c), c)).alias(c) for c in df_labels.columns]).show()
#http://www.datasciencemadesimple.com/count-of-missing-nanna-and-null-values-in-pyspark/
```

```
+-------+------+---------+----------+
|ImageID|Source|LabelName|Confidence|
+-------+------+---------+----------+
|      0|     0|        0|         0|
+-------+------+---------+----------+
```

<u>df_image_parquet_withdata –</u>

Issue Faced

While determining if our dataframe, df_image_parquet_withdata had any Nulls, we ran into quite some trouble since our data column had a byte type array which is not compatible with isnan() function. As illustrated below.

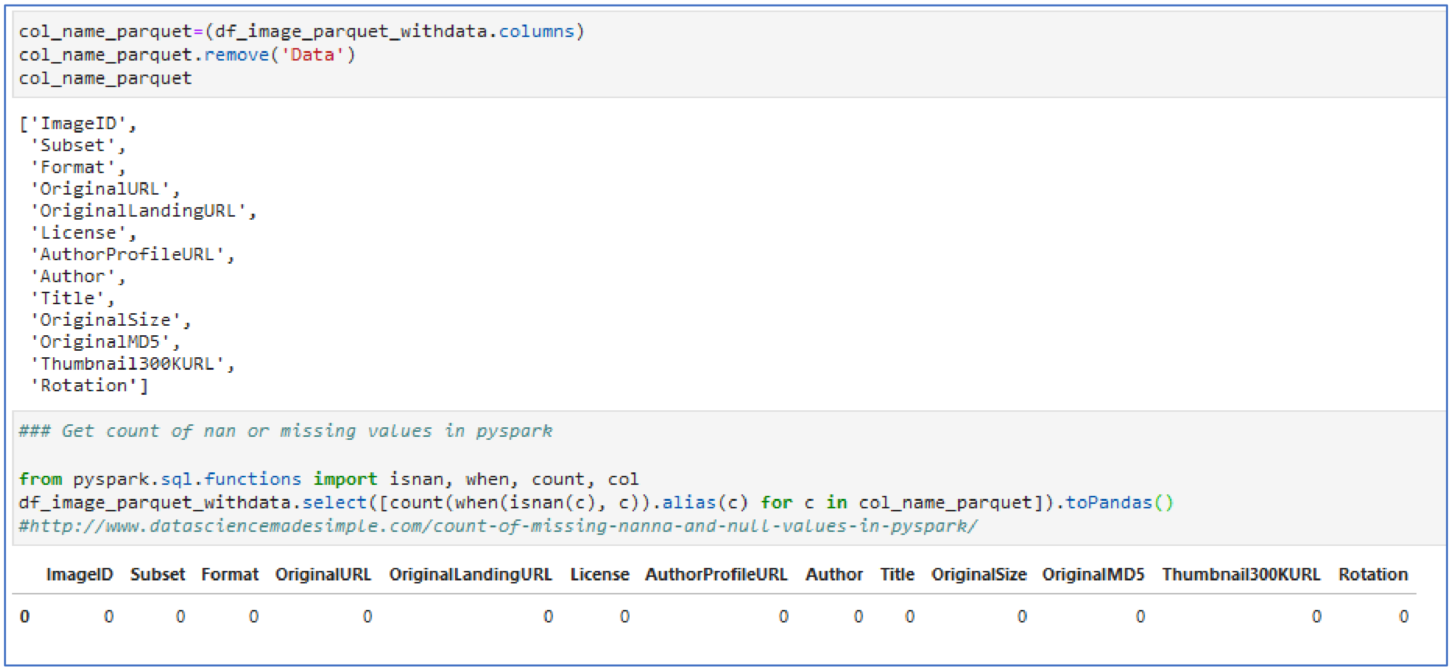


```
### Get count of nan or missing values in pyspark

from pyspark.sql.functions import isnan, when, count, col
df_image_parquet_withdata.select([count(when(isnan(c), c)).alias(c) for c in df_image_parquet_withdata[columns=]]).show()
#http://www.datasciencemadesimple.com/count-of-missing-nanna-and-null-values-in-pyspark/
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
/scratch/data603_virtualenv/msitut1_2/lib64/python3.6/site-packages/pyspark/sql/utils.py in deco(*a, **kw)
     62         try:
---> 63             return f(*a, **kw)
     64         except py4j.protocol.Py4JJavaError as e:

/scratch/data603_virtualenv/msitut1_2/lib64/python3.6/site-packages/py4j/protocol.py in get_return_value(answer, gateway_client, targe
t_id, name)
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:

Py4JJavaError: An error occurred while calling o694.select.
: org.apache.spark.sql.AnalysisException: cannot resolve 'isnan(`Data`)' due to data type mismatch: argument 1 requires (double or flo
at) type, however, '`Data`' is of binary type.;;
'Aggregate [count(CASE WHEN isnan(cast(ImageID#179 as double)) THEN ImageID END) AS ImageID#1164L, count(CASE WHEN isnan(cast(Subset#1
80 as double)) THEN Subset END) AS Subset#1166L, count(CASE WHEN isnan(cast(Format#181 as double)) THEN Format END) AS Format#1168L, c
ount(CASE WHEN isnan(Data#182) THEN Data END) AS Data#1170, count(CASE WHEN isnan(cast(OriginalURL#183 as double)) THEN OriginalURL EN
```

In order to fix this issue, we created a list of dataframe column names and removed the ‘Data’ column from that and gave those specific columns to isnan() method to check for null values.

We are considering this workaround assuming that the ‘Data’ column must have values, and which can be found later on when we carry out chip_values using UDF in part3. **If ‘data’ column has any null value, then the UDF function will not work successfully in part 3.**



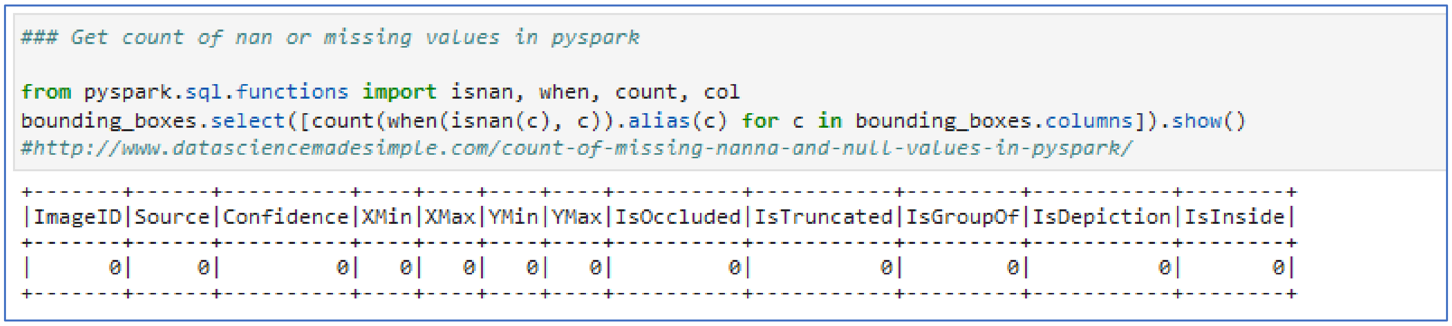
```
col_name_parquet=(df_image_parquet_withdata.columns)
col_name_parquet.remove('Data')
col_name_parquet
```

```
['ImageID',
 'Subset',
 'Format',
 'OriginalURL',
 'OriginalLandingURL',
 'License',
 'AuthorProfileURL',
 'Author',
 'Title',
 'OriginalSize',
 'OriginalMD5',
 'Thumbnail300KURL',
 'Rotation']
```

```
### Get count of nan or missing values in pyspark

from pyspark.sql.functions import isnan, when, count, col
df_image_parquet_withdata.select([count(when(isnan(c), c)).alias(c) for c in col_name_parquet]).toPandas()
#http://www.datasciencemadesimple.com/count-of-missing-nanna-and-null-values-in-pyspark/
```

	ImageID	Subset	Format	OriginalURL	OriginalLandingURL	License	AuthorProfileURL	Author	Title	OriginalSize	OriginalMD5	Thumbnail300KURL	Rotation
0	0	0	0	0	0	0	0	0	0	0	0	0	0

bounding_boxes –

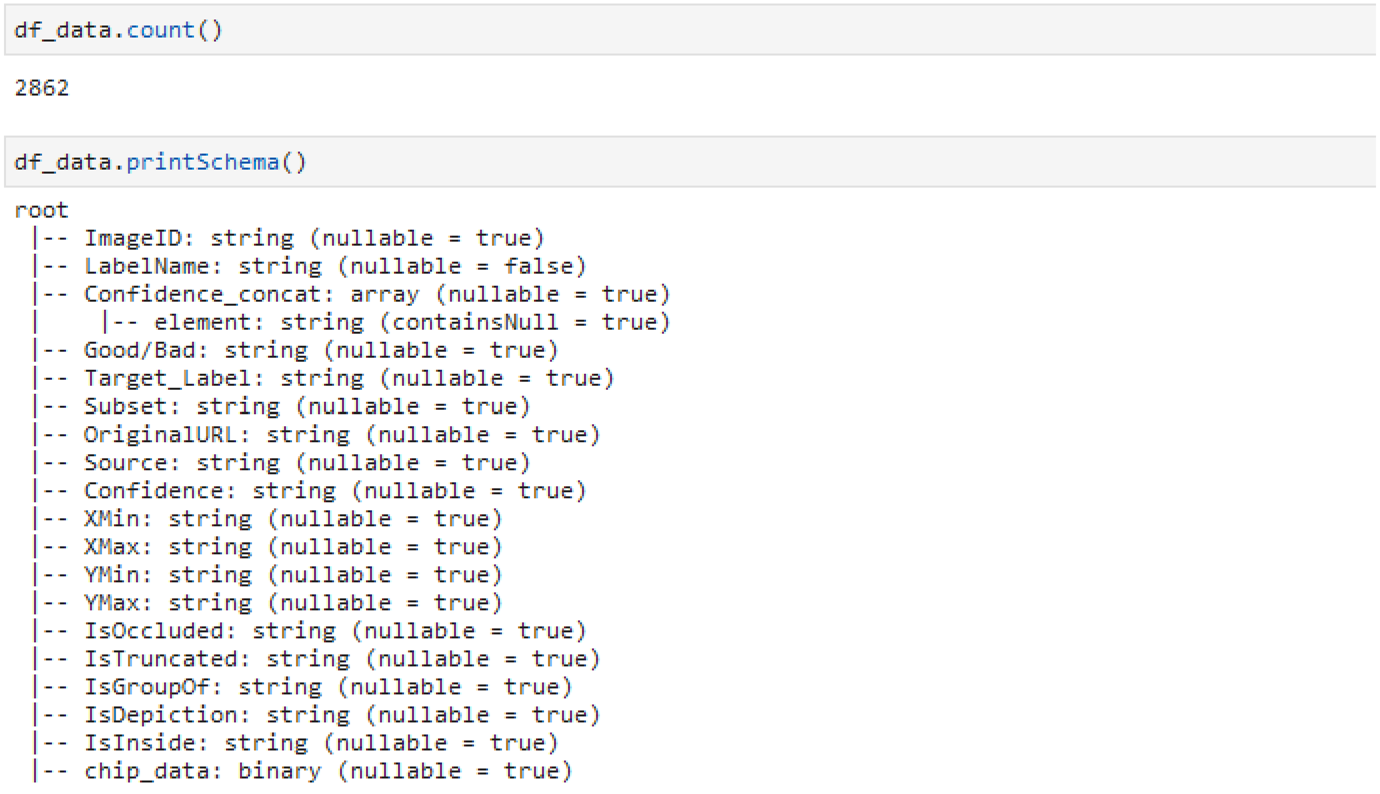


```
### Get count of nan or missing values in pyspark

from pyspark.sql.functions import isnan, when, count, col
bounding_boxes.select([count(when(isnan(c), c)).alias(c) for c in bounding_boxes.columns]).show()
#http://www.datasciencemadesimple.com/count-of-missing-nanna-and-null-values-in-pyspark/
```

```
+-------+------+----------+----+----+----+----+----------+-----------+---------+-----------+--------+
|ImageID|Source|Confidence|XMin|XMax|YMin|YMax|IsOccluded|IsTruncated|IsGroupOf|IsDepiction|IsInside|
+-------+------+----------+----+----+----+----+----------+-----------+---------+-----------+--------+
|      0|     0|         0|   0|   0|   0|   0|         0|          0|        0|          0|       0|
+-------+------+----------+----+----+----+----+----------+-----------+---------+-----------+--------+
```

As illustrated above, there are no null/nan values observed in all the dataframes.

After carryout data for specific label, our data frame looks like-

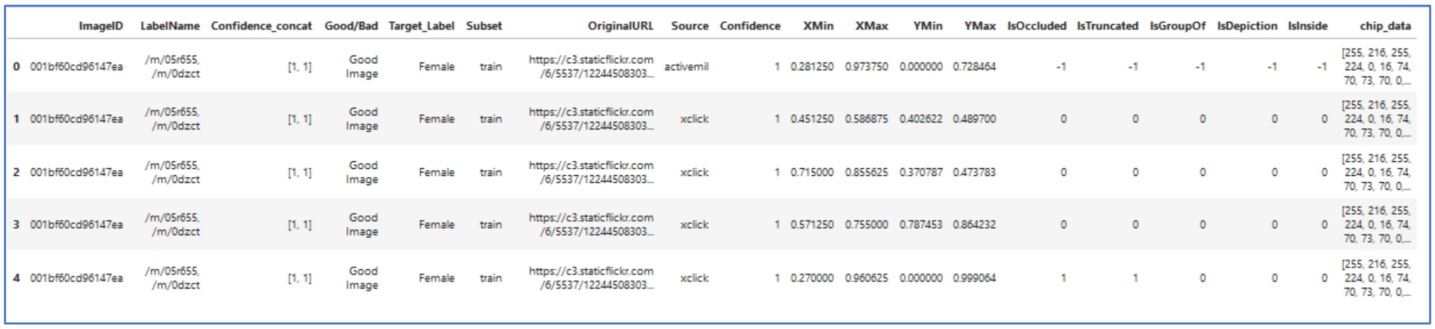


```
df_data.count()
```

```
2862
```

```
df_data.printSchema()
```

```
root
 |-- ImageID: string (nullable = true)
 |-- LabelName: string (nullable = false)
 |-- Confidence_concat: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- Good/Bad: string (nullable = true)
 |-- Target_Label: string (nullable = true)
 |-- Subset: string (nullable = true)
 |-- OriginalURL: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Confidence: string (nullable = true)
 |-- XMin: string (nullable = true)
 |-- XMax: string (nullable = true)
 |-- YMin: string (nullable = true)
 |-- YMax: string (nullable = true)
 |-- IsOccluded: string (nullable = true)
 |-- IsTruncated: string (nullable = true)
 |-- IsGroupOf: string (nullable = true)
 |-- IsDepiction: string (nullable = true)
 |-- IsInside: string (nullable = true)
 |-- chip_data: binary (nullable = true)
```



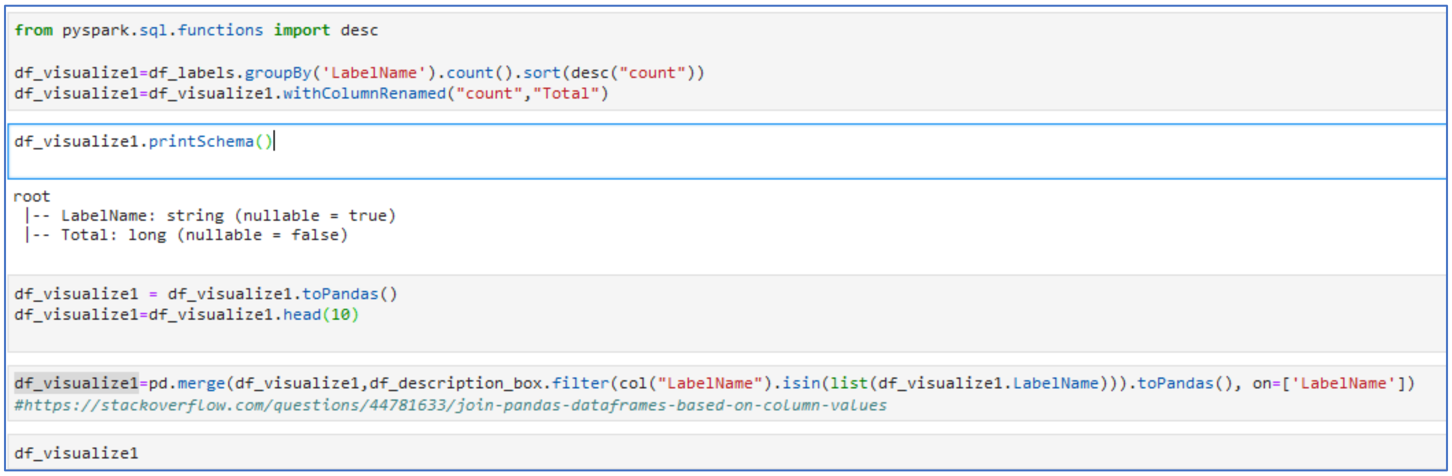
	ImageID	LabelName	Confidence_concat	Good/Bad	Target_Label	Subset	OriginalURL	Source	Confidence	XMin	XMax	YMin	YMax	IsOccluded	IsTruncated	IsGroupOf	IsDepiction	IsInside	chip_data
0	001bf60cd96147ea	/m/05r655, /m/0dzct	[1, 1]	Good Image	Female	train	https://c3.staticflickr.com /6/5537/12244508303...	activemil	1	0.281250	0.973750	0.000000	0.728464	-1	-1	-1	-1	-1	[255, 216, 255, 224, 0, 16, 74, 70, 73, 70, 0,...
1	001bf60cd96147ea	/m/05r655, /m/0dzct	[1, 1]	Good Image	Female	train	https://c3.staticflickr.com /6/5537/12244508303...	xclick	1	0.451250	0.586875	0.402622	0.489700	0	0	0	0	0	[255, 216, 255, 224, 0, 16, 74, 70, 73, 70, 0,...
2	001bf60cd96147ea	/m/05r655, /m/0dzct	[1, 1]	Good Image	Female	train	https://c3.staticflickr.com /6/5537/12244508303...	xclick	1	0.715000	0.855625	0.370787	0.473783	0	0	0	0	0	[255, 216, 255, 224, 0, 16, 74, 70, 73, 70, 0,...
3	001bf60cd96147ea	/m/05r655, /m/0dzct	[1, 1]	Good Image	Female	train	https://c3.staticflickr.com /6/5537/12244508303...	xclick	1	0.571250	0.755000	0.787453	0.864232	0	0	0	0	0	[255, 216, 255, 224, 0, 16, 74, 70, 73, 70, 0,...
4	001bf60cd96147ea	/m/05r655, /m/0dzct	[1, 1]	Good Image	Female	train	https://c3.staticflickr.com /6/5537/12244508303...	xclick	1	0.270000	0.960625	0.000000	0.999064	1	1	0	0	0	[255, 216, 255, 224, 0, 16, 74, 70, 73, 70, 0,...

This data is saved as parquet file to use further in Part 3 in feature extraction

<u>Exploratory data analysis-</u>

For EDA, we visualized 2 analysis –

First, Top 10 different labels with highest number of Images available in them. For that we carried out the data required from df_labels and df_description box and turned it into pandas dataframe which is used to visualize data using matplotlib library.



```
from pyspark.sql.functions import desc

df_visualize1=df_labels.groupBy('LabelName').count().sort(desc("count"))
df_visualize1=df_visualize1.withColumnRenamed("count","Total")
```

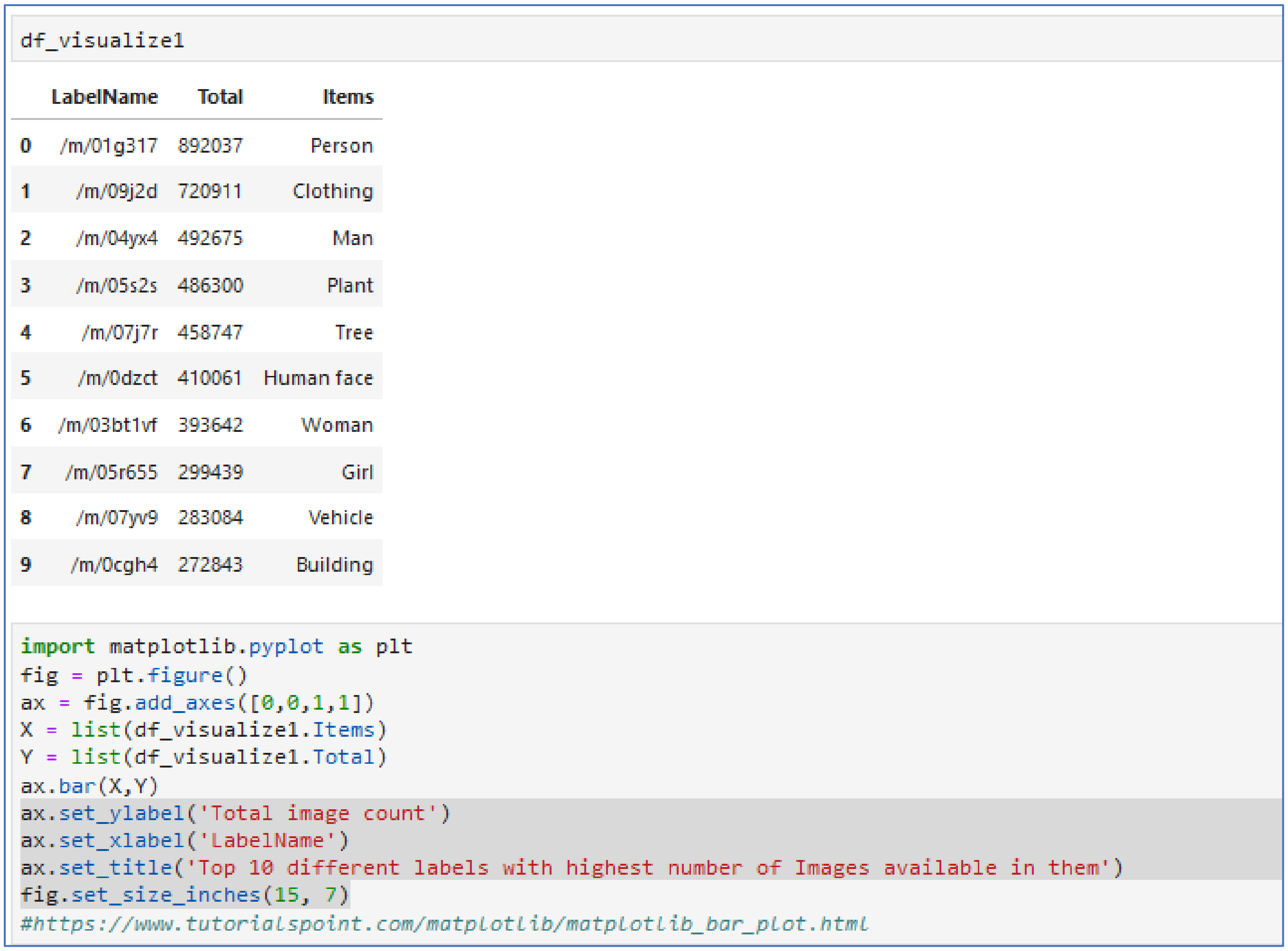
```
df_visualize1.printSchema()
```

```
root
 |-- LabelName: string (nullable = true)
 |-- Total: long (nullable = false)
```

```
df_visualize1 = df_visualize1.toPandas()
df_visualize1=df_visualize1.head(10)
```

```
df_visualize1=pd.merge(df_visualize1,df_description_box.filter(col("LabelName").isin(list(df_visualize1.LabelName))).toPandas(), on=['LabelName'])
#https://stackoverflow.com/questions/44781633/join-pandas-dataframes-based-on-column-values
```

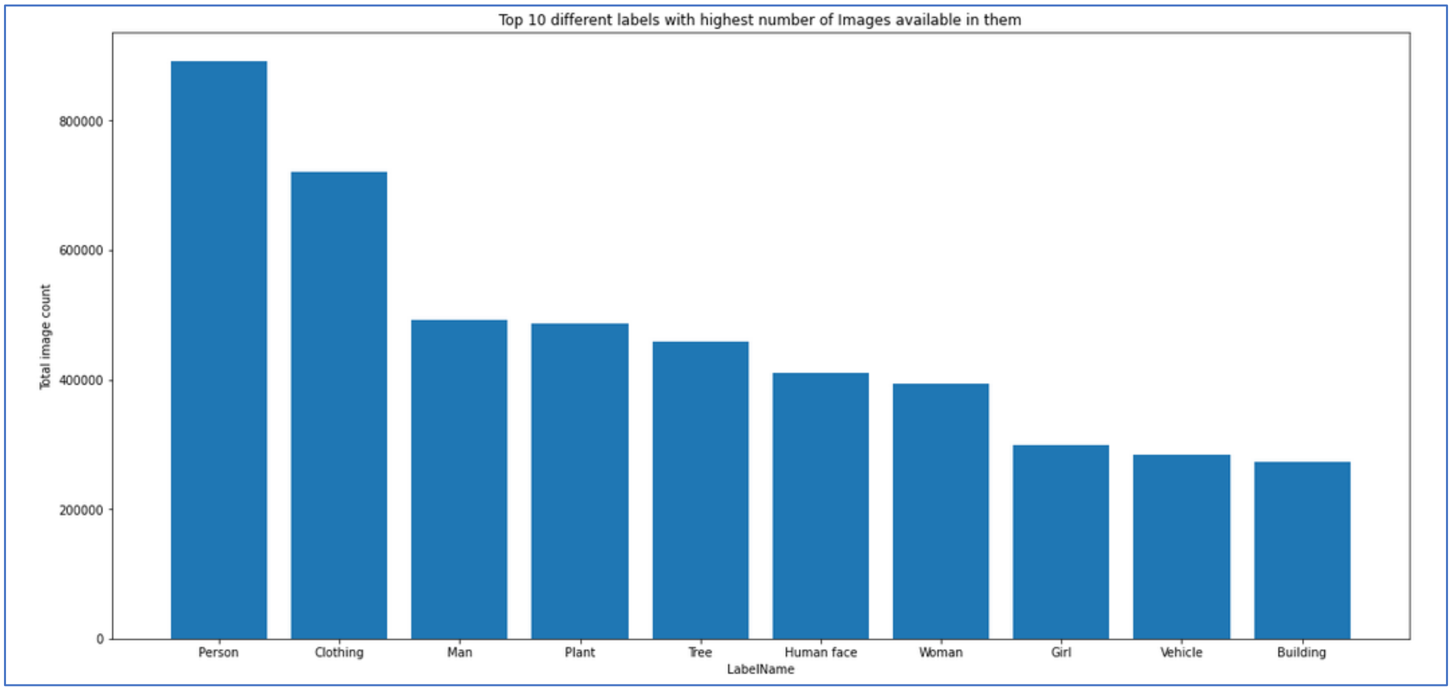
```
df_visualize1
```



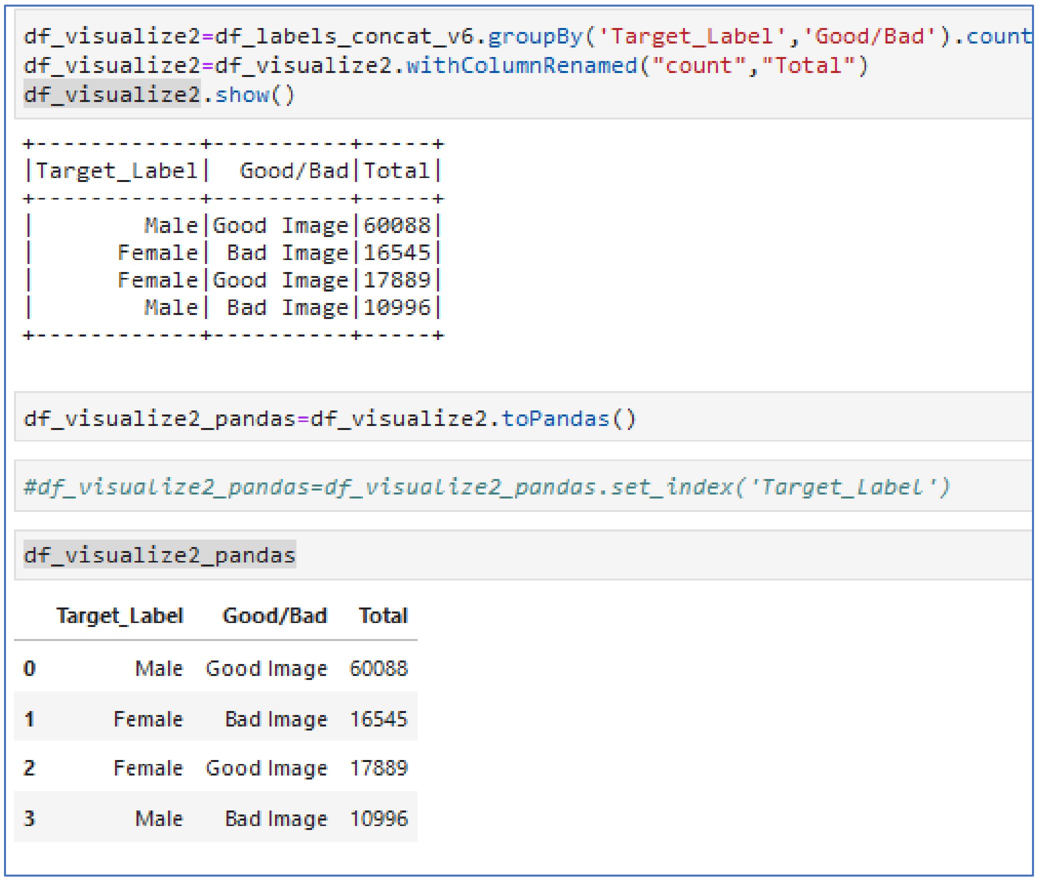
```
df_visualize1
```

	LabelName	Total	Items
0	/m/01g317	892037	Person
1	/m/09j2d	720911	Clothing
2	/m/04yx4	492675	Man
3	/m/05s2s	486300	Plant
4	/m/07j7r	458747	Tree
5	/m/0dzct	410061	Human face
6	/m/03bt1vf	393642	Woman
7	/m/05r655	299439	Girl
8	/m/07yv9	283084	Vehicle
9	/m/0cgh4	272843	Building

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
X = list(df_visualize1.Items)
Y = list(df_visualize1.Total)
ax.bar(X,Y)
ax.set_ylabel('Total image count')
ax.set_xlabel('LabelName')
ax.set_title('Top 10 different labels with highest number of Images available in them')
fig.set_size_inches(15, 7)
#https://www.tutorialspoint.com/matplotlib/matplotlib_bar_plot.html
```



Second analysis was for the how many male and female images are Good images and Bad images.



```
df_visualize2=df_labels_concat_v6.groupBy('Target_Label','Good/Bad').count
df_visualize2=df_visualize2.withColumnRenamed("count","Total")
df_visualize2.show()
```

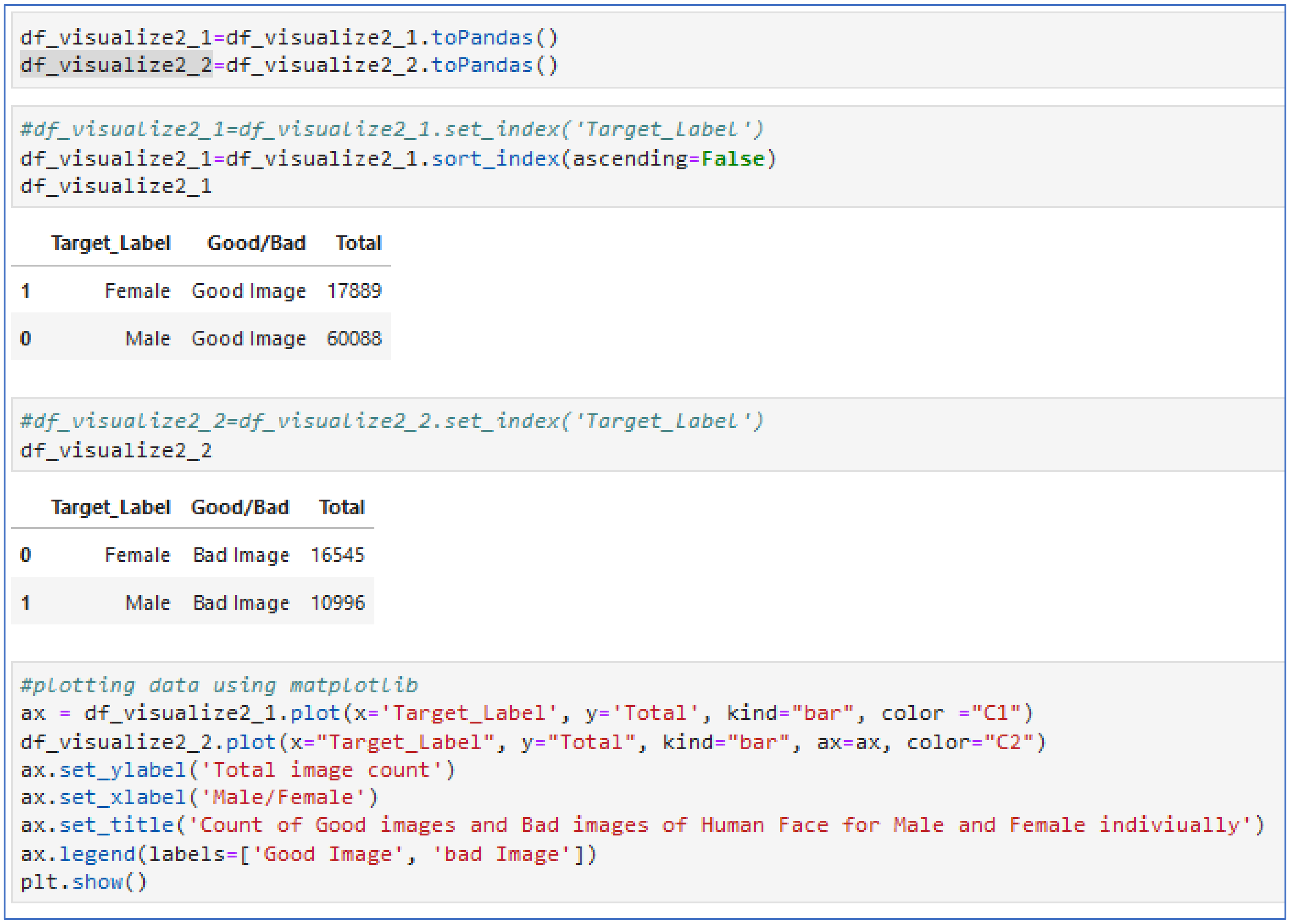
```
+------------+----------+-----+
|Target_Label|  Good/Bad|Total|
+------------+----------+-----+
|        Male|Good Image|60088|
|      Female| Bad Image|16545|
|      Female|Good Image|17889|
|        Male| Bad Image|10996|
+------------+----------+-----+
```

```
df_visualize2_pandas=df_visualize2.toPandas()
```

```
#df_visualize2_pandas=df_visualize2_pandas.set_index('Target_Label')
```

```
df_visualize2_pandas
```

	Target_Label	Good/Bad	Total
0	Male	Good Image	60088
1	Female	Bad Image	16545
2	Female	Good Image	17889
3	Male	Bad Image	10996



```python
df_visualize2_1=df_visualize2_1.toPandas()
df_visualize2_2=df_visualize2_2.toPandas()
```

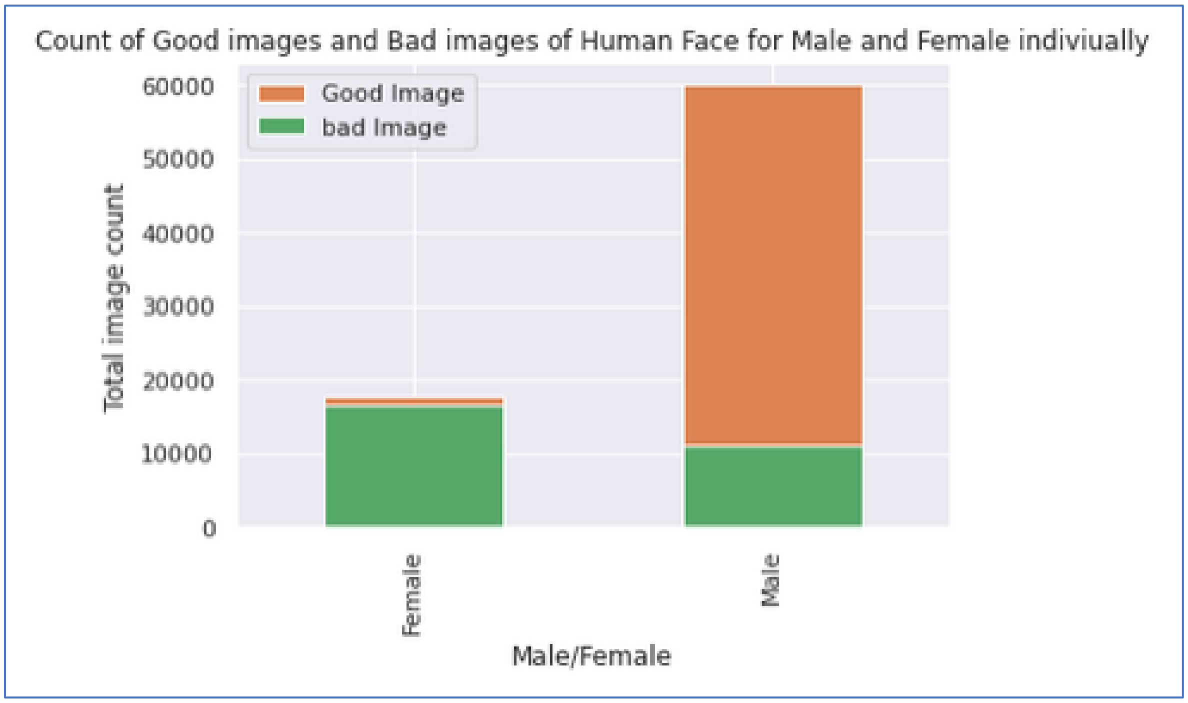
```python
#df_visualize2_1=df_visualize2_1.set_index('Target_Label')
df_visualize2_1=df_visualize2_1.sort_index(ascending=False)
df_visualize2_1
```

	Target_Label	Good/Bad	Total
1	Female	Good Image	17889
0	Male	Good Image	60088

```python
#df_visualize2_2=df_visualize2_2.set_index('Target_Label')
df_visualize2_2
```

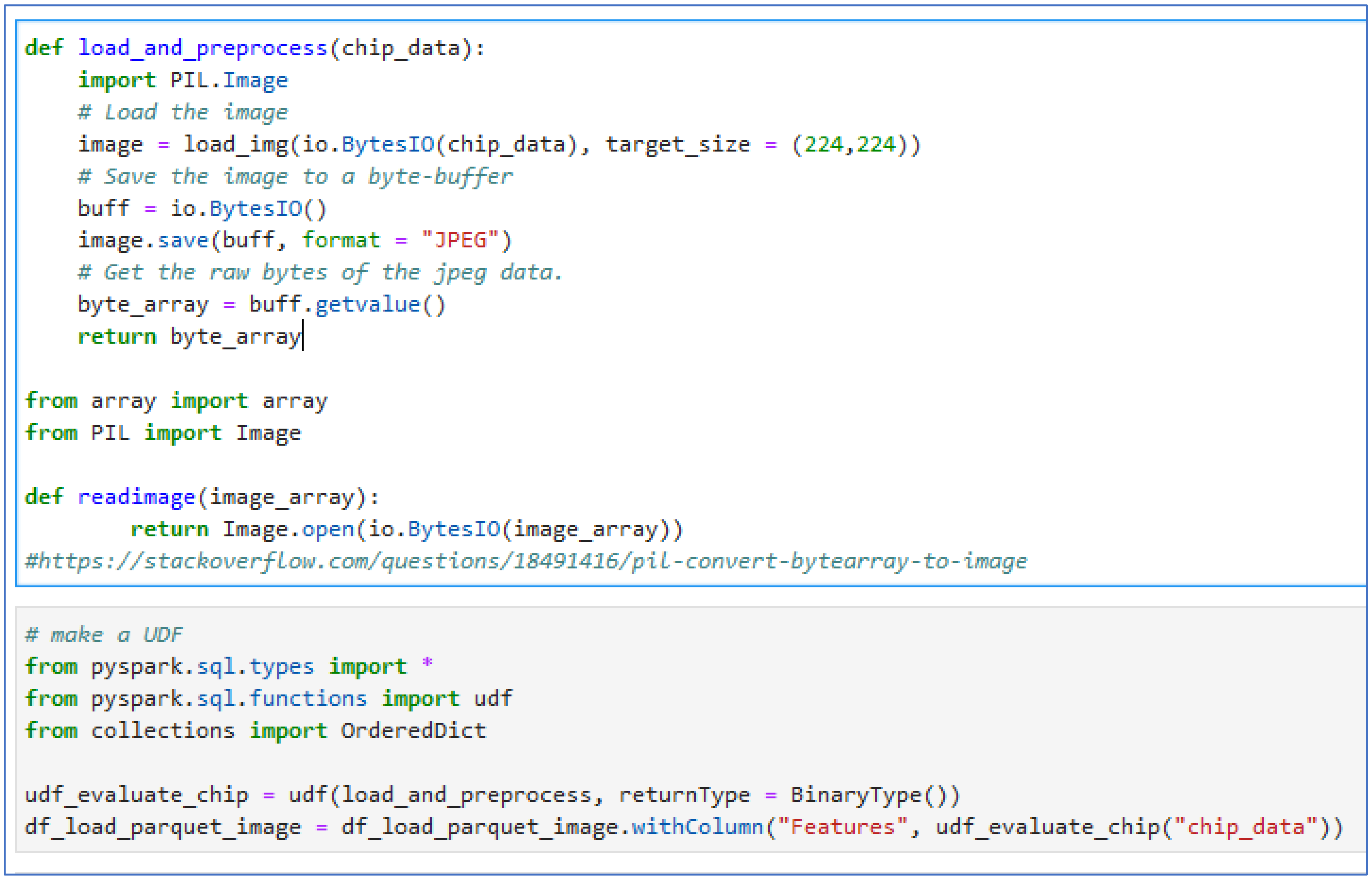
	Target_Label	Good/Bad	Total
0	Female	Bad Image	16545
1	Male	Bad Image	10996

```python
#plotting data using matplotlib
ax = df_visualize2_1.plot(x='Target_Label', y='Total', kind="bar", color ="C1")
df_visualize2_2.plot(x="Target_Label", y="Total", kind="bar", ax=ax, color="C2")
ax.set_ylabel('Total image count')
ax.set_xlabel('Male/Female')
ax.set_title('Count of Good images and Bad images of Human Face for Male and Female indiviually')
ax.legend(labels=['Good Image', 'bad Image'])
plt.show()
```



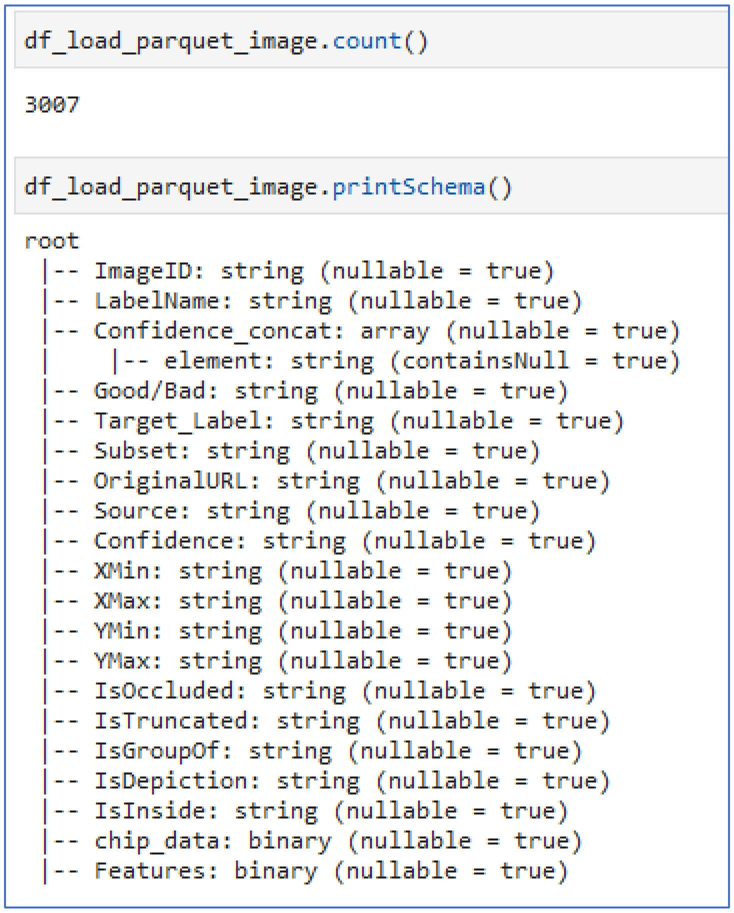
Part 3: Feature Extraction-

For feature extraction, we are using PIL, Image, IO libraries to carry out chip_data to byte array and assign those values in new column called as Features by using UDF function.



```
def load_and_preprocess(chip_data):
    import PIL.Image
    # Load the image
    image = load_img(io.BytesIO(chip_data), target_size = (224,224))
    # Save the image to a byte-buffer
    buff = io.BytesIO()
    image.save(buff, format = "JPEG")
    # Get the raw bytes of the jpeg data.
    byte_array = buff.getvalue()
    return byte_array

from array import array
from PIL import Image

def readimage(image_array):
        return Image.open(io.BytesIO(image_array))
#https://stackoverflow.com/questions/18491416/pil-convert-bytearray-to-image
```

```
# make a UDF
from pyspark.sql.types import *
from pyspark.sql.functions import udf
from collections import OrderedDict

udf_evaluate_chip = udf(load_and_preprocess, returnType = BinaryType())
df_load_parquet_image = df_load_parquet_image.withColumn("Features", udf_evaluate_chip("chip_data"))
```



```
df_load_parquet_image.count()
```

```
3007
```

```
df_load_parquet_image.printSchema()
```

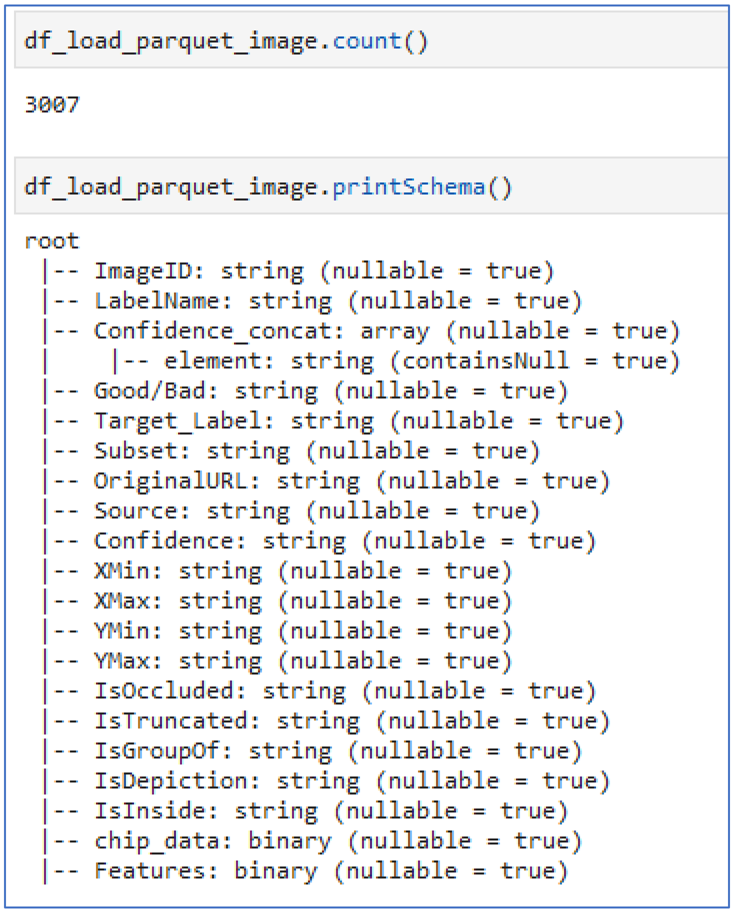
```
root
 |-- ImageID: string (nullable = true)
 |-- LabelName: string (nullable = true)
 |-- Confidence_concat: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- Good/Bad: string (nullable = true)
 |-- Target_Label: string (nullable = true)
 |-- Subset: string (nullable = true)
 |-- OriginalURL: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Confidence: string (nullable = true)
 |-- XMin: string (nullable = true)
 |-- XMax: string (nullable = true)
 |-- YMin: string (nullable = true)
 |-- YMax: string (nullable = true)
 |-- IsOccluded: string (nullable = true)
 |-- IsTruncated: string (nullable = true)
 |-- IsGroupOf: string (nullable = true)
 |-- IsDepiction: string (nullable = true)
 |-- IsInside: string (nullable = true)
 |-- chip_data: binary (nullable = true)
 |-- Features: binary (nullable = true)
```

Issue –

While implementing UDF, we ran into a ‘cannot run anywhere due to node and executor blacklist’. In order to overcome this issue, we had to set spark.blacklist.enabled to false https://kb.informatica.com/solution/23/Pages/73/608104.aspx’.

We face further issues due to improper return type of UDF. Making sure the return type of UDF is spark recognized.

After resolving errors, our final data frame looks like below -



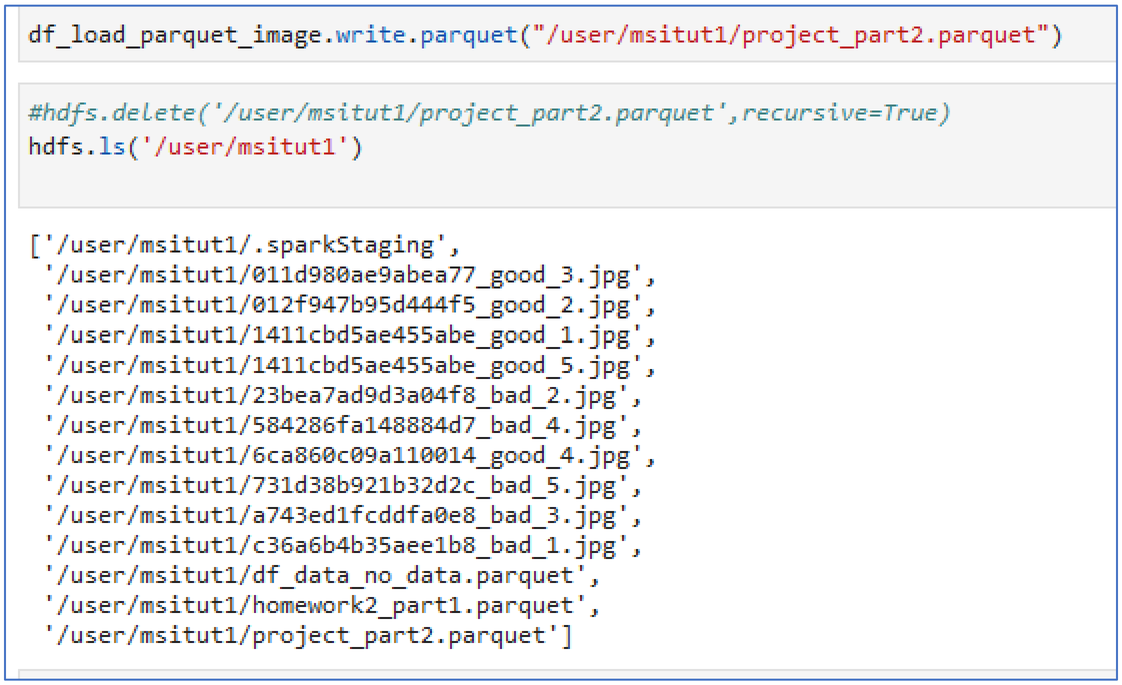
```
df_load_parquet_image.count()
```

```
3007
```

```
df_load_parquet_image.printSchema()
```

```
root
 |-- ImageID: string (nullable = true)
 |-- LabelName: string (nullable = true)
 |-- Confidence_concat: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- Good/Bad: string (nullable = true)
 |-- Target_Label: string (nullable = true)
 |-- Subset: string (nullable = true)
 |-- OriginalURL: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Confidence: string (nullable = true)
 |-- XMin: string (nullable = true)
 |-- XMax: string (nullable = true)
 |-- YMin: string (nullable = true)
 |-- YMax: string (nullable = true)
 |-- IsOccluded: string (nullable = true)
 |-- IsTruncated: string (nullable = true)
 |-- IsGroupOf: string (nullable = true)
 |-- IsDepiction: string (nullable = true)
 |-- IsInside: string (nullable = true)
 |-- chip_data: binary (nullable = true)
 |-- Features: binary (nullable = true)
```

Saving this as parquet file to use in next part 5 while training the model



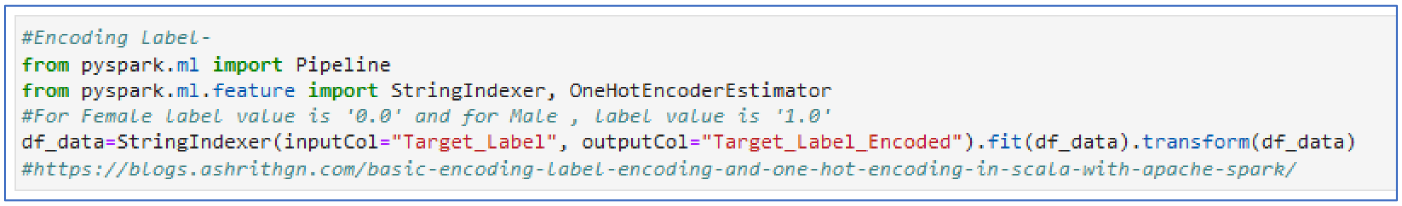
```
df_load_parquet_image.write.parquet("/user/msitut1/project_part2.parquet")
```

```
#hdfs.delete('/user/msitut1/project_part2.parquet',recursive=True)
hdfs.ls('/user/msitut1')
```

```
['/user/msitut1/.sparkStaging',
 '/user/msitut1/011d980ae9abea77_good_3.jpg',
 '/user/msitut1/012f947b95d444f5_good_2.jpg',
 '/user/msitut1/1411cbd5ae455abe_good_1.jpg',
 '/user/msitut1/1411cbd5ae455abe_good_5.jpg',
 '/user/msitut1/23bea7ad9d3a04f8_bad_2.jpg',
 '/user/msitut1/584286fa148884d7_bad_4.jpg',
 '/user/msitut1/6ca860c09a110014_good_4.jpg',
 '/user/msitut1/731d38b921b32d2c_bad_5.jpg',
 '/user/msitut1/a743ed1fcddfa0e8_bad_3.jpg',
 '/user/msitut1/c36a6b4b35aee1b8_bad_1.jpg',
 '/user/msitut1/df_data_no_data.parquet',
 '/user/msitut1/homework2_part1.parquet',
 '/user/msitut1/project_part2.parquet']
```

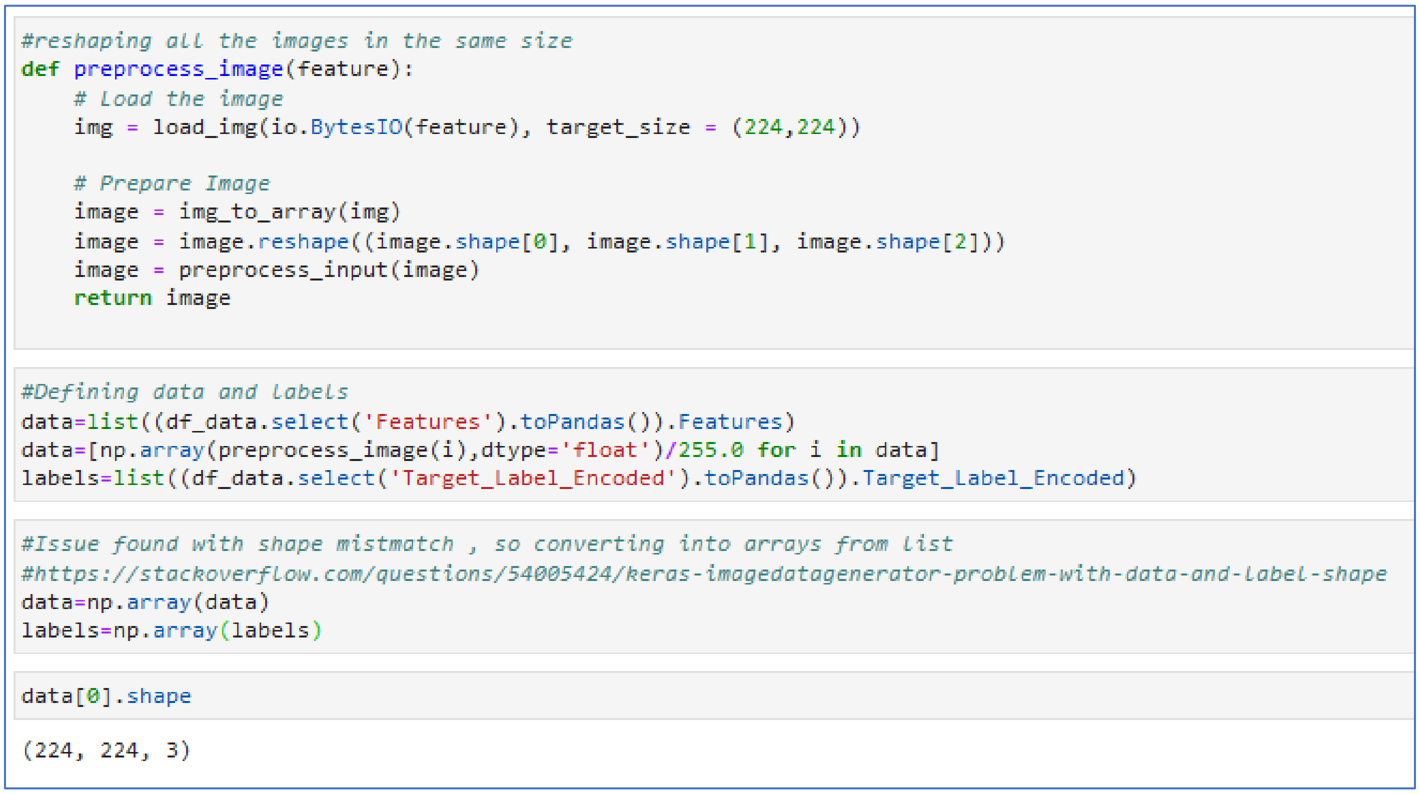
Here our features would be the array from the feature column which contains the bytearray of image and target value would be the labels stored in Target_Label column.

For target labels, we are required to encode the categorical label. We are converting label names (“Female”/” Male”) encoded format using pyspark.ml.feature StringIndexer library as shown below-



```
#Encoding Label-
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoderEstimator
#For Female label value is '0.0' and for Male , label value is '1.0'
df_data=StringIndexer(inputCol="Target_Label", outputCol="Target_Label_Encoded").fit(df_data).transform(df_data)
#https://blogs.ashrithgn.com/basic-encoding-label-encoding-and-one-hot-encoding-in-scala-with-apache-spark/
```

We have features here with different shapes which is a problem as it does not fulfill model/CNN requirements. For that, we have to reshaping all the feature’s arrays using numpy.reshape in a function and calling that function in a list as below



```
#reshaping all the images in the same size
def preprocess_image(feature):
    # Load the image
    img = load_img(io.BytesIO(feature), target_size = (224,224))

    # Prepare Image
    image = img_to_array(img)
    image = image.reshape((image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    return image
```

```
#Defining data and labels
data=list((df_data.select('Features').toPandas()).Features)
data=[np.array(preprocess_image(i),dtype='float')/255.0 for i in data]
labels=list((df_data.select('Target_Label_Encoded').toPandas()).Target_Label_Encoded)
```

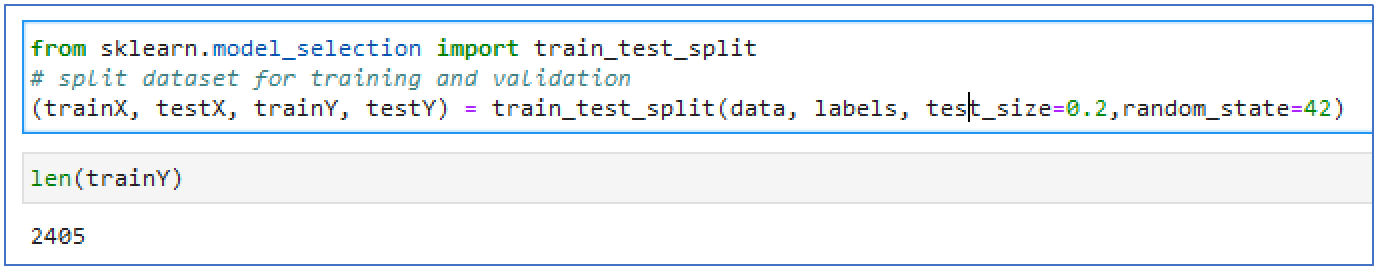
```
#Issue found with shape mismatch , so converting into arrays from list
#https://stackoverflow.com/questions/54005424/keras-imagedatagenerator-problem-with-data-and-label-shape
data=np.array(data)
labels=np.array(labels)
```

```
data[0].shape
```

```
(224, 224, 3)
```

Train and Test data splitting-

We are using sklearn.model_selection’s train_test_split method to split the data and labels into training and testing sets.

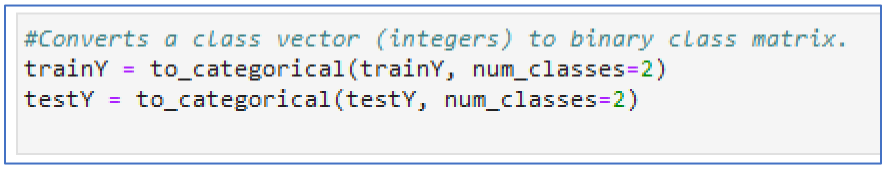


```
from sklearn.model_selection import train_test_split
# split dataset for training and validation
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2,random_state=42)
```

```
len(trainY)
```

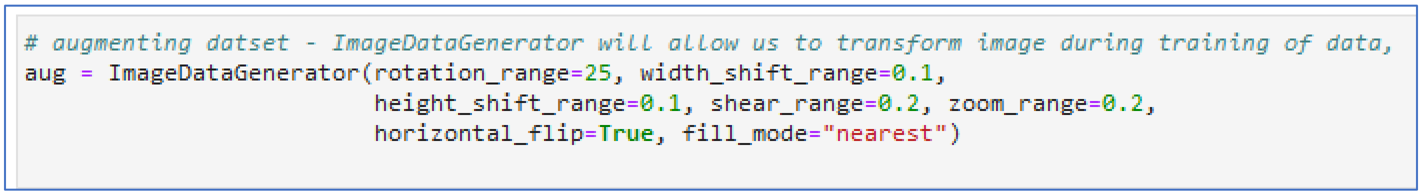
```
2405
```

We are using keras to_categorical method to convert a class vector (integers) to binary class matrix as model target values requirements.



```
#Converts a class vector (integers) to binary class matrix.
trainY = to_categorical(trainY, num_classes=2)
testY = to_categorical(testY, num_classes=2)
```

We are using ImageDataGenerator which will allow us to transform image during training of data-



```
# augmenting datset - ImageDataGenerator will allow us to transform image during training of data,
aug = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
                         height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
                         horizontal_flip=True, fill_mode="nearest")
```

Our data is ready to be trained in part4.

Part 4: Model Training-

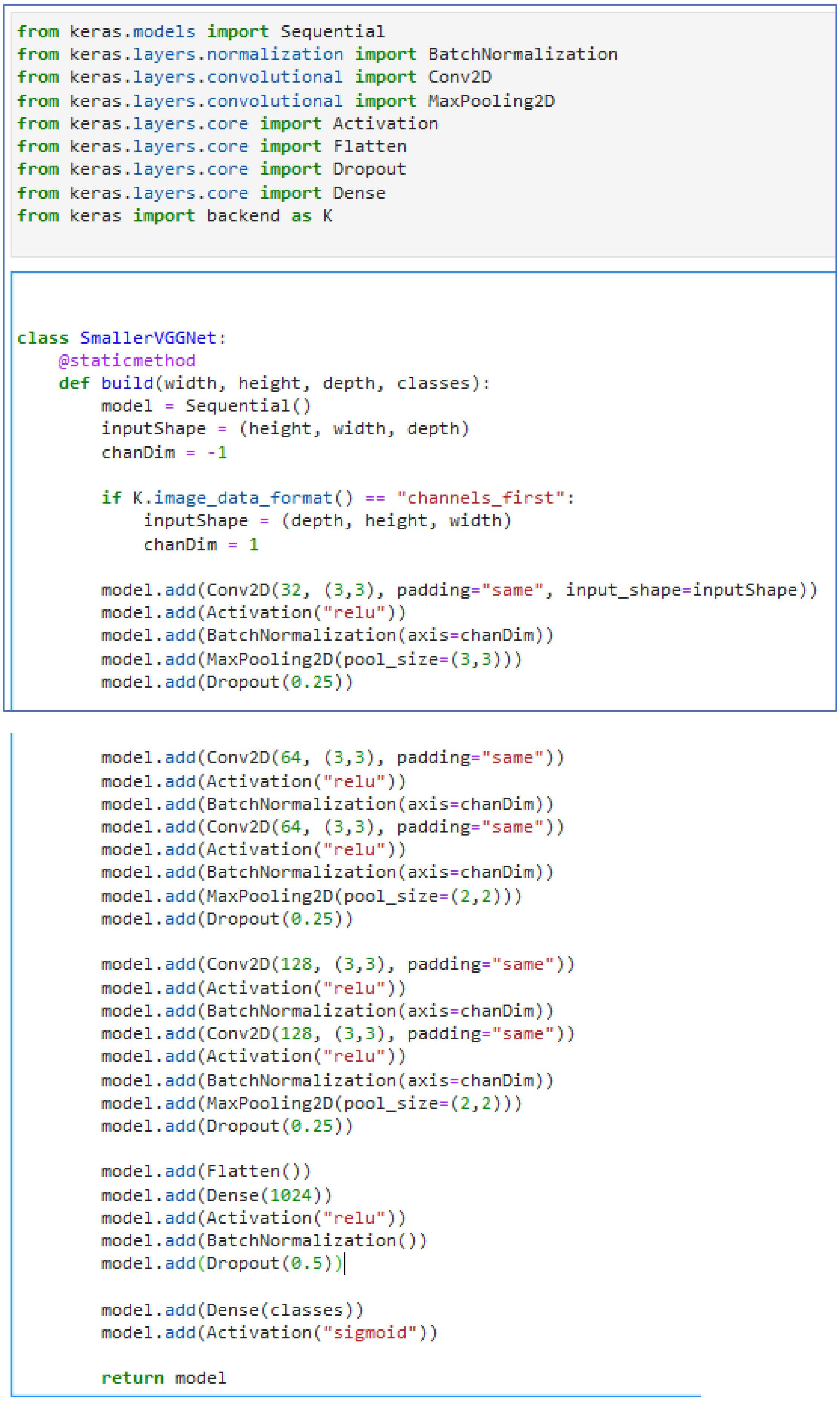
Here we will be using a pre- defined model saved in Model.ipynb. We are using pre-defined model from source- - [https://github.com/arunponnusamy/gender-detection-keras.](https://github.com/arunponnusamy/gender-detection-keras)

This model is a neural network model which is implemented using keras and tensorflow modules. We have used layers of convolutional neural networks in 2 dimensions along with ReLU and Sigmoid as activation functions.

We are using MaxPooling2D to add a pooling layer after the convolution layer to order layers within a convolution neural network that can be repeated in a given model one or more times.

Batch normalization is used to reduce the sum by what the values of the hidden unit's shift (covariance shift).

Our model looks like-

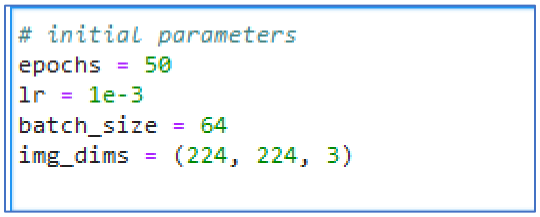


```python
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dropout
from keras.layers.core import Dense
from keras import backend as K
```

```python


class SmallerVGGNet:
    @staticmethod
    def build(width, height, depth, classes):
        model = Sequential()
        inputShape = (height, width, depth)
        chanDim = -1

        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)
            chanDim = 1

        model.add(Conv2D(32, (3,3), padding="same", input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(3,3)))
        model.add(Dropout(0.25))

        model.add(Conv2D(64, (3,3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(Conv2D(64, (3,3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(128, (3,3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(Conv2D(128, (3,3), padding="same"))
        model.add(Activation("relu"))
        model.add(BatchNormalization(axis=chanDim))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.25))

        model.add(Flatten())
        model.add(Dense(1024))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dropout(0.5))

        model.add(Dense(classes))
        model.add(Activation("sigmoid"))

        return model
```

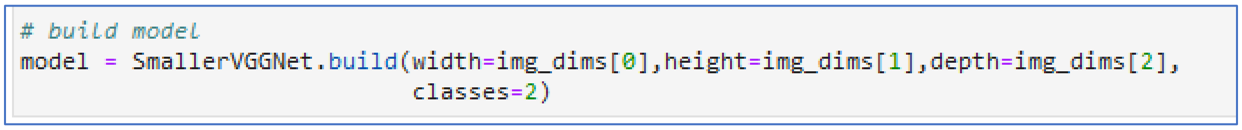
Training the model –

Initializing parameter such as number of epochs(epochs), learning rate (lr), batch size (batch_size), image dimensions (img_dims)



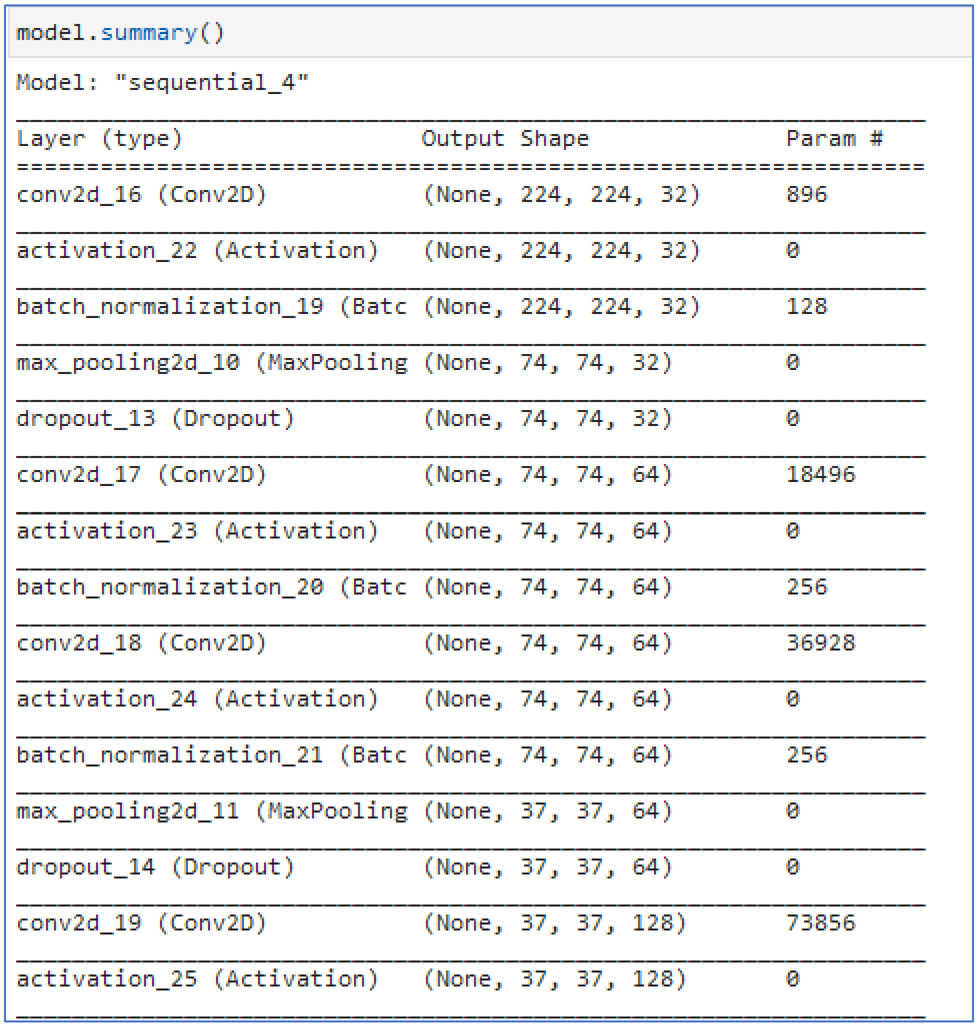
```
# initial parameters
epochs = 50
lr = 1e-3
batch_size = 64
img_dims = (224, 224, 3)
```

Building model –

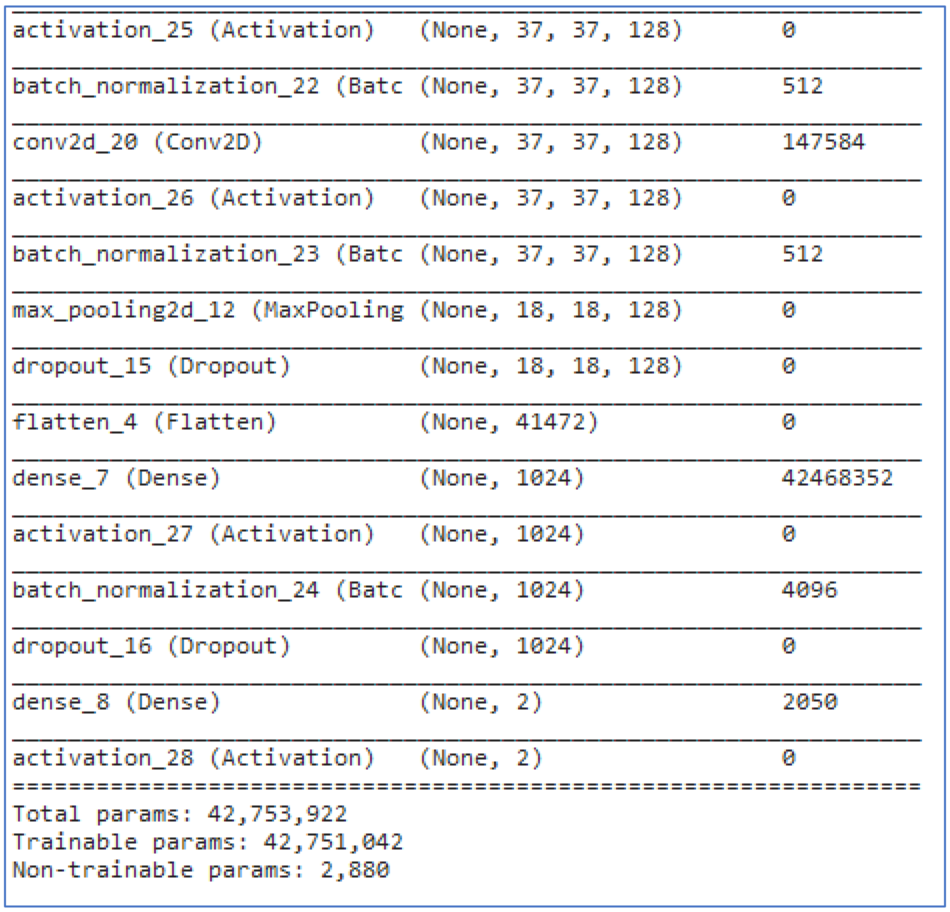


```
# build model
model = SmallerVGGNet.build(width=img_dims[0],height=img_dims[1],depth=img_dims[2],
                            classes=2)
```

Model summary looks like-



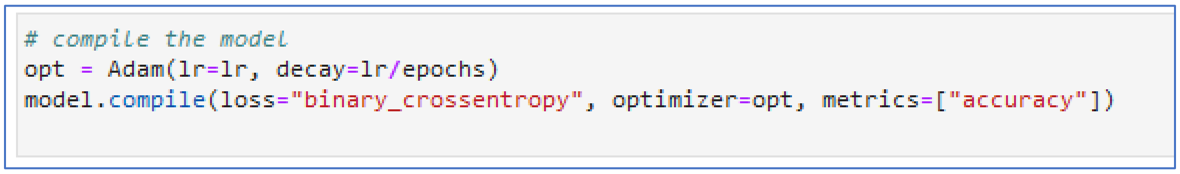
```
model.summary()

Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_16 (Conv2D)           (None, 224, 224, 32)      896       
_________________________________________________________________
activation_22 (Activation)   (None, 224, 224, 32)      0         
_________________________________________________________________
batch_normalization_19 (Batc (None, 224, 224, 32)      128       
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 74, 74, 32)        0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 74, 74, 32)        0         
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 74, 74, 64)        18496     
_________________________________________________________________
activation_23 (Activation)   (None, 74, 74, 64)        0         
_________________________________________________________________
batch_normalization_20 (Batc (None, 74, 74, 64)        256       
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 74, 74, 64)        36928     
_________________________________________________________________
activation_24 (Activation)   (None, 74, 74, 64)        0         
_________________________________________________________________
batch_normalization_21 (Batc (None, 74, 74, 64)        256       
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 37, 37, 64)        0         
_________________________________________________________________
dropout_14 (Dropout)         (None, 37, 37, 64)        0         
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 37, 37, 128)       73856     
_________________________________________________________________
activation_25 (Activation)   (None, 37, 37, 128)       0         
_________________________________________________________________
```



```
_________________________________________________________________
activation_25 (Activation)   (None, 37, 37, 128)       0
_________________________________________________________________
batch_normalization_22 (Batc (None, 37, 37, 128)       512
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 37, 37, 128)       147584
_________________________________________________________________
activation_26 (Activation)   (None, 37, 37, 128)       0
_________________________________________________________________
batch_normalization_23 (Batc (None, 37, 37, 128)       512
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 18, 18, 128)       0
_________________________________________________________________
dropout_15 (Dropout)         (None, 18, 18, 128)       0
_________________________________________________________________
flatten_4 (Flatten)          (None, 41472)             0
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              42468352
_________________________________________________________________
activation_27 (Activation)   (None, 1024)              0
_________________________________________________________________
batch_normalization_24 (Batc (None, 1024)              4096
_________________________________________________________________
dropout_16 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 2050
_________________________________________________________________
activation_28 (Activation)   (None, 2)                 0
=================================================================
Total params: 42,753,922
Trainable params: 42,751,042
Non-trainable params: 2,880
```

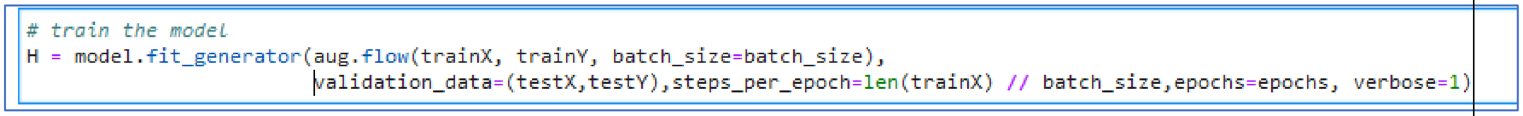
We have compiled the model using Adam optimizer with assigned learning rate, which is based on adaptive estimation of first order and second-order moments.

Also, we have used "binary-crossentropy" as loss function for this classification model.

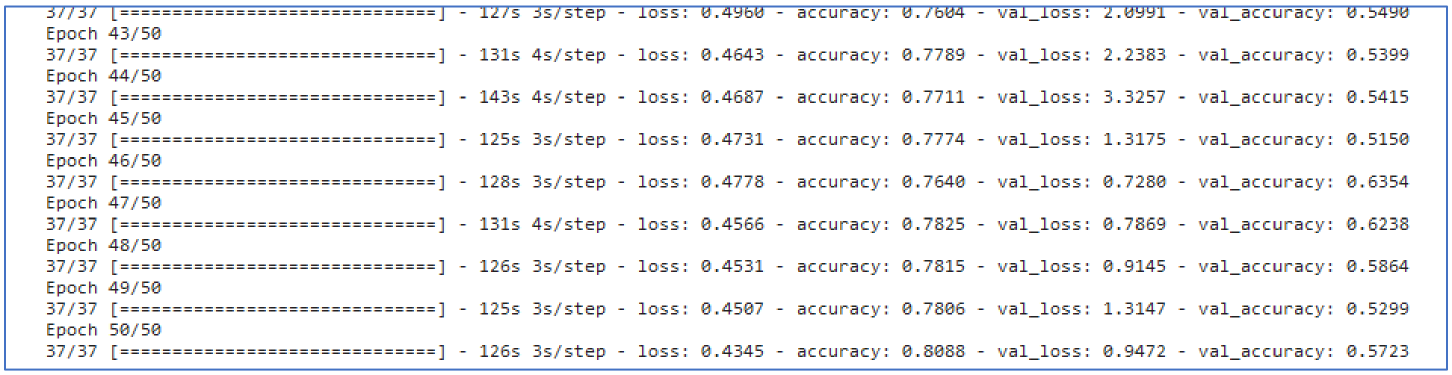


```
# compile the model
opt = Adam(lr=lr, decay=lr/epochs)
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
```

Training the model –



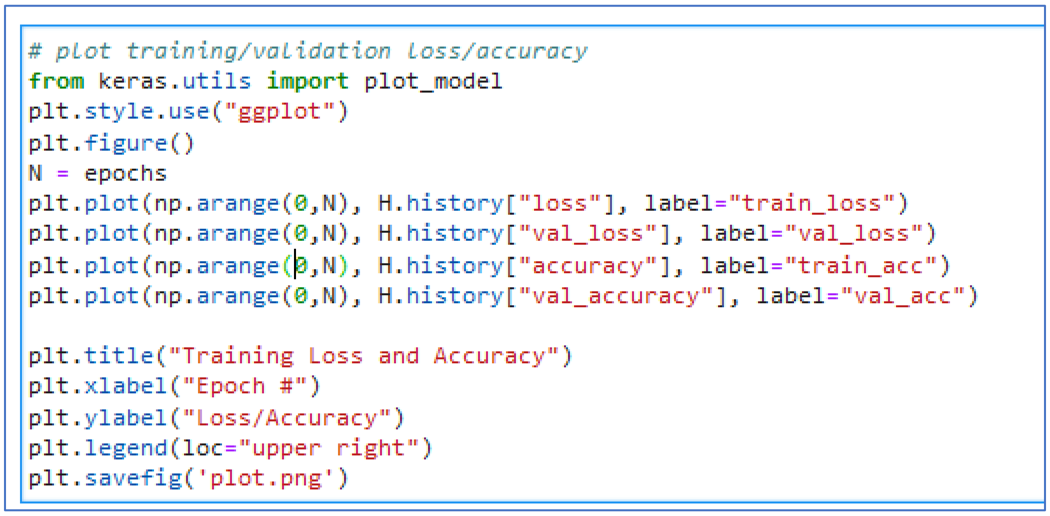
```
# train the model
H = model.fit_generator(aug.flow(trainX, trainY, batch_size=batch_size),
                        validation_data=(testX,testY),steps_per_epoch=len(trainX) // batch_size,epochs=epochs, verbose=1)
```



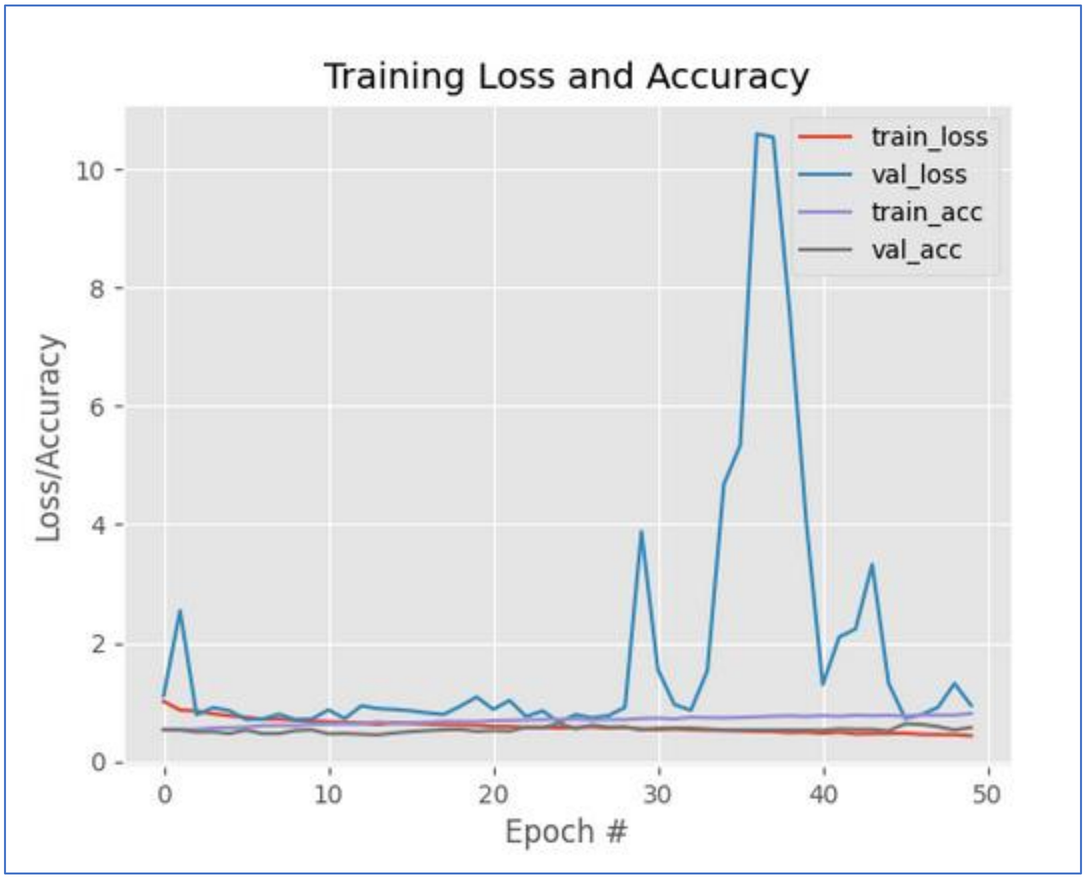
```
37/37 [==============================] - 127s 3s/step - loss: 0.4960 - accuracy: 0.7604 - val_loss: 2.0991 - val_accuracy: 0.5490
Epoch 43/50
37/37 [==============================] - 131s 4s/step - loss: 0.4643 - accuracy: 0.7789 - val_loss: 2.2383 - val_accuracy: 0.5399
Epoch 44/50
37/37 [==============================] - 143s 4s/step - loss: 0.4687 - accuracy: 0.7711 - val_loss: 3.3257 - val_accuracy: 0.5415
Epoch 45/50
37/37 [==============================] - 125s 3s/step - loss: 0.4731 - accuracy: 0.7774 - val_loss: 1.3175 - val_accuracy: 0.5150
Epoch 46/50
37/37 [==============================] - 128s 3s/step - loss: 0.4778 - accuracy: 0.7640 - val_loss: 0.7280 - val_accuracy: 0.6354
Epoch 47/50
37/37 [==============================] - 131s 4s/step - loss: 0.4566 - accuracy: 0.7825 - val_loss: 0.7869 - val_accuracy: 0.6238
Epoch 48/50
37/37 [==============================] - 126s 3s/step - loss: 0.4531 - accuracy: 0.7815 - val_loss: 0.9145 - val_accuracy: 0.5864
Epoch 49/50
37/37 [==============================] - 125s 3s/step - loss: 0.4507 - accuracy: 0.7806 - val_loss: 1.3147 - val_accuracy: 0.5299
Epoch 50/50
37/37 [==============================] - 126s 3s/step - loss: 0.4345 - accuracy: 0.8088 - val_loss: 0.9472 - val_accuracy: 0.5723
```

Recevied accuracy of 80%.

To understand the performance of our model, we carried out accuracy and loss grapf of training and validation set-



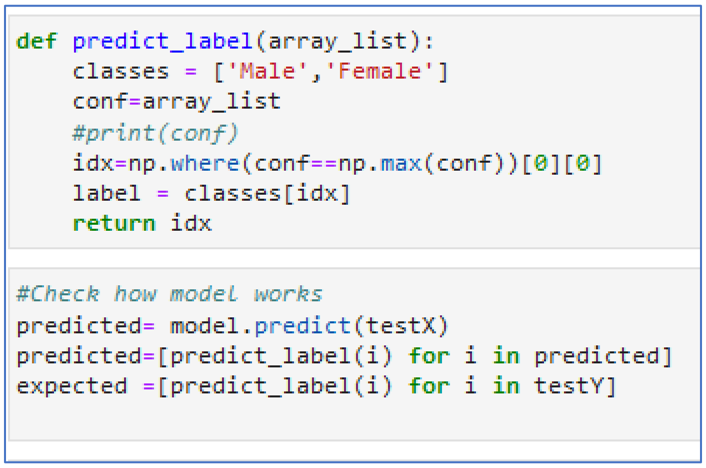
```python
# plot training/validation loss/accuracy
from keras.utils import plot_model
plt.style.use("ggplot")
plt.figure()
N = epochs
plt.plot(np.arange(0,N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0,N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0,N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0,N), H.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper right")
plt.savefig('plot.png')
```



Model Evaluation –

We have performed evaluation based on a few metrics such as classification report, confusion matrix.

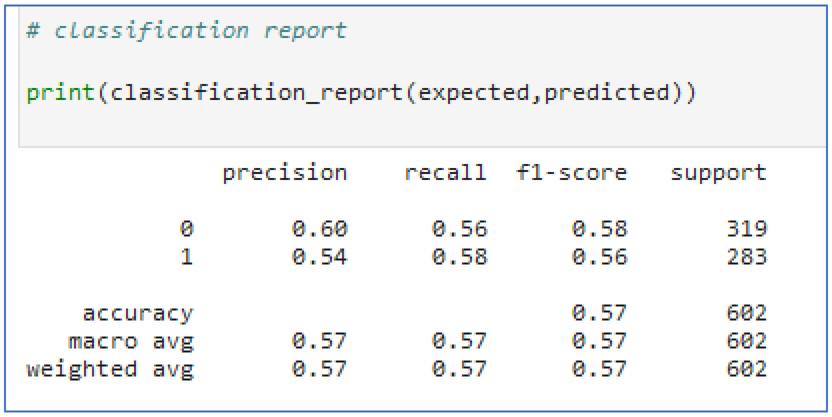
Predicted values and expected values are generating array values which need to be converted to labels. Therefore, we created a function that will carry out the label from an array of predicted and expected values.



```
def predict_label(array_list):
    classes = ['Male','Female']
    conf=array_list
    #print(conf)
    idx=np.where(conf==np.max(conf))[0][0]
    label = classes[idx]
    return idx
```

```
#Check how model works
predicted= model.predict(testX)
predicted=[predict_label(i) for i in predicted]
expected =[predict_label(i) for i in testY]
```

Classification Report for our model-

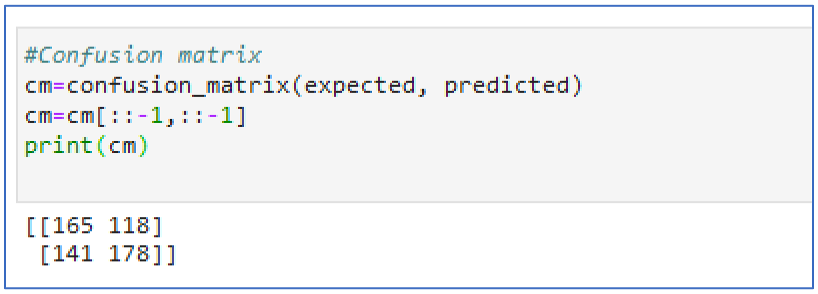


```
# classification report

print(classification_report(expected,predicted))
```

```
              precision    recall  f1-score   support

           0       0.60      0.56      0.58       319
           1       0.54      0.58      0.56       283

    accuracy                           0.57       602
   macro avg       0.57      0.57      0.57       602
weighted avg       0.57      0.57      0.57       602
```

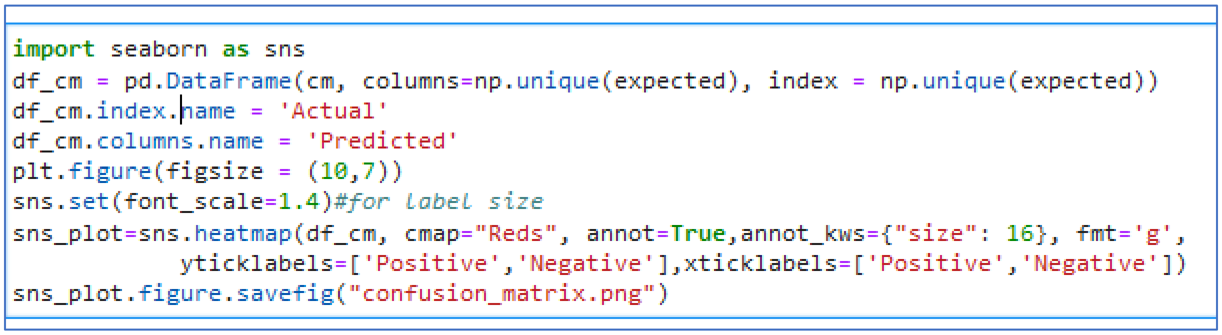
Confusion Matrix for our model-

We have created a confusion matrix to get the performance of our model by carrying out recall, precision scores.

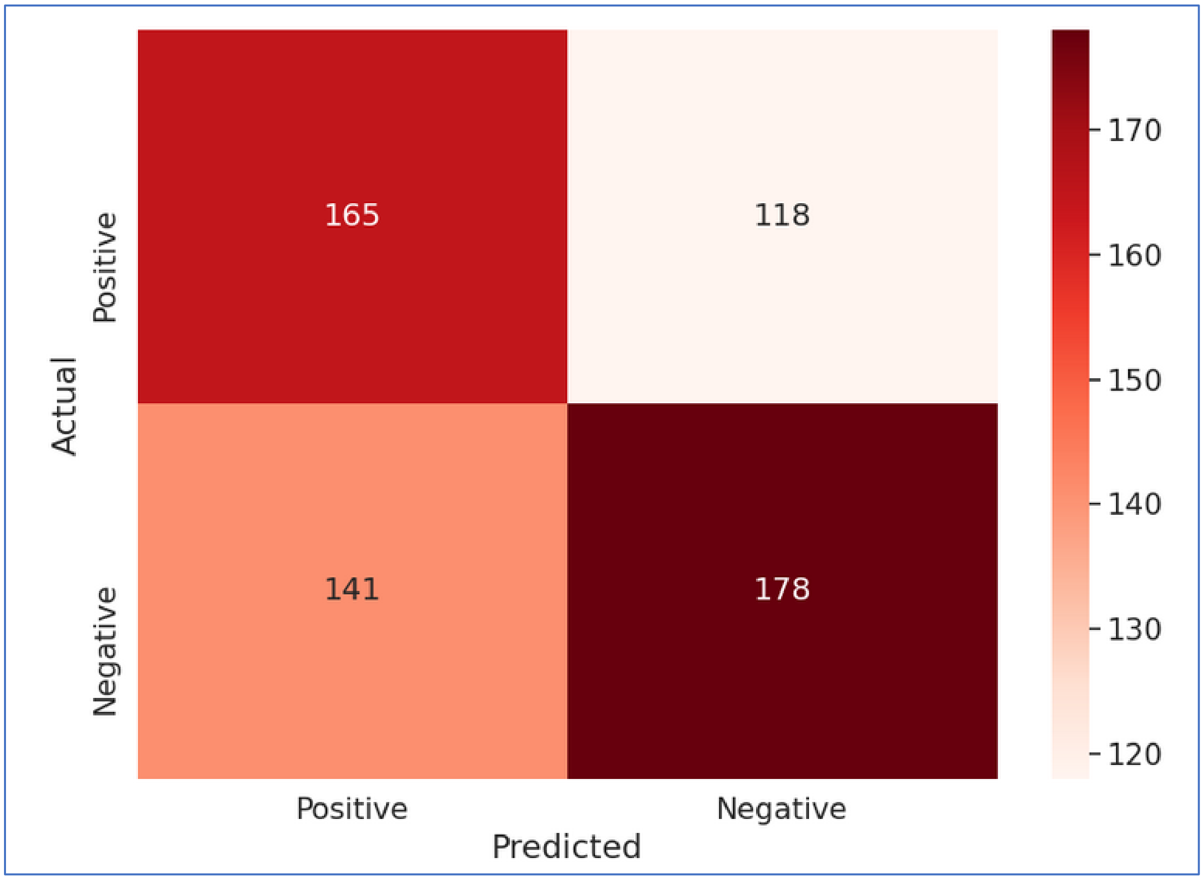


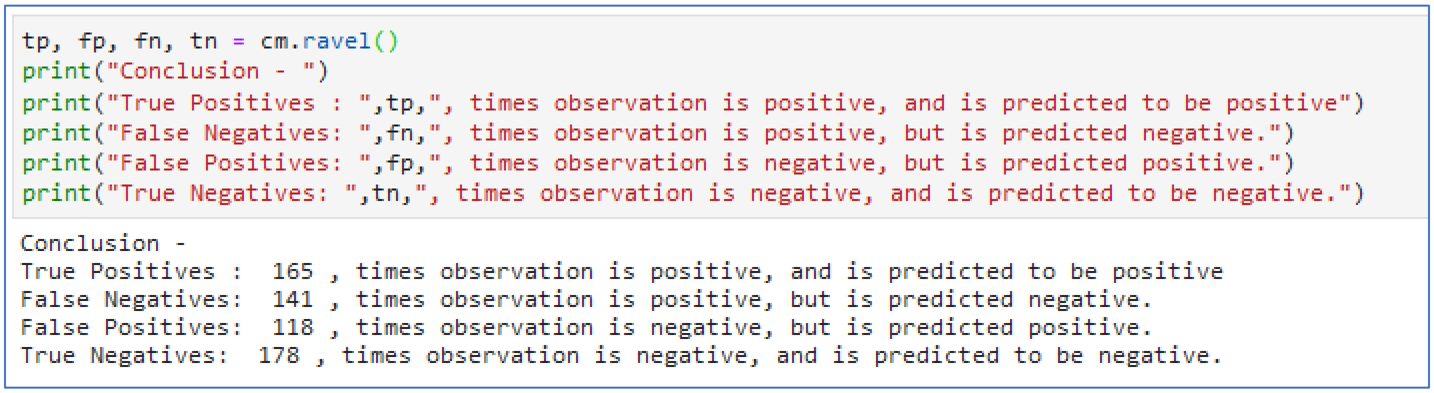
```
#Confusion matrix
cm=confusion_matrix(expected, predicted)
cm=cm[::-1,::-1]
print(cm)
```

```
[[165 118]
 [141 178]]
```



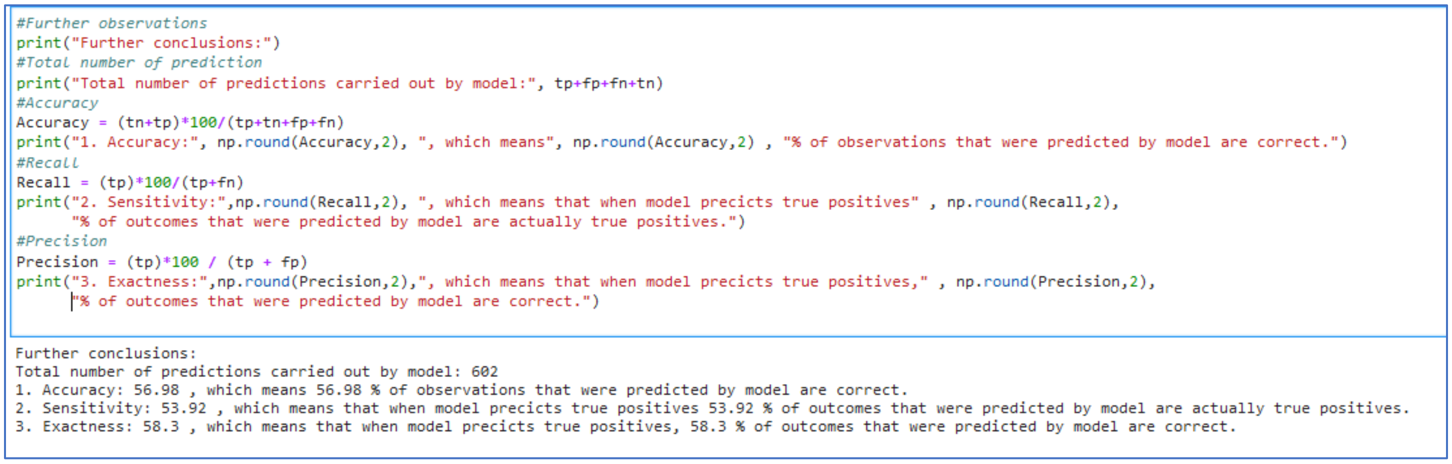
```
import seaborn as sns
df_cm = pd.DataFrame(cm, columns=np.unique(expected), index = np.unique(expected))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns_plot=sns.heatmap(df_cm, cmap="Reds", annot=True,annot_kws={"size": 16}, fmt='g',
            yticklabels=['Positive','Negative'],xticklabels=['Positive','Negative'])
sns_plot.figure.savefig("confusion_matrix.png")
```





```
tp, fp, fn, tn = cm.ravel()
print("Conclusion - ")
print("True Positives : ",tp,", times observation is positive, and is predicted to be positive")
print("False Negatives: ",fn,", times observation is positive, but is predicted negative.")
print("False Positives: ",fp,", times observation is negative, but is predicted positive.")
print("True Negatives: ",tn,", times observation is negative, and is predicted to be negative.")
```

```
Conclusion -
True Positives :  165 , times observation is positive, and is predicted to be positive
False Negatives:  141 , times observation is positive, but is predicted negative.
False Positives:  118 , times observation is negative, but is predicted positive.
True Negatives:  178 , times observation is negative, and is predicted to be negative.
```



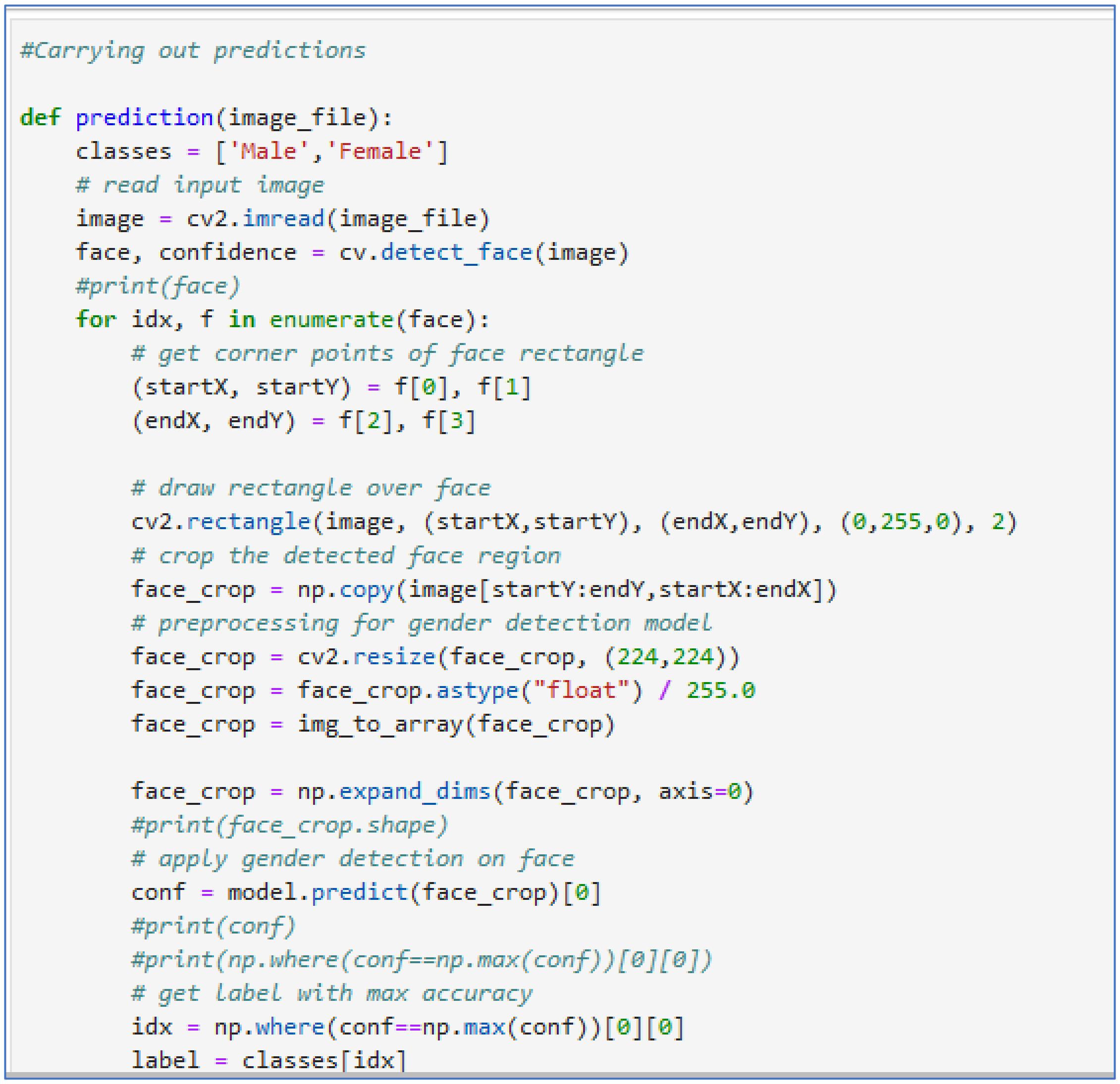
```
#Further observations
print("Further conclusions:")
#Total number of prediction
print("Total number of predictions carried out by model:", tp+fp+fn+tn)
#Accuracy
Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
print("1. Accuracy:", np.round(Accuracy,2), ", which means", np.round(Accuracy,2) , "% of observations that were predicted by model are correct.")
#Recall
Recall = (tp)*100/(tp+fn)
print("2. Sensitivity:",np.round(Recall,2), ", which means that when model precicts true positives" , np.round(Recall,2),
      "% of outcomes that were predicted by model are actually true positives.")
#Precision
Precision = (tp)*100 / (tp + fp)
print("3. Exactness:",np.round(Precision,2),", which means that when model precicts true positives," , np.round(Precision,2),
      "% of outcomes that were predicted by model are correct.")
```

```
Further conclusions:
Total number of predictions carried out by model: 602
1. Accuracy: 56.98 , which means 56.98 % of observations that were predicted by model are correct.
2. Sensitivity: 53.92 , which means that when model precicts true positives 53.92 % of outcomes that were predicted by model are actually true positives.
3. Exactness: 58.3 , which means that when model precicts true positives, 58.3 % of outcomes that were predicted by model are correct.
```

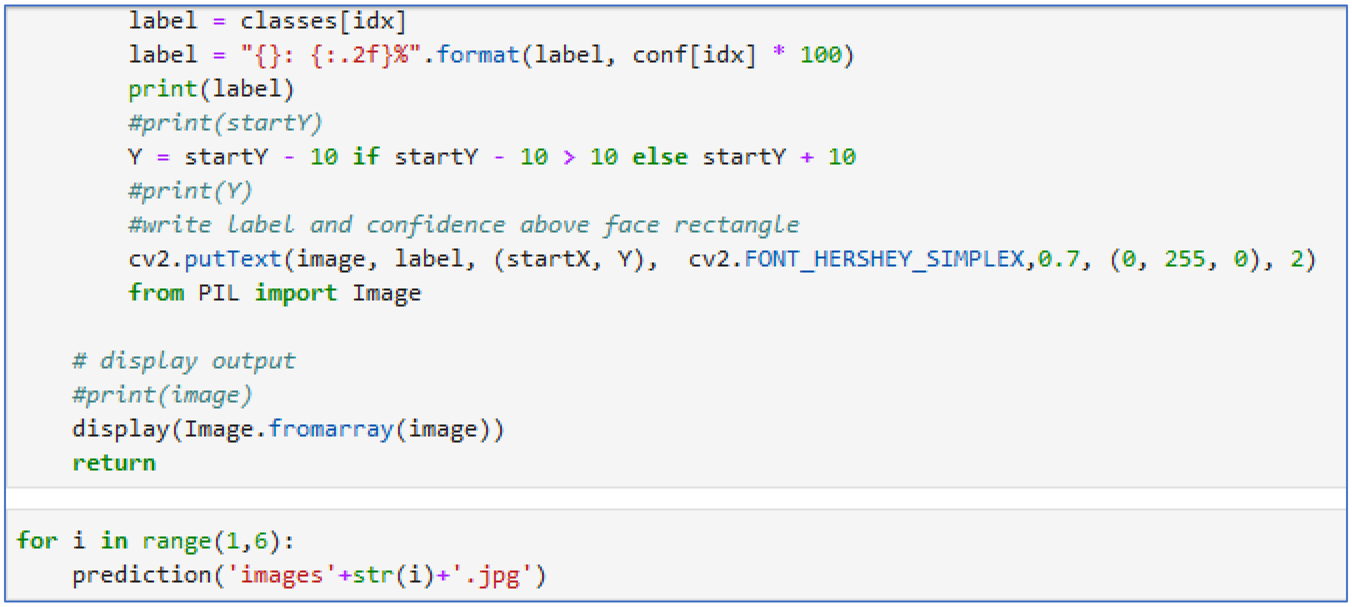
Part 5: Predictions-

We are carrying out predictions by applying random human faces images to model and model will predict gender of the image and create a rectangle box in the picture with gender label and confidence on it.

This will be implemented using opencv and cvlib library.

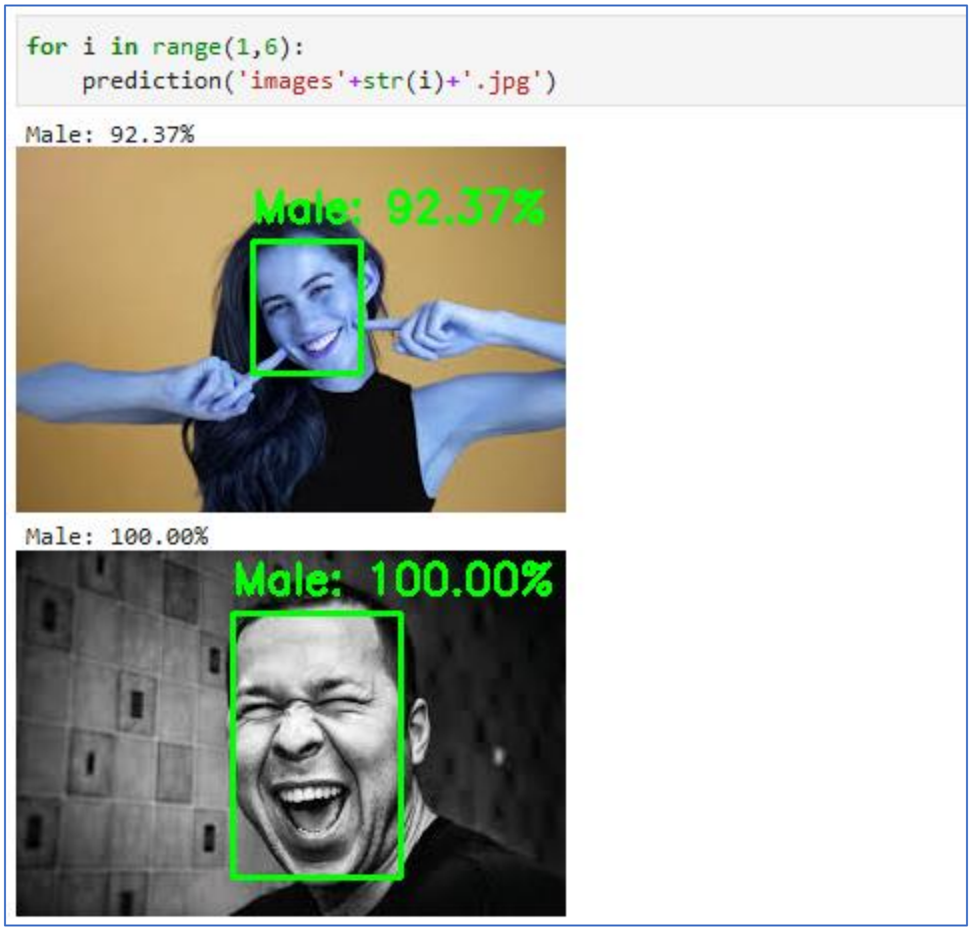


```
#Carrying out predictions

def prediction(image_file):
    classes = ['Male','Female']
    # read input image
    image = cv2.imread(image_file)
    face, confidence = cv.detect_face(image)
    #print(face)
    for idx, f in enumerate(face):
        # get corner points of face rectangle
        (startX, startY) = f[0], f[1]
        (endX, endY) = f[2], f[3]

        # draw rectangle over face
        cv2.rectangle(image, (startX,startY), (endX,endY), (0,255,0), 2)
        # crop the detected face region
        face_crop = np.copy(image[startY:endY,startX:endX])
        # preprocessing for gender detection model
        face_crop = cv2.resize(face_crop, (224,224))
        face_crop = face_crop.astype("float") / 255.0
        face_crop = img_to_array(face_crop)

        face_crop = np.expand_dims(face_crop, axis=0)
        #print(face_crop.shape)
        # apply gender detection on face
        conf = model.predict(face_crop)[0]
        #print(conf)
        #print(np.where(conf==np.max(conf))[0][0])
        # get label with max accuracy
        idx = np.where(conf==np.max(conf))[0][0]
        label = classes[idx]
```



```python
        label = classes[idx]
        label = "{}: {:.2f}%".format(label, conf[idx] * 100)
        print(label)
        #print(startY)
        Y = startY - 10 if startY - 10 > 10 else startY + 10
        #print(Y)
        #write label and confidence above face rectangle
        cv2.putText(image, label, (startX, Y),  cv2.FONT_HERSHEY_SIMPLEX,0.7, (0, 255, 0), 2)
        from PIL import Image

    # display output
    #print(image)
    display(Image.fromarray(image))
    return
```

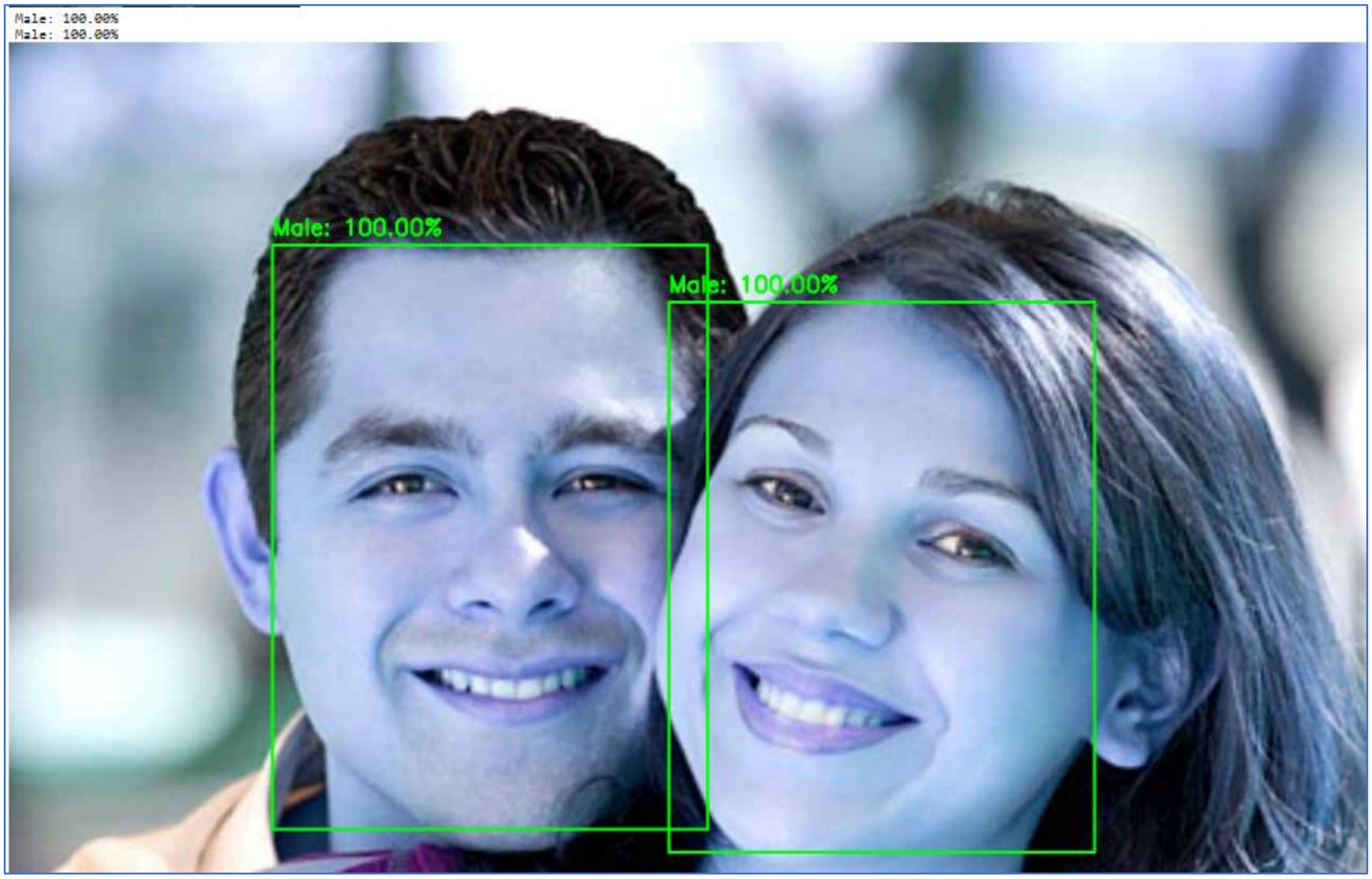
```python
for i in range(1,6):
    prediction('images'+str(i)+'.jpg')
```



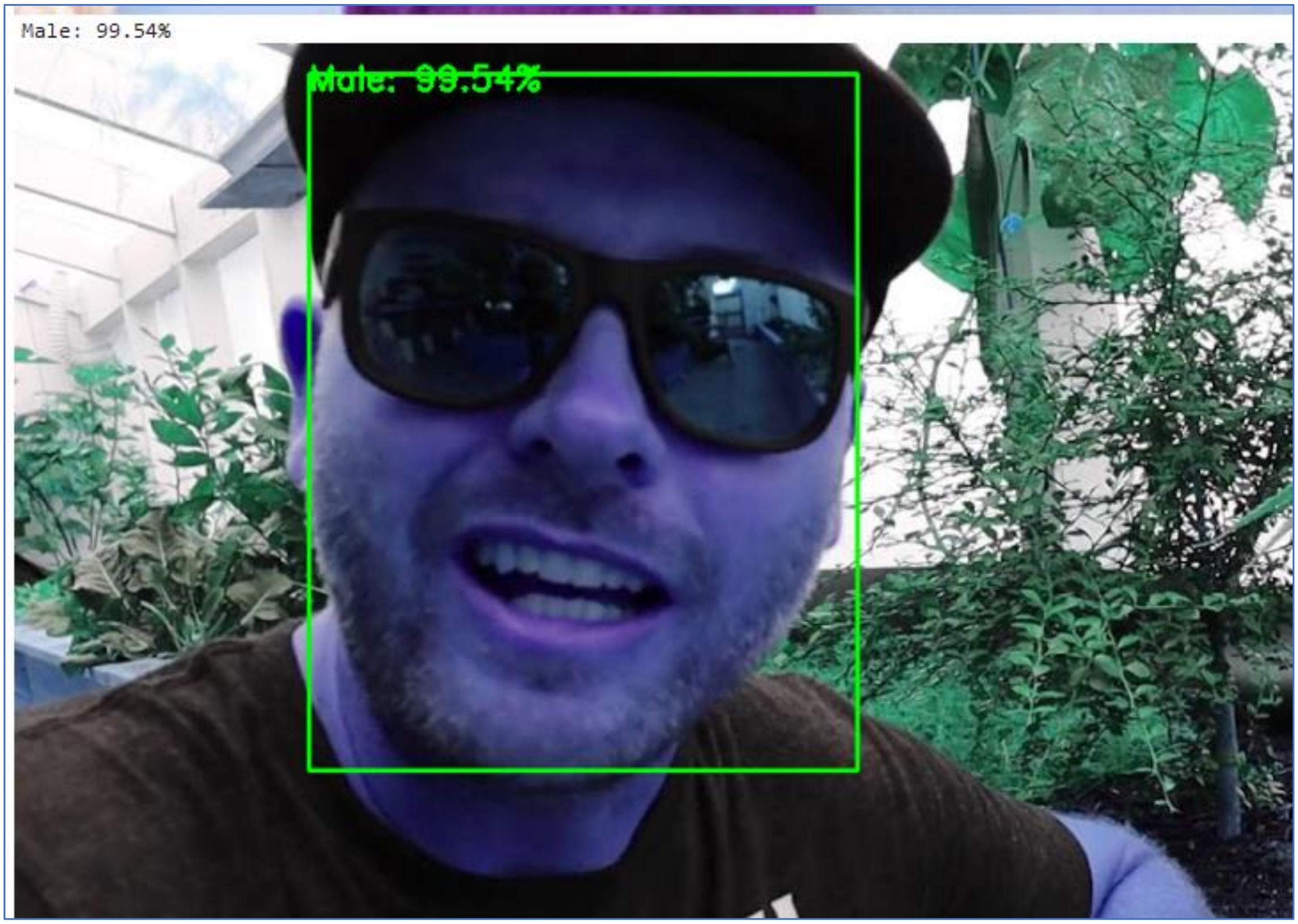
```
for i in range(1,6):
    prediction('images'+str(i)+'.jpg')
```

```
Male: 92.37%
```

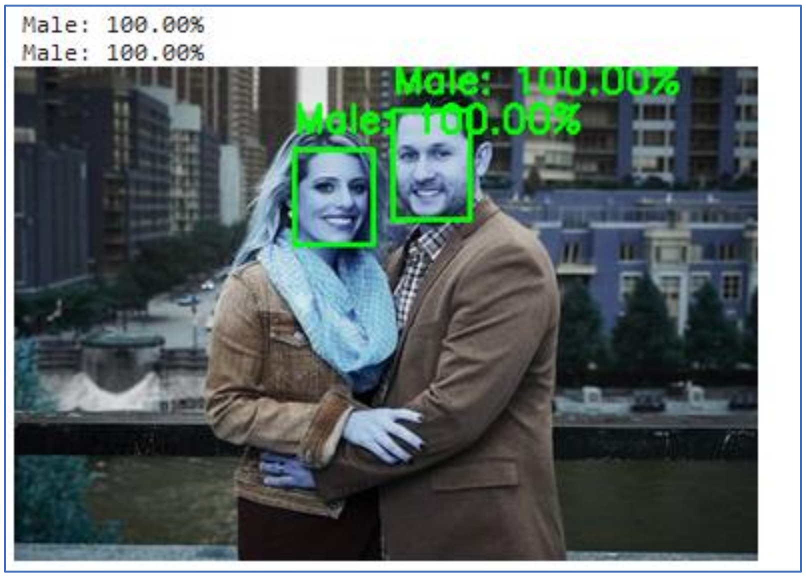
```
Male: 100.00%
```



```
Male: 100.00%
Male: 100.00%
```



```
Male: 99.54%
```



```
Male: 100.00%
Male: 100.00%
```

Conclusion –

So, our model is successfully able to predict Male faces but unable to predict female faces. This might be because we have trained our model with 150 images of male and female. Because we tried to run 50 epochs, but the validation and accuracy graph shows not much of an improvement in accuracy of the validation set.

We can achieve better accuracy if we increase the sample input consisting of image ids of male and female faces from 150 to more and train our model.

References-

- http://www.datasciencemadesimple.com/count-of-missing-nanna-and-null-values-in-pyspark/
- http://www.datasciencemadesimple.com/subset-or-filter-data-with-multiple-conditions-in-pyspark/
- https://stackoverflow.com/questions/44558139/how-to-read-csv-without-header-and-name-them-with-names-while-reading-in-pyspark
- http://www.datasciencemadesimple.com/count-of-missing-nanna-and-null-values-in-pyspark/
- https://stackoverflow.com/questions/44781633/join-pandas-dataframes-based-on-column-values
- https://www.tutorialspoint.com/matplotlib/matplotlib_bar_plot.html
- https://stackoverflow.com/questions/41788919/concatenating-string-by-rows-in-pyspark
- https://stackoverflow.com/questions/51565395/pyspark-create-new-column-and-fill-in-based-on-conditions-of-two-other-columns
- http://www.datasciencemadesimple.com/count-of-missing-nanna-and-null-values-in-pyspark/
- https://stackoverflow.com/questions/36905717/un-persisting-all-dataframes-in-pyspark
- https://stackoverflow.com/questions/18491416/pil-convert-bytearray-to-image
- https://blogs.ashrithgn.com/basic-encoding-label-encoding-and-one-hot-encoding-in-scala-with-apache-spark/
- https://stackoverflow.com/questions/54005424/keras-imagedatagenerator-problem-with-data-and-label-shape