BIG DATA PROJECT

IMDB DATSET

GAURAV PUJARI – PES2201800149

ABHISHEK YOGAN – PES2201800164

ABSTRACT- Movie recommendation system predict what kind of movies a user will like based on the attributes present In previous movies and its beneficial for organizations that collect large amounts of data from the customers and improving their marketing strategies and provide the best suggestions possible. Our project aims at recommending top 3 movies for every user who has rated previous movies and enhance the user satisfaction. We have used ALS model for our project.

First we import the necessary libraries required:

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql import functions as func
import os
import sys
import pandas as pd
from pyspark.ml.recommendation import ALS
from pyspark.sql.types import *
import re
from pyspark.sql.functions import *
from pyspark.ml.evaluation import RegressionEvaluator
```

```
import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.style.use("seaborn-pastel")
```

Only 2 files were necessary out of all the files from the dataset which are title.basics and title.ratings for analysis and recommendation system. The files were taken from the imdb dataset in Kaggle.

```
# creating a spark session.

spark = SparkSession.builder.appName("SparkSQL").getOrCreate()

# we will be loading our files in hdfs and reading it from hdfs to perform the required analysis and visualization.

read_file1=spark.read.option("header","true").option("inferSchema", "true").option("sep","\t").csv("hdfs://127.0.0.1:9000/input1/title.basics.tsv")

read_file2=spark.read.option("header","true").option("inferSchema", "true").option("sep","\t").csv("hdfs://127.0.0.1:9000/input1/title.ratings.tsv")

#once the dataset is loaded pre-processing is done and we will be removing (dropping)the columns which are not required. And since our movie id which is tconst not numeric, we will convert it to numeric.

drop_cols1=['isAdult','startYear','endYear','runtimeMinutes']

df1 = read_file1.drop(*drop_cols1)

#read_file1=read_file1.select([column for column in read_file1.columns if column not in drop_cols1]).columns

#df1.show()

#df1.tconst=pd.Categorical(df1.tconst)
```

#df1["tconst_new"]=df1.tconst.cat.codes

df2 = df1.withColumn('tconst', regexp_replace('tconst', '^tt', '0'))

df3 = read_file2.withColumn('tconst', regexp_replace('tconst', '^tt', '0'))

#typecasting

df2 = df2.withColumn("tconst", df2["tconst"].cast('int'))

df3 = df3.withColumn("tconst", df3["tconst"].cast('int'))

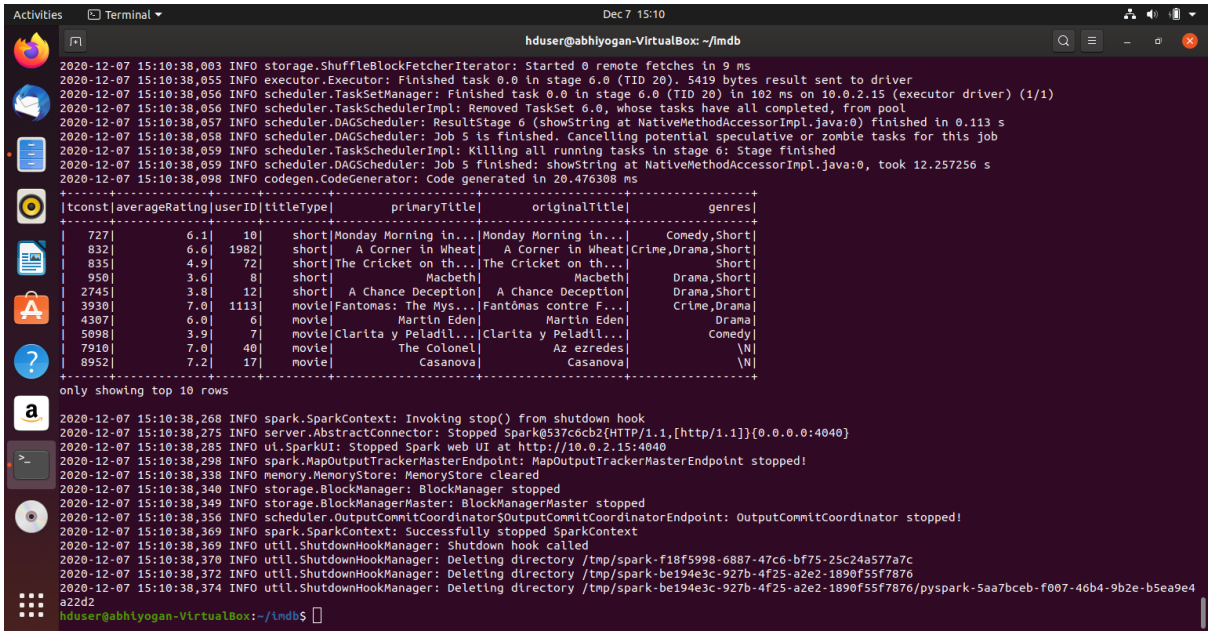
#df3 = df3.withColumn("tconst",round(rand()*(1000-5)+5,0))

#joining the dataframes.

joined_table=df3.join(df2, ['tconst']).dropDuplicates()

joined_table.show(10)

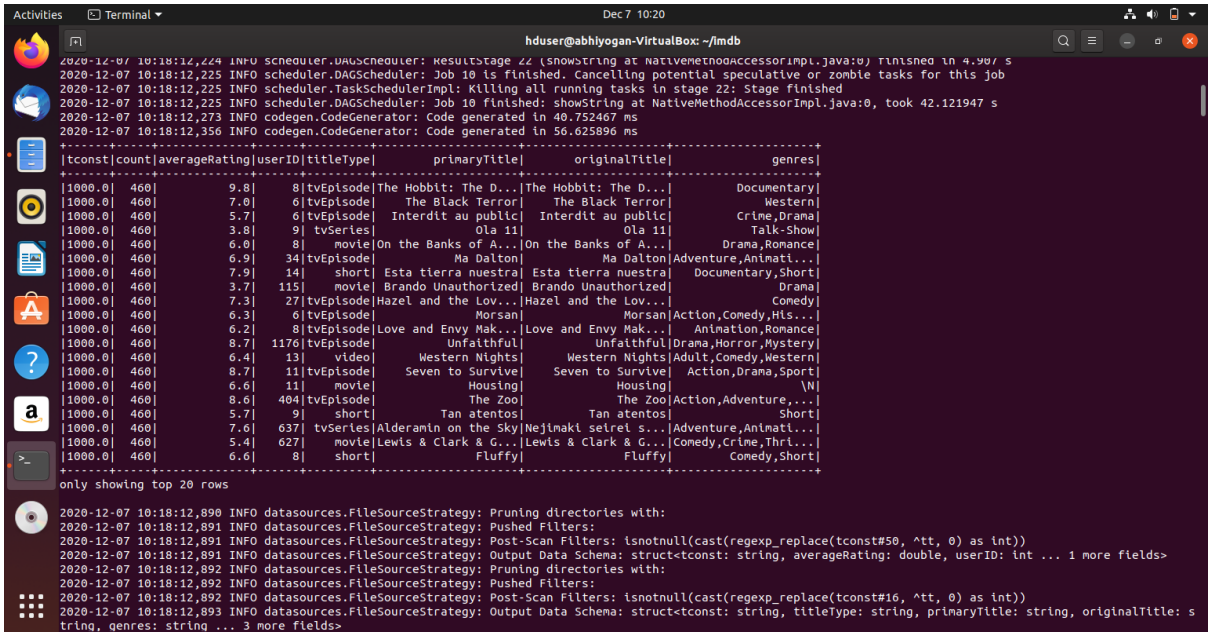
joined_table = joined_table.withColumn("tconst",round(rand()*(1000-5)+5,0))



Queries:

Finding the most rated movie in our data set

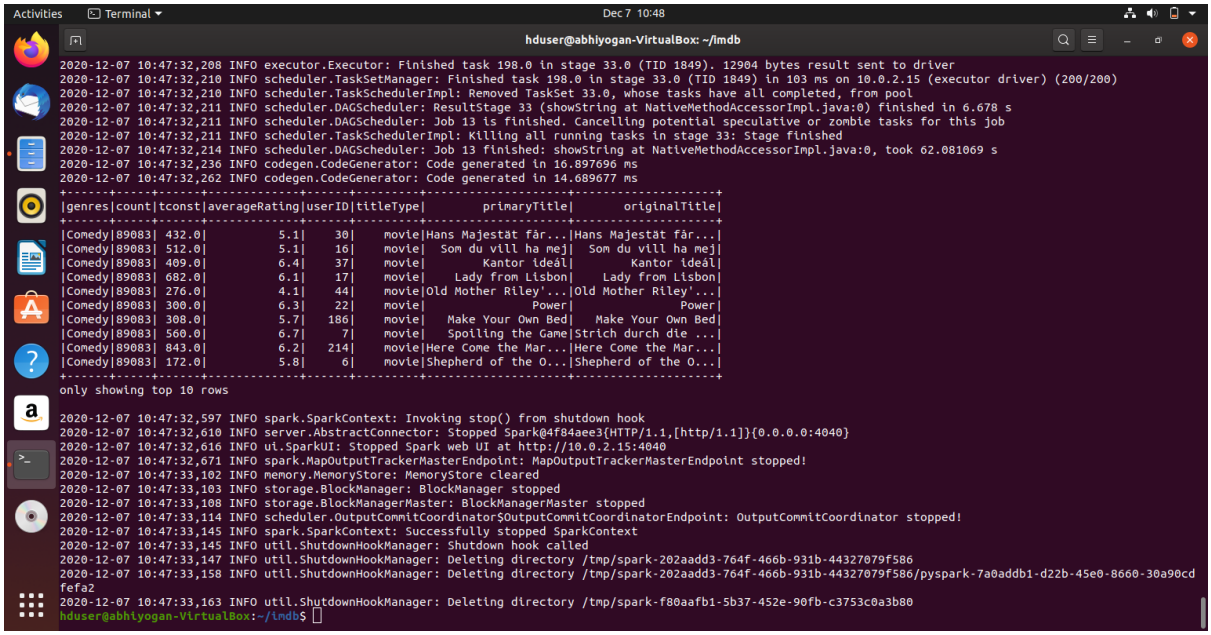
```
def most_rated_movie():
    most_rated = joined_table.groupby('tconst').count().orderBy(func.asc("count"))
        most_rated = most_rated.join(joined_table, ['tconst'])
        most_rated = most_rated.orderBy("count", descending = False)
        return most_rated


#y = most_rated_movie()
#y.show()
```



This tells what movies users like the most.

To find the most like genre:

```
def most_liked_genre():
        genre = joined_table.groupby('genres').count().orderBy(func.desc("count"))
        genre = genre.join(joined_table, ['genres'])
        genre = genre.orderBy("count", ascending = False)
        return genre
```



We find that comedy is the most liked genre.

Graph Plotting:

A bar plot for the most rated movie:

plot = most_rated_movie()

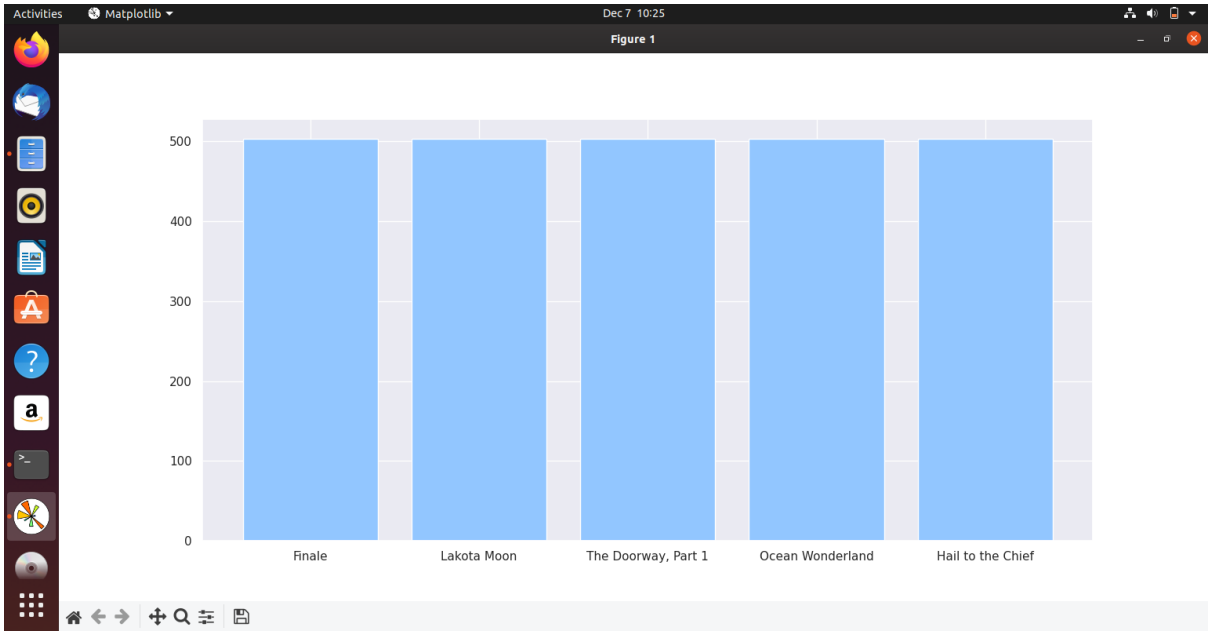
grouping = plot.orderBy("count", descending = False).select(['primaryTitle', 'count']).limit(5)

s = grouping.toPandas()['primaryTitle'].values.tolist()

t = grouping.toPandas()['count'].values.tolist()

plt.bar(s,t)

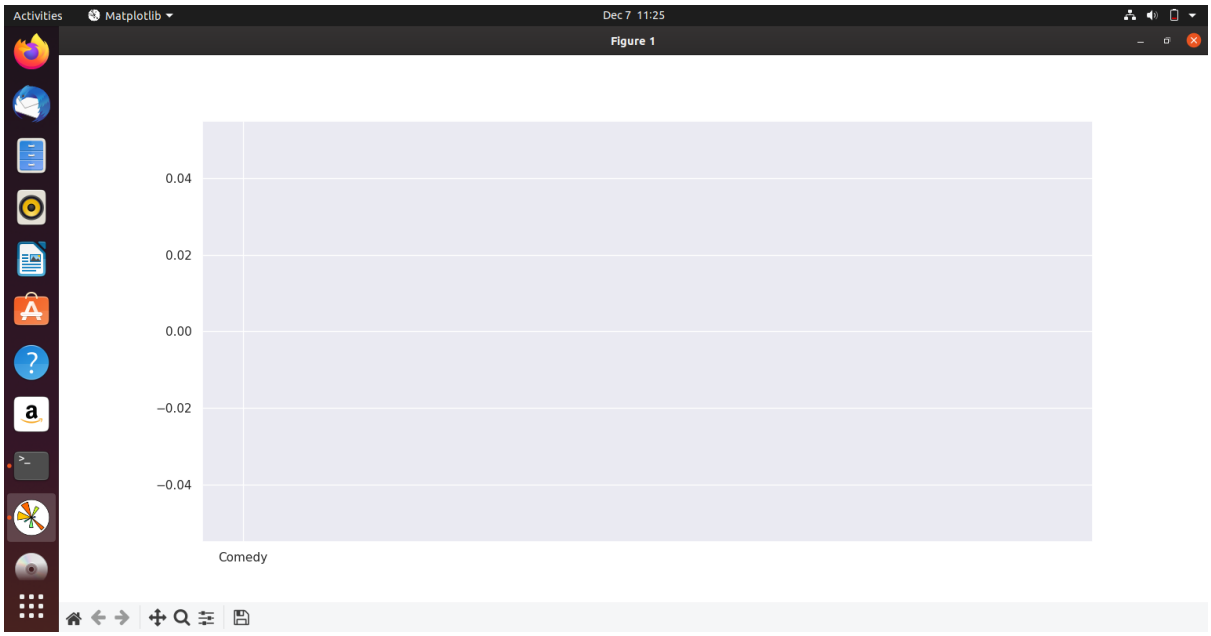
plt.show()



Plotting a histogram plot for the most liked genre:

```
b = most_liked_genre()

grouping = b.orderBy("count", ascending = False).select(['genres', 'count']).limit(5)

s = grouping.toPandas()['genres'].values.tolist()

t = grouping.toPandas()['count'].values.tolist()

#plt.scatter(s,t)
```

#plt.show()



Since comedy was the only most liked genre we got the plot for only comedy genre.

scatter plot for genre and avg rating:

ALS MODEL:

#Now we perform alternating least squares method , userid is not the only factor affecting the rating ,genre also plays a role , genre is the latent factor in our project.

als= ALS(maxIter=10, regParam=0.5, userCol="userID",itemCol="tconst", ratingCol="averageRating", nonnegative = True, implicitPrefs=False, coldStartStrategy="drop")

train, test = joined_table.randomSplit([0.8, 0.2])

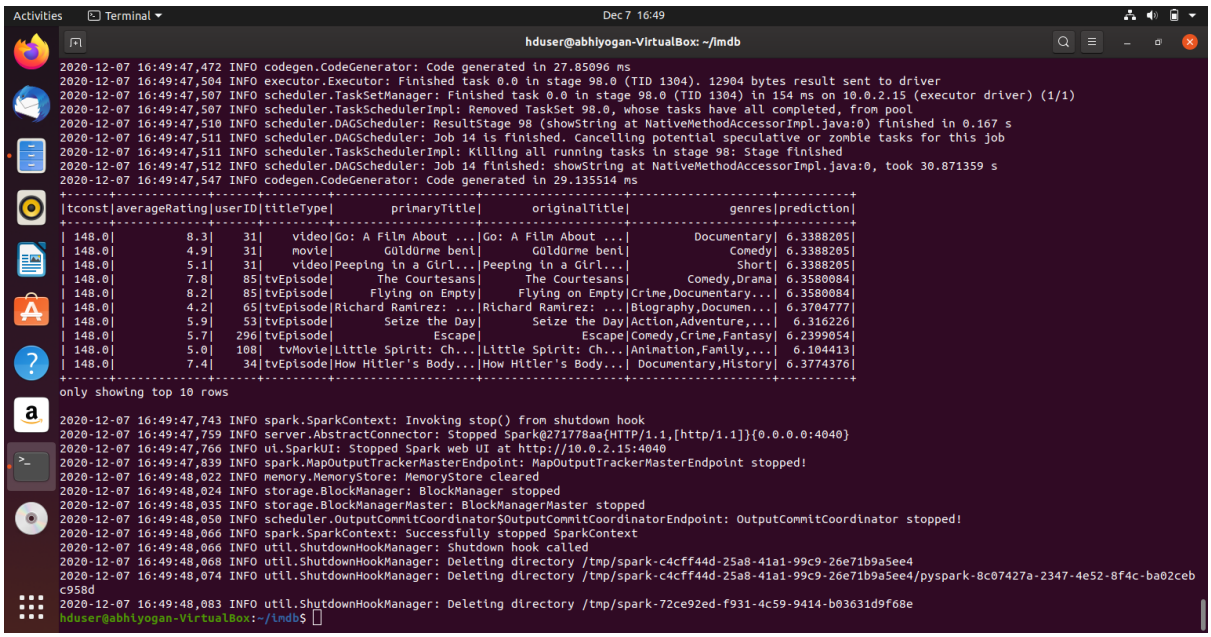
#training the model

alsModel = als.fit(train)

#generating predictions

prediction = alsModel.transform(test)

prediction.show(10)



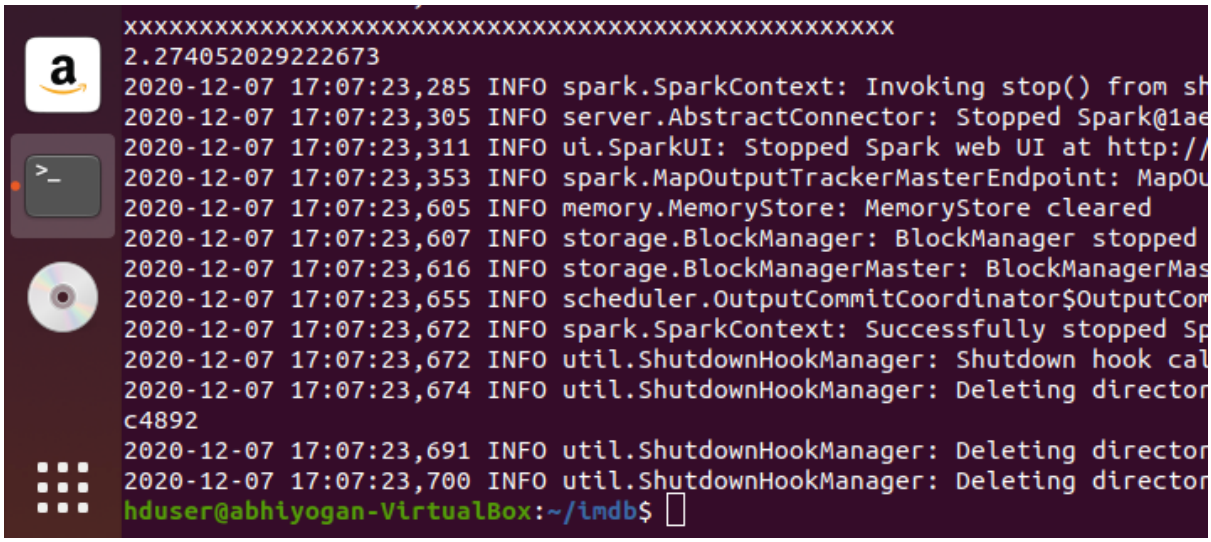
Root mean square value turns out to be 2.2740 which is a decent value , taking into consideration our dataset which was sparse and had NAN values.

evaluator = RegressionEvaluator(metricName="mse", labelCol="averageRating", predictionCol="prediction")

mse = evaluator.evaluate(prediction)

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

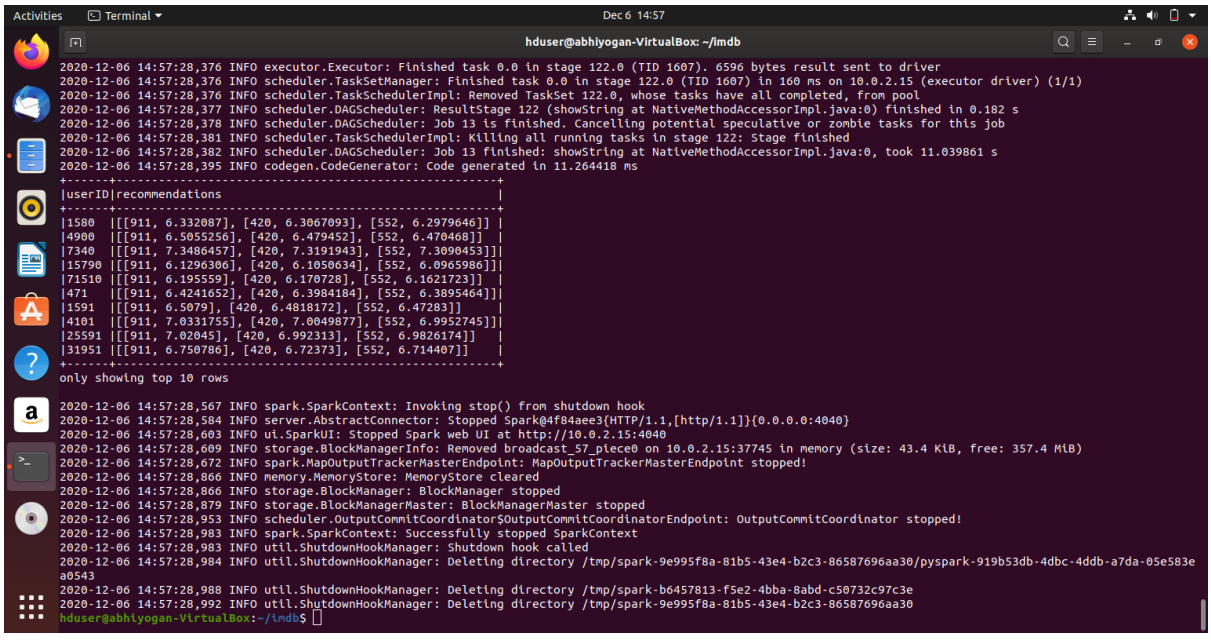
print(mse)



recommendation output of top 3 movies for all the users.:

recommended_movie_df = alsModel.recommendForAllUsers(3)

recommended_movie_df.show(10, False)



Conclusions:

Considering the dataset our recommender system provides fairly good results. Adding more valuable data to the dataset could make the recommender

system even better considering the fact that how our dataset was filled with garbage values.