Introduction

As my Hands-On project I have chosen to perform Stellar classification on the Stellar Classification dataset from Kaggle. Stellar Classification is an interesting and an important topic in astronomy, as it helps us understand the behavior and properties of different types of stars observed in space. Stellar classification is an essential area of study in astronomy that involves categorizing stars based on their spectral features. The Observed celestial bodies can be classified into 3 main categories: GALAXY, STAR and QSO(quasars). This has played a very crucial role in our understanding of the universe. The early efforts to catalog stars and their distribution has led to the realization that constitute our galaxy.

The key problem here is to identify from a limited set of parameters with high accuracy regarding the type of celestial body being referred as the data can often be highly time and location specific which contributes to non-specific correlations between the label and features. Additionally, many of the features in the dataset are purely recording and measurement based and have to be discarded as they negatively affect the modelling. As a primary approach to this problem, I have employed various EDA techniques to identify the key features and use an array of ML classifiers to compare and utilize the best models to achieve the best results in the classification.

Background

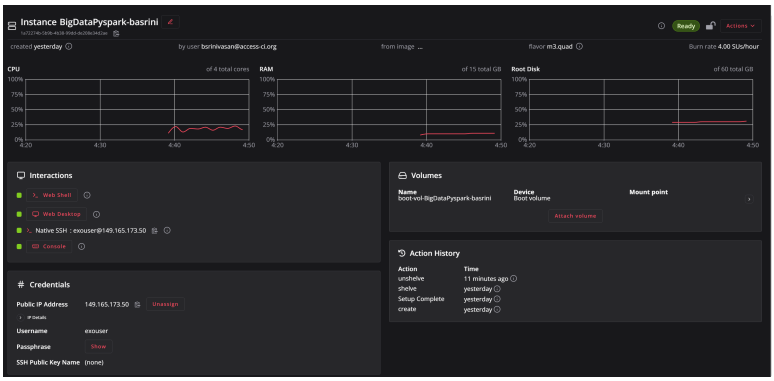
I have always had a fascination towards the night sky and celestial phenomenons, I used to own a small scale telescope and used to wait long time for clear skies to observe the space in the night and also on trek/camp nights. Knowing the properties and behavior of different types of stars can help understand how galaxies form and evolve eventually leading to the mystery of life as it is. Such an intriguing concept that has been tormenting mankind since "the beginning of time" can almost exclusively be answered by observing such phenomenon first hand. This project helps us identify what to expect for and what sort of behavior we could get to observe and help us reach a step closer towards the unknown.

Our eyes and often inferences from anyone elses' observation can often lead to misclassification of bodies and can potentially lead to missing out on rare celestial events which would be truly unfortunate. Given the advancement in technology and SOTA clasification techinque we can provide an accurate and near perfect classification of the data giving us a fighting chance!! Thus I chose this as my project.

Methodology

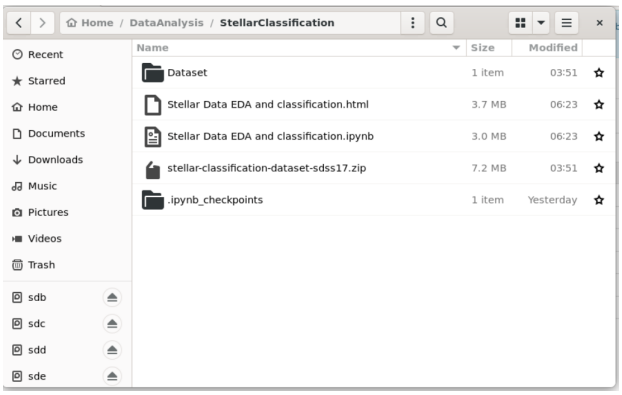
Setup

- J2 instance was my choice of environment to code and run the dataset while using Anaconda installation for all the tools necessary.
- The work is entirely performed in a Jupyter notebook with necessary conda environment.
- I followed a multi-step methodology to classify stars into different types.
- I used the m3.quad core 16 GB version of J2 with 60 GB volume instance with latest Ubuntu OS.



J2 Instance

- The installation of anaconda and jupyter notebook and required libraries was straightforward using th eprevious assignments as tutorials.
- The folder structure used is fairly simple as follows and only needs the Jupyter notebook and your latest kaggle.json in the core .kaggle folder for the complete project recreation on any device.



Folder Structure in J2

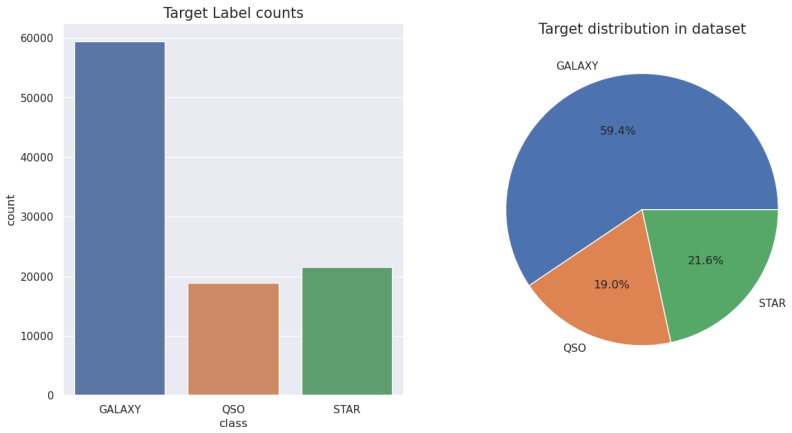
Data Cleaning, Feature Engineering and EDA

- The data is directly loaded into the storage location using kaggle api and anr user kaggle api token(updated each diffrent day being used).

- The primary data format is CSV from the kaggle data set ina zip format which contains heterogenous mixtures of features that need to be loaded cleaned and processed during EDA. The

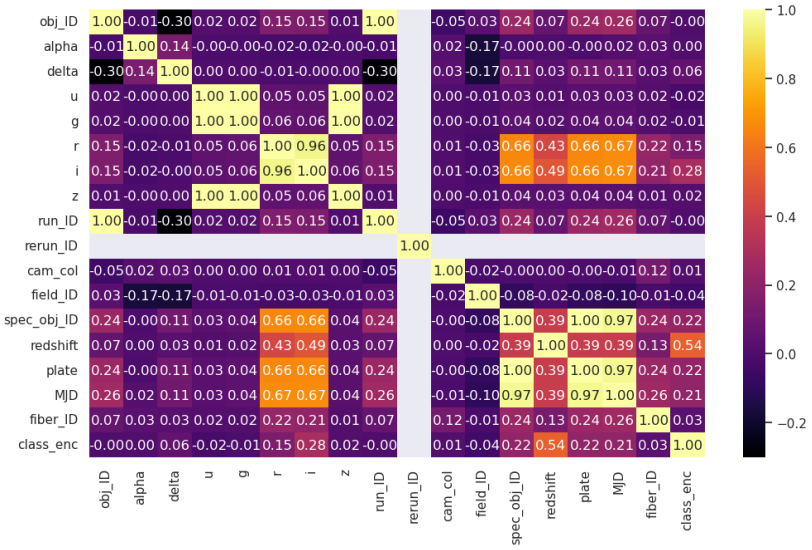
processes performed by me for the EDA using PySpark RDD as the primary source of operations and ocassionally pandas data frame are:-

- Class distribution visualization to check for null values and class bias/imbalances. This is an innate problem if the dataset as majority of the observed celestial objects are part of a galaxy and mitigating this can only be done by gathering more data as under sampling ends up reducing the correlation and supersampling would mean overfitting each class and may cause issue down the line.



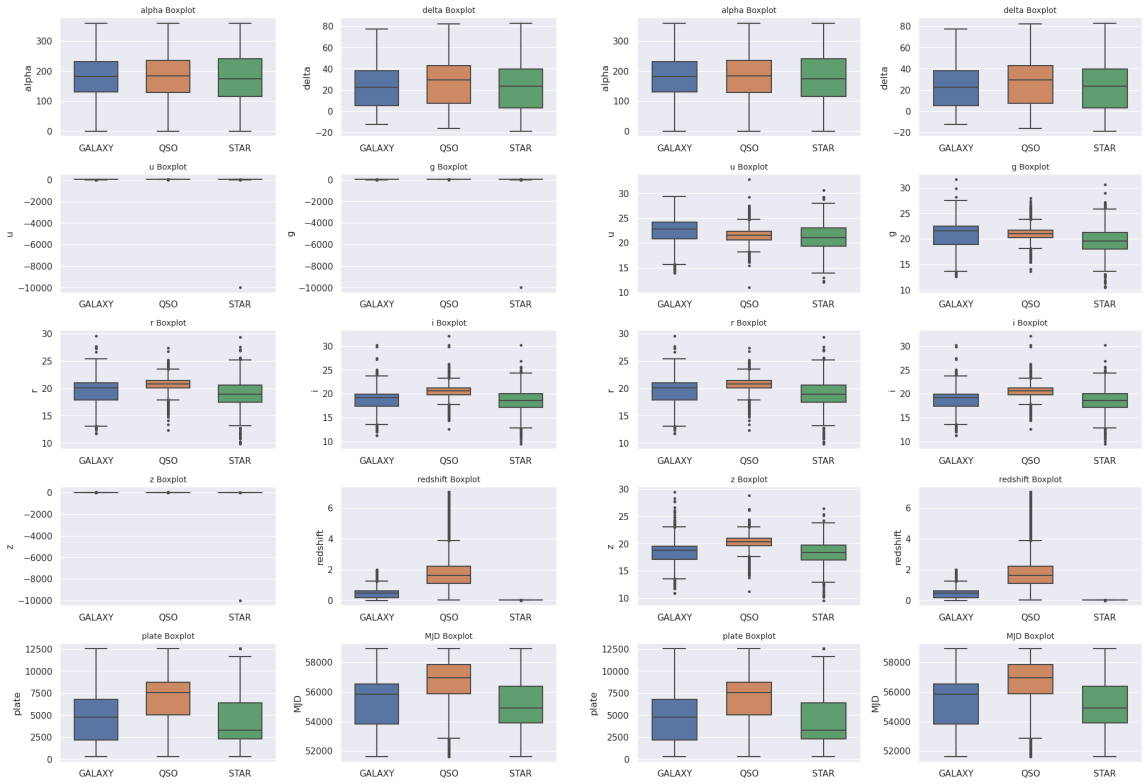
Class distribution visualization

- Corelation analysis of the different features wrt the target classes to decide on which features can be used to obtain better results.



Correlation Heatmap

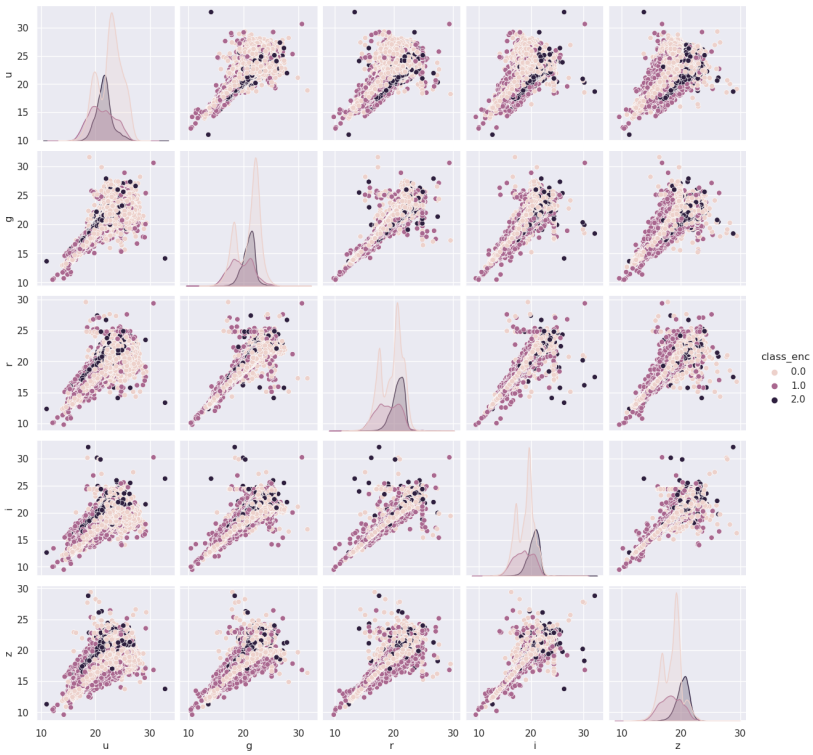
- Outlier identification using box plots of the features by converting the spark DF to pandas DF temporarily for ease of plotting.



Data Cleaning and outlier removal

In [ ]:
```
plot_features = ['alpha', 'delta', 'u', 'g', 'r', 'i', 'z', 'redshift', 'plate', 'MJD']
class_map = {0: 'GALAXY', 2: 'QSO', 1: 'STAR'}
plt.figure(figsize=(10, 30))
for (_, col) in zip(range(len(plot_features)), plot_features):
    plt.subplot(len(Meaningful_cols), 2, _+1)
    sns.boxplot(x=PlotDF['class_enc'].map(class_map), y=col, data=PlotDF,
                width=0.6, flierprops={'marker': '.'})
    plt.title(label='{} Boxplot'.format(col), fontsize=10)
    plt.xlabel(xlabel=None)
    _ += 1
plt.tight_layout()
plt.show()
```

- Pairplots to recognize the type of correalation and decide which columns to keep or remove which gives use the final set of features and for the Data frame.



Key Feature Pairplots

- In the end I chose the following features as they showed strong positive correlations with the target - ['u', 'g', 'r', 'i', 'z', 'redshift'].

Machine learning

- After the data cleaning phase I went ahead with the creation of the machine learning pipelines what usually involved the pyspark feature assembly using the VectorAssembler and our choice of model, for which I chose an array of 6 Models overall to train and test the data.
- The sample of train and test data is given in below image.

```
training_data.show(5)
testing_data.show(5)
```

```
+--------+--------+--------+--------+--------+------------+-----+
|       u|       g|       r|       i|       z|    redshift|label|
+--------+--------+--------+--------+--------+------------+-----+
| 12.2624|10.51139|10.06854|13.41786|10.22551| 2.498567E-5|  1.0|
|14.15199|10.73097| 9.82207|9.469903|9.612333| 5.092877E-5|  1.0|
|14.50678|12.67902|11.74664|11.29956|10.91847| 0.007589578|  0.0|
|14.56906|13.06488|12.52759| 12.5511|12.32606|-1.597281E-4|  1.0|
|14.64617|15.04207|15.46415|15.75953|16.00825|-1.741084E-4|  1.0|
+--------+--------+--------+--------+--------+------------+-----+
only showing top 5 rows

+--------+--------+--------+--------+--------+------------+-----+
|       u|       g|       r|       i|       z|    redshift|label|
+--------+--------+--------+--------+--------+------------+-----+
|14.26381|12.03599|11.64166|11.51527|12.55128| 1.579277E-4|  1.0|
|14.64972|12.63574| 12.1434|11.96598|13.13046| 2.895463E-4|  1.0|
|14.92536| 13.5602|13.10489|12.83076|12.64798| 0.006129465|  0.0|
|14.93182| 14.0644|13.96133|14.13364| 14.2673|-2.679997E-4|  1.0|
|15.38325|14.37224|13.87742|13.62063|13.30364|  0.01931902|  0.0|
+--------+--------+--------+--------+--------+------------+-----+
only showing top 5 rows
```

Sample Data being fed to the Machine learning models

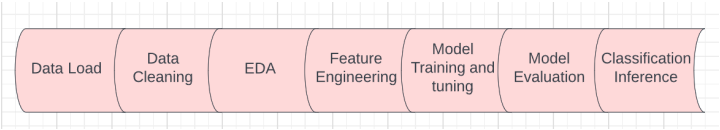
In [ ]:
```
(training_data, testing_data) = Star_MLDF.randomSplit([0.7, 0.3], seed=42)

# Assemble features into a single column
FeatAssm = VectorAssembler(inputCols=Star_MLDF.columns[:-1], outputCol="features")

# Standard Normalize the features
StdScale = StandardScaler(inputCol="features", outputCol="scaledFeatures")

# MinMax Normalize the features
MinMax = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")

layers = [len(Star_MLDF.columns[:-1]),32,16,8,4,3]

# Create logistic regression model
lr = LogisticRegression(featuresCol="scaledFeatures", labelCol="label")
dt = DecisionTreeClassifier(featuresCol="features", labelCol="label")
rf = RandomForestClassifier(featuresCol="features", labelCol="label")
ovr = OneVsRest(featuresCol='scaledFeatures', labelCol = 'label',classifier=lr)
nb = NaiveBayes(featuresCol="scaledFeatures", labelCol="label", modelType="gaussian")
mlp = MultilayerPerceptronClassifier(layers=layers,blockSize=256,seed=42)


#Creating the pipeline for ML Execution for
#all models with varied operations based on model requirements
pipeline_lr = Pipeline(stages=[FeatAssm, StdScale, lr])
pipeline_dt = Pipeline(stages=[FeatAssm, dt])
pipeline_rf = Pipeline(stages=[FeatAssm, rf])
pipeline_ovr = Pipeline(stages=[FeatAssm, StdScale, ovr])
pipeline_nb = Pipeline(stages=[FeatAssm,MinMax,nb])
pipeline_mlp = Pipeline(stages=[FeatAssm, mlp])
```

- On a whole I follow a simple pipeline that involves the feature assembly, scaling if necessary, parametergrid and crossvalidator for parameter tuning ,then training the model and testing it and storing the performance metrics.

In [ ]:
```
# Logistic regression tuning
paramGrid_lr = ParamGridBuilder() \
    .addGrid(lr.maxIter, [100, 150]) \
    .addGrid(lr.regParam, [0.01, 0.1, 1.0]) \
    .addGrid(lr.elasticNetParam, [0.5, 1.0]) \
    .build()

AccEval = MulticlassClassificationEvaluator(labelCol="label",
          predictionCol="prediction", metricName="accuracy")

# Create cross-validator
crossval_lr = CrossValidator(estimator=pipeline_lr,
                          estimatorParamMaps=paramGrid_lr,
                          evaluator=AccEval,
                          numFolds=5)

# Decision tree tuning
paramGrid_dt = ParamGridBuilder() \
             .addGrid(dt.maxDepth, [ 10, 20, 30]) \
             .addGrid(dt.impurity, ['gini', 'entropy']) \
             .build()

# Create 5-fold CrossValidator
```

```
crossval_dt = CrossValidator(estimator = pipeline_dt,
                      estimatorParamMaps = paramGrid_dt,
                      evaluator = AccEval,
                      numFolds = 5)

# Random Forest tuning
paramGrid_rf = ParamGridBuilder() \
            .addGrid(rf.numTrees, [50, 100]) \
            .addGrid(rf.maxDepth, [5, 10, 15]) \
            .build()

# Create cross-validator
crossval_rf = CrossValidator(estimator=pipeline_rf,
                         estimatorParamMaps=paramGrid_rf,
                         evaluator=AccEval,
                         numFolds=5)

# Naive Bayes tuning
paramGrid_nb = ParamGridBuilder() \
            .addGrid(nb.smoothing, [0.1,0.5,1.0,1.5,2.0,2.5]) \
            .build()

# Create cross-validator
crossval_nb = CrossValidator(estimator=pipeline_nb,
                         estimatorParamMaps=paramGrid_nb,
                         evaluator=AccEval,
                         numFolds=5)

predictions_lr = model_lr.transform(testing_data)
print("|----------Logistic Regression Model prediction complete------------|")
predictions_dt = model_dt.transform(testing_data)
print("|----------Decision Tree Model prediction complete------------|")
predictions_rf = model_rf.transform(testing_data)
print("|----------Random Forest Model prediction complete------------|")
predictions_ovr = model_ovr.transform(testing_data)
print("|----------One vs Rest using LogR Model prediction complete------------|")
predictions_nb = model_nb.transform(testing_data)
print("|----------Naive Bayes (Gaussian) prediction complete------------|")
predictions_mlp = model_mlp.transform(testing_data)
print("|----------MLP Model prediction complete------------|")
print("\n\n")

# Evaluate the model using Multi class evaluater
MCeval = MulticlassClassificationEvaluator(labelCol="label",
         predictionCol="prediction", metricName="accuracy")


LR_acc = MCeval.evaluate(predictions_lr)
DT_acc = MCeval.evaluate(predictions_dt)
RF_acc = MCeval.evaluate(predictions_rf)
OVR_acc = MCeval.evaluate(predictions_ovr)
NB_acc = MCeval.evaluate(predictions_nb)
MLP_acc = MCeval.evaluate(predictions_mlp)

print("LogisticRegModel Accuracy = {:.2f}%".format(LR_acc * 100))
print("DecisionTreeModel Accuracy = {:.2f}%".format(DT_acc * 100))
print("RandomForestModel Accuracy = {:.2f}%".format(RF_acc * 100))
print("OneVRest Accuracy = {:.2f}%".format(OVR_acc * 100))
print("NaiveBayes Accuracy = {:.2f}%".format(NB_acc * 100))
print("MultiLayerPerceptron Accuracy = {:.2f}%".format(MLP_acc * 100))
```

- As can be seen above I have chose the following 6 classifiers 'Logistic Regression', 'Random Forest', 'Decision Tree','One Vs Rest (LogReg)', 'Naive Bayes', 'Multi Layer Perceptron' and the 'Logistic Regression', 'Random Forest', 'Decision Tree' and 'Naive Bayes' are parameter optimized and running 5 folds.
- The execution times of each model is also displayed and gives us a sense of load for training the model and the effectiveness of hardware. Logistic Regression Model trained in 141.70 seconds, Decision Tree Model trained in 48.88 seconds, Random Forest Model trained in 408.39 seconds, One vs Rest using LogR Model trained in 5.79 seconds, Naive Bayes Model trained in 15.05 seconds, MLP Model trained in 11.40 seconds in the J2 instance.
- The Workflow pipepline for the project operation is provided below.



Workflow pipeline followed

Results

The base models perform nearly similarly, yielding very close accuracies and prediction metrics however MLP significantly suffers from data imbalance. The Fig below provides the consolidated Model performances.

Model	F1 Score	Precision	Recall	Accuracy
Logistic Regression	0.93	0.97	0.97	92.62
Decision Tree	0.96	0.97	0.97	96.43
Random Forest	0.97	0.97	0.98	97.01
One Vs Rest (LR)	0.95	0.95	0.96	95.07
Naive Bayes	0.92	0.96	0.91	92.08
Multi Layer Perceptron	0.91	0.83	0.91	80.46

Consolidated Model performance

This isnt enough to estimate the performance and effect of models and the data's innate properties. To understand the same we plot the F1 metrics for each label classification as below.

F1 Model metrics

From this matrix we can clearly understand that there is a heavy false positive rate for the GALAXY class and this is especially true for the MLP which basically tries to capture the essence of data and since the class imbalance clearly leans towards GALAXY most of the models also follow somewhat in the same direction. Overall in terms of sheer accuracy we have LogisticRegModel Accuracy = 92.62%, DecisionTreeModel Accuracy = 96.43%, RandomForestModel Accuracy = 97.01%, OneVRest Accuracy = 95.07%, NaiveBayes Accuracy = 92.08%, MultiLayerPerceptron Accuracy = 80.46%.

We can say that Random forest seems to perform the best here at the same time it has the highes execution time due for the given hyoer paramerter tuning i.e. 408.39 seconds which might not seem much by itself but relatively slightly more than 3X the time required to run the next most intensive classifier and nearly 80X the classifier with lowest runtime and the relative improvement in performance wrt the same are 4.39% and 4.93% (difference).

Discussion

Continuing the discussion from results, we can say that the most efficient out of all the models tried seems to be optimized run of Decision tree algorithm, the key points that make it a better choice are that it doesn't need feature scaling as it learns the thresholds for the tree traversal, and the run time is highly optimized wrt the accuracy and performance observed, additionally it is relatively difficult to saturate the decision tree algorithm when compared to heavy models like MLP which suffer significantly from lack of really large amounts of data which also has lesser class bias. In the consolidated model result we can see that essentially the Precision and recall of Decision tree is nearly the same and Random forest that makes it a good baseline model for estimation, with the baseline established We can improve the pipeline as the data velocity and volume increases as we add in new data points to the RDD thus allowing for smoother addressing the 3 Vs of Big data.

Now, Moving to the holistic view of the project and to correlate with the various skills I have learnt over the coursework, Going all the way back to the initial module of "Lifecycles and Pipelines" this project allows me to apply in practice the abilities to effectively implement a Pipeline and Schema for the

Classification task at hand while keeping in mind the scope of future updates to the data. This is reflected in the project as nearly 1 click execution. The modules of "Processing and Analytics" allowed me to learn regarding working with virtual machines like J2 allowing me to perform the project independently on J2 instance without any hitches other that the initial decisions to check which hardware config was most suited for the project.

The module of "Processing and Analytics" also introduce me to the Spark framework which was a tremendously useful resource during the project for "Ingesting and Storing" heterogenous data from the dataset and access in any format I wanted by using various Map and Reduce techniques like:

In [ ]:
```
Unique_vals = Star_rawDF.select('class').distinct().rdd.flatMap(lambda x: x).collect()

Star_rawDF.select(*(sum(col(c).isNull().cast("int")).alias(c) for c in Star_rawDF.columns)).s

ClassbiasDict = {key:Star_rawDF.filter(col("class") == key).count() for key in Unique_vals}
```

These made my life easy as they executed powerful commands quickly due to their distributed data procecssing capabilities. But it doesn't stop their, their extensive libraries supported all of the commonly needed ML and data processing functions which i utilized at various stages in the pipline which allowed me to maintain system consistency with data format. With the option to expand into noSQL data storage linking like MongoDB allows me to future proof the schema updation in case I decide to migrate to them.

Furthermore, The project can also independently be deployed onto GCP or AWS environment with a bit of tweaking and automation or it could be parallelized and run on VMs/containers for low level hive-like data management or on a completely cloud based operation which makes this project scope rather huge. I personally aimed this to be completely autonomous however that is not yet possible due to certain functionalities like Outlier estimation, Bias mitigation strategy decision etc, that require human decisions to be executed. I also understood the usage of SQL like and often SQL queries that can be run on to of PySpark which makes it such an appealing platform for automated pipelines and I am sure i would love to use this often and again in the near future and also in industrial settings.

As such due to the well preparedness from the coursework and engagements from my fellow course colleagues, I faced much lesser amount of turbulance wrt any stage however there were dead ends that I encountered especially while testing Multiclass classifiers from pyspark. I wanted to also tryout the XGboosted tree, GradientBoosted tree and SVMs for the classification but they were note really conducive wrt the application with multiclass classification and lacked usage and resources for debugging. And debugging any of th errors went on a total wormholes due to the long and esoteric errors pyspark enforces and also the initial setup of pyspark without the lib findspark() was nearly a nightmare.

I would like to take this opportunity to thank all you all Professors and TAs for making such a well rounded and diverse course that allowed me to expand my tech and Data science skill set. I would carry all these skills forward with the eagerness to learn further in a Hands-On setting whenever the opportunity presents.

Conclusion

- We can clearly see the performances of the various classifiers and we can clearly see that due to class imbalance we see a lot of false positives for the GALAXY labels especially MLP which depends on the Data balance for its weight training.
- In general the basic classifiers seem to outperform the more computationally heavy models especially Tree algorithms that also don't require input scaling seem to perform the best overall.
- To improve upon this result the primary objective would be to obtain more data to allow for a better prediction across labels.
- The ability of PySpark parallel processing expedited the process of the prediction exponentially as compared to regular execution which can be seen from the execution times for the models in the code.
- This concludes the process of Big Data processing and Classification of Stellar Classification Dataset using Pyspark Distributed processing.
- The Performance can be further tweaked using hyper parameters but more achievable results will be produced using a more capable environment and using Deep learning using GPU.
- To summarize, in this project I have **used the J2 environment** to perform **EDA on the Stellar Dataset**, device a **Pipeline for Machine learning and Classification** for the **csv based heterogenous data** using **Distributed processing via PySpark**.
- Additional improvements can be done wrt the usage of NoSQL DB for streaming the data directly on Virtual machines.

Submission files

- I will be submitting the Jupyter notebook and the PDF execution of the notebook as the project submission files along with this report.

References

[1] Abdurro'uf et al., The Seventeenth data release of the Sloan Digital Sky Surveys: Complete Release of MaNGA, MaStar and APOGEE-2 DATA (Abdurro'uf et al. submitted to ApJS) [arXiv:2112.02026]

[2] fedesoriano. (January 2022). Stellar Classification Dataset - SDSS17. Retrieved [Date Retrieved] from https://www.kaggle.com/fedesoriano/stellar-classification-dataset-sdss17

[3] https://spark.apache.org/docs/latest/ml-features

[4] https://spark.apache.org/docs/latest/ml-pipeline.html

[5] https://spark.apache.org/docs/latest/ml-classification-regression.html

[6] https://spark.apache.org/docs/latest/ml-tuning.html

[7] https://matplotlib.org/3.1.1/gallery/statistics/boxplot.html

[8] https://www.geeksforgeeks.org/python-seaborn-pairplot-method/

[9] https://likegeeks.com/seaborn-histplot/

[10]
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.evaluation.MulticlassClassificatior

[11]
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.mllib.evaluation.MulticlassMetrics.ht

[12] https://www.kaggle.com/code/waleedfaheem/stellar-classification-and-supervised-learning

In [ ]: