STUDENT PERFORMANCE ANALYSIS

PROGRAMMING CONCEPT & PRACTICE

ADUAYE-ODIETE, JOSHUA

Table of Contents

Introduction

In recent years, higher education institutions have become more concerned about identifying characteristics that contribute to improved student performance.
(Zeineddine). The ability to predict the likelihood of a student dropping out of a course (Zeineddine), early detection of student's that are at risk and properly allocating courses or resources (Lubna) have been some of the focus.
Through business intelligence and machine learning, predictive analysis aims to make predictions from historical data that can be used by businesses to make informed, data-driven decisions (Evermann).

Problem Analysis

The SHU Data Analytics team wants to explore different programming and analytics techniques to analyse and evaluate student performance with the aim of building a predictive model that can be used to make informed decisions.
The data provided by the team for the purpose of this analysis consist of two datasets (student-mat.csv) which contains student performance records for Mathematics subject and (student-por.csv) for Portuguese subject. The datasets are stored in csv files and would need to be retrieved and read into an appropriate data structure to which python pandas' library would be used(**pandas**). The data would be explored and visualized using pythons matplotlib library which can be used to create sophisticated graphs and colour maps(**matplotlib**) and seaborn which is based on matplotlib and produces a higher-level informative statistical graphics(**seaborn**).

Furthermore, some statistical functions would be required in other to carry out descriptive statistics on the dataset using a custom module (**statistic_module**).

Finally, a predictive model would be implemented using regression and classification.

Implementation

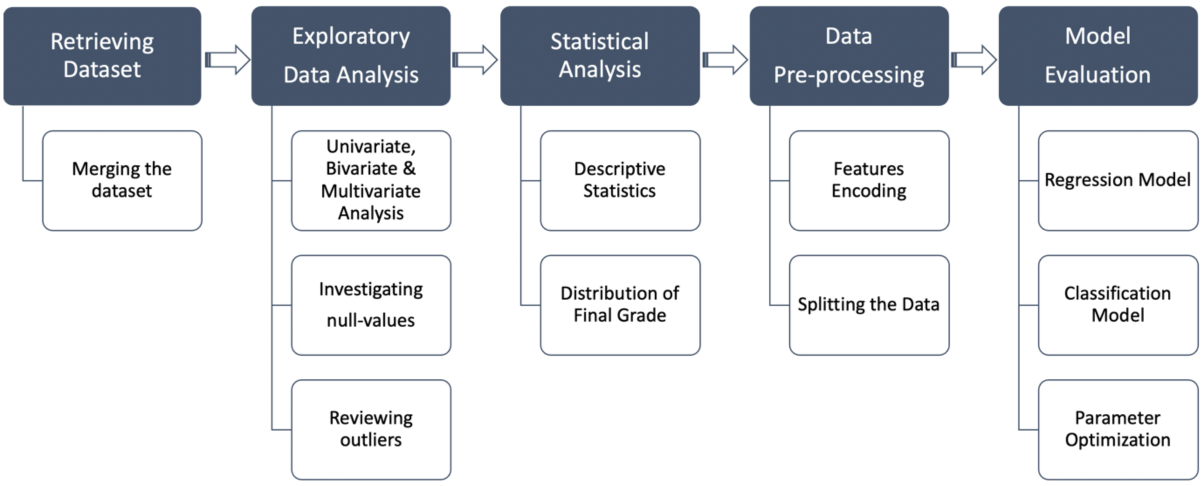


Fig 1.0 Implementation Process

The implementation would be carried out in three sections as shown in Fig 1.0:

Exploratory Data Analysis (EDA):

The dataset was retrieved and merged using pandas and investigated for missing values. Univariate, bivariate and multivariate analysis was carried out on the variables. Then the imported statistical module is used to carry out descriptive statistics on the merged dataset.

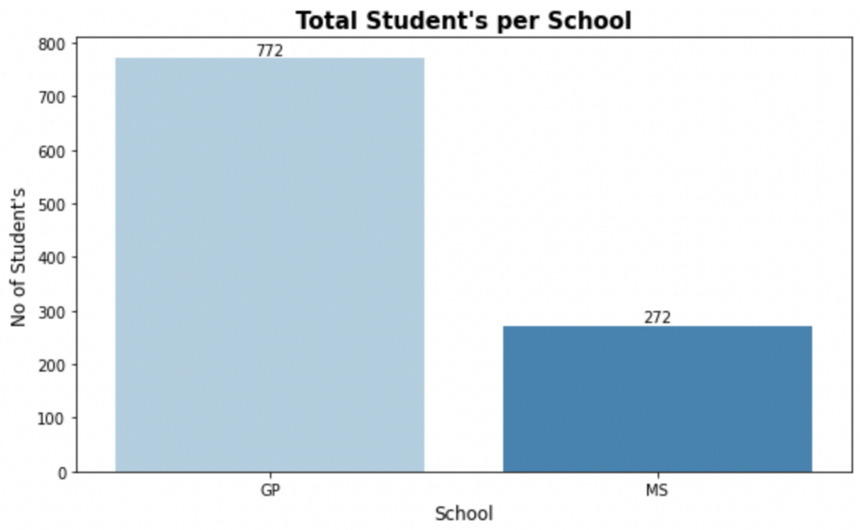


Fig 2.0 Total students per school

It was observed that about 73.9% of the students records where from Gabriel Pereira(GP) school, while students from Mousinho da Silveira(MS) accounted for 26.1% respectively.

The Final Grade(G3) has a negatively skewed distribution which is nearly normal. The grades are more spread out about the mean grade 11.34 with majority falling within the grade band 10 – 17 and a few higher than 17. However, it is interesting to note that the mean G3 is slightly higher than the median 11, which could be as a result of the influence of the 53 zero-grades on the distribution as shown in Fig 3.0.

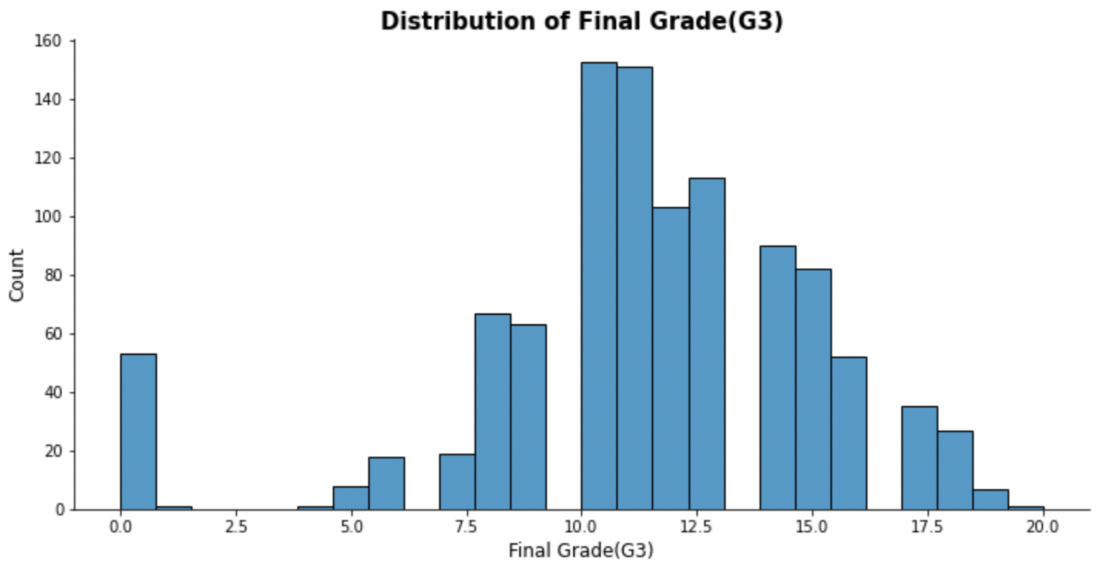


Fig 3.0 Distribution of final grade.

G1 and G2 is observed to have a very high positive correlation with the final grade G3 this means that the first and second period grades greatly influences the outcome of the final grade. Although the Medu, Studytime and Father's education(Fedu) are also positively correlated with the final grade G3. However, number of past class failures has a negative correlation with the final grade thus, it would mean that as the number of past failures increases, the final grade would also tend to decrease as shown in Fig 4.0

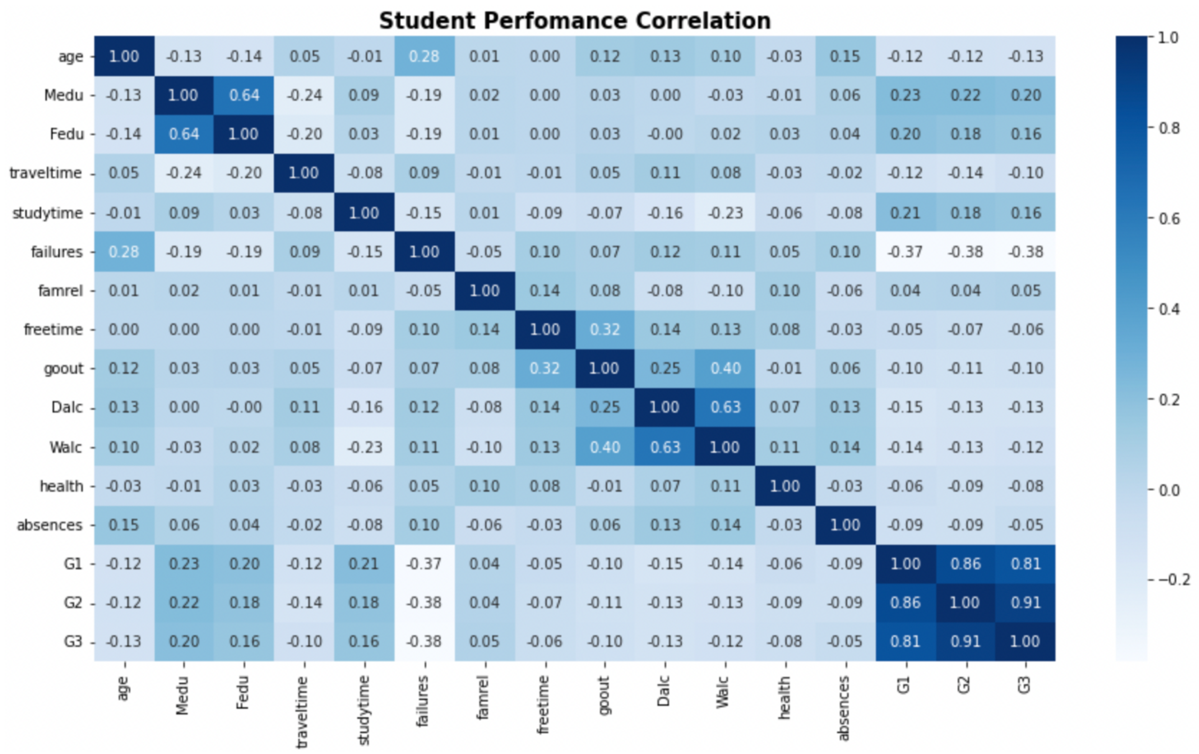


Fig 4.0. Correlation of features with the final grade

Data Pre-processing

Before performing model evaluation, the data is pre-processed by converting categorical variables to numerical variables using LabelEncoder() from scikit learn. Its features are extracted to its **Xclass** and **yclass** and splitted, using 80% of the data for training and 20% for evaluation.

Model Evaluation

Regression

Mean Square Error	Root Mean Square Error	R-Square	Accuracy
2.91	1.71	0.78	0.34

Table 1.0 Estimated R-square and Accuracy

The Random Forest Classifier (RFC) was used for evaluation and 78% of the variation within the data could be explained by the RFC model. Thus, we can be confident with the assumption that the RFC model could be used to predict future values of the final grade(G3). However, it was observed that this model could only predict G3 correctly up to 34%.

The second grade G2 had a larger effect in predicting the final grade in the RFC model as shown in Fig 5.0

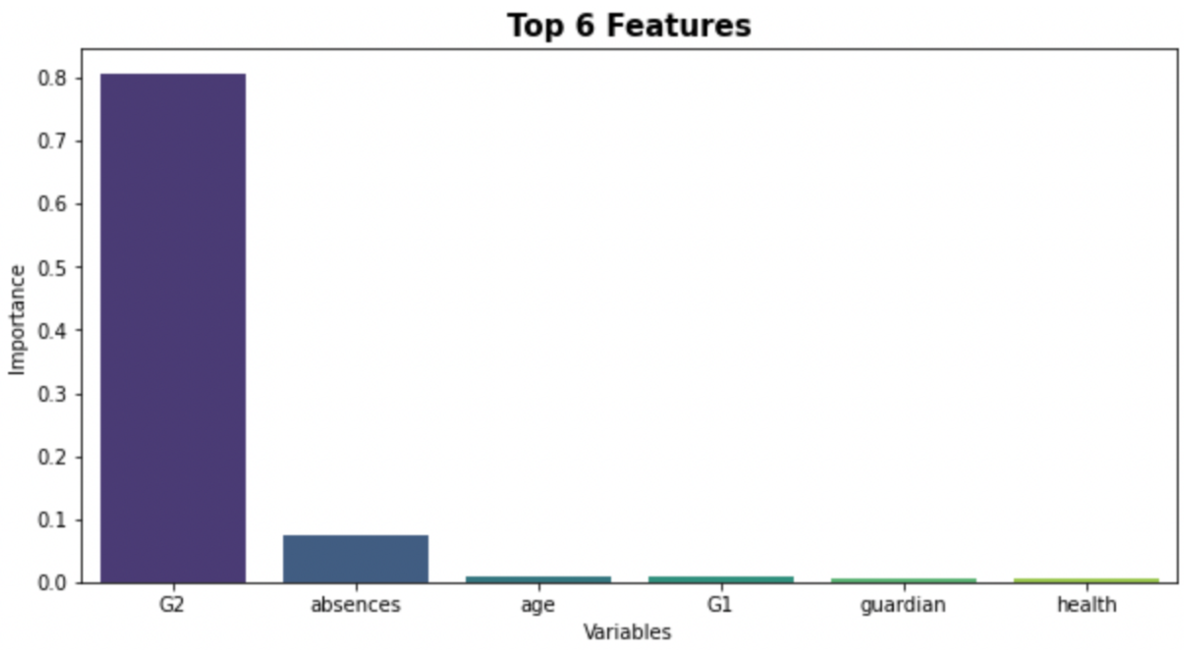


Fig 5.0 Feature Importance plot for top 6 features.

Classification

It was observed that the category of the final grade was imbalanced and there would be a need to handle class imbalance on the datasets before fitting it to a classification model. Three techniques were employed to handle this:

- Random Under sampler
- Random Over sampler
- Smoteenn

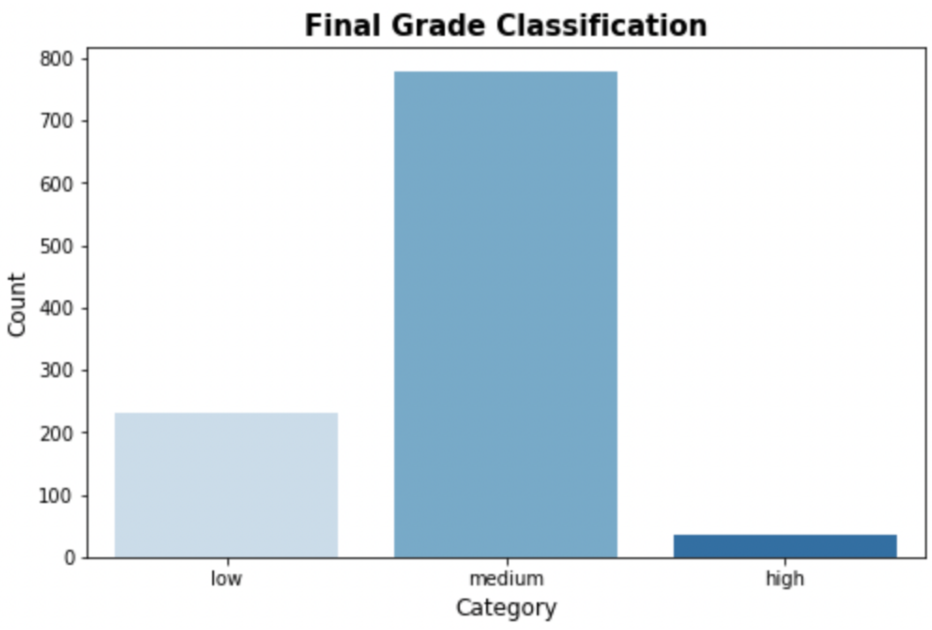


Fig 6.0 Category of final grade(G3)

The aim of handling class imbalance was to allow the training of the model to be unbiased when predicting future final grades.

The model was evaluated with three scikit learn model to which their weighted average F1-score was compared to get the best model.

- Support Vector Machine (SVC)
- Random Forest Classifier (RFC)
- Multi-Layer Perceptron Neural Networks (MLP)

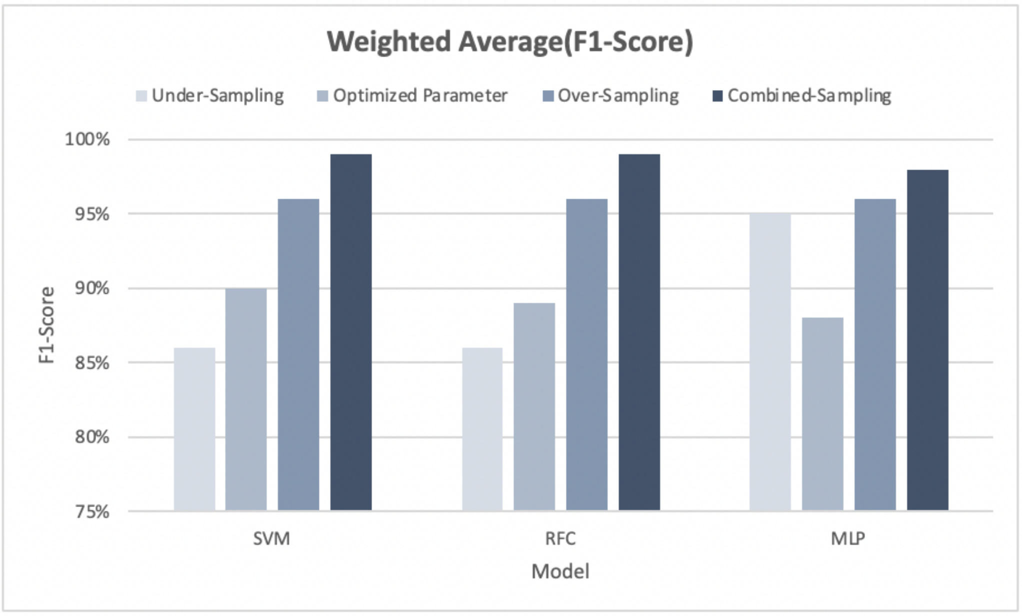


Fig 7.0 Weighted Average F1-Score for classification models.

Also Randomized Search CV was used to optimize the parameters of the model as shown in Table 2.0.

Parameter Optimization

Model	Best Score	Best Parameters
SVC	0.886	kernel= 'linear', C= 1
RFC	0.893	n_estimators= 1000, max_features= auto, criterion = entropy, bootstrap= True
MLP	0.892	solver = 'sgd', learning_rate = 'constant', activation = identity, max_iter= 1800, hidden_layer_sizes = (50,50,50)

Table 2.0 Optimized Parameters

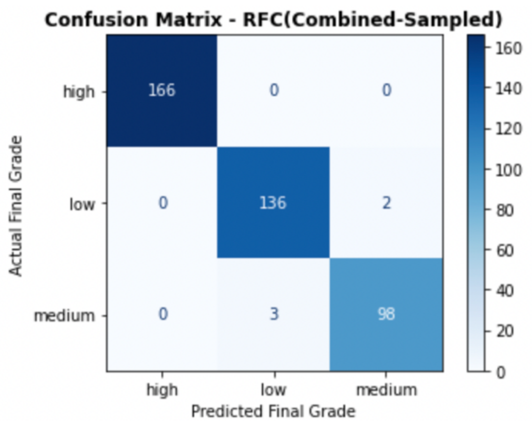
Conclusion

Handling class imbalance using Random Under Sampler yielded an F1-score of 86% for both the SVM and RFC models and 95% for MLP. This technique involved removing samples from the majority class distributions (Medium and Low) to balance the minority class distribution (High). The limitation of this technique could be that the samples drawn from the majority class could be biased and influences future predictions of the models. Using Random Over Sampler, the F1-Score for all three models stood at 96%. The major limitation of this technique is that it could lead to overfitting. Furthermore, using the combined sampling technique SMOTEENN, it was observed that the F1-score for both SVM and RFC stood at 99% and 98% for MLP.

Justification

CLASSIFICATION REPORT RFC- SMOTEENN

	Precision	Recall	F1- Score	Support
high	1.00	1.00	1.00	166
low	0.98	0.99	0.98	138
medium	0.98	0.97	0.98	101
accuracy			0.99	405
macro avg	0.99	0.99	0.99	405
weighted avg	0.99	0.99	0.99	405



The RFC model using combined sampling techniques (SMOTEENN) outperformed the other models, being able to correctly predict **166** high grades from **166** high grades when test. Similarly, it was also able to correctly predict **136** low grades out of **139** lows grades tested and 98 medium grades from **100** medium grades tested. The F1-score which is a weighted harmonic mean of precision and recall stood at **99**%.

Recommendation

The Random Forest Classifier Model using a combine sampling technique (SMOTEEN) is recommended in predicting the final grade of the student performance based on the performance and outcome of its confusion matrix and correlation report.
Precision: The model was able to correctly predict 100% of the student's performance with a high final grade(G3), 98% for a low G3 and 98% for a medium G3.
Recall: 100% of the students' performance grades with a high G3 where correctly identified, 99% for a low G3 and 7% for a medium G3.
F1-Score: 100% of positive predictions for a high final grade were correct, while 98% for a low and 98% for a medium G3 respectively.

Reflection

Exploring the different programming and analytical techniques for exploratory data analysis and predictive modelling has expended my knowledge of python programming language. Been able to write statistical functions using object-oriented programming and carryout model evaluation with python's scikit learn library has made be appreciate the versatility of the python language. I have been able to horn my skills in writing python programs as well as building predictive model using python. I faced a couple of challenges while trying to obtain the feature importance report and optimizing parameters for MLP classifier as the as number of max iterations where exceeded each time the program was ran. Further, I also encountered difficulties handling errors within my statistics module. I would have loved to build a dashboard to explore the variables within the dataset using pythons plotly library.

The pseudocode for the statistics module can be found in the appendix.cs module can be found in the appendix.cs module can be found in the appendix.

Appendix

Pseudocode

```
Define the Statistics Class:
- Initialize objects of the Statistics Class (self and data).
- Handling Exceptions
- Define a method that retrieves column names of numerical and datetime datatypes getColumn.
    o Select all numerical and datetime columns from a dataframe and assign them to a variable num_var.
    o Convert the columns of num_var to a list and assign to a variable column_
    o Output column_

Computing basic statistics
- Define a method that returns the maximum value Maxi
    o Create an empty list maxi
    o Iterate over each column in a dataframe which has numerical or datetime datatype.
    o Get the maximum value for each column and assign them to a variable maxi_
    o Attach the values of maxi_ to the empty list maxi
    o Output the values of maxi as a pandas dataframe whose columns are getColumn

- Define a method that returns the minimum value Mini
    o Create an empty list mini
    o Iterate over each column in a dataframe which has numerical or datetime datatype.
    o Get the minimum value for each column and assign them to a variable mini_
    o Attach the values of mini_ to the empty list mini
    o Output the values of mini as a pandas dataframe whose columns are getColumn

- Define a method that returns the count of all non-null values Count
    o Create an empty list count
    o Iterate over each column in a dataframe which has numerical or datetime datatype.
    o Get the count of each column non-null value and assign them to a variable count__
    o Attach the values of count_ to the empty list count
    o Output the values of count as a pandas dataframe whose columns are getColumn

- Define a method that returns the mean value Mean
    o Create an empty list mean_
    o Iterate over each column in a dataframe which has numerical or datetime datatype.
    o Get the mean value for each column and assign them to a variable calc_mean
    o Attach the values of calc_mean to the empty list mean_
    o Output the values of mean_ as a pandas dataframe whose columns are getColumn


- Define a method that returns the mean value Median
    o Create an empty list median_
    o Iterate over each column in a dataframe which has numerical or datetime datatype.
```

    - Get the median value for each column and assign them to a variable new_median
    - Attach the values of new_median to the empty list median_
    - Output the values of median_ as a pandas dataframe whose columns are **getColumn**

- Define a method that returns the standard deviation value **Standev**
    - Create an empty list standev
    - Iterate over each column in a dataframe which has numerical or datetime datatype.
    - Get the standard deviation value for each column using numpy and assign them to a variable standev_
    - Attach the values of standev_ to the empty list standev
    - Output the values of standev as a pandas dataframe whose columns are **getColumn**

- Define a method that returns the variance value **Variance**
    - Create an empty list variance
    - Iterate over each column in a dataframe which has numerical or datetime datatype.
    - Get the variance value for each column using numpy and assign them to a variable variance_
    - Attach the values of variance_ to the empty list variance
    - Output the values of variance as a pandas dataframe whose columns are **getColumn**

- Define a method that returns the skewness value **Skewness**
    - Create an empty list skewness
    - Iterate over each column in a dataframe which has numerical or datetime datatype.
    - Get the skewness value for each column and assign them to a variable skew_
    - Attach the values of skew_ to the empty list skewness
    - Output the values of skewness as a pandas dataframe whose columns are **getColumn**

- Define a method that returns the kurtosis value **Kurtosis**
    - Create an empty list kurtosis_
    - Iterate over each column in a dataframe which has numerical or datetime datatype.
    - Get the kurtosis value for each column and assign them to a variable kurt
    - Attach the values of kurt to the empty list kurtosis_
    - Output the values of kurtosis_ as a pandas dataframe whose columns are **getColumn**

Computing Percentiles

- Define a method that returns the 25th percentile value **Percent25**
    - Create an empty list percent25
    - Iterate over each column in a dataframe which has numerical or datetime datatype.
    - Get the 25th percentile value for each column using numpy and assign them to a variable percent_25
    - Attach the values of percent_25 to the empty list percent25
    - Output the values of percent25 as a pandas dataframe whose columns are **getColumn**

- Define a method that returns the 50th percentile value **Percent50**
    - Create an empty list percent50
    - Iterate over each column in a dataframe which has numerical or datetime datatype.
    - Get the 50th percentile value for each column using numpy and assign them to a variable percent_50
    - Attach the values of percent_50 to the empty list percent50
    - Output the values of percent50 as a pandas dataframe whose columns are **getColumn**

- Define a method that returns the 75th percentile value **Percent75**
    - Create an empty list percent75
    - Iterate over each column in a dataframe which has numerical or datetime datatype.
    - Get the 75th percentile value for each column using numpy and assign them to a variable percent_75
    - Attach the values of percent_75 to the empty list percent75
    - Output the values of percent75 as a pandas dataframe whose columns are **getColumn**

Computing the descriptive statistics

- Define a method that returns the descriptive statistics value **descStat**
    - Combine methods (Count, Mean, Standev, Variance, Mini, Percent25, Median, Percent 75, Maxi, Skewness and Kurtosis) on its columns using pandas concat method.
    - Attach the values of the combined methods to a variable observe
    - Transpose the values in the variable observe and assign it to a variable stats
    - Output stats as a pandas dataframe.

Reviewing possible errors

- Except AttributeError occurs:
    - Output (AttributeError: Object has no attribute')

- Except NameError occurs,
    - Output ('NameError: A variable isn't defined')

- Except TypeError Occurs:
    - Output ('TypeError: Object is not callable')

- Except KeyError Occurs:
    - Output ('KeyError: Key not available in columns')

- Except SyntaxError Occurs:
    - Output ('SyntaxError: Invalid Syntax')