ABOUT THE DATASET:

Context:

The dataset is made up of data acquired from big Scania vehicles in normal use. The technology in question is the Air Pressure technology (APS), which produces pressurised air for use in a variety of truck tasks like braking and gear changes. The dataset's positive class includes component failures for a specific component of the APS system. The negative class includes trucks that fail for reasons unrelated to the APS. Experts chose a subset of all available data to create the data collection.

Content:

The dataset contains 60,000 cases and 171 variables. Applied ***Supervised Learning*** technique on the dataset which has a category as “Class” which contains a total of 1000 in the positive class and 59000 in the negative class. There are 48,000 cases in the training set and 12,000 cases in the test set. Values missing are indicated by "na" For proprietary reasons, the data's attribute names have been made anonymous. It is made up of histograms with bins with various conditions as well as single numerical counters.

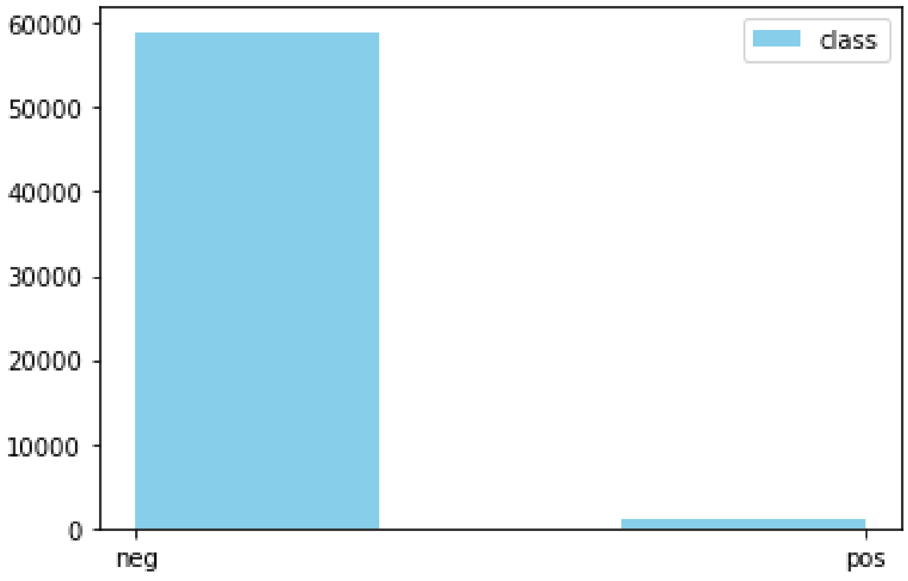
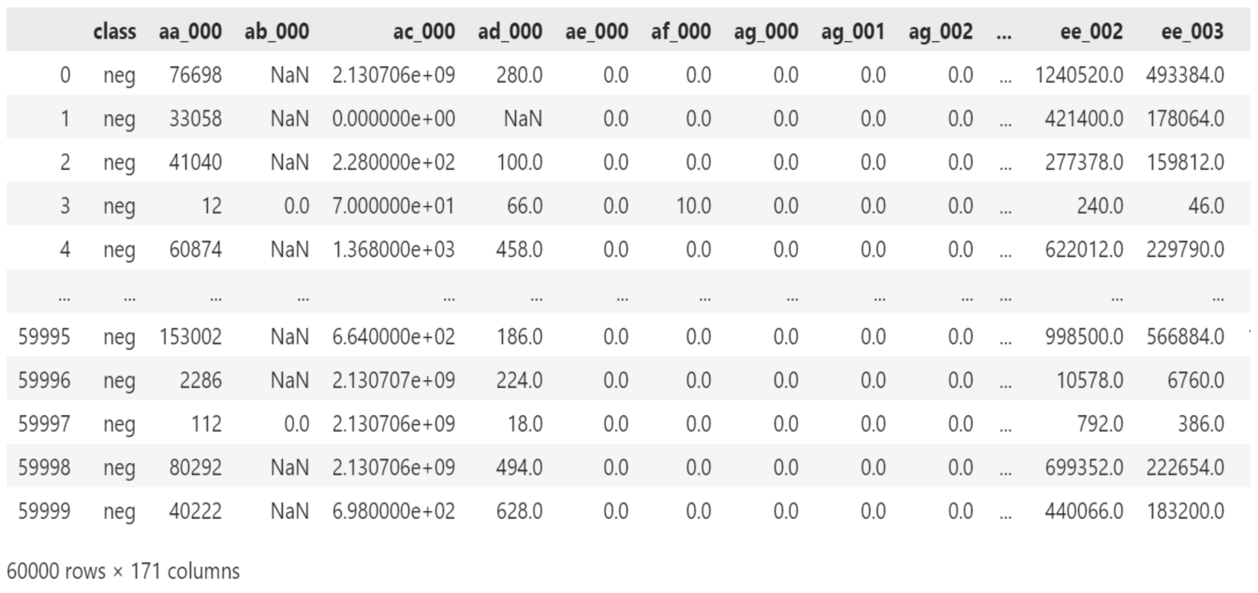


Fig 1: Histogram with number of positive and negative classes.

Data Source:

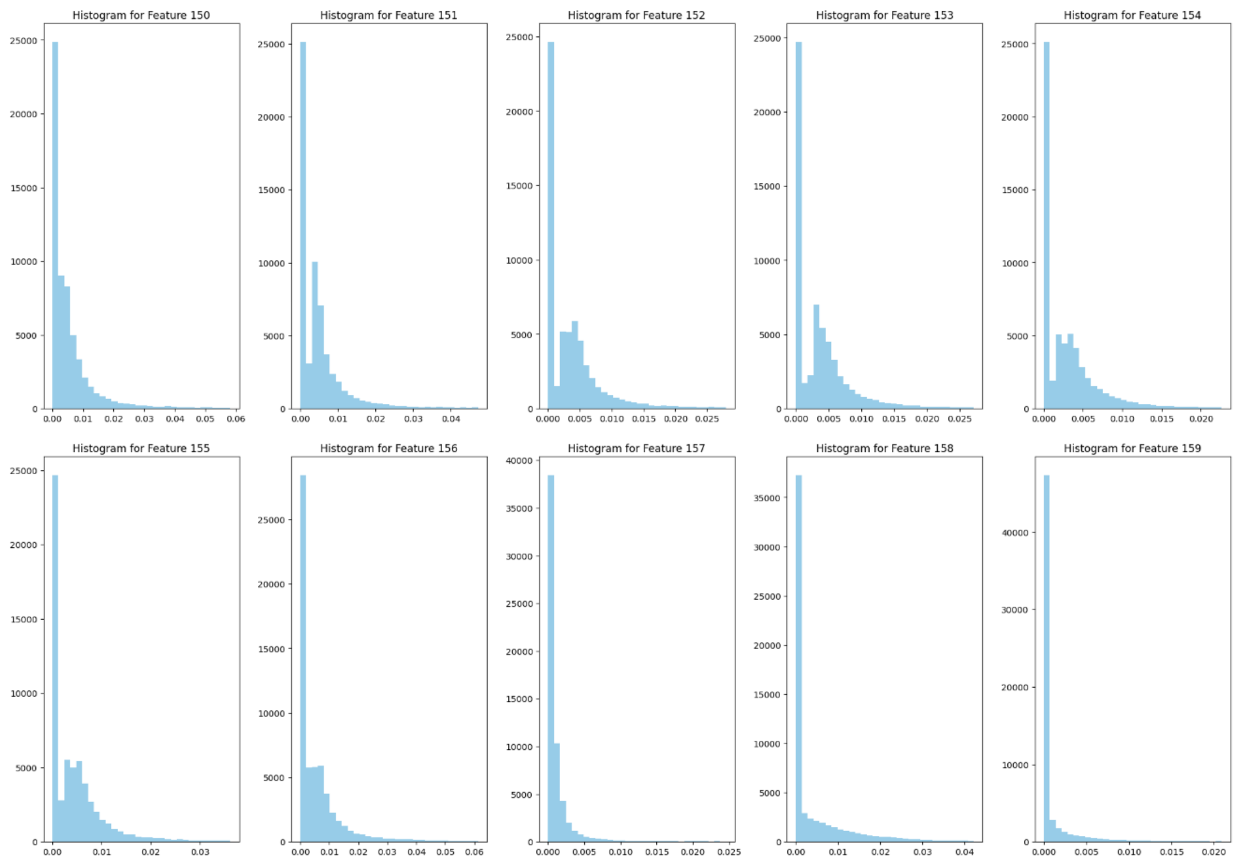
This file is a component of the Scania Trucks APS Failure and Operational Data. It was imported from UCI ML Repository.

Data set:



	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee_002	ee_003
0	neg	76698	NaN	2.130706e+09	280.0	0.0	0.0	0.0	0.0	0.0	...	1240520.0	493384.0
1	neg	33058	NaN	0.000000e+00	NaN	0.0	0.0	0.0	0.0	0.0	...	421400.0	178064.0
2	neg	41040	NaN	2.280000e+02	100.0	0.0	0.0	0.0	0.0	0.0	...	277378.0	159812.0
3	neg	12	0.0	7.000000e+01	66.0	0.0	10.0	0.0	0.0	0.0	...	240.0	46.0
4	neg	60874	NaN	1.368000e+03	458.0	0.0	0.0	0.0	0.0	0.0	...	622012.0	229790.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
59995	neg	153002	NaN	6.640000e+02	186.0	0.0	0.0	0.0	0.0	0.0	...	998500.0	566884.0
59996	neg	2286	NaN	2.130707e+09	224.0	0.0	0.0	0.0	0.0	0.0	...	10578.0	6760.0
59997	neg	112	0.0	2.130706e+09	18.0	0.0	0.0	0.0	0.0	0.0	...	792.0	386.0
59998	neg	80292	NaN	2.130706e+09	494.0	0.0	0.0	0.0	0.0	0.0	...	699352.0	222654.0
59999	neg	40222	NaN	6.980000e+02	628.0	0.0	0.0	0.0	0.0	0.0	...	440066.0	183200.0

60000 rows × 171 columns

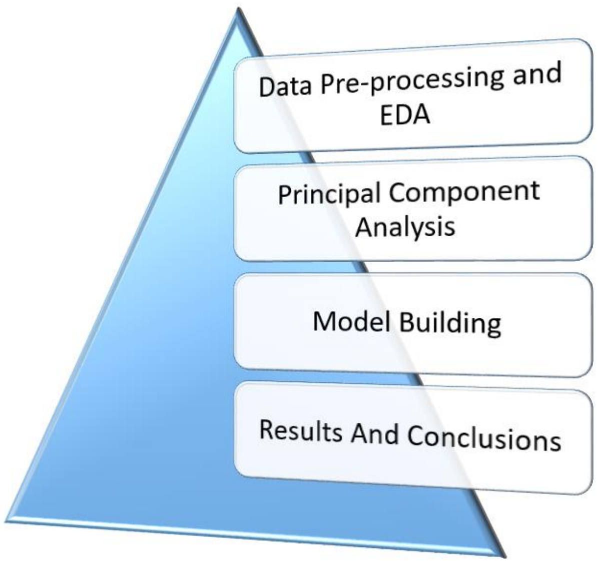


#Histograms drawn to check data distribution

LIBRARIES USED:

- ***Pandas*****:** Helped us in uploading the data and data manipulation.
- ***NumPy:*** Fundamental package for scientific computing in Python.
- ***Matplotlib:*** Versatile plotting library for creating visualisations in Python.
- ***scikit-learn (sklearn):*** Popular machine learning library with a wide range of algorithms.
- ***SciPy:*** Library for advanced mathematical operations and scientific computing.

WORKFLOW OF THE ASSIGNMENT:



DATA PRE-PROCESSING:

1) Checking for Duplicates:

- In this step, we examined the dataset to identify and eliminate any duplicate records as it can distort statistical analyses, compromise the accuracy of insights, and lead to biassed conclusions.
- There were **0** duplicates found in our dataset.

2) Checking for the Missing Values:

- A comprehensive evaluation of missing values was conducted to assess the data completeness in our dataset.
- Out of 170 features, a total of **850,015 missing values** were identified. Remarkably, the category "Class" (y) exhibited no missing values.
- However, the remaining 170 columns contained slightly over 8% missing values collectively.

- Notably, 8 columns were found to have more than 60% missing values (NaNs) and were consequently removed from the dataset. These columns include: "br_000", "bq_000", "bp_000", "bo_000", "ab_000", "cr_000", "bn_000", and "bm_000". Even, after removing these columns, **481,740 entries** were still missing, accounting for approximately 5% of the total dataset.

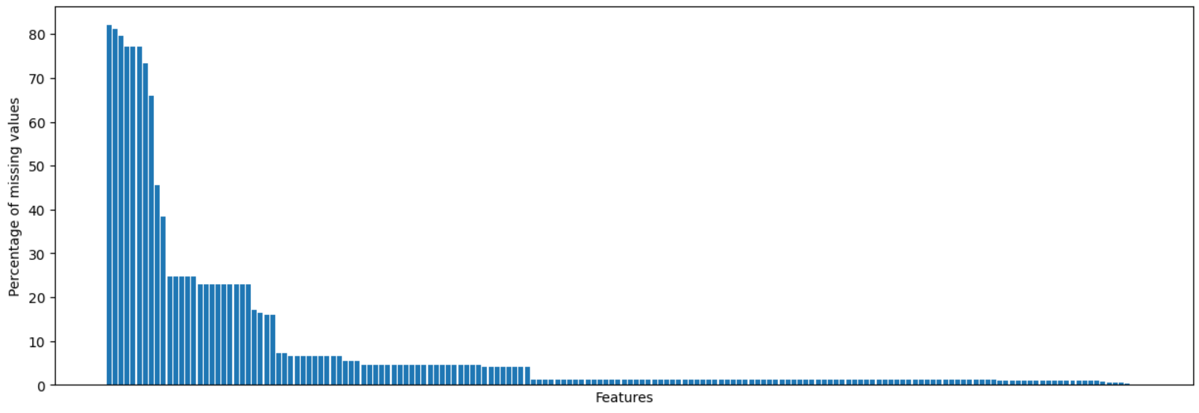


Fig: The following figure shows the percentage of missing values in each Feature.

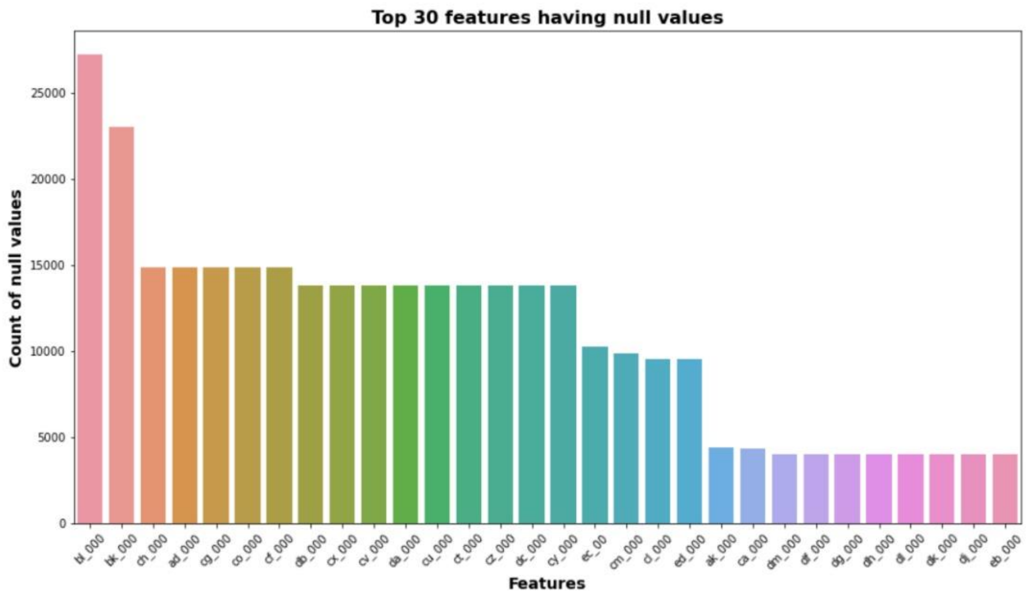


Fig: Top 30 features which are having maximum null values.

3) Calculating the Sparsity of the Dataset:

- The sparsity of the dataset is calculated to quantify the proportion of missing values across all variables. This metric provides valuable insights into the overall data completeness.
- The formula used for calculating the sparsity is:
  sparsity = 1 - count_nonzero(df) / df.size
- The sparsity of the data frame is **0.33.** Even though 33% is quite a high percentage, it is still far from 50%. After that, the Scipy "issparse" function was applied to confirm that the dataset was not sparse.

4) Data Scaling:

To facilitate equitable and insightful comparisons among features exhibiting varying units or magnitudes, we utilize data scaling methodologies. Specifically, we employ the **MinMaxScaler**, a technique that standardizes independent variables to a uniform range of 0 to 1. By implementing this transformation, we ensure that all features harmonize on a common scale, thereby fostering equitable contributions to the learning process of the machine learning model.

5) Dealing with Outliers:

Identification and management of outliers are imperative for robust statistical analyses and model development.

- The upper and lower limits of the dataset were calculated and displayed, showing only the outliers.
- The outliers were then selected and replaced with NaN.
- At this point, the total of missing values increased to 692,782, which now represents approximately 7% of the dataset.

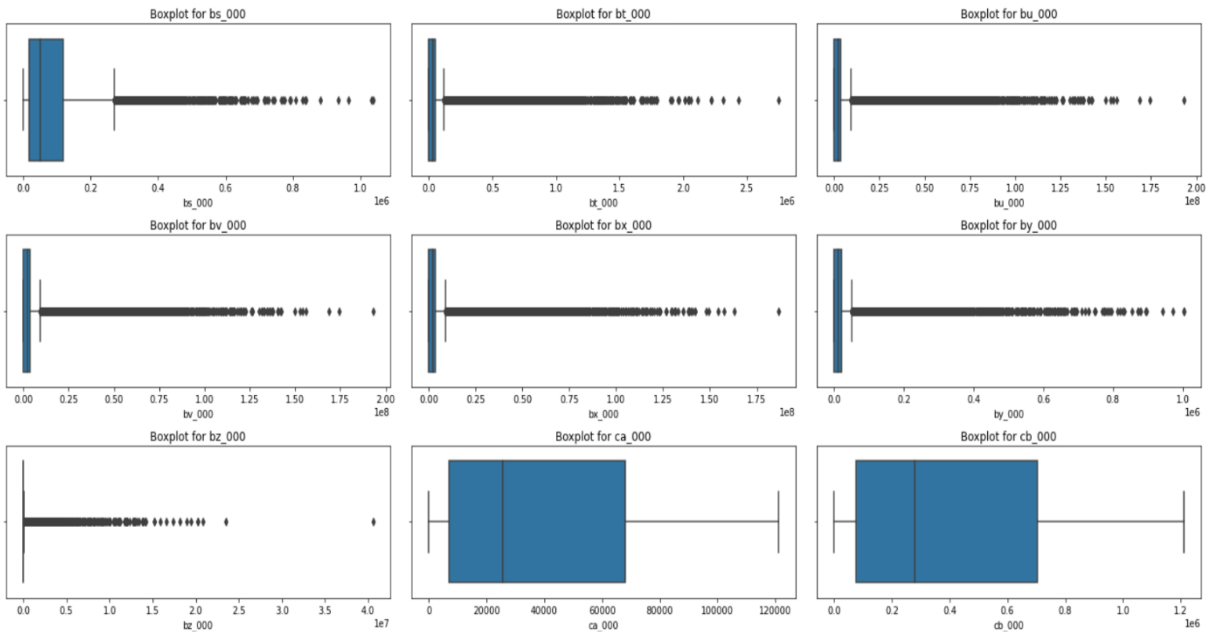


Fig: This boxplot shows the number of outliers in each feature.

6) Data Imputation:

Imputing missing values is a critical step in preserving the integrity of the dataset.

- In our case, missing values will be replaced with the **median** (using the fillna() function) to avoid any interference in the shape of the distribution.
- While the mean is sensitive to every value, the median is not drastically affected by a few extreme values, and because of that, it is known as a resistant measure of centre.

7) Encoding:

- Categorical variables are transformed into a numerical format suitable for machine learning models through encoding.
- Label encoding is applied (using LabelEncoder()), which converts "class" variables into binary, indicating **"neg" as 0** and **"pos" as 1.**

APPLYING PRINCIPAL COMPONENT ANALYSIS:

- Principal Component Analysis (PCA) is a technique used to simplify large datasets. It works by transforming the data into a new set of variables, ideally with **fewer dimensions** than the original set. This results in a more condensed dataset that will deliver the same results because it focuses on its principal components.

- The scaler is activated or called to maintain **98%** of the variability in this case. Then, the dataset—still without the column 'class'—is fitted into the scaler. The plot below illustrates that the **number of components** necessary to achieve the aforementioned variance is **9**.

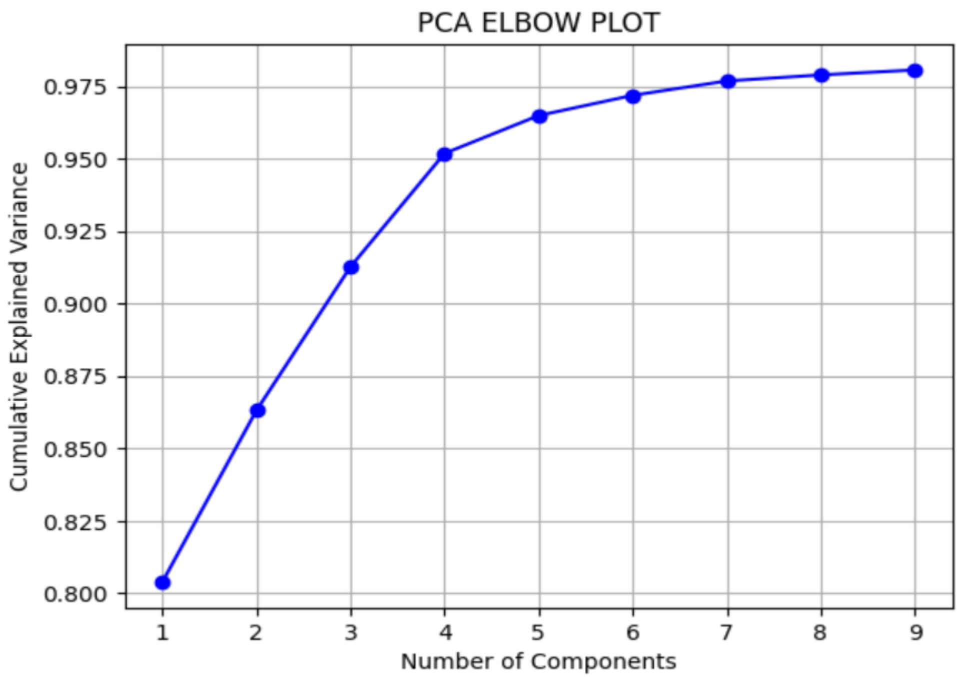


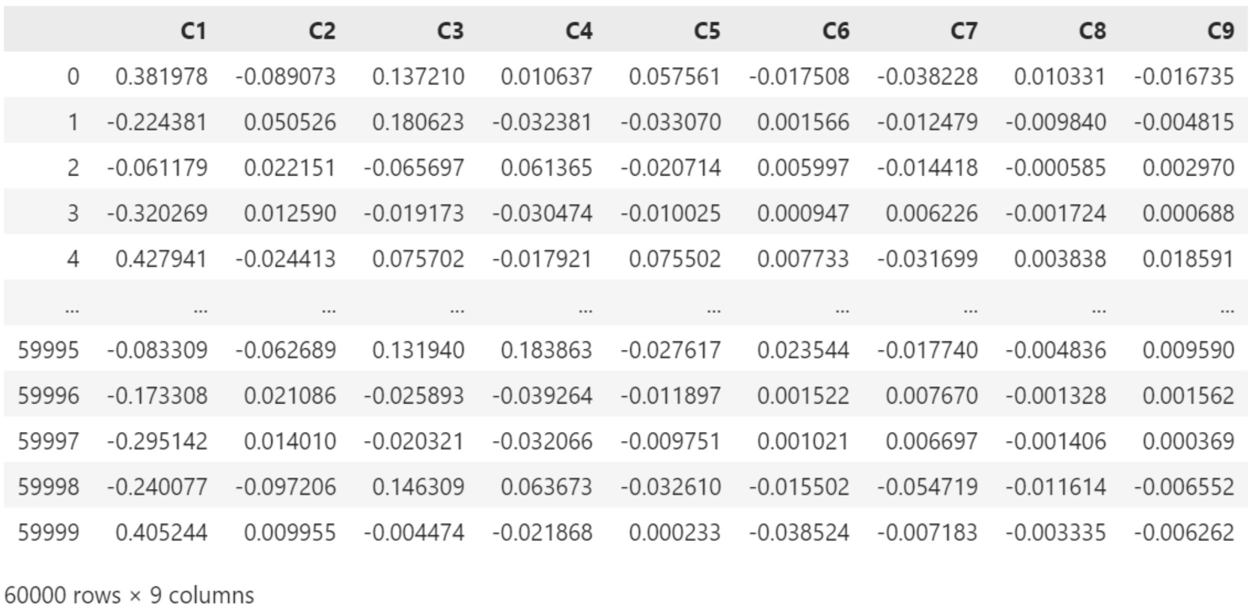
Fig: The scree plot shows 9 features of the data showing 98% of the variability.

The scree plot helps to decide how many principal components to keep. In the plot, the cutoff point for the number of components to keep depends on the specific amount of variability that is to be explained.

- By using the function:

```
pca = PCA(0.98)
X_pca = pca.fit_transform(X)
```

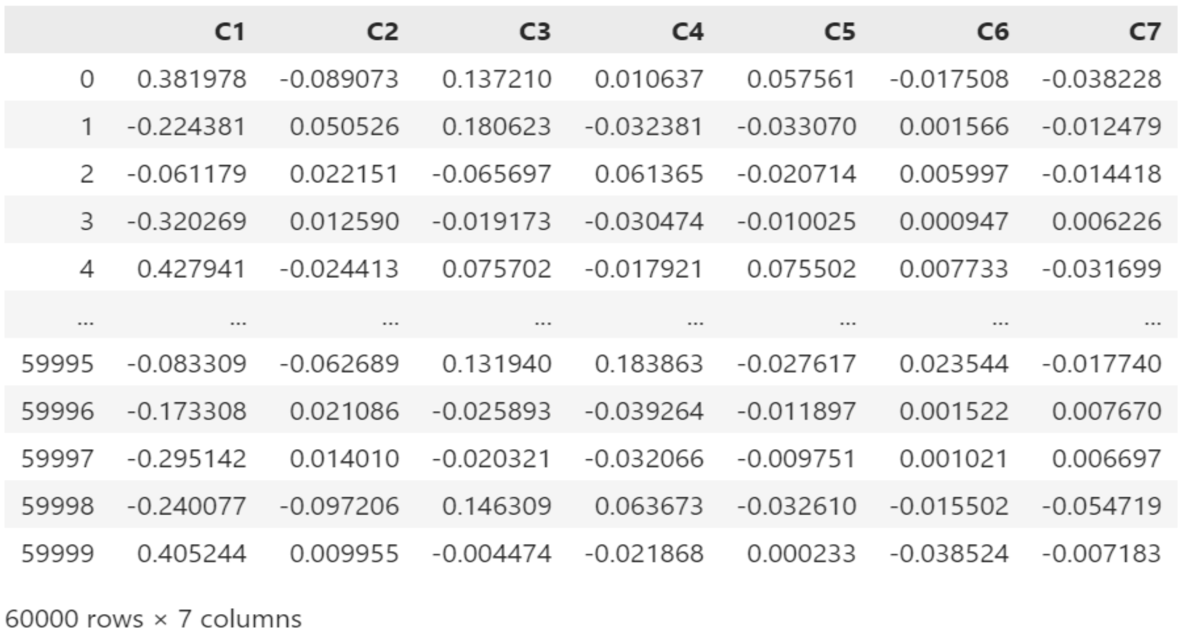
- It was found that (60000,9) of the data explains 98% of variability.
- The variance explained by each of the 9 components is 0.80389628, 0.05919089, 0.04957223, 0.03914245, 0.01306158,0.00698549, 0.00496309, 0.00209328, 0.001708 which corresponds to 98% of the variation.



	C1	C2	C3	C4	C5	C6	C7	C8	C9
0	0.381978	-0.089073	0.137210	0.010637	0.057561	-0.017508	-0.038228	0.010331	-0.016735
1	-0.224381	0.050526	0.180623	-0.032381	-0.033070	0.001566	-0.012479	-0.009840	-0.004815
2	-0.061179	0.022151	-0.065697	0.061365	-0.020714	0.005997	-0.014418	-0.000585	0.002970
3	-0.320269	0.012590	-0.019173	-0.030474	-0.010025	0.000947	0.006226	-0.001724	0.000688
4	0.427941	-0.024413	0.075702	-0.017921	0.075502	0.007733	-0.031699	0.003838	0.018591
...	...	...	...	...	...	...	...	...	...
59995	-0.083309	-0.062689	0.131940	0.183863	-0.027617	0.023544	-0.017740	-0.004836	0.009590
59996	-0.173308	0.021086	-0.025893	-0.039264	-0.011897	0.001522	0.007670	-0.001328	0.001562
59997	-0.295142	0.014010	-0.020321	-0.032066	-0.009751	0.001021	0.006697	-0.001406	0.000369
59998	-0.240077	-0.097206	0.146309	0.063673	-0.032610	-0.015502	-0.054719	-0.011614	-0.006552
59999	0.405244	0.009955	-0.004474	-0.021868	0.000233	-0.038524	-0.007183	-0.003335	-0.006262

60000 rows × 9 columns

We streamlined the initial 9 features down to 8, and subsequently to **7**. Through analysis, it was determined that utilizing 7 principal components preserves approximately 98% (0.97681201) of the essential feature information, demonstrating the effectiveness of this reduction in dimensionality.



	C1	C2	C3	C4	C5	C6	C7
0	0.381978	-0.089073	0.137210	0.010637	0.057561	-0.017508	-0.038228
1	-0.224381	0.050526	0.180623	-0.032381	-0.033070	0.001566	-0.012479
2	-0.061179	0.022151	-0.065697	0.061365	-0.020714	0.005997	-0.014418
3	-0.320269	0.012590	-0.019173	-0.030474	-0.010025	0.000947	0.006226
4	0.427941	-0.024413	0.075702	-0.017921	0.075502	0.007733	-0.031699
...	...	...	...	...	...	...	...
59995	-0.083309	-0.062689	0.131940	0.183863	-0.027617	0.023544	-0.017740
59996	-0.173308	0.021086	-0.025893	-0.039264	-0.011897	0.001522	0.007670
59997	-0.295142	0.014010	-0.020321	-0.032066	-0.009751	0.001021	0.006697
59998	-0.240077	-0.097206	0.146309	0.063673	-0.032610	-0.015502	-0.054719
59999	0.405244	0.009955	-0.004474	-0.021868	0.000233	-0.038524	-0.007183

60000 rows × 7 columns

Fig: The 7 principal components.

MODEL BUILDING:

Which model was used? And Why?

Logistic regression was chosen for this task due to its:

- *Suitability for binary classification:* It is a statistical method used for binary classification tasks, where the goal is to predict the probability of an instance belonging either positive or negative class.
- *Interpretability:* Logistic regression provides insights into the relationship between features and is widely used in situations where interpretability, efficiency, and simplicity are important considerations.

Training and Testing Split:

- The data is divided into training and testing sets using train_test_split. This ensures the model is evaluated on unseen data to assess its generalizability.
- A test size of **20%** is used, leaving **80%** of the data for training.
- A random state of 0 is set for reproducibility (ensuring the same split if the code is run multiple times).

Class Weighting:

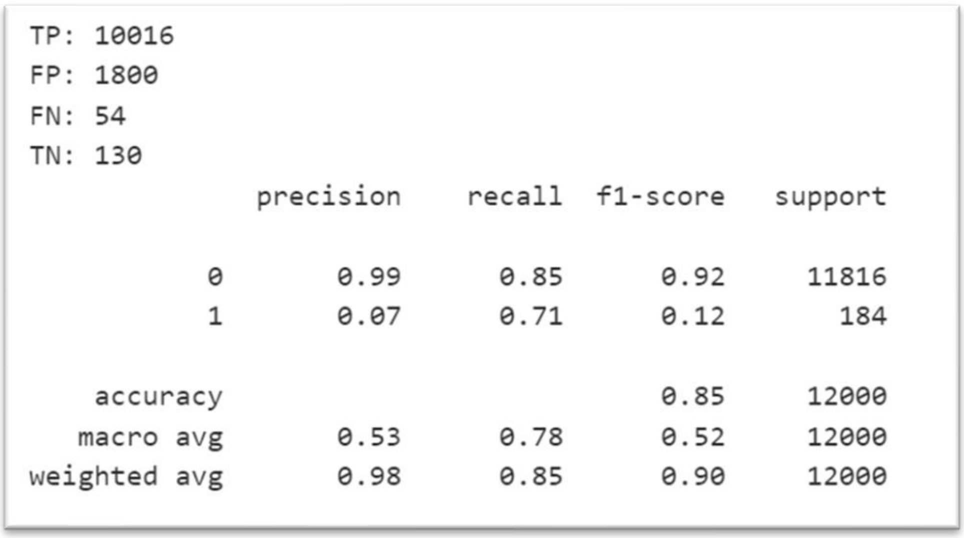
- Class weights are incorporated into the logistic regression model using the **class_weight** parameter because the target classes are imbalanced (i.e., Positive: 1000 & Negative: 59000).
- The code calculates a weight based on the ratio of the negative class (y == 0) to the positive class (y == 1) within the training data.
- Assigning a higher weight to the minority class helps the model focus on learning from less frequent examples and potentially improves its performance in predicting that class.

Model Training:

- A logistic regression model is instantiated using the *LogisticRegression* class from the scikit-learn library.
- The model undergoes hyperparameter tuning, with careful adjustment of the inverse regularization parameter (C) set to 0.1. This parameter serves to regulate the model's complexity and serves as a safeguard against overfitting.

Model Evaluation:

- The *confusion_matrix* function is employed to evaluate the model's performance by comparing predicted labels with true labels.
- Individual metrics like True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN) are extracted from the confusion matrix.
- Additionally, a classification report is generated, providing a detailed breakdown of the model's performance metrics such as precision, recall, F1-score, and support for each class.



```
TP: 10016
FP: 1800
FN: 54
TN: 130
              precision    recall  f1-score   support

           0       0.99      0.85      0.92     11816
           1       0.07      0.71      0.12       184

    accuracy                           0.85     12000
   macro avg       0.53      0.78      0.52     12000
weighted avg       0.98      0.85      0.90     12000
```

Fig: Classification results of the model

The classification results of the model are presented above, where class 0 represents the negative class and class 1 represents the positive class.

- With TP (True Positives) at 10,016, FP (False Positives) at 1,800, FN (False Negatives) at 54, and TN (True Negatives) at 130, these values provide a snapshot of the performance of a classification model. The high TP count indicates the model's effectiveness in correctly identifying positive instances. However, the presence of FP suggests a notable rate of false alarms, while FN indicates instances where positive cases were missed. The low TN count indicates a smaller portion of correctly identified negative instances. Analysing these values enables a deeper understanding of the model's strengths and weaknesses, informing strategies for optimization and refinement.

- **Precision:** The precision for the negative class is notably high at 99%, indicating that the model effectively identifies true negative instances. However, for the positive class, the precision is substantially lower at 7%, suggesting a high rate of false positives.

- **Recall:** The negative class exhibits a robust recall of 85%, signifying the model's adeptness in capturing the majority of actual negative instances. In contrast, the positive class demonstrates a recall of 71%, suggesting a moderate proficiency in identifying true positive instances.

- **F1-score:** The F1-score, which balances precision and recall, is considerably higher for the negative class (0.92) compared to the positive class (0.12). This indicates that the model achieves a better balance between precision and recall for the negative class.

- **Support:** The dataset exhibits a significant class imbalance, with the negative class comprising a considerable number of instances (11,816), while the positive class is notably underrepresented, with only 184 instances.

- **Accuracy:** The model demonstrates an 85% accuracy rate, indicating its proficiency in accurately classifying the majority of cases. However, it's essential to recognize that this high accuracy is largely attributable to the prevalence of negative instances within the dataset.

In essence, although the model exhibits praiseworthy proficiency in recognizing negative instances, its capability to detect positive instances falls notably short. This incongruity underscores the need for additional investigation and perhaps the adoption of methodologies aimed at mitigating class imbalances, thereby augmenting the model's precision in classifying positive instances.

CONCLUSION:

- The primary objective of this project was to accurately forecast the target variable "class" through a comprehensive analysis of a dataset featuring numerous features.
- During the analysis, it became apparent that high-dimensional datasets significantly complicate even the most fundamental exploratory procedures, making execution and interpretation more challenging.
- Following the application of various tools and techniques for data cleaning and imputation, the dimensions were effectively reduced from 170 to 163, with all features now possessing numerical values.
- Given the impracticality of fitting models with high-dimensional datasets, Principal Component Analysis (PCA) was employed as a dimensionality reduction technique. By ensuring that 98% of the variance was retained, the dataset was condensed to just 7 features from its original 170 columns.
- Although the logistic regression model achieved an overall accuracy of 85% in classifying data points, it is notable that the precision for the minority class (1) was low, at 0.07. This indicates a higher rate of false positives (1800) for that class (1).

Future Scope:

- Experiment with alternative classification algorithms (e.g., decision trees, random forests, support vector machines) to identify models better suited to the dataset.
- Fine-tune hyperparameters to optimise model performance and generalizability. Additionally, explore other dimension reduction techniques.
- Explore ensemble learning methods to leverage the strengths of multiple models and improve predictive accuracy.
- Address class imbalance through specialised techniques such as oversampling, under sampling, or utilising algorithms designed for imbalanced data.
- Consider incorporating domain-specific knowledge or feature engineering techniques to enhance model interpretability and performance.
- Explore advanced machine learning techniques like deep learning for potential improvements in predictive accuracy and robustness.