EXPERIMENT NO: 1	DATE:
EXPERIMENT NAME: New York Yellow Taxi Demand prediction using Machine Learning	

1) BUSINESS UNDERSTANDING

Medallion (yellow) cabs are concentrated in the borough of Manhattan but can be hailed anywhere throughout the five boroughs of New York City with a raised hand or from a taxi stand.

In the New York city, people use taxis at a much higher frequency than most places. Instead of booking customers by phone ahead of time, there is still a majority of New York taxi drivers that pick-up passengers on street. Hailing a cab is as simple as stepping off the curb and holding out your arm out.

In 2011, Uber announced it was launching its ride sharing service in New York City, ushering in a new era of disruption in the taxi service industry.

By allowing anyone to enroll as a taxi driver with their own private vehicle and making it as easy as a click of a cell phone app to book a ride, Uber began establishing a new network of cabs outside of the medallion system.As per recent studies, Taxi patronage has considerably declined since 2011 due to competition from these ride share services.

What if there existed a way to predict taxi ridership that gives valuable insights to taxi dispatchers — as in how to position cabs where they are most needed, how many taxis to dispatch, and how ridership varies over time. Such prediction will help dispatchers immensely in making important decisions that could revive their profit margin.

2) MACHINE LEARNING PROBLEM FORMULATION

- Let X be the feature matrix with dimensions (N, D) where N) is the number of data points (taxi trips) and D is the number of features.
- Let Y be the target variable, which represents the demand for each trip. Y is a vector of length N.
- X*i* represents the feature vector for the *i*-th taxi trip, where X*i* is a vector of length D.
- The goal is to find a model that predicts the demand Y based on the feature matrix X. This can be represented as a function

  Y*i*=f(X*i*) for *i*=1,2,3 ...N

- Choose a regression model, such as linear regression, to estimate the function f(X).In the case of linear regression, the model might be represented as:

  Yi= β0+β1Xi1+β2Xi2+…+βDXiD+ϵi

- Where β0 is the intercept. β1, β2 and so on are the coefficient of each feature.
- ϵi is the error term of ith prediction.
- Define a loss function to measure the model's performance. Common choices for regression problems include Mean Squared Error (MSE) or Mean Absolute Error (MAE):

$$MSE = 1/N * \sum_{i=1}^{N} (Yi - f(Xi))^2$$

3) EXPLORATORY DATA ANALYSIS:

The data used for building our solution was collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP).

Taxi Types:

- Yellow Taxi - Yellow Medallion Taxicabs:
  These are the iconic NYC yellow taxis that offer transportation exclusively through street-hails. The number of these taxis is restricted by a limited number of medallions issued by the TLC (Taxi and Limousine Commission). To avail of this mode of transportation, you simply stand on the street and signal an available taxi by raising your hand. These pickups are not scheduled in advance.
- For Hire Vehicles (FHVs):
  FHV transportation is arranged by pre-booking through a dispatcher or limo company. These FHVs are not allowed to pick up passengers through street hails, as those rides are not considered pre-arranged.
- Green Taxi - Street Hail Livery (SHL):
  The SHL program enables livery vehicle owners to license their vehicles and equip them with distinctive green borough taxi branding, meters, credit card machines, and, importantly, the authorization to accept street hails in addition to pre-scheduled rides.

For the purpose of our modelling we'll be using the yellow taxi trip data from January 2015.

In 2015, a series of yellow taxi trip data files were recorded, each corresponding to a specific month. These files varied in size, with January having the largest size at 1.84 GB, and August being the smallest at 1.62 GB. Each file contained records ranging from 11,130,304 to 13,351,609, and they all shared a common feature count of 19. These records provide valuable insights into taxi trips throughout the year.

The dataset contains various fields providing information about yellow taxi trips. These fields include:

1. VendorID: indicating the service provider.
2. lpep_pickup_datetime and lpep_dropoff_datetime: for the start and end times of the trip.
3. Passenger_count: for the number of passengers.
4. Trip_distance: for the trip's distance.
5. PULocationID and DOLocationID: for the pick-up and drop-off locations.
6. RateCodeID: specifying the final rate code.
7. Store_and_fwd_flag: indicating if the record was stored before transmission.
8. Payment_type: for the passenger's payment method.

9. Fare_amount: representing the fare calculated by the meter.
10. Extra: accounting for additional charges.
11. MTA_tax: as the automatically triggered tax.
12. Improvement_surcharge: for the surcharge.
13. Tip_amount: for tips (auto-populated for credit card transactions).
14. Tolls_amount: for toll expenses.
15. Total_amount: for the total charge to passengers, excluding cash tips.

These fields collectively provide detailed information about each taxi trip, enabling comprehensive analysis and insights.

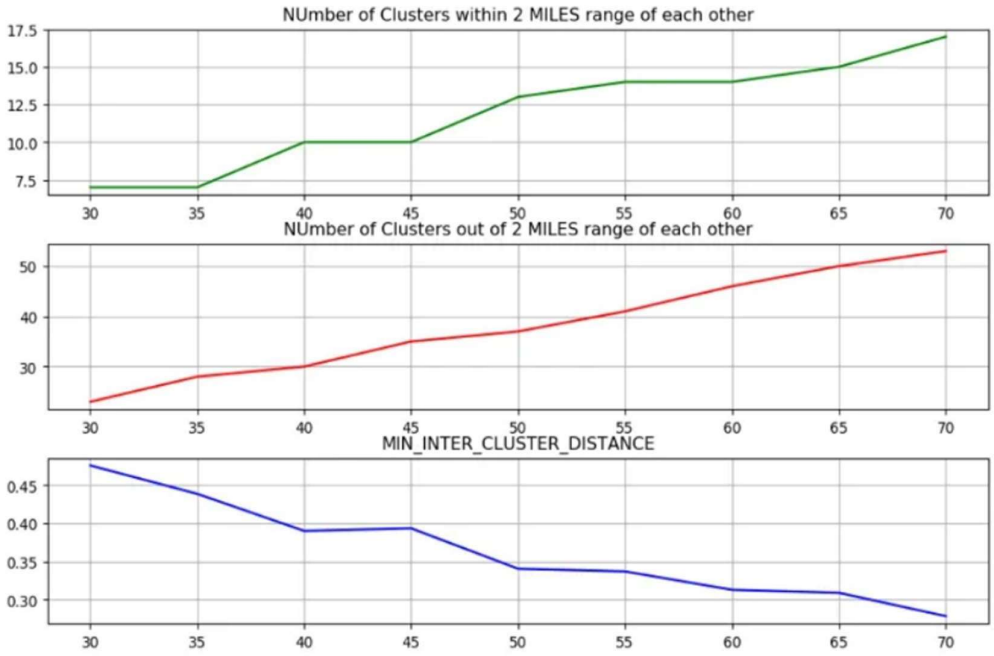
For data cleaning univariate analysis of the data is performed wherein all erroneous data was removed. As a part of the data cleaning phase we check if the pick-up latitude and longitude fall within NYC and remove the ones that don't.

Pick up Map After Cleaning concentrating only in NYC

Good data preparation allows for efficient analysis and limits the errors and inaccuracies that occur due to erroneous data.

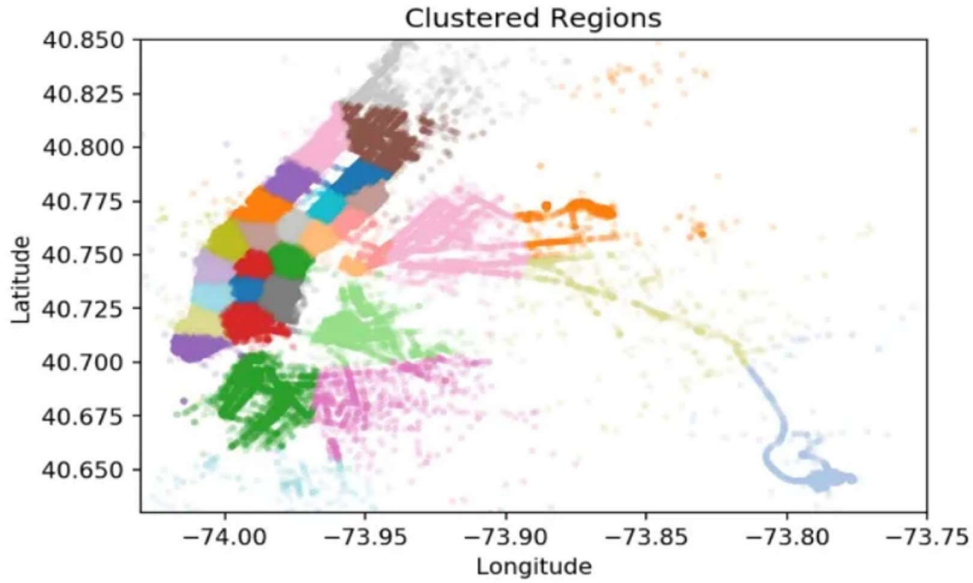
1. Clustering and segmentation:

- To address our objective of predicting pick-up locations within specific time intervals, we must first delineate regions and time slots from the data. We'll start by partitioning New York City into K clusters using the latitude and longitude coordinates of pick-up locations. We'll employ K-Means clustering to group these pick-up points into distinct regions.
- K-Means clustering tends to create clusters of relatively equal sizes, resulting in smaller, densely populated clusters for regions with a higher number of pick-ups and larger, more sparsely populated clusters for regions with fewer pick-ups.



- Based on our data analysis, we've noticed that a taxi can typically travel a distance of up to 2 miles in a 10-minute interval. Consequently, we aim to ensure that the distance between inner cluster points is greater than 2 miles but not less than 0.5 miles.
- To find the optimal K value that adheres to this criterion, we conducted experiments with various cluster values within the range of 30 to 70. After

thorough experimentation, we determined that having 30 clusters is the most suitable choice.



The number of clusters within the 2-mile radius : 7.0
The number of clusters outside the 2-mile radius : 23.0
The minimum inter cluster distance : 0.475971
As we can see from the graphs that the above condition is met, we choose that as the optimal value for k as 30.

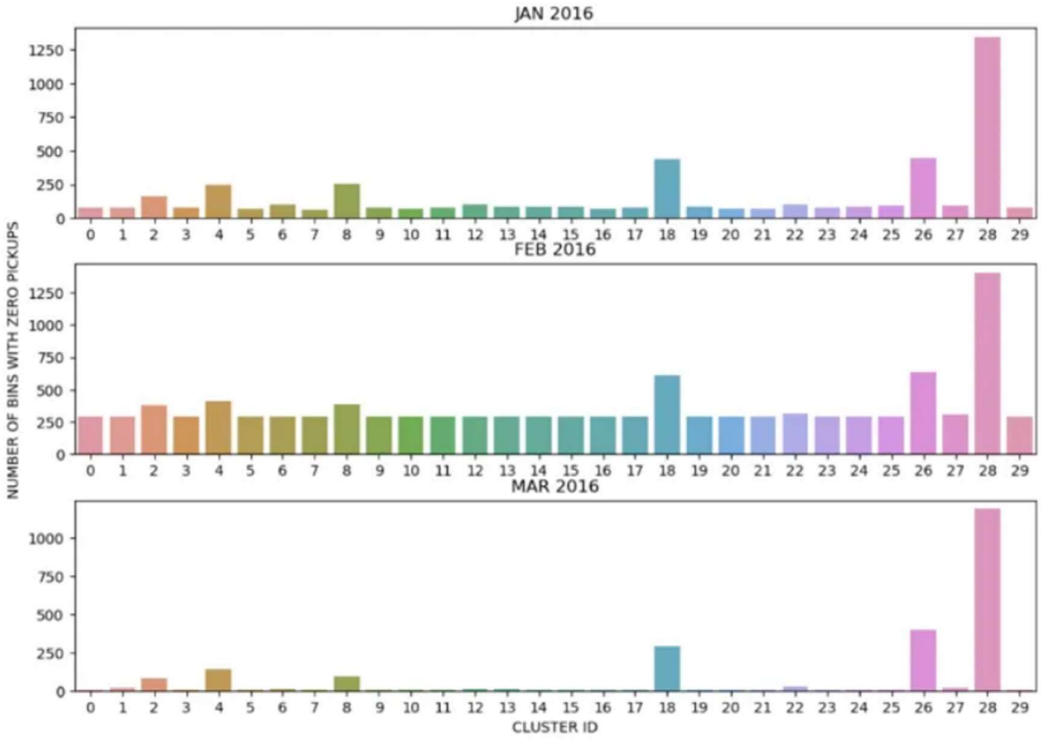
2. Time Binning

- We have divided each region into time intervals of 10 minutes, which means that every time bin covers a span of 10 minutes. The time information in the data is originally presented in the "YYYY-MM-DD HH:MM:SS" format, but we've converted it into Unix time format to facilitate minute and hour-level time analysis.
- This conversion allows us to create a total of 4,464 possible 10-minute interval bins.
  Total number of pickup_bins from clusters : 130567

Number of possible pickup_bins from all clusters : 133920
Now, with any region-ID and 10 min time bin, we can predict the number of pickups.

3. Smoothing the Data

- Now that the data is divided into 10-minute interval bins, each bin should contain at least one pickup. There are chances that a time bin may contain zero pick-ups leading to ratio feature error. In order to smooth this outlier:



- We count the total number of pickups in each bin. There would be no value if there were no pickups in a time bin. Fill the missing pickup time bins with 0. Then, we add a few pickups from neighboring bins to the bin which contains zero pickups in order to make all the neighboring bin pickups equal.

4. Time Series and Fourier Transforms

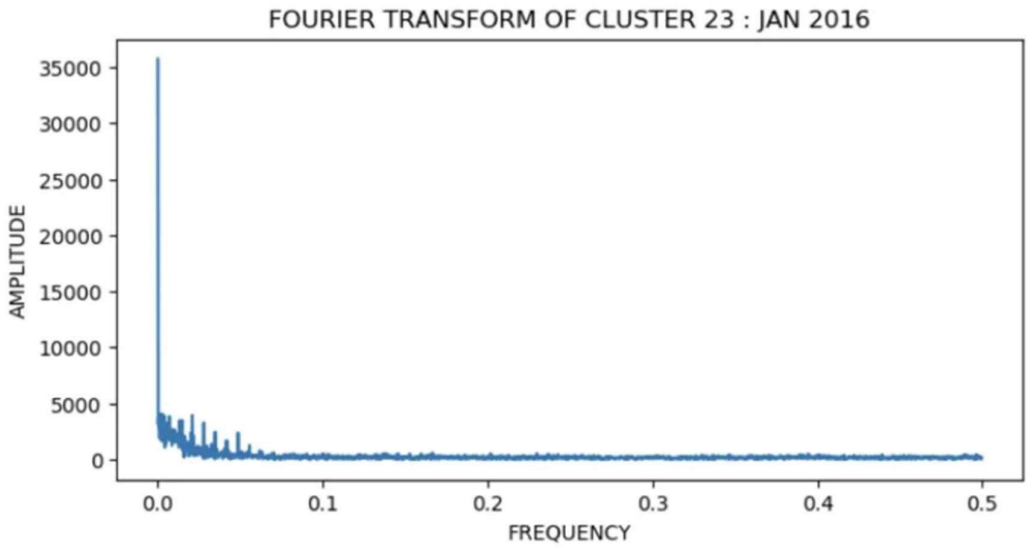
- In essence, any waveform can be expressed as a combination of countless sine waves, each characterized by its own amplitude and frequency. When we

  examine the monthly pickup counts within each cluster, we notice a recurring pattern. However, this pattern doesn't adhere to a single, constant frequency; it's aperiodic in nature. Instead, it's constructed from an infinite set of sine waves, each with its unique frequency.

- The Fourier transform provides us with a tool to transform our pattern from the time domain (where we observe the number of pickups over time) into the

frequency domain (which can be thought of as bins representing different pickup frequencies).

- For each cluster, there exists a distinctive pattern, and by using the Fourier transform, we can extract the most prominent frequencies and their corresponding amplitudes, which constitute our cluster's pattern.
- These frequencies and amplitudes act as valuable features. They reflect the demand characteristics within that specific cluster. We can then incorporate these frequency and amplitude data into our prediction model to forecast the number of future pickups accurately.



- Plotting the Amplitude and frequency of the sinusoidal components, we see that
  1. The pattern whose repetition is very high will have a high frequency component and vice versa.
  2. There are high frequency in morning and evening time durations as the pick-ups are high during peak hours.

  The same applies to day and night but with less frequencies.

- We've observed a significant amplitude at the frequency of f=0. As the frequency of a sine wave approaches zero, the sine wave becomes more like an axis-parallel or linear component, resembling a DC (direct current) component. However, since the wave we've Fourier transformed is periodic and repeats, the DC component appears somewhat ambiguous because it captures information from the previous part of the wave. As a result, we choose not to take its amplitude and frequency into consideration.

4) MACHINE LEARNING MODELS

Baseline Models:

In addressing our business problem, the initial step involved the implementation of baseline models to establish a foundation for predictions. We explored three primary baseline models:

- Simple Moving Averages
- Weighted Moving Averages
- Exponential Weighted Moving Average (EWMA)

For each of these models, we examined two distinct approaches to capture variations in the data:

- Using Ratios (2016 data to 2015 data): Ratio = P_2016 / P_2015
- Using Previous Known Values of 2016 Data

Comparison of Baseline Models:

To evaluate the performance of these baseline models, we employed two key metrics:

- Mean Absolute Percentage Error (MAPE): This metric helped us assess the accuracy of our predictions.
- Mean Squared Error (MSE): A measure employed to gauge the model's performance, particularly in handling outliers.

Simple Moving Averages

The Simple Moving Averages model employs the previous n values to predict the next value. Utilizing ratio values (Rt), we calculate the moving average using:

$Rt = (Rt-1 + Rt-2 + Rt-3 \ldots Rt-n) / n$

For the 2016 values, the moving average is computed as:

$Pt = (Pt-1 + Pt-2 + Pt-3 \ldots Pt-n) / n$

Weighted Moving Averages

The Weighted Moving Averages model introduces a weighting scheme that emphasizes the latest values, considering them more influential in predicting the future. Utilizing ratio values (Rt), the formula for weighted moving averages is:

$Rt = (N*Rt-1 + (N-1)*Rt-2 + (N-2)*Rt-3 \ldots 1*Rt-n) / (N*(N+1)/2)$

For the 2016 values, the weighted moving average is computed using a weighted sum:

$Pt=(N*Pt-1+(N-1)*Pt-2+(N-2)*Pt-3 \ldots . 1*Pt-n)/(N*(N+1)/2)$

Exponential Moving Averages (EMA)

Exponential Moving Averages (EMA) offer a logical approach to assigning weights using a single hyperparameter alpha (α) in the range of 0 to 1. The weights and window sizes are configured based on this parameter.

The EMA formula is:

$R't = \alpha*Rt-1 + (1-\alpha)*R't-1$

For the 2016 values, the exponential moving average is computed as:

$P't=\alpha*Pt-1+(1-\alpha)*P't-1$

We have chosen MAPE (Mean Absolute Percentage Error) and MSE (Mean Squared Error) as our error metrics to gauge the accuracy and performance of our models, respectively.

Error Metric Matrix (Forecasting Methods)

Method	MAPE	MSE
Moving Averages (Ratios)	0.182115517339	400.0625504032258
Moving Averages (2016 Values)	0.14292849687	174.84901993727598
Weighted Moving Averages (Ratios)	0.178486925438	384.01578741039424
Weighted Moving Averages (2016 Values)	0.135510884362	162.46707549283155
Exponential Moving Averages (Ratios)	0.177835501949	378.34610215053766
Exponential Moving Averages (2016 Values)	0.135091526367	159.73614471326164

From the error metric matrix, it is evident that the most effective forecasting model for our prediction is:

$P't = \alpha * Pt-1 + (1-\alpha) * P't-1$ (Exponential Moving Averages using 2016 Values).

Feature Engineering:

After doing all the data cleaning and preparation and baseline modelling, we now have an arsenal of features which would help build a good predication model. We see that, from baseline modeling, how Exponential weighted moving averages gives the best forecasting among the rest. We will use this as a feature while building the regression model along with others we got from data preparation stage.

Following baseline modelling and model selection, we proceeded with feature engineering to enhance the prediction model. We carefully curated a set of 14 crucial features that would contribute to a robust regression model:

- Latitude and Longitude of the Cluster
- Top Three Amplitudes with Respective Frequencies from Fourier Transform

- Five Previous Pickup Densities from the Cluster Relevant to the Pickup Density to be Predicted
- The Pickup Density Predicted by the EMA_PREV Baseline Model

	LAT	LON	AMP1	AMP2	AMP3	FREQ1	FREQ2	FREQ3	P5	P4	P3	P2	P1	EMA_PRED
2	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	0.0	0.0	0.0	0.0	106.0	74.0
3	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	0.0	0.0	0.0	106.0	181.0	148.0
4	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	0.0	0.0	106.0	181.0	263.0	228.0
5	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	0.0	106.0	181.0	263.0	228.0	228.0
6	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	106.0	181.0	263.0	228.0	241.0	237.0

Regression Models

Test and train split:

For test and train split by time, we take 3 months of 2016 pickup data and split it such that for every region we have 70% data in train and 30% in the test. As this is a time-series problem, we have to split our train and test data on the basis of time.

Linear Regression

Data Pre-processing: To prepare the data for modelling, we start by standardizing the training and testing data. Standardization involves removing the mean and scaling the data to have a unit variance. This step helps in achieving a consistent scale and ensuring that each feature contributes equally to the learning process.

Model Selection and Training: In this case, the Stochastic Gradient Descent (SGD) Regressor is chosen for model training. SGDRegressor is ideal for regression problems, especially when dealing with a large number of training and testing instances. It offers efficiency by randomly selecting a training instance for each iteration, allowing faster computation.

Code:

```
from sklearn.linear_model import LinearRegression

# Create and train the Linear Regression model

lr_reg = LinearRegression().fit(df_train, tsne_train_output)
```

The df_train represents the training dataset, and tsne_train_output contains the corresponding target values for training.

Prediction and Model Evaluation: Next, we use the trained model to make predictions on both the training and testing datasets. These predictions are then rounded to obtain discrete values.

Code:

```
# Predictions on the testing dataset

y_pred_test = lr_reg.predict(df_test)

lr_test_predictions = [round(value) for value in y_pred_test]


# Predictions on the training dataset

y_pred_train = lr_reg.predict(df_train)

lr_train_predictions = [round(value) for value in y_pred_train]
```

Evaluation Metrics:

To assess the model's performance, we utilize two evaluation metrics:

- Mean Absolute Percentage Error (MAPE): A measure of the accuracy of the model's predictions, expressed as a percentage of the actual values.
- Mean Squared Error (MSE): A measure of the average squared differences between the predicted and actual values.

The results of the model evaluation are as follows:

- Train MAPE: 0.1175, Train MSE: 238.5128
- Test MAPE: 0.1165, Test MSE: 218.275

Random Forest Regression

Model Selection and Hyperparameter Tuning: Random Forest Regression is chosen as the model due to its ability to maintain accuracy for a large proportion of data and prevent overfitting, especially when using a suitable number of trees.

In this case, hyperparameter tuning is performed using RandomizedSearchCV. This method randomly samples a fixed number of parameter settings from specified distributions, providing an efficient way to explore a wide range of hyperparameter combinations.

Code:

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import RandomizedSearchCV

# Define the RandomForestRegressor with specified hyperparameters

regr1 = RandomForestRegressor(

    max_features='sqrt',

    min_samples_leaf=4,

    min_samples_split=3,

    n_estimators=40,

    n_jobs=-1

)

# Fit the model with the training data

regr1.fit(df_train, tsne_train_output)
```

The hyperparameters chosen are:

max_features: 'sqrt'

min_samples_leaf: 4

min_samples_split: 3

n_estimators: 40 (number of trees in the forest)

n_jobs: -1 (utilize all available CPU cores)

Prediction and Model Evaluation:

Using the trained Random Forest model, predictions are made on both the training and testing datasets. The predicted values are then rounded to obtain discrete values.

```
# Predictions on the testing dataset
y_pred_test = regr1.predict(df_test)
rndf_test_predictions = [round(value) for value in y_pred_test]


# Predictions on the training dataset
y_pred_train = regr1.predict(df_train)
rndf_train_predictions = [round(value) for value in y_pred_train]
```

Evaluation Metrics: To assess the model's performance, we calculate two key evaluation metrics:

- Mean Absolute Percentage Error (MAPE): Evaluates the accuracy of the model's predictions, represented as a percentage of the actual values.
- Mean Squared Error (MSE): Measures the average squared differences between the predicted and actual values.

The performance of the model is as follows:

- Train MAPE: 0.1266, Train MSE: 272.4468
- Test MAPE: 0.1261, Test MSE: 250.1323

XGBoost Regressor

Model Selection and Hyperparameter Tuning:

XGBoost Regressor, known for its speed and performance, is selected as the model for this phase. It's an implementation of gradient boosted decision trees.

For this model, hyperparameter tuning is crucial to achieve optimal performance. The chosen hyperparameters are:

```
import xgboost as xgb


x_model = xgb.XGBRegressor(
    learning_rate=0.1,
    n_estimators=1000,
    max_depth=3,
    min_child_weight=3,
    gamma=0,
    subsample=0.8,
    reg_alpha=200,
    reg_lambda=200,
    colsample_bytree=0.8,
    nthread=4
)
# Fit the model with the training data
x_model.fit(df_train, tsne_train_output)
```

Key hyperparameters include:

- learning_rate: 0.1 (controls the step size shrinkage during training)
- n_estimators: 1000 (number of boosting rounds)
- max_depth: 3 (maximum depth of a tree)
- min_child_weight: 3 (minimum sum of instance weight needed in a child)

- gamma: 0 (minimum loss reduction required to make a further partition on a leaf node)
- subsample: 0.8 (the fraction of samples used for fitting the trees)
- reg_alpha and reg_lambda: 200 (L1 and L2 regularization terms on weights)
- colsample_bytree: 0.8 (the fraction of features used for each boosting round)

Prediction and Model Evaluation:

Using the trained XGBoost model, predictions are made on both the training and testing datasets. The predicted values are then rounded to obtain discrete values.

```
# Predictions on the testing dataset
y_pred_test = x_model.predict(df_test)
xgb_test_predictions = [round(value) for value in y_pred_test]


# Predictions on the training dataset
y_pred_train = x_model.predict(df_train)
xgb_train_predictions = [round(value) for value in y_pred_train]
```

Evaluation Metrics:

To assess the model's performance, we calculate two key evaluation metrics:

- Mean Absolute Percentage Error (MAPE): Evaluates the accuracy of the model's predictions, represented as a percentage of the actual values.
- Mean Squared Error (MSE): Measures the average squared differences between the predicted and actual values.

The performance of the model is as follows:

- Train MAPE: 0.1151, Train MSE: 225.9773
- Test MAPE: 0.1158, Test MSE: 213.0978

5)RESULTS

We can see that XGBoost performed based on TEST_MAPE was better than other models. Linear Regression was able to achieve similar values as XGBoost. But, we choose XGBoost so as to avoid overfitting.

Some of the more important features used by these models are pickupdensity(P(t-1)) followed by EMA Model using the previous pickup densities. Using these predictions, the Yellow taxi owners can decide where to station themselves during their hours of least pickup.

	TEST_MAPE	TEST_MSE
XGBOOST_REG	0.1158	213.0978
LINEAR_REGRESSION	0.1165	218.2750
EMA_PREV	0.1205	218.6057
WMA_PREV	0.1206	220.4236
SMA_PREV	0.1231	232.9855
RANDOM_FORESTS_REG	0.1261	250.1323
EMA_RATIOS	0.1574	473.4599
WMA_RATIOS	0.1580	489.5664
SMA_RATIOS	0.1613	524.1292

6) KEY COMPONENTS

1) Dask:

Dask is a Python library that enables parallel and distributed computing for handling large-scale data processing tasks. It provides a flexible, dynamic task scheduler for processing data in parallel, making it easier to work with large datasets, scale computations, and improve performance in data analytics and machine learning applications.

2) Graphviz:

Graphviz is an open-source graph visualization software used to create and display graphical representations of graphs and networks. It provides a range of tools for drawing, rendering, and analysing graphs, making it valuable for visualizing data structures, network diagrams, and flowcharts in fields like computer science, data analysis, and software engineering.

3) kmeans clustering:

K-means clustering is an unsupervised machine learning algorithm used for grouping data points into clusters based on their similarity. It assigns each data point to the nearest cluster center (centroid) and iteratively updates the centroids to minimize the sum of squared distances. It is widely used in data analysis, image segmentation, and pattern recognition.

4) Time binning:

Time binning is a technique in data analysis that involves dividing a continuous time period into discrete intervals or bins. It's used to aggregate and organize time-series data into manageable chunks for analysis. Time binning helps identify patterns, trends, and statistics within specific time intervals, making it easier to interpret and visualize temporal data.

5) Fourier transform:

The Fourier transform is a mathematical technique used to analyze and decompose complex signals or functions into their constituent sinusoidal components. It's essential in signal processing and image analysis, helping to reveal the frequency domain characteristics of a signal, making it useful for tasks like filtering, compression, and pattern recognition.

6) Simple moving average:

A Simple Moving Average (SMA) is a widely used technical analysis tool in finance. It calculates the average value of a dataset over a fixed time period, providing a smoothed representation of a trend. SMAs are commonly used to identify price trends and support trading decisions in the stock market.

7) Weighted moving average:

A Weighted Moving Average (WMA) is a variation of the simple moving average where different weights are assigned to data points within a time period. It places more significance on recent data, allowing for a more responsive indicator in data analysis. WMAs are useful for capturing short-term trends and reducing lag in trend analysis.

8) Exponential Weighted Moving Average (EWMA):

EWMA is a time-series forecasting method that assigns exponentially decreasing weights to historical data points. Recent data is given more importance, making it effective for capturing short-term trends and smoothing noisy time-series data.

9) Mean Absolute Percentage Error (MAPE):

MAPE is a widely used metric for measuring the accuracy of forecasting models. It calculates the percentage difference between predicted and actual values, providing an easy-to-interpret assessment of a model's forecasting performance.

10) Grid Search:

Grid Search is a hyperparameter tuning technique in machine learning. It systematically explores a predefined set of hyperparameter combinations to find the best model performance. It's a deterministic method that exhaustively searches through all specified parameter combinations.

11) Random Search:

Random Search is another hyperparameter optimization method. Instead of exhaustively evaluating all combinations like Grid Search, it randomly samples hyperparameter values. While it may not guarantee finding the best combination, it is computationally more efficient and often leads to good results.

EXPERIMENT NO: 2	DATE:
EXPERIMENT NAME: Personalized Cancer Diagnosis using Machine Learning	

1) BUSINESS PROBLEM/PROBLEM STATEMENT

We obtain a tumour sample from a patient who appears to have cancer, and we sequence the DNA of the sample to determine the patient's genetic makeup. A tumour may have thousands of genomic alterations after being sequenced. In summary, a "mutation" is a slight alteration in a gene that leads to cancer. There is also the crucial fact that every gene has a variant connected to it. We must now categorize the gene and its variant to determine which class it belongs to (there are a total of 9 classes). Cancer only affects a select few classes. Let's clarify the workflow once more.

- A molecular pathologist chooses a list of interesting genetic variants for analysis.
- In the medical literature, the molecular pathologist looks for examples that somehow relate to the genetic variations of interest.
- Finally, the molecular pathologist spends a significant amount of time classifying each variant by reviewing the evidence associated with it.

Steps 1 and 2 can be completed quickly and easily in this situation. However, Step 3 takes a lot of time. Step 3 will be replaced by a machine learning model in our approach.

As a result, the problem statement is to categorize genetic variations using data from clinical literature or research publications that are text-based.

Business limitations of this issue:

- The algorithm must be comprehensible so that a cancer specialist can explain to the patient why the model was assigned to a certain class.
- There is no requirement for low latency, therefore the patient can wait for the results. Since minimal latency is not necessary, we can use sophisticated machine learning models.
- Errors cost a lot of money.

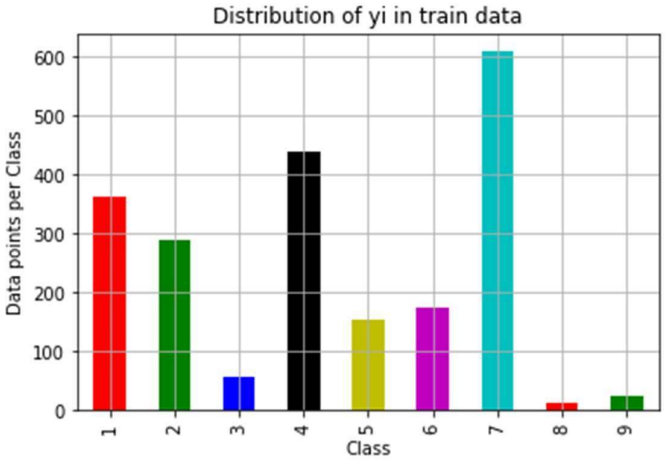
- It needs to have a probability of belonging to a class rather than a specific class.

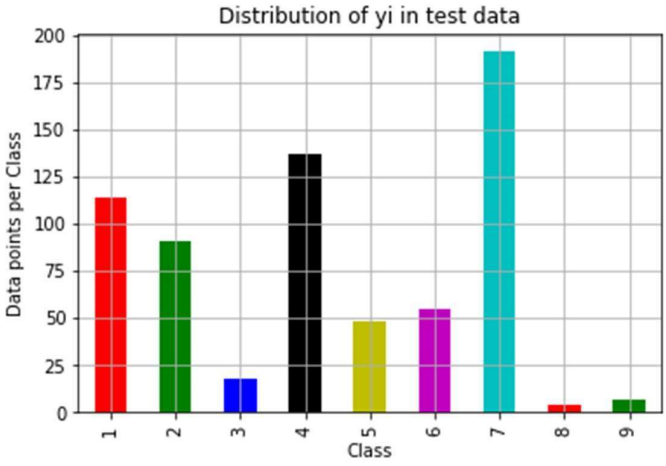
2) MACHINE LEARNING PROBLEM FORMULATION

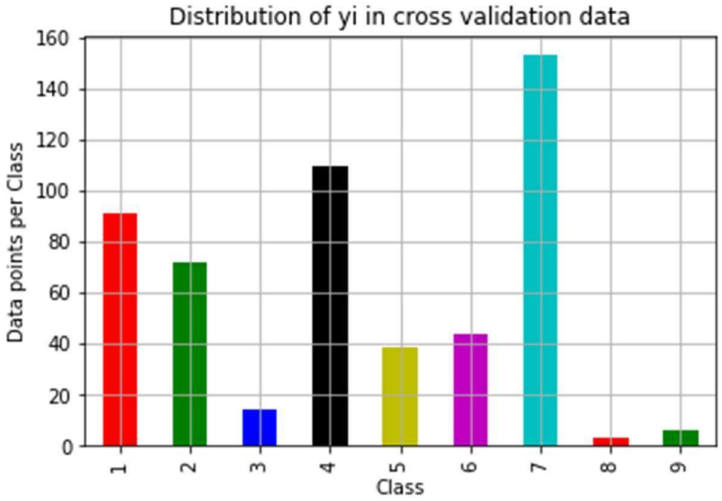
- The nine classes that need to be classified have already been mentioned. It is a multiclass classification problem as a result.
- Performance indicator: Confusion matrix with many classes of Log-Loss (Log-Loss is chosen since it truly uses probability, which is our business requirement).

Machine learning Objective: Predict the probability of each data point belonging to each of the 9 classes.

Machine learning constraints: These are same as Business constraints, which are previously highlighted, are also present in machine learning constraints.



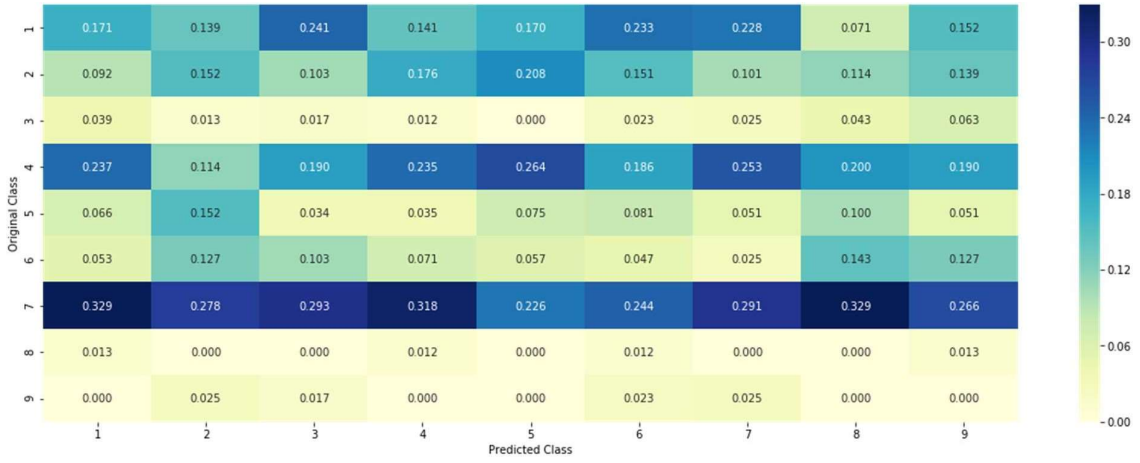

Distribution of yi in test data
Data points per Class
Class
0
25
50
75
100
125
150
175
200
1
2
3
4
5
6
7
8
9


Distribution of yi in cross validation data
Data points per Class
Class
0
20
40
60
80
100
120
140
160
1
2
3
4
5
6
7
8
9

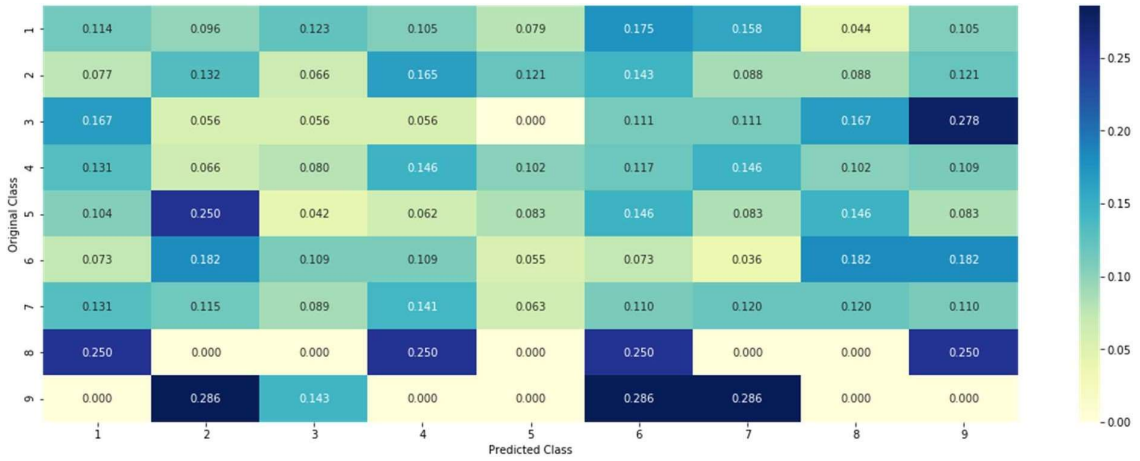
3) EXPLORATORY DATA ANALYSIS

Since we already know that log-loss can range from zero to infinity. In order to determine whether our ML model is sound, we first define a random model whose log-loss is compared to our ML model. Our random model produces a log loss of approximately 2.5 after receiving the data.

Even so, we noticed that the diagonal elements of the accuracy and recall matrix, which represent the precision and recall across all classes, appeared to be quite low due to the random model. The matrices for precision and recall are attached below. It is also abundantly obvious from the aforementioned distributions that classes 1, 2, 4, and 7 make up the majority.



Precision of a random model



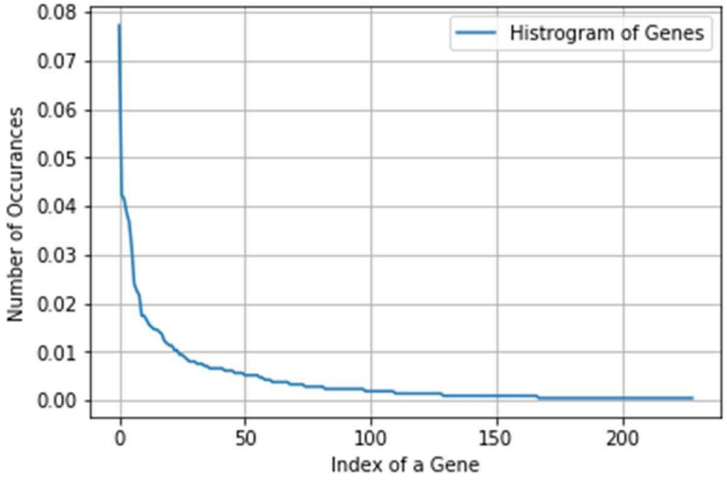
Recall of random model

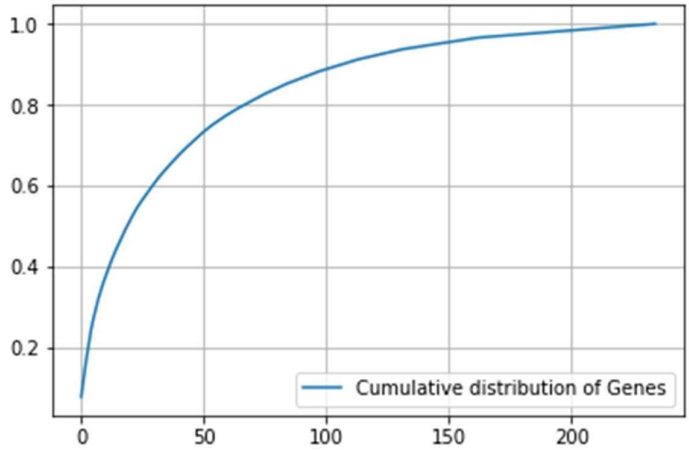
Univariate Analysis:

Before utilising a feature, we determine its applicability for predicting the class label in a variety of ways. If the functionality is not useful, we can easily delete it.

1. Feature Gene

That gene is a categorical trait, as we all know. This leads us to conclude that there are 235 different sorts of distinct genes, of which the top 50 most frequent genes account for roughly 75% of the data.

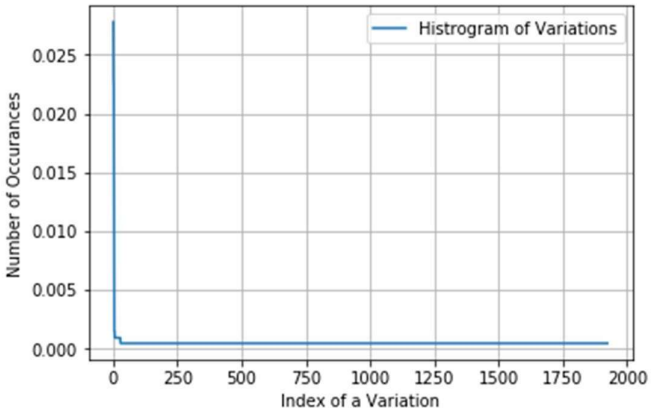




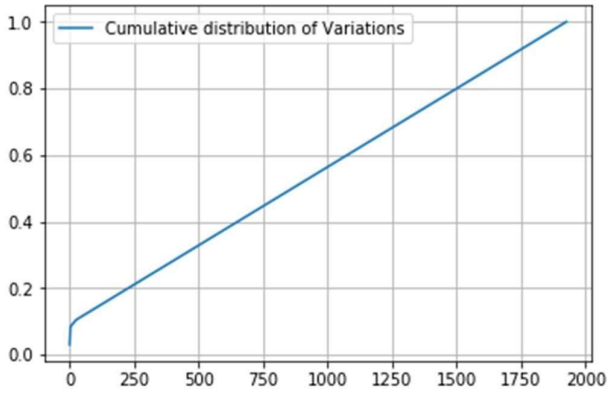
By using one hot encoding and response coding, we are now featuring the gene into the vector. Then, using a calibrated classifier, we built a single straightforward logistic regression model and applied gene features and class labels to it. We discover that the log-loss values for the Train, CV, and Test are essentially the same and that the log loss value is less than 2.5 (the Random Classifier value). Therefore, we can conclude that Gene is a crucial component of our classification. Since test and CV errors are nearly comparable to train mistakes, we can also draw the conclusion that genes are stable features.

2. Differentiation Feature

Here, variation is a categorical feature as well, and we found that 1927 of the 2124 variations contained in the training data were unique, indicating that the majority of variations only occurred once or twice.



CDF of variations looks as follows:



Since the cumulative distribution is linear, the majority of variances in training data only occur once or twice. With the help of one hot encoding and response

coding, we highlight the variation into a vector. Similar to what we did earlier for the gene feature, we construct a straightforward LR model, apply data to it, and discover that the log loss values of Train, CV, and Test are lower than those of the Random Model.

However, there is a considerable disparity between test log loss and gene feature, indicating that variation feature is unstable. However, since the log loss is lower than the Random Model, we must use the variation feature even though it is unstable.

3. Text Function

In text data, there are 53,000 distinct characters words that appear in the training data. We also note that most terms appear just a few times on average, which is typical of text data. By using Response Coding and BOW (one hot encoding), we vectorize the text data.

As in earlier instances, we apply it to the simple model LR and see that the log loss values for Train, CV, and Test are lower than those for the Random Model. The test feature is a steady feature, according to the distributions of CV and Test data.

Combine all the features now in two different ways.

One hot encoding: Due to text data, it is discovered that one hot encoding results in a dimensionality of 55,517.

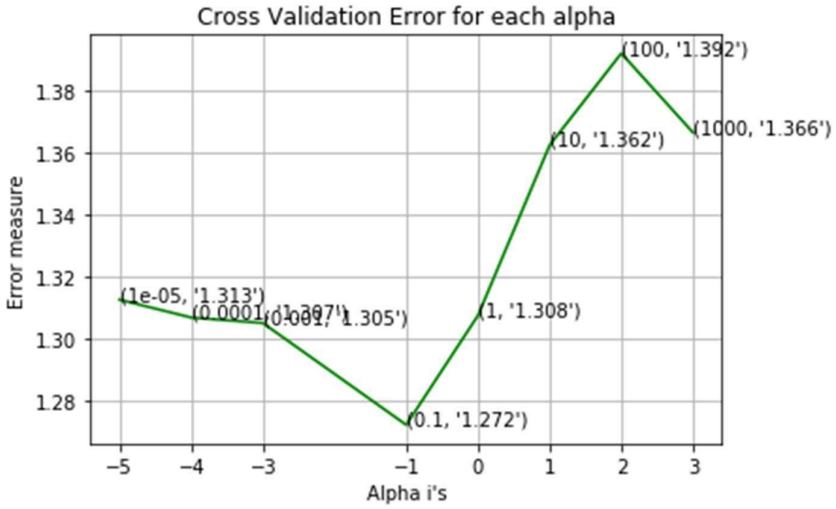
Response Coding: Response Coding reveals that there are 27 dimensions (each feature has 9 dimensions).

4) MACHINE LEARNING MODELS

Approach No. 1

Naive Bayes:

We are aware that the NB model is a foundational paradigm for text data. The training data is now applied to the model, and the CV data is utilised to choose the optimal hyper-parameter (alpha).

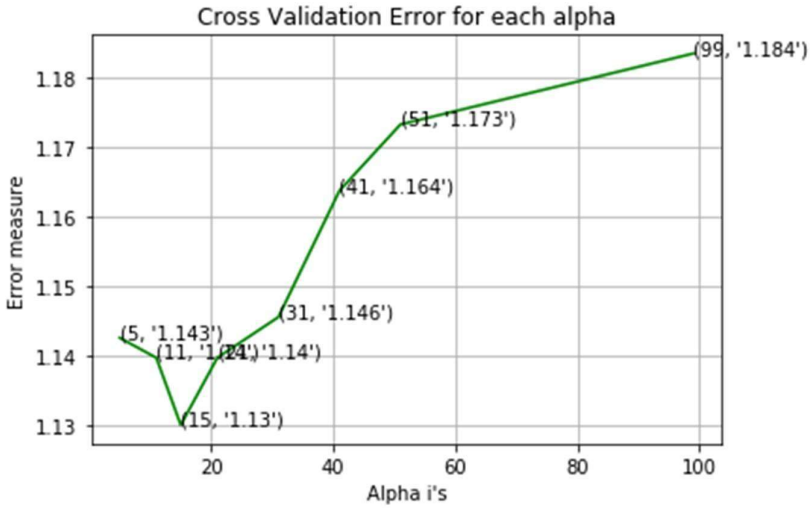


We fit the model using the highest alpha. After applying the test data to the model, we discovered that the log-loss value was 1.27, which is significantly lower than that of the Random Model. Additionally, we learn that 39.8% of all cases were incorrectly classified in this place. Additionally, we looked at each class's probability for each piece of data and interpreted each point. This is to determine why a certain class is being predicted at random. We come to the conclusion that there is very little chance that a misclassified point actually belongs to the anticipated class. The precision and recall matrix reveals that the majority of class 2 points were correctly predicted as 7. Similarly, class 1's most common score is anticipated to be 4.

Approach No. 2

K-Nearest Neighbours:

Even though we are aware that the k-NN model cannot be interpreted (this is a business restriction), we nevertheless utilise it to calculate the log loss numbers. Response coding is used rather than one-hot encoding since k-NN is cursed by dimensionality. After feeding the model with data, we get the ideal hyper-parameter(k).



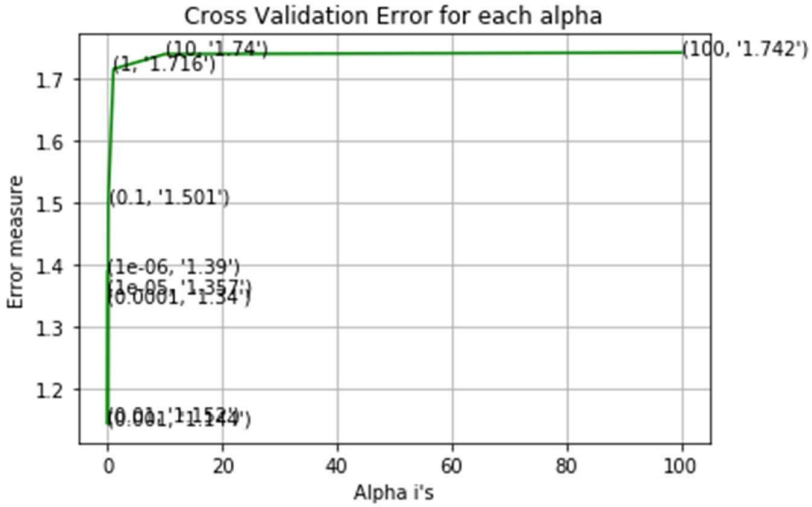
The model is fitted using the best k, and test data is then applied to the model. The log-loss value, which is less than the NB model, is 1.002. However, 39.47 percent of the points were incorrectly identified (nearly as many as the NB model). It was discovered that the majority of class 2 points in the K-NN model were anticipated to be 7. Similarly, the majority of class 1 points were anticipated to be 4.

Approach 3:

Logistic Regression:

The LR model performed exceptionally well with univariate analysis, as we have already shown. As a result, we thoroughly analysed LR using both balanced and unbalanced data.

With class balancing, we also know that LR is interpretable and does well with big dimension data. Therefore, we oversampled points from lower classes, and utilised the model's training data and the CV data to determine the ideal hyper-parameter (lambda).



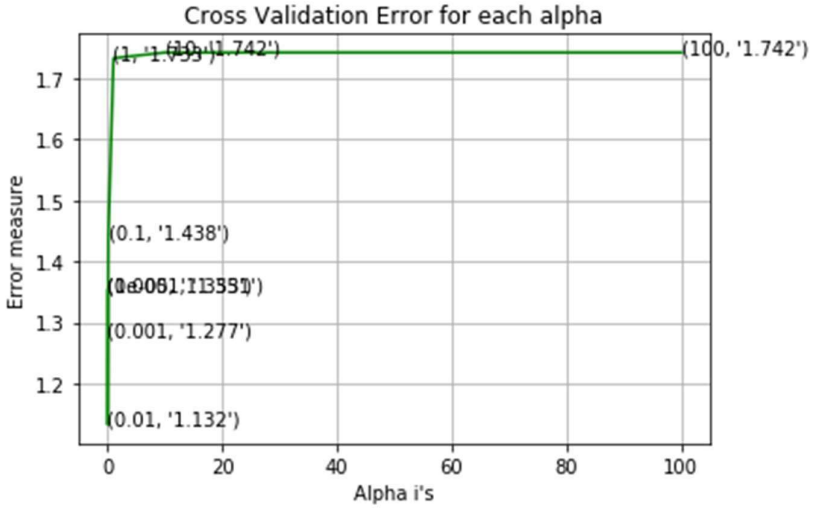
The model was fitted using the best lambda, and test data was applied to the model. The log-loss value is approximately k-NN at 1.048. However, the percentage of incorrectly categorised points is just 34.77 percent (smaller than NB and K-NN). LR is superior to k-NN and NB because it is interpretable and has fewer incorrectly categorised points.

Log loss and incorrectly categorised scores rise in the absence of class balancing. As a result, we employ class balancing.

Approach 4:

Svm:

As the Linear SVM (with class balancing) is interpretable and excels with high dimension data, we choose it. We are unable to apply RBF Kernel SVM since it is not interpretable. Now, we utilise the model's training data and the CV data to determine the optimal hyper-parameter (C).



We apply test data to the model and fit it with the highest C. The current log-loss value is 1.06 (close to LR), which is significantly lower than the random model. Here, there are 36.47 percent (more than LR) of all instances that were incorrectly classified. Due to the class balance strategy, lesser classes performed well.

Approach 5:

5. Random Forest:

5.1) One-hot encoding: Decision Trees typically perform well with low-dimension data. It can also be understood. Changes to the Random Forest Classifier's base learner and maximum depth yield the best base learner and maximum depth values of 2000 and 10, respectively.After that, test data is applied to the model that has been fitted with the best hyper-parameters. The overall number of misclassified points is 36.84 percent, which is more than LR, and the resulting log loss value is 1.097, which is close to LR.

5.2) Response Coding: We discover that the optimal base learners and maximum depth in the Random Forest Classifier are 100 and 5, respectively. After fitting the model using the best hyper-parameters, we discovered that the train log loss was 0.052 and the CV log loss was 1.325, indicating that the model is even more overfitted. Most optimal hyper-parameters. We avoid using RF+Response Coding because of this.

Approach 6:

6. Stacking Classifier:

We kept LR as the meta classifier and stacked three classifiers: LR, SVM, and NB. The training data is now applied to the model, and the CV data is utilised to identify the ideal hyper-parameter. We apply the test data to the model and fit it using the best hyperparameter. The log-loss value is 1.08, which is significantly less than the value for the random model. Additionally, we learn that 36.2 percent of all cases were incorrectly classified in this place. Even though we utilised a sophisticated model in this case, we still received results that were very similar to LR. Furthermore, we are aware that the stacking classifier cannot be understood.

MODELS	TRAIN LOG LOSS	CV LOG LOSS	TEST LOG LOSS	PERCENTAGE OF MISCLASSIFIED POINTS
NB(ONE HOT ENCODING)	0.90	1.27	1.21	39.84
KNN(RESPONSE CODING)	0.705	1.130	1.002	39.47
LR+BALANCING(ONE HOT)	0.614	1.143	1.048	34.77
LR+IMBALANCE(ONE HOT)	0.628	1.185	1.054	36.28
SVM+BALANCING(ONE HOT)	0.739	1.132	1.063	36.47
RF(ONE HOT)	0.703	1.192	1.097	36.84
RF(RESPONSE CODING)	0.052	1.325	1.211	46.8
STACKING CLASSIFIER (NB,LR,SVM)	0.663	1.177	1.081	36.24

It is evident from the following table that the train and CV log loss (almost 20 times) have changed significantly due to response coding (RF). This indicates that the model is over-fitted, therefore we discard it. The log loss values in the stacking classifier (an ensemble) are almost identical to those in LR+Balancing. According to the above table, LR+Balancing is the model that best satisfies our company's needs or real-world requirements, such as interpretability and having better log loss values than other models.

5) RESULTS

We investigated using TF-IDF Vectorizer instead of CountVectorizer (for one-hot encoding of features), and the results are rather decent.

MODELS	TRAIN LOG LOSS	CV LOG LOSS	TEST LOG LOSS	PERCENTAGE OF MISCLASSIFIED POINTS
NB(ONE HOT ENCODING)	0.906	1.229	1.225	41
LR+BALANCING(ONE HOT ENCODING)	0.57	1.114	1.113	36.8
LR+WITHOUT BALANCING(ONE HOT)	0.56	1.13	1.12	36.09
LINEAR SVM(ONE HOT)	0.68	1.15	1.19	37.78
RANDOM FOREST(ONE HOT)	0.64	1.12	1.16	37.7
RANDOM FOREST(RESPONSE CODING)	0.05	1.38	1.44	48.8
STACKING CLASSIFIER	0.63	1.16	1.19	38.9
MAXIMUM VOTING CLASSIFIER	0.84	1.16	1.19	40.1

We can see that TF-IDF Vectorizer's outcomes are marginally superior to Count Vectorizer's. Additionally, we can see that Random Forest with response coding is overfitted because to the significant discrepancy between train and CV loss whereas Logistic Regression performs well in comparison to other techniques.

6) KEY COMPONENTS

1) Multi-Log Loss:

Multi-log loss, or multi-class logarithmic loss, is a key metric for evaluating the accuracy of personalized cancer prediction models. It quantifies the dissimilarity between predicted and actual class probabilities. Lower multi-log loss values indicate better model performance in assigning cancer subtypes to patients, enhancing the precision of individualized predictions.

2) Laplace Smoothing:

Laplace smoothing is crucial for managing sparse data in cancer prediction. It adjusts probability estimates by adding a small constant to each class probability, preventing zero probabilities and reducing overfitting. This regularization technique helps ensure robust and reliable personalized predictions in scenarios with limited patient samples.

3) Response Coding:

Response coding is a feature engineering method used in personalized cancer prediction. It encodes categorical features, such as genetic mutations or cancer types, into numeric representations, aiding machine learning algorithms in making predictions specific to individual patient profiles.

4) Univariate Analysis:

Univariate analysis involves the examination of individual predictor variables in personalized cancer prediction. It helps identify the significance of each feature in isolation, contributing to the selection of relevant input variables and improving model accuracy.

5) Text Vectorization:

Text vectorization transforms textual patient records and medical literature into numerical data, making it suitable for machine learning models. This is crucial for incorporating textual information like clinical notes and research findings into personalized cancer prediction, enriching the input data and improving prediction accuracy.

6) SGDClassifier:

Stochastic Gradient Descent (SGD) classifiers are valuable tools for training personalized cancer prediction models. They offer efficient optimization techniques, enabling the models to adapt quickly to large datasets and achieve better generalization performance for individual patient profiles.

7) Calibrated Classifier:

Calibrated classifiers are employed to refine the probability estimates produced by prediction models. In personalized cancer prediction, they help calibrate the model's confidence scores, ensuring more reliable risk assessments and treatment recommendations for each patient.

8) MultinomialNB:

Multinomial Naïve Bayes (MultinomialNB) is an algorithm used in personalized cancer prediction, particularly for text-based data. It's well-suited for classifying patient records or research papers into cancer subtypes, leveraging the Naïve Bayes framework to calculate probabilities and make personalized predictions based on textual information.

EXPERIMENT NO: 3	DATE:
EXPERIMENT NAME: Microsoft Malware detection	

1. BUSINESS PROBLEM/PROBLEM STATEMENT

Over the recent years, the malware sector has experienced significant expansion, with cybercriminal syndicates making substantial investments in evasive technologies to outsmart conventional safeguards. This has compelled anti-malware groups and communities to develop more resilient software for the detection and elimination of these threats. A fundamental aspect of safeguarding computer systems against malware attacks is the ability to determine whether a particular file or software is malicious.

Hence, the objective of this case study is to detect whether the given software is a malware or not and detect the malware type.

2. MACHINE LEARNING PROBLEM FORMULATION

About Data:

In the given dataset for each malware, we have given the two types of files. They are:

1. **.ASM file:** The ASM files, short for assembly language, represents a code file. Suppose you are tasked with coding in C++, and the C++ compiler transforms your code into an ASM file. This ASM file is then processed by an assembler, which translates it into a binary file consisting of only 0s and 1s. Within .asm files, you encounter keywords such as 'pop,' 'jump,' 'push,' 'move,' and others

2. **.bytes files** : The binary file (the raw data without the PE header) contains hexadecimal values, with each value falling within the range of 00 to FF, representing the binary content of the file

Here we have a total of 200GB data where 50 GB data of bytes files and 150 GB is .asm file.So the name of the nine malware is as follow:

1. Ramnit
2. Lollipop
3. Kelihos_ver3
4. Vundo
5. Simda
6. Tracur
7. Kelihos_ver1
8. Obfuscator.ACY
9. Gatak

Sample .asm file :

```
.text:00401030 56                                  push    esi
.text:00401031 8B F1                               mov     esi, ecx
.text:00401043 74 09                               jz      short
loc_40104E
.text:00401045 56                                  push    esi
.text:00401046 E8 6C 1E 00 00    call      ??3@YAXPAX@Z; operator
delete(void *)
.text:0040104B 83 C4 04                            add     esp, 4
.text:0040104E
.text:0040104E            loc_40104E:          ; CODE XREF:
.text:00401043j
.text:0040104E 8B C6                               mov     eax, esi
.text:00401050 5E                                  pop     esi
.text:00401051 C2 04 00                                    retn
```

Sample .bin file

```
6A 04 68 00 10 00 00 68 66 3E 18 00 6A 00 FF 15
D0 63 52 00 8B 4C 24 04 6A 00 6A 40 68 66 3E 18
00 50 89 01 FF 15 9C 63 52 00 50 FF 15 CC 63 52
00 B8 07 00 00 00 C2 04 00 CC CC CC CC CC CC CC
B8 FE FF FF FF C3 CC CC CC CC CC CC CC CC CC CC
B8 09 00 00 00 C3 CC CC CC CC CC CC CC CC CC CC
B8 01 00 00 00 C3 CC CC CC CC CC CC CC CC CC CC
B8 FD FF FF FF C2 04 00 CC CC CC CC CC CC CC CC
8B 4C 24 04 8D 81 D6 8D 82 F7 81 F1 60 4F 15 0B
23 C1 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC
8B 4C 24 04 8B C1 35 45 CF 3F FE 23 C1 25 BA 3D
C5 05 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC
8B 4C 24 04 B8 1F CD 98 AE F7 E1 C1 EA 1E 69 D2
FA C9 D6 5D 56 8B F1 2B F2 B8 25 95 2A 16 F7 E1
8B C1 2B C2 D1 E8 03 C2 C1 E8 1C 69 C0 84 33 73
```

Given the dataset, the goal is to categorize files into one of the nine malware classes. This task is translated into a multiclass classification problem. For evaluating model performance, we employ metrics such as multiclass log loss, precision, and recall, providing a comprehensive evaluation of model accuracy.

This project delves into the critical domain of malware detection, aiming to provide a comprehensive understanding of the problem, the data, and the application of machine learning techniques in addressing it. Hence, we use multiclass log loss function to find the probability of each class.

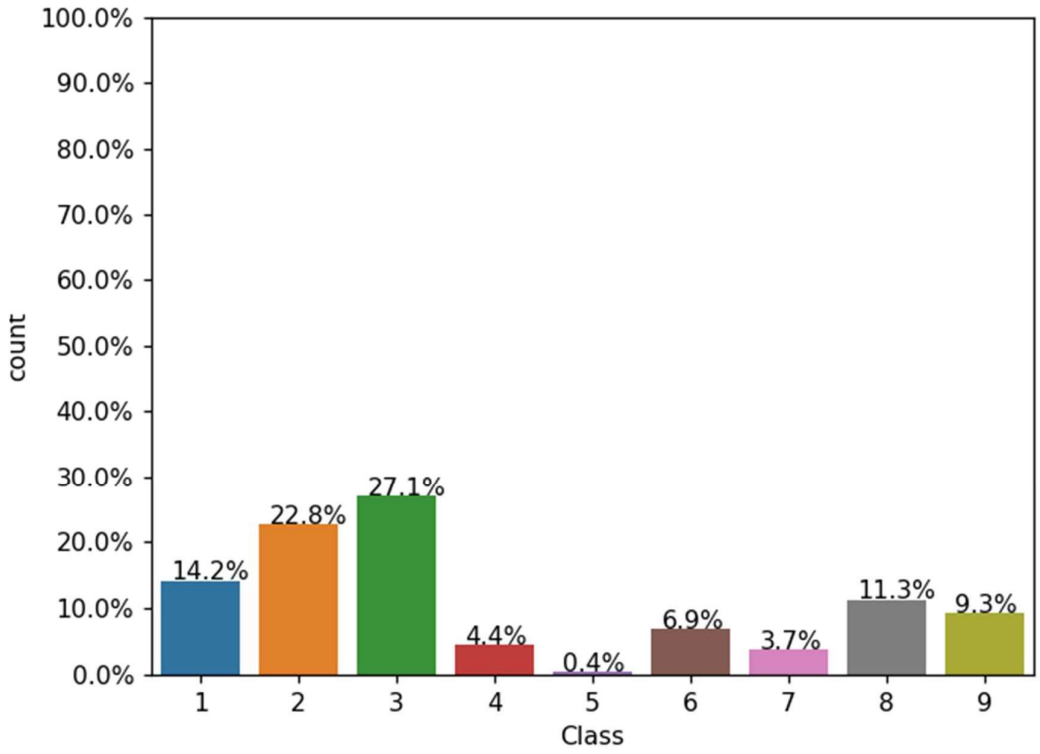
$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

Where,

N	*No of Rows in Test set*
M	*No of Fault Delivery Classes*
Y_{ij}	*1 if observation belongs to Class j; else 0*
P_{ij}	*Predicted Probability that observation belong to Class j*

3. EXPLORATORY DATA ANALYSIS

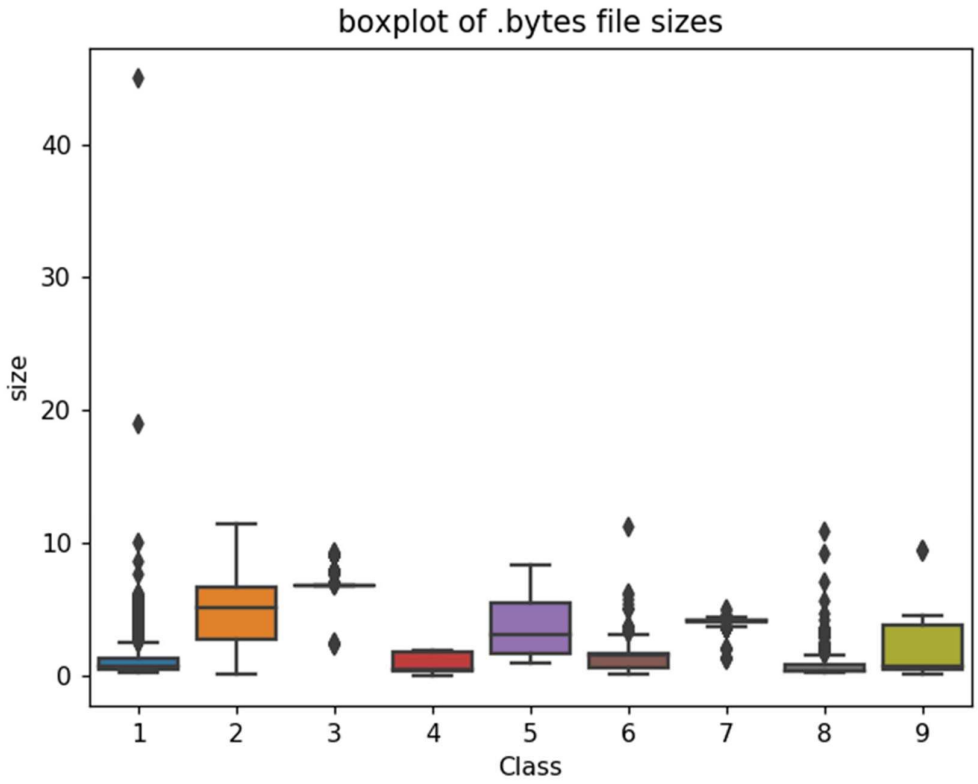
In the process of Exploratory Data Analysis (EDA), the first step involves segregating the bytes files from the ASM files. It's important to consider that the dataset is imbalanced. As an initial step, histograms are generated to visualize the distribution of class labels.



From this histogram of the class label, we can say that class 5 has very very few-data—points, and classes 2 and 3 have a good amount of data–points-available.

Feature extraction

The feature extraction process begins by examining the .bytes files, with a focus on the file size as a potential feature. A box plot is generated to assess the utility of file size as a discriminative feature for class label determination. If the box plots of all class labels exhibit substantial overlap, it can be inferred that file size alone is not a valuable feature for distinguishing between the classes.

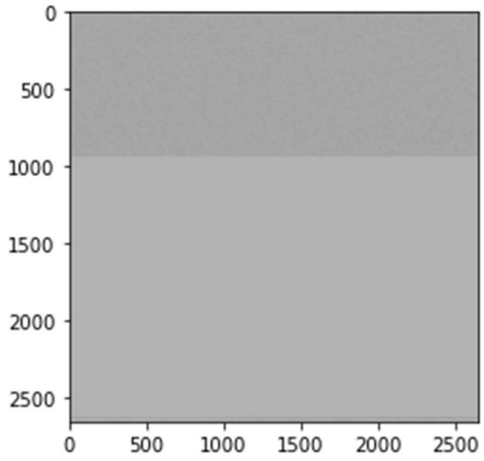


Based on the analysis of the box plot, it is evident that the file size feature proves to be useful in determining the class labels.

Moving on to the second aspect of feature extraction, .bytes files are known to contain hexadecimal values within the range of 00 to FF. As a result, all hexadecimal values within this range are considered as input features for further analysis.

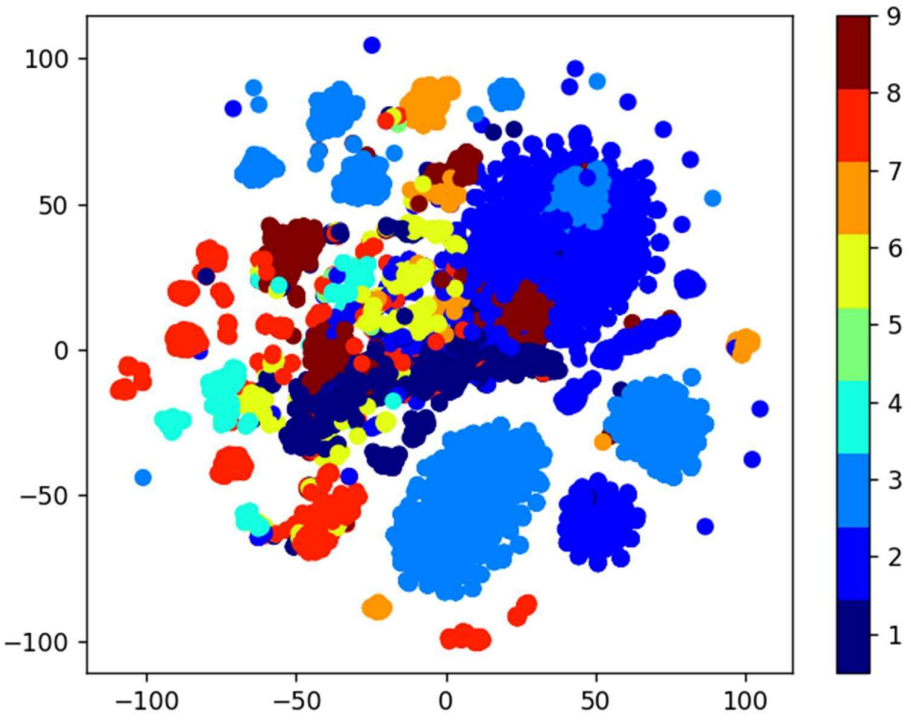
00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,. . .

A pixel-based feature extraction approach is applied to the .bytes files. This involves converting each file into an image representation, and from that image, the first 100 pixels are selected to serve as input values for further analysis.



For **feature encoding**, a bi-gram Bag-of-Words (BOW) approach is employed. This entails processing each .bytes file and counting the occurrences of individual features, such as 00, 01, 02, and so on. In addition, bi-grams are generated by pairing adjacent features, such as 00 01, 01 02, 02 03, and their occurrences are counted as well. This BOW encoding technique is applied to create feature representations for analysis.

Multivariate analysis is performed on the features extracted from the .bytes files. This analysis involves the use of t-SNE (t-distributed Stochastic Neighbor Embedding), which is a dimension-reduction technique. The data is initially transformed into a 2-dimensional space using t-SNE. This reduction in dimensionality aids in gaining insights into the usefulness of the features for effectively classifying the class labels.



The t-SNE analysis indicates that the features are indeed valuable for classifying the class labels, confirming their effectiveness in the classification task.

4. MACHINE LEARNING MODELS

1. Random Model:

- In this step, a random model is applied to the dataset. The random model predicts class labels randomly, and the multiclass log loss on cross-validation (cv) data is found to be 2.45. This high log loss value serves as a baseline, indicating the worst possible performance for the model. The objective is to minimize the log loss value.

2. K-Nearest Neighbours (KNN) on .bytes Files:

- KNN is employed as a neighbourhood-based classification method on the .bytes files.

- Train and test log loss values are 0.078 and 0.245, respectively.

- A substantial difference between train and test loss indicates overfitting to the training data.

- Precision indicates well-classification for most classes, except for class 5, which has limited data points.

3. Logistic Regression on .bytes Files:

- Logistic regression is applied to the .bytes files.

- Train and test log loss values are 0.1918 and 0.3168.

- The model exhibits a slight overfitting tendency.

- Class 5 is not well-predicted due to the limited amount of data for this class.

4. Random Forest on .bytes Files:

- Random Forest (RF) is applied to the .bytes data, and it performs well.

- Train and test log loss values are 0.0342 and 0.0971, indicating good model fitting.

- Precision suggests that most class labels, including class 5, are well-predicted.

- Recall confirms correct classifications for most class labels.

5. XGBoost on .bytes Files:

- XGBoost, an ensemble model-based technique, is applied to the data.

- Train and test log loss values are 0.022 and 0.079, representing relatively good log loss values compared to other models.

- Precision demonstrates accurate predictions for most class labels, including class 5.

- Recall further affirms the correct classification of class labels.

These steps encompass the application of various models to the dataset, each with its respective performance metrics and characteristics. The goal is to identify the model that provides the best classification performance while considering overfitting and limitations in data points for certain classes.

Extract Feature from .asm files

Now we extract the feature from the .asm file, All the files make up about 150 GB.

Here we extracted **52 features** from all the .asm files which are important.

This file contains registers, keywords, opcodes, prefixes from these 4 parts of the file we extract a total of 52 features, and we include the file size–as a feature, so in total, we have 53 features here.

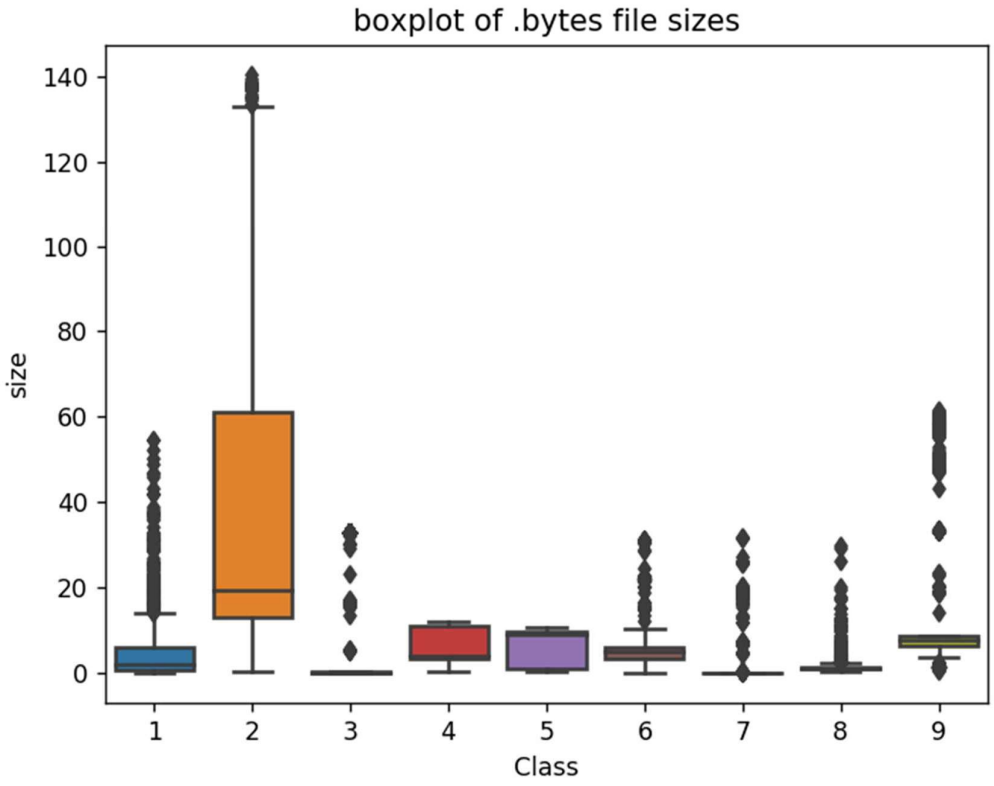
Prefixes=
['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE']

opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']

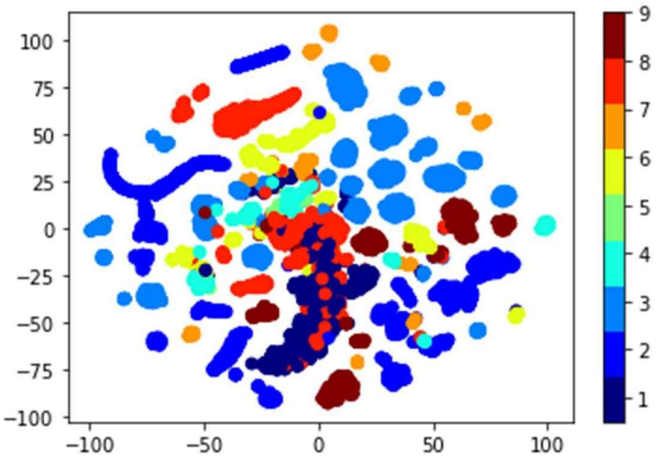
```
keywords = ['.dll','std::',':dword']
registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
```

Distribution of .asm file sizes

The distribution of .asm file sizes is visualized through the use of a box plot. This analysis is performed to assess whether the file size serves as a valuable feature for determining class labels, as indicated by the spread and distribution of file sizes among the different classes.



Multivariate analysis is conducted on the features extracted from the .asm files, similar to the approach used for the .bytes files. In this case, t-SNE (t-distributed Stochastic Neighbor Embedding) is applied to reduce the dimensionality and visualize the data in a 2-dimensional space. This analysis aims to determine whether the 53 features are effective in separating data points based on class labels.



The plot resulting from the t-SNE analysis of the .asm file features reveals distinct clusters for different classes. This observation supports the conclusion that these features are indeed useful for separating class labels. The clear separation of data points into distinct clusters demonstrates the effectiveness of these features in distinguishing between the classes.

Apply the model on.ASM files

1. Random Model on .asm Files:

- A random model is applied to the .asm files to establish a high benchmark for log loss.

- The log loss on the test data for the random model is found to be 2.493.

2. K-Nearest Neighbors (KNN) on .asm Files:

- KNN is applied to the .asm dataset, utilizing a neighborhood-based approach.

- Train and test log loss values are observed to be 0.0476 and 0.089, respectively, with a significant difference indicating overfitting.

- Precision indicates well-classified data for all classes except class 5, likely due to limited data for this class.

3. Logistic Regression on .asm Files:

- Logistic regression is employed on the .asm files.

- Train and test log loss values are 0.781 and 0.742.

- Log loss values suggest a well-fitted model, but it struggles to predict class 5 due to data scarcity.

- Recall indicates that it misclassifies classes 5, 6, 7, and 9.

4. Random Forest on .bytes Files:

- Random Forest (RF) is applied to the data with favorable results.

- Train and test log loss values are 0.0271 and 0.0462.

- Precision demonstrates accurate predictions for most class labels, including class 5.

- Recall affirms correct classification for most class labels, and log loss values indicate a well-fitted model.

5. XGBoost on .bytes Files:

- XGBoost is applied to the data, producing excellent results.

- Precision suggests accurate predictions for most class labels, including class

- Train and test log loss values are 0.0241 and 0.0371, indicating good performance.

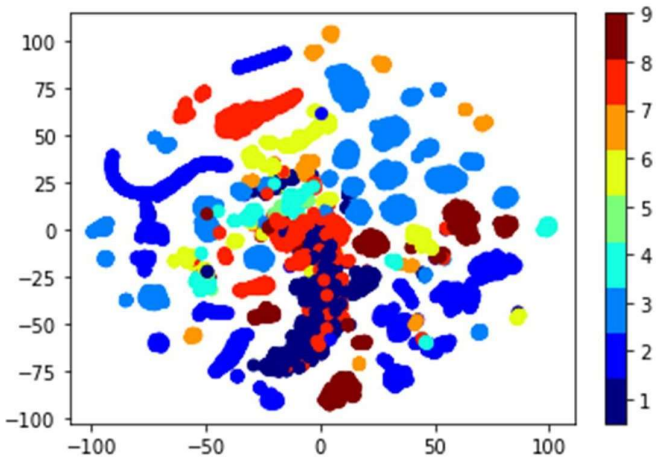
- Recall further confirms correct classification for most class labels, and the log loss values suggest a well-fitted model, outperforming RF.

Final Model:

- Both .asm and .bytes files are merged, resulting in a total of 66k features.

- The XGBoost model, which performed well individually, is chosen as the final model for this combined dataset.

- t-SNE is applied to evaluate the suitability of the 66k features for separating class labels in the multivariate space, providing insights into the data's separability.



The multivariate analysis conducted on the combined .asm and .bytes file features indicates that these features are indeed highly valuable for separating data points. t-SNE is applied to the combined dataset, and by adjusting perplexity values, it's observed that the data forms distinct and meaningful clusters. This reinforces the

conclusion that the features are effective in separating data points, making them essential for the classification task.

Random Forest Classifier on Final Features:

- Random Forest (RF) is applied to the final dataset, and it performs effectively.

- Train and test log loss values are 0.03143 and 0.09624.

- Precision indicates accurate predictions for most class labels, including class 5.

- Recall further confirms correct classification for most class labels, and log loss values suggest a well-fitted model, indicating that RF works well for this dataset.

LGBMClassifier on Final Features:

- Although XGBoost was the winner model for both .asm and .bytes files, due to faster computing times, LightGBM (LGBM) is applied to the final dataset.

- LGBM is applied to the final data and produces similar results.

- Precision shows accurate predictions for most class labels, including class 5.

- Log loss values indicate a well-fitted model, with a test loss of 0.01, which is even better than RF.

- Therefore, LGBM/XGBoost is demonstrated to be effective for this dataset.

In summary, both Random Forest and LGBM/XGBoost models perform well on the final features, with LGBM exhibiting slightly faster performance. These models are effective for this classification task.

5.RESULTS

File Name	Model Name	Test Loss
Bytes File	Random Model	2.46
Bytes File	KNN	0.497
Bytes File	LR	0.316
Bytes File	RF	0.097
Bytes File	XGBOOST	0.069
ASM File	Random Model	2.49
ASM File	KNN	0.101
ASM File	LR	0.741
ASM File	RF	0.0467
ASM File	XGBOOST	0.037
Bytes + Asm Files	RF	0.091
Bytes + Asm Files	XGBOOST/LGBM	0.017

Machine learning models, including Logistic Regression, Random Forest, K-Nearest Neighbors, XGBoost, and LGBM, are evaluated for malware classification.

Specialized 53 keywords are identified in the .asm files and used as features.

The XGBoost or LGBM model yields the lowest test loss, achieving a log loss of 0.01, signifying exceptional performance in malware classification.

6) KEY COMPONENTS

1) ASM File:

In Microsoft malware detection, ASM (Assembly) files play a critical role in analysing potential threats. ASM files contain low-level, human-readable instructions that allow security researchers to delve into the inner workings of suspicious software. By disassembling and inspecting ASM code, analysts can uncover malware's functionality, identify vulnerabilities, and understand its evasion techniques. This process aids in crafting effective countermeasures and enhancing the accuracy of malware detection algorithms.

2) t-SNE:

t-SNE (t-Distributed Stochastic Neighbor Embedding) is employed in Microsoft's malware detection to visualize high-dimensional data and reveal patterns in feature space. It aids in clustering malware samples based on behavioral or code similarities, making it easier to categorize and understand malware families. t-SNE can simplify the exploration of large datasets, assisting security experts in identifying novel threats and making informed decisions about threat mitigation strategies.

3) Random Search for Hyperparameter Tuning:

Random search is a hyperparameter optimization technique used in Microsoft malware detection models. By randomly selecting hyperparameters, it efficiently explores the hyperparameter space, saving time and computational resources compared to grid search. This method helps fine-tune machine learning algorithms, such as random forests, XGBoost, and logistic regression, improving their performance and adaptability to evolving malware threats.

4) Penalty in Logistic Regression for Malware Detection:

In logistic regression models for malware detection at Microsoft, the choice of penalty terms, such as L1 (Lasso) or L2 (Ridge), is crucial. These penalties help prevent overfitting and guide the model to select the most relevant features while suppressing noise. The appropriate choice of penalty can significantly impact the model's ability to generalize well and detect both known and novel malware variants accurately.

5) Hyperparameter Tuning:

Hyperparameter tuning is an essential practice in Microsoft's malware detection framework. For random forest, parameters like the number of trees, maximum depth, and feature selection criteria are adjusted to optimize model performance. XGBoost models benefit from tuning parameters such as learning rate, tree depth, and regularization. In logistic regression, hyperparameters like the penalty term, solver, and class weights are fine-tuned to improve the model's ability to identify malware while minimizing false positives. This iterative process ensures that the detection models are continually refined and remain effective against evolving threats.