Introduction to Data Science
(CSE – 0327)

"Predicting failures and RUL of Metro Trains using MetroPT-3 Dataset"

By: -

Jalaj Rastogi	21UCC054
Ashwin Singh	21UCC128
Harsh Dhuppar	21UCC045
Devansh Chechani	21UCS059

Course Instructors: -

Dr. Lal Upendra Singh
Dr. Subrat K. Dash
Dr. Aloke Dutta

LNMIIT
The LNM Institute of
Information Technology

Department of Computer Science Engineering
The LNM Institute of Information Technology

November 2023

TABLE OF CONTENTS

PROJECT OBJECTIVE

The aim of this project to select a dataset from the UCI Machine Learning Repository and perform the following steps:

1. Data pre-processing and visualization.
2. Apply ML Classification Algorithms on the processed data.
3. Output the results of the testing set and compare them.

Language Used:

Python is used to ally us with our work on the dataset and the following Python Libraries are used to perform the tasks:

- scikit_learn
- matplotlib
- seaborn
- numpy
- pandas

INTRODUCTION TO DATASET

The dataset was collected to support the development of predictive maintenance, anomaly detection, and remaining useful life (RUL) prediction models for compressors using deep learning and machine learning methods. It consists of multivariate time series data obtained from several analogue and digital sensors installed on the compressor of a train. The data span between February and August 2020 and includes 15 signals, such as pressures, motor current, oil temperature, and electrical signals of air intake valves. The monitoring and logging of industrial equipment events, such as temporal behaviour and fault events, were obtained from records generated by the sensors. The data were logged at 1Hz by an onboard embedded device.

The dataset consists of:

- Data Set Characteristics: Multivariate, Tabular, Time-Series
- Subject Area: Computer Science
- Associated Tasks: Classification
- No. of Instances: 1516948
- No. of Features: 15
- Feature Type: Real
- Missing Values: No
- Date Donated: 21st March 2023

Attribute Information:

1. **TP2 (bar) –** the measure of the pressure on the compressor.
2. **TP3 (bar) –** the measure of the pressure generated at the pneumatic panel.
3. **H1 (bar) –** the measure of the pressure generated due to pressure drop when the discharge of the cyclonic separator filter occurs.
4. **DV pressure (bar) –** the measure of the pressure drops generated when the towers discharge air dryers; a zero reading indicates that the compressor is operating under load.
5. **Reservoirs (bar) –** the measure of the downstream pressure of the reservoirs, which should be close to the pneumatic panel pressure (TP3).
6. **Motor Current (A) –** the measure of the current of one phase of the three-phase motor; it presents values close to 0A - when it turns off, 4A - when working offloaded, 7A - when working under load, and 9A - when it starts working.
7. **Oil Temperature (ºC) –** the measure of the oil temperature on the compressor.
8. **COMP -** the electrical signal of the air intake valve on the compressor; it is active when there is no air intake, indicating that the compressor is either turned off or operating in an offloaded state.

9. **DV electric –** the electrical signal that controls the compressor outlet valve; it is active when the compressor is functioning under load and inactive when the compressor is either off or operating in an offloaded state.
10. **TOWERS –** the electrical signal that defines the tower responsible for drying the air and the tower responsible for draining the humidity removed from the air; when not active, it indicates that tower one is functioning; when active, it indicates that tower two is in operation.
11. **MPG –** the electrical signal responsible for starting the compressor under load by activating the intake valve when the pressure in the air production unit (APU) falls below 8.2 bar; it activates the COMP sensor, which assumes the same behaviour as the MPG sensor.
12. **LPS –** the electrical signal that detects and activates when the pressure drops below 7 bars.
13. **Pressure Switch -** the electrical signal that detects the discharge in the air-drying towers.
14. **Oil Level –** the electrical signal that detects the oil level on the compressor; it is active when the oil is below the expected values.
15. **Caudal Impulse –** the electrical signal that counts the pulse outputs generated by the absolute amount of air flowing from the APU to the reservoirs.

Dataset Citation (all the relevant information about the dataset is obtained from the same source)**:**

- Davari,Narjes, Veloso,Bruno, Ribeiro ,Rita, and Gama,Joao. (2023). MetroPT-3 Dataset. UCI Machine Learning Repository. https://doi.org/10.24432/C5VW3R.

DATA ANALYSIS

Importing Libraries

We start the project by importing the required libraries that will be used to perform specific tasks: -

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV
    from sklearn.preprocessing import StandardScaler, MinMaxScaler
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.svm import SVC
    from sklearn import svm
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.linear_model import LogisticRegression
    from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
    from sklearn.compose import ColumnTransformer
    from sklearn.pipeline import Pipeline
```

1. **numpy:** Numerical computing library that provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
2. **pandas:** Data manipulation and analysis library offering data structures like DataFrames for efficient handling and manipulation of structured data.
3. **matplotlib:** 2D plotting library for creating static, animated, and interactive visualizations in Python.
4. **seaborn:** Statistical data visualization library based on matplotlib; it provides a high-level interface for drawing attractive and informative statistical graphics.
5. **scikit-learn:** Machine learning library in Python that provides simple and efficient tools for data analysis and modeling, including classification, regression, clustering, and more.

Understanding the Dataset

- The data is first imported and stored in a pandas dataframe so that it can be worked upon.
- To understand the dataset we are working with, we use **data.info()** which returns the information about the dataset in a concise manner (count, missing values and the data type of the features in the dataset).

```
[ ]  data = pd.read_csv("metropt3_data.csv")
     print(data.info())

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 1516948 entries, 0 to 1516947
     Data columns (total 17 columns):
      #   Column           Non-Null Count    Dtype
     ---  ------           --------------    -----
      0   Unnamed: 0       1516948 non-null  int64
      1   timestamp        1516948 non-null  object
      2   TP2              1516948 non-null  float64
      3   TP3              1516948 non-null  float64
      4   H1               1516948 non-null  float64
      5   DV_pressure      1516948 non-null  float64
      6   Reservoirs       1516948 non-null  float64
      7   Oil_temperature  1516948 non-null  float64
      8   Motor_current    1516948 non-null  float64
      9   COMP             1516948 non-null  float64
      10  DV_eletric       1516948 non-null  float64
      11  Towers           1516948 non-null  float64
      12  MPG              1516948 non-null  float64
      13  LPS              1516948 non-null  float64
      14  Pressure_switch  1516948 non-null  float64
      15  Oil_level        1516948 non-null  float64
      16  Caudal_impulses  1516948 non-null  float64
     dtypes: float64(15), int64(1), object(1)
     memory usage: 196.7+ MB
     None
```

- We can use **data.head()** command to get a quick and preliminary view of the data. This command returns the top *n* instances. The default value of *n* is 5 but we have set it to 10 here.

	Unnamed: 0	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG
0	0	2020-02-01 00:00:00	-0.012	9.358	9.340	-0.024	9.358	53.600	0.0400	1.0	0.0	1.0	1.0
1	10	2020-02-01 00:00:10	-0.014	9.348	9.332	-0.022	9.348	53.675	0.0400	1.0	0.0	1.0	1.0
2	20	2020-02-01 00:00:19	-0.012	9.338	9.322	-0.022	9.338	53.600	0.0425	1.0	0.0	1.0	1.0
3	30	2020-02-01 00:00:29	-0.012	9.328	9.312	-0.022	9.328	53.425	0.0400	1.0	0.0	1.0	1.0
4	40	2020-02-01 00:00:39	-0.012	9.318	9.302	-0.022	9.318	53.475	0.0400	1.0	0.0	1.0	1.0
5	50	2020-02-01 00:00:49	-0.012	9.306	9.290	-0.024	9.308	53.500	0.0400	1.0	0.0	1.0	1.0
6	60	2020-02-01 00:00:59	-0.012	9.296	9.280	-0.024	9.298	53.375	0.0400	1.0	0.0	1.0	1.0
7	70	2020-02-01 00:01:09	-0.014	9.286	9.270	-0.024	9.286	53.550	0.0400	1.0	0.0	1.0	1.0
8	80	2020-02-01 00:01:19	-0.012	9.276	9.258	-0.022	9.276	53.425	0.0400	1.0	0.0	1.0	1.0
9	90	2020-02-01 00:01:29	-0.012	9.264	9.248	-0.022	9.264	53.375	0.0400	1.0	0.0	1.0	1.0

LPS	Pressure_switch	Oil_level	Caudal_impulses
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0
0.0	1.0	1.0	1.0

- Further, we use **data.describe()** to understand the ranges and value distribution of our attributes. As all the features of the dataset are numerical, this helps us get a fair idea about the dataset.

  The **describe()** command provides a summary of basic statistical values for the attributes present in the DataFrame. The values we can observe for each column are count, mean, standard deviation, minimum, maximum, and all quartiles. The values play a great role in the decision-making of pre-processing steps. From these, we can judge whether our data needs outlier handling or normalization, or any other such processing.

```
[ ]  print(data.describe().round(2))
     print(data.columns)
```

	Unnamed: 0	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric
count	1516948.00	1516948.00	1516948.00	1516948.00	1516948.00	1516948.00	1516948.00	1516948.00	1516948.00	1516948.00
mean	7584735.00	1.37	8.98	7.57	0.06	8.99	62.64	2.05	0.84	0.16
std	4379053.12	3.25	0.64	3.33	0.38	0.64	6.52	2.30	0.37	0.37
min	0.00	-0.03	0.73	-0.04	-0.03	0.71	15.40	0.02	0.00	0.00
25%	3792367.50	-0.01	8.49	8.25	-0.02	8.49	57.78	0.04	1.00	0.00
50%	7584735.00	-0.01	8.96	8.78	-0.02	8.96	62.70	0.04	1.00	0.00
75%	11377102.50	-0.01	9.49	9.37	-0.02	9.49	67.25	3.81	1.00	0.00
max	15169470.00	10.68	10.30	10.29	9.84	10.30	89.05	9.30	1.00	1.00

Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses
1516948.00	1516948.00	1516948.00	1516948.00	1516948.00	1516948.00
0.92	0.83	0.00	0.99	0.90	0.94
0.27	0.37	0.06	0.09	0.29	0.24
0.00	0.00	0.00	0.00	0.00	0.00
1.00	1.00	0.00	1.00	1.00	1.00
1.00	1.00	0.00	1.00	1.00	1.00
1.00	1.00	0.00	1.00	1.00	1.00
1.00	1.00	1.00	1.00	1.00	1.00

DATA PRE-PROCESSING

According to the documentation provided for the dataset, the following have already been performed on the data: -

- Data Segmentation
- Normalization
- Feature Extraction

To further process the data, we perform the following steps: -

Labelling the Dataset

- The initial dataset is unlabelled. The first column(Unnamed:0) is dropped using **data.drop("Unnamed: 0", axis=1)** as this column does not affect the target value and there is no use of this column in determining the failure rate and detecting anomalies.

```
[ ]  data = data.drop("Unnamed: 0", axis=1)

[ ]  print(data.columns)

     Index(['timestamp', 'TP2', 'TP3', 'H1', 'DV_pressure', 'Reservoirs',
            'Oil_temperature', 'Motor_current', 'COMP', 'DV_eletric', 'Towers',
            'MPG', 'LPS', 'Pressure_switch', 'Oil_level', 'Caudal_impulses'],
           dtype='object')
```

Correcting Datatypes

- The datatype of the feature *timestamp* is converted to datetime from string as time-series based computations can't be performed on string datatype.

```
[ ]  import datetime

[ ]  print(f"Data type of timestamp is {type(data.timestamp[0])}")

     Data type of timestamp is <class 'str'>

[ ]  data["timestamp"] = data["timestamp"].apply(pd.to_datetime, format = "%Y-%m-%d %H:%M:%S")
     print(f"New data type of timestamp is {type(data.timestamp[0])}")

     New data type of timestamp is <class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

Determining the Failure Rate

- A new feature *status* is created and added to the dataframe.

timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG	LPS
2020-02-01 00:00:00	-0.012	9.358	9.340	-0.024	9.358	53.600	0.0400	1.0	0.0	1.0	1.0	0.0
2020-02-01 00:00:10	-0.014	9.348	9.332	-0.022	9.348	53.675	0.0400	1.0	0.0	1.0	1.0	0.0
2020-02-01 00:00:19	-0.012	9.338	9.322	-0.022	9.338	53.600	0.0425	1.0	0.0	1.0	1.0	0.0
2020-02-01 00:00:29	-0.012	9.328	9.312	-0.022	9.328	53.425	0.0400	1.0	0.0	1.0	1.0	0.0
2020-02-01 00:00:39	-0.012	9.318	9.302	-0.022	9.318	53.475	0.0400	1.0	0.0	1.0	1.0	0.0

Pressure_switch	Oil_level	Caudal_impulses	status
1.0	1.0	1.0	0
1.0	1.0	1.0	0
1.0	1.0	1.0	0
1.0	1.0	1.0	0
1.0	1.0	1.0	0

The data provided for the same is binary where 1 represents that failure has occurred and 0 represents that there was no failure. Failure is judged and set to 1 according to the table provided in the documentation of the dataset. All the tuples with timestamps falling within the start and end times are labelled as failures.

Nr.	Start Time	End Time	Failure	Severity	Report
#1	4/18/2020 0:00	4/18/2020 23:59	**Air leak**	High stress	
#1	5/29/2020 23:30	5/30/2020 6:00	**Air Leak**	High stress	Maintenance on 30Apr at 12:00
#3	6/5/2020 10:00	6/7/2020 14:30	**Air Leak**	High stress	Maintenance on 8Jun at 16:00
#4	7/15/2020 14:30	7/15/2020 19:00	**Air Leak**	High stress	Maintenance on 16Jul at 00:00

```
[ ] def to_datetime(xs):
        result = []
        format = "%Y-%m-%d %H:%M:%S"
        for x in xs:
            result.append(pd.to_datetime(x,format = format))
        return result

    failure_start = to_datetime(["2020-04-18 00:00:00", "2020-05-29 23:30:00", "2020-06-05 10:00:00", "2020-07-15 14:30:00"])
    failure_end  = to_datetime(["2020-04-18 23:59:00", "2020-05-30 06:00:00", "2020-06-07 14:30:00", "2020-07-15 19:00:00"])
```

```
[ ] def in_between(x,start,end):
        start_con = x>=start
        end_con = x<=end
        inbetween_con = start_con and end_con
        if inbetween_con:
            return 1
        else:
            return 0

    failure_indx = []
    import numpy as np
    for i,(start_tim, end_tim) in enumerate(zip(failure_start,failure_end)):
        mask = labeled_data['timestamp'].apply(in_between, start=start_tim, end=end_tim)
        indx = labeled_data.index[mask==True].tolist()
        failure_indx+=indx
    print(f" Found {len(failure_indx)} samples representing failure state")

     Found 29954 samples representing failure state
```

```
[ ] labeled_data['status'].iloc[failure_indx] = 1
```

Balancing the Data

- Further, any imbalances in the data are removed based on the status values. Firstly, we analyse the number of failures to non-failures in the dataset and the perform random sampling to reduce the size of the dataset as well as balance the data for further computations.

```
[ ] #seperating the failures
    pos_data = labeled_data[labeled_data['status']==1]
    neg_data = labeled_data[labeled_data['status']==0]
```

```
print(f"Positive dataset\n {pos_data.info()}\n")
print(f"Negative dataset\n {neg_data.info()}\n")
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29954 entries, 562564 to 1172714
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   timestamp         29954 non-null  datetime64[ns]
 1   TP2               29954 non-null  float64
 2   TP3               29954 non-null  float64
 3   H1                29954 non-null  float64
 4   DV_pressure       29954 non-null  float64
 5   Reservoirs        29954 non-null  float64
 6   Oil_temperature   29954 non-null  float64
 7   Motor_current     29954 non-null  float64
 8   COMP              29954 non-null  float64
 9   DV_eletric        29954 non-null  float64
 10  Towers            29954 non-null  float64
 11  MPG               29954 non-null  float64
 12  LPS               29954 non-null  float64
 13  Pressure_switch   29954 non-null  float64
 14  Oil_level         29954 non-null  float64
 15  Caudal_impulses   29954 non-null  float64
 16  status            29954 non-null  int64
dtypes: datetime64[ns](1), float64(15), int64(1)
memory usage: 4.1 MB
Positive dataset
 None
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1486994 entries, 0 to 1516947
Data columns (total 17 columns):
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   timestamp         1486994 non-null  datetime64[ns]
 1   TP2               1486994 non-null  float64
 2   TP3               1486994 non-null  float64
 3   H1                1486994 non-null  float64
 4   DV_pressure       1486994 non-null  float64
 5   Reservoirs        1486994 non-null  float64
 6   Oil_temperature   1486994 non-null  float64
 7   Motor_current     1486994 non-null  float64
 8   COMP              1486994 non-null  float64
 9   DV_eletric        1486994 non-null  float64
 10  Towers            1486994 non-null  float64
 11  MPG               1486994 non-null  float64
 12  LPS               1486994 non-null  float64
 13  Pressure_switch   1486994 non-null  float64
 14  Oil_level         1486994 non-null  float64
 15  Caudal_impulses   1486994 non-null  float64
 16  status            1486994 non-null  int64
dtypes: datetime64[ns](1), float64(15), int64(1)
memory usage: 204.2 MB
Negative dataset
 None
```

We can observe that we have 29954 cases of failures and 1486994 cases of no failures, hence 50x more negative samples which is highly imbalanced. Hence, we sample out same number of samples from the negative data to balance it.

```
n_positives = int(pos_data['status'].count())
sub_neg_data = neg_data.sample(n_positives, random_state = 42)
print(f"Negative dataset after subsampling {sub_neg_data.info()}")

<class 'pandas.core.frame.DataFrame'>
Int64Index: 29954 entries, 1306134 to 1276653
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   timestamp         29954 non-null  datetime64[ns]
 1   TP2               29954 non-null  float64
 2   TP3               29954 non-null  float64
 3   H1                29954 non-null  float64
 4   DV_pressure       29954 non-null  float64
 5   Reservoirs        29954 non-null  float64
 6   Oil_temperature   29954 non-null  float64
 7   Motor_current     29954 non-null  float64
 8   COMP              29954 non-null  float64
 9   DV_eletric        29954 non-null  float64
 10  Towers            29954 non-null  float64
 11  MPG               29954 non-null  float64
 12  LPS               29954 non-null  float64
 13  Pressure_switch   29954 non-null  float64
 14  Oil_level         29954 non-null  float64
 15  Caudal_impulses   29954 non-null  float64
 16  status            29954 non-null  int64
dtypes: datetime64[ns](1), float64(15), int64(1)
memory usage: 4.1 MB
Negative dataset after subsampling None
```

```
[ ] merged_data = pd.concat([pos_data, sub_neg_data], axis = 0)
```

Detecting Outliers

- We use IQR (Interquartile Range) method to detect and drop outliers from each feature. IQR is a measure of statistical dispersion, which is the spread of the data. It is defined as the difference between the 75th and 25th percentiles of the data and may also be called as midspread, middle 50%, fourth spread, or H-spread.

```python
[ ]  def investigate_outliers(data,c):
         q1 = data[c].quantile(0.25)
         q3 = data[c].quantile(0.75)
         iqr = q3-q1
         ll=q1-1.5*iqr
         ul=q3+1.5*iqr
         num_outliers = data[data[c]<ll][c].count() + data[data[c]>ul][c].count()
         if num_outliers>0:
             print(f"Found {num_outliers} oulier(s) for feature {c}")
         return {'col': c, 'n_outliers': num_outliers, 'll': ll, 'ul': ul, 'q1': q1, 'q3':q3}

     clean_data = merged_data.copy()
     for i in range(5):
         for c in clean_data.columns:
             if c not in ["Unnamed: 0", "timestamp"]:
                 cue = investigate_outliers(clean_data,c)
                 if cue["n_outliers"]>0 and (cue["q1"] != cue["q3"]):
                     print(f"Dropping {cue['n_outliers']} from column {c}")
                     clean_data = clean_data[clean_data[c]>cue["ll"]]
                     clean_data = clean_data[clean_data[c]<cue["ul"]]
                     print(f"{clean_data.shape[0]} samples left\n")
                 elif (cue["q1"]== cue["q3"]):
                     print("Skipping .. data has Q1 equals to Q3")
                     print(f"{clean_data.shape[0]} rows left\n")


     for c in clean_data.columns:
         if c not in ["Unnamed:0", "timestamp", "COMP", 'status']:
             cue = investigate_outliers(clean_data,c)
```

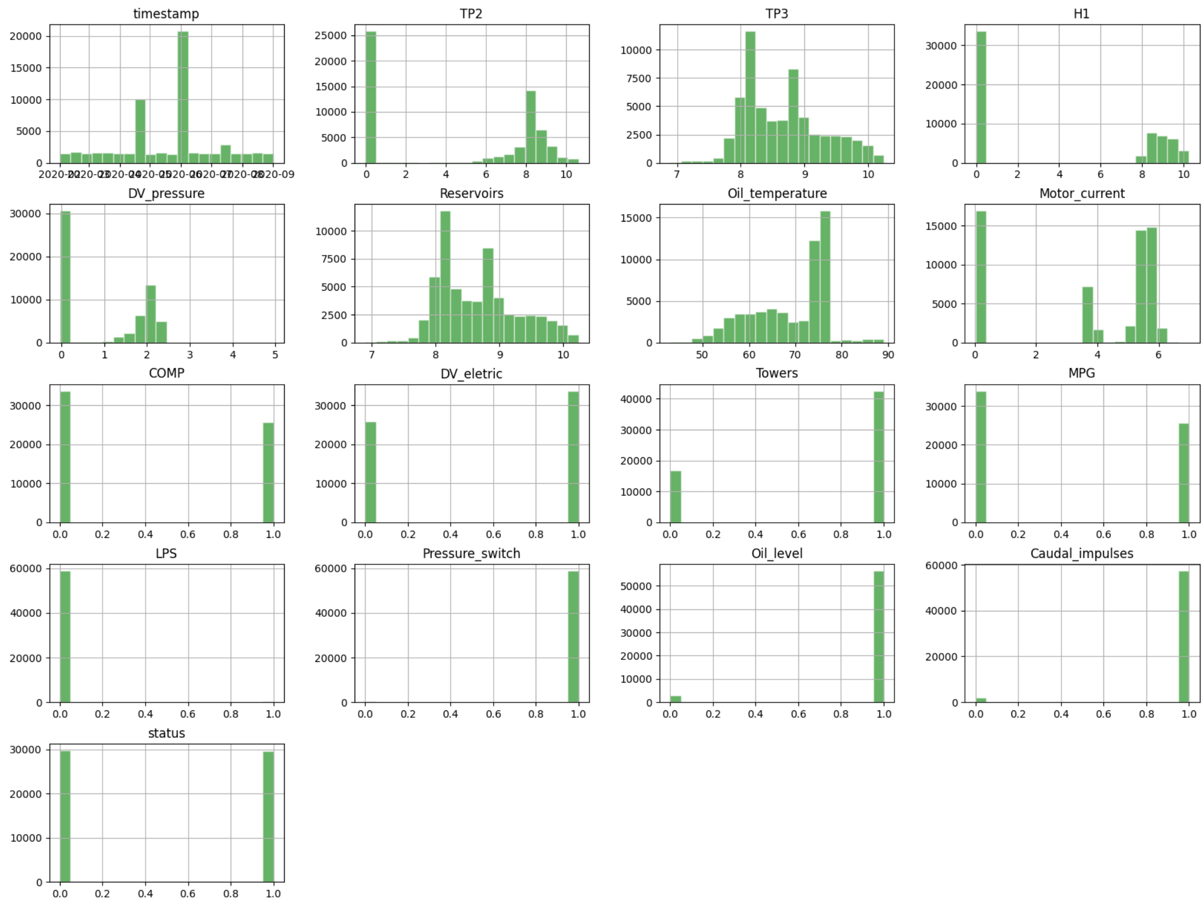
```
Found 424 oulier(s) for feature TP3
Dropping 424 from column TP3
59484 samples left

Found 5 oulier(s) for feature DV_pressure
Dropping 5 from column DV_pressure
59479 samples left

Found 3 oulier(s) for feature Reservoirs
Dropping 3 from column Reservoirs
59476 samples left

Found 29 oulier(s) for feature Oil_temperature
Dropping 29 from column Oil_temperature
59447 samples left

Found 395 oulier(s) for feature LPS
Skipping .. data has Q1 equals to Q3
59447 rows left

Found 402 oulier(s) for feature Pressure_switch
Skipping .. data has Q1 equals to Q3
59447 rows left

Found 2897 oulier(s) for feature Oil_level
Skipping .. data has Q1 equals to Q3
59447 rows left

Found 1948 oulier(s) for feature Caudal_impulses
Skipping .. data has Q1 equals to Q3
59447 rows left
```

```python
[ ]  #Investigate the columns with binary values
     binary_cols = ['LPS', 'Pressure_switch', 'Oil_level', 'Caudal_impulses']
     clean_data[binary_cols] = clean_data[binary_cols].apply(np.round)
```

DATA VISUALIZATION

Histogram

```
clean_data.hist(figsize=(20,15), ec='white',bins=20, color='green', alpha=0.6)
plt.show()
```

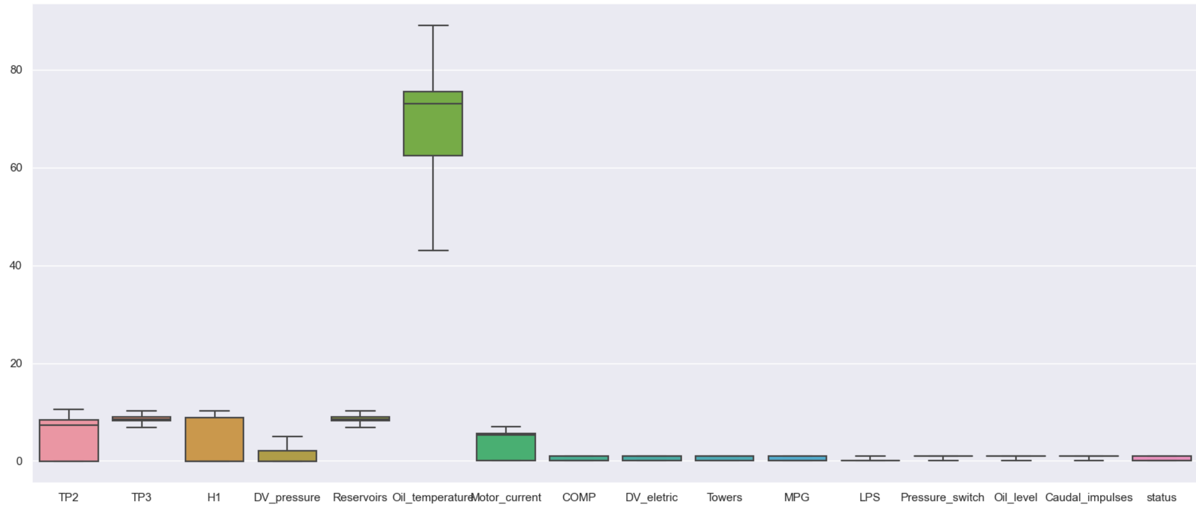


INFERENCE:

- This graph gives the histogram plots for each attribute in our dataset.
- It tells the frequency distribution of each value in each of the attributes.
- The x-values give the range of values of the attribute and the y-values gives the count of each range.
- We confirm that the attributes timestamp, tp2, tp3, HI, DV_pressure, Reservoirs, Oil_temperature and motor current are indeed continuous whereas COMP, DV_electric,Towers, MPG, LPS, Pressure,Switch , Oil level and Caudal_impulses are binary, i.e, 0 and 1.

Box Plot

```
[ ]  sns.set(rc={'figure.figsize':(20,8.27)})
     sns.boxplot(clean_data, autorange = True)
```

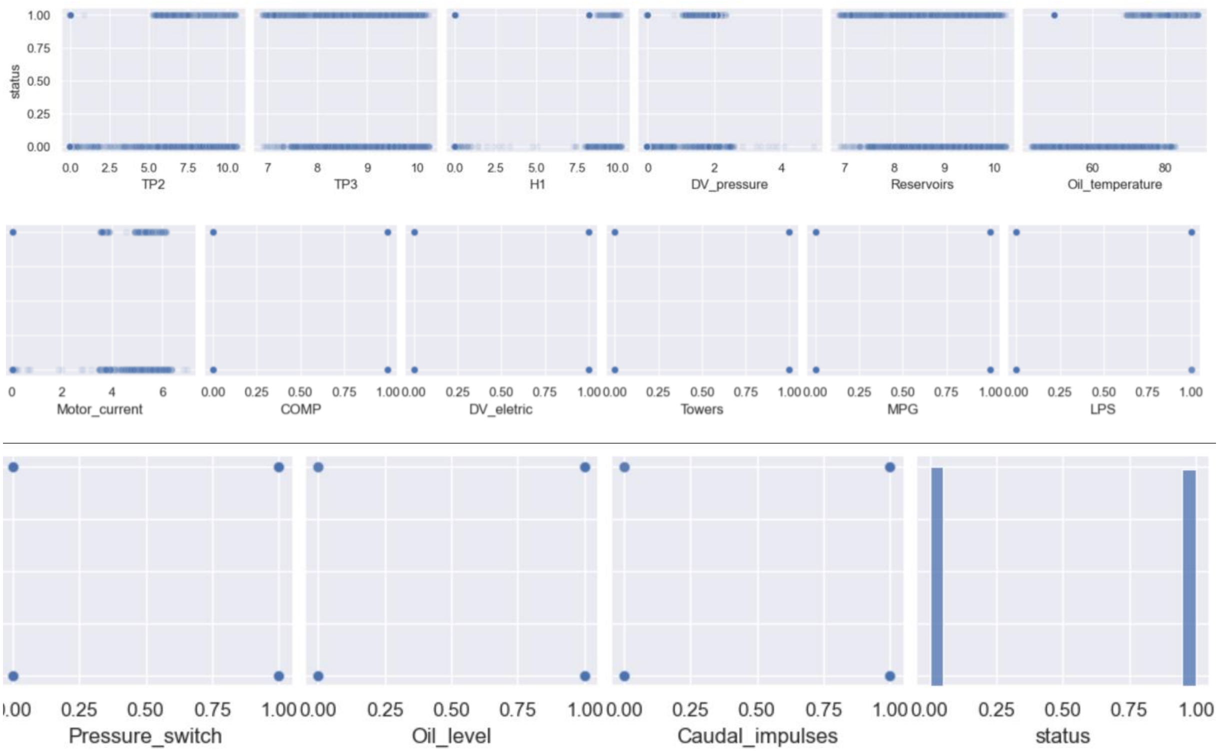


INFERENCE:

- This graph gives the box plots for each attribute in our dataset.
- It tells us about the ranges between which each value lies as well as the interquartile range of each feature.
- It can be observed that the pre-processing has worked well and there are no outliers present in the processed data.

Pair Plot

```
sns.pairplot(clean_data,  y_vars = ['status'] , plot_kws=  {'alpha' : 0.1})
```



INFERENCE:

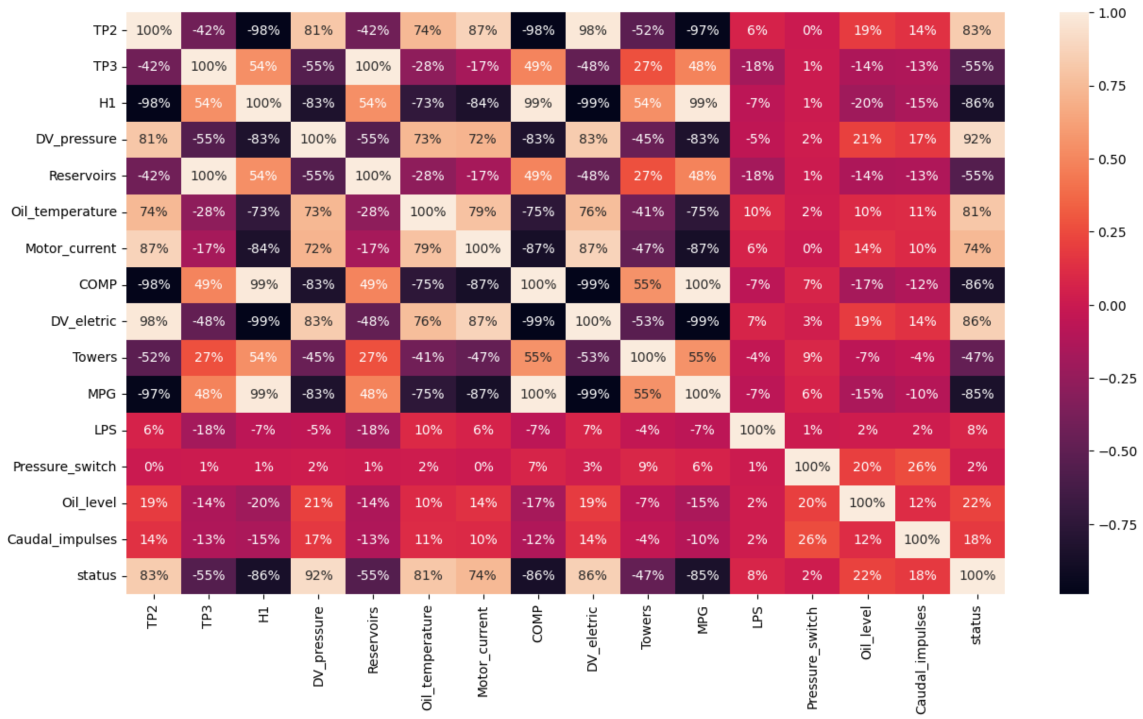
- We plot out the pair plots between ‘status’ and all the other attributes to analyse the trends for further analysis.
- This tells us about any relations between the attributes.

Correlation Matrix

```
clean_data.corr().round(2)
```

	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	status
TP2	1.00	-0.42	-0.98	0.81	-0.42	0.74	0.87	-0.98	0.98	-0.52	-0.97	0.06	0.00	0.19	0.14	0.83
TP3	-0.42	1.00	0.54	-0.55	1.00	-0.28	-0.17	0.49	-0.48	0.27	0.48	-0.18	0.01	-0.14	-0.13	-0.55
H1	-0.98	0.54	1.00	-0.83	0.54	-0.73	-0.84	0.99	-0.99	0.54	0.99	-0.07	0.01	-0.20	-0.15	-0.86
DV_pressure	0.81	-0.55	-0.83	1.00	-0.55	0.73	0.72	-0.83	0.83	-0.45	-0.83	-0.05	0.02	0.21	0.17	0.92
Reservoirs	-0.42	1.00	0.54	-0.55	1.00	-0.28	-0.17	0.49	-0.48	0.27	0.48	-0.18	0.01	-0.14	-0.13	-0.55
Oil_temperature	0.74	-0.28	-0.73	0.73	-0.28	1.00	0.79	-0.75	0.76	-0.41	-0.75	0.10	0.02	0.10	0.11	0.81
Motor_current	0.87	-0.17	-0.84	0.72	-0.17	0.79	1.00	-0.87	0.87	-0.47	-0.87	0.06	0.00	0.14	0.10	0.74
COMP	-0.98	0.49	0.99	-0.83	0.49	-0.75	-0.87	1.00	-0.99	0.55	1.00	-0.07	0.07	-0.17	-0.12	-0.86
DV_eletric	0.98	-0.48	-0.99	0.83	-0.48	0.76	0.87	-0.99	1.00	-0.53	-0.99	0.07	0.03	0.19	0.14	0.86
Towers	-0.52	0.27	0.54	-0.45	0.27	-0.41	-0.47	0.55	-0.53	1.00	0.55	-0.04	0.09	-0.07	-0.04	-0.47
MPG	-0.97	0.48	0.99	-0.83	0.48	-0.75	-0.87	1.00	-0.99	0.55	1.00	-0.07	0.06	-0.15	-0.10	-0.85
LPS	0.06	-0.18	-0.07	-0.05	-0.18	0.10	0.06	-0.07	0.07	-0.04	-0.07	1.00	0.01	0.02	0.02	0.08
Pressure_switch	0.00	0.01	0.01	0.02	0.01	0.02	0.00	0.07	0.03	0.09	0.06	0.01	1.00	0.20	0.26	0.02
Oil_level	0.19	-0.14	-0.20	0.21	-0.14	0.10	0.14	-0.17	0.19	-0.07	-0.15	0.02	0.20	1.00	0.12	0.22
Caudal_impulses	0.14	-0.13	-0.15	0.17	-0.13	0.11	0.10	-0.12	0.14	-0.04	-0.10	0.02	0.26	0.12	1.00	0.18
status	0.83	-0.55	-0.86	0.92	-0.55	0.81	0.74	-0.86	0.86	-0.47	-0.85	0.08	0.02	0.22	0.18	1.00

```
[ ]  plt.figure(figsize=(15, 8))
     sns.heatmap(clean_data.corr().round(2), annot=True, fmt=".0%", linewidths=.0)
     plt.show()
```



INFERENCE:

- A correlation matrix is a statistical tool that shows the strength and direction of relationships between two attributes. The grid values give the value of correlation between the attributes, the values closer to 1 or -1 represent a stronger relation.
- A heatmap is a color-coded representation of the correlation matrix, here a darker-blue shade represents a stronger relation, and a whiter shade represents weak relation.
- Using these two metrics we observe that our target variable "status" has high correlation with TP2, H1, DV_pressure, Oil_temparature, Motor_current, COMP, DV_electric and MPG.

ML CLASSIFICATION ALGORITHMS

Before applying any classification algorithms, we need to perform the following tasks: -

- Split the dataset into **X(input)** and **Y(output)** as 2 separate variables.

```
[ ]  X = data.iloc[:, 2:-1]
     y = data.iloc[:, -1]
```

- Next, we split our **X** and **Y** variables into training data and testing data.
- We use the **train_test_split()** function of **sklearn.model_selection** library to split the dataset. **(80% training and 20% testing)**

```
[ ]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

After getting a cleaned dataset, we can now apply our prediction algorithms, to predict the failure rate and detect anomalies during metro trips.

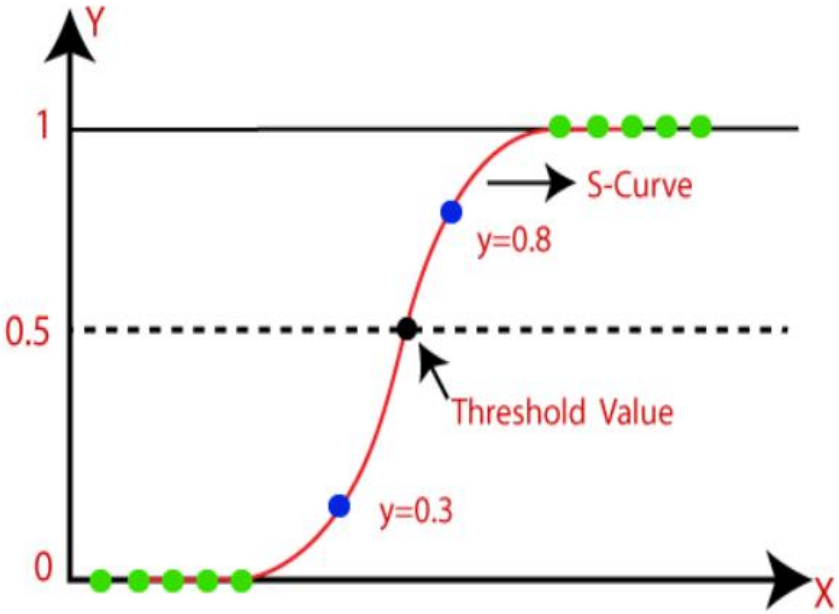
- In our project, instead of applying just one, we use 5 different classification algorithms to predict the results.
    - Logistic Regression
    - KNN Classifier
    - Naïve Bayes Classifier
    - Random Forest Classifier
    - Support Vector Machine
- The motivation to apply all these algorithms was that we wanted to compare their accuracy results to see which algorithm works better on our dataset.

❖ For each case, we've visualized the Confusion Matrix along with it.

❖ We've also displayed the accuracy percentage for each case too.

Logistic Regression

Logistic Regression is used for predicting the categorical dependent variable using a given set of independent variables. It is used for solving the classification problems. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). The curve from the logistic function indicates the likelihood of something. It basically follows the linear regression model, but the continuous output value is passed through a function called as “Sigmoid Function”, which defines the probability to either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.



Logistic Regression is widely used in various fields, including medicine, finance, and social sciences, for tasks such as predicting disease occurrence, credit risk, and customer churn. It's a fundamental algorithm in the field of machine learning and is relatively simple yet effective for binary classification problems.

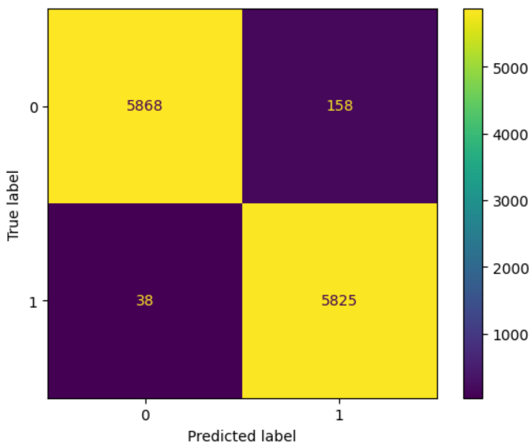
Implementation

```python
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

```python
y_pred = lr.predict(X_test)
```

```python
from sklearn.metrics import accuracy_score
print("Accuracy Score: {:,.3f}".format(accuracy_score(y_test, y_pred) * 100))
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
from sklearn.metrics import roc_curve, auc, roc_auc_score

y_prob = lr.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc_score = roc_auc_score(y_test, y_prob)

# Plot ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print(f'AUC Score: {auc_score:.4f}')
```

Confusion Matrix



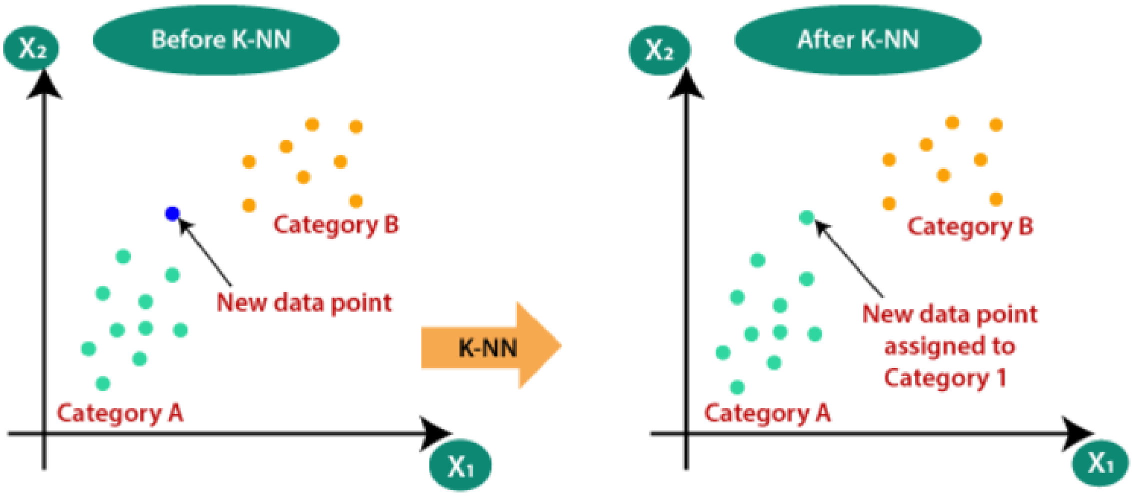
Classification Report

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      6026
           1       0.97      0.99      0.98      5863

    accuracy                           0.98     11889
   macro avg       0.98      0.98      0.98     11889
weighted avg       0.98      0.98      0.98     11889
```

Accuracy Score – 98.351%

KNN Classifier

K-Nearest Neighbour, based on Supervised Learning algorithm assumes the similarity between the new case and available cases and put the new case into the category that is most similar to the available categories. It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. It is used mainly for the Classification problems. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.



KNN is used in image recognition, recommendation systems, healthcare for medical diagnosis, finance in credit scoring, anomaly detection in cybersecurity, environmental monitoring, text mining, and robotics for decision-making.

Implementation

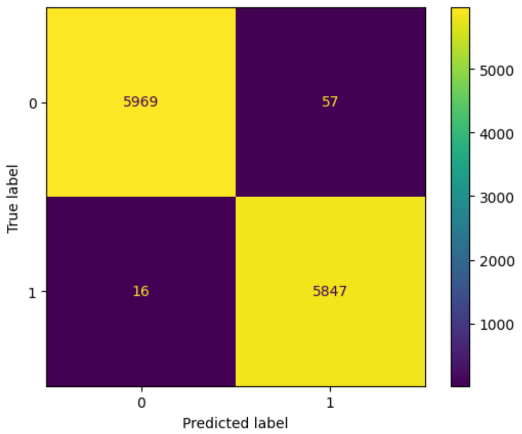
The default value of 'k' in scikit learns implementation of KNN is 5, while this may be a good starting point, we may need to adjust it after analysing different values of accuracy and error rates for different values of K.

Nonetheless, we will initially use the default implementation of KNN without specifying a k value.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
```

```
[ ]  y_pred = knn.predict(X_test)
```

Confusion Matrix (Default KNN)



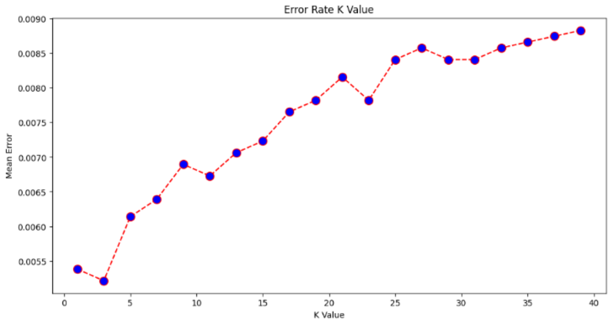
Classification Report(Default KNN)

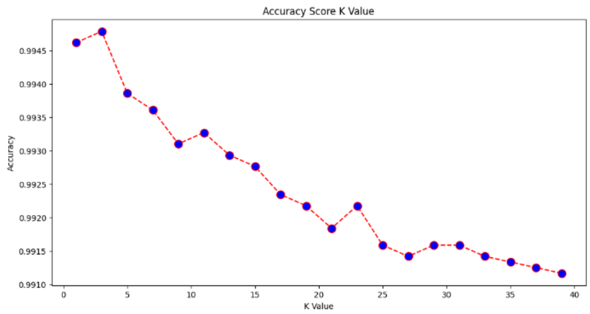
```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      6026
           1       0.99      1.00      0.99      5863

    accuracy                           0.99     11889
   macro avg       0.99      0.99      0.99     11889
weighted avg       0.99      0.99      0.99     11889
```

Accuracy Score – 99.386%

We can see the default KNN, i.e., k=5, gives an accuracy score of 99.386%. But we can do better by tweaking this value, for that we need to analyse the accuracy and error rates for different k values. We plot accuracy and error for all the odd k values between 1 and 40 :

```
[ ]  error = []
     accuracy = []
     # Calculating error and accuracy for K values between 1 and 40
     for i in range(1,40,2):
         knn = KNeighborsClassifier(n_neighbors=i)
         knn.fit(X_train, y_train)
         pred_i = knn.predict(X_test)
         error.append(np.mean(pred_i != y_test))
         accuracy.append(accuracy_score(y_test, pred_i))
```





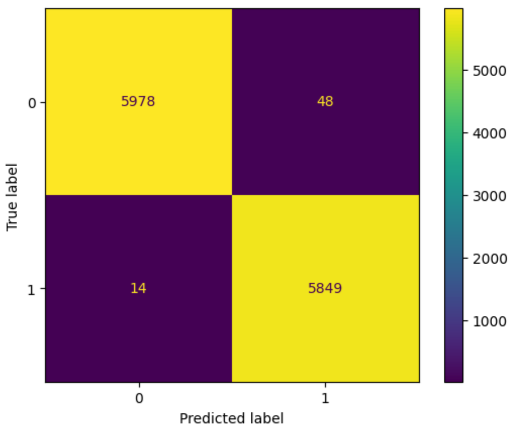
We observe that k=3 gives the least error and the maximum accuracy among all the k values.

Hence we now use the KNeighbourClassifier with a modifier.

```
[ ] y_pred1 = knn.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy Score: {:,.3f}".format(accuracy_score(y_test, y_pred1) * 100))
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred1)
plt.show()
print(classification_report(y_test, y_pred1))

from sklearn.metrics import roc_curve, auc, roc_auc_score
import matplotlib.pyplot as plt

# Assuming X_test is your test set features and y_test is the corresponding true labels
y_prob_knn = knn.predict_proba(X_test)[:, 1]  # Note: KNN does not have predict_proba by default

# ROC Curve and AUC Score
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_prob_knn)
auc_score_knn = roc_auc_score(y_test, y_prob_knn)

# Plot ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr_knn, tpr_knn, label=f'AUC = {auc_score_knn:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.title('ROC Curve for KNN Model')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print(f'AUC Score for KNN Model: {auc_score_knn:.4f}')
```

Confusion Matrix (K=3)



Classification Report (K=3)

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      6026
           1       0.99      1.00      0.99      5863

    accuracy                           0.99     11889
   macro avg       0.99      0.99      0.99     11889
weighted avg       0.99      0.99      0.99     11889
```

Accuracy Score – 99.479%

Naïve Bayes Classifier

Naïve Bayes classifier is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. It is a probabilistic classifier, which means it predicts based on the probability of an object.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, which can be described as:

- **Naïve**: Assumes that the occurrence of a certain feature is independent of the occurrence of other features.
- **Bayes**: Depends on the principle of Bayes' Theorem

Bayes theorem, which is used to determine the probability of a hypothesis with prior knowledge in Naïve Bayes' Classifier depends on the conditional probability.

The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

where,

- **P(A|B)** is **Posterior probability**: Probability of hypothesis A on the observed event B.
- **P(B|A)** is **Likelihood probability**: Probability of the evidence given that the probability of a hypothesis is true.
- **P(A)** is **Prior Probability:** Probability of hypothesis before observing the evidence**.**
- **P(B)** is **Marginal Probability:** Probability of Evidence**.**

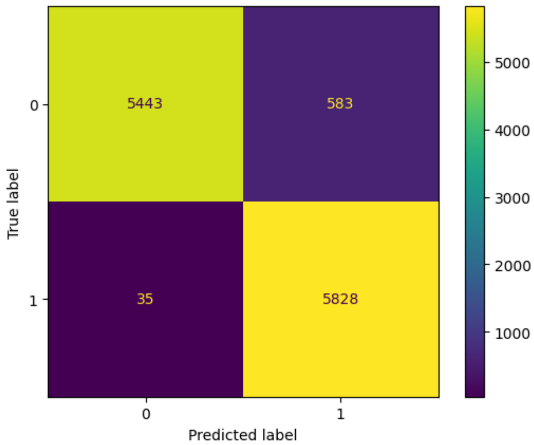
Naïve Bayes Classifier is extensively used in spam filtration, sentimental analysis, and classifying articles.

Implementation

```python
gaussian_classifier = GaussianNB()

# Train the models
gaussian_classifier.fit(X_train, y_train)

# Predictions
y_pred_gaussian = gaussian_classifier.predict(X_test)
```

```python
from sklearn.metrics import accuracy_score
print("Accuracy Score: {:,.3f}".format(accuracy_score(y_test, y_pred_gaussian) * 100))
print(classification_report(y_test, y_pred_gaussian))
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred_gaussian)
plt.show()

from sklearn.metrics import roc_curve, auc, roc_auc_score
import matplotlib.pyplot as plt

# Assuming X_test is your test set features and y_test is the corresponding true labels
y_prob_NB = gaussian_classifier.predict_proba(X_test)[:, 1]

# ROC Curve and AUC Score
fpr_NB, tpr_NB, thresholds_NB = roc_curve(y_test, y_prob_NB)
auc_score_NB = roc_auc_score(y_test, y_prob_NB)

# Plot ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr_NB, tpr_NB, label=f'AUC = {auc_score_NB:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.title('ROC Curve for NB Model')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print(f'AUC Score for NB Model: {auc_score_NB:.4f}')
```

Confusion Matrix



Classification Report

```
              precision    recall  f1-score   support

           0       0.99      0.90      0.95      6026
           1       0.91      0.99      0.95      5863

    accuracy                           0.95     11889
   macro avg       0.95      0.95      0.95     11889
weighted avg       0.95      0.95      0.95     11889
```

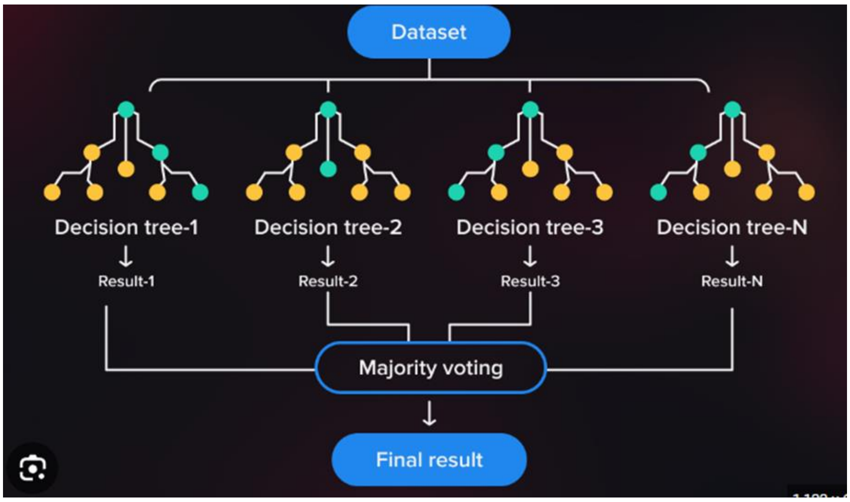
Accuracy Score – 94.802%

Random Forest Classifier

Random Forest a supervised learning algorithm, can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

Random Forest is a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

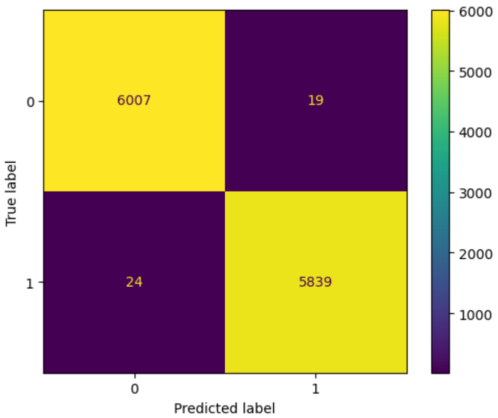


Random Forest Algorithm is used in banking for loan risk identification, medicine for disease trends, land use analysis for categorizing areas, and marketing for identifying the trends. It is capable of handling large datasets with high dimensionality and enhances the accuracy of the model and prevents the overfitting issue.

Implementation

```python
random_forest_classifier = RandomForestClassifier(random_state=42)

random_forest_classifier.fit(X_train, y_train)

y_pred_rf = random_forest_classifier.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)*100
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
classification_rep_rf = classification_report(y_test, y_pred_rf)


print(f'Random Forest Accuracy: {accuracy_rf:.4f}')
print('Random Forest Classification Report:')
print(classification_report(y_test, y_pred_rf))
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred_rf)
plt.show()
from sklearn.metrics import roc_curve, auc, roc_auc_score
import matplotlib.pyplot as plt


y_prob_rf = random_forest_classifier.predict_proba(X_test)[:, 1]


fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_prob_rf)
auc_score_rf = roc_auc_score(y_test, y_prob_rf)


plt.figure(figsize=(8, 8))
plt.plot(fpr_rf, tpr_rf, label=f'AUC = {auc_score_rf:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.title('ROC Curve for Random Forest Model')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print(f'AUC Score for Random forest Model: {auc_score_rf:.4f}')
```

Confusion Matrix



Classification Report

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6026
           1       1.00      1.00      1.00      5863

    accuracy                           1.00     11889
   macro avg       1.00      1.00      1.00     11889
weighted avg       1.00      1.00      1.00     11889
```

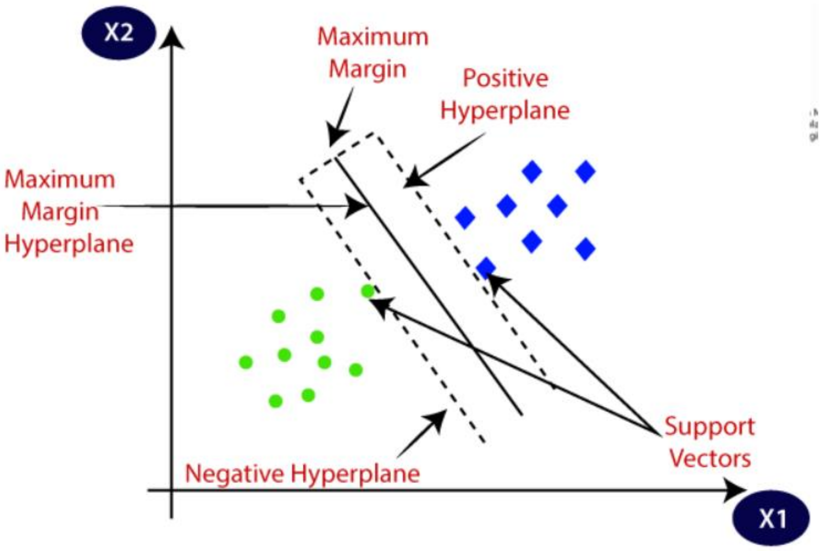
Accuracy Score – 99.6383%

Support Vector Machine (SVM)

Support Vector Machine or SVM, a Supervised Learning algorithm, is used for Classification as well as Regression problem. Primarily, it is used for Classification problem in Machine Learning.

SVM algorithm creates the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

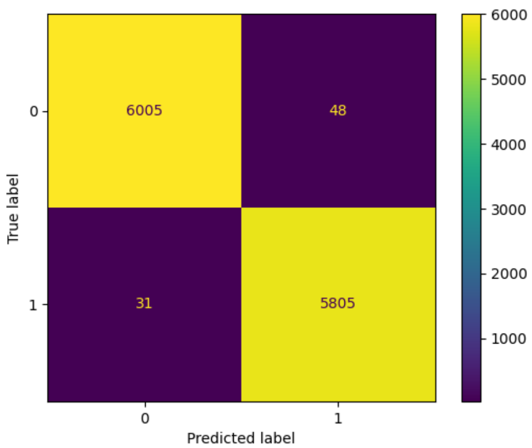


Support Vector Machine (SVM) algorithm is used for Face detection, image classification, text categorization, etc.

Implementation

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train1 = scaler.fit_transform(X_train)
X_test1 = scaler.transform(X_test)

# Create an SVM classifier with probability estimates enabled
svm_classifier = SVC(probability=True, random_state=42)

# Training
svm_classifier.fit(X_train1, y_train)

# Make predictions on the test set
y_pred_svm = svm_classifier.predict(X_test1)

# Evaluate the performance
accuracy_svm = accuracy_score(y_test, y_pred_svm)*100
print(f'SVM Accuracy: {accuracy_svm:.4f}')

# Confusion Matrix
disp = ConfusionMatrixDisplay.from_estimator(svm_classifier, X_test1, y_test)
plt.show()

# ROC-AUC score
y_prob_svm = svm_classifier.predict_proba(X_test1)[:, 1]
fpr_svm, tpr_svm, thresholds_svm = roc_curve(y_test, y_prob_svm)
auc_score_svm = roc_auc_score(y_test, y_prob_svm)

plt.figure(figsize=(8, 8))
plt.plot(fpr_svm, tpr_svm, label=f'AUC = {auc_score_svm:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.title('ROC Curve for SVM Model')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

Confusion Matrix



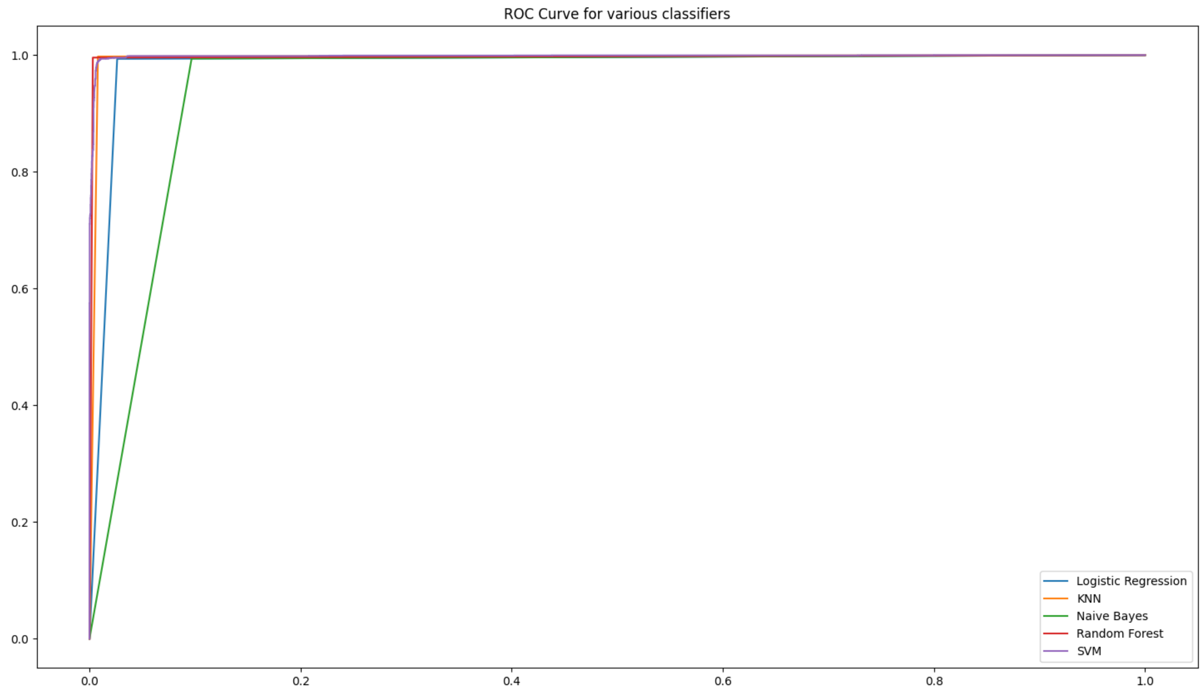
Classification Report

	precision	recall	f1-score	support
0	0.99	0.99	0.99	6053
1	0.99	0.99	0.99	5836
accuracy			0.99	11889
macro avg	0.99	0.99	0.99	11889
weighted avg	0.99	0.99	0.99	11889

Accuracy Score – 99.3355%

CONCLUSION

To summarize our observations from the implementations of the algorithms, we can infer the following: -



Algorithm	Accuracy
Logistic Regression	98.351%
KNN (K=3)	99.479%
Naïve Bayes	94.802%
Random Forest	99.6383%
Support Vector Machine	99.3355%

From the results, we can observe that **Random Forest Classifier** provides the highest accuracy of **99.6383%** for the provided dataset.

References

1. Davari,Narjes, Veloso,Bruno, Ribeiro ,Rita, and Gama,Joao. (2023). MetroPT-3 Dataset. UCI Machine Learning Repository. https://doi.org/10.24432/C5VW3R.
2. N. Davari, B. Veloso, R. P. Ribeiro, P. M. Pereira and J. Gama, "Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry," 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA), Porto, Portugal, 2021, pp. 1-10, doi: 10.1109/DSAA53316.2021.9564181.
3. Official documentations of seaborn, matplotlib, sckit-learn, pandas, and numpy.
4. Class presentation of IDS