CREDIT CARD FRAUD DETECTION AND ENERGY EFFICIENCY MODEL USING MACHINE LEARNING ALGORITHMS

Raja Ritika Reddy Buttreddy

I. Introduction

The aim of this project is to compare different machine learning algorithms between two data sets, one being a classification problem and the other being regression problem. The data sets given were credit card fraud detection and Energy efficiency model.Models in this project achieve a reasonable level of accuracy. This project will also look into whether the metric used accurately captures the performance of the classifier that was trained on the studied dataset. This step ensures that the algorithms do not under or over fit the data. To improve the machine learning algorithms, pre-processing techniques were also used.

II. CREDIT CARD FRAUD DETECTION

The growing number of customers and businesses who use credit cards to make purchases has resulted in a huge increase in fraud instances. A credit card is a card that is assigned to a customer (cardholder) and allows them to purchase goods and services within their credit limit or withdraw cash in advance. Credit cards give the cardholder a time advantage, allowing them to repay their debts later in a specified time frame by carrying it over to the next billing cycle. Credit card fraud are easy targets. Without any risks, a significant amount can be withdrawn in a short period of time without the owner's knowledge. Fraudsters always try to make every fraudulent transaction appear legitimate, making fraud detection a difficult task.
As technology advances, banks are transitioning to EMV cards, which are smart cards that store their data on integrated circuits rather than magnetic stripes. This has made some on-card payments safer, but it has also increased the rate of card-not-present fraud. Even so, there is a risk that thieves will misuse the credit cards. There are numerous machine learning techniques available to address this issue **?**.

A. Data Set

Credit Card Fraud Detection is the first data set that this project works with. This is a two-class classification problem with two options: fraud and not-fraud. The dataset was compiled from credit card transactions made by European cardholders in September 2013. The dataset is imperfect and unbalanced, with over 284,807 genuine transactions versus only 492 (0.172 percent) fraudulent transactions.
This dataset contains 28 features derived from Principal Component Analysis (PCA). 'Time' and 'Amount' are the only features that have not been transformed by PCA. Fraud is indicated by a 1 for the 'Class' feature and a 0 otherwise. Following data pre-processing, three classification algorithms are used to train models for this dataset.

B. Data Assessment

I originally started by looking over the data set. I searched for possible duplicate and null values before inspecting the distribution of each dataset column. The data was then scaled using sci-kit learn's standard scaler. As previously stated, the data set is highly skewed, with only 492 fraudulent transactions and over 2,84,807 non-fraudulent transactions. It is not ideal to train any machine learning model directly on the dataset.
A new dataset for training was created using a balanced sampling technique. This sampling method divides the data into fraudulent and non-fraudulent transactions, which are stored as valid and invalid variables. The incorrect variables include data from the dataset, which randomly selects 492 non-fraudulent transactions.The variables contain the non-fraudulent dataset.
The dataset was divided into three sections: training, validation, and testing. A training dataset is a collection of data that is used to train the model. A validation dataset is a subset of the original dataset that is used to assess model competence while adjusting the model's hyperparameters. When a quality from the validation dataset is used in the model configuration, the assessment becomes increasingly skewed. The test dataset is a subset of data used to provide an objective evaluation of the final model fit on the training dataset.

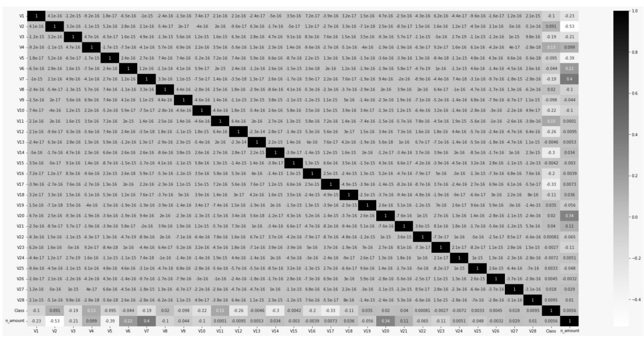


Fig. 1. Corelation Matrix

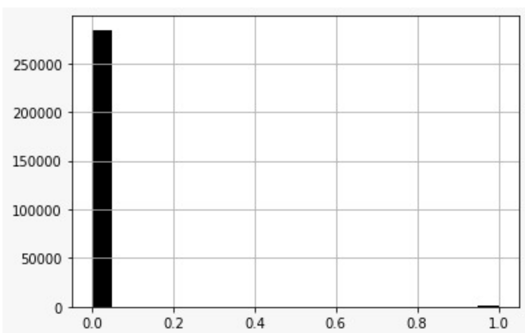


Fig. 2. "Class" label histogram

C. Algorithm-1

Logistic regressions were one of the three algorithms I used. In regression analysis, logistic regression is used to estimate the parameters of a logistic model. Logistic Regression is a machine learning technique that uses probability to perform predictive analysis.

A Logistic Regression model works similarly to a Linear Regression model. However, Logistic Regression employs a more sophisticated cost function, known as the 'Sigmoid function' or sometimes as the 'logistic function,' rather than a linear function. The logistic regression hypothesis limits the cost function to a range of 0 to 1. As a result, linear functions can't describe it because it could have a value greater than 1 or less than 0, which isn't possible according to the logistic regression hypothesis.

The Logistic Regression model produced significantly better results. Given the simplicity of Logistic Regression, this model outperformed my expectations in terms of classification report (accuracy, recall, and f1score).

accuracy			0.95	170589
macro avg	0.95	0.95	0.95	170589
weighted avg	0.95	0.95	0.95	170589

Fig. 3. Analysis of logistic regression maetrics

D. Algorithm-2

Random forests, also known as random decision forests, are a group learning approach for classification, regression, and other problems that generates a large number of decision trees during training. One of the most important features of the Random Forest Algorithm is its ability to handle both classification and regression datasets. The random forest algorithm outperforms the other two algorithms.

```
Accuracy: 0.99961
Precision: 0.94017
Recall: 0.80882
F1-score: 0.86957
```

Fig. 4. Analysis of Random Forest

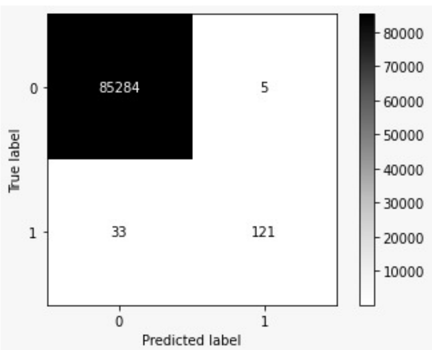


Fig. 5. Matrix for Random forest

E. Algorithm-3

A decision tree is a tool for making decisions that incorporates a tree-like model of decisions and potential outcomes, such as chance event outcomes, resource costs, and utility. It is one way of displaying an algorithm that is made up entirely of conditional control statements. A decision tree is a flowchart-like structure with nodes representing "tests" on various attributes.

Accuracy, precision, recall, F1-score, and confusion matrix were the metrics used to evaluate this model.

```
Accuracy: 0.99920
Precision: 0.72667
Recall: 0.80147
F1-score: 0.76224
```

Fig. 6. Analysis of Decision tree

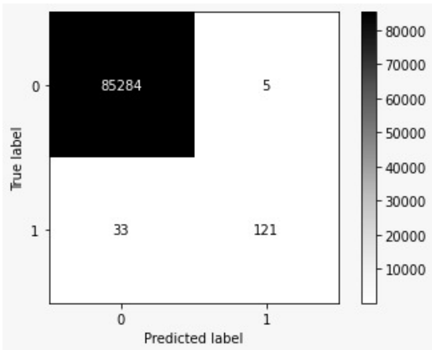


Fig. 7. Matrix for Decision tree

F. Solving class imbalance

One method for dealing with unbalanced datasets is to oversample the minority class. The simplest method involves copying instances from the minority class, even if these examples provide no new information to the model. Instead, new instances can be created by combining existing ones. The Synthetic Minority Oversampling Technique (SMOTE) is a data augmentation technique for the minority population. To put it simply: When compared to the number of non-fraud rows, the number of fraud rows is very small. As a result of this imbalance, a flawed model emerges. To compensate for the number difference, we can generate new samples artificially. We can 'create' the most recent data by using existing data. This is known as oversampling.

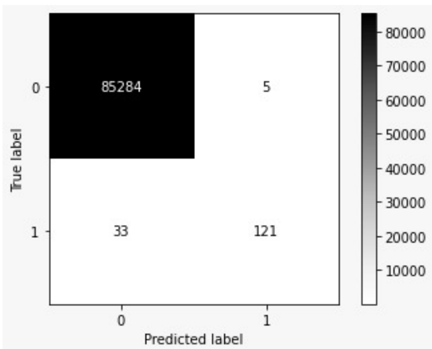


Fig. 8. Matrix for Random Forest Algorithm after SMOT

G. Evaluation

Confusion matrices are matrices that represent the sum of expected and actual values as counts. The output "TN" represents the number of correctly identified negative cases and stands for True Negative. Similarly, "TP" stands for True Positive, and it denotes the number of positively identified occurrences. The abbreviation "FP" stands for False Positive value, which is the number of genuine negative cases classified as positive; "FN" stands for False Negative value, which is the number of genuine positive cases classified as negative. One of the most commonly used criteria for categorization is accuracy. The following formula is used to calculate a model's accuracy (through a confusion matrix).

$$\text{Accuracy} = \frac{\text{TN+TP}}{\text{TN+FP+FN+TP}}$$

From the given dataset, three machine learning algorithms were used to detect credit card fraud. To evaluate the algorithms, 70% of the dataset was used for training and 30% for testing and validation. The logistic regression, decision tree, and random forest classifiers have accuracy scores of 98%, 98%, and 94%, respectively. The comparison results show that the Random Forest technique outperforms the Logistic Regression and decision tree techniques. While this appears to be excessive in comparison to the other models' results, it is important to remember that under-sampling was used. One possible explanation for this result is that the number of fraudulent and non-fraudulent occurrences in the dataset must be balanced.

III. ENERGY EFFICIENCY DATASET

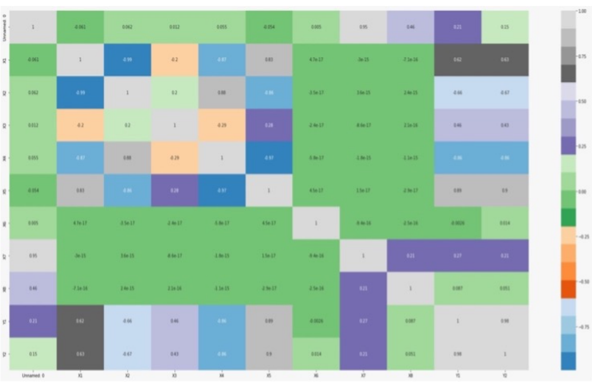


Fig. 9. Correlation between each variable from the dataset

A. Data set

The Energy Efficiency data set was used as the second data set. The dataset includes eight attributes or features, denoted by X1...X8, as well as two responses or outcomes, denoted by y1 and y2. Our goal is to forecast the energy. When I first started working on a new data collection, the first thing I did was investigate it. I made a series of graphs to help people see and understand the data. To begin, I visualized the data set. The first is used to investigate the relationship between each characteristic in the data set and to generate a correlation coefficient matrix heat map, as shown in the image below.

B. Polynomial Regression

Polynomial regression, like many other machine learning concepts, is based on statistics. Statistical analysis is performed when there is a non-linear relationship between the value of xx and the related conditional mean of yy.

C. Support Vector Regression

Support Vector Regression is a supervised learning technique used to predict discrete values. Support Vector Regression and SVMs are both based on the same logic. The primary idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane with the most points.

D. Linear Regression

Linear regression is a linear technique for modeling the connection between a scalar response and one or more explanatory factors. Linear regression is a statistical technique for modeling the connection between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). When there is only one explanatory variable, simple linear regression is used. Linear regression is a well-known and widely used regression technique. It's one of the most fundamental regression algorithms. One of its primary advantages is the ease with which the results can be comprehended. When performing basic linear regression, you usually begin with a predefined set of input-output (x-y) pairs. You have made the following observations.When there are two or more independent variables, multiple or multivariate linear regression is used.

IV. CONCLUSION

On the whole, the models produced fairly accurate results. As we predicted, the models' accuracy improved as they progressed. This is demonstrated by the fact that the Polynomial Regression has the highest value of all. Continue to examine the data and look for items that do not have a strong association in order to improve these models. With more time, one can discover a method to reduce the dimensionality of the data by removing features that have little impact on the regression problem. With a score of 0.998, we discovered that the polynomial regression model has a superior effect, implying that our data set is more consistent with the polynomial regression model.This project also goes into detail about how machine learning can be used to improve fraud detection results, including the algorithm, pseudocode, implementation description, and experimentation results

V. FUTURE WORKS

Even as we did not achieve our goal of 100 percent fraud detection accuracy, we did design a system that, given enough time and data, can come very close. There is always room for improvement in any endeavor of this magnitude. Because of the nature of this project, many algorithms may be linked as modules, and their results may be pooled to improve the accuracy of the final output. Other algorithms can be used to improve these models even further. The output of these algorithms, however, must be in the same format as the others. Once that condition is met, the modules can be easily added, as shown in the code. As a result, the project is extremely adaptable and versatile.Additional development opportunities are included in the dataset. As previously stated, the precision of the algorithms grows in proportion to the size of the dataset. As a result, more data will almost certainly improve the model's ability to detect fraud while decreasing the number of false positives. This, however, necessitates explicit approval from the banks.

```
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, cross_validate
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import roc_curve, roc_auc_score
import itertools
from collections import Counter
from sklearn.manifold import TSNE
from sklearn import preprocessing
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

```
df = pd.read_csv ("creditcard 2.csv")
```

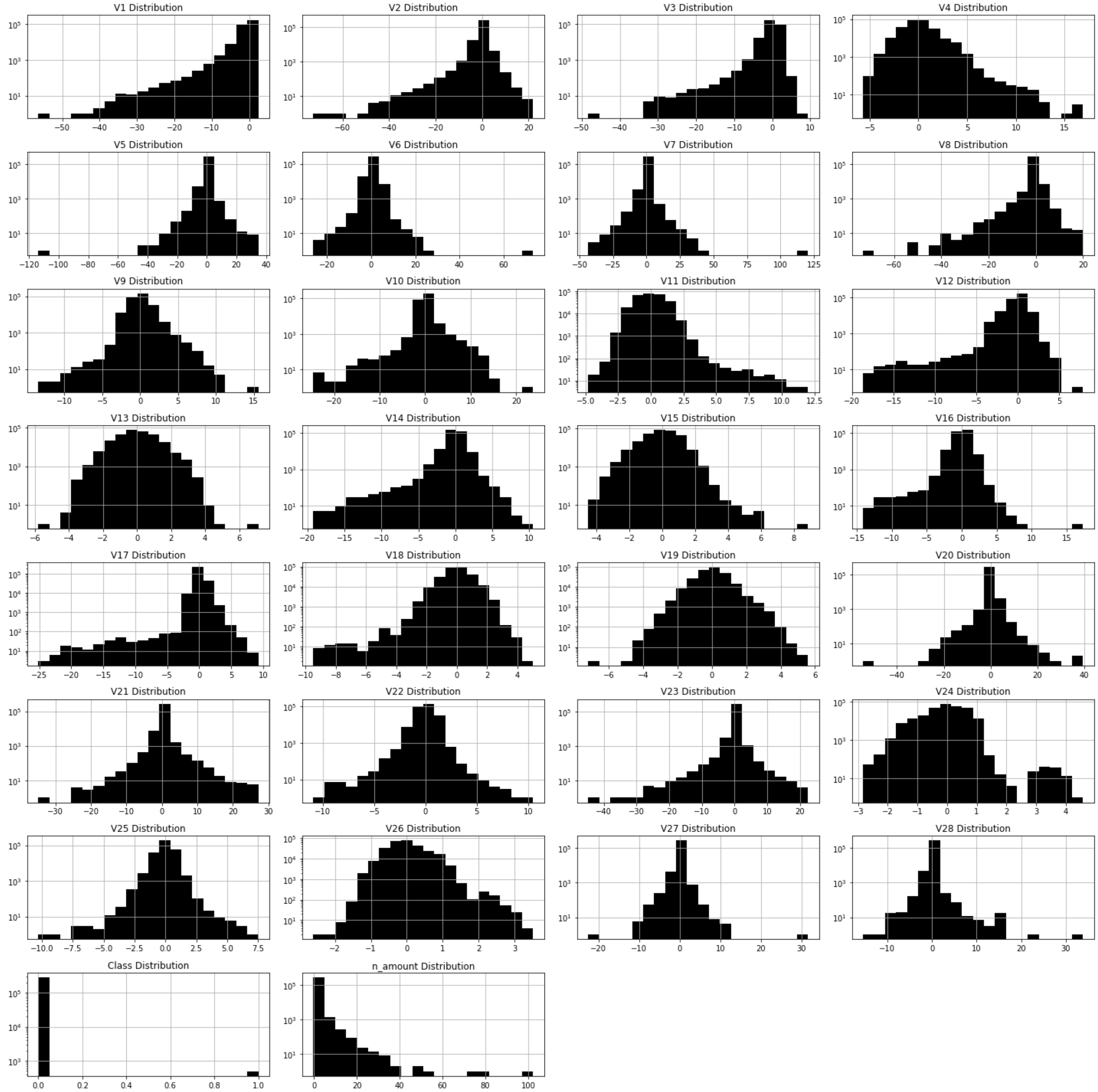
```
df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
```

```
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

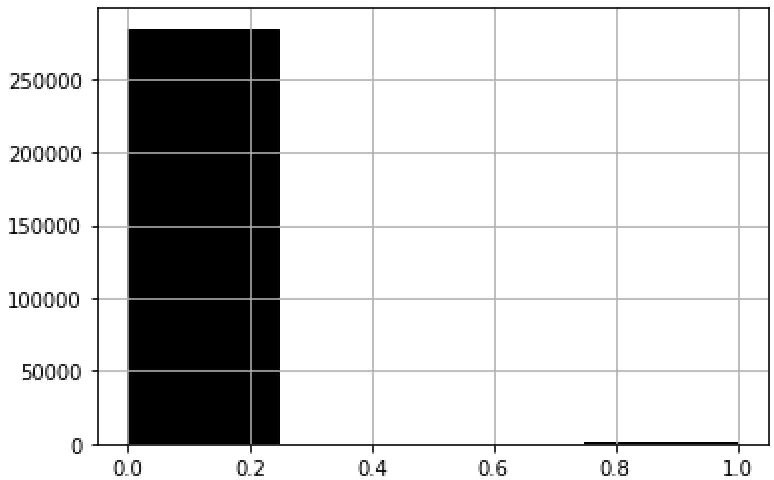
```
#round(100 * (df.isnull().sum()/len(card)),2).sort_values(ascending=False)
#round(100 * (df.isnull().sum(axis=1)/len(card)),2).sort_values(ascending=False)
#no null values in the dataset
```

```
fig=plt.figure(figsize=(20,20))
for i, feature in enumerate(df.columns):
    ax=fig.add_subplot(8,4,i+1)
    df[feature].hist(bins=20,ax=ax,facecolor='black')
    ax.set_title(feature+" Distribution",color='black')
    ax.set_yscale('log')
fig.tight_layout()
plt.show()
```

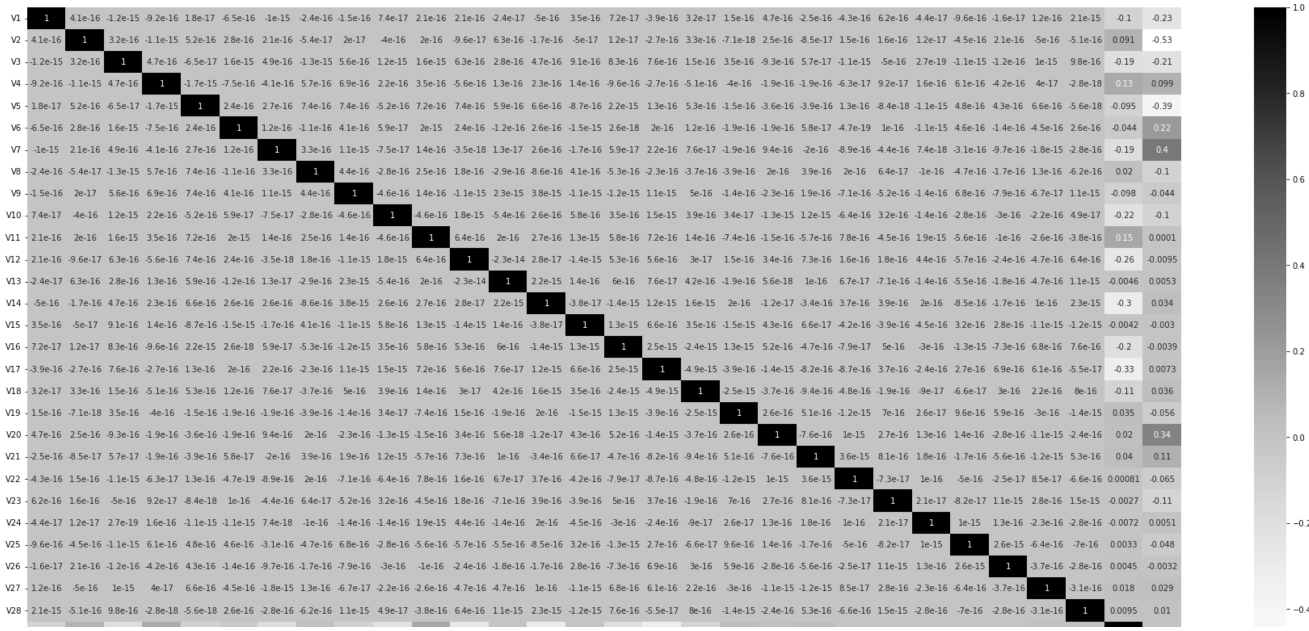

V1 Distribution
V2 Distribution
V3 Distribution
V4 Distribution
V5 Distribution
V6 Distribution
V7 Distribution
V8 Distribution
V9 Distribution
V10 Distribution
V11 Distribution
V12 Distribution
V13 Distribution
V14 Distribution
V15 Distribution
V16 Distribution
V17 Distribution
V18 Distribution
V19 Distribution
V20 Distribution
V21 Distribution
V22 Distribution
V23 Distribution
V24 Distribution
V25 Distribution
V26 Distribution
V27 Distribution
V28 Distribution
Class Distribution
n_amount Distribution

```
plt.grid(False)
df["Class"].hist(bins=4,facecolor='black')
print(df["Class"].value_counts())
```

```
0    284315
1       492
Name: Class, dtype: int64
```




```
plt.figure(figsize = (32,15))
sns.heatmap(df.corr(), annot = True, cmap="Greys")
plt.show()
```




```
#Scaling
scaler = StandardScaler()
df["n_amount"] = scaler.fit_transform(df["Amount"].values.reshape(-1, 1))
#dropping amount and time columns, not needed
df.drop(["Amount", "Time"], inplace= True, axis= 1)

y = df["Class"]
X = df.drop(["Class"], axis= 1)
```

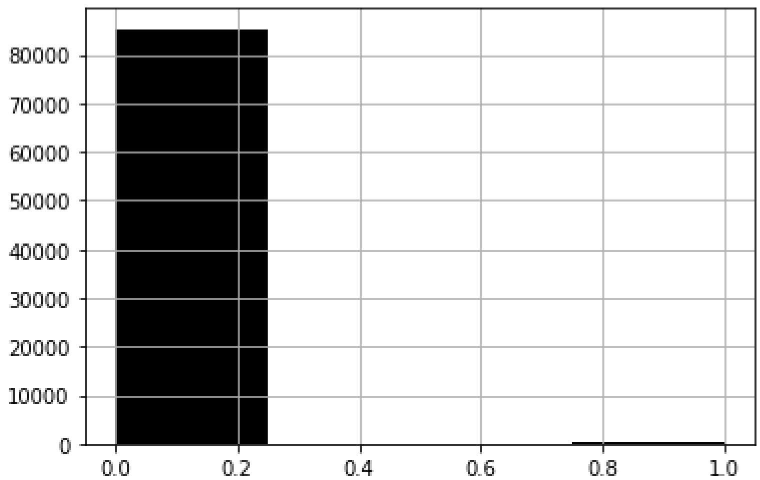
```
(X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size= 0.3, random_st
```

```
# checking the distribution of the split
print(y_train.value_counts())
plt.grid(False)
y_train.hist(bins=4,facecolor='black')
plt.show()
```

```
0    199026
1       338
Name: Class, dtype: int64
```

```
print(y_test.value_counts())
y_test.hist(bins=4,facecolor='black')
plt.show()
```

```
0    85289
1      154
Name: Class, dtype: int64
```




```
#model object
#running logistic regression
model = LogisticRegression()
model.fit(X_train, y_train)

#predicting and metrics
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred, target_names=["Not-Fraud", "Fraud"]))
print(f'Accuracy: {str(accuracy_score(y_test, y_pred) * 100)}%')
```

```
              precision    recall  f1-score   support

   Not-Fraud       1.00      1.00      1.00     85289
       Fraud       0.88      0.58      0.70       154

    accuracy                           1.00     85443
   macro avg       0.94      0.79      0.85     85443
weighted avg       1.00      1.00      1.00     85443

Accuracy: 99.9098814414288%
```

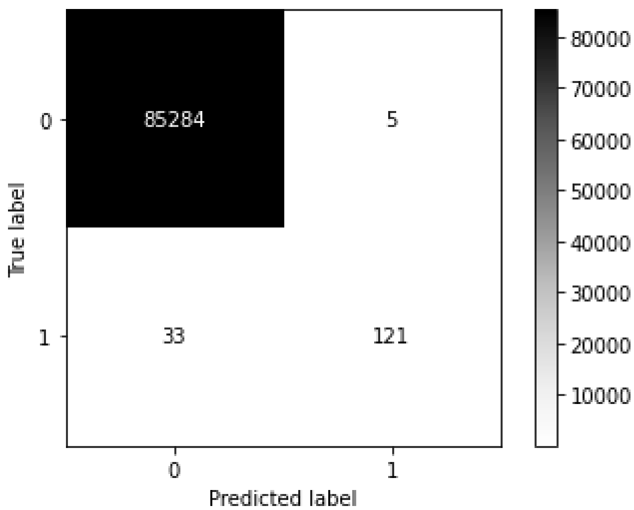
```
# Decision Tree Classifier
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)

y_pred = decision_tree.predict(X_test)
```

```
print(classification_report(y_test, y_pred, target_names=["Not-Fraud", "Fraud"]))
print(f'Accuracy: {str(accuracy_score(y_test, y_pred) * 100)}%')
```

```
              precision    recall  f1-score   support

   Not-Fraud       1.00      1.00      1.00     85289
       Fraud       0.80      0.76      0.78       154

    accuracy                           1.00     85443
   macro avg       0.90      0.88      0.89     85443
weighted avg       1.00      1.00      1.00     85443

Accuracy: 99.92275552122467%
```

```
# Confusion Matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap="Greys")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f707cee2b90
```

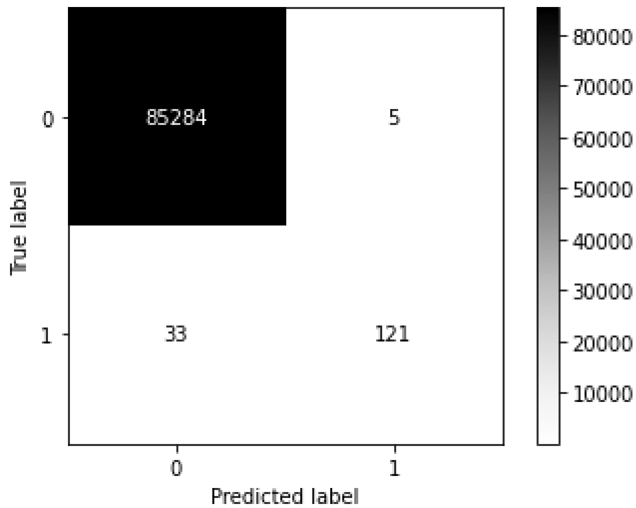



```
# Random Forest
random_forest = RandomForestClassifier(n_estimators= 100)
random_forest.fit(X_train, y_train)

y_pred = random_forest.predict(X_test)
print(classification_report(y_test, y_pred, target_names=["Not-Fraud", "Fraud"]))
print(f'Accuracy: {str(accuracy_score(y_test, y_pred) * 100)}%')
```

```
              precision    recall  f1-score   support

   Not-Fraud       1.00      1.00      1.00     85289
       Fraud       0.95      0.77      0.85       154

    accuracy                           1.00     85443
   macro avg       0.98      0.88      0.92     85443
weighted avg       1.00      1.00      1.00     85443

Accuracy: 99.95084442259751%
```

```
# Plot confusion matrix for Random Forests
# Confusion Matrix

ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap="Greys")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f707cd6f690
```



Solving Class Imbalance:

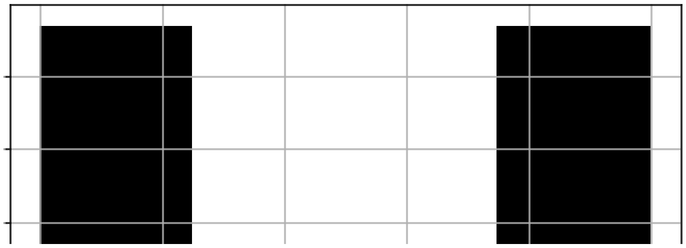
The number of fraud rows are very few compared to the number of non-fraud rows. This imbalance leads to a bad model.

To fix this, we can artificialy generate new samples to compensate for the number difference. Using existing data, we can 'make' new data. This is called over sampling.

This is known as the Synthetic Minority Oversampling Technique (SMOTE)

```
# Performing oversampling on RF and DT
from imblearn.over_sampling import SMOTE

X_new, y_new = SMOTE().fit_resample(X, y)


print(y_new.value_counts())
plt.grid(False)
y_new.hist(bins=4,facecolor='black')
plt.show()
```

```
0    284315
1    284315
Name: Class, dtype: int64
```




```
(X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size= 0.3, random_st
```

```
# Build the Random Forest classifier on the new dataset
random_forest = RandomForestClassifier(n_estimators= 100)
random_forest.fit(X_train, y_train)

y_pred = random_forest.predict(X_test)
print(classification_report(y_test, y_pred, target_names=["Not-Fraud", "Fraud"]))
print(f'Accuracy: {str(accuracy_score(y_test, y_pred) * 100)}%')
```

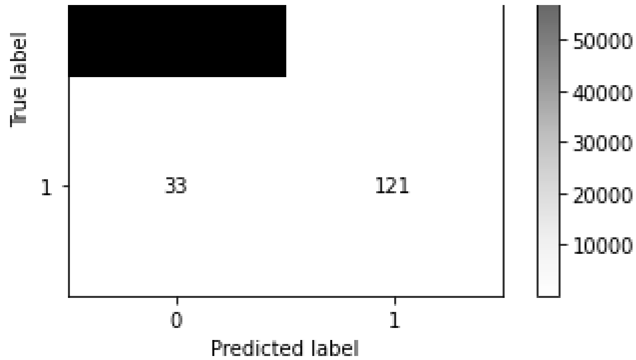
	precision	recall	f1-score	support
Not-Fraud	1.00	1.00	1.00	85289
Fraud	0.96	0.79	0.86	154
accuracy			1.00	85443
macro avg	0.98	0.89	0.93	85443
weighted avg	1.00	1.00	1.00	85443

```
Accuracy: 99.95552590615966%
```

```
# Plot confusion matrix for Random Forests
# Confusion Matrix

ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap="Greys")
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f707ccf36d0
```

Now it is evident that after addressing the class imbalance problem, our Random forest classifier with SMOTE performs far better than the Random forest classifier without SMOTE



✓ 0s completed at 18:07

```
url="https://archive.ics.uci.edu/ml/machine-learning-databases/00242/ENB2012_data.xls
```

```
import pandas as pd
dataset=pd.read_excel(url)
```

```
dataset
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28
...	...	...	...	...	...	...	...	...	...	...
763	0.64	784.0	343.0	220.50	3.5	5	0.4	5	17.88	21.40
764	0.62	808.5	367.5	220.50	3.5	2	0.4	5	16.54	16.88
765	0.62	808.5	367.5	220.50	3.5	3	0.4	5	16.44	17.11
766	0.62	808.5	367.5	220.50	3.5	4	0.4	5	16.48	16.61
767	0.62	808.5	367.5	220.50	3.5	5	0.4	5	16.64	16.03

768 rows × 10 columns

```
dataset.to_csv("data.csv",encoding='utf-8')
```

```
df=pd.read_csv("./data.csv")
```

```
df
```

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28

```
df.isnull().sum()
```

```
Unnamed: 0    0
X1            0
X2            0
X3            0
X4            0
X5            0
X6            0
X7            0
X8            0
Y1            0
Y2            0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  768 non-null    int64
 1   X1          768 non-null    float64
 2   X2          768 non-null    float64
 3   X3          768 non-null    float64
 4   X4          768 non-null    float64
 5   X5          768 non-null    float64
 6   X6          768 non-null    int64
 7   X7          768 non-null    float64
 8   X8          768 non-null    int64
 9   Y1          768 non-null    float64
 10  Y2          768 non-null    float64
dtypes: float64(8), int64(3)
memory usage: 66.1 KB
```

```
df.describe()
```

	Unnamed: 0	X1	X2	X3	X4	X5	X6
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.00000	768.000000
mean	383.500000	0.764167	671.708333	318.500000	176.604167	5.25000	3.500000
std	221.846794	0.105777	88.086116	43.626481	45.165950	1.75114	1.118763
min	0.000000	0.620000	514.500000	245.000000	110.250000	3.50000	2.000000
25%	191.750000	0.682500	606.375000	294.000000	140.875000	3.50000	2.750000
50%	383.500000	0.750000	673.750000	318.500000	183.750000	5.25000	3.500000

```
tar1=df['Y1']
tar2=df['Y2']
data=df.drop(columns=['Y1','Y2'])
```

```
print(tar1,tar2,data)
```

```
0      15.55
1      15.55
2      15.55
3      15.55
4      20.84
       ...
763    17.88
764    16.54
765    16.44
766    16.48
767    16.64
Name: Y1, Length: 768, dtype: float64 0      21.33
1      21.33
2      21.33
3      21.33
4      28.28
       ...
763    21.40
764    16.88
765    17.11
766    16.61
767    16.03
Name: Y2, Length: 768, dtype: float64      Unnamed: 0    X1     X2     X3     X4
0             0  0.98  514.5  294.0  110.25  7.0   2  0.0   0
1             1  0.98  514.5  294.0  110.25  7.0   3  0.0   0
2             2  0.98  514.5  294.0  110.25  7.0   4  0.0   0
3             3  0.98  514.5  294.0  110.25  7.0   5  0.0   0
4             4  0.90  563.5  318.5  122.50  7.0   2  0.0   0
..          ...   ...    ...    ...     ...  ...  ..  ...  ..
763         763  0.64  784.0  343.0  220.50  3.5   5  0.4   5
764         764  0.62  808.5  367.5  220.50  3.5   2  0.4   5
765         765  0.62  808.5  367.5  220.50  3.5   3  0.4   5
766         766  0.62  808.5  367.5  220.50  3.5   4  0.4   5
767         767  0.62  808.5  367.5  220.50  3.5   5  0.4   5
```

```
[768 rows x 9 columns]
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

```
data=sc.fit_transform(data)
```

```
pd.DataFrame(data)
```

	0	1	2	3	4	5	6	7	
0	-1.729797	2.041777	-1.785875	-0.561951	-1.470077	1.0	-1.341641	-1.760447	-1.8145
1	-1.725286	2.041777	-1.785875	-0.561951	-1.470077	1.0	-0.447214	-1.760447	-1.8145
2	-1.720776	2.041777	-1.785875	-0.561951	-1.470077	1.0	0.447214	-1.760447	-1.8145
3	-1.716265	2.041777	-1.785875	-0.561951	-1.470077	1.0	1.341641	-1.760447	-1.8145
4	-1.711755	1.284979	-1.229239	0.000000	-1.198678	1.0	-1.341641	-1.760447	-1.8145
...	...	...	...	...	...	...	...	...	
763	1.711755	-1.174613	1.275625	0.561951	0.972512	-1.0	1.341641	1.244049	1.4113
764	1.716265	-1.363812	1.553943	1.123903	0.972512	-1.0	-1.341641	1.244049	1.4113
765	1.720776	-1.363812	1.553943	1.123903	0.972512	-1.0	-0.447214	1.244049	1.4113
766	1.725286	-1.363812	1.553943	1.123903	0.972512	-1.0	0.447214	1.244049	1.4113
767	1.729797	-1.363812	1.553943	1.123903	0.972512	-1.0	1.341641	1.244049	1.4113

768 rows × 9 columns

```
#train test split
from sklearn.model_selection import train_test_split
x_train1,x_test1,y_train1,y_test1=train_test_split(data,tar1,test_size=0.3)
x_train2,x_test2,y_train2,y_test2=train_test_split(data,tar2,test_size=0.3)
```

```
print(x_train1,x_train2)
```

```
[[ 0.1240402  -0.22861593  0.16235226 ...  1.34164079  0.11736313
   0.12097168]
 [-0.07893467 -0.51241501  0.44067043 ...  0.4472136   0.11736313
  -0.5242106 ]
 [ 1.65762815 -0.51241501  0.44067043 ...  1.34164079  1.2440492
   1.41133622]
 ...
 [-0.35407839  0.24438254 -0.39428407 ... -0.4472136   0.11736313
  -1.16939287]
```

```
 [-1.22010453  0.24438254 -0.39428407 ... -0.4472136  -1.00932293
  -0.5242106 ]
 [ 0.17816684 -0.98541347  0.99730676 ...  1.34164079  0.11736313
   0.12097168]] [[ 0.10148744 -0.03941654 -0.1159659  ...  0.4472136   0.1173631
   0.12097168]
 [-0.34054674 -0.03941654 -0.1159659  ... -1.34164079  0.11736313
  -1.16939287]
 [ 0.06540302  0.52818162 -0.67260223 ...  0.4472136   0.11736313
   0.12097168]
 ...
 [-0.64275377  2.04177671 -1.78587489 ... -0.4472136  -1.00932293
   1.41133622]
 [-0.5254794  -0.22861593  0.16235226 ...  1.34164079 -1.00932293
   1.41133622]
 [ 0.69688041  0.90658039 -0.95092039 ...  0.4472136   1.2440492
  -1.16939287]]
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

```
#for Y1
lr.fit(x_train1,y_train1)
```

```
LinearRegression()
```

```
from sklearn.metrics import r2_score,mean_squared_error
```

```
print(r2_score(lr.predict(x_train1),y_train1))
print(mean_squared_error(lr.predict(x_train1),y_train1))
```

```
0.9120646537104825
7.9190621272554615
```

```
#for y2
lr.fit(x_train2,y_train2)
```

```
LinearRegression()
```

```
print(r2_score(lr.predict(x_train2),y_train2))
print(mean_squared_error(lr.predict(x_train2),y_train2))
```

```
0.8795333304986596
9.36598124256685
```

```
#test for Y1
print(r2_score(lr.predict(x_test1),y_test1))
print(mean_squared_error(lr.predict(x_test1),y_test1))
```

```
0.8560671451548303
11.627398717547873
```

```
#test for Y2
print(r2_score(lr.predict(x_test2),y_test2))
print(mean_squared_error(lr.predict(x_test2),y_test2))
```

```
0.8562339465751755
11.498797573475294
```

```
import numpy as np
```

```
print("Y1 prediction test:",lr.predict(x_test1))
```

```
Y1 prediction test: [34.62906266 32.85159729 31.72340485 16.99468706 35.95685569
 29.45318323 32.65605034 17.54596944 15.68301995 33.41324765 31.65822254
 32.26921885 31.7443731  15.82228054 17.5684097  31.98818777 30.12523827
 29.44429939 29.70978891 17.437419   39.83342113 15.69128984 17.14749195
 33.39080739 15.74820227 15.15417782 37.99291189 15.32728452 36.3030691
 19.43962909 31.30724889 36.51445241 14.53091524 32.03092983 36.1502642
 13.84251071 19.25762643 36.6426786  17.05986938 33.37050558 16.08087138
 29.44216095 33.60732259 14.70402194 17.16779376 31.37243121 38.14357834
 11.90643021 16.92950474 32.80885523 13.93689079 24.37634435 32.83129548
 29.70289022 29.75253098 17.74827391 34.49869802 15.59539738 15.17661808
 17.50322738 31.5006574  17.64186191 15.4013628  15.48898537 31.70310304
 17.53393753 31.57059996 15.95201912 15.50991325 14.02451337 17.30768042
 35.38264951 17.66216372 30.18828215 27.64653222 30.44901142 29.50948171
 19.34311056 31.57126639 29.37911707 17.39316455 19.0193374  36.38408777
 13.71000763 39.89860345 35.42753002 33.38933538 13.84037226 15.29343842
 34.06698603 19.4599309  12.71070781 35.29502694 32.2467786  14.57365731
 15.5744695  31.76614692 14.44329267 16.06056957 37.88498751 36.40438958
 15.42380305 36.23788678 17.41560481 34.67180472 31.65675052 26.37655717
 17.15638791 15.75709822 31.69949259 30.06005595 31.68066279 19.61273579
 19.12726179 16.0178275  31.55029815 14.02237492 37.81980519 17.65389383
 17.48078712 33.54361228 36.2964652  17.70704423 29.77497123 30.36352729
 35.99959776 39.98622602 19.36555081 31.41517327 31.61548047 35.27472513
 29.6649084  36.25372313 32.61330827 14.4005506  38.0132137  19.48237115
 13.9707369  19.27792824 17.37223668 15.88532442 15.33618048 27.83994073
 39.87616319 14.68158169 14.48603473 32.16129447 29.92563768 30.14554008
 33.58635435 33.95906164 14.42299086 39.81098088 36.44927009 31.63792073
 36.02203801 16.39386474 33.43207744 31.54882614 33.65153667 12.84107245
 40.02896809 34.17491041 14.15487801 31.57273841 36.12996239 12.40293861
 16.14605369 36.42682984 32.07367189 17.55423934 38.12327653 15.19691989
 15.21936014 17.90107881 17.62004772 15.04625344 36.53475422 31.17688425
 19.19244411 31.45791534 18.81489449 15.59690976 32.63574853 32.88221478
 38.08053446 12.53760111 31.30511044 27.51616758 13.95933105 27.49372732
 16.84402061 15.13173757 16.28594035 12.690406   17.880777   40.07171015
 17.19023402 33.69427873 32.18159628 17.85833674 33.87143907 31.39273302
```

```
32.74367291 14.48817318 15.61721157 16.10331163 14.61853781 14.2831042
11.86368814 13.92799484 17.77222655 14.08755724 29.77283279 10.46439846
29.86045536 34.71454679 32.78641498 36.57749628 17.35193487 18.98800119
38.16601859 37.94803138 15.77891242 32.11641396 18.90251707 19.06207947
36.38194933 16.4366068  29.73009072]
```

```
Y1_pred=lr.predict(x_test1)
```

```
pd.DataFrame(Y1_pred).describe()
```

	0
count	231.000000
mean	25.196424
std	9.007482
min	10.464398
25%	16.092092
50%	29.379117
75%	32.841446
max	40.071710

```
#Y2 prediction
Y2_pred=lr.predict(x_test2)
pd.DataFrame(Y2_pred).describe()
```

	0
count	231.000000
mean	25.592013
std	8.962726
min	9.291992
25%	16.307311
50%	29.622166
75%	33.380656
max	40.071710

```
pd.DataFrame(Y2_pred).to_csv('Y2_lr_test.csv')
pd.DataFrame(Y1_pred).to_csv('Y1_lr_test.csv')
```

Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures()
```

```
p_train1=poly.fit_transform(x_train1)
p_train2=poly.fit_transform(x_train2)
p_test1=poly.fit_transform(x_test1)
p_test2=poly.fit_transform(x_test2)
```

```
pl=LinearRegression()
```

```
#Y1
pl.fit(p_train1,y_train1)
```

```
LinearRegression()
```

```
print(mean_squared_error(pl.predict(p_train1),y_train1))
print(r2_score(pl.predict(p_train1),y_train1))
```

```
0.18352657799084715
0.9981232850743926
```

```
#test Y1
print(mean_squared_error(pl.predict(p_test1),y_test1))
print(r2_score(pl.predict(p_test1),y_test1))
```

```
0.27710778685189763
0.9974597391528797
```

```
#for Y2
pl.fit(p_train2,y_train2)
```

```
LinearRegression()
```

```
print(mean_squared_error(pl.predict(p_train2),y_train2))
print(r2_score(pl.predict(p_train2),y_train2))
```

```
2.303609881454906
0.9728379617215155
```

```
#test
print(mean_squared_error(pl.predict(p_test2),y_test2))
print(r2_score(pl.predict(p_test2),y_test2))
```

```
2.9420768716955052
0.9665518934566184
```

```
pred_y1_poly=pl.predict(p_test1)
pred_y2_poly=pl.predict(p_test2)
```

```
pd.DataFrame(pred_y1_poly).to_csv('Y1_poly_test.csv')
pd.DataFrame(pred_y2_poly).to_csv('Y2_poly_test.csv')
```

```
print(pd.DataFrame(pred_y1_poly))
```

```
             0
0    40.033746
1    37.975024
2    30.881749
3    15.814724
4    33.874144
..         ...
226  18.011348
227  21.019350
228  43.062973
229  16.307784
230  29.056553

[231 rows x 1 columns]
```

```
pd.DataFrame(pred_y1_poly).describe()
```

	0
count	231.000000
mean	25.545219
std	9.650604
min	11.266578
25%	15.811453
50%	25.991551
75%	33.736829
max	43.299679

```
pd.DataFrame(pred_y2_poly).describe()
```

	0
count	231.000000
mean	25.694197
std	9.399036
min	10.709786
25%	15.982388
50%	26.312103
75%	33.738880

Decision tree

```
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor(max_depth=6,min_samples_leaf=10)
```

```
#for Y1
dt.fit(x_train1,y_train1)
```

```
DecisionTreeRegressor(max_depth=6, min_samples_leaf=10)
```

```
print(mean_squared_error(dt.predict(x_train1),y_train1))
print(r2_score(dt.predict(x_train1),y_train1))
```

```
1.1235147243268866
0.9883995611615297
```

```
#test Y1
print(mean_squared_error(dt.predict(x_test1),y_test1))
print(r2_score(dt.predict(x_test1),y_test1))
```

```
1.2180050794951345
0.9885550386768165
```

```
pd.DataFrame(dt.predict(x_test1)).to_csv("Y1_tree_test.csv")
```

```
#for y2
dt.fit(x_train2,y_train2)
```

```
DecisionTreeRegressor(max_depth=6, min_samples_leaf=10)
```

```
print(mean_squared_error(dt.predict(x_train2),y_train2))
print(r2_score(dt.predict(x_train2),y_train2))
```

```
2.546583846512955
0.9698867498865877
```

```
#test
print(mean_squared_error(dt.predict(x_test2),y_test2))
print(r2_score(dt.predict(x_test2),y_test2))
```

```
4.35498442918061
0.9486740072172756
```

```
pd.DataFrame(dt.predict(x_test2)).to_csv("Y2_tree_test.csv")
```

```
pd.DataFrame(dt.predict(x_test2)).describe()
```

	0
count	231.000000
mean	25.607538
std	9.231381
min	11.697500
25%	15.686000
50%	27.428947
75%	33.448929
max	42.320667

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
```

```
#y1
rf.fit(x_train1,y_train1)
```

```
RandomForestRegressor()
```

```
print(mean_squared_error(rf.predict(x_train1),y_train1))
print(r2_score(rf.predict(x_train1),y_train1))
```

```
0.033432741061452595
0.9996584326248548
```

```
#test y1
print(mean_squared_error(rf.predict(x_test1),y_test1))
print(r2_score(rf.predict(x_test1),y_test1))
```

```
0.32304350385281094
0.997034368712027
```

```
pd.DataFrame(rf.predict(x_test1)).to_csv("Y1_rf_test.csv")
```

```
#for y2
rf.fit(x_train2,y_train2)
```

```
RandomForestRegressor()
```

```
print(mean_squared_error(rf.predict(x_train2),y_train2))
print(r2_score(rf.predict(x_train2),y_train2))
```

```
0.47821228581005437
0.9944257783491156
```

```
#test
print(mean_squared_error(rf.predict(x_test2),y_test2))
print(r2_score(rf.predict(x_test2),y_test2))
```

```
4.188850588528143
0.95151991919091
```

```
pd.DataFrame(rf.predict(x_test2)).to_csv("Y2_rf_test.csv")
```

```
pd.DataFrame(rf.predict(x_test2)).describe()
```

	0
count	231.000000
mean	25.612680
std	9.315536
min	11.130700
25%	15.795600
50%	27.220200
75%	33.611150
max	43.199500

SvR

```
from sklearn.svm import SVR
svr=SVR(kernel='linear')
```

```
#for Y1
svr.fit(x_train1,y_train1)
```

```
SVR(kernel='linear')
```

```
print(mean_squared_error(svr.predict(x_train1),y_train1))
print(r2_score(svr.predict(x_train1),y_train1))
```

```
8.798208848977211
0.8987727009854425
```

```
#test Y1
print(mean_squared_error(svr.predict(x_test1),y_test1))
print(r2_score(svr.predict(x_test1),y_test1))
```

```
8.90344422982757
0.8998139426026828
```

✓ 0s completed at 18:13