Credit Card Fraud Detection - Group 3

M. Snehashish Reddy - AM.EN.U4CSE20246

B. Abhinav - AM.EN.U4CSE20216

G. Praveen Naidu - AM.EN.U4CSE20224

Ch. Pranav Vamsi Krishna - AM.EN.U4CSE20220

G. Nandan Chakravarthi - AM.EN.U4CSE20226

O. Hitesh - AM.EN.U4CSE20252

Before We Begin (About the Dataset)

- The dataset contains transactions made by credit cards in September 2013 by European cardholders.
- This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
- It contains only numerical input variables which are the result of a PCA transformation.
- Unfortunately, due to confidentiality issues, the original features cannot be obtained.

Our Goals

- Gather Sense of the data from the numerical variables.
- Scale and Distribute the data so that it can be used by ML/DL Models.
- Understand the distribution of the data that was provided.

1. Gather Sense of the data

- We need to gather some **basic sense** of the data.
- Except for the **transaction** and **amount** columnsm we have absolutely no clue what the other columns are (privacy reasons).
- What we do know, is that the data has already been scaled.

2. Initial Preprocessing

Importing all required libraries

In [ ]:
```
# Data-preprocessing and Graphing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Scaling, Sampling, and Shuffling of the data
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import StratifiedKFold

# Classification Models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

# Validation Scores
from sklearn.model_selection import cross_val_score
```

Loading the data into a dataframe

In [ ]:
```
credit_df = pd.read_csv('./creditcard.csv')
credit_df.head()
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

5 rows × 31 columns

Getting a statistical description of the data

In [ ]:
```
credit_df.describe()
```

Out[ ]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

8 rows × 31 columns

- From here, it can be observed that the transaction amount is relatively low, with the mean standing at around 88.34 USD

Checking for null values in the data

In [ ]:
```
credit_df.isnull().sum()
```

Out[ ]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

- No null values are present in the dataset

Verifying the columns in the dataset

In [ ]:
```
credit_df.columns
```

Out[ ]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

Checking how skewed the data really is

In [ ]:
```
print('No Frauds', round(credit_df['Class'].value_counts()[0]/len(credit_df) * 100,
print('Frauds', round(credit_df['Class'].value_counts()[1]/len(credit_df) * 100,3),
```

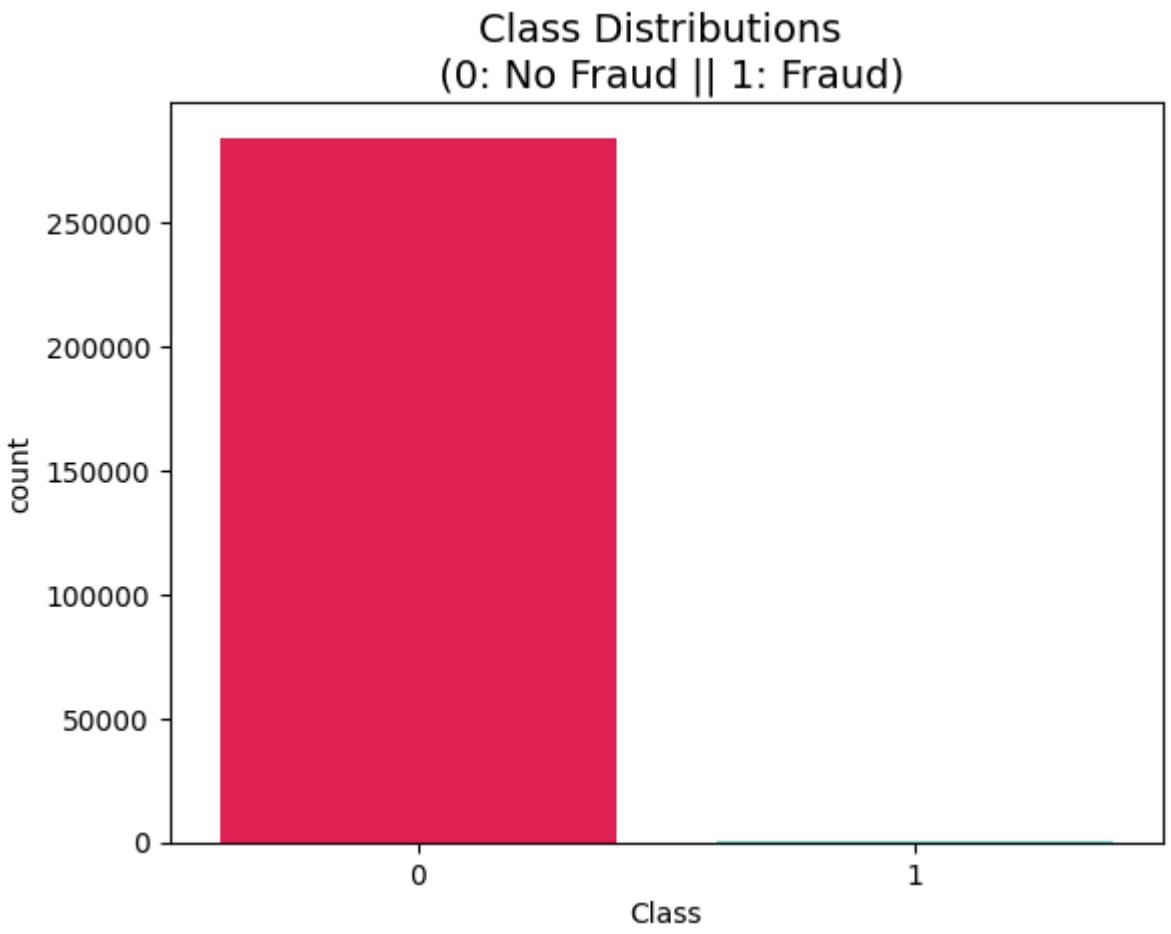
```
No Frauds 99.827 % of time in the data
Frauds 0.173 % of time in the data
```

- The dataset is extremely imbalanced.

- Almost all of the transaction are non-fraud.
- This will be a huge challenge in training a model, as it will definitely overfit, if the data is used as is.
- We need to convert the data into a more useful form that's not highly skewed.

Plotting the dataset balance in Seaborn

In [ ]:
```
colors = ["#FF0043", "#69E2E5"]

sns.countplot(x='Class', palette=colors, data=credit_df)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
plt.show()
```



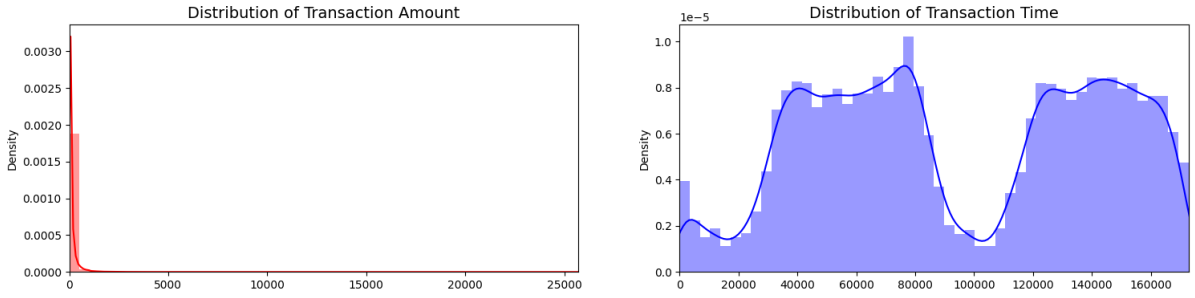
- The distribution gives us an even better idea of skewed the dataset really is
- Techniques have to be applied to make the distributions less skewed

Plotting a distance plot for transaction amounts and transaction times to check if those columns need to be separately scaled

In [ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount_val = credit_df['Amount'].values
time_val = credit_df['Time'].values
```

```
sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])



plt.show()
```

```
C:\Users\ashis\AppData\Local\Temp\ipykernel_7448\1483170820.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(amount_val, ax=ax[0], color='r')
C:\Users\ashis\AppData\Local\Temp\ipykernel_7448\1483170820.py:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(time_val, ax=ax[1], color='b')
```



- It also appears that the **Time** and the **Amount** columns need to be scaled before further analysis is performed on them
- The skewness of the data can be handled using sub-samples which have a 50-50 ratio of fraud and non-fraud transactions

3. Scaling the amount and time columns

Using RobustScaler from sklearn to scale amount and time

In [ ]:
```
rob_scaler = RobustScaler()

credit_df['scaled_amount'] = rob_scaler.fit_transform(credit_df['Amount'].values.re
credit_df['scaled_time'] = rob_scaler.fit_transform(credit_df['Time'].values.reshap

# Dropping the unscaled columns
credit_df.drop(['Time', 'Amount'], axis=1, inplace=True)
```

In [ ]:
```
scaled_amount = credit_df['scaled_amount']
scaled_time = credit_df['scaled_time']

credit_df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
credit_df.insert(0, 'scaled_amount', scaled_amount)
credit_df.insert(1, 'scaled_time', scaled_time)

# Scaling is complete
credit_df.head()
```

Out[ ]:

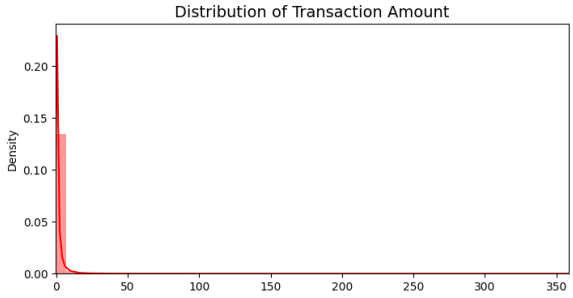
	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6	
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.0
2	4.983721	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7
3	1.418291	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2
4	0.670579	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5

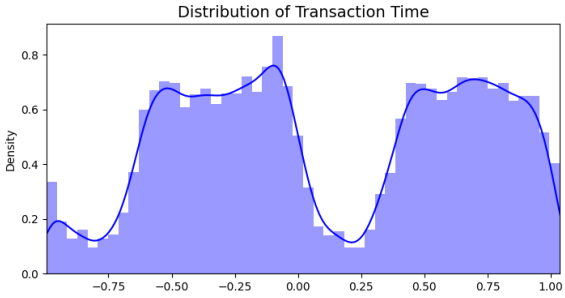
5 rows × 31 columns

Plotting another distanceplot after scaling

In [ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount_val = credit_df['scaled_amount'].values
time_val = credit_df['scaled_time'].values

sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])



plt.show()
```

```
C:\Users\ashis\AppData\Local\Temp\ipykernel_7448\1287034360.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(amount_val, ax=ax[0], color='r')
C:\Users\ashis\AppData\Local\Temp\ipykernel_7448\1287034360.py:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(time_val, ax=ax[1], color='b')
```





- The columns have been sufficiently scaled without losing the features

Splitting the Data (Original DataFrame)

In [ ]:

```
print('No Frauds', round(credit_df['Class'].value_counts()[0]/len(credit_df) * 100,
print('Frauds', round(credit_df['Class'].value_counts()[1]/len(credit_df) * 100,2),

X = credit_df.drop('Class', axis=1)
y = credit_df['Class']

sss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

for train_index, test_index in sss.split(X, y):
    print("Train:", train_index, "Test:", test_index)
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]

# We already have X_train and y_train for undersample data thats why I am using ori
# original_Xtrain, original_Xtest, original_ytrain, original_ytest = train_test_spl

# Check the Distribution of the labels
```

```
# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(original_ytrain, return_counts=T
test_unique_label, test_counts_label = np.unique(original_ytest, return_counts=True
print('-' * 100)

print('Label Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
```

```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
Train: [ 30473  30496  31002 ... 284804 284805 284806] Test: [    0     1     2
... 57017 57018 57019]
Train: [     0      1      2 ... 284804 284805 284806] Test: [ 30473  30496  31002
... 113964 113965 113966]
Train: [     0      1      2 ... 284804 284805 284806] Test: [ 81609  82400  83053
... 170946 170947 170948]
Train: [     0      1      2 ... 284804 284805 284806] Test: [150654 150660 150661
... 227866 227867 227868]
Train: [     0      1      2 ... 227866 227867 227868] Test: [212516 212644 213092
... 284804 284805 284806]
----------------------------------------------------------------------------------
------------------
Label Distributions:

[0.99827076 0.00172924]
[0.99827952 0.00172048]
```

Preventing Model Overfitting with Random Under-Sampling

- We are aiming for a 50/50 ratio for training in the dataset but, considering how low the fraud data is, we will be needing to remove a lot of information bringing the *non-fraud transactions* to the same number of the fraud transactions.
- After implementing it, we will have samples of our dataframe with 50/50 ratios in regard to fraud/non-fraud transactions.
- We will of course be shuffling the data to see if our models can maintain the same accuracy everytime it's being run

In [ ]:
```
# Since our classes are highly skewed we should make them equivalent in order to ha

# Lets shuffle the data before creating the subsamples

df = credit_df.sample(frac=1)

# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
```

```
non_fraud_df = df.loc[df['Class'] == 0][:492]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)

new_df.head()
```

Out[ ]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	
250397	0.108293	0.824645	0.125696	0.652337	-0.560628	-0.358196	1.027344	-0.85
43428	4.781527	-0.507372	-16.526507	8.584972	-18.649853	9.505594	-13.793819	-2.83
4209	1.016558	-0.950916	1.299369	-0.852354	-0.974292	-1.573454	1.406209	3.29
18472	-0.297911	-0.648046	-1.060676	2.608579	-2.971679	4.360089	3.738853	-2.72
190368	2.150493	0.518227	-2.272473	2.935226	-4.871394	2.419012	-1.513022	-0.48

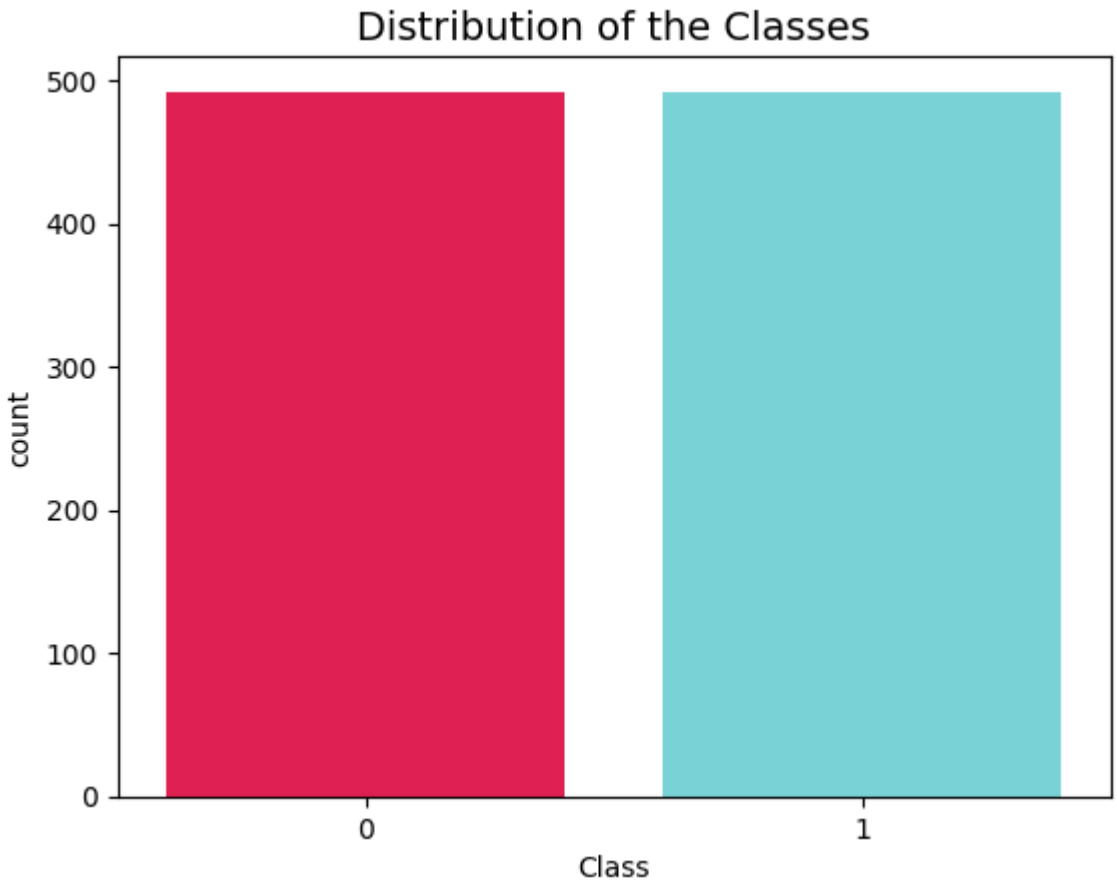
5 rows × 31 columns

Verifying the distrubution of the classes in our samples

In [ ]:

```
print('Distribution of the Classes in the subsample dataset')
print(new_df['Class'].value_counts()/len(new_df))



sns.countplot(x='Class', palette=colors, data=new_df)
plt.title('Distribution of the Classes', fontsize=14)
plt.show()
```

```
Distribution of the Classes in the subsample dataset
0    0.5
1    0.5
Name: Class, dtype: float64
```



- It can clearly be seen that the skewedness of the data has been mostly fixed.
- We will proceed with undersampling of the data to train our models

Training multiple models and comparing their accuracies

In [ ]:
```
# Undersampling before cross validation
X = new_df.drop('Class', axis=1)
y = new_df['Class']


# Explicit usage for undersampling.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Turn the values into an array for feeding the classification algorithms.
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_test = y_test.values

# Implementing simple classifiers

classifiers = {
    "LogisiticRegression": LogisticRegression(),
    "KNearest": KNeighborsClassifier(),
    "Support Vector Classifier": SVC(),
    "DecisionTreeClassifier": DecisionTreeClassifier()
```

```
}


for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    training_score = cross_val_score(classifier, X_train, y_train, cv=5)
    print("Classifiers: ", classifier.__class__.__name__, "Has a training score of"
```

```
Classifiers:  LogisticRegression Has a training score of 94.0 % accuracy score
Classifiers:  KNeighborsClassifier Has a training score of 94.0 % accuracy score
Classifiers:  SVC Has a training score of 93.0 % accuracy score
Classifiers:  DecisionTreeClassifier Has a training score of 91.0 % accuracy score
```

This completes the FDS Project from data pre-processing to model training