Importing the Dependencies

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('creditcard.csv')
```

```
# first 5 rows of the dataset
credit_card_data.head()
```

	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
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

```
credit_card_data.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

```
# dataset informations
credit_card_data.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
```

```
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

```
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
```

```
# distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

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

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

```
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

```
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

```
# statistical measures of the data
legit.Amount.describe()
```

```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
```

```
max       25691.160000
Name: Amount, dtype: float64
```

```
fraud.Amount.describe()
```

```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

```
# compare the values for both transactions
credit_card_data.groupby('Class').mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.0096
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5687

Under-Sampling

Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

Number of Fraudulent Transactions --> 492

```
legit_sample = legit.sample(n=492)
```

Concatenating two DataFrames

```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

```
new_dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7
203131	134666.0	-1.220220	-1.729458	-1.118957	-0.266099	0.823338	-0.098556	-0.407751
95383	65279.0	-1.295124	0.157326	1.544771	-2.468209	-1.683113	-0.623764	-0.371798
99706	67246.0	-1.481168	1.226490	1.857550	2.980777	-0.672645	0.581449	-0.143172
153895	100541.0	-0.181013	1.395877	1.204669	4.349279	1.330126	1.277520	1.568221
249976	154664.0	0.475977	-0.573662	0.480520	-2.524647	-0.616284	-0.361317	-0.347861

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050

```
new_dataset['Class'].value_counts()
```

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

```
new_dataset.groupby('Class').mean()
```

	Time	V1	V2	V3	V4	V5	V6	V
Class								
0	96783.638211	-0.053037	0.055150	-0.036786	-0.046439	0.077614	-0.023218	-0.00070
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.56873

Splitting the data into Features & Targets

```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

```
print(X)
```

```
             Time        V1        V2  ...       V27       V28  Amount
203131  134666.0 -1.220220 -1.729458  ...  0.173995 -0.023852  155.00
95383    65279.0 -1.295124  0.157326  ...  0.317321  0.105345   70.00
99706    67246.0 -1.481168  1.226490  ... -0.546577  0.076538   40.14
153895  100541.0 -0.181013  1.395877  ... -0.229857 -0.329608  137.04
249976  154664.0  0.475977 -0.573662  ...  0.058961  0.012816   19.60
...          ...       ...       ...  ...       ...       ...     ...
279863  169142.0 -1.927883  1.125653  ...  0.292680  0.147968  390.00
280143  169347.0  1.378559  1.289381  ...  0.389152  0.186637    0.76
280149  169351.0 -0.676143  1.126366  ...  0.385107  0.194361   77.89
281144  169966.0 -3.113832  0.585864  ...  0.884876 -0.253700  245.00
281674  170348.0  1.991976  0.158476  ...  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

```
print(Y)
```

```
203131    0
95383     0
99706     0
153895    0
249976    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Split the data into Training data & Testing Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

Model Training

Logistic Regression

```
model = LogisticRegression()
```

```
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Model Evaluation

Accuracy Score

```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9415501905972046
```

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9390862944162437
```

```
# Convert continuous target variable into binary classification labels
# Assuming 1 represents fraud and 0 represents non-fraud
y_train_binary = (y_train > 0).astype(int)

# Resampling to balance the classes
oversampler = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversampler.fit_resample(X_train, y_train_binary)

# Model training and evaluation
models = {
    'Random Forest': RandomForestClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'KNN': KNeighborsClassifier(),
    'Logistic Regression': LogisticRegression(),
    'SVM': SVC()
}

# Train and evaluate models on resampled data
for name, model in models.items():
    model.fit(X_resampled, y_resampled)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)

    print(f'{name} Metrics:')
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1 Score: {f1:.4f}')
    print(f'ROC AUC: {roc_auc:.4f}')
    print()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Undefir
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Undefir
  _warn_prf(average, modifier, msg_start, len(result))
Random Forest Metrics:
Accuracy: 0.9992
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
ROC AUC: 0.5000

Decision Tree Metrics:
Accuracy: 0.9975
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
```

```
ROC AUC: 0.4992

KNN Metrics:
Accuracy: 0.9992
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
ROC AUC: 0.5000

Logistic Regression Metrics:
Accuracy: 0.9900
Precision: 0.0769
Recall: 1.0000
F1 Score: 0.1429
ROC AUC: 0.9950

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Converger
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
SVM Metrics:
Accuracy: 0.8234
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
ROC AUC: 0.4121
```

```
# Importing necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import RandomOverSampler

# Load your dataset
# Assuming 'data' is your DataFrame with features and labels, and 'target' is your label col
# Replace 'data.csv' with your actual dataset file
data = pd.read_csv('creditcard.csv')

# Handling missing values
imputer = SimpleImputer(strategy='mean')
data = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)

# Splitting the data into features and labels
X = data.drop('Class', axis=1)  # Assuming 'Class' is the label column
y = data['Class']

# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert continuous target variable into binary classification labels
# Assuming 1 represents fraud and 0 represents non-fraud
y_train_binary = (y_train > 0).astype(int)
y_test_binary = (y_test > 0).astype(int)

# Resampling to balance the classes for training data only
oversampler = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversampler.fit_resample(X_train, y_train_binary)

# Model training and evaluation
models = {
    'Random Forest': RandomForestClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'KNN': KNeighborsClassifier(),
    'Logistic Regression': LogisticRegression(),
    'SVM': SVC()
}

# Train and evaluate models on training data
print("Training Data Metrics:")
for name, model in models.items():
    model.fit(X_resampled, y_resampled)
```

```
    y_train_pred = model.predict(X_resampled)

    accuracy_train = accuracy_score(y_resampled, y_train_pred)
    precision_train = precision_score(y_resampled, y_train_pred)
    recall_train = recall_score(y_resampled, y_train_pred)
    f1_train = f1_score(y_resampled, y_train_pred)
    roc_auc_train = roc_auc_score(y_resampled, y_train_pred)

    print(f'{name} Metrics:')
    print(f'Accuracy: {accuracy_train:.4f}')
    print(f'Precision: {precision_train:.4f}')
    print(f'Recall: {recall_train:.4f}')
    print(f'F1 Score: {f1_train:.4f}')
    print(f'ROC AUC: {roc_auc_train:.4f}')
    print()

# Evaluate models on testing data
print("Testing Data Metrics:")
for name, model in models.items():
    y_test_pred = model.predict(X_test)

    accuracy_test = accuracy_score(y_test_binary, y_test_pred)
    precision_test = precision_score(y_test_binary, y_test_pred)
    recall_test = recall_score(y_test_binary, y_test_pred)
    f1_test = f1_score(y_test_binary, y_test_pred)
    roc_auc_test = roc_auc_score(y_test_binary, y_test_pred)

    print(f'{name} Metrics:')
    print(f'Accuracy: {accuracy_test:.4f}')
    print(f'Precision: {precision_test:.4f}')
    print(f'Recall: {recall_test:.4f}')
    print(f'F1 Score: {f1_test:.4f}')
    print(f'ROC AUC: {roc_auc_test:.4f}')
    print()
```

```
Training Data Metrics:
Random Forest Metrics:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
ROC AUC: 1.0000

Decision Tree Metrics:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
ROC AUC: 1.0000

KNN Metrics:
```

```
Accuracy: 0.9996
Precision: 0.9992
Recall: 1.0000
F1 Score: 0.9996
ROC AUC: 0.9996

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression Metrics:
Accuracy: 0.9961
Precision: 0.9923
Recall: 1.0000
F1 Score: 0.9961
```

```
from sklearn.ensemble import RandomForestClassifier

# Instantiate the Random Forest classifier
rf_classifier = RandomForestClassifier()

# Train the Random Forest classifier on the resampled training data
rf_classifier.fit(X_resampled, y_resampled)

# Predictions on training data
y_train_pred_rf = rf_classifier.predict(X_resampled)

# Predictions on testing data
y_test_pred_rf = rf_classifier.predict(X_test)

# Calculate accuracy on training data
accuracy_train_rf = accuracy_score(y_resampled, y_train_pred_rf)

# Calculate accuracy on testing data
accuracy_test_rf = accuracy_score(y_test_binary, y_test_pred_rf)
```