# Analysis Report

> **Note**
>
> Source code files are included below as markdown code blocks, whole project source is in this link

## Task 4: Report Findings

### 1. Ratios

```
Class
0    99.827251
1     0.172749
```

| Sampling Technique | Class Ratios | Selected Features | Accuracy | Decision Tree | | | KNN |
|---|---|---|---|---|---|---|---|
| | | | | *Precision* | *Recall* | *F-Measure* | *Precision* |
| Random | 0: 99.834% 1: 0.166 % | `['Time', 'V4', 'V9', 'V14', 'V17']` | `0.9988` | `[0.9997, 0.5588]` | `[0.9991, 0.7917]` | `0.6552` | `[0.9985, 0.0]` |
| Stratified | 0: 99.828% 1: 0.172% | `['V10', 'V12', 'V14', 'V17', 'V28']` | `0.9993` | `[0.9997, 0.75]` | `[0.9996, 0.8077]` | `0.7778` | `[0.9997, 0.875]` |
| Clusterd | 0: 99.785% 1: 0.215% | `['Time', 'V9', 'V11', 'V18', 'V28']` | `0.9989` | `[0.9994, 0.6698]` | `[0.4667, 0.3333]` | `0.5714` | `[0.9997, 0.8]` |

## Task 5: Conclusion

## 1. Best Sampling Technique

The **Stratified Sampling** technique produced the best results based on the evaluation metrics:

- **Highest Accuracy** for both Decision Tree (`0.9993`) and KNN (`0.9995`).
- **Balanced Precision, Recall, and F-measure**:
  - **Decision Tree**: F-measure `0.7778`
  - **KNN**: F-measure `0.84`

This means that Stratified Sampling ensures better representation of the minority class, leading to improved classifier performance.

---

## 2. Features Selected for Each Sampling Technique

- **Random Sampling**: `['Time', 'V4', 'V9', 'V14', 'V17']`
- **Stratified Sampling**: `['V10', 'V12', 'V14', 'V17', 'V28']`
- **Clustered Sampling**: `['Time', 'V9', 'V11', 'V18', 'V28']`

---

## 3. Classifier Performance Summary

### Random Sampling

- Decision Tree: Moderate F-measure `0.6552`
- KNN: Poor performance (F-measure `0.0` for Class 1).

### Stratified Sampling

- Decision Tree: Balanced performance with F-measure `0.7778`.
- KNN: Best performance with F-measure `0.84`.

### Clustered Sampling

- Decision Tree: Low F-measure `0.5714`.
- KNN: Moderate F-measure `0.64`.

---

# Code

- File Structure (Separate `.py` & `.csv` files to minimize memory usage on my potato laptop):

```
.
└── assignment-2
    ├── main.py
    ├── task2.py
    ├── creditcard.csv
    ├── rand.csv
    ├── strat.csv
    ├── cluster.csv
    └── report.md
```

`task2.py`

```python
import pandas as pd
import numpy as np
df = pd.read_csv('creditcard.csv')


# Random
randSample = df.sample(n=50000, random_state=42)
randSample.to_csv('rand.csv', index=False)

n = 50000/len(df)

# Strat
strat_sample=df.groupby('Class').apply(lambda x: x.sample(frac=n))
strat_sample.to_csv('strat.csv', index=False)

# Cluster
clusters = np.array_split(df, 4)

cClusters = np.random.choice(len(clusters), 2 , replace=False)
cSample = np.concatenate([clusters[i] for i in cClusters])
cdf = pd.DataFrame(cSample, columns=df.columns)

cdf.to_csv('cluster.csv', index=False)
```

`main.py`

```python

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.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.feature_selection import SequentialFeatureSelector


# Print the ratio for each class.
def calcRatio(dFrame):
    return dFrame.value_counts(normalize=True) *100



def getFeatures(data):
    X = data.iloc[:, 0:30]
    y = data['Class']

    trainData, testData, trainLabel, testLabel = train_test_split(X, y,
test_size=0.2, random_state=42)

    rf = RandomForestClassifier()
    rf.fit(trainData, trainLabel)

    # Initialize Feature Selector
    sfs = SequentialFeatureSelector( rf, direction="forward",
scoring="accuracy",  n_features_to_select=5)
    sfs.fit(trainData, trainLabel)
    selected_features = X.columns[sfs.get_support()].tolist()
    print("Selected Features:", selected_features)
    # ============================
    # Step 3: Create a New Dataset with Selected Features    #
============================    X_selected = data[selected_features]

    # Split the new dataset with test_size=0.33

    X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
test_size=0.33, random_state=42)

    # Initialize classifiers
    dtClassifier = DecisionTreeClassifier(random_state=42)
    knn_classifier = KNeighborsClassifier(n_neighbors=7)

    # Decision Tree Classifier
    print("\nDecision Tree Results:")
    print("-----------------------------------+-")
```

```
    dtClassifier.fit(X_train, y_train)
    dtPred = dtClassifier.predict(X_test)
    dtAccuracy = accuracy_score(y_test, dtPred)
    dtPrecision = precision_score(y_test, dtPred, average=None)
    dtRecall = recall_score(y_test, dtPred, average=None)
    dtF1 = f1_score(y_test, dtPred )

    print(f"Accuracy: {dtAccuracy}")
    print(f"Precision DT: {dtPrecision}")
    print(f"Recall DT: {dtRecall}")
    print(f"F1 DT: {dtF1}")

    # KNN Classifier
    print("\nKNN Results:")
    print("----------------------------------------")

    knn_classifier.fit(X_train, y_train)
    knn_pred = knn_classifier.predict(X_test)

    knn_accuracy = accuracy_score(y_test, knn_pred)
    knn_precision = precision_score(y_test, knn_pred, average=None)
    knn_recall = recall_score(y_test, knn_pred, average=None)
    knnF1 = f1_score(y_test, knn_pred )

    print(f"Accuracy KNN: {knn_accuracy}")
    print(f"Precision KNN: {knn_precision}")
    print(f"Recall KNN: {knn_recall}")
    print(f"F1 KNN: {knnF1}")



# df = pd.read_csv("creditcard.csv")
# print(calcRatio(df["Class"]))
# print("====================================")


# Random
rdf = pd.read_csv("rand.csv")

print("Ratio rdf:")
print(calcRatio(rdf["Class"]))

getFeatures(rdf)

print("++++============================================================+++++++")
```

```
# Stratified
sdf = pd.read_csv("strat.csv")

print("Ratio Stratified:")
print(calcRatio(sdf["Class"]))
getFeatures(sdf)

print("++++============================================================+++++++")

# Cluster
cdf = pd.read_csv("cluster.csv")

print("Ratio Cluster:")
print(calcRatio(cdf["Class"]))

getFeatures(cdf)
```