In [117]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import time
```

In [118]:
```
# Load the CSV file into a DataFrame
df = pd.read_csv("D:\BITS M.TECH\SEM 2\ML Ops\Assignments\creditcard.csv", nrows = 1

# Separate features and target variable
X = df.drop(['Class'], axis=1)
y = df['Class']
```

In [119]:
```
df.shape
```

Out[119]: (150000, 31)

In [120]:
```
df.head()
```

Out[120]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 31 columns

In [121]:
```
# Vanilla Section
print("\n--- Part 1: Vanilla Section ---")

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

start_time_vanilla = time.time()

# Preprocessing
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)

vanilla_preprocessing_time = time.time() - start_time_vanilla
print("Vanilla Section Preprocessing Time:", vanilla_preprocessing_time)
```

```
--- Part 1: Vanilla Section ---
Vanilla Section Preprocessing Time: 0.08977031707763672
```

In [122]:
```
# Training vanilla Random Forest
start_time_rf = time.time()

model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(X_train_normalized, y_train)

y_pred_rf = model_rf.predict(X_test_normalized)

vanilla_rf_time_taken = time.time() - start_time_rf
print("Vanilla RF Training Time:", vanilla_rf_time_taken)
```

```
Vanilla RF Training Time: 102.76230335235596
```

In [123]:
```
# Logging metrics for vanilla Random Forest
vanilla_rf_accuracy = accuracy_score(y_test, y_pred_rf)
vanilla_rf_report = classification_report(y_test, y_pred_rf)

print("\nVanilla Random Forest Training Time:", vanilla_rf_time_taken)
print("Vanilla Random Forest Accuracy:", vanilla_rf_accuracy)
print("Vanilla Random Forest Report:\n", vanilla_rf_report)
```

```
Vanilla Random Forest Training Time: 102.76230335235596
Vanilla Random Forest Accuracy: 0.9994
Vanilla Random Forest Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     29933
           1       0.98      0.75      0.85        67

    accuracy                           1.00     30000
   macro avg       0.99      0.87      0.92     30000
weighted avg       1.00      1.00      1.00     30000
```

Optimisation - Scaling model size

In [124]:
```
# Parallel Training Section
print("\n--- Part 2: Parallel Training Section ---")
start_time_rf = time.time()
start_time_parallel = time.time()
```

```
--- Part 2: Parallel Training Section ---
```

In [125]:
```
# Define function to train Random Forest
def train_rf(X_train, y_train):
    model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
    model_rf.fit(X_train, y_train)
    return model_rf
```

In [126]:
```
# Parallel training of Random Forest
num_cores = multiprocessing.cpu_count()
print("Num cores",num_cores)
models_rf = Parallel(n_jobs=num_cores)(delayed(train_rf)(X_train_normalized, y_train
```

```
Num cores 8
```

In [127]:
```
parallel_time_taken = time.time() - start_time_rf
print("Parallel RF Training Time:", parallel_time_taken)
```

```
Parallel RF Training Time: 185.56848883628845
```

In [128]:
```
# Predict using all parallel trained models
start_time_prediction = time.time()
predictions_rf = [model.predict(X_test_normalized) for model in models_rf]
prediction_time_taken = time.time() - start_time_prediction
```

In [129]:
```
# Combine predictions and calculate metrics
combined_predictions_rf = np.vstack(predictions_rf)
final_predictions_rf = np.mean(combined_predictions_rf, axis=0)
```

In [130]:
```
# Calculate metrics for combined predictions
accuracy_rf = accuracy_score(y_test, final_predictions_rf)
report_rf = classification_report(y_test, final_predictions_rf)
```

In [131]:
```
print("\nParallel Random Forest Training Time:", parallel_time_taken)
print("Parallel Random Forest Prediction Time:", prediction_time_taken)
print("Parallel Random Forest Accuracy:", accuracy_rf)
print("Parallel Random Forest Report:\n", report_rf)
```

```
Parallel Random Forest Training Time: 185.56848883628845
Parallel Random Forest Prediction Time: 1.5604314804077148
Parallel Random Forest Accuracy: 0.9994
Parallel Random Forest Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     29933
           1       0.98      0.75      0.85        67

    accuracy                           1.00     30000
   macro avg       0.99      0.87      0.92     30000
weighted avg       1.00      1.00      1.00     30000
```

Optimisation - Scaling data size

In [106]:
```
from dask_ml.datasets import make_classification
from dask_ml.model_selection import train_test_split
from dask_ml.ensemble import BlockwiseVotingClassifier
import sklearn.linear_model
import time
```

In [107]:
```
# Parallel Training Section
print("\n--- Part 2: Parallel Training Section ---")
start_time_rf = time.time()
start_time_parallel = time.time()
```

```
--- Part 2: Parallel Training Section ---
```

In [108]:
```
# generate a large classification dataset
X, y = make_classification(n_samples=1000, chunks=150)

# split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

In [109]:
```
# train a random forest classifier on the training set
subestimator = sklearn.linear_model.RidgeClassifier(random_state=0)
clf = BlockwiseVotingClassifier(subestimator, voting='hard', classes=[0, 1])
clf.fit(X_train, y_train)

# predict the labels of the test set
y_pred = clf.predict(X_test)

# evaluate the accuracy of the classifier
accuracy = (y_pred == y_test).mean().compute()
print(accuracy)
```

```
0.66
```

In [112]:
```
parallel_time_taken = time.time() - start_time_rf
print("Parallel RF Training Time:", parallel_time_taken)
```

```
Parallel RF Training Time: 51.59974932670593
```