Praktikum 4.
Anomaly Detection
(Fraud Detection of Credit Card)
Random Forest dan Logistic Regression

By: Rastri Prathivi, M.Kom.

```python
# import library
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, plot_roc_curve, confusion_matrix, plot_confusion_matrix

```

UNTUK READ DATASET SILAKAN DISESUAIKAN SENDIRI CODINGNYA SEHINGGA DATA DAPAT DIBACA

```python
# Read the dataset
df = pd.read_csv("creditcard.csv")
df.head()
```

Out[2]:

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

5 rows × 31 columns

```
[ ]
# Printing quick information about the dataset
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
```

[ ]
```python
# Checking missing values in each column
df.isnull().sum()
```

Out[4]:

```
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
```

[ ]
```python
# Identify duplicate values and mark all the duplicates as true
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.duplicated.html
df[df.duplicated(keep=False)]
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7
32	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.7
33	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.7
34	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.6
35	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.6
112	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.
...	...	...	...	...	...	...	...	...
283485	171627.0	-1.457978	1.378203	0.811515	-0.603760	-0.711883	-0.471672	-0.
284190	172233.0	-2.667936	3.160505	-3.355984	1.007845	-0.377397	-0.109730	-0.
284191	172233.0	-2.667936	3.160505	-3.355984	1.007845	-0.377397	-0.109730	-0.
284192	172233.0	-2.691642	3.123168	-3.339407	1.017018	-0.293095	-0.167054	-0.
284193	172233.0	-2.691642	3.123168	-3.339407	1.017018	-0.293095	-0.167054	-0.

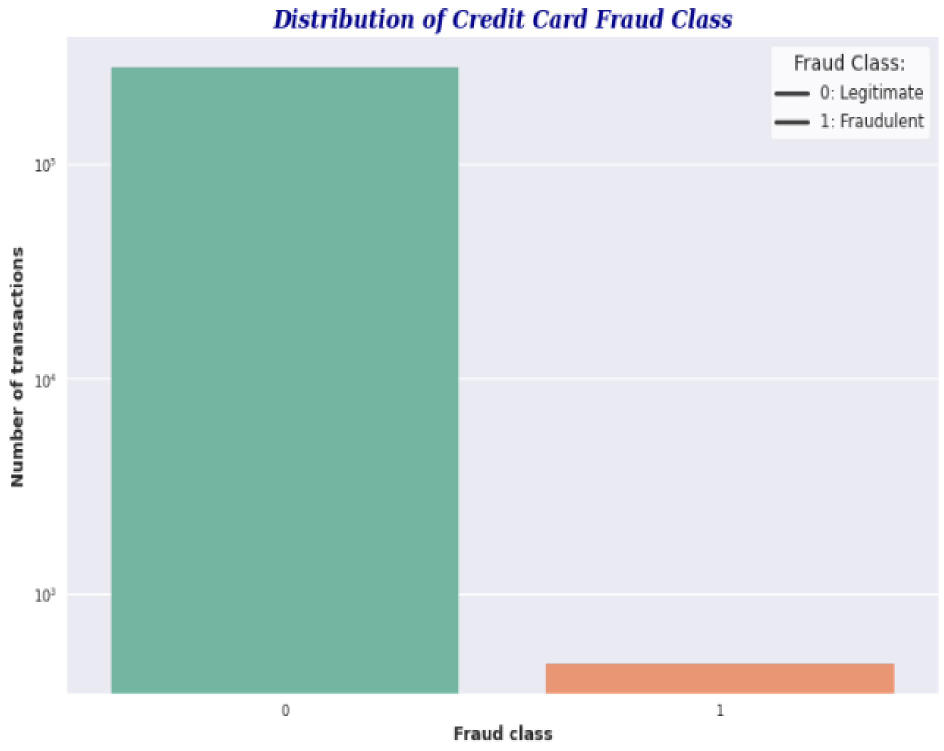
1854 rows × 31 columns

```
[ ]
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop_duplicates.html
# drop data duplicated
df = df.drop_duplicates(keep='first')
```

```
[ ]
# Check the distribution of the credit card fraud cases
class_proportion = df['Class'].value_counts()
class_proportion
```

```
Out[7]:
0    283253
1       473
Name: Class, dtype: int64
```

```python
# Plotting a barchart to see the the distribution of the credit card fraud cases
plt.style.use('seaborn')
font1 = {'family': 'serif',
         'fontstyle': 'italic',
         'fontsize': 16,
         'fontweight': 'bold',
         'color': 'DarkBlue'}
font2 = {'weight': 'bold', 'size': 12}
font3 = {'weight': 'normal', 'size': 12}

fig, ax = plt.subplots(figsize=(12, 8))
sns.barplot(class_proportion.index, class_proportion.values, palette='Set2')
ax.set_title('Distribution of Credit Card Fraud Class', fontdict=font1)
ax.set_xlabel('Fraud class', fontdict=font2)
ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
ax.set_ylabel('Number of transactions', fontdict=font2)
ax.set_yscale('log')
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels=['0: Legitimate', '1: Fraudulent'], prop= font3,
          title ='Fraud Class:', title_fontsize=14,
          frameon=True, facecolor='white')
plt.show()
```



Feature Engineering and Data Modeling

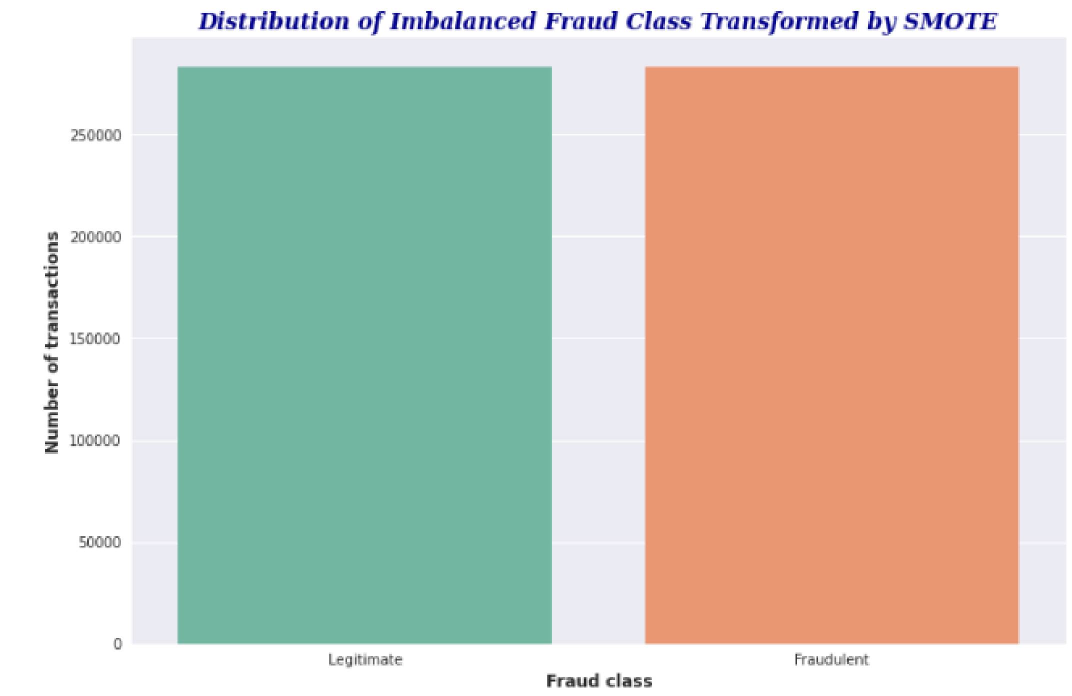
```
[ ]
# Check the proportion of the fraud cases and identify the imbalance
df['Class'].value_counts(normalize=True)
```

```
[ ]
# Arrange the dataset into features matrix and target vector
# Drop the 'Time' variable as it does not that much help our analysis
X = df.drop(columns=(['Time', 'Class']))
y = df['Class']
```

```
[ ]
# Make a SMOTE instance, then fit and apply it in one step
# to create an oversampled version of our dataset.

sm = SMOTE(sampling_strategy='auto', random_state=3, k_neighbors=5)
X_oversampled , y_oversampled = sm.fit_resample(X, y)
```

```
[ ]
# Summarize the fraud class distribution of the new SMOTE-transformed dataset
unique_original, counts_original = np.unique(y, return_counts=True)
unique_oversampled, counts_oversampled = np.unique(y_oversampled, return_counts=True)

print('Original fraud class distribution:', dict(zip(unique_original, counts_original)))
print('New transformed fraud class distribution:',dict(zip(unique_oversampled, counts_oversampled)))
```

```
# Visualize the SMOTE-transformed target variable
plt.style.use('seaborn')
font1 = {'family': 'serif',
         'fontstyle': 'italic',
         'fontsize': 16,
         'fontweight': 'bold',
         'color': 'DarkBlue'}
font2 = {'weight': 'bold', 'size': 12}

fig, ax = plt.subplots(figsize=(12, 8))
sns.countplot(y_oversampled, palette='Set2', ax=ax)
ax.set_title('Distribution of Imbalanced Fraud Class Transformed by SMOTE', fontdict=font1)
ax.set_xlabel('Fraud class', fontdict=font2)
ax.set_xticklabels(['Legitimate', 'Fraudulent'])
ax.set_ylabel('Number of transactions', fontdict=font2)
plt.show()
```



Random Forest Classifier(RFC)

```
[ ]
# Separate the transformed features matrix and target vector into random train and test subsets
X_train, X_test, y_train, y_test = train_test_split(X_oversampled, y_oversampled, random_state=3)
```

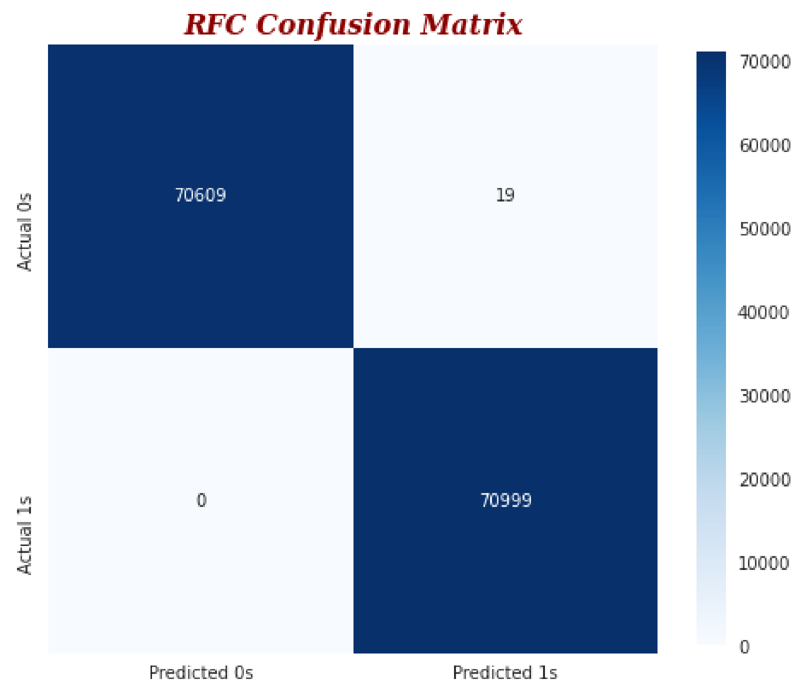
```
[ ]
# Instantiate and fit the model
rfc = RandomForestClassifier(n_estimators=150)
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier(n_estimators=150)
```

```
[ ]
# Model Evalution -classification accuracy
training_rfc_accuracy = rfc.score(X_train, y_train)
testing_rfc_accuracy = rfc.score(X_test, y_test)

print("Training RFC Accuracy:", training_rfc_accuracy)
print("Testing RFC Accuracy:", testing_rfc_accuracy )
```

```
Training RFC Accuracy: 1.0
Testing RFC Accuracy: 0.9998658447894823
```

```python
# Plotting the confusion matrix
fig, ax = plt.subplots(figsize=(8, 8))
font1 = {'family': 'serif',
         'fontstyle': 'italic',
         'fontsize': 16,
         'fontweight': 'bold',
         'color': 'DarkRed'}
font2 = {'weight': 'bold', 'size': 12}

sns.heatmap(confusion_matrix(y_test, rfc.predict(X_test)),
            cmap='Blues',
            square=True,
            annot=True,
            fmt='d',
            cbar_kws={'shrink': 0.8},
            xticklabels=['Predicted 0s', 'Predicted 1s'],
            yticklabels=['Actual 0s', 'Actual 1s'])
ax.set_title('RFC Confusion Matrix', fontdict=font1)
plt.show()
```



```python
# Model evaluation - Sensitivity, Specificity and Precision

TN, FP, FN, TP = confusion_matrix(y_test, rfc.predict(X_test)).flatten()
print("True Negatives:", TN)
print("False Positives:", FP)
print("False Negatives:", FN)
print("True Positives:", TP)

sensitivity = TP/(TP + FN)
specificity = TN/(TN + FP)
precision = TP/(TP + FP)
print("\nSensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
```

```
True Negatives: 70609
False Positives: 19
False Negatives: 0
True Positives: 70999

Sensitivity: 1.0
Specificity: 0.9997309848785184
Precision: 0.9997324621926835
```

```
# Check the predicted probabilities for every observation in the test data subset
# Note that the default classification threshold is 0.5

testing_probabilities= rfc.predict_proba(X_test)
testing_probabilities
```

```
# Convert the testing probabilities into a dataframe
testing_probabilities_df = pd.DataFrame(testing_probabilities, columns=['1 - $p$(X_test)', '$p$(X_test)'])
testing_probabilities_df.head()
```

```
# Get predictions
rfc.predict(X_test)
```

Out[20]:

	1 - p(X_test)	p(X_test)
0	0.0	1.0
1	0.0	1.0
2	0.0	1.0
3	1.0	0.0
4	0.0	1.0

```
[ ]
# Model evaluation -AUC
# Calculate AUC for both training and testing subsets
# Only probabilities being in the positive class is needed for the calculation, that is the second column
training_rfc_AUC = roc_auc_score(y_train, rfc.predict_proba(X_train)[:, 1])
testing_rfc_AUC = roc_auc_score(y_test, rfc.predict_proba(X_test)[:, 1])

print("Training RFC AUC:", training_rfc_AUC)
print("Testing RFC AUC:", testing_rfc_AUC)
```

```
Training RFC AUC: 1.0
Testing RFC AUC: 0.9999915933086132
```

Logist Regression (LGR) Model

```
[ ]
# Separate the transformed features matrix and target vector into random train and test subsets
X_train, X_test, y_train, y_test = train_test_split(X_oversampled, y_oversampled, random_state=3)
```

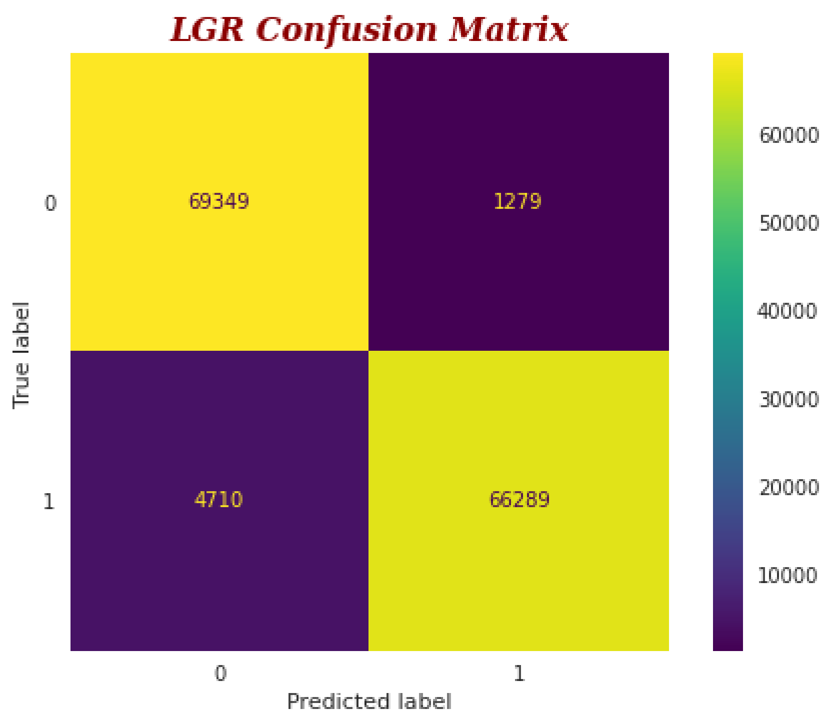
```
[ ]
# define dictionary of hyperparameters
params = {'penalty': ['l1', 'l2'],
          'C': [0.0001, 0.001, 0.01, 10, 50, 100],
          'class_weight': [None, 'balanced']}
```

```
[ ]
# Instantiate Logistic Regression model. N.B: the default solver doesn't support l1 regularization
# Instantiate Grid Search to find the best hyperparameters and fit the model
lgr = LogisticRegression(solver='liblinear')
gs = GridSearchCV(lgr, params, cv = 5)
gs.fit(X_train, y_train)
```

```
[ ]
# Model evaluation - accuracy
training_lgr_accuracy = gs.score(X_train, y_train)
testing_lgr_accuracy = gs.score(X_test, y_test)

print("Training LGR Accuracy:", training_lgr_accuracy)
print("Testing LGR Accuracy:", testing_lgr_accuracy)
```

```
Training LGR Accuracy: 0.958647050101323
Testing LGR Accuracy: 0.9577128654846887
```

```python
# Plotting the confusion matrix
font1 = {'family': 'serif',
         'fontstyle': 'italic',
         'fontsize': 16,
         'fontweight': 'bold',
         'color': 'DarkRed'}

plot_confusion_matrix(gs, X_test, y_test, values_format='d')
plt.title('LGR Confusion Matrix', fontdict=font1)
plt.grid(False)
plt.show()
```



[ ]

```
# Model evaluation - Sensitivity, Specificity and Precision
TN, FP, FN, TP = confusion_matrix(y_test, gs.predict(X_test)).flatten()
print("True Negatives:", TN)
print("False Positives:", FP)
print("False Negatives:", FN)
print("True Positives:", TP)

sensitivity = TP/(TP + FN)
specificity = TN/(TN + FP)
precision = TP/(TP + FP)
print("\nSensitivity:", sensitivity)
print("Specificity:", specificity)
print("Precision:", precision)
```

```
True Negatives: 69349
False Positives: 1279
False Negatives: 4710
True Positives: 66289

Sensitivity: 0.933661037479401l
Specificity: 0.9818910347171094
Precision: 0.981070921146l046
```

```python
# Model evaluation -AUC
# Calculate AUC for both training and testing subsets
# Only probabilities being in the positive class is needed for the calculation, that is the second column
training_lgr_AUC = roc_auc_score(y_train, gs.predict_proba(X_train)[:, 1])
testing_lgr_AUC = roc_auc_score(y_test, gs.predict_proba(X_test)[:, 1])

print("Training LGR AUC:", training_lgr_AUC)
print("Testing LGR AUC:", testing_lgr_AUC)
```

```
Training LGR AUC: 0.9918803796502905
Testing LGR AUC: 0.9915963855793377
```