Telkom University

Laporan Tugas Besar Mata Kuliah Penambangan Data

Klasifikasi Menggunakan *Model Decision Tree*
dan Analisis Aturan Asosiasi

Oleh:

JOSHUA GALILEA

1301170212/ IF-41-GAB03

Pendahuluan

Pada kesempatan kali ini akan dilakukan klasifikasi data menggunakan model *decision tree* dan diakhiri dengan analisis aturan asosiasi. Pada proses klasifikasi data akan digunakan dua model *decision tree,* dengan jumlah fitur yang berbeda. Penentuan parameter depth, pada masing-masing model *decision tree*, akan dilakukan sebelum proses validasi model. Setelah depth masing-masing model *decision tree* ditentukan, akan dilakukan validasi menggunakan metode *stratified k-fold*, sehingga dapat dibandingkan hasil antara model satu dengan model lainnya. Parameter hasil yang akan diperbandingkan antara lain akurasi, recall, dan nilai *mean absolute error* (MAE). Hipotesis yang akan dibuktikan pada hasil dari dua model *decision tree* adalah, "**Dengan memangkas jumlah kolom yang memiliki korelasi rendah terhadap kolom target dapat meningkatkan performa (akurasi) model dan memperkecil kemungkinan misklasifikasi.**" Terakhir, akan dicari aturan asosiasi yang melibatkan keseluruhan data. Selanjutnya pertanyaan yang akan dibuktikan dari hasil analisis aturan asosiasi adalah, **"Rangkaian Item transaksi yang seperti apa yang menyababkan fraudulent dan non-faudulent?"**

Telkom University

Klasifikasi Menggunakan *Model Decision Tree* dan Analisis Aturan Asosiasi

Klasifikasi

Exploratory Data Analysis (EDA)

Pada tahap ini akan dilakukan eksplorasi pada data untuk mendapatkan gambaran secara *holistic* pada data yang digunakan. Bentuk eksplorasi yang akan dilakukan antara lain, mencari nilai statistik (mean, quartil, min, dan max) pada setiap kolom, mencari dan menghitung jumlah baris "null" setiap kolom, memeriksa *imbalace class*, dan mencari korelasi setiap kolom terhadap kolom *fraud*.

EDA - Mencari Nilai Statistik pada Setiap Kolom

Untuk mencari nilai statistik pada setiap kolom, pada python, akan digunakan satu baris perintah sederhana, yakni *data.describe()*. Hasilnya dapat dilihat sebagaimana berikut.

```
data.describe()
```

	trustLevel	totalScanTimeInSeconds	grandTotal	lineItemVoids	scansWithoutRegistration	quantityModifications	scannedLineItemsPerSecond	valuePerSecond
count	498121.000000	498121.000000	498121.000000	498121.000000	498121.000000	498121.000000	4.981210e+05	4.981210e+05
mean	3.503257	915.608772	18.856831	5.495926	5.001281	2.499015	8.320717e+11	4.302878e+12
std	1.707662	528.772880	29.511867	3.447683	3.163795	1.708182	1.611261e+13	3.845879e+13
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	5.461496e-04	0.000000e+00
25%	2.000000	458.000000	1.080000	3.000000	2.000000	1.000000	8.498584e-03	2.734761e-02
50%	4.000000	916.000000	2.730000	5.000000	5.000000	2.000000	1.676338e-02	5.460687e-02
75%	5.000000	1374.000000	32.740000	8.000000	8.000000	4.000000	3.325123e-02	1.094419e-01
max	6.000000	1831.000000	99.990000	11.000000	10.000000	5.000000	9.666667e+14	9.993333e+14

```
data.describe()
```

alScanTimeInSeconds	grandTotal	lineItemVoids	scansWithoutRegistration	quantityModifications	scannedLineItemsPerSecond	valuePerSecond	lineItemVoidsPerPosition	fraud
498121.000000	498121.000000	498121.000000	498121.000000	498121.000000	4.981210e+05	4.981210e+05	4.981210e+05	498121.000000
915.608772	18.856831	5.495926	5.001281	2.499015	8.320717e+11	4.302878e+12	8.006081e+12	0.047633
528.772880	29.511867	3.447683	3.163795	1.708182	1.611261e+13	3.845879e+13	4.049927e+13	0.212989
1.000000	0.000000	0.000000	0.000000	0.000000	5.461496e-04	0.000000e+00	0.000000e+00	0.000000
458.000000	1.080000	3.000000	2.000000	1.000000	8.498584e-03	2.734761e-02	3.846154e-02	0.000000
916.000000	2.730000	5.000000	5.000000	2.000000	1.676338e-02	5.460687e-02	2.222222e-01	0.000000
1374.000000	32.740000	8.000000	8.000000	4.000000	3.325123e-02	1.094419e-01	5.294118e-01	0.000000
1831.000000	99.990000	11.000000	10.000000	5.000000	9.666667e+14	9.993333e+14	3.666667e+14	1.000000

Terlihat sebuah tabel yang memberikan informasi nilai statistik pada setiap kolom, yakni, jumlah data, rataan, standar deviasi, nilai minimum, quartil 1, nilai tengah, quartil 3 dan nilai maksimum.

EDA - Mencari dan Menghitung Jumlah Baris "Null" pada Setiap Kolom

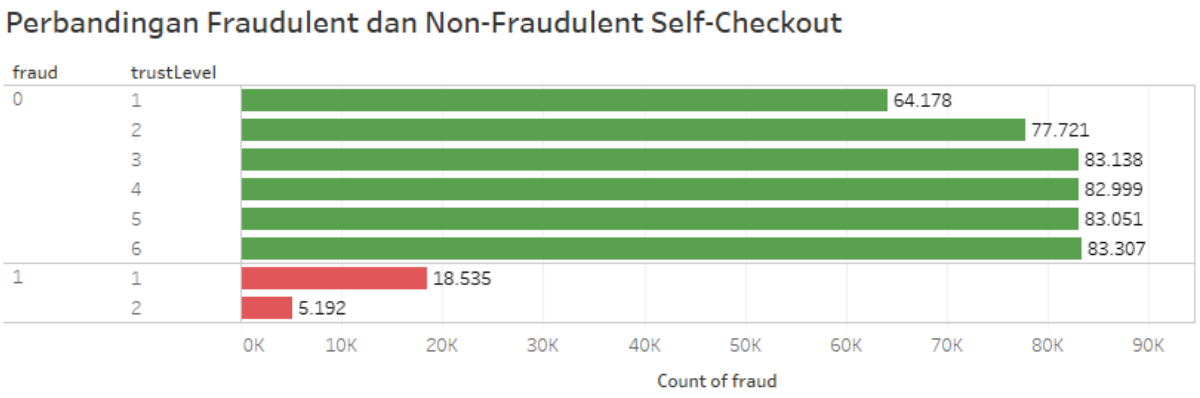
Untuk mencari nilai statistik pada setiap kolom, pada python, akan digunakan satu baris perintah sederhana, yakni *data.info()*. Hasilnya dapat dilihat sebagaimana berikut.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 498121 entries, 0 to 498120
Data columns (total 10 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   trustLevel                 498121 non-null  int64
 1   totalScanTimeInSeconds     498121 non-null  int64
 2   grandTotal                 498121 non-null  float64
 3   lineItemVoids              498121 non-null  int64
 4   scansWithoutRegistration   498121 non-null  int64
 5   quantityModifications      498121 non-null  int64
 6   scannedLineItemsPerSecond  498121 non-null  float64
 7   valuePerSecond             498121 non-null  float64
 8   lineItemVoidsPerPosition   498121 non-null  float64
 9   fraud                      498121 non-null  int64
dtypes: float64(4), int64(6)
memory usage: 38.0 MB
```

Terlihat sebuah tabel yang memberikan informasi mengenai jumlah data *non-null,* yang mengartikan bahwa data yang digunakan tidak memiliki missing value.

EDA - Memeriksa *Imbalace Class*

Untuk mencari kehadiran *imbalace class* pada data, digunakan kolom acuan *fraud*, sehingga ditemukan hasil seperti dibawah ini.



Jelas terlihat, pada gambar di atas, terdapat ketidakseimbangan kelas pada data, yang mana kelas *non-fraudulent* (*fraud* = 0) mendominasi hingga sebesar 95,237% (474.394 entri data), sementara kelas *fraudulent* (*fraud* = 1) hanya sebesar 4,763% (23.727 entri data).

Mencari Korelasi Setiap Kolom Terhadap Kolom *Fraud*

Untuk mencari korelasi antar kolom, digunakan bantuan dari library python seaborn dan matplotlib, sedangkan langkah-langkah yang dilakukan adalah sebagai berikut:

1. Mencari nilai korelasi antar kolom

```
corr = data.corr()
corr
```

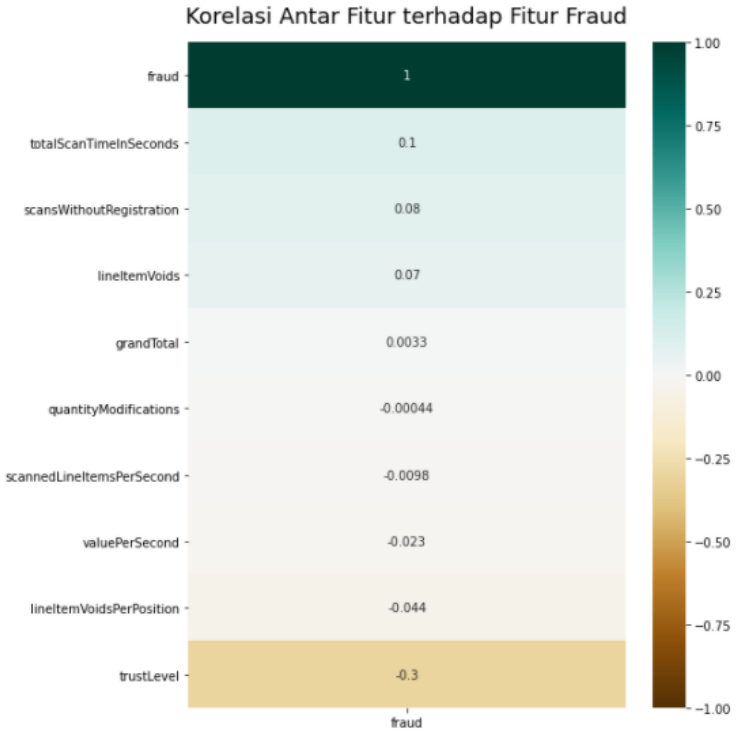
	trustLevel	totalScanTimeInSeconds	grandTotal	lineItemVoids	scansWithoutRegistration	quantityModifications	scannedLineItemsPerSecond	valuePerSecond
trustLevel	1.000000	0.000065	0.000690	-0.000436	0.000518	-0.000806	-0.001649	0.000308
totalScanTimeInSeconds	0.000065	1.000000	0.000734	-0.000619	-0.000886	-0.000602	-0.088432	-0.187608
grandTotal	0.000690	0.000734	1.000000	-0.002432	0.001393	0.000854	0.000782	0.033683
lineItemVoids	-0.000436	-0.000619	-0.002432	1.000000	0.001962	-0.000974	-0.001164	-0.000149
scansWithoutRegistration	0.000518	-0.000886	0.001393	0.001962	1.000000	0.000079	-0.001280	-0.002136
quantityModifications	-0.000806	-0.000602	0.000854	-0.000974	0.000079	1.000000	-0.003478	-0.000888
scannedLineItemsPerSecond	-0.001649	-0.088432	0.000782	-0.001164	-0.001280	-0.003478	1.000000	0.370087
valuePerSecond	0.000308	-0.187608	0.033683	-0.000149	-0.002136	-0.000888	0.370087	1.000000
lineItemVoidsPerPosition	-0.000667	0.001253	-0.001742	0.199080	-0.001079	0.000082	-0.008901	-0.001253
fraud	-0.299178	0.101605	0.003275	0.069627	0.080411	-0.000437	-0.009770	-0.022943

```
corr = data.corr()
corr
```

vel	totalScanTimeInSeconds	grandTotal	lineItemVoids	scansWithoutRegistration	quantityModifications	scannedLineItemsPerSecond	valuePerSecond	lineItemVoidsPerPosition	fraud
000	0.000065	0.000690	-0.000436	0.000518	-0.000806	-0.001649	0.000308	-0.000667	-0.299178
065	1.000000	0.000734	-0.000619	-0.000886	-0.000602	-0.088432	-0.187608	0.001253	0.101605
590	0.000734	1.000000	-0.002432	0.001393	0.000854	0.000782	0.033683	-0.001742	0.003275
436	-0.000619	-0.002432	1.000000	0.001962	-0.000974	-0.001164	-0.000149	0.199080	0.069627
518	-0.000886	0.001393	0.001962	1.000000	0.000079	-0.001280	-0.002136	-0.001079	0.080411
306	-0.000602	0.000854	-0.000974	0.000079	1.000000	-0.003478	-0.000888	0.000082	-0.000437
549	-0.088432	0.000782	-0.001164	-0.001280	-0.003478	1.000000	0.370087	-0.008901	-0.009770
308	-0.187608	0.033683	-0.000149	-0.002136	-0.000888	0.370087	1.000000	-0.001253	-0.022943
567	0.001253	-0.001742	0.199080	-0.001079	0.000082	-0.008901	-0.001253	1.000000	-0.044210
178	0.101605	0.003275	0.069627	0.080411	-0.000437	-0.009770	-0.022943	-0.044210	1.000000

2. Visualisasi nilai korelasi setiap kolom terhadap kolom *fraud*

```
plt.figure(figsize=(8, 10))
heatmap = sns.heatmap(corr[['fraud']].sort_values(by='fraud', ascending=False),
                      vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Korelasi Antar Fitur terhadap Fitur Fraud',
                  fontdict={'fontsize':18}, pad=16);
```



Dari hasil visualisasi, nampak kolom (selain *fraud*) yang memiliki korelasi paling kuat dengan kolom *fraud* adalah kolom *trustLevel* (-0,3) dan kolom *totalScanTimeInSeconds* (0.1), sementara itu kolom dengan korelasi terendah adalah kolom *quantityModifications* (-0.00044) dan kolom *grandTotal* (0,0033).

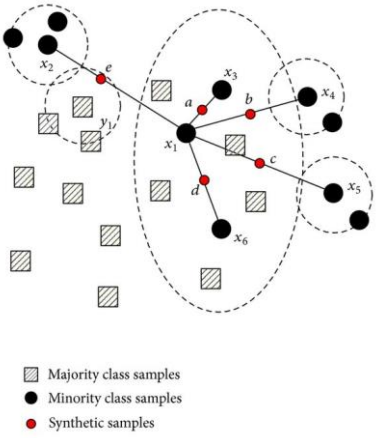
Preprocessing

Dikarenakan proses klasifikasi akan menggunakan dua buah model *decision tree*, dengan memanfaakan jumlah fitur yang berbeda, maka akan terdapat dua tahap preprocessing, yakni preprocessing untuk model yang menggunakan semua fitur dan preprocessing untuk model yang melibatkan beberapa fitur saja.

Preprocessing – Semua Fitur

Preprocessing - Penanganan kelas tidak imbang menggunakan teknik SMOTE.

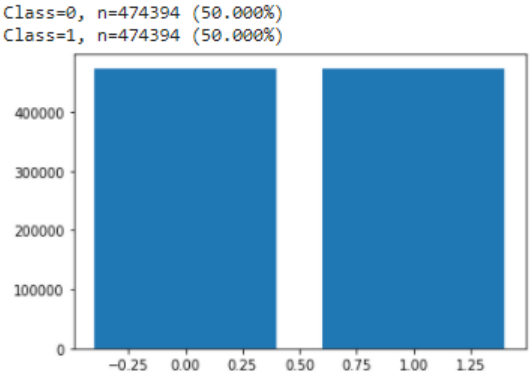
Teknik *Synthetic Minority Oversampling Technique* (SMOTE), yang akan digunakan dalam menangani kasus kelas tak imbang, merupakan teknik *oversampling* data pada kelas minoritas. Teknik ini bekerja dengan cara men-*generate* titik-titik (representasi baris data) baru yang terletak antar titik yang tergolong sebagai kelas minoritas. Generasi titik baru akan usai ketika kelas minoritas memiliki jumlah yang sama dengan kelas mayoritas. Ilustrasi teknik SMOTE dapat dilihat pada gambar dibawah ini.



Pada python, implementasi teknik SMOTE dapat dilakukan dengan menggunakan bantuan dari *library imblearn.over_sampling,* dilanjutkan dengan menuliskan beberapa baris seperti dibawah ini.

```
oversample = SMOTE()
X, y = oversample.fit_resample(data_x, data_y)
```

Hasilnya, kini jumlah data antar kelas menjadi setara.



Preprocessing – Fitur Terpilih

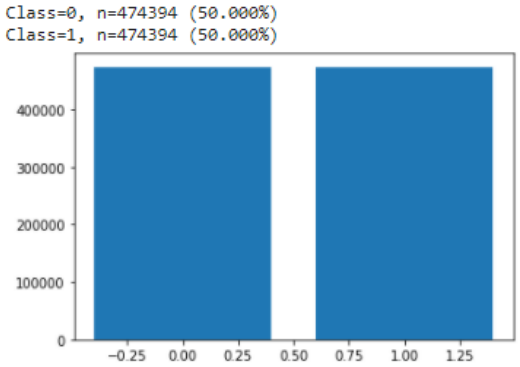
Preprocessing – Menghapus 3 kolom dengan korelasi terendah

Untuk membuat dua model model *decision tree* dan membandingkan dua hasil yang berbeda, maka diperlukan pembeda antar kedua model tersebut. Pembeda antar model yang diputuskan adalah perbedaan penggunaan kolom pada data. Untuk itu, akan dilakukan penghapusan 3 kolom dengan korelasi paling rendah terhadap kolom *fraud*. Ketiga kolom yang dihilangkan adalah kolom *grandTotal, quantityModifications*, dan *scannedLineItemsPerSecond.*

```
data_xs = x_tubes.loc[:, x_tubes.columns != 'fraud']
data_xs = data_xs.drop(['grandTotal', 'quantityModifications',
                        'scannedLineItemsPerSecond'], axis=1)
```

Preprocessing - Penanganan kelas tidak imbang menggunakan teknik SMOTE.

Dengan menggunakan teknik yang sama dalam menangani kelas yang tak imbang, teknik SMOTE, hasil yang sama terjadi pada data yang 3 fiturnya telah terlebih dahulu dihilangkan.



Klasifikasi

Setelah dilakukan preprocessing, maka selanjutnya adalah klasifikasi. Sebelum dilakukan klasifikasi, terlebih dahulu akan dicari parameter kedalaman pohon untuk mendapatkan hasil klasifikasi terbaik, pada kedua model yang menggunakan dataset hasil preprocessing yang berbeda. Untuk itu memudahkan proses pencarian parameter kedalamna terbaik, akan digunakan library *GridSearchCV*. *GridSearchCV* akan melakukan perulangan sejumlah parameter yang ditentukan dan melakukan penyesuaian parameter model pada data training. Akhirnya, parameter terbaik dari akan ditemukan, pada kasus ini adalah parameter kedalaman *decision tree*. Berikut ini merupakan parameter *depth* terbaik masing- masing model *decision tree* (19).

Klasifikasi - Decission Tree - All Features

```
[34] parameters = {'max_depth':range(3,20)}
     model = GridSearchCV(tree.DecisionTreeClassifier(),
                          parameters, n_jobs=-1)
     model.fit(X=X_all, y=y_all)
     tree_model = model.best_estimator_
     print (model.best_score_, model.best_params_)

     0.9934895933262047 {'max_depth': 19}
```

Klasifikasi - Decission Tree - Selected Features

```
[38] parameters = {'max_depth':range(3,20)}
     model = GridSearchCV(tree.DecisionTreeClassifier(),
                          parameters, n_jobs=-1)
     model.fit(X=X_sel, y=y_sel)
     tree_model = model.best_estimator_
     print (model.best_score_, model.best_params_)

     0.9912467335746118 {'max_depth': 19}
```

Klasifikasi dan Validasi Model D*ecission Tree* Menggunakan Metode *Stratified K-Fold*

Setelah parameter depth berhasil ditentukan, maka selanjutnya adalah klasifikasi menggunakan kedua data dengan model *decision tree.* Kedua model *decision tree* akan memiliki *depth* sebesar 19, sesuai dengan yang ditemukan pada proses sebelumnya. Setelah dilakukan klasifikasi, setiap model akan divalidasi menggunakan *stratified k-fold* validation. Alasan digunakannya *stratified k-fold* adalah metode ini akan memperhitungkan ratio antar kelas dan secara imbang membagi *datapoints train set* dan *test set* (setiap baris data (*datapoint*) akan berperan sebagai *test set* sebanyak satu kali dan tidak berulang).

```
TRAIN: [ 0  1  5  6  7  8  9 10 11 12 13 14] TEST: [ 2  3  4 15]
TRAIN: [ 0  2  3  4  5  7  8  9 11 12 13 15] TEST: [ 1  6 10 14]
TRAIN: [ 1  2  3  4  5  6  8  9 10 13 14 15] TEST: [ 0  7 11 12]
TRAIN: [ 0  1  2  3  4  6  7 10 11 12 14 15] TEST: [ 5  8  9 13]
```

Contoh, terdapat 16 baris (*datapoint*), layaknya gambar diatas. Datapoint 0 s.d 11 memiliki kelas yang berbeda dengan datapoint 12 s.d 15, maka perbandingan jumlah kelas menjadi 3:1. Dengan Stratified K-*Fold* pembagian jumlah kelas pada *train set* dan *test set* akan mengacu pada perbandingan 3:1. Keseimbangan komposisi kelas pada *train set* dan *test set* dapat menghindari misklasifikasi. Menilik kembali kedua data hasil preprocess, kedua data tersebut memiliki jumlah kelas yang seimbang, yaitu dengan ratio kelas *fraud* = 0 : *fraud* = 1 adalah 1:1. Dengan menggunakan *Stratified k-fold, test set* akan memiliki komposisi kelas (kolom *fraud*) sebesar 1:1, meskipun random, namun komposisi tetap seimbang. Selanjutnya, hasil validasi akan ditayangkan menggunakan visualisasi *confusion matrix.*

Selain hasil klasifikasi yang ditayangkan pada *confusion matrix*, akan terdapat pula parameter pengukuran lainnya seperti nilai akurasi, nilai *recall*, dan nilai *mean absolute error.*

Klasifikasi – Semua Kolom

Berikut ini adalah barisan kode dan hasil klasifikasi beserta validasinya menggunakan Stratified K-*Fold*, dengan menggunakan K = 5 (80% *train set*, 20% *test set*), pada model *decision tree* yang menggunakan dataset kolom yang utuh. Keterangan tambahan: misklasifikasi diwakili nilai error yang diukur menggunakan MAE.

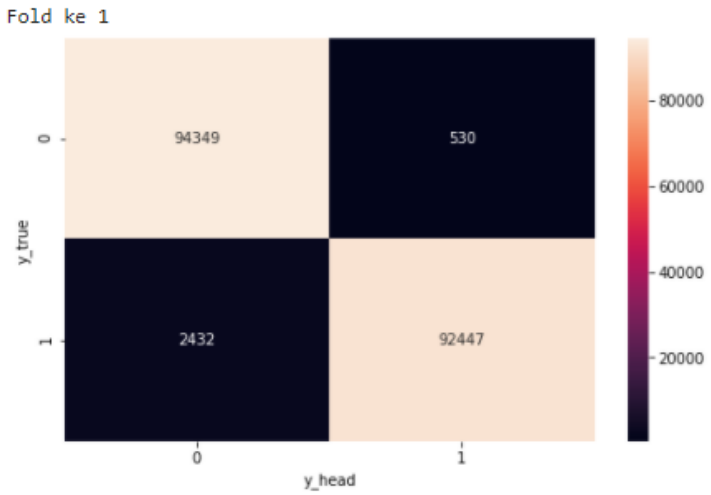
```
def get_score(model,X_train, X_test, y_train, y_test):
  model.fit(X_train, y_train)
  ypred = model.predict(X_test)
  ytest = y_test
  f, ax = plt.subplots(figsize=(8,5))
  sns.heatmap(confusion_matrix(ytest, ypred), annot=True, fmt=".0f", ax=ax)
  plt.xlabel("y_head")
  plt.ylabel("y_true")
  plt.show()
  print (classification_report(ytest, ypred))
  print ("Mean Absolute Error Fold ke-{} : {}".format(i,MAE(ytest, ypred)))
  return model.score(X_test, y_test)
```

```
folds = StratifiedKFold(n_splits = 5)
ytest = []
ypred = []
score = []
i = 1

for train_index, test_index in folds.split (X_all, y_all):
  X_train = X_all.iloc[train_index].loc[:]
  X_test = X_all.iloc[test_index].loc[:]
  y_train = y_all.iloc[train_index].loc[:]
  y_test = y_all.iloc[test_index].loc[:]
  print("Fold ke {}".format(i))
  score.append(get_score(tree.DecisionTreeClassifier(max_depth=19),
                         X_train, X_test, y_train, y_test))
  i = i + 1
```

Fold Ke -1

Fold ke 1

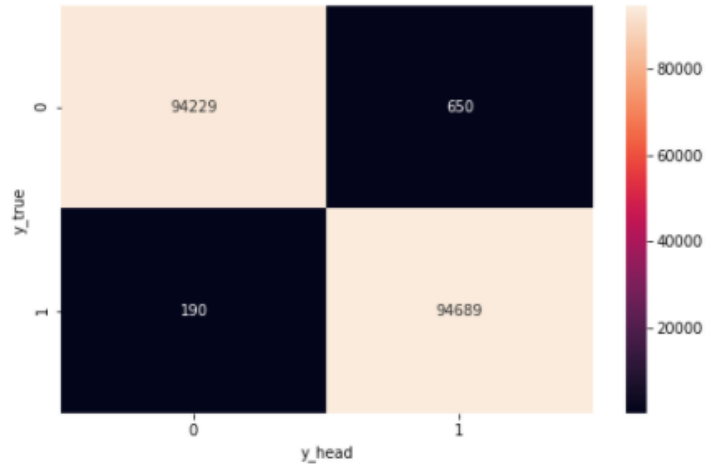


	precision	recall	f1-score	support
0	0.97	0.99	0.98	94879
1	0.99	0.97	0.98	94879
accuracy			0.98	189758
macro avg	0.98	0.98	0.98	189758
weighted avg	0.98	0.98	0.98	189758

Mean Absolute Error Fold ke-1 : 0.015609355073303893

Fold Ke -2

Fold ke 2

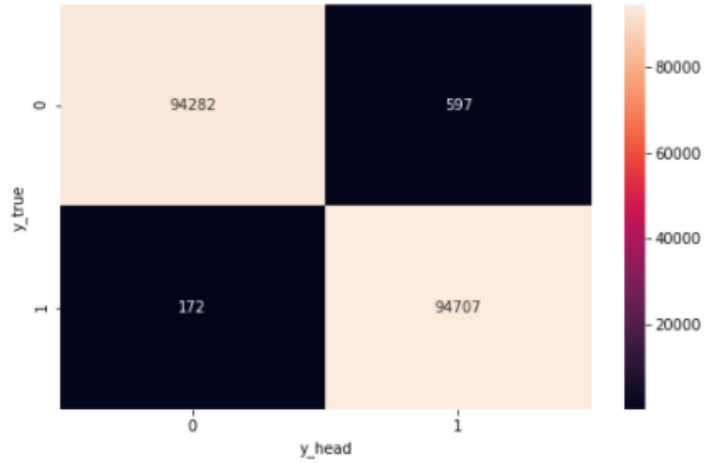


	precision	recall	f1-score	support
0	1.00	0.99	1.00	94879
1	0.99	1.00	1.00	94879
accuracy			1.00	189758
macro avg	1.00	1.00	1.00	189758
weighted avg	1.00	1.00	1.00	189758

Mean Absolute Error Fold ke-2 : 0.0044266908378039396

Fold Ke -3

Fold ke 3

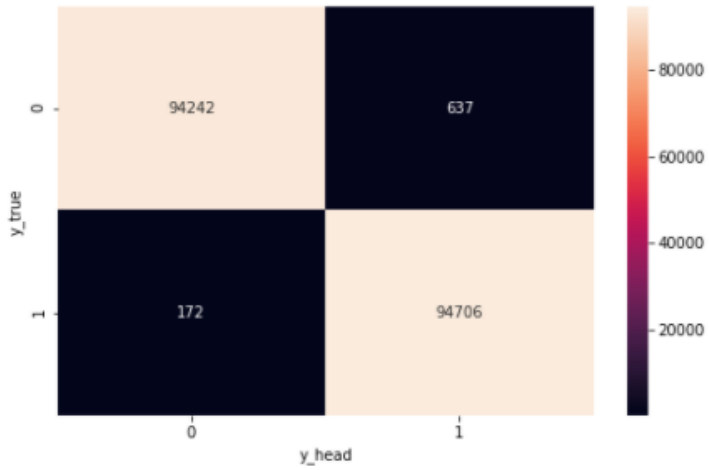


	precision	recall	f1-score	support
0	1.00	0.99	1.00	94879
1	0.99	1.00	1.00	94879
accuracy			1.00	189758
macro avg	1.00	1.00	1.00	189758
weighted avg	1.00	1.00	1.00	189758

Mean Absolute Error Fold ke-3 : 0.004052530064608607

Fold Ke -4

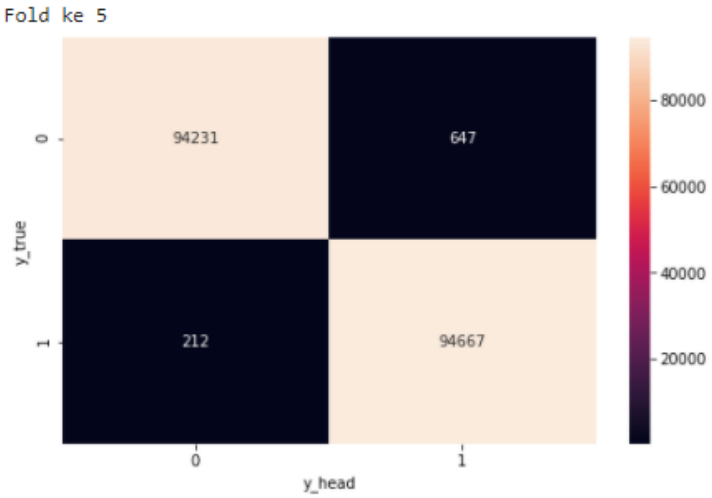
Fold ke 4



	precision	recall	f1-score	support
0	1.00	0.99	1.00	94879
1	0.99	1.00	1.00	94878
accuracy			1.00	189757
macro avg	1.00	1.00	1.00	189757
weighted avg	1.00	1.00	1.00	189757

Mean Absolute Error Fold ke-4 : 0.0042633473336952

Fold Ke -5



	precision	recall	f1-score	support
0	1.00	0.99	1.00	94878
1	0.99	1.00	1.00	94879
accuracy			1.00	189757
macro avg	1.00	1.00	1.00	189757
weighted avg	1.00	1.00	1.00	189757

Mean Absolute Error Fold ke-5 : 0.004526842224529266

Dari hasil validasi, didapatkan rataan akurasi sebesar 0.993424246893212 dan rataan MAE sebesar 0.006554673737694869, untuk model *decision tree* yang menggunakan data kolom yang utuh.

Klasifikasi – Kolom Terpilih

Dengan menggunakan langkah yang sama dengan bagian sebelumnya. Berikut ini adalah barisan kode dan hasil klasifikasi beserta validasinya menggunakan Stratified K-*Fold*, dengan menggunakan K = 5 (80% *train set*, 20% *test set*), pada model *decision tree* yang menggunakan dataset kolom terpilih.

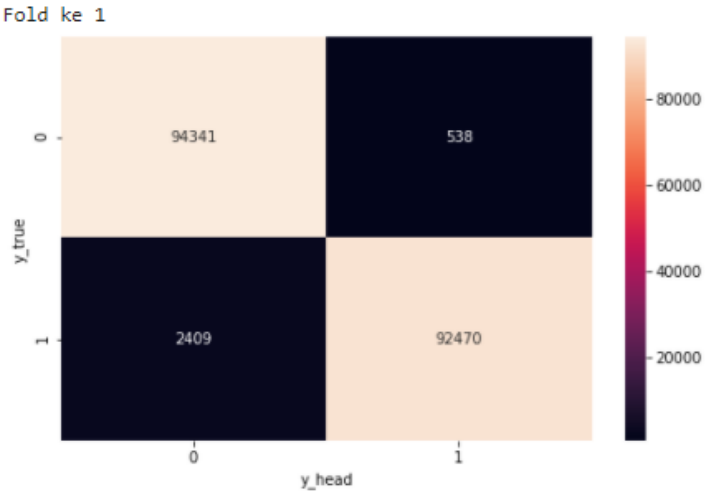
```
def get_score(model,X_train, X_test, y_train, y_test, mae_score):
  model.fit(X_train, y_train)
  ypred = model.predict(X_test)
  ytest = y_test
  f, ax = plt.subplots(figsize=(8,5))
  sns.heatmap(confusion_matrix(ytest, ypred), annot=True, fmt=".0f", ax=ax)
  plt.xlabel("y_head")
  plt.ylabel("y_true")
  plt.show()
  print (classification_report(ytest, ypred))
  print ("Mean Absolute Error Fold ke-{} : {}".format(i,MAE(ytest, ypred)))
  mae_score.append(MAE(ytest, ypred))
  return model.score(X_test, y_test)
```

```
folds = StratifiedKFold(n_splits = 5)
ytest = []
ypred = []
score = []
MAE_score = []
i = 1

for train_index, test_index in folds.split (X_all, y_all):
  X_train = X_all.iloc[train_index].loc[:]
  X_test = X_all.iloc[test_index].loc[:]
  y_train = y_all.iloc[train_index].loc[:]
  y_test = y_all.iloc[test_index].loc[:]
  print("Fold ke {}".format(i))
  score.append(get_score(tree.DecisionTreeClassifier(max_depth=19),
                         X_train, X_test, y_train, y_test, MAE_score))
  i = i + 1
```

Fold Ke – 1

Fold ke 1

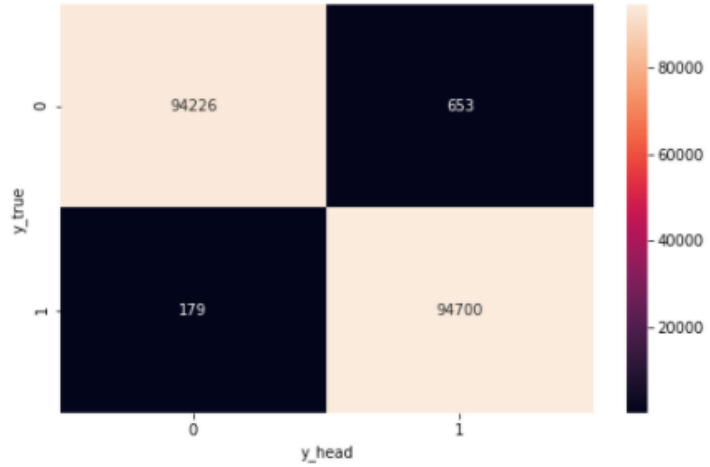


	precision	recall	f1-score	support
0	0.98	0.99	0.98	94879
1	0.99	0.97	0.98	94879
accuracy			0.98	189758
macro avg	0.98	0.98	0.98	189758
weighted avg	0.98	0.98	0.98	189758

Mean Absolute Error Fold ke-1 : 0.015530307022628823

Fold Ke – 2

Fold ke 2

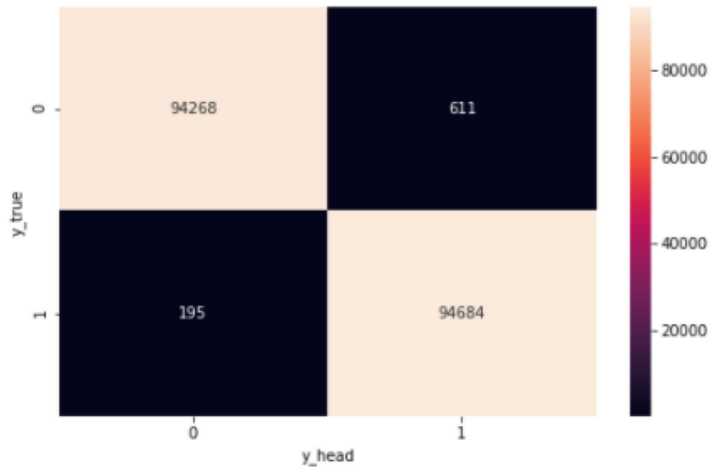


	precision	recall	f1-score	support
0	1.00	0.99	1.00	94879
1	0.99	1.00	1.00	94879
accuracy			1.00	189758
macro avg	1.00	1.00	1.00	189758
weighted avg	1.00	1.00	1.00	189758

Mean Absolute Error Fold ke-2 : 0.004384531877443902

Fold Ke – 3

```
Fold ke 3
```

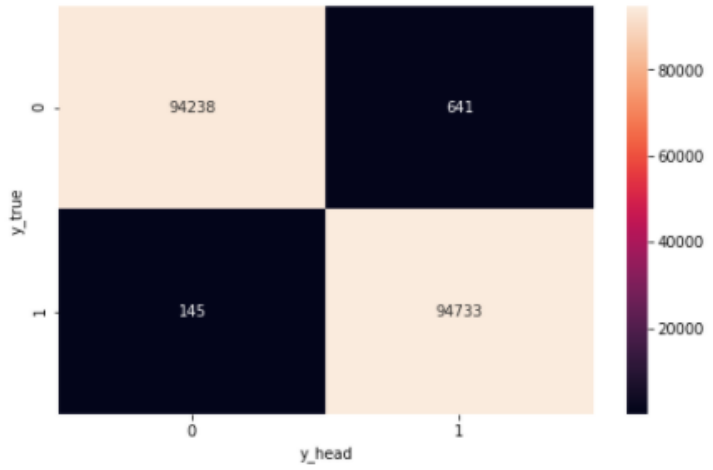


	precision	recall	f1-score	support
0	1.00	0.99	1.00	94879
1	0.99	1.00	1.00	94879
accuracy			1.00	189758
macro avg	1.00	1.00	1.00	189758
weighted avg	1.00	1.00	1.00	189758

```
Mean Absolute Error Fold ke-3 : 0.00424751525627378
```

Fold Ke – 4

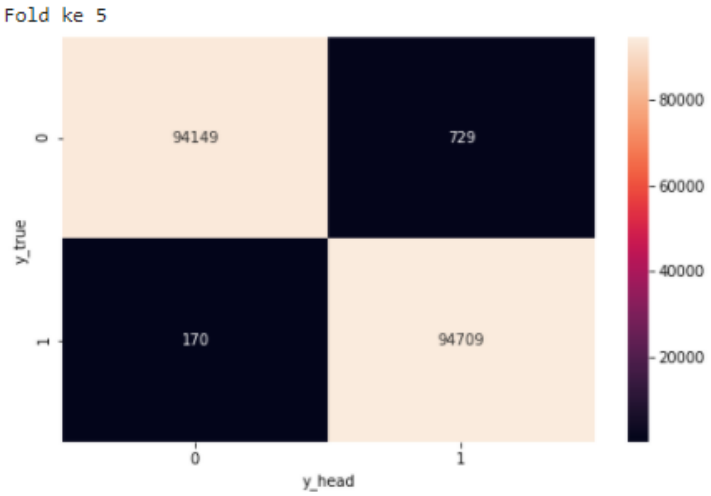
```
Fold ke 4
```



	precision	recall	f1-score	support
0	1.00	0.99	1.00	94879
1	0.99	1.00	1.00	94878
accuracy			1.00	189757
macro avg	1.00	1.00	1.00	189757
weighted avg	1.00	1.00	1.00	189757

```
Mean Absolute Error Fold ke-4 : 0.0041421396839115 29
```

Fold Ke - 5



	precision	recall	f1-score	support
0	1.00	0.99	1.00	94878
1	0.99	1.00	1.00	94879
accuracy			1.00	189757
macro avg	1.00	1.00	1.00	189757
weighted avg	1.00	1.00	1.00	189757

Mean Absolute Error Fold ke-5 : 0.0047376381371965195

Dari hasil validasi, didapatkan rataan akurasi sebesar 0.9934200308694262 dan rataan MAE sebesar 0.006579969130573899, untuk model *decision tree* yang menggunakan data kolom terpilih.

Kesimpulan - Klasifikasi

	Model DT – data utuh	Model DT – data terpilh
Akurasi	0.993424246893212	0.9934200308694262
Error (MAE)	0.006554673737694869	0.006579969130573899

Dari tabel di atas, dapat disipulkan bahwa kedua model decission tree memiliki performa klasifikasi yang baik, yakni memiliki akurasi yang tinggi dan error yang rendah (tidak underfitting maupun overfitting). Namun, model *decision tree* yang menggunakan data kolom utuh unggul, walaupun tidak jauh, dari segi akurasi dan memiliki error (misklasifikasi) yang lebih rendah, dibanding model *decision tree* yang menggunakan data terpilih. Sehingga, jawaban dari hipotesis, "**Dengan memangkas jumlah kolom yang memiliki korelasi rendah terhadap kolom target dapat meningkatkan performa**

(akurasi) model dan memperkecil kemungkinan misklasifikasi.", adalah hipotesis ini terbukti tidak benar dan tidak terbukti pada pelaksanaan kegiatan klasifikasi saat ini.

Telkom University

Klasifikasi Menggunakan *Model Decision Tree* dan Analisis Aturan Asosiasi

Aturan Asosiasi

Preprocessing

Data yang digunakan dalam proses analisis aturan asosiasi merupakan data yang memuat kolom utuh dan sudah memiliki kelas yang seimbang (dilakukan pada tahap preprocessing di bagian klasifikasi (sebelumnya)).

Preprocessing - Reduksi Jumlah Data

Dikarenakan data memiliki jumlah baris sebanyak dua kali jumlah dataset murni (belum dilakukan preprocessing) dan dikarenakan banyaknya data cukup membebani proses komputasi, sehingga memakan lebih banyak waktu, maka akan dilakukan reduksi data hingga 50%. Reduksi data yang dilakukan tetap memperhatikan keseimbangan antar kelas. Reduksi data dilakukan dengan cara mengambil masing-masing 50% sample dari setiap kelas. Lantas, data yang digunakan dalam proses analisis asosiasi memiliki sebanyak 474394 entri data (baris). Berikut ini adalah barisan kode yang digunakan untuk mereduksi data.

```
count = len(aturan)/4
data1 = aturan[aturan['fraud']==0].sample(int(count))
data2 = aturan[aturan['fraud']==1].sample(int(count))
aturan = data1.append(data2, ignore_index = True)
aturan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 474394 entries, 0 to 474393
Data columns (total 10 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   trustLevel                 474394 non-null  float64
 1   totalScanTimeInSeconds     474394 non-null  float64
 2   grandTotal                 474394 non-null  float64
 3   lineItemVoids              474394 non-null  float64
 4   scansWithoutRegistration   474394 non-null  float64
 5   quantityModifications      474394 non-null  float64
 6   scannedLineItemsPerSecond  474394 non-null  float64
 7   valuePerSecond             474394 non-null  float64
 8   lineItemVoidsPerPosition   474394 non-null  float64
 9   fraud                      474394 non-null  int64
dtypes: float64(9), int64(1)
```

Preprocessing – Kategorisasi Data pada Kolom Spesifik

Untuk memperlancar proses analisis aturan asosiasi, maka ditetapkan kategori dengan syarat berjangka untuk masing-masing kolom desimal yang akan dikonversi datanya menjadi data kategorikal. Kategori yang akan ditetapkan pada seluruh kolom desimal adalah sebagai berikut.

$$kategori = \begin{cases} lo, & min \leq x < Q1 \\ med, & Q1 < x \leq Q3 \\ hi, & Q3 < x \leq max \end{cases}$$

Terlihat pada aturan diatas, bahwa terdapat parameter *min* (nilai minimum), Q1 (kuartil 1), Q3 (kuartil 3), dan *max* (nilai maksimal), yang mana merupakan nilai statistik yang didapat dari masing-masing kolom desimal yang hendak dikonversi menjadi kolom kategorikal. Kemudian aturan tersebut deterapkan terhadap seluruh kolom desimal, terkecuali kolom *trustLevel.* Nampak implementasi dan hasil kategorisasi data kolom desimal, dalam barisan kode berikut ini.

```
col = aturan.columns
skip_col = ['trustLevel', 'fraud']
for i in range (len(col)):
  if col[i] not in skip_col:
    temp = aturan[col[i]].describe()
    min, Q1, Q3, max = temp['min']-1,temp['25%'],temp['75%'],temp['max']+1
    aturan[col[i]] = pd.cut(aturan[col[i]], bins=[min,Q1,Q3,max], labels=['lo','med','hi'])
aturan.head(10)
```

	trustLevel	totalScanTimeInSeconds	grandTotal	lineItemVoids	scansWithoutRegistration	quantityModifications	scannedLineItemsPerSecond	valuePerSecond	lineItemVoidsPerPosition
0	1.0	med	lo	med	hi	med	med	lo	hi
1	6.0	med	med	hi	med	med	med	hi	hi
2	2.0	med	med	hi	lo	med	med	hi	hi
3	4.0	lo	med	lo	med	lo	hi	hi	med
4	4.0	hi	med	med	hi	lo	lo	med	hi
5	1.0	med	med	med	hi	med	lo	hi	med
6	5.0	lo	hi	med	med	med	med	hi	hi
7	6.0	med	med	med	lo	med	med	med	med
8	5.0	med	lo	lo	lo	med	lo	med	lo
9	3.0	med	lo	med	med	med	lo	lo	lo

tLevel	totalScanTimeInSeconds	grandTotal	lineItemVoids	scansWithoutRegistration	quantityModifications	scannedLineItemsPerSecond	valuePerSecond	lineItemVoidsPerPosition	fraud
1.0	med	lo	med	hi	med	med	lo	hi	0
6.0	med	med	hi	med	med	med	hi	hi	0
2.0	med	med	hi	lo	med	med	hi	hi	0
4.0	lo	med	lo	med	lo	hi	hi	med	0
4.0	hi	med	med	hi	lo	lo	med	hi	0
1.0	med	med	med	hi	med	lo	hi	med	0
5.0	lo	hi	med	med	med	med	hi	hi	0
6.0	med	med	med	lo	med	med	med	med	0
5.0	med	lo	lo	lo	med	lo	med	lo	0
3.0	med	lo	med	med	med	lo	lo	lo	0

Preprocessing – One Hot Encoding

Setelah proses kategorisasi data desimal selesai, selanjutnya adalah mengubah seluruh kategori data menjadi satuan kolom menggunakan metode *one hot encoding.* Hal ini dilakukan untuk mempermudah pembacaan *frequent itemset* dan aturan asosiasi yang akan dihasilkan di tahap selanjutnya. Proses *one hot encoding* akan dilakukan terhadap keseluruhan kolom, dan berikut adalah barisan kode implementasinya.

```
enc = OneHotEncoder()
kategori = ['trustLevel','totalScanTimeInSeconds','grandTotal','lineItemVoids',
            'scansWithoutRegistration','quantityModifications','scannedLineItemsPerSecond',
            'valuePerSecond','lineItemVoidsPerPosition','fraud']
enc.fit(aturan[kategori])
onehot = pd.DataFrame(data=enc.transform(aturan[kategori]).toarray(),
                      columns=enc.get_feature_names(kategori))
aturan = onehot
```

Aturan Asosiasi

Setelah dataset telah dibersihkan, maka prose selanjutnya adalah analisis aturan asosiasi dengan tahapan pembentukan *frequent itemset* dan pembentukan aturan asosiasi.

Aturan Asosiasi – *Frequent itemset*

Frequent itemset generation dilakuka dengan menggunakan algoritma apriori. Untuk menemukan daftar transaksi yang tingkat kemunculannya sesuai dengan yang diharapkan, maka perlu ditetapkan nilai *minimum support*. Pada kesempatan kali ini, akan ditentukan nilai *minimum support* sebesar 10%, yang artinya item yang akan muncul pada *frequent itemset* memiliki tingkat kemunculan minimal 10% dari segenap transaksi yang ada. *Minimum support* sebesar 10% setara dengan minimum support sebesar 20% pada masing-masing kelas (fraud = 0 & *fraud* = 1) yang tersedia pada dataset ini. Misal dari total 10 transaksi terdapat *item* "Z" yang muncul pada 2 transaksi, artinya *item* "Z" memiliki nilai support 20%, yang mana kemunculannya lebih besar dari nilai minimum support (10%), sehingga *item* "Z" akan masuk dalam daftar frequent itemset. Berikut ini merupakan kode dan hasil 499 *frequent itemset* yang dihasilkan menggunakan algoritma apriori.

```
# creating Frequent Item Set
frq_items = apriori(aturan, min_support = 0.1, use_colnames = True)
frq_items['lenght'] = frq_items['itemsets'].apply(lambda x: len(x))
print(len(frq_items))
frq_items.head(10)
```

```
499
```

	support	itemsets	lenght
0	0.529581	(trustLevel_1)	1
1	0.120147	(trustLevel_2)	1
2	0.250001	(totalScanTimeInSeconds_hi)	1
3	0.250001	(totalScanTimeInSeconds_lo)	1
4	0.499998	(totalScanTimeInSeconds_med)	1
5	0.250001	(grandTotal_hi)	1
6	0.250016	(grandTotal_lo)	1
7	0.499983	(grandTotal_med)	1
8	0.214377	(lineItemVoids_hi)	1
9	0.254898	(lineItemVoids_lo)	1

	support	itemsets	lenght
489	0.128252	(trustLevel_1, lineItemVoids_med, fraud_1, qua...	5
490	0.110885	(trustLevel_1, lineItemVoids_med, valuePerSeco...	5
491	0.145990	(trustLevel_1, lineItemVoids_med, fraud_1, sca...	5
492	0.122811	(trustLevel_1, lineItemVoids_med, valuePerSeco...	5
493	0.100043	(trustLevel_1, fraud_1, scansWithoutRegistrati...	5
494	0.114854	(trustLevel_1, valuePerSecond_med, fraud_1, sc...	5
495	0.116313	(trustLevel_1, fraud_1, scannedLineItemsPerSec...	5
496	0.131321	(trustLevel_1, valuePerSecond_med, fraud_1, sc...	5
497	0.100632	(totalScanTimeInSeconds_med, lineItemVoids_med...	5
498	0.105773	(lineItemVoids_med, valuePerSecond_med, fraud_...	5

Aturan Asosiasi – *Rules generation*

Setelah ditemukan sebanyak 499 *frequent itemset*, lantas selanjutnya adalah proses pembentukan rule. Selama proses pembentukan rule berlangsung, akan pula dihitung nilai confident masing-masing rule yang terbentuk. Nilai confident berperan penting dalam pembentukan strong rule, yang mana semakin besar nilai confident, maka rule yang dihasilkan akan semakin kuat (baik). Kemudian, selain menggunakan nilai confident untuk menentukan stong rule, dilakukan pula perhitungan parameter strong rule lainnya, yaitu perhitungan nilai lift. Nilai lift yang lebih besar dari 1.0 menyatakan bahwa hubungan antara anteseden dan konsekuen lebih signifikan (kuat) daripada yang diharapkan. Semakin besar nilai lift, semakin signifikan hubungannya. Lantas, berikut ini merupakan barisan kode, 10 rule transaksi *fraudulent,* dan 10 rule transaksi *non-fraudulent.*

Fraudulent Transaction

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(scannedLineItemsPerSecond_med, trustLevel_1, ...	(fraud_1)	0.108815	0.5	0.108652	0.998508	1.997017	0.054245	335.201299
2	(scannedLineItemsPerSecond_med, lineItemVoids_...	(fraud_1)	0.148758	0.5	0.145990	0.981394	1.962789	0.071611	26.873572
3	(scannedLineItemsPerSecond_med, valuePerSecond...	(fraud_1)	0.134009	0.5	0.131321	0.979944	1.959889	0.064317	24.930588
4	(scansWithoutRegistration_med, scannedLineItem...	(fraud_1)	0.102480	0.5	0.100043	0.976222	1.952444	0.048803	21.027682
5	(scannedLineItemsPerSecond_med, trustLevel_1, ...	(fraud_1)	0.119243	0.5	0.116313	0.975428	1.950856	0.056691	20.348201
6	(scannedLineItemsPerSecond_med, grandTotal_med...	(fraud_1)	0.105419	0.5	0.102691	0.974125	1.948250	0.049982	19.323802
7	(lineItemVoids_med, valuePerSecond_med, trustL...	(fraud_1)	0.126184	0.5	0.122811	0.973271	1.946543	0.059719	18.706563
8	(scannedLineItemsPerSecond_med, trustLevel_1, ...	(fraud_1)	0.207393	0.5	0.201628	0.972201	1.944403	0.097931	17.986472
9	(scannedLineItemsPerSecond_med, lineItemVoids_...	(fraud_1)	0.114386	0.5	0.110885	0.969390	1.938781	0.053692	16.334738
10	(scannedLineItemsPerSecond_med, valuePerSecond...	(fraud_1)	0.118612	0.5	0.114854	0.968313	1.936626	0.055548	15.779305

Non-fraudulent Transaction

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1	(scannedLineItemsPerSecond_lo, totalScanTimeIn...	(fraud_0)	0.121490	0.500000	0.120176	0.989190	1.978381	0.059432	46.255217
52	(lineItemVoids_med, lineItemVoidsPerPosition_hi)	(fraud_0)	0.112326	0.500000	0.104164	0.927337	1.854674	0.048001	6.881069
89	(scannedLineItemsPerSecond_lo, lineItemVoidsPe...	(fraud_0)	0.113444	0.500000	0.103075	0.908598	1.817195	0.046353	5.470319
127	(scannedLineItemsPerSecond_hi, fraud_0)	(totalScanTimeInSeconds_lo)	0.122603	0.250176	0.108631	0.886042	3.541676	0.077959	6.579853
140	(scannedLineItemsPerSecond_lo)	(fraud_0)	0.250001	0.500000	0.215410	0.861635	1.723269	0.090409	3.613620
146	(scannedLineItemsPerSecond_lo, valuePerSecond_...	(fraud_0)	0.136667	0.500000	0.117029	0.856310	1.712620	0.048696	3.479712
153	(scannedLineItemsPerSecond_lo, lineItemVoids_med)	(fraud_0)	0.126182	0.500000	0.107577	0.852556	1.705112	0.044486	3.391117
160	(scannedLineItemsPerSecond_lo, grandTotal_med)	(fraud_0)	0.121863	0.500000	0.103536	0.849613	1.699227	0.042605	3.324764
164	(scannedLineItemsPerSecond_lo, quantityModific...	(fraud_0)	0.127095	0.500000	0.107554	0.846251	1.692502	0.044007	3.252050
262	(fraud_0, valuePerSecond_hi)	(totalScanTimeInSeconds_lo)	0.160354	0.250176	0.121677	0.758804	3.033082	0.081561	3.108778

Kesimpulan – Aturan Asosiasi

Jika diambil 3 aturan asosiasi dengan nilai *confident* dan *lift* tertinggi, sebagai jawaban untuk pertanyaan, **"Rangkaian Item transaksi yang seperti apa yang menyababkan fraudulent dan non-faudulent?",** maka 3 aturan transaksi terkategorisasi *fraudulent*, dengan nilai confident dan lift terbesar, adalah sebagai berikut.

```
0. frozenset({'scannedLineItemsPerSecond_med', 'trustLevel_1', 'totalScanTimeInSeconds_hi'}) -> frozenset({'fraud_1'})
2. frozenset({'scannedLineItemsPerSecond_med', 'lineItemVoids_med', 'trustLevel_1', 'lineItemVoidsPerPosition_med'}) -> frozenset({'fraud_1'})
3. frozenset({'scannedLineItemsPerSecond_med', 'valuePerSecond_med', 'trustLevel_1', 'lineItemVoidsPerPosition_med'}) -> frozenset({'fraud_1'})
```

Sementara, 3 aturan transaksi *non-fraudulent*, dengan nilai confident dan lift terbesar, nampak sebagai berikut.

```
1. frozenset({'scannedLineItemsPerSecond_lo', 'totalScanTimeInSeconds_med'}) -> frozenset({'fraud_0'})
52. frozenset({'lineItemVoids_med', 'lineItemVoidsPerPosition_hi'}) -> frozenset({'fraud_0'})
89. frozenset({'scannedLineItemsPerSecond_lo', 'lineItemVoidsPerPosition_hi'}) -> frozenset({'fraud_0'})
```

Lampiran

Tautan slide presentasi

https://docs.google.com/presentation/d/1lMlaAxyG-uUdQRQAzphOAnPNZu2NlUwiDQO0UCobSJg/edit?usp=sharing

Tautan video penjelasan

https://youtu.be/zdYvv1w27BQ

Tautan *Google Colab*

https://colab.research.google.com/drive/16uco6l2g00rEyH7AMczQvWBNruOM86_b?usp=sharing