

**ANALISA MEMPREDIKSI KLASIFIKASI MACHINE LEARNING
UNTUK PELANGGAN TERTARIK MEMBELI KENDARAAN BARU
ATAU TIDAK**

LAPORAN TUGAS BESAR MACHINE LEARNING

Oleh:

DEWI LAYLATURROHMAH

NIM. 2041720163

DHERISMA HANINDITA UTAMI

NIM. 2041720018

MAULANA BINTANG IRFANSYAH

NIM. 2041720132



**PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG**

2022

DAFTAR ISI

DAFTAR ISI.....	ii
ABSTRAK	1
BAB 1 PENDAHULUAN	2
1. Latar Belakang	2
BAB 2 METODE.....	3
1. Formulasi Masalah.....	3
2. Eksplorasi dan Persiapan data.....	5
3. Eksperimen	9
4. Pemodelan.....	12
5. Evaluasi Model	18
BAB 3 KESIMPULAN	24

ABSTRAK

Kecerdasan buatan digunakan untuk menemukan solusi untuk berbagai masalah yang mencakup dalam pembelajaran mesin. Artikel ini menyajikan ulasan pemecahan masalah dari penelitian-penelitian terkini dengan mengklasifikasikan machine learning menjadi tiga kategori : pembelajaran terarah, pembelajaran tidak terarah, dan pembelajaran reinforcement. Hasil ulasan menunjukkan ketiga kategori masih berpeluang digunakan dalam beberapa kasus terkini dan dapat ditingkatkan untuk mengurangi beban komputasi dan mempercepat kinerja untuk mendapatkan tingkat akurasi dan presisi yang tinggi. Tujuan ulasan artikel ini diharapkan dapat menemukan celah dan dijadikan pedoman untuk penelitian pada masa yang akan datang.

Perkembangan terkait ilmu pengetahuan dan teknologi terus meningkat. Kemajuan terkait teknologi membuat manusia untuk menikmati kemudahan. Tetapi dalam segala kemudahan tersebut, mendeteksi prediksi pelanggan dalam menentukan tertarik pada kendaraan baru atau tidak. Jika kami memiliki informasi sebelumnya yang sangat mendekati perkiraan tentang hal diatas dan banyak lagi lebih banyak situasi kehidupan sehari hari yang dapat membuat keputusan yang tepat, ini membantu dalam masa depan kendaraan. Dalam beberapa decade saat ini, data telah bergerak menuju konsep big data untuk transportasi. Dalam pekerjaan ini, kami berencana menggunakan mesin pembelajaran, genetika, komputasi lunak, dan algoritma pembelajaran mendalam untuk menganalisis data besar untuk sistem transportasi dengan kompleksitas yang jauh berkurang, yang pada akhirnya membantu untuk pelatihan yang tepat dari ketertarikan pengendara.

BAB 1

PENDAHULUAN

1. Latar Belakang

Transportasi merupakan suatu sarana yang tidak dapat dipisahkan dalam kehidupan sehari-hari. Sepeda motor merupakan salah satu sarana transportasi yang mayoritas digunakan oleh masyarakat di Indonesia karena sangat efisien digunakan dalam aktifitas sehari-hari [1] (<https://www.fortuna-motor.co.id/sepeda-motor/>). Pada saat ini tingkat penggunaan kendaraan bermotor dari tahun ketahun semakin meningkat, hal tersebut dilihat dari kepadatan arus lalu lintas yang terjadi. Permasalahan yang terjadi disebabkan karena mudahnya masyarakat umum untuk membeli kendaraan bermotor secara kredit ataupun bayar tunai. Pertumbuhan pengguna kendaraan yang semakin meningkat berdasarkan data yang dikeluarkan Badan Pusat Statistik (BPS) menginformasikan bahwa pertumbuhan kendaraan dari beberapa tahun jumlahnya bertambah. Pertambahan tersebut tidak dipungkiri dari terdapatnya berapa pelaku usaha yang bergerak dibidang penjualan kendaraan bermotor yang menawarkan berbagai jenis merk kendaraan dengan berbagai teknik promosi untuk menggaet dan menarik konsumen atau pembeli yang memerlukannya. Metode ini akan memprediksi apakah pelanggan tertarik untuk membeli kendaraan baru atau tidak berdasarkan data pelanggan di dealer. Percobaan ini menggunakan eksperimen model klasifikasi label pada dataset. Percobaan dimulai dari membuat model klasifikasi yang dibuat akan digunakan untuk memprediksi label dari dataset testing yang telah disediakan, dan pada akhirnya antara hasil prediksi dan label sebenarnya dibandingkan untuk mengatur performansi dari model.

BAB 2

METODE

1. Formulasi Masalah

Permasalahan yang akan diselesaikan pada tugas tahap kedua classification dengan metode supervised learning adalah memprediksi apakah pelanggan tertarik untuk membeli kendaraan baru atau tidak berdasarkan data pelanggan di dealer.

A. Deskripsi fitur pada dataset

Penjelasan dari fitur-fitur dataset :

1. Id : identitas pelanggan yang terdaftar
2. Jenis_kelamin : jenis kelamin
3. Umur : umur
4. Sim : kepemilikan sim
5. Kode daerah : kode daerah tempat tinggal pelanggan
6. Sudah asuransi : sudah pernah asuransi / belum
7. Umur kendaraan : umur kendaraan
8. Kendaraan rusak : mobil pelanggan pernah rusak atau belum
9. Premi : jumlah premi yang harus dibayarkan
10. Kanal penjualan : kode kanal untuk menghubungi pelanggan (email, telpon, dll)
11. Lama berlangganan : durasi pelanggan menjadi klien perusahaan
12. Tertarik : tertarik atau tidak

B. Deskripsi Model

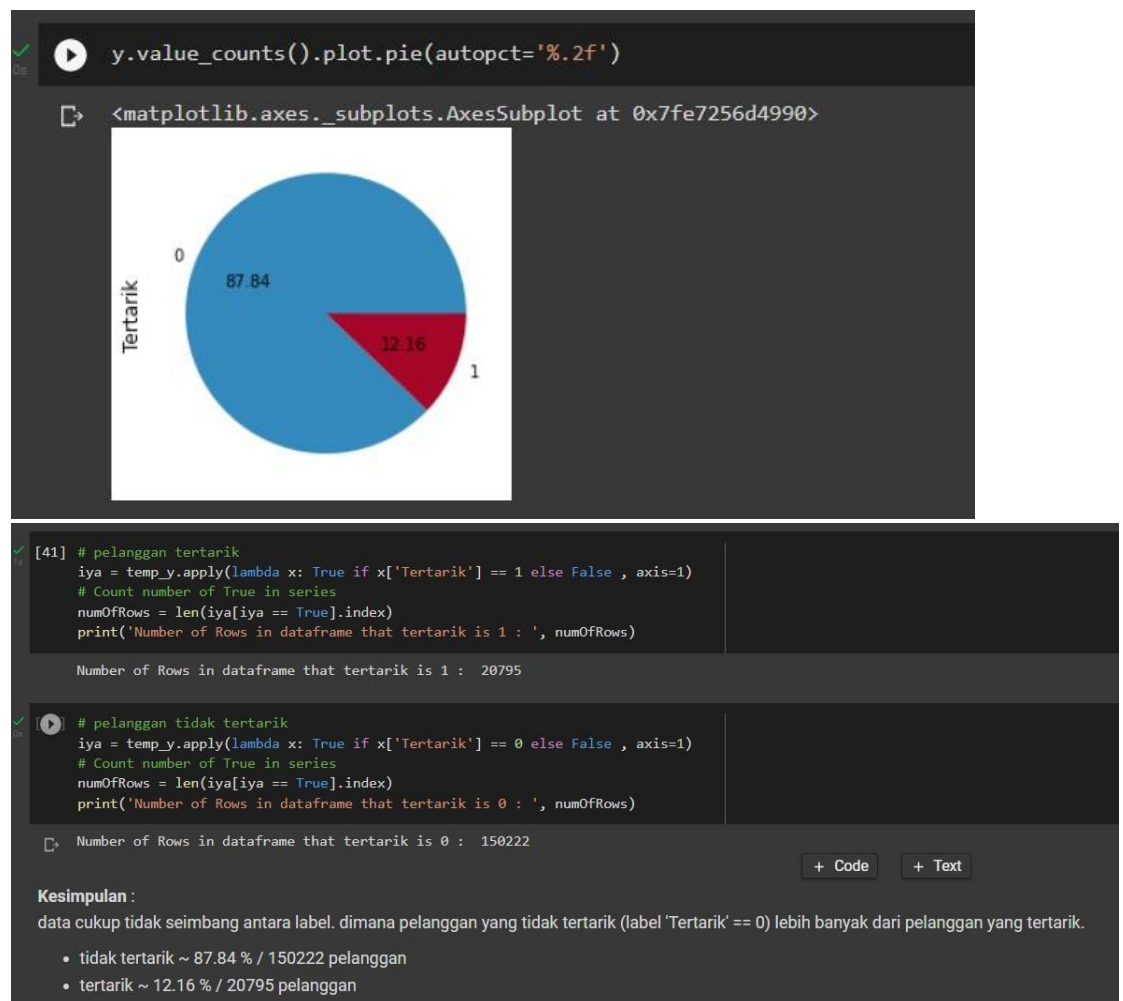
Diberikan 2 file excel berisi data historis pelanggan dealer mobil yang tertarik / membeli dan tidak tertarik / tidak jadi beli mobil. data terdiri dari berbagai macam fitur.

File Name	File Size	File Type	File Date
kendaraan_test	2.256 KB	Microsoft Excel Co...	03/11/2021 23:28
kendaraan_train	14.884 KB	Microsoft Excel Co...	03/11/2021 23:25
kendaraan_train	13.977 KB	Microsoft Excel W...	03/11/2021 23:26

Dari dataset tersebut, akan dilakukan percobaan eksperimen model klasifikasi label pada dataset. percobaannya dimulai dari membuat model klasifikasi berdasarkan dataset pelatihan yang disediakan, kemudian dengan model klasifikasi yang dibuat akan digunakan untuk memprediksi label dari dataset testing yang telah disediakan. Dan pada akhirnya, antara hasil prediksi dan label sebenarnya dibandingkan untuk mengukur performansi dari model.

dalam percobaan membuat model klasifikasi, dengan tujuan membandingkan, kami membuat 3 model klasifikasi antara lain :

- Random Forest
 - Naive Bayes
 - Logistic Regression
- dari ketiga model diatas cukup populer digunakan dalam kasus klasifikasi, khususnya ketika variabel independen y (label) merupakan kategorial.



Setelah dilakukan analisis terhadap label data training, dapat disimpulkan bahwa dataset merupakan imbalanced dataset (data tidak setara). Dimana jika di buat visualisasinya, label '0' / tidak tertarik berjumlah lebih banyak dari label '1'/tertarik, sebesar 87% dan 12%.

Sehingga dari kasus diatas, kami akan membuat 2 eksperimen pada dataset, melakukan overSampling dan underSampling.

C. Deskripsi Eksperimen

Dalam percobaan klasifikasi ini, kami melakukan 2 eksperimen pada tahap preprocessing data sebelum dimasukkan ke tahap pembuatan model, Eksperimen 1 :

- Mengelola nilai yang kosong (missing values) : dilakukan pembuangan pada baris tersebut.
- Label data training yang tidak setara (imbalanced dataset) : dilakukan oversampling supaya data yang setara.

Setelah mendapatkan prediksi kami melakukan evaluasi performansi dari model dengan menghitung nilai akurasi model, nilai precision model, nilai recall model, nilai f-1 model dan membuat confusion matrix untuk melihat jumlah prediksi yang bersifat True positive, True negative, False positive, False negative. Dari masing-masing nilai evaluasi tersebut akan dilakukan perbandingan antara masing-masing model untuk melihat model yang memiliki performansi terbaik pada kasus ini.

2. Eksplorasi dan Persiapan data

Eksplorasi dan persiapan data yang saya lakukan pada tugas tahap kedua classification dengan metode supervised learning adalah sebagai berikut.

1. Mendrop kolom “id” karena kami mengira bahwa kolom tersebut tidak terlalu berpengaruh pada tugas classification ini.

```
1 dataTrain = dataTrain.drop(['id'], axis = 1)
```

1 dataTrain.head(10)											
	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertarik
0	Wanita	30.0	1.0	33.0	1.0	< 1 Tahun	Tidak	28029.0	152.0	97.0	0
1	Pria	48.0	1.0	39.0	0.0	> 2 Tahun	Pernah	25800.0	29.0	158.0	0
2	NaN	21.0	1.0	46.0	1.0	< 1 Tahun	Tidak	32733.0	160.0	119.0	0
3	Wanita	58.0	1.0	48.0	0.0	1-2 Tahun	Tidak	2630.0	124.0	63.0	0
4	Pria	50.0	1.0	35.0	0.0	> 2 Tahun	NaN	34857.0	88.0	194.0	0
5	Pria	21.0	1.0	35.0	1.0	< 1 Tahun	Tidak	22735.0	152.0	171.0	0
6	Wanita	33.0	1.0	8.0	0.0	NaN	Pernah	32435.0	124.0	215.0	1
7	Pria	23.0	NaN	28.0	1.0	< 1 Tahun	Tidak	26869.0	152.0	222.0	0
8	Wanita	20.0	1.0	8.0	1.0	< 1 Tahun	Tidak	30786.0	160.0	31.0	0
9	NaN	54.0	1.0	29.0	0.0	> 2 Tahun	Pernah	88883.0	124.0	28.0	1

2. Menghapus kolom yang memiliki nilai duplikat.

```
drop duplicate column

[ ] 1 duplicate = list(dataTrain.duplicated())
    2 print("Data Duplikasi :", duplicate.count(True))

Data Duplikasi : 169

[ ] 1 dataTrain.drop_duplicates(inplace=True)

[ ] 1 duplicate = list(dataTrain.duplicated())
    2 print("Data Duplikasi :", duplicate.count(True))

Data Duplikasi : 0
```

3. Mengubah nilai kategorial menjadi nilai numerik agar dapat mudah untuk dilakukannya pengolahan data. untuk tahap ini, kami menggunakan fungsi preprocessing dari library sklearn.

```
2 from sklearn import preprocessing

1 dataTrain.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 171017 entries, 0 to 171016
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   Jenis_Kelamin          171017 non-null object  
1   Umur                   171017 non-null float64  
2   SIM                    171017 non-null float64  
3   Kode_Daerah            171017 non-null float64  
4   Sudah_Asuransi         171017 non-null float64  
5   Umur_Kendaraan         171017 non-null object  
6   Kendaraan_Rusak        171017 non-null object  
7   Premi                  171017 non-null float64  
8   Kanal_Penjualan        171017 non-null float64  
9   Lama_Berlangganan      171017 non-null float64  
10  Tertarik               171017 non-null int64  
dtypes: float64(7), int64(1), object(3)
memory usage: 14.4+ MB

1 # melihat berapa banyak total kategori pada jenis_kelamin, umur_kendaraan, kendaraan_rusak
2 cols = ['Jenis_Kelamin', 'Umur_Kendaraan', 'Kendaraan_Rusak']
3 dataTrain[cols].nunique()

Jenis_Kelamin      2
Umur_Kendaraan     3
Kendaraan_Rusak    2
dtype: int64
```

Berikut hasilnya :


```

1 le = preprocessing.LabelEncoder()
2 dataTrain[cols] = dataTrain[cols].apply(le.fit_transform)
3 ###
4 # jenis kelamin : wanita/1 || pria/0
5 # umur kendaraan : < 1 tahun/1 || 1-2 tahun/0 || >2 tahun/2
6 # kendaraan rusak : tidak/1 || pernah/0
7 ###

```

1 dataTrain.head(10)

	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertarik
0	1	30.0	1.0	33.0	1.0	1	1	28029.0	152.0	97.0	0
1	0	48.0	1.0	39.0	0.0	2	0	25800.0	29.0	158.0	0
2	1	58.0	1.0	48.0	0.0	0	1	2630.0	124.0	63.0	0
3	0	21.0	1.0	35.0	1.0	1	1	22735.0	152.0	171.0	0
4	1	20.0	1.0	8.0	1.0	1	1	30786.0	160.0	31.0	0
5	0	25.0	1.0	14.0	1.0	1	1	34212.0	152.0	282.0	0
6	0	66.0	1.0	24.0	1.0	0	1	38616.0	145.0	281.0	0
7	0	31.0	1.0	8.0	0.0	1	0	2630.0	152.0	132.0	0
8	1	24.0	1.0	30.0	1.0	1	1	27285.0	152.0	215.0	0
9	1	22.0	1.0	15.0	0.0	1	0	38289.0	152.0	225.0	0

- Melakukan normalisasi pada dataset. Pada normalisasi kami menggunakan metode normalisasi Min-Max scaler yang merubah semua range nilai menjadi range nilai dari 0 - 1.

```

6 from sklearn.preprocessing import MinMaxScaler

1 cols = ['Umur','Kode_Daerah','Premi','Kanal_Penjualan','Lama_Berlangganan']
2
3 # dataTrain = normalise_min_max(dataTrain[cols])
4
5 scaler = MinMaxScaler()
6
7 dataTrain[cols] = scaler.fit_transform(dataTrain[cols].values)
8 dataTrain.sample(10)

```

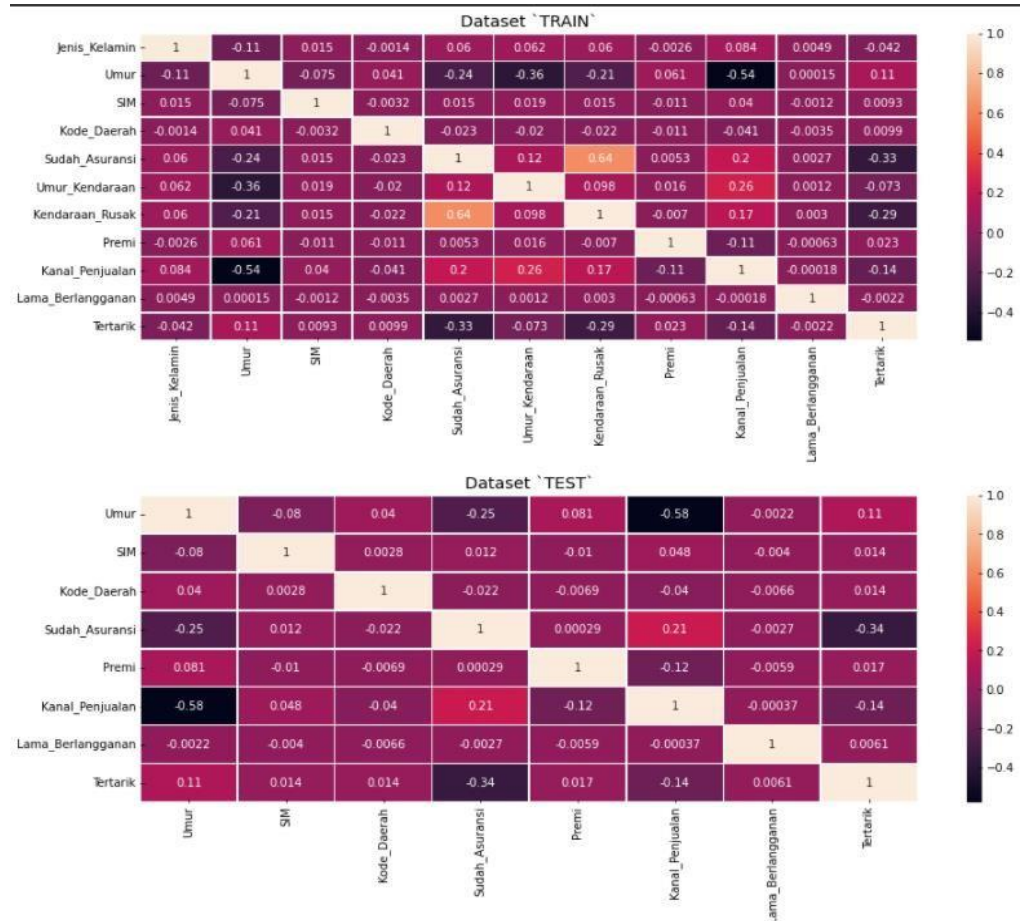
	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertarik
73258	0	0.446154	1.0	0.538462	0.0	0	0	0.051937	0.759259	0.359862	0
221328	1	0.092308	1.0	0.788462	1.0	1	1	0.043298	0.932099	0.397924	0
91319	0	0.584615	1.0	0.538462	0.0	0	0	0.000000	0.759259	0.089965	0
167489	1	0.107692	1.0	0.576923	1.0	1	1	0.061596	0.981481	0.653979	0
18642	1	0.092308	1.0	0.153846	1.0	1	1	0.088309	0.932099	0.826990	0
68485	0	0.061538	1.0	0.923077	1.0	1	1	0.000000	0.932099	0.224913	0
80894	0	0.046154	1.0	0.884615	1.0	1	1	0.054108	0.932099	0.114187	0
27777	2	0.107692	1.0	0.346154	1.0	1	1	0.051937	0.932099	0.352941	0
182944	0	0.107692	1.0	0.288462	1.0	1	1	0.057127	0.932099	0.224913	0
169021	1	0.092308	1.0	0.576923	0.0	1	2	0.040191	0.932099	0.743945	0

- Melakukan visualisasi data
 - Membuat heatmap untuk melihat nilai korelasi antara masing-masing fitur.

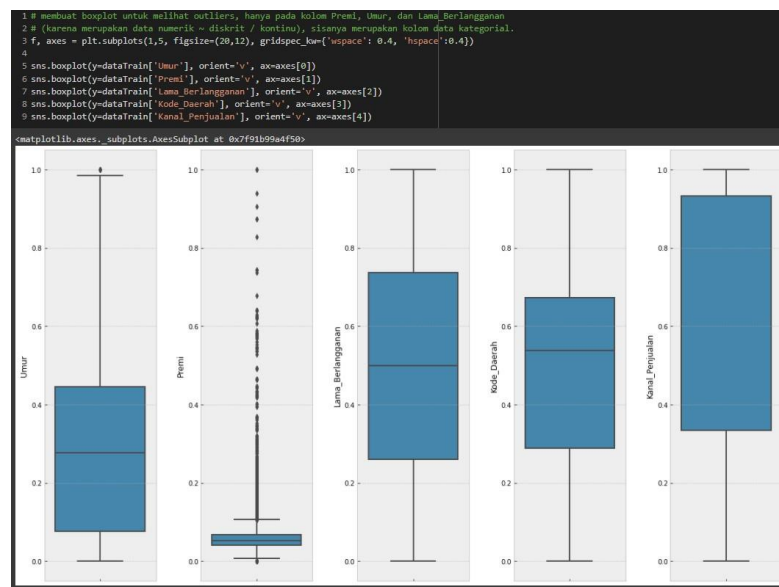
```

1 plt.figure(figsize=(15, 5))
2 sns.heatmap(dataTrain.corr(), annot = True,linewidths=.5)
3 plt.title("Dataset `TRAIN`" ,y=1.1)
4 plt.show()
5
6 plt.figure(figsize=(15, 5))
7 sns.heatmap(dataTest.corr(), annot = True,linewidths=.5)
8 plt.title("Dataset `TEST`" ,y=1.1)
9 plt.show()

```



- Membuat boxplot untuk mengecek outlier pada kolom premi, umur, dan lama berlangganan. (hanya tiga kolom tersebut karena kolom lainnya merupakan kategorikal).



Dari hasil boxplot, dapat dilihat terdapat nilai pencilan (outlier) yang cukup banyak pada fitur 'premi'. Namun jika kita analisa fitur 'premi', premi merupakan jumlah premi yang harus dibayarkan pelanggan per tahunnya, dan karena pada dataset tidak dijelaskan lebih lanjut untuk produk asuransi dari masing-masing pelanggan, maka kita tidak bisa menentukan apakah nilai pencilan pada fitur 'premi' merupakan nilai pencilan yang valid. Masing-masing pelanggan bisa saja membeli produk asuransi yang berbeda sehingga nilai pada fitur 'premi' menjadi terdistribusi merata seperti pada di boxplot. Sehingga kami memutuskan bahwa nilai pencilan (outliers) pada fitur 'premi' merupakan nilai yang normal dan tidak perlu untuk melakukan pembuangan outliers.

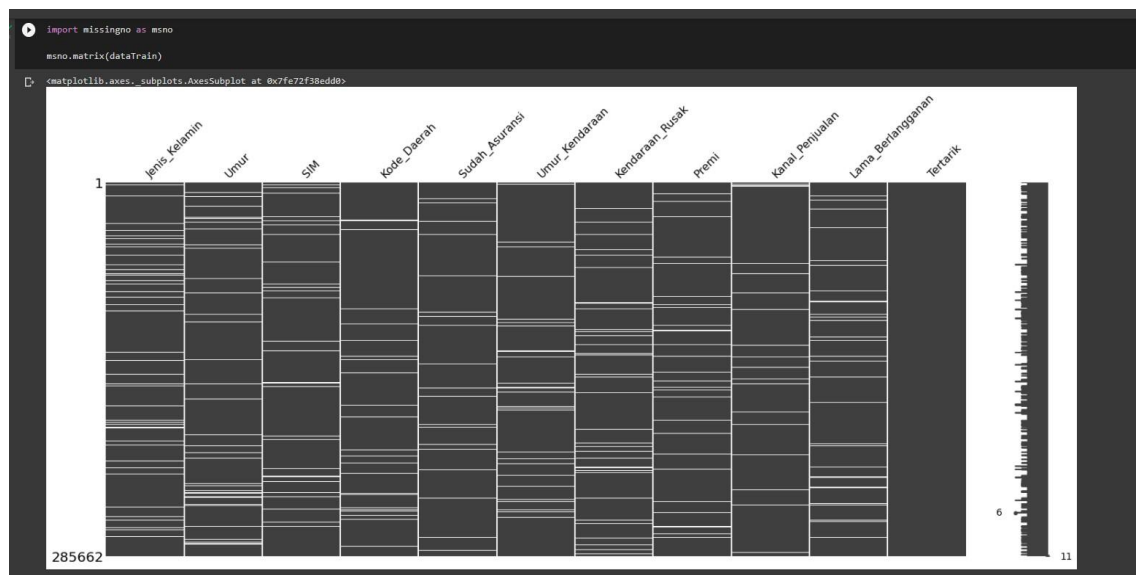
Begitu pun juga fitur 'umur', yang apabila pada kasus permasalahan ini (pelanggan tertarik membeli / tidak tertarik membeli), tidak melepas kemungkinan bahwa terdapat pelanggan yang cenderung lebih tua dibanding lainnya, sehingga outliers (pencilan) pada fitur 'umur' dianggap normal / dapat terjadi sehingga tidak perlu untuk melakukan pembuangan outliers.

3. Eksperimen

1. Eksperimen 1 :

- Mengelola nilai yang kosong (missing values) : dilakukan pembuangan pada baris tersebut.

Nilai kosong (missing values) cukup tersebar.



Untuk masing-masing fitur, terdapat nilai kosong sekitar 4-5%.

```
#Finding Missing values percentage in all columns
miss = pd.DataFrame(dataTrain.isnull().sum())
miss = miss.rename(columns={0:"miss_count"})
miss["miss_%"] = (miss.miss_count/len(dataTrain.index))*100
miss
```

	miss_count	miss_%
Jenis_Kelamin	14439	5.054575
Umur	14199	4.970560
SIM	14404	5.042323
Kode_Daerah	14291	5.002766
Sudah_Asuransi	14229	4.981062
Umur_Kendaraan	14275	4.997164
Kendaraan_Rusak	14187	4.966359
Premi	14510	5.079430
Kanal_Penjualan	14297	5.004866
Lama_Berlangganan	13926	4.874992
Tertarik	0	0.000000

Membuang baris yang terdapat nilai kosongnya.

```
# drop data yang ada NaN value nya
dataTrain.dropna(inplace = True)
# reset index
dataTrain.reset_index(inplace=True,drop=True)
# show data
dataTrain.head()
```

	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertarik
0	Wanita	30.0	1.0	33.0	1.0	< 1 Tahun	Tidak	28029.0	152.0	97.0	0
1	Pria	48.0	1.0	39.0	0.0	> 2 Tahun	Pernah	25800.0	29.0	158.0	0
2	Wanita	58.0	1.0	48.0	0.0	1-2 Tahun	Tidak	2630.0	124.0	63.0	0
3	Pria	21.0	1.0	35.0	1.0	< 1 Tahun	Tidak	22735.0	152.0	171.0	0
4	Wanita	20.0	1.0	8.0	1.0	< 1 Tahun	Tidak	30786.0	160.0	31.0	0

Setelah dilakukan pembuangan baris, terdapat sekitar 40% baris yang dibuang pada dataset

```
persentase_drop = ((rows_before - rows_after)/rows_before)*100
persentase_drop
```

40.13309435626719

- o Label data training yang tidak setara (imbalanced dataset) : dilakukan oversampling supaya data yang setara.

Untuk melakukan oversampling, kami menggunakan fungsi randomoversampling dari library imblearn.over_sampling

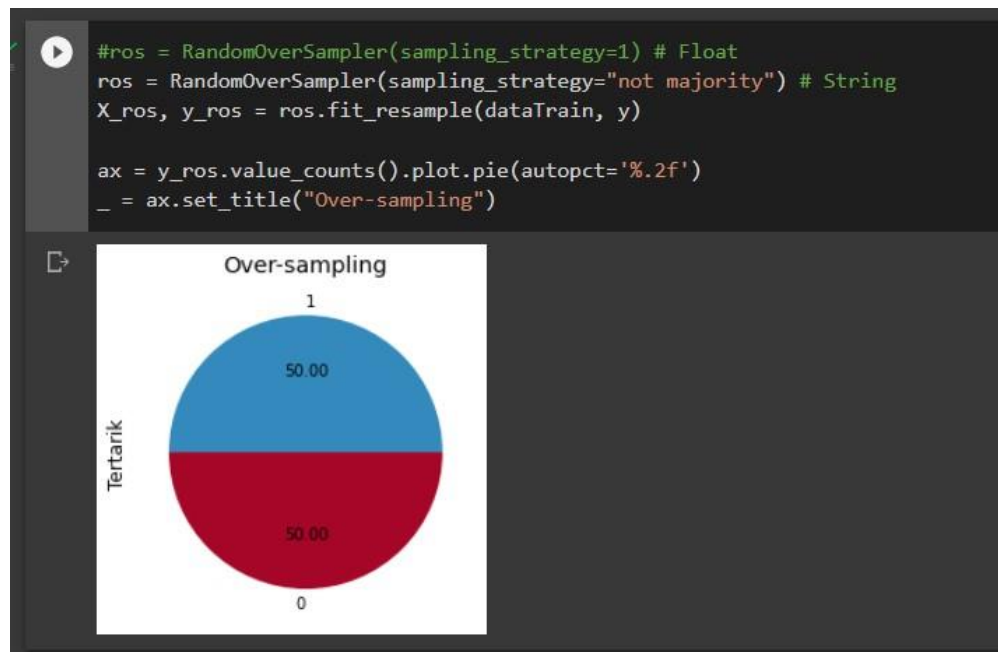
```
from imblearn.over_sampling import RandomOverSampler
```

Jumlah dataset sebelum dilakukan oversampling

```
dataTrain.shape
(171017, 10)

[46] y.shape
(171017,)
```

Melakukan overSampling



Jumlah dataset setelah dilakukan oversampling

```
[48] X_ros.shape
(300444, 10)

y_ros.shape
(300444,)
```

terjadi penambahan yang cukup banyak setelah dilakukan oversampling.

4. Pemodelan

1. Decision Tree

a. Mengimplementasikan rumus entropy

```
def entropy(y):  
    hist = np.bincount(y)  
    ps = hist / len(y)  
    return -np.sum([p * np.log2(p) for p in ps if p > 0])
```

b. Membuat class yang bisa menyimpan informasi node

```
class Node:  
    def __init__(  
        self, feature=None, threshold=None, left=None, right=None, *, value=None  
    ):  
        self.feature = feature  
        self.threshold = threshold  
        self.left = left  
        self.right = right  
        self.value = value  
  
    def is_leaf_node(self):  
        return self.value is not None
```

c. Pada kodingan ini pertama melakukan deklarasi parameternya terlebih dahulu dan pada metode fit akan dilakukannya pengecekan dan pembentukan tree dari metode grow tree yang melakukan pemilihan split yang terbaik dengan metode greedy search dan membentuk lagi dari split tersebut.


```

class DecisionTree:
    def __init__(self, min_samples_split=2, max_depth=100, n_feats=None):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
        self.root = None

    def fit(self, X, y):
        self.n_feats = X.shape[1] if not self.n_feats else min(self.n_feats, X.shape[1])
        self.root = self._grow_tree(X, y)

    def predict(self, X):
        return np.array([self._traverse_tree(x, self.root) for x in X])

    def _grow_tree(self, X, y, depth=0):
        n_samples, n_features = X.shape
        n_labels = len(np.unique(y))

        # stopping criteria
        if (
            depth >= self.max_depth
            or n_labels == 1
            or n_samples < self.min_samples_split
        ):
            leaf_value = self._most_common_label(y)
            return Node(value=leaf_value)

        feat_idx = np.random.choice(n_features, self.n_feats, replace=False)

        # greedily select the best split according to information gain
        best_feat, best_thresh = self._best_criteria(X, y, feat_idx)

        # grow the children that result from the split
        left_idx, right_idx = self._split(X[:, best_feat], best_thresh)
        left = self._grow_tree(X[left_idx, :], y[left_idx], depth + 1)
        right = self._grow_tree(X[right_idx, :], y[right_idx], depth + 1)
        return Node(best_feat, best_thresh, left, right)

```

```

def _best_criteria(self, X, y, feat_idx):
    best_gain = -1
    split_idx, split_thresh = None, None
    for feat_idx in feat_idx:
        X_column = X[:, feat_idx]
        thresholds = np.unique(X_column)
        for threshold in thresholds:
            gain = self._information_gain(y, X_column, threshold)

            if gain > best_gain:
                best_gain = gain
                split_idx = feat_idx
                split_thresh = threshold

    return split_idx, split_thresh

def _information_gain(self, y, X_column, split_thresh):
    # parent loss
    parent_entropy = entropy(y)

    # generate split
    left_idx, right_idx = self._split(X_column, split_thresh)

    if len(left_idx) == 0 or len(right_idx) == 0:
        return 0

    # compute the weighted avg. of the loss for the children
    n = len(y)
    n_l, n_r = len(left_idx), len(right_idx)
    e_l, e_r = entropy(y[left_idx]), entropy(y[right_idx])
    child_entropy = (n_l / n) * e_l + (n_r / n) * e_r

    # information gain is difference in loss before vs. after split
    ig = parent_entropy - child_entropy
    return ig

def _split(self, X_column, split_thresh):
    left_idx = np.argwhere(X_column <= split_thresh).flatten()
    right_idx = np.argwhere(X_column > split_thresh).flatten()
    return left_idx, right_idx

def _traverse_tree(self, x, node):
    if node.is_leaf_node():
        return node.value

    if x[node.feature] <= node.threshold:
        return self._traverse_tree(x, node.left)
    return self._traverse_tree(x, node.right)

def _most_common_label(self, y):
    counter = Counter(y)
    most_common = counter.most_common(1)[0][0]
    return most_common

```


2. Random Forest

- a. Pada kodingan ini, mengambil index dari sampel yang kemudian akan mengembalikan x dan y pada indeks tersebut. Selanjutnya, kodingan most common label mencari label dari y yang paling banyak

```
from collections import Counter

import numpy as np

def bootstrap_sample(X, y):
    n_samples = X.shape[0]
    idxs = np.random.choice(n_samples, n_samples, replace=True)
    return X[idxs], y[idxs]

def most_common_label(y):
    counter = Counter(y)
    most_common = counter.most_common(1)[0][0]
    return most_common
```

- b. Pada kodingan ini, mengimplementasikan random forest yang didapatkan dari beberapa decision tree. Pertama - tama kami mendeklarasikan terlebih dahulu parameter yang akan digunakan. Lalu membuat looping yang menghasilkan decision tree yang saling berbeda dari metode sampel sebelumnya. Selanjutnya, terdapat metode predict yang digunakan untuk mencari prediksi yang terbaik.

```
class RandomForest:
    def __init__(self, n_trees=10, min_samples_split=2, max_depth=100, n_feats=None):
        self.n_trees = n_trees
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
        self.trees = []

    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            tree = DecisionTree(
                min_samples_split=self.min_samples_split,
                max_depth=self.max_depth,
                n_feats=self.n_feats,
            )
            X_samp, y_samp = bootstrap_sample(X, y)
            tree.fit(X_samp, y_samp)
            self.trees.append(tree)

    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in self.trees])
        tree_preds = np.swapaxes(tree_preds, 0, 1)
        y_pred = [most_common_label(tree_pred) for tree_pred in tree_preds]
        return np.array(y_pred)
```

- c. Pada kodingan ini kami mengubah dataset yang telah kami oleh menjadi bentuk array agar bisa digunakan untuk model Random Forest

```
[ ] clf = RandomForest(n_trees=3, max_depth=10)

[ ] clf.fit(data_training, y_training)

[ ] y_pred_rf = clf.predict(data_testing)

[ ] def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy

[ ] acc = accuracy(y_testing, y_pred_rf)

    print("Accuracy:", acc)

Accuracy: 0.6899808980037364
```

3. Naive Bayes

- a. Pada bagian ini pertama - tama kami mengimplementasikan prior probability untuk digunakan pada metode fit seperti menghitung mean, var, dan prior. Setelah itu, kodingan tersebut akan mengimplementasikan perhitungan posterior probability dan class conditional yang setelahnya akan dipilih class yang memiliki probabilitas yang tertinggi.

```
class NaiveBayes:
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # calculate mean, var, and prior for each class
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self._classes):
            X_c = X[y == c]
            self._mean[idx, :] = X_c.mean(axis=0)
            self._var[idx, :] = X_c.var(axis=0)
            self._priors[idx] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        posteriors = []

        # calculate posterior probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._pdf(idx, x)))
            posterior = prior + posterior
            posteriors.append(posterior)

        # return class with highest posterior probability
        return self._classes[np.argmax(posteriors)]

    def _pdf(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

- b. Pada kodingan ini, kami membuat model dan melakukan perhitungan akurasi dari pemodelan yang telah dijalankan

```
nb = NaiveBayes()
nb.fit(data_training, y_training)

[ ] y_pred_nb = nb.predict(data_testing)

[ ] def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy

[ ] acc = accuracy(y_testing, y_pred_nb)

print("Accuracy:", acc)

Accuracy: 0.5820651147169336
```

4. Logistic Regression

- a. Pada kodingan ini, pertama - tama kami mendeklarasikan parameter yang akan digunakan, lalu kita mengimplementasikan looping gradient descent yang pertama menggunakan approximation yang telah diimplementasikan dan mengaplikasikan fungsi sigmoid yang telah dideklarasikan. Setelah itu, kami mengoperasikan gradien dan meng update parameter.

```
class LogisticRegression:
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        # gradient descent
        for _ in range(self.n_iters):
            # approximate y with linear combination of weights and x, plus bias
            linear_model = np.dot(X, self.weights) + self.bias
            # apply sigmoid function
            y_predicted = self._sigmoid(linear_model)

            # compute gradients
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)
            # update parameters
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return np.array(y_predicted_cls)

    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
```

- b. Pada kodingan ini, kami membuat model dan melakukan perhitungan akurasi dari pemodelan yang telah dijalankan

```
[ ] regressor = LogisticRegression(learning_rate=0.0001, n_iters=1000)
    regressor.fit(data_training, y_training)

def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy

[ ] y_pred_lr = regressor.predict(data_testing)

    print("LR classification accuracy:", accuracy(y_training, y_pred_lr))
```

5. Evaluasi Model

Evaluasi Model untuk meng-evaluasi model klasifikasi yang telah dibuat, kami menggunakan metode evaluasi seperti :

- Nilai akurasi model
Nilai akurasi model menunjukkan berapa banyak prediksi yang benar.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of all predictions}}$$

Namun nilai akurasi model sendiri tidak bisa menunjukkan apakah model cukup optimal. Dikarenakan nilai akurasi 90 % pada suatu model dianggap tidak optimal apabila pada dataset ternyata terdapat kategori label terdiri dari 90% label A dan 10 % label B (dataset tidak setara / *imbalanced dataset*). oleh karena itu, Dalam kasus ini, kami membutuhkan metrik lain untuk mengevaluasi model.

- Nilai dari precision and recall
Precision menunjukkan seberapa bagus model kita ketika prediksinya positif.

$$\text{Precision} = \frac{TP}{TP + FP}$$

sedangkan, recall adalah seberapa baik model kami dalam memprediksi kelas positif dengan benar.

$$Recall = \frac{TP}{TP + FN}$$

- F1 score

F1 score adalah weighted average dari nilai precision dan recall.

$$F1_score = 2 \frac{Precision * Recall}{Precision + Recall}$$

- Confusion matrix

Confusion matrix bukanlah metrik untuk mengevaluasi model, tetapi memberikan wawasan tentang prediksi. bentuk dari confusion matrix seperti sebagai berikut,

Confusion matrix for binary classification			
Actual value	A	TP	FN
	B	FP	TN
		A	B
		Predicted value	

Dengan melihat confusion matrix, kita dapat menganalisa jumlah,

- True Positive (TP), memprediksi kelas A sebagai kelas A
- True Negative (TN), memprediksi kelas B sebagai kelas B
- False Positive (FP), memprediksi kelas B sebagai kelas A
- False Negative (FN), memprediksi kelas A sebagai kelas B

Untuk menghitung nilai evaluasi diatas, kami menggunakan library sklearn yang sudah terdapat fungsi khusus untuk menghitungnya.

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
```

1. Evaluasi model dari eksperimen 1

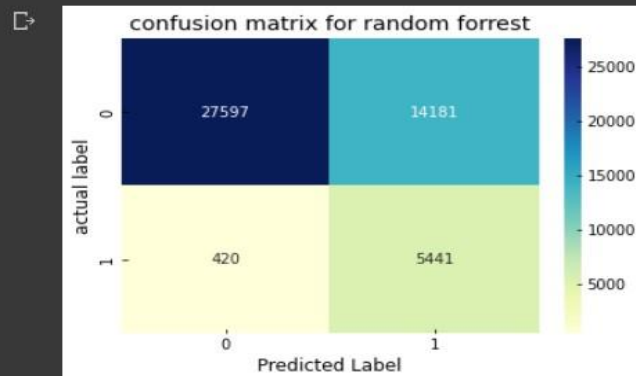
- Nilai akurasi model, nilai precision model, nilai recall model, nilai f-1 model dan confusion matrix

A. Model random forest

```
[73] # random forest
accuracy_rf = accuracy_score(y_testing, y_pred_rf)
precision_rf = precision_score(y_testing, y_pred_rf)
recall_rf = recall_score(y_testing, y_pred_rf)
f1_rf = f1_score(y_testing, y_pred_rf)
print("evaluasi random forrest")
print('accuracy_random_Forest : %.3f' %accuracy_rf)
print('precision_random_Forest : %.3f' %precision_rf)
print('recall_random_Forest : %.3f' %recall_rf)
print('f1-score_random_Forest : %.3f' %f1_rf)

evaluasi random forrest
accuracy_random_Forest : 0.694
precision_random_Forest : 0.277
recall_random_Forest : 0.928
f1-score_random_Forest : 0.427
```

```
# confusion matrix random forrest
cm = confusion_matrix(y_testing, y_pred_rf)
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('confusion matrix for random forrest')
plt.xlabel('Predicted Label')
plt.ylabel('actual label')
plt.show()
```

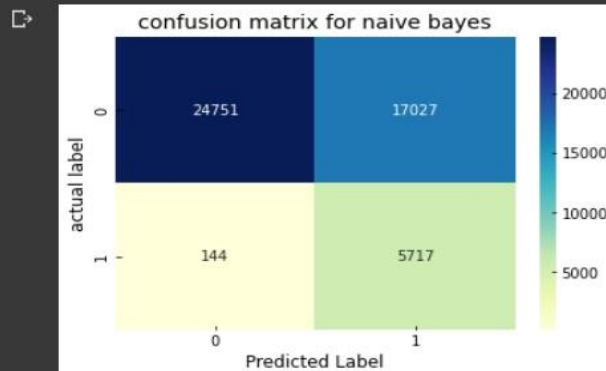


B. Model naive bayes

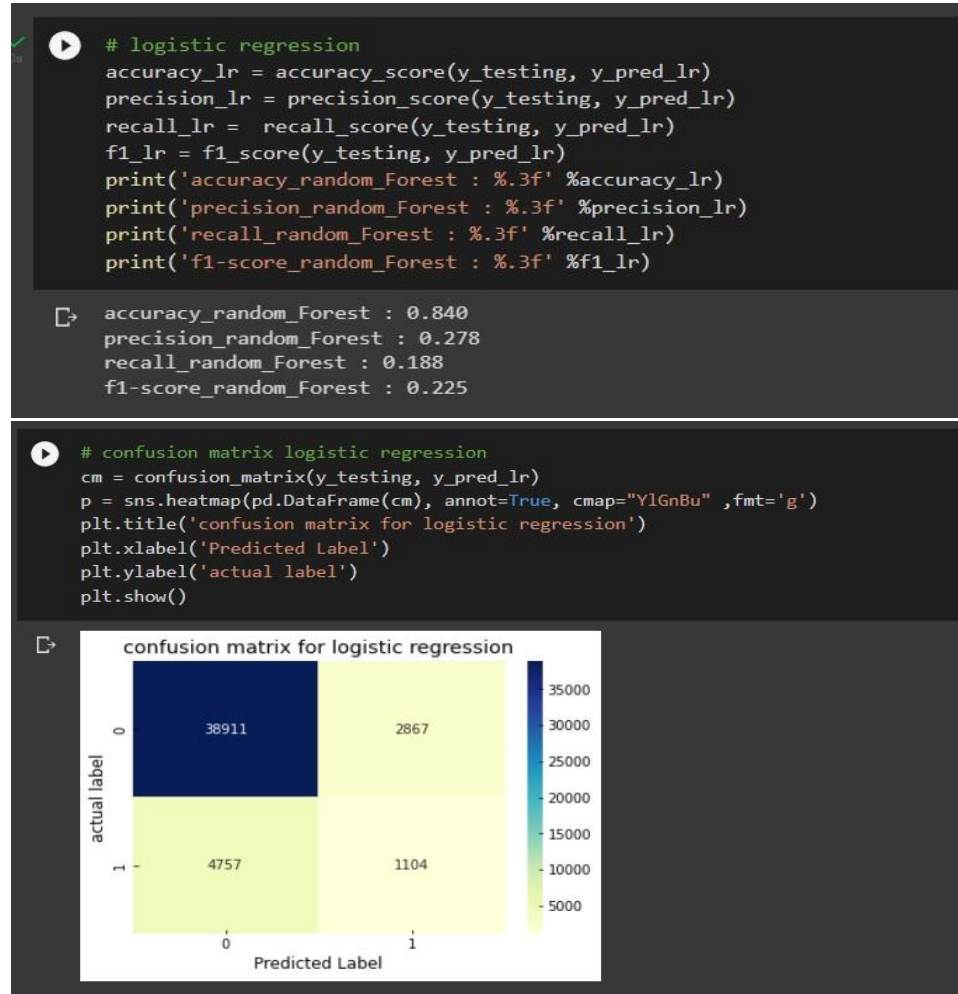
```
# naive bayes
accuracy_nb = accuracy_score(y_testing, y_pred_nb)
precision_nb = precision_score(y_testing, y_pred_nb)
recall_nb = recall_score(y_testing, y_pred_nb)
f1_nb = f1_score(y_testing, y_pred_nb)
print("evaluasi naive bayes")
print('accuracy_random_Forest : %.3f' %accuracy_nb)
print('precision_random_Forest : %.3f' %precision_nb)
print('recall_random_Forest : %.3f' %recall_nb)
print('f1-score_random_Forest : %.3f' %f1_nb)
```

```
evaluasi naive bayes
accuracy_random_Forest : 0.640
precision_random_Forest : 0.251
recall_random_Forest : 0.975
f1-score_random_Forest : 0.400
```

```
# confusion matrix naive bayes
cm = confusion_matrix(y_testing, y_pred_nb)
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('confusion matrix for naive bayes')
plt.xlabel('Predicted Label')
plt.ylabel('actual label')
plt.show()
```

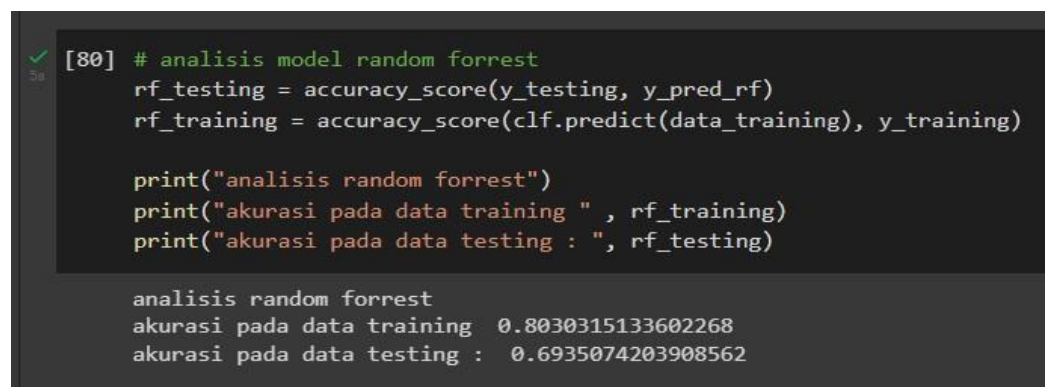


C. Model logistic regression



- Analisis overfitting pada model

A. Model Random Forest



B. Model naive bayes

```
✓ 20s # analisis model naive bayes
nb_testing = accuracy_score(y_testing, y_pred_nb)
nb_training = accuracy_score(nb.predict(data_training), y_training)

print("analisis naive bayes")
print("akurasi pada data training " , nb_training)
print("akurasi pada data testing : ", nb_testing)

➤ analisis naive bayes
akurasi pada data training  0.7844856279373195
akurasi pada data testing :  0.6395600243497974
```

C. Model Logistic Regression

```
⏪ # analisis model logistic regression
lr_testing = accuracy_score(y_testing, y_pred_lr)
lr_training = accuracy_score(regressor.predict(data_training), y_training)

print("analisis logistic regression")
print("akurasi pada data training " , lr_training)
print("akurasi pada data testing : ", lr_testing)

➤ analisis logistic regression
akurasi pada data training  0.5620947664123763
akurasi pada data testing :  0.839963055479754
```

BAB 3

KESIMPULAN

1. Eksperimen 1 :

- mengelola nilai yang kosong (missing values) : dilakukan pembuangan pada baris tersebut.
- label data training yang tidak setara (imbalanced dataset) : dilakukan oversampling supaya data yang setara.

Hasil analisis

memilih model terbaik berdasarkan nilai akurasi model

	Eksperimen 1	
Model	akurasi di training	akurasi di testing
Random Forest	0.803	0.693
Naive bayes	0.784	0.639
Logistic regression	0.562	0.839

Berdasarkan hasil akurasi dari ketiga model yang digunakan (random forest, naive bayes, logistic regression), model yang paling optimal untuk digunakan dalam memprediksi data training pada kasus kami adalah dengan menggunakan model logistic regression. Untuk kedua eksperimen diatas, cenderung menghasilkan pola yang sama pada nilai akurasi yang dihasilkan. Dimana untuk model random forest dan naive bayes, akurasi pada data training cenderung lebih besar dibandingkan akurasi pada data testing. Berbeda dengan model logistic regression, yang akurasi pada data testingnya, jauh lebih besar dibandingkan akurasi pada data training.

Memilih model yang terbaik berdasarkan nilai recall (recall adalah seberapa baik model kami dalam memprediksi kelas positif dengan benar) berdasarkan hasil analisis nilai-nilai pada confusion matrix masing-masing model, khususnya pada kasus percobaan kami, yaitu memprediksi apakah pelanggan tertarik untuk membeli kendaraan baru atau tidak berdasarkan data

pelanggan di dealer. Dengan asumsi kami bahwa hasil prediksi label nantinya digunakan untuk menyusun strategi penjualan kendaraan kedepannya. kami merasa kebenaran dalam memprediksi pelanggan yang tertarik, memiliki bobot / prioritas yang lebih tinggi dibandingkan kebenaran dalam memprediksi pelanggan yang tidak tertarik. Dikarenakan bagi dealer, untuk melakukan penjualan dengan efektif, dealer pastinya hanya akan menjual kendaraannya pada pelanggan yang memang tertarik, tidak perlu membuang waktu untuk menjual kendaraan kepada pelanggan yang tidak tertarik. sehingga memilih model yang memiliki nilai recall yang tinggi dapat dijadikan pertimbangan nantinya dalam memilih model mana yang paling bagus dalam memprediksi pelanggan yang tertarik.

Nilai recall untuk masing-masing model

	Eksperimen 1
Model	
random forest	0.928
naive bayes	0.975
logistic regression	0.188

Sehingga untuk model yang paling bagus dalam memprediksi pelanggan yang tertarik, merupakan model naive bayes.