Makalah Penyisihan Data Mining Joints 2019

by Sour Soup Team

I. Latar Belakang

Pak Blankon, seorang pengusaha baru tertarik untuk membangun usaha kos-kosan di Yogyakarta. Dia merupakan seorang pendatang di Yogyakarta. Karena dia pendatang baru di Yogyakarta, dia kebingungan mencari target pasar yang sesuai pada suatu daerah dengan jenis usahanya dan apa saja faktor yang mempengaruhinya. Kebetulan, Pak Blankon memiliki seorang teman yang bisa membantu dia memprediksi apakah di suatu area itu cocok untuk dibangun kos putra, putri, atau campur berdasarkan faktor-faktor tertentu. Jika kami diposisikan sebagai teman Pak Blankon yang baik hati dan senang untuk membantu, diharapkan dapat membantu dia. Maka dari itu, kami akan membuat model yang dapat menentukan dari suatu daerah akan cocok untuk kos putra, putri atau campur.

II. Tujuan dan Manfaat

Tujuan dari proses data *mining* ini adalah untuk membantu Pak Blankon untuk memprediksi tempat usaha kos-kosan di Yogyakarta berdasarkan fasilitas yang dimiliki, luas kamar, jumlah kamar, jumlah pencarian pada kos tersebut, dan *place of interest* pada daerah tertentu. Sehingga, pada akhir proses data *mining* ini, didapatkan hasil prediksi apakah area tersebut cocok untuk dibangun kos putra, putri, atau campuran.

III. Metode Data *Mining*

Algoritma yang diterapkan pada proses *data mining* ini adalah metode *voting classifier*. Pada metode ini kami menggabungkan *gradient boosting classifier, light gradient boosting classifier,* dan *extreme gradient boosting classifier* dengan bobot masing-masing *classifier* adalah 1, 2, dan 1. Adapun *software* yang digunakan untuk membangun model tersebut adalah *jupyter notebook,* dengan menggunakan Bahasa pemrograman python. *Dataset* yang kami gunakan adalah keseluruhan data *train* yang terdapat pada file 'train.csv' yang telah diberikan.

IV. Analisis

1. *Feature* atau atribut pada *dataset*

   Pada *dataset* terdapat 15 atribut dan 1 kelas (*gender)*. Adapun atributnya adalah '*fac_1*' – '*fac_8*' (fasilitas yang dimiliki suatu kos-kosan), '*poi_1*' – '*poi_3*' (*place of interest*), '*size'* (luas kamar kos), *'price_monthly'* (harga sewa perbulannya), *'room_count'* (jumlah kamar yang dimiliki suatu kos-kosan), dan '*total_call*' (jumlah pencarian ke suatu kos). Contoh *dataset* terdapat pada gambar 1.

	id	fac_1	fac_2	fac_3	fac_4	fac_5	fac_6	fac_7	fac_8	poi_1	poi_2	poi_3	size	price_monthly	room_count	total_call	gender
0	1	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1778.0	10038.0	4106.0	9.00	1500000.0	6.0	72	campur
1	2	1.0	1.0	0.0	1.0	1.0	1.0	0.0	NaN	4548.0	9332.0	6867.0	12.00	1500000.0	30.0	56	campur
2	3	1.0	NaN	1.0	1.0	1.0	1.0	0.0	1.0	5174.0	9021.0	3693.0	12.00	1600000.0	20.0	109	campur
3	4	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1490.0	8954.0	2139.0	8.25	1500000.0	15.0	54	campur
4	5	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1688.0	8851.0	2145.0	14.85	2100000.0	10.0	19	campur

Gambar 1 Contoh *dataset* yang terdapat pada file 'train.csv'.

2. Mengatasi *Missing Data*

   Pada saat menganalisis *dataset,* kami menemukan *missing data* pada beberapa atribut yang harus diatasi dengan baik sesuai dengan kondisi masing-masing kolom/atribut. Detil *missing data* pada *dataset* dapat dilihat pada Tabel 1.

Tabel 1 *Missing data* pada masing-masing atribut.

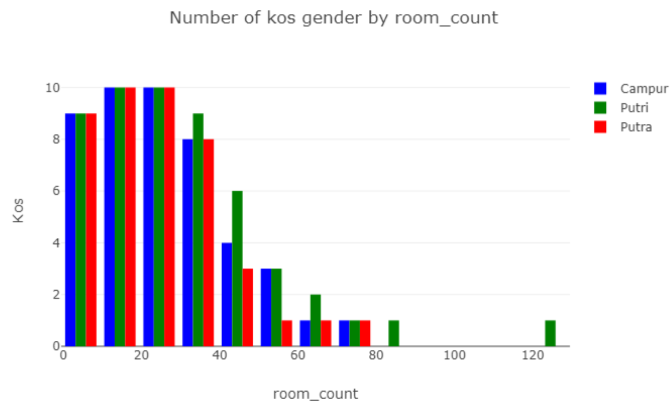
	Total	Percent
poi_3	86	2.611600
price_monthly	85	2.581233
fac_7	78	2.368661
fac_2	74	2.247191
fac_5	72	2.186456
fac_4	71	2.156089
room_count	70	2.125721
fac_8	68	2.064986
size	68	2.064986
poi_2	67	2.034619
poi_1	67	2.034619
fac_6	64	1.943517
fac_1	63	1.913149
fac_3	62	1.882782
gender	0	0.000000
total_call	0	0.000000
id	0	0.000000

   a. Mengatasi *missing data* pada atribut '*fac_1*' – '*fac_8*'

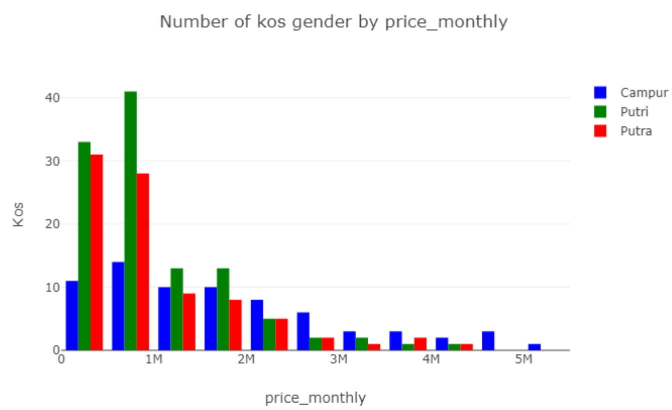
      Untuk atribut ini kami mengganti nilai *NaN* (*missing data)* dengan 0 yang artinya suatu kos dianggap tidak memiliki fasilitas tersebut.

   b. Mengatasi *missing data* pada atribut '*room_count*', '*price_monthly*', dan '*size*'

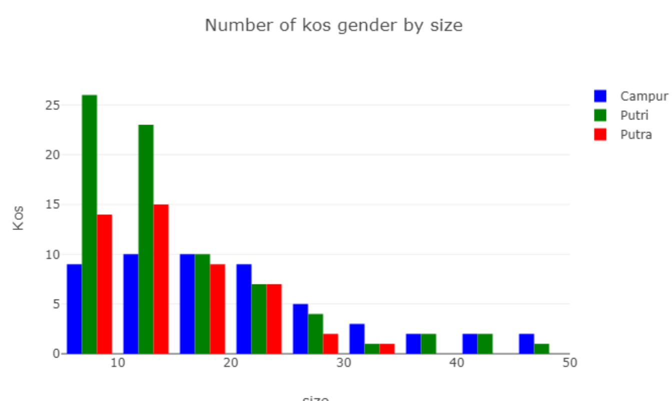
Untuk atribut-atribut tersebut kami mengganti nilai *NaN* dengan median pada masing-masing atribut. Hal ini dikarenakan atribut-atribut tersebut skew, hal tersebut digambarkan pada gambar 2, 3 dan 4.



Gambar 2 Distribusi atribut *'room_count'* terhadap *gender.*



Gambar 3 Distribusi atribut *'price_monthly'* terhadap *gender.*



Gambar 4 Distribusi atribut *'size'* terhadap *gender.*

c. Mengatasi *missing data* pada atribut '*poi_1', 'poi_2', dan 'poi_3'*

Untuk atribut-atribut tersebut kami melakukan pendekatan yang berbeda dari atribut lainnya. Menggunakan metode *linear regression* kami akan memprediksi nilai dari satu atribut dari 2 atribut lainnya. Akan tetapi jika ada suatu data dengan nilai *NaN* lebih atau sama dengan dua atribut, data tersebut kami *drop* atau dihapus. Sebagai contoh, kami mengganti *NaN* pada atribut '*poi_1*' dengan hasil prediksi *linear regression* atribut '*poi_2*' dan '*poi_3*'. Tetapi jika pada suatu data, atribut '*poi_1*' dan '*poi_2*' bernilai NaN, maka data tersebut akan dihapus.

3. *Feature Engineering*

a. Jumlah Fasilitas

Feature engineering yang pertama kali kami lakukan adalah dengan menambah atribut jumlah fasilitas yang dimiliki oleh suatu kos. Prosesnya sederhana, hanya dengan menghitung jumlah angka 1 pada seluruh atribut fasilitas ('*fac_1*' – '*fac_8*').

b. Total Jarak

Selain itu kami menambah atribut baru yaitu 'total_jarak'. Atribut tersebut merupakan penjumlahan antara '*poi_1*' – '*poi_3*' pada setiap *dataset.*

c. Premium Kos

Proses selanjutnya adalah menambah atribut '*premium*' yang artinya apakah kos tersebut harga sewa perbulannya lebih dari 2.000.000 atau tidak.

d. *Wanted* Kos

Proses selanjutnya adalah menambah atribut '*wanted*' yang artinya apakah kos tersebut jumlah pencariannya lebih dari 100 atau tidak.

e. Kategorisasi

Pada proses ini kami menambah atribut dengan mengategorikan nilai pada atribut yang bertipe *number*.

i. *Price Monthly*

Pada atribut '*price_monthly*' kami mengategorikannya berdasarkan *quartile* pada data atribut tersebut. Sehingga terdapat atribut baru yaitu '*cat_price*' yang memiliki nilai unik [0,1,2]. Nilai quartile atribut '*price_monthly*' dan kategorinya terdapat pada tabel 2.

Tabel 2 Kategorisasi atribut *'price_monthly'.*

	price_q
0	(154999.999, 500000.0]
1	(500000.0, 850000.0]
2	(850000.0, 5000000.0]

ii. *Size*

Pada atribut '*size*' kami mengategorikan berdasarkan quartile pada data atribut tersebut. Sehingga terdapat atribut baru yaitu '*cat_size*' yang memiliki nilai unik [0,1,2]. Nilai quartile atribut '*size*' dan kategorinya terdapat pada tabel 3.

Tabel 3 Kategorisasi atribut *'size'.*

	size_q
0	(5.999, 9.0]
1	(9.0, 12.0]
2	(12.0, 48.0]

iii. Jarak

Pada atribut 'total_jarak' kami mengategorikan berdasarkan quartile pada data atribut tersebut. Sehingga terdapat atribut baru yaitu 'cat_jarak' yang memiliki nilai unik [0,1,2]. Nilai quartile atribut 'total_jarak' dan kategorinya terdapat pada tabel 4.

Tabel 4 Kategorisasi atribut *'total_jarak'.*

	jarak_q
0	(10448.911, 14949.0]
1	(14949.0, 20375.0]
2	(20375.0, 150297.0]

iv. *POI*

Pada atribut '*poi_1*' – '*poi_3*' kami mengategorikan nilainya menjadi jauh atau dekat dengan membagi 2 data pada atribut-atribut tersebut. Sehingga pada akhir proses ini terdapat atribut baru yaitu *'cat_poi_1'*, *'cat_poi_2',* dan *'cat_poi_3'* yang memiliki nilai unik [0,1]. Nilai kategorisasi poi dapat dilihat pada tabel 5.

Tabel 5 Kategorisasi atribut *'poi_1', 'poi_2',* dan *'poi_3'.*

	poi_1_q		poi_2_q		poi_3_q
0	(518.999, 3961.0]	0	(167.999, 9249.0]	0	(323.999, 3930.0]
1	(3961.0, 48675.0]	1	(9249.0, 55105.0]	1	(3930.0, 46517.0]

4. *Feature Selection*

   Pada proses ini kami menghapus fitur fac_7 karena hampir 90% isinya adalah 0 (tidak ada fasilitas), kami juga melakukan *encoding* nilai yang bertipe *number* dengan teknik *mean encode.* Teknik tersebut melakukan *encoding* dengan menghitung jumlah kemunculan nilai unik atribut dibagi dengan total data. Hal ini dilakukan agar hasil encode memiliki relasi dengan kelas *output*.

5. *Modelling*

   Pada proses ini kami menerapkan algoritma *voting classifier* yang menggabungkan *gradient boosting classifier, light gradient boosting classifier,* dan *extreme gradient boosting classifier* dengan bobot masing-masingnya adalah 1, 2, dan 1. Voting *classifier* karena algoritma ini menghitung rata-rata hasil probabilitas kelas dari tiga *classifier* yang telah kami pilih, sehingga hasil akan lebih stabil dibandingkan hanya menggunakan satu *classifier*. *Dataset* yang telah melalui proses-proses sebelumnya akan dimodelkan dengan algoritma tersebut. Model yang telah jadi diterapkan pada data *test* yang ada pada file 'test_data.csv' untuk mengklasifikasi data tersebut dan disimpan pada file .csv.

V. Kesimpulan

Proses data *mining* yang kami lakukan dimulai dengan melakukan proses *exploratory data analysis* (analisis *dataset*) yang terdiri dari proses mengatasi *missing data*, *feature engineering,* dan *feature selection*. Setelah itu kami lakukan *modelling* dengan menggunakan metode *voting classifier*. Hasil pemodelan diterapkan pada data *test* untuk dilakukan klasifikasi pada data *test* tersebut. Pada akhirnya, hasil prediksi klasifikasi dengan model yang kami bangun disimpan pada file 'Submmission_19.csv'.

VI. Dokumentasi

Variable Explanation

- Id : id kost
- fac_ 1 - fac_8 : fasilitas kost. Nilai 0 berarti tidak ada fasilitas, nilai 1 berarti ada fasilitas.
- poi_1 : jarak ke POI 1
- poi_2 : jarak ke POI 2
- poi_3 : jarak ke POI 3
- size : luas kamar
- room_count : jumlah kamar
- total_call : jumlah pencarian ke kost tersebut
- gender : jenis gender yang ditampung oleh kost tersebut (dependent)

Preparing the Tools

```
[ ]  from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier
     from sklearn.linear_model import LinearRegression
     from catboost import CatBoostClassifier
     import lightgbm as lgb
     import xgboost as xgb

     from sklearn.model_selection import train_test_split, KFold, GridSearchCV
     from sklearn.metrics import roc_auc_score,roc_curve,auc
     from sklearn.feature_selection import SelectFromModel
     from sklearn.model_selection import StratifiedKFold
     from sklearn.metrics import classification_report
     from sklearn.preprocessing import StandardScaler
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import roc_auc_score

     from sklearn import metrics
     from sklearn.svm import SVC

     import matplotlib.pyplot as plt
     from pathlib import Path
     import seaborn as sns
     import pandas as pd
     import numpy as np
     import random
     import scipy
     import time
     import sys
     import os

     from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
     import plotly.figure_factory as ff
     import plotly.graph_objs as go
     from plotly import tools

     init_notebook_mode(connected=True)
```

Functions

```
[ ]  def add_noise(series, noise_level):
         return series * (1 + noise_level * np.random.randn(len(series)))
```

```python
def target_encode(trn_series=None,
                  tst_series=None,
                  target=None,
                  min_samples_leaf=1,
                  smoothing=1,
                  noise_level=0):
    """
    Smoothing is computed like in the following paper by Daniele Micci-Barreca
    https://kaggle2.blob.core.windows.net/forum-message-attachments/225952/7441/high%20cardinality%20categoricals.pdf
    trn_series : training categorical feature as a pd.Series
    tst_series : test categorical feature as a pd.Series
    target : target data as a pd.Series
    min_samples_leaf (int) : minimum samples to take category average into account
    smoothing (int) : smoothing effect to balance categorical average vs prior
    """
    assert len(trn_series) == len(target)
    assert trn_series.name == tst_series.name
    temp = pd.concat([trn_series, target], axis=1)
    # Compute target mean
    averages = temp.groupby(by=trn_series.name)[target.name].agg(["mean", "count"])
    # Compute smoothing
    smoothing = 1 / (1 + np.exp(-(averages["count"] - min_samples_leaf) / smoothing))
    # Apply average function to all target data
    prior = target.mean()
    # The bigger the count the less full_avg is taken into account
    averages[target.name] = prior * (1 - smoothing) + averages["mean"] * smoothing
    averages.drop(["mean", "count"], axis=1, inplace=True)
    # Apply averages to trn and tst series
    ft_trn_series = pd.merge(
        trn_series.to_frame(trn_series.name),
        averages.reset_index().rename(columns={'index': target.name, target.name: 'average'}),
        on=trn_series.name,
        how='left')['average'].rename(trn_series.name + '_mean').fillna(prior)
    # pd.merge does not keep the index so restore it
    ft_trn_series.index = trn_series.index
    ft_tst_series = pd.merge(
        tst_series.to_frame(tst_series.name),
        averages.reset_index().rename(columns={'index': target.name, target.name: 'average'}),
        on=tst_series.name,
        how='left')['average'].rename(trn_series.name + '_mean').fillna(prior)
    # pd.merge does not keep the index so restore it
    ft_tst_series.index = tst_series.index
    return add_noise(ft_trn_series, noise_level), add_noise(ft_tst_series, noise_level)
```

```python
def get_categories(data, val):
    tmp = data[val].value_counts()
    return pd.DataFrame(data={'Number': tmp.values}, index=tmp.index).reset_index()

def get_gender_categories(data, val):
    tmp = data.groupby('gender')[val].value_counts()
    return pd.DataFrame(data={'Number': tmp.values}, index=tmp.index).reset_index()

def draw_trace_bar(data_df,color='Blue'):
    trace = go.Bar(
            x = data_df['index'],
            y = data_df['Number'],
            marker=dict(color=color),
            text=data_df['index']
        )
    return trace


def plot_bar(data_df, title, xlab, ylab,color='Blue'):
    trace = draw_trace_bar(data_df, color)
    data = [trace]
    layout = dict(title = title,
              xaxis = dict(title = xlab, showticklabels=True, tickangle=0,
                          tickfont=dict(
                            size=10,
                            color='black'),),
              yaxis = dict(title = ylab),
              hovermode = 'closest'
             )
    fig = dict(data = data, layout = layout)
    iplot(fig, filename='draw_trace')
```

```python
def plot_two_bar(data_df1, data_df2, title1, title2, xlab, ylab):
    trace1 = draw_trace_bar(data_df1, color='Blue')
    trace2 = draw_trace_bar(data_df2, color='Lightblue')

    fig = tools.make_subplots(rows=1,cols=2, subplot_titles=(title1,title2))
    fig.append_trace(trace1,1,1)
    fig.append_trace(trace2,1,2)

    fig['layout']['xaxis'].update(title = xlab)
    fig['layout']['xaxis2'].update(title = xlab)
    fig['layout']['yaxis'].update(title = ylab)
    fig['layout']['yaxis2'].update(title = ylab)
    fig['layout'].update(showlegend=False)


    iplot(fig, filename='draw_trace')
```

```python
def plot_gender_bar(data_df, var, ytitle= 'Number of kos',title= 'Number of kos gender by {}'):
    dfC = data_df[data_df['gender']=='campur']
    dfPi = data_df[data_df['gender']=='putri']
    dfPu = data_df[data_df['gender']=='putra']


    traceC = go.Bar(
        x = dfC[var],y = dfC['Number'],
        name='Campur',
        marker=dict(color="Blue"),
        text=dfC['Number']
    )
    tracePi = go.Bar(
        x = dfPi[var],y = dfPi['Number'],
        name='Putri',
        marker=dict(color="Green"),
        text=dfPi['Number']
    )

    tracePu = go.Bar(
        x = dfPu[var],y = dfPu['Number'],
        name='Putra',
        marker=dict(color="Red"),
        text=dfPu['Number']
    )

    data = [traceC, tracePi,tracePu]
    layout = dict(title = title.format(var),
          xaxis = dict(title = var, showticklabels=True),
          yaxis = dict(title = ytitle),
          hovermode = 'closest'
    )
    fig = dict(data=data, layout=layout)

    iplot(fig, filename='draw_trace')
```

```python
def draw_trace_histogram(data_df,color='Blue'):
    trace = go.Histogram(
            x = data_df['index'],
            y = data_df['Number'],
            marker=dict(color=color),
            text=data_df['index']
        )
    return trace

def plot_two_histogram(data_df1, data_df2, title1, title2, xlab, ylab):
    trace1 = draw_trace_histogram(data_df1, color='Blue')
    trace2 = draw_trace_histogram(data_df2, color='Lightblue')

    fig = tools.make_subplots(rows=1,cols=2, subplot_titles=(title1,title2))
    fig.append_trace(trace1,1,1)
    fig.append_trace(trace2,1,2)

    fig['layout']['xaxis'].update(title = xlab)
    fig['layout']['xaxis2'].update(title = xlab)
    fig['layout']['yaxis'].update(title = ylab)
    fig['layout']['yaxis2'].update(title = ylab)
    fig['layout'].update(showlegend=False)


    iplot(fig, filename='draw_trace')
```

```python
def plot_survived_histogram(data_df, var):
    dfC = data_df[data_df['gender']=='campur']
    dfPi = data_df[data_df['gender']=='putri']
    dfPu = data_df[data_df['gender']=='putra']


    traceC = go.Histogram(
        x = dfC[var],y = dfC['Number'],
        name='Campur',
        marker=dict(color="Blue"),
        text=dfC['Number']
    )
    tracePi = go.Histogram(
        x = dfPi[var],y = dfPi['Number'],
        name='Putri',
        marker=dict(color="Green"),
        text=dfPi['Number']
    )

    tracePu = go.Histogram(
        x = dfPu[var],y = dfPu['Number'],
        name='Putra',
        marker=dict(color="Red"),
        text=dfPu['Number']
    )

    data = [traceC, tracePi,tracePu]
    layout = dict(title = 'Number of kos gender by {}'.format(var),
          xaxis = dict(title = var, showticklabels=True),
          yaxis = dict(title = 'Number of passengers'),
          hovermode = 'closest'
    )
    fig = dict(data=data, layout=layout)

    iplot(fig, filename='draw_trace')
```

```python
[ ] def shape(df):
        return '{:,} rows - {:,} columns'.format(df.shape[0], df.shape[1])
```

```python
[ ] def missing_data(data):
        total = data.isnull().sum().sort_values(ascending = False)
        percent = (data.isnull().sum()/data.isnull().count()*100).sort_values(ascending = False)
        return pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
```

```
[ ]  def poi_column(df):
         for i in range(1,4):
             i = str(i)
             df['poi_'+i] = df['poi_'+i].astype('float')
     #    df['jml_fac'] = df[['fac_'+str(i) for i in range(1,9)]].agg('sum',axis=1).astype('category')
         return df
```

```
[ ]  df_train = pd.read_csv('train.csv',engine='python')
     df_test = pd.read_csv('test_data.csv',engine='python')
     df_train.columns = df_train.columns.str.lower()
     df_test.columns = df_test.columns.str.lower()

     print(f'Train Shape: {shape(df_train)}')
     print(f'Test Shape: {shape(df_test)}')
```

```
Train Shape: 3,293 rows - 17 columns
Test Shape: 824 rows - 16 columns
```

Data Exploration

```
[ ]  df_train.head()
```

	id	fac_1	fac_2	fac_3	fac_4	fac_5	fac_6	fac_7	fac_8	poi_1	poi_2	poi_3	size	price_monthly	room_count	total_call	gender
0	1	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1778.0	10038.0	4106.0	9.00	1500000.0	6.0	72	campur
1	2	1.0	1.0	0.0	1.0	1.0	1.0	0.0	NaN	4548.0	9332.0	6867.0	12.00	1500000.0	30.0	56	campur
2	3	1.0	NaN	1.0	1.0	1.0	1.0	0.0	1.0	5174.0	9021.0	3693.0	12.00	1600000.0	20.0	109	campur
3	4	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1490.0	8954.0	2139.0	8.25	1500000.0	15.0	54	campur
4	5	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1688.0	8851.0	2145.0	14.85	2100000.0	10.0	19	campur

```
[ ]  df_train.describe()
```

	id	fac_1	fac_2	fac_3	fac_4	fac_5	fac_6	fac_7	fac_8	poi_1	
count	3293.000000	3230.000000	3219.000000	3231.000000	3222.000000	3221.000000	3229.000000	3215.000000	3225.000000	3226.000000	3226.
mean	1647.000000	0.261610	0.608263	0.456515	0.562384	0.641416	0.427687	0.004666	0.518450	4679.478921	9920.
std	950.751545	0.439579	0.488214	0.498183	0.496170	0.479659	0.494820	0.068156	0.499737	3569.137245	4714.
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	519.000000	168.
25%	824.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2355.500000	7875.
50%	1647.000000	0.000000	1.000000	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	3961.000000	9241.
75%	2470.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	5900.750000	12422.
max	3293.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	48675.000000	55105.

```
[ ]  df_test.describe()
```

	id	fac_1	fac_2	fac_3	fac_4	fac_5	fac_6	fac_7	fac_8	poi_1	poi_2	
count	824.000000	824.000000	824.000000	824.000000	824.000000	824.000000	824.000000	824.000000	824.000000	824.000000	824.000000	
mean	3705.500000	0.260922	0.625000	0.496359	0.523058	0.595874	0.450243	0.007282	0.695388	4542.400485	9971.084951	
std	238.012605	0.439404	0.484417	0.500290	0.499771	0.491020	0.497820	0.085072	0.460522	3175.305908	4261.309222	
min	3294.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	568.000000	306.000000	
25%	3499.750000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2340.500000	7864.750000	
50%	3705.500000	0.000000	1.000000	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	4003.500000	9441.000000	
75%	3911.250000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	5759.500000	12151.500000	
max	4117.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	33290.000000	37721.000000	3

```
[ ]  for i in range(1,9):
         print(f"fac_{i} : {df_train['fac_'+str(i)].value_counts()}")
```

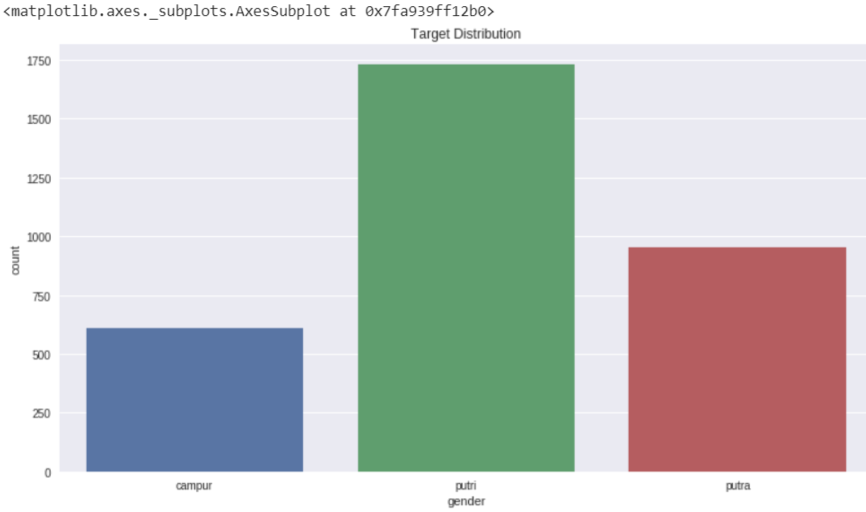
```
fac_1 : 0.0    2385
1.0     845
Name: fac_1, dtype: int64
fac_2 : 1.0    1958
0.0    1261
Name: fac_2, dtype: int64
fac_3 : 0.0    1756
1.0    1475
Name: fac_3, dtype: int64
fac_4 : 1.0    1812
0.0    1410
Name: fac_4, dtype: int64
fac_5 : 1.0    2066
0.0    1155
Name: fac_5, dtype: int64
fac_6 : 0.0    1848
1.0    1381
Name: fac_6, dtype: int64
fac_7 : 0.0    3200
1.0      15
Name: fac_7, dtype: int64
fac_8 : 1.0    1672
0.0    1553
Name: fac_8, dtype: int64
```

```
[ ]  fig, ax = plt.subplots(figsize=(13,7))
     ax.set_title('Target Distribution')
     sns.countplot(df_train['gender'])
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:1428: FutureWarning:

remove_na is deprecated and is a private function. Do not use.

<matplotlib.axes._subplots.AxesSubplot at 0x7fa939ff12b0>
```



Missing Data

```
[ ] missing_data(df_train)
```

	Total	Percent
poi_3	86	2.611600
price_monthly	85	2.581233
fac_7	78	2.368661
fac_2	74	2.247191
fac_5	72	2.186456
fac_4	71	2.156089
room_count	70	2.125721
fac_8	68	2.064986
size	68	2.064986
poi_2	67	2.034619
poi_1	67	2.034619
fac_6	64	1.943517
fac_1	63	1.913149
fac_3	62	1.882782
gender	0	0.000000
total_call	0	0.000000
id	0	0.000000

```
[ ] missing_data(df_test)
```

	Total	Percent
total_call	0	0.0
room_count	0	0.0
price_monthly	0	0.0
size	0	0.0
poi_3	0	0.0
poi_2	0	0.0
poi_1	0	0.0
fac_8	0	0.0
fac_7	0	0.0
fac_6	0	0.0
fac_5	0	0.0
fac_4	0	0.0
fac_3	0	0.0
fac_2	0	0.0
fac_1	0	0.0
id	0	0.0

Facility

```
[ ] def fac_column(df):
        for i in range(1,9):
            i = str(i)
            df['fac_'+i] = df['fac_'+i].fillna(0)
            df['fac_'+i] = df['fac_'+i].astype('bool')
        return df
```

```
[ ] df_train = fac_column(df_train)
    df_test = fac_column(df_test)
```

Room Count

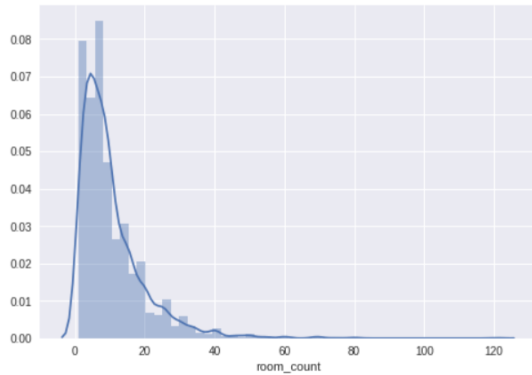
Untuk room count gunakan median

```
[ ]  sns.distplot(df_train['room_count'].dropna())
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py:6521: MatplotlibDeprecationWarning:

The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' instead.

<matplotlib.axes._subplots.AxesSubplot at 0x7fa9361c3128>
```




```
[ ]  df_train['room_count'] =  df_train['room_count'].fillna(df_train['room_count'].median())
```

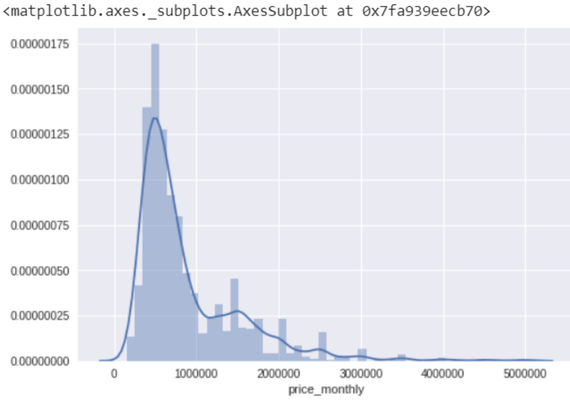
Price Monthly

```
[ ]  sns.distplot(df_train['price_monthly'].dropna())
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py:6521: MatplotlibDeprecationWarning:

The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' instead.

<matplotlib.axes._subplots.AxesSubplot at 0x7fa939eecb70>
```

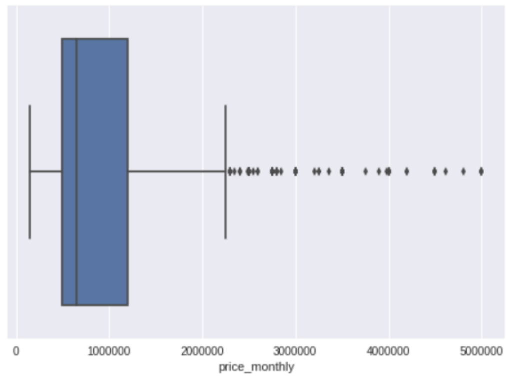


```
[ ] sns.boxplot(df_train['price_monthly'].dropna())
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:454: FutureWarning:

remove_na is deprecated and is a private function. Do not use.

<matplotlib.axes._subplots.AxesSubplot at 0x7fa9363e22e8>
```



Karena skew kita pakai median

```
[ ] df_train['price_monthly'] = df_train['price_monthly'].fillna(df_train['price_monthly'].median())
```

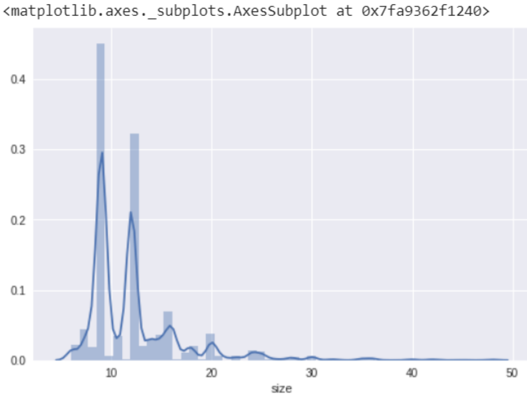
Size

```
[ ] sns.distplot(df_train['size'].dropna())
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py:6521: MatplotlibDeprecationWarning:

The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' instead.

<matplotlib.axes._subplots.AxesSubplot at 0x7fa9362f1240>
```




```
[ ] df_train['size'] = df_train['size'].fillna(df_train['size'].median())
```

POI

```
[ ] df_train['jumlah_nan'] = df_train['poi_1'].isnull().astype(int) + df_train['poi_2'].isnull().astype(int) + df_train['poi_3'].isnull().astype(int)
```

```
[ ] df_train = df_train[df_train['jumlah_nan'] < 2]
    df_train.sample()
```

	id	fac_1	fac_2	fac_3	fac_4	fac_5	fac_6	fac_7	fac_8	poi_1	poi_2	poi_3	size	price_monthly	room_count	total_call	gen
1625	1626	False	True	False	False	True	True	False	True	19617.0	9692.0	20519.0	9.0	5000000.0	3.0	3	carr

```
[ ] full_poi = df_train[df_train['poi_1'].notnull()]
    full_poi = full_poi[full_poi['poi_2'].notnull()]
    full_poi = full_poi[full_poi['poi_3'].notnull()]
    full_poi = full_poi[['poi_1','poi_2','poi_3']]
```

```
[ ] full_poi.head()
```

	poi_1	poi_2	poi_3
0	1778.0	10038.0	4106.0
1	4548.0	9332.0	6867.0
2	5174.0	9021.0	3693.0
3	1490.0	8954.0	2139.0
4	1688.0	8851.0	2145.0

```
[ ] lr_poi1 = LinearRegression(normalize=True)
    lr_poi2 = LinearRegression(normalize=True)
    lr_poi3 = LinearRegression(normalize=True)
```

```
[ ] lr_poi1.fit(full_poi.drop('poi_1',axis=1),full_poi['poi_1'])
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
```

```
[ ] lr_poi2.fit(full_poi.drop('poi_2',axis=1),full_poi['poi_2'])
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
```

```
[ ] lr_poi3.fit(full_poi.drop('poi_3',axis=1),full_poi['poi_3'])
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
```

```
[ ] missing_poi1 = df_train[df_train['poi_1'].isnull()][['poi_2','poi_3']]
```

```
[ ] missing_poi1.head()
```

	poi_2	poi_3
69	9851.0	1513.0
151	822.0	10024.0
251	11379.0	6884.0
342	6994.0	4039.0
420	8404.0	2668.0

```
[ ] poi1_preds = lr_poi1.predict(missing_poi1)
```

```
[ ] missing_poi2 = df_train[df_train['poi_2'].isnull()][['poi_1','poi_3']]
```

```
[ ] poi2_preds = lr_poi2.predict(missing_poi2)
```

```
[ ] missing_poi3 = df_train[df_train['poi_3'].isnull()][['poi_1','poi_2']]
```

```
[ ] poi3_preds = lr_poi3.predict(missing_poi3)
```

```
[ ] df_train[df_train['poi_1'].isnull()]['poi_1'] = poi1_preds
```

```
[ ] poi1_preds
```

```
array([ 1639.38707029,  8554.50116334,  6574.29058746,  3677.73252809,
        2560.29487943,  2382.1385886 ,  9552.26699894, 11635.7403325 ,
        1944.39096787,  3511.81182276,  6022.35158162,  5553.43380921,
        7217.41094853,  3512.26296785,  4373.58594207,  1838.41337716,
        2774.53470273,  4517.68712238,  2529.7796723 ,  6315.07611387,
        4261.86703267,  7198.33181781,  5020.70420928,  3422.75107676,
        6347.54224887,  1649.5351259 ,  3332.518466  ,  2649.12339716,
        1322.0498376 ,  3467.86050289, 10269.85696534,  4397.75365406,
       12349.70258189,  5015.41724098,  4995.87511923,  6248.43658898,
        7634.19434928,  1693.8607546 ,  2764.37866429,  4807.60990208,
        6468.12794407,  2682.80526052,   714.26668487,  2470.08443414,
        5082.13861332,  3943.99118164,  5194.5758219 ,  2912.44448093,
        1499.25046281,  3723.5935232 ,  5123.91025922,  3948.33791913,
        6094.77775434,  2464.88587378,  2841.27439772,  3448.23617304,
        2993.93253478,  8223.50386961,  4211.77129581,  2076.77840767,
        5082.16160679,  4546.48757943,  3694.78567765,  8529.92298487])
```

```
[ ] df_train.loc[df_train['poi_1'].isnull(),'poi_1'] = poi1_preds
```

```
[ ] df_train.loc[df_train['poi_2'].isnull(),'poi_2'] = poi2_preds
```

```
[ ] df_train.loc[df_train['poi_3'].isnull(),'poi_3'] = poi3_preds
```

```
[ ] missing_data(df_train)
```

	Total	Percent
jumlah_nan	0	0.0
gender	0	0.0
fac_1	0	0.0
fac_2	0	0.0
fac_3	0	0.0
fac_4	0	0.0
fac_5	0	0.0
fac_6	0	0.0
fac_7	0	0.0
fac_8	0	0.0
poi_1	0	0.0
poi_2	0	0.0
poi_3	0	0.0
size	0	0.0
price_monthly	0	0.0
room_count	0	0.0
total_call	0	0.0
id	0	0.0

Feature Engineering

Jumlah Fasilitas

```
[ ] df_train['jml_fac'] = df_train[['fac_'+str(i) for i in range(1,9)]].agg('sum',axis=1).astype('float')
    df_test['jml_fac'] = df_test[['fac_'+str(i) for i in range(1,9)]].agg('sum',axis=1).astype('float')
```

rata-rata harga dalam jumlah dasilitas yang sama

```
[ ] df_train['rata_harga'] = df_train.groupby('jml_fac')['price_monthly'].transform('median')
    df_test['rata_harga'] = df_test.groupby('jml_fac')['price_monthly'].transform('median')
```

```
[ ] df_train.head()
```

	id	fac_1	fac_2	fac_3	fac_4	fac_5	fac_6	fac_7	fac_8	poi_1	poi_2	poi_3	size	price_monthly	room_count	total_call	gender	j
0	1	True	True	True	True	True	True	False	True	1778.0	10038.0	4106.0	9.00	1500000.0	6.0	72	campur	
1	2	True	True	False	True	True	True	False	False	4548.0	9332.0	6867.0	12.00	1500000.0	30.0	56	campur	
2	3	True	False	True	True	True	True	False	True	5174.0	9021.0	3693.0	12.00	1600000.0	20.0	109	campur	
3	4	True	True	True	True	True	True	False	False	1490.0	8954.0	2139.0	8.25	1500000.0	15.0	54	campur	
4	5	True	True	False	True	True	True	False	True	1688.0	8851.0	2145.0	14.85	2100000.0	10.0	19	campur	

```
[ ] df_test.head()
```

	id	fac_1	fac_2	fac_3	fac_4	fac_5	fac_6	fac_7	fac_8	poi_1	poi_2	poi_3	size	price_monthly	room_count	total_call	jml_fac	r
0	3294	False	True	False	False	False	False	False	True	2634	8854	2007	21.0	700000	1	2	2.0	
1	3295	True	True	True	True	True	True	False	True	6569	3512	7341	12.0	1200000	16	30	7.0	
2	3296	False	True	True	True	True	False	False	True	10623	883	11250	12.0	700000	8	5	5.0	
3	3297	False	True	True	True	True	False	False	True	10592	876	11216	12.0	700000	16	6	5.0	
4	3298	False	True	False	False	True	False	False	True	1623	10204	3931	13.6	700000	6	39	3.0	

Total Distance

```
[ ] df_train['total_jarak'] = df_train[['poi_'+str(i) for i in range(1,4)]].agg('sum',axis=1)
    df_test['total_jarak'] = df_test[['poi_'+str(i) for i in range(1,4)]].agg('sum',axis=1)
```

Premium

```
[ ] df_train['premium'] = df_train['price_monthly'] > 2000000
    df_test['premium'] = df_test['price_monthly'] > 2000000
```

Wanted

```
[ ] df_train['wanted'] = df_train['total_call'] > 100
    df_test['wanted'] = df_test['total_call'] > 100
```

Categorization

Harga

```
[ ] df_train['gender'] = df_train['gender'].map({'putra' : 1, 'putri' : 2, 'campur' : 3})
```

```
[ ] df_train['price_q'] = pd.qcut(df_train['price_monthly'], 3)
```

```
[ ] df_train['price_q'].dtypes
```

```
CategoricalDtype(categories=[(154999.999, 500000.0], (500000.0, 850000.0], (850000.0, 5000000.0]]
              ordered=True)
```

```
df_train[['price_q', 'gender']].groupby(['price_q'], as_index=False).mean().sort_values(by='price_q', ascending=True)
```

	price_q	gender
0	(154999.999, 500000.0]	1.701058
1	(500000.0, 850000.0]	1.848512
2	(850000.0, 5000000.0]	2.153920

```
def col_price(row):
    if row['price_monthly'] <= 500000:
        return 0
    if row['price_monthly'] <= 850000:
        return 1
    return 2
```

```
all_data = [df_train, df_test]
for dataset in all_data:
    dataset['cat_price'] = dataset.apply (lambda row: col_price(row), axis=1)
    dataset['cat_price'] = dataset['cat_price'].astype('category')
```

```
df_train['cat_price'].value_counts()
```

```
0    1134
1    1109
2    1046
Name: cat_price, dtype: int64
```

Size

```
df_train['size_q'] = pd.qcut(df_train['size'], 3)
```

```
df_train['size_q'].dtype
```

```
CategoricalDtype(categories=[(5.999, 9.0], (9.0, 12.0], (12.0, 48.0]]
                 ordered=True)
```

```
df_train[['size_q', 'gender']].groupby(['size_q'], as_index=False).mean().sort_values(by='size_q', ascending=True)
```

	size_q	gender
0	(5.999, 9.0]	1.776471
1	(9.0, 12.0]	1.912109
2	(12.0, 48.0]	2.081707

```
def col_size(row):
    if row['size'] <= 9:
        return 0
    if row['size'] <= 12:
        return 1
#     if row['price_monthly'] <=
    return 2
```

```
all_data = [df_train, df_test]
for dataset in all_data:
    dataset['cat_size'] = dataset.apply (lambda row: col_size(row), axis=1)
    dataset['cat_size'] = dataset['cat_size'].astype('category')
```

```
df_train['cat_size'].value_counts()
```

```
0    1445
1    1024
2     820
Name: cat_size, dtype: int64
```

Jarak

```
[ ] df_train['jarak_q'] = pd.qcut(df_train['total_jarak'], 3)
```

```
[ ] df_train['jarak_q'].dtype
```

```
CategoricalDtype(categories=[(10448.911, 14949.0], (14949.0, 20375.0], (20375.0, 150297.0]]
              ordered=True)
```

```
[ ] df_train[['jarak_q', 'gender']].groupby(['jarak_q'], as_index=False).mean().sort_values(by='jarak_q', ascending=True)
```

	jarak_q	gender
0	(10448.911, 14949.0]	1.862352
1	(14949.0, 20375.0]	1.888686
2	(20375.0, 150297.0]	1.933394

```
[ ] def col_jarak(row):
        if row['total_jarak'] <= 14949.0:
            return 0
        if row['total_jarak'] <= 20375.0:
            return 1
        return 2
```

```
[ ] all_data = [df_train, df_test]
    for dataset in all_data:
        dataset['cat_jarak'] = dataset.apply (lambda row: col_jarak(row), axis=1)
        dataset['cat_jarak'] = dataset['cat_jarak'].astype('category')
```

```
[ ] df_train['cat_jarak'].value_counts()
```

```
0    1097
2    1096
1    1096
Name: cat_jarak, dtype: int64
```

POI

```
[ ] df_train['poi_1_q'] = pd.qcut(df_train['poi_1'], 2)
```

```
[ ] df_train['poi_1_q'].dtypes
```

```
CategoricalDtype(categories=[(518.999, 3961.0], (3961.0, 48675.0]]
              ordered=True)
```

```
[ ] df_train[['poi_1_q', 'gender']].groupby(['poi_1_q'], as_index=False).mean().sort_values(by='poi_1_q', ascending=True)
```

	poi_1_q	gender
0	(518.999, 3961.0]	1.893074
1	(3961.0, 48675.0]	1.896531

```
[ ] df_train.head()
```

	fac_1	fac_2	fac_3	fac_4	fac_5	fac_6	fac_8	poi_1	poi_2	poi_3	...	rata_harga	total_jarak	premium	wanted	cat_price	cat_siz
0	True	True	True	True	True	True	True	1778.0	10038.0	4106.0	...	1700000.0	15922.0	False	False	2	
1	True	True	False	True	True	True	False	4548.0	9332.0	6867.0	...	918500.0	20747.0	False	False	2	
2	True	False	True	True	True	True	True	5174.0	9021.0	3693.0	...	1425000.0	17888.0	False	True	2	
3	True	True	True	True	True	True	False	1490.0	8954.0	2139.0	...	1425000.0	12583.0	False	False	2	
4	True	True	False	True	True	True	True	1688.0	8851.0	2145.0	...	1425000.0	12684.0	True	False	2	

5 rows × 26 columns

```
[ ]  all_data = [df_train, df_test]
```

```
[ ]  for dataset in all_data:
         dataset['cat_poi_1'] = dataset['poi_1'] > 3961
         print(dataset['cat_poi_1'].head())
         dataset['cat_poi_1'] = dataset['cat_poi_1'].astype(bool)
```

```
0    False
1     True
2     True
3    False
4    False
Name: cat_poi_1, dtype: bool
0    False
1     True
2     True
3     True
4    False
Name: cat_poi_1, dtype: bool
```

```
[ ]  df_train['poi_1'].head()
```

```
0    1778.0
1    4548.0
2    5174.0
3    1490.0
4    1688.0
Name: poi_1, dtype: float64
```

```
[ ]  df_test['poi_1'].head()
```

```
0     2634
1     6569
2    10623
3    10592
4     1623
Name: poi_1, dtype: int64
```

```
[ ]  df_train = df_train.drop(['poi_1_q'], axis=1)
```

```
[ ]  df_train['poi_2_q'] = pd.qcut(df_train['poi_2'], 2)
```

```
[ ]  df_train[['poi_2_q', 'gender']].groupby(['poi_2_q'], as_index=False).mean().sort_values(by='poi_2_q', ascending=True)
```

	poi_2_q	gender
0	(167.999, 9249.0]	1.861398
1	(9249.0, 55105.0]	1.928224

```
[ ]  all_data = [df_train, df_test]
     for dataset in all_data:
         dataset['cat_poi_2'] = dataset['poi_2'] > 9249
         print(dataset['cat_poi_2'].head())
         dataset['cat_poi_2'] = dataset['cat_poi_2'].astype(bool)
```

```
0     True
1     True
2    False
3    False
4    False
Name: cat_poi_2, dtype: bool
0    False
1    False
2    False
3    False
4     True
Name: cat_poi_2, dtype: bool
```

```
[ ] df_test.describe()
```

	id	poi_1	poi_2	poi_3	size	price_monthly	room_count	total_call	jml_fac	rata_harga	to
count	824.000000	824.000000	824.000000	824.000000	824.000000	8.240000e+02	824.000000	824.000000	824.000000	8.240000e+02	8
mean	3705.500000	4542.400485	9971.084951	4629.087379	12.056274	8.915922e+05	9.436893	39.989078	3.654126	7.819175e+05	191
std	238.012605	3175.305908	4261.309222	3224.638561	4.621858	5.796271e+05	9.327230	53.785666	2.009826	3.421051e+05	82
min	3294.000000	568.000000	306.000000	286.000000	6.000000	2.500000e+05	1.000000	1.000000	0.000000	4.000000e+05	124
25%	3499.750000	2340.500000	7864.750000	2570.250000	9.000000	5.000000e+05	4.000000	9.000000	2.000000	5.500000e+05	141
50%	3705.500000	4003.500000	9441.000000	3899.500000	12.000000	7.000000e+05	7.000000	21.000000	4.000000	7.000000e+05	167
75%	3911.250000	5759.500000	12151.500000	5867.250000	12.000000	1.200000e+06	12.000000	46.000000	5.000000	8.000000e+05	214
max	4117.000000	33290.000000	37721.000000	30963.000000	42.000000	5.000000e+06	100.000000	623.000000	7.000000	1.500000e+06	1019

```
[ ] df_train['poi_3_q'] = pd.qcut(df_train['poi_3'], 2)
```

```
[ ] df_train[['poi_3_q', 'gender']].groupby(['poi_3_q'], as_index=False).mean().sort_values(by='poi_3_q', ascending=True)
```

	poi_3_q	gender
0	(323.999, 3930.0]	1.871733
1	(3930.0, 46517.0]	1.917883

```
[ ] all_data = [df_train, df_test]
    for dataset in all_data:
        dataset['cat_poi_3'] = dataset['poi_3'] > 3930
        print(dataset['cat_poi_3'].head())
        dataset['cat_poi_3'] = dataset['cat_poi_3'].astype(bool)
```

```
0     True
1     True
2    False
3    False
4    False
Name: cat_poi_3, dtype: bool
0    False
1     True
2     True
3     True
4     True
Name: cat_poi_3, dtype: bool
```

```
[ ] df_train = df_train.drop(['poi_2_q', 'poi_3_q'], axis=1)
```

```
[ ] df_train = df_train.drop(['jarak_q', 'price_q','size_q'], axis=1)
```

```
[ ] cat_columns = df_train.select_dtypes('category').columns
```

```
[ ] cat_columns
```

```
Index(['cat_price', 'cat_size', 'cat_jarak'], dtype='object')
```

```
[ ] df_train.drop(['id','jumlah_nan','fac_7'],axis=1,inplace=True)
    df_test.drop(['id','fac_7'],axis=1,inplace=True)
```

Feature Selection

```
[ ] X =df_train.drop(['gender'],axis=1)
    y = df_train['gender']
```

```
[ ] X[X.select_dtypes('number').columns] = StandardScaler().fit_transform(X[X.select_dtypes('number').columns])
    df_test[df_test.select_dtypes('number').columns] = StandardScaler().fit_transform(df_test[df_test.select_dtypes('number').columns])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/data.py:645: DataConversionWarning:

Data with input dtype int64, float64 were all converted to float64 by StandardScaler.

/usr/local/lib/python3.6/dist-packages/sklearn/base.py:464: DataConversionWarning:

Data with input dtype int64, float64 were all converted to float64 by StandardScaler.

/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/data.py:645: DataConversionWarning:

Data with input dtype int64, float64 were all converted to float64 by StandardScaler.

/usr/local/lib/python3.6/dist-packages/sklearn/base.py:464: DataConversionWarning:

Data with input dtype int64, float64 were all converted to float64 by StandardScaler.
```

```
[ ] for i in cat_columns:
        X[i], df_test[i] = target_encode(trn_series=X[i],tst_series=df_test[i],target=df_train['gender'])
```

Modelling

```
[ ] from sklearn.ensemble import VotingClassifier
    from sklearn.metrics import accuracy_score
```

```
[ ] eclf = VotingClassifier(estimators=[
        ('xgb',xgb_model), ('lgb',lgb_model), ('gb',gb)], voting='soft', weights=[1,2,1])

    cv = KFold(n_splits=10, shuffle=True)
    scores_eclf = []
    for train_index,test_index in cv.split(X,y):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        eclf.fit(X_train, y_train)
        y_predeclf = eclf.predict(X_test)
        scores_eclf.append(accuracy_score(y_predeclf, y_test))
```

```
[ ] eclf.fit(X,y)
```

```
VotingClassifier(estimators=[('xgb', XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
        colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
        max_depth=3, min_child_weight=1, missing=None, n_estimators=320,
        n_jobs=1, nthread=None, objective='multi:softprob', random_...    subsample=0.85, tol=0.0001, validation_fraction=0.1,
                verbose=0, warm_start=True))],
          flatten_transform=None, n_jobs=None, voting='soft',
          weights=[1, 2, 1])
```

Submission

```
[ ] df_submission = pd.read_csv('Sample_submission.csv')
```

```
[ ] final_pred = eclf.predict(df_test)
```

```
[ ] df_submission['gender'] = final_pred
```

```
[ ] df_submission['gender'] = df_submission['gender'].map({1 : 'putra',2 : 'putri', 3:'campur'})
```

```
[ ] df_submission.to_csv('Submission_19.csv',index=False)
```