BINAR

[DS1-12] - Challenge Chapter 2

Data Science

Anggota Tim

TIM	NAMA
DS1-12	Taufiq Qurohman Ruki
	Aleisya Zahari Salam

Latar Belakang

Telekomunikasi menjadi salah satu pilar utama dalam menghubungkan jutaan orang di seluruh dunia melalui teknologi informasi yang terus berkembang.

Peningkatan permintaan akan koneksi internet yang cepat dan andal, didorong oleh inovasi teknologi seperti 5G dan infrastruktur serat optik, telah memicu persaingan yang semakin ketat antara perusahaan telekomunikasi dan penyedia layanan internet (ISP)

Perubahan pola konsumen telekomunikasi yang terus berubah

Tujuan

Tujuan dari masalah ini adalah untuk mengidentifikasi pelanggan yang kemungkinan akan beralih menggunakan layanan komunikasi melalui klasifikasi dan prediksi.

Dataset

1. **state**: negara bagian atau wilayah geografis di mana pelanggan berada.
2. **account_length**: Panjang waktu sejak akun pelanggan dibuat atau aktif.
3. **area_code**: Kode area telepon yang terkait dengan lokasi geografis pelanggan.
4. **international_plan**: Variabel biner yang menunjukkan apakah pelanggan memiliki atau tidak memiliki paket atau layanan internasional tambahan.
5. **voice_mail_plan**: Variabel biner yang menunjukkan apakah pelanggan memiliki atau tidak memiliki layanan kotak pesan suara.
6. **number_vmail_messages**: Jumlah pesan suara yang diterima oleh pelanggan.
7. **total_day_minutes**: Total jumlah menit yang digunakan pelanggan selama jam-jam siang hari.
8. **total_day_calls**: Total jumlah panggilan yang dilakukan pelanggan selama jam-jam siang hari.
9. **total_day_charge**: Total biaya yang dibebankan kepada pelanggan untuk penggunaan layanan selama jam siang.
10. **total_eve_minutes**: Total jumlah menit yang digunakan pelanggan selama jam-jam sore atau malam.
11. **total_eve_calls**: Total jumlah panggilan yang dilakukan pelanggan selama jam-jam sore atau malam.
12. **total_eve_charge**: Total biaya yang dibebankan kepada pelanggan untuk penggunaan layanan selama jam sore atau malam.
13. **total_night_minutes**: Total jumlah menit yang digunakan pelanggan selama jam-jam malam.
14. **total_night_calls**: Total jumlah panggilan yang dilakukan pelanggan selama jam-jam malam.
15. **total_night_charge**: Total biaya yang dibebankan kepada pelanggan untuk penggunaan layanan selama jam malam.
16. **total_intl_minutes**: Total jumlah menit yang digunakan pelanggan untuk panggilan internasional.
17. **total_intl_calls**: Total jumlah panggilan internasional yang dilakukan oleh pelanggan.
18. **total_intl_charge**: Total biaya yang dibebankan kepada pelanggan untuk panggilan internasional.
19. **number_customer_service_calls**: Jumlah panggilan layanan pelanggan yang dilakukan oleh pelanggan.
20. **churn**: Variabel biner yang menunjukkan apakah pelanggan tetap (no) atau berpindah (yes) dari layanan tersebut.

Library

Import Library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import scipy.stats as stats
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
import altair as alt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

[1] ✓ 4.4s Python

Library yang dipakai dalam case study ini yaitu library python dan beberapa library dari scikit learn

Read Data

Berikut merupakan isi dari dataset yang akan digunakan

Read Dataset

```
df = pd.read_csv('Data_Train.csv')
df.head()
```

✓ 0.0s Python

l_eve_minutes	total_eve_calls	total_eve_charge	total_night_minutes	total_night_calls	total_night_charge	total_intl_minutes	total_intl_calls	total_intl_charge	number_customer_service_calls	churn
195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	no
121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	no
61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	no
148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	no
348.5	108	29.62	212.6	118	9.57	7.5	7	2.03	3	no

Exploratory Data Analysis (EDA)

Cek Info Dataset

Pertama yang dilakukan adalah dengan mengecek terkait dengan info dataset yang digunakan. Hasilnya ataset yang digunakan memiliki 4250 baris dan 20 kolom. Serta dengan infor dataset seperti yang ditunjukkan dibawah.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4250 entries, 0 to 4249
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   state                         4250 non-null   object
 1   account_length                4250 non-null   int64
 2   area_code                     4250 non-null   object
 3   international_plan            4250 non-null   object
 4   voice_mail_plan               4250 non-null   object
 5   number_vmail_messages         4250 non-null   int64
 6   total_day_minutes             4250 non-null   float64
 7   total_day_calls               4250 non-null   int64
 8   total_day_charge              4250 non-null   float64
 9   total_eve_minutes             4250 non-null   float64
 10  total_eve_calls               4250 non-null   int64
 11  total_eve_charge              4250 non-null   float64
 12  total_night_minutes           4250 non-null   float64
 13  total_night_calls             4250 non-null   int64
 14  total_night_charge            4250 non-null   float64
 15  total_intl_minutes            4250 non-null   float64
 16  total_intl_calls              4250 non-null   int64
 17  total_intl_charge             4250 non-null   float64
 18  number_customer_service_calls 4250 non-null   int64
 19  churn                         4250 non-null   object
dtypes: float64(8), int64(7), object(5)
memory usage: 664.2+ KB
```

```
baris, kolom = df.shape
print('Dataset ini terdiri dari: ')
print(f'{baris} baris')
print(f'{kolom} kolom')
```

```
Dataset ini terdiri dari:
4250 baris
20 kolom
```

Data Quality Check

Kemudian yang dilakukan selanjutnya adalah mengecek kondisi dari data yang digunakan. Hasilnya dataset yang digunakan tidak memiliki missing value maupun data duplikat.

1. Data Quality Check

```
list_items = []
for col in df.columns:
    list_items.append([col, df[col].dtype, df[col].isna().sum(), 100*df[col].isna().sum()/len(df[col]), df[col].nunique(), df[col].unique()[:5]])
desc_df = pd.DataFrame(data=list_items, columns= 'Feature, Data Type, Null, Null %, Unique, Unique Sample'.split(','))
desc_df
```

✓ 0.0s Python

	Feature	Data Type	Null	Null %	Unique	Unique Sample
0	state	object	0	0.0	51	[OH, NJ, OK, MA, MO]
1	account_length	int64	0	0.0	215	[107, 137, 84, 75, 121]
2	area_code	object	0	0.0	3	[area_code_415, area_code_408, area_code_510]
3	international_plan	object	0	0.0	2	[no, yes]
4	voice_mail_plan	object	0	0.0	2	[yes, no]
5	number_vmail_messages	int64	0	0.0	46	[26, 0, 24, 37, 27]
6	total_day_minutes	float64	0	0.0	1843	[161.6, 243.4, 299.4, 166.7, 218.2]
7	total_day_calls	int64	0	0.0	120	[123, 114, 71, 113, 88]
8	total_day_charge	float64	0	0.0	1843	[27.47, 41.38, 50.9, 28.34, 37.09]
9	total_eve_minutes	float64	0	0.0	1773	[195.5, 121.2, 61.9, 148.3, 348.5]
10	total_eve_calls	int64	0	0.0	123	[103, 110, 88, 122, 108]
11	total_eve_charge	float64	0	0.0	1572	[16.62, 10.3, 5.26, 12.61, 29.62]
12	total_night_minutes	float64	0	0.0	1757	[254.4, 162.6, 196.9, 186.9, 212.6]
13	total_night_calls	int64	0	0.0	128	[103, 104, 89, 121, 118]
14	total_night_charge	float64	0	0.0	992	[11.45, 7.32, 8.86, 8.41, 9.57]

```
# cek data duplicate
print('Jumlah data duplicate: ', df.duplicated().sum())
```

✓ 0.0s

```
Jumlah data duplicate:  0
```

Descriptive Statistic

Pada Dari data tersebut, rata-rata panjang akun pelanggan sekitar 100 hari. Mayoritas pelanggan tidak memiliki pesan voicemail. Penggunaan menit dan panggilan pada siang dan malam hari serupa, dengan rata-rata sekitar 180 menit dan 100 panggilan untuk siang hari, serta sekitar 200 menit dan panggilan untuk malam hari, dengan biaya siang hari sekitar $30. Penggunaan internasional rata-rata sekitar 10 menit dengan biaya panggilan sekitar $2.77. Rata-rata panggilan layanan pelanggan adalah sekitar 1.56. Data tersebut memiliki 4250 entri dengan 51 nilai unik untuk negara bagian, dengan 'WV' (West Virginia) menjadi yang paling umum (139 entri). Kode area '415' muncul paling sering (2108 entri). Mayoritas pelanggan tidak memiliki rencana internasional (3854 dari 4250) atau voicemail (3138 dari 4250). Sebagian besar pelanggan (3652 dari 4250) tetap berlangganan layanan.

2. Descriptive Statistic

```
df.describe().T
```

✓ 0.1s

	count	mean	std	min	25%	50%	75%	max
account_length	4250.0	100.236235	39.698401	1.0	73.0000	100.00	127.0000	243.00
number_vmail_messages	4250.0	7.631765	13.439882	0.0	0.0000	0.00	16.0000	52.00
total_day_minutes	4250.0	180.259600	54.012373	0.0	143.3250	180.45	216.2000	351.50
total_day_calls	4250.0	99.907294	19.850817	0.0	87.0000	100.00	113.0000	165.00
total_day_charge	4250.0	30.644682	9.182096	0.0	24.3650	30.68	36.7500	59.76
total_eve_minutes	4250.0	200.173906	50.249518	0.0	165.9250	200.70	233.7750	359.30
total_eve_calls	4250.0	100.176471	19.908591	0.0	87.0000	100.00	114.0000	170.00
total_eve_charge	4250.0	17.015012	4.271212	0.0	14.1025	17.06	19.8675	30.54
total_night_minutes	4250.0	200.527882	50.353548	0.0	167.2250	200.45	234.7000	395.00
total_night_calls	4250.0	99.839529	20.093220	0.0	86.0000	100.00	113.0000	175.00
total_night_charge	4250.0	9.023892	2.265922	0.0	7.5225	9.02	10.5600	17.77
total_intl_minutes	4250.0	10.256071	2.760102	0.0	8.5000	10.30	12.0000	20.00
total_intl_calls	4250.0	4.426353	2.463069	0.0	3.0000	4.00	6.0000	20.00
total_intl_charge	4250.0	2.769654	0.745204	0.0	2.3000	2.78	3.2400	5.40
number_customer_service_calls	4250.0	1.559059	1.311434	0.0	1.0000	1.00	2.0000	9.00

```
df.describe(exclude= np.number).T
```

✓ 0.0s

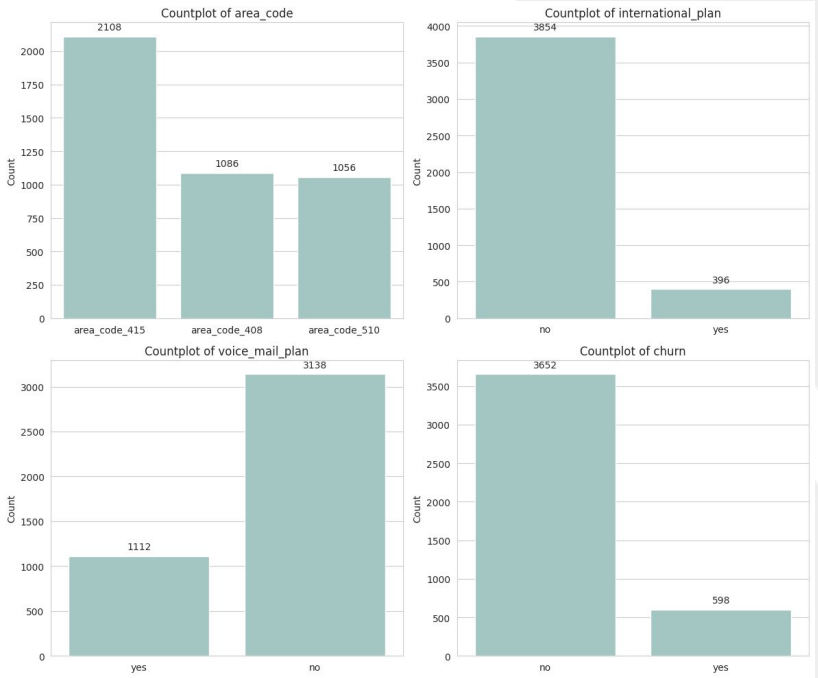
	count	unique	top	freq
state	4250	51	WV	139
area_code	4250	3	area_code_415	2108
international_plan	4250	2	no	3854
voice_mail_plan	4250	2	no	3138
churn	4250	2	no	3652

Univariate Analysis

Selanjutnya dilakukan univariate analysis. Univariate Analysis adalah proses statistik yang berkonsentrasi pada pemahaman dan interpretasi data dari satu variabel tunggal dalam suatu dataset. Hasilnya are_code_415 merupakan area paling banyak, pelanggan kebanyakan tidak memiliki layanan tambahan internasional, tidak memiliki pesan suara,dan tidak meninggalkan layanan.

Frequency Plot

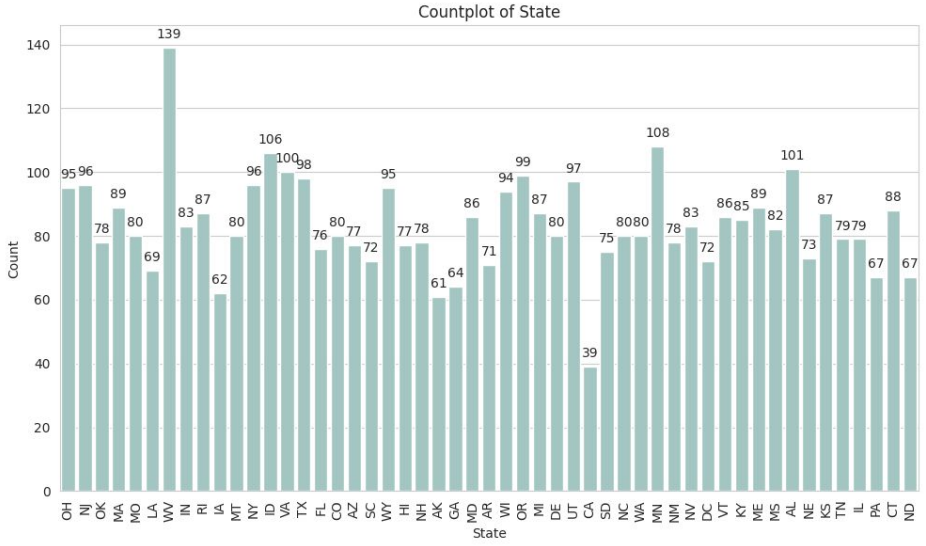
```
cats_1 = ['area_code', 'international_plan', 'voice_mail_plan', 'churn']

color = '#9eccc8'

sns.set_style("whitegrid")

fig, axes = plt.subplots(len(cats)//2, 2, figsize=(12, 10))

axes = axes.flatten()

for i, cat in enumerate(cats_1):
    sns.countplot(x=cat, data=df, ax=axes[i], color=color)
    axes[i].set_title(f'Countplot of {cat}')
    axes[i].set_xlabel('')
    axes[i].set_ylabel('Count')

    for p in axes[i].patches:
        axes[i].annotate(format(p.get_height(), '.0f'),
                         (p.get_x() + p.get_width() / 2., p.get_height()),
                         ha = 'center', va = 'center',
                         xytext = (0, 10),
                         textcoords = 'offset points')

plt.tight_layout()
plt.show()
```



Univariate Analysis

Insight: Wilayah WV menjadi wilayah paling banyak yang ada pada dataset.

```
color = '#9eccc8'

sns.set_style("whitegrid")

# Create countplot for the 'state' column
plt.figure(figsize=(10, 6))
sns.countplot(x='state', data=df, color=color)
plt.title('Countplot of State')
plt.xlabel('State')
plt.ylabel('Count')

# Show the count values on top of each bar
for p in plt.gca().patches:
    plt.gca().annotate(format(p.get_height(), '.0f'),
                       (p.get_x() + p.get_width() / 2., p.get_height()),
                       ha = 'center', va = 'center',
                       xytext = (0, 10),
                       textcoords = 'offset points')

# Rotate x labels for better readability
plt.xticks(rotation=90)

# Show plot
plt.tight_layout()
plt.show()
```

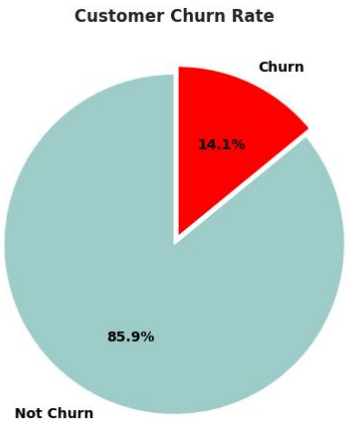


Univariate Analysis

Insight: **14.1%** dari **4250** pelanggan yang meninggalkan layanan. Hal ini perlu diperhatikan karena tingkat churn rate **yang dapat diterima sekitar 5% - 7%**

```python
churn_rate = df['churn'].value_counts()
explode = (0.05, 0)
text_props = {'color': 'black', 'weight': 'bold'}

plt.figure(figsize=(7, 5))
plt.pie(churn_rate, labels=['Not Churn', 'Churn'], autopct='%1.1f%%', startangle=90, explode = explode, colors = [ '#9eccc8', '#ff0000'], textprops=text_props)
plt.axis('equal')
plt.title('Customer Churn Rate', fontweight='bold', pad=20)
plt.show()
```

Python



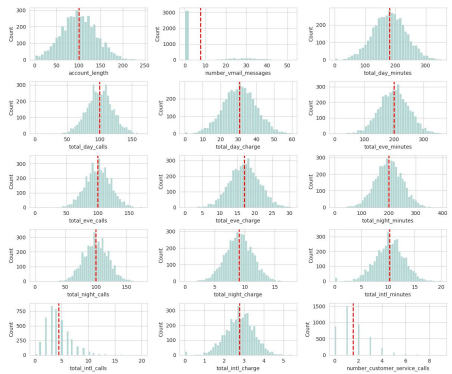
Univariate Analysis

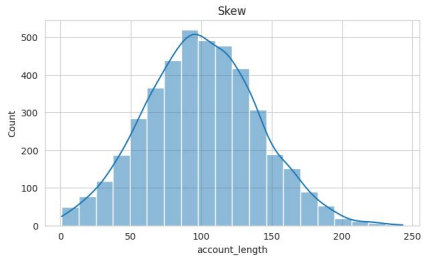
Insight, berdasarkan hasil cek outlier dan distribusi didapat beberapa fitur yang termasuk ke dalam:

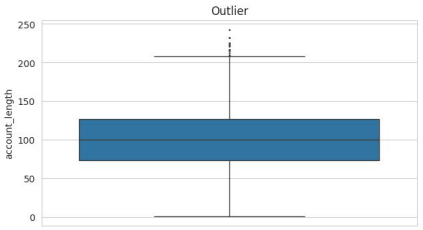
Kolom distribusi normal: ['total_day_minutes', 'total_day_charge', 'total_eve_minutes', 'total_eve_charge', 'total_night_minutes', 'total_night_calls', 'total_night_charge']

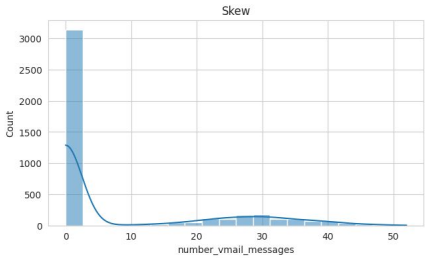
Kolom distribusi tidak normal: ['account_length', 'number_vmail_messages', 'total_day_calls', 'total_eve_calls', 'total_intl_minutes', 'total_intl_calls', 'total_intl_charge', 'number_customer_service_calls']

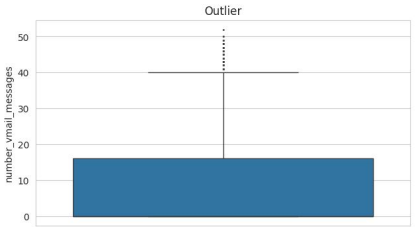
```
for col in nums:
    fig, ax = plt.subplots(1,2, figsize=(16, 4))
    sns.histplot(df[col], bins=20, kde=True, ax=ax[0])
    ax[0].set_title('Skew')
    sns.boxplot(df[col], fliersize=1, ax=ax[1])
    ax[1].set_title('Outlier')
```











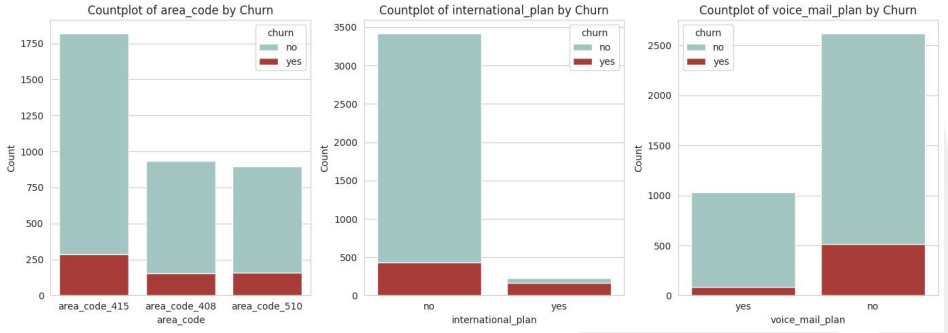
Bivariate Analysis

Kemudian dilakukan analisis bivariate. Analisis bivariat adalah jenis analisis statistik yang melibatkan hubungan antara dua variabel.

2.2 Bivariate Analysis

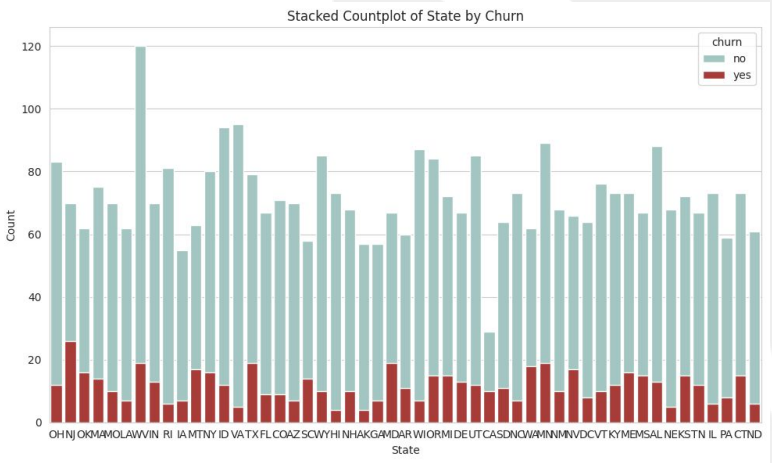
2.2.1 Categorical data

```
cats = ['area_code', 'international_plan', 'voice_mail_plan']

color = '#9eccc8'

sns.set_style("whitegrid")

fig, axes = plt.subplots(1, len(cats), figsize=(14, 5))

for i, cat in enumerate(cats):
    sns.countplot(x=cat, hue='churn', data=df, ax=axes[i], palette=['#9eccc8', '#b92a27'], dodge=False)
    axes[i].set_title(f'Countplot of {cat} by Churn')
    axes[i].set_xlabel(cat)
    axes[i].set_ylabel('Count')

plt.tight_layout()
plt.show()
```



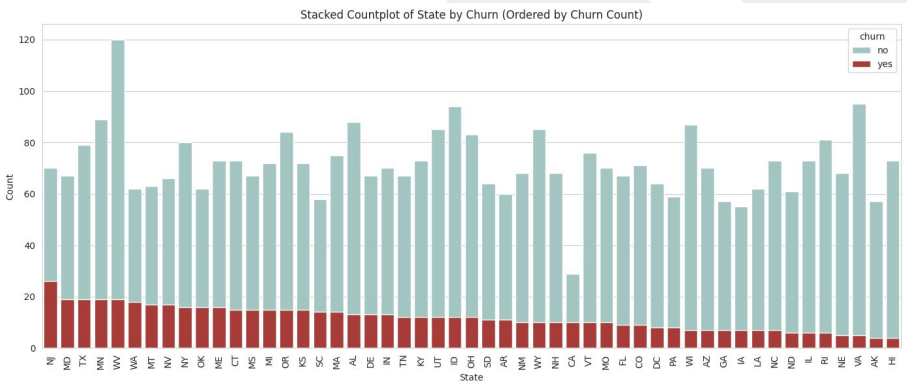
Bivariate Analysis

```
color = '#9eccc8'

sns.set_style("whitegrid")

# Create countplot for the 'state' column stacked by 'churn'
plt.figure(figsize=(10, 6))
sns.countplot(x='state', hue='churn', data=df, palette=['#9eccc8', '#b92a27'], dodge=False)
plt.title('Stacked Countplot of State by Churn')
plt.xlabel('State')
plt.ylabel('Count')

# Show plot
plt.tight_layout()
plt.show()
```



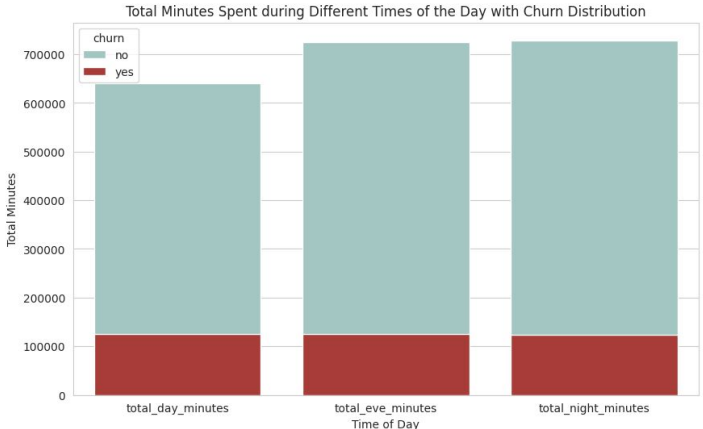
Bivariate Analysis

```
# Calculate churn count for each state
churn_counts = df[df['churn'] == 'yes']['state'].value_counts().sort_values(ascending=False)

# Select the states with highest churn counts
highest_churn_states = churn_counts.index

color = '#9eccc8'

sns.set_style("whitegrid")

# Create countplot for the 'state' column stacked by 'churn'
plt.figure(figsize=(14, 6))
sns.countplot(x='state', hue='churn', data=df[df['state'].isin(highest_churn_states)],
              palette=['#9eccc8', '#b92a27'], dodge=False, order=highest_churn_states)
plt.title('Stacked Countplot of State by Churn (Ordered by Churn Count)')
plt.xlabel('State')
plt.ylabel('Count')

# Show plot
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Bivariate Analysis

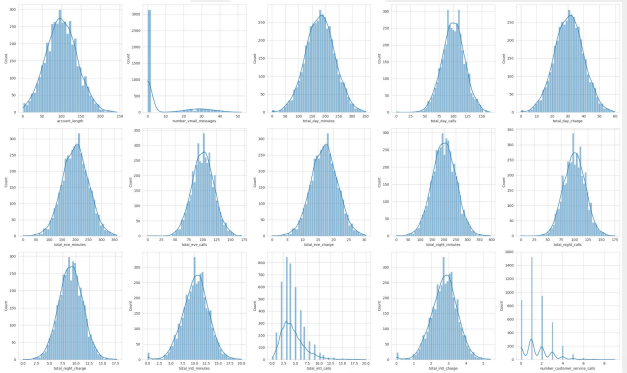
```
# Create a new DataFrame with total minutes spent during different times of the day
time_of_day_df = df[['total_day_minutes', 'total_eve_minutes', 'total_night_minutes', 'churn']].copy()


# Melt the DataFrame to plot total minutes against churn for each time period
time_of_day_melted = pd.melt(time_of_day_df, id_vars=['churn'], var_name='time_of_day', value_name='total_minutes')

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(x='time_of_day', y='total_minutes', hue='churn', data=time_of_day_melted, estimator=sum, palette=['#9eccc8', '#b92a27'], ci=None, dodge=False)
plt.title('Total Minutes Spent during Different Times of the Day with Churn Distribution')
plt.xlabel('Time of Day')
plt.ylabel('Total Minutes')
plt.show()
```



Bivariate Analysis

Berdasarkan hasil visualisasi data, cukup sekilas banyak yang memiliki distribusi normal, namun untuk dipastikan akan dilakukan tes distribusi normal.

```
plt.figure(figsize=(25, 15))
n=3
for i in range(0, len(nums)):
    plt.subplot(n, math.ceil(len(nums)/n), i+1)
    sns.histplot(df[nums[i]], kde=True,  palette=['#9eccc8'], multiple='stack')
    plt.tight_layout()
```



Cek Outlier

Tahap ini dilakukan pengecekan outlier, didapat bahwa 735 dari data termasuk outlier. Dari setiap kolom yang ada outlier yang dimiliki masing-masing kolom tidak lebih dari 10% data yang ada, oleh karena itu outlier tidak dihapus karena dianggap tidak banyak dan setiap informasi yang ada penting.

3. cek Outlier, cek Distribusi Normal, dan cek korelasi data

```
print(f'Total rows: {len(df)}')
Kolom_numerik1 = ['account_length', 'number_vmail_messages', 'total_day_minutes',
       'total_day_calls', 'total_day_charge', 'total_eve_minutes',
       'total_eve_calls', 'total_eve_charge', 'total_night_minutes',
       'total_night_calls', 'total_night_charge', 'total_intl_minutes',
       'total_intl_calls', 'total_intl_charge',
       'number_customer_service_calls']

outlier = []
no_outlier = []
is_outlier = []
low_lim = []
high_lim = []


filtered_entries = np.array([True] * len(df))
for col in Kolom_numerik1:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    low_limit = Q1 - (IQR * 1.5)
    high_limit = Q3 + (IQR * 1.5)

    #filter outlier
    filter_outlier = ((df[col] >= low_limit) & (df[col] <= high_limit))
    outlier.append(len(df[~filter_outlier]))
    no_outlier.append(len(df[filter_outlier]))
    is_outlier.append(df[col][~filter_outlier].any())
    low_lim.append(low_limit)
    high_lim.append(high_limit)
```

```
Total rows: 4250
Outlier All Data : 735
Not Outlier All Data : 3515
```

	Column Name	is Outlier	Lower Limit	Upper Limit	Outlier	No Outlier
0	account_length	True	-8.00000	208.00000	20	4230
1	number_vmail_messages	True	-24.00000	40.00000	86	4164
2	total_day_minutes	True	34.01250	325.51250	25	4225
3	total_day_calls	True	48.00000	152.00000	28	4222
4	total_day_charge	True	5.78750	55.32750	26	4224
5	total_eve_minutes	True	64.15000	335.55000	34	4216
6	total_eve_calls	True	46.50000	154.50000	24	4226
7	total_eve_charge	True	5.45500	28.51500	34	4216
8	total_night_minutes	True	66.01250	335.91250	37	4213
9	total_night_calls	True	45.50000	153.50000	33	4217
10	total_night_charge	True	2.96625	15.11625	37	4213
11	total_intl_minutes	True	3.25000	17.25000	62	4188
12	total_intl_calls	True	-1.50000	10.50000	100	4150
13	total_intl_charge	True	0.89000	4.65000	62	4188
14	number_customer_service_calls	True	-0.50000	3.50000	335	3915

Cek Distribusi

Pada tahap ini dilakukan pengecekan distribusi. Hasilnya:

Kolom distribusi normal: ['total_day_minutes', 'total_day_charge', 'total_eve_minutes', 'total_eve_charge', 'total_night_minutes', 'total_night_calls', 'total_night_charge']

Kolom distribusi tidak normal: ['account_length', 'number_vmail_messages', 'total_day_calls', 'total_eve_calls', 'total_intl_minutes', 'total_intl_calls', 'total_intl_charge', 'number_customer_service_calls']

```python
# Filter numeric columns from DataFrame
numeric_data = df[['account_length', 'number_vmail_messages', 'total_day_minutes',
       'total_day_calls', 'total_day_charge', 'total_eve_minutes',
       'total_eve_calls', 'total_eve_charge', 'total_night_minutes',
       'total_night_calls', 'total_night_charge', 'total_intl_minutes',
       'total_intl_calls', 'total_intl_charge',
       'number_customer_service_calls']]

# Perform Shapiro-Wilk test for normality on each numeric column
shapiro_results = {}
for col in numeric_data.columns:
    shapiro_results[col] = stats.shapiro(numeric_data[col])

# Output results
for col, result in shapiro_results.items():
    print(f"{col}: p-value = {result.pvalue}, Is Normally Distributed? {result.pvalue > 0.05}")
```

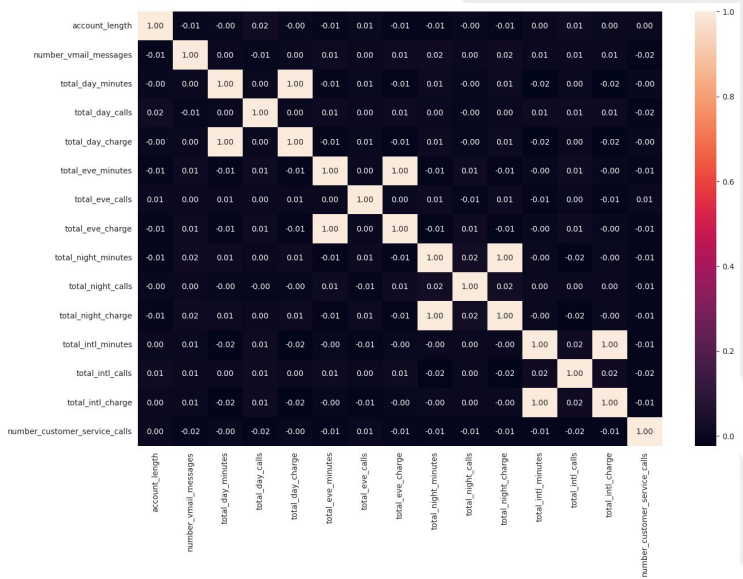
```
account_length: p-value = 2.3153396112854807e-05, Is Normally Distributed? False
number_vmail_messages: p-value = 2.321285267015352e-71, Is Normally Distributed? False
total_day_minutes: p-value = 0.5995237843892769, Is Normally Distributed? True
total_day_calls: p-value = 0.0002274792335780996, Is Normally Distributed? False
total_day_charge: p-value = 0.5997389394768184, Is Normally Distributed? True
total_eve_minutes: p-value = 0.5265584883881479, Is Normally Distributed? True
total_eve_calls: p-value = 0.010840760857371467, Is Normally Distributed? False
total_eve_charge: p-value = 0.5260544276501232, Is Normally Distributed? True
total_night_minutes: p-value = 0.771238136896553, Is Normally Distributed? True
total_night_calls: p-value = 0.053776641759196955, Is Normally Distributed? True
total_night_charge: p-value = 0.7644820217867112, Is Normally Distributed? True
total_intl_minutes: p-value = 1.8883615754859446e-13, Is Normally Distributed? False
total_intl_calls: p-value = 5.597402137065115e-46, Is Normally Distributed? False
total_intl_charge: p-value = 1.7742924929330753e-13, Is Normally Distributed? False
number_customer_service_calls: p-value = 7.9525235811473e-50, Is Normally Distributed? False
```

Cek korelasi

Tahap ini dilakukan pengecekan korelasi untuk setiap fitur numerik. Dan hasilnya seperti dibawah.

- korelasi tinggi dapat ditemukan antara total charge dan total minutes, oleh karena itu salah satu akan dihapus. kolom yang dihapus yaitu `total_day_minutes`, `total_eve_minutes`, `total_night_minutes`, dan `total_intl_minutes`.
- kolom `state` diputuskan akan di hapus karena memiliki banyak unique value yaitu 51, kolom ini tidak memiliki banyak informasi yang berguna.

```
# numerical
df_nums = df[nums]
plt.figure(figsize=(15,10))
sns.heatmap(df_nums.corr(), annot=True, fmt='.2f')
```



Chi-Square & P-Value

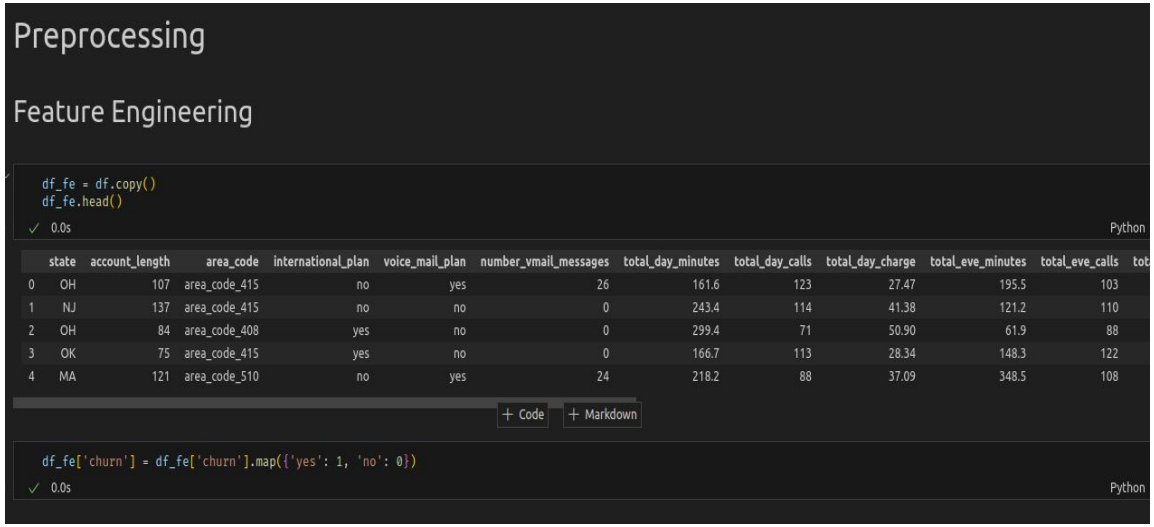
```
# categorical
from scipy.stats import chi2_contingency
cats2 = ['state', 'area_code', 'international_plan', 'voice_mail_plan']
chi2_array, p_array = [], []
for column in cats2:

    crosstab = pd.crosstab(df[column], df['churn'])
    chi2, p, dof, expected = chi2_contingency(crosstab)
    chi2_array.append(chi2)
    p_array.append(p)

df_chi = pd.DataFrame({
    'Variable': cats2,
    'Chi-square': chi2_array,
    'p-value': p_array
})
df_chi.sort_values(by='Chi-square', ascending=False)
```

	Variable	Chi-square	p-value
2	international_plan	282.653490	1.983190e-63
0	state	85.993673	1.169028e-03
3	voice_mail_plan	55.109814	1.139804e-13
1	area_code	1.216654	5.442606e-01

Pada tahap ini kami menerapkan chi-square dan p-value untuk melihat keterkaitan antar fitur kategorikal. Hasilnya area_code akan dihapus karena memiliki p-value yang tinggi, dalam artian tidak memiliki hubungan yang baik terhadap fitur lainnya.

Data Preprocessing

Feature Engineering



Preprocessing

Feature Engineering

```
df_fe = df.copy()
df_fe.head()
```

✓ 0.0s Python

	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_eve_calls	tot
0	OH	107	area_code_415	no	yes	26	161.6	123	27.47	195.5	103	
1	NJ	137	area_code_415	no	no	0	243.4	114	41.38	121.2	110	
2	OH	84	area_code_408	yes	no	0	299.4	71	50.90	61.9	88	
3	OK	75	area_code_415	yes	no	0	166.7	113	28.34	148.3	122	
4	MA	121	area_code_510	no	yes	24	218.2	88	37.09	348.5	108	

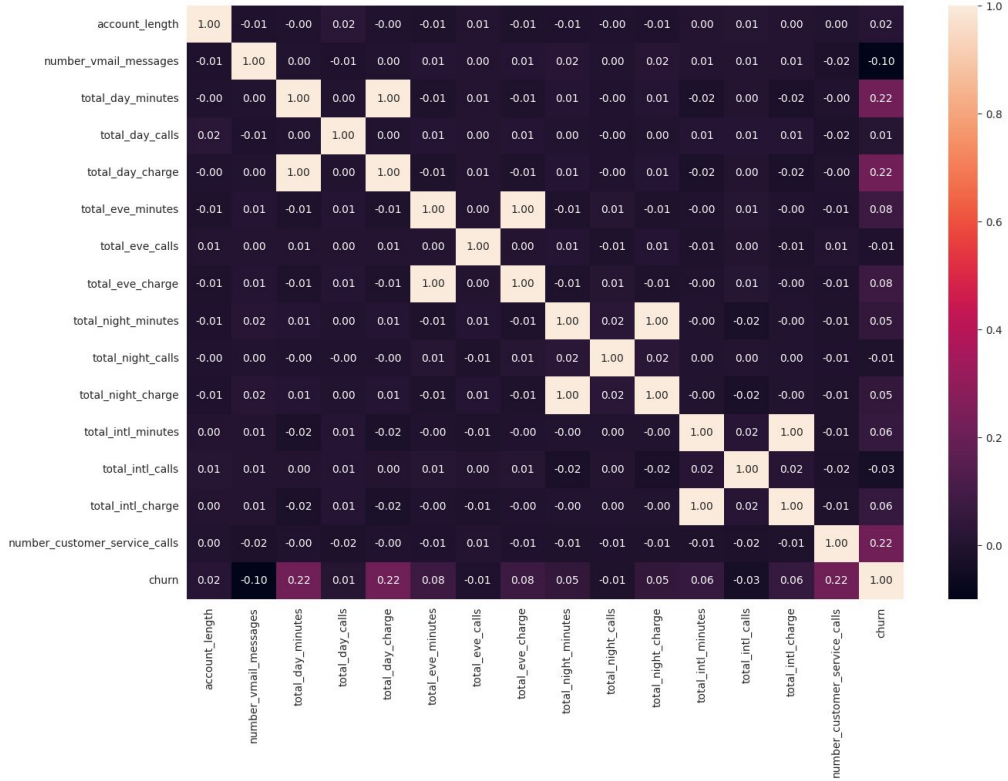
+ Code + Markdown

```
df_fe['churn'] = df_fe['churn'].map({'yes': 1, 'no': 0})
```

✓ 0.0s Python

Tahap awal yang dilakukan pada preprocessing yaitu dengan mengubah fitur churn dari kategorikal menjadi numerik menggunakan **map**..

Cek korelasi



- korelasi tinggi dapat ditemukan antara total charge dan total minutes, oleh karena itu salah satu akan dihapus. kolom yang dihapus yaitu `total_day_minutes`, `total_eve_minutes`, `total_night_minutes`, dan `total_intl_minutes`.
- kolom `state` diputuskan akan di hapus karena memiliki banyak unique value yaitu 51, kolom ini tidak memiliki banyak informasi yang berguna.

Hapus dan grup kolom

Tahap selanjutnya, menghapus beberapa kolom yang memiliki korelasi lemah berdasarkan analisis sebelumnya dan kemudian mengelompokkan fitur yang ada berdasarkan distribusi dan jenis tipe datanya..

```
# drop kolom yang tidak dipakai
df_fe = df_fe.drop(columns=['state', 'total_day_minutes', 'total_eve_minutes', 'total_night_minutes', 'total_intl_minutes'])
```

✓ 0.0s

```
nums2 = ['account_length', 'number_vmail_messages',
         'total_day_calls', 'total_day_charge',
         'total_eve_calls', 'total_eve_charge',
         'total_night_calls', 'total_night_charge',
         'total_intl_calls', 'total_intl_charge',
         'number_customer_service_calls']

normal1 = ['total_day_charge', 'total_eve_charge',
           'total_night_calls', 'total_night_charge']
non_normal1 = ['account_length', 'number_vmail_messages',
               'total_day_calls', 'total_eve_calls',
               'total_intl_calls',
               'total_intl_charge', 'number_customer_service_calls']

cats2 = ['area_code', 'international_plan', 'voice_mail_plan']
```

✓ 0.0s

Encoding dan Normalisasi data

```
# standar scaler
scaler = StandardScaler()

# transform data non normal
df_fe[non_normal1] = scaler.fit_transform(df_fe[non_normal1])

# minmax scaler
minmax = MinMaxScaler()

# transform data normal
df_fe[normal1] = minmax.fit_transform(df_fe[normal1])

# label encoder
label = LabelEncoder()

# label encoder untuk kolom churn
df_fe['churn'] = label.fit_transform(df_fe['churn'])
```

✓ 0.0s

```
# one hot encoder
ohe = OneHotEncoder()

# transform data kategorikal
ohe.fit(df_fe[cats2])

# transform data kategorikal
df_fe_ohe = ohe.transform(df_fe[cats2]).toarray()
df_fe_ohe = pd.DataFrame(df_fe_ohe, columns=ohe.get_feature_names_out(cats2))

# drop kolom kategorikal
df_fe = df_fe.drop(columns=cats2)
```

Setelah pemilihan fitur dilakukan,selanjutnya melakukan encoding dan normalisasi data terhadap data untuk mengubah data kategorikal menjadi numerik maupun agar datanya terstandarisasi. Teknik yang digunakan diantaranya **Standar Scaller, MinMax Scaller, Lebel Encoder, dan OneHotEncoder**

Split Dataset

Berikut hasil encoding dan selanjutnya akan dilakukan pembagian dataset menjadi data train dan test.

	account_length	number_vmail_messages	total_day_calls	total_day_charge	total_eve_calls	total_eve_charge	total_night_calls	total_night_charge	total_intl_calls	total_intl_charge	number_cus
0	0.170399	1.366857	1.163449	0.459672	0.141841	0.544204	0.588571	0.644344	-0.579164	1.248591	
1	0.926186	-0.567911	0.710014	0.692436	0.493490	0.337263	0.594286	0.411930	0.232927	0.698342	
2	-0.409038	-0.567911	-1.456398	0.851740	-0.611691	0.172233	0.508571	0.498593	1.045017	-1.328187	
3	-0.635774	-0.567911	0.659633	0.474230	1.096316	0.412901	0.691429	0.473270	-0.579164	-0.053219	
4	0.523099	1.218029	-0.599910	0.620649	0.393019	0.969876	0.674286	0.538548	1.045017	-0.992669	

```
# bagi data menjadi data train dan data test
X = df_fe.drop('churn', axis=1)
y = df_fe['churn']
```

✓ 0.0s Python

Smote

```
# bagi data menjadi data train dan data test
X = df_fe.drop('churn', axis=1)
y = df_fe['churn']
```

✓ 0.0s

```
# balance data dengan SMOTE
from imblearn.over_sampling import SMOTE

smote = SMOTE()
X_smote, y_smote = smote.fit_resample(X, y)

print(f'Jumlah data sebelum SMOTE: {len(X)}')
print(f'Jumlah data setelah SMOTE: {len(X_smote)}')
print(f'Jumlah data churn 0: {len(y_smote[y_smote == 0])}')
print(f'Jumlah data churn 1: {len(y_smote[y_smote == 1])}')
```

✓ 0.4s

```
Jumlah data sebelum SMOTE: 4250
Jumlah data setelah SMOTE: 7304
Jumlah data churn 0: 3652
Jumlah data churn 1: 3652
```

```
# bagi data menjadi data train dan data test
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=0.2, random_state=42)
```

✓ 0.0s

Dikarenakan datasetnya tidak balance, maka diterapkan teknik **smote** supaya datanya balance. Yang kemudian dilanjutkan dengan melakukan proses split data.

Data Modeling & Evaluation

Modeling

```
# logistic regression
logreg = LogisticRegression()

# fit model
logreg.fit(X_train, y_train)
```

✓ 0.2s

```
LogisticRegression()
```

```
# knn
knn = KNeighborsClassifier()

# fit model
knn.fit(X_train, y_train)
```

✓ 0.0s

```
KNeighborsClassifier()
```

```
# decision tree
tree = DecisionTreeClassifier()

# fit model
tree.fit(X_train, y_train)
```

✓ 0.1s

```
DecisionTreeClassifier()
```

```
# svm
svm = SVC()

# fit model
svm.fit(X_train, y_train)
```

✓ 2.7s

```
SVC()
```

Algoritma yang coba kami pakai pada tahap permodelan ini yaitu **Logistic Regression, KNN, Decision Tree, dan SVM**

Tes Prediksi

```
# predict
y_pred_logreg = logreg.predict(X_test)
y_pred_knn = knn.predict(X_test)
y_pred_tree = tree.predict(X_test)
y_pred_svm = svm.predict(X_test)
```

✓ 1.1s

```
# akurasi
print('Akurasi Logistic Regression: ', accuracy_score(y_test, y_pred_logreg))
print('Akurasi KNN: ', accuracy_score(y_test, y_pred_knn))
print('Akurasi Decision Tree: ', accuracy_score(y_test, y_pred_tree))
print('Akurasi SVM: ', accuracy_score(y_test, y_pred_svm))
```

✓ 0.0s

```
Akurasi Logistic Regression:  0.7994524298425736
Akurasi KNN:  0.8459958932238193
Akurasi Decision Tree:  0.9041752224503764
Akurasi SVM:  0.8829568788501027
```

Setelah model didefinisikan dan dilatih, kemudian dilakukan tahap uji coba terhadap data test yang sudah di split sebelumnya.

Evalusai Model

```
print('Classification Report SVM: ')
print(classification_report(y_test, y_pred_svm))
```

✓ 0.0s

```
Classification Report KNN:
              precision    recall  f1-score   support

           0       0.96      0.73      0.83       758
           1       0.77      0.97      0.86       703

    accuracy                           0.85      1461
   macro avg       0.87      0.85      0.84      1461
weighted avg       0.87      0.85      0.84      1461

Classification Report Decision Tree:
              precision    recall  f1-score   support

           0       0.92      0.89      0.91       758
           1       0.89      0.92      0.90       703

    accuracy                           0.90      1461
   macro avg       0.90      0.90      0.90      1461
weighted avg       0.90      0.90      0.90      1461

Classification Report SVM:
              precision    recall  f1-score   support

           0       0.91      0.86      0.88       758
           1       0.86      0.91      0.88       703

    accuracy                           0.88      1461
   macro avg       0.88      0.88      0.88      1461
weighted avg       0.88      0.88      0.88      1461
```

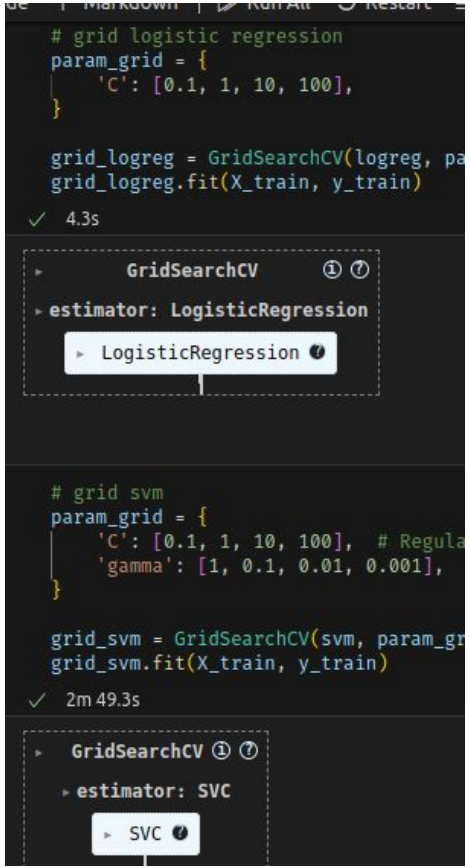
```
# Evaluasi logistic regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

print('Mean Absolute Error Logistic Regression: {:.2f}'.format(mean_absolute_error(y_test, y_pred_logreg)))
print('Mean Squared Error Logistic Regression: {:.2f}'.format(mean_squared_error(y_test, y_pred_logreg)))
print('Root Mean Squared Error Logistic Regression: {:.2f}'.format(np.sqrt(mean_squared_error(y_test, y_pred_logreg))))
```

✓ 0.0s

```
Mean Absolute Error Logistic Regression: 0.20
Mean Squared Error Logistic Regression: 0.20
Root Mean Squared Error Logistic Regression: 0.45
```

Selanjutnya dilakukan tahap evaluasi terhadap model dengan menggunakan Classification Report untuk model klasifikasi dan MEA,MSE, RMSE untuk prediksi.

Tunning Model



Untuk memperbagus model, maka dilakukan tunning terhadap model yang telah dilatih sebelumnya dengan menggunakan GridSearch.

Hasil Tunning Model

Predict using estimator

```
best_knn = grid_knn.best_estimator_
best_tree = grid_tree.best_estimator_
best_logreg = grid_logreg.best_estimator_
best_svm = grid_svm.best_estimator_
```

✓ 0.0s

```
# KNN Tunning
knn_tun = best_knn.predict(X_test)
print('Akurasi KNN Tunning: ', accuracy_score(y_test, knn_tun))
print('Classification Report KNN Tunning: ')
print(classification_report(y_test, knn_tun))
```

✓ 0.4s

```
Akurasi KNN Tunning:  0.8863791923340179
Classification Report KNN Tunning:
              precision    recall  f1-score   support

           0       0.98      0.80      0.88       758
           1       0.82      0.98      0.89       703

    accuracy                           0.89      1461
   macro avg       0.90      0.89      0.89      1461
weighted avg       0.90      0.89      0.89      1461
```

```
# SVM Tunning
svm_tun = best_svm.predict(X_test)
print('Akurasi SVM Tunning: ', accuracy_score(y_test, svm_tun))
print('Classification Report SVM Tunning: ')
print(classification_report(y_test, svm_tun))
```

✓ 0.7s

```
Akurasi SVM Tunning:  0.9671457905544147
Classification Report SVM Tunning:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       758
           1       0.96      0.97      0.97       703

    accuracy                           0.97      1461
   macro avg       0.97      0.97      0.97      1461
weighted avg       0.97      0.97      0.97      1461
```

```
# Decision Tree Tunning
tree_tun = best_tree.predict(X_test)
print('Akurasi Decision Tree Tunning: ', accuracy_score(y_test, tree_tun))
print('Classification Report Decision Tree Tunning: ')
print(classification_report(y_test, tree_tun))
```

✓ 0.0s

```
Akurasi Decision Tree Tunning:  0.9000684462696783
Classification Report Decision Tree Tunning:
              precision    recall  f1-score   support

           0       0.89      0.92      0.91       758
           1       0.91      0.88      0.89       703

    accuracy                           0.90      1461
   macro avg       0.90      0.90      0.90      1461
weighted avg       0.90      0.90      0.90      1461
```

```
# Logistic Regression Tunning
logreg_tun = best_logreg.predict(X_test)
print('Akurasi Logistic Regression Tunning: ', accuracy_score(y_test, l
# mse, mae, rmse
print('Mean Absolute Error Logistic Regression Tunning: ', mean_absolut
print('Mean Squared Error Logistic Regression Tunning: ', mean_squared_
print('Root Mean Squared Error Logistic Regression Tunning: ', np.sqrt(
)
```

✓ 0.0s

```
Akurasi Logistic Regression Tunning:  0.7994524298425736
Mean Absolute Error Logistic Regression Tunning:  0.2005475701574264
Mean Squared Error Logistic Regression Tunning:  0.2005475701574264
Root Mean Squared Error Logistic Regression Tunning:  0.44782537909036196
```

Kemudian dilakukan prediksi dan evaluasi ulang terhadap model yang telah ditunning dan terdapat beberapa peningkatan dari hasil model sebelumnya.

Testing Dengan Data Baru

Tes SVM Model

Testing predict

```
df_test = pd.read_csv('Data_Test.csv')
df_test.head()
```

✓ 0.0s Python

	id	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_eve_calls
0	1	KS	128	area_code_415	no	yes	25	265.1	110	45.07	197.4	99
1	2	AL	118	area_code_510	yes	no	0	223.4	98	37.98	220.6	101
2	3	IA	62	area_code_415	no	no	0	120.7	70	20.52	307.2	76
3	4	VT	93	area_code_510	no	no	0	190.7	114	32.42	218.2	111
4	5	NE	174	area_code_415	no	no	0	124.3	76	21.13	277.1	112

```
# svm
y_pred_svm = best_svm.predict(df_test)

# simpan hasil prediksi
df_test['churn'] = y_pred_svm
```

✓ 0.3s Python

```
df_test.head(5)
```

✓ 0.0s Python

ber_customer_service_calls	area_code_area_code_408	area_code_area_code_415	area_code_area_code_510	international_plan_no	international_plan_yes	voice_mail_plan_no	voice_mail_plan_yes	churn
-0.497639	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0
-1.281734	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0
1.854646	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0
1.070551	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0
1.070551	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0

```
# save to csv dengan mengambil
df_test.to_csv('hasil_prediksi.csv', index=False)
```

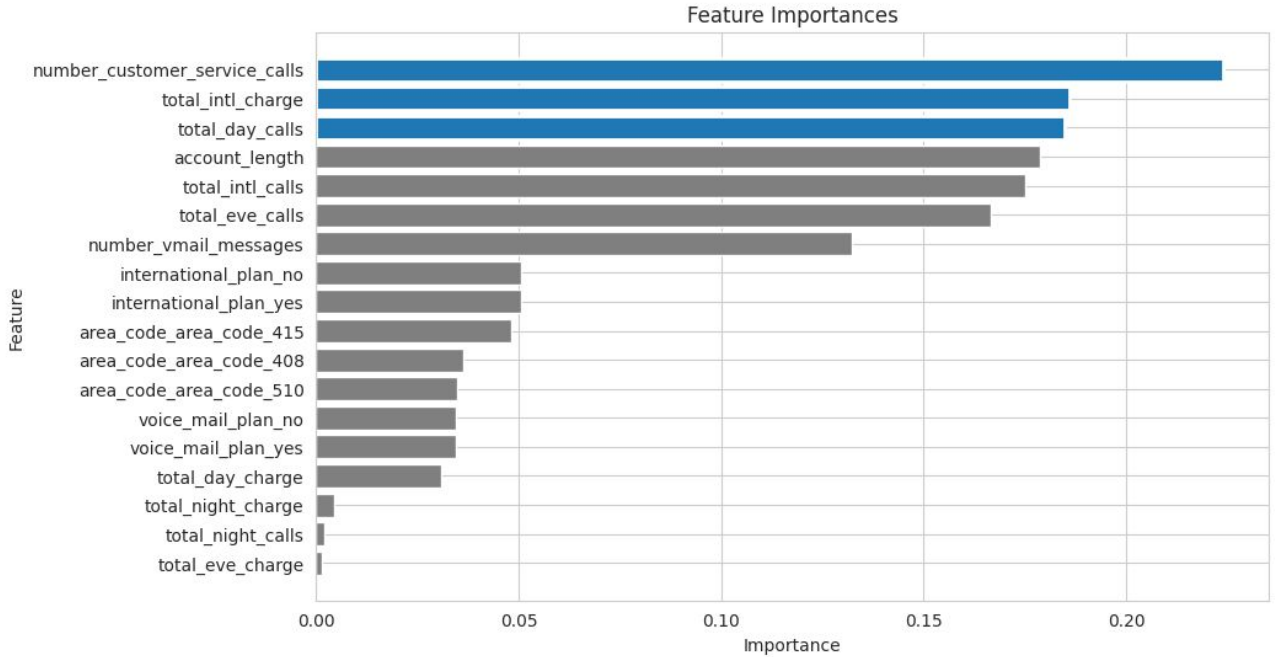
✓ 0.0s Python

Testing dilakukan menggunakan Model SVM dikarenakan memiliki performa yang bagus dan hasilnya model ini bekerja dengan baik untuk prediksi dan klasifikasi.

Business Insight

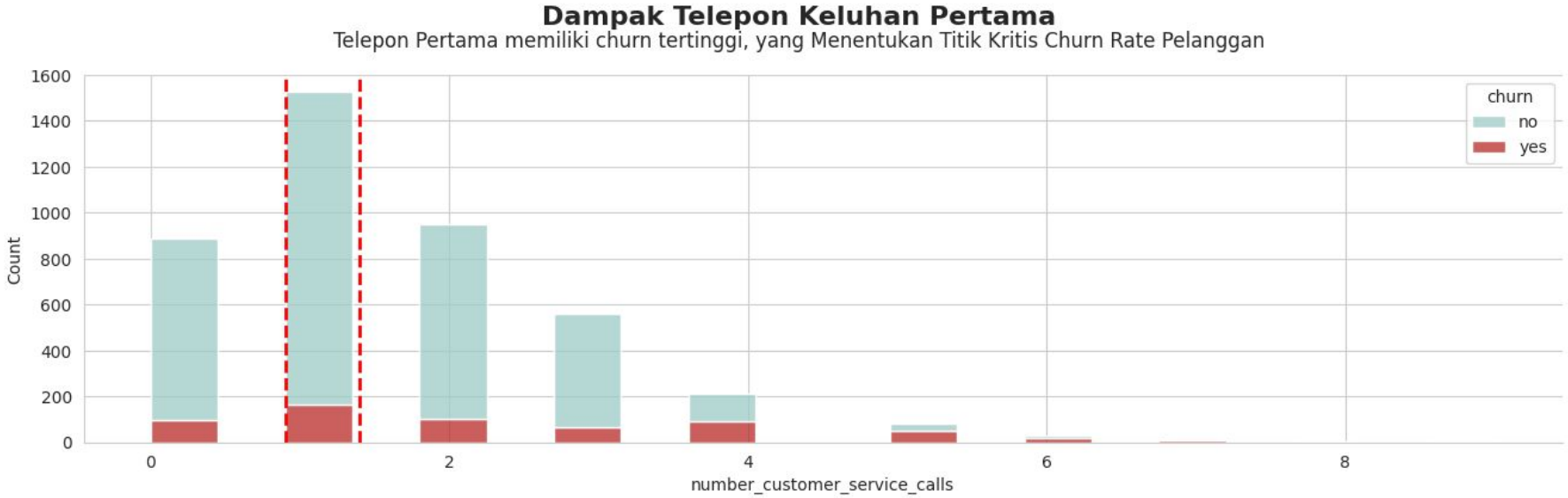
BINAR

Feature Importance

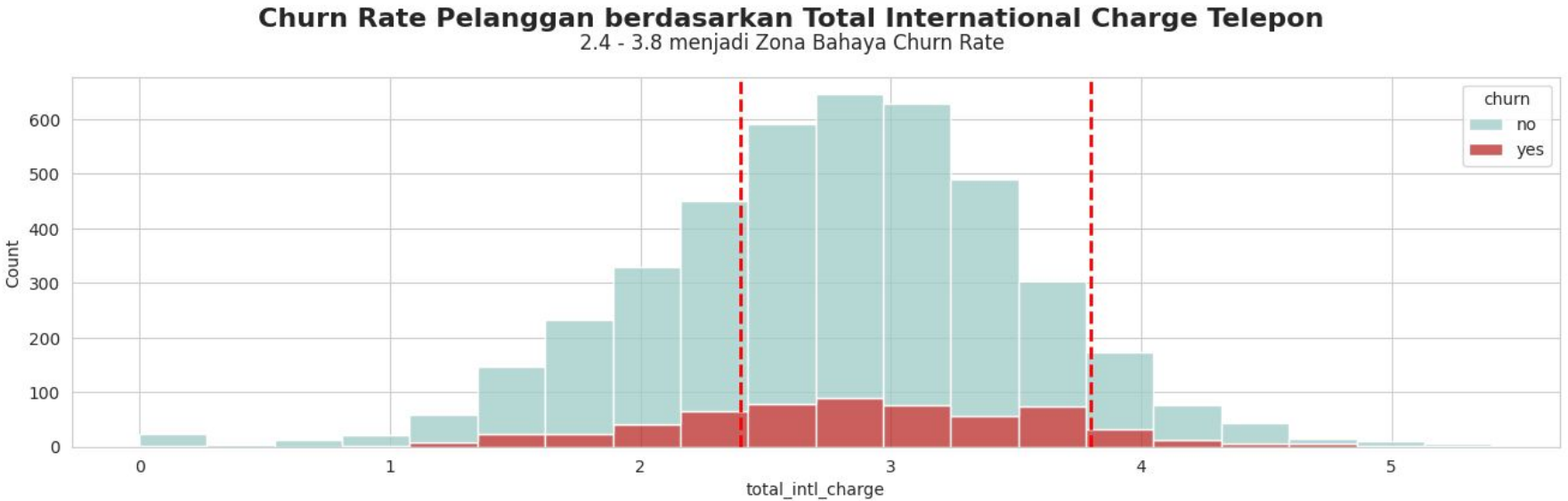


Berdasarkan model yang telah ditentukan dan dilatih, berikut adalah 3 fitur terpenting yang mempengaruhi pelanggan melakukan churn.

Business Insight



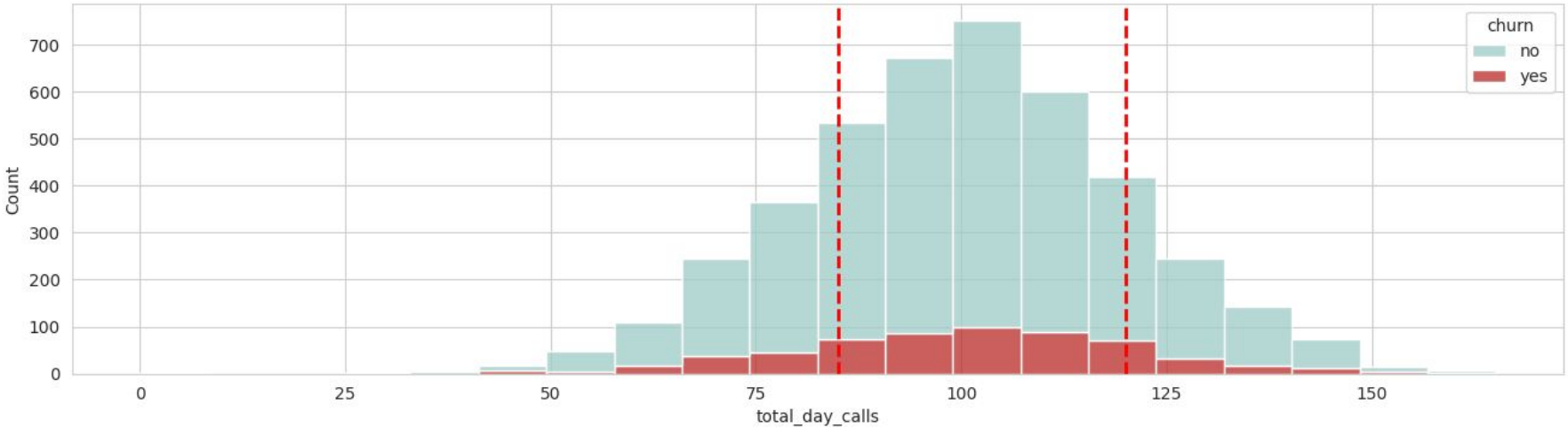
Business Insight



Business Insight

Churn Rate Pelanggan berdasarkan Total International Charge Telepon

Terdapat lonjakan churn rate pada pelanggan dengan total days call di kisaran jumlah 85-120.
Kelompok pelanggan ini perlu mendapat perhatian khusus untuk memahami alasan churn



Kesimpulan dan Saran

Kesimpulan dan Saran

Kesimpulan

Dari beberapa algoritma yang diuji pada kasus ini, algoritma SVM muncul sebagai yang paling efektif untuk klasifikasi dan prediksi churn. Namun, tahapan preprocessing merupakan aspek yang paling vital dalam proses ini. Dengan memperhatikan pengelolaan data yang tepat sebelum masuk ke dalam model, ini dapat memastikan bahwa model yang dihasilkan memberikan prediksi yang akurat dan berguna. Oleh karena itu, kombinasi penggunaan algoritma SVM dengan tahapan preprocessing yang cermat dapat menjadi pendekatan yang efektif dalam menangani masalah churn dalam dataset yang diberikan.

Saran

Berdasarkan hasil analisis, disarankan untuk melanjutkan pengembangan dengan fokus pada peningkatan kualitas data, termasuk proses preprocessing yang cermat untuk membersihkan data dan melakukan transformasi jika diperlukan. Selain itu, pertimbangkan untuk menggabungkan pendekatan yang berbeda atau menggunakan ensemble learning guna meningkatkan kemampuan model dalam menangani kompleksitas data churn. Evaluasi yang cermat terhadap model yang dikembangkan sangat penting, termasuk pengujian performa terhadap data validasi atau data uji yang tidak pernah dilihat sebelumnya, serta perhatikan faktor-faktor bisnis yang mungkin memengaruhi churn. Terus memantau performa model secara berkala dan memperbarui model sesuai kebutuhan akan membantu memastikan relevansi dan efektivitasnya dalam mengatasi masalah churn.

Report Pembagian Tugas

Report Pembagian Tugas

Nama	Tasklist/Deliverable
Taufiq Qurohman Ruki	Latar belakang,tujuan, preprocessing (Encode, normlisasi,dan smote), evaluasi model, testing dengan dataset baru, dan bikin ppt.
Aleisya Zahari Salam	Proses EDA ,Fitur selection, Preprocessing (Pengelompokkan data), modelling, tunning data, feature importance & business insight, ppt

Note: Pengerjaan task ini kami berdua setiap orangnya mencoba secara bersama-sama semua tahapan, kemudian diakhir menggabungkan atau saling melengkapi dalam tiap tahapannya.

BINAR

Thank You