KLASIFIKASI *CUSTOMER CHURN* DENGAN *MACHINE LEARNING*

Syafaatul Khayati (149368779101-598)
The Colab Notebook

LANGKAH PENGERJAAN

Proses pembuatan model *machine learning* untuk klasifikasi pada *challenge* ini secara umum bisa dibagi menjadi 3 tahap dengan tujuan akhir prediksi *unseen data* yang belum memiliki label

01 EDA

Langkah awal untuk mengenali data, menghilangkan *outliers*, dan bagaimana tiap variabel berinteraksi

02 Feature Engineering

Pemilihan variabel yang dianggap penting dan transformasi agar data bisa diproses oleh model ML

03 Machine Learning

Dilakukan prosedur pemilihan model, *hyperparameter tuning*, hingga konstruksi model

Exploratory Data Analysis

01

AKSES *FILE* DAN TAMPILKAN DATA

File CSV yang digunakan disimpan dalam Google Drive dan diakses dengan menghubungkan (*mounting*) *drive* tersebut ke Colab Notebook. Data dibaca ke Pandas *Dataframe* menggunakan fungsi **.read_csv()** dan ditampilkan cuplikan awalnya dengan fungsi **.head()**

File CSV -> Pandas *Dataframe*

```
df_raw = pd.read_csv('train.csv')
```

Tampilkan 5 baris pertama

```
df_raw.head()
```

it's 20 columns long!!!

gth	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	total_eve_charge	total_night_minutes	total_night_calls	total_night_charge	total_intl_minutes	total_intl_ca
107	area_code_415	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	
137	area_code_415	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	
84	area_code_408	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	
75	area_code_415	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	
121	area_code_510	no	yes	24	218.2	88	37.09	348.5	108	29.62	212.6	118	9.57	7.5	

JENIS DATA

Pembagian ini dilakukan untuk memudahkan pemilihan metode analisis dan *modelling*

Kategoris

- **Nominal Data**:
  - state dan area_code: menunjukkan lokasi customer
  - international_plan dan voice_mail_plan: *categorical binary data* (*boolean*) yang menunjukkan paket layanan *customer*
  - churn: variabel target *binary classification* yang menunjukkan apakah customer berhenti berlangganan/tidak
- **Ordinal Data:** -

Numerik

- **Discrete Data**:
  - account_length: lama berlangganan
  - number_vmail_messages: pesan suara yang diterima
  - total_day_calls, total_eve_calls, total_night_calls, total_intl_calls, number_customer_service_calls: jumlah panggilan
- **Continuous Data**:
  - total_day_minutes, total_eve_minutes, total_night_minutes, total_intl_minutes: lama panggilan
  - total_day_charge, total_eve_charge, total_night_charge, total_intl_charge: banyak tagihan

INFORMASI UMUM

Untuk mengetahui isu awal dari *dataset*, dijalankan beberapa perintah seperti **.info()**, **isnull().mean()** untuk mencari rasio *missing values*, dan **.duplicated()** untuk mengecek baris ganda/bukan.

Dataset ini tidak memiliki *missing values* dan baris duplikat.

Informasi tiap kolom dalam *dataframe*

```
df_raw.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
```

It tells us:
There are 4250 rows and 20 columns with already appropriate data types and no missing values!

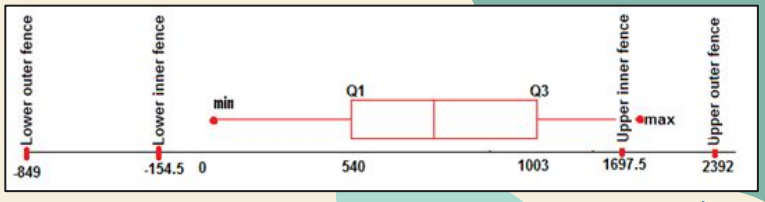
UNIVARIAT NUMERICAL DATA ANALYSIS

Analisis ini melihat distribusi frekuensi nilai variabel menggunakan *bar plot* (untuk nilai diskrit) dan histogram (untuk nilai kontinu), serta melakukan ***outlier detection***.

OUTLIERS DETECTION

Dalam melakukan *univariat numerical analysis*, dilakukan pendeteksian galat berupa nilai ekstrim (*outliers*) pada variabel tersebut. Di sini, digunakan **Tukey's Box Plot** dimana *inner fence* membatasi nilai yang dianggap normal dengan *possible outlier* (*low probability of outlier*), dan *outer fence* membatasi *possible outlier* dengan *probable outlier* (*high probability of outlier*). Di sini, data dianggap mencurigakan bila masuk ke *probable outlier*. Metode ini dipilih karena cukup intuitif dan tidak mengasumsikan normalitas distribusi.

Ilustrasi Tukey's *Box Plot*



Sumber

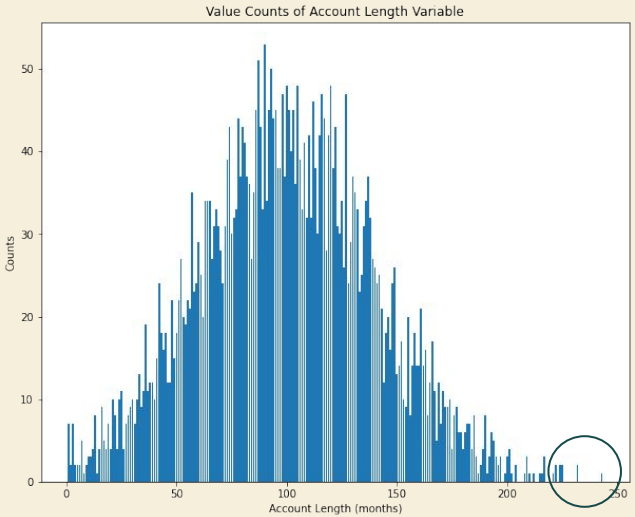
Implementasi Tukey's *Box Plot*

```
def get outliers (df,col):
    q3 = df[col].quantile(0.75)
    q1 = df[col].quantile(0.25)
    iqr = q3 - q1

    # find inner fence
    inner fence low = q1 -  1.5*iqr
    inner_fence_up = q3 +  1.5*iqr

    # find outer fence
    outer fence low = q1 -  3*iqr
    outer_fence_up = q3 +  3*iqr

    # if between inner-outer fence then possible, else if
outside outer fence then probable
    possible outliers = df [((outer fence low <= df [col]) &
(df[col] < inner fence low )) | ((outer_fence_up >= df [col])
& (df[col] > inner fence up ))]
    probable outliers = df [(df[col] < outer_fence_low ) |
(df[col] > outer_fence_up )]

    return possible_outliers , probable_outliers
```

UNIVARIAT NUMERICAL ANALYSIS:

ACCOUNT_LENGTH

Grafik ini menggambarkan banyak orang yang berlangganan selama jangka waktu tertentu:



Bar plot tersebut menunjukkan bahwa nilai ekstrim di sisi kanan grafik meningkatkan nilai *central tendency* dan *measure of spread* sehingga menghasilkan standar deviasi cukup tinggi, yaitu 39 bulan.

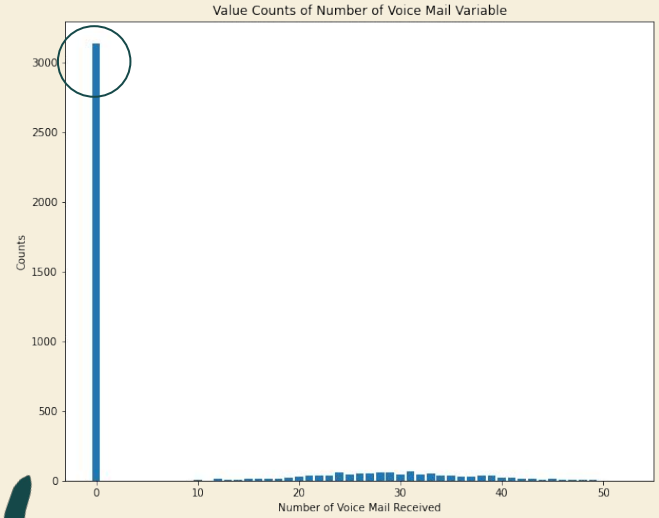
Namun demikian, berdasar uji *outlier* menggunakan fungsi **get_outlier()**, terlihat walaupun ada 20 *possible outliers*, tidak ada *probable outliers* dalam dataset tersebut. Karena tidak ada alasan kuat untuk mencurigai kesalahan *input* atau *error* lainnya, maka tidak ada data yang perlu dihapus atau dikoreksi nilainya.

"normal *outlier*" and removing it may causes the model to be biased!

UNIVARIAT NUMERICAL ANALYSIS:

NUMBER_VMAIL_MESSAGES

Now THIS is a suspicious data!



Untuk variabel ini, *bar plot* menunjukkan tingginya frekuensi orang yang tidak menerima pesan suara (*voicemail*) sama sekali. Hal ini cukup mencurigakan, sehingga dilakukan pengecekan status variabel *voice_mail_plan* dari semua orang yang memiliki o pesan untuk mengetahui lebih lanjut alasan fenomena ini. Ternyata, mayoritas orang tidak menerima *voicemail* karena tidak mendaftarkan diri ke program *voicemail*.

```
df_raw[df_raw['number_vmail_messages']==0]['voice_mail_plan'].value_counts()
```

```
no     3138
yes       1
Name: voice_mail_plan, dtype: int64
```

Most people do not sign up for the voicemail plan!

UNIVARIAT NUMERICAL ANALYSIS:

NUMBER_VMAIL_MESSAGES

Berdasarkan *discovery* tersebut, muncul dugaan bahwa ada korelasi yang cukup tinggi antara kedua variabel. Untuk mengkonfirmasi dugaan tersebut, dilakukan *encoding* variabel *voice_mail_plan* (0 untuk 'no', 1 untuk 'yes') kemudian dicari korelasinya menggunakan fungsi **.corr()**.

Correlation matrix yang dihasilkan menunjukkan Pearson *correlation coefficient* dengan nilai 0.95014. Hal ini mengindikasikan adanya **colinearity** antara kedua variabel, sehingga *number_vmail_messages* bisa di-*drop* pada fase *feature engineering* dan hanya *keep voice_mail_plan* saja.

```
df_temp = df_raw.loc[:,['number_vmail_messages','voice_mail_plan']]
df_temp['voice_mail_plan'] = df_temp['voice_mail_plan'].apply(lambda x: 0 if x=='no' else 1)
corr_mtx = df_temp.corr()
corr_mtx
```

	number_vmail_messages	voice_mail_plan
number_vmail_messages	1.000000	0.954014
voice_mail_plan	0.954014	1.000000

Very high correlation since most people has 0 value in both variables!

UNIVARIAT NUMERICAL ANALYSIS:

DAY, NIGHT, & EVE VARIABLES

Perfect correlation between *total_day_minutes* and *total_day_charge*

	total_day_minutes	total_day_calls	total_day_charge
total_day_minutes	1.000000	0.000747	1.000000
total_day_calls	0.000747	1.000000	0.000751
total_day_charge	1.000000	0.000751	1.000000

The same thing with night...

	total_night_minutes	total_night_calls	total_night_charge
total_night_minutes	1.000000	0.023815	0.999999
total_night_calls	0.023815	1.000000	0.023798
total_night_charge	0.999999	0.023798	1.000000

...and evening!

	total_eve_minutes	total_eve_calls	total_eve_charge
total_eve_minutes	1.000000	0.003101	1.00000
total_eve_calls	0.003101	1.000000	0.00312
total_eve_charge	1.000000	0.003120	1.00000

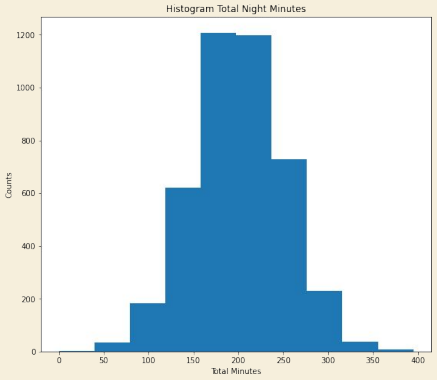
Dalam *dataset* ini, penggunaan telepon *customer* pada periode waktu tertentu dicatat dalam 3 variabel, yaitu *total calls* (jumlah panggilan), *total minutes* (lamanya panggilan), dan *total charge* (banyak tagihan).

Ternyata, apabila dicari koefisien korelasi Pearson antara ketiganya, akan didapatkan korelasi sempurna antara *total minutes* dan *total charge*. Dengan demikian, bisa dipilih salah satu variabel saja sebagai *input features* pada model *machine learning*. Di sini, baik untuk variabel *day, night*, maupun *eve* (*evening*), dipilih *total minutes* dan *total calls*, sedangkan *total charge* akan di-*drop*.

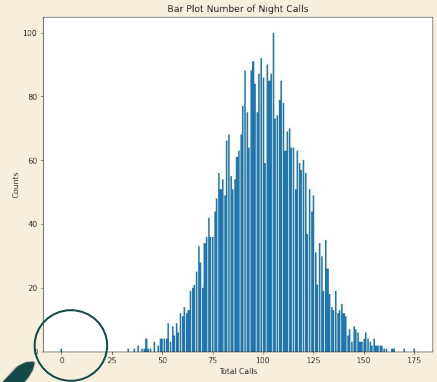
UNIVARIAT NUMERICAL ANALYSIS:

DAY, NIGHT, & EVE VARIABLES

Total_night_minutes histogram



Total_night_calls bar plot



Grafik di samping menggambarkan jumlah dan durasi panggilan pada malam hari, dimana terlihat ada *probable outlier* untuk variabel jumlah panggilan, yaitu pengguna dengan jumlah panggilan o saat malam dan intensitas normal di periode lain. Fenomena tersebut mungkin saja terjadi karena pengguna terkait benar-benar tidak melakukan aktivitas saat malam hari, sehingga *data points* tersebut bukan pasti *error* dan tidak perlu dihapus.

```
poss, prob = get_outliers(df_raw,'total_night_calls')
prob
```

_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
214.3	138	36.43	232.2	99	19.74	0.0	0	0.0	

a probable outlier but it doesn't seem to be an error, so we'll keep it!

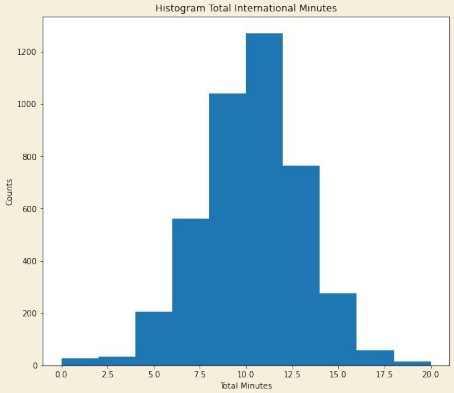
Hal ini juga terjadi untuk variabel *total_day_calls* dan *total_eve_calls*, sehingga juga tidak ada data yang dihapus untuk variabel tersebut

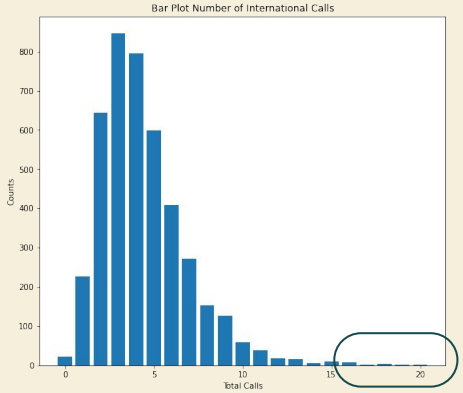
UNIVARIAT NUMERICAL ANALYSIS:

INTERNATIONAL VARIABLES

	international_plan	total_intl_minutes	total_intl_calls	total_intl_charge
international_plan	1.000000	0.023815	0.006956	0.023799
total_intl_minutes	0.023815	1.000000	0.019328	0.999993
total_intl_calls	0.006956	0.019328	1.000000	0.019414
total_intl_charge	0.023799	0.999993	0.019414	1.000000

very high correlation!





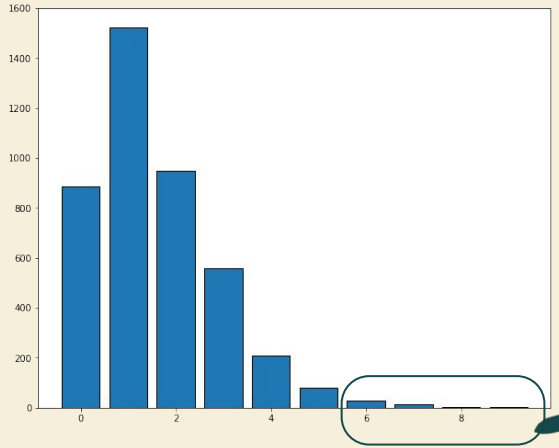
a probable outlier
(but we will keep it!)

Pada *dataset* ini, ada 4 variabel yang berhubungan dengan panggilan internasional yang diduga *colinear* satu sama lain. Ternyata, koefisien korelasi menunjukkan tingginya hubungan antara *total_intl_minutes* dan *total_intl_charge* saja, sehingga variabel tagihan akan di-*drop* dan yang lain di-*keep*.

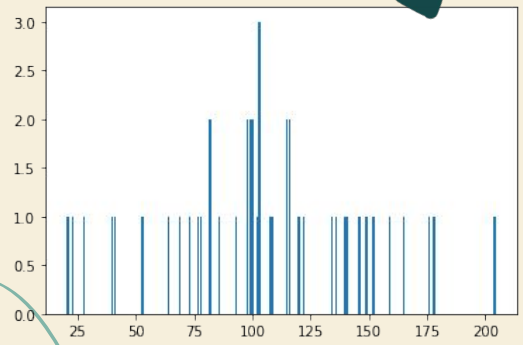
Di sisi lain, *central tendency total_intl_calls* relatif rendah dibanding jenis panggilan lain, sehingga 16 kali panggilan sudah dianggap *probable outlier*. Karena masih masuk akal, maka data ini akan tetap dipertahankan dalam *dataset*.

UNIVARIAT NUMERICAL ANALYSIS:

NUMBER_CUSTOMER_SERVICE_CALLS



>6 calls to customer service is considered a probable outlier



but these people have been a customer for such a long time!

Pada variabel *number_customer_service_calls*, karena *central tendency* variabelnya rendah, *upper outer fence*-nya pun cukup rendah, sehingga 6 kali panggilan pun sudah dianggap tidak masuk akal dan dikategorikan sebagai *probable outlier*.

Padahal, para pelanggan yang menelepon CS >= 6 kali tersebut sudah menggunakan layanan ISP paling tidak sekitar 2 tahun. Wajar bila para pelanggan tersebut memiliki masalah tertentu dengan *service* yang ditawarkan atau ada masalah seperti pembayaran.

Dengan demikian, tidak dilakukan penghapusan data lagi di variabel ini.

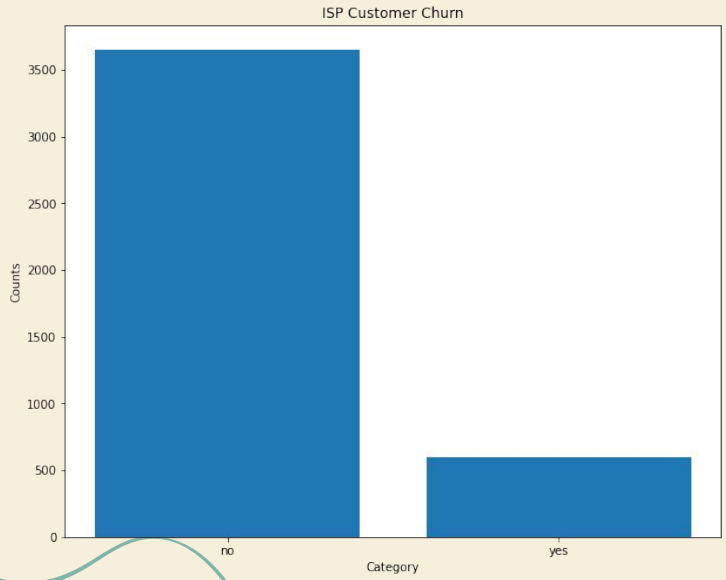
UNIVARIAT CATEGORICAL DATA ANALYSIS

Tahap ini memahami karakteristik data, memastikan konsistensi kategori tiap variabel, dan melihat persebaran frekuensi untuk tiap kategori.

UNIVARIAT CATEGORICAL ANALYSIS:

CHURN

Grafik ini menggambarkan frekuensi pelanggan yang lanjut (churn='no') dan berhenti (churn='yes') berlangganan

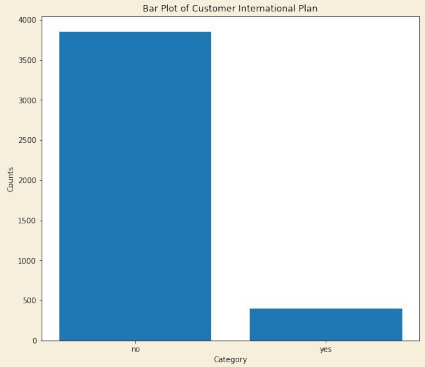


Pada *bar plot*, terlihat bahwa jumlah *customer* yang melanjutkan berlangganan jauh lebih banyak daripada yang berhenti. Idealnya, dilakukan teknik seperti *Synthetic Minority Oversampling Technique* (SMOTE) untuk mengatasi *imbalanced classification*, namun pada *challenge* ini *dataset training* akan diterima seadanya dengan asumsi *dataset* representatif terhadap kondisi populasi.

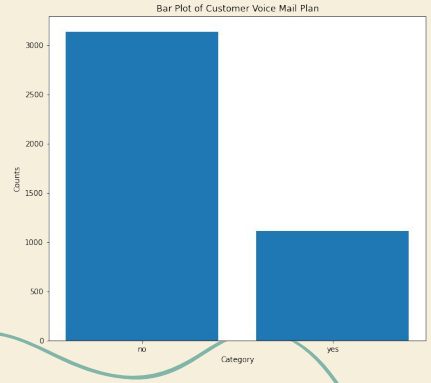
Label kategori biner ini juga perlu di-*encode* pada *feature engineering* dengan mengganti 'no' menjadi 0 dan 'yes' menjadi 1.

UNIVARIAT CATEGORICAL ANALYSIS:

INTERNATIONAL_PLAN and VOICE_MAIL_PLAN



More people choose to not sign up for the ISP's international plan



Most people also do not use the voicemail service

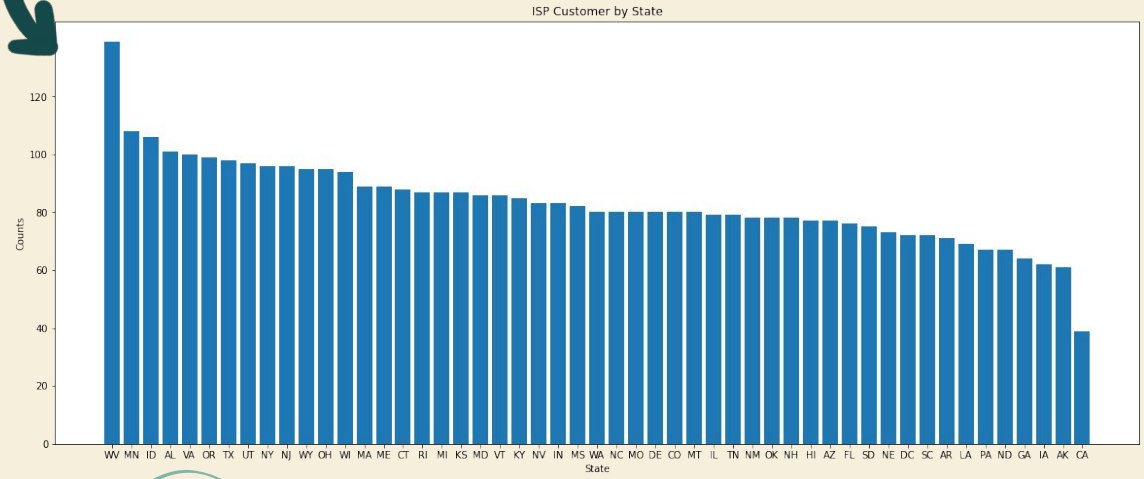
Kedua variabel ini menunjukkan status langganan *customer* terhadap layanan *international plan* dan *voicemail plan*. Pengguna yang tidak memiliki layanan *voicemail* tidak bisa menerima pesan suara sama sekali (sehingga *number_vmail_messages*=0), sedangkan orang yang tidak memiliki *international plan* masih bisa melakukan panggilan internasional namun dengan *rate* yang lebih tinggi.

UNIVARIAT CATEGORICAL ANALYSIS:

STATE

But there are only 50 states in the US???

```
df_raw[[c for c in df_raw.columns if c in nom_col]].nunique(axis=0)

state                51
area_code             3
international_plan    2
voice_mail_plan       2
churn                 2
dtype: int64
```

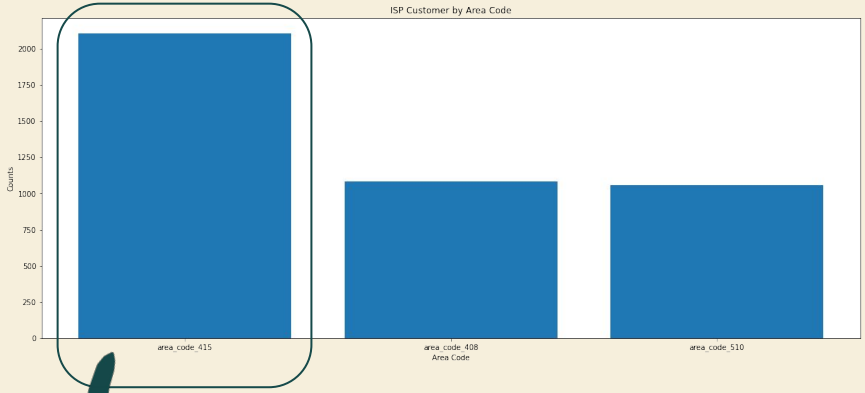


Pada analisis awal, terlihat bahwa variabel ini memiliki 51 *unique values*, sedangkan hanya ada 50 negara bagian di Amerika Serikat. Ternyata, Washington *state* (WA) dibedakan dengan Washington DC (DC), ibukota AS yang ada di negara bagian lain.

Berdasarkan *bar plot* di samping, bisa dilihat banyak *data points* dari tiap *state*. Variabel ini memiliki *cardinality* sangat tinggi, sehingga akan diperlukan *binning* pada tahap *feature engineering*.

UNIVARIAT CATEGORICAL ANALYSIS:

AREA_CODE



Same area_code but different states, all over the US?

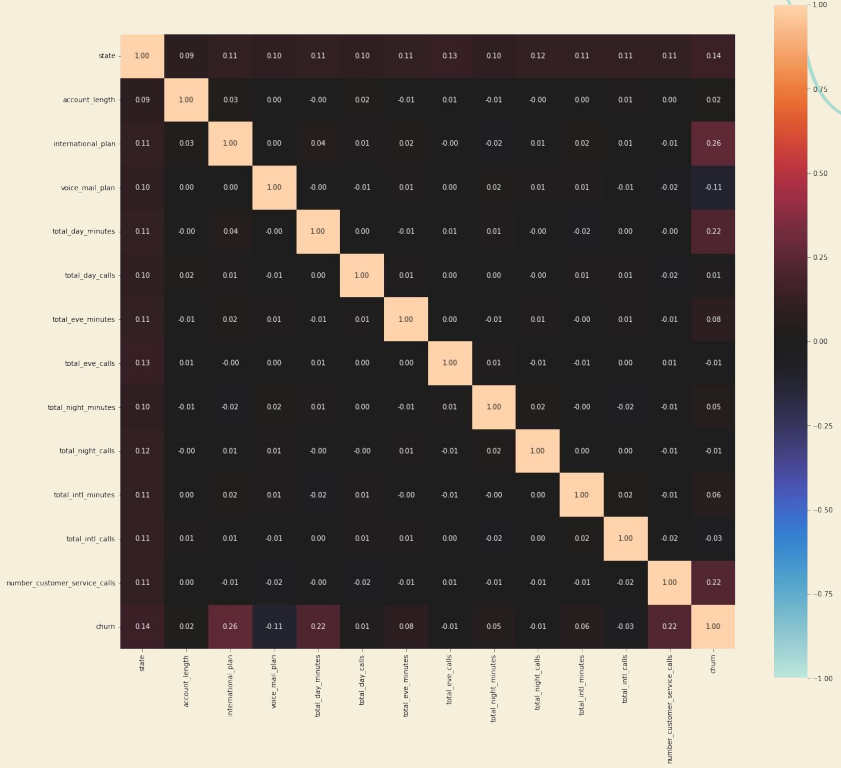
```
df_raw[df_raw['area_code']=='area_code_415']['state'].value_counts()

WV    62
ID    59
OR    56
AL    54
VA    54
      ..
ND    30
OK    29
GA    25
IA    24
CA    19
Name: state, Length: 51, dtype: int64
```

Di *dataset* ini, hanya ada 3 kategori dalam variabel kode area. Hal ini cukup mencurigakan karena *area code* di USA umumnya dibagi berdasarkan wilayah geografisnya, sehingga karena ada 51 negara bagian maka pasti ada >51 kode area.

Ternyata, untuk data dengan kode area 415 yang seharusnya ada di California (CA), tercatat ada 51 *unique values* pada variabel *state*. Ini menunjukkan variabel *area_code* kemungkinan besar adalah *error* yang harus di *drop* pada *feature engineering*.

HUBUNGAN ANTAR VARIABEL

Analisis ini melihat korelasi antar variabel, sehingga bisa diperkirakan bagaimana nilai satu variabel berpengaruh dengan variabel lain. Karena ada variabel nominal *state*, digunakan library *dython* yang mengizinkan perhitungan korelasi nominal-numerik dengan metode Correlation Ratio



Feature Engineering

02

STEP ONE:

DROP VARIABEL YANG TIDAK DIPERLUKAN

```
df_clean =
df_raw.copy().drop(axis='columns',labels=['number_vmail_mes
sages','total_day_charge', 'total_night_charge',
'total_eve_charge','total_intl_charge','area_code'])
```

Berdasarkan EDA yang telah dilakukan sebelumnya, diketahui variabel-variabel yang *redundant* atau *unnecessary*, yaitu:

- *number_vmail_messages*, karena *colinear* dengan *voice_mail_plan*
- *total_day_charge*, *total_night_charge*, *total_eve_charge* , dan *total_intl_charge*, karena memiliki korelasi yang tinggi dengan variabel durasi masing-masing
- *area_code*, karena diduga mengandung galat

STEP TWO:

ENCODE KATEGORI BINER

```
df_clean['international_plan'] = df_clean['international_plan'].apply(lambda x: 0 if x=='no' else
1)
df_clean['voice_mail_plan'] = df_clean['voice_mail_plan'].apply(lambda x: 0 if x=='no' else 1)
df_clean['churn'] = df_clean['churn'].apply(lambda x: 0 if x=='no' else 1)
```

Di sini, jawaban ‘yes’ dan ‘no’ akan direpresentasikan dengan angka, yaitu 0 untuk ‘no’ dan 1 untuk ‘yes’. Hal ini diimplementasikan dengan *lambda function* karena pada saat pengerjaan dianggap lebih sederhana dan tidak memerlukan *transformer* tertentu (misal *sklearn.preprocessing.OneHotEncoder*())

STEP THREE:

DATA BINNING PADA VARIABEL *STATE*

```
df_clean_state = df_clean.copy()
df_clean_state['state'] = df_clean_state['state'].apply(lambda x: 'Northeast' if x in lst_northeast else 'Midwest' if x in lst_midwest else 'South' if x in lst_south else 'West')
df_clean_state['state'].value_counts()
```

```
South        1432
West         1114
Midwest       928
Northeast     776
Name: state, dtype: int64
```

Di sini, dilakukan pengelompokan *state* sehingga berkurang dari 51 variasi menjadi 4 *unique values* saja. Pengelompokan ini dilakukan berdasar area geografis, khususnya menurut badan sensus Amerika Serikat. Sebenarnya ini bukan cara paling optimal untuk melakukan *binning*, tapi salah satu cara paling sederhana bila variabel yang akan di-*bin* adalah lokasi geografis.

Terlihat bahwa setelah proses *binning*, South memiliki data paling banyak sejumlah 1432 baris.

STEP THREE:

ONE HOT ENCODING VARIABEL *STATE*

```
df_final = pd.get_dummies(df_clean_state,prefix='state',
                          columns=['state'],
                          drop_first=True)
```

state_Northeast	state_South	state_West
0	0	0
1	0	0
0	0	0
0	1	0
1	0	0

Midwest!

This reduces the amount of input features we use for our model!

OneHotEncoding bertujuan mengubah data kategori nominal yang berbentuk *string* menjadi *int* yang bisa diproses model *machine learning*. Di sini, digunakan fungsi **.get_dummies()** dari Pandas dengan parameter *drop_first=True*. Ini menyebabkan kolom **state_** hasil OneHotEncoding hanya ada 3, karena Midwest direpresentasikan dengan nilai 0 pada kolom *state_Northeast, state_South*, dan *state_West*.

STEP FOUR:

PISAH *FEATURES* DAN LABEL

```
X = df_final.copy().drop(axis='columns',labels=['churn'])
y = df_final.iloc[:, 12]
```

For X: copy final df and drop churn column
For y: take the 12 column of final df

```
X.shape
```

```
(4250, 15)
```

```
y.shape
```

```
(4250,)
```

Di sini, *df_final* dibagi menjadi 2, yaitu:

- *Pandas Dataframe* X yang berisi 4250 *data points* dengan masing-masing 15 fitur
- *Pandas Series* y yang berisi 4250 baris

VALIDATION SETUP

Di sini, dilakukan *train-test split* untuk *dataframe* X dan *series* y dengan rasio 85%-15%.

Dilakukan juga pembuatan **StratifiedKFold** *object* dengan nama **skf**, dimana data di-*split* menjadi 10 bagian dengan *random state* 420. Jenis *sampling* ini digunakan karena *imbalance* pada data untuk memastikan ada label 0 dan 1 yang proporsional dengan *dataset* pada tiap *fold*.

Train-Test Split

```
X_cv, X_test, y_cv, y_test =
train_test_split(X, y,
                 test_size = 0.15,
random_state = 420)
```

StratifiedKFold object

```
skf = StratifiedKFold(n_splits=10,
shuffle=True, random_state=420)
```

Machine Learning

03

PERBANDINGAN METODE

Pada *challenge* ini, akan dibandingan 2 model, yaitu *Support Vector Machine* (SVM) yang merupakan model berbasis jarak dengan asumsi distribusi normal pada data, dan *eXtreme Gradient Boosting* (XGBoost) yang merupakan *tree-based model* yang bisa menangani data dengan tipe distribusi apapun.

FUNGSI COMPARE_MODEL()

Fungsi ini berguna untuk menjalankan fungsi *cross_validate*() dari *sklearn* pada kedua model, kemudian menampilkan *metrics* yang dihasilkan, yaitu akurasi, presisi, *recall*, dan skor F1.

Di sini, digunakan StratifiedKFold *object*, **skf**, yang sudah dibuat sehingga keduanya akan diuji dengan data yang sama.

```
def compare_model(X_train, y_train):
  dfs = []
  models = [
           ('SVM', make pipeline(RobustScaler(),SVC())),
           ('XGB', xgb.XGBClassifier())
         ]

  # all the metrics we want to know for each folds
  scorer acc = make scorer (accuracy score)
  scorer prec = make scorer (precision score)
  scorer rec = make scorer (recall score)
  scorer f1 = make scorer (f1 score)
  scorers = {'acc':scorer acc,
             'prec':scorer prec,
             'rec':scorer rec,
             'f1':scorer_f1}

  # do cv for every model called name in models
  for name, model in models:
        cv_results = cross_validate (model,
                                     X train,
                                     Y train,
                                     cv=skf,
                                     scoring=scorers)
        clf = model.fit (X train, y train)
        this df = pd.DataFrame (cv_results)
        this df['model'] = name
        dfs.append (this df)
  final = pd.concat (dfs, ignore_index= True)
  return final
```

HASIL PERBANDINGAN

Hasil rata-rata tiap *fold* pada *cross validation* tersebut menunjukkan bahwa walaupun XGB memiliki waktu *processing* yang lebih lama dari SVM, semua *metric* yang diinginkan menunjukkan performa lebih baik, terutama *recall*.

```
result.groupby(by='model').mean()
```

model	fit_time	score_time	test_acc	test_prec	test_rec	test_f1
SVM	0.155750	0.024201	0.908358	0.847402	0.426510	0.564922
XGB	0.189257	0.004320	0.952937	0.934227	0.719059	0.811010

SVM has very low recall which results in way lower F1 score

Berdasarkan hasil tersebut,
XGBoost dipilih sebagai metode yang digunakan pada model *machine learning* untuk memprediksi *customer churn*.

Hyperparameters yang Akan Dioptimalisasi

```
params =
{'tree method':['exact','hist','approx'],
  'objective':['binary:logistic','binary:hinge'],
  'n estimators':[100,500,1000],
  'learning rate':[0.05,0.1,1],
  'colsample bytree':[0.5,0.8,1],
  'max leaves':[0],
  'random state':[420]}
```

Instance GridSearchCV yang digunakan untuk mencari *hyperparameter* optimal

```
grid = GridSearchCV(clf,
                    param grid=params ,
                    scoring=scorers ,
                    refit= 'f1',
                    cv=skf ,
                    n jobs= 1,
                    error score= 'raise',
                    return_train_score= True)
```

HYPERPARAMETER TUNING

Dengan GridSearchCV

Di sini, dilakukan pengujian beberapa nilai *hyperparameters* dari model XGBClassifier (kelas XGBoost khusus untuk klasifikasi), dengan skor F1 sebagai *evaluation metrics* utama (*refit='f1'*)

HASIL *TUNING*

Setelah dilakukan *fitting* terhadap data *training*, GridSearchCV akan mengecek semua kombinasi nilai *hyperparameters* yang diberikan, kemudian menawarkan properti **.best_params_** yang menunjukkan kombinasi parameter terbaik. Hasil *metrics* menunjukkan peningkatan dari model *default* pada saat perbandingan dengan SVM.

Best hyperparameter combinations...

```
{'colsample_bytree': 0.8,
 'learning_rate': 0.1,
 'max_leaves': 0,
 'n_estimators': 1000,
 'objective': 'binary:logistic',
 'random_state': 420,
 'tree_method': 'approx'}
```

```
acc: 0.9557016268499104
prec: 0.9217299354648161
rec: 0.7523921568627451
f1: 0.8267446239327683
```

...which results in the highest average F1 score across all cross-validation folds.

MODEL TESTING

```
clf = xgb.XGBClassifier (colsample_bytree= 0.8,
                         learning_rate= 0.1,
                         max_leaves= 0,
                         n_estimators= 1000,
                         objective= 'binary:logistic' ,
                         random_state= 420,
                         tree_method= 'approx')
clf.fit(X_cv,y_cv)
y_pred = clf.predict (X_test)
print(classification_report (y_test,y_pred))
```

Classification report

	precision	recall	f1-score	support
0	0.96	0.99	0.98	549
1	0.96	0.78	0.86	89
accuracy			0.96	638
macro avg	0.96	0.88	0.92	638
weighted avg	0.96	0.96	0.96	638

even higher metrics than in cross validation!

Akan dibuat *instance* XGB baru dengan *hyperparameter* terbaik hasil *tuning* dengan *GridSearchCV*. *Instance* baru ini akan dilatih menggunakan seluruh data *train* kemudian digunakan memprediksi data *test*.

Classification report menunjukkan hasil yang sangat memuaskan dengan skor F1 86% dan keseluruhan *metric* yang lebih baik dari saat *tuning* di GridSearchCV

Prediksi *Unseen Data*

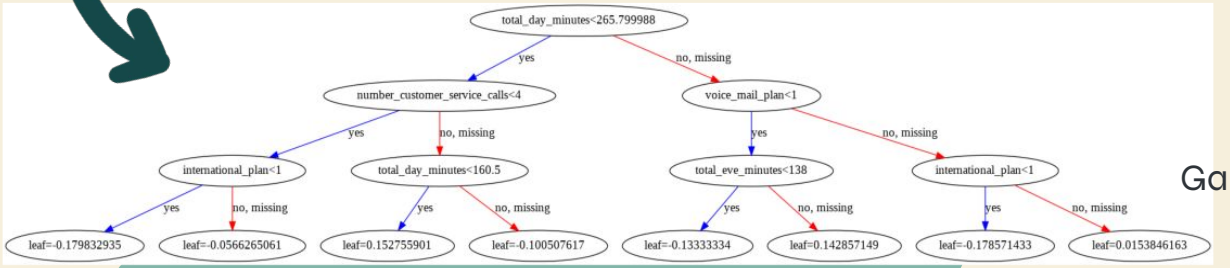
FINAL MODEL

Untuk memprediksi data baru yang belum pernah dilihat sebelumnya dari *file* ***test.csv***, dibuat *instance* XGBClassifer yang dilatih menggunakan semua data dari *file* ***train.csv***.

Model akhir dilatih dengan semua data *train.csv*

```
clf final = xgb.XGBClassifier (colsample bytree= 0.8,
                        learning rate= 0.1,
                        max leaves= 0,
                        n estimators= 1000,
                        objective= 'binary:logistic' ,
                        random state= 420,
                        tree_method= 'approx')
clf_final.fit (X,y)
```

Decision Tree dari model akhir



Salah satu keunggulan model *decision tree* adalah prosesnya yang jelas (*bukan black box*). Gambar di samping adalah *decision tree* dari *clf_final* tadi.

TRANSFORMASI *FEATURES*

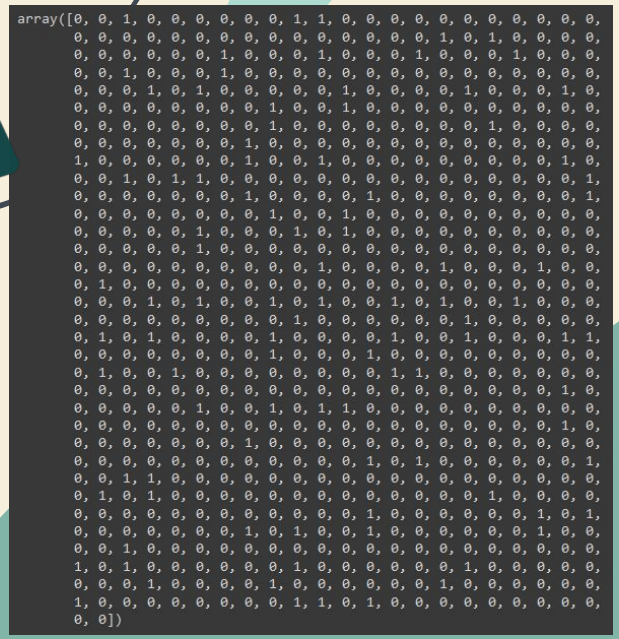
Sebelumnya, sebelum kita memasukkan data *train* ke algoritma *machine learning*, dilakukan *feature engineering* terlebih dahulu. Sekarang, ketika ada *unseen data* yang ingin diprediksi, data tersebut juga harus ditransformasi hingga sama formatnya dengan *input features* yang digunakan untuk melatih model.

Walaupun *standard practice*-nya dilakukan menggunakan *pipeline* dari *sklearn*, di sini dibuat fungsi *model_feature*() yang mengembalikan *dataframe* bersih.

df.info() dari fungsi *model_feature*()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750 entries, 0 to 749
Data columns (total 15 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   account_length                750 non-null    int64
 1   international_plan            750 non-null    int64
 2   voice_mail_plan               750 non-null    int64
 3   total_day_minutes             750 non-null    float64
 4   total_day_calls               750 non-null    int64
 5   total_eve_minutes             750 non-null    float64
 6   total_eve_calls               750 non-null    int64
 7   total_night_minutes           750 non-null    float64
 8   total_night_calls             750 non-null    int64
 9   total_intl_minutes            750 non-null    float64
 10  total_intl_calls              750 non-null    int64
 11  number_customer_service_calls 750 non-null    int64
 12  state_Northeast               750 non-null    uint8
 13  state_South                   750 non-null    uint8
 14  state_West                    750 non-null    uint8
dtypes: float64(4), int64(8), uint8(3)
memory usage: 72.6 KB
```

```
output = clf_final.predict(input_df)
```



PREDIKSI LABEL

Prediksi label bisa dilakukan dengan memanggil fungsi **.predict()**. Hasil yang diberikan berbentuk *array* seperti gambar di samping.