Credit Card Approval

written by IRIS TENTAN

Problem Statement Understanding

Problem Statement

IRIS TENTAN adalah sebuah lembaga keuangan yang menyediakan berbagai layanan perbankan termasuk penerbitan kartu kredit. Setiap bulannya, IRIS TENTAN menerima ribuan aplikasi kartu kredit dari calon nasabah. Namun, beberapa masalah utama yang dihadapi bank ini adalah:

- **High Rejection Rate:** Tingkat penolakan aplikasi kartu kredit yang tinggi karena tidak terpenuhinya kriteria kelayakan.
- **Manual Review Cost:** Proses peninjauan aplikasi yang manual dan memakan waktu, yang meningkatkan biaya operasional.
- **Credit Risk:** Pemberian kartu kredit kepada nasabah yang berisiko tinggi untuk gagal bayar, yang dapat menyebabkan kerugian finansial bagi bank.

Dengan tantangan-tantangan ini, manajemen IRIS TENTAN meminta tim data untuk menganalisis dan membangun model yang dapat secara akurat memprediksi kelayakan pengajuan kartu kredit. Model ini diharapkan dapat:

- **Menurunkan tingkat penolakan yang tidak perlu** dengan memberikan prediksi yang lebih akurat.
- **Mengurangi biaya dan waktu dalam peninjauan aplikasi** melalui otomatisasi.
- **Meminimalkan risiko kredit** dengan mendeteksi aplikasi berisiko tinggi lebih awal.

Roles

Sebagai **tim data** di IRIS TENTAN, berikut adalah beberapa roles beserta PIC yang berkontribusi dalam penyelesaian permasalahan ini:

- **Lead Data Science**

  Sebagai koordinator project

  PIC: `Netri Alia Rahmi`

- **Data Scientist**

  Membuat insight business dan rekomendasi kebijakan kredit

  PIC: `Rizki Widianto` dan `Sukma Sekar Devita`

Goals

IRIS TENTAN ingin meningkatkan **tingkat persetujuan aplikasi yang layak** dan mengurangi **risiko kredit** sehingga dapat meningkatkan **keuntungan** dan **efisiensi operasional**.

Objectives

Membangun model klasifikasi untuk **memprediksi pengajuan kartu kredit mana yang akan disetujui** berdasarkan kriteria kelayakan, dengan tujuan untuk **mengurangi biaya operasional**, **mempercepat proses persetujuan**, dan **meminimalkan risiko kredit**.

Business Metrics

- **Precision**

  Mengukur proporsi aplikasi yang disetujui yang benar-benar layak. Precision yang tinggi memastikan bahwa hanya aplikasi yang benar-benar berkualitas yang disetujui.

  Indicator Precision : Di atas 85% dianggap baik.

- **Recall**

  Mengukur proporsi aplikasi layak yang berhasil diidentifikasi dan disetujui oleh model. Recall yang tinggi memastikan bahwa tidak banyak aplikasi layak yang ditolak.

  Indicator Recall : Di atas 75% dianggap memadai.

- **F1 Score**

Menggabungkan Precision dan Recall untuk memberikan ukuran keseluruhan performa model. F1 Score yang tinggi menunjukkan keseimbangan antara Precision dan Recall.

Indicator F1 Score : Di atas 80% dianggap ideal.

- **Approval Rate**

  Persentase aplikasi yang disetujui terhadap total aplikasi yang diterima

  Indikator AR: 60%-80% menunjukkan portofolio kredit bank yang sehat

References :

- **Thomas, L. C. (2010)**. Consumer Credit Models: Pricing, Profit, and Portfolios. *Oxford University Press*.
- **Hand, D. J., & Henley, W. E. (1997)**. Statistical Classification Methods in Consumer Credit Scoring: A Review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 160(3), 523-541. DOI: 10.1111/1467-985X.00078.
- **Crook, J. N., Edelman, D. B., & Thomas, L. C. (2007)**. Recent developments in consumer credit risk assessment. *European Journal of Operational Research*, 183(3), 1447-1465.
- **Finextra. (2022)**. "AI in Banking: Credit Risk Management". Diakses 25 Agustus 2024. https://www.finextra.com/ai-credit-risk-management
- **Basel Committee on Banking Supervision. (2021)**. "Credit Risk Management Principles". Bank for International Settlements. https://www.bis.org/bcbs/credit_risk
- **Singh, J. (2020)**. "How to Improve Credit Card Approval Models". Journal of Financial Analytics 10(3): 75-89. https://doi.org/10.1108/JFA.2020.03 ___

Table of Contents

====== STAGE 2 ======

`Stage 2 (Data PreProcessing)`

Data Cleansing/Preprocessing

Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math

from datetime import datetime

from scipy.stats import skew
from scipy.stats import kurtosis

# Ignores any warning
import warnings
warnings.filterwarnings("ignore")

sns.set(rc={'figure.figsize':(15, 8)})
sns.set_style("whitegrid")

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.options.display.float_format = lambda x : '{:.0f}'.format(x) if
```

```
round(x,0) == x else '{:.3f}'.format(x)

# from matplotlib import rcParams
# rcParams['figure.figsize'] = 12, 4
# rcParams['lines.linewidth'] = 3
# rcParams['xtick.labelsize'] = 'x-large'
# rcParams['ytick.labelsize'] = 'x-large'

import matplotlib as mp
%matplotlib inline
import textwrap
import matplotlib.ticker as mticker
import matplotlib.patches as mpatches
import matplotlib.gridspec as gridspec
from matplotlib.colors import LinearSegmentedColormap
```

Import Raw Datasets

```
df=pd.read_csv('/kaggle/input/iconicit/datacredit_fe.csv')
df=df.drop('Unnamed: 0', axis=1)
df.head()
```

```
    Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
0  5008827      M         Y             Y         0         180000
1  5009744      F         Y             N         0         315000
2  5009746      F         Y             N         0         315000
3  5009749      F         Y             N         0            NaN
4  5009752      F         Y             N         0         315000

            Type_Income         EDUCATION Marital_status
Housing_type  \
0             Pensioner  Higher education        Married  House /
apartment
1  Commercial associate  Higher education        Married  House /
apartment
2  Commercial associate  Higher education        Married  House /
apartment
3  Commercial associate  Higher education        Married  House /
apartment
4  Commercial associate  Higher education        Married  House /
apartment

   Birthday_count  Employed_days  Mobile_phone  Work_Phone  Phone
EMAIL_ID  \
0          -18772         365243             1           0      0
0
1          -13557           -586             1           1      1
0
2             NaN           -586             1           1      1
0
```

```
3          -13557           -586              1           1      1
0
4          -13557           -586              1           1      1
0

  Type_Occupation  Family_Members  label  Age  Tenure
Unemployment_duration  \
0             NaN               2      1   51       0
365243
1             NaN               2      1   37   1.605
0
2             NaN               2      1  NaN   1.605
0
3             NaN               2      1   37   1.605
0
4             NaN               2      1   37   1.605
0

   Is_currently_employed  Children_to_family_ratio  \
0                      0                         0
1                      1                         0
2                      1                         0
3                      1                         0
4                      1                         0

   Children_employment_impact  Income_per_year_employed Income_sgmt  \
0                           0                         0      Medium
1                           0                196203.072        High
2                           0                196203.072        High
3                           0                         0         Low
4                           0                196203.072        High

      Age_group
0  Senior Adult
1         Adult
2   Young Adult
3         Adult
4         Adult

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 28 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Ind_ID                       1548 non-null   int64
 1   GENDER                       1541 non-null   object
 2   Car_Owner                    1548 non-null   object
 3   Propert_Owner                1548 non-null   object
```

```
 4   CHILDREN                     1548 non-null   int64
 5   Annual_income                1525 non-null   float64
 6   Type_Income                  1548 non-null   object
 7   EDUCATION                    1548 non-null   object
 8   Marital_status               1548 non-null   object
 9   Housing_type                 1548 non-null   object
 10  Birthday_count               1526 non-null   float64
 11  Employed_days                1548 non-null   int64
 12  Mobile_phone                 1548 non-null   int64
 13  Work_Phone                   1548 non-null   int64
 14  Phone                        1548 non-null   int64
 15  EMAIL_ID                     1548 non-null   int64
 16  Type_Occupation              1060 non-null   object
 17  Family_Members               1548 non-null   int64
 18  label                        1548 non-null   int64
 19  Age                          1526 non-null   float64
 20  Tenure                       1548 non-null   float64
 21  Unemployment_duration        1548 non-null   int64
 22  Is_currently_employed        1548 non-null   int64
 23  Children_to_family_ratio     1548 non-null   float64
 24  Children_employment_impact   1548 non-null   float64
 25  Income_per_year_employed     1548 non-null   float64
 26  Income_sgmt                  1548 non-null   object
 27  Age_group                    1548 non-null   object
dtypes: float64(7), int64(11), object(10)
memory usage: 338.8+ KB
```

```
cat_cols = ['GENDER', 'Car_Owner', 'Propert_Owner', 'Type_Income',
'EDUCATION',
            'Marital_status', 'Housing_type', 'Type_Occupation',
'Income_sgmt',
            'Age_group']
num_cols = ['CHILDREN', 'Annual_income', 'Birthday_count',
'Employed_days',
            'Family_Members', 'Age', 'Tenure',
'Unemployment_duration',
            'Children_to_family_ratio', 'Children_employment_impact',
            'Income_per_year_employed']
```

Handling Duplicate Rows

```
df[df.duplicated(keep=False)].sort_values(by=list(df.columns.values)).
head()
```

```
Empty DataFrame
Columns: [Ind_ID, GENDER, Car_Owner, Propert_Owner, CHILDREN,
Annual_income, Type_Income, EDUCATION, Marital_status, Housing_type,
Birthday_count, Employed_days, Mobile_phone, Work_Phone, Phone,
EMAIL_ID, Type_Occupation, Family_Members, label, Age, Tenure,
```

```
Unemployment_duration, Is_currently_employed,
Children_to_family_ratio, Children_employment_impact,
Income_per_year_employed, Income_sgmt, Age_group]
Index: []
```

```
df.duplicated().sum()
```

```
0
```

```
print(f"Data Frame Dimension Before Duplicate Removal: {df.shape}")
df = df.drop_duplicates().reset_index(drop=True)
print(f"Data Frame Dimension After Duplicate Removal: {df.shape}")
```

```
Data Frame Dimension Before Duplicate Removal: (1548, 28)
Data Frame Dimension After Duplicate Removal: (1548, 28)
```

```
df.duplicated(subset=["Ind_ID"]).sum()
```

```
0
```

Kesimpulan

- Berdasarkan hasil pengecekan, tidak ditemui baris data yang memiliki duplikat. Sehingga kami tidak perlu melakukan handling duplicated data
- Pada pengecekan duplikat subset untuk ID tidak ditemukan ada nya ID customer yang sama

Handling Invalid Values

Mengecek isi unique values dari data

```
for x in df.columns:
    unq = df[x].astype(str).unique()
    unq_sorted = sorted(unq)
    print(f'===== {x} =====')

    if len(unq_sorted) >= 10:
        unq_display = list(unq_sorted[:10]) + ['.....']
        print(f'{unq_display}')
    else:
        print(f'{unq_sorted}')

    print()
```

```
===== Ind_ID =====
['5008827', '5008865', '5008889', '5009000', '5009023', '5009053',
'5009074', '5009118', '5009146', '5009195', '.....']

===== GENDER =====
['F', 'M', 'nan']
```

```
===== Car_Owner =====
['N', 'Y']

===== Propert_Owner =====
['N', 'Y']

===== CHILDREN =====
['0', '1', '14', '2', '3', '4']

===== Annual_income =====
['103500.0', '105750.0', '108000.0', '112500.0', '114750.0',
'115200.0', '116100.0', '117000.0', '119250.0', '119700.0', '.....']

===== Type_Income =====
['Commercial associate', 'Pensioner', 'State servant', 'Working']

===== EDUCATION =====
['Academic degree', 'Higher education', 'Incomplete higher', 'Lower
secondary', 'Secondary / secondary special']

===== Marital_status =====
['Civil marriage', 'Married', 'Separated', 'Single / not married',
'Widow']

===== Housing_type =====
['Co-op apartment', 'House / apartment', 'Municipal apartment',
'Office apartment', 'Rented apartment', 'With parents']

===== Birthday_count =====
['-10004.0', '-10014.0', '-10020.0', '-10021.0', '-10028.0', '-
10035.0', '-10049.0', '-10066.0', '-10072.0', '-10075.0', '.....']

===== Employed_days =====
['-1000', '-1002', '-1007', '-101', '-1017', '-102', '-1022', '-1023',
'-1032', '-10364', '.....']

===== Mobile_phone =====
['1']

===== Work_Phone =====
['0', '1']

===== Phone =====
['0', '1']

===== EMAIL_ID =====
['0', '1']

===== Type_Occupation =====
['Accountants', 'Cleaning staff', 'Cooking staff', 'Core staff',
'Drivers', 'HR staff', 'High skill tech staff', 'IT staff',
```

```
'Laborers', 'Low-skill Laborers', '.....']

===== Family_Members =====
['1', '15', '2', '3', '4', '5', '6']

===== label =====
['0', '1']

===== Age =====
['21.0', '22.0', '23.0', '24.0', '25.0', '26.0', '27.0', '28.0',
'29.0', '30.0', '.....']

===== Tenure =====
['0.0', '0.2', '0.2410958904109589', '0.2438356164383561',
'0.2602739726027397', '0.2630136986301369', '0.2657534246575342',
'0.2767123287671232', '0.2794520547945205', '0.284931506849315',
'.....']

===== Unemployment_duration =====
['0', '365243']

===== Is_currently_employed =====
['0', '1']

===== Children_to_family_ratio =====
['0.0', '0.3333333333333333', '0.5', '0.6', '0.6666666666666666',
'0.75', '0.9333333333333332', '1.0']

===== Children_employment_impact =====
['0.0', '0.2410958904109589', '0.284931506849315',
'0.2986301369863013', '0.3013698630136986', '0.3205479452054794',
'0.3397260273972602', '0.3452054794520548', '0.3671232876712328',
'0.3698630136986301', '.....']

===== Income_per_year_employed =====
['0.0', '10036.66361136572', '100698.2288828338',
'100973.36065573768', '101100.85227272726', '10141.671240395171',
'101657.824933687', '101667.8129298487', '101772.80550774526',
'10220.773202700911', '.....']

===== Income_sgmt =====
['High', 'Low', 'Medium']

===== Age_group =====
['Adult', 'Senior Adult', 'Young Adult']
```

1. Melakukan penyederhanaan `Marital_Status`

Akan dilakukan replace data / menyatukan yang memiliki arti yang sama agar mengurangi jumlah dimensi maupun redudansi pada data

- Mengganti kategori `Widow`, `Separated` menjadi `Divorce`
- Mengganti kategori `Civil marriage` menjadi `Married`
- Mempertahankan kategori `Single/not married` menjadi `Single`

```
marital_status_mapping = {
    'Civil marriage': 'Married',
    'Married': 'Married',
    'Separated': 'Separated/Widow',
    'Widow': 'Separated/Widow',
    'Single / not married': 'Single'
}
df['Marital_status']= df['Marital_status'].map(marital_status_mapping)
```

2. Melakukan penyederhanaan `Education_Simple`

Untuk kategori `Academic Degree` dan `Higher Education` juga kurang lebih sama. Maka dari itu, baris yang memiliki kategori `Academic degree` akan dihapus dan digantikan dengan kategori `Higher Education`.

```
df['EDUCATION'] = df['EDUCATION'].replace(['Academic degree'],'Higher education')
```

3. Menghapus kolom `Mobile_phone`

Untuk kolom `Mobile_phone` karena hanya ada 1 nilai maka kita akan menghapus datanya.

```
df=df.drop('Mobile_phone',axis=1)

df.head()
```

```
    Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
0  5008827      M         Y              Y         0         180000
1  5009744      F         Y              N         0         315000
2  5009746      F         Y              N         0         315000
3  5009749      F         Y              N         0            NaN
4  5009752      F         Y              N         0         315000

            Type_Income          EDUCATION Marital_status
Housing_type  \
0             Pensioner  Higher education        Married  House /
apartment
1  Commercial associate  Higher education        Married  House /
apartment
2  Commercial associate  Higher education        Married  House /
```

```
apartment
3  Commercial associate  Higher education        Married  House /
apartment
4  Commercial associate  Higher education        Married  House /
apartment

   Birthday_count  Employed_days  Work_Phone  Phone  EMAIL_ID
Type_Occupation  \
0          -18772         365243           0      0         0
NaN
1          -13557           -586           1      1         0
NaN
2             NaN           -586           1      1         0
NaN
3          -13557           -586           1      1         0
NaN
4          -13557           -586           1      1         0
NaN

   Family_Members  label  Age  Tenure  Unemployment_duration  \
0               2      1   51       0                 365243
1               2      1   37   1.605                      0
2               2      1  NaN   1.605                      0
3               2      1   37   1.605                      0
4               2      1   37   1.605                      0

   Is_currently_employed  Children_to_family_ratio  \
0                      0                         0
1                      1                         0
2                      1                         0
3                      1                         0
4                      1                         0

   Children_employment_impact  Income_per_year_employed Income_sgmt  \
0                           0                         0      Medium
1                           0                196203.072        High
2                           0                196203.072        High
3                           0                         0         Low
4                           0                196203.072        High

      Age_group
0  Senior Adult
1         Adult
2   Young Adult
3         Adult
4         Adult
```

5. Melakukan Mapping sesuai risiko kredit

```
# Mapping untuk Type_Income
type_income_mapping = {
    'Commercial associate': 4,
    'State servant': 3,
    'Working': 2,
    'Pensioner': 1
}

# Mapping untuk EDUCATION
education_mapping = {
    'Higher education': 4,
    'Secondary / secondary special': 3,
    'Incomplete higher': 2,
    'Lower secondary': 1
}

# Mapping untuk Marital_status
marital_status_mapping = {
    'Married': 3,
    'Separated/Widow': 2,
    'Single': 1
}

# Mapping untuk Housing_type
housing_type_mapping = {
    'House / apartment': 6,
    'Co-op apartment': 5,
    'Municipal apartment': 4,
    'Office apartment': 3,
    'Rented apartment': 2,
    'With parents': 1
}

# Mapping untuk GENDER
gender_mapping = {
    'M': 0,
    'F': 1
}

# Mapping untuk Car_Owner
car_owner_mapping = {
    'N': 0,
    'Y': 1
}

# Mapping untuk Propert_Owner
propert_owner_mapping = {
    'N': 0,
```

```
    'Y': 1
}

# Mapping untuk Income_sgmt
income_sgmt_mapping = {
    'H': 1,
    'Medium': 0,
    'Low': -1
}

# Mapping untuk Age_group
age_group_mapping = {
    'Senior Adult': 1,
    'Adult': 0,
    'Young Adult': -1
}

# Mapping untuk Type_Occupation
type_occupation_mapping = {
    'Managers': 18,
    'High skill tech staff': 17,
    'IT staff': 16,
    'Accountants': 15,
    'HR staff': 14,
    'Core staff': 13,
    'Medicine staff': 12,
    'Sales staff': 11,
    'Realty agents': 10,
    'Secretaries': 9,
    'Private service staff': 8,
    'Security staff': 7,
    'Drivers': 6,
    'Cooking staff': 5,
    'Cleaning staff': 4,
    'Waiters/barmen staff': 3,
    'Laborers': 2,
    'Low-skill Laborers': 1
}
df['GENDER'] = df['GENDER'].map(gender_mapping)
df['Car_Owner'] = df['Car_Owner'].map(car_owner_mapping)
df['Propert_Owner'] = df['Propert_Owner'].map(propert_owner_mapping)
df['Income_sgmt'] = df['Income_sgmt'].map(income_sgmt_mapping)
df['Age_group'] = df['Age_group'].map(age_group_mapping)
df['Type_Income'] = df['Type_Income'].map(type_income_mapping)
df['EDUCATION'] = df['EDUCATION'].map(education_mapping)
df['Marital_status'] =
df['Marital_status'].map(marital_status_mapping)
df['Housing_type'] = df['Housing_type'].map(housing_type_mapping)
df['Type_Occupation'] =
df['Type_Occupation'].map(type_occupation_mapping)
```

Kesimpulan

Pada `Marital_Status` dan `Education` replace data / menyatukan yang memiliki arti yang sama agar mengurangi jumlah dimensi maupun redudansi pada data.

Data Splitting

Data preparation adalah proses mengubah data mentah menjadi bentuk yang sesuai untuk pemodelan.

Salah satu masalah dalam `Data Preprocessing` serta menerapkan `Transformasi` pada `Seluruh Kumpulan Data` dapat menyebabkan masalah yang disebut sebagai `data leakage` (kebocoran data). Di mana pengetahuan tentang kumpulan pengujian (data test) bocor ke dalam kumpulan data yang digunakan untuk melatih model (data train). Hal ini dapat mengakibatkan estimasi performa model yang salah/bias saat membuat prediksi pada data baru.

Data preparation harus dilakukan `fit` dengan `training dataset` saja. Artinya, setiap koefisien atau model yang disiapkan untuk proses penyiapan data hanya boleh menggunakan deretan data dalam training dataset.

Setelah melakukan `fit`, metode data preparation kemudian dapat diterapkan `apply / transform` ke data train, dan ke data test.

1. Split Data.
2. Fit Data Preparation on Training Dataset.
3. Apply Data Preparation to Train and Test Datasets.
4. Evaluate Models.

Pada proses `Split Data` ini, kami akan membagi nya menjadi `training set` dan `testing set` dengan proporsi `75:25`

```
from sklearn.model_selection import train_test_split

# splitting tha data
df_train, df_test = train_test_split(df, test_size=0.25,
stratify=df[['label']], random_state=42)
df_train.reset_index(drop=True, inplace=True)
df_test.reset_index(drop=True, inplace=True)

print(df_train.shape)
print(df_test.shape)
```

```
(1161, 27)
(387, 27)
```

```
df_train.head()
```

```
    Ind_ID  GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
\
```

```
0  5024636       1          0              1         0             NaN

1  5023692       1          0              1         0          135000

2  5148525       1          0              1         0          180000

3  5028446       1          1              1         0          162000

4  5069289       1          0              1         0          270000


   Type_Income  EDUCATION  Marital_status  Housing_type
Birthday_count  \
0            2          3               3             6          -
19922
1            2          3               1             6          -
15036
2            1          3               3             6          -
23365
3            2          3               1             6          -
18071
4            4          2               3             6          -
14783

   Employed_days  Work_Phone  Phone  EMAIL_ID  Type_Occupation  \
0          -1128           0      0         0              NaN
1          -3339           0      0         0              NaN
2         365243           0      0         0              NaN
3          -2822           0      0         0                2
4          -3397           0      0         0               13

   Family_Members  label  Age  Tenure  Unemployment_duration  \
0               2      0   54   3.090                      0
1               1      0   41   9.148                      0
2               2      0   64       0                 365243
3               1      0   49   7.732                      0
4               2      0   40   9.307                      0

   Is_currently_employed  Children_to_family_ratio  \
0                      1                         0
1                      1                         0
2                      0                         0
3                      1                         0
4                      1                         0

   Children_employment_impact  Income_per_year_employed
Income_sgmt  \
0                           0                         0          -1

1                           0                14757.412            0
```

```
2                           0                         0            0

3                           0                 20953.225            0

4                           0                 29010.892          NaN


   Age_group
0          1
1          0
2          1
3          1
4          0

df_test.head()

    Ind_ID  GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
\
0  5047670       1          0              0         1         180000

1  5135344       0          0              1         0         112500

2  5033728       1          1              1         1         202500

3  5036795       0          1              1         0         215100

4  5118115       0          0              0         1         270000


   Type_Income  EDUCATION  Marital_status  Housing_type
Birthday_count  \
0            3          4               3             6          -
14069
1            4          4               3             6          -
20103
2            2          3               3             6          -
13666
3            4          3               3             6          -
19175
4            4          3               3             6          -
13538

   Employed_days  Work_Phone  Phone  EMAIL_ID  Type_Occupation  \
0          -2500           1      1         1               13
1           -555           0      0         0                7
2          -2089           1      1         0               13
3          -1285           0      1         0                2
4          -5422           1      1         0                6

   Family_Members  label  Age  Tenure  Unemployment_duration  \
```

```
0                3      0   38   6.849                        0
1                2      1   55   1.521                        0
2                3      0   37   5.723                        0
3                2      1   52   3.521                        0
4                3      0   37  14.855                        0

   Is_currently_employed  Children_to_family_ratio  \
0                      1                     0.333
1                      1                         0
2                      1                     0.333
3                      1                         0
4                      1                     0.333

   Children_employment_impact  Income_per_year_employed
Income_sgmt  \
0                       6.849                     26280             0

1                           0                 73986.486            -1

2                       5.723                 35381.762             0

3                           0                 61098.444             0

4                      14.855                 18175.950           NaN


   Age_group
0          0
1          1
2          0
3          1
4          0
```

Handling Missing Value

Melakukan pengecekan jumlah Missing Values dan Persentase nya

```
print('Missing values status:', df.isnull().values.any())
nvc = pd.DataFrame(df.isnull().sum(), columns=['Total Null Values'])
nvc['Percentage'] = (nvc['Total Null Values']/df.shape[0])*100
nvc["Data Type"] = [df[col].dtype for col in df.columns]
nvc["NULL Train"] = df_train[nvc.index].isnull().sum()
nvc["NULL Test"] = df_test[nvc.index].isnull().sum()
nvc.sort_values(by=["Total Null Values", "Percentage"],
ascending=False, inplace=True)
nvc
```

```
Missing values status: True
```

	Total Null Values	Percentage	Data Type
Type_Occupation	488	31.525	float64
Income_sgmt	460	29.716	float64
Annual_income	23	1.486	float64
Birthday_count	22	1.421	float64
Age	22	1.421	float64
GENDER	7	0.452	float64
Ind_ID	0	0	int64
Car_Owner	0	0	int64
Propert_Owner	0	0	int64
CHILDREN	0	0	int64
Type_Income	0	0	int64
EDUCATION	0	0	int64
Marital_status	0	0	int64
Housing_type	0	0	int64
Employed_days	0	0	int64
Work_Phone	0	0	int64
Phone	0	0	int64
EMAIL_ID	0	0	int64
Family_Members	0	0	int64
label	0	0	int64
Tenure	0	0	float64
Unemployment_duration	0	0	int64
Is_currently_employed	0	0	int64
Children_to_family_ratio	0	0	float64
Children_employment_impact	0	0	float64
Income_per_year_employed	0	0	float64
Age_group	0	0	int64

	NULL Train	NULL Test
Type_Occupation	373	115
Income_sgmt	338	122
Annual_income	15	8
Birthday_count	21	1
Age	21	1
GENDER	5	2
Ind_ID	0	0
Car_Owner	0	0
Propert_Owner	0	0
CHILDREN	0	0
Type_Income	0	0
EDUCATION	0	0
Marital_status	0	0
Housing_type	0	0
Employed_days	0	0
Work_Phone	0	0
Phone	0	0
EMAIL_ID	0	0
Family_Members	0	0
label	0	0

```
Tenure                              0          0
Unemployment_duration               0          0
Is_currently_employed               0          0
Children_to_family_ratio            0          0
Children_employment_impact          0          0
Income_per_year_employed            0          0
Age_group                           0          0
```

Observations:

- **Kolom `Type_Occupation`**:
  - Terdapat 488 baris data kosong (373 pada data train, 115 pada data test) dengan persentase sebesar 31,525% dari keseluruhan data.
- **Kolom `Annual_income`**:
  - Terdapat 23 baris data kosong (15 pada data train, 8 pada data test) dengan persentase sebesar 1,486% dari keseluruhan data.
- **Kolom `Birthday_count` dan `Age`**:
  - Masing-masing kolom memiliki 22 baris data kosong (21 pada data train, 1 pada data test) dengan persentase sebesar 1,421% dari keseluruhan data.
- **Kolom `GENDER`**:
  - Terdapat 7 baris data kosong (5 pada data train, 2 pada data test) dengan persentase sebesar 0,452% dari keseluruhan data.

Untuk menangani missing values pada kolom-kolom di atas, beberapa metode yang dapat dilakukan adalah:

- Drop Rows Missing Values

- Cara ini efektif jika jumlah data kosong relatif kecil dan tidak terlalu signifikan dalam mempengaruhi analisis. Namun, tidak direkomendasikan untuk kolom `Type_Occupation` karena tingginya persentase data kosong.

- Imputation Median

- Mengisi data kosong dengan nilai median dapat menjadi solusi untuk kolom numerik seperti `Annual_income`, `Birthday_count`, dan `Age`.
- Metode:
  - Menggunakan fungsi `fillna` atau `SimpleImputer` dari `sklearn`.

- Multivariate Approach

- Pendekatan ini lebih kompleks namun efektif, terutama jika ingin mempertahankan sebanyak mungkin data. Cocok untuk kolom dengan tipe data numerik.
- Persiapan:
  - Pastikan semua data dalam bentuk numerik.
  - Drop kolom yang tidak relevan seperti data tanggal jika ada.
- Metode:
  - `KNNImputer` atau K-Nearest Neighbor.
  - `MICE` atau Multiple Imputation by Chained Equation:

- Menggunakan `IterativeImputer`.
- Menggunakan MICE dengan model `LightGBM`.

Drop Rows

karena data yang kita miliki terbatas maka untuk proses ini tidak akan kita gunakan

```
# print("Jumlah rows sebelum drop missing values (train set) =",
df_train.shape[0])
# print("Jumlah rows sebelum drop missing values (test set) =",
df_test.shape[0])

# columns_to_dropna = ['Type_Occupation', 'Annual_income',
'Birthday_count', 'Age', 'GENDER']

# df_train = df_train.dropna(subset=columns_to_dropna)
# df_test = df_test.dropna(subset=columns_to_dropna)

# print("Jumlah rows setelah drop missing values (train set) =",
df_train.shape[0])
# print("Jumlah rows setelah drop missing values (test set) =",
df_test.shape[0])
```

Pros

- Straightforward and simple to use.
- Beneficial when missing values have no importance.

Cons

- Using this approach can lead to information loss, which can introduce bias to the final dataset.
- Data set with a large proportion of missing value can be significantly decreased, which can impact the result of all statistical analysis on that data set.

Imputation

Imputation (Median), karena Highly Positively Skewed

Imputation using `fillna`

```
# print("Jumlah rows sebelum imputation (train set) =",
df_train.shape[0])
# print("Jumlah rows sebelum imputation (test set) =",
df_test.shape[0])

# df_train['Annual_income'].fillna(df_train['Annual_income'].median(),
inplace=True)
#
df_train['Birthday_count'].fillna(df_train['Birthday_count'].median(),
inplace=True)
```

```
# df_train['Age'].fillna(df_train['Age'].median(), inplace=True)

# df_test['Annual_income'].fillna(df_test['Annual_income'].median(),
inplace=True)
# df_test['Birthday_count'].fillna(df_test['Birthday_count'].median(),
inplace=True)
# df_test['Age'].fillna(df_test['Age'].median(), inplace=True)

# print("Jumlah rows setelah imputation missing values (train set) =",
df_train.shape[0])
# print("Jumlah rows setelah imputation missing values (test set) =",
df_test.shape[0])
```

Imputation using `SimpleImputer`

```
# print("Jumlah rows sebelum imputation (train set) =",
df_train.shape[0])
# print("Jumlah rows sebelum imputation (test set) =",
df_test.shape[0])

# median_imputer = SimpleImputer(strategy='median')
# df_train[['Annual_income', 'Birthday_count', 'Age']] =
median_imputer.fit_transform(df_train[['Annual_income',
'Birthday_count', 'Age']])
# df_test[['Annual_income', 'Birthday_count', 'Age']] =
median_imputer.transform(df_test[['Annual_income', 'Birthday_count',
'Age']])

# print("Jumlah rows setelah imputation missing values (train set) =",
df_train.shape[0])
# print("Jumlah rows setelah imputation missing values (test set) =",
df_test.shape[0])
```

Pros

- Simplicity and ease of implementation are some of the benefits of the mean and median imputation.
- The imputation is performed using the existing information from the non-missing data; hence no additional data is required.
- Mean and median imputation can provide a good estimate of the missing values, respectively for normally distributed data, and skewed data.

Cons

- We cannot apply these two strategies to categorical columns. They can only work for numerical ones.
- Mean imputation is sensitive to outliers and may not be a good representation of the central tendency of the data. Similarly to the mean, the median also may not better represent the central tendency.

Multivariate Approach

Multivariate Approach (MICE Imputation, KNN Imputer, dll)

Multiple imputations compensate for missing data and produce multiple datasets by regression model and are considered the solver of the old problem of univariate imputation. The univariate imputes data only from a specific column where the data cell was missing. Multivariate imputation works simultaneously, with all variables in all columns, whether missing or observed. It has emerged as a principal method of solving missing data problems. All incomplete datasets analyzed before Multiple Imputation by Chained Equations (MICE) presented were misdiagnosed; results obtained were invalid and should not be countable to yield reasonable conclusions.

In multiple imputations, following a typical model tends to preserve variances, means, covariance, linear regression coefficients, and correlation. MI, therefore, was designed to restore variability that is lost during single imputation and consequently correct it. Rounding procedures included increasing the variability of imputed values.

Multiple imputations help to ensure that the missing data is uncertain by generating various reasonable imputed data sets and integrating findings from each of them properly. Multiple imputations employ the imputation, analysis, and pooling processes

The advantages of multiple imputation are :

- Results in unbiased estimates, providing more validity than ad hoc approaches to missing data
- Uses all available data, preserving sample size and statistical power

Pros

- Multiple imputation is powerful at dealing with missing data in multiple variables and multiple data types.
- The approach can produce much better results than mean and median imputations.
- Many other algorithms, such as K-Nearest Neighbors, Random forest, and neural networks, can be used as the backbone of the multiple imputation prediction for making predictions.

Cons

- Multiple imputation assumes that the data is missing at random (MAR).
- Despite all the benefits, this approach can be computationally expensive compared to other techniques, especially when working with large datasets.
- This approach requires more effort than the previous ones.

Dikarenakan pada proses multivariate approach memerlukan semua feature bertipe numerical sehingga untuk data yang masih bertipe object/string dan date akan di drop pada data temp

```
df_ma_train = df_train.copy()
df_ma_test = df_test.copy()

df_ma_train.head()
```

```
    Ind_ID  GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
\
0  5024636       1          0              1         0            NaN

1  5023692       1          0              1         0         135000

2  5148525       1          0              1         0         180000

3  5028446       1          1              1         0         162000

4  5069289       1          0              1         0         270000


   Type_Income  EDUCATION  Marital_status  Housing_type
Birthday_count  \
0            2          3               3             6          -
19922
1            2          3               1             6          -
15036
2            1          3               3             6          -
23365
3            2          3               1             6          -
18071
4            4          2               3             6          -
14783

   Employed_days  Work_Phone  Phone  EMAIL_ID  Type_Occupation  \
0          -1128           0      0         0              NaN
1          -3339           0      0         0              NaN
2         365243           0      0         0              NaN
3          -2822           0      0         0                2
4          -3397           0      0         0               13

   Family_Members  label  Age  Tenure  Unemployment_duration  \
0               2      0   54   3.090                      0
1               1      0   41   9.148                      0
2               2      0   64       0                 365243
3               1      0   49   7.732                      0
4               2      0   40   9.307                      0

   Is_currently_employed  Children_to_family_ratio  \
0                      1                         0
1                      1                         0
2                      0                         0
3                      1                         0
4                      1                         0

   Children_employment_impact  Income_per_year_employed
Income_sgmt  \
0                           0                         0           -1
```

```
1                                0                 14757.412              0
2                                0                       0                0
3                                0                 20953.225              0
4                                0                 29010.892            NaN
```

```
   Age_group
0          1
1          0
2          1
3          1
4          0
```

Implementation Multivariate Approach

Two main methods we use here to impute missing values

- `KNN Imputer` or K-Nearest Neighbor
- `MICE` or Multiple Imputation by Chained Equation

Here, Multiple imputations are performed using `sklearn` and `fancyimpute`.

To install fancyimpute

```
pip install fancyimpute
```

Imputation using `KNNImputer`

```
# print("Jumlah rows sebelum imputation (train set) =",
df_train.shape[0])
# print("Jumlah rows sebelum imputation (test set) =",
df_test.shape[0])

# df_train_to_impute = df_train[columns_to_impute].copy()
# df_test_to_impute = df_test[columns_to_impute].copy()

# knn_imputer = KNNImputer(n_neighbors=5)

# df_imputed_train = pd.DataFrame(
#     knn_imputer.fit_transform(df_train_to_impute),
#     columns=df_train_to_impute.columns
# )
# df_train[columns_to_impute] =
df_imputed_train[columns_to_impute].copy()

# df_imputed_test = pd.DataFrame(
#     knn_imputer.transform(df_test_to_impute),
#     columns=df_test_to_impute.columns
```

```
# )
# df_test[columns_to_impute] =
df_imputed_test[columns_to_impute].copy()

# print("Jumlah rows setelah imputation missing values (train set) =",
df_train.shape[0])
# print("Jumlah rows setelah imputation missing values (test set) =",
df_test.shape[0])
```

pada fancyimpute KNN Imputer hanya ada fungsi fit transform sehingga tidak digunakan pada split data test

```
from fancyimpute import KNN
knn_imputer.transform(df_ma_train)
```

Imputation using `MICE` with `IterativeImputer`

MICE Imputation, short for 'Multiple Imputation by Chained Equation' is an advanced missing data imputation technique that uses multiple iterations of Machine Learning model training to predict the missing values using known values from other features in the data as predictors.

```
# print("Jumlah rows sebelum imputation (train set) =",
df_train.shape[0])
# print("Jumlah rows sebelum imputation (test set) =",
df_test.shape[0])

# # Note: make sure to import the enable_iterative_imputer function
before
# # you import the IterativeImputer, as the feature is classified as
# # experimental, and failing to do so will result in an ImportError.
# from sklearn.experimental import enable_iterative_imputer
# from sklearn.impute import IterativeImputer
# imputer = IterativeImputer(max_iter=10, random_state=0)

# # Train
# df_imputed_train = pd.DataFrame(
#     imputer.fit_transform(df_ma_train),
#     columns=df_ma_train.columns
# )

# df_train[columns_to_impute] =
df_imputed_train[columns_to_impute].copy()

# # Test
# df_imputed_test = pd.DataFrame(
#     imputer.transform(df_ma_test),
#     columns=df_ma_test.columns
# )
```

```
# df_test[columns_to_impute] =
df_imputed_test[columns_to_impute].copy()

# from fancyimpute import IterativeImputer
# mice_imputer = IterativeImputer()
# # filling the missing value with mice imputer

# # Train
# df_imputed_train = pd.DataFrame(
#     mice_imputer.fit_transform(df_ma_train),
#     columns=df_ma_train.columns
# )

# df_train[columns_to_impute] =
df_imputed_train[columns_to_impute].copy()

# # Test
# df_imputed_test = pd.DataFrame(
#     mice_imputer.transform(df_ma_test),
#     columns=df_ma_test.columns
# )

# df_test[columns_to_impute] =
df_imputed_test[columns_to_impute].copy()

# print("Jumlah rows setelah imputation missing values (train set) =",
df_train.shape[0])
# print("Jumlah rows setelah imputation missing values (test set) =",
df_test.shape[0])
```

Imputation using `MICE` with `LightGBM`

MICE imputation can be made more efficient using the `miceforest` package. It is expected to perform significantly better because it implements `lightgbm` algorithm in the backend to do the imputation.

LightGBM splits the tree leaf-wise as opposed to other boosting algorithms that grow tree level-wise. LightGBM is a gradient boosting ensemble method that is used by the Train Using AutoML tool and is based on decision trees.

This package can be installed using either pip or conda, through conda-forge:

- Using pip

  `pip install miceforest --no-cache-dir`

- Using conda

  `conda install -c conda-forge miceforest`

```
%%capture
!pip install miceforest --no-cache-dir

print("Jumlah rows sebelum imputation (train set) =",
df_train.shape[0])
print("Jumlah rows sebelum imputation (test set) =", df_test.shape[0])
```

```
Jumlah rows sebelum imputation (train set) = 1161
Jumlah rows sebelum imputation (test set) = 387
```

```
import miceforest as mf

df_ma_train_amp = mf.ampute_data(df_ma_train,
perc=0.25,random_state=1991)
columns_to_impute = ['Type_Occupation', 'GENDER','Annual_income',
'Birthday_count', 'Age']
# Train

# Create kernel.
kds = mf.ImputationKernel(
  data = df_ma_train,
  save_all_iterations_data=True,
  random_state=1991
)

# Run the MICE algorithm
kds.mice(iterations=5, n_estimators=50)

# Return the completed dataset.
df_imputed_train = kds.complete_data()
df_train[columns_to_impute] =
df_imputed_train[columns_to_impute].copy()

# Test
new_data_imputed = kds.impute_new_data(df_ma_test)
# Return a completed dataset
df_imputed_test = new_data_imputed.complete_data(0)
df_test[columns_to_impute] = df_imputed_test[columns_to_impute].copy()

print("Jumlah rows setelah imputation missing values (train set) =",
df_train.shape[0])
print("Jumlah rows setelah imputation missing values (test set) =",
df_test.shape[0])
```

```
Jumlah rows setelah imputation missing values (train set) = 1161
Jumlah rows setelah imputation missing values (test set) = 387
```

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1161 entries, 0 to 1160
```

```
Data columns (total 27 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Ind_ID                       1161 non-null   int64  
 1   GENDER                       1161 non-null   float64
 2   Car_Owner                    1161 non-null   int64  
 3   Propert_Owner                1161 non-null   int64  
 4   CHILDREN                     1161 non-null   int64  
 5   Annual_income                1161 non-null   float64
 6   Type_Income                  1161 non-null   int64  
 7   EDUCATION                    1161 non-null   int64  
 8   Marital_status               1161 non-null   int64  
 9   Housing_type                 1161 non-null   int64  
 10  Birthday_count               1161 non-null   float64
 11  Employed_days                1161 non-null   int64  
 12  Work_Phone                   1161 non-null   int64  
 13  Phone                        1161 non-null   int64  
 14  EMAIL_ID                     1161 non-null   int64  
 15  Type_Occupation              1161 non-null   float64
 16  Family_Members               1161 non-null   int64  
 17  label                        1161 non-null   int64  
 18  Age                          1161 non-null   float64
 19  Tenure                       1161 non-null   float64
 20  Unemployment_duration        1161 non-null   int64  
 21  Is_currently_employed        1161 non-null   int64  
 22  Children_to_family_ratio     1161 non-null   float64
 23  Children_employment_impact   1161 non-null   float64
 24  Income_per_year_employed     1161 non-null   float64
 25  Income_sgmt                  823 non-null    float64
 26  Age_group                    1161 non-null   int64  
dtypes: float64(10), int64(17)
memory usage: 245.0 KB

df_test.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 387 entries, 0 to 386
Data columns (total 27 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Ind_ID                       387 non-null    int64  
 1   GENDER                       387 non-null    float64
 2   Car_Owner                    387 non-null    int64  
 3   Propert_Owner                387 non-null    int64  
 4   CHILDREN                     387 non-null    int64  
 5   Annual_income                387 non-null    float64
 6   Type_Income                  387 non-null    int64  
 7   EDUCATION                    387 non-null    int64  
 8   Marital_status               387 non-null    int64  
 9   Housing_type                 387 non-null    int64  
```

```
 10  Birthday_count               387 non-null    float64
 11  Employed_days                387 non-null    int64
 12  Work_Phone                   387 non-null    int64
 13  Phone                        387 non-null    int64
 14  EMAIL_ID                     387 non-null    int64
 15  Type_Occupation              387 non-null    float64
 16  Family_Members               387 non-null    int64
 17  label                        387 non-null    int64
 18  Age                          387 non-null    float64
 19  Tenure                       387 non-null    float64
 20  Unemployment_duration        387 non-null    int64
 21  Is_currently_employed        387 non-null    int64
 22  Children_to_family_ratio     387 non-null    float64
 23  Children_employment_impact   387 non-null    float64
 24  Income_per_year_employed     387 non-null    float64
 25  Income_sgmt                  265 non-null    float64
 26  Age_group                    387 non-null    int64
dtypes: float64(10), int64(17)
memory usage: 81.8 KB
```

```
Q1 = df["Annual_income"].quantile(.25)
print(Q1)
Q3 = df["Annual_income"].quantile(.75)
print(Q3)

def income_sgmt(x):
    if (x is None) or (type(x) not in [int, float]) :
        segment = "None"
    else:
        if x >= Q3:
            segment = "High"
        elif x < Q3 and x >= Q1:
            segment = "Medium"
        else:
            segment = "Low"
    return segment

df["Income_sgmt"] = df["Annual_income"].apply(lambda x:
income_sgmt(x))
df_train["Income_sgmt"] = df_train["Annual_income"].apply(lambda x:
income_sgmt(x))
df_test["Income_sgmt"] = df_test["Annual_income"].apply(lambda x:
income_sgmt(x))
```

```
121500.0
225000.0
```

Kesimpulan

Berdasarkan hasil pengecekan, Untuk kolom `Type_Occupation` terdapat missing values 31%, diikuti oleh Income_sgmt, Annual_income, birthday count, age, dan gender. Dikarenakan data kita terbatas, sehingga untuk prosesnya kita tidak akan melakukan penghapusan baris (Drop Rows), melainkan dilakukan proses Imputation.

Pada proses handling missing values ini kita menggunakan `Imputation using MICE with LightGBM`. Imputasi MICE (Multiple Imputasi by Chained Equation) dapat lebih efisien karena menggunakan beberapa iterasi pelatihan model Machine Learning dan menggunakan data yang tersedia di fitur lain untuk memperkirakan nilai dari missing values yang diperhitungkan oleh karena itu diharapkan kinerjanya jauh lebih baik. Selain itu, Imputasi MICE dapat lebih efisien menggunakan `miceforest` karena diharapkan kinerjanya jauh lebih baik karena mengimplementasikan algortima `lightgbm` di backend untuk melakukan imputasi. `LightGBM` dikenal dengan akurasi prediksi yang tinggi. Menggabungkannya dengan algortima `mice` menjadikannya algortima yang kuat untuk imputasi.

□ Handling Outliers

Mengecek Outlier pada tiap columns di data train

```
print(f'Jumlah baris: {len(df_train)}')

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

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

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

    filtered_entries = ((df_train[col] >= low_limit) & (df_train[col]
<= high_limit)) & filtered_entries

print("Outlier All Data :", len(df_train[~filtered_entries]))
print("Not Outlier All Data :", len(df_train[filtered_entries]))
```

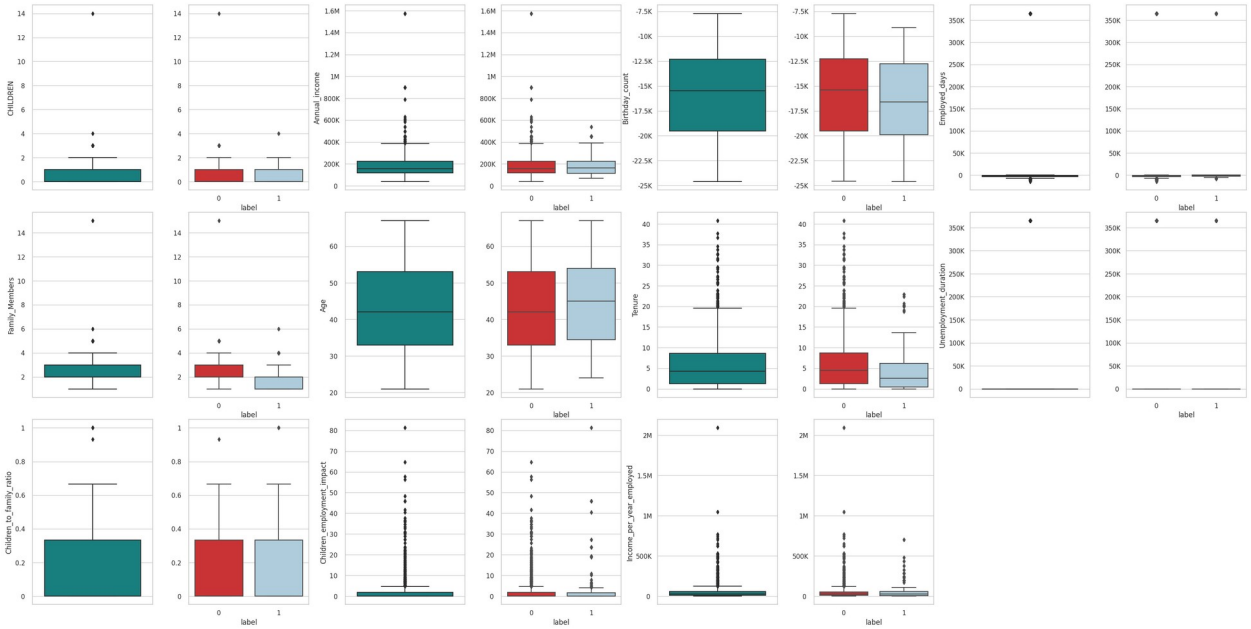
```
print()

pd.DataFrame({
    "Column Name":num_cols,
    "is Outlier": is_outlier,
    "Lower Limit": low_lim,
    "Upper Limit": high_lim,
    "Outlier":outlier,
    "No Outlier":no_outlier
})
```

```
Jumlah baris: 1161
Outlier All Data : 590
Not Outlier All Data : 571
```

	Column Name	is Outlier	Lower Limit	Upper Limit	Outlier	No Outlier
0	CHILDREN	True	-1.500	2.500	14	1147
1	Annual_income	True	-45000	387000	49	1112
2	Birthday_count	False	-30399.500	-1419.500	0	1161
3	Employed_days	True	-7193	3591	267	894
4	Family_Members	True	0.500	4.500	14	1147
5	Age	False	3	83	0	1161
6	Tenure	True	-9.838	19.707	66	1095
7	Unemployment_duration	True	0	0	201	960
8	Children_to_family_ratio	True	-0.500	0.833	3	
9	Children_employment_impact	True	-2.786	4.644	210	
10	Income_per_year_employed	True	-61479.580	123776.379	118	

```
8          1158
9           951
10         1043
```

```
from math import log10, floor
def format_func(value, tick_number=None):
    num_thousands = 0 if abs(value) < 1000 else floor
(log10(abs(value))/3)
    value = round(value / 1000**num_thousands, 2)
    return f'{value:g}'+' KMGTPEZY'[num_thousands]

plt.figure(figsize=(30, 15))
n = 3
j = 0

colors=["#e31a1c", "#a6cee3"]

for i in range(0, len(num_cols)*2, 2):
    ax1 = plt.subplot(n, math.ceil(len(num_cols)*2/n), i+1)
    sns.boxplot(y=df_train[num_cols[j]], color='#088F8F', orient='v')

    ax2 = plt.subplot(n, math.ceil(len(num_cols)*2/n), i+2,
sharey=ax1)
    sns.boxplot(x=df_train["label"], y=df_train[num_cols[j]],
orient='v', palette=colors)

    ax2.yaxis.set_major_formatter(plt.FuncFormatter(format_func))

    ax2.set_ylabel(None)

    plt.tight_layout(pad=0.1)

    j+=1
```



IQR (Interquartile Range)

IQR to be more robust since outliers are not included in the calculation of percentiles.

```
# def handling_outliers_iqr(data, cols):
#     q1 = data[cols].quantile(q=0.25)
#     q3 = data[cols].quantile(q=0.75)
#     iqr = q3-q1
#     fence_low  = q1 - 1.5*iqr
#     fence_high = q3 + 1.5*iqr

#     lower_outlier = data[data[cols] < fence_low]
#     upper_outlier = data[data[cols] > fence_high]

#     filter_outliers = (data[cols] >= fence_low) & (data[cols] <=
fence_high)
#     filtered = data[filter_outliers]
#     return filtered

# cols = ["Year_Birth","Income"]
# print("Jumlah data sebelum handling outliers :", df_train.shape[0])

# for i in cols:
#     df_train = handling_outliers_iqr(df_train, i)
#     print("Jumlah data setelah handling outliers ({}) :
{}".format(i, df_train.shape[0]))
```

Z-Score

Z-score is highly dependent on normality. If your data is non-normal (bimodal, extreme outlier, etc) then z-score will give poor results since the outliers are included in the mean and standard deviation calculations.

```
# from scipy import stats
# def handling_outliers_zscore(data, cols):
#     thres = 3
#     mean = np.mean(df_train[cols])
#     std = np.std(df_train[cols])
#     zscore = abs((df_train[cols]-mean)/std)
#     # zscore = abs(stats.zscore(data[cols]))

#     outlier = data[zscore >= thres]

#     filter_outliers = zscore < thres # negative value none, bcs
absolute
#     filtered = data[filter_outliers]
#     return filtered

# cols = ["Year_Birth","Income"]
# print("Jumlah data sebelum handling outliers :", df_train.shape[0])

# for i in cols:
#     df_train = handling_outliers_zscore(df_train, i)
#     print("Jumlah data setelah handling outliers ({}) :
{}".format(i, df_train.shape[0]))
```

Isolation Forest

Isolation Forest is an algorithm specifically designed to identify outliers in a dataset. Unlike traditional methods that rely on distance or density measures, Isolation Forest works by isolating observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

The idea is that outliers are fewer and have distinct characteristics that make them easier to isolate than normal points. Therefore, an outlier will require fewer random splits to isolate. The algorithm is particularly effective when the data is not normally distributed and when dealing with high-dimensional datasets.

Isolation Forest is robust against the curse of dimensionality and does not require the assumption of normality in the data, making it suitable for a wide range of applications where traditional methods like Z-Score may not perform well.

```
# from sklearn.ensemble import IsolationForest
# import pandas as pd

# def remove_outliers_iforest(data, cols, contamination=0.01,
random_state=42):
```

```
#     """
#     Removes outliers using Isolation Forest for specified columns in
the dataset.

#     Parameters:
#     - data: DataFrame, the dataset from which to remove outliers.
#     - cols: list, the columns to apply Isolation Forest on.
#     - contamination: float, the proportion of outliers in the data
set.
#     - random_state: int, random seed for reproducibility.

#     Returns:
#     - DataFrame, the dataset with outliers removed.
#     """
#     # Initialize Isolation Forest model
#     iforest = IsolationForest(contamination=contamination,
random_state=random_state)

#     # Fit the model to the specified columns
#     data['outlier'] = iforest.fit_predict(data[cols])

#     # Remove outliers (where the 'outlier' column equals -1)
#     data_cleaned = data[data['outlier'] != -1].copy()

#     # Drop the 'outlier' column as it's no longer needed
#     data_cleaned.drop(columns=['outlier'], inplace=True)

#     return data_cleaned

# # Daftar kolom yang akan diperiksa dengan IForest
# cols = [
#     'CHILDREN', 'Annual_income', 'Employed_days', 'Family_Members',
#     'Tenure', 'Unemployment_duration', 'Children_to_family_ratio',
#     'Children_employment_impact', 'Income_per_year_employed'
# ]

# # Tampilkan jumlah data sebelum penanganan outliers
# print("Jumlah data sebelum handling outliers:", df_train.shape[0])

# # Terapkan fungsi untuk menghapus outliers menggunakan Isolation
Forest
# df_train = remove_outliers_iforest(df_train, cols,
contamination=0.01)

# # Tampilkan jumlah data setelah penanganan outliers
# print("Jumlah data setelah handling outliers:", df_train.shape[0]
```

Elliptic Envelope

The **Elliptic Envelope** method is a statistical approach used for outlier detection based on the assumption that the data follows a Gaussian (normal) distribution. It works by fitting a multivariate Gaussian distribution to the data, creating an "elliptical envelope" around the majority of the data points. Observations that fall outside this envelope are considered outliers.

This method is effective when the data is approximately normally distributed. The Elliptic Envelope method estimates the mean and covariance of the data and uses these estimates to define the decision boundary (the elliptical envelope) that separates inliers from outliers.

However, the Elliptic Envelope may not perform well on data that does not follow a normal distribution, and its performance can degrade with the presence of noise or when the data contains significant outliers that can distort the mean and covariance estimates.

```
# from sklearn.covariance import EllipticEnvelope
# import pandas as pd

# def remove_outliers_elliptic(data, cols, contamination=0.01,
random_state=42):
#     """
#     Removes outliers using Elliptic Envelope for specified columns
in the dataset.

#     Parameters:
#     - data: DataFrame, the dataset from which to remove outliers.
#     - cols: list, the columns to apply Elliptic Envelope on.
#     - contamination: float, the proportion of outliers in the data
set.
#     - random_state: int, random seed for reproducibility.

#     Returns:
#     - DataFrame, the dataset with outliers removed.
#     """
#     # Initialize Elliptic Envelope model
#     elliptic_env = EllipticEnvelope(contamination=contamination,
random_state=random_state)

#     # Fit the model to the specified columns
#     elliptic_env.fit(data[cols])

#     # Predict outliers (-1 indicates an outlier)
#     data['outlier'] = elliptic_env.predict(data[cols])

#     # Remove outliers (where the 'outlier' column equals -1)
#     data_cleaned = data[data['outlier'] != -1].copy()

#     # Drop the 'outlier' column as it's no longer needed
#     data_cleaned.drop(columns=['outlier'], inplace=True)
```

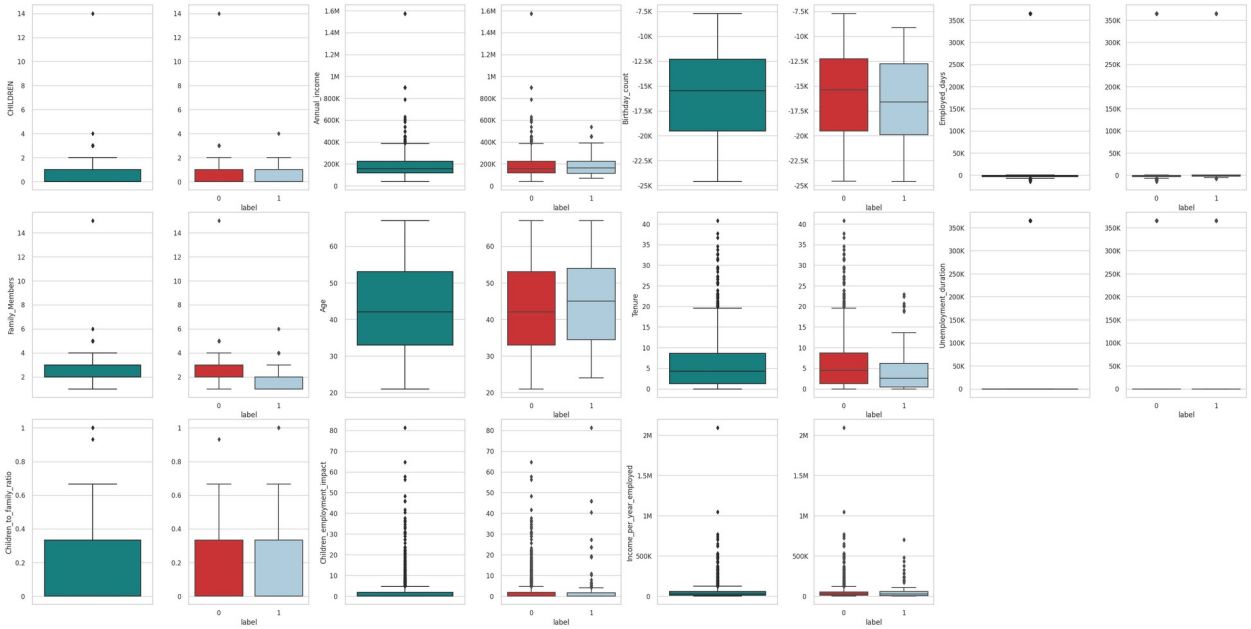
```
#     return data_cleaned

# # Daftar kolom yang akan diperiksa dengan Elliptic Envelope
# cols = [
#     'CHILDREN', 'Annual_income', 'Employed_days', 'Family_Members',
#     'Tenure', 'Unemployment_duration', 'Children_to_family_ratio',
#     'Children_employment_impact', 'Income_per_year_employed'
# ]


# # Tampilkan jumlah data sebelum penanganan outliers
# print("Jumlah data sebelum handling outliers:", df_train.shape[0])

# # Terapkan fungsi untuk menghapus outliers menggunakan Elliptic
Envelope
# df_train_cleaned = remove_outliers_elliptic(df_train, cols,
contamination=0.01)

# # Tampilkan jumlah data setelah penanganan outliers
# print("Jumlah data setelah handling outliers:",
df_train_cleaned.shape[0])

# # Lihat data yang telah dibersihkan
# print(df_train_cleaned)

from sklearn.neighbors import LocalOutlierFactor
import pandas as pd

def remove_outliers_lof_train_test(train_data, test_data, cols,
contamination=0.01, n_neighbors=20):
    """
    Removes outliers using Local Outlier Factor (LOF) for specified
columns in the dataset.

    Parameters:
    - train_data: DataFrame, the training dataset.
    - test_data: DataFrame, the testing dataset.
    - cols: list, the columns to apply LOF on.
    - contamination: float, the proportion of outliers in the data
set.
    - n_neighbors: int, the number of neighbors to use for calculating
the local density.

    Returns:
    - train_data_cleaned: DataFrame, the training dataset with
outliers removed.
    - test_data_cleaned: DataFrame, the testing dataset with outliers
removed.
    """
    # Initialize LOF model with novelty=False for training data
```

```
    lof_train = LocalOutlierFactor(n_neighbors=n_neighbors,
contamination=contamination, novelty=False)

    # Fit the model to the training data and predict outliers (-1
indicates an outlier)
    train_data['outlier'] = lof_train.fit_predict(train_data[cols])

    # Remove outliers from training data (where the 'outlier' column
equals -1)
    train_data_cleaned = train_data[train_data['outlier'] != -
1].copy()

    # Initialize LOF model with novelty=True for testing data
    lof_test = LocalOutlierFactor(n_neighbors=n_neighbors,
contamination=contamination, novelty=True)

    # Fit the model on training data only (as novelty=True, we do not
use fit_predict)
    lof_test.fit(train_data_cleaned[cols])

    # Predict outliers on the testing data
    test_data['outlier'] = lof_test.predict(test_data[cols])

    # Remove outliers from testing data (where the 'outlier' column
equals -1)
    test_data_cleaned = test_data[test_data['outlier'] != -1].copy()

    # Drop the 'outlier' column as it's no longer needed
    train_data_cleaned.drop(columns=['outlier'], inplace=True)
    test_data_cleaned.drop(columns=['outlier'], inplace=True)

    return train_data_cleaned, test_data_cleaned

# Daftar kolom yang akan diperiksa dengan LOF
cols = [
    'CHILDREN', 'Annual_income', 'Employed_days', 'Family_Members',
    'Tenure', 'Unemployment_duration', 'Children_to_family_ratio',
    'Children_employment_impact', 'Income_per_year_employed'
]

# Tampilkan jumlah data sebelum penanganan outliers
print("Jumlah data sebelum handling outliers (Train):",
df_train.shape[0])
print("Jumlah data sebelum handling outliers (Test):",
df_test.shape[0])

# Terapkan fungsi untuk menghapus outliers menggunakan Local Outlier
Factor
df_train_cleaned, df_test_cleaned =
remove_outliers_lof_train_test(df_train, df_test, cols,
```

```
contamination=0.01, n_neighbors=20)

# Tampilkan jumlah data setelah penanganan outliers
print("Jumlah data setelah handling outliers (Train):",
df_train_cleaned.shape[0])
print("Jumlah data setelah handling outliers (Test):",
df_test_cleaned.shape[0])
```

```
Jumlah data sebelum handling outliers (Train): 1161
Jumlah data sebelum handling outliers (Test): 387
Jumlah data setelah handling outliers (Train): 1149
Jumlah data setelah handling outliers (Test): 378
```

```
from math import log10, floor
def format_func(value, tick_number=None):
    num_thousands = 0 if abs(value) < 1000 else floor
(log10(abs(value))/3)
    value = round(value / 1000**num_thousands, 2)
    return f'{value:g}'+' KMGTPEZY'[num_thousands]

plt.figure(figsize=(30, 15))
n = 3
j = 0

colors=["#e31a1c", "#a6cee3"]

for i in range(0, len(num_cols)*2, 2):
    ax1 = plt.subplot(n, math.ceil(len(num_cols)*2/n), i+1)
    sns.boxplot(y=df_train[num_cols[j]], color='#088F8F', orient='v')

    ax2 = plt.subplot(n, math.ceil(len(num_cols)*2/n), i+2,
sharey=ax1)
    sns.boxplot(x=df_train["label"], y=df_train[num_cols[j]],
orient='v', palette=colors)

    ax2.yaxis.set_major_formatter(plt.FuncFormatter(format_func))

    ax2.set_ylabel(None)

    plt.tight_layout(pad=0.1)

    j+=1
```



Feature Transformation (Numeric)

Mengecek Skewness di tiap kolom untuk menentukan jenis Transformation

```
skew_type_list = []
skew_val_list = []
kurtosis_val_list = []

for column in num_cols:
    data = df_train[column].dropna(axis=0)
    q1 = data.quantile(q=0.25)
    mean = round(data.mean(), 3)
    median = data.median()
    mode = data.mode()[0]
    q3 = data.quantile(q=0.75)
    # skew_val = round(data.skew(),3)
    skew_val = round(skew(data, nan_policy="omit"),3)
    kurtosis_val = round(kurtosis(data, nan_policy="omit"),3)

    if (mean == median == mode) or (-0.2 < skew_val < 0.2):
        skew_type = "Normal Distribution (Symmetric)"
    elif mean < median < mode:
        skew_type = "Negatively Skewed"
        if skew_val <= -1:
            skew_type = "Highly Negatively Skewed"
        elif -0.5 >= skew_val > -1:
            skew_type = "Moderately Negatively Skewed"
        else :
            skew_type = "Moderately Normal Distribution (Symmetric)"
    else:
```

```
        skew_type = "Positively Skewed"
        if skew_val >= 1:
            skew_type = "Highly Positively Skewed"
        elif 0.5 <= skew_val < 1:
            skew_type = "Moderately Positively Skewed"
        else :
            skew_type = "Moderately Normal Distribution (Symmetric)"
    skew_type_list.append(skew_type)
    skew_val_list.append(skew_val)
    kurtosis_val_list.append(kurtosis_val)

dist = pd.DataFrame({
    "Column Name":num_cols,
    "Skewness": skew_val_list,
    "Kurtosis": kurtosis_val_list,
    "Type of Distribution": skew_type_list
})

exclude = ["CHILDREN", "Family_Members"]

log_cols = sorted(list(dist[
    dist["Type of Distribution"].str.contains("Positively Skewed") &
    ~dist["Column Name"].isin(exclude)
]["Column Name"].values))

norm_cols = sorted(list(dist[
    dist["Type of Distribution"].str.contains("Normal Distribution") &

    ~dist["Column Name"].isin(exclude)
]["Column Name"].values))

print("Log Transformation =", log_cols)
print("Normalisasi/Standardization =", norm_cols)

Log Transformation = ['Annual_income', 'Children_employment_impact',
'Children_to_family_ratio', 'Employed_days',
'Income_per_year_employed', 'Tenure', 'Unemployment_duration']
Normalisasi/Standardization = ['Age', 'Birthday_count']
```

Berikut adalah observasi yang baik berdasarkan data dan transformasi yang akan dilakukan:

Observasi Transformasi Data

Dari hasil temuan dan analisis awal, kita dapat menentukan beberapa transformasi yang perlu dilakukan untuk meningkatkan performa model dan memastikan distribusi data lebih sesuai dengan asumsi normalitas:

- **Scaling and Converting to a Normal Distribution:**
  - **Log Transformation**:

- Transformasi logaritma digunakan untuk mengurangi skewness pada data yang memiliki distribusi positif skewed dan untuk mengubah data menjadi lebih mendekati distribusi normal. Transformasi ini sangat berguna untuk variabel yang memiliki skala nilai yang besar atau menyimpang dari distribusi normal.

Adapun daftar kolom yang akan kita lakukan log transformation adalah:

- `Annual_income`
- `Children_employment_impact`
- `Children_to_family_ratio`
- `Employed_days`
- `Income_per_year_employed`
- `Tenure`
- `Unemployment_duration`

Transformasi log pada kolom-kolom ini diharapkan dapat meratakan distribusi, mengurangi pengaruh outlier, dan membuat data lebih sesuai dengan asumsi distribusi normal yang dibutuhkan oleh beberapa model statistik.

- **Just Scaling:**
  - **Normalization/Standardization**:
    - Normalisasi dan standardisasi digunakan untuk meratakan skala data sehingga semua variabel memiliki rentang nilai yang serupa. Ini penting untuk algoritma yang sensitif terhadap skala data, seperti regresi linier, K-means clustering, atau analisis komponen utama (PCA).

    Adapun daftar kolom yang akan kita lakukan normalisasi atau standardisasi adalah:

    - `Age`
    - `Birthday_count`

    Transformasi ini akan membantu dalam memastikan bahwa variabel-variabel tersebut tidak mendominasi model karena skala yang berbeda, serta memudahkan proses pembelajaran pada algoritma yang memerlukan data berskala seragam.
- **Kolom yang Tidak Memerlukan Transformasi:**
  - Beberapa kolom tidak perlu melakukan transformasi karena rentang nilai yang masih wajar dan distribusinya sudah cukup normal atau tidak mempengaruhi model secara signifikan.

Choice Determination:

- Pada proses `Feature Transformation / Scaling` ini kita menggunakan `Yeo-Johnson Transformation` pada kolom-kolom yang masih memiliki skala yang besar, karena dari hasilnya kita bisa melihat hasil bentuk curve yang lebih Normal Distribusi. Dan sangat cocok untuk penggunaan Algoritma berbasis tree.

Karena akhirnya dari proses evaluasi model, didapatkan penggunaan transformasi yang hanya menggunakan `Yeo-Johnson Transformation` pada kolom-kolom yang masih memiliki skala yang besar, memberikan hasil yang lumayan akurat maka untuk kolom skew maupun normal akan dilakukan transformasi yang sama.

```
log_cols = log_cols + norm_cols
```

Sehingga untuk `Normalization` dan `Standardization` tidak akan digunakan

Normalization

```
# from sklearn.preprocessing import MinMaxScaler
# # create a scaler object
# scaler = MinMaxScaler()
# # fit and transform the data
# df_train[norm_cols] =
pd.DataFrame(scaler.fit_transform(df_train[norm_cols]),
columns=df_train[norm_cols].columns)
# df_train[norm_cols].describe()

# df_test[norm_cols] =
pd.DataFrame(scaler.transform(df_test[norm_cols]),
columns=df_test[norm_cols].columns)
# df_test[norm_cols].describe()
```

Standardization

```
# from sklearn.preprocessing import StandardScaler

# # create a scaler object
# std_scaler = StandardScaler()
# std_scaler
# # fit and transform the data
# df_train[norm_cols] =
pd.DataFrame(std_scaler.fit_transform(df_train[norm_cols]),
columns=df_train[norm_cols].columns)
# df_train[norm_cols].describe()

# df_test[norm_cols] =
pd.DataFrame(std_scaler.transform(df_test[norm_cols]),
columns=df_test[norm_cols].columns)
# df_test[norm_cols].describe()
```

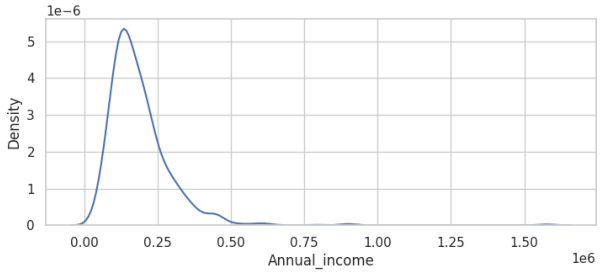
Log Transformation

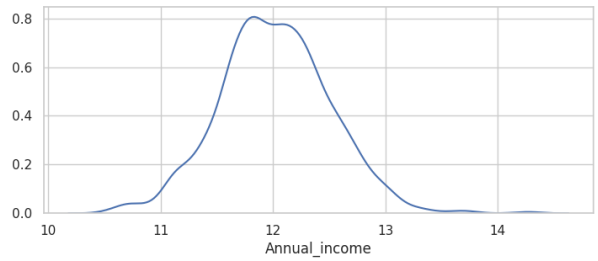
Numerical variables may have high skewed and non-normal distribution (Gaussian Distribution) caused by outliers, highly exponential distributions, etc. Therefore we go for data transformation.

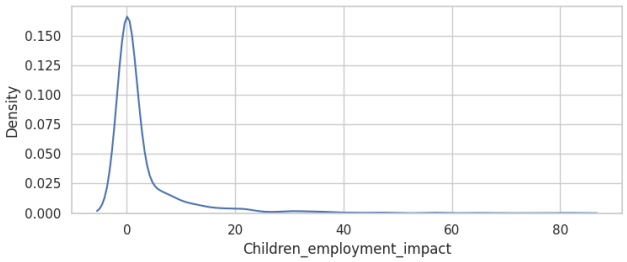
In Log transformation each variable of x will be replaced by log(x) with base 10, base 2, or natural log.

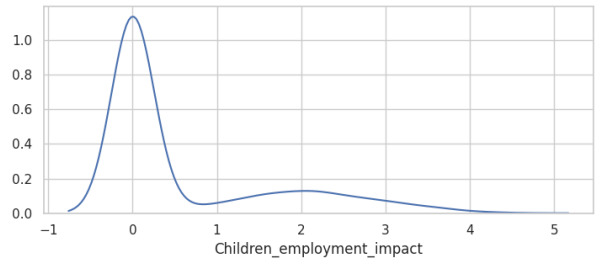
Observation: Error diakibatkan karena nilai dataset bernilai negatif dan box cox tidak bisa menghandle data negatif. sehingga kita menggunakan yeo-johnson

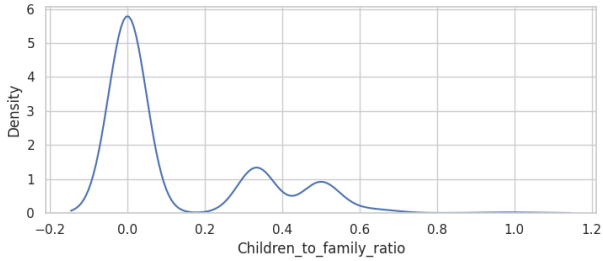
```
fig, ax = plt.subplots(len(log_cols),2,figsize=(15,30))
for i in range(0,len(log_cols)):
    kde1 = sns.kdeplot(df_train[log_cols[i]], ax=ax[i][0])
    kde2 = sns.kdeplot(np.log(df_train[log_cols[i]]+1), ax=ax[i][1])
    kde2.set_ylabel(None)
    plt.tight_layout()
```

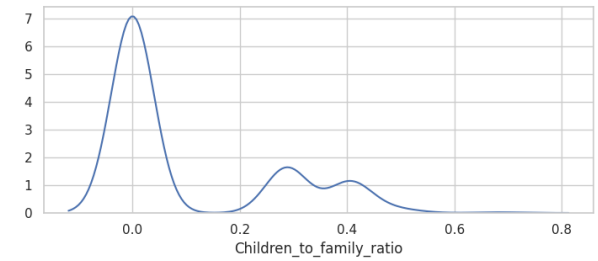

1e−6
Density
Annual_income
1e6

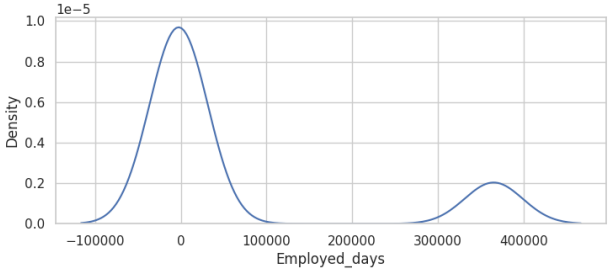

Annual_income

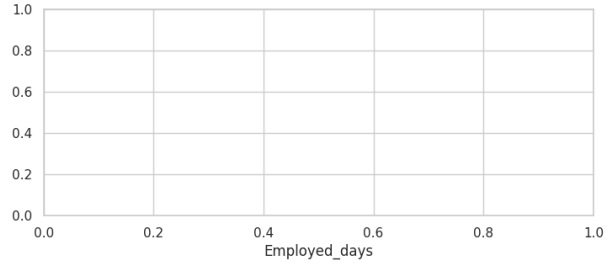

Density
Children_employment_impact

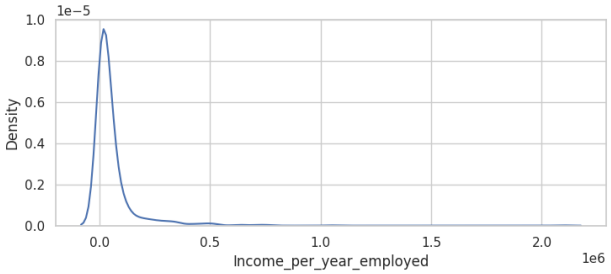

Children_employment_impact

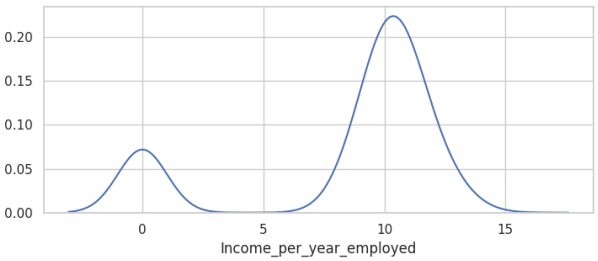

Density
Children_to_family_ratio

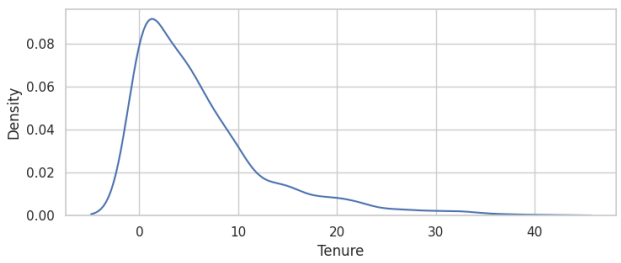

Children_to_family_ratio

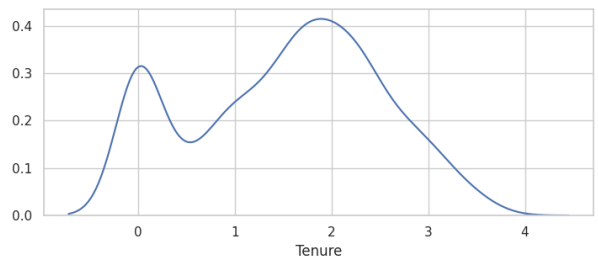

1e−5
Density
Employed_days


Employed_days


1e−5
Density
Income_per_year_employed
1e6


Income_per_year_employed


Density
Tenure


Tenure


1e−5

```
# # Train
# # add 1 bcs err : divide by zero encountered in log
# # df[log_cols] = np.log(df[log_cols]+1)
# df_train[log_cols] = np.log1p(df_train[log_cols])
# df_train[log_cols].describe()

# # Test
# df_test[log_cols] = np.log1p(df_test[log_cols])
# df_test[log_cols].describe()
```

Box-Cox Transformation

Box-cox transformation works pretty well for many data natures. The below image is the mathematical formula for Box-cox transformation.

$$y(\lambda) = \begin{cases} \frac{y^{\lambda}-1}{\lambda}, & \text{if } \lambda \neq 0; \\ \log y, & \text{if } \lambda = 0. \end{cases}$$

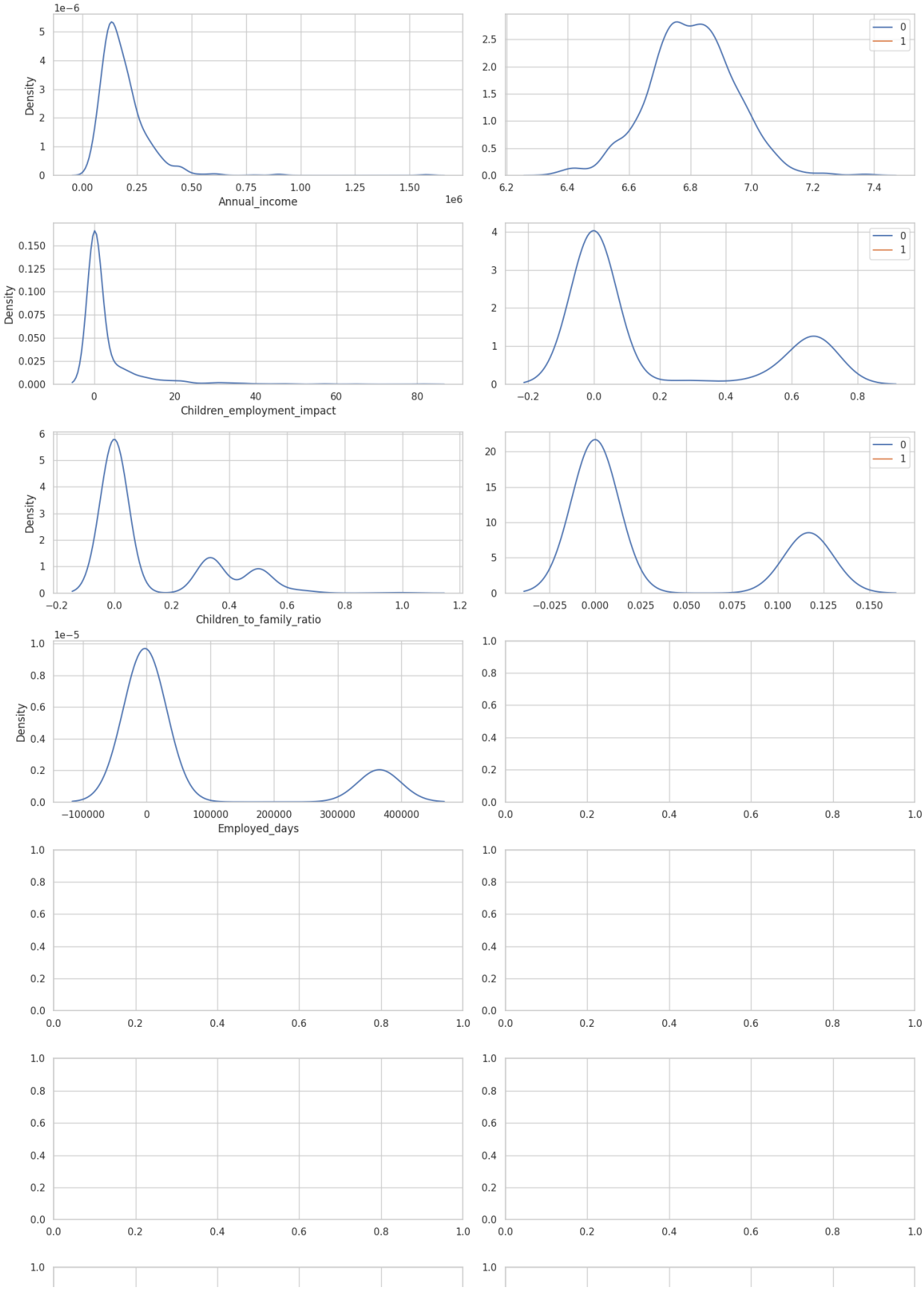
All the values of lambda vary from -5 to 5 are considered and the best value for the data is selected. The "Best" value is one that results in the best skewness of the distribution. Log transformation will take place when we have lambda is zero.

- **with Scipy**

```
from scipy.stats import boxcox
fig, ax = plt.subplots(len(log_cols),2,figsize=(15,30))
for i in range(0,len(log_cols)):
    kde1 = sns.kdeplot(df_train[log_cols[i]], ax=ax[i][0])
    kde2 = sns.kdeplot(boxcox(df_train[log_cols[i]]+1),
warn_singular=False, ax=ax[i][1])
    kde2.set_ylabel(None)
    plt.tight_layout()
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[166], line 5
      3 for i in range(0,len(log_cols)):
      4     kde1 = sns.kdeplot(df_train[log_cols[i]], ax=ax[i][0])
----> 5     kde2 = sns.kdeplot(boxcox(df_train[log_cols[i]]+1),
warn_singular=False, ax=ax[i][1])
      6     kde2.set_ylabel(None)
      7     plt.tight_layout()

File
/opt/conda/lib/python3.10/site-packages/scipy/stats/_morestats.py:1130
```

```
, in boxcox(x, lmbda, alpha, optimizer)
   1127     raise ValueError("Data must not be constant.")
   1129 if np.any(x <= 0):
-> 1130     raise ValueError("Data must be positive.")
   1132 # If lmbda=None, find the lmbda that maximizes the log-
likelihood function.
   1133 lmax = boxcox_normmax(x, method='mle', optimizer=optimizer)

ValueError: Data must be positive.
```


1e−6
Density
Annual_income
1e6
0
1
Children_employment_impact
Children_to_family_ratio
1e−5
Employed_days

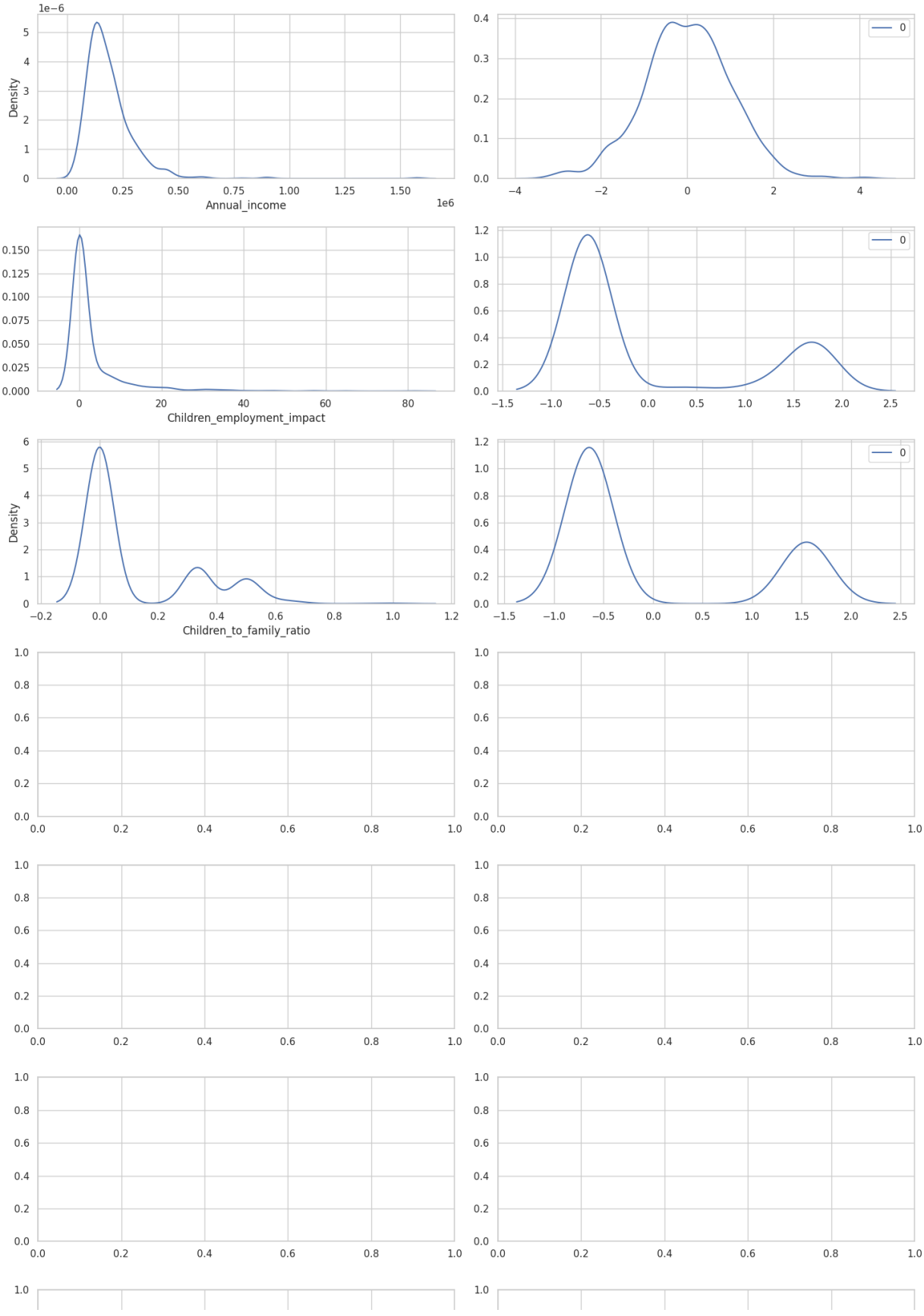
```
# # Train
# for i in log_cols:
#     df_train[i], parameters = boxcox(df_train[i]+1)

# df_train[log_cols].describe()

# # Test
# for i in log_cols:
#     df_test[i], parameters = boxcox(df_test[i]+1)

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

- **with Sklearn**

```
from sklearn.preprocessing import PowerTransformer

fig, ax = plt.subplots(len(log_cols),2,figsize=(15,30))
for i in range(0,len(log_cols)):
    pt = PowerTransformer(method='box-cox')
    data = pt.fit_transform(df_train[[log_cols[i]]]+1)
    kde1 = sns.kdeplot(df_train[log_cols[i]], ax=ax[i][0])
    kde2 = sns.kdeplot(data, ax=ax[i][1])
    kde2.set_ylabel(None)
    plt.tight_layout()
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[169], line 6
      4 for i in range(0,len(log_cols)):
      5     pt = PowerTransformer(method='box-cox')
----> 6     data = pt.fit_transform(df_train[[log_cols[i]]]+1)
      7     kde1 = sns.kdeplot(df_train[log_cols[i]], ax=ax[i][0])
      8     kde2 = sns.kdeplot(data, ax=ax[i][1])

File
/opt/conda/lib/python3.10/site-packages/sklearn/utils/_set_output.py:1
40, in _wrap_method_output.<locals>.wrapped(self, X, *args, **kwargs)
    138 @wraps(f)
    139 def wrapped(self, X, *args, **kwargs):
--> 140     data_to_wrap = f(self, X, *args, **kwargs)
    141     if isinstance(data_to_wrap, tuple):
    142         # only wrap the first output for cross decomposition
    143         return (
    144             _wrap_data_with_container(method, data_to_wrap[0],
X, self),
    145             *data_to_wrap[1:],
    146         )
```

```
File
/opt/conda/lib/python3.10/site-packages/sklearn/preprocessing/_data.py
:3103, in PowerTransformer.fit_transform(self, X, y)
   3086 """Fit `PowerTransformer` to `X`, then transform `X`.
   3087
   3088 Parameters
   (...)
   3100     Transformed data.
   3101 """
   3102 self._validate_params()
-> 3103 return self._fit(X, y, force_transform=True)

File
/opt/conda/lib/python3.10/site-packages/sklearn/preprocessing/_data.py
:3106, in PowerTransformer._fit(self, X, y, force_transform)
   3105 def _fit(self, X, y=None, force_transform=False):
-> 3106     X = self._check_input(X, in_fit=True, check_positive=True)
   3108     if not self.copy and not force_transform:  # if call from
fit()
   3109         X = X.copy()  # force copy so that fit does not change
X inplace

File
/opt/conda/lib/python3.10/site-packages/sklearn/preprocessing/_data.py
:3339, in PowerTransformer._check_input(self, X, in_fit,
check_positive, check_shape)
   3337     warnings.filterwarnings("ignore", r"All-NaN (slice|axis)
encountered")
   3338     if check_positive and self.method == "box-cox" and
np.nanmin(X) <= 0:
-> 3339         raise ValueError(
   3340             "The Box-Cox transformation can only be "
   3341             "applied to strictly positive data"
   3342         )
   3344 if check_shape and not X.shape[1] == len(self.lambdas_):
   3345     raise ValueError(
   3346         "Input data has a different number of features "
   3347         "than fitting data. Should have {n}, data has
{m}".format(
   3348             n=len(self.lambdas_), m=X.shape[1]
   3349         )
   3350     )

ValueError: The Box-Cox transformation can only be applied to strictly
positive data
```


1e−6
Density
Annual_income
1e6
0
Children_employment_impact
Children_to_family_ratio

```
# # Train
# pt = PowerTransformer(method='box-cox')
# df_train[log_cols] = pt.fit_transform(df_train[log_cols]+1)
# df_train[log_cols].describe()

# # Test
# df_test[log_cols] = pt.transform(df_test[log_cols]+1)
# df_test[log_cols].describe()
```

Yeo-Johnson Transformation

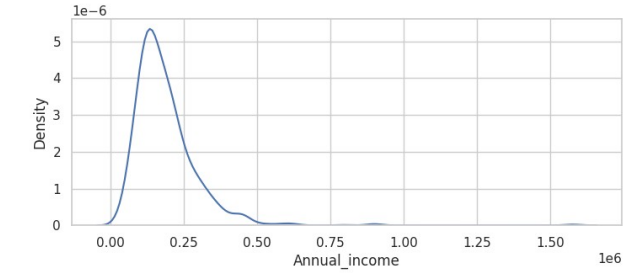
- The advantage of the Yeo-Johnson transformation over the box-cox transformation is that, by default, the input (or parameter) of the Yeo-Johnson transformation can be a negative value. Unlike the Box-Cox transform, it does not require the values for each input variable to be strictly positive.
- This is a huge advantage over the box-cox transformation since we don't have to generate strictly positive values before applying the transformation.
- It supports zero values and negative values. This means we can apply it to our dataset without scaling it first.

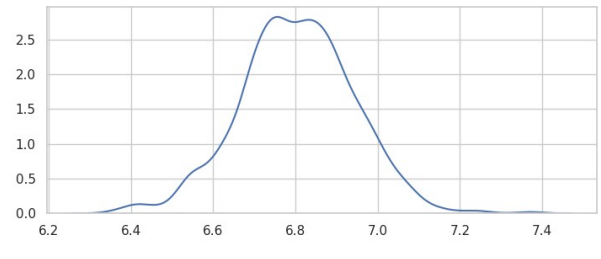
$$\psi(\lambda, y) = \begin{cases} ((y+1)^{\lambda} - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y+1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y+1)^{2-\lambda} - 1)]/(2-\lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y+1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

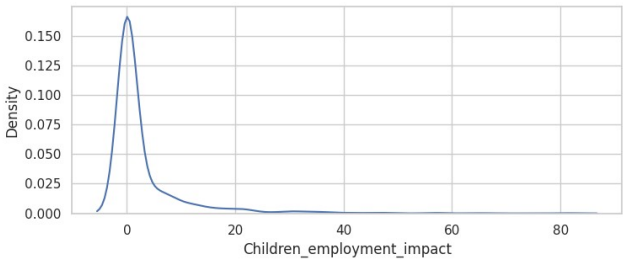
References :

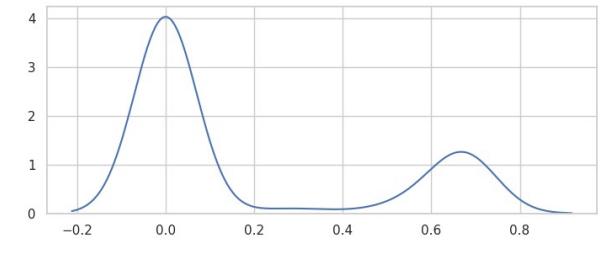
- https://statisticaloddsandends.wordpress.com/2021/02/19/the-box-cox-and-yeo-johnson-transformations-for-continuous-variables/
- https://www.stat.umn.edu/arc/yjpower.pdf
- **with scipy**

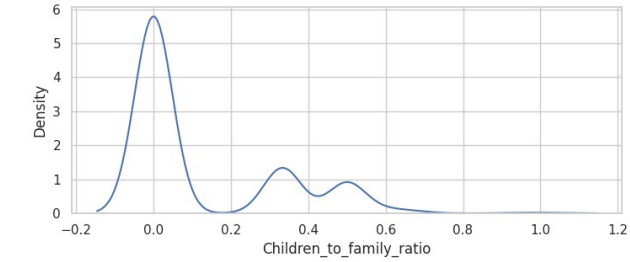
```
from scipy.stats import yeojohnson

fig, ax = plt.subplots(len(log_cols),2,figsize=(15,30))
for i in range(0,len(log_cols)):
    data, fitted_lambda = yeojohnson(df_train[log_cols[i]],lmbda=None)
    kde1 = sns.kdeplot(df_train[log_cols[i]], ax=ax[i][0])
    kde2 = sns.kdeplot(data, ax=ax[i][1])
    kde2.set_ylabel(None)
    plt.tight_layout()
```

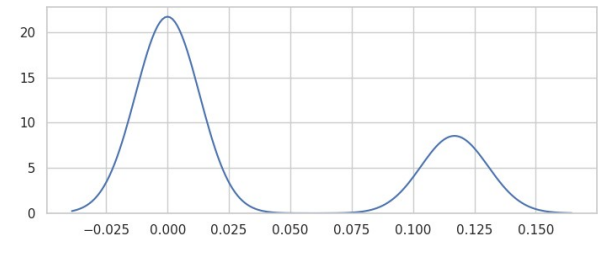

1e−6
Density
Annual_income
1e6

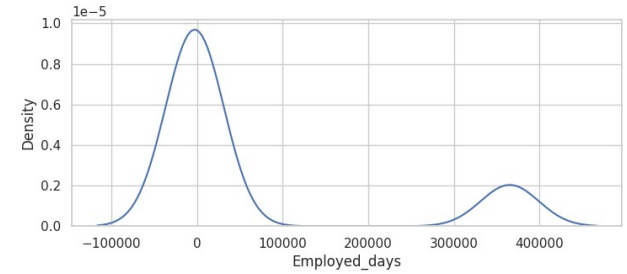


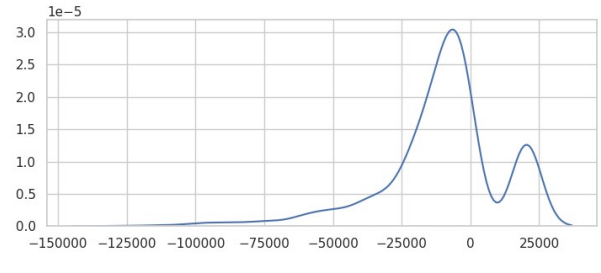

Density
Children_employment_impact

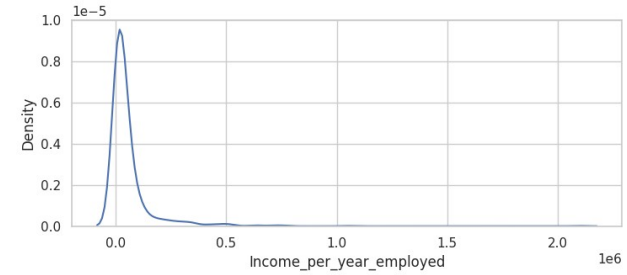


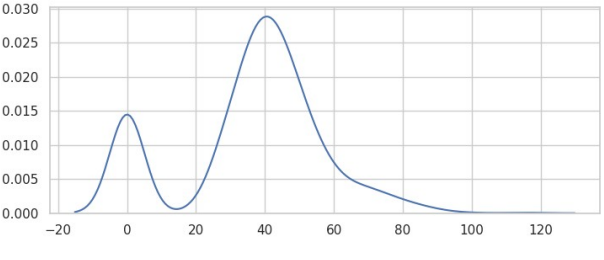

Density
Children_to_family_ratio

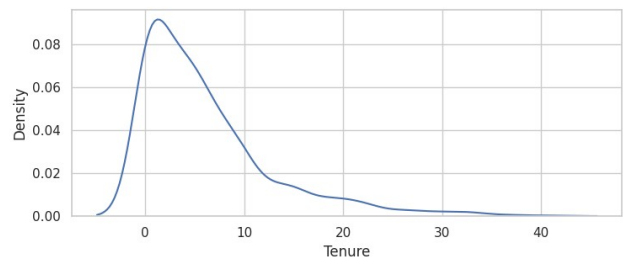


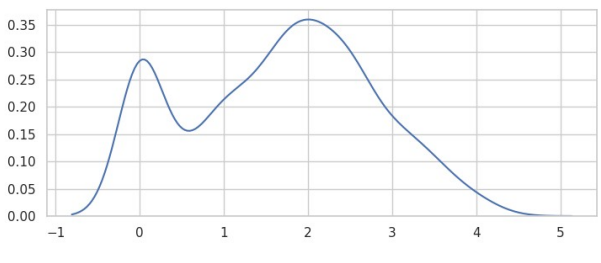

1e−5
Density
Employed_days


1e−5

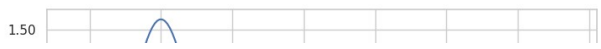

1e−5
Density
Income_per_year_employed
1e6




Density
Tenure



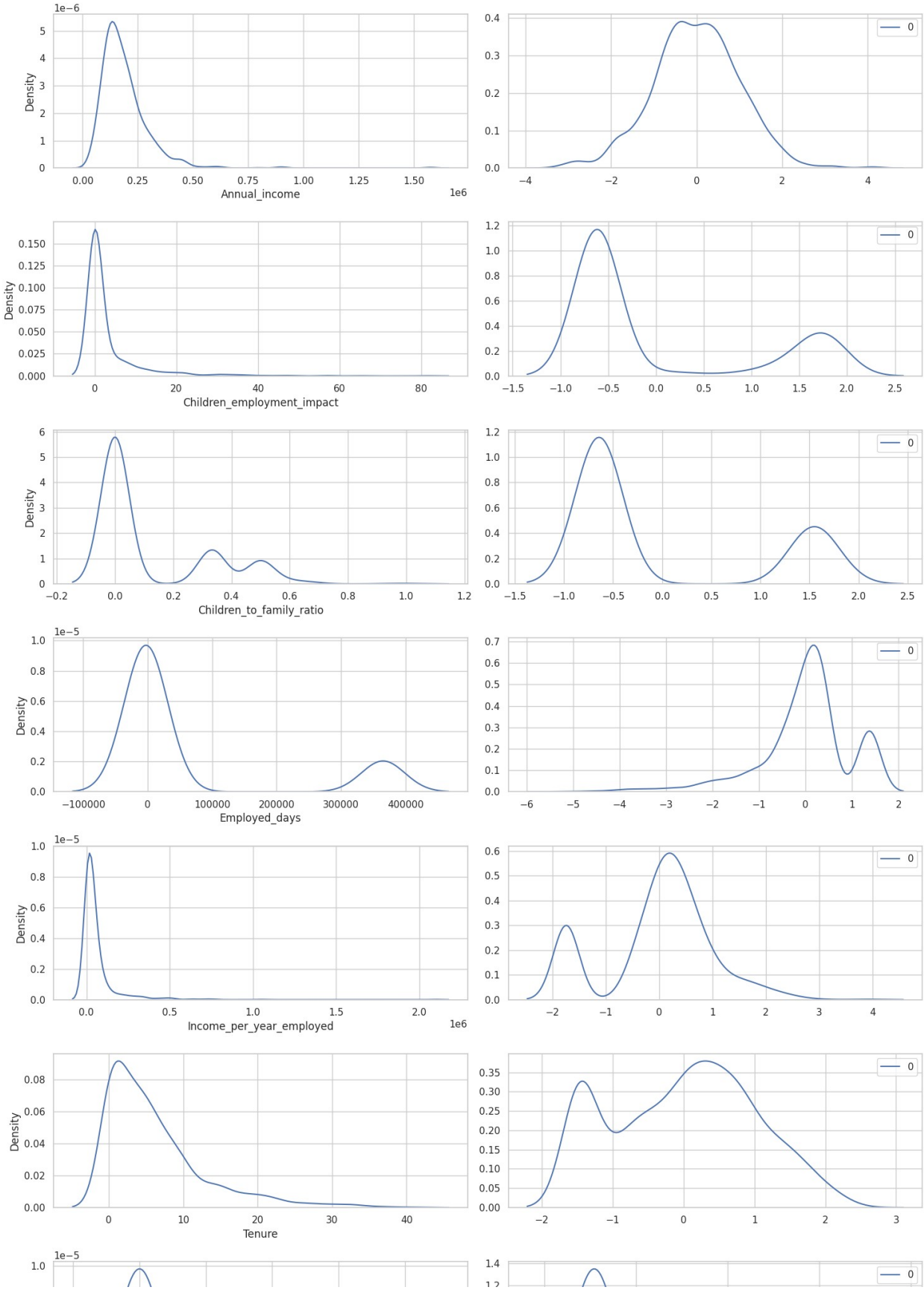

1e−5



```
# # Train
# for i in log_cols:
#     df_train[i], fitted_lambda = yeojohnson(df_train[i], lmbda=None)

# df_train[log_cols].describe()

# # Test
# for i in log_cols:
#     df_test[i], fitted_lambda = yeojohnson(df_test[i], lmbda=None)

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

- **with sklearn**

```
from sklearn.preprocessing import PowerTransformer

fig, ax = plt.subplots(len(log_cols),2,figsize=(15,30))
for i in range(0,len(log_cols)):
    pt = PowerTransformer(method='yeo-johnson')
    data = pt.fit_transform(df_train[[log_cols[i]]]+1)
    kde1 = sns.kdeplot(df_train[log_cols[i]], ax=ax[i][0])
    kde2 = sns.kdeplot(data, ax=ax[i][1])
    kde2.set_ylabel(None)
    plt.tight_layout()
```


1e−6
Density
Annual_income
1e6
0
Children_employment_impact
Children_to_family_ratio
1e−5
Employed_days
Income_per_year_employed
Tenure

```
# Train
pt = PowerTransformer(method='yeo-johnson')
df_train[log_cols] = pt.fit_transform(df_train[log_cols])
df_train[log_cols].describe()
```

```
       Annual_income  Children_employment_impact
Children_to_family_ratio  \
count           1161                        1161
1161
mean           0.000                      -0.000
0
std            1.000                       1.000
1.000
min           -3.225                      -0.621                      -
0.643
25%           -0.695                      -0.621                      -
0.643
50%           -0.078                      -0.621                      -
0.643
75%            0.637                       1.270
1.484
max            4.098                       1.819
1.717

       Employed_days  Income_per_year_employed  Tenure
Unemployment_duration  \
count           1161                      1161    1161
1161
mean           0.000                    -0.000  -0.000
0.000
std            1.000                     1.000   1.000
1.000
min           -5.257                    -1.754  -1.511
-0.458
25%           -0.360                    -0.337  -0.754
-0.458
50%            0.121                     0.152   0.103
-0.458
75%            0.391                     0.549   0.738
-0.458
max            1.374                     3.784   2.422
2.185

         Age  Birthday_count
count   1161            1161
mean  -0.000          -0.000
std    1.000           1.000
min   -2.236          -1.826
25%   -0.857          -0.860
50%   -0.020           0.036
```

```
75%    0.862           0.845
max    1.830           2.291

# Test
df_test[log_cols] = pt.transform(df_test[log_cols])
df_test[log_cols].describe()

       Annual_income  Children_employment_impact
Children_to_family_ratio  \
count            387                         387
387
mean           0.076                       0.019
0.018
std            1.000                       1.001
1.006
min           -3.496                      -0.621                    -
0.643
25%           -0.616                      -0.621                    -
0.643
50%            0.193                      -0.621                    -
0.643
75%            0.637                       1.249
1.484
max            3.175                       1.810
1.698

       Employed_days  Income_per_year_employed  Tenure
Unemployment_duration  \
count            387                       387     387
387
mean           0.002                     0.100  -0.013
-0.048
std            0.926                     1.070   1.001
0.958
min           -4.372                    -1.754  -1.511
-0.458
25%           -0.430                    -0.297  -0.893
-0.458
50%            0.119                     0.176   0.106
-0.458
75%            0.413                     0.709   0.802
-0.458
max            1.374                     2.969   2.263
2.185

         Age  Birthday_count
count    387             387
mean   0.023          -0.013
std    0.975           0.977
min   -2.105          -1.885
```

```
25%   -0.757          -0.802
50%    0.066          -0.036
75%    0.787           0.795
max    1.894           2.116

df_train.head()

    Ind_ID  GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
\
0  5024636       1          0              1         0         -2.815

1  5023692       1          0              1         0         -0.396

2  5148525       1          0              1         0          0.193

3  5028446       1          1              1         0         -0.021

4  5069289       1          0              1         0          0.992


   Type_Income  EDUCATION  Marital_status  Housing_type
Birthday_count  \
0            2          3               3             6          -
0.940
1            2          3               1             6
0.138
2            1          3               3             6          -
1.602
3            2          3               1             6          -
0.554
4            4          2               3             6
0.199

   Employed_days  Work_Phone  Phone  EMAIL_ID  Type_Occupation  \
0          0.236           0      0         0                2
1         -0.422           0      0         0               11
2          1.374           0      0         0               13
3         -0.254           0      0         0                2
4         -0.441           0      0         0               13

   Family_Members  label    Age  Tenure  Unemployment_duration  \
0               2      0  0.936  -0.159                -0.458
1               1      0 -0.107   0.795                -0.458
2               2      0  1.634  -1.511                 2.185
3               1      0  0.556   0.633                -0.458
4               2      0 -0.195   0.812                -0.458

   Is_currently_employed  Children_to_family_ratio  \
0                      1                    -0.643
1                      1                    -0.643
2                      0                    -0.643
```

```
3                        1                      -0.643
4                        1                      -0.643

   Children_employment_impact  Income_per_year_employed Income_sgmt  \
0                      -0.621                    -1.754         Low
1                      -0.621                    -0.094      Medium
2                      -0.621                    -1.754      Medium
3                      -0.621                     0.061      Medium
4                      -0.621                     0.216        High

   Age_group  outlier
0          1        1
1          0        1
2          1        1
3          1        1
4          0        1
```

Kesimpulan

Berdasarkan hasil pengecekan pada beberapa fitur yang telah diproses menggunakan transformation sebelumnya, dapat diketahui bahwa keseluruhan nilai skewnessnya sudah memiliki rentang yang lebih seragam (tidak jauh dan tidak terlalu bervariasi). Sehingga dapat disimpulkan bahwa teknik fitur transformation yang telah kami lakukan sudah valid.

Feature Encoding (Categoric)

Mengecek feature categorical yang masih memiliki nilai betype string/object

Dari hasil temuan, kita dapat menentukan beberapa encoding yang akan kita lakukan :

- **Label Encoding :**
  - LabelEncoder
  - Manually Mapped

Choice Determination:

- Pada proses `Label Encoding` ini kita menggunakan `Manually Mapped`, karena kita bisa menentukan secara fleksible urutan/order dari categorical feature

Label Encoding

1. Menggunakan `LabelEncoder`

```
# from sklearn.preprocessing import LabelEncoder

# cat = cat_str.copy()
# cat.remove("Marital_Status")

# le = LabelEncoder()

# for i in cat_str:
```

```
#     le.fit(df_train[i])
#     df_train[i] = le.transform(df_train[i])
#     df_test[i] = le.transform(df_test[i])
#     print(le.classes_)
```

1. Menggunakan Metode `Mapping`

```
# Mapping untuk Type_Income
type_income_mapping = {
    'Commercial associate': 4,
    'State servant': 3,
    'Working': 2,
    'Pensioner': 1
}

# Mapping untuk EDUCATION
education_mapping = {
    'Higher education': 4,
    'Secondary / secondary special': 3,
    'Incomplete higher': 2,
    'Lower secondary': 1
}

# Mapping untuk Marital_status
marital_status_mapping = {
    'Married': 3,
    'Separated/Widow': 2,
    'Single': 1
}

# Mapping untuk Housing_type
housing_type_mapping = {
    'House / apartment': 6,
    'Co-op apartment': 5,
    'Municipal apartment': 4,
    'Office apartment': 3,
    'Rented apartment': 2,
    'With parents': 1
}

# Mapping untuk GENDER
gender_mapping = {
    'M': 0,
    'F': 1
}

# Mapping untuk Car_Owner
car_owner_mapping = {
    'N': 0,
    'Y': 1
```

```
}

# Mapping untuk Propert_Owner
propert_owner_mapping = {
    'N': 0,
    'Y': 1
}

# Mapping untuk Income_sgmt
income_sgmt_mapping = {
    'High': 1,
    'Medium': 0,
    'Low': -1
}

# Mapping untuk Age_group
age_group_mapping = {
    'Senior Adult': 1,
    'Adult': 0,
    'Young Adult': -1
}

# Mapping untuk Type_Occupation
type_occupation_mapping = {
    'Managers': 18,
    'High skill tech staff': 17,
    'IT staff': 16,
    'Accountants': 15,
    'HR staff': 14,
    'Core staff': 13,
    'Medicine staff': 12,
    'Sales staff': 11,
    'Realty agents': 10,
    'Secretaries': 9,
    'Private service staff': 8,
    'Security staff': 7,
    'Drivers': 6,
    'Cooking staff': 5,
    'Cleaning staff': 4,
    'Waiters/barmen staff': 3,
    'Laborers': 2,
    'Low-skill Laborers': 1
}
df['GENDER'] = df['GENDER'].map(gender_mapping)
df['Car_Owner'] = df['Car_Owner'].map(car_owner_mapping)
df['Propert_Owner'] = df['Propert_Owner'].map(propert_owner_mapping)
df['Income_sgmt'] = df['Income_sgmt'].map(income_sgmt_mapping)
df_train['Income_sgmt'] =
df_train['Income_sgmt'].map(income_sgmt_mapping)
df_test['Income_sgmt'] =
```

```
df_test['Income_sgmt'].map(income_sgmt_mapping)
df['Age_group'] = df['Age_group'].map(age_group_mapping)
df['Type_Income'] = df['Type_Income'].map(type_income_mapping)
df['EDUCATION'] = df['EDUCATION'].map(education_mapping)
df['Marital_status'] =
df['Marital_status'].map(marital_status_mapping)
df['Housing_type'] = df['Housing_type'].map(housing_type_mapping)
df['Type_Occupation'] =
df['Type_Occupation'].map(type_occupation_mapping)

df_train.head()
```

```
    Ind_ID  GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
\
0  5024636       1          0              1         0         -2.815

1  5023692       1          0              1         0         -0.396

2  5148525       1          0              1         0          0.193

3  5028446       1          1              1         0         -0.021

4  5069289       1          0              1         0          0.992


   Type_Income  EDUCATION  Marital_status  Housing_type
Birthday_count  \
0            2          3               3             6           -
0.940
1            2          3               1             6
0.138
2            1          3               3             6           -
1.602
3            2          3               1             6           -
0.554
4            4          2               3             6
0.199

   Employed_days  Work_Phone  Phone  EMAIL_ID  Type_Occupation  \
0          0.236           0      0         0                2
1         -0.422           0      0         0               11
2          1.374           0      0         0               13
3         -0.254           0      0         0                2
4         -0.441           0      0         0               13

   Family_Members  label    Age  Tenure  Unemployment_duration  \
0               2      0  0.936  -0.159                 -0.458
1               1      0 -0.107   0.795                 -0.458
2               2      0  1.634  -1.511                  2.185
3               1      0  0.556   0.633                 -0.458
```

```
4               2      0 -0.195   0.812                  -0.458

   Is_currently_employed  Children_to_family_ratio  \
0                      1                    -0.643
1                      1                    -0.643
2                      0                    -0.643
3                      1                    -0.643
4                      1                    -0.643

   Children_employment_impact  Income_per_year_employed
Income_sgmt  \
0                      -0.621                    -1.754           -1

1                      -0.621                    -0.094            0

2                      -0.621                    -1.754            0

3                      -0.621                     0.061            0

4                      -0.621                     0.216            1


   Age_group  outlier
0          1        1
1          0        1
2          1        1
3          1        1
4          0        1

df_test.head()

    Ind_ID  GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
\
0  5047670       1          0              0         1          0.193

1  5135344       0          0              1         0         -0.778

2  5033728       1          1              1         1          0.429

3  5036795       0          1              1         0          0.548

4  5118115       0          0              0         1          0.992


   Type_Income  EDUCATION  Marital_status  Housing_type
Birthday_count  \
0            3          4               3             6
0.376
1            4          4               3             6          -
0.976
2            2          3               3             6
```

```
0.479
3            4           3                3             6           -
0.787
4            4           3                3             6
0.512

   Employed_days  Work_Phone  Phone  EMAIL_ID  Type_Occupation  \
0         -0.153           1      1         1               13
1          0.370           0      0         0                7
2         -0.029           1      1         0               13
3          0.195           0      1         0                2
4         -1.159           1      1         0                6

   Family_Members  label    Age  Tenure  Unemployment_duration  \
0               3      0 -0.376   0.519                -0.458
1               2      1  1.010  -0.640                -0.458
2               3      0 -0.469   0.355                -0.458
3               2      1  0.787  -0.057                -0.458
4               3      0 -0.469   1.289                -0.458

   Is_currently_employed  Children_to_family_ratio  \
0                      1                     1.484
1                      1                    -0.643
2                      1                     1.484
3                      1                    -0.643
4                      1                     1.484

   Children_employment_impact  Income_per_year_employed
Income_sgmt  \
0                       1.691                     0.168             0

1                      -0.621                     0.731            -1

2                       1.658                     0.316             0

3                      -0.621                     0.617             0

4                       1.774                    -0.003             1


   Age_group  outlier
0          0        1
1          1        1
2          0        1
3          1        1
4          0        1
```

Kesimpulan

Berdasarkan hasil pengecekan pada beberapa fitur yang telah diproses menggunakan encoding sebelumnya, dapat diketahui bahwa keseluruhan nilai telah beripe numeric sesuai dengan nilai yang kita assign. Sehingga dapat disimpulkan bahwa teknik fitur encoding yang telah kami lakukan sudah valid dan kami.

Feature Selection

1. Drop Unnecessary Feature

- Drop kolom `Ind_ID` karena memiliki banyak kategori dan tidak berguna untuk pemodelan
- Drop kolom `Phone` karena hanya memiliki satu nilai, tidak memberikan informasi yang signifikan terhadap model prediksi

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

2. Univariate Selection

```
# define X and y
X_train = df_train.drop(['label'], axis=1) #features
y_train = df_train['label'] #target
```

- **ANOVA F-value**

`ANOVA F-value` estimates the degree of linearity between the input feature (i.e., independent features) and the output feature (i.e., dependent feature). A high F-value indicates a high degree of linearity and a low F-value indicates a low degree of linearity.

Scikit-learn provides two functions to calculate F-value:

1. `sklearn.feature_selection.f_regression` for regression problems
2. `sklearn.feature_selection.f_classif` for classification problems

Disadvantage:

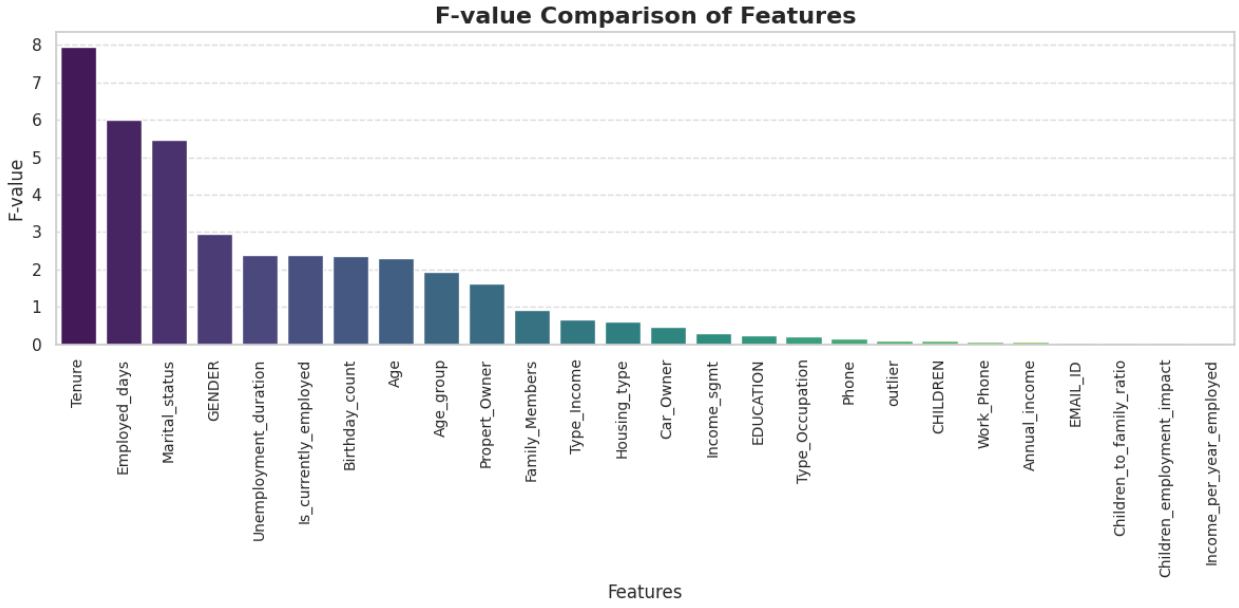
ANOVA F-value only captures the linear relationships between input and output feature.

```
X_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1161 entries, 0 to 1160
Data columns (total 26 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   GENDER                       1161 non-null   float64
 1   Car_Owner                    1161 non-null   int64  
 2   Propert_Owner                1161 non-null   int64  
 3   CHILDREN                     1161 non-null   int64
```

```
 4   Annual_income                1161 non-null   float64
 5   Type_Income                  1161 non-null   int64
 6   EDUCATION                    1161 non-null   int64
 7   Marital_status               1161 non-null   int64
 8   Housing_type                 1161 non-null   int64
 9   Birthday_count               1161 non-null   float64
 10  Employed_days                1161 non-null   float64
 11  Work_Phone                   1161 non-null   int64
 12  Phone                        1161 non-null   int64
 13  EMAIL_ID                     1161 non-null   int64
 14  Type_Occupation              1161 non-null   float64
 15  Family_Members               1161 non-null   int64
 16  Age                          1161 non-null   float64
 17  Tenure                       1161 non-null   float64
 18  Unemployment_duration        1161 non-null   float64
 19  Is_currently_employed        1161 non-null   int64
 20  Children_to_family_ratio     1161 non-null   float64
 21  Children_employment_impact   1161 non-null   float64
 22  Income_per_year_employed     1161 non-null   float64
 23  Income_sgmt                  1161 non-null   int64
 24  Age_group                    1161 non-null   int64
 25  outlier                      1161 non-null   int64
dtypes: float64(11), int64(15)
memory usage: 236.0 KB
```

```python
from sklearn.feature_selection import f_classif
feature_names = X_train.columns

# Calculate F-values
f_value = f_classif(X_train, y_train)

# Create a DataFrame with feature names and their corresponding F-values
fs = pd.DataFrame({
    "feature_names": feature_names,
    "f_value": f_value[0]
}).sort_values("f_value", ascending=False)

# Set up the color palette
palette = sns.color_palette("viridis", len(fs))

# Create the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(data=an, x="feature_names", y="f_value", palette=palette)

# Customize plot
plt.xticks(rotation=90, fontsize=10)  # Rotate x labels for better readability
plt.ylabel("F-value", fontsize=12)
plt.xlabel("Features", fontsize=12)
```

```
plt.title("F-value Comparison of Features", fontsize=16,
weight='bold')
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add horizontal grid
lines

# Adjust layout for better fit
plt.tight_layout()

# Show plot
plt.show()
```




```
# Assuming 'feature_importance' is an existing list where you want to
add these features
feature_importance = []

# Collecting the top 20 features from 'fs["feature_names"]' if not
already in the list
for i in an["feature_names"].values[:20]:
    if i not in feature_importance:
        feature_importance.append(i)

# Displaying the feature importance list in a more visually appealing
way
print("Top Features based on Importance:")
print("=" * 30)
for index, feature in enumerate(feature_importance, 1):
    print(f"{index}. {feature}")
```

```
Top Features based on Importance:
==============================
1. Tenure
```

```
2. Employed_days
3. Marital_status
4. GENDER
5. Unemployment_duration
6. Is_currently_employed
7. Birthday_count
8. Age
9. Age_group
10. Propert_Owner
11. Family_Members
12. Type_Income
13. Housing_type
14. Car_Owner
15. Income_sgmt
16. EDUCATION
17. Type_Occupation
18. Phone
19. outlier
20. CHILDREN
```

- **Variance Threshold**

Variance Threshold removes the features whose variance is below the pre-defined `threshold` value. By default, it removes all zero-variance features, i.e. features that have the same value in all samples.

Advantage:

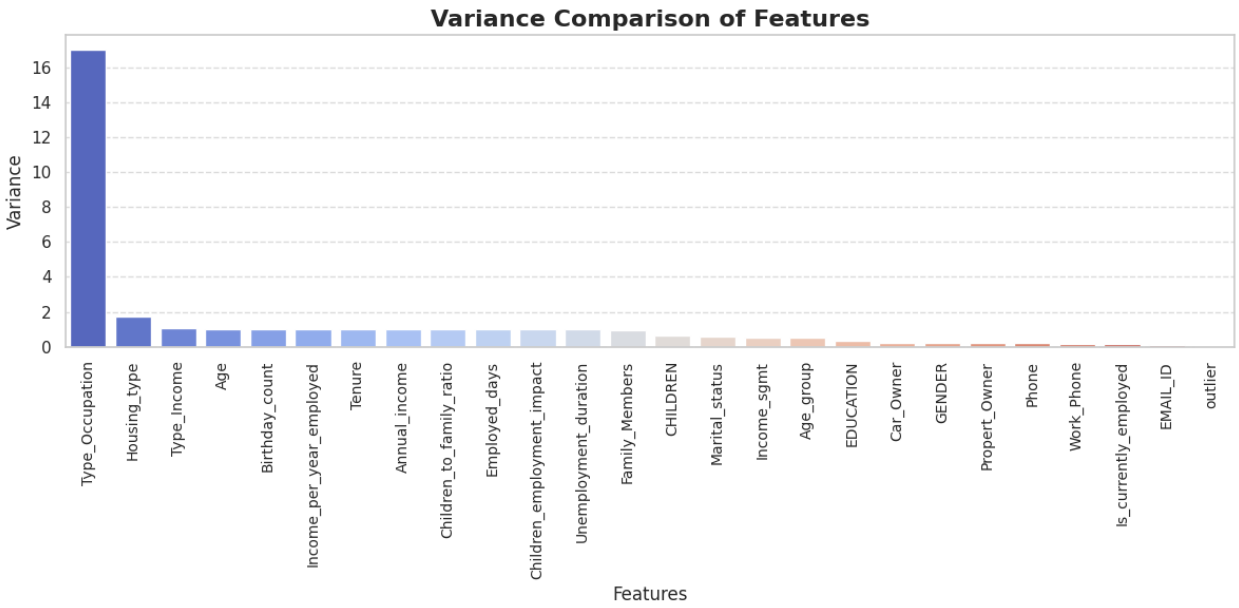
This can be used for unsupervised learning.

Disadvantage:

Variance Threshold only considers the relationship among the features but not the relationship between input features with the output feature.

```
# import VarianceThreshold
from sklearn.feature_selection import VarianceThreshold
# create VarianceThreshold object
selector = VarianceThreshold(threshold=0.0)
# train and transform
selector.fit_transform(X_train)
# print the name and variance of each feature
# for feature in zip(feature_names, selector.variances_):
#     print(feature)

fs = pd.DataFrame({
    "feature_names": feature_names,
    "variances": selector.variances_
}).sort_values("variances", ascending=False)
```

```
# Set up the color palette
palette = sns.color_palette("coolwarm", len(fs))

# Create the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(data=fs, x="feature_names", y="variances",
palette=palette)

# Customize plot
plt.xticks(rotation=90, fontsize=10)  # Rotate x labels for better
readability
plt.ylabel("Variance", fontsize=12)
plt.xlabel("Features", fontsize=12)
plt.title("Variance Comparison of Features", fontsize=16,
weight='bold')
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add horizontal grid
lines

# Adjust layout for better fit
plt.tight_layout()

# Show plot
plt.show()
```




```
# Assuming 'feature_importance' is an existing list where you want to
add these features
feature_importance = []

# Collecting the top 20 features from 'fs["feature_names"]' if not
already in the list
```

```
for i in fs["feature_names"].values[:20]:
    if i not in feature_importance:
        feature_importance.append(i)

# Displaying the feature importance list in a more visually appealing
way
print("Top Features based on Importance:")
print("=" * 30)
for index, feature in enumerate(feature_importance, 1):
    print(f"{index}. {feature}")
```

```
Top Features based on Importance:
==============================
1. Type_Occupation
2. Housing_type
3. Type_Income
4. Age
5. Birthday_count
6. Income_per_year_employed
7. Tenure
8. Annual_income
9. Children_to_family_ratio
10. Employed_days
11. Children_employment_impact
12. Unemployment_duration
13. Family_Members
14. CHILDREN
15. Marital_status
16. Income_sgmt
17. Age_group
18. EDUCATION
19. Car_Owner
20. GENDER
```

- **Mutual information**

Mutual information (MI) measures the dependence of one variable to another by quantifying the amount of information obtained about one feature, through the other feature. MI is symmetric and non-negative and is equal to zero if and only if two random variables are independent, and higher values mean higher dependency.

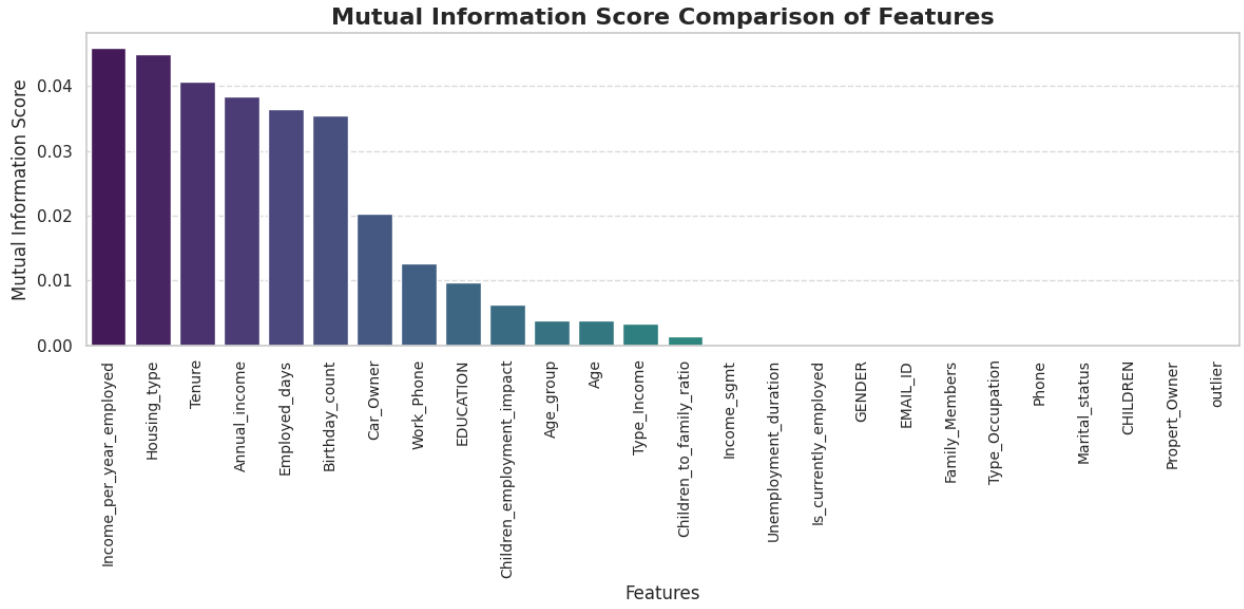
Scikit-learn provides two functions to calculate F-value:

- `sklearn.feature_selection.mutual_info_regression` for regression problems
- `sklearn.feature_selection.mutual_info_classif` for classification problems

Advantage:

MI can capture non-linear relationships between input and output feature.

```
# import mutual_info_classif
from sklearn.feature_selection import mutual_info_regression
# create mutual_info_classif object
MI_score = mutual_info_regression(X_train, y_train, random_state=0)
# Print the name and mutual information score of each feature
# for feature in zip(feature_names, MI_score):
#     print(feature)

fs = pd.DataFrame({
    "feature_names": feature_names,
    "MI_score": MI_score
}).sort_values("MI_score", ascending=False)

# Set up the color palette
palette = sns.color_palette("viridis", len(fs))

# Create the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(data=fs, x="feature_names", y="MI_score", palette=palette)

# Customize plot
plt.xticks(rotation=90, fontsize=10)  # Rotate x labels for better
readability
plt.ylabel("Mutual Information Score", fontsize=12)
plt.xlabel("Features", fontsize=12)
plt.title("Mutual Information Score Comparison of Features",
fontsize=16, weight='bold')
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add horizontal grid
lines

# Adjust layout for better fit
plt.tight_layout()

# Show plot
plt.show()
```




```python
# Assuming 'feature_importance' is an existing list where you want to add these features
feature_importance = []

# Collecting the top 20 features from 'fs["feature_names"]' if not already in the list
for i in fs["feature_names"].values[:20]:
    if i not in feature_importance:
        feature_importance.append(i)

# Displaying the feature importance list in a more visually appealing way
print("Top Features based on Importance:")
print("=" * 30)
for index, feature in enumerate(feature_importance, 1):
    print(f"{index}. {feature}")
```

```
Top Features based on Importance:
==============================
1. Income_per_year_employed
2. Housing_type
3. Tenure
4. Annual_income
5. Employed_days
6. Birthday_count
7. Car_Owner
8. Work_Phone
9. EDUCATION
10. Children_employment_impact
11. Age_group
12. Age
```

```
13. Type_Income
14. Children_to_family_ratio
15. Income_sgmt
16. Unemployment_duration
17. Is_currently_employed
18. GENDER
19. EMAIL_ID
20. Family_Members
```

- **Scikit-learn's SelectKBest**

SelectKBest selects the features using a function (in this case ANOVA F-value) and then "removes all but the k highest scoring features".

Statistical tests can be used to select those features that have the strongest relationship with the output variable. `Mutual information, ANOVA F-test and chi square` are some of the most popular methods of univariate feature selection.

ERROR :

`Input X must be non-negative`

The error message you got that:

Tells that: Pearson's chi-square test (goodness of fit) does not apply to negative values. It occurred because the chi-square test assumes frequencies distribution and a frequency can't be a negative number. But, `sklearn.feature_selection.chi2` asserts the input as non-negative.

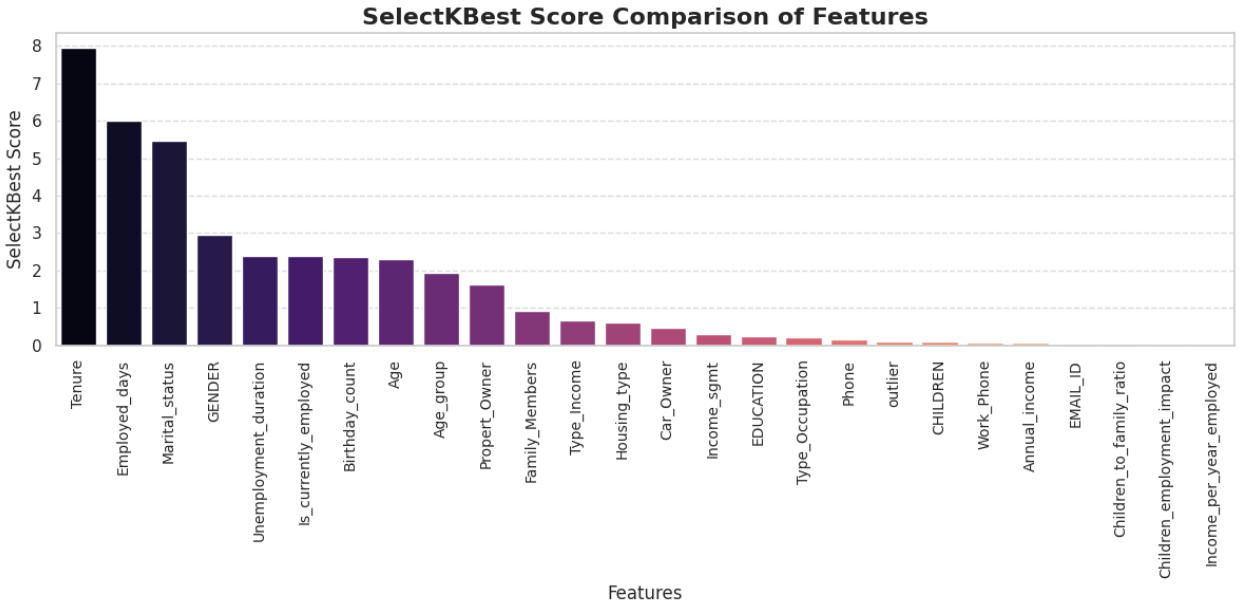
If data transformation is for some reason not possible (e.g. a negative value is an important factor), then you should pick another statistic to score your features:

```
sklearn.feature_selection.f_classif computes ANOVA f-value
sklearn.feature_selection.mutual_info_classif

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif,
mutual_info_classif

# apply SelectKBest class to extract top 10 best features

# computes chi2
# semuanya harus positif
# bestfeatures = SelectKBest(score_func=chi2, k=10)

# computes ANOVA f-value
bestfeatures = SelectKBest(score_func=f_classif, k=10)

fit = bestfeatures.fit(X_train, y_train)
```

```
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X_train.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe
columns
featureScores.sort_values('Score', ascending=False, inplace=True)
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
                    Specs  Score
17                 Tenure  7.943
10          Employed_days  6.009
7           Marital_status  5.471
0                   GENDER  2.958
18  Unemployment_duration  2.402
19  Is_currently_employed  2.402
9           Birthday_count  2.351
16                     Age  2.317
24             Age_group  1.945
2            Propert_Owner  1.617
```

```
# Set up the color palette
palette = sns.color_palette("magma", len(featureScores))

# Create the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(data=featureScores, x="Specs", y="Score", palette=palette)

# Customize plot
plt.xticks(rotation=90, fontsize=10)  # Rotate x labels for better
readability
plt.ylabel("SelectKBest Score", fontsize=12)
plt.xlabel("Features", fontsize=12)
plt.title("SelectKBest Score Comparison of Features", fontsize=16,
weight='bold')
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Add horizontal grid
lines

# Adjust layout for better fit
plt.tight_layout()

# Show plot
plt.show()
```



3. Feature Importance

You can get the feature importance of each feature of your dataset by using the feature importance property of the model.

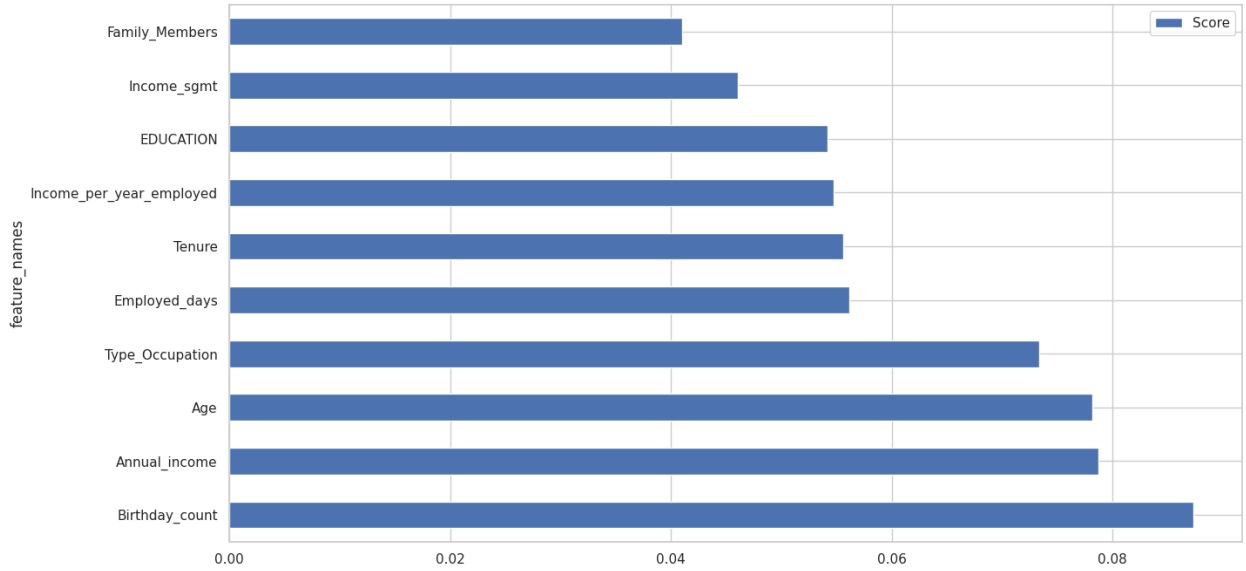
Feature importance gives you a score for each feature of your data, the higher the score more important or relevant is the feature towards your output variable.

Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top 10 features for the dataset.

```
from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()
model.fit(X_train,y_train)
print(model.feature_importances_) #use inbuilt class
feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_,
index=X_train.columns)
feat_importances =
pd.DataFrame(feat_importances).reset_index(names="feature_names") \
    .rename(columns={0:"Score"}) \
    .sort_values("Score", ascending=False).reset_index(drop=True)
feat_importances.nlargest(10, "Score").plot(x="feature_names",
y="Score", kind='barh')
plt.show()
```

```
[0.02874933 0.03463354 0.03221124 0.02135011 0.07874544 0.03972664
 0.05422394 0.03300248 0.03327969 0.08734826 0.05613655 0.0242704
 0.03103921 0.01411122 0.0733389  0.04105368 0.07820527 0.05562964
```

```
 0.0038034  0.00296535 0.02010045 0.02375177 0.05472143 0.04603971
 0.02826824 0.0032941 ]
```




```python
# Assuming 'feature_importance' is an existing list where you want to add these features
feature_importance = []

# Collecting the top 20 features from 'fs["feature_names"]' if not already in the list
for i in feat_importances["feature_names"].values[:20]:
    if i not in feature_importance:
        feature_importance.append(i)

# Displaying the feature importance list in a more visually appealing way
print("Top Features based on Importance:")
print("=" * 30)
for index, feature in enumerate(feature_importance, 1):
    print(f"{index}. {feature}")
```

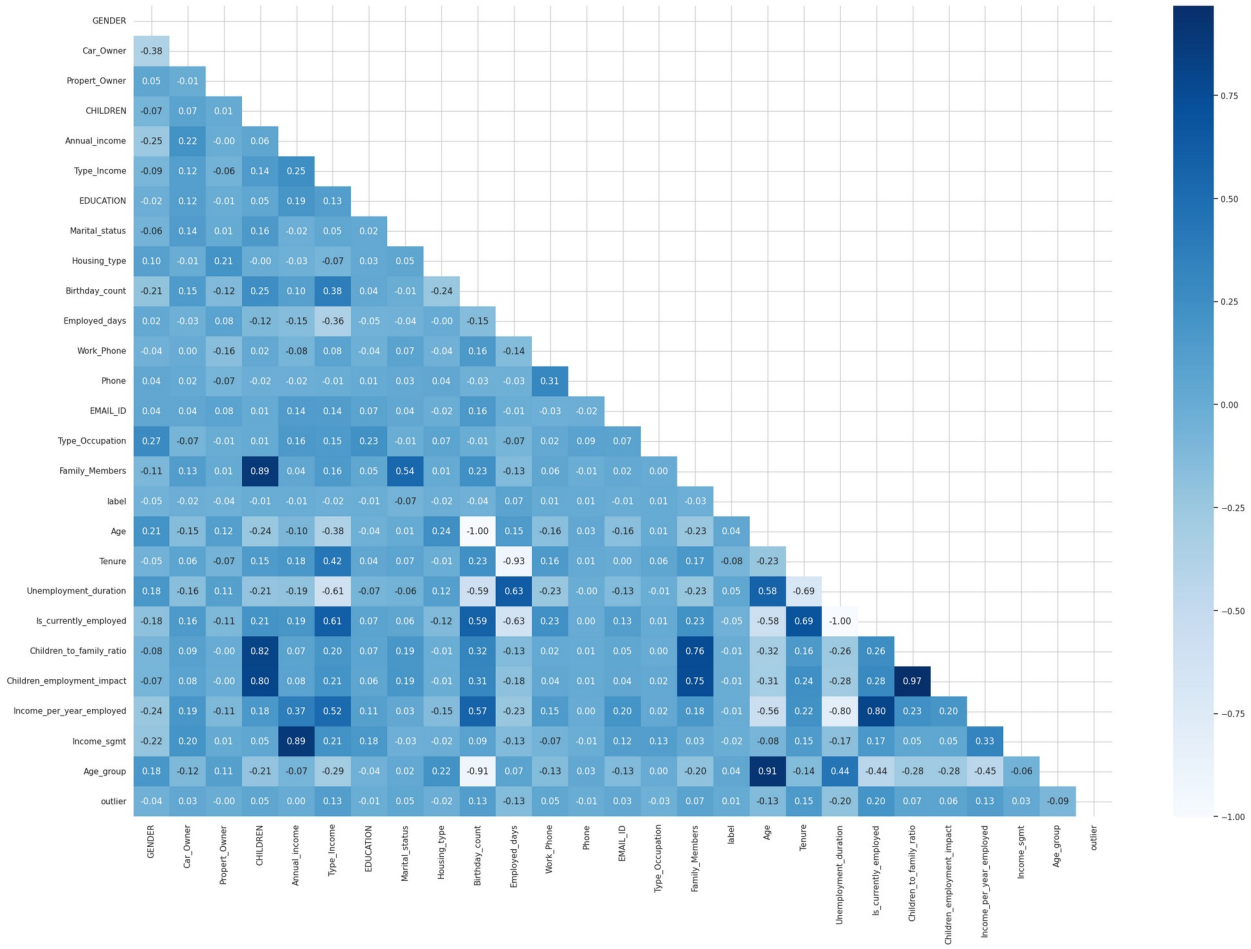
```
Top Features based on Importance:
==============================
1. Birthday_count
2. Annual_income
3. Age
4. Type_Occupation
5. Employed_days
6. Tenure
7. Income_per_year_employed
8. EDUCATION
9. Income_sgmt
```

```
10. Family_Members
11. Type_Income
12. Car_Owner
13. Housing_type
14. Marital_status
15. Propert_Owner
16. Phone
17. GENDER
18. Age_group
19. Work_Phone
20. Children_employment_impact
```

4. Correlation Matrix with Heatmap

- Correlation states how the features are related to each other or the target variable.
- Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable)
- Heatmap makes it easy to identify which features are most related to the target variable, we will plot heatmap of correlated features using the seaborn library.
- Cek Feature Redundan pada korelasi Antar Feature, Drop salah satunya, yang rendah korelsinya dengan Response (target)

```
plt.figure(figsize=(30,20))
corr = df_train.corr(numeric_only=True)
mask = np.triu(np.ones_like(corr, dtype=np.bool_))
sns.heatmap(corr, cmap='Blues', annot=True, fmt='.2f', mask=mask)

<Axes: >
```



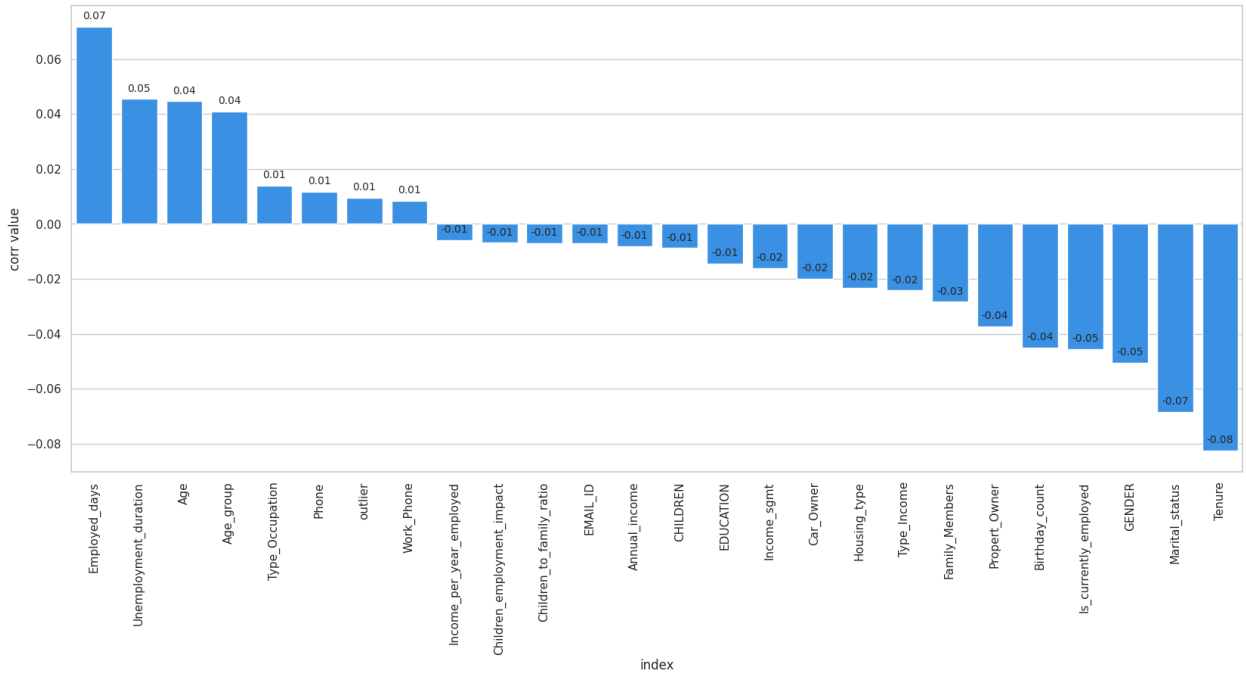
Checking Correlation with Target (Response)

```
corr = df_train.corrwith(df_train["label"], numeric_only=True)
corr = corr.reset_index(name='corr value')
corr["Corr Type"] = corr["corr value"].apply(lambda x : "Positif" if x
>= 0 else "Negatif")
corr["corr value"] = corr["corr value"].apply(lambda x : abs(x))
corr = corr.sort_values('corr value', ascending=False,
ignore_index=True)
corr.head(10)
```

	index	corr value	Corr Type
0	label	1	Positif
1	Tenure	0.083	Negatif
2	Employed_days	0.072	Positif
3	Marital_status	0.069	Negatif
4	GENDER	0.050	Negatif
5	Is_currently_employed	0.045	Negatif
6	Unemployment_duration	0.045	Positif
7	Birthday_count	0.045	Negatif

```
8                    Age       0.045   Positif
9              Age_group       0.041   Positif
```

```
corr = df_train.corrwith(df_train["label"], numeric_only=True)
corr = corr.reset_index(name='corr value')
corr = corr.sort_values('corr value', ascending=False)[1:]

plt.figure(figsize=(20, 8))
ax = sns.barplot(x='index', y="corr value", data=corr,
order=corr["index"], color='dodgerblue')
for p in ax.patches:
    ax.annotate(
        f'{p.get_height():0.2f}',
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha = 'center',
        va = 'center',
        xytext = (0, 10),
        fontsize=10,
        textcoords = 'offset points')
plt.xticks(rotation=90)
plt.show()
```




```
target = "label"
high_corr_cols = [i for i in list(corr[corr["corr value"] > 0.05]
["index"].values) if i != target]
print(high_corr_cols)
```

```
['Employed_days']
```

```
for i in corr["index"].values[:20]:
    if i not in feature_importance:
        feature_importance.append(i)
feature_importance
```

```
['Birthday_count',
 'Annual_income',
 'Age',
 'Type_Occupation',
 'Employed_days',
 'Tenure',
 'Income_per_year_employed',
 'EDUCATION',
 'Income_sgmt',
 'Family_Members',
 'Type_Income',
 'Car_Owner',
 'Housing_type',
 'Marital_status',
 'Propert_Owner',
 'Phone',
 'GENDER',
 'Age_group',
 'Work_Phone',
 'Children_employment_impact',
 'Unemployment_duration',
 'outlier',
 'Children_to_family_ratio',
 'EMAIL_ID',
 'CHILDREN']
```

5. Multicollinearity Check (Drop Redundancy)

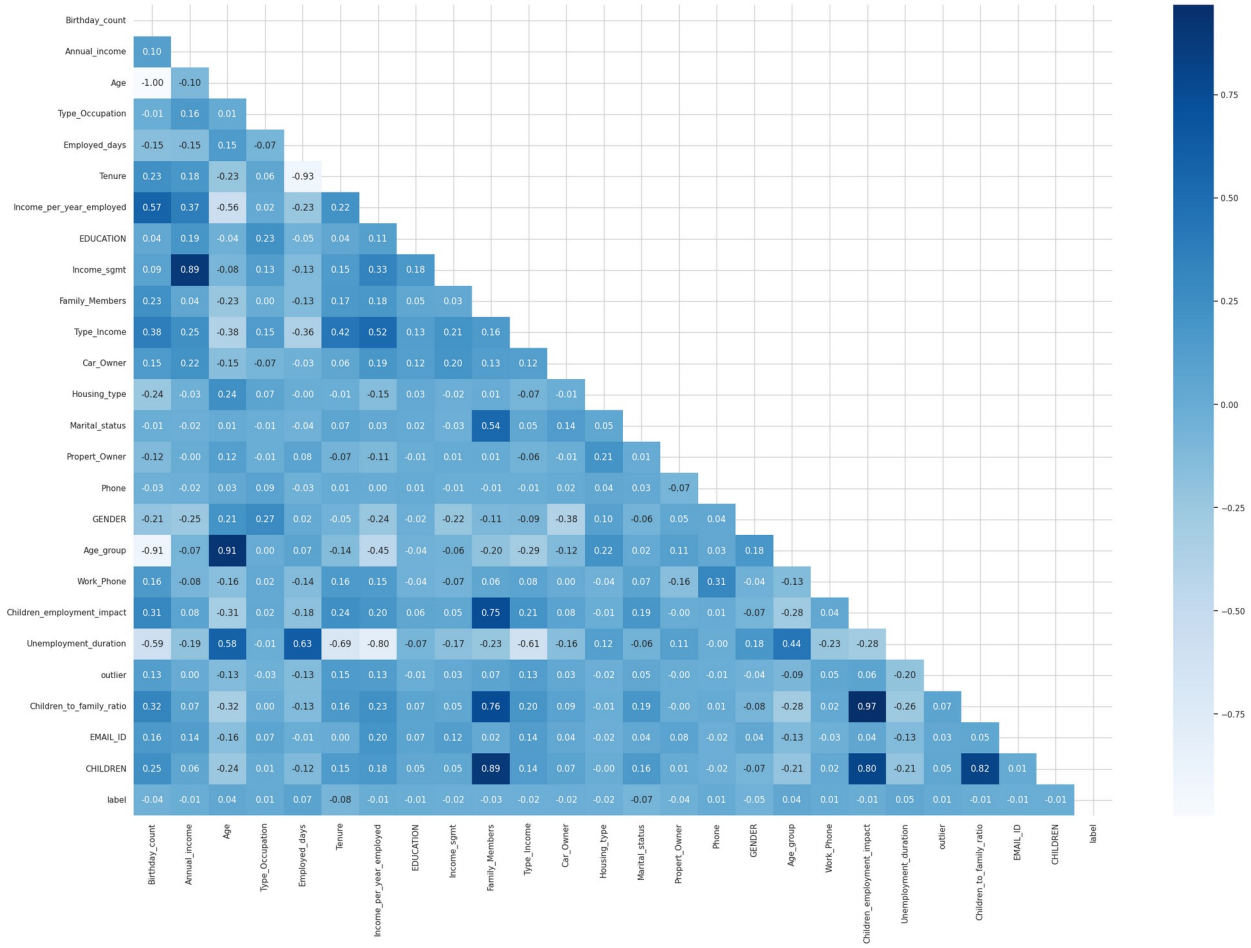
- Multicollinearity means independent variables in a model are correlated.
- Multicollinearity among independent variables can reduce the performance of the model.
- Multicollinearity reduces the statistical significance of the independent variables.

Dari feature yang telah kita pilih dari gabungan Top 20 akan di lakukan pengecekan kembali melalui `redudansi antar feature`. Pada proses ini kita memilih antar feature yang memiliki korelasi diatas `threshold > 0.70`, yang kemudian akan di `bandingkan korelasinya dengan Target` untuk `drop salah satu feature`

1. Correlation Coefficient

```
plt.figure(figsize=(30,20))
corr = df_train[feature_importance+["label"]].corr(numeric_only=True)
mask = np.triu(np.ones_like(corr, dtype=np.bool_))
sns.heatmap(corr, cmap='Blues', annot=True, fmt='.2f', mask=mask)
```

```
<Axes: >
```



Manampilkan Korelasi Feature > Threshold 0.70

Akan ada info mengenai Feature apa saja yang perlu di drop

```
def corrtarget(x):
    target = "label"
    return df_train[x].corr(df_train[target])

def corrresp(x):
    target = "label"
    col1 = x["A"]
    col2 = x["B"]

    cor1 = df_train[col1].corr(df_train[target])
    cor2 = df_train[col2].corr(df_train[target])

    if cor1 < cor2:
        return col1
    else:
```

```
        return col2
    return col1

corr_matrix = df_train[feature_importance].corr()
target = "label"

# Flatten correlation matrix.
flat_cm = corr_matrix.stack().reset_index()
flat_cm.columns = ['A', 'B', 'correlation']
flat_cm = flat_cm.loc[flat_cm.correlation < 1, :]
flat_cm = flat_cm.sort_values("correlation", ascending=False)
redundan = flat_cm[flat_cm["correlation"] >=
0.7].reset_index(drop=True)
redundan['A vs Target'] = redundan['A'].apply(lambda x: corrtarget(x))
redundan['B vs Target'] = redundan['B'].apply(lambda x: corrtarget(x))
redundan = redundan.drop_duplicates(subset=["correlation"])
redundan["drop"] = redundan.apply(corrresp, axis=1)
redundan
```

```
                              A                              B
correlation  \
0   Children_employment_impact    Children_to_family_ratio
0.967
2                   Age_group                            Age
0.914
4                     CHILDREN               Family_Members
0.894
6                Annual_income                  Income_sgmt
0.886
8                     CHILDREN    Children_to_family_ratio
0.817
10                    CHILDREN  Children_employment_impact
0.802
12              Family_Members    Children_to_family_ratio
0.759
14  Children_employment_impact               Family_Members
0.747

    A vs Target  B vs Target                       drop
0        -0.007       -0.007  Children_to_family_ratio
2         0.041        0.045                 Age_group
4        -0.009       -0.028            Family_Members
6        -0.008       -0.016               Income_sgmt
8        -0.009       -0.007                  CHILDREN
10       -0.009       -0.007                  CHILDREN
12       -0.028       -0.007            Family_Members
14       -0.007       -0.028            Family_Members
```

2. VIF (Variance Inflation Factor)

How do we calculate Multi-collinearity ?

This is where VIF (Variance Inflation Factor) come into the picture. The variance inflation factor (VIF) identifies the strength of correlation among the predictors (features).

Multicollinearity reduces the statistical significance of the independent variables. VIF is used to detect these variables. A large variance inflation factor (VIF) on an independent variable indicates a highly collinear relationship to the other variables that should be considered or adjusted for in the structure of the model and selection of independent variables.

$$VIF_i = \frac{1}{1 - R_i^2}$$

VIF value will always be greater than 1. Here are some rules for VIF

- 1 = not correlated.
- Between 1 and 5 = moderately correlated.
- Greater than 5 = highly correlated.

If all the independent variables are orthogonal to each other, then VIF = 1.0. If there is perfect correlation, then VIF = infinity.

```
# calculate VIF scores for each feature
from statsmodels.stats.outliers_influence import
variance_inflation_factor as vif
from statsmodels.tools.tools import add_constant

X = add_constant(df_train[feature_importance])

vif_df = pd.DataFrame([vif(X.values, i)
               for i in range(X.shape[1])],
              index=X.columns).reset_index()
vif_df.columns = ['feature','vif_score']
vif_df = vif_df.loc[vif_df.feature!='const']
vif_df['vif_score'] = round(vif_df['vif_score'], 4)
```

```
vif_df.sort_values("vif_score", ascending=False, inplace=True)
vif_df
```

	feature	vif_score
3	Age	134.210
1	Birthday_count	133.120
10	Family_Members	27.347
21	Unemployment_duration	26.568
6	Tenure	23.866
25	CHILDREN	21.097
23	Children_to_family_ratio	20.775
20	Children_employment_impact	19.934
7	Income_per_year_employed	15.434
5	Employed_days	9.599
18	Age_group	6.788
2	Annual_income	6.227
14	Marital_status	5.654
9	Income_sgmt	4.733
11	Type_Income	1.704
17	GENDER	1.407
4	Type_Occupation	1.257
12	Car_Owner	1.244
19	Work_Phone	1.240
16	Phone	1.144
13	Housing_type	1.122
8	EDUCATION	1.113
24	EMAIL_ID	1.104
15	Propert_Owner	1.100
22	outlier	1.055

Drop Redundant Features

```
hold = ["Tenure", "Unemployment_duration"]
for i in list(redundan["drop"].unique()):
    if i not in hold:
        feature_importance.remove(i)

feature_importance = sorted(feature_importance)
feature_importance
```

```
['Age',
 'Annual_income',
 'Birthday_count',
 'Car_Owner',
 'Children_employment_impact',
 'EDUCATION',
 'EMAIL_ID',
 'Employed_days',
 'GENDER',
```

```
 'Housing_type',
 'Income_per_year_employed',
 'Marital_status',
 'Phone',
 'Propert_Owner',
 'Tenure',
 'Type_Income',
 'Type_Occupation',
 'Unemployment_duration',
 'Work_Phone',
 'outlier']
```

Kesimpulan

Berdasarkan hasil pengecekan pada beberapa fitur yang telah diproses menggunakan feature selection :

- Drop Unnecessary Feature
- Univariate Selection
  - Anova F-value
  - Variance Threshold
  - Mutual Information
  - SelectKBest
- Feature Importance
- Pearson Correlation
- Multicollinearity Check (Drop Redundancy)

Maka telah didapatkan feature yang akan digunakan pada proses modelling sebagai berikut :

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

```
     Age  Annual_income  Birthday_count  Car_Owner  \
0  0.936         -2.815          -0.940          0
1 -0.107         -0.396           0.138          0
2  1.634          0.193          -1.602          0
3  0.556         -0.021          -0.554          1
4 -0.195          0.992           0.199          0

   Children_employment_impact  EDUCATION  EMAIL_ID  Employed_days
GENDER  \
0                      -0.621          3         0          0.236
1
1                      -0.621          3         0         -0.422
1
2                      -0.621          3         0          1.374
1
3                      -0.621          3         0         -0.254
1
```

```
4                      -0.621          2         0         -0.441
1

   Housing_type  Income_per_year_employed  Marital_status  Phone  \
0             6                    -1.754               3      0
1             6                    -0.094               1      0
2             6                    -1.754               3      0
3             6                     0.061               1      0
4             6                     0.216               3      0

   Propert_Owner  Tenure  Type_Income  Type_Occupation
Unemployment_duration  \
0              1  -0.159            2                2
-0.458
1              1   0.795            2               11
-0.458
2              1  -1.511            1               13
2.185
3              1   0.633            2                2
-0.458
4              1   0.812            4               13
-0.458

   Work_Phone  outlier
0           0        1
1           0        1
2           0        1
3           0        1
4           0        1
```

Handling Imbalanced Data

Status risiko highly imbalanced, dengan 15% Response dan 85% No Response. Itu sebabnya diperlukan resampling.

Note: Saat menerapkan machine learning algorithms dengan data yang tidak seimbang, model yang diperoleh akan lebih condong ke kelas mayoritas. Artinya model akan memprediksi kelas mayoritas bukan kelas minoritas.

Jika kita ingin melakukan klasifikasi, maka seharusnya melakukan `stratified train_test_split` terlebih dahulu untuk menjaga ketidakseimbangannya (imbalance). Sehingga dataset test dan train memiliki distribusi yang sama, kemudian jangan pernah menyentuh test set lagi. Kemudian lakukan pengambilan sampel ulang hanya pada data train.

Summary : You must apply `SMOTE` after splitting into `training and test`, not before. Doing SMOTE before is bogus and defeats the purpose of having a separate test set.

Separate Dataset Xy Train Test

```
X_train = df_train.drop(['label'], axis=1)
[feature_importance].reset_index(drop=True) #features
y_train = df_train['label'].reset_index(drop=True) #target
print(X_train.shape, y_train.shape)
```

```
(1161, 20) (1161,)
```

```
X_test = df_test.drop(['label'], axis=1)
[feature_importance].reset_index(drop=True) #features
y_test = df_test['label'].reset_index(drop=True) #target
print(X_test.shape, y_test.shape)
```

```
(387, 20) (387,)
```

```
X = pd.concat([X_train,X_test])
y = pd.concat([y_train,y_test])
```

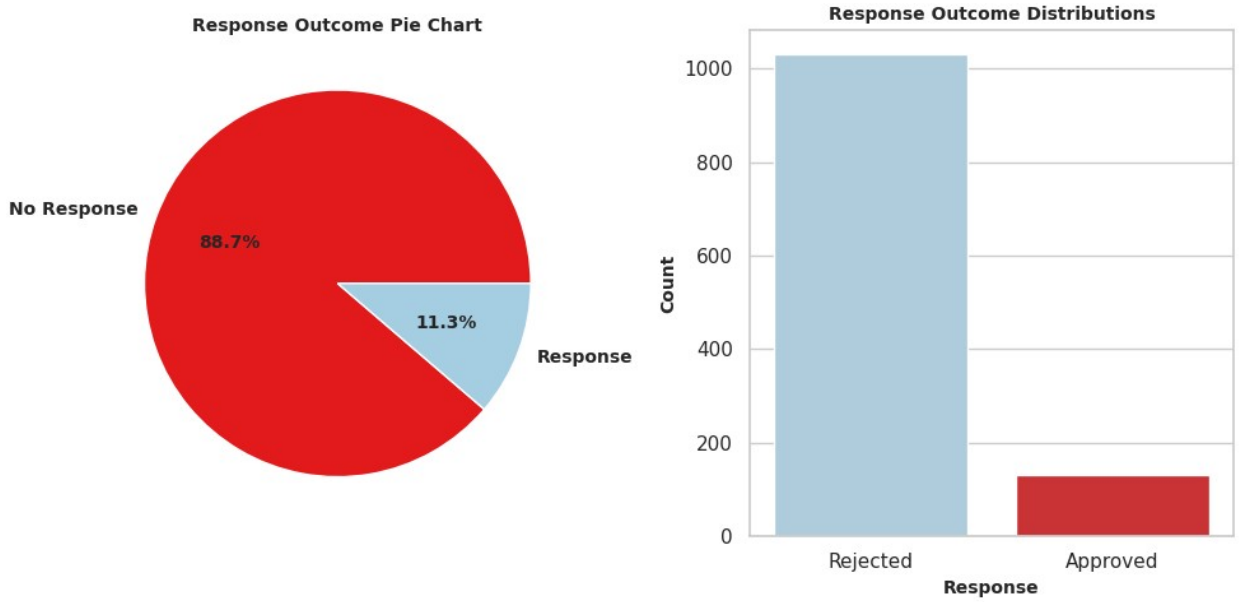
```
X.shape
```

```
(1548, 20)
```

```
y.shape
```

```
(1548,)
```

```
fig = plt.figure(figsize = (10, 5))

plt.subplot(121)
plt.pie(y_train.value_counts(),
        labels = ['No Response', 'Response'],
        autopct = '%.1f%%',
        radius = 1,
        colors=["#e31a1c", "#a6cee3"],
        textprops={'fontsize': 10, 'fontweight': 'bold'})
plt.title('Response Outcome Pie Chart', fontsize = 10, fontweight =
'bold')

plt.subplot(122)
resp = y_train.apply(lambda x: "Rejected" if x == 0 else "Approved")
t = sns.countplot(x=resp, palette=["#a6cee3", "#e31a1c"])
t.set_xlabel('Response', fontweight = 'bold', fontsize = 10)
t.set_ylabel('Count', fontweight = 'bold', fontsize = 10)

plt.title('Response Outcome Distributions', fontsize = 10, fontweight
= 'bold')
plt.tight_layout()
```

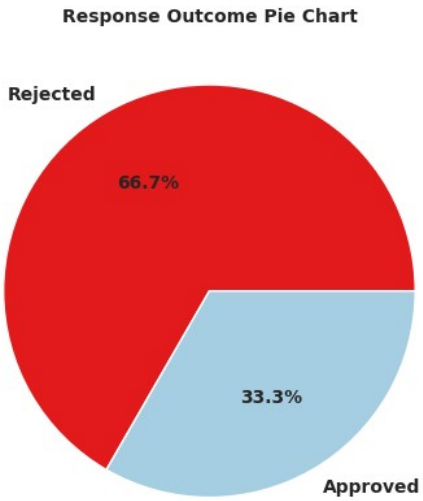


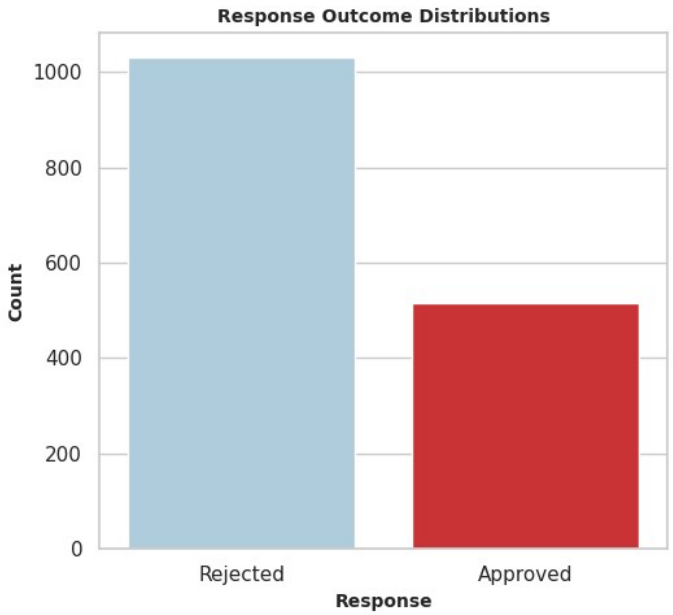

```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler, SMOTE

print('Before OverSampling, the shape of X_train:
{}'.format(X_train.shape))
print('Before OverSampling, the shape of y_train: {} \
n'.format(y_train.shape))

print("Before OverSampling, counts of label '1':
{}".format(sum(y_train == 1))) # Response
print("Before OverSampling, counts of label '0': {} \
n".format(sum(y_train == 0))) # No Response

# Undersampling
# us = RandomUnderSampler(sampling_strategy = 0.5)
# X_balanced_res, y_balanced_res = us.fit_resample(X_train,y_train)

# Oversampling
# os = RandomOverSampler(sampling_strategy = 0.5)
# X_balanced_res, y_balanced_res = os.fit_resample(X_train,y_train)

# Oversampling SMOTE
sm = SMOTE(sampling_strategy=0.5, random_state = 2)
X_balanced_res, y_balanced_res = sm.fit_resample(X_train,y_train)

print('After OverSampling, the shape of X_train:
{}'.format(X_balanced_res.shape))
print('After OverSampling, the shape of y_train: {} \
n'.format(y_balanced_res.shape))

print("After OverSampling, counts of label '1':
```

```
{}".format(sum(y_balanced_res == 1)))
print("After OverSampling, counts of label '0':
{}".format(sum(y_balanced_res == 0)))

X_train = X_balanced_res
y_train = y_balanced_res
```

```
Before OverSampling, the shape of X_train: (1161, 20)
Before OverSampling, the shape of y_train: (1161,)

Before OverSampling, counts of label '1': 131
Before OverSampling, counts of label '0': 1030

After OverSampling, the shape of X_train: (1545, 20)
After OverSampling, the shape of y_train: (1545,)

After OverSampling, counts of label '1': 515
After OverSampling, counts of label '0': 1030
```

```
fig = plt.figure(figsize = (10, 5))

plt.subplot(121)
plt.pie(y_train.value_counts(),
        labels = ['Rejected', 'Approved'],
        autopct = '%.1f%%',
        radius = 1,
        colors=["#e31a1c", "#a6cee3"],
        textprops={'fontsize': 10, 'fontweight': 'bold'})
plt.title('Response Outcome Pie Chart', fontsize = 10, fontweight =
'bold')

plt.subplot(122)
resp = y_train.apply(lambda x: "Rejected" if x == 0 else "Approved")
t = sns.countplot(x=resp, palette=["#a6cee3", "#e31a1c"])
t.set_xlabel('Response', fontweight = 'bold', fontsize = 10)
t.set_ylabel('Count', fontweight = 'bold', fontsize = 10)

plt.title('Response Outcome Distributions', fontsize = 10, fontweight
= 'bold')
plt.tight_layout()
```





====== STAGE 3 ======

`Stage 3 (ML Modelling & Evaluation)`

Machine Learning Techniques

Here are some algorithms that will be tested to determine the best model to predict customer response in Marketing Campign:

1. Decision Tree
2. Random Forest
3. Logistic Regression
4. Gaussian Naive Bayes
5. K-Nearest Neighbor
6. MLP Classifier (Neural Network)
7. Adaboost Classifier
8. XGBoost Classifier
9. LGBM Classifier
10. Gradient Boosting Classifier
11. Support Vector Machine
12. Catboost Classifier

Target Explanation

- `True Positive (TP)`, when a case was positive and predicted positive
- `False Positive (FP)`, when a case was negative but predicted positive
- `True Negative (TN)`, when a case was negative and predicted negative

- `False Negative (FN)`, when a case was positive but predicted negative
- `Positive / 1` = Customer yang Response
- `Negative / 0` = Customer yang tidak Response
- `False Negative` = Customer yang Response tetapi diprediksi tidak Response
- `False Positive` = Customer yang tidak Response tetapi diprediksi Response

Parameter Evaluasi Model

- `Precision` as Primary Parameter Evaluation
  - Meningkatkan Response Rate dan Cost Marketing
  - Mereduksi False Positif (Customer yang diprediksi akan merespon, namun kenyataannya tidak)
- `Recall` as Secondary Parameter Evaluation
  - Mengoptimalkan Revenue Rate
  - Mereduksi False Negative (Customer yang diprediksi tidak mengikuti campaign, namun pada kenyataannya berkeinginan ikut campaign)
- `F1 Score` for check positif and negative score (Imbalance Data). Sama pentingnya untuk masalah kita, kita membutuhkan trade-off antara Precision dan Recall, oleh karena itu, kami menggunakan skor f1 sebagai metrik. Skor f1 didefinisikan sebagai rata-rata harmonik dari Precision dan Recall.

Interpretation

- `Precision` – What percent of your predictions were correct? :

  Dari test data yang diprediksi positif (Response), berarti x% yg sesungguhnya positif (Response)

- `Recall` – What percent of the positive cases did we catch? :

  Dari semua yang sebenarnya positif (Response), yang berhasil diprediksi positif (Response) x%

`Target Metrics` = `Precision` / `Recall` & `F1 Score`

- `Precision` adalah rasio prediksi benar positif dibandingkan dengan keseluruhan hasil yang diprediksi positf. Pilih algoritma yang memiliki `Precision` tinggi, jika skenario yang dipilih adalah `False Negative lebih baik terjadi daripada False Positif`.

  `Precission = (TP) / (TP+FP)`

- `Recall` adalah rasio kasus dengan prediksi benar positif dibandingkan dengan keseluruhan data yang benar positif. Pilih algoritma yang memiliki `Recall` tinggi, jika skenario yang dipilih adalah `False Positive lebih baik terjadi daripada False Negative`.

  `Recall = (TP) / (TP + FN)`

- `F1-Score` atau dikenal juga dengan nama F-Measure didapatkan dari perbandingan rata-rata presisi dengan recall yang dibobotkan

  `F1 Score = 2 * (Recall*Precission) / (Recall + Precission)`

Description Classification Report :

It is a python method under sklearn metrics API, useful when we need class-wise metrics alongside global metrics. It provides precision, recall, and F1 score at individual and global levels.

- The `precision` will be "how many are correctly classified among that class" (Percentage of correct positive predictions relative to total positive predictions.)
- The `recall` means "how many of this class you find over the whole number of element of this class". (Percentage of correct positive predictions relative to total actual positives.)
- The `f1-score` is the harmonic mean between precision & recall. The closer to 1, the better the model.
- The `support` is the number of occurence of the given class in dataset. These values simply tell us how many players belonged to each class in the test dataset. (to check balanced dataset/Proportion)
- `Accuracy` is The sum of the true positives and true negatives over the total number of samples.
- `Macro Average` is The mean average of the precision/recall/F1-score of all the classes.
- `Weighted Average` is Calculates the scores for each class independent of one another but when it adds them together it takes into account the number of true classifications of each class.

Fbeta-Measure

The `F-measure` balances the `precision and recall`

On some problems, we might be interested in an F-measure with more attention put on precision, such as when false positives are more important to minimize, but false negatives are still important.

The solution is the `Fbeta-measure`

The Fbeta-measure measure is an abstraction of the F-measure where the balance of precision and recall in the calculation of the harmonic mean is controlled by a coefficient called beta.

`Fbeta = ((1 + beta^2) * Precision * Recall) / (beta^2 * Precision + Recall)`

The choice of the beta parameter will be used in the name of the Fbeta-measure.

For example, a beta value of 2 is referred to as F2-measure or F2-score. A beta value of 1 is referred to as the F1-measure or the F1-score.

Three common values for the beta parameter are as follows:

- `F0.5-Measure (beta=0.5)`: More weight on precision, less weight on recall.
- `F1-Measure (beta=1.0)`: Balance the weight on precision and recall.
- `F2-Measure (beta=2.0)`: Less weight on precision, more weight on recall

References

- https://rey1024.medium.com/mengenal-accuracy-precission-recall-dan-specificity-serta-yang-diprioritaskan-b79ff4d77de8
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- https://www.statology.org/sklearn-classification-report/
- https://muthu.co/understanding-the-classification-report-in-sklearn/#:~:text=A%20Classification%20report%20is%20used,classification%20report%20as%20shown%20below.
- https://machinelearningmastery.com/fbeta-measure-for-machine-learning/


```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB # Gaussian Naive Bayes
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC # Support Vector Machine/Classifier
from sklearn.neural_network import MLPClassifier # Neural Network
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
import lightgbm as lgb
from lightgbm import LGBMClassifier

from catboost import CatBoostClassifier  # Import CatBoost Classifier

from sklearn import metrics
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, recall_score,
precision_score, f1_score, fbeta_score
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.inspection import permutation_importance
import shap

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
```

```
from scipy.stats import uniform

import re
import warnings
warnings.filterwarnings('ignore')
```

Function for Model Evaluation

```
train_classifier_list = []
train_modelname_list = []
train_accuracy_list = []
train_precision_list = []
train_recall_list = []
train_fbeta_score_list= []
train_f1_score_list= []
train_roc_auc_score_list = []
train_cross_val_f1_list = []
train_cross_val_rocauc_list = []

test_classifier_list = []
test_modelname_list = []
test_accuracy_list = []
test_precision_list = []
test_recall_list = []
test_fbeta_score_list= []
test_f1_score_list= []
test_roc_auc_score_list = []
test_cross_val_f1_list = []
test_cross_val_rocauc_list = []

from sklearn.utils import shuffle
from sklearn.model_selection import KFold
X_s, y_s = shuffle(X, y, random_state=42)
kf = KFold(10, shuffle=True, random_state=0)
```

- Function Evaluation for Recap Classification

```
def eval_classification(model, model_name, save=True):
    # predict train
    y_train_pred = model.predict(X_train)
    y_train_pred_prob = model.predict_proba(X_train)

    # predict test
    y_test_pred = model.predict(X_test)
    y_test_pred_prob = model.predict_proba(X_test)

    # cross validation
    cv_score_f1 = cross_validate(model, X_s, y_s, cv=kf, scoring='f1',
return_train_score=True)
    cv_score_rocauc = cross_validate(model, X_s, y_s, cv=kf,
```

```
scoring='roc_auc', return_train_score=True)


    accuracy_train = round(accuracy_score(y_train, y_train_pred), 3)
    precision_train = round(precision_score(y_train, y_train_pred), 3)
    recall_train = round(recall_score(y_train, y_train_pred), 3)
    fbeta_s_train = round(fbeta_score(y_train, y_train_pred,
beta=0.5), 3)
    f1_s_train = round(f1_score(y_train, y_train_pred), 3)
    csf_score_train = round(cv_score_f1['train_score'].mean(), 3)
    rocauc_score_train = round(roc_auc_score(y_train,
y_train_pred_prob[:, 1]), 3)
    csr_score_train = round(cv_score_rocauc['train_score'].mean(), 3)

    accuracy_test = round(accuracy_score(y_test, y_test_pred), 3)
    precision_test = round(precision_score(y_test, y_test_pred), 3)
    recall_test = round(recall_score(y_test, y_test_pred), 3)
    fbeta_s_test = round(fbeta_score(y_test, y_test_pred, beta=0.5),
3)
    f1_s_test = round(f1_score(y_test, y_test_pred), 3)
    csf_score_test = round(cv_score_f1['test_score'].mean(), 3)
    rocauc_score_test = round(roc_auc_score(y_test,
y_test_pred_prob[:, 1]), 3)
    csr_score_test = round(cv_score_rocauc['test_score'].mean(), 3)

    if save :

        # save report detail train
        train_classifier_list.append(model)
        train_modelname_list.append(model_name)
        train_accuracy_list.append(accuracy_train)
        train_precision_list.append(precision_train)
        train_recall_list.append(recall_train)
        train_fbeta_score_list.append(fbeta_s_train)
        train_f1_score_list.append(f1_s_train)
        train_cross_val_f1_list.append(csf_score_train)
        train_roc_auc_score_list.append(rocauc_score_train)
        train_cross_val_rocauc_list.append(csr_score_train)

        # save report detail test
        test_classifier_list.append(model)
        test_modelname_list.append(model_name)
        test_accuracy_list.append(accuracy_test)
        test_precision_list.append(precision_test)
        test_recall_list.append(recall_test)
        test_fbeta_score_list.append(fbeta_s_test)
        test_f1_score_list.append(f1_s_test)
        test_cross_val_f1_list.append(csf_score_test)
        test_roc_auc_score_list.append(rocauc_score_test)
```

```
        test_cross_val_rocauc_list.append(csr_score_test)


    metrics_summary = pd.DataFrame({
        'Evaluation Metrics' : ["Accuracy", "Precision", "Recall",
"F0.5 Score", "F1 Score", "F1 Score (crossval)", "ROC AUC", "ROC AUC
(crossval)"],
        'Train' : [accuracy_train, precision_train, recall_train,
fbeta_s_train, f1_s_train, csf_score_train, rocauc_score_train,
csr_score_train],
        'Test' : [accuracy_test, precision_test, recall_test,
fbeta_s_test, f1_s_test, csf_score_test, rocauc_score_test,
csr_score_test]})

    metrics_summary["Diff Range"] = metrics_summary['Train'] -
metrics_summary['Test']
    return metrics_summary.reset_index(drop =
True).style.format(precision=3).background_gradient(cmap='Purples')

# define function to see the best tuning hyperparameter
def show_best_hyperparameter(model):
    print(model.best_estimator_.get_params())
```

- Function Evaluation for Training

```
def model_eval_train(classifier, model_name, X_train, y_train):
    # predict data train
    y_train_pred = classifier.predict(X_train)
    y_train_pred_prob = classifier.predict_proba(X_train)

    # print classification report
    print('Classification Report Training Model ('+model_name+'):\n')
    accuracy = round(accuracy_score(y_train, y_train_pred), 3)
    precision = round(precision_score(y_train, y_train_pred), 3)
    recall = round(recall_score(y_train, y_train_pred), 3)
    fbeta_s = round(fbeta_score(y_train, y_train_pred, beta=0.5), 3)
    f1_s = round(f1_score(y_train, y_train_pred), 3)
    rocauc_score = round(roc_auc_score(y_train, y_train_pred_prob[:,
1]), 3)

    # c_val_score = round(cross_val_score(classifier, X_s, y_s, cv=kf,
scoring='roc_auc').mean()  , 3)
    cv_score_f1 = cross_validate(classifier, X_s, y_s, cv=kf,
scoring='f1', return_train_score=True)
    csf_score = round(cv_score_f1['train_score'].mean(), 3)

    cv_score_rocauc = cross_validate(classifier, X_s, y_s, cv=kf,
scoring='roc_auc', return_train_score=True)
    csr_score = round(cv_score_rocauc['train_score'].mean(), 3)
```

```
    print(f'Accuracy = {accuracy}')
    print(f'Precision = {precision}')
    print(f'Recall = {recall}')
    print(f'F0.5 Score = {fbeta_s}')
    print(f'F1 Score = {f1_s}')
    print(f'Cross Val F1 (k=5) = {csf_score}')
    print(f'ROC AUC = {rocauc_score}')
    print(f'Cross Val ROC AUC (k=5) = {csr_score}\n')

    print(classification_report(y_train, y_train_pred))

    # form confusion matrix as a dataFrame
    conf_matrix = pd.DataFrame((confusion_matrix(y_train,
y_train_pred)), ('No Response', 'Response'), ('No Response',
'Response'))
    tn, fp, fn, tp = confusion_matrix(y_train, y_train_pred).ravel()

    print("==== Actual Data (Train) =====")
    print("Total =", len(y_train))
    print("No Response =", len(y_train[y_train == 0]))
    print("Response =", len(y_train[y_train == 1]))
    print("==== Predicted Data (Train) =====")
    print("TP = {}, FP = {}, TN = {}, FN = {}".format(tp, fp, tn, fn))
    print("Predictly Correct =", tn+tp)
    print("Predictly Wrong =", fn+fp, "\n")

    # plot confusion matrix
    plt.figure(figsize=[8,5])

    c_matrix = confusion_matrix(y_train, y_train_pred)
    names = ['True Negative', 'False Positive', 'False Negative',
'True Positive']
    counts = ['{0:0.0f}'.format(value) for value in
c_matrix.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in
c_matrix.flatten() / np.sum(c_matrix)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(names, counts,
percentages)]
    labels = np.asarray(labels).reshape(2, 2)

    heatmap = sns.heatmap(conf_matrix, annot = labels,
annot_kws={'size': 13}, fmt='', cmap='Greens')
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(),
rotation=0, ha='right', fontsize=13)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(),
rotation=0, ha='right', fontsize=13)

    plt.title('Confusion Matrix for Training Model ('+model_name+')\
n', fontsize=13, color='black')
    plt.ylabel('Actual Label', fontsize=13)
```

```
    plt.xlabel('\nPredicted Label', fontsize=13)
    plt.show()
    print("\n")

    # ROC AUC Curve
    plt.figure(figsize=[8,5])
    fpr, tpr, threshold = roc_curve(y_train, y_train_pred_prob[:, 1])
    plt.plot(fpr, tpr, label = model_name+' (Area (Score) =
%0.2f)'%rocauc_score)
    plt.plot([0,1],[0,1],'r--')
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Credit Card Approval')
    plt.legend(loc="lower right")
    plt.show()
```

- Function Evaluation for Testing

```
def model_eval_test(classifier, model_name, X_test, y_test):

     # predict data test
    y_test_pred = classifier.predict(X_test)
    y_test_pred_prob = classifier.predict_proba(X_test)


    # print classification report
    print('Classification Report Testing Model ('+model_name+'):\n')
    accuracy = round(accuracy_score(y_test, y_test_pred), 3)
    precision = round(precision_score(y_test, y_test_pred), 3)
    recall = round(recall_score(y_test, y_test_pred), 3)
    fbeta_s = round(fbeta_score(y_test, y_test_pred, beta=0.5), 3)
    f1_s = round(f1_score(y_test, y_test_pred), 3)
    rocauc_score = round(roc_auc_score(y_test, y_test_pred_prob[:,
1]), 3)

    # c_val_score = round(cross_val_score(classifier, X_s, y_s , cv=kf
, scoring='roc_auc').mean()  , 3)
    cv_score_f1 = cross_validate(classifier, X_s, y_s, cv=kf,
scoring='f1', return_train_score=True)
    csf_score = round(cv_score_f1['test_score'].mean(), 3)

    cv_score_rocauc = cross_validate(classifier, X_s, y_s, cv=kf,
scoring='roc_auc', return_train_score=True)
    csr_score = round(cv_score_rocauc['test_score'].mean(), 3)

    print(f'Accuracy = {accuracy}')
    print(f'Precision = {precision}')
    print(f'Recall = {recall}')
```

```
    print(f'F0.5 Score = {fbeta_s}')
    print(f'F1 Score = {f1_s}')
    print(f'Cross Val F1 (k=5) = {csf_score}')
    print(f'ROC AUC = {rocauc_score}')
    print(f'Cross Val ROC AUC (k=5) = {csr_score}\n')

    print(classification_report(y_test, y_test_pred))

    # form confusion matrix as a dataFrame
    conf_matrix = pd.DataFrame((confusion_matrix(y_test,
y_test_pred)), ('No Response', 'Response'), ('No Response',
'Response'))
    tn, fp, fn, tp = confusion_matrix(y_test, y_test_pred).ravel()

    print("==== Actual Data (Test) =====")
    print("Total =", len(y_test))
    print("No Response =", len(y_test[y_test == 0]))
    print("Response =", len(y_test[y_test == 1]))
    print("==== Predicted Data (Test) =====")
    print("TP = {}, FP = {}, TN = {}, FN = {}".format(tp, fp, tn, fn))
    print("Predictly Correct =", tn+tp)
    print("Predictly Wrong =", fn+fp, "\n")

    # plot confusion matrix
    plt.figure(figsize=[8,5])

    c_matrix = confusion_matrix(y_test, y_test_pred)
    names = ['True Negative', 'False Positive', 'False Negative',
'True Positive']
    counts = ['{0:0.0f}'.format(value) for value in
c_matrix.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in
c_matrix.flatten() / np.sum(c_matrix)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(names, counts,
percentages)]
    labels = np.asarray(labels).reshape(2, 2)

    heatmap = sns.heatmap(conf_matrix, annot = labels,
annot_kws={'size': 13}, fmt='', cmap='Oranges')
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(),
rotation=0, ha='right', fontsize=13)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(),
rotation=0, ha='right', fontsize=13)

    plt.title('Confusion Matrix for Testing Model ('+model_name+')\n',
fontsize=13, color='black')
    plt.ylabel('Actual Label', fontsize=13)
    plt.xlabel('\nPredicted Label', fontsize=13)
    plt.show()
    print("\n")
```

```
    # ROC AUC Curve
    plt.figure(figsize=[8,5])
    fpr, tpr, threshold = roc_curve(y_test, y_test_pred_prob[:, 1])
    plt.plot(fpr, tpr, label = model_name+' (Area (Score) =
%0.2f)'%rocauc_score)
    plt.plot([0,1],[0,1],'r--')
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Credit Card Approval')
    plt.legend(loc="lower right")
    plt.show()
```

- Function Plotting Feature Importance

```
def feature_importance_plot(classifier, model_name, X_train=None):

    model_name = re.sub('\s*\(\w*\)\s*', '', model_name)

    # important features
    # K-Nearest Neighbors/MLP Classifier = No Function/Method to Check
Feature Importance
    if model_name == "Logistic Regression":
        # Logistic Regression
        imp = pd.DataFrame(data={
            'Attribute': X_train.columns,
            'Importance': log_model.coef_[0]
        })
        imp = imp.sort_values(by='Importance',
ascending=False).set_index("Attribute")
        ft_imp = imp["Importance"]
    elif model_name in ["Naive Bayes", "K-Nearest Neighbors", "MLP
Classifier", "Support Vector Machine"]:
        imp = pd.DataFrame(data={
            'Attribute': X_train.columns,
            'Importance': permutation_importance(classifier, X_test,
y_test).importances_mean
        })
        imp = imp.sort_values(by='Importance',
ascending=False).set_index("Attribute")
        ft_imp = imp["Importance"]
    else:
        # Decision Tree
        # Random Forest
        # XGBoost Classifier
        # Gradient Boosting Classifier
        ft_imp = classifier.feature_importances_
```

```
    importances = pd.Series(ft_imp,
index=X.columns).sort_values(ascending=False)

    plt.figure(figsize=(10,7))
    fig = importances.plot(kind ='barh', color='#7faadb', width=0.8)
    plt.title('Features Importance Plot '+model_name+'\n',
fontsize=14)

    # Annotate every single Bar with its value, based on it's width
    N = len(importances)
    thickness = range(1, N+1)
    thick_sum = sum(thickness)
    barmax = max(importances)
    for i, p in enumerate(fig.patches):
            fig.annotate("%.2f" % (p.get_width()),
                         (p.get_x() + p.get_width(), p.get_y() + 1),
                         xytext=(5, 15), textcoords='offset points')
            fig.set_ylim(N, -1)
            fig.set_xlim(0,barmax*1.4)

    fig.figure.tight_layout()
    plt.show()
```

- Function Plotting SHAP

```
def shap_plot(model, model_name, data=None):

    model_name = re.sub('\s*\(\w*\)\s*', '', model_name)

    tree = ["Decision Tree", "Random Forest",
            "XGBoost Classifier", "Gradient Boosting Classifier"]

    shap.initjs()

    if model_name in tree:

        if model_name in ["XGBoost Classifier", "Gradient Boosting
Classifier"] :
            explainer = shap.TreeExplainer(model)
            shap_values = explainer.shap_values(data)
            expected = explainer.expected_value
        else:
            explainer = shap.TreeExplainer(model)
            shap_values = explainer.shap_values(data)[1]
            expected = explainer.expected_value[1]


        fig = plt.figure()
        ax0 = fig.add_subplot(121)
        shap.summary_plot(shap_values, data, show = False)
```

```
        ax1 = fig.add_subplot(122)
        shap.summary_plot(shap_values, data, plot_type='bar', show =
False, cmap = "plasma")
        plt.gcf().set_size_inches(20,8)
        plt.tight_layout()
        plt.show()

        return shap.force_plot(expected, shap_values[0],
data.iloc[0,:], plot_cmap = "PkYg", show = False)

    else :
        explainer = shap.Explainer(model.predict, data)
        shap_values = explainer(data)

        fig = plt.figure()
        ax0 = fig.add_subplot(121)
        shap.summary_plot(shap_values.values, data, show = False)
        ax1 = fig.add_subplot(122)
        shap.summary_plot(shap_values.values, data, plot_type='bar',
show = False, cmap = "plasma")
        plt.gcf().set_size_inches(20,8)
        plt.tight_layout()
        plt.show()

        return shap.force_plot(shap_values.base_values[0],
shap_values.values[0], data.iloc[0,:], plot_cmap = "PkYg", show =
False)
```

Modelling

Pada tahap awal modelling ini, kami menggunakan beberapa algoritma Machine Learning. Harapannya agar output model comparison yang dihasilkan lebih kaya dan decision yang dipilih lebih akurat.

1. Decision Tree

Decision tree membagi data berdasarkan serangkaian keputusan yang dibuat pada fitur-fitur. Setiap node internal pada pohon keputusan mewakili keputusan berdasarkan fitur-fitur tersebut, sedangkan daun pohon mewakili label atau kelas hasil klasifikasi.

```
# train the model
dt_model =
DecisionTreeClassifier(random_state=42).fit(X_train,y_train)
print(dt_model)
eval_classification(dt_model, "Decision Tree")
```

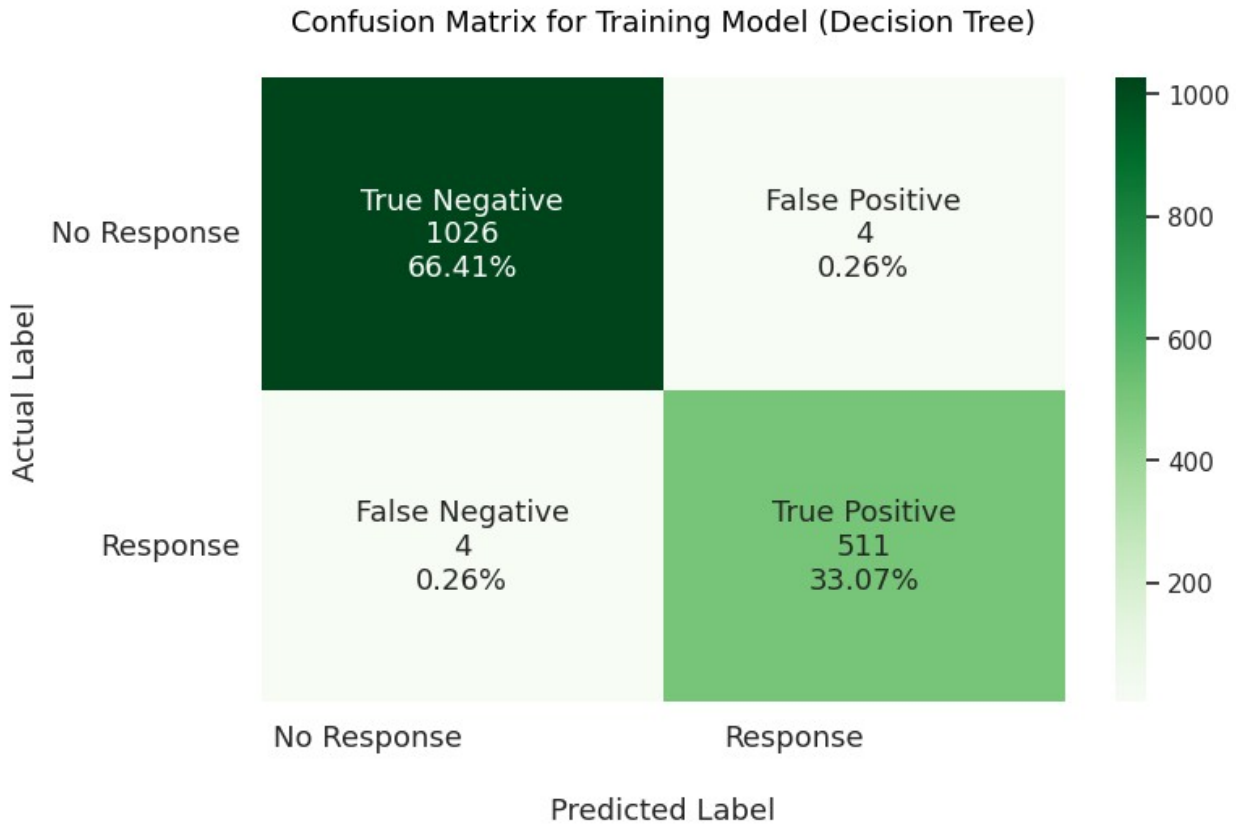
```
DecisionTreeClassifier(random_state=42)
```

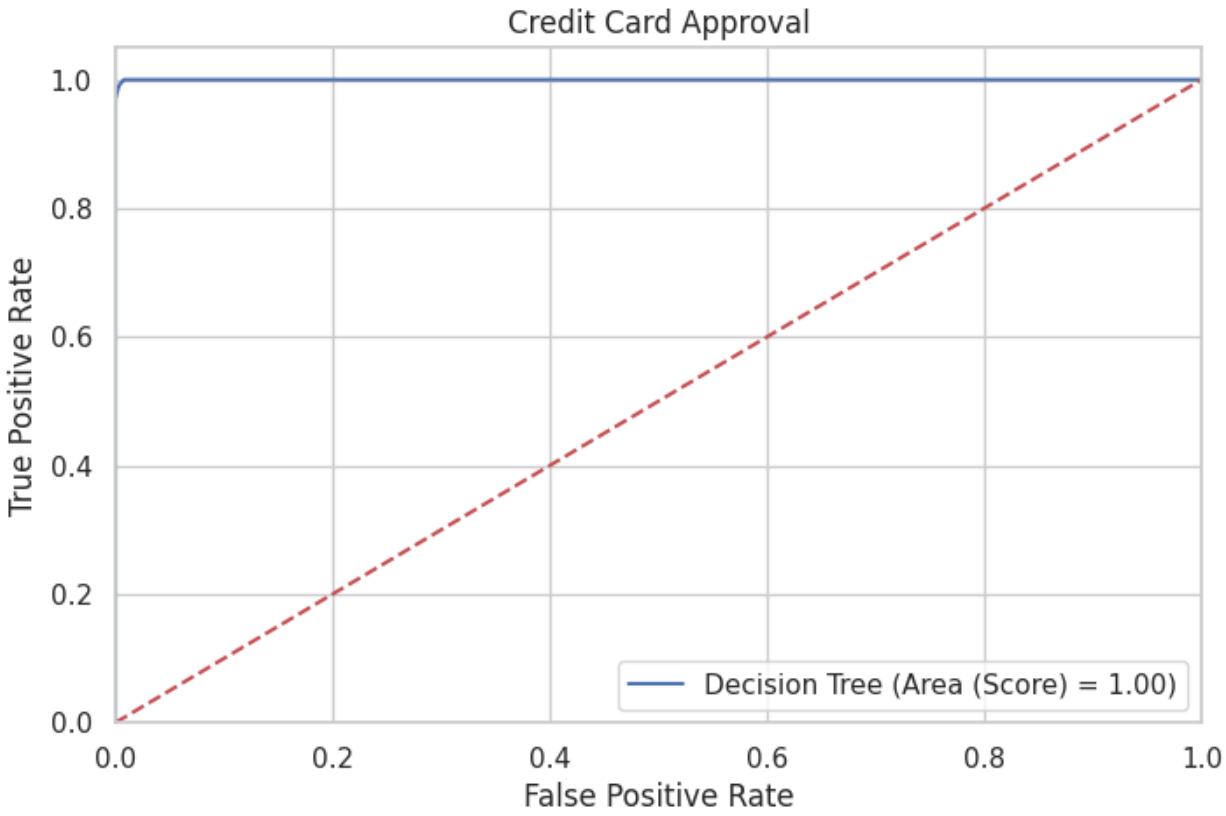
```
<pandas.io.formats.style.Styler at 0x787302d0d180>
```

Performance of Training Model

```
model_eval_train(dt_model, "Decision Tree", X_train, y_train)
```

```
Classification Report Training Model (Decision Tree):

Accuracy = 0.995
Precision = 0.992
Recall = 0.992
F0.5 Score = 0.992
F1 Score = 0.992
Cross Val F1 (k=5) = 0.978
ROC AUC = 1.0
Cross Val ROC AUC (k=5) = 1.0

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1030
           1       0.99      0.99      0.99       515

    accuracy                           0.99      1545
   macro avg       0.99      0.99      0.99      1545
weighted avg       0.99      0.99      0.99      1545

==== Actual Data (Train) =====
Total = 1545
No Response = 1030
Response = 515
==== Predicted Data (Train) =====
TP = 511, FP = 4, TN = 1026, FN = 4
Predictly Correct = 1537
Predictly Wrong = 8
```


Confusion Matrix for Training Model (Decision Tree)
True Negative
1026
66.41%
False Positive
4
0.26%
False Negative
4
0.26%
True Positive
511
33.07%
No Response
Response
Actual Label
No Response
Response
Predicted Label
1000
800
600
400
200

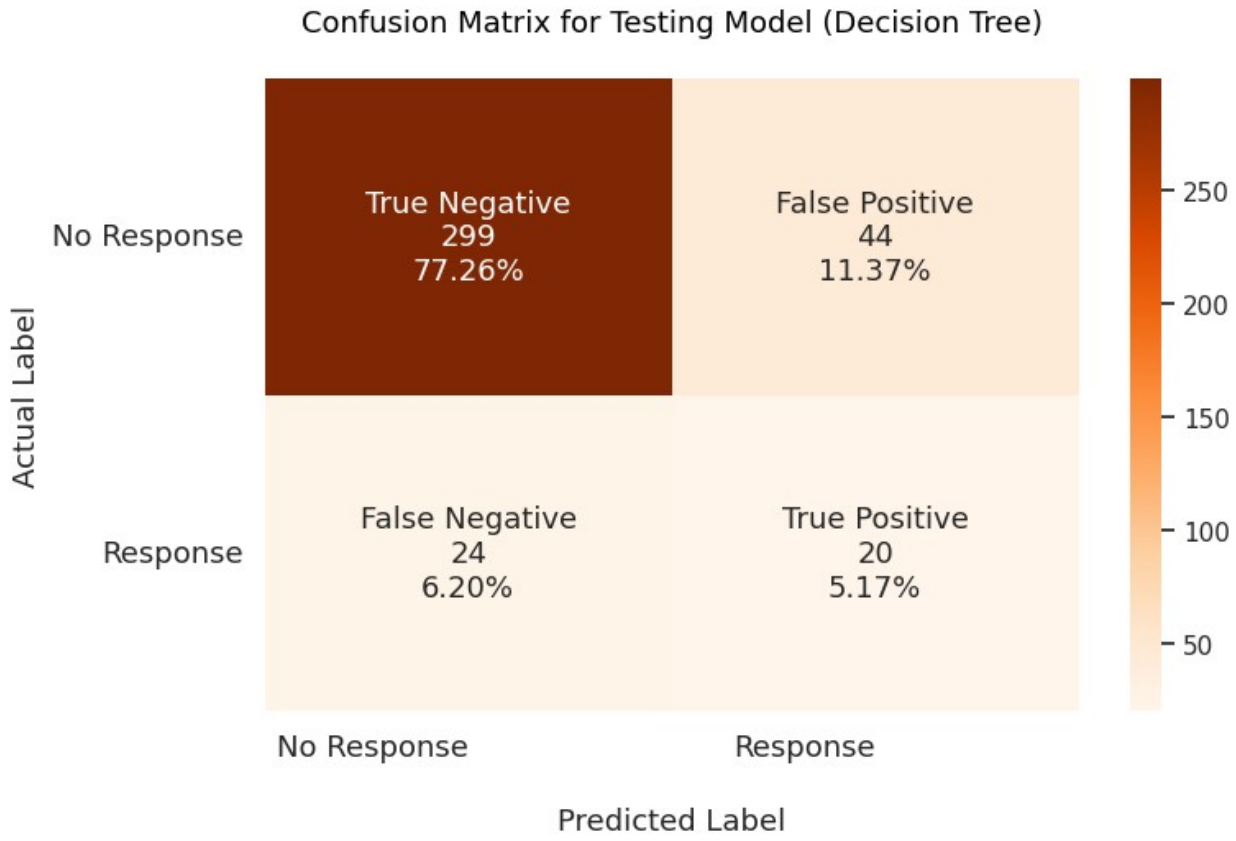


Performance of Testing Model

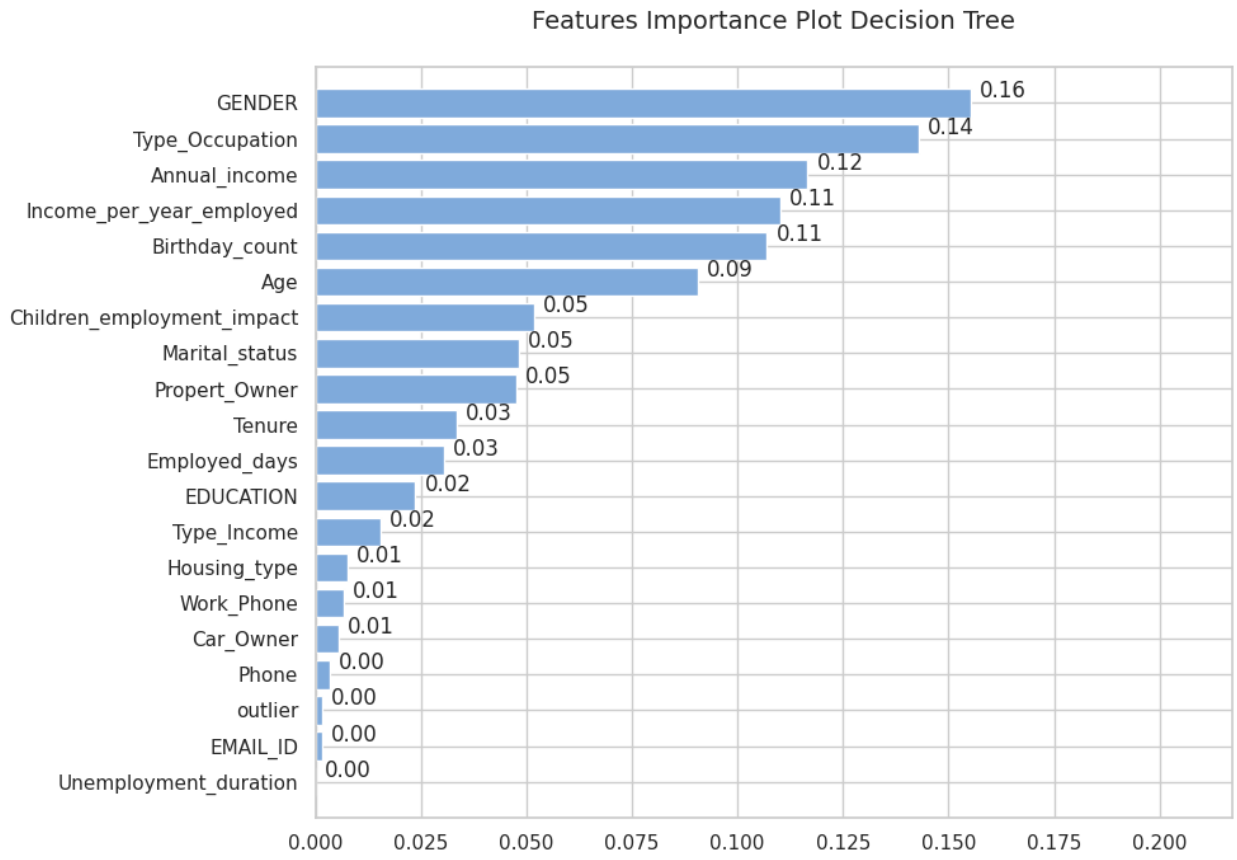
```
model_eval_test(dt_model, "Decision Tree", X_test, y_test)
```

```
Classification Report Testing Model (Decision Tree):

Accuracy = 0.824
Precision = 0.312
Recall = 0.455
F0.5 Score = 0.333
F1 Score = 0.37
Cross Val F1 (k=5) = 0.395
ROC AUC = 0.671
Cross Val ROC AUC (k=5) = 0.684

              precision    recall  f1-score   support

           0       0.93      0.87      0.90       343
           1       0.31      0.45      0.37        44

    accuracy                           0.82       387
   macro avg       0.62      0.66      0.63       387
weighted avg       0.86      0.82      0.84       387

==== Actual Data (Test) =====
```

```
Total = 387
No Response = 343
Response = 44
==== Predicted Data (Test) =====
TP = 20, FP = 44, TN = 299, FN = 24
Predictly Correct = 319
Predictly Wrong = 68
```

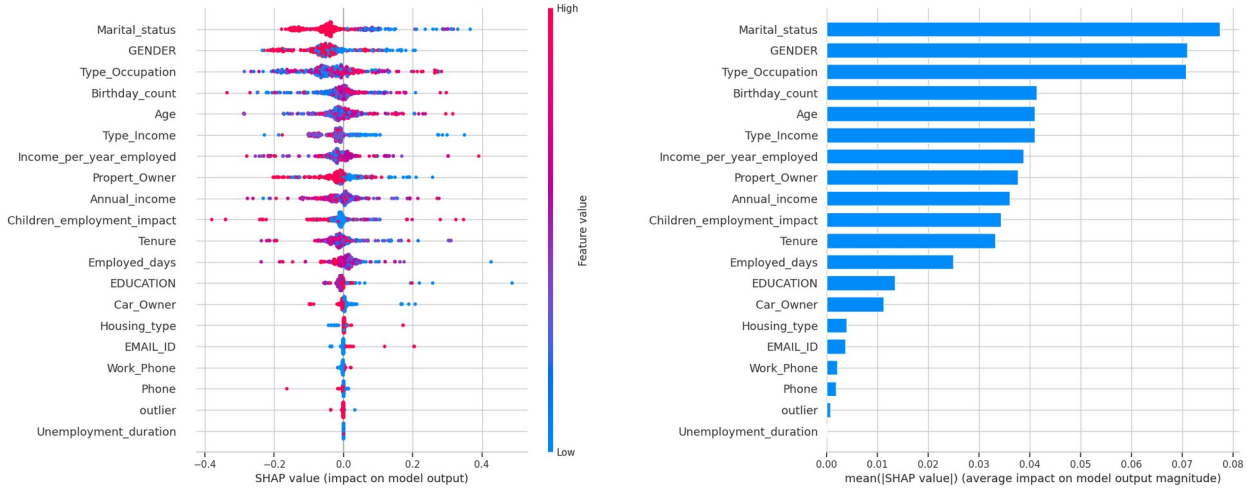



```
acc_dt_train=round(dt_model.score(X_train,y_train)*100,2)
acc_dt_test=round(dt_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_dt_train))
print("Testing Accuracy: {} %".format(acc_dt_test))

Training Accuracy: 99.48 %
Testing Accuracy: 82.43 %

feature_importance_plot(dt_model, "Decision Tree")
```




```
shap_plot(dt_model, "Decision Tree", X_test)
```

```
<IPython.core.display.HTML object>
```




```
<shap.plots._force.AdditiveForceVisualizer at 0x78730266f1c0>
```

Display the Tree

Preparation

- pip install graphviz
- conda install graphviz or

You need to run

`conda install python-graphviz`

instead of

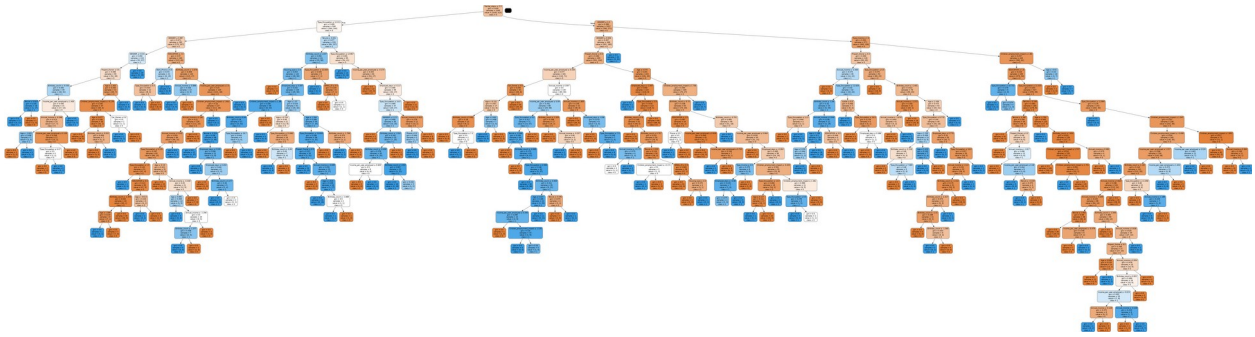
`pip install graphviz`

to get these bindings, which also work with conda's Graphviz package.

or

- Download and install graphviz-2.38.msi (use the newest version) from https://graphviz.gitlab.io/_pages/Download/Download_windows.html
- Set the path variable
  - Control Panel > System and Security > System > Advanced System Settings > Environment Variables > Path > Edit
  - add 'C:\Program Files (x86)\Graphviz2.38\bin'

```
%%capture
!pip install pydotplus

from sklearn.tree import export_graphviz
from six import StringIO
# from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(dt_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,
                feature_names = list(X_test.columns),
                class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('assets\decision-tree-graph.png')
Image(graph.create_png())
```



2. Random Forest

Metode ini menggabungkan prediksi dari beberapa decision tree untuk menghasilkan prediksi akhir yang lebih stabil dan akurat.

```
# train the model
rf_model = RandomForestClassifier(random_state=2).fit(X_train,
y_train)
print(rf_model)
eval_classification(rf_model, "Random Forest")
```

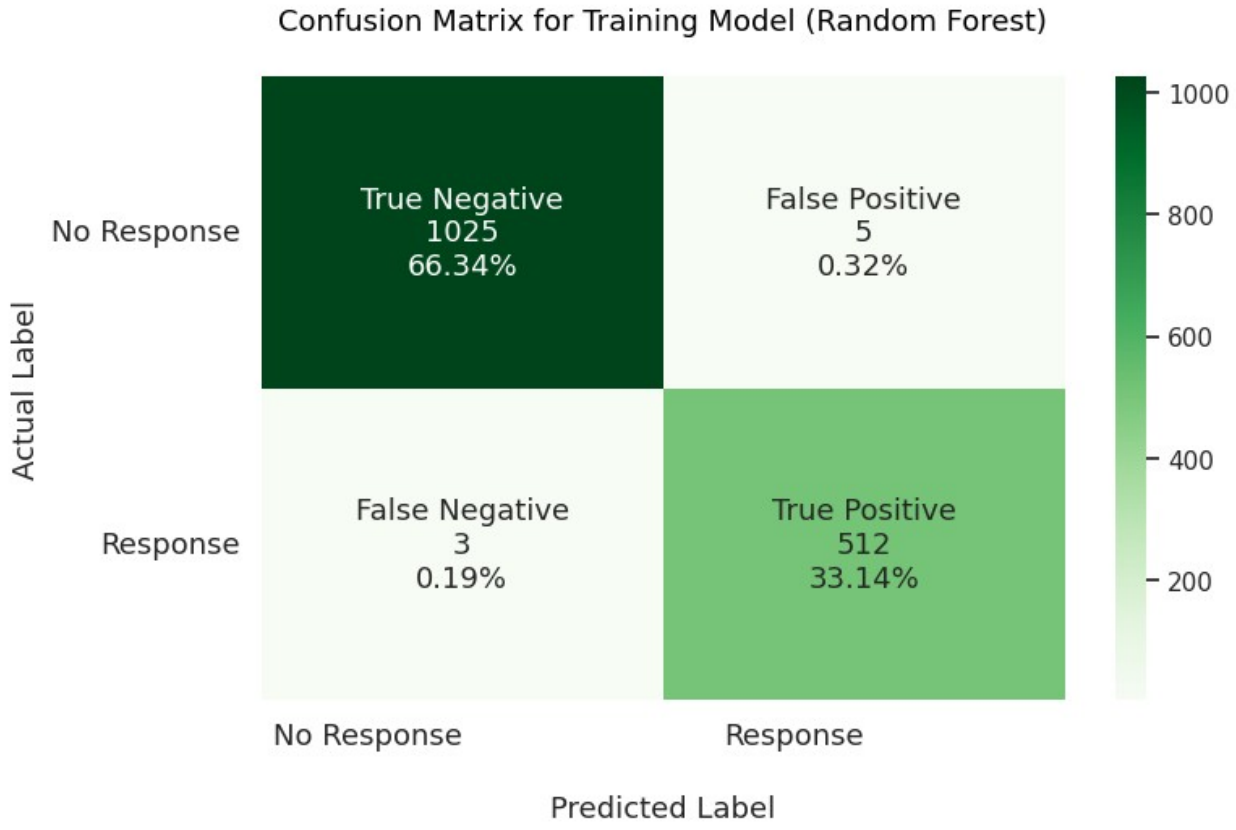
```
RandomForestClassifier(random_state=2)
```

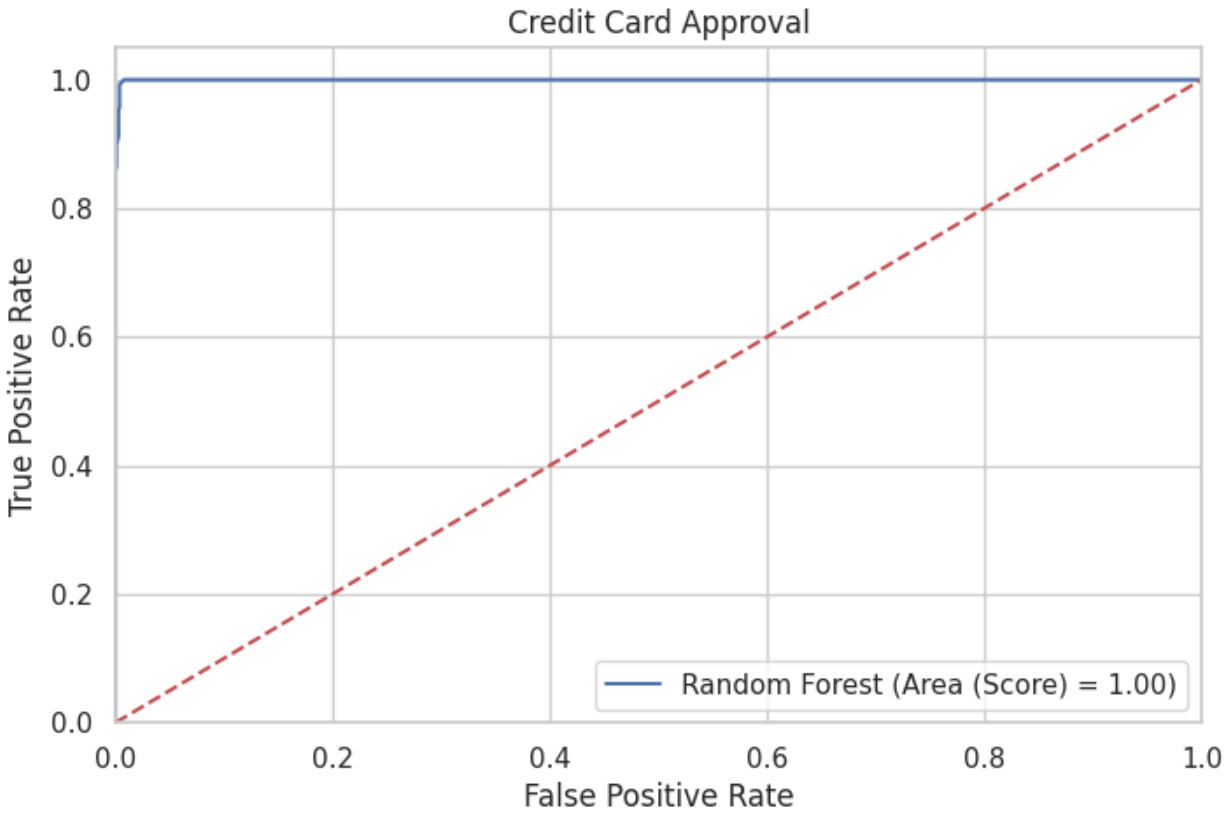
```
<pandas.io.formats.style.Styler at 0x787300d58580>
```

```
model_eval_train(rf_model, "Random Forest", X_train, y_train)
```

```
Classification Report Training Model (Random Forest):

Accuracy = 0.995
Precision = 0.99
Recall = 0.994
F0.5 Score = 0.991
F1 Score = 0.992
Cross Val F1 (k=5) = 0.979
ROC AUC = 1.0
Cross Val ROC AUC (k=5) = 0.999

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1030
           1       0.99      0.99      0.99       515

    accuracy                           0.99      1545
   macro avg       0.99      0.99      0.99      1545
weighted avg       0.99      0.99      0.99      1545

==== Actual Data (Train) =====
Total = 1545
No Response = 1030
```

```
Response = 515
==== Predicted Data (Train) =====
TP = 512, FP = 5, TN = 1025, FN = 3
Predictly Correct = 1537
Predictly Wrong = 8
```

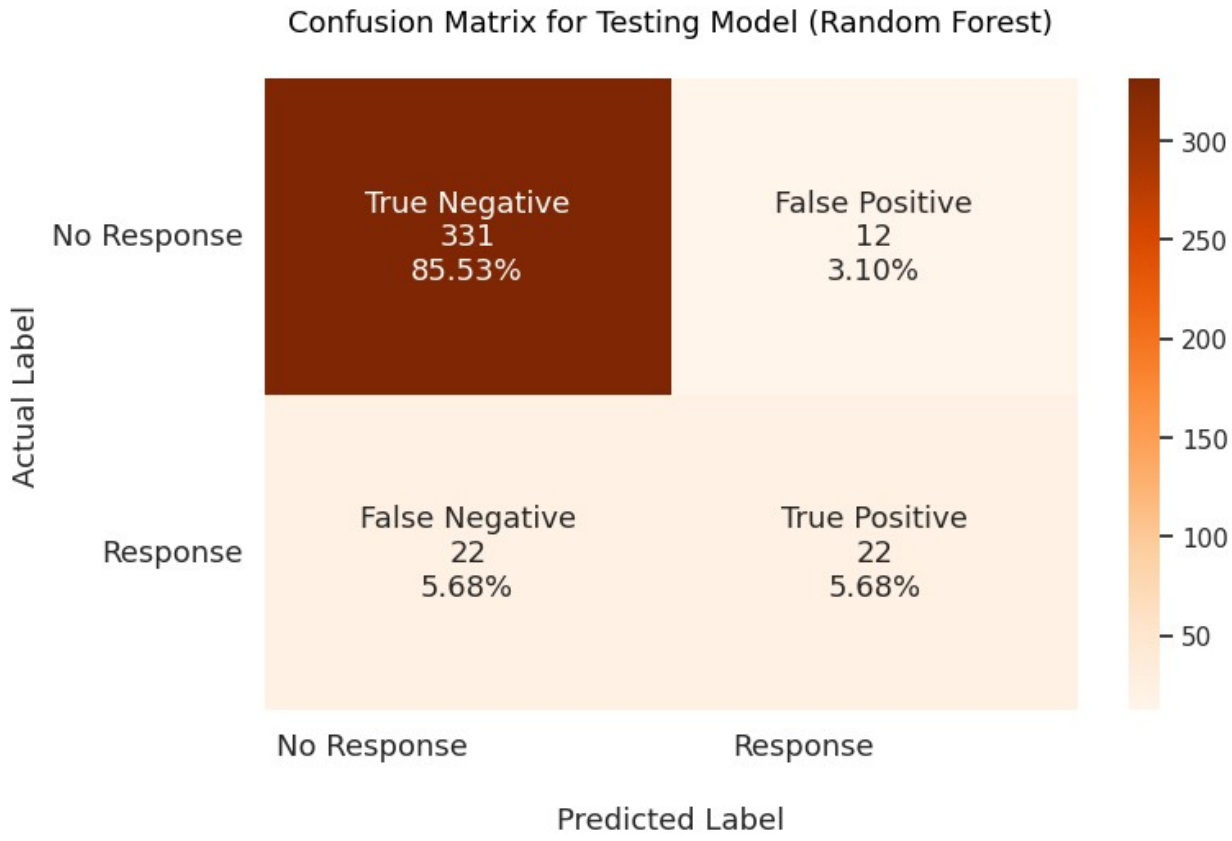


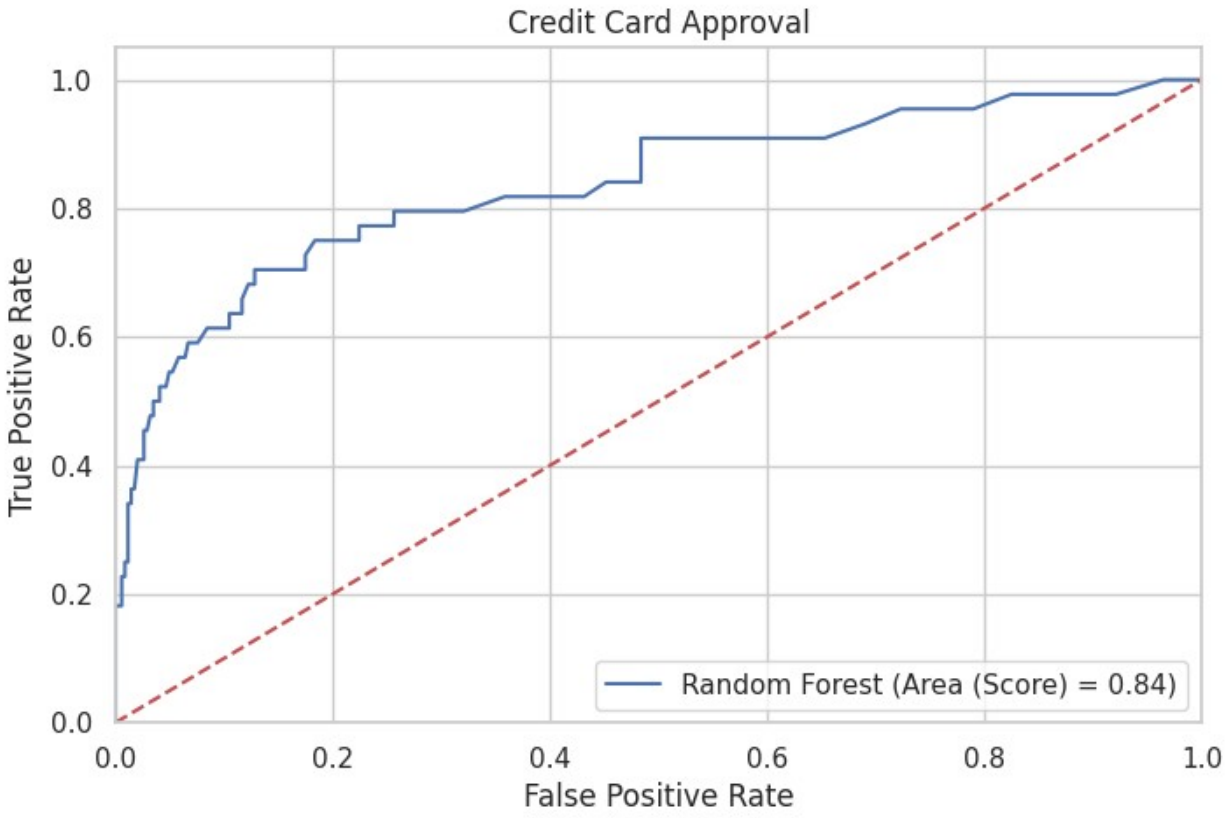


Performance of Testing Model

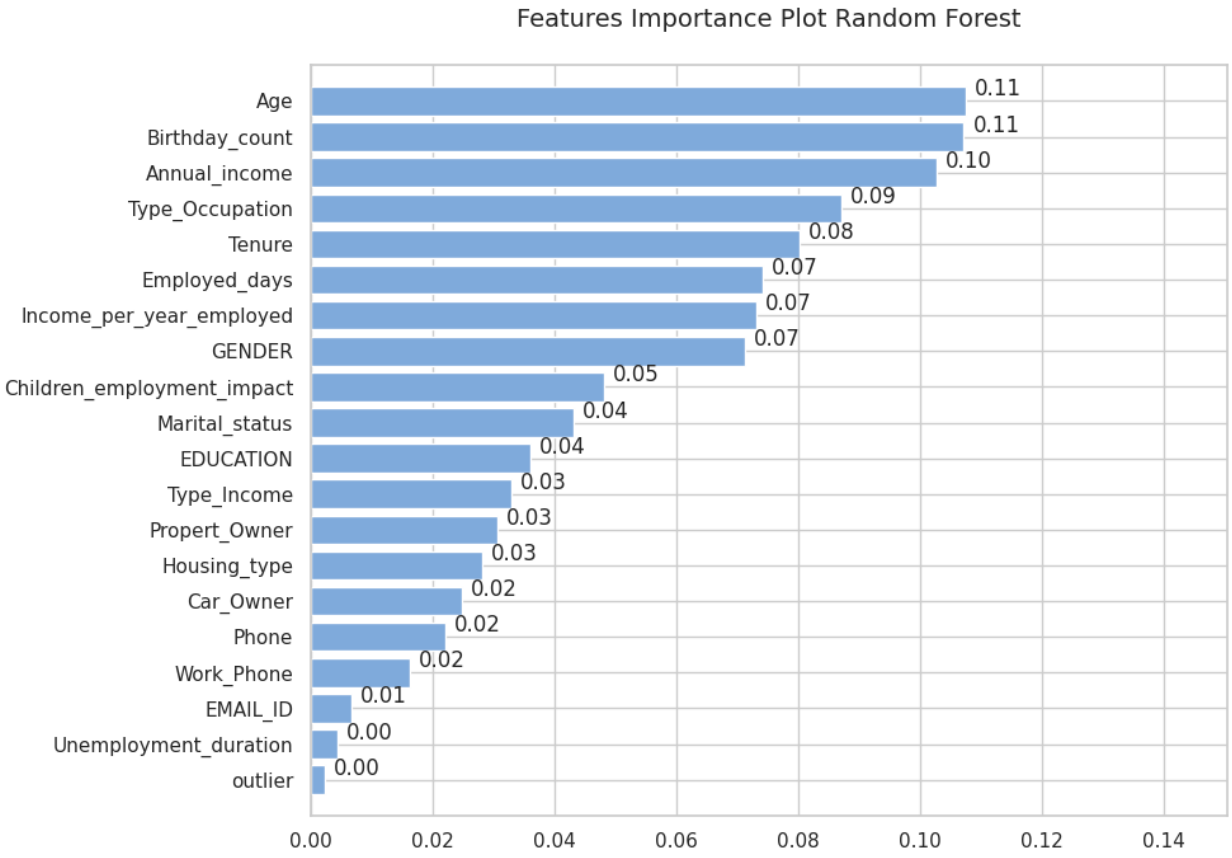
```
model_eval_test(rf_model, "Random Forest", X_test, y_test)

Classification Report Testing Model (Random Forest):

Accuracy = 0.912
Precision = 0.647
Recall = 0.5
F0.5 Score = 0.611
F1 Score = 0.564
Cross Val F1 (k=5) = 0.51
ROC AUC = 0.836
Cross Val ROC AUC (k=5) = 0.805

              precision    recall  f1-score   support

           0       0.94      0.97      0.95       343
           1       0.65      0.50      0.56        44

    accuracy                           0.91       387
   macro avg       0.79      0.73      0.76       387
weighted avg       0.90      0.91      0.91       387

==== Actual Data (Test) =====
```

```
Total = 387
No Response = 343
Response = 44
==== Predicted Data (Test) =====
TP = 22, FP = 12, TN = 331, FN = 22
Predictly Correct = 353
Predictly Wrong = 34
```



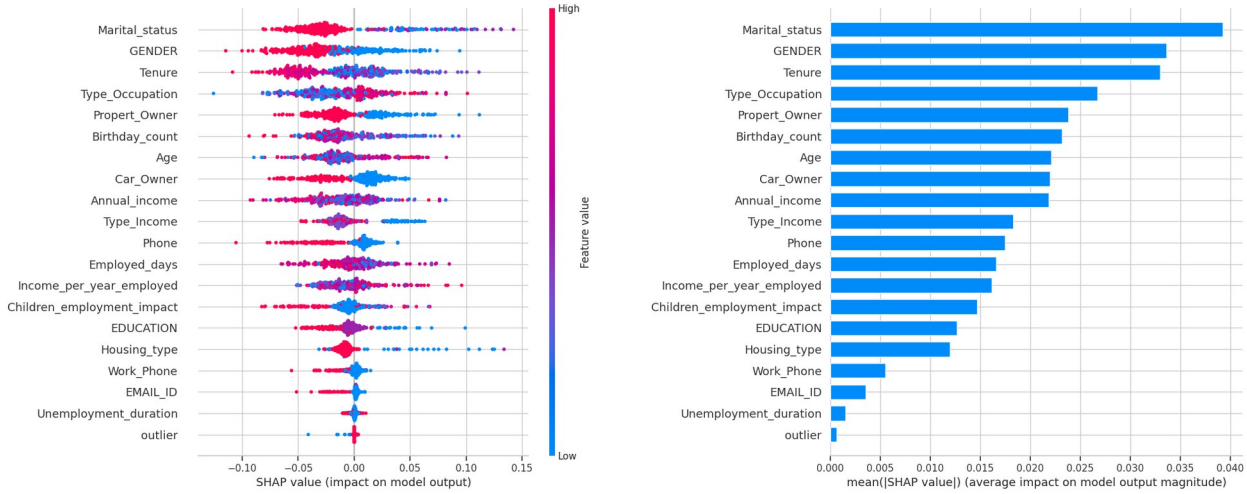



```
acc_rf_train=round(rf_model.score(X_train,y_train)*100,2)
acc_rf_test=round(rf_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_rf_train))
print("Test Accuracy: {} %".format(acc_rf_test))

Training Accuracy: 99.48 %
Test Accuracy: 91.21 %

feature_importance_plot(rf_model, "Random Forest")
```




```
shap_plot(rf_model, "Random Forest", X_test)
```

```
<IPython.core.display.HTML object>
```

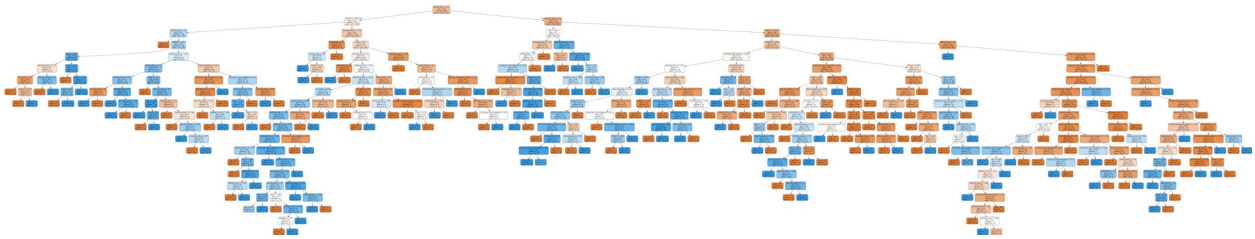



```
<shap.plots._force.AdditiveForceVisualizer at 0x787301a58c10>
```

Display the Tree

```
from sklearn.tree import export_graphviz

estimator = rf_model.estimators_[0]

export_graphviz(estimator, rounded=True, proportion=False,
out_file='assets\\random-forest-tree.dot',
                feature_names=X_test.columns, precision=2,
filled=True)

from subprocess import call
call(['dot', '-Tpng', 'assets\\random-forest-tree.dot', '-o',
'assets\\random-forest-tree.png', '-Gdpi=600'])

from IPython.display import Image
Image(filename = 'assets\\random-forest-tree.png')
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by
0.326095 to fit
```



3. Logistic Regression

Metode ini digunakan untuk memodelkan hubungan antara variabel independen dan variabel dependen biner atau multi-kategori. Regresi logistik menghasilkan probabilitas kelas sebagai output.

```
# train the model
log_model = LogisticRegression(solver='lbfgs', max_iter=len(X_train),
random_state=42).fit(X_train, y_train)
print(log_model)
eval_classification(log_model, "Logistic Regression")
```

```
LogisticRegression(max_iter=1545, random_state=42)
```

```
<pandas.io.formats.style.Styler at 0x787301a8dd20>
```

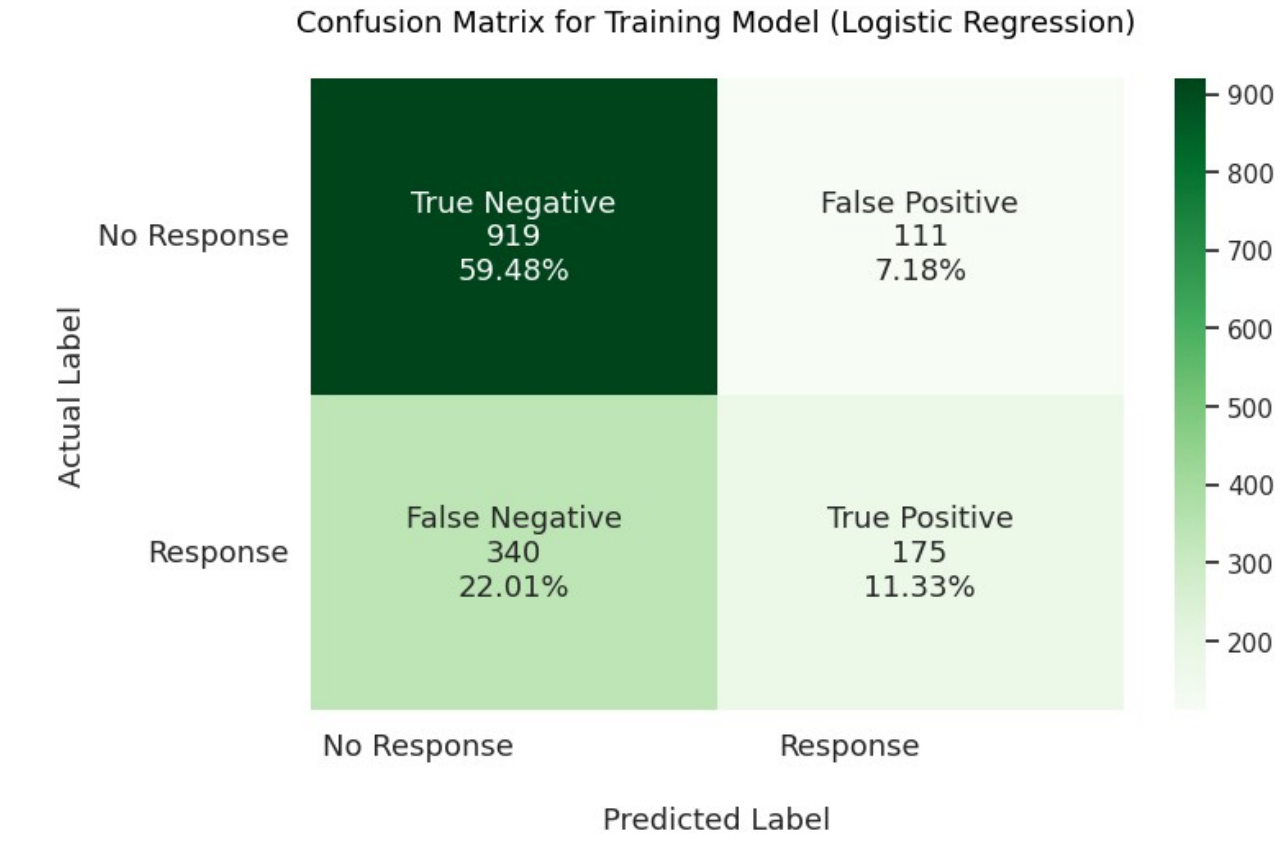
Performance of Training Model

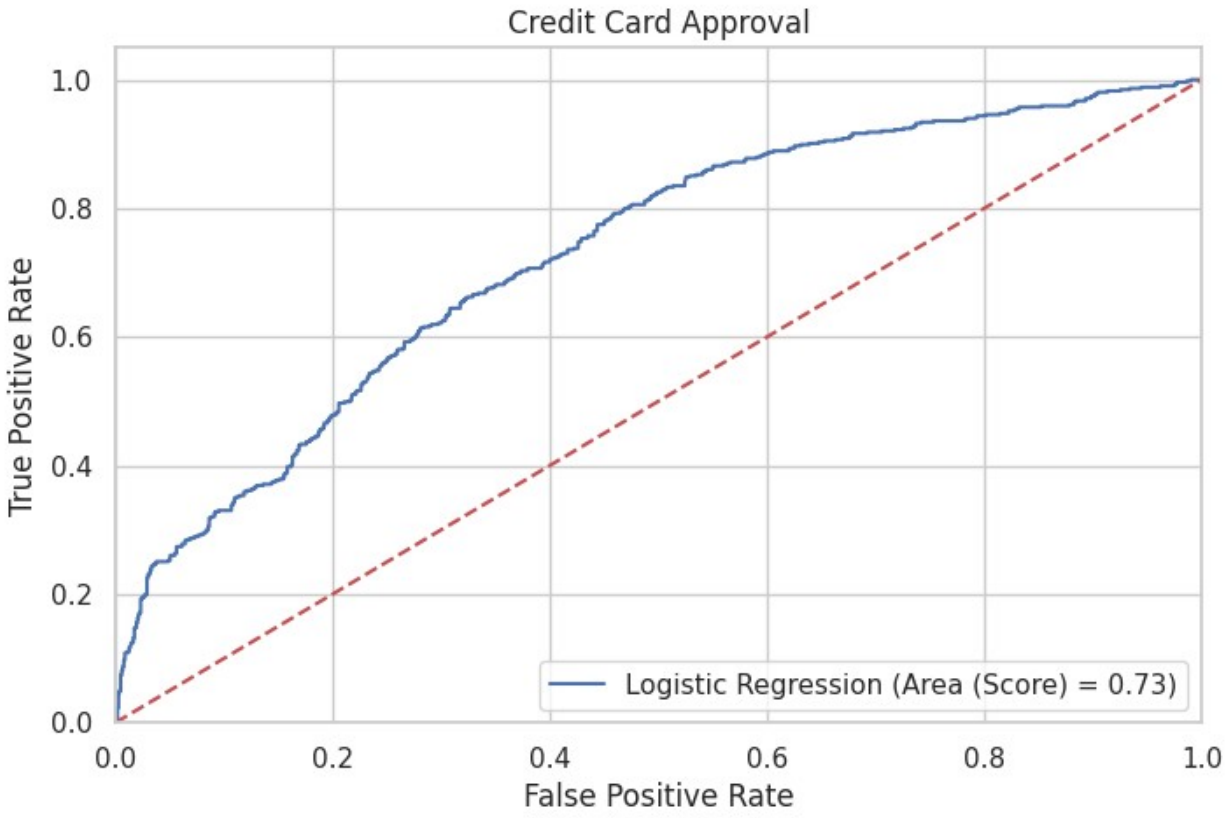
```
model_eval_train(log_model, "Logistic Regression", X_train, y_train)
```

```
Classification Report Training Model (Logistic Regression):
```

```
Accuracy = 0.708
```

```
Precision = 0.612
Recall = 0.34
F0.5 Score = 0.527
F1 Score = 0.437
Cross Val F1 (k=5) = 0.001
ROC AUC = 0.728
Cross Val ROC AUC (k=5) = 0.64

              precision    recall  f1-score   support

           0       0.73      0.89      0.80      1030
           1       0.61      0.34      0.44       515

    accuracy                           0.71      1545
   macro avg       0.67      0.62      0.62      1545
weighted avg       0.69      0.71      0.68      1545

==== Actual Data (Train) =====
Total = 1545
No Response = 1030
Response = 515
==== Predicted Data (Train) =====
TP = 175, FP = 111, TN = 919, FN = 340
Predictly Correct = 1094
Predictly Wrong = 451
```

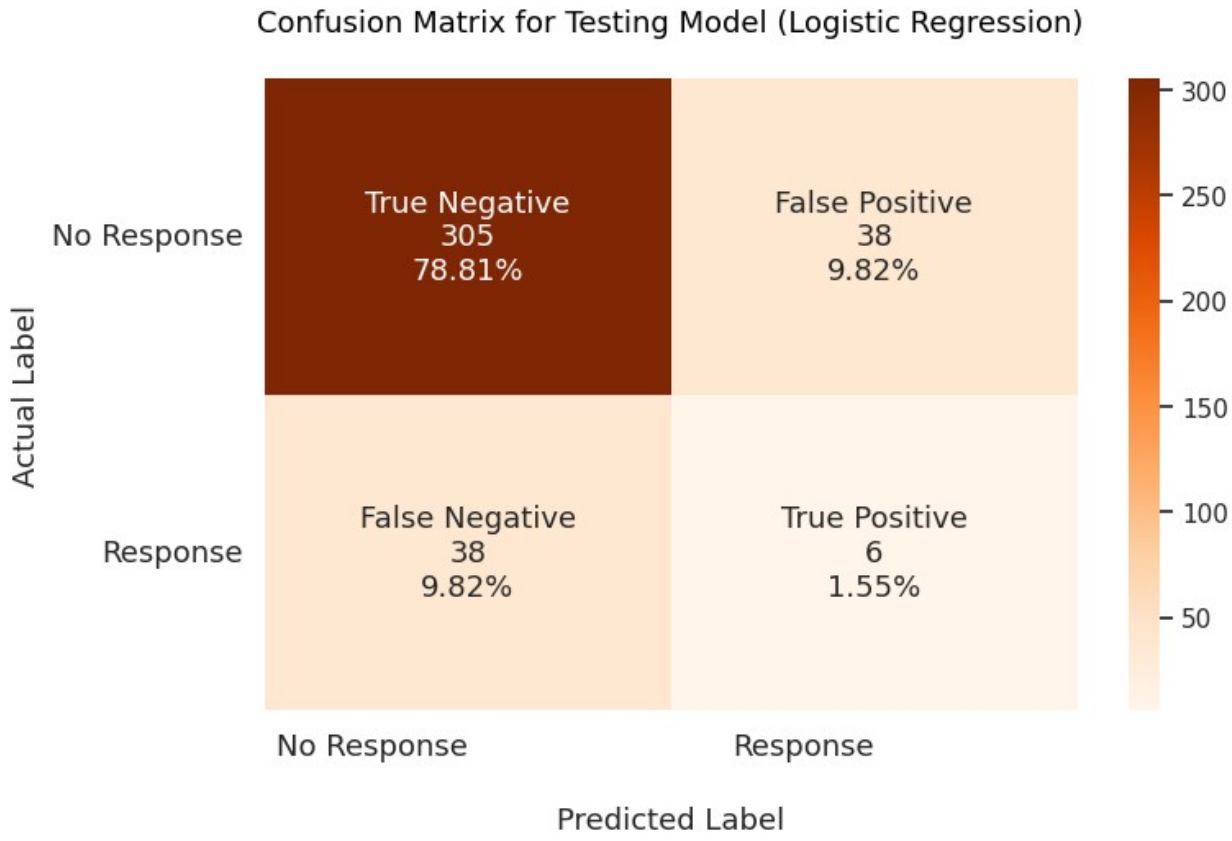

Confusion Matrix for Training Model (Logistic Regression)
True Negative
919
59.48%
False Positive
111
7.18%
False Negative
340
22.01%
True Positive
175
11.33%
No Response
Response
Actual Label
No Response
Response
Predicted Label
900
800
700
600
500
400
300
200

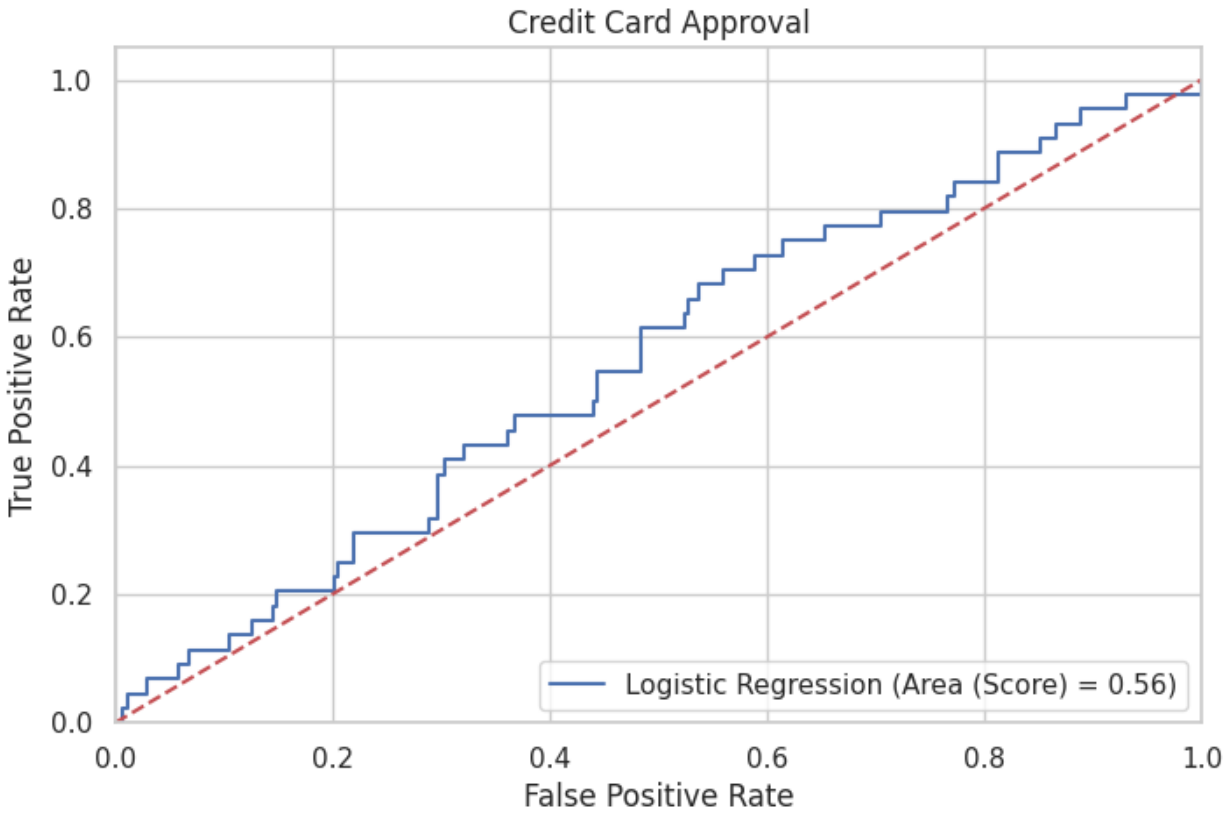


Performance of Testing Model

```
model_eval_test(log_model, "Logistic Regression", X_test, y_test)

Classification Report Testing Model (Logistic Regression):

Accuracy = 0.804
Precision = 0.136
Recall = 0.136
F0.5 Score = 0.136
F1 Score = 0.136
Cross Val F1 (k=5) = 0.0
ROC AUC = 0.562
Cross Val ROC AUC (k=5) = 0.574

              precision    recall  f1-score   support

           0       0.89      0.89      0.89       343
           1       0.14      0.14      0.14        44

    accuracy                           0.80       387
   macro avg       0.51      0.51      0.51       387
weighted avg       0.80      0.80      0.80       387

==== Actual Data (Test) =====
```

```
Total = 387
No Response = 343
Response = 44
==== Predicted Data (Test) =====
TP = 6, FP = 38, TN = 305, FN = 38
Predictly Correct = 311
Predictly Wrong = 76
```

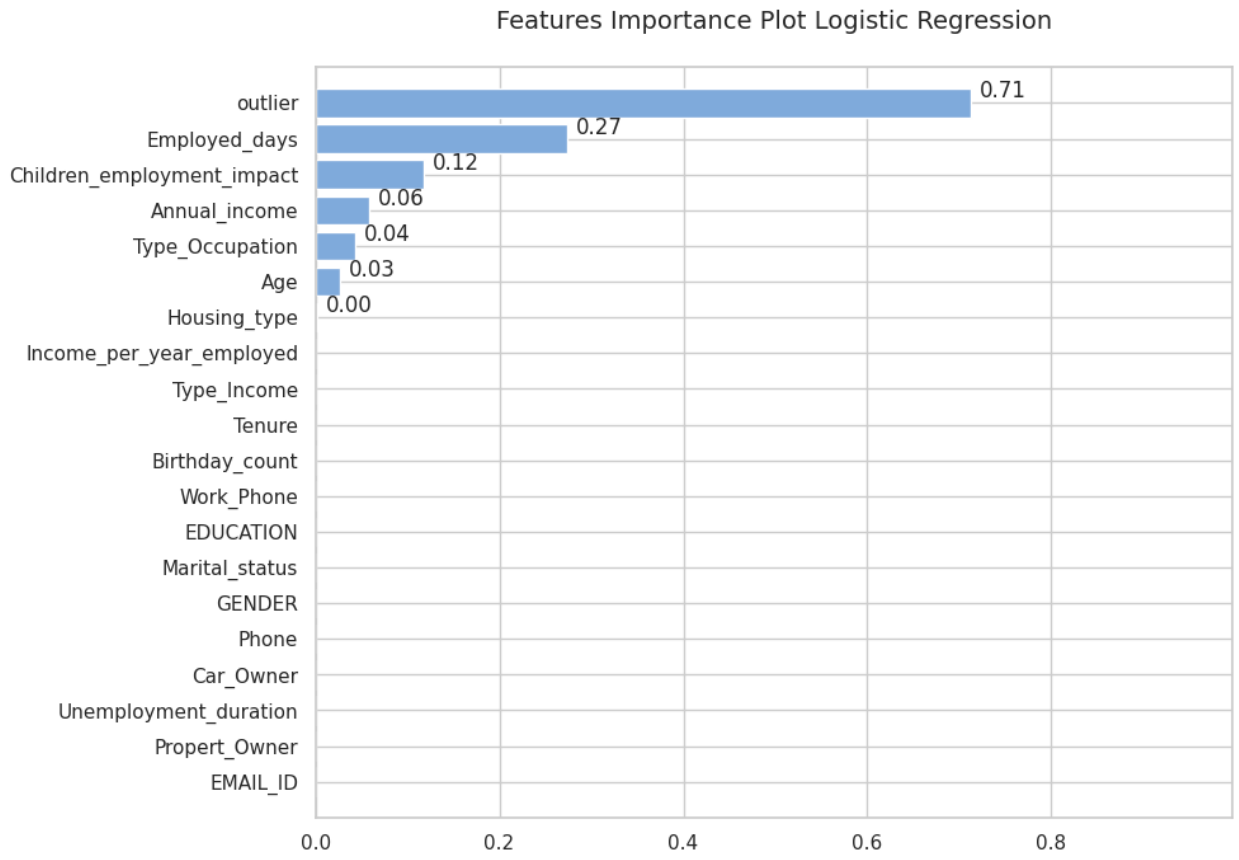





```
acc_log_train=round(log_model.score(X_train,y_train)*100,2)
acc_log_test=round(log_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_log_train))
print("Test Accuracy: {} %".format(acc_log_test))
```

```
Training Accuracy: 70.81 %
Test Accuracy: 80.36 %
```

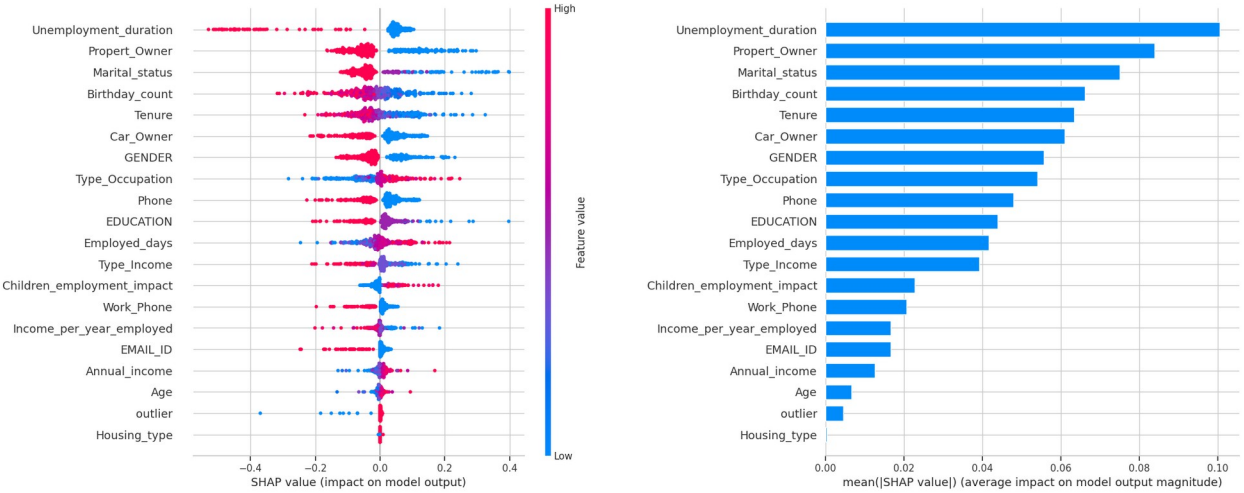
```
feature_importance_plot(log_model, "Logistic Regression", X_train)
```




```
shap_plot(log_model, "Logistic Regression", X_test)
```

```
<IPython.core.display.HTML object>
```

```
PermutationExplainer explainer: 388it [01:11,  5.03it/s]
```




```
<shap.plots._force.AdditiveForceVisualizer at 0x78735e5f9180>
```

4. Naive Bayes

Metode ini didasarkan pada Teorema Bayes dan mengasumsikan independensi fitur. Naive Bayes menghitung probabilitas kelas berdasarkan probabilitas fitur dan mengambil keputusan berdasarkan probabilitas tertinggi.

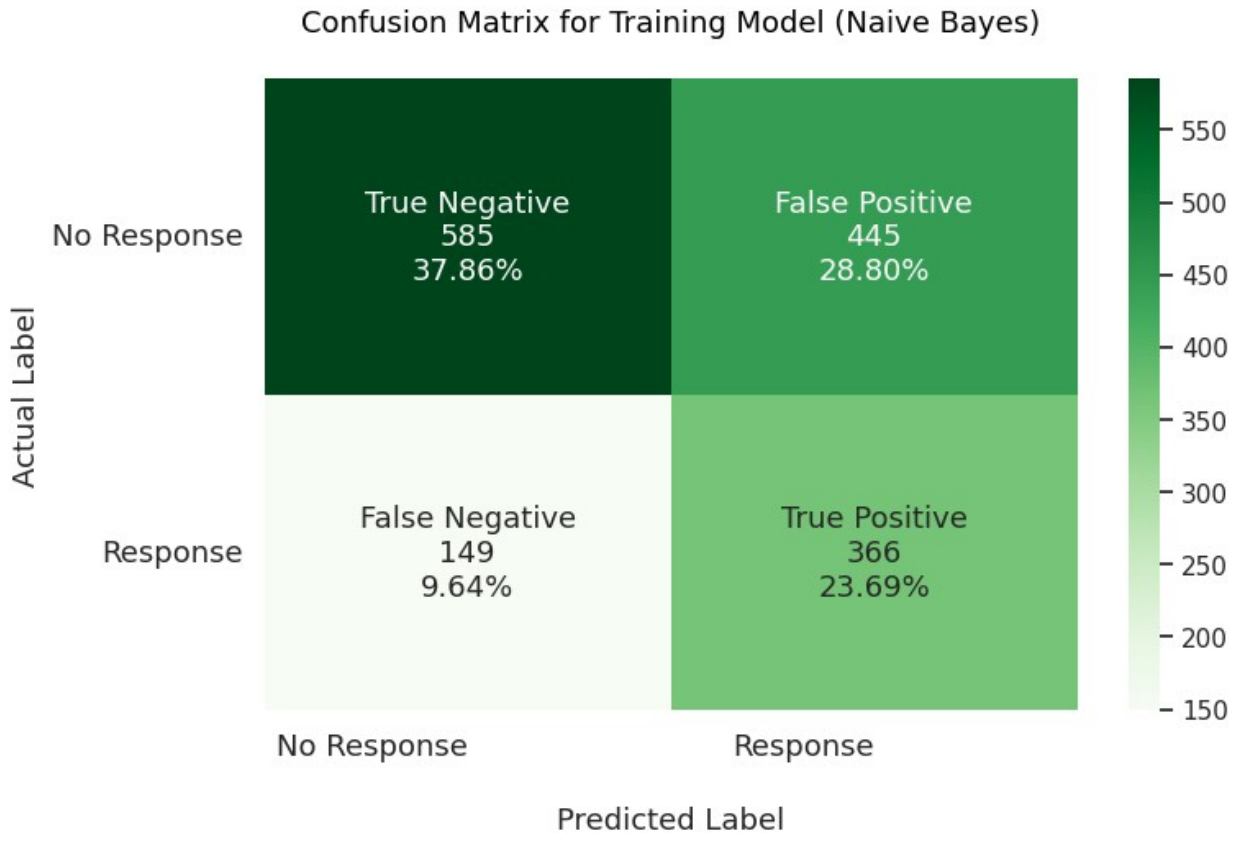
```
# train the model
gnb_model = GaussianNB().fit(X_train, y_train)
print(gnb_model)
eval_classification(gnb_model, "Naive Bayes")
```

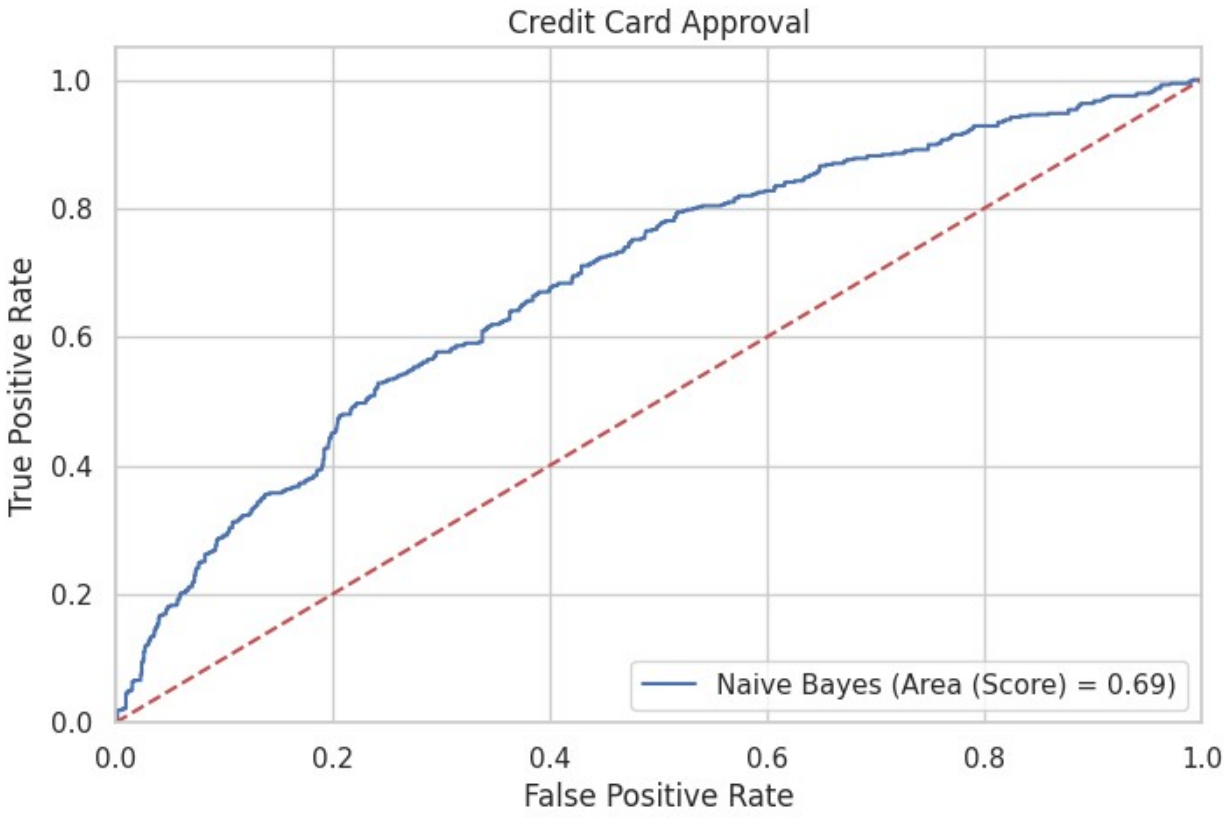
```
GaussianNB()

<pandas.io.formats.style.Styler at 0x78738104b400>
```

Performance of Training Model

```
model_eval_train(gnb_model, "Naive Bayes", X_train, y_train)
```

```
Classification Report Training Model (Naive Bayes):

Accuracy = 0.616
Precision = 0.451
Recall = 0.711
F0.5 Score = 0.487
F1 Score = 0.552
Cross Val F1 (k=5) = 0.125
ROC AUC = 0.688
Cross Val ROC AUC (k=5) = 0.633

              precision    recall  f1-score   support

           0       0.80      0.57      0.66      1030
           1       0.45      0.71      0.55       515

    accuracy                           0.62      1545
   macro avg       0.62      0.64      0.61      1545
weighted avg       0.68      0.62      0.63      1545

==== Actual Data (Train) =====
Total = 1545
No Response = 1030
Response = 515
==== Predicted Data (Train) =====
TP = 366, FP = 445, TN = 585, FN = 149
Predictly Correct = 951
Predictly Wrong = 594
```

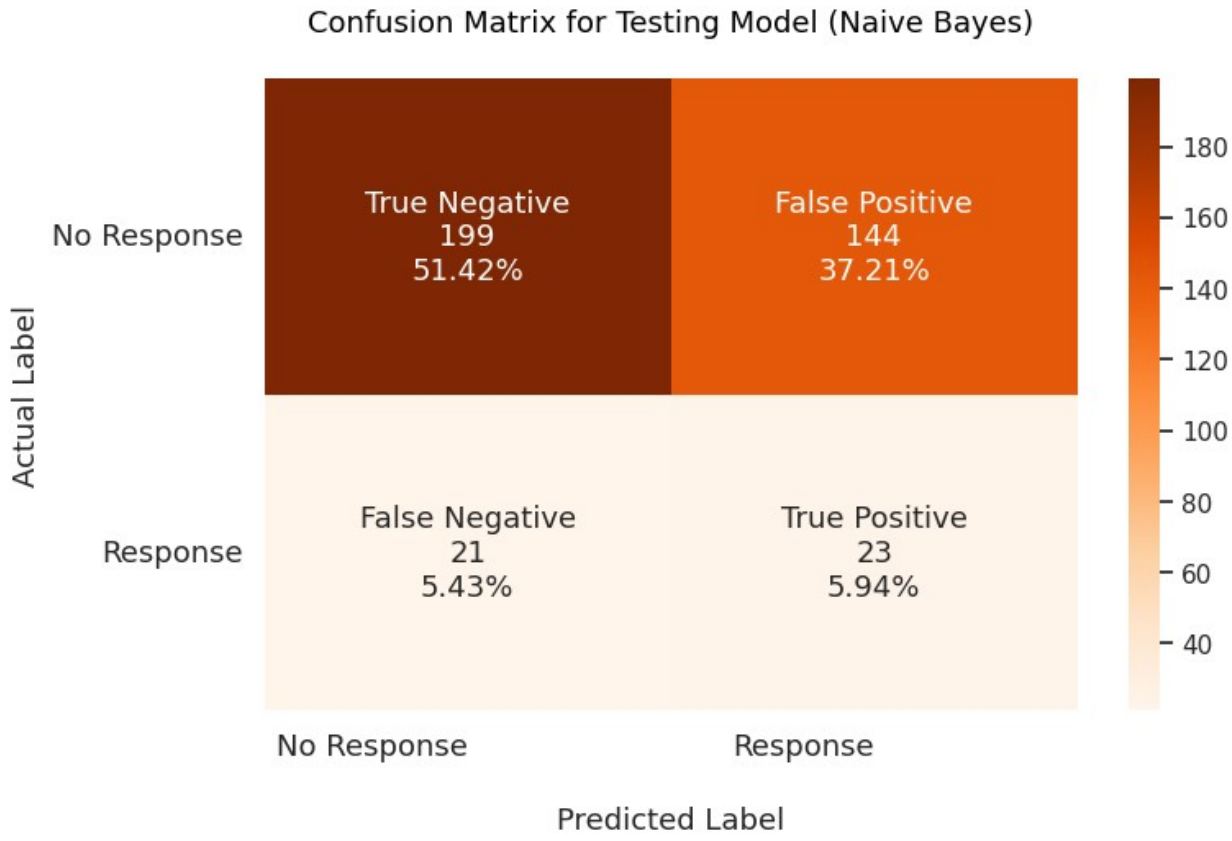

Confusion Matrix for Training Model (Naive Bayes)
True Negative
585
37.86%
False Positive
445
28.80%
False Negative
149
9.64%
True Positive
366
23.69%
No Response
Response
Actual Label
No Response
Response
Predicted Label
550
500
450
400
350
300
250
200
150

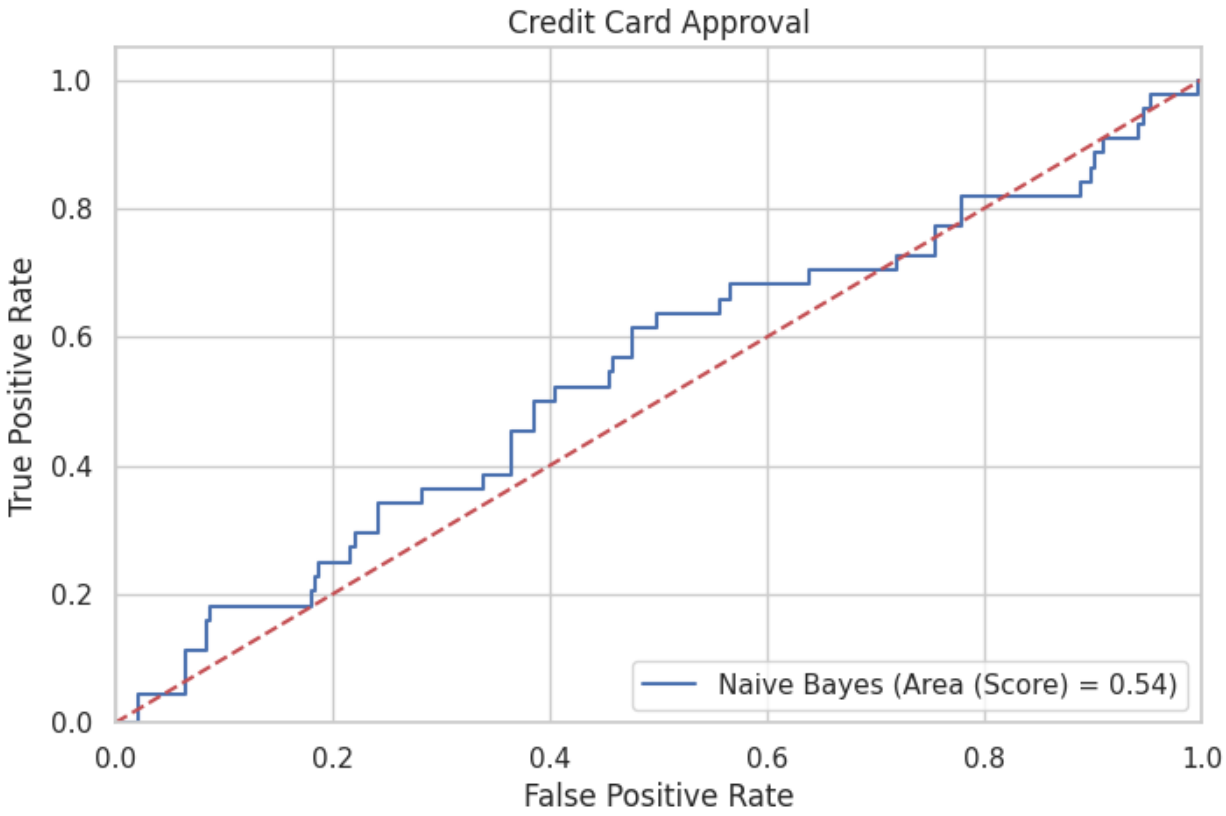


Performance of Testing Model

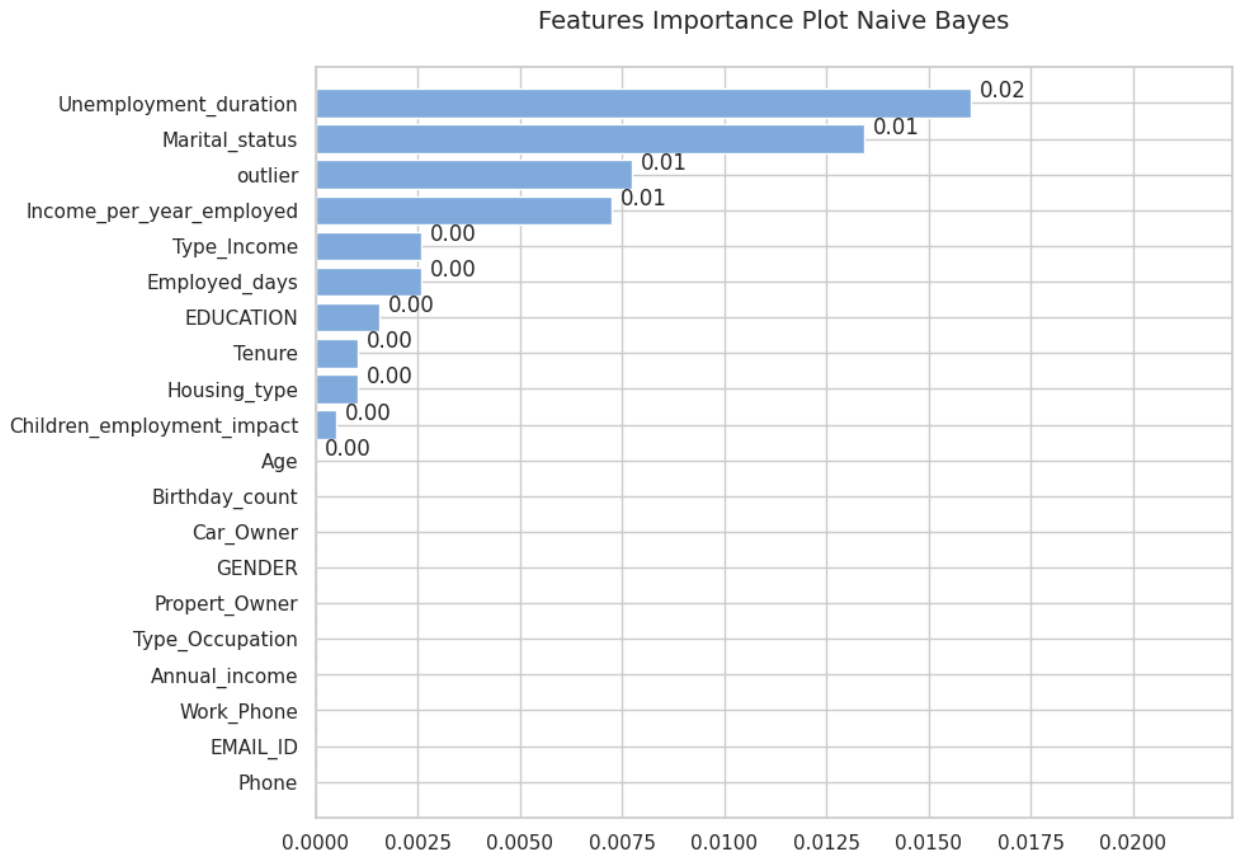
```
model_eval_test(gnb_model, "Naive Bayes", X_test, y_test)

Classification Report Testing Model (Naive Bayes):

Accuracy = 0.574
Precision = 0.138
Recall = 0.523
F0.5 Score = 0.162
F1 Score = 0.218
Cross Val F1 (k=5) = 0.083
ROC AUC = 0.541
Cross Val ROC AUC (k=5) = 0.577

              precision    recall  f1-score   support

           0       0.90      0.58      0.71       343
           1       0.14      0.52      0.22        44

    accuracy                           0.57       387
   macro avg       0.52      0.55      0.46       387
weighted avg       0.82      0.57      0.65       387

==== Actual Data (Test) =====
```

```
Total = 387
No Response = 343
Response = 44
==== Predicted Data (Test) =====
TP = 23, FP = 144, TN = 199, FN = 21
Predictly Correct = 222
Predictly Wrong = 165
```





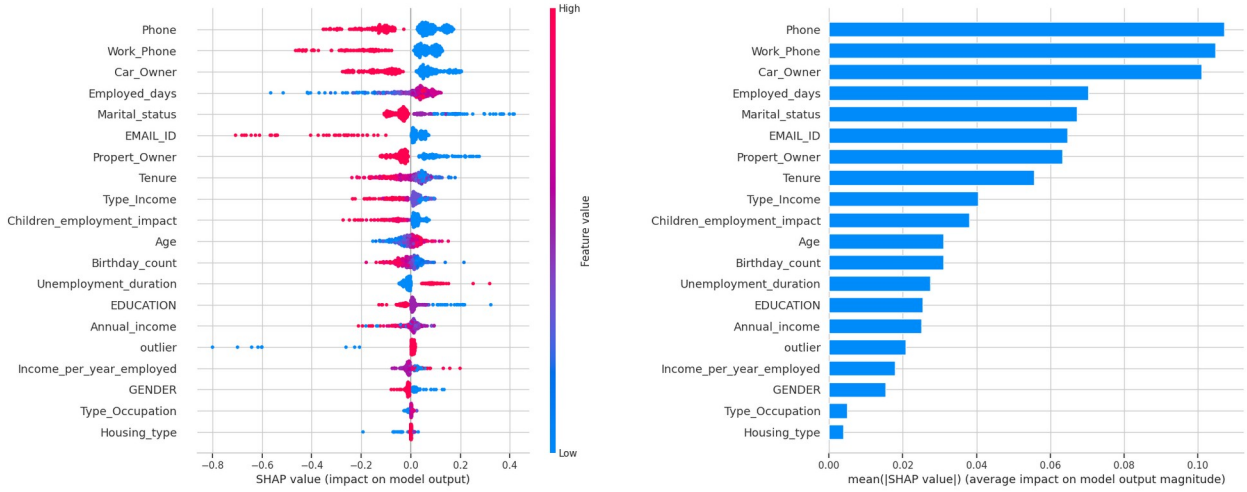

```
acc_gnb_train=round(gnb_model.score(X_train,y_train)*100,2)
acc_gnb_test=round(gnb_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_gnb_train))
print("Test Accuracy: {} %".format(acc_gnb_test))

Training Accuracy: 61.55 %
Test Accuracy: 57.36 %

feature_importance_plot(gnb_model, "Naive Bayes", X_train)
```




```
shap_plot(gnb_model, "Naive Bayes", X_test)
```

```
<IPython.core.display.HTML object>
```

```
PermutationExplainer explainer: 388it [00:17,  9.61it/s]
```




```
<shap.plots._force.AdditiveForceVisualizer at 0x787301476fe0>
```

5. K-Nearest Neighbors

Memprediksi label data berdasarkan kelas mayoritas dari K tetangga terdekat dalam ruang fitur. K adalah parameter yang menentukan jumlah tetangga yang akan digunakan dalam klasifikasi.

```
# train the model
knn_model = KNeighborsClassifier().fit(X_train,y_train)
print(knn_model)
eval_classification(knn_model, "K-Nearest Neighbors")
```

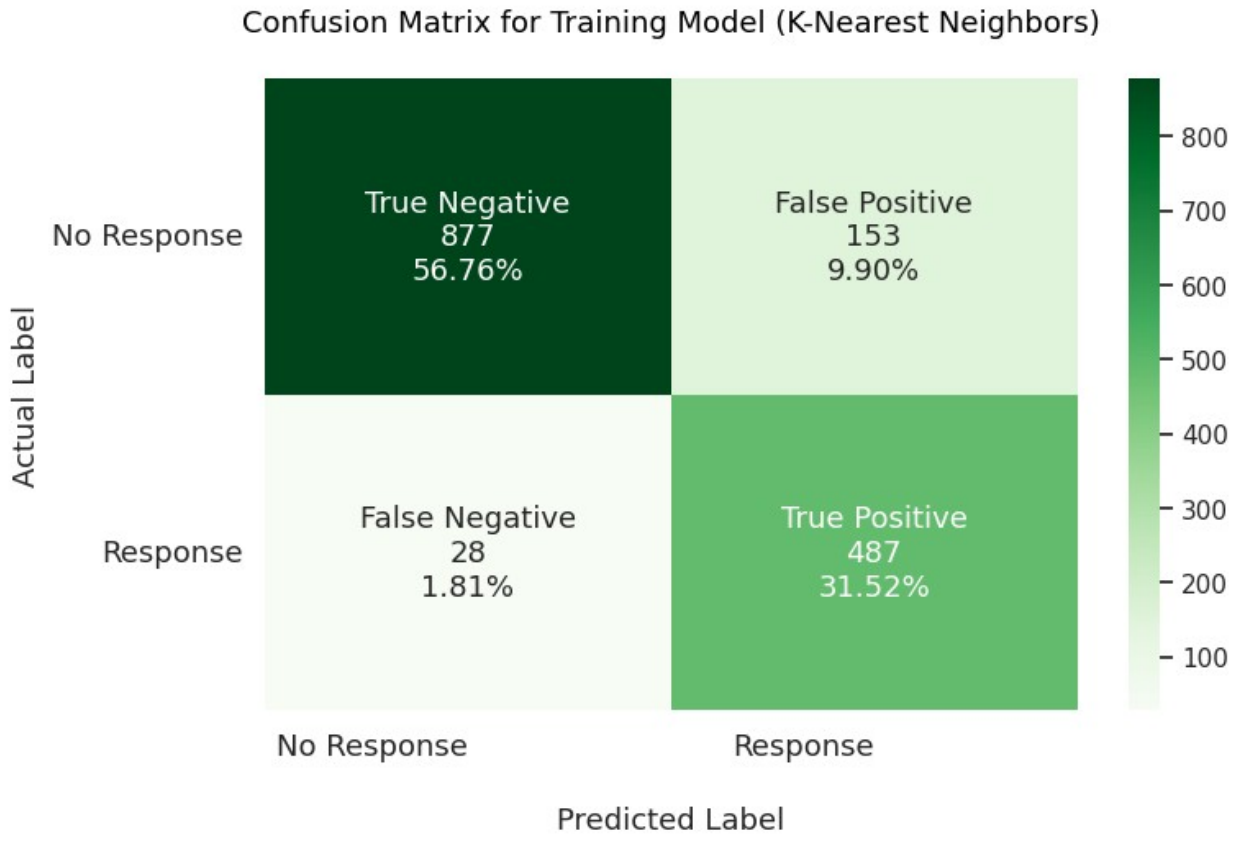
```
KNeighborsClassifier()
```

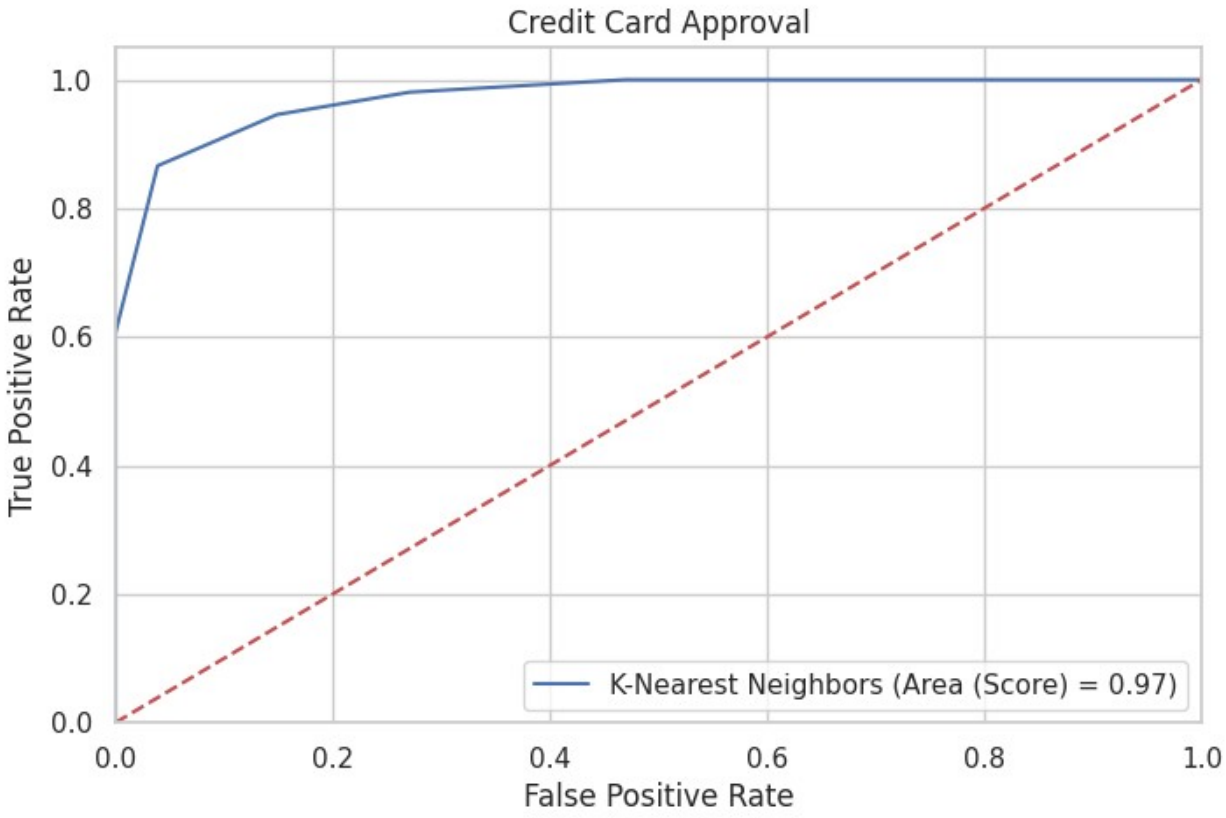
```
<pandas.io.formats.style.Styler at 0x787358a8a5f0>
```

Performance of Training Model

```
model_eval_train(knn_model, "K-Nearest Neighbors", X_train, y_train)
```

```
Classification Report Training Model (K-Nearest Neighbors):

Accuracy = 0.883
Precision = 0.761
Recall = 0.946
F0.5 Score = 0.792
F1 Score = 0.843
Cross Val F1 (k=5) = 0.288
ROC AUC = 0.973
Cross Val ROC AUC (k=5) = 0.896

              precision    recall  f1-score   support

           0       0.97      0.85      0.91      1030
           1       0.76      0.95      0.84       515

    accuracy                           0.88      1545
   macro avg       0.86      0.90      0.87      1545
weighted avg       0.90      0.88      0.89      1545

==== Actual Data (Train) =====
Total = 1545
No Response = 1030
Response = 515
==== Predicted Data (Train) =====
TP = 487, FP = 153, TN = 877, FN = 28
Predictly Correct = 1364
Predictly Wrong = 181
```

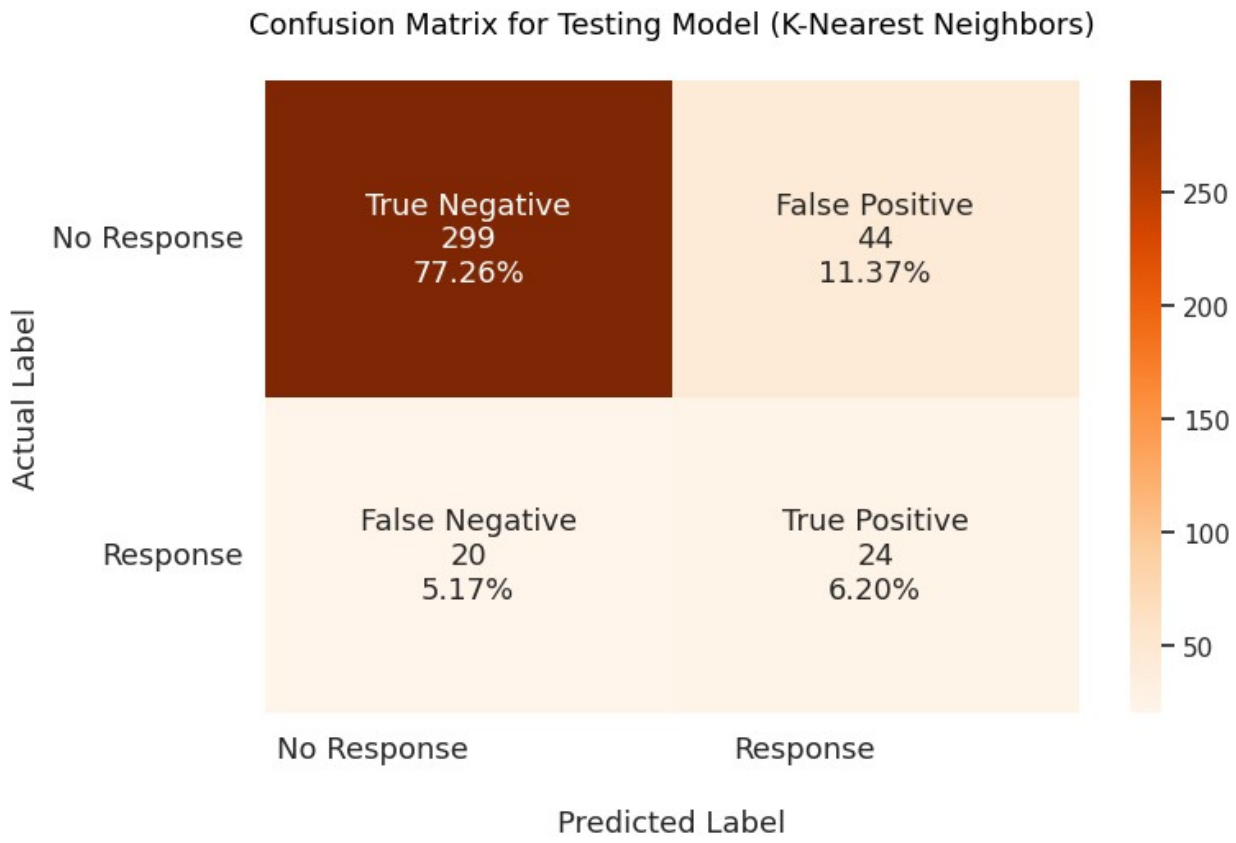

Confusion Matrix for Training Model (K-Nearest Neighbors)
True Negative
877
56.76%
False Positive
153
9.90%
False Negative
28
1.81%
True Positive
487
31.52%
No Response
Response
Actual Label
No Response
Response
Predicted Label
800
700
600
500
400
300
200
100

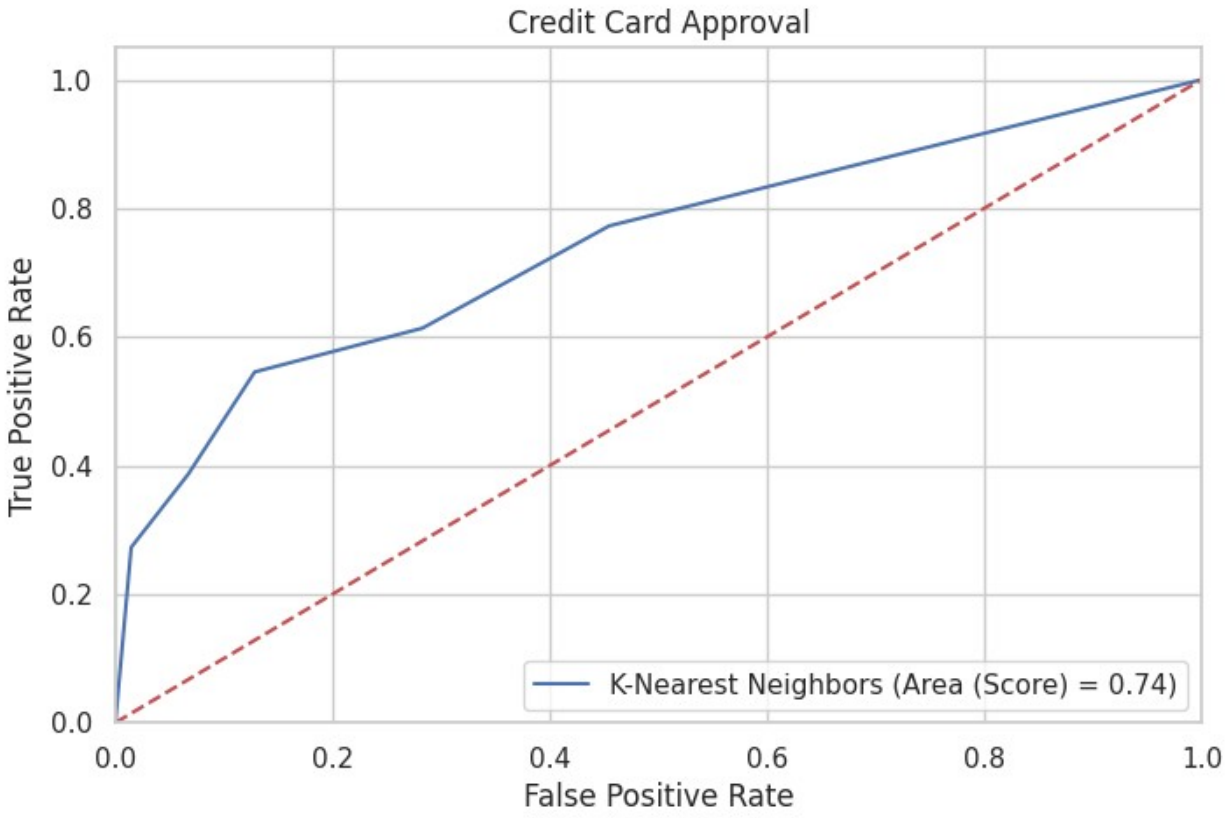


Performance of Testing Model

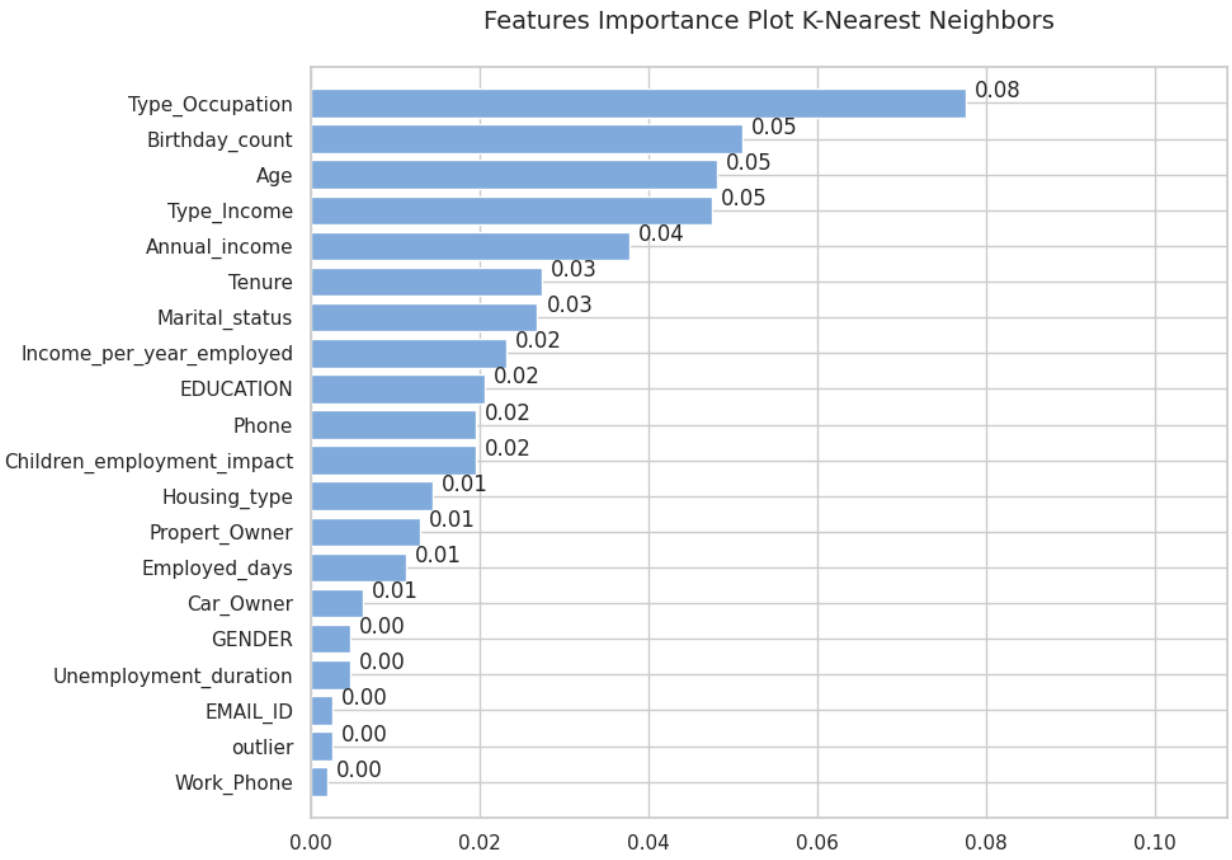
```
model_eval_test(knn_model, "K-Nearest Neighbors", X_test, y_test)

Classification Report Testing Model (K-Nearest Neighbors):

Accuracy = 0.835
Precision = 0.353
Recall = 0.545
F0.5 Score = 0.38
F1 Score = 0.429
Cross Val F1 (k=5) = 0.057
ROC AUC = 0.74
Cross Val ROC AUC (k=5) = 0.634

              precision    recall  f1-score   support

           0       0.94      0.87      0.90       343
           1       0.35      0.55      0.43        44

    accuracy                           0.83       387
   macro avg       0.65      0.71      0.67       387
weighted avg       0.87      0.83      0.85       387

==== Actual Data (Test) =====
```

```
Total = 387
No Response = 343
Response = 44
==== Predicted Data (Test) =====
TP = 24, FP = 44, TN = 299, FN = 20
Predictly Correct = 323
Predictly Wrong = 64
```






```
acc_knn_train=round(knn_model.score(X_train,y_train)*100,2)
acc_knn_test=round(knn_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_knn_train))
print("Test Accuracy: {} %".format(acc_knn_test))

Training Accuracy: 88.28 %
Test Accuracy: 83.46 %

feature_importance_plot(knn_model, "K-Nearest Neighbors", X_train)
```




```
# shap_plot(knn_model, "K-Nearest Neighbors", X_test)
```

6. MLP Classifier (Neural Network)

MLPClassifier adalah singkatan dari Multi-layer Perceptron classifier yang dalam namanya terhubung ke Neural Network. Tidak seperti algoritme klasifikasi lain seperti Support Vectors Machine atau Naive Bayes Classifier, MLPClassifier mengandalkan Neural Network yang mendasari untuk melakukan tugas klasifikasi.

```
# # train the model
# mlp_model = MLPClassifier(random_state=42,
max_iter=len(X_train)).fit(X_train, y_train)

# eval_classification(mlp_model, "MLP Classifier")
```

```
<pandas.io.formats.style.Styler at 0x787301360820>
```

Performance of Training Model

```
# model_eval_train(mlp_model, "MLP Classifier", X_train, y_train)
```

Performance of Testing Model

```
# model_eval_test(mlp_model, "MLP Classifier", X_test, y_test)

# acc_mlp_train=round(mlp_model.score(X_train,y_train)*100,2)
# acc_mlp_test=round(mlp_model.score(X_test,y_test)*100,2)
# print("Training Accuracy: {} %".format(acc_mlp_train))
# print("Test Accuracy: {} %".format(acc_mlp_test))

# feature_importance_plot(mlp_model, "MLP Classifier", X_train)

# shap_plot(mlp_model, "MLP Classifier", X_test)
```

7. Adaboost Classifier

is meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

```
# train the model
adab_model = AdaBoostClassifier(random_state=42).fit(X_train, y_train)
eval_classification(adab_model, "Adaboost Classifier")
```

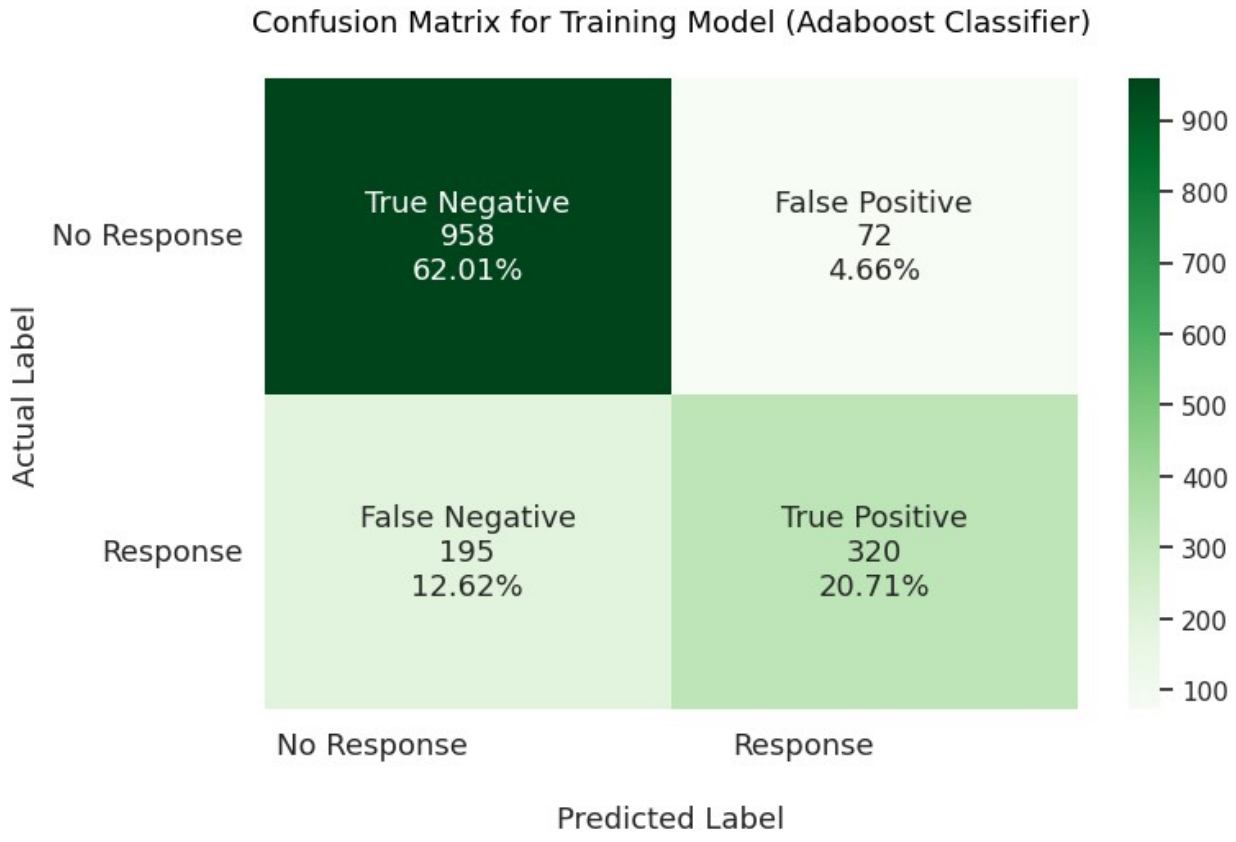
```
<pandas.io.formats.style.Styler at 0x7872fe14b040>
```

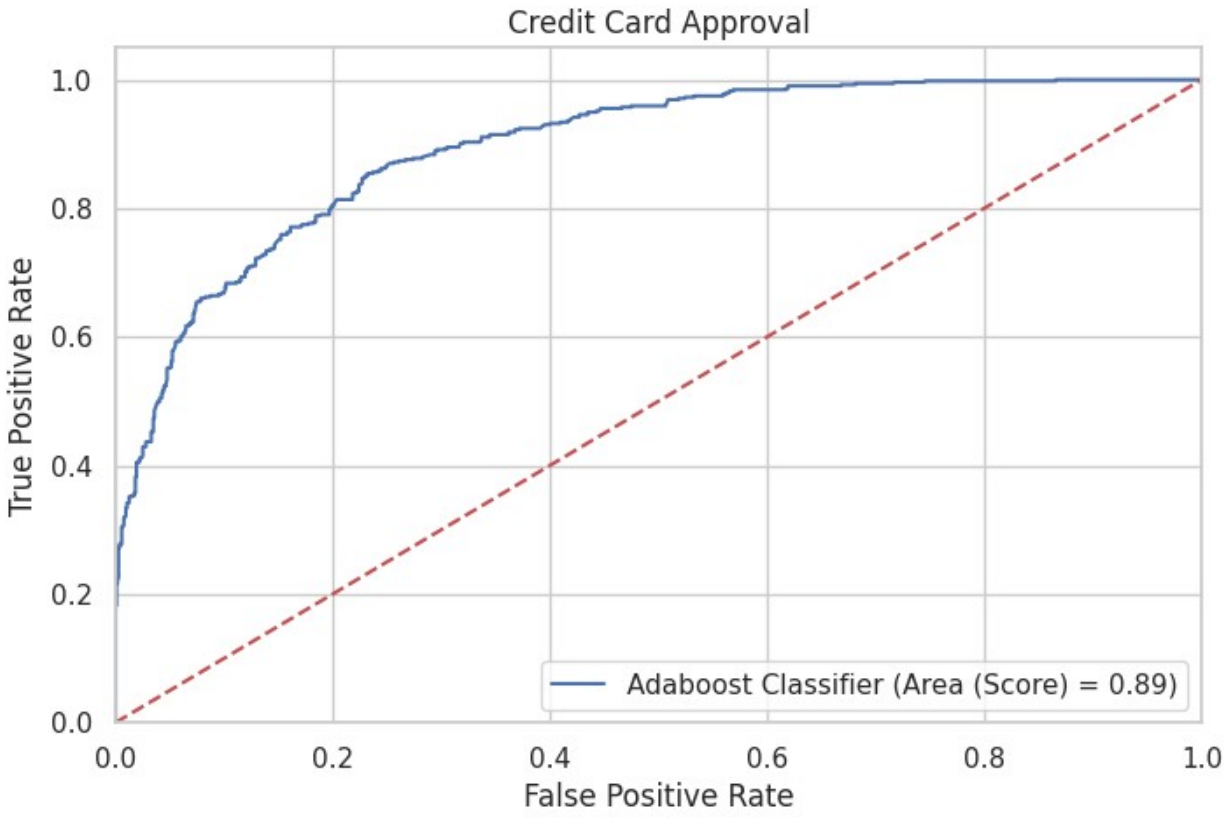
Performance of Training Model

```
model_eval_train(adab_model, "Adaboost Classifier", X_train, y_train)
```

```
Classification Report Training Model (Adaboost Classifier):

Accuracy = 0.827
Precision = 0.816
Recall = 0.621
F0.5 Score = 0.768
F1 Score = 0.706
Cross Val F1 (k=5) = 0.18
ROC AUC = 0.894
Cross Val ROC AUC (k=5) = 0.844

              precision    recall  f1-score   support

           0       0.83      0.93      0.88      1030
           1       0.82      0.62      0.71       515

    accuracy                           0.83      1545
   macro avg       0.82      0.78      0.79      1545
weighted avg       0.83      0.83      0.82      1545

==== Actual Data (Train) =====
Total = 1545
```

```
No Response = 1030
Response = 515
==== Predicted Data (Train) =====
TP = 320, FP = 72, TN = 958, FN = 195
Predictly Correct = 1278
Predictly Wrong = 267
```

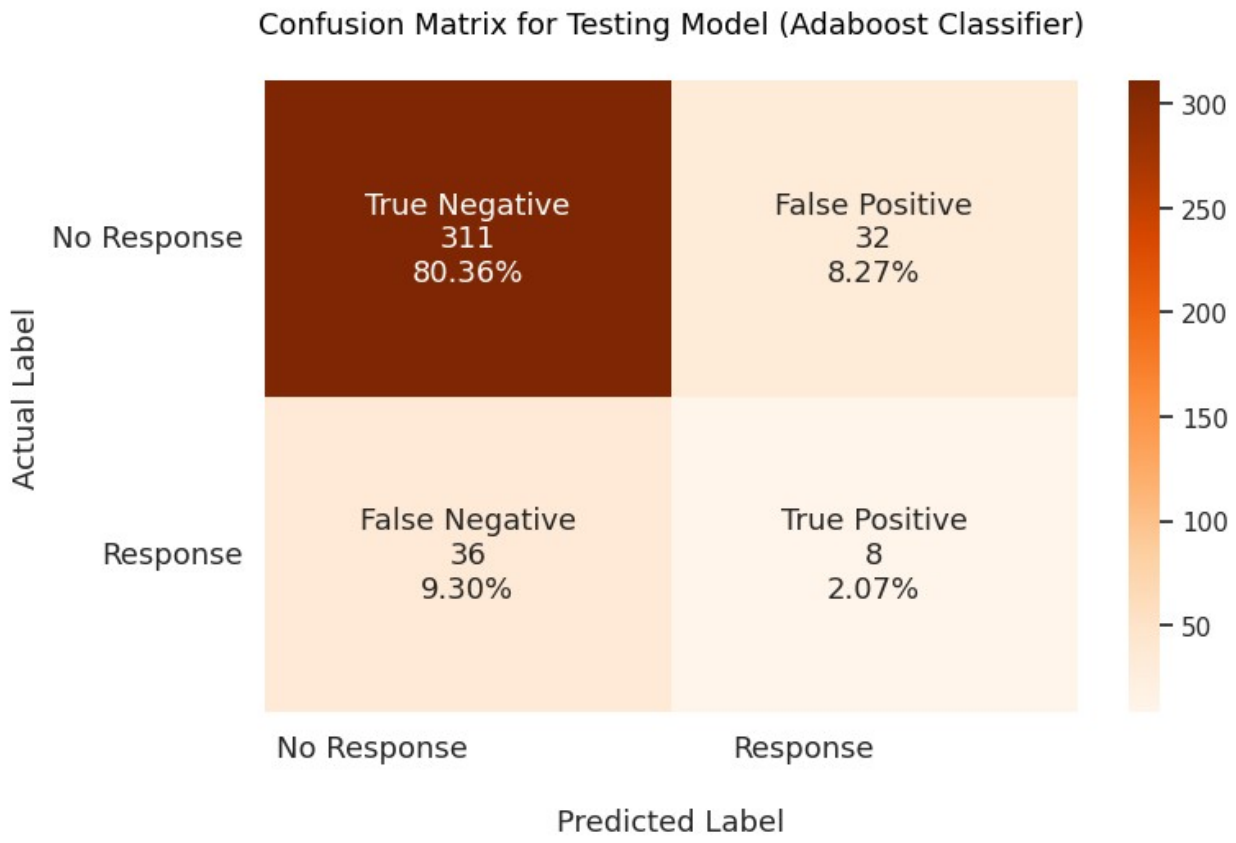


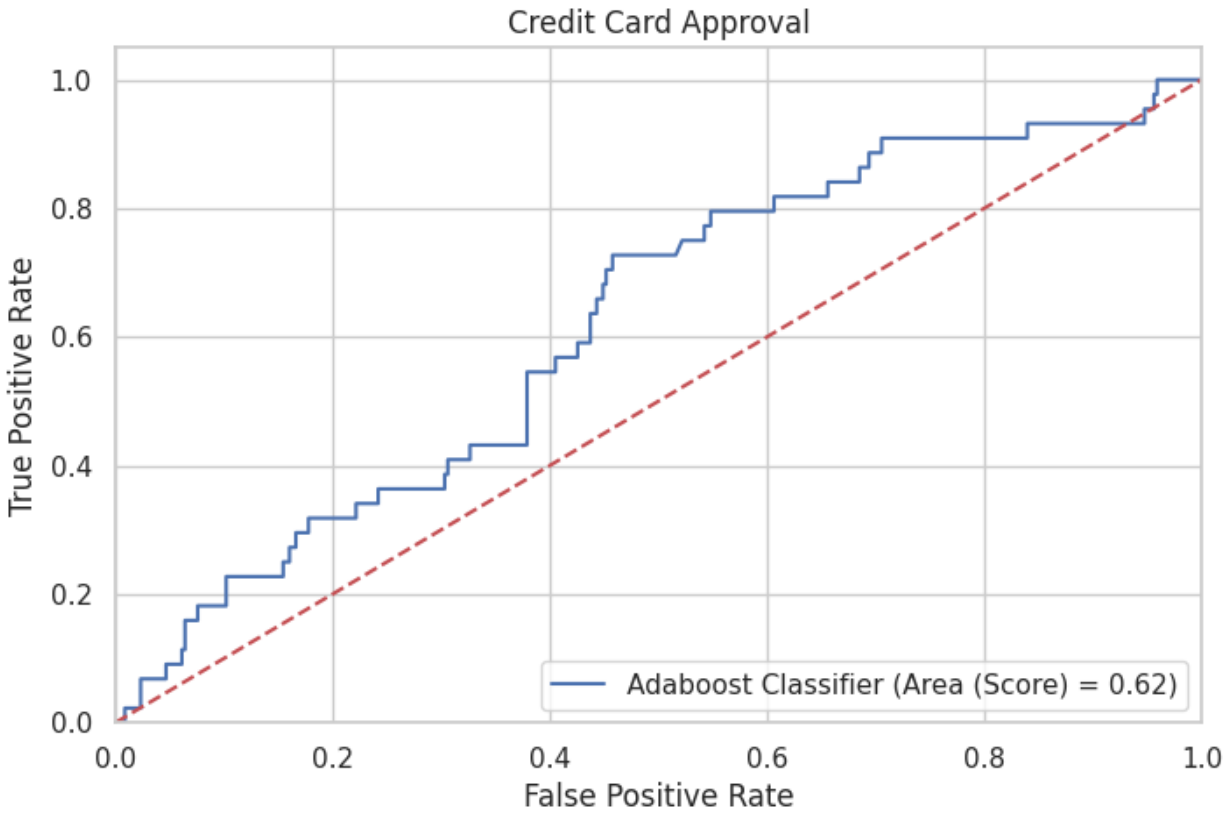


Performance of Testing Model

```
model_eval_test(adab_model, "Adaboost Classifier", X_test, y_test)

Classification Report Testing Model (Adaboost Classifier):

Accuracy = 0.824
Precision = 0.2
Recall = 0.182
F0.5 Score = 0.196
F1 Score = 0.19
Cross Val F1 (k=5) = 0.139
ROC AUC = 0.621
Cross Val ROC AUC (k=5) = 0.688

              precision    recall  f1-score   support

           0       0.90      0.91      0.90       343
           1       0.20      0.18      0.19        44

    accuracy                           0.82       387
   macro avg       0.55      0.54      0.55       387
weighted avg       0.82      0.82      0.82       387

==== Actual Data (Test) =====
```

```
Total = 387
No Response = 343
Response = 44
==== Predicted Data (Test) =====
TP = 8, FP = 32, TN = 311, FN = 36
Predictly Correct = 319
Predictly Wrong = 68
```

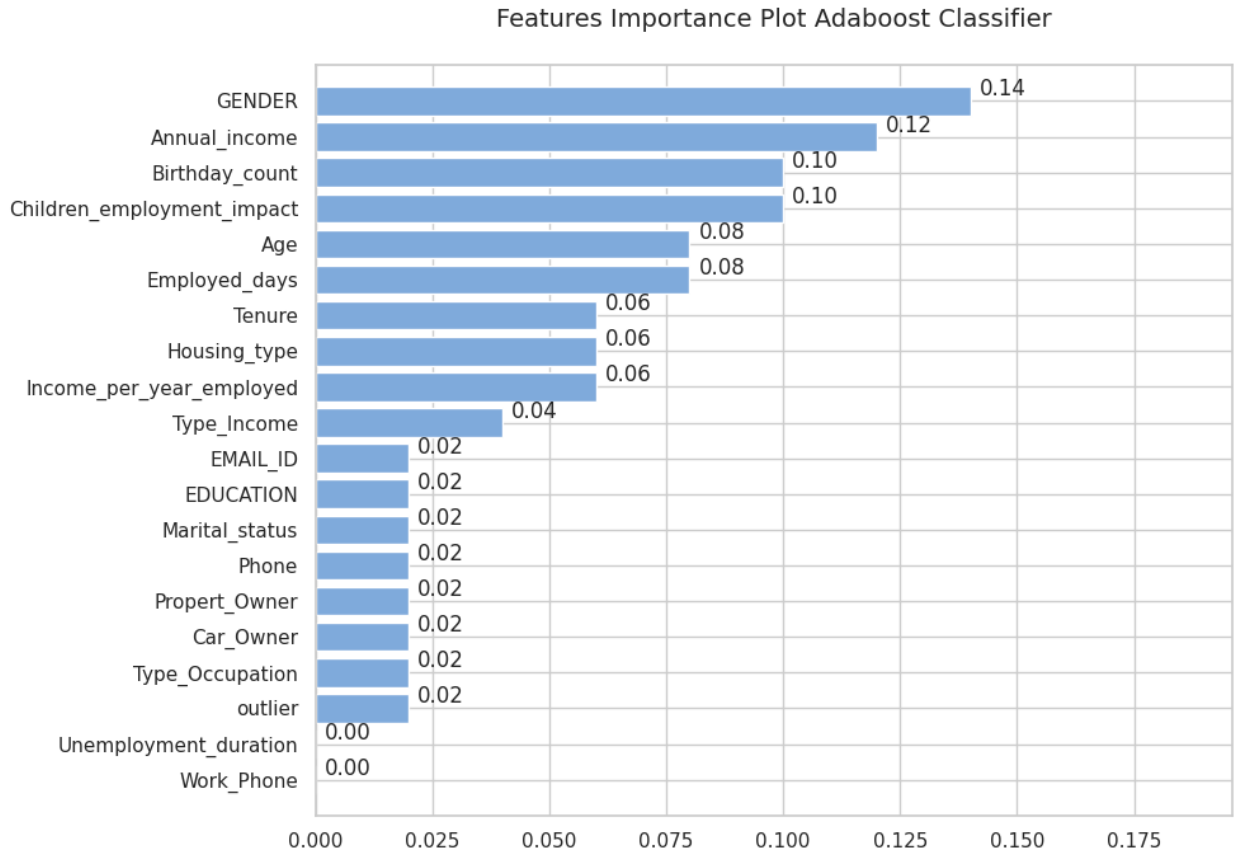





```
acc_adab_train=round(adab_model.score(X_train,y_train)*100,2)
acc_adab_test=round(adab_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_adab_train))
print("Test Accuracy: {} %".format(acc_adab_test))
```

```
Training Accuracy: 82.72 %
Test Accuracy: 82.43 %
```

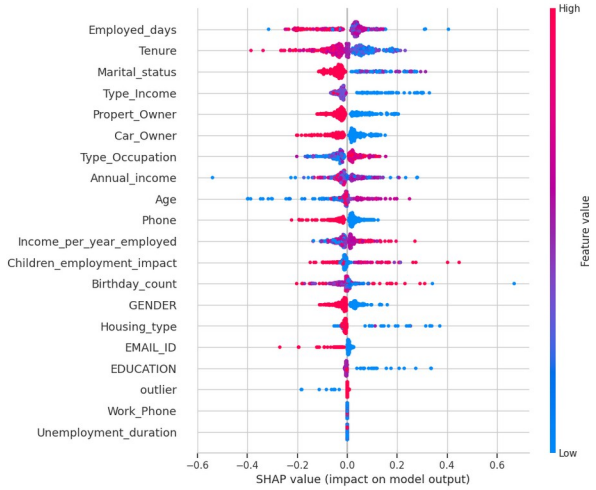
```
feature_importance_plot(adab_model, "Adaboost Classifier")
```

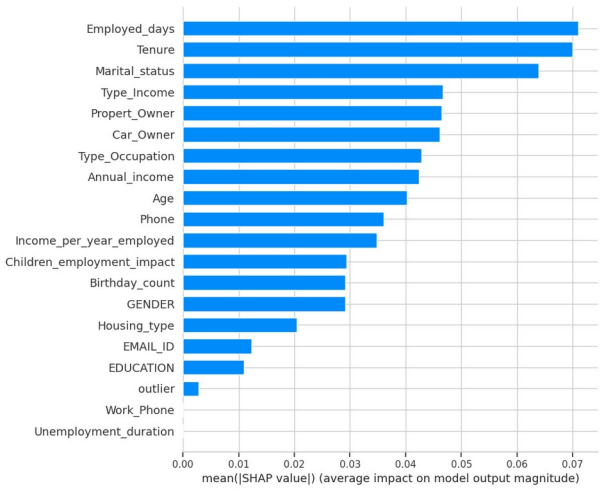



```
shap_plot(adab_model, "Adaboost Classifier", X_test)
```

```
<IPython.core.display.HTML object>
```

```
PermutationExplainer explainer: 388it [02:18,  2.59it/s]
```






```
<shap.plots._force.AdditiveForceVisualizer at 0x7873020098a0>
```

8. XGBoost Classifier

XGBoost (Extream Gradient Boosting) merupakan pengembangan dari algoritma Gradient Tree Boosting yang berbasis algoritma ensemble, secara efisien dapat menangani permasalahan machine learning yang berskala besar.

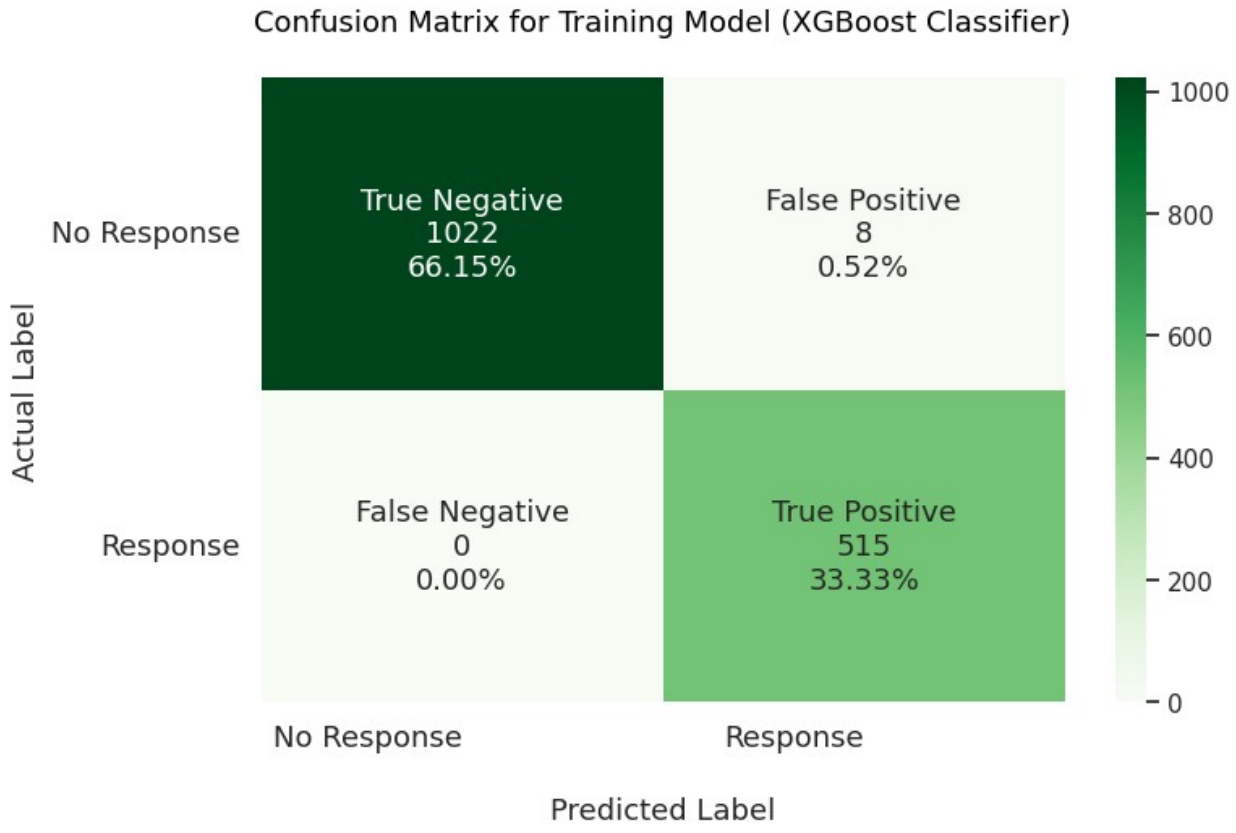
```
# train the model
xgb_model = XGBClassifier(random_state=42).fit(X_train, y_train)
eval_classification(xgb_model, "XGBoost Classifier")
```

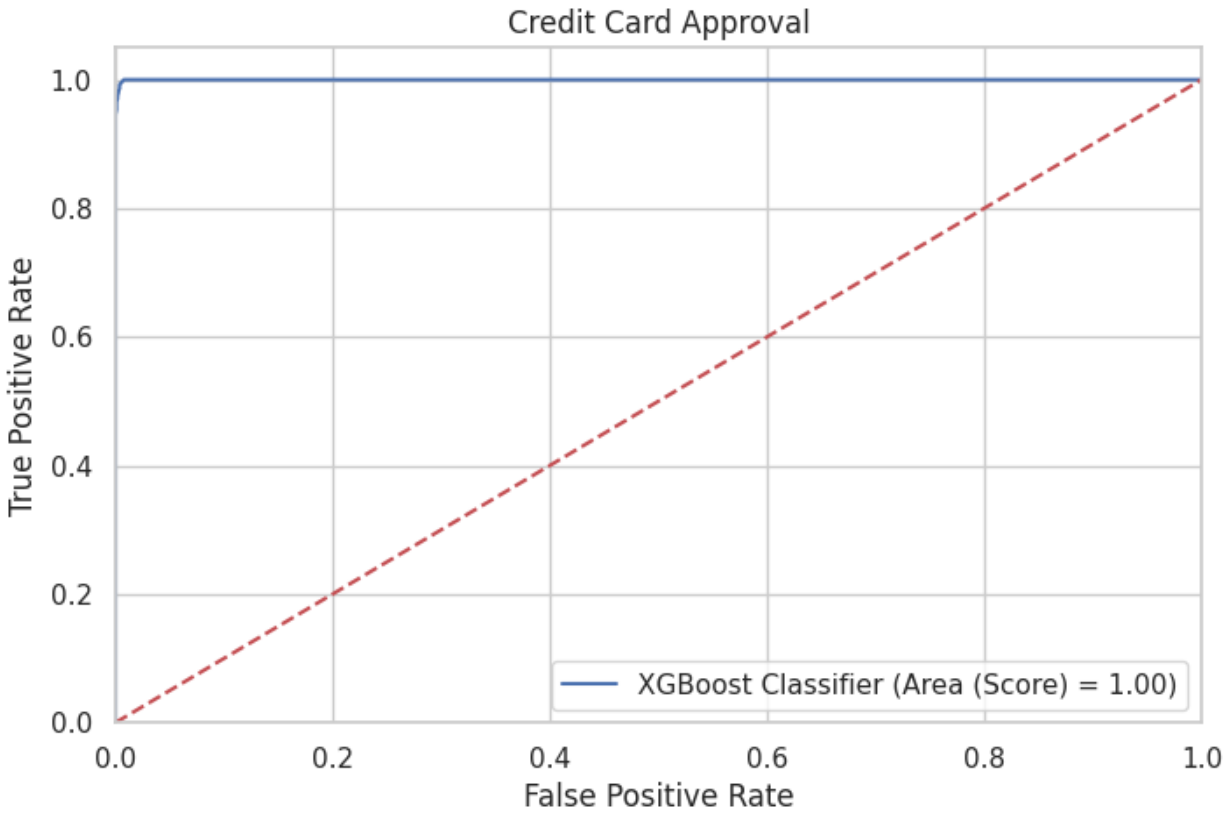
```
<pandas.io.formats.style.Styler at 0x7872fcd20070>
```

Performance of Training Model

```
model_eval_train(xgb_model, "XGBoost Classifier", X_train, y_train)
```

```
Classification Report Training Model (XGBoost Classifier):

Accuracy = 0.995
Precision = 0.985
Recall = 1.0
F0.5 Score = 0.988
F1 Score = 0.992
Cross Val F1 (k=5) = 0.978
ROC AUC = 1.0
Cross Val ROC AUC (k=5) = 1.0

              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1030
           1       0.98      1.00      0.99       515

    accuracy                           0.99      1545
   macro avg       0.99      1.00      0.99      1545
weighted avg       0.99      0.99      0.99      1545

==== Actual Data (Train) =====
Total = 1545
No Response = 1030
Response = 515
==== Predicted Data (Train) =====
TP = 515, FP = 8, TN = 1022, FN = 0
Predictly Correct = 1537
Predictly Wrong = 8
```

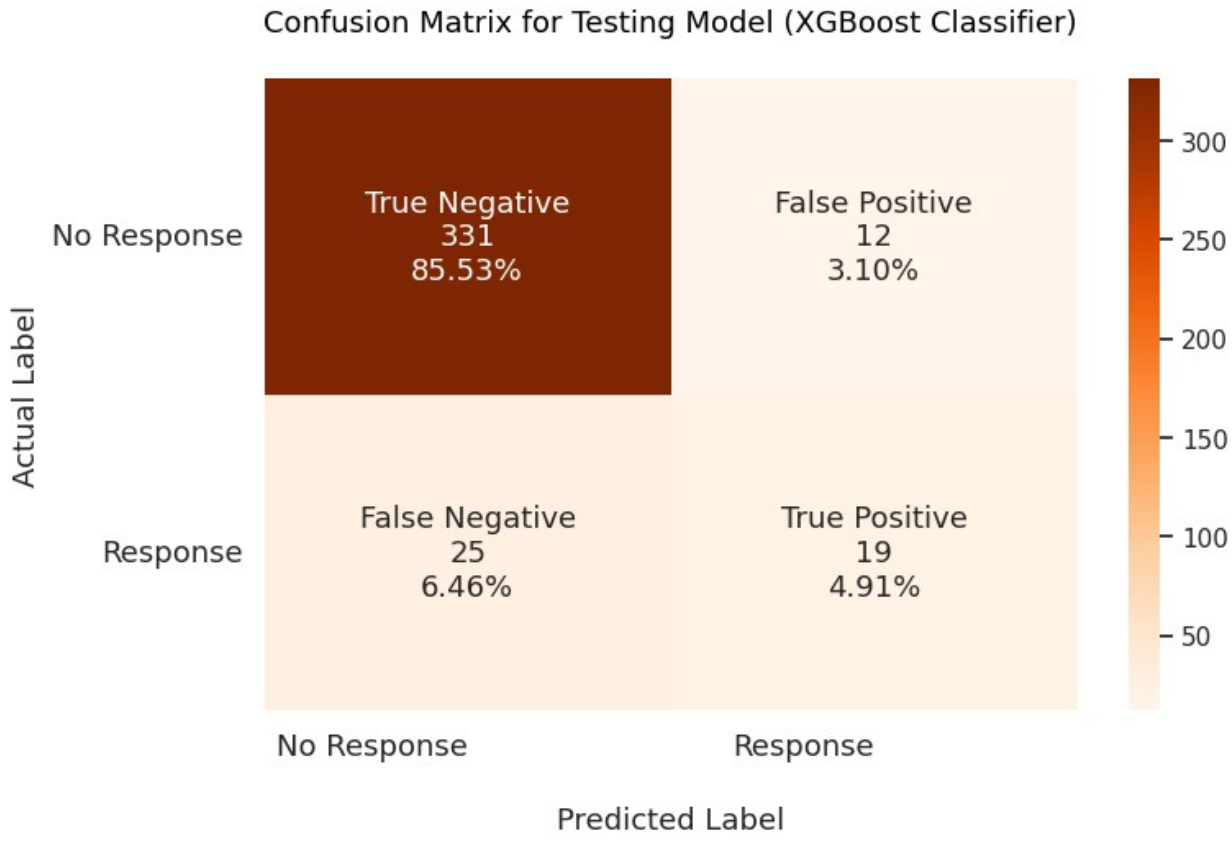

Confusion Matrix for Training Model (XGBoost Classifier)
True Negative
1022
66.15%
False Positive
8
0.52%
False Negative
0
0.00%
True Positive
515
33.33%
No Response
Response
Actual Label
No Response
Response
Predicted Label
1000
800
600
400
200
0

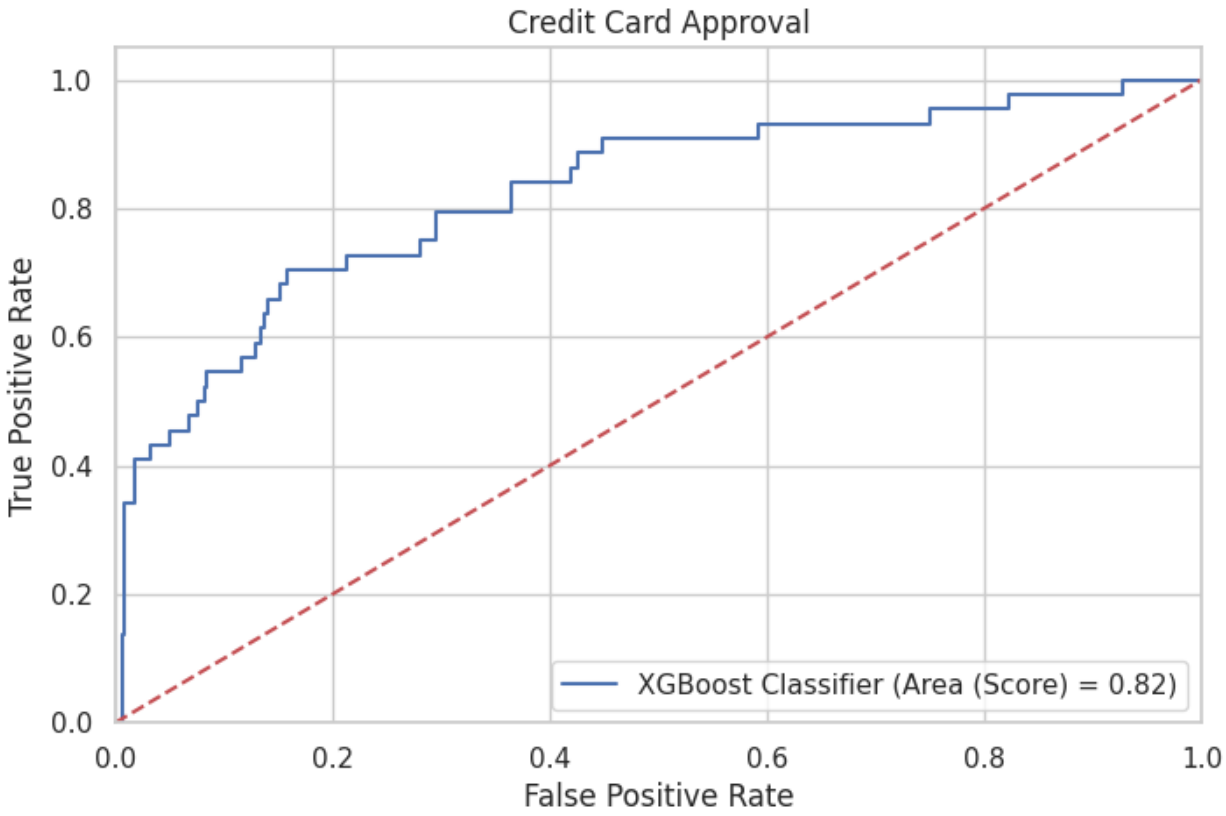


Performance of Testing Model

```
model_eval_test(xgb_model, "XGBoost Classifier", X_test, y_test)

Classification Report Testing Model (XGBoost Classifier):

Accuracy = 0.904
Precision = 0.613
Recall = 0.432
F0.5 Score = 0.565
F1 Score = 0.507
Cross Val F1 (k=5) = 0.506
ROC AUC = 0.825
Cross Val ROC AUC (k=5) = 0.799

              precision    recall  f1-score   support

           0       0.93      0.97      0.95       343
           1       0.61      0.43      0.51        44

    accuracy                           0.90       387
   macro avg       0.77      0.70      0.73       387
weighted avg       0.89      0.90      0.90       387

==== Actual Data (Test) =====
```

```
Total = 387
No Response = 343
Response = 44
==== Predicted Data (Test) =====
TP = 19, FP = 12, TN = 331, FN = 25
Predictly Correct = 350
Predictly Wrong = 37
```

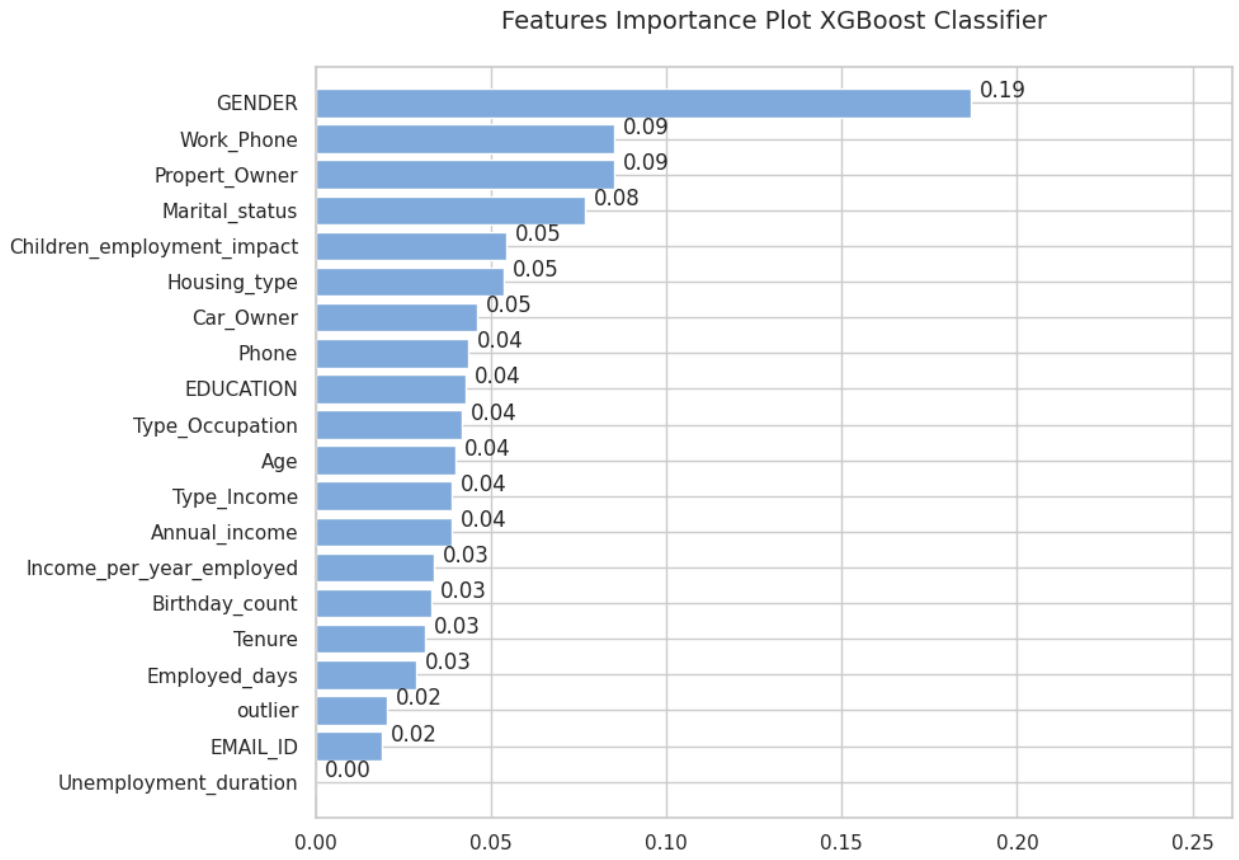





```
acc_xgb_train=round(xgb_model.score(X_train,y_train)*100,2)
acc_xgb_test=round(xgb_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_xgb_train))
print("Test Accuracy: {} %".format(acc_xgb_test))
```

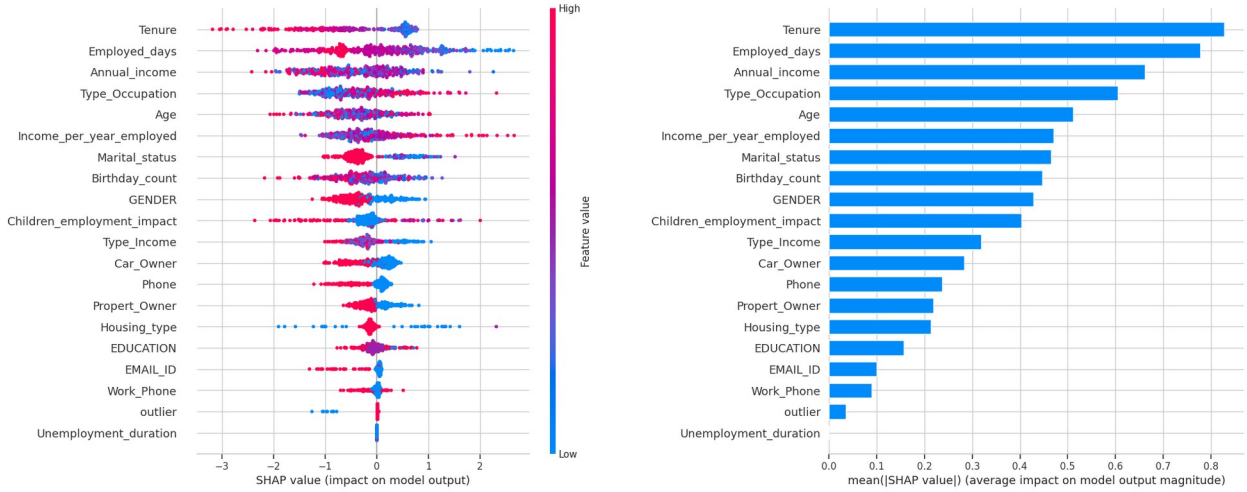
```
Training Accuracy: 99.48 %
Test Accuracy: 90.44 %
```

```
feature_importance_plot(xgb_model, "XGBoost Classifier")
```




```
shap_plot(xgb_model, "XGBoost Classifier", X_test)
```

```
<IPython.core.display.HTML object>
```




```
<shap.plots._force.AdditiveForceVisualizer at 0x78735523e4d0>
```

9. LGBM Classifier

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

```
# train the model
lgbm_model = LGBMClassifier(random_state=0).fit(X_train, y_train)
eval_classification(lgbm_model, "LGBM Classifier")
```

```
[LightGBM] [Info] Number of positive: 515, number of negative: 1030
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000558 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1821
[LightGBM] [Info] Number of data points in the train set: 1545, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.333333 ->
initscore=-0.693147
[LightGBM] [Info] Start training from score -0.693147
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000327 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1327
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000426 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 164, number of negative: 1229
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
```

```
of testing was 0.000377 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1321
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.117732 ->
initscore=-2.014090
[LightGBM] [Info] Start training from score -2.014090
[LightGBM] [Info] Number of positive: 151, number of negative: 1242
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000352 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.108399 ->
initscore=-2.107198
[LightGBM] [Info] Start training from score -2.107198
[LightGBM] [Info] Number of positive: 153, number of negative: 1240
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000383 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.109835 ->
initscore=-2.092429
[LightGBM] [Info] Start training from score -2.092429
[LightGBM] [Info] Number of positive: 156, number of negative: 1237
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000442 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111989 ->
initscore=-2.070588
[LightGBM] [Info] Start training from score -2.070588
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000421 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1322
[LightGBM] [Info] Number of data points in the train set: 1393, number
```

```
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000363 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 155, number of negative: 1239
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000346 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1324
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111191 ->
initscore=-2.078635
[LightGBM] [Info] Start training from score -2.078635
[LightGBM] [Info] Number of positive: 162, number of negative: 1232
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000417 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.116212 ->
initscore=-2.028798
[LightGBM] [Info] Start training from score -2.028798
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000343 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1327
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000420 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 164, number of negative: 1229
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000377 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1321
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.117732 ->
initscore=-2.014090
[LightGBM] [Info] Start training from score -2.014090
[LightGBM] [Info] Number of positive: 151, number of negative: 1242
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000351 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.108399 ->
initscore=-2.107198
[LightGBM] [Info] Start training from score -2.107198
[LightGBM] [Info] Number of positive: 153, number of negative: 1240
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000385 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.109835 ->
initscore=-2.092429
[LightGBM] [Info] Start training from score -2.092429
[LightGBM] [Info] Number of positive: 156, number of negative: 1237
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000859 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
```

```
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111989 ->
initscore=-2.070588
[LightGBM] [Info] Start training from score -2.070588
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000420 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1322
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000374 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 155, number of negative: 1239
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000326 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1324
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111191 ->
initscore=-2.078635
[LightGBM] [Info] Start training from score -2.078635
[LightGBM] [Info] Number of positive: 162, number of negative: 1232
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000396 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.116212 ->
initscore=-2.028798
[LightGBM] [Info] Start training from score -2.028798

<pandas.io.formats.style.Styler at 0x787361d574f0>
```

Performance of Training Model

```
model_eval_train(lgbm_model, "LGBM Classifier", X_train, y_train)

Classification Report Training Model (LGBM Classifier):

[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000347 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1327
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000431 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 164, number of negative: 1229
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000392 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1321
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.117732 ->
initscore=-2.014090
[LightGBM] [Info] Start training from score -2.014090
[LightGBM] [Info] Number of positive: 151, number of negative: 1242
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000338 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.108399 ->
initscore=-2.107198
[LightGBM] [Info] Start training from score -2.107198
```

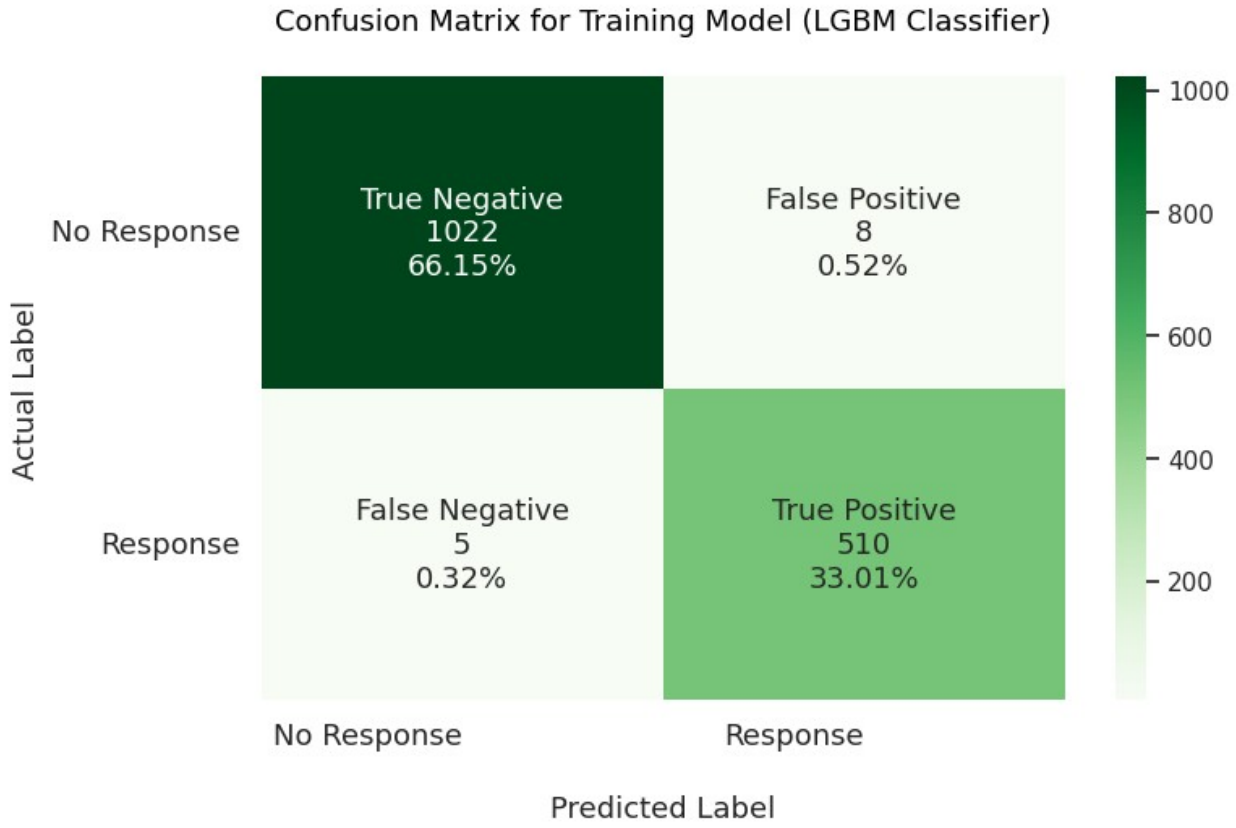
```
[LightGBM] [Info] Number of positive: 153, number of negative: 1240
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000393 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.109835 ->
initscore=-2.092429
[LightGBM] [Info] Start training from score -2.092429
[LightGBM] [Info] Number of positive: 156, number of negative: 1237
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000461 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111989 ->
initscore=-2.070588
[LightGBM] [Info] Start training from score -2.070588
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000441 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1322
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000382 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 155, number of negative: 1239
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000330 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
```

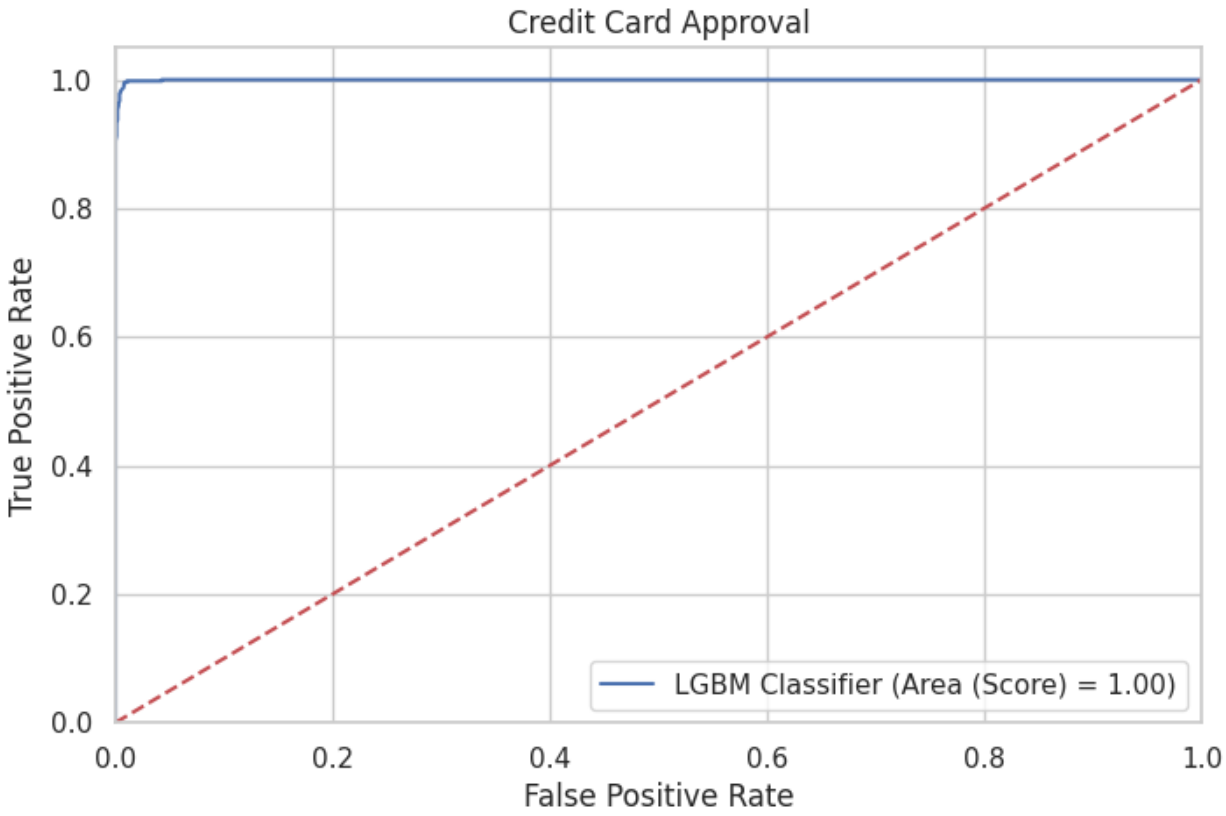
```
[LightGBM] [Info] Total Bins 1324
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111191 ->
initscore=-2.078635
[LightGBM] [Info] Start training from score -2.078635
[LightGBM] [Info] Number of positive: 162, number of negative: 1232
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000387 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.116212 ->
initscore=-2.028798
[LightGBM] [Info] Start training from score -2.028798
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000329 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1327
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000437 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 164, number of negative: 1229
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000397 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1321
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.117732 ->
initscore=-2.014090
```

```
[LightGBM] [Info] Start training from score -2.014090
[LightGBM] [Info] Number of positive: 151, number of negative: 1242
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000332 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.108399 ->
initscore=-2.107198
[LightGBM] [Info] Start training from score -2.107198
[LightGBM] [Info] Number of positive: 153, number of negative: 1240
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000928 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.109835 ->
initscore=-2.092429
[LightGBM] [Info] Start training from score -2.092429
[LightGBM] [Info] Number of positive: 156, number of negative: 1237
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000418 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111989 ->
initscore=-2.070588
[LightGBM] [Info] Start training from score -2.070588
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000421 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1322
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000389 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
```

```
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 155, number of negative: 1239
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000328 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1324
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111191 ->
initscore=-2.078635
[LightGBM] [Info] Start training from score -2.078635
[LightGBM] [Info] Number of positive: 162, number of negative: 1232
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000390 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.116212 ->
initscore=-2.028798
[LightGBM] [Info] Start training from score -2.028798
Accuracy = 0.992
Precision = 0.985
Recall = 0.99
F0.5 Score = 0.986
F1 Score = 0.987
Cross Val F1 (k=5) = 0.973
ROC AUC = 1.0
Cross Val ROC AUC (k=5) = 1.0

              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1030
           1       0.98      0.99      0.99       515

    accuracy                           0.99      1545
   macro avg       0.99      0.99      0.99      1545
weighted avg       0.99      0.99      0.99      1545

==== Actual Data (Train) =====
Total = 1545
No Response = 1030
Response = 515
```

```
==== Predicted Data (Train) =====
TP = 510, FP = 8, TN = 1022, FN = 5
Predictly Correct = 1532
Predictly Wrong = 13
```





Performance of Testing Model

```
model_eval_test(lgbm_model, "LGBM Classifier", X_test, y_test)

Classification Report Testing Model (LGBM Classifier):

[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000387 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1327
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000442 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
```

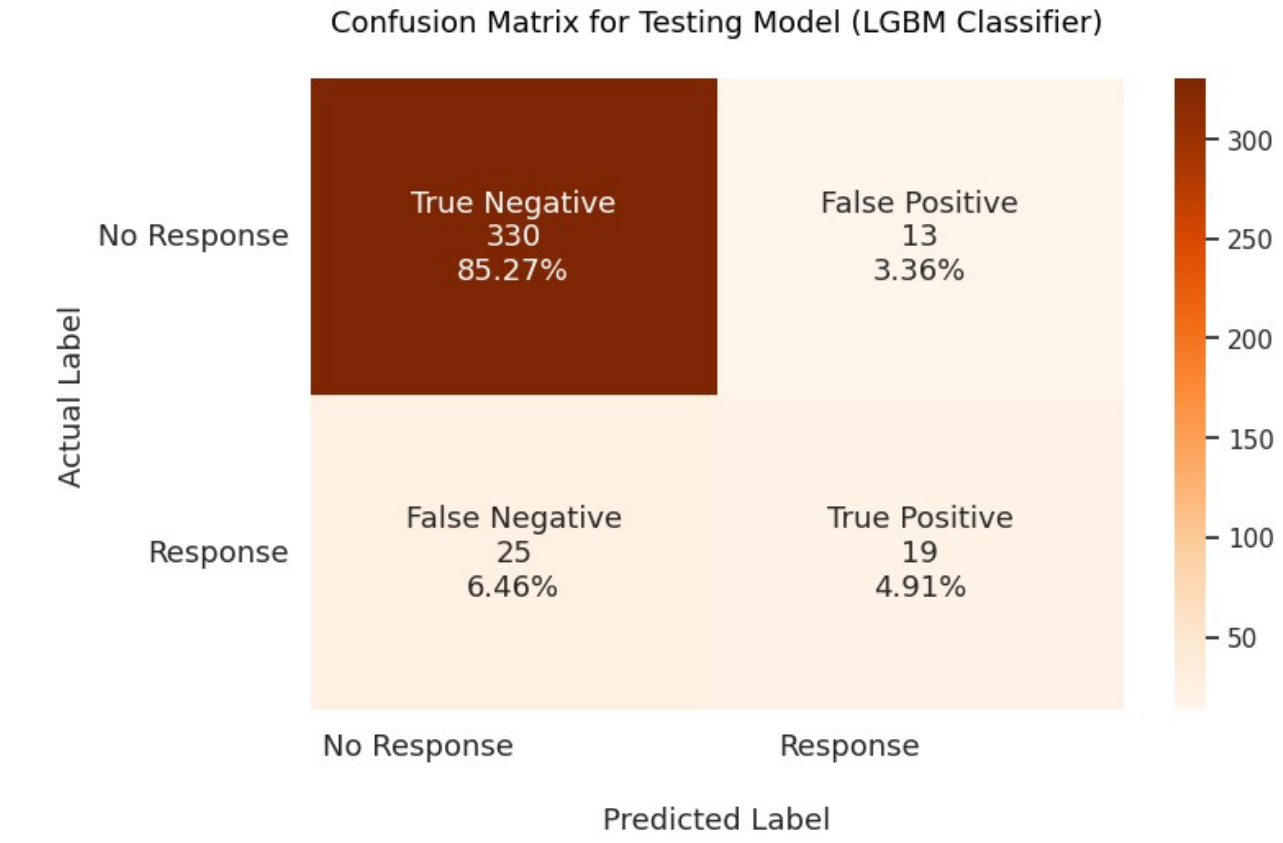
```
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 164, number of negative: 1229
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000373 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1321
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.117732 ->
initscore=-2.014090
[LightGBM] [Info] Start training from score -2.014090
[LightGBM] [Info] Number of positive: 151, number of negative: 1242
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000344 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.108399 ->
initscore=-2.107198
[LightGBM] [Info] Start training from score -2.107198
[LightGBM] [Info] Number of positive: 153, number of negative: 1240
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000390 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.109835 ->
initscore=-2.092429
[LightGBM] [Info] Start training from score -2.092429
[LightGBM] [Info] Number of positive: 156, number of negative: 1237
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000422 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111989 ->
initscore=-2.070588
[LightGBM] [Info] Start training from score -2.070588
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
```

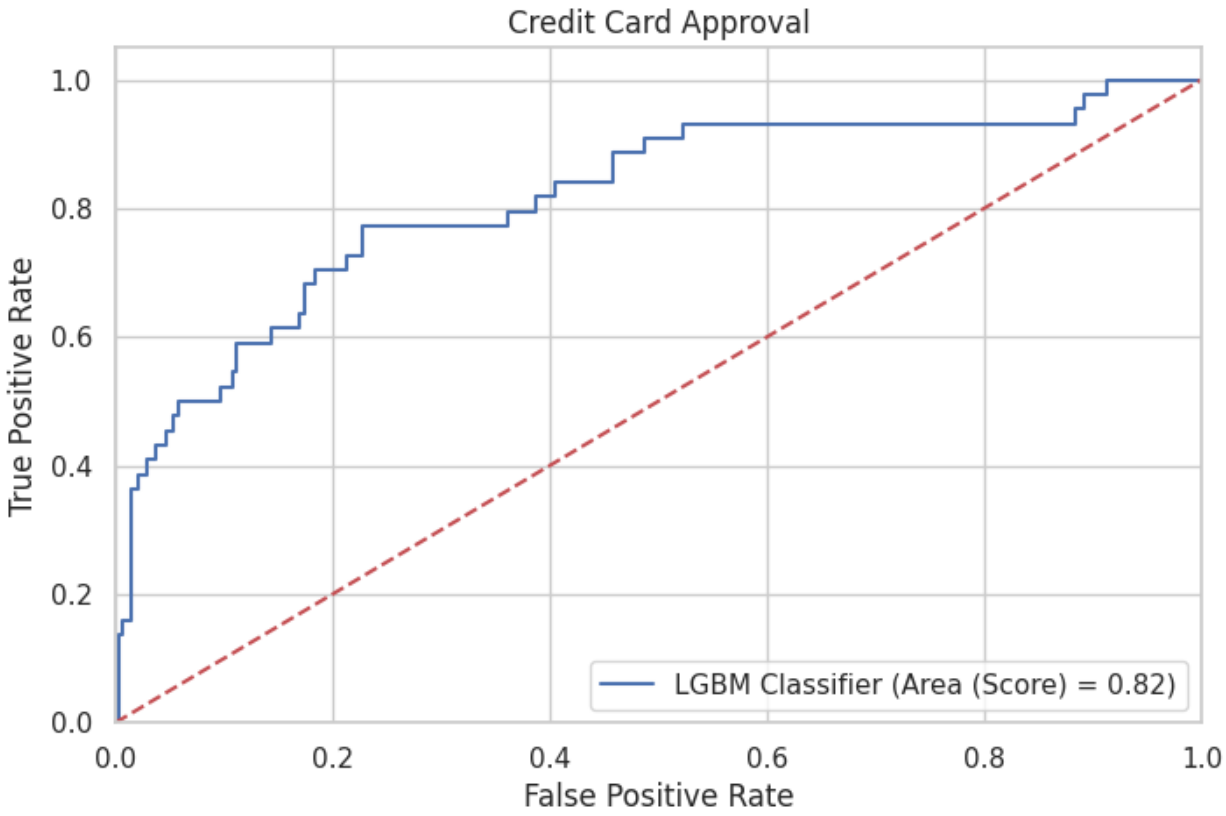
```
of testing was 0.000432 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1322
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000382 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 155, number of negative: 1239
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000325 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1324
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111191 ->
initscore=-2.078635
[LightGBM] [Info] Start training from score -2.078635
[LightGBM] [Info] Number of positive: 162, number of negative: 1232
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000382 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.116212 ->
initscore=-2.028798
[LightGBM] [Info] Start training from score -2.028798
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000324 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1327
[LightGBM] [Info] Number of data points in the train set: 1393, number
```

```
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000821 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 164, number of negative: 1229
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000395 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1321
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.117732 ->
initscore=-2.014090
[LightGBM] [Info] Start training from score -2.014090
[LightGBM] [Info] Number of positive: 151, number of negative: 1242
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000336 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.108399 ->
initscore=-2.107198
[LightGBM] [Info] Start training from score -2.107198
[LightGBM] [Info] Number of positive: 153, number of negative: 1240
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000385 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.109835 ->
initscore=-2.092429
[LightGBM] [Info] Start training from score -2.092429
[LightGBM] [Info] Number of positive: 156, number of negative: 1237
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
```

```
of testing was 0.000416 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111989 ->
initscore=-2.070588
[LightGBM] [Info] Start training from score -2.070588
[LightGBM] [Info] Number of positive: 160, number of negative: 1233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000420 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1322
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.114860 ->
initscore=-2.042032
[LightGBM] [Info] Start training from score -2.042032
[LightGBM] [Info] Number of positive: 157, number of negative: 1236
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000353 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1325
[LightGBM] [Info] Number of data points in the train set: 1393, number
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.112706 ->
initscore=-2.063390
[LightGBM] [Info] Start training from score -2.063390
[LightGBM] [Info] Number of positive: 155, number of negative: 1239
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000349 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1324
[LightGBM] [Info] Number of data points in the train set: 1394, number
of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.111191 ->
initscore=-2.078635
[LightGBM] [Info] Start training from score -2.078635
[LightGBM] [Info] Number of positive: 162, number of negative: 1232
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.000426 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1323
[LightGBM] [Info] Number of data points in the train set: 1394, number
```

```
of used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.116212 ->
initscore=-2.028798
[LightGBM] [Info] Start training from score -2.028798
Accuracy = 0.902
Precision = 0.594
Recall = 0.432
F0.5 Score = 0.552
F1 Score = 0.5
Cross Val F1 (k=5) = 0.492
ROC AUC = 0.816
Cross Val ROC AUC (k=5) = 0.79

              precision    recall  f1-score   support

           0       0.93      0.96      0.95       343
           1       0.59      0.43      0.50        44

    accuracy                           0.90       387
   macro avg       0.76      0.70      0.72       387
weighted avg       0.89      0.90      0.89       387

==== Actual Data (Test) =====
Total = 387
No Response = 343
Response = 44
==== Predicted Data (Test) =====
TP = 19, FP = 13, TN = 330, FN = 25
Predictly Correct = 349
Predictly Wrong = 38
```

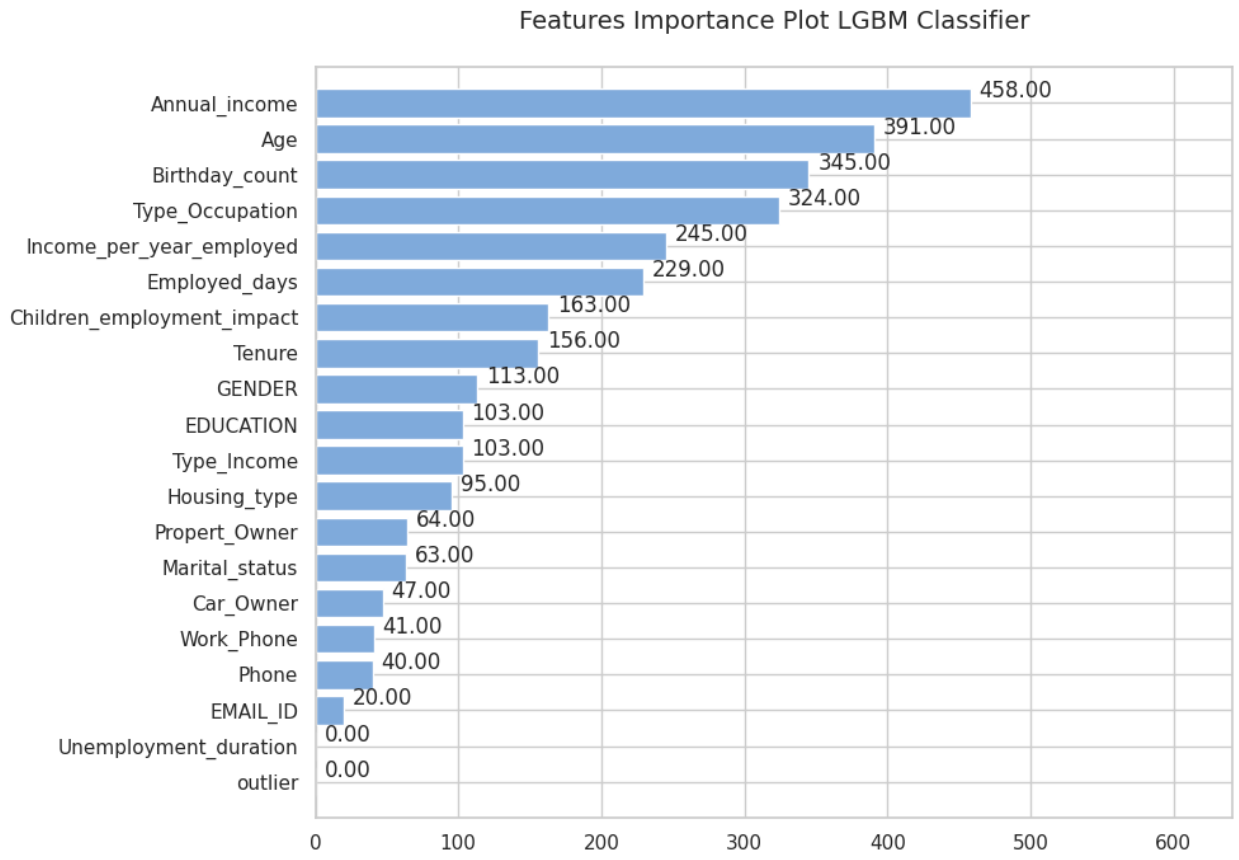

Confusion Matrix for Testing Model (LGBM Classifier)
True Negative
330
85.27%
False Positive
13
3.36%
False Negative
25
6.46%
True Positive
19
4.91%
No Response
Response
Actual Label
No Response
Response
Predicted Label
300
250
200
150
100
50




```
acc_lgbm_train=round(lgbm_model.score(X_train,y_train)*100,2)
acc_lgbm_test=round(lgbm_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_lgbm_train))
print("Test Accuracy: {} %".format(acc_lgbm_test))
```

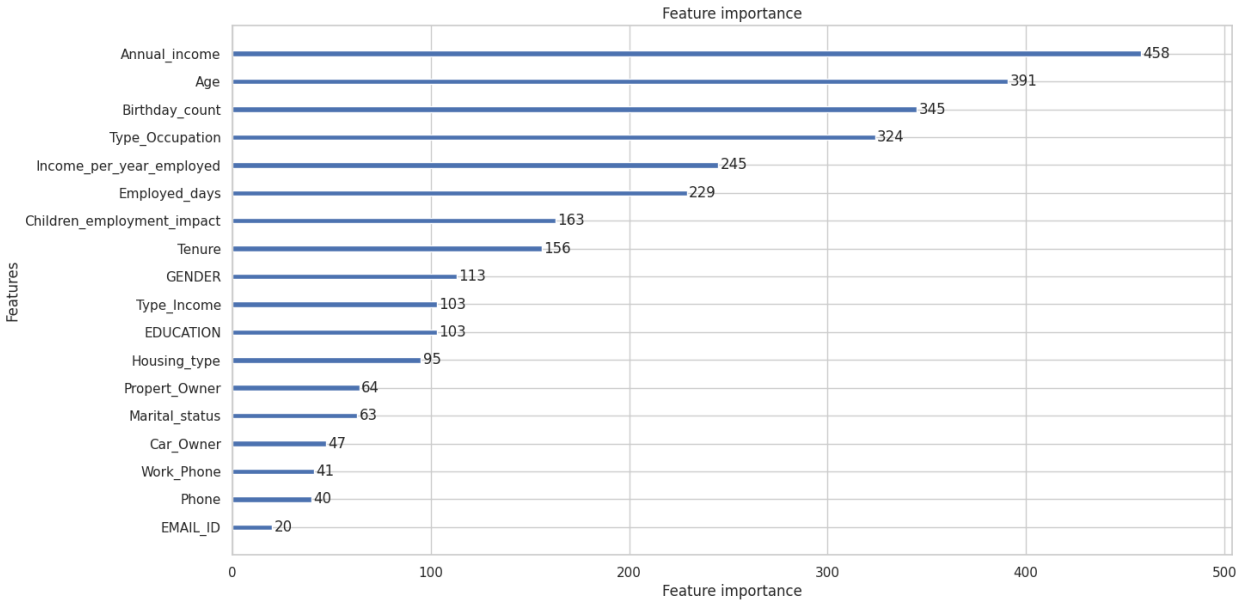
```
Training Accuracy: 99.16 %
Test Accuracy: 90.18 %
```

```
feature_importance_plot(lgbm_model, "LGBM Classifier")
```




```
lgb.plot_importance(lgbm_model)
```

```
<Axes: title={'center': 'Feature importance'}, xlabel='Feature
importance', ylabel='Features'>
```



```
shap_plot(lgbm_model, "LGBM Classifier", X_test)

lgb.plot_tree(lgbm_model,figsize=(30,40))
```

10. Gradient Boosting Classifier

Gradient boosting merupakan algoritma klasifikasi machine learning yang menggunakan ensamble dari decision tree untuk memprediksi nilai. Gradient boosting termasuk supervised learning berbasis decision tree yang dapat digunakan untuk klasifikasi

```
# train the model
# gb_model = GradientBoostingClassifier().fit(X_train, y_train)
gb_model = GradientBoostingClassifier(learning_rate=0.01,
n_estimators=400, max_depth=13, random_state=42).fit(X_train, y_train)
eval_classification(gb_model, "Gradient Boosting Classifier")
```

Performance of Training Model

```
model_eval_train(gb_model, "Gradient Boosting Classifier", X_train,
y_train)
```

Performance of Testing Model

```
model_eval_test(gb_model, "Gradient Boosting Classifier", X_test,
y_test)

acc_gb_train=round(gb_model.score(X_train,y_train)*100,2)
acc_gb_test=round(gb_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_gb_train))
print("Test Accuracy: {} %".format(acc_gb_test))

feature_importance_plot(gb_model, "Gradient Boosting Classifier")

shap_plot(gb_model, "Gradient Boosting Classifier", X_test)
```

11. Support Vector Machine

Support Vector Machine atau SVM adalah algoritme pembelajaran mesin yang diawasi yang dapat digunakan untuk klasifikasi dan regresi. Cara kerja SVM didasarkan pada SRM atau Structural Risk Minimization yang dirancang untuk mengolah data menjadi Hyperplane yang mengklasifikasikan ruang input menjadi dua kelas.

```
# train the model
svm_model = SVC(kernel='rbf', probability=True,
random_state=42).fit(X_train, y_train)
print(svm_model)
eval_classification(svm_model, "Support Vector Machine")

SVC(probability=True, random_state=42)
```

```
<pandas.io.formats.style.Styler at 0x7872fe7173a0>
```

Performance of Training Model

```
model_eval_train(svm_model, "Support Vector Machine", X_train,
y_train)
```

Performance of Testing Model

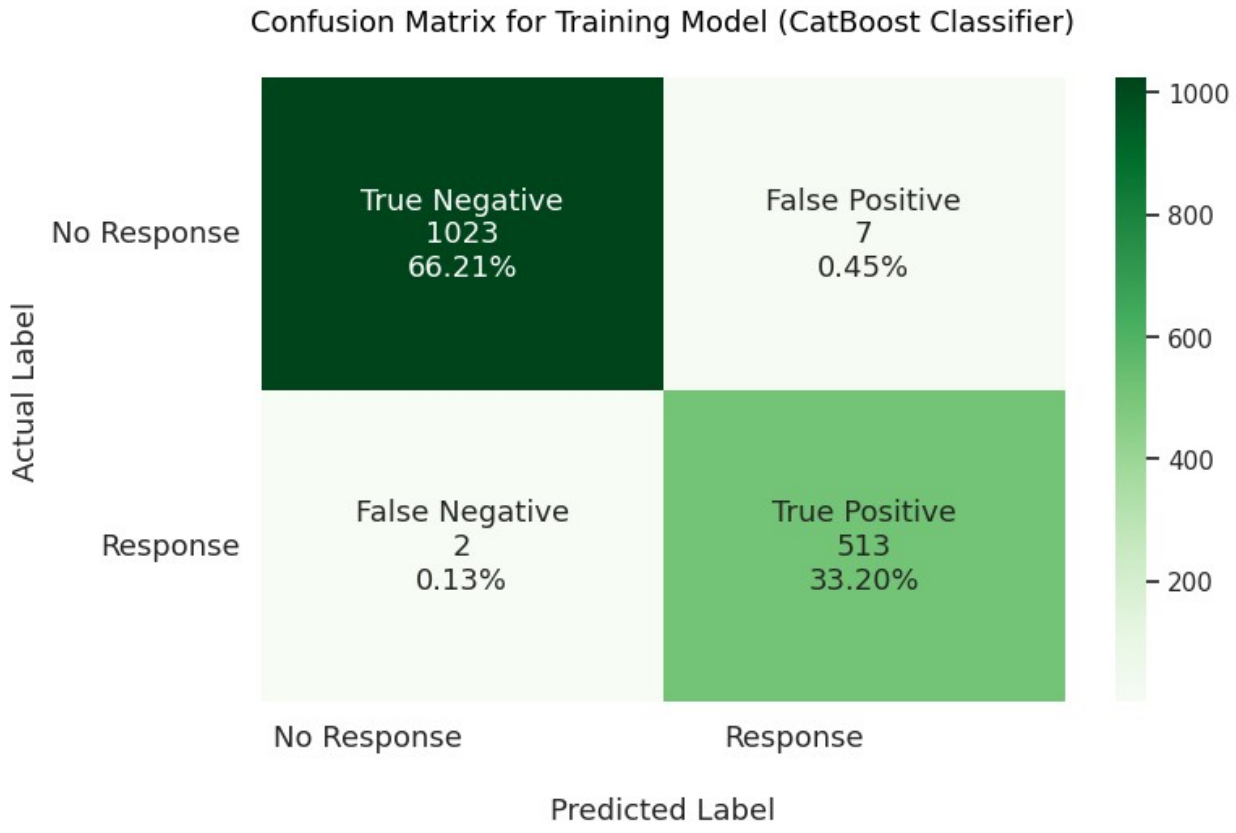
```
model_eval_test(svm_model, "Support Vector Machine", X_test, y_test)

acc_svm_train=round(svm_model.score(X_train,y_train)*100,2)
acc_svm_test=round(svm_model.score(X_test,y_test)*100,2)
print("Training Accuracy: {} %".format(acc_svm_train))
print("Test Accuracy: {} %".format(acc_svm_test))

feature_importance_plot(svm_model, "Support Vector Machine", X_train)

# shap_plot(svm_model, "Support Vector Machine", X_test)
```

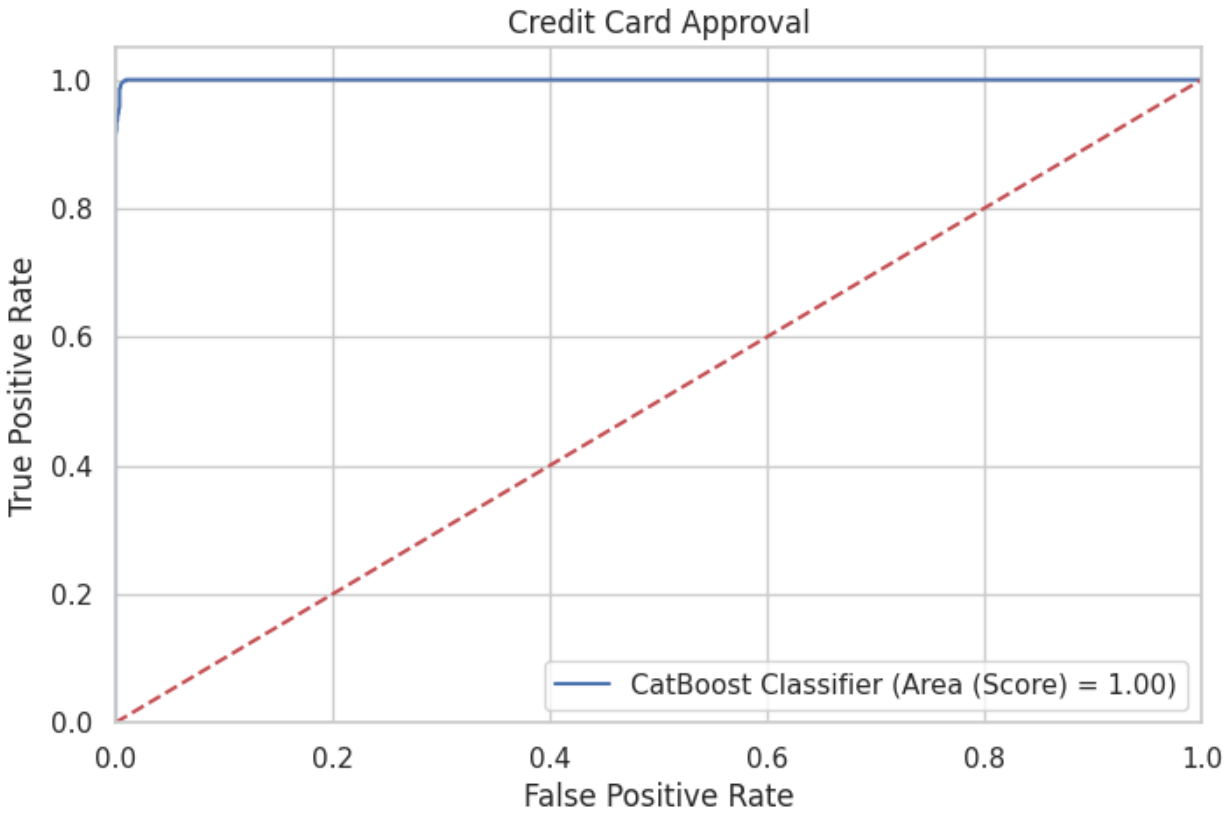
12. CatBoost Classification

CatBoost adalah algoritme pembelajaran mesin berbasis boosting yang dirancang khusus untuk menangani data yang mengandung variabel kategorikal dengan efisien. CatBoost bekerja dengan menciptakan serangkaian model pohon keputusan yang digabungkan secara bertahap untuk meningkatkan akurasi prediksi. Algoritme ini unggul dalam hal kecepatan pelatihan, performa prediksi yang tinggi, dan kemampuannya menangani variabel kategorikal secara langsung tanpa perlu encoding yang rumit.

```
# Train the model
catboost_model = CatBoostClassifier(
    iterations=1000,
    learning_rate=0.03,
    depth=6,
    random_state=42,
    verbose=0
).fit(X_train, y_train)

print(catboost_model)
eval_classification(catboost_model, "CatBoost Classifier")

<catboost.core.CatBoostClassifier object at 0x787358a8b010>

<pandas.io.formats.style.Styler at 0x787358b009d0>

model_eval_train(catboost_model, "CatBoost Classifier", X_train,
y_train)

Classification Report Training Model (CatBoost Classifier):

Accuracy = 0.994
```

```
Precision = 0.987
Recall = 0.996
F0.5 Score = 0.988
F1 Score = 0.991
Cross Val F1 (k=5) = 0.956
ROC AUC = 1.0
Cross Val ROC AUC (k=5) = 1.0

              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1030
           1       0.99      1.00      0.99       515

    accuracy                           0.99      1545
   macro avg       0.99      0.99      0.99      1545
weighted avg       0.99      0.99      0.99      1545

==== Actual Data (Train) =====
Total = 1545
No Response = 1030
Response = 515
==== Predicted Data (Train) =====
TP = 513, FP = 7, TN = 1023, FN = 2
Predictly Correct = 1536
Predictly Wrong = 9
```


Confusion Matrix for Training Model (CatBoost Classifier)
True Negative
1023
66.21%
False Positive
7
0.45%
False Negative
2
0.13%
True Positive
513
33.20%
No Response
Response
Actual Label
No Response
Response
Predicted Label
1000
800
600
400
200



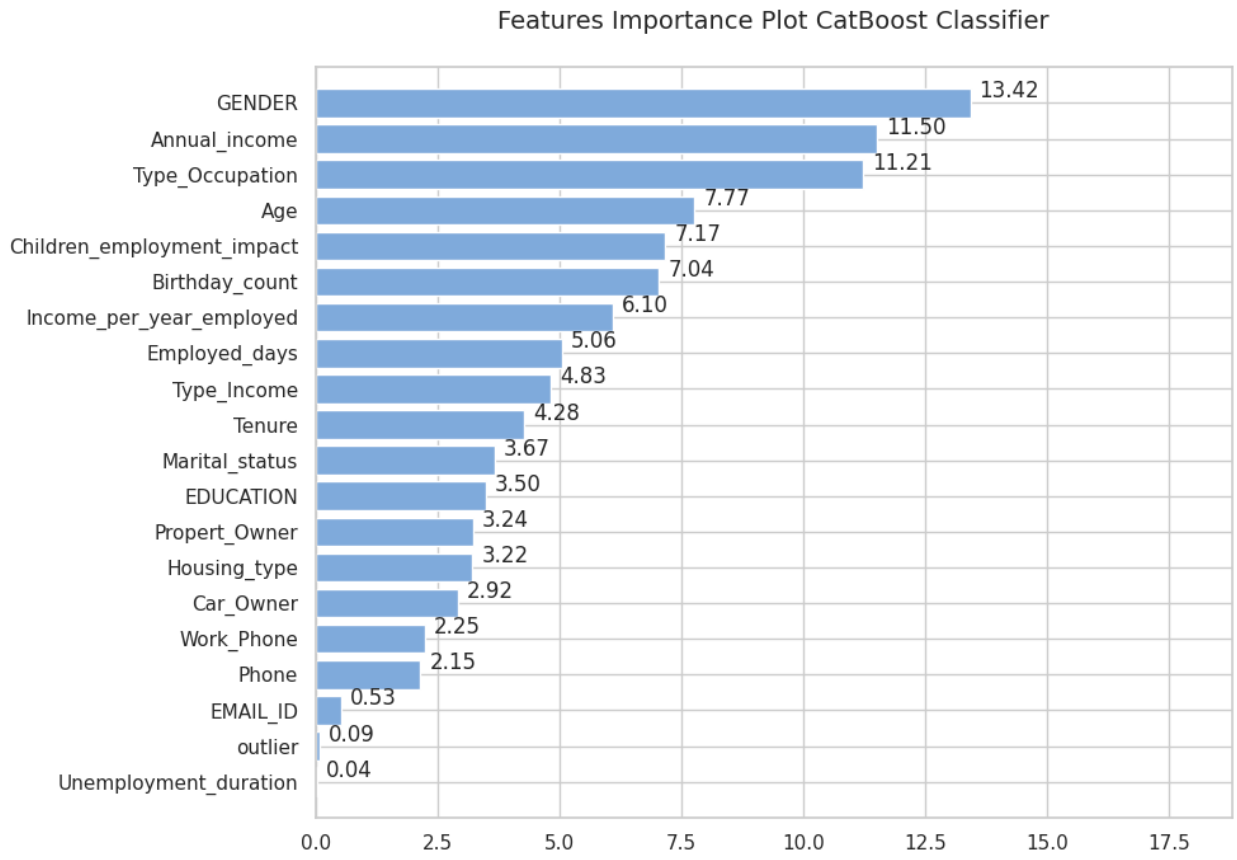

```
model_eval_test(catboost_model, "CatBoost Classifier", X_test, y_test)
```

```
Classification Report Testing Model (CatBoost Classifier):
```

```
acc_catboost_train = round(catboost_model.score(X_train, y_train) *
100, 2)
acc_catboost_test = round(catboost_model.score(X_test, y_test) * 100,
2)
print("Training Accuracy: {} %".format(acc_catboost_train))
print("Test Accuracy: {} %".format(acc_catboost_test))
```

```
Training Accuracy: 99.42 %
Test Accuracy: 91.47 %
```

```
feature_importance_plot(catboost_model, "CatBoost Classifier",
X_train)
```




```
shap_plot(catboost_model, "CatBoost Classifier", X_test)
```

Summary

```
results_eval = pd.DataFrame({
    "Models" : train_modelname_list,
    "Precision (Train)": train_precision_list,
    "Precision (Test)": test_precision_list,
    "Recall (Train)": train_recall_list,
    "Recall (Test)": test_recall_list,
    "F0.5 Score (Train)" : train_fbeta_score_list,
    "F0.5 Score (Test)" : test_fbeta_score_list,
    "F1 Score (Train)" : train_f1_score_list,
    "F1 Score (Test)" : test_f1_score_list
})

results_eval.drop_duplicates(inplace = True)

results_eval.sort_values(by=["F0.5 Score (Test)", "Precision (Test)",
"Recall (Test)"], ascending=[False, False, False]).reset_index(drop =
True).style.format(precision=3).background_gradient(cmap="Purples")
```

```
<pandas.io.formats.style.Styler at 0x78735e482410>
```

```
import pickle
pickle.dump(rf_model, open('rf_best_model.pkl', 'wb'))

rf_best_model_temp = pickle.load(open('rf_best_model.pkl', 'rb'))
```

Hyperparameter Tuning

Kemudian, pada tahap selanjutnya kami melakukan hyperparameter tuning dari beberapa model. Kemudian akan fokus ke beberapa model yang telah kami pilih berdasarkan nilai precision, recall dan f1 score tertinggi sebelumnya. Hal ini kami lakukan dengan menganalisa output learning curve agar output akhir modelling dari jenis metode tersebut dapat memberikan hasil yang lebih optimal.

```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import StratifiedKFold

def show_best_hyperparameter(model, hyperparameters):
    for key, value in hyperparameters.items() :
        print('Best '+key+':', model.best_params_()[key])

from sklearn.metrics import make_scorer
precision_scorer = make_scorer(precision_score, zero_division=0)
custom_scoring = {"precision": precision_scorer, "recall": "recall",
"f1": "f1"}
```

1. Decision Tree

Tuning CV

```
# # List of hyperparameter
# max_depth = [int(x) for x in np.linspace(1, 110, num = 30)] #
Maximum number of levels in tree
# min_samples_split = [2, 5, 10, 100] # Minimum number of samples
required to split a node
# min_samples_leaf = [1, 2, 4, 10, 20, 50] # Minimum number of samples
required at each leaf node
# max_features = [None, 'sqrt'] # Number of features to consider at
every split
# criterion = ['gini','entropy']
# splitter = ['best','random']

# hyperparameters = dict(max_depth=max_depth,
#                        min_samples_split=min_samples_split,
#                        min_samples_leaf=min_samples_leaf,
#                        max_features=max_features,
#                        criterion=criterion,
#                        splitter=splitter)

# dt = DecisionTreeClassifier(random_state=42)
```

```
# cv = StratifiedKFold(n_splits=5)

# dt_model_ht = RandomizedSearchCV(dt, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # dt_model_ht = GridSearchCV(dt, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = dt_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# dt_model_ht = DecisionTreeClassifier(splitter = 'random',
min_samples_split = 2,
#                                      min_samples_leaf = 1,
max_features = None,
#                                      max_depth = 53, criterion =
'entropy', random_state=42)
# dt_model_ht.fit(X_train,y_train)

# eval_classification(dt_model_ht, "Decision Tree (HT)")
```

Learning Curve

- **min_samples_leaf**

```
# def draw_learning_curve(params_values):
#     train_scores = []
#     test_scores = []

#     for i in params_values:
#         model = DecisionTreeClassifier(random_state = 42,
min_samples_leaf=i)
#         model.fit(X_train, y_train)

#         #Eval on Train
#         y_pred_train_proba = model.predict_proba(X_train)
#         train_auc = roc_auc_score(y_train, y_pred_train_proba[:, 1])
#         train_scores.append(train_auc)

#         #Eval on Test
#         y_pred_proba = model.predict_proba(X_test)
#         test_auc = roc_auc_score(y_test, y_pred_proba[:, 1])
#         test_scores.append(test_auc)

#         # print('params values: ' + str(i) + '; train: ' +
```

```
str(train_auc) + '; test: ' + str(test_auc))

#     fig = plt.figure(figsize =(10, 4))
#     plt.plot(params_values, train_scores, label='Train',
linewidth=2.5, marker ='.', markersize=10)
#     plt.plot(params_values, test_scores, label='Test',
linewidth=2.5, marker ='.', markersize=10)
#     plt.xlabel('min_samples_leaf')
#     plt.ylabel('AUC')
#     plt.title('Learning Curve')
#     plt.legend()
#     plt.show()

# params_values = [int(x) for x in np.linspace(95, 110, 15)]
# draw_learning_curve(params_values)
```

- **max_depth**

```
# def draw_learning_curv(params_values):
#     train_scores = []
#     test_scores = []

#     for c in params_values:
#         model = DecisionTreeClassifier(random_state = 42,
min_samples_leaf = 95, max_depth=c)
#         model.fit(X_train, y_train)

#         #Eval on Train
#         y_pred_train_proba = model.predict_proba(X_train)
#         train_auc = roc_auc_score(y_train, y_pred_train_proba[:, 1])
#         train_scores.append(train_auc)

#         #Eval on Test
#         y_pred_proba = model.predict_proba(X_test)
#         test_auc = roc_auc_score(y_test, y_pred_proba[:, 1])
#         test_scores.append(test_auc)

#         # print('params values: ' + str(c) + '; train: ' +
str(train_auc) + '; test: ' + str(test_auc))

#     fig = plt.figure(figsize =(10, 4))
#     plt.plot(params_values, train_scores, label='Train',
linewidth=2.5, marker ='.', markersize=10)
#     plt.plot(params_values, test_scores, label='Test',
linewidth=2.5, marker ='.', markersize=10)
#     plt.xlabel('max_depth')
#     plt.ylabel('AUC')
#     plt.title('Learning Curve')
#     plt.legend()
#     plt.show()
```

```
# params_values = [int(x) for x in np.linspace(1, 10, num=10)]
# draw_learning_curv(params_values)
```

- **min_sample_split**

```
# def draw_learning_curves(params_values):
#     train_scores = []
#     test_scores = []

#     for i in params_values:
#         model = DecisionTreeClassifier(random_state = 42,
min_samples_leaf = 95, max_depth=4, min_samples_split=i)
#         model.fit(X_train, y_train)

#         #Eval on Train
#         y_pred_train_proba = model.predict_proba(X_train)
#         train_auc = roc_auc_score(y_train, y_pred_train_proba[:, 1])
#         train_scores.append(train_auc)

#         #Eval on Test
#         y_pred_proba = model.predict_proba(X_test)
#         test_auc = roc_auc_score(y_test, y_pred_proba[:, 1])
#         test_scores.append(test_auc)

#         # print('param values: ' + str(i) + '; train: ' +
str(train_auc) + '; test: ' + str(test_auc))
#     fig = plt.figure(figsize =(10, 4))
#     plt.plot(params_values, train_scores, label='Train',
linewidth=2.5, marker ='.', markersize=10)
#     plt.plot(params_values, test_scores, label='Test',
linewidth=2.5, marker ='.', markersize=10)
#     plt.xlabel('min_samples_split')
#     plt.ylabel('AUC')
#     plt.title('Learning Curve')
#     plt.legend()
#     plt.show()

# params_values = [int(x) for x in np.linspace(2, 160, 160)]
# draw_learning_curves(params_values)
```

Manually Tuning

Melakukan tuning kembali menggunakan hyperparameter best/optimal sebelumnya untuk melihat perubahan evaluasi model

```
# from sklearn.tree import DecisionTreeClassifier
# dt_model_ht2 = DecisionTreeClassifier(random_state = 42,
min_samples_leaf = 95,
#                               max_depth=4, min_samples_split=20,
```

```
#                               splitter = 'best', criterion = 'gini')
# dt_model_ht2.fit(X_train, y_train)

# eval_classification(dt_model_ht2, "Decision Tree (HT2)")
```

2. Random Forest

Tuning CV

```
# # List of hyperparameter
# n_estimators = [int(x) for x in np.linspace(1, 200, 50)]
# criterion = ['gini', 'entropy']
# max_depth = [int(x) for x in np.linspace(2, 100, 50)]
# min_samples_split = [int(x) for x in np.linspace(2, 20, 10)]
# min_samples_leaf = [int(x) for x in np.linspace(2, 20, 10)]
# hyperparameters = dict(n_estimators=n_estimators,
criterion=criterion, max_depth=max_depth,
#                        min_samples_split=min_samples_split,
min_samples_leaf=min_samples_leaf)

# rf = RandomForestClassifier(random_state=42)
# cv = StratifiedKFold(n_splits=5)

# rf_model_ht = RandomizedSearchCV(rf, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # rf_model_ht = GridSearchCV(rf, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = rf_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param

# rf_model_ht1 = RandomForestClassifier(max_depth = 15,
min_samples_leaf = 1, min_samples_split = 2, n_estimators = 300,
random_state=42)
# rf_model_ht1.fit(X_train,y_train)

# eval_classification(rf_model_ht1, "Random Forest (HT1)")

# rf_model_ht2 = RandomForestClassifier(n_estimators = 98,
min_samples_split = 18,
#                                       min_samples_leaf = 4, max_depth
= 90,
#                                       criterion = 'entropy',
```

```
random_state=42)
# rf_model_ht2.fit(X_train,y_train)

# eval_classification(rf_model_ht2, "Random Forest (HT2)")
```

Learning Curve

- **n_estimators**

```
# param_values = [int(x) for x in np.linspace(1, 200, 10)] #
n_estimators

# train_scores = []
# test_scores = []

# for c in param_values:
#     model = RandomForestClassifier(random_state=42, n_estimators=c)
#cek param n estimator
#     model.fit(X_train, y_train)

#     # eval on train
#     y_pred_train_proba = model.predict_proba(X_train)
#     train_auc = roc_auc_score(y_train, y_pred_train_proba[:,1])
#     train_scores.append(train_auc)

#     # eval on test
#     y_pred_proba = model.predict_proba(X_test)
#     test_auc = roc_auc_score(y_test, y_pred_proba[:,1])
#     test_scores.append(test_auc)

#     # print('param value: ' + str(c) + '; train: ' + str(train_auc)
+ '; test: '+ str(test_auc))

# fig = plt.figure(figsize =(10, 4))
# plt.plot(param_values, train_scores, label='Train', linewidth=2.5,
marker ='.', markersize=10)
# plt.plot(param_values, test_scores, label='Test', linewidth=2.5,
marker ='.', markersize=10)

# plt.legend()
# plt.show()
```

- **max_depth**

```
# param_values = [int(x) for x in np.linspace(2, 20, 10)] # max_depth

# train_scores = []
# test_scores = []

# for c in param_values:
#     model = RandomForestClassifier(random_state=42, n_estimators=
```

```
23, max_depth = c) # cek param max_depth
#     model.fit(X_train, y_train)

#     # eval on train
#     y_pred_train_proba = model.predict_proba(X_train)
#     train_auc = roc_auc_score(y_train, y_pred_train_proba[:,1])
#     train_scores.append(train_auc)

#     # eval on test
#     y_pred_proba = model.predict_proba(X_test)
#     test_auc = roc_auc_score(y_test, y_pred_proba[:,1])
#     test_scores.append(test_auc)

#     # print('param value: ' + str(c) + '; train: ' + str(train_auc)
+ '; test: '+ str(test_auc))

# fig = plt.figure(figsize =(10, 4))
# plt.plot(param_values, train_scores, label='Train', linewidth=2.5,
marker ='.', markersize=10)
# plt.plot(param_values, test_scores, label='Test', linewidth=2.5,
marker ='.', markersize=10)

# plt.legend()
# plt.show()
```

- **min_samples_split**

```
# param_values = [int(x) for x in np.linspace(2, 20, 15)] #
min_samples_split

# train_scores = []
# test_scores = []

# for c in param_values:
#     model = RandomForestClassifier(random_state=42, n_estimators=
23, max_depth = 4, min_samples_split = c) # cek param min_sample_split
#     model.fit(X_train, y_train)

#     # eval on train
#     y_pred_train_proba = model.predict_proba(X_train)
#     train_auc = roc_auc_score(y_train, y_pred_train_proba[:,1])
#     train_scores.append(train_auc)

#     # eval on test
#     y_pred_proba = model.predict_proba(X_test)
#     test_auc = roc_auc_score(y_test, y_pred_proba[:,1])
#     test_scores.append(test_auc)

#     # print('param value: ' + str(c) + '; train: ' + str(train_auc)
+ '; test: '+ str(test_auc))
```

```
# fig = plt.figure(figsize =(10, 4))
# plt.plot(param_values, train_scores, label='Train', linewidth=2.5,
marker ='.', markersize=10)
# plt.plot(param_values, test_scores, label='Test', linewidth=2.5,
marker ='.', markersize=10)

# plt.legend()
# plt.show()
```

- **min_samples_leaf**

```
# param_values = [int(x) for x in np.linspace(2, 20, 15)] #
min_samples_leaf

# train_scores = []
# test_scores = []

# for c in param_values:
#     model = RandomForestClassifier(random_state=42, n_estimators=
23, max_depth = 4, min_samples_split = 3, min_samples_leaf = c) # cek
param min_sample_leaf
#     model.fit(X_train, y_train)

#     # eval on train
#     y_pred_train_proba = model.predict_proba(X_train)
#     train_auc = roc_auc_score(y_train, y_pred_train_proba[:,1])
#     train_scores.append(train_auc)

#     # eval on test
#     y_pred_proba = model.predict_proba(X_test)
#     test_auc = roc_auc_score(y_test, y_pred_proba[:,1])
#     test_scores.append(test_auc)

#     # print('param value: ' + str(c) + '; train: ' + str(train_auc)
+ '; test: '+ str(test_auc))

# fig = plt.figure(figsize =(10, 4))
# plt.plot(param_values, train_scores, label='Train', linewidth=2.5,
marker ='.', markersize=10)
# plt.plot(param_values, test_scores, label='Test', linewidth=2.5,
marker ='.', markersize=10)

# plt.legend()
# plt.show()
```

Manually Tuning

Melakukan tuning kembali menggunakan hyperparameter best/optimal sebelumnya untuk melihat perubahan evaluasi model

```
# n_estimators = [23]
# criterion = ['gini', 'entropy']
# max_depth = [4]
# min_samples_split = [3]
# min_samples_leaf = [4]

# hyperparameters = dict(n_estimators=n_estimators,
criterion=criterion, max_depth=max_depth,
#                        min_samples_split=min_samples_split,
min_samples_leaf=min_samples_leaf)

# rf = RandomForestClassifier(random_state=42)
# rf_model_ht3 = RandomizedSearchCV(rf, hyperparameters,
scoring='precision', random_state=42, cv=5)
# rf_model_ht3.fit(X_train, y_train)

# print("Best: %f using %s" % (rf_model_ht3.best_score_,
rf_model_ht3.best_params_))

# rf_model_ht3 = RandomForestClassifier(n_estimators = 23,
min_samples_split = 13,
#                                       min_samples_leaf = 11,
max_depth = 4,
#                                       criterion = 'gini',
random_state = 42)
# rf_model_ht3.fit(X_train, y_train)

# eval_classification(rf_model_ht3, "Random Forest (HT3)")
```

3. Logistic Regression

Tuning CV

```
# # List of hyperparameter
# penalty = ['l1', 'l2']
# C = [float(x) for x in np.linspace(0.0001, 1, 100)]
# hyperparameters = dict(penalty=penalty, C=C)

# logreg = LogisticRegression(random_state=42)
# cv = StratifiedKFold(n_splits=5)

# logreg_model_ht = RandomizedSearchCV(logreg, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # logreg_model_ht = GridSearchCV(logreg, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = logreg_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
```

```
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# logreg_model_ht = LogisticRegression(penalty = 'l2', C =
0.47479999999999994, random_state=42)
# logreg_model_ht.fit(X_train,y_train)

# eval_classification(logreg_model_ht, "Logistic Regression (HT)")
```

4. Naive Bayes

Tuning CV

```
# # List of hyperparameter
# var_smoothing = np.logspace(0,-9, num=100)
# hyperparameters = dict(var_smoothing=var_smoothing)

# gnb = GaussianNB()
# cv = StratifiedKFold(n_splits=5)

# # gnb_model_ht = RandomizedSearchCV(gnb, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# gnb_model_ht = GridSearchCV(gnb, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = gnb_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# gnb_model_ht = GaussianNB(var_smoothing=0.0657933224657568)
# gnb_model_ht.fit(X_train,y_train)

# eval_classification(gnb_model_ht, "Naive Bayes (HT)")
```

5. K-Nearest Neighbors

Tuning CV

```
# # List of hyperparameter
# leaf_size = list(range(1,100))
# n_neighbors = list(range(1,100))
# p=[1,2,3]
```

```
# algorithm = ['auto', 'ball_tree', 'kd_tree', 'brute']
# hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors,
p=p, algorithm=algorithm)

# knn = KNeighborsClassifier()
# cv = StratifiedKFold(n_splits=5)

# knn_model_ht = RandomizedSearchCV(knn, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # knn_model_ht = GridSearchCV(knn, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = knn_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# knn_model_ht = KNeighborsClassifier(p = 2, n_neighbors = 13,
leaf_size = 34, algorithm = 'brute')
# knn_model_ht.fit(X_train,y_train)

# eval_classification(knn_model_ht, "K-Nearest Neighbors (HT)")
```

6. MLP Classifier

Tuning CV

```
# # List of hyperparameter
# hyperparameters = {
#     'hidden_layer_sizes': [(10,30,10),(20,)],
#     'activation': ['tanh', 'relu'],
#     'solver': ['sgd', 'adam'],
#     'alpha': [0.0001, 0.05],
#     'learning_rate': ['constant','adaptive'],
# }

# mlp = MLPClassifier(random_state=42, max_iter=len(X_train))
# cv = StratifiedKFold(n_splits=5)

# mlp_model_ht = RandomizedSearchCV(mlp, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # mlp_model_ht = GridSearchCV(mlp, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = mlp_model_ht.fit(X_train, y_train)
```

```
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# mlp_model_ht = MLPClassifier(random_state=42, max_iter=len(X_train),
solver = 'adam', learning_rate = 'adaptive', hidden_layer_sizes=
(20,), alpha = 0.0001, activation = 'relu')
# mlp_model_ht.fit(X_train,y_train)

# eval_classification(mlp_model_ht, "MLP Classifier (HT)")
```

7. Adaboost Classifier

Tuning CV

```
# # List of hyperparameter
# hyperparameters = dict(n_estimators = [int(x) for x in
np.linspace(start = 50, stop = 2000, num = 2000)], # Jumlah iterasi
#                        learning_rate = [float(x) for x in
np.linspace(start = 0.001, stop = 0.1, num = 200)],
#                        algorithm = ['SAMME', 'SAMME.R']
#                       )

# adab = AdaBoostClassifier(random_state=42)
# cv = StratifiedKFold(n_splits=5)

# adab_model_ht = RandomizedSearchCV(adab, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # adab_model_ht = GridSearchCV(adab, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = adab_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# adab_model_htt = AdaBoostClassifier(n_estimators = 1634,
learning_rate = 0.09801005025125628, algorithm = 'SAMME.R',
random_state=42)
# adab_model_htt.fit(X_train,y_train)
```

```
# eval_classification(adab_model_ht, "Adaboost Classifier (HT)")
```

8. XGBoost Classifier

Tuning CV

```
# # List of hyperparameter
# #Menjadikan ke dalam bentuk dictionary
# hyperparameters = {
#                     'max_depth' : [int(x) for x in np.linspace(10,
110, num = 11)],
#                     'min_child_weight' : [int(x) for x in
np.linspace(1, 20, num = 11)],
#                     'gamma' : [float(x) for x in np.linspace(0, 1,
num = 11)],
#                     'tree_method' : ['auto', 'exact', 'approx',
'hist'],

#                     'colsample_bytree' : [float(x) for x in
np.linspace(0, 1, num = 11)],
#                     'eta' : [float(x) for x in np.linspace(0, 1, num
= 100)],

#                     'lambda' : [float(x) for x in np.linspace(0, 1,
num = 11)],
#                     'alpha' : [float(x) for x in np.linspace(0, 1,
num = 11)]
#                     }

# xgb = XGBClassifier(random_state=42)
# cv = StratifiedKFold(n_splits=5)

# xgb_model_ht = RandomizedSearchCV(xgb, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # xgb_model_ht = GridSearchCV(xgb, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = xgb_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# xgb_model_ht = XGBClassifier(tree_method = 'hist', min_child_weight
= 6, max_depth = 10,
#                              reg_lambda = 0.1, gamma = 0.1, eta =
```

```
0.14141414141414144,
#                                colsample_bytree = 0.2, alpha = 0.9)
# xgb_model_ht.fit(X_train,y_train)

# eval_classification(xgb_model_ht, "XGBoost Classifier (HT)")
```

Learning Curve

- **max_depth**

```
# param_values = [int(x) for x in np.linspace(1, 10, 10)] # max_depth

# train_scores = []
# test_scores = []

# for c in param_values:
#     model = XGBClassifier(max_depth=c, random_state=42)
#     model.fit(X_train, y_train)

#     # eval on train
#     y_pred_train_proba = model.predict_proba(X_train)
#     train_auc = roc_auc_score(y_train, y_pred_train_proba[:,1])
#     train_scores.append(train_auc)

#     # eval on test
#     y_pred_proba = model.predict_proba(X_test)
#     test_auc = roc_auc_score(y_test, y_pred_proba[:,1])
#     test_scores.append(test_auc)

#     # print('param value: ' + str(c) + '; train: ' + str(train_auc)
+ '; test: '+ str(test_auc))

# fig = plt.figure(figsize =(10, 4))
# plt.plot(param_values, train_scores, label='Train', linewidth=2.5,
marker ='.', markersize=10)
# plt.plot(param_values, test_scores, label='Test', linewidth=2.5,
marker ='.', markersize=10)

# plt.legend()
# plt.show()
```

- **gamma**

```
# param_values= [float(x) for x in np.linspace(0, 100, num = 10)]

# train_scores = []
# test_scores = []

# for c in param_values:
#     model = XGBClassifier(max_depth=1, gamma=c, random_state=42)
#     model.fit(X_train, y_train)
```

```
#     # eval on train
#     y_pred_train_proba = model.predict_proba(X_train)
#     train_auc = roc_auc_score(y_train, y_pred_train_proba[:,1])
#     train_scores.append(train_auc)

#     # eval on test
#     y_pred_proba = model.predict_proba(X_test)
#     test_auc = roc_auc_score(y_test, y_pred_proba[:,1])
#     test_scores.append(test_auc)

#     # print('param value: ' + str(c) + '; train: ' + str(train_auc)
+ '; test: '+ str(test_auc))

# fig = plt.figure(figsize =(10, 4))
# plt.plot(param_values, train_scores, label='Train', linewidth=2.5,
marker ='.', markersize=10)
# plt.plot(param_values, test_scores, label='Test', linewidth=2.5,
marker ='.', markersize=10)

# plt.legend()
# plt.show()
```

- **tree_method**

```
# param_values=['auto', 'exact', 'approx', 'hist']

# train_scores = []
# test_scores = []

# for c in param_values:
#     model = XGBClassifier(max_depth=1, gamma=0, tree_method=c,
random_state=42)
#     model.fit(X_train, y_train)

#     # eval on train
#     y_pred_train_proba = model.predict_proba(X_train)
#     train_auc = roc_auc_score(y_train, y_pred_train_proba[:,1])
#     train_scores.append(train_auc)

#     # eval on test
#     y_pred_proba = model.predict_proba(X_test)
#     test_auc = roc_auc_score(y_test, y_pred_proba[:,1])
#     test_scores.append(test_auc)

#     # print('param value: ' + str(c) + '; train: ' + str(train_auc)
+ '; test: '+ str(test_auc))

# fig = plt.figure(figsize =(10, 4))
# plt.plot(param_values, train_scores, label='Train', linewidth=2.5,
```

```
marker ='.', markersize=10)
# plt.plot(param_values, test_scores, label='Test', linewidth=2.5,
marker ='.', markersize=10)

# plt.legend()
# plt.show()
```

- **min_child_weight**

```
# param_values= [int(x) for x in np.linspace(1, 20, num = 10)]

# train_scores = []
# test_scores = []

# for c in param_values:
#     model = XGBClassifier(max_depth=1, gamma=0,
tree_method='approx', min_child_weight=c, random_state=42)
#     model.fit(X_train, y_train)

#     # eval on train
#     y_pred_train_proba = model.predict_proba(X_train)
#     train_auc = roc_auc_score(y_train, y_pred_train_proba[:,1])
#     train_scores.append(train_auc)

#     # eval on test
#     y_pred_proba = model.predict_proba(X_test)
#     test_auc = roc_auc_score(y_test, y_pred_proba[:,1])
#     test_scores.append(test_auc)

#     # print('param value: ' + str(c) + '; train: ' + str(train_auc)
+ '; test: '+ str(test_auc))

# fig = plt.figure(figsize =(10, 4))
# plt.plot(param_values, train_scores, label='Train', linewidth=2.5,
marker ='.', markersize=10)
# plt.plot(param_values, test_scores, label='Test', linewidth=2.5,
marker ='.', markersize=10)

# plt.legend()
# plt.show()
```

Manually Tuning

Melakukan tuning kembali menggunakan hyperparameter best/optimal sebelumnya untuk melihat perubahan evaluasi model

```
# precision_scorer = make_scorer(precision_score, zero_division=0)
# prec_scoring = {"precision": precision_scorer}

# hyperparameters = {
```

```
#                     'max_depth' : [1],
#                     'gamma' : [0],
#                     'tree_method' : ['approx'],
#                     'min_child_weight' : [5],

#                     'colsample_bytree' : [float(x) for x in
np.linspace(0, 1, num = 5)],
#                     'eta' : [float(x) for x in np.linspace(0, 1, num
= 5)],

#                     'lambda' : [float(x) for x in np.linspace(0, 1,
num = 5)],
#                     'alpha' : [float(x) for x in np.linspace(0, 1,
num = 5)]
#                     }

# from xgboost import XGBClassifier
# xgb = XGBClassifier(random_state=42)
# xgb_model_ht2 = RandomizedSearchCV(xgb, hyperparameters, cv=7,
random_state=42, scoring=prec_scoring, refit="precision", n_iter=20)
# xgb_model_ht2.fit(X_train,y_train)

# print("Best: %f using %s" % (xgb_model_ht2.best_score_,
xgb_model_ht2.best_params_))

# xgb_model_ht2 = XGBClassifier(tree_method = 'auto', min_child_weight
= 5,
#                               max_depth = 1, reg_lambda = 0.25,
gamma = 0,
#                               eta = 0.25, colsample_bytree = 0.25,
#                               alpha = 0.0, random_state = 42)
# xgb_model_ht2.fit(X_train, y_train)

# eval_classification(xgb_model_ht2, "XGBoost Classifier (HT2)")
```

9. LGBM Classifier

Tuning CV

```
# # List of hyperparameter
# hyperparameters = {
#     'num_leaves': list(range(20,50,10)),
#     'n_estimators': list(range(100,600,100)),
#     'objective': ['binary'],
#     'min_child_samples':list(range(20,50,10))
# }

# lgbm = LGBMClassifier(random_state=0)
# cv = StratifiedKFold(n_splits=5)
```

```
# lgbm_model_ht = RandomizedSearchCV(lgbm, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # lgbm_model_ht = GridSearchCV(lgbm, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = lgbm_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# lgbm_model_ht = LGBMClassifier(objective = 'binary', num_leaves =
40, n_estimators = 400, min_child_samples = 40, random_state=0)
# lgbm_model_ht.fit(X_train,y_train)

# eval_classification(lgbm_model_ht, "LGBM Classifier (HT)")
```

10. Gradient Boosting Classifier

Tuning CV

```
# # List of hyperparameter
# hyperparameters = {
#     "n_estimators": [1, 2, 5, 10, 20, 50, 100, 200, 500],
#     "max_leaf_nodes": [2, 5, 10, 20, 50, 100],
#     "max_depth":[1,3,5,7,9, 11, 13],
#     "learning_rate":[0.01,0.1,1,10,100]
# }

# gb = GradientBoostingClassifier(random_state=42)
# cv = StratifiedKFold(n_splits=5)

# gb_model_ht = RandomizedSearchCV(gb, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # gb_model_ht = GridSearchCV(gb, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = gb_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))
```

```
# gb_model_ht = GradientBoostingClassifier(n_estimators = 100,
max_leaf_nodes = 50, max_depth = 7, learning_rate = 1,
random_state=42)
# gb_model_ht.fit(X_train,y_train)

# eval_classification(gb_model_ht, "Gradient Boosting Classifier
(HT)")
```

11. Support Vector Machine

Tuning CV

```
# # List of hyperparameter
# kernel = ['poly', 'rbf', 'sigmoid', 'linear']
# C = np.logspace(-4,4,20)
# gamma = ['scale', 'auto', 100, 1, 0.1, 0.01, 0.001]

# hyperparameters = dict(kernel=kernel, C=C, gamma=gamma)

# svm = SVC(probability=True, random_state=42)
# cv = StratifiedKFold(n_splits=5)

# svm_model_ht = RandomizedSearchCV(svm, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1", random_state=1)
# # svm_model_ht = GridSearchCV(svm, hyperparameters, cv=cv,
scoring=custom_scoring, refit="f1")

# grid_result = svm_model_ht.fit(X_train, y_train)
# # summarize results
# print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
# # means = grid_result.cv_results_['mean_test_f1']
# # stds = grid_result.cv_results_['std_test_f1']
# # params = grid_result.cv_results_['params']
# # for mean, stdev, param in zip(means, stds, params):
# #     print("%f (%f) with: %r" % (mean, stdev, param))

# svm_model_ht = SVC(kernel = 'rbf', gamma = 1, C = 3792.690190732246,
probability=True, random_state=42)
# svm_model_ht.fit(X_train,y_train)

# eval_classification(svm_model_ht, "Support Vector Machine (HT)")
```

12. Catboost Classifier

Tuning CV

```
%%capture
!pip install optuna
import optuna
```

```
from catboost import CatBoostClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer

# def objective(trial):
#     # Definisikan hyperparameter yang ingin di-tuning
#     params = {
#         'iterations': trial.suggest_int('iterations', 100, 1000),
#         'depth': trial.suggest_int('depth', 4, 10),
#         'learning_rate': trial.suggest_float('learning_rate', 1e-3,
0.3),
#         'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 1e-3,
10.0),
#         'border_count': trial.suggest_int('border_count', 1, 255),
#         'random_strength': trial.suggest_float('random_strength',
1e-3, 10.0),
#         'bagging_temperature':
trial.suggest_float('bagging_temperature', 0.0, 1.0),
#         'od_type': trial.suggest_categorical('od_type', ['IncToDec',
'Iter']),
#         'od_wait': trial.suggest_int('od_wait', 10, 50)
#     }

#     # Inisialisasi model dengan parameter yang dihasilkan oleh
Optuna
#     model = CatBoostClassifier(
#         **params,
#         verbose=0
#     )

#     # Evaluasi model dengan cross-validation
#     scores = cross_val_score(model, X_train, y_train, cv=3,
scoring='accuracy')
#     accuracy = scores.mean()
#     return accuracy

def objective(trial):
    # Definisikan hyperparameter yang ingin di-tuning
    params = {
#         'iterations': trial.suggest_int('iterations', 100, 1000),
#         'depth': trial.suggest_int('depth', 4, 10),
#         'learning_rate': trial.suggest_float('learning_rate', 1e-3,
0.3),
#         'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 1e-3,
10.0),
#         'border_count': trial.suggest_int('border_count', 1, 255),
#         'random_strength': trial.suggest_float('random_strength',
1e-3, 10.0),
#         'bagging_temperature':
```

```
trial.suggest_float('bagging_temperature', 0.0, 1.0),
        'od_type': trial.suggest_categorical('od_type', ['IncToDec',
'Iter']),
#         'od_wait': trial.suggest_int('od_wait', 10, 50)
    }

    # Inisialisasi model dengan parameter yang dihasilkan oleh Optuna
    model = CatBoostClassifier(
        **params,
        verbose=0
    )

    # Evaluasi model dengan cross-validation
    scores = cross_val_score(model, X_train, y_train, cv=3,
scoring='accuracy')
    accuracy = scores.mean()
    return accuracy

# Mulai proses tuning hyperparameter
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

# Tampilkan hasil terbaik
print("Best trial:")
trial = study.best_trial
print(f"  Value: {trial.value}")
print("  Params: ")
for key, value in trial.params.items():
    print(f"    {key}: {value}")

best_params = trial.params
best_model = CatBoostClassifier(**best_params, verbose=0)
best_model.fit(X_train, y_train)

# Evaluasi pada data test
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")
```

Stacking

Melakukan kombinasi dari banyak model klasifikasi

```
from catboost import CatBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier
```

```
from xgboost import XGBClassifier
def algorithm_pipeline(X_train_data, X_test_data, y_train_data,
y_test_data, model, hyperparameters, cv=5, scoring_fit='recall'):
    rs = RandomizedSearchCV(estimator=model,
param_distributions=hyperparameters, cv=cv, n_jobs=-1,
scoring=scoring_fit, verbose=0, random_state=42) # Use GridSearch
better if you have more resource & time
    rs.fit(X_train_data, y_train_data)
    best_model = rs.best_estimator_

    pred = rs.predict(X_test_data)
    pred_proba = rs.predict(X_test_data)
    score = roc_auc_score(y_test_data, pred_proba)
    return [best_model, pred, score]

# Tambahkan CatBoostClassifier ke dalam daftar model yang akan dilatih
models_to_train = [
    LogisticRegression(),
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    AdaBoostClassifier(random_state=42),
    XGBClassifier(random_state=42),
    CatBoostClassifier(verbose=0, random_state=42)  # Tambahkan
CatBoostClassifier
]

# Definisikan hyperparameters untuk CatBoost
grid_parameters = [
    # Hyperparameters untuk LogisticRegression
    {'C': [0.1, 1.0, 10.0]},
    # Hyperparameters untuk KNeighborsClassifier
    {'n_neighbors': [3, 5, 7]},
    # Hyperparameters untuk DecisionTreeClassifier
    {'max_depth': [None, 10, 20]},
    # Hyperparameters untuk RandomForestClassifier
    {'n_estimators': [100, 200], 'max_depth': [None, 10, 20]},
    # Hyperparameters untuk AdaBoostClassifier
    {'n_estimators': [50, 100], 'learning_rate': [0.01, 0.1, 1.0]},
    # Hyperparameters untuk XGBClassifier
    {'n_estimators': [100, 200], 'learning_rate': [0.01, 0.1, 0.3]},
    # Hyperparameters untuk CatBoostClassifier
    {
        'iterations': [100, 500, 1000],
        'depth': [4, 6, 8],
        'learning_rate': [0.01, 0.03, 0.1],
        'l2_leaf_reg': [3, 5, 7],
        'border_count': [32, 64, 128],
        'bagging_temperature': [0.0, 0.5, 1.0],
```

```
        'od_type': ['IncToDec', 'Iter'],
        'od_wait': [10, 20, 30]
    }
]

# Lanjutkan dengan pipeline yang sudah ada
models_preds_scores = []

for i, model in enumerate(models_to_train):
    hyperparameters = grid_parameters[i]
    result = algorithm_pipeline(X_train, X_test, y_train, y_test,
model, hyperparameters, cv=5)
    models_preds_scores.append(result)

for result in models_preds_scores:
    print('Model: {0}, Score: {1}'.format(type(result[0]).__name__,
result[2])) # score in training data
```

```
Model: LogisticRegression, Score: 0.5098727802809435
Model: KNeighborsClassifier, Score: 0.6887755102040817
Model: DecisionTreeClassifier, Score: 0.6718791412668964
Model: RandomForestClassifier, Score: 0.709780015902465
Model: AdaBoostClassifier, Score: 0.5285250463821892
Model: XGBClassifier, Score: 0.6984163795388286
Model: CatBoostClassifier, Score: 0.7027895573813941
```

Stacking Implementation

```
from sklearn.ensemble import StackingClassifier
# Define level-1 algorithms
level1 = list()
level1.append(('lr', LogisticRegression()))
level1.append(('rf', RandomForestClassifier()))
level1.append(('ab', AdaBoostClassifier()))
level1.append(('xg', XGBClassifier()))
level1.append(('cb', CatBoostClassifier(verbose=0, random_state=42)))
# Tambahkan CatBoostClassifier

# Define level-2 algorithm
level2 = XGBClassifier()

# Fit model
stacking_model = StackingClassifier(estimators=level1,
final_estimator=level2, cv=5)
stacking_model.fit(X_train, y_train)
```

```
StackingClassifier(cv=5,
                   estimators=[('lr', LogisticRegression()),
                               ('rf', RandomForestClassifier()),
                               ('ab', AdaBoostClassifier()),
```

```
                                ('xg',
                                 XGBClassifier(base_score=None,
booster=None,
                                               callbacks=None,
                                               colsample_bylevel=None,
                                               colsample_bynode=None,
                                               colsample_bytree=None,
                                               device=None,

early_stopping_rounds=None,

enable_categorical=False,
                                               eval_metric=None,
                                               feature_types=...
                                                  grow_policy=None,
                                                  importance_type=None,

interaction_constraints=None,
                                                  learning_rate=None,
                                                  max_bin=None,

max_cat_threshold=None,

max_cat_to_onehot=None,
                                                  max_delta_step=None,
                                                  max_depth=None,
                                                  max_leaves=None,

min_child_weight=None,
                                                  missing=nan,

monotone_constraints=None,
                                                  multi_strategy=None,
                                                  n_estimators=None,
n_jobs=None,

num_parallel_tree=None,

random_state=None, ...))

eval_classification(stacking_model, "Stacking Classifier")

<pandas.io.formats.style.Styler at 0x7872fdedc9a0>
```

Voting Classifier

```
from sklearn.ensemble import VotingClassifier
from catboost import CatBoostClassifier

vote_model = VotingClassifier(
```

```
    estimators = [
        ('lr', LogisticRegression(C=4.281332398719396)),
        ('knn', KNeighborsClassifier(algorithm='brute', leaf_size=48,
n_neighbors=17, p=1)),
        ('dt', DecisionTreeClassifier()),
        ('rf', RandomForestClassifier()),
        ('xg', XGBClassifier()),
        ('cb', CatBoostClassifier(verbose=0, random_state=42))  #
Tambahkan CatBoostClassifier
    ], voting = 'soft')

vote_model.fit(X_train, y_train)
```

```
VotingClassifier(estimators=[('lr',
LogisticRegression(C=4.281332398719396)),
                             ('knn',
                              KNeighborsClassifier(algorithm='brute',
                                                   leaf_size=48,
n_neighbors=17,
                                                   p=1)),
                             ('dt', DecisionTreeClassifier()),
                             ('rf', RandomForestClassifier()),
                             ('ab', AdaBoostClassifier()),
                             ('xg',
                              XGBClassifier(base_score=None,
booster=None,
                                            callbacks=None,
                                            colsample_bylevel=None,
                                            colsample_bynod...
                                            learning_rate=None,
max_bin=None,
                                            max_cat_threshold=None,
                                            max_cat_to_onehot=None,
                                            max_delta_step=None,
max_depth=None,
                                            max_leaves=None,
                                            min_child_weight=None,
missing=nan,
                                            monotone_constraints=None,
                                            multi_strategy=None,
                                            n_estimators=None,
n_jobs=None,
                                            num_parallel_tree=None,
                                            random_state=None, ...)),
                             ('cb',
                              <catboost.core.CatBoostClassifier object
at 0x7872fe1997e0>)],
                 voting='soft')
```

```
eval_classification(vote_model, "Voting Classifier")
```

```
<pandas.io.formats.style.Styler at 0x7872fe2cc100>
```

Model Comparison

Melihat summary metrics dari berbagai algoritma yang telah dibuat

* HT --> Hyperparameter Tuning

* HT2 --> Manually Tuning

Perbandingan `precision`, `recall` dan `f1 score` pada train test tertinggi

- Tanpa Tuning

```
results_eval = pd.DataFrame({
    "Models" : train_modelname_list,
    "Precision (Train)": train_precision_list,
    "Precision (Test)": test_precision_list,
    "Recall (Train)": train_recall_list,
    "Recall (Test)": test_recall_list,
    "F0.5 Score (Train)" : train_fbeta_score_list,
    "F0.5 Score (Test)" : test_fbeta_score_list,
    "F1 Score (Train)" : train_f1_score_list,
    "F1 Score (Test)" : test_f1_score_list
})

results_eval.drop_duplicates(inplace = True)

results_eval_notht =
results_eval[~results_eval["Models"].str.contains("HT")]

results_eval_notht = results_eval_notht.sort_values(by=["F0.5 Score
(Test)", "Precision (Test)", "Recall (Test)"], ascending=[False,
False, False]).reset_index(drop = True)

results_eval_notht.style.format(precision=3).background_gradient(cmap=
'Purples')
```

```
<pandas.io.formats.style.Styler at 0x7872fe0f9720>
```

- Dengan Tuning

```
results_eval_ht =
results_eval[results_eval["Models"].str.contains("HT")]
results_eval_ht = results_eval_ht.sort_values(by=["F0.5 Score (Test)",
"Precision (Test)", "Recall (Test)"], ascending=[False, False,
False]).reset_index(drop = True)

results_eval_ht.style.format(precision=3).background_gradient(cmap='Pu
rples')
```

```
<pandas.io.formats.style.Styler at 0x78735e3e7cd0>
```

	Models	Precision (Train)	Precision (Test)	Recall (Train)	Recall (Test)	F0.5 Score (Train)	F0.5 Score (Test)	F1 Score (Train)	F1 Score (Test)
0	Random Forest (HT1)	0.997	0.772	0.993	0.530	0.996	0.707	0.995	0.629
1	Random Forest (HT2)	0.972	0.724	0.912	0.506	0.959	0.667	0.941	0.596
2	XGBoost Classifier (HT)	0.976	0.710	0.920	0.530	0.964	0.665	0.947	0.607
3	LGBM Classifier (HT)	0.999	0.681	0.992	0.566	0.997	0.655	0.995	0.618
4	Random Forest (HT3)	0.867	0.702	0.584	0.482	0.790	0.643	0.698	0.571
5	XGBoost Classifier (HT2)	0.826	0.595	0.699	0.566	0.797	0.589	0.757	0.580
6	Gradient Boosting Classifier (HT)	0.996	0.609	0.994	0.470	0.996	0.575	0.995	0.531
7	Logistic Regression (HT)	0.755	0.528	0.660	0.675	0.734	0.552	0.704	0.593
8	MLP Classifier (HT)	0.904	0.523	0.933	0.542	0.909	0.527	0.918	0.533
9	Naive Bayes (HT)	0.736	0.512	0.480	0.518	0.665	0.513	0.581	0.515
10	Decision Tree (HT)	1.000	0.484	0.990	0.554	0.998	0.497	0.995	0.517
11	K-Nearest Neighbors (HT)	0.763	0.465	0.871	0.639	0.783	0.492	0.814	0.538
12	Decision Tree (HT2)	0.719	0.441	0.574	0.542	0.685	0.458	0.639	0.486
13	Support Vector Machine (HT)	0.997	0.650	0.993	0.157	0.996	0.399	0.995	0.252

Perbandingan `precision`, `recall` dan `f1 score` pada train test dengan Total Difference terendah

- Tanpa Tuning

```
results_diff = pd.DataFrame({
    "Models" : train_modelname_list,
    "Precision (Train)": train_precision_list,
    "Precision (Test)": test_precision_list,
    "Recall (Train)": train_recall_list,
    "Recall (Test)": test_recall_list,
    "F0.5 Score (Train)" : train_fbeta_score_list,
```

```
    "F0.5 Score (Test)" : test_fbeta_score_list,
    "F1 Score (Train)" : train_f1_score_list,
    "F1 Score (Test)" : test_f1_score_list
})

results_diff.drop_duplicates(inplace = True)

results_diff["Total Diff"] = (results_diff["Precision (Train)"] -
results_diff["Precision (Test)"]) + \
                   (results_diff["Recall (Train)"] -
results_diff["Recall (Test)"]) + \
                   (results_diff["F0.5 Score (Train)"] -
results_diff["F0.5 Score (Test)"])

results_diff_notht =
results_diff[~results_diff["Models"].str.contains("HT")]

results_diff_notht = results_diff_notht.sort_values(by=["Total Diff",
"F0.5 Score (Test)", "Precision (Test)", "Recall (Test)"],
ascending=[True, False, False, False]).reset_index(drop = True)

results_diff_notht.style.format(precision=3).background_gradient(cmap=
'Greens')
```

```
<pandas.io.formats.style.Styler at 0x7873025de9e0>
```

- Dengan Tuning

```
results_diff_ht =
results_diff[results_diff["Models"].str.contains("HT")]

results_diff_ht = results_diff_ht.sort_values(by=["Total Diff", "F0.5
Score (Test)", "Precision (Test)", "Recall (Test)"], ascending=[True,
False, False, False]).reset_index(drop = True)

results_diff_ht.style.format(precision=3).background_gradient(cmap='Gr
eens')
```

```
<pandas.io.formats.style.Styler at 0x787358be7c40>
```

	Models	Precision (Train)	Precision (Test)	Recall (Train)	Recall (Test)	Score (Train)	Score (Test)	Score (Train)	Score (Test)	Total Diff
0	Naive Bayes (HT)	0.736	0.512	0.480	0.518	0.665	0.513	0.581	0.515	0.338
1	Logistic Regression (HT)	0.755	0.528	0.660	0.675	0.734	0.552	0.704	0.593	0.394
2	Random Forest (HT3)	0.867	0.702	0.584	0.482	0.790	0.643	0.698	0.571	0.414
3	Decision Tree (HT2)	0.719	0.441	0.574	0.542	0.685	0.458	0.639	0.486	0.537
4	XGBoost Classifier (HT2)	0.826	0.595	0.699	0.566	0.797	0.589	0.757	0.580	0.572
5	K-Nearest Neighbors (HT)	0.763	0.465	0.871	0.639	0.783	0.492	0.814	0.538	0.821
6	Random Forest (HT2)	0.972	0.724	0.912	0.506	0.959	0.667	0.941	0.596	0.946
7	XGBoost Classifier (HT)	0.976	0.710	0.920	0.530	0.964	0.665	0.947	0.607	0.955
8	Random Forest (HT1)	0.997	0.772	0.993	0.530	0.996	0.707	0.995	0.629	0.977
9	LGBM Classifier (HT)	0.999	0.681	0.992	0.566	0.997	0.655	0.995	0.618	1.086
10	MLP Classifier (HT)	0.904	0.523	0.933	0.542	0.909	0.527	0.918	0.533	1.154
11	Gradient Boosting Classifier (HT)	0.996	0.609	0.994	0.470	0.996	0.575	0.995	0.531	1.332
12	Decision Tree (HT)	1.000	0.484	0.990	0.554	0.998	0.497	0.995	0.517	1.453
13	Support Vector Machine (HT)	0.997	0.650	0.993	0.157	0.996	0.399	0.995	0.252	1.780

Evaluation Metrics pada Train Urutan Tertinggi

```
results_train = pd.DataFrame({
    'Model (Train)': train_modelname_list,
    'Accuracy': train_accuracy_list,
    'Precision': train_precision_list,
    'Recall': train_recall_list,
    'F0.5 Score': train_fbeta_score_list,
    'F1 Score': train_f1_score_list,
```

```
    'Cross Val F1 (k=5)': train_cross_val_f1_list,
    'ROC AUC': train_roc_auc_score_list,
    'Cross Val ROC AUC (k=5)': train_cross_val_rocauc_list
})

results_train.drop_duplicates(inplace = True)

results_train = results_train.sort_values(by=['F0.5 Score',
'Precision', 'Recall'], ascending=False).reset_index(drop = True)

results_train.style.format(precision=3).background_gradient(cmap='Blue
s')
```

```
<pandas.io.formats.style.Styler at 0x787358a5a260>
```

Evaluation Metrics pada Test Urutan Tertinggi

```
results_test = pd.DataFrame({
    'Model (Test)': test_modelname_list,
    'Accuracy': test_accuracy_list,
    'Precision': test_precision_list,
    'Recall': test_recall_list,
    'F0.5 Score': test_fbeta_score_list,
    'F1 Score': test_f1_score_list,
    'Cross Val F1 (k=5)': test_cross_val_f1_list,
    'ROC AUC': test_roc_auc_score_list,
    'Cross Val ROC AUC (k=5)': test_cross_val_rocauc_list
})

results_test.drop_duplicates(inplace = True)

results_test = results_test.sort_values(by=['F0.5 Score', 'Precision',
'Recall'], ascending=False).reset_index(drop = True)

results_test.style.format(precision=3).background_gradient(cmap='Orang
es')
```

```
<pandas.io.formats.style.Styler at 0x7872fe0f9150>
```

```
results_test_notht = pd.DataFrame({
    'Model (Test)': test_modelname_list,
    'Accuracy': test_accuracy_list,
    'Precision': test_precision_list,
    'Recall': test_recall_list,
    'F0.5 Score': test_fbeta_score_list,
    'F1 Score': test_f1_score_list,
    'Cross Val F1 (k=5)': test_cross_val_f1_list,
    'ROC AUC': test_roc_auc_score_list,
    'Cross Val ROC AUC (k=5)': test_cross_val_rocauc_list
})
```

```
results_test_notht.drop_duplicates(inplace = True)

results_test_notht = results_test[~results_test["Model
(Test)"].str.contains("HT")]
results_test_notht = results_test_notht[~results_test_notht["Model
(Test)"].isin(["Voting Classifier", "Stacking Classifier"])]

results_test_notht = results_test_notht.sort_values(by=['F0.5 Score',
'Precision', 'Recall'], ascending=False).reset_index(drop = True)

results_test_notht.style.format(precision=3).background_gradient(cmap=
'Oranges')
```

```
<pandas.io.formats.style.Styler at 0x7872fe0f9b70>
```

```
results_diff_ht =
results_diff[results_diff["Models"].str.contains("HT")]

results_diff_ht = results_diff_ht.sort_values(by=["Total Diff", "F0.5
Score (Test)", "Precision (Test)", "Recall (Test)"], ascending=[True,
False, False, False]).reset_index(drop = True)

results_diff_ht.style.format(precision=3).background_gradient(cmap='Gr
eens')
```

```
<pandas.io.formats.style.Styler at 0x7872fe0f9cf0>
```

	Models	Precision (Train)	Precision (Test)	Recall (Train)	Recall (Test)	F0.5 Score (Train)	F0.5 Score (Test)	F1 Score (Train)	F1 Score (Test)	Total Diff
0	Naive Bayes (HT)	0.736	0.512	0.480	0.518	0.665	0.513	0.581	0.515	0.338
1	Logistic Regression (HT)	0.755	0.528	0.660	0.675	0.734	0.552	0.704	0.593	0.394
2	Random Forest (HT3)	0.867	0.702	0.584	0.482	0.790	0.643	0.698	0.571	0.414
3	Decision Tree (HT2)	0.719	0.441	0.574	0.542	0.685	0.458	0.639	0.486	0.537
4	XGBoost Classifier (HT2)	0.826	0.595	0.699	0.566	0.797	0.589	0.757	0.580	0.572
5	K-Nearest Neighbors (HT)	0.763	0.465	0.871	0.639	0.783	0.492	0.814	0.538	0.821
6	Random Forest (HT2)	0.972	0.724	0.912	0.506	0.959	0.667	0.941	0.596	0.946
7	XGBoost Classifier (HT)	0.976	0.710	0.920	0.530	0.964	0.665	0.947	0.607	0.955
8	Random Forest (HT1)	0.997	0.772	0.993	0.530	0.996	0.707	0.995	0.629	0.977
9	LGBM Classifier (HT)	0.999	0.681	0.992	0.566	0.997	0.655	0.995	0.618	1.086
10	MLP Classifier (HT)	0.904	0.523	0.933	0.542	0.909	0.527	0.918	0.533	1.154
11	Gradient Boosting Classifier (HT)	0.996	0.609	0.994	0.470	0.996	0.575	0.995	0.531	1.332
12	Decision Tree (HT)	1.000	0.484	0.990	0.554	0.998	0.497	0.995	0.517	1.453
13	Support Vector Machine (HT)	0.997	0.650	0.993	0.157	0.996	0.399	0.995	0.252	1.780