Rakamin Academy

Laporan Final Project Byte Blazers

- Muhamad Faiz Widagdo
- Robiatul Adawiyah
- Chianti Ridhwan
- Lulu Safira
- Retno Debbi Yulisya
- Imam Luthfi
- Melliza Nastasia Izazi

Rakamin Academy

STAGE 2

Pre-Processing

Handling Missing Value

Handling missing values dilakukan untuk mengatasi keberadaan nilai-nilai yang kosong atau hilang dalam suatu dataset. Hal ini sangat penting karena nilai yang hilang dapat memengaruhi hasil analisis dan model prediktif.

1.) Handling Missing Value

Untuk mendeteksi missing value digunakan .isnull().sum() kemudian menghapus baris yang mengandung nilai yang kosong dengan .dropna()

```
[25] # Mendeteksi nilai yang hilang
     missing_values = df.isnull().sum()

     # Menghapus baris yang mengandung nilai yang hilang
     df = df.dropna()
     df.head()
```

Setelah dilakukan Handling Missing Value **tidak terdapat data yang hilang** (*missing value*)

2.) Mengecek Kembali Missing Value

```
[ ] df.isna().sum()
```

```
Age                    0
Employment Type        0
GraduateOrNot          0
AnnualIncome           0
FamilyMembers          0
ChronicDiseases        0
FrequentFlyer          0
EverTravelledAbroad    0
TravelInsurance        0
dtype: int64
```

Handling Duplicate Values

Handling data duplicates bertujuan untuk mengatasi masalah ketika kita memiliki informasi yang sama atau sangat mirip dalam dataset. Ini bisa terjadi ketika kita memiliki baris yang memiliki nilai yang sama dalam beberapa kolom

Pengecekan Data Duplicate

```
[ ] df.duplicated().any()

    True
```

Untuk mengecek data yang duplicate, menggunakan df.duplicates( ). Dari hasil tersebut ditemukan bahwa ada data yang duplicate.

Maka dari itu kami **menghapus data duplicate** tersebut menggunakan df.drop_duplicates( ).

```
[ ] #Menangani Data Duplikat:
    duplicate_rows = df[df.duplicated()]

    # Menghapus data duplikat
    df = df.drop_duplicates()
    df.head()
```

	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
0	31	Government Sector	Yes	400000	6	1	No	No	0
1	31	Private Sector/Self Employed	Yes	1250000	7	0	No	No	0
2	34	Private Sector/Self Employed	Yes	500000	4	1	No	No	1
3	28	Private Sector/Self Employed	Yes	700000	3	1	No	No	0
4	28	Private Sector/Self Employed	Yes	700000	8	1	Yes	No	0

Handling Redundant Data

Data Redundant adalah suatu kondisi ketika keberadaan informasi yang berlebihan atau berulang dalam suatu dataset.

Penanganan data redundan penting untuk menjaga kualitas data, mengoptimalkan penggunaan sumber daya, dan mencegah kesalahan analisis yang disebabkan oleh duplikasi atau ketidaksempurnaan dalam dataset. Penanganan data redundant adalah dengan melakukan pembersihan data dan menghapus atau menggabungkan informasi yang berulang.

Pada dataset yang digunakan, kami tidak menemukan adanya data redundant yang perlu ditangani. Sehingga Handling Redundant Data tidak dilakukan.

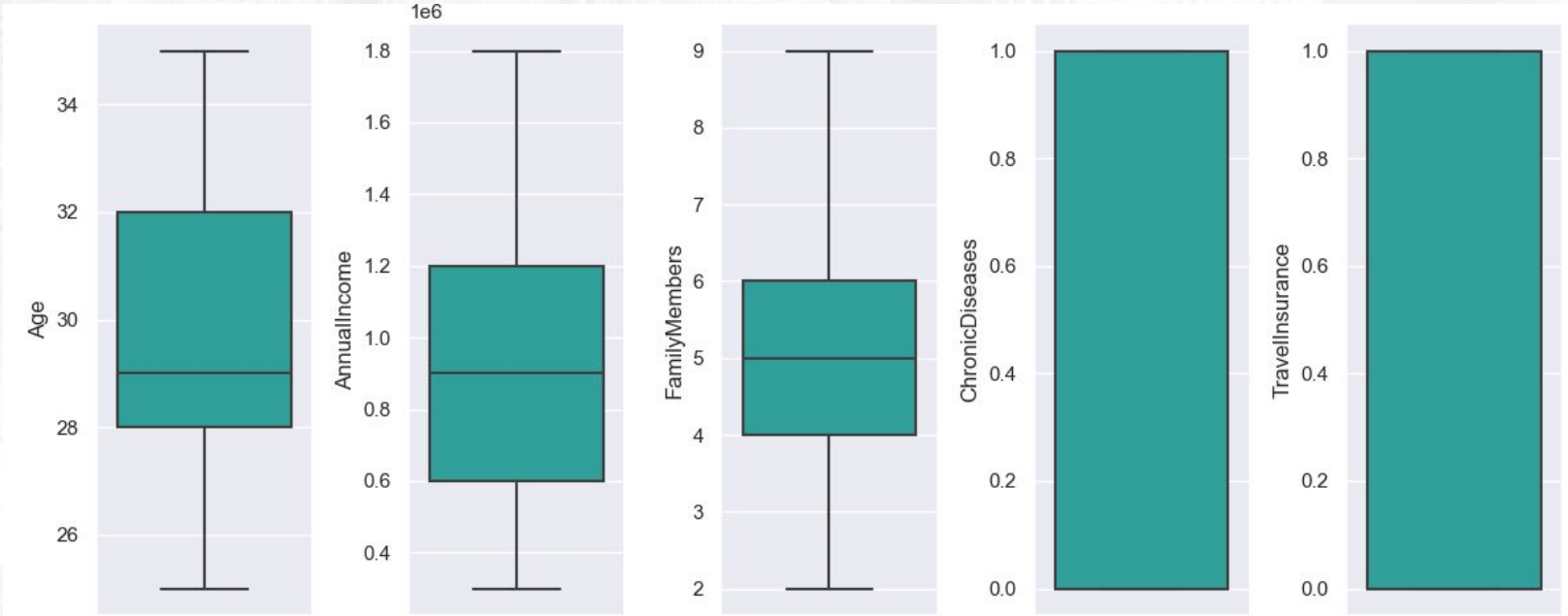
Handling Outliers

Handling outliers dilakukan untuk mengatasi nilai-nilai yang sangat jauh atau tidak biasa dalam suatu dataset yang dapat mempengaruhi analisis statistik dan model prediktif, sehingga menangani mereka membantu mencegah kesalahan atau distorsi dalam interpretasi hasil analisis data.

Pengecekan Outlier

```
# Boxplot untuk melihat outliers

features = num
for i in range(0, len(features)):
    plt.subplot(1, len(features), i+1)
    sns.boxplot(y=df[features[i]], color='lightseagreen', orient='v')
    plt.tight_layout()
```



Boxplot di samping menunjukkan bahwa **tidak terdapat outlier** pada data.

Feature Encoding

```
[33] # Membagi predictor dan target variable
     y = df['TravelInsurance']
     x = df.drop('TravelInsurance', axis = 1)
     data_total = df
```

Untuk memudahkan membangun model machine learning di tahap selanjutnya, kami akan melakukan convert **'Yes/No'** menjadi **1/0 (binary)** pada kolom **GraduateOrNot, FrequentFlyer, EverTravelledAbroad** dan **ChronicDiseases**

```
[34] # Fungsi untuk converting Yes/No menjadi 1/0 (Binary)

     def convert_yesno_to_binary(x):
         if x == "Yes":
             return 1
         else:
             return 0
```

```
[35] x['GraduateOrNot'] = x['GraduateOrNot'].apply(convert_yesno_to_binary)
     x['FrequentFlyer'] = x['FrequentFlyer'].apply(convert_yesno_to_binary)
     x['EverTravelledAbroad'] = x['EverTravelledAbroad'].apply(convert_yesno_to_binary)
```

Untuk **Employment Type** kami akan mengubah **1/0 (binary)** menjadi:

- **Government Sector : 1**
- **Private Sector : 0**

```
[37] # Converting Employment Type ke binary

     def convert_employmenttype_to_binary(employmenttype):
         if employmenttype == "Government Sector":
             return 1
         else:
             return 0
```

```
[38] # Converting EmployementType to Binary
     x['Employment Type'] = x['Employment Type'].apply(convert_employmenttype_to_binary)
```

Feature Transformation

Kemudian dalam membangun model Machine Learning, kami menambah feature baru berupa pengelompokan annual income untuk membantu analisis data.
Pada Feature CatAnIncome terdapat 3 pengelompokan berdasarkan **AnnualIncome**, yairu :

1. **Low** (0 - 600.000)
2. **Mid** (600.001 - 1.250.000)
3. **High** (1.250.001 - 1.800.000)

```
#Menambah Category Income

# Explore AnnualIncome columns
df['AnnualIncome'].describe()

# Categorizing AnnualIncome
bins = [0, 600000, 1250000, 1800000] # 0-600.000 -> low / 600.001 - 1.250.000 -> mid / 1.250.001 - 1.800.000 -> high
groupNames = ["low", "mid", "high"]
df['CatAnIncome'] = pd.cut(df['AnnualIncome'], bins, labels = groupNames, include_lowest = True)
df.head()
```

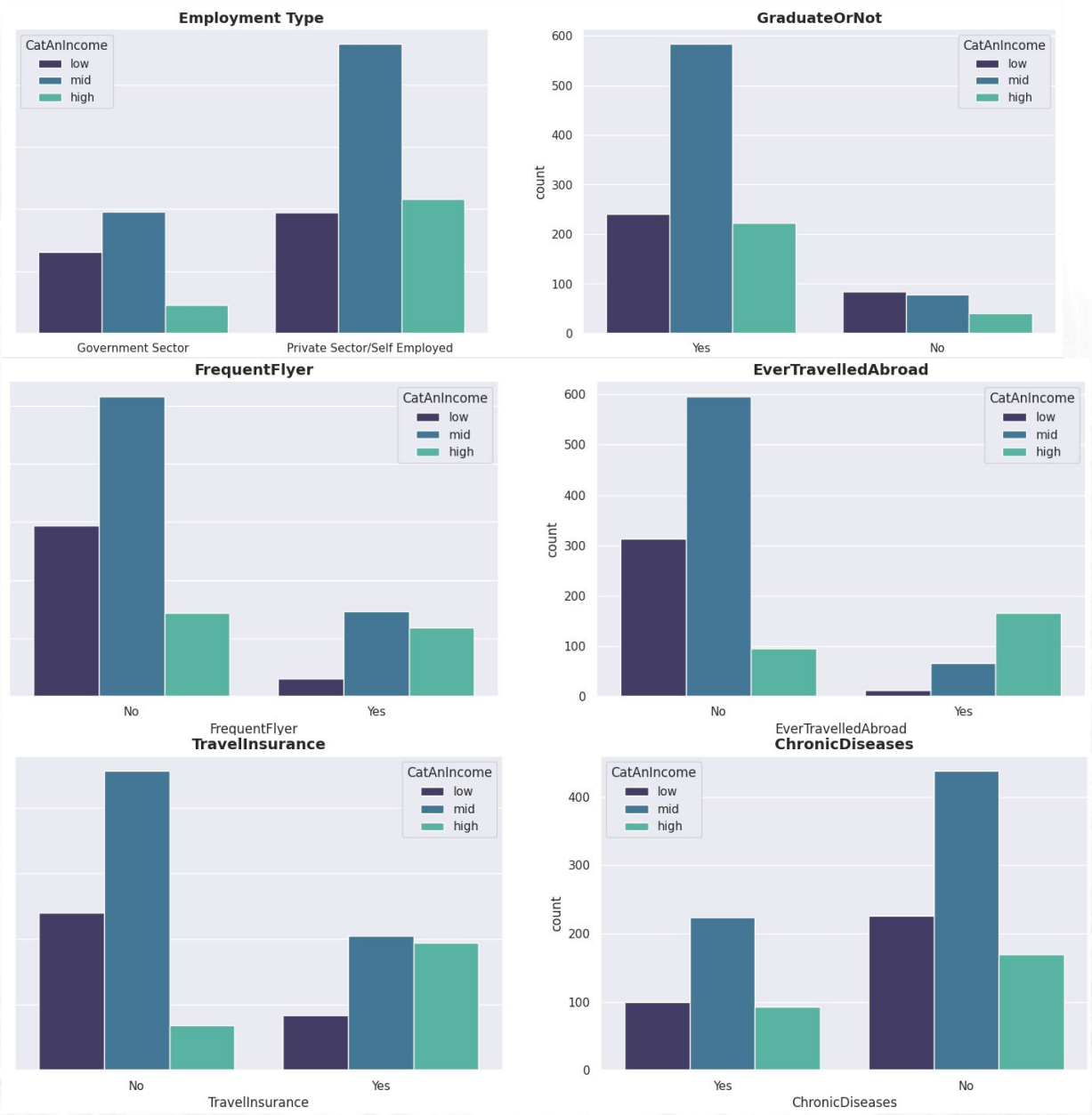
	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance	CatAnIncome
0	31	Government Sector	Yes	400000	6	1	No	No	0	low
1	31	Private Sector/Self Employed	Yes	1250000	7	0	No	No	0	mid
2	34	Private Sector/Self Employed	Yes	500000	4	1	No	No	1	low
3	28	Private Sector/Self Employed	Yes	700000	3	1	No	No	0	mid
4	28	Private Sector/Self Employed	Yes	700000	8	1	Yes	No	0	mid

Feature Engineering

Berikut di sebelah kanan adalah visualisasi data dari penambahan feature engineering CatAnIncome

```
# Categorical
plt.figure(figsize=(18,18))
for i in range(0, len(cat)):
    plt.subplot(3,2,i+1)
    sns.countplot(x=df[cat[i]], hue = df['CatAnIncome'], palette="mako")
    # Set plot title
    plt.title(cat[i], weight='bold', fontsize = 14)
```

Pelanggan yang memiliki penghasilan yang tinggi cenderung akan membeli Asuransi Perjalanan



Normalization

Rakamin Academy

Normalization kami lakukan untuk menjaga konsistensi skala antar fitur dalam dataset karena fitur dengan skala yang tidak seimbang dapat mempengaruhi proses *Machine Learning* dan analisis data.

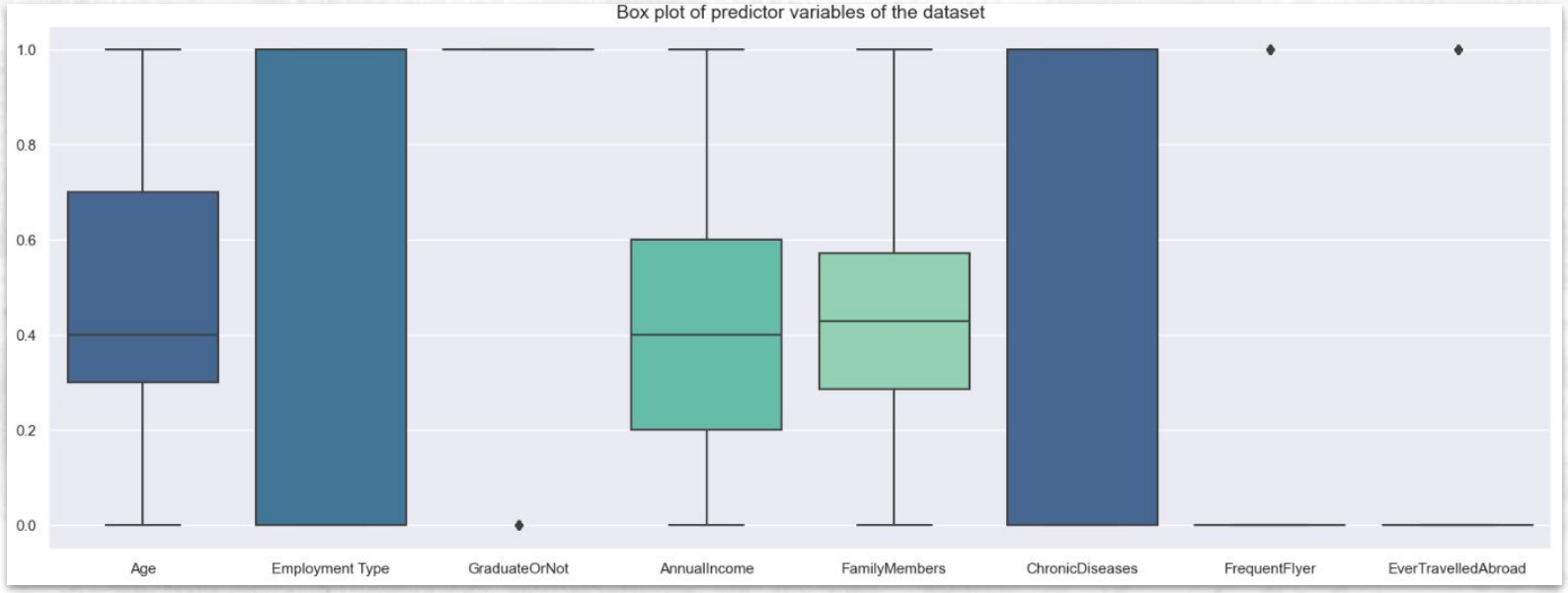
```
# Kolom numerical yang akan dilakukan normalization
num_cols=['AnnualIncome','Age', 'FamilyMembers']

# Import library
from sklearn import preprocessing
feature_to_scale = num_cols
min_max_scaler = preprocessing.MinMaxScaler()

x[feature to scale] = min max scaler.fit transform(x[feature to scale])
```

```
plt.figure(figsize=(20,7))
sns.boxplot(data=x, palette=['#1E121F', '#301F3C', '#413C7B', '#38669E',
                             '#357BA4', '#41B8AD', '#58CBAE', '#8ADBB2'])
plt.title("Box plot of predictor variables of the dataset", size=14)
```

➔ Lalu kami melakukan pengecekan pada predictor variables setelah dilakukan normalisasi



Class Imbalanced

Rakamin Academy

1. Pengecekan Class Imbalance

```
df['Employment Type'].value_counts()
```

```
Private Sector/Self Employed    876
Government Sector               373
Name: Employment Type, dtype: int64
```

2. Menentukan X fitur dan y target untuk dilakukan oversampling

```
X = df[[col for col in df.columns if (str(df[col].dtype) != 'object')
        and col != 'Employment Type']]
y = df['Employment Type'].values
print(X.shape)
print(y.shape)
```

```
(1249, 5)
(1249,)
```

Pada dataset ini kami meningkatkan jumlah sampel minoritas (Government Sector) dengan menciptakan sampel sintesis menggunakan Oversampling metode SMOTE

3. Handling imbalance data dengan resampling data

```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler, SMOTE
import pandas as pd

print("Original class distribution:")
print(pd.Series(y).value_counts())

# Create a SMOTE object with the desired sampling_strategy
X_over_SMOTE, y_over_SMOTE = SMOTE().fit_resample(X_numeric, y)

print("\nClass distribution after SMOTE:")
print(pd.Series(y_over_SMOTE).value_counts())
```

In Percentage

```
Original class distribution:
Private Sector/Self Employed    876      ~70%
Government Sector               373      ~30%
dtype: int64

Class distribution after SMOTE:
Government Sector               876
Private Sector/Self Employed    876
dtype: int64
```

Rakamin Academy

Website Url : ***https://github.com/BYTE-BLAZERS/Stage_2.git***

Github CLI : **gh repo clone BYTE-BLAZERS/Stage_2**