

# **IF3070 DASAR INTELEGENSI ARTIFISIAL**

## **TUGAS BESAR 2**

Implementasi dan Analisis Algoritma *K-Nearest Neighbors* dan *Gaussian Naive-Bayes* pada PhiUSIIL *Phishing* URL Dataset



### **Disusun oleh:**

Thalita Zahra Sutejo	18222023
Irfan Musthofa	18222056
Eleanor Cordelia	18222059
Muhammad Faiz Atharrahman	18222063

**PRODI SISTEM DAN TEKNOLOGI INFORMASI  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG**

**2024**

## DAFTAR ISI

<b>BAB I PENDAHULUAN.....</b>	<b>1</b>
1.1 Latar Belakang.....	1
1.2 Deskripsi Masalah.....	1
<b>BAB II IMPLEMENTASI ALGORITMA.....</b>	<b>3</b>
2.1 Implementasi Algoritma K-Nearest Neighbor (KNN).....	3
a. KNN from Scratch.....	3
b. KNN by Scikit-Learn.....	6
2.2 Implementasi Algoritma Naive Bayes.....	7
a. Naive Bayes from Scratch.....	7
b. Naive Bayes by Scikit-Learn.....	12
<b>BAB III CLEANING DAN PREPROCESSING DATA.....</b>	<b>14</b>
<b>CLEANING DATA.....</b>	<b>14</b>
3.1 Handling Missing Data.....	14
3.2 Dealing with Outliers.....	50
3.3 Data Validation.....	54
3.4 Removing Duplicates.....	55
3.5 Feature Engineering.....	56
<b>PREPROCESSING DATA.....</b>	<b>58</b>
3.6 Feature Scaling.....	58
3.7 Feature Encoding.....	59
3.8 Handling Imbalanced Dataset.....	59
<b>BAB IV PERBANDINGAN HASIL PREDIKSI.....</b>	<b>61</b>
4.1 Analisis KNN.....	61
4.2 Analisis Gaussian Naive Bayes.....	61
4.3 Kesimpulan.....	62
4.4 Saran.....	62
<b>BAB V KONTRIBUSI.....</b>	<b>63</b>
<b>REFERENSI.....</b>	<b>65</b>

# **BAB I**

## **PENDAHULUAN**

### **1.1 Latar Belakang**

Tugas Besar 2 mata kuliah IF3070 Dasar Intelegensi Buatan dirancang untuk memberikan mahasiswa pengalaman praktis dalam mengimplementasikan algoritma pembelajaran mesin pada permasalahan nyata. Pada tugas ini, mahasiswa diminta untuk membangun algoritma K-Nearest Neighbors (KNN) dan Gaussian Naive-Bayes secara manual (from *scratch*) serta membandingkan hasilnya dengan implementasi menggunakan pustaka *scikit-learn*. Dataset yang digunakan adalah PhiUSIIL *Phishing URL Dataset*, yang berisi fitur-fitur URL untuk membedakan antara URL *legitimate* dan phishing. Selain implementasi algoritma, mahasiswa juga diminta untuk melakukan tahapan *cleaning* dan *preprocessing* data agar model yang dibuat dapat menghasilkan performa yang optimal. Melalui tugas ini, diharapkan mahasiswa mampu memahami konsep pembelajaran mesin secara mendalam sekaligus mengembangkan kemampuan analisis dan evaluasi model.

### **1.2 Deskripsi Masalah**

Pada era digital saat ini, banyak pengguna internet menghadapi risiko serangan siber melalui URL *phishing* yang sulit dibedakan dengan URL *legitimate*. Untuk membantu mengatasi masalah ini, kami diminta untuk membangun model pembelajaran mesin yang dapat mendeteksi URL phishing dengan akurat menggunakan dataset PhiUSIIL Phishing URL *Dataset*. *Dataset* ini berisi berbagai fitur teknis URL, seperti struktur source code dan elemen-elemen URL, yang digunakan untuk menentukan apakah sebuah URL termasuk phishing (label 0) atau legitimate (label 1). Dalam tugas ini, kami harus mengimplementasikan algoritma K-Nearest Neighbors (KNN) dan Gaussian Naive Bayes (GNB) secara manual (from *scratch*) dan membandingkannya dengan implementasi menggunakan pustaka *scikit-learn*. Selain itu, kami juga perlu melakukan preprocessing data untuk memastikan model dapat menghasilkan performa

terbaik. Hasil dari model ini akan dievaluasi berdasarkan akurasinya, dan kami juga diminta untuk menyimpan serta memuat model agar prediksinya dapat direproduksi. Melalui tugas ini, kami belajar memahami algoritma pembelajaran mesin secara lebih mendalam sekaligus mengembangkan kemampuan analisis data dan evaluasi model.

## BAB II

### IMPLEMENTASI ALGORITMA

#### 2.1 Implementasi Algoritma K-Nearest Neighbor (KNN)

##### a. KNN *from Scratch*

K-Nearest Neighbor (KNN) adalah algoritma machine learning berbasis instance yang digunakan untuk klasifikasi dan regresi. KNN bekerja dengan cara membandingkan data yang baru dengan data yang sudah ada dalam dataset berdasarkan jarak (misalnya, jarak Euclidean, Manhattan, atau Minkowski). Dalam proses klasifikasi, data baru akan diberi label berdasarkan mayoritas kelas dari K tetangga terdekatnya, sedangkan dalam regresi, nilai output dihitung sebagai rata-rata atau median dari nilai tetangga terdekat. Algoritma ini sederhana namun efektif, terutama pada dataset kecil atau dengan distribusi data yang jelas, tetapi menjadi kurang efisien pada dataset besar karena memerlukan komputasi jarak untuk setiap instance saat prediksi. KNN juga sangat bergantung pada pilihan parameter K dan metrik jarak yang digunakan.

##### 1. Inisialisasi Kelas KNN (`__init__`)

```
def __init__(self, k=3, metric='euclidean',
weights='uniform', p=1):
    if k < 1 or not isinstance(k, int):
        raise ValueError("Invalid neighbor count. k must
be an integer greater than 0.")
    if metric not in ['manhattan', 'euclidean',
'minkowski']:
        raise ValueError("Invalid distance metric. Valid
metric: 'euclidean', 'manhattan', or 'minkowski'.")
    if weights not in ['uniform', 'distance']:
        raise ValueError("Invalid weights mode. Valid
mode: 'uniform' or 'distance'")
    if p < 1 or not isinstance(p, int):
        raise ValueError("Invalid minkowski distance
variable. p must be an integer greater than 0.")

    self.k = k
    self.metric = metric
    self.weights = weights
```

```
self.p = p
```

Metode inisialisasi untuk kelas KNN ini digunakan untuk mengatur parameter awal model K-Nearest Neighbors (KNN) sesuai dengan input pengguna. Metode ini menerima tiga parameter: *k*, *metric*, dan *p*, dengan nilai default masing-masing 3, 'euclidean', dan 1. Parameter *k* menentukan jumlah tetangga terdekat yang akan dipertimbangkan dalam proses klasifikasi atau regresi. Nilai *k* harus berupa bilangan bulat positif, dan jika nilai yang diberikan kurang dari 1 atau bukan bilangan bulat, akan dilemparkan kesalahan berupa *ValueError*. Parameter *metric* digunakan untuk menentukan metode penghitungan jarak, yang dapat berupa 'euclidean', 'manhattan', atau 'minkowski'. Jika pengguna memasukkan nilai selain ketiga opsi ini, program akan mengeluarkan kesalahan yang sama. Sementara itu, parameter *p* digunakan untuk menentukan nilai eksponen dalam penghitungan jarak Minkowski, jika metrik jarak yang dipilih adalah 'minkowski'. Nilai *p* juga harus berupa bilangan bulat positif; jika tidak, program akan mengeluarkan kesalahan. Setelah validasi parameter selesai, nilai-nilai tersebut disimpan dalam atribut *self.k*, *self.metric*, dan *self.p* untuk digunakan dalam metode-metode lain di kelas KNN.

## 2. Fungsi fit

```
def fit(self, X_train, y_train):  
    self.X_train = X_train  
    self.y_train = y_train
```

Fungsi *fit* pada kelas KNN digunakan untuk melatih model dengan data yang diberikan. Fungsi ini menerima dua parameter, yaitu *X\_train* dan *y\_train*. Parameter *X\_train* merupakan data fitur yang digunakan sebagai referensi untuk mencari tetangga terdekat saat proses prediksi, sementara *y\_train* adalah label atau target yang sesuai dengan data pada *X\_train*. Fungsi ini tidak melakukan proses pelatihan seperti algoritma pembelajaran mesin lainnya, karena KNN adalah metode berbasis instance (*lazy learning*) yang hanya menyimpan data pelatihan untuk digunakan dalam penghitungan jarak pada tahap prediksi. Setelah fungsi ini dipanggil, data pelatihan *X\_train* dan *y\_train* disimpan sebagai atribut instance, yaitu *self.X\_train* dan *self.y\_train*, sehingga dapat diakses oleh metode lain dalam kelas, seperti metode prediksi.

### 3. Fungsi predict untuk memprediksi keseluruhan titik data uji

```
def predict(self, X_test):
    if self.X_train is None or self.y_train is None:
        raise Exception("Invalid data")

    predictions = []
    for i, row in X_test.iterrows():
        match self.metric:
            case 'euclidean':
                distances =
np.sqrt(np.sum(np.square(row[np.array(X_test.columns)] -
self.X_train), axis=1))
            case 'manhattan':
                distances =
np.sum(np.abs(row[np.array(X_test.columns)] -
self.X_train), axis=1)
            case 'minkowski':
                distances =
np.sum(np.abs(row[np.array(X_test.columns)] -
self.X_train) ** self.p, axis=1) ** (1 / self.p)
            case _:
                raise ValueError("Invalid distance
metric. Valid metric: 'euclidean', 'manhattan', or
'minkowski'.")

        # Get the k nearest neighbors
        neighbors = np.argsort(distances)[:self.k]

        # Get the most common class
        classes = Counter(self.y_train.iloc[neighbors])
        max_class = classes.most_common(1)[0][0]

        predictions.append([i, max_class])
    return pd.DataFrame(predictions, columns=['id',
'label'])
```

Fungsi predict pada kelas KNN bertugas untuk melakukan prediksi berdasarkan data uji yang diberikan. Fungsi ini menerima satu parameter, yaitu `X_test`, yang merupakan data fitur yang akan diprediksi. Sebelum melanjutkan, fungsi memverifikasi apakah data pelatihan (`self.X_train` dan `self.y_train`) telah diinisialisasi melalui metode `fit`. Jika data pelatihan belum ada, fungsi akan mengeluarkan pengecualian.

Dalam proses prediksi, fungsi menghitung jarak antara setiap sampel pada `X_test` dengan semua data pada `X_train`, menggunakan metrik jarak yang ditentukan oleh

parameter `self.metric` (yaitu 'euclidean', 'manhattan', atau 'minkowski'). Jarak dihitung menggunakan formula yang sesuai: jarak Euclidean dihitung dengan akar kuadrat dari jumlah kuadrat selisih fitur, jarak Manhattan dengan jumlah nilai mutlak selisih fitur, dan jarak Minkowski dengan generalisasi eksponen `self.p`. Jika metrik jarak tidak valid, fungsi akan mengeluarkan pengecualian.

Setelah jarak dihitung, fungsi menentukan `k` tetangga terdekat dengan mengurutkan jarak dan mengambil indeks `k` terkecil. Label dari tetangga ini diperoleh dari data pelatihan `y_train`, dan label yang paling umum di antaranya dipilih sebagai prediksi untuk sampel tersebut. Proses ini diulangi untuk setiap baris di `X_test`, dan hasil prediksi dikumpulkan dalam bentuk daftar yang mencakup indeks sampel dan label prediksi.

Hasil akhir dikembalikan dalam bentuk `DataFrame` dengan dua kolom: 'id', yang berisi indeks sampel dari `X_test`, dan 'label', yang berisi hasil prediksi. Dengan demikian, fungsi ini mengimplementasikan langkah utama algoritma KNN untuk menghasilkan prediksi berdasarkan data tetangga terdekat.

#### b. KNN by *Scikit-Learn*

Dengan *scikit-learn*, proses pelatihan dan evaluasi model KNN dapat dilakukan dengan hanya beberapa baris kode, tanpa harus mengimplementasikan setiap detail perhitungan jarak atau pengurutan data secara manual. Misalnya, fungsi seperti `KNeighborsClassifier` memungkinkan untuk langsung menentukan jumlah tetangga terdekat (`K`), sementara metode seperti `fit` dan `predict` digunakan untuk pelatihan dan prediksi. Penerapan ini tidak hanya mempercepat proses pengembangan, tetapi juga memastikan akurasi dan efisiensi dengan memanfaatkan optimasi yang sudah tersedia di pustaka tersebut. Berikut adalah implementasi algoritma KNN menggunakan *scikit-learn* untuk dataset yang sudah diproses.

```
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train_numerical_all, y_train)
```



```
y_pred_knn = knn.predict(X_val)

print(f'KNN Accuracy: {accuracy_score(y_val,
y_pred_knn)}')
print(f'KNN F1 Score: {f1_score(y_val, y_pred_knn,
average="macro")}')
print(classification_report(y_val, y_pred_knn))
```

Kode di atas menunjukkan langkah-langkah membuat model KNN menggunakan *scikit-learn*. Parameter *n\_neighbors* menentukan jumlah tetangga terdekat, sedangkan *fit* digunakan untuk melatih model dengan data latih. Selanjutnya, fungsi *predict* dapat memprediksi label data validasi dan hasil evaluasi disajikan dalam bentuk metrik seperti akurasi, *F1 score* dan laporan klasifikasi. Implementasi ini menggambarkan kepraktisan penggunaan *library modern* dalam mengembangkan model pembelajaran mesin.

## 2.2 Implementasi Algoritma *Naive Bayes*

### a. *Naive Bayes from Scratch*

*Naive Bayes* adalah algoritma pembelajaran mesin yang berbasis pada *Teorema Bayes*, dengan asumsi independensi yang kuat (atau "*naive*") antara fitur-fitur yang digunakan dalam model. Algoritma ini sering digunakan dalam kasus klasifikasi, terutama deteksi *phishing* URL. *Naive Bayes* bekerja dengan menghitung probabilitas posterior dari setiap kelas untuk *instance* baru berdasarkan data pelatihan, dan prediksi kelas dibuat dengan memilih kelas yang memiliki probabilitas tertinggi. Pembuatan algoritma ini dimulai dengan menghitung probabilitas prior untuk setiap kelas serta probabilitas kondisional dari fitur-fitur yang ada terhadap kelas tertentu.

```
import math

class NaiveBayes:
    def __init__(self):
        self.target = 'label'
        self.pv = {}
        self.table = {}

    def _calculate_probability(self, x, mean, std):
        if std == 0:
```

```

        std = 1e-10
        exponent = -((x - mean) ** 2) / (2 * std ** 2)
        coefficient = 1 / (math.sqrt(2 * math.pi) * std)
        probability = coefficient * math.exp(exponent)
        return probability

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

        for i in range(2):
            self.pv[i] = len(y_train[y_train == i]) /
len(y_train)

        data = X_train.join(y_train)

        for feature in np.array(X_train.columns):
            for k in range(2):
                k_data = data[data[self.target] == k]
                if not k_data.empty:
                    mean = k_data[feature].mean()
                    std = k_data[feature].std()
                    self.table[(feature, k)] = {'mean':
mean, 'std': std}

    def predict(self, X_test):
        predictions = []
        for i, row in X_test.iterrows():
            class_probability = []
            for k in range(2):
                p = self.pv[k]
                for feature in X_test.columns:
                    if feature in
np.array(self.X_train.columns):
                        value = row[feature]
                        mean = self.table[(feature,
k)][['mean']]
                        std = self.table[(feature,
k)][['std']]
                        p *=
self._calculate_probability(value, mean, std)
                    else:
                        value = row[feature]
                        if (feature, value, k) in
self.table:
                            p *= self.table[(feature,
value, k)]

                        class_probability.append(p)
            predictions.append([i,
np.argmax(class_probability)])

```

```
return pd.DataFrame(predictions, columns=['id',
self.target])
```

## 1. Kelas NaiveBayes

Kelas ini mengimplementasikan algoritma **Gaussian Naive Bayes** dari awal. Tujuan utamanya adalah membangun model klasifikasi berbasis probabilistik menggunakan asumsi bahwa setiap fitur bersifat independen.

## 2. Inisialisasi `__init__`

```
def __init__(self):
    self.target = 'label'
    self.pv = {}
    self.table = {}
```

**self.target:** Nama kolom target dalam dataset, default-nya adalah 'label'.

**self.pv:** Menyimpan probabilitas prior dari setiap kelas.

**self.table:** Menyimpan parameter distribusi (mean dan standar deviasi) untuk setiap fitur dalam setiap kelas.

## 3. Fungsi `_calculate_probability`

```
def _calculate_probability(self, x, mean, std):
    if std == 0:
        std = 1e-10
    exponent = -((x - mean) ** 2) / (2 * std ** 2)
    coefficient = 1 / (math.sqrt(2 * math.pi) * std)
    probability = coefficient * math.exp(exponent)
    return probability
```

Fungsi ini menghitung probabilitas *Gaussian (Normal Distribution)* menggunakan formula distribusi normal.

- **x:** Nilai fitur.
- **mean:** Rata-rata fitur dalam kelas tertentu.
- **std:** Standar deviasi fitur dalam kelas tertentu.

Penanganan *Zero Variance*: Jika std adalah nol, nilai kecil (1e-10) digunakan untuk menghindari pembagian dengan nol.

## 4. Fungsi `fit`

```
def fit(self, X_train, y_train):  
    self.X_train = X_train  
    self.y_train = y_train
```

Fungsi fit digunakan untuk melatih model Naive Bayes dengan menghitung probabilitas prior setiap kelas serta parameter distribusi Gaussian (mean dan standar deviasi) untuk setiap fitur di setiap kelas. Data pelatihan, terdiri dari fitur `X_train` dan target `y_train`, disimpan dalam atribut internal untuk referensi selanjutnya. Probabilitas prior dihitung berdasarkan proporsi jumlah data dari setiap kelas dibandingkan dengan total data, memberikan dasar untuk menghitung probabilitas posterior pada tahap prediksi.

### Menghitung Probabilitas Prior

```
for i in range(2):  
    self.pv[i] = len(y_train[y_train == i]) /  
    len(y_train)
```

Probabilitas prior dihitung sebagai proporsi jumlah data dari setiap kelas terhadap total data.

### Menghitung Parameter Gaussian

```
data = X_train.join(y_train)  
  
for feature in np.array(X_train.columns):  
    for k in range(2):  
        k_data = data[data[self.target] == k]  
        if not k_data.empty:  
            mean = k_data[feature].mean()  
            std = k_data[feature].std()  
            self.table[(feature, k)] = {'mean': mean,  
            'std': std}
```

Selanjutnya, fungsi fit menghitung mean dan standar deviasi untuk setiap fitur dalam setiap kelas. Data fitur dan label digabungkan menggunakan fungsi `join`, sehingga memungkinkan pengelompokan data berdasarkan kelas. Untuk setiap fitur dan kelas, mean serta standar deviasi dihitung dan disimpan dalam dictionary `self.table` dengan format (feature, class) sebagai kunci. Jika data pada kelas tertentu kosong, penghitungan parameter dilewati untuk menghindari error. Parameter ini nantinya digunakan dalam perhitungan probabilitas Gaussian selama prediksi.

## 5. Fungsi **predict**

```
def predict(self, X_test):  
    predictions = []
```

Fungsi **predict** digunakan untuk menghasilkan prediksi berdasarkan data uji (**X\_test**) dengan memanfaatkan model yang telah dilatih sebelumnya. Proses ini melibatkan perhitungan probabilitas posterior untuk setiap kelas menggunakan aturan Bayes, di mana probabilitas posterior dihitung dengan mengalikan probabilitas prior kelas dan probabilitas setiap fitur dalam kelas tersebut. Hasil prediksi adalah kelas dengan probabilitas posterior tertinggi.

### Iterasi Melalui Data Uji

```
for i, row in X_test.iterrows():  
    class_probability = []  
    for k in range(2):  
        p = self.pv[k]
```

Bagian ini memulai iterasi untuk setiap baris di **X\_test** menggunakan **iterrows**. Pada setiap iterasi, probabilitas posterior untuk setiap kelas (dalam hal ini  $k = 0$  dan  $k = 1$ ) dihitung. Probabilitas awal diatur ke probabilitas prior (**self.pv[k]**).

### Perhitungan Probabilitas Posterior

```
for feature in X_test.columns:  
    if feature in np.array(self.X_train.columns):  
        value = row[feature]  
        mean = self.table[(feature, k)]['mean']  
        std = self.table[(feature, k)]['std']  
        p *= self._calculate_probability(value, mean,  
std)
```

Setiap fitur dalam baris data uji digunakan untuk memperbarui probabilitas posterior. Probabilitas dihitung berdasarkan distribusi Gaussian dengan mean dan standar deviasi yang dihitung selama pelatihan. Jika fitur tersebut ditemukan di data pelatihan (**self.X\_train**), maka probabilitas posterior diperbarui menggunakan fungsi **\_calculate\_probability**.

### Menentukan Kelas dengan Probabilitas Tertinggi

```
class_probability.append(p)
```

```
predictions.append([i, np.argmax(class_probability)])
```

Setelah probabilitas posterior untuk setiap kelas dihitung, nilai probabilitas tersebut disimpan dalam `class_probability`. Kemudian, kelas dengan probabilitas posterior tertinggi ditentukan menggunakan `np.argmax(class_probability)`. Prediksi ini disimpan bersama dengan indeks baris dalam `predictions`, yang nantinya akan dikembalikan sebagai output fungsi.

## 6. *Output Hasil Prediksi*

Bagian ini mengembalikan hasil prediksi dalam bentuk DataFrame yang memuat dua kolom:

1. **id**: Indeks dari data uji yang diproses.
2. **self.target**: Label kelas hasil prediksi untuk setiap baris data uji.

```
return pd.DataFrame(predictions, columns=['id',  
self.target])
```

Prediksi yang telah dihitung untuk setiap baris data uji disusun ke dalam DataFrame untuk mempermudah pengolahan lebih lanjut, seperti analisis hasil atau evaluasi performa model.

### b. *Naive Bayes by Scikit-Learn*

Dengan *scikit-learn*, proses pembuatan model *Naive Bayes* menjadi lebih sederhana dan terstruktur. *Naive Bayes* adalah algoritma klasifikasi berbasis probabilitas yang memanfaatkan Teorema Bayes dengan asumsi independensi antar fitur. Algoritma ini sering digunakan untuk masalah klasifikasi teks, deteksi spam, dan prediksi sederhana karena efisiensi komputasinya. Pada implementasi ini, digunakan Gaussian Naive Bayes, yang cocok untuk fitur numerik dengan distribusi normal.

```
from sklearn.model_selection import train_test_split  
from sklearn.naive_bayes import GaussianNB  
from sklearn.metrics import accuracy_score,  
classification_report, confusion_matrix
```

```
model = GaussianNB()

model.fit(X_train, y_train)
y_pred = model.predict(X_val)

print("Accuracy:", accuracy_score(y_val, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val,
y_pred))
print("Classification Report:\n",
classification_report(y_val, y_pred))
```

Kode di atas menunjukkan langkah-langkah penerapan Gaussian Naive Bayes menggunakan *scikit-learn*. Model **GaussianNB** diinisialisasi dan dilatih menggunakan dataset **X\_train** dan target **y\_train**. Setelah model dilatih, dilakukan prediksi pada dataset validasi **X\_val**. Hasil evaluasi model mencakup nilai akurasi, *confusion matrix*, dan *classification report*, yang memberikan gambaran tentang performa model, seperti akurasi keseluruhan, presisi, dan *recall* untuk setiap kelas. Dengan metrik ini, pengguna dapat memahami sejauh mana model mampu mengklasifikasikan data dengan baik.

## **BAB III**

### ***CLEANING DAN PREPROCESSING DATA***

#### ***CLEANING DATA***

##### ***3.1 Handling Missing Data***

Dalam proses persiapan data, penanganan nilai kosong merupakan langkah krusial untuk memastikan kelengkapan dan akurasi dataset. Setiap kolom dalam dataset ini merepresentasikan fitur-fitur spesifik yang terkait dengan deteksi URL phishing, seperti yang dijelaskan dalam penelitian ["PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning"](#). Deskripsi dan metodologi yang dijabarkan dalam penelitian tersebut menjadi panduan utama untuk mengisi nilai kosong, memastikan metode imputasi yang digunakan sesuai dengan semantik dan pola dari setiap fitur.

Teknik yang kami pilih untuk menangani data yang hilang adalah kombinasi dari ***Mean, Median, or Mode Imputation*** dan ***Domain-Specific Strategies***.

Alasan kami memilih beberapa teknik tersebut, yaitu:

a. ***Mean, Median, or Mode Imputation:***

Untuk data numerik, kami menganalisis distribusi data (*skewness*) untuk menentukan apakah distribusi normal atau tidak. Jika distribusi normal, kami menggunakan *mean* sebagai nilai imputasi. Jika data menunjukkan *skewness* yang tinggi, *median* digunakan untuk mengurangi bias. Strategi ini sederhana dan efektif, terutama karena sebagian besar nilai yang hilang berasal dari data yang hilang secara acak (*Missing at Random/MAR*).

b. ***Mode Imputation:***

Untuk data kategorikal, kami mengisi nilai kosong berdasarkan mode dari data yang dikelompokkan menggunakan kolom terkait, yaitu kolom kategorikal lainnya yang sudah lengkap dan terisi (*grouping*). Hal ini memastikan imputasi mempertimbangkan pola data di setiap kolom kategorikal tetap terjaga.



c. ***Domain-Specific Strategies***:

Untuk 26 kolom dalam dataset, kami menggunakan logika berbasis domain berdasarkan pemahaman mendalam tentang sifat fitur tersebut. Sebagai contoh, `IsDomainIP` dihitung berdasarkan pola IP address yang valid, sedangkan `TLDLegitimateProb` diimputasi menggunakan probabilitas rata-rata berbasis TLD yang telah diketahui.

d. **Penghindaran Penghapusan Data (Deletion)**:

Kami menghindari teknik seperti *Listwise Deletion* karena dapat menyebabkan kehilangan informasi penting, terutama dalam dataset besar dengan nilai yang beragam. Walaupun data seperti URL, *Domain*, TLD, dan *Title* memiliki cukup banyak data yang hilang, kami memanfaatkan data yang ada pada baris terkait, agar dapat memaksimalkan hasil prediksi.

Kami tidak menggunakan ***Constant Value Imputation*** karena metode ini berpotensi menciptakan bias jika nilai konstan, seperti 0, tidak relevan atau tidak sesuai dengan konteks data. Misalnya, untuk data kategorikal atau numerik tertentu, pengisian dengan nilai konstan dapat mengaburkan pola yang ada dalam data asli. Selain itu, kami juga tidak menggunakan ***Imputation Using Predictive Models***, meskipun metode ini sering menghasilkan hasil yang lebih presisi, karena kompleksitas dan waktu komputasinya yang tinggi. Dalam kasus ini, metode yang lebih sederhana seperti imputasi berbasis *mean*, *median*, atau *mode* sudah cukup efektif untuk menjaga kualitas data.

Bagian berikut akan menjelaskan strategi imputasi untuk setiap kolom, memastikan nilai-nilai kosong ditangani dengan tepat. Logika imputasi meliputi pengisian nilai berdasarkan kolom terkait hingga penggunaan metode statistik seperti mean atau median, tergantung pada distribusi kolom tersebut. Berikut adalah implementasi langkah-langkah pengisian nilai kosong:

## I. Implementasi *Domain-Specific Strategies*

Fungsi dalam Implementasi *Domain-Specific Strategies* akan diimplementasikan pada *dataset* **X\_train** dan **X\_val** dengan memproses setiap baris menggunakan fungsi **apply**. Setelah fungsi dijalankan, kolom terkait di *dataset* akan terisi dengan sesuai dengan kolom lainnya yang berhubungan.

### a. Kolom URLLength

```
def fill_url_length(data, url_col='URL',
url_length_col='URLLength'):
    data[url_length_col] = data.apply(
        lambda row: len(str(row[url_col])) if
pd.isnull(row[url_length_col]) and
pd.notnull(row[url_col]) else row[url_length_col],
        axis=1
    )
    return data
```

### Implementasi

```
X_train = fill_url_length(X_train, url_col='URL',
url_length_col='URLLength')
X_train[['URL', 'URLLength']]
```

```
X_val = fill_url_length(X_val, url_col='URL',
url_length_col='URLLength')
X_val[['URL', 'URLLength']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **URLLength** dalam *dataset*. Fungsi **fill\_url\_length** mengambil dua parameter utama: **url\_col**, yang menunjukkan kolom yang berisi data *URL*, dan **url\_length\_col**, yang menunjukkan kolom tempat panjang *URL* akan disimpan. Fungsi ini memeriksa apakah suatu nilai di kolom **URLLength** kosong (*NaN*) dan apakah nilai di kolom **URL** pada baris yang sama tersedia (tidak kosong). Jika kedua kondisi tersebut terpenuhi, maka panjang string dari *URL* dihitung menggunakan fungsi **len** dan hasilnya disimpan ke kolom **URLLength**. Jika nilai

di **URLLength** sudah ada, maka nilai tersebut dipertahankan. Fungsi ini diimplementasikan pada *dataset X\_train dan X\_val* dengan memproses setiap baris menggunakan fungsi **apply**. Setelah fungsi dijalankan, kolom **URLLength** di *dataset* akan terisi dengan panjang *URL* untuk semua baris yang memiliki data *URL* yang valid.

## b. Kolom Domain

```
def fill_domain(data, url_col='URL',
                domain_col='Domain'):
    def get_domain(url):
        return urlparse(url).netloc if pd.notnull(url)
    else None

    data[domain_col] = data.apply(
        lambda row: get_domain(row[url_col]) if
        pd.isnull(row[domain_col]) and pd.notnull(row[url_col])
        else row[domain_col],
        axis=1
    )
    return data
```

## Implementasi

```
X_train = fill_domain(X_train, url_col='URL',
                      domain_col='Domain')
X_train[['URL', 'Domain']]
```

```
X_val = fill_domain(X_val, url_col='URL',
                    domain_col='Domain')
X_val[['URL', 'Domain']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **Domain** dalam *dataset* dengan cara mengekstrak *domain* dari kolom **URL**. Fungsi **fill\_domain** menerima parameter **url\_col** untuk menunjukkan kolom yang berisi data *URL*, dan **domain\_col** untuk kolom yang akan menyimpan hasil ekstraksi *domain*. Di dalam fungsi ini, terdapat fungsi bantu **get\_domain** yang menggunakan **urlparse** untuk mengekstrak *netloc* dari *URL* jika nilai *URL* tidak kosong (*not null*). Kemudian, fungsi **apply** digunakan untuk memproses setiap

baris data: jika kolom **Domain** kosong (*NaN*) dan kolom **URL** berisi nilai yang valid, maka fungsi **get\_domain** dipanggil untuk mengisi nilai **Domain**. Jika nilai pada **Domain** sudah ada, maka nilai tersebut dipertahankan. Fungsi ini memastikan kolom **Domain** terisi sesuai dengan informasi yang diekstrak dari kolom **URL**.

### c. Kolom DomainLength

```
def fill_domain_length(data, domain_col='Domain',
                        domain_length_col='DomainLength'):
    data[domain_length_col] = data.apply(
        lambda row: len(str(row[domain_col]))
        if pd.isnull(row[domain_length_col]) and
        pd.notnull(row[domain_col])
        else row[domain_length_col],
        axis=1
    )
    return data
```

### Implementasi

```
X_train = fill_domain_length(X_train,
                              domain_col='Domain', domain_length_col='DomainLength')
X_train[['Domain', 'DomainLength']]
```

```
X_val = fill_domain_length(X_val, domain_col='Domain',
                            domain_length_col='DomainLength')
X_val[['Domain', 'DomainLength']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **DomainLength** dalam *dataset* dengan menghitung panjang string dari kolom **Domain**. Fungsi **fill\_domain\_length** menerima parameter **domain\_col**, yang menunjuk kolom yang berisi data *domain*, dan **domain\_length\_col**, yang akan menyimpan panjang dari *domain* tersebut. Dengan menggunakan fungsi **apply**, setiap baris diperiksa: jika kolom **DomainLength** kosong (*NaN*) dan kolom **Domain** memiliki nilai yang valid, maka panjang string dari nilai pada kolom **Domain** dihitung menggunakan fungsi **len** dan disimpan ke kolom **DomainLength**. Jika nilai **DomainLength** sudah ada, maka nilai tersebut tidak

diubah. Setelah dijalankan, kode ini memastikan kolom **DomainLength** terisi dengan panjang dari *domain* yang valid di setiap baris *dataset*.

#### d. Kolom IsDomainIP

```
def fill_is_domain_ip(data, domain_col='Domain',
is_domain_ip_col='IsDomainIP'):
    def is_ipaddress(domain):
        if pd.notnull(domain):
            ip_pattern = r'^(\d{1,3}\.){3}\d{1,3}$'
            if re.match(ip_pattern, domain):
                parts = domain.split('.')
                return all(0 <= int(part) <= 255 for part
in parts)
            return False

    data[is_domain_ip_col] = data.apply(
        lambda row: (1 if is_ipaddress(row[domain_col])
else 0)
        if pd.isnull(row[is_domain_ip_col]) and
pd.notnull(row[domain_col])
        else row[is_domain_ip_col],
        axis=1
    )
    return data
```

#### Implementasi

```
X_train = fill_is_domain_ip(X_train, domain_col='Domain',
is_domain_ip_col='IsDomainIP')
X_train[['Domain', 'IsDomainIP']]
```

```
X_val = fill_is_domain_ip(X_val, domain_col='Domain',
is_domain_ip_col='IsDomainIP')
X_val[['Domain', 'IsDomainIP']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **IsDomainIP** dalam *dataset* dengan menentukan apakah nilai dalam kolom **Domain** merupakan alamat IP yang valid. Fungsi **fill\_is\_domain\_ip** menerima parameter **domain\_col** sebagai kolom yang berisi data *domain*, dan **is\_domain\_ip\_col** sebagai kolom yang akan menyimpan indikator apakah *domain* tersebut adalah alamat IP. Di dalamnya, fungsi bantu **is\_ipaddress** memeriksa apakah *domain* sesuai dengan pola alamat IPv4 menggunakan *regular expression*

dan memastikan setiap segmen bernilai antara 0 hingga 255. Selanjutnya, fungsi **apply** digunakan untuk memproses setiap baris: jika **IsDomainIP** kosong dan **Domain** berisi nilai, maka **is\_ipaddress** menentukan apakah **Domain** adalah alamat IP; jika ya, **IsDomainIP** diisi dengan 1, jika tidak, diisi dengan 0. Setelah dijalankan, kolom **IsDomainIP** akan terisi dengan indikator biner yang menunjukkan apakah *domain* pada setiap baris merupakan alamat IP yang valid.

**e. Kolom TLD**

```
def fill_tld(data, domain_col='Domain', tld_col='TLD'):  
    def generate_tld(domain):  
        if pd.notnull(domain):  
            parts = domain.split('.')  
            if len(parts) > 1:  
                return parts[-1]  
            return np.nan  
  
    data[tld_col] = data.apply(  
        lambda row: generate_tld(row[domain_col])  
        if pd.isnull(row[tld_col])  
        else row[tld_col],  
        axis=1  
    )  
    return data
```

**Implementasi**

```
X_train = fill_tld(X_train, domain_col='Domain',  
tld_col='TLD')  
X_train[['URL', 'Domain', 'TLD']]
```

```
X_val = fill_tld(X_val, domain_col='Domain',  
tld_col='TLD')  
X_val[['URL', 'Domain', 'TLD']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **TLD** dalam *dataset* dengan mengekstrak *Top-Level Domain* (TLD) dari kolom **Domain**. Fungsi **fill\_tld** menerima parameter **domain\_col** sebagai kolom yang berisi data *domain*, dan **tld\_col** sebagai kolom yang akan menyimpan TLD yang diekstraksi. Di dalamnya, fungsi bantu **generate\_tld** memeriksa apakah *domain* tidak kosong, kemudian memisahkan *domain* berdasarkan tanda titik ('.')

dan mengembalikan bagian terakhir sebagai TLD jika terdapat lebih dari satu bagian. Selanjutnya, fungsi **apply** digunakan untuk memproses setiap baris: jika kolom **TLD** kosong dan **Domain** berisi nilai, maka **generate\_tld** dipanggil untuk mengekstrak TLD; jika tidak, nilai **TLD** dipertahankan. Setelah dijalankan, kolom **TLD** akan terisi dengan *Top-Level Domain* yang sesuai untuk setiap baris dalam *dataset*.

#### f. Kolom CharContinuationRate

```
def fill_char_continuation_rate(data, url_col='URL',
                                char_rate_col='CharContinuationRate'):
    def generate_char_continuation_rate(url):
        if pd.notnull(url):
            sequences = re.findall(r'[a-zA-Z0-9_]+', url)
            total_sequence_length = sum(len(seq) for seq
in sequences)
            total_url_length = len(url)
            return total_sequence_length /
total_url_length if total_url_length > 0 else np.nan
        return np.nan

    data[char_rate_col] = data.apply(
        lambda row:
generate_char_continuation_rate(row[url_col])
        if pd.isnull(row[char_rate_col]) else
row[char_rate_col],
        axis=1
    )
    return data
```

#### Implementasi

```
X_train = fill_char_continuation_rate(X_train,
url_col='URL', char_rate_col='CharContinuationRate')
X_train[['URL', 'CharContinuationRate']]
```

```
X_val= fill_char_continuation_rate(X_val, url_col='URL',
char_rate_col='CharContinuationRate')
X_val[['URL', 'CharContinuationRate']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **CharContinuationRate** dalam *dataset* dengan menghitung rasio karakter alfanumerik berurutan dalam URL. Fungsi **fill\_char\_continuation\_rate**

menerima parameter **url\_col** sebagai kolom yang berisi data URL, dan **char\_rate\_col** sebagai kolom yang akan menyimpan rasio karakter berurutan. Di dalamnya, fungsi bantu **generate\_char\_continuation\_rate** memeriksa apakah URL tidak kosong, kemudian menggunakan *regular expression* untuk menemukan semua urutan karakter alfanumerik dalam URL. Panjang total urutan ini dibandingkan dengan panjang total URL untuk menghitung rasio karakter berurutan. Selanjutnya, fungsi **apply** digunakan untuk memproses setiap baris: jika kolom **CharContinuationRate** kosong, maka **generate\_char\_continuation\_rate** dipanggil untuk menghitung rasio; jika tidak, nilai **CharContinuationRate** dipertahankan. Setelah dijalankan, kolom **CharContinuationRate** akan terisi dengan rasio karakter alfanumerik berurutan untuk setiap URL dalam *dataset*.

#### g. Kolom TLDLegitimateProb

```
# Distribution analysis to determine the imputation
method
def fill_tld_legitimate_prob(data, tld_col='TLD',
tld_prob_col='TLDLegitimateProb',
is_domain_ip_col='IsDomainIP'):

    skewness = data[tld_prob_col].skew()
    if skewness > 1 or skewness < -1:
        global_fill_value = data[tld_prob_col].median()
        print("Uses median for imputation because
distribution is skewed.")
    else:
        global_fill_value = data[tld_prob_col].mean()
        print("Uses the mean for imputation because of
the normal distribution.")

    print(f"Global fill value: {global_fill_value}")

    tld_prob_mean =
data.groupby(tld_col)[tld_prob_col].mean()

    def fill_tld_legit_prob(row):
        if pd.isnull(row[tld_prob_col]):
            if row[is_domain_ip_col] == 1:
                return 0
```



```

        if pd.notnull(row[tld_col]) and row[tld_col]
in tld_prob_mean.index: # TLD not found
            return tld_prob_mean[row[tld_col]]
        if pd.isnull(row[tld_col]) and
pd.isnull(row[is_domain_ip_col]): # TLD dan IsDomainIP
not found
            return global_fill_value
        return row[tld_prob_col]
    data[tld_prob_col] = data.apply(fill_tld_legit_prob,
axis=1)
    data[tld_prob_col] =
data[tld_prob_col].fillna(global_fill_value)

    return data

```

## Implementasi

```

X_train = fill_tld_legitimate_prob(
    X_train,
    tld_col='TLD',
    tld_prob_col='TLDLegitimateProb',
    is_domain_ip_col='IsDomainIP'
)
X_train[['TLD', 'TLDLegitimateProb']]

```

```

X_val = fill_tld_legitimate_prob(
    X_val,
    tld_col='TLD',
    tld_prob_col='TLDLegitimateProb',
    is_domain_ip_col='IsDomainIP'
)
X_val[['TLD', 'TLDLegitimateProb']]

```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **TLDLegitimateProb** dalam *dataset* dengan mempertimbangkan distribusi data dan informasi terkait kolom **TLD** serta **IsDomainIP**. Fungsi **fill\_tld\_legitimate\_prob** pertama-tama menghitung nilai skewness dari kolom **TLDLegitimateProb** untuk menentukan apakah distribusinya miring atau normal. Jika skewness lebih besar dari 1 atau kurang dari -1, distribusi dianggap miring, dan nilai median digunakan sebagai pengganti global; jika tidak, nilai mean digunakan. Selanjutnya, rata-rata **TLDLegitimateProb** dihitung untuk setiap **TLD**. Fungsi **fill\_tld\_legit\_prob** kemudian diterapkan pada setiap baris:

jika **TLDLegitimateProb** kosong dan **IsDomainIP** bernilai 1, maka diisi dengan 0; jika **TLD** valid dan ditemukan dalam rata-rata yang dihitung, nilai tersebut digunakan; jika **TLD** dan **IsDomainIP** keduanya kosong, nilai global digunakan. Setelah itu, setiap nilai kosong yang tersisa diisi dengan nilai global. Proses ini memastikan bahwa kolom **TLDLegitimateProb** terisi dengan nilai yang sesuai berdasarkan distribusi data dan informasi terkait **TLD** serta **IsDomainIP**.

#### h. Kolom URLCharProb

```
# Function to calculate character probabilities
def calculate_char_prob(df, url_col):
    char_count = {}
    total_chars = 0
    for url in df[url_col].dropna():
        for char in url.lower():
            if char.isalnum(): # Only consider
alphanumeric characters
                char_count[char] = char_count.get(char,
0) + 1
                total_chars += 1
    return {char: count / total_chars for char, count in
char_count.items()}

# Function to calculate URLCharProb
def fill_url_char_prob(data, url_col='URL',
char_prob_col='URLCharProb', char_prob=None):
    if char_prob is None:
        raise ValueError("Character probabilities
(`char_prob`) must be provided.")

    def calculate_url_char_prob(url):
        if pd.notnull(url):
            total_prob = sum(char_prob.get(char, 0) for
char in url.lower() if char.isalnum())
            n = len(url)
            return total_prob / n if n > 0 else np.nan
        return np.nan

    data[char_prob_col] = data.apply(
        lambda row: calculate_url_char_prob(row[url_col])
        if pd.isnull(row[char_prob_col]) else
row[char_prob_col],
        axis=1
    )
    return data
```

## Implementasi

```
char_prob = calculate_char_prob(X_train, url_col='URL')
X_train = fill_url_char_prob(X_train, url_col='URL',
                             char_prob=char_prob)
X_train[['URL', 'URLCharProb']]
```

```
char_prob = calculate_char_prob(X_train, url_col='URL')
X_val = fill_url_char_prob(X_val, url_col='URL',
                           char_prob=char_prob)
X_val[['URL', 'URLCharProb']]
```

Kode ini terdiri dari dua fungsi utama yang bertujuan mengisi nilai kosong (*missing values*) pada kolom **URLCharProb** berdasarkan probabilitas kemunculan karakter alfanumerik dalam URL. Fungsi **calculate\_char\_prob** menghitung distribusi probabilitas untuk setiap karakter alfanumerik dalam kolom **URL** dengan membagi jumlah kemunculan setiap karakter dengan total karakter. Probabilitas yang dihasilkan digunakan dalam fungsi **fill\_url\_char\_prob** untuk menghitung rata-rata probabilitas setiap URL dengan menjumlahkan probabilitas masing-masing karakter dalam URL, lalu membaginya dengan panjang URL. Jika kolom **URLCharProb** kosong, nilai ini dihitung; jika tidak, nilai sebelumnya dipertahankan. Dengan demikian, kolom **URLCharProb** terisi dengan representasi probabilitas karakter dalam URL, yang membantu mendeteksi pola URL yang mungkin mencurigakan.

### i. Kolom TLDDLength

```
def fill_tld_length(data, tld_col='TLD',
                    tld_length_col='TLDDLength'):
    def calculate_tld_length(tld):
        if pd.notnull(tld):
            return len(str(tld))
        return np.nan

    data[tld_length_col] = data.apply(
        lambda row: calculate_tld_length(row[tld_col])
        if pd.isnull(row[tld_length_col]) else
        row[tld_length_col],
        axis=1
```

```
)
return data
```

## Implementasi

```
X_train = fill_tld_length(X_train, tld_col='TLD',
tld_length_col='TLDDLength')
X_train[['TLD', 'TLDDLength']]
```

```
X_val = fill_tld_length(X_val, tld_col='TLD',
tld_length_col='TLDDLength')
X_val[['TLD', 'TLDDLength']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **TLDDLength** dalam *dataset* dengan menghitung panjang dari *Top-Level Domain* (TLD) yang terdapat dalam kolom **TLD**. Fungsi **fill\_tld\_length** menerima parameter **tld\_col** sebagai kolom yang berisi data TLD, dan **tld\_length\_col** sebagai kolom yang akan menyimpan panjang TLD yang dihitung. Di dalamnya, fungsi bantu **calculate\_tld\_length** memeriksa apakah TLD tidak kosong, kemudian mengembalikan panjang dari TLD tersebut. Selanjutnya, fungsi **apply** digunakan untuk memproses setiap baris: jika kolom **TLDDLength** kosong, maka **calculate\_tld\_length** dipanggil untuk menghitung panjang TLD; jika tidak, nilai **TLDDLength** dipertahankan. Setelah dijalankan, kolom **TLDDLength** akan terisi dengan panjang dari setiap TLD yang sesuai untuk setiap baris dalam *dataset*.

### j. Kolom NoOfSubDomain

```
def fill_no_of_subdomains(data, domain_col='Domain',
subdomain_col='NoOfSubDomain'):
    def calculate_no_of_subdomains(domain):
        if pd.notnull(domain):
            parts = domain.split('.')
            return len(parts) - 2 if len(parts) > 2 else
0
            return np.nan

    data[subdomain_col] = data.apply(
```

```

        lambda row:
calculate_no_of_subdomains(row[domain_col])
        if pd.isnull(row[subdomain_col]) else
row[subdomain_col],
        axis=1
    )
    return data

```

## Implementasi

```

X_train = fill_no_of_subdomains(X_train,
domain_col='Domain', subdomain_col='NoOfSubDomain')
X_train[['Domain', 'NoOfSubDomain']]

```

```

X_val = fill_no_of_subdomains(X_val, domain_col='Domain',
subdomain_col='NoOfSubDomain')
X_val[['Domain', 'NoOfSubDomain']]

```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **NoOfSubDomain** dalam *dataset* dengan menghitung jumlah subdomain dari setiap entri di kolom **Domain**. Fungsi **fill\_no\_of\_subdomains** menerima parameter **domain\_col** sebagai kolom yang berisi data domain, dan **subdomain\_col** sebagai kolom yang akan menyimpan jumlah subdomain yang dihitung. Di dalamnya, fungsi bantu **calculate\_no\_of\_subdomains** memeriksa apakah domain tidak kosong; jika tidak, domain dipisahkan berdasarkan tanda titik ('.'). Jika jumlah bagian yang dihasilkan lebih dari dua, fungsi mengembalikan jumlah bagian dikurangi dua sebagai jumlah subdomain; jika tidak, mengembalikan nol. Selanjutnya, metode **apply** digunakan untuk memproses setiap baris: jika kolom **NoOfSubDomain** kosong, maka **calculate\_no\_of\_subdomains** dipanggil untuk menghitung jumlah subdomain; jika tidak, nilai **NoOfSubDomain** dipertahankan. Setelah dijalankan, kolom **NoOfSubDomain** akan terisi dengan jumlah subdomain yang sesuai untuk setiap domain dalam *dataset*.

### k. Kolom HasObfuscation

```

def fill_has_obfuscation(data, url_col='URL',
obfuscation_col='HasObfuscation'):
    def detect_advanced_obfuscation(url):
        if pd.notnull(url):
            if len(re.findall(r'[_]', url)) > 3: # Rule
1: Too many special characters
                return 1
            if re.search(r'[a-zA-Z]+\d+|\d+[a-zA-Z]+',
url): # Rule 2: Mixed alphanumeric patterns
                return 1
            if len(url) % 4 == 0 and
re.match(r'^[A-Za-z0-9+/]*={0,2}$', url): # Rule 3:
Base64 encoded strings
                try:
                    base64.b64decode(url, validate=True)
                    return 1
                except Exception:
                    pass
            reversed_url = url[::-1] # Rule 4: Reversed
strings
            if re.match(r'^[a-zA-Z0-9.\-]+\$',
reversed_url):
                return 1
            if not re.search(r'[a-zA-Z]{3,}', url): #
Rule 5: Randomized strings without meaningful words
                return 1
            return 0 # No obfuscation detected
        return 0 # Missing values treated as no
obfuscation

    data[obfuscation_col] = data.apply(
        lambda row:
detect_advanced_obfuscation(row[url_col])
        if pd.isnull(row[obfuscation_col]) and
pd.notnull(row[url_col])
        else row[obfuscation_col],
        axis=1
    )
    data[obfuscation_col] =
data[obfuscation_col].apply(lambda x: 1 if x == True else
0)

    return data

```

## Implementasi

```

X_train = fill_has_obfuscation(X_train, url_col='URL',
obfuscation_col='HasObfuscation')
X_train[['URL', 'HasObfuscation']]

```

```
X_val = fill_has_obfuscation(X_val, url_col='URL',
obfuscation_col='HasObfuscation')
X_val[['URL', 'HasObfuscation']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **HasObfuscation** dalam *dataset*. Kolom ini menunjukkan apakah sebuah URL memiliki elemen *obfuscation* (penyamaran) atau tidak, yang sering digunakan untuk mengelabui pengguna atau mendeteksi pola URL yang mencurigakan. Fungsi **fill\_has\_obfuscation** memanfaatkan fungsi bantu **detect\_advanced\_obfuscation** untuk mendeteksi pola berdasarkan lima aturan utama, seperti banyaknya karakter khusus, pola campuran huruf dan angka, string base64 yang valid, URL yang dibalik (*reversed*), atau kurangnya kata bermakna dalam URL. Nilai pada kolom **HasObfuscation** akan diisi dengan 1 jika pola *obfuscation* terdeteksi atau 0 jika tidak ada pola yang mencurigakan. Nilai yang sudah ada di kolom **HasObfuscation** tidak akan diubah. Implementasi ini memastikan setiap URL dievaluasi secara sistematis, memberikan data yang lebih bersih untuk analisis lebih lanjut.

## I. Kolom NoOfObfuscatedChar

```
def fill_no_of_obfuscated_characters(data, url_col='URL',
obf_char_col='NoOfObfuscatedChar'):
    def count_obfuscated_characters(url):
        if pd.notnull(url):
            hex_count =
len(re.findall(r'%[0-9a-fA-F]{2}', url)) # Count %XX
hexadecimal patterns
            at_count = url.count('@') # Count @ symbol
            return hex_count + at_count # Total
obfuscated characters
        return np.nan

    data[obf_char_col] = data.apply(
        lambda row:
count_obfuscated_characters(row[url_col])
        if pd.isnull(row[obf_char_col]) else
row[obf_char_col],
        axis=1
    )
    return data
```

## Implementasi

```
X_train = fill_no_of_obfuscated_characters(X_train,
url_col='URL', obf_char_col='NoOfObfuscatedChar')
X_train[['URL', 'NoOfObfuscatedChar']]
```

```
X_val = fill_no_of_obfuscated_characters(X_val,
url_col='URL', obf_char_col='NoOfObfuscatedChar')
X_val[['URL', 'NoOfObfuscatedChar']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **NoOfObfuscatedChar** dalam *dataset* dengan menghitung jumlah karakter yang menunjukkan adanya *obfuscation* dalam URL yang terdapat pada kolom **URL**. Fungsi **fill\_no\_of\_obfuscated\_characters** menerima parameter **url\_col** sebagai kolom yang berisi data URL, dan **obf\_char\_col** sebagai kolom yang akan menyimpan jumlah karakter yang disamarkan. Di dalamnya, fungsi bantu **count\_obfuscated\_characters** memeriksa apakah URL tidak kosong; jika tidak, fungsi ini menghitung jumlah pola heksadesimal yang diawali dengan '%' (seperti '%20') dan simbol '@' dalam URL, yang sering digunakan dalam teknik *obfuscation*. Selanjutnya, metode **apply** digunakan untuk memproses setiap baris: jika kolom **NoOfObfuscatedChar** kosong, maka **count\_obfuscated\_characters** dipanggil untuk menghitung jumlah karakter yang disamarkan; jika tidak, nilai **NoOfObfuscatedChar** dipertahankan. Setelah dijalankan, kolom **NoOfObfuscatedChar** akan terisi dengan jumlah karakter yang disamarkan dalam setiap URL pada *dataset*.

### m. Kolom ObfuscationRatio

```
def fill_obfuscation_ratio(data,
obf_char_col='NoOfObfuscatedChar',
url_length_col='URLLength',
obf_ratio_col='ObfuscationRatio'):
    def calculate_obfuscation_ratio(no_of_obfchar,
url_length):
        if pd.notnull(no_of_obfchar) and
pd.notnull(url_length) and url_length > 0:
```



```

        return no_of_obfchar / url_length
    return np.nan

    data[obf_ratio_col] = data.apply(
        lambda row:
        calculate_obfuscation_ratio(row[obf_char_col],
        row[url_length_col])
        if pd.isnull(row[obf_ratio_col]) else
        row[obf_ratio_col],
        axis=1
    )
    return data

```

## Implementasi

```

X_train = fill_obfuscation_ratio(X_train,
obf_char_col='NoOfObfuscatedChar',
url_length_col='URLLength',
obf_ratio_col='ObfuscationRatio')
X_train[['NoOfObfuscatedChar', 'URLLength',
'ObfuscationRatio']]

```

```

X_val = fill_obfuscation_ratio(X_val,
obf_char_col='NoOfObfuscatedChar',
url_length_col='URLLength',
obf_ratio_col='ObfuscationRatio')
X_val[['NoOfObfuscatedChar', 'URLLength',
'ObfuscationRatio']]

```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **ObfuscationRatio** dalam *dataset* dengan menghitung rasio karakter yang disamarkan terhadap panjang URL. Fungsi **fill\_obfuscation\_ratio** menerima parameter **obf\_char\_col** sebagai kolom yang berisi jumlah karakter yang disamarkan, **url\_length\_col** sebagai kolom panjang URL, dan **obf\_ratio\_col** sebagai kolom yang akan menyimpan rasio yang dihitung. Di dalamnya, fungsi bantu **calculate\_obfuscation\_ratio** memeriksa apakah jumlah karakter yang disamarkan dan panjang URL tidak kosong serta panjang URL lebih besar dari nol; jika ya, fungsi mengembalikan hasil pembagian antara jumlah karakter yang disamarkan dengan panjang URL. Selanjutnya, metode **apply** digunakan untuk memproses setiap baris: jika kolom **ObfuscationRatio** kosong, maka

**calculate\_obfuscation\_ratio** dipanggil untuk menghitung rasio; jika tidak, nilai **ObfuscationRatio** dipertahankan. Setelah dijalankan, kolom **ObfuscationRatio** akan terisi dengan rasio karakter yang disamakan terhadap panjang URL untuk setiap entri dalam *dataset*.

#### n. Kolom NoOfLettersInURL

```
def fill_no_of_letters_in_url(data, url_col='URL',
                              letters_col='NoOfLettersInURL'):
    def calculate_no_of_letters(url):
        if pd.notnull(url):
            return sum(c.isalpha() for c in url)
        return np.nan

    data[letters_col] = data.apply(
        lambda row: calculate_no_of_letters(row[url_col])
        if pd.isnull(row[letters_col]) else
        row[letters_col],
        axis=1
    )
    return data
```

#### Implementasi

```
X_train = fill_no_of_letters_in_url(X_train,
                                     url_col='URL', letters_col='NoOfLettersInURL')
X_train[['URL', 'NoOfLettersInURL']]
```

```
X_val = fill_no_of_letters_in_url(X_val, url_col='URL',
                                   letters_col='NoOfLettersInURL')
X_val[['URL', 'NoOfLettersInURL']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **NoOfLettersInURL** dalam *dataset* dengan menghitung jumlah huruf dalam setiap URL yang terdapat pada kolom **URL**. Fungsi **fill\_no\_of\_letters\_in\_url** menerima parameter **url\_col** sebagai kolom yang berisi data URL, dan **letters\_col** sebagai kolom yang akan menyimpan jumlah huruf yang dihitung. Di dalamnya, fungsi bantu **calculate\_no\_of\_letters** memeriksa apakah URL tidak kosong; jika tidak, fungsi ini menghitung jumlah karakter

dalam URL yang merupakan huruf dengan menggunakan generator ekspresi yang memeriksa setiap karakter menggunakan metode **isalpha()**. Selanjutnya, metode **apply** digunakan untuk memproses setiap baris: jika kolom **NoOfLettersInURL** kosong, maka **calculate\_no\_of\_letters** dipanggil untuk menghitung jumlah huruf; jika tidak, nilai **NoOfLettersInURL** dipertahankan. Setelah dijalankan, kolom **NoOfLettersInURL** akan terisi dengan jumlah huruf dalam setiap URL pada *dataset*.

#### o. Kolom LetterRatioInURL

```
def fill_letter_ratio_in_url(data,
    letters_col='NoOfLettersInURL',
    url_length_col='URLLength',
    ratio_col='LetterRatioInURL'):
    def calculate_letter_ratio(no_of_letters,
        url_length):
        if pd.notnull(no_of_letters) and
pd.notnull(url_length) and url_length > 0:
            return no_of_letters / url_length
        return np.nan

    data[ratio_col] = data.apply(
        lambda row:
calculate_letter_ratio(row[letters_col],
row[url_length_col])
        if pd.isnull(row[ratio_col]) else row[ratio_col],
        axis=1
    )
    return data
```

#### Implementasi

```
X_train = fill_letter_ratio_in_url(X_train,
    letters_col='NoOfLettersInURL',
    url_length_col='URLLength', ratio_col='LetterRatioInURL')
X_train[['NoOfLettersInURL', 'URLLength',
'LetterRatioInURL']]
```

```
X_val = fill_letter_ratio_in_url(X_val,
    letters_col='NoOfLettersInURL',
    url_length_col='URLLength', ratio_col='LetterRatioInURL')
X_val[['NoOfLettersInURL', 'URLLength',
'LetterRatioInURL']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **LetterRatioInURL** dalam *dataset* dengan menghitung rasio jumlah huruf terhadap panjang URL. Fungsi **fill\_letter\_ratio\_in\_url** menerima parameter **letters\_col** sebagai kolom yang berisi jumlah huruf dalam URL, **url\_length\_col** sebagai kolom panjang URL, dan **ratio\_col** sebagai kolom yang akan menyimpan rasio yang dihitung. Di dalamnya, fungsi bantu **calculate\_letter\_ratio** memeriksa apakah jumlah huruf dan panjang URL tidak kosong serta panjang URL lebih besar dari nol; jika ya, fungsi mengembalikan hasil pembagian antara jumlah huruf dengan panjang URL. Selanjutnya, metode **apply** digunakan untuk memproses setiap baris: jika kolom **LetterRatioInURL** kosong, maka **calculate\_letter\_ratio** dipanggil untuk menghitung rasio; jika tidak, nilai **LetterRatioInURL** dipertahankan. Setelah dijalankan, kolom **LetterRatioInURL** akan terisi dengan rasio jumlah huruf terhadap panjang URL untuk setiap entri dalam *dataset*.

**p. Kolom NoOfDegitsInURL**

```
def fill_no_of_digits_in_url(data, url_col='URL',
                             digits_col='NoOfDegitsInURL'):
    def calculate_no_of_digits(url):
        if pd.notnull(url):
            return sum(c.isdigit() for c in url)
        return np.nan

    data[digits_col] = data.apply(
        lambda row: calculate_no_of_digits(row[url_col])
        if pd.isnull(row[digits_col]) else
        row[digits_col],
        axis=1
    )
    return data
```

**Implementasi**

```
X_train = fill_no_of_digits_in_url(X_train,
                                    url_col='URL', digits_col='NoOfDegitsInURL')
X_train[['URL', 'NoOfDegitsInURL']]
```

```
X_val = fill_no_of_digits_in_url(X_val, url_col='URL',
                                digits_col='NoOfDegitsInURL')
X_val[['URL', 'NoOfDegitsInURL']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **NoOfDigitsInURL** dalam *dataset* dengan menghitung jumlah digit numerik dalam setiap URL yang terdapat pada kolom **URL**. Fungsi **fill\_no\_of\_digits\_in\_url** menerima parameter **url\_col** sebagai nama kolom yang berisi data URL, dan **digits\_col** sebagai nama kolom yang akan menyimpan jumlah digit yang dihitung. Di dalamnya, fungsi bantu **calculate\_no\_of\_digits** memeriksa apakah URL tidak kosong; jika tidak, fungsi ini menghitung jumlah karakter dalam URL yang merupakan digit numerik dengan menggunakan generator ekspresi yang memeriksa setiap karakter menggunakan metode *is\_digit()*. Selanjutnya, metode *apply()* digunakan untuk memproses setiap baris: jika kolom **NoOfDigitsInURL** kosong, maka **calculate\_no\_of\_digits** dipanggil untuk menghitung jumlah digit; jika tidak, nilai **NoOfDigitsInURL** dipertahankan. Setelah dijalankan, kolom **NoOfDigitsInURL** akan terisi dengan jumlah digit numerik dalam setiap URL pada *dataset*.

#### q. Kolom DegitRatioInURL

```
def fill_digits_ratio_in_url(data, url_col='URL',
                             ratio_col='DegitRatioInURL'):
    def calculate_digits_ratio(url):
        if pd.notnull(url) and len(url) > 0:
            return sum(c.isdigit() for c in url) /
len(url)
        return np.nan

    data[ratio_col] = data.apply(
        lambda row: calculate_digits_ratio(row[url_col])
        if pd.isnull(row[ratio_col]) else row[ratio_col],
        axis=1
    )
    return data
```

## Implementasi

```
X_train = fill_digits_ratio_in_url(X_train,
url_col='URL', ratio_col='DegitRatioInURL')
X_train[['URL', 'DegitRatioInURL']]
```

```
X_val= fill_digits_ratio_in_url(X_val, url_col='URL',
ratio_col='DegitRatioInURL')
X_val[['URL', 'DegitRatioInURL']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **DegitRatioInURL** dalam *dataset* dengan menghitung rasio jumlah digit terhadap panjang URL. Fungsi **fill\_digits\_ratio\_in\_url** menerima parameter **url\_col** sebagai nama kolom yang berisi data URL, dan **ratio\_col** sebagai nama kolom yang akan menyimpan rasio digit yang dihitung. Di dalamnya, fungsi bantu **calculate\_digits\_ratio** memeriksa apakah URL tidak kosong dan panjangnya lebih dari nol; jika ya, fungsi ini menghitung rasio dengan membagi jumlah karakter yang merupakan digit numerik dengan panjang total URL. Selanjutnya, metode *apply()* digunakan untuk memproses setiap baris: jika kolom **DegitRatioInURL** kosong, maka **calculate\_digits\_ratio** dipanggil untuk menghitung rasio digit; jika tidak, nilai **DegitRatioInURL** dipertahankan. Setelah dijalankan, kolom **DegitRatioInURL** akan terisi dengan rasio jumlah digit terhadap panjang URL untuk setiap entri dalam *dataset*.

### r. Kolom NoOfEqualsInURL

```
def fill_no_of_equals_in_url(data, url_col='URL',
equals_col='NoOfEqualsInURL'):
    def calculate_no_of_equals(url):
        if pd.notnull(url):
            return sum(c == "=" for c in url)
        return np.nan

    data[equals_col] = data.apply(
        lambda row: calculate_no_of_equals(row[url_col])
        if pd.isnull(row[equals_col]) else
        row[equals_col],
        axis=1
    )
```

```
return data
```

### Implementasi

```
X_train = fill_no_of_equals_in_url(X_train,  
url_col='URL', equals_col='NoOfEqualsInURL')  
X_train[['URL', 'NoOfEqualsInURL']]
```

```
X_val = fill_no_of_equals_in_url(X_val, url_col='URL',  
equals_col='NoOfEqualsInURL')  
X_val[['URL', 'NoOfEqualsInURL']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **NoOfEqualsInURL** dalam *dataset* dengan menghitung jumlah tanda sama dengan ("=") dalam setiap URL yang terdapat pada kolom **URL**. Fungsi **fill\_no\_of\_equals\_in\_url** menerima parameter **url\_col** sebagai nama kolom yang berisi data URL, dan **equals\_col** sebagai nama kolom yang akan menyimpan jumlah tanda sama dengan yang dihitung. Di dalamnya, fungsi bantu **calculate\_no\_of\_equals** memeriksa apakah URL tidak kosong; jika tidak, fungsi ini menghitung jumlah karakter "=" dalam URL dengan menggunakan ekspresi generator yang memeriksa setiap karakter. Selanjutnya, metode *apply()* digunakan untuk memproses setiap baris: jika kolom **NoOfEqualsInURL** kosong, maka **calculate\_no\_of\_equals** dipanggil untuk menghitung jumlah tanda sama dengan; jika tidak, nilai **NoOfEqualsInURL** dipertahankan. Setelah dijalankan, kolom **NoOfEqualsInURL** akan terisi dengan jumlah tanda sama dengan dalam setiap URL pada *dataset*.

#### s. Kolom NoOfQMarkInURL

```
def fill_no_of_qmark_in_url(data, url_col='URL',  
qmark_col='NoOfQMarkInURL'):  
    def calculate_no_of_qmark(url):  
        if pd.notnull(url):  
            return sum(c == "?" for c in url)  
        return np.nan  
  
    data[qmark_col] = data.apply(  

```

```

        lambda row: calculate_no_of_qmark(row[url_col])
        if pd.isnull(row[qmark_col]) else row[qmark_col],
        axis=1
    )
    return data

```

## Implementasi

```

X_train = fill_no_of_qmark_in_url(X_train, url_col='URL',
qmark_col='NoOfQMarkInURL')
X_train[['URL', 'NoOfQMarkInURL']]

```

```

X_val= fill_no_of_qmark_in_url(X_val, url_col='URL',
qmark_col='NoOfQMarkInURL')
X_val[['URL', 'NoOfQMarkInURL']]

```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **NoOfEqualsInURL** dalam *dataset* dengan menghitung jumlah tanda sama dengan ("=") dalam setiap URL yang terdapat pada kolom **URL**. Fungsi **fill\_no\_of\_equals\_in\_url** menerima parameter **url\_col** sebagai nama kolom yang berisi data URL, dan **equals\_col** sebagai nama kolom yang akan menyimpan jumlah tanda sama dengan yang dihitung. Di dalamnya, fungsi bantu **calculate\_no\_of\_equals** memeriksa apakah URL tidak kosong; jika tidak, fungsi ini menghitung jumlah karakter "=" dalam URL dengan menggunakan ekspresi generator yang memeriksa setiap karakter. Selanjutnya, metode *apply()* digunakan untuk memproses setiap baris: jika kolom **NoOfEqualsInURL** kosong, maka **calculate\_no\_of\_equals** dipanggil untuk menghitung jumlah tanda sama dengan; jika tidak, nilai **NoOfEqualsInURL** dipertahankan. Setelah dijalankan, kolom **NoOfEqualsInURL** akan terisi dengan jumlah tanda sama dengan dalam setiap URL pada *dataset*.

### t. Kolom NoOfAmpersandInURL

```

def fill_no_of_ampersand_in_url(data, url_col='URL',
ampersand_col='NoOfAmpersandInURL'):
    def calculate_no_of_ampersand(url):
        if pd.notnull(url):

```



```

        return sum(c == "&" for c in url)
    return np.nan

    data[ampersand_col] = data.apply(
        lambda row:
        calculate_no_of_ampersand(row[url_col])
        if pd.isnull(row[ampersand_col]) else
        row[ampersand_col],
        axis=1
    )
    return data

```

## Implementasi

```

X_train = fill_no_of_ampersand_in_url(X_train,
url_col='URL', ampersand_col='NoOfAmpersandInURL')
X_train[['URL', 'NoOfAmpersandInURL']]

```

```

X_val = fill_no_of_ampersand_in_url(X_val, url_col='URL',
ampersand_col='NoOfAmpersandInURL')
X_val[['URL', 'NoOfAmpersandInURL']]

```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **NoOfAmpersandInURL** dalam *dataset* dengan menghitung jumlah karakter ampersand ("&") dalam setiap URL yang terdapat pada kolom **URL**. Fungsi **fill\_no\_of\_ampersand\_in\_url** menerima parameter **url\_col** sebagai nama kolom yang berisi data URL, dan **ampersand\_col** sebagai nama kolom yang akan menyimpan jumlah ampersand yang dihitung. Di dalamnya, fungsi bantu **calculate\_no\_of\_ampersand** memeriksa apakah URL tidak kosong; jika tidak, fungsi ini menghitung jumlah karakter "&" dalam URL dengan menggunakan ekspresi generator yang memeriksa setiap karakter. Selanjutnya, metode *apply()* digunakan untuk memproses setiap baris: jika kolom **NoOfAmpersandInURL** kosong, maka **calculate\_no\_of\_ampersand** dipanggil untuk menghitung jumlah ampersand; jika tidak, nilai **NoOfAmpersandInURL** dipertahankan. Setelah dijalankan, kolom **NoOfAmpersandInURL** akan terisi dengan jumlah karakter "&" dalam setiap URL pada *dataset*.

#### u. Kolom NoOfOtherSpecialCharsInURL

```
def fill_no_of_special_chars_in_url(data, url_col='URL',
special_chars_col='NoOfOtherSpecialCharsInURL'):
    def calculate_no_of_specials(url):
        if pd.notnull(url):
            return sum(c.lower() not in
"0123456789abcdefghijklmnopqrstuvwxyz" for c in url)
            return np.nan

        data[special_chars_col] = data.apply(
            lambda row:
calculate_no_of_specials(row[url_col])
            if pd.isnull(row[special_chars_col]) else
row[special_chars_col],
            axis=1
        )
    return data
```

#### Implementasi

```
X_train = fill_no_of_special_chars_in_url(X_train,
url_col='URL',
special_chars_col='NoOfOtherSpecialCharsInURL')
X_train[['URL', 'NoOfOtherSpecialCharsInURL']]
```

```
X_val = fill_no_of_special_chars_in_url(X_val,
url_col='URL',
special_chars_col='NoOfOtherSpecialCharsInURL')
X_val[['URL', 'NoOfOtherSpecialCharsInURL']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **NoOfOtherSpecialCharsInURL** dalam *dataset* dengan menghitung jumlah karakter khusus (*special characters*) dalam setiap URL yang terdapat pada kolom **URL**. Fungsi **fill\_no\_of\_special\_chars\_in\_url** menerima parameter **url\_col** sebagai nama kolom yang berisi data URL, dan **special\_chars\_col** sebagai nama kolom yang akan menyimpan jumlah karakter khusus yang dihitung. Di dalamnya, fungsi bantu **calculate\_no\_of\_specials** memeriksa apakah URL tidak kosong; jika tidak, fungsi ini menghitung jumlah karakter yang bukan huruf atau angka dengan menggunakan ekspresi generator yang memeriksa setiap karakter dalam URL. Selanjutnya, metode *apply()* digunakan untuk memproses

setiap baris: jika kolom **NoOfOtherSpecialCharsInURL** kosong, maka **calculate\_no\_of\_specials** dipanggil untuk menghitung jumlah karakter khusus; jika tidak, nilai **NoOfOtherSpecialCharsInURL** dipertahankan. Setelah dijalankan, kolom **NoOfOtherSpecialCharsInURL** akan terisi dengan jumlah karakter khusus dalam setiap URL pada *dataset*.

#### v. Kolom SpacialCharRatioInURL

```
def fill_specials_ratio_in_url(data, url_col='URL',
ratio_col='SpacialCharRatioInURL'):
    def calculate_specials_ratio(url):
        if pd.notnull(url) and len(url) > 0:
            return sum(c.lower() not in
"0123456789abcdefghijklmnopqrstuvwxyz" for c in url) /
len(url)
        return np.nan

    data[ratio_col] = data.apply(
        lambda row:
calculate_specials_ratio(row[url_col])
        if pd.isnull(row[ratio_col]) else row[ratio_col],
        axis=1
    )
    return data
```

#### Implementasi

```
X_train = fill_specials_ratio_in_url(X_train,
url_col='URL', ratio_col='SpacialCharRatioInURL')
X_train[['URL', 'SpacialCharRatioInURL']]
```

```
X_val= fill_specials_ratio_in_url(X_val, url_col='URL',
ratio_col='SpacialCharRatioInURL')
X_val[['URL', 'SpacialCharRatioInURL']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **SpacialCharRatioInURL** dalam *dataset* dengan menghitung rasio karakter khusus (*special characters*) terhadap panjang total setiap URL yang terdapat pada kolom **URL**. Fungsi **fill\_specials\_ratio\_in\_url** menerima parameter **url\_col** sebagai nama kolom yang berisi data URL, dan **ratio\_col**

sebagai nama kolom yang akan menyimpan rasio karakter khusus yang dihitung. Di dalamnya, fungsi bantu **calculate\_specials\_ratio** memeriksa apakah URL tidak kosong dan memiliki panjang lebih dari nol; jika demikian, fungsi ini menghitung jumlah karakter yang bukan huruf atau angka dengan menggunakan ekspresi generator yang memeriksa setiap karakter dalam URL, kemudian membagi jumlah tersebut dengan panjang total URL untuk mendapatkan rasio. Selanjutnya, metode *apply()* digunakan untuk memproses setiap baris: jika kolom **SpacialCharRatioInURL** kosong, maka **calculate\_specials\_ratio** dipanggil untuk menghitung rasio karakter khusus; jika tidak, nilai **SpacialCharRatioInURL** dipertahankan. Setelah dijalankan, kolom **SpacialCharRatioInURL** akan terisi dengan rasio karakter khusus terhadap panjang total dalam setiap URL pada *dataset*.

#### w. Kolom IsHTTPS

```
def fill_is_https(data, url_col='URL',
https_col='IsHTTPS'):
    def calculate_ishttps(url):
        if pd.notnull(url):
            return 1 if "https://" in url else 0
        return np.nan

    data[https_col] = data.apply(
        lambda row: calculate_ishttps(row[url_col])
        if pd.isnull(row[https_col]) else row[https_col],
        axis=1
    )
    return data
```

#### Implementasi

```
X_train = fill_is_https(X_train, url_col='URL',
https_col='IsHTTPS')
X_train[['URL', 'IsHTTPS']]
```

```
X_val = fill_is_https(X_val, url_col='URL',
https_col='IsHTTPS')
X_val[['URL', 'IsHTTPS']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **IsHTTPS** dalam *dataset* dengan menentukan apakah URL pada kolom **URL** menggunakan protokol *HTTPS*. Fungsi **fill\_is\_https** menerima parameter **url\_col** sebagai nama kolom yang berisi data URL, dan **https\_col** sebagai nama kolom yang akan menyimpan indikator penggunaan *HTTPS*. Di dalamnya, fungsi bantu **calculate\_ishttps** memeriksa apakah URL tidak kosong; jika tidak, fungsi ini mengembalikan nilai 1 jika URL dimulai dengan "https://", dan 0 jika tidak. Selanjutnya, metode *apply()* digunakan untuk memproses setiap baris: jika kolom **IsHTTPS** kosong, maka **calculate\_ishttps** dipanggil untuk menentukan apakah URL menggunakan *HTTPS*; jika tidak, nilai **IsHTTPS** dipertahankan. Setelah dijalankan, kolom **IsHTTPS** akan terisi dengan nilai 1 untuk URL yang menggunakan *HTTPS* dan 0 untuk yang tidak, membantu dalam analisis lebih lanjut terkait keamanan atau karakteristik URL dalam *dataset*.

#### x. Kolom HasTitle

```
def fill_has_title(data, title_col='Title',
has_title_col='HasTitle'):
    data[has_title_col] = data.apply(
        lambda row: row[has_title_col]
        if not pd.isnull(row[has_title_col])
        else (1 if not pd.isnull(row[title_col]) else
row[has_title_col]),
        axis=1
    )
    return data
```

#### Implementasi

```
X_train = fill_has_title(X_train, title_col='Title',
has_title_col='HasTitle')
X_train[['Title', 'HasTitle']]
```

```
X_val = fill_has_title(X_val, title_col='Title',
has_title_col='HasTitle')
X_val[['Title', 'HasTitle']]
```

Kode ini bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **HasTitle** dalam *dataset* dengan menentukan apakah kolom **Title** memiliki nilai yang tidak kosong. Fungsi **fill\_has\_title** menerima parameter **title\_col** sebagai nama kolom yang berisi judul, dan **has\_title\_col** sebagai nama kolom yang akan menyimpan indikator keberadaan judul. Metode *apply()* digunakan untuk memproses setiap baris: jika kolom **HasTitle** kosong, maka akan diisi dengan nilai 1 jika kolom **Title** tidak kosong, menandakan bahwa judul tersedia; jika tidak, nilai **HasTitle** dipertahankan. Setelah fungsi ini dijalankan, kolom **HasTitle** akan terisi dengan nilai 1 untuk baris yang memiliki judul dan nilai 0 untuk yang tidak, sehingga memudahkan analisis lebih lanjut terkait keberadaan judul dalam *dataset*.

#### y. Kolom DomainTitleMatchScore

```
def fill_domain_title_match_score(data,
title_col='Title', domain_col='Domain',
score_col='DomainTitleMatchScore'):
    def calculate_match_score(title, domain):
        tSet = title.split(" ") if pd.notnull(title) else
        []
        txtDomain = domain.split(".")[:-1] if
pd.notnull(domain) else []
        txtDomain = [i for i in txtDomain if i != "www"]
        txtDomain = ".".join(txtDomain)

        score = 0
        baseScore = 100 / len(txtDomain) if
len(txtDomain) > 0 else 0

        for element in tSet:
            if element in txtDomain:
                n = len(element)
                score += baseScore * n
                txtDomain = txtDomain.replace(element,
                "")

                if score > 99.9:
                    score = 100
        return score

    data[score_col] = data.apply(
        lambda row: calculate_match_score(row[title_col],
```

```

row[domain_col])
    if pd.isnull(row[score_col])
    and pd.notnull(row[title_col])
    and pd.notnull(row[domain_col])
    else row[score_col],
    axis=1
)
return data

```

## Implementasi

```

X_train = fill_domain_title_match_score(
    X_train,
    title_col='Title',
    domain_col='Domain',
    score_col='DomainTitleMatchScore'
)
X_train[['URL', 'Title', 'DomainTitleMatchScore']]

```

```

X_val= fill_domain_title_match_score(
    X_val,
    title_col='Title',
    domain_col='Domain',
    score_col='DomainTitleMatchScore'
)
X_val[['URL', 'Title', 'DomainTitleMatchScore']]

```

Fungsi **fill\_domain\_title\_match\_score** bertujuan untuk mengisi nilai kosong (*missing values*) pada kolom **DomainTitleMatchScore** dengan menghitung tingkat kesesuaian antara kolom **Title** dan **Domain** dalam *dataset*. Sub-fungsi **calculate\_match\_score** membagi **Title** menjadi kata-kata terpisah dan **Domain** menjadi segmen-segmen, mengabaikan "www". Skor dihitung berdasarkan panjang kata yang cocok antara **Title** dan **Domain**, dengan rentang nilai 0 hingga 100; skor lebih tinggi menunjukkan kesesuaian yang lebih kuat. Setelah fungsi ini dijalankan, kolom **DomainTitleMatchScore** akan terisi dengan nilai yang mencerminkan tingkat kesesuaian antara **Title** dan **Domain**, yang dapat digunakan untuk analisis lebih lanjut dalam konteks keamanan atau karakteristik URL dalam *dataset*.

## z. Kolom URLTitleMatchScore

```
def fill_url_title_match_score(data, title_col='Title',
url_col='URL', score_col='URLTitleMatchScore'):
    def calculate_url_title_match_score(title, url):
        if pd.notnull(title) and pd.notnull(url):
            tSet = title.split(" ")
            txtURL = (
                urlparse(url).netloc.split(".")[:-1] +
                [i for i in urlparse(url).path.split("/")
if i != ""]
            )
            txtURL = [i for i in txtURL if i != "www"]
            txtURL = ".".join(txtURL)

            score = 0
            baseScore = 100 / len(txtURL) if len(txtURL)
> 0 else 0

            for element in tSet:
                if element in txtURL:
                    n = len(element)
                    score += baseScore * n
                    txtURL = txtURL.replace(element, "")
                    if score > 99.9:
                        score = 100
            return score
        return np.nan

    data[score_col] = data.apply(
        lambda row:
calculate_url_title_match_score(row[title_col],
row[url_col])
        if pd.isnull(row[score_col]) else row[score_col],
        axis=1
    )
    return data
```

## Implementasi

```
X_train = fill_url_title_match_score(
    X_train,
    title_col='Title',
    url_col='URL',
    score_col='URLTitleMatchScore'
)
X_train[['URL', 'Title', 'URLTitleMatchScore']]
```



```
X_val= fill_url_title_match_score(
    X_val,
    title_col='Title',
    url_col='URL',
    score_col='URLTitleMatchScore'
)
X_val[['URL', 'Title', 'URLTitleMatchScore']]
```

Fungsi **fill\_url\_title\_match\_score** digunakan untuk mengisi nilai kosong (*missing values*) pada kolom **URLTitleMatchScore** dengan menghitung tingkat kesesuaian antara **Title** dan **URL**. Sub-fungsi **calculate\_url\_title\_match\_score** membagi **Title** menjadi kumpulan kata dan **URL** menjadi bagian-bagian terpisah, termasuk domain dan path. Kesesuaian dihitung berdasarkan panjang kata dalam **Title** yang cocok dengan bagian dalam **URL**, menghasilkan skor antara 0 hingga 100, di mana skor yang lebih tinggi menunjukkan kesesuaian yang lebih kuat. Setelah fungsi ini diterapkan, kolom **URLTitleMatchScore** akan memiliki nilai yang mencerminkan hubungan semantik antara **Title** dan **URL**.

## II. Implementasi *Data Imputation: Mean, Median, Mode*

### a. *Data Numerical*

```
def fill_numerical_columns(data, numerical_columns,
    skewness_threshold=3):
    for col in numerical_columns:
        if col in data.columns:
            skewness = data[col].skew()
            if skewness > skewness_threshold or skewness
            < -skewness_threshold:
                fill_value = data[col].median()
                print(f"Column {col}: Using median for
                imputation because distribution is highly skewed.")
            else:
                fill_value = data[col].mean()
                print(f"Column {col}: Using mean for
                imputation because distribution is approximately
                normal.")
            data[col] = data[col].fillna(fill_value)
    print("Missing values after imputation:")
    print(data[numerical_columns].isnull().sum())
    return data
```

Fungsi **fill\_numerical\_columns** dirancang untuk menangani *missing values* pada kolom numerik dalam sebuah *dataframe*. Untuk setiap kolom numerik yang ditentukan, fungsi ini pertama-tama menghitung nilai *skewness* (kemencengan) distribusi data. Jika nilai *skewness* melebihi ambang batas yang ditentukan (secara default 3) atau kurang dari negatif ambang batas tersebut, distribusi dianggap sangat miring, dan nilai median digunakan untuk imputasi karena lebih tahan terhadap pencilan. Sebaliknya, jika distribusi mendekati normal, nilai rata-rata digunakan untuk imputasi. Setelah menentukan nilai imputasi yang sesuai, fungsi ini menggantikan *missing values* dengan nilai tersebut. Setelah semua kolom numerik diproses, fungsi menampilkan jumlah *missing values* yang tersisa di setiap kolom numerik untuk memastikan imputasi telah berhasil. Pendekatan ini memastikan bahwa imputasi nilai hilang mempertimbangkan distribusi data, sehingga mengurangi potensi bias yang disebabkan oleh pencilan atau distribusi yang tidak normal.

#### Implementasi: Untuk X\_train

```
X_train = fill_numerical_columns(X_train,  
numerical_columns, skewness_threshold=3)
```

#### Implementasi: Untuk X\_val

```
X_val = fill_numerical_columns(X_train, numerical_columns,  
skewness_threshold=3)
```

#### b. Data Categorical

```
def fill_categorical_columns(data, categorical_columns,  
initial_group_by):  
    group_by_columns = initial_group_by.copy()  
  
    for col in categorical_columns:  
        mode_values = (  
            data.groupby(group_by_columns)[col]  
            .agg(lambda x: x.mode()[0] if not  
x.mode().empty else None)  
        )
```

```

        global_mode = data[col].mode()[0] if not
data[col].mode().empty else None

        def fill_value(row):
            if pd.isnull(row[col]):
                group_key = tuple(row[gb] for gb in
group_by_columns)
                return mode_values.get(group_key,
global_mode)
            return row[col]

        data[col] = data.apply(fill_value, axis=1)

        group_by_columns.append(col)

    for col in categorical_columns:
        if data[col].isnull().sum() > 0:
            global_mode = data[col].mode()[0] if not
data[col].mode().empty else None
            data[col] = data[col].fillna(global_mode)

    return data

```

### Implementasi: Untuk X\_train

```

categorical_columns_to_fill = [
    'IsDomainIP', 'IsHTTPS', 'HasTitle', 'IsResponsive',
'Pay', 'HasHiddenFields', 'Robots',
    'Crypto', 'HasDescription', 'Bank',
'HasExternalFormSubmit', 'HasFavicon',
    'HasSubmitButton', 'HasPasswordField',
'NoOfSelfRedirect', 'HasCopyrightInfo',
    'NoOfURLRedirect', 'HasSocialNet'
]

group_by_columns = ['HasObfuscation']
X_train = fill_categorical_columns(X_train,
categorical_columns_to_fill, group_by_columns)
print(f"Missing values after imputation:")
print(X_train.isnull().sum())

```

### Implementasi: Untuk X\_val

```

X_val = fill_categorical_columns(X_val,
categorical_columns_to_fill, group_by_columns)
print(f"Missing values after imputation:")
print(X_val.isnull().sum())

```

Fungsi **fill\_categorical\_columns** dirancang untuk menangani *missing values* pada kolom-kolom kategorikal dengan **pendekatan berbasis pengelompokan (*grouping*)**. Prosesnya dimulai dengan menentukan kolom-kolom kategorikal yang akan diisi dan kolom-kolom awal untuk pengelompokan. Untuk setiap kolom kategorikal, fungsi ini menghitung nilai modus (nilai yang paling sering muncul) berdasarkan kelompok yang ditentukan oleh kolom pengelompokan. Jika suatu entri memiliki nilai kosong, maka akan diisi dengan modus dari kelompoknya; jika tidak ada modus yang tersedia, akan digunakan modus global dari seluruh data. Setelah mengisi nilai kosong berdasarkan pengelompokan, fungsi ini menambahkan kolom yang baru diisi ke dalam daftar kolom pengelompokan dan melanjutkan ke kolom berikutnya. Terakhir, jika masih ada nilai kosong yang tersisa, fungsi ini mengisi mereka dengan modus global untuk memastikan tidak ada *missing values* yang tersisa dalam kolom-kolom kategorikal tersebut. Pendekatan ini memastikan bahwa imputasi nilai dilakukan dengan mempertimbangkan struktur data dan hubungan antar kolom, sehingga hasilnya lebih akurat dibandingkan dengan metode imputasi sederhana seperti pengisian dengan nilai tetap atau penghapusan data yang hilang.

Hasil akhir dari proses imputasi menunjukkan bahwa **seluruh *missing value* dalam *dataset* berhasil diatasi tanpa perlu melakukan penghapusan baris atau kolom**. Setiap nilai yang hilang diisi berdasarkan strategi yang telah dirancang dengan mempertimbangkan karakteristik distribusi data, baik numerik maupun kategorikal. Dengan demikian, *dataset* tetap utuh, sehingga memaksimalkan informasi yang tersedia untuk proses analisis dan model prediktif.

### **3.2 Dealing with Outliers**

*Dealing with outliers* merupakan langkah penting untuk memastikan kualitas data numerik sebelum proses analisis dan model *building*. Dalam hal ini, proses dilakukan pada dua set data, yaitu *X\_train\_numerical\_all* dan *X\_val\_numerical\_all*, dengan pendekatan bertahap. Tahap pertama adalah mengevaluasi distribusi setiap fitur

berdasarkan *skewness* untuk menentukan metode deteksi *outliers* yang sesuai, fitur dengan distribusi normal dianalisis menggunakan *z-scores*, sedangkan fitur dengan distribusi tidak normal menggunakan metode IQR (*Interquartile Range*). Untuk menangani outliers, digunakan dua pendekatan utama, yaitu transformasi data (*log transform* dan *square root transform*) untuk meratakan distribusi, serta *clipping* untuk membatasi nilai *outliers* pada rentang tertentu. Pendekatan ini bertujuan untuk mengurangi pengaruh nilai ekstrim tanpa menghapus data secara langsung, sehingga memastikan data yang digunakan lebih bersih, stabil, dan siap mendukung analisis serta pengembangan model yang lebih akurat. Berikut adalah *function code* yang digunakan dalam mengerjakan *dealing with outliers*.

**a. Fungsi untuk Mendeteksi Outliers**

```
def detect_outliers(df, skewness_threshold=3,
z_score_threshold=3):
    results = []

    for col in df.columns:

        skewness = df[col].skew()

        if abs(skewness) <= skewness_threshold:
            z_scores = zscore(df[col].dropna())
            outliers = np.abs(z_scores) >
z_score_threshold
            method = "Z-Score"
        else:

            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            outliers = (df[col] < lower_bound) | (df[col]
> upper_bound)
            method = "IQR"

        outlier_count = outliers.sum()

        results.append({
            'Feature': col,
            'Outlier Count': outlier_count,
            'Method': method
        })
```

```
return pd.DataFrame(results)
```

**b. Handling Outliers with Log Transform**

```
def outlier_handlers_logform(X):
    X_numerical_filtered =
X_train[numerical_columns].loc[:,
(X_train[numerical_columns].max() <= 1000) &
(X_train[numerical_columns].min() >= 0)]
    return np.log1p(X_numerical_filtered)

X_train_log_transformed_filtered =
outlier_handlers_logform(X_train)
cols_per_row = 4
num_cols = X_train_log_transformed_filtered.shape[1]
num_rows = (num_cols + cols_per_row - 1) // cols_per_row

fig, axes = plt.subplots(nrows=num_rows,
ncols=cols_per_row, figsize=(16, 4 * num_rows))
axes = axes.flatten()

for i, column in
enumerate(X_train_log_transformed_filtered.columns):

X_train_log_transformed_filtered.boxplot(column=column,
ax=axes[i])
    axes[i].set_title(f'Log Transformed Box Plot for
{column}')
    axes[i].set_ylabel('Log Values')

for j in range(i + 1, len(axes)):
    axes[j].set_visible(False)

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

Fungsi *outlier\_handlers\_logform* memfilter kolom numerik dengan nilai dalam rentang 0 hingga 1000, kemudian menerapkan transformasi logaritmik menggunakan `np.log1p`, yang aman untuk data dengan nilai nol. Hasil transformasi disimpan di `X_train_log_transformed_filtered` dan divisualisasikan menggunakan box plot untuk setiap kolom, yang menunjukkan distribusi data setelah transformasi. Proses ini memastikan *outliers* diminimalkan dan distribusi data menjadi lebih normal, mempersiapkannya untuk analisis atau model lebih

lanjut.

### c. *Handling Outliers with SQRT*

```
def outlier_handlers_sqrt(X):
    X_small_range = X_train[numerical_columns].loc[:,
(X_train[numerical_columns].max() <= 15) &
(X_train[numerical_columns].min() >= 0)]
    return np.sqrt(X_small_range)

X_train_sqrt_transformed = outlier_handlers_sqrt(X_train)

cols_per_row = 4
num_cols = X_train_sqrt_transformed.shape[1]
num_rows = (num_cols + cols_per_row - 1) // cols_per_row

fig, axes = plt.subplots(nrows=num_rows,
ncols=cols_per_row, figsize=(16, 4 * num_rows))
axes = axes.flatten()

for i, column in
enumerate(X_train_sqrt_transformed.columns):
    X_train_sqrt_transformed.boxplot(column=column,
ax=axes[i])
    axes[i].set_title(f'Sqrt Transformed Box Plot for
{column}')
    axes[i].set_ylabel('Sqrt Values')

for j in range(i + 1, len(axes)):
    axes[j].set_visible(False)

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

Fungsi *outlier\_handlers\_sqrt* memfilter kolom numerik dengan rentang nilai 0 hingga 15, lalu menerapkan *np.sqrt* pada data. Hasil transformasi divisualisasikan dengan box plot untuk memeriksa apakah *outliers* berhasil diminimalkan dan distribusi data menjadi lebih stabil.

### d. *Handling Outliers with Clipping*

```
def outlier_handlers_clip(df, lower_percentile=0.01,
upper_percentile=0.99):
    clipped_df = df.copy()
```

```

        for col in clipped_df[numerical_columns].columns:
            lower_bound =
np.percentile(clipped_df[col].dropna(), lower_percentile
* 100)
            upper_bound =
np.percentile(clipped_df[col].dropna(), upper_percentile
* 100)
            clipped_df[col] = np.clip(clipped_df[col],
lower_bound, upper_bound)

        return clipped_df

```

Fungsi ini menangani *outliers* dengan membatasi nilai kolom numerik berdasarkan persentil. Pertama, fungsi membuat salinan data untuk diproses. Kemudian, untuk setiap kolom numerik, batas bawah dan atas dihitung menggunakan `np.percentile` sesuai nilai persentil yang ditentukan (default 1% dan 99%). Nilai pada kolom tersebut disesuaikan menggunakan `np.clip`, sehingga nilai di luar batas bawah atau atas akan diatur ke batas tersebut. Terakhir, data yang telah di-*clipping* dikembalikan sebagai hasil.

### 3.3 Data Validation

Tahap **Data Validation** merupakan langkah penting dalam proses pengolahan data untuk memastikan integritas dan konsistensi *dataset*. Pada tahap ini, dilakukan pemeriksaan terhadap tipe data, kesesuaian label pada variabel kategorikal, serta rentang nilai pada variabel numerik agar sesuai dengan ekspektasi. Proses validasi ini membantu mencegah kesalahan saat analisis data atau implementasi model, karena data yang tidak sesuai dapat menyebabkan kegagalan atau hasil yang tidak akurat dalam prediksi.

```

# Data Types Validation

# FINAL:
# Change all 'object' features into 'string'
X_train = X_train.apply(lambda col: col.astype('string') if
col.dtypes == 'object' else col)
X_val = X_val.apply(lambda col: col.astype('string') if
col.dtypes == 'object' else col)

# Change all categorical 'float64' into 'int'

```



```

columns_to_validate = categorical_columns_filtered +
discrete_columns
X_train[columns_to_validate] =
X_train[columns_to_validate].apply(lambda col:
col.astype('int'))
X_val[columns_to_validate] =
X_val[columns_to_validate].apply(lambda col:
col.astype('int'))

# Binary categorical features validation (must contains 1 or 0)
classes_X_train_dict_validate = {col: X_train[col].unique()
for col in categorical_columns_filtered}
classes_X_val_dict_validate = {col: X_val[col].unique() for
col in categorical_columns_filtered}

```

Kode di atas melakukan validasi tipe data dan konsistensi nilai pada *dataset* **X\_train** dan **X\_val**. Langkah pertama mengubah semua kolom bertipe **object** menjadi tipe **string**, karena tipe **object** bersifat umum dan dapat menimbulkan inkonsistensi saat pengolahan data. Selanjutnya, kolom kategorikal bertipe **float64** diubah menjadi **int** untuk mencerminkan sifatnya sebagai nilai diskrit. Kode kemudian memeriksa apakah semua fitur kategorikal biner hanya memiliki nilai *0* atau *1*, dengan membuat kamus unik untuk setiap kolom kategorikal dalam *dataset* **X\_train** dan **X\_val**. Validasi ini memastikan bahwa *dataset* memiliki tipe data dan nilai yang konsisten sebelum melanjutkan ke tahap berikutnya.

### 3.4 Removing Duplicates

Proses penghapusan duplikat pada *dataset* dilakukan menggunakan fungsi `drop_duplicates()` secara langsung pada `DataFrame` **X\_train**. Langkah ini bertujuan untuk memastikan bahwa setiap baris data bersifat unik dengan menghapus baris-baris yang memiliki nilai identik di seluruh kolom. Setelah proses penghapusan, jumlah baris dan kolom diverifikasi menggunakan atribut `shape` untuk memastikan data yang tersisa sesuai dengan harapan. Selain itu, data **X\_train** setelah penghapusan duplikat ditampilkan dalam bentuk tabel dengan styling untuk mempermudah pengecekan dan interpretasi. Proses ini penting untuk menjaga kualitas data dan mencegah redudansi yang dapat mempengaruhi analisis atau performa model secara negatif. Semua perubahan langsung diterapkan pada `DataFrame` **X\_train**, sehingga data yang digunakan sudah bersih dari duplikat.

```
num_duplicates = X_train.duplicated().sum()
print(f"Number of duplicated columns: {num_duplicates}")

X_train = X_train.drop_duplicates()
```

### 3.5 Feature Engineering

Pada tahap *feature engineering*, dilakukan dua proses utama, yaitu *feature selection* dan *new features*. Tujuan dari langkah-langkah ini adalah untuk meningkatkan kualitas data yang akan digunakan dalam model *machine learning* dengan memilih fitur yang paling relevan serta menciptakan fitur baru yang lebih informatif. Berikut adalah implementasi *feature selection* dan *new features*.

#### a. Feature Selection

```
def feature_selection(X, y):
    # 1. Feature Selection

    # Chi2 Selector and Anova Selector
    chi2_selector = SelectKBest(score_func=chi2, k=20)
    anova_selector = SelectKBest(score_func=f_classif, k=15)

    # a) Fit kedua selector dengan data & label
    chi2_selector.fit(X[categorical_columns_filtered], y)
    anova_selector.fit(X[numerical_columns], y)

    # b) Ambil nama kolom terpilih
    selected_columns_chi2 =
X[categorical_columns_filtered].columns[chi2_selector.get_support()]
    selected_columns_anova =
X[numerical_columns].columns[anova_selector.get_support()]

    # c) Gabungkan kolom terpilih
    selected_cols_union =
selected_columns_chi2.union(selected_columns_anova)
    X_selected = X[selected_cols_union]

    print(f"Selected Features: {X_selected.columns.tolist()}")

    return X_selected

X_train_feature_selection = X_train.copy()
y_train_feature_selection = y_train.copy()

X_val_feature_selection = X_val.copy()
y_val_feature_selection = y_val.copy()
```

```

feature_selection(X_train_feature_selection,
y_train_feature_selection)
feature_selection(X_val_feature_selection,
y_val_feature_selection)

```

Pada tahap *features selection*, digunakan metode Chi-Square (Chi2) dan ANOVA F-Test untuk memilih fitur-fitur yang paling relevan terhadap target. Chi2 digunakan untuk fitur kategorikal, di mana 20 fitur terbaik dipilih berdasarkan relevansinya, sementara ANOVA F-Test diterapkan pada fitur numerikal untuk memilih 15 fitur terbaik. Setelah fitur-fitur terpilih dari kedua metode digabungkan, dataset diperbarui dengan hanya menyertakan fitur yang paling informatif. Langkah ini bertujuan untuk menyederhanakan model, mengurangi *noise*, dan meningkatkan kemampuan model dalam memahami pola data yang signifikan.

## b. New Features

```

def new_features(X):
    # a) NoOfPops -> Sum(NoOfPopup, NoOfiFrame)
    X['NoOfPops'] = X['NoOfPopup'] + X['NoOfiFrame']

    # b) NoOfHref -> Sum(NoOfSelfRef, NoOfEmptyRef,
    NoOfExternalRef)
    X['NoOfHref'] = X['NoOfSelfRef'] + X['NoOfEmptyRef'] +
    X['NoOfExternalRef']

    # c) HasSolidInfo -> HasCopyrightInfo || HasSocialNet
    X['HasSolidInfo'] = ((X['HasCopyrightInfo'] == 1) |
    (X['HasSocialNet'] == 1)).astype(int)

```

Selain *feature selection*, dibuat fitur-fitur baru untuk menambah informasi yang dapat membantu model dalam mengenali pola-pola yang tidak tertangkap oleh fitur asli. Contohnya, fitur NoOfPops dibuat dengan menjumlahkan NoOfPops dan NoOfiFrame untuk merepresentasikan total elemen *pop-up*, sedangkan fitur NoOfHref mencerminkan total *hyperlink* dari berbagai kategori dengan menjumlahkan NoOfSelfRef, NoOfEmptyRef, dan NoOfExternalRef. Fitur HasSolidInfo dibuat menggunakan logika yang mengevaluasi keberadaan informasi hak cipta (HasCopyrightInfo) atau jejaring

sosial (HasSocialNet). Dengan fitur-fitur baru ini, dataset menjadi lebih kaya akan informasi yang relevan, sehingga meningkatkan kemampuan model untuk membuat prediksi yang lebih akurat.

## ***PREPROCESSING DATA***

### ***3.6 Feature Scaling***

*Feature scaling* merupakan langkah penting dalam *preprocessing data* agar semua fitur memiliki skala yang seimbang. Hal ini dilakukan supaya model *machine learning* bisa memproses data dengan lebih baik, tanpa terpengaruh oleh perbedaan skala antar fitur. Dalam tugas besar ini, kami mencoba tiga jenis metode *feature scaling*, yaitu *min-max scaling*, *robust scaling*, dan *standardization*. Ketiga metode tersebut kami uji untuk melihat mana yang paling sesuai dengan karakteristik data kami, seperti distribusi nilai, keberadaan *outlier*, dan kebutuhan model yang digunakan. Setelah melakukan analisis, kami memutuskan untuk menggunakan *standardization* sebagai metode *scaling* utama. *Standardization* atau *z-score* dipilih karena dapat menangani data dengan variabilitas tinggi atau keberadaan *outlier*. Metode ini mengubah nilai fitur agar memiliki rata-rata 0 dan standar deviasi 1, sehingga lebih cocok untuk model yang membutuhkan data terdistribusi mendekati normal atau setidaknya seimbang di sekitar nol. Sementara itu, meskipun *min-max scaling* terlihat lebih sederhana karena memetakan nilai langsung ke rentang [0,1], metode ini kurang efektif karena rentan terhadap *outlier*, yang bisa membuat data di tengah rentang menjadi terlalu terkompresi. Dengan pertimbangan tersebut, *standardization* kami pilih untuk memastikan hasil model yang optimal dan akurat. Berikut adalah fungsi *standardization* yang kami terapkan dalam tugas besar ini.

```
def apply_standardization(data):  
    standard_scaler = StandardScaler()  
    data[numerical_columns] =  
    standard_scaler.fit_transform(data[numerical_columns])  
    return data
```

Fungsi *apply\_standardization* melakukan proses *standardization* pada kolom numerik dalam dataset, menggunakan *StandardScaler* dari pustaka *scikit-learn*. Fungsi

ini menghitung rata-rata dan standar deviasi setiap kolom numerik, lalu mengubah nilai-nilainya agar memiliki rata-rata 0 dan standar deviasi 1. Proses ini membantu menyamakan skala data sehingga model machine learning dapat bekerja lebih efektif dan akurat.

### 3.7 Feature Encoding

*Feature encoding* tidak dilakukan pada proses ini karena *dataset* yang digunakan telah disiapkan dengan *encoding* yang sesuai. *Dataset* tersebut sudah memiliki format yang kompatibel dengan kebutuhan analisis atau model yang akan digunakan, sehingga tidak diperlukan langkah tambahan untuk mengubah atau menyesuaikan fitur-fitur dalam data. Hal ini memungkinkan efisiensi waktu dan meminimalkan risiko kesalahan akibat manipulasi data yang tidak diperlukan.

### 3.8 Handling Imbalanced Dataset

Ketidakseimbangan *dataset* menyebabkan model lebih cenderung memprediksi kelas mayoritas, sehingga mengabaikan kelas minoritas yang mungkin justru lebih penting. Oleh karena itu, diperlukan teknik khusus untuk menangani masalah ini, salah satunya adalah *oversampling*. *Oversampling* bertujuan untuk meningkatkan jumlah sampel pada kelas minoritas sehingga distribusi kelas menjadi lebih seimbang dan model dapat belajar secara adil dari semua kelas. Berikut adalah *code* yang kami implementasikan dalam melakukan *oversampling* menggunakan metode SMOTE (*Synthetic Minority Oversampling Technique*).

```
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced =
smote.fit_resample(X_train_numerical_all, y_train)

balanced_class_distribution =
pd.Series(y_train_balanced).value_counts()
```

Pertama, SMOTE diinisialisasi dengan `random_state=42` (*random state*) untuk memastikan hasil yang konsisten. Kemudian, fungsi `fit_resample` digunakan untuk membuat *dataset* baru yang seimbang, yaitu `X_train_balanced` (fitur) dan

`y_train_balanced` (label). Hasilnya, distribusi kelas pada data latih menjadi lebih merata, yang dihitung menggunakan `value_counts`. SMOTE akan menghasilkan data tambahan dengan membuat sampel baru berdasarkan interpolasi data minoritas, sehingga data lebih bervariasi tanpa hanya menduplikasi sampel asli.

## BAB IV

### PERBANDINGAN HASIL PREDIKSI

#### 4.1 Analisis KNN

Berdasarkan hasil pengujian implementasi KNN yang dibuat dari *scratch* dan *library scikit-learn*, didapatkan bahwa keduanya menghasilkan akurasi yang sangat baik dan mirip. Akurasi dari implementasi KNN yang dibuat dari *scratch* mencapai 98,07% dan akurasi dari implementasi KNN milik *library scikit-learn* mencapai 98,65% Hal ini membuktikan bahwa implementasi KNN yang dibuat dari *scratch* sudah berfungsi dengan sangat baik.

Pada kenyataan, untuk mendapatkan angka sebesar itu kami dapatkan dengan hanya menggunakan 10.000 data pertama dalam pembelajaran KNN. Waktu yang dibutuhkan untuk mengeksekusi implementasi KNN dari *scratch* memang cukup lebih lama, daripada milik *library scikit-learn*, yaitu sekitar 4 menit pada KNN dari *scratch* dan 5 milisekon pada KNN milik *library scikit-learn*. Hal ini terjadi karena *scikit-learn* menggunakan bahasa C yang jauh lebih baik ketimbang dengan *python*. Selain itu tentunya *library scikit-learn* dibuat oleh orang yang sudah sangat *high-end* di bidangnya.

Berikut adalah perbandingan kedua hasil.

```
===== KNN from Scratch =====
Akurasi KNN : 0.9806986930664863

Classification Report (KNN):
              precision    recall  f1-score   support

     0       0.98         0.76         0.86         2152
     1       0.98         1.00         0.99        25929

 accuracy          0.98          0.98          0.98        28081
  macro avg       0.98          0.88          0.92        28081
weighted avg       0.98          0.98          0.98        28081

Confusion Matrix (KNN):
[[ 1646   506]
 [   36 25893]]
```

```

===== KNN by Scikit-Learn =====
Akurasi KNN : 0.9864321071186923

Classification Report (KNN):
              precision    recall  f1-score   support

         0       0.96      0.86      0.91       2152
         1       0.99      1.00      0.99      25929

 accuracy      0.98
 macro avg     0.98
 weighted avg  0.99

Confusion Matrix (KNN):
[[ 1844   308]
 [    73 25856]]

```

## 4.2 Analisis Gaussian Naive Bayes

Berdasarkan hasil pengujian implementasi Gaussian Naive Bayes yang dibuat dari *scratch* dan *library scikit-learn*, didapatkan bahwa keduanya menghasilkan akurasi yang lalalala. Akurasi dari implementasi KNN yang dibuat dari *scratch* mencapai 98,6% dan akurasi dari implementasi KNN milik *library scikit-learn* mencapai 95,43% Hal ini membuktikan bahwa implementasi KNN yang dibuat dari *scratch* sudah berfungsi dengan sangat baik.

Pada kenyataan eksekusinya, sama dengan kasus KNN, memang waktu yang dibutuhkan untuk mengeksekusi implementasi Naive Bayes dari *scratch* lebih lama sedikit daripada milik *library scikit-learn* karena *scikit-learn* menggunakan bahasa C yang jauh lebih baik ketimbang dengan *python*. Selain itu tentunya *library scikit-learn* dibuat oleh orang yang sudah sangat *high-end* di bidangnya.



Berikut adalah perbandingan kedua hasil.

```
===== Gaussian Naive Bayes from Scratch =====
Akurasi NB : 0.9864321071186923

Classification Report (NB):
              precision    recall  f1-score   support

     0       0.96       0.86       0.91       2152
     1       0.99       1.00       0.99      25929

 accuracy       0.99       0.99       0.99      28081
  macro avg       0.98       0.93       0.95      28081
weighted avg       0.99       0.99       0.99      28081

Confusion Matrix (NB):
[[ 1844   308]
 [   73 25856]]
```

```
===== Gaussian Naive Bayes from Scikit-Learn =====
Akurasi : 0.9543107439193761

Classification Report:
              precision    recall  f1-score   support

     0       0.63       0.95       0.76       2152
     1       1.00       0.95       0.97      25929

 accuracy       0.95       0.95       0.95      28081
  macro avg       0.82       0.95       0.87      28081
weighted avg       0.97       0.95       0.96      28081

Confusion Matrix:
[[ 2049   103]
 [ 1180 24749]]
```

### 4.3 Kesimpulan

Berdasarkan waktu eksekusi, KNN membutuhkan waktu terlalu lama daripada Gaussian Naive Bayes. Hal ini terjadi karena cara kerja KNN yang didasari oleh *query* dari *dataset* yang ingin diprediksi, dimana setiap data akan di-*query* satu persatu. Sehingga penggunaan KNN pada *dataset* yang sangat besar seperti pada kasus tugas besar ini sangat tidak disarankan.

Sedangkan waktu eksekusi pada model Gaussian Naive Bayes lebih cepat karena hanya didasari oleh probabilitas yang sudah dievaluasi sebelumnya dan mencari pilihan dengan probabilitas terbesar walaupun angka probabilitasnya tidak terlalu beda. Tetapi, konsekuensi yang didapatkan adalah hasil akurasi tidak sebaik KNN.

#### 4.4 Saran

Untuk meningkatkan performa model, terutama dalam mengenali kelas minoritas, terdapat beberapa langkah yang dapat dilakukan. Pertama, pertimbangkan penggunaan algoritma alternatif seperti *Random Forest* atau *Gradient Boosting*. Kedua algoritma ini lebih efektif dalam menangani ketidakseimbangan data karena mampu menangkap pola yang lebih kompleks dibandingkan dengan KNN. Kedua, coba terapkan teknik *oversampling* tambahan untuk lebih meningkatkan distribusi data pada kelas minoritas dengan teknik selain SMOTE. Alternatif lain yang dapat dicoba adalah pengaturan ulang bobot pada algoritma KNN (*Weighted KNN*) agar tetangga yang lebih relevan mendapatkan pengaruh lebih besar. Ketiga, tambahkan fitur baru yang lebih relevan (*creating new features*) untuk membantu model memahami pola-pola yang lebih signifikan pada data minoritas, sehingga dapat meningkatkan performa model secara keseluruhan. Dengan langkah-langkah ini, diharapkan model dapat memberikan hasil yang lebih seimbang tanpa mengorbankan akurasi kelas mayoritas.

## BAB V KONTRIBUSI

Berikut adalah pembagian kerja pengerjaan Tugas Besar IF3070 Dasar Intelegensi Artifisial 2, yaitu:

NIM	Nama	Bagian Pengerjaan
18222023	Thalita Zahra Sutejo	<ol style="list-style-type: none"> <li>1. <i>Import Data, Feature Specification: Categorical and Numerical listing dan Split Training Set dan Validation Set</i></li> <li>2. Implementasi <i>Handling Missing Value</i> for X_train dan X_val: <i>Domain Specific Strategies</i> 16 column dan Pengerjaan Laporan Bagian <i>Handling Missing Value</i> (Bagian III.1 (I) - a sampai z)</li> <li>3. Implementasi <i>Handling Missing Value</i> for X_train dan X_val: <i>Data Imputation</i> for 51 column with <i>Mean, Median, or Mode Imputation (Numerical and Categorical)</i> dan Pengerjaan Laporan Bagian <i>Handling Missing Value</i> (Bagian III.1 (II) - a dan b)</li> <li>4. <i>Code Compile Preprocessing Pipeline dan Function Making: Data Cleaning</i></li> <li>5. Implementasi Algoritma <i>Naive-Bayes</i> dengan <i>Scikit Learn</i> serta Pengerjaan Laporan Bagian Implementasi Algoritma <i>Naive-Bayes</i></li> <li>6. Laporan bagian <i>Data Validation</i></li> <li>7. README Github</li> </ol>
18222056	Irfan Musthofa	<ol style="list-style-type: none"> <li>1. <i>Code Compile Preprocessing Pipeline dan Function Making: Preprocessing</i></li> <li>2. Implementasi <i>feature engineering</i></li> <li>3. Implementasi <i>data validation</i></li> <li>4. Melakukan analisis model</li> <li>5. Implementasi dan <i>debugging</i> bagian <i>Preprocessing</i></li> <li>6. Laporan Bagian Perbandingan Hasil Prediksi dan kesimpulan.</li> <li>7. Implementasi evaluasi model dan <i>debug</i> sickit learn dengan evaluasi.</li> </ol>

<b>18222059</b>	Eleanor Cordelia	<ol style="list-style-type: none"> <li>1. Implementasi <i>Dealing with Outliers</i> for X_train dan X_val + Laporan <i>Outliers</i></li> <li>2. Implementasi <i>Remove Duplicates</i> + Laporan <i>Duplicates</i></li> <li>3. Implementasi <i>Feature Scaling: Min-Max Scaling, Robust, and Standardization</i> + Laporan <i>Feature Scaling</i></li> <li>4. Implementasi <i>Handling Imbalanced Dataset</i> using SMOTE method + Laporan <i>Handling Imbalanced Dataset</i></li> <li>5. Implementasi <i>K-Nearest Neighbor (KNN)</i> dengan Scikit Learn dan Analisis + Laporan Bagian Implementasi Algoritma KNN</li> <li>6. <i>Error Analysis</i> (Analisis Model)</li> <li>7. Laporan Bagian Latar Belakang, Deskripsi Masalah, <i>Feature Engineering</i>, dan saran.</li> </ol>
<b>18222063</b>	Muhammad Faiz Atharrahman	<ol style="list-style-type: none"> <li>1. <i>Handling Missing Value: Domain Specific Strategies</i> 10 column</li> <li>2. Implementasi <i>K-Nearest Neighbor (KNN)</i> dan Laporan Bagian Implementasi Algoritma <i>K-Nearest Neighbor (KNN)</i></li> <li>3. Implementasi <i>Naive-Bayes from Scratch</i> dan Laporan Bagian Implementasi Algoritma <i>Naive-Bayes</i></li> <li>4. <i>Testing and Improvement</i> untuk Algoritma <i>K-Nearest Neighbor (KNN)</i></li> <li>5. <i>Testing and Improvement</i> untuk Algoritma <i>Naive-Bayes</i></li> <li>6. <i>Handling Kaggle Submission with Dataset Test</i></li> </ol>

## REFERENSI

- UCI Machine Learning Repository. (n.d.). *PHIUSHIL Phishing URL Dataset*. Retrieved from <https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset>
- Scikit-learn Developers. (2023). Gaussian Naive Bayes. Retrieved from [https://scikit-learn.org/stable/modules/generated/sklearn.naive\\_bayes.GaussianNB.html](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html)
- Scikit-learn Developers. (2023). Naive Bayes. Retrieved from [https://scikit-learn.org/stable/modules/naive\\_bayes.html](https://scikit-learn.org/stable/modules/naive_bayes.html)
- Scikit-learn Developers. (2023). Classification Report. Retrieved from [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\\_report.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York: Springer.
- GeeksforGeeks Team. (2023). Python Program to Validate an IP Address. Retrieved from [https://www.geeksforgeeks.org/python-program-to-validate-an-ip-address/?utm\\_source=chatgpt.com](https://www.geeksforgeeks.org/python-program-to-validate-an-ip-address/?utm_source=chatgpt.com)
- GeeksforGeeks Team. (2023). Working with Missing Data in Pandas. Retrieved from <https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/>
- Saturn Cloud Team. (2023). How to Fill Missing Values of One Column from Another Column in Pandas. Retrieved from <https://saturncloud.io/blog/how-to-fill-missing-values-of-one-column-from-another-column-in-pandas/>
- Sling Academy Team. (2023). Pandas DataFrame fillna Method - 5 Examples. Retrieved from <https://www.slingacademy.com/article/pandas-dataframe-fillna-method-5-examples/>
- Python Software Foundation. (2023). urllib.parse — Parse URLs into components. Retrieved from <https://docs.python.org/3/library/urllib.parse.html>
- Sciencedirect. (2023). *A comprehensive analysis of phishing URL detection techniques*. Retrieved from <https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558?via%3Dihub>
- Scikit-learn Developers. (2023). *K-Nearest Neighbors algorithm*. Retrieved from <https://scikit-learn.org/1.5/modules/neighbors.html>
- Scikit-learn Developers. (2023). *Naive Bayes*. Retrieved from [https://scikit-learn.org/1.5/modules/naive\\_bayes.html](https://scikit-learn.org/1.5/modules/naive_bayes.html)

Author(s). (2023). *A comprehensive analysis of phishing URL detection techniques*. Computers & Security. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0167404823004558>

Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach (4th ed.)*. Pearson Education.