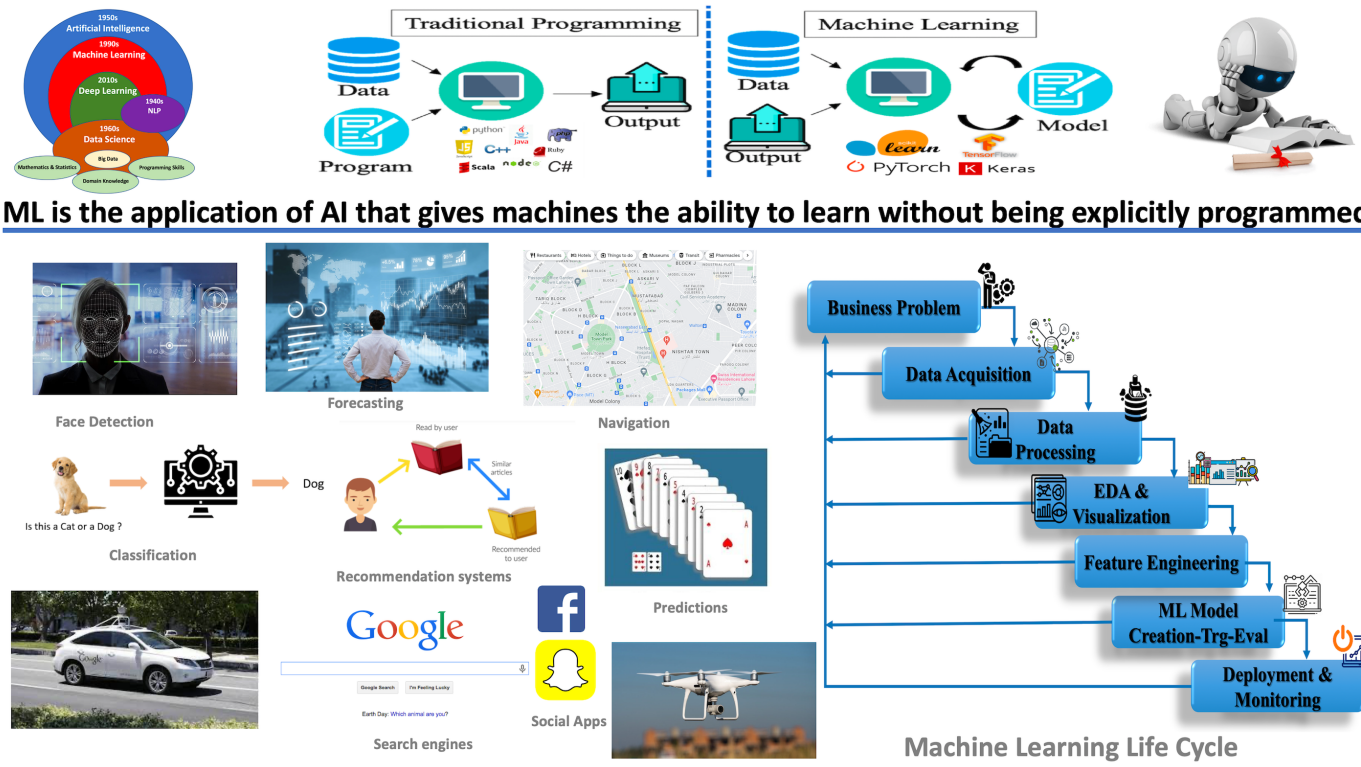
Department of Data Science

Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.8 (Data Preprocessing: Missing Values Imputation)

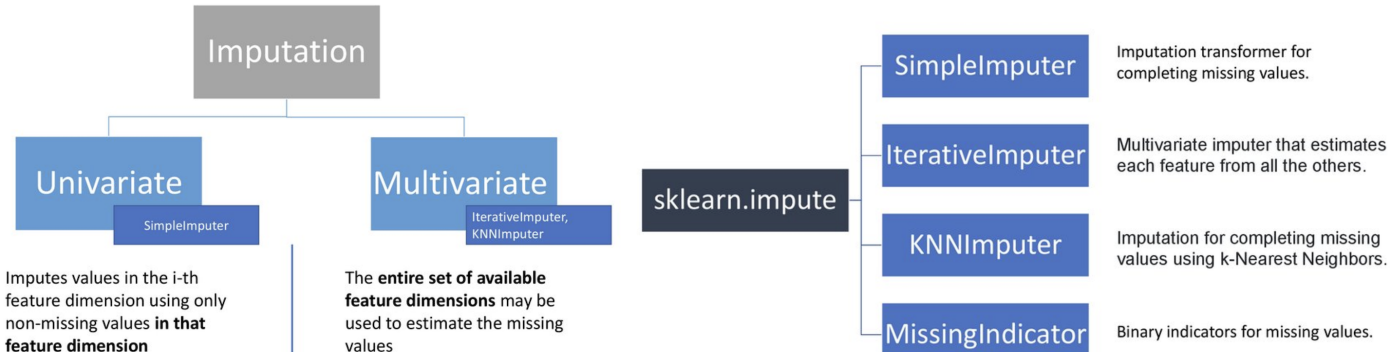
Open in Colab (https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)



Learning agenda of this notebook

- Overview of Data Pre-Processing and Feature Engineering
- Missing Data and its Types
  - MCAR
  - MAR

    - MNAR
- Univariate Imputation
    - Handling Missing Values using Panda's `fillna()` method
    - Handling Missing Values using sklearn's `SimpleImputer()` transformer
    - Use of Column Transformer
- Multivariate Imputation
    - Handling Missing Values using sklearn's `IterativeImputer()` transformer
    - Handling Missing Values using sklearn's `KNNImputer()` transformer



1. Overview of Data Pre-Processing and Feature Engineering

- `Data Preprocessing` involves actions that we need to perform on the dataset in order to make it ready to be fed to the machine learning model.
- `Feature Engineering` is the process of using domain knowledge to extract features from raw data via data mining techniques.

City	Size	Covered Area	No of bedrooms	Trees near by	No of bathrooms	Schools near by	Construction Date	Price
Lahore	2000	3500	3	1	3	1	25/10/2001	20.5 M
Karachi	2600	3000	2	0	4	1	16/05/1990	18 M
Islamabad	1800	2000	3	1	3	2	25/11/1995	20 M
Shaikhupura	1600	2600	1	2	NaN	0	08/06/2020	5 M
Lahore	2600	2000	3	3	1	1	03/09/2016	4 M
Karachi	3000	1000	2	2	1	NaN	19/01/1980	6 M
Islamabad	2000	3600	44	4	3	3	21/07/1999	30 M
Lahore	1000	2000	3	NaN	1	2	12/04/2015	10 M

- Pre-processing package of sklearn provides a bundle of utility functions and transformer classes for data preprocessing (will cover later).
    - **Detecting and handling outliers**
        - Univariate (Z-Score, IQR, Percentiles)
        - Multivariate Analysis (Depth-based, Distance-based, Density-based methods)
        - Trimming, Capping/Winsorization, Discritization
    - **Missing values Imputation**

    - Univariate Imputation (Panda's `fillna()` method, Sklearn's `SimpleImputer()` transformer)
    - Multivariate Imputation (Sklearn's `IterativeImputer()` and `KNNImputer()` transformers)
  - **Encoding Categorical Features**
    - Encode Nominal i/p features using Pandas `get_dummies()` and Scikit-Learn's `OneHotEncoder()`
    - Encode Ordinal i/p features using Scikit-Learn's `OrdinalEncoder()`
    - Encode categorical o/p label using Scikit-Learn's `LabelEncoder()`
  - **Feature Scaling**
    - Use numPy to perform maxabs, minmax, standard and robust scaling
    - Use Sklearn's `MaxAbsScalar` , `MinMaxScalar` , `StandardScalar` , `RobustScalar` transformers
  - **Extracting Information**
    - Use Sklearn's `CountVectorizer` , `DictVectorizer` , `TfidfVectorizer` , and `TfidfTransformer`
  - **Combining Information**
    - Use `FeatureUnion` , `Pipeline` , `PCA`

2. Missing Data and its Types

> **Missing data, or missing values, occur when no data value is stored for the variable in an observation .**

Inference and Missing Data, Donald Robin:
(https://www.math.wsu.edu/faculty/xchen/stat115/lectureNotes3/Rubin%20Inference%20and%20Missing%20D

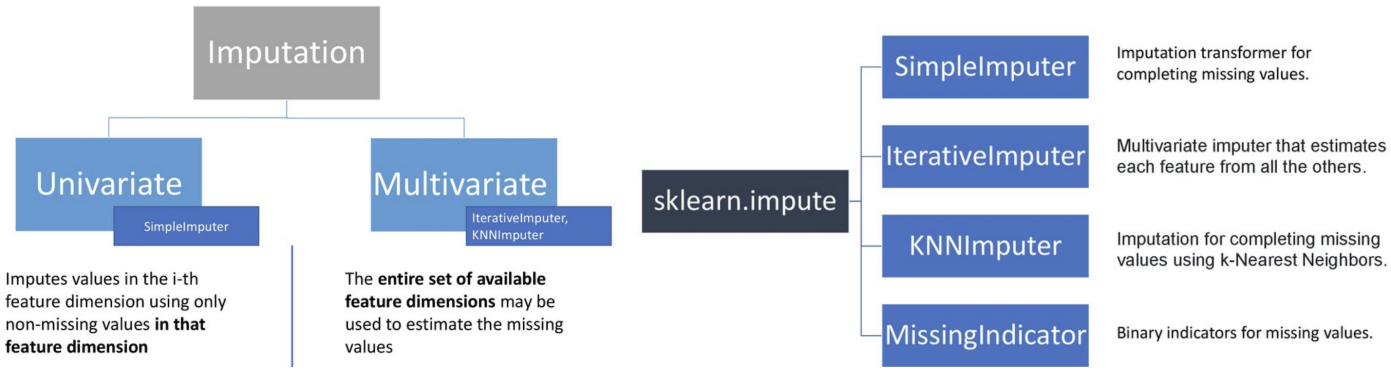
- **Missing Completely At Random (MCAR):**
  - Missing data is INDEPENDENT of observed and unobserved variables in the dataset (No relationship).
  - In the case of MCAR, the data could be missing due to human error, some system/equipment failure, loss of sample, or some unsatisfactory technicalities while recording the values.
  - For example, incomplete filling of Google survey form due to Internet connection failure.
  - If your data is MCAR, the statistical analysis remains unbiased.
- **Missing At Random (MAR):**
  - Missing data is DEPENDENT on some observed variable(s) in the dataset (Some relationship).
  - For example, if you check the survey data, you may find that all the people have answered their ‘Gender’ but ‘Age’ values are mostly missing for people who have answered their ‘Gender’ as ‘female’.
  - If your data is MAR, the statistical analysis might result in bias. Getting an unbiased estimate of the parameters can be done only by modeling the missing data.
- **Missing Not At Random (MNAR):**
  - Missing data is DEPENDENT on observed as well as unobserved variable(s) in the dataset.
  - For example, People having less income may refuse to share that information in a survey.
  - If your data is MNAR, the statistical analysis might result in bias. Getting an unbiased estimate of the parameters can be done only by modeling the missing data.

3. Techniques to Handle Missing Data

- **Why do we need to handle missing values?**

- If the missing values are not handled properly, you may end up building a biased machine learning model which will lead to incorrect results.
- Scikit-Learn's implementation of K-nearest and Naive Bayes do not support the presence of missing values.

- **How to treat missing values?**
  - Analyze each column with missing values carefully to understand the reasons behind the missing values as it is crucial to find out the strategy for handling the missing values. There are 2 primary ways of handling missing values:
    - Deleting the Missing values: Drop rows (List-wise deletion) having missing values or drop the entire column
    - Imputing the Missing Values: Replace the missing value with a value
      - `Univariate Imputation`
      - `Multivariate Imputation`



4. Handling Missing Values using Pandas

a. Load Dataset

In [1]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
df = pd.read_csv('datasets/loan-eligibility.csv')
df
```

Out[1]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

614 rows × 13 columns

b. Identify Missing values

In [2]:

```
# To check the count of non-null values in each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [3]:

```
# Returns a Boolean same sized object
df.isna()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	
609	False	False	False	False	False	False	False	
610	False	False	False	False	False	False	False	
611	False	False	False	False	False	False	False	
612	False	False	False	False	False	False	False	
613	False	False	False	False	False	False	False	

614 rows × 13 columns

In [4]:

```
# Returns a series object showing count of missing values under each column
df.isna().sum()
```

Out[4]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [5]:

```
# List comprehension to get the list of column names having missing values
cols = [var for var in df.columns if df[var].isna().sum() > 0]
cols
```

Out[5]:

```
['Gender',
 'Married',
 'Dependents',
 'Self_Employed',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

In [6]:

```
# Find the total number of missing values from the entire dataset
df.isna().sum().sum()
```

Out[6]:

```
149
```

c. Deleting the Missing values

Complete Case Analysis(CCA): This is a quite straightforward method of handling the Missing Data, which directly removes the rows that have missing data i.e we consider only those rows where we have complete data i.e data is not missing. This method is also popularly known as “Listwise deletion”.

- **Assumptions:**
  - Data is Missing At Random(MAR).
  - Missing data is completely removed from the table.
- **Advantages:**
  - Easy to implement.
  - No Data manipulation required.

- **Limitations:**
  - Deleted data can be informative.
  - Can lead to the deletion of a large part of the data.
  - Can create a bias in the dataset, if a large amount of a particular type of variable is deleted from it.
  - The production model will not know what to do with Missing data.
- **When to Use:**
  - Data is MCAR (Missing Completely At Random) or MAR(Missing At Random).
  - Good for Mixed, Numerical, and Categorical data.
  - Missing data is not more than 5% – 6% of the dataset.
  - Data doesn't contain much information and will not bias the dataset.

In [7]:

```
1 df.isna().mean()*100
```

Out[7]:

```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           3.583062
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

(i) Deleting the entire row

- The `df.dropna()` method is used to drop the rows/columns having NaN values:

`df.dropna(axis, how, subset, inplace)`

- Where,
  - axis=0 is used to drop the row with `NaN` values (default)
  - axis=1 is used to drop the column with `NaN` values
  - how='any' is used to drop the row/column, if any single value in it is `NaN` (default)
  - how='all' is used to drop the row/column, if all values in it are `NaN`
  - inplace=False will return the new dataframe (default)
  - inplace=True will make change to original dataframe and returns None

In [8]:

```
# You can use dropna() method to drop all the rows having NaN value
df1 = df.dropna(axis=0, how='any', inplace=False)
print(df1.shape)
df1.isnull().sum()
```

```
(480, 13)
```

Out[8]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

```
>- 134 rows out of 614 rows have been deleted, means you have deleted 22% of rows from your dataset :(
```

(ii) Deleting the entire column

- If a certain column has lot of missing values then you can choose to drop the entire column.
- But this is an extreme case and should only be used when there are many null values in the column.

In [9]:

```
# You can use dropna() method to drop all the rows having NaN value
df1 = df.dropna(axis=1, how='any', inplace=False)
print(df1.shape)
df1.isnull().sum()
```

```
(614, 6)
```

Out[9]:

```
Loan_ID              0
Education            0
ApplicantIncome      0
CoapplicantIncome    0
Property_Area        0
Loan_Status          0
dtype: int64
```

- 7 columns out of 13 have been deleted, which is ofcourse not good :(

d. Imputing the Missing Value (Univariate)

Imputation is a technique used for replacing the missing data with some substitute value to retain most of the information in the dataset

Load Dataset

In [10]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
df = pd.read_csv('datasets/loan-eligibility.csv')
df.head()
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [11]:

```
1 df.isnull().sum()
```

Out[11]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

e. Pandas `df.fillna()` Method for Imputation of Missing Values

- The `df.fillna()` method is used to fill NaN values using the specified `value` argument or the specified `method` argument.
- The only required argument is either the `value`, with which we want to replace the missing values OR the `method` to be used to replace the missing values

`df.fillna(value, method, inplace)`

- Where,
  - `value` argument specifies the value to be imputed at the place of missing values (required)
  - `method` can be either
    - `ffill`, which means moves forward and fill NaN with previous value
    - `bfill`, which means moves backward and fill NaN with previous value
  - inplace=False will return the new dataframe with missing values filled (default)
  - inplace=True will make change to original dataframe with missing values filled and returns None

(i) Replacing With Arbitrary Value

- If you can make an educated guess about the missing value then you can replace it with that value.
- This option is used when the data is not missing at random. If data is missing at random you should prefer mean/median

In [12]:

```
# replace all NaNs under all the columns with a string value "missing"
df1 = df.fillna(value="missing")
df1.isnull().sum()
```

Out[12]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [13]:

```
#Replace the missing value under the Dependents column with '0'
df1 = df.copy()
df1['Dependents'] = df['Dependents'].fillna(value=0, inplace=False)
df1.isnull().sum()
```

Out[13]:

```
Loan_ID              0
Gender              13
Married              3
Dependents           0
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

(ii) Replacing With Mean

- This is the most common method of imputing missing values of numeric columns. However, if there are outliers then the mean will not be appropriate. In such cases, outliers need to be treated first.
- While computing the mean the number of entities will be the number of non-null values

In [14]:

```
df['LoanAmount'].mean()
```

Out[14]:

```
146.41216216216216
```

In [15]:

```
df['Credit_History'].mean()
```

Out[15]:

```
0.8421985815602837
```

In [16]:

```
#Replace the missing values under the 'LoanAmount' and 'Credit_History' columns
df1['LoanAmount'] = df['LoanAmount'].fillna(value = df['LoanAmount'].mean())
df1['Credit_History'] = df['Credit_History'].fillna(value = df['Credit_History']
df1.isnull().sum()
```

Out[16]:

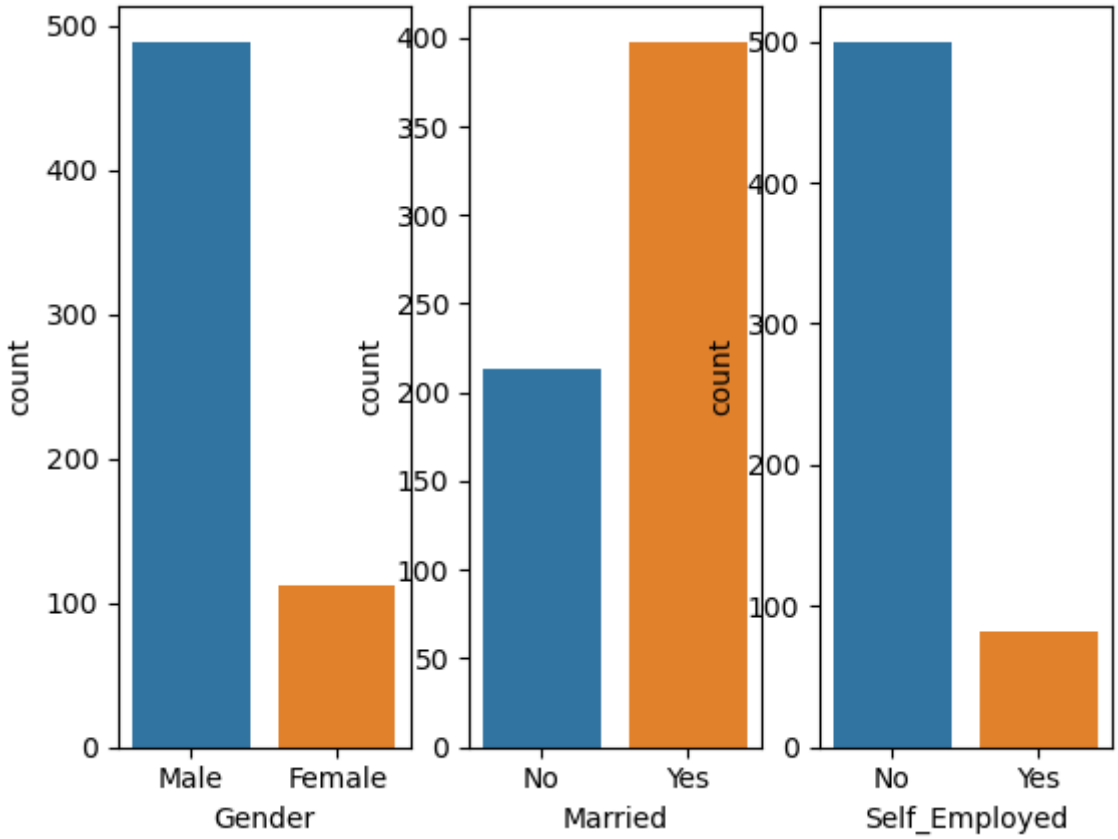
```
Loan_ID               0
Gender               13
Married               3
Dependents            0
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     14
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

(iii) Replacing With Mode

- Mode is the most frequently occurring value. It is used in the case of categorical features.
- You can use the `fillna()` method for imputing the categorical columns 'Gender', 'Married', and 'Self_Employed'.

In [17]:

```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3)
sns.countplot(x='Gender', data = df, ax=ax1)
sns.countplot(x='Married', data = df, ax=ax2)
sns.countplot(x='Self_Employed', data = df, ax=ax3)
plt.show();
```



In [18]:

```
type(df.mode())
```

Out[18]:

```
pandas.core.frame.DataFrame
```

In [19]:

```
#Replace the missing values for categorical columns with mode
#Replace the missing values under the 'Gender', 'Married', and 'Self_Employed' c
df1['Gender'] = df['Gender'].fillna(value = df['Gender'].mode()[0])
df1['Married'] = df['Married'].fillna(value = df['Married'].mode()[0])
df1['Self_Employed'] = df['Self_Employed'].fillna(value = df['Self_Employed'].mo
df1.isnull().sum()
```

Out[19]:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     14
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

(iv) Replacing With Median

- Median is the middlemost value. It's better to use the median value for imputation in the case of outliers.
- Moreover, we all know that if the distribution is skewed then the mean gets shifted, so median is a better candidate than mean.
- You can use 'fillna' method for imputing the column 'Loan_Amount_Term' with the median value.

In [20]:

```
df['Loan_Amount_Term'].median()
```

Out[20]:

```
360.0
```

In [21]:

```
df1['Loan_Amount_Term']= df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].me
df1.isnull().sum()
```

Out[21]:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term      0
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

(v) Replacing with previous value `ffill` or next value `bfill`

- In some cases, imputing the values with the previous value instead of mean, mode or median is more appropriate. This is called forward fill. It is mostly used in time series data.

In [22]:

```
x = pd.Series(range(1,7))
x[2] = np.nan
x[4] = np.nan
x
```

Out[22]:

```
0    1.0
1    2.0
2    NaN
3    4.0
4    NaN
5    6.0
dtype: float64
```

In [23]:

```
# Forward-Fill
x.fillna(method='ffill', inplace=False)
```

Out[23]:

```
0    1.0
1    2.0
2    2.0
3    4.0
4    4.0
5    6.0
dtype: float64
```

In [24]:

```
# Backward-Fill
x.fillna(method='bfill', inplace=False)
```

Out[24]:

```
0    1.0
1    2.0
2    4.0
3    4.0
4    6.0
5    6.0
dtype: float64
```

f. Check the impact of Imputation

- **Check out the following before after the imputation:**
  - Variance
  - Distribution
  - Outliers
  - Covariance and correlation with other columns

Check out change in `variance` of `LoanAmount` column, if we impute it with mean vs median

In [25]:

```
df['LoanAmount_mean'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['LoanAmount_median'] = df['LoanAmount'].fillna(df['LoanAmount'].median())

print("Variance of original LoanAmount with missing values:", np.var(df['LoanAmo
print("Variance of imputed LoanAmount with mean:", np.var(df['LoanAmount_mean'])
print("Variance of imputed LoanAmount with median:", np.var(df['LoanAmount_media
```

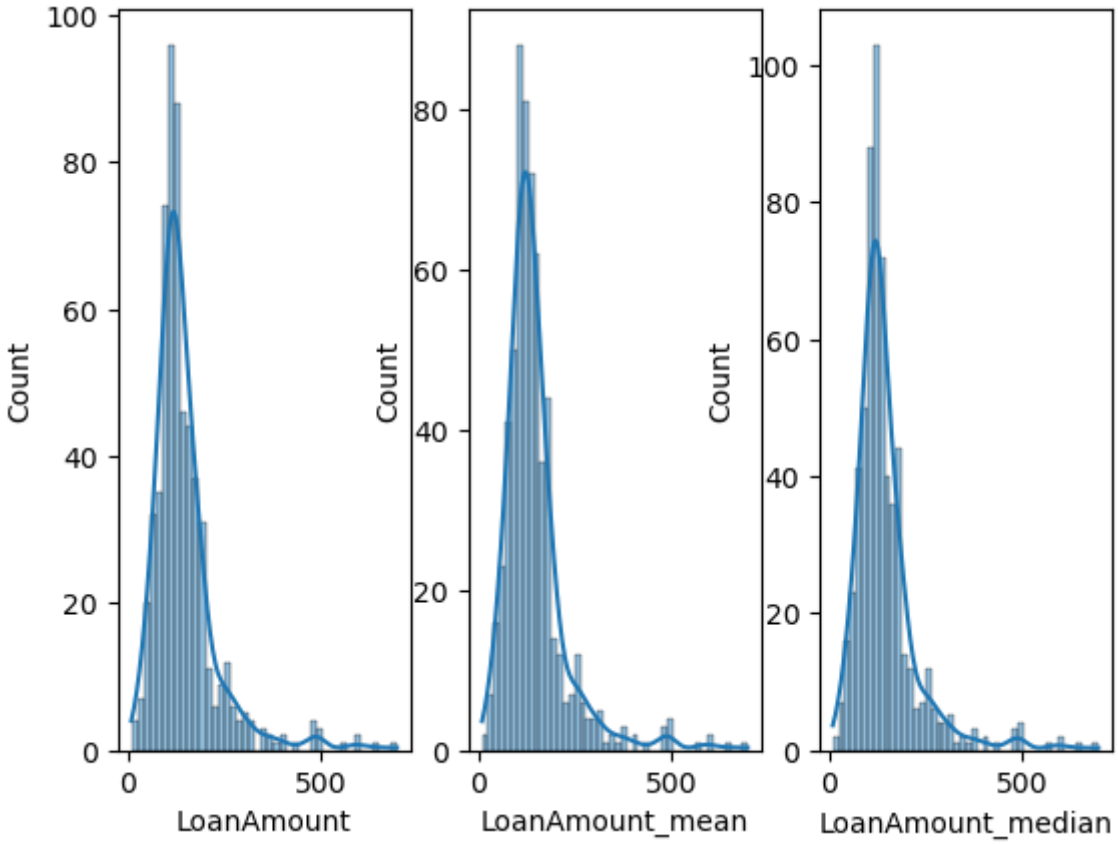
```
Variance of original LoanAmount with missing values: 7312.816608838569
Variance of imputed LoanAmount with mean: 7050.7938638964715
Variance of imputed LoanAmount with median: 7062.505490774439
```

Check out change in `distribution` of `LoanAmount` column, if we impute it with mean vs median

In [26]:

```
df['LoanAmount_mean'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['LoanAmount_median'] = df['LoanAmount'].fillna(df['LoanAmount'].median())


fig, (ax1,ax2,ax3) = plt.subplots(nrows=1, ncols=3)
sns.histplot(df['LoanAmount'], kde=True, ax=ax1)
sns.histplot(df['LoanAmount_mean'], kde=True, ax=ax2)
sns.histplot(df['LoanAmount_median'], kde=True, ax=ax3)
plt.show();
```



5. Handling Missing Values using Scikit-Learn Transformers

Data Transformers in Scikit-Learn: (https://scikit-learn.org/stable/data_transforms.html)

Transformers are objects that transform a dataset in order to prepare it for predictive modeling.

a. The `impute.SimpleImputer()` method of Scikit-Learn

Step 1: Import appropriate imputer class

```
from sklearn.impute import
SimpleImputer,IterativeImputer,KNNImputer
```

Step 2: Define imputer instance

```
imp = SimpleImputer(missing_values=np.nan, strategy='mean',
fill_value=None)
```

Step 3: Fit imputer instance on the dataset

```
imp.fit(X)
```

Step 4: Transform the dataset

Load Dataset

In [27]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
df = pd.read_csv('datasets/loan-eligibility.csv')
df.head()
```

Out[27]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [28]:

```
df.isnull().sum()
```

Out[28]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Impute a constant value 0 for missing values under the Dependents Column

In [29]:

```
from sklearn.impute import SimpleImputer
# define imputer
imp = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value=0)
```

In [30]:

```
# fit on the dataset
imp.fit(df.iloc[:,3:4])
```

Out[30]:

```
SimpleImputer(fill_value=0, strategy='constant')
```

In [31]:

```
# transform the dataset
df.iloc[:,3:4] = imp.transform(df.iloc[:,3:4])
```

In [32]:

```
# verify
df.isnull().sum()
```

Out[32]:

```
Loan_ID               0
Gender               13
Married               3
Dependents            0
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Impute `most_frequent` value for missing values under the `Gender`, `Married` and `Self-Employed` Column

In [33]:

```
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
df.iloc[:,[1,2,5]] = imp.fit_transform(df.iloc[:,[1,2,5]])
```

In [34]:

```
df.isnull().sum()
```

Out[34]:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Impute `mean` value for missing values under the `LoanAmount`, `Loan_Amount_Term` and `Credit_History` Column

In [35]:

```
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
df.iloc[:,8:11] = imp.fit_transform(df.iloc[:,8:11])
```

In [36]:

```
#verify
df.isnull().sum()
```

Out[36]:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term      0
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

Limitation of SimpleImputer

- SimpleImputer is a transformer that works on entire data and it cannot be applied on a particular column.
- For each missing value type, we have to define a separate imputer and fit-transform one by one.
- Solution: `ColumnTransformer` that allows transformation in different columns with different imputations and applies at the same time.

b. The `compose.ColumnTransformer()` method of Scikit-Learn

ColumnTransformer allows transformation in different columns with different imputations and applies at the same time

Step 1: Import ColumnTransformer

```
from sklearn.compose import ColumnTransformer
```

Step 2: Create SimpleImputer instances

```
imp_const = SimpleImputer(missing_values=np.nan,
strategy='constant', fill_value=0)
imp_mode = SimpleImputer(missing_values=np.nan,
strategy='most_frequent')
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
```

Step 3: Define ColumnTransformer instance

```
column_trans = ColumnTransformer([
                                    ('impute_dep', imp_const,
[3]),
                                    ('impute_gend-marr-emp',
imp_mode, [1,2,5]),
                                    ('impute_loan-amt-cr',
imp_mean, [8,9,10])
                                   ],
                                    remainder='passthrough')
```

Step 4: Call fit_transform on ColumnTransformer instance

```
result_arr = column_trans.fit_transform(df)
```

Load Dataset

In [37]:

```
1 df = pd.read_csv('datasets/loan-eligibility.csv')
2 df.isnull().sum()
```

Out[37]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [38]:

```
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

# create the transformers
imp_const = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value
imp_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')

# define ColumnTransformer
column_trans = ColumnTransformer(
                                 [('impute_dep', imp_const, [3]),
                                  ('impute_gend-marr-emp', imp_mode, [1,2,5]),
                                  ('impute_loan-amt-cr', imp_mean, [8,9,10])],
                                  remainder='passthrough')

#Call fit-transform
result_arr = column_trans.fit_transform(df)
print(result_arr)
print(result_arr.shape)
```

```
[['0' 'Male' 'No' ... 0.0 'Urban' 'Y']
 ['1' 'Male' 'Yes' ... 1508.0 'Rural' 'N']
 ['0' 'Male' 'Yes' ... 0.0 'Urban' 'Y']
 ...
 ['1' 'Male' 'Yes' ... 240.0 'Urban' 'Y']
 ['2' 'Male' 'Yes' ... 0.0 'Urban' 'Y']
 ['0' 'Female' 'No' ... 0.0 'Semiurban' 'N']]
(614, 13)
```

In [39]:

```
df_imputed = pd.DataFrame(data=result_arr, columns = df.columns)
#verify
df_imputed.isnull().sum()
```

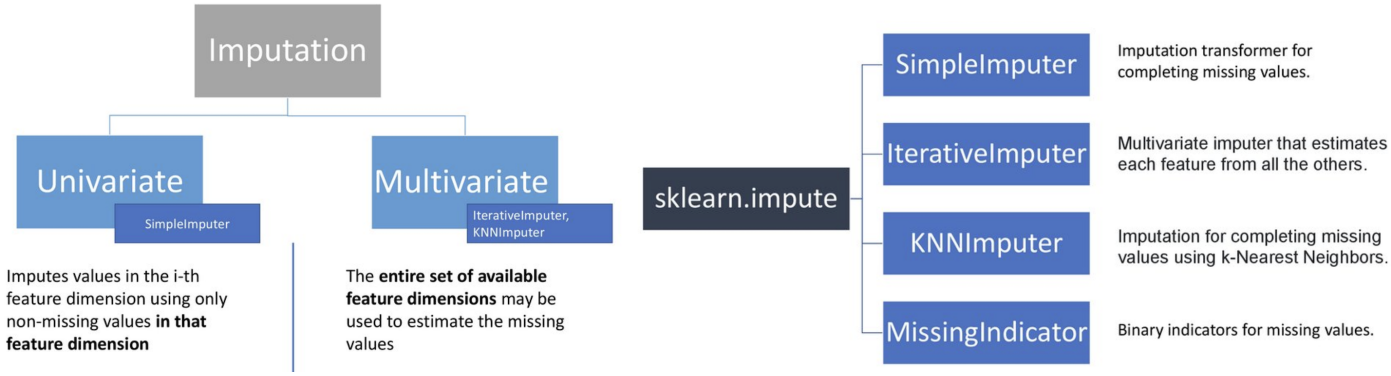
Out[39]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Limitation of ColumnTransformer

- In a ColumnTransformer we cannot apply multiple transforms to a single column
- Solution: `Pipeline` which is a sequence of operations where output of one operation becomes input to its subsequent operation

c. The Multivariate Imputation



(i) The `impute.IterativeImputer()` method of Scikit-Learn

IterativeImputer impute the missing values by modeling each feature having missing values as a function of other features in a round-robin fashion

How the Algorithm Work?

- Step 1: A feature column having NaN is designated as output and the other feature columns are treated as inputs.
- Step 2: Regressor predicts missing output.
- Step 3: This is done for each feature in an iterative fashion, and then repeated for specified iterations.
- Step 4: Results from the final iteration are used for imputation

Feature # 1	Feature # 2	Feature # 3	Feature # 4	Feature # 5
1	2	12	0.33	100
NaN	2	13	0.66	112
5	NaN	NaN	0.34	212
Output	Input			

Feature # 1	Feature # 2	Feature # 3	Feature # 4	Feature # 5
1	2	12	0.33	100
NaN	2	13	0.66	112
5	NaN	NaN	0.34	212
Input	Output	Input		

Feature # 1	Feature # 2	Feature # 3	Feature # 4	Feature # 5
1	2	12	0.33	100
NaN	2	13	0.66	112
5	NaN	NaN	0.34	212
Input	Output	Input		

```
it_imputer = IterativeImputer(estimator=None,
                              missing_values=np.nan,
                              initial_strategy='mean',
                              n_nearest_features=None,
                              max_iter=10,
                              imputation_order='ascending')
```

Sample Code

In [40]:

```
import pandas as pd
X = pd.DataFrame({
                  'x1':[np.nan,2,3,4,5],
                  'x2':[6,np.nan,8,9,10],
                  'x3':[11,12,np.nan,14,15],
                  'x4':[16,17,18,19,np.nan]
                  })
X
```

Out[40]:

	x1	x2	x3	x4
0	NaN	6.0	11.0	16.0
1	2.0	NaN	12.0	17.0
2	3.0	8.0	NaN	18.0
3	4.0	9.0	14.0	19.0
4	5.0	10.0	15.0	NaN

In [41]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [42]:

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
it_imp = IterativeImputer(estimator=lr, missing_values=np.nan, initial_strategy=
type(it_imp)
```

Out[42]:

```
sklearn.impute._iterative.IterativeImputer
```

In [43]:

```
it_imp.fit_transform(X)
```

Out[43]:

```
array([[ 1.,  6., 11., 16.],
       [ 2.,  7., 12., 17.],
       [ 3.,  8., 13., 18.],
       [ 4.,  9., 14., 19.],
       [ 5., 10., 15., 20.]])
```

(ii) The `impute.KNNImputer()` method of Scikit-Learn

Impute the missing values by using the mean value from K nearest neighbours found in the training set.

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

$$nan\_distance = \sqrt{weight * (\text{distance from present coordinates})^2}$$

$$weight = \frac{\text{Total number of coordinates}}{\text{number of present coordinates}}$$

	x1	x2	x3	x4
0	23.0	NaN	57	11.0
1	NaN	35.0	58	2.0
2	13.0	41.0	61	8.0
3	30.0	NaN	71	NaN
4	25.0	50.0	69	NaN

$$\text{dist(r0, r1)} = \sqrt{\tfrac{3}{2}((58-57)^2 + (2-11)^2)} = 11.09$$

$$\text{dist(r2, r1)} = \sqrt{\tfrac{3}{3}((35-41)^2 + (58-61)^2) + (2-8)^2)} = 9$$

$$\text{dist(r3, r1)} = \sqrt{\tfrac{3}{1}((58-71)^2)} = 22.5$$

$$\text{dist(r4, r1)} = \sqrt{\tfrac{3}{2}((35-50)^2 + (58-69)^2)} = 22.7$$

$$\text{Imputed Value} = \frac{23+13}{2} = 18$$

```
knn_im = KNNImputer(missing_values=np.nan,
                    n_neighbors=5,
                    weights='uniform',
                    metric='nan_euclidean')
```

In [44]:

```
import pandas as pd
X = pd.DataFrame({
                  'x1':[23,np.nan, 13, 30,25],
                  'x2':[np.nan, 35,41,np.nan,50],
                  'x3':[57,58,61,71,69],
                  'x4':[11,2,8,np.nan, np.nan]
                  })
X
```

Out[44]:

	x1	x2	x3	x4
0	23.0	NaN	57	11.0
1	NaN	35.0	58	2.0
2	13.0	41.0	61	8.0
3	30.0	NaN	71	NaN
4	25.0	50.0	69	NaN

In [45]:

```
from sklearn.impute import KNNImputer
knn_imp = KNNImputer(n_neighbors=2)
type(knn_imp)
```

Out[45]:

```
sklearn.impute._knn.KNNImputer
```

In [46]:

```
knn_imp.fit_transform(X)
```

Out[46]:

```
array([[23. , 38. , 57. , 11. ],
       [18. , 35. , 58. ,  2. ],
       [13. , 41. , 61. ,  8. ],
       [30. , 42.5, 71. ,  6.5],
       [25. , 50. , 69. ,  9.5]])
```

Task To Do (Assignment)

(i) Use `fillna()` and save the resulting dataframe

(ii) Apply `SimpleImputer` using `ColumnTransformer` and save the resulting dataframe

In [48]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
df = pd.read_csv('datasets/loan-eligibility.csv')
df
```

Out[48]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

614 rows × 13 columns