

In [2]:



How to install the environment for writing codes?

- Install Anaconda

<https://www.anaconda.com/download/success>

- Installing Jupyter
- Setup Environment
- Download Libraries
 - pip install pandas
 - pip install scikit-learn

Reading Data and Checking for Missing Values

Loading dataset

```
In [1]: import pandas as pd

# Load your dataset into a DataFrame
df = pd.read_csv('Diabetes Missing Data.csv')

# Preview the first 5 rows
df.head()
```

```
Out[1]:
```

	Pregnant	Glucose	Diastolic_BP	Skin_Fold	Serum_Insulin	BMI	Diabetes_Pedigree	Age
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33

Basic Summary Statistics

Use the describe() method to get basic statistics such as mean, median, minimum, maximum, and quartiles for each numeric column.

```
In [2]: # Get descriptive statistics for numerical columns
df.describe()
```

```
Out[2]:
```

	Pregnant	Glucose	Diastolic_BP	Skin_Fold	Serum_Insulin	BMI	Diabetes
count	768.000000	763.000000	733.000000	541.000000	394.000000	757.000000	
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	
std	3.369578	30.535641	12.382158	10.476982	118.775855	6.924988	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	
25%	1.000000	99.000000	64.000000	22.000000	76.250000	27.500000	
50%	3.000000	117.000000	72.000000	29.000000	125.000000	32.300000	
75%	6.000000	141.000000	80.000000	36.000000	190.000000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

Check Missing Values

It is crucial to check for missing data and understand how much of the data is missing in each column.

```
In [3]: # Check for missing values
df.isnull().sum()
```

```
Out[3]: Pregnant          0
        Glucose          5
        Diastolic_BP     35
        Skin_Fold        227
        Serum_Insulin     374
        BMI              11
        Diabetes_Pedigree 0
        Age              0
        Class            0
        dtype: int64
```

```
In [4]: len(df)
```

```
Out[4]: 768
```

How to handle missing values?

When to Remove Columns with Missing Values

- Use Case: When a column has a large amount of missing data, say more than 30-50% of its values are missing.

```
In [6]: missing_percentage = df.isnull().mean() * 100

# Filter columns with more than 30% (or 50%) missing values
columns_with_missing_30 = missing_percentage[missing_percentage > 30]

# Display the columns with more than 30% missing values
print("Columns with more than 30% missing values:")
print(columns_with_missing_30)
```

```
Columns with more than 30% missing values:
Serum_Insulin    48.697917
dtype: float64
```

How Remove column

```
In [7]: missing_percentage = df.isnull().mean() * 100

threshold = 30

# Filter columns with more than the threshold percentage of missing values
columns_to_drop = missing_percentage[missing_percentage > threshold].index

# Drop the columns
df_cleaned = df.drop(columns=columns_to_drop)

# Display the resulting dataframe after removing columns with too many missing values
print(f"Columns dropped: {list(columns_to_drop)}")
print(f"Remaining columns: {df_cleaned.columns}")
```

```
Columns dropped: ['Serum_Insulin']
Remaining columns: Index(['Pregnant', 'Glucose', 'Diastolic_BP', 'Skin_Fold', 'BMI',
                          'Diabetes_Pedigree', 'Age', 'Class'],
                          dtype='object')
```

Imputation with Mean/Median/Mode

When to Use: When the missing values are numerical and the missing data is random (MCAR), this method is suitable.

- Mean Imputation: Replace missing values with the mean of the column.
- Median Imputation: Use the median when data is skewed or has outliers.
- Mode Imputation: For categorical data, use the mode.

```
In [13]: data = pd.DataFrame({
          'A': [1, 2, None, 4],
          'B': [5, None, 7, 8],
          'C': ['cat', None, 'dog', 'cat']
        })

# Mean imputation for numerical columns
data['A'].fillna(data['A'].mean(), inplace=True)

# Median imputation for numerical columns
data['B'].fillna(data['B'].median(), inplace=True)

# Mode imputation for categorical columns
data['C'].fillna(data['C'].mode()[0], inplace=True)

print(data)
```

	A	B	C
0	1.000000	5.0	cat
1	2.000000	7.0	cat
2	2.333333	7.0	dog
3	4.000000	8.0	cat

Imputation Using Forward/Backward Fill

When to Use: In time series data or ordered data, where the previous or next observation can provide a good estimate for the missing value.

- Forward Fill: Use the previous valid value to fill the missing data.
- Backward Fill: Use the next valid value to fill missing data.

```
In [14]: # Example dataset (ordered)
data = pd.DataFrame({
          'A': [1, None, 3, None, 5],
          'B': [None, 6, None, 8, None]
        })

# Forward fill
```

```
data_forward_fill = data.fillna(method='ffill')

# Backward fill
data_backward_fill = data.fillna(method='bfill')

print(data_forward_fill)
print(data_backward_fill)
```

```

      A      B
0  1.0  NaN
1  1.0  6.0
2  3.0  6.0
3  3.0  8.0
4  5.0  8.0
      A      B
0  1.0  6.0
1  3.0  6.0
2  3.0  8.0
3  5.0  8.0
4  5.0  NaN
```

K-Nearest Neighbors (KNN) Imputation

- KNN imputation identifies the k-nearest samples and uses them to estimate the missing values.
- Pros: Captures relationships between variables and provides a more robust estimate than mean/median imputation.
- Cons: Computationally expensive, especially with large datasets; results may vary depending on the choice of k.

In []: `### pip install scikit-learn`

In [15]: `from sklearn.impute import KNNImputer`

```
# Example dataset
data = pd.DataFrame({
    'A': [1, 2, None, 4],
    'B': [5, None, 7, 8],
    'C': [9, 10, 11, None]
})

# KNN imputation (use n_neighbors=2, you can adjust this)
imputer = KNNImputer(n_neighbors=2)
data_imputed_knn = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)

print(data_imputed_knn)
```

```

      A      B      C
0  1.0  5.0  9.0
1  2.0  6.0 10.0
2  3.0  7.0 11.0
3  4.0  8.0 10.5
```

Key Considerations to Choose the Best Approach:

- Percentage of Missing Data:

Low (<5%): Consider deletion or simple imputation (mean/median/mode). Moderate (5-20%): Consider more sophisticated methods like KNN.

Type of Data:

- Numerical: Mean, median, KNN.
- Categorical: Mode imputation, KNN imputation.
- Time Series: Interpolation or forward/backward fill.

In []: