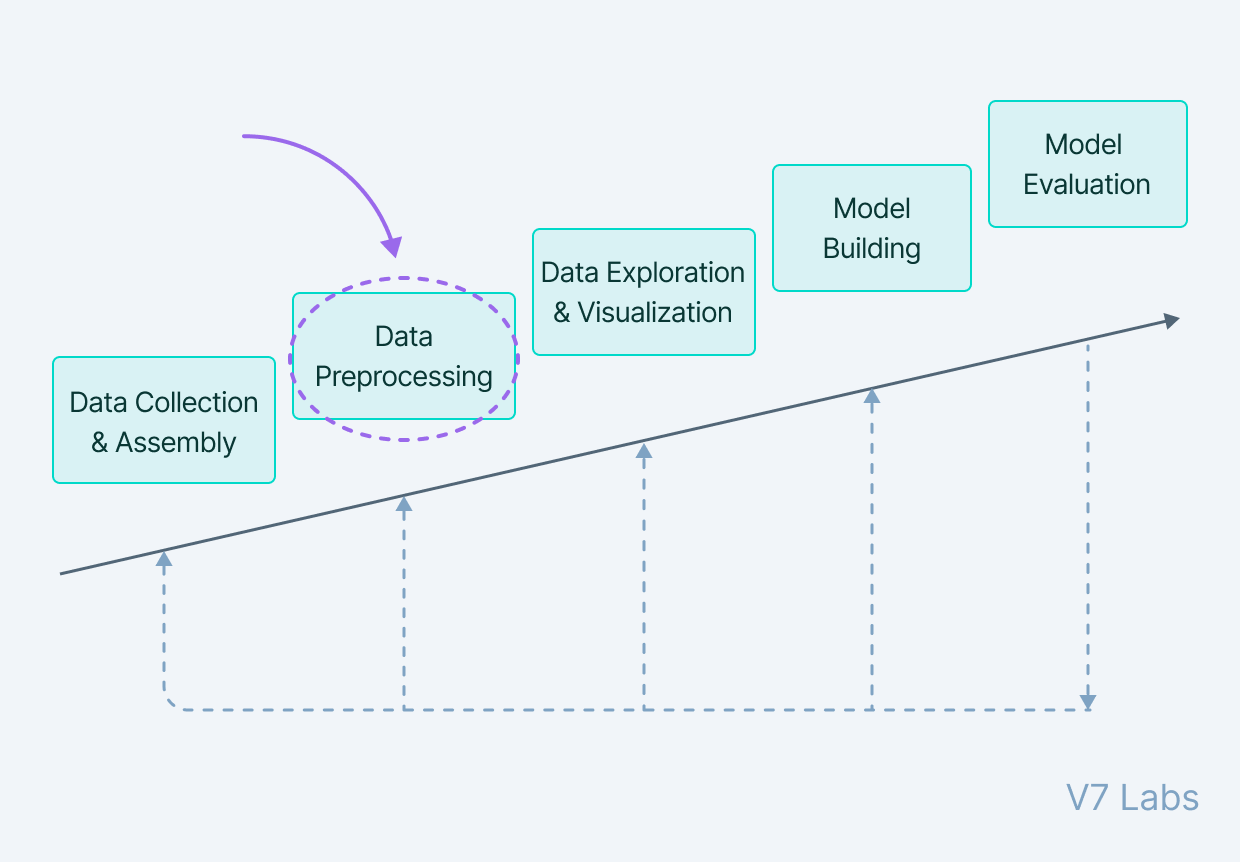
**Why is data preprocessing required?**

Data preprocessing is mainly required for the following:

* **Accurate data:** For making the data readable for machine learning model, it needs to be accurate with no missing value, redundant or duplicate values.
* **Trusted data:** The updated data should be as accurate or trusted as possible.
* **Understandable data:** The data updated needs to be interpreted correctly.



**Examples of data preprocessing for different data set types with Python**

Since data comes in various formats, let us discuss how different data types can be converted into a format that the ML model can read accurately. Let us see how to feed correct features from datasets with:

* Missing values
* Outliers
* Overfitting
* Data with no numerical values
* Different date formats

### Data Cleaning

[Data Cleaning](https://www.v7labs.com/blog/data-cleaning-guide) is particularly done as part of data preprocessing to clean the data by filling missing values, smoothing the noisy data, resolving the inconsistency, and removing outliers.

#### 1. Missing values

Here are a few ways to solve this issue:

* Ignore those tuples

This method should be considered when the dataset is huge and numerous missing values are present within a tuple.

* Fill in the missing values

There are many methods to achieve this, such as filling in the values manually, predicting the missing values using regression method, or numerical methods like attribute mean.

#### 2. Noisy Data

It involves removing a random error or variance in a measured variable. It can be done with the help of the following techniques:

* Binning

It is the technique that works on sorted data values to smoothen any noise present in it. The data is divided into equal-sized bins, and each bin/bucket is dealt with independently. All data in a segment can be replaced by its mean, median or boundary values.

* Regression

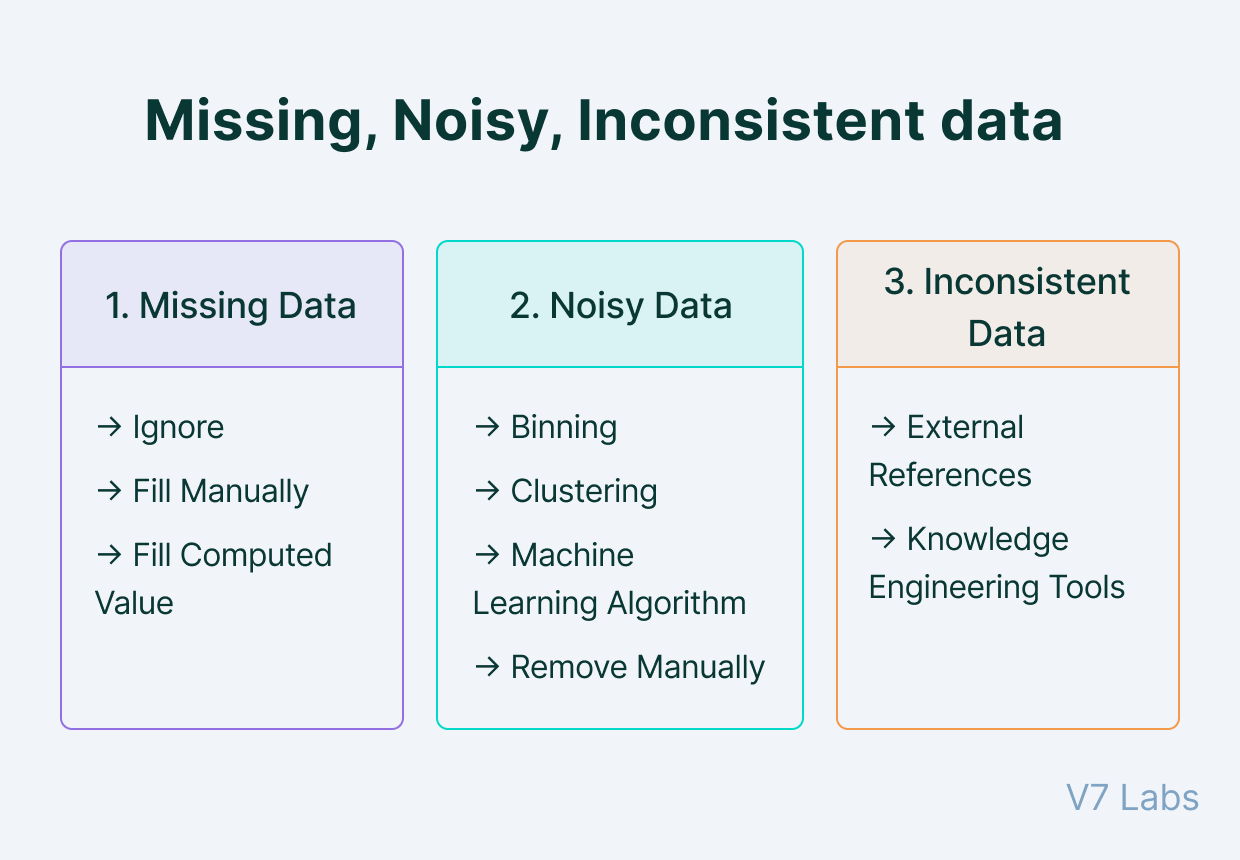
This data mining technique is generally used for prediction. It helps to smoothen noise by fitting all the data points in a regression function. The linear regression equation is used if there is only one independent attribute; else Polynomial equations are used.

* Clustering

Creation of groups/clusters from data having similar values. The values that don't lie in the cluster can be treated as noisy data and can be removed.

#### 3. Removing outliers

Clustering techniques group together similar data points. The tuples that lie outside the cluster are outliers/inconsistent data.



For dropping the values, following Python codes are used:

|  |  |
| --- | --- |
|  |  |

#Dropping columns in the data higher than 60% threshold

data = data[data.columns[data.isnull().mean() < threshold]]

#Dropping rows in the data higher than 60% threshold

data = data.loc[data.isnull().mean(axis=1) < threshold]

By using the above Python codes, the missed values will be dropped and the machine learning model will learn on the rest of the data.

Let us see the Python codes for numerical imputation, which are as follows:

#For filling all the missed values as 0

data = data.fillna(0)

#For replacing missed values with median of columns

data = data.fillna(data.median())

**Categorical imputation**

This technique of imputation is nothing but replacing the missed values in the data with the one which occurs the maximum number of times in the column. But, in case there is no such value that occurs frequently or dominates the other values, then it is best to fill the same as “NAN”.

The following Python code can be used here:

#Categorical imputation

data['column\_name'].fillna(data['column\_name'].value\_counts().idxmax(), inplace=True)

### Outliers

An outlier differs significantly from other values and is too distanced from the mean of the values. Such values that are considered outliers are usually due to some systematic errors or flaws.

Let us see the following Python codes for identifying and removing outliers with [standard deviation](https://quantra.quantinsti.com/glossary/Standard-Deviation):

#For identifying the outliers with the standard deviation method

outliers = [x for x in data if x < lower or x > upper]

print('Identified outliers: %d' % len(outliers))

#Remove outliers

outliers\_removed = [x for x in data if x >= lower and x <= upper]

print('Non-outlier observations: %d' % len(outliers\_removed))

In the codes above, “lower” and “upper” signify the upper and lower limit in the data set.

**Overfitting**

In both machine learning and statistics, [overfitting](https://blog.quantinsti.com/machine-learning-basics/) occurs when the model fits the data too well or simply put when the model is too complex.

Overfitting model learns the detail and noise in the training data to such an extent that it negatively impacts the performance of the model on new data/test data.

The overfitting problem can be solved by decreasing the number of features/inputs or by increasing the number of training examples to make the machine learning algorithms more generalised.

The most common solution is regularisation in an overfitting case. Binning is the technique that helps with the regularisation of the data which also makes you lose some data every time you regularise it.

For instance, in the case of numerical binning, the data can be as follows:

|  |  |
| --- | --- |
| **Stock value** | **Bin** |
| 100-250 | Lowest |
| 251-400 | Mid |
| 401-500 | High |

Here is the Python code for binning:

|  |  |
| --- | --- |
|  | data['bin'] = pd.cut(data['value'], bins=[100,250,400,500], labels=["Lowest", "Mid", "High"]) |

[view raw](https://gist.github.com/quantra-go-algo/bb93428bf25b6860ef458482c5d3b22a/raw/400e0f7ade16f0714a5bc990a493e485853951d5/Binning.py)[Binning.py](https://gist.github.com/quantra-go-algo/bb93428bf25b6860ef458482c5d3b22a#file-binning-py)hosted with ❤ by [GitHub](https://github.com/)

Your output should look something like this:

Value Bin

0 102 Low

1 300 Mid

2 107 Low

3 470 High

**Data with no numerical values**

In the case of the data set with no numerical values, it becomes impossible for the machine learning model to learn the information.

The machine learning model can only handle numerical values and thus, it is best to spread the values in the columns with assigned binary numbers “0” or “1”. This technique is known as **one-hot encoding**.

In this type of technique, the grouped columns already exist. For instance, below I have mentioned a grouped column:

|  |  |
| --- | --- |
| **Infected** | **Covid variants** |
| 2 | Delta |
| 4 | Lambda |
| 5 | Omicron |
| 6 | Lambda |
| 4 | Delta |
| 3 | Omicron |
| 5 | Omicron |
| 4 | Lambda |
| 2 | Delta |

Now, the above-grouped data can be encoded with the binary numbers ”0” and “1” with one hot encoding technique. This technique subtly converts the categorical data into a numerical format in the following manner:

|  |  |  |  |
| --- | --- | --- | --- |
| **Infected** | **Delta** | **Lambda** | **Omicron** |
| 2 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 |
| 6 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 |
| 5 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 |

Hence, it results in better handling of grouped data by converting the same into encoded data for the machine learning model to grasp the encoded (which is numerical) information quickly.

**Different date formats**

With the different date formats such as “25-12-2021”, “25th December 2021” etc. the machine learning model needs to be equipped with each of them. Or else, it is difficult for the machine learning model to understand all the formats.

With such a data set, you can preprocess or decompose the data by mentioning three different columns for the parts of the date, such as Year, Month and Day.

In Python, the preprocessing of the data with different columns for the date will look like this:

|  |  |
| --- | --- |
|  | #Convert to datetime object |
|  | df['Date'] = pd.to\_datetime(df['Date']) |
|  |  |
|  | #Decomposition |
|  | df['Year'] = df['Date'].dt.year |
|  | df['Month'] = df['Date'].dt.month |
|  | df['Day'] = df['Date'].dt.day |
|  | df[['Year','Month','Day']].head() |

[view raw](https://gist.github.com/quantra-go-algo/ee83c1077463b213f9463d6933a5edce/raw/4a3b21357e1ec6f1122e943cd67e5d49850981bf/Decomposing%20date.py)[Decomposing date.py](https://gist.github.com/quantra-go-algo/ee83c1077463b213f9463d6933a5edce#file-decomposing-date-py)hosted with ❤ by [GitHub](https://github.com/)

Output:

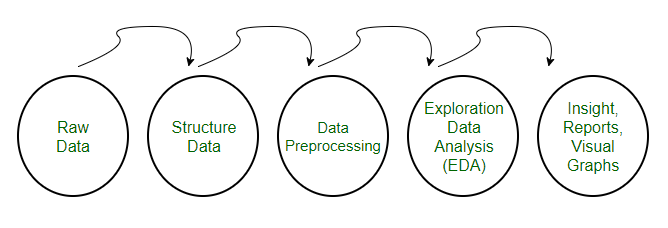
|  |  |  |
| --- | --- | --- |
| **Year** | **Month** | **Day** |
| 2019 | 1 | 5 |
| 2019 | 3 | 8 |
| 2019 | 3 | 3 |
| 2019 | 1 | 27 |
| 2019 | 2 | 8 |

In the output above, the data set is in date format which is numerical. And because of decomposing the date in different parts such as Year, Month and Day, the machine learning model will be able to learn the date format.

**Dataset-** <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

**Data Preprocessing**

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.



### ****Need of Data Preprocessing****

* For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
* Another aspect is that the data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithm are executed in one data set, and best out of them is chosen.

## Steps in Data ****Preprocessing****

**Step 1: Import the necessary libraries**

|  |
| --- |
| # importing libraries  import pandas as pd  import scipy  import numpy as np  from sklearn.preprocessing import MinMaxScaler  import seaborn as sns  import matplotlib.pyplot as plt |

#### Step 2: Load the dataset

Dataset link: [https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database]

|  |
| --- |
| # Load the dataset  df = pd.read\_csv('Geeksforgeeks/Data/diabetes.csv')  print(df.head()) |

**Output**:

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI

0 6 148 72 35 0 33.6 \

1 1 85 66 29 0 26.6

2 8 183 64 0 0 23.3

3 1 89 66 23 94 28.1

4 0 137 40 35 168 43.1

DiabetesPedigreeFunction Age Outcome

0 0.627 50 1

1 0.351 31 0

2 0.672 32 1

3 0.167 21 0

4 2.288 33 1

#### Check the data info

|  |
| --- |
| df.info() |

**Output:**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pregnancies 768 non-null int64

1 Glucose 768 non-null int64

2 BloodPressure 768 non-null int64

3 SkinThickness 768 non-null int64

4 Insulin 768 non-null int64

5 BMI 768 non-null float64

6 DiabetesPedigreeFunction 768 non-null float64

7 Age 768 non-null int64

8 Outcome 768 non-null int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB

As we can see from the above info that the our dataset has 9 columns and each columns has 768 values. There is no Null values in the dataset.

We can also check the null values using df.isnull()

|  |
| --- |
| df.isnull().sum() |

**Output**:

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

DiabetesPedigreeFunction 0

Age 0

Outcome 0

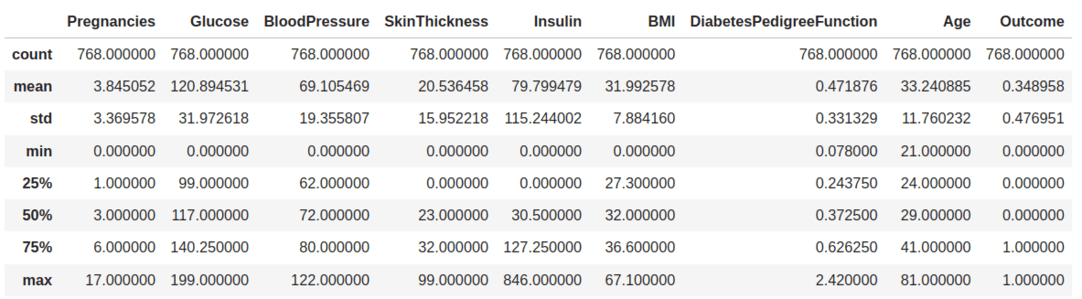
dtype: int64

#### Step 3: Statistical Analysis

In statistical analysis, first, we use the df.describe() which will give a descriptive overview of the dataset.

|  |
| --- |
| df.describe() |

**Output**:



*Data summary*

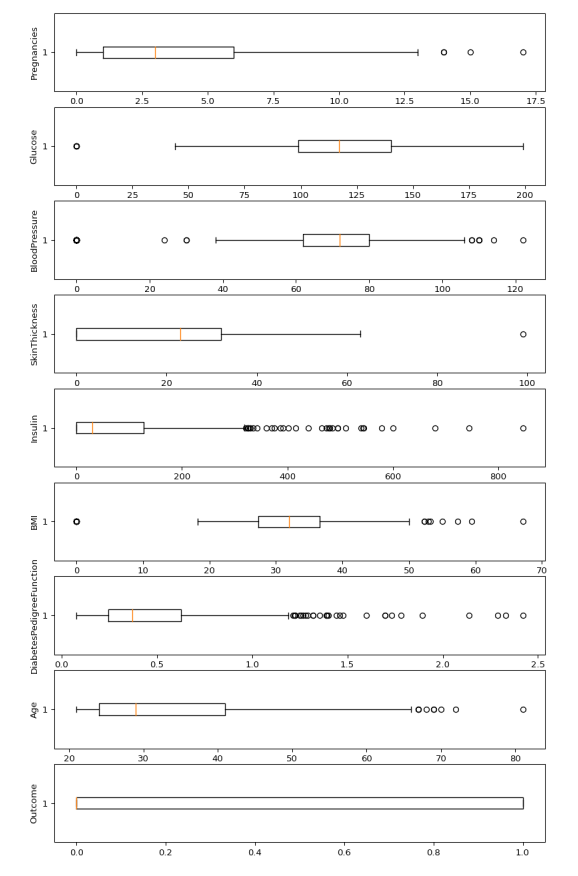
The above table shows the count, mean, standard deviation, min, 25%, 50%, 75%, and max values for each column. When we carefully observe the table we will find that. Insulin, Pregnancies, BMI, BloodPressure columns has outliers.

Let’s plot the boxplot for each column for easy understanding.

#### Step 4: Check the [outliers](https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/):

|  |
| --- |
| # Box Plots  fig, axs = plt.subplots(9,1,dpi=95, figsize=(7,17))  i = 0  for col in df.columns:      axs[i].boxplot(df[col], vert=False)      axs[i].set\_ylabel(col)      i+=1  plt.show() |

**Output**:



*Boxplots*

from the above boxplot, we can clearly see that all most every column has some amounts of outliers.

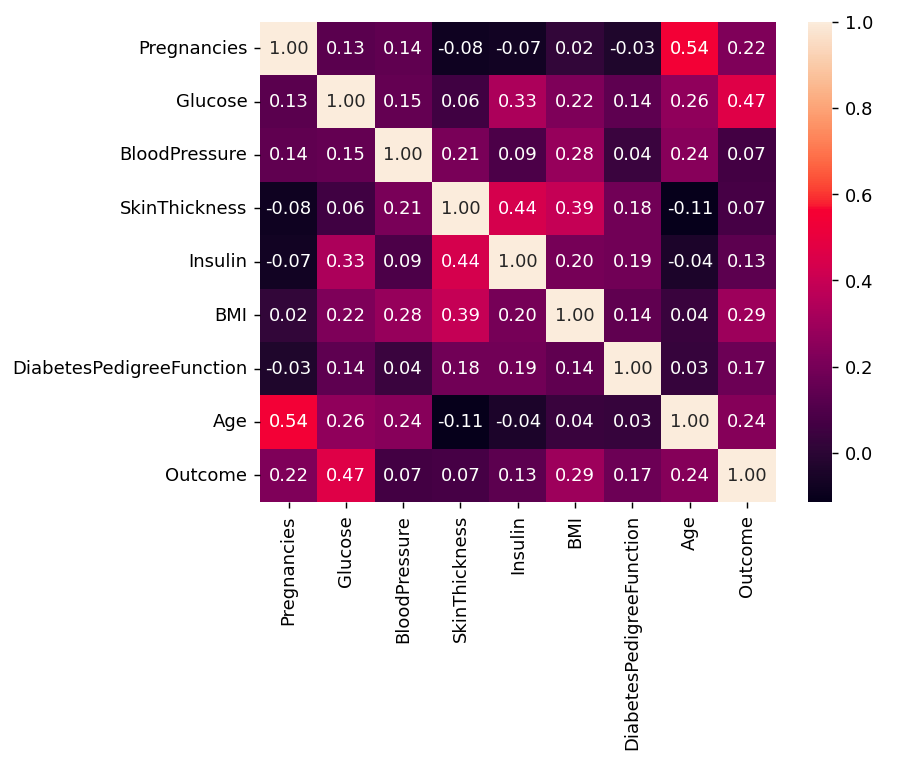
#### Drop the outliers

|  |
| --- |
| # Identify the quartiles  q1, q3 = np.percentile(df['Insulin'], [25, 75])  # Calculate the interquartile range  iqr = q3 - q1  # Calculate the lower and upper bounds  lower\_bound = q1 - (1.5 \* iqr)  upper\_bound = q3 + (1.5 \* iqr)  # Drop the outliers  clean\_data = df[(df['Insulin'] >= lower\_bound)                  & (df['Insulin'] <= upper\_bound)]      # Identify the quartiles  q1, q3 = np.percentile(clean\_data['Pregnancies'], [25, 75])  # Calculate the interquartile range  iqr = q3 - q1  # Calculate the lower and upper bounds  lower\_bound = q1 - (1.5 \* iqr)  upper\_bound = q3 + (1.5 \* iqr)  # Drop the outliers  clean\_data = clean\_data[(clean\_data['Pregnancies'] >= lower\_bound)                          & (clean\_data['Pregnancies'] <= upper\_bound)]      # Identify the quartiles  q1, q3 = np.percentile(clean\_data['Age'], [25, 75])  # Calculate the interquartile range  iqr = q3 - q1  # Calculate the lower and upper bounds  lower\_bound = q1 - (1.5 \* iqr)  upper\_bound = q3 + (1.5 \* iqr)  # Drop the outliers  clean\_data = clean\_data[(clean\_data['Age'] >= lower\_bound)                          & (clean\_data['Age'] <= upper\_bound)]      # Identify the quartiles  q1, q3 = np.percentile(clean\_data['Glucose'], [25, 75])  # Calculate the interquartile range  iqr = q3 - q1  # Calculate the lower and upper bounds  lower\_bound = q1 - (1.5 \* iqr)  upper\_bound = q3 + (1.5 \* iqr)  # Drop the outliers  clean\_data = clean\_data[(clean\_data['Glucose'] >= lower\_bound)                          & (clean\_data['Glucose'] <= upper\_bound)]      # Identify the quartiles  q1, q3 = np.percentile(clean\_data['BloodPressure'], [25, 75])  # Calculate the interquartile range  iqr = q3 - q1  # Calculate the lower and upper bounds  lower\_bound = q1 - (0.75 \* iqr)  upper\_bound = q3 + (0.75 \* iqr)  # Drop the outliers  clean\_data = clean\_data[(clean\_data['BloodPressure'] >= lower\_bound)                          & (clean\_data['BloodPressure'] <= upper\_bound)]      # Identify the quartiles  q1, q3 = np.percentile(clean\_data['BMI'], [25, 75])  # Calculate the interquartile range  iqr = q3 - q1  # Calculate the lower and upper bounds  lower\_bound = q1 - (1.5 \* iqr)  upper\_bound = q3 + (1.5 \* iqr)  # Drop the outliers  clean\_data = clean\_data[(clean\_data['BMI'] >= lower\_bound)                          & (clean\_data['BMI'] <= upper\_bound)]      # Identify the quartiles  q1, q3 = np.percentile(clean\_data['DiabetesPedigreeFunction'], [25, 75])  # Calculate the interquartile range  iqr = q3 - q1  # Calculate the lower and upper bounds  lower\_bound = q1 - (1.5 \* iqr)  upper\_bound = q3 + (1.5 \* iqr)    # Drop the outliers  clean\_data = clean\_data[(clean\_data['DiabetesPedigreeFunction'] >= lower\_bound)                          & (clean\_data['DiabetesPedigreeFunction'] <= upper\_bound)] |

#### Step 5: [Correlation](https://www.geeksforgeeks.org/exploring-correlation-in-python/)

|  |
| --- |
| #correlation  corr = df.corr()    plt.figure(dpi=130)  sns.heatmap(df.corr(), annot=True, fmt= '.2f')  plt.show() |

**Output**:



*Correlation*

We can also camapare by single columns in descending order

|  |
| --- |
| corr['Outcome'].sort\_values(ascending = False) |

**Output**:

Outcome 1.000000

Glucose 0.466581

BMI 0.292695

Age 0.238356

Pregnancies 0.221898

DiabetesPedigreeFunction 0.173844

Insulin 0.130548

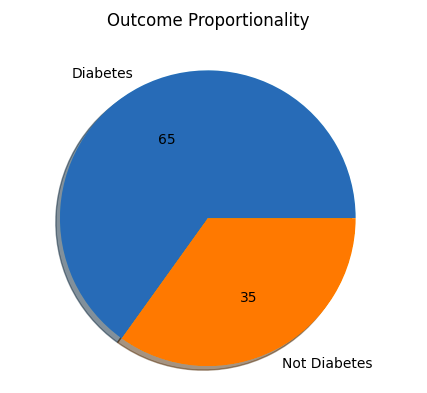
SkinThickness 0.074752

BloodPressure 0.0

#### Check Outcomes Proportionality

|  |
| --- |
| plt.pie(df.Outcome.value\_counts(),          labels= ['Diabetes', 'Not Diabetes'],          autopct='%.f', shadow=True)  plt.title('Outcome Proportionality')  plt.show() |

**Output**:



*Outcome Proportionality*

#### Step 6: Separate independent features and Target Variables

|  |
| --- |
| # separate array into input and output components  X = df.drop(columns =['Outcome'])  Y = df.Outcome |

#### Step 7: [Normalization or Standardization](https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/)

**Normalization**

* MinMaxScaler scales the data so that each feature is in the range [0, 1].
* It works well when the features have different scales and the algorithm being used is sensitive to the scale of the features, such as k-nearest neighbors or neural networks.
* Rescale your data using scikit-learn using the [MinMaxScaler](https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/).

|  |
| --- |
| # initialising the MinMaxScaler  scaler = MinMaxScaler(feature\_range=(0, 1))    # learning the statistical parameters for each of the data and transforming  rescaledX = scaler.fit\_transform(X)  rescaledX[:5] |

**Output**:

array([[0.353, 0.744, 0.59 , 0.354, 0. , 0.501, 0.234, 0.483],

[0.059, 0.427, 0.541, 0.293, 0. , 0.396, 0.117, 0.167],

[0.471, 0.92 , 0.525, 0. , 0. , 0.347, 0.254, 0.183],

[0.059, 0.447, 0.541, 0.232, 0.111, 0.419, 0.038, 0. ],

[0. , 0.688, 0.328, 0.354, 0.199, 0.642, 0.944, 0.2 ]])

#### Standardization

* Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.
* We can standardize data using scikit-learn with the [StandardScaler](https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/)class.
* It works well when the features have a normal distribution or when the algorithm being used is not sensitive to the scale of the features

|  |
| --- |
| from sklearn.preprocessing import StandardScaler    scaler = StandardScaler().fit(X)  rescaledX = scaler.transform(X)  rescaledX[:5] |

**Output**:

array([[ 0.64 , 0.848, 0.15 , 0.907, -0.693, 0.204, 0.468, 1.426],

[-0.845, -1.123, -0.161, 0.531, -0.693, -0.684, -0.365, -0.191],

[ 1.234, 1.944, -0.264, -1.288, -0.693, -1.103, 0.604, -0.106],

[-0.845, -0.998, -0.161, 0.155, 0.123, -0.494, -0.921, -1.042],

[-1.142, 0.504, -1.505, 0.907, 0.766, 1.41 , 5.485,

Why Do We Need Data Preprocessing?

Data Preprocessing is an important step in the machine learning algorithm. Imagine a situation where you are working on an assignment at your college, and the lecturer does not provide the raw headings and the idea of the topic. In this case, it will be very difficult for you to complete that assignment because raw data is not presented well to you. The same is the case in [Machine Learning](https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-machine-learning). Suppose the Data preprocessing step is missing while implementing the machine learning algorithm. In that case, it will definitely affect your work at the end, when it will be the final stage of applying the available data set to your algorithm.

While performing data preprocessing, it is important to ensure data accuracy so that it doesn't affect your machine learning algorithm at the final stage.

Steps in Data Preprocessing

There are six steps of data preprocessing in machine learning

Step 1: Import the Libraries

The foremost step of data preprocessing in machine learning includes importing some libraries. A library is basically a set of functions that can be called and used in the algorithm. There are many libraries available in different programming languages.

Step 2: Import the Loaded Data

The next important step is to load the data which has to be used in the machine learning algorithm. This is the most important machine learning preprocessing step. Collected data is to be imported for further assessment.

Once the data is loaded, checking for noisy or missing content is important.

Step 3: Check for Missing Values

Assess the loaded data and check for missing values. If missing values have been found, there are particularly two ways to resolve this issue:

* Either Remove the entire row that contains a missing value. However, removing the entire row can generate a possibility of losing some important data. This approach is useful if the dataset is very large
* Or Estimate the value by taking the mean, median or mode.

Step 4: Arrange the Data

Machine learning modules cannot understand non-numeric data. It is important to arrange the data in a numerical form in order to prevent any problems at later stages. Converting all text values into numerical form is the solution to this problem. You can use the LabelEncoder() function to do this.

Step 5: Do Scaling

Scaling is a technique that can convert data values into shorter ranges. Rescaling and Standardization can be used for scaling the data.

Step 6: Distribute Data into Training, Evaluation and Validation Sets

The final step is to distribute data in three different sets, namely

* Training
* Validation
* Evaluation

The training set is to train the data

The validation set is to validate the data

The evaluation set is to evaluate the data

Data Preprocessing Examples

An example to explain data preprocessing is explained using the table below. Appropriate data preprocessing techniques in machine learning will be applied to solve the problem.

|  |  |  |
| --- | --- | --- |
| Name | Age | Gender |
| John | 27 | Male |
| George | 26 | Female |
| Olivia | 25 | Male |
| Jack | 30 | Male |

Here in the table above, we can see that there are three variables, namely Name, Age and Gender. We can see that #2 and #3 have been assigned the wrong gender.

We can use data cleaning here to remove the inappropriate data rows, as we know that this data is already corrupt.

After data mining, the data table will look like:

|  |  |  |
| --- | --- | --- |
| Name | Age | Gender |
| John | 27 | Male |
| Jack | 30 | Male |

Else, we can do manual data transformation, which will make the table look like this:

|  |  |  |
| --- | --- | --- |
| Name | Age | Gender |
| John | 27 | Male |
| George | 26 | Male |
| Olivia | 25 | Female |
| Jack | 30 | Male |

Once the issue is fixed, the next step is to perform data reduction by descending the age.

|  |  |  |
| --- | --- | --- |
| Name | Age | Gender |
| Jack | 30 | Male |
| John | 27 | Male |
| George | 26 | Male |
| Olivia | 25 | Female |

Now, the issue is fixed, and the data set is complete and ready to be used for machine learning models and algorithms.

## Best Practices

The best practices for data preprocessing in machine learning include:

### Data Cleaning

Data cleaning is important to detect any missing values or noisy data that can corrupt the entire data set.

### Categorize the Data

It is important to categorize the data as machine learning algorithms can only handle numerical values. Categorizing the data will prevent problems at the later stages.

### Data Reduction

Reduce the data and arrange it in a way that simplifies the objective behind running and processing the data.

### Integrating

Integrate the data set and prepare the raw material for processing in the machine learning algorithm.