1. What are the key tasks that machine learning entails? What does data pre-processing imply?

1. Collecting data, 2. Preparing the Data, 3. Choosing a Model, 4. Training the Model, 5. Evaluating the Model, 6. Parameter Tuning, 7. Making Predictions

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

* Getting the dataset
* Importing libraries
* Importing datasets
* Finding Missing Data
* Encoding Categorical Data
* Splitting dataset into training and test set
* Feature scaling

1) Get the Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the dataset.

Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another dataset. To use the dataset in our code, we usually put it into a CSV file. However, sometimes, we may also need to use an HTML or xlsx file.

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

import numpy as nm

Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.

Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:

import matplotlib.pyplot as mpt

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:

Import pandas as pd

3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. To set a working directory in Spyder IDE, we need to follow the below steps:

Save your Python file in the directory which contains dataset.

Go to File explorer option in Spyder IDE, and select the required directory.

Click on F5 button or run option to execute the file.

read\_csv() function:

Now to import the dataset, we will use read\_csv() function of pandas library, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL.

We can use read\_csv function as below:

data\_set= pd.read\_csv('Dataset.csv')

Here, data\_set is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset. Once we execute the above line of code, it will successfully import the dataset in our code. We can also check the imported dataset by clicking on the section variable explorer, and then double click on data\_set.

Extracting dependent and independent variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are three independent variables that are Country, Age, and Salary, and one is a dependent variable which is Purchased.

Extracting independent variable:

To extract an independent variable, we will use iloc[ ] method of Pandas library. It is used to extract the required rows and columns from the dataset.

x= data\_set.iloc[:,:-1].values

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable.

Extracting dependent variable:

To extract dependent variables, again, we will use Pandas .iloc[] method.

y= data\_set.iloc[:,3].values

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

4) Handling Missing data:

The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.

Ways to handle missing data:

There are mainly two ways to handle missing data, which are:

By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.

To handle missing values, we will use Scikit-learn library in our code, which contains various libraries for building machine learning models. Here we will use Imputer class of sklearn.preprocessing library. Below is the code for it:

#handling missing data (Replacing missing data with the mean value)

from sklearn.preprocessing import Imputer

imputer= Imputer(missing\_values ='NaN', strategy='mean', axis = 0)

#Fitting imputer object to the independent variables x.

imputerimputer= imputer.fit(x[:, 1:3])

#Replacing missing data with the calculated mean value

x[:, 1:3]= imputer.transform(x[:, 1:3])

5) Encoding Categorical data:

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, Country, and Purchased.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

For Country variable:

Firstly, we will convert the country variables into categorical data. So to do this, we will use LabelEncoder() class from preprocessing library.

#Catgorical data

#for Country Variable

from sklearn.preprocessing import LabelEncoder

label\_encoder\_x= LabelEncoder()

x[:, 0]= label\_encoder\_x.fit\_transform(x[:, 0])

Dummy Variables:

Dummy variables are those variables which have values 0 or 1. The 1 value gives the presence of that variable in a particular column, and rest variables become 0. With dummy encoding, we will have a number of columns equal to the number of categories.

In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use OneHotEncoder class of preprocessing library.

#for Country Variable

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

label\_encoder\_x= LabelEncoder()

x[:, 0]= label\_encoder\_x.fit\_transform(x[:, 0])

#Encoding for dummy variables

onehot\_encoder= OneHotEncoder(categorical\_features= [0])

x= onehot\_encoder.fit\_transform(x).toarray()

6) Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset.

Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, random\_state=0)

Explanation:

In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.

In the second line, we have used four variables for our output that are

x\_train: features for the training data

x\_test: features for testing data

y\_train: Dependent variables for training data

y\_test: Independent variable for testing data

In train\_test\_split() function, we have passed four parameters in which first two are for arrays of data, and test\_size is for specifying the size of the test set. The test\_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.

The last parameter random\_state is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

7) Feature Scaling

Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

For feature scaling, we will import StandardScaler class of sklearn.preprocessing library as:

from sklearn.preprocessing import StandardScaler

Now, we will create the object of StandardScaler class for independent variables or features. And then we will fit and transform the training dataset.

st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train)

For test dataset, we will directly apply transform() function instead of fit\_transform() because it is already done in training set.

x\_test= st\_x.transform(x\_test)

**Code:**

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data\_set= pd.read\_csv('Dataset.csv')

#Extracting Independent Variable

x= data\_set.iloc[:, :-1].values

#Extracting Dependent variable

y= data\_set.iloc[:, 3].values

#handling missing data(Replacing missing data with the mean value)

from sklearn.preprocessing import Imputer

imputer= Imputer(missing\_values ='NaN', strategy='mean', axis = 0)

#Fitting imputer object to the independent varibles x.

imputerimputer= imputer.fit(x[:, 1:3])

#Replacing missing data with the calculated mean value

x[:, 1:3]= imputer.transform(x[:, 1:3])

#for Country Variable

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

label\_encoder\_x= LabelEncoder()

x[:, 0]= label\_encoder\_x.fit\_transform(x[:, 0])

#Encoding for dummy variables

onehot\_encoder= OneHotEncoder(categorical\_features= [0])

x= onehot\_encoder.fit\_transform(x).toarray()

#encoding for purchased variable

labelencoder\_y= LabelEncoder()

y= labelencoder\_y.fit\_transform(y)

# Splitting the dataset into training and test set.

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, random\_state=0)

#Feature Scaling of datasets

from sklearn.preprocessing import StandardScaler

st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train)

x\_test= st\_x.transform(x\_test)

2. Describe quantitative and qualitative data in depth. Make a distinction between the two.

Quantitative data refers to any information that can be quantified, counted or measured, and given a numerical value. Qualitative data is descriptive in nature, expressed in terms of language rather than numerical values. Quantitative research is based on numeric data.

Quantitative = Quantity

Quantitative data are

measures of values or counts and are expressed as numbers.

data about numeric variables (e.g. how many, how much or how often).

Qualitative = Quality

Qualitative data are

measures of 'types' and may be represented by a name, symbol, or a number code.

Qualitative data are data about categorical variables (e.g. what type).

Data collected about a numeric variable will always be quantitative and data collected about a categorical variable will always be qualitative. Therefore, you can identify the type of data, prior to collection, based on whether the variable is numeric or categorical.

Importance of quantitative and qualitative data

Quantitative and qualitative data provide different outcomes, and are often used together to get a full picture of a population. For example, if data are collected on annual income (quantitative), occupation data (qualitative) could also be gathered to get more detail on the average annual income for each type of occupation.

Quantitative and qualitative data can be gathered from the same data unit depending on whether the variable of interest is numerical or categorical.

3. Create a basic data collection that includes some sample records. Have at least one attribute from each of the machine learning data types.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CustomerID | Genre | Age | Annual Income (k$) | Spending Score (1-100) |
| 1 | Male | 19 | 15 | 39 |
| 2 | Male | 21 | 15 | 81 |
| 3 | Female | 20 | 16 | 6 |
| 4 | Female | 23 | 16 | 77 |
| 5 | Female | 31 | 17 | 40 |
| 6 | Female | 22 | 17 | 76 |
| 7 | Female | 35 | 18 | 6 |
| 8 | Female | 23 | 18 | 94 |
| 9 | Male | 64 | 19 | 3 |
| 10 | Female | 30 | 19 | 72 |

4. What are the various causes of machine learning data issues? What are the ramifications?

1. Duplicate data · 2. Inaccurate data · 3. Ambiguous data · 4. Hidden data · 5. Inconsistent data · 6. Too much data · 7. Data Downtime.

Noisy data, incomplete data, inaccurate data, and unclean data lead to less accuracy in classification and low-quality results. Hence, data quality can also be considered as a major common problem while processing machine learning algorithms.

5. Demonstrate various approaches to categorical data exploration with appropriate examples.

Categorical Variable/Data (or Nominal variable): Such variables take on a fixed and limited number of possible values. For example – grades, gender, blood group type, etc. Also, in the case of categorical variables, the logical order is not the same as categorical data e.g. “one”, “two”, “three”. But the sorting of these variables uses logical order. For example, gender is a categorical variable and has categories – male and female and there is no intrinsic ordering to the categories. A purely categorical variable is one that simply allows you to assign categories, but you cannot clearly order the variables. Terms related to Variability Metrics :

Mode : Most frequently occurring value in the given data Example-

Data = ["Car", "Bat", "Bat", "Car", "Bat", "Bat", "Bat", "Bike"]

Mode = "Bat"

Expected Value : When working in machine learning, categories have to be associated with a numeric value, so as to give understanding to the machine. This gives an average value based on a category’s probability of occurrence i.e. Expected Value. It is calculated by –

-> Multiply each outcome by its probability of occurring.

-> Sum these values

So, it is the sum of values times their probability of occurrence often used to sum up factor variable levels.

Bar Charts : Frequency of each category plotted as bars. Loading Libraries –

import matplotlib.pyplot as plt

import numpy as np

Data –

label = ['Car', 'Bike', 'Truck', 'Cycle', 'Jeeps', 'Ambulance']

no\_vehicle = [941, 854, 4595, 2125, 942, 509]

Indexing Data –

index = np.arange(len(label))

print ("Total Labels : ", len(label))

print ("Indexing : ", index)

Output:

Total Labels : 6

Indexing : [0 1 2 3 4 5]

Bar Graph –

plt.bar(index, no\_vehicle)

plt.xlabel('Type', fontsize = 15)

plt.ylabel('No of Vehicles', fontsize = 15)

plt.xticks(index, label, fontsize = 10, rotation = 30)

plt.title('Market Share for Each Genre 1995-2017')

plt.show()

Pie Charts : Frequency of each category plotted as pie or wedges. It is a circular graph, where the arc length of each slice is proportional to the quantity it represents.

plt.figure(figsize =(8, 8))

plt.pie(no\_vehicle, labels = label,

startangle = 90, autopct ='%.1f %%')

plt.show()

6. How would the learning activity be affected if certain variables have missing values? Having said that, what can be done about it?

All the machine learning algorithms don't support missing values but some ML algorithms are robust to missing values in the dataset. The k-NN algorithm can ignore a column from a distance measure when a value is missing. Naive Bayes can also support missing values when making a prediction.

Many machine learning algorithms fail if the dataset contains missing values. However, algorithms like K-nearest and Naive Bayes support data with missing values. You may end up building a biased machine learning model which will lead to incorrect results if the missing values are not handled properly

One way of handling missing values is the deletion of the rows or columns having null values. If any columns have more than half of the values as null then you can drop the entire column. In the same way, rows can also be dropped if having one or more columns values as null.

When dealing with missing data, data scientists can use two primary methods to solve the error: imputation or the removal of data. The imputation method develops reasonable guesses for missing data. It's most useful when the percentage of missing data is low.

7. Describe the various methods for dealing with missing data values in depth.

Analyze each column with missing values carefully to understand the reasons behind the missing of those values, as this information is crucial to choose the strategy for handling the missing values.

There are 2 primary ways of handling missing values:

Deleting the Missing values

Imputing the Missing Values

Deleting the Missing value

Generally, this approach is not recommended. It is one of the quick and dirty techniques one can use to deal with missing values. If the missing value is of the type Missing Not At Random (MNAR), then it should not be deleted.

If the missing value is of type Missing At Random (MAR) or Missing Completely At Random (MCAR) then it can be deleted (In the analysis, all cases with available data are utilized, while missing observations are assumed to be completely random (MCAR) and addressed through pairwise deletion.)

The disadvantage of this method is one might end up deleting some useful data from the dataset.

There are 2 ways one can delete the missing data values:

Deleting the entire row (listwise deletion)

If a row has many missing values, you can drop the entire row. If every row has some (column) value missing, you might end up deleting the whole data. The code to drop the entire row is as follows:

IN:

df = train\_df.dropna(axis=0)

df.isnull().sum()

OUT:

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

Deleting the entire column

If a certain column has many missing values, then you can choose to drop the entire column. The code to drop the entire column is as follows:

IN:

df = train\_df.drop(['Dependents'],axis=1)

df.isnull().sum()

OUT:

Loan\_ID 0

Gender 13

Married 3

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

Imputing the Missing Value

There are many imputation methods for replacing the missing values. You can use different python libraries such as Pandas, and Sci-kit Learn to do this. Let’s go through some of the ways of replacing the missing values.

Replacing with an arbitrary value

If you can make an educated guess about the missing value, then you can replace it with some arbitrary value using the following code. E.g., in the following code, we are replacing the missing values of the ‘Dependents’ column with ‘0’.

IN:

#Replace the missing value with '0' using 'fiilna' method

train\_df['Dependents'] = train\_df['Dependents'].fillna(0)

train\_df[‘Dependents'].isnull().sum()

OUT:

0

Replacing with the mean

This is the most common method of imputing missing values of numeric columns. If there are outliers, then the mean will not be appropriate. In such cases, outliers need to be treated first. You can use the ‘fillna’ method for imputing the columns ‘LoanAmount’ and ‘Credit\_History’ with the mean of the respective column values.

IN:

#Replace the missing values for numerical columns with mean

train\_df['LoanAmount'] = train\_df['LoanAmount'].fillna(train\_df['LoanAmount'].mean())

train\_df['Credit\_History'] = train\_df[‘Credit\_History'].fillna(train\_df['Credit\_History'].mean())

OUT:

Loan\_ID 0

Gender 13

Married 3

Dependents 15

Education 0

Self\_Employed 32

ApplicantIncome 0

CoapplicantIncome 0

LoanAmount 0

Loan\_Amount\_Term 0

Credit\_History 0

Property\_Area 0

Loan\_Status 0

dtype: int64

Replacing with the 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:

#Replace the missing values for categorical columns with mode

train\_df['Gender'] = train\_df['Gender'].fillna(train\_df['Gender'].mode()[0])

train\_df['Married'] = train\_df['Married'].fillna(train\_df['Married'].mode()[0])

train\_df['Self\_Employed'] = train\_df[‘Self\_Employed'].fillna(train\_df['Self\_Employed'].mode()[0])

train\_df.isnull().sum()

OUT:

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

Replacing with the median

The median is the middlemost value. It’s better to use the median value for imputation in the case of outliers. You can use the ‘fillna’ method for imputing the column ‘Loan\_Amount\_Term’ with the median value.

train\_df['Loan\_Amount\_Term']= train\_df['Loan\_Amount\_Term'].fillna(train\_df['Loan\_Amount\_Term'].median())

Replacing with the previous value – forward fill

In some cases, imputing the values with the previous value instead of the mean, mode, or median is more appropriate. This is called forward fill. It is mostly used in time series data. You can use the ‘fillna’ function with the parameter ‘method = ffill’

IN:

import pandas as pd

import numpy as np

test = pd.Series(range(6))

test.loc[2:4] = np.nan

test

OUT:

0 0.0

1 1.0

2 Nan

3 Nan

4 Nan

5 5.0

dtype: float64

IN:

# Forward-Fill

test.fillna(method=‘ffill')

OUT:

0 0.0

1 1.0

2 1.0

3 1.0

4 1.0

5 5.0

dtype: float64

Replacing with the next value – backward fill

In backward fill, the missing value is imputed using the next value.

IN:

# Backward-Fill

test.fillna(method=‘bfill')

OUT:

0 0.0

1 1.0

2 5.0

3 5.0

4 5.0

5 5.0

dtype: float64

Interpolation

Missing values can also be imputed using interpolation. Pandas’ interpolate method can be used to replace the missing values with different interpolation methods like ‘polynomial,’ ‘linear,’ and ‘quadratic.’ The default method is ‘linear.’

IN:

test.interpolate()

OUT:

0 0.0

1 1.0

2 2.0

3 3.0

4 4.0

5 5.0

dtype: float64

How to Impute Missing Values for Categorical Features?

There are two ways to impute missing values for categorical features as follows:

Impute the Most Frequent Value

We will use ‘SimpleImputer’ in this case, and as this is a non-numeric column, we can’t use mean or median, but we can use the most frequent value and constant.

IN:

import pandas as pd

import numpy as np

X = pd.DataFrame({'Shape':['square', 'square', 'oval', 'circle', np.nan]})

X

Shape

OUT:

0 square

1 square

2 oval

3 circle

4 NaN

IN:

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='most\_frequent')

imputer.fit\_transform(X)

OUT:

array([['square'],

['square'],

['oval'],

['circle'],

['square']], dtype=object)

As you can see, the missing value is imputed with the most frequent value, ’square.’

Impute the Value “Missing”

We can impute the value “missing,” which treats it as a separate category.

IN:

imputer = SimpleImputer(strategy='constant', fill\_value='missing')

imputer.fit\_transform(X)

OUT:

array([['square'],

['square'],

['oval'],

['circle'],

['missing']], dtype=object)

In any of the above approaches, you will still need to OneHotEncode the data (or you can also use another encoder of your choice). After One Hot Encoding, in case 1, instead of the values ‘square,’ ‘oval,’ and’ circle,’ you will get three feature columns. And in case 2, you will get four feature columns (4th one for the ‘missing’ category). So it’s like adding the missing indicator column in the data. There is another way to add a missing indicator column, which we will discuss further.

How to Impute Missing Values Using Sci-kit Learn Library?

We can impute missing values using the sci-kit library by creating a model to predict the observed value of a variable based on another variable which is known as regression imputation.

Univariate Approach

In a Univariate approach, only a single feature is taken into consideration. You can use the class SimpleImputer and replace the missing values with mean, mode, median, or some constant value.

Let’s see an example:

IN:

import numpy as np

from sklearn.impute import SimpleImputer

imp = SimpleImputer(missing\_values=np.nan, strategy='mean')

imp.fit([[1, 2], [np.nan, 3], [7, 6]])

OUT: SimpleImputer()

IN:

X = [[np.nan, 2], [6, np.nan], [7, 6]]

print(imp.transform(X))

OUT:

[[4. 2. ]

[6. 3.666...]

[7. 6. ]]

Multivariate Approach

In a multivariate approach, more than one feature is taken into consideration. There are two ways to impute missing values considering the multivariate approach. Using KNNImputer or IterativeImputer classes.

Let’s take an example of a titanic dataset.

Suppose the feature ‘age’ is well correlated with the feature ‘Fare’ such that people with lower fares are also younger and people with higher fares are also older. In that case, it would make sense to impute low age for low fare values and high age for high fare values. So here, we are taking multiple features into account by following a multivariate approach.

IN:

import pandas as pd

df = pd.read\_csv('http://bit.ly/kaggletrain', nrows=6)

cols = ['SibSp', 'Fare', 'Age']

X = df[cols]

X

IN:

from sklearn.experimental import enable\_iterative\_imputer

from sklearn.impute import IterativeImputer

impute\_it = IterativeImputer()

impute\_it.fit\_transform(X)

Let’s see how IterativeImputer works. For all rows in which ‘Age’ is not missing, sci-kit learn runs a regression model. It uses ‘Sib sp’ and ‘Fare’ as the features and ‘Age’ as the target. And then, for all rows for which ‘Age’ is missing, it makes predictions for ‘Age’ by passing ‘Sib sp’ and ‘Fare’ to the training model. So it actually builds a regression model with two features and one target and then makes predictions on any places where there are missing values. And those predictions are the imputed values.

Nearest Neighbors Imputations (KNNImputer)

Missing values are imputed using the k-Nearest Neighbors approach, where a Euclidean distance is used to find the nearest neighbors. Let’s take the above example of the titanic dataset to see how it works.

IN:

from sklearn.impute import KNNImputer

impute\_knn = KNNImputer(n\_neighbors=2)

impute\_knn.fit\_transform(X)

OUT:

array([[ 1. , 7.25 , 22. ],

[ 1. , 71.2833, 38. ],

[ 0. , 7.925 , 26. ],

[ 1. , 53.1 , 35. ],

[ 0. , 8.05 , 35. ],

[ 0. , 8.4583, 30.5 ]])

In the above example, the n\_neighbors=2. So sci-kit learn finds the two most similar rows measured by how close the ‘Sib sp’ and ‘Fare’ values are to the row which has missing values. In this case, the last row has a missing value. And the third row and the fifth row have the closest values for the other two features. So the average of the ‘Age’ feature from these two rows is taken as the imputed value.

How to Use “Missingness” as a Feature?

In some cases, while imputing missing values, you can preserve information about which values were missing and use that as a feature. This is because sometimes, there may be a relationship between the reason for missing values (also called the “missingness”) and the target variable you are trying to predict. In such cases, you can add a missing indicator to encode the “missingness” as a feature in the imputed data set.

Where can we use this?

Suppose you are predicting the presence of a disease. Now, imagine a scenario where a missing age is a good predictor of the disease because we don’t have records for people in poverty. The age values are not missing at random. They are missing for people in poverty, and poverty is a good predictor of disease. Thus, missing age or “missingness” is a good predictor of disease.

IN:

import pandas as pd

import numpy as np

X = pd.DataFrame({'Age':[20, 30, 10, np.nan, 10]})

X

IN:

from sklearn.impute

import SimpleImputer

# impute the mean

imputer = SimpleImputer()

imputer.fit\_transform(X)

IN:

imputer = SimpleImputer(add\_indicator=True)

imputer.fit\_transform(X)

In the above example, the second column indicates whether the corresponding value in the first column was missing or not. ‘1’ indicates that the corresponding value was missing, and ‘0’ indicates that the corresponding value was not missing.

If you don’t want to impute missing values but only want to have the indicator matrix, then you can use the ‘MissingIndicator’ class from scikit learn.

8. What are the various data pre-processing techniques? Explain dimensionality reduction and function selection in a few words.

While both methods are used for reducing the number of features in a dataset, there is an important difference. Feature selection is simply selecting and excluding given features without changing them. Dimensionality reduction transforms features into a lower dimension.

9.

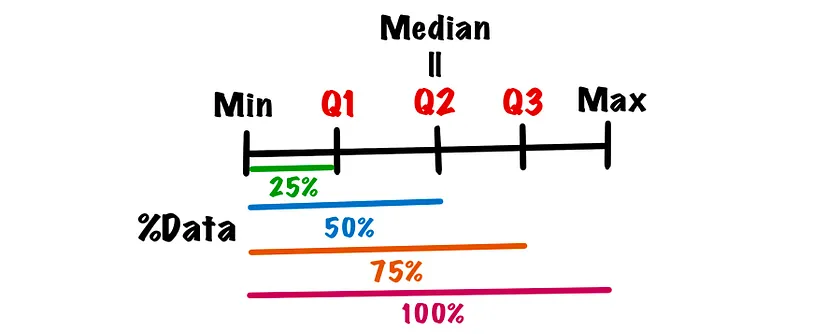
i. What is the IQR? What criteria are used to assess it?

Outliers are data points that lie outside the overall pattern in a distribution. Thus, a data point that is distant from the remaining data points in the sample is NOT necessarily an outlier. Instead, a data point deviating from the model fit (the pattern of underlying population) is an outlier.

In this article, we will discuss how to find an outlier using IQR method and box plot in 1-dimensional data.

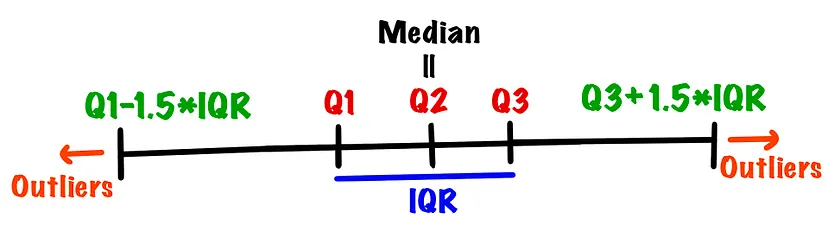
IQR method

One common technique to detect outliers is using IQR (interquartile range). In specific, IQR is the middle 50% of data, which is Q3-Q1. Q1 is the first quartile, Q3 is the third quartile, and quartile divides an ordered dataset into 4 equal-sized groups. In Python, we can use percentile function in NumPy package to find Q1 and Q3.



Quartile demonstration

The interquartile range method defines outliers as values larger than Q3 + 1.5 \* IQR or the values smaller than Q1 – 1.5 \* IQR.



Outlier demonstration

Say we have collected the midterm grade of 500 students and stored the data in an array called grades. We want to know if there are students getting extremely high or extremely low score. In other words, we want to find the outliers in terms of midterm grade.

First, we use percentile function to find Q1 and Q3. The first argument is the data, and the second argument is the percentiles to compute. We can either pass 1 percentile at a time (method 1) or store the percentiles we want to get in a list (method 2).

# method 1.

Q1 = np.percentile(grades , 25)

Q3 = np.percentile(grades , 75)

# method 2.

Q1,Q3 = np.percentile(grades , [25,75])

Then, we can compute IQR and the limit of non-outliers. To recap, IQR is Q3-Q1, and the upper limit for non-outliers is Q3+1.5\*IQR, the lower limit for non-outliers is Q1-1.5\*IQR.

IQR = Q3 - Q1

ul = Q3+1.5\*IQR

ll = Q1-1.5\*IQR

In this example, ul (upper limit) is 99.5, ll (lower limit) is 7.5. Thus, the grades above 99.5 or below 7.5 are considered as outliers. We can use indexing to find the exact outliers.

outliers = grades[(grades > ul) | (grades < ll)]

outliers

This returns array([ 0, 7, 4, 3, 0, 4, 2, 7, 6, 100, 1, 3, 0,

3, 100, 100, 100, 100, 4, 0, 3, 6, 6, 6, 100, 7,

6, 100, 100, 6, 3, 6, 1, 6, 0]). As we expected, the values above 99.5 or below 7.5 are outliers.

Box plot

Box plot visually shows the 5 statistical summary of a dataset, Min, Q1, Q2 (median), Q3, and Max. Further, it denotes the outliers according to the IQR method. One thing to note is that when an outlier is detected, the whisker will correspondingly change to the upper limit (Q3+1.5\*IQR) or lower limit (Q1–1.5\*IQR).

To get a concrete idea, let’s look at the box plots based on our data and the hypothetical data, which has NO outliers.

fig = plt.figure(figsize=(6,5))

hypo = np.random.randint(20, 81, size=500)

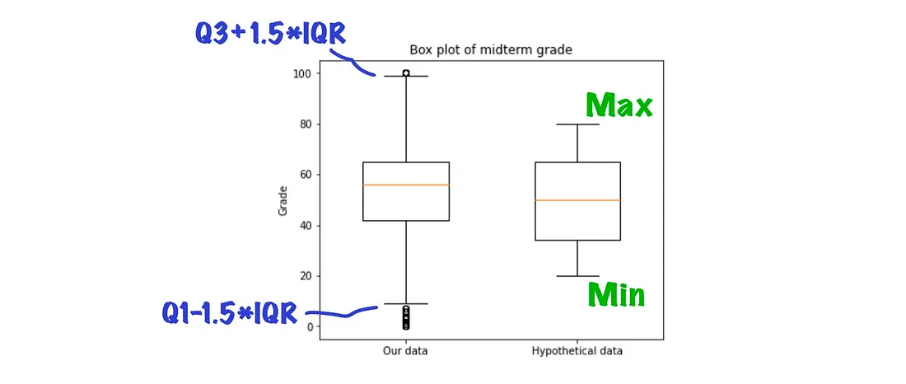
plt.boxplot([grades, hypo], widths=0.5)

plt.xticks([1,2],['Our data', 'Hypothetical data'])

plt.ylabel('Grade')

plt.title('Box plot of midterm grade')

plt.show()



Box plot demonstration

For the box plot on the left, there are dots on both the top and the bottom of the box. These dots are exactly the outliers we calculated before. Since there are outliers on both direction, the upper whisker changes from Max to Q3+1.5\*IQR, the bottom whisker changes from Min to Q1–1.5\*IQR.

For the box plot on the right, there is NO dot hence NO outliers. In this case, we don’t need to spend effort doing further analysis or finding which points are outliers. Therefore, box plot is a good choice for initial investigation of outliers.

⚡ For the box plot on the left, it seems like there is only 1 outlier data point on the top. However, if we go back to the previous section, we can see a lot of data points have value 100 and considered outliers. In fact, these outliers overlap with each other and can NOT be clearly displayed on the box plot. This is one of the drawbacks of box plot. Thus, a further analysis is always required to avoid misunderstanding of the data set.

💛💛💛 If you like this article, make sure to follow me! It really encourages me and motivates me to keep sharing. Thank you so much.

Coding

# import packages

import numpy as np

import matplotlib.pyplot as plt

# Create sample data

np.random.seed(102)

grades = np.concatenate([[50,52,53,55,56,60,61,62,65,67]\*20, np.random.randint(0, 101, size=300)])

# IQR

# Find Q1, Q3

# 1.

Q1 = np.percentile(grades , 25)

Q3 = np.percentile(grades , 75)

# 2.

Q1,Q3 = np.percentile(grades , [25,75])

# Find IQR, upper limit, lower limit

IQR = Q3 - Q1

ul = Q3+1.5\*IQR

ll = Q1-1.5\*IQR

# Find outliers

outliers = grades[(grades > ul) | (grades < ll)]

print(outliers)

# Box plot

fig = plt.figure(figsize=(6,5))

hypo = np.random.randint(20, 81, size=500)

plt.boxplot([grades, hypo], widths=0.5)

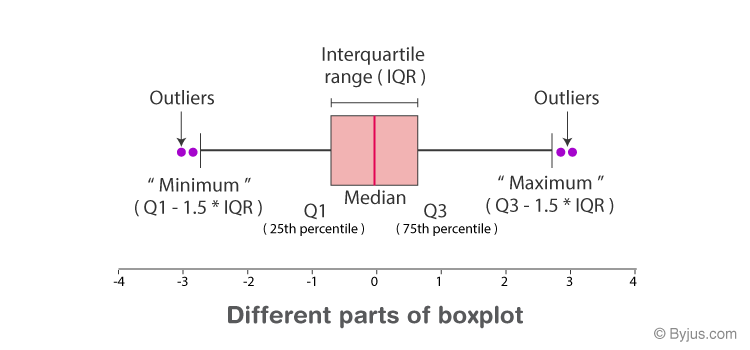
plt.xticks([1,2],['Our data', 'Hypothetical data'])

plt.ylabel('Grade')

plt.title('Box plot of midterm grade')

plt.show()

ii. Describe the various components of a box plot in detail? When will the lower whisker surpass the upper whisker in length? How can box plots be used to identify outliers?



Minimum: The minimum value in the given dataset

First Quartile (Q1): The first quartile is the median of the lower half of the data set.

Median: The median is the middle value of the dataset, which divides the given dataset into two equal parts. The median is considered as the second quartile.

Third Quartile (Q3): The third quartile is the median of the upper half of the data.

Maximum: The maximum value in the given dataset.

Apart from these five terms, the other terms used in the box plot are:

Interquartile Range (IQR): The difference between the third quartile and first quartile is known as the interquartile range. (i.e.) IQR = Q3-Q1

Outlier: The data that falls on the far left or right side of the ordered data is tested to be the outliers. Generally, the outliers fall more than the specified distance from the first and third quartile.

(i.e.) Outliers are greater than Q3+(1.5 . IQR) or less than Q1-(1.5 . IQR).

10. Make brief notes on any two of the following:

1. Data collected at regular intervals

Interval data, also called an integer, is defined as a data type which is measured along a scale, in which each point is placed at equal distance from one another. Interval data always appears in the form of numbers or numerical values where the distance between the two points is standardized and equal.

Interval data cannot be multiplied or divided, however, it can be added or subtracted. Interval data is measured on an interval scale. A simple example of interval data: The difference between 100 degrees Fahrenheit and 90 degrees Fahrenheit is the same as 60 degrees Fahrenheit and 70 degrees Fahrenheit.

In market research or in any other forms of social, economic or business research interval data plays a pivotal role. What makes interval data so popular and in-demand is because interval data supports almost all statistical test and transformations in obtaining quantitative data.

Interval data has very distinctive attributes that make it distinct in comparison to nominal data, ordinal data or even ratio data. Interval data doesn’t have a defined absolute zero point which is present in ratio data. The lack of absolute point zero makes comparisons of direct magnitudes impossible. For example, Object A is twice as large as Object B is not a possibility in interval data.

2. The gap between the quartiles

The interquartile range or IQR is the range of the middle half of a set of data. It is the difference between the upper quartile and the lower quartile.

3. Use a cross-tab

A crosstab is a table showing the relationship between two or more variables. Where the table only shows the relationship between two categorical variables, a crosstab is also known as a contingency table.

1. Make a comparison between:

1. Data with nominal and ordinal values

Nominal data is classified without a natural order or rank, whereas ordinal data has a predetermined or natural order. On the other hand, numerical or quantitative data will always be a number that can be measured.

2. Histogram and box plot

Box plot - gives the quartiles and indicate the median data to compare easily

Histogram - gives only the count

3. The average and median

The average is calculated by adding up all of the individual values and dividing this total by the number of observations. The median is calculated by taking the “middle” value, the value for which half of the observations are larger and half are smaller.