1. **Handling Categorical Data**

import pandas as pd

#from patsy import dmatrices

df = pd.DataFrame({'A': ['high', 'medium', 'low'], 'B': [10,20,30]}, index=[0, 1, 2])

print(df)

**Result:**

A B

0 high 10

1 medium 20

2 low 30

1. **Rescaling a Feature [using MinMaxScaler()]**

importnumpy as np

fromsklearn import preprocessing

feature = np.array([[-500.5],[-100.1], [0],[100.1], [900.9]])

minmax\_scale = preprocessing.MinMaxScaler(feature\_range=(0,1))

scaled\_feature = minmax\_scale.fit\_transform(feature)

scaled\_feature

**Result:**

array([[0. ],

[0.28571429],

[0.35714286],

[0.42857143],

[1. ]])

1. **Rescaling a Feature [using StandardScaler()]**

importnumpy as np

fromsklearn import preprocessing

feature = np.array([[-1000.1],[-200.2], [500.5],[600.6], [9000.9]])

scaler = preprocessing.StandardScaler()

standardized = scaler.fit\_transform(feature)

print(standardized)

print("Mean: ", round(standardized.mean()))

print("Standard Deviation: ", round(standardized.std()))

print("Variance: ", round(standardized.var()))

**Result:**

[[-0.76058269] Mean: 0

[-0.54177196] Standard Deviation: 1

[-0.35009716] Variance: 1

[-0.32271504]

[ 1.97516685]]

1. **Normalizing Data [using Normalizer(norm=”l2”)]**

importnumpy as np

fromsklearn.preprocessing import Normalizer

features = np.array([[0.5,0.5],[1.1,3.4], [1.5,20.2],[1.63,34.4], [10.9,3.3]])

normalizer = Normalizer(norm="l2")

normalizer.transform(features)

**Result:**

array([[0.70710678, 0.70710678],

[0.30782029, 0.95144452],

[0.07405353, 0.99725427],

[0.04733062, 0.99887928],

[0.95709822, 0.28976368]])

1. **Normalizing Data [using Normalizer(norm=”l1”)]**

importnumpy as np

fromsklearn.preprocessing import Normalizer

features = np.array([[0.5,0.5],[1.1,3.4], [1.5,20.2],[1.63,34.4], [10.9,3.3]])

normalizer\_1 = Normalizer(norm="l1").transform(features)

print(normalizer\_1)

print("Sum of the first observation\'s values: ", normalizer\_1[0,0] + normalizer\_1[0,1])

**Result:**

[[0.5 0.5 ]

[0.24444444 0.75555556]

[0.06912442 0.93087558]

[0.04524008 0.95475992]

[0.76760563 0.23239437]]

Sum of the first observation's values: 1.0

1. **Deleting Observations with Missing Values (using NumPy)**

importnumpy as np

# Create feature matrix

features = np.array([[1.1, 11.1],

[2.2, 22.2],

[3.3, 33.3],

[4.4, 44.4],

[np.nan, 55]])

# Keep only observations that are not (denoted by ~) missing

features[~np.isnan(features).any(axis=1)]

**Result:**

array([[ 1.1, 11.1],

[ 2.2, 22.2],

[ 3.3, 33.3],

[ 4.4, 44.4]])

1. **Deleting Observations with Missing Values (using dropna() of Pandas)**

# Load library

import pandas as pd

# Load data

importnumpy as np

# Create feature matrix

features = np.array([[1.1, 11.1],

[2.2, 22.2],

[3.3, 33.3],

[4.4, 44.4],

[np.nan, 55]])

dataframe = pd.DataFrame(features, columns=["feature\_1", "feature\_2"])

# Remove observations with missing values

dataframe.dropna()

**Result:**

|  | **feature\_1** | **feature\_2** |
| --- | --- | --- |
| **0** | 1.1 | 11.1 |
| **1** | 2.2 | 22.2 |
| **2** | 3.3 | 33.3 |
| **3** | 4.4 | 44.4 |

1. **(a) Univariate Analysis**

fromsklearn import datasets

importnumpy as np

import pandas as pd

importmatplotlib.pyplot as plt

iris = datasets.load\_iris()

# Let's convert to dataframe

iris = pd.DataFrame(data= np.c\_[iris['data'], iris['target']],columns= iris['feature\_names'] + ['species'])

# replace the values with class labels

iris.species = np.where(iris.species == 0.0, 'setosa', np.where(iris.species==1.0,'versicolor', 'virginica'))

# let's remove spaces from column name

iris.columns = iris.columns.str.replace (' ', '')

iris.describe()

**Result:**

**sepallength(cm) sepalwidth(cm) petallength(cm) petalwidth(cm)**

**count** 150.000000 150.000000 150.000000 150.000000

**mean** 5.843333 3.057333 3.758000 1.199333

**std** 0.828066 0.435866 1.765298 0.762238

**min** 4.300000 2.000000 1.000000 0.100000

**25%** 5.100000 2.800000 1.600000 0.300000

**50%** 5.800000 3.000000 4.350000 1.300000

**75%** 6.400000 3.300000 5.100000 1.800000

**max** 7.900000 4.400000 6.900000 2.500000

**(b) Multivariate Analysis**

# print the mean for each column by species

iris.groupby(by = "species").mean()

# plot for mean of each feature for each label class

iris.groupby(by = "species").mean().plot(kind="bar")

plt.title('Class vs Measurements')

plt.ylabel('mean measurement(cm)')

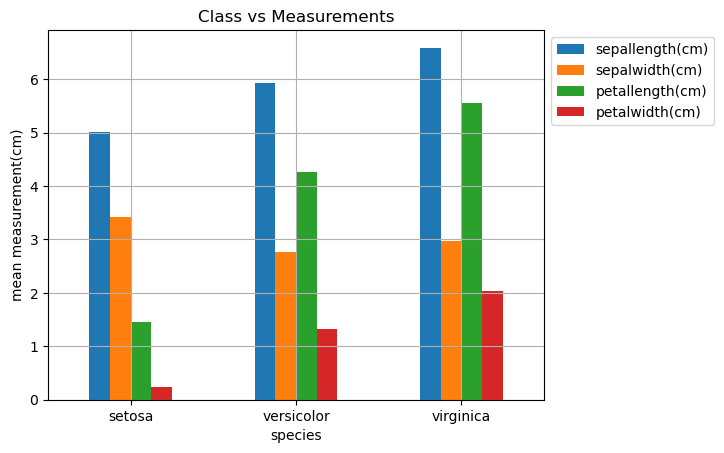
plt.xticks(rotation=0) # manage the xticks rotation

plt.grid(True)

# Use bbox\_to\_anchor option to place the legend outside plot area to be tidy

plt.legend(loc="upper left", bbox\_to\_anchor=(1,1))

**Result:**



1. **Imputing Missing Values (predict the missing values using KNNImputer))**

# import necessary libraries

importnumpy as np

import pandas as pd

# import the KNNimputer class

fromsklearn.impute import KNNImputer

# create dataset for marks of a student

dict = {'Maths':[80, 90, np.nan, 95],

'Chemistry': [60, 65, 56, np.nan],

'Physics':[np.nan, 57, 80, 78],

'Biology' : [78,83,67,np.nan]}

# creating a data frame from the list

Before\_imputation = pd.DataFrame(dict)

#print dataset before imputaion

print("Data Before performing imputation\n",Before\_imputation)

# create an object for KNNImputer

imputer = KNNImputer(n\_neighbors=2)

After\_imputation = imputer.fit\_transform(Before\_imputation)

# print dataset after performing the operation

print("\n\nAfter performing imputation\n",After\_imputation)

**Result:**

Data Before performing imputation

Maths Chemistry Physics Biology

0 80.0 60.0 NaN 78.0

1 90.0 65.0 57.0 83.0

2 NaN 56.0 80.0 67.0

3 95.0 NaN 78.0 NaN

After performing imputation

[[80. 60. 68.5 78. ]

[90. 65. 57. 83. ]

[87.5 56. 80. 67. ]

[95. 58. 78. 72.5]]

1. **Imputing Missing Values [using SimpleImputer library of sklearn]**

import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

students = [[85, 'M', 'verygood'],

[95, 'F', 'excellent'],

[75, None,'good'],

[np.NaN, 'M', 'average'],

[70, 'M', 'good'],

[np.NaN, None, 'verygood'],

[92, 'F', 'verygood'],

[98, 'M', 'excellent']]

dfstd = pd.DataFrame(students)

dfstd.columns = ['marks', 'gender', 'result']

print("Data Before performing imputation\n",dfstd)

# Missing values is represented using NaN and hence specified. If it

# is empty field, missing values will be specified as follows:

#

# Imputing with mean value

imputer = SimpleImputer(missing\_values=np.NaN, strategy='mean')

dfstd.marks= imputer.fit\_transform(dfstd['marks'].values.reshape(-1,1))[:,0]

print("\n\nAfter performing imputation\n",dfstd)

**Result:**

Data Before performing imputation

marks gender result

0 85.0 M verygood

1 95.0 F excellent

2 75.0 None good

3 NaN M average

4 70.0 M good

5 NaN None verygood

6 92.0 F verygood

7 98.0 M excellent

After performing imputation

marks gender result

0 85.000000 M verygood

1 95.000000 F excellent

2 75.000000 None good

3 85.833333 M average

4 70.000000 M good

5 85.833333 None verygood

6 92.000000 F verygood

7 98.000000 M excellent

**Other Options:**

# Imputing with **median value**

#

imputer = SimpleImputer(missing\_values=np.NaN, strategy='median')

#

# Imputing with **most frequent / mode value**

#

imputer = SimpleImputer(missing\_values=np.NaN, strategy='most\_frequent')

#

# Imputing with **constant value**; The command below replaces the missing

# value with constant value such as 80

#

imputer = SimpleImputer(missing\_values=np.NaN, strategy='constant', fill\_value=80)

1. **know\_your\_data\_IRIS**

from sklearn import datasets

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sb

iris = datasets.load\_iris()

# Let's convert to dataframe

iris = pd.DataFrame(data= np.c\_[iris['data'], iris['target']],columns= iris['feature\_names'] + ['species'])

iris.columns = iris.columns.str.replace (' ', '')

#########################################

iris.describe()

#########################################

print(iris)

#########################################

print("Histogram")

iris.hist()

#########################################

print("Line graph")

iris.plot()

#########################################

print("Distance Plot")

sb.distplot(iris['petallength(cm)'])

#########################################

print("Box plot")

sb.boxenplot(iris['petallength(cm)'])

#########################################