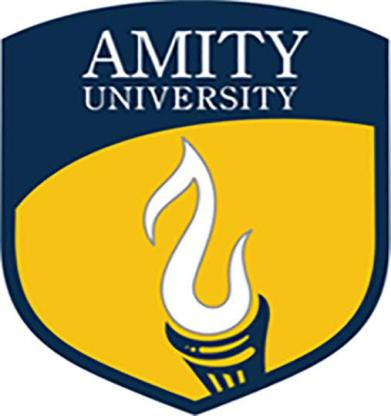
**AIML-301**

**Practical Lab File**



**AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY**

**Submitted By: Submitted To:**

Vaibhav Mr.Roshan Lal Chokkar

A2305219046 Assistant Professor-III

6CSE1Y Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH, NOIDA

SESSION 2021-22

**INDEX**

| **S. No** | **Name of Experiment** | **Date of Submission** | **Date of Evaluation** | **Total Marks** | **Marks Obtained** | **Remarks/ Signature** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | **Python Environment** | 10.01.2022 | 07.03.2022 |  |  |  |
| 2 | **Pandas Commands** | 17.01.2022 | 07.03.2022 |  |  |  |
| 3 | **Linear Regression** | 23.01.2022 | 07.03.2022 |  |  |  |
| 4 | **Logistic Regression** | 31.01.2022 | 07.03.2022 |  |  |  |
| 5 | **Gradient Descent** | 07.02.2022 |  |  |  |  |
| 6 | **Missing Values** | 14.02.2022 |  |  |  |  |
| 7 |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |

**EXPERIMENT-1**

**AIM: Python programming environment**

**INTRODUCTION:**

Python is a widely used interpreted, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming.

In Python, we don’t need to declare the type of variable because it is a dynamically typed language.

For example, x = 10

Here, x can be anything such as String, int, etc.

There are many features in Python, some of which are discussed below :-

**1. Easy to code**

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

**2. Free and Open Source**

Python language is freely available at the official website and you can download it. Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

**3. Object-Oriented Language**

One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

**4. High-Level Language**

Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

**5. Python is Portable language**

Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

**6. Interpreted Language**

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called bytecode.

**7. Large Standard Library**

Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

**8. Dynamically Typed Language**

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don’t need to specify the type of variable. Python also supports multiple inheritance.

**Hardware and Software requirements:**

Operating Systems and CPU architecture:

* Windows 7 or 10
* Mac OS X 10.11 or higher, 64-bit
* Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)
* x86 64-bit CPU (Intel / AMD architecture)
* Python v3.9.1 is the first version supporting macOS 11 Big Sur. With Xcode 11 and later it is now possible to build “Universal 2” binaries which work on Apple Silicon.
* RAM and free disk space:
* 4 GB RAM
* 5 GB free disk space

**Platform used:**

Jupyter notebook (Windows 11)

**Python 2 and Python 3 key differences:**

| **Comparison Parameter** | **Python 2** | **Python 3** |
| --- | --- | --- |
| Year of Release | Python 2 was released in the year 2000. | Python 3 was released in the year 2008. |
| “Print” Keyword | In Python 2, print is considered to be a statement and not a function. | In Python 3, print is considered to be a function and not a statement. |
| Storage of Strings | In Python 2, strings are stored as ASCII by default. | In Python 3, strings are stored as UNICODE by default. |
| Exceptions | In Python 2, exceptions are enclosed in notations. | In Python 3, exceptions are enclosed in parentheses. |
| Iteration | In Python 2, the xrange() function has been defined for iterations. | In Python 3, the new Range() function was introduced to perform iterations. |
| Ease of Syntax | Python 2 has more complicated syntax than Python 3. | Python 3 has an easier syntax compared to Python 2. |
| Usage in today’s times | Python 2 is no longer in use since 2020. | Python 3 is more popular than Python 2 and is still in use in today’s times. |
| Application | Python 2 was mostly used to become a DevOps Engineer. It is no longer in use after 2020. | Python 3 is used in a lot of fields like Software Engineering, Data Science, etc. |

**Python installation:**

The latest version of python is **3.10.0** but I am using version **3.7.9**

Visit the link [*https://www.python.org/downloads/*](https://www.python.org/downloads/) to download the latest release of [Python](https://www.javatpoint.com/python-tutorial).

## **Step 1 −** Select Version of Python to InstallTable Description automatically generated

**Step - 2: Click on the Install Now**

Graphical user interface, text, application

Description automatically generated

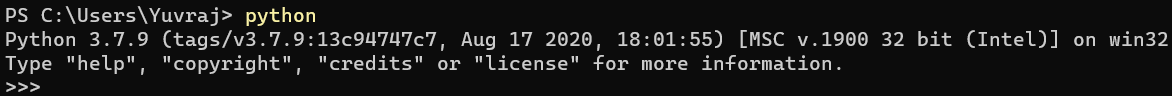
The installation process will take few minutes to complete and once the installation is successful, the following screen is displayed.

**Graphical user interface, text, application

Description automatically generated**

## **Step 3 −** Verify Python is installed on Windows

* Open the command prompt.
* Type ‘python’ and press enter.



The latest version of Python is **3.10.0** but I have downloaded the Python version **3.7.9**

**EXPERIMENT 2**

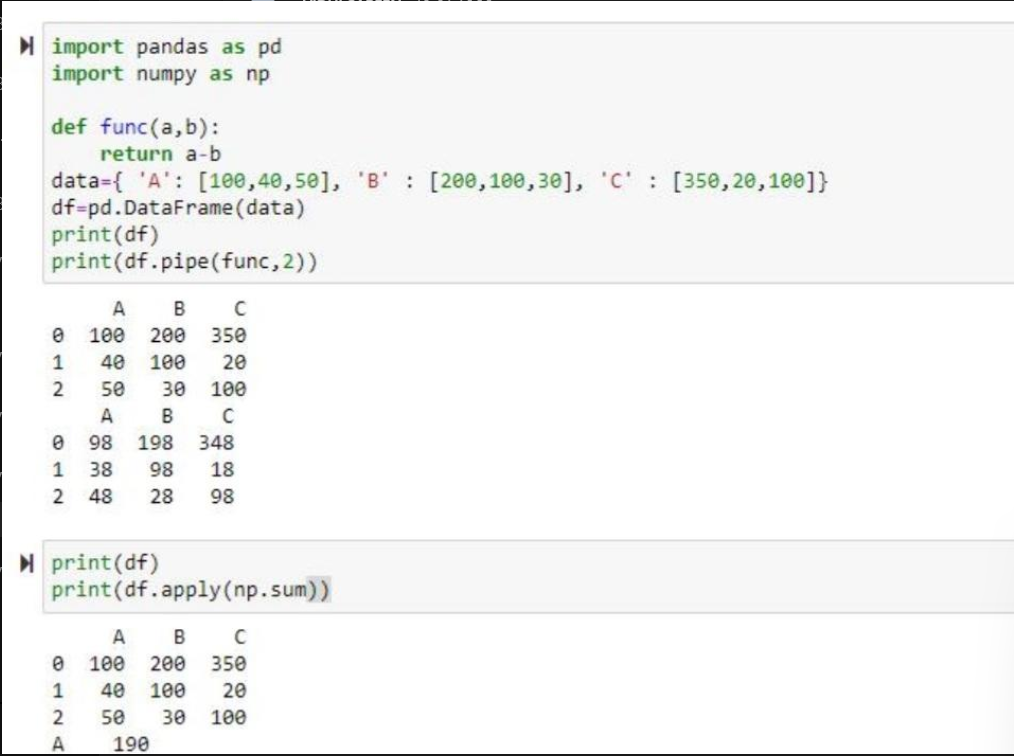
**AIM-** To Implement PANDAS FUNCTIONS.

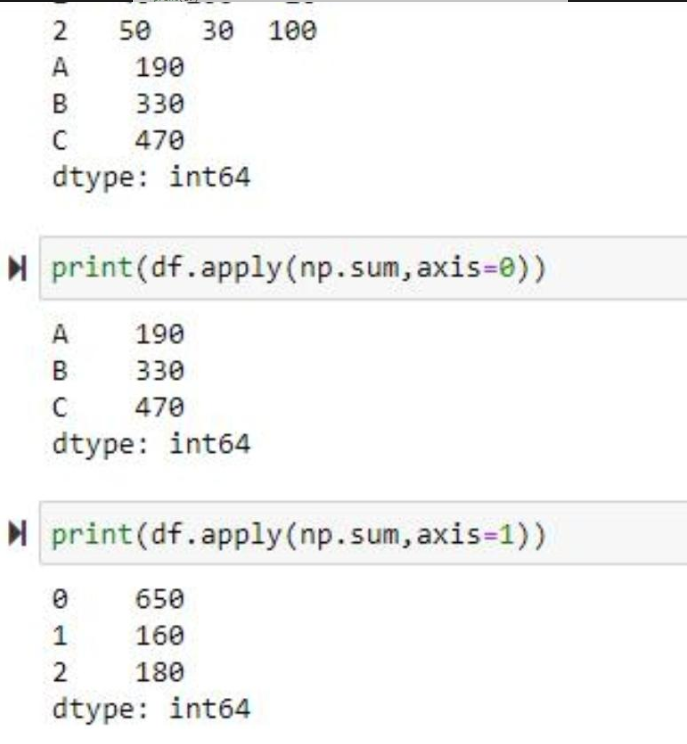
**THEORY-**

Pandas is a predominantly used **python data analysis library**. It provides many functions and methods to expedite the data analysis process. What makes pandas so common is its functionality, flexibility, and simple syntax.

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns

**CODE & output-**

****

****

***EXPERIMENT 3***

**AIM**- TO IMPLEMENT LINEAR REGRESSION

**THEORY**- Regression searches for relationships among [variables](https://realpython.com/python-variables/).

For example, you can observe several employees of some company and try to understand how their salaries depend on the **features**, such as experience, level of education, role, city they work in, and so on.

**Linear regression** is probably one of the most important and widely used regression techniques. It’s among the simplest regression methods. One of its main advantages is the ease of interpreting results.

**Multiple Linear Regression** attempts to model the relationship between two or more features and a response by fitting a linear equation to observed data. The steps to perform multiple linear Regression are almost similar to that of simple linear Regression. The Difference Lies in the evaluation. We can use it to find out which factor has the highest impact on the predicted output and now different variables relate to each other.

**Code-**

1. **LINEAR REGRESSION-**

import NumPy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# Number of observations/points

n = np.size(x)

# Mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# Calculating cross-deviation and deviation about x

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# Calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# Plotting the actual points as scatter plot

plt.scatter(x, y, color = "m",

marker = "o", s = 30)

# Predicted response vector

y\_pred = b[0] + b[1]\*x

# Plotting the regression line

plt.plot(x, y\_pred, color = "g")

# Putting labels

plt.xlabel('x')

plt.ylabel('y')

# Function to show plot

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

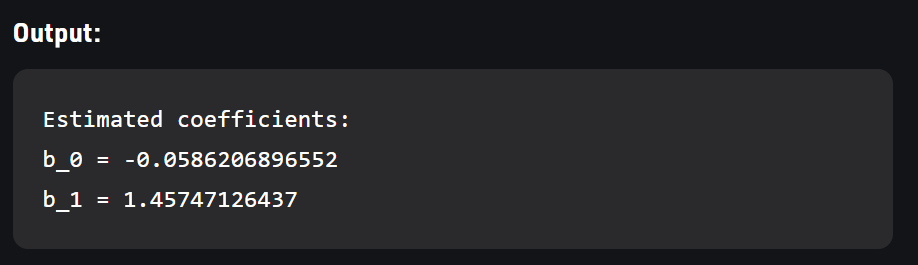
# plotting regression line

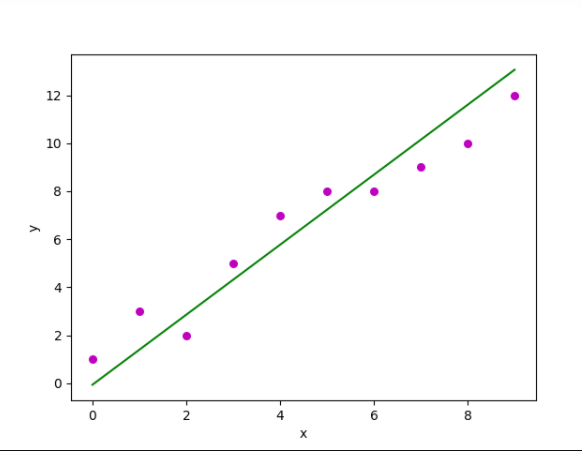
plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT-**





**B. MULTIPLE REGRESSION-**

**Code-**

import numpy as np

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

def generate\_dataset(n):

x = []

y = []

random\_x1 = np.random.rand()

random\_x2 = np.random.rand()

for i in range(n):

x1 = i

x2 = i/2 + np.random.rand()\*n

x.append([1, x1, x2])

y.append(random\_x1 \* x1 + random\_x2 \* x2 + 1)

return np.array(x), np.array(y)

x, y = generate\_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()

ax = fig.gca(projection ='3d')

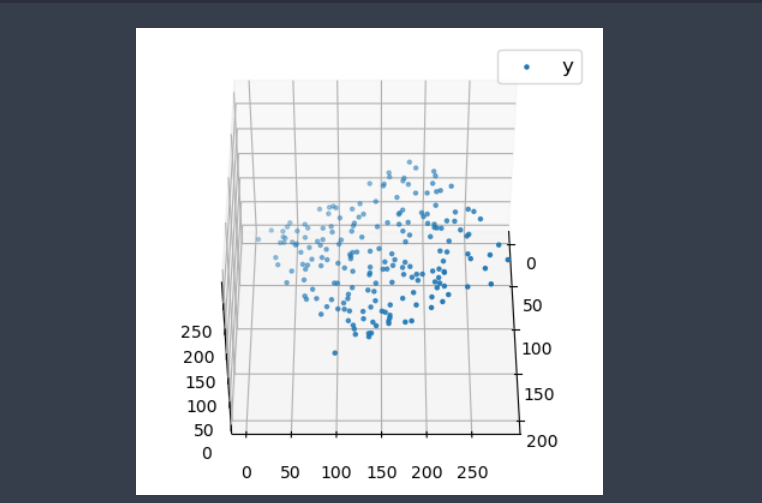
ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)

ax.legend()

ax.view\_init(45, 0)

plt.show()

**output-**

****

**EXPERIMENT 4**

AIM: Implementing Logistic Regression Algorithm

**CODE:**

import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix,accuracy\_score,roc\_curve,roc\_auc\_score

import sweetviz

from category\_encoders.one\_hot import OneHotEncoder

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv('adult.data')

columns = ['age','workclass','fnlwgt','education','education-num','marital-status','occupation','relationship','race','sex','capital-gain','capital-loss','hours-per-week','native-country','income']

for colum in df.columns:

if df[colum].dtype == object:

df[colum] = OneHotEncoder().fit\_transform(df[colum])

df = MinMaxScaler().fit\_transform(df)

df = pd.DataFrame(df, columns= columns)

correlations = df.corr()['income'].drop('income')

def get\_features(correlation\_threshold):

abs\_corrs = correlations.abs()

high\_correlations = abs\_corrs[abs\_corrs > correlation\_threshold].index.values.tolist()

return high\_correlations

features = get\_features(0.13)

x = df[features]

y = df.income

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,random\_state= 4)

classifier = LogisticRegression()

classifier.fit(x\_train,y\_train)

# print(classifier.score(x\_test,y\_test))

predictions = classifier.predict(x\_test)

print("confusion\_matrix:-")

print(confusion\_matrix(y\_test,predictions))

probs = (classifier.predict\_proba(x\_test)[:,1])

fpr, tpr, thresholds = roc\_curve(y\_test, probs)

accuracy\_ls = []

for thres in thresholds:

y\_pred = np.where(probs > thres, 1, 0)

accuracy\_ls.append(accuracy\_score(y\_test, y\_pred, normalize=True))

accuracy\_ls = pd.concat([pd.Series(thresholds), pd.Series(accuracy\_ls)],

axis=1)

accuracy\_ls.columns = ['thresholds', 'accuracy']

accuracy\_ls.sort\_values(by='accuracy', ascending=False, inplace=True)

threshold = accuracy\_ls.iloc[1,0]

#print(threshold)

preds = np.where(classifier.predict\_proba(x\_test)[:,1] > threshold, 1, 0)

print("RMSE Score:-", accuracy\_score(y\_test,preds))

# Output:

# Screenshot (934)

**EXPERIMENT 5-**

**AIM**- TO IMPLEMENT GRADIENT DESCENT

**THEORY**-

[**Stochastic gradient descent**](https://en.wikipedia.org/wiki/Stochastic_gradient_descent) is an optimization algorithm often used in machine learning applications to find the model parameters that correspond to the best fit between predicted and actual outputs. It’s an inexact but powerful technique.

Stochastic gradient descent is widely used in machine learning applications. Combined with [backpropagation](https://brilliant.org/wiki/backpropagation/), it’s dominant in [neural network](https://realpython.com/python-keras-text-classification/#a-primer-on-deep-neural-networks) training applications.

**CODE-**

import warnings

warnings.filterwarnings("ignore")

from sklearn.datasets import load\_boston

from random import seed

from random import randrange

from csv import reader

from math import sqrt

from sklearn import preprocessing

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from prettytable import PrettyTable

from sklearn.linear\_model import SGDRegressor

from sklearn import preprocessing

from sklearn.metrics import mean\_squared\_error

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

X = load\_boston().data

Y = load\_boston().target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=0)

scaler = preprocessing.StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

X\_train = pd.DataFrame(data = X\_train, columns=load\_boston().feature\_names)

X\_train['Price'] = list(y\_train)

X\_test = pd.DataFrame(data = X\_test, columns=load\_boston().feature\_names)

X\_test['Price'] = list(y\_test)

def sgd\_regressor(X, y, learning\_rate=0.2, n\_epochs=1000, k=40):

w = np.random.randn(1,13) # Randomly initializing weights

b = np.random.randn(1,1) # Random intercept value

epoch=1

while epoch <= n\_epochs:

temp = X.sample(k)

X\_tr = temp.iloc[:,0:13].values

y\_tr = temp.iloc[:,-1].values

Lw = w

Lb = b

loss = 0

y\_pred = []

sq\_loss = []

for i in range(k):

Lw = (-2/k \* X\_tr[i]) \* (y\_tr[i] - np.dot(X\_tr[i],w.T) - b)

Lb = (-2/k) \* (y\_tr[i] - np.dot(X\_tr[i],w.T) - b)

w = w - learning\_rate \* Lw

b = b - learning\_rate \* Lb

y\_predicted = np.dot(X\_tr[i],w.T)

y\_pred.append(y\_predicted)

loss = mean\_squared\_error(y\_pred, y\_tr)

print("Epoch: %d, Loss: %.3f" %(epoch, loss))

epoch+=1

learning\_rate = learning\_rate/1.02

return w,b

def predict(x,w,b):

y\_pred=[]

for i in range(len(x)):

temp\_ = x

X\_test = temp\_.iloc[:,0:13].values

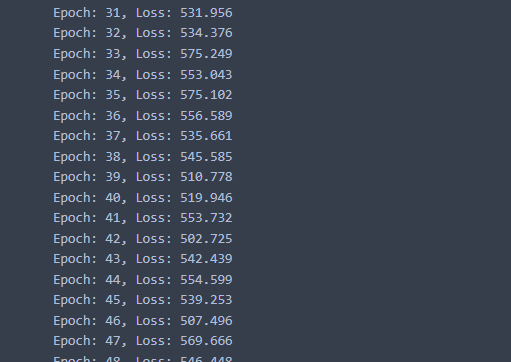
y = np.asscalar(np.dot(w,X\_test[i])+b)

y\_pred.append(y)

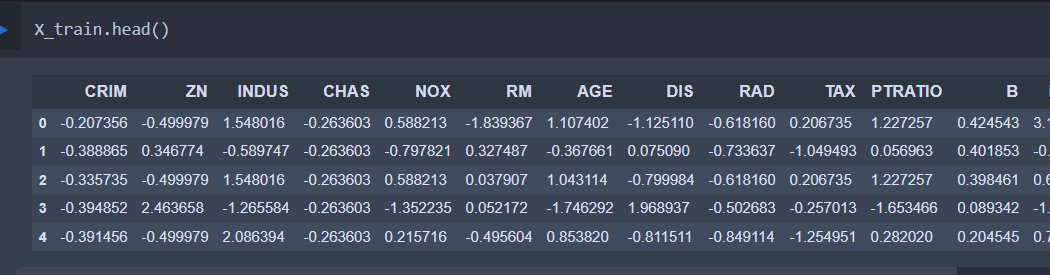
return np.array(y\_pred)

w,b = sgd\_regressor(X\_train,y\_train)

y\_pred\_customsgd = predict(X\_test,w,b)



**OUTPUT**

****

from matplotlib.pyplot import figure

plt.figure(figsize=(25,6))

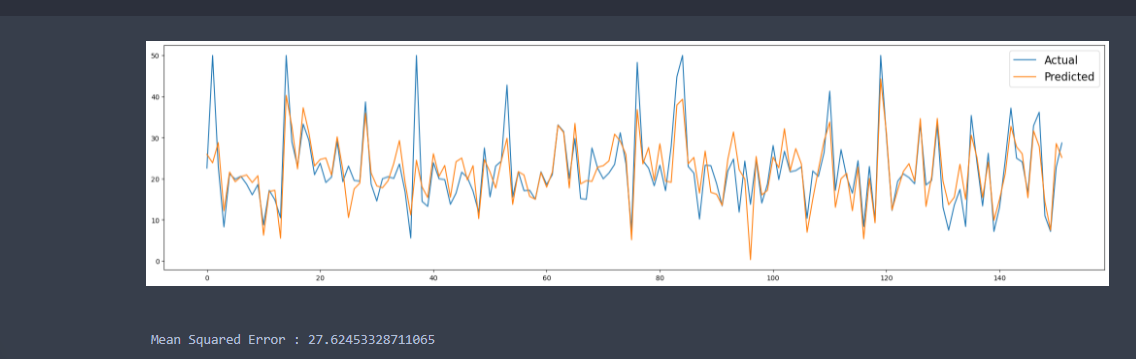
plt.plot(y\_test, label='Actual')

plt.plot(y\_pred\_customsgd, label='Predicted')

plt.legend(prop={'size': 16})

plt.show()

print('Mean Squared Error :',mean\_squared\_error(y\_test, y\_pred\_customsgd))



**EXPERIMENT - 6**

**AIM:** To use any data to apply the concept of missing values.

**DATASET USED:** Used the Kaggle Melbourne Housing Dataset. This data gives different sales prices with respect to type of houses in Melbourne.

**THEORY:**

**Deleting Rows**

This method commonly used to handle the null values. Here, we either delete a particular row if it has a null value for a particular feature and a particular column if it has more than 70-75% of missing values. This method is advised only when there are enough samples in the data set. One has to make sure that after we have deleted the data, there is no addition of bias. Removing the data will lead to loss of information which will not give the expected results while predicting the output.

**Replacing With Mean/Median/Mode**

This strategy can be applied on a feature which has numeric data like the age of a person or the ticket fare. We can calculate the mean, median or mode of the feature and replace it with the missing values. This is an approximation which can add variance to the data set. But the loss of the data can be negated by this method which yields better results compared to removal of rows and columns. Replacing with the above three approximations are a statistical approach of handling the missing values. This method is also called as leaking the data while training. Another way is to approximate it with the deviation of neighbouring values. This works better if the data is linear.

**Assigning An Unique Category**

A categorical feature will have a definite number of possibilities, such as gender, for example. Since they have a definite number of classes, we can assign another class for the missing values. Here, the features Cabin and Embarked have missing values which can be replaced with a new category, say, U for ‘unknown’. This strategy will add more information into the dataset which will result in the change of variance.

**INPUTS & OUTPUTS:**

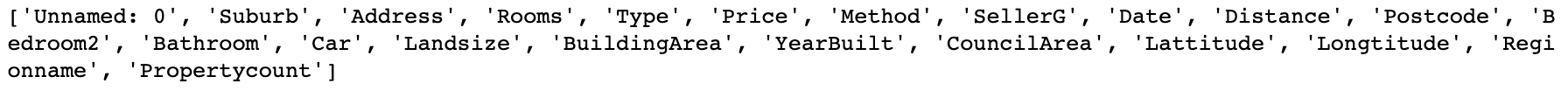
import pandas as pd

import numpy as np

import seaborn as sns

data = pd.read\_csv(“melb\_data.csv")

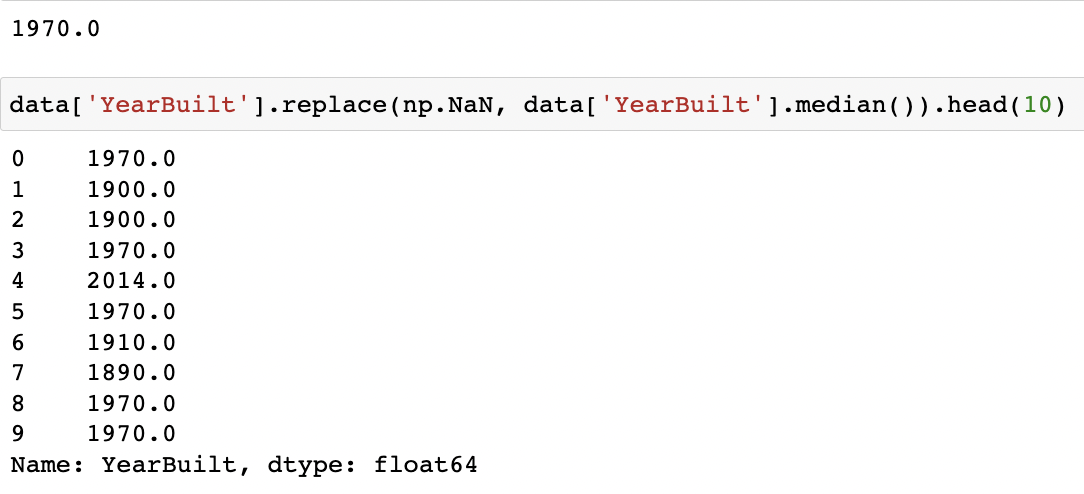
print(data.shape)

print(list(data.columns))

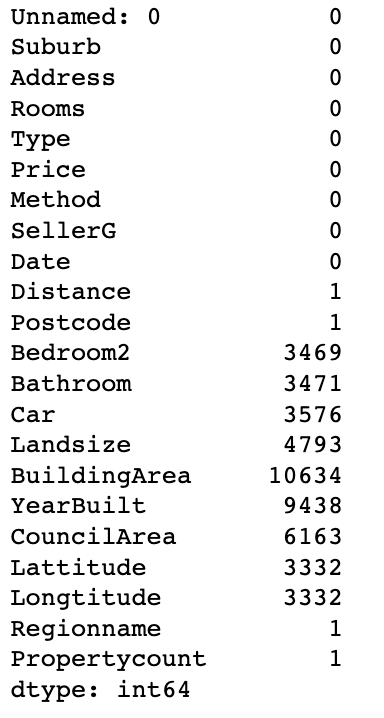
data.isnull().sum()

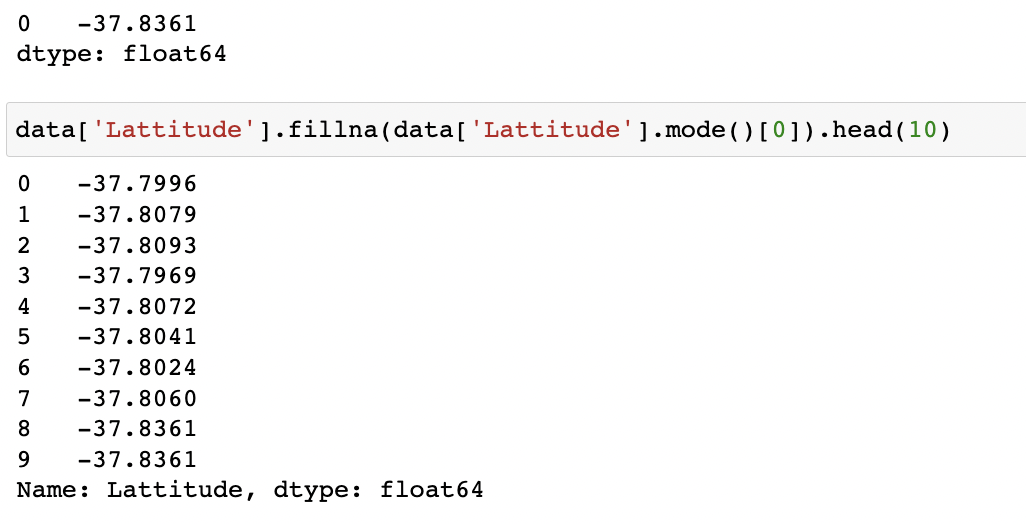
# Replacing With Mean/Median/Mode:

data[‘YearBuilt'].median()

data['YearBuilt'].replace(np.NaN, data[‘YearBuilt'].median()).head(10)

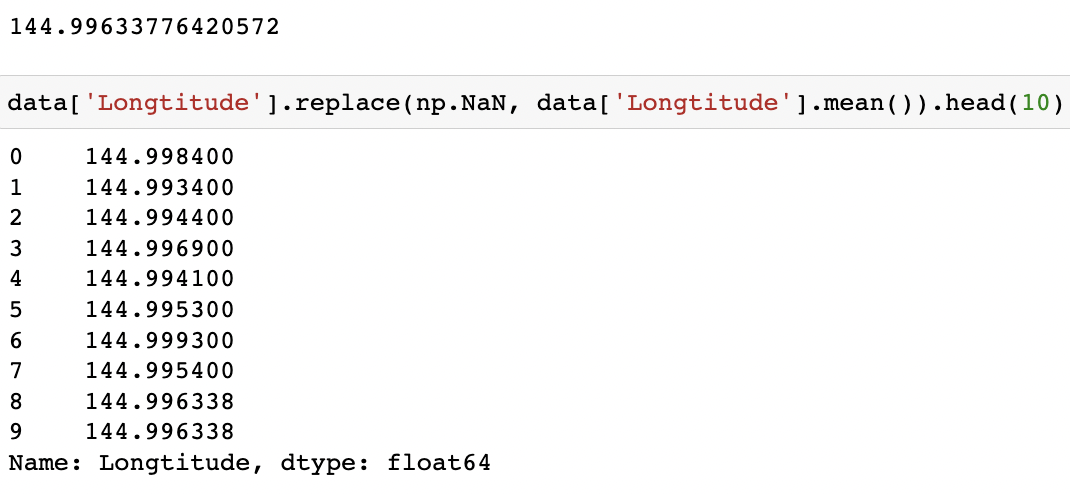
data[‘Lattitude'].head(10)

data[‘Lattitude'].mode()

data[‘Lattitude'].fillna(data['Lattitude'].mode()[0]).head(10)

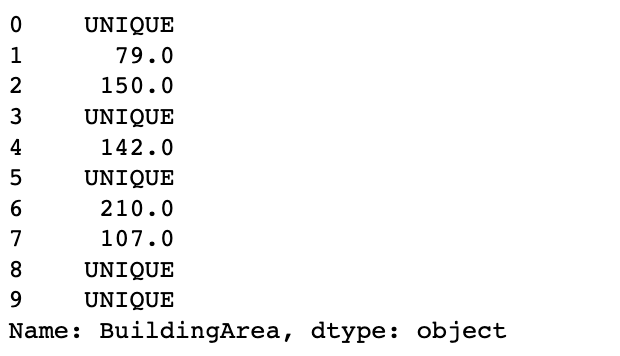
data[‘Longtitude’].head(10)

data[‘Longtitude'].mean()

data['Longtitude'].replace(np.NaN, data[‘Longtitude’].mean()).head(10)

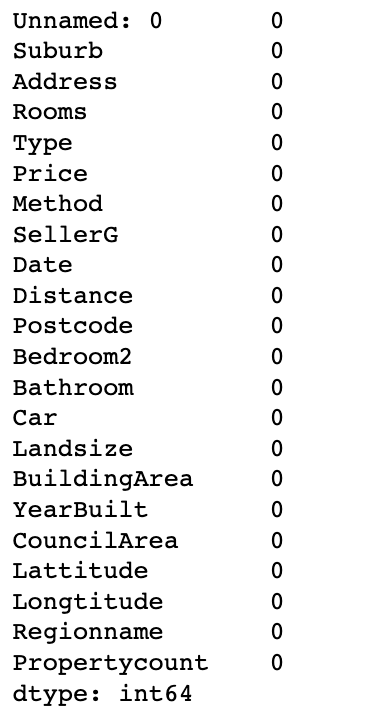
# Assigning a unique catagory:

data[‘BuildingArea'].head(10)

data[‘BuildingArea'].fillna('UNIQUE').head(10)

# Deleting rows:

data.dropna(inplace = True)

data.isnull().sum()

**RESULT:** Used the dataset to apply the concept of missing values.