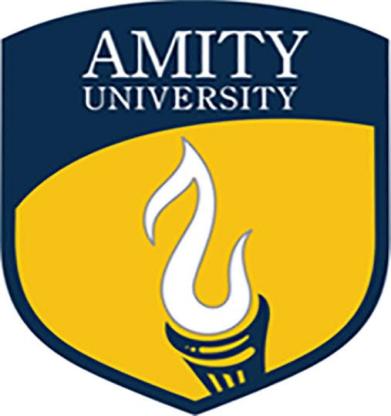
**AIML-301**

**Practical Lab File**



**AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY**

**Submitted By: Submitted To:**

Vibhu Gupta Mr.Roshan Lal Chokkar

A2305219217 Assistant Professor- III

B.tech CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMITTY 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** | 17.01.2022 | 07.03.2022 |  |  |  |
| 2 | **Pandas Commands** | 24.01.2022 | 07.03.2022 |  |  |  |
| 3 | **Linear Regression** | 31.01.2022 | 07.03.2022 |  |  |  |
| 4 | **Logistic Regression** | 07.02.2022 | 07.03.2022 |  |  |  |
| 5 | **Gradient Descent** | 14.02.2022 | 07.03.2022 |  |  |  |
| 6 | **Missing Values** | 28.02.2022 | 07.03.2022 |  |  |  |
| 7 | **Underfitting & Overfitting** | 07.03.2022 | 11.04.2022 |  |  |  |
| 8 | **K-means Cluster** | 14.03.2022 | 11.04.2022 |  |  |  |
| 9 | **SVM for Cancer classification** | 28.03.2022 | 11.04.2022 |  |  |  |
| 10 | **Open-Ended** | 04.04.2022 | 11.04.2022 |  |  |  |

**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/" \t "_blank) to download the latest release of [Python](https://www.javatpoint.com/python-tutorial).

## **Table Description automatically generatedStep 1 −** Select Version of Python to Install

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

**Graphical user interface, text, application

Description automatically generated**

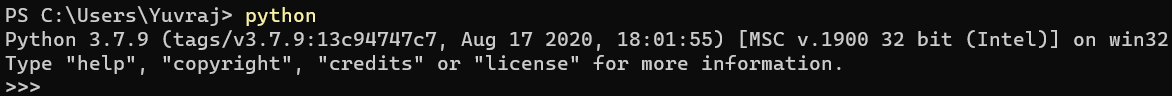
The installation process will take few minutes to complete and once the installation is successful, the following screen is displayed.

**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: Implement Pandas functions.

**CODE:**

Import pandas as pd

Import numpy as np

df = pd.read\_csv(“iris\_data.csv”)

df.head()

df.tail()

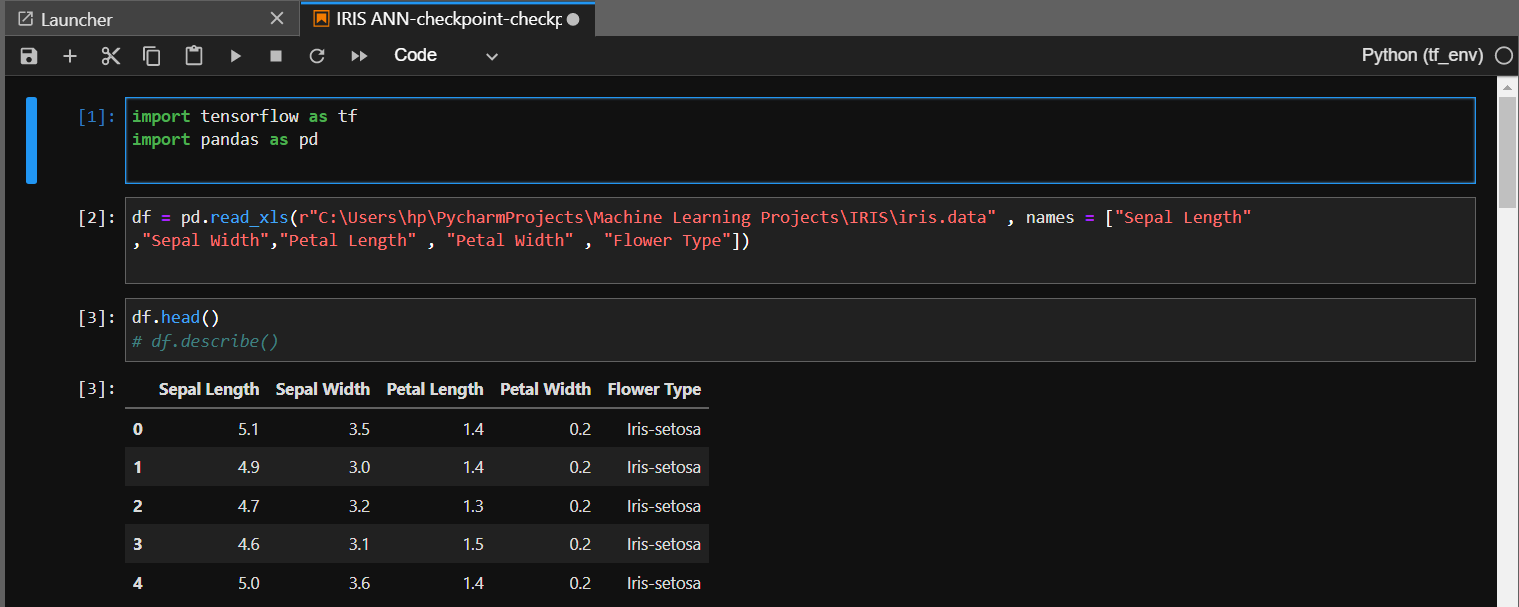
df.describe()

df.isnull().any()

df.isnull().sum()

df.petal\_width.unique()

# **Output:**



**EXPERIMENT - 3**

# AIM: Implement Linear Regression Algorithm using multiple variables.

**THEORY:** The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric.

The linear equation assigns one scale factor to each input value or column, called a coefficient and represented by the capital Greek letter Beta (B).

**DATEST USED:** used Kaggle Adult income dataset used to predict the income of an adult depending upon various features like 'school','sex','age','address','famsize','Pstatus','Medu','Fedu','Mjob','Fjob','reason','guardian','traveltime','studytime','failures','schoolsup','famsup','paid','activities','nursery','higher','internet','romantic','famrel','freetime','goout','Dalc','Walc','health','absences'

**CODE:**

## Importing Libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

import sweetviz

from category\_encoders.one\_hot import OneHotEncoder

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

from sklearn.linear\_model import LinearRegression

## Reading the File and Checking for null values

df = pd.read\_csv('abc.csv',delimiter= ';')

## Scaling and Encoding the data

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=['school','sex','age','address','famsize','Pstatus','Medu','Fedu','Mjob','Fjob','reason','guardian','traveltime','studytime','failures','schoolsup','famsup','paid','activities','nursery','higher','internet','romantic','famrel','freetime','goout','Dalc','Walc','health','absences','G1','G2','G3'])

correlations = df.corr()['G3'].drop('G3')

## Choosing the best threshold for improving the model

def get\_features(correlation\_threshold):

    abs\_corrs = correlations.abs()

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

    return high\_correlations

## Final Threshold with greatest Score

features = get\_features(0.06)

# print(len(features))

X = df[features]

Y = df.G3

## Spliting, Fiting, getting the score of Model

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

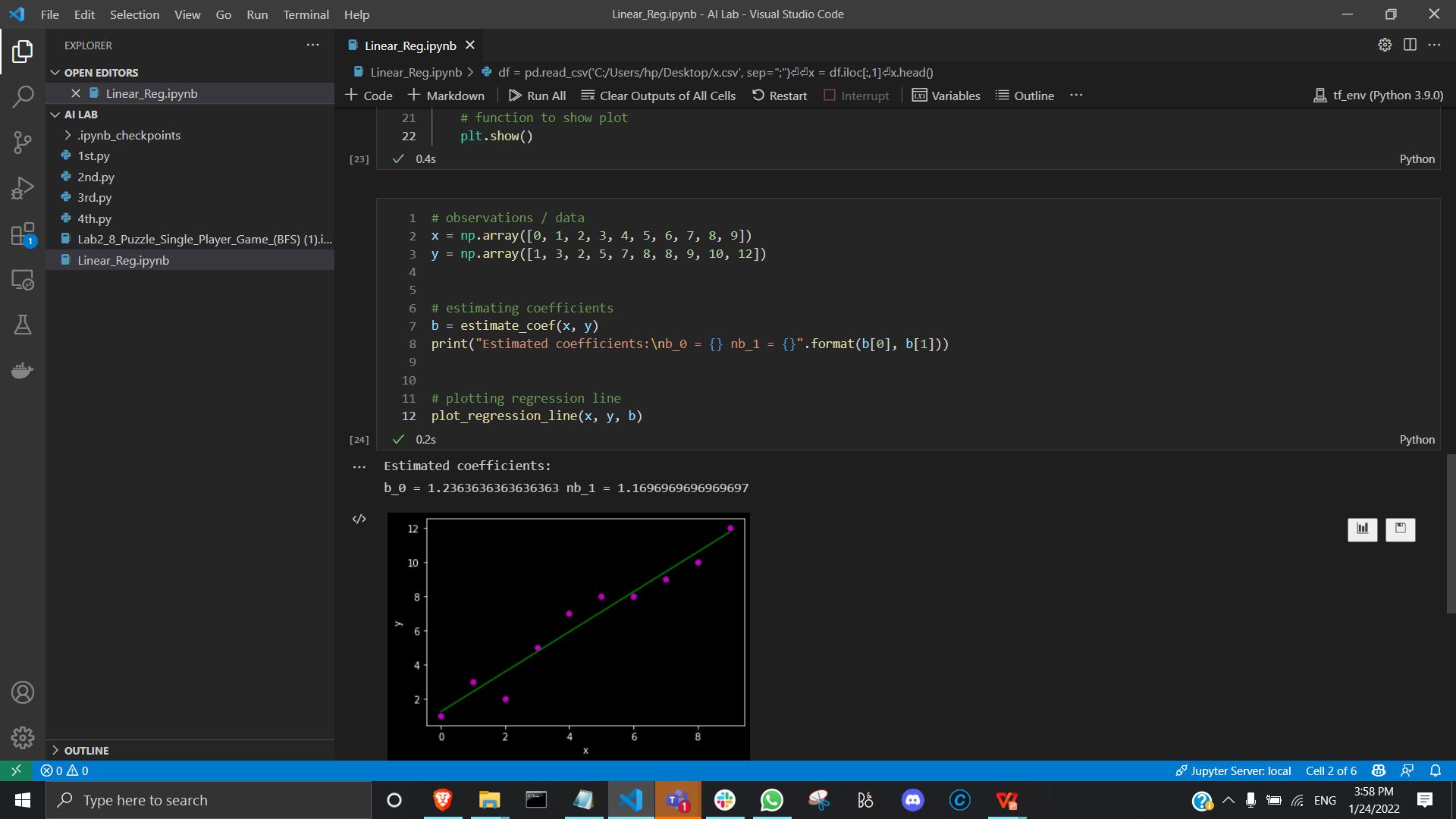
regressor = LinearRegression()

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

print("R2 Score of the regression model is :-",regressor.score(x\_test, y\_test))

print("The coefficients of each features",regressor.coef\_)

# Output:



**EXPERIMENT 4**

# AIM: Implementing Logistic Regression Algorithm

# **THEORY:** Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a very important aspect of Logistic regression and is dependent on the classification problem itself.

# The decision for the value of the threshold value is majorly affected by the values of precision and recall.

# **DATASET USED:** Used kaggle Dataset with following features 'age', 'workclass', 'fnlwgt', 'education','education-num','marital-status','occupation','relationship','race','sex','capital-gain','capital-loss','hours-per-week','native-country','income'

**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 use any data to apply the concept of gradient descent.

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

**THEORY:**

**Gradient Descent:** Gradient descent is an optimization algorithm that's used when training a machine learning model. It's based on a convex function and tweaks its parameters iteratively, to minimize a given function to its local minimum.

**CODE:**

import pandas as pd

Import numpy as np

Import matplotlib.pyplot as plt

dataset = pd.read\_csv(“USA\_Housing.csv”)

dataset.head()

X = dataset[“Avg. Area Income”]

Y = dataset[“Price”]

X.isnull().sum()

Y. isnull().sum()

plt.scatter(X,Y)

plt.xlabel(“Avg. Income”)

plt.rcParams[‘figure.figsize’] = (5.0,5.0)

plt.ylabel(“House Price”)

Text(0,0.5,’House\_Price’)

Chart, scatter chart

Description automatically generated

def mean squared\_error(y\_true, y predicted):

cost = np.sum ((y\_true-y\_predicted)\*\*2) / len(y\_true) return cost

def gradient descent (x, y, iterations = 1000, rate = 0 0.0000000001,stopping

threshold 1=1e-5) :

current weight = 0.1

current bias s = 0.01

iterations = iterations

learning\_rate= learning\_rate

n = float(len(x))

costs = []

weights = []

previous\_cost = None

for i in range(iterations):

y predicted = (current\_weight x) + current\_bias current\_cost = mean

squared\_error(y, y predicted)

if previous\_cost and abs(previous cost-current\_cost)<=stopping\_threshold:

break

previous\_cost = current\_cost

costs.append(current\_cost)

weights.append(current\_weight)

weight\_derivative = -(2/n) Sum(x (y-y predicted)) bias\_derivative = -(2/n)

sum(y-y\_predicted)

current weight = current\_weight - (learning\_rate= weight\_derivative)

current\_bias= current\_bias= (learning\_ratel bias\_derivative)

return current weight, current bias

m, c = 1, gradient\_descent (X, Y)

y\_pred= m\*X + c

plt.scatter(X,Y)

plt.plot(X,Y\_pred, ‘-y’)

Chart, scatter chart

Description automatically generated

**RESULT:** Used the dataset to apply the concept of gradient descent.

**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.

**CODE & OUTPUTS:**

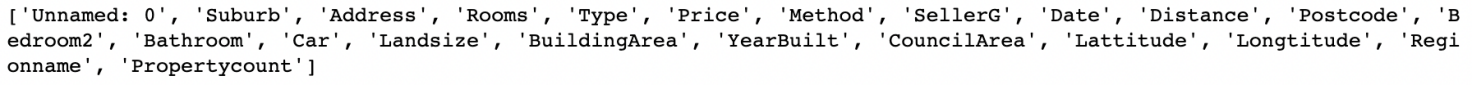
import pandas as pd

import numpy as np

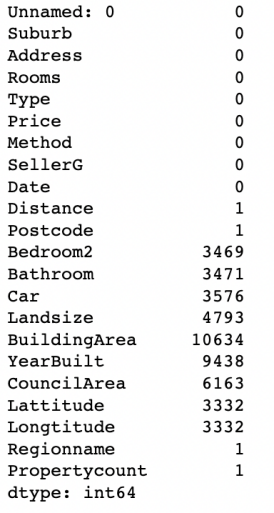
import seaborn as sns

data = pd.read\_csv(“melb\_data.csv")

print(data.shape)

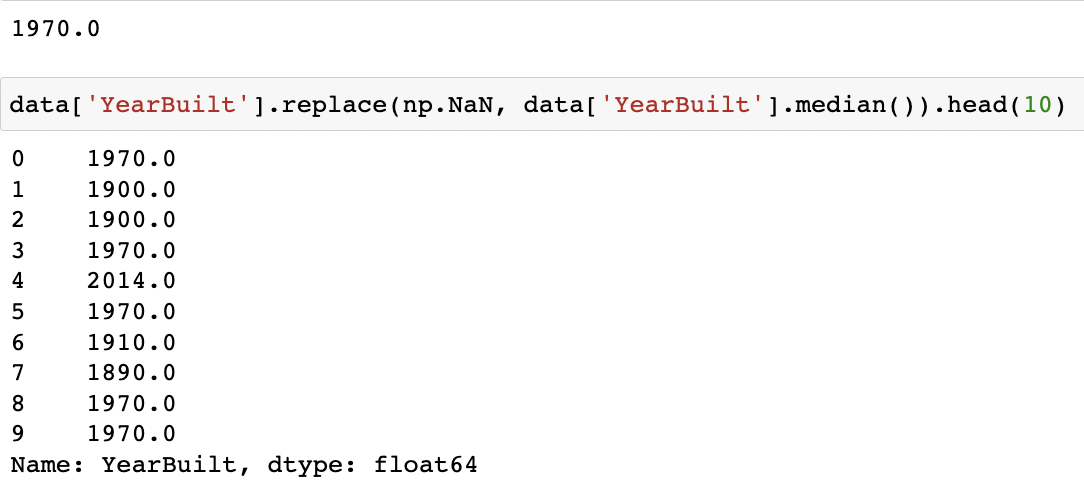
print(list(data.columns))

data.isnull().sum()



# Replacing With Mean/Median/Mode:

data[‘YearBuilt'].median()

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

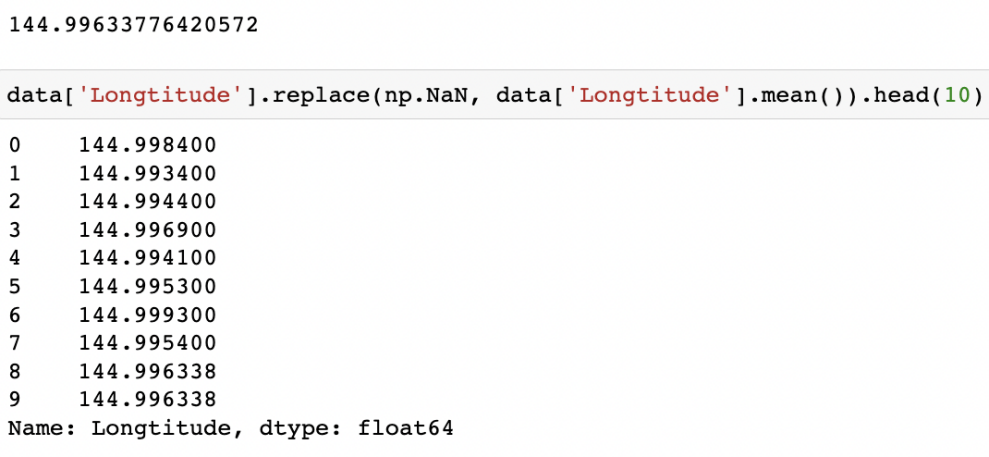
data[‘Lattitude'].head(10)

data[‘Lattitude'].mode()

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

data[‘Longtitude’].head(10)

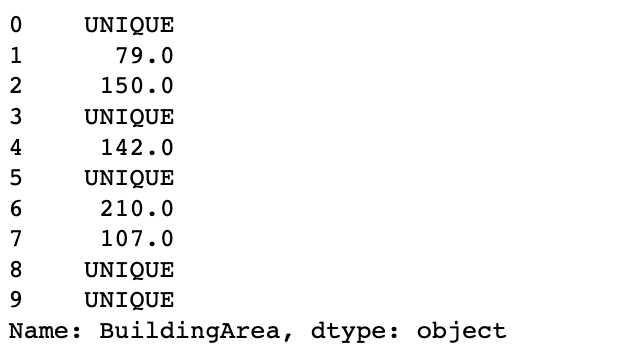
data[‘Longtitude'].mean()

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

# Assigning a unique catagory:

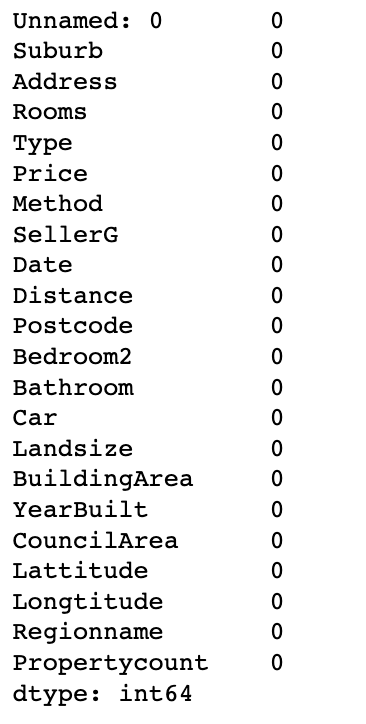
data[‘BuildingArea'].head(10)

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



# Deleting rows:

data.dropna(inplace = True)

data.isnull().sum()

**EXPERIMENT-7**

**AIM: To implement overfitting and underfitting on a dataset.**

**THEORY:**

**Underfitting:**   
A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data. Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have fewer data to build an accurate model and also when we try to build a linear model with fewer non-linear data.

**Techniques to reduce underfitting:**

1. Increase model complexity
2. Increase the number of features, performing feature engineering

**Overfitting:**   
A statistical model is said to be overfitted when we train it with a lot of data *(just like fitting ourselves in oversized pants!)*. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too many details and noise.

**Techniques to reduce overfitting:**

1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).

**CODE**:

from sklearn.datasets import make\_classification

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

from sklearn.metrics import accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

from matplotlib import pyplot

# create dataset

X, y = make\_classification(n\_samples=10000, n\_features=20, n\_informative=5,

n\_redundant=15, random\_state=1)

# split into train test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

# define lists to collect scores

train\_scores, test\_scores = list(), list()

# define the tree depths to evaluate

values = [i for i in range(1, 51)]

# evaluate a decision tree for each depth

for i in values:

# configure the model

model = KNeighborsClassifier(n\_neighbors=i)

# fit model on the training dataset

model.fit(X\_train, y\_train)

# evaluate on the train dataset

train\_yhat = model.predict(X\_train)

22

train\_acc = accuracy\_score(y\_train, train\_yhat)

train\_scores.append(train\_acc)

# evaluate on the test dataset

test\_yhat = model.predict(X\_test)

test\_acc = accuracy\_score(y\_test, test\_yhat)

test\_scores.append(test\_acc)

# summarize progress

print('>%d, train: %.3f, test: %.3f' % (i, train\_acc, test\_acc))

# plot of train and test scores vs number of neighbors

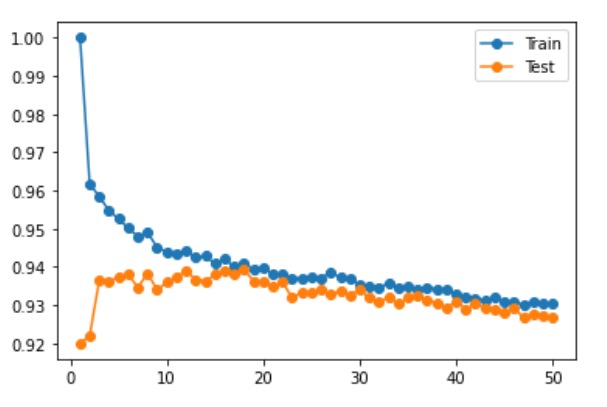
pyplot.plot(values, train\_scores, '-o', label='Train')

pyplot.plot(values, test\_scores, '-o', label='Test')

pyplot.legend()

pyplot.show()

**OUTPUT:**



**EXPERIMENT- 8**

**AIM:** To use any data to apply K-Means Algorithms

**DATASET USED:** Synthetic Data created with the help of numpy and pandasin the form a numpy array

**THEORY:**

As discussed we have created the synthetic data and used it to train our K-eans Clustering Model.

1. Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on. In this algorithm we determine 2 things:

//.

* Determines the best value for K center points or centroids by an iterative process.
* Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

**CODE & OUTPUT:**

import matplotlib.pyplot as plt

%matplotlib inline

import numpy as np

from sklearn.cluster import KMeans

X = np.array([[5,3],

     [10,15],

     [15,12],

     [24,10],

     [30,45],

     [85,70],

     [71,80],

     [60,78],

     [55,52],

     [80,91],])

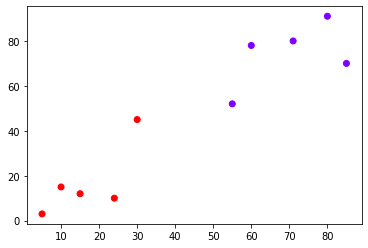
kmeans = KMeans(n\_clusters=2)

kmeans.fit(X)

print(kmeans.cluster\_centers\_)

print(kmeans.labels\_)

plt.scatter(X[:,0],X[:,1], c=kmeans.labels\_, cmap='rainbow')

****

**EXPERIMENT- 9**

**AIM:** To build a SVM model for Cancer classification

**DATASET USED:** Scikit learn breast caner dataset

**THEORY:**

A Support Vector Machine (SVM) is a binary linear classification whose decision boundary is explicitly constructed to minimize generalization error. It is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression and even outlier detection.

SVM is well suited for classification of complex but small or medium sized datasets.

**CODE & OUTPUT:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.svm import SVC

%matplotlib inline

#Import Cancer data from the Sklearn library

# Dataset can also be found here (http://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28diagnostic%29)

from sklearn.datasets import load\_breast\_cancer

cancer = load\_breast\_cancer()

df\_cancer = pd.DataFrame(np.c\_[cancer['data'], cancer['target']], columns = np.append(cancer['feature\_names'], ['target']))

df\_cancer.head()

sns.pairplot(df\_cancer, vars = ['mean radius', 'mean texture', 'mean perimeter', 'mean area',

'mean smoothness'] )

df\_cancer['target'].value\_counts()

plt.figure(figsize=(20,12))

sns.heatmap(df\_cancer.corr(), annot=True)

X = df\_cancer.drop(['target'], axis = 1)

y = df\_cancer['target']

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 = 20)

svc\_model = SVC()

svc\_model.fit(X\_train, y\_train)

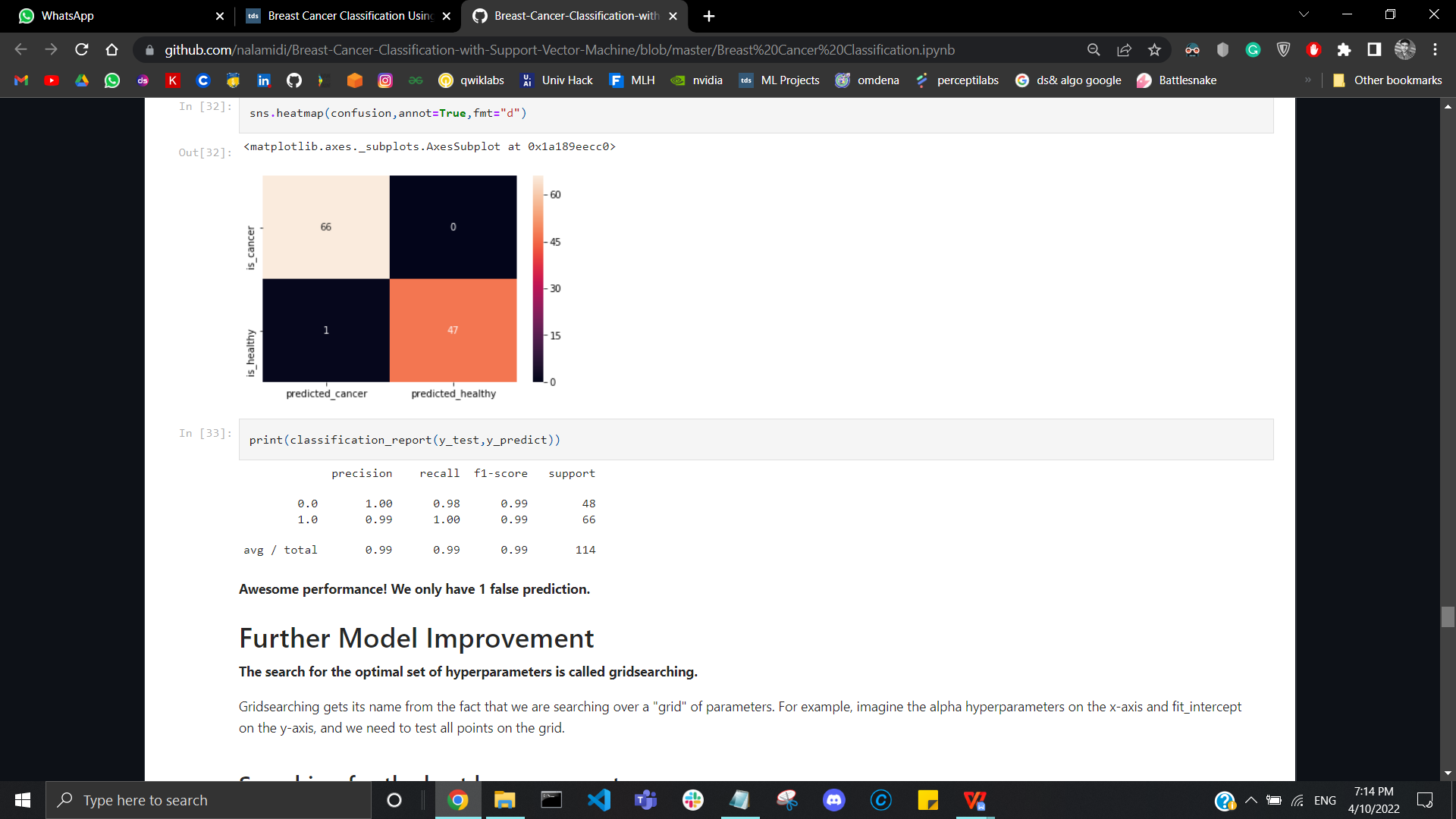
y\_predict = svc\_model.predict(X\_test)

cm = np.array(confusion\_matrix(y\_test, y\_predict, labels=[1,0]))

confusion = pd.DataFrame(cm, index=['is\_cancer', 'is\_healthy'],

columns=['predicted\_cancer','predicted\_healthy'])

print(classification\_report(y\_test, y\_predict))



**EXPERIMENT- 10**

**AIM:** To build a model for classification and regression using random forest algorithm

**DATASET USED:**

**THEORY:**

**CODE & OUTPUT:**