LABORATORY MANUAL

Computer Engineering Department

Machine Learning

Submitted by

Wania Urooj Suleman | 49178

Course Instructor

Saniya Ashraf

Lecturer, Department of Software Engineering,

Faculty of ICT, BUITEMS, Quetta.

Lab Engineer

Engr, Laila Baloch

Lab Engineer, Department of Software Engineering,

Faculty of ICT, BUITEMS, Quetta.

Session fall – 2022

BALOCHISTAN UNIVERSITY OF INFORMATION TECHNOLOGY, ENGINEERING AND MANAGEMENT SCIENCES, QUETTA.

CERTIFICATE

This is certified that Miss **Wania Urooj Suleman** bearing CMS ID **#49178** has successfully completed the laboratory manual of **Machine Learning** in her **7th semester** of **(BS)** Computer Engineering, Fall 202**2** under the supervision of his/her instructor **Saniya Ashraf.**

Lab Manual Marks **Instructor Signature**

TABLE OF CONTANT S

Table of Contents

CLO's PLO's MAPPING

Course Learning Outcomes				
S#	**CLO**	**Domain**	**Taxonomy level**	**PLO**
1	***Implement and analyze*** existing learning algorithms, including well-studied methods for classification, regression, structured prediction, clustering, and representation learning	Psychomotor	4	3
2	***Proficiently*** Integrate multiple facets of practical machine learning in a single system: data preprocessing, learning, regularization, and model selection	Psychomotor	3	5

LIST OF EXPERIMENTS

S#	**Experiment Descriptions**	**(CLO,PLO)**
1	Introduction and Installation of Python Machine Learning Stack (Anaconda), Interactive Terminal (IPython/Jupyter), Plotting (MatPlotLib/PyPlot), Version Control System (BitBucket/GitHub)	[CLO 2, PLO 5]
2	Introduction to NumPy and Pandas DataFrame	[CLO 2, PLO 5]
3	Linear Regression -Import Dataset, Reading and understanding the data. -Visualizing the data -Data Preparation -Splitting the data into training and test sets -Building a linear model -Residual analysis of the train data -Making predictions and evaluation	[CLO 1, PLO 3]
4	Dataset Exploration and Visualization	[CLO 1, PLO 3]
5	KNN Classifier -Estimate the accuracy of the classifier on future data, using the test data -Use the trained k-NN classifier model to classify new, previously unseen objects -Plot the decision boundaries of the k-NN classifier -Classification accuracy to the choice of the 'k' parameter K-NN classification accuracy to the train/test split proportion	[CLO 1, PLO 3]
6	Practice KNN Classification and Linear Regression on Diamond-Price dataset	[CLO 2, PLO 5]
7	Implement K- Nearest Neighbor Regression	[CLO 2, PLO 5]

8	Decision Trees, Tree ensembles and Random Forest Classification	[CLO 1, PLO 3]
9	Open-ended Lab	[CLO 2, PLO 5]
10	Support Vector Machine	[CLO 1, PLO 3]
11	ANN -Explore the inner workings of neurons/units and layers as well as a regression model	[CLO 1, PLO 3]
12	Optimize neural network, check and remove overfitting	[CLO 2, PLO 5]

Lab 1_____29/09/2022

1. Objectives:

After completing this lab, Students will be able to;

- Python Machine Learning Stack (Anaconda)
- Interactive Terminal (IPython/Jupyter)
- Plotting (MatPlotLib/PyPlot)
- Version Control System (BitBucket/GitHub

2. Corresponding CLO and PLO:

- CLO 2, PLO 5

3. Theory:

Google Colab

Colab or Colab notebook is an environment that is now provided by Google. This environment is based on Jupyter Notebook. It is one of the most efficient platforms for ML in the market.

Jupyter Notebook

Jupyter notebook is one of the most used platforms/ Machine Learning tools in the industry. It is a very efficient and fast processing platform.

Jupyter supports three languages, which are Julia, Python, and R. Hence combining all the three we get the name Jupyter. Jupyter notebook is great for Python coding. You can use Jupyter to run your ML codes.

Google Colab's environment is based on Jupyter. It allows us to store and share live code in the form of notebooks. We can also access it through various GUIs. These GUIs include anaconda navigator, winpython navigator, etc.

4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

5. Procedure:

1 Python Machine Learning Stack (Anaconda)

You will use Python in this course. In order to prepare for future assignments Labs and the final project, you will be asked to install Python and its useful packages via Anaconda. Anaconda is a high performance distribution of Python and R and includes over 100 of the most popular Python, R and Scala packages for data science. More information on Anaconda can be found here.

Follow these instructions to install Anaconda.

Make sure you confirm that Anaconda is installed and working by opening a terminal window and running the command

>> conda list
If Anaconda is installed and working, this will display a list of installed packages and their versions. You can also confirm that Anaconda is installed and working by opening a terminal window and running the command

>> python

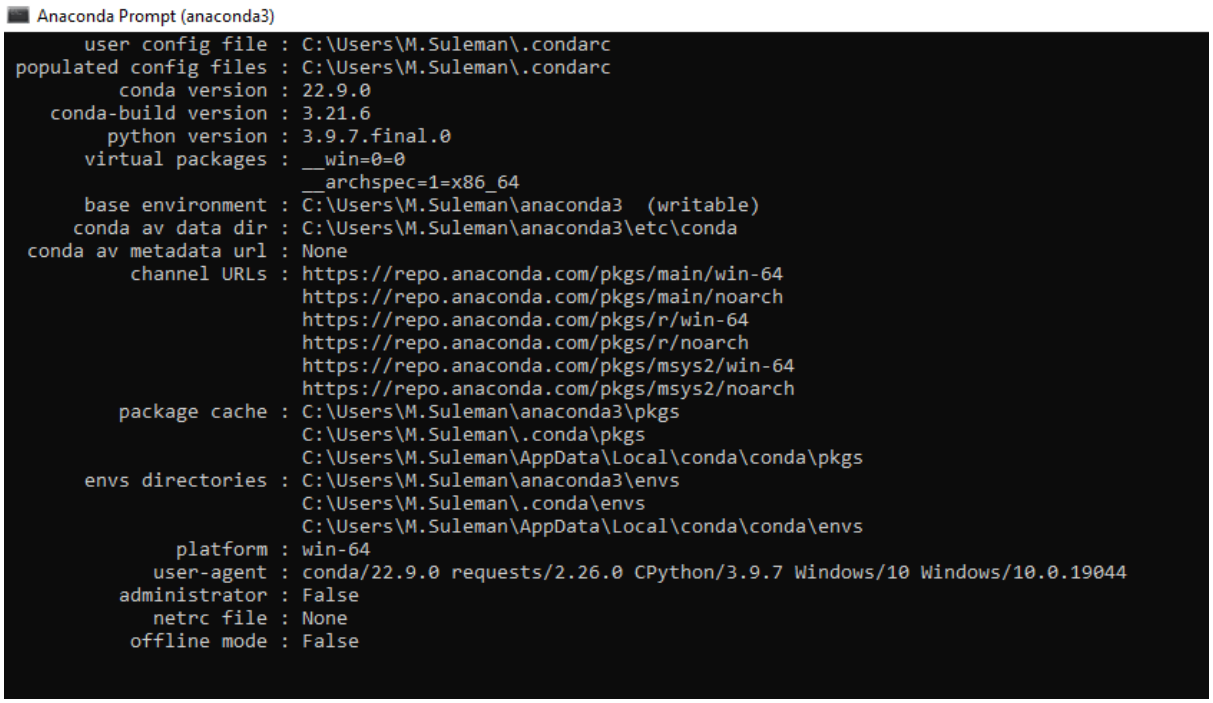
to run the Python shell. If Anaconda is installed and working, the version information it displays when it starts up will include "Continuum Analytics, Inc.". Use the command

>> quit()

to exit the Python shell.
Read this Cheat sheet to learn how to use the "conda" command.

1.1 Task 1

In your terminal, run >> conda info



2 Interactive Terminal (IPython/Jupyter)

IPython/Jupyter is an interactive computational environment in which you can combine code execution, rich text, mathematics, plots, and rich media. Follow this IPython Tutorial and Jupyter Documentation to get up and running on IPython/Jupyter. For more on IPython/Jupyter, check out this great Gallery of Jupyter Notebooks.

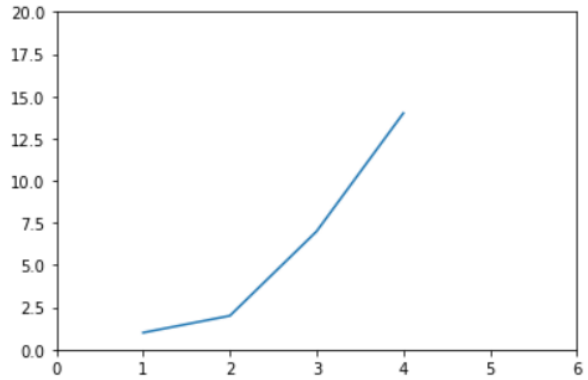
3 Plotting (MatPlotLib/PyPlot)

Matplotlib is the main plotting library for Python and is capable of very powerful publication quality graphics. Check out this Matplotlib Gallery if you would like to learn more about plotting using Matplotlib. Pyplot is a library within Matplotlib that is there to ease the transition from MATLAB to Python. It has a collection of MATLAB-like functions that makes plotting in Python as easy as in MATLAB. Please read through the following Pyplot Tutorial.

3.1 Task 2

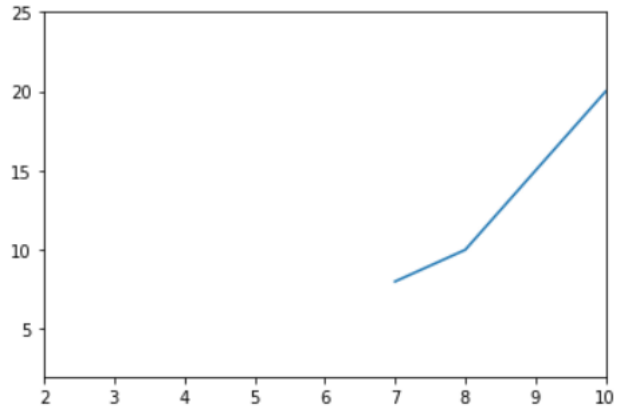
Run the following script in IPython and paste the figure created by the script into your report
import matplotlib.pyplot as plt plt.plot([1,2,3,4], [1,2,7,14])
plt.axis([0, 6, 0, 20]) plt.show()

```
In [9]: import matplotlib.pyplot as plt
        plt.plot([1,2,3,4], [1,2,7,14])
        plt.axis([0, 6, 0, 20])
        plt.show()
```



3.2 Task 3

```
In [11]: plt.plot([10,9,8,7],[20,15,10,8])
         plt.axis([2,10,2,25])
         plt.show()
```

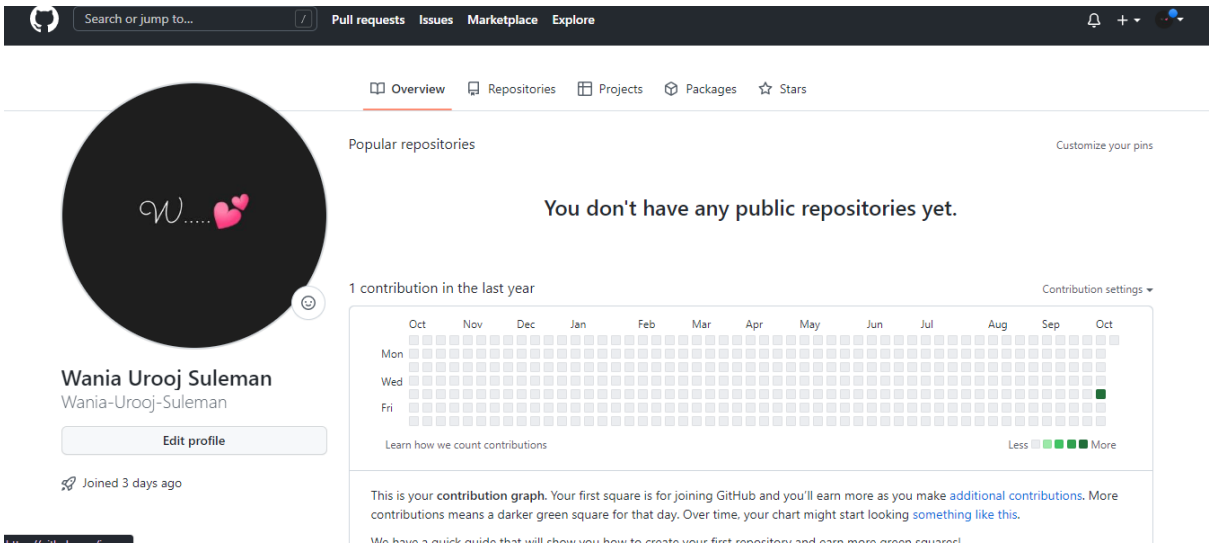


4 Version Control System (BitBucket/GitHub

When you are working on a big project with your team, managing the changes in your code will be challenging. Version control systems (VCS) like Git help streamline this process. Read this article on why VCS is necessary. Bitbucket and Github are two commonly used web-based hosting services for projects that use Git version control systems. In this course, you will use Github.

1.1 4.1 Task 4

Register for a student account here for free private repository access for future projects and go through these tutorials. insert a screenshot of your user page in github in your report.



6. Observations:

After conducting this lab, I have learnt about Jupyter notebook and learnt how to create GitHub account also leant about plot function in Jupyter that how to plot a figure and display in it.

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE) , but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab 2._____06/10/2022

1. Objectives:

After completing this lab, Students will be able to;

- Pandas DataFrame
- NumPy

2. Corresponding CLO and PLO:

- CLO 2, PLO 5

3. Theory:

Pandas

Pandas is one of the most basic and easiest libraries to practice in ML. It is usually used for Python.

Pandas handle data manipulation and other data-related functions. It provides fast and efficient data structures. These data structures make structured and time-series data very easy to work with. Its goal is to become the most advanced data manipulation tool in the world.

Numpy

Numpy is a Machine Learning tool used in scientific calculations. More advanced libraries like Tensorflow and Theano run on NumPy.

For math-based calculations, this library comes in hand. It has an N-dimensional array, other sophisticated tools for data calculation.

Numpy is a Python-based tool. It is multi-dimensional storage for generic data. Hence, we use it in various databases as well.

4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

5. Procedure

Colabs

Machine Learning Crash Course uses Colaboratories (Colabs) for all programming exercises. Colab is Google's implementation of Jupyter Notebook. For more information about Colabs and how to use them, go to Welcome to Colaboratory.

Pandas DataFrame UltraQuick Tutorial

This Colab introduces **DataFrames**, which are the central data structure in the pandas API. This Colab is not a comprehensive DataFrames tutorial. Rather, this Colab provides a very quick introduction to the parts of DataFrames required to do the other Colab exercises in Machine Learning Crash Course.

A DataFrame is similar to an in-memory spreadsheet. Like a spreadsheet:

- A DataFrame stores data in cells.
- A DataFrame has named columns (usually) and numbered rows.

Import NumPy and pandas modules

Run the following code cell to import the NumPy and pandas modules.

```
import numpy as np
import pandas as pd
```

Creating a DataFrame

The following code cell creates a simple DataFrame containing 10 cells organized as follows:

- **5** rows
- 2 columns, one named temperature and the other named activity

The following code cell instantiates a pd.DataFrame class to generate a DataFrame. The class takes two arguments:

- The rst argument provides the data to populate the 10 cells. The code cell calls np.array to generate the 5x2 NumPy array.
- The second argument identi es the names of the two columns.

```
# Create and populate a 5x2 NumPy array. my_data = np.array([[0, 3], [10, 7], [20,
9], [30, 14], [40, 15]])

# Create a Python list that holds the names of the two columns. my_column_names
= ['temperature', 'activity']

# Create a DataFrame. my_dataframe = pd.DataFrame(data=my_data,
columns=my_column_names)

# Print the entire DataFrame print(my_dataframe)
```

Adding a new column to a DataFrame

You may add a new column to an existing pandas DataFrame just by assigning values to a new column name. For example, the following code creates a third column named adjusted in my_dataframe :

```
# Create a new column named adjusted. my_dataframe["adjusted"] =
my_dataframe["activity"] + 2

# Print the entire DataFrame print(my_dataframe)
```

Specifying a subset of a DataFrame

Pandas provide multiples ways to isolate speci c rows, columns, slices or cells in a DataFrame.

```
print("Rows #0, #1, and #2:") print(my_dataframe.head(3), '\n')

print("Row #2:")
print(my_dataframe.iloc[[2]], '\n')

print("Rows #1, #2, and #3:") print(my_dataframe[1:4], '\n')

print("Column 'temperature':") print(my_dataframe['temperature'])
```

Task 1: Create a DataFrame

Do the following:

1. Create an 3x4 (3 rows x 4 columns) pandas DataFrame in which the columns are named Eleanor , Chidi , Tahani , and Jason . Populate each of the 12 cells in the DataFrame with a random integer between 0 and 100, inclusive.
2. Output the following:
   - the entire DataFrame
   - the value in the cell of row #1 of the Eleanor column
3. Create a fth column named Janet , which is populated with the row-by-row sums of Tahani and Jason .

To complete this task, it helps to know the NumPy basics covered in the NumPy UltraQuick Tutorial.

In [13]:
```
import numpy as np
import pandas as pd
my_column_names = ['Eleanor', 'Chidi', 'Tahani', 'Jason']
my_data = np.random.randint(low=0, high=101, size=(3, 4))

# Create a DataFrame.
df = pd.DataFrame(data=my_data, columns=my_column_names)
print(df)

# Print the value in row #1 of the Eleanor column.
print("\nSecond row of the Eleanor column: %d\n" % df['Eleanor'][1])
# Create a column named Janet whose contents are the sum
# of two other columns.
df['Janet'] = df['Tahani'] + df['Jason']
print(df)
```

```
   Eleanor  Chidi  Tahani  Jason
0       89     17      99     65
1       80     28      94     48
2       92     75      17     52

Second row of the Eleanor column: 80

   Eleanor  Chidi  Tahani  Jason  Janet
0       89     17      99     65    164
1       80     28      94     48    142
2       92     75      17     52     69
```

Copying a DataFrame (optional)

Pandas provides two different ways to duplicate a DataFrame:

- **Referencing.** If you assign a DataFrame to a new variable, any change to the DataFrame or to the new variable will be re ected in the other.
- **Copying.** If you call the pd.DataFrame.copy method, you create a true independent copy. Changes to the original DataFrame or to the copy will not be re ected in the other.

The difference is subtle, but important.

Create a reference by assigning my_dataframe to a new variable.
print("Experiment with a reference:") reference_to_df = df

# Print the starting value of a particular cell.
print(" Starting value of df: %d" % df['Jason'][1]) print(" Starting value of reference_to_df: %d\n" % reference_to_df['Jason'][1])

Modify a cell in df.
df.at[1, 'Jason'] = df['Jason'][1] + 5 print(" Updated df: %d" % df['Jason'][1]) print(" Updated reference_to_df: %d\n\n" % reference_to_df['Jason'][1])

Create a true copy of my_dataframe print("Experiment with a true copy:") copy_of_my_dataframe = my_dataframe.copy()

```
# Print the starting value of a particular cell.
print("  Starting value of my_dataframe: %d" % my_dataframe['activity'][1])

print(" Starting value of copy_of_my_dataframe: %d\n" % copy_of_my_dataframe['activity'][

# Modify a cell in df.
my_dataframe.at[1, 'activity'] = my_dataframe['activity'][1] + 3 print("  Updated
my_dataframe: %d" % my_dataframe['activity'][1])
int("  copy_of_my_dataframe does not get updated: %d" % copy_of_my_dataframe['activity']
```

Colabs

Machine Learning Crash Course uses Colaboratories (Colabs) for all programming exercises. Colab is Google's implementation of Jupyter Notebook. For more information about Colabs and how to use them, go to Welcome to Colaboratory.

2. NumPy UltraQuick Tutorial

Numpy is a Python library for creating and manipulating matrices, the main data structure used by ML algorithms. Matrices are mathematical objects used to store values in rows and columns.

Python calls matrices *lists*, NumPy calls them *arrays* and TensorFlow calls them *tensors*. Python represents matrices with the list data type.

This Colab is not an exhaustive tutorial on NumPy. Rather, this Colab teaches you just enough to use NumPy in the Colab exercises of Machine Learning Crash Course.

Import NumPy module

Run the following code cell to import the

NumPy module: `import numpy as np`

Populate arrays with speci c numbers

Call `np.array` to create a NumPy array with your own hand-picked values. For example, the following call to `np.array` creates an 8-element array:

```
one_dimensional_array = np.array([1.2, 2.4, 3.5, 4.7, 6.1, 7.2, 8.3, 9.5]) print(one_dimensional_array)
```

You can also use `np.array` to create a two-dimensional array. To create a two-dimensional array specify an extra layer of square brackets. For example, the following call creates a 3x2 array:

```
two_dimensional_array = np.array([[6, 5], [11, 7], [4, 8]]) print(two_dimensional_array)
```

To populate an array with all zeroes, call `np.zeros` . To populate an array with all ones, call `np.ones` .

Populate arrays with sequences of numbers

You can populate an array with a sequence of numbers:

```
sequence_of_integers = np.arange(5, 12) print(sequence_of_integers)
```

Notice that `np.arange` generates a sequence that includes the lower bound (5) but not the upper bound (12).

Populate arrays with random numbers

NumPy provides various functions to populate arrays with random numbers across certain ranges. For example, `np.random.randint` generates random integers between a low and high value. The following call populates a 6-element array with random integers between 50 and 100.

```
random_integers_between_50_and_100 = np.random.randint(low=50, high=101, size=(6))
print(random_integers_between_50_and_100)
```

Note that the highest generated integer `np.random.randint` is one less than the `high` argument.

To create random oating-point values between 0.0 and 1.0, call `np.random.random` . For example:

```
random_floats_between_0_and_1 = np.random.random([6]) print(random_floats_between_0_and_1)
```

Mathematical Operations on NumPy Operands

If you want to add or subtract two arrays, linear algebra requires that the two operands have the same dimensions. Furthermore, if you want to multiply two arrays, linear algebra imposes strict rules on the dimensional compatibility of operands. Fortunately, NumPy uses a trick called **broadcasting** to virtually expand the smaller operand to dimensions compatible for linear algebra. For example, the following operation uses broadcasting to add 2.0 to the value of every item in the array created in the previous code cell:

```
random_floats_between_2_and_3 = random_floats_between_0_and_1 + 2.0
print(random_floats_between_2_and_3)
```

The following operation also relies on broadcasting to multiply each cell in an array by 3:

```
random_integers_between_150_and_300 = random_integers_between_50_and_100 * 3
print(random_integers_between_150_and_300)
```

Task 1: Create a Linear Dataset

Your goal is to create a simple dataset consisting of a single feature and a label as follows:

1. Assign a sequence of integers from 6 to 20 (inclusive) to a NumPy array named `feature` .

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_openml
```

In [8]:
```
one_dimensional_array = np.array([1.3, 2.6, 3.6, 4.8, 6.2, 7.2, 8.3, 9.5])
print(one_dimensional_array)
```

```
[1.3 2.6 3.6 4.8 6.2 7.2 8.3 9.5]
```

In [7]:
```
two_dimensional_array = np.array([[7, 4], [12, 8], [3, 7]])
print(two_dimensional_array)
```

```
[[ 7  4]
 [12  8]
 [ 3  7]]
```

In [6]:
```
sequence_of_integers = np.arange(10, 15)
print(sequence_of_integers)
```

```
[10 11 12 13 14]
```

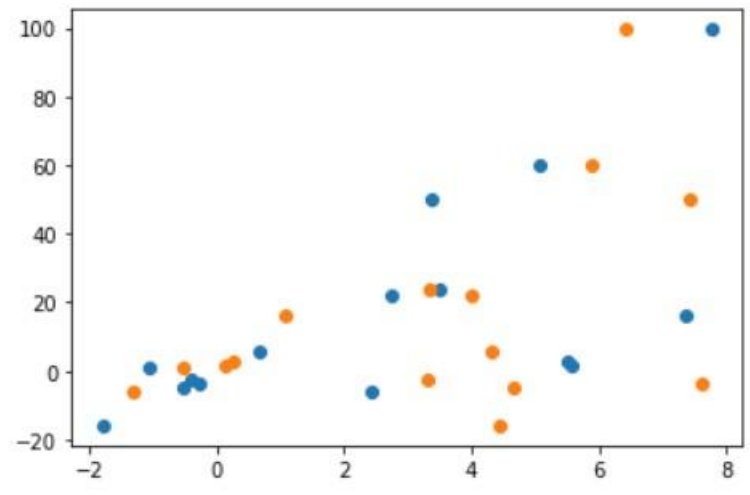
In [14]:
```
random_integers_between_50_and_100 = np.random.randint(low=50, high=101, size=
print(random_integers_between_50_and_100)
```

```
[79 81 96 55 70 89]
```

Task 2: Add Some Noise to the Dataset

To make your dataset a little more realistic, insert a little random noise into each element of the label array you already created. To be more precise, modify each value assigned to label by adding a *different* random oating-point value between -2 and +2.

```
In [15]: random_integers_between_150_and_300 = random_integers_between_50_and_100 * 3
         print(random_integers_between_150_and_300)

         [237 243 288 165 210 267]

In [16]: feature = np.arange(6, 21)
         print(feature)
         label = (feature * 2) + 6
         print(label)

         [ 6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
         [18 20 22 24 26 28 30 32 34 36 38 40 42 44 46]

In [17]: noise = (np.random.random([30]) * 4) -2
         print(noise)
         label = label + noise
         print(label)

         [ 1.77836363 -0.25288041 -1.96087943  1.7589272   1.69854343  0.98695701
          -0.34130971 -1.76438339 -1.75744121 -1.379007    0.64910959 -1.79453791
          -0.44590983  1.86204413 -0.39735324]
         [19.77836363 19.74711959 20.03912057 25.7589272  27.69854343 28.98695701
          29.65869029 30.23561661 32.24255879 34.620993   38.64910959 38.20546209
          41.55409017 45.86204413 45.60264676]

In [22]: dataset = (np.random.random([15]) * 10) -2
         print(dataset)
         label = label + dataset
         print(label)

         [ 0.03317864  4.90482399 -1.48229062 -1.39915689  7.1387404   3.29648513
         [ 22.10812495   3.03546028   1.09435407   5.7908618    1.53511695
           99.58911636 -15.89579423  16.03530114  49.90073861  -3.8979923
           -6.26183638  23.50271161  -2.53287354  -4.70920135  59.89003292]
```



6. Observations:

After conducting this lab, I have learnt about data frames and numpy function and how these functions works. And perform some tasks of data frames.

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE) , but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 3_____13/10/2022

1. Objectives:

After completing this lab, Students will be able to;

- Implement Linear Regression
- Import Dataset, Reading and understanding the data.
- Visualizing the data
- Data Preparation
- Splitting the data into training and test sets
- Building a linear model
- Residual analysis of the train data
- Making predictions using the final model and evaluation

2. Corresponding CLO and PLO:

- CLO 1, PLO 3

3. Theory:

Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

Before attempting to fit a linear model to observed data, a modeler should first determine whether or not there is a relationship between the variables of interest. This does not necessarily imply that one variable *causes* the other (for example, higher SAT scores do not *cause* higher college grades), but that there is some significant association between the two variables. A scatterplot can be a helpful tool in determining the strength of the relationship between two variables. If there appears to be no association between the proposed explanatory and dependent variables (i.e., the scatterplot does not indicate any increasing or decreasing trends), then fitting a linear regression model to the data probably will not provide a useful model. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables.

4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

5. Procedure:

The goal of this Lab is to build a linear regression model. The following notebooks gives guideline to implements linear regression model.
https://github.com/Saniya-Ashraf/saniya-ashraf.github.io/blob/master/ML/Linear%20Regression.ipynb

In [25]:
```
#training and testing model

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
lin_model = LinearRegression()
lin_model.fit(x_train, y_train)
```

Out[25]:
```
LinearRegression()
```

In [29]:
```
from sklearn.metrics import r2_score
# model evaluation for training set
y_train_predict = lin_model.predict(x_train)
testPred = lin_model.predict(x_test)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = r2_score(y_train, y_train_predict)
print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
# model evaluation for testing set
y_test_predict = lin_model.predict(x_test)
rmse = (mean_squared_error(y_test, y_test_predict))
r2 = r2_score(y_test, y_test_predict)
print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
```

```
The model performance for testing set
--------------------------------------
RMSE is 345.6270646575515
R2 score is 0.05156354868851443
```

In [13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_openml
```

In [23]:
```
features = (np.random.random([30, 2])* 20) - 10
```

In [22]:
```
x = pd.DataFrame(features, columns = ['f1','f2'])
y = label
print(x.head())
#print(y.dtypes)
```

```
         f1        f2
0 -4.223295  5.858741
1  4.807735 -5.502450
2 -3.288513  1.114602
3  4.081084 -6.487159
4 -2.578313 -1.301211
```

In [24]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(22, 2)
(8, 2)
(22, )
```

In [32]:
```
from sklearn.datasets import load_boston
from sklearn.datasets import fetch_openml

housing = fetch_openml(name="house_prices", as_frame=True)
```

In [33]:
```
print(housing.keys())
```

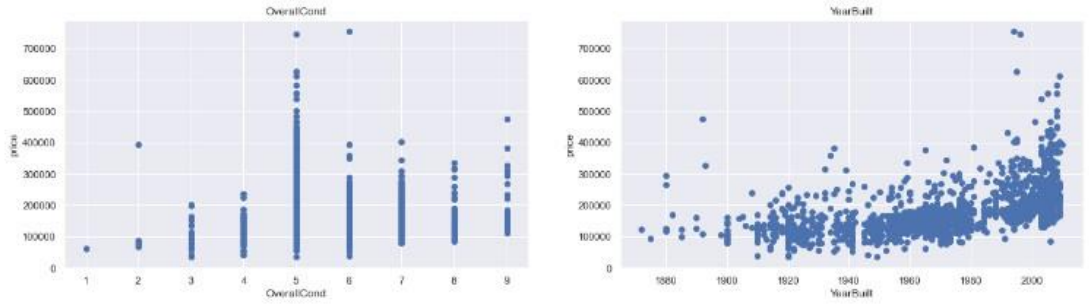
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_
names', 'DESCR', 'details', 'url'])
```

In [34]:
```
housing.target
```

Out[34]:
```
0       208500.0
1       181500.0
2       223500.0
3       140000.0
4       250000.0
          ...
1455    175000.0
1456    210000.0
1457    266500.0
1458    142125.0
1459    147500.0
Name: SalePrice, Length: 1460, dtype: float64
```

In [42]:

```
# OverallCond','YearBuilt'

#lets plot these

plt.figure(figsize=(20, 5))

features = ['OverallCond','YearBuilt']
target = housing_df['price']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    a = housing_df[col]
    b = target
    plt.scatter(a, b, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('price')
```



6. Observations:

In this lab, I have learnt about linear regression and regressions and how machine learning works with linear regression and how supervised learning predicts models for linear regression.

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE) , but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 4_____20/10/2022

1. Objectives:

After completing this lab, Students will be able to;

- Classification with KNN
- Exploring dataset

2. Corresponding CLO and PLO:

- CLO 1, PLO 3

3. Theory:

K-Nearest Neighbor(KNN):

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

5. Procedure:

Fruit Dataset:
https://github.com/Saniya-Ashraf/saniya-ashraf.github.io/blob/master/ML/fruit_data_with_colors%20(1).txt

```
In [1]:
# for this lab we use the breast cancer dataset

import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
import seaborn as sns
cancer = load_breast_cancer()

#print(cancer.DESCR) # Print the data set description
```

```
In [2]:
cancer.keys()
```

```
Out[2]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

```
In [3]:
len(cancer['feature_names'])
```

```
Out[3]: 30
```

Question 0 (Example)
How many features does the breast cancer dataset have?

I

```
In [4]:
cancer.feature_names
```

```
Out[4]: array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Question 1
Scikit-learn works with lists, numpy arrays, scipy-sparse matrices, and pandas DataFrames, so converting the dataset to a DataFrame is not necessary for training this model. Using a DataFrame does however help make many things easier such as munging data, so let's practice creating a classifier with a pandas DataFrame.

Convert the sklearn.dataset cancer to a DataFrame.

```
In [5]: cancer.target_names

Out[5]: array(['malignant', 'benign'], dtype='<U9')

In [6]: cancer.target

Out[6]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
               0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
               1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
               1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
               1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
               0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
               1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
               0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
               1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
               1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
               0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
               0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
               1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
               1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
               1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
               1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
               1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])

In [7]:
```

Question 2

What is the class distribution? (i.e. how many instances of malignant (encoded 0) and how many benign (encoded 1)?)

```
In [7]: type(cancer)

Out[7]: sklearn.utils.Bunch

In [8]: cancerdf=pd.DataFrame(np.c_[cancer.data, cancer.target], columns=[list(cancer.feature_names)+['target']])
        cancerdf.head()
```

Out[8]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000

5 rows × 31 columns

```
In [9]: X=cancerdf.iloc[:,0:-1]
        #X=cancerdf.drop('target', axis=1)
        y=cancerdf.iloc[:,-1]   # all rows of last columns
        y.tail()

Out[9]: 564    0.0
        565    0.0
        566    0.0
        567    0.0
        568    1.0
        Name: (target,), dtype: float64
```

Question 3

Split the DataFrame into X (the data) and y (the labels).

In [10]:
```
print("X_train",X.shape)
print("y_train",y.shape)
```

```
X_train (569, 30)
y_train (569,)
```

Question 4

Using train_test_split, split X and y into training and test sets (X_train, X_test, y_train, and y_test).

Set the random number generator state to 0 using random_state=0 to make sure your results match the autograder!

This function should return a tuple of length 4: (X_train, X_test, y_train, y_test), *where*

- X_train *has shape* (426, 30)
- X_test *has shape* (143, 30)
- y_train *has shape* (426,)
- y_test *has shape* (143,)

In [11]:
```
from sklearn.model_selection import train_test_split
```

In [12]:
```
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.2, random_state=0)
```

In [13]:
```
print("X_train",X_train.shape)
print("y_train",y_train.shape)
print("X_test",X_test.shape)
print("y_test",y_test.shape)
```

```
X_train (455, 30)
y_train (455,)
X_test (114, 30)
y_test (114,)
```

In [14]:
```
y_test
```

Out[14]:
```
512    0.0
457    1.0
439    1.0
298    1.0
37     1.0
      ...
213    0.0
519    1.0
432    0.0
516    0.0
500    1.0
Name: (target,), Length: 114, dtype: float64
```

Question 5

Using KNeighborsClassifier, fit a k-nearest neighbors (knn) classifier with X_train, y_train and using one nearest neighbor (n_neighbors = 1).

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
kn=KNeighborsClassifier(n_neighbors=1)
```

In [16]:
```
kn.fit(X_train,y_train)
```

Out[16]:
```
KNeighborsClassifier(n_neighbors=1)
```

Question 6

Using your knn classifier, predict the class label using the mean value for each feature.

In [17]:
```
cancerdf.mean()[:-1].values.reshape(1, -1)
```

Out[17]:
```
array([[1.41272917e+01, 1.92896485e+01, 9.19690334e+01, 6.54889104e+02,
        9.63602812e-02, 1.04340984e-01, 8.87993158e-02, 4.89191459e-02,
        1.81161863e-01, 6.27976098e-02, 4.05172056e-01, 1.21685343e+00,
        2.86605923e+00, 4.03370791e+01, 7.04097891e-03, 2.54781388e-02,
        3.18937163e-02, 1.17961371e-02, 2.05422988e-02, 3.79490387e-03,
        1.62691898e+01, 2.56772232e+01, 1.07261213e+02, 8.80583128e+02,
        1.32368594e-01, 2.54265044e-01, 2.72188483e-01, 1.14606223e-01,
        2.90075571e-01, 8.39458172e-02]])
```

Question 7

In [18]:
```
kn.predict(X_test)
```

Out[18]:
```
array([1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1.,
       0., 0., 0., 0., 1., 1., 1., 0., 1., 1., 1., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 0., 1., 0., 1., 0., 0., 1., 1., 1., 0., 0.,
       1., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 0.,
       0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 0., 0., 1., 0.,
       1., 1., 1., 0., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 0., 1., 1., 0., 0., 1.])
```

In [19]:
```
X_test.shape
```

Out[19]:
```
(114, 30)
```

Question 8

Find the score (mean accuracy) of your knn classifier using X_test and y_test.

Score (accuracy)

In [20]:
```
kn.score(X_test,y_test)
```

Out[20]: 0.9122807017543859

New Patient Data

In [21]:
```
patient1 = [17.99,
 10.38,
 122.8,
 1001.0,
 0.1184,
 0.2776,
 0.3001,
 0.1471,
 0.2419,
 0.07871,
 1.095,
 0.9053,
 8.589,
 153.4,
 0.006399,
 0.04904,
 0.05373,
 0.01587,
 0.03003,
 0.006193,
 25.38,
 17.33,
 184.6,
 2019.0,
 0.1622,
 0.6656,
 0.7119,
 0.2654,
 0.4601,
 0.1189]
```

In [22]:
```
patient1 = np.array([patient1])
```

In [ ]:

Optional plot

Try using the plotting function below to visualize the differet predicition scores between training and test sets, as well as malignant and benign cells.

In [23]:
```
type(patient1)
```

Out[23]: numpy.ndarray

In [24]:
```
pred=kn.predict(patient1)
pred
```

Out[24]: array([0.])

In [25]:
```
if pred[0] == 0:
  print('Patient has Cancer (malignant tumor)')
else:
  print('Patient has no Cancer (benign tumor)')
```

Patient has Cancer (malignant tumor)

In [26]:
```
#sns.distplot(X_test,y_test, bins=20)
#plt.show()
```

In [27]:
```
def accuracy_plot():
    import matplotlib.pyplot as plt
    %matplotlib notebook
```

In [28]:
```
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.2, random_state=0)
```

In [29]:
```
 # Find the training and testing accuracies by target value (i.e. malignant, benign)
train_X = X_train[y_train==0]
train_y = y_train[y_train==0]
train_X = X_train[y_train==1]
train_y = y_train[y_train==1]
```

In [32]:
```
test_X = X_test[y_test==0]
test_y = y_test[y_test==0]
test_X = X_test[y_test==1]
test_y = y_test[y_test==1]

knn =kn.fit(X_train,y_train)

scores = [knn.score(mal_train_X, mal_train_y), knn.score(ben_train_X, ben_train_y), knn.score(mal_test_X, mal_test_y), knn.score(ben_test_X, ben_test_


plt.figure()

    # Plot the scores as a bar chart
bars = plt.bar(np.arange(4), scores, color=['#4c72b0','#4c72b0','#55a868','#55a868'])

    # directly label the score onto the bars
for bar in bars:
height = bar.get_height()
plt.gca().text(bar.get_x() + bar.get_width()/2, height*.90, '{0:.{1}f}'.format(height, 2), ha='center', color='w', fontsize=11)

    # remove all the ticks (both axes), and tick labels on the Y axis
plt.tick_params(top='off', bottom='off', left='off', right='off', labelleft='off', labelbottom='on')

    # remove the frame of the chart
for spine in plt.gca().spines.values():
spine.set_visible(False)

plt.xticks([0,1,2,3], ['Malignant\nTraining', 'Benign\nTraining', 'Malignant\nTest', 'Benign\nTest'], alpha=0.8);
plt.title('Training and Test Accuracies for Malignant and Benign Cells', alpha=0.8)
```

6. Observations:

After completing this lab, I have learnt about Classification with KNN and Exploring dataset with the records of hospital to determine breast cancers.

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE) , but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 5 and 6_____27/10/2022

1. Objectives:

After completing this lab, Students will be able to;

- Implementation of KNN

2. Corresponding CLO and PLO:

- CLO 1, PLO 3
- CLO 2, PLO 5

3. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

4. Procedure:

- Importing required modules and Loading Data.
- Examining the data
- Creating Train-Test split
- Create classifier Object
- Train the classifier (fit the estimator) using the training data
- Estimate the accuracy of the classifier on future data, using the test data
- Use the trained k-NN classifier model to classify new, previously unseen objects
- Plot the decision boundaries of the k-NN classifier
- How sensitive is k-NN classification accuracy to the choice of the 'k' parameter?
- How sensitive is k-NN classification accuracy to the train/test split proportion?

Click the following link of download the notebook.
https://github.com/Saniya-Ashraf/saniya-ashraf.github.io/raw/master/ML/Classification%20with%20KNN%20(2).zip

Lab 6 Task:

Out[2]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31

In [3]:
```
df=dataframe.drop(["cut","color","clarity", "Unnamed: 0"], axis=1)
df.head()
```

Out[3]:

	carat	depth	table	price	x	y	z
0	0.23	61.5	55.0	326	3.95	3.98	2.43
1	0.21	59.8	61.0	326	3.89	3.84	2.31
2	0.23	56.9	65.0	327	4.05	4.07	2.31
3	0.29	62.4	58.0	334	4.20	4.23	2.63
4	0.31	63.3	58.0	335	4.34	4.35	2.75

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
carat    0
depth    0
table    0
price    0
x        0
y        0
z        0
dtype: int64
```

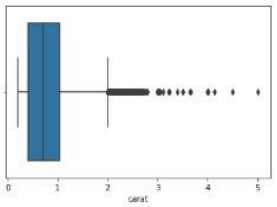
In [5]:
```
plt.figure(figsize=(16,9))
sn.heatmap(df.corr(), annot=True)
```

Out[5]: <AxesSubplot:>

In [6]:
```
sn.boxplot(df["carat"])
```

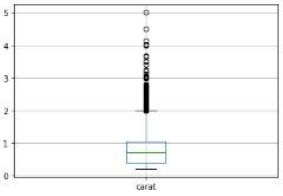
```
/home/saeedullah/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[6]: <AxesSubplot:xlabel='carat'>

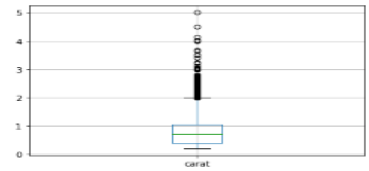


In [7]:
```
df.boxplot("carat")
```

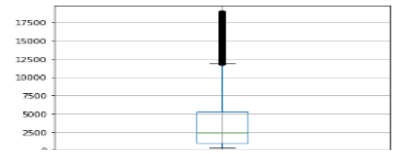
Out[7]: <AxesSubplot:>



```
In [7]: df.boxplot("carat")
Out[7]: <AxesSubplot:>
```




```
In [8]: df.boxplot("price")
Out[8]: <AxesSubplot:>
```



KNN Classifier

```
In [ ]: from sklearn.neighbors import KNeighborsClassifier
knnc=KNeighborsClassifier()
```

```
In [ ]: knnc.fit(X_train, y_train)
```

```
In [ ]: knnc.score(X_train, y_train)
```

```
In [ ]: knnc.score(X_test, y_test)
```

LogisticRegression()

```
In [ ]: from sklearn.linear_model import LogisticRegression
lg=LogisticRegression()
```

```
In [ ]: lg.fit(X_train, y_train)
```

```
In [ ]: lg.score(X_train,y_train)
```

```
In [ ]: lg.score(X_test, y_test)
```

Linear Regression()

```
In [ ]: from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

```
In [ ]: lr.fit(X_train, y_train)
```

```
In [ ]: lr.intercept_
```

```
In [ ]: lr.coef_
```

```
In [ ]: lr.score(X_train, y_train)
```

```
In [ ]: lr.score(X_test, y_test)
```

Type *Markdown* and LaTeX: α^2

KNN Model() Regression

```
In [19]: from sklearn.neighbors import KNeighborsRegressor
knnr=KNeighborsRegressor()
```

```
In [20]: knnr.fit(X_train, y_train)
Out[20]: KNeighborsRegressor()
```

```
In [21]: knnr.score(X_train,y_train)
Out[21]: 0.9132014106627262
```

```
In [ ]: knnr.score(X_test, y_test)
```

train_test_split()

In [16]:
```
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)
```

In [ ]:
```
#z_train, z_test, y_train, y_test = train_test_split(z,y, test_size=0.2, random_state=0)
```

Standard Scaling()

In [17]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [18]:
```
sc.fit(X_train)
# Standard Scaler()
X_train= sc.transform(X_train)
X_test= sc.transform(X_test)
```

For Carat features:

In [ ]:
```
#z_train=sc.transform(z_train)
#z_test=sc.transform(z_test)
```

In [ ]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
print(z_train.shape)
print(z_test.shape)
```

train_test_split()

In [16]:
```
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)
```

In [ ]:
```
#z_train, z_test, y_train, y_test = train_test_split(z,y, test_size=0.2, random_state=0)
```

Standard Scaling()

In [17]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [18]:
```
sc.fit(X_train)
# Standard Scaler()
X_train= sc.transform(X_train)
X_test= sc.transform(X_test)
```

For Carat features:

In [ ]:
```
#z_train=sc.transform(z_train)
#z_test=sc.transform(z_test)
```

In [ ]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
print(z_train.shape)
print(z_test.shape)
```

5. Observation:

After completing this lab, I have learnt about implementation of KNN on diamond set.

6. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE), but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 7_____03/11/2022

1. Objectives:

After completing this lab, Students will be able to;

- Implement K- Nearest Neighbor Regression

2. Corresponding CLO and PLO:

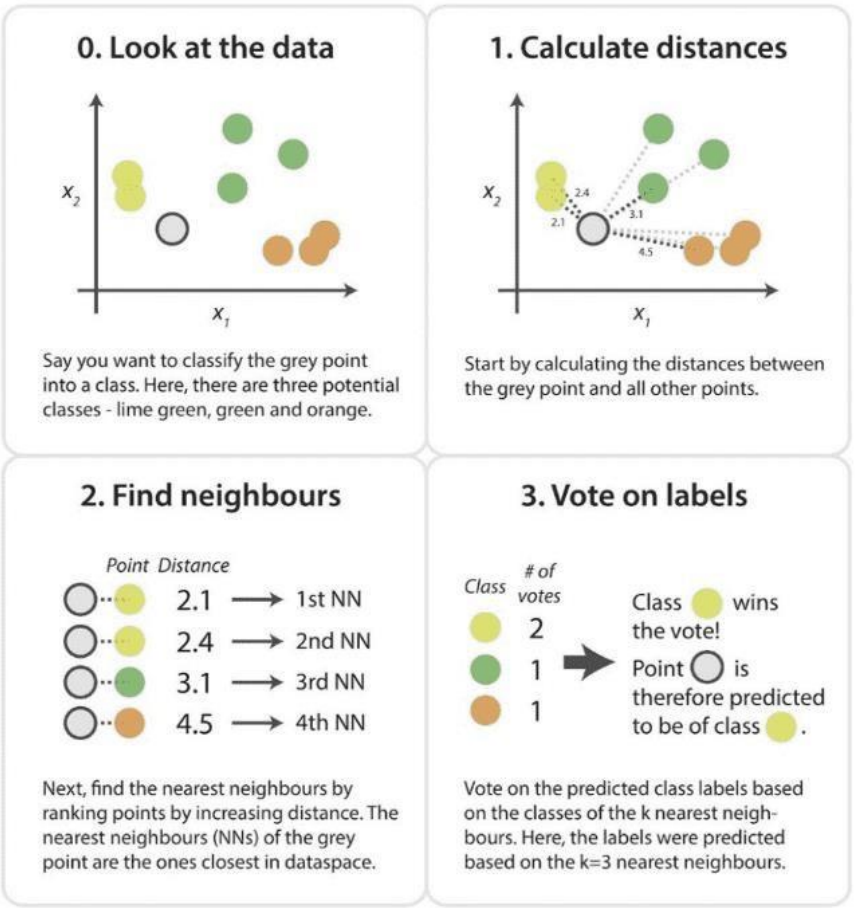
- CLO 2, PLO 5

3. Theory:

Introduction

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closet to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and class holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

Let see the below example to make it a better understanding



4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

5. Procedure:

This notebooks brings the concept of using Regression Analysis in association with KNN.
This algorithm gets the **k** nearest results for that regression and creates an average to get to the final result, making a possibly more robust solution.

```
# Data Manipulation
import pandas as pd
import numpy as np

# DataViz
import seaborn as sns
import matplotlib.pyplot as plt

# Modeling
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
```

Dataset

In [2]:

```
# Load and view dataset
df = sns.load_dataset('diamonds')
df.head()
```

Out[2]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31

	carat	cut	color	clarity	depth	table	price	x	y	z
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [3]:

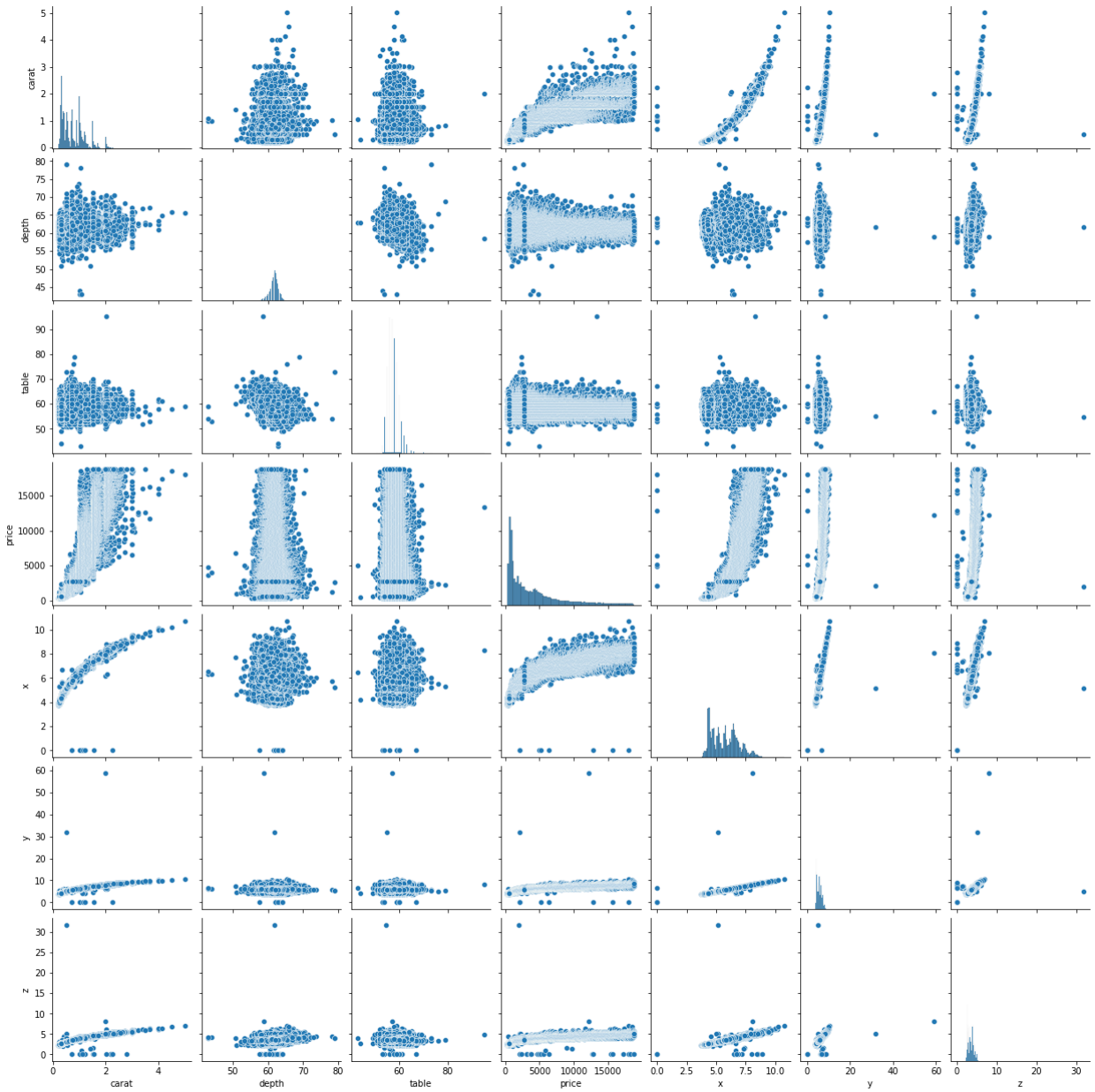
```
# Data shape
print(f'Number of rows: {df.shape[0]} | Columns (variables): {df.shape[1]}')
```

```
Number of rows: 53940 | Columns (variables): 10
```

Data Exploration

In [4]:

```
# Let's see the possible best variables for modeling the carat
sns.pairplot(df);
```



In [5]:

```
df.corr().style.background_gradient(cmap='coolwarm')
```

Out[5]:

	carat	depth	table	price	x	y	z
carat	1.000000	0.028224	0.181618	0.921591	0.975094	0.951722	0.953387
depth	0.028224	1.000000	-0.295779	-0.010647	-0.025289	-0.029341	0.094924
table	0.181618	-0.295779	1.000000	0.127134	0.195344	0.183760	0.150929
price	0.921591	-0.010647	0.127134	1.000000	0.884435	0.865421	0.861249

	carat	depth	table	price	x	y	z
x	0.975094	-0.025289	0.195344	0.884435	1.000000	0.974701	0.970772
y	0.951722	-0.029341	0.183760	0.865421	0.974701	1.000000	0.952006
z	0.953387	0.094924	0.150929	0.861249	0.970772	0.952006	1.000000

Let's use the X, Y and Z information to model *carrat*. As per the pairplot above, we see that Y and Z have an apparent linear relationship with the target variable, while X shows an exponential curve. We will address that.

Missing Values

In [6]:

```
# Checking for missing values
df.isna().sum()
```

Out[6]:

```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

No missing values found. Let's move on.

Train Test Split

In order to prevent *data leakage* (when components of the training dataset leak to the response variable and influences the model fitting, creating a wrong sensation of good fit), we're spliting the data now, before any other formatting.

In [7]:

```
X = df[['x', 'y', 'z']]
y = df.carat
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2,
random_state=12)
```

In [8]:

```
print(f'Train:  {X_train.shape,  y_train.shape}     \nTest: {X_test.shape,
y_test.shape}')
```

```
Train: ((43152, 3), (43152,))
```

```
Test: ((10788, 3), (10788,))
```

Removing Outliers

Linear Regression is very affected by outliers, as it can interfere in the curve slope calculation.

Let's remove those using the 1.5 x IQR rule: everything above or below 1.5 times the inter-quantile range will be removed.

In [9]:

```
# Extract the descriptive statistics for the explanatory variables for IQR calculation
X_train.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
x	43152.0	5.731283	1.122783	0.0	4.71	5.69	6.54	10.74
y	43152.0	5.734739	1.150275	0.0	4.72	5.71	6.54	58.90
z	43152.0	3.538536	0.708813	0.0	2.91	3.53	4.03	31.80

In [10]:

```
# Extracting the quantiles
x_25 = X_train.describe().T.loc['x', '25%']
x_75 = X_train.describe().T.loc['x', '75%']
y_25 = X_train.describe().T.loc['y', '25%']
y_75 = X_train.describe().T.loc['y', '75%']
z_25 = X_train.describe().T.loc['z', '25%']
z_75 = X_train.describe().T.loc['z', '75%']

# Calculate IQRs
IQR_x = 1.5 * (x_75 - x_25)
IQR_y = 1.5 * (y_75 - y_25)
IQR_z = 1.5 * (z_75 - z_25)
```

Remove the Outliers

In [11]:

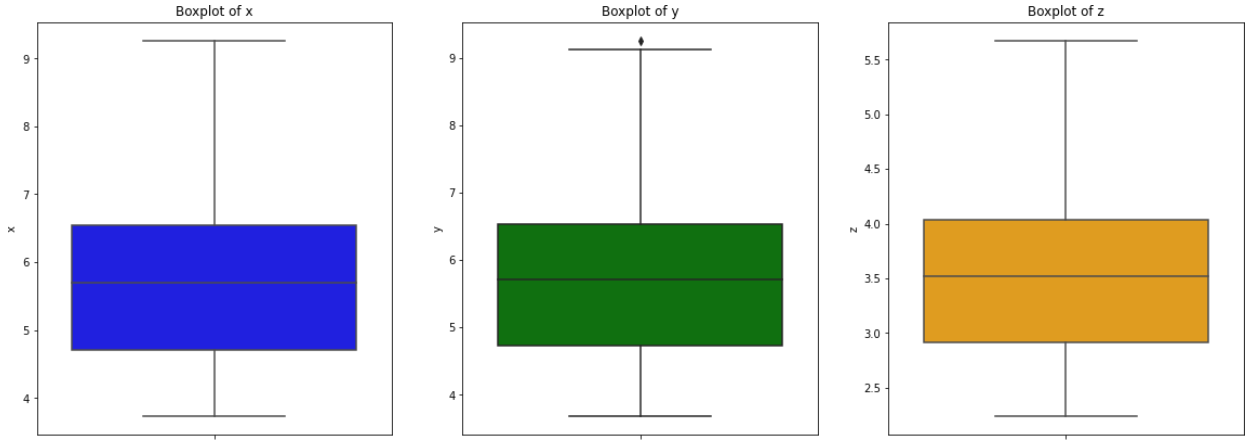
```
# Remove outliers from the variable 'x'
X_train = X_train.query(' x >= (@x_25 - @IQR_x) & x <= (@x_75 + @IQR_x) ')
y_train = y_train[X_train.index]

# Remove outliers from the variable 'y'
X_train = X_train.query(' z >= (@y_25 - @IQR_y) & y <= (@y_75 + @IQR_y) ')
y_train = y_train[X_train.index]
```

```
# Remove outliers from the variable 'z'
X_train = X_train.query(' x >= (@z_25 - @IQR_z) & z <= (@z_75 + @IQR_z) ')
y_train = y_train[X_train.index]
```

Check the results : Clean dataset

In [12]:

```
# setup figure
fig, ax = plt.subplots(1,3, figsize=(20,7))

# plot1
g1 = sns.boxplot(y=X_train.x, color='blue', ax=ax[0])
g1.set_title('Boxplot of x')

# plot2
g2 = sns.boxplot(y=X_train.y, color='green', ax=ax[1])
g2.set_title('Boxplot of y')

# plot3
g3 = sns.boxplot(y=X_train.z, color='orange', ax=ax[2])
g3.set_title('Boxplot of z');
```

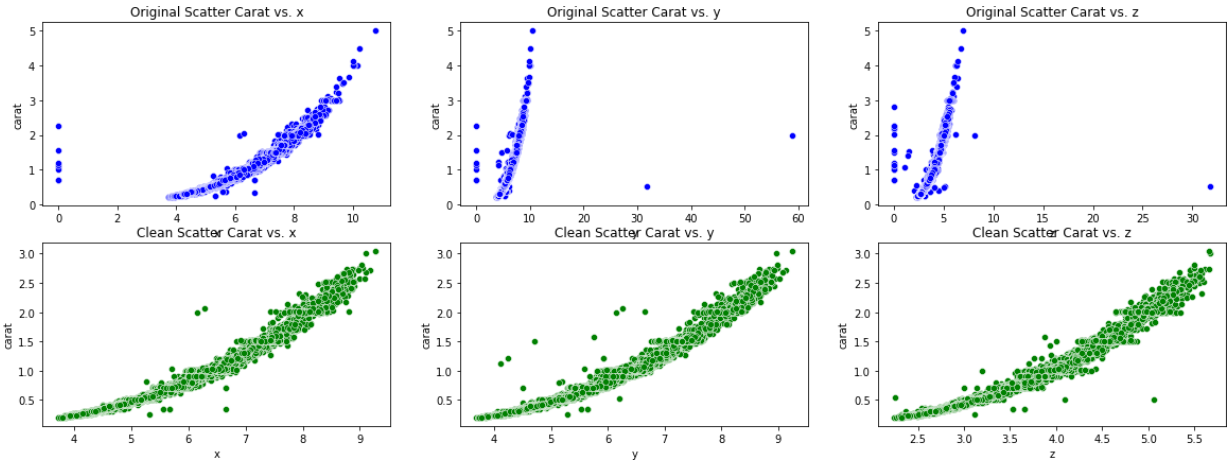


Let's see the scatteplots now that we have removed the outliers

In [13]:

```
# setup figure
fig, ax = plt.subplots(2,3, figsize=(20,7))

for i, var in enumerate(['x', 'y', 'z']):
  # Original data plots
  g= sns.scatterplot(data= df, x=var, y='carat', color='blue', ax=ax[0][i])
  g.set_title(f'Original Scatter Carat vs. {var}')


df_clean = pd.concat([X_train, y_train], axis=1)

for i, var in enumerate(['x', 'y', 'z']):
  # Cleaned data plots
  g= sns.scatterplot(data= df_clean, x=var, y='carat', color='green',
ax=ax[1][i])
```

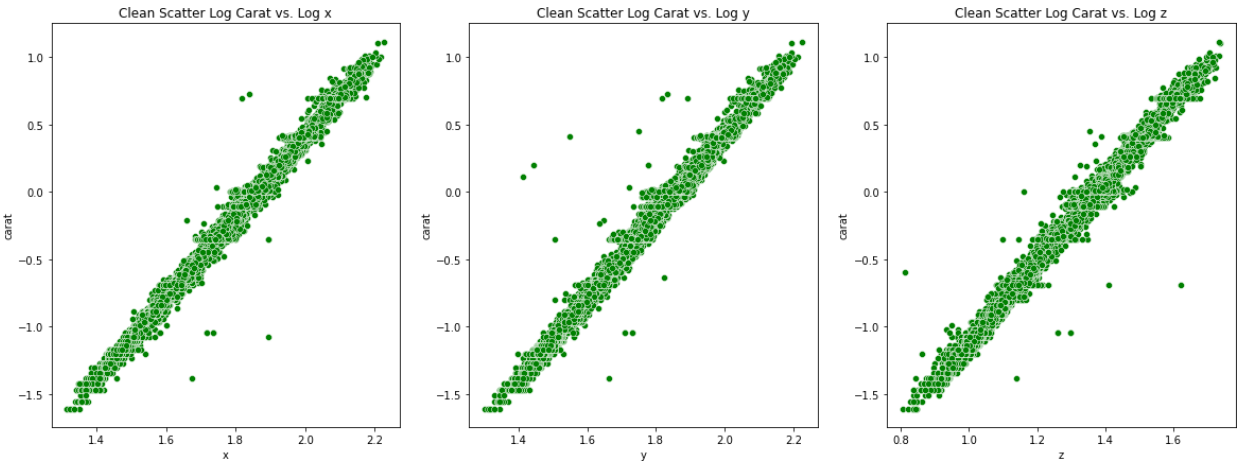
```
  g.set_title(f'Clean Scatter Carat vs. {var}')
```



We see that the relationships are almost linear, but they have a little curve, what's some exponential degree that could be linearized using a log transformation.

In [14]:

```
# setup figure
fig, ax = plt.subplots(1,3, figsize=(20,7))

for i, var in enumerate(['x', 'y', 'z']):
  log_var = np.log(df_clean[var])
  # Cleaned data plots
  g= sns.scatterplot(x=log_var, y= np.log(df_clean.carat), color='green',
ax=ax[i]);
  g.set_title(f'Clean Scatter Log Carat vs. Log {var}');
```



The linear relationship is much better after a log transformation. So let's do that prior to modeling.

In [15]:

```
# Log transformation
X_log = np.log(X_train)
y_log = np.log(y_train)
```

Modeling

Linear Regression

In [27]:

```
# Instance and fit
lr_model = LinearRegression().fit(X_train, y_train)

# Score
score_lr = lr_model.score(X_test, y_test)
score_lr
```

Out[27]:

```
0.9534250731184013
```

Linear Regression with Log Transformation

In [26]:

```
# Instance and fit
lrLog_model = LinearRegression().fit(X_log, y_log)

# Remove zeroes
X_test_log = X_test[(X_test.x > 0) & (X_test.y > 0) & (X_test.z > 0) ]
y_test_log = y_test[X_test_log.index]

# Log Transform X_test and y test
X_test_log = np.log(X_test_log)
y_test_log = np.log(y_test_log)

# Score
score_log = lrLog_model.score(X_test_log, y_test_log)
score_log
```

Out[26]:

```
0.9988479198247474
```

In [18]:

```
# Predictions
preds = lrLog_model.predict(X_test_log)

# Performance
pd.DataFrame({ 'True Value': np.exp(y_test_log),
               'Prediction': np.exp(preds)}).head(5)
```

Out[18]:

	True Value	Prediction
45936	0.51	0.517710
23023	0.35	0.348205

	True Value	Prediction
34325	0.39	0.391145
38578	0.40	0.403460
15979	1.20	1.225848

KNN Regression

In [28]:

```
# Instance and fit
knn_model = KNeighborsRegressor(n_neighbors=5).fit(X_train, y_train)

# Score
score_knn = knn_model.score(X_test, y_test)
score_knn
```

Out[28]:

```
0.9978517407802342
```

In [20]:

```
# Predictions
preds = knn_model.predict(X_test)

# Performance
performance = pd.DataFrame({ 'True Value': y_test,
                             'Prediction': preds,
                             'Error': y_test - preds})
# View
performance
```

Out[20]:

	True Value	Prediction	Error
45936	0.51	0.516	-0.006
23023	0.35	0.348	0.002
34325	0.39	0.396	-0.006
38578	0.40	0.400	0.000

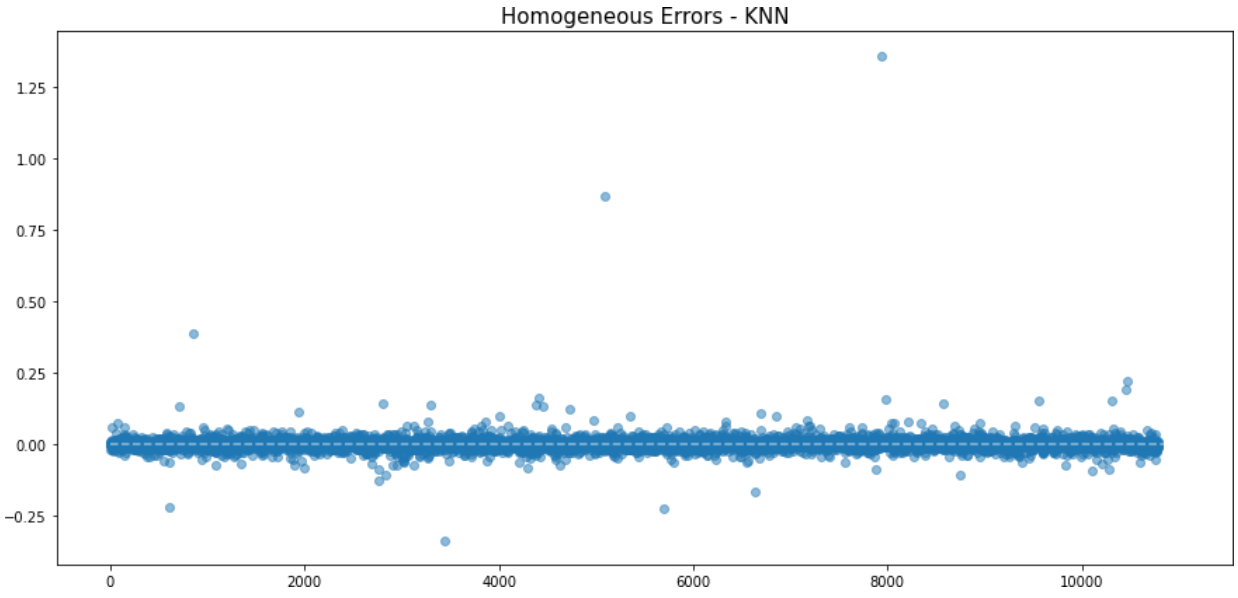
	True Value	Prediction	Error
15979	1.20	1.220	-0.020
...	...	...	...
2106	0.90	0.900	0.000
18207	1.25	1.258	-0.008
37536	0.40	0.406	-0.006
5815	0.73	0.722	0.008
35887	0.32	0.320	0.000

10788 rows × 3 columns

Model Evaluation

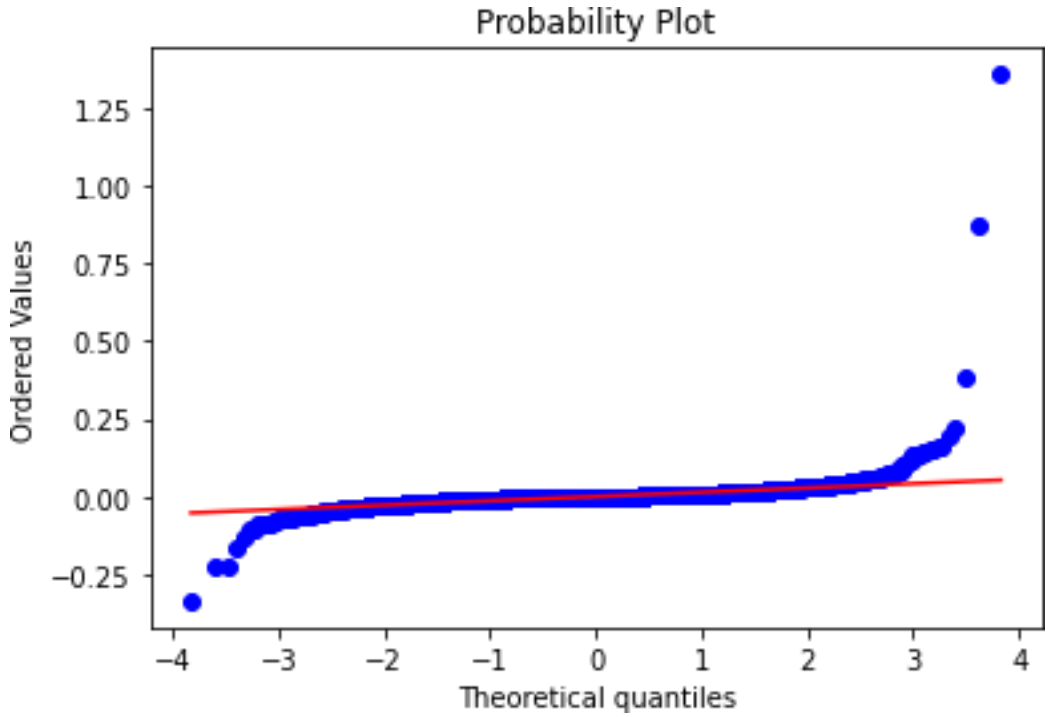
In [32]:

```
plt.figure(figsize=(15,7))
# Errors
ax_x= performance['True Value']
ax_y= performance['Prediction']
yerr= performance['Error']

plt.scatter(range(len(yerr)), yerr, alpha=.5)
plt.title('Homogeneous Errors - KNN', size=15);
plt.hlines(y=0,   xmin=0,   xmax=11000,   linestyle='--',   color='white',
alpha=.5);
#plt.ylim(-.3, .3);
```



In [22]:

```
from scipy.stats import probplot
#QQ Plot
probplot(yerr, dist='norm', plot=plt);
```



The model presents some outliers, but most of the predictions are good. It should be enhanced with a little more clenup in the data, though.

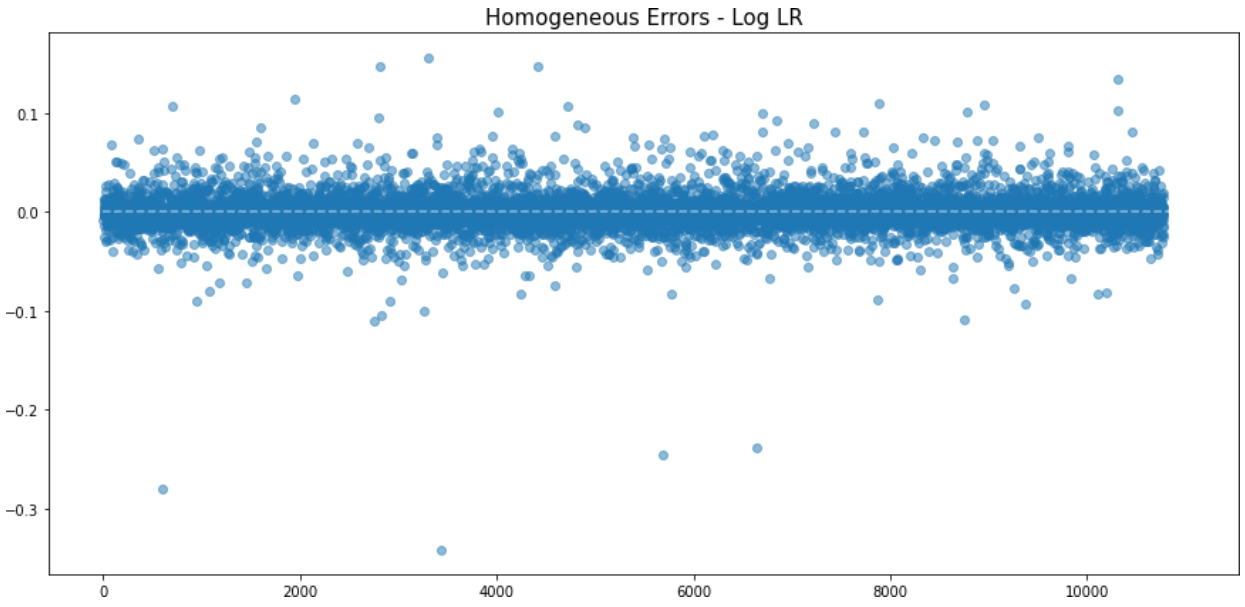
In [23]:

```
# Predictions
preds = lrLog_model.predict(X_test_log)

# Performance
```

```
LR_log_performance = pd.DataFrame({ 'True Value': np.exp(y_test_log),
                                    'Prediction': np.exp(preds),
                                    'Error':       np.exp(y_test_log)       -
np.exp(preds)})
```

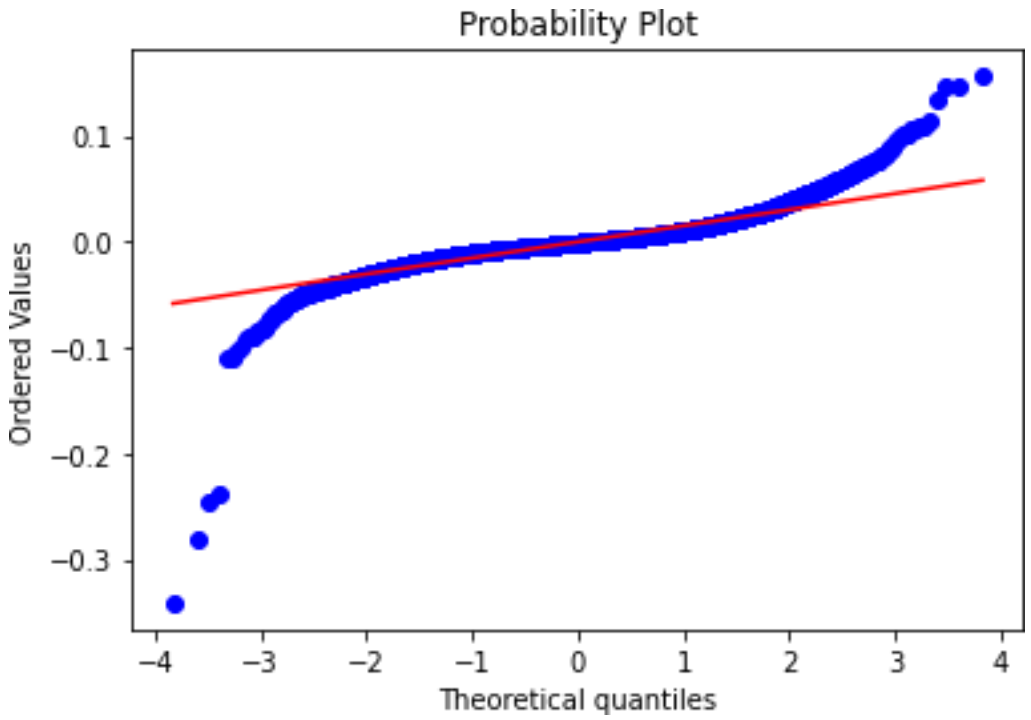
In [33]:

```
plt.figure(figsize=(15,7))
# Errors
ax_x= LR_log_performance['True Value']
ax_y= LR_log_performance['Prediction']
yerr= LR_log_performance['Error']

plt.scatter(range(len(yerr)), yerr, alpha=.5)
plt.title('Homogeneous Errors - Log LR', size=15);
plt.hlines(y=0,    xmin=0,    xmax=11000,    linestyle='--',    color='white',
alpha=.5);
#plt.ylim(-.3, .3);
```



In [25]:

```
# QQ Plot
probplot(yerr, dist='norm', plot=plt);
```



The model with Log transformation also has a couple of wrong predictions too on the lower side, but it looks even better than the KNN model.

Model Comparison

```
pd.DataFrame({'Linear Regression':[score_lr],
              'Log_Linear Regression': [score_log],
              'KNN Regression': [score_knn]})
```

Out[31]:

	Linear Regression	Log_Linear Regression	KNN Regression
0	0.953425	0.998848	0.997852

6. Observations:

After completing this lab, I have learnt about K-Nearest Regression, model evaluation and plotting its graph.

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE) , but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 8_____10/11/2022

1. Objectives:

After completing this lab, Students will be able to;

- **Decision Trees Classification**

2. Corresponding CLO and PLO:

- CLO 1, PLO 3

3. Theory:

Decision Tree Classification:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.

It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

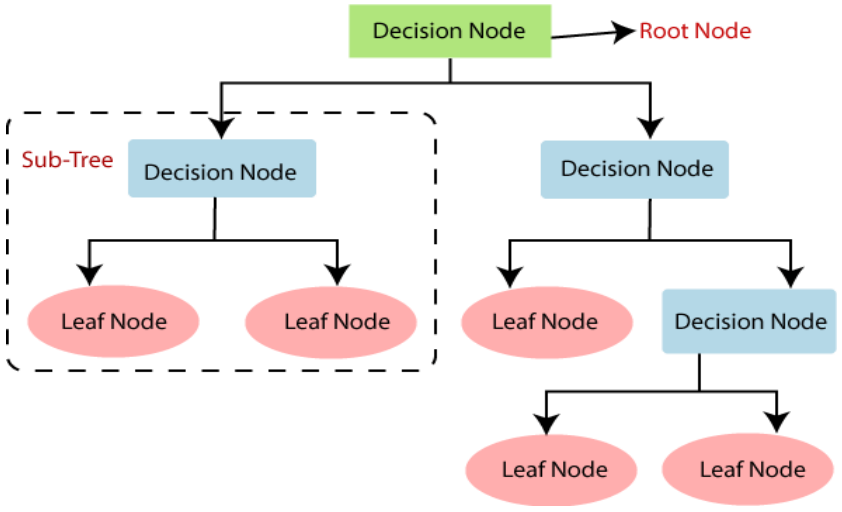
In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.

A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

Below diagram explains the general structure of a decision tree:

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.

Decision Tree Classification Algorithm



4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

5. Procedure:

In this exercise, you will implement a decision tree from scratch and apply it to the task of classifying whether a mushroom is edible or poisonous.

Outline

Packages
Problem Statement
Dataset
One hot encoded dataset
Decision Tree Refresher
Calculate entropy
Exercise 1
Split dataset
Exercise 2
Calculate information gain
Exercise 3
Get best split
Exercise 4
Building the tree

Click on the following files to access Labs Notebooks and python files.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
dataset.head(2)
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	

```
dataset.isnull().sum()
```

```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
```

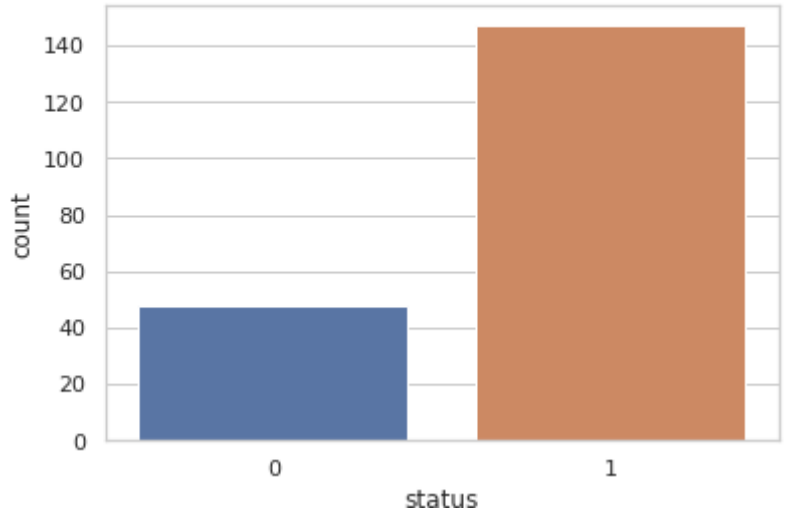
```
Disorder = dataset[dataset['status']==1]
Normal = dataset[dataset['status']== 0]
```

```
print(Disorder.shape)
print(Normal.shape)
```

```
(147, 24)
(48, 24)
```

```
sns.set(style="whitegrid")
sns.countplot(dataset.status)
plt.show()
```

```
lowing variable as a keyword arg: x. From version 0.12, the o
d passing other arguments without an explicit keyword will re
  warnings.warn(
```




```
X=dataset.drop(["status" , "name"], axis=1)
X.shape
```

```
(195, 22)
```

```
y=dataset["status"]
y.shape
```

```
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)
```

```
from sklearn.utils.class_weight import compute_sample_weight
w_train = compute_sample_weight('balanced', y_train)
print(w_train, y_train)
```

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
```

```
log.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
log.fit(X_train, y_train, sample_weight=w_train)
```

```
LogisticRegression()
```

```
log.score(X_train, y_train)
```

```
0.8461538461538461
```

```
log.score(X_train, y_train, sample_weight=w_train)
```

```
0.8447814451382695
```

```
from sklearn.tree import DecisionTreeClassifier
DTree = DecisionTreeClassifier(criterion= 'gini', max_depth=1)
```

```
DTree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=10)
```

```
DTree.fit(X_train, y_train, sample_weight=w_train)
```

```
DecisionTreeClassifier(max_depth=1)
```

```
DTree.score(X_train, y_train, sample_weight=w_train)
```

```
0.8086529884032114
```

```
DTree.score(X_train, y_train)
```

```
0.8589743589743589
```

```
from sklearn.tree import DecisionTreeClassifier
DTree = DecisionTreeClassifier(criterion= 'gini', max_depth=5)
```

```
DTree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=5)
```

```
DTree.fit(X_train, y_train, sample_weight=w_train)
```

```
DecisionTreeClassifier(max_depth=5)
```

```
DTree.score(X_train, y_train, sample_weight=w_train)
```

```
0.9872881355932203
```

```
DTree.score(X_train, y_train)
```

```
0.9807692307692307
```

```
from sklearn.tree import DecisionTreeClassifier
DTree = DecisionTreeClassifier(criterion= 'gini', max_depth=10)
```

```
DTree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=10)
```

```
DTree.fit(X_train, y_train, sample_weight=w_train)
```

```
DecisionTreeClassifier(max_depth=10)
```

```
DTree.score(X_train, y_train, sample_weight=w_train)
```

```
1.0
```

```
DTree.score(X_train, y_train)
```

```
1.0
```

6. Observations:

In this lab, I have learnt about Decision Tree Classification and its implemetations.

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE), but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 9 (Open-ended)_____24/11/2022

1. Objectives:

After completing this lab, Students will be able to;

- Credit Card Fraud Detection
- Utilize your machine learning (ML) modeling skills by using classification model of your choice to recognize fraudulent credit card transactions

2. Corresponding CLO and PLO:

- CLO 2, PLO 5

3. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

4. Procedure:

Lab Task :

You will use a real dataset to train each of these models. The dataset includes information about transactions made by credit cards in September 2013 by European cardholders. You will use the trained model to assess if a credit card transaction is legitimate or not.

Introduction

Imagine that you work for a financial institution and part of your job is to build a model that predicts if a credit card transaction is fraudulent or not. You can model the problem as a binary classification problem. A transaction belongs to the positive class (1) if it is a fraud, otherwise it belongs to the negative class (0).

You have access to transactions that occured over a certain period of time. The majority of the transactions are normally legitimate and only a small fraction are non-legitimate. Thus, typically you have access to a dataset that is highly unbalanced. This is also the case of the current dataset: only 492 transactions out of 284,807 are fraudulent (the positive class - the frauds - accounts for 0.172% of all transactions).

To train the model you can use part of the input dataset and the remaining data can be used to assess the quality of the trained model. First, let's download the dataset.

```
import files necessary
train a classifier using scikit-learn


# Evaluate your classifier
# run inference and compute the probabilities of the test samples
# to belong to the class of fraudulent transactions
```

```
# Sample code to get predictions: predicted_labels = model.predict_probality(X_test)[:,1]
#testing_metric_according_to_model_used, or score(true_labels, predicted_labels)

#compute accuracy metric of your choice.
```

Code:

Import Libraries

In [1]:
```
# Import the libraries we need to use in this lab
from __future__ import print_function
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize, StandardScaler
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.metrics import roc_auc_score
import time
import warnings
warnings.filterwarnings('ignore')
```

Dataset Analysis

In this section you will read the dataset in a Pandas dataframe and visualize its content. You will also look at some data statistics.
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

In [3]:
```
# read the input data
raw_data = pd.read_csv('creditcard.csv')
print("There are " + str(len(raw_data)) + " observations in the credit card fraud dataset.")
print("There are " + str(len(raw_data.columns)) + " variables in the dataset.")

# display the first rows in the dataset
raw_data.head()
```

```
There are 284807 observations in the credit card fraud dataset.
There are 31 variables in the dataset.
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.1
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.1
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.3
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.6
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.2

5 rows × 31 columns

In [4]:
```
n_replicas = 10

# inflate the original dataset
big_raw_data = pd.DataFrame(np.repeat(raw_data.values, n_replicas, axis=0), columns=raw_data.columns)

print("There are " + str(len(big_raw_data)) + " observations in the inflated credit card fraud dataset.")
print("There are " + str(len(big_raw_data.columns)) + " variables in the dataset.")

# display first rows in the new dataset
big_raw_data.head()
```

```
There are 2848070 observations in the inflated credit card fraud dataset.
There are 31 variables in the dataset.
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
2	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
3	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
4	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853

5 rows × 31 columns

Each row in the dataset represents a credit card transaction. As shown above, each row has 31 variables. One variable (the last variable in the table above) is called Class and represents the target variable. Your objective will be to train a model that uses the other variables to predict the value of the Class variable. Let's first retrieve basic statistics about the target variable.

Note: For confidentiality reasons, the original names of most features are anonymized V1, V2 .. V28. The values of these features are the result of a PCA transformation (dont worry about that for now) and are numerical. The feature 'Class' is the target variable and it takes two values: 1 in case of fraud and 0 otherwise. For more information about the dataset please visit this webpage: https://www.kaggle.com/mlg-ulb/creditcardfraud.

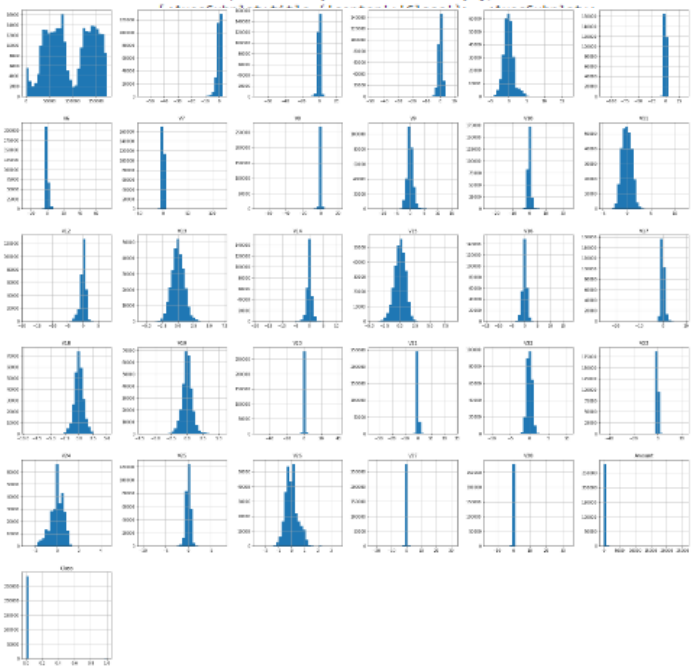
In [5]:
```
# get the set of distinct classes
labels = big_raw_data.Class.unique()

# get the count of each class
sizes = big_raw_data.Class.value_counts().values

# plot the class value counts
fig, ax = plt.subplots()
ax.pie(sizes, labels=labels, autopct='%1.3f%%')
ax.set_title('Target Variable Value Counts')
plt.show()
```

As shown above, the Class variable has two values: 0 (the credit card transaction is legitimate) and 1 (the credit card transaction is fraudulent). Thus, you need to model a binary classification problem. Moreover, the dataset is highly unbalanced, the target variable classes are not represented equally. This case requires special attention when training or when evaluating the quality of a model. One way of handing this case at train time is to bias the model to pay more attention to the samples in the minority class. The models under the current study will be configured to take into account the class weights of the samples at train/fit time.

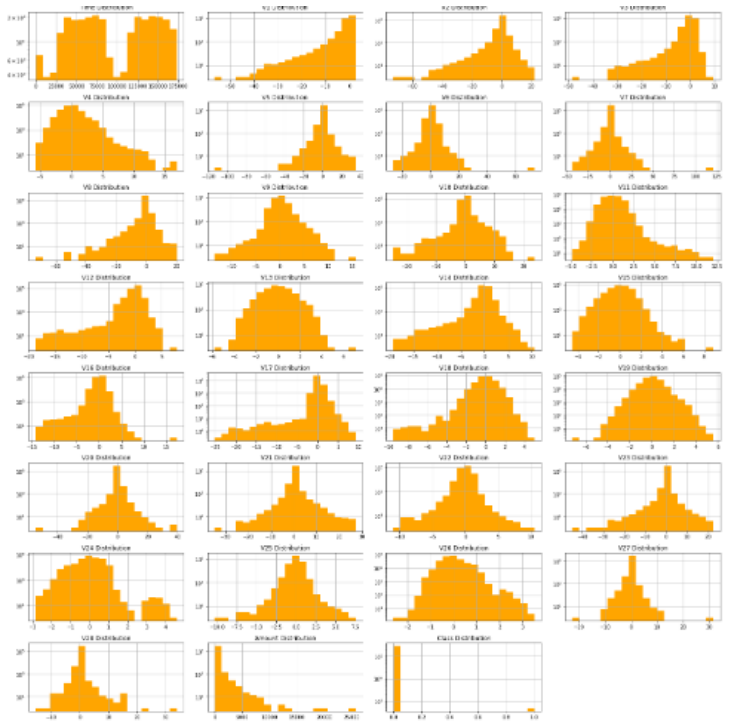
The credit card transactions have different amounts. plot a histogram that shows the distribution of these amounts? What is the range of these amounts (min/max)? print the percentiles of the amount values?

In [6]:
```
big_raw_data.hist(bins=30, figsize=(30, 30))
```

Out[6]:
```
array([[<AxesSubplot:title={'center':'Time'}>,
        <AxesSubplot:title={'center':'V1'}>,
        <AxesSubplot:title={'center':'V2'}>,
        <AxesSubplot:title={'center':'V3'}>,
        <AxesSubplot:title={'center':'V4'}>,
        <AxesSubplot:title={'center':'V5'}>],
       [<AxesSubplot:title={'center':'V6'}>,
        <AxesSubplot:title={'center':'V7'}>,
        <AxesSubplot:title={'center':'V8'}>,
        <AxesSubplot:title={'center':'V9'}>,
        <AxesSubplot:title={'center':'V10'}>,
        <AxesSubplot:title={'center':'V11'}>],
       [<AxesSubplot:title={'center':'V12'}>,
        <AxesSubplot:title={'center':'V13'}>,
        <AxesSubplot:title={'center':'V14'}>,
        <AxesSubplot:title={'center':'V15'}>,
        <AxesSubplot:title={'center':'V16'}>,
        <AxesSubplot:title={'center':'V17'}>],
       [<AxesSubplot:title={'center':'V18'}>,
        <AxesSubplot:title={'center':'V19'}>,
        <AxesSubplot:title={'center':'V20'}>,
        <AxesSubplot:title={'center':'V21'}>,
        <AxesSubplot:title={'center':'V22'}>,
        <AxesSubplot:title={'center':'V23'}>],
       [<AxesSubplot:title={'center':'V24'}>,
        <AxesSubplot:title={'center':'V25'}>,
        <AxesSubplot:title={'center':'V26'}>,
        <AxesSubplot:title={'center':'V27'}>,
        <AxesSubplot:title={'center':'V28'}>,
        <AxesSubplot:title={'center':'Amount'}>],
```



In [7]:
```
def draw_histograms(dataframe, features, rows, cols):
    fig=plt.figure(figsize=(20,20))
    for i, feature in enumerate(features):
        ax=fig.add_subplot(rows,cols,i+1)
        dataframe[feature].hist(bins=20,ax=ax,facecolor='orange')
        ax.set_title(feature+" Distribution", color='black')
        ax.set_yscale('log')
    fig.tight_layout()
    plt.show()
draw_histograms(big_raw_data,big_raw_data.columns,8,4)
```



In [ ]:
```
import numpy as np

# Array of data
arr = [[5,6,8],[6,9,2]]

# Finding the 90 percentile
x = np.percentile(arr, 90)
print(x)
```

In [8]:
```
#How to Find Percentiles of Several DataFrame Columns
#The following code shows how to find the 95th percentile value for a several columns in a pandas DataFrame:

#import numpy as np
#import pandas as pd

#create DataFrame
#df = pd.DataFrame({'var1': [25, 12, 15, 14, 19, 23, 25, 29, 33, 35],
#                   'var2': [5, 7, 7, 9, 12, 9, 9, 4, 14, 15],
 #                  'var3': [11, 8, 10, 6, 6, 5, 9, 12, 13, 16]})

#find 95th percentile of each column
big_raw_data.quantile(.95)



#find 95th percentile of just columns var1 and var2
#df[['var1', 'var2']].quantile(.95)

#var1    34.10
#var2    14.55
```

Out[8]:
```
Time      164144.000000
V1             2.081227
V2             1.808585
V3             2.062646
V4             2.566522
V5             2.098980
V6             3.160388
V7             1.407643
V8             1.050008
V9             1.780807
V10            1.548559
V11            1.614042
V12            1.243067
V13            1.607878
V14            1.393667
V15            1.373097
V16            1.325263
V17            1.274626
V18            1.394400
V19            1.286165
V20            0.836168
V21            0.537869
```

Dataset Preprocessing

In this subsection you will prepare the data for training.

In [9]:
```
# data preprocessing such as scaling/normalization is typically useful for
# linear models to accelerate the training convergence

# standardize features by removing the mean and scaling to unit variance
big_raw_data.iloc[:, 1:30] = StandardScaler().fit_transform(big_raw_data.iloc[:, 1:30])
data_matrix = big_raw_data.values

# X: feature matrix (for this analysis, we exclude the Time variable from the dataset)
X = data_matrix[:, 1:30]

# y: labels vector
y = data_matrix[:, 30]

# data normalization
X = normalize(X, norm="l1")

# print the shape of the features matrix and the labels vector
print('X.shape=', X.shape, 'y.shape=', y.shape)
```

```
X.shape= (2848070, 29) y.shape= (2848070,)
```

Dataset Train/Test Split

Now that the dataset is ready for building the classification models, you need to first divide the pre-processed dataset into a subset to be used for training the model (the train set) and a subset to be used for evaluating the quality of the model (the test set).

In [ ]:
```

```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
print('X_train.shape=', X_train.shape, 'Y_train.shape=', y_train.shape)
print('X_test.shape=', X_test.shape, 'Y_test.shape=', y_test.shape)
```

```
X_train.shape= (1993649, 29) Y_train.shape= (1993649,)
X_test.shape= (854421, 29) Y_test.shape= (854421,)
```

removing class imbalance

In [11]:
```
# compute the sample weights to be used as input to the train routine so that
# it takes into account the class imbalance present in this dataset
w_train = compute_sample_weight('balanced', y_train)
```

Build a classifier

In [13]:
```
#import files necessary
#train a classifier using scikit-learn

from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
# your code here
```

Out[13]:
```
LogisticRegression()
```

In [15]:
```
## Evaluate your classifier
# run inference and compute the probabilities of the test samples
# to belong to the class of fraudulent transactions

# Sample code to get predictions: predicted_labels = model.predict_probality(X_test)[:,1]
#testing_metric_according_to_model_used, or score(true_labels, predicted_labels)

#compute accuracy metric of your choice.


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

#your code here
from sklearn.metrics import accuracy_score
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, y_train)
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, y_test)

print('Accuracy on Training data : ', training_data_accuracy)
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy on Training data :  0.9993807796257014
Accuracy score on Test Data :  0.999314155433913 8
```

5. Observations:

In this lab, I have performed an open ended lab in which I have applied all the functions which I have leant in the previous labs, where I have predicted credit card fraud detection.

6. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE) , but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 10_____1/12/2022

1. Objectives:

After completing this lab, Students will be able to;

- Credit Card Fraud Detection using Scikit-Learn and Snap ML
- Classification tree Support Vector Machine
- Perform basic data preprocessing in Python
- Model a classification task using the Scikit-Learn and Snap ML Python APIs
- Train Suppport Vector Machine and Decision Tree models using Scikit-Learn and Snap ML
- Run inference and assess the quality of the trained models

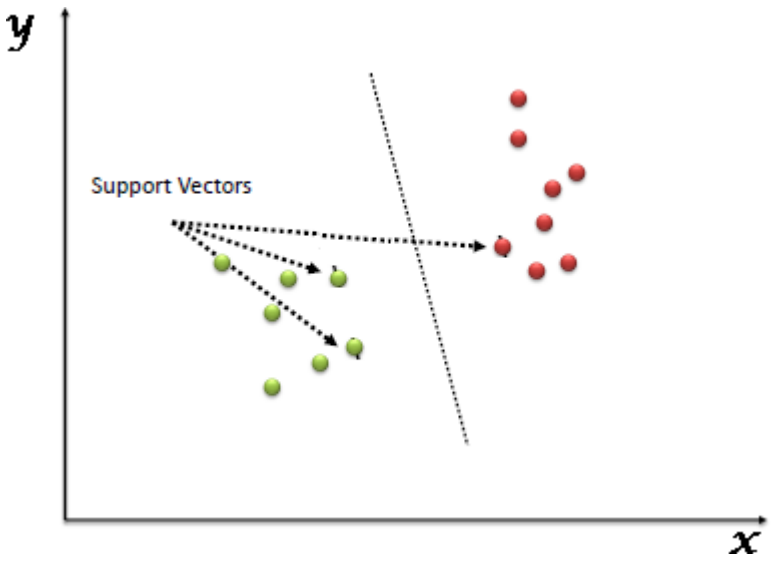
2. Corresponding CLO and PLO:

- CLO 1, PLO 3

3. Theory:

Support Vector Machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).



SVM_1

Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook

5. Procedure:

Lab Task :

In this exercise session you will consolidate your machine learning (ML) modeling skills by using two popular classification models to recognize fraudulent credit card transactions. These models are: Decision Tree and Support Vector Machine. You will use a real dataset to train each of these models. The dataset includes information about
transactions made by credit cards in September 2013 by European cardholders. You will use the trained model to assess if a credit card transaction is legitimate or not.

In the current exercise session, you will practice not only the Scikit-Learn Python interface, but also the Python API offered by the Snap Machine Learning (Snap ML) library. Snap ML is a high-performance IBM library for ML modeling. It provides highly-efficient CPU/GPU implementations of linear models and tree-based models. Snap ML not only accelerates ML algorithms through system awareness, but it also offers novel ML algorithms with best-in-class accuracy. For more information, please visit https://www.zurich.ibm.com/snapml/ information page.

In [4]:
```
# read the input data
raw_data = pd.read_csv('creditcardfraud/creditcard.csv')
print("There are " + str(len(raw_data)) + " observations in the credit card fraud dataset.")
print("There are " + str(len(raw_data.columns)) + " variables in the dataset.")

# display the first rows in the dataset
raw_data.head()
```

```
There are 284807 observations in the credit card fraud dataset.
There are 31 variables in the dataset.
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

In [5]:
```
n_replicas = 10

# inflate the original dataset
big_raw_data = pd.DataFrame(np.repeat(raw_data.values, n_replicas, axis=0), columns=raw_data.columns)

print("There are " + str(len(big_raw_data)) + " observations in the inflated credit card fraud dataset.")
print("There are " + str(len(big_raw_data.columns)) + " variables in the dataset.")

# display first rows in the new dataset
```

In [5]:
```
n_replicas = 10

# inflate the original dataset
big_raw_data = pd.DataFrame(np.repeat(raw_data.values, n_replicas, axis=0), columns=raw_data.columns)

print("There are " + str(len(big_raw_data)) + " observations in the inflated credit card fraud dataset.")
print("There are " + str(len(big_raw_data.columns)) + " variables in the dataset.")

# display first rows in the new dataset
big_raw_data.head()
```

```
There are 2848070 observations in the inflated credit card fraud dataset.
There are 31 variables in the dataset.
```

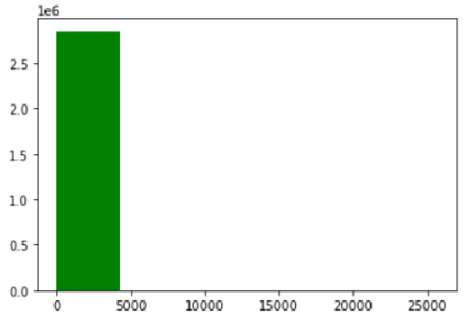
Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.
1	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.
2	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.
3	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.
4	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.

5 rows × 31 columns

In [6]:
```
# get the set of distinct classes
labels = big_raw_data.Class.unique()

# get the count of each class
sizes = big_raw_data.Class.value_counts().values
```

In [7]:
```
# we provide our solution here
plt.hist(big_raw_data.Amount.values, 6, histtype='bar', facecolor='g')
plt.show()

print("Minimum amount value is ", np.min(big_raw_data.Amount.values))
print("Maximum amount value is ", np.max(big_raw_data.Amount.values))
print("90% of the transactions have an amount less or equal than ", np.percentile(raw_data.Amount.values, 90))
```




```
Minimum amount value is  0.0
Maximum amount value is  25691.16
90% of the transactions have an amount less or equal than  203.0
```

```
# standardize features by removing the mean and scaling to unit variance
big_raw_data.iloc[:, 1:30] = StandardScaler().fit_transform(big_raw_data.iloc[:, 1:30])
data_matrix = big_raw_data.values

# X: feature matrix (for this analysis, we exclude the Time variable from the dataset)
X = data_matrix[:, 1:30]

# y: labels vector
y = data_matrix[:, 30]

# data normalization
X = normalize(X, norm="l1")

# print the shape of the features matrix and the labels vector
print('X.shape=', X.shape, 'y.shape=', y.shape)
```

```
X.shape= (2848070, 29) y.shape= (2848070,)
```

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
print('X_train.shape=', X_train.shape, 'Y_train.shape=', y_train.shape)
print('X_test.shape=', X_test.shape, 'Y_test.shape=', y_test.shape)
```

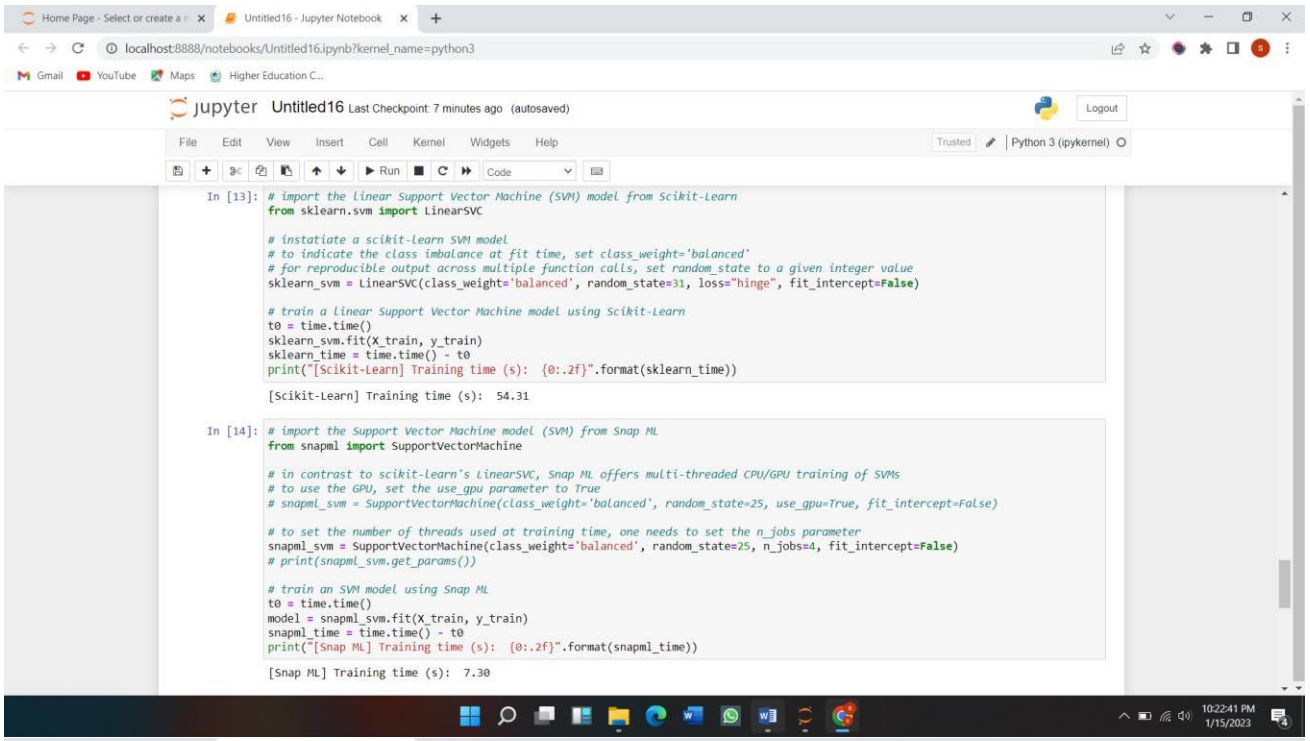
```
X_train.shape= (1993649, 29) Y_train.shape= (1993649,)
X_test.shape= (854421, 29) Y_test.shape= (854421,)
```

In [10]:
```
# compute the sample weights to be used as input to the train routine so that
# it takes into account the class imbalance present in this dataset
w_train = compute_sample_weight('balanced', y_train)

# import the Decision Tree Classifier Model from scikit-learn
from sklearn.tree import DecisionTreeClassifier

snapml_dt = DecisionTreeClassifier(max_depth=4, random_state=45, n_jobs=4)

# train a Decision Tree Classifier model using Snap ML
t0 = time.time()
snapml_dt.fit(X_train, y_train, sample_weight=w_train)
snapml_time = time.time()-t0
print("[Snap ML] Training time (s):  {0:.5f}".format(snapml_time))
```

```
[Snap ML] Training time (s):  2.51329
```

In [12]:
```
# Snap ML vs Scikit-Learn training speedup
training_speedup = sklearn_time/snapml_time
print('[Decision Tree Classifier] Snap ML vs. Scikit-Learn speedup : {0:.2f}x '.format(training_speedup))

# run inference and compute the probabilities of the test samples
# to belong to the class of fraudulent transactions
sklearn_pred = sklearn_dt.predict_proba(X_test)[:,1]

# evaluate the Compute Area Under the Receiver Operating Characteristic
# Curve (ROC-AUC) score from the predictions
sklearn_roc_auc = roc_auc_score(y_test, sklearn_pred)
print('[Scikit-Learn] ROC-AUC score : {0:.3f}'.format(sklearn_roc_auc))

# run inference and compute the probabilities of the test samples
# to belong to the class of fraudulent transactions
snapml_pred = snapml_dt.predict_proba(X_test)[:,1]

# evaluate the Compute Area Under the Receiver Operating Characteristic
# Curve (ROC-AUC) score from the prediction scores
snapml_roc_auc = roc_auc_score(y_test, snapml_pred)
print('[Snap ML] ROC-AUC score : {0:.3f}'.format(snapml_roc_auc))
```

```
[Decision Tree Classifier] Snap ML vs. Scikit-Learn speedup : 9.21x
[Scikit-Learn] ROC-AUC score : 0.966
[Snap ML] ROC-AUC score : 0.966
```



Home Page - Select or create a n | Untitled16 - Jupyter Notebook

localhost:8888/notebooks/Untitled16.ipynb?kernel_name=python3

Gmail YouTube Maps Higher Education C...

jupyter Untitled16 Last Checkpoint: 7 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

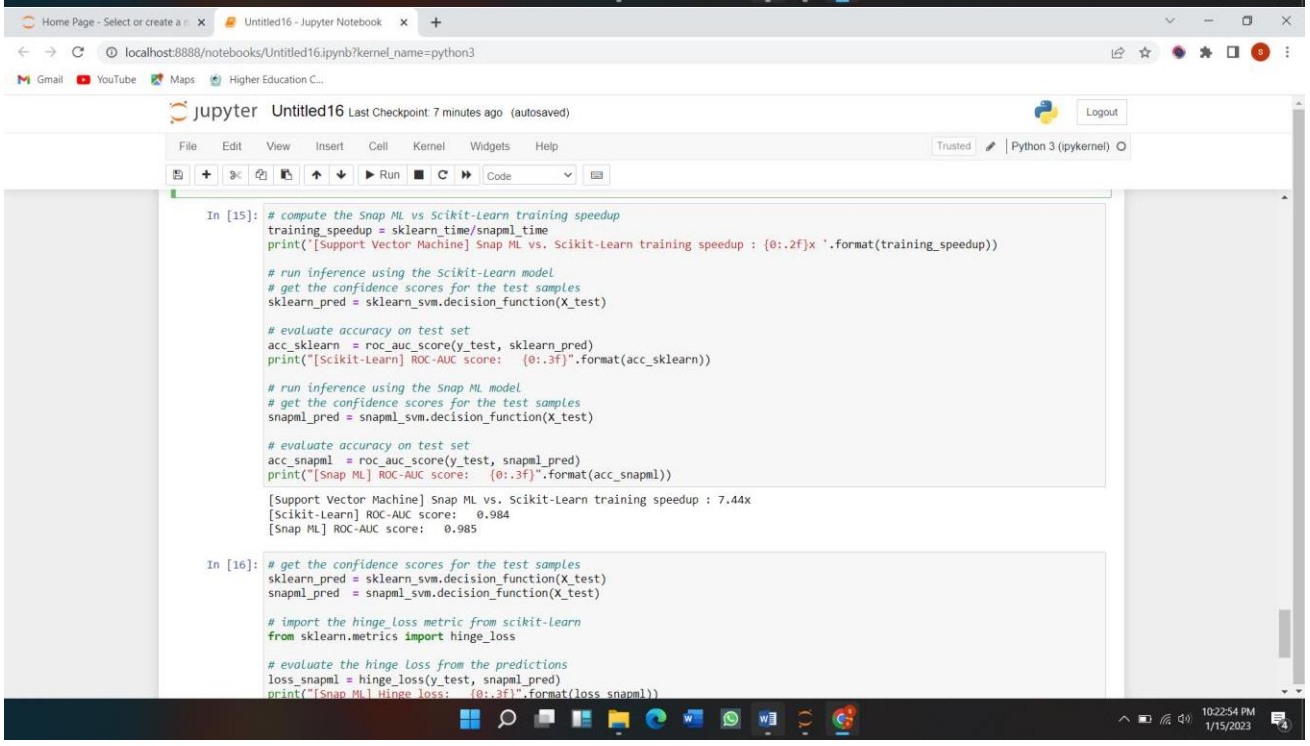
In [13]:
```
# import the linear Support Vector Machine (SVM) model from Scikit-Learn
from sklearn.svm import LinearSVC

# instatiate a scikit-learn SVM model
# to indicate the class imbalance at fit time, set class_weight='balanced'
# for reproducible output across multiple function calls, set random_state to a given integer value
sklearn_svm = LinearSVC(class_weight='balanced', random_state=31, loss="hinge", fit_intercept=False)

# train a linear Support Vector Machine model using Scikit-Learn
t0 = time.time()
sklearn_svm.fit(X_train, y_train)
sklearn_time = time.time() - t0
print("[Scikit-Learn] Training time (s):  {0:.2f}".format(sklearn_time))
```

```
[Scikit-Learn] Training time (s):  54.31
```

In [14]:
```
# import the Support Vector Machine model (SVM) from Snap ML
from snapml import SupportVectorMachine

# in contrast to scikit-learn's LinearSVC, Snap ML offers multi-threaded CPU/GPU training of SVMs
# to use the GPU, set the use_gpu parameter to True
# snapml_svm = SupportVectorMachine(class_weight='balanced', random_state=25, use_gpu=True, fit_intercept=False)

# to set the number of threads used at training time, one needs to set the n_jobs parameter
snapml_svm = SupportVectorMachine(class_weight='balanced', random_state=25, n_jobs=4, fit_intercept=False)
# print(snapml_svm.get_params())

# train an SVM model using Snap ML
t0 = time.time()
model = snapml_svm.fit(X_train, y_train)
snapml_time = time.time() - t0
print("[Snap ML] Training time (s):  {0:.2f}".format(snapml_time))
```

```
[Snap ML] Training time (s):  7.30
```

10:22:41 PM 1/15/2023



Home Page - Select or create a n | Untitled16 - Jupyter Notebook

localhost:8888/notebooks/Untitled16.ipynb?kernel_name=python3

Gmail YouTube Maps Higher Education C...

jupyter Untitled16 Last Checkpoint: 7 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

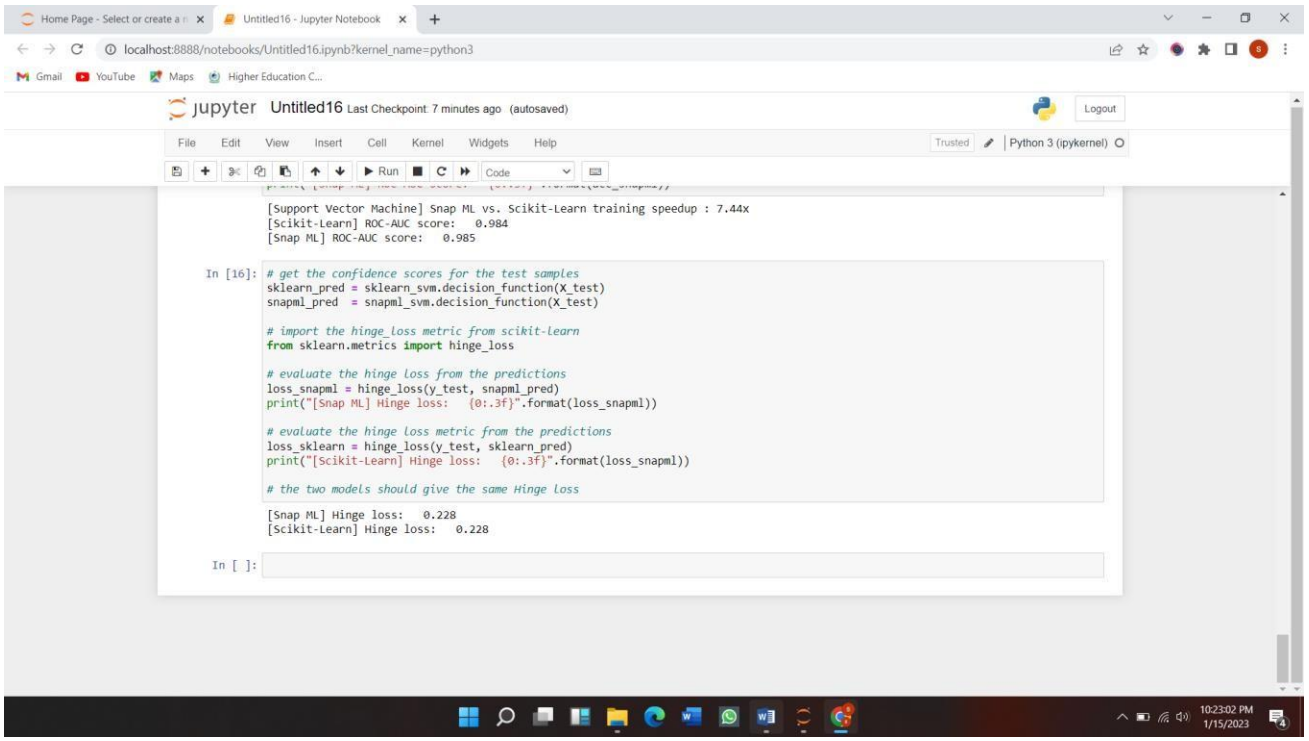
Run Code

In [15]:
```
# compute the Snap ML vs Scikit-Learn training speedup
training_speedup = sklearn_time/snapml_time
print('[Support Vector Machine] Snap ML vs. Scikit-Learn training speedup : {0:.2f}x '.format(training_speedup))

# run inference using the Scikit-Learn model
# get the confidence scores for the test samples
sklearn_pred = sklearn_svm.decision_function(X_test)

# evaluate accuracy on test set
acc_sklearn  = roc_auc_score(y_test, sklearn_pred)
print("[Scikit-Learn] ROC-AUC score:   {0:.3f}".format(acc_sklearn))

# run inference using the Snap ML model
# get the confidence scores for the test samples
snapml_pred = snapml_svm.decision_function(X_test)

# evaluate accuracy on test set
acc_snapml  = roc_auc_score(y_test, snapml_pred)
print("[Snap ML] ROC-AUC score:   {0:.3f}".format(acc_snapml))
```

```
[Support Vector Machine] Snap ML vs. Scikit-Learn training speedup : 7.44x
[Scikit-Learn] ROC-AUC score:   0.984
[Snap ML] ROC-AUC score:   0.985
```

In [16]:
```
# get the confidence scores for the test samples
sklearn_pred = sklearn_svm.decision_function(X_test)
snapml_pred  = snapml_svm.decision_function(X_test)

# import the hinge_loss metric from scikit-learn
from sklearn.metrics import hinge_loss

# evaluate the hinge loss from the predictions
loss_snapml = hinge_loss(y_test, snapml_pred)
print("[Snap ML] Hinge loss:   {0:.3f}".format(loss_snapml))
```

10:22:54 PM 1/15/2023



6. Observations:

After completing this lab, I have learnt about Credit Card Fraud Detection using Scikit-Learn and Snap ML, Classification tree Support Vector Machine, Perform basic data preprocessing in Python, Model a classification task using the Scikit-Learn and Snap ML Python APIs, Train Support Vector Machine and Decision Tree models using Scikit-Learn and Snap ML Run inference and assess the quality of the trained models.

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE) , but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 11_____15/12/2022

1. Objectives:

After completing this lab, Students will be able to;

- Neurons and Layers
- Explore the inner workings of neurons/units and layers as well as a regression model

2. Corresponding CLO and PLO:

- CLO 1, PLO 3

3. Theory:

Introduction

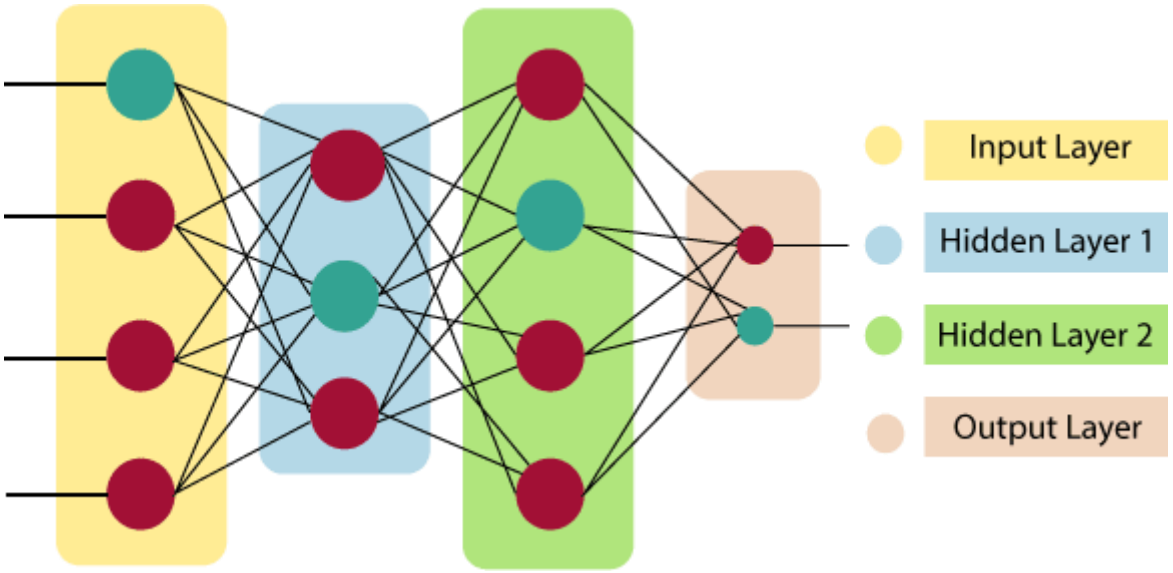
ANN is inspired by the biological neural network. For simplicity, in computer science, it is represented as a set of layers. These layers are categorized into three classes which are input, hidden, and output.

Knowing the number of input and output layers and number of their neurons is the easiest part. Every network has a single input and output layers. The number of neurons in the input layer equals the number of input variables in the data being processed. The number of neurons in the output layer equals the number of outputs associated with each input. But the challenge is knowing the number of hidden layers and their neurons.

Here are some guidelines to know the number of hidden layers and neurons per each hidden layer in a classification problem:

Based on the data, draw an expected decision boundary to separate the classes.
Express the decision boundary as a set of lines. Note that the combination of such lines must yield to the decision boundary.
The number of selected lines represents the number of hidden neurons in the first hidden layer.
To connect the lines created by the previous layer, a new hidden layer is added. Note that a new hidden layer is added each time you need to create connections among the lines in the previous hidden layer.
The number of hidden neurons in each new hidden layer equals the number of connections to be made.

Artificial Neural Network primarily consists of three layers:



Input Layer:
As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:
The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:
The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook
- TensorFlow

5. Procedure:

In this lab we will explore the inner workings of neurons/units and layers as well as a regression model

Packages

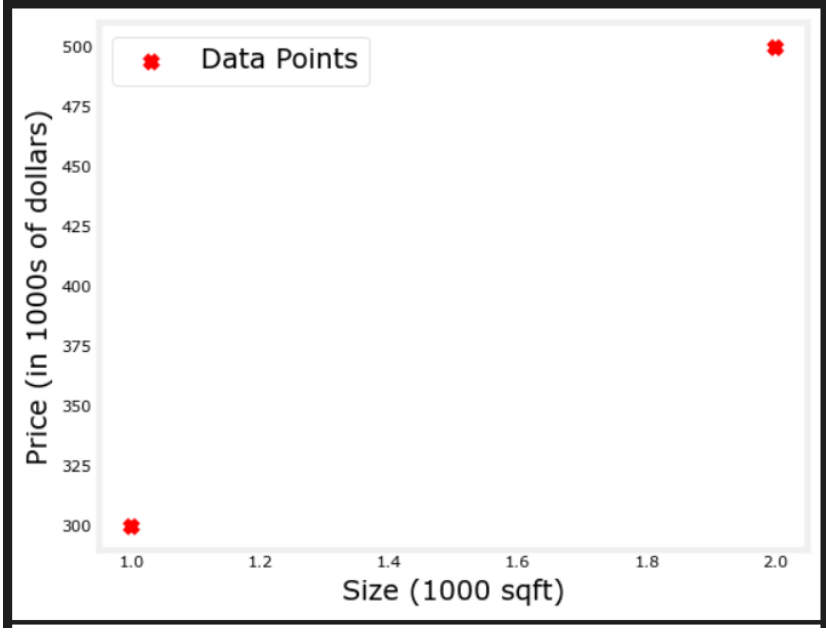
Tensorflow and Keras

Tensorflow is a machine learning package developed by Google. In 2019, Google integrated Keras into Tensorflow and released Tensorflow 2.0. Keras is a framework developed independently by François Chollet that creates a simple, layer-centric interface to Tensorflow. This course will be using the Keras interface.

Click on the below icon to access the Jupyter notebook files for today's Lab task:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras import Sequential
from tensorflow.keras.losses import MeanSquaredError, BinaryCrossentropy
from tensorflow.keras.activations import sigmoid
from lab_utils_common import dlc
from lab_neurons_utils import plt_prob_1d, sigmoidnp, plt_linear, plt_logistic
plt.style.use('./deeplearning.mplstyle')
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
tf.autograph.set_verbosity(0)
```

```
X_train = np.array([[1.0], [2.0]], dtype=np.float32)           #(size in 1000 square feet)
Y_train = np.array([[300.0], [500.0]], dtype=np.float32)       #(price in 1000s of dollars)

fig, ax = plt.subplots(1,1)
ax.scatter(X_train, Y_train, marker='x', c='r', label="Data Points")
ax.legend( fontsize='xx-large')
ax.set_ylabel('Price (in 1000s of dollars)', fontsize='xx-large')
ax.set_xlabel('Size (1000 sqft)', fontsize='xx-large')
plt.show()
```



```
linear_layer = tf.keras.layers.Dense(units=1, activation = 'linear', )
```

Let's examine the weights.

```
linear_layer.get_weights()
```

```
[]
```

There are no weights as the weights are not yet instantiated. Let's try the model on one example in `X_train`. This will trigger the instantiation of Note, the input to the layer must be 2-D, so we'll reshape it.

```
a1 = linear_layer(X_train[0].reshape(1,1))
print(a1)
```

```
tf.Tensor([[0.26]], shape=(1, 1), dtype=float32)
```

```
w, b= linear_layer.get_weights()
print(f"w = {w}, b={b}")
```

```
w = [[0.26]], b=[0.]
```

```
set_w = np.array([[200]])
set_b = np.array([100])

(variable) linear_layer: Any                    numpy arrays
linear_layer.set_weights([set_w, set_b])
print(linear_layer.get_weights())
```

```
[array([[200.]], dtype=float32), array([100.], dtype=float32)]
```

```
a1 = linear_layer(X_train[0].reshape(1,1))
print(a1)
alin = np.dot(set_w,X_train[0].reshape(1,1)) + set_b # WX + b
print(alin)
```

```
tf.Tensor([[300.]], shape=(1, 1), dtype=float32)
[[300.]]
```

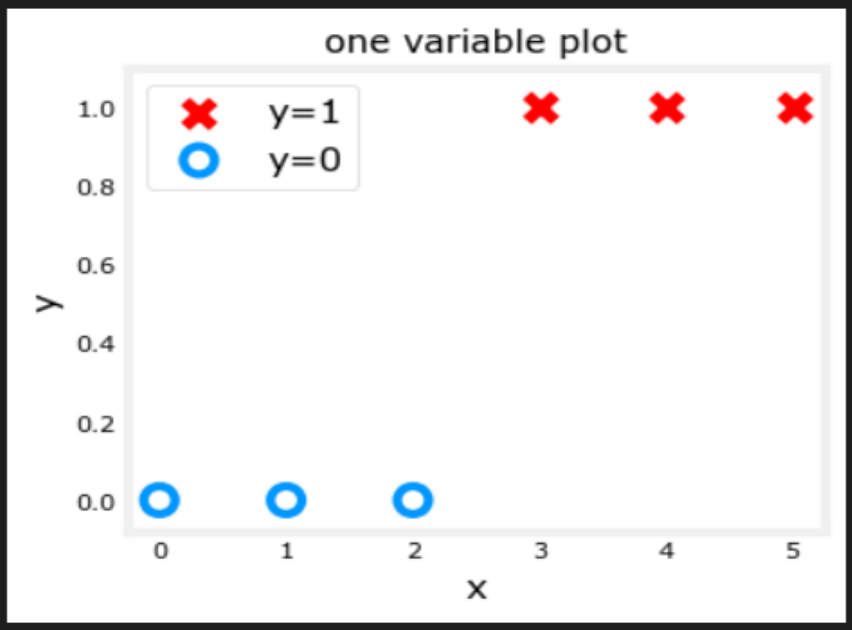
```
X_train = np.array([0., 1, 2, 3, 4, 5], dtype=np.float32).reshape(-1,1)  # 2-D Matrix
Y_train = np.array([0,  0, 0, 1, 1, 1], dtype=np.float32).reshape(-1,1)  # 2-D Matrix
```

```
pos = Y_train == 1
neg = Y_train == 0
X_train[pos]
```

```
array([3., 4., 5.], dtype=float32)
```

```
pos = Y_train == 1
neg = Y_train == 0

fig,ax = plt.subplots(1,1,figsize=(4,3))
ax.scatter(X_train[pos], Y_train[pos], marker='x', s=80, c = 'red', label="y=1")
ax.scatter(X_train[neg], Y_train[neg], marker='o', s=100, label="y=0", facecolors='none',
           edgecolors=dlc["dlblue"],lw=3)

ax.set_ylim(-0.08,1.1)
ax.set_ylabel('y', fontsize=12)
ax.set_xlabel('x', fontsize=12)
ax.set_title('one variable plot')
ax.legend(fontsize=12)
plt.show()
```

...



```
model = Sequential(
    [
        tf.keras.layers.Dense(1, input_dim=1,  activation = 'sigmoid', name='Layer1')
    ]
)
```

`model.summary()` shows the layers and number of parameters in the model. There is only one layer in this model and that layer has only one un
two parameters, w and b.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Layer1 (Dense)              (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

```
logistic_layer = model.get_layer('Layer1')
w,b = logistic_layer.get_weights()
print(w,b)
print(w.shape,b.shape)
```

```
[[0.28]] [0.]
(1, 1) (1,)
```

Let's set the weight and bias to some known values.

```
set_w = np.array([[2]])
set_b = np.array([-4.5])
# set_weights takes a list of numpy arrays
logistic_layer.set_weights([set_w, set_b])
print(logistic_layer.get_weights())
```

```
[array([[2.]], dtype=float32), array([-4.5], dtype=float32)]
```

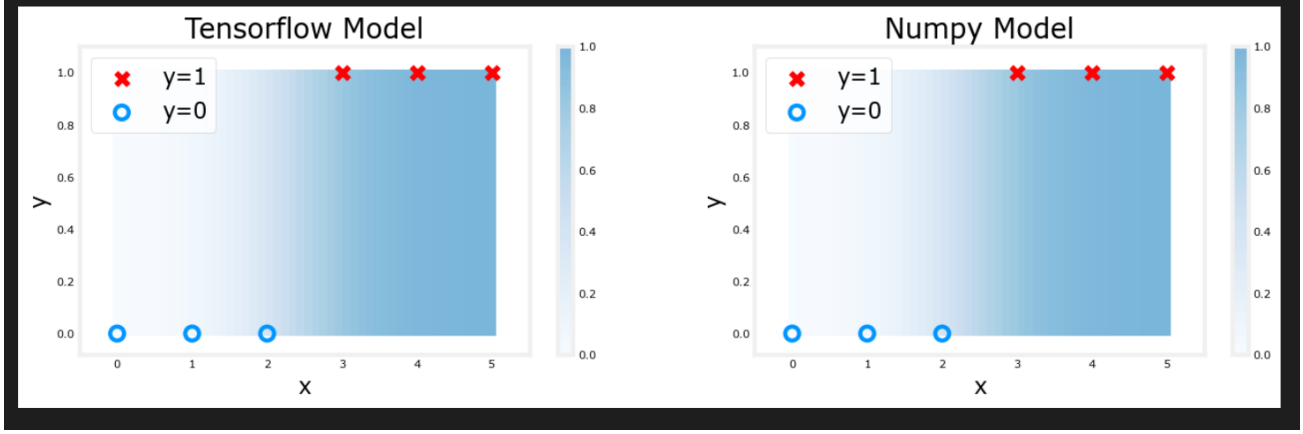
Let's compare equation (2) to the layer output.

```
a1 = model.predict(X_train[0].reshape(1,1))
print(a1)
alog = sigmoidnp(np.dot(set_w,X_train[0].reshape(1,1)) + set_b)
print(alog)
```

```
[[0.01]]
```

```
plt_logistic(X_train, Y_train, model, set_w, set_b, pos, neg)
```

Python



6. Observations:

After performing this lab, I have learnt about Neurons and Layers and Explore the inner workings of neurons/units and layers as well as a regression model

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE) , but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Lab. 12_____05/01/2023

1. Objectives:

After completing this lab, Students will be able to;

- Optimize neural network
- check and remove overfitting

2. Corresponding CLO and PLO:

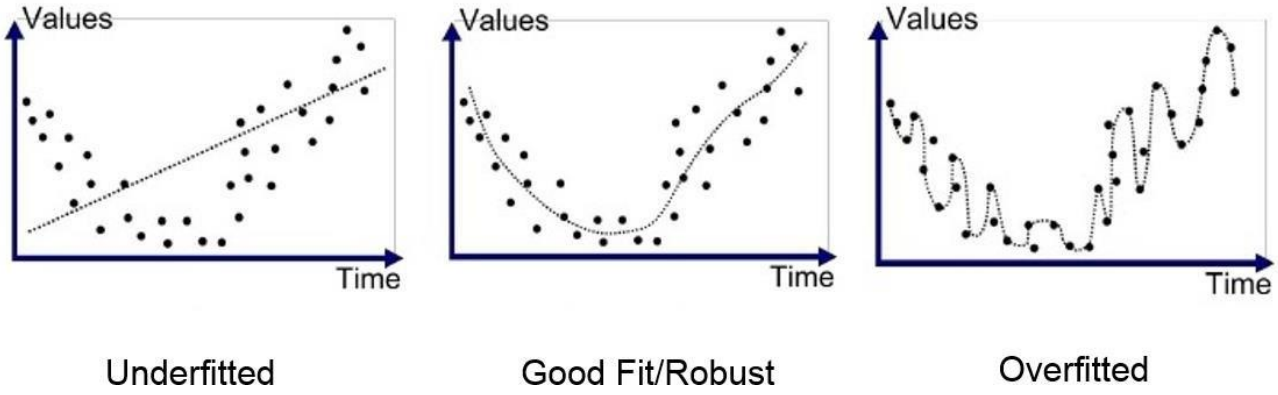
- CLO 2, PLO 5

3. Theory:

Improving the Performance of a Neural Network

Neural networks are machine learning algorithms that provide state of the accuracy on many use cases. But, a lot of times the accuracy of the network we are building might not be satisfactory or might not take us to the top positions on the leaderboard in data science competitions. Therefore, we are always looking for better ways to improve the performance of our models. There are many techniques available that could help us achieve that. Follow along to get to know them and to build your own accurate neural network.

- **Check for Overfitting**

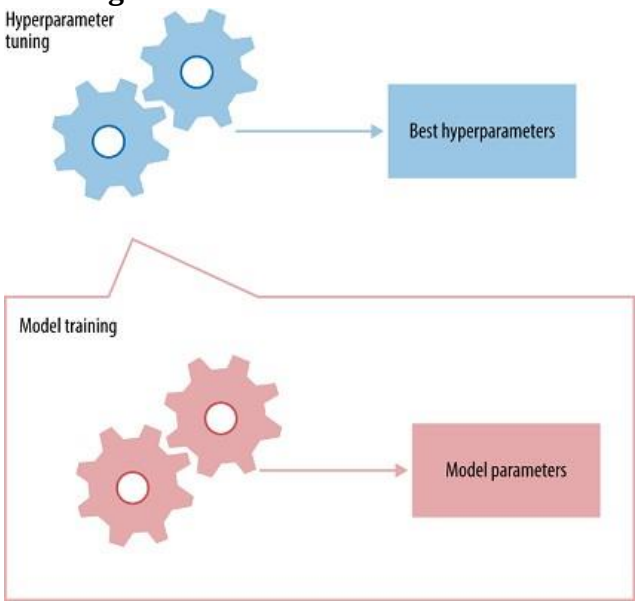


The first step in ensuring your neural network performs well on the testing data is to verify that your neural network does not overfit. Ok, stop, what is overfitting? overfitting happens when your model starts to memorise values from the training data instead of learning from them. Therefore, when your model encounters a data it hasn't seen before, it is unable to perform well on them. To give you a better understanding, let's look at an analogy. We all would have a classmate who is good at memorising, and suppose a test on maths is coming up. You and your friend, who is good at memorising start studying from the text book. Your friend goes on memorising each formula, question and answer from the textbook but you, on the other hand, are smarter than him, so you decide to build on intuition and work out problems and learn how these formulas come into play. Test day arrives, if the problems in the test paper are taken straight out of the textbooks, then you can expect your memorising friend to do better on it but, if the problems are new ones that involve applying intuition, you do better on the test and your memorising friend fails miserably.

How to identify if your model is overfitting? you can just cross check the training accuracy and testing accuracy. If training accuracy is much higher than testing accuracy then you can posit that

your model has overfitted. You can also plot the predicted points on a graph to verify. There are some techniques to avoid overfitting:

- Regularisation of data (L1 or L2).
- Dropouts — Randomly dropping connections between neurons, forcing the network to find new paths and generalise.
- Early Stopping — Precipitates the training of the neural network, leading to reduction in error in the test set.
- **Hyperparameter Tuning**

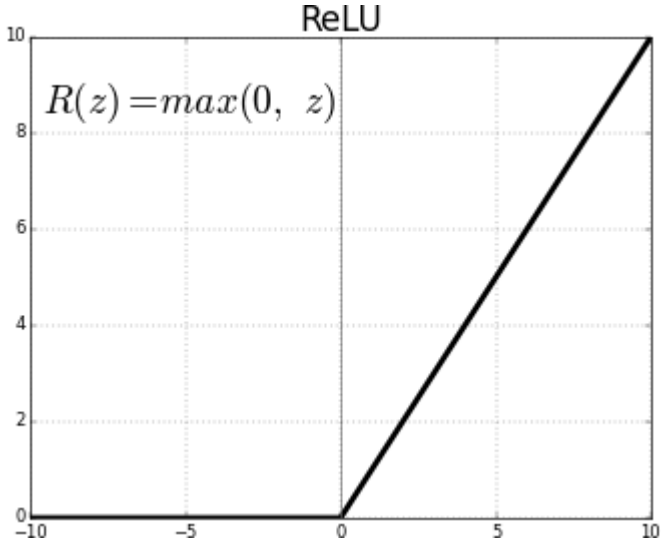


Hyperparameters are values that you must initialise to the network, these values can't be learned by the network while training. E.x: In a convolutional neural network, some of the hyperparameters are kernel size, the number of layers in the neural network, activation function, loss function, optimizer used(gradient descent, RMSprop), batch size, number of epochs to train etc.

Each neural network will have its best set of hyperparameters which will lead to maximum accuracy. You might ask, "there are so many hyperparameters, how do I choose what to use for each?", Unfortunately, there is no direct method to identify the best set of hyperparameter for each neural network so it is mostly obtained through trial and error. But, there are some best practices for some hyperparameters which are mentioned below,

- **Learning Rate** — Choosing an optimum learning rate is important as it decides whether your network converges to the global minima or not. Selecting a high learning rate almost never gets you to the global minima as you have a very good chance of overshooting it. Therefore, you are always around the global minima but never converge to it. Selecting a small learning rate can help a neural network converge to the global minima but it takes a huge amount of time. Therefore, you have to train the network for a longer period of time. A small learning rate also makes the network susceptible to getting stuck in local minimum. i.e the network will converge onto a local minima and unable to come out of it due to the small learning rate. Therefore, you must be careful while setting the learning rate.
- **Network Architecture** — There is no standard architecture that gives you high accuracy in all test cases. You have to experiment, try out different architectures, obtain inference from the result and try again. One idea that I would suggest is to use proven architectures instead of building one of your own. E.x: for image recognition task, you have VGG net, Resnet, Google's Inception network etc. These are all open sourced and have proven to be highly accurate, therefore, you could just copy their architecture and tweak them for your purpose.

- **Optimizers and Loss function** — There is a myriad of options available for you to choose from. In fact, you could even define your custom loss function if necessary. But the commonly used optimizers are RMSprop, Stochastic Gradient Descent and Adam. These optimizers seem to work for most of the use cases. Commonly used loss functions are categorical cross entropy if your use case is a classification task. If you are performing a regression task, mean squared error is the commonly used loss function. Feel free to experiment with the hyperparameters of these optimizers and also with different optimizers and loss functions.
- **Batch Size & Number of Epochs** — Again, there is no standard value for batch size and epochs that works for all use cases. You have to experiment and try out different ones. In general practice, batch size values are set as either 8, 16, 32… The number of epochs depends on the developer's preference and the computing power he/she has.



ReLU Activation Funciton

- Activation Function — Activation functions map the non-linear functional inputs to the outputs. Activation functions are highly important and choosing the right activation function helps your model to learn better. Nowadays, Rectified Linear Unit(ReLU) is the most widely used activation function as it solves the problem of vanishing gradients. Earlier Sigmoid and Tanh were the most widely used activation function. But, they suffered from the problem of vanishing gradients, i.e during backpropagation, the gradients diminish in value when they reach the beginning layers. This stopped the neural network from scaling to bigger sizes with more layers. ReLU was able to overcome this problem and hence allowed neural networks to be of large sizes.

4. Equipment's:

- Anaconda
- Python
- Google Colab
- Jupyter Notebook
- TensorFlow

5. Procedure:

Lab Task :

1. Optimize the given neural network by increasing accuracy

2. check and remove overfitting with the methods you know

```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
```

+ Code + Markdown

```
(X_train,y_train),(X_test,y_test) = keras.datasets.mnist.load_data()
```

```
X_test.shape
```

```
y_train
```

```
import matplotlib.pyplot as plt
plt.imshow(X_train[2])
```

```
X_train = X_train/255
X_test = X_test/255
```

```
X_train[0]
```

```
model = Sequential()
```

```
X_train[0]
```

```
model = Sequential()

model.add(Flatten(input_shape=(28,28)))
model.add(Dense(64,activation='relu'))
model.add(Dense(10,activation='softmax'))
```

```
model.summary()
```

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

```
history = model.fit(X_train,y_train,epochs=25,validation_split=0.2)
```

```
y_prob = model.predict(X_test)
```

```
y_pred = y_prob.argmax(axis=1)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
plt.imshow(X_test[1])
```

```
model.predict(X_test[1].reshape(1,28,28)).argmax(axis=1)
```

6. Observations:

After performing this lab, I have learnt about Optimize neural network , check and remove overfitting

7. Rubrics:

Demonstration	Absent	Student is unable to follow the provided instructions properly. The students can name the hardware or simulation platform, but unable to implement anything or on the software.	Student can understand the provided laboratory instruction and familiar with the lab environment (Trainer/Software/IDE), but cannot implement on the platform practically or on the software.	Student has followed instructions to construct the fundamental schematic/block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code and have successfully executed the program/run circuit on software platform.	Student perfectly implemented a working model/logic/circuit/block diagram/code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results.
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recoded along with detailed procedure	Appropriate computation or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all aspects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[80-100]
Marks	0.0	0.01-0.20	0.21-0.40	0.41-0.60	0.61-0.80	0.81-1.0
Date		**Total Marks**		**Instructor's Signature**		