Infotech

Surat

An internship report submitted to

UKA TARSADIA UNIVERSITY

Final Year Student of

Bachelor of Technology
in
Information Technology

By
Rajkumar Munjapara(202003103510281)
Mit Lathiya(202003103510299)

Guided by
Mr. Mayur Godhani
CEO of BVM Infotech Pvt.Ltd

Ms.Dhwani Patel
Teaching Assistant
Department of IT & CS

Department of Information Technology and Cyber Security
Chhotubhai Gopalbhai Patel Institute of Technology
Bardoli, Gujarat
2023

CERTIFICATE

This is to certify that work embodied in this report entitled **Summer Internship** as carried out by **Rajkumar Munjapara(202003103510281), Mit Lathiya(202003103510299)** in partial fulfilment of the degree of Bachelor of Technology in Information Technology, Chhotubhai Gopalbhai Patel Institute of Technology, UTU, Bardoli during the academic year 2023-2024.

Date:

Place:

Ms. Dhwani Patel	Ms. Purvi H. Tandel
Teaching Assistant,	Head of the Dept.,
Department of IT & CS	Department of IT & CS
CGPIT, UTU, Bardoli.	CGPIT, UTU, Bardoli.

Signature of External Examiner

Chhotubhai Gopalbhai Patel Institute of Technology
UkaTarsadia University
Bardoli – 394350

BVM Infotech
Address: 1051, Silver Business Point
VIP Circle, Uttran, Surat, Gujarat
Email: hr.bvminfotech@gmail.com
Mobile Number: +91 70694 59872

Summer Internship Offer Letter

Rajkumar V. Munjapara
Enrollment No.:- 202003103510281
CGPIT, Uka Tarsadiya University, Bardoli

Date: 23-04-2023

BVM Infotech is pleased to offer you an educational Internship opportunity as **Machine learning** Intern. You will report directly to our Software associate –

As you will be receiving academic credit for this position, you will not be paid. For this position your major duties will be like working on the assigned project by us.

Your schedule will be 5 days a week for daily 8 hours [excluding all weekends] of every month beginning from 01-05-2023[Monday] and will conclude on 30-06-2023 [Friday].

You need to follow each and every rule and regulation of our organization. We

welcome you to BVM Infotech and look forward to a fruitful engagement with you.

Please review and sign to confirm acceptance.

Company Stamp

Sincerely**,**

CEO Signature

[illegible]

Mayur Godhani

BVM Infotech Pvt.Ltd
For BVM INFOTECH
[illegible]
Proprietor

TABLE OF CONTENTS

ACKNOWLEDGEMENT. **V**
List of Figures **2**
Chapter 1: Introduction 3
Error! Bookmark not defined.3
Error! Bookmark not defined.3
Error! Bookmark not defined.3
Chapter 2: Training activities 4
Chapter 3: Modules 5
Error! Bookmark not defined.5
Chapter 4: Implementation 6
Error! Bookmark not defined.6
Error! Bookmark not defined.7
Conclusion and Future Scope **19**
References **20**

LIST OF FIGURES

Figure 4.2.1.Import Libraries and dataset	7
Figure 4.2.2.manage data and encode	8
Figure 4.2.3. training and testing data	9
Figure 4.2.4.feature scaling	10
Figure 4.2.5.Simple Linear Regression	11
Figure 4.2.6.Simple Linear Regression Visualising the Training Set	12
Figure 4.2.7.Simple Linear Regression Visualising the Testing Set	13
Figure 4.2.8.Naive Bayes model	14
Figure 4.2.9.Naive Bayes model visualising the Training set	15
Figure 4.2.10.Naive Bayes model visualising the Testing set	16
Figure 4.2.11.K-Means Clustering	17
Figure 4.2.12. Visualising the Clusters	18

Chapter 1: Introduction

1.1 Overview

Machine learning is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. It is concerned with creating systems that can automatically improve their performance on a task through experience or exposure to data.

1.2 Problem Definition

Machine learning is a subset of artificial intelligence that focuses on developing algorithms capable of learning from data and making predictions or decisions without explicit programming. In this field, the primary problem is to:

- **Model Data Relationships:** Define a mathematical model that captures the relationships between input variables (features) and an output variable (target) based on historical data.
- **Learn from Data:** Use the model to automatically adjust its parameters through training, minimizing errors in predictions or classifications. Generalization: Ensure the model can generalize from the training data to make accurate predictions on new, unseen data.
- **Optimize Performance:** Fine-tune model hyperparameters and preprocessing techniques to achieve the highest predictive accuracy and minimize biases.
- **Ethical Considerations:** Address ethical concerns such as fairness, transparency, and bias mitigation to ensure responsible use of machine learning in various applications.

1.3 Scope/Application

- Healthcare and medicine
- Finance
- E-Commerce
- Natural Language processing
- Computer Vision
- Manufacturing and Industry
- Internet of Things(IOT)
- Energy Utilities
- Education
- Recommendation System
- Agriculture

Chapter 2: Training activities

- **Python Programming:**
  Python programming refers to the practice of writing, testing, and executing code in the Python programming language. Python is a versatile, high-level, and dynamically typed language known for its simplicity and readability. It is widely used in various domains, including web development, data analysis, scientific computing, artificial intelligence, machine learning, and more.

- **Libraries of python:**
  - **Numpy:**numpy-ml is a growing collection of machine learning models, algorithms, and tools written exclusively in NumPy and the Python standard library. The purpose of the project is to provide reference implementations of common machine learning components for rapid prototyping and experimentation.
  - **pandas:**Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays.
  - **Sklearn:**Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.
  - **Matplotlib:**Matplotlib is one of the plotting library in python which is however widely in use for machine learning application with its numerical mathematics extension- Numpy to create static, animated and interactive visualisations.
  - **Scipy:**SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack.

- **Google Collab:**Collab notebooks are Jupyter notebooks that run in the cloud and are highly integrated with Google Drive, making them easy to set up, access, and share. If you are unfamiliar with Google Colab or Jupyter notebooks, please spend some time exploring the Colab welcome site.

Chapter 3: Modules

3.1 System Modules
There isn't a specific "system module" in machine learning as part of a standard library or framework. However, machine learning systems encompass a variety of components and libraries that work together to develop, train, and deploy machine learning models. These components may include:

3.1.1 Data Preparation: Libraries like NumPy, Pandas, and Scikit-Learn are often used to prepare and preprocess data for machine learning tasks.

3.1.2 Machine Learning Frameworks: Frameworks like TensorFlow, PyTorch, and Keras provide tools and APIs for building, training, and deploying machine learning models.

3.1.3 Model Evaluation and Metrics: Scikit-Learn and other libraries offer tools to evaluate the performance of machine learning models using various metrics such as accuracy, precision, recall, and F1-score.

3.1.4 Data Visualization: Libraries like Matplotlib, Seaborn, and Plotly help visualize data, model results, and insights.

3.1.5 Deployment and Serving: Tools like Flask, FastAPI, and cloud-based platforms are used to deploy machine learning models as web services or APIs.

3.1.6 Data Storage and Retrieval: Databases and data storage systems (e.g., SQL, NoSQL databases) are essential for storing and managing large datasets.

3.1.7 Distributed Computing: Distributed computing frameworks like Apache Spark and Dask are used for handling big data and parallel processing in machine learning tasks.

3.1.8 Model Versioning and Management: Tools like MLflow and Kubeflow help manage machine learning model versions, deployments, and monitoring.

3.1.9 GPU and Accelerated Computing:GPUs (Graphics Processing Units) and accelerators like NVIDIA CUDA are often used to speed up training deep learning models.

3.1.10 AutoML (Automated Machine Learning): AutoML tools and platforms automate the process of selecting, training, and tuning machine learning models.

3.1.11 Hyperparameter Tuning: Libraries and platforms like Hyperopt and Optuna automate the search for optimal hyperparameters for machine learning models.

Chapter 4: Implementation

4.1 Hardware and Software requirement:

1)Hardware requirement:

- CPU:Multi Core
- GPU(Optional)
- SSD:256 GB
- RAM: 4 GB(minimum)

2)Software requirement:

- IDE(pycharm/Jupyter lab/visual studio code/google collab)
- Database tools(SQL/MongoDB/SQLite/Excel sheet)
- Cloud Platforms(Microsoft Azure/AWS/IBM watson)

4.2 Snapshots:

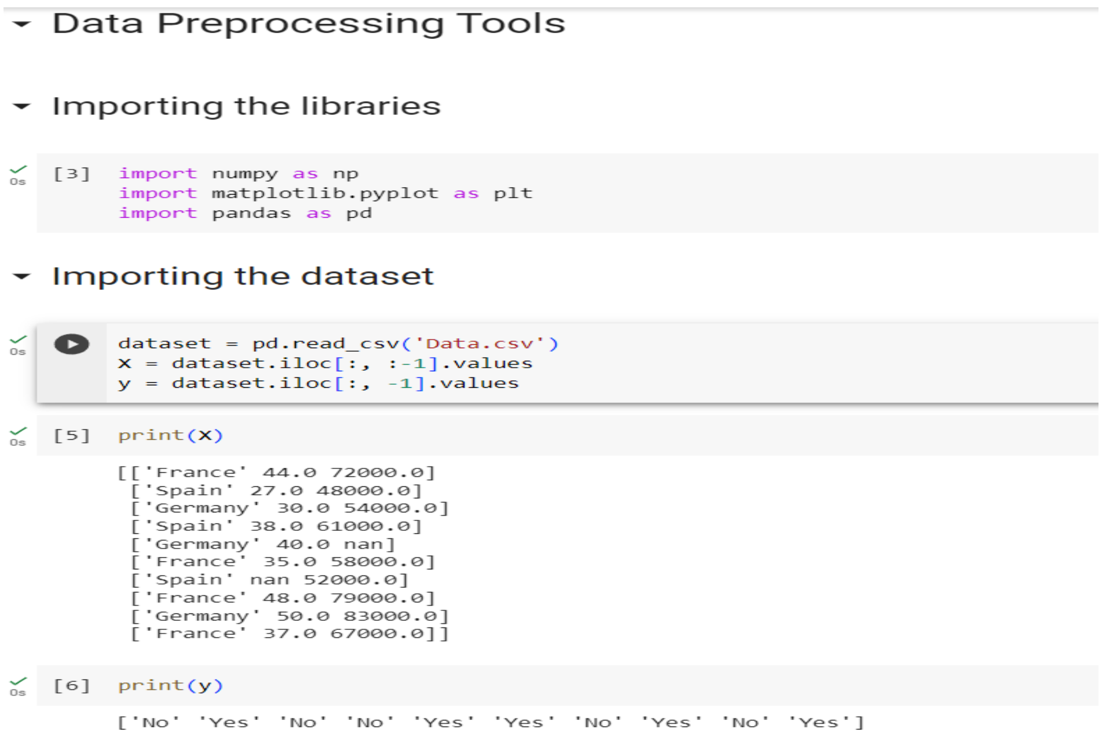
Data Preprocessing part:

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

- Numpy
- matplotlib
- pandas

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

we form a variable of X and Y to categories column in features and labels.



```
[3] import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
```

```
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

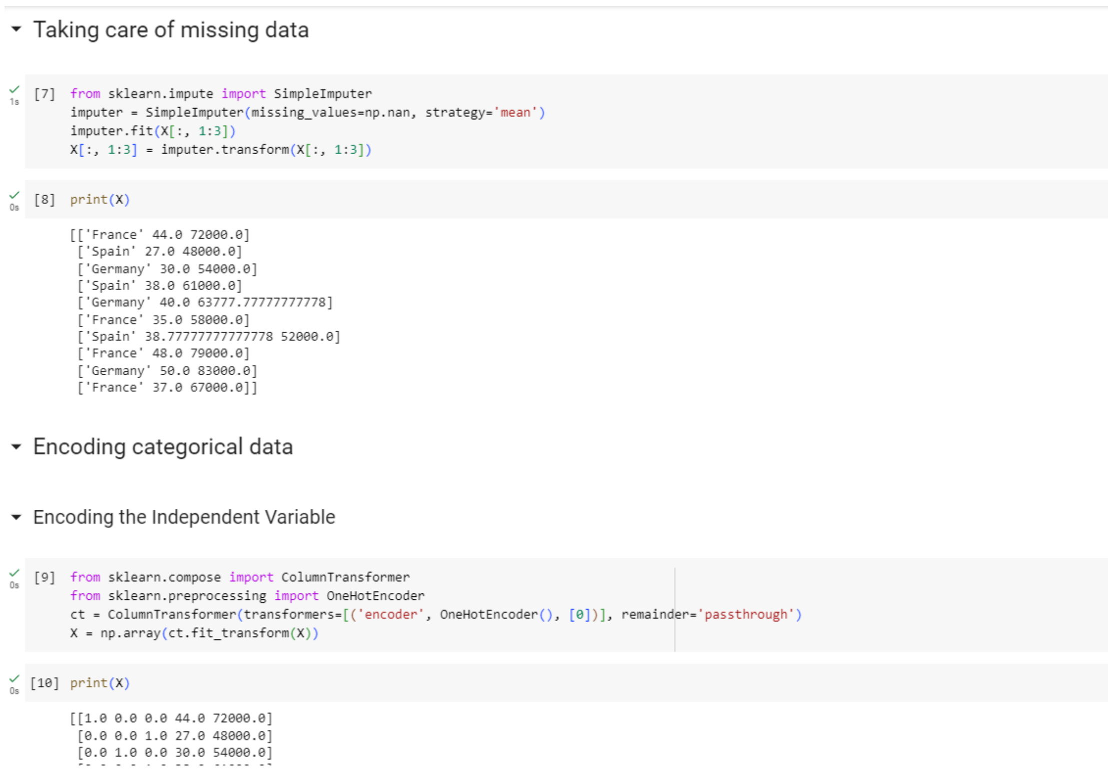
```
[5] print(X)

[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

```
[6] print(y)

['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

Figure 4.2.1 Import Libraries and dataset

Taking care of missing data: SimpleImputer is a class in the sklearn. impute module that can be used to replace missing values in a dataset, using a variety of input strategies. SimpleImputer is designed to work with numerical data, but can also handle categorical data represented as strings.

Encoding Categorical data:if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.



Taking care of missing data

```
[7] from sklearn.impute import SimpleImputer
    imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
    imputer.fit(X[:, 1:3])
    X[:, 1:3] = imputer.transform(X[:, 1:3])
```

```
[8] print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

Encoding categorical data

Encoding the Independent Variable

```
[9] from sklearn.compose import ColumnTransformer
    from sklearn.preprocessing import OneHotEncoder
    ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
    X = np.array(ct.fit_transform(X))
```

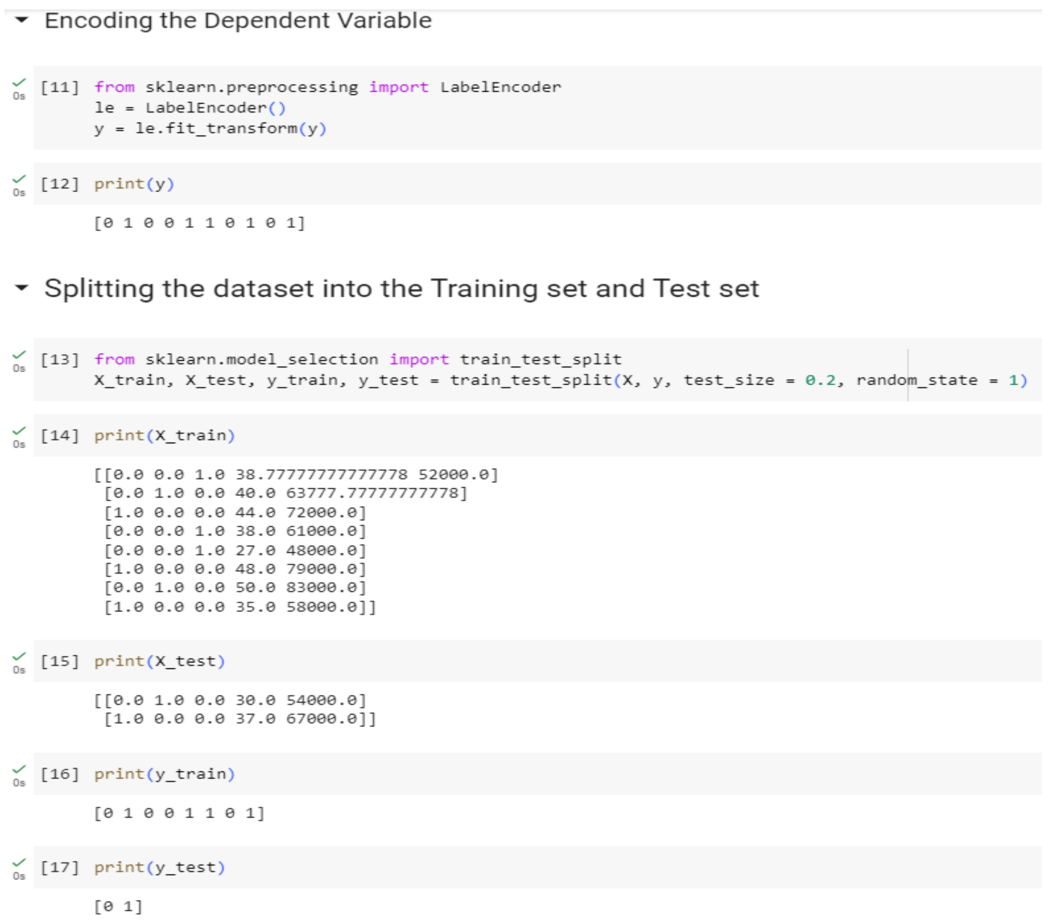
```
[10] print(X)
```

```
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
```

Figure 4.2.2 manage data and encode

Encoding Dependent variable: we have imported **LabelEncoder** class of **sklearn library**. This class has successfully encoded the variables into digits.

Splitting dataset into training and testing set: In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model and the ration between train and testing dataset is ideal 0.2 or 0.4



Encoding the Dependent Variable

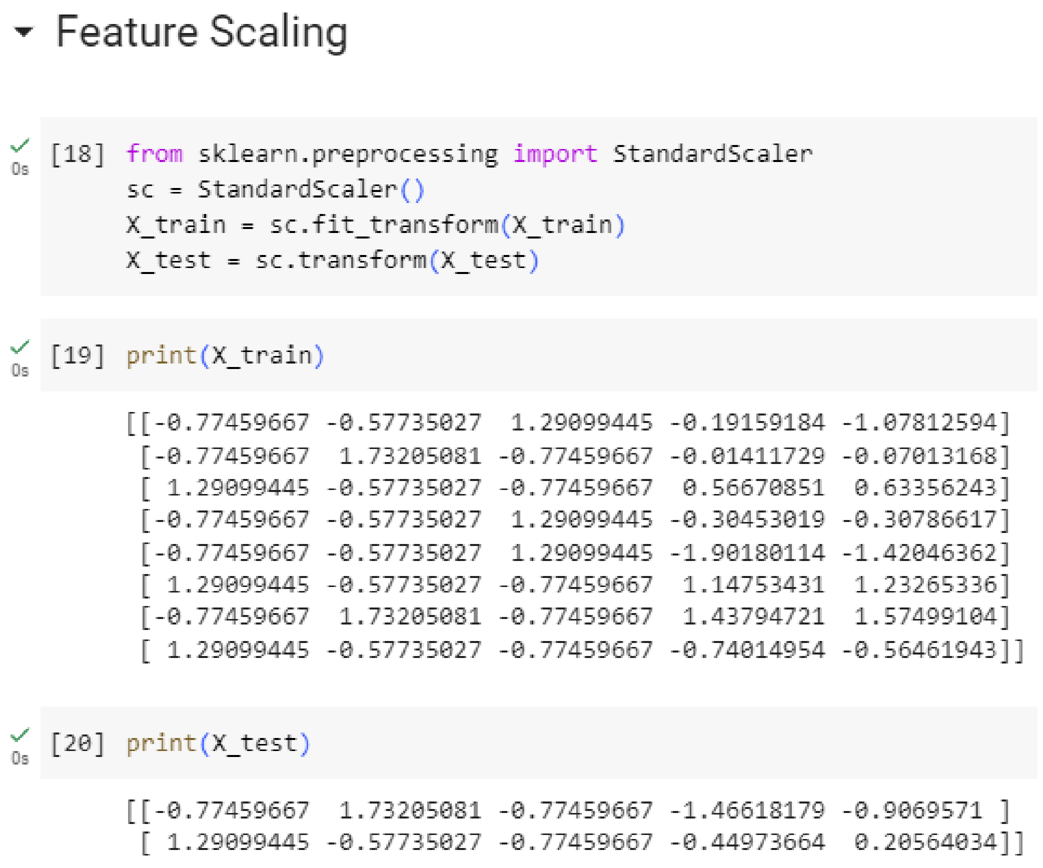
```
[11] from sklearn.preprocessing import LabelEncoder
     le = LabelEncoder()
     y = le.fit_transform(y)

[12] print(y)

     [0 1 0 0 1 1 0 1 0 1]
```

Splitting the dataset into the Training set and Test set

```
[13] 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 = 1)

[14] print(X_train)

     [[0.0 0.0 1.0 38.77777777777778 52000.0]
      [0.0 1.0 0.0 40.0 63777.77777777778]
      [1.0 0.0 0.0 44.0 72000.0]
      [0.0 0.0 1.0 38.0 61000.0]
      [0.0 0.0 1.0 27.0 48000.0]
      [1.0 0.0 0.0 48.0 79000.0]
      [0.0 1.0 0.0 50.0 83000.0]
      [1.0 0.0 0.0 35.0 58000.0]]

[15] print(X_test)

     [[0.0 1.0 0.0 30.0 54000.0]
      [1.0 0.0 0.0 37.0 67000.0]]

[16] print(y_train)

     [0 1 0 0 1 1 0 1]

[17] print(y_test)

     [0 1]
```

Figure 4.2.3 training and testing data

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



▾ Feature Scaling

```
[18] from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

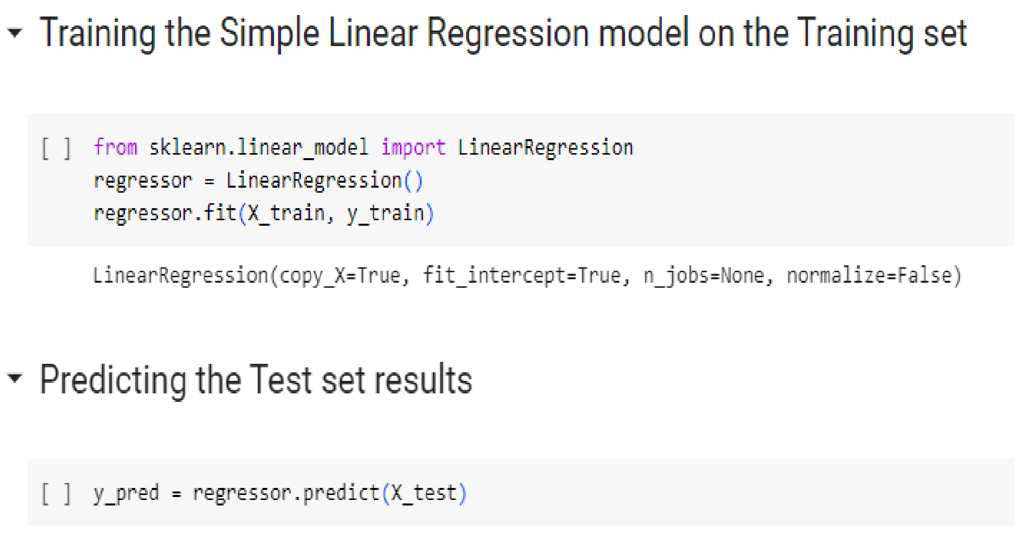
```
[19] print(X_train)

     [[-0.77459667 -0.57735027  1.29099445 -0.19159184 -1.07812594]
      [-0.77459667  1.73205081 -0.77459667 -0.01411729 -0.07013168]
      [ 1.29099445 -0.57735027 -0.77459667  0.56670851  0.63356243]
      [-0.77459667 -0.57735027  1.29099445 -0.30453019 -0.30786617]
      [-0.77459667 -0.57735027  1.29099445 -1.90180114 -1.42046362]
      [ 1.29099445 -0.57735027 -0.77459667  1.14753431  1.23265336]
      [-0.77459667  1.73205081 -0.77459667  1.43794721  1.57499104]
      [ 1.29099445 -0.57735027 -0.77459667 -0.74014954 -0.56461943]]
```

```
[20] print(X_test)

     [[-0.77459667  1.73205081 -0.77459667 -1.46618179 -0.9069571 ]
      [ 1.29099445 -0.57735027 -0.77459667 -0.44973664  0.20564034]]
```

Figure 4.2.4 feature scaling

Regression Part(Simple Linear regression):

Fitting the Simple Linear Regression to the Training Set: Now the second step is to fit our model to the training dataset. To do so, we will import the LinearRegression class of the linear_model library from the scikit-learn. After importing the class, we are going to create an object of the class named as a regressor. The code for this is given below:



Training the Simple Linear Regression model on the Training set

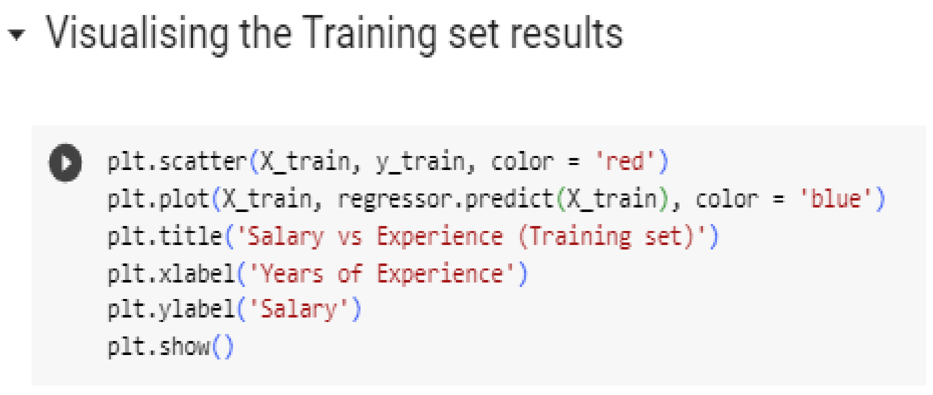
```
[ ] from sklearn.linear_model import LinearRegression
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)

    LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Predicting the Test set results

```
[ ] y_pred = regressor.predict(X_test)
```

Figure 4.2.5 Simple Linear Regression

Visualizing the Training set results: Now in this step, we will visualize the training set result. To do so, we will use the scatter() function of the pyplot library, which we have already imported in the pre-processing step. The **scatter () function** will create a scatter plot of observations.



Visualising the Training set results

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

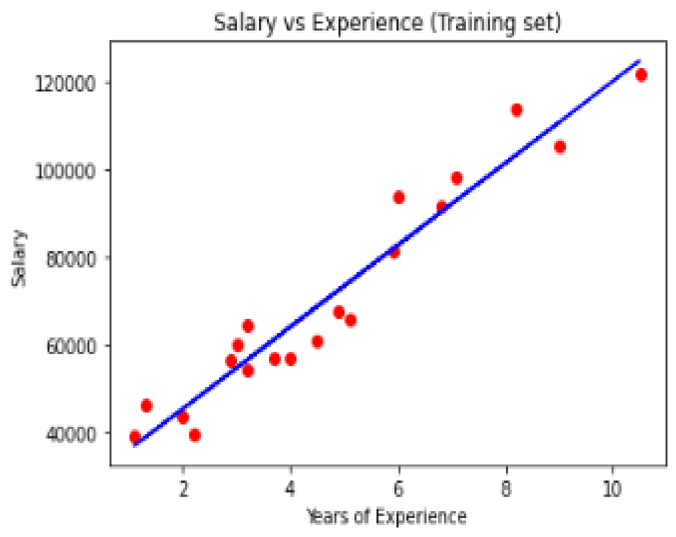
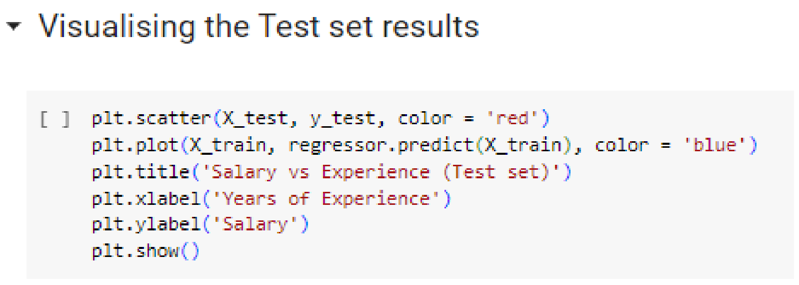


Figure 4.2.6 Simple Linear Regression Visualising the Training Set

Visualizing the Test set results: In the previous step, we have visualized the performance of our model on the training set. Now, we will do the same for the Test set. The complete code will remain the same as the above code, except in this, we will use x_test, and y_test instead of x_train and y_train.



Visualising the Test set results

```
[ ]  plt.scatter(X_test, y_test, color = 'red')
     plt.plot(X_train, regressor.predict(X_train), color = 'blue')
     plt.title('Salary vs Experience (Test set)')
     plt.xlabel('Years of Experience')
     plt.ylabel('Salary')
     plt.show()
```

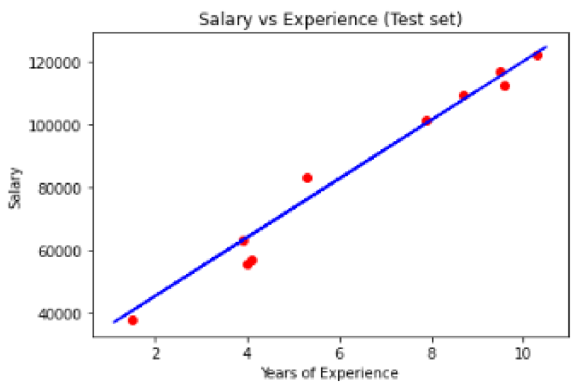


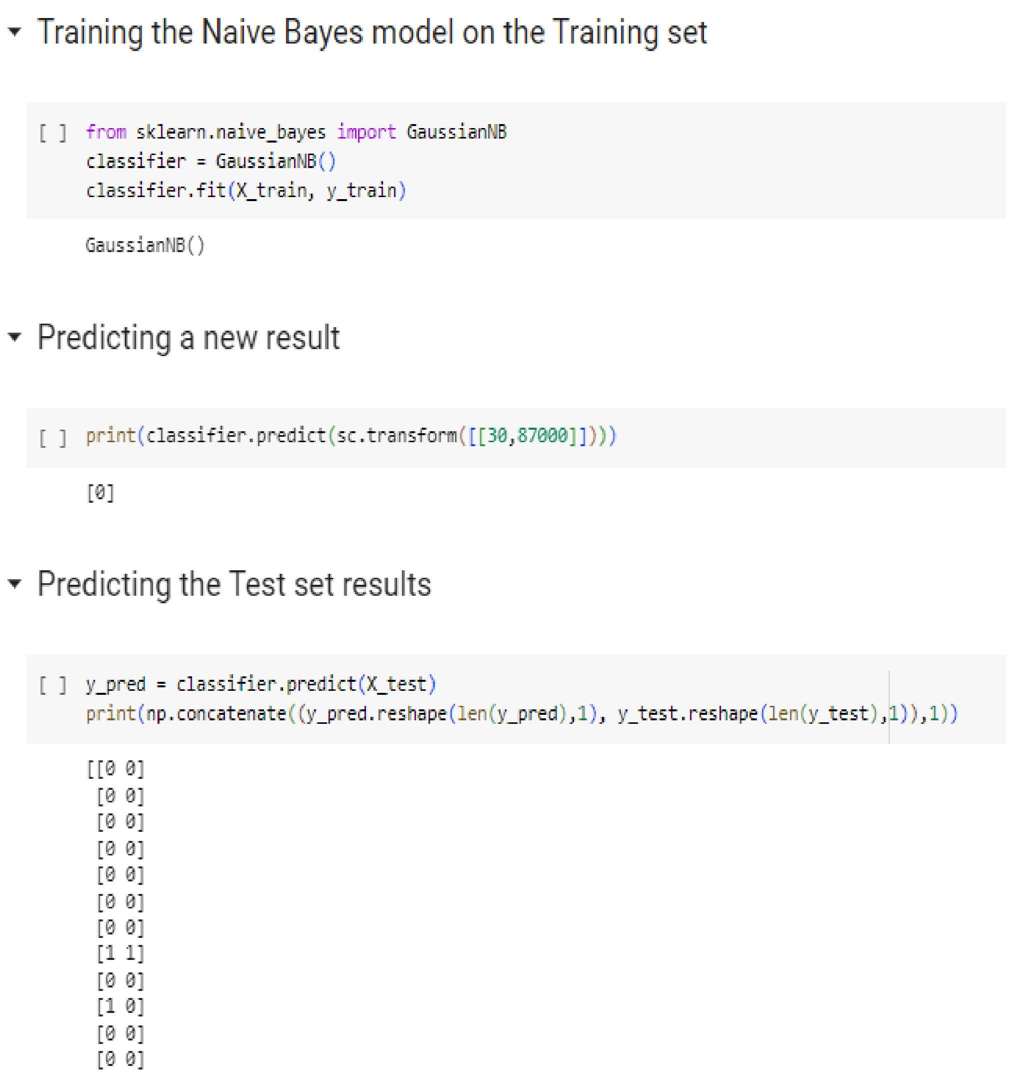
Figure 4.2.7 Simple Linear Regression Visualising the Testing Set

Classification (Naive bayes classifier):

- **Fitting Naive Bayes to the Training Set:**After the pre-processing step,we will fit the Naive Bayes model to the Training set.

Once we build our model then we can predict new result based on training data set.

Predicting Test Set result:once a model is ready we can predict new value(test dataset) based on training dataset and it is denoted as y_pred.



Training the Naive Bayes model on the Training set

```
[ ] from sklearn.naive_bayes import GaussianNB
    classifier = GaussianNB()
    classifier.fit(X_train, y_train)

    GaussianNB()
```

Predicting a new result

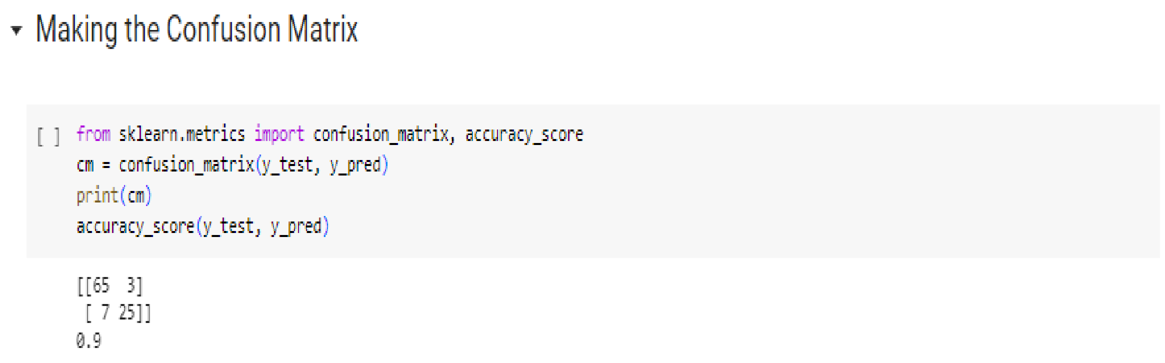
```
[ ] print(classifier.predict(sc.transform([[30,87000]])))

    [0]
```

Predicting the Test set results

```
[ ] y_pred = classifier.predict(X_test)
    print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

    [[0 0]
     [0 0]
     [0 0]
     [0 0]
     [0 0]
     [0 0]
     [0 0]
     [1 1]
     [0 0]
     [1 0]
     [0 0]
     [0 0]
```

Figure 4.2.8 Naive Bayes model

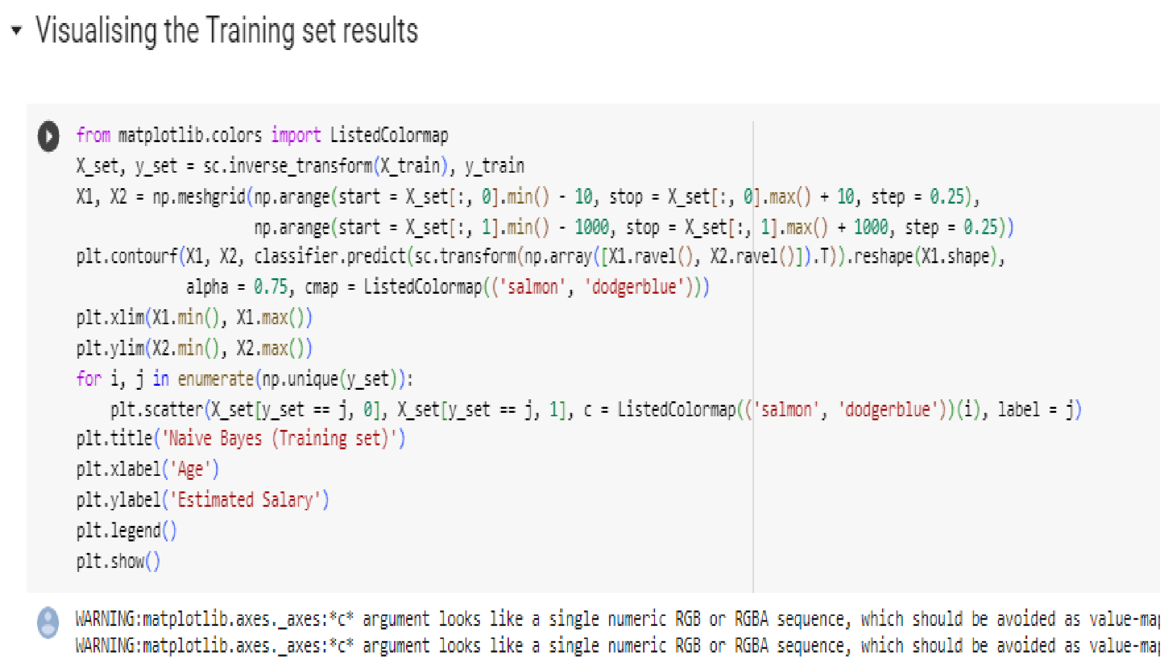
Confusion Matrix:A confusion matrix represents the prediction summary in matrix form. It shows how many prediction are correct and incorrect per class. It helps in understanding the classes that are being confused by model as other class.



Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 7 25]]
0.9
```



Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('salmon', 'dodgerblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('salmon', 'dodgerblue'))(i), label = j)
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
```

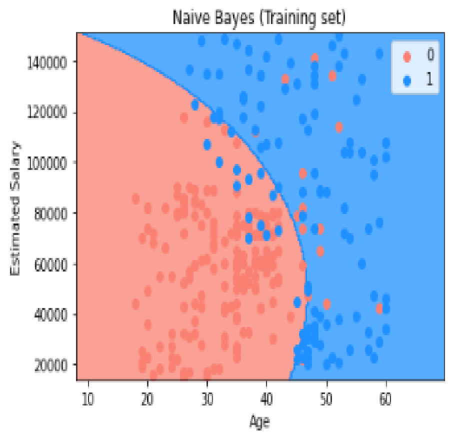
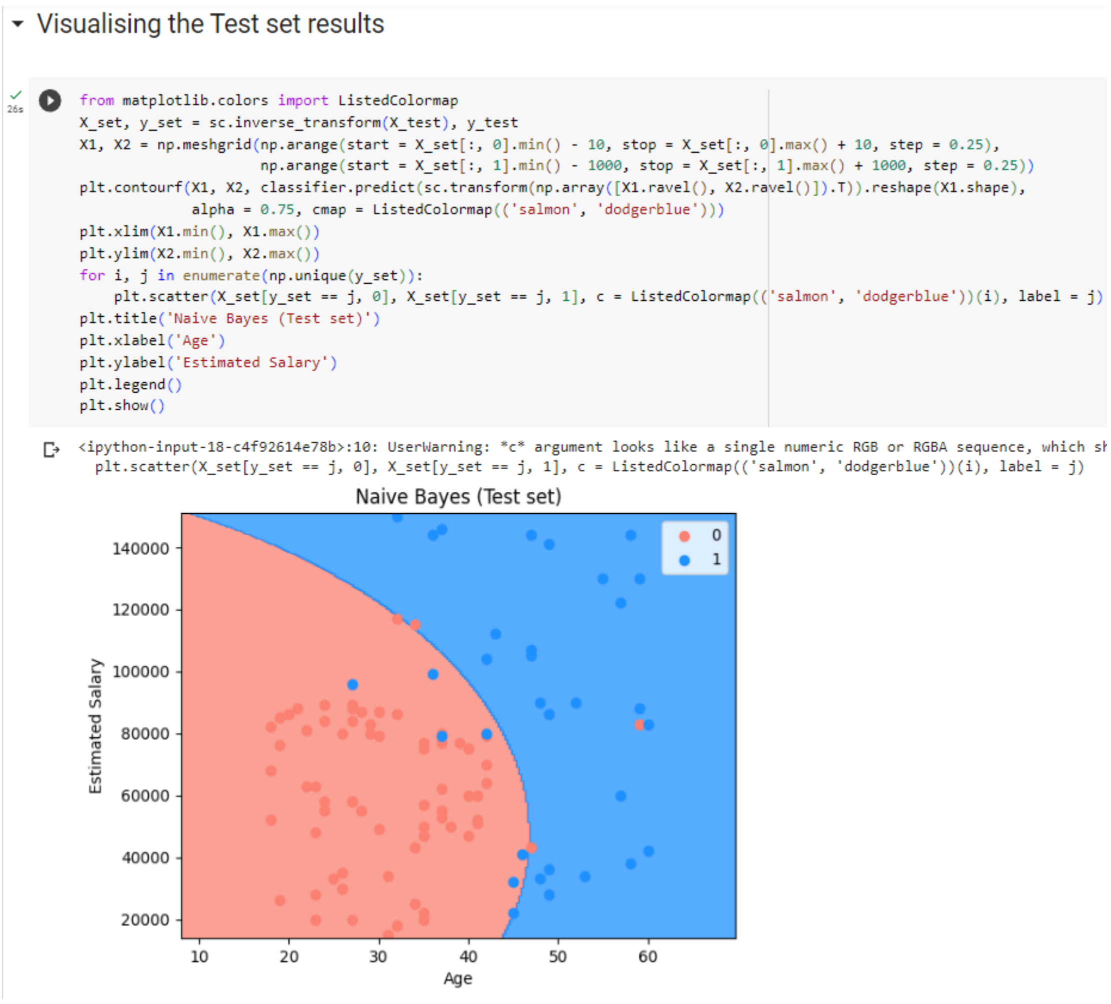


Figure 4.2.9 Naive Bayes Visualising the training Result

Visualizing Data set: The below output is final output for test set data. As we can see the classifier has created a Gaussian curve to divide the "purchased" and "not purchased" variables. There are some wrong predictions which we have calculated in Confusion matrix. But still it is pretty good classifier.



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('salmon', 'dodgerblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('salmon', 'dodgerblue'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-18-c4f92614e78b>:10: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which sh
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('salmon', 'dodgerblue'))(i), label = j)
```

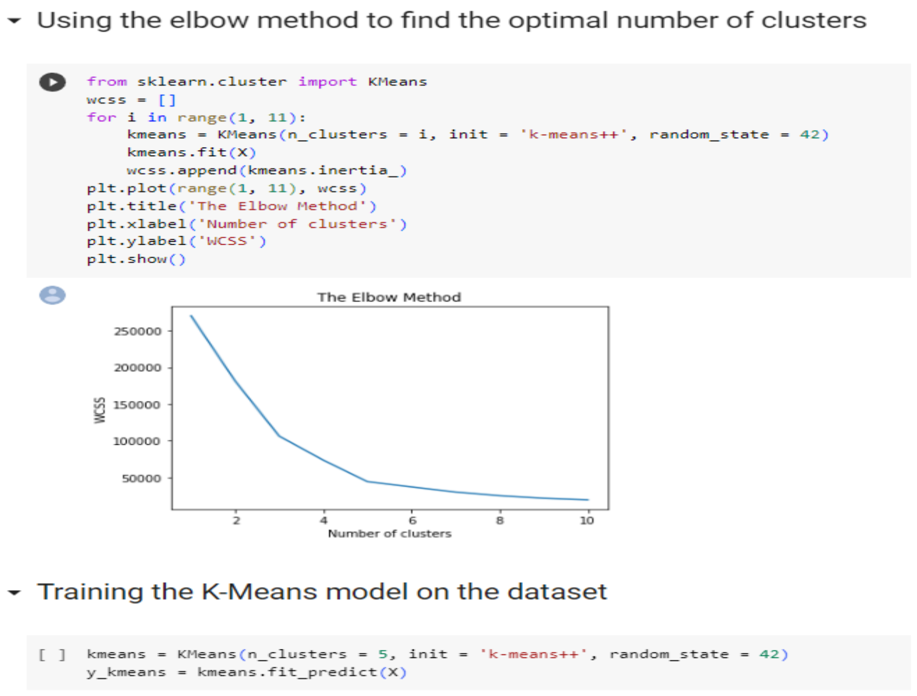
Figure 4.2.10 Naive Bayes Visualising the testing Result

Clustering Part(K Means clustering):

In clustering algorithm , we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10.

Training the K-means algorithm on the training dataset: As we have got the number of clusters, so we can now train the model on the dataset.To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using i, we will use 5, as we know there are 5 clusters that need to be formed. The code is given below:



Using the elbow method to find the optimal number of clusters

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

Training the K-Means model on the dataset

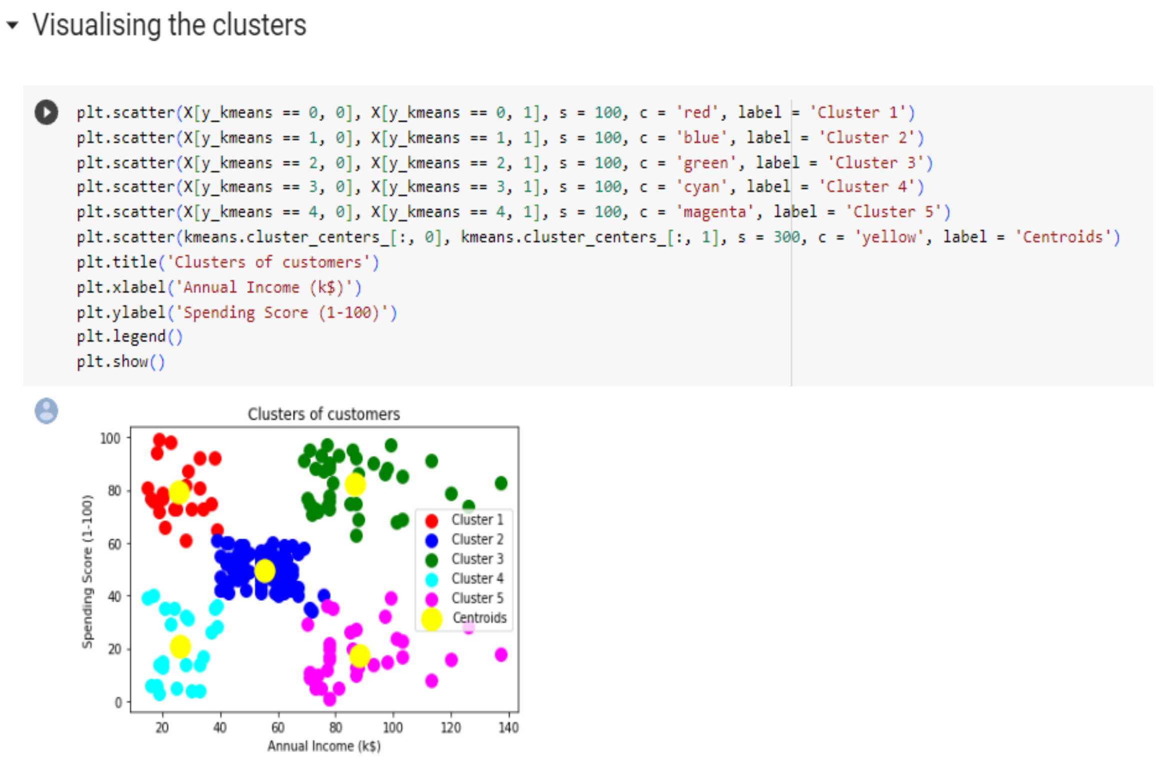
```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

Figure 4.2.11 K-Means Clustering

Visualizing the Clusters:

The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

To visualize the clusters will use scatter plot using mtp.scatter() function of matplotlib.



Visualising the clusters

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Figure 4.2.12 Visualising the Clusters

Conclusion and Future Scope

Conclusion:

In summary, my summer machine learning internship has been an enriching experience. I've acquired practical skills, collaborated effectively, and successfully completed projects. I'm grateful for the knowledge gained and look forward to furthering my career in machine learning.

Future Scope:

To build the Movie recommendation systems and significant and continues to expand as technology evolves and user preferences change. Movie recommendation systems offer valuable benefits to both users and content providers.

1. Enhanced User Experience: Movie recommendation systems help users discover new movies and TV shows based on their preferences. This enhances the overall user experience, making it more personalized and engaging.

2.Increased User Engagement: By providing relevant movie recommendations, platforms can keep users engaged for longer periods, increasing user retention and satisfaction.

3. Content Discovery: These systems assist users in discovering a wide range of content, including hidden gems, niche genres, and foreign films, which they might not have encountered otherwise.

4. Revenue Generation: Content providers and streaming platforms can boost their revenue by retaining subscribers and encouraging more content consumption.

5. Data Utilization: Recommendation systems rely on user data, including viewing history, ratings, and preferences. This data is a valuable resource for understanding user behavior and preferences, which can inform content acquisition and production decisions.

6. Personalization: Movie recommendations can be personalized to each user, improving the likelihood of them finding content they genuinely enjoy.

References

- **javatpoint:** https://www.javatpoint.com/ **(10/05/2023, 11/05/2023, 13/05/2023)**
- **kaggle:** https://www.kaggle.com/ **(15/05/2023, 16/05/2023, 17/05/2023)**
- **scikit-learn:** https://scikit-learn.org/ **(25/05/2023, 26/05/2023, 27/05/2023)**