LAB MANUAL

Name: SHAH HETVI

Enrollment No.: 180850131016

Class: CSE sem. 7

Subject: Machine Learning

Subject Code: 3170724

HJD INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH, KERA
COMPUTER ENGINEERING DEPARTMENT

HJD INSTITUTE OF TECHNICAL
EDUCATION AND RESEARCH, KERA

Approved by AICTE and affiliated to
Gujarat Technological University

CERTIFICATE

This is to certify that **Miss SHAH HETVI Enrollment No 180850131016** of programme **BE 7th SEM Computer Engineering** has satisfactorily completed **her** term work in **Machine Learning(3170724)** for the term ending in 2021-2022.

Miss. Priyanka Ahir
(Lab In-charge)

Mr. Vishal Bhimani
(Head Of Department)

INDEX

PRACTICAL 1 : TO STUDY ABOUT THE TOOLS 1).SCKIT-LEARN 2).JUPYTER LAB

1). SCKIT-LEARN

Where did it come from?

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007.

Later Matthieu Brucher joined the project and started to use it as apart of his thesis work. In 2010 INRIA got involved and the first public release (v0.1 beta) was published in late January 2010.

What is scikit-learn?

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D/3D plotting
- **IPython**: Enhanced interactive console
- **Sympy**: Symbolic mathematics
- **Pandas**: Data structures and analysis

Extensions or modules for SciPy care conventionally named SciKits. As such, the module provides learning algorithms and is named scikit-learn.

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

Although the interface is Python, c-libraries are leverage for performance such as numpy for arrays and matrix operations, LAPACK, LibSVM and the careful use of cython.

Some popular groups of models provided by scikit-learn include:

- **Clustering**: for grouping unlabeled data such as KMeans.
- **Cross Validation**: for estimating the performance of supervised models on unseen data.
- **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior.
- **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- **Ensemble methods**: for combining the predictions of multiple supervised models.
- **Feature extraction**: for defining attributes in image and text data.
- **Feature selection**: for identifying meaningful attributes from which to create supervised models.

- **Parameter Tuning**: for getting the most out of supervised models.
- **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
- **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

2) . JUPYTER LAB

JupyterLab is an interactive development environment for working with notebooks, code and data. **Most importantly, JupyterLab has full support for Jupyter notebooks.** Additionally, JupyterLab enables you to use text editors, terminals, data file viewers, and other custom components side by side with notebooks in a tabbed work area.

JupyterLab provides a high level of integration between notebooks, documents, and activities:

- Drag-and-drop to reorder notebook cells and copy them between notebooks.
- Run code blocks interactively from text files (.py, .R, .md, .tex, etc.).
- Link a code console to a notebook kernel to explore code interactively without cluttering up the notebook with temporary scratch work.
- Edit popular file formats with live preview, such as Markdown, JSON, CSV, Vega, VegaLite, and more.

JupyterLab has been over three years in the making, with over 11,000 commits and 2,000 releases of npm and Python packages.

The Evolution of the Jupyter Notebook

Project Jupyter exists to develop open-source software, open standards, and services for interactive and reproducible computing.

Since 2011, the Jupyter Notebook has been our flagship project for creating reproducible computational narratives. The Jupyter Notebook enables users to create and share documents that combine live code with narrative text, mathematical equations, visualizations, interactive controls, and other rich output. It also provides building blocks for interactive computing with data: a file browser, terminals, and a text editor.

The Jupyter Notebook has become ubiquitous with the rapid growth of data science and machine learning and the rising popularity of open-source software in industry and academia:

- Today there are millions of users of the Jupyter Notebook in many domains, from data science and machine learning to music and education. Our international community comes from almost every country on earth.[1]
- The Jupyter Notebook now supports over 100 programming languages, most of which have been developed by the community.

- There are over 1.7 million public Jupyter notebooks hosted on GitHub. Authors are publishing Jupyter notebooks in conjunction with scientific research, academic journals, data journalism, educational courses, and books.

At the same time, the community has faced challenges in using various software workflows with the notebook alone, such as running code from text files interactively. The classic Jupyter Notebook, built on web technologies from 2011, is also difficult to customize and extend.

- You can arrange multiple documents and activities side by side in the work area using tabs and splitters. Documents and activities integrate with each other, enabling new workflows for interactive computing, for example:

- Code Consoles provide transient scratchpads for running code interactively, with full support for rich output. A code console can be linked to a notebook kernel as a computation log from the notebook, for example.

- Kernel-backed documents enable code in any text file (Markdown, Python, R, LaTeX, etc.) to be run interactively in any Jupyter kernel.

- Notebook cell outputs can be mirrored into their own tab, side by side with the notebook, enabling simple dashboards with interactive controls backed by a kernel.

- Multiple views of documents with different editors or viewers enable live editing of documents reflected in other viewers. For example, it is easy to have live preview of Markdown, Delimiter-separated Values, or Vega/Vega-Lite documents.

JupyterLab also offers a unified model for viewing and handling data formats. JupyterLab understands many file formats (images, CSV, JSON, Markdown, PDF, Vega, Vega-Lite, etc.) and can also display rich kernel output in these formats. See File and Output Formats for more information.

To navigate the user interface, JupyterLab offers customizable keyboard shortcuts and the ability to use key maps from vim, emacs, and Sublime Text in the text editor.

JupyterLab extensions can customize or enhance any part of JupyterLab, including new themes, file editors, and custom components.

JupyterLab is served from the same server and uses the same notebook document format as the classic Jupyter Notebook.

PRACTICAL 2 : WAP TO IMPLEMENT LINEAR REGRESSION

Simple Linear Regression

Code:

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
```

```
 # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

Output:

Estimated coefficients:

b_0 = -0.0586206896552

b_1 = 1.45747126437

PRACTICAL 3 : WAP TO IMPLEMENT POLYNOMIAL ALGORTHM

Code

```
def add(A, B, m, n):

    size = max(m, n);
    sum = [0 for i in range(size)]

    # Initialize the product polynomial
    for i in range(0, m, 1):
        sum[i] = A[i]

    # Take ever term of first polynomial
    for i in range(n):
        sum[i] += B[i]

    return sum

# A utility function to print a polynomial
def printPoly(poly, n):
    for i in range(n):
        print(poly[i], end = "")
        if (i != 0):
            print("x^", i, end = "")
        if (i != n - 1):
            print(" + ", end = "")

# Driver Code
if __name__ == '__main__':

    # The following array represents
    # polynomial 5 + 10x^2 + 6x^3
    A = [5, 0, 10, 6]

    # The following array represents
    # polynomial 1 + 2x + 4x^2
    B = [1, 2, 4]
    m = len(A)
    n = len(B)

    print("First polynomial is")
    printPoly(A, m)
    print("\n", end = "")
    print("Second polynomial is")
    printPoly(B, n)
    print("\n", end = "")
    sum = add(A, B, m, n)
    size = max(m, n)
```

```
print("sum polynomial is")
    printPoly(sum, size)
```

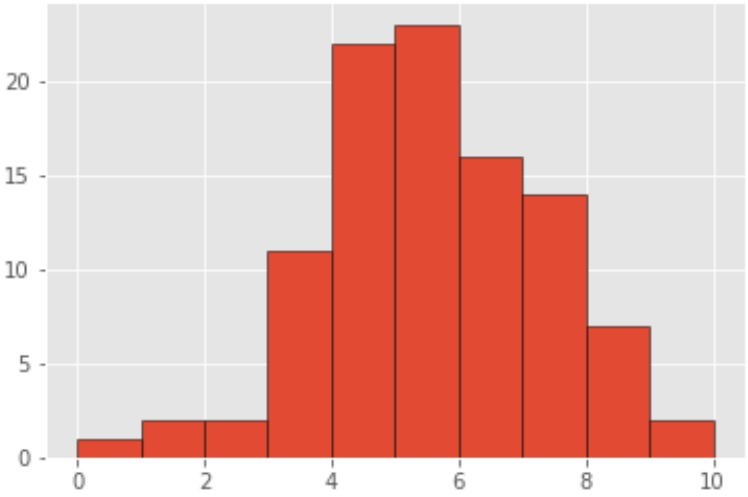
Output:

```
First polynomial is
5 + 0x^1 + 10x^2 + 6x^3
Second polynomial is
1 + 2x^1 + 4x^2
Sum polynomial is
6 + 2x^1 + 14x^2 + 6x^3
```

PRACTICAL 4 : WAP TO CALCULATE MEAN AND MEDIAN FROM DATASET AND IMPLEMENT HISTOGRAM.

Code

```
import numpy as np
from scipy import stats

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
matplotlib.style.use('ggplot')

np.random.seed(1)
data = np.round(np.random.normal(5, 2, 100))
plt.hist(data, bins=10, range=(0,10), edgecolor='black')
plt.show()
```



MEAN

Numpy implements a `mean` function for calculating the mean:

```
mean = np.mean(data)
mean
```

Output:

5.0999999999999996

MEDIAN

Numpy also implements a `median` function for calculating the median:

```
np.median(data)
```

Output:

5.0

PRACTICAL 5 : WAP TO CALCULATE STANDARD DEVIATION AND VARIANCE AND FIND OUTLIERS

Code

```
import numpy as np
from scipy import stats

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
matplotlib.style.use('ggplot')

np.random.seed(1)
data = np.round(np.random.normal(5, 2, 100))
plt.hist(data, bins=10, range=(0,10), edgecolor='black')
plt.show()
```

VARIANCE:
Numpy implements the variance as a function np.var()

```
np.var(data)
```

Output:
3.06999999999998

STANDARD DEVIATION:
This is implemented in Numpy as np.std()

```
np.std(data)
```

Output:
1.7521415467935231

FIND OUTLIERS:

Code:

```
# Importing
import sklearn
from sklearn.datasets import load_boston
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
bos_hou = load_boston()
```

```
# Create the dataframe
column_name = bos_hou.feature_names
df_boston = pd.DataFrame(bos_hou.data)
df_boston.columns = column_name
df_boston.head()
```

Output:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

PRACTICAL 6 : HOUSE PRICE PREDICTION BASED ON FEATURES AND TARGET VARIABLES.

```
# load the libraries
import sklearn
from sklearn.metrics import mean_squared_error as MSE
from sklearn.pre-processing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import load_boston

import pandas as pd
import numpy as np

# load the data from load_boston function
data = load_boston()

# store the list of features in the dataset using feature_names
array = data.feature_names
array = np.append(array,['MEDV'])

# separate the data and target values in the dataset
data, target = data.data, data.target

# split the data into training and test set to avoid overfitting
Xtrain, Xtest, Ytrain, Ytest = train_test_split(data,target,test_size=0.3)
print(Xtrain.shape,Ytrain.shape)

TARGET_PRICE = 'MEDV'

# instantiate the LabelEncoder
le = LabelEncoder()
# the data is stored in a csv file for your reference
df = pd.read_csv('./boston.csv')

y = df[TARGET_PRICE]
# drop the target price column, as this is the training_data
df = df.drop([TARGET_PRICE], axis=1)
# tranform the categorical values into numerical values
df['CHAS'] = le.fit_transform(df['CHAS'])
        x = df
# instantiate the RandomForestRegressor with all the processors available
dt = RandomForestRegressor(criterion='mae',n_jobs=-1, n_estimators=10,max_depth=6,
min_samples_leaf=1, random_state=3
)# fit the random forest to training data
dt.fit(Xtrain,Ytrain)
# predict the output for test data
y_predicted = dt.predict(Xtest)
# find the accuracy of predcition using training data
accuracy = dt.score(Xtest,Ytest)
```

```
# compute the Mean Square error using MSE function from sklearn.metrics module.
MSE_score = MSE(Ytest,y_predicted)
# print the final results
print("Training Accuracy:",dt.score(Xtrain,Ytrain))
print("Testing Accuracy:",accuracy)
print("Mean Squared Error",MSE_score.mean())
```

Output:

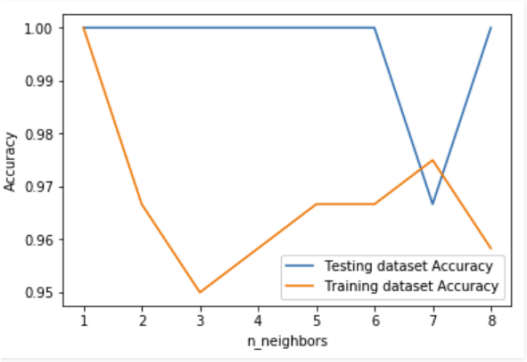
```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'  'B' 'LSTAT']
(354, 13) (354,)
Training Accuracy: 0.9365453347551779
Testing Accuracy: 0.799293144193572
Mean Squared Error 18.157684374999995
```

PRACTICAL 7 : WAP TO IMPLEMENT K NEAREST NEIGHBOUR ALGORITHM

Code:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

Output



PRACTICAL 8 : WAP TO DEAL WITH MISSING DATA IN ML

Code:

```
""" PART 1 Importing Libraries """
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

""" PART 2 Importing Data """
data_sets = pd.read_csv('C:\\Users\\Admin\\Desktop\\Data_for_Missing_Values.csv')

print ("Data Head : \n", data_sets.head())
print ("\n\nData Describe : \n", data_sets.describe())

""" PART 3 Input and Output Data """

# All rows but all columns except last
X = data_sets.iloc[:, :-1].values

# TES
# All rows but only last column
Y = data_sets.iloc[:, 3].values

print("\n\nInput : \n", X)
print("\n\nOutput: \n", Y)


""" PART 4 Handling the missing values """

# We will use sklearn library >> preprocessing package
# Imputer class of that package
from sklearn.preprocessing import Imputer

# Using Imputer function to replace NaN
# values with mean of that parameter value
imputer = Imputer(missing_values = "NaN", strategy = "mean", axis = 0)

# Fitting the data, function learns the stats
imputer = imputer.fit(X[:, 1:3])

# fit_transform() will execute those
# stats on the input ie. X[:, 1:3]
X[:, 1:3] = imputer.fit_transform(X[:, 1:3])

# filling the missing value with mean
print("\n\nNew Input with Mean Value for NaN : \n", X)
```

Output:

Data Head :

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes

Data Describe :

	Age	Salary
count	9.000000	9.000000
mean	38.777778	63777.777778
std	7.693793	12265.579662
min	27.000000	48000.000000
25%	35.000000	54000.000000
50%	38.000000	61000.000000
75%	44.000000	72000.000000
max	50.000000	83000.000000

Input :

```
[['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]]
```

Output:

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

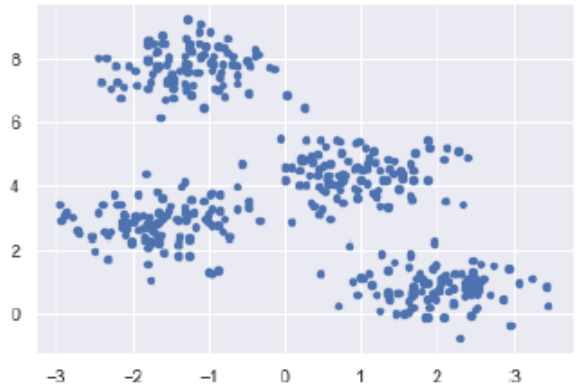
New Input with Mean Value for NaN :

[['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]]

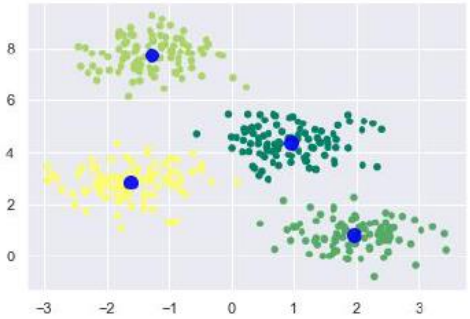
PRACTICAL 9 : APPLYING K-MEANS ALGORITHM FOR MALWARE ANALYSIS

Code:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans

# The following code will generate the 2D, containing four blobs
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples = 400, centers = 4, cluster_std = 0.60, random_state = 0)

# Next, the following code will help us to visualize the dataset −
plt.scatter(X[:, 0], X[:, 1], s = 20);
plt.show()
```




```
#Next, make an object of KMeans along with providing number of clusters, train the model and do
# the prediction as follows −
kmeans = KMeans(n_clusters = 4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Now, with the help of following code we can plot and visualize the cluster's centers picked by k-
# means Python estimator −
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples = 400, centers = 4, cluster_std = 0.60, random_state = 0)

# Next, the following code will help us to visualize the dataset −
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 20, cmap = 'summer')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c = 'blue', s = 100, alpha = 0.9);
plt.show()
```



PRACTICAL 10 : WAP OF SPAM CLASSIFICATION USING SVM

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn import svm

data = pd.read_csv('spam.csv')
```

	Label	EmailText
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ã□_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

```
# Splitting our data into X and y.
X = data['EmailText'].values
y = data['Label'].values

# Splitting our data into training and testing.
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)

# Converting String to Integer
cv = CountVectorizer()
X_train = cv.fit_transform(X_train)
X_test = cv.transform(X_test)

Applying SVM algorithm
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
```

Accuracy
print(classifier.score(X_test,y_test))
>> 0.9766816143497757

count_Class=pd.value_counts(data["v1"], sort= True)
count_Class.plot(kind= 'bar', color= ["blue", "orange"])
plt.title('Bar chart')
plt.show()