ABSTRACT

The machine learning field, which can be briefly defined as enabling computers to make successful predictions using past experiences, has exhibited an impressive development recently with the help of the rapid increase in the storage capacity and processing power of computers. Together with many other disciplines, machine learning methods have been widely employed in bioinformatics. The difficulties and cost of biological analyses have led to the development of sophisticated machine learning approaches for this application area. In this chapter, we first review the fundamental concepts of machine learning such as feature assessment, unsupervised versus supervised learning and types of classification. Then, we point out the main issues of designing machine learning experiments and their performance evaluation. Finally, we introduce some supervised learning methods.

TABLE OF CONTENT

Introduction to Machine learning

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.

Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field.

Machine learning (ML) is a category of algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available

How Machine Learning works?

Machine learning algorithms are often categorized as supervised or unsupervised. Supervised algorithms require a data scientist or data analyst with machine learning skills to provide both input and desired output, in addition to furnishing feedback about the accuracy of predictions during

Unsupervised algorithms do not need to be trained with desired outcome data. Instead, they use an iterative approach called deep learning to review data and arrive at conclusions. Unsupervised learning algorithms -- also called neural networks -- are used for more complex processing tasks than supervised learning systems, including image recognition, speech-to-text and natural language generation. These neural networks work by combing through millions of examples of training data and automatically identifying often subtle correlations between many variables. Once trained, the algorithm can use its bank of associations to interpret new data. These algorithms have only become feasible in the age of big data, as they require massive amounts of training data model should analyze and use to develop predictions. Once training is complete, the algorithm will apply what was learned to new data.

Unsupervised algorithms do not need to be trained with desired outcome data. Instead, they use an iterative approach called deep learning to review data and arrive at conclusions. Unsupervised learning algorithms -- also called neural networks -- are used for more complex processing tasks than supervised learning systems, including image recognition, speech-to-text and natural language generation. These neural networks work by combing through millions of examples of training data and automatically identifying often subtle correlations between many variables. Once trained, the algorithm can use its bank of associations to interpret new data. These algorithms have only become feasible in the age of big data, as they require massive amounts of training data.

Advantages of Machine Learning

1. Trends and Patterns Are Identified With Ease

Machine Learning is adept at reviewing large volumes of data and identifying patterns and trends that might not be apparent to a human. For instance, a machine learning program may successfully pinpoint a causal relationship between two events. This makes the technology highly effective at data mining, particularly on a continual, ongoing basis, as would be required for an algorithm.

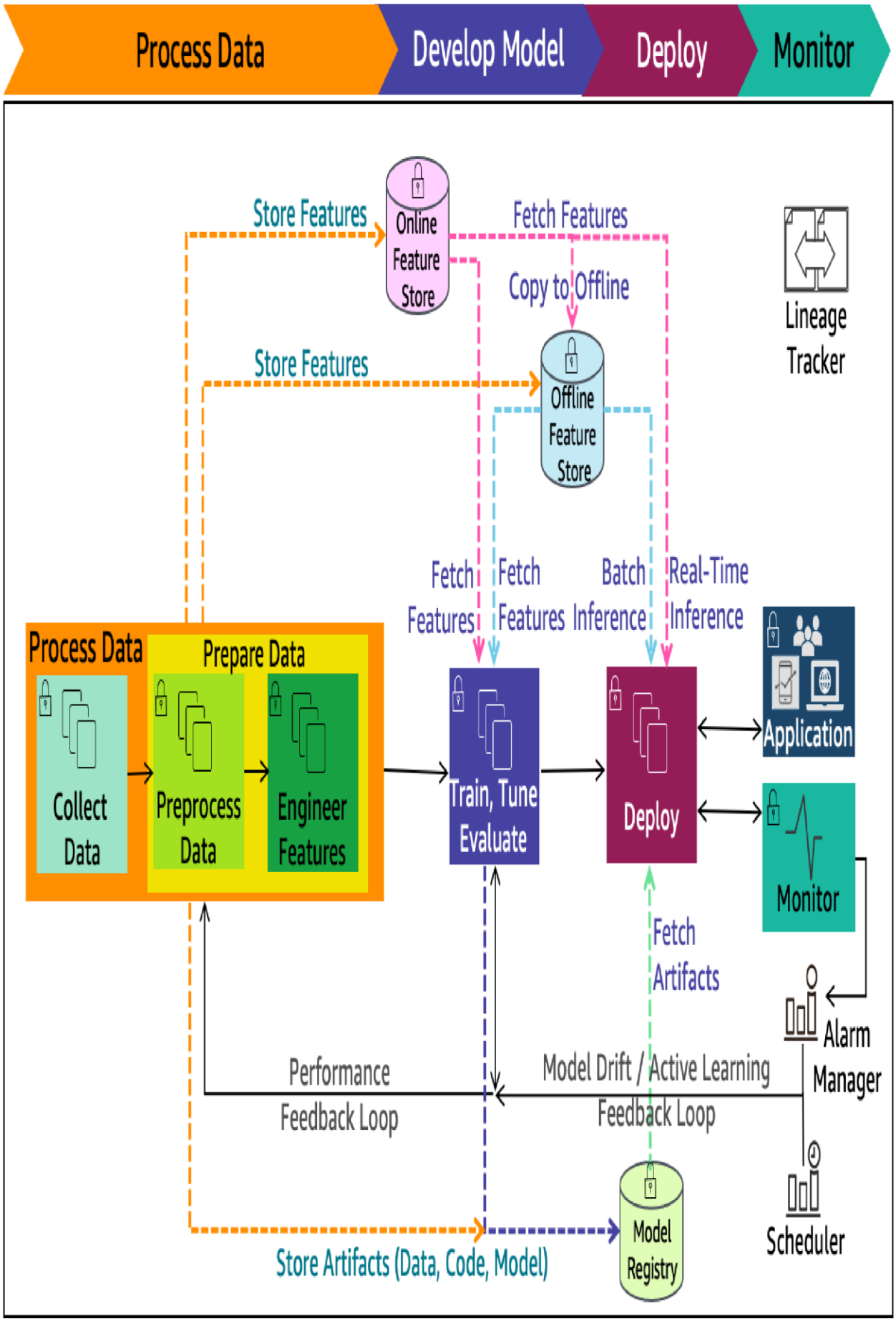
2. Machine Learning Improves Over Time

Machine Learning technology typically improves efficiency and accuracy over time thanks to the ever-increasing amounts of data that are processed. This gives the algorithm or program more "experience," which can, in turn, be used to make better decisions or predictions.

A great example of this improvement over time involves weather prediction models. Predictions are made by looking at past weather patterns and events; this data is then used to determine what's most likely to occur in a particular scenario. The more data you have in your data set, the greater the accuracy of the model.

Architecture of ML Model

Different Machine Learning architectures are needed for different purposes. A car is a motor vehicle that gets you to work and to do road trips, a tractor tugs a plough, an 18-wheeler transports lots of merchandise.

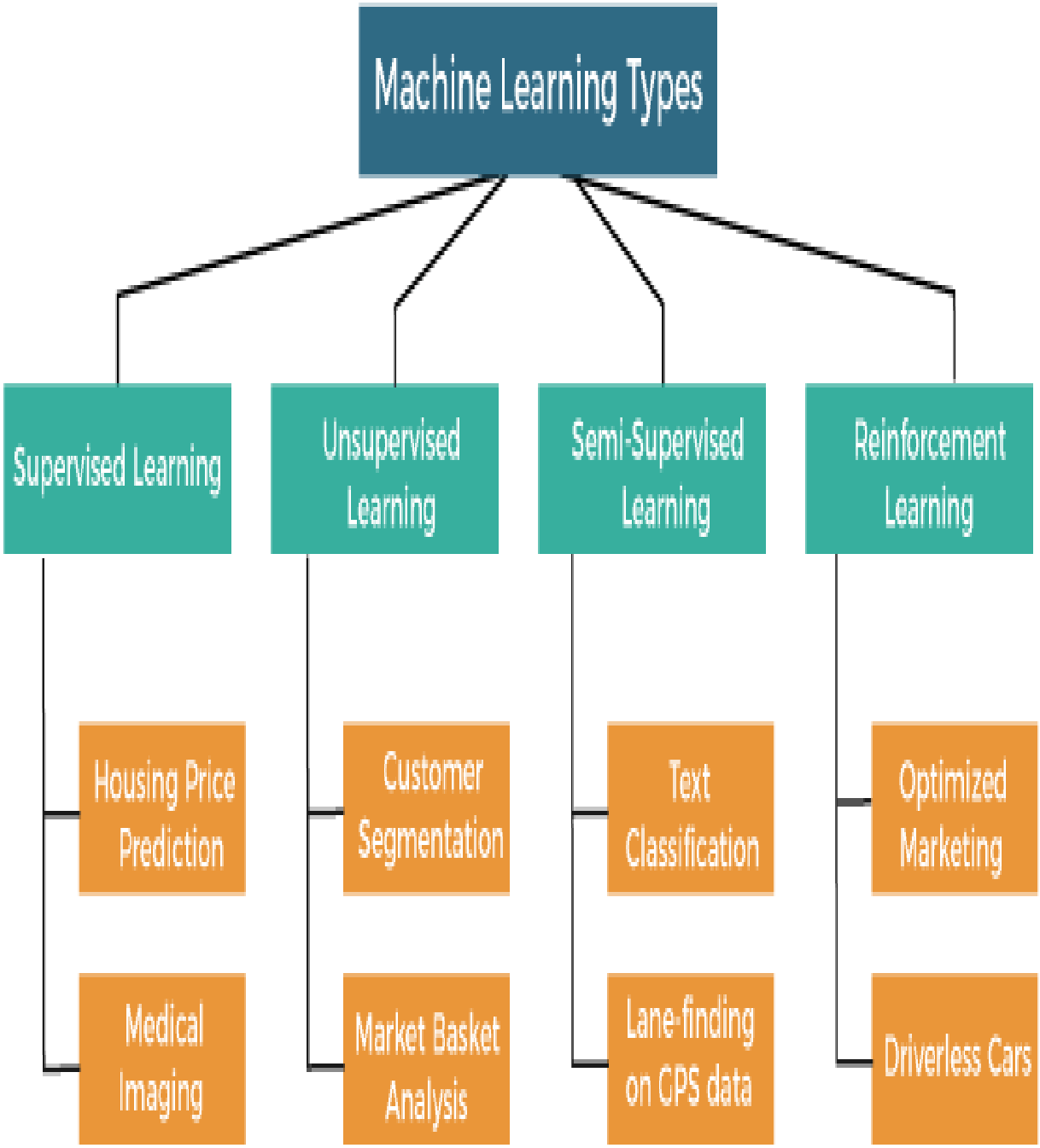

Process Data
Develop Model
Deploy
Monitor
Store Features
Online Feature Store
Fetch Features
Copy to Offline
Lineage Tracker
Store Features
Offline Feature Store
Fetch Features
Fetch Features
Batch Inference
Real-Time Inference
Process Data
Prepare Data
Collect Data
Preprocess Data
Engineer Features
Train, Tune Evaluate
Deploy
Application
Monitor
Fetch Artifacts
Alarm Manager
Performance Feedback Loop
Model Drift / Active Learning Feedback Loop
Scheduler
Model Registry
Store Artifacts (Data, Code, Model)

Types of machine learning

Machine learning is a subset of AI, which enables the machine to automatically learn from data, improve performance from past experiences, and make predictions. Machine learning contains a set of algorithms that work on a huge amount of data. Data is fed to these algorithms to train them, and on the basis of training, they build the model & perform a specific task.These ML algorithms help to solve different business problems like Regression, Classification, Forecasting, Clustering, and Associations, etc.

Based on the methods and way of learning, machine learning is divided into mainly four types, which are:

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-Supervised Machine Learning
4. Reinforcement Learning



5.

Types of machine learning algorithm

Machine Learning algorithms are the programs that can learn the hidden patterns from the data, predict the output, and improve the performance from experiences on their own. Different algorithms can be used in machine learning for different tasks, such as simple linear regression that can be used **for prediction problem**s like **stock market prediction,** and **the KNN algorithm can be used for classification problems**.

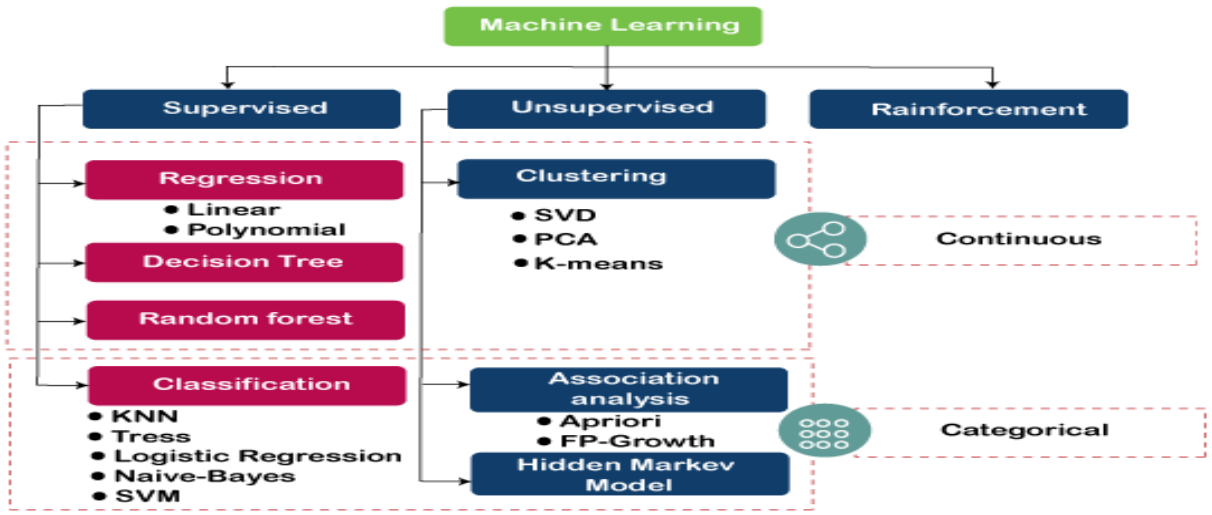
In this topic, we will see the overview of some popular and most commonly used machine learning algorithms along with their use cases and categories.

Types of Machine Learning Algorithms

Machine Learning Algorithm can be broadly classified into three types:

1. **Supervised Learning Algorithms**
2. **Unsupervised Learning Algorithms**
3. **Reinforcement Learning algorithm**

The below diagram illustrates the different ML algorithm, along with the categories:



Introduction of NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

a powerful N-dimensional array object
sophisticated (broadcasting) functions tools
for integrating C/C++ and Fortran code
useful linear algebra, Fourier transform, and random number capabilities
Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.
NumPy is licensed under the BSD license, enabling reuse with few restrictions. The core functionality of NumPy is its "ND array", for n-dimensional array, data structure. These arrays are stride views on memory. In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type. NumPy has built-in support for memory- mapped arrays.
Here are some functions that are defined in this NumPy Library.

1. zeros (shape [, dtype, order]) - Return a new array of given shape and type, filled with zeros.
2. array (object [, dtype, copy, order, lubok, ndim]) - Create an array
3. as array (a [, dtype, order]) - Convert the input to an array.
4. As an array (a [, dtype, order]) - Convert the input to an ND array, but pass ND array subclasses througharange([start,] stop [, step,] [, dtype]) - Return evenly spaced values within a given interval.
6. linspace (start, stop [, num, endpoint, ...]) - Return evenly spaced numbers over a specified interval. etc. there many functions which are used to perform specified operation on the given input values

Implementation: import
numpy as np

Introduction of pandas

Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc. In this tutorial, we will learn the various features of Python Pandas and how to use them in practice.
The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.
In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.
Prior to Pandas, Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.
Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Key Features of Pandas
Fast and efficient DataFrame object with default and

customized indexing.

Tools for loading data into in-memory data objects from

different file formats.

Data alignment and integrated handling of missing data.

Reshaping and pivoting of date sets.

Label-based slicing, indexing and subsetting of large data

sets.

Columns from a data structure can be deleted or inserted.

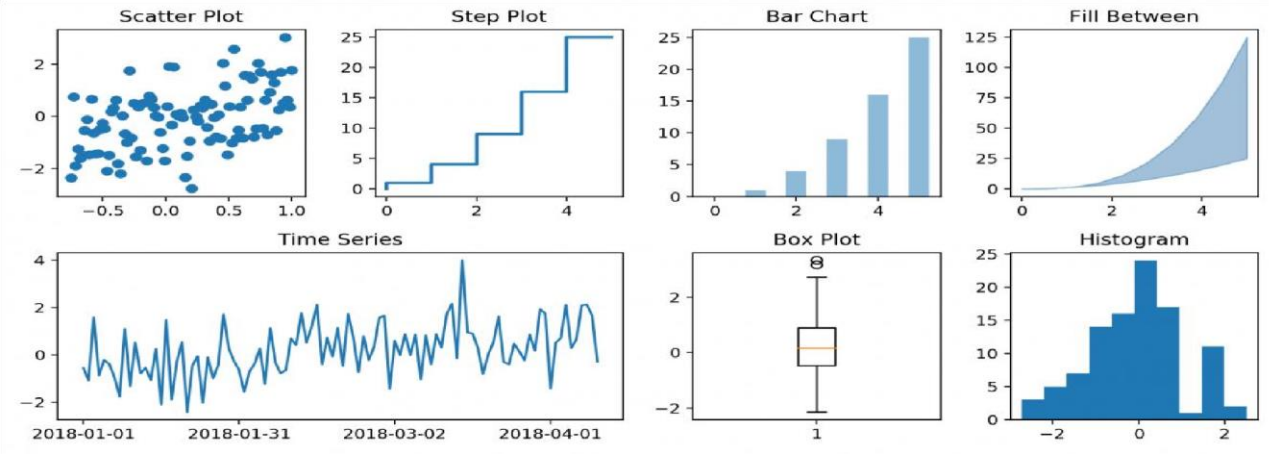
Group by data for aggregation and transformations.

High performance merging and joining of data.

Time Series functionality Implementation:
```
import pandas as pd
```

Introduction of matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.
Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.
For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users



Implementation:
Import matplotlib.pyplot as plt

Introduction of scikit-learn

Scikit-learn (formerly scikits. learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from INRIA took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012. As of 2018, scikit-learn is under active development. Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR.

Advantages of using Scikit-Learn:

Scikit-learn provides a clean and consistent interface to tons of different models. It provides you with many options for each model, but also chooses sensible defaults. Its documentation is exceptional, and it helps you to understand the models as well as how to use them properly. It is also actively being developer.

Project overview

project01

November 16, 2024

1 HEART DISEASE PREDICTION MODEL

[1]:
```
import numpy as np import
pandas as pd # %
matplotlib inline import
matplotlib.pyplot as plt #
import seaborn as sns #
model implimplement

from sklearn.linear_model import
LogisticRegression from sklearn.neighbors
import KNeighborsClassifier from
sklearn.ensemble import RandomForestClassifier
#model evaluation

from sklearn.model_selection import
train_test_split,cross_val_score from
sklearn.model_selection import
RandomizedSearchCV,GridSearchCV from sklearn.metrics import
confusion_matrix,classification_report from sklearn.metrics
import precision_score,recall_score,f1_score from
sklearn.metrics import RocCurveDisplay
```

1.1 Load data

[2]:
```
df=pd.read_csv('heart_data_01.csv')
df

df.shape
```

[2]:

	Disease	Age	Sex	cp	fbs	pain	trestbps	chol	restecg	ecg_01
0	-1	63	1	1	0	0	145	233	1	0
1	1	67	1	0	0	0	160	286	0	0
2	1	67	1	0	0	0	120	229	0	0
3	-1	37	1	0	0	1	130	250	0	0
4	-1	41	0	0	1	0	130	204	0	0
..	…	…	…	..	…	…	…	…	…	…
294	1	57	0	0	0	0	140	241	0	0
295	1	45	1	1	0	0	110	264	0	0

```
296       1   68    1   0    0     0       144 193        1       0
297        1 57     1   0    0     0       130 131        0       0
298        1 57     0   0    1     0       130 236        0       0
     target thalach angina oldpeak slope exang ca defect thal
0         1  150   0     2.3   0     1 0   0     1
1         1  108   1     1.5   0     0     3     0     0
2         1  129   1     2.6   0     0     2     1     0
3         0  187   0     3.5   0     1     0     0     0
4         1  172   0     1.4   1     0     0     0     0
..      …      …     …      …    …     … ..      …   …
294       0  123   1     0.2   0     0     0     1     0
295       0  132   0     1.2   0     0     0     1     0
296       0  141   0     3.4   0     0     2     1     0
297       0  115   1     1.2   0     0     1     1     0 298 1     174   0
             0.0   0     0     1     0     0

[299 rows x 19 columns]
```

[4]:
```
df.tail()
```

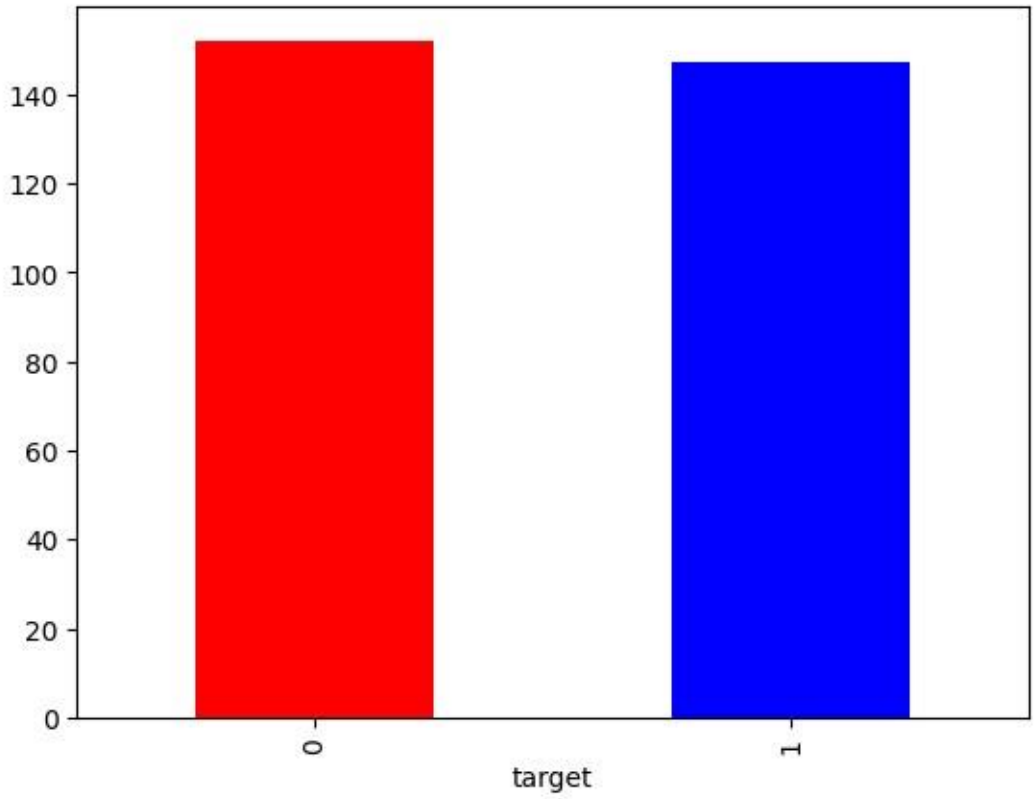
[4]:
```
      Disease Age Sex cp fbs pain trestbps chol restecg ecg_01 \
294       1   57    0   0    0     0       140 241        0       0
295        1 45     1   1    0     0       110 264        0       0
296        1 68     1   0    0     0       144 193        1       0
297        1 57     1   0    0     0       130 131        0       0
298        1 57     0   0    1     0       130 236        0       0
     target thalach angina oldpeak slope exang ca defect thal
294       0  123   1     0.2   0     0     0     1     0
295       0  132   0     1.2   0     0     0     1     0
296       0  141   0     3.4   0     0     2     1     0
297       0  115   1     1.2   0     0     1     1     0
298       1  174   0     0.0   0     0     1     0     0
```

[5]:
```
df["target"].value_counts()
```

[5]:
```
target
0    152
1    147
Name: count, dtype: int64
```

[6]:
```
df["target"].value_counts().plot(kind="bar",color=['red','blue'])
```

[6]:
```
<Axes: xlabel='target'>
```



[7]:
```
df.info()
```

```
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to
298 Data columns (total 19
columns):
#   Column   Non-Null Count Dtype
--- ------   -------------- -----
0   Disease  299 non-null int64
1   Age      299 non-null int64
2   Sex      299 non-null int64
3   cp       299 non-null int64
4   fbs      299 non-null int64
5   pain     299 non-null int64
6   trestbps 299 non-null int64
7   chol     299 non-null int64
8   restecg  299 non-null int64
9   ecg_01   299 non-null int64
10 target    299 non-null int64
11 thalach   299 non-null int64
```

```
 12 angina    299 non-null int64
 13 oldpeak   299 non-null float64
 14 slope     299 non-null int64
 15 exang     299 non-null int64
 16 ca        299 non-null int64
 17 defect    299 non-null int64
 18 thal      299 non-null int64
dtypes: float64(1), int64(18)
memory usage: 44.5 KB
```

[8]:
```
df.isnull().sum()
```

[8]:
```
Disease   0
Age       0
Sex       0
cp        0
fbs       0
pain      0
trestbps  0
chol      0
restecg   0
ecg_01    0
target    0
thalach   0
angina    0
oldpeak   0
slope     0
exang     0
ca        0
defect    0
thal      0
dtype: int64
```

[9]:
```
df.describe()
```

[9]:
```
Disease Age Sex cp fbs pain \ count 299.000000 299.000000 299.000000
299.000000 299.000000 299.000000
mean   -0.076923  54.528428   0.675585   0.076923   0.163880   0.280936
std     0.998709   9.020950   0.468941   0.266916   0.370787   0.450210
min    -1.000000  29.000000   0.000000   0.000000   0.000000   0.000000
25%    -1.000000  48.000000   0.000000   0.000000   0.000000   0.000000
50%    -1.000000  56.000000   1.000000   0.000000   0.000000   0.000000
75%     1.000000  61.000000   1.000000   0.000000   0.000000   1.000000
max     1.000000  77.000000   1.000000   1.000000   1.000000   1.000000

                                                                thalach
        trestbps      chol    restecg     ecg_01     target    \
count   299.000000   299.000000   299.000000   299.000000   299.000000
299.000000 mean  131.668896  247.100334  0.147157  0.013378  0.491639
```

```
149.505017  std  17.705668  51.914779  0.354856  0.115079  0.500768
22.954927  min  94.000000  126.000000  0.000000  0.000000  0.000000
71.000000
25%   120.000000 211.000000  0.000000   0.000000   0.000000 133.000000
50%   130.000000 242.000000  0.000000   0.000000   0.000000 153.000000
75%   140.000000 275.500000  0.000000   0.000000   1.000000 165.500000
max   200.000000 564.000000  1.000000   1.000000   1.000000 202.000000

         angina    oldpeak      slope      exang         ca     defect \
count 299.000000 299.000000 299.000000 299.000000 299.000000
299.000000
mean    0.327759   1.051839   0.468227   0.070234   0.672241   0.384615
std     0.470183   1.163809   0.499826   0.255970   0.937438   0.487320
min     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
25%     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
50%     0.000000   0.800000   0.000000   0.000000   0.000000   0.000000
75%     1.000000   1.600000   1.000000   0.000000   1.000000   1.000000
max     1.000000   6.200000   1.000000   1.000000   3.000000   1.000000

           thal
count 299.000000
mean  0.060201
std   0.238257
min   0.000000
25%   0.000000
50%   0.000000
75%   0.000000
max   1.000000
```

[10]:
```
df.Sex.value_counts()
```

[10]:
```
Sex
1   202
0    97
Name: count, dtype: int64
```

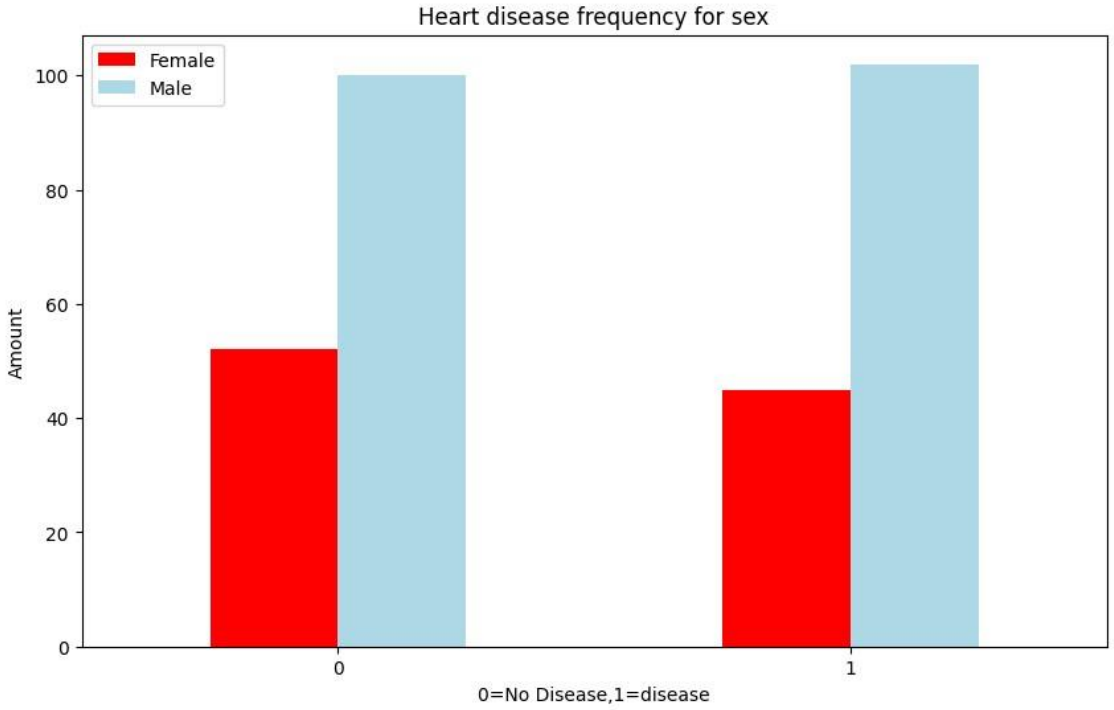
[11]:
```
pd.crosstab(df.target,df.Sex)
```

[11]:
```
Sex    0    1
target
0      52 100
1      45 102
```

[12]:
```
pd.crosstab(df.target,df.Sex).
 ↪plot(kind='bar',figsize=(10,6),color=["red","ligh
tblue"]) plt.title("Heart disease frequency for
sex") plt.xlabel("0=No Disease,1=disease")
```

```
    plt.ylabel("Amount") plt.legend(["Female","Male"]);
    plt.xticks(rotation=0)
```

[12]: (array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])



[13]:
```
df["thalach"].value_counts()
```

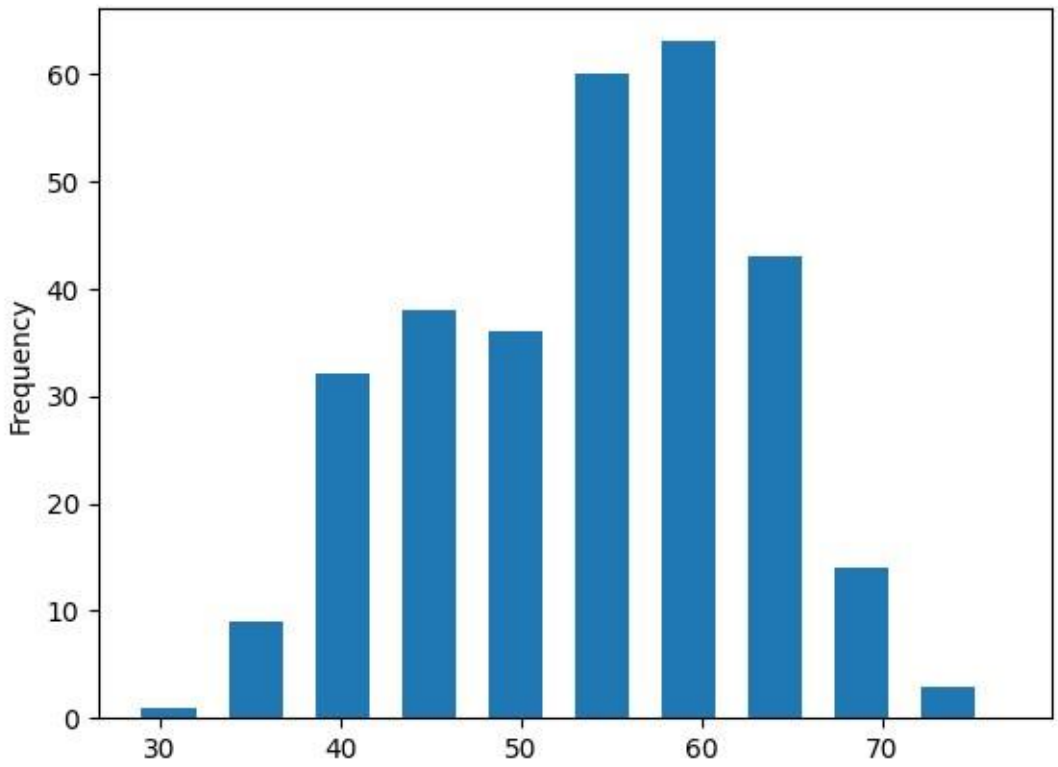
[13]:
```
thalach
162    11
160     9
163     9
152     8
150     7
       ..
177     1
127     1
97      1
190     1
90      1
Name: count, Length: 91, dtype: int64
```

1.2 Age vs Max Heart rate

[14]:
```
plt.figure(figsize=(10,6))
plt.scatter(df.Age[df.target==1],
            df.thalach[df.target==1],
            color='blue')
```

```
#negative result
plt.scatter(df.Age[df.target==0],
            df.thalach[df.target==0],
            color='lightblue')
plt.title("Heart Disease in function of Age and Max Heart Rate ")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease","No disease"]);
```

[15]:
```
# check the distributon of age
df.Age.plot.hist(width=3);
```

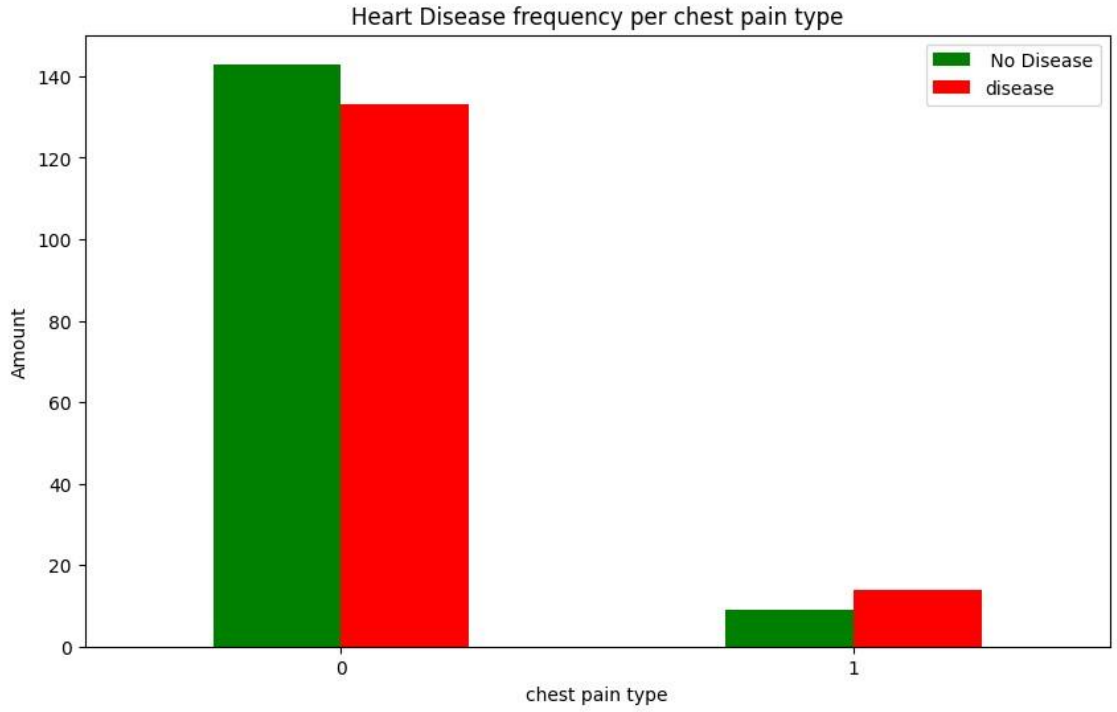


[16]:
```
# compair chest pain vs target
pd.crosstab(df.cp,df.target)
```

[16]:
```
target     0
1 cp
0       143 133
1         9   14
```

[17]:
```
pd.crosstab(df.cp,df.target).plot(kind='bar',
                                  figsize=(10,6),
                                  color=["green","red"])
plt.title("Heart Disease frequency per chest pain type ")
plt.xlabel("chest pain type")
plt.ylabel("Amount")
plt.legend([" No Disease","disease"]);
plt.xticks(rotation=0)
```

[17]:
```
(array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])
```



[18]:
```
df.head()
```

[18]:
```
     Disease Age Sex cp fbs pain trestbps chol restecg ecg_01 target \
0        -1  63    1     1     0     0     145   233   1     0     1
1        1   67    1     0     0     0     160   286   0     0     1
2        1   67    1     0     0     0     120   229   0     0     1
3        -1  37    1     0     0     1     130   250   0     0     0
4        -1  41    0     0     1     0     130   204   0     0     1

   thalach angina oldpeak slope exang ca defect thal
0      150   0      2.3   0      1 0   0      1
1      108   1      1.5   0      0 3   0      0 2   129   1     2.6
              0      0 2   1      0
3      187   0      3.5   0      1 0   0      0
4      172   0      1.4   1      0 0   0      0
```

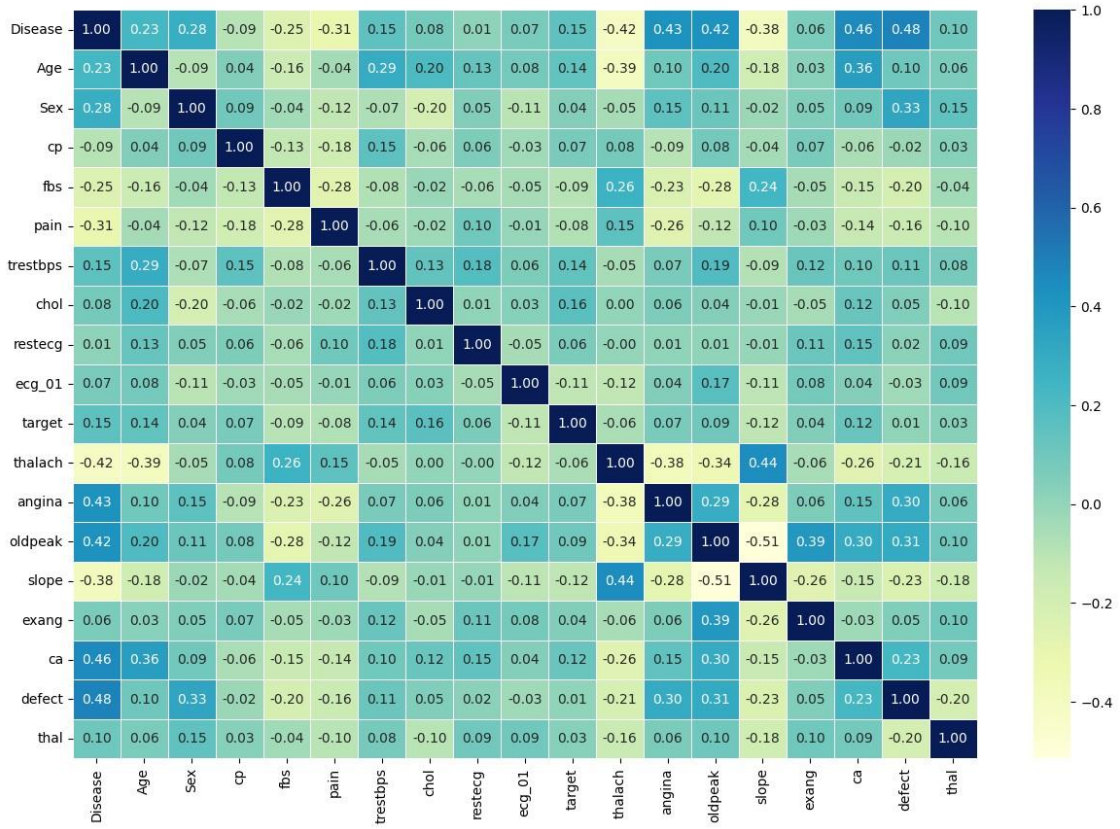
[19]:
```
# correlation matrics
df.corr()
```

[19]:
```
          Disease      Age       Sex       cp       fbs      pain \
Disease    1.000000 0.225775 0.283300 -0.091024 -0.246763 -
0.310013 Age      0.225775 1.000000 -0.091813 0.042989 -
0.162418 -0.043286 Sex      0.283300 -0.091813 1.000000
0.092803 -0.040600 -0.123170 cp  -0.091024 0.042989 0.092803
```

```
1.000000 -0.127802 -0.180439 fbs -0.246763 -0.162418 -
0.040600 -0.127802 1.000000 -0.276725 pain -0.310013 -
0.043286 -0.123170 -0.180439 -0.276725 1.000000 trestbps
0.152840 0.290696 -0.065521 0.150260 -0.080136 -0.055227
chol 0.078722 0.203377 -0.195907 -0.055531 -0.015501 -
0.024900 restecg 0.013111 0.128676 0.045862 0.057231 -
0.056382 0.097436 ecg_01 0.067379 0.083677 -0.105856 -
0.033615 -0.051552 -0.008015 target 0.149680 0.139149
0.038425 0.067592 -0.091995 -0.078853 thalach -0.415031 -
0.392342 -0.052064 0.081268 0.256764 0.150852
angina   0.425476 0.095108 0.149038 -0.094614 -0.232138 -
         0.262072
oldpeak  0.424672 0.197376 0.110237 0.084343 -0.281040 -
         0.121395
slope   -0.384730 -0.183068 -0.022648 -0.044501 0.236417
        0.099450
exang    0.060585 0.026018 0.050676 0.068006 -0.050966 -
         0.026198
ca       0.460442 0.362605 0.093185 -0.059834 -0.153886 -
         0.138891
defect   0.481585 0.104754 0.327572 -0.021830 -0.201429 -
         0.157658
thal     0.104142 0.060092 0.145351 0.032472 -0.036080 -
         0.095631

        trestbps     chol restecg   ecg_01   target  thalach \
Disease  0.152840 0.078722 0.013111 0.067379 0.149680 -
         0.415031
Age      0.290696 0.203377 0.128676 0.083677 0.139149 -
         0.392342
Sex     -0.065521 -0.195907 0.045862 -0.105856 0.038425 -
        0.052064
cp       0.150260 -0.055531 0.057231 -0.033615 0.067592
         0.081268
fbs     -0.080136 -0.015501 -0.056382 -0.051552 -0.091995
        0.256764
pain    -0.055227 -0.024900 0.097436 -0.008015 -0.078853
        0.150852
trestbps 1.000000 0.132284 0.177623 0.058177 0.141046 -
0.048053 chol    0.132284 1.000000 0.006664 0.032914 0.160090
0.002179 restecg 0.177623 0.006664 1.000000 -0.048370 0.063599
-0.003387 ecg_01 0.058177 0.032914 -0.048370 1.000000 -
0.114513 -0.120705 target  0.141046 0.160090 0.063599 -
0.114513 1.000000 -0.063417 thalach -0.048053 0.002179 -
0.003387 -0.120705 -0.063417 1.000000
angina   0.065885 0.056388 0.011637 0.042728 0.068687 -0.376359
oldpeak  0.191615 0.040431 0.009093 0.167688 0.090282 -0.341262
```

```
slope   -0.087458 -0.014490 -0.011390 -0.109266 -0.118375 0.442894
exang    0.121396 -0.050027 0.107496 0.081915 0.043866 -0.055172
ca       0.098773 0.119000 0.145478 0.040781 0.122813 -0.264246
defect   0.110482 0.050995 0.020898 -0.032220 0.006347 -0.209410
thal     0.075538 -0.098157 0.093319 0.092917 0.032359 -0.158969

          angina  oldpeak    slope    exang       ca   defect    thal
Disease  0.425476 0.424672 -0.384730 0.060585 0.460442 0.481585
         0.104142
Age      0.095108 0.197376 -0.183068 0.026018 0.362605 0.104754
         0.060092
Sex      0.149038 0.110237 -0.022648 0.050676 0.093185 0.327572
         0.145351
cp      -0.094614  0.084343  -0.044501  0.068006  -0.059834  -0.021830
        0.032472
fbs     -0.232138 -0.281040 0.236417 -0.050966 -0.153886 -0.201429 -
        0.036080
pain    -0.262072 -0.121395 0.099450 -0.026198 -0.138891 -0.157658 -
        0.095631
trestbps  0.065885  0.191615  -0.087458  0.121396  0.098773  0.110482
0.075538 chol 0.056388 0.040431 -0.014490 -0.050027 0.119000 0.050995
-0.098157  restecg  0.011637  0.009093  -0.011390  0.107496  0.145478
0.020898  0.093319  ecg_01  0.042728  0.167688  -0.109266  0.081915
0.040781  -0.032220  0.092917  target  0.068687  0.090282  -0.118375
0.043866  0.122813  0.006347  0.032359  thalach  -0.376359  -0.341262
0.442894 -0.055172 -0.264246 -0.209410 -0.158969
angina 1.000000 0.289573 -0.283956 0.059028 0.145570 0.297415 0.062916

oldpeak         0.289573 1.000000 -0.514906 0.393261 0.295832 0.306719
0.102466

slope     -0.283956 -0.514906 1.000000 -0.257901 -0.151212 -0.232087 -
0.181135

exang 0.059028 0.393261 -0.257901 1.000000 -0.029606 0.051734 0.095509

ca    0.145570 0.295832 -0.151212 -0.029606 1.000000 0.225453 0.088639

defect        0.297415 0.306719 -0.232087 0.051734 0.225453 1.000000 -
0.200089

thal  0.062916 0.102466 -0.181135 0.095509 0.088639 -0.200089 1.000000
```

[20]:
```
import seaborn as sns
corr_mat=df.corr()
fig,ax=plt.subplots(figsize=(15,10))
ax = sns.heatmap(corr_mat,
                 annot=True,

                 linewidths=0.5,
                 fmt="0.2f",
                 cmap="YlGnBu")
```



1.3 Modeling

[21]:
```
df.head()
```

[21]:

	Disease	Age	Sex	cp	fbs	pain	trestbps	chol	restecg	ecg_01	target	thalach	angina	oldpeak	slope	exang	ca	defect	thal
0	-1	63	1	1	0	0	145	233	1	0	1	150	0	2.3	0	1	0	0	1
1	1	67	1	0	0	0	160	286	0	0	1								
2	1	67	1	0	0	0	120	229	0	0	1	3	-1						
		37	1	0	0	1	130	250	0	0	0								
4	-1	41	0	0	1	0	130	204	0	0	1								

```
1      108  1     1.5   0     0 3   0     0 2   129   1     2.6
             0     0 2   1     0
3      187       0      3.5      0       1 0         0     0
4      172       0      1.4      1      0            0     0
```

[22]:
```
x=df.drop("target",axis=1)

y=df["target"]
```

[23]:
```
y
```

[23]:
```
0      1
1      1
2      1
3      0
4      1
..
294    0
295    0
296    0
297    0
298    1
      Name: target, Length: 299, dtype: int64
```

[24]:
```

x
```

[24]:
```
      Disease Age Sex cp fbs pain trestbps chol restecg ecg_01 \
0         -1 63  1     1     0     0     145 233    1     0
1         1 67   1     0     0     0     160 286    0     0
2         1 67   1     0     0     0     120 229    0     0
3         -1 37  1     0     0     1     130 250    0     0
4         -1 41  0     0     1     0     130 204    0     0
      ..        … … … .. …    …       …   …      …     …
      294       1 57   0     0     0     0     140 241    0     0
      295       1 45   1     1     0     0     110 264    0     0
      296       1 68   1     0     0     0     144 193    1     0
```

```
297       1 57   1     0     0     0     130 131    0     0
298       1 57   0     0     1     0     130 236    0     0

     thalach angina oldpeak slope exang ca defect thal
0        150 0     2.3   0     1     0     0     1
1        108 1     1.5   0     0     3     0     0
2        129 1     2.6   0     0     2     1     0 3   187   0     3.5   0
             1     0     0     0
4        172       0     1.4     1     0   0       0     0
..       …     …       …    …    … ..     …   …
294      123 1     0.2   0     0     0     1     0
295      132 0     1.2   0     0     0     1     0
296      141 0     3.4   0     0     2     1     0
297      115 1     1.2   0     0     1     1     0
298      174 0     0.0   0     0     1     0     0
[299 rows x 18 columns]
```

[25]:
```
#split data
np.random.seed(42)
x_train,x_test,y_train,y_test=train_test_split(x,
                                               y,test_size=0.2)
```

[26]:
```
x_train
```

[26]:
```
     Disease Age Sex cp fbs pain trestbps chol restecg ecg_01 \
6          1   62    0   0    0     0       140 268        0       0
183        1  60    0   0    0     0       158 305        0       0
185       -1  42    1   0    0     1       120 240        1       0
146        1  57    1   0    0     0       165 289        1       0
30        -1  69    0   1    0     0       140 239        0       0
..        … … … .. …    …       …   …       …     …
188        1  69    1   0    0     1       140 254        0       0
71         1  67    1   0    0     0       125 254        1       0
106        1  59    1   0    0     0       140 177        0       0
270        1  46    1   0    0     0       140 311        0       0
102       -1  57    0   0    0     0       128 303        0       0

     thalach angina oldpeak slope exang ca defect thal
6        160       0     3.6     0     1   2       0     0
183      161       0     0.0     1     0   0       0     0
185   194   0     0.8   0     1     0     1     0 146 124
0     1.0   0     0     3     1     0
30       151       0     1.8     1     0   2       0     0
..       …     …       …    …    … ..     …   …
188      146       0     2.0     0     0   3       1     0
71       163       0     0.2     0     0   2       1     0
```

```
106     162      1      0.0      1      0   1       1     0
270     120      1      1.8      0      0   2       1     0
102     159      0      0.0      1      0   1       0     0

[239 rows x 18 columns]
```

[27]:
```
y_train
```

[27]:
```
6     1
183   1
185   0
146   1
30    0
..
188   1
71    0
106   0
270   0
102   1
Name: target, Length: 239, dtype: int64
```

2 machie learning model

[28]:
```
# check  model
models= { "Logistic Regression":LogisticRegression(),
          "KNN": KNeighborsClassifier(),
          "Randon Forest": RandomForestClassifier()}
def fit_and_score(models,x_train,x_test,y_train,y_test):
    np.random.seed(42)
    model_score={}
    for name,model in models.items():
        model.fit(x_train,y_train)
        model_score[name]=model.score(x_test,y_test)
    return model_score
```

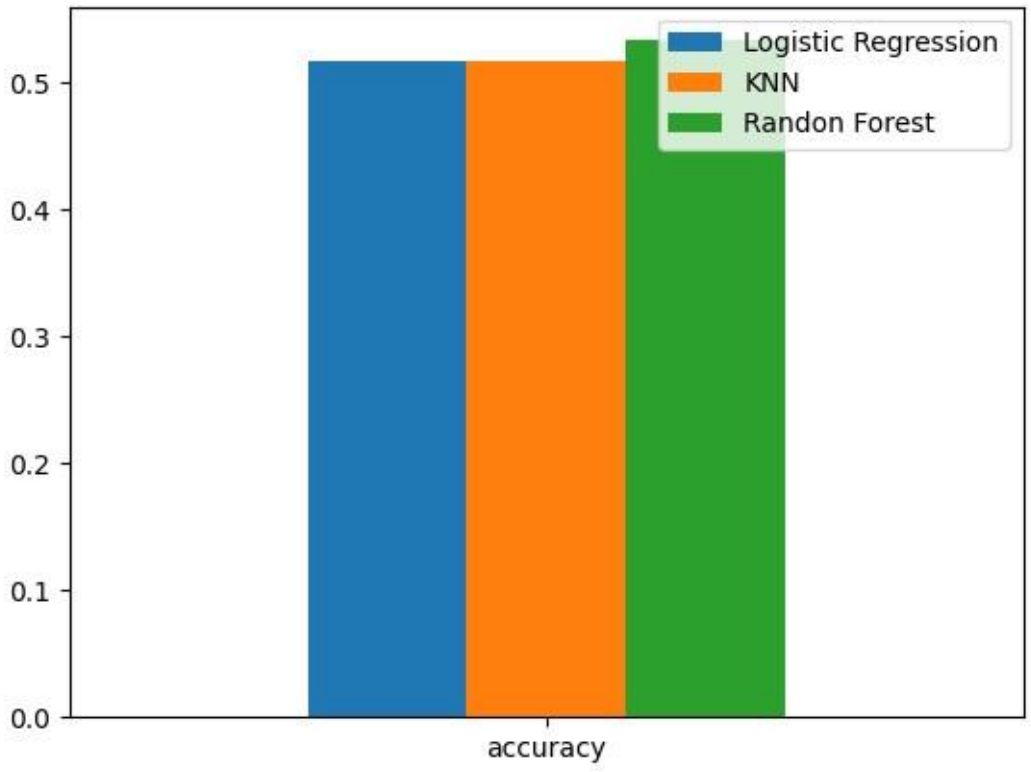
[29]:
```
model_score = fit_and_score(models,
                            x_train=x_train,
                            x_test=x_test,
                            y_train=y_train,
                            y_test=y_test)
model_score
```

```
    C:\Users\dell\miniconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs
                                                                    failed
    to converge (status=1):
    STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

    Increase the number of iterations (max_iter) or scale the data as
        shown in: https://scikit-
        learn.org/stable/modules/preprocessing.html
    Please also refer to the documentation for alternative solver
    options:
        https://scikit-
        learn.org/stable/modules/linear_model.html#logistic-
    regression
      n_iter_i = _check_optimize_result(
```

[29]:
```
{'Logistic Regression': 0.5166666666666667,
 'KNN': 0.5166666666666667,
 'Randon Forest': 0.5333333333333333}
```

[30]:
```
model_compare=pd.DataFrame(model_score,index=["accuracy"])
model_compare.plot.bar()
plt.xticks(rotation=0)
```

[30]:
```
(array([0]), [Text(0, 0, 'accuracy')])
```



[31]:
```
# hyperparameter tuning
train_score=[]
test_score=[]
neighbors=range(1,21)
knn=KNeighborsClassifier()
for i in neighbors:
    knn.set_params(n_neighbors=i)
    # fit algrithm
    knn.fit(x_train,y_train)
    train_score.append(knn.score(x_train,y_train))
    test_score.append(knn.score(x_test,y_test))
```

[32]:
```
train_score
```

[32]:
```
[1.0,
 0.7698744769874477,
 0.7280334728033473,
 0.7196652719665272,
 0.7071129707112971,
 0.7112970711297071,
 0.698744769874477,
 0.6861924686192469,
 0.6820083682008368,
 0.6652719665271967,
 0.6652719665271967,
 0.6569037656903766,
 0.6694560669456067,
 0.6401673640167364,
 0.6192468619246861,
 0.6234309623430963,
 0.6359832635983264,
 0.6192468619246861,
 0.6108786610878661,
 0.602510460251046]
```

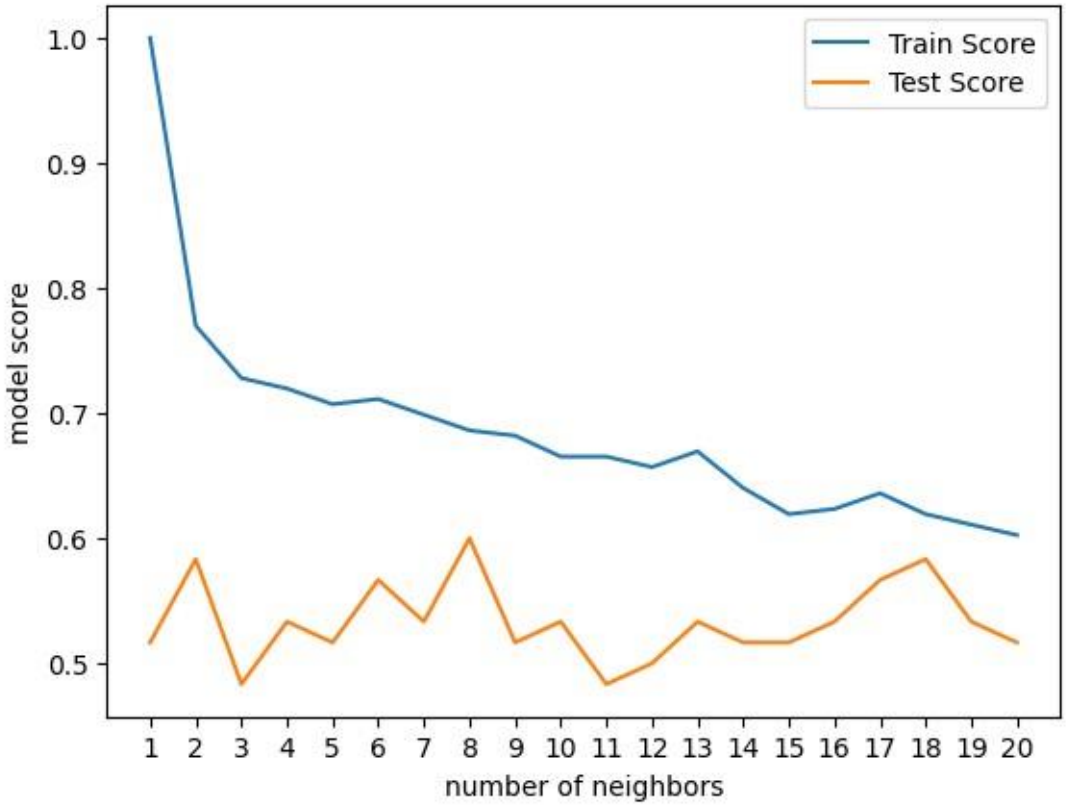
[33]:
```
test_score
```

[33]:
```
[0.5166666666666667,
 0.5833333333333334,
 0.48333333333333334,
 0.5333333333333333,
 0.5166666666666667,
 0.5666666666666667,
 0.5333333333333333,
 0.6,
 0.5166666666666667,
 0.5333333333333333,
```

```
0.48333333333333334,
0.5,
0.5333333333333333,
0.5166666666666667,
0.5166666666666667,
0.5333333333333333,
0.5666666666666667,
0.5833333333333334,
0.5333333333333333,
0.5166666666666667]
```

[34]:
```
    plt.plot(neighbors,train_score,label="Train
Score")
plt.plot(neighbors,test_score,label="Test
Score")   plt.xlabel("number   of   neighbors")
plt.xticks(np.arange(1,21,1)) plt.ylabel("model
score")   plt.legend()   print(f"Maximun   knn
score:{max(test_score)*100:0.2f}")
```

```
Maximun knn score:60.00
```



[35]:
```
log_grid={"C":np.logspace(-4,4,20),
          "solver":["liblinear"]}
```

[36]:
```
np.random.seed(42)
log_reg=RandomizedSearchCV(LogisticRegression(),
                           param_distributions=log_grid,
                           cv=5,
                           n_iter=20,
                           verbose=True)
# fit the model
log_reg.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

[36]:
```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_iter=20,
                   param_distributions={'C': array([1.00000000e-04,
       2.63665090e-04, 6.95192796e-04, 1.83298071e-03,
         4.83293024e-03, 1.27427499e-02, 3.35981829e-02, 8.85866790e-02,
         2.33572147e-01, 6.15848211e-01, 1.62377674e+00, 4.28133240e+00,
      1.12883789e+01, 2.97635144e+01, 7.84759970e+01, 2.06913808e+02,
      5.45559478e+02, 1.43844989e+03, 3.79269019e+03, 1.00000000e+04]),
                                        'solver': ['liblinear']},
                   verbose=True)
```

[37]:
```
log_reg.best_params_
```

[37]:
```
{'solver': 'liblinear', 'C': 0.0001}
```

[38]:
```
log_reg.score(x_test,y_test)
```

[38]: 0.5833333333333334

[39]:
```
# rf_grid={"n_estimators": np.arange(10.1000,50),
#          'bootstrap': [True, False],
#           "max_depth": [None]
#         "min_samples_split": np.arange(2,20,2),
#         "min_samples_leaf":np.arange(1,20,2)}
```

[40]:
```
# np.random.seed(42)
# rsr=RandomizedSearchCV(RandomForestClassifier(),
#                        param_distributions= rf_grid,
#                        cv=5,
#                        n_iter=20,
#                        verbose=True)
# rsr.fit(x_train,y_train)
```

[41]:
```
# hyperparameter tuning with grid search

log_grid={"C":np.logspace(-4,4,20),
         "solver":["liblinear"]}
```

```
log_reg=GridSearchCV(LogisticRegression(),
                           param_grid=log_grid,
                           cv=5,

                           verbose=True)
# fit the model
log_reg.fit(x_train,y_train);
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

[42]:
```
log_reg.best_params_
```

[42]: {'C': 0.0001, 'solver': 'liblinear'}

[43]:
```
log_reg.score(x_test,y_test)
```

[43]: 0.5833333333333334

[44]:
```
# evaluating our model
y_pre=log_reg.predict(x_test)
y_pre
```

[44]:
```
array([0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
                                                       1, 1, 1, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1], dtype=int64)
```

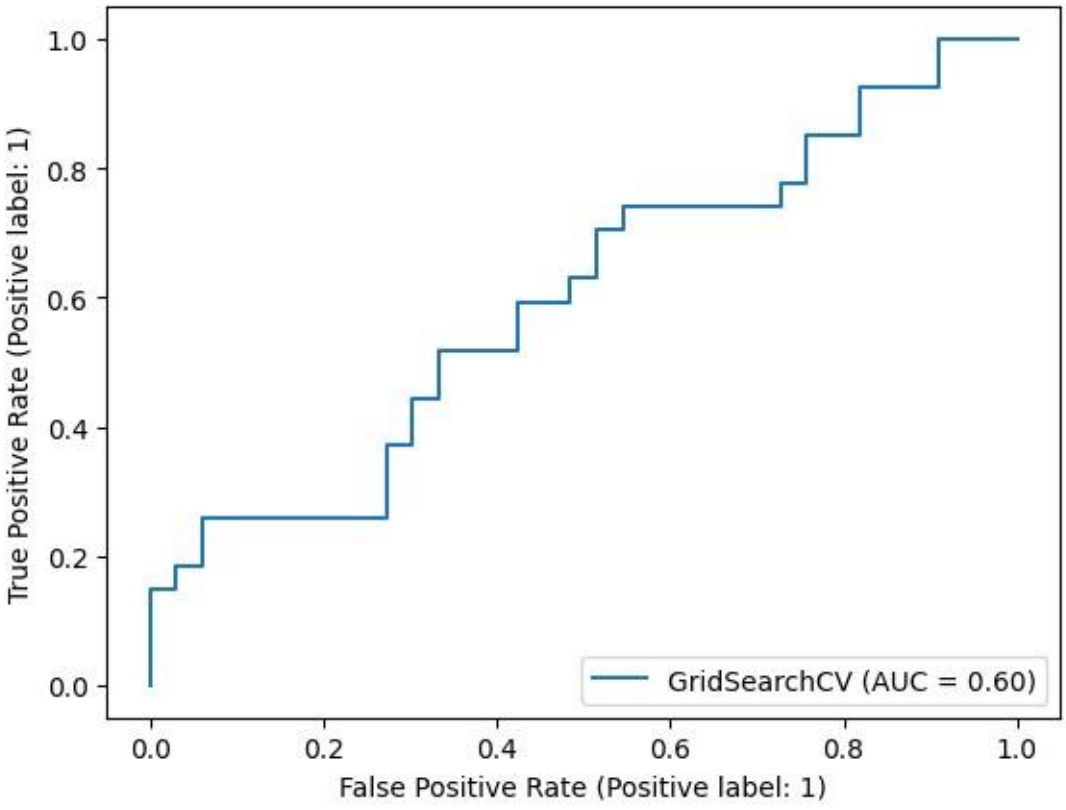
[45]:
```
y_test
```

```
[45]: 281   0
      265   0
      164   0
      9     1
      77    1
      278   0
      93    0
      109   0
      5     0
      173   1
      97    1
      195   1
      184   0
      154   1
      57    1
      60    0
      147   0
      108   0
      63    0
      140   0
      155   1
      104   0
      247   1
      46    0
      42    0
      275   0
      280   0
      116   1
      213   1
      236   0
      17    0
      239   0
      33    0
      24    1
      45    1
      7     0
```

```
113    1
194    1
111    1
92     0
75     1
82     1
118    1
76     1
129    0
197    1
210    1
288    0
219    1
178    1
144    1
186    0
84     0
248    0
277    0
73     1
244    0
25     0
209    0
59     1
Name: target, dtype: int64
```

[46]:
```
RocCurveDisplay.from_estimator(
...    log_reg, x_test, y_test)
```

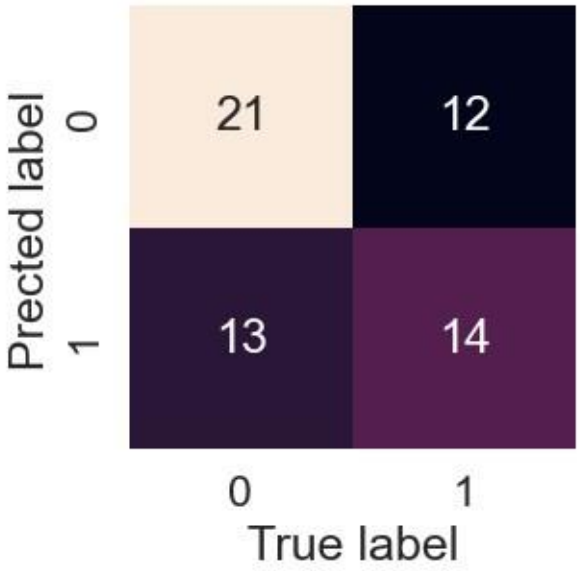
[46]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x220ceeebe50>



[47]:
```
print(confusion_matrix(y_test,y_pre))
```

```
[[21 12]
 [13 14]]
```

[48]:
```
sns.set(font_scale=1.5)
def plots(y_test,y_pre):
    fig,ax=plt.subplots(figsize=(3,3))
    ax=sns.heatmap(confusion_matrix(y_test,y_pre),
                   annot=True,
                   cbar=False)
    plt.xlabel("True label")
    plt.ylabel("Prected label")
plots(y_test,y_pre)
```



3 Classification Report

[49]:
```
print(classification_report(y_test,y_pre))
```

```
              precision    recall  f1-score   support

           0       0.62      0.64      0.63        33
           1       0.54      0.52      0.53        27

    accuracy                           0.58        60
   macro avg       0.58      0.58      0.58        60
weighted avg       0.58      0.58      0.58        60
```

[50]:
```
log_reg.best_params_
```

[50]:
```
{'C': 0.0001, 'solver': 'liblinear'}
```

[52]:
```
clf=LogisticRegression(C=0.0001,
                       solver="liblinear")
```

[53]:
```
cv_acc=cross_val_score(clf,
                       x,
                       y,
                       cv=5,
                       scoring="accuracy")
```

```
cv_acc
```

[53]:
```
array([0.61666667, 0.55 , 0.63333333, 0.7    , 0.57627119])
```

[54]:
```
cv_acc=np.mean(cv_acc)
cv_acc
```

[54]:
```
0.6152542372881357
```

[55]:
```
cv_precision=cross_val_score(clf,
                      x,
                      y,
                      cv=5,
                      scoring="precision")
cv_precision=np.mean(cv_precision)
cv_precision
```

[55]:
```
0.6206652802630942
```

[56]:
```
cv_recall =cross_val_score(clf,
                      x,
                      y,
                      cv=5,
                      scoring="recall")
cv_recall= np.mean(cv_recall)
cv_recall
```

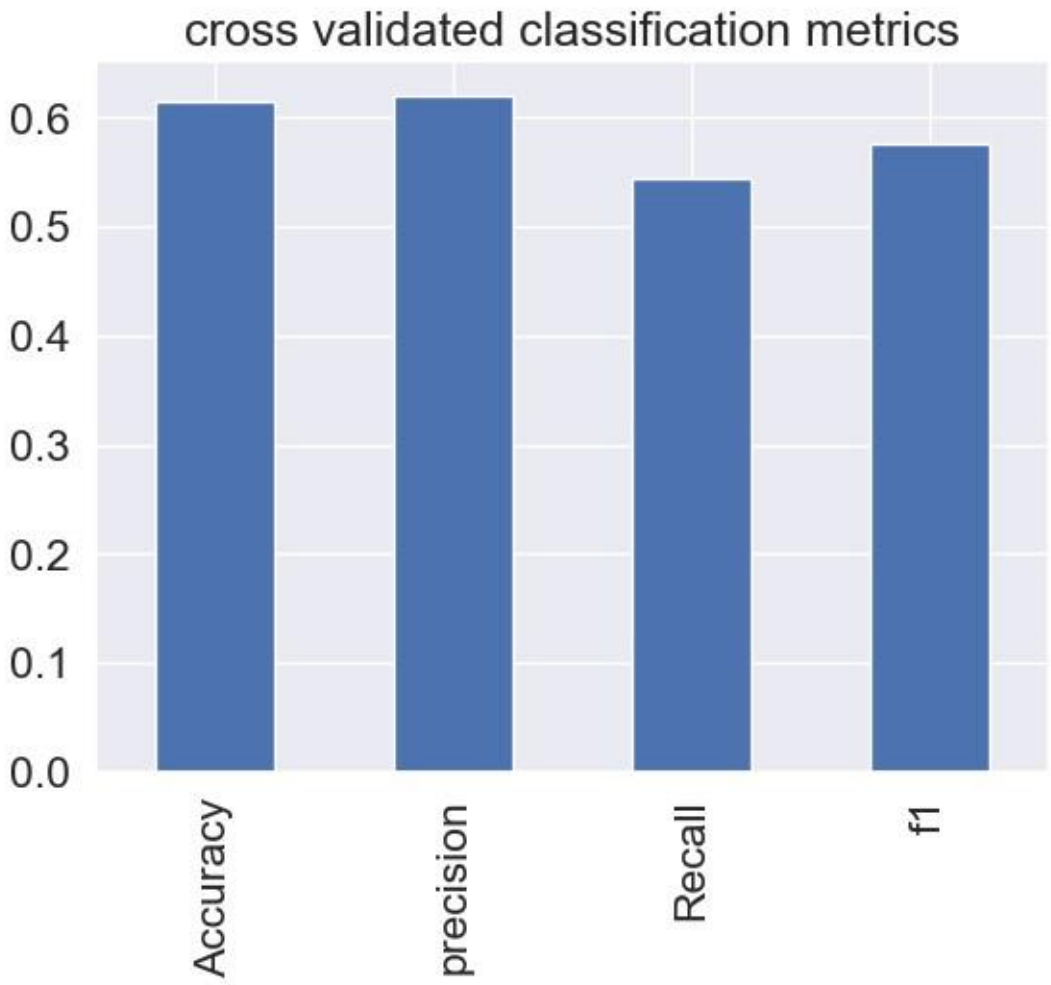
[56]:
```
0.5450574712643678
```

[58]:
```
cv_f1score=cross_val_score(clf,
                      x,
                      y,
                      cv=5,
                      scoring="f1")
cv_f1score=np.mean(cv_f1score)
cv_f1score
```

[58]:
```
0.5765739586187583
```

[59]:
```
cvplot=pd.DataFrame({"Accuracy":cv_acc,
                "precision": cv_precision,
                  "Recall": cv_recall,
                "f1":cv_f1score},index=[0])
             cvplot.T.plot.bar(title="cross validated classification
                                                    metrics",legend=False)
```

[59]:
```
<Axes: title={'center': 'cross validated classification metrics'}>
```


cross validated classification metrics
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Accuracy
precision
Recall
f1

Certificate of Internship

This is to certify that

SHUBHAM MISHRA

did his/her internship with us between

25 JUNE TO 24 JULY OF 2024 (4 WEEKS)

As a part of internship, he/she completed training on

MACHINE LEARNING

followed by submitting a report on

SHUBHAM MISHRA PROJECT_REPORT

Mayur Dev Sewak
Head, Internship & Operations
Eisystems Services

Mallika Srivastava
Head, Training Delivery
Eisystems Technologies

EI Systems
Unconventional Ideas. Systematic Approach.

Robokwik

Date of Issue : 09-Aug-24
Certificate ID : EIS/AICT/24SP-1545
AICTE ORID : CORPORATE65a015a925f071704990121
Internship Name / Phase ID: EISYS-4/8/12/16 2024

Validate at robokwik.com/certificatevalidation

PROJECT : HEART DIEASES PREDICTION

Summer Internship Project report submitted in partial fulfillment of the requirement for the degree of

Computer Science and Engineering Technology

By

(SHUBHAM MISHRA (CSJMA21001390163))

To

(Er.SHAH ALAM)

Assistant Professor

To

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY CSJM UNIVERSITY, KANPUR