Student name: Roja Kamble - 11454258

Assignment: Classification using Python (scikit-learn) on load_digits dataset

Task 1: Logistic Regression (30 pts)

Load the load_digits dataset and save it in a varuable named digits

In [201]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits
```

Out[201]:
```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
```

Check the type of your data

In [202]:
```
digits.data.shape
```

Out[202]:
```
(1797, 64)
```

Print to show there are 1797 images (8 by 8 images for a dimensionality of 64)

In [203]:
```
digits.images.shape
```

Out[203]:
```
(1797, 8, 8)
```

In [204]:
```
print('Image Data Shape:' , digits.data.shape)
```

```
Image Data Shape: (1797, 64)
```

Print to show there are 1797 labels (integers from 0–9)

In [205]:
```
print("Label Data Shape:", digits.target.shape)
```

```
Label Data Shape: (1797,)
```

Display sample data

In [206]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Print keys and their length

In [207]:
```
# data frame columns and length and print them.

print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits (https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits)

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
```

Graduate Studies in Science and Engineering, Bogazici University.
- E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
- Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
Linear dimensionalityreduction using relevance weighted LDA. School of
Electrical and Electronic Engineering Nanyang Technological University.
2005.
- Claudio Gentile. A New Approximate Maximal Margin Classification
Algorithm. NIPS. 2000.

Assiging the X and Y as data and target values (This part is optional, you can keep it as data and target as well)

In [208]:
```
x=np.array(digits.data)
y=np.array(digits.target)
```

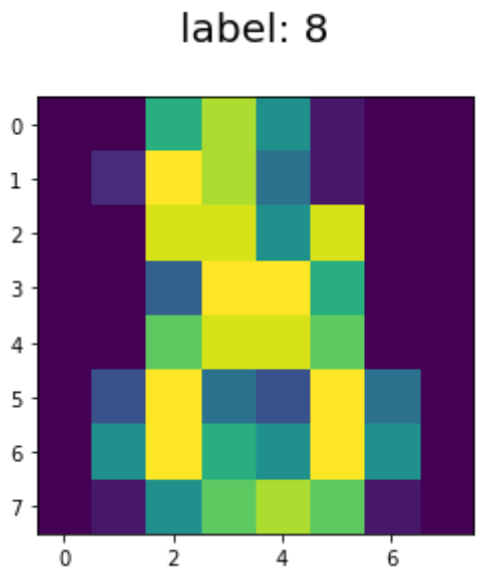
Printing the Shape

In [209]:
```
print(x.shape,y.shape)
```

```
(1797, 64) (1797,)
```

Visualize the Images and Labels in the load_digits Dataset

In [210]:
```
for index, (image, label) in enumerate(zip(digits.data, digits.target)):
    plt.imshow(np.reshape(image, (8,8)))
    plt.title('label: %i\n' % label, fontsize = 20)
```



In [211]:
```
plt.figure(figsize=(20,4))
for i in range(0,10,10):
    for index, (image, label) in enumerate(zip(x[i:i+5], y[i:i+5])):
        plt.subplot(2, 5, index + 1)
        plt.imshow(np.reshape(image,(8,8)), cmap=plt.cm.gray)
        plt.title('Training: %i\n' % label, fontsize = 20)
```

Split Data into Training (75%) and Test Sets (25%) (Digits Dataset) using train_test split

In [212]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state =0)
```

Import the Logistic regression model

In [213]:
```
from sklearn.linear_model import LogisticRegression
```

Make an instance of the Model (No tuning)

In [214]:
```
reg = LogisticRegression()
```

Train the model on the data and store the information learned from the data (hint: use fit function)

Train the model on the data and store the information learned from the data (Hint: use fit function)

Model will learn the relationship between digits (x_train) and labels (y_train)

In [215]:
```
reg.fit(x_train,y_train)
```

```
/Users/roja/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:763: Convergen
ceWarning: 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 (https://scikit-learn.org/stable/module
s/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-lear
n.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[215]:
```
LogisticRegression()
```

Print the number of classes

In [216]:
```
#output classes in the model

print(reg)

print(y_train)
```

```
LogisticRegression()
[2 8 9 ... 7 7 8]
```

Predict labels for new data (new images)

Predict for One Observation (image)

In [217]:
```
one_test = x_test[0]
one_test = one_test.reshape(1,-1)
reg.predict(one_test)
```

Out[217]:
```
array([2])
```

Predict for Multiple Observations (images) at Once

In [218]:
```
mul_test = x_test[:100]
reg.predict(mul_test)
```

Out[218]:
```
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5, 2, 8, 6, 6, 6, 6, 1, 0, 5, 8, 8, 7,
       8, 4, 7, 5, 4, 9, 2, 9, 4, 7, 6, 8, 9, 4, 3, 1, 0, 1, 8, 6, 7, 7,
       1, 0, 7, 6, 2, 1, 9, 6, 7, 9, 0, 0, 9, 1, 6, 3, 0, 2, 3, 4, 1, 9,
       2, 6, 9, 1, 8, 3, 5, 1, 2, 8, 2, 2, 9, 7, 2, 3, 6, 0, 5, 3, 7, 5,
       1, 2, 9, 9, 3, 1, 4, 7, 4, 8, 5, 8])
```

Make predictions on entire test data

In [219]:
```
y_pred = reg.predict(x_test)
print(y_pred)
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 9 1 6 3 0 2 3 4 1 9 2 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 4 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 1 5 2 5 8 4 1 7 0 6 1 5 5 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 8 7 6 7 6 5 6 0 8 8 9 9 6 1 0 4 1 6 3 8 6 7 4 9 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 1 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 8 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 3 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 3 3 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 9 8 5 7 8 2 8 4 3 7 2 6 9 9 5 1 0 8 2 8 9
 5 6 2 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 8 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 1 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 1 1 5 5 9 1 3 7 0 0 3 0 4 5 8 9 3 4 3 1 8 9 8 3 6 3 1 6 2
 1 7 5 5 1 9]
```

In [220]:
```
# Importing Libraries

from sklearn.metrics import precision_recall_fscore_support
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
```

Measure Model Performance (Digits Dataset) before tuning

Use score method to get accuracy of model before tuning

In [221]:
```
reg.score(x_test,y_test)
```

Out[221]:
```
0.9511111111111111
```

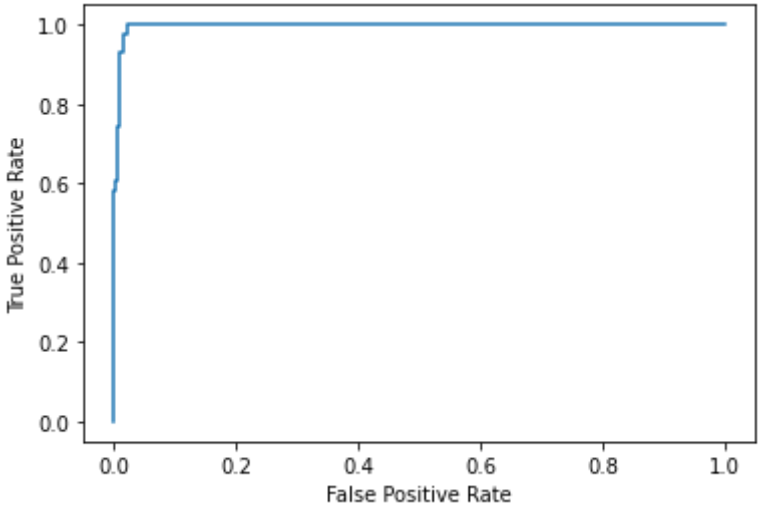
Show precision, recall and F1 Score

In [222]:
```
from sklearn.metrics import precision_recall_fscore_support

precision_recall_fscore_support(y_test, y_pred)
```

Out[222]:
```
(array([1.        , 0.88888889, 0.97560976, 0.91489362, 0.97368421,
        0.9787234 , 1.        , 0.9787234 , 0.91489362, 0.9       ]),
 array([1.        , 0.93023256, 0.90909091, 0.95555556, 0.97368421,
        0.95833333, 0.98076923, 0.95833333, 0.89583333, 0.95744681]),
 array([1.        , 0.90909091, 0.94117647, 0.93478261, 0.97368421,
        0.96842105, 0.99029126, 0.96842105, 0.90526316, 0.92783505]),
 array([37, 43, 44, 45, 38, 48, 52, 48, 48, 47]))
```

Print ROC Curve

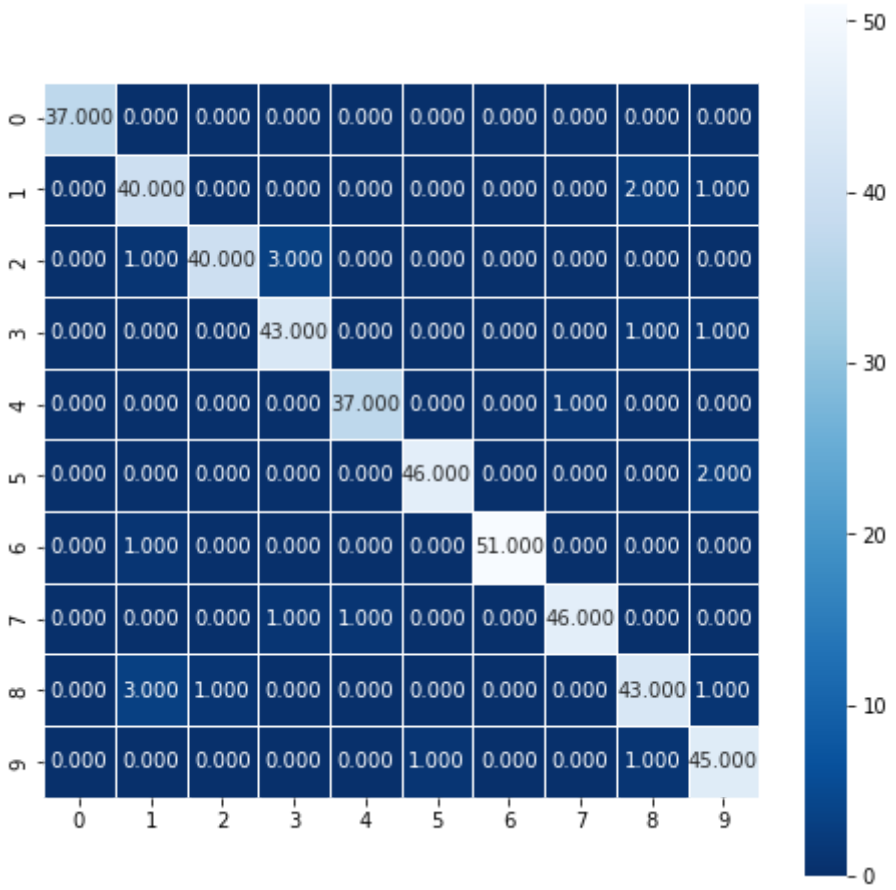
In [223]:
```
y_proba = reg.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,y_proba,pos_label=1)

plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Show the Confusion Matrix (Digits Dataset) to describe the performance of the model (Hint: y_test, predictions) using either seaborn or matplotlib

In [224]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
```

In [225]:
```
cm = metrics.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,8))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
```



Tune the hyperparameters of the logistic regression model using GridSearchCV and save it in a variable called clf

In [226]:
```
from sklearn.model_selection import GridSearchCV

param={'penalty': ['l1','l2'], 'C':[0.01, 0.1, 1, 10, 100]}

clf = GridSearchCV(reg, param_grid = param, scoring='accuracy', cv=10)
```

Train the model

In [227]:
```
clf.fit(x_train,y_train)
```

```
/Users/roja/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:763: Convergen
ceWarning: 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 (https://scikit-learn.org/stable/module
s/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-lear
n.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Measure Model Performance (Digits Dataset) after tuning

Print out the tuned-hyperparameters

In [228]:
```
print(clf.best_params_)
```

```
{'C': 0.01, 'penalty': 'l2'}
```

Print out the tuned the training accuracy

In [229]:
```
clf.score(x_train,y_train)
```

Out[229]:
```
0.9925760950259837
```

Use the values of the hyperparameters returned by the GridSearchCV() function, to build your model using the training dataset:

In [230]:
```
y_predclf = clf.predict(x_test)
```

In [231]:
```
print(y_predclf)
print(clf.score(x_test,y_test))
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 9 1 6 3 0 2 3 4 1 9 2 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 4 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 1 5 2 5 8 4 1 7 0 6 1 5 5 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 8 7 6 7 6 5 6 0 8 8 9 8 6 1 0 4 1 6 3 8 6 7 4 9 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 1 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 3 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 6 7 2 0 9 6 0 4 2 0 7 9 8 5 7 8 2 8 4 3 7 2 6 9 9 5 1 0 8 2 8 9
 5 6 2 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 8 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 1 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 2 1 5 5 9 1 3 7 0 0 7 0 4 5 9 9 3 4 3 1 8 9 8 3 6 2 1 6 2
 1 7 5 5 1 9]
0.9666666666666667
```

Task 2: Logistic Regression on MNIST (30 pts)

Now import the MNIST dataset and perform the whole process again

In [232]:
```
from sklearn.datasets import load_digits
mnist = load_digits()
```

In [233]:
```
from sklearn.model_selection import train_test_split

train_img, test_img, train_lbl, test_lbl = train_test_split(mnist.data, mnist.target, test_size=1/7.0, r
```

In [234]:
```
logisticRegr = LogisticRegression(solver = 'lbfgs')
logisticRegr.fit(train_img, train_lbl)
```

```
/Users/roja/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:763: Convergen
ceWarning: 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 (https://scikit-learn.org/stable/module
s/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-lear
n.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[234]:
```
LogisticRegression()
```

In [235]:
```
predict = logisticRegr.predict(test_img)
predict
```

Out[235]:
```
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5, 2, 8, 6, 6, 6, 6, 1, 0, 5, 8, 8, 7,
       8, 4, 7, 5, 4, 9, 2, 9, 4, 7, 6, 8, 9, 4, 3, 1, 0, 1, 8, 6, 7, 7,
       1, 0, 7, 6, 2, 1, 9, 6, 7, 9, 0, 0, 9, 1, 6, 3, 0, 2, 3, 4, 1, 9,
       2, 6, 9, 1, 8, 3, 5, 1, 2, 8, 2, 2, 9, 7, 2, 3, 6, 0, 5, 3, 7, 5,
       1, 2, 8, 9, 3, 1, 4, 7, 4, 8, 5, 8, 5, 5, 2, 5, 9, 0, 7, 1, 4, 7,
       3, 4, 8, 9, 7, 9, 8, 2, 1, 5, 2, 5, 8, 4, 1, 7, 0, 6, 1, 5, 5, 9,
       9, 5, 9, 9, 5, 7, 5, 6, 2, 8, 6, 9, 6, 1, 5, 1, 5, 9, 9, 1, 5, 3,
       6, 1, 8, 9, 8, 7, 6, 7, 6, 5, 6, 0, 8, 8, 9, 8, 6, 1, 0, 4, 1, 6,
       3, 8, 6, 7, 4, 9, 6, 3, 0, 3, 3, 3, 0, 7, 7, 5, 7, 8, 0, 7, 1, 9,
       6, 4, 5, 0, 1, 4, 6, 4, 3, 3, 0, 9, 5, 9, 2, 1, 4, 2, 1, 6, 8, 9,
       2, 4, 9, 3, 7, 6, 2, 3, 3, 1, 6, 9, 3, 6, 3, 2, 2, 0, 7, 6, 1, 1,
       9, 7, 2, 7, 8, 5, 5, 7, 5, 2, 3, 7, 2, 7, 5])
```

In [236]:
```
score = logisticRegr.score(test_img, test_lbl)
print(score)
```

```
0.9688715953307393
```

Task 3: Compare 5 different models' accuracies (40 pts for testing 4 models + 20 bonus pts for testing all 5)

Now, perform classification task on the same load_digits dataset using Logistic Regression with L1 penalty (Lasso), Decision Tree, KNN, SVM and Naive Bayes

For each model, tune the hyperparameters, select the best parameters for each model (except Naive Bayes) and re-train the model and compare the accuracies of all 5 models after re-training on best parameters.

Logistic Regression Model

In [237]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(penalty='l1',solver='liblinear')

model.fit(x_train, y_train)
```

Out[237]:
```
LogisticRegression(penalty='l1', solver='liblinear')
```

In [238]:
```
model.predict(x_test)
```

Out[238]:
```
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5, 2, 8, 6, 6, 6, 6, 1, 0, 5, 8, 8, 7,
       8, 4, 7, 5, 4, 9, 2, 9, 4, 7, 6, 8, 9, 4, 3, 8, 0, 1, 8, 6, 7, 7,
       1, 0, 7, 6, 2, 1, 9, 6, 7, 9, 0, 0, 5, 1, 6, 3, 0, 2, 3, 4, 1, 9,
       2, 6, 9, 1, 8, 3, 5, 1, 2, 8, 2, 2, 9, 7, 2, 3, 6, 0, 5, 3, 7, 5,
       1, 2, 9, 9, 3, 1, 4, 7, 4, 8, 5, 9, 5, 5, 2, 5, 9, 0, 7, 1, 4, 1,
       3, 4, 8, 9, 7, 9, 8, 2, 6, 5, 2, 5, 8, 4, 1, 7, 0, 6, 1, 5, 5, 9,
       9, 5, 9, 9, 5, 7, 5, 6, 2, 8, 6, 9, 6, 1, 5, 1, 5, 9, 8, 1, 5, 3,
       6, 1, 8, 9, 8, 7, 6, 7, 6, 5, 6, 0, 8, 8, 9, 8, 6, 1, 0, 4, 1, 6,
       3, 8, 6, 7, 4, 1, 6, 3, 0, 3, 3, 3, 0, 7, 7, 5, 7, 8, 0, 7, 1, 9,
       6, 4, 5, 0, 1, 4, 6, 4, 3, 3, 0, 9, 5, 3, 2, 8, 4, 2, 1, 6, 9, 9,
       2, 4, 9, 3, 7, 6, 2, 3, 3, 1, 6, 9, 3, 6, 3, 3, 2, 0, 7, 6, 1, 1,
       9, 7, 2, 7, 8, 5, 5, 7, 5, 3, 3, 7, 2, 7, 5, 5, 7, 0, 9, 1, 6, 5,
       9, 7, 4, 3, 8, 0, 3, 6, 4, 6, 3, 2, 6, 8, 8, 8, 4, 6, 7, 5, 2, 4,
       5, 3, 2, 4, 6, 9, 4, 5, 4, 3, 4, 6, 2, 9, 0, 6, 7, 2, 0, 9, 6, 0,
       4, 2, 0, 7, 9, 8, 5, 7, 8, 2, 8, 4, 3, 7, 2, 6, 9, 1, 5, 1, 0, 8,
       2, 8, 9, 5, 6, 2, 2, 7, 2, 1, 5, 1, 6, 4, 5, 0, 9, 4, 1, 1, 7, 0,
       8, 9, 0, 5, 4, 3, 8, 8, 6, 5, 3, 4, 4, 4, 8, 8, 7, 0, 9, 6, 3, 5,
       2, 3, 0, 8, 2, 3, 1, 3, 3, 0, 0, 4, 6, 0, 7, 7, 6, 2, 0, 4, 4, 2,
       3, 7, 1, 9, 8, 6, 8, 5, 6, 2, 2, 3, 1, 7, 7, 8, 0, 9, 3, 2, 1, 5,
       5, 9, 1, 3, 7, 0, 0, 3, 0, 4, 5, 9, 3, 3, 4, 3, 1, 8, 9, 5, 3, 6,
       3, 1, 6, 2, 1, 7, 5, 5, 1, 9])
```

In [239]:
```
model.score(x_test,y_test)
```

Out[239]:
```
0.9466666666666667
```

Decision Tree Model, Random state = 7

In [240]:
```
from sklearn.tree import DecisionTreeRegressor
reg7 = DecisionTreeRegressor(random_state = 7)
reg7.fit(x_train,y_train)
```

Out[240]:
```
DecisionTreeRegressor(random_state=7)
```

In [241]:
```
reg7.predict(x_test)
```

Out[241]:
```
array([2., 8., 2., 6., 6., 7., 1., 9., 9., 5., 2., 8., 6., 6., 6., 6., 1.,
       0., 5., 8., 8., 8., 7., 4., 7., 5., 4., 9., 2., 9., 4., 7., 6., 3.,
       9., 4., 3., 1., 0., 1., 9., 6., 7., 7., 9., 0., 7., 5., 2., 1., 9.,
       6., 7., 9., 0., 0., 5., 1., 6., 3., 0., 2., 3., 4., 1., 9., 9., 6.,
       0., 1., 8., 3., 5., 1., 2., 8., 2., 2., 9., 7., 2., 3., 6., 0., 8.,
       3., 7., 5., 1., 2., 0., 9., 3., 1., 7., 7., 4., 7., 5., 8., 5., 5.,
       2., 3., 9., 0., 5., 1., 4., 7., 3., 4., 8., 9., 7., 9., 8., 0., 6.,
       3., 3., 5., 3., 4., 8., 7., 0., 6., 1., 5., 8., 1., 3., 5., 9., 9.,
       5., 7., 5., 6., 2., 8., 6., 9., 6., 2., 5., 1., 5., 9., 9., 1., 5.,
       3., 6., 1., 7., 7., 7., 7., 6., 7., 6., 5., 6., 0., 8., 8., 9., 9.,
       6., 1., 0., 7., 1., 6., 3., 8., 6., 7., 4., 9., 6., 3., 0., 3., 3.,
       5., 0., 7., 7., 5., 7., 8., 0., 7., 9., 9., 6., 4., 5., 0., 1., 4.,
       6., 4., 3., 3., 0., 9., 5., 3., 2., 3., 4., 8., 1., 0., 8., 9., 2.,
       4., 7., 2., 7., 6., 1., 3., 3., 1., 6., 9., 3., 6., 3., 2., 2., 0.,
       7., 6., 1., 1., 3., 7., 3., 7., 8., 6., 5., 7., 5., 2., 3., 7., 8.,
       8., 5., 5., 7., 0., 9., 1., 6., 5., 9., 7., 4., 3., 8., 0., 3., 6.,
       4., 6., 3., 2., 6., 8., 9., 8., 4., 6., 7., 5., 2., 4., 5., 3., 2.,
       4., 6., 9., 0., 5., 7., 3., 4., 6., 2., 9., 0., 7., 7., 9., 0., 9.,
       6., 6., 4., 2., 0., 7., 1., 8., 5., 9., 8., 2., 9., 4., 3., 9., 2.,
       6., 9., 1., 5., 1., 0., 8., 2., 6., 9., 6., 6., 5., 2., 7., 2., 1.,
       5., 1., 6., 4., 5., 0., 9., 4., 1., 1., 7., 0., 9., 9., 0., 6., 4.,
       3., 8., 6., 6., 5., 3., 4., 4., 6., 8., 8., 7., 0., 9., 6., 5., 5.,
       2., 3., 0., 8., 3., 3., 1., 3., 3., 0., 0., 4., 6., 0., 3., 3., 6.,
       2., 0., 4., 4., 2., 3., 7., 8., 9., 9., 6., 3., 5., 6., 2., 2., 3.,
       1., 7., 7., 3., 0., 8., 3., 2., 1., 5., 9., 9., 1., 3., 7., 0., 0.,
       7., 0., 9., 5., 7., 5., 3., 4., 3., 1., 8., 9., 2., 8., 6., 2., 1.,
       6., 8., 1., 7., 1., 3., 4., 9.])
```

In [242]:
```
reg7.score(x_test,y_test)
```

Out[242]:
```
0.6507339261651889
```

Decision Tree Model, Random state = 5

In [243]:
```
from sklearn.tree import DecisionTreeRegressor
reg5 = DecisionTreeRegressor(random_state = 5)
reg5.fit(x_train,y_train)
```

Out[243]:
```
DecisionTreeRegressor(random_state=5)
```

In [244]:
```
reg5.predict(x_test)
```

Out[244]:
```
array([2., 8., 2., 6., 6., 7., 1., 9., 8., 5., 2., 8., 6., 6., 6., 6., 1.,
       0., 5., 8., 8., 7., 8., 4., 7., 5., 4., 9., 3., 9., 4., 7., 6., 8.,
       9., 4., 3., 1., 0., 1., 9., 6., 7., 7., 2., 6., 7., 7., 2., 1., 9.,
       7., 7., 9., 0., 0., 5., 1., 6., 3., 0., 2., 3., 4., 1., 9., 8., 6.,
       0., 1., 8., 3., 5., 1., 2., 7., 2., 2., 9., 7., 2., 3., 6., 0., 0.,
       3., 7., 5., 1., 2., 0., 9., 3., 1., 7., 7., 4., 8., 5., 8., 5., 6.,
       2., 6., 9., 0., 5., 1., 4., 7., 3., 4., 8., 9., 7., 9., 8., 0., 6.,
       3., 2., 5., 3., 4., 7., 7., 0., 6., 1., 5., 5., 1., 3., 5., 9., 9.,
       3., 7., 5., 6., 2., 8., 6., 9., 6., 1., 5., 1., 5., 9., 9., 1., 5.,
       3., 6., 1., 7., 5., 8., 7., 6., 5., 6., 5., 6., 0., 8., 8., 9., 8.,
       6., 1., 0., 7., 1., 6., 3., 8., 6., 7., 4., 9., 6., 3., 6., 3., 3.,
       5., 0., 7., 7., 5., 7., 8., 0., 7., 9., 9., 6., 4., 5., 0., 1., 4.,
       6., 4., 3., 3., 0., 9., 5., 3., 2., 3., 4., 8., 1., 2., 8., 9., 2.,
       4., 7., 6., 7., 6., 1., 3., 3., 1., 6., 9., 3., 6., 3., 2., 2., 0.,
       7., 6., 1., 1., 3., 7., 2., 7., 8., 6., 5., 7., 5., 2., 3., 7., 8.,
       7., 5., 5., 7., 0., 9., 1., 6., 5., 9., 7., 8., 3., 8., 0., 3., 6.,
       4., 4., 3., 2., 6., 8., 8., 8., 4., 6., 7., 5., 2., 4., 5., 3., 2.,
       4., 6., 9., 0., 5., 4., 3., 4., 6., 2., 9., 0., 7., 7., 9., 0., 9.,
       0., 6., 4., 2., 0., 7., 2., 8., 6., 9., 8., 2., 8., 4., 3., 9., 2.,
       6., 9., 1., 5., 1., 0., 8., 1., 5., 9., 5., 6., 5., 2., 7., 2., 1.,
       5., 1., 6., 4., 5., 0., 9., 4., 1., 1., 7., 0., 9., 9., 0., 6., 4.,
       7., 8., 7., 6., 5., 7., 4., 4., 6., 8., 8., 7., 0., 9., 6., 5., 5.,
       2., 3., 0., 8., 8., 3., 1., 3., 3., 0., 0., 4., 6., 0., 2., 2., 6.,
       1., 6., 4., 4., 2., 3., 7., 8., 9., 9., 6., 2., 6., 6., 2., 2., 3.,
       1., 7., 7., 2., 0., 8., 3., 2., 1., 5., 5., 9., 1., 3., 7., 0., 0.,
       7., 0., 9., 5., 7., 5., 3., 4., 3., 1., 8., 9., 5., 8., 6., 2., 1.,
       6., 8., 1., 3., 1., 3., 8., 9.])
```

In [245]:
```
reg5.predict(x_test)
```

Out[245]:
```
array([2., 8., 2., 6., 6., 7., 1., 9., 8., 5., 2., 8., 6., 6., 6., 6., 1.,
       0., 5., 8., 8., 7., 8., 4., 7., 5., 4., 9., 3., 9., 4., 7., 6., 8.,
       9., 4., 3., 1., 0., 1., 9., 6., 7., 7., 2., 6., 7., 7., 2., 1., 9.,
       7., 7., 9., 0., 0., 5., 1., 6., 3., 0., 2., 3., 4., 1., 9., 8., 6.,
       0., 1., 8., 3., 5., 1., 2., 7., 2., 2., 9., 7., 2., 3., 6., 0., 0.,
       3., 7., 5., 1., 2., 0., 9., 3., 1., 7., 7., 4., 8., 5., 8., 5., 6.,
       2., 6., 9., 0., 5., 1., 4., 7., 3., 4., 8., 9., 7., 9., 8., 0., 6.,
       3., 2., 5., 3., 4., 7., 7., 0., 6., 1., 5., 5., 1., 3., 5., 9., 9.,
       3., 7., 5., 6., 2., 8., 6., 9., 6., 1., 5., 1., 5., 9., 9., 1., 5.,
       3., 6., 1., 7., 5., 8., 7., 6., 5., 6., 5., 6., 0., 8., 8., 9., 8.,
       6., 1., 0., 7., 1., 6., 3., 8., 6., 7., 4., 9., 6., 3., 6., 3., 3.,
       5., 0., 7., 7., 5., 7., 8., 0., 7., 9., 9., 6., 4., 5., 0., 1., 4.,
       6., 4., 3., 3., 0., 9., 5., 3., 2., 3., 4., 8., 1., 2., 8., 9., 2.,
       4., 7., 6., 7., 6., 1., 3., 3., 1., 6., 9., 3., 6., 3., 2., 2., 0.,
       7., 6., 1., 1., 3., 7., 2., 7., 8., 6., 5., 7., 5., 2., 3., 7., 8.,
       7., 5., 5., 7., 0., 9., 1., 6., 5., 9., 7., 8., 3., 8., 0., 3., 6.,
       4., 4., 3., 2., 6., 8., 8., 8., 4., 6., 7., 5., 2., 4., 5., 3., 2.,
       4., 6., 9., 0., 5., 4., 3., 4., 6., 2., 9., 0., 7., 7., 9., 0., 9.,
       0., 6., 4., 2., 0., 7., 2., 8., 6., 9., 8., 2., 8., 4., 3., 9., 2.,
       6., 9., 1., 5., 1., 0., 8., 1., 5., 9., 5., 6., 5., 2., 7., 2., 1.,
       5., 1., 6., 4., 5., 0., 9., 4., 1., 1., 7., 0., 9., 9., 0., 6., 4.,
       7., 8., 7., 6., 5., 7., 4., 4., 6., 8., 8., 7., 0., 9., 6., 5., 5.,
       2., 3., 0., 8., 8., 3., 1., 3., 3., 0., 0., 4., 6., 0., 2., 2., 6.,
       1., 6., 4., 4., 2., 3., 7., 8., 9., 9., 6., 2., 6., 6., 2., 2., 3.,
       1., 7., 7., 2., 0., 8., 3., 2., 1., 5., 5., 9., 1., 3., 7., 0., 0.,
       7., 0., 9., 5., 7., 5., 3., 4., 3., 1., 8., 9., 5., 8., 6., 2., 1.,
       6., 8., 1., 3., 1., 3., 8., 9.])
```

Decision Tree Model, Random state = 3

In [246]:
```
from sklearn.tree import DecisionTreeRegressor
reg3 = DecisionTreeRegressor(random_state = 3)
reg3.fit(x_train,y_train)
```

Out[246]:
```
DecisionTreeRegressor(random_state=3)
```

In [247]:
```
reg3.predict(x_test)
```

Out[247]:
```
array([2., 8., 2., 6., 6., 7., 1., 9., 8., 3., 2., 8., 6., 6., 6., 6., 1.,
       0., 5., 8., 8., 8., 8., 4., 7., 5., 4., 9., 2., 9., 4., 7., 6., 8.,
       9., 4., 3., 1., 0., 1., 9., 6., 7., 7., 9., 0., 7., 7., 2., 1., 9.,
       6., 7., 9., 0., 0., 5., 1., 6., 3., 0., 2., 3., 4., 1., 9., 8., 6.,
       3., 1., 8., 3., 5., 1., 2., 8., 2., 2., 9., 7., 2., 3., 6., 0., 0.,
       3., 7., 5., 1., 2., 9., 9., 3., 1., 7., 7., 4., 8., 5., 8., 5., 5.,
       2., 5., 9., 0., 5., 1., 4., 7., 3., 4., 8., 9., 7., 9., 8., 0., 6.,
       5., 2., 5., 8., 4., 8., 7., 0., 6., 1., 5., 7., 1., 3., 5., 9., 9.,
       7., 7., 5., 6., 2., 8., 6., 9., 6., 1., 5., 1., 5., 9., 9., 1., 5.,
       3., 6., 1., 7., 7., 8., 7., 6., 8., 6., 5., 6., 0., 8., 8., 9., 9.,
       6., 1., 0., 7., 1., 6., 3., 8., 6., 7., 4., 9., 6., 3., 0., 5., 3.,
       3., 0., 7., 7., 5., 8., 8., 0., 7., 9., 9., 6., 4., 3., 0., 1., 4.,
       6., 4., 3., 3., 0., 9., 5., 3., 2., 3., 4., 8., 1., 5., 8., 9., 2.,
       4., 3., 6., 7., 6., 1., 3., 3., 1., 6., 9., 3., 6., 3., 2., 2., 0.,
       7., 6., 1., 1., 3., 7., 2., 7., 8., 5., 5., 7., 5., 2., 3., 7., 5.,
       7., 6., 5., 7., 0., 9., 1., 6., 5., 9., 8., 4., 3., 8., 6., 3., 6.,
       4., 6., 3., 2., 6., 8., 9., 8., 4., 6., 7., 5., 2., 4., 5., 3., 2.,
       4., 6., 9., 0., 6., 4., 3., 4., 6., 2., 9., 0., 8., 7., 9., 0., 9.,
       6., 2., 4., 2., 0., 7., 3., 8., 5., 9., 8., 2., 9., 4., 3., 9., 2.,
       6., 9., 1., 0., 1., 0., 8., 3., 6., 9., 5., 6., 6., 2., 7., 2., 1.,
       5., 1., 6., 4., 5., 0., 9., 4., 1., 1., 7., 0., 9., 9., 0., 7., 4.,
       7., 8., 7., 6., 5., 7., 4., 4., 5., 8., 8., 7., 0., 9., 6., 3., 5.,
       2., 3., 0., 8., 7., 3., 1., 3., 7., 0., 0., 4., 6., 0., 2., 6., 6.,
       8., 6., 4., 4., 2., 3., 7., 8., 9., 9., 6., 3., 5., 6., 2., 2., 3.,
       1., 8., 7., 3., 0., 3., 3., 2., 1., 5., 5., 9., 1., 3., 7., 0., 0.,
       7., 0., 9., 5., 7., 3., 3., 4., 3., 1., 8., 9., 5., 8., 6., 2., 1.,
       6., 8., 1., 7., 2., 5., 8., 9.])
```

In [248]:
```
reg3.predict(x_test)
```

Out[248]:
```
array([2., 8., 2., 6., 6., 7., 1., 9., 8., 3., 2., 8., 6., 6., 6., 6., 1.,
       0., 5., 8., 8., 8., 8., 4., 7., 5., 4., 9., 2., 9., 4., 7., 6., 8.,
       9., 4., 3., 1., 0., 1., 9., 6., 7., 7., 9., 0., 7., 7., 2., 1., 9.,
       6., 7., 9., 0., 0., 5., 1., 6., 3., 0., 2., 3., 4., 1., 9., 8., 6.,
       3., 1., 8., 3., 5., 1., 2., 8., 2., 2., 9., 7., 2., 3., 6., 0., 0.,
       3., 7., 5., 1., 2., 9., 9., 3., 1., 7., 7., 4., 8., 5., 8., 5., 5.,
       2., 5., 9., 0., 5., 1., 4., 7., 3., 4., 8., 9., 7., 9., 8., 0., 6.,
       5., 2., 5., 8., 4., 8., 7., 0., 6., 1., 5., 7., 1., 3., 5., 9., 9.,
       7., 7., 5., 6., 2., 8., 6., 9., 6., 1., 5., 1., 5., 9., 9., 1., 5.,
       3., 6., 1., 7., 7., 8., 7., 6., 8., 6., 5., 6., 0., 8., 8., 9., 9.,
       6., 1., 0., 7., 1., 6., 3., 8., 6., 7., 4., 9., 6., 3., 0., 5., 3.,
       3., 0., 7., 7., 5., 8., 8., 0., 7., 9., 9., 6., 4., 3., 0., 1., 4.,
       6., 4., 3., 3., 0., 9., 5., 3., 2., 3., 4., 8., 1., 5., 8., 9., 2.,
       4., 3., 6., 7., 6., 1., 3., 3., 1., 6., 9., 3., 6., 3., 2., 2., 0.,
       7., 6., 1., 1., 3., 7., 2., 7., 8., 5., 5., 7., 5., 2., 3., 7., 5.,
       7., 6., 5., 7., 0., 9., 1., 6., 5., 9., 8., 4., 3., 8., 6., 3., 6.,
       4., 6., 3., 2., 6., 8., 9., 8., 4., 6., 7., 5., 2., 4., 5., 3., 2.,
       4., 6., 9., 0., 6., 4., 3., 4., 6., 2., 9., 0., 8., 7., 9., 0., 9.,
       6., 2., 4., 2., 0., 7., 3., 8., 5., 9., 8., 2., 9., 4., 3., 9., 2.,
       6., 9., 1., 0., 1., 0., 8., 3., 6., 9., 5., 6., 6., 2., 7., 2., 1.,
       5., 1., 6., 4., 5., 0., 9., 4., 1., 1., 7., 0., 9., 9., 0., 7., 4.,
       7., 8., 7., 6., 5., 7., 4., 4., 5., 8., 8., 7., 0., 9., 6., 3., 5.,
       2., 3., 0., 8., 7., 3., 1., 3., 7., 0., 0., 4., 6., 0., 2., 6., 6.,
       8., 6., 4., 4., 2., 3., 7., 8., 9., 9., 6., 3., 5., 6., 2., 2., 3.,
       1., 8., 7., 3., 0., 3., 3., 2., 1., 5., 5., 9., 1., 3., 7., 0., 0.,
       7., 0., 9., 5., 7., 3., 3., 4., 3., 1., 8., 9., 5., 8., 6., 2., 1.,
       6., 8., 1., 7., 2., 5., 8., 9.])
```

Decision Tree Model, Random state = 9

In [249]:
```
from sklearn.tree import DecisionTreeRegressor
reg9 = DecisionTreeRegressor(random_state = 9)
reg9.fit(x_train,y_train)
```

Out[249]:
```
DecisionTreeRegressor(random_state=9)
```

In [250]:
```
reg9.predict(x_test)
```

Out[250]:
```
array([2., 8., 2., 6., 6., 7., 1., 9., 8., 9., 2., 8., 6., 6., 6., 6., 1.,
       0., 5., 8., 8., 8., 8., 4., 7., 5., 4., 9., 2., 9., 4., 7., 6., 8.,
       9., 4., 3., 1., 0., 1., 9., 6., 7., 7., 9., 0., 4., 5., 2., 1., 9.,
       8., 7., 9., 0., 0., 8., 1., 6., 3., 0., 2., 3., 4., 1., 9., 9., 6.,
       0., 1., 8., 3., 5., 1., 2., 8., 2., 2., 9., 7., 2., 2., 6., 0., 0.,
       3., 7., 5., 1., 2., 9., 9., 3., 1., 7., 7., 4., 7., 5., 8., 5., 5.,
       2., 5., 9., 0., 7., 1., 4., 5., 3., 4., 8., 9., 7., 9., 8., 0., 6.,
       5., 2., 5., 5., 4., 1., 7., 0., 6., 1., 5., 5., 1., 3., 5., 9., 9.,
       5., 7., 5., 6., 2., 8., 6., 9., 6., 1., 5., 1., 5., 9., 9., 1., 5.,
       3., 6., 1., 7., 5., 8., 7., 6., 5., 6., 5., 6., 0., 8., 8., 9., 8.,
       6., 1., 0., 7., 1., 6., 3., 8., 6., 7., 4., 9., 6., 3., 0., 5., 3.,
       5., 0., 7., 7., 9., 7., 8., 0., 7., 9., 9., 6., 4., 5., 0., 1., 4.,
       6., 4., 3., 3., 0., 9., 5., 3., 2., 3., 4., 8., 1., 0., 8., 9., 2.,
       4., 7., 6., 7., 6., 1., 3., 3., 1., 6., 9., 3., 6., 3., 2., 2., 0.,
       7., 6., 1., 1., 3., 7., 2., 7., 8., 5., 5., 7., 5., 2., 3., 7., 8.,
       8., 6., 5., 7., 0., 5., 1., 6., 5., 9., 8., 8., 3., 8., 0., 3., 6.,
       4., 5., 3., 2., 6., 8., 7., 8., 4., 6., 7., 5., 2., 4., 5., 3., 2.,
       4., 6., 9., 0., 6., 4., 3., 4., 6., 2., 9., 0., 8., 9., 5., 0., 9.,
       6., 4., 4., 2., 0., 7., 1., 8., 5., 9., 8., 2., 9., 4., 3., 9., 2.,
       6., 9., 1., 3., 1., 0., 8., 2., 5., 9., 5., 6., 6., 2., 7., 2., 1.,
       5., 1., 6., 4., 3., 0., 9., 4., 1., 1., 7., 0., 9., 9., 0., 5., 4.,
       7., 8., 6., 6., 5., 3., 4., 4., 6., 8., 8., 7., 0., 9., 6., 3., 5.,
       6., 3., 0., 8., 8., 3., 1., 3., 5., 0., 0., 4., 6., 0., 3., 2., 6.,
       2., 0., 4., 4., 2., 3., 7., 8., 9., 9., 6., 3., 5., 6., 2., 2., 3.,
       1., 8., 7., 3., 0., 3., 3., 2., 1., 5., 5., 9., 1., 3., 7., 0., 0.,
       7., 0., 9., 5., 6., 3., 3., 4., 3., 1., 8., 9., 5., 8., 6., 2., 1.,
       6., 8., 1., 7., 1., 5., 8., 9.])
```

In [251]:
```
reg9.predict(x_test)
```

Out[251]:
```
array([2., 8., 2., 6., 6., 7., 1., 9., 8., 9., 2., 8., 6., 6., 6., 6., 1.,
       0., 5., 8., 8., 8., 8., 4., 7., 5., 4., 9., 2., 9., 4., 7., 6., 8.,
       9., 4., 3., 1., 0., 1., 9., 6., 7., 7., 9., 0., 4., 5., 2., 1., 9.,
       8., 7., 9., 0., 0., 8., 1., 6., 3., 0., 2., 3., 4., 1., 9., 9., 6.,
       0., 1., 8., 3., 5., 1., 2., 8., 2., 2., 9., 7., 2., 2., 6., 0., 0.,
       3., 7., 5., 1., 2., 9., 9., 3., 1., 7., 7., 4., 7., 5., 8., 5., 5.,
       2., 5., 9., 0., 7., 1., 4., 5., 3., 4., 8., 9., 7., 9., 8., 0., 6.,
       5., 2., 5., 5., 4., 1., 7., 0., 6., 1., 5., 5., 1., 3., 5., 9., 9.,
       5., 7., 5., 6., 2., 8., 6., 9., 6., 1., 5., 1., 5., 9., 9., 1., 5.,
       3., 6., 1., 7., 5., 8., 7., 6., 5., 6., 5., 6., 0., 8., 8., 9., 8.,
       6., 1., 0., 7., 1., 6., 3., 8., 6., 7., 4., 9., 6., 3., 0., 5., 3.,
       5., 0., 7., 7., 9., 7., 8., 0., 7., 9., 9., 6., 4., 5., 0., 1., 4.,
       6., 4., 3., 3., 0., 9., 5., 3., 2., 3., 4., 8., 1., 0., 8., 9., 2.,
       4., 7., 6., 7., 6., 1., 3., 3., 1., 6., 9., 3., 6., 3., 2., 2., 0.,
       7., 6., 1., 1., 3., 7., 2., 7., 8., 5., 5., 7., 5., 2., 3., 7., 8.,
       8., 6., 5., 7., 0., 5., 1., 6., 5., 9., 8., 8., 3., 8., 0., 3., 6.,
       4., 5., 3., 2., 6., 8., 7., 8., 4., 6., 7., 5., 2., 4., 5., 3., 2.,
       4., 6., 9., 0., 6., 4., 3., 4., 6., 2., 9., 0., 8., 9., 5., 0., 9.,
       6., 4., 4., 2., 0., 7., 1., 8., 5., 9., 8., 2., 9., 4., 3., 9., 2.,
       6., 9., 1., 3., 1., 0., 8., 2., 5., 9., 5., 6., 6., 2., 7., 2., 1.,
       5., 1., 6., 4., 3., 0., 9., 4., 1., 1., 7., 0., 9., 9., 0., 5., 4.,
       7., 8., 6., 6., 5., 3., 4., 4., 6., 8., 8., 7., 0., 9., 6., 3., 5.,
       6., 3., 0., 8., 8., 3., 1., 3., 5., 0., 0., 4., 6., 0., 3., 2., 6.,
       2., 0., 4., 4., 2., 3., 7., 8., 9., 9., 6., 3., 5., 6., 2., 2., 3.,
       1., 8., 7., 3., 0., 3., 3., 2., 1., 5., 5., 9., 1., 3., 7., 0., 0.,
       7., 0., 9., 5., 6., 3., 3., 4., 3., 1., 8., 9., 5., 8., 6., 2., 1.,
       6., 8., 1., 7., 1., 5., 8., 9.])
```

KNN Neighbours Model, Neighbor = 7

In [252]:
```
from sklearn.neighbors import KNeighborsClassifier

knn7 = KNeighborsClassifier(n_neighbors=7)
knn7.fit(x_train, y_train)
print(knn7.predict(x_test))
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 5 1 6 3 0 2 3 4 1 9 7 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 6 5 2 5 3 4 1 7 0 6 1 5 9 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 7 7 6 7 6 5 6 0 8 1 9 3 6 1 0 4 1 6 3 8 6 7 4 9 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 3 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 5 8 5 7 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 1 9
 5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 3 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 1 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 2 1 5 5 9 1 3 7 0 0 7 0 4 5 9 3 3 4 3 1 8 9 8 3 6 2 1 6 2
 1 7 5 5 1 9]
```

In [253]:
```
knn7.score(x_test,y_test)
```

Out[253]:
```
0.9777777777777777
```

KNN Neighbors Model, Neighbor = 5

In [254]:
```
from sklearn.neighbors import KNeighborsClassifier

knn5 = KNeighborsClassifier(n_neighbors=5)
knn5.fit(x_train, y_train)
print(knn5.predict(x_test))
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 5 1 6 3 0 2 3 4 1 9 3 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 6 5 2 5 3 4 1 7 0 6 1 5 9 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 7 7 6 7 6 5 6 0 8 8 9 3 6 1 0 4 1 6 3 8 6 7 4 9 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 3 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 5 8 5 7 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 5 9
 5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 3 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 8 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 2 1 5 5 9 1 3 7 0 0 7 0 4 5 9 3 3 4 3 1 8 9 8 3 6 2 1 6 2
 1 7 5 5 1 9]
```

In [255]:
```
knn5.score(x_test,y_test)
```

Out[255]: 0.98

KNN Neigbours Model, Neighbor = 3

In [256]:
```
from sklearn.neighbors import KNeighborsClassifier

knn3 = KNeighborsClassifier(n_neighbors=3)
knn3.fit(x_train, y_train)
print(knn3.predict(x_test))
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 5 1 6 3 0 2 3 4 1 9 2 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 6 5 2 5 3 4 8 7 0 6 1 5 9 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 8 7 6 7 6 5 6 0 8 8 9 3 6 1 0 4 1 6 3 8 6 7 4 9 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 2 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 9 8 5 7 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 5 9
 5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 3 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 8 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 2 1 5 5 9 1 3 7 0 0 7 0 4 5 9 3 3 4 3 1 8 9 8 3 6 2 1 6 2
 1 7 5 5 1 9]
```

In [257]:
```
knn3.score(x_test,y_test)
```

Out[257]:
```
0.9866666666666667
```

KNN Neighbours Model, Neighbor = 1

In [258]:
```
from sklearn.neighbors import KNeighborsClassifier

knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(x_train, y_train)
print(knn1.predict(x_test))
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 5 1 6 3 0 2 3 4 1 9 3 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 6 5 2 5 8 4 8 7 0 6 1 5 3 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 8 7 6 7 6 5 6 0 8 8 9 8 6 1 0 4 1 6 3 8 6 7 4 9 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 3 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 5 8 5 4 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 1 9
 5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 3 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 8 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 2 1 5 5 9 1 3 7 0 0 7 0 4 5 9 3 3 4 3 1 8 9 8 3 6 2 1 6 2
 1 7 5 5 1 9]
```

In [259]:
```
knn1.score(x_test,y_test)
```

Out[259]:
```
0.9911111111111112
```

KNN Neighbours Model, Neighbor = 9

In [260]:
```
from sklearn.neighbors import KNeighborsClassifier

knn9 = KNeighborsClassifier(n_neighbors=9)
knn9.fit(x_train, y_train)
print(knn9.predict(x_test))
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 5 1 6 3 0 2 3 4 1 9 7 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 6 5 2 5 3 4 1 7 0 6 1 5 9 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 7 7 6 7 6 5 6 0 8 1 9 3 6 1 0 4 1 6 3 8 6 7 4 9 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 3 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 5 8 5 7 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 1 9
 5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 3 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 1 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 2 1 5 5 9 1 3 7 0 0 7 0 4 5 9 3 3 4 3 1 8 9 8 3 6 2 1 6 2
 1 7 5 5 1 9]
```

In [261]:
```
knn9.score(x_test,y_test)
```

Out[261]: 0.9777777777777777

SVM Model

In [262]:
```
clf = svm.SVC(kernel='rbf', probability=True)

clf.fit(x_train, y_train)
```

Out[262]: SVC(probability=True)

In [263]:
```
clf.score(x_test,y_test)
```

Out[263]: 0.9911111111111112

Naive Bayes Model

In [264]:
```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(x_train,y_train)
print(gnb.predict(x_test))
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 3 6 7 9 0 0 5 8 6 3 0 2 3 4 1 9 8 6 8 8 8 3 5 1
 2 1 2 1 9 7 1 3 6 0 5 3 7 5 1 8 9 9 3 1 7 7 4 8 5 1 5 5 8 5 8 0 7 1 7 7 3
 4 8 9 7 7 8 1 6 5 8 5 5 4 1 7 0 6 8 5 8 1 1 5 9 9 5 7 5 6 8 8 6 7 6 1 5 1
 7 9 9 1 5 3 6 1 8 9 7 7 6 7 6 5 6 0 8 8 3 8 6 1 0 7 1 6 3 8 6 7 4 3 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 1 9 6 4 7 0 1 4 6 4 3 8 0 9 5 3 1 1 4 8 1 6 8 9 2 4
 9 3 7 6 8 3 3 1 6 9 8 6 3 1 2 0 7 6 1 1 8 7 1 7 1 5 5 7 5 3 8 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 1 6 8 8 8 4 6 7 5 2 1 7 3 8 4 6 9 4 5 7 3
 4 6 2 8 0 1 7 8 0 3 6 0 4 8 0 7 8 7 5 7 8 2 8 4 3 7 2 6 7 1 1 1 0 8 2 8 8
 5 6 8 3 7 8 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 7 8 8 8 2 5 3 7 4 4 8 8 7 0
 4 6 3 5 2 3 0 8 8 3 1 3 3 0 0 4 6 0 7 7 6 8 0 4 4 2 3 7 1 9 8 6 3 5 6 2 2
 3 1 7 7 8 0 3 3 8 1 5 5 9 1 3 7 0 0 4 0 4 5 9 3 3 4 7 1 8 9 8 3 6 8 1 6 8
 1 7 5 5 1 9]
```

In [265]:
```
gnb.score(x_test,y_test)
```

Out[265]: 0.8333333333333334

Comparision between models:

Logistic Regression Model Score = 0.9466666666666667

Decision Tree Model Score = 0.6479663661506501

KNN Nieghbors Model Score = 0.9777777777777777

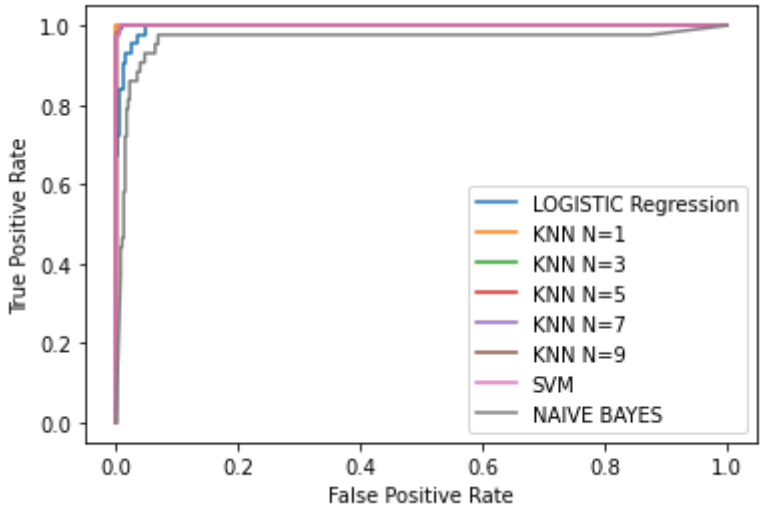
SVM Model Score = 0.9911111111111112

Naive Bayes Model Score = 0.8333333333333334

Plotting graph to visualize the comparision of models:

In [266]:
```
# LOGISTIC
y_probal = model.predict_proba(x_test)[::,1]
fprl, tprl, _ = metrics.roc_curve(y_test,y_probal,pos_label=1)
plt.plot(fprl,tprl,label="LOGISTIC Regression")

# KNN
y_probak3 = knn1.predict_proba(x_test)[::,1]
fprk1, tprk1, _ = metrics.roc_curve(y_test,y_probak3,pos_label=1)
plt.plot(fprk1,tprk1,label="KNN N=1")

y_probak3 = knn3.predict_proba(x_test)[::,1]
fprk3, tprk3, _ = metrics.roc_curve(y_test,y_probak3,pos_label=1)
plt.plot(fprk3,tprk3,label="KNN N=3")

y_probak3 = knn5.predict_proba(x_test)[::,1]
fprk5, tprk5, _ = metrics.roc_curve(y_test,y_probak5,pos_label=1)
plt.plot(fprk5,tprk5,label="KNN N=5")

y_probak7 = knn7.predict_proba(x_test)[::,1]
fprk7, tprk7, _ = metrics.roc_curve(y_test,y_probak5,pos_label=1)
plt.plot(fprk7,tprk7,label="KNN N=7")

y_probak9 = knn9.predict_proba(x_test)[::,1]
fprk9, tprk9, _ = metrics.roc_curve(y_test,y_probak5,pos_label=1)
plt.plot(fprk9,tprk9,label="KNN N=9")

# DECISION TREE
# y_probad3 = reg3.predict(x_test)[::,1]
# fprd3, tprd3, _ = metrics.roc_curve(y_test,y_probad3,pos_label=1)
# plt.plot(fprd3,tprd3,label="Decision Tree depth=3")

# SVM
y_proba = clf.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,y_proba,pos_label=1)
plt.plot(fpr,tpr,label="SVM")

# NAIVE BAYES
y_probag = gnb.predict_proba(x_test)[::,1]
fprg, tprg, _ = metrics.roc_curve(y_test,y_probag,pos_label=1)
plt.plot(fprg,tprg,label="NAIVE BAYES")

# PLOTTING GRAPH
```

```
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

plt.legend(loc='best')

plt.show()
```



In [ ]: