Machine Learning

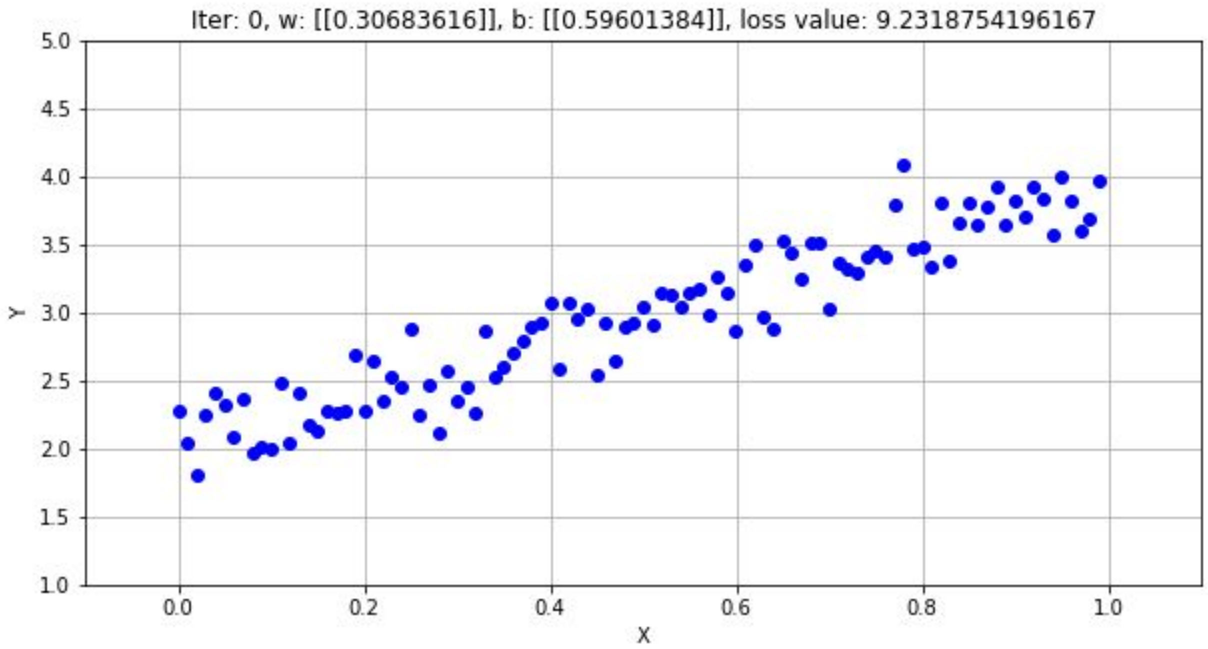
Linear Regression

Professor Ernesto Lee

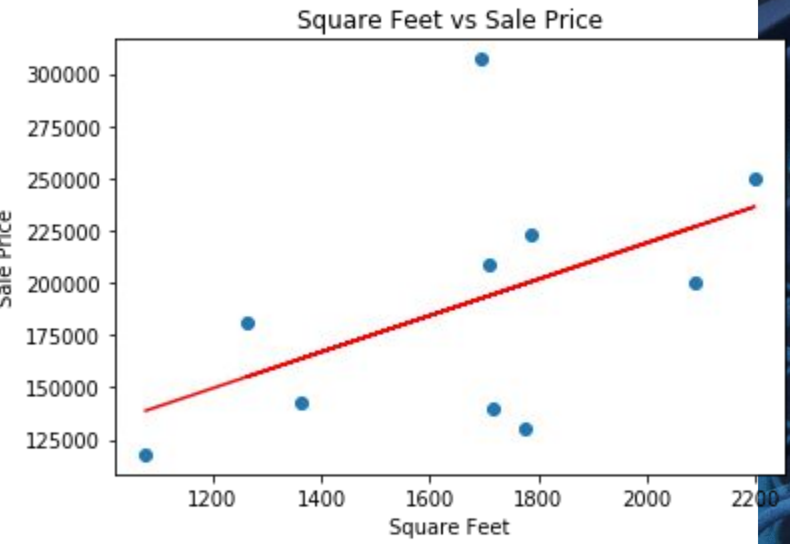
HTTPS://BIT.LY/MLTRAIN

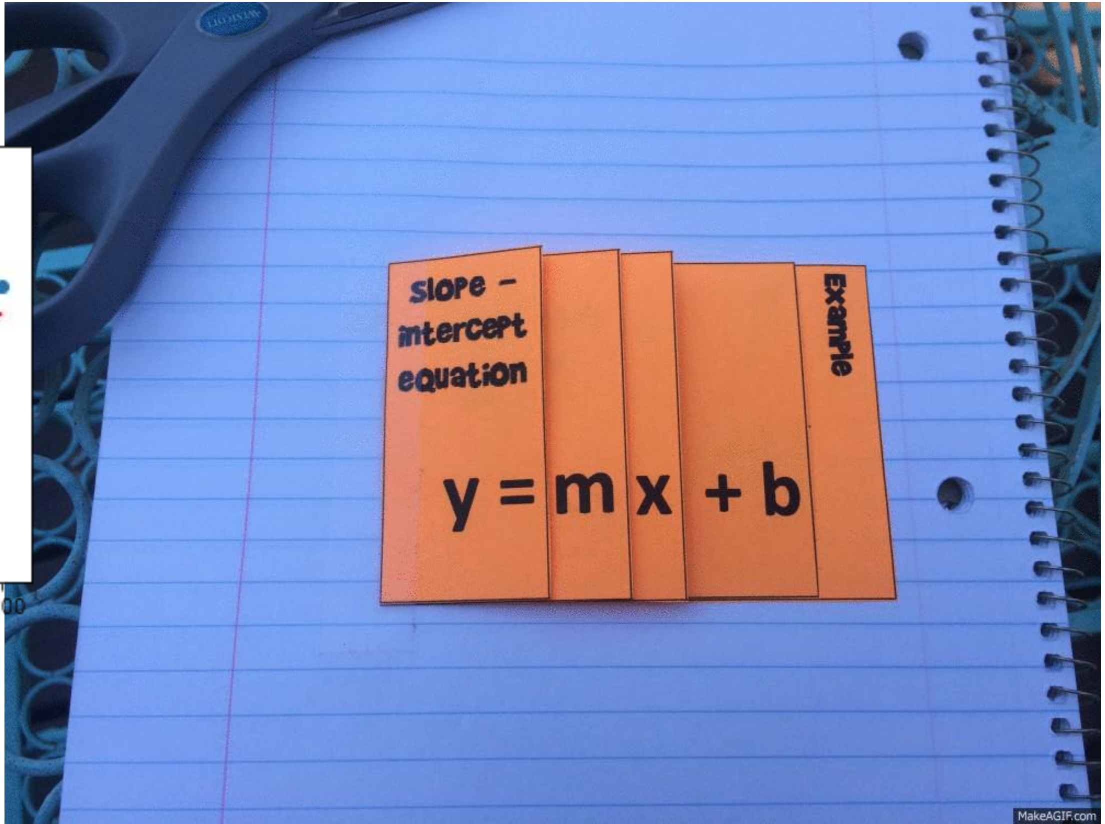
Go here for your LAB ENVIRONMENT

Linear Relationships



Regression Algorithm





Linear Regression with Scikit-Learn

Import Your Libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics
```

Pull in Your Data

You can access your data here:
https://bit.ly/21homes

Or here:

https://raw.githubusercontent.com/fenago/pythonml/main/data/HousePrice.csv

----

```
df =
pd.read_csv('https://raw.githubusercontent.com/fenago/pythonml/main/data/HousePrice.csv')
```

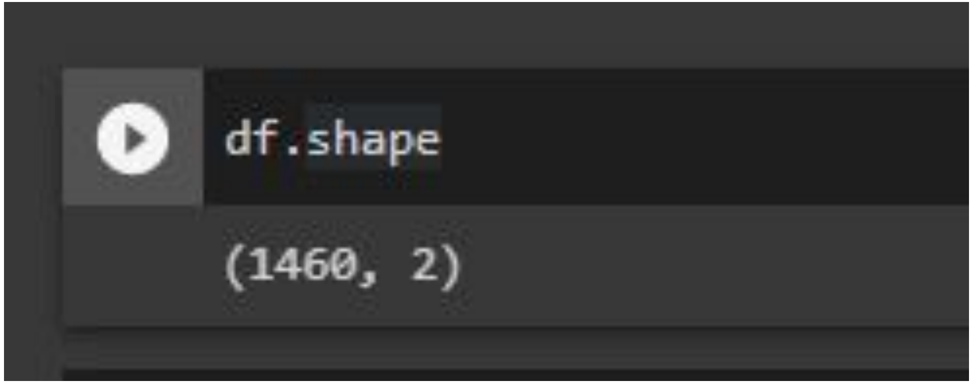
Putting the Columns You Want In Your Data

```
df2 = df[["open", "close"]] #open and close are the columns
```

Make sure you have the columns you want and need!

Understand Your Data with Shape

`df.shape`



Understand Your Data: HEAD

`df.head()`

```
df.head()
```

	SquareFeet	SalePrice
0	1710	208500
1	1262	181500
2	1786	223500
3	1717	140000
4	2198	250000

Understand Your Data: Descriptive Statistics

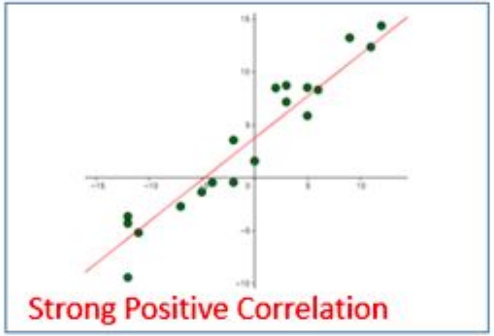
`df.describe()`

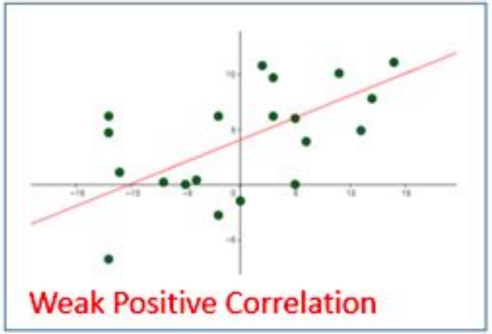
```
[8] df.describe()
```

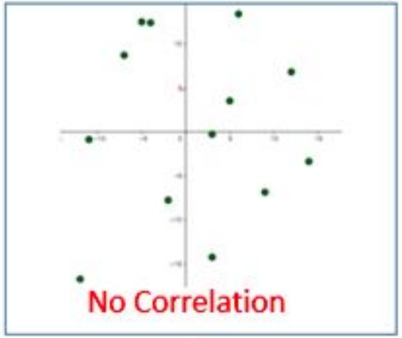
	SquareFeet	SalePrice
count	1460.000000	1460.000000
mean	1515.463699	180921.195890
std	525.480383	79442.502883
min	334.000000	34900.000000
25%	1129.500000	129975.000000
50%	1464.000000	163000.000000
75%	1776.750000	214000.000000
max	5642.000000	755000.000000

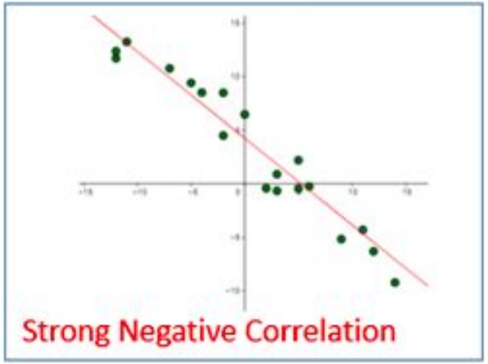
Find Your Correlations In your Datasets

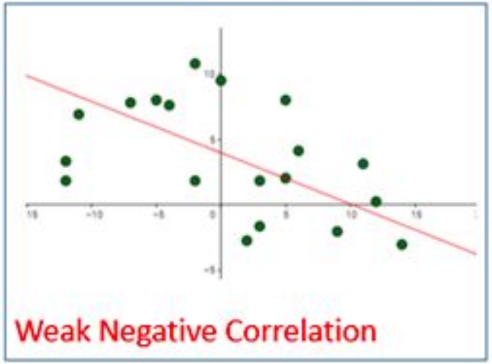
```
# correlation between 2 Specific Columns

print(df['SquareFeet'].corr(df['SalesPrice']))



# pair-wise correlation between all columns


print(df.corr())
```


Strong Positive Correlation


Weak Positive Correlation


No Correlation


Strong Negative Correlation


Weak Negative Correlation

HeatMap

```
# Correlation between different variables
corr = df.corr()
# Set up the matplotlib plot configuration
f, ax = plt.subplots(figsize=(12, 10))
# Generate a mask for upper traingle
mask = np.triu(np.ones_like(corr, dtype=bool))
# Configure a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)
# Draw the heatmap
sns.heatmap(corr, annot=True, mask = mask, cmap=cmap)
```

Visualize Your Data: PLOT

```
df.plot(x='SquareFeet', y='SalePrice', style='*')

plt.title('Square Feet vs Sale Price')

plt.xlabel('Square Feet')

plt.ylabel('Sale Price')

plt.show()
```

Prepare Your Data: Split Into Training and Test Sets

```
X = df.iloc[:, :-1].values

y = df.iloc[:, 1].values
```

Train Test Split

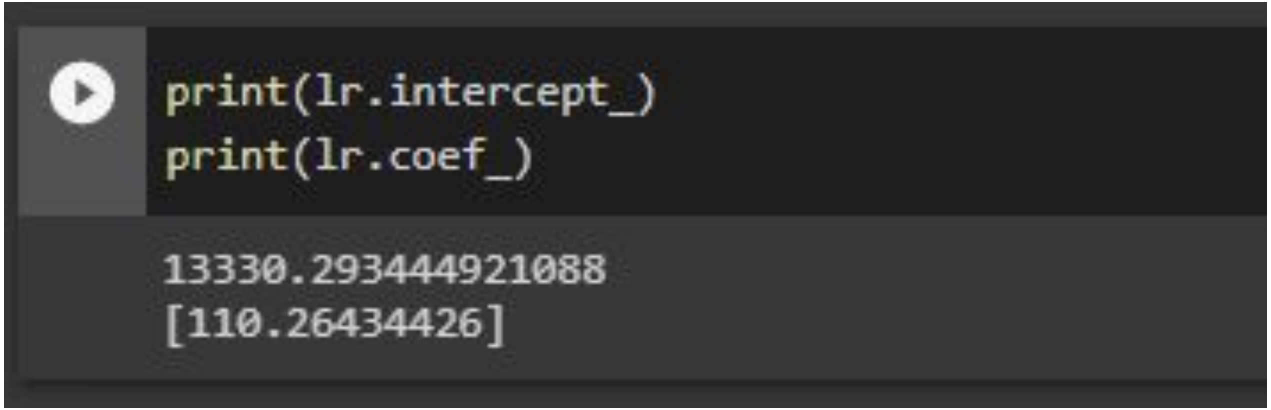
```
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=0)
```

Run the Model

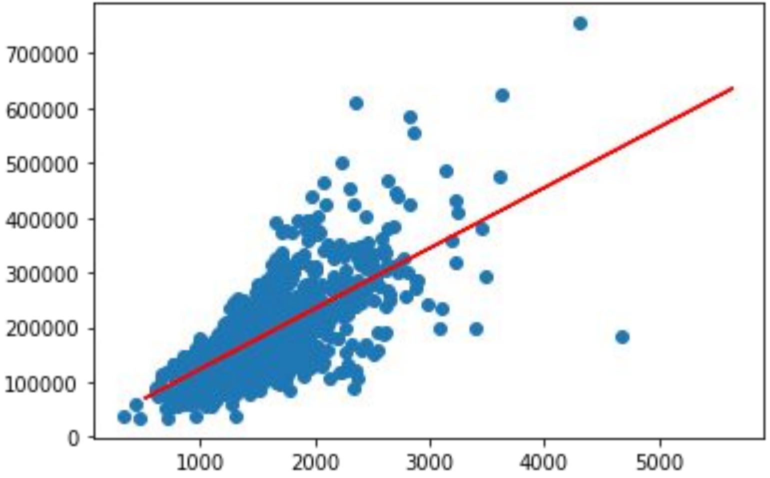
```
def get_cv_scores(model):
    scores = cross_val_score(model, X_train, y_train, cv=10, scoring='r2')

    print('CV Mean: ', np.mean(scores))
    print('STD: ', np.std(scores))
    print('\n')

lr = LinearRegression().fit(X_train, y_train)
get_cv_scores(lr)
```

View the Coefficients

```
print(lr.intercept_)

print(lr.coef_)
```



Predictions




```
y_pred = lr.predict(X_test)
```

```
plt.scatter(X_train, y_train)

plt.plot(X_test, y_pred, color='red')

plt.show()
```

Predict with NEW unseen Data

```
lr.predict([[2515]])
```

Evaluate Predicted Values from Actuals

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

df.head()
```


```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head()
```


	Actual	Predicted
0	200624	290645.119259
1	133000	187327.428687
2	110000	145978.299590
3	192000	236284.797539
4	88000	133738.957377

R-Squared (R2)

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

df.head()


print('R-Squared:',metrics.r2_score(df['Actual'],df['Predicted']))
------
```

Summary Definition

Define R-Squared: Coefficient of determination means a statistical measurement of the correlation between two variables.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

Evaluate Your Model

```
print('Mean Absolute Error:',
metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:',
metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 39364.76724953735
Mean Squared Error: 3913788296.4027987
Root Mean Squared Error: 62560.277304394986
```

https://scikit-learn.org/

https://github.com/scikit-learn/scikit-learn

Tuning Hyperparameters with Simple Linear Regression

https://bit.ly/ucidata

Hint: To create a new dataframe with selected columns - do this:

```
df2 = df[["open", "close"]] #open and close are the colomns
```

Lab 1

Read in this DATA:

```
pd.read_csv('https://raw.githubusercontent.com/fenago/pythonml/main/data/poverty.txt',sep="\t")
```

Apply what you have learned to create Simple Linear Models.

This dataset of size $n = 51$ are for the 50 states and the District of Columbia in the United States (poverty.txt). The variables are y = year 2002 birth rate per 1000 females 15 to 17 years old and x = poverty rate, which is the percent of the state's population living in households with incomes below the federally defined poverty level. (Data source: *Mind On Statistics*, 3rd edition, Utts and Heckard).

Lab 2

Read in this DATA:

```
pd.read_csv('https://raw.githubusercontent.com/fenago/pythonml/main/data/lungfunction.txt',sep="\t")
```

Apply what you have learned to create Simple Linear Models.

This dataset of size $n = 51$ are for the 50 states and the District of Columbia in the United States (poverty.txt). The variables are y = year 2002 birth rate per 1000 females 15 to 17 years old and x = poverty rate, which is the percent of the state's population living in households with incomes below the federally defined poverty level. (Data source: *Mind On Statistics*, 3rd edition, Utts and Heckard).

Multivariate Linear Regression

Our Dataset

Data Dictionary:

http://people.sc.fsu.edu/~jburkardt/datasets/regression/x16.txt

Dataset:

```
pd.read_csv("https://raw.githubusercontent.com/fenago/pythonml/main/data/petrol_consumption.csv")
```

Load the Libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline
```

Load the Data

```
df =
pd.read_csv("https://raw.githubusercontent.com/fenago/pythonml/main/data/petrol_consumption.csv")

df.head()

df.describe()
```

Find Your Correlations In your Datasets

```
# correlation between 2 Specific Columns
print(df['Petrol_tax'].corr(df['Petrol_Consumption']))



# pair-wise correlation between all columns

print(df.corr())
```

Prepare the Data

```
X = dataset[['Petrol_tax', 'Average_income',
'Paved_Highways','Population_Driver_licence(%)']]

y = dataset['Petrol_Consumption']


#Execute below to divide into train/test sets

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)
```

Train the Algorithm as Before

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_train, y_train)
```

What Coefficients Did it Find?

```
coeff_df = pd.DataFrame(regressor.coef_, X.columns,
columns=['Coefficient'])
```

```
coeff_df
```

	Coefficient
Petrol_tax	-24.196784
Average_income	-0.81680
Paved_Highways	-0.000522
Population_Driver_license(%)	1324.675464

Predictions

```
y_pred = regressor.predict(X_test)

df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

df
```

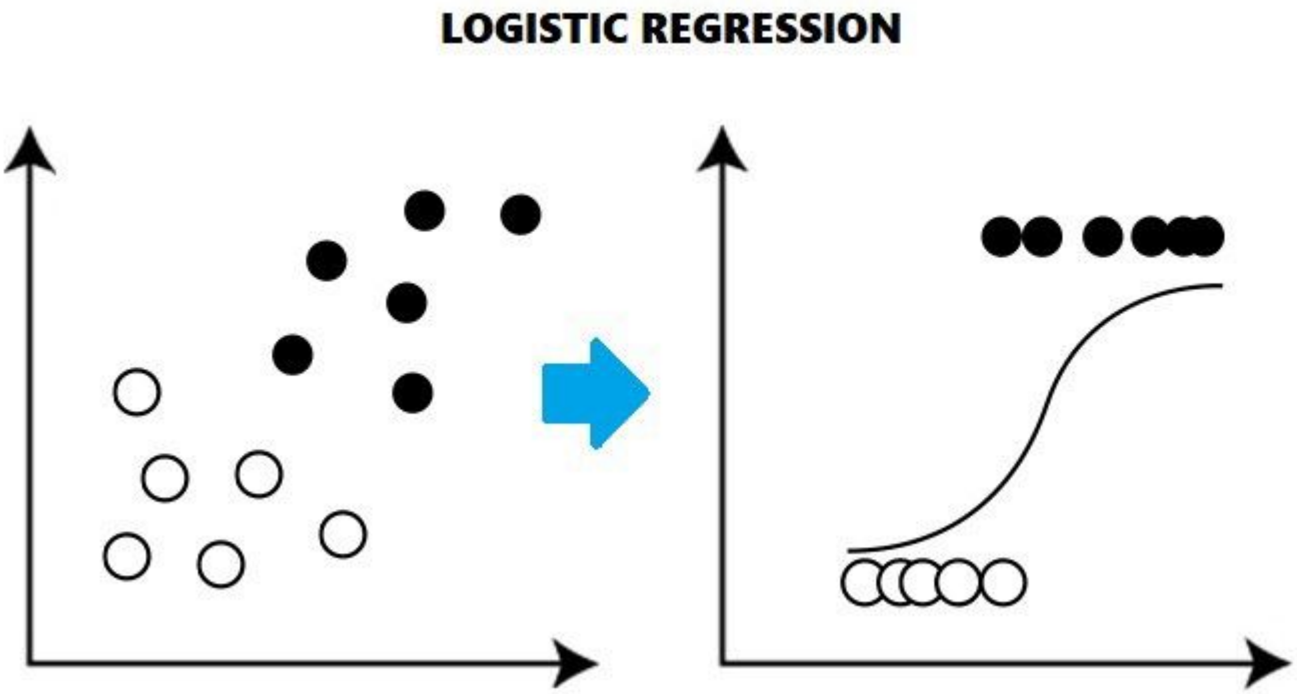
	Actual	Predicted
36	640	643.176639
22	464	411.950913
20	649	683.712762
38	648	728.049522
18	865	755.473801

Evaluate the Algorithm

```
from sklearn import metrics

print('Mean Absolute Error:', 
metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', 
metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', 
np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print('R-Squared:',metrics.r2_score(df['Actual'], 
df['Predicted']))
```

Logistic Regression

Logistic Regression

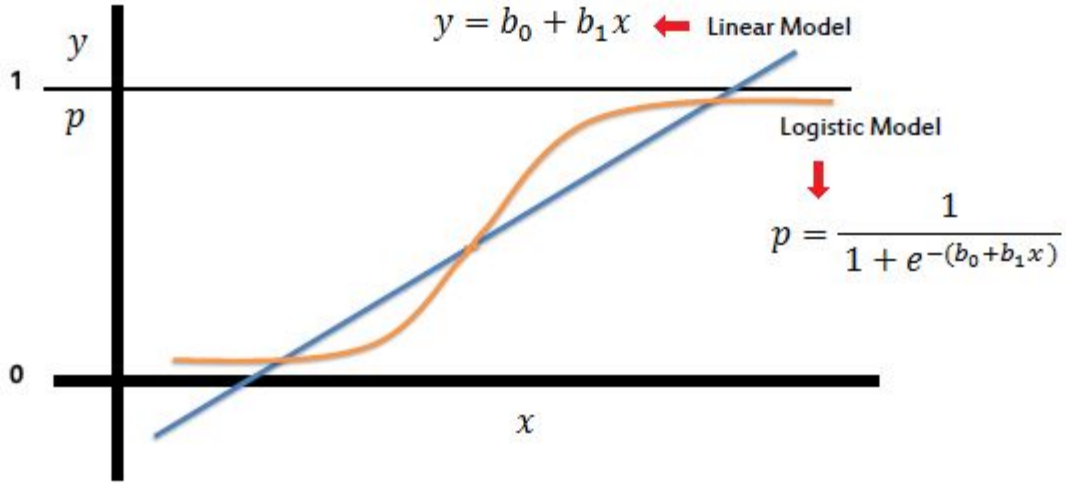


Logistic Regression

when b0+b1X == 0, then the p will be 0.5,

similarly,b0+b1X > 0, then the p will be going towards 1 and

b0+b1X < 0, then the p will be going towards 0.

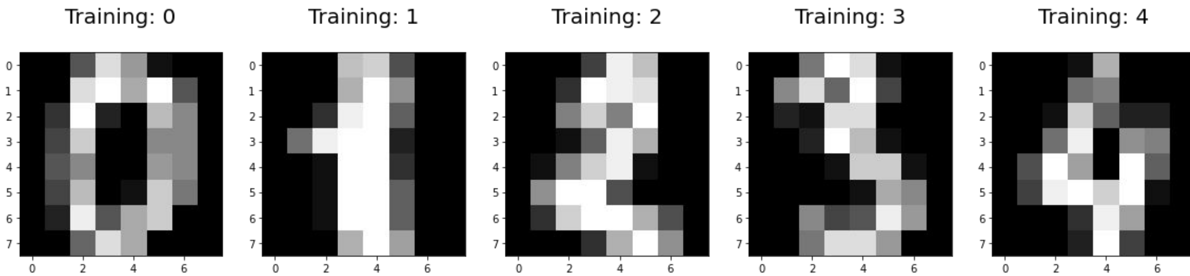


Loading the Data

```
from sklearn.datasets import load_digits

digits = load_digits()
```

Showing the Images and Labels

```
import numpy as np

import matplotlib.pyplot as plt


plt.figure(figsize=(20,4))


for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[0:5])):

  plt.subplot(1, 5, index + 1)

  plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)

  plt.title('Training: %i\n' % label, fontsize = 20)
```



Split the Data into Training and Test Sets

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test =
train_test_split(digits.data, digits.target, test_size=0.25,
random_state=0)
```

Modeling

```
from sklearn.linear_model import LogisticRegression

logisticRegr = LogisticRegression()

logisticRegr.fit(x_train, y_train)
```

```
#predict for one image
logisticRegr.predict(x_test[0].reshape(1,-1))

#predict for multiple images
logisticRegr.predict(x_test[0:10])

#for the entire dataset
predictions = logisticRegr.predict(x_test)
```

```
[14] #predict for one image
     logisticRegr.predict(x_test[0].reshape(1,-1))
```

```
array([2])
```

```
[15] #predict for multiple images
     logisticRegr.predict(x_test[0:10])
```

```
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5])
```

```
[16] #for the entire dataset
     predictions = logisticRegr.predict(x_test)
```

```
[17] predictions
```

```
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, 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])
```

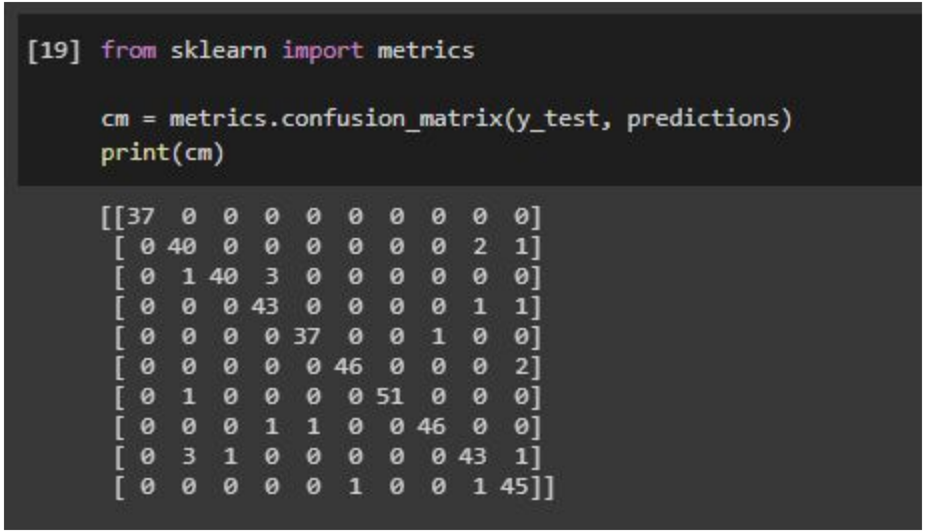
Evaluation

```
#Accuracy = correct predictions / total number of data points

#Use score method to get accuracy of model

score = logisticRegr.score(x_test, y_test)

print(score)
```

```
[18] #Use score method to get accuracy of model
     score = logisticRegr.score(x_test, y_test)
     print(score)

     0.9511111111111111
```

Confusion Matrix

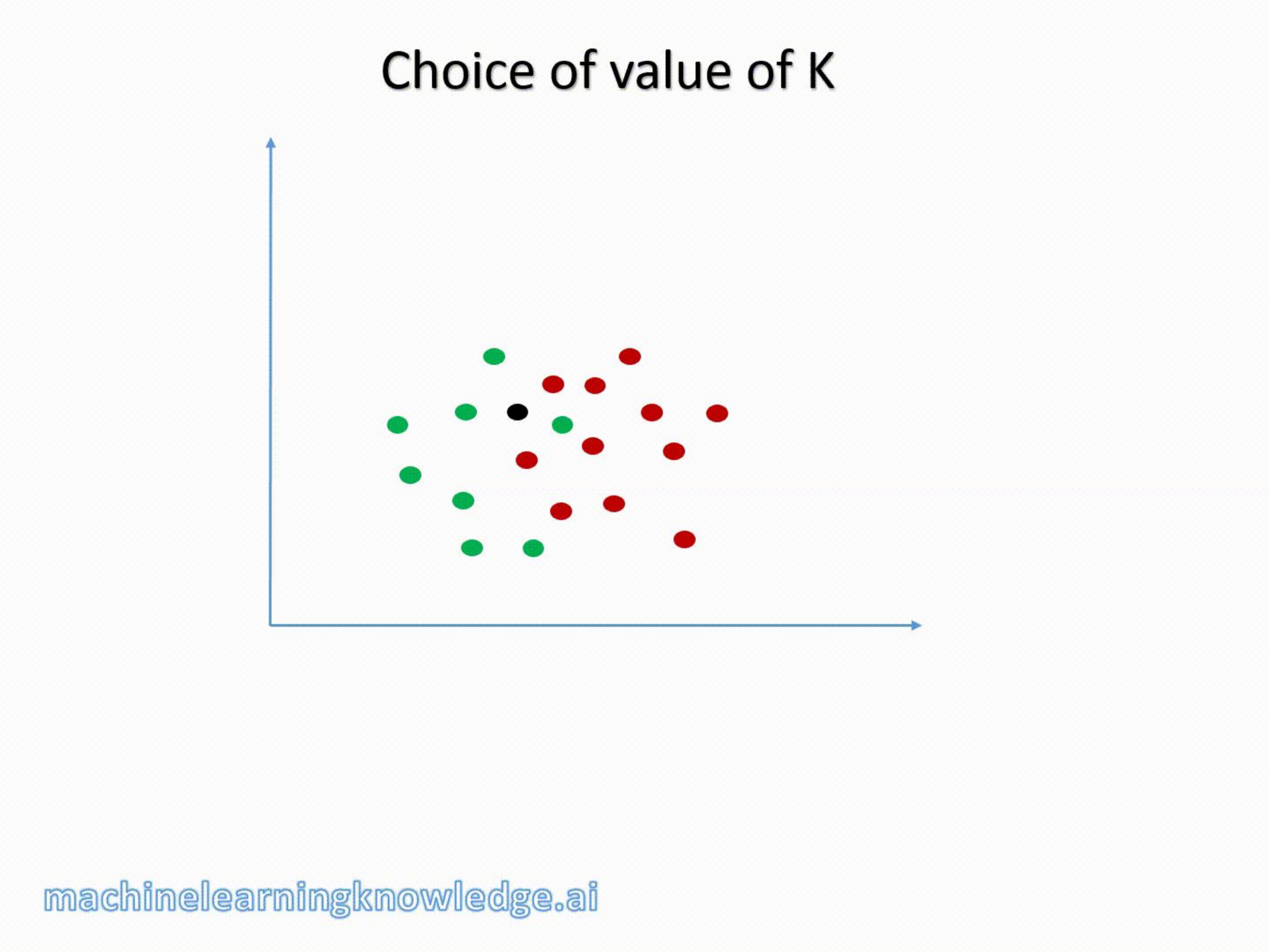
```
from sklearn import metrics

cm = metrics.confusion_matrix(y_test, predictions)

print(cm)
```



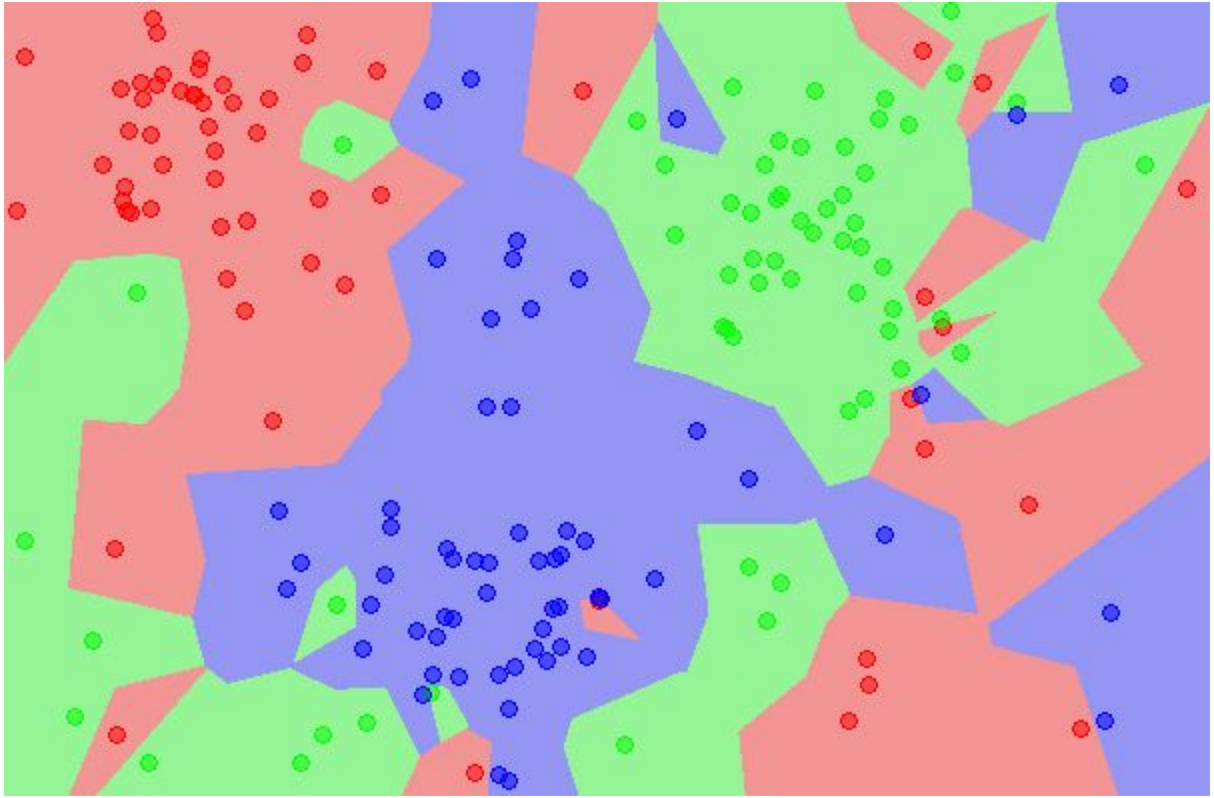
```
[19] from sklearn import metrics

     cm = metrics.confusion_matrix(y_test, predictions)
     print(cm)

     [[37  0  0  0  0  0  0  0  0  0]
      [ 0 40  0  0  0  0  0  0  2  1]
      [ 0  1 40  3  0  0  0  0  0  0]
      [ 0  0  0 43  0  0  0  0  1  1]
      [ 0  0  0  0 37  0  0  1  0  0]
      [ 0  0  0  0  0 46  0  0  0  2]
      [ 0  1  0  0  0  0 51  0  0  0]
      [ 0  0  0  1  1  0  0 46  0  0]
      [ 0  3  1  0  0  0  0  0 43  1]
      [ 0  0  0  0  0  1  0  0  1 45]]
```

KNN

KNN



How Does KNN Work?

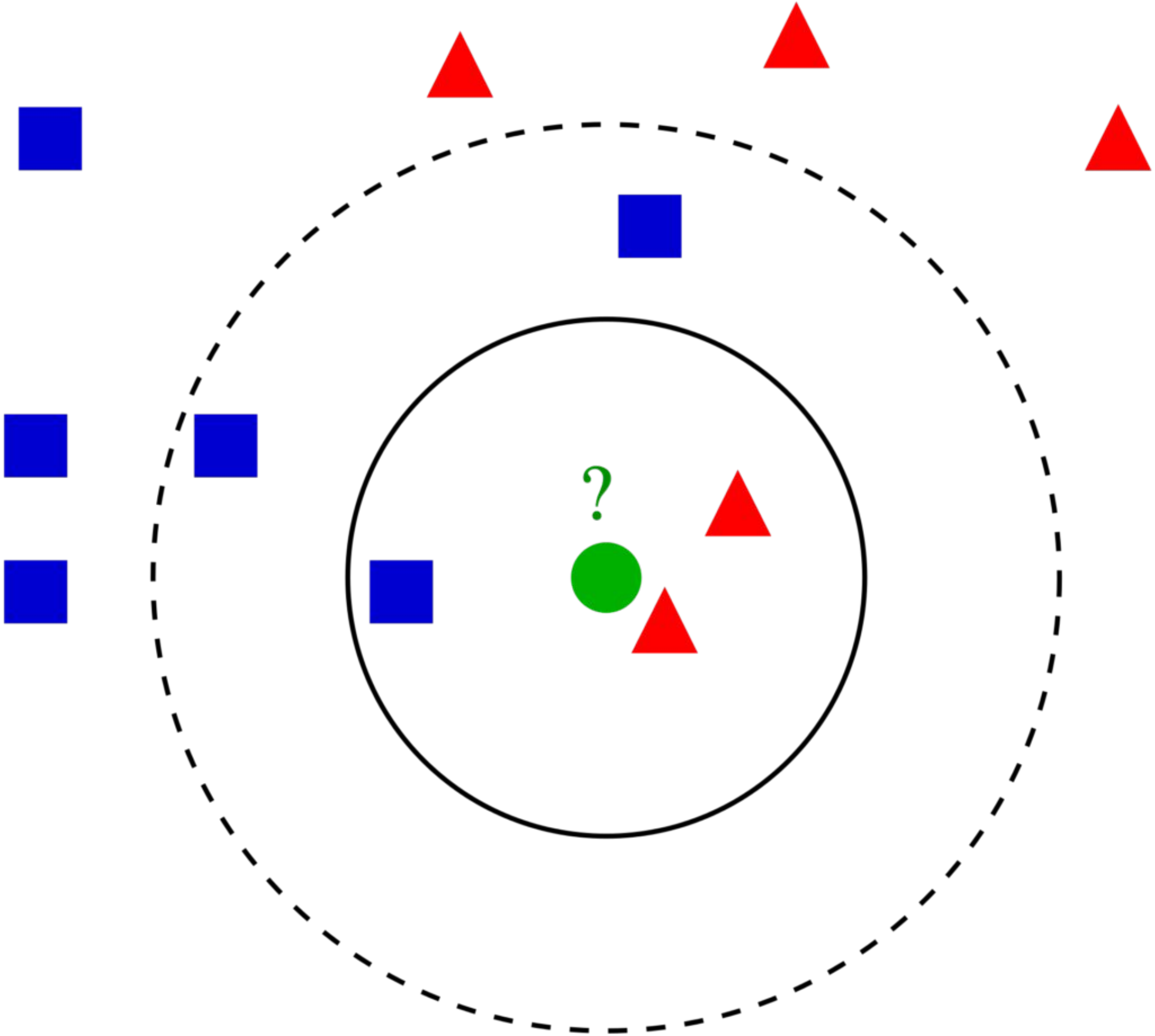


Distance in KNN

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Nearest?



Steps to Solve a KNN Problem

1. Load and store the data.
2. Calculate the distance from x (new data point) to all other data points.
3. Sort all the distances from your data in ascending order.
4. Initialize the K value for the nearest data points.
5. Make a prediction based on the majority of data points with the same label within the K value.
6. Evaluate your machine learning model.

The Use Case…

we want to create a machine model that will allow botanists to classify different species of iris flowers.

Let's Build The Model

Imports

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib
```

Data Preparation

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

print("Keys of iris_dataset:\n", iris_dataset.keys())
```

OUTPUT

```
Keys of iris_dataset:
 dict_keys(['data', 'target', 'target_names', 'DESCR',
'feature_names', 'filename'])
```

Data Type

```
print("Data Type:", type(iris_dataset['data']))
```

OUTPUT:

```
Data Type: <class 'numpy.ndarray'>
```

Shape

```
print("Shape of Data:", iris_dataset['data'].shape)
```

Features

```
print("First 10 Samples and Their Features:\n",
iris_dataset['data'][:10])
```

Target

```
print("Type of Target:", type(iris_dataset['target']))

print("Shape of Target:", iris_dataset['target'].shape)

print(iris_dataset['target'])
```

Target Names

```
print("Target names:", iris_dataset['target_names'])
```

Description

```
print(iris_dataset['DESCR'])
```

OR

```
print(iris_dataset['DESCR'][:500] + "\n...")
```

Feature Names

```
print("Feature Names:", iris_dataset['feature_names'])
```

Training and Testing Data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], 
random_state=0)
```

Validate the Shape of the Data and Test the Dataset

```
print("X_train Shape:", X_train.shape)

print("y_train Shape:", y_train.shape)


print("X_test Shape:", X_test.shape)

print("y_test Shape:", y_test.shape)
```

Visualize Your Data

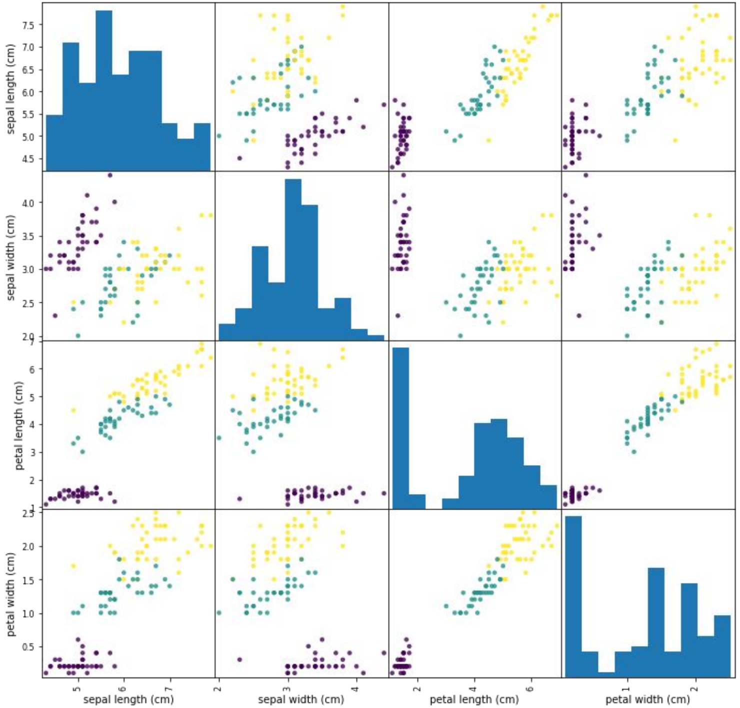
Create the DF for Visualizing

```
df = pd.DataFrame(X_train,
columns=iris_dataset.feature_names)

df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.9	3.0	4.2	1.5
1	5.8	2.6	4.0	1.2
2	6.8	3.0	5.5	2.1
3	4.7	3.2	1.3	0.2
4	6.9	3.1	5.1	2.3

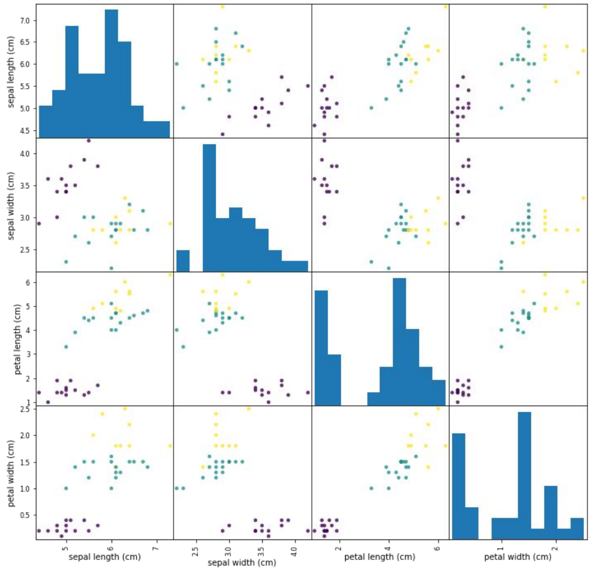
Show the Visualization

```
pd.plotting.scatter_matrix(df,c=y_train,figsize=(12,12),
marker='o',s=20,alpha=.8)

plt.show()
```



Do the same with X

```
df = pd.DataFrame(X_test,
columns=iris_dataset.feature_names)

pd.plotting.scatter_matrix(df,c=y_test,figsize=(12,12),
marker='o',s=20,alpha=.8)

plt.show()
```



Create the Model

```
from sklearn.neighbors import KNeighborsClassifier

knnObject = KNeighborsClassifier(n_neighbors=1)

knnObject.fit(X_train, y_train)
```

Making Predictions

Predictions on New Data

Imagine you are a machine learning engineer for a company. A client of yours reached out to you to verify an iris species they found in the wild. They only gave us the following information:

- Sepal length: 40 cm
- Sepal width: 10 cm
- Petal length: 5 cm
- Petal width: 2 cm

```
newIris = np.array([[40, 10, 5, 2]])

print("newIris Shape:", newIris.shape)
```

Predict

```
prediction = knnObject.predict(newIris)
print("Prediction Value:", prediction)
print("Predicted Target Name:",
       iris_dataset['target_names'][prediction])
```

Model Evaluation

See Your Test Predictions

```
testSetPredictions = knnObject.predict(X_test)

print("Test Set Predictions:", testSetPredictions)
```

Check for Accuracy

```
accuaracy = round(knnObject.score(X_test, y_test),2)

print("The Test Set Accuracy is:",accuracy)
```

Breast Cancer Use Case - KNN

Initialize the Libraries

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

Import the Data Set

It is in the data folder and named dataR2.csv

```
data=pd.read_csv("./data/dataR2.csv")
```

Understand Your Data

`data.shape`

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	Classification
0	48	23.500000	70	2.707	0.467409	8.8071	9.702400	7.99585	417.114	1
1	83	20.690495	92	3.115	0.706897	8.8438	5.429285	4.06405	468.786	1
2	82	23.124670	91	4.498	1.009651	17.9393	22.432040	9.27715	554.697	1
3	68	21.367521	77	3.226	0.612725	9.8827	7.169560	12.76600	928.220	1
4	86	21.111111	92	3.549	0.805386	6.6994	4.819240	10.57635	773.920	1
...	...	...	...	...	...	...	...	...	...	...
111	45	26.850000	92	3.330	0.755688	54.6800	12.100000	10.96000	268.230	2
112	62	26.840000	100	4.530	1.117400	12.4500	21.420000	7.32000	330.160	2
113	65	32.050000	97	5.730	1.370998	61.4800	22.540000	10.33000	314.050	2
114	72	25.590000	82	2.820	0.570392	24.9600	33.750000	3.27000	392.460	2
115	86	27.180000	138	19.910	6.777364	90.2800	14.110000	4.35000	90.090	2

116 rows × 10 columns

Find Missing Values

```
data.isna().sum()
```

```
Age               0
BMI               0
Glucose           0
Insulin           0
HOMA              0
Leptin            0
Adiponectin       0
Resistin          0
MCP.1             0
Classification    0
dtype: int64
```

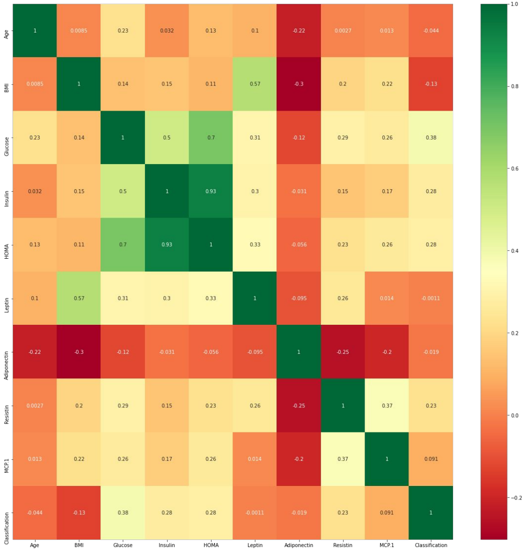
Exploratory Data Analysis

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116 entries, 0 to 115
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             116 non-null    int64
 1   BMI             116 non-null    float64
 2   Glucose         116 non-null    int64
 3   Insulin         116 non-null    float64
 4   HOMA            116 non-null    float64
 5   Leptin          116 non-null    float64
 6   Adiponectin     116 non-null    float64
 7   Resistin        116 non-null    float64
 8   MCP.1           116 non-null    float64
 9   Classification  116 non-null    int64
dtypes: float64(7), int64(3)
memory usage: 9.2 KB
```

Heatmaps and Correlation

```
#Heatmap to find correlation
plt.subplots(figsize=(20,20))
sns.heatmap(data.corr(),cmap='RdYlGn',annot=True)
```



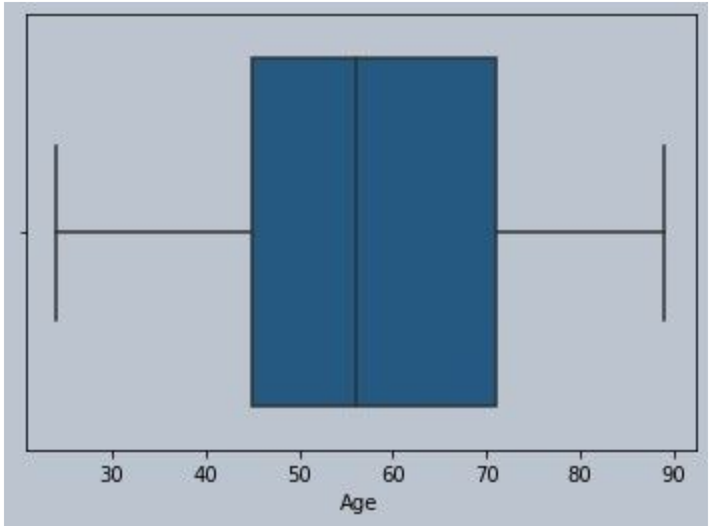
Columns

```
data.columns
```

KNN is Sensitive to Outliers

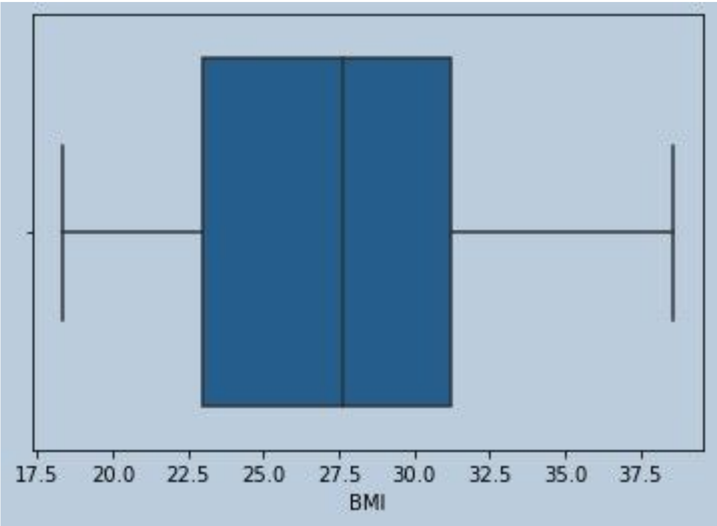
Outliers for Age?

```
#No outliers for age

 sns.boxplot(data['Age'])
```



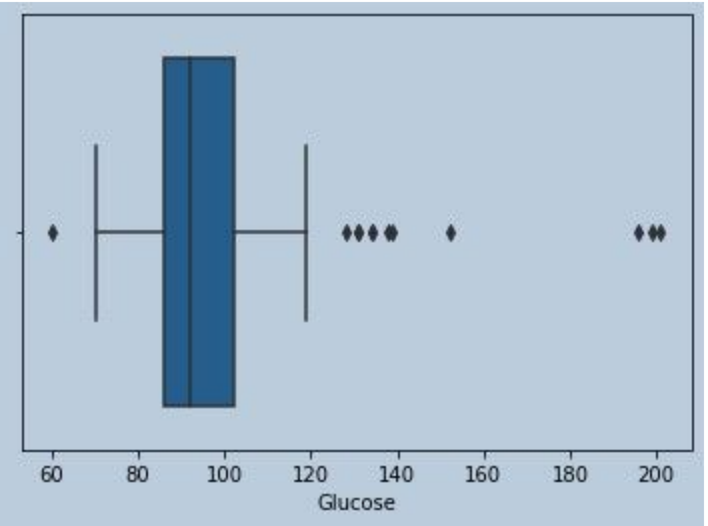
Outliers for BMI?

```
#NO outliers for BMI

sns.boxplot(data['BMI'])
```



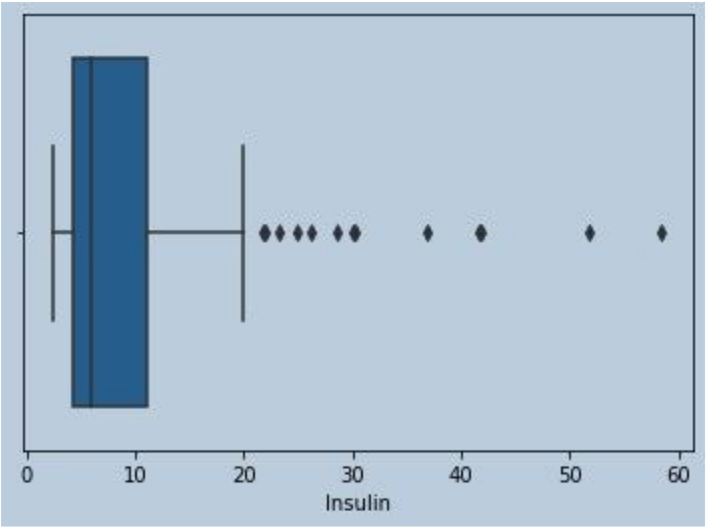
Outliers for Glucose

```
#Some outliers are there for Glucose and  data is Skewed

sns.boxplot(data['Glucose'])
```



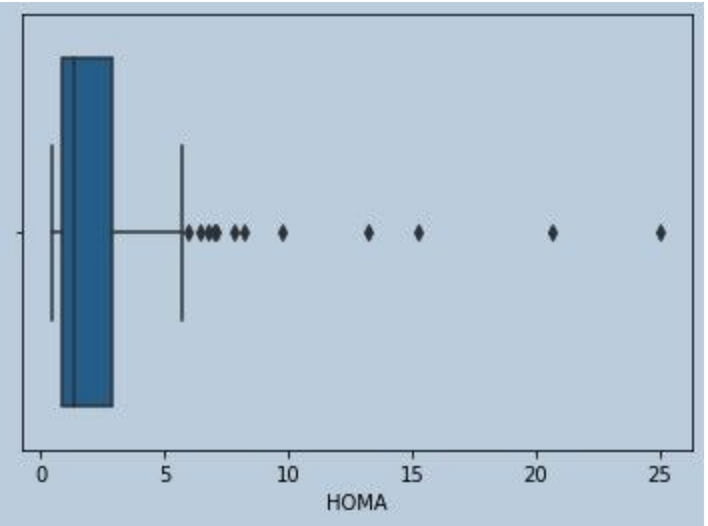
Outliers for Insulin?

```
#Outliers are present in Insulin

sns.boxplot(data['Insulin'])
```



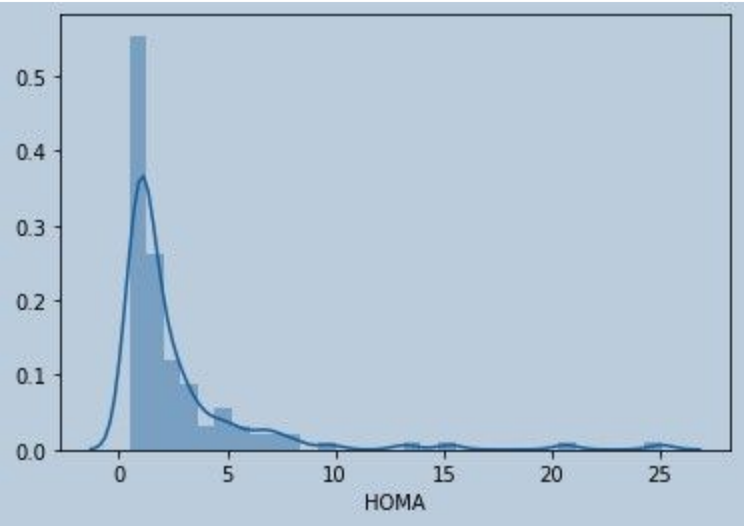
Outliers for Homa

```
#lots of Outliers in Homa

sns.boxplot(data['HOMA'])
```



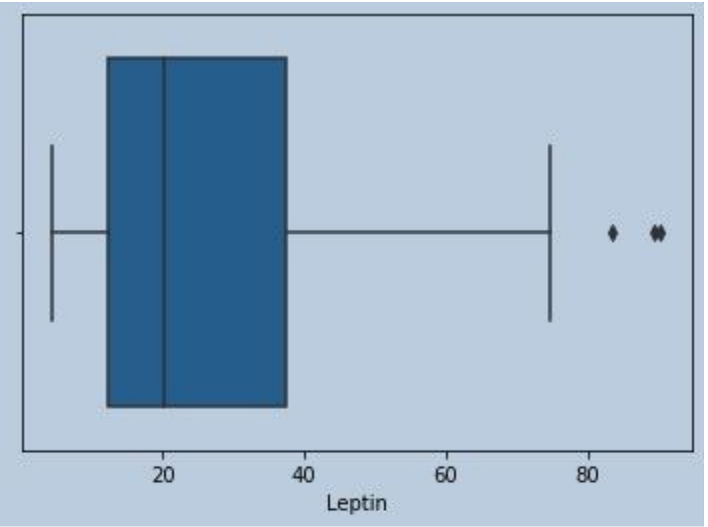
Distribution of HOMA

```
#Distribution plot of HOMA

sns.distplot(data['HOMA'])
```

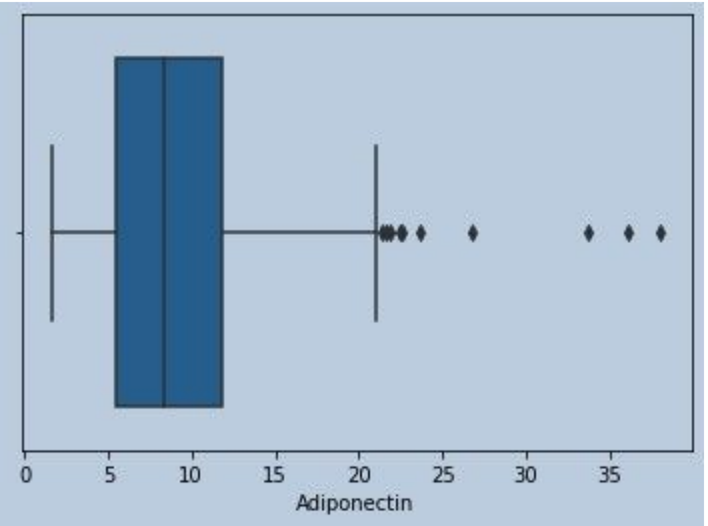


Outliers for Leptin

```
#Outliers present for Leptin

sns.boxplot(data['Leptin'])
```

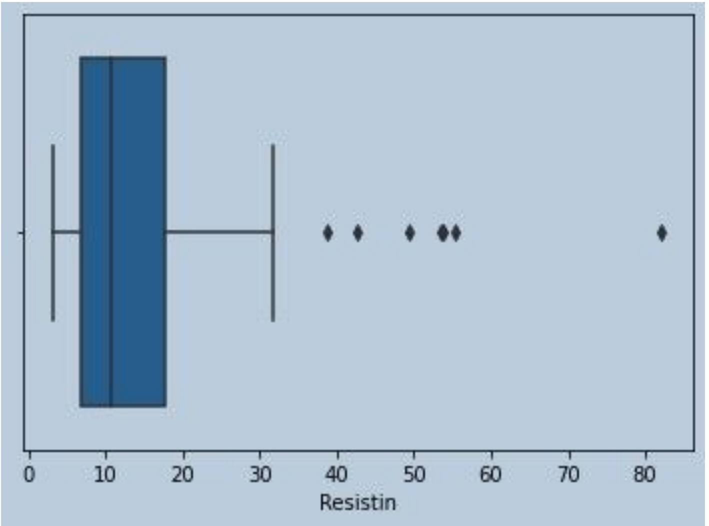


```
#Outliers present for Adiponectin

sns.boxplot(data['Adiponectin'])
```



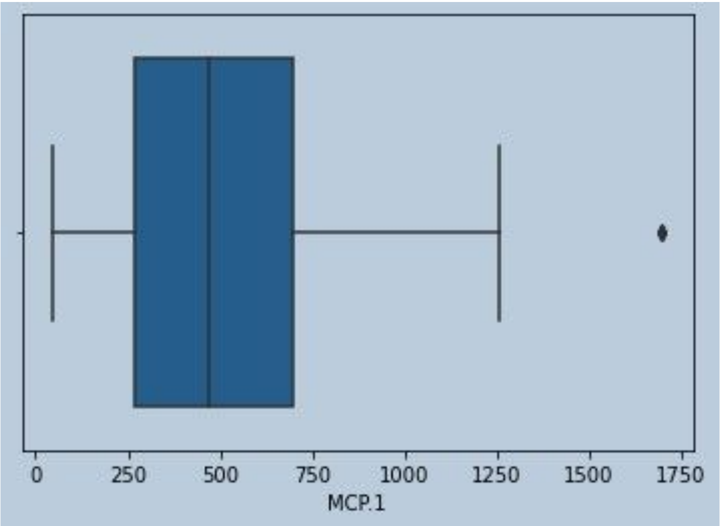
Outliers for Resistin

```
#Ouliers present for Resistin

sns.boxplot(data['Resistin'])
```



Outliers for MCP.1?

```
#Outliers present for MCP.1

sns.boxplot(data['MCP.1'])
```



Remove Outliers

```
#Removing Outliers Since they may affect prediction for KNN (quantile method)

    cancer=data.copy()

    insulinQ1=cancer['Insulin'].quantile(0.25)

    insulinQ3=cancer['Insulin'].quantile(0.75)

    insulinIQR=insulinQ3-insulinQ1

    lowerliminsulin=insulinQ1-1.5*insulinIQR

    upperliminsulin=insulinQ3+1.5*insulinIQR

    insulrem=cancer[(cancer['Insulin']>lowerliminsulin)&(upperliminsulin > 
cancer['Insulin'])]
```

```
sns.boxplot(insulrem['Glucose'])

glucoseQ1=insulrem['Glucose'].quantile(0.25)

glucoseQ3=insulrem['Glucose'].quantile(0.75)

glucoseIQR=glucoseQ3-glucoseQ1

upperlimglucose=glucoseQ3+1.5*glucoseIQR

lowerlimglucose=glucoseQ1-1.5*glucoseIQR

glucoserem=insulrem[(insulrem['Glucose'] >
lowerlimglucose)&(upperlimglucose > insulrem['Glucose'])]
```

```
sns.boxplot(glucoserem['HOMA'])

homaQ1=glucoserem['HOMA'].quantile(0.25)

homaQ3=glucoserem['HOMA'].quantile(0.75)

homaIQR=homaQ3-homaQ1

upperlimhoma=homaQ3+1.5*homaIQR

lowerlimhoma=homaQ1-1.5*homaIQR

homarem=glucoserem[(glucoserem['HOMA'] >
lowerlimhoma)&(upperlimhoma > glucoserem['HOMA'])]
```

```
sns.boxplot(homarem['Adiponectin'])

AdiponectinQ1=homarem['Adiponectin'].quantile(0.25)

AdiponectinQ3=homarem['Adiponectin'].quantile(0.75)

AdiponectinIQR=AdiponectinQ3-AdiponectinQ1

upperlimAdiponectin=AdiponectinQ3+1.5*AdiponectinIQR

lowerlimAdiponectin=AdiponectinQ1-1.5*AdiponectinIQR

adirem=homarem[(homarem['Adiponectin'] >
lowerlimAdiponectin)&(upperlimAdiponectin >
homarem['Adiponectin'])]
```

```
sns.boxplot(adirem['Resistin'])
```

```
sns.boxplot(adirem['Leptin'])
```

```
sns.boxplot(adirem['MCP.1'])
```

```
# create the features from data
X=mcprem.iloc[:,0:9]

# create the target variable from data
Y=mcprem.iloc[:,9]
```

Standardize

Standardize to Bring All to the Same Scale

```
from sklearn.preprocessing import StandardScaler

ss=StandardScaler()

X=ss.fit_transform(X)

X=pd.DataFrame(X)
```

Split Data

```
from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.3)
```

Build the Classifier

Build the Model

```
#Finding accuracies  on TrainData and Test data with euclidean distance(by default p=2)

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score

for x in range(5,10,2):

        knn=KNeighborsClassifier(n_neighbors=x,metric='minkowski',weights='distance')

        knn.fit(xtrain,ytrain)

        train_ypred=knn.predict(xtrain)

        acc_train_score=accuracy_score(train_ypred,ytrain)

        test_ypred=knn.predict(xtest)

        acc_test_score=accuracy_score(test_ypred,ytest)

        print(f'Accuracy score for  train data and test data is {acc_train_score} and {acc_test_score} respectively for {x} neighbours')
```

Build the Model with Euclidean Distance

```
knn=KNeighborsClassifier(n_neighbors=7,metric='minkowski',weights='distance')
knn.fit(xtrain,ytrain)
trainypred=knn.predict(xtrain)
```

Run a Metric Report

```
from sklearn.metrics import classification_report

print(classification_report(trainypred,ytrain))

accuracy_score(trainypred,ytrain)

testypredicted=knn.predict(xtest)

from sklearn.metrics import accuracy_score

accuracy_score(testypredicted,ytest)
```

```
              precision    recall  f1-score   support

           1       0.92      0.71      0.80        31
           2       0.64      0.89      0.74        18

    accuracy                           0.78        49
   macro avg       0.78      0.80      0.77        49
weighted avg       0.82      0.78      0.78        49
```

Pickles

```
import pickle

#Save our model as a pickle to a file

pickle.dump(knn, open("my_knn_model.pickle.dat", "wb"))

# delete the existing knn model from the environment

del knn

#Load the pickled object from the file

load_knn=pickle.load(open("my_knn_model.pickle.dat", "rb"))

# Use the loaded model to make predictions

load_knn.predict(xtest)
```

Summary