# Program 1 - FIND S

```
import csv
hypo=['%','%','%','%','%','%'];
with open('Training_examples.csv') as csv_file:
	readcsv = csv.reader(csv_file, delimiter=',')
	data = []
	print("\nThe given training examples are:")
	for row in readcsv:
		print(row)
		if row[len(row)-1].upper() == "YES":
			data.append(row)
print("\nThe positive examples are:")
for x in data:
	print(x)
TotalExamples = len(data)
print("The steps of the Find-s algorithm are\n",hypo)
list = []
d=len(data[0])-1
for j in range(d):
	list.append(data[0][j])
hypo=list;
for i in range(TotalExamples):
	for k in range(d):
		if hypo[k]!=data[i][k]:
			hypo[k]='?'
	print(hypo)
print("\nThe maximally specific Find-s hypothesis for the given training examples is");
print(hypo)
```

INPUT CSV

| | | | | | | |
|---|---|---|---|---|---|---|
| Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| Sunny | Warm | High | Strong | Warm | Same | Yes |
| Rainy | Cold | High | Strong | Warm | Change | No |
| Sunny | Warm | High | Strong | Cool | Change | Yes |

PROGRAM 2 – CANDIDATE PROGRAM

```
import numpy as np
import pandas as pd

data = pd.DataFrame(data=pd.read_csv('Training_examples.csv'))
print(data)
concepts = np.array(data.iloc[:,0:-1])
print("concepts",concepts)
target = np.array(data.iloc[:,-1])
print("target",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)

    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    print("ind-------",indices)
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific_h, general_h

s_final, g_final = learn(concepts, target)
```

```
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

INPUT CSV

| C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|---|---|---|---|---|---|---|
| Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| Sunny | Warm | High | Strong | Warm | Same | Yes |
| Rainy | Cold | High | Strong | Warm | Change | No |
| Sunny | Warm | High | Strong | Cool | Change | Yes |

# PROGRAM 3

```
import pandas as pd
from pandas import DataFrame
df_tennis = pd.DataFrame(data = pd.read_csv('tennis.csv'))
print("\n Given Play Tennis Data Set:\n\n",df_tennis)

def entropy(probs):
   import math
   return sum( [-prob*math.log(prob, 2) for prob in probs] )

def entropy_of_list(a_list):
   from collections import Counter
   cnt = Counter(x for x in a_list)   # Counter calculates the propotion of class
   num_instances = len(a_list)*1.0   # = 14
   probs = [x / num_instances for x in cnt.values()]  # x means no of YES/NO
   return entropy(probs) # Call Entropy :

total_entropy = entropy_of_list(df_tennis['PlayTennis'])
print("\n Total Entropy of PlayTennis Data Set:",total_entropy)

def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
   df_split = df.groupby(split_attribute_name)
   nobs = len(df.index) * 1.0
   df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs]
})#[target_attribute_name]
   df_agg_ent.columns = ['Entropy', 'PropObservations']

   # Calculate Information Gain:
   new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
   old_entropy = entropy_of_list(df[target_attribute_name])
   return old_entropy - new_entropy

print('Info-gain for Outlook is :'+str( information_gain(df_tennis, 'Outlook', 'PlayTennis')),"\n")
print('\n Info-gain for Humidity is: ' + str( information_gain(df_tennis, 'Humidity', 'PlayTennis')),"\n")
print('\n Info-gain for Wind is:' + str( information_gain(df_tennis, 'Wind', 'PlayTennis')),"\n")
print('\n Info-gain for Temperature is:' + str( information_gain(df_tennis,
'Temperature','PlayTennis')),"\n")

def id3(df, target_attribute_name, attribute_names, default_class=None):

   from collections import Counter
   cnt = Counter(x for x in df[target_attribute_name])# class of YES /NO
   if len(cnt) == 1:
      return next(iter(cnt))  # next input data set, or raises StopIteration when EOF is hit.

   elif df.empty or (not attribute_names):
      return default_class  # Return None for Empty Data Set

   else:
```

```
    default_class = max(cnt.keys()) #No of YES and NO Class
    gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names]
    index_of_max = gainz.index(max(gainz)) # Index of Best Attribute
    best_attr = attribute_names[index_of_max]
    tree = {best_attr:{}}
    remaining_attribute_names = [i for i in attribute_names if i != best_attr]

    for attr_val, data_subset in df.groupby(best_attr):
        subtree = id3(data_subset,
                      target_attribute_name,
                      remaining_attribute_names,
                      default_class)
        tree[best_attr][attr_val] = subtree
    return tree

attribute_names = list(df_tennis.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('PlayTennis')
print("Predicting Attributes:", attribute_names)
# Run Algorithm:
from pprint import pprint
tree = id3(df_tennis,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
pprint(tree)
attribute = next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree Keys:\n",tree[attribute].keys())
```

# INPUT CSV

| | PlayTennis | Outlook | Temperature | Humidity | Wind |
|---|---|---|---|---|---|
| 0 | No | Sunny | Hot | High | Weak |
| 1 | No | Sunny | Hot | High | Strong |
| 2 | Yes | Overcast | Hot | High | Weak |
| 3 | Yes | Rain | Mild | High | Weak |
| 4 | Yes | Rain | Cool | Normal | Weak |
| 5 | No | Rain | Cool | Normal | Strong |
| 6 | Yes | Overcast | Cool | Normal | Strong |
| 7 | No | Sunny | Mild | High | Weak |
| 8 | Yes | Sunny | Cool | Normal | Weak |
| 9 | Yes | Rain | Mild | Normal | Weak |
| 10 | Yes | Sunny | Mild | Normal | Strong |
| 11 | Yes | Overcast | Mild | High | Strong |
| 12 | Yes | Overcast | Hot | Normal | Weak |
| 13 | No | Rain | Mild | High | Strong |

# PROGRAM 4 – BACKPROPOGATION ALGORITHM

```
import numpy as np

id = np.array([[2,9],[1,5],[3,6]], dtype=float)
eo = np.array([[92],[86],[89]], dtype = float)
id = id/np.amax(id,axis=0)
eo = eo/100

def sigmoid(x):
    return 1/(1+np.exp(-x))

def derivative_sigmoid(x):
    return x*(1-x)

epoch = 5
lr = 0.1
il_neurons = 2
hd_neurons = 3
ol_neurons = 1

hw = np.random.uniform(size=(il_neurons,hd_neurons))
hb = np.random.uniform(size=(1,hd_neurons))
ow = np.random.uniform(size=(hd_neurons,ol_neurons))
ob = np.random.uniform(size=(1,ol_neurons))
print("Input :", id)

for i in range(epoch):
    hidden_input = np.dot(id,hw)
    hidden_input = hidden_input +hb
    hlo = sigmoid(hidden_input)

    outputlayer_input = np.dot(hlo,ow)
    outputlayer_input = outputlayer_input + ob
    olo = sigmoid(outputlayer_input)

    ol_error = eo-olo
    ol_gradient = derivative_sigmoid(olo)
    ol_error_correction =ol_error*ol_gradient

    hl_error = ol_error_correction.dot(ow.T)
    hl_gradient = derivative_sigmoid(hlo)
    hl_error_correction= hl_error*hl_gradient

    ow+= hlo.T.dot(ol_error_correction)*lr

    hw+= id.T.dot(hl_error_correction)*lr

print("Expected Output", eo)
print("Actual Output", olo)
```

PROGRAM 5 – Naïve's Bayes Classifier

```
print("\nNaive Bayes Classifier for concept learning problem")
import csv
import random
import math
import operator

def safe_div(x,y):
    if y == 0:
        return 0
    return x / y

def loadCsv(filename):
        lines = csv.reader(open(filename))
        dataset = list(lines)
        for i in range(len(dataset)):
                dataset[i] = [float(x) for x in dataset[i]]
        return dataset

def splitDataset(dataset, splitRatio):
        trainSize = int(len(dataset) * splitRatio)
        trainSet = []
        copy = list(dataset)
        i=0
        while len(trainSet) < trainSize:
                trainSet.append(copy.pop(i))
        return [trainSet, copy]

def separateByClass(dataset):
        separated = {}
        for i in range(len(dataset)):
                vector = dataset[i]
                if (vector[-1] not in separated):
                        separated[vector[-1]] = []
                separated[vector[-1]].append(vector)

        return separated

def mean(numbers):
        return safe_div(sum(numbers),float(len(numbers)))

def stdev(numbers):
        avg = mean(numbers)
        variance = safe_div(sum([pow(x-avg,2) for x in numbers]),float(len(numbers)-1))
        return math.sqrt(variance)

def summarize(dataset):
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
        del summaries[-1]
```

```
return summaries

def summarizeByClass(dataset):
	separated = separateByClass(dataset)
	summaries = {}
	for classValue, instances in separated.items():
		summaries[classValue] = summarize(instances)

	return summaries

def calculateProbability(x, mean, stdev):
	exponent = math.exp(-safe_div(math.pow(x-mean,2),(2*math.pow(stdev,2))))
	final = safe_div(1 , (math.sqrt(2*math.pi) * stdev)) * exponent
	return final

def calculateClassProbabilities(summaries, inputVector):
	probabilities = {}
	for classValue, classSummaries in summaries.items():
		probabilities[classValue] = 1
		for i in range(len(classSummaries)):
			mean, stdev = classSummaries[i]
			x = inputVector[i]
			probabilities[classValue] *= calculateProbability(x, mean, stdev)
	return probabilities

def predict(summaries, inputVector):
	probabilities = calculateClassProbabilities(summaries, inputVector)
	bestLabel, bestProb = None, -1
	for classValue, probability in probabilities.items():
		if bestLabel is None or probability > bestProb:
			bestProb = probability
			bestLabel = classValue
	return bestLabel

def getPredictions(summaries, testSet):
	predictions = []
	for i in range(len(testSet)):
		result = predict(summaries, testSet[i])
		predictions.append(result)
	return predictions

def getAccuracy(testSet, predictions):
	correct = 0
	for i in range(len(testSet)):
		if testSet[i][-1] == predictions[i]:
			correct += 1
	accuracy = safe_div(correct,float(len(testSet))) * 100.0
	return accuracy

def main():
	filename = 'ConceptLearning.csv'
```

```
	splitRatio = 0.75
	dataset = loadCsv(filename)
	trainingSet, testSet = splitDataset(dataset, splitRatio)
	print('Split {0} rows into'.format(len(dataset)))
	print('Number of Training data: ' + (repr(len(trainingSet))))
	print('Number of Test Data: ' + (repr(len(testSet))))
	print("\nThe values assumed for the concept learning attributes are\n")
	print("OUTLOOK=> Sunny=1 Overcast=2 Rain=3\nTEMPERATURE=> Hot=1 Mild=2 
Cool=3\nHUMIDITY=> High=1 Normal=2\nWIND=> Weak=1 Strong=2")
	print("TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5")
	print("\nThe Training set are:")
	for x in trainingSet:
		print(x)
	print("\nThe Test data set are:")
	for x in testSet:
		print(x)
	print("\n")

	# prepare model
	summaries = summarizeByClass(trainingSet)

	# test model
	predictions = getPredictions(summaries, testSet)
	actual = []
	for i in range(len(testSet)):
		vector = testSet[i]
		actual.append(vector[-1])
	# Since there are five attribute values, each attribute constitutes to 20% accuracy. So if all 
attributes match with predictions then 100% accuracy
	print('Actual values: {0}%'.format(actual))
	print('Predictions: {0}%'.format(predictions))
	accuracy = getAccuracy(testSet, predictions)
	print('Accuracy: {0}%'.format(accuracy))

main()
```

INPUT CSV

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 5 |
| 1 | 1 | 1 | 2 | 5 |
| 2 | 1 | 1 | 2 | 10 |
| 3 | 2 | 1 | 1 | 10 |
| 3 | 3 | 2 | 1 | 10 |
| 3 | 3 | 2 | 2 | 5 |
| 2 | 3 | 2 | 2 | 10 |
| 1 | 2 | 1 | 1 | 5 |
| 1 | 3 | 2 | 1 | 10 |
| 3 | 2 | 2 | 2 | 10 |
| 1 | 2 | 2 | 2 | 10 |
| 2 | 2 | 1 | 2 | 10 |
| 2 | 1 | 2 | 1 | 10 |
| 3 | 2 | 1 | 2 | 5 |
| 1 | 2 | 1 | 2 | 10 |
| 1 | 2 | 1 | 2 | 5 |

# PROGRAM 6 – Naives Bayesian Classifier

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import numpy as np

categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups(subset='train',categories=categories,shuffle=True)
twenty_test = fetch_20newsgroups(subset='test',categories=categories,shuffle=True)
print(len(twenty_train.data))
print(len(twenty_test.data))
print(twenty_train.target_names)

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_tf = count_vect.fit_transform(twenty_train.data)
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_tf)

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
mod = MultinomialNB()
mod.fit(X_train_tfidf, twenty_train.target)
X_test_tf = count_vect.transform(twenty_test.data)
X_test_tfidf = tfidf_transformer.transform(X_test_tf)
predicted = mod.predict(X_test_tfidf)

print("Accuracy:", accuracy_score(twenty_test.target, predicted))
print(classification_report(twenty_test.target,predicted,target_names=twenty_test.target_names))
print("confusion matrix is \n",metrics.confusion_matrix(twenty_test.target, predicted))
```

# PROGRAM 7 – Bayesian network

```
import numpy as np
from urllib.request import urlopen
import urllib

import sklearn as skl
import pandas as pd
import pgmpy

Cleveland_data_URL = 'http://archive.ics.uci.edu/ml/machine-learning-databases/heart-
disease/processed.hungarian.data'
#names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca',
'thal', 'heartdisease']
names = ['1','2','3','4','5','6','7','8','9','10','11','12','13','heartdisease']
heartDisease = pd.read_csv(urlopen(Cleveland_data_URL), names = names) #gets Cleveland
data

from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator
#model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),
#                       ('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),
#                      ('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])

# Learing CPDs using Maximum Likelihood Estimators
model =
BayesianModel([('1','4'),('1','6'),('2','4'),('2','4'),('9','4'),('4','heartdisease'),('6','heartdisease'),('heartd
isease','7'),('heartdisease','8'),('heartdisease','5')])
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

print(model.get_cpds('1'))
print(model.get_cpds('5'))
print(model.get_cpds('2'))

model.get_independencies()
# Doing exact inference using Variable Elimination
from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'1': 22})
print(q['heartdisease'])
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'5': 128})
print(q['heartdisease'])
```

## PROGRAM 8 –EM & k-Means

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

#%matplotlib inline

l1 = [0,1,2]

def rename(s):
	l2 = []
	for i in s:
		if i not in l2:
			l2.append(i)

	for i in range(len(s)):
		pos = l2.index(s[i])
		s[i] = l1[pos]

	return s

iris = datasets.load_iris()
print("\n IRIS TARGET NAMES:\n",iris.target_names)
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

print("Actual Target is:\n", iris.target)

model = KMeans(n_clusters=3)
model.fit(X)		# computes k means clustering,  model.labels_=returns clustered array

plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
```

```
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.show()

km = rename(model.labels_)
print("\nWhat KMeans thought: \n", km)
print("Accuracy of KMeans is ",sm.accuracy_score(y, km))
print("Confusion Matrix for KMeans is \n",sm.confusion_matrix(y, km))

from sklearn import preprocessing
scaler = preprocessing.StandardScaler()#Standardize features by removing the mean and scaling to unit
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
print("\n",xs.sample(5))

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_cluster_gmm = gmm.predict(xs)

plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_cluster_gmm], s=40)
plt.title('GMM Classification')
plt.show()

em = rename(y_cluster_gmm)
print("\nWhat EM thought: \n", em)
print("Accuracy of EM is ",sm.accuracy_score(y, em))
print("Confusion Matrix for EM is \n", sm.confusion_matrix(y, em))
```

## Program 8 (version2)

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

df = load_iris()
X = df["data"]
Y = df["target"]

from sklearn.cluster import KMeans
km_cluster = KMeans(n_clusters = 3)
km_cluster.fit(X)
km_predictions = km_cluster.predict(X)
print("KM Thought")
print(km_predictions)
plt.scatter(X[:, 0], X[:, 1], c = km_predictions) # Sepal length vs Sepal width (in cm)
plt.show()

from sklearn.mixture import GaussianMixture
em_cluster = GaussianMixture(n_components = 3)
em_cluster.fit(X)
em_predictions = em_cluster.predict(X)
print("EM Thought")
print(em_predictions)
plt.scatter(X[:, 0], X[:, 1], c = em_predictions)
plt.show()

#Comparing their accuracies
from sklearn.metrics import accuracy_score, confusion_matrix
km_accuracy = accuracy_score(Y, km_predictions)
em_accuracy = accuracy_score(Y, em_predictions)
km_confusion = confusion_matrix(Y, km_predictions)
em_confusion = confusion_matrix(Y, em_predictions)
print("Accuracy of KMeans is ",km_accuracy)
print("Accuracy of EM is ",em_accuracy)
print("Confusion matrix of KMeans: \n", km_confusion)
print("Confusion matrix of EM: \n", em_confusion)
```

# PROGRAM 9 – k – nearest neighbor algorithm

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split
iris_dataset=load_iris()

print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris_dataset.target_names)
for i in range(len(iris_dataset.target_names)):
    print("\n[{0}]:[{1}]".format(i,iris_dataset.target_names[i]))

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

print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)

i=10
for i in range(len(X_test)):
    x = X_test[i]
    x_new = np.array([x])
    prediction = kn.predict(x_new)
    print("\n Actual : {0} {1}, Predicted
:{2}{3}".format(y_test[i],iris_dataset["target_names"][y_test[i]],prediction,iris_dataset["target_n
ames"][ prediction]))

print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_test)))
```

PROGRAM 10 – Locally Weighted Regression

```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.layouts import gridplot

def plot_lwr(tau):
    # prediction
    domain = np.linspace(-3, 3, num=300)
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text = 'tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot

def local_regression(x0, X, Y, tau):
    # add bias term
    x0 = np.r_[1, x0] # r_ => Translates slice objects to concatenation along the first axis
    X = np.c_[np.ones(len(X)), X] # c_ => Translates slice objects to concatenation along the
second axis.
    # fit model: normal equations with kernel
    xw = X.T * radial_kernel(x0, X, tau)
    beta = np.linalg.pinv(xw @ X) @ xw @ Y
    # predict value
    return x0 @ beta

def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))

n = 1000
# generate dataset
X = np.linspace(-3, 3, num = n)
Y = np.log(np.abs(X ** 2 - 1) + .5)
# jitter X (Draw random samples from a normal (Gaussian) distribution)
X += np.random.normal(scale=.1, size=n)

show(gridplot([
    [plot_lwr(10.), plot_lwr(1.)],
    [plot_lwr(0.1), plot_lwr(0.01)]
]))
```

# Program 10 (modified)

```
import math
import pylab as plt
import math
import numpy as np

n = 100
x = np.linspace(0, 2 * math.pi, n)[:,np.newaxis]
y =np.sin(x) + 0.3*np.random.randn(n)[:,np.newaxis]

print("len",x)
print("len2",len(y))
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly = PolynomialFeatures(degree = 5)
X_poly = poly.fit_transform(x)
poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)

plt.plot(x, y, color = 'blue' , label='y noisy')
plt.plot(x, lin2.predict(poly.fit_transform(x)), color = 'red', label='y pred')
plt.title('Local Regression')
plt.legend()
plt.show()
```