**Machine Learning Lab File Course Code - BCSE3093**



**School of Computing Science and Engineering Greater Noida, Uttar Pradesh**

**Fall 2020 - 2021**

Under the Supervision of

**Mr. Pratyush Kumar Deka**

**Submitted by**

Name –Khushi Gupta

Admission no. – 18SCSE1010063 Enrollment no. – 18021011314

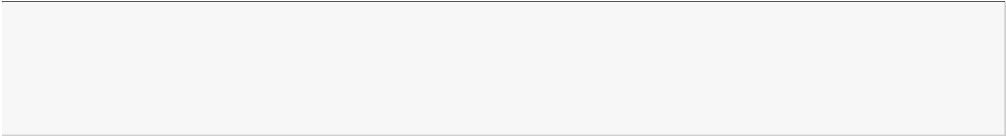
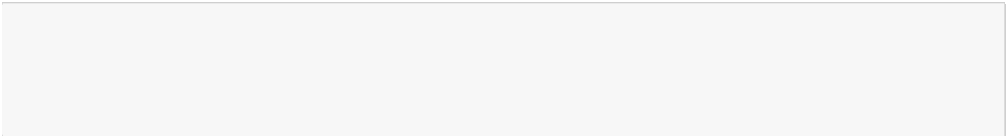
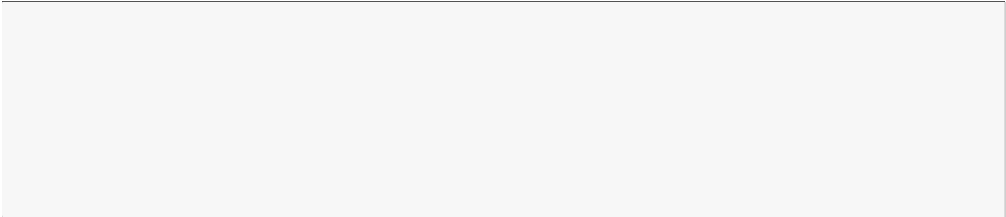
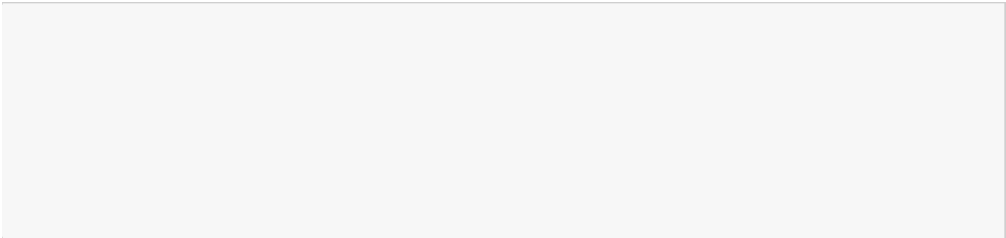
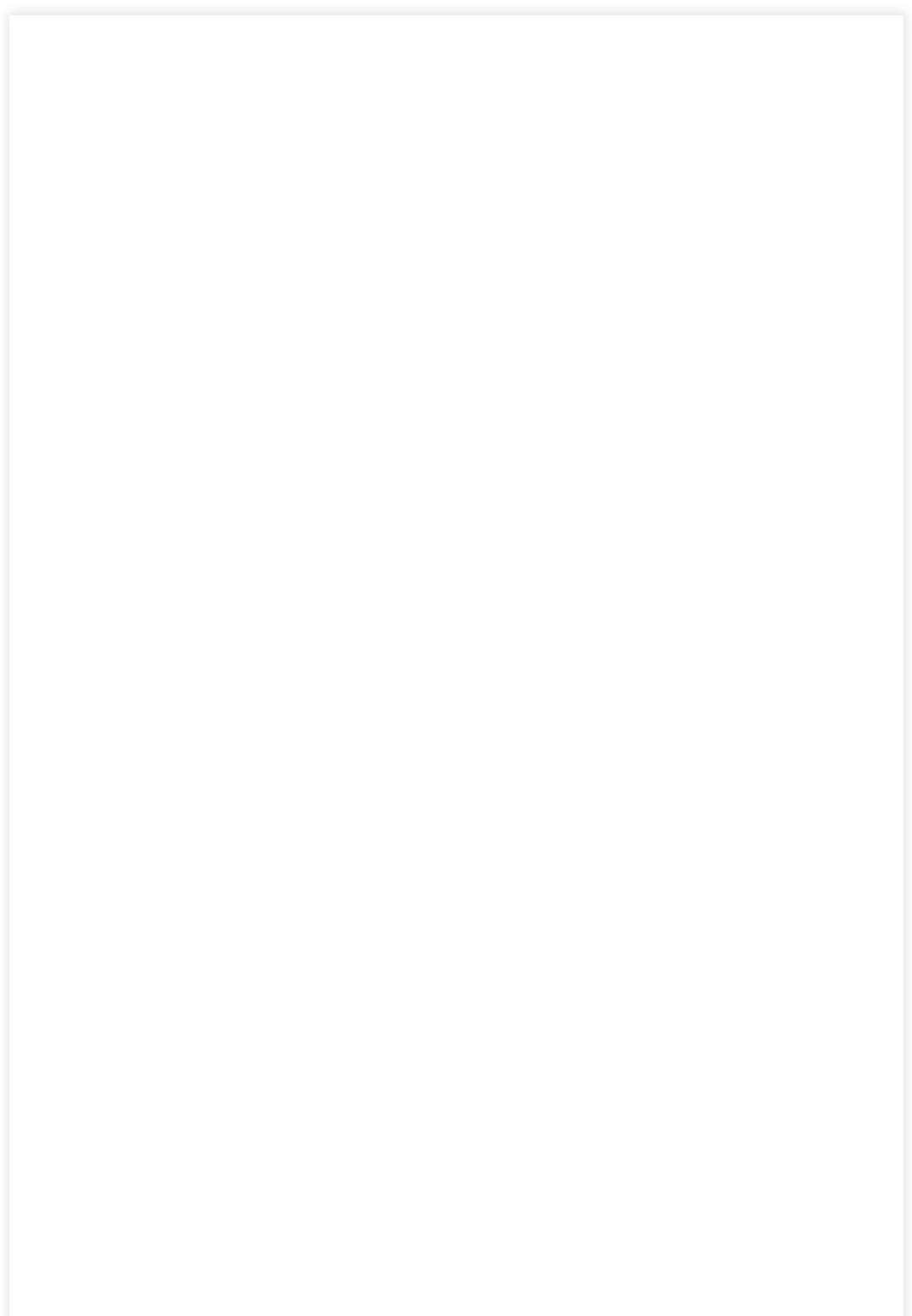
Elective Section - 2

**School of Computing Science & Engineering**

**Course Name- Machine Learning Lab Course Code- BCSE3093**

|  |  |
| --- | --- |
| **S. No.** | **Title of Lab Experiments** |
| 1. | Software setup of machine learning environment |
| 2. | Python revision and introduction to NumPy |
| 3. | Implement Linear Regression with one variable and multiple variable |
| 4. | Implement Logistic Regression to recognize hand-written digits |
| 5. | Implement k-nearest neighbors algorithm |
| 6. | Implement Support Vector Machines to build a spam classifier |

In [ ]:



*# EXPERIMENT 2*

*#Title : Python revision and introduction to NumPy*

*#Theory:Python is a high-level, dynamically typed multiparadigm programming language. #Python code isoften said to be almost like pseudocode, since it allows you to*

express

*#very powerful ideas invery few lines of code while being very readable. As an*

example,

*#here is an implementation ofthe classic quicksort algorithm in Python:*

In [1]:

**def** quicksort(arr):

**if** len(arr) <= 1:

**return** arr

pivot = arr[len(arr) // 2]

left = [x **for** x **in** arr **if** x < pivot] middle = [x **for** x **in** arr **if** x == pivot] right = [x **for** x **in** arr **if** x > pivot]

**return** quicksort(left) + middle + quicksort(right) print(quicksort([3,6,8,10,1,2,1]))

[1, 1, 2, 3, 6, 8, 10]

In [3]:

x = 3

print(x + 1) *# Addition* print(x - 1) *# Subtraction* print(x \* 2) *# Multiplication* print(x \*\* 2) *# Exponentiation*

4

2

6

9

In [4]:

t, f = **True**, **False**

print(t **and** f) *# Logical AND;* print(t **or** f) *# Logical OR;* print(**not** t) *# Logical NOT;* print(t != f) *# Logical XOR;*

False True False True

In [5]:

s = "hello"

print(s.capitalize()) *# Capitalize a string*

print(s.upper()) *# Convert a string to uppercase; prints "HELLO"* print(s.rjust(7)) *# Right-justify a string, padding with spaces* print(s.center(7)) *# Center a string, padding with spaces*

print(s.replace('l', '(ell)')) *# Replace all instances of one substring with anoth*

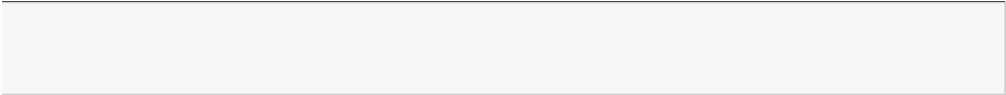
print(' world '.strip()) *# Strip leading and trailing whitespace*

Hello HELLO

hello hello

he(ell)(ell)o world

In [6]:



xs = [3, 1, 2] *# Create a list*

print(xs, xs[2])

print(xs[-1])

[3, 1, 2] 2

2

In [7]:

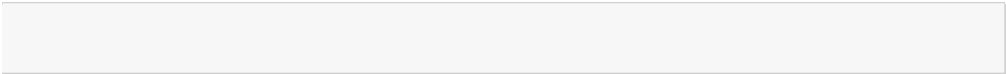


xs[2] = 'foo' print(xs)

*# Lists can contain elements of different types*

[3, 1, 'foo']

In [8]:

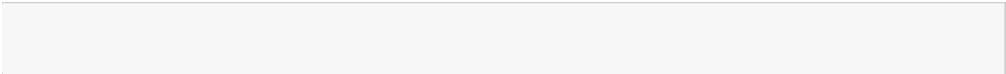


xs.append('bar') *# Add a new element to the end of the list*

print(xs)

[3, 1, 'foo', 'bar']

In [9]:

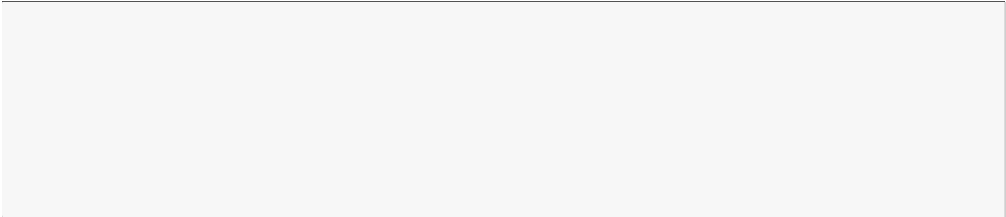


x = xs.pop() print(x, xs)

*# Remove and return the last element of the list*

bar [3, 1, 'foo']

In [10]:



nums = list(range(5))

*# range is a built-in function that creates a list of inte*

print(nums) *# Prints "[0, 1, 2, 3, 4]"*

print(nums[2:4]) *# Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"* print(nums[2:]) *# Get a slice from index 2 to the end; prints "[2, 3, 4]"* print(nums[:2]) *# Get a slice from the start to index 2 (exclusive); prints "[0* print(nums[:]) *# Get a slice of the whole list; prints ["0, 1, 2, 3, 4]"*

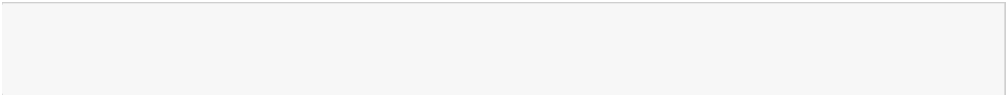
print(nums[:-1]) *# Slice indices can be negative; prints ["0, 1, 2, 3]"*

nums[2:4] = [8, 9] *# Assign a new sublist to a slice*

print(nums) *# Prints "[0, 1, 8, 9, 4]"*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [0, | 1, | 2, | 3, | 4] |
| [2, | 3] |  |  |  |
| [2, | 3, | 4] |  |  |
| [0, | 1] |  |  |  |
| [0, | 1, | 2, | 3, | 4] |
| [0, | 1, | 2, | 3] |  |
| [0, | 1, | 8, | 9, | 4] |

In [11]:



animals = ['cat', 'dog', 'monkey']

**for** idx, animal **in** enumerate(animals): print('#**{}**: **{}**'.format(idx + 1, animal))

#1: cat

#2: dog

#3: monkey

In [12]:



nums = [0, 1, 2, 3, 4]

even\_squares = [x \*\* 2 **for** x **in** nums **if** x % 2 == 0]

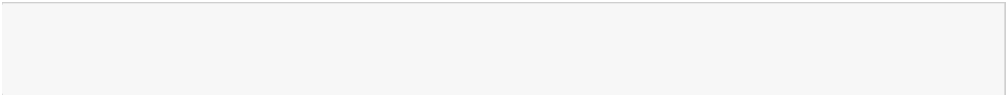




print(even\_squares)

[0, 4, 16]

In [15]:



d = {'cat': 'cute', 'dog': 'furry'} *# Create a new dictionary with some data* print(d['cat']) *# Get an entry from a dictionary; prints "cute"* print('cat' **in** d) *# Check if a dictionary has a given key; prints "True"*

cute True

In [16]:



animals = {'cat', 'dog'}

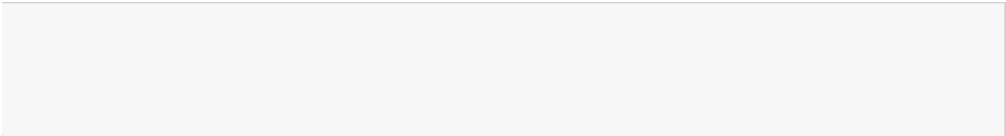
**True False**

print('cat' **in** animals) *# Check if an element is in a set; prints "True"*

print('fish' **in** animals) *# prints "False*

True False

In [17]:



d = {(x, x + 1): x **for** x **in** range(10)} *# Create a dictionary with tuple keys*

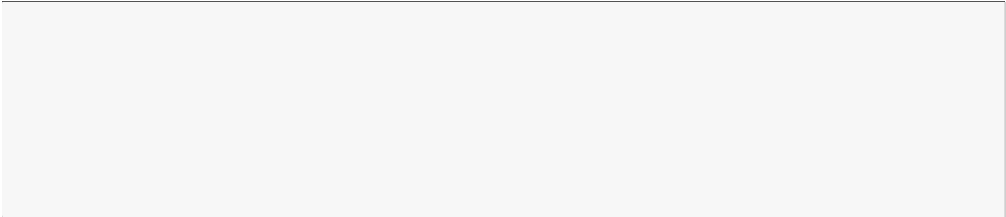
t = (5, 6) *# Create a tuple*

print(type(t)) print(d[t]) print(d[(1, 2)])

<class 'tuple'> 5

1

In [18]:



**def** sign(x):

**if** x > 0:

**return** 'positive'

**elif** x < 0:

**return** 'negative'

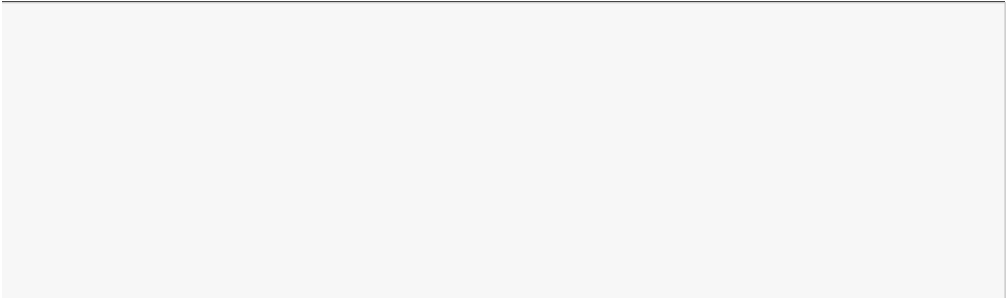
**else**:

**return** 'zero'

**for** x **in** [-1, 0, 1]: print(sign(x))

negative zero positive

In [19]:



**class Greeter**:

*# Constructor*

**def** init (self, name):

self.name = name *# Create an instance variable # Instance method*

**def** greet(self, loud=**False**):

**if** loud:

print('HELLO, **{}**'.format(self.name.upper())) **else**:

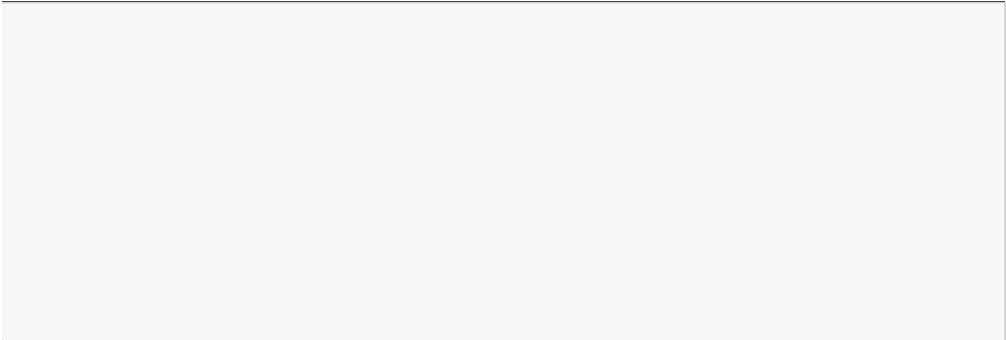
print('Hello, **{}**!'.format(self.name))

g = Greeter('Fred') *# Construct an instance of the Greeter class* g.greet() *# Call an instance method; prints "Hello, Fred"* g.greet(loud=**True**) *# Call an instance method; prints "HELLO, FRED!"*



Hello, Fred! HELLO, FRED

In [1]:



**import numpy as np**

a = np.array([1, 2, 3]) *# Create a rank 1 array*

print(type(a), a.shape, a[0], a[1], a[2])

a[0] = 5 *# Change an element of the array*

print(a)

b = np.array([[1,2,3],[4,5,6]]) *# Create a rank 2 array*

print(b)

b = np.array([[1,2,3],[4,5,6]]) *# Create a rank 2 array*

print(b)

a = np.zeros((2,2)) *# Create an array of all zeros*

print(a)

<class 'numpy.ndarray'> (3,) 1 2 3

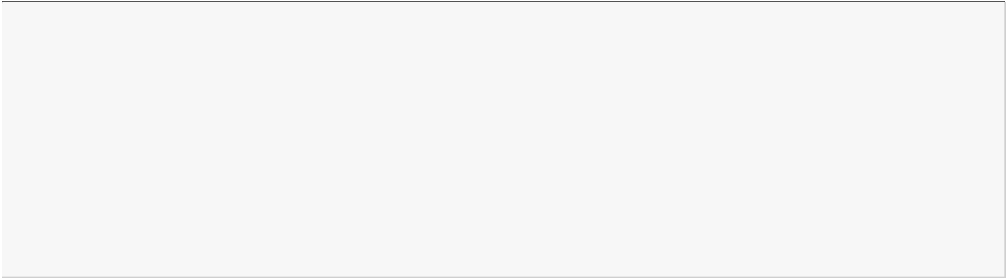
[5 2 3]

|  |  |  |
| --- | --- | --- |
| [[1 | 2 | 3] |
| [4 | 5 | 6]] |
| [[1 | 2 | 3] |
| [4 | 5 | 6]] |

[[0. 0.]

[0. 0.]]

In [2]:



**import numpy as np**

*# Create the following rank 2 array with shape (3, 4)*

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

*# Use slicing to pull out the subarray consisting of the first 2 rows # and columns 1 and 2; b is the following array of shape (2, 2):*

*# [[2 3]*

*# [6 7]]*

b = a[:2, 1:3]

print (b)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *[[ 1* | *2* | *3* | *4]* |
| *#* | *[ 5* | *6* | *7* | *8]* |
| *#* | *[ 9* | *10* | *11* | *12]]* |

[[2 3]

[6 7]]

In [3]:



x = np.array([1, 2]) *# Let numpy choose the datatype*

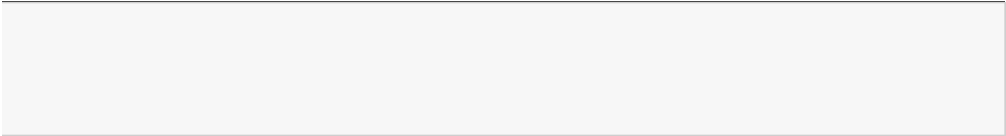
y = np.array([1.0, 2.0]) *# Let numpy choose the datatype*

z = np.array([1, 2], dtype=np.int64) *# Force a particular datatype*

print(x.dtype, y.dtype, z.dtype)

int32 float64 int64

In [4]:



x = np.array([[1,2],[3,4]], dtype=np.float64)

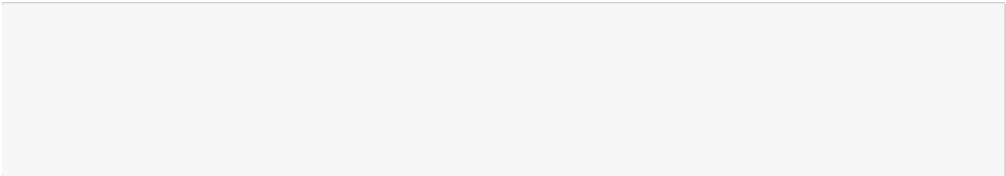
y = np.array([[5,6],[7,8]], dtype=np.float64) *# Elementwise sum; both produce the array* print(x + y)

print(np.add(x, y))

|  |  |
| --- | --- |
| [[ 6. | 8.] |
| [10. | 12.]] |
| [[ 6. | 8.] |
| [10. | 12.]] |



In [5]:



*# Elementwise difference; both produce the array*

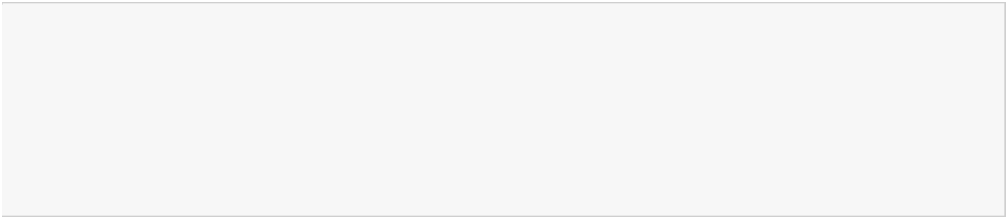
print(x - y) print(np.subtract(x, y))

*# Elementwise product; both produce the array*

print(x \* y) print(np.multiply(x, y))

|  |  |
| --- | --- |
| [[-4. | -4.] |
| [-4. | -4.]] |
| [[-4. | -4.] |
| [-4. | -4.]] |
| [[ 5. | 12.] |
| [21. | 32.]] |
| [[ 5. | 12.] |
| [21. | 32.]] |

In [6]:



*# We will add the vector v to each row of the matrix x, # storing the result in the matrix y*

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

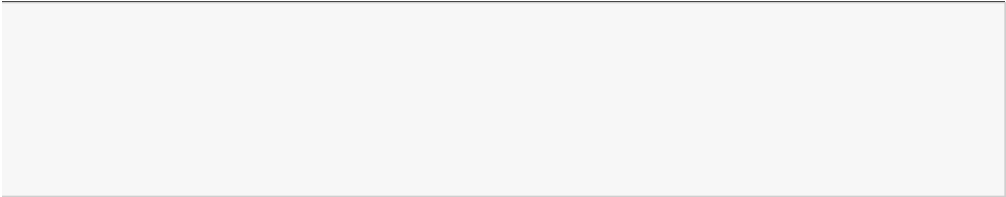
v = np.array([1, 0, 1])

y = np.empty\_like(x) *# Create an empty matrix with the same shape as x # Add the vector v to each row of the matrix x with an explicit loop* **for** i **in** range(4):

y[i, :] = x[i, :] + v print(y)

|  |  |  |
| --- | --- | --- |
| [[ 2 | 2 | 4] |
| [ 5 | 5 | 7] |
| [ 8 | 8 | 10] |
| [11 | 11 | 13]] |

In [7]:



vv = np.tile(v, (4, 1)) *# Stack 4 copies of v on top of each other*

print(vv) *# Prints "[[1 0 1]*

*# [1 0 1]*

*# [1 0 1]*

*# [1 0 1]]"*

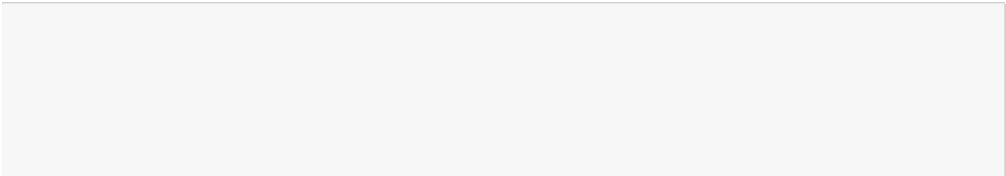
y = x + vv *# Add x and vv elementwise*

print(y)

|  |  |  |
| --- | --- | --- |
| [[1 | 0 | 1] |
| [1 | 0 | 1] |
| [1 | 0 | 1] |
| [1 | 0 | 1]] |
| [[ 2 |  | 2 4] |
| [ 5 |  | 5 7] |
| [ 8 |  | 8 10] |

[11 11 13]]

In [8]:



**import numpy as np**

*# We will add the vector v to each row of the matrix x, # storing the result in the matrix y*

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

v = np.array([1, 0, 1])

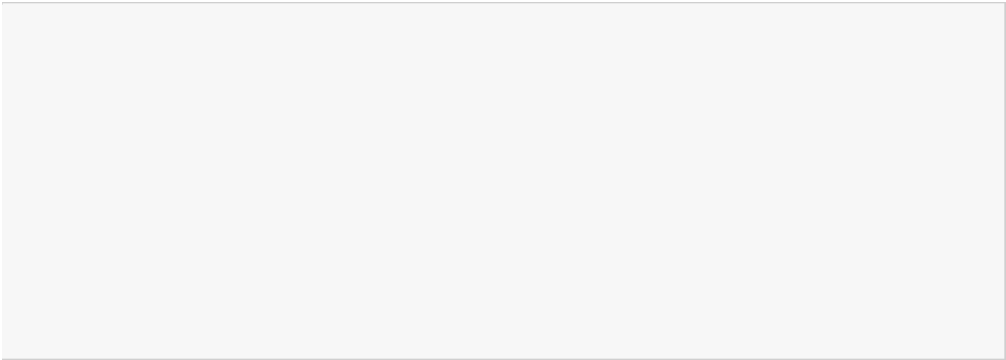
y = x + v *# Add v to each row of x using broadcasting*

print (y)

|  |  |  |
| --- | --- | --- |
| [[ 2 | 2 | 4] |
| [ 5 | 5 | 7] |
| [ 8 | 8 | 10] |
| [11 | 11 | 13]] |



In [9]:



**import matplotlib.pyplot as plt**

%**matplotlib** inline

*# Compute the x and y coordinates for points on a sine curve*

x = np.arange(0, 3 \* np.pi, 0.1) y = np.sin(x)

*# Plot the points using matplotlib*

plt.plot(x, y) y\_sin = np.sin(x) y\_cos = np.cos(x)

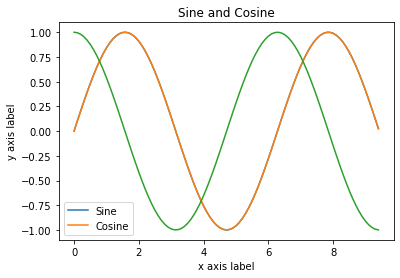
*# Plot the points using matplotlib*

plt.plot(x, y\_sin) plt.plot(x, y\_cos) plt.xlabel('x axis label') plt.ylabel('y axis label') plt.title('Sine and Cosine')

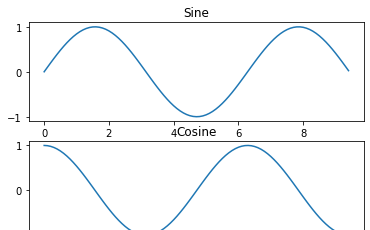
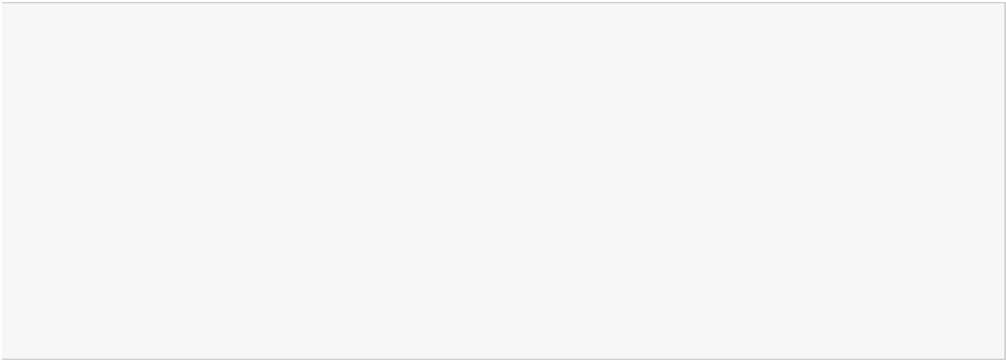
plt.legend(['Sine', 'Cosine'])

Out[9]:

<matplotlib.legend.Legend at 0x24712b1fbc8>



In [10]:



*# Compute the x and y coordinates for points on sine and cosine curves*

x = np.arange(0, 3 \* np.pi, 0.1) y\_sin = np.sin(x)

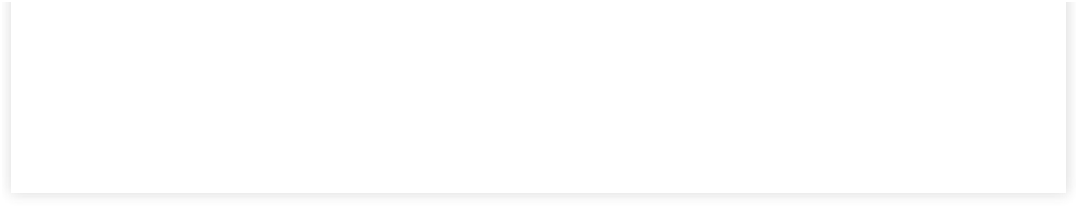
y\_cos = np.cos(x)

*# Set up a subplot grid that has height 2 and width 1, # and set the first such subplot as active.* plt.subplot(2, 1, 1)

*# Make the first plot* plt.plot(x, y\_sin) plt.title('Sine')

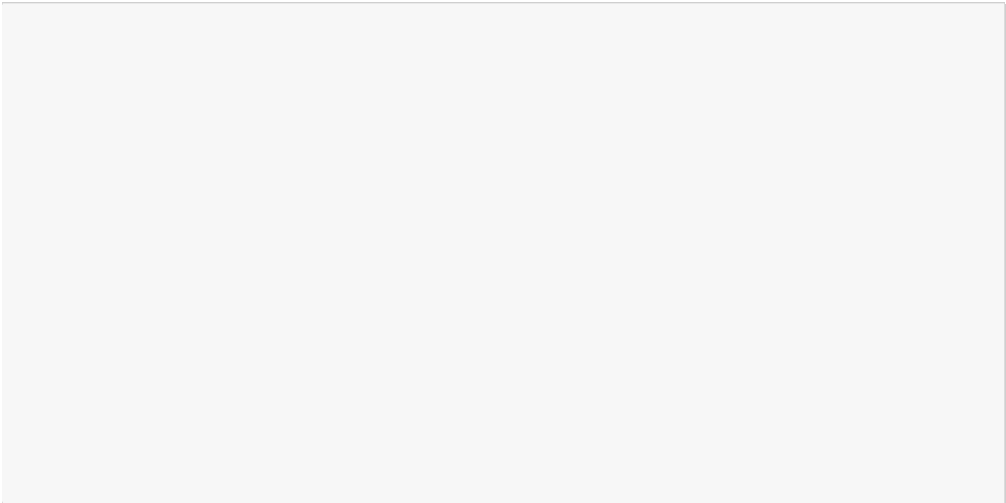
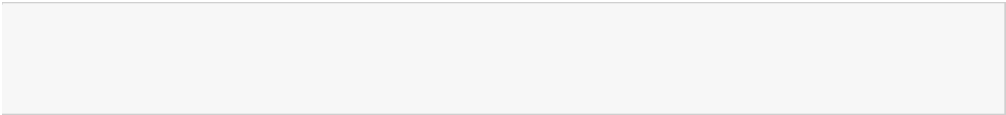
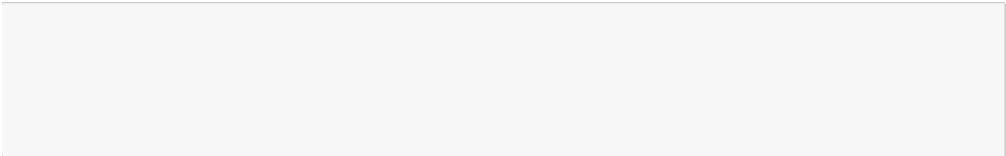
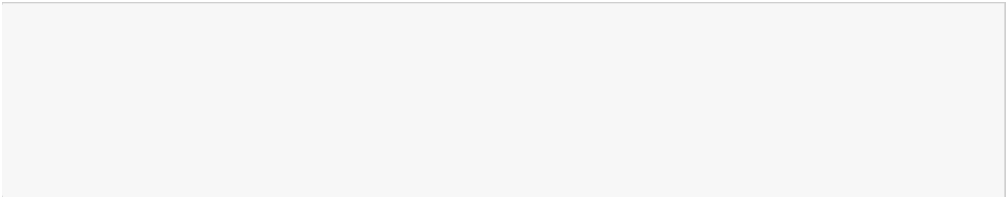
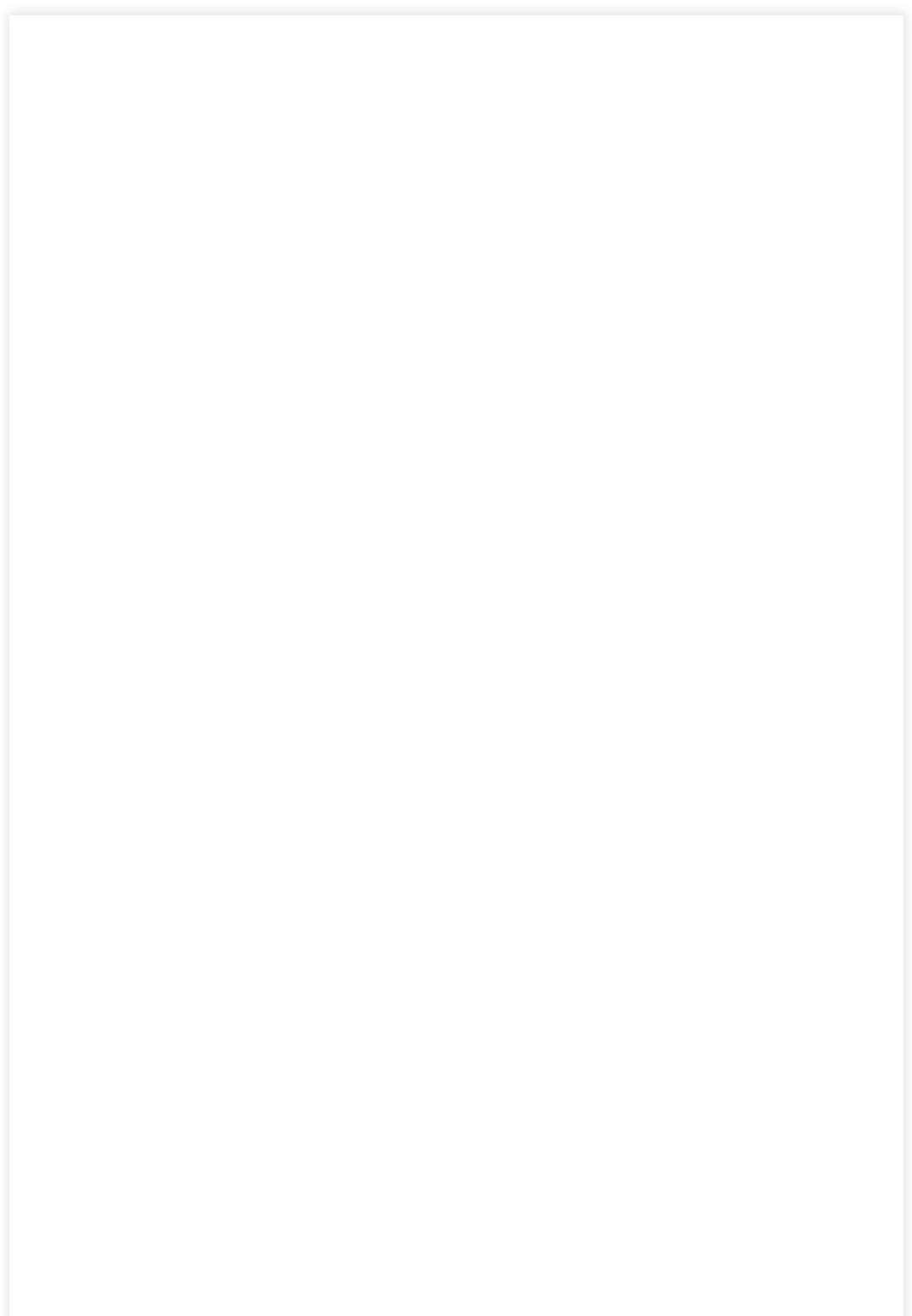
*# Set the second subplot as active, and make the second plot.*

plt.subplot(2, 1, 2) plt.plot(x, y\_cos) plt.title('Cosine') *# Show the figure.* plt.show()



In [ ]:

In [ ]:



*#Experiment 3*

*#Title : Implement Linear Regression with one variable and multiple variable*

*#Theory : linear regression is a linear approach to modeling the relationship #between a scalar response (or dependent variable) and one or more explanatory #variables (or independent variables). The case of one explanatory variable is #called simple linear regression.*

In [1]:

**from sklearn.ensemble import** RandomForestClassifier clf = RandomForestClassifier(random\_state=0)

X = [[ 1, 2, 3], *# 2 samples, 3 features*

[11, 12, 13]]

y = [0, 1] *# classes of each sample*

clf.fit(X, y)

Out[1]:

RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,

criterion='gini', max\_depth=None, max\_features='auto', max\_leaf\_nodes=None, max\_samples=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=None, oob\_score=False, random\_state=0, verbose=0, warm\_start=False)

In [2]:

**from sklearn.preprocessing import** StandardScaler X = [[0, 15],

[1, -10]]

StandardScaler().fit(X).transform(X)

Out[2]:

array([[-1., 1.],

[ 1., -1.]])

In [3]:

**from sklearn.preprocessing import** StandardScaler **from sklearn.linear\_model import** LogisticRegression **from sklearn.pipeline import** make\_pipeline

**from sklearn.datasets import** load\_iris

**from sklearn.model\_selection import** train\_test\_split

**from sklearn.metrics import** accuracy\_score

*# create a pipeline object*

pipe = make\_pipeline( StandardScaler(), LogisticRegression(random\_state=0)

)

*# load the iris dataset and split it into train and test sets*

X, y = load\_iris(return\_X\_y=**True**)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

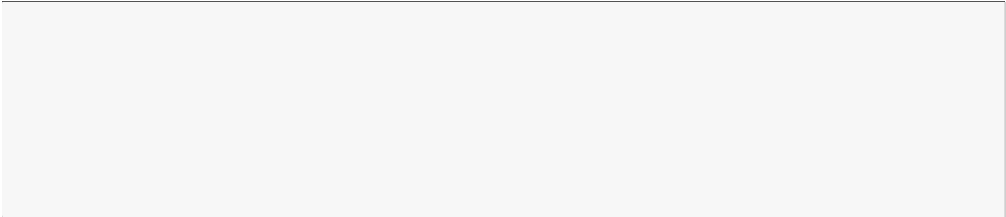
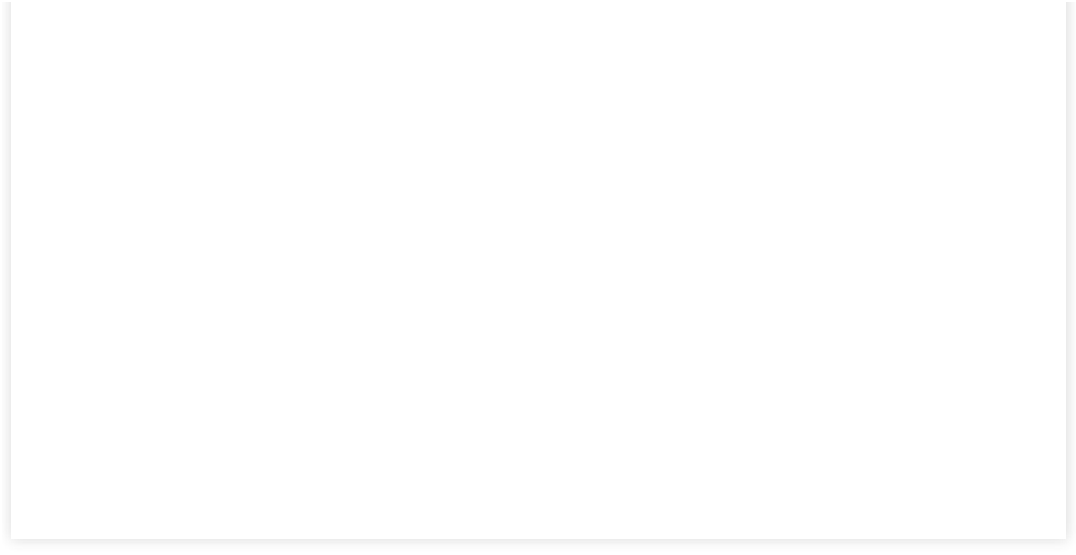
*# fit the whole pipeline*

pipe.fit(X\_train, y\_train)

*# we can now use it like any other estimator*

accuracy\_score(pipe.predict(X\_test), y\_test)

Out[3]:



0.9736842105263158

In [4]:

**from sklearn.datasets import** make\_regression

**from sklearn.linear\_model import** LinearRegression

**from sklearn.model\_selection import** cross\_validate

X, y = make\_regression(n\_samples=1000, random\_state=0) lr = LinearRegression()

result = cross\_validate(lr, X, y) *# defaults to 5-fold CV*

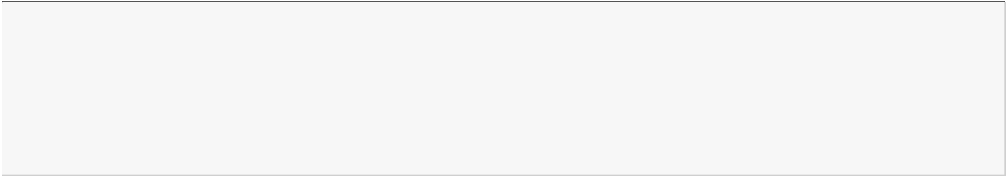
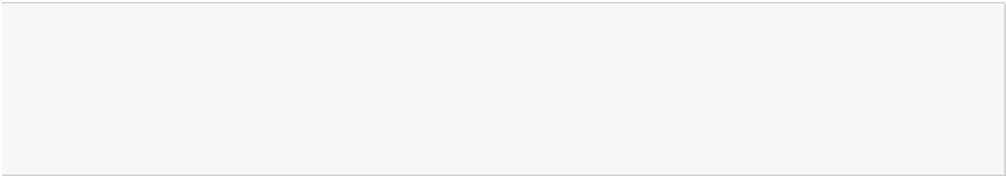
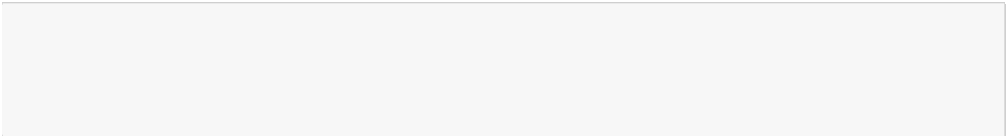
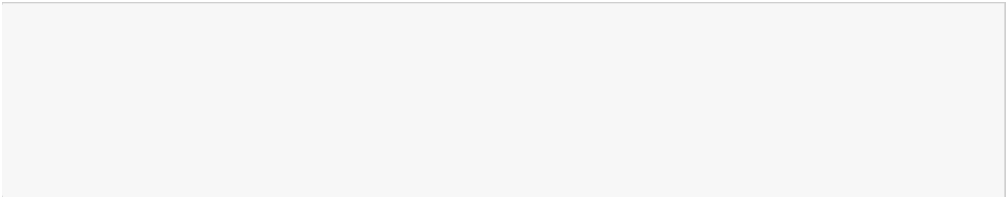
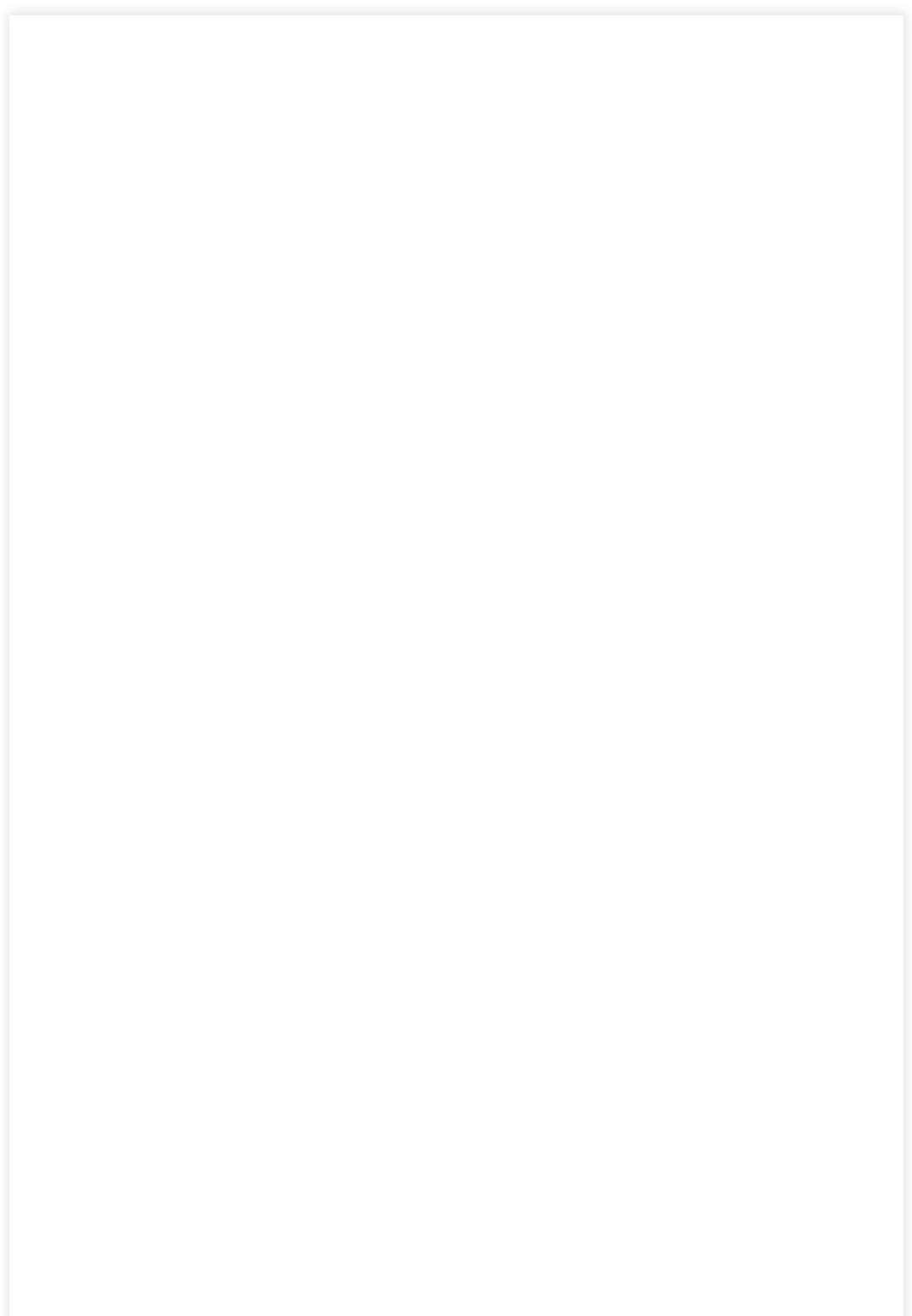
result['test\_score'] *# r\_squared score is high because dataset is easy*

Out[4]:

array([1., 1., 1., 1., 1.])

In [ ]:

In [ ]:



*#Experiment 4*

*#Title : Implement Logistic Regression to recognize hand-written digits*

*#Theory : Logistic Regression is an algorithm that is admirably suited for #discovering the link between features or cues and some particular outcome.*

*#Logistic regression is one of the most important analytic tools in the social and natural sci ences.*

In [1]:

**from sklearn import** linear\_model

reg = linear\_model.LinearRegression() reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

reg.coef\_

Out[1]:

array([0.5, 0.5])

In [2]:

**from sklearn import** linear\_model reg = linear\_model.Ridge(alpha=.5)

reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1])

reg.coef\_ reg.intercept\_

Out[2]: 0.1363636363636364

In [3]:

# import numpy as np

**from sklearn import** linear\_model

reg = linear\_model.RidgeCV(alphas=np.logspace(-6, 6, 13))

reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1])

reg.alpha\_

Out[3]: 0.01

In [4]:

**from sklearn import** linear\_model reg = linear\_model.Lasso(alpha=0.1) reg.fit([[0, 0], [1, 1]], [0, 1])

reg.predict([[1, 1]])

Out[4]:

array([0.8])

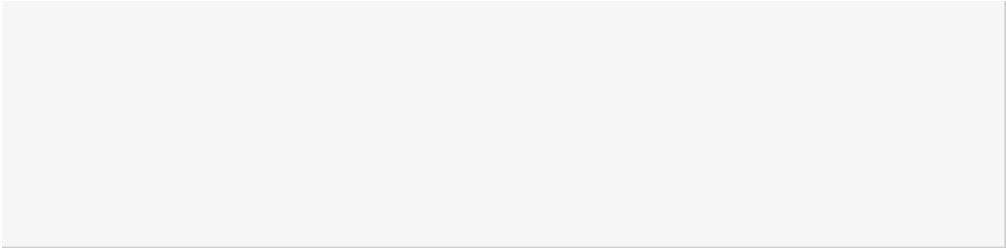
In [5]:

*#First example of OLS*

# import numpy as np

**from sklearn.linear\_model import** LinearRegression





X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])

*# y = 1 \* x\_0 + 2 \* x\_1 + 3*

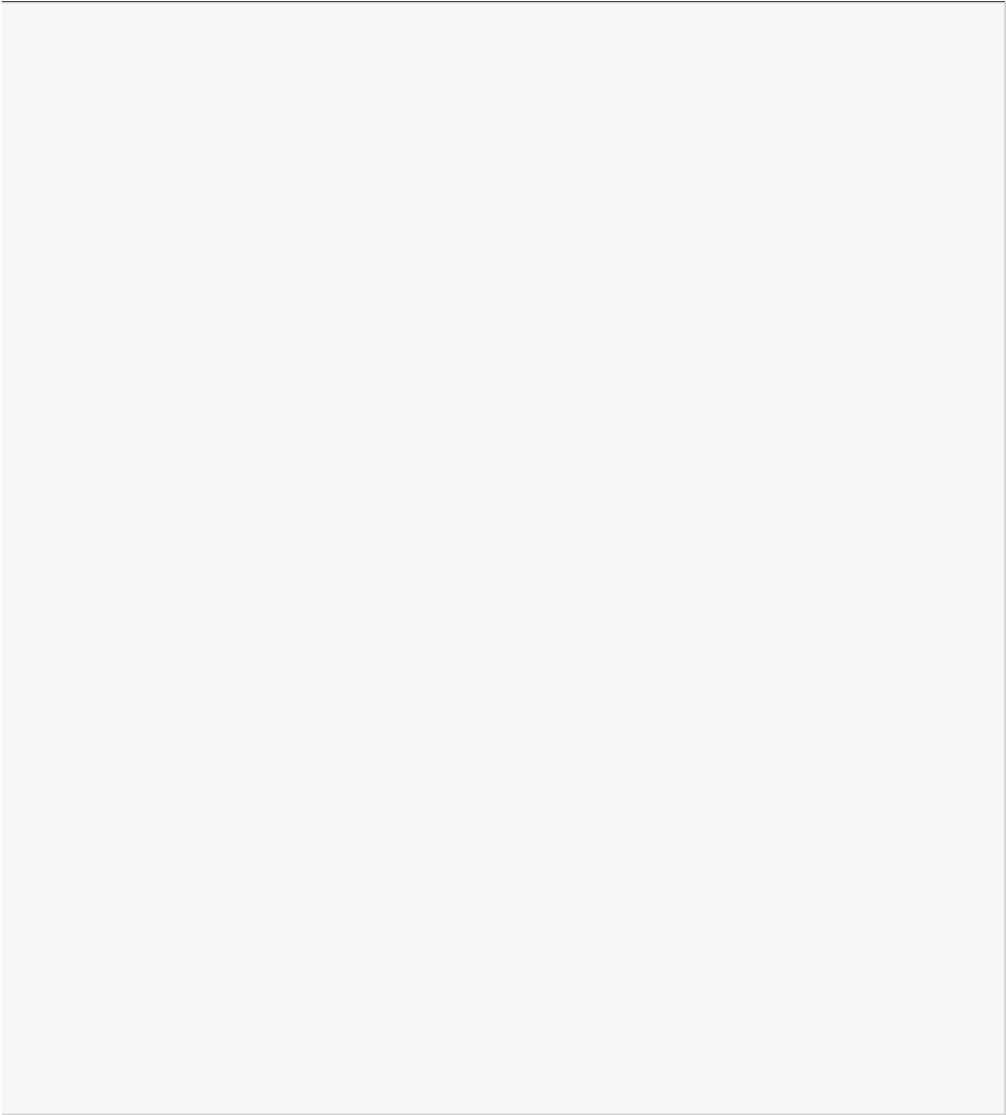
y = np.dot(X, np.array([1, 2])) + 3 reg = LinearRegression().fit(X, y) reg.score(X, y)

reg.coef\_ reg.intercept\_

reg.predict(np.array([[3, 5]]))

Out[5]:

array([16.])

In [6]:

*#Linear Regression with in-built Diabetes dataset*

print( doc )

*# Code source: Jaques Grobler # License: BSD 3 clause*

# import matplotlib.pyplot as plt import numpy as np

**from sklearn import** datasets, linear\_model

**from sklearn.metrics import** mean\_squared\_error, r2\_score

*# Load the diabetes dataset*

diabetes\_X, diabetes\_y = datasets.load\_diabetes(return\_X\_y=**True**)

*# Use only one feature*

diabetes\_X = diabetes\_X[:, np.newaxis, 2]

*# Split the data into training/testing sets* diabetes\_X\_train = diabetes\_X[:-20] diabetes\_X\_test = diabetes\_X[-20:]

*# Split the targets into training/testing sets* diabetes\_y\_train = diabetes\_y[:-20] diabetes\_y\_test = diabetes\_y[-20:]

*# Create linear regression object*

regr = linear\_model.LinearRegression()

*# Train the model using the training sets*

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

*# Make predictions using the testing set*

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

*# The coefficients* print('Coefficients: **\n**', regr.coef\_) *# The mean squared error*

print('Mean squared error: **%.2f**'

% mean\_squared\_error(diabetes\_y\_test, diabetes\_y\_pred)) *# The coefficient of determination: 1 is perfect prediction* print('Coefficient of determination: **%.2f**'

% r2\_score(diabetes\_y\_test, diabetes\_y\_pred))

*# Plot outputs*

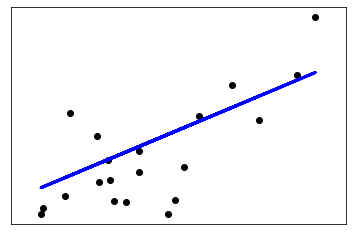
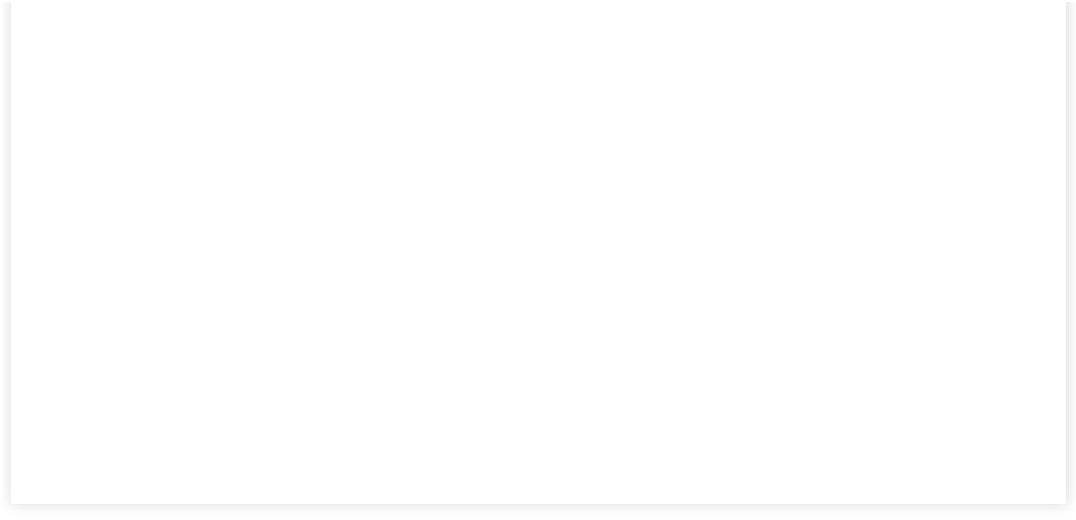
plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color='black') plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color='blue', linewidth=3)

plt.xticks(())

plt.yticks(()) plt.show()

Automatically created module for IPython interactive environment Coefficients:

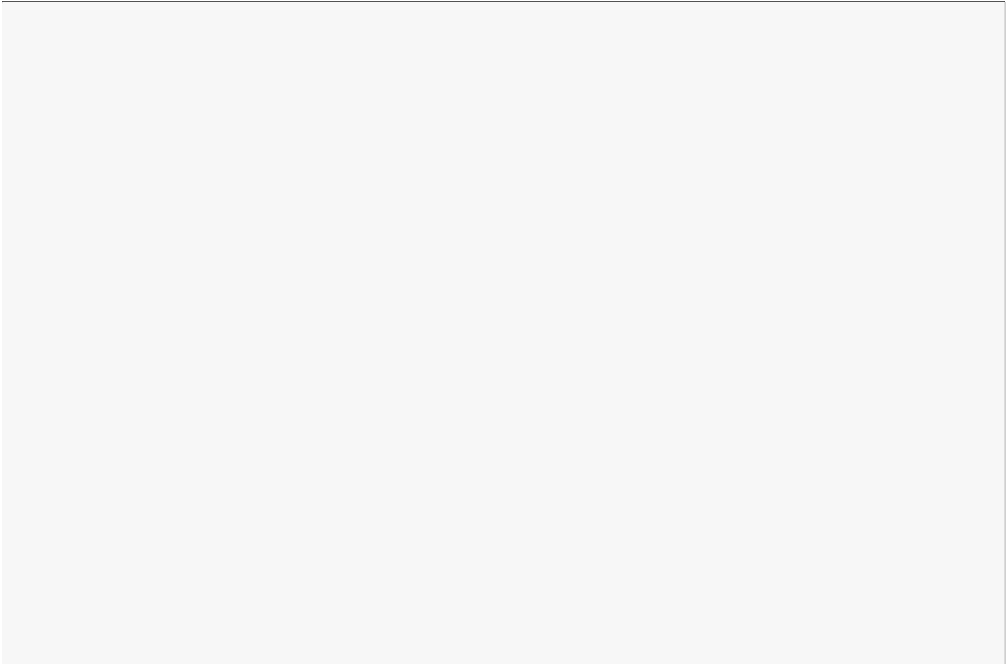
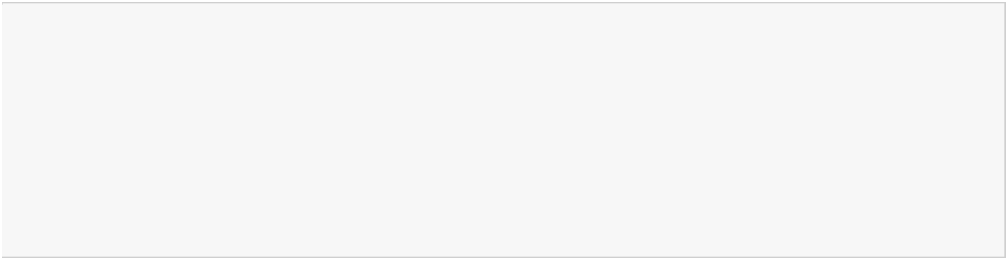
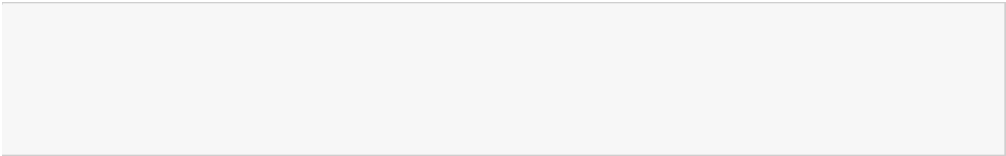
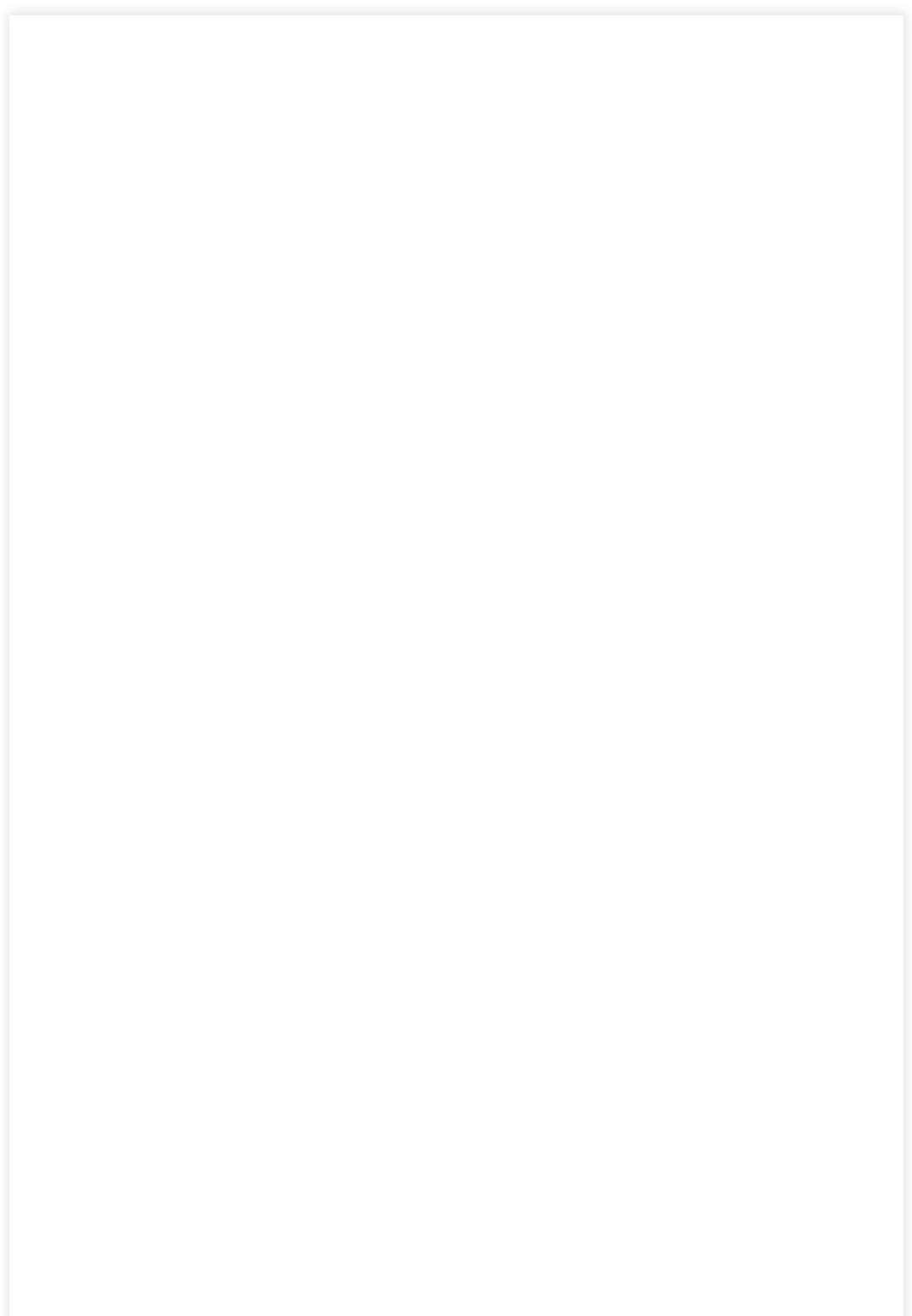
[938.23786125]



Mean squared error: 2548.07 Coefficient of determination: 0.47

In [ ]:

In [ ]:



*#Experiment 5*

*#Title : Implement k-nearest neighbors algorithm*

*# Theory: K nearest neighbors is a simple algorithm that stores*

*#all available cases and classifies new cases based on a similarity measure*

In [1]:

*#1.Study Logistic Regression in Scikit-learn*

**from sklearn.datasets import** load\_iris

**from sklearn.linear\_model import** LogisticRegression X, y = load\_iris(return\_X\_y=**True**)

clf = LogisticRegression(random\_state=0).fit(X, y) clf.predict(X[:2, :])

clf.predict\_proba(X[:2, :])

clf.score(X, y)

C:\Users\hp\anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

extra\_warning\_msg=\_LOGISTIC\_SOLVER\_CONVERGENCE\_MSG)

Out[1]: 0.9733333333333334

In [2]:

*#2. Build a classifier using Logistic Regression with in-built Iris dataset*

print( doc )

# import numpy as np

**import matplotlib.pyplot as plt**

**from sklearn.linear\_model import** LogisticRegression

**from sklearn import** datasets

*# import some data to play with*

iris = datasets.load\_iris()

X = iris.data[:, :2] *# we only take the first two features.*

Y = iris.target

logreg = LogisticRegression(C=1e5)

*# Create an instance of Logistic Regression Classifier and fit the data.*

logreg.fit(X, Y)

*# Plot the decision boundary. For that, we will assign a color to each # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].*

x\_min, x\_max = X[:, 0].min() - .5, X[:, 0].max() + .5

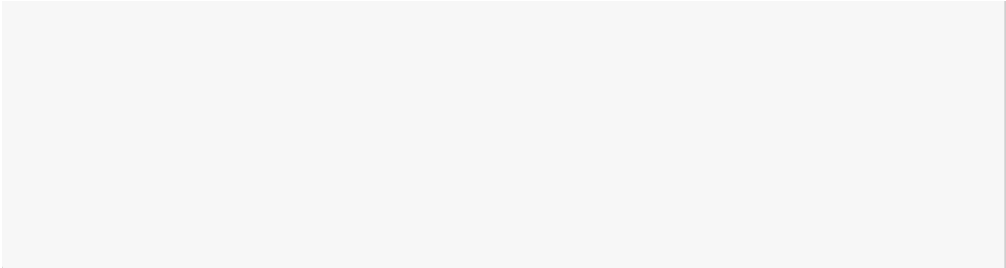
y\_min, y\_max = X[:, 1].min() - .5, X[:, 1].max() + .5 h = .02 *# step size in the mesh*

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h)) Z = logreg.predict(np.c\_[xx.ravel(), yy.ravel()])

*# Put the result into a color plot* Z = Z.reshape(xx.shape) plt.figure(1, figsize=(4, 3))

plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)





*# Plot also the training points*

plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired) plt.xlabel('Sepal length')

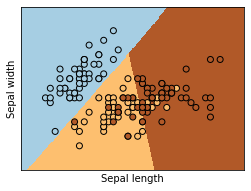
plt.ylabel('Sepal width')

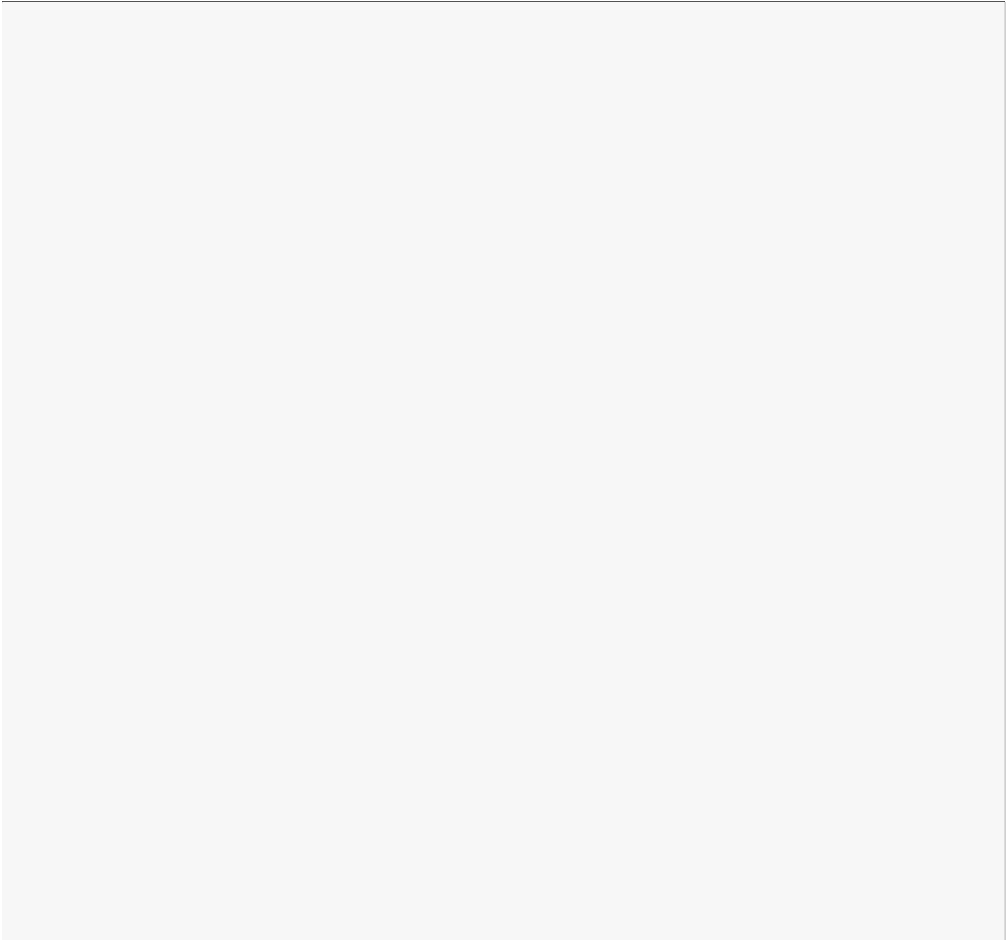
plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max()) plt.xticks(())

plt.yticks(()) plt.show()

Automatically created module for IPython interactive environment



In [3]:

*#3.Change features from sepal length and width to petal length and width. Build the classifier aga in and discuss the output.*

print( doc )

# import numpy as np

**import matplotlib.pyplot as plt**

**from sklearn.linear\_model import** LogisticRegression

**from sklearn import** datasets

*# import some data to play with*

iris = datasets.load\_iris()

X = iris.data[:, :2] *# we only take the first two features.*

Y = iris.target

logreg = LogisticRegression(C=1e5)

*# Create an instance of Logistic Regression Classifier and fit the data.*

logreg.fit(X, Y)

*# Plot the decision boundary. For that, we will assign a color to each # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].*

x\_min, x\_max = X[:, 0].min() - .5, X[:, 0].max() + .5

y\_min, y\_max = X[:, 1].min() - .5, X[:, 1].max() + .5 h = .02 *# step size in the mesh*

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h)) Z = logreg.predict(np.c\_[xx.ravel(), yy.ravel()])

*# Put the result into a color plot* Z = Z.reshape(xx.shape) plt.figure(1, figsize=(6, 6))

plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

*# Plot also the training points*

plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired) plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

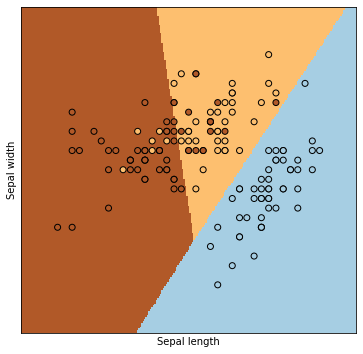
plt.xlim(xx.max(), xx.min())

plt.ylim(yy.max(), yy.min()) plt.xticks(())

plt.yticks(()) plt.show()



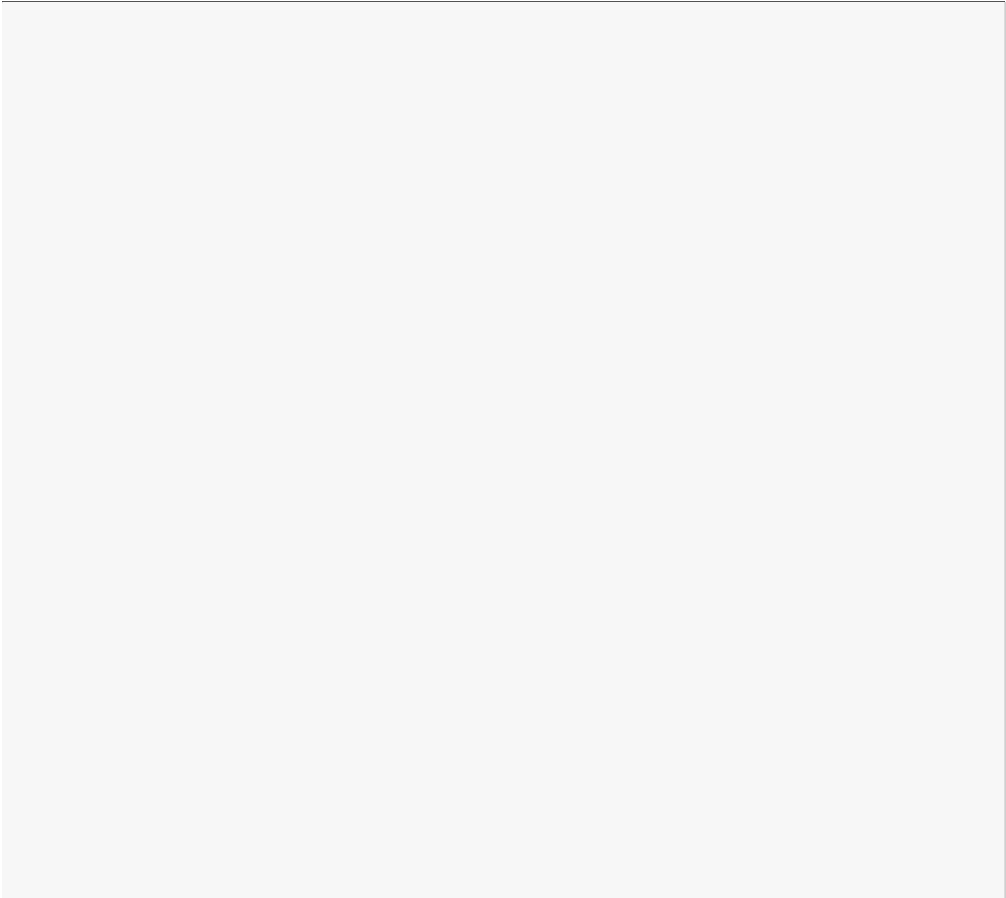
Automatically created module for IPython interactive environment



In [ ]:



sepal width **and** sepal length get changed also graph get swapped.

In [4]:

*#4.Consider all the features and build the classifier again and discuss the output.*

print( doc )

# import numpy as np

**import matplotlib.pyplot as plt**

**from sklearn.linear\_model import** LogisticRegression

**from sklearn import** datasets

*# import some data to play with*

iris = datasets.load\_iris()

X = iris.data[:, :2] *# we only take the first two features.*

Y = iris.target

logreg = LogisticRegression(C=1e5)

*# Create an instance of Logistic Regression Classifier and fit the data.*

logreg.fit(X, Y)

*# Plot the decision boundary. For that, we will assign a color to each # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].*

x\_min, x\_max = X[:, 0].min() - .7, X[:, 0].max() + .7

y\_min, y\_max = X[:, 1].min() - .7, X[:, 1].max() + .7 h = .02 *# step size in the mesh*

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h)) Z = logreg.predict(np.c\_[xx.ravel(), yy.ravel()])

*# Put the result into a color plot* Z = Z.reshape(xx.shape) plt.figure(1, figsize=(10, 6))

plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

*# Plot also the training points*

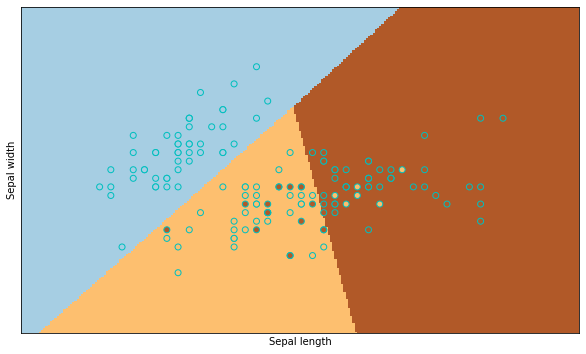
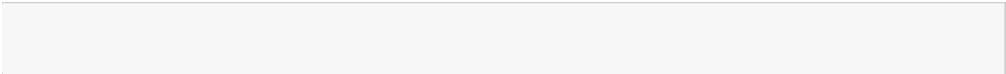
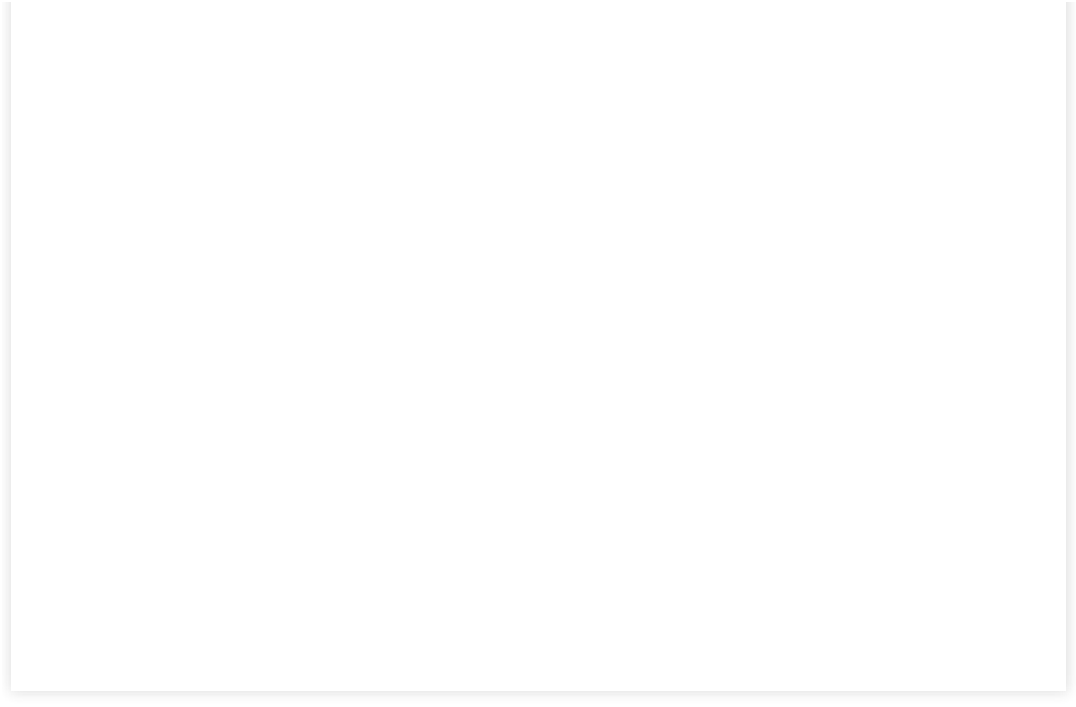
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='c', cmap=plt.cm.Paired) plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max()) plt.xticks(())

plt.yticks(()) plt.show()

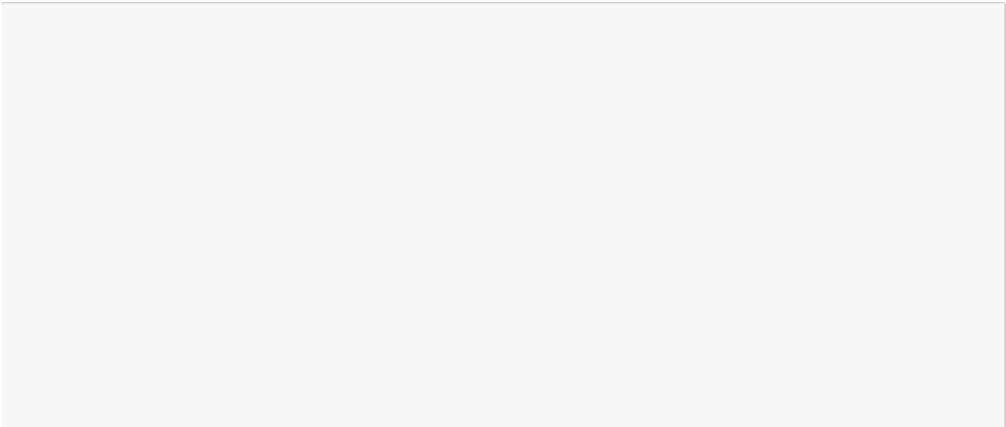
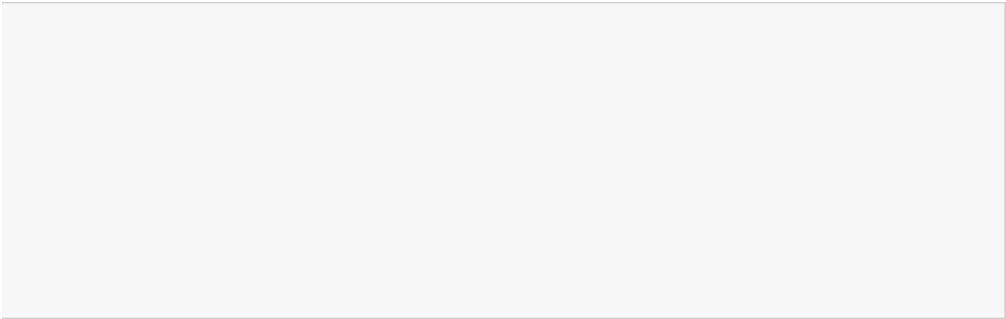
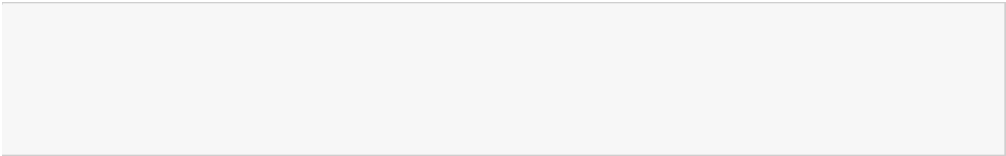
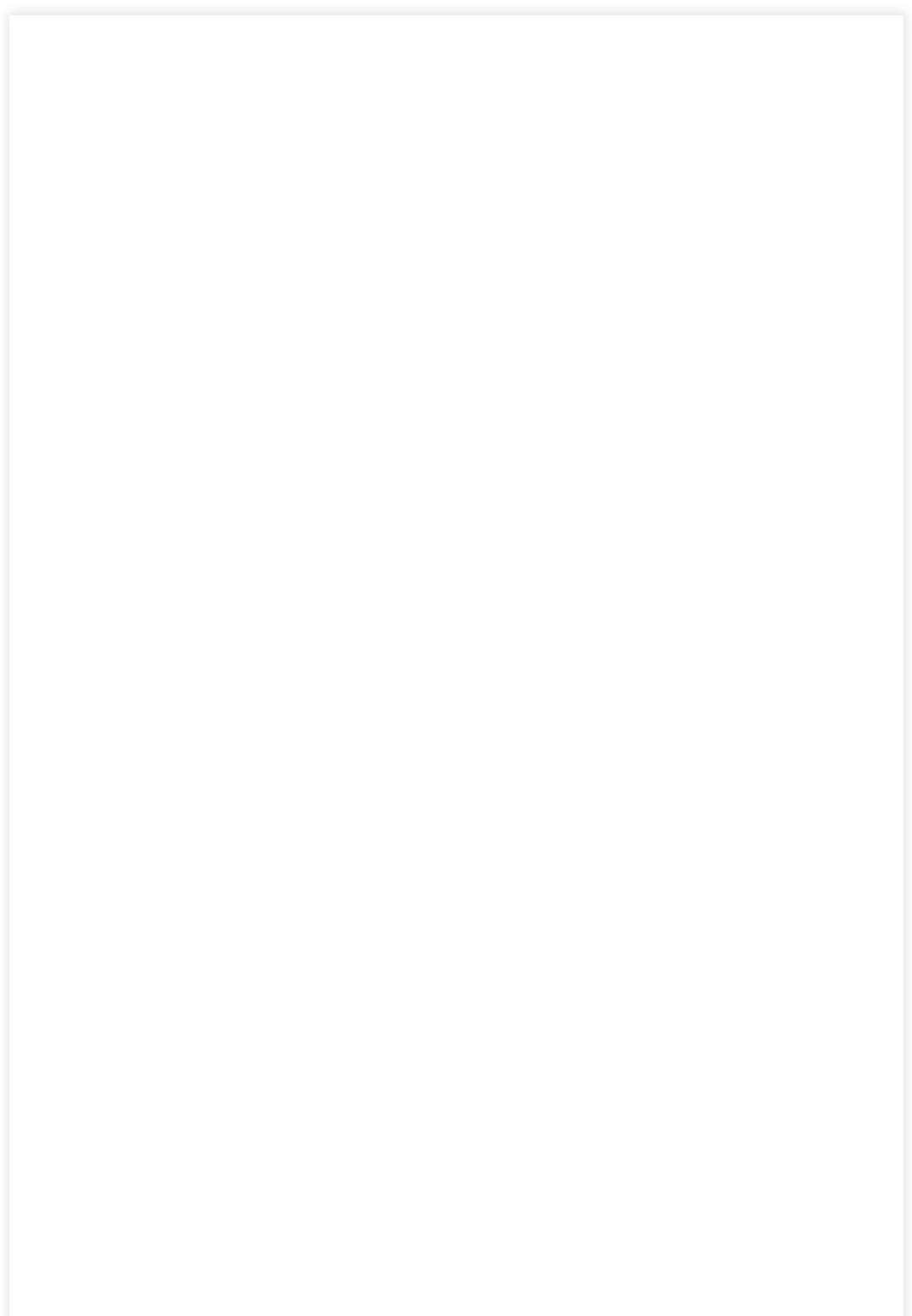


Automatically created module for IPython interactive environment

In [ ]:

length **and** width get changed. Also colour of points get changed. Spaces between points also get cha nged.

In [ ]:



*#Experiment6*

*#Title: Implement Support Vector Machines to build a spam classifier*

*#Theory: A support vector machine is a supervised machine learning model used for classification.*

In [1]:

*#Nearest Neighbors Classification in Scikit-learn* **from sklearn.neighbors import** NearestNeighbors **import numpy as np**

X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])

nbrs = NearestNeighbors(n\_neighbors=2, algorithm='ball\_tree').fit(X) distances, indices = nbrs.kneighbors(X)

indices

|  |  |  |  |
| --- | --- | --- | --- |
| distances |  | | |
| Out[1]: |
| array([[0. | , | 1. | ], |
| [0. | , | 1. | ], |
| [0. , 1.41421356], | | | |
| [0. | , | 1. | ], |
| [0. | , | 1. | ], |

[0. , 1.41421356]])

In [2]:

nbrs.kneighbors\_graph(X).toarray()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[2]:  array([[1., | 1., | 0., | 0., | 0., | 0.], |
| [1., | 1., | 0., | 0., | 0., | 0.], |
| [0., | 1., | 1., | 0., | 0., | 0.], |
| [0., | 0., | 0., | 1., | 1., | 0.], |
| [0., | 0., | 0., | 1., | 1., | 0.], |
| [0., | 0., | 0., | 0., | 1., | 1.]]) |

In [3]:

*#Build a classifier using k-Nearest Neighbors Classifier with in-built Iris dataset and plot the d ecision boundaries of each class*

print( doc )

# import numpy as np

**import matplotlib.pyplot as plt**

**from matplotlib.colors import** ListedColormap

**from sklearn import** neighbors, datasets n\_neighbors = 15

*# import some data to play with*

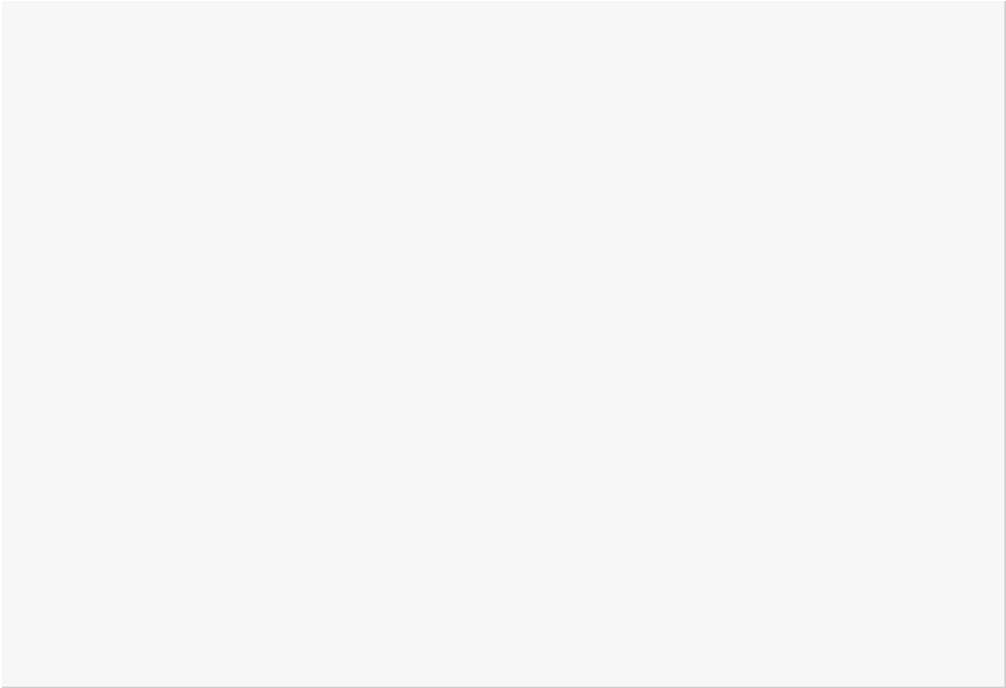
iris = datasets.load\_iris()

*# we only take the first two features. We could avoid this ugly # slicing by using a two-dim dataset*

X = iris.data[:, :2] y = iris.target

h = .02 *# step size in the mesh*



*# Create color maps*

cmap\_light = ListedColormap(['orange', 'cyan', 'cornflowerblue']) cmap\_bold = ListedColormap(['darkorange', 'c', 'darkblue'])

**for** weights **in** ['uniform', 'distance']:

*# we create an instance of Neighbours Classifier and fit the data.* clf = neighbors.KNeighborsClassifier(n\_neighbors, weights=weights) clf.fit(X, y)

*# Plot the decision boundary. For that, we will assign a color to each # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].*

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h)) Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

*# Put the result into a color plot* Z = Z.reshape(xx.shape) plt.figure()

plt.pcolormesh(xx, yy, Z, cmap=cmap\_light)

*# Plot also the training points*

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap\_bold, edgecolor='k', s=20)

plt.xlim(xx.min(), xx.max())

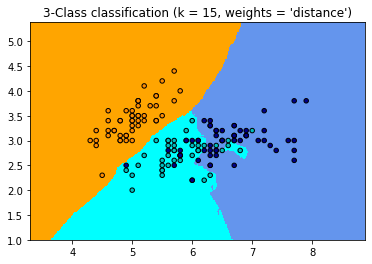
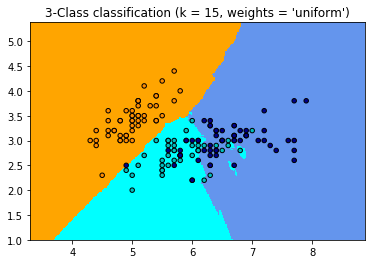
plt.ylim(yy.min(), yy.max())

plt.title("3-Class classification (k = **%i**, weights = '**%s**')"

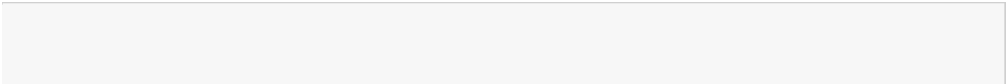
% (n\_neighbors, weights))

plt.show()

Automatically created module for IPython interactive environment



In [4]:



*#Change features from sepal length and width to petal length and width. Build the classifier again and discuss the output.*

print( doc )

# import numpy as np

**import matplotlib.pyplot as plt**

**from matplotlib.colors import** ListedColormap

**from sklearn import** neighbors, datasets n\_neighbors = 15

*# import some data to play with*

iris = datasets.load\_iris()

*# we only take the first two features. We could avoid this ugly # slicing by using a two-dim dataset*

X = iris.data[:, :2] y = iris.target

h = .02 *# step size in the mesh # Create color maps*

cmap\_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])

cmap\_bold = ListedColormap(['darkorange', 'c', 'darkblue'])

**for** weights **in** ['uniform', 'distance']:

*# we create an instance of Neighbours Classifier and fit the data.* clf = neighbors.KNeighborsClassifier(n\_neighbors, weights=weights) clf.fit(X, y)

*# Plot the decision boundary. For that, we will assign a color to each # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].*

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 2

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 2 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h)) Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

*# Put the result into a color plot* Z = Z.reshape(xx.shape) plt.figure()

plt.pcolormesh(xx, yy, Z, cmap=cmap\_light)

*# Plot also the training points*

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap\_bold, edgecolor='k', s=30)

plt.xlim(xx.min(), xx.max())

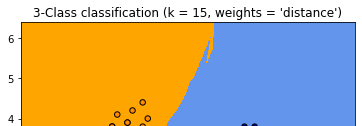
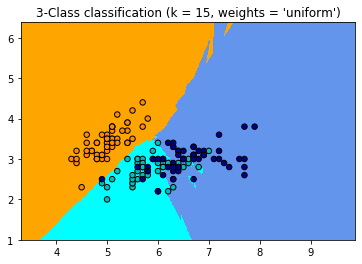
plt.ylim(yy.min(), yy.max())

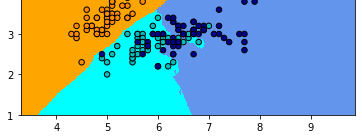
plt.title("3-Class classification (k = **%i**, weights = '**%s**')"

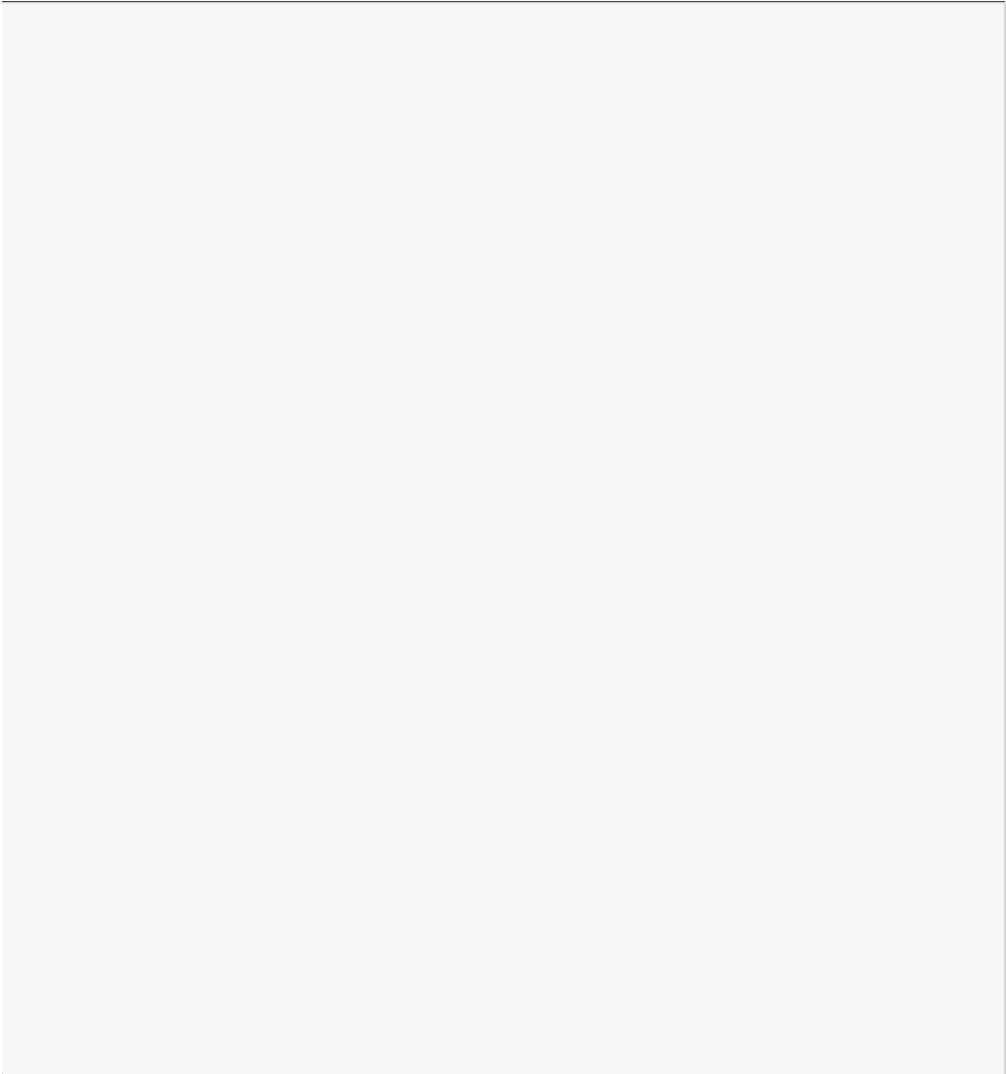
% (n\_neighbors, weights))

plt.show()

Automatically created module for IPython interactive environment





In [5]:

*#Consider all the features and build the classifier again and discuss the output.*

print( doc )

# import numpy as np

**import matplotlib.pyplot as plt**

**from matplotlib.colors import** ListedColormap

**from sklearn import** neighbors, datasets n\_neighbors = 15

*# import some data to play with*

iris = datasets.load\_iris()

*# we only take the first two features. We could avoid this ugly # slicing by using a two-dim dataset*

X = iris.data[:, :2] y = iris.target

h = .05 *# step size in the mesh # Create color maps*

cmap\_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])

cmap\_bold = ListedColormap(['darkgreen', 'black', 'red'])

**for** weights **in** ['uniform', 'distance']:

*# we create an instance of Neighbours Classifier and fit the data.* clf = neighbors.KNeighborsClassifier(n\_neighbors, weights=weights) clf.fit(X, y)

*# Plot the decision boundary. For that, we will assign a color to each # point in the mesh [x\_min, x\_max]x[y\_min, y\_max].*

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 5

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 5 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h)) Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

*# Put the result into a color plot* Z = Z.reshape(xx.shape) plt.figure()

plt.pcolormesh(xx, yy, Z, cmap=cmap\_light)

*# Plot also the training points*

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap\_bold, edgecolor='k', s=30)

plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max())

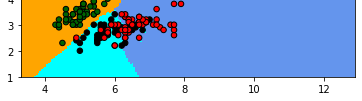
plt.title("3-Class classification (k = **%i**, weights = '**%s**')"

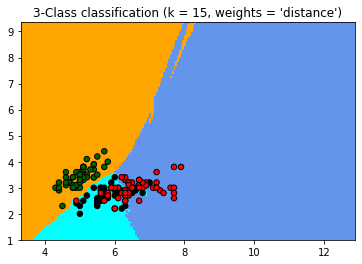
% (n\_neighbors, weights))

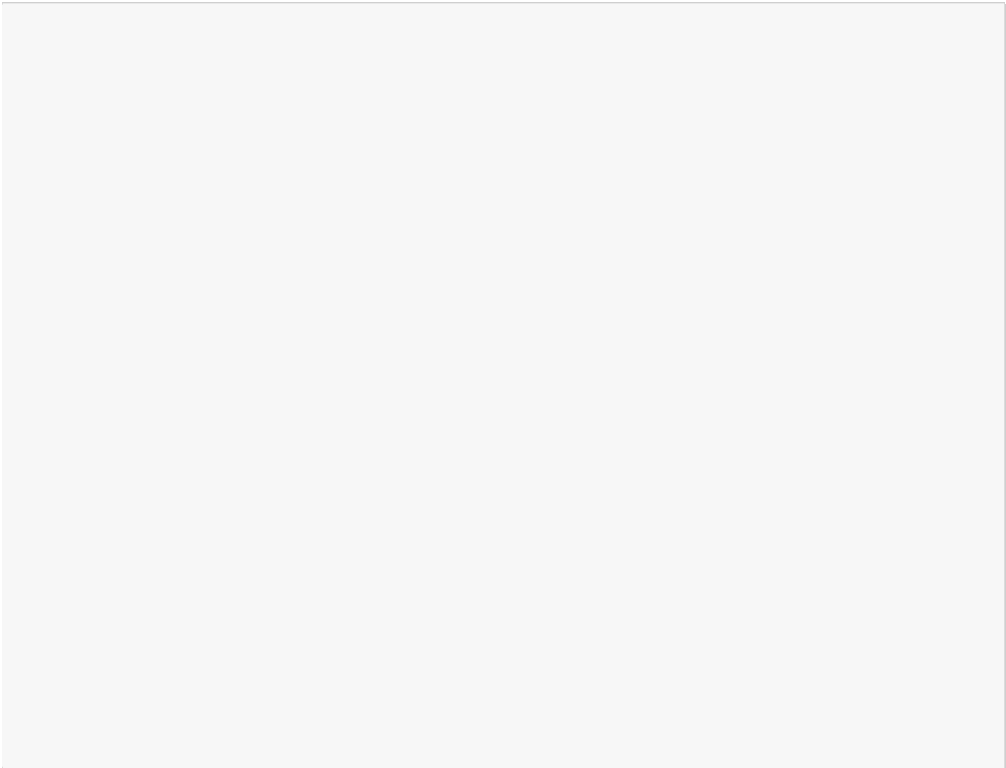
plt.show()

Automatically created module for IPython interactive environment







In [6]:

*#Demonstrate the resolution of a regression problem using a k-Nearest Neighbor and the interpolati on of the target*

print( doc )

*# Generate sample data*

# import numpy as np

**import matplotlib.pyplot as plt from sklearn import** neighbors

np.random.seed(0)

X = np.sort(5 \* np.random.rand(40, 1), axis=0) T = np.linspace(0, 5, 500)[:, np.newaxis]

y = np.sin(X).ravel()

*# Add noise to targets*

y[::5] += 1 \* (0.5 - np.random.rand(8))

*# #############################################################################*

*# Fit regression model*

n\_neighbors = 5

**for** i, weights **in** enumerate(['uniform', 'distance']):

knn = neighbors.KNeighborsRegressor(n\_neighbors, weights=weights) y\_ = knn.fit(X, y).predict(T)

plt.subplot(2, 1, i + 1)

plt.scatter(X, y, color='darkorange', label='data') plt.plot(T, y\_, color='navy', label='prediction') plt.axis('tight')

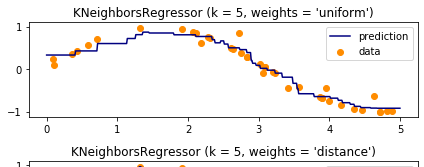
plt.legend()

plt.title("KNeighborsRegressor (k = **%i**, weights = '**%s**')" % (n\_neighbors,

weights))

plt.tight\_layout() plt.show()

Automatically created module for IPython interactive environment



prec! ie tion data

0

0 4