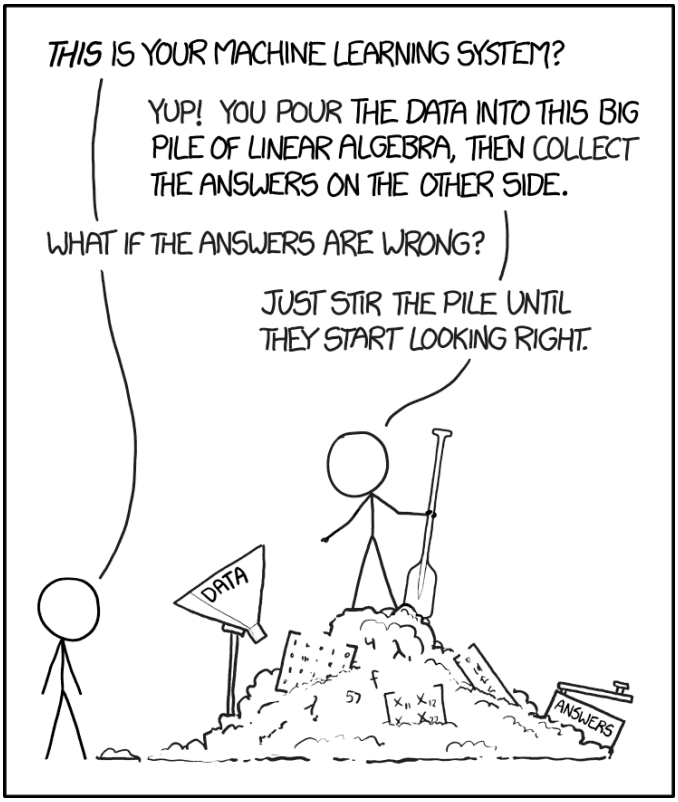
Prediction - Scikit-learn

Author List:

Sana Iqbal, significant edits for fall 2017

Kevin Li, Ikhlaq Sidhu, Spring 2017

Original Sources: http://scikit-learn.org,http://archive.ics.uci.edu/ml/datasets/Iris (http://scikit-learn.org,http://archive.ics.uci.edu/ml/datasets/Iris)



Our predictive machine learning models perform two types of tasks:

- **CLASSIFICATION**:

LABELS ARE DISCRETE VALUES.

Here the model is trained to classify each instance into a set of predefined discrete classes. On inputting a feature vector into the model, the trained model is able to predict a class of that instance.

Eg: We train our model using income and expenditure data of bank customers using **defaulter or non-defaulter** as labels. When we input income and expenditure data of any customer in this model, it will predict whether the customer is going to default or not.

- **REGRESSION**:

LABELS ARE CONTINUOUS VALUES.

Here the model is trained to predict a continuous value for each instance. On inputting a feature vector into the model, the trained model is able to predict a continuous value for that instance.

Eg: We train our model using income and expenditure data of bank customers using **default amount** as the label. This model when input with income and expenditure data of any customer will be able to predict the default amount the customer might end up with.

Machine learning model building steps

Understanding the data

Understand your dataset using graphs, descriptive analysis etc.

Cleaning the data

Removing outliers, Imputation of Null values, Removing null values

Building the model

Build using appropriate algorithms

Testing and iteration

Having testing outcomes and improve through each iteration

What is scikit-learn?

- Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python
- This library is built upon SciPy (Scientific Python)
- The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data
- We can do supervised and unsupervised learning with Scikit-learn

- **TO GET STARTED:**:

We will use python library -SCIKIT-LEARN for our classification and regression models.

1. Install numpy, scipy, scikit-learn.
2. Download the dataset provided and save it in your current working directory.
3. In the following sections you will:

   3.1 Read the dataset into the python program.

   3.2 Look into the dataset characteristics, check for feature type - categorical or numerical.

   3.3 Find feature distributions to check sufficiency of data.

   3.4 Divide the dataset into training and validation subsets.

   3.5 Fit models with training data using scikit-learn library.

   3.6 Calculate training error,this gives you the idea of bias in your model.

   3.7 Test model prediction accuracy using validation data,this gives you bias and variance error in the model.

   3.8 Report model performance on validation data using different metrics.

   3.9 Save the model parameters in a pickle file so that it can be used for test data.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

ASSUMPTIONS:

> Problem:

'Iris 'setosa' species has medicinal benefits and we want to make the process of identifying an iris species scalable'

> Type:

Classification

> Data :

Flower morphology data collected by floriculture department

Understanding the data

In [28]:
```
file_path='iris_classification.csv'
data=pd.read_csv(file_path)
```

In [32]:
```
# lets us look at the data
data.head(5)
```

Out[32]:

	sepal_length	sepal_width	species
0	5.1	3.5	setosa
1	5.2	4.1	setosa
2	6.4	3.2	versicolor
3	6.3	2.5	virginica
4	6.5	2.8	versicolor

In [30]:
```
# lets us check unique labels:
data['species'].value_counts()
```

Out[30]:
```
versicolor    50
virginica     50
setosa        50
Name: species, dtype: int64
```

In [31]:
```
from sklearn.utils import shuffle
# SHUFFLE data instances to randomize the distribution of different classes
# Check if data has any NAN  values, you can choose to drop NAN
# containing rows or replace NAN  values with mean. median,or any assumed value.

data= shuffle(data).reset_index(drop=True)
```

In [6]:
```
print('Number of NaNs in the dataframe:\n',data.isnull().sum())
data.head()
```

```
Number of NaNs in the dataframe:
 sepal_length    0
sepal_width     0
species         0
dtype: int64
```

Out[6]:

	sepal_length	sepal_width	species
0	4.9	2.4	versicolor
1	6.3	2.8	virginica
2	5.8	2.8	virginica
3	5.8	2.7	versicolor
4	4.5	2.3	setosa

In [33]:
```
# Our functions take in features and labels as arrays  so we need to separate them
# GET FEATURES X FROM THE DATA

X=data.iloc[:,:-1]
X.head()
```

Out[33]:

	sepal_length	sepal_width
0	5.1	3.5
1	5.2	4.1
2	6.4	3.2
3	6.3	2.5
4	6.5	2.8

In [34]:
```
# GET LABELS Y FROM THE DATA
Y=data['species']
print (Y.value_counts()) #gives the count of each label in the dataset
```

```
versicolor    50
virginica     50
setosa        50
Name: species, dtype: int64
```

In [35]:
```
print ('''\nWe will map our class labels to integers and then use in modeling.
The mapping is:'versicolor': 0, 'virginica': 1,'setosa' :2 \n''')

Y=Y.map({'versicolor': 0, 'virginica': 1,'setosa' :2})
print (Y.value_counts()) #gives the count of each label in the dataset

Y.head()
```

```
We will map our class labels to integers and then use in modeling.
The mapping is:'versicolor': 0, 'virginica': 1,'setosa' :2

2    50
1    50
0    50
Name: species, dtype: int64
```

Out[35]:
```
0    2
1    2
2    0
3    1
4    0
Name: species, dtype: int64
```

In [36]:
```
# should do sanity check on data often
print("Feature vector shape=", X.shape)
print("Class shape=", Y.shape)
```

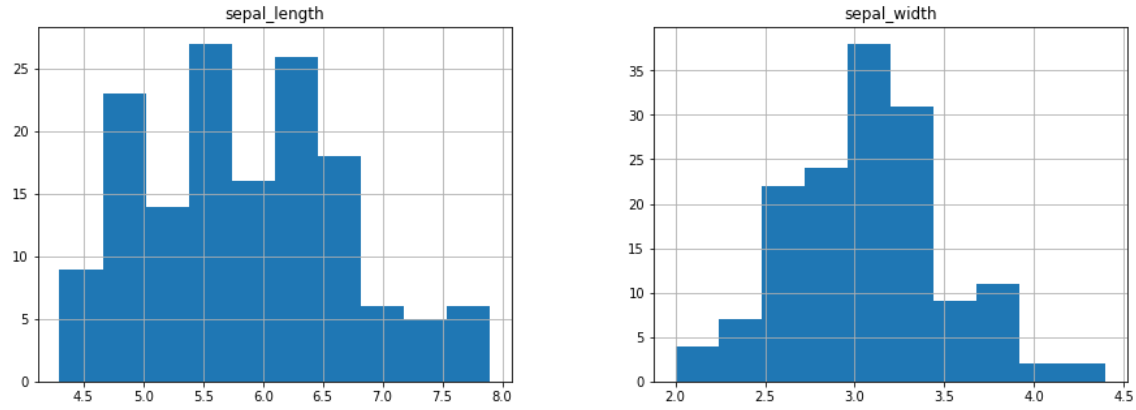
```
Feature vector shape= (150, 2)
Class shape= (150,)
```

In [11]:
```
# More summary about our data
data.describe()
```

Out[11]:

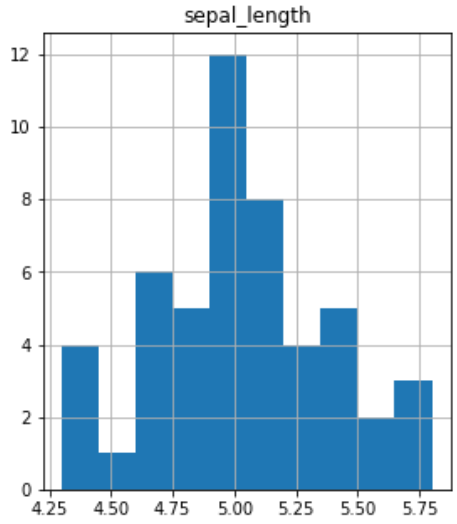
	sepal_length	sepal_width
count	150.000000	150.000000
mean	5.843333	3.054000
std	0.828066	0.433594
min	4.300000	2.000000
25%	5.100000	2.800000
50%	5.800000	3.000000
75%	6.400000	3.300000
max	7.900000	4.400000

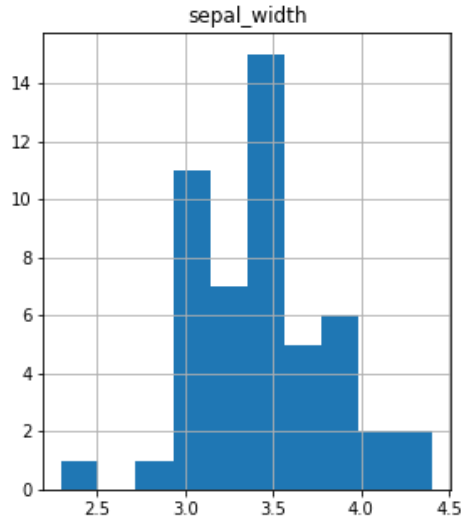
In [12]:
```
# Get feature distribution of each continuous valued feature (sepal_length and sepal_width)
data.hist(figsize=(15,5))
plt.show()
```

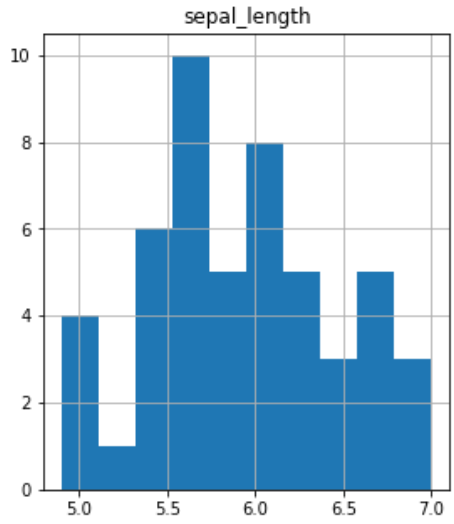


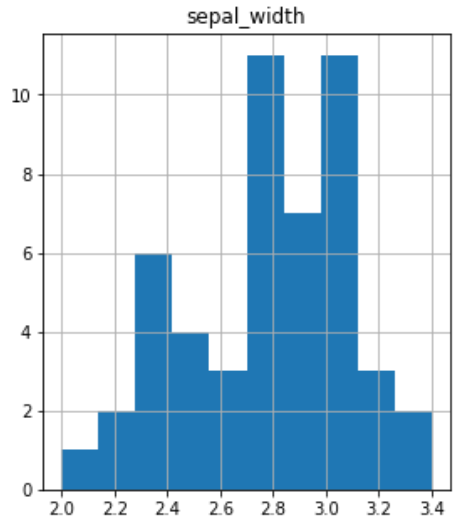
In [13]:
```
# Check feature distribution of each class, to get an overview of feature and clas
s relationshhip,
# also useful in validating data
print(data.groupby('species').count())


data.groupby('species').hist(figsize=(10,5))
plt.show()
```

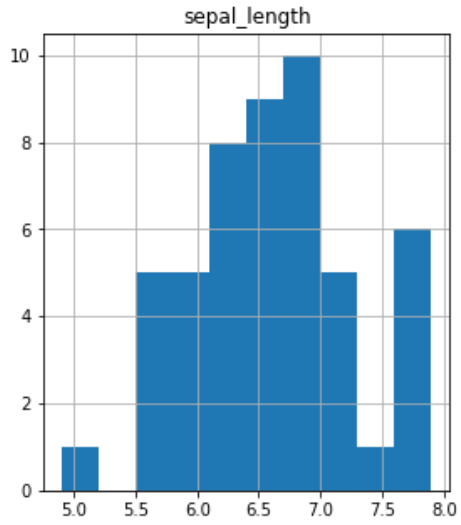
species	sepal_length	sepal_width
setosa	50	50
versicolor	50	50
virginica	50	50

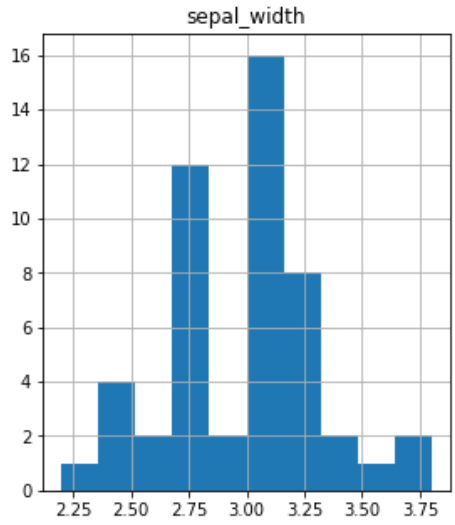












Cleaning the model

We do not have any NA values in the dataset

- Use Pandas functions to remove na values, **Sklearn.preprocessing.Imputer** to impute the missing values.
- Scaling the variables using **sklearn.preprocessing.StandardScaler**
  - The standard score of a sample x is calculated as:

    z = (x - u) / s
- Scaling using Sklearn Min Max scaler
  - X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

    X_scaled = X_std * (max - min) + min

Building the Model

USE SKLEARN INBUILT FUNCTION TO BUILD A LOGISTIC REGRESSION MODEL

For Details check : http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression.fit (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression.fit)

Our Hypothesis : Species of Iris is dependent on sepal length and width of the flower. In order to check the validity of our trained model, we keep a part of our dataset hidden from the model during training, called **Test data**.

Test data labels are predicted using the trained model and compared with the actual labels of the data.This gives us the idea about how well the model can be trusted for its predictive power.

We split our data into train/test -

- **Training set** : The sample of data used to fit your model.
- **Test set** : The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

- **Validation set**: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters

---

K-folds Cross Validation

Also, if our data set is small we will have fewer examples for validation. This will not give us a a good estimatiion of model error. We can use k-fold crossvalidation in such situations. In k-fold cross-validation, the shuffled training data is partitioned into k disjoint sets and the model is trained on k –1 sets and validated on the kth set. This process is repeated k times with each set chosen as the validation set once. The cross-validation accuracy is reported as the average accuracy of the k iterations

In [14]:
```
# Split data into training and test set using sklearn function

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_st
ate=100)
print ('Number of samples in training data:',len(x_train))
print ('Number of samples in test data:',len(x_test))
```

```
Number of samples in training data: 120
Number of samples in test data: 30
```

Train our Logistic Regression Model

In [15]:
```
from sklearn import linear_model

# Name our logistic regression object
LogisticRegressionModel = linear_model.LogisticRegression()

# we create an instance of logistic Regression Classifier and fit the data.
print ('Training a logistic Regression Model..')
LogisticRegressionModel.fit(x_train, y_train)
```

```
Training a logistic Regression Model..
```

Out[15]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Test -----Training Accuracy and Test Accuracy

TRAINING ACCURACY

In [16]:
```
# TRAINING ACCURACY

training_accuracy=LogisticRegressionModel.score(x_train,y_train)
print ('Training Accuracy:',training_accuracy)
```

```
Training Accuracy: 0.7583333333333333
```

In [17]:
```
# Lets us see how it is done ab-initio. Estimate prediction error logically for ea
ch sample
# and save it in an array called Loss_Array


# this line below will predict a category for every row in x_train
predicted_label = LogisticRegressionModel.predict(x_train)



def find_error(actual_label,predicted_label):
    '''actual_label= label in data
        predicted_label= label predicted by the model

    '''

    Loss_Array = np.zeros(len(actual_label)) #create an empty array to store loss
values
#     print(Loss_Array)

    for i,value in enumerate(actual_label):

        if value == predicted_label[i]:
            Loss_Array[i] = 0
        else:
            Loss_Array[i] = 1

    print ("Y-actualLabel   Z-predictedLabel   Error \n")
    for i,value in enumerate(actual_label):
        print (value,"\t\t" ,predicted_label[i],"\t\t",Loss_Array[i])

    error_rate=np.average(Loss_Array)
    print ("\nThe error rate is ", error_rate)
    print ('\nThe accuracy of the model is ',1-error_rate )


find_error(y_train,predicted_label)
```

Y-actualLabel	Z-predictedLabel	Error
0	1	1.0
1	1	0.0
0	1	1.0
0	1	1.0
0	1	1.0
1	1	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0
2	2	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
2	2	0.0
0	1	1.0
0	2	1.0
1	1	0.0
1	1	0.0
1	2	1.0
1	1	0.0
1	1	0.0
1	1	0.0
2	2	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	1	1.0
1	1	0.0
2	2	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
2	2	0.0
0	1	1.0
0	1	1.0
2	2	0.0
1	1	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	1	1.0
1	1	0.0
2	2	0.0
0	0	0.0
0	1	1.0
0	1	1.0
0	0	0.0
2	2	0.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
2	2	0.0
0	1	1.0
2	2	0.0
1	1	0.0
0	2	1.0
1	1	0.0
1	1	0.0

Testing the model

In [37]:
```
# TEST ACCURACY:
# we will find accuracy of the model
# using data that was not used for training the model

test_accuracy=LogisticRegressionModel.score(x_test,y_test)
print('Accuracy of the model on unseen test data: ',test_accuracy)
```

```
Accuracy of the model on unseen test data:  0.8
```

Accuracy can be misleading!

Other measures of performance - Confusion matrix, Precision, Recall

Confusion matrix

What is a confusion matrix?

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known.

Precision

Precision (P) is defined as the number of true positives (T_p) over the number of true positives plus the number of false positives (F_p)

Recall

Recall (R) is defined as the number of true positives (T_p) over the number of true positives plus the number of false negatives (F_n)

In [20]:
```
from sklearn.metrics import confusion_matrix
y_true = y_test
y_pred = LogisticRegressionModel.predict(x_test)
ConfusionMatrix=pd.DataFrame(confusion_matrix(y_true, y_pred),columns=['Predicted
0','Predicted 1','Predicted 2'],index=['Actual 0','Actual 1','Actual 2'])
print ('Confusion matrix of test data is: \n',ConfusionMatrix)
```

```
Confusion matrix of test data is:
           Predicted 0  Predicted 1  Predicted 2
Actual 0            5            6            0
Actual 1            0           11            0
Actual 2            0            0            8
```

In [21]:
```
from sklearn.metrics import precision_score
print("Average precision for the 3 classes is - ", precision_score(y_true, y_pred,
average = None) )
```

```
Average precision for the 3 classes is -  [1.         0.64705882 1.        ]
```

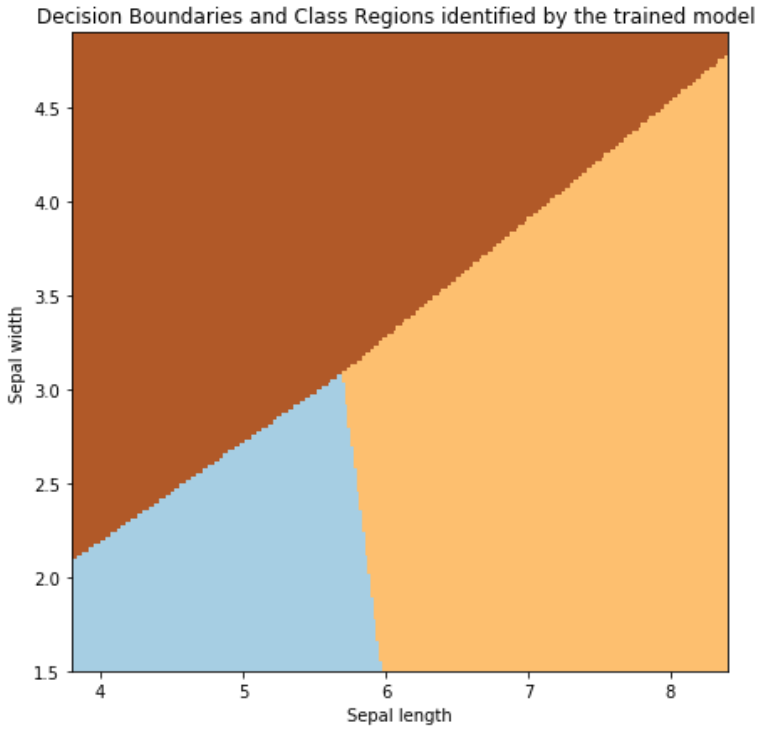
In [22]:
```
from sklearn.metrics import recall_score
print("Average recall for the 3 classes is - ", recall_score(y_true, y_pred, avera
ge = None) )
```

```
Average recall for the 3 classes is -  [0.45454545 1.         1.        ]
```

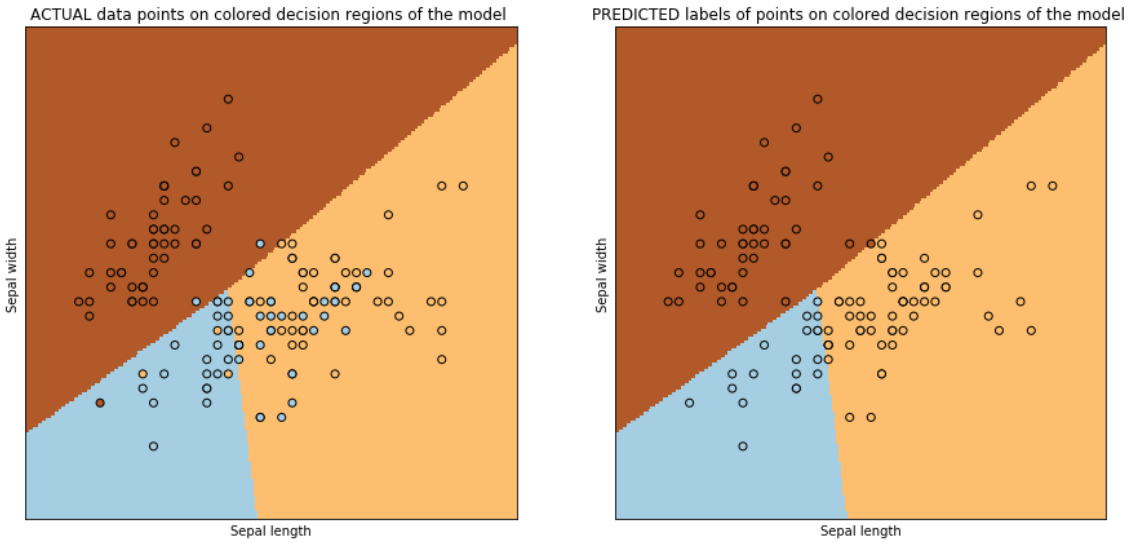
In [38]:
```
# PLOT THE DECISION BOUNDARIES:
# 1.create meshgrid of all points between

'''
For that we will create a mesh between [x_min, x_max]x[y_min, y_max].
We will choose a 2d vector space ranging from values +- 0.5 from our
min and max values of sepal_length and sepal_width.
Then we will divide that whole region in a grid of  0.02 units cell size.

'''

h = 0.02  # step size in the mesh
x_min = X['sepal_length'].min() - .5
x_max = X['sepal_length'].max() + .5
y_min = X['sepal_width'].min() - .5
y_max = X['sepal_width'].max() + .5



# print x_min, x_max, y_min, y_max

sepal_length_range = np.arange(x_min, x_max, h)
sepal_width_range = np.arange(y_min, y_max, h)



# Create datapoints for the mesh
sepal_length_values, sepal_width_values = np.meshgrid(sepal_length_range, sepal_wi
dth_range)


# Predict species for the fictious data in meshgrid
predicted_species = LogisticRegressionModel.predict(np.c_[sepal_length_values.rave
l(), sepal_width_values.ravel()])


print(sepal_length_range)
print ('Finished predicting species')

# another approach is to make an array Z2 which has all the predicted values in (x
r,yr).

predicted_species2= np.arange(len(sepal_length_range)*len(sepal_width_range)).resh
ape(len(sepal_length_range),len(sepal_width_range))
for yni in range(len(sepal_width_range)):
    for xni in range(len(sepal_length_range)):
#         print (xni, yni, LogisticRegressionModel.predict([[xr[xni],yr[yni]]]))

        predicted_species2[xni,yni] =LogisticRegressionModel.predict([[sepal_lengt
h_range[xni],sepal_width_range[yni]]])
print ('Finished predicted_species2')
```

```
[3.8  3.82 3.84 3.86 3.88 3.9  3.92 3.94 3.96 3.98 4.   4.02 4.04 4.06
 4.08 4.1  4.12 4.14 4.16 4.18 4.2  4.22 4.24 4.26 4.28 4.3  4.32 4.34
 4.36 4.38 4.4  4.42 4.44 4.46 4.48 4.5  4.52 4.54 4.56 4.58 4.6  4.62
 4.64 4.66 4.68 4.7  4.72 4.74 4.76 4.78 4.8  4.82 4.84 4.86 4.88 4.9
 4.92 4.94 4.96 4.98 5.   5.02 5.04 5.06 5.08 5.1  5.12 5.14 5.16 5.18
 5.2  5.22 5.24 5.26 5.28 5.3  5.32 5.34 5.36 5.38 5.4  5.42 5.44 5.46
 5.48 5.5  5.52 5.54 5.56 5.58 5.6  5.62 5.64 5.66 5.68 5.7  5.72 5.74
 5.76 5.78 5.8  5.82 5.84 5.86 5.88 5.9  5.92 5.94 5.96 5.98 6.   6.02
 6.04 6.06 6.08 6.1  6.12 6.14 6.16 6.18 6.2  6.22 6.24 6.26 6.28 6.3
 6.32 6.34 6.36 6.38 6.4  6.42 6.44 6.46 6.48 6.5  6.52 6.54 6.56 6.58
 6.6  6.62 6.64 6.66 6.68 6.7  6.72 6.74 6.76 6.78 6.8  6.82 6.84 6.86
 6.88 6.9  6.92 6.94 6.96 6.98 7.   7.02 7.04 7.06 7.08 7.1  7.12 7.14
 7.16 7.18 7.2  7.22 7.24 7.26 7.28 7.3  7.32 7.34 7.36 7.38 7.4  7.42
 7.44 7.46 7.48 7.5  7.52 7.54 7.56 7.58 7.6  7.62 7.64 7.66 7.68 7.7
 7.72 7.74 7.76 7.78 7.8  7.82 7.84 7.86 7.88 7.9  7.92 7.94 7.96 7.98
 8.   8.02 8.04 8.06 8.08 8.1  8.12 8.14 8.16 8.18 8.2  8.22 8.24 8.26
 8.28 8.3  8.32 8.34 8.36 8.38 8.4 ]
Finished predicting species
Finished predicted_species2
```

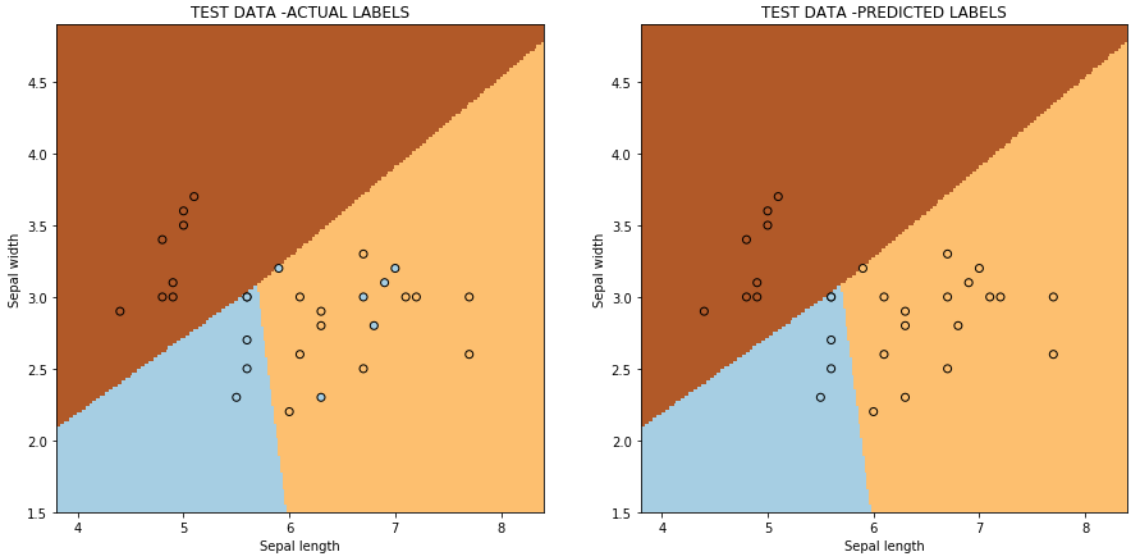
In [24]:
```
# Put the result into a color plot
predicted_species = predicted_species.reshape(sepal_length_values.shape)
plt.figure(figsize=(7,7))
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.pcolormesh(sepal_length_values,sepal_width_values,predicted_species , cmap=plt
.cm.Paired)
plt.title('Decision Boundaries and Class Regions identified by the trained model '
)
#plt.colorbar()
plt.show()
```



In [25]:
```
# Plot also the training points

fig, ax = plt.subplots(nrows=1,ncols=2,figsize=(15,7))

plt.subplot(1,2,1)
plt.pcolormesh(sepal_length_values,sepal_width_values,predicted_species , cmap=plt
.cm.Paired)

plt.scatter(X['sepal_length'], X['sepal_width'], c=Y, edgecolors='k', cmap=plt.cm.
Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
#plt.colorbar()
plt.xlim(sepal_length_values.min(),sepal_length_values.max())
plt.ylim(sepal_width_values.min(), sepal_width_values.max())
plt.xticks(())
plt.yticks(())
plt.title('ACTUAL data points on colored decision regions of the model ')

# Put the result into a color plot of decison boundary

plt.subplot(1,2,2)
plt.title('PREDICTED labels of points on colored decision regions of the model ')
plt.pcolormesh(sepal_length_values,sepal_width_values,predicted_species, cmap=plt.
cm.Paired)
#plt.colorbar()
label=np.unique(y_test)

plt.scatter(x_train['sepal_length'], x_train['sepal_width'], c=predicted_label, ed
gecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(sepal_length_values.min(),sepal_length_values.max())
plt.ylim(sepal_width_values.min(), sepal_width_values.max())
plt.xticks(())
plt.yticks(())



plt.show()
```



In [26]:
```
fig, ax = plt.subplots(nrows=1,ncols=2,figsize=(15,7))

plt.subplot(1,2,1)
plt.pcolormesh(sepal_length_values,sepal_width_values,predicted_species, cmap=plt.
cm.Paired)
#plt.colorbar()
label=np.unique(y_test)
plt.title('TEST DATA -ACTUAL LABELS')
# Plot also the training points
plt.scatter(x_test['sepal_length'], x_test['sepal_width'], c=y_test,label=np.uniqu
e(y_test), edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')


plt.subplot(1,2,2)
plt.pcolormesh(sepal_length_values,sepal_width_values,predicted_species, cmap=plt.
cm.Paired)
#plt.colorbar()
# Plot also the training points
plt.scatter(x_test['sepal_length'], x_test['sepal_width'], c=LogisticRegressionMod
el.predict(x_test), edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('TEST DATA -PREDICTED LABELS')


plt.show()
```



Simple Linear Regression Example

Linear regression is a predictive modeling technique for predicting a numeric response variable based on features.
"Linear" in the name linear regression refers to the fact that this method fits a model where response bears linear relationship with features. (ie Z is proportional to first power of x)

Z = X0 + a(X1) + b(X2) +.... where:
Z: predicted response
X0: intercept
a,b,..: Coefficients of X1,X2..

If Y is the actual response and Z is the predicted response,
Y-Z= Residual
Average Residual defines model performance,residual equal to zero represents a perfect fit model.

In [27]:
```
'''Source: Scikit learn
Code source: Jaques Grobler
License: BSD 3 clause'''
from sklearn import linear_model

example_dff = pd.DataFrame(np.random.randint(0,100,size=(100, 1)),columns=['X'])
example_dff['C']=5.1*example_dff['X']
# example_dff['C']=5.1*example_dff['X']**2



# FEATURES
X_reg=example_dff[['X']]

# Y
Y_reg=example_dff['C']

# Create linear regression object
LinearRegressionModel= linear_model.LinearRegression()

# Train the model using the training sets
LinearRegressionModel.fit(X_reg, Y_reg)
Z_reg=LinearRegressionModel.predict(X_reg)

# The coefficients
print('Coefficients:', LinearRegressionModel.coef_)
# The mean squared error
print("Mean squared error:",np.mean((Z_reg - Y_reg) ** 2))

# Plot outputs
plt.scatter(X_reg['X'], Y_reg,  color='red')
plt.plot(X_reg['X'], Z_reg, color='blue',
         linewidth=3)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression using data with one feature -X')
plt.xticks(())
plt.yticks(())

plt.show()
```

```
Coefficients: [5.1]
Mean squared error: 6.611167145340472e-27
```