Lesson 12 Linear Regression Last Update Apr 18, 2021

Regression problems are supervised learning problems in which the response is continuous.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). The variable y is dependent on the independent variable x. This means if you change x then y will change, Changing the value of y has no effect on x, therefore y is dependent on the independent variable x.

Linear Regression is a linear relationship between x (input) and y(output). If we plot the independent variable (x) on the x-axis and dependent variable (y) on the y-axis, linear regression gives us a straight line that best fits the data points using the line equation. (note: linear means a straight line)

y= mx + b

Where **b** is the **y-intercept** and **m** is the **slope** of the line.

The y **-intercept** of a graph is the point where the graph crosses the *y* –axis when x = 0

The slope m of a line is $\frac{y2 - y1}{x2 - x1} = \frac{1 - (-2)}{4 - 0} = \frac{3}{4}$

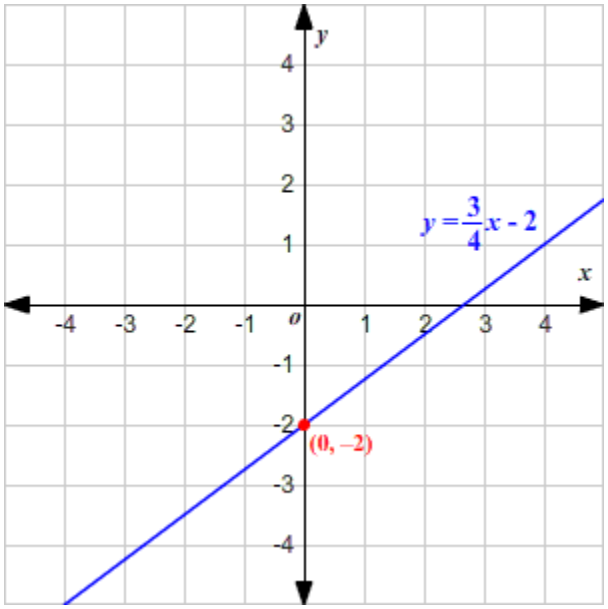
In the following graph m = 3/4 of has its *y* -intercept at −2 .

The equation of the line is y = mx + b = 3/4 x – 2

Using the equation of the line for any point x then y can be calculated.

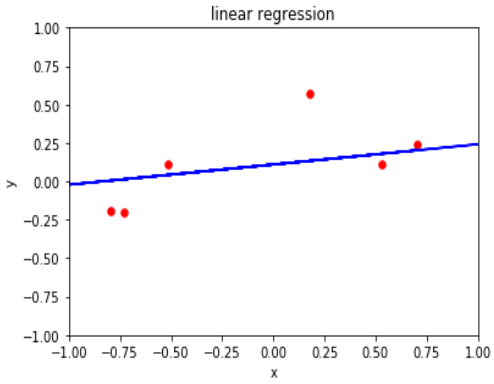
Example:

if x = 4 then y = mx + b = 3/4 (4) – 2 = 3 – 2 = 1



The linear regression algorithm gives us the most optimal value for the y-intercept and the slope m. The y and x variables are the data features and are fixed values that are not to be changed. The values that we can control are the intercept(b) and slope(m). There can be multiple straight lines depending upon the values of intercept and slope. Basically what the linear regression algorithm does, is fit multiple lines on the data points and returns the line that results in the least error.

Linear regression is about drawing a straight line through as set of related x and y points.



We will make an array of n random points then draw a straight line through them using the Equation for a line:

y = m x + b

m = slope of line
b = the y intercept of the line

where m is calculated as follows:

$$m = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum x y)}{n(\sum x^2) - (\sum x)^2}$$

b is calculated as follows:

$$b = \frac{n(\sum x y) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

We make random data accordingly to the normal distribution using the numby **normal** function.

using.numpy.normal(mean, std, size)

To use numpy functions you need to import the numpy module

import numpy as np

We make n number of x points with mean 0 and std of 1

```
n = 10
x = np.random.normal(0,1,n)
print(x)
```

```
[-1.03957644 -0.02273678 -1.1722456  -0.99033947 -0.07772752 -0.57631302
  0.27126607  0.20078075  1.2738824   0.67596517]
```

We make n number of y points with mean 0 abs std of .25

```
y = np.random.normal(0,.25,n)
print(y)
```

```
[-0.19086382 -0.15621885  0.10118205  0.19180085 -0.18573354 -0.34741558
  0.62865553  0.14470035  0.36359903  0.01695659]
```

Next we calculate the slope **m** and intercept **b** from the above linear regression equations:

$$m = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2}$$

$$b = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

We then predict the y values from the equation:

ypred = m*x + b

Note: **y** and **ypred** are different values. **y** is the **original y** value for x, **ypred** is the **predicted y** value for the corresponding x values. The ypred value forms a straight line when plotted with x. The ypred values are the calculated values using the equation for the line from the calculated **m** slope a **b** y-intercept.

Before we plot the data we set the bounds of our graph to -1 and 1 for both x and y axis

```
plt.axis([-1, 1, -1, 1])
```

We use **matplot** to plot the test data as a scatter plot

plt.scatter(x, y, color = "r", marker = "o", s = 30)

and then plot the regression line using matplot **plot** function.

plt.plot(x, ypred, color = "b")

Where **x** is random numbers between -1 to 1 and **ypred** is the calculated predicted y value for each random x value.

You need to import matplot before you can use it

import matplotlib.pyplot as plt

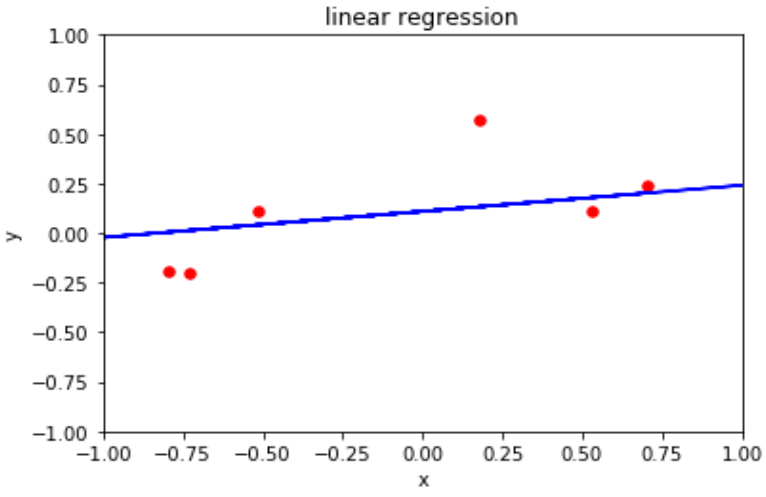
We can plot the points and the linear regression line from the calculated slope m and intercept b using the equation of the line ypred = mx + b

Linear Regression model calculates results for m and b:

m = 0.13159616440312247

b = 0.10946191382798436

Here is the x and y points and linear regression line ypred = mx + b plotted on the graph. The graph shows a straight line because we are plotting the ypred values for each corresponding x value, using the y=mx+b line equation where the m and b values have been **fitted** with the x and y original values.



Here is the complete program

```
"""
linearRegression.py
"""
import numpy as np
import matplotlib.pyplot as plt

print('Linear Regression')
print('y = mx + b')

n = 10  # number of points

# make random  normal distribution x,y points
# mean , std, size
x = np.random.normal(0,1,n)
y = np.random.normal(0,.25,n)

# ypred = mx + b

# calculate m
#     ( ∑ y )( ∑ x^2) – (∑ x) (∑ x y )
# m = ----------------------------
#          n ( ∑ x^2) - ( ∑ x)^2

top = (np.sum(y)*np.sum(x*x) - np.sum(x) * np.sum(x*y))
bottom = (n * np.sum(x*x) - np.sum(x) * np.sum(x))
m = top / bottom
print('m = ',m)

# calculate b
#        n   ( ∑ x y )   – (∑ x) (∑ y )
# b = ----------------------------------------
#        n ( ∑ x^2) - ( ∑ x)^2

top = ( n * np.sum(x*y)  -  np.sum(x) * np.sum(y))
bottom = (n * np.sum(x*x) - np.sum(x) * np.sum(x))
b = top / bottom
print('b = ',b)

# calculate  y2 = mx + b
ypred = m*x + b
```

```
# plot regression line

# plotting x and y points as a scatter plot
plt.scatter(x, y, color = "r", marker = "o", s = 30)

# set x-y axis dimensions
plt.axis([-1, 1, -1, 1])

# plotting the regression line
plt.plot(x, ypred, color = "b")

# set title
plt.title('linear regression')

# set x,y labels
plt.xlabel('x')
plt.ylabel('y')

# show plot
plt.show()
```

Todo

Try different values of **mean**, **std** and **n size** in the following normal distribution for x and y

```
x = np.random.normal(0,1,n)    # where 0 = mean, 1 = stn and n = size
y = np.random.normal(0,.25,n)    # where 0 = mean, .25 = std and n = size
```

LEAST SQUARES

Here is a more simplified linear regression using the least squares method.

y =mx + b

Let **X** be the independent variable and **Y** be the dependent variable

Calculate slope m:

$$m = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

where :

xi is the independent variable data set
yi is the dependent variable data set

$\sum (x_i - \bar{x})(y_i - \bar{y})$ is the sum of the data set subtracting each mean

$\sum (x_i - \bar{x})^2$ is the sum of errors squares

Calculate b from mean y, mean x and slope m:

$$b = \bar{y} - m * \bar{x}$$

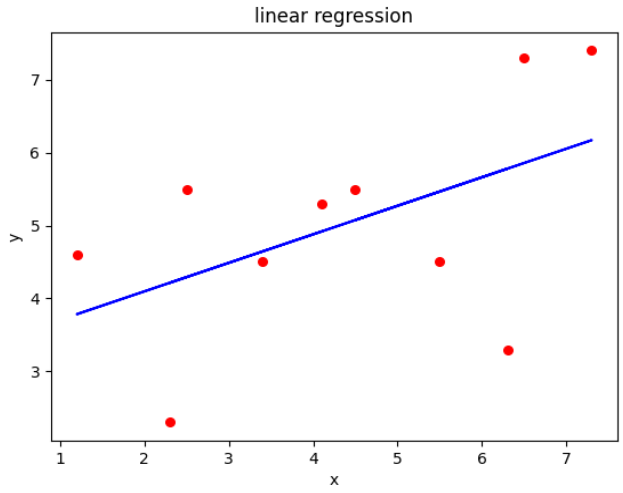
Calculate the y prediction y_pred

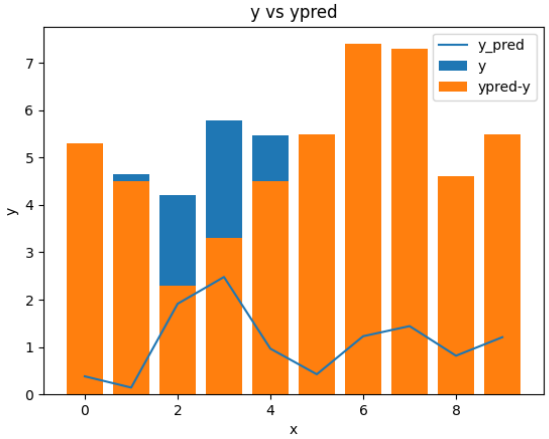
$$Y\_pred = m * x + b$$

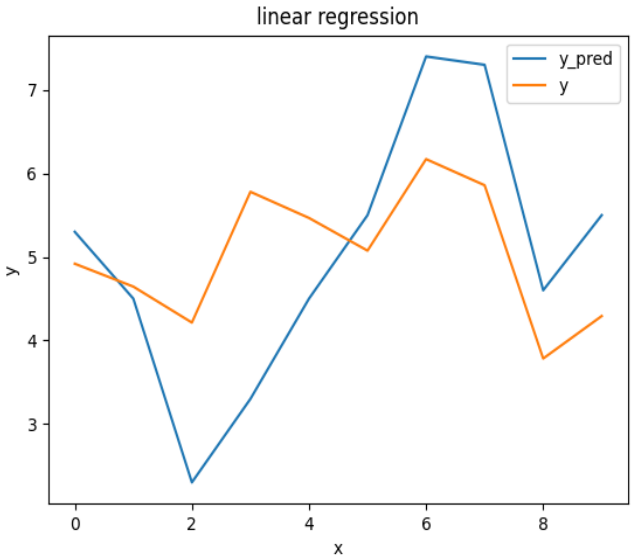
Linear Regression Homework Question 1

Write python program using the Least Squares algorithm and plot the results using matplot. Try large data set's with different std's You may want to make a LeastSquares class that would store the m and b value and has a **fit** function that receives a list of x and y values to fit the data and a **predict** function that will return a predicted value for a x value or return a list of predicted y values for a list of x values. Plot a scatter plot, barchart or histogram of y, y_pred and include a plot of abs(y-y_pred). Plot the y and y prediction values on the same bar chart (stacked)

You should get something like this:







LEAST SQUARES FITTING USING SCIPY

Scipy (Scientific python) is a library package has many machine learning algorithms for you and operates on numpy arrays.

Scipy has the **curve_fit** function that takes a function **f** formula and data set x and y and uses the least squares algorithm to fit the data using the specified function **f**.

Scipy uses the non-linear least squares algorithm to fit a function, f, to data.

Assumes: **ydata = f(xdata, *params) + eps**

Where **xdata** is a list of x values

*params are a list of received argument values in our case will be the **m** and **b**

Our **f** function just returns the value **y** for the equation of line for the values **m** and **b** received through ***params**. * is a unpacking operator converting the argument m,b into separate values m and b so that you can use then in the function separately.

```
def  f(x, m, b):
      return m*x + b
```

eps is the smallest representable positive number such that 1.0 + eps != 1.0. (usually about .000001)

The Scipy **curve_fit** function returns 2 arrays **popt** and **pcov**

Results	**Description**
pop a 1d array	Optimal values for the parameters so that the sum of the squared residuals of f(xdata, *popt) - ydata is minimized
pcov a 2d array	The estimated covariance of popt. The diagonals provide the variance of the parameter estimate.

We call curve_fit like this:

pop, pcov = curve_fit(f, x, y)

where f is the function that returns y = mx + b

```
def  f(x, m, b):
        return m*x + b
```

We obtain slope **m** from **pop** and y-intercept **b** from **pcov**

m = pop[0]

b = pcov[0,1]

we print out the obtained values m and b

m = 0.02414388106444587 b = 0.00024679354443633

We calculate predicted y from line equation ypred = mx + b with m and b values

ypred = m*x + b

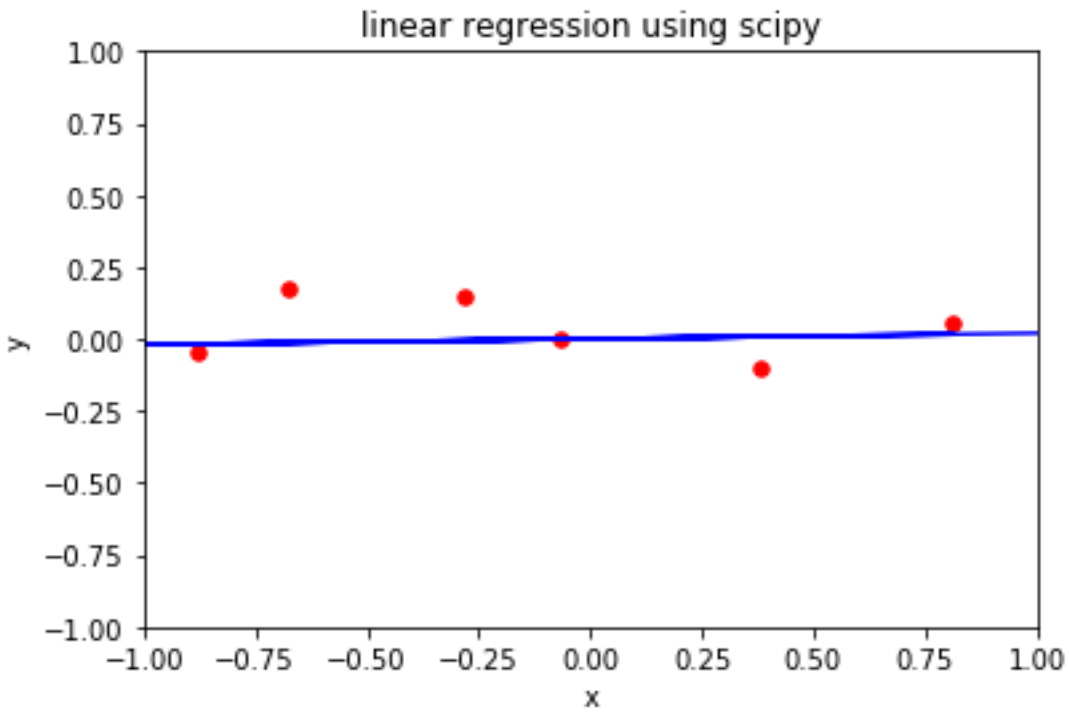
Lastly we plot the test data

plt.scatter(x, y, color = "r", marker = "o", s = 30)

we plot the regression line

plt.plot(x, ypred, color = "b")

our plot is as follows:



Here is the complete program:

```
"""
linearRegScipi.py
"""
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

print('Linear Regression using scipy')
print('y = mx + b')
n = 10  # number of points
# make random  normal distribution x,y points
# mean , std, size
x = np.random.normal(0,1,n)
y = np.random.normal(0,.25,n)
# straight line function y = f(x)
def  f(x, m, b):
     return m*x + b
# ypred = mx + b
pop, pcov= curve_fit(f, x, y) # your data x, y to fit
m = pop[0]
b = pcov[0,1]
print("m = ",m, "b = ", b)
```

```
# calculate  ypred = mx + b
ypred = m*x + b
# plot regression line
# plotting as scatter plot
plt.scatter(x, y, color = "r", marker = "o", s = 30)
# axis dimensions
plt.axis([-1, 1, -1, 1])
# plotting the regression line
plt.plot(x, ypred, color = "b")
# title
plt.title('linear regression using scipy')
# x,y labels
plt.xlabel('x')
plt.ylabel('y')
# show plot
plt.show()
```

Todo

Try different values of **mean**, **std** and **n size** in the following normal distribution for x and y

x = np.random.normal(0,1,n) # where 0 = mean, 1 = stn and n = size

y = np.random.normal(0,.25,n) # where 0 = mean, .25 = std and n = size

Linear regression using SkLearn

Sklearn also known as (SciKit) is a machine learning library for Python. It features several regression, classification and clustering algorithms including SVMs, gradient boosting, k-means and random forests. It is designed to work with Python Numpy and Scipy.

You can install sklearn onto python as follows:

pip install sklearn

We will use **sklearn** to do linear regression on a random data set.

You first need to import the required sklearn libraries.

```
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

Skilearn has the **make_regression** function to make test data for us with a specified injected noise. We will have lots of noise.

```
x, y = datasets.make_regression(n_samples=n, n_features=1, noise=10)
```

For accurate testing we split up out test data into test data and training data sets.

We split 80% of the data to the training set while 20% of the data to test set using the sklearn **train_test_split** function using **test_size** parameter.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,    random_state=0)
```

Once we split the real data into train and test set's we make a linear model

```
reg = LinearRegression()
```

From the linear regression mode we fit the data using the training set x_train and y_train using the sklearn **fit** function

```
reg.fit(x_train, y_train)
```

We train the data using the x_train and y_train data to calculate the m slope and b y-intercept.

After the data is fitted we can retrieve the slope m and y intercept b

```
m = reg.coef_
b = reg.intercept_
```

We then print **m** and **b** out

```
m =  14.795286457868691 b =  0.5981177409678305
```

We can then predict the points from the test data using the sklearn **predict function** and the **x_test** data. (predict always expects a 2 dimensional array)

y_pred = reg.predict(x_test)

We use the x_test data to calculate the y predictions. The slope m and b y-intercept has been previously calculated in the **fit** function using the x_train and y_train data sets.

We then we use matplot to plot the results

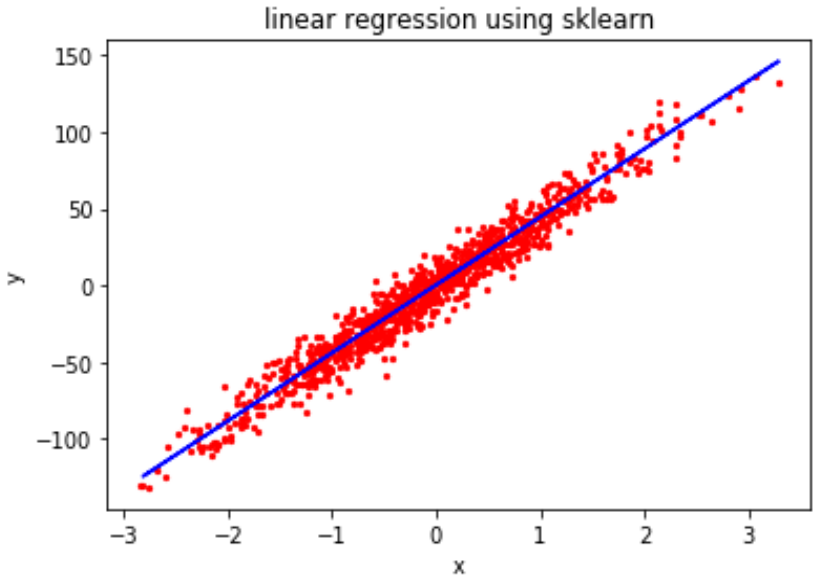
We first plot the complete x and y data as a scatter plot

plt.scatter(x , y , color = "r", marker = "o", s = 5)

then plot the regression line using the **y_test** data x_test for the x axis and the predicted y data **y_pred** for the y axis

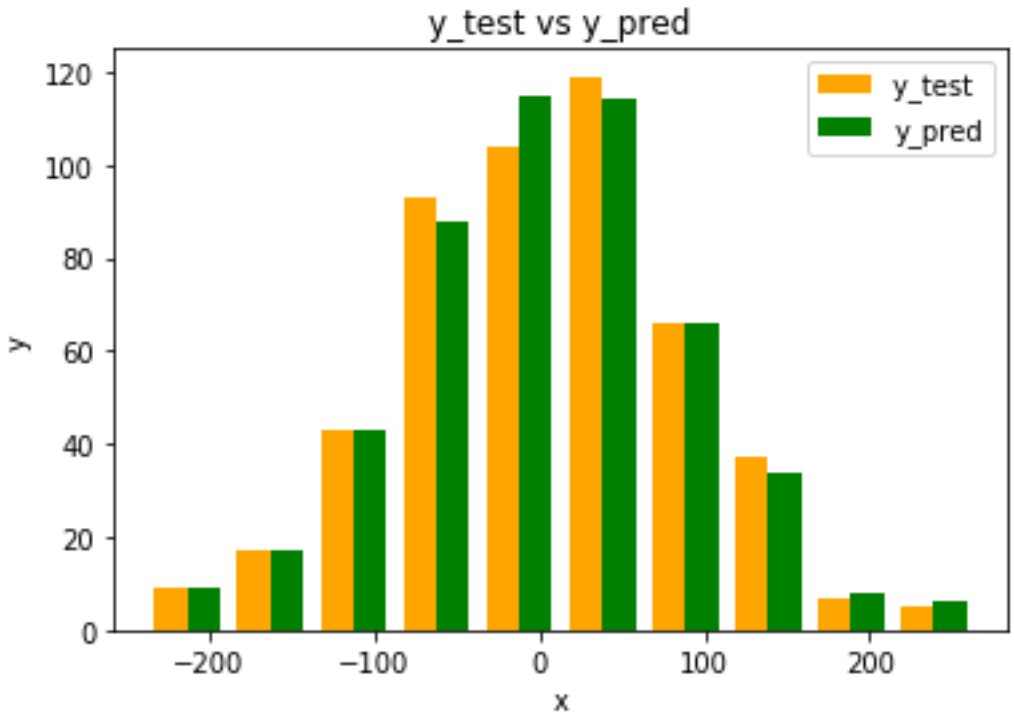
plt.plot(x_test, y_pred, color = "b")

Here is out plot:



At the end of the program we plot the y test data vs y predicted data on a histogram for comparison.

plt.hist([y_test, y_pred], color=['orange', 'green'])



Here is the complete sklearn test program:

```
"""
linearRegSklearn.py
"""
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

print('Linear Regression using Sklearn')
print('y = mx + b')

n = 1000   # number of points

# make random  normal distribution x,y points
x, y = datasets.make_regression(n_samples=n, n_features=1, noise=10)
```

```
# split 80% of the data to the training set
# 20% of the data to test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.2, random_state=0)

# make regression model
reg = LinearRegression()

# fit model with training data
reg.fit(x_train, y_train)

# retrieve slope and y-intercept b
m = reg.coef_
b = reg.intercept_
print("m = ",m, "b = ", b)

# calculate y prediction values
y_pred = reg.predict(x_test)

# plotting x,ydata as scatter plot
plt.scatter(x , y , color = "r", marker = "o", s = 5)

# plotting the regression line
plt.plot(x_test, y_pred, color = "b")

# title
plt.title('linear regression using sklearn')

# x,y labels
plt.xlabel('x')
plt.ylabel('y')

# show plot
plt.show()

# plot train and prediction data as histogram
plt.hist([y_test, y_pred], color=['orange', 'green'])
plt.xlabel("x")
plt.ylabel("y")
plt.legend(['y ','y_pred'])
plt.title('y vs ypred')
plt.show()
```

Todo

Try different values data split percentages in **train_test_split**

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.2, random_state=0)

Try different noise levels.

x, y = datasets.make_regression(n_samples=n, n_features=1, noise=10)

Try predicting individual values using the Sklearn predict function.

You need to put the x test value in a 2 dimensional array. Sklearn expects the x_test values as a 2 dimensional array. Like this **([[20]])** The predict function returns a 1 dimensional array of values.

Example:

```
y_pred = reg.predict([[20]])

print("y_pred",y_pred[0],"x_test",20)
```

```
y_pred 1356.5079194842547 x_test 20
```

Prediction Models using Linear Regression

Simple linear regression is an approach for predicting a **quantitative response** using a **single feature** (or "predictor" or "input variable"). It takes the following form:

$$y=b0+b1x$$

Where:

y is the response
x is the feature
b0 is the intercept
b1 is the coefficient for x

Together, $b0$ and $b1$ are called the **model coefficients**. To create your model, you must "learn" the values of these coefficients. And once we've learned these coefficients, we can use the model for prediction.

Example Sales Model

We want to model how many sales there are for amounts paid tor TV, newspaper and radio advertising categories.

	TV	radio	newspaper	sales
0	123	34	12	54
1	234	23	32	34
2	213	76	24	23
3	432	45	42	78
4	432	23	12	54
5	125	12	54	34
6	543	32	32	23
7	233	31	24	65
8	213	67	34	54
9	324	45	45	34

We will store the above data into a pandas DataFrame using a dictionary.

We need to import **pandas** for the data frame, **sklearn.linear_model** for linear regression and **sklearn.model_selection** and for train test and split to be used later.

```
import pandas as pd
import sklearn
from sklearn.linear_model  import LinearRegression
from sklearn.model_selection import train_test_split
```

We make a dictionary of the above sample data

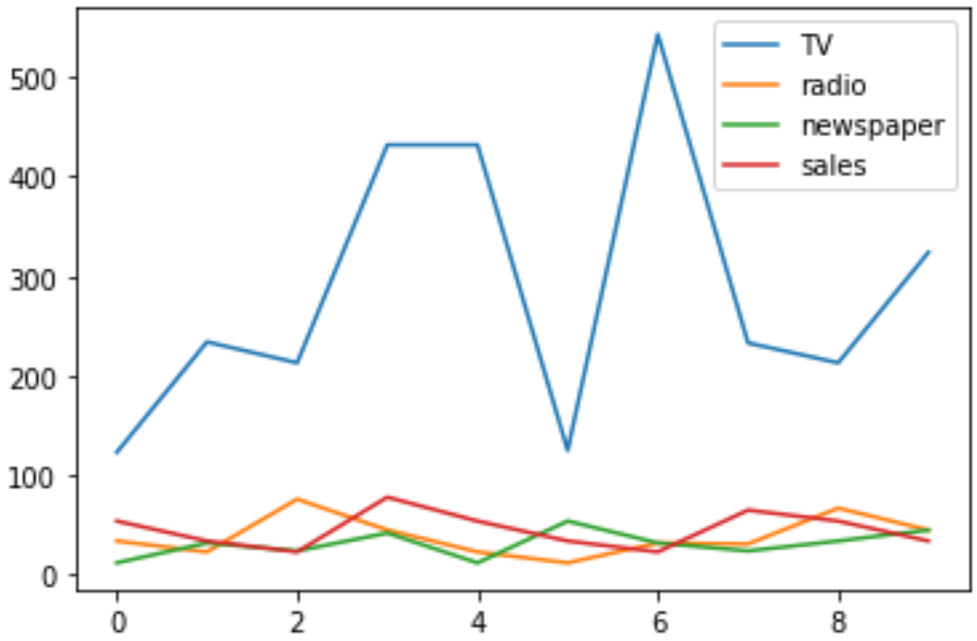
```
data = { 'TV': [123,234,213,432,432,125,543,233,213,324],
'radio':[34,23,76,45,23,12,32,31,67,45],
'newspaper':[12,32,24,42,12,54,32,24,34,45],
'sales':[54,34,23,78,54,34,23,65,54,34]}
```

Using the above dictionary we load into a data frame

```
df = pd.DataFrame(data)
```

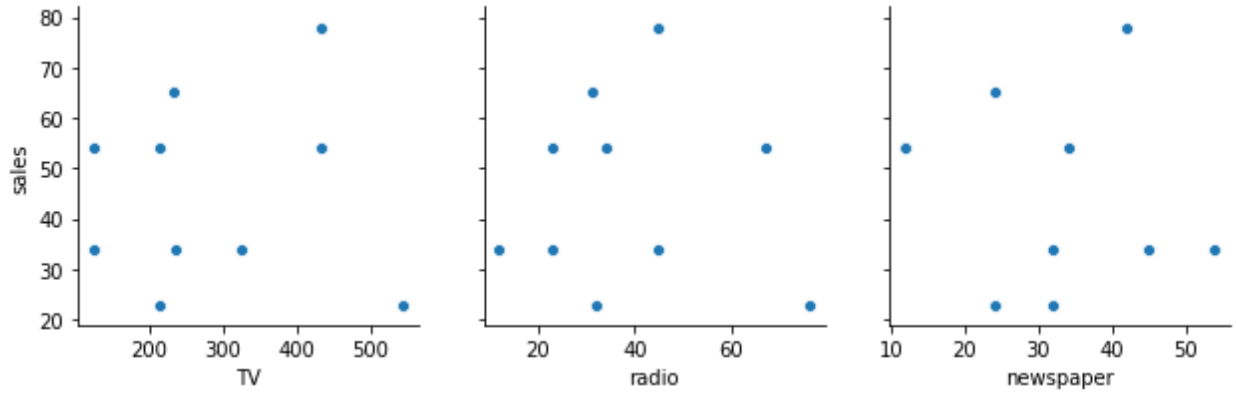
We can use pandas to plot the data frame

```
# show data frame
print(df)

# plot data frame
df.plot()
plt.show()
```



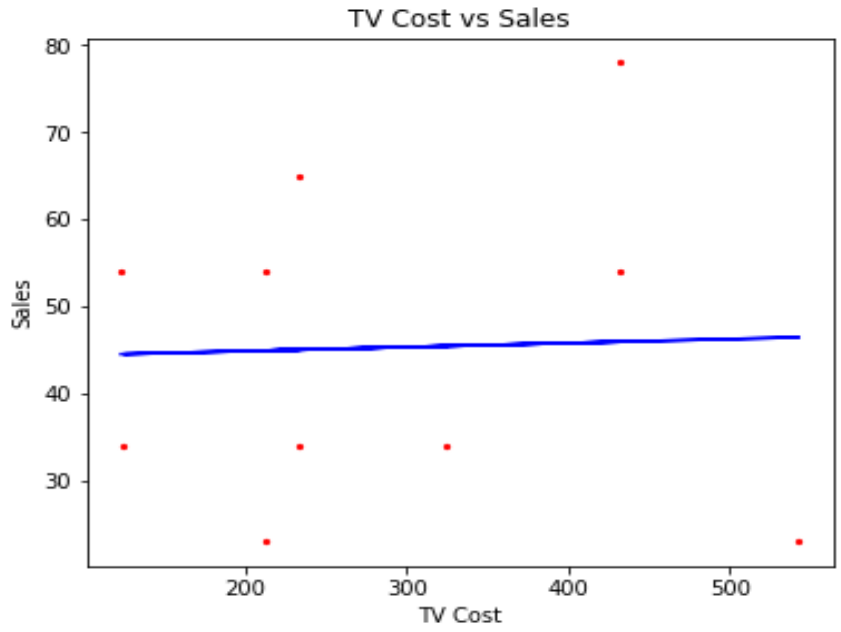
We can also use the pair plot function from **seaborn** to plot out the three advertising categories:

```
import seaborn as sns
sns.pairplot(df, x_vars=['TV','radio','newspaper'], y_vars='sales', height=3)
plt.show()
```



We then use use sklearn to calculate regression **for each** advertising category TV, radio and newspaier features.

```
# create X and y
feature_cols = ['TV']
X = df[feature_cols]
y = df.sales

# instantiate and fit (lm for linear model)
lmTV = LinearRegression()
lmTV.fit(X, y)

# calculate y prediction values
y_pred = lmTV.predict(X)

# plotting x,ydata as scatter plot
plt.scatter(X , y , color = "r", marker = "o", s = 5)

# plotting the regression line
plt.xlabel('TV Cost')
plt.ylabel('Sales')
plt.title('TV Cost vs Sales')
plt.plot(X, y_pred, color = "b")
plt.show()
```



print the coefficients

```
print ('intercept: ',lmTV.intercept_)
print ('coefficient: ',lmTV.coef_[0])
```

intercept: 43.980405165026205

coefficient: [0.00459469]

predict sales forTV advertising cost of 200 using intercept and coefficient

```
sales = lmTV.intercept_ * lmTV.coef_* 200
print('Sales for TV: ',sales)
```

prediction Sales for TV 200: [44.89934307]

predict sales for TV advertising spend of 200 using sklearn for prediction

Note: We put the value 200 into a data frame for the TV column feature.

```
X_new = pd.DataFrame({'TV': [200]})
sales = lmTV.predict(X_new)
print("sklearn prediction Sales for TV 200:: ",sales)
```

```
sklearn prediction Sales for TV 200::  [44.89934307]
```

R-Squared value

The most common way to evaluate the overall fit of a linear model is by the **R-squared** value. R-squared is the **proportion of variance explained**, meaning the proportion of variance in the observed data that is explained by the model, or the reduction in error over the **null model**. (The null model just predicts the mean of the observed response, and thus it has an intercept and no slope.)

R-squared is between 0 and 1, and higher is better because it means that more variance is explained by the model.

```
r_squared = lmTV.score(X, y)
print("R-Squared: ",r_squared)
```

```
R-Squared:  0.0012325129765657916
```

Here is the complete program:

```
# Prediction Models using Linear Regression
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.linear_model  import LinearRegression
from sklearn.model_selection import train_test_split
```

```
data = { 'TV': [123,234,213,432,432,125,543,233,213,324],
        'radio':[34,23,76,45,23,12,32,31,67,45],
        'newspaper':[12,32,24,42,12,54,32,24,34,45],
        'sales':[54,34,23,78,54,34,23,65,54,34]}

#Using the above dictionary we load into a data frame
df = pd.DataFrame(data)
#We can use pandas to plot the data frame
# show data frame
print(df)
# plot data frame
df.plot()
plt.show()

# pair plot
import seaborn as sns
sns.pairplot(df, x_vars=['TV','radio','newspaper'], y_vars='sales', height=3)
plt.show()

# create X and y
feature_cols = ['TV']
X = df[feature_cols]
y = df.sales

# instantiate and fit (lm for linear model)
lmTV = LinearRegression()
lmTV.fit(X, y)

# calculate y prediction values
y_pred = lmTV.predict(X)

# plotting x,ydata as scatter plot
plt.scatter(X , y , color = "r", marker = "o", s = 5)

# plotting the regression line
plt.xlabel('TV Cost')
plt.ylabel('Sales')
plt.title('TV Cost vs Sales')
plt.plot(X, y_pred, color = "b")
plt.show()

print ('intercept: ',lmTV.intercept_)
print ('coefficient: ',lmTV.coef_[0])
```

```
sales = lmTV.intercept_ * lmTV.coef_* 200
print('Sales for TV: ',sales)

X_new = pd.DataFrame({'TV': [200]})
sales = lmTV.predict(X_new)
print("sklearn prediction Sales for TV 200:: ",sales)


r_squared = lmTV.score(X, y)
print("R-Squared: ",r_squared)
```

Linear Regression Homework 2

Using the above program "Prediction Models using Linear Regression" do predictions also for Newspaper and Radio. Plot the x,y points the y prediction regression lines and then predict some values. Calculate and print out the r_squared value. Make 3 separate subplots each one for TV, Newspaper and Radio Put all three plots in a 1 row by 3 column grid plot.

Hints:

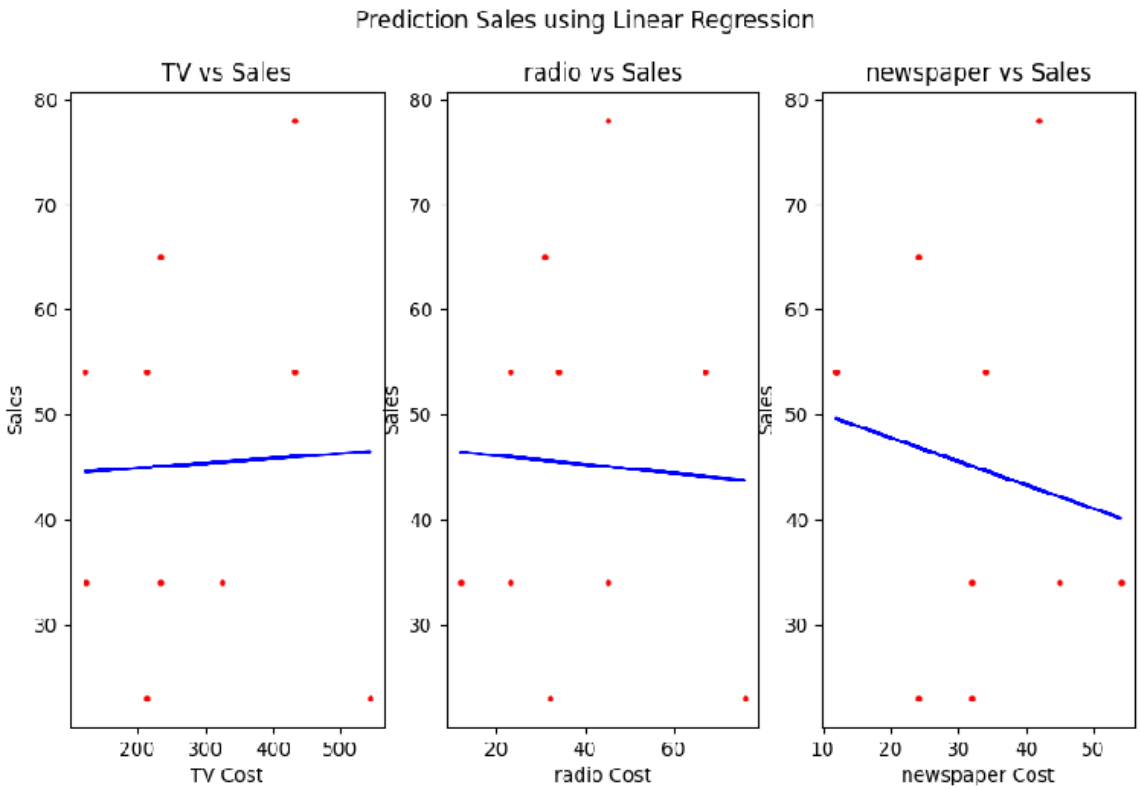
Use the axes array from the subplot to reference each plot.

```
fig, axs = plt.subplots(1,3,figsize=(10,6))
```

Use a enumerate loop that has a index counter and selects the feature from a list of features.

```
features = ['TV','radio','newspaper']
for i,feature in enumerate(features):
```

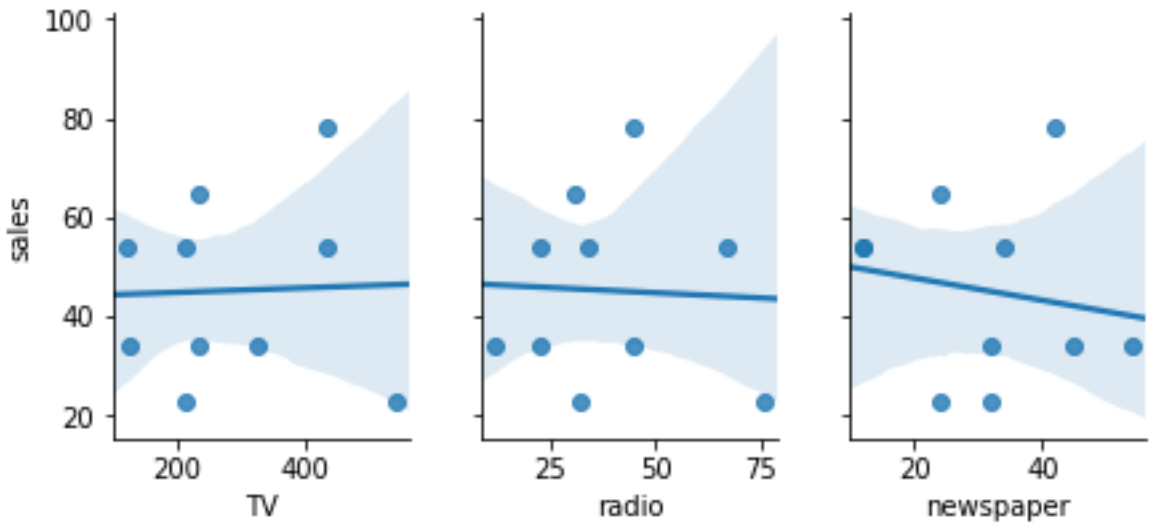
You should get something like this:



Plotting multiple regression lines with Seaborn

We can use seaborn to plot multiple regression line for our TV,radio, and newspaper advertising amounts.

sns.pairplot(df, x_vars=['TV','radio','newspaper'], y_vars='sales', height=3, aspect=0.7, kind='reg')
plt.show()



Multiple Linear Regression

Simple linear regression can easily be extended to include multiple features. This is called **multiple linear regression**:

y=b0+b1x1+b2x2...+bnxn

Each **x** represents a different feature, and each feature has its own coefficient.

In our advertising mode this can be b0..b3:

y=b0+b1×TV+b2×$Radio$+b3×$Newspaper$

Where **y** is the estimated sales for combined cost of TV, radio and newspaper advertising costs

Using sklearn to estimate these coefficients:

```
feature_cols = ['TV', 'radio', 'newspaper']

X = df[feature_cols]
y = df.sales
```

make linear regression model **lm** and fit points

```
lm = LinearRegression()
lm.fit(X, y)
```

print out the intercept and the coefficients

```
print ("intercept: ",lm.intercept_)
print ("coefficient: ",lm.coef_)
```

```
intercept:  53.55388522521896

coefficient:  [ 0.00369613 -0.05436979 -0.23169985]
```

Notice we now have 3 different coefficients 1 for each feature 'TV', 'radio', 'newspaper'

```
print(feature_cols, lm.coef_)
```

```
['TV', 'radio', 'newspaper'] [ 0.00369613 -0.05436979 -0.23169985]
```

What the coefficients mean:

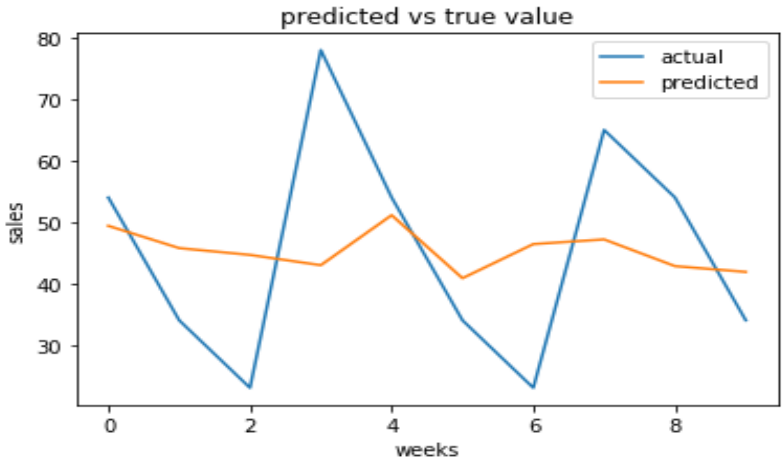
For a given amount of Radio and Newspaper ad spending, an **increase of $1000 in TV ad spending** is associated with an **increase in Sales of 3.69**

We can get a list of predicted values combined for the three X features

```
# predicte values
y_pred = lm.predict(X)
```

We can plot out the actual value vs the predicted values

```
plt.plot(y,label="actual")
plt.plot(y_pred,label="predicted")
plt.legend()
plt.title('predicted vs true value')
plt.xlabel('weeks')
plt.ylabel('sales')
plt.show()
```



Model Evaluation Metrics for Evaluating Linear Regression

For regression algorithms, three evaluation metrics are commonly used:

Mean Absolute Error
Mean Squared Error
Root Mean Squared Error

Mean Absolute Error (MAE) is the mean of the absolute value of the errors between predicted value and true value It is calculated as:

$$MAE = \frac{1}{n} \sum_{j=1}^{n} (ypredj - yj)$$

Mean Squared Error (MSE) is the mean of the squared errors between predicted value and true value and is calculated as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (ypredi - yi)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors between predicted value and true value:

$$RMSE = SQRT \frac{1}{n} \sum_{i=1}^{n} (ypredi - yi)^2$$

We can use out previous predicted and actual true value to calculate MAE, MSE, RMSE. Where y = true value.

```
print ("MAE:",sklearn.metrics.mean_absolute_error(y, y_pred))
print ("MSE:",sklearn.metrics.mean_squared_error(y, y_pred))
print ("RMSE:",np.sqrt(sklearn.metrics.mean_squared_error(y, y_pred)))
```

```
MAE: 14.30775238306735
MSE: 296.31803637577434
RMSE: 17.213890797137477
```

Not very accurate?

Here is the complete program for multiple linear regression:

```
# Multiple Linear Regression
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn
from sklearn.linear_model import LinearRegression

# make dictionary of advertising costs to sales result
data = { 'TV': [123,234,213,432,432,125,543,233,213,324],
      'radio':[34,23,76,45,23,12,32,31,67,45],
      'newspaper':[12,32,24,42,12,54,32,24,34,45],
      'sales':[54,34,23,78,54,34,23,65,54,34]}
```

```
# load dictionary into a data frame
df = pd.DataFrame(data)

feature_cols = ['TV', 'radio', 'newspaper']
X = df[feature_cols]
y = df.sales
#make linear regression model lm and fit points
lm = LinearRegression()
lm.fit(X, y)
#print out the intercept and the coefficients
print ("intercept: ",lm.intercept_)
print ("coefficient: ",lm.coef_)

print(feature_cols, lm.coef_)

# predicte values
y_pred = lm.predict(X)
#We can plot out the actual value vs the predicted values
plt.plot(y,label="actual")
plt.plot(y_pred,label="predicted")
plt.legend()
plt.title('predicted vs true value')
plt.xlabel('weeks')
plt.ylabel('sales')
plt.show()

print ("MAE:",sklearn.metrics.mean_absolute_error(y, y_pred))
print ("MSE:",sklearn.metrics.mean_squared_error(y, y_pred))
print ("RMSE:",np.sqrt(sklearn.metrics.mean_squared_error(y, y_pred)))
```

Model Evaluation Using Train/Test Split

We will now split our data into 2 sets a train set for fitting the data and a test set to test our prediction

```
X = df[['TV', 'radio', 'newspaper']]
y = df.sales

# split data into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

# make linear regression model
lm = LinearRegression()
```

```
# fit train data
lm.fit(X_train, y_train)

# predict data using x test set
y_pred = lm.predict(X_test)

# calculate error between test y test set and y prediction
print ("RMSE:",np.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred)))
```

```
RMSE: 40.621959078406384
```

Todo:

Run the above code:

Then:

(1) remove newspaper and run code again and record RMSE

```
X = df[['TV', 'radio']]
```

(2) remove radio and run code again and record RMSE

```
X = df[['TV', 'newspaper']]
```

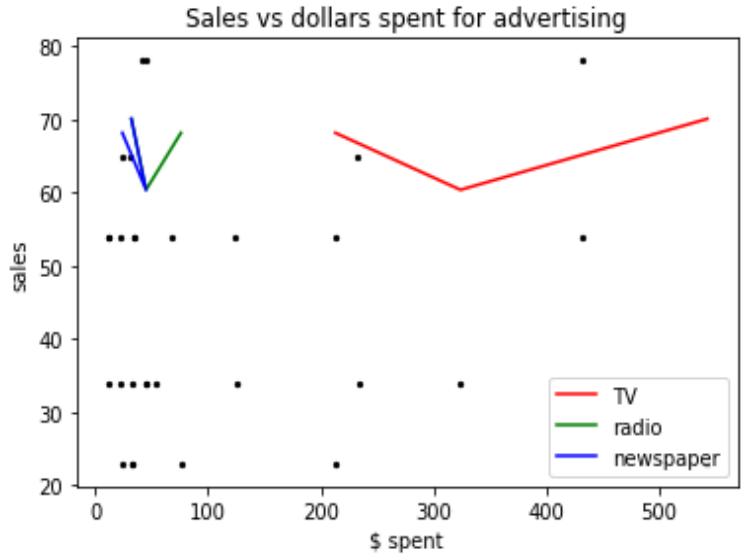
(3) remove TV and run code again and record RMSE test result

```
X = df[['radio', 'newspaper']]
```

(4) Which one has the highest RMSE score?

Next We plot all the regression lines for each feature: TV, radio and newspaper.

```
# plot sales vs dollars spent per feature

# plot TV
plt.scatter(X["TV"] , y , color = "k", marker = "o", s = 5)
plt.plot(X_test["TV"], y_pred, color = "r",label="TV")

# plot radio
plt.scatter(X["radio"] , y , color = "k", marker = "o", s = 5)
plt.plot(X_test["radio"], y_pred, color = "g",label="radio")

# plot newspaper
plt.scatter(X["newspaper"] , y , color = "k", marker = "o", s = 5)
plt.plot(X_test["newspaper"], y_pred, color = "b",label="newspaper")

# add a Legend
plt.legend()

# plot title an axis labels
plt.title('Sales vs dollars spent for advertising')
plt.xlabel('$ spent')
plt.ylabel('sales')

plt.show()
```



The graph defiantly show TV advertising is most costly for an increase in sales

Here is the complete program:

```
# model evaluation
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn
from sklearn.linear_model  import LinearRegression
from sklearn.model_selection import train_test_split

# make dictionary of advertising costs to sales result
data = { 'TV': [123,234,213,432,432,125,543,233,213,324],
       'radio':[34,23,76,45,23,12,32,31,67,45],
       'newspaper':[12,32,24,42,12,54,32,24,34,45],
       'sales':[54,34,23,78,54,34,23,65,54,34]}

# load dictionary into a data frame
df = pd.DataFrame(data)

X = df[['TV', 'radio', 'newspaper']]
y = df.sales

# split data into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

# make linear regression model
lm = LinearRegression()

# fit train data
lm.fit(X_train, y_train)

# predict data using x test set
y_pred = lm.predict(X_test)

# calculate error between test y test set and y prediction
print ("RMSE:",np.sqrt(sklearn.metrics.mean_squared_error(y_test, y_pred)))

# plot sales vs dollars spent per feature

# plot TV
plt.scatter(X["TV"] , y , color = "k", marker = "o", s = 5)
plt.plot(X_test["TV"], y_pred, color = "r",label="TV")
```

```
# plot radio
plt.scatter(X["radio"] , y , color = "k", marker = "o", s = 5)
plt.plot(X_test["radio"], y_pred, color = "g",label="radio")

# plot newspaper
plt.scatter(X["newspaper"] , y , color = "k", marker = "o", s = 5)
plt.plot(X_test["newspaper"], y_pred, color = "b",label="newspaper")

# add a Legend
plt.legend()

# plot title an axis labels
plt.title('Sales vs dollars spent for advertising')
plt.xlabel('$ spent')
plt.ylabel('sales')

plt.show()
```

REGRESSION HOMEWORK QUESTION 3

We want to predict the best method of delivering parcels using trucks using gas, electric or hybrid (gas and electric). The delivery company wants to know to buy more gas, electric or hybrid trucks.

Our model is

Delivery costs = b0 m+ b1 gas + b2 electric + b3 hybrid

Sample data is available:

	Gas	Electric	Hybrid	Cost
0	23	34	12	123
1	32	23	32	567
2	23	23	24	345
3	15	31	23	234
4	20	23	12	432
5	16	12	13	321
6	16	32	12	122
7	33	31	24	343
8	24	35	16	234
9	36	36	23	478

Use sklearn to calculate the coefficients, then find out the most efficient (lowest cost) of buying more gas, electric or hybrid trucks.

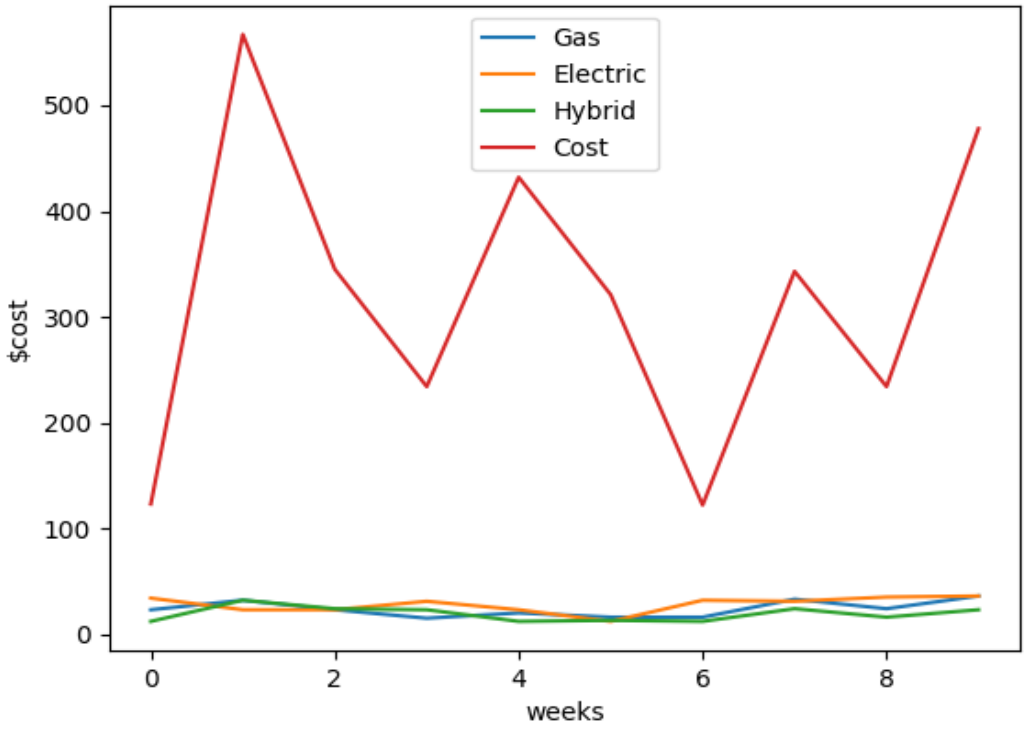
You may want to make a enumeration loop similar to what we did in Question 2 to make your programming task easier. But plot all feature on one plot rather than seperate plots.

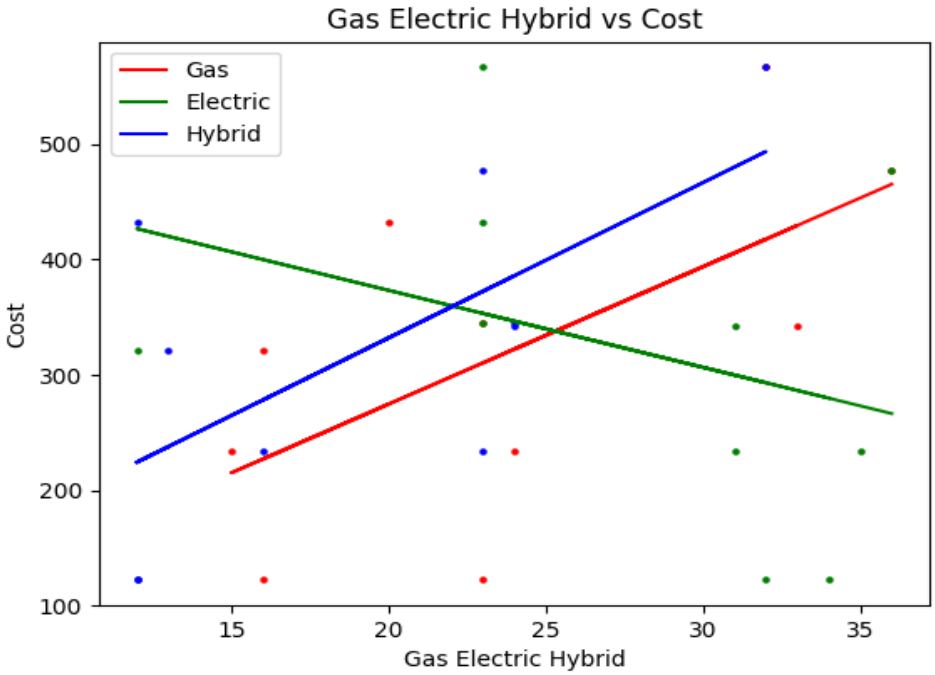
Make scatter and line plots of your results.

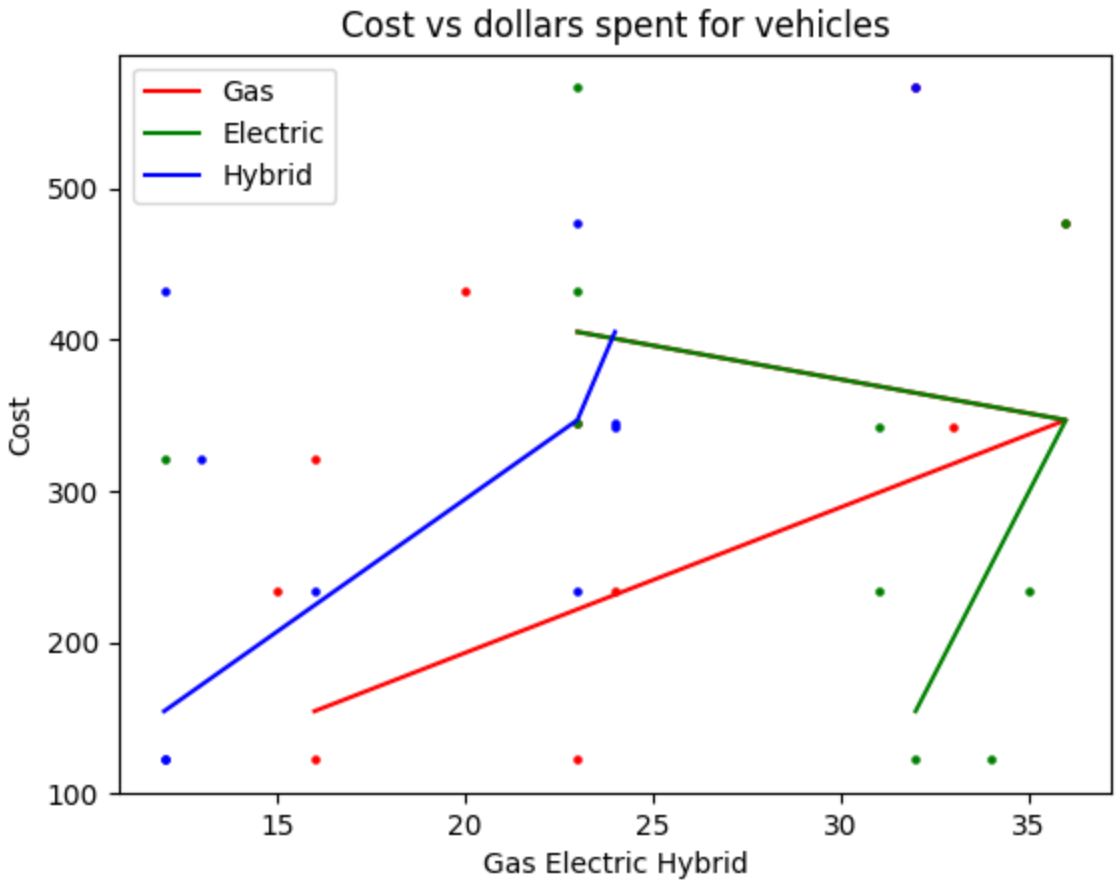
Put all your regression homework in a py file called regression_homework.py

You may have something like this:

```
   Gas  Electric  Hybrid  Cost
0   23       34      12   123
1   32       23      32   567
2   23       23      24   345
3   15       31      23   234
4   20       23      12   432
5   16       12      13   321
6   16       32      12   122
7   33       31      24   343
8   24       35      16   234
9   36       36      23   478
Gas intercept:  35.887897595034815
Gas coefficient:  [11.93328161]
Electric intercept:  507.1295719844358
Electric coefficient:  [-6.68677043]
Hybrid intercept:  62.35403025513659
Hybrid coefficient:  [13.48408219]
Combined intercept:  226.111552230183
Combined coefficient:  [ 12.46903092 -10.92949918   5.39536592]
all
RMSE: 85.31669817753428
omit gas
RMSE: 129.90167447170555
omit electric
RMSE: 81.91623331718274
omit hybrid
RMSE: 66.54192393144336
```







END