LINEAR REGRESSION MODELS

DONE BY:SAHITI EMANI
STUDENT ID:19556
GUIDED BY:PROF.HENRY CHANG

TABLE OF CONTENTS:

INTRODUCTION:

- LINEAR REGRESSION :The **independent variable** is the cause, and the **dependent variable** is the effect where there is a relationship of cause and effect.
- **Least squares linear regression** is a method for predicting the value of a dependent variable *Y*, based on the value of an independent variable *X*.
- A tool - a software package (e.g., Excel) or a graphing calculator - to find b_0 and b_1. You enter the *X* and *Y* values into your program or calculator, and the tool solves for each parameter.
- In the unlikely event that you find yourself on a desert island without a computer or a graphing calculator, you can solve for b_0 and b_1 "by hand". Here are the equations.
  - $b_1 = \Sigma [(x_i - x)(y_i - y)] / \Sigma [(x_i - x)_2]$
  - $b_1 = r * (s_y / s_x)$
  - $b_0 = y - b_1 * x$
- b_0 is the constant in the regression equation, b_1 is the regression coefficient, r is the correlation between x and y, x_i is the *X* value of observation *i*, y_i is the *Y* value of observation *i*, x is the mean of *X*, y is the mean of *Y*, s_x is the standard deviation of *X*, and s_y is the standard deviation of *Y*.

PROPERTIES OF A REGRESSION LINE:

When the regression parameters (b_0 and b_1) are defined as described above, the regression line has the following properties.

- The line minimizes the sum of squared differences between observed values (the *y* values) and predicted values (the ŷ values computed from the regression equation).
- The regression line passes through the mean of the *X* values (x) and through the mean of the *Y* values (y).
- The regression constant (b_0) is equal to the **y intercept** of the regression line.
- The regression coefficient (b_1) is the average change in the dependent variable (*Y*) for a 1-unit change in the independent variable (*X*). It is the slope of the regression line.

The least squares regression line is the only straight line that has all of these properties.

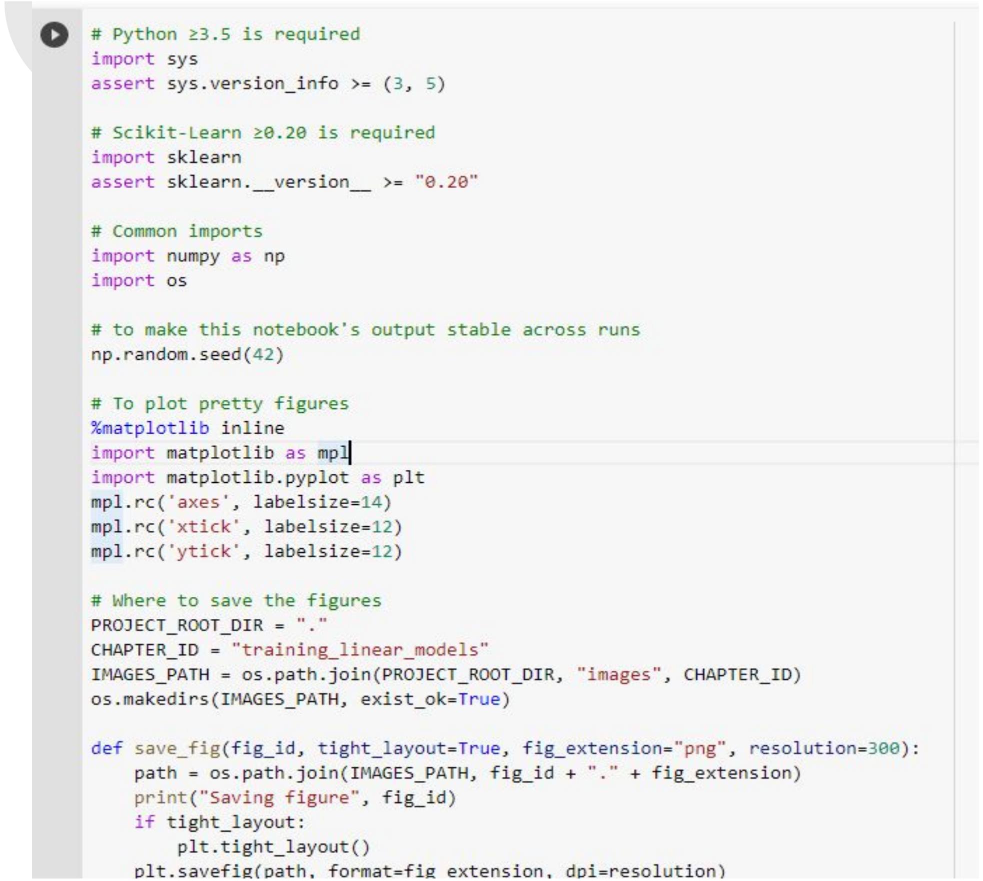
HOW TO CHOOSE THE RIGHT REGRESSION MODEL:

- Choosing the correct linear regression model can be difficult yet, a research team tasked to investigate typically measures many variables but includes only some of them in the model. The analysts try to eliminate the variables that are not related and include only those with a true relationship.

STATISTICAL METHODS FOR FINDING THE BEST REGRESSION MODEL:

- **Adjusted R-squared and Predicted R-squared**
- **P-values for predictor**
- **Step-wise Rgression and Best subset regression.**

LINEAR REGRESSION USING GENERAL EQUATION:



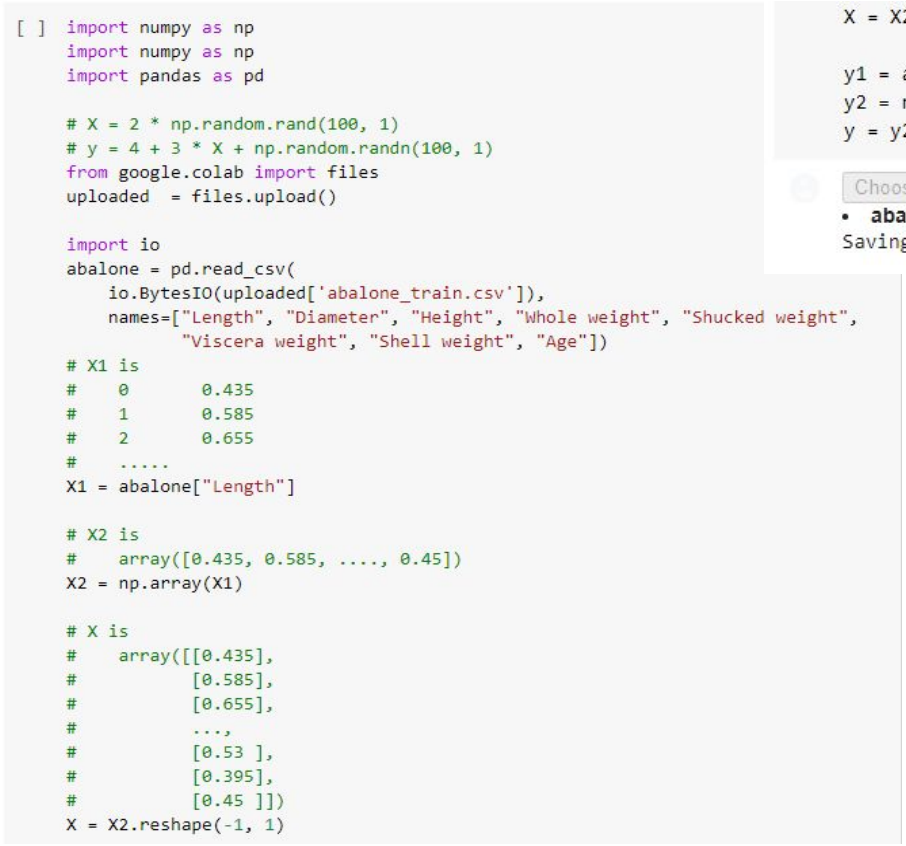
```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "training_linear_models"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Import required for the project

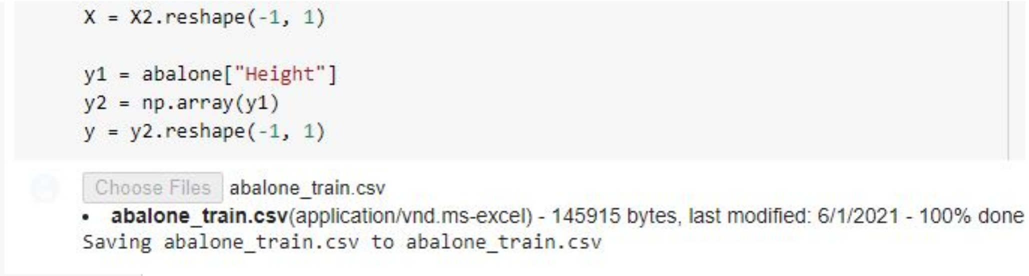
1. **Sklearn** : Python module to implement machine learning algorithm
2. Pandas : Python module for Data analysis
3. **Numpy** : Python module to create and manipulate multidimensional arrays.
4. **Matplotlib** : Python module for Data Visualization. To plot different graphs and save them to different formats

UPLOAD THE DOWNLOADED DATAFILE(.CSV FILE):

Linear regression using the Normal Equation



```
[ ] import numpy as np
    import numpy as np
    import pandas as pd

    # X = 2 * np.random.rand(100, 1)
    # y = 4 + 3 * X + np.random.randn(100, 1)
    from google.colab import files
    uploaded  = files.upload()

    import io
    abalone = pd.read_csv(
        io.BytesIO(uploaded['abalone_train.csv']),
        names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight",
               "Viscera weight", "Shell weight", "Age"])
    # X1 is
    #    0       0.435
    #    1       0.585
    #    2       0.655
    #    .....
    X1 = abalone["Length"]

    # X2 is
    #    array([0.435, 0.585, ...., 0.45])
    X2 = np.array(X1)

    # X is
    #    array([[0.435],
    #           [0.585],
    #           [0.655],
    #           ...,
    #           [0.53 ],
    #           [0.395],
    #           [0.45 ]])
    X = X2.reshape(-1, 1)
```



```
X = X2.reshape(-1, 1)

y1 = abalone["Height"]
y2 = np.array(y1)
y = y2.reshape(-1, 1)
```

Choose Files abalone_train.csv

- **abalone_train.csv**(application/vnd.ms-excel) - 145915 bytes, last modified: 6/1/2021 - 100% done

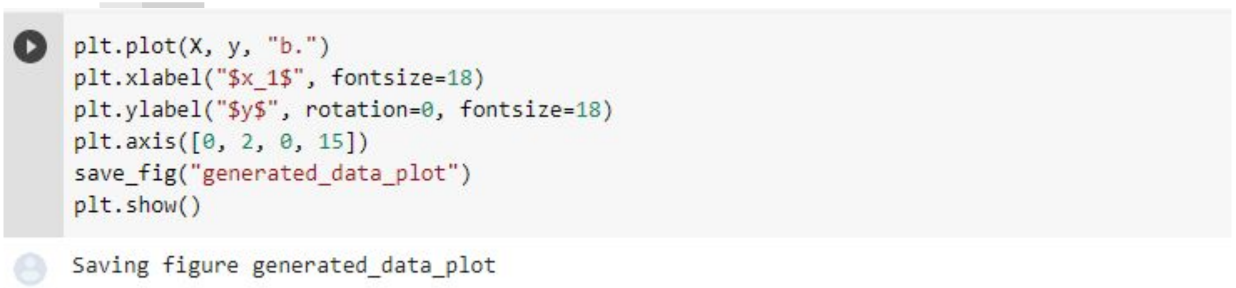
```
Saving abalone_train.csv to abalone_train.csv
```

Loading data from csv file using pandas and adding names to column. This will return a dataframe and save it to abalone

X1 -> Data from Length column in abalone is saved to X1
X2 -> X1 is converted to 1D array using numpy
X   -> converts X2 array to -1, 1 shape

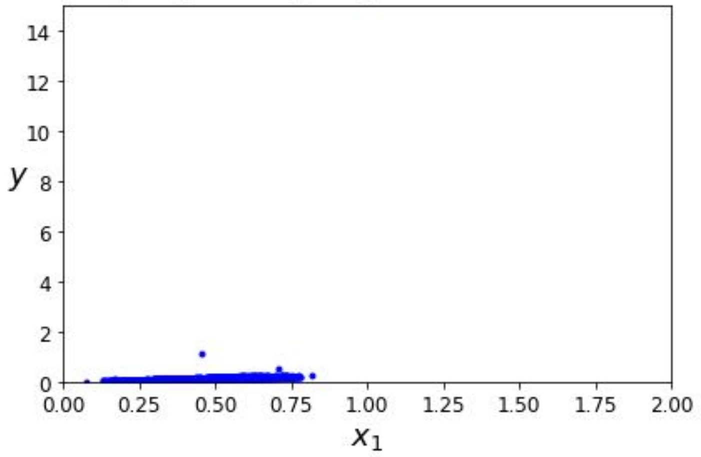
Y1 -> Data from Height column in abalone is saved to y1
Y2 -> Y1 is converted to 1D array using Numpy
Y   -> Converts Y2 array to -1,1 shape

GRAPH:PLOT FOR X AND Y



```
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 2, 0, 15])
save_fig("generated_data_plot")
plt.show()
```

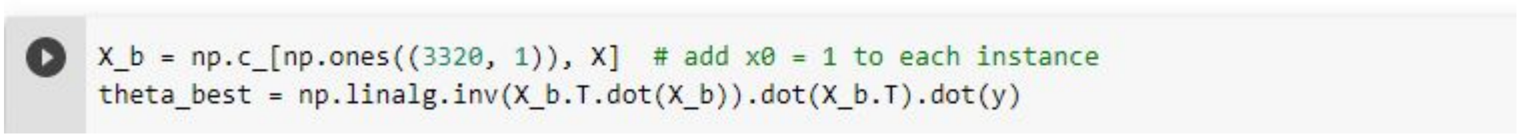
```
Saving figure generated_data_plot
```



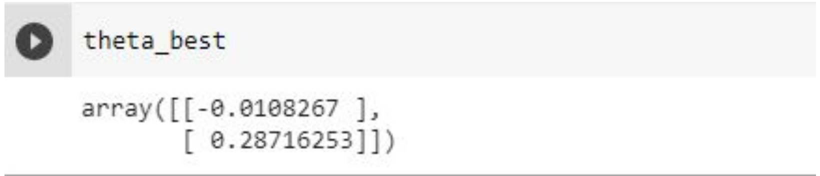
1. Plot plt has been declared using Matplotlib
2. X and Y axis are provided with values from X, Y arrays from the above the steps
3. Labels on the axis were provided using xlabel and ylabel
4. plt.axis() is used to set min and max values for x and y respectively
5. Saved the graph with the name “generated_data_plot” and displayed on screen

THETA CALCULATION:

theta _best is calculated using the linalg.inv() function from numpy , which is used for calculating the inverse of a function.



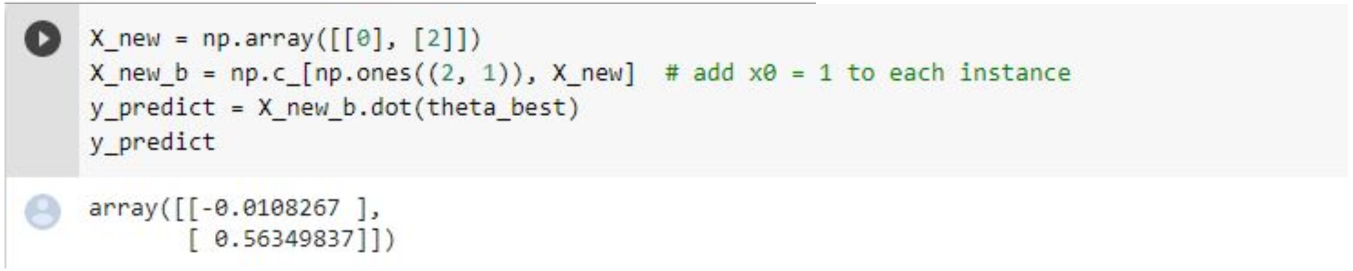
```
X_b = np.c_[np.ones((3320, 1)), X]  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```



```
theta_best
```

```
array([[-0.0108267 ],
       [ 0.28716253]])
```

Theta_best value is displayed

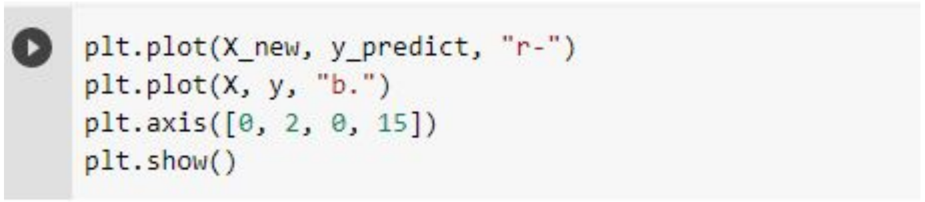


```
X_new = np.array([[0], [2]])
X_new_b = np.c_[np.ones((2, 1)), X_new]  # add x0 = 1 to each instance
y_predict = X_new_b.dot(theta_best)
y_predict
```

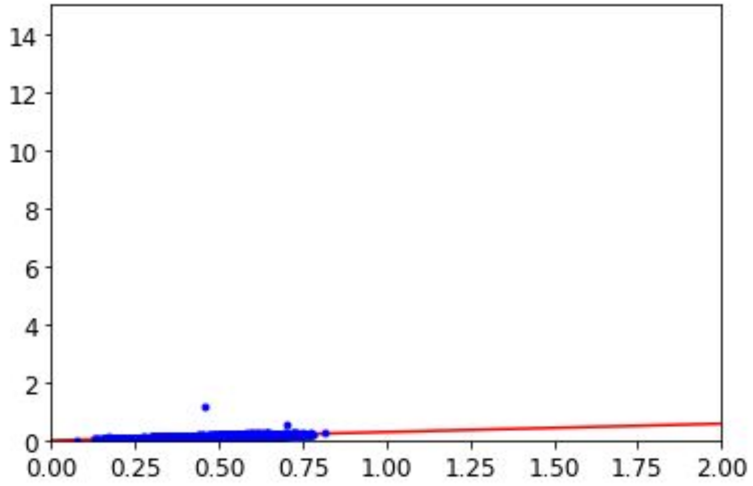
```
array([[-0.0108267 ],
       [ 0.56349837]])
```

Y_predict is calculated and displayed, using x_new_b and theta calculated in the above

PLOT X_NEW AND Y_PREDICT:

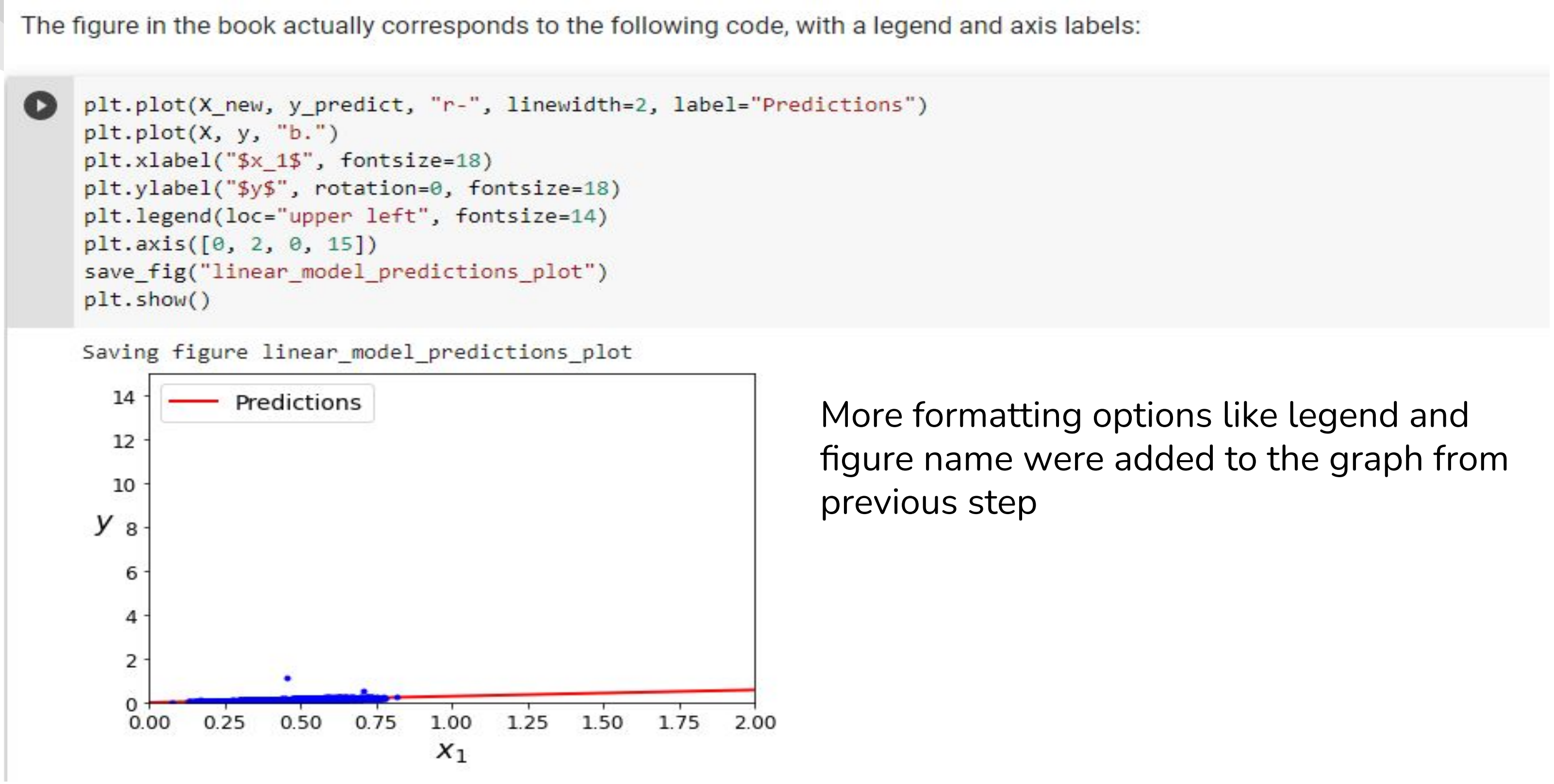


```
plt.plot(X_new, y_predict, "r-")
plt.plot(X, y, "b.")
plt.axis([0, 2, 0, 15])
plt.show()
```



1. X_new and y_predict value plotted to graph and is represented using red line (r-)
2. X and y value are plotted to graph and is represented using blue dots (b.)

ADDITION OF FEW FEATURES:



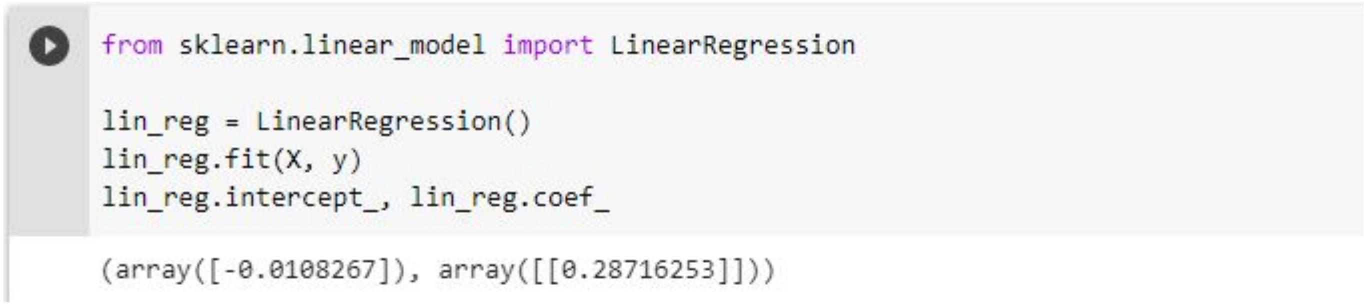
The figure in the book actually corresponds to the following code, with a legend and axis labels:

```
plt.plot(X_new, y_predict, "r-", linewidth=2, label="Predictions")
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 2, 0, 15])
save_fig("linear_model_predictions_plot")
plt.show()
```

```
Saving figure linear_model_predictions_plot
```

More formatting options like legend and figure name were added to the graph from previous step

CREATION OF LINEAR REGRESSION MODEL:



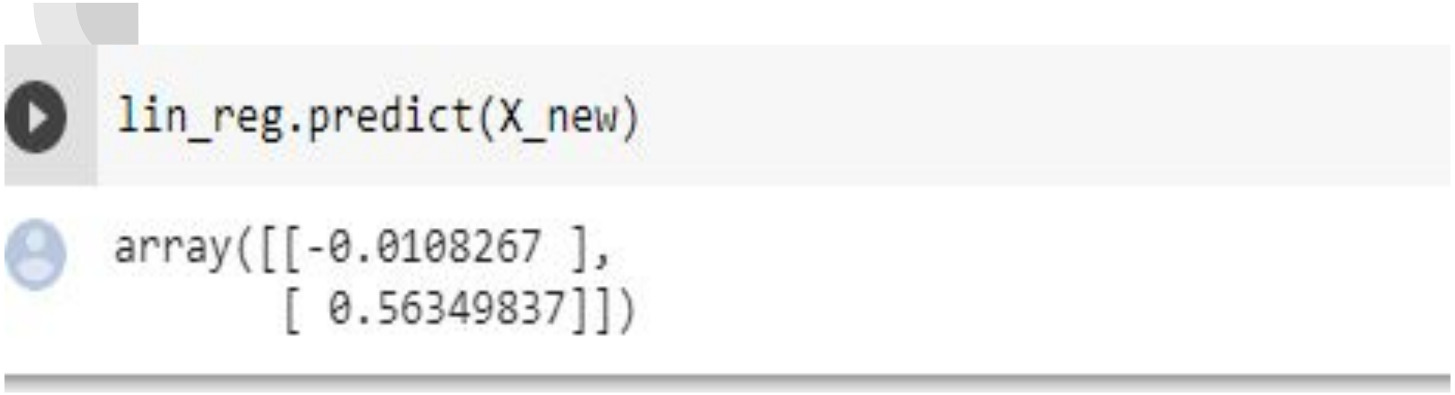
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)
lin_reg.intercept_, lin_reg.coef_
```

```
(array([-0.0108267]), array([[0.28716253]]))
```

LinearRegression module has been imported from sklearn
X, y values were provided to linear regression algorithm using fit() method.

Intercept and coefficient from linear regression were displayed

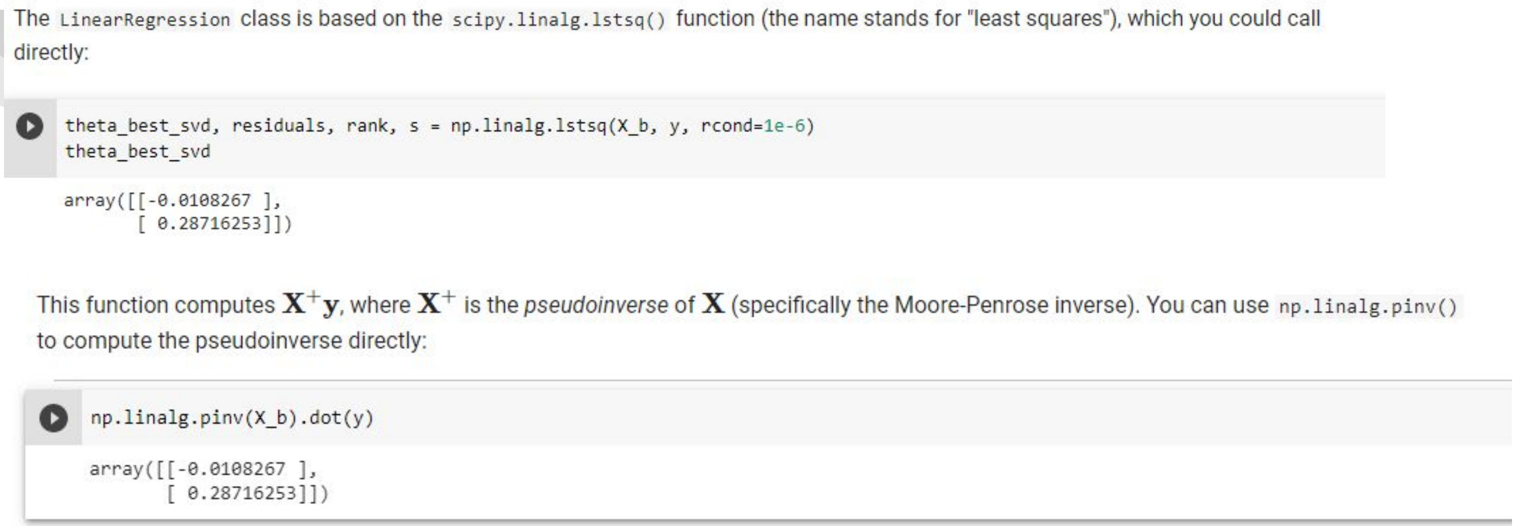
NEW DATASET IS PREDICTED:



```
lin_reg.predict(X_new)
```

```
array([[-0.0108267 ],
       [ 0.56349837]])
```

Now our linear regression object is ready to predict y values.
X_new has provided to predict y value and the same is displayed on the screen



The `LinearRegression` class is based on the `scipy.linalg.lstsq()` function (the name stands for "least squares"), which you could call directly:

```
theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
theta_best_svd
```

```
array([[-0.0108267 ],
       [ 0.28716253]])
```

This function computes $\mathbf{X}^{+}\mathbf{y}$, where $\mathbf{X}^{+}$ is the *pseudoinverse* of $\mathbf{X}$ (specifically the Moore-Penrose inverse). You can use `np.linalg.pinv()` to compute the pseudoinverse directly:

```
np.linalg.pinv(X_b).dot(y)
```

```
array([[-0.0108267 ],
       [ 0.28716253]])
```

Theta_best value can found from sklearn from the following two function

- Np.linalg.lstsq (least squares)
- Np.linalg.pinv (Pseudo inverse of x)

CONCLUSION:

- Implementation of linear regression algorithm with given module.
- Usage of new methods in this topic i.e NORMAL EQUATION AND SCIKIT_LEARN

NORMAL EQUATION:It is an analytical approach to Linear Regression with a Least Square Cost Function. We can directly find out the value of θ without using Gradient Descent.

SCIKIT_LEARN:It is very useful library for machine learning in Python.

- It helps in selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction through a consistent interface in Python.