

CS550 - Machine Learning and Business Intelligence

Jupyter: Training Linear Models

By Soe Wunna (19651)

Instructor: Dr. Chang, Henry

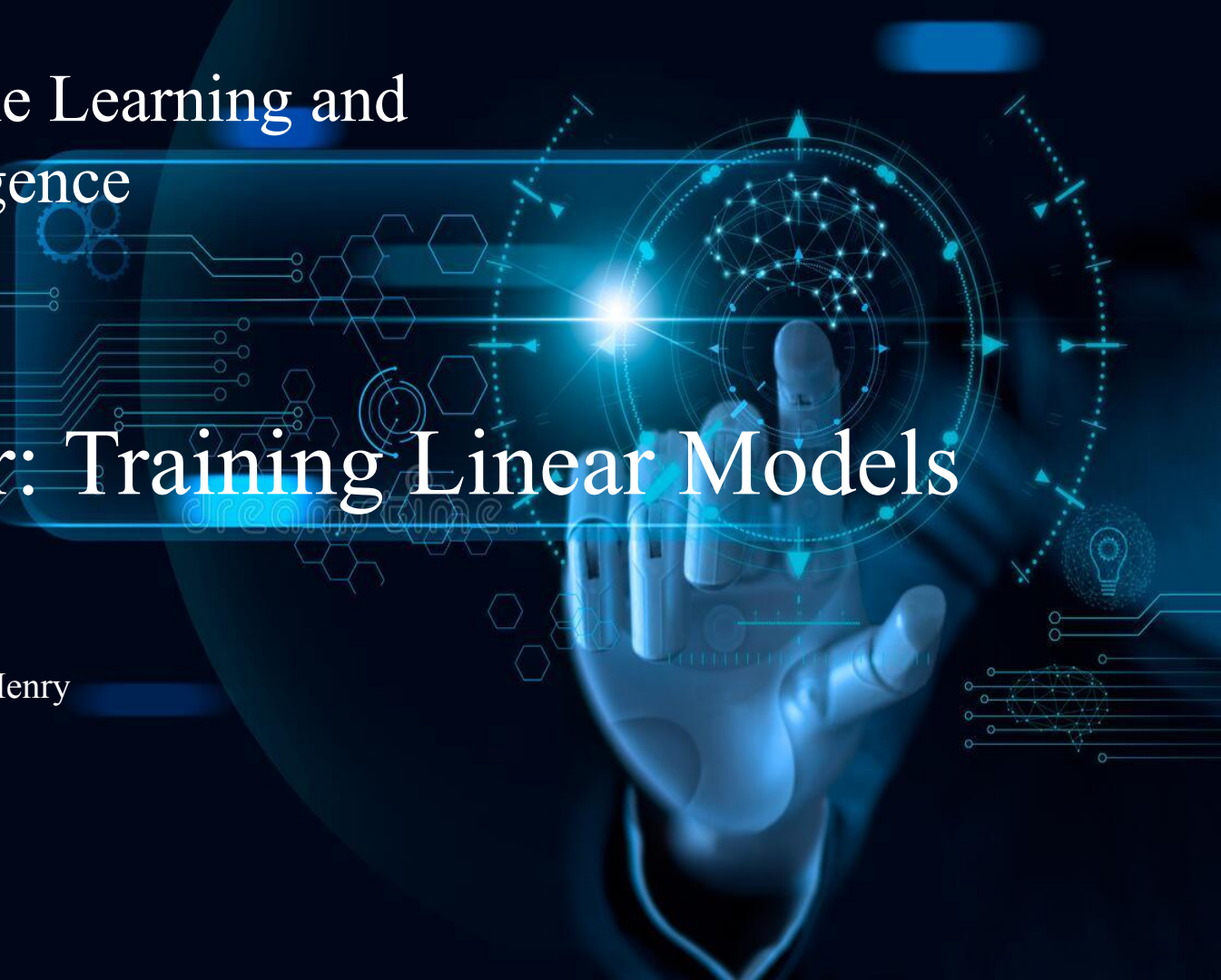
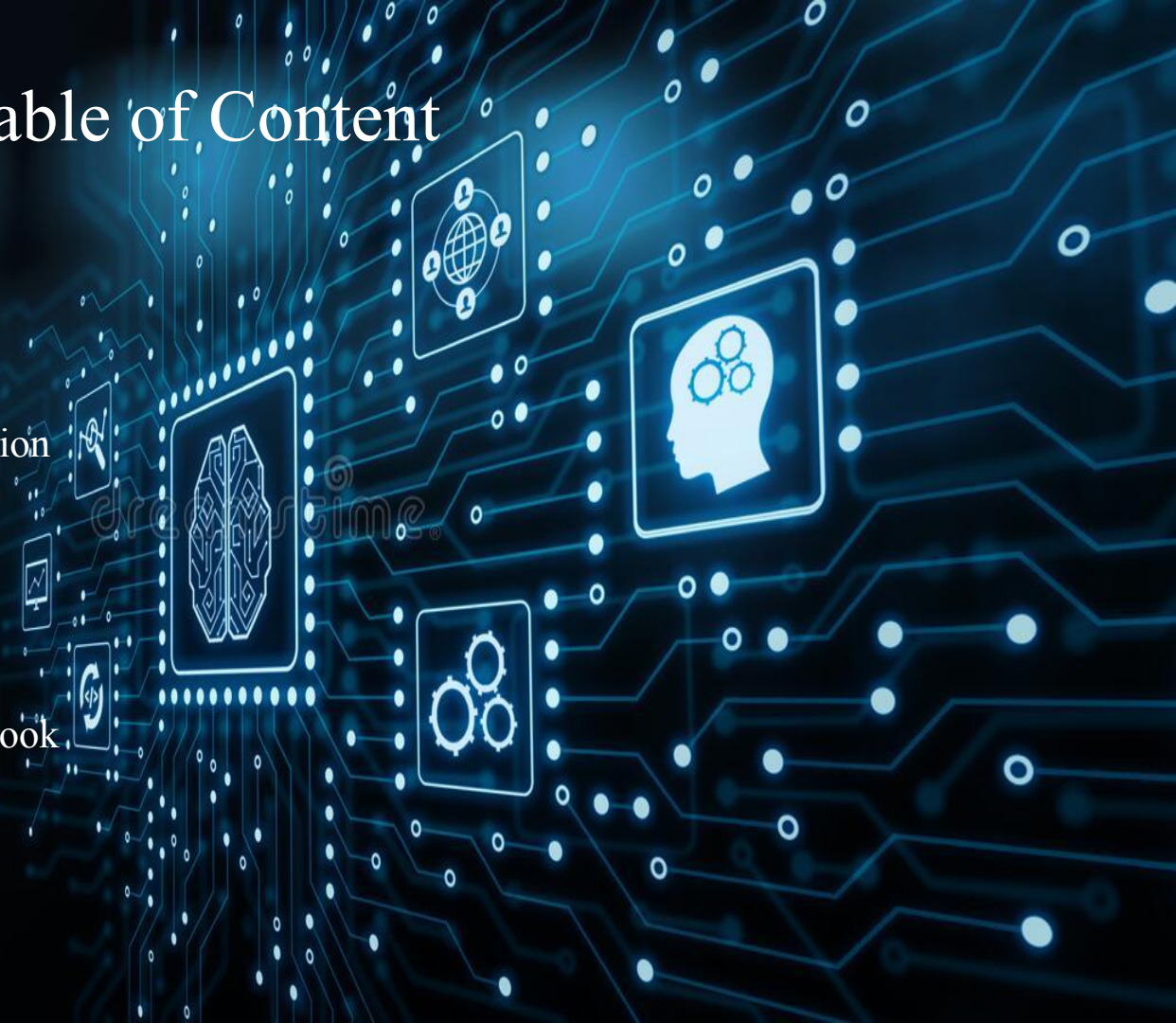


Table of Content

1. Introduction
2. Linear Regression
3. The Normal Equation
4. Derivation of the Normal Equation
5. Google Colab and ipynb
6. Modifying the Code
7. Loading CSV Files into Colab
8. Running Result of Entire Notebook
9. Conclusion
10. References

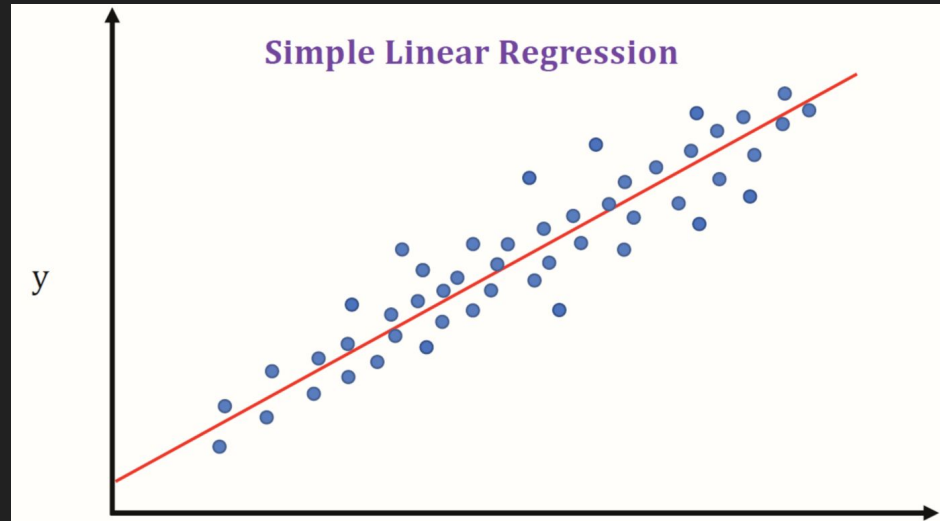


1. Introduction

In this presentation, we will discuss about Linear Regression Model and Normal Equation for Machine Learning, and apply normal equation to the python notebook file.

2. Linear Regression

Linear Regression is the supervised Machine Learning model in which the model finds the best fit linear line between the independent and dependent variable i.e it finds the linear relationship between the dependent and independent variable.



3. Normal Equation

The normal equation is a method to find the optimal solution for linear regression, which is a statistical model that aims to find the best linear relationship between a dependent variable and one or more independent variables. It is used to determine the parameters (coefficients) of the model that minimize the sum of the squared differences between the observed values and the values predicted by the model. The normal equation provides a closed-form solution to this problem, which can be calculated directly from the training data without the need for iterative optimization. The equation is given by:

$$\theta = (X^T X)^{-1} \cdot (X^T y)$$

4. Derivation of the Normal Equation

$$\theta = (X^T X)^{-1} \cdot (X^T y)$$

where θ is the vector of coefficients, X is the design matrix that contains the independent variables, y is the vector of dependent variables, and T is the transpose operator. The normal equation has a computational cost of $O(n^3)$, where n is the number of features, making it more computationally efficient than some iterative optimization methods when the number of features is small. However, it can be impractical to use when the number of features is very large, as the computational cost and memory requirements become prohibitively high.

The normal equation is derived by finding the parameters that minimize the mean squared error (MSE) between the observed values and the values predicted by the model. The MSE is defined as:

$$\text{MSE} = (1/m) * \sum (y_i - h_{\theta}(x_i))^2$$

where m is the number of training examples, y_i is the observed value for the i -th training example, $h_{\theta}(x_i)$ is the predicted value for the i -th training example, and θ is the vector of coefficients. The goal is to find the values of θ that minimize the MSE.

To do this, we take the partial derivative of the MSE with respect to each coefficient in θ and set it equal to zero. This results in the normal equation:

$$X^T X \theta = X^T y$$

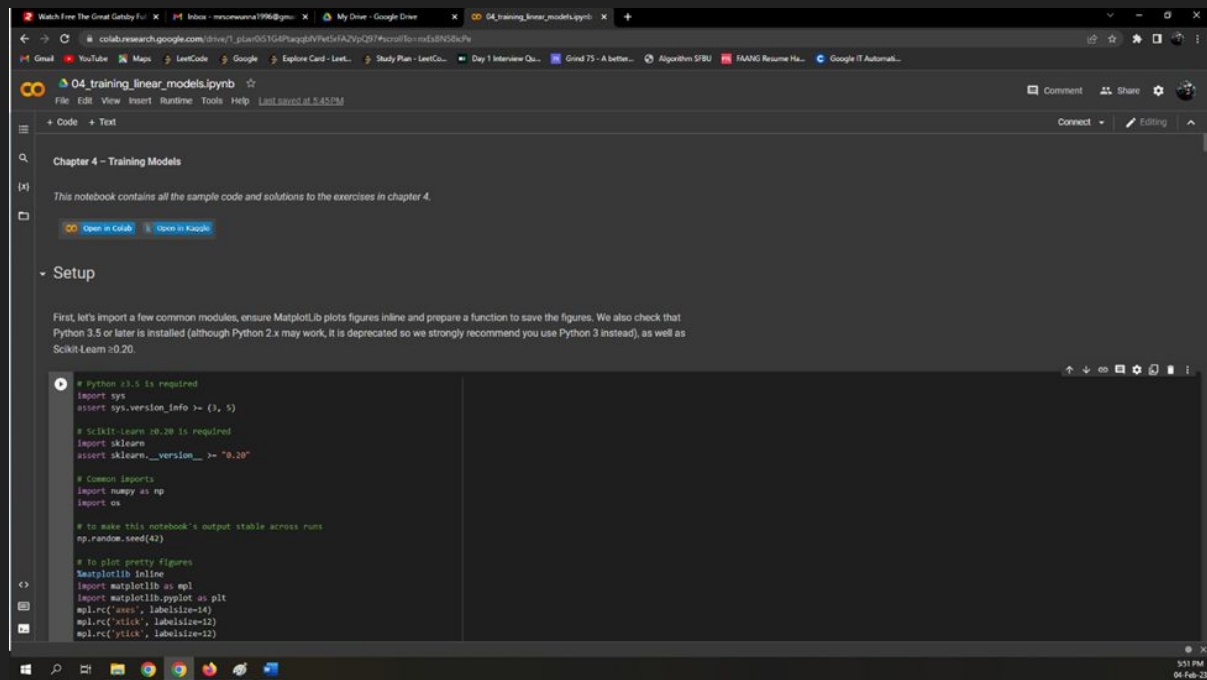
where X is the design matrix that contains the independent variables, T is the transpose operator, and y is the vector of dependent variables. Solving for θ , we get:

$$\theta = (X^T X)^{-1} X^T y$$

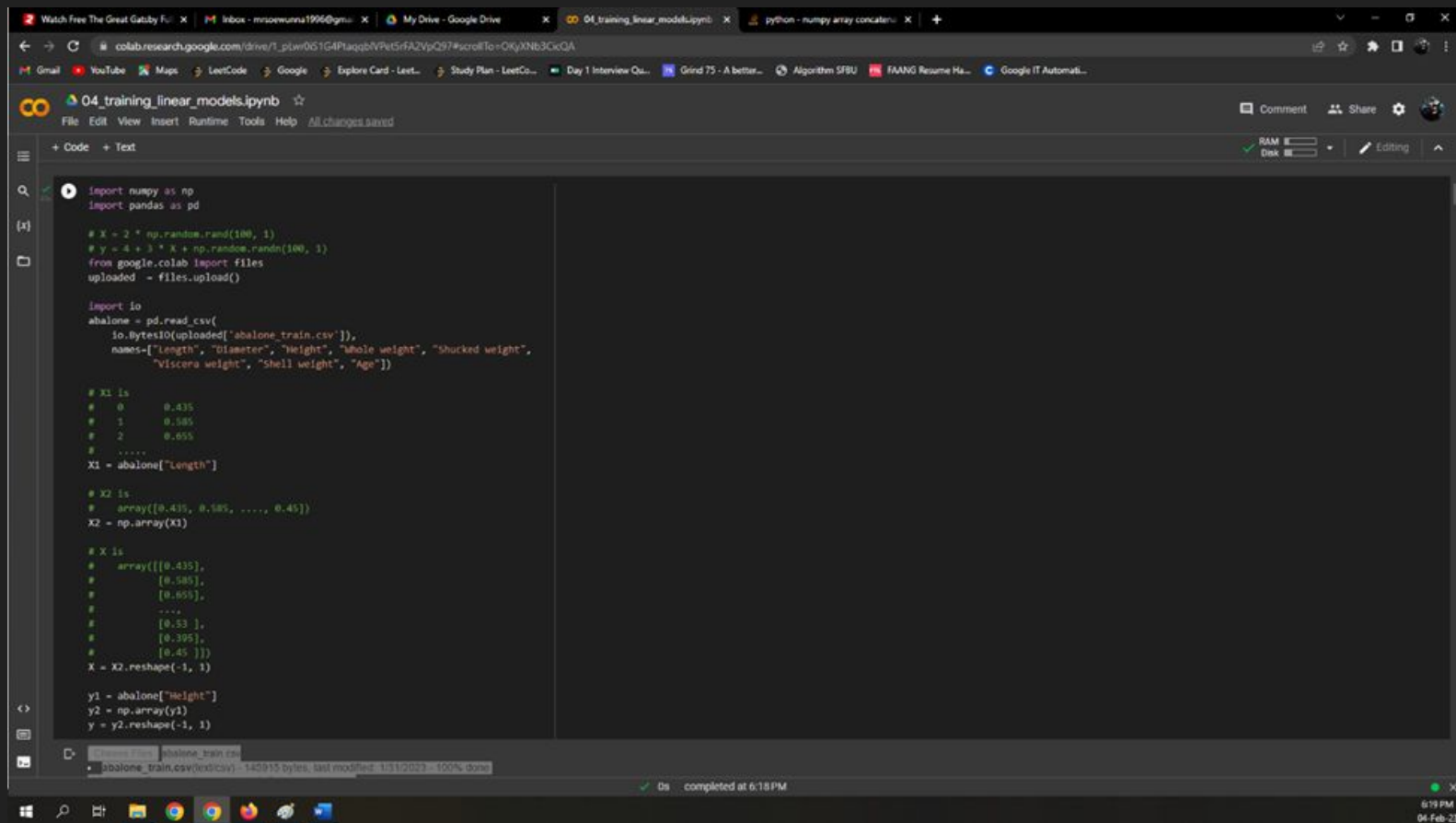
This equation provides the optimal solution for the linear regression problem in a closed-form manner.

5. Google Colab and ipynb

Google Colab is a free online platform for machine learning research and education. It provides access to a Jupyter Notebook-style environment that runs in the cloud, with access to high-performance computing resources, including GPUs and TPUs. This makes it an ideal platform for prototyping and experimenting with machine learning models, as users can quickly implement and train models on large datasets without having to worry about setting up their own hardware infrastructure.



6. Modifying the Code



The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'Watch Free The Great Gatsby Full...', 'Inbox - mrsowunna1996@gmail.com', 'My Drive - Google Drive', '04_training_linear_models.ipynb', and 'python - numpy array concatenation...'. The notebook title is '04_training_linear_models.ipynb'. The code in the cell is as follows:

```
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 ],
#        [0.398],
#        [0.45 ]])
X = X2.reshape(-1, 1)

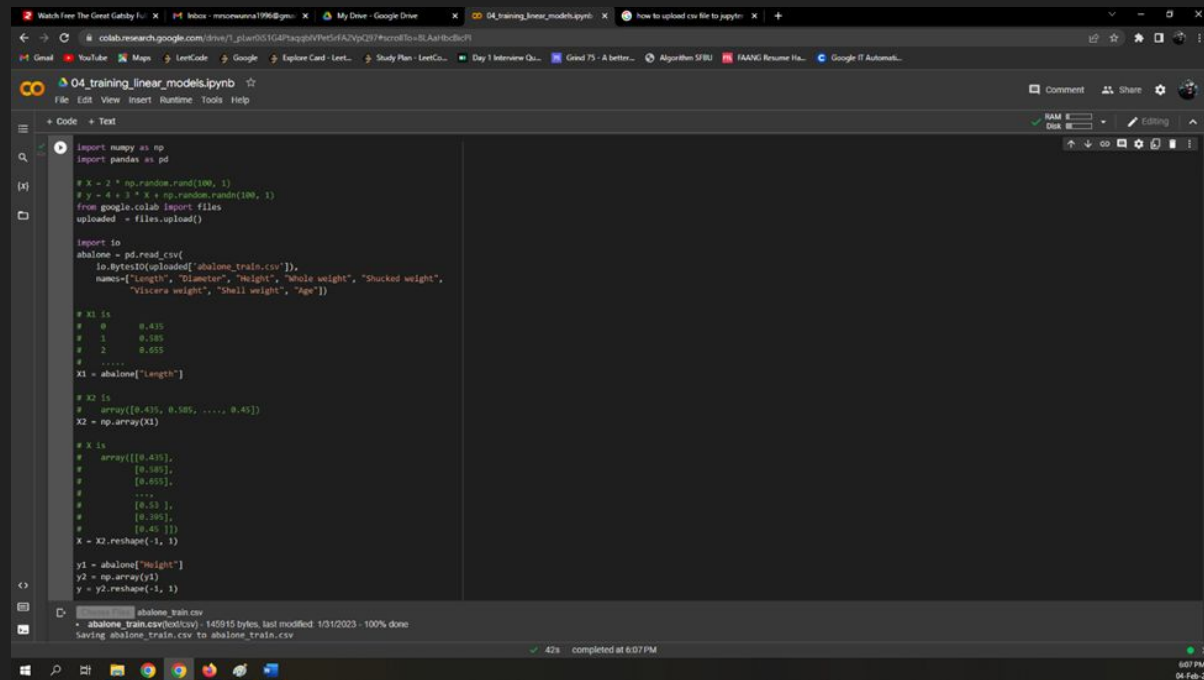
y1 = abalone['Weight']
y2 = np.array(y1)
y = y2.reshape(-1, 1)
```

At the bottom of the notebook, a file named 'abalone_train.csv' is shown as uploaded, with a size of 140915 bytes, last modified on 1/31/2023, and 100% done. The status bar at the bottom indicates '0s completed at 6:18 PM' and the date '04-Feb-23'.

7. Loading CSV File into Colab

CSV (Comma-Separated Values) files are a common format for storing and exchanging data, and they are widely used in machine learning notebooks. The primary use of CSV files in machine learning is to load and process data for training and testing models.

In a machine learning notebook, CSV files can be loaded using various libraries such as Pandas, NumPy, or native Python functions.



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo and various icons for file management and sharing. The notebook title is "O4_training_linear_models.ipynb". The code editor displays the following Python code:

```
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)

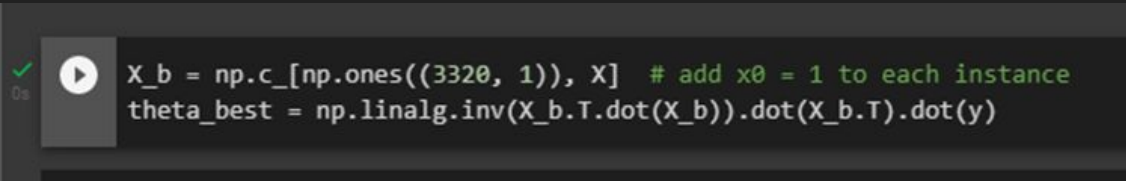
# X is
# array([[0.435],
#        [0.585],
#        [0.655],
#        ...,
#        [0.53 ],
#        [0.389],
#        [0.45 ]])
X = X2.reshape(-1, 1)

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

Below the code editor, a file named "abalone_train.csv" is shown as uploaded. The status bar at the bottom indicates "42s completed at 6:07 PM".

8. Running Result of Entire Notebook

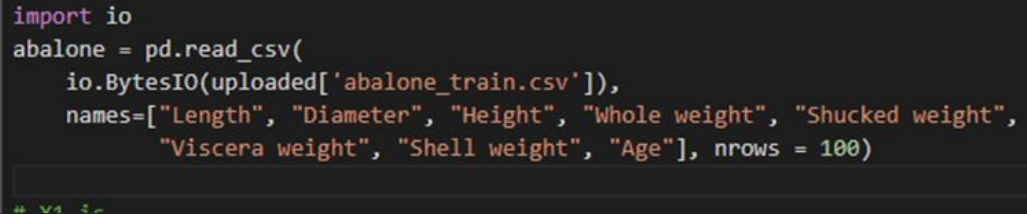
Before running the entire notebook, we have to do either one of the following modifications.

A screenshot of a Jupyter Notebook code cell. On the left, there is a green checkmark and a play button icon. The code cell contains two lines of Python code: `X_b = np.c_[np.ones((3320, 1)), X] # add x0 = 1 to each instance` and `theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)`.

```
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)
```

Changing 100 to 3320 and the problem has been solved.

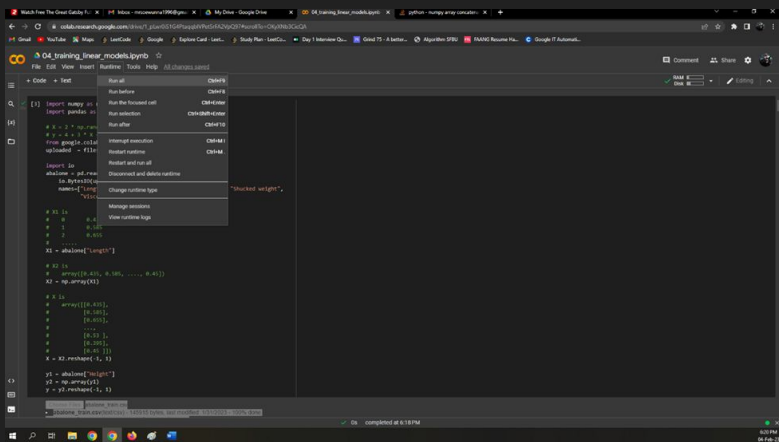
(OR)

A screenshot of a Python code snippet. The code imports the 'io' module and uses 'pd.read_csv' to read a CSV file. It specifies the file path as 'abalone_train.csv' and sets 'nrows = 100'. The 'names' parameter is a list of column names: 'Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell weight', and 'Age'.

```
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"], nrows = 100)
```

We need to define `nrows = 100`.

8. Running Result of Entire Notebook



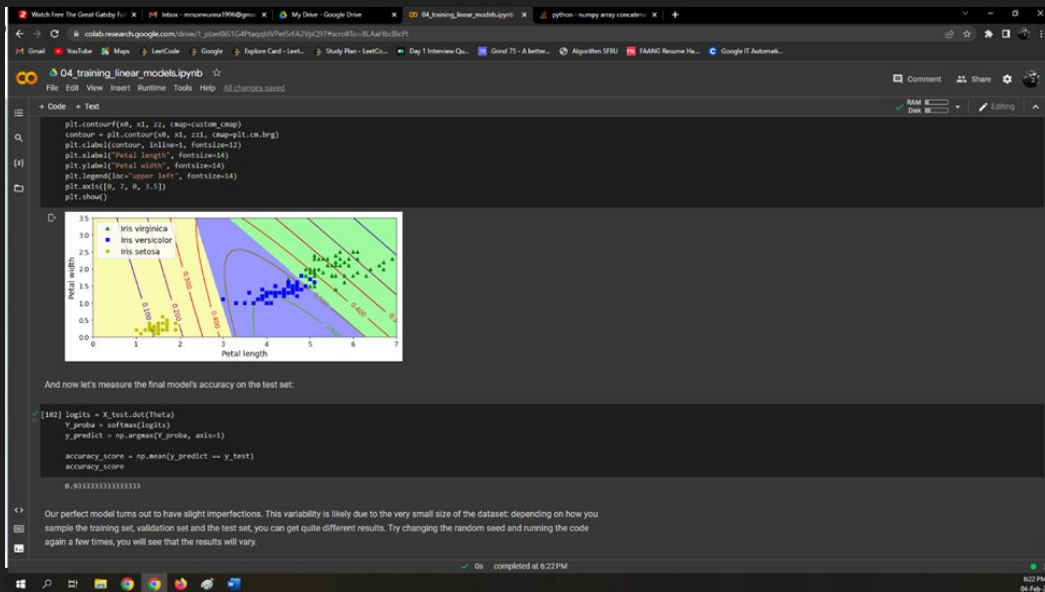
```
# Import numpy as np
import numpy as np

# Load data
from sklearn.datasets import load_iris
iris = load_iris()

# Preprocess data
X = iris.data
y = iris.target

# Train model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X, y)

# Evaluate model
from sklearn.metrics import accuracy_score
X_test, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
y_predict = model.predict(X_test)
accuracy_score = accuracy_score(y_test, y_predict)
```



```
plt.contourf(x0, x1, z1, cmap=cm.set2)
contour = plt.contour(x0, x1, z1, cmap=plt.cm.brg)
plt.xlabel('Petal length', fontsize=12)
plt.ylabel('Petal width', fontsize=12)
plt.legend(loc='upper left', fontsize=14)
plt.axis([0, 7, 0, 3.5])
plt.show()
```

And now let's measure the final model's accuracy on the test set:

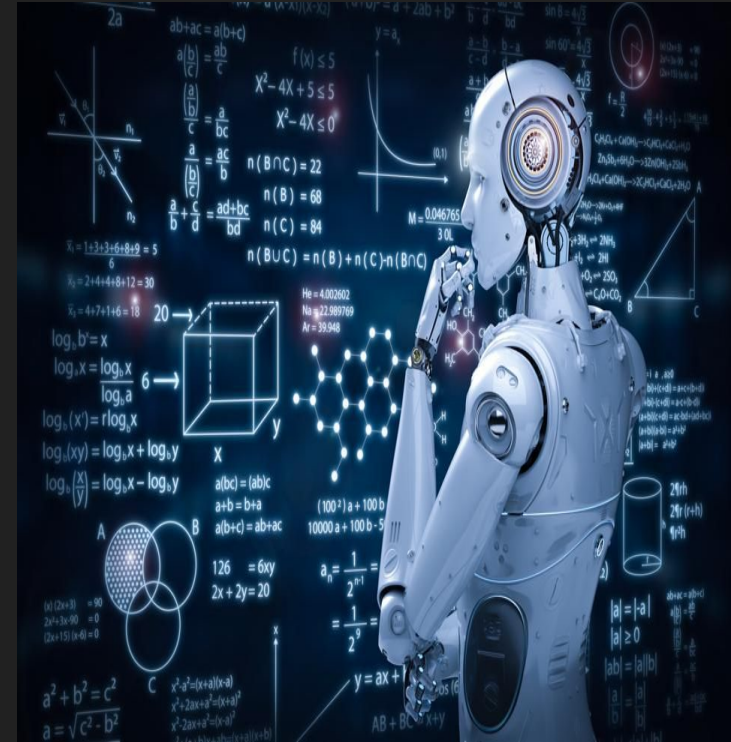
```
[182] logits = X_test.dot(theta)
y_probe = softmax(logits)
y_predict = np.argmax(y_probe, axis=1)
accuracy_score = np.mean(y_predict == y_test)
accuracy_score
```

0.9333333333333333

Our perfect model turns out to have slight imperfections. This variability is likely due to the very small size of the dataset: depending on how you sample the training set, validation set and the test set, you can get quite different results. Try changing the random seed and running the code again a few times, you will see that the results will vary.

9. Conclusion

In conclusion, the normal equation is a useful tool in linear regression, a popular machine learning algorithm used to model the relationship between a dependent variable and one or more independent variables. The normal equation provides a closed-form solution to finding the optimal coefficients for the linear regression model that minimize the mean squared error between the observed and predicted values. The normal equation is computationally efficient when the number of features is small, and it has the advantage of providing a direct solution without the need for iterative optimization. However, its computational cost and memory requirements increase rapidly with the number of features, making it less practical for large-scale problems. Despite these limitations, the normal equation remains an important concept in the field of machine learning and is widely used in practice for solving linear regression problems.



10. References

1. https://hc.labnet.sfbu.edu/~henry/sfbu/course/data_science/algorithm/slide/linear_regression_example.html#lf
2. <https://www.geeksforgeeks.org/ml-normal-equation-in-linear-regression/>
3. <https://www.datacamp.com/tutorial/tutorial-normal-equation-for-linear-regression><https://www.datacamp.com/tutorial/tutorial-normal-equation-for-linear-regression>
4. <https://stackoverflow.com/questions/65545279/linear-regression-why-does-normal-equation-give-huge-error>