Question:

Process

1. Follow the procedure mentioned in Chapter 4 – Training Linear Models to make it work on Colab.
2. Save the abalone_train.cvs to a local drive
   - Note: the abalone_train.cvs has this format
     -
     - names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight",
     - "Viscera weight", "Shell weight", "Age"])
3. Change the process mentioned in Step 1 by reading CVS test data from a local drive : abalone_train.cvs
   - Process
     a. You can modify the code in Linear regression using the Normal Equation. Instead of reading random data
        b.
        c. import numpy as np
        d. X = 2 * np.random.rand(100, 1)
        e. y = 4 + 3 * X + np.random.randn(100, 1)

     You need to read data from a local drive

```
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)

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

f. There is one more line you need to modify to make the complete process work.

Answer:

First of all, we go to Colab and set up woth:

```
# 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)
```

Instead of reading random data with:

```
import numpy as np

X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

Read the data from local with:

```
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)

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

选择文件 abalone_train.csv

And we have a abalone_train.csv file, it used to store the test data and look like

0.435	0.335	0.11	0.334	0.1355	0.0775	0.0965	7
0.585	0.45	0.125	0.874	0.3545	0.2075	0.225	6
0.655	0.51	0.16	1.092	0.396	0.2825	0.37	14
0.545	0.425	0.125	0.768	0.294	0.1495	0.26	16
0.545	0.42	0.13	0.879	0.374	0.1695	0.23	13
0.57	0.45	0.145	0.751	0.2825	0.2195	0.2215	10
0.47	0.36	0.13	0.472	0.182	0.114	0.15	10
0.61	0.45	0.19	1.0805	0.517	0.2495	0.2935	10
0.52	0.425	0.125	0.79	0.372	0.205	0.19	8
0.485	0.39	0.12	0.599	0.251	0.1345	0.169	8
0.625	0.495	0.155	1.025	0.46	0.1945	0.34	9
0.615	0.495	0.16	1.255	0.5815	0.3195	0.3225	12
0.455	0.35	0.14	0.5185	0.221	0.1265	0.135	10
0.475	0.355	0.115	0.5195	0.279	0.088	0.1325	7
0.385	0.3	0.1	0.2895	0.1215	0.063	0.09	7
0.67	0.525	0.165	1.6085	0.682	0.3145	0.4005	11
0.615	0.52	0.15	1.3435	0.629	0.2605	0.345	10
0.52	0.4	0.13	0.5825	0.233	0.1365	0.18	10
0.635	0.495	0.18	1.596	0.617	0.317	0.37	11
0.72	0.575	0.23	2.2695	0.8835	0.3985	0.665	16
0.57	0.435	0.15	0.8295	0.3875	0.156	0.245	10
0.725	0.575	0.24	2.21	1.351	0.413	0.5015	13
0.435	0.35	0.11	0.384	0.143	0.1005	0.125	13
0.685	0.55	0.2	1.7725	0.813	0.387	0.49	11
0.575	0.445	0.145	0.876	0.3795	0.1615	0.27	10
0.575	0.435	0.13	1.0105	0.368	0.222	0.32	10
0.625	0.445	0.16	1.09	0.46	0.2965	0.304	11
0.355	0.27	0.075	0.1775	0.079	0.0315	0.054	6
0.565	0.48	0.175	0.957	0.3885	0.215	0.275	18
0.47	0.365	0.12	0.582	0.29	0.092	0.146	8

After upload the .csv file it will look like this

选择文件 abalone_train.csv

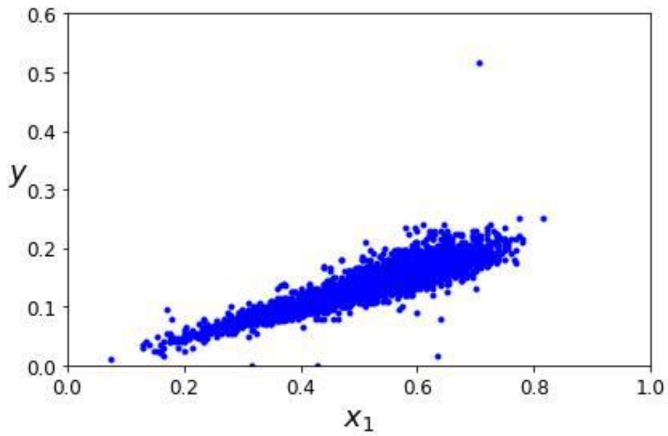
- **abalone_train.csv**(application/vnd.ms-excel) - 145915 bytes, last modified: 2021/5/26 - 100% done

```
Saving abalone_train.csv to abalone_train (3).csv
```

Then we create the data show in a plot graph

```
[41] plt.plot(X, y, "b.")
     plt.xlabel("$x_1$", fontsize=18)
     plt.ylabel("$y$", rotation=0, fontsize=18)
     plt.axis([0, 1, 0, 0.6])
     save_fig("generated_data_plot")
     plt.show()
```

```
Saving figure generated_data_plot
```



Get the linear regression equations' values and graph

```
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)
```

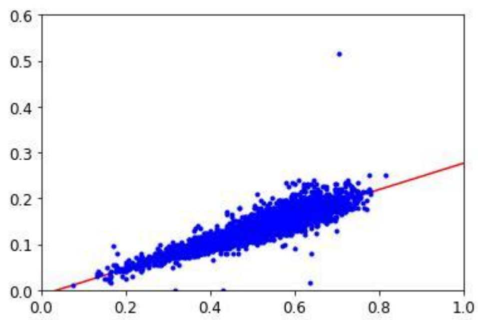
```
[43] theta_best
```

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

```
[44] 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]])
```

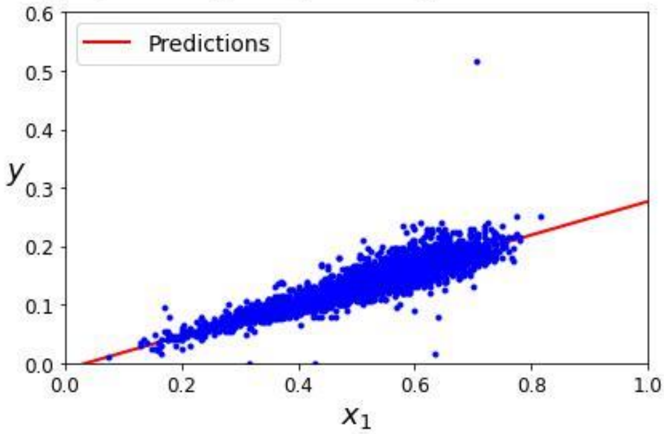
```
plt.plot(X_new, y_predict, "r-")
plt.plot(X, y, "b.")
plt.axis([0, 1, 0, 0.6])
plt.show()
```



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, 1, 0, 0.6])
save_fig("linear_model_predictions_plot")
plt.show()
```

```
Saving figure linear_model_predictions_plot
```




```
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]]))
```

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

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

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 $X^{+}y$, where X^{+} is the *pseudoinverse* of X

This is the values of linear regression base on these data

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

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