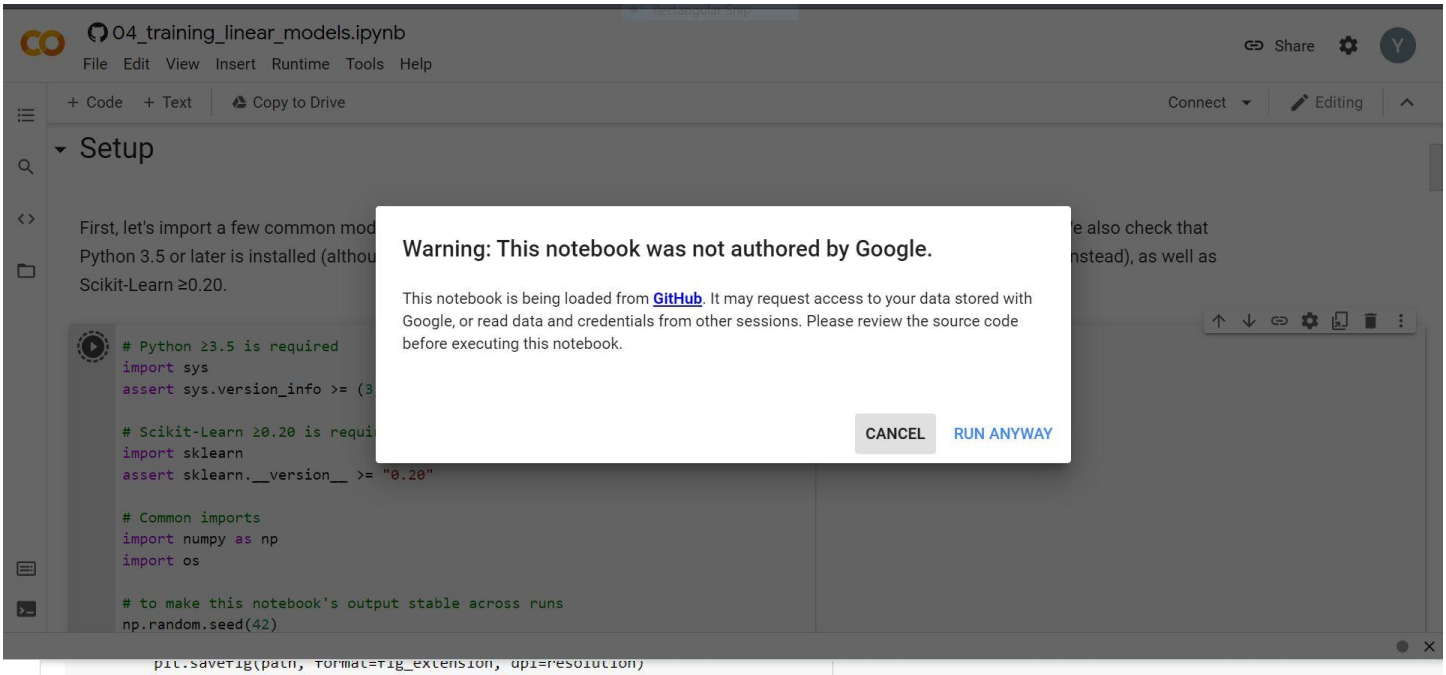
Linear Regression using Normal Equation

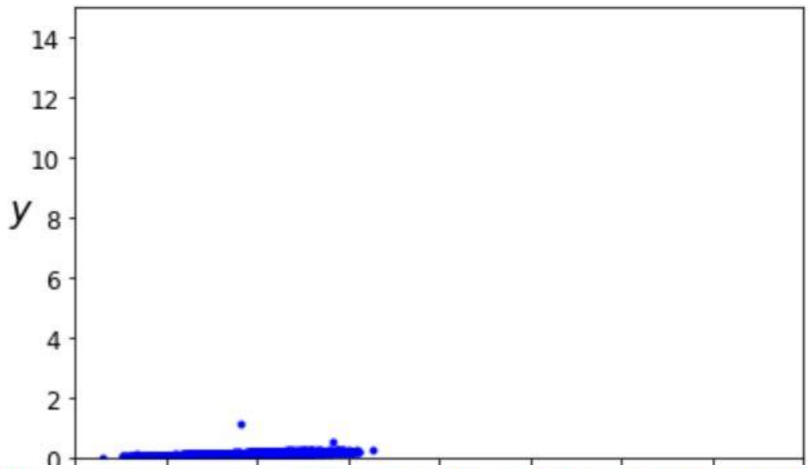


Linear regression using the Normal Equation

```
[9] 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
```

```
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
```




```
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)
```

```
[33] theta_best
```

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

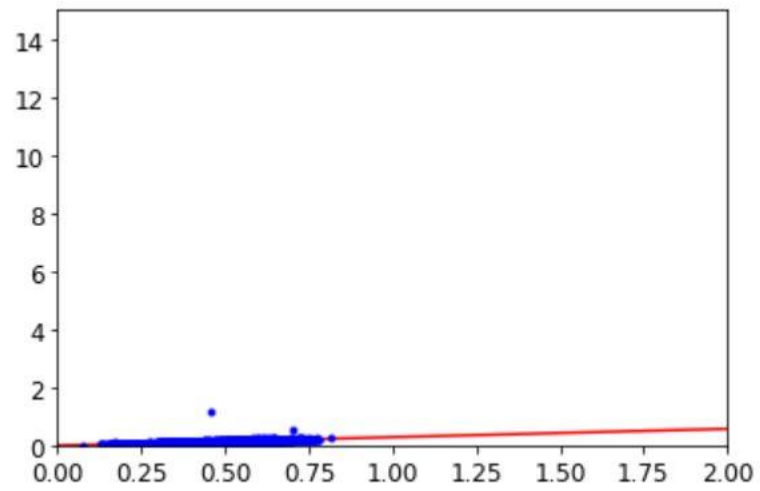
```
[34] X_new = np.array([[0], [2]])
     X_new_b = np.c_[np.ones((2, 1)), X_new]  # add x0 = 1 to each instance
     y_predict = X_new_b.dot(theta_best)
     y_predict
```

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

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

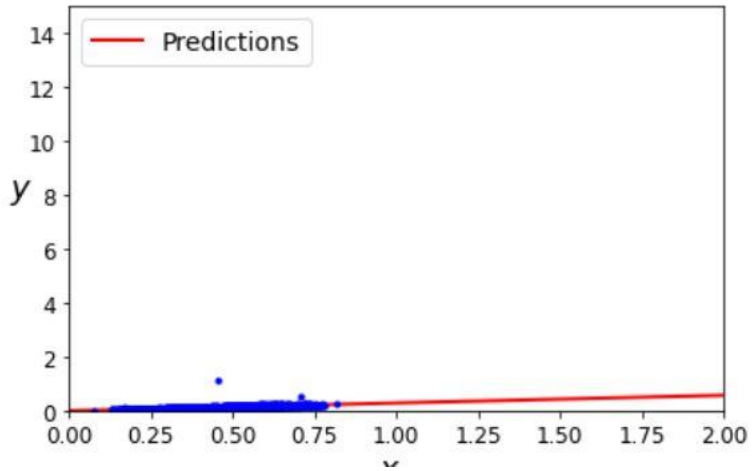
Change parameter from 100 to 3320, the row size in the file.

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




```
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
```



```
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]]))
```

```
[38] 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]])
```


```
[38] 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:

```
[39] 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:

```
[40] np.linalg.pinv(X_b).dot(y)
```

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