Input data

INTRODUCTION TO TENSORFLOW IN PYTHON

Isaiah Hull
Economist

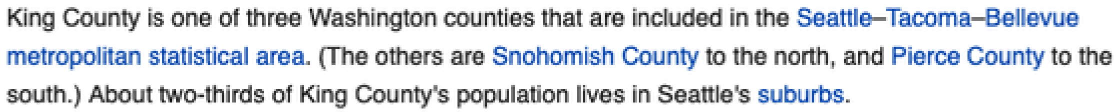
NUMERIC DATA

```
       price  bedrooms  bathrooms  sqft_living
    221900.0         3       1.00         1180
    538000.0         3       2.25         2570
    180000.0         2       1.00          770
    604000.0         4       3.00         1960
    510000.0         3       2.00         1680
   1225000.0         4       4.50         5420
    257500.0         3       2.25         1715
    291850.0         3       1.50         1060
    229500.0         3       1.00         1780
    323000.0         3       2.50         1890
    662500.0         3       2.50         3560
    468000.0         2       1.00         1160
```

IMAGE DATA

```
[[164, 161, 159, ...,  79,  87, 131],
 [161, 162, 164, ...,  98, 117, 146],
 [147, 151, 151, ..., 155, 165, 167],  93, 137],
 ...,                                  123, 152],
 [178, 172, 176, ..., 178, 175, 123], 172, 174],  91, 135],
 [ 84,  82,  86, ..., 168, 192, 175],            119, 148],
 [157, 158, 162, ..., 157, 179, 174]] 180, 128], 165, 167],
             [ 88,  87,  88, ..., 169, 194, 177],
             [159, 158, 161, ..., 156, 180, 175]] 176, 122],
                         [ 87,  83,  85, ..., 164, 189, 172],
                         [158, 156, 159, ..., 151, 174, 169]]
```

TEXT DATA



King County is one of three Washington counties that are included in the Seattle–Tacoma–Bellevue metropolitan statistical area. (The others are Snohomish County to the north, and Pierce County to the south.) About two-thirds of King County's population lives in Seattle's suburbs.

```
0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Importing data for use in TensorFlow

- **Data can be imported using `tensorflow`**
  - Useful for managing complex pipelines
  - Not necessary for this chapter
- **Simpler option used in this chapter**
  - Import data using `pandas`
  - Convert data to `numpy` array
  - Use in `tensorflow` without modification

How to import and convert data

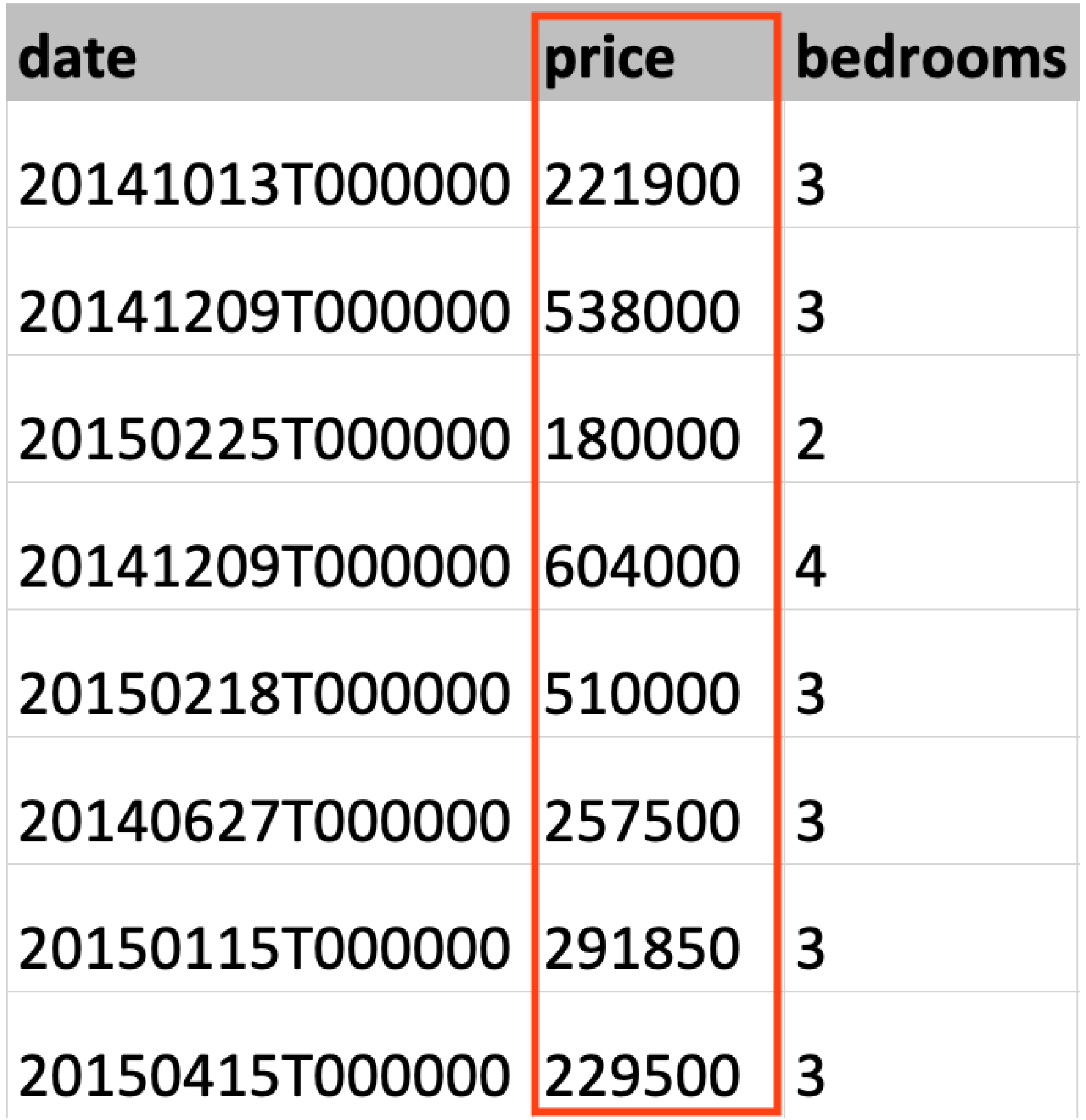
```python
# Import numpy and pandas
import numpy as np
import pandas as pd

# Load data from csv
housing = pd.read_csv('kc_housing.csv')

# Convert to numpy array
housing = np.array(housing)
```

- We will focus on data stored in csv format in this chapter
- Pandas also has methods for handling data in other formats
    - E.g. `read_json()` , `read_html()` , `read_excel()`

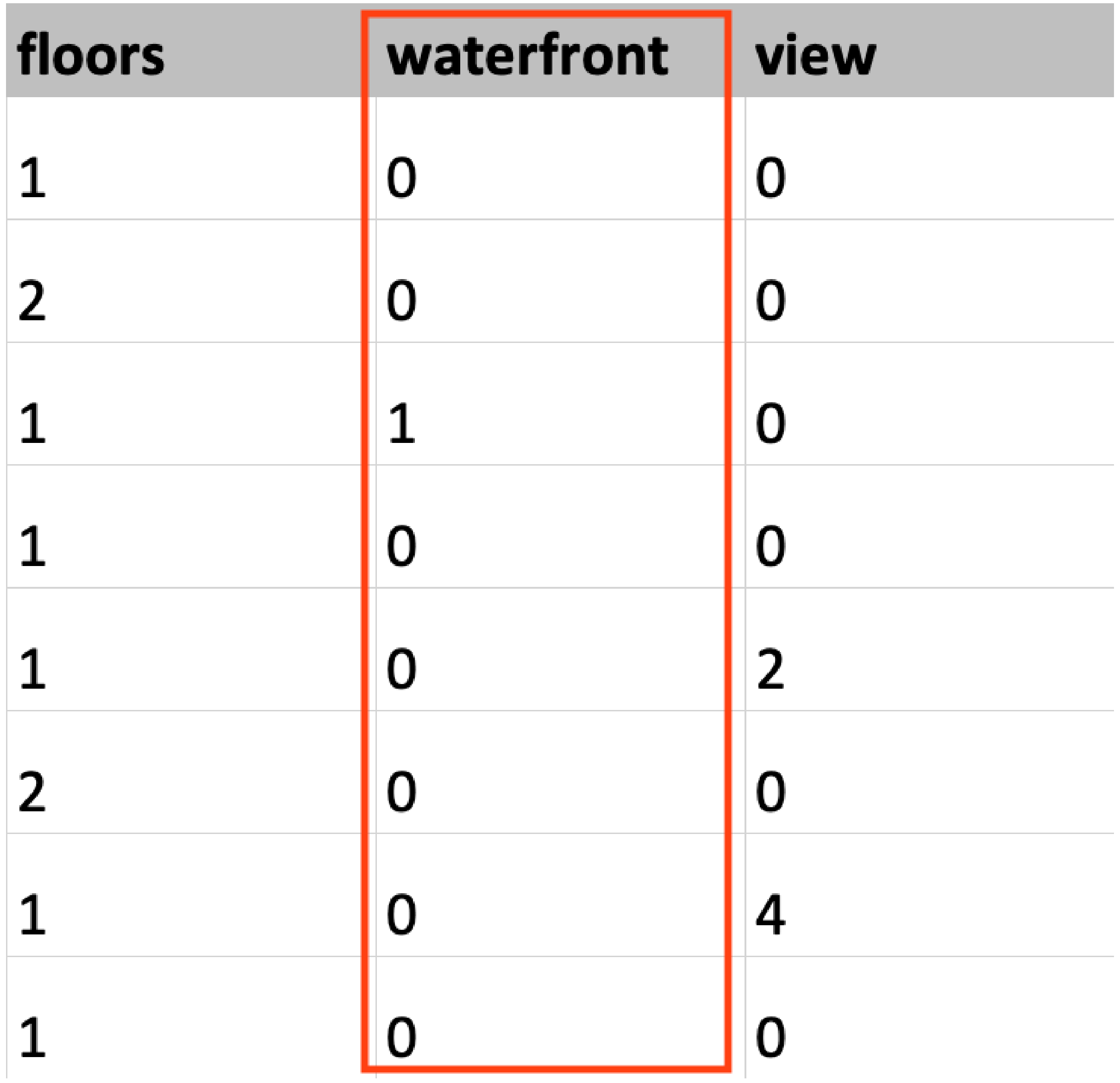
Parameters of read_csv()

Parameter	Description	Default
`filepath_or_buffer`	Accepts a file path or a URL.	`None`
`sep`	Delimiter between columns.	`,`
`delim_whitespace`	Boolean for whether to delimit whitespace.	`False`
`encoding`	Specifies encoding to be used if any.	`None`

Using mixed type datasets



date	price	bedrooms
20141013T000000	221900	3
20141209T000000	538000	3
20150225T000000	180000	2
20141209T000000	604000	4
20150218T000000	510000	3
20140627T000000	257500	3
20150115T000000	291850	3
20150415T000000	229500	3



floors	waterfront	view
1	0	0
2	0	0
1	1	0
1	0	0
1	0	2
2	0	0
1	0	4
1	0	0

Setting the data type

```
# Load KC dataset
housing = pd.read_csv('kc_housing.csv')

# Convert price column to float32
price = np.array(housing['price'], np.float32)

# Convert waterfront column to Boolean
waterfront = np.array(housing['waterfront'], np.bool)
```

Setting the data type

```
# Load KC dataset
housing = pd.read_csv('kc_housing.csv')

# Convert price column to float32
price = tf.cast(housing['price'], tf.float32)

# Convert waterfront column to Boolean
waterfront = tf.cast(housing['waterfront'], tf.bool)
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON

Loss functions

INTRODUCTION TO TENSORFLOW IN PYTHON

Isaiah Hull
Economist

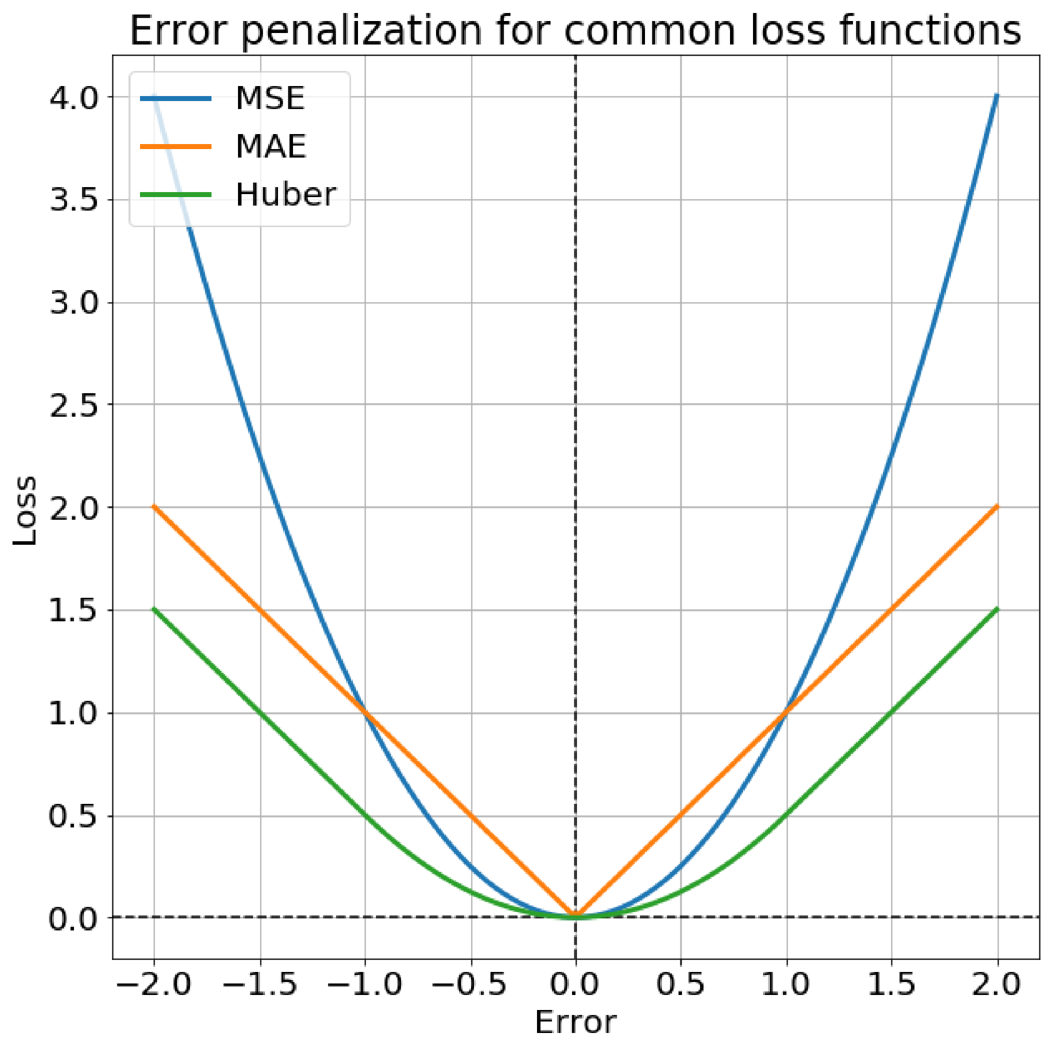
Introduction to loss functions

- **Fundamental `tensorflow` operation**
    - Used to train a model
    - Measure of model fit
- **Higher value -> worse fit**
    - Minimize the loss function

Common loss functions in TensorFlow

- **TensorFlow has operations for common loss functions**
  - Mean squared error (MSE)
  - Mean absolute error (MAE)
  - Huber error
- **Loss functions are accessible from `tf.keras.losses()`**
  - `tf.keras.losses.mse()`
  - `tf.keras.losses.mae()`
  - `tf.keras.losses.Huber()`

Why do we care about loss functions?



- **MSE**
  - Strongly penalizes outliers
  - High sensitivity near minimum
- **MAE**
  - Scales linearly with size of error
  - Low sensitivity near minimum
- **Huber**
  - Similar to MSE near minimum
  - Similar to MAE away from minimum

Defining a loss function

```python
# Import TensorFlow under standard alias
import tensorflow as tf


# Compute the MSE loss
loss = tf.keras.losses.mse(target, predictions)
```

Defining a loss function

```python
# Define a loss function to compute the MSE
def loss_function(intercept, slope, target, features):
    # Compute the predictions for a linear model
    predictions = intercept + features*slope

    # Return the loss
    return tf.keras.losses.mse(target, predictions)
```

```python
# Compute the loss for given input data and model parameters
loss_function(intercept, slope, prices, size)
```

Common loss functions

Loss	Name	Operation
MSE	Mean Squared Error	`tf.keras.losses.mse()`
MAE	Mean Absolute Error	`tf.keras.losses.mae()`
Huber	Huber Error	`tf.keras.losses.Huber()`

Other loss functions

Loss	Name	Operation
MAPE	Mean Absolute Percentage Error	`tf.keras.losses.mape()`
MSLE	Mean Squared Logarithmic Error	`tf.keras.losses.msle()`

- **MAPE** 10% error for value of 100 treated same as 10% error for value of 1
- **MSLE** Compress size of losses and preserve ordering

Let's practice!

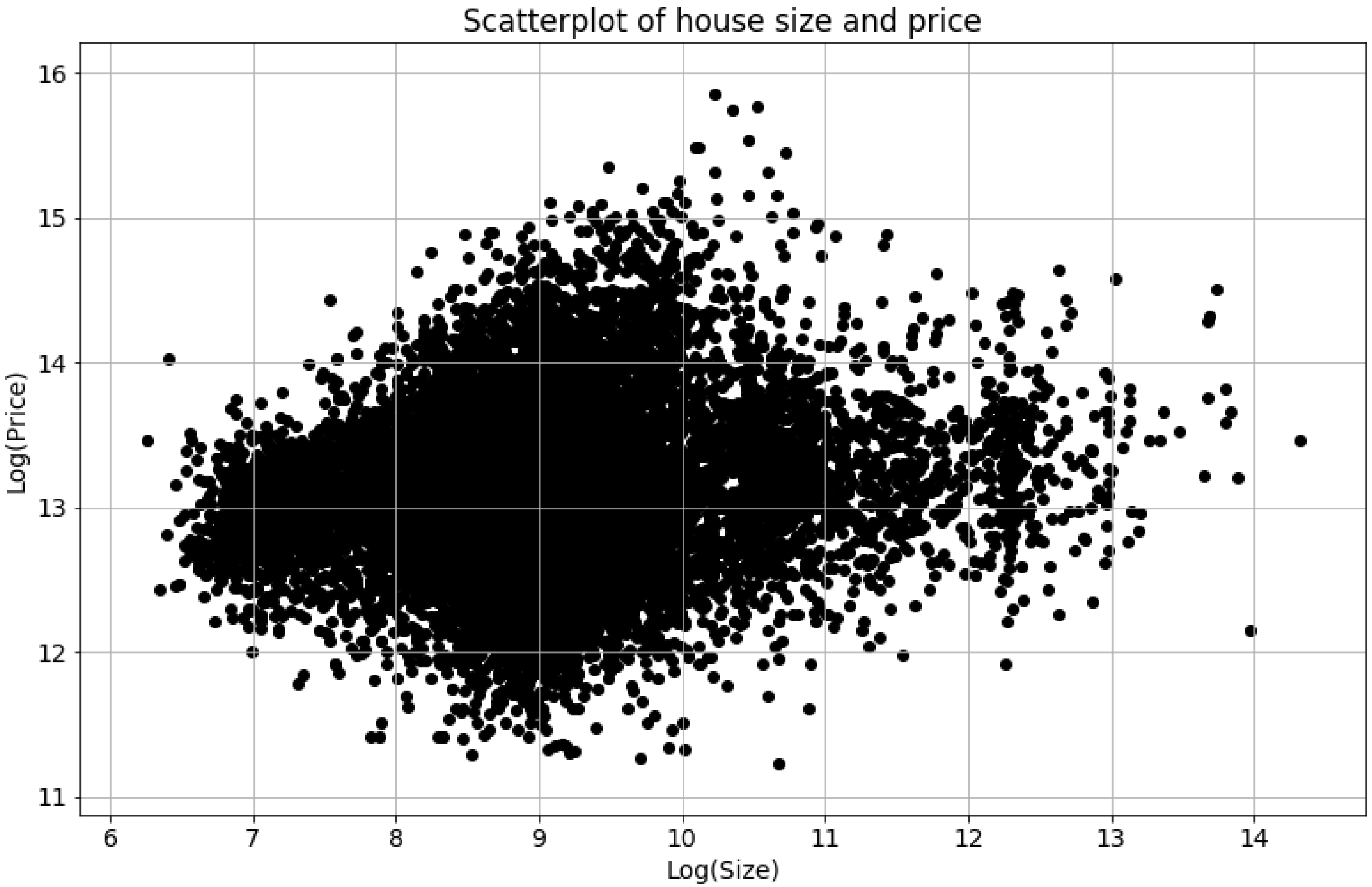
INTRODUCTION TO TENSORFLOW IN PYTHON

Linear regression

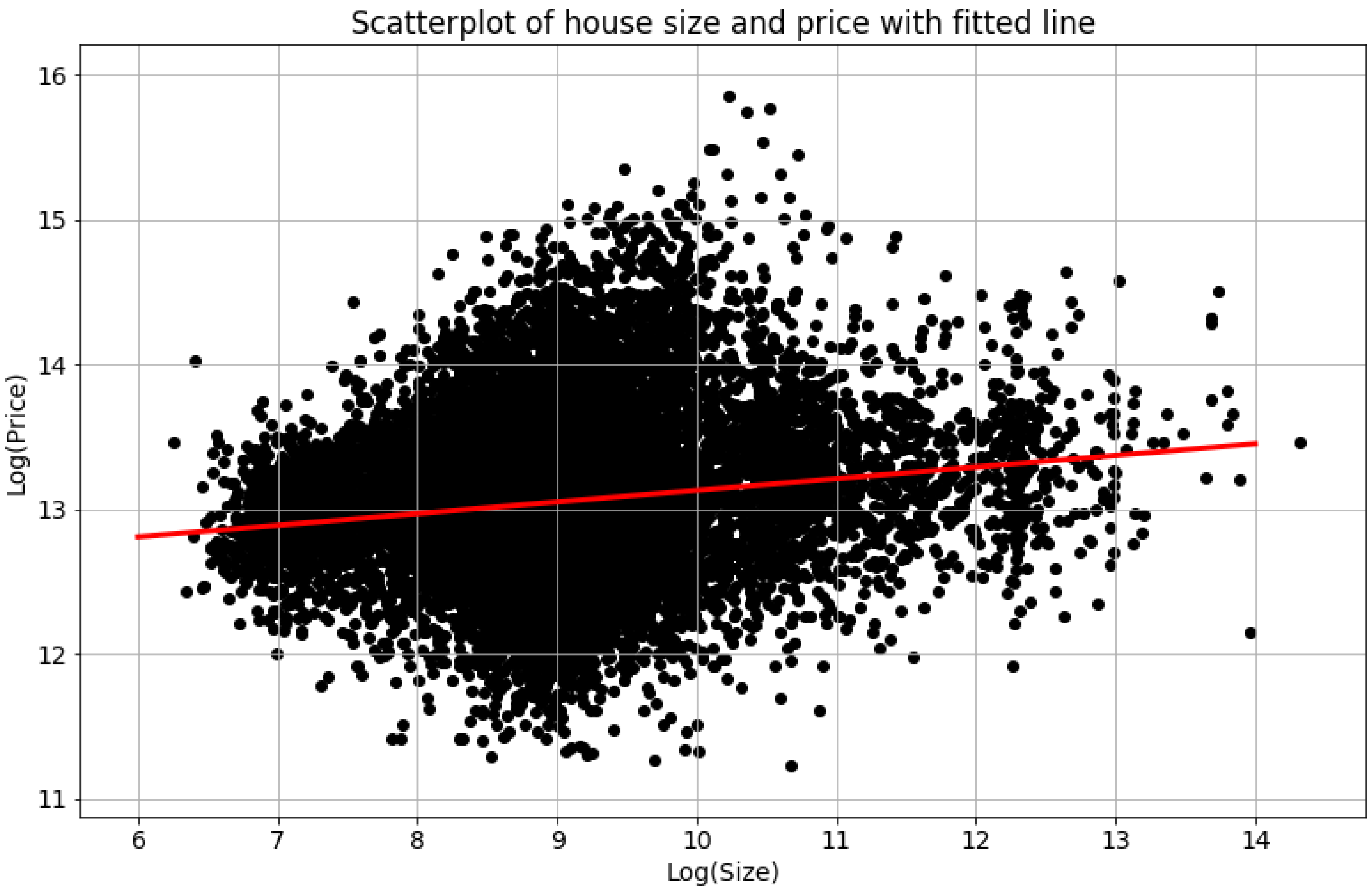
INTRODUCTION TO TENSORFLOW IN PYTHON

Isaiah Hull
Economist

What is a linear regression?



What is a linear regression?



The linear regression model

- **A linear regression model assumes a linear relationship:**
    - $price = intercept + size * slope + error$
- **This is an example of a univariate regression.**
    - There is only one feature, `size` .
- **Multiple regression models have more than one feature.**
    - E.g. `size` and `location`

Linear regression in TensorFlow

```
# Define the targets and features
price = np.array(housing['price'], np.float32)
size = np.array(housing['sqft_living'], np.float32)

# Define the intercept and slope
intercept = tf.Variable(0.1, np.float32)
slope = tf.Variable(0.1, np.float32)
```

```
# Compute the predicted values and loss function
def loss_function(intercept, slope, size, price):
    predictions = intercept + size*slope
    return tf.keras.losses.mse(price, predictions)
```

Linear regression in TensorFlow

```python
# Define an optimization operation
opt = tf.keras.optimizers.Adam()
```

```python
# Minimize the loss function and print the loss
for j in range(1000):
    opt.minimize(lambda: loss_function(intercept, slope, size, price),\
    var_list=[intercept, slope])
    print(loss_function(intercept, slope, size, price))
```

```
tf.Tensor(10.909373, shape=(), dtype=float32)
...
tf.Tensor(0.15479447, shape=(), dtype=float32)
```

```python
# Print the trained parameters
print(intercept.numpy(), slope.numpy())
```

Let's practice!

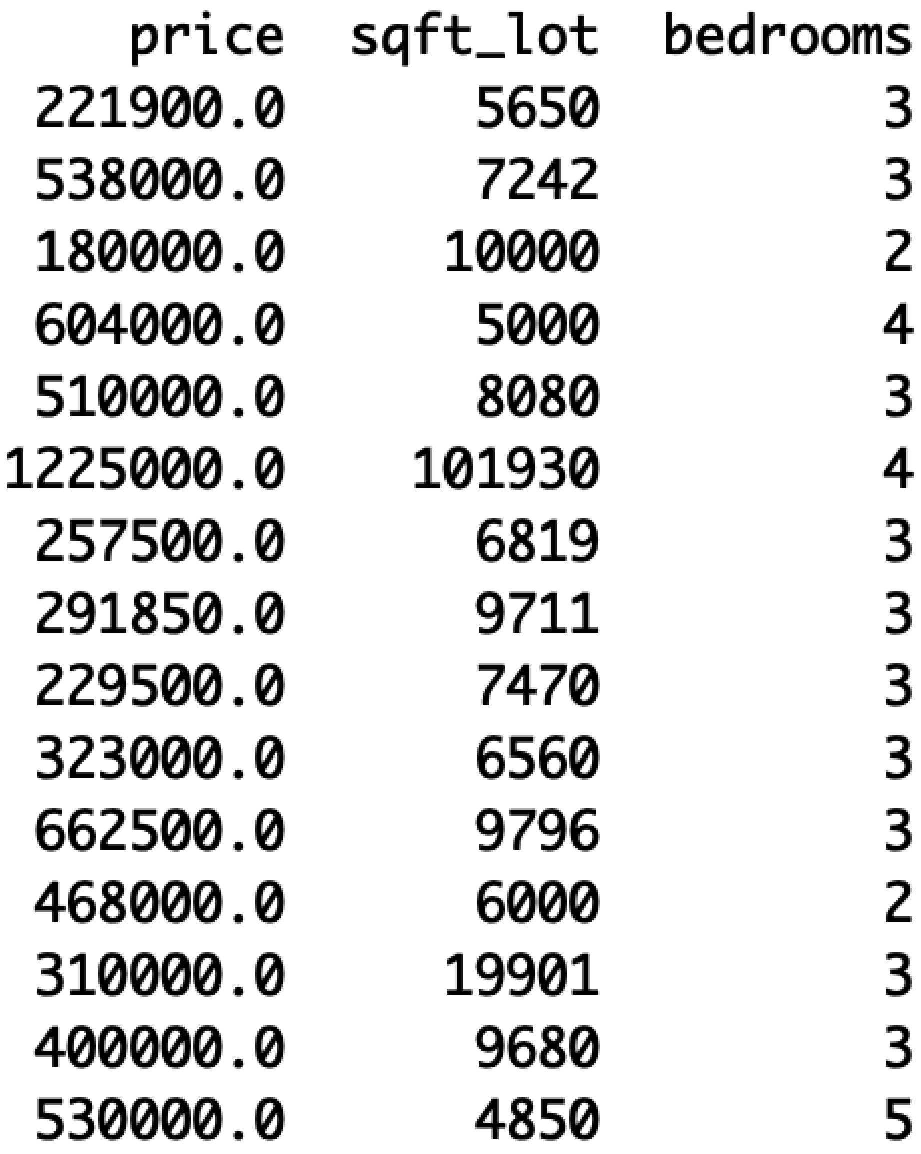
INTRODUCTION TO TENSORFLOW IN PYTHON

Batch training

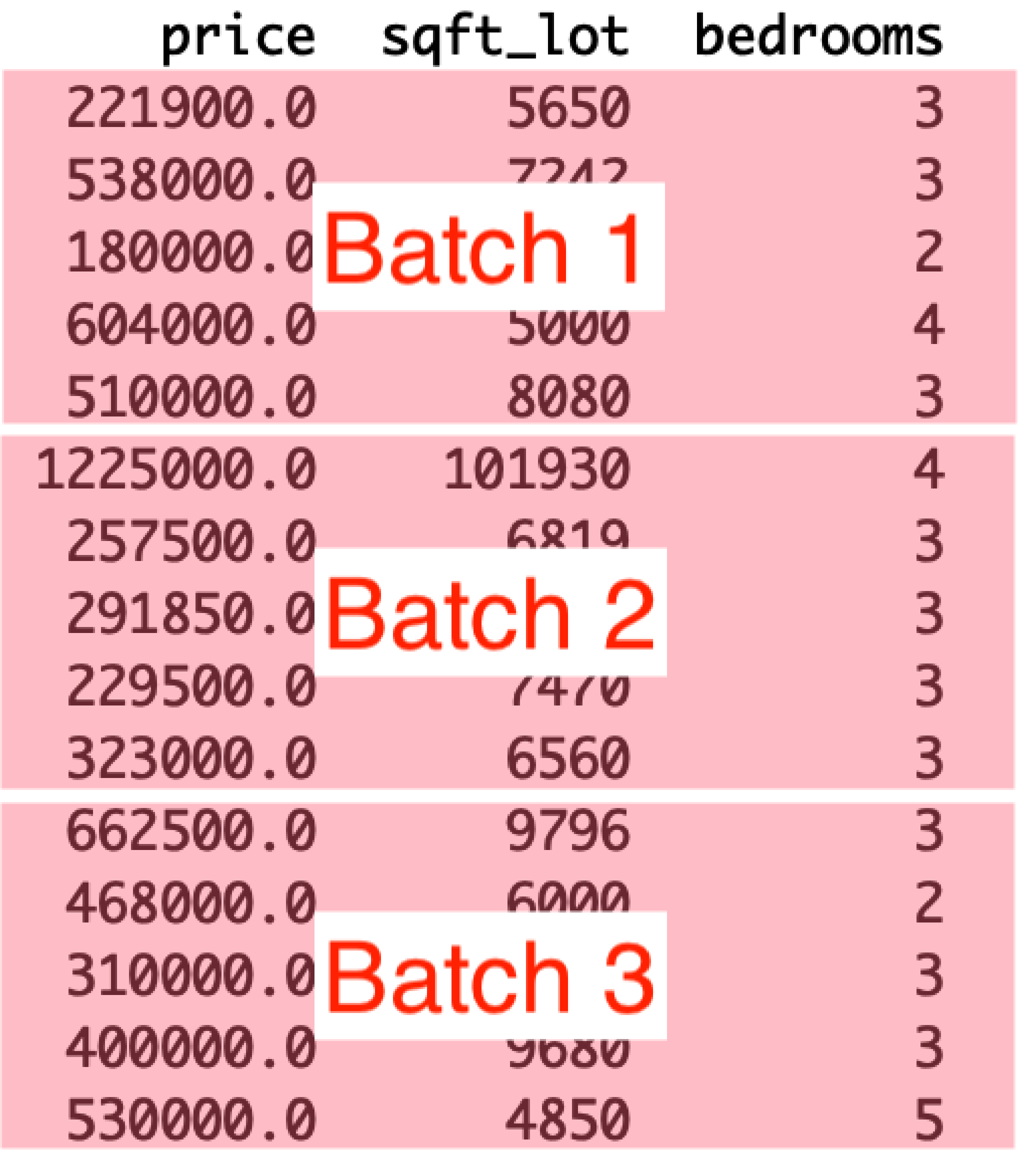
INTRODUCTION TO TENSORFLOW IN PYTHON

Isaiah Hull
Economist

What is batch training?



```
       price  sqft_lot  bedrooms
    221900.0      5650         3
    538000.0      7242         3
    180000.0     10000         2
    604000.0      5000         4
    510000.0      8080         3
   1225000.0    101930         4
    257500.0      6819         3
    291850.0      9711         3
    229500.0      7470         3
    323000.0      6560         3
    662500.0      9796         3
    468000.0      6000         2
    310000.0     19901         3
    400000.0      9680         3
    530000.0      4850         5
```



```
       price  sqft_lot  bedrooms
    221900.0      5650         3
    538000.0      7242         3
    180000.0     10000         2
    604000.0      5000         4
    510000.0      8080         3
   1225000.0    101930         4
    257500.0      6819         3
    291850.0      9711         3
    229500.0      7470         3
    323000.0      6560         3
    662500.0      9796         3
    468000.0      6000         2
    310000.0     19901         3
    400000.0      9680         3
    530000.0      4850         5
```

The chunksize parameter

- `pd.read_csv()` allows us to load data in batches
  - Avoid loading entire dataset
  - `chunksize` parameter provides batch size

```
# Import pandas and numpy
import pandas as pd
import numpy as np

# Load data in batches
for batch in pd.read_csv('kc_housing.csv', chunksize=100):
    # Extract price column
    price = np.array(batch['price'], np.float32)

    # Extract size column
    size = np.array(batch['size'], np.float32)
```

Training a linear model in batches

```
# Import tensorflow, pandas, and numpy
import tensorflow as tf
import pandas as pd
import numpy as np
```

```
# Define trainable variables
intercept = tf.Variable(0.1, tf.float32)
slope = tf.Variable(0.1, tf.float32)
```

Training a linear model in batches

```python
# Compute predicted values and return loss function
def loss_function(intercept, slope, features, target):
    predictions = intercept + features*slope
    return tf.keras.losses.mse(target, predictions)
```

```python
# Define optimization operation
opt = tf.keras.optimizers.Adam()
```

Training a linear model in batches

```
# Load the data in batches from pandas
for batch in pd.read_csv('kc_housing.csv', chunksize=100):
    # Extract the target and feature columns
    price_batch = np.array(batch['price'], np.float32)
    size_batch = np.array(batch['lot_size'], np.float32)

    # Minimize the loss function
    opt.minimize(lambda: loss_function(intercept, slope, size_batch,\
    price_batch), var_list=[intercept, slope])
```

```
# Print parameter values
print(intercept.numpy(), slope.numpy())
```

Full sample versus batch training

- **Full Sample**
  1. One step per epoch
  2. Accepts dataset without modification
  3. Limited by memory

- **Batch Training**
  1. Multiple steps per epoch
  2. Requires division of dataset
  3. No limit on dataset size

Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON