# Deep Learning with Keras

ACTL3143 & ACTL5111 Deep Learning for Actuaries
Patrick Laub

# Lecture Outline

- **California House Price Prediction**

- EDA & Baseline Model

- Our First Neural Network

- Force positive predictions

- Preprocessing

- Early Stopping

- Quiz

# Data science always starts with the data!

The target variable is the median house value for California districts, expressed in $100,000's. This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).



Dall–E's rendition of the this dataset.

Source: Scikit-learn documentation.

# Columns

- `MedInc` median income in block group

- `HouseAge` median house age in block group

- `AveRooms` average number of rooms per household

- `AveBedrms` average # of bedrooms per household

- `Population` block group population

- `AveOccup` average number of household members

- `Latitude` block group latitude

- `Longitude` block group longitude

- `MedHouseVal` median house value (**target**)

Source: Scikit-learn documentation.

# Import the data

```
1  from sklearn.datasets import fetch_california_housing
2
3  features, target = fetch_california_housing(
4      as_frame=True, return_X_y=True)
5  features
```

|       | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Lon |
|-------|--------|----------|----------|-----------|------------|----------|----------|-----|
| 0     | 8.3252 | 41.0     | 6.984127 | 1.023810  | 322.0      | 2.555556 | 37.88    | −122 |
| 1     | 8.3014 | 21.0     | 6.238137 | 0.971880  | 2401.0     | 2.109842 | 37.86    | −122 |
| 2     | 7.2574 | 52.0     | 8.288136 | 1.073446  | 496.0      | 2.802260 | 37.85    | −122 |
| ...   | ...    | ...      | ...      | ...       | ...        | ...      | ...      | ... |
| 20637 | 1.7000 | 17.0     | 5.205543 | 1.120092  | 1007.0     | 2.325635 | 39.43    | −121 |
| 20638 | 1.8672 | 18.0     | 5.329513 | 1.171920  | 741.0      | 2.123209 | 39.43    | −121 |
| 20639 | 2.3886 | 16.0     | 5.254717 | 1.162264  | 1387.0     | 2.616981 | 39.37    | −121 |

20640 rows × 8 columns

# What is the target?

```
1  target
```

```
0          4.526
1          3.585
2          3.521
           ...
20637      0.923
20638      0.847
20639      0.894
Name: MedHouseVal, Length: 20640, dtype:
float64
```

Why predict this? Let's pretend we are these guys.



## The Silicon Valley Elite Who Want to Build a City From Scratch

A mysterious company has spent $800 million in an effort to buy thousands of acres of San Francisco Bay Area land. The people behind the deals are said to be a who's who of the tech industry.

Share full article    1.8K

From left, Michael Moritz, Reid Hoffman, Marc Andreessen and Chris Dixon, four prominent Silicon Valley investors, have backed Flannery Associates.  Bloomberg; The New York Times; Clara Mokri for The New York Times; Getty Images; Reuters

Source: Dougherty and Griffith (2023), The Silicon Valley Elite Who Want to Build a City From Scratch, New York Times.

# An entire ML project



**Deployment**

Actuaries bring a holistic systems focus to help optimise deployment, maintenance and governance of the data science and AI pipeline, end-to-end and with ethics and fairness always at the fore.

**New problem to solve**

Evaluation and continuous evolution through actuarial oversight and machine learning.

**Commercial understanding**

Effective data science is anchored in strong commercial acumen. Actuaries engage key stakeholders to diagnose the challenge and identify where data innovation will unlock tangible value.

**Ethics & fairness**

**Modelling**

Actuaries have been producing great data models for decades and it's a central feature of modern actuarial training. Actuaries are trained to ensure technical models will deliver the promised business uplift on deployment.

**Acquire, cleanse, curate, validate**

Actuaries enhance pre-analytics, working closely with data engineers and data scientists, by offering rigorous technical, analytical and statistical skills to help validate data breadth and quality and the technical and commercial viability of the proposed solution.

ML life cycle

Source: Actuaries Institute, Do Data Better.

# Questions to answer in ML project

You fit a few models to the training set, then ask:

1. **(Selection)** Which of these models is the best?

2. **(Future Performance)** How good should we expect the final model to be on unseen data?

# Set aside a fraction for a test set

```
1  from sklearn.model_selection import train_test_split
2
3  X_train, X_test, y_train, y_test = train_test_split(
4      features, target, random_state=42
5  )
```
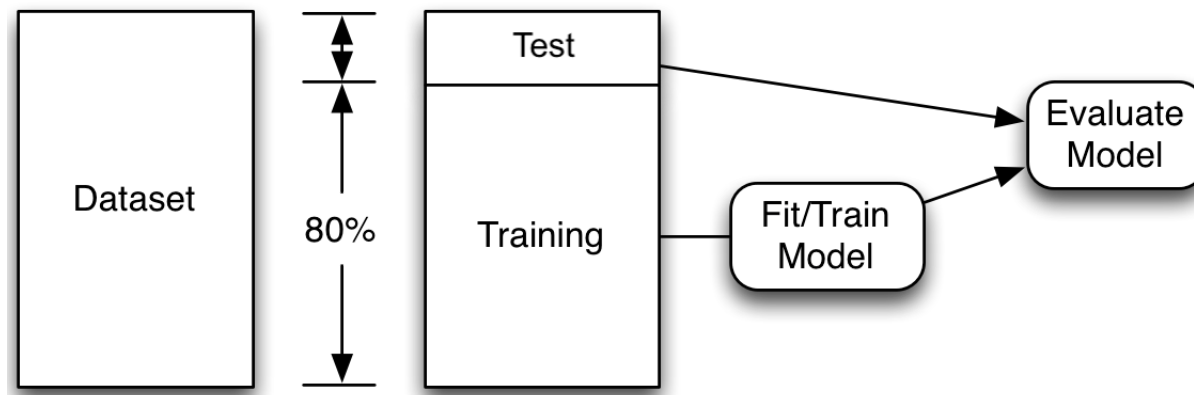


Illustration of a typical training/test split.

Note: Compare X_/y_ names, capitals & lowercase.



Our use of sklearn.

Adapted from: Heaton (2022), Applications of Deep Learning, Part 2.1: Introduction to Pandas, and this random site.

# Basic ML workflow
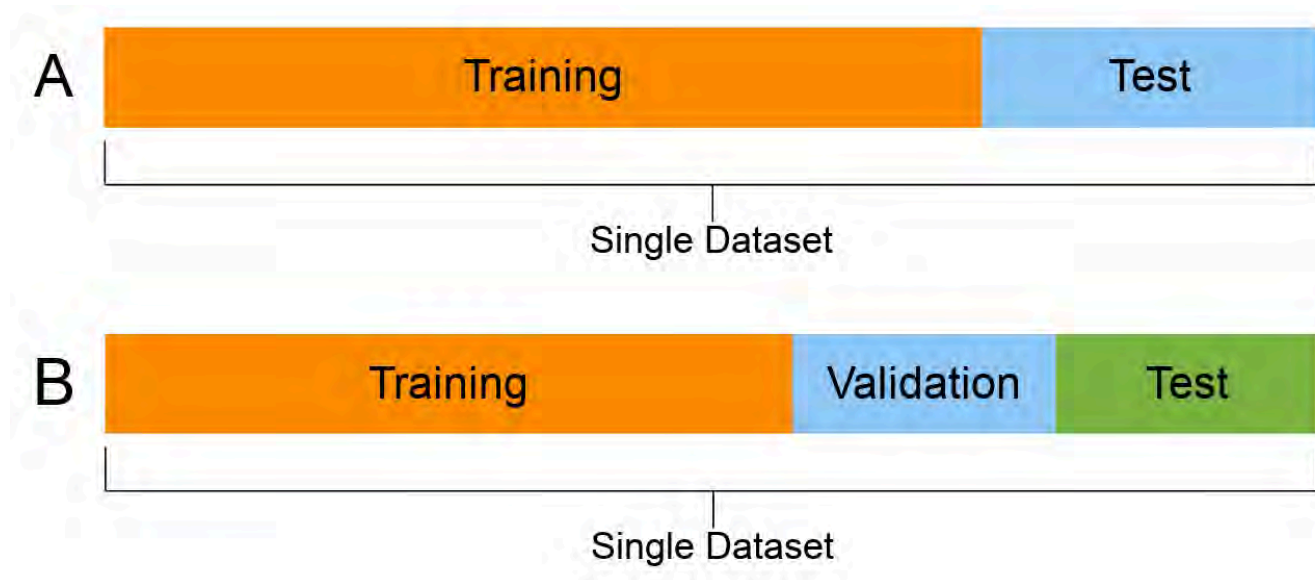


Splitting the data.

1. For each model, fit it to the *training set*.

2. Compute the error for each model on the *validation set*.

3. Select the model with the lowest validation error.

4. Compute the error of the final model on the *test set*.

Source: Wikipedia.

# Split three ways

```python
# Thanks https://datascience.stackexchange.com/a/15136
X_main, X_test, y_main, y_test = train_test_split(
    features, target, test_size=0.2, random_state=1
)

# As 0.25 x 0.8 = 0.2
X_train, X_val, y_train, y_val = train_test_split(
    X_main, y_main, test_size=0.25, random_state=1
)

X_train.shape, X_val.shape, X_test.shape
```

((12384, 8), (4128, 8), (4128, 8))

# Why not use test set for both?

*Thought experiment*: have $m$ classifiers: $f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})$.

They are just as good as each other in the long run

$$\mathbb{P}(\, f_i(\mathbf{X}) = Y \,) \;=\; 90\%, \quad \text{for } i = 1, \ldots, m.$$

Evaluate each model on the test set, some will be better than others.
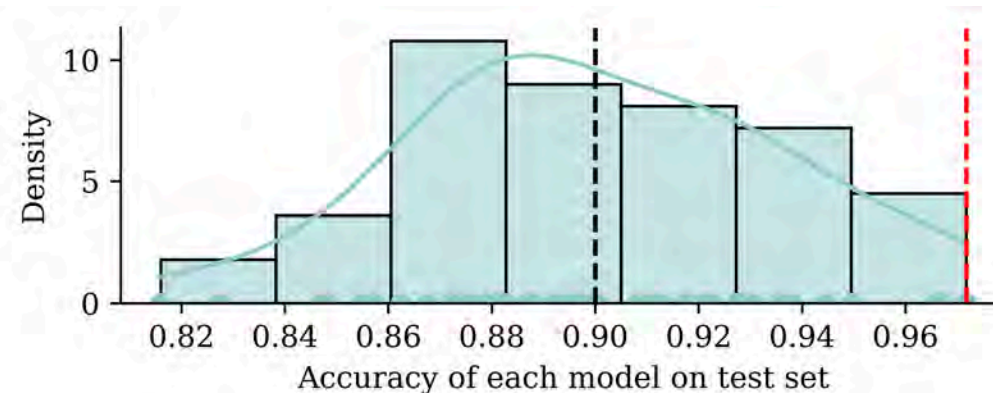


Take the best, you'd think it has $\approx 98\%$ accuracy!

# Lecture Outline

- California House Price Prediction
- **EDA & Baseline Model**
- Our First Neural Network
- Force positive predictions
- Preprocessing
- Early Stopping
- Quiz

# The training set

```
1  X_train
```

|  | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Long |
|---|---|---|---|---|---|---|---|---|
| 9107 | 4.1573 | 19.0 | 6.162630 | 1.048443 | 1677.0 | 2.901384 | 34.63 | −118. |
| 13999 | 0.4999 | 10.0 | 6.740000 | 2.040000 | 108.0 | 2.160000 | 34.69 | −116. |
| 5610 | 2.0458 | 27.0 | 3.619048 | 1.062771 | 1723.0 | 3.729437 | 33.78 | −118. |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 8539 | 4.0727 | 18.0 | 3.957845 | 1.079625 | 2276.0 | 2.665105 | 33.90 | −118. |
| 2155 | 2.3190 | 41.0 | 5.366265 | 1.113253 | 1129.0 | 2.720482 | 36.78 | −119. |
| 13351 | 5.5632 | 9.0 | 7.241087 | 0.996604 | 2280.0 | 3.870968 | 34.02 | −117. |

12384 rows × 8 columns

# Location

Python's `matplotlib` package $\approx$ R's basic `plot`s.

```
1  import matplotlib.pyplot as plt
2
3  plt.scatter(features["Longitude"], features["Latitude"])
```
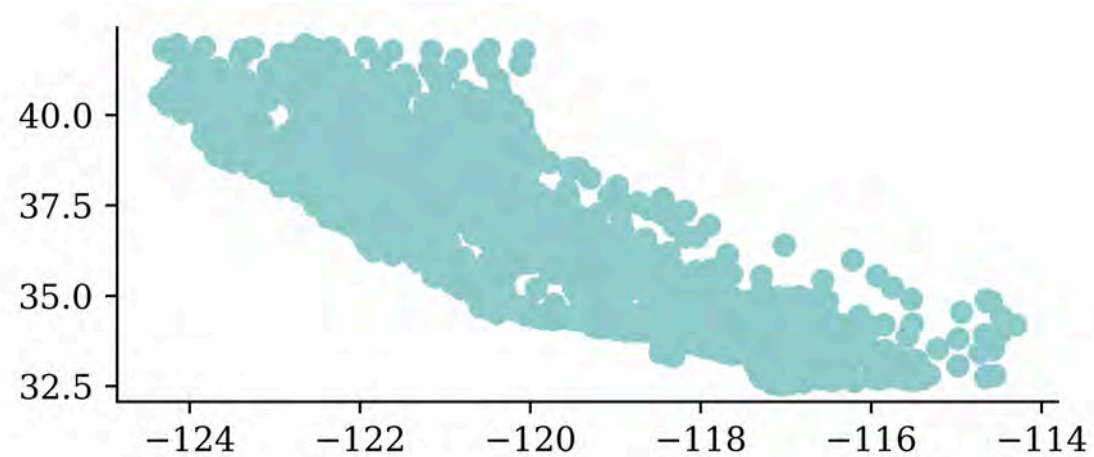
# Location EDA

```python
1  plt.scatter(features["Longitude"], features["Latitude"], c=target, cmap="coolwarm")
2  plt.colorbar()
```
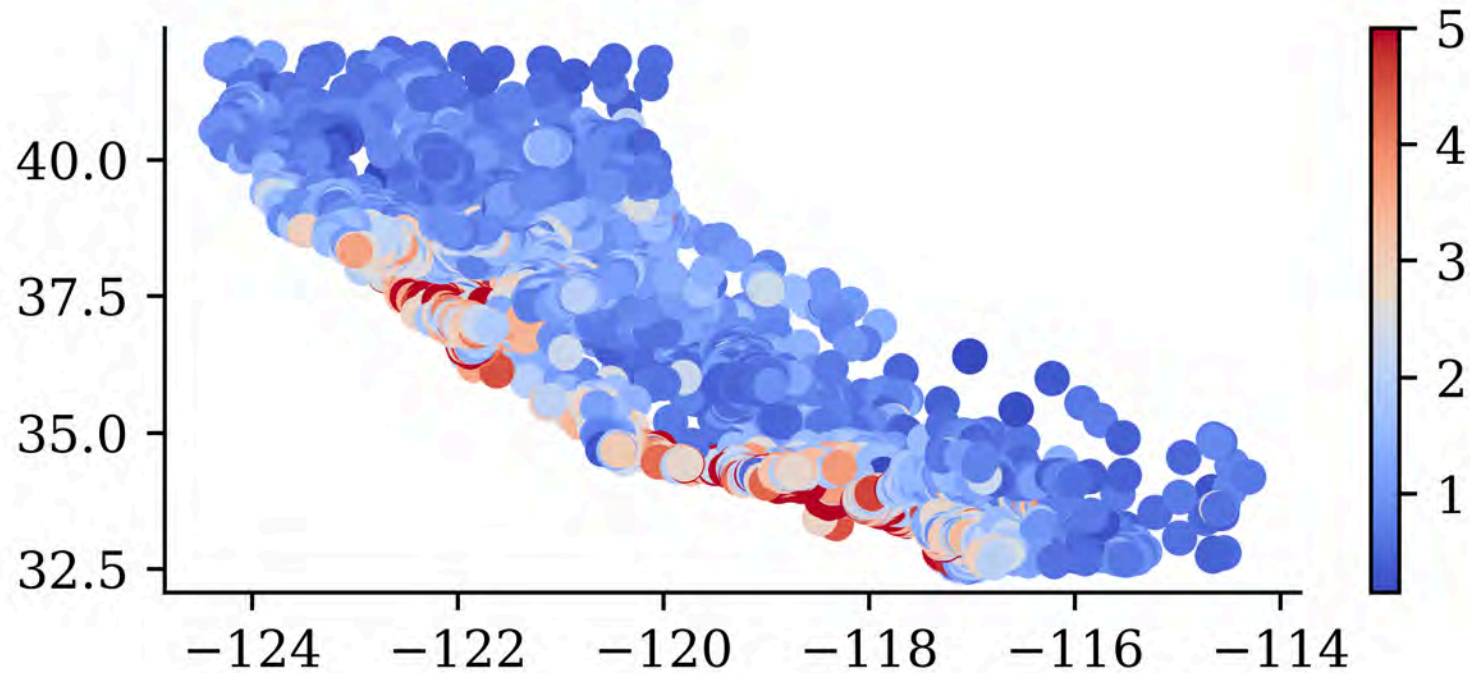
# Pandas can make plots directly

```
1  both = pd.concat([features, target], axis=1)
2  both.plot(kind="scatter", x="Longitude", y="Latitude", c="MedHouseVal", cmap="coolwarm")
```
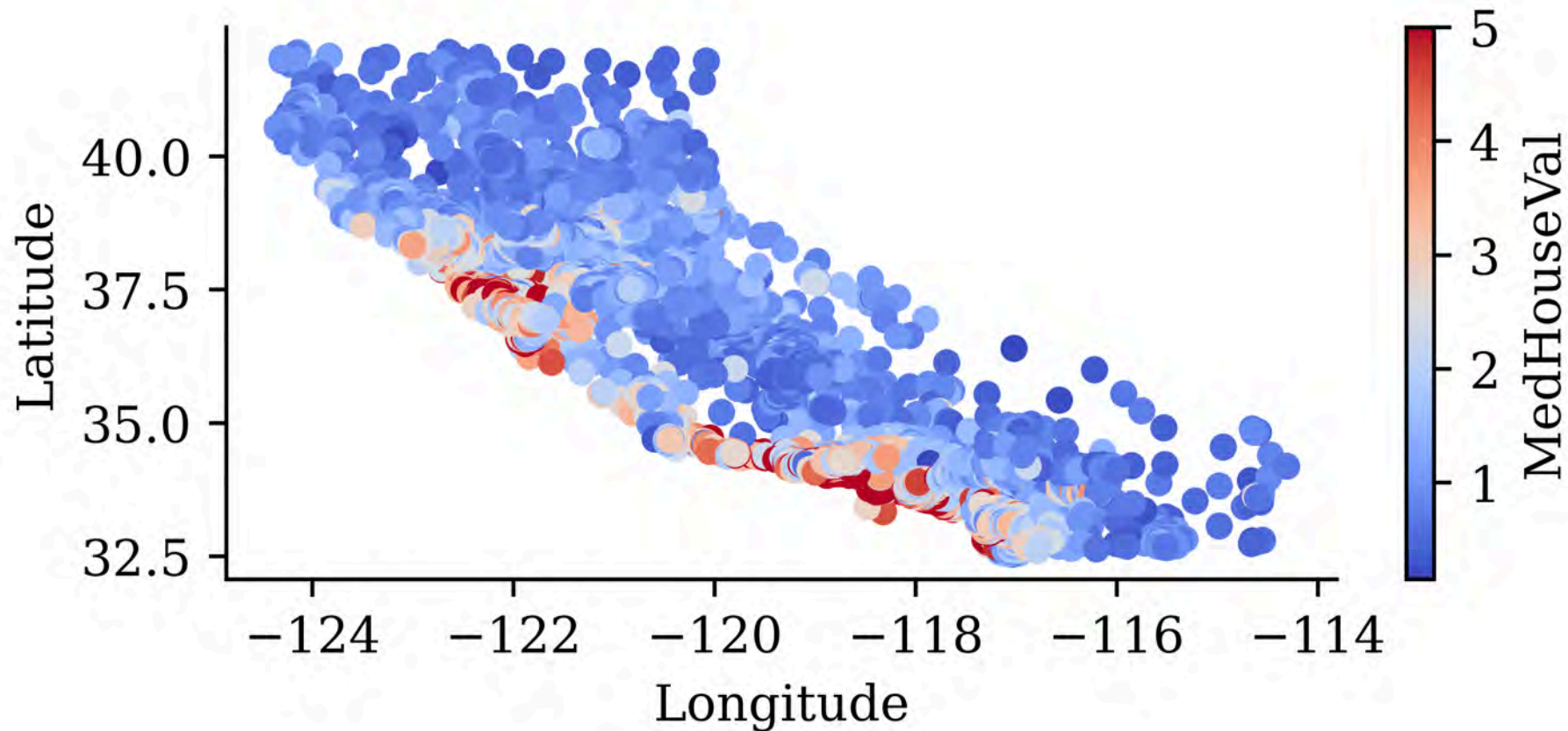
# Features

```
1  print(list(features.columns))
```

```
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude',
'Longitude']
```

## How many?

```
1  num_features = len(features.columns)
2  num_features
```

```
8
```

## Or

```
1  num_features = features.shape[1]
2  features.shape
```

```
(20640, 8)
```

# Linear Regression

$$\hat{y}_i = w_0 + \sum_{j=1}^{p} w_j x_{ij}.$$

```
1  from sklearn.linear_model import LinearRegression
2
3  lr = LinearRegression()
4  lr.fit(X_train, y_train);
```

The $w_0$ is in `lr.intercept_` and the others are in

```
1  print(lr.coef_)
```

```
[ 4.34267965e-01  9.88284781e-03 -9.39592954e-02  5.86373944e-01
 -1.58360948e-06 -3.59968968e-03 -4.26013498e-01 -4.41779336e-01]
```

# Make some predictions

```
1  X_train.head(3)
```

|  | MedInc | HouseAge | AveRooms | AveBedrms | Population |
|---|---|---|---|---|---|
| 9107 | 4.1573 | 19.0 | 6.162630 | 1.048443 | 1677.0 |
| 13999 | 0.4999 | 10.0 | 6.740000 | 2.040000 | 108.0 |
| 5610 | 2.0458 | 27.0 | 3.619048 | 1.062771 | 1723.0 |

```
1  y_pred = lr.predict(X_train.head(3))
2  y_pred
```

```
array([1.81699287, 0.0810446 , 1.62089363])
```

```
1  prediction = lr.intercept_
2  for w_j, x_0j in zip(lr.coef_, X_train.iloc[0]):
3      prediction += w_j * x_0j
4  prediction
```

```
1.8169928680677785
```

# Plot the predictions
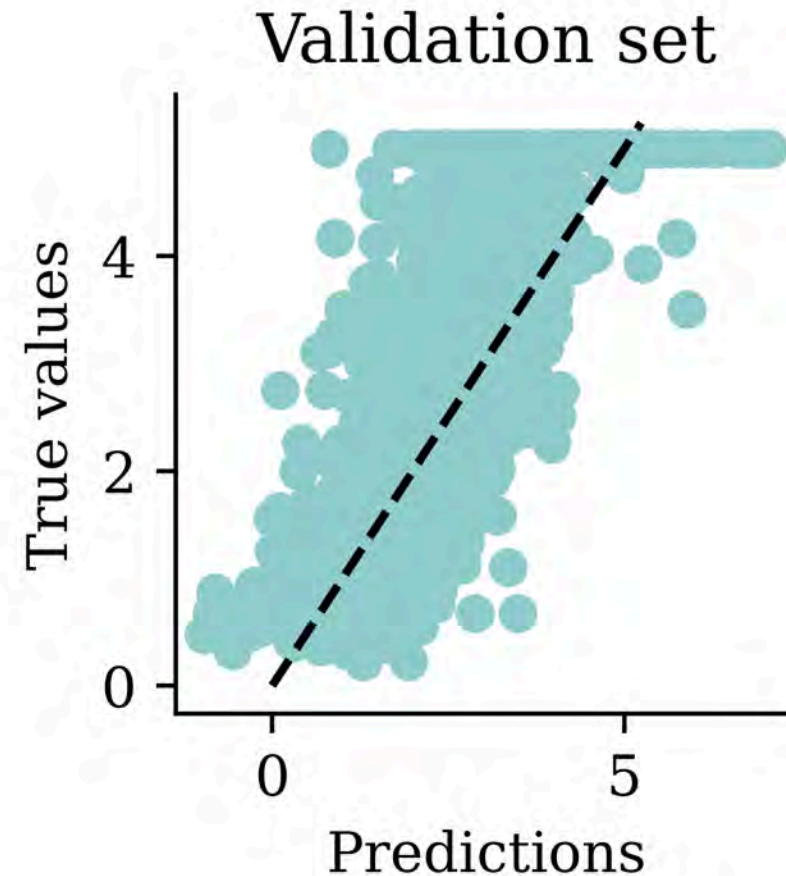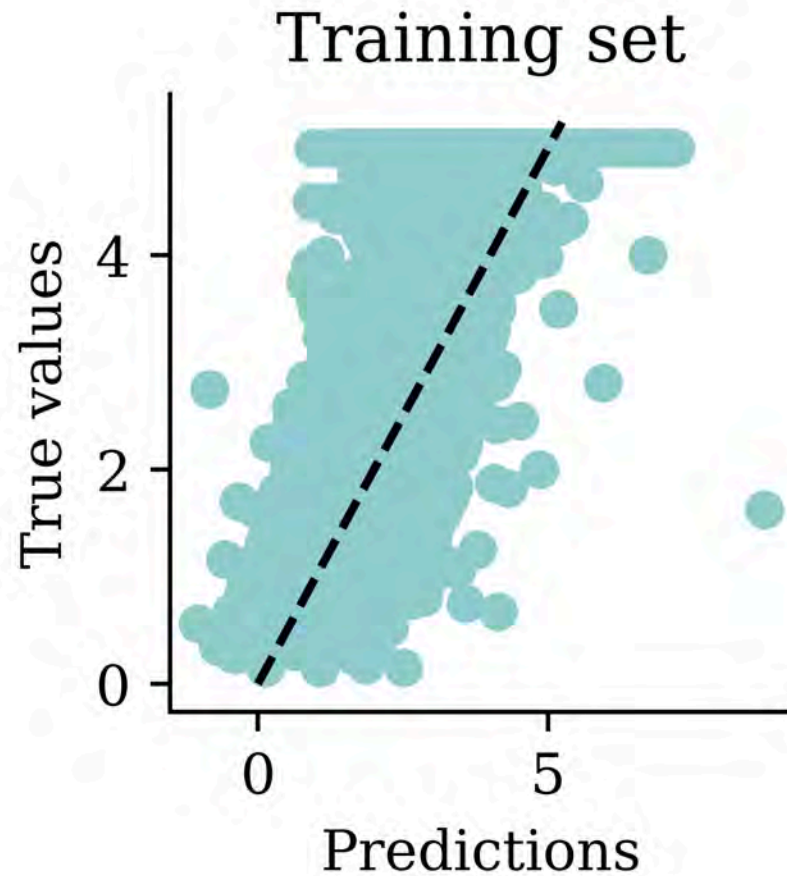
# Calculate mean squared error

```python
import pandas as pd

y_pred = lr.predict(X_train)
df = pd.DataFrame({"Predictions": y_pred, "True values": y_train})
df["Squared Error"] = (df["Predictions"] - df["True values"]) ** 2
df.head(4)
```

|       | Predictions | True values | Squared Error |
|-------|-------------|-------------|---------------|
| 9107  | 1.816993    | 2.281       | 0.215303      |
| 13999 | 0.081045    | 0.550       | 0.219919      |
| 5610  | 1.620894    | 1.745       | 0.015402      |
| 13533 | 1.168949    | 1.199       | 0.000903      |

```python
df["Squared Error"].mean()
```

```
0.5291948207479792
```

# Using `mean_squared_error`

```
1  df["Squared Error"].mean()
```

0.5291948207479792

```
1  from sklearn.metrics import mean_squared_error as mse
2
3  mse(y_train, y_pred)
```

0.5291948207479792

## Store the results in a dictionary:

```
1  mse_lr_train = mse(y_train, lr.predict(X_train))
2  mse_lr_val = mse(y_val, lr.predict(X_val))
3
4  mse_train = {"Linear Regression": mse_lr_train}
5  mse_val = {"Linear Regression": mse_lr_val}
```

💡 **Tip**

Think about the units of the mean squared error. Is there a variation which is more interpretable?

# Lecture Outline

- California House Price Prediction

- EDA & Baseline Model

- **Our First Neural Network**

- Force positive predictions

- Preprocessing

- Early Stopping

- Quiz

# What are Keras and TensorFlow?

Keras is common way of specifying, training, and using neural networks. It gives a simple interface to *various backend* libraries, including Tensorflow.
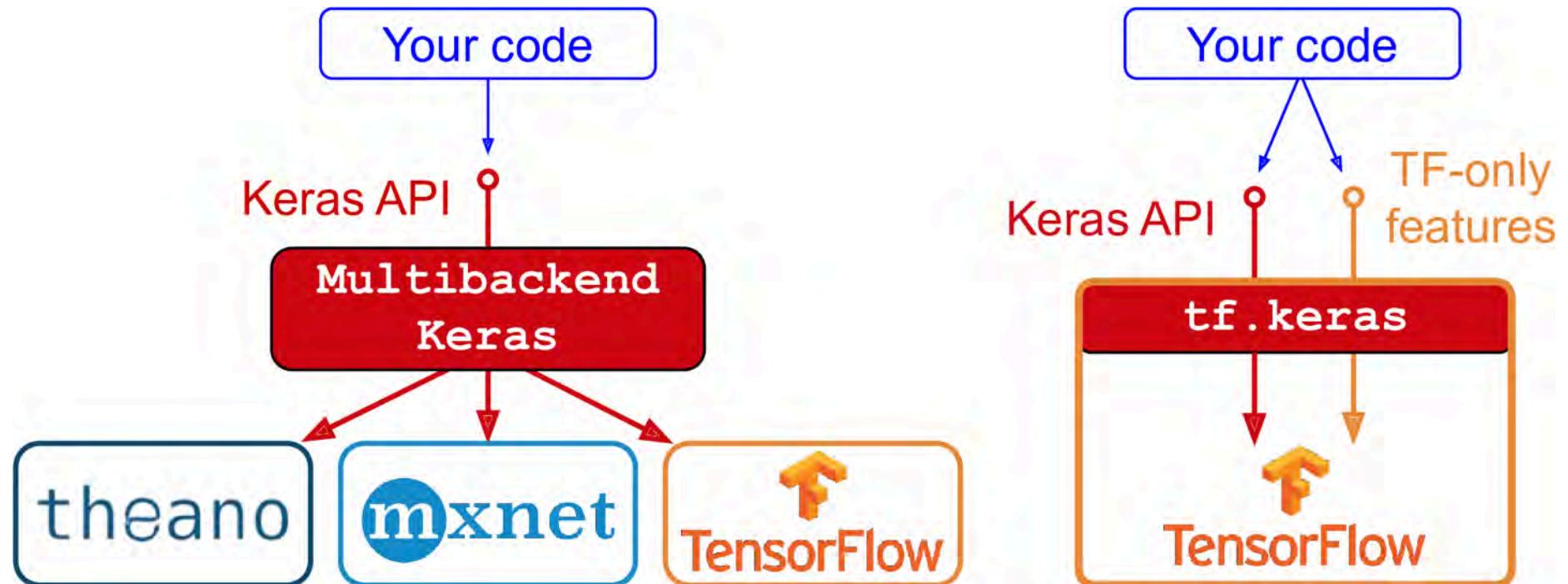


Keras as a independent interface, and Keras as part of Tensorflow.

# Create a Keras ANN model

Decide on the architecture: a simple fully-connected network with one hidden layer with 30 neurons.

Create the model:

```python
from keras.models import Sequential
from keras.layers import Dense, Input

model = Sequential(
    [Input((num_features,)),
     Dense(30, activation="leaky_relu"),
     Dense(1, activation="leaky_relu")]
)
```

# Inspect the model

```
1  model.summary()
```

**Model: "sequential"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense (Dense) | (None, 30) | 270 |
| dense_1 (Dense) | (None, 1) | 31 |

**Total params:** 301 (1.18 KB)
**Trainable params:** 301 (1.18 KB)
**Non-trainable params:** 0 (0.00 B)

# The model is initialised randomly

```
1  model = Sequential([Dense(30, activation="leaky_relu"), Dense(1, activation="leaky_relu"
2  model.predict(X_val.head(3), verbose=0)
```

```
array([[-91.88699  ],
       [-57.336792 ],
       [ -1.2164348]], dtype=float32)
```

```
1  model = Sequential([Dense(30, activation="leaky_relu"), Dense(1, activation="leaky_relu"
2  model.predict(X_val.head(3), verbose=0)
```

```
array([[-63.595753],
       [-34.14082 ],
       [ 17.690414]], dtype=float32)
```

# Controlling the randomness

```python
1  import random
2
3  random.seed(123)
4
5  model = Sequential([Dense(30, activation="leaky_relu"), Dense(1, activation="leaky_relu"
6
7  display(model.predict(X_val.head(3), verbose=0))
8
9  random.seed(123)
10 model = Sequential([Dense(30, activation="leaky_relu"), Dense(1, activation="leaky_relu"
11
12 display(model.predict(X_val.head(3), verbose=0))
```

```
array([[ 1.3595750e+03],
       [ 8.2818079e+02],
       [-1.2993939e+00]], dtype=float32)
array([[ 1.3595750e+03],
       [ 8.2818079e+02],
       [-1.2993939e+00]], dtype=float32)
```

# Fit the model

```
 1  random.seed(123)
 2
 3  model = Sequential([
 4      Dense(30, activation="leaky_relu"),
 5      Dense(1, activation="leaky_relu")
 6  ])
 7
 8  model.compile("adam", "mse")
 9  %time hist = model.fit(X_train, y_train, epochs=5, verbose=False)
10  hist.history["loss"]
```

```
CPU times: user 2.26 s, sys: 189 ms, total: 2.45 s
Wall time: 2.14 s
```

```
[18765.189453125,
 178.23837280273438,
 103.30640411376953,
 48.04053497314453,
 18.11093303833008]
```

# Make predictions

```
1  y_pred = model.predict(X_train[:3], verbose=0)
2  y_pred
```

```
array([[ 0.5477159 ],
       [-1.525452  ],
       [-0.25848356]], dtype=float32)
```

> ⓘ **Note**
>
> The `.predict` gives us a 'matrix' not a 'vector'. Calling `.flatten()` will convert it to a 'vector'.
>
> ```
> 1  print(f"Original shape: {y_pred.shape}")
> 2  y_pred = y_pred.flatten()
> 3  print(f"Flattened shape: {y_pred.shape}")
> 4  y_pred
> ```
>
> ```
> Original shape: (3, 1)
> Flattened shape: (3,)
>
> array([ 0.5477159 , -1.525452  , -0.25848356], dtype=float32)
> ```

# Plot the predictions
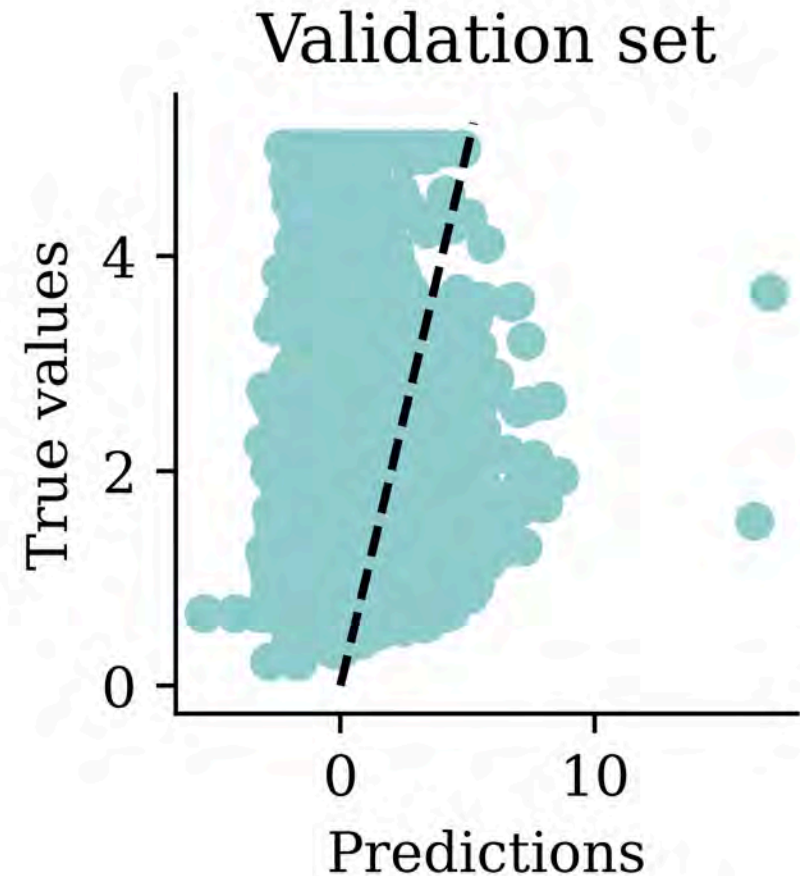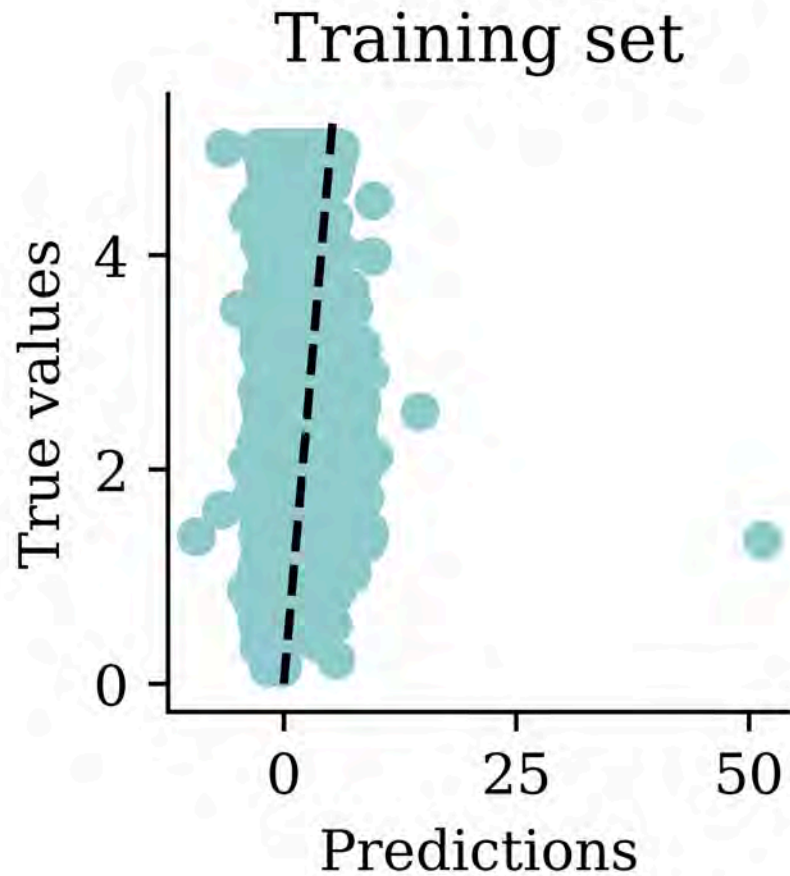
# Assess the model

```
1  y_pred = model.predict(X_val, verbose=0)
2  mse(y_val, y_pred)
```

8.391657291598232

```
1  mse_train["Basic ANN"] = mse(
2      y_train, model.predict(X_train, verbose=0)
3  )
4  mse_val["Basic ANN"] = mse(y_val, model.predict(X_val, verbose=0))
```

## Some predictions are negative:

```
1  y_pred = model.predict(X_val, verbose=0)
2  y_pred.min(), y_pred.max()
```

(-5.371005, 16.863848)

```
1  y_val.min(), y_val.max()
```

(0.225, 5.00001)

# Lecture Outline

- California House Price Prediction

- EDA & Baseline Model

- Our First Neural Network

- **Force positive predictions**

- Preprocessing

- Early Stopping

- Quiz

# Try running for longer

```
 1  random.seed(123)
 2
 3  model = Sequential([
 4      Dense(30, activation="leaky_relu"),
 5      Dense(1, activation="leaky_relu")
 6  ])
 7
 8  model.compile("adam", "mse")
 9
10  %time hist = model.fit(X_train, y_train, epochs=50, verbose=False)
```

```
CPU times: user 22.5 s, sys: 1.56 s, total: 24.1 s
Wall time: 24.5 s
```

# Loss curve

```
1  plt.plot(range(1, 51), hist.history["loss"])
2  plt.xlabel("Epoch")
3  plt.ylabel("MSE");
```
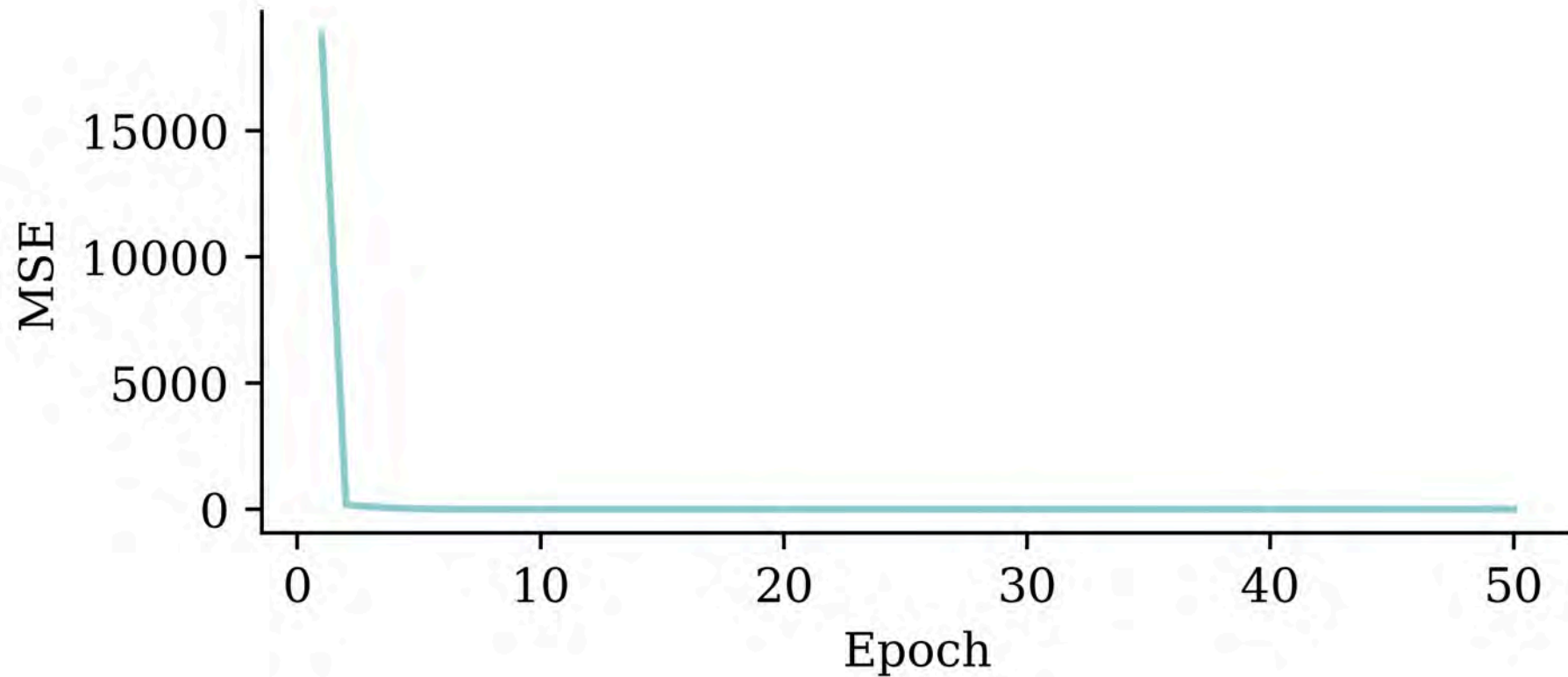
# Loss curve

```
1  plt.plot(range(2, 51), hist.history["loss"][1:])
2  plt.xlabel("Epoch")
3  plt.ylabel("MSE");
```
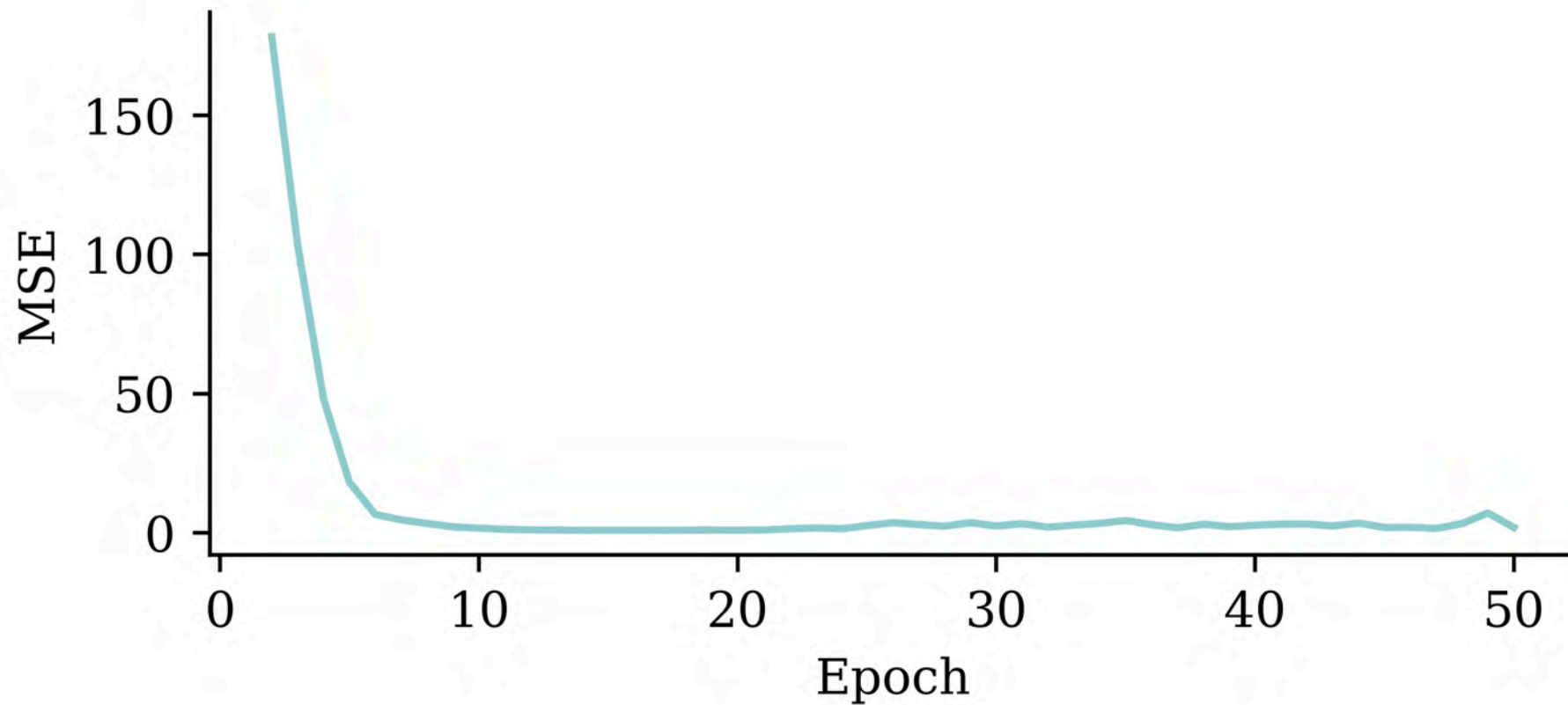
# Predictions

```
1  y_pred = model.predict(X_val, verbose=0)
2  print(f"Min prediction: {y_pred.min():.2f}")
3  print(f"Max prediction: {y_pred.max():.2f}")
```

```
Min prediction: -0.79
Max prediction: 12.92
```

```
1  plt.scatter(y_pred, y_val)
2  plt.xlabel("Predictions")
3  plt.ylabel("True values")
4  add_diagonal_line()
```

```
1  mse_train["Long run ANN"] = mse(
2      y_train, model.predict(X_train, ver
3  )
4  mse_val["Long run ANN"] = mse(y_val, mo
```
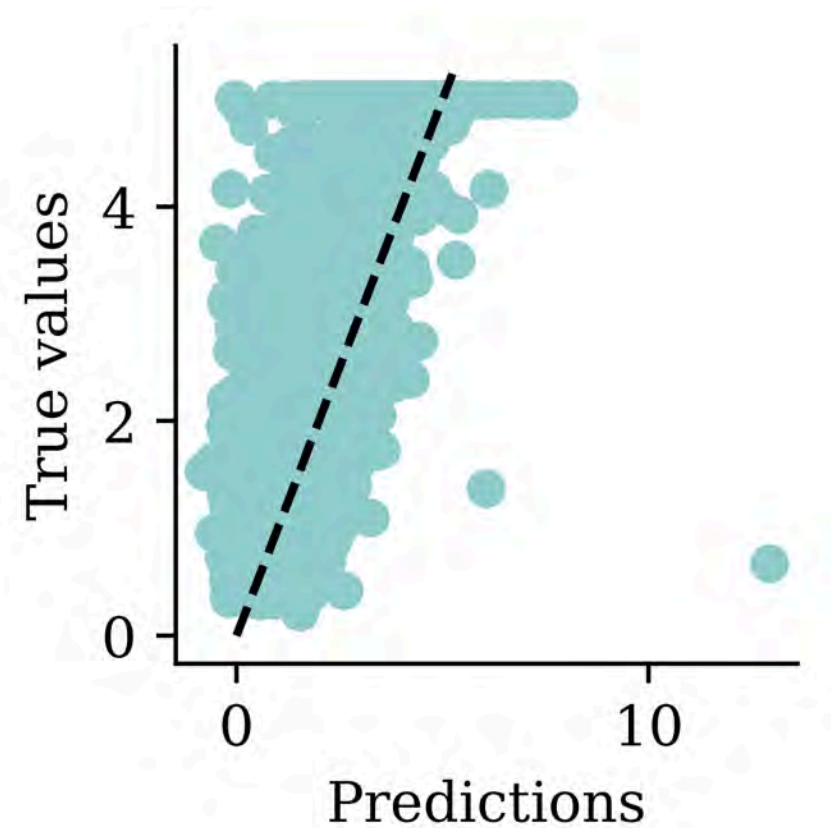
# Try different activation functions
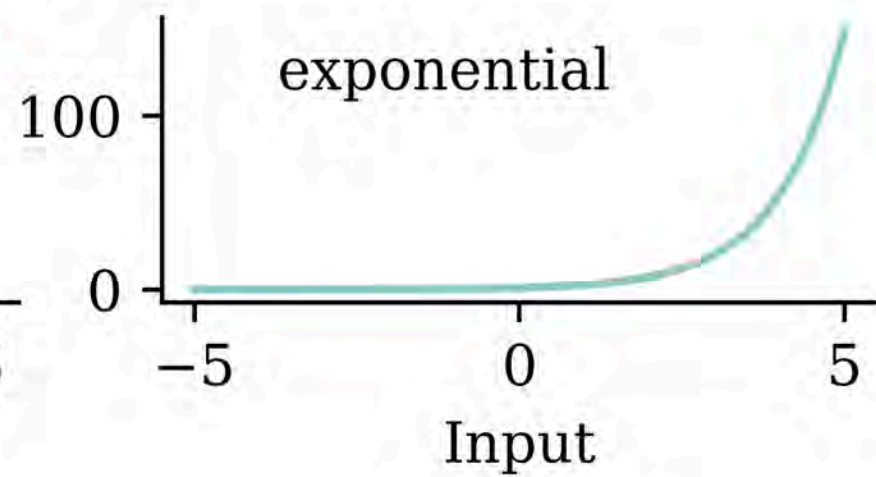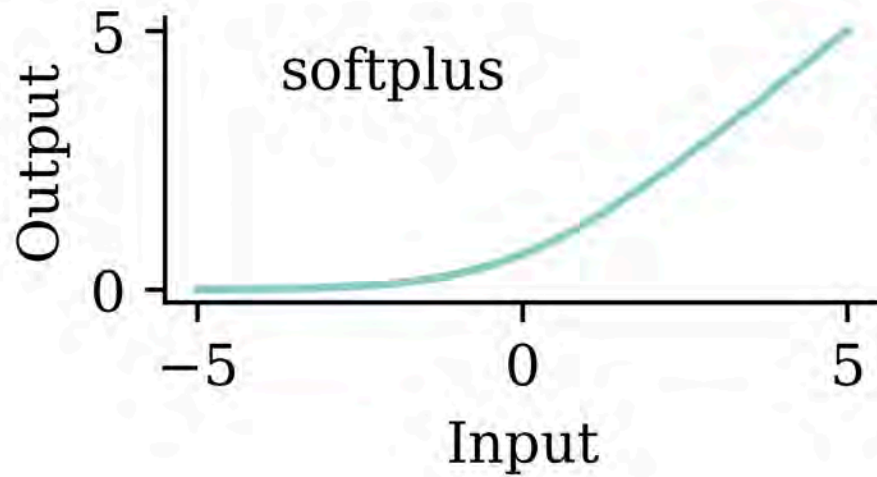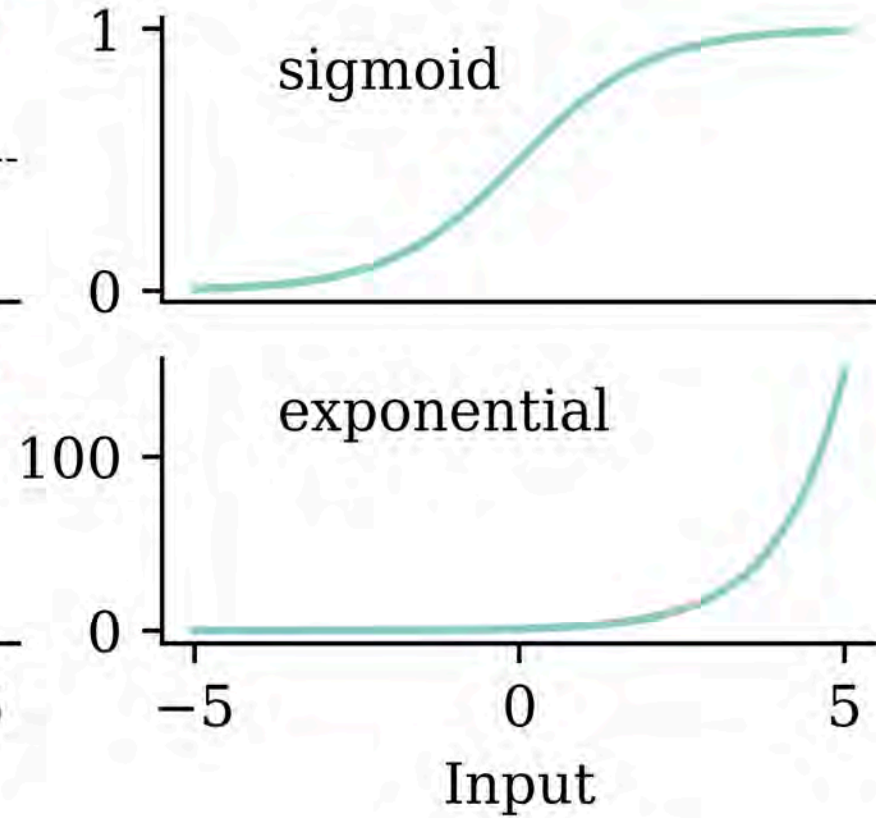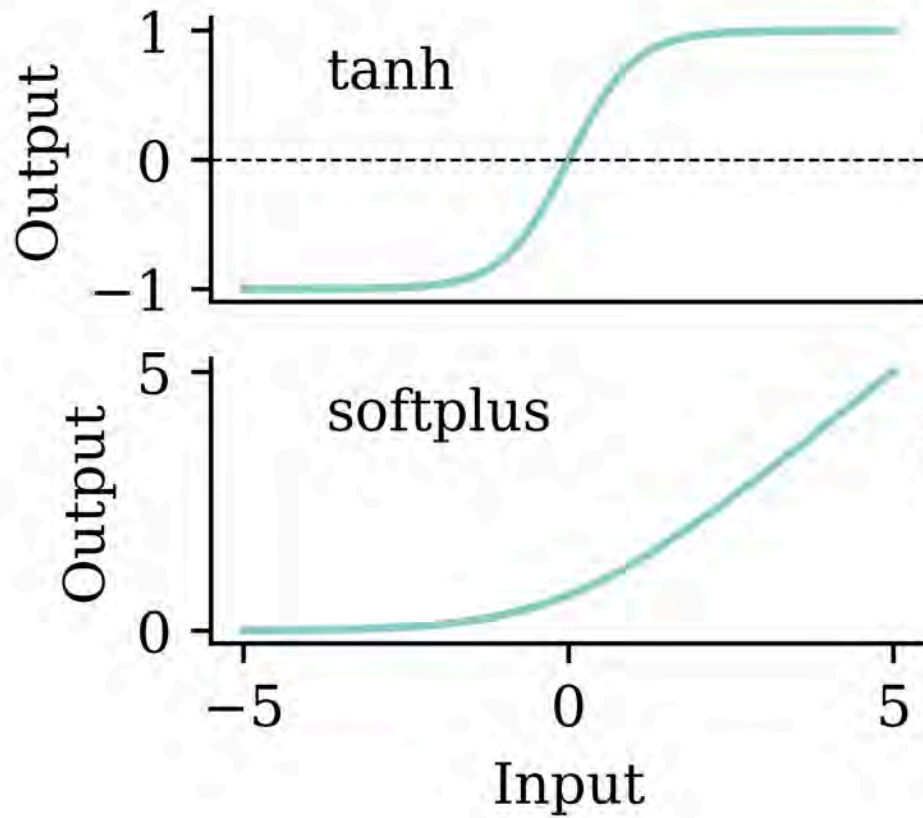
# Enforce positive outputs (softplus)

```python
 1  random.seed(123)
 2
 3  model = Sequential([
 4      Dense(30, activation="leaky_relu"),
 5      Dense(1, activation="softplus")
 6  ])
 7
 8  model.compile("adam", "mse")
 9
10  %time hist = model.fit(X_train, y_train, epochs=50, \
11      verbose=False)
12
13  import numpy as np
14  losses = np.round(hist.history["loss"], 2)
15  print(losses[:5], "...", losses[-5:])
```

```
CPU times: user 14.3 s, sys: 909 ms, total: 15.2 s
Wall time: 10.3 s
[1.856457e+04 5.640000e+00 5.640000e+00 5.640000e+00 5.640000e+00] ... [5.64 5.64 5.64 5.64
5.64]
```

# Plot the predictions
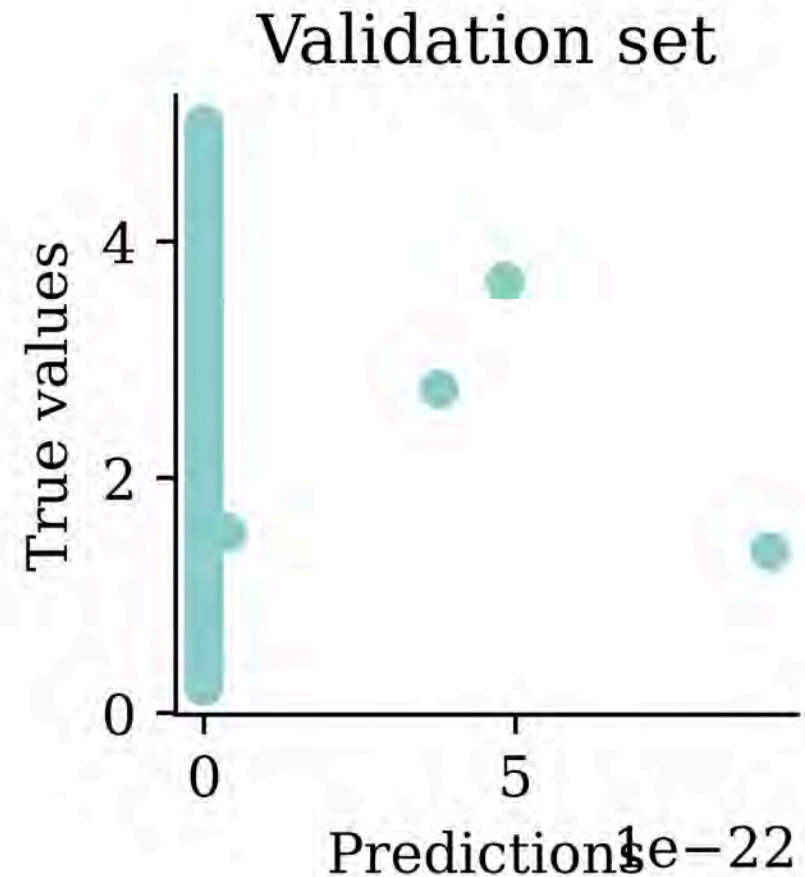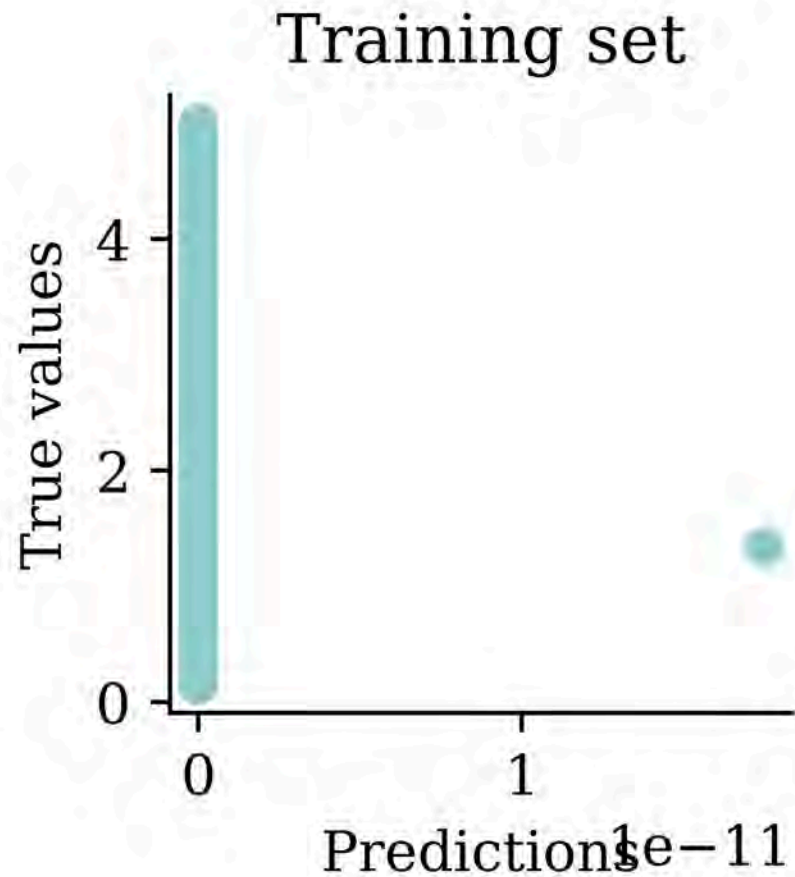
# Enforce positive outputs ($e^x$)

```python
1   random.seed(123)
2
3   model = Sequential([
4       Dense(30, activation="leaky_relu"),
5       Dense(1, activation="exponential")
6   ])
7
8   model.compile("adam", "mse")
9
10  %time hist = model.fit(X_train, y_train, epochs=5, verbose=False)
11
12  losses = hist.history["loss"]
13  print(losses)
```

```
CPU times: user 1.83 s, sys: 122 ms, total: 1.95 s
Wall time: 1.43 s
[nan, nan, nan, nan, nan]
```

# Same as transforming the target

### 4.1. Model

We fitted the following model:

$$\ln(\text{MEDIAN VALUE}) = \text{INTERCEPT} + \beta_2 \text{ MEDIAN INCOME} + \beta_3 \text{ MEDIAN INCOME}^2 + \beta_4 \text{ MEDIAN INCOME}^3$$

$$+ \beta_5 \ln(\text{MEDIAN(AGE)}) + \beta_6 \ln(\text{TOTAL ROOMS/POPULATION})$$

$$+ \beta_7 \ln(\text{BEDROOMS/POPULATION}) + \beta_8 \ln(\text{POPULATION/HOUSEHOLDS})$$

$$+ \beta_9 \ln(\text{HOUSEHOLDS}) \tag{8}$$

The polynomial regression used by researchers who first studied this dataset.

> **ⓘ Note**
>
> Fitting ln(Median Value) is mathematically identical to the `exponential` activation function in the final layer (but metrics are in different units).

Source: Pace and Barry (1997), Sparse Spatial Autoregressions, Statistics & Probability Letters.

# Good to know others results

Table 1
OLS and SAR estimates for median housing prices across 20 640 California census block groups

|  | $B_{ols}$ | $t_{ols}$ | $B_{sar}$ | $t_{sar}$ |
|---|---|---|---|---|
| INTERCEPT | 11.4939 | 275.7518 | 11.6637 | 402.5925 |
| MEDIAN INCOME | 0.4790 | 45.7768 | 0.0349 | 4.7104 |
| MEDIAN INCOME$^2$ | $-0.0166$ | $-9.4841$ | 0.0100 | 8.4280 |
| MEDIAN INCOME$^3$ | $-0.0002$ | $-1.9157$ | $-0.0007$ | $-12.2444$ |
| ln (MEDIAN AGE) | 0.1570 | 33.6123 | $-0.0421$ | $-11.0942$ |
| ln(TOTAL ROOMS/POPULATION) | $-0.8582$ | $-56.1280$ | 0.3098 | 24.5768 |
| ln(BEDROOMS/POPULATION) | 0.8043 | 38.0685 | $-0.1926$ | $-11.8049$ |
| ln (POPULATION/HOUSEHOLDS) | $-0.4077$ | $-20.8762$ | $-0.0342$ | $-2.3582$ |
| ln (HOUSEHOLDS) | 0.0477 | 13.0792 | 0.0034 | 1.5569 |
| $\alpha$ |  |  | 0.8536 |  |
| $R^2$ | 0.6078 |  | 0.8594 |  |
| Median $|e|$ | 0.2101 |  | 0.1084 |  |
| Execution time |  |  | 1130 s |  |
| Number of likelihood evaluations |  |  | 10 |  |

That basic model gets $R^2$ of 0.61, but their fancy model gets 0.86.

Source: Pace and Barry (1997), Sparse Spatial Autoregressions, Statistics & Probability Letters.

# GPT can double-check these results



Asking GPT to check it.

I'd previously given it the CSV of the data.



The code it wrote & ran.

# Lecture Outline

- California House Price Prediction

- EDA & Baseline Model

- Our First Neural Network

- Force positive predictions

- **Preprocessing**

- Early Stopping

- Quiz

# Re-scaling the inputs

```python
1  from sklearn.preprocessing import StandardScaler, MinMaxScaler
2
3  scaler = StandardScaler()
4  scaler.fit(X_train)
5
6  X_train_sc = scaler.transform(X_train)
7  X_val_sc = scaler.transform(X_val)
8  X_test_sc = scaler.transform(X_test)
```

```python
1  plt.hist(X_train.iloc[:, 0])
2  plt.hist(X_train_sc[:, 0])
3  plt.legend(["Original", "Scaled"]);
```
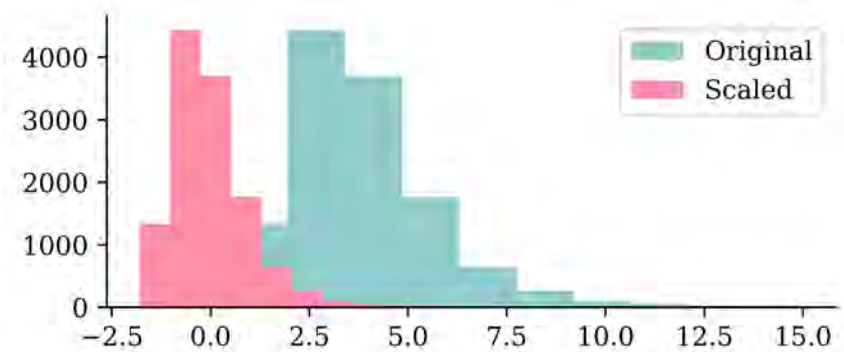
# Same model with scaled inputs

```
 1  random.seed(123)
 2
 3  model = Sequential([
 4      Dense(30, activation="leaky_relu"),
 5      Dense(1, activation="exponential")
 6  ])
 7
 8  model.compile("adam", "mse")
 9
10  %time hist = model.fit( \
11      X_train_sc, \
12      y_train, \
13      epochs=50, \
14      verbose=False)
```
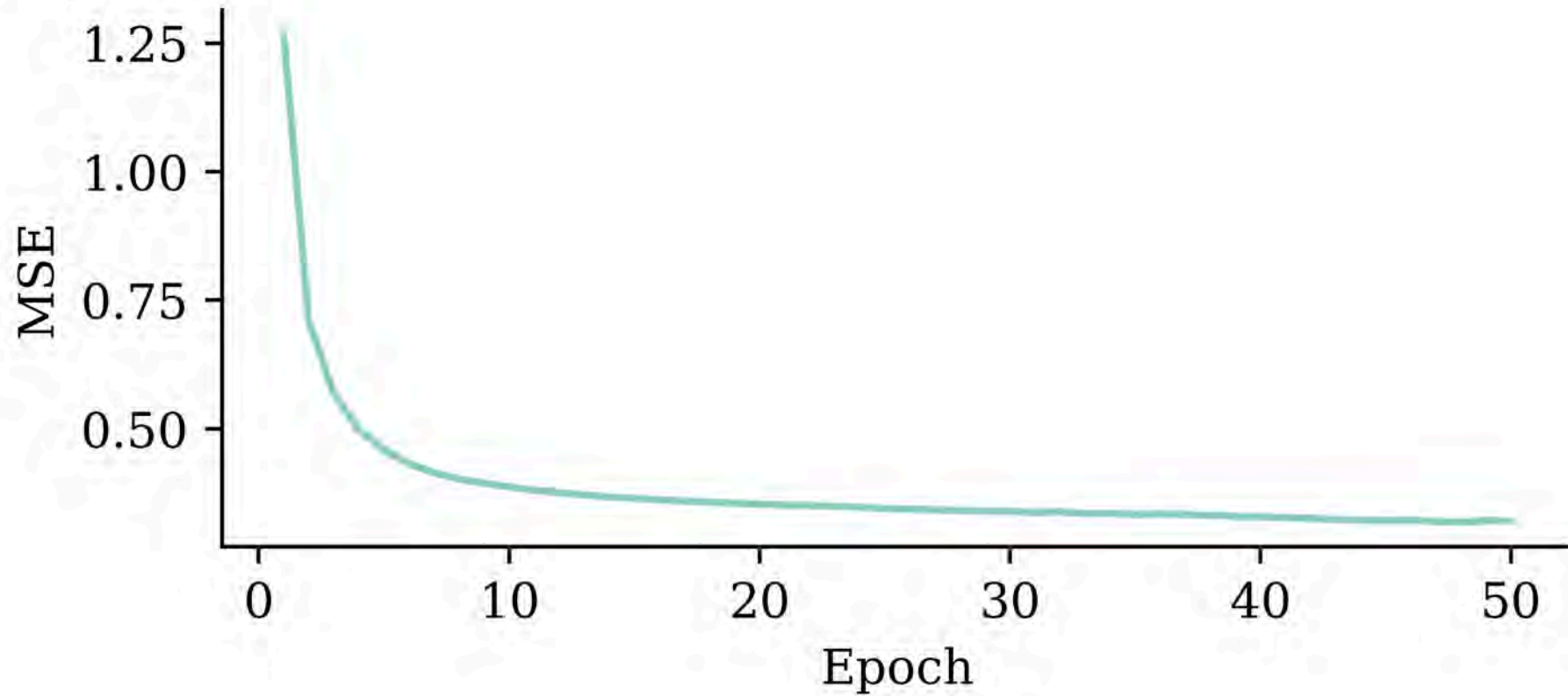
```
CPU times: user 14.4 s, sys: 916 ms, total: 15.3 s
Wall time: 10.3 s
```
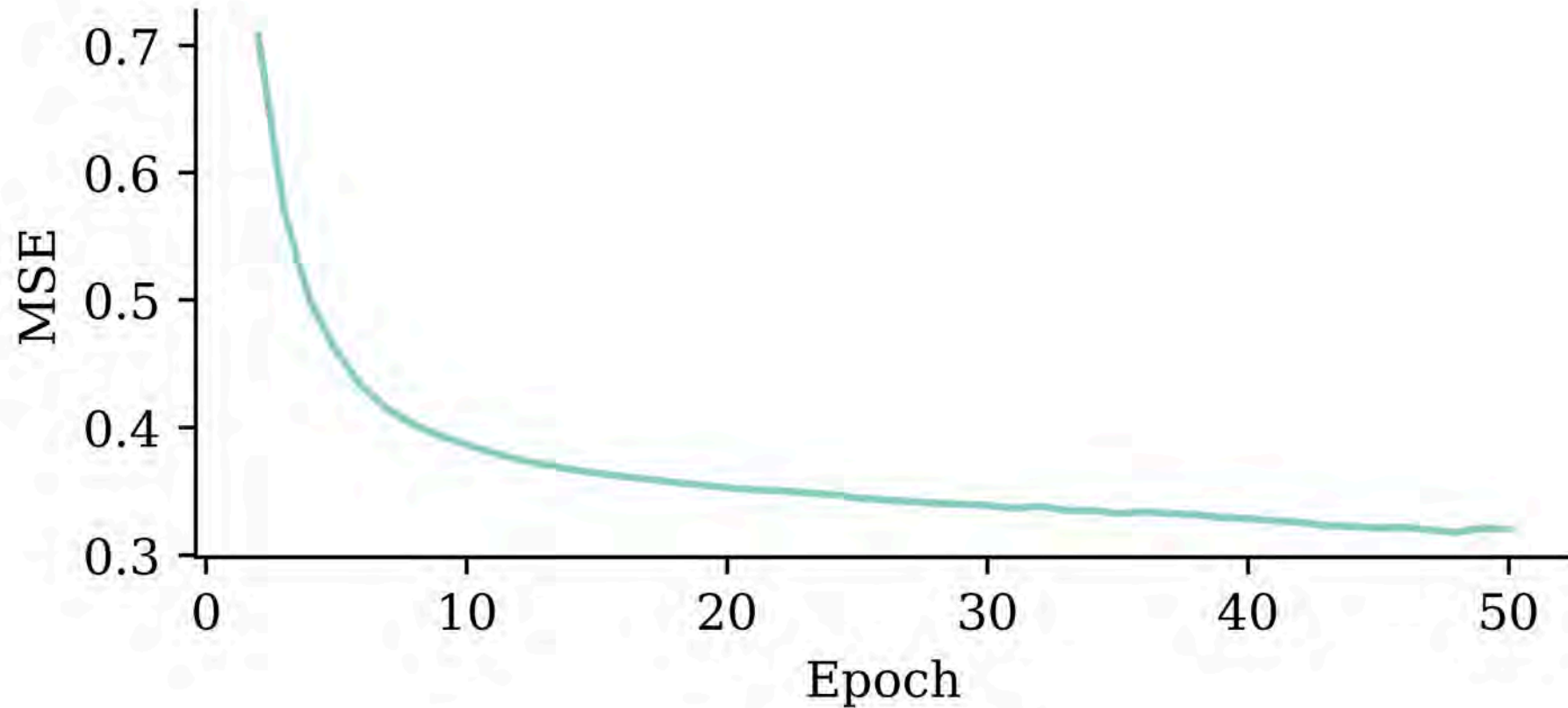
# Loss curve

```
1  plt.plot(range(1, 51), hist.history["loss"])
2  plt.xlabel("Epoch")
3  plt.ylabel("MSE");
```

# Loss curve

```
1  plt.plot(range(2, 51), hist.history["loss"][1:])
2  plt.xlabel("Epoch")
3  plt.ylabel("MSE");
```
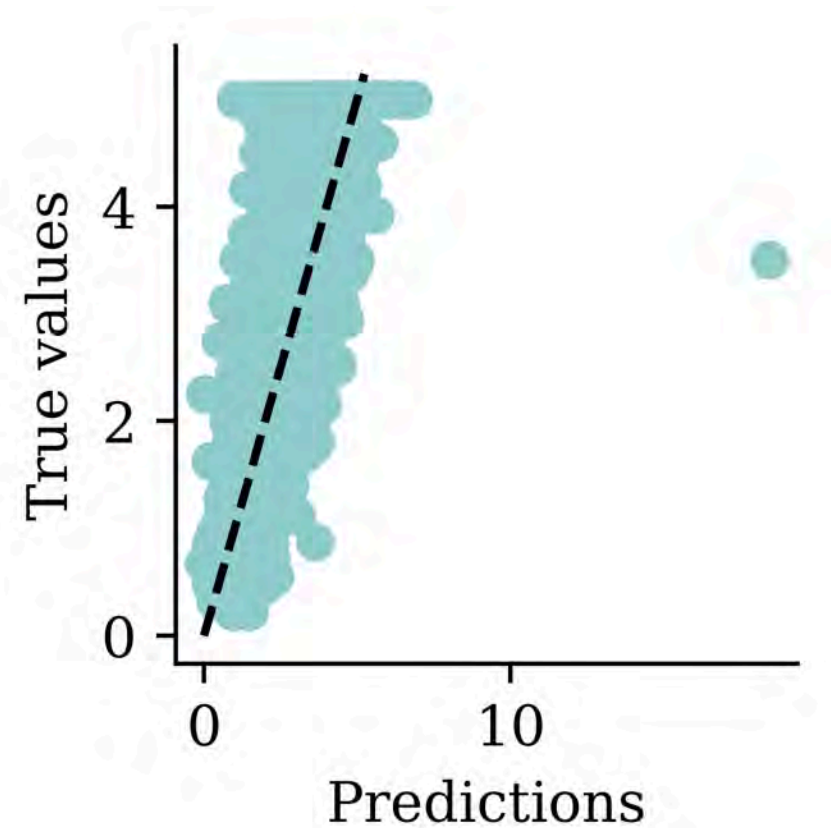
# Predictions

```
1  y_pred = model.predict(X_val_sc, verbose=0)
2  print(f"Min prediction: {y_pred.min():.2f}")
3  print(f"Max prediction: {y_pred.max():.2f}")
```

```
Min prediction: 0.00
Max prediction: 18.45
```

```
1  plt.scatter(y_pred, y_val)
2  plt.xlabel("Predictions")
3  plt.ylabel("True values")
4  add_diagonal_line()
```

```
1  mse_train["Exp ANN"] = mse(
2      y_train, model.predict(X_train_sc,
3  )
4  mse_val["Exp ANN"] = mse(y_val, model.p
```

# Comparing MSE (smaller is better)

On training data:

```
1  mse_train
```

```
{'Linear Regression': 0.5291948207479792,
 'Basic ANN': 8.374382131620425,
 'Long run ANN': 0.9770473035600079,
 'Exp ANN': 0.3182808342909683}
```

On validation data (expect *worse*, i.e. bigger):

```
1  mse_val
```

```
{'Linear Regression': 0.5059420205381367,
 'Basic ANN': 8.391657291598232,
 'Long run ANN': 0.9279673788287134,
 'Exp ANN': 0.36969620817676596}
```

# Comparing models (train)

```
1  train_results = pd.DataFrame(
2      {"Model": mse_train.keys(), "MSE": mse_train.values()}
3  )
4  train_results.sort_values("MSE", ascending=False)
```

|   | Model | MSE |
|---|---|---|
| 1 | Basic ANN | 8.374382 |
| 2 | Long run ANN | 0.977047 |
| 0 | Linear Regression | 0.529195 |
| 3 | Exp ANN | 0.318281 |

# Comparing models (validation)

```
1  val_results = pd.DataFrame(
2      {"Model": mse_val.keys(), "MSE": mse_val.values()}
3  )
4  val_results.sort_values("MSE", ascending=False)
```

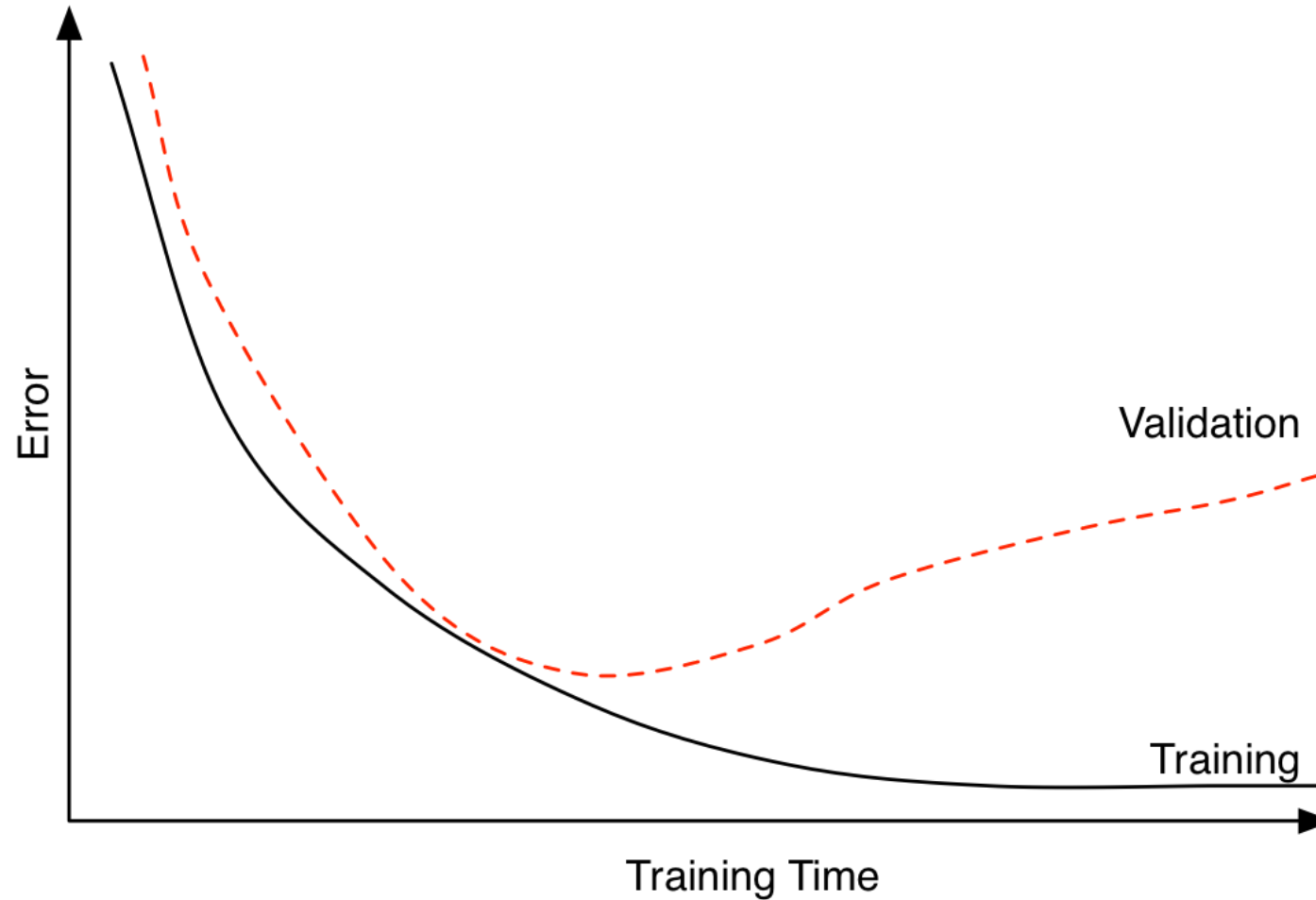|   | Model | MSE |
|---|---|---|
| 1 | Basic ANN | 8.391657 |
| 2 | Long run ANN | 0.927967 |
| 0 | Linear Regression | 0.505942 |
| 3 | Exp ANN | 0.369696 |

# Lecture Outline

- California House Price Prediction

- EDA & Baseline Model

- Our First Neural Network

- Force positive predictions

- Preprocessing

- **Early Stopping**

- Quiz

# Choosing when to stop training



Illustrative loss curves over time.

Source: Heaton (2022), Applications of Deep Learning, Part 3.4: Early Stopping.

# Try early stopping

Hinton calls it a "beautiful free lunch"

```python
from keras.callbacks import EarlyStopping

random.seed(123)
model = Sequential([
    Dense(30, activation="leaky_relu"),
    Dense(1, activation="exponential")
])
model.compile("adam", "mse")

es = EarlyStopping(restore_best_weights=True, patience=15)

%time hist = model.fit(X_train_sc, y_train, epochs=1_000, \
    callbacks=[es], validation_data=(X_val_sc, y_val), verbose=False)
print(f"Keeping model at epoch #{len(hist.history['loss'])-10}.")
```

```
CPU times: user 9.13 s, sys: 588 ms, total: 9.72 s
Wall time: 7.12 s
Keeping model at epoch #14.
```
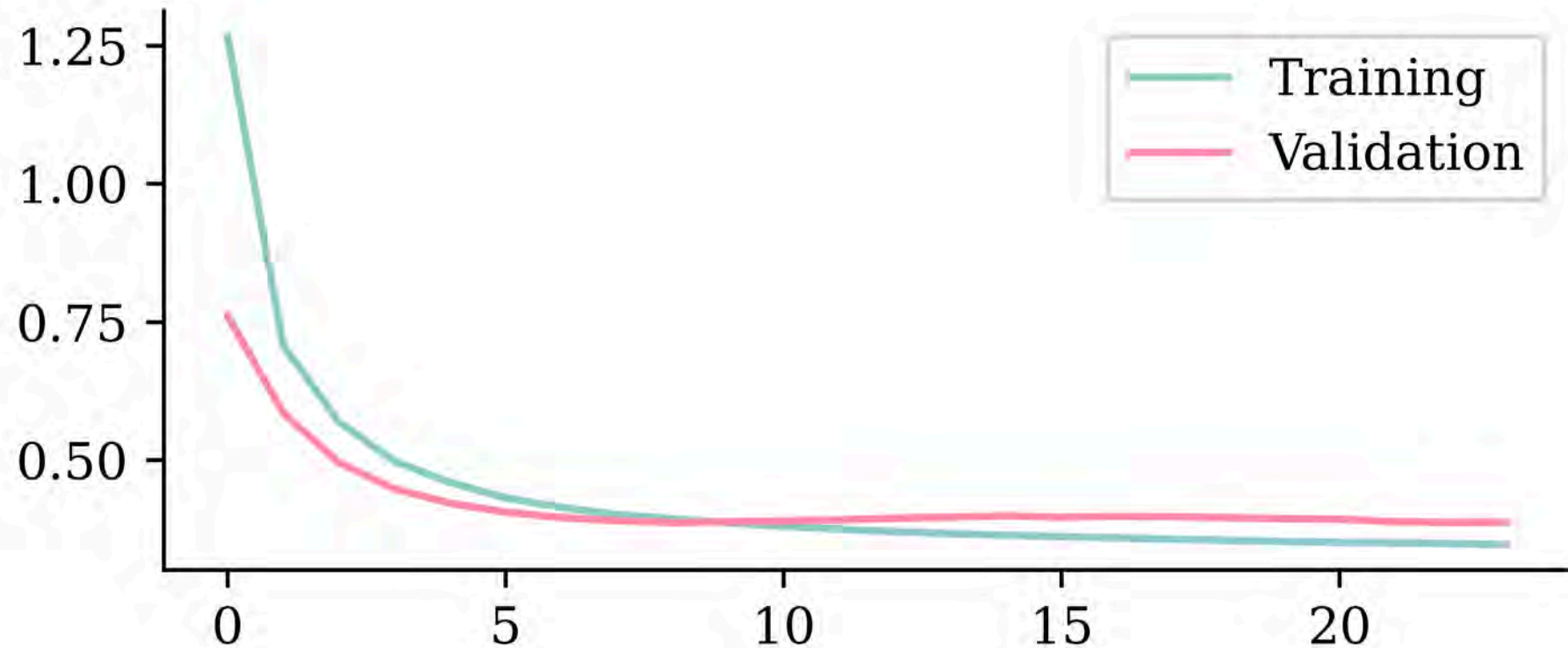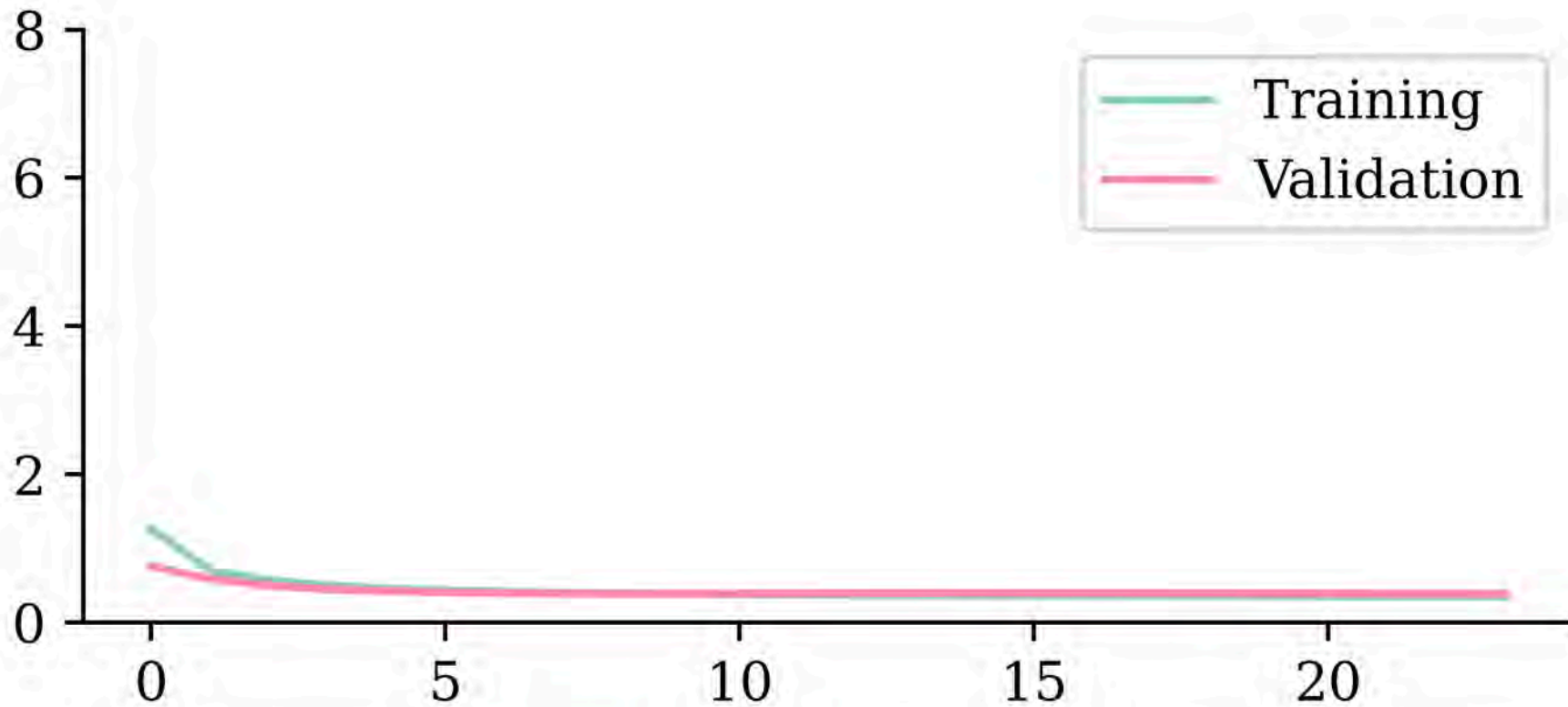
# Loss curve

```
1  plt.plot(hist.history["loss"])
2  plt.plot(hist.history["val_loss"])
3  plt.legend(["Training", "Validation"]);
```
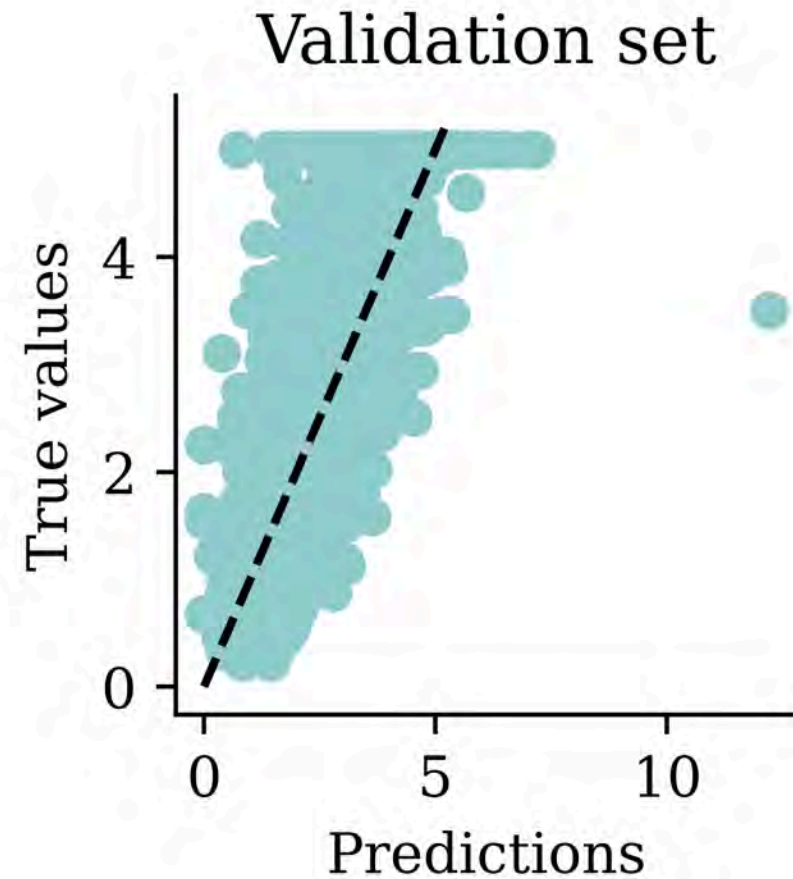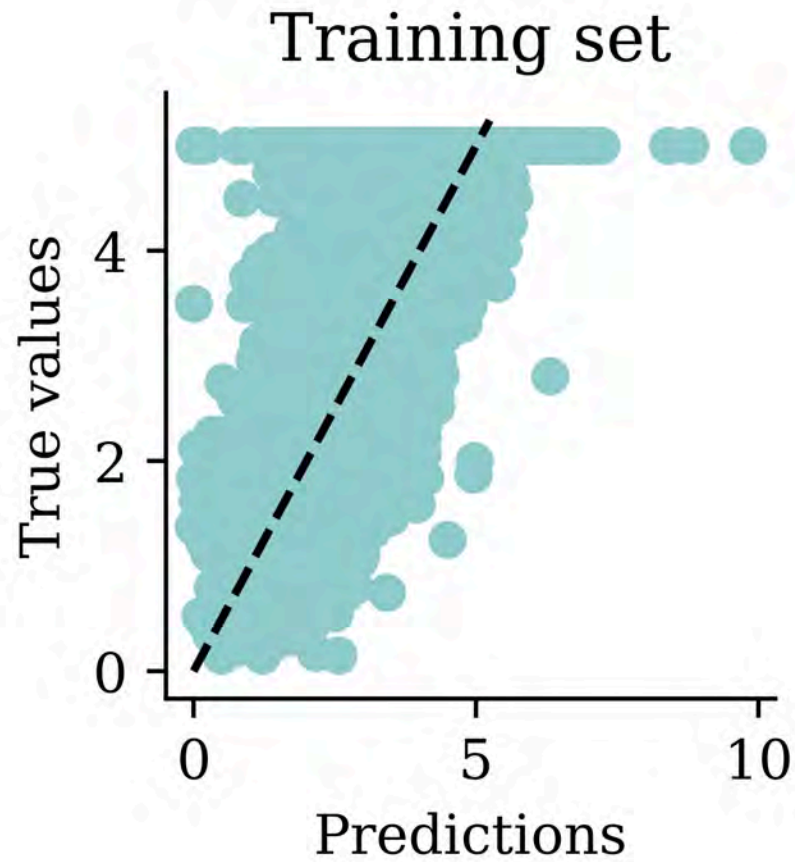
# Loss curve II

```
1  plt.plot(hist.history["loss"])
2  plt.plot(hist.history["val_loss"])
3  plt.ylim([0, 8])
4  plt.legend(["Training", "Validation"]);
```

# Predictions

# Comparing models (validation)

| | Model | MSE |
|---|---|---|
| 1 | Basic ANN | 8.391657 |
| 2 | Long run ANN | 0.927967 |
| 0 | Linear Regression | 0.505942 |
| 4 | Early stop ANN | 0.386975 |
| 3 | Exp ANN | 0.369696 |

# The test set

Evaluate *only the final/selected model* on the test set.

```
1  mse(y_test, model.predict(X_test_sc, verbose=0))
```

0.4026048522207643

```
1  model.evaluate(X_test_sc, y_test, verbose=False)
```

0.4026048183441162

# Another useful callback

```python
from pathlib import Path
from keras.callbacks import ModelCheckpoint

random.seed(123)
model = Sequential(
    [Dense(30, activation="leaky_relu"), Dense(1, activation="exponential")]
)
model.compile("adam", "mse")
mc = ModelCheckpoint(
    "best-model.keras", monitor="val_loss", save_best_only=True
)
es = EarlyStopping(restore_best_weights=True, patience=5)
hist = model.fit(
    X_train_sc,
    y_train,
    epochs=100,
    validation_split=0.1,
    callbacks=[mc, es],
    verbose=False,
)
Path("best-model.keras").stat().st_size
```

19215

# Lecture Outline

- **California House Price Prediction**

- EDA & Baseline Model

- Our First Neural Network

- Force positive predictions

- Preprocessing

- Early Stopping

- Quiz

# Critique this 💩 regression code

```
1  X_train = features[:80]; X_test = features[81:]
2  y_train = targets[:80]; y_test = targets[81:]
```

```
1  model = Sequential([
2      Input((2,)),
3    Dense(32, activation='relu'),
4      Dense(32, activation='relu'),
5    Dense(1, activation='sigmoid')
6  ])
7  model.compile(optimizer="adam", loss='mse')
8  es = EarlyStopping(patience=10)
9  fitted_model = model.fit(X_train, y_train, epochs=5,
10     callbacks=[es], verbose=False)
```

```
1  trainMAE = model.evaluate(X_train, y_train, verbose=False)
2  hist = model.fit(X_test, y_test, epochs=5,
3      callbacks=[es], verbose=False)
4  hist.history["loss"]
5  testMAE = model.evaluate(X_test, y_test, verbose=False)
```
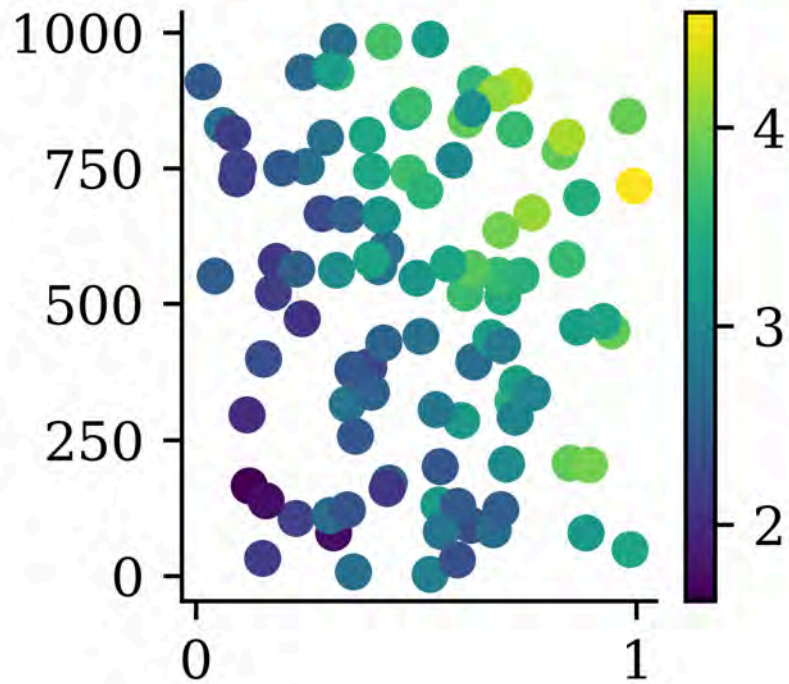
```
1  f"Train MAE: {testMAE:.2f} Test MAE: {trainMAE:.2f}"
```

```
'Train MAE: 4.82 Test MAE: 4.32'
```
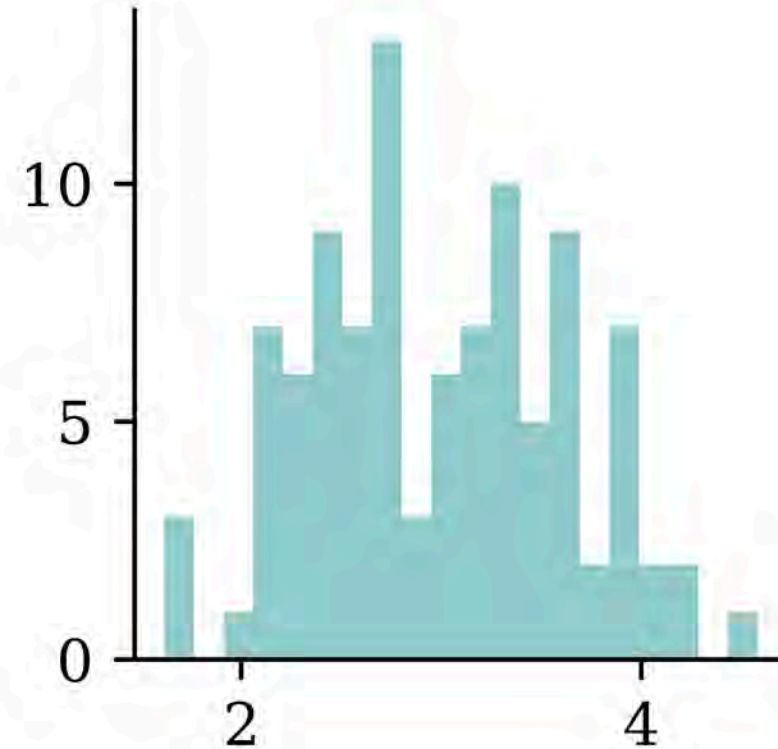
UNSW
SYDNEY

# The data

```
1  plt.scatter(x, y, c=targets)
2  plt.colorbar()
```

```
1  plt.hist(targets, bins=20);
```

# Package Versions

```
1  from watermark import watermark
2  print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch
```

```
Python implementation: CPython
Python version       : 3.11.9
IPython version      : 8.24.0

keras     : 3.3.3
matplotlib: 3.9.0
numpy     : 1.26.4
pandas    : 2.2.2
seaborn   : 0.13.2
scipy     : 1.11.0
torch     : 2.3.1
tensorflow: 2.16.1
tf_keras  : 2.16.0
```

# Glossary

- callbacks

- cost/loss function

- early stopping

- epoch

- Keras, Tensorflow, PyTorch

- matplotlib

- targets

- training/test split

- validation set