INFO5002: Intro to Python for Info Sys

Week 11

Slides created by: Zachary Doucet

Northeastern University

Week 11

I. Machine Learning

II. Linear Regression

III. Multiple Regression

Review

Array as primitive

- Numpy uses the array as a primitive—kind of like a Python list.

```
x = np.array([1,2,3,4,5,6])
```

- Arrays can be indexed and are mutable like python.

```
x[0] = 4
```

- Can also do slicing but be careful—returns a view instead of a copy. Mutating the view mutates the original!

```
y = x[:3]
y[0] = 6
```

Conditional selection

- Can pass a conditional within brackets:

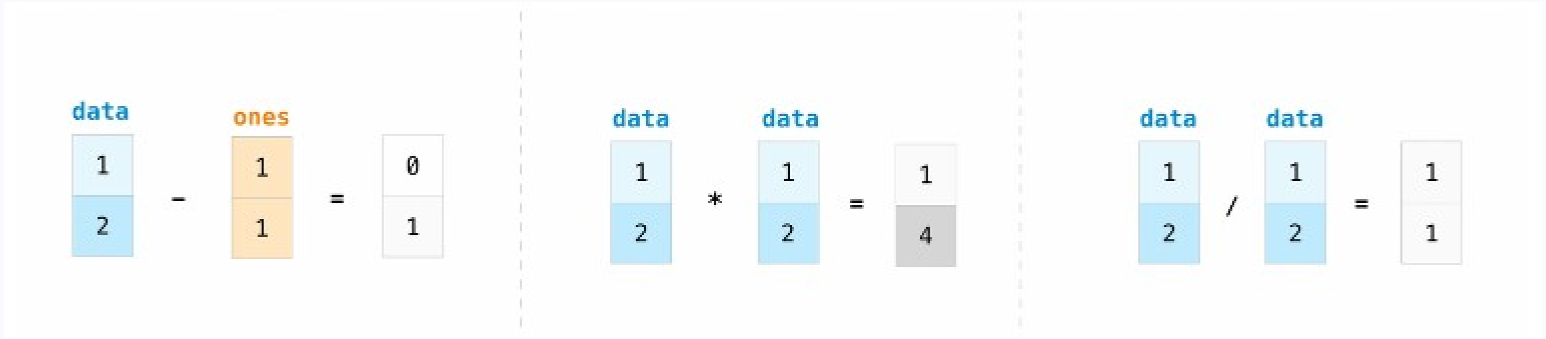
```
b = a[(a > 18) & (a < 25)]
```

- & is same as python’s and and | is same as python’s or.

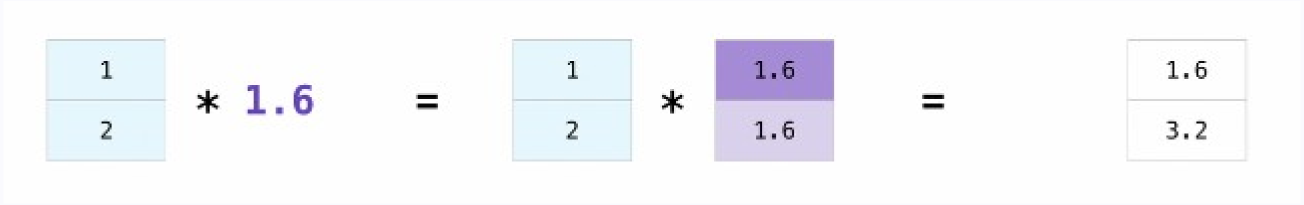
```
divisible_2_or_3 = a[(a%2==0)|(a%3==0)]
```

Operations

- +, -, /, * all do the operation element wise.



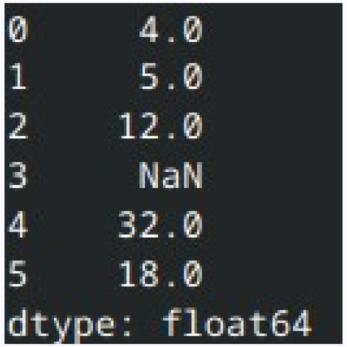
- If you try to perform an operation with two different shapes, it will attempt to broadcast to make it work.



Series and DataFrame as primitive

- Series: labelled one-dimensional array holding data of any type; integers, strings, Python object's.

```
s = pd.Series([4, 5, 12, np.nan, 32, 18])
```



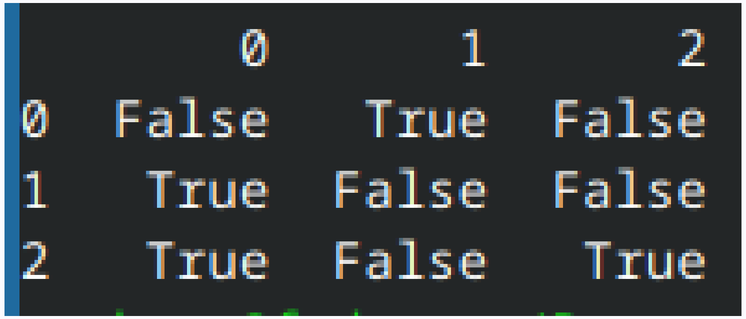
```
0     4.0
1     5.0
2    12.0
3     NaN
4    32.0
5    18.0
dtype: float64
```

- DataFrame: two-dimensional data structure that acts like a table with rows and columns.

```
df = pd.DataFrame(np.array, index=row_names, columns=colum_names)
```

Missing data

- df.dropna(): drop any row with missing data.
- df.fillna(value=None): fill any missing data with *value*.
- df.isna(): returns a new DataFrame where all cells with missing values are set to False; otherwise, True.



```
       0      1      2
0  False   True  False
1   True  False  False
2   True  False   True
```

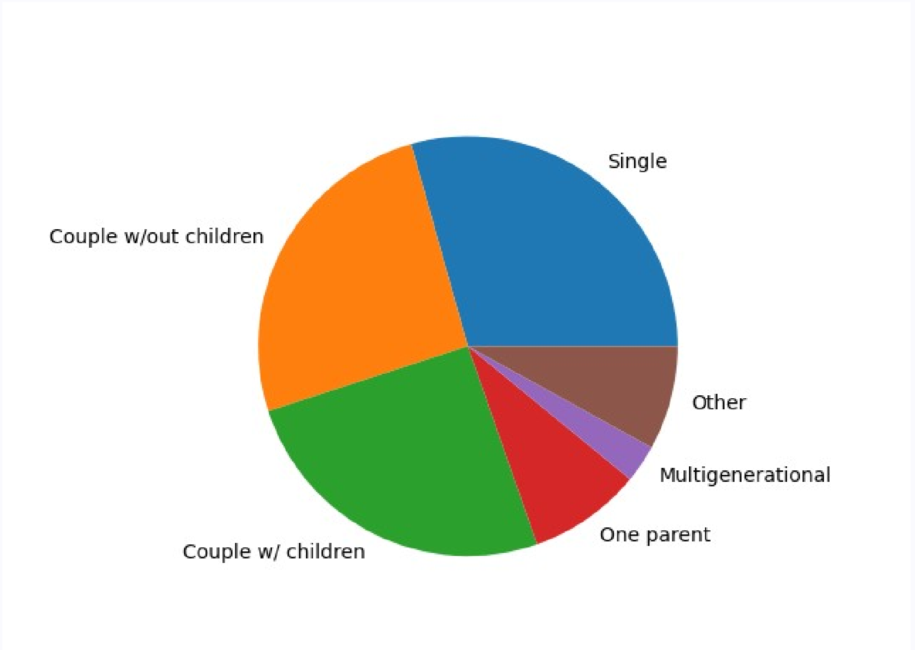
Importing/Exporting

- pd.read_csv(“filepath.csv”): to read a csv and load as DataFrame.
- df.to_csv(“filepath.csv”): save DataFrame to csv.
- pd.read_parquet(“filepath.parquet”)
- df.to_parquet(“filepath.parquet”)
- pd.read_excel(“filepath.xlsx”)
- df.to_excel(“filepath.xlsx”)

Pie Chart

- You give a 1D collection (x) where the proportion of each is computed as: `x / sum(x)`
- Can optional give a string list of labels.

```
plt.pie(x, labels=labels])
```

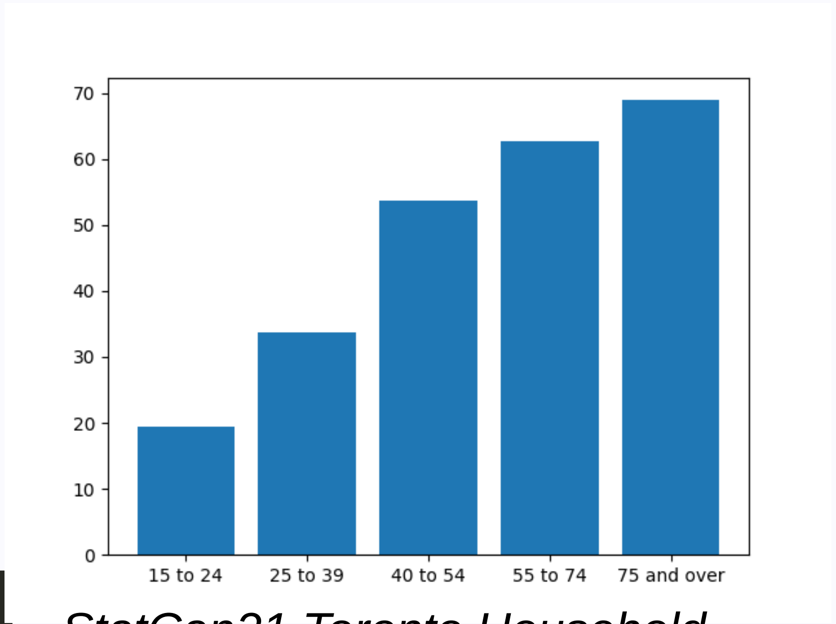


StatCan21 Houshold Disttribution

Bar Charts

- You give the bar labels (x) and the height of each (height)
- Can optionally specify:
  - Width of each bar (width)
  - Bar alignment:
    “center” or “edge”

```
plt.bar(x, height,
   width=0.8, align="center")
```



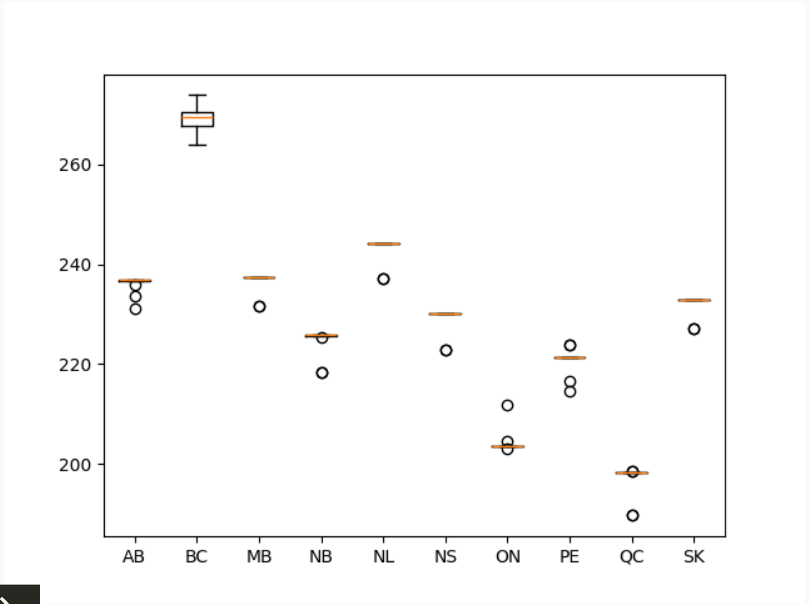
StatCan21 Toronto Household ownership rate by age.

Box Plot

- Give data as a 2D collection where each entry is a column and for each column you give all the raw data.

- Can optionally specify:
  - Labels (tick_labels)

```
plt.boxplot(x, tick_labels=labels)
```

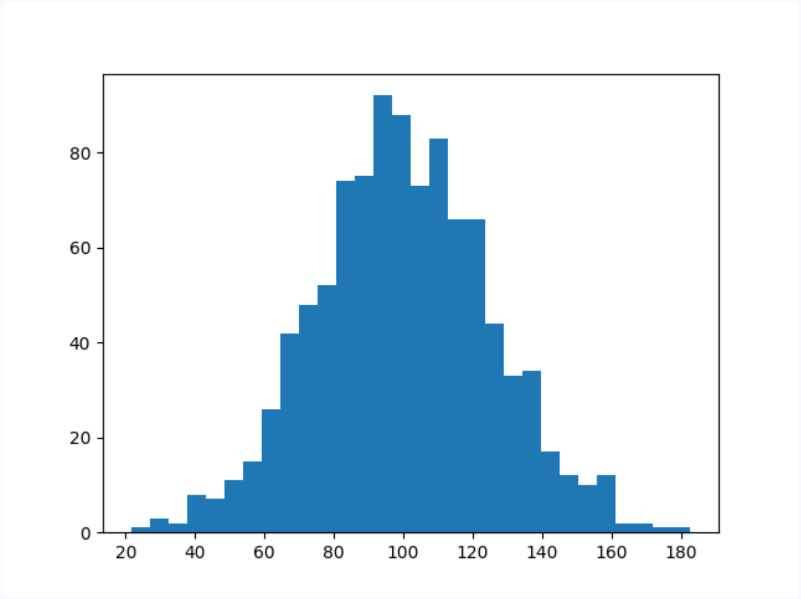


StatCan24 Average monthly egg prices per province for 2024.

Histogram

- Simply pass all of your data (x).
- Can optionally specify:
  - Number subdivisions (bins)

```
plt.hist(x, bins=10)
```

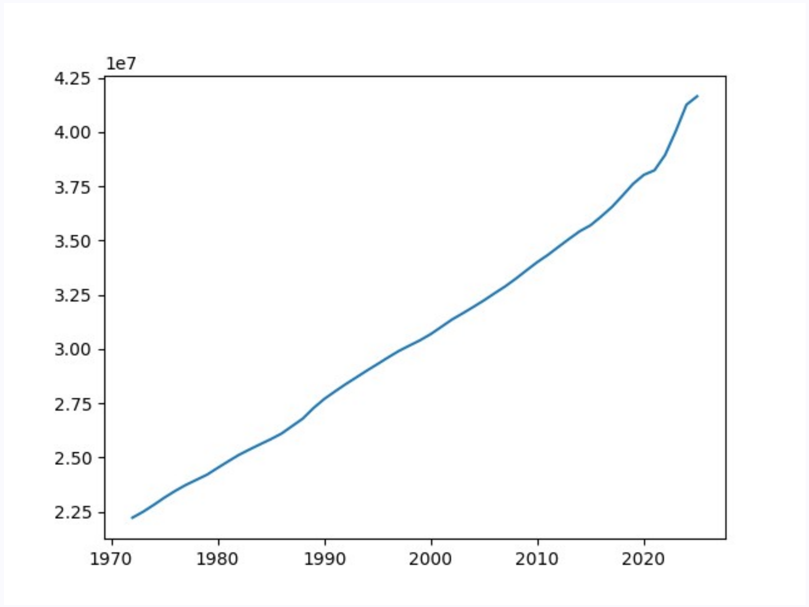


Histogram of a random distribution at mean 100 and std div of 25 with 30 subdivisions.

Line Charts

- Give your numerical x and y data.
- Can optionally pass:
  - Basic formating
  - To scale x or y (scalex, scaley)

```
plt.plot(x, y, [fmt]
   scalex=True, scaley=True)
```

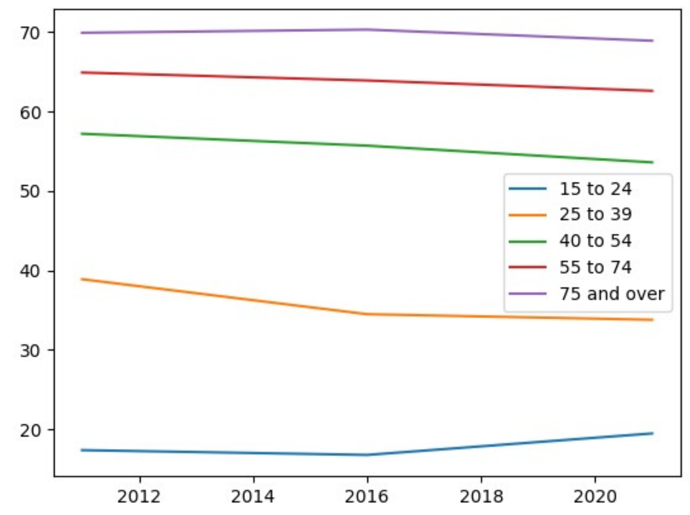


Canada yearly population since StatCan.

Multi-Line Charts

- Can call plot multiple times to place multiple lines on same chart.
- You can specify legend's label with label.

```
plt.plot(x1, y1, label="ax1")
plt.plot(x2, y2, label="ax2")
plt.plot(x3, y3, label="ax3")
```

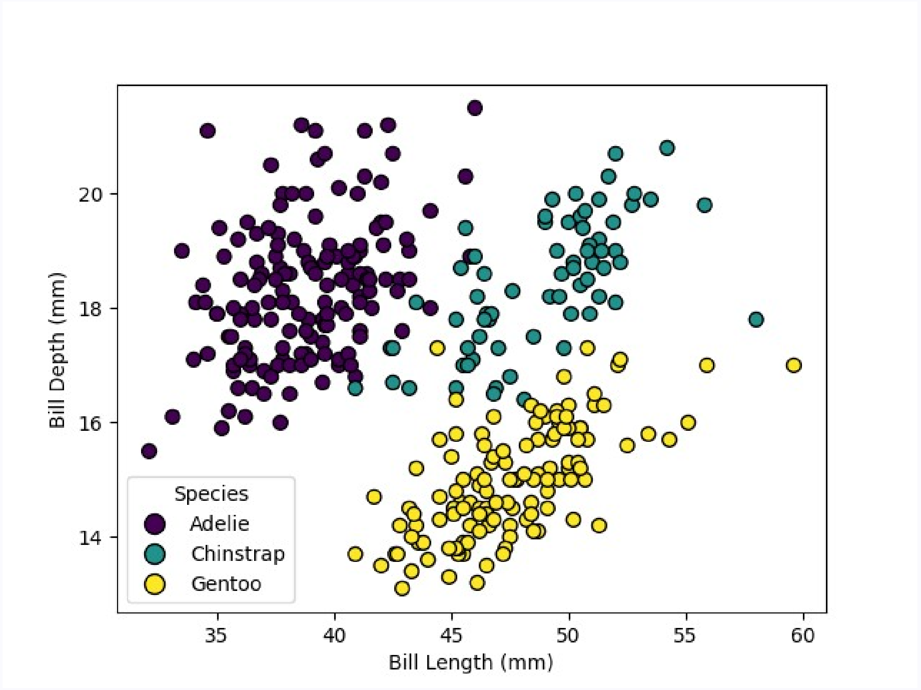


StatCan houshold ownership rates by age group 2011, 2016, and 2021.

Scatterplot

- Pass in the x and y.
- Can optionally specify:
  - Colour each gets (c) as a 1D collection for each entry.

```
plt.scatter(x, y, c=colours)
```

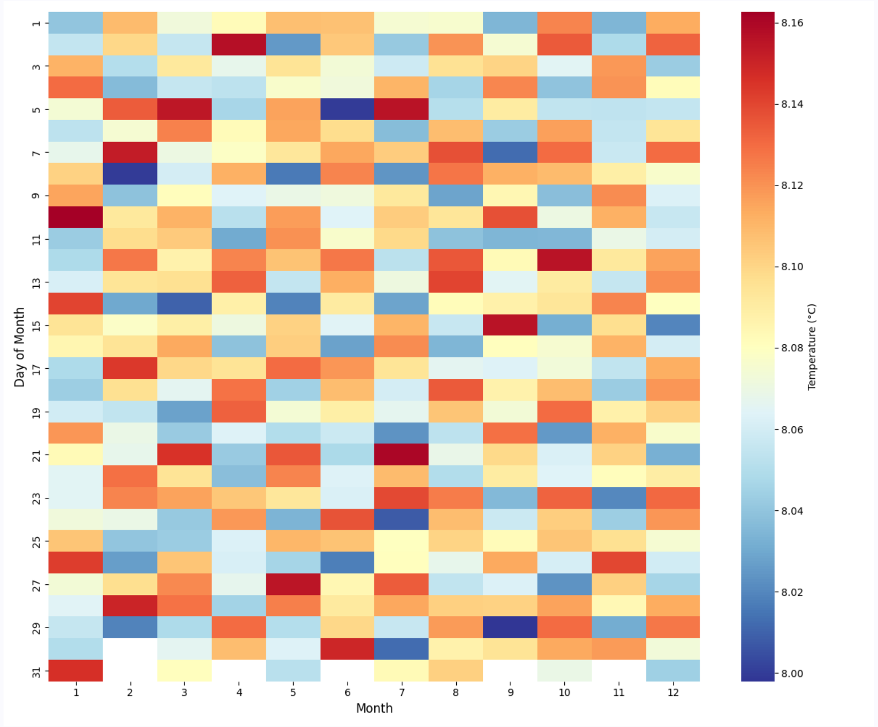


3 different peinguin samples Bill Length vs Bill Depth, Palmer Peinguins.

Heatmap

- You pass a pandas DataFrame.
- Many optional arguments.

```
sns.heatmap(dataframe)
```

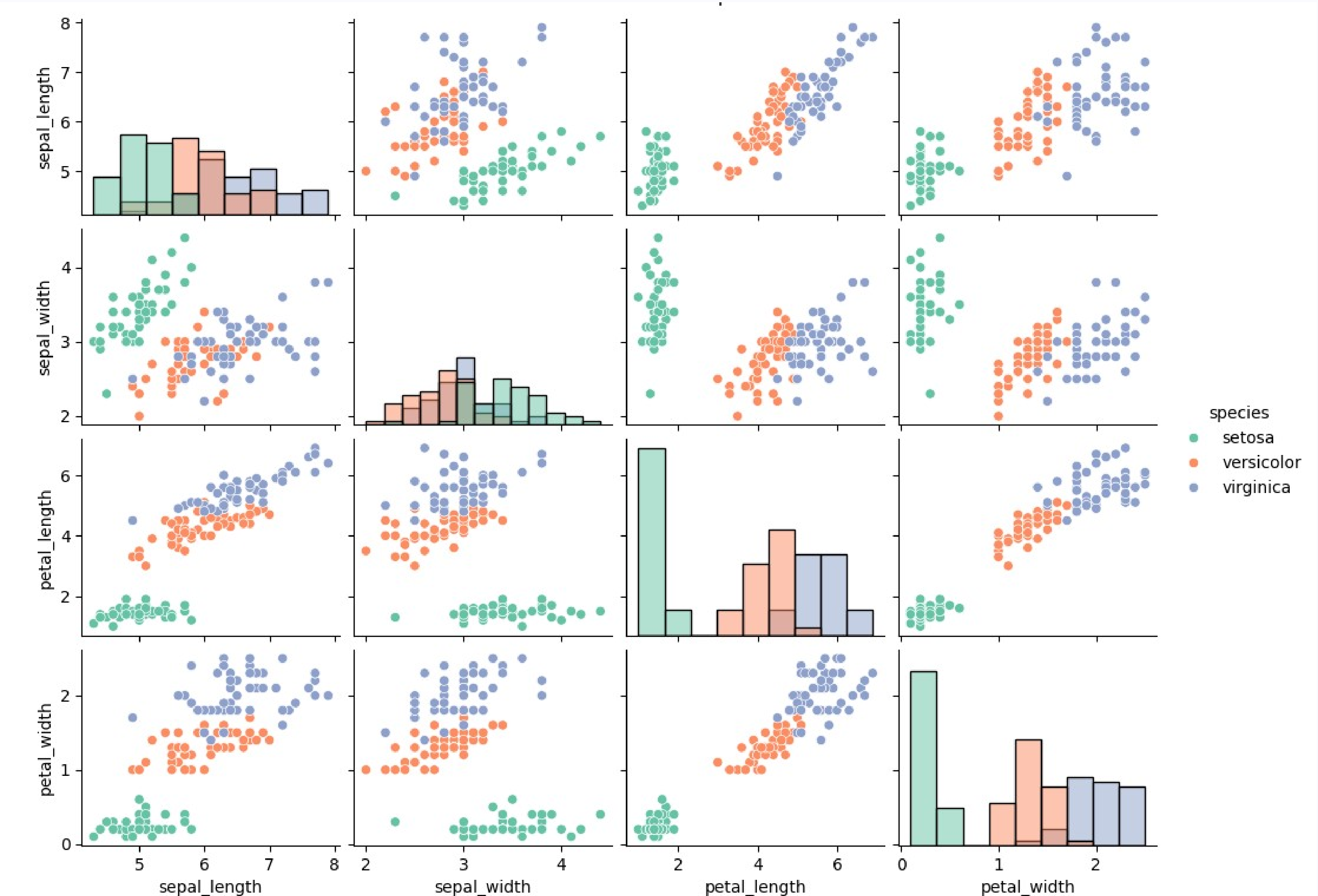


Average daily sea temperatures for each month sector 11417 off coast Vancouver 1.0m depth, ERDDAP

Pair Plot

- Pass in pandas DataFrame as first argument.
- Specify which variable should change colours (hue).
- Can optionally specify:
  - palette: seaborn colour pallete
  - vars: limit which variables to use (otherwise all)
  - diag_kind: “auto”, “hist”, “kde”, None

```
sns.pairplot(data,
    hue=hue,
    palette=palette,
    vars=vars,
    diag_kind="auto")
```


sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica

Machine Learning

DSfS 153-163

Model

- Aims to tell a relationship between variables.

- Is imperfect.

ML — model fitting

- Machine Learning is the process of fitting models to given data to minimise a given loss function.
  - Can think of fitting a linear line on a scatterplot.
- Every model has different parametres and these are learned from the data provided.

Types of training

- Supervised: give data and labels.
- Unsupervised: give data no labels.
- Semi-supervised: some data has labels.
- Online: keep learning as new data comes in.
- Reinforcement: use feedback on performance to update.

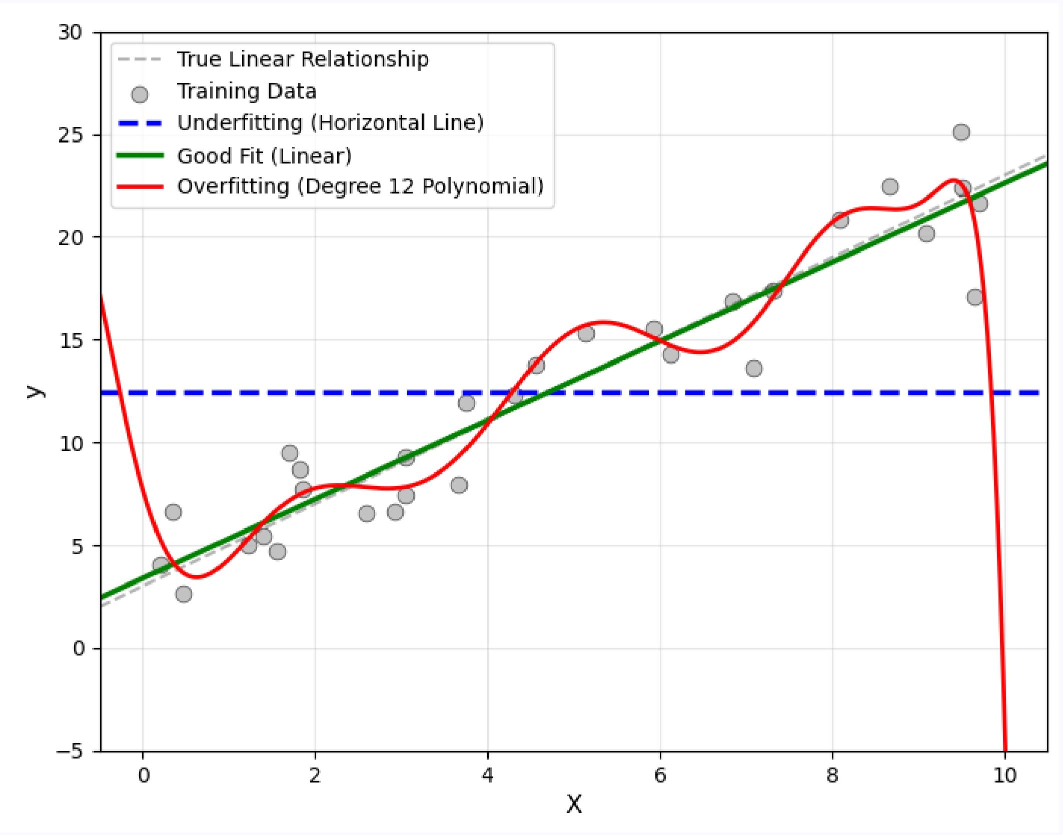
How to train

- We split our data into training and test datasets (≈2:1).
- Fit our model to the training data.
- Test the accuracy of our model on the validation dataset.

```python
import random

def split(xs: list, ys: list, percent: float):
    assert len(xs) == len(ys)

    idxs = [i for i in range(len(xs))]
    random.shuffle(idxs)

    idx = int(len(idxs) * percent);
    train_idxs = idxs[:idx]
    test_idxs = idxs[idx:]

    return (
        [xs[i] for i in train_idxs],
        [xs[i] for i in test_idxs],
        [ys[i] for i in train_idxs],
        [ys[i] for i in test_idxs]
    )

x_train, x_test, y_train, y_test = split(x, y, 0.8)
```

Dangers of ML

- Underfitting: where the model performs poorly on our data.
- Overfitting: where the model performs very well on our provided data but fails when given new data (poor generalisation).



Training Set

What if I evaluate multiple models

- If you fit multiple models on the train data and then choose the model that performs best on the test data, you are meta-training.
    - Test set restricted to performance **only**!
- Split data into train (for training), validation (choosing best model), test (to see performance).

Measuring performance in binary classification

- True Positive: Predict pos, is pos.
- False Positive (Type I error): Predict pos, is neg.
- False Negative (Type II error): Predict neg, is pos.
- True Negative: Predict neg, is neg.

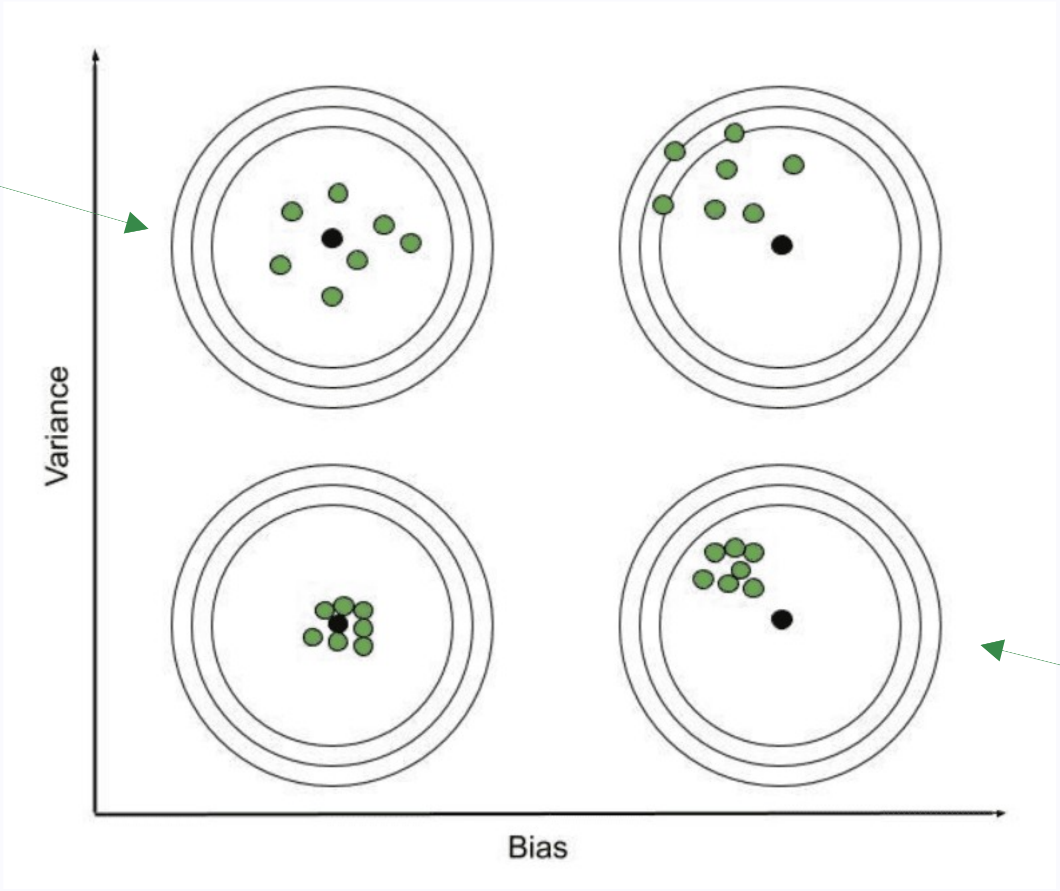
Can create a confusion matrix

	Pos	Neg
Predict pos	TP	FP
Predict neg	FN	TN

- Accuracy: (TP + TN) / total
- Precision: TP / (TP + FP)
- Recall: TP / (TP + FN)
- F1-score: 2 * p * r / (p + r)

Bias vs Variance

overfitting



underfitting

Geekforgeeks

Evaluating non-binary

- Accuracy: # correct / total
    - Classification: giving the right label.
    - Regression: within a given range from correct value.

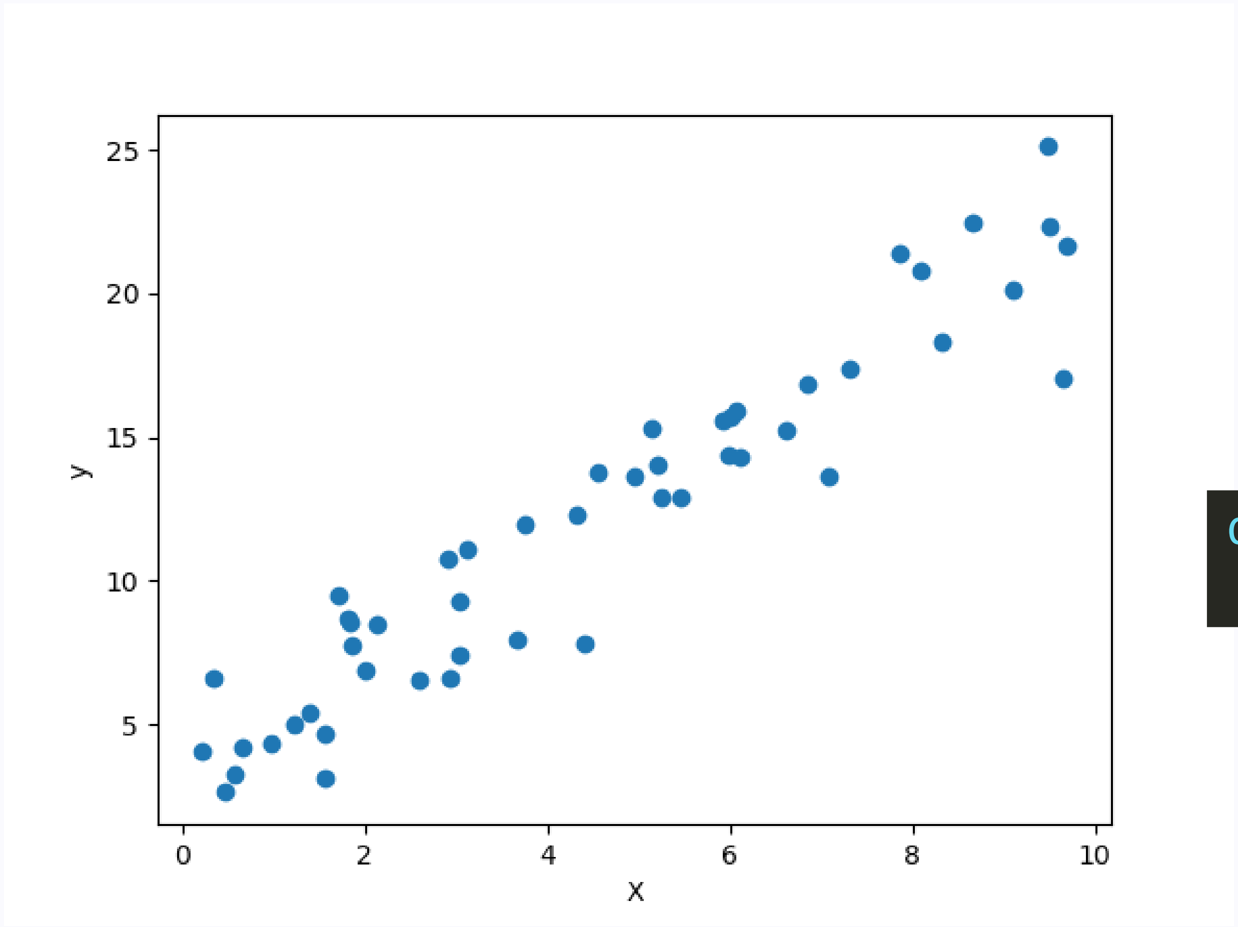
Features

- Variables you give your model to predict an output.
- You should choose features carefully based on domain expertise.
- Sometimes features are not provided and you must extract it from some data.
- Sometimes you have too many features and can reduce it with dimensionality reduction.

Simple Linear Regression

DSfS 185-190

What if our data looks like this?



Looks like a linear relationship!

$$y_i = m x_i + b$$

```
def predict(x, m, b):
    return m * x + b
```

Performance is sum of squared errors

- Error is simply the difference between what we predict and the actual.

```
def error(y_pred, y):
    return y_pred - y
```

- We can maybe calculate our total error of our model!
  - But if one point has error -1 and the other +1 => they cancel.

- Instead we need to square the errors.

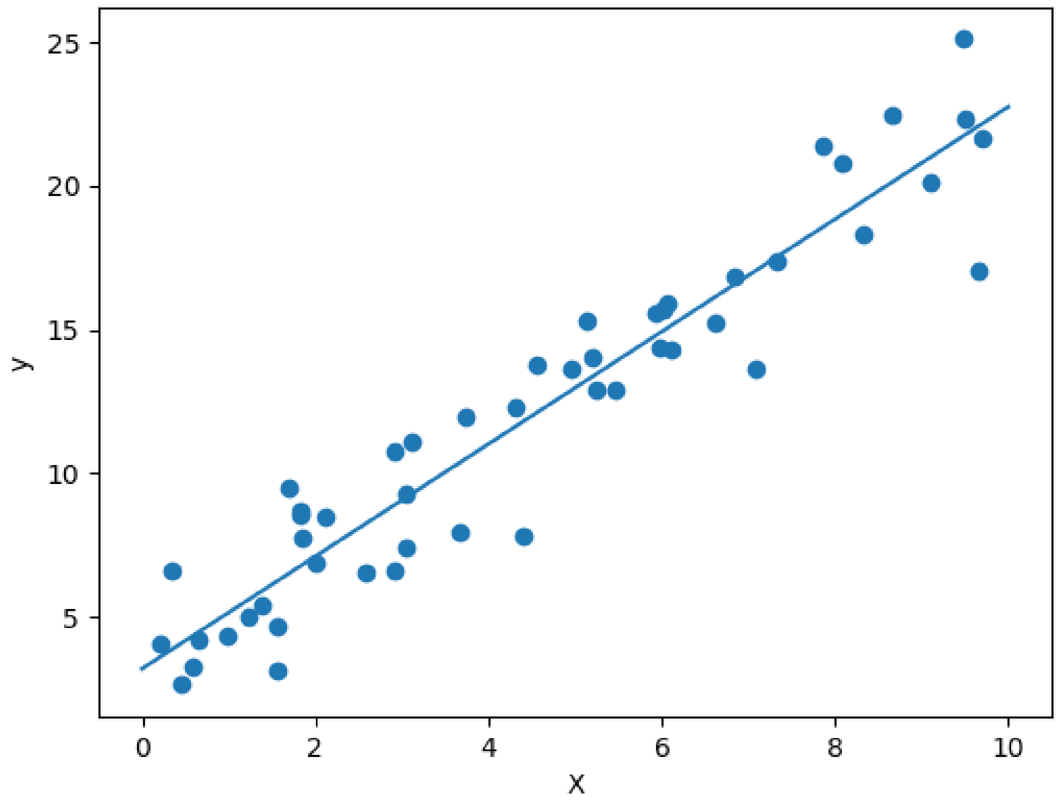
```
def sqr_error(x, y, m, b):
    return sum([error(predict(x_i, m, b), y_i)**2
      for x_i, y_i in zip(x, y)])
```

We should choose *m* and *b* to minimise square error

$$b = \bar{y} - m\bar{x}$$

$$m = \frac{Cov(x, y)}{Var(x)} = \frac{\sum((x_i - \bar{x})(y_i - \bar{y}))}{\sum(x_i - \bar{x})^2}$$

The result



- Doesn’t that look good.
- We need a way to evaluate how well it does besides error.

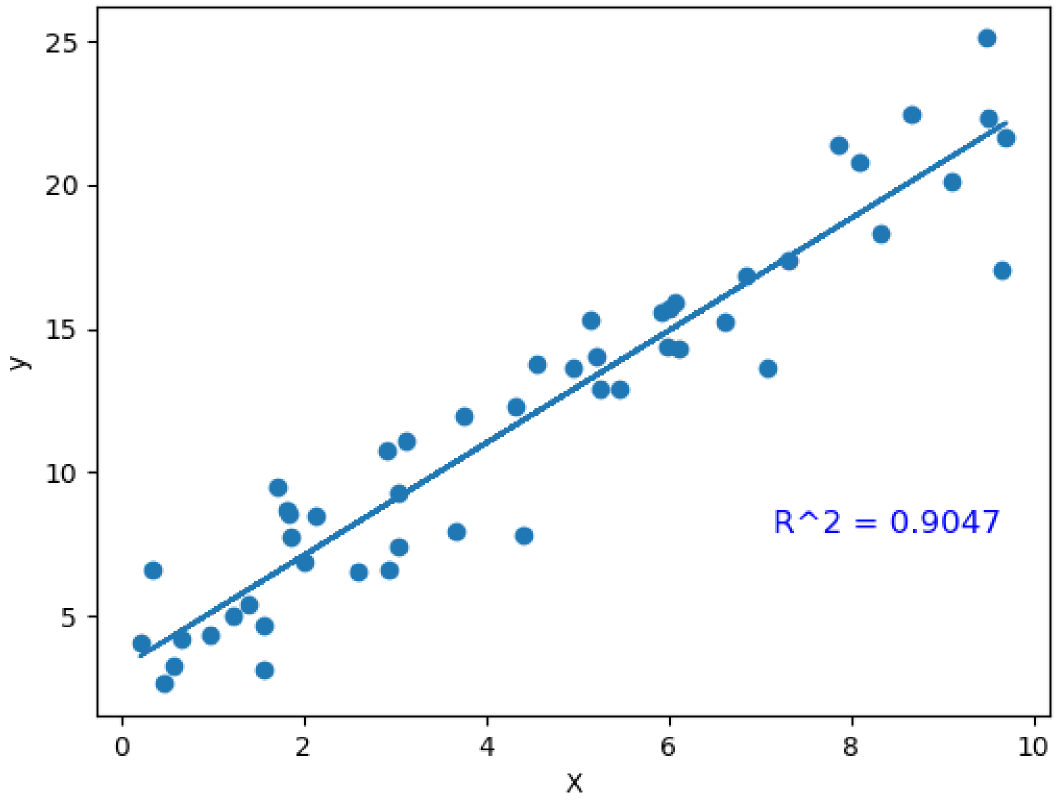
Coefficient of determination (R-squared)

- Measures the total variation of the dependent variable that is captured by the model.

$$R^2 = 1 - \frac{SSE}{SS_{tot}} = \frac{\sum (y_i - \hat{y})^2}{\sum (y_i - \bar{y})^2}$$

- Score of 1 means perfectly encapsulated.

The result



Closed form—so done?

- The closed form is expensive to compute if you have a lot of data.
- We can instead look at the data and infer what would be the best direction to improve our performance.
  - This is called gradient descent.

Gradient Descent

- We first need to create a user-defined cost function that quantifies how “wrong” a prediction is.

$$\underbrace{J(m,b)}_{\text{Cost function as a function of } m \text{ and } b}=SSE$$

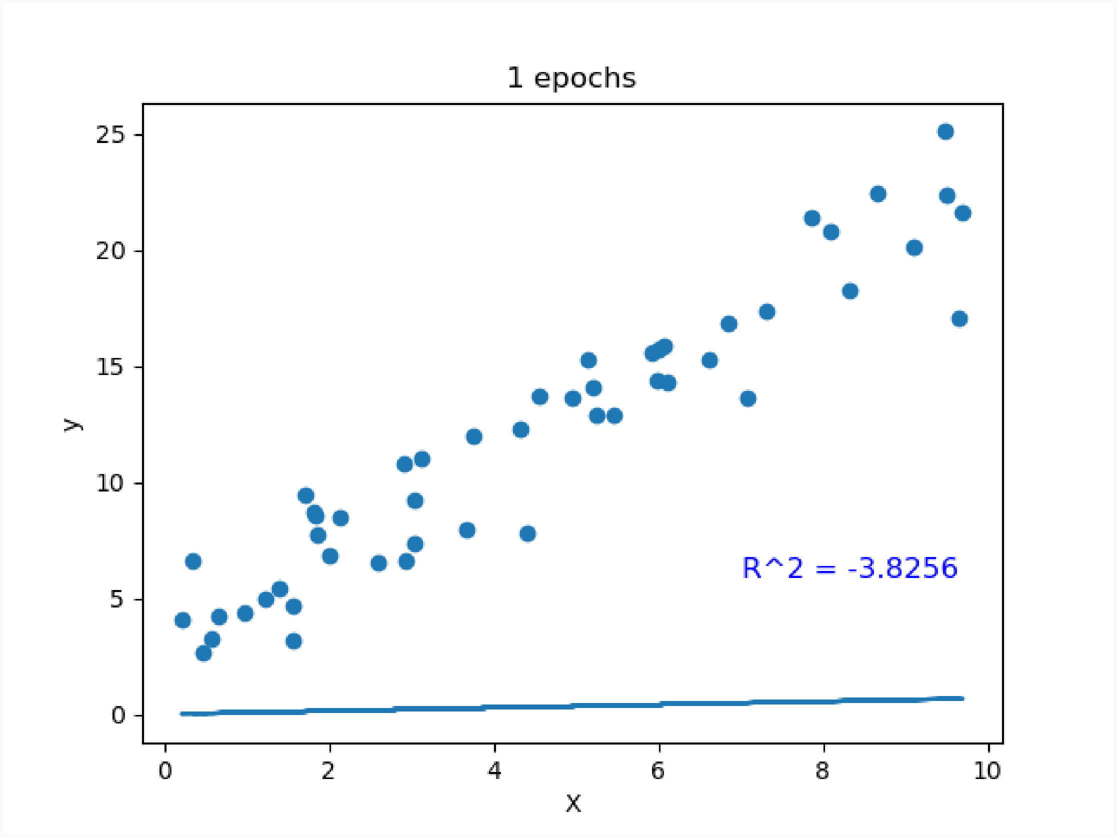
$$J(m,b)=\frac{1}{2n}SSE=\frac{1}{2n}\sum(y_i-m*x_i-b)^2$$

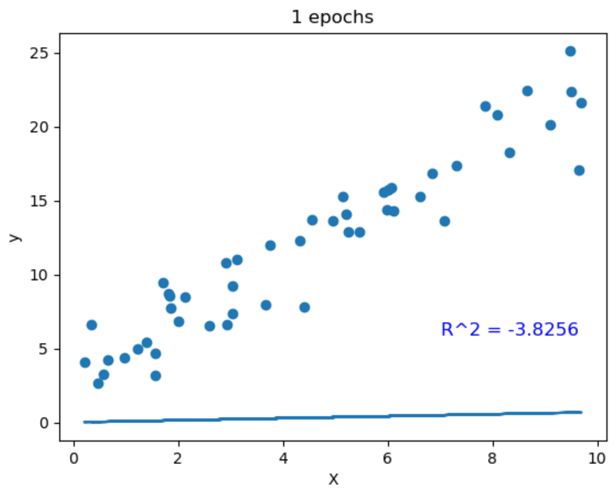
Take derivative relative to each param to know dir of descent.

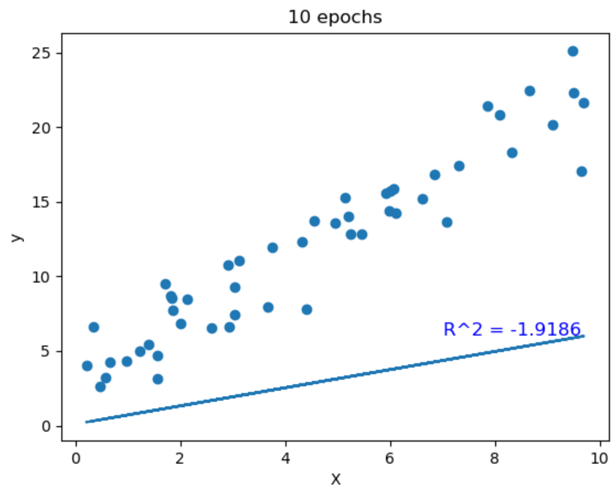
$$\frac{\partial J}{\partial m} = -\frac{1}{n} \sum x_i \left(y_i - m * x_i - b \right)$$

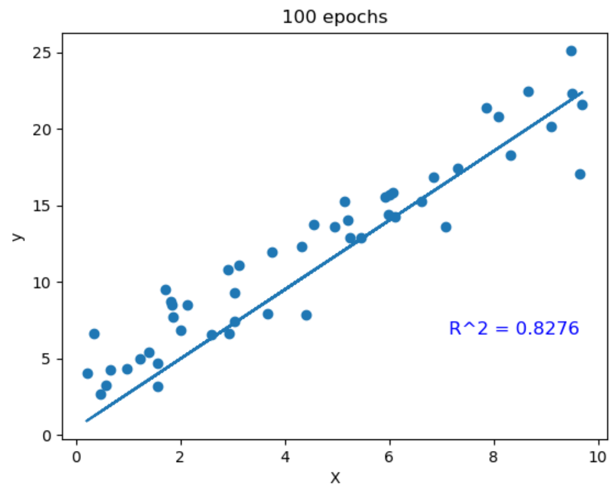
$$\frac{\partial J}{\partial b} = -\frac{1}{n} \sum y_i - m * x_i - b$$

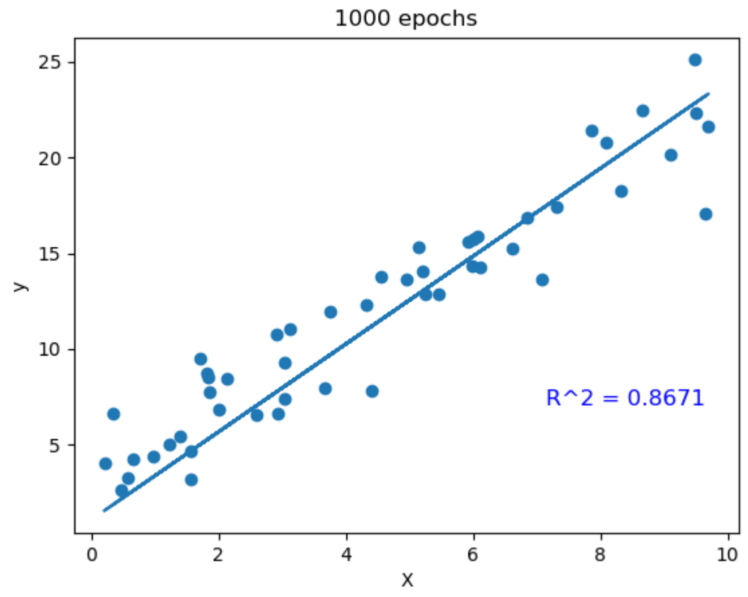
The result

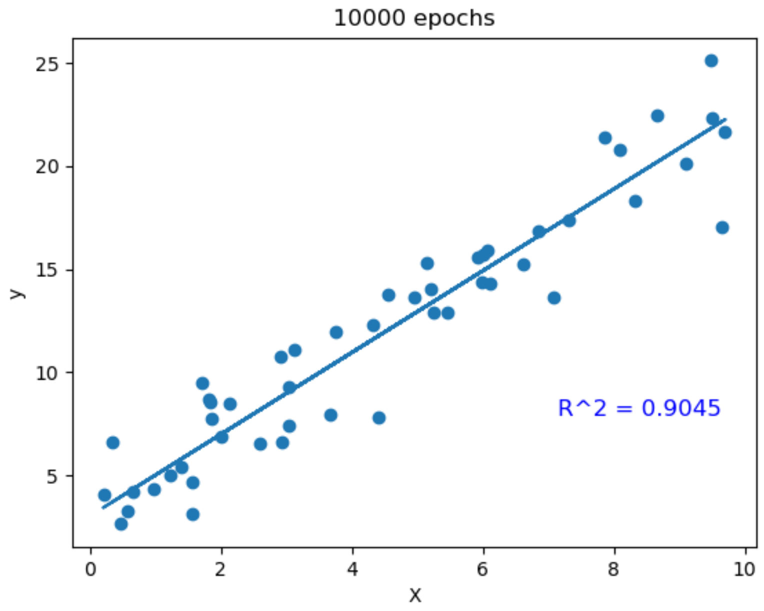



1 epochs
R^2 = -3.8256
x
y


10 epochs
R^2 = -1.9186
x
y


100 epochs
R^2 = 0.8276
x
y


1000 epochs
R^2 = 0.8671
x
y


10000 epochs
R^2 = 0.9045
x
y

Lingo

- Hyperparametres are variables that are not learned but defined by the user (the lr and num epochs).
- Epoch: single pass-through all of the data.

Multiple Regression

DSfS 191-202

What if we have more than 1 feature?

- We **cannot** use simple linear regression, as we predict a value based on a single independent variable.

$$\hat{y} = m * x + b$$

- We need a new formula.

$$\hat{y} = \alpha + \underset{\underset{\text{Feature 1}}{\uparrow}}{\beta_1} * x_1 + \beta_2 * x_2 + \ldots + \underset{\underset{\text{Feature k}}{\uparrow}}{\beta_k} * x_k$$

General formula

$$\hat{y}_i = \alpha + \beta_1 * x_{i,1} + \beta_2 * x_{i,2} + \ldots + \beta_k * x_{i,k}$$

$$\hat{y} = \alpha + X\beta$$

$$\hat{y} = [1; X]\grave{\beta}$$

Beta needs another dimension to do this

```python
import numpy as np

def predict(X, beta):
    return np.matmul(X, beta)
```

We can still use SSE

- But our input data needs to follow three restrictions:
  - Each feature should not be perfectly correlated to another (e.g Celsius and Fahrenheit).
  - You need to have more data than features.
  - All columns in matrix should be linearly independent.

The gradient stay very similar

$$J(\beta_j) = \frac{1}{2n} SSE_j$$

$$\frac{\partial J}{\partial \beta_j} = -\frac{1}{n} \sum_i^n x_{i,j} (y_i - \beta_j * x_{i,j})$$

Can vectorise as

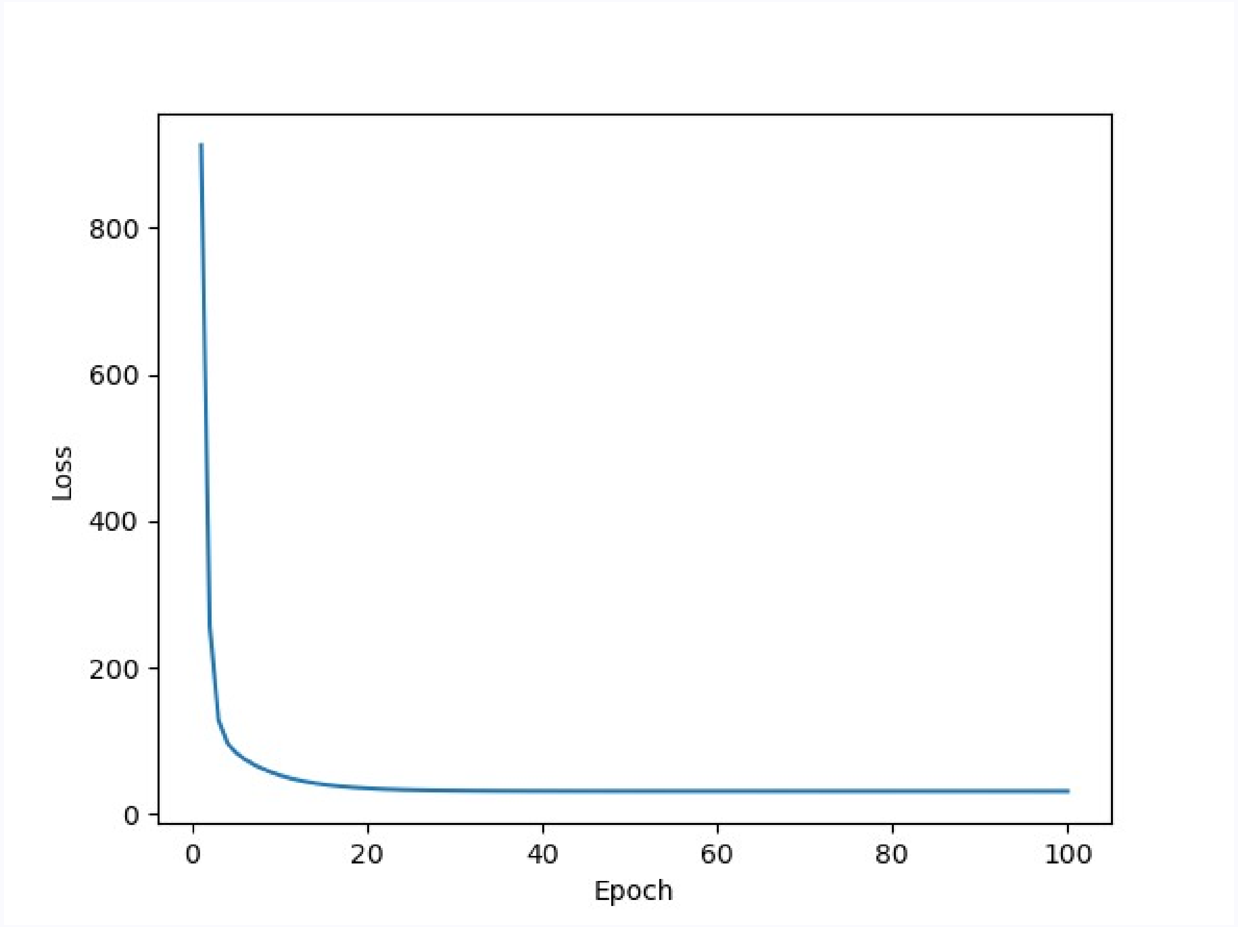
$$\nabla_\beta J = -\frac{1}{n} X^T (y - \hat{y}) \qquad \hat{y} = X\beta$$

General training paradigm

```
def fit(X, y):
    # define beta, num_epochs, batch_size, lr

    with tqdm.trange(num_epochs) as t:
        for _ in t:
            # This is an epoch
            for batch_idx in range(0, X.shape[0], batch_size):
                # This is a step (going through a batch)
                X_batch = X[batch_idx:batch_idx+batch_size]
                y_batch = y[batch_idx:batch_idx+batch_size]
                grad = gradient(X_batch, y_batch, beta)
                beta = beta - lr * grad

            epoch_loss = np.mean(squared_error(X, y, beta))
            t.set_description(f"Loss: {epoch_loss:.4f}")

    return beta, losses
```

Useful external library that show progress

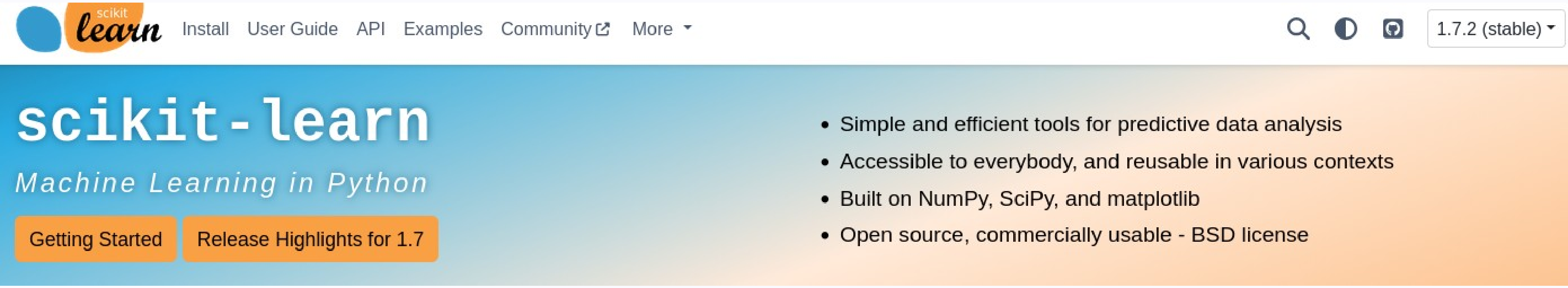
Result



Hyperparametre Tuning

- Domain expert selection
- Manual Search (Trial and error)
- Grid search
- Random search
- Bayesian optimisation (advanced)

Is there a library?



scikit-learn

Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

- Scikit-learn implements many data science tools so that you do not need to re-implement from scratch.