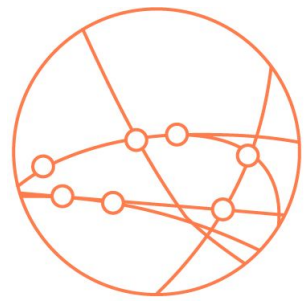




# Hello!

José Rebelo



**LDS SA**

# SLU07 - Regression with Linear Regression

November 26, 2023



# 1. Introduction

# Motivation

- First step in understanding how to use historical data to make predictions
- Simple model
- For many problems, it is the first model you should try



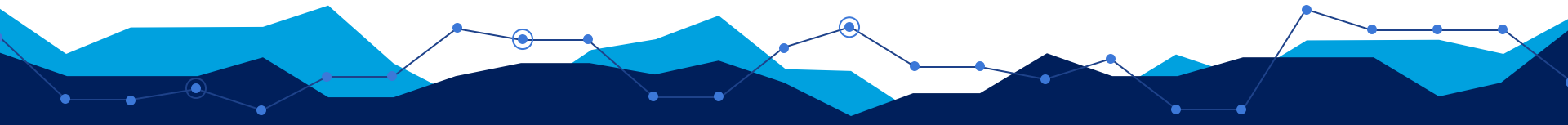
# Overview

- What is a regression?
- Simple Linear Regression
- Multiple Linear Regression
- Solving Linear Regression
- Using Sklearn
- Pros and Cons

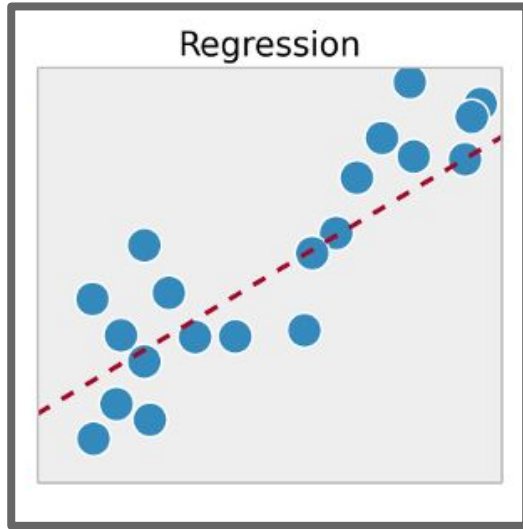




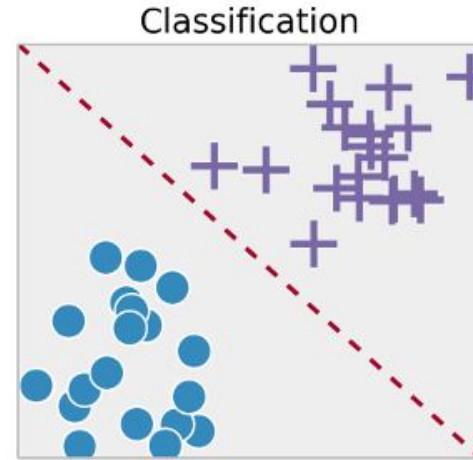
## 2. Topic Explanation



# What is a regression?



SLU 07



SLU 09





# What is a regression?

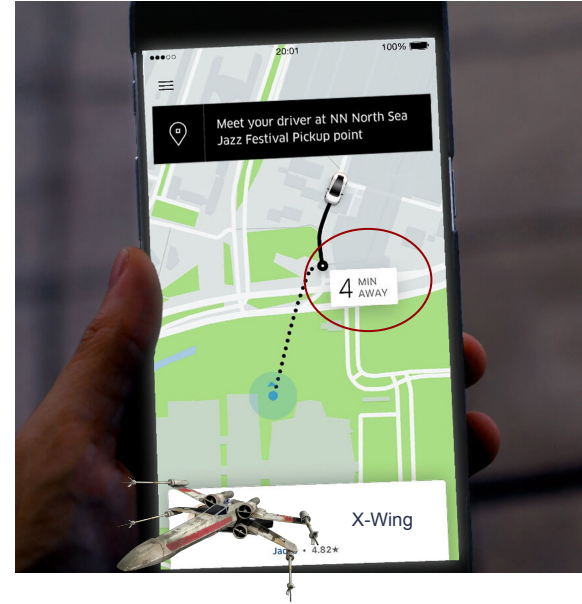
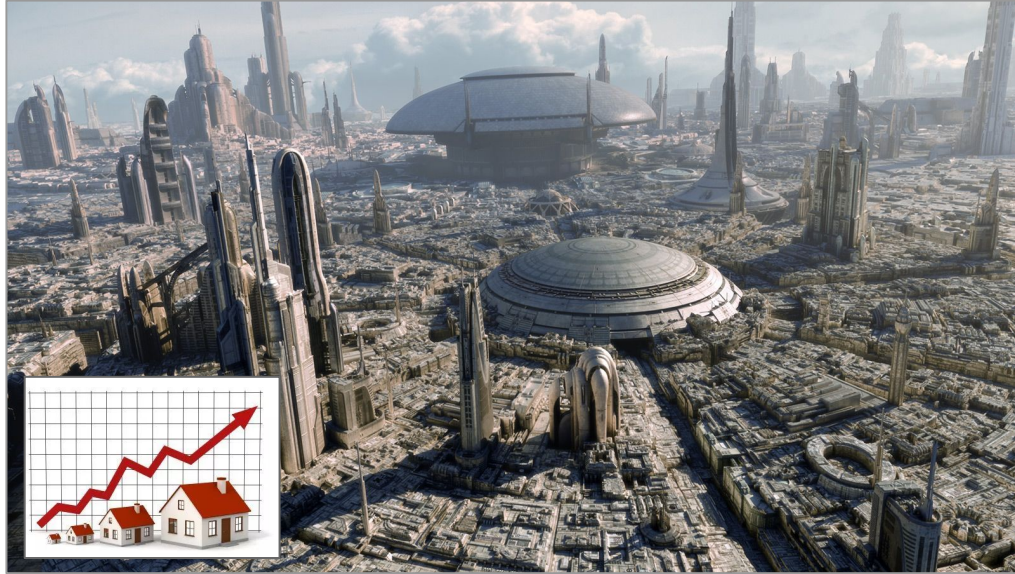
A modeling task which objective is to create a (linear or nonlinear) map between the **independent variables** and a set of **continuous dependent variables**.



**Dependent variables:** referred to as the target, are the values we want to predict

**Independent variables:** features we will use as input to our model to make estimates

# What is a regression?





# Simple Linear Regression

A **linear regression** is a regression where we define our model between the features and target as **linear**.

A **simple linear regression** is a special case of linear regression where we have only **one feature** and one target.



# Simple Linear Regression

The value of the y axis when the input is equal to 0 - **intercept**

The input

$$\hat{y} = \beta_0 + \beta_1 x$$

The prediction along the line.

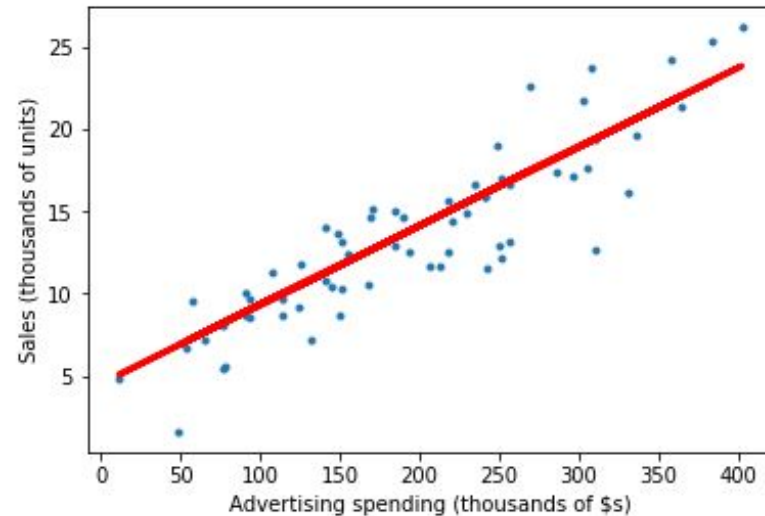
The slope of the line - **coefficient**

```
y_hat = betas[0] + betas[1] * x
```



# Simple Linear Regression


For some given data points, how do we find the best regression line fitting this data?



# How do we evaluate our model?

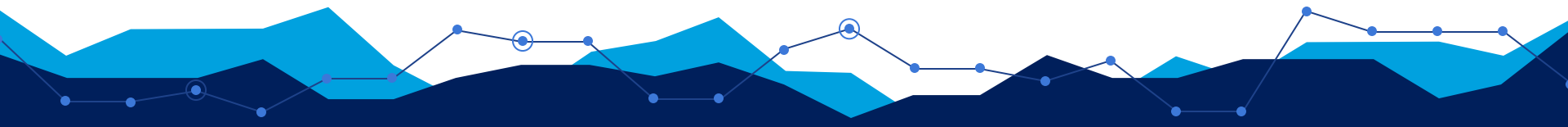
Define a metric to be used, for example, the **Mean Squared Error**:

Based on the difference between the correct targets from our data and our model estimates

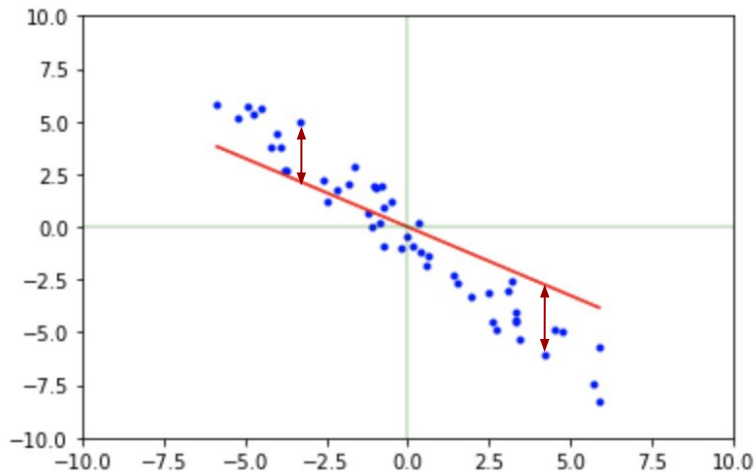
$$MSE(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^m \boxed{(y_i - \hat{y}_i)^2}$$


m - Number of points in the dataset

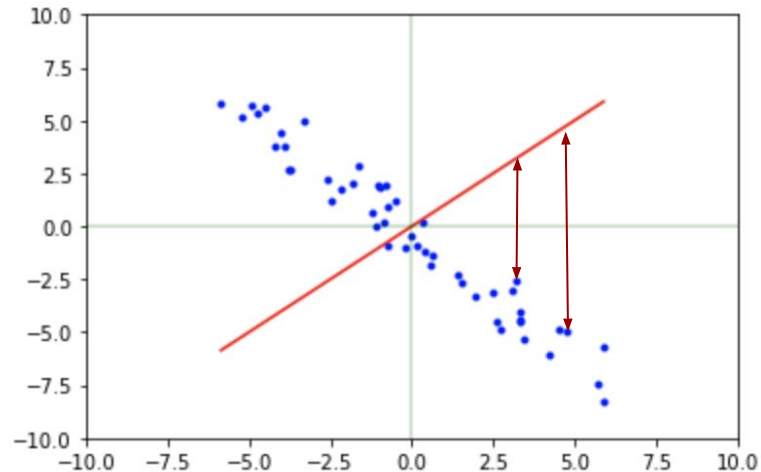
```
((y - y_hat)**2).mean()
```



# How do we evaluate our model?



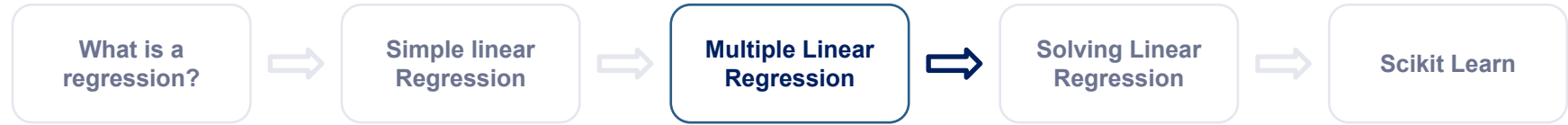
'Mean Squared Error (MSE): 3.3796217422362402'



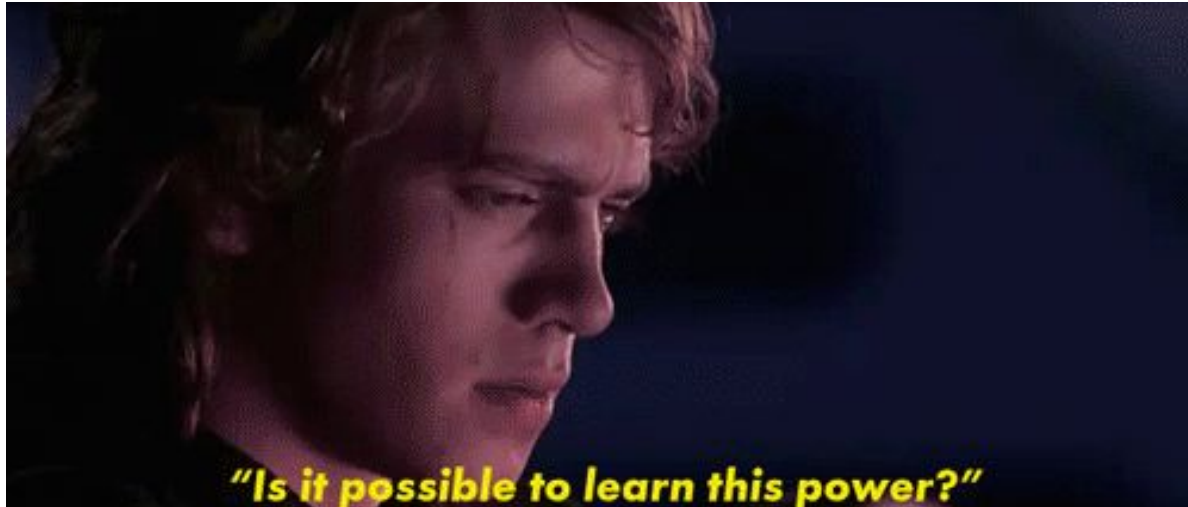
'Mean Squared Error (MSE): 47.20916026219286'





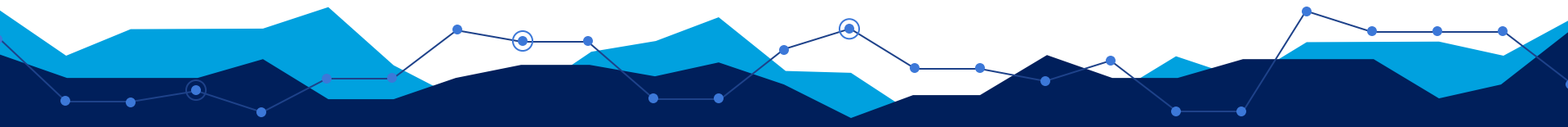


But what if we want to consider more than one feature?



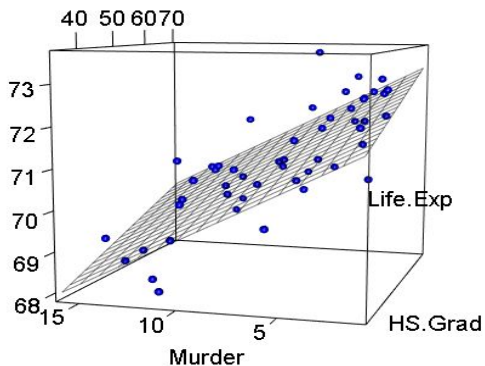
## Multiple linear regression

A **multiple linear regression** is a linear regression where we have **many features** and want to predict one target.



# Multiple Linear Regression

- For each variable added, we add a new parameter
- With 2 variables the visualization becomes a plane
- With more than 2 variables we can not visualize it, but it is called a **hyperplane**



$$\text{Life Expectancy} = \beta_0 + \beta_1 \text{ Murder} + \beta_2 \text{ Health Services}$$



# Multiple Linear Regression

Unrolled expression, with all inputs and coefficients explicit:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \cdots + \beta_n x_n$$

Or we can collapse all of our coefficients and corresponding inputs in a sum:

The value of y when all inputs are 0

The input feature i

$$\hat{y} = \beta_0 + \sum_{i=1}^n \beta_i x_i$$

The prediction

The coefficient for feature i - its weight



# Multiple Linear Regression

Vectorized form:

The value of y when all inputs are 0

The input vector (or matrix, for m samples)

$$\hat{y} = \beta_0 + \vec{\beta} X$$

The prediction

The coefficients vector

Python implementation (for one sample):

```
def output_for_a_linear_regression_with_fancy_name(x, betas):  
    return betas[0] + x.dot(betas[1:])
```



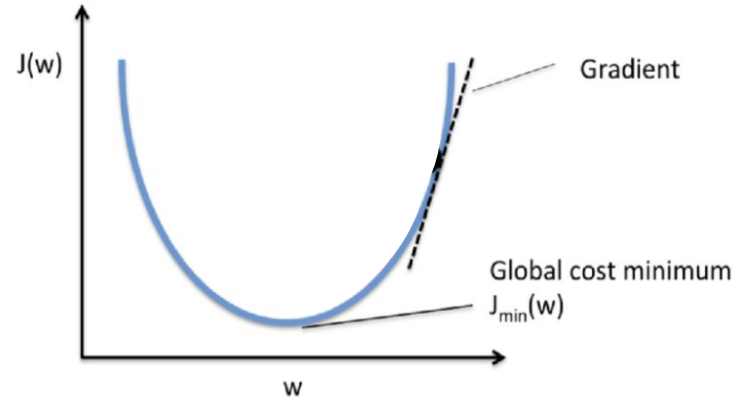
# Cost Function

To optimize the problem we want to find the weights for which our error is minimum.

To do this, we begin by defining the Cost Function

$$J(\beta_0, \dots, \beta_n) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}_i(\beta_0, \dots, \beta_n) - y_i)^2$$

Goal:  $\text{minimum}_{\beta_0, \dots, \beta_n} J(\beta_0, \dots, \beta_n)$





# Iterative Method: Gradient Descent

In order to minimize the coefficients, we can use an iterative method such as gradient descent. The algorithm is the following:

Repeat until convergence:  $\{\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta_0, \dots, \beta_n)\}$

- alpha is a constant, called the learning rate
- for each iteration, all the betas must be updated simultaneously
- each iteration uses all the training examples



# Derivatives of the Cost Function

We need to find the derivatives of the cost function with respect to the weights:

$$\frac{\partial J}{\partial \beta_0} = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i(\beta_0, \dots, \beta_n) - y_i)$$

$$\frac{\partial J}{\partial \beta_j} = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i(\beta_0, \dots, \beta_n) - y_i) x_j$$



# Closed Form Solution

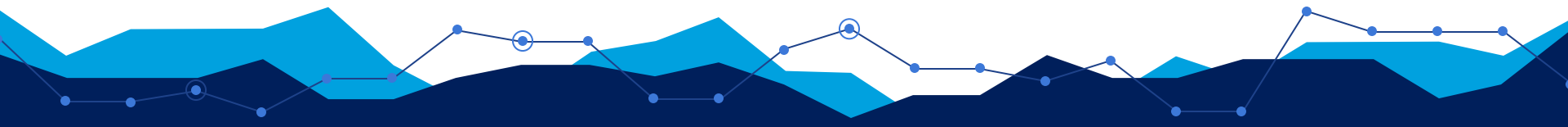
For the **simple linear model**, you can find the weights with:

$$\frac{dJ}{d\beta_1} = 0 \rightarrow \beta_1 = \frac{\sum_i^N (x_i - \bar{x})(y_i - \bar{y})}{\sum_i^N (x_i - \bar{x})^2} = \boxed{\frac{cov(x, y)}{var(x)}}$$

$$\frac{dJ}{d\beta_0} = 0 \rightarrow \boxed{\beta_0 = \bar{y} - \beta_1 \bar{x}}$$



NOTE: If you're interested in seeing the proof, check out the bonus notebook



# Normal Equation

For the general case of the **multiple linear regression**, which can be rewritten as:

$$\hat{y} = \beta_0 + X' \beta \longrightarrow \hat{y} = X \vec{\beta} \quad \text{where} \quad X = [\vec{1} | X']$$

we find the parameters using the **Normal Equation**:

$$\vec{\beta} = (X^T X)^{-1} (X^T \vec{y})$$



**What is a  
regression?**



**Simple linear  
Regression**



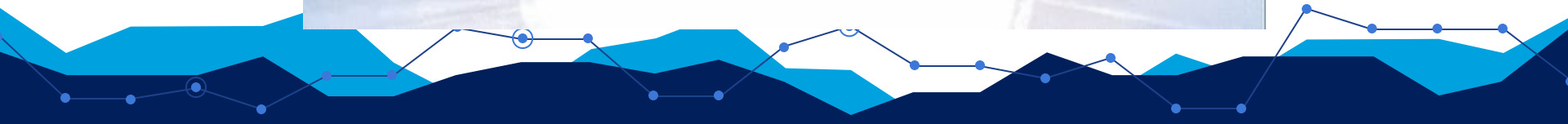
**Multiple Linear  
Regression**



**Ordinary Least  
Squares**



**Solving Linear  
Regression**



# scikit-learn

Machine Learning in Python

Getting Started

Release Highlights for 0.23

GitHub

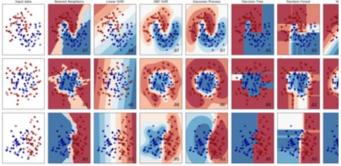
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

## Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...

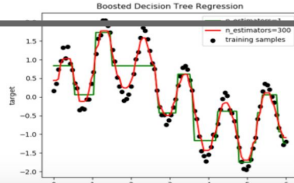


## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...



## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...



Two regressors classes:

- **LinearRegression**
- **SGDRegressor**



# Scikit-learn example

Import some data:

Scikit Learn includes a really nice list of datasets. To use them, go to `sklearn.datasets`

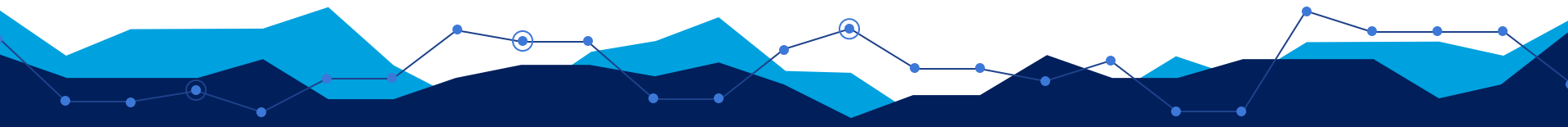
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston

data = load_boston()
#print(data['DESCR'])
```

Boston house  
prices

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	medv
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

median value of  
owner-occupied  
homes in  
\$1000s



# Scikit-learn example: *LinearRegression* class

Here we adjust the betas to the dataset

```
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()
```

```
lr.fit(X, y)
```

```
print('\nTargets for the first 5 rows: \n', y.head(5).values)
```

```
print('\nPredictions for the first 5 rows: \n', lr.predict(X.head(5)))
```

```
print('\nTotal Score\n', lr.score(X, y))
```

Targets for the first 5 rows:

```
[24.  21.6 34.7 33.4 36.2]
```

Predictions for the first 5 rows:

```
[30.00821269 25.0298606  30.5702317  28.60814055 27.94288232]
```

Total Score

```
0.7406077428649427
```

This is the  $R^2$  score

predict is used to make ... predictions!





# Scikit-learn example: *LinearRegression* class

```
import numpy as np

a = pd.Series(lr.coef_, index=x.columns,
              name='Features Coefficients (sorted by magnitude)')
index = a.abs().sort_values(ascending=False).index
a.loc[index]
```

NOX	-17.795759
RM	3.804752
CHAS	2.688561
DIS	-1.475759
PTRATIO	-0.953464
LSTAT	-0.525467
RAD	0.305655
CRIM	-0.107171
ZN	0.046395
INDUS	0.020860
TAX	-0.012329
B	0.009393
AGE	0.000751

Name: Features Coefficients (sorted by magnitude), dtype: float64



# Linear Regression - pros / cons

## Pros:

- Simple Model
- Computationally efficient
- Interpretability of the Output

## Cons:

- Overly-Simplistic
- Categorical features
- Feature scaling is required
- Assumes that there is no multicollinearity (when solving with the normal equation)



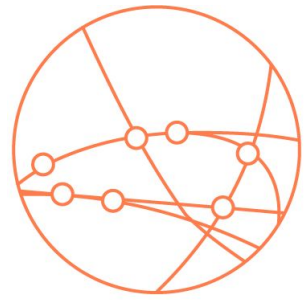


# 3. Recap

# Recap

- Simple linear regressions and Multiple linear regressions: the only difference is the number of inputs (and coefficients)
- Features should not be highly correlated - no multicollinearity!
- You can fit linear regressions with both closed form solutions — eg. ordinary least squares - and iterative methods — eg. gradient descent





**LDS SA**

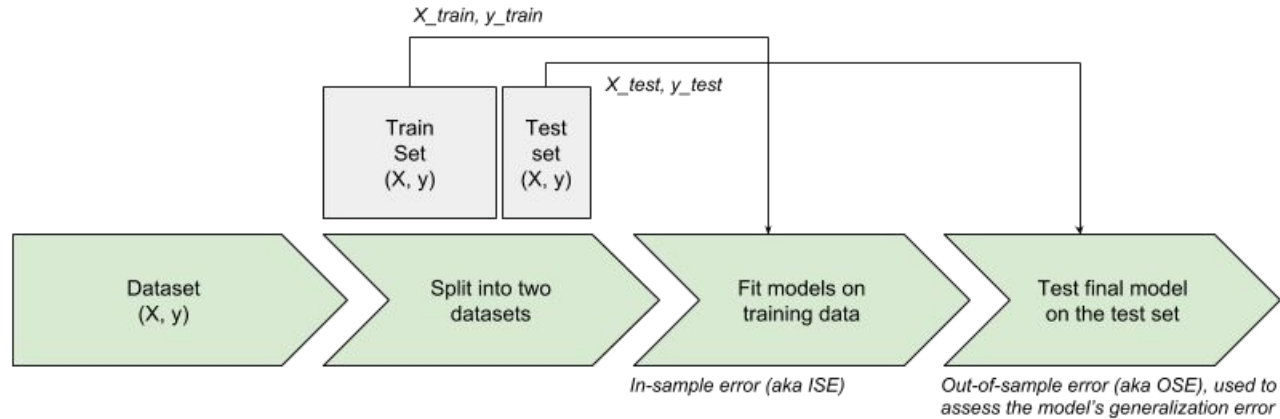
# SLU08 - Metrics for Regression

November 26, 2023



# 1. Introduction

# Motivation



The quality of a regression model is how well its predictions match up against actual values, but  
**how do we actually evaluate quality?**

# Motivation

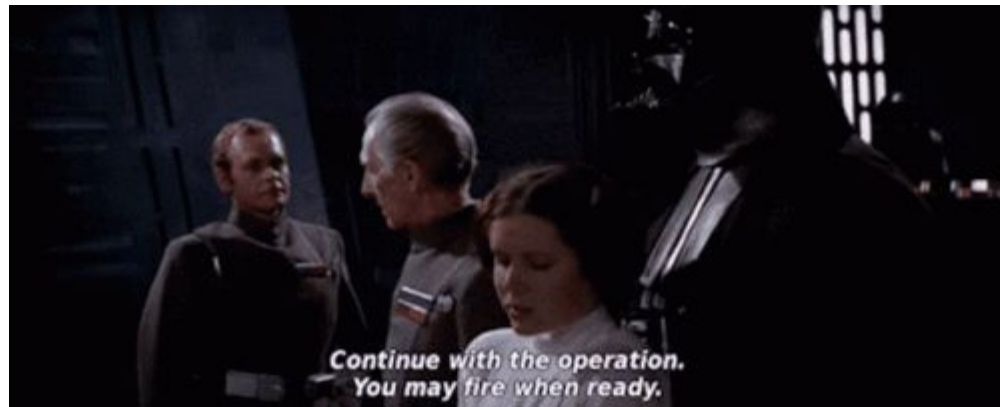
- **Metrics** are what we rely on to **assess models' performance** and **choose between different models**
- Sometimes the metric is previously defined, sometimes you are involved/responsible for the process of defining it
- **Interpreting the metrics correctly** allows you to use them for model selection and parameter assessment
- This presentation is not an exhaustive review of the existing regression metrics. The main purpose is to introduce you to the most common ones, knowing that there are several more out there that may be more appropriate for your future projects





# Overview

- Loss VS Metric
- Metrics for Regression
  - Error
  - Mean Absolute Error (MAE)
  - Mean Squared Error (MSE)
  - Root Mean Squared Error (RMSE)
  - Coefficient of Determination ( $R^2$ )
  - Adjusted  $R^2$  score
- Using the Metrics
  - Hold out method
  - Choosing the right metric





## 2. Topic Explanation

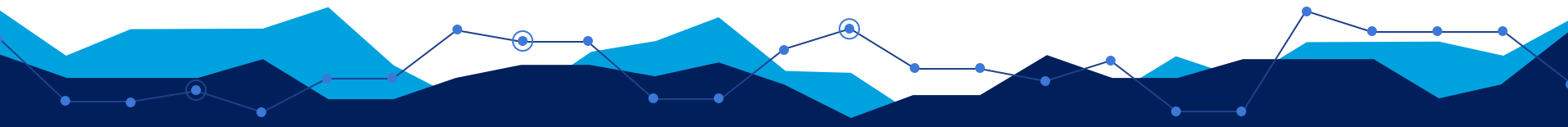
**Loss VS  
Metric**



**Metrics for  
Regression**



**Using the  
Metrics**



## LOSS / Cost

The function that your model tries to optimize during the model training phase.



## METRIC

The function that you want to minimize, related to specific business needs of your organisation.

Sometimes, our function is both a loss and a metric.



**Loss VS Metric**



**Metrics for  
Regression**

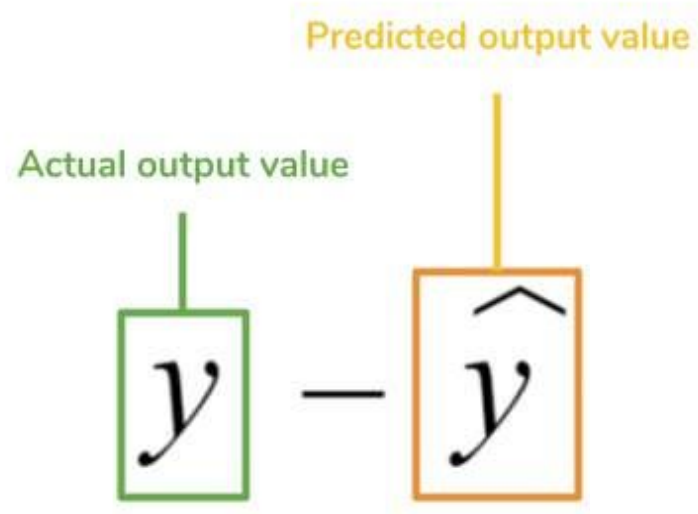


**Using the  
Metrics**



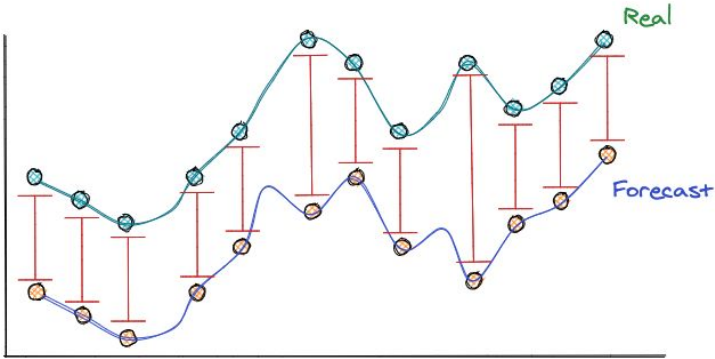
# Error

- ◉ In terms of model performance, error refers to the **difference between the actual output value and the predicted output value**.
- ◉ **The lower** the difference between the observed outcome 'y' and the predicted outcome ' $\hat{y}$ ', **the better** the performance of the model.
- ◉ A higher difference means a worse performance of the model.



# Mean Absolute Error (MAE)

$$MAE = \frac{1}{N} \sum_{n=1}^N |y_n - \hat{y}_n|$$



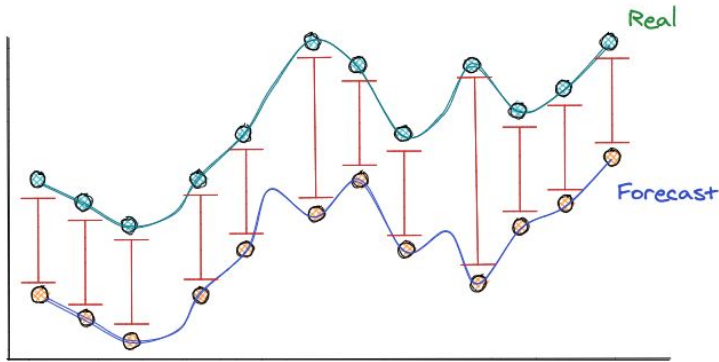
- Corresponds to the **average of the absolute difference between the predicted values and observed values.**
- More robust to outliers.
- Interpretability: The output is measured in the same unit as the target.
- The lower, the better.

Sum all the red segments and divide by the number of points



# Mean Squared Error (MSE)

$$MSE = \frac{1}{N} \sum_{n=1}^N (y_n - \hat{y}_n)^2$$



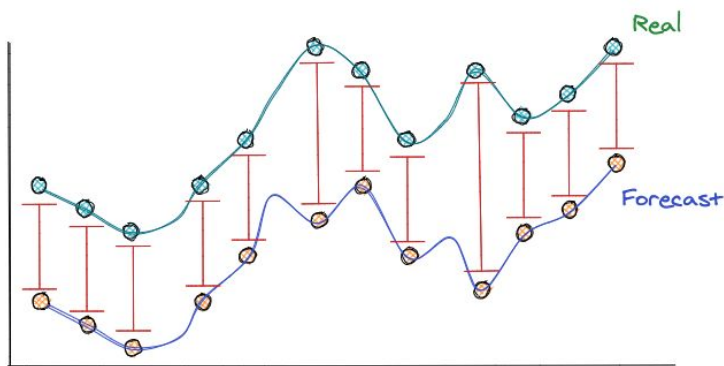
Sum all the red segments squared and divide by the number of points

- Measures the **average of the errors squared**. It basically calculates the difference between the estimated and the actual value, squares these results and then computes their average.
- More sensitive to outliers.
- Interpretability: Loses interpretability due to the squaring. The output is not measured in the same unit as the target.
- The lower, the better.



# Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{MSE}$$



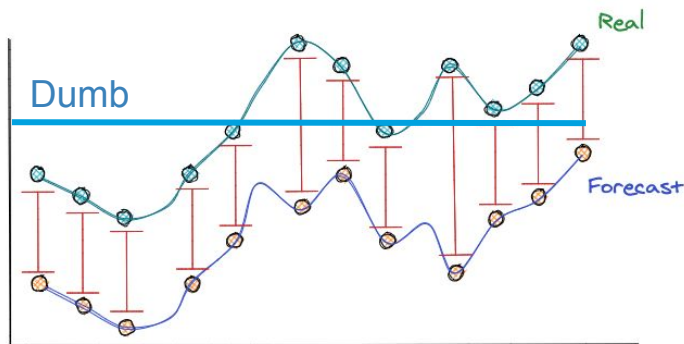
Sum all the red segments squared and divide by the number of points and then square root of everything

- Calculates the **average of the squared errors** across all samples but, in addition, takes **the square root of the result**.
- More sensitive to outliers.
- Interpretability: The output is measured in the same unit as the target.
- The lower, the better.

# Coefficient of Determination ( $R^2$ )

$$R^2 = 1 - \frac{MSE(y, \hat{y})}{MSE(y, \bar{y})}$$

$$\bar{y} = \frac{1}{N} \sum_{n=1}^N y_n$$



- Represents the **proportion of the variance for the dependent variable  $y$  that's explained by the independent variables  $X$ .**
- It has values between - infinity and 1.
- Allows to compare your regression model with a (dumb) predictor that just predicts the mean value of the target ( $R^2 = 0$ ).
- The higher (closer to 1), the better.
- $R^2 < 0$  means that your model is worse than the dumb model.

# Adjusted R<sup>2</sup>

$$R_{adj}^2 = 1 - \frac{N - 1}{N - K - 1}(1 - R^2)$$

- ◉ Just like R<sup>2</sup>, also represents the **proportion of the variance** for the dependent variable y that's explained by the independent variables X, **but adjusts for the number of features in the model.**
- ◉ Used, instead of R<sup>2</sup>, to compare models with different numbers of features.
- ◉ The more “useless” features are added to the model, the lower the Adjusted R<sup>2</sup> value, differently from what would happen with R<sup>2</sup>.
- ◉ The higher, the better.





Loss VS Metric



Metrics for  
Regression



Using the  
Metrics

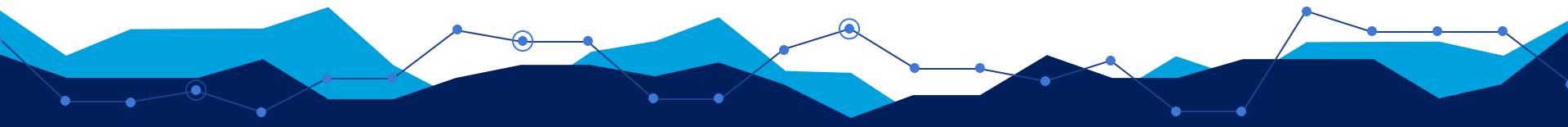


# Using the metrics

You can use your metrics and the hold-out method to choose the best estimator:

```
from sklearn.model_selection import train_test_split  
  
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
```

```
clfs = {  
    'LinearRegressor': clf_1,  
    'SGDRegressor': clf_2  
}  
  
for key, clf in clfs.items():  
    clf.fit(x_train_clf, y_train)
```



# Using the metrics

You can use your metrics and the hold-out method to choose the best estimator:

```
mse = lambda y, y_hat: ((y - y_hat)**2).mean()
```

```
train_score = mse(y_train, y_hat_train)
```

```
test_score = mse(y_test, y_hat_test)
```

Mean Squared Error (MSE)

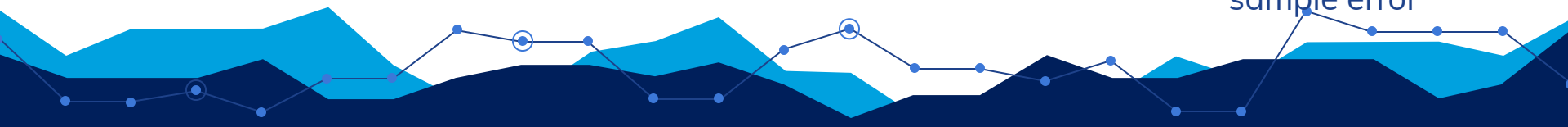
LinearRegression estimator ISE with MSE: 3.325767470418408

SGDRegressor estimator ISE with MSE: 3.3772588724157466

LinearRegression estimator OSE with MSE: 6.472721234757533

SGDRegressor estimator OSE with MSE: 6.171080279404505

Best out of  
sample error



# Using the metrics

## What is the best metric...

- If you want to penalize your bad predictions very much

Choose **MSE** because it measures the average squared error of your predictions.

- If you want to easily interpret the metric value

If you want a metric just to compare between two models from interpretation point of view, choose **MAE** or **RMSE**.

- If you want a more robust to outliers metric

If your data has outliers, you will probably need a metric that is more robust to it, being **MAE** your best option!

- If your model has a high number of features

If choosing between  $R^2$  and Adjusted  $R^2$ , choose **Adjusted  $R^2$** ! This metric will increase if you add useful features and it will decrease if you add less useful predictors.



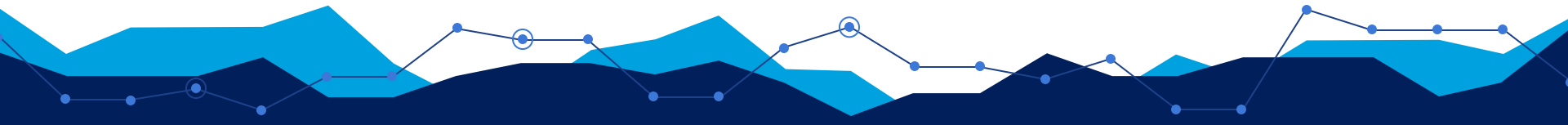


# 3. Recap



# Recap

- Some of the mostly used metrics are computed directly from the error of all the samples: Mean Absolute Error (**MAE**), Mean Squared Error (**MSE**) and Root Mean Squared Error (**RMSE**)
- The coefficient of determination -  **$R^2$**  - can also be used to measure how good the model explains the data and its adjusted version (**Adjusted  $R^2$** ) allows for it to be applied to models with different numbers of predictors
- These metrics can be used to perform **model selection** based on how good they perform in **unseen data**





## 5. Q&A