Regression

Regression

- Alpaydin E (2022) Maschinelles Lernen. De Gruyter, Oldenbourg https://doi.org/10.1515/9783110740196
- Aurélien G (2019) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media, Inc., Sebastopol
- Burkov A (2019) Machine Learning kompakt. mitp Verlags GmbH & Co. KG, Frechen
- Fahrmeir L, Kneib T, Lang S (2019) Regression: Modelle, Methoden und Anwendungen. Springer, Heidelberg
- Frochte J (2021) Maschinelles Lernen: Grundlagen und Algorithmen in Python. Carl Hanser Verlag, München
- Frost I (2018) Einfache lineare Regression. Springer, Wiesbaden
- Harrison M (2020) Machine Learning – Die Referenz. dpunkt, Heidelberg
- Hess T (2006) Methodenspektrum der Wirtschaftsinformatik: Überblick und Portfoliobildung. Ludwig-Maximilians-Universität, München
- Lee WM (2019) Python Machine Learning. John Wiley & Sons, Inc., Indianapolis
- Planing P (2022) Statistik Grundlagen https://statistikgrundlagen.de/ebook/. Abruf am 2022-07-09
- Raschka S, Mirjalili V (2018) Machine Learning mit Python und ScikitLearn und TensorFlow. mitp Verlags GmbH & Co. KG, Frechen

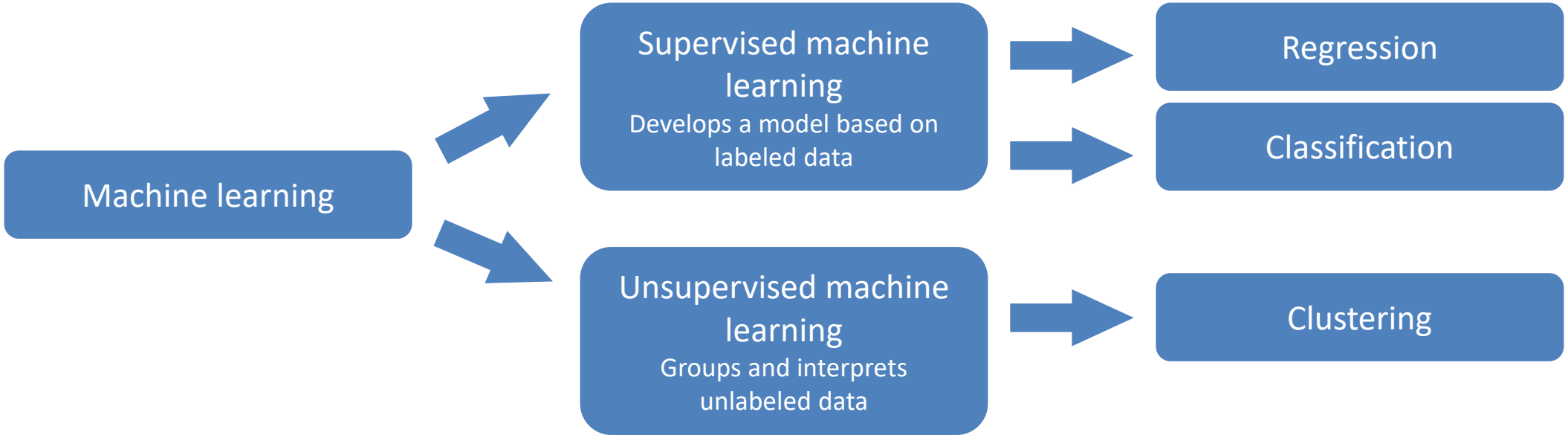
Regression

- Linear Regression with Normal Equation
- Gradient Descent and Linear Regression with Stochastic Gradient Descent
- Polynomial Regression
- Ridge Regression
- Lasso Regression
- Elastic Net and Comparison
- Early Stopping
- Regression Trees

Recap: Supervised vs unsupervised learning

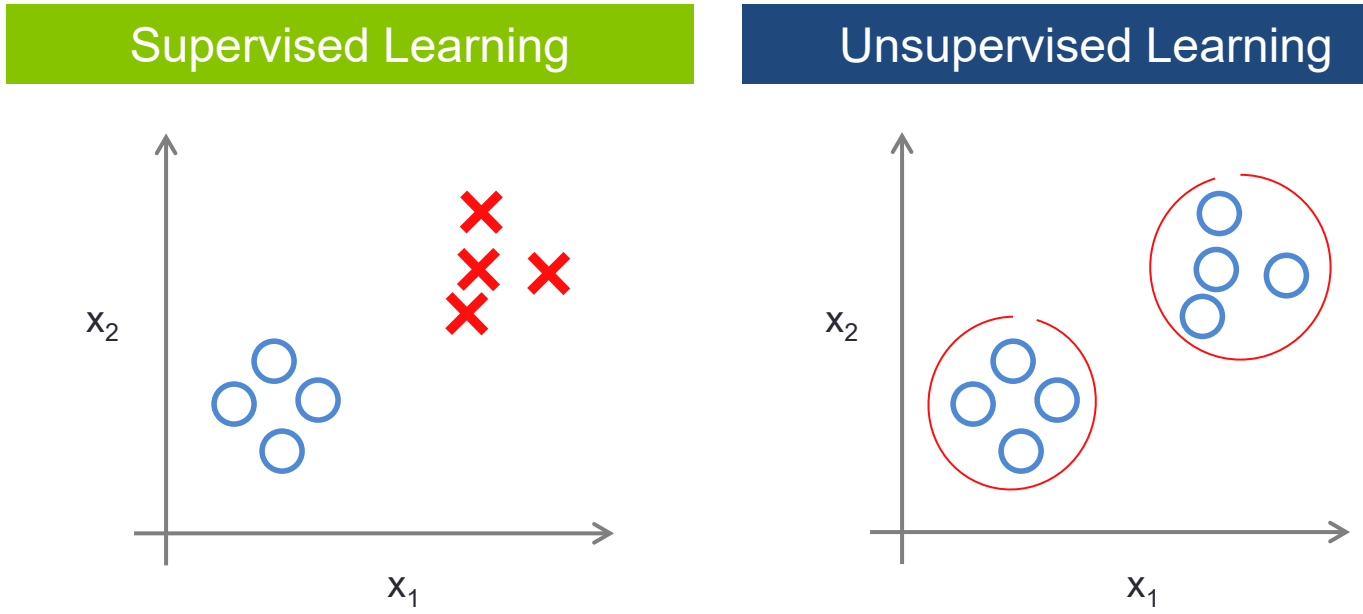
- Supervised machine learning ("Making predictions based on labeled data")
  - Refers to the process and techniques for assigning a label to each object by inferring a function from the data with supervision from a "teacher" (i.e., assigned labels on training samples)
  - Typical supervised learning tasks and approaches consist of regression, classification, or prediction.

- Unsupervised machine learning ("Finding structure in unlabeled data")
  - Refers to learning processes and approaches for which no labels are given and for which a learning system estimates the categorization and structures in its input
  - Unsupervised learning can discover hidden patterns, detect outliers in data, or conduct feature learning
  - Typical unsupervised learning approaches include clustering, profiling, data reduction



Recap: Supervised vs unsupervised learning

- Another example to show differences:



- Supervised Learning:
  - We know *ex-ante* that two categories exist (blue and red)
- Unsupervised learning
  - The result shows two groups (*ex-post)*.
- Supervised learning is analogous to learning with a teacher and then applying that knowledge to new data.

Source: Andrew Ng

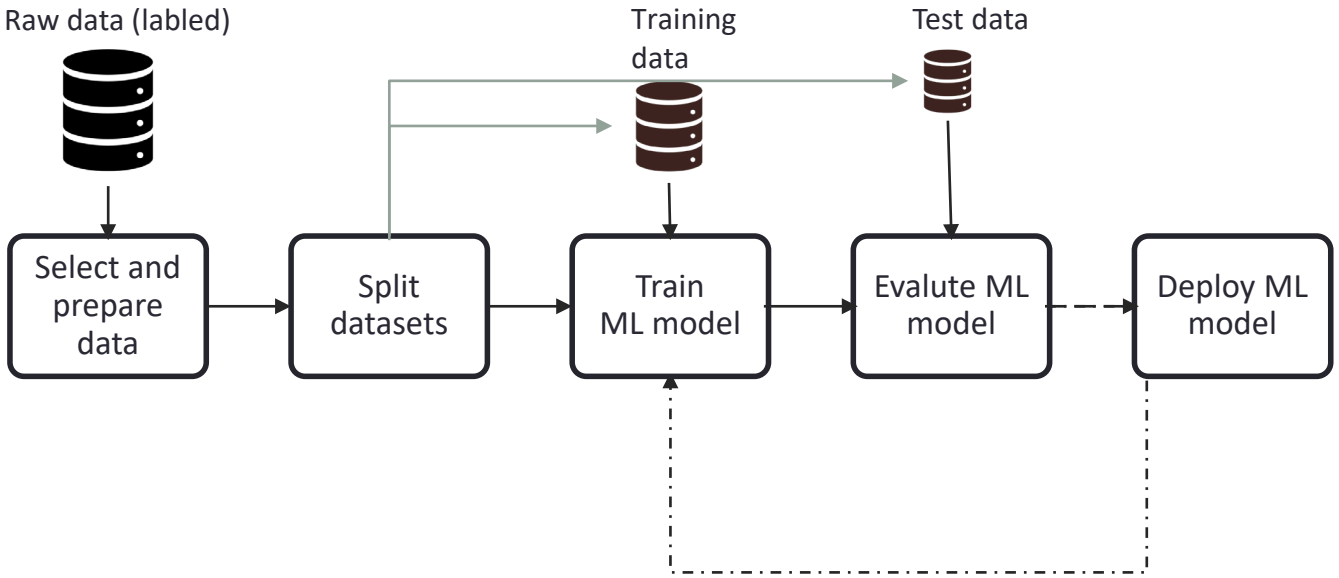
Recap: Supervised vs unsupervised learning

- Regression is considered as an instance of supervised learning since numeric values of a test set are known and used to train the algorithm

- Clustering is an unsupervised approach
  - No training data is needed and used
  - Clustering is often used to “reveal” characteristic groups
  - Clustering is discussed in the later lecture

Regression process

- 1. Select and prepare data
  - Choosing features and manipulating the dataset
- 2. Split dataset
  - Train and test dataset
- 3. Train ML model
  - Input train dataset into a machine learning model
- 4. Evaluate ML model
  - If desired performance is not reached: tune the model and repeat Step 3, otherwise deploy ML model



Training data and test data

- In machine learning, we usually split our data into a training and test set

- Training Set
  - Used to learn the model
  - Often around 60-80% of the raw data

- Test Set
  - Separated from the training set
  - Used *only* to evaluate the accuracy of the learned model over subsequent data
  - No more learning / adjustment of parameters when applying the test set
  - Disjunct from the training data

- There are the advanced mechanism for splitting, training, and validating the data, e.g., cross-validation.

Regression

- **Linear Regression with Normal Equation**
- Linear Regression with (Stochastic) Gradient Descent
- Polynomial Regression
- Ridge Regression
- Lasso Regression
- Elastic Net and Comparison
- Early Stopping
- Regression Trees

- In the following, we base on the lectures known from „Data Analytics with Python“ to introduce the core concepts that should be known related to ‚regression‘

- If you are familiar with those slides and the knowledge, please feel free to continue with the next chapter.
  - Otherwise, please study the following ‚Recap‘ slides

Recap: Regression

- Introduction:
  - Linear regression, also known as simple linear regression or bivariate linear regression, is used to explain the value of a dependent variable based on the value of an independent variable.
  - One could use linear regression to understand whether we can explain exam performance based on revision time
    - The dependent variable would be "exam performance", measured from 0-100 marks, and
    - The independent variable would be "revision time", measured in hours
  - One could use linear regression to understand whether we can explain cigarette consumption based on smoking duration
    - The dependent variable would be "cigarette consumption", measured in terms of the number of cigarettes consumed daily, and
    - The independent variable would be "smoking duration", measured in days)

- Characteristics:
  - Contrary to correlation (which has the purpose of examining the strength and direction of the relationship), regression focuses on the relative impact
  - Regressions are a frequently used method in statistical analysis
  - Most of the applications can be assigned to one of the two following categories:
    - **Quantification** of the relationship between a dependent variable y and one or several independent variables x.
      This way, one can also identify the x_j that affects y, and those x_i that are redundant. (as part of the Exploratory Data Analysis)
    - **Prediction** of the value of a dependent variable y (based on a model built by regression) using the independent variable x. (as part of Machine Learning)

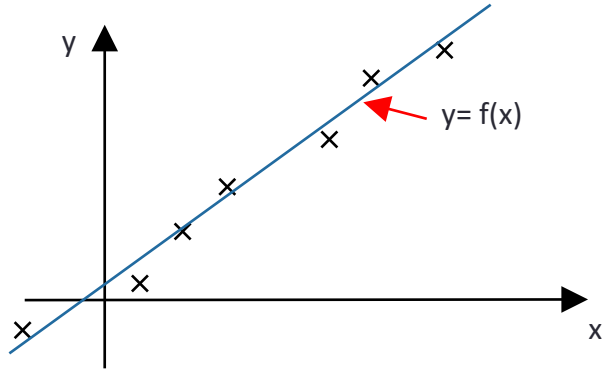
Recap: Regression

- Regression in Exploratory Data Analysis (EDA) or the context of Machine Learning (ML)
  - Regression can be used as part of the Exploratory Data Analysis (EDA) or in Machine Learning (ML) context.
  - Our focus [in Data Analytics with Python] is on EDA and in that context, regression offers an algorithm to draw an optimized (straight) line between two or more variables, which helps to understand the relationship between the variables.
  - In contrast, ML typically uses regressions to predict an unknown variable [which is the focus of the course *Data Science and Machine Learning*]

≪ Regression can use used for exploratory data analysis and for machine learning ≫

Recap: Regression

- Notation:
  - A (1-dimensional / n-dimensional = multiple) regression can be expressed as
    - $y = f(x) + e$
    - $y = f(x_1, x_2, \ldots, x_n) + e$
    - With y being the dependent and x the independent variable(s)



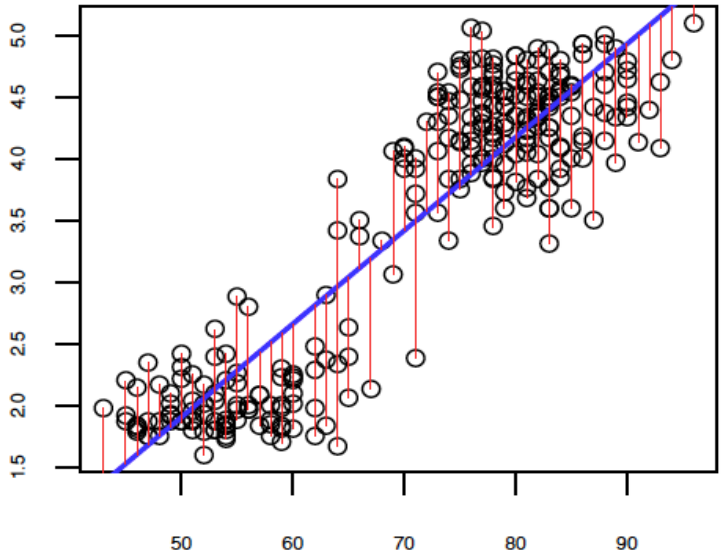
  - Independent variables are also known as repressors or explanatory variables
  - e denotes the error (also called residual or it is often recommended to use the term noise)
  - This generic statistical model can also be described mathematically; we will first focus **on linear regression**, meaning a regression between one dependent variable and a single independent variable:

$$Y = \beta_0 + \beta_1 \cdot X + \epsilon$$

  - … and with fitted values:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot X \qquad e = Y - \hat{Y}$$

    - with the residuals
    - for example, how to visualize a fitted model and error terms, see the following picture on the right



  - The fitting procedure is done by statistic programs, with the aim, …
    - … to find $\hat{\beta}_0, \hat{\beta}_1$ in a way to minimize $\sum_{i=1}^{n} e_i^2$ when n data of (x_i, y_i) are observable.

Recap: Regression: Linear regression

- Conditions for a (simple) linear regression:

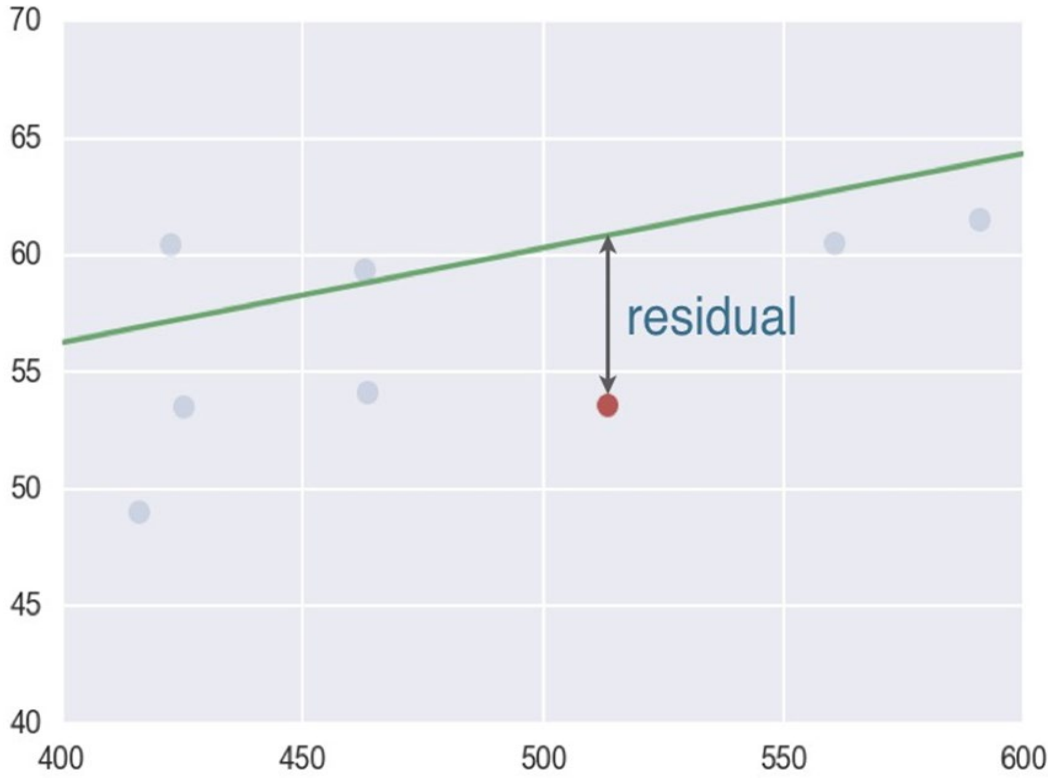
- The dependent variable Y has a linear relationship to the independent variable X.
  - You might use a scatterplot and show that a linear relationship exists for X Y (e.g., see the first Figure on the last page) while the residual plot shows a random pattern (e.g., no non-random inverted u-shaped curve).

- For each value of X, the probability distribution of Y has the same standard deviation σ.
  - If this condition is satisfied, the variability of the residuals will be relatively constant across all values of X, which is easily checked in a residual plot.

- For any given value of X, …
  - … the Y values are independent, as indicated by a random pattern on the residual plot.
  - … the Y values are roughly normally distributed (e.g., symmetric).
    - A slight skewness is ok if the sample size is large; you might use a histogram to show the shape of the distribution.

Recap: Regression: Linear regression

- What do we mean when talking about residuals:
  - A residual is the vertical distance between a data point and the regression line.
  - Each data point has one residual.
    - They are positive if they are above the regression line and negative if they are below.
    - If the regression line passes through the point, the residual is zero.
  - In other words, the residual is the error that is not explained by the regression line.
  - The residual can also be expressed with an equation and the residual thus reflects the difference between the observed value of the dependent variable (y) and the predicted value.



Recap: Regression: Linear regression

- The objective of linear regression is to …
  - … find the straight line (also called [least squares] regression line), which best represents observations in a bivariate data set.

- Given that *Y* is a dependent variable, and *X* is an independent variable, the population regression line is:
  - $Y = \beta_0 + \beta_1 X$
    - with β_0 as a constant,
    - β_1 is the regression coefficient,
    - X is the value of the independent variable,
    - Y is the value of the dependent variable.

- Given a random sample of observations, the population regression line is estimated by:
  - $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$
    - with $\hat{\beta}_0$ is a constant,
    - $\hat{\beta}_1$ is the regression coefficient,
    - x is the value of the independent variable,
    - $\hat{y}$ is the *predicted* value of the dependent variable.

Recap: Regression: Linear regression

- Using Normal Equation to define a regression line
  - As mentioned, there are many computational tools to find $\hat{\beta}_0$ and $\hat{\beta}_1$
  - However, you can also do it by hand, with the following equations.

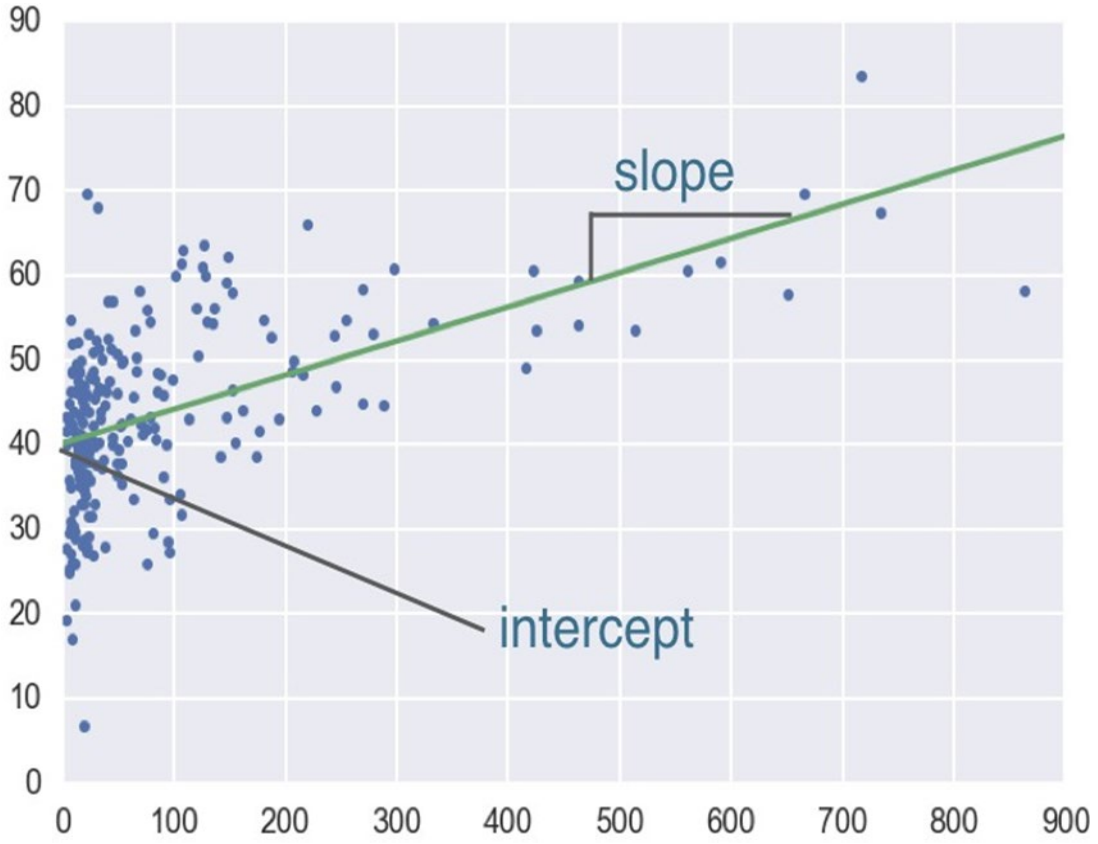
$$\hat{\beta}_1 = \Sigma [(x_i - \bar{x})(y_i - \bar{y})] / \Sigma [(x_i - \bar{x})^2]$$
$$\hat{\beta}_1 = r * (s_y / s_x)$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 * \bar{x}$$

    - with $\hat{\beta}_0$ is the constant in the regression equation,
    - $\hat{\beta}_1$ is the regression coefficient,
    - r is the correlation between x and y,
    - x_i is the *X* value of observation *i*,
    - y_i is the *Y* value of observation *i*,
    - $\bar{x}$ is the mean of *X*,
    - $\bar{y}$ is the mean of *Y*,
    - s_x is the standard deviation of *X*,
    - s_y is the standard deviation of *Y*.

  - see the following pages for a hand on example

Recap: Regression: Linear regression

- Properties of the Regression Line:
  - The regression line minimizes the sum of squared differences between …
    - …. observed values, which are indicated by the y values and
    - … predicted values, which are reflected by the ŷ values that are from the regression equation.
  - The regression line passes through the mean of the *X* values ($\bar{x}$) and the mean of the *Y* values ($\bar{y}$).
  - The regression constant ($\hat{\beta}_0$) is equal to the y-intercept of the regression line.
  - The regression coefficient ($\hat{\beta}_1$) is the average change in the dependent variable (*Y*) for a 1-unit change in the independent variable (*X*).
    It is the slope of the regression line.
  - *Note*: The regression line is the only straight line with all these properties.



Recap: Regression: Evaluation criterion

- It is necessary to evaluate how well the regression model fits the data
  - We want to measure how well the parameters of the regression model have been set
  - We want to compare different regression models to examine which model performs better

- The most commonly used evaluation criteria are discussed in the next slides
  - Coefficient of Determination (R^2)
  - Adjusted R^2
  - Mean Absolute Error (MAE)
  - Mean Squared Error (MSE)
  - Root Mean Squared Error (RMSE)

Recap: Regression: Evaluation criterion - Coefficient of Determination (R^2)

- Coefficient of Determination (R^2)
  - This evaluation criterion is a key output of regression analysis, and it is interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variable.
  - It ranges from 0 to 1.
    - $R^2 = 0$:
      means that we cannot predict the dependent variable from the independent variable.
    - $R^2 = 1$:
      means we can predict the dependent variable without error from the independent variable.
- Interpretation:
  - It indicates the extent to which the independent variables explain the dependent variable.
- Example:
  - An R^2 of 0.25 means that 25 percent of the variance in *Y* is predictable from *X* (or explained by X).
- Whether this is good depends on the context!!
  - E.g., R^2=25% would be great to explain changes in the stock market (→ see 'Electronic Finance') while it would be low when aiming to explain differences in an employee's intention to quit their job.

Recap: Regression: Evaluation criterion - Coefficient of Determination (R^2)

- Coefficient of Determination (R^2)
  - Formula:
    - To keep it simple, we focus on the coefficient of determination (R^2) for a linear regression model with one independent variable

$$R^2 = \{ (1 / N) * \Sigma [(x_i - \bar{x}) * (y_i - \bar{y})] / (\sigma_x * \sigma_y) \}^2$$

    - with N as the number of observations used to fit the model,
    - Σ is the summation symbol,
    - x_i is the x value for observation i,
    - $\bar{x}$ is the mean x value, $\bar{x}$ $\bar{y}$
    - y_i is the y value for observation i,
    - $\bar{y}$ is the mean y value,
    - σ_x is the standard deviation of x,
    - σ_y is the standard deviation of y.

Recap: Regression: Evaluation criterion – Adjusted R^2

- Adjusted R^2
  - Contrary to R^2, the adjusted R^2 also indicates how well terms fit a curve or line but adjusts for the number of terms in a model.
    - So, when you add more useless variables to a model, the adjusted R^2 will decrease. (E.g., when you intend to explain the relationship between perceived usefulness and intention to use and you add size, age, and preferred color)
    - On the contrary, when you add more useful variables, the adjusted R^2 will increase. (E.g., when you intend to explain the relationship between perceived usefulness and intention to use and you add perceived ease of use or social influence)
  - However, the adjusted R^2 will always be less than or equal to R^2.
  - Formula:

$$R^2_{adj} = 1 - \left[\frac{(1-R^2)(n-1)}{n-k-1}\right]$$

    - With n as the number of points in a data sample
    - k as the number of independent regressors (variables in a model without constants)

- Why is adjusted R^2 required?
  - R^2 increases with every predictor/independent variable added to a model. So, with an increasing R^2, it can appear to be a better fit with the more predictors/independent variables one adds to the model.
  - Similarly, with too many predictors/independent variables in the model, one can run into the issue of over-fitting (meaning that the model is too complex for the data. d.g. due to too low sample size) data.

Recap: Regression: Evaluation criterion – Further criteria

- Mean Absolute Error (MAE)
  - Definition:
    - It reflects the is the mean of the absolute value of the errors.
  - Formula

$$\frac{1}{n}\sum_{i=1}^{n}|y_1 - \overline{y}_i|$$

- Mean Squared Error (MSE)
  - Definition:
    - It reflects the mean of the squared errors.
  - Formula

$$\frac{1}{n}\sum_{i=1}^{n}(y_1 - \overline{y}_i)^2$$

- Root Mean Squared Error (RMSE)
  - Definition:
    - It reflects the square root of the mean of the squared errors.
  - Formula

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_1 - \overline{y}_i)^2}$$

Recap: Regression: Example (Linear regression)

- Context:
  - Five students took a math aptitude test during the last Data Science course before beginning their course.

- Questions:
  1) What linear regression equation best shows the relationship between math aptitude scores and data science performance?
  2) How well does the regression equation fit the data?

- Starting point
  - x_i shows scores of the aptitude test
  - y_i shows data science grades

Student	x_i	y_i
1	95	85
2	85	95
3	80	70
4	70	65
5	60	70

Recap: Regression: Example (Linear regression)

- Context:
  - Five students took a math aptitude test during the last Data Science course before beginning their course.

- Finding the regression equation
  - We calculate sums and mean scores (see last two rows)
  - Based on that we calculate deviation scores (difference between a student‘s score and average score)
  - We also calculate the squares of the deviation scores and the product of both

Student	x_i	y_i	$(x_i-\bar{x})$	$(y_i-\bar{y})$	$(x_i-\bar{x})^2$	$(y_i-\bar{y})^2$	$(x_i-\bar{x})(y_i-\bar{y})$
1	95	85	17	8	289	64	136
2	85	95	7	18	49	324	126
3	80	70	2	-7	4	49	-14
4	70	65	-8	-12	64	144	96
5	60	70	-18	-7	324	49	126
Sum	390	385			730	630	470
Mean	78	77					

Recap: Regression: Example (Linear regression)

- Concerning question 1:
  - Considering the linear form of the regression $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ we need to solve for $\hat{\beta}_0$ and $\hat{\beta}_1$.
  - First, we solve for the regression coefficient ($\hat{\beta}_1$):
    - $\hat{\beta}_1 = \Sigma [(x_i - \bar{x})(y_i - \bar{y})] / \Sigma [(x_i - \bar{x})^2]$
    - $\hat{\beta}_1 = 470/730$
    - $\hat{\beta}_1 = 0.644$
  - Second, we solve for the regression slope ($\hat{\beta}_0$):
    - $\hat{\beta}_0 = \bar{y} - b_1 * \bar{x}$
    - $\hat{\beta}_0 = 77 - (0.644)(78)$
    - $\hat{\beta}_0 = 26.768$
  - Finally, we got $\hat{y} = 26.768 + 0.644x$ as our regression equation
  - Interpretation:
    - +26.768 is the constant
    - +0.644 is the coefficient
    - x is the predictor/independent variable
    - The coefficient value represents the mean change in the response given a one-unit change in the predictor.
      For example, with the coefficient of +0.644, the mean response value increases by 0.644 for every one unit change in the predictor/independent variable (a negative value implies that the response variable decreases if the predictor/independent variable increases).

Recap: Regression: Example (Linear regression)

- Concerning question 2:
  - How to Find the coefficient of determination.

- Throwback:
  - Whenever you use a regression equation, you should ask how well the equation fits the data. One way to assess fit is to check the coefficient of determination, which can be computed from the following formula:

$$R^2 = \{ (1 / N) * \Sigma [(x_i - \bar{x}) * (y_i - \bar{y})] / (\sigma_x * \sigma_y) \}^2$$

    - with N as the number of observations used to fit the model,
    - Σ is the summation symbol,
    - x_i is the x value for observation i,
    - $\bar{x}$ is the mean x value,
    - y_i is the y value for observation i,
    - $\bar{y}$ is the mean y value,
    - σ_x is the standard deviation of x,
    - σ_y is the standard deviation of y.

Recap: Regression: Example (Linear regression)

- Concerning question 2:
  - How to Find the coefficient of determination.
  - We begin by computing the standard deviation of x (σ_x):

$\sigma_x = \text{sqrt} [\Sigma (x_i - \bar{x})^2 / N]$
$\sigma_x = \text{sqrt}(730/5) = \text{sqrt}(146) = 12.083$

  - Next, we find the standard deviation of y, (σ_y):

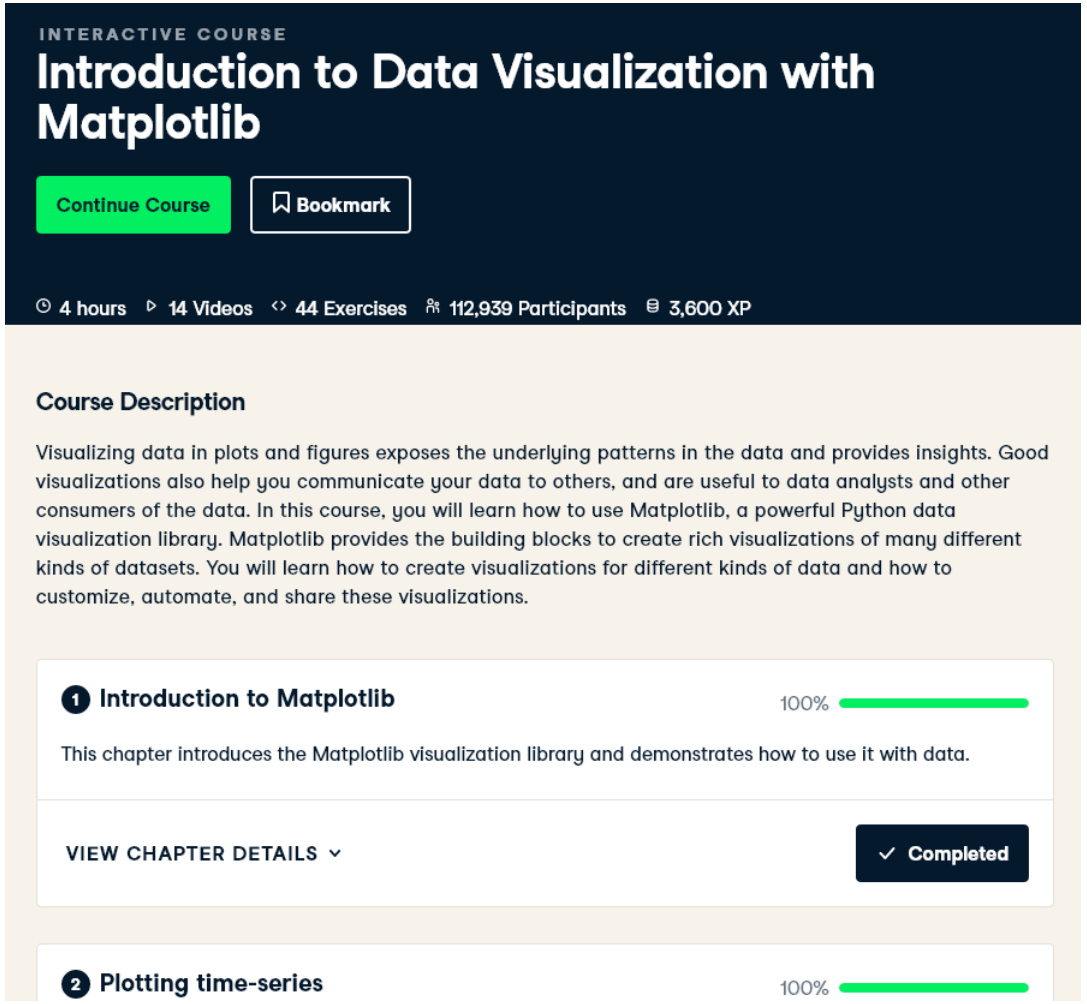
$\sigma_y = \text{sqrt} [\Sigma (y_i - \bar{y})^2 / N]$
$\sigma_y = \text{sqrt}(630/5) = \text{sqrt}(126) = 11.225$

  - Finally, we compute the coefficient of determination (R^2): $R^2 = \{ (1 / N) * \Sigma [(x_i - \bar{x}) * (y_i - \bar{y})] / (\sigma_x * \sigma_y) \}^2$

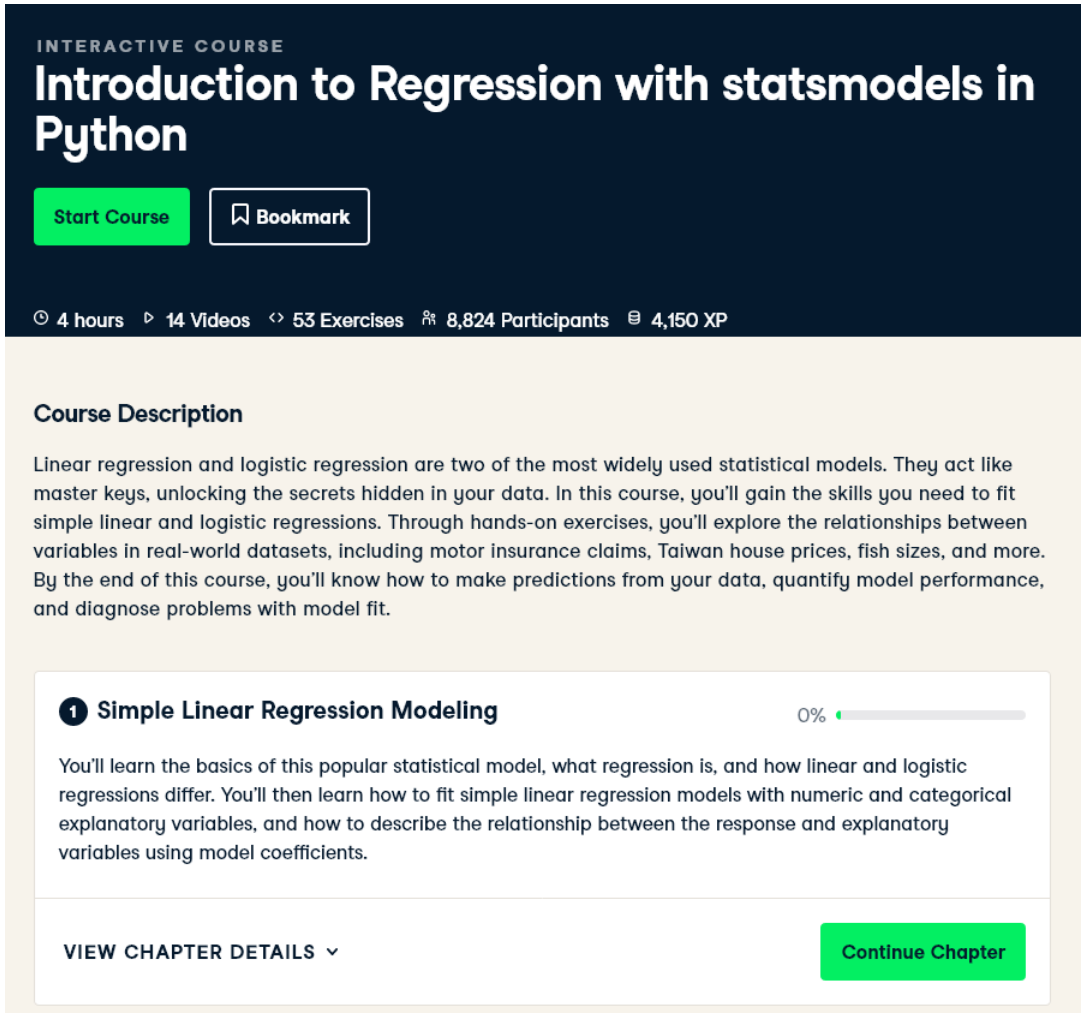
$R^2 = [(1/5) * 470 / (12.083 * 11.225)]^2$
$R^2 = (94 / 135.632)^2 = (0.693)^2 = 0.48$

- Interpretation:
  - A coefficient of determination equal to 0.48 indicates that:
    - The math aptitude scores explain 48% of the variation in data science grades.

Recap: Datacamp



https://www.datacamp.com/courses/introduction-to-data-visualization-with-matplotlib



https://www.datacamp.com/courses/introduction-to-regression-with-statsmodels-in-python

Regression for EDA vs. Regression for Machine Learning

- Goal:
  - Regression in exploratory data analysis is used to explain the dependent variable
  - Regression in machine learning is used to predict the dependent variable

- Terminology:
  - The regression constant $\hat{\beta}_0$, which is equal to the y-intercept of the regression line, is often called the bias term θ_0 in machine learning
  - The regression coefficient ($\hat{\beta}_i$), which is the average change in the dependent variable (Y) for a 1-unit change in the independent variable (X_i), is often called the feature weights θ_i in machine learning
  - The evaluation criterion (e.g., MSE) often refers to the cost function in machine learning

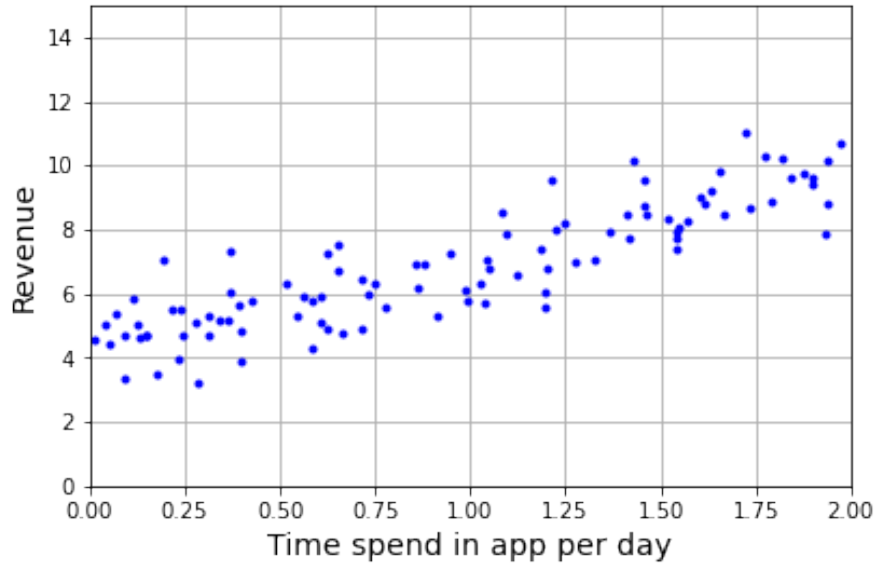
- Approach
  - When using regression models for (supervised) machine learning the data set is, just as in classification, split into training and test data
    - Training data is the set of the data on which the actual training takes place.
    - The test data is the set that informs us about the final accuracy
  - When using regression models for exploratory data analysis, one does not split the data into different data sets, but one puts more focus on analytic validity (e.g., multicollinearity)

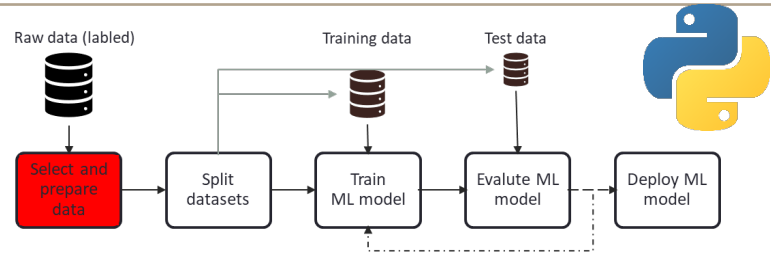
- Python
  - The statsmodels Python library has specialized in regression models for exploratory data analysis
  - The sklearn Python library has specialized in regression models for machine learning

Example

- We use the following “mobile app dataset” for regression

- Context:
  - The revenue of a mobile app user is influenced by many factors, including the time a user spends in the app per day
  - As a data scientist your job is to use supervised machine learning to predict the revenue of mobile app user



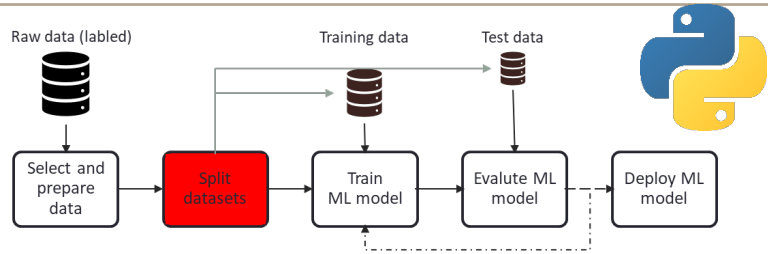


Linear regression

Select and prepare data

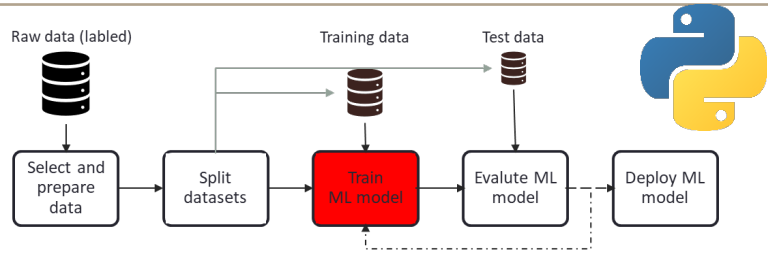
```
# Import libraries
import pandas as pd

# read the dataset
data = pd.read_csv("https://raw.githubusercontent.com/VAWi-DataScience/Data-Science-and-Machine-Learning/main/Lecture/Data/03_Regression_data1.csv",sep=',')

# Show top 5-records
data.head()
```

```
# Splitting dataset in two parts: feature set and target label
X = data.X.array.reshape(-1, 1)
y = data.y.array.reshape(-1, 1)
```



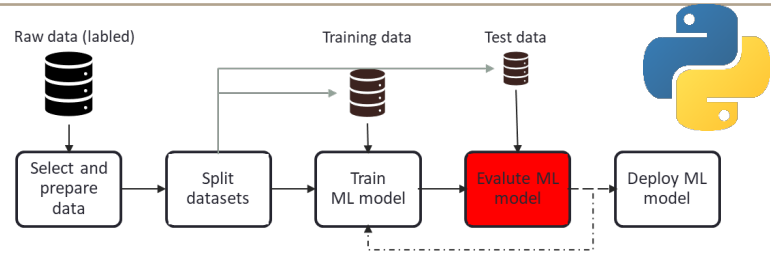
Split dataset

```
#Import train_test_split function
from sklearn.model_selection import train_test_split

## Split X and y into training and testing sets
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



Train ML model

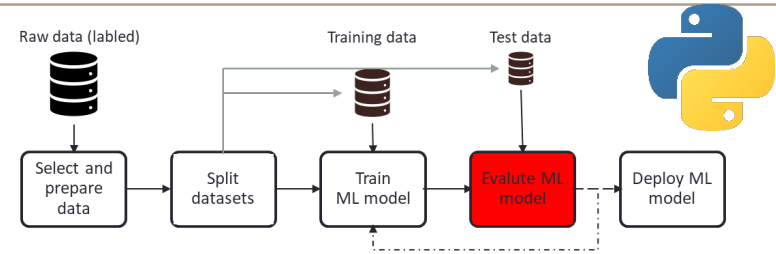
```
# Import linear regression model
from sklearn.tree import LinearRegression

# Create a linear regression model
lin_reg = LinearRegression()

# Train the linear model using training dataset
lin_reg.fit(X_train, y_train)
```



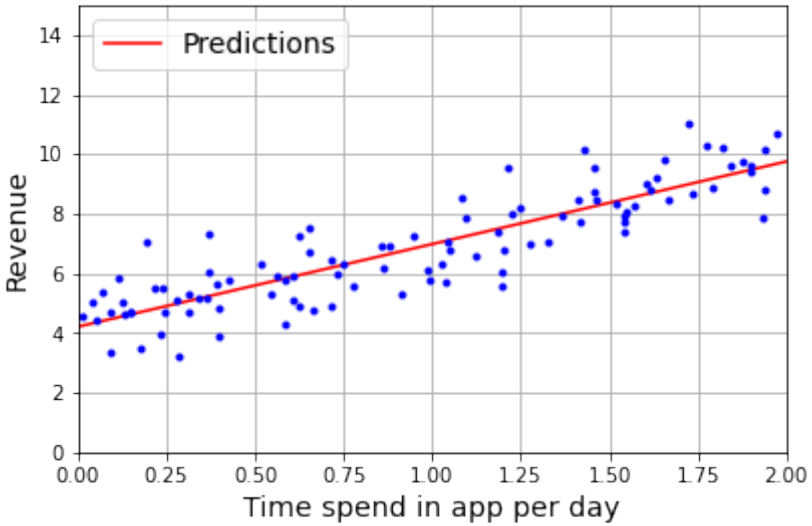
Evaluate (training data)

```
predictions_train = lin_reg.predict(X_train)
# Import the required libraries
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Evaluate mean absolute error
print('Mean Absolute Error(MAE):', mean_absolute_error(y_train,predictions_train))

# Evaluate mean squared error
print("Mean Squared Error(MSE):", mean_squared_error(y_train, predictions_train))

# Evaluate root mean squared error
print("Root Mean Squared Error(RMSE):", np.sqrt(mean_squared_error(y_train, predictions_train)))

# Evaluate R2-square
print("R2-Square:",r2_score(y_train, predictions_train))
```

```
Mean Absolute Error(MAE): 0.7347297926919522
Mean Squared Error(MSE): 0.8476788564209705
Root Mean Squared Error(RMSE): 0.9206947683249702
R2-Square: 0.7582381034538057
```



Visualize regression line



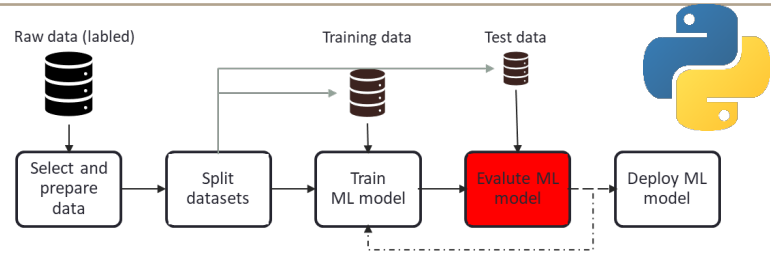
Intercept and coefficients

```
lin_reg.intercept_, lin_reg.coef_
```

```
(array([4.14291332]), array([[2.79932366]]))
```

The following model is estimated:

$\hat{y} = 4{,}14291332 + 2{,}79932366 * X$



Evaluate (test data)

```
#Predict the given test set
predictions_test = lin_reg.predict(X_test)
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Evaluate mean absolute error
print('Mean Absolute Error(MAE):', mean_absolute_error(y_test,predictions_test))

# Evaluate mean squared error
print("Mean Squared Error(MSE):", mean_squared_error(y_test, predictions_test))

# Evaluate root mean squared error
print("Root Mean Squared Error(RMSE):", np.sqrt(mean_squared_error(y_test, predictions_test)))

# Evaluate R2-square
print("R2-Square:",r2_score(y_test, predictions_test))
```

```
Mean Absolute Error(MAE): 0.5913425779189778
Mean Squared Error(MSE): 0.6536995137170021
Root Mean Squared Error(RMSE): 0.8085168605026132
R2-Square: 0.8072059636181391
```

Regression

- Linear Regression with Normal Equation
- **Linear Regression with (Stochastic) Gradient Descent**
- Polynomial Regression
- Ridge Regression
- Lasso Regression
- Elastic Net and Comparison
- Early Stopping
- Regression Trees

Gradient Descent

- What we have done so far:
  - We have trained the regression model by using the normal equation.
    - Training the regression model means that we have calculated the “optimal” parameters for the regression model
  - Optimizing regression with normal equation gets slow when we have a large number of features
  - There exist other algorithms to optimize regression models, which we now take a closer look at

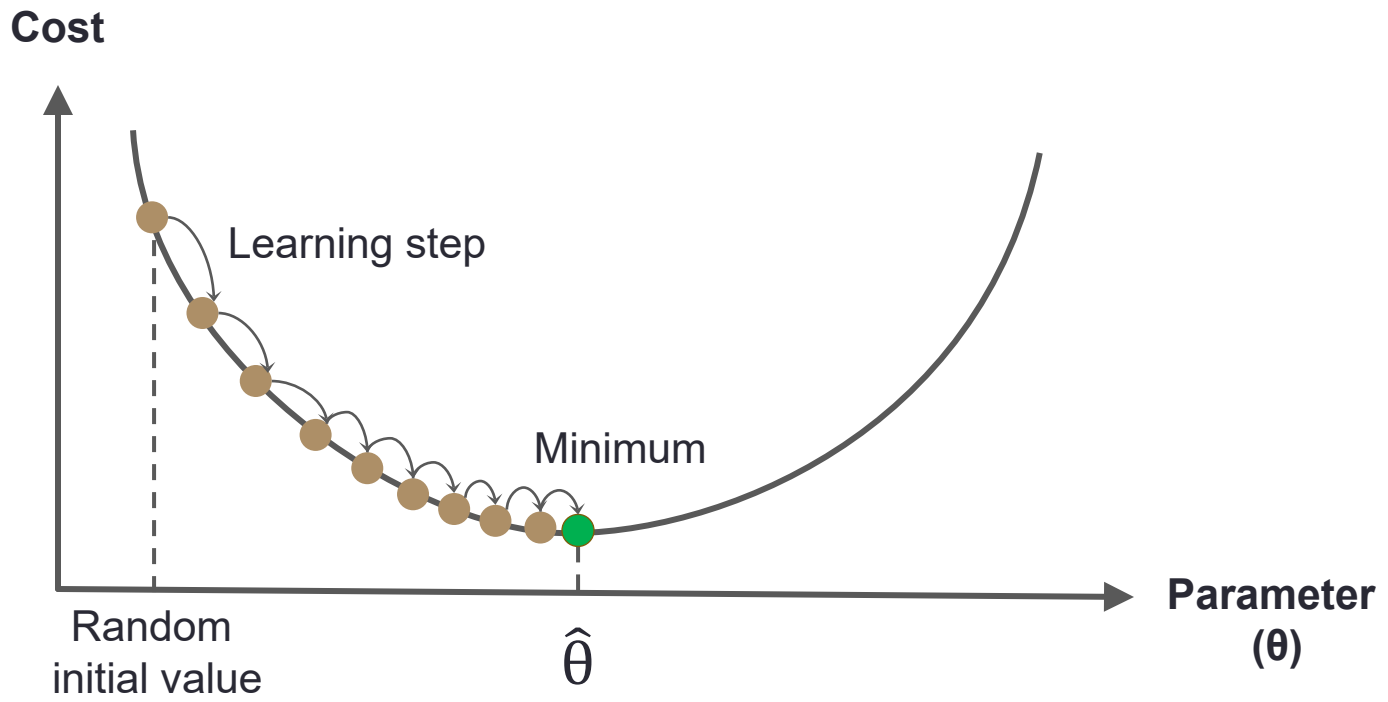
- Gradient Descent
  - Now we will look at a very different way to train a linear regression model, which is better suited for cases where there are a large number of features or too many training instances to fit in the computer memory.
  - Gradient Descent is an optimization algorithm capable of finding optimal solutions to a wide range of problems, such as optimizing neural networks for Deep Learning.

≪ Regression models can be trained and optimized with different methods, such as the normal equation or gradient descent.≫

≪ Gradient descent is used for various machine learning tasks, such as in Deep Learning, to find the optimal parameters.≫

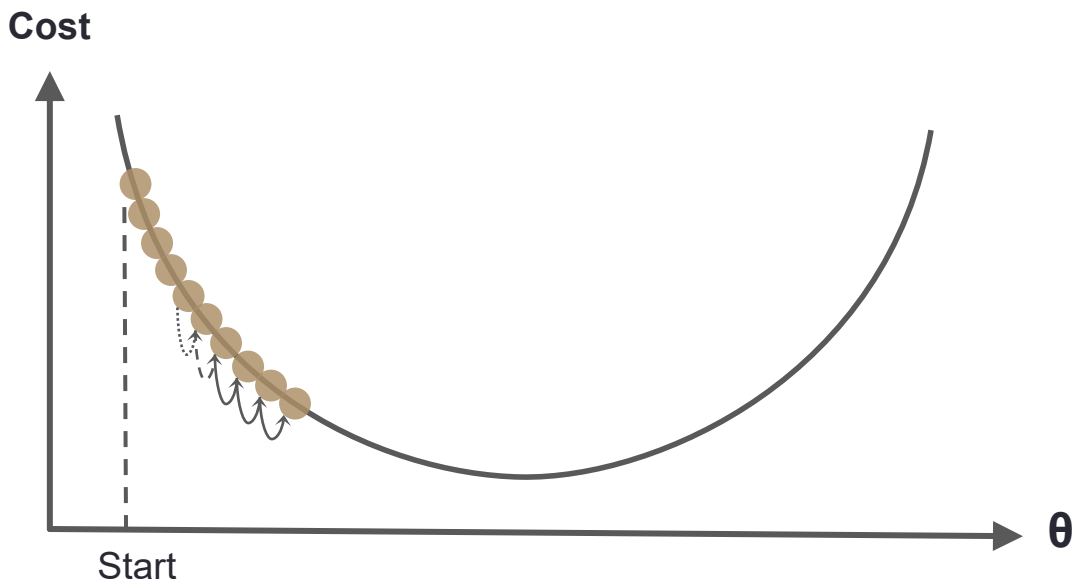
Gradient Descent

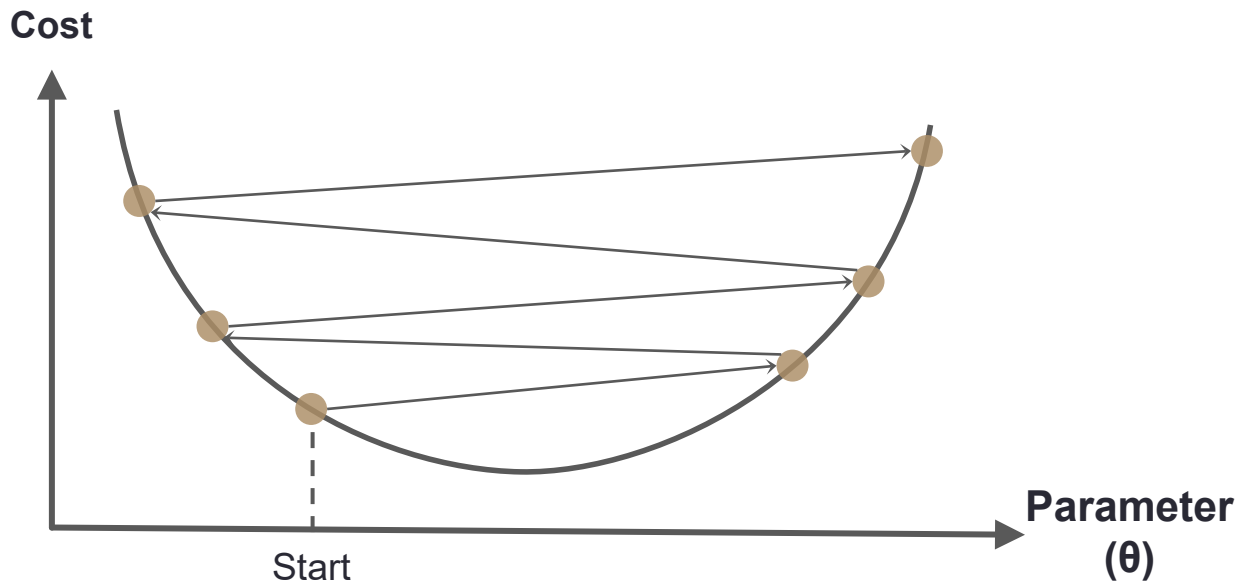
- The general idea of Gradient Descent is to tweak parameters (θ) iteratively to minimize a cost function.
  - For simple regression models,
    - Parameters (θ) are the intercept or the regression coefficient, which are often initialized randomly
    - Cost function could be MSE
  - The Gradient Descent measures the local gradient of the error function about the parameter vector θ, and it goes in the direction of descending gradient.
    - Once the gradient is zero, you have reached a minimum.
    - Gradient is just a fancy word for slope or steepness.



Gradient Descent: Learning rate

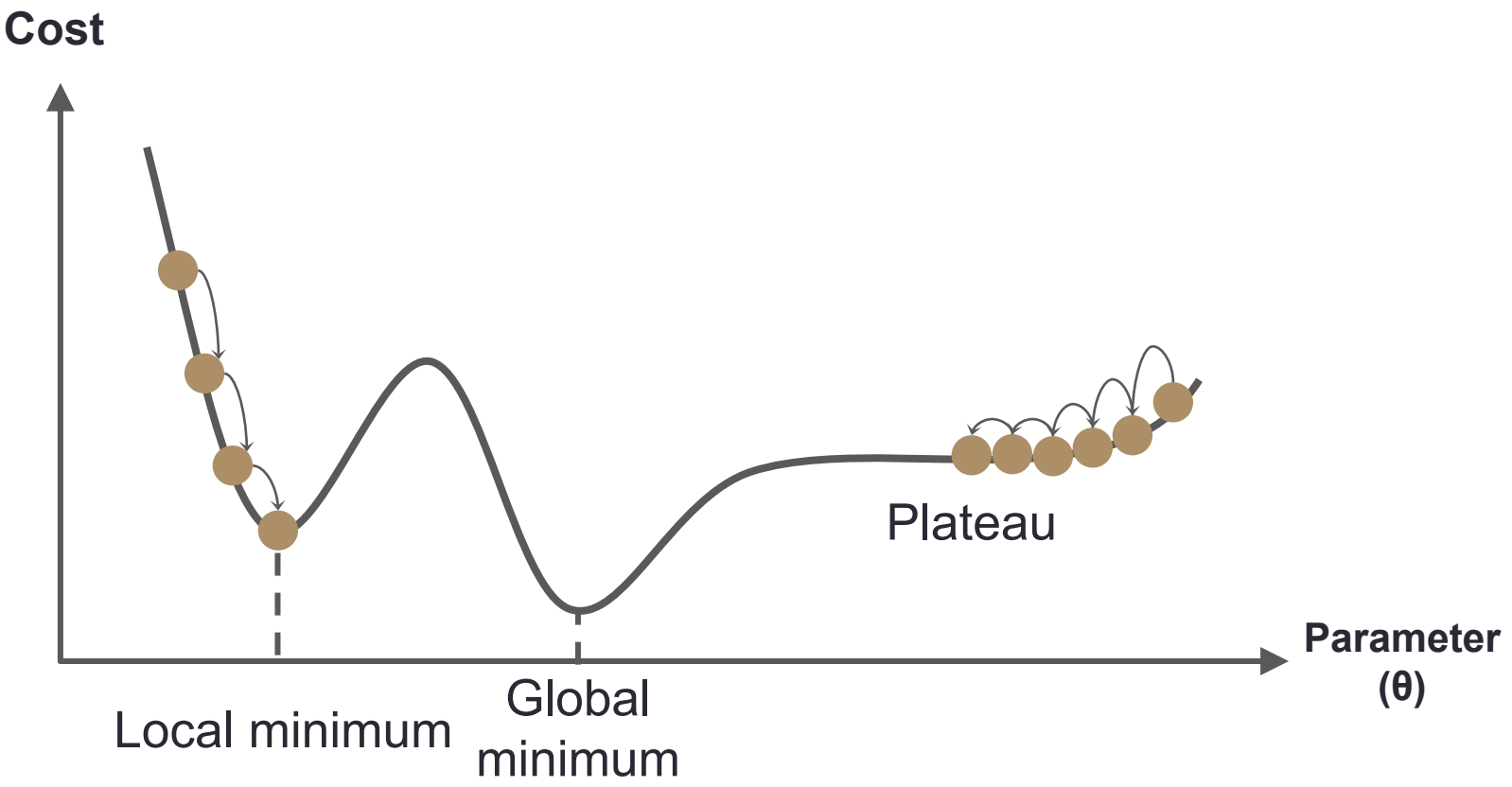
- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter.
  - If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time (see left Figure).
  - On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge, with larger and larger values, failing to find a good solution (see right Figure).





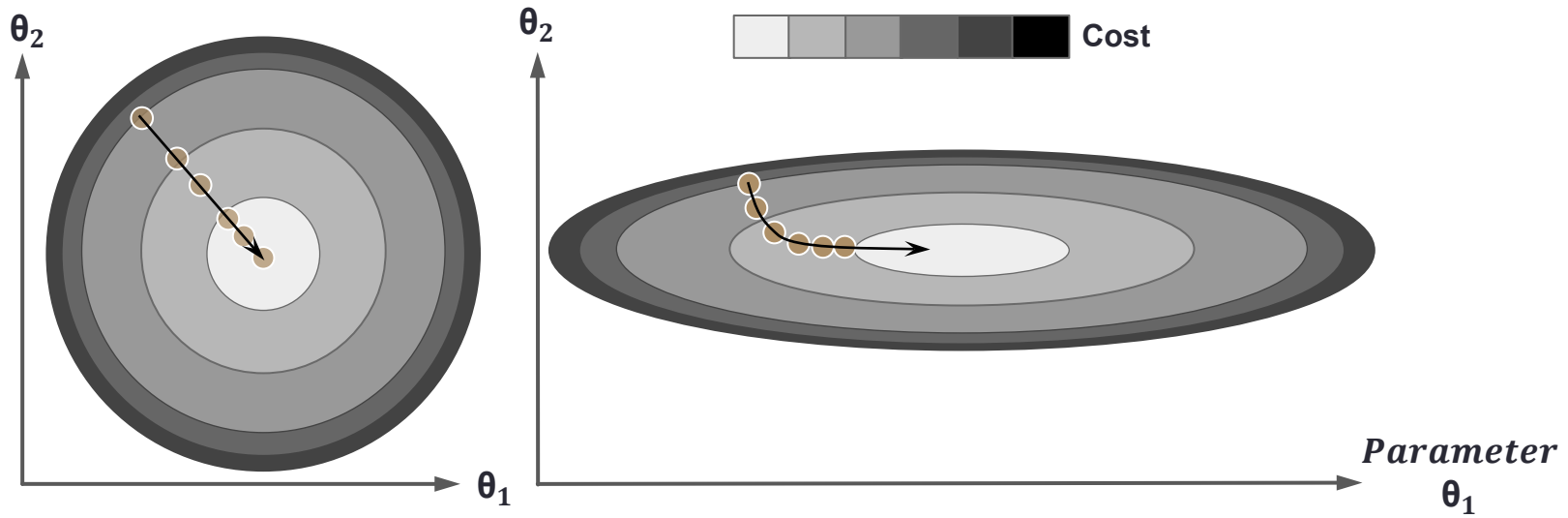
Challenges of Gradient Descent

- Finally, not all cost functions look like nice, regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum difficult.
  - If the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum.
  - If it starts on the right, then it will take a very long time to cross the plateau. And if you stop too early, you will never reach the global minimum.



Gradient Descent

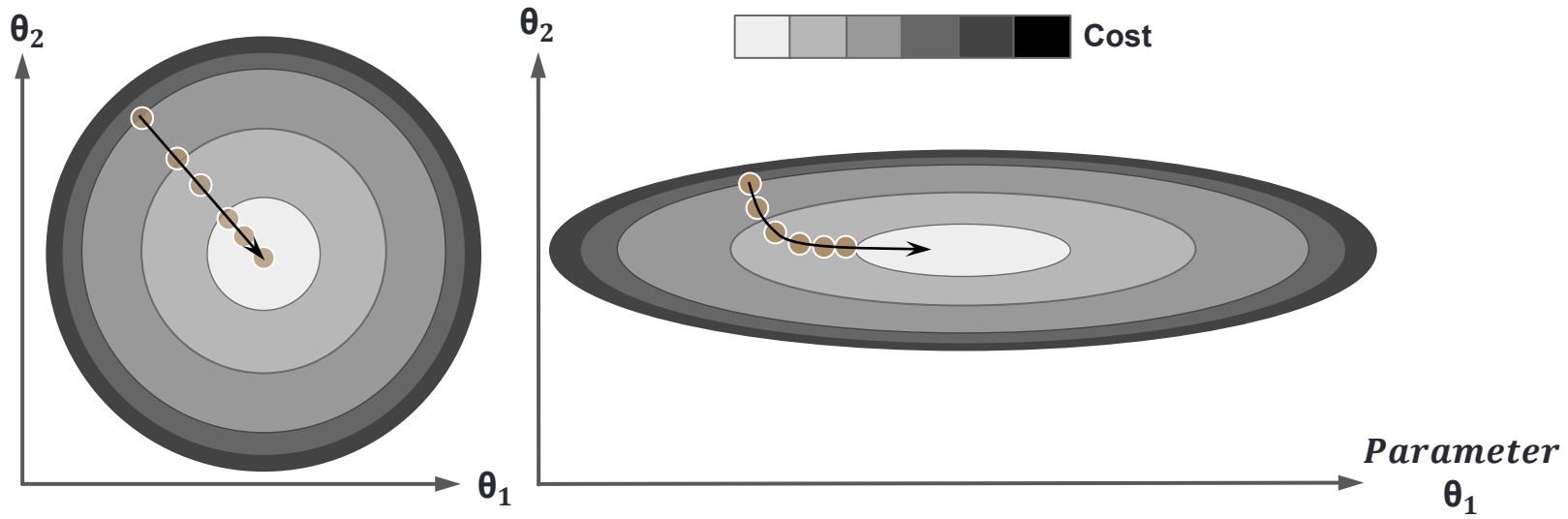
- Fortunately, the cost function for a Linear Regression model happens to be a convex function, which means that if you pick any two points on the curve, the line segment joining them never crosses the curve.
  - This implies that there are no local minima, just one global minimum. It is also a continuous function with a slope that never changes abruptly.
  - These two facts have a great consequence: Gradient Descent is guaranteed to approach arbitrarily close to the global minimum (if you wait long enough and if the learning rate is not too high)
  - In fact, the cost function has the shape of a bowl, but it can be an elongated bowl if the features have very different scales.
  - The Figure shows Gradient Descent on a training set where features 1 and 2 have the same scale (on the left), and on a training set where feature 1 has much smaller values than feature 2 (on the right).
  - When using Gradient Descent, you should ensure that all features have a similar scale, or else it will take much longer to converge.



≪ When using Gradient Descent, features should be scaled. ≫

Gradient Descent

- Training a regression model with Gradient Descent means that it all parameters need to be considered
  - The more parameters a regression model has the harder the search for the best parameters is
  - Each additional feature in a regression model, leads to more parameters that need to be considered
  - For reasons of simplicity, we demonstrate the concept of gradient descent with only two features

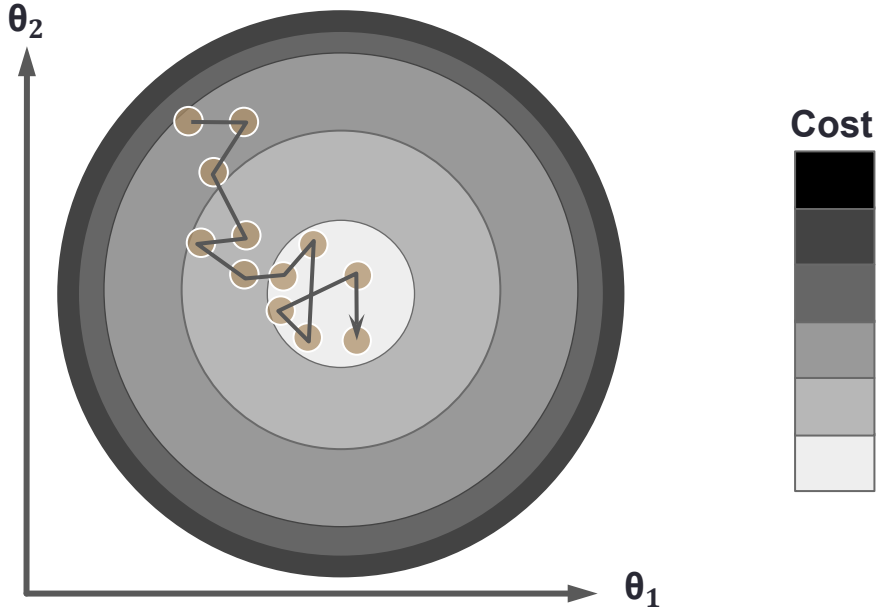


Gradient Descent: Limitation

- The limitation of gradient descent
  - The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.
  - An alternative approach is “stochastic gradient descent”, which we cover next

Stochastic Gradient Descent

- Stochastic Gradient Descent picks a random instance in the training set at every step and computes the gradients based only on that single instance.
- Advantages:
  - Working on a single instance at a time makes the algorithm much faster because it has very little data to manipulate at every iteration.
  - It also makes it possible to train on huge training sets, since only one instance needs to be in memory at each iteration.
- Disadvantages:
  - Due to its stochastic (i.e., random) nature, this algorithm is much less regular.
  - Instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average.
  - Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down (see Figure). So once the algorithm stops, the final parameter values are good, but not optimal.



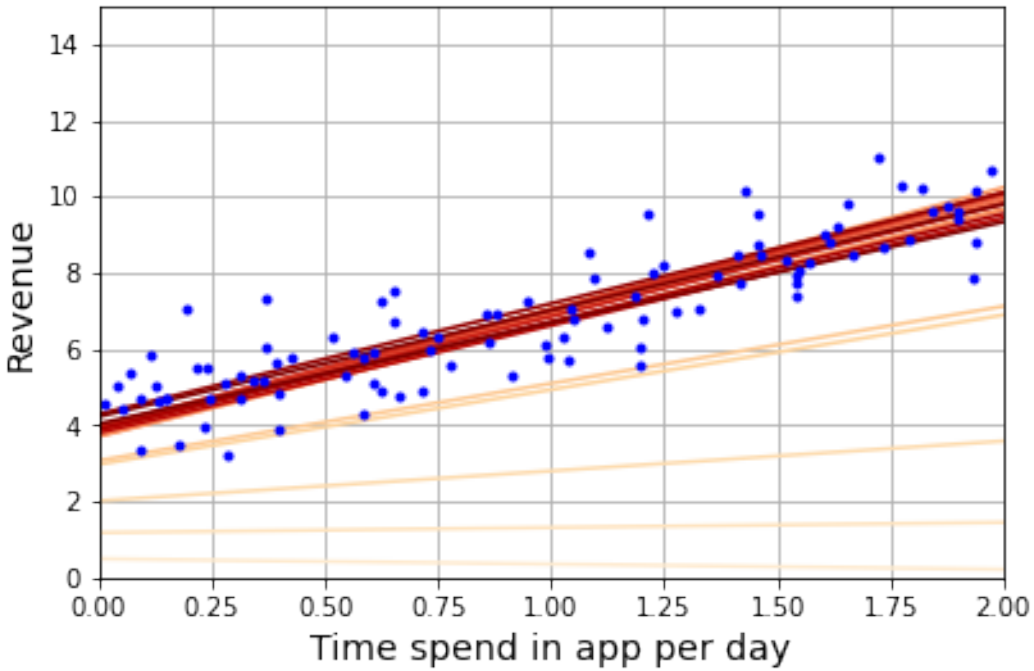
Stochastic Gradient Descent

- When the cost function has several local minima, the stochastic nature can help the algorithm jump out of local minima, so Stochastic Gradient Descent has a better chance of finding the global minimum.
  - Randomness is good to escape from local optima
  - Randomness is bad because it means that the algorithm can never settle at the minimum

- One solution to this dilemma is to gradually reduce the learning rate.
  - The steps start out large (which helps make quick progress and escape local minima), then get smaller and smaller, allowing the algorithm to settle at the global minimum.
  - The function that determines the learning rate at each iteration is called the learning schedule.
    - If the learning rate is reduced too quickly, you may get stuck in a local minimum, or even end up frozen halfway to the minimum.
    - If the learning rate is reduced too slowly, you may jump around the minimum for a long time and end up with a suboptimal solution if you halt training too early.

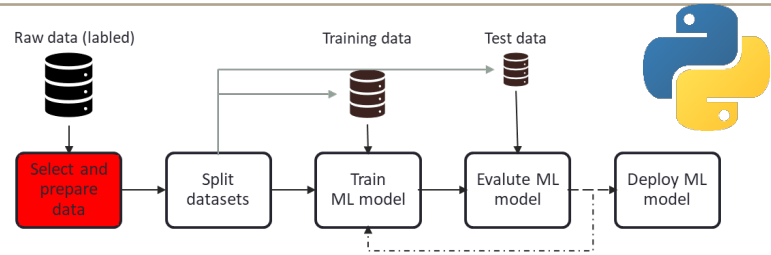
Stochastic Gradient Descent

- Example of the learning rate for stochastic gradient descent
  - By convention we iterate by rounds of m iterations; each round is called an epoch
  - Here we use 50 epochs
  - The figure shows the first 20 epochs



Stochastic Gradient Descent

- When using Stochastic Gradient Descent, the training instances must be independent and identically distributed to ensure that the parameters get pulled toward the global optimum, on average.
  - A simple way to ensure this is to shuffle the instances during training (e.g., pick each instance randomly, or shuffle the training set at the beginning of each epoch).
  - If you do not shuffle the instances - for example, if the instances are sorted by the label - then SGD will start by optimizing for one label, then the next, and so on, and it will not settle close to the global minimum.



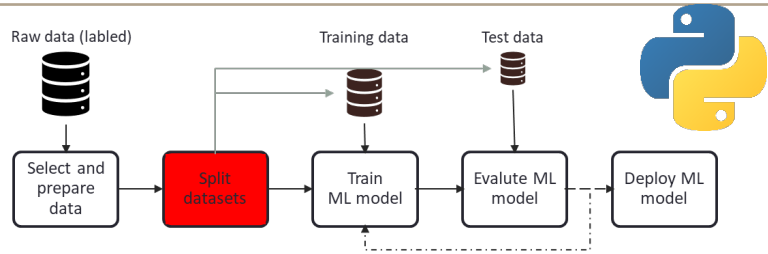
Linear regression with SGD

Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
data = pd.read_csv("https://raw.githubusercontent.com/VAWi-DataScience/Data-Science-and-Machine-Learning/main/Lecture/Data/03_Regression_data1.csv",sep=',')

# Show top 5-records
data.head()
```

```
# Splitting dataset in two parts: feature set and target label
X = data.X.array.reshape(-1, 1)
y = data.y.array.reshape(-1, 1)
```

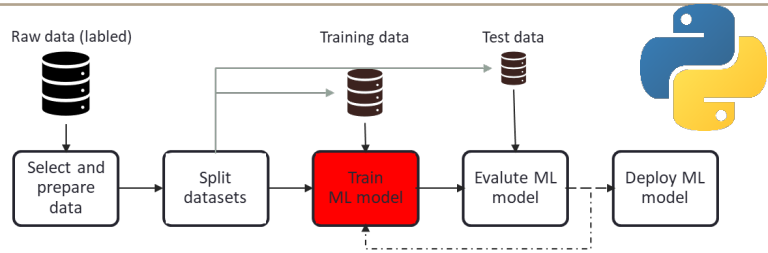
Linear regression with SGD



Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Linear regression with SGD



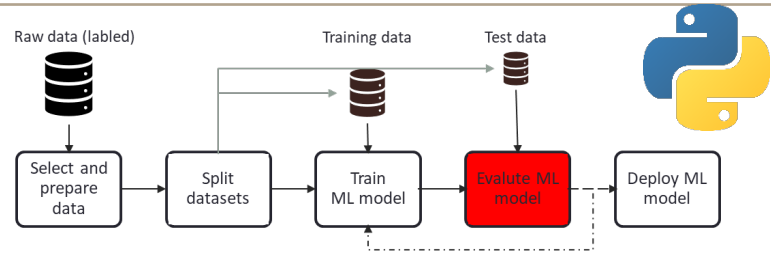
Train ML model

```
# Import linear regression model
from sklearn.linear_model import SGDRegressor

# Create linear regression model
sgd_reg = SGDRegressor(max_iter=1000, tol=1e-5, penalty=None, eta0=0.01,
                       n_iter_no_change=100, random_state=42)

# Fit the linear regression model
sgd_reg.fit(X_train, y_train.ravel())# y.ravel() because fit() expects 1D targets
```

Note:

- The SGDRegressor class, uses the MSE cost function by default
- The SGD runs for 1,000 epochs (max_iter) or until the loss drops by less than 10–5 (tol) during 100 epochs (n_iter_no_change).
- It starts with a learning rate of 0.01 (eta0), using the default learning schedule
- It does not use any regularization (penalty=None), which we discuss later in this chapter

Linear regression with SGD



Evaluate (training data)

```
predictions_train_sgd_reg = sgd_reg.predict(X_train)
# Import the required libraries
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Evaluate mean absolute error
print('Mean Absolute Error(MAE):', mean_absolute_error(y_train,predictions_train_sgd_reg))

# Evaluate mean squared error
print("Mean Squared Error(MSE):", mean_squared_error(y_train, predictions_train_sgd_reg))

# Evaluate root mean squared error
print("Root Mean Squared Error(RMSE):", np.sqrt(mean_squared_error(y_train,
predictions_train_sgd_reg)))

# Evaluate R2-square
print("R2-Square:",r2_score(y_train, predictions_train_sgd_reg))
```

```
Mean Absolute Error(MAE): 0.7348245459810183
Mean Squared Error(MSE): 0.8476797356384964
Root Mean Squared Error(RMSE): 0.9206952457998773
R2-Square: 0.758237852696936
```

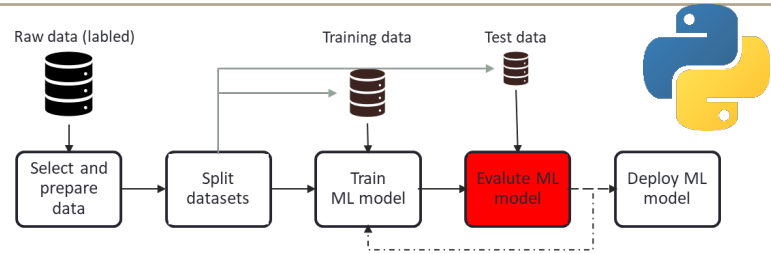
```
sgd_reg.intercept_, sgd_reg.coef_
```

```
(array([4.14119539]), array([2.80043269]))
```

The following model is estimated:

$\hat{y} = 4{,}14119539 + 2{,}80043269 * X$

Linear regression with SGD



Evaluate (test data)

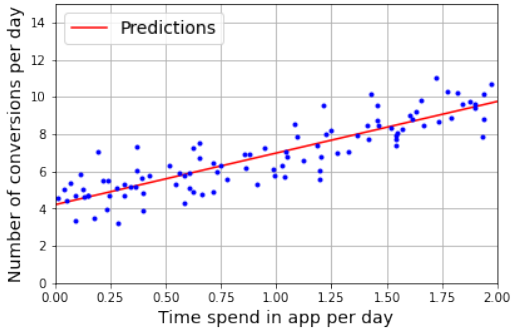
```
predictions_test_sgd_reg = sgd_reg.predict(X_test)
# Import the required libraries
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Evaluate mean absolute error
print('Mean Absolute Error(MAE):', mean_absolute_error(y_test,predictions_test_sgd_reg))

# Evaluate mean squared error
print("Mean Squared Error(MSE):", mean_squared_error(y_test, predictions_test_sgd_reg))

# Evaluate root mean squared error
print("Root Mean Squared Error(RMSE):", np.sqrt(mean_squared_error(y_test,
predictions_test_sgd_reg)))

# Evaluate R2-square
print("R2-Square:",r2_score(y_test, predictions_test_sgd_reg))
```

```
Mean Absolute Error(MAE): 0.5916924455104067
Mean Squared Error(MSE): 0.6541158038528608
Root Mean Squared Error(RMSE): 0.8087742601324927
R2-Square: 0.8070831881625757
```

Comparision: Normal Equation VS. Stochastic Gradient Descent

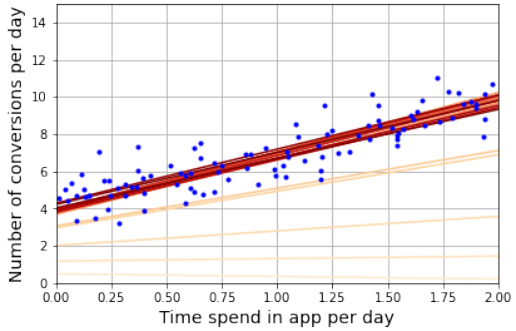
- Comparison of linear regression algorithms
  - There is almost no difference after training, these algorithms provide similar regression model
  - Only little differences exist in the parameters

Normal Equation



$\hat{y} = 4{,}14291332 + 2{,}79932366 * X$

Stochastic GD



$\hat{y} = 4{,}14119539 + 2{,}80043269 * X$

Algorithm	Large number of training instances	Out-of-core support	Large number of features	Hyperparameter	Scaling required
Normal Equation	Fast	No	Slow	0	No
Stochastic GD	Fast	Yes	Fast	≥2	Yes

Regression

- Linear Regression with Normal Equation
- Linear Regression with (Stochastic) Gradient Descent
- **Polynomial Regression**
- Ridge Regression
- Lasso Regression
- Elastic Net and Comparison
- Early Stopping
- Regression Trees

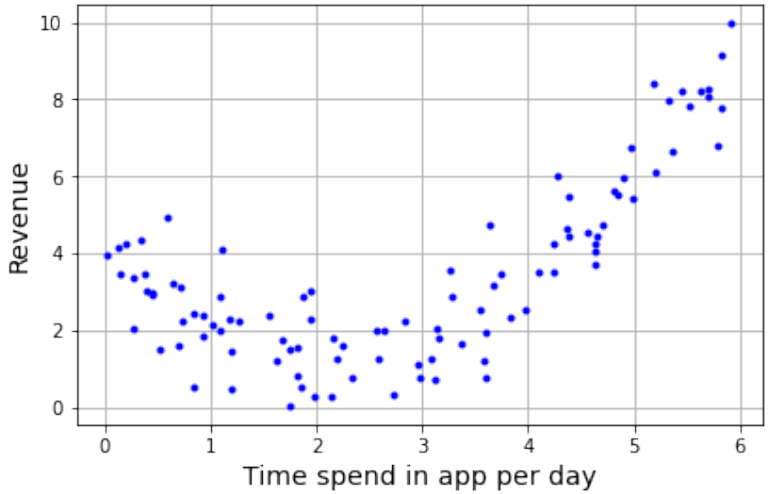
Polynomial Regression

- Most data is more complex than a straight line, e.g, its nonlinear
  - A simple linear regression model will not fill the data well
  - We need to enhance the regression model to better fit the data
  - A simple way to do this is to add powers of each feature as new features, then train a linear model on this extended set of features.
  - This technique is called polynomial regression.

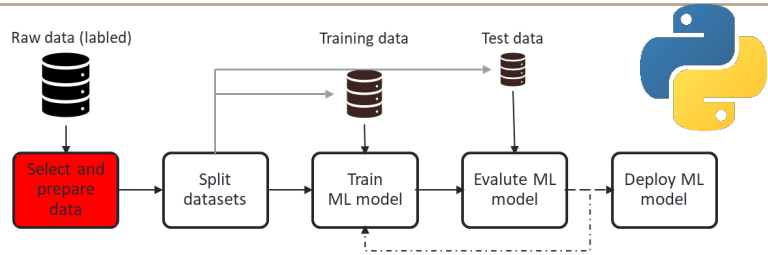
Example

- We use the following new “mobile app dataset”, which differs from the previous one, for polynomial regression

- Context:
  - The revenue of a mobile app user is influenced by many factors, including the time a user spends in the app per day
  - As a data scientist your job is to use supervised machine learning to predict the revenue of mobile app user



Polynomial Regression

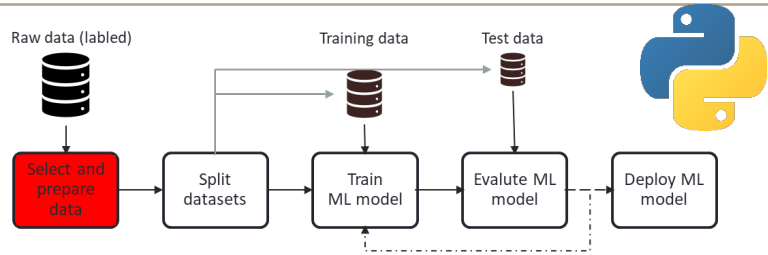


Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
df = pd.read_csv("https://raw.githubusercontent.com/VAWi-DataScience/Data-Science-and-Machine-Learning/main/Lecture/Data/03_Regression_data2.csv",sep=',')
df.head()
```

	Time	Revenue
0	2.247241	1.617611
1	5.704286	8.061859
2	4.391964	4.452506
3	3.591951	0.779585
4	0.936112	1.846257

Polynomial Regression



- Clearly, a straight line will never fit this data properly.
  - So Scikit-Learn's "PolynomialFeatures" class is used to transform the training data, adding the square (second-degree polynomial) of each feature in the training set as a new feature (in this case there is just one feature).

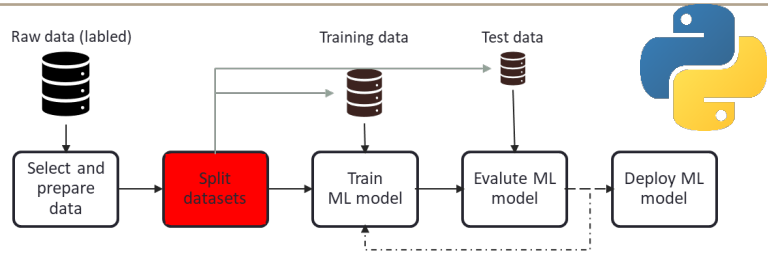
```
from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X) # adding the square
```

  - X_poly now contains the original feature of X plus the square of this feature.

```
X_poly[0:4] #show top 4 rows
```

```
array([[2.24724071e+00, 5.05009082e+00],
       [5.70428584e+00, 3.25388769e+01],
       [4.39196365e+00, 1.92893447e+01],
       [3.59195091e+00, 1.29021113e+01]])
```

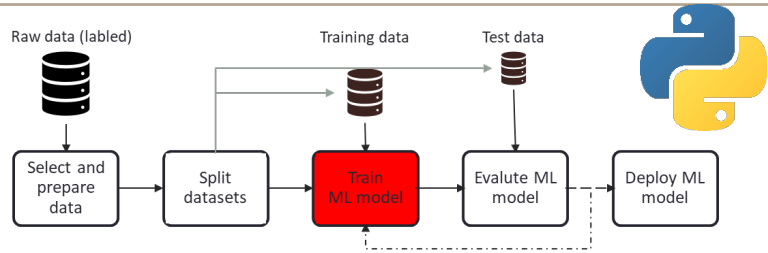
Polynomial Regression



Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split

# Partition data into training and testing set
X_train,X_test, y_train, y_test = train_test_split(X_poly, df.Revenue, test_size=0.2,
random_state=42)
```

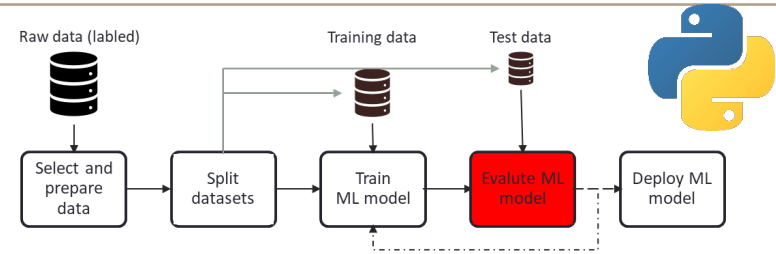
Polynomial Regression



Train ML model

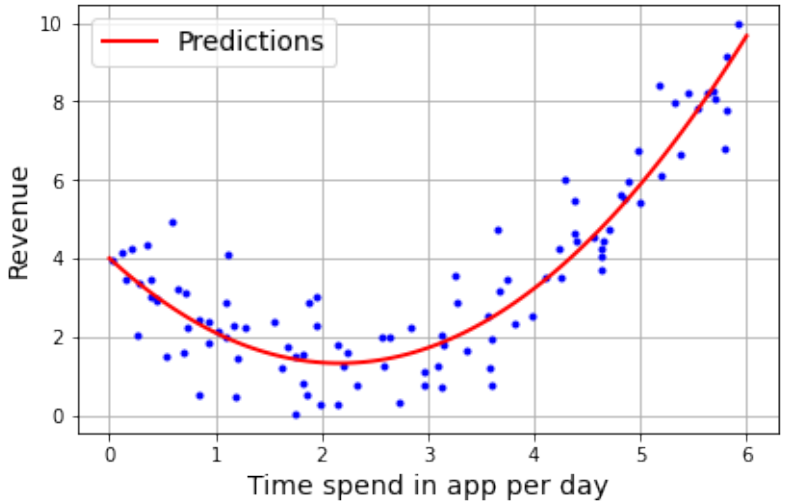
```
# With the new data a regression model can be used to fit this extended training data.

from sklearn.tree import LinearRegression
lin_reg_poly = LinearRegression()
lin_reg_poly.fit(X_train, y_train)
```

Polynomial Regression



Evaluate (training data)

```
predictions_train_lin_reg_poly = lin_reg_poly.predict(X_train)
# Import the required libraries
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Evaluate mean absolute error
print('Mean Absolute Error(MAE):', mean_absolute_error(y_train,predictions_train_lin_reg_poly))

# Evaluate mean squared error
print("Mean Squared Error(MSE):", mean_squared_error(y_train, predictions_train_lin_reg_poly))

# Evaluate root mean squared error
print("Root Mean Squared Error(RMSE):", np.sqrt(mean_squared_error(y_train,
predictions_train_lin_reg_poly)))

# Evaluate R2-square
print("R2-Square:",r2_score(y_train, predictions_train_lin_reg_poly))
```




```
Mean Absolute Error(MAE): 0.7034466665093438
Mean Squared Error(MSE): 0.8147153703416314
Root Mean Squared Error(RMSE): 0.9026158487095335
R2-Square: 0.8497157422674118
```

The model estimates the following regression:

$\hat{y}$ = 4.013066787886217 + X^2*-2.4607229* X * 0.56726223

```
lin_reg_poly.intercept_, lin_reg_poly.coef_
```

```
(4.013066787886217, array([-2.4607229 ,  0.56726223]))
```

Polynomial Regression

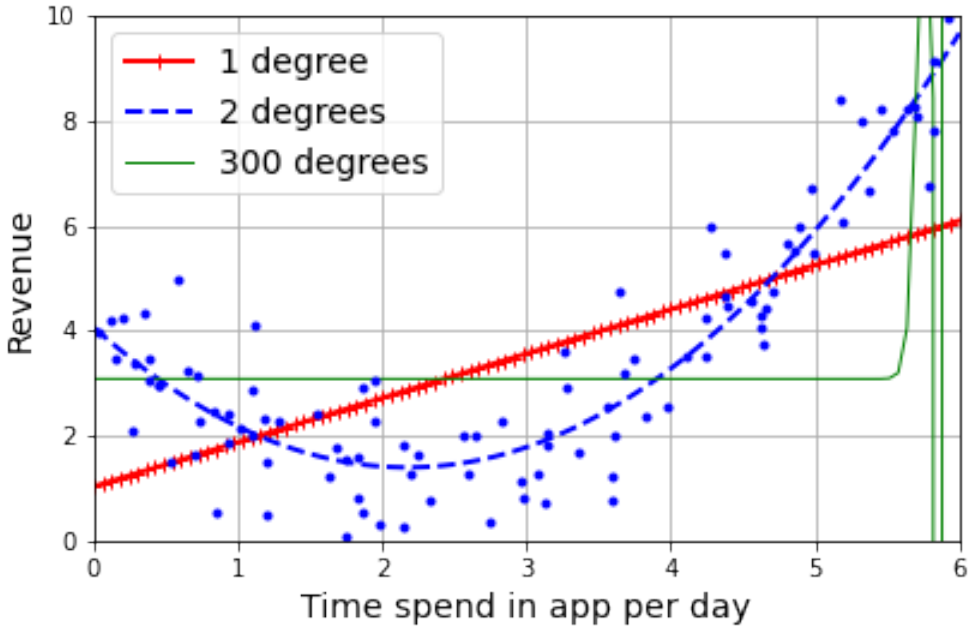
- Note: that when there are multiple features, Polynomial Regression is capable of finding relationships between features (which is something a plain Linear Regression model cannot do).
  - This is made possible by the fact that “PolynomialFeatures” also adds all combinations of features up to the given degree.
  - For example, if there were two features a and b, “PolynomialFeatures” with “degree=3” would not only add the features a^2, a^3 , b^2, and b^3, but also the combinations ab, a^2 b, and ab^2.

Polynomial Regression: Comparison

- If a high degree polynomial regression is performed, the training data will likely fit much better than with a simple linear regression.
  - For example, Figure applies a 300-degree polynomial model to the preceding training data and compares the result with a pure linear model and a quadratic model (second-degree polynomial).

- This high-degree (300-degree) Polynomial Regression model is severely overfitting the training data, while the linear model is underfitting it.

- Next, we examine a method to evaluate if a regression model is overfitting or underfitting

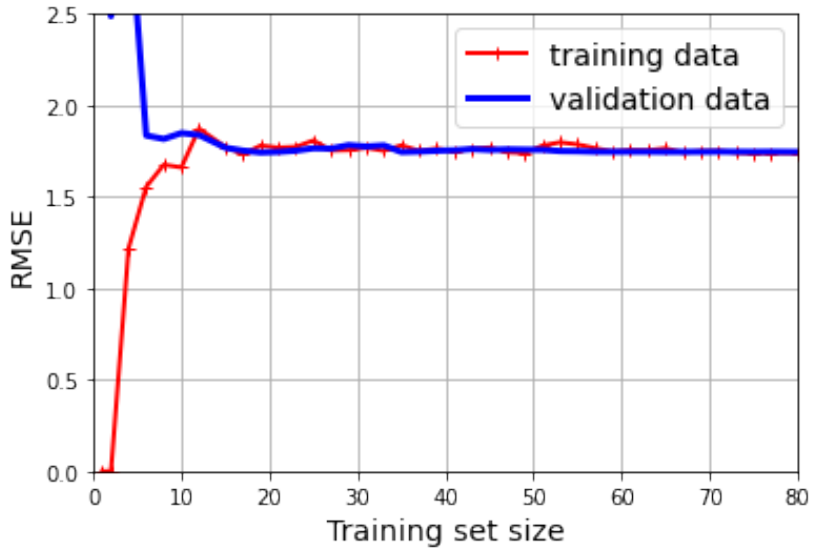


Polynomial Regression: Learning curves of the plain Linear Regression model

- Is the model over or underfitting?
  - The question now is how to decide how complex should be the model and how to determine the model is overfitting or underfitting the data.
- Learning curves are a way to see this:
  - These are plots of the model's performance on the training set and the validation set as a function of the training set size (or the training iteration).
  - To generate the plots, evaluate the model (e.g., with RMSE) at regular intervals during training on both the training set and the validation set, and plot the results
    - For each model generated with the respective subset of the training set, the root mean square error (RMSE) is calculated.
    - In addition, the RMSE is also calculated with all validation data for each model generated with this subset.
- As mentioned in the regression part, it is core to machine learning that we differ between training and validation data set.
- For this example, we start with an 80/20 split between training and validation data set (see next slides for illustrations)

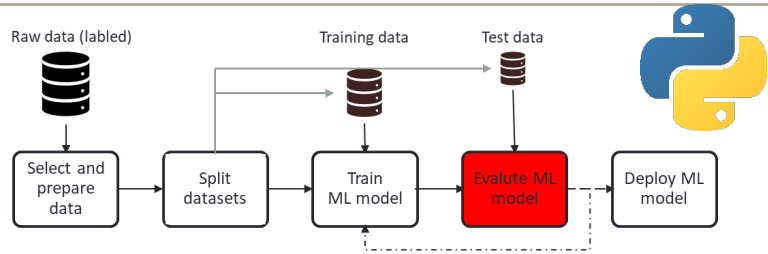
Polynomial Regression: Learning curves of the plain Linear Regression model

- Look at the learning curves of the simple linear regression model (without any polynomial features) with up to 80 training and 20 validation
  - The representation of the learning curve of a model is based on an increasing number of training data.



Polynomial Regression: Learning curves

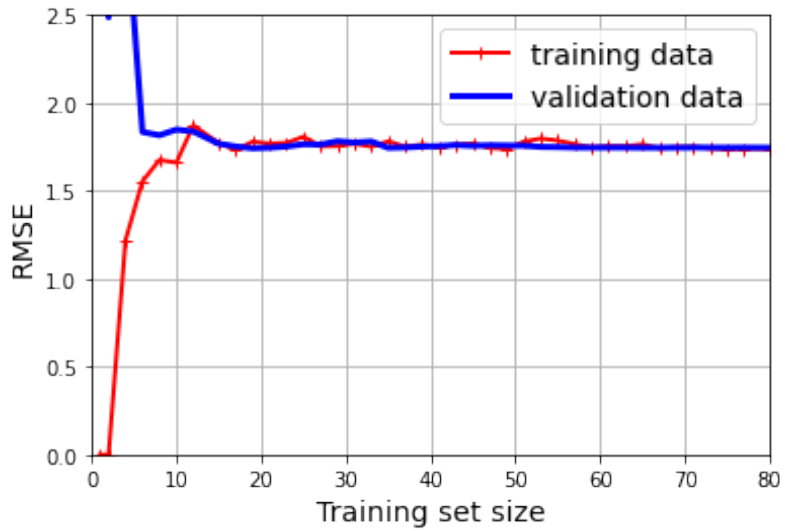

```python
from sklearn.model_selection import learning_curve

train_sizes, train_scores, valid_scores = learning_curve(
    LinearRegression(), df.Time.array.reshape(-1, 1), df.Revenue, train_sizes=np.linspace(0.01,
1.0, 40),
    scoring="neg_root_mean_squared_error")
train_errors = -train_scores.mean(axis=1)
valid_errors = -valid_scores.mean(axis=1)

# plotting the figure
plt.figure(figsize=(6, 4))
plt.plot(train_sizes, train_errors, "r-+", linewidth=2, label="training data")
plt.plot(train_sizes, valid_errors, "b-", linewidth=3, label="validation data")
plt.xlabel("Training set size")
plt.ylabel("RMSE")
plt.grid()
plt.legend(loc="upper right")
plt.axis([0, 80, 0, 2.5])
```



Polynomial Regression: Learning curves

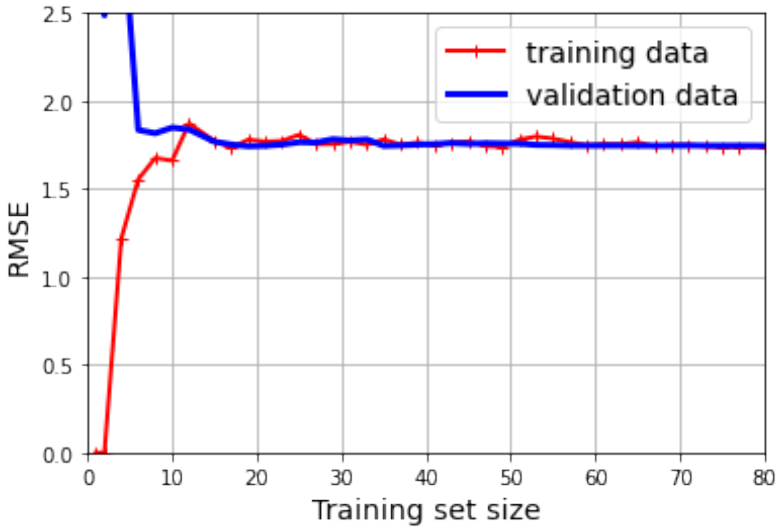
- This model that's underfitting deserves a bit of explanation.

- First, let's look at the performance on the training data (red line):
  - when there are just one or two instances in the training set, the model can fit them perfectly, which is why the curve starts at zero.
  - But as new instances are added to the training set, it becomes impossible for the model to fit the training data perfectly, both because the data is noisy and because it is not linear at all.
  - The error on the training data goes up until it reaches a plateau, at which point adding new instances to the training set doesn't make the average error much better or worse.



Polynomial Regression: Learning curves

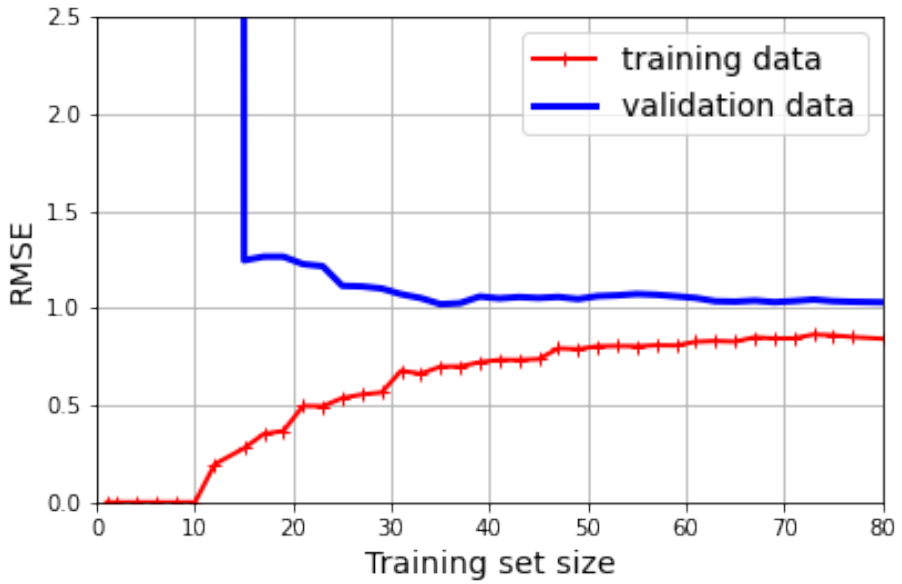
- Let's look at the performance of the model on the validation data.
  - When the model is trained on very few training instances, it is incapable of generalizing properly, which is why the validation error is initially quite big.
  - Then, as the model is shown more training examples, it learns, and thus the validation error slowly goes down.
  - However, once again a straight line cannot do a good job modeling the data, so the error ends up at a plateau, very close to the other curve.



- The learning curves are typical of a model that's underfitting.
  - Both curves have reached a plateau; they are close and fairly high.
  - If your model is underfitting the training data, adding more training examples will not help.
  - You need to use a more complex model or come up with better features (e.g. here polynomial features)

Polynomial Regression: Learning curves

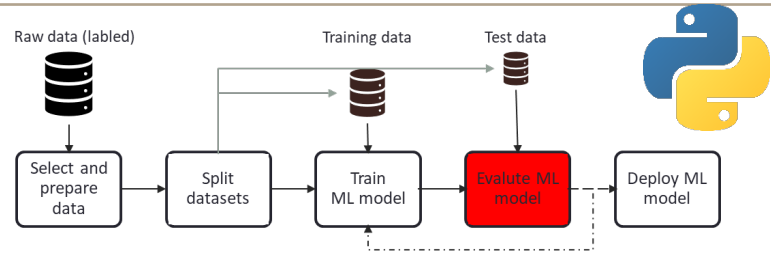
- Now let's look at the learning curves of a 10th-degree polynomial model on the same data (code is on next slide)



Polynomial Regression: Learning curves

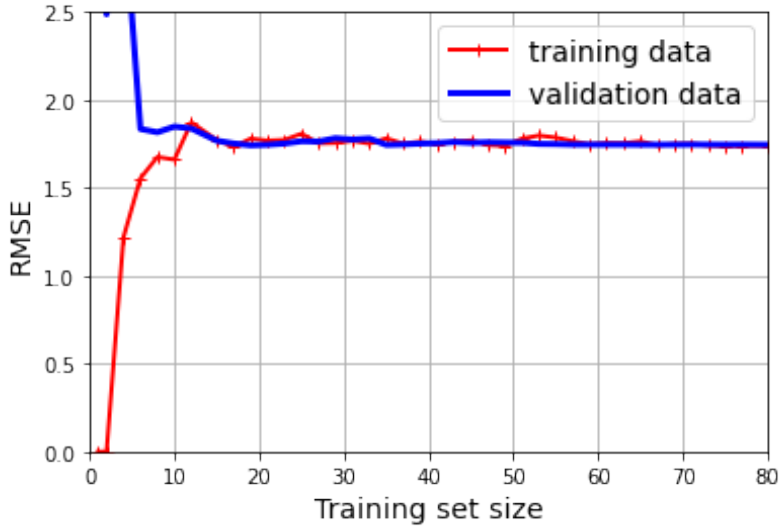

```
from sklearn.pipeline import make_pipeline

polynomial_regression = make_pipeline(
    PolynomialFeatures(degree=10, include_bias=False),
    LinearRegression())

train_sizes, train_scores, valid_scores = learning_curve(
   polynomial_regression, df.Time.array.reshape(-1, 1), df.Revenue,
train_sizes=np.linspace(0.01, 1.0, 40),
   scoring="neg_root_mean_squared_error")
```

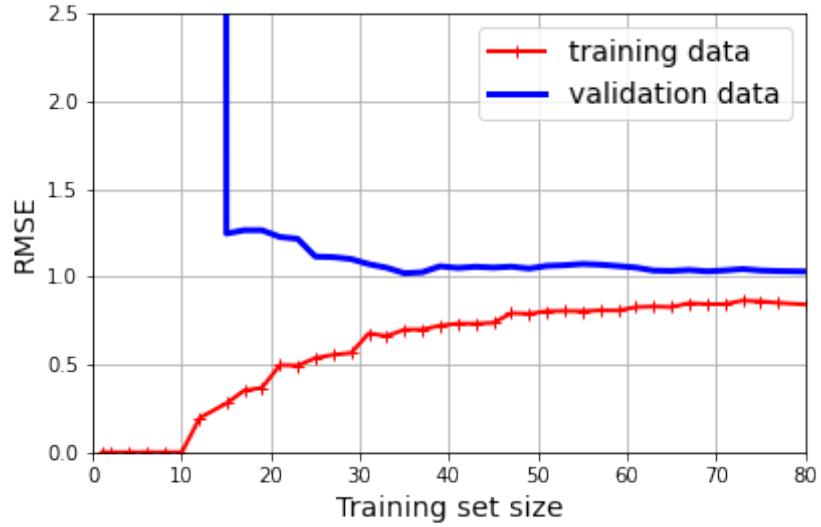
```
train_errors = -train_scores.mean(axis=1)
valid_errors = -valid_scores.mean(axis=1)

plt.figure(figsize=(6, 4))
plt.plot(train_sizes, train_errors, "r-+", linewidth=2, label="training data")
plt.plot(train_sizes, valid_errors, "b-", linewidth=3, label="validation data")
plt.legend(loc="upper right")
plt.xlabel("Training set size")
plt.ylabel("RMSE")
plt.grid()
plt.axis([0, 80, 0, 2.5])
plt.show()
```



Polynomial Regression: Learning curves

- These learning curves look a bit like the previous ones, but there are two very important differences:
  - The error on the training data is much lower than with the Linear Regression model.
  - There is a gap between the curves.
    - This means that the model performs significantly better on the training data than on the validation data, which is the hallmark of an overfitting model.
    - If a much larger training set were used, however, the two curves would continue to get closer.





→ One way to improve an overfitting model is to feed it more training data until the validation error reaches the training error.
We will learn more about over- and underfitting in the next lectures.

Regression

- Linear Regression with Normal Equation
- Linear Regression with (Stochastic) Gradient Descent
- Polynomial Regression
- **Ridge Regression**
- Lasso Regression
- Elastic Net and Comparison
- Early Stopping
- Regression Trees

Regularized Linear Models

- A good way to reduce overfitting is to regularize the model (i.e., to constrain it):
  - the fewer degrees of freedom it has, the harder it will be for it to overfit the data.
  - A simple way to regularize a polynomial model is to reduce the number of polynomial degrees.

- For a linear model, regularization is typically achieved by constraining the weights of the model.
  - We will now look at Ridge Regression, Lasso Regression and Elastic Net, which implement three different ways to constrain the weights.

Regularized Linear Models: Ridge regression

- Ridge Regression (also called Tikhonov regularization) is a regularized version of Linear Regression:
  - a regularization term is added to the cost function.
  - This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible.
  - Note that the regularization term should only be added to the cost function during training.
  - Once the model is trained, you want to use the unregularized performance measure to evaluate the model's performance.

- The hyperparameter α controls how much you want to regularize the model.
  - If α = 0, then Ridge Regression is just Linear Regression.
  - If α is very large, then all weights end up very close to zero and the result is a flat line going through the data's mean.

Regularized Linear Models: Ridge regression

- It is important to scale the data (e.g., using a StandardScaler) before performing Ridge Regression, as it is sensitive to the scale of the input features. This is true of most regularized models.

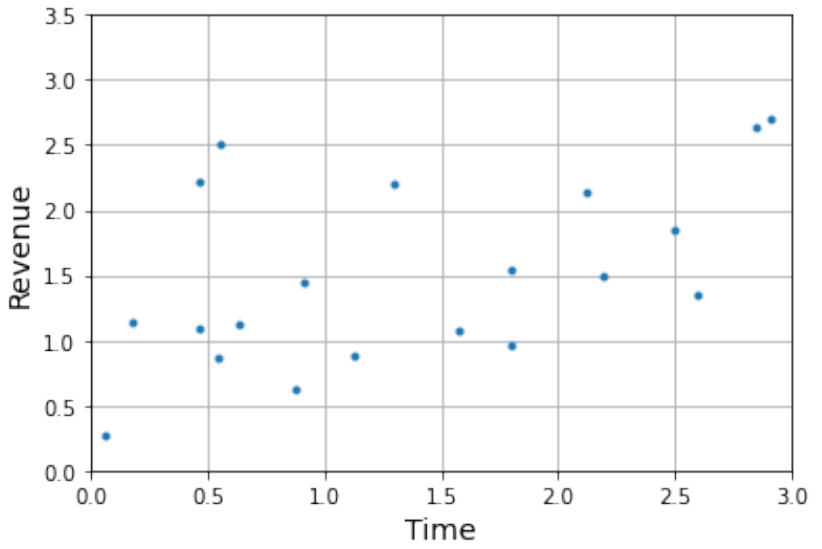
- Ridge Regression with Normal Equation (closed-form solution), a variant of Equation uses a matrix factorization technique by André-Louis Cholesky:

$$\hat{\theta} = (X^T \cdot X + \alpha \cdot A)^{-1} \cdot X^T \cdot y$$

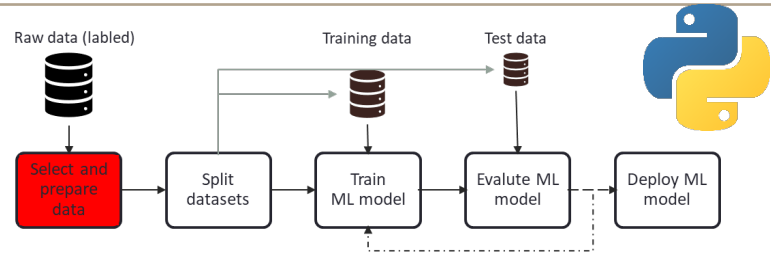
Example

- We use the following new “mobile app dataset”, which differs from the previous ones.

- Context:
  - The revenue of a mobile app user is influenced by many factors, including the time a user spends in the app per day
  - As a data scientist your job is to use supervised machine learning to predict the revenue of mobile app user



Regularized Linear Models: Ridge regression

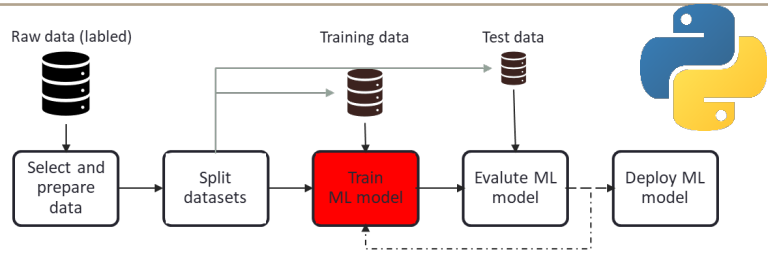


Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
data3 = pd.read_csv("https://raw.githubusercontent.com/VAWi-DataScience/Data-Science-and-Machine-Learning/main/Lecture/Data/03_Regression_data3.csv",sep=',')
data3.head()
```

	Time	Revenue
0	1.123620	0.886589
1	2.852143	2.635570
2	2.195982	1.492642
3	1.795975	0.956452
4	0.468056	2.211127

Regularized Linear Models: Ridge regression




```python
from sklearn.linear_model import Ridge

ridge_reg = Ridge(alpha=0.1, solver="cholesky")
ridge_reg.fit(data3.Time.array.reshape(-1, 1), data3.Revenue)
```

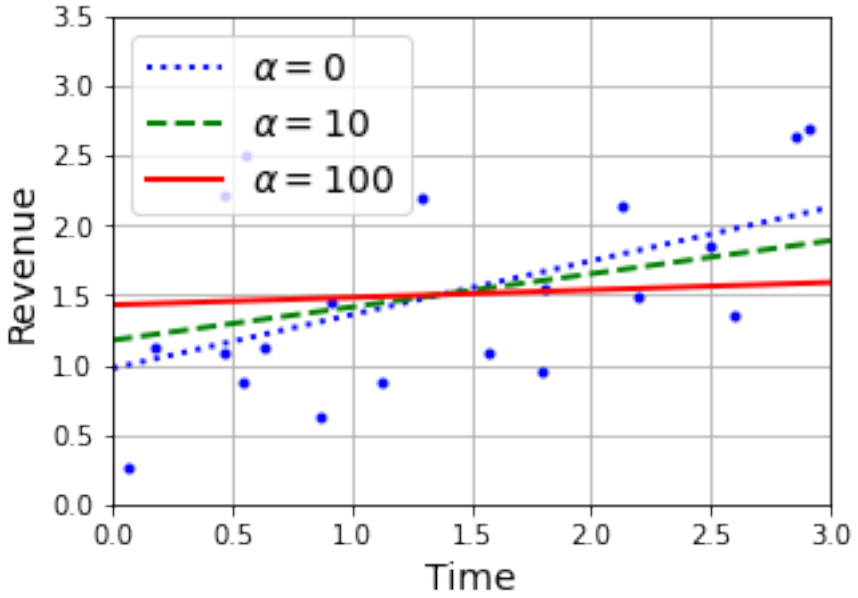
Alternative when using Stochastic Gradient Descend:

- The penalty hyperparameter sets the type of regularization term, where "l2" corresponds to Ridge regression.

Regularized Linear Models: Ridge regression

- Figure shows several Ridge models trained on some linear data using different α values.
  - Plain Ridge models are used, leading to linear predictions.
  - Note how increasing α leads to flatter (i.e. less extreme) predictions

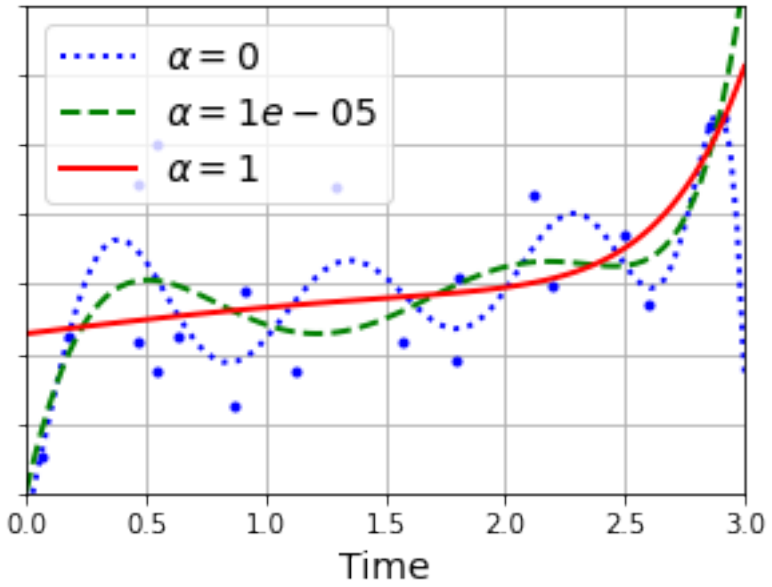
- The hyperparameter α controls how much you want to regularize the model.
  - If α = 0, then Ridge Regression is just Linear Regression.
  - If α is very large, then all weights end up very close to zero and the result is a flat line going through the data's mean.



Regularized Linear Models: Ridge regression

- The data is first expanded using PolynomialFeatures(degree=10), then it is scaled using a StandardScaler, and finally the Ridge models are applied to the resulting features

- This is Polynomial Regression with Ridge regularization.



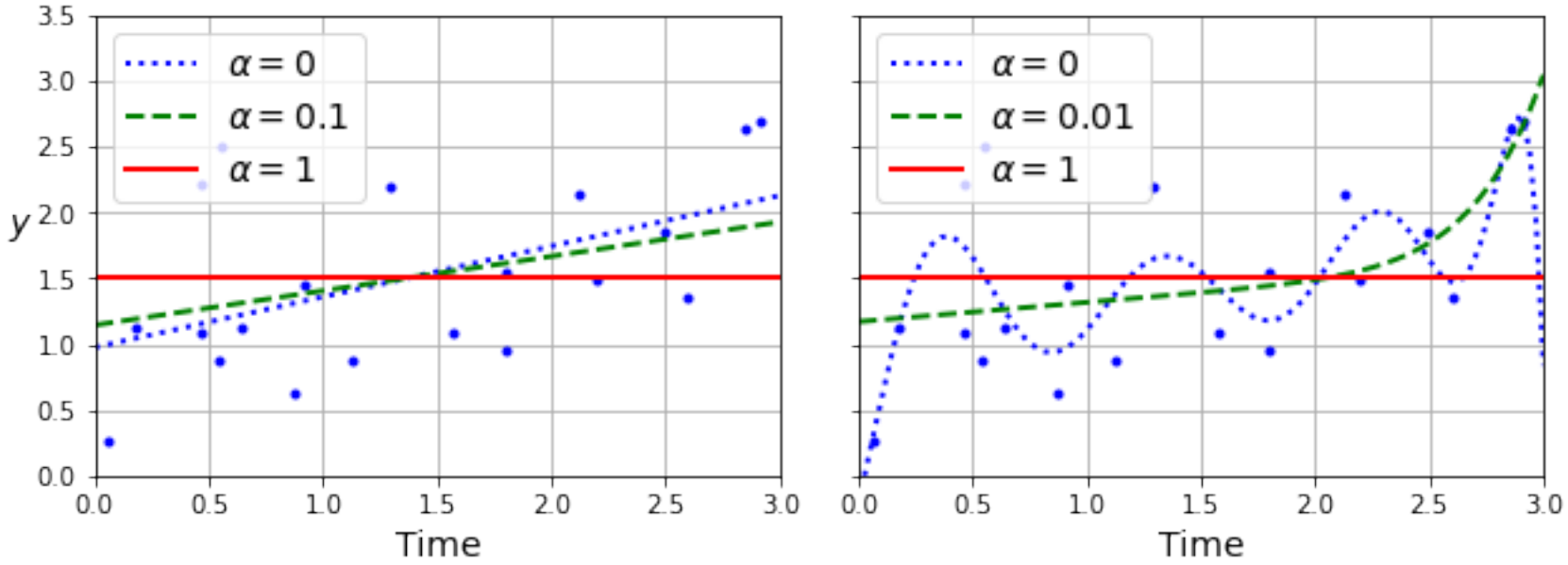
Regression

- Linear Regression with Normal Equation
- Linear Regression with (Stochastic) Gradient Descent
- Polynomial Regression
- Ridge Regression
- **Lasso Regression**
- Elastic Net and Comparison
- Early Stopping
- Regression Trees

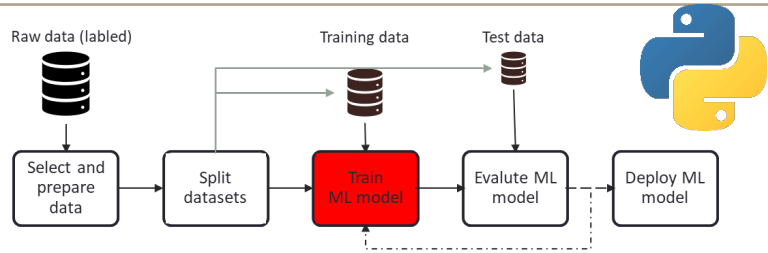
Regularized Linear Models: Lasso regression

- Least Absolute Shrinkage and Selection Operator Regression (usually simply called Lasso Regression) is another regularized version of Linear Regression.

- Left figure shows a linear regression optimized with lasso regularization
- Right figure shows a polynomial regression optimized with lasso regularization



Regularized Linear Models: Lasso regression

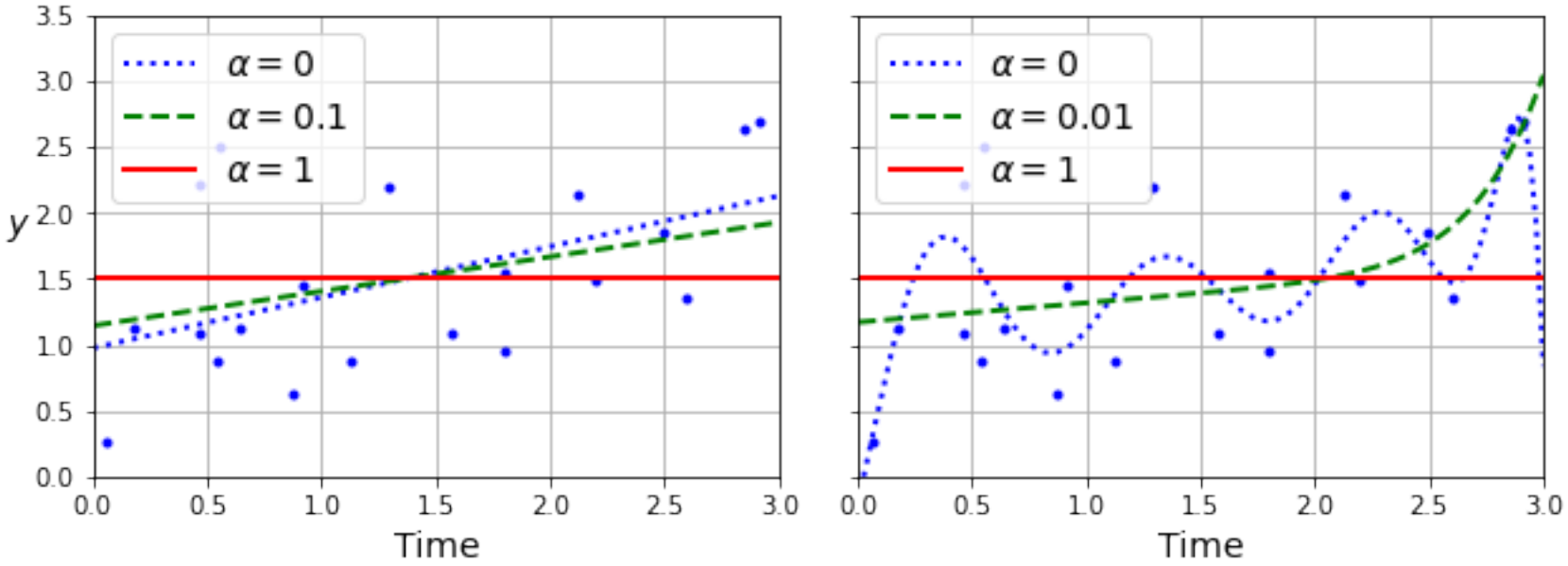



```
from sklearn.linear_model import Lasso

lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(data3.Time.array.reshape(-1, 1), data3.Revenue)
```

Alternative when using Stochastic Gradient Descend:
The penalty hyperparameter sets the type of regularization term, where "l1" corresponds to Lasso-regression.

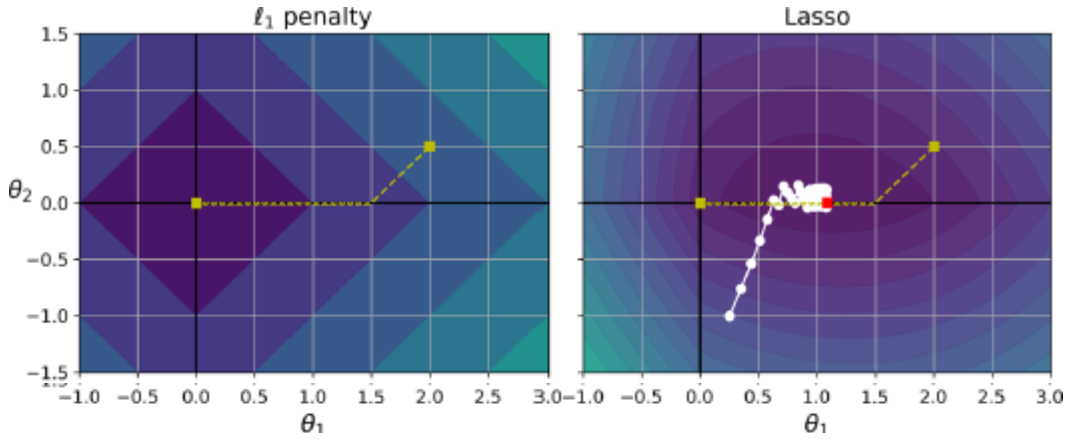
Regularized Linear Models: Lasso regression

- An important characteristic of Lasso Regression is that it tends to eliminate the weights of the least important features (i.e. set them to zero).
  - For example, the dashed line in the right plot (with α = 0.01) looks roughly cubic:
    - all the weights for the high-degree polynomial features are equal to zero.
    - In other words, Lasso Regression automatically performs feature selection and out-puts a sparse model (i.e. with few nonzero feature weights).



Regularized Linear Models: Lasso regression

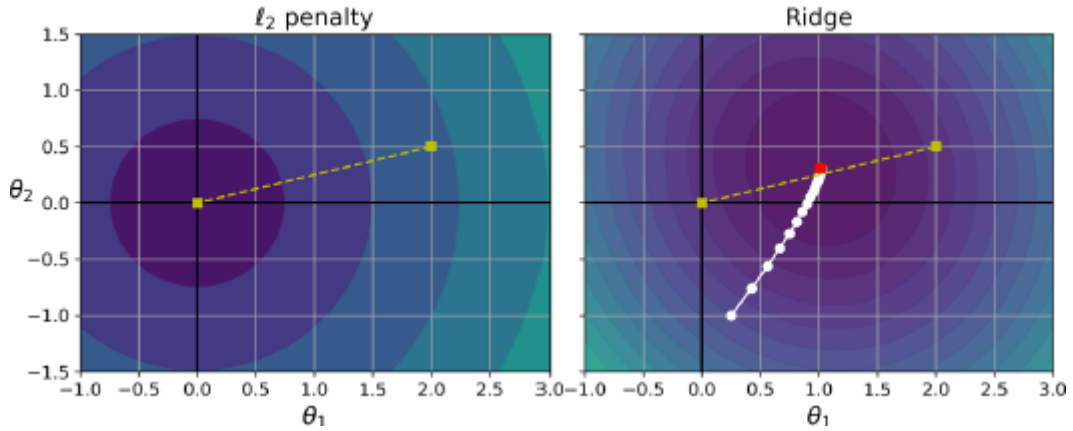
- You can get a sense of why this is the case by looking at the Figure:
  - the axes represent two model parameters, and the background contours represent different loss functions.
- In the left plot, the contures represent the ℓ1, drops linearly as you get closer to any axis.
  - For example, if you initialize the model parameters to θ1 = 2 und θ2 = 0,5 , running Gradient Descent will decrement both parameters equally (as represented by the dashed yellow line);
  - Therefore, θ2 will reach 0 first (since it was closer to 0 to begin with).
  - After that, Gradient Descent will roll down the gutter until it reaches θ1 = 0.
- In the right plot, the contours represent Lasso's function.
  - The small white circles show the path that Gradient Descent takes to optimize some model parameters that were initialized around θ1 = 0.25 and θ2 = –1:
    - notice once again how the path quickly reaches θ2 = 0, then rolls down the gutter and ends up bouncing around the global optimum (represented by the red square).
    - If we increased α, the global optimum would move left along the dashed yellow line, while if we decreased α, the global optimum would move right.



Regularized Linear Models: Lasso regression

- The two plots show the same thing but with an ℓ2 penalty instead.
  - In the left plot, you can see that the Gradient Descent just takes a straight path toward that point.
  - In the right plot, the contours represent Ridge Regression's function

- There are two main differences with Lasso.
  - First, the gradients get smaller as the parameters approach the global optimum, so Gradient Descent naturally slows down, which helps convergence (as there is no bouncing around).
  - Second, the optimal parameters (represented by the red square) get closer and closer to the origin when you increase α, but they never get eliminated entirely.



  - To avoid Gradient Descent from bouncing around the optimum at the end when using Lasso, you need to gradually reduce the learning rate during training (it will still bounce around the optimum, but the steps will get smaller and smaller, so it will converge).

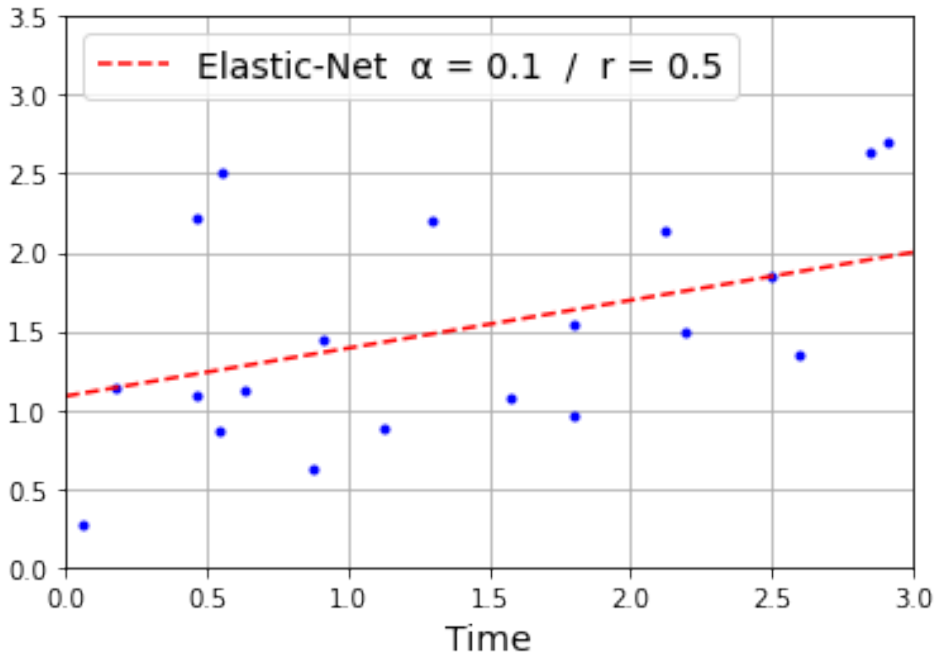
Regression

- Linear Regression with Normal Equation
- Linear Regression with (Stochastic) Gradient Descent
- Polynomial Regression
- Ridge Regression
- Lasso Regression
- **Elastic Net and Comparison**
- Early Stopping
- Regression Trees

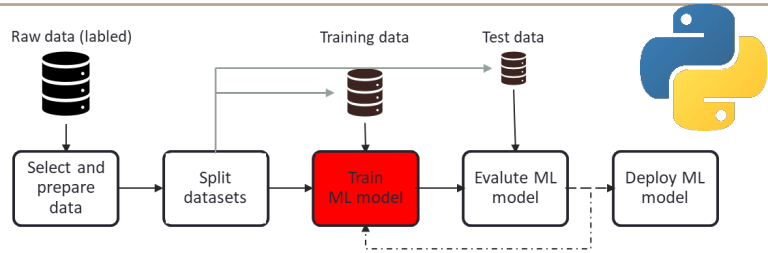
Regularized Linear Models: Elastic Net

- Elastic Net is a middle ground between Ridge Regression and Lasso Regression. The regularization term is a simple mix of both Ridge and Lasso's regularization terms, and you can control the mix ratio r. When r = 0, Elastic Net is equivalent to Ridge Regression, and when r = 1, it is equivalent to Lasso Regression.

- So when should you use plain Linear Regression (i.e., without any regularization), Ridge, Lasso, or Elastic Net?
  - It is almost always preferable to have at least a little bit of regularization, so generally you should avoid plain Linear Regression.
  - Ridge is a good default, but if you suspect that only a few features are useful, you should prefer Lasso or Elastic Net because they tend to reduce the useless features' weights down to zero, as we have discussed.
  - In general, Elastic Net is preferred over Lasso because Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.



Regularized Linear Models: Elastic Net



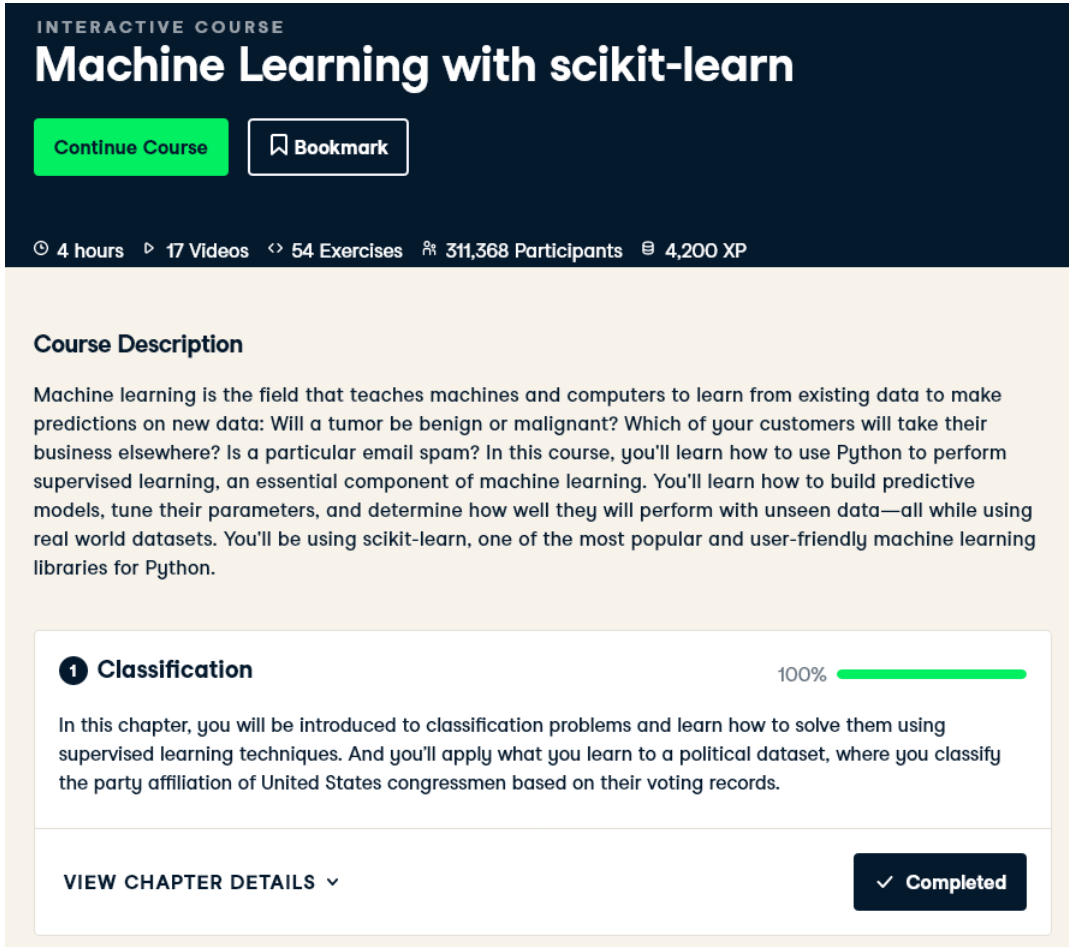

```
from sklearn.linear_model import ElasticNet

elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_net.fit(data3.Time.array.reshape(-1, 1), data3.Revenue)
```

Regularized Linear Models: Comparison

- Comparison of Linear Regression, polynomial Regression, Ridge Regression, Lasso Regression, and Elastic Net.

	Linear Regression	Polynomial Regression	Ridge-Regression, Lasso-Regression and Elastic Net
Target function	▪ linear relationship (straight line)	▪ quadratic relation ▪ Polynomial of higher degree	▪ linear relationship ▪ quadratic relation ▪ Polynomial of higher degree
Hyperparameters and methods	▪ no Hyperparameters in linear regression with normal equation ▪ ≥ 2 Hyperparameters with Stochastic Gradient Descent	▪ no Hyperparameters in linear regression with normal equation ▪ ≥ 2 Hyperparameters with Stochastic Gradient Descent	▪ 1 hyperparameter plus mixing ratio for Elastic Net ▪ Stochastic gradient method with ridge regression (l2) or lasso regression (l1)
Scaling and Centering	▪ not required	▪ not required	▪ Required for quadratic polynomial and polynomials of higher degree

Datacamp



https://www.datacamp.com/courses/machine-learning-with-scikit-learn

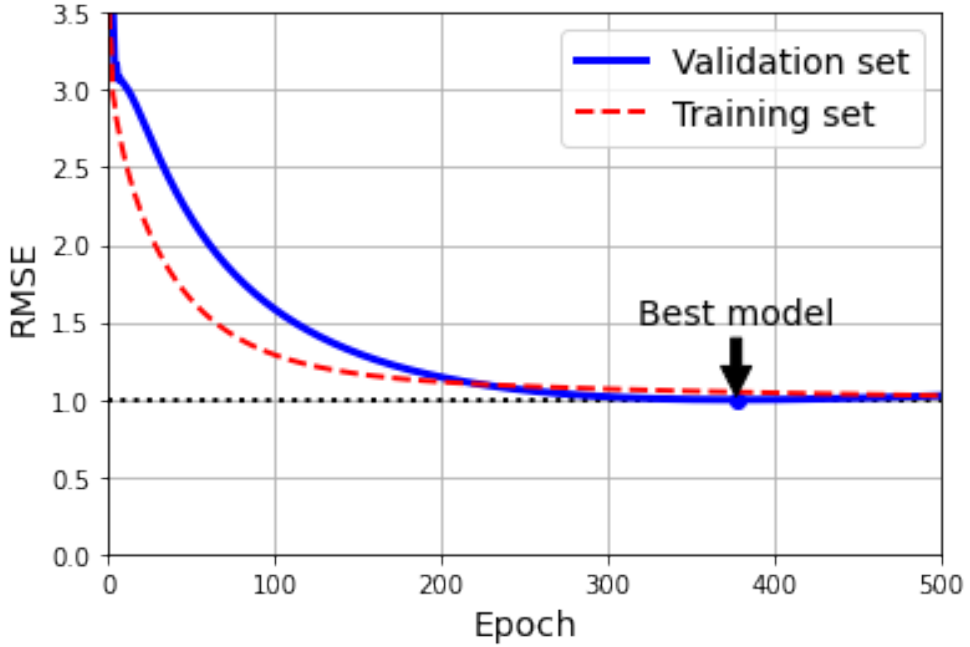
Regression

- Linear Regression with Normal Equation
- Linear Regression with (Stochastic) Gradient Descent
- Polynomial Regression
- Ridge Regression
- Lasso Regression
- Elastic Net and Comparison
- **Early Stopping**
- Regression Trees

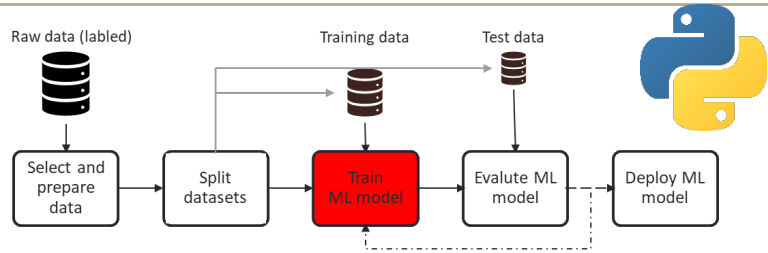
Regularized Linear Models: Early Stopping

- A very different way to regularize iterative learning algorithms such as Gradient Descent is to stop training as soon as the validation error reaches a minimum. This is called early stopping.

- The Figure shows a complex model (in this case, a high-degree Polynomial Regression model).



Regularized Linear Models: Early Stopping




```python
from copy import deepcopy
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

preprocessing = make_pipeline(PolynomialFeatures(degree=90, include_bias=False),
                              StandardScaler())
X_train_prep = preprocessing.fit_transform(X_train)
X_valid_prep = preprocessing.transform(X_valid)
sgd_reg = SGDRegressor(penalty=None, eta0=0.002, random_state=42)
n_epochs = 500
best_valid_rmse = float('inf')

for epoch in range(n_epochs):
    sgd_reg.partial_fit(X_train_prep, y_train)
    y_valid_predict = sgd_reg.predict(X_valid_prep)
    val_error = mean_squared_error(y_valid, y_valid_predict, squared=False)
    if val_error < best_valid_rmse:
        best_valid_rmse = val_error
        best_model = deepcopy(sgd_reg)
```

Regularized Linear Models: Early Stopping

- As the iterations go by the algorithm learns, and its prediction error (RMSE) on the training set goes down, along with its prediction error on the validation set.
  - After a while though, the validation error stops decreasing and starts to go back up. This indicates that the model has started to overfit the training data.
  - With early stopping you just stop training as soon as the validation error reaches the minimum. It is such a simple and efficient regularization technique.
  - With Stochastic Gradient Descent, the curves are not so smooth, and it may be hard to know whether you have reached the minimum or not.

- One solution is to stop only after the validation error has been above the minimum for some time, then roll back the model parameters to the point where the validation error was at a minimum.

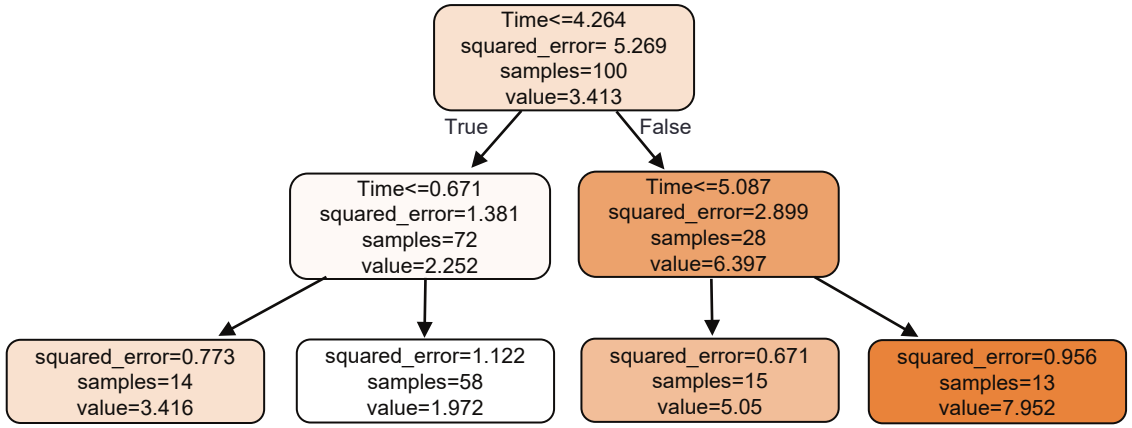
Regression

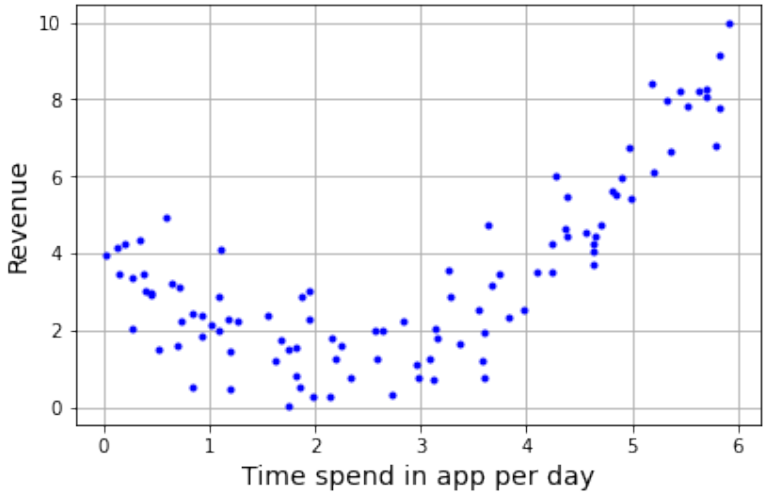
- Linear Regression with Normal Equation
- Linear Regression with (Stochastic) Gradient Descent
- Polynomial Regression
- Ridge Regression
- Lasso Regression
- Elastic Net and Comparison
- Early Stopping
- **Regression Trees**

Regression Trees

- Decision Trees, which we used so far for classification tasks, are also capable of performing regression tasks.
  - The main difference is that instead of predicting a class in each node, it predicts a value.
  - To build a regression tree one can use the Scikit-Learn's DecisionTreeRegressor class

- Example (using the previous data set)
  - Suppose you want to make a prediction for a new instance with Time = 5.
  - You traverse the tree starting at the root, and you eventually reach the leaf node that predicts value = 5.05.
  - This prediction is the average target value of the 15 training instances associated with this leaf node, and it results in a mean squared error equal to 0.671 over these 15 instances.

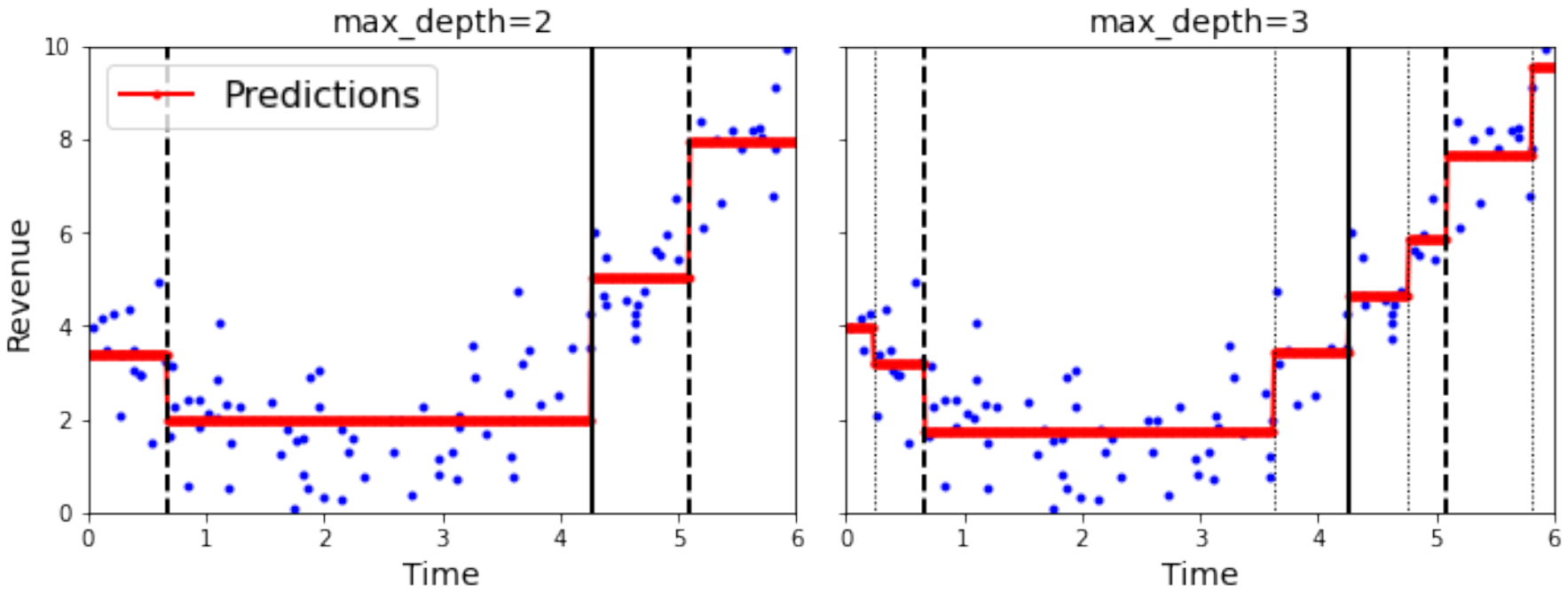




Regression Trees

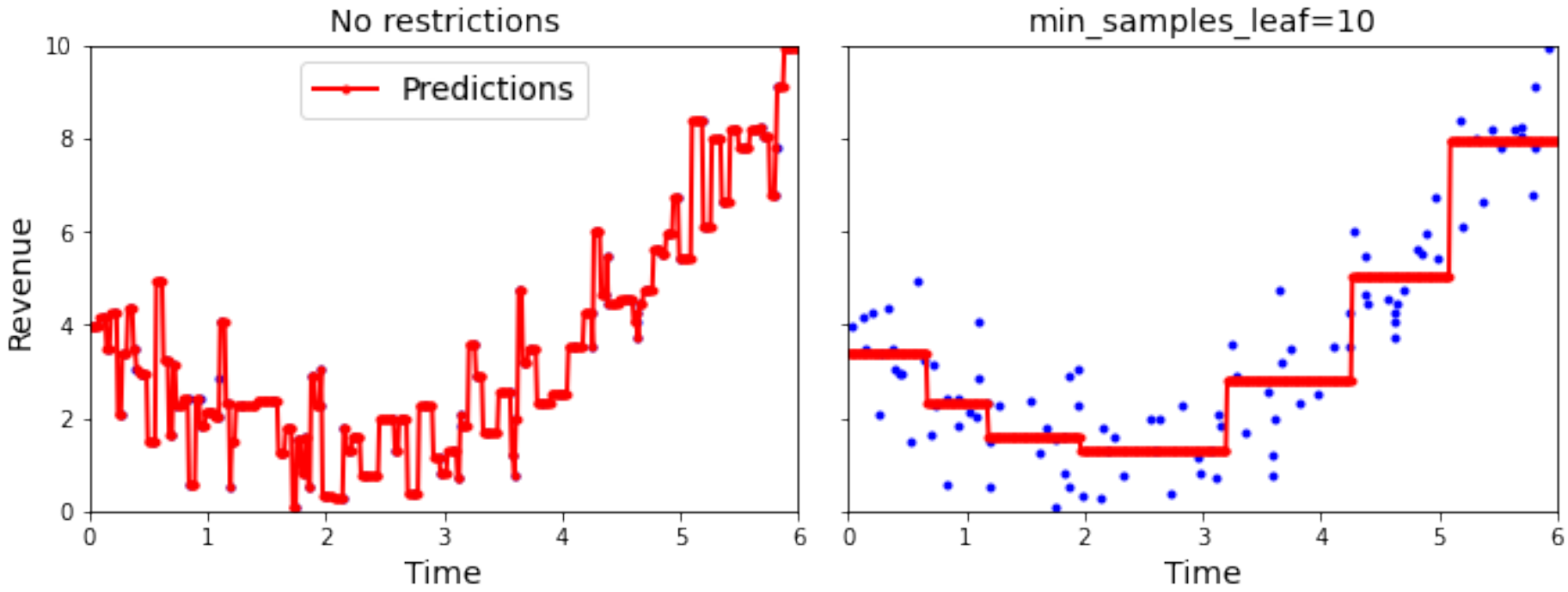
- This model's predictions are represented on the left in Figure.
  - Notice how the predicted value for each region is always the average target value of the instances in that region.
  - The algorithm splits each region in a way that makes most training instances as close as possible to that predicted value.

- If you set max_depth = 3, you get the predictions represented on the right.

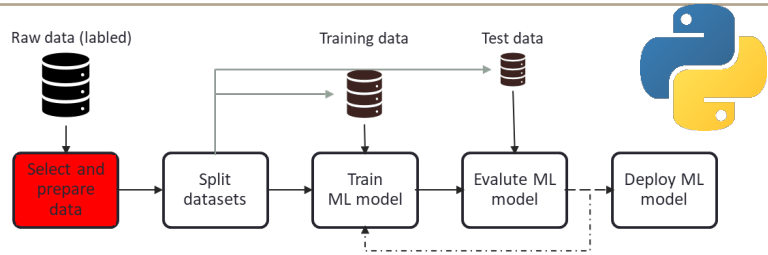


Regression Trees

- Just like for classification tasks, regression trees are prone to overfitting
  - Without any regularization (i.e., using the default hyperparameters), you get the predictions on the left in Figure.
  - These predictions are obviously overfitting the training set very badly.
  - Just setting min_samples_leaf = 10 results in a much more reasonable model, represented on the right in Figure.



Regression Trees

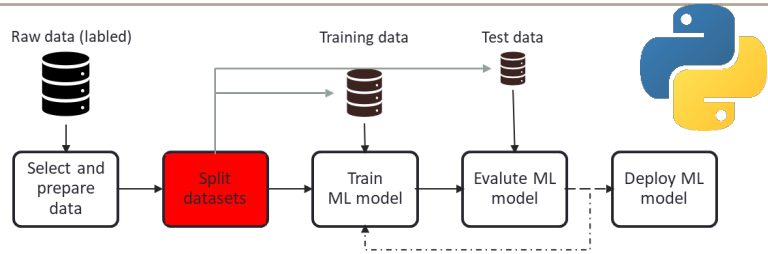


Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
data = pd.read_csv("https://raw.githubusercontent.com/VAWi-DataScience/Data-Science-and-Machine-Learning/main/Lecture/Data/03_Regression_data1.csv",sep=',')
data.head()
```

	Time	Revenue
0	2.247241	1.617611
1	5.704286	8.061859
2	4.391964	4.452506
3	3.591951	0.779585
4	0.936112	1.846257

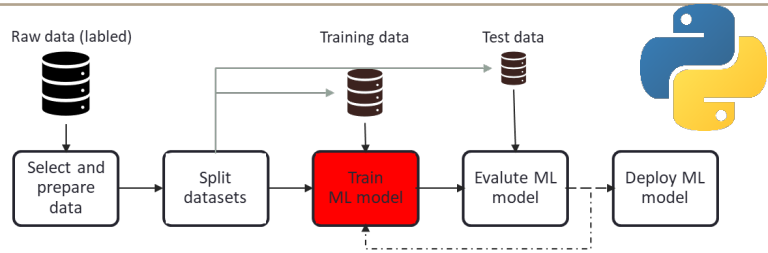
Regression Trees



Split dataset

```
# import train_test_split function
from sklearn.model_selection import train_test_split

# Split X and y into training and testing sets
X_train,X_test, y_train, y_test = train_test_split(df.Time.array.reshape(),df.Revenue,
test_size=0.2, random_state=42)
```

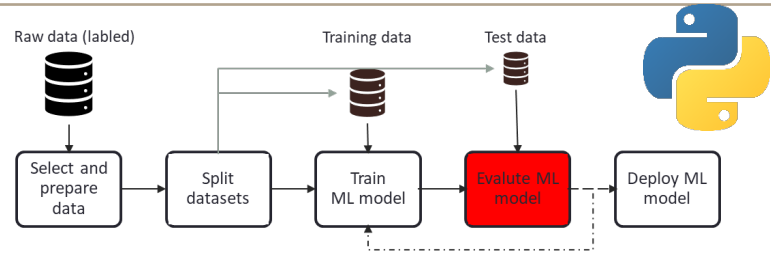
Regression Trees



Train ML model

```
from sklearn.tree import DecisionTreeRegression
tree_reg = DecisionTreeRegressor(min_samples_leaf=5, max_depth=4, random_state=42)
tree_reg.fit(X_train, y_train)
```

Regression Trees

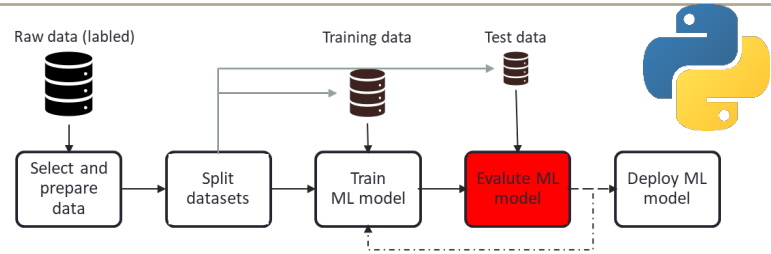


Evaluate (training data)

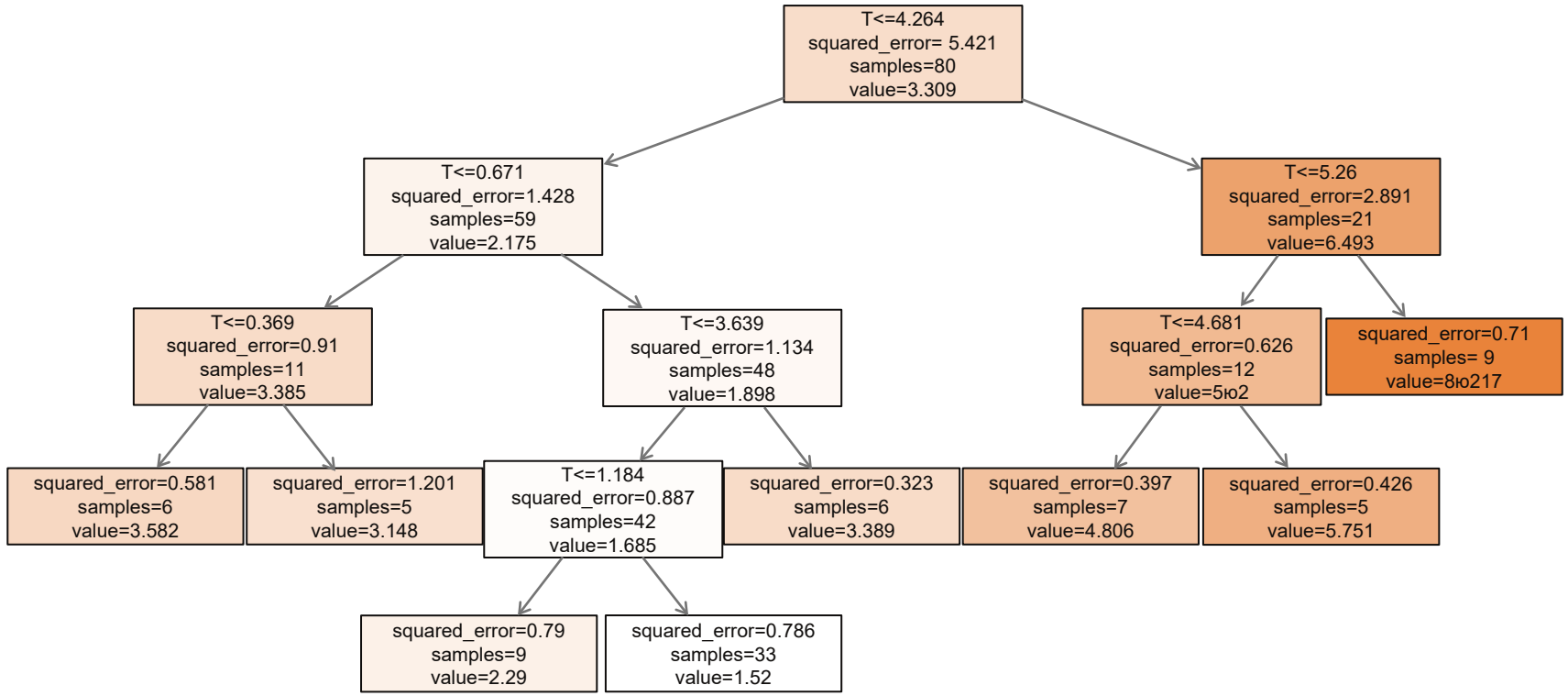
```
predictions_train_tree_reg = tree_reg.predict(X_train)
# Import the required libraries
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Evaluate mean absolute error
print('Mean Absolute Error(MAE):', mean_absolute_error(y_train,predictions_train_tree_reg))

# Evaluate mean squared error
print("Mean Squared Error(MSE):", mean_squared_error(y_train, predictions_train_tree_reg))

# Evaluate root mean squared error
print("Root Mean Squared Error(RMSE):", np.sqrt(mean_squared_error(y_train,
predictions_train_tree_reg)))

# Evaluate R2-square
print("R2-Square:",r2_score(y_train, predictions_train_tree_reg))
```

```
Mean Absolute Error(MAE): 0.6432604099211322
Mean Squared Error(MSE): 0.6972290152885586
Root Mean Squared Error(RMSE): 0.8350024043609447
R2-Square: 0.8713875436174398
```

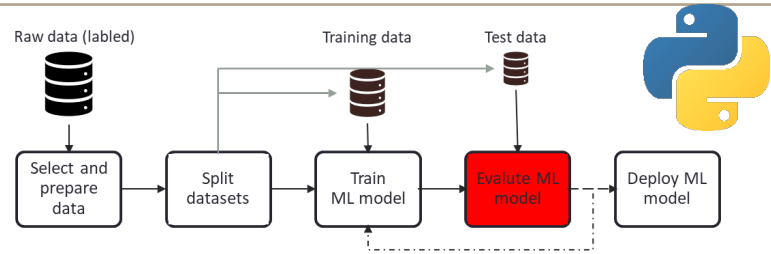
Regression Trees




```python
import matplotlib as plt
from sklearn import tree
plt.figure(figsize=(40,20))  # customize according to the size of your tree
_ = tree.plot_tree(tree_reg,feature_names = "Time", filled=True)
plt.show()
```



Regression Trees



Evaluate (test data)

```
predictions_test_tree_reg = tree_reg.predict(X_test)
# Import the required libraries
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Evaluate mean absolute error
print('Mean Absolute Error(MAE):', mean_absolute_error(y_test,predictions_test_tree_reg))

# Evaluate mean squared error
print("Mean Squared Error(MSE):", mean_squared_error(y_test, predictions_test_tree_reg))

# Evaluate root mean squared error
print("Root Mean Squared Error(RMSE):", np.sqrt(mean_squared_error(y_test,
predictions_test_tree_reg)))

# Evaluate R2-square
print("R2-Square:",r2_score(y_test, predictions_test_tree_reg))
```

```
Mean Absolute Error(MAE): 0.6432604099211322
Mean Squared Error(MSE): 0.6972290152885586
Root Mean Squared Error(RMSE): 0.8350024043609447
R2-Square: 0.8713875436174398
```

Regression Summary (1 of 2)

- The simplest representative of Regression is the simple **Linear Regression**, because it models the influence of a variable on a target variable by a linear function. It is well suited as a foundation for understanding more complex models.
- A **Hyperparameter** is a property of a learning algorithm that is usually assigned a numeric value and this value then influences how the algorithm works.
- **Stochastic Gradient Descent** picks a random instance in the training set at every step and computes the gradients based only on that single instance. Obviously, working on a single instance at a time makes the algorithm much faster because it has very little data to manipulate at every iteration.
- If a high degree **Polynomial Regression** is performed, the training data will likely fit much better than with a simple linear regression.

Regression Summary (2 of 2)

- A good way to reduce overfitting is to regularize the model: A simple way to regularize a polynomial model is to reduce the number of polynomial degrees. Other methods are the Ridge Regression or Lasse Regression.
- **Ridge Regression:** a regularization term is added to the cost function. This forces the learning algorithm to keep the model weights as small as possible.
- An important characteristic of **Lasso Regression** is that it tends to eliminate the weights of the least important features.
- **Elastic Net** is a is a simple mix between Ridge Regression and Lasso Regression and you can control the mix ratio.
- **Early Stopping**: A different way to regularize iterative learning algorithms such as Gradient Descent is to stop training as the validation error reaches a minimum.
- **Decision Trees** are versatile Machine Learning algorithms that can perform both classification and regression tasks.