**Table of Contents**

[**Introduction to Machine Learning** 2](#_Toc164954757)

[Introduction 2](#_Toc164954758)

[Boots that fit 2](#_Toc164954759)

[What are machine learning models? 2](#_Toc164954760)

[How to think about models 2](#_Toc164954761)

[Models are often simple code 2](#_Toc164954762)

[Select a model 2](#_Toc164954763)

[Parameters are discovered during training 3](#_Toc164954764)

[What are inputs and outputs? 3](#_Toc164954765)

[The objective 3](#_Toc164954766)

[The data 3](#_Toc164954767)

[The optimizer 3](#_Toc164954768)

[How to use a model 4](#_Toc164954769)

[Training versus using a model 4](#_Toc164954770)

[Data for use, data for training 4](#_Toc164954771)

[I've finished training. What now? 4](#_Toc164954772)

[Summary 5](#_Toc164954773)

[**Build classical machine learning models with supervised learning 5**](#_Toc164954774)

[Define supervised learning 5](#_Toc164954775)

[What is unsupervised learning? 5](#_Toc164954776)

[What is supervised learning? 5](#_Toc164954777)

[Labels are only for learning 6](#_Toc164954778)

[What is unsupervised learning? 6](#_Toc164954779)

[What is supervised learning? 6](#_Toc164954780)

[Labels are only for learning 7](#_Toc164954781)

[Optimize models by using gradient descent 7](#_Toc164954782)

[Gradient descent 7](#_Toc164954783)

[Local minima 7](#_Toc164954784)

[Instability 8](#_Toc164954785)

[Summary 8](#_Toc164954786)

[**Introduction to data for machine learning 10**](#_Toc164954787)

[Good, bad, and missing data 10](#_Toc164954788)

['Representative' - What does that mean? 10](#_Toc164954789)

[What is a data error? 10](#_Toc164954790)

[What is complete data? 10](#_Toc164954791)

[Examine different types of data 11](#_Toc164954792)

[Continuous, ordinal, and categorical data 11](#_Toc164954793)

[Datatypes 11](#_Toc164954794)

[Derived datatypes 12](#_Toc164954795)

[Too many choices? 12](#_Toc164954796)

[Summary 12](#_Toc164954797)

[What are data? 12](#_Toc164954798)

[What makes a good dataset? 12](#_Toc164954799)

[Thinking about data 12](#_Toc164954800)

[**Train and understand regression models in machine learning 13**](#_Toc164954801)

[Introduction 13](#_Toc164954802)

[What is regression? 13](#_Toc164954803)

[Simple linear regression 13](#_Toc164954804)

[Fitting linear regression 13](#_Toc164954805)

[Strengths of regression 14](#_Toc164954806)

[Predictable and easy to interpret 14](#_Toc164954807)

[Easy to extrapolate 14](#_Toc164954808)

[Optimal fitting is usually guaranteed 14](#_Toc164954809)

[Multiple linear regression and R-squared 14](#_Toc164954810)

[Multiple linear regression 14](#_Toc164954811)

[Multiple linear regression has assumptions 15](#_Toc164954812)

[Goodness of fit: R2 15](#_Toc164954813)

[Multiple linear regression and R-squared 16](#_Toc164954814)

[Multiple linear regression 16](#_Toc164954815)

[Multiple linear regression has assumptions 16](#_Toc164954816)

[Goodness of fit: R2 16](#_Toc164954817)

[Summary 17](#_Toc164954818)

[**Refine and test machine learning models 18**](#_Toc164954819)

[Introduction 18](#_Toc164954820)

[Scenario: Training avalanche rescue dogs 18](#_Toc164954821)

[Normalization and standardization 18](#_Toc164954822)

[Normalization versus standardization 18](#_Toc164954823)

[Why do we need to scale? 18](#_Toc164954824)

[Scaling gives learning a better starting point 18](#_Toc164954825)

[Standardization lets parameters train at the same speed 19](#_Toc164954826)

[Scaling helps with multiple features 19](#_Toc164954827)

[Do I always need to scale? 19](#_Toc164954828)

[**Create and understand classification models in machine learning 20**](#_Toc164954829)

[Introduction 20](#_Toc164954830)

[Scenario: Predicting avalanches with machine learning 20](#_Toc164954831)

[Act 8 20](#_Toc164954832)

[What are classification models? 20](#_Toc164954833)

[What is logistic regression? 20](#_Toc164954834)

[Converting outputs to categories 21](#_Toc164954835)

[Assessing a classification model 21](#_Toc164954836)

[A refresher on cost 21](#_Toc164954837)

[Cost functions for classification 21](#_Toc164954838)

[Log loss 21](#_Toc164954839)

[Why not MSE? 22](#_Toc164954840)

[Improving classification models 22](#_Toc164954841)

[Provide more features 22](#_Toc164954842)

[The good and bad sides of independent features 22](#_Toc164954843)

[Think about your features 23](#_Toc164954844)

[Avoiding overtraining 23](#_Toc164954845)

[Avoiding undertraining 23](#_Toc164954846)

[Summary 23](#_Toc164954847)

[**Select And Customize Architectures And Hyperparameters Using Random Forest 24**](#_Toc164954848)

[Introduction 24](#_Toc164954849)

[Scenario: Predicting sports results using machine learning 24](#_Toc164954850)

[Learning objectives 24](#_Toc164954851)

[Decision trees and model architecture 24](#_Toc164954852)

[What's a decision tree? 24](#_Toc164954853)

[How are decision trees trained? 25](#_Toc164954854)

[Strengths and weaknesses of decision trees 25](#_Toc164954855)

[Model architecture affects overfitting 26](#_Toc164954856)

[Activity 10 26](#_Toc164954857)

[Random forests and selecting architectures 26](#_Toc164954858)

[What’s a random forest? 26](#_Toc164954859)

[How is a random forest trained? 26](#_Toc164954860)

[Advantages of random forest 26](#_Toc164954861)

[Disadvantages of random forest 26](#_Toc164954862)

[How can I customize these architectures? 27](#_Toc164954863)

[Act 11 27](#_Toc164954864)

# React

## React Components

A Component is one of the core building blocks of React. Components in React basically return a piece of JSX code that tells what should be rendered on the screen.

### Components

**Functional**: Simple JS functions and are stateless

function MyComponent(props) {

  return <h1>Hello, {props.name}</h1>;

}

**Class-based:** Uses JS classes to create stateful components

class MyComponent extends Component {

  render() {

    return <h1>Hello, {this.props.name}</h1>;

  }

}

**Nested:** Creates component inside another component. Components can be nested to create more complex UIs. This helps in breaking down the UI into smaller, reusable pieces.

function ChildComponent(props) {

    return <p>Child Component: {props.data}</p>;

  }

function ParentComponent() {

    const data = "Hello from Parent";

    return (

      <div>

        <h1>Parent Component</h1>

        <ChildComponent data={data} />

      </div>

    );

  }

### Using State in Components

**State in Function Components (using Hooks):**

import React, { useState } from 'react';

function MyComponent() {

  const [count, setCount] = useState(0);

  return (

    <div>

      <p>Count: {count}</p>

      <button onClick={() => setCount(count + 1)}>Increment</button>

    </div>

  );

}

**State in Class Components:**

class MyComponent extends Component {

    constructor(props) {

      super(props);

      this.state = { count: 0 };

    }

    render() {

      return (

        <div>

          <p>Count: {this.state.count}</p>

          <button onClick={() => this.setState({ count: this.state.count + 1 })}>

            Increment

          </button>

        </div>

      );

    }

  }

## Managing Data Inside and Outside Components: State and Props

### ****Props****

**Props (Properties)** are used to pass data from one component to another. They are read-only and are immutable within the component that receives them.

**Passing Props:** Props are passed as attributes to components in JSX.

#### <ChildComponent propName="value" />

**Accessing Props: Props are accessed inside the child component via the props object.**

#### function ChildComponent(props) { return <p>Value passed: {props.propName}</p>; }

### State

**State** is used to manage data within a component. It is mutable and can be updated using the ***setState*** method. State is primarily used with class components, but can also be managed in functional components using Hooks (**useState**).

**State in Class Components:** **State is initialized in the constructor or directly in the class body using the *state* object.**

#### class MyComponent extends React.Component {

#### constructor(props) {

#### super(props);

#### this.state = {

#### count: 0,

#### };

#### }

#### render() {

#### return <p>Count: {this.state.count}</p>;

#### }

#### }

**Updating State:** State should be updated using *setState* to ensure React re-renders components properly. It accepts an object or a function that returns an object representing the updated state.

#### this.setState((prevState) => ({

#### count: prevState.count + 1,

#### }));

### Example:

import React, { useState } from 'react';

// Parent Component

function Tutorial() {

  const [parentState, setParentState] = useState('Parent State');

  return (

    <div>

      <ChildComponent propName={parentState} />

    </div>

  );

}

// Child Component

function ChildComponent(props) {

  const [childState, setChildState] = useState('Child State');

  return (

    <div>

      <p>Parent Prop: {props.propName}</p>

      <p>Child State: {childState}</p>

      <button onClick={() => setChildState('Updated Child State')}>

        Update Child State

      </button>

    </div>

  );

}

export default Tutorial;

* **ParentComponent** manages **parentState** using **useState** and passes it to **ChildComponent** as a prop (propName).
* **ChildComponent** manages its own state (**childState**) using **useState** and updates it locally with a button click.

## React Events

In React, events are handled in a way similar to handling events on DOM elements. However, there are some syntactic differences and unique behaviors specific to React.

### Key Concepts of React Events

1. **Synthetic Events:** React events are wrapped in a cross-browser wrapper called "SyntheticEvent." This wrapper ensures that the events have consistent properties across different browsers.
2. **CamelCase Naming:** Event names in React are written in camelCase instead of lowercase. For example, the HTML event onclick becomes onClick in React.

### Event Handling

1. **onClick**: Triggers when a clickable element is clicked.

<button onClick={handleClick}>Click Me</button>

function handleClick() {

console.log('Button clicked');

}

1. **onChange:** Fires when the value of an input, select, or textarea changes.

<input type="text" value={value} onChange={handleChange} />

function handleChange(event) {

setValue(event.target.value);

}

1. **onSubmit:** Executes when a form is submitted.

<form onSubmit={handleSubmit}>

<input type="submit" value="Submit" />

</form>

function handleSubmit(event) {

event.preventDefault();

console.log('Form submitted');

}

1. **onKeyDown / onKeyUp:** Fires when a key is pressed or released while focus is on an element.

<input type="text" onKeyDown={handleKeyDown} />

function handleKeyDown(event) {

console.log('Key pressed:', event.key);

}

**Preventing Form Submission:**

function handleSubmit(event) {

event.preventDefault();

// Handle form submission logic

}

**Passing Parameters to Event Handlers**

<button onClick={() => handleClick(param)}>Click Me</button>

function handleClick(param) {

console.log('Button clicked with parameter:', param);

}

## React Hooks Cheat Sheet

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

### ****Key Terms and Definitions****

* State: Data that changes over time and affects what is rendered by the component.
* Props: Read-only attributes passed from a parent component to a child component.
* Effect: Side operations like fetching data, updating the DOM, or subscribing to services.
* Ref: A reference to a DOM element or a mutable object that persists across renders.
* Memoization: Optimization technique to cache results of expensive function calls and reuse them.

### Hook Rules

* **Inside Function Components**: Hooks must be used within React functional components to ensure they are part of React's lifecycle management.
* **Top Level Only**: Hooks should always be called at the top level of the component function to maintain consistent order across renders.
* **Non-Conditional**: Hooks must not be called conditionally to prevent breaking the consistent call order required by React's hooks system.

### Basic Hooks

1. **useState**: Manage state in functional components.

const [state, setState] = useState(initialValue);

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

1. **Purpose**: Perform side effects in functional components (e.g., data fetching, subscriptions).

useEffect(() => {

// effect code

return () => {

// cleanup code

};

}, [dependencies]);

import React, { useEffect, useState } from 'react';

function FetchData() {

const [data, setData] = useState(null);

useEffect(() => {

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => setData(data));

}, []); // Empty array means effect runs once after initial render

return <div>{data ? data : 'Loading...'}</div>;

}

A close-up of a web page

Description automatically generatedA screenshot of a computer

Description automatically generated

useEffect() has one required parameters. It is a callback function to put a side effect.

1. What is a "side effect" in React? What are some examples?

- Any code that affects an outside system.

- local storage, API, websockets, two states to keep in sync

2. What is NOT a "side effect" in React? Examples?

- Anything that React is in charge of.

- Maintaining state, keeping the UI in sync with the data,

  render DOM elements

3. When does React run your useEffect function? When does it NOT run

   the effect function?

- As soon as the component loads (first render)

- On every re-render of the component (assuming no dependencies array)

- Will NOT run the effect when the values of the dependencies in the

  array stay the same between renders

4. How would you explain what the "dependecies array" is?

- Second paramter to the useEffect function

- A way for React to know whether it should re-run the effect function

# Node Express

## Basic Routing

1. Import the http module.
2. Create a server for the app to listen to requests.
   * Get the URL path
   * Set the response header
   * Define routes.

I have been thinking of this since the day after yesterday. I have this girl, a friend of mine, who asked me if it would be okay for me if someone started courting her. Her question threw me for a loop, and I haven't been able to stop thinking about it.

This girl is someone I have been friend. We have shared so much, and she means a lot to me. But here is the thing, I have got feelings here, deep ones, since I have uncrushed my long-time crush. Feelings that go beyond just friendship. But I've always kept them hidden because the thought of losing her, even as just a friend, terrifies me.

When she asked me about the courting stuff, my heart sank. All I wanted to say was, "I wish it were me."

But instead, I told her that it was okay as long as she was happy.