Building a Frontend Application for Your FastAPI Backend

> **Objective:** In this lecture, we will learn how to build a simple and interactive frontend application that communicates with your FastAPI backend serving machine learning models. This session focuses on enabling users to interact with your API through a user-friendly web interface rather than tools like Swagger UI or Postman. You will explore a modern frontend technology called Streamlit, to create forms for uploading files, triggering batch predictions, and displaying results. By the end of this session, you will be able to design and deploy a basic web interface that makes API calls to your FastAPI service, thereby completing the full loop from model prediction to end-user interaction.

Context:

In this lecture, we will build a frontend for interacting with the endpoints we created in the last lecture (`predict` and `predict_batch`) for benchpress limit prediction.

Why Are Frontend Applications Necessary for Serving ML Models to End Users?

While RESTful APIs built with frameworks like FastAPI are powerful tools for exposing machine learning models, they are typically accessed programmatically or via developer tools like Postman. However, end users often lack the technical expertise or patience to interact with APIs directly. This is where frontend applications become essential.

Key Reasons:

1. **Improved User Experience**

    - Frontend applications provide a **graphical interface** (e.g., file upload forms, prediction result displays) that make

  it easy and intuitive for users to interact with complex machine learning services.
- This bridges the gap between **technical APIs** and **non-technical users**.

2. **Accessibility**

   - It allows your model to reach **wider audiences**, including clients, stakeholders, or the general public.

3. **Better Error Handling & Feedback**

   - Instead of raw JSON responses, frontends can show **friendly error messages**, progress indicators, and real-time feedback.
   - This helps users understand what's happening and how to fix input issues.

4. **Branding and Customization**

   - A frontend allows you to incorporate **branding**, colors, logos, and layout tailored to your project or organization.
   - This adds **credibility and professionalism** to your ML solution.

5. **Rapid Prototyping and Demonstrations**

   - Frontend apps make it easy to **demo your model** to others without having them write code.
   - Ideal for **presentations, pitch decks, and testing with early users**.

6. **Integration with Other Frontend Tools**

   - Enables embedding ML functionality into **dashboards, business tools, or client portals.**
   - You can combine your model with **data visualization, analytics, or user management** components.

By building a frontend, you are not just exposing a model. You are delivering a **complete, usable product**. It's the difference between a backend service and a real-world application.

Project Structure and Organization

The folder layout remains the same as in the `ml_fastapi_project_batch` folder. The only addition is a frontend file:

```
streamlit/
│
├── main.py                # Entry point, defines FastAPI
app & routes
```

```
├── models.py           # Pydantic models for
request/response validation
├── predict.py          # Prediction logic and model
interaction
├── utils.py            # Utilities (e.g., loading model,
preprocessing)
├── frontend.py        # Frontend file (streamlit)
│
├── artifacts/
│   └── my_model.pkl         # Serialized ML model using
joblib or pickle
│   └── my_scaler.pkl         # Serialized Scikit-learn
scaler using joblib or
├── data/
│   └── sample_data.csv   # Sample data to use for upload
│
├── test_api.py          # Optional test scripts for
endpoints
│
├── requirements.txt         # All project dependencies
├── README.md                # Project overview and setup
guide
```

What is Streamlit and Why Use It?

Origin & Purpose

Streamlit is an open-source Python framework created specifically to **help data scientists and machine learning engineers** build interactive web applications **without needing full-stack development skills**.

Traditionally, deploying a machine learning model or data visualization tool on the web would require working with **HTML, CSS**, **JavaScript**, and perhaps a backend framework like Flask or Django. For many ML practitioners, this introduces a significant learning curve and development overhead.

Streamlit was created to solve this exact problem—it allows you to turn Python scripts into interactive web apps in just a few lines of code. The goal is to **focus on data, models, and logic**, not web infrastructure.

Key Features

Streamlit's design philosophy is **simplicity, speed, and Python-first development**. Let's break down the features that make it so powerful:

1. **Lightweight, Reactive UI**

- You write UI components directly in Python, and Streamlit automatically tracks changes and updates the app in real-time.
- Streamlit uses a reactive programming model: **when the script runs, it rebuilds the entire app from top to bottom** on each user interaction, which simplifies debugging and state handling.

2. Built-in Widgets for Interactivity

- Streamlit offers a rich set of **pre-built components** like:
  - `st.button`, `st.slider`, `st.selectbox`, `st.file_uploader`, etc.
- These widgets are used to **collect user input** or create interactivity without writing any HTML/JS code.

3. Easy to Deploy and Share

- Streamlit apps can be deployed in just a few clicks via **Streamlit Community Cloud**, or hosted on other platforms using Docker or cloud services like AWS.
- Built-in support for **GitHub integration** makes sharing public projects seamless.

4. Rapid Prototyping

- Ideal for quickly testing ideas or building a front-facing UI for your ML models.

- You can go from a **Jupyter notebook to a web app in under 30 minutes.**

- This is excellent for:
  - Internal tools
  - Demos for stakeholders
  - Experimenting with ML pipelines

▯ Installing and Setting Up Streamlit

- To install streamlit, we sue the command below:

```
pip install streamlit
```

- Let's look at a basic streamlit application:

```python
# base_app.py
import streamlit as st

st.title("My First Streamlit App")
st.write("Hello, world!")
```

- How do we run a python file housing a streamlit application? Instead of the regular `python base_app.py`, we use the following command:

  ```
  streamlit run app.py
  ```

Core Streamlit Components

In **Streamlit**, **widgets** are interactive UI elements that allow users to **input data**, **make selections**, or **trigger actions** directly from the frontend of your app — all defined using **pure Python code.**

Think of widgets as the **“form elements”** of your Streamlit app. They are how users interact with your app and influence the behavior of your logic or models dynamically.

Widgets are at the **core of what makes Streamlit interactive.** They transform a static Python script into a **dynamic, user-driven interface.** Their key roles include:

1. Collecting User Input

Widgets gather input from users, such as:

- Text
- Numbers
- Files
- Option selections
- Dates, times, etc.

This input can then be used to:

- Pass into ML models
- Filter or manipulate datasets
- Call backend APIs (e.g., a FastAPI prediction endpoint)

2. Triggering Actions

Certain widgets like `st.button`, `st.form`, and `st.checkbox` allow users to **control when code executes.**

For example:

- Run a prediction only when a user clicks "Submit"
- Show extra options when a checkbox is selected
- Upload a file and wait for processing

3. Controlling App Flow and Layout

Widgets can conditionally show or hide parts of the app using simple `if` statements.

4. **Reactively Updating the UI**

Whenever a widget’s value changes, Streamlit **re-runs the script from top to bottom**, ensuring the UI is always in sync with user inputs. This reactive model makes it easy to build responsive apps without managing app state manually.

Examples of Common Streamlit Widgets

Widget Type	Streamlit Function	Example Use Case
Text Input	`st.text_input()`	Enter a name or comment
File Uploader	`st.file_uploader()`	Upload CSV/Excel files for prediction
Button	`st.button()`	Submit or run a task
Title/Headers	`st.title()`, `st.header()`	Structure and navigation
Number Input	`st.number_input()`	Set numeric parameters like thresholds
Slider	`st.slider()`	Adjust model parameters (e.g., alpha)
Checkbox	`st.checkbox()`	Toggle advanced options
Select Box	`st.selectbox()`	Choose from a list of options
Radio Buttons	`st.radio()`	Select one model from a group
Date Input	`st.date_input()`	Pick a date for filtering data
Data Display	`st.dataframe(df)`, `st.table()`	Show results

Streamlit vs Alternatives

Feature	Streamlit	Dash (Plotly)	Flask + HTML/CSS
Target Audience	ML & data science practitioners	Data science & analytics teams	Web developers
Language	Python only	Python only	Python (backend) + HTML/CSS/JS (frontend)
Ease of Use	Very easy	Moderate	Steep (requires frontend knowledge)
Custom UI	Limited but fast	Flexible but complex	Fully customizable
Widgets & Interactivity	Built-in, easy to use	Rich, customizable	Requires JavaScript
Learning Curve	Very low	Moderate	High
Deployment Options	Streamlit Cloud, Docker, cloud VM	Heroku, Dash Enterprise, etc.	Heroku, AWS, Azure, etc.

Feature	Streamlit	Dash (Plotly)	Flask + HTML/CSS
Reactive Programming	Yes (auto reruns script)	Yes (callback-based)	Manual (JS/AJAX integration)
Use Cases	ML demos, dashboards, simple apps	Complex dashboards, BI tools	Full web apps, complex UIs

Why Integrate Streamlit with FastAPI?

While Streamlit is great for user interaction and UI, it is not designed to be a backend server for scalable machine learning inference.

FastAPI, on the other hand:

- Is optimized for handling API requests
- Supports asynchronous I/O and background tasks
- Can easily serve pre-trained ML models

By combining them:

- Streamlit handles the frontend (presentation & user input)
- FastAPI handles the backend (data processing & model predictions)

Building Interactivity and Layouts for our FastAPI application

As at the last lecture, our fastapi app was only accessible via swaggerUI. Here, we will focus on building an interface to communicate with the fastapi application.

Step 1: Adding a Title for our Application

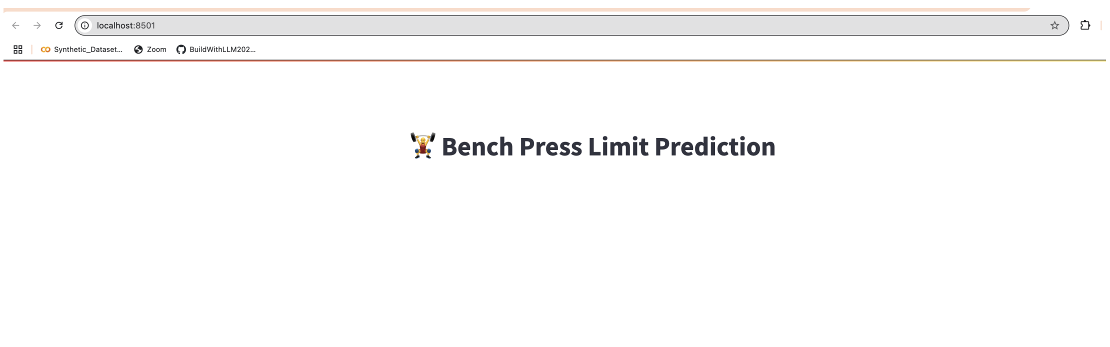
We will start by adding a title for our application using the streamlit `st.title` function.

```
# frontend.py

import streamlit as st
from PIL import Image
import requests

# Title of the app
st.title("♂ Bench Press Limit Prediction")
```

Then when we run the streamlit command:

```
streamlit run frontend.py
```

Output: Notice that our app is still run locally on localhost. Once completed, we will deploy it on a live server.



Step 2: Adding a Sidebar to Our Application

In our FastAPI backend, we implemented two key functionalities:

- **Single-point prediction** (predicting for one data point)
- **Batch prediction** (predicting for multiple data points via file upload)

Wouldn't it be great if users could easily switch between these two options in our frontend? This is where a **sidebar** becomes very useful.

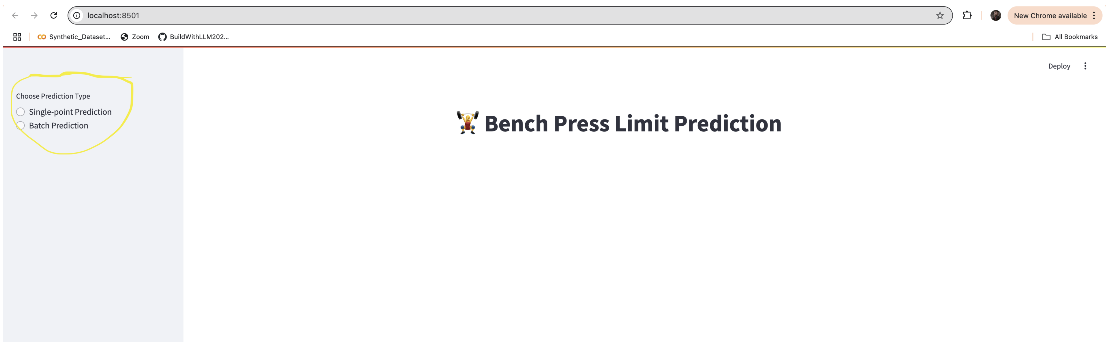
In **Streamlit**, the `st.sidebar` module allows us to add UI elements to a fixed panel on the left side of the app. This is ideal for navigation or global controls.

To let users choose between single or batch prediction, we will use the `st.sidebar.radio()` method. This widget displays a list of options as **radio buttons**, and returns the selected value. Based on the user's choice, we can then show the corresponding page or input form in the main area of the app.

Action:

- Use `st.sidebar.radio()` to create a sidebar menu
- Display **"Single Prediction"** and **"Batch Prediction"** as options
- Use the user's selection to control what is displayed in the main app

```python
import streamlit as st
from PIL import Image
import requests

# Title of the app
st.title("🏋️‍♂️ Bench Press Limit Prediction")

# Sidebar for selecting prediction type (no default selected)
prediction_type = st.sidebar.radio("Choose Prediction Type",
                                   ["Single-point Prediction",
"Batch Prediction"],
                                  index=None)  # index=0 to
ensure the first item is blank
```
Output:



Choose Prediction Type
- Single-point Prediction
- Batch Prediction

🏋‍♂ Bench Press Limit Prediction

Step 3: Adding a Home Page to Our Application

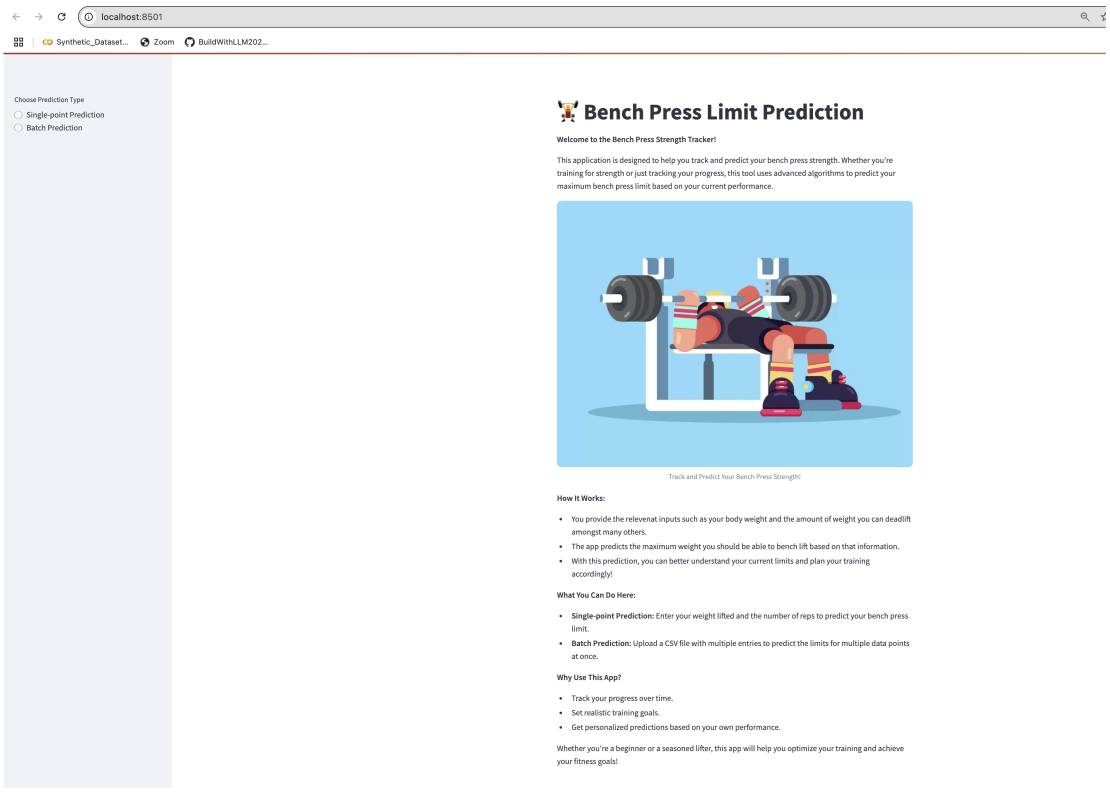
In the previous step, we added a sidebar with two options: "Single-point Prediction" and "Batch Prediction". But what happens if the user doesn’t select either option?

To handle this scenario gracefully, we will create a Home Page as the default view. This ensures that when no prediction type is selected, the user is still presented with meaningful content or guidance.

Remember we already achieved this by setting the default value of the `st.sidebar.radio()` widget to None. So next, we will use conditional logic to display the Home Page contents when no prediction path has been chosen.

- we will display:
  - A markdown Text: (The text content of the webpage should always be written as markdown in a python string and passed to the `st.markdown` function.
  - An Image from the internet which has been saved in the `artifacts` folder. Images can be displayed using the `st.image` function. The `use_container_width` parameter enables properly display.

```
#frontend.py

import streamlit as st
from PIL import Image
import requests

# Title of the app
st.title("🏋‍♂ Bench Press Limit Prediction")

# Sidebar for selecting prediction type (no default selected)
prediction_type = st.sidebar.radio("Choose Prediction Type",
                                  ["Single-point Prediction",
"Batch Prediction"],
                                    index=None)  # index=0 to
ensure the first item is blank

# Home Page (Welcome Message) - Displayed if no prediction type
is selected
if prediction_type == None:
    # Display the introduction page when no prediction type is
selected
    st.markdown("""
        **Welcome to the Bench Press Strength Tracker!**

        This application is designed to help you track and
predict your bench press strength. Whether you're training for
strength or just tracking your progress, this tool uses advanced
algorithms to predict your maximum bench press limit based on
your current performance.
    """)

    image = Image.open("artifacts/benchpress.webp")  # Make sure
to put the correct path to your image
    st.image(image, caption="Track and Predict Your Bench Press
Strength!", use_container_width=True)

    st.markdown("""
        **How It Works:**
        - You provide the relevenat inputs such as your body
weight and the amount of weight you can deadlift amongst many
others.
        - The app predicts the maximum weight you should be able
to bench lift based on that information.
        - With this prediction, you can better understand your
current limits and plan your training accordingly!

        **What You Can Do Here:**
        - **Single-point Prediction:** Enter your weight lifted
and the number of reps to predict your bench press limit.
        - **Batch Prediction:** Upload a CSV file with multiple
entries to predict the limits for multiple data points at once.

        **Why Use This App?**
        - Track your progress over time.
        - Set realistic training goals.
        - Get personalized predictions based on your own
performance.

        Whether you're a beginner or a seasoned lifter, this app
will help you optimize your training and achieve your fitness
goals!
    """)
```
Output



🏋️ **Bench Press Limit Prediction**

Welcome to the Bench Press Strength Tracker!

This application is designed to help you track and predict your bench press strength. Whether you're training for strength or just tracking your progress, this tool uses advanced algorithms to predict your maximum bench press limit based on your current performance.

Track and Predict Your Bench Press Strength!

How It Works:

- You provide the relevenat inputs such as your body weight and the amount of weight you can deadlift amongst many others.
- The app predicts the maximum weight you should be able to bench lift based on that information.
- With this prediction, you can better understand your current limits and plan your training accordingly!

What You Can Do Here:

- **Single-point Prediction:** Enter your weight lifted and the number of reps to predict your bench press limit.
- **Batch Prediction:** Upload a CSV file with multiple entries to predict the limits for multiple data points at once.

Why Use This App?

- Track your progress over time.
- Set realistic training goals.
- Get personalized predictions based on your own performance.

Whether you're a beginner or a seasoned lifter, this app will help you optimize your training and achieve your fitness goals!

Step 4: Building the Single-point Prediction Page

Once the user selects **"Single-point Prediction"** from the sidebar, the main area of the app should display a new section with the relevant input fields required for making a single prediction.

To build this interface, we need to replicate all the inputs expected by the **`/predict` endpoint** in our FastAPI backend.

Remember Pseudo-features? These are the **individual parameters** required in the payload for prediction. Every feature that your FastAPI model expects must be represented with a corresponding input widget in Streamlit.

1. Model Type

- The user needs to select which model to use.
- For this, we'll use a **dropdown menu** via the `st.selectbox` widget.

  ```
  model_type = st.selectbox("Choose a model type", ["model_A",
  "model_B", "model_C"])
  ```

2. Categorical Features (e.g., Sex)

- Categorical fields like "Sex" are best handled with dropdowns to limit input options and avoid typos.

- Again, we’ll use `st.selectbox` here.

  ```
  sex = st.selectbox("Sex", ["Male", "Female", "Other"])
  ```

3. **Numerical Features (e.g., Age, Best Deadlift, Bodyweight)**

- For numeric inputs, we’ll use `st.number_input`, which restricts user entries to numbers only.
- We can also define:
  - **Minimum value**
  - **Maximum value** (optional)
  - **Default value**

Example:

```
bodyweight_kg = st.number_input("Bodyweight (kg)", min_value=0.0,
value=70.0)
```
This creates a field labeled **"Bodyweight (kg):"** with:

- A minimum allowable value of `0.0`
- A default (pre-filled) value of `70.0`

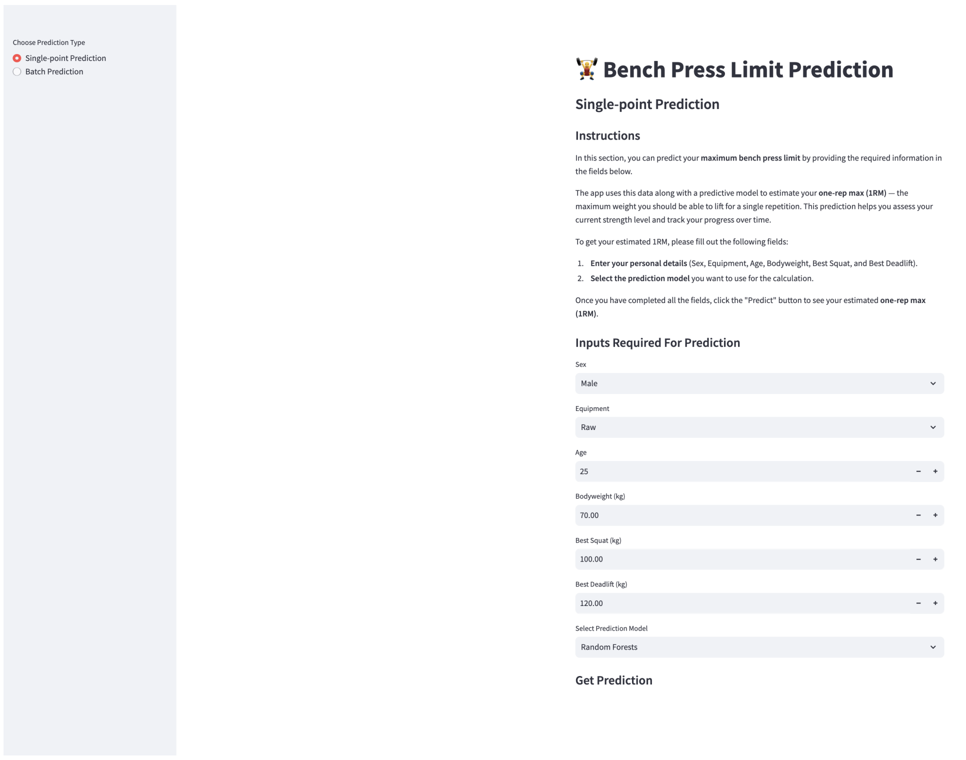
You can use similar widgets for fields like **age**, **deadlift**, or **squat total**, adjusting ranges and defaults as needed.

```
#frontend.py

import streamlit as st
from PIL import Image
import requests

# Title of the app
st.title("🏋️‍♂️ Bench Press Limit Prediction")

# Sidebar for selecting prediction type (no default selected)
prediction_type = st.sidebar.radio("Choose Prediction Type",
                                  ["Single-point Prediction",
"Batch Prediction"],
                                  index=None)  # index=0 to
ensure the first item is blank

# Home Page (Welcome Message) - Displayed if no prediction type
is selected
if prediction_type == None:
    # Display the introduction page when no prediction type is
selected
    st.markdown("""
        **Welcome to the Bench Press Strength Tracker!**

        This application is designed to help you track and
predict your bench press strength. Whether you're training for
strength or just tracking your progress, this tool uses advanced
```

```
algorithms to predict your maximum bench press limit based on
your current performance.
    """)

    image = Image.open("artifacts/benchpress.webp")  # Make sure
to put the correct path to your image
    st.image(image, caption="Track and Predict Your Bench Press
Strength!", use_container_width=True)

    st.markdown("""
        **How It Works:**
        - You provide the relevenat inputs such as your body
weight and the amount of weight you can deadlift amongst many
others.
        - The app predicts the maximum weight you should be able
to bench lift based on that information.
        - With this prediction, you can better understand your
current limits and plan your training accordingly!

        **What You Can Do Here:**
        - **Single-point Prediction:** Enter your weight lifted
and the number of reps to predict your bench press limit.
        - **Batch Prediction:** Upload a CSV file with multiple
entries to predict the limits for multiple data points at once.

        **Why Use This App?**
        - Track your progress over time.
        - Set realistic training goals.
        - Get personalized predictions based on your own
performance.

        Whether you're a beginner or a seasoned lifter, this app
will help you optimize your training and achieve your fitness
goals!
    """)

# When user selects Single-point Prediction
elif prediction_type == "Single-point Prediction":
    st.markdown("### Single-point Prediction")
    st.markdown("""
    #### Instructions
    In this section, you can predict your **maximum bench press
limit** by providing the required information in the fields
below.

    The app uses this data along with a predictive model to
estimate your **one-rep max (1RM)** — the maximum weight you
should be able to lift for a single repetition. This prediction
helps you assess your current strength level and track your
progress over time.

    To get your estimated 1RM, please fill out the following
fields:
    1. **Enter your personal details** (Sex, Equipment, Age,
Bodyweight, Best Squat, and Best Deadlift).
    2. **Select the prediction model** you want to use for the
```

```
calculation.

    Once you have completed all the fields, click the "Predict"
button to see your estimated **one-rep max (1RM)**.
    """)

    st.markdown("#### Inputs Required For Prediction")

    # Input fields for personal data (to match the Pydantic
model)
    sex = st.selectbox("Sex", options=["Male", "Female"],
index=0)
    equipment = st.selectbox("Equipment", options=["Raw",
"Wraps", "Single-ply", "Multi-ply"], index=0)
    age = st.number_input("Age", min_value=18, max_value=100,
value=25)
    bodyweight_kg = st.number_input("Bodyweight (kg)",
min_value=0.0, value=70.0)
    best_squat_kg = st.number_input("Best Squat (kg)",
min_value=0.0, value=100.0)
    best_deadlift_kg = st.number_input("Best Deadlift (kg)",
min_value=0.0, value=120.0)

    # Dropdown to select the prediction model
    model_option = st.selectbox(
        "Select Prediction Model",
        options=["Random Forests", "Decision Trees", "Gradient
Boosting"],
        index=0
    )

    st.markdown("#### Get Prediction")
```

Output:



Choose Prediction Type

[✓ Single-point Prediction]
Batch Prediction

🏋️ **Bench Press Limit Prediction**

Single-point Prediction

Instructions

In this section, you can predict your **maximum bench press limit** by providing the required information in the fields below.

The app uses this data along with a predictive model to estimate your **one-rep max (1RM)** — the maximum weight you should be able to lift for a single repetition. This prediction helps you assess your current strength level and track your progress over time.

To get your estimated 1RM, please fill out the following fields:

1. **Enter your personal details** (Sex, Equipment, Age, Bodyweight, Best Squat, and Best Deadlift).
2. **Select the prediction model** you want to use for the calculation.

Once you have completed all the fields, click the "Predict" button to see your estimated **one-rep max (1RM)**.

Inputs Required For Prediction

Sex: Male

Equipment: Raw

Age: 25

Bodyweight (kg): 70.00

Best Squat (kg): 100.00

Best Deadlift (kg): 120.00

Select Prediction Model: Random Forests

Get Prediction

Step 5: Prediction Logic and Trigger for Single-Point Prediction:

Now, we connect the frontend with the backend. Remember the expected payload for the "/predict" endpoint?

It looked like this:

```
{
    "Features": {
        "Sex": Male,
        "Equipment": Wraps,
        "Age": 45,
        "BodyweightKg": 130,
        "BestSquatKg": 60,
        "BestDeadliftKg": 110
    },
    "model": Random Forests
}
```

We must ensure that the input collected from the user via streamlit widget are also structured in this nature before passing into the endpoint.

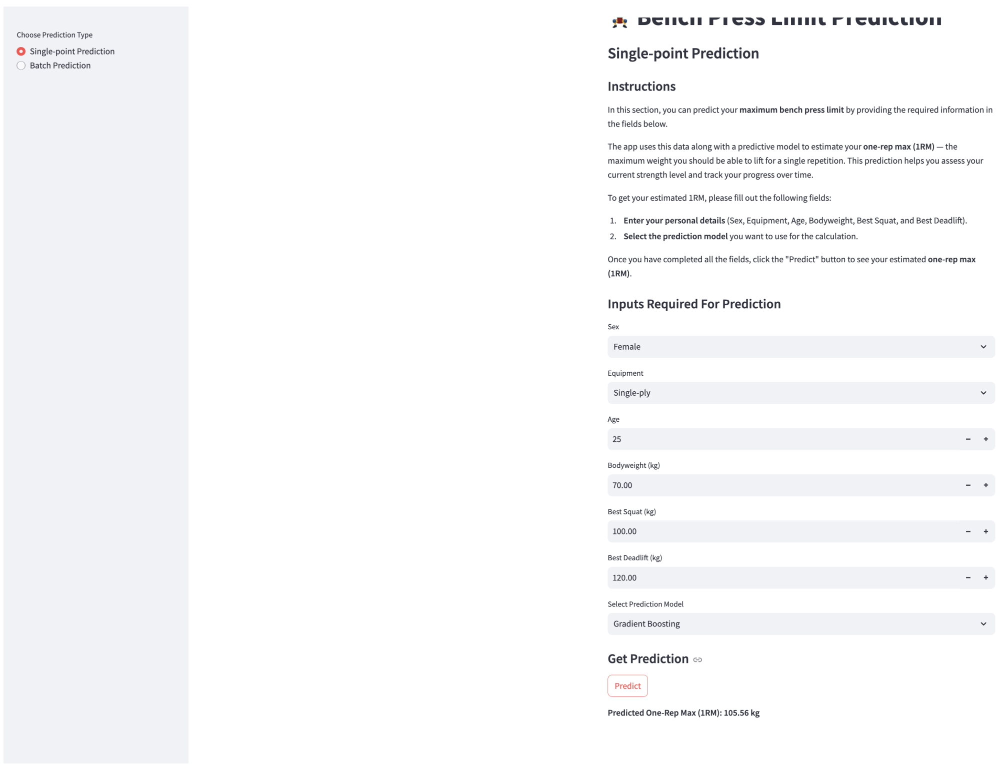
At this point, you need to have a running instance of the backend on localhost.

The api_url will be: "http://0.0.0.0:8000/predict". This is because, the backend is running locally on port 8000 and we are interested in the "/predict" endpoint.

We must always check the response code. If it is 200, then we got a good response, which must then be format for display. `st.write` is used to write into the screen for display.

```
#frontend.py
.
.
.
    st.markdown("#### Get Prediction")

    if st.button("Predict"):
        if best_deadlift_kg > 0 and best_squat_kg > 0:
            # Prepare the payload to send to FastAPI
            payload = {
                "Features": {
                    "Sex": sex,
                    "Equipment": equipment,
                    "Age": age,
                    "BodyweightKg": bodyweight_kg,
                    "BestSquatKg": best_squat_kg,
                    "BestDeadliftKg": best_deadlift_kg
                },
                "model": model_option  # Pass the selected model
type
            }

            # FastAPI endpoint URL for single-point prediction
            api_url = "http://0.0.0.0:8000/predict"

            # Make the POST request to FastAPI
            response = requests.post(api_url, json=payload)

            if response.status_code == 200:
                prediction = response.json().get("prediction",
"Error in prediction")
                st.write(f"**Predicted One-Rep Max (1RM):
{prediction:.2f} kg**")
            else:
                st.write("Error in prediction request.")
        else:
            st.write("Please provide valid inputs for both
best_deadlift_kg and best_squat_kg.")
```

Output:



Single-point Prediction

Instructions

In this section, you can predict your **maximum bench press limit** by providing the required information in the fields below.

The app uses this data along with a predictive model to estimate your **one-rep max (1RM)** — the maximum weight you should be able to lift for a single repetition. This prediction helps you assess your current strength level and track your progress over time.

To get your estimated 1RM, please fill out the following fields:

1. **Enter your personal details** (Sex, Equipment, Age, Bodyweight, Best Squat, and Best Deadlift).
2. **Select the prediction model** you want to use for the calculation.

Once you have completed all the fields, click the "Predict" button to see your estimated **one-rep max (1RM)**.

Inputs Required For Prediction

Sex: Female
Equipment: Single-ply
Age: 25
Bodyweight (kg): 70.00
Best Squat (kg): 100.00
Best Deadlift (kg): 120.00
Select Prediction Model: Gradient Boosting

Get Prediction

Predicted One-Rep Max (1RM): 105.56 kg

Step 6: Adding a Batch-Prediction Page to Our Application

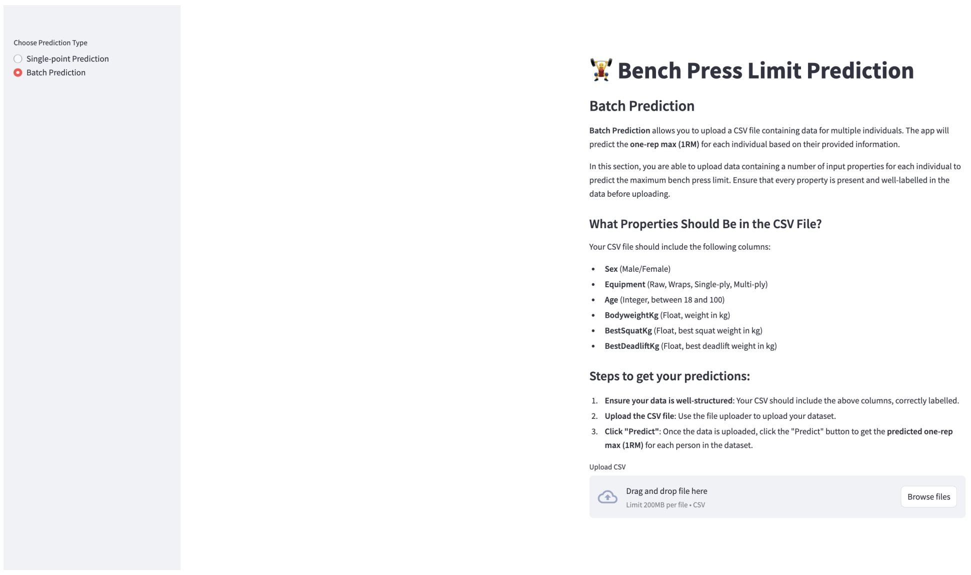
Now, we have to define what happens, when you pick "Batch Prediction" on the side bar. Similar to "Single-point Prediction", we need page content here as well. We will used markdown for the text components.

Remember the goal here is to be able to upload a dataset. `st.file_uploader` empowers us to be able to do that. We specify the title of the widget and the file type permitted for upload (CSV).

Once the file has been updated, uploaded_file is no longer `None`. Therefore, we should be able to show a preview of the recently uploaded file with `st.write`.

```python
#frontend.py
.
.
.

# When user selects Batch Prediction
elif prediction_type == "Batch Prediction":
    st.markdown("### Batch Prediction")
    st.markdown("""
    **Batch Prediction** allows you to upload a CSV file
containing data for multiple individuals.
    The app will predict the **one-rep max (1RM)** for each
individual based on their provided information.

    In this section, you are able to upload data containing a
number of input properties for each individual to predict the
maximum bench press limit. Ensure that every property is present
and well-labelled in the data before uploading.

    #### What Properties Should Be in the CSV File?
    Your CSV file should include the following columns:
    - **Sex** (Male/Female)
    - **Equipment** (Raw, Wraps, Single-ply, Multi-ply)
    - **Age** (Integer, between 18 and 100)
    - **BodyweightKg** (Float, weight in kg)
    - **BestSquatKg** (Float, best squat weight in kg)
    - **BestDeadliftKg** (Float, best deadlift weight in kg)

    #### Steps to get your predictions:
    1. **Ensure your data is well-structured**: Your CSV should
include the above columns, correctly labelled.
    2. **Upload the CSV file**: Use the file uploader to upload
your dataset.
    3. **Click "Predict"**: Once the data is uploaded, click the
"Predict" button to get the **predicted one-rep max (1RM)** for
each person in the dataset.
    """)

    uploaded_file = st.file_uploader("Upload CSV", type="csv")

    if uploaded_file is not None:
        # Process the uploaded CSV file
        import pandas as pd
        df = pd.read_csv(uploaded_file)

        # Display the first few rows of the data
        st.write("### Uploaded Data Preview")
        st.write(df.head())
        st.write("### Select a Prediction Model")
        # Dropdown to select the prediction model
        model_option = st.selectbox(
            "Select Prediction Model",
            options=["Random Forests", "Decision Trees",
"Gradient Boosting"],
            index=0
        )
```

Output 1:



Choose Prediction Type

Single-point Prediction [✓ Batch Prediction]

Bench Press Limit Prediction

Batch Prediction

Batch Prediction allows you to upload a CSV file containing data for multiple individuals. The app will predict the **one-rep max (1RM)** for each individual based on their provided information.

In this section, you are able to upload data containing a number of input properties for each individual to predict the maximum bench press limit. Ensure that every property is present and well-labelled in the data before uploading.

What Properties Should Be in the CSV File?

Your CSV file should include the following columns:

- **Sex** (Male/Female)
- **Equipment** (Raw, Wraps, Single-ply, Multi-ply)
- **Age** (Integer, between 18 and 100)
- **BodyweightKg** (Float, weight in kg)
- **BestSquatKg** (Float, best squat weight in kg)
- **BestDeadliftKg** (Float, best deadlift weight in kg)

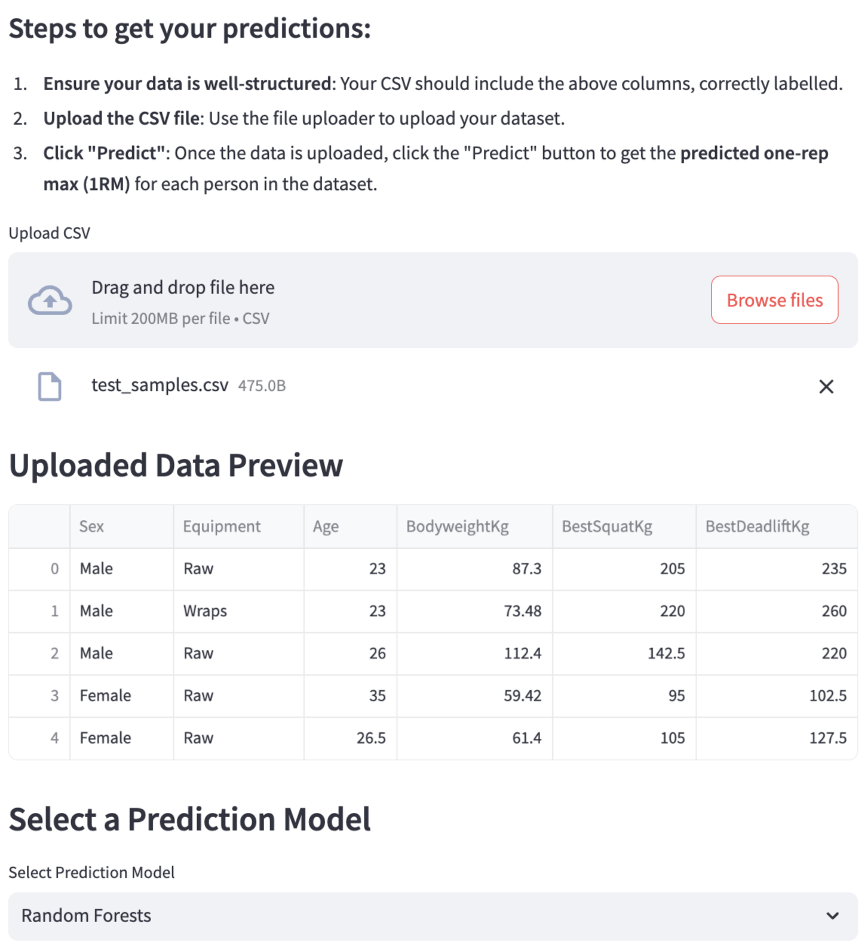
Steps to get your predictions:

1. **Ensure your data is well-structured**: Your CSV should include the above columns, correctly labelled.
2. **Upload the CSV file**: Use the file uploader to upload your dataset.
3. **Click "Predict"**: Once the data is uploaded, click the "Predict" button to get the **predicted one-rep max (1RM)** for each person in the dataset.

Upload CSV

After uploading the sample data, we see that a subset of the upload file is shown Then we are able to select a prediction model.

Output 2:



Steps to get your predictions:

1. **Ensure your data is well-structured**: Your CSV should include the above columns, correctly labelled.
2. **Upload the CSV file**: Use the file uploader to upload your dataset.
3. **Click "Predict"**: Once the data is uploaded, click the "Predict" button to get the **predicted one-rep max (1RM)** for each person in the dataset.

Upload CSV

Drag and drop file here
Limit 200MB per file • CSV

test_samples.csv 475.0B

Uploaded Data Preview

	Sex	Equipment	Age	BodyweightKg	BestSquatKg	BestDeadliftKg
0	Male	Raw	23	87.3	205	235
1	Male	Wraps	23	73.48	220	260
2	Male	Raw	26	112.4	142.5	220
3	Female	Raw	35	59.42	95	102.5
4	Female	Raw	26.5	61.4	105	127.5

Select a Prediction Model

Select Prediction Model

Random Forests

Step 7: Adding a Prediction Logic and Trigger for Batch-Prediction

Likewise, we need to connect this page to the "/predict_batch" endpoint. The trigger her is the :"predict" button. once clicked, we must send a payload containing the file and the model selected to the endpoint.

Do you remember the required payload for the "/predict_batch" endpoint?

```
@app.post("/predict_batch")
async def predict_batch_endpoint(file: UploadFile = File(...),
model: ModelType = ModelType.rf):
```

This means that model is expected to be passed as a query parameter. We will use the request library to send our input to the backend. `uploaded_file.getvalue()` ensures every metadata related to the uploaded file is captured. The model type will be sent using the data parameter.

We will extract the predictions and then display the output file containing the predictions with `st.write`.

```
#frontend.py
.
.
.
    uploaded_file = st.file_uploader("Upload CSV", type="csv")

    if uploaded_file is not None:
        # Process the uploaded CSV file
        import pandas as pd
        df = pd.read_csv(uploaded_file)

        # Display the first few rows of the data
        st.write("### Uploaded Data Preview")
        st.write(df.head())

        st.write("### Select a Prediction Model")
        # Dropdown to select the prediction model
        model_option = st.selectbox(
            "Select Prediction Model",
            options=["Random Forests", "Decision Trees",
"Gradient Boosting"],
            index=0
        )
        # FastAPI endpoint URL for batch prediction
        api_url = "http://0.0.0.0:8000/predict_batch"

        # Button to trigger batch prediction
        if st.button("Predict"):
            # Prepare the payload for batch prediction (file and
model)

            # Send the request to FastAPI
            response = requests.post(api_url, files={"file":
uploaded_file.getvalue()}, data={"model": model_option})

            # Check if the response is successful
            if response.status_code == 200:
                predictions = response.json().get("predictions",
[])
                df['predicted_limit'] = predictions
                st.write("### Prediction Results")
                st.write(df)  # Display the updated DataFrame
with predictions
            else:
                st.write(f"Error in batch prediction request:
{response.status_code}")
```

Next Steps: Deploying both the Streamlit Frontend And FastAPI Backend Applications