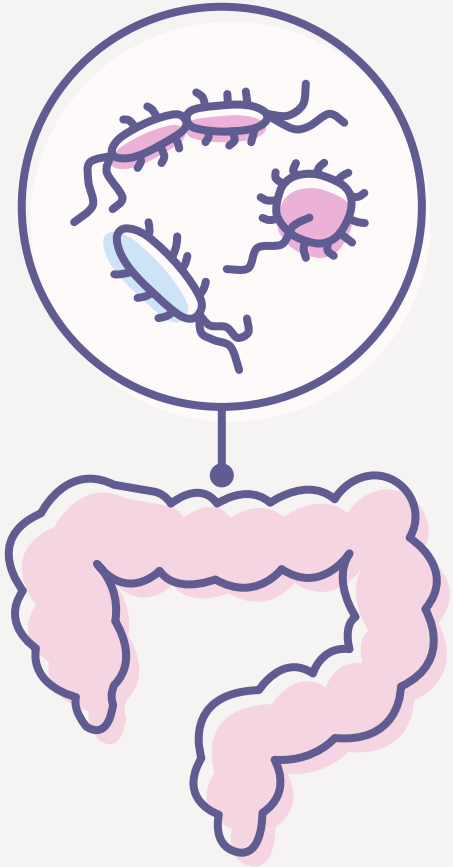


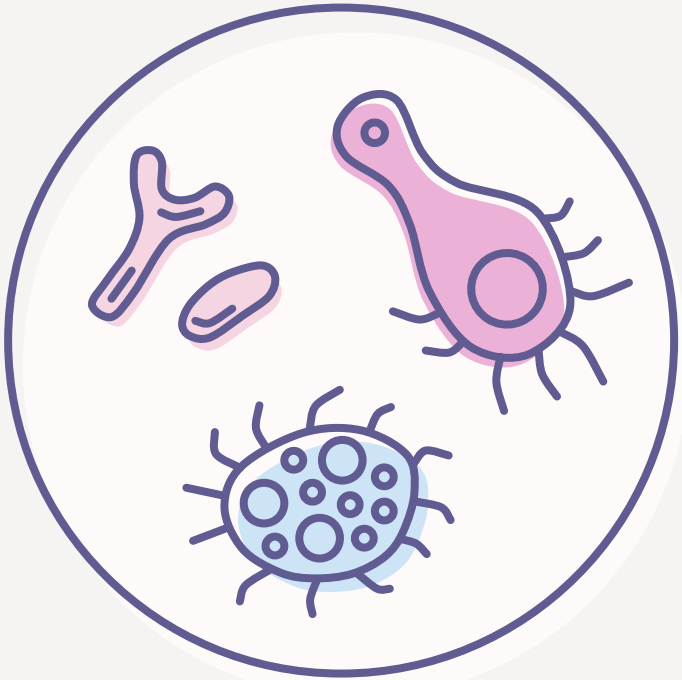
Analysis of Gut Microbiome Dynamics Following Prebiotic Supplementation

Universidade do Minho
Escola de Engenharia

Project in Bioinformatics

Vanessa Rodriguez | PG49131

Supervisors:
Clarisse Nobre
Andreia Salvador

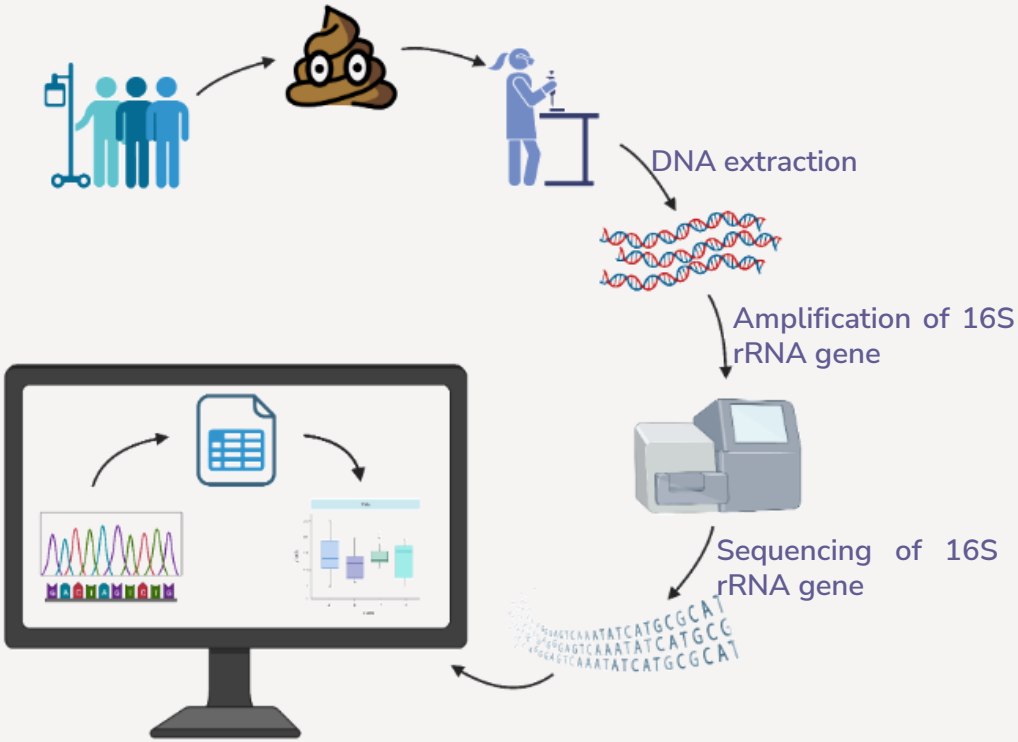


Introduction

Human gut microbiota is a complex ecosystem of trillions of microorganisms that play essential roles in digestion, immunity, and overall health.

Prebiotics are a substrate that is selectively utilized by host microorganisms conferring a health benefit (Ex: fructo-oligosaccharides)

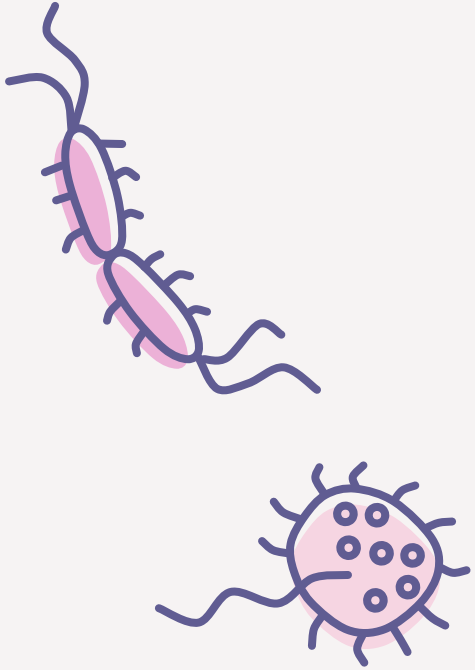
Introduction



Data file containing taxonomic profiling and relative abundance

Why certain bacterial taxa thrive after administration with prebiotics? What is their function?

Aims



Develop a pipeline that identifies the major differences in microbiota composition between treatments, and that predicts the function of the most relevant microorganisms in the microbiota

Methodology

$$\text{Normalized abundance} = \frac{\text{Relative abundance}}{\text{Copy number of 16S rRNA gene}}$$

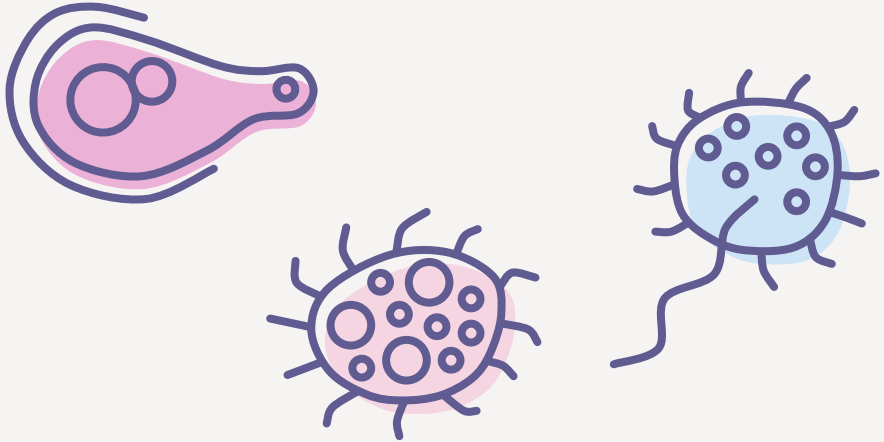
01 **Normalize 16S rRNA gene copy number**

Statistical Analysis using MicrobiomeAnalyst

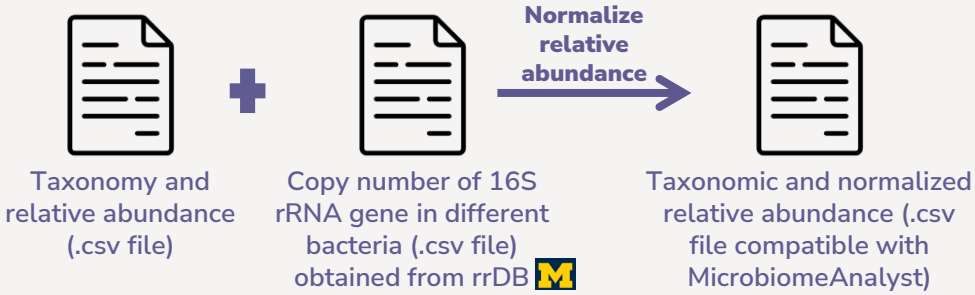
02 **Selection of relevant taxa**

03 **Functional Analysis**

01



Normalize 16S rRNA copy number



Klappenbach JA, Saxman PR, Cole JR, Schmidt TM. rrndb: the Ribosomal RNA Operon Copy Number Database. Nucleic Acids Res. 2001 Jan 1;29(1):181-4. doi: 10.1093/nar/29.1.181. PMID: 11125085; PMCID: PMC29826.

Normalize 16S Copy Number

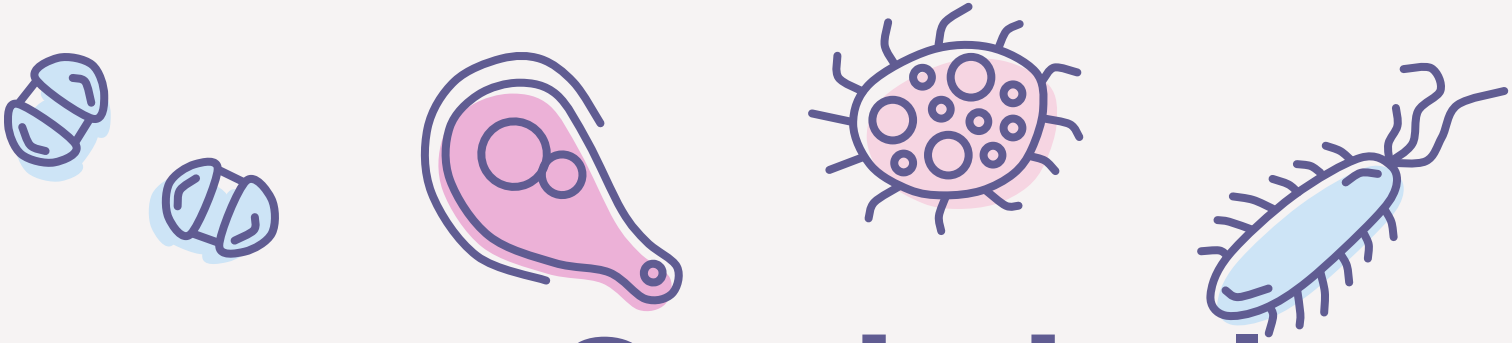
```python
# === Read the original CSV file  ===
with open("cellulose_data.csv", 'r', encoding='utf-8') as f:
    lines = f.readlines()

# Extract #NAME and #CLASS
name_line = lines[0].strip().split(",")
class_line = lines[1].strip().split(",")
name_line[0] = "Taxon"
class_line[0] = "Group"

# Header and data
header = name_line
data = [line.strip().split(",") for line in lines[2:]]
df = pd.DataFrame(data, columns=header)

# Transform columns to float
sample_cols = header[1:]
df[sample_cols] = df[sample_cols].apply(pd.to_numeric, errors='coerce')

# === Normalize using 16S copy numbers ===
copy_df = pd.read_csv("16S_copy_numbers.csv")
copy_dict = dict(zip(copy_df['Taxon'], copy_df['CopyNumber']))

def find_copy_number(taxon_string):
    taxa = taxon_string.split(";")[::-1]
    for t in taxa:
        if t.strip() in copy_dict:
            return copy_dict[t.strip()]
    return np.nan

df["CopyNumber"] = df["Taxon"].apply(find_copy_number)
corrected = df[sample_cols].div(df["CopyNumber"], axis=0)

# Export normalized matrix
final_df = pd.concat([df[["Taxon"]], corrected], axis=1)
final_df.to_csv("matrix_normalized_microbiomeanalyst.csv", index=False)

# Remove "Taxon" from the header and add #NAME and #CLASS
name_line_out = name_line[1:]
class_line_out = class_line[1:]

# Manually write two lines + matrix without header
with open("matrix_normalized_microbiomeanalyst.csv", "w", encoding="utf-8") as f
    f.write("#NAME," + ",".join(name_line_out) + "\n")
    f.write("#CLASS," + ",".join(class_line_out) + "\n")
    final_df.to_csv(f, index=False, header=False)
```

#NAME	Donor.A.CTR.24h
Bacteria;Firmicutes;Bacilli;Lactobacillales;Lactobacillaceae;Ligilactobacillus	0.3988932151526945

Taxon	CopyNumber
Ligilactobacillus	7

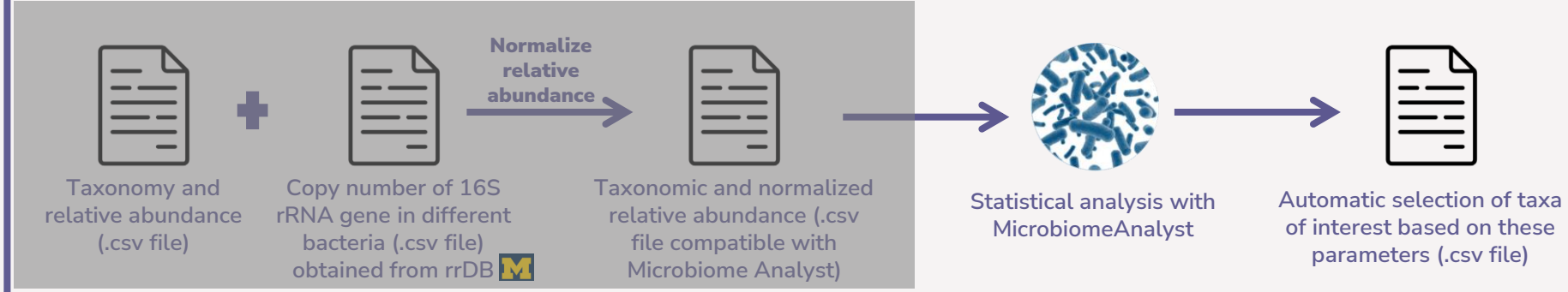
$$\text{Normalized abundance} = \frac{\text{Relative abundance}}{\text{Copy number of 16S rRNA gene}}$$

#NAME	Donor.A.CTR.24h
Bacteria;Firmicutes;Bacilli;Lactobacillales;Lactobacillaceae;Ligilactobacillus	0.0569847450218135



02

Statistical Analysis & Selection of Microorganisms



P-value	Identifies taxa with statistically signficant abundance differences
LDA Score	Measures the effect the treatment has
Mean Abundance	Ensures that the taxa is abundant
Core in Group	Ensures consistency, meaning that taxa are not only present in one or two samples

Lu Y, Zhou G, Ewald J, Pang Z, Shiri T, Xia J. 2023. MicrobiomeAnalyst 2.0: comprehen sive statistical, functional and integrative analysis of microbiome data. Nucleic Acids Re search. 51(1):310–18

Selection of microorganisms

Identify bacterial taxa that show:

P-value -> statistically significant changes in abundance

LDA Score -> biologically meaningful effect sizes,

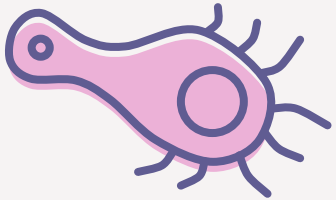
Mean Abundance -> relatively abundant in the microbial community

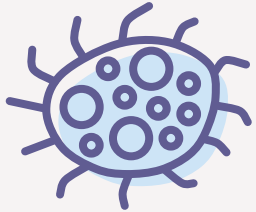
Core Microbiome -> consistently present across a relevant proportion of samples

File (.csv) with taxa meeting all selection criteria

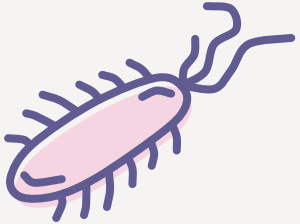
File (.csv) with the taxa that do not meet at least one criterion

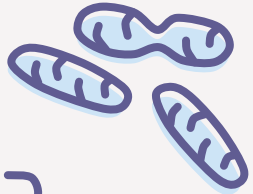
```python
# Load the input files
df_univar = pd.read_csv("univar_test_output.csv")
df_abund = pd.read_csv("taxa_abund.csv")
df_lefse = pd.read_csv("lefse_de_output.csv")
df_core = pd.read_csv("core_microbiome.csv")

# Standardize and rename columns for clarity and consistency
df_univar.rename(columns={df_univar.columns[0]: "Taxon", "P.Value": "P-value"}, inplace=True)
df_abund.rename(columns={df_abund.columns[0]: "Taxon", "Mean" : "Mean Abundance (%)"}, inplace=True)
df_lefse.rename(columns={df_lefse.columns[0]: "Taxon", "LDA": "LDA Score"}, inplace=True)
df_core.rename(columns={df_core.columns[0]: "Taxon", "Prevelance": "Core Microbiome"}, inplace=True)

# Merge all dataframes on 'Taxon'
df_merged = df_univar.merge(df_abund[["Taxon", "Mean Abundance (%)"]], on="Taxon", how="inner")
df_merged = df_merged.merge(df_lefse[["Taxon", "LDAscore"]], on="Taxon", how="inner")
df_merged = df_merged.merge(df_core[["Taxon", "Core Microbiome"]], on="Taxon", how="left")

# Save the final merged table
df_merged.to_csv("bacteria_analysis_summary.csv", index=False)

print("✅ Summary file saved as: bacteria_analysis_summary.csv")

# Load your combined table
df = pd.read_csv("bacteria_analysis_summary.csv")

# Define thresholds
lda_cutoff = 1
pval_cutoff = 0.05
abundance_cutoff = 0.001
core_cutoff = 0.1

# Filter conditions
selected = df[
    (df["LDAscore"] >= lda_cutoff) &
    (df["Pvalues"] <= pval_cutoff) &
    (df["Mean Abundance (%)"] >= abundance_cutoff) &
    (df["Core Microbiome"] >= core_cutoff)
]

# Save outputs
selected.to_csv("selected_taxa.csv", index=False)

# Save excluded taxa
excluded = df[~df["Taxon"].isin(selected["Taxon"])]
excluded.to_csv("excluded_taxa.csv", index=False)
```



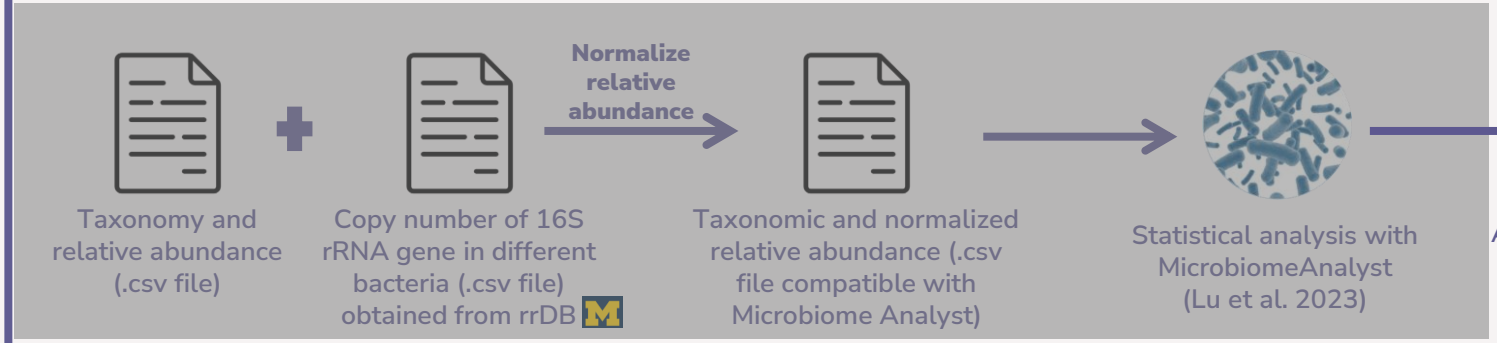


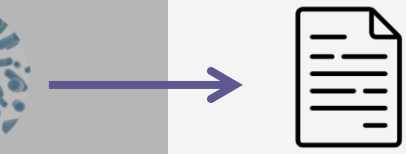
03





Functional Analysis





Automatic selection of taxa of interest based on these parameters (.csv file)

Extraction of EC numbers associated with the selected bacterial taxa

Compatibility Matrix (table with common EC numbers and respective taxonomic assignment)

Extraction of EC numbers associated to the selected metabolic pathways

Input by the user: KEGG map ID (selection of metabolic pathways of interest: https://www.genome.jp/kegg/pathway.html)

Identification of microorganisms and respective enzymes that can potentially be involved in the metabolic pathways of interest(.csv file)

Kanehisa M, Goto S. KEGG: kyoto encyclopedia of genes and genomes. Nucleic Acids Res. 2000 Jan 1;28(1):27-30. doi: 10.1093/nar/28.1.27. PMID: 10592173; PMCID: PMC102409.

```python
def read_pathway_ids(file_path):
    with open(file_path, 'r') as f:
        return [line.strip() for line in f if line.strip()]

def get_ecs_for_pathway(pathway_id):
    """Tries to obtain ECs directly, if it does not exist, uses KOs and converts to ECs."""
    url_ec = f"https://rest.kegg.jp/link/enzyme/{pathway_id}"
    response_ec = requests.get(url_ec)
    if response_ec.status_code == 200 and response_ec.text.strip():
        return [line.split("\t")[1].split(":")[1] for line in response_ec.text.strip().split("\n")]

    # fallback to KO → EC
    url_ko = f"https://rest.kegg.jp/link/ko/{pathway_id}"
    response_ko = requests.get(url_ko)
    if response_ko.status_code != 200 or not response_ko.text.strip():
        return []
    ko_ids = [line.split("\t")[1].split(":")[1] for line in response_ko.text.strip().split("\n")]

    ec_set = set()
    for ko in ko_ids:
        url_ko_ec = f"https://rest.kegg.jp/link/enzyme/ko:{ko}"
        response_ko_ec = requests.get(url_ko_ec)
        if response_ko_ec.status_code == 200 and response_ko_ec.text.strip():
            for line in response_ko_ec.text.strip().split("\n"):
                ec = line.split("\t")[1].split(":")[1]
                ec_set.add(ec)
    return sorted(ec_set)

def get_pathway_name(pathway_id):
    url = f"https://rest.kegg.jp/get/{pathway_id}"
    response = requests.get(url)
    if response.status_code != 200:
        return ""
    for line in response.text.split("\n"):
        if line.startswith("NAME"):
            return line.replace("NAME", "").strip()
    return ""
```

Identify enzymes in metabolic pathways of interest

def read_pathway_ids(file_path)

↓

reads a list of KEGG pathway IDs from a file

def get_ecs_for_pathway(pathway_id)

↓

given a KEGG pathway, returns all associated EC numbers

def get_pathway_name(pathway_id)

↓

retrieves the name of a KEGG pathway

```python
def get_kegg_organism_list():
    url = "https://rest.kegg.jp/list/organism"
    response = requests.get(url)
    if response.status_code != 200:
        print("❌ Failed to retrieve KEGG organism list.")
        return {}

    organism_dict = {}
    for line in response.text.strip().split("\n"):
        parts = line.split("\t")
        if len(parts) >= 3:
            kegg_code = parts[1]
            name = parts[2].split(",")[0].lower()
            organism_dict[name] = kegg_code
    return organism_dict

def find_kegg_code(genus, species, organism_dict):
    full_name = f"{genus} {species}".lower().strip()
    genus = genus.lower().strip()

    # Tries to match the name of the taxa (Genus + Species)
    for name, code in organism_dict.items():
        if full_name == name or full_name in name:
            return code

    # Tries to match just the Genus
    for name, code in organism_dict.items():
        if genus in name.split():
            return code

    return None

def get_ec_numbers_for_organism(kegg_code):
    url = f"http://rest.kegg.jp/link/enzyme/{kegg_code}"
    response = requests.get(url)
    if response.status_code != 200 or not response.text.strip():
        return []
    return sorted(set(line.split("\t")[1].split(":")[1] for line in response.text.strip().split("\n")))
```

Identify EC numbers of specific bacteria

def get_keg_organism_list()

↓

Fetches all organisms in the KEGG database

def find_keg_code(genus, species, organism_dict)

↓

Finds the KEGG organism code of selected taxa

def get_ec_numbers_for_organism(keg_code)

↓

Gets the list of EC numbers annotated in KEGG for a given organism

```python
def process_organisms(file_path, sep=","):
    df = pd.read_csv(file_path, sep=sep)
    if 'Genus' not in df.columns or 'Species' not in df.columns:
        raise ValueError("Input file must contain 'Genus' and 'Species' columns.")
    organism_dict = get_kegg_organism_list()
    results = []
    for _, row in df.iterrows():
        genus = str(row['Genus'])
        species = str(row['Species']) if pd.notnull(row['Species']) else ""
        organism_name = f"{genus} {species}".strip()
        kegg_code = find_kegg_code(genus, species, organism_dict)
        if kegg_code:
            ec_numbers = get_ec_numbers_for_organism(kegg_code)
        else:
            ec_numbers = []
        results.append({
            "Organism": organism_name,
            "KEGG Code": kegg_code if kegg_code else "Not found",
            "EC Count": len(ec_numbers),
            "EC Numbers": "; ".join(ec_numbers) if ec_numbers else "None"})
    return pd.DataFrame(results)

def map_pathways_to_ecs(pathway_ids):
    ec_to_pathways = defaultdict(list)
    for pid in pathway_ids:
        ecs = get_ecs_for_pathway(pid)
        pname = get_pathway_name(pid)
        for ec in ecs:
            ec_to_pathways[ec].append((pid, pname))
    return ec_to_pathways

def build_compatibility_table(ec_df, ec_to_pathways):
    output_rows = []
    for ec in ec_df.index:
        if ec in ec_to_pathways:
            for sample in ec_df.columns:
                if ec_df.at[ec, sample] > 0:
                    for pid, pname in ec_to_pathways[ec]:
                        output_rows.append([sample, ec, pid, pname])
    return pd.DataFrame(output_rows, columns=["Bacterium", "EC Number", "Pathway ID", "Pathway Name"])
```

Identify which bacteria have enzymes related to the given pathways

def process_organisms(file_path, sep=“,”)

↓

Fetches KEGG code and retrieves EC numbers for each bacterium

def map_pathways_to_ecs(pathway_id)

↓

Maps EC numbers to the pathways

def build_compatibility_table(ec_df, ec_to_pathways)

↓

Crosses the ECs each organisms has with ECs in the selected pathways

Results

Effect of cellulose in the microbiota

Genus	Species
Blautia	obeum
Romboutsia	sp DR1
Dorea	formicigenerans
Blautia	faecis

File with selected bacteria

```
map00500
map00040
map00030
map00010
map00020
map00620
```

File with pathway the KEGG map ID of the metabolic pathways

Blautia obeum	3.2.1.21	map00500	Starch and sucrose metabolism	beta-glucosidase
Romboutsia sp DR1	3.2.1.21	map00500	Starch and sucrose metabolism	beta-glucosidase
Blautia faecis	3.2.1.21	map00500	Starch and sucrose metabolism	beta-glucosidase
Roseburia intestinalis	3.2.1.21	map00500	Starch and sucrose metabolism	beta-glucosidase
Blautia sp Marseille P3087	3.2.1.21	map00500	Starch and sucrose metabolism	beta-glucosidase
Roseburia inulinivorans	3.2.1.21	map00500	Starch and sucrose metabolism	beta-glucosidase
bacterium YE57	3.2.1.21	map00500	Starch and sucrose metabolism	beta-glucosidase
Blautia obeum	3.2.1.26	map00500	Starch and sucrose metabolism	beta-fructofuranosidase
Blautia faecis	3.2.1.26	map00500	Starch and sucrose metabolism	beta-fructofuranosidase
Blautia sp Marseille P3087	3.2.1.26	map00500	Starch and sucrose metabolism	beta-fructofuranosidase
bacterium YE57	3.2.1.28	map00500	Starch and sucrose metabolism	alpha,alpha-trehalase
Roseburia intestinalis	3.2.1.31	map00040	Pentose and glucuronate interconversions	beta-glucuronidase
Roseburia inulinivorans	3.2.1.31	map00040	Pentose and glucuronate interconversions	beta-glucuronidase
Romboutsia sp DR1	3.2.1.4	map00500	Starch and sucrose metabolism	cellulase
Roseburia intestinalis	3.2.1.4	map00500	Starch and sucrose metabolism	cellulase
bacterium YE57	3.2.1.4	map00500	Starch and sucrose metabolism	cellulase

File with the compatibility table containing the functional analysis results

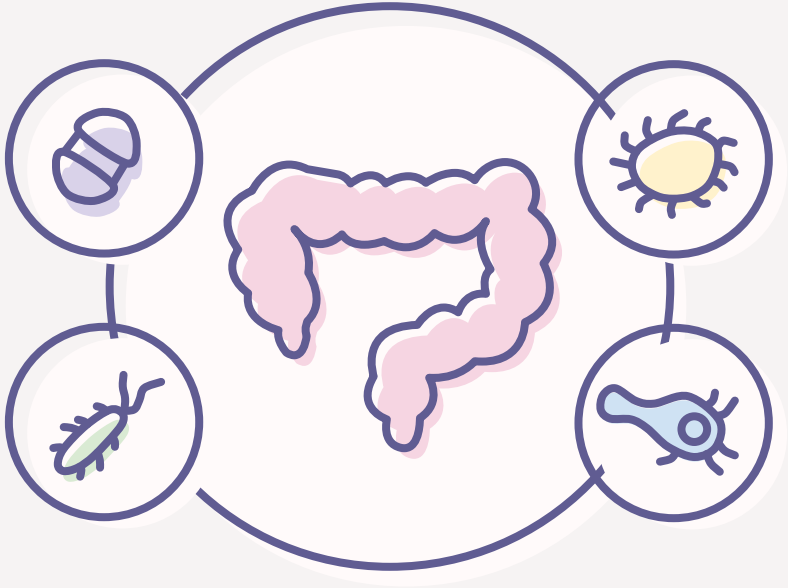
Functional results

Primary cellulose degraders

From the selected taxa only two contain the enzymes responsible for cellulose degradation

Downsteam consumers

All the selected taxa can metabolize glucose released from cellulose hydrolysis



Cross-feeding interactions

Interspecies cooperation is observed since primary degraders release sugars that are consumed by downstream consumers

Conclusions

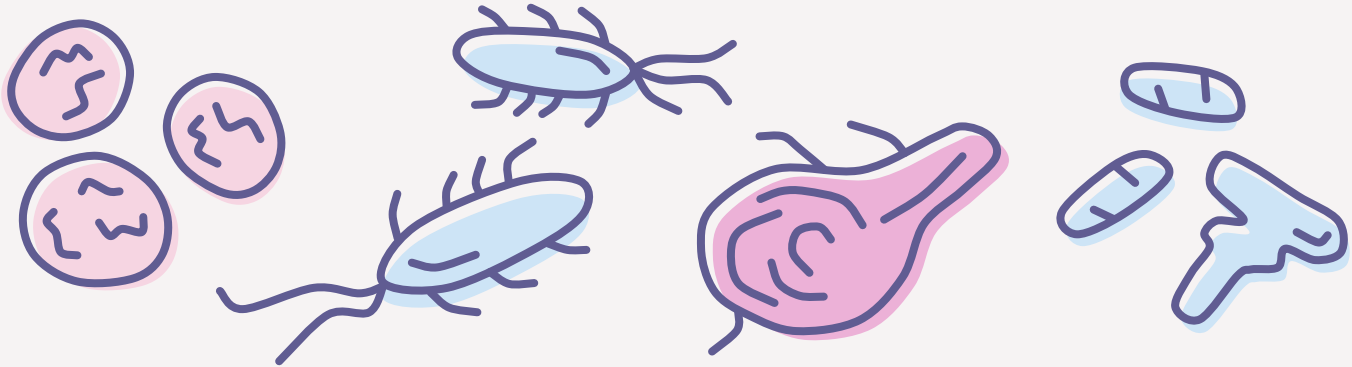
A pipeline that analyzes a file with taxa and relative abundance, performs a statistical study to interpret which taxa has biological revelevance and gives the metabolic function of these bacteria when comparing the enzymes of metabolic pathways of interessent was achieved

Future Perspectives

- Integrate UPIMAPI, a bioinformatic tool that retrieves UniProt-based information

- Perform barplots, heatmaps and other types of graphs that helps the user better understand the results of the compatibility table

Thank you for your attention



Original idea

GUMPP

GUMPP unifies taxonomic and functional profiling using QIIME2 and PICRUSt2, with built-in support for KEGG, MetaCyc, and enzyme commission (EC) number annotations.

The initial idea was to develop a costume tool that built on GUMPP's output to highlight taxon-specific enzyme repertoires providing new insights into the metabolic basis of microbiota-prebiotic interactions.

To utilize GUMPP or PICRUSt2, bioinformatic tools such as Ubuntu and Docker will be necessary and while possible, these tools are not very user-friendly for non-bioinformatics expert.