Horizontal_UKB

May 18, 2025

1 Proteomic Insights into Inflammatory Bowel Disease biomarker discovery in the UK Biobank.

Authors: Francisco Salamanca[1], David Gomez[2], Daniel Bonilla[3]

Affiliations: 1. MSc Bioinformatics Student, Universidad Nacional de Colombia
2. MSc Industrial Engineering Student, Universidad Nacional de Colombia 3. System Engineering Student, Universidad Nacional de Colombia

Objective: The primary objective of this project is to develop a predictive model for inflammatory bowel disease (IBD) relapse and new onset, leveraging longitudinal proteomics data from the UK Biobank. The model aims to capture early signals of disease activity or onset, particularly focusing on proteomic biomarkers trajectories.

Background: IBD is a chronic and relapsing inflammatory disorder that includes Crohn's disease and ulcerative colitis. Despite advances in treatment, predicting disease progression and relapse remains a major clinical challenge. Multi-omics profiling provides a promising avenue for identifying molecular signatures associated with IBD activity over time.

Approach: - **Data**: UK Biobank data including proteomics (Olink panels), genomics, and clinical data. - **Participants**: Individuals with multiple time-point measurements for proteomics and relevant metadata. - **Outcome Variables**: - IBD diagnosis and subtypes (if available), - Relapse indicators or clinical events related to disease progression.

Methods: - Preprocessing of datasets and harmonization of participant identifiers. - Extraction of **relevant** clinical covariates (e.g., medication use, smoking, alcohol, BMI) from the datasets. - Identification of temporal patterns and trajectory modeling of proteomic profiles. - Machine learning models (e.g., random forest, survival models, neural networks) adjusted for medical variables known to be associated with IBD. - Validation using cross-validation and/or independent subsets of the data.

Expected Results: - Identification of omics-based biomarkers predictive of IBD relapse or future diagnosis. - Insights into the molecular mechanisms underlying disease progression. - A prototype predictive tool to aid in risk stratification and early clinical intervention.

Deliverables: - A cleaned longitudinal dataset. - Statistical and machine learning models with performance metrics. - Visualizations of longitudinal profiles and feature importance. - Final report and optional manuscript draft for publication.

1.0.1 Package Import and Path Configuration

This section imports all required packages and defines the file paths needed for the analysis.

[ ]:
```
#Load modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import os
from datetime import datetime
from scipy.stats import skew
import math
import src.visualizaciones as visualizaciones
import src.limpieza as limpieza
import src.transformaciones as transformaciones
import src.modelos as modelos
```

[ ]:
```
path = "data/UK_BIOBANK_DATA"
path_graphs = "outputs/graphs"
path_results = "outputs/results"
```

1.0.2 Data Loading and Initial Exploration

This section loads UK Biobank data modules by data type, each containing a unique identifier who refers to the other tables. A preliminary exploration is conducted to examine the structure and key features of the datasets.

The Preprocessing of each of those tables are found in indivudual notebooks at /home

Proteomics *This table includes protein expression levels measured across all individuals.*

[ ]:
```
#Charge dataframes
#Proteomics
Proteomics_df = pd.read_csv(os.path.join(path, "olink_data.tsv"), sep="\t")
Proteomics_chars_df = pd.read_csv(os.path.join(path, "Proteomics_modified_data/
 ↪olink_chars_table.tsv"), sep="\t")

#Crear columna con el nombre de las proteinas y #ordenar df
cols = Proteomics_chars_df.columns.tolist()
Proteomics_chars_df.insert(cols.index("ValueType"),␣
 ↪"pname",Proteomics_chars_df["Field"].str.split(";").str[0])
Proteomics_chars_df.head()
```

[ ]:
```
#Encuentra columnas duplicadas y eliminalas
Proteomics_df = Proteomics_df.loc[:, ~Proteomics_df.columns.duplicated()]
```

[ ]:
```
##Cambiar las columnas de Olink_proteomics, pasarlas de formato:␣
 ↪FieldID_instance a formato: Nombre_instancia

# Extraer columnas (excepto 'eid')
original_cols = Proteomics_df.columns.tolist()
data_cols = original_cols[1:]

# Extraer FieldID y instance de los nombres de columna
fids = [int(re.search(r'f_(\d+)_', col).group(1)) if re.search(r'f_(\d+)_',␣
 ↪col) else None for col in data_cols]
instances = [re.sub(r'f_\d+_|f_NA_', '', col) for col in data_cols]

# Crear DataFrame auxiliar
col_df = pd.DataFrame({'old': data_cols, 'fids': fids, 'instance': instances})

# Unir con olinkchar para obtener los 'pname'
col_df = col_df.merge(Proteomics_chars_df[['FieldID', 'pname']],␣
 ↪left_on='fids', right_on='FieldID', how='left')

# Crear nuevos nombres
col_df['newname'] = col_df['pname'] + '_' + col_df['instance']

# Asignar nuevos nombres de columna
Proteomics_df.columns = ['eid'] + col_df['newname'].tolist()

# Obtener la columna 'eid', que contiene los IDs de los participantes
eids = Proteomics_df['eid']
```

[ ]:
```
Proteomics_df.head()
```

[ ]:
```
# Convert column names to strings
Proteomics_df.columns = Proteomics_df.columns.astype(str)

# Hay proteinas que no estan en las 4 sets longitudinales, por lo que se tiene␣
 ↪que decidir o tener en cuenta!!!
# Ejemplo:
filtered_columns = [col for col in Proteomics_df.columns if col.
 ↪startswith('CD6')]
print(filtered_columns)
```

Phenotypes *This table contains data on participants lifestyle habits and behavioral traits.*

[ ]:
```
#Charge the dataframes
phenotypes_df = pd.read_csv(os.path.join(path,"phenotype_data.tsv"), sep="\t")
phenotypes_df.head()
```

[ ]:
```
##Phenotypes DF ####
##¿Cuales fenotipos estan mas associados a la condicion de tener ibd?
#Se sabe por bibliografia que: smoking y alcohol

# Seleccionar columnas cuyos nombres contienen _20116_ o _20117_
phenodata_esential_df = phenotypes_df.filter(regex=r'eid|_20116_|_20117_',␣
 ↪axis=1)

# Crear un diccionario para renombrar columnas
new_columns = {
    col: col.replace('f_20116', 'Smoking').replace('f_20117', 'Alcohol').
 ↪removesuffix('_0')
    for col in phenodata_esential_df.columns
}

# Renombrar las columnas
phenodata_esential_df.rename(columns=new_columns, inplace=True)

# Mostrar el DataFrame resultante
phenodata_esential_df.head()
```

[ ]:
```
# Reemplazar -3 por np.nan en columnas numéricas, excepto 'eid'
cols_to_replace = phenodata_esential_df.select_dtypes(include='number').columns.
 ↪drop('eid', errors='ignore')
phenodata_esential_df[cols_to_replace] = phenodata_esential_df[cols_to_replace].
 ↪replace(-3, np.nan)
phenodata_esential_df.head()
```

Etnicity *This table contains Etnicity info of the different participants according to the genetics. is coded in a PCA base*

[ ]:
```
#Charge dataframes
etnicidad_df = pd.read_csv(os.path.join(path,"genomics_data.tsv"), sep="\t")
etnicidad_df.head()
```

[ ]:
```
#Filtrar por el id "Genetic principal components (22009)", el campo el cual me␣
 ↪indica etnicidad

etnicidad_df.columns = etnicidad_df.columns.str.replace("f_22009_0", "PC",␣
 ↪regex=True)
etnicidad_df.head()
```

Physical Measures *This table provides clinical measurements of the participants, including weight, height, and related physical data.*

[ ]:
```
#Charge dataframes
physical_measures_df = pd.read_csv(os.path.join(path, "Physical_measures_data.
 ↪tsv"), sep="\t")
physical_measures_df.head()
```

[ ]:
```
# 1. Seleccionar columnas relevantes de physical_measures_df
columns_to_keep = ['eid'] + [col for col in physical_measures_df.columns if re.
 ↪search(r'_48_|_49_|_23104_|_23100_|_23101_|_23102_', col)]
phys_esential = physical_measures_df[columns_to_keep].copy()


# 2. Calcular WHR para cada visita (0 a 3), Crear una nueva columna 'WHR' para␣
 ↪cada punto, donde sea la relacion entre la cintura y la cadera
for i in range(4):
    phys_esential[f'WHR_{i}'] = phys_esential[f'f_48_{i}_0'] /␣
 ↪phys_esential[f'f_49_{i}_0']

# 3. Renombrar columnas
phys_esential.columns = (
    phys_esential.columns
    .str.replace('f_48', 'Waist', regex=True)
    .str.replace('f_49', 'Hip', regex=True)
    .str.replace('f_23104', 'BMI', regex=True)
    .str.replace('f_23100', 'wFatMass', regex=True)
    .str.replace('f_23101', 'wFatFreeMass', regex=True)
    .str.replace('f_23102', 'wWaterMass', regex=True)
    .str.replace('_0$', '', regex=True)
    .str.replace(r'WHR$', 'WHR_0', regex=True)
)

phys_esential.head()
```

First Occurrences *This table documents the initial diagnosis dates of different diseases for each participant in the database.*

[ ]:
```
#Load dataframes
first_occurences_df = pd.read_csv(os.path.join(path, "FirstOccurrences_data.
 ↪tsv"), sep="\t")
first_occurences_chars_df = pd.read_csv(os.path.join(path,␣
 ↪"FirstOccurrences_chars.tsv"), sep="\t")
```

Population Characteristics *This table contains data on demographic traits, social factors, and general lifestyle information of the participants.*

[ ]:
```
#Load dataframes
popchar_df = pd.read_csv(os.path.join(path, "popchar_data.tsv"), sep="\t")
popchar_df.head()
```

```
[ ]: # 1. Seleccionar columnas relevantes de popchars_df
columns_to_keep = ['eid'] + [col for col in popchar_df.columns if re.
 ↪search(r'_31_|_34_|_52_', col)]
popchars_esential = popchar_df[columns_to_keep].copy()

# 2. Renombrar columnas
popchars_esential.columns = (
    popchars_esential.columns
    .str.replace('f_31', 'Sex', regex=True)
    .str.replace('f_34', 'YearofBirth', regex=True)
    .str.replace('f_52', 'MonthofBirth', regex=True)
    .str.replace('_0$', '', regex=True)
)

popchars_esential.head()
```

Recruitment *This table includes participant recruitment dates and basic enrollment information.*

```
[ ]: #Load dataframes
recruitment_df = pd.read_csv(os.path.join(path, "recruitment_data.tsv"),␣
 ↪sep="\t")
recruitment_df.head()
```

Touchscreen *This table contains answers to various touchscreen questionnaires on topics such as health, habits, and lifestyle.*

```
[ ]: #Load dataframes
touchscreen_df = pd.read_csv(os.path.join(path, "touchscreen_data.tsv"),␣
 ↪sep="\t")
touchscreen_chars_df = pd.read_csv(os.path.join(path, "touchscreen_chars.tsv"),␣
 ↪sep="\t")
```

1.0.3 Covariate Selection for IBD

This section aims to identify relevant and significant variables within the UK Biobank data that can serve as covariates in the development of a proteomic biomarker discovery model for Inflammatory Bowel Disease (IBD).

1.0.4 Dataset Assembly

This section focuses on merging the proteomic data with the previously identified covariates to generate a final, modeling-ready dataset.

```
[ ]: # Obtener todas las fechas del dataset FIRSTOCCURRENCES

dates_raw = []

# Recolectar todas las fechas únicas de cada columna a partir de la columna 2
```

```
for col in first_occurences_df.columns[1:]:
    unique_vals = first_occurences_df[col].dropna().unique()
    dates_raw.extend([str(val) for val in unique_vals])

# Quitar duplicados
dates_raw = list(set(dates_raw))

# Eliminar fechas específicas por posición
bad_indices = [19006, 19007, 19008, 19009, 19050]  # Ajustado a base-0
dates_raw0 = [date for i, date in enumerate(dates_raw) if i not in bad_indices]

# Intentar convertir a formato fechas
dates = []
failed_dates = []

for date_str in dates_raw0:
    try:
        parsed = pd.to_datetime(date_str, errors='raise')
        dates.append(parsed)
    except:
        failed_dates.append(date_str)

# Reintentar con las fallidas
for date_str in failed_dates:
    try:
        parsed = pd.to_datetime(date_str, errors='raise')
        dates.append(parsed)
    except:
        pass  # Puedes guardar las que siguen fallando si quieres analizarlas

# Ordenar y eliminar duplicados
dates = sorted(set(dates))

# Convertir a DataFrame
dates_df = pd.DataFrame(dates, columns=['Date'])

dates_df.head()
```

[ ]:
```
#Obtener todas las enfermedades gastrointestinales del dataset FIRSTOCCURRENCES

# Obtener el último término de cada jerarquía en la columna 'Path'
disease_umbrellas = (
    first_occurences_chars_df['Path']
    .dropna()
    .unique()
)
```

```
# Aplicar transformación a cada string: dividir por '>', invertir, tomar el primero y limpiar espacios
disease_umbrellas = [
    path.split('>')[-1].strip() for path in disease_umbrellas
]

# Filtrar por 'Digestive system disorders' y 'Date'
digestive_disorders = first_occurences_chars_df[
    first_occurences_chars_df['Path'].str.contains('Digestive system disorders', na=False) &
    (first_occurences_chars_df['ValueType'] == 'Date')
]

# Convertir a DataFrame
digestive_disorders_df = pd.DataFrame(digestive_disorders)
digestive_disorders_df.head()
```

[ ]:
```
# Obtener los campos que tienen la enfermedad IBD: UC y CD del dataset FIRSTOCCURRENCES_Chars

# Filtrar campos que contienen K52, K50 o K51 en la columna 'Field', y que sean del tipo 'Date'
ibd_fields = first_occurences_chars_df[
    first_occurences_chars_df['Field'].str.contains('K52|K50|K51', na=False) &
    (first_occurences_chars_df['ValueType'] == 'Date')
]

# Convertir a DataFrame
ibd_fields_df = pd.DataFrame(ibd_fields)
ibd_fields_df.head()
```

[ ]:
```
#Filtrar por individuos que tienen olink del dataset recruitment_data

# Filtrar recruitmentdata_raw para quedarse con los individuos con datos Olink, es decir, aquellos que tienen un eid en el dataset de Proteomics
olink_recruits = recruitment_df[recruitment_df['eid'].isin(eids)]

# Filtrar first_ocurrences también por los mismos IDs del dataset Proteomics
olink_first_ocurrences = first_occurences_df[first_occurences_df['eid'].isin(eids)]

# Seleccionar la columna FieldID de ibd_fields y conversirla a string
field_ids = ibd_fields['FieldID'].astype(str).tolist()

# Crear patrón regex: '131626|131628|131630|...'
pattern = '|'.join(field_ids)
```

```
# Filtrar columnas que contienen alguna coincidencia con los IDs + 'eid'
columns_to_keep = ['eid'] + [col for col in olink_first_ocurrences.columns if␣
 ↪re.search(pattern, col)]

# Seleccionar columnas
fosid = olink_first_ocurrences[columns_to_keep].copy()

# Renombrar columnas: quitar 'f_' y '_0', luego reemplazar IDs por etiquetas
fosid.columns = (
    fosid.columns
    .str.replace('f_', '', regex=False)
    .str.replace('_0', '', regex=False)
    .str.replace('131626', 'CD', regex=False)
    .str.replace('131628', 'UC', regex=False)
    .str.replace('131630', 'IBD', regex=False)
)


fosid.head()
```

[ ]:
```
# Contar valores no nulos por columna del DataFrame fosid
non_na_counts = fosid.notna().sum()

# Obtener los IDs únicos de pacientes enfermos con IBD, UC o CD
sick = pd.concat([
    fosid.loc[fosid['CD'].notna(), 'eid'],
    fosid.loc[fosid['UC'].notna(), 'eid'],
    fosid.loc[fosid['IBD'].notna(), 'eid']
]).unique()

#convertir a DataFrame
sick_df = pd.DataFrame(sick, columns=['eid'])
sick_df.head()
```

[ ]:
```
#Filtrar las columnas que están relacionadas con trastornos digestivos

# Crear un patrón regex a partir de los FieldIDs
pattern = '|'.join(map(str, digestive_disorders['FieldID']))

# Seleccionar columnas: 'eid' + aquellas que coincidan con el patrón
ddsid = olink_first_ocurrences[['eid'] + [col for col in olink_first_ocurrences.
 ↪columns if re.search(pattern, col)]]

# Conertir a DataFrame
ddsid = pd.DataFrame(ddsid)
ddsid.head()
```

[ ]:
```
# Filtrar filas donde todas las columnas excepto 'eid' son NA

digestive_cols = ddsid.columns.drop('eid')
mask = ddsid[digestive_cols].isna().sum(axis=1) == len(digestive_cols)
popcontrols = ddsid[mask]

# Obtener los Ids de los controles poblacionales (Sin enfermedades digestivas)
popcontrolsids = popcontrols['eid'].values

# Convertir a DataFrame
popcontrolsids_df = pd.DataFrame(popcontrolsids, columns=['eid'])
popcontrolsids_df.head()
```

[ ]:
```
# Seleccionar columnas específicas de olink_recruits
timestamps = olink_recruits.loc[:, [
    'eid',
    'f_53_0_0', 'f_53_1_0', 'f_53_2_0', 'f_53_3_0',
    'f_21003_0_0', 'f_21003_1_0', 'f_21003_2_0', 'f_21003_3_0',
    'f_54_0_0'
]].copy()

# Renombrar columnas de interes
timestamps.columns = [
    'eid', 'TM1', 'TM2', 'TM3', 'TM4',
    'Age1', 'Age2', 'Age3', 'Age4',
    'AssessmentCentre'
]

# Hacer merge con fosid por 'eid'
infodata = pd.merge(fosid, timestamps, on='eid', how='left')

infodata.head()
```

[ ]:
```
# Convertir las columnas de fecha a tipo datetime
date_cols = ['CD', 'UC', 'IBD', 'TM1', 'TM2', 'TM3', 'TM4']
infodata[date_cols] = infodata[date_cols].apply(pd.to_datetime, errors='coerce')

# Crear nuevas columnas con la diferencia en días
infodata['CD2T1'] = (infodata['CD'] - infodata['TM1']).dt.days
infodata['CD2T2'] = (infodata['CD'] - infodata['TM2']).dt.days
infodata['CD2T3'] = (infodata['CD'] - infodata['TM3']).dt.days
infodata['CD2T4'] = (infodata['CD'] - infodata['TM4']).dt.days

infodata['UC2T1'] = (infodata['UC'] - infodata['TM1']).dt.days
infodata['UC2T2'] = (infodata['UC'] - infodata['TM2']).dt.days
infodata['UC2T3'] = (infodata['UC'] - infodata['TM3']).dt.days
infodata['UC2T4'] = (infodata['UC'] - infodata['TM4']).dt.days
```

```
infodata['IBD2T1'] = (infodata['IBD'] - infodata['TM1']).dt.days
infodata['IBD2T2'] = (infodata['IBD'] - infodata['TM2']).dt.days
infodata['IBD2T3'] = (infodata['IBD'] - infodata['TM3']).dt.days
infodata['IBD2T4'] = (infodata['IBD'] - infodata['TM4']).dt.days

# Guardar como nuevo DataFrame
infodata0 = infodata.copy()

# Dataframe
infodata0_df = pd.DataFrame(infodata0)
infodata0_df.head()
```

[ ]:
```
# Calcular variables tiempo relativo a diagnóstico
for disease in ['CD', 'UC', 'IBD']:
    for timepoint in ['TM1', 'TM2', 'TM3', 'TM4']:
        infodata[f'{disease}2{timepoint}'] = (infodata[disease] -␣
 ↪infodata[timepoint]).dt.days


# Pivotear las columnas CD, UC, IBD a formato largo
infodata_long = infodata.melt(
    id_vars=[col for col in infodata.columns if col not in ['CD', 'UC', 'IBD']],
    value_vars=['CD', 'UC', 'IBD'],
    var_name='Disease',
    value_name='DiagnosedAt'
)

# Calcular diferencias DxTM1 a DxTM4 (días entre diagnóstico y visitas TM)
for i in range(1, 5):
    infodata_long[f'DxTM{i}'] = (pd.to_datetime(infodata_long['DiagnosedAt'],␣
 ↪errors='coerce') -
                                 pd.to_datetime(infodata_long[f'TM{i}'],␣
 ↪errors='coerce')).dt.days

# Crear categorías Pre/Post diagnóstico
for i in range(1, 5):
    infodata_long[f'TCategory{i}'] = np.where(
        infodata_long[f'DxTM{i}'] < 0,
        'Pre-diagnosis',
        'Post-diagnosis'
    )

# Pivotear para reorganizar variables TM, Age, DxTM, TCategory con número de␣
 ↪visita
# Primero, hacemos melt dejando columnas fijas
```

```
id_vars = ['eid', 'Disease', 'DiagnosedAt', 'AssessmentCentre'] + [col for col␣
 ↪in infodata_long.columns if not any(s in col for s in ['TM', 'Age', 'DxTM',␣
 ↪'TCategory'])]
melt_vars = [col for col in infodata_long.columns if any(s in col for s in␣
 ↪['TM', 'Age', 'DxTM', 'TCategory'])]

# Para simplicidad, hacemos melt explícito con esas columnas:
cols_to_melt = ['TM1', 'TM2', 'TM3', 'TM4',
                'Age1', 'Age2', 'Age3', 'Age4',
                'DxTM1', 'DxTM2', 'DxTM3', 'DxTM4',
                'TCategory1', 'TCategory2', 'TCategory3', 'TCategory4']

df_melt = infodata_long.melt(
    id_vars=['eid', 'Disease', 'DiagnosedAt', 'AssessmentCentre'],
    value_vars=cols_to_melt,
    var_name='VariableTime',
    value_name='Value'
)

# Extraer nombre base y número de visita
df_melt[['Variable', 'Times']] = df_melt['VariableTime'].str.
 ↪extract(r'([A-Za-z]+)(\d+)')

# Pivot para expandir variables a columnas
infodata1 = df_melt.pivot_table(
    index=['eid', 'Disease', 'DiagnosedAt', 'AssessmentCentre', 'Times'],
    columns='Variable',
    values='Value',
    aggfunc='first'
).reset_index()

# Reordenar columnas
cols_order = ['eid', 'Disease', 'DiagnosedAt', 'AssessmentCentre', 'Times',␣
 ↪'Age', 'TM', 'DxTM', 'TCategory']
infodata1 = infodata1[cols_order]

# Convertir a datetime
infodata1_df = pd.DataFrame(infodata1)
infodata1_df.head()
```

[ ]:
```
# Filtrar filas con eid en 'sick' y al menos un tiempo relativo a diagnóstico␣
 ↪positivo (>0)
mask_sick = infodata0['eid'].isin(sick)
time_cols =␣
 ↪['CD2T1','CD2T2','CD2T3','CD2T4','UC2T1','UC2T2','UC2T3','UC2T4','IBD2T1','IBD2T2','IBD2T3'

# Crear una máscara que chequea si alguna de las columnas time_cols es > 0
```

```
mask_time_positive = infodata0[time_cols].gt(0).any(axis=1)
sickdb = infodata0.loc[mask_sick & mask_time_positive]


# Filtrar sick + controles poblacionales
combined_array = np.concatenate((sick, popcontrolsids))

mask_sickpopcon = infodata0['eid'].isin(combined_array)
sickpopcon = infodata0.loc[mask_sickpopcon]
sickpopcon = infodata0.loc[mask_sickpopcon]

# Solo controles poblacionales
mask_popcon = infodata0['eid'].isin(popcontrolsids)
pocon = infodata0.loc[mask_popcon]

# Filtrar filas donde CD no es NA (no nulo)
sickpopcon_cd = sickpopcon.loc[sickpopcon['CD'].notna()]
infodata0_cd = infodata0.loc[infodata0['CD'].notna()]
cds = infodata0.loc[infodata0['CD'].notna()]
```

[ ]:
```
# Filtrar subset de personas con diagnóstico exclusivo

cds_only = sickpopcon.loc[
    sickpopcon['CD'].notna() &
    sickpopcon['UC'].isna() &
    sickpopcon['IBD'].isna()
]

uc_only = sickpopcon.loc[
    sickpopcon['UC'].notna() &
    sickpopcon['CD'].isna() &
    sickpopcon['IBD'].isna()
]

ibd_only = sickpopcon.loc[
    sickpopcon['IBD'].notna() &
    sickpopcon['CD'].isna() &
    sickpopcon['UC'].isna()
]

uc_ibd_only = sickpopcon.loc[
    sickpopcon['UC'].notna() &
    sickpopcon['IBD'].notna() &
    sickpopcon['CD'].isna()
]

cd_ibd_only = sickpopcon.loc[
```

```
    sickpopcon['CD'].notna() &
    sickpopcon['IBD'].notna() &
    sickpopcon['UC'].isna()
]

# Filtrar cds_only donde TM2, TM3 y TM4 sean NA
cds_only_missing_TM234 = cds_only.loc[
    cds_only['TM2'].isna() &
    cds_only['TM3'].isna() &
    cds_only['TM4'].isna()
]

# Combinar con controles poblacionales (pocon)
cds_popc = pd.concat([cds_only, pocon], ignore_index=True)
uc_popc = pd.concat([uc_only, pocon], ignore_index=True)
ibd_popc = pd.concat([ibd_only, pocon], ignore_index=True)
```

[ ]:
```
# Crear lista de eids con solo enfermedad CD, UC o IBD
disease_only_eids = pd.concat([cds_only['eid'], uc_only['eid'],␣
 ↪ibd_only['eid']]).unique()

# Combinar los grupos ibd_only, cds_only, uc_only y controles poblacionales␣
 ↪(pocon)
disease_popc = pd.concat([ibd_only, cds_only, uc_only, pocon],␣
 ↪ignore_index=True)

# Seleccionar columnas específicas en disease_popc (similar a select)
cols_of_interest = ['eid', 'CD', 'UC', 'IBD', 'TM1', 'Age1',␣
 ↪'AssessmentCentre', 'CD2T1', 'UC2T1', 'IBD2T1']
disease_popc_0 = disease_popc.loc[:, cols_of_interest]

#Filtrar solo con los pacientes
donly = infodata1.loc[infodata1['eid'].isin(disease_only_eids)]

#Tablas de conteos
# Conteo categorías en la visita Times=2 por enfermedad
print(donly.loc[(donly['Disease'] == 'CD') & (donly['Times'] == '2'),␣
 ↪'TCategory'].value_counts())
print(donly.loc[(donly['Disease'] == 'UC') & (donly['Times'] == '2'),␣
 ↪'TCategory'].value_counts())
print(donly.loc[(donly['Disease'] == 'IBD') & (donly['Times'] == '2'),␣
 ↪'TCategory'].value_counts())

#Identificar IDs con post-diagnóstico en la primera visita y filtrar␣
 ↪pre-diagnóstico en otras visitas
t1post = donly.loc[
```

```
    (donly['Disease'] == 'IBD') & (donly['Times'] == '1') & (donly['TCategory']==
 ↪'Post-diagnosis'),
    'eid'
].unique()

#Contar filas donde para esos eid, en otras visitas no 1, la categoría sea
 ↪Pre-diagnosis
count_pre_other_times = donly.loc[
    (donly['eid'].isin(t1post)) &
    (donly['Disease'] == 'IBD') &
    (donly['Times'] != '1') &
    (donly['TCategory'] == 'Pre-diagnosis')
].shape[0]

#Filtrar ibd_popc con IBD NA
ibd_na = ibd_popc.loc[ibd_popc['IBD'].isna()]
```

[ ]:
```
# Crear dataframe ibd con nuevas variables y joins
# Crear columna Disease con prioridad CD > UC > IBD > Control
def classify_disease(row):
    if pd.notna(row['CD']):
        return 'CD'
    elif pd.notna(row['UC']):
        return 'UC'
    elif pd.notna(row['IBD']):
        return 'IBD'
    else:
        return 'Control'

ibd = disease_popc_0.copy()
ibd['Disease'] = ibd.apply(classify_disease, axis=1)

# Crear columna Time_Category según IBD2T1
ibd['Time_Category'] = np.where(ibd['IBD2T1'] < 0, 'Pre-diagnosis',
 ↪'Post-diagnosis')
ibd.loc[ibd['Time_Category'].isna(), 'Time_Category'] = 'Control'

# Convertir Time_Category a categoría ordenada según valores únicos ordenados
unique_levels = sorted(ibd['Time_Category'].dropna().unique())
ibd['Time_Category'] = pd.Categorical(ibd['Time_Category'],
 ↪categories=unique_levels, ordered=True)
```

[ ]:
```
# COVARIABLES


# Hacer left joins (merge) por eid con otros dataframes productos de la
 ↪covariable selection
```

```
# Join con phenodata_esential_df
ibd = ibd.merge(phenodata_esential_df, on='eid', how='left')

# join with PCAS form genomics
pc_columns = ['eid'] + [f'PC_{i}' for i in range(1,6)]
ibd = ibd.merge(etnicidad_df.loc[:, pc_columns], on='eid', how='left')

# Join con olinkdata
ibd = ibd.merge(Proteomics_df, on='eid', how='left')

# Convertimos los nombres de columnas a string
ibd.columns = ibd.columns.astype(str)

# Seleccionar primeras 12 columnas y columnas con PC_ y que terminen en '_0',␣
 ↪excepto algunas columnas excluidas
cols = list(ibd.columns[:12]) # primeras 12 column

# Columnas con 'PC_'
cols += [col for col in ibd.columns if 'PC_' in col]

# Columnas que terminan en '_0', excluyendo algunas específicas
cols_0 = [col for col in ibd.columns if col.endswith('_0')]
exclude_cols = ['WHR_0', 'BMI_0', 'YearOfBirth_0', 'MonthOfBirth_0']
cols_0_filtered = [col for col in cols_0 if col not in exclude_cols]

# Concatenar todas las columnas finales
cols += cols_0_filtered

# Subset final
ibd0 = ibd[cols].copy()

# Asegurar que Time_Category tenga orden correcto
ibd0['Time_Category'] = pd.Categorical(
    ibd0['Time_Category'],
    categories=['Control', 'Pre-diagnosis', 'Post-diagnosis'],
    ordered=True
)
```

[ ]:
```
#Guardar el dataframe final
#NO ESTAN SALIENDO LAS COLUMNAS DE LOS DIFERENTES TIEMPOS QUE ESTAN EN INFODATAO
ibd0.to_csv(os.path.join(path, "ibd_final_data.tsv"), index=False)
```

1.0.5 Preprocessing (Raw to Tidy)

Preprocessing of the merged dataset, including NA handling, outlier removal, and variable transformations to obtain a tidy and consistent format for modeling.

[ ]:
```
#Lo correcto es tratar cada proteína individualmente

#Porque:
#-Cada proteína tiene distinta dispersión y rango.
#-Los outliers son específicos por variable, y afectan interpretaciones␣
 ↪clínicas.
#-Muchas técnicas de reducción de dimensión, clustering o modelos␣
 ↪multivariantes asumen datos limpios por variable.
```

[ ]:
```
# Cargar el DataFrame
ibd0 = pd.read_csv(os.path.join(path, "ibd_final_data.tsv"), sep=",")


# Columnas a excluir
cols_excluir = ['Smoking_0', 'Alcohol_0']

# Seleccionar columnas que terminan en '_0' pero excluyendo las que no quieres
cols_0_filtradas = [col for col in ibd0.columns if col.endswith('_0') and col␣
 ↪not in cols_excluir]

# Agregar explícitamente 'disease' si está en el DataFrame
if 'Disease' in ibd0.columns:
    cols_0_filtradas.append('Disease')
if 'Time_Category' in ibd0.columns:
    cols_0_filtradas.append('Time_Category')

# Filtrar el DataFrame
ibd0_filtrado = ibd0[cols_0_filtradas]
```

[ ]:
```
# Graficar la distribución de las proteínas y guardar la figura generada

visualizaciones.graficar_distribucion_proteinas(
    proteinas=ibd0_filtrado,
    path=path_graphs,
    clase='Disease',
    nombre='Distribución_de_Proteínas_por_Enfermedad.pdf',
    mostrar=False,
);

print("Distribución de proteínas por enfermedad guardada en la carpeta de␣
 ↪gráficos.")
```

[ ]:
```
# Quitar columnas con más del 80% de NAs

# Calculo de NAs por columna
countnas = ibd0_filtrado.isna().sum()
```

```
# Mostrar porcentaje de NAs por columna en un DataFrame, ordenado descendente
percent_nas = pd.DataFrame(ibd0_filtrado.isna().mean() * 100,␣
 ↪columns=['Percent_NA'])
percent_nas_sorted = percent_nas.sort_values(by='Percent_NA', ascending=False)

ibd0_filtrado_clean= limpieza.eliminar_nas_col(ibd0_filtrado, 0.8)
```

[ ]:
```
# Eliminar outliers usando el método IQR
ibd0_filtrado_sin_outliers = limpieza.eliminar_outliers_iqr(ibd0_filtrado_clean)
ibd0_filtrado_sin_outliers
```

[ ]:
```
#Normalizar cada proteína dependiendo de su asimetria,
ibd0_normalizado = transformaciones.
 ↪normalizar_proteinas(ibd0_filtrado_sin_outliers, clase='Disease')
ibd0_normalizado
```

[ ]:
```
# Graficar nuevamente las distribuciones de las proteínas transformadas y␣
 ↪guardar la figura generada

visualizaciones.graficar_distribucion_proteinas(
    proteinas=ibd0_normalizado,
    path=path_graphs,
    clase='Disease',
    nombre='Distribución_de_Proteínas_Normalizadas_por_Enfermedad.pdf',
    mostrar=False,
)

print("Distribución de proteínas normalizadas por enfermedad guardada en la␣
 ↪carpeta de gráficos.")
```

[ ]:
```
# Graficar los boxplots de las proteínas y guardar la figura generada
visualizaciones.graficar_boxplots_proteinas(
    df=ibd0_filtrado,
    path=path_graphs,
    clase_x='Disease',
    hue="Time_Category",
    nombre='Boxplot_de_Proteinas.pdf',
    mostrar=False,
)

print("Boxplot de proteínas guardado en la carpeta de gráficos.")
```

[ ]:
```
# Graficar los boxplots de las proteínas normalizadas y guardar la figura␣
 ↪generada
visualizaciones.graficar_boxplots_proteinas(
    df=ibd0_normalizado,
    path=path_graphs,
```

```
    clase_x='Disease',
    hue="Time_Category",
    nombre='Boxplot_de_Proteinas_normalizadas.pdf',
    mostrar=False,
)

print("Boxplot de proteínas normalizadas guardado en la carpeta de gráficos.")
```

1.0.6 Visualizations

This section presents key plots for interpreting data structure, group differences, and model insights.

[ ]:
```
##Funcion ejemplo para plotear longitudinalmente


# Supongamos que `dat` es tu DataFrame equivalente a cds_only

# Por ejemplo, plotear edad en función del tiempo relativo al diagnóstico (YearsRelativeToDx),
# con puntos separados por eid (pacientes) o Disease si hay más

#plt.figure(figsize=(10,6))
#sns.scatterplot(
#    data=ibd0,
#    x='YearsRelativeToDx',  # o la variable de tiempo que tengas
#    y='Age',                # o variable que quieras visualizar
#    hue='eid',              # colorear por paciente, si hay muchos puedes omitir
#    s=30                    # tamaño de puntos, ajusta a psize=1.5 como sea necesario
#)

#plt.title('Diagnoses timeline CDs Only')
#plt.xlabel('Years Relative to Diagnosis')
#plt.ylabel('Age')

#plt.legend([],[], frameon=False)  # oculta leyenda si muchos ids
#plt.show()
```

[ ]:
```
# Crear variables anuales para cada enfermedad
ibd0['UC2T1_years'] = ibd0['UC2T1'] / 365.25
ibd0['IBD2T1_years'] = ibd0['IBD2T1'] / 365.25
ibd0['CD2T1_years'] = ibd0['CD2T1'] / 365.25
ibd0
```

[ ]:
```
#Diagnoses timeline IBD patients, tiempo relativo al diagnóstico
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=ibd0,
    x='IBD2T1_years',
    y='Age1',           # o 'Age', si así se llama la edad en tu df
    hue='eid',          # puedes usar 'Disease' si prefieres menos colores
    s=30
)

plt.title('Diagnoses timeline IBD patients')
plt.xlabel('Years Relative to Diagnosis')
plt.ylabel('Age at Measurement')

plt.legend([], [], frameon=False)  # Oculta la leyenda si hay muchos 'eid'
plt.show()
```

[ ]:
```
##Diagnoses timeline CD patients, tiempo relativo al diagnóstico

plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=ibd0,
    x='CD2T1_years',
    y='Age1',           # o 'Age', si así se llama la edad en tu df
    hue='eid',          # puedes usar 'Disease' si prefieres menos colores
    s=30
)

plt.title('Diagnoses timeline CD patients')
plt.xlabel('Years Relative to Diagnosis')
plt.ylabel('Age at Measurement')

plt.legend([], [], frameon=False)  # Oculta la leyenda si hay muchos 'eid'
plt.show()
```

[ ]:
```
##Diagnoses timeline UC patients, tiempo relativo al diagnóstico

plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=ibd0,
    x='UC2T1_years',
    y='Age1',           # o 'Age', si así se llama la edad en tu df
    hue='eid',          # puedes usar 'Disease' si prefieres menos colores
    s=30
)

plt.title('Diagnoses timeline UC patients')
plt.xlabel('Years Relative to Diagnosis')
```

```
plt.ylabel('Age at Measurement')

plt.legend([], [], frameon=False)  # Oculta la leyenda si hay muchos 'eid'
plt.show()
```

1.0.7 Models

Here we fit and evaluate various models for IBD biomarker discovery using proteomic and covariate data.

Model: Logistic Regression per Protein - Biomarker detector

MetaAnalysis Between Cohorts

Model: Survival analysis

Model: Association Rules

Model: AI