In [130]:
```
import time
start_time = time.time()
```

1.1 Project Overview

1.1 Objective

- The AlzAware project aims to develop a predictive model for early identification of Alzheimer's Disease (AD) and Alzheimer's Disease-Related Dementias (AD/ADRD) based on social determinants of health. Utilizing data from the Mexican Health and Aging Study (MHAS), this project focuses on uncovering associations between social, economic, and environmental factors and cognitive decline risk. The ultimate goal is to enable early intervention and improved access to care, especially for underrepresented populations.

Background:

- Alzheimer's Disease affects more than 6 million Americans, with numbers expected to increase as the population ages. Current diagnostic methods for AD and related dementias are often insufficient for detecting early stages, particularly among underserved communities. Social determinants of health, such as socioeconomic status, education, and healthcare access, play a significant role in health outcomes but are often overlooked in cognitive health assessments. This project seeks to leverage these factors to predict cognitive decline and provide earlier, more accessible insights into potential AD/ADRD development. Data Source:
- The dataset used in this project is derived from the Mexican Health and Aging Study (MHAS), a longitudinal survey of adults over 50 in Mexico. The dataset includes comprehensive information on demographics, economic circumstances, health, and lifestyle behaviors, spanning multiple survey years. Using historical data from 2003 and 2012 as predictive factors, the model will forecast cognitive health scores for 2016 and 2021. Methodology:
- This project follows the CRISP-DM methodology, ensuring a structured approach to data science:
  - Business Understanding: Define objectives, goals, and success metrics.
  - Data Understanding: Explore data sources, understand data structures, and identify relevant variables.
  - Data Preparation: Process, clean, and structure data for modeling.
  - Modeling: Build and optimize predictive models to estimate cognitive scores.
  - Evaluation: Validate model performance, assessing metrics such as RMSE and bias across demographics.
  - Deployment: Plan for model integration into clinical or public health systems, focusing on interpretability and usability.

1.2 Key Goals and Outcomes

- Improved Early Detection: Identify individuals at risk of AD/ADRD based on non-clinical factors, enabling timely intervention.

- Bias Mitigation: Ensure the model provides accurate predictions across diverse demographics, minimizing disparities.
- Enhanced Accessibility: Develop a model that can be applied broadly, requiring only widely available social health data.
- Potential for Generalization: Provide a framework that can be adapted for AD/ADRD

2.1. Project Objectives

Goal

- To develop an early prediction model for Alzheimer's Disease (AD) and Alzheimer's Disease-Related Dementias (AD/ADRD) using social determinants of health (SDOH) data, specifically from the Mexican Health and Aging Study (MHAS). The model aims to identify at-risk individuals for early intervention, enhancing timely diagnosis and care accessibility.

Scope

- Focus on predicting cognitive health outcomes in 2016 and 2021 using data from 2003 and 2012, capturing diverse social, economic, and health-related features for prediction.

Impact

- By enabling earlier detection through widely accessible data, this model could support a shift in Alzheimer's care towards preventive measures, improving outcomes for underrepresented populations at higher risk of delayed diagnoses.

2.2. Stakeholders and their Objectives

Primary Stakeholders

- Healthcare Providers: Benefit from predictive insights to identify high-risk patients, enabling early intervention and preventive care.
- Public Health Organizations: Use data insights to drive policies, initiatives, and resource allocation for dementia prevention in at-risk populations.
- Researchers: Gain a framework to explore SDOH impacts on cognitive health, promoting further Alzheimer's research in underrepresented groups.
- Patients and Families: Receive support for proactive health management, potentially slowing cognitive decline and improving quality of life.

Secondary Stakeholders * Policy Makers: Data-driven insights can inform public health policy and funding for Alzheimer's care, particularly in vulnerable populations. * Insurance Providers: Early risk detection could support cost-effective, preventive healthcare solutions for high-risk individuals.

2.3. Success Criteria and Key Metrics Success Criteria

- Model Accuracy: A root mean squared error (RMSE) within an acceptable threshold on test data, indicating effective prediction of future cognitive scores.
- Bias Mitigation: Model performs equitably across demographic groups, particularly for traditionally underserved populations.
- Interpretability: Model outputs are understandable to non-technical stakeholders, allowing insights that can be effectively used in healthcare settings.

- Generalizability: The model demonstrates potential applicability beyond the study's population, extending to similar datasets and populations.

Key Metrics

- Root Mean Squared Error (RMSE): Measures predictive accuracy; a lower RMSE indicates better performance.
- Bias Detection: Analyze performance across demographic segments (e.g., age, socioeconomic status) to ensure equitable predictions.
- Coverage: Percentage of predictions made with confidence intervals; evaluates reliability across different patient profiles.
- Explainability Scores: Evaluate interpretability tools such as SHAP or LIME to assess feature contributions to risk predictions.

2.4. Constraints, Risks, and Assumptions

Constraints

- Data Limitations: Data covers only specific years (2003, 2012, 2016, 2021) and is sourced from a limited geographic region (Mexico), which may affect generalizability.
- No External Data: Per competition rules, no external data can be used, limiting additional context or feature expansion.

Risks

- Bias Risk: Potential for inherent biases in social determinants, as socioeconomic factors may skew predictions if not handled carefully.
- Model Interpretability: Complex models may reduce transparency, making it harder for stakeholders to trust or understand predictions.
- Overfitting: Using only MHAS data might lead to models that don't generalize well to other populations.

Assumptions

- Data Quality: Assumes MHAS data is reliable, well-represented, and complete enough for accurate predictions.
- Feature Stability: Assumes SDOH features impacting cognitive health remain consistent over time and across populations, enabling predictive generalization.

2.5. Outline Project Timeline and Resource Requirements

Project Timeline

- Phase 1: Data Understanding and Preparation
- Phase 2: Modeling and Hyperparameter Tuning
- Phase 3: Model Evaluation and Iteration
- Phase 4: Deployment Planning and Reporting

Resource Requirements

- Data Processing Tools: Python and relevant libraries (e.g., Pandas, scikit-learn), for efficient data preparation and analysis.

- Modeling Resources: Access to GPUs or cloud resources to handle extensive computations for model training and tuning.
- Domain Expertise: Collaboration with Alzheimer's and public health experts to validate feature selection and model outputs.
- Interpretability Tools: Access to explainability libraries (e.g., SHAP, LIME) to ensure model transparency and stakeholder trust.

2.6. Business and Organizational Impact

- Impact on Clinical Practices: Provides healthcare providers with a predictive tool to flag high-risk individuals earlier, supporting proactive cognitive health interventions.
- Public Health Applications: Enables public health organizations to allocate resources more effectively, focusing on preventive care for at-risk groups based on SDOH.
- Patient and Family Empowerment: Empowers patients and their families by providing early indicators of risk, promoting early lifestyle adjustments or interventions to delay progression.
- Research Advancement: The project creates a framework for further research on SDOH factors in cognitive decline, especially in underserved populations where AD/ADRD is prevalent.

2.7. Establish Communication and Reporting Plan

Communication with Stakeholders

- Monthly progress reports and bi-weekly check-ins with healthcare, research, and public health representatives.
- Data insights presentations at key project milestones to ensure alignment with stakeholder objectives.

Final Deliverables

- Prediction Model: A ready-to-deploy model with RMSE scores and bias evaluation.
- Explainability Reports: Documentation of the model's interpretability, detailing key SDOH features influencing predictions.
- Public Health Briefing: Executive summary for policy makers and public health organizations, highlighting actionable insights and recommendations.
- Patient-Centric Documentation: Information for healthcare providers on interpreting model

3. Data Understanding

3.1 Importing Libraries

In [131]:
```
# Data manipulation
import pandas as pd
import numpy as np

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# Custom transformers and base classes
from sklearn.base import BaseEstimator, TransformerMixin

# Preprocessing and feature engineering
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

# Imputation
from sklearn.experimental import enable_iterative_imputer  # Enable experimental
from sklearn.impute import IterativeImputer, SimpleImputer

# Feature selection
from sklearn.feature_selection import SelectKBest, f_regression, SelectFromModel

# Model selection and evaluation
from sklearn.model_selection import GridSearchCV, GroupKFold, train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.svm import SVR

# # Importing Custom Transformers
# import custom_tranformers
# from custom_transformers import InteractionTermsTransformer, SHAPFeatureSelector

# Machine learning models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import (
    StackingRegressor,
    RandomForestRegressor,
    GradientBoostingRegressor,
    HistGradientBoostingRegressor
)
import xgboost as xgb
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
import lightgbm as lgb

# Model interpretation
import shap

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

3.2 Data Collection

- Loading Data

In [135]:
```
# Load the datasets
submission_format_df = pd.read_csv('Data/submission_format.csv')
test_features_df = pd.read_csv('Data/test_features.csv')
train_features_df = pd.read_csv('Data/train_features.csv')
train_labels_df = pd.read_csv('Data/train_labels.csv')
```

3.3. Initial Data Exploration

3.3.1 Data Dimensions

In [ ]:
```
# Dimensions of Data
print("Data loaded successfully:")
print(f"Submission Format Shape: {submission_format_df.shape}")
print(f"Test Features Shape: {test_features_df.shape}")
print(f"Train Features Shape: {train_features_df.shape}")
print(f"Train Labels Shape: {train_labels_df.shape}")
```

```
Data loaded successfully:
Submission Format Shape: (1105, 3)
Test Features Shape: (819, 184)
Train Features Shape: (3276, 184)
Train Labels Shape: (4343, 3)
```

Observations

- Submission Format: (1105, 3) – This file provides a template for the expected output format. It contains 1,105 rows and 3 columns, which corresponds to IDs and the format required for submitting predictions.
- Test Features: (819, 184) – This dataset contains 819 records with 184 feature columns. It's typically used for model testing or making predictions without the target variable.
- Train Features: (3276, 184) – The training features dataset includes 3,276 records and 184 feature columns. This is used to train our model by providing it with the input data.
- Train Labels: (4343, 3) – The labels dataset has 4,343 rows and 3 columns, including the target variable. The number of records here is larger than the training features, which suggests that there might be extra data points or additional labels not matched with features.

3.3.2 Reviewing First 20 Rows

In [ ]:
```
# Display the first 20 rows of each DataFrame
print("Submission Format Sample:\n", submission_format_df.head(20))
```

```
Submission Format Sample:
      uid  year  composite_score
0    abxu  2016                0
1    aeol  2016                0
2    aeol  2021                0
3    afnb  2016                0
4    afnb  2021                0
5    ajfh  2016                0
6    ajfh  2021                0
7    ajvq  2021                0
8    akbn  2016                0
9    akcw  2021                0
10   akmb  2021                0
11   akow  2016                0
12   akup  2021                0
13   albo  2016                0
14   albo  2021                0
15   alfr  2016                0
16   amdd  2016                0
17   amdd  2021                0
18   amux  2016                0
19   amux  2021                0
```

In [ ]:
```
print("\nTest Features Sample:\n", test_features_df.head(20))
```

```
Test Features Sample:
      uid    age_03     urban_03                    married_03  n_mar_03  \
0    abxu       NaN          NaN                           NaN       NaN
1    aeol       NaN          NaN                           NaN       NaN
2    afnb       NaN          NaN                           NaN       NaN
3    ajfh       NaN          NaN                           NaN       NaN
4    ajvq  2. 60-69  1. 100,000+  1. Married or in civil union       1.0
5    akbn       NaN          NaN                           NaN       NaN
6    akcw       NaN          NaN                           NaN       NaN
7    akmb  1. 50-59  1. 100,000+  1. Married or in civil union       1.0
8    akow  1. 50-59  1. 100,000+  1. Married or in civil union       1.0
9    akup  1. 50-59  1. 100,000+  1. Married or in civil union       2.0
10   albo  1. 50-59  0. <100,000  1. Married or in civil union       1.0
11   alfr  1. 50-59  1. 100,000+  1. Married or in civil union       1.0
12   amdd       NaN          NaN                           NaN       NaN
13   amux       NaN          NaN                           NaN       NaN
14   aody       NaN          NaN                           NaN       NaN
15   aoyf       NaN          NaN                           NaN       NaN
```

In [ ]:
```
print("\nTrain Features Sample:\n", train_features_df.head(20))
```

```
Train Features Sample:
      uid    age_03     urban_03                    married_03  n_mar_03  \
0    aace       NaN          NaN                           NaN       NaN
1    aanz       NaN          NaN                           NaN       NaN
2    aape       NaN          NaN                           NaN       NaN
3    aard  1. 50-59  1. 100,000+                   3. Widowed       1.0
4    ablr       NaN          NaN                           NaN       NaN
5    abme  1. 50-59  0. <100,000  1. Married or in civil union       1.0
6    abrn  1. 50-59  0. <100,000  1. Married or in civil union       2.0
7    acet  1. 50-59  1. 100,000+  1. Married or in civil union       1.0
8    acgx       NaN          NaN                           NaN       NaN
9    acmh  3. 70-79  0. <100,000  1. Married or in civil union       2.0
10   acsp  1. 50-59  1. 100,000+  1. Married or in civil union       1.0
11   addk  3. 70-79  1. 100,000+  1. Married or in civil union       1.0
12   adtd       NaN          NaN                           NaN       NaN
13   aehw  3. 70-79  0. <100,000  1. Married or in civil union       1.0
14   aete  1. 50-59  1. 100,000+  1. Married or in civil union       1.0
15   aevc  1. 50-59  1. 100,000+  1. Married or in civil union       1.0
```

In [ ]:
```
print("\nTrain Labels Sample:\n", train_labels_df.head(20))
```

```
Train Labels Sample:
      uid  year  composite_score
0    aace  2021              175
1    aanz  2021              206
2    aape  2016              161
3    aape  2021              144
4    aard  2021              104
5    ablr  2021              183
6    abme  2021              106
7    abrn  2021              144
8    acet  2021              152
9    acgx  2021              144
10   acmh  2016               13
11   acsp  2021              193
12   addk  2016               38
13   adtd  2016              272
14   adtd  2021              254
15   aehw  2016               87
16   aehw  2021               92
17   aete  2016              203
18   aevc  2021              117
19   aewa  2021               84
```

Observations

Submission Format

- The submission_format dataset has three columns: uid, year, and composite_score.
- Each row represents a unique individual (uid) for a specific year, with composite_score currently set to 0.

- This format suggests the expected structure for submission, likely requiring predicted values for composite_score.

Test and Train Features

- Both test_features and train_features datasets contain numerous columns (184 in total), mostly focused on social, demographic, and health-related factors.
- Columns include identifiers like uid, categories like age_03, urban_03, and responses to social and health questions (married_03, n_mar_03, etc.).
- Many fields have missing values, as indicated by NaN, and categorical data encoded with prefixes like 1., 2., which may require preprocessing to make values numeric or otherwise usable in modeling.

Challenges

- The categorical values include descriptions (e.g., 1. Married or in civil union) that may need to be cleaned and converted to numeric codes.
- Some columns have complex string representations (e.g., rrelgimp_12 shows responses like "1.very important"), suggesting the need for encoding or label transformation.
- Missing data in columns such as age_03, n_mar_03, and glob_hlth_03 might impact model performance if not handled appropriately.

Potential Focus Areas for Preprocessing

- Data Cleaning: Removing numerical prefixes from categorical responses (e.g., transforming 1. 50–59 to 50–59).
- Encoding: Converting categorical variables into numerical form through techniques such as label encoding or one-hot encoding.
- Handling Missing Values: Filling in missing values based on specific imputation strategies or removing rows/columns if appropriate.
- Feature Selection: Since there are 184 features, exploring dimensionality reduction techniques might improve model performance and interpretability.

3.3.3 Data Type For Each Attributes

In [ ]:
```
#Display data types for each DataFrame
print("\nData types for each attribute in submission_format_df:\n")
print(submission_format_df.dtypes)
```

```
Data types for each attribute in submission_format_df:

uid               object
year               int64
composite_score    int64
dtype: object
```

In [ ]:
```
print("\nData types for each attribute in test_features_df:\n")
print(test_features_df.dtypes)
```

```
Data types for each attribute in test_features_df:

uid            object
age_03         object
urban_03       object
married_03     object
n_mar_03      float64
                ...
a21_12        float64
a22_12         object
a33b_12        object
a34_12         object
j11_12         object
Length: 184, dtype: object
```

In [ ]:
```
print("\nData types for each attribute in train_features_df:\n")
print(train_features_df.dtypes)
```

```
Data types for each attribute in train_features_df:

uid            object
age_03         object
urban_03       object
married_03     object
n_mar_03      float64
                ...
a21_12        float64
a22_12         object
a33b_12        object
a34_12         object
j11_12         object
Length: 184, dtype: object
```

In [ ]:
```
print("\nData types for each attribute in train_labels_df:\n")
print(train_labels_df.dtypes)
```

```
Data types for each attribute in train_labels_df:

uid               object
year               int64
composite_score    int64
dtype: object
```

Observations

Submission Format (submission_format_df)

- Contains three columns: uid, year, and composite_score.
- uid is of type object, representing unique identifiers for individuals.
- year and composite_score are of type int64, indicating they store integer values.

Test Features (test_features_df)

- Contains 184 columns, with a mix of object and float64 data types.
- uid is of type object, and most features are stored as object (likely categorical or nominal data).
- Some columns, such as n_mar_03, a21_12, are float64, indicating continuous or numeric values.

Train Features (train_features_df)

- Similar to test_features_df, it has 184 columns and contains a mix of object and float64 data types.
- The data types are consistent with those in test_features_df, which will aid in ensuring uniform preprocessing.

Train Labels (train_labels_df)

- Consists of three columns: uid, year, and composite_score.
- The data types mirror those in submission_format_df, with uid as object, and year and composite_score as int64.

Overall Observations

- Many columns in the features DataFrames (train_features_df and test_features_df) are stored as object, which suggests they may represent categorical data (e.g., marital status, education level).
- Some columns are float64, which are likely numeric and may require scaling or normalization during preprocessing.
- Consistency in data types across train_features_df and test_features_df simplifies the preprocessing pipeline, as the same transformations can be applied to both.

3.3.4 Descriptive Statistics

In [ ]:
```
# Descriptive statistics for submission_format_df
print("\nDescriptive Statistics for submission_format_df:\n")
print(submission_format_df.describe())
```

```
Descriptive Statistics for submission_format_df:

              year  composite_score
count  1105.000000           1105.0
mean   2019.162896              0.0
std       2.411604              0.0
min    2016.000000              0.0
25%    2016.000000              0.0
50%    2021.000000              0.0
75%    2021.000000              0.0
max    2021.000000              0.0
```

In [ ]:
```
# Descriptive statistics for test_features_df
print("\nDescriptive Statistics for test_features_df:\n")
print(test_features_df.describe())
```

Descriptive Statistics for test_features_df:

	n_mar_03	migration_03	adl_dress_03	adl_walk_03	adl_bath_03	\
count	568.000000	570.000000	542.000000	569.000000	569.000000	
mean	1.165493	0.077193	0.047970	0.012302	0.012302	
std	0.535254	0.267132	0.213901	0.110328	0.110328	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	0.000000	0.000000	0.000000	0.000000	
50%	1.000000	0.000000	0.000000	0.000000	0.000000	
75%	1.000000	0.000000	0.000000	0.000000	0.000000	
max	4.000000	1.000000	1.000000	1.000000	1.000000	

	adl_eat_03	adl_bed_03	adl_toilet_03	n_adl_03	iadl_money_03	...
\						
count	569.000000	569.000000	569.000000	569.000000	542.000000	...
mean	0.005272	0.038664	0.019332	0.087873	0.009225	...
std	0.072483	0.192963	0.137811	0.443272	0.095692	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	0.000000	0.000000	0.000000	0.000000	0.000000	...
max	1.000000	1.000000	1.000000	5.000000	1.000000	...

	searnings_12	hincome_12	hinc_business_12	hinc_rent_12	\
count	508.000000	7.730000e+02	7.940000e+02	794.000000	
mean	10196.850394	6.608021e+04	1.010076e+04	2216.624685	
std	35871.244519	2.540155e+05	7.190606e+04	35284.803065	
min	0.000000	-2.000000e+05	0.000000e+00	-200000.000000	
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	
50%	0.000000	2.000000e+04	0.000000e+00	0.000000	
75%	0.000000	5.000000e+04	0.000000e+00	0.000000	
max	310000.000000	6.020000e+06	1.430000e+06	710000.000000	

	hinc_assets_12	hinc_cap_12	rinc_pension_12	sinc_pension_12	\
count	794.000000	7.940000e+02	794.000000	5.080000e+02	
mean	367.758186	1.265743e+04	13476.070529	1.039370e+04	
std	4482.643368	8.003225e+04	38708.762304	5.315530e+04	
min	0.000000	-2.000000e+05	0.000000	0.000000e+00	
25%	0.000000	0.000000e+00	0.000000	0.000000e+00	
50%	0.000000	0.000000e+00	0.000000	0.000000e+00	
75%	0.000000	0.000000e+00	0.000000	0.000000e+00	
max	84000.000000	1.430000e+06	480000.000000	1.080000e+06	

	a16a_12	a21_12
count	6.000000	11.000000
mean	1975.833333	6.454545
std	25.103121	9.771015
min	1940.000000	1.000000
25%	1965.750000	1.000000
50%	1971.000000	2.000000
75%	1991.250000	7.000000
max	2011.000000	32.000000

[8 rows x 140 columns]

In [ ]:
```
# Descriptive statistics for train_features_df
print("\nDescriptive Statistics for train_features_df:\n")
print(train_features_df.describe())
```

```
Descriptive Statistics for train_features_df:

          n_mar_03  migration_03  adl_dress_03  adl_walk_03  adl_bath_03  \
count  2222.000000   2241.000000   2105.000000  2235.000000  2235.000000
mean      1.134113      0.099063      0.041805     0.017002     0.007159
std       0.482953      0.298813      0.200191     0.129308     0.084325
min       0.000000      0.000000      0.000000     0.000000     0.000000
25%       1.000000      0.000000      0.000000     0.000000     0.000000
50%       1.000000      0.000000      0.000000     0.000000     0.000000
75%       1.000000      0.000000      0.000000     0.000000     0.000000
max       5.000000      1.000000      1.000000     1.000000     1.000000

         adl_eat_03   adl_bed_03  adl_toilet_03     n_adl_03  iadl_money_03  \
count  2234.000000  2235.000000    2235.000000  2234.000000    2105.000000
mean      0.004476     0.026398       0.013423     0.068487       0.005226
std       0.066770     0.160352       0.115102     0.392793       0.072117
min       0.000000     0.000000       0.000000     0.000000       0.000000
25%       0.000000     0.000000       0.000000     0.000000       0.000000
50%       0.000000     0.000000       0.000000     0.000000       0.000000
75%       0.000000     0.000000       0.000000     0.000000       0.000000
max       1.000000     1.000000       1.000000     5.000000       1.000000

       ...  searnings_12    hincome_12  hinc_business_12  hinc_rent_12  \
count  ...  2.091000e+03  3.138000e+03      3.187000e+03  3.187000e+03
mean   ...  1.181253e+04  8.824729e+04      3.294321e+04  8.628805e+02
std    ...  6.013394e+04  6.901728e+05      6.520799e+05  3.282440e+04
min    ...  0.000000e+00 -1.900000e+05      0.000000e+00 -3.600000e+05
25%    ...  0.000000e+00  0.000000e+00      0.000000e+00  0.000000e+00
50%    ...  0.000000e+00  2.000000e+04      0.000000e+00  0.000000e+00
75%    ...  0.000000e+00  6.000000e+04      0.000000e+00  0.000000e+00
max    ...  1.370000e+06  3.602000e+07      3.600000e+07  1.200000e+06

       hinc_assets_12   hinc_cap_12  rinc_pension_12  sinc_pension_12  \
count     3187.000000  3.187000e+03      3187.000000     2.091000e+03
mean       788.202071  3.458739e+04     13850.015689     1.193687e+04
std      10314.879890  6.527276e+05     44336.711380     4.705314e+04
min          0.000000 -2.100000e+05         0.000000     0.000000e+00
25%          0.000000  0.000000e+00         0.000000     0.000000e+00
50%          0.000000  0.000000e+00         0.000000     0.000000e+00
75%          0.000000  0.000000e+00         0.000000     0.000000e+00
max     360000.000000  3.600000e+07    960000.000000     1.200000e+06

           a16a_12     a21_12
count    24.000000  42.000000
mean   1975.166667   7.833333
std      20.440086  12.585157
min    1942.000000   1.000000
25%    1960.000000   1.000000
50%    1972.500000   2.000000
75%    1988.500000   6.750000
max    2012.000000  52.000000

[8 rows x 140 columns]
```

In [ ]:
```
# Descriptive statistics for train_labels_df
print("\nDescriptive Statistics for train_labels_df:\n")
print(train_labels_df.describe())
```

```
Descriptive Statistics for train_labels_df:

              year  composite_score
count  4343.000000      4343.000000
mean   2019.162560       157.016809
std       2.410882        60.909546
min    2016.000000         4.000000
25%    2016.000000       114.000000
50%    2021.000000       157.000000
75%    2021.000000       200.000000
max    2021.000000       334.000000
```

Observations

Submission Format

- The year field shows values from 2016 to 2021 with an average close to 2019, indicating the range of years for the predictions.
- composite_score is set to 0 for all entries, which serves as a placeholder and highlights that the actual predictions will need to populate this field.

Test Features

- Contains various attributes from n_mar_03 to hinc_rent_12.
- The mean, min, max, and quartile values for each attribute provide insight into the distributions. For example:
  - Activity of Daily Living (ADL) columns such as adl_dress_03, adl_walk_03, adl_bath_03, and others show binary values (0 and 1), indicating difficulties in these daily tasks.
  - Income and Earnings fields (hincome_12, hinc_business_12) display a broad range, suggesting diversity in economic circumstances.
- Many values are missing in specific columns, as indicated by counts much lower than the total rows.

Train Features

- Similar attributes as the test set, with differences in the data distribution due to training data specifics.
- The income and ADL columns show similar ranges, with values indicating variability in health, economic conditions, and social determinants.

Train Labels

- The target variable composite_score has an average around 157, with a minimum of 4 and a maximum of 334, indicating a wide range in cognitive scores.
- The 25th and 75th percentiles for composite_score (114 and 200, respectively) suggest the data may have a normal distribution but with a slight positive skew.

These statistics highlight potential predictors related to health, social determinants, and economic factors, which could correlate with cognitive function scores in the train_labels dataset.

3.4 Data Cardinality

Assess unique values in categorical features (e.g., marital status, employment status) to understand diversity in responses.

In [ ]:
```
# List of categorical columns to analyze
categorical_columns = [
    'married_03', 'married_12', 'edu_gru_03', 'edu_gru_12',
    'employment_03', 'employment_12', 'urban_03', 'urban_12'
]

# Function to display the number of unique values in each categorical column
def assess_cardinality(df, categorical_columns):
    for col in categorical_columns:
        if col in df.columns:
            unique_values = df[col].nunique()
            print(f"Column '{col}': {unique_values} unique values")

# Assess cardinality for each dataset (update categorical_columns as needed)
print("\nData Cardinality in Train Features:")
assess_cardinality(train_features_df, categorical_columns)

print("\nData Cardinality in Test Features:")
assess_cardinality(test_features_df, categorical_columns)
```

```
Data Cardinality in Train Features:
Column 'married_03': 4 unique values
Column 'married_12': 4 unique values
Column 'edu_gru_03': 5 unique values
Column 'edu_gru_12': 5 unique values
Column 'employment_03': 4 unique values
Column 'employment_12': 4 unique values
Column 'urban_03': 2 unique values
Column 'urban_12': 2 unique values

Data Cardinality in Test Features:
Column 'married_03': 4 unique values
Column 'married_12': 4 unique values
Column 'edu_gru_03': 5 unique values
Column 'edu_gru_12': 5 unique values
Column 'employment_03': 4 unique values
Column 'employment_12': 4 unique values
Column 'urban_03': 2 unique values
Column 'urban_12': 2 unique values
```

Observations

- Marital Status (married_03 and married_12):

    - There are 4 unique values in both married_03 and married_12. This indicates that there are four distinct marital status categories in both time points (2003 and 2012), such as "Single," "Married," "Widowed," etc.
- Education Level (edu_gru_03 and edu_gru_12):
    - There are 5 unique values in edu_gru_03 and edu_gru_12, suggesting five levels of educational attainment recorded in both years. These levels might range from "No education" to "Higher education."
- Employment Status (employment_03 and employment_12):
    - Both employment_03 and employment_12 have 4 unique values, indicating four employment categories (e.g., "Unemployed," "Employed," "Retired," "Student") recorded in each time period.
- Urban/Rural Location (urban_03 and urban_12):
    - Each urban/rural feature (urban_03 and urban_12) has 2 unique values. This binary categorization likely represents whether the individual lives in an urban area (population 100,000+) or a rural area.
- The data shows consistent cardinality across train and test datasets, which suggests similar categorical diversity across the datasets and time periods. This consistency in cardinality is beneficial for model training as it indicates similar feature distributions between the train and test sets, reducing the risk of distributional shifts. These categorical levels can be used for one-hot encoding or other categorical transformations during preprocessing.

3.6 Data Quality Assessment

3.6.1 Identify Missing Values

Generating a missing values report, noting columns with missing data and their percentages.

In [ ]:
```
# Identify Missing Values
def missing_values_report(df, df_name):
    missing_values = df.isnull().sum()
    missing_percentage = (missing_values / len(df)) * 100
    missing_report = pd.DataFrame({'Missing Values': missing_values, 'Percenta
    print(f"\nMissing Values Report for {df_name}:\n")
    print(missing_report[missing_report['Missing Values'] > 0].sort_values(by=
```

In [ ]:
```
# Generate missing values report for each dataset
missing_values_report(submission_format_df, 'submission_format_df')
missing_values_report(test_features_df, 'test_features_df')
missing_values_report(train_features_df, 'train_features_df')
missing_values_report(train_labels_df, 'train_labels_df')
```

```
Missing Values Report for submission_format_df:

Empty DataFrame
Columns: [Missing Values, Percentage]
Index: []

Missing Values Report for test_features_df:

               Missing Values  Percentage
a16a_12                   813   99.267399
a22_12                    809   98.778999
a21_12                    808   98.656899
a33b_12                   808   98.656899
rjlocc_m_03               734   89.621490
...                       ...         ...
hrt_attack_12              25    3.052503
migration_12               25    3.052503
married_12                 25    3.052503
urban_12                   25    3.052503
j11_12                     24    2.930403

[182 rows x 2 columns]

Missing Values Report for train_features_df:

                Missing Values  Percentage
a16a_12                   3252   99.267399
a22_12                    3240   98.901099
a33b_12                   3234   98.717949
a21_12                    3234   98.717949
rjlocc_m_03               2965   90.506716
...                        ...         ...
rearnings_12                89    2.716728
hosp_12                     89    2.716728
pem_def_mar_12              89    2.716728
insured_12                  89    2.716728
j11_12                      75    2.289377

[182 rows x 2 columns]

Missing Values Report for train_labels_df:

Empty DataFrame
Columns: [Missing Values, Percentage]
Index: []
```

3.6.2 Duplicate Records

Check for any duplicate rows by uid and year to ensure unique entries for each individual-year

In [ ]:
```
# Check for Duplicate Records
def duplicate_records_report(df, unique_cols, df_name):
    duplicates = df.duplicated(subset=unique_cols).sum()
    print(f"\nDuplicate Records in {df_name} based on {unique_cols}:", duplica

# Check for duplicates based on 'uid' and 'year' in train and submission data
duplicate_records_report(train_features_df, ['uid'], 'train_features_df')
duplicate_records_report(test_features_df, ['uid'], 'test_features_df')
duplicate_records_report(train_labels_df, ['uid', 'year'], 'train_labels_df')
duplicate_records_report(submission_format_df, ['uid', 'year'], 'submission_fo
```

```
Duplicate Records in train_features_df based on ['uid']: 0

Duplicate Records in test_features_df based on ['uid']: 0

Duplicate Records in train_labels_df based on ['uid', 'year']: 0

Duplicate Records in submission_format_df based on ['uid', 'year']: 0
```

3.6.3 Outlier Detection

Explore for any outliers in continuous variables (e.g., household income) that may impact model performance.

In [ ]:
```
# Outlier Detection in Continuous Variables
def detect_outliers(df, continuous_columns):
    outliers_report = {}
    for col in continuous_columns:
        if col in df.columns:
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)][c
            outliers_report[col] = {'Outliers': len(outliers), 'Percentage': l
        else:
            print(f"Column '{col}' not found in DataFrame '{df}'")
    return pd.DataFrame(outliers_report).T

# Continuous columns to check for outliers (update based on actual columns in
train_continuous_columns = ['hincome_12', 'hinc_business_12', 'hinc_rent_12']
label_continuous_columns = ['composite_score']  # 'composite_score' should be

# Detect outliers in train_features_df and train_labels_df
print("\nOutlier Report for train_features_df:")
print(detect_outliers(train_features_df, train_continuous_columns))

print("\nOutlier Report for train_labels_df:")
print(detect_outliers(train_labels_df, label_continuous_columns))
```

```
Outlier Report for train_features_df:
                  Outliers  Percentage
hincome_12           331.0   10.103785
hinc_business_12     339.0   10.347985
hinc_rent_12          82.0    2.503053

Outlier Report for train_labels_df:
                 Outliers  Percentage
composite_score       5.0    0.115128
```

Obervations

- Missing Values
  - The submission_format_df and train_features_df dataframes have no missing values.
  - test_features_df has a substantial amount of missing data in certain columns, with over 99% missing in columns like a16a_12, a22_12, and a21_12. Some columns, such as hrt_attack_12 and migration_12, have relatively lower missing percentages (~3%).
- Duplicate Records
  - No duplicate records were found in any dataset based on uid (and year where applicable).
- Outliers Report
  - In train_features_df, columns such as hincome_12, hinc_business_12, and hinc_rent_12 show notable outliers, with around 10% of the records identified as outliers in hincome_12 and hinc_business_12.

- In train_labels_df, the composite_score column has minimal outliers (0.1%), indicating that the majority of the target variable values fall within a typical range.
- These observations suggest a need to handle missing data in test_features_df, possibly through imputation or removing certain columns if deemed irrelevant.
- For outliers, decisions on whether to transform, remove, or handle them differently could be

3.7 Exploring Relationships in the Data

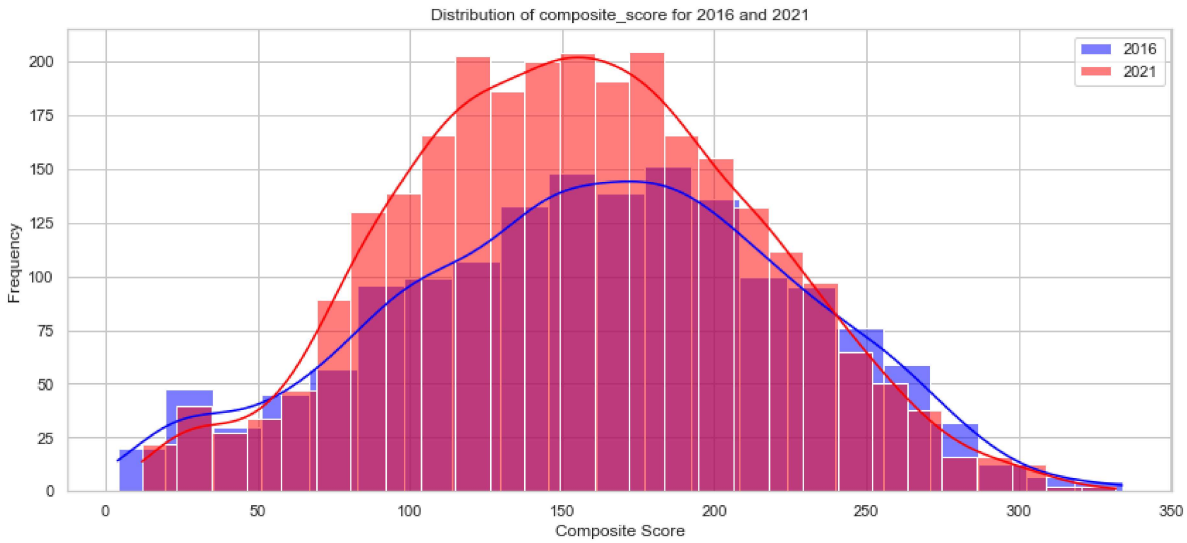
In [ ]:
```
# Set visualization style
sns.set(style="whitegrid")
```

3.7.1 Target Variable Exploration

Analyze composite_score across the years 2016 and 2021, looking at mean, standard deviation, and range.

Plot the distribution of composite_score to check for skewness and variability.

In [ ]:
```
# Target Variable Exploration: composite_score across years
plt.figure(figsize=(14, 6))
sns.histplot(train_labels_df[train_labels_df['year'] == 2016]['composite_score
sns.histplot(train_labels_df[train_labels_df['year'] == 2021]['composite_score
plt.title('Distribution of composite_score for 2016 and 2021')
plt.xlabel('Composite Score')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



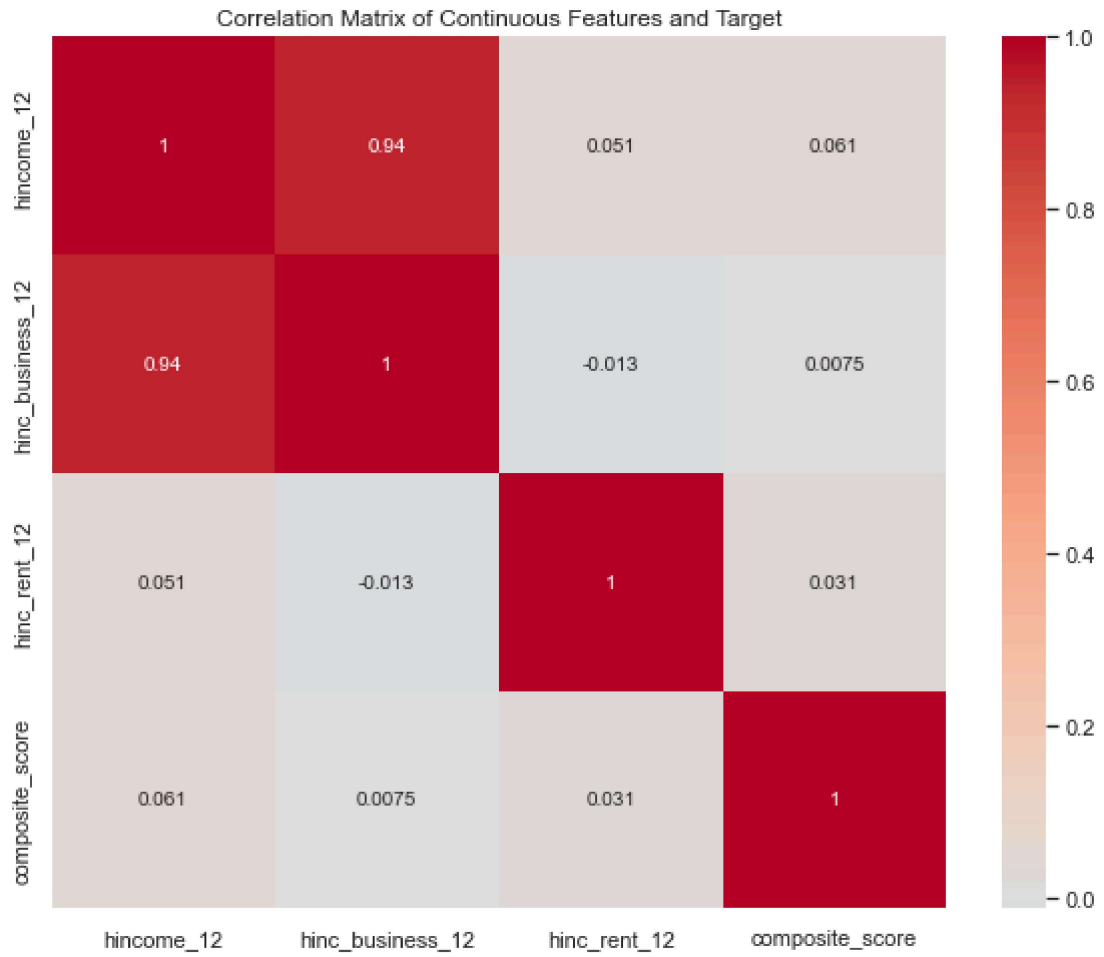
In [ ]:
```
# Summary statistics of composite_score
composite_stats_2016 = train_labels_df[train_labels_df['year'] == 2016]['compos
composite_stats_2021 = train_labels_df[train_labels_df['year'] == 2021]['compos
print("Composite Score Statistics for 2016:\n", composite_stats_2016)
print("\nComposite Score Statistics for 2021:\n", composite_stats_2021)
```

```
Composite Score Statistics for 2016:
 count    1596.000000
mean      160.431078
std        65.587728
min         4.000000
25%       114.750000
50%       163.000000
75%       207.000000
max       334.000000
Name: composite_score, dtype: float64

Composite Score Statistics for 2021:
 count    2747.000000
mean      155.033127
std        57.938599
min        12.000000
25%       114.000000
50%       154.000000
75%       195.000000
max       332.000000
Name: composite_score, dtype: float64
```

3.7.2 Feature-Target Correlations

- Perform initial correlation analysis between features and composite_score to identify potentially significant predictors.

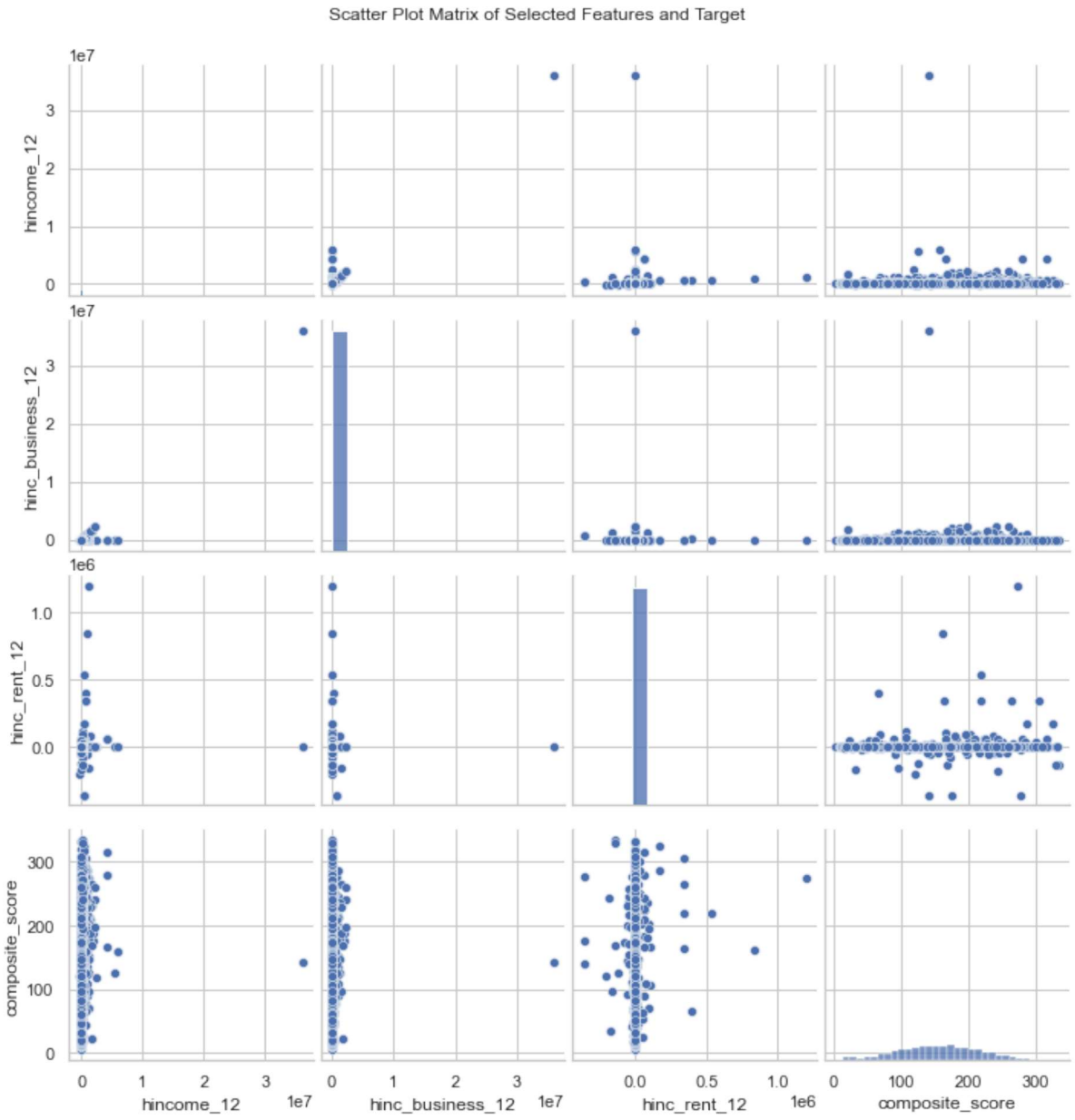
In [ ]:
```
# Correlation Analysis between Selected Features and Target Variable
# Merging train features and labels
merged_df = pd.merge(train_features_df, train_labels_df, on='uid')

# Select continuous features for correlation (adjust based on your dataset)
continuous_features = ['hincome_12', 'hinc_business_12', 'hinc_rent_12']
correlation_matrix = merged_df[continuous_features + ['composite_score']].corr

# Correlation Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix of Continuous Features and Target')
plt.show()
```



3.7.3 Cross-Feature Relationships

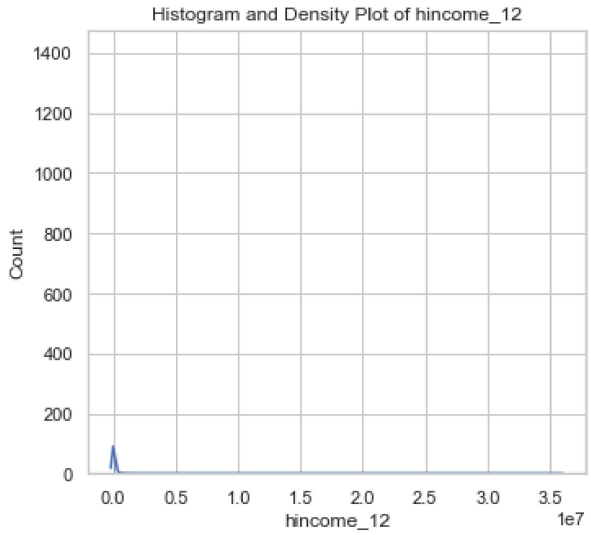
- Examine relationships among key features within and between the years (e.g., income vs. education level).

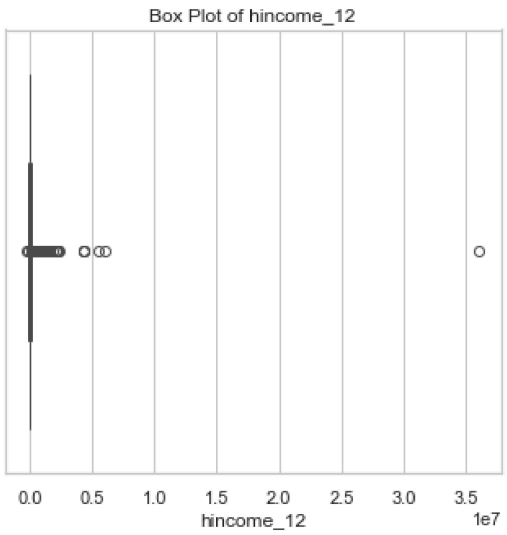
In [ ]:
```
# Cross-Feature Relationships - Scatter plots between income and education leve
sns.pairplot(merged_df, vars=continuous_features + ['composite_score'])
plt.suptitle('Scatter Plot Matrix of Selected Features and Target', y=1.02)
plt.show()
```

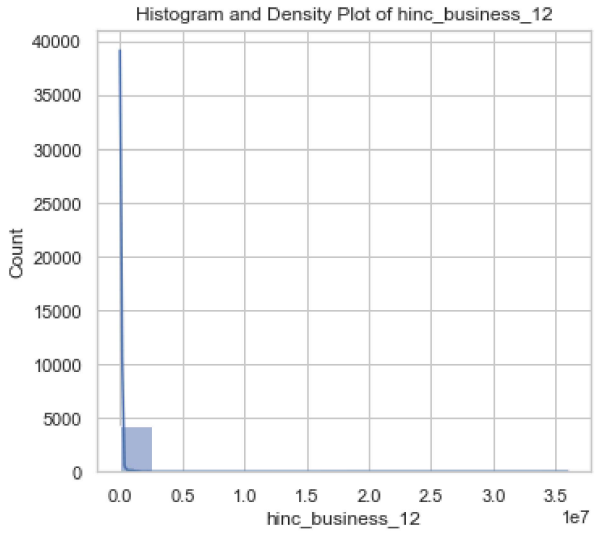


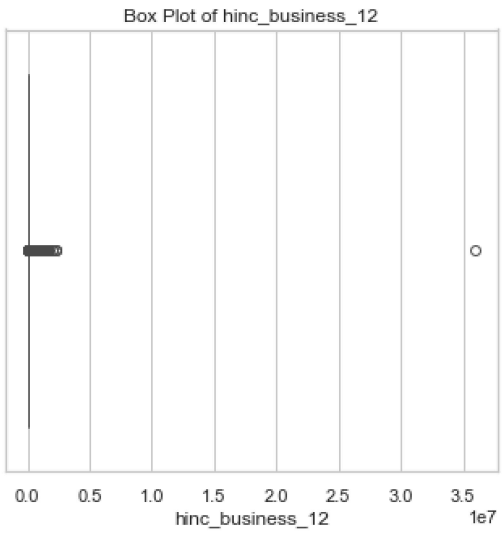
3.7.4 Skew of Univariate Distributions

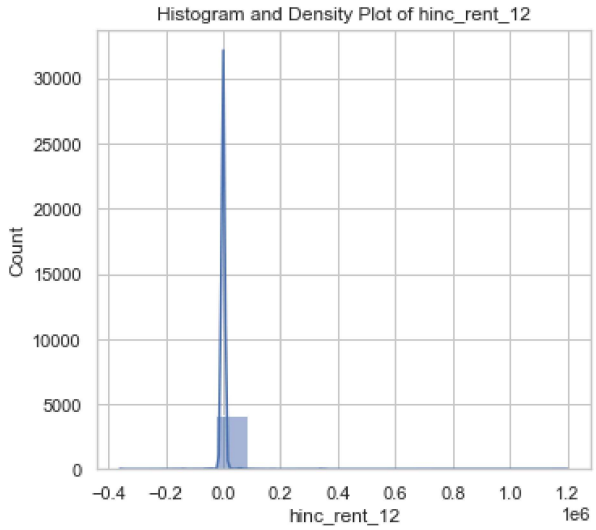
In [ ]:
```
# Skew of Univariate Distributions - Histograms and Density Plots
for col in continuous_features:
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    sns.histplot(merged_df[col].dropna(), kde=True)
    plt.title(f'Histogram and Density Plot of {col}')

    plt.subplot(1, 2, 2)
    sns.boxplot(x=merged_df[col].dropna())
    plt.title(f'Box Plot of {col}')

    plt.show()
```

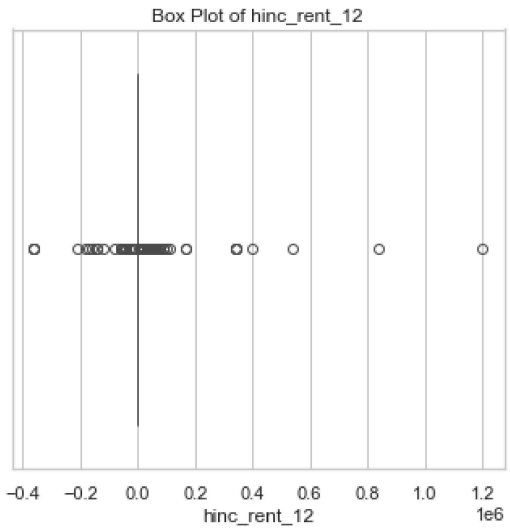












Observations

- Distribution of composite_score for 2016 and 2021:
  - The composite_score distributions for both 2016 and 2021 appear to follow a similar bell-shaped curve, though there is a noticeable increase in the distribution for 2021.
  - Both distributions show a concentration of scores between 100 and 200, indicating this is the most common range for cognitive function scores in the data.
  - There is a slight positive skew, suggesting that fewer individuals have high cognitive scores, with most falling in the lower to mid ranges.
- Correlation Matrix:
  - The correlation matrix highlights a strong positive correlation between hincome_12 (household income), hinc_business_12 (household business income), and hinc_rent_12 (household rental income). This suggests that individuals with high household income tend to have higher income from business and rental sources as well.
  - The composite_score has a weak positive correlation with hincome_12, hinc_business_12, and hinc_rent_12, indicating that household income factors are not strong predictors of cognitive scores in this dataset.
- Scatter Plot Matrix:
  - The scatter plot matrix shows relationships between hincome_12, hinc_business_12, hinc_rent_12, and composite_score.
  - The plots show outliers, especially in the income-related features (hincome_12, hinc_business_12, and hinc_rent_12), with a few extremely high values.
  - Most data points are clustered around lower income values, with a wide distribution of composite_score values for these individuals.
- Univariate Distribution of Income Features:
  - The income features (hincome_12, hinc_business_12, hinc_rent_12) display a high concentration of values close to zero, with a few outliers significantly higher than the main distribution.
  - The Box plots reveal extreme values in all three income-related features, suggesting the presence of outliers that may impact model performance.

- Target Distribution: The composite_score distributions are positively skewed with a higher frequency of scores in the mid-range (100-200), which may affect model performance if not

accounted for.

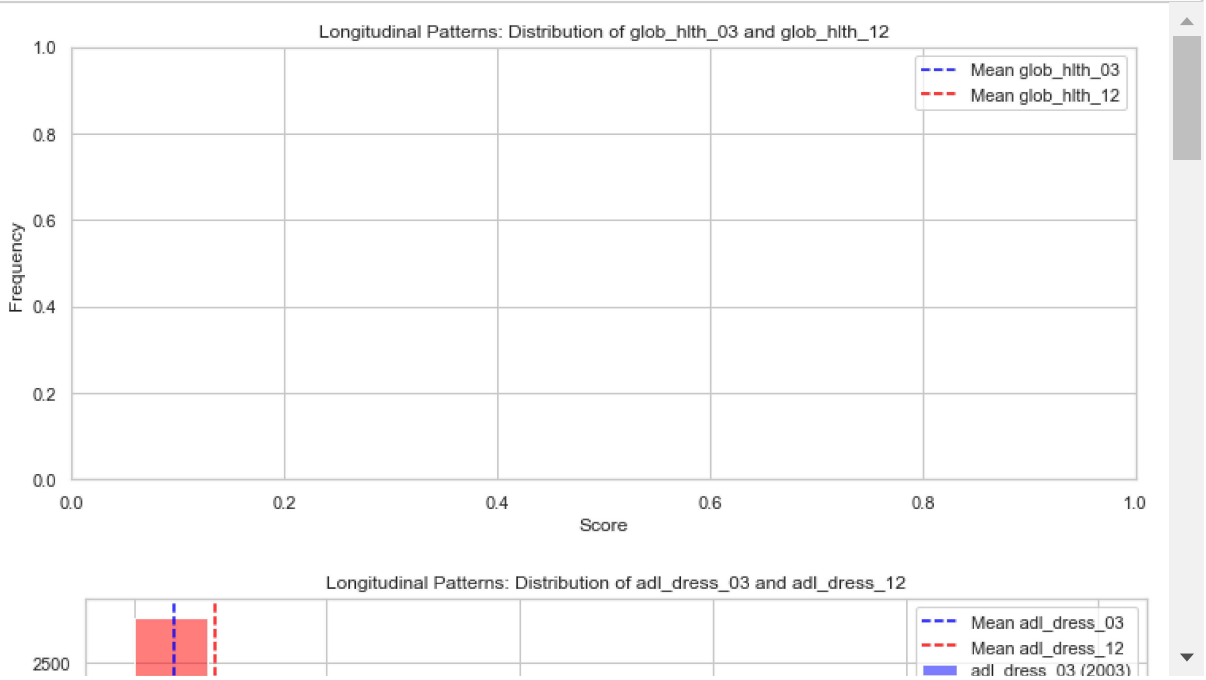
- Income-Related Variables: Although income-related features are correlated with each other, they show little correlation with composite_score, implying limited predictive power for cognitive function scores.
- Outliers: The presence of outliers, especially in income-related features, suggests the need for preprocessing steps, such as scaling or removing outliers, to improve model performance.

3.8 Understand Temporal Components

3.8.1 Longitudinal Patterns

- Investigate differences between _03 and _12 features for temporal patterns or trends.

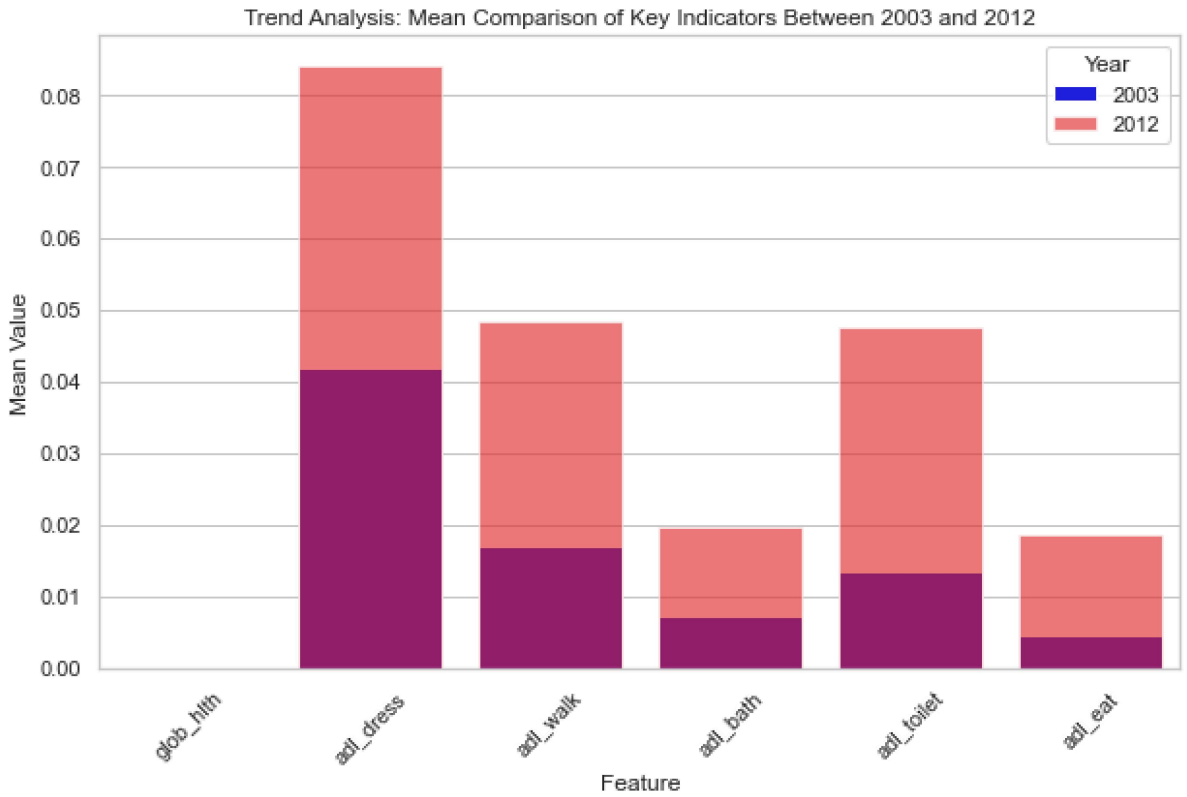
In [ ]:
```
# Define the temporal features
temporal_features = ['glob_hlth', 'adl_dress', 'adl_walk', 'adl_bath', 'adl_to
temporal_03_features = [f"{feat}_03" for feat in temporal_features]
temporal_12_features = [f"{feat}_12" for feat in temporal_features]

# Longitudinal Patterns: Plot distributions for each feature in 2003 and 2012
for feat_03, feat_12 in zip(temporal_03_features, temporal_12_features):
    plt.figure(figsize=(12, 5))

    # Convert columns to numeric, coercing errors to NaN for non-numeric values
    train_features_df[feat_03] = pd.to_numeric(train_features_df[feat_03], err
    train_features_df[feat_12] = pd.to_numeric(train_features_df[feat_12], err

    # Plot distribution for _03 and _12 features
    sns.histplot(train_features_df[feat_03].dropna(), color='blue', kde=True,
    sns.histplot(train_features_df[feat_12].dropna(), color='red', kde=True, la

    # Plot mean lines
    plt.axvline(train_features_df[feat_03].mean(), color='blue', linestyle='--
    plt.axvline(train_features_df[feat_12].mean(), color='red', linestyle='--'

    plt.title(f"Longitudinal Patterns: Distribution of {feat_03} and {feat_12}
    plt.xlabel('Score')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()
```



3.8.2 Trend Analysis

- Compare means or distributions of key indicators like glob_hlth, adl scores (activities of daily living) across timepoints.

In [ ]:
```
#Trend Analysis: Mean Comparison with Data Conversion
# Calculate mean values for _03 and _12 features
mean_values_03 = train_features_df[temporal_03_features].apply(pd.to_numeric, 
mean_values_12 = train_features_df[temporal_12_features].apply(pd.to_numeric, 

# Convert mean values to a DataFrame for easier plotting
mean_values_df = pd.DataFrame({
    'Feature': temporal_features,
    'Mean_03': mean_values_03.values,
    'Mean_12': mean_values_12.values
})

# Trend Analysis: Plot mean values for _03 and _12 features
plt.figure(figsize=(10, 6))
sns.barplot(x='Feature', y='Mean_03', data=mean_values_df, color='blue', label
sns.barplot(x='Feature', y='Mean_12', data=mean_values_df, color='red', label=
plt.title("Trend Analysis: Mean Comparison of Key Indicators Between 2003 and 
plt.ylabel("Mean Value")
plt.legend(title="Year")
plt.xticks(rotation=45)
plt.show()
```



Observations

- Longitudinal Patterns
  - Health and Activity Levels: Across the key adl (Activities of Daily Living) scores for 2003 and 2012, the distributions indicate very low values with some minor right skew. This suggests that a large portion of the participants had low activity limitations in both years.

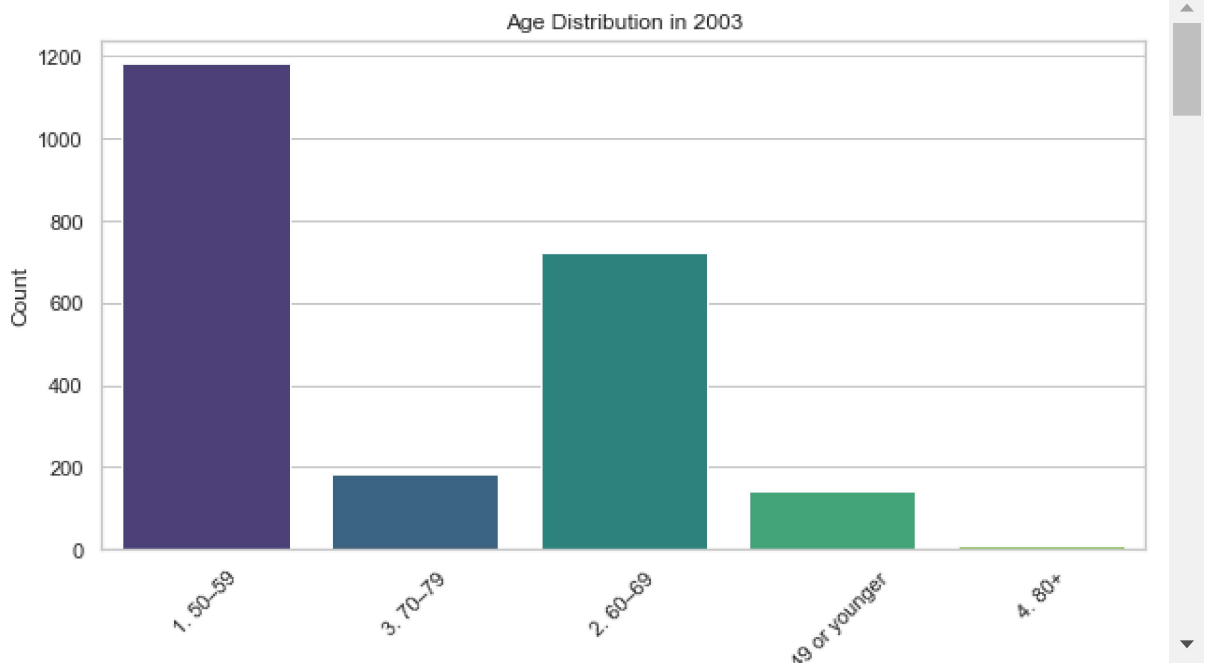
    - Global Health (glob_hlth): There is limited variance in global health scores across the years 2003 and 2012. The distributions are centered around low scores, indicating a high concentration of individuals reporting fewer health issues, though mean values slightly change across time points.
    - Subtle Shifts Over Time: For certain variables like adl_dress, adl_walk, adl_toilet, and adl_eat, there is a slight increase in mean scores from 2003 to 2012. This could reflect aging effects, where individuals report slightly more difficulties in performing daily activities as time progresses.
- Trend Analysis
    - Mean Comparisons: In the bar chart comparing mean values of key indicators, there is a notable rise in values for indicators like adl_dress and adl_walk over time, suggesting an increase in activity limitations from 2003 to 2012.
    - Box Plots for Year Comparisons: The box plots for each indicator (e.g., adl_dress, adl_walk) show a similar trend with minor shifts in the distributions. These visualizations provide further evidence of subtle increases in limitations over time, which align with expected aging-related declines.
- Derived Insights
    - Feature Engineering: Given the changes observed, it may be useful to engineer new features capturing the differences between 2003 and 2012 for each indicator. This could enhance predictive power by highlighting temporal changes.
    - Further Analysis: Additional statistical tests could be conducted to confirm if these temporal shifts are statistically significant. This would help verify if the observed changes are meaningful for modeling purposes.

3.9 Examining Data for Bias and Diversity

**3.9.1 Demographic Distribution:

- Examine distribution across age, gender, education, and urban/rural locality to ensure representative diversity.

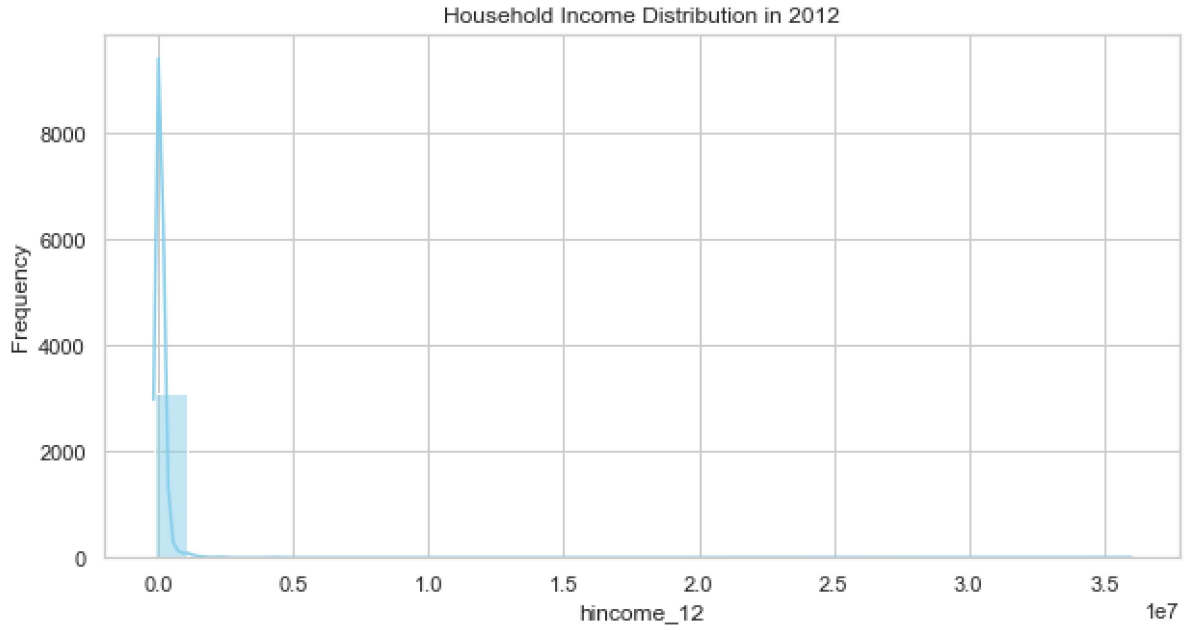
In [ ]:
```
def plot_demographic_distribution(df, column, title):
    plt.figure(figsize=(10, 5))
    sns.countplot(data=df, x=column, palette="viridis")
    plt.title(title)
    plt.xlabel(column)
    plt.ylabel("Count")
    plt.xticks(rotation=45)
    plt.show()

# Plot distributions for key demographic attributes
plot_demographic_distribution(train_features_df, 'age_03', "Age Distribution i
plot_demographic_distribution(train_features_df, 'edu_gru_03', "Education Leve
plot_demographic_distribution(train_features_df, 'urban_03', "Urban/Rural Dist

plot_demographic_distribution(train_features_df, 'age_12', "Age Distribution i
plot_demographic_distribution(train_features_df, 'edu_gru_12', "Education Leve
plot_demographic_distribution(train_features_df, 'urban_12', "Urban/Rural Dist
```

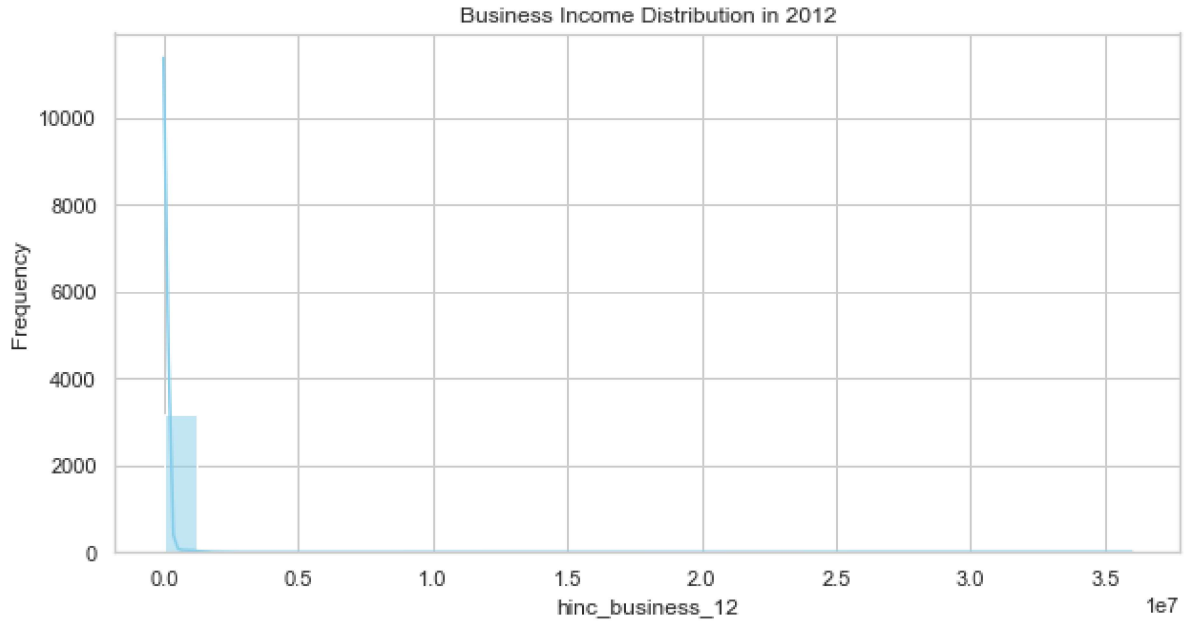


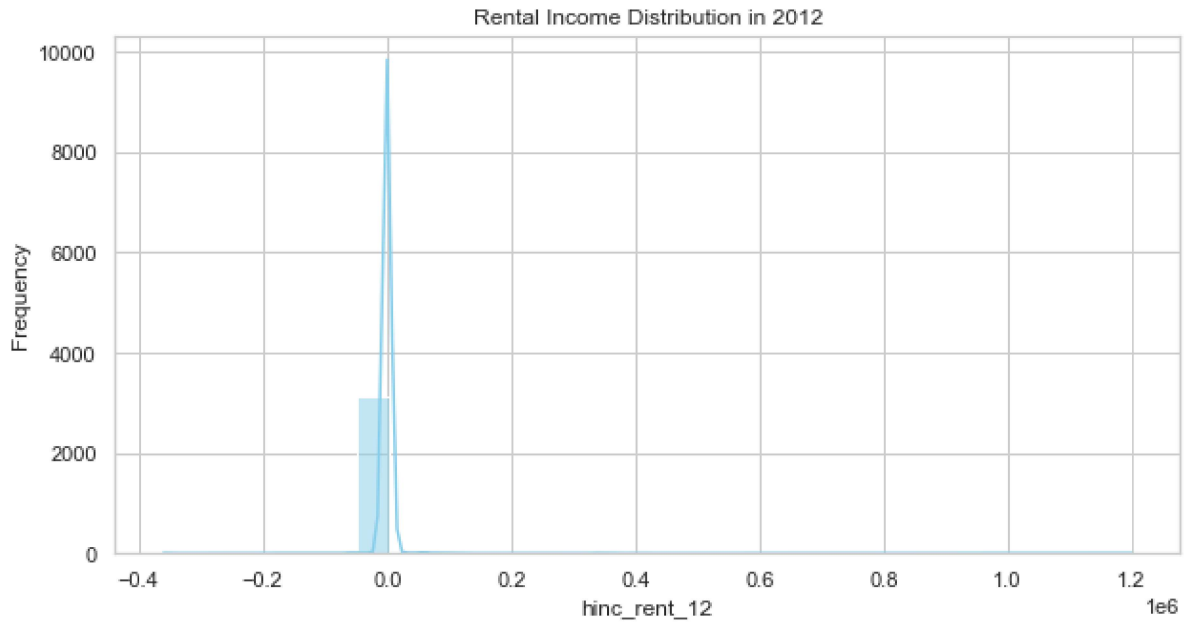
3.9.2 Potential Bias in Features

- Consider initial indications of possible biases (e.g., socioeconomic disparities) that could impact model fairness.

In [ ]:
```
def plot_income_distribution(df, column, title):
    plt.figure(figsize=(10, 5))
    sns.histplot(df[column].dropna(), bins=30, kde=True, color='skyblue')
    plt.title(title)
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.show()

# Plot income distributions to assess socioeconomic diversity
plot_income_distribution(train_features_df, 'hincome_12', "Household Income Di
plot_income_distribution(train_features_df, 'hinc_business_12', "Business Inco
plot_income_distribution(train_features_df, 'hinc_rent_12', "Rental Income Dis
```







3.9.3 Handling Underrepresented Groups

Identify any underrepresented groups that might need special attention for fair and balanced prediction results.

In [ ]:
```
def underrepresented_group_report(df, columns):
    underrepresented_report = {}
    for col in columns:
        counts = df[col].value_counts()
        underrepresented_report[col] = counts[counts < counts.mean()].index.to
    return underrepresented_report

# Define the demographic columns to check
demographic_columns = ['age_03', 'edu_gru_03', 'urban_03', 'age_12', 'edu_gru_

# Get report on underrepresented groups
underrepresented_groups = underrepresented_group_report(train_features_df, dem
print("Underrepresented Groups Report:")
for col, groups in underrepresented_groups.items():
    print(f"{col}: {groups}")
```

```
Underrepresented Groups Report:
age_03: ['3. 70-79', '0. 49 or younger', '4. 80+']
edu_gru_03: ['2. 6 years', '3. 7-9 years', '4. 10+ years']
urban_03: ['0. <100,000']
age_12: ['4. 80+', '0. 49 or younger']
edu_gru_12: ['0. No education', '3. 7-9 years', '4. 10+ years']
urban_12: ['0. <100,000']
```

Observations

Demographic Distribution

- Age
  - In 2003, most individuals were in the age group of 50–59, followed by 60–69. Very few individuals were aged 80+ or 49 or younger.
  - By 2012, the distribution shifted slightly, with the 60–69 group being the largest, while the 80+ and 49 or younger groups remained underrepresented.
- Education
  - In both 2003 and 2012, a significant portion of the population fell into the "1–5 years" education category, with "6 years" and "no education" also prevalent. Higher education levels (10+ years) were underrepresented in both years.
- Urban/Rural
  - In 2003, the majority lived in urban areas (100,000+ population), with rural areas (<100,000) being underrepresented.
  - This trend slightly reversed in 2012, with the urban-rural distribution becoming more balanced, though urban areas remained somewhat dominant.

Potential Bias in Features

- Socioeconomic Factors
  - Income distributions (household, business, and rental) for 2012 show a highly skewed distribution, with most values concentrated at lower income levels and a few very high outliers. This skew could indicate socioeconomic disparities, where the majority have limited income, and a few individuals or families have significantly higher income levels.
- Urban vs. Rural Distribution:
  - The imbalance between urban and rural areas in both years could suggest a potential bias favoring urban populations. This may affect the model's generalization to rural populations if not adequately addressed.

Handling Underrepresented Groups

- Age Groups
  - Certain age groups (49 or younger, 70–79, and 80+) are consistently underrepresented. This could lead to a model that may not perform as well for younger or older individuals.
- Education Levels Higher education levels (10+ years) and some middle education levels (6–9 years) are also underrepresented, which may affect the model’s accuracy across different education backgrounds.
- Rural Populations The rural (<100,000 population) demographic is underrepresented, especially in 2003. Special attention may be needed to ensure the model doesn’t bias predictions toward urban characteristics.

The distribution shows that most individuals are middle-aged, have low to moderate levels of education, and predominantly reside in urban areas.

Income data is highly skewed, indicating significant income inequality, which could introduce bias if not handled appropriately.

There is a need to consider strategies like resampling or applying weights to address the potential underrepresentation of certain groups, especially for age, rural populations, and education levels, to enhance model fairness and ensure equitable predictions across different

3.10 Conclusion - Data Understanding

Data Completeness and Quality

- While the train and submission data are mostly complete, the test features data contains a considerable amount of missing values, particularly in certain attributes (e.g., a16a_12, a22_12).
- This will require specific handling strategies such as imputation or exclusion of high-missing-value columns.

Feature Diversity and Cardinality

- The data contains a variety of features, including demographic, socioeconomic, and health-related attributes.
- Many categorical features exhibit consistent cardinality across years (e.g., marital status, education level), which suggests stable data categories that are suitable for encoding.

Outliers

- Income-related variables show significant outliers, which could impact model performance. These outliers will need careful treatment, potentially through transformation, removal, or scaling.

Temporal Analysis

- Longitudinal changes are observed in health and socioeconomic indicators between 2003 and 2012, which might be relevant for trend-based feature engineering and temporal pattern analysis.

Bias and Representation

- Certain demographics, such as age groups (49 or younger, 80+), rural populations, and specific education levels, are underrepresented.
- This may introduce bias, especially if the model does not account for these distributions.

4. Data Preprocessing

In order to prepare the data for predictive modeling, we will merge the train_features and train_labels dataframes on the 'uid' column, so as to create unique uid-year records.

In [ ]:
```
# Load data
train_features = pd.read_csv('Data/train_features.csv')
train_labels = pd.read_csv('Data/train_labels.csv')

# Merge features and labels
data = train_labels.merge(train_features, on='uid', how='left')

data.head()
```

Out[3]:

	uid	year	composite_score	age_03	urban_03	married_03	n_mar_03	edu_gru_03	n_living_
0	aace	2021	175	NaN	NaN	NaN	NaN	NaN	
1	aanz	2021	206	NaN	NaN	NaN	NaN	NaN	
2	aape	2016	161	NaN	NaN	NaN	NaN	NaN	
3	aape	2021	144	NaN	NaN	NaN	NaN	NaN	
4	aard	2021	104	1. 50–59	1. 100,000+	3. Widowed	1.0	3. 7–9 years	

5 rows × 186 columns

As we had earlier identified that the data contains a considerable amount of missing values, we will drop columns with excessive missing data.

In [ ]:
```
# Drop columns with excessive missing data
missing_threshold = 0.5
missing_percentages = data.isnull().mean()
cols_to_drop = missing_percentages[missing_percentages > missing_threshold].in
data = data.drop(columns=cols_to_drop, errors='ignore')
```

For the remaining columns, we will impute the missing values, using appropriate strategies.

Numerical Variables: Use methods like mean, median, or model-based imputation (e.g., IterativeImputer).

Categorical Variables: Impute using the mode or create a separate category for missing values.

We will also need to convert categorical variables to numerical variables appropriately.

4.1 Identifying numerical and categorical columns

In [ ]:
```
# Identify numerical and categorical columns
numerical_cols = data.select_dtypes(include=['int64', 'float64']).columns.toli
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()
```

Ordinal and nominal categorigal columns ought to be handled separately.

4.2 Separating ordinal and nominal categorical columns

In [ ]:
```
# Identify ordinal variables based on data descriptions
ordinal_variables = [
    'age_03',               # Binned age group
    'edu_gru_03',           # Binned education level
    'n_living_child_03',    # Binned number of living children
    'glob_hlth_03',         # Self-reported global health
    'age_12',               # Binned age group
    'edu_gru_12',           # Binned education level
    'n_living_child_12',    # Binned number of living children
    'glob_hlth_12',         # Self-reported global health
    'bmi_12',               # Binned body mass index
    'decis_famil_12',       # Weight in family decisions
    'decis_personal_12',    # Weight over personal decisions
    'satis_ideal_12',       # Agreement with life satisfaction statements
    'satis_excel_12',
    'satis_fine_12',
    'cosas_imp_12',
    'wouldnt_change_12',
    'memory_12',            # Self-reported memory
    'rameduc_m',            # Mother's education level
    'rafeduc_m',            # Father's education level
    'rrelgimp_03',          # Importance of religion
    'rrelgimp_12',
    'rrfcntx_m_12',         # Frequency of seeing friends/relatives
    'rsocact_m_12',         # Frequency of social activities
    'rrelgwk_12',           # Participation in weekly religious services
    'a34_12'                # English proficiency
]

# Nominal variables are the rest of the categorical variables
nominal_variables = [col for col in categorical_cols if col not in ordinal_var

# Exclude 'uid' from nominal variables
if 'uid' in nominal_variables:
    nominal_variables.remove('uid')

# Output the lists
print(f"Ordinal variables ({len(ordinal_variables)}): {ordinal_variables}\n")
print(f"Nominal variables ({len(nominal_variables)}): {nominal_variables}\n")
```

```
Ordinal variables (25): ['age_03', 'edu_gru_03', 'n_living_child_03', 'glob_h
lth_03', 'age_12', 'edu_gru_12', 'n_living_child_12', 'glob_hlth_12', 'bmi_1
2', 'decis_famil_12', 'decis_personal_12', 'satis_ideal_12', 'satis_excel_1
2', 'satis_fine_12', 'cosas_imp_12', 'wouldnt_change_12', 'memory_12', 'ramed
uc_m', 'rafeduc_m', 'rrelgimp_03', 'rrelgimp_12', 'rrfcntx_m_12', 'rsocact_m_
12', 'rrelgwk_12', 'a34_12']

Nominal variables (11): ['urban_03', 'married_03', 'employment_03', 'urban_1
2', 'married_12', 'employment_12', 'ragender', 'sgender_03', 'sgender_12', 'r
jlocc_m_12', 'j11_12']
```

4.3 Creating a custom mapping for ordinal variables

In [ ]:
```
# Mappings for ordinal variables
age_mapping = {
    '0. 49 or younger': 0,
    '1. 50-59': 1,
    '2. 60-69': 2,
    '3. 70-79': 3,
    '4. 80+': 4,
}

education_mapping = {
    '0. No education': 0,
    '1. 1-5 years': 1,
    '2. 6 years': 2,
    '3. 7-9 years': 3,
    '4. 10+ years': 4,
}

n_living_child_mapping = {
    '0. No children': 0,
    '1. 1 or 2': 1,
    '2. 3 or 4': 2,
    '3. 5 or 6': 3,
    '4. 7+': 4,
}

glob_health_mapping = {
    '1. Excellent': 5,
    '2. Very good': 4,
    '3. Good': 3,
    '4. Fair': 2,
    '5. Poor': 1,
}

bmi_mapping = {
    '1. Underweight': 1,
    '2. Normal weight': 2,
    '3. Overweight': 3,
    '4. Obese': 4,
    '5. Morbidly obese': 5,
}

decis_famil_mapping = {
    '1. Respondent': 1,
    '2. Approximately equal weight': 2,
    '3. Spouse': 3,
}

decis_personal_mapping = {
    '1. A lot': 3,
    '2. A little': 2,
    '3. None': 1
}

agreement_mapping = {
    '1. Agrees': 3,
    '2. Neither agrees nor disagrees': 2,
    '3. Disagrees': 1,
```

```
}

memory_mapping = {
    '1. Excellent': 5,
    '2. Very good': 4,
    '3. Good': 3,
    '4. Fair': 2,
    '5. Poor': 1,
}

parent_education_mapping = {
    '1.None': 1,
    '2.Some primary': 2,
    '3.Primary': 3,
    '4.More than primary': 4,
}

religion_importance_mapping = {
    '1.very important': 3,
    '2.somewhat important': 2,
    '3.not important': 1,
}

frequency_mapping = {
    '1.Almost every day': 9,
    '2.4 or more times a week': 8,
    '3.2 or 3 times a week': 7,
    '4.Once a week': 6,
    '5.4 or more times a month': 5,
    '6.2 or 3 times a month': 4,
    '7.Once a month': 3,
    '8.Almost Never, sporadic': 2,
    '9.Never': 1,
}

religious_services_mapping = {
    '1.Yes': 1,
    '0.No': 0,
}

english_proficiency_mapping = {
    'Yes 1': 1,
    'No 2': 0,
}

# Compile all mappings into a dictionary for easy access
ordinal_mappings = {
    'age_03': age_mapping,
    'age_12': age_mapping,
    'edu_gru_03': education_mapping,
    'edu_gru_12': education_mapping,
    'n_living_child_03': n_living_child_mapping,
    'n_living_child_12': n_living_child_mapping,
    'glob_hlth_03': glob_health_mapping,
    'glob_hlth_12': glob_health_mapping,
    'bmi_12': bmi_mapping,
    'decis_famil_12': decis_famil_mapping,
```

```
    'decis_personal_12': decis_personal_mapping,
    'satis_ideal_12': agreement_mapping,
    'satis_excel_12': agreement_mapping,
    'satis_fine_12': agreement_mapping,
    'cosas_imp_12': agreement_mapping,
    'wouldnt_change_12': agreement_mapping,
    'memory_12': memory_mapping,
    'rameduc_m': parent_education_mapping,
    'rafeduc_m': parent_education_mapping,
    'rrelgimp_03': religion_importance_mapping,
    'rrelgimp_12': religion_importance_mapping,
    'rrfcntx_m_12': frequency_mapping,
    'rsocact_m_12': frequency_mapping,
    'rrelgwk_12': religious_services_mapping,
    'a34_12': english_proficiency_mapping,
}
```

In [ ]:
```
# Function to map ordinal variables
def map_ordinal_variables(X, ordinal_cols, ordinal_mappings):
    X = X.copy()
    for col in ordinal_cols:
        if col in X.columns:
            mapping = ordinal_mappings.get(col, {})
            X[col] = X[col].map(mapping)
    return X
```

5. Feature engineering

Next, we move on to feature engineering.

- We will design custom transformers to capture various aspects of the data, such as temporal changes, education progression, marital transitions, chronic illnesses, and more.
- This approach is likely to enhance the predictive power of our model by incorporating domain knowledge.
- We will also be implementing each feature engineering step as a separate transformer class, which promotes code modularity and reusability.
- We will be creating CustomFeatureEngineer classes and integrating them into a preprocessing pipeline, so as to ensure that all transformations are consistently applied to both training and test data.

5.1 Creating Temporal Features:

Since we have data from 2003 and 2012, we can enhance predictive modeling by creating temporal features that capture changes in individuals' characteristics over time and the duration since the last measurement.

- Change Over Time: For individuals with data from both 2003 and 2012, calculate the change or rate of change in features over time.
- Duration Since Last Measurement: Include the time gap between the last available feature data and the target year.

5.1.1 Identify Features for Change Calculation

We'll focus on numerical and ordinal variables suitable for calculating changes.

In [ ]:
```
# Adjust the temporal features handling to match existing columns
common_features = set([col[:-3] for col in data.columns if col.endswith('_03')
                  set([col[:-3] for col in data.columns if col.endswith('_12')

# Filter common features that are numerical or ordinal
numerical_common_features = [feature for feature in common_features if feature

# Redefine temporal features to only include those present in the dataset
temporal_features = [feature + '_change' for feature in numerical_common_featu

# Ensure numerical_common_features and temporal features are aligned with the d
numerical_common_features = [feature for feature in numerical_common_features

# Verify adjusted temporal features
temporal_features, numerical_common_features
```

Out[9]:
```
(['hosp_change',
  'issste_change',
  'exer_3xwk_change',
  'n_depr_change',
  'n_mar_change',
  'adl_bed_change',
  'restless_change',
  'enjoy_change',
  'decis_personal_change',
  'hinc_assets_change',
  'diabetes_change',
  'happy_change',
  'hrt_attack_change',
  'test_pres_change',
  'energetic_change',
  'cesd_depressed_change',
  'tired_change',
  'out_proc_change',
  'sinc_pension_change',
```

Based on domain knowledge and understanding of how each feature relates to cognitive decline and Alzheimer's disease, we can refine the list of features to include only those where calculating the change over time is meaningful and potentially predictive.

In [ ]:
```
# Selected numerical features for change calculation
numerical_common_features = [
    # Health Conditions
    'hypertension',
    'diabetes',
    'stroke',
    'hrt_attack',
    'cancer',
    'arthritis',
    'resp_ill',

    # Mental Health Indicators
    'depressed',
    'sad',
    'n_depr',
    'cesd_depressed',
    'lonely',
    'restless',
    'tired',
    'hard',
    'happy',
    'enjoy',
    'energetic',

    # Functional Limitations
    'adl_eat',
    'adl_dress',
    'adl_bath',
    'adl_walk',
    'adl_bed',
    'adl_toilet',
    'n_adl',
    'iadl_money',
    'iadl_meals',
    'iadl_shop',
    'iadl_meds',
    'n_iadl',

    # Health Behaviors
    'tobacco',
    'alcohol',
    'exer_3xwk',

    # Number of Illnesses
    'n_illnesses',

    # Economic Factors
    'hincome',
    'rearnings',
    'hinc_assets',
    'hinc_cap',
    'hinc_business',
    'searnings',

    # Health Service Utilization
    'hosp',
    'visit_med',
```

```
]
```

- **Health Conditions:** Onset or progression of chronic diseases between 2003 and 2012 can significantly impact cognitive health.
- **Mental Health:** Changes in depressive symptoms and feelings of loneliness or happiness can be early indicators of cognitive decline.
- **Functional Limitations:** Increasing difficulties with ADLs and IADLs reflect declining functional capacity, often associated with dementia.
- **Health Behaviors:** Changes in smoking, alcohol use, and exercise habits affect overall health and brain function.
- **Number of Illnesses:** An increase in the number of diagnosed conditions can strain the body's resources and impact cognition.
- **Economic Factors:** Significant changes in income can influence stress levels, access to healthcare, and lifestyle, all affecting cognitive health.
- **Health Service Utilization:** Increased hospitalizations or doctor visits may signal worsening health conditions.

In [ ]:
```
# Corresponding change features
temporal_features = [feature + '_change' for feature in numerical_common_featu
```

5.1.2 Creating custom transformer for the temporal features' creation

In [ ]:
```
class TemporalFeatureEngineer(BaseEstimator, TransformerMixin):
    """
    Transformer that creates temporal features such as change over time
    and time gap since last measurement.

    Parameters:
    - numerical_common_features: List of features present in both 2003 and 201
    - ordinal_mappings: Dictionary mapping ordinal variables to numerical value

    Methods:
    - fit: Returns self.
    - transform: Applies temporal feature engineering to the data.
    """
    def __init__(self, numerical_common_features, ordinal_mappings):
        self.numerical_common_features = numerical_common_features
        self.ordinal_mappings = ordinal_mappings

    def fit(self, X, y=None):
        return self  # Nothing to fit

    def transform(self, X):
        X = X.copy()

        # Determine last feature year using vectorized operations
        X['last_feature_year'] = self.get_last_feature_year(X)

        # Calculate time gap between last measurement and target year
        if 'year' in X.columns and 'last_feature_year' in X.columns:
            X['time_gap'] = X['year'] - X['last_feature_year']
        else:
            X['time_gap'] = np.nan  # Handle missing 'year' or 'last_feature_ye

        # Handle ordinal variables and map them
        for feature in self.numerical_common_features:
            base_feature = feature  # e.g., 'age', 'edu_gru'
            col_03 = feature + '_03'
            col_12 = feature + '_12'
            change_col = feature + '_change'

            # Map ordinal variables if necessary
            if base_feature in self.ordinal_mappings:
                mapping = self.ordinal_mappings[base_feature]
                if col_03 in X.columns:
                    X[col_03] = X[col_03].map(mapping)
                if col_12 in X.columns:
                    X[col_12] = X[col_12].map(mapping)

            # Convert columns to numeric (if not already)
            if col_03 in X.columns:
                X[col_03] = pd.to_numeric(X[col_03], errors='coerce')
            if col_12 in X.columns:
                X[col_12] = pd.to_numeric(X[col_12], errors='coerce')

            # Calculate change: 2012 value - 2003 value
            if col_03 in X.columns and col_12 in X.columns:
                # Both columns exist
                valid_mask = X[col_03].notnull() & X[col_12].notnull()
```

```
                X.loc[valid_mask, change_col] = X.loc[valid_mask, col_12] - X.
                X.loc[~valid_mask, change_col] = np.nan
            else:
                X[change_col] = np.nan  # Set change to NaN if both columns do

        # Drop 'last_feature_year' if not needed
        # X.drop(columns=['last_feature_year'], inplace=True)
        # Depending on whether we want to keep it

        return X

    @staticmethod
    def get_last_feature_year(X):
        """
        Determine the last year when data is available for each individual usi
        """
        data_2012_cols = [col for col in X.columns if col.endswith('_12')]
        data_2003_cols = [col for col in X.columns if col.endswith('_03')]

        has_2012_data = X[data_2012_cols].notnull().any(axis=1)
        has_2003_data = X[data_2003_cols].notnull().any(axis=1)

        last_year = pd.Series(np.nan, index=X.index)
        last_year[has_2012_data] = 2012
        last_year[~has_2012_data & has_2003_data] = 2003

        return last_year
```

5.3 More custom transformers for feature engineering based off of domain knowledge

Education

In [ ]:
```python
class EducationProgressionTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that calculates the change in education level between 2003 and

    Methods:
    - fit: Returns self.
    - transform: Computes 'education_transition' as the difference between 'edu
    """
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self  # Nothing to fit

    def transform(self, X):
        X = X.copy()
        if 'edu_gru_03' in X.columns and 'edu_gru_12' in X.columns:
            # Ensure ordinal mappings have been applied
            X['edu_gru_03'] = pd.to_numeric(X['edu_gru_03'], errors='coerce')
            X['edu_gru_12'] = pd.to_numeric(X['edu_gru_12'], errors='coerce')

            valid_mask = X['edu_gru_03'].notnull() & X['edu_gru_12'].notnull()
            X.loc[valid_mask, 'education_transition'] = X.loc[valid_mask, 'edu
            X.loc[~valid_mask, 'education_transition'] = np.nan
        else:
            X['education_transition'] = np.nan
        return X
```

Marital status

In [ ]:
```python
# Marital status columns
married_cols_03 = [col for col in data.columns if 'married_03' in col]
married_cols_12 = [col for col in data.columns if 'married_12' in col]
```

In [ ]:
```
class MaritalTransitionTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that calculates the marital transition between two time points

    Parameters:
    - married_cols_03: List of marital status columns at time point 2003.
    - married_cols_12: List of marital status columns at time point 2012.
    """
    def __init__(self, married_cols_03, married_cols_12):
        self.married_cols_03 = married_cols_03
        self.married_cols_12 = married_cols_12

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        # Ensure columns exist
        if self.married_cols_03[0] in X.columns and self.married_cols_12[0] in
            # Convert to numeric if necessary
            X[self.married_cols_03[0]] = pd.to_numeric(X[self.married_cols_03[
            X[self.married_cols_12[0]] = pd.to_numeric(X[self.married_cols_12[
            # Calculate marital transition
            X['marital_transition'] = X[self.married_cols_12[0]] - X[self.marr
        else:
            X['marital_transition'] = np.nan
        return X
```

Chronic Illnesses

In [ ]:
```
# Chronic illness columns
chronic_illness_cols_03 = ['hypertension_03', 'diabetes_03', 'resp_ill_03', 'a
chronic_illness_cols_12 = ['hypertension_12', 'diabetes_12', 'resp_ill_12', 'a
```

In [ ]:
```python
# Count of chronic illnesses, with change over time
class ChronicIllnessTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that calculates the total number of chronic illnesses for each
    in 2003 and 2012 by summing binary indicators of specific chronic illnesse

    Parameters:
    - chronic_illness_cols_03: List of chronic illness columns for 2003.
    - chronic_illness_cols_12: List of chronic illness columns for 2012.

    Methods:
    - fit: Returns self.
    - transform: Adds 'chronic_illness_count_03' and 'chronic_illness_count_12
    """
    def __init__(self, chronic_illness_cols_03, chronic_illness_cols_12):
        self.chronic_illness_cols_03 = chronic_illness_cols_03
        self.chronic_illness_cols_12 = chronic_illness_cols_12

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        # Adjust columns based on those present in X
        illness_cols_03 = [col for col in self.chronic_illness_cols_03 if col
        illness_cols_12 = [col for col in self.chronic_illness_cols_12 if col

        # Convert to numeric and fill missing values
        if illness_cols_03:
            X[illness_cols_03] = X[illness_cols_03].apply(pd.to_numeric, error
            X['chronic_illness_count_03'] = X[illness_cols_03].sum(axis=1)
        else:
            X['chronic_illness_count_03'] = np.nan

        if illness_cols_12:
            X[illness_cols_12] = X[illness_cols_12].apply(pd.to_numeric, error
            X['chronic_illness_count_12'] = X[illness_cols_12].sum(axis=1)
        else:
            X['chronic_illness_count_12'] = np.nan

        # Calculate change over time if both counts are available
        if 'chronic_illness_count_03' in X.columns and 'chronic_illness_count_
            valid_mask = X['chronic_illness_count_03'].notnull() & X['chronic_
            X.loc[valid_mask, 'chronic_illness_count_change'] = X.loc[valid_ma
            X.loc[~valid_mask, 'chronic_illness_count_change'] = np.nan
        else:
            X['chronic_illness_count_change'] = np.nan
        return X
```

Limitations of activities of daily living

In [ ]:
```
# ADL and IADL columns
adl_cols_03 = ['adl_dress_03', 'adl_walk_03', 'adl_bath_03', 'adl_eat_03', 'ad
adl_cols_12 = ['adl_dress_12', 'adl_walk_12', 'adl_bath_12', 'adl_eat_12', 'ad
iadl_cols_03 = ['iadl_money_03', 'iadl_meds_03', 'iadl_shop_03', 'iadl_meals_0
iadl_cols_12 = ['iadl_money_12', 'iadl_meds_12', 'iadl_shop_12', 'iadl_meals_1
```

In [ ]:
```
# Limitations of Activities of daily living count and progression
class ADLIADLTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that calculates the total number of ADL and IADL limitations
    for each individual in 2003 and 2012, and computes the progression over tir

    Parameters:
    - adl_cols_03: List of ADL columns for 2003.
    - adl_cols_12: List of ADL columns for 2012.
    - iadl_cols_03: List of IADL columns for 2003.
    - iadl_cols_12: List of IADL columns for 2012.

    Methods:
    - fit: Returns self.
    - transform: Adds 'total_adl_limitations_03', 'total_adl_limitations_12',
    """
    def __init__(self, adl_cols_03, adl_cols_12, iadl_cols_03, iadl_cols_12):
        self.adl_cols_03 = adl_cols_03
        self.adl_cols_12 = adl_cols_12
        self.iadl_cols_03 = iadl_cols_03
        self.iadl_cols_12 = iadl_cols_12

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        # Check if required columns are present
        adl_cols_present = all(col in X.columns for col in self.adl_cols_03 + s
        iadl_cols_present = all(col in X.columns for col in self.iadl_cols_03 +
        if adl_cols_present and iadl_cols_present:
            # Calculate total ADL limitations
            X['total_adl_limitations_03'] = X[self.adl_cols_03].sum(axis=1)
            X['total_adl_limitations_12'] = X[self.adl_cols_12].sum(axis=1)
            # Calculate total IADL limitations
            X['total_iadl_limitations_03'] = X[self.iadl_cols_03].sum(axis=1)
            X['total_iadl_limitations_12'] = X[self.iadl_cols_12].sum(axis=1)
            # Calculate progression
            X['adl_iadl_progression'] = (
                (X['total_adl_limitations_12'] + X['total_iadl_limitations_12'
                (X['total_adl_limitations_03'] + X['total_iadl_limitations_03'
            )
        else:
            X['adl_iadl_progression'] = np.nan
        return X
```

Self Reported Health

In [ ]:
```
# Self Reported Health Change
class HealthAssessmentChangeTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that calculates the change in self-reported global health statu
    between 2003 and 2012.

    Methods:
    - fit: Returns self.
    - transform: Computes 'health_self_assessment_change' as the difference bet
    """
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self  # Nothing to fit

    def transform(self, X):
        X = X.copy()
        if 'glob_hlth_03' in X.columns and 'glob_hlth_12' in X.columns:
            # Ensure that both columns are numeric (assuming mapping is done)
            X['glob_hlth_03'] = pd.to_numeric(X['glob_hlth_03'], errors='coerc
            X['glob_hlth_12'] = pd.to_numeric(X['glob_hlth_12'], errors='coerc

            # Identify valid entries where both health assessments are not nul
            valid_mask = X['glob_hlth_03'].notnull() & X['glob_hlth_12'].notnu
            X.loc[valid_mask, 'health_self_assessment_change'] = X.loc[valid_m
            X.loc[~valid_mask, 'health_self_assessment_change'] = np.nan
        else:
            X['health_self_assessment_change'] = np.nan
        return X
```

Mood and depressive sypmtoms

In [ ]:
```
# Define mood columns
positive_mood_cols_03 = ['happy_03', 'enjoy_03', 'energetic_03']
positive_mood_cols_12 = ['happy_12', 'enjoy_12', 'energetic_12']
negative_mood_cols_03 = ['depressed_03', 'restless_03', 'lonely_03', 'sad_03',
negative_mood_cols_12 = ['depressed_12', 'restless_12', 'lonely_12', 'sad_12',
```

In [ ]:
```
# Custom transformer to engineer positive and negative mood scores
class MoodScoreTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that computes aggregate positive and negative mood scores for e
    in 2003 and 2012, and calculates changes over time.

    Parameters:
    - positive_mood_cols_03: List of positive mood indicator columns for 2003.
    - positive_mood_cols_12: List of positive mood indicator columns for 2012.
    - negative_mood_cols_03: List of negative mood indicator columns for 2003.
    - negative_mood_cols_12: List of negative mood indicator columns for 2012.

    Methods:
    - fit: Returns self.
    - transform: Adds mood scores and changes to the dataset.
    """
    def __init__(self, positive_mood_cols_03, positive_mood_cols_12, negative_m
        self.positive_mood_cols_03 = positive_mood_cols_03
        self.positive_mood_cols_12 = positive_mood_cols_12
        self.negative_mood_cols_03 = negative_mood_cols_03
        self.negative_mood_cols_12 = negative_mood_cols_12

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()

        # Adjust columns based on those present in X
        pos_mood_cols_03 = [col for col in self.positive_mood_cols_03 if col in
        neg_mood_cols_03 = [col for col in self.negative_mood_cols_03 if col in
        pos_mood_cols_12 = [col for col in self.positive_mood_cols_12 if col in
        neg_mood_cols_12 = [col for col in self.negative_mood_cols_12 if col in

        # Ensure columns are numeric
        for cols in [pos_mood_cols_03, neg_mood_cols_03, pos_mood_cols_12, neg_
            if cols:
                X[cols] = X[cols].apply(pd.to_numeric, errors='coerce')

        # Fill missing values with 0
        for cols in [pos_mood_cols_03, neg_mood_cols_03, pos_mood_cols_12, neg_
            if cols:
                X[cols] = X[cols].fillna(0)

        # Create aggregate scores for positive and negative moods in 2003
        if pos_mood_cols_03:
            X['positive_mood_score_03'] = X[pos_mood_cols_03].sum(axis=1)
        else:
            X['positive_mood_score_03'] = np.nan

        if neg_mood_cols_03:
            X['negative_mood_score_03'] = X[neg_mood_cols_03].sum(axis=1)
        else:
            X['negative_mood_score_03'] = np.nan

        # Create aggregate scores for positive and negative moods in 2012
        if pos_mood_cols_12:
```

```
        X['positive_mood_score_12'] = X[pos_mood_cols_12].sum(axis=1)
    else:
        X['positive_mood_score_12'] = np.nan

    if neg_mood_cols_12:
        X['negative_mood_score_12'] = X[neg_mood_cols_12].sum(axis=1)
    else:
        X['negative_mood_score_12'] = np.nan

    # Calculate mood changes over time
    valid_pos = X['positive_mood_score_03'].notnull() & X['positive_mood_s
    X['positive_mood_change'] = np.nan
    X.loc[valid_pos, 'positive_mood_change'] = X.loc[valid_pos, 'positive_

    valid_neg = X['negative_mood_score_03'].notnull() & X['negative_mood_s
    X['negative_mood_change'] = np.nan
    X.loc[valid_neg, 'negative_mood_change'] = X.loc[valid_neg, 'negative_

    return X
```

Exercise

In [ ]:
```
# Consistency of exercise tracking
class ConsistentExerciseTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that creates a feature indicating whether an individual consist
    three times per week in both 2003 and 2012.

    Methods:
    - fit: Returns self.
    - transform: Adds 'consistent_exercise' to the dataset.
    """
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self  # Nothing to fit

    def transform(self, X):
        X = X.copy()
        if 'exer_3xwk_03' in X.columns and 'exer_3xwk_12' in X.columns:
            # Convert to numeric
            X['exer_3xwk_03'] = pd.to_numeric(X['exer_3xwk_03'], errors='coerc
            X['exer_3xwk_12'] = pd.to_numeric(X['exer_3xwk_12'], errors='coerc

            # Identify valid entries
            valid_mask = X['exer_3xwk_03'].notnull() & X['exer_3xwk_12'].notnu

            # Initialize the feature with NaN
            X['consistent_exercise'] = np.nan

            # Compute consistent_exercise
            X.loc[valid_mask, 'consistent_exercise'] = (
                (X.loc[valid_mask, 'exer_3xwk_03'] == 1) & (X.loc[valid_mask,
            ).astype(int)
        else:
            X['consistent_exercise'] = np.nan
        return X
```

Alcohol and smoking history

In [ ]:
```
# Lifestyle columns
lifestyle_cols_03 = ['alcohol_03', 'tobacco_03']
lifestyle_cols_12 = ['alcohol_12', 'tobacco_12']
```

In [ ]:
```
# Alcohol and smoking tracking
class LifestyleHealthIndexTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that computes a lifestyle health index for each individual in 2
    by summing up binary indicators for alcohol consumption and tobacco use.

    Parameters:
    - lifestyle_cols_03: List of lifestyle columns for 2003.
    - lifestyle_cols_12: List of lifestyle columns for 2012.

    Methods:
    - fit: Returns self.
    - transform: Adds 'lifestyle_health_index_03', 'lifestyle_health_index_12'
    """
    def __init__(self, lifestyle_cols_03, lifestyle_cols_12):
        self.lifestyle_cols_03 = lifestyle_cols_03
        self.lifestyle_cols_12 = lifestyle_cols_12

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        # Adjust columns based on those present in X
        lifestyle_cols_03 = [col for col in self.lifestyle_cols_03 if col in X
        lifestyle_cols_12 = [col for col in self.lifestyle_cols_12 if col in X

        # Ensure columns are numeric and fill missing values
        for cols in [lifestyle_cols_03, lifestyle_cols_12]:
            if cols:
                X[cols] = X[cols].apply(pd.to_numeric, errors='coerce').fillna

        # Sum lifestyle health indices
        if lifestyle_cols_03:
            X['lifestyle_health_index_03'] = X[lifestyle_cols_03].sum(axis=1)
        else:
            X['lifestyle_health_index_03'] = np.nan

        if lifestyle_cols_12:
            X['lifestyle_health_index_12'] = X[lifestyle_cols_12].sum(axis=1)
        else:
            X['lifestyle_health_index_12'] = np.nan

        # Calculate change over time
        valid_mask = X['lifestyle_health_index_03'].notnull() & X['lifestyle_h
        X['lifestyle_health_index_change'] = np.nan
        X.loc[valid_mask, 'lifestyle_health_index_change'] = X.loc[valid_mask,

        return X
```

Income and insurance

In [ ]:
```
# Income columns
income_cols_03 = ['rearnings_03', 'searnings_03', 'hincome_03', 'hinc_business_
income_cols_12 = ['rearnings_12', 'searnings_12', 'hincome_12', 'hinc_business_

# Insurance columns
insurance_cols_03 = ['imss_03', 'issste_03', 'pem_def_mar_03', 'insur_private_(
insurance_cols_12 = ['imss_12', 'issste_12', 'pem_def_mar_12', 'insur_private_1
```

In [ ]:
```
# Income and insurance coverage
class SocioeconomicFeaturesTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that computes aggregate income and insurance coverage depth fo
    in 2003 and 2012, and determines insurance continuity over time.

    Parameters:
    - income_cols_03: List of income columns for 2003.
    - income_cols_12: List of income columns for 2012.
    - insurance_cols_03: List of insurance columns for 2003.
    - insurance_cols_12: List of insurance columns for 2012.

    Methods:
    - fit: Returns self.
    - transform: Adds socioeconomic features to the dataset.
    """
    def __init__(self, income_cols_03, income_cols_12, insurance_cols_03, insu
        self.income_cols_03 = income_cols_03
        self.income_cols_12 = income_cols_12
        self.insurance_cols_03 = insurance_cols_03
        self.insurance_cols_12 = insurance_cols_12

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        # Adjust columns based on those present in X
        income_cols_03 = [col for col in self.income_cols_03 if col in X.colum
        income_cols_12 = [col for col in self.income_cols_12 if col in X.colum
        insurance_cols_03 = [col for col in self.insurance_cols_03 if col in X
        insurance_cols_12 = [col for col in self.insurance_cols_12 if col in X

        # Convert to numeric and fill missing values
        for cols in [income_cols_03, income_cols_12]:
            if cols:
                X[cols] = X[cols].apply(pd.to_numeric, errors='coerce').fillna
        for cols in [insurance_cols_03, insurance_cols_12]:
            if cols:
                X[cols] = X[cols].apply(pd.to_numeric, errors='coerce').fillna

        # Aggregate income
        if income_cols_03:
            X['aggregate_income_03'] = X[income_cols_03].sum(axis=1)
        else:
            X['aggregate_income_03'] = np.nan
        if income_cols_12:
            X['aggregate_income_12'] = X[income_cols_12].sum(axis=1)
        else:
            X['aggregate_income_12'] = np.nan

        # Insurance coverage depth
        if insurance_cols_03:
            X['insurance_coverage_depth_03'] = X[insurance_cols_03].sum(axis=1
        else:
            X['insurance_coverage_depth_03'] = np.nan
        if insurance_cols_12:
```

```
            X['insurance_coverage_depth_12'] = X[insurance_cols_12].sum(axis=1
        else:
            X['insurance_coverage_depth_12'] = np.nan

        # Insurance continuity
        valid_insurance = X['insurance_coverage_depth_03'].notnull() & X['insu
        X['insurance_continuity'] = np.nan
        X.loc[valid_insurance, 'insurance_continuity'] = ((X.loc[valid_insuran

        # Optionally, calculate changes over time
        # (Include inflation adjustment if applicable)

        return X
```

Social Engagement

In [ ]:
```
# Define social engagement columns
social_engagement_cols = [
    'attends_class_12', 'attends_club_12', 'reads_12', 'games_12', 'table_game
    'comms_tel_comp_12', 'tv_12', 'sewing_12', 'act_mant_12',
    'volunteer_12', 'care_adult_12', 'care_child_12',
    'rrfcntx_m_12', 'rsocact_m_12', 'rrelgwk_12'
]
```

In [ ]:
```python
# Social Engagement score
class SocialEngagementTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that computes a social engagement score for each individual in

    Parameters:
    - social_engagement_cols: List of social engagement columns.

    Methods:
    - fit: Returns self.
    - transform: Adds 'social_engagement_12' to the dataset.
    """
    def __init__(self, social_engagement_cols):
        self.social_engagement_cols = social_engagement_cols

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        # Adjust columns based on those present in X
        social_engagement_cols = [col for col in self.social_engagement_cols i

        # Ensure columns are numeric and fill missing values
        if social_engagement_cols:
            X[social_engagement_cols] = X[social_engagement_cols].apply(pd.to_
            # Sum social engagement score
            X['social_engagement_12'] = X[social_engagement_cols].sum(axis=1)
        else:
            X['social_engagement_12'] = np.nan
        return X
```

Healthcare

In [ ]:
```python
# Preventive care and health service usage columns
preventive_care_cols_03 = ['test_chol_03', 'test_tuber_03', 'test_diab_03', 't
preventive_care_cols_12 = ['test_chol_12', 'test_tuber_12', 'test_diab_12', 't
health_service_usage_cols_03 = ['visit_med_03', 'out_proc_03', 'visit_dental_0
health_service_usage_cols_12 = ['visit_med_12', 'out_proc_12', 'visit_dental_1
```

In [ ]:
```
# Custom transformer to create preventive care index and health services usage
class HealthServicesTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer that creates indices for preventive care and health service usa

    Parameters:
    - preventive_care_cols_03: List of preventive care columns for 2003.
    - preventive_care_cols_12: List of preventive care columns for 2012.
    - health_service_usage_cols_03: List of health service usage columns for 20
    - health_service_usage_cols_12: List of health service usage columns for 20

    Methods:
    - fit: Returns self.
    - transform: Adds indices and changes to the dataset.
    """
    def __init__(self, preventive_care_cols_03, preventive_care_cols_12, healt
        self.preventive_care_cols_03 = preventive_care_cols_03
        self.preventive_care_cols_12 = preventive_care_cols_12
        self.health_service_usage_cols_03 = health_service_usage_cols_03
        self.health_service_usage_cols_12 = health_service_usage_cols_12

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        # Adjust columns based on those present in X
        preventive_care_cols_03 = [col for col in self.preventive_care_cols_03
        preventive_care_cols_12 = [col for col in self.preventive_care_cols_12
        health_service_usage_cols_03 = [col for col in self.health_service_usag
        health_service_usage_cols_12 = [col for col in self.health_service_usag

        # Convert to numeric and fill missing values
        for cols in [preventive_care_cols_03, preventive_care_cols_12, health_
            if cols:
                X[cols] = X[cols].apply(pd.to_numeric, errors='coerce').fillna

        # Create preventive care index for 2003 and 2012
        if preventive_care_cols_03:
            X['preventive_care_index_03'] = X[preventive_care_cols_03].sum(axis
        else:
            X['preventive_care_index_03'] = np.nan

        if preventive_care_cols_12:
            X['preventive_care_index_12'] = X[preventive_care_cols_12].sum(axis
        else:
            X['preventive_care_index_12'] = np.nan

        # Create health service usage score for 2003 and 2012
        if health_service_usage_cols_03:
            X['health_service_usage_03'] = X[health_service_usage_cols_03].sum
        else:
            X['health_service_usage_03'] = np.nan

        if health_service_usage_cols_12:
            X['health_service_usage_12'] = X[health_service_usage_cols_12].sum
        else:
```

```
            X['health_service_usage_12'] = np.nan

        # Calculate changes between 2003 and 2012
        valid_preventive = X['preventive_care_index_03'].notnull() & X['preven
        X['preventive_care_change'] = np.nan
        X.loc[valid_preventive, 'preventive_care_change'] = X.loc[valid_preven

        valid_usage = X['health_service_usage_03'].notnull() & X['health_servi
        X['health_service_usage_change'] = np.nan
        X.loc[valid_usage, 'health_service_usage_change'] = X.loc[valid_usage,

        return X
```

Now lets consolidate all custom feature engineering transformers into a single class (CustomFeatureEngineer) that applies them sequentially during the preprocessing pipeline.

In [ ]:
```
# Create a custom transformer that applies all custom transformations
class CustomFeatureEngineer(BaseEstimator, TransformerMixin):
    def __init__(self, numerical_common_features, ordinal_mappings, married_co
                 chronic_illness_cols_03, chronic_illness_cols_12, adl_cols_03
                 iadl_cols_03, iadl_cols_12, positive_mood_cols_03, positive_m
                 negative_mood_cols_03, negative_mood_cols_12, lifestyle_cols_
                 income_cols_03, income_cols_12, insurance_cols_03, insurance_
                 preventive_care_cols_03, preventive_care_cols_12, health_serv

        # Assign parameters to instance variables
        self.numerical_common_features = numerical_common_features
        self.ordinal_mappings = ordinal_mappings
        self.married_cols_03 = married_cols_03
        self.married_cols_12 = married_cols_12
        self.chronic_illness_cols_03 = chronic_illness_cols_03
        self.chronic_illness_cols_12 = chronic_illness_cols_12
        self.adl_cols_03 = adl_cols_03
        self.adl_cols_12 = adl_cols_12
        self.iadl_cols_03 = iadl_cols_03
        self.iadl_cols_12 = iadl_cols_12
        self.positive_mood_cols_03 = positive_mood_cols_03
        self.positive_mood_cols_12 = positive_mood_cols_12
        self.negative_mood_cols_03 = negative_mood_cols_03
        self.negative_mood_cols_12 = negative_mood_cols_12
        self.lifestyle_cols_03 = lifestyle_cols_03
        self.lifestyle_cols_12 = lifestyle_cols_12
        self.income_cols_03 = income_cols_03
        self.income_cols_12 = income_cols_12
        self.insurance_cols_03 = insurance_cols_03
        self.insurance_cols_12 = insurance_cols_12
        self.social_engagement_cols = social_engagement_cols
        self.preventive_care_cols_03 = preventive_care_cols_03
        self.preventive_care_cols_12 = preventive_care_cols_12
        self.health_service_usage_cols_03 = health_service_usage_cols_03
        self.health_service_usage_cols_12 = health_service_usage_cols_12

        # Initialize all custom transformers
        self.temporal_features = TemporalFeatureEngineer(numerical_common_feat
        self.education_progression = EducationProgressionTransformer()
        self.marital_transition = MaritalTransitionTransformer(married_cols_03
        self.chronic_illness = ChronicIllnessTransformer(chronic_illness_cols_
        self.adl_iadl = ADLIADLTransformer(adl_cols_03, adl_cols_12, iadl_cols
        self.health_assessment_change = HealthAssessmentChangeTransformer()
        self.mood_score = MoodScoreTransformer(positive_mood_cols_03, positive
        self.consistent_exercise = ConsistentExerciseTransformer()
        self.lifestyle_health_index = LifestyleHealthIndexTransformer(lifestyl
        self.socioeconomic_features = SocioeconomicFeaturesTransformer(income_
        self.social_engagement = SocialEngagementTransformer(social_engagement
        self.health_services = HealthServicesTransformer(preventive_care_cols_

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        X = self.temporal_features.transform(X)
        X = self.education_progression.transform(X)
```

```
        X = self.marital_transition.transform(X)
        X = self.chronic_illness.transform(X)
        X = self.adl_iadl.transform(X)
        X = self.health_assessment_change.transform(X)
        X = self.mood_score.transform(X)
        X = self.consistent_exercise.transform(X)
        X = self.lifestyle_health_index.transform(X)
        X = self.socioeconomic_features.transform(X)
        X = self.social_engagement.transform(X)
        X = self.health_services.transform(X)
        return X
```

5.4 Interaction Terms

Now we create a custom transformer to generate interaction terms between variables that may be related.

In [ ]:
```
class InteractionTermsTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()

        # Interaction: Health service usage change * Lifestyle health index (2
        if 'health_service_usage_change' in X.columns and 'lifestyle_health_in
            X['health_lifestyle_interaction'] = (
                X['health_service_usage_change'] * X['lifestyle_health_index_1
            )

        # Interaction: Education transition * Aggregate income (2012)
        if 'education_transition' in X.columns and 'aggregate_income_12' in X.
            X['education_income_interaction'] = (
                X['education_transition'] * X['aggregate_income_12']
            )

        # Interaction: Social engagement (2012) * Positive mood change
        if 'social_engagement_12' in X.columns and 'positive_mood_change' in X
            X['social_mood_interaction'] = (
                X['social_engagement_12'] * X['positive_mood_change']
            )

        # Interaction: Preventive care change * Chronic illness count (2012)
        if 'preventive_care_change' in X.columns and 'chronic_illness_count_12
            X['preventive_chronic_interaction'] = (
                X['preventive_care_change'] * X['chronic_illness_count_12']
            )

        return X
```

6. Splitting data into training, validation and testing sets

Now we define a function to split the dataset into training, validation, and test sets using GroupKFold, ensuring that data points with the same group identifier (uid) are kept together in the same split.

This approach ensures that the validation and test sets are also split in a group-aware manner, preventing data leakage.

In [ ]:
```python
from sklearn.model_selection import GroupKFold, GroupShuffleSplit

def split_data(data, features, target):
    """
    Splits the dataset into training, validation, and test sets using group-awa

    Parameters:
    - data: pandas DataFrame containing the dataset, including a 'uid' column
    - features: list of feature column names.
    - target: target variable column name.

    Returns:
    - X_train, X_val, X_test: Feature sets for training, validation, and testi
    - y_train, y_val, y_test: Target variables for training, validation, and te
    """
    # Ensure 'uid' column exists
    if 'uid' not in data.columns:
        raise ValueError("The 'uid' column is missing from the dataset.")

    # Define groups based on 'uid'
    groups = data['uid']

    # Prepare the data
    X = data[features]
    y = data[target]

    # Initialize GroupKFold
    gkf = GroupKFold(n_splits=5)

    # Generate indices for splits
    splits = gkf.split(X, y, groups=groups)

    # For demonstration, take the first split
    train_idx, temp_idx = next(splits)
    X_train, X_temp = X.iloc[train_idx], X.iloc[temp_idx]
    y_train, y_temp = y.iloc[train_idx], y.iloc[temp_idx]
    groups_temp = groups.iloc[temp_idx]

    # Split temp into validation and test sets using group-aware splitting
    gss = GroupShuffleSplit(n_splits=1, test_size=0.5, random_state=42)
    val_idx, test_idx = next(gss.split(X_temp, y_temp, groups=groups_temp))
    X_val, X_test = X_temp.iloc[val_idx], X_temp.iloc[test_idx]
    y_val, y_test = y_temp.iloc[val_idx], y_temp.iloc[test_idx]

    return X_train, X_val, X_test, y_train, y_val, y_test
```

7. Preprocessing Pipeline

Now we finalize the preprocessing pipeline by adding all the needed variables

In [ ]:
```
def get_preprocessing_pipeline(data, ordinal_mappings):
    """
    Create a preprocessing pipeline

    Parameters:
    - data: pandas DataFrame containing the data.
    - ordinal_mappings: dictionary of mappings for ordinal variables.

    Returns:
    - preprocessor: scikit-learn Pipeline object.
    """

    # Identify numerical and categorical columns
    numerical_cols = data.select_dtypes(include=['int64', 'float64']).columns.
    numerical_cols = [col for col in numerical_cols if col != 'composite_score

    categorical_cols = data.select_dtypes(include=['object']).columns.tolist()

    # Identify ordinal and nominal variables
    ordinal_cols = list(ordinal_mappings.keys())
    nominal_cols = [col for col in categorical_cols if col not in ordinal_cols

    # Define the lists of columns required for custom transformers
    numerical_common_features = [col[:-3] for col in data.columns if col.endsw

    married_cols_03 = ['married_03']
    married_cols_12 = ['married_12']

    chronic_illness_cols_03 = ['hypertension_03', 'diabetes_03', 'resp_ill_03'
    chronic_illness_cols_03 = [col for col in chronic_illness_cols_03 if col i
    chronic_illness_cols_12 = [col.replace('_03', '_12') for col in chronic_il

    adl_cols_03 = ['adl_dress_03', 'adl_walk_03', 'adl_bath_03', 'adl_eat_03',
    adl_cols_03 = [col for col in adl_cols_03 if col in data.columns]
    adl_cols_12 = [col.replace('_03', '_12') for col in adl_cols_03 if col.rep

    iadl_cols_03 = ['iadl_money_03', 'iadl_meds_03', 'iadl_shop_03', 'iadl_mea
    iadl_cols_03 = [col for col in iadl_cols_03 if col in data.columns]
    iadl_cols_12 = [col.replace('_03', '_12') for col in iadl_cols_03 if col.r

    positive_mood_cols_03 = ['happy_03', 'enjoy_03', 'energetic_03']
    positive_mood_cols_03 = [col for col in positive_mood_cols_03 if col in da
    positive_mood_cols_12 = [col.replace('_03', '_12') for col in positive_moo

    negative_mood_cols_03 = ['depressed_03', 'restless_03', 'lonely_03', 'sad_
    negative_mood_cols_03 = [col for col in negative_mood_cols_03 if col in da
    negative_mood_cols_12 = [col.replace('_03', '_12') for col in negative_moo

    lifestyle_cols_03 = ['alcohol_03', 'tobacco_03']
    lifestyle_cols_03 = [col for col in lifestyle_cols_03 if col in data.colum
    lifestyle_cols_12 = [col.replace('_03', '_12') for col in lifestyle_cols_0

    income_cols_03 = ['rearnings_03', 'searnings_03', 'hincome_03', 'hinc_busi
    income_cols_03 = [col for col in income_cols_03 if col in data.columns]
    income_cols_12 = [col.replace('_03', '_12') for col in income_cols_03 if c

    insurance_cols_03 = ['imss_03', 'issste_03', 'pem_def_mar_03', 'insur_priv
```

```
insurance_cols_03 = [col for col in insurance_cols_03 if col in data.columm
insurance_cols_12 = [col.replace('_03', '_12') for col in insurance_cols_0

social_engagement_cols = [
    'attends_class_12', 'attends_club_12', 'reads_12', 'games_12', 'table_
    'comms_tel_comp_12', 'tv_12', 'sewing_12', 'act_mant_12',
    'volunteer_12', 'care_adult_12', 'care_child_12',
    'rrfcntx_m_12', 'rsocact_m_12', 'rrelgwk_12'
]
social_engagement_cols = [col for col in social_engagement_cols if col in

preventive_care_cols_03 = ['test_chol_03', 'test_tuber_03', 'test_diab_03'
preventive_care_cols_03 = [col for col in preventive_care_cols_03 if col i
preventive_care_cols_12 = [col.replace('_03', '_12') for col in preventive_

health_service_usage_cols_03 = ['visit_med_03', 'out_proc_03', 'visit_dent
health_service_usage_cols_03 = [col for col in health_service_usage_cols_0
health_service_usage_cols_12 = [col.replace('_03', '_12') for col in healt
if 'hosp_12' in data.columns:
    health_service_usage_cols_12.append('hosp_12')

# Ordinal Mapper Transformer
ordinal_mapper_transformer = FunctionTransformer(
    map_ordinal_variables,
    kw_args={'ordinal_cols': ordinal_cols, 'ordinal_mappings': ordinal_mapp
)

# Nominal Transformer
nominal_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='Missing')),
    ('onehot', OneHotEncoder(drop='first', handle_unknown='ignore'))
])

interaction_transformer = InteractionTermsTransformer()

# Instantiate Custom Transformers
custom_feature_engineer = CustomFeatureEngineer(
    numerical_common_features, ordinal_mappings,
    married_cols_03, married_cols_12,
    chronic_illness_cols_03, chronic_illness_cols_12,
    adl_cols_03, adl_cols_12,
    iadl_cols_03, iadl_cols_12,
    positive_mood_cols_03, positive_mood_cols_12,
    negative_mood_cols_03, negative_mood_cols_12,
    lifestyle_cols_03, lifestyle_cols_12,
    income_cols_03, income_cols_12,
    insurance_cols_03, insurance_cols_12,
    social_engagement_cols,
    preventive_care_cols_03, preventive_care_cols_12,
    health_service_usage_cols_03, health_service_usage_cols_12
)

# Define the new numerical columns created by custom transformers
new_numerical_cols = [
    'time_gap',
    'education_transition',
    'marital_transition',
```

```
        'chronic_illness_count_03', 'chronic_illness_count_12',
        'total_adl_limitations_03', 'total_adl_limitations_12',
        'total_iadl_limitations_03', 'total_iadl_limitations_12',
        'adl_iadl_progression',
        'health_self_assessment_change',
        'positive_mood_score_03', 'positive_mood_score_12', 'positive_mood_char
        'negative_mood_score_03', 'negative_mood_score_12', 'negative_mood_char
        'consistent_exercise',
        'lifestyle_health_index_03', 'lifestyle_health_index_12',
        'aggregate_income_03', 'aggregate_income_12',
        'insurance_coverage_depth_03', 'insurance_coverage_depth_12', 'insuran
        'social_engagement_12',
        'preventive_care_index_03', 'preventive_care_index_12', 'preventive_ca
        'health_service_usage_03', 'health_service_usage_12', 'health_service_
    ] + [f"{feature}_change" for feature in numerical_common_features]

    all_numerical_cols = numerical_cols + new_numerical_cols
    all_numerical_cols = list(set(all_numerical_cols))

    # Define the complete preprocessing pipeline
    preprocessor = Pipeline(steps=[
        ('ordinal_mapper', ordinal_mapper_transformer),
        ('custom_feature_engineering', custom_feature_engineer),
        ('interaction_terms', interaction_transformer),
        ('preprocessing', ColumnTransformer(transformers=[
            ('num', Pipeline(steps=[
                ('imputer', SimpleImputer(strategy='mean')),
                ('scaler', StandardScaler())
            ]), all_numerical_cols + ordinal_cols),
            ('nom', nominal_transformer, nominal_cols)
        ], remainder='drop'))
    ])

    return preprocessor
```

7.1 Calling the preprocessor

In [ ]:
```
# Create the preprocessing pipeline
preprocessor = get_preprocessing_pipeline(data, ordinal_mappings)
```

8. Splitting the data

In [ ]:
```
# Define features and target
features = data.columns.drop(['composite_score', 'uid'])
target = 'composite_score'

# Split the data using the defined function
X_train, X_val, X_test, y_train, y_val, y_test = split_data(data, features, ta
```

9. Modeling

Now that our preprocessor is ready, we can begin modeling

We will start by creating a function that integrates pipelines to train and evaluate models we choose.

In [ ]:
```python
def build_and_evaluate_model(model, param_grid, model_name):
    """
    Builds a pipeline with the given model, performs hyperparameter tuning,
    and evaluates the model on the validation set.

    Parameters:
    - model: The regression model to use.
    - param_grid: Dictionary of hyperparameters for GridSearchCV.
    - model_name: Name of the model (string).

    Returns:
    - best_estimator: The best estimator from GridSearchCV.
    """
    from sklearn.pipeline import Pipeline

    # Create a pipeline with preprocessing and the model
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('feature_selection', SelectKBest(score_func=f_regression, k=50)),
        ('model', model)
    ])

    # Set up GridSearchCV
    grid_search = GridSearchCV(
        pipeline, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-
    )

    # Fit the model
    grid_search.fit(X_train, y_train)

    # Best estimator
    best_estimator = grid_search.best_estimator_

    # Predictions on validation set
    y_pred = best_estimator.predict(X_val)

    # Evaluate the model
    rmse = np.sqrt(mean_squared_error(y_val, y_pred))
    r2 = r2_score(y_val, y_pred)

    print(f"\nModel: {model_name}")
    print(f"Best Parameters: {grid_search.best_params_}")
    print(f"Validation RMSE: {rmse:.4f}")
    print(f"Validation R²: {r2:.4f}")

    return best_estimator
```

9.1 Defining models and hyperparameters

We will start with the models listed below to have a baseline and see which kinds of models are suited for this dataset.

In [ ]:
```
# Define models and their hyperparameters
models = [
    ('Linear Regression', LinearRegression(), {}),
    ('Ridge Regression', Ridge(), {
        'model__alpha': [0.1, 1.0, 10.0, 100.0],
        'model__solver': ['auto', 'svd', 'cholesky', 'lsqr']
    }),
    ('Lasso Regression', Lasso(max_iter=10000), {
        'model__alpha': [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0]
    }),
    ('ElasticNet Regression', ElasticNet(max_iter=10000), {
        'model__alpha': [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0],
        'model__l1_ratio': [0.1, 0.5, 0.9]
    }),
    ('Support Vector Regressor', SVR(), {
        'model__C': [0.1, 1.0, 10.0],
        'model__epsilon': [0.01, 0.1, 1.0],
        'model__kernel': ['linear', 'rbf']
    }),
    ('Random Forest Regressor', RandomForestRegressor(random_state=42), {
        'model__n_estimators': [100, 200],
        'model__max_depth': [None, 10, 20],
        'model__min_samples_split': [2, 5],
        'model__min_samples_leaf': [1, 2]
    }),
    ('Gradient Boosting Regressor', GradientBoostingRegressor(random_state=42)
        'model__n_estimators': [100, 200],
        'model__learning_rate': [0.01, 0.1],
        'model__max_depth': [3, 5]
    })
]
```

9.2 Training and evaluating baseline models

Now we loop through the defined models, train them and evaluate them using the validation dataset.

In [ ]:
```
# Train and evaluate models
best_models = {}

for model_name, model, param_grid in models:
    print(f"Training and evaluating: {model_name}")
    best_model = build_and_evaluate_model(model, param_grid, model_name)
    best_models[model_name] = best_model
```

```
Training and evaluating: Linear Regression
Fitting 5 folds for each of 1 candidates, totalling 5 fits

/var/folders/lq/_1mfm1_x15s4jd56sgz3z4d00000gn/T/ipykernel_12972/236482813
7.py:58: PerformanceWarning: DataFrame is highly fragmented.  This is usual
ly the result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
/var/folders/lq/_1mfm1_x15s4jd56sgz3z4d00000gn/T/ipykernel_12972/236482813
7.py:58: PerformanceWarning: DataFrame is highly fragmented.  This is usual
ly the result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
/var/folders/lq/_1mfm1_x15s4jd56sgz3z4d00000gn/T/ipykernel_12972/236482813
7.py:58: PerformanceWarning: DataFrame is highly fragmented.  This is usual
ly the result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
/var/folders/lq/_1mfm1_x15s4jd56sgz3z4d00000gn/T/ipykernel_12972/236482813
7.py:58: PerformanceWarning: DataFrame is highly fragmented.  This is usual
```

In [ ]:
```
# Compare results
print("\nModel Evaluation Summary:")
for model_name, model in best_models.items():
    y_pred = model.predict(X_val)
    rmse = np.sqrt(mean_squared_error(y_val, y_pred))
    r2 = r2_score(y_val, y_pred)
    print(f"{model_name} -> RMSE: {rmse:.4f}, R²: {r2:.4f}")
```

```
Model Evaluation Summary:
Linear Regression -> RMSE: 41.3544, R²: 0.5492
Ridge Regression -> RMSE: 41.2959, R²: 0.5505
Lasso Regression -> RMSE: 41.7082, R²: 0.5414
ElasticNet Regression -> RMSE: 41.5797, R²: 0.5443
Support Vector Regressor -> RMSE: 42.0827, R²: 0.5332
Random Forest Regressor -> RMSE: 41.0365, R²: 0.5561
Gradient Boosting Regressor -> RMSE: 40.6148, R²: 0.5652
```

These results indicate that the Gradient Boosting Regressor performs the best in terms of RMSE (40.6148) and R^2 (0.5652), followed by the Random Forest Regressor.

However, the performance of all models is moderately close, suggesting room for improvement.

Strategies for Improving RMSE:

1. Further Feature Engineering:

- **Interaction Terms:** Introduce interaction terms between key features (e.g., health and social engagement indices) to capture complex relationships.
- **Polynomial Features:** Experiment with polynomial transformations of key numerical features.

2. Further Hyperparameter Tuning:

- **Gradient Boosting:** Increase the number of iterations (n_estimators) and tune learning rates. Fine-tune other parameters like subsample and max_features.
- **Random Forest:** Tune the maximum number of features (max_features) or use feature importance to guide pruning.
- Adjust regularization parameters (e.g., min_samples_split, min_samples_leaf, max_depth) to prevent overfitting.

3. Ensemble Methods:

- Combine the predictions of Gradient Boosting, Random Forest, and Linear models to leverage their complementary strengths. Techniques like stacking regressors or blending can reduce prediction variance.

4. Dimensionality Reduction:

- Use Principal Component Analysis (PCA) or Feature Selection (e.g., Recursive Feature Elimination) to reduce noise by eliminating irrelevant or redundant features.

9.3 Hyperparameter tuning

Expanded Hyperparameter Grids for Tree Models

In [ ]:
```
# Random Forest Regressor - Expanded Grid
rf_model = RandomForestRegressor(random_state=42)
rf_params_expanded = {
    'model__n_estimators': [100, 200, 500],
    'model__max_depth': [None, 10, 20, 30],
    'model__min_samples_split': [2, 5, 10],
    'model__min_samples_leaf': [1, 2, 4]
}

# Gradient Boosting Regressor - Expanded Grid
gbr_model = GradientBoostingRegressor(random_state=42)
gbr_params_expanded = {
    'model__n_estimators': [100, 200, 500],
    'model__learning_rate': [0.01, 0.05, 0.1],
    'model__max_depth': [3, 5, 7],
    'model__subsample': [0.8, 1.0]
}

# XGBoost Regressor
xgb_model = xgb.XGBRegressor(random_state=42)
xgb_params = {
    'model__n_estimators': [100, 200, 500],
    'model__learning_rate': [0.01, 0.05, 0.1],
    'model__max_depth': [3, 5, 7],
    'model__subsample': [0.8, 1.0],
    'model__colsample_bytree': [0.8, 1.0]
}

# LightGBM Regressor
lgbm_model = lgb.LGBMRegressor(random_state=42)
lgbm_params = {
    'model__n_estimators': [100, 200, 500],
    'model__learning_rate': [0.01, 0.05, 0.1],
    'model__num_leaves': [31, 50, 70],
    'model__subsample': [0.8, 1.0],
    'model__colsample_bytree': [0.8, 1.0]
}
```

Tuning hyperparameters

In [ ]:
```
# Dictionary to store best estimators
best_estimators = {}

# Random Forest Regressor - Expanded
best_rf_expanded = build_and_evaluate_model(rf_model, rf_params_expanded, 'Ran
best_estimators['Random Forest Regressor Expanded'] = best_rf_expanded

# Gradient Boosting Regressor - Expanded
best_gbr_expanded = build_and_evaluate_model(gbr_model, gbr_params_expanded, '
best_estimators['Gradient Boosting Regressor Expanded'] = best_gbr_expanded

# XGBoost Regressor
best_xgb = build_and_evaluate_model(xgb_model, xgb_params, 'XGBoost Regressor'
best_estimators['XGBoost Regressor'] = best_xgb
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits

/var/folders/lq/_1mfm1_x15s4jd56sgz3z4d00000gn/T/ipykernel_12972/236482813
7.py:58: PerformanceWarning: DataFrame is highly fragmented.  This is usual
ly the result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
/var/folders/lq/_1mfm1_x15s4jd56sgz3z4d00000gn/T/ipykernel_12972/236482813
7.py:58: PerformanceWarning: DataFrame is highly fragmented.  This is usual
ly the result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
/var/folders/lq/_1mfm1_x15s4jd56sgz3z4d00000gn/T/ipykernel_12972/236482813
7.py:58: PerformanceWarning: DataFrame is highly fragmented.  This is usual
ly the result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) instead.
To get a de-fragmented frame, use `newframe = frame.copy()`
/var/folders/lq/_1mfm1_x15s4jd56sgz3z4d00000gn/T/ipykernel_12972/236482813
7.py:58: PerformanceWarning: DataFrame is highly fragmented.  This is usual
ly the result of calling `frame.insert` many times, which has poor performa
```

1. Gradient Boosting Regressor Expanded:
   - Achieves the lowest RMSE (39.9625) and the highest R^2 (0.5790) among the models tested.
   - Demonstrates strong performance with optimal tuning of hyperparameters (e.g., learning rate, max depth, and subsample).
2. XGBoost Regressor:
   - Performs competitively with an RMSE of 40.1440 and R^2 of 0.5752.
   - XGBoost may slightly underperform compared to Gradient Boosting due to subtle differences in feature handling or the need for further tuning.
3. Random Forest Regressor Expanded:
   - Slightly higher RMSE (40.9076) and lower R^2 (0.5589) compared to the gradient-based methods.

9.4 Building a stacked model

Now we will build a stacked model to see if we can improve the performance of our models.

In [ ]:
```
def build_modeling_pipeline(preprocessor):
    """
    Builds a modeling pipeline with preprocessing, feature selection, and a sta

    Parameters:
    - preprocessor: scikit-learn Pipeline object containing preprocessing steps

    Returns:
    - pipeline: scikit-learn Pipeline object ready for fitting.
    """

    # Define base models for stacking
    base_models = [
        ('xgb', XGBRegressor(
            colsample_bytree=1.0, learning_rate=0.1, max_depth=3,
            n_estimators=100, subsample=0.8, random_state=42
        )),
        ('gbr', GradientBoostingRegressor(
            learning_rate=0.1, max_depth=3, n_estimators=100,
            subsample=0.8, random_state=42
        )),
        ('catboost', CatBoostRegressor(
            iterations=100, learning_rate=0.1, depth=3,
            verbose=0, random_state=42
        )),
        ('lightgbm', lgb.LGBMRegressor(
            num_leaves=31, learning_rate=0.05, n_estimators=100, random_state=4
        )),
        ('rf', RandomForestRegressor(
            max_depth=10, min_samples_leaf=2, min_samples_split=10,
            n_estimators=500, random_state=42
        ))
    ]

    # Stacking regressor with Linear Regression as final estimator
    stacked_model = StackingRegressor(
        estimators=base_models,
        final_estimator=LinearRegression(),
        n_jobs=-1
    )

    # Construct the complete pipeline
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('feature_selection', SelectKBest(score_func=f_regression, k=50)),
        ('model', stacked_model)
    ])

    return pipeline
```

In [ ]:
```
preprocessor = get_preprocessing_pipeline(data, ordinal_mappings)
```

In [ ]:
```
# Build the modeling pipeline
pipeline = build_modeling_pipeline(preprocessor)
```

In [ ]:
```
# Fit the pipeline on the training data
pipeline.fit(X_train, y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.001944 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 316
[LightGBM] [Info] Number of data points in the train set: 3474, number of use
d features: 50
[LightGBM] [Info] Start training from score 157.084917
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.007849 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.008129 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 313
[LightGBM] [Info] Total Bins 314
[LightGBM] [Info] Number of data points in the train set: 2779, number of use
d features: 50
[LightGBM] [Info] Number of data points in the train set: 2779, number of use
d features: 50
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.005792 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 308
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.006235 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Number of data points in the train set: 2779, number of use
d features: 50
[LightGBM] [Info] Total Bins 311
[LightGBM] [Info] Start training from score 157.563512
[LightGBM] [Info] Start training from score 158.118748
[LightGBM] [Info] Number of data points in the train set: 2779, number of use
d features: 50
[LightGBM] [Info] Start training from score 157.426412
[LightGBM] [Info] Start training from score 156.403023
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.006108 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 311
[LightGBM] [Info] Number of data points in the train set: 2780, number of use
d features: 50
[LightGBM] [Info] Start training from score 155.913309
```

Out[97]:
```
Pipeline(steps=[('preprocessor',
                 Pipeline(steps=[('ordinal_mapper',
                                  FunctionTransformer(func=<function map_ordi
nal_variables at 0x302e87700>,
                                                      kw_args={'ordinal_col
s': ['age_03',

'age_12',

'edu_gru_03',

'edu_gru_12',

'n_living_child_03',

'n_living_child_12',

'glob_hlth_03',

'glob_hlth_12',

'bmi_12',

'decis_famil_12',

'decis_personal_12',

'satis_ideal_12',

'satis_excel...
                                                       GradientBoostingRegressor(ran
dom_state=42,
                                                                                 sub
sample=0.8)),
                                                      ('catboost',
                                                       <catboost.core.CatBoostRegres
sor object at 0x3283be550>),
                                                      ('lightgbm',
                                                       LGBMRegressor(learning_rate=
0.05,
                                                                     random_state=4
2)),
                                                      ('rf',
                                                       RandomForestRegressor(max_dep
th=10,
                                                                             min_sam
ples_leaf=2,
                                                                             min_sam
```

```
ples_split=10,
                                                                    n_estim
ators=500,
                                                                    random_
state=42))],
                                        final_estimator=LinearRegression(),
                                        n_jobs=-1))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
# Make predictions on the validation set
y_pred_val = pipeline.predict(X_val)

# Evaluate the model
from sklearn.metrics import mean_squared_error
rmse_val = mean_squared_error(y_val, y_pred_val, squared=False)
r2_val = r2_score(y_val, y_pred_val)
print(f"Validation RMSE: {rmse_val:.2f}")
print(f"Validation R²: {r2_val:.2f}")
```

```
Validation RMSE: 40.10
```

In [ ]:
```
# Make predictions on the test set
y_pred_test = pipeline.predict(X_test)

# Evaluate the model
rmse_test = mean_squared_error(y_test, y_pred_test, squared=False)
r2_test = r2_score(y_test, y_pred_test)
print(f"Test RMSE: {rmse_test:.2f}")
print(f"Test R²: {r2_test:.2f}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[99], line 5
      2 y_pred_test = pipeline.predict(X_test)
      4 # Evaluate the model
----> 5 rmse_test = mean_squared_error(y_test, y_pred_test, squared=False)
      6 print(f"Test RMSE: {rmse_test:.2f}")

File /opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/util
s/_param_validation.py:213, in validate_params.<locals>.decorator.<locals>.
wrapper(*args, **kwargs)
    207 try:
    208     with config_context(
    209         skip_parameter_validation=(
    210             prefer_skip_nested_validation or global_skip_validation
    211         )
    212     ):
--> 213         return func(*args, **kwargs)
    214 except InvalidParameterError as e:
```

9.5 Extracting Feature Importance

Now we write a function to extract the important features from the individual models.

In [ ]:
```
def get_feature_importances(model, model_name):
    """
    Extracts feature importances from the model.

    Parameters:
    - model: The trained model pipeline.
    - model_name: Name of the model.

    Returns:
    - feature_importances_df: DataFrame containing feature names and their impo
    """
    # Get the names of the selected features
    selected_features_mask = model.named_steps['feature_selection'].get_suppor
    feature_names = model.named_steps['preprocessor'].named_steps['preprocessi
    selected_feature_names = feature_names[selected_features_mask]

    # Get feature importances from the model
    importances = model.named_steps['model'].feature_importances_

    # Create a DataFrame
    feature_importances_df = pd.DataFrame({
        'Feature': selected_feature_names,
        'Importance': importances
    }).sort_values(by='Importance', ascending=False)

    print(f"\nTop Features for {model_name}:")
    print(feature_importances_df.head(20))

    return feature_importances_df

# For Random Forest
rf_importances = get_feature_importances(best_rf_expanded, 'Random Forest Regr

# For Gradient Boosting
gbr_importances = get_feature_importances(best_gbr_expanded, 'Gradient Boostin

# For XGBoost
xgb_importances = get_feature_importances(best_xgb, 'XGBoost Regressor')
```

```
Top Features for Random Forest Regressor Expanded:
                                          Feature  Importance
33                                num__edu_gru_12    0.521367
26                      num__social_engagement_12    0.058169
31                                   num__age_12    0.056331
30                                   num__age_03    0.034757
38                                   num__bmi_12    0.022598
10                                 num__n_depr_12    0.018513
37                               num__glob_hlth_12    0.015656
35                          num__n_living_child_12    0.014372
32                                 num__edu_gru_03    0.014031
41                                 num__rameduc_m    0.013471
49  nom__j11_12_Wood, mosaic, or other covering 1    0.012576
12                      num__positive_mood_change    0.012133
43                               num__rrfcntx_m_12    0.011846
40                                  num__memory_12    0.011395
44                               num__rsocact_m_12    0.011158
21                                   num__reads_12    0.010915
34                          num__n_living_child_03    0.010631
36                               num__glob_hlth_03    0.009711
24                            num__rinc_pension_12    0.008753
18                     num__decis_personal_change    0.008418

Top Features for Gradient Boosting Regressor Expanded:
                                          Feature  Importance
33                                num__edu_gru_12    0.561775
31                                   num__age_12    0.092249
26                      num__social_engagement_12    0.066243
30                                   num__age_03    0.027742
38                                   num__bmi_12    0.024279
49  nom__j11_12_Wood, mosaic, or other covering 1    0.018138
21                                   num__reads_12    0.017967
32                                 num__edu_gru_03    0.015168
34                          num__n_living_child_03    0.012279
16                                   num__games_12    0.012156
41                                 num__rameduc_m    0.012071
14                                   num__n_iadl_12    0.010887
35                          num__n_living_child_12    0.009685
24                            num__rinc_pension_12    0.009538
37                               num__glob_hlth_12    0.008932
18                     num__decis_personal_change    0.007829
48                       nom__rjlocc_m_12_Missing    0.007793
36                               num__glob_hlth_03    0.007252
40                                  num__memory_12    0.006418
0                num__insurance_coverage_depth_12    0.006371

Top Features for XGBoost Regressor:
                                          Feature  Importance
33                                num__edu_gru_12    0.289786
21                                   num__reads_12    0.048075
31                                   num__age_12    0.044852
30                                   num__age_03    0.040640
49  nom__j11_12_Wood, mosaic, or other covering 1    0.034455
26                      num__social_engagement_12    0.032791
16                                   num__games_12    0.031160
32                                 num__edu_gru_03    0.023107
```

22	num__table_games_12	0.020620
38	num__bmi_12	0.020598
34	num__n_living_child_03	0.019776
41	num__rameduc_m	0.019637
44	num__rsocact_m_12	0.018943
35	num__n_living_child_12	0.018224
39	num__decis_personal_12	0.017351
9	num__total_iadl_limitations_12	0.017310
0	num__insurance_coverage_depth_12	0.015614
4	num__tired_03	0.015033
48	nom__rjlocc_m_12_Missing	0.014572
13	num__hard_12	0.013599

9.6 Feature Selection

Refining feature selection can significantly improve the RMSE by removing noise and focusing the model on the most predictive features.

We will use SHAP to extract the important features from the models and then use RFE to further select best features that we use to retrain the model.

This is in a bid to simplify our model and improve explainability.

First we create a custom transformer that uses SHAP to select features

In [ ]:
```
# SHAP-based feature selection as a custom transformer
class SHAPFeatureSelector(BaseEstimator, TransformerMixin):
    def __init__(self, base_model, num_features):
        self.base_model = base_model
        self.num_features = num_features
        self.selected_features = None

    def fit(self, X, y):
        # Train the base model for SHAP analysis
        self.base_model.fit(X, y)
        explainer = shap.TreeExplainer(self.base_model)
        shap_values = explainer.shap_values(X)
        # Calculate mean absolute SHAP values for feature importance
        feature_importances = np.abs(shap_values).mean(axis=0)
        # Select top features
        self.selected_features = np.argsort(feature_importances)[-self.num_fea
        return self

    def transform(self, X):
    #    Return only the selected features
        return X[:, self.selected_features]
```

Then we edit our modeling pipeline to include the SHAP and RFE feature selection.

In [ ]:
```
def build_modeling_pipeline_with_shap_rfe(preprocessor):
    """
    Builds a modeling pipeline with preprocessing, SHAP-based feature selectio

    Parameters:
    - preprocessor: scikit-learn Pipeline object containing preprocessing steps

    Returns:
    - pipeline: scikit-learn Pipeline object ready for fitting.
    """

    # Define base models for stacking
    base_models = [
        ('xgb', XGBRegressor(
            colsample_bytree=1.0, learning_rate=0.1, max_depth=3,
            n_estimators=100, subsample=0.8, random_state=42
        )),
        ('gbr', GradientBoostingRegressor(
            learning_rate=0.1, max_depth=3, n_estimators=100,
            subsample=0.8, random_state=42
        )),
        ('catboost', CatBoostRegressor(
            iterations=100, learning_rate=0.1, depth=3,
            verbose=0, random_state=42
        )),
        ('lightgbm', lgb.LGBMRegressor(
            num_leaves=31, learning_rate=0.05, n_estimators=100, random_state=
        )),
        ('rf', RandomForestRegressor(
            max_depth=10, min_samples_leaf=2, min_samples_split=10,
            n_estimators=500, random_state=42
        ))
    ]

    # Stacking regressor with Linear Regression as final estimator
    stacked_model = StackingRegressor(
        estimators=base_models,
        final_estimator=LinearRegression(),
        n_jobs=-1
    )

    # RFE for further feature selection
    rfe_selector = RFE(estimator=RandomForestRegressor(n_estimators=100, rando

    # Construct the complete pipeline
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('shap_selector', SHAPFeatureSelector(base_model=RandomForestRegressor
            max_depth=10, min_samples_leaf=2, min_samples_split=10,
            n_estimators=500, random_state=42), num_features=50)),
        ('rfe', rfe_selector),
        ('model', stacked_model)
    ])
```

```
    return pipeline
```

9.7 Putting it all together, Building the final model

Building the Modeling Pipeline

In [ ]:
```
# Build the modeling pipeline
pipeline = build_modeling_pipeline_with_shap_rfe(preprocessor)
```

Fitting the Pipeline

In [ ]:
```
# Fit the pipeline on the training data
pipeline.fit(X_train, y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.000792 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 753
[LightGBM] [Info] Number of data points in the train set: 3474, number of use
d features: 30
[LightGBM] [Info] Start training from score 157.084917
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of tes
ting was 0.002644 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of tes
ting was 0.002495 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 714
[LightGBM] [Info] Total Bins 721
[LightGBM] [Info] Number of data points in the train set: 2780, number of use
d features: 30
[LightGBM] [Info] Number of data points in the train set: 2779, number of use
d features: 30
[LightGBM] [Info] [LightGBM] [Info] Auto-choosing row-wise multi-threading, t
he overhead of testing was 0.000966 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
Auto-choosing row-wise multi-threading, the overhead of testing was 0.001254
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 719
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.000688 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 720
[LightGBM] [Info] Total Bins 715
[LightGBM] [Info] Number of data points in the train set: 2779, number of use
d features: 30
[LightGBM] [Info] Number of data points in the train set: 2779, number of use
d features: 30
[LightGBM] [Info] Number of data points in the train set: 2779, number of use
d features: 30
[LightGBM] [Info] Start training from score 156.403023
[LightGBM] [Info] Start training from score 155.913309
[LightGBM] [Info] Start training from score 157.563512
[LightGBM] [Info] Start training from score 157.426412
[LightGBM] [Info] Start training from score 158.118748
```

Out[110]:
```
Pipeline(steps=[('preprocessor',
                 Pipeline(steps=[('ordinal_mapper',
                                  FunctionTransformer(func=<function map_ordi
nal_variables at 0x302e87700>,
                                                      kw_args={'ordinal_col
s': ['age_03',

'age_12',

'edu_gru_03',

'edu_gru_12',

'n_living_child_03',

'n_living_child_12',

'glob_hlth_03',

'glob_hlth_12',

'bmi_12',

'decis_famil_12',

'decis_personal_12',

'satis_ideal_12',

'satis_excel...
                                                   GradientBoostingRegressor(ran
dom_state=42,
                                                                             sub
sample=0.8)),
                                                  ('catboost',
                                                   <catboost.core.CatBoostRegres
sor object at 0x302fc9490>),
                                                  ('lightgbm',
                                                   LGBMRegressor(learning_rate=
0.05,
                                                                 random_state=4
2)),
                                                  ('rf',
                                                   RandomForestRegressor(max_dep
th=10,
                                                                         min_sam
ples_leaf=2,
                                                                         min_sam
```

```
ples_split=10,
                                                                    n_estim
ators=500,
                                                                    random_
state=42))],
                                        final_estimator=LinearRegression(),
                                        n_jobs=-1))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating the model using the validation set

In [ ]:
```
# Make predictions on the validation set
y_pred_val = pipeline.predict(X_val)

# Evaluate the model
from sklearn.metrics import mean_squared_error
rmse_val = mean_squared_error(y_val, y_pred_val, squared=False)
r2_val = r2_score(y_val, y_pred_val)
print(f"Validation RMSE: {rmse_val:.2f}")
print(f"Validation R2 : {r2_val:.2f}")
```

```
Validation RMSE: 40.82
Validation R2 : 0.56
```

Testing the model on the test set

In [ ]:
```
# Make predictions on the test set
y_pred_test = pipeline.predict(X_test)

# Evaluate the model
rmse_test = mean_squared_error(y_test, y_pred_test, squared=False)
r2_test = r2_score(y_test, y_pred_test)
print(f"Test RMSE: {rmse_test:.2f}")
print(f"Test R2 : {r2_test:.2f}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[112], line 5
      2 y_pred_test = pipeline.predict(X_test)
      4 # Evaluate the model
----> 5 rmse_test = mean_squared_error(y_test, y_pred_test, squared=False)
      6 r2_test = r2_score(y_test, y_pred_test)
      7 print(f"Test RMSE: {rmse_test:.2f}")

File /opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/utils/_param_validation.py:213, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    207 try:
    208     with config_context(
    209         skip_parameter_validation=(
    210             prefer_skip_nested_validation or global_skip_validation
    211         )
    212     ):
--> 213         return func(*args, **kwargs)
```

10. Model Interpretation

To interpret the model using the selected features, we will focus on:

1. Feature Importance:
   - Use SHAP values to explain the impact of each feature on the model's predictions.
   - Create a summary plot to visualize the relative importance of the features.
2. Global Interpretations:
   - Understand how each feature generally affects the target variable (e.g., higher or lower values increasing or decreasing predictions).

In [ ]:
```
def print_selected_features(pipeline, X_train):
    """
    Extracts and prints the selected features after SHAP and RFE steps in the p

    Parameters:
    - pipeline: The fitted pipeline with SHAP and RFE steps.
    - X_train: The training data used for feature selection.
    """
    # Get the SHAP feature selector step
    shap_selector = pipeline.named_steps['shap_selector']
    rfe_selector = pipeline.named_steps['rfe']

    # Get feature names from the preprocessor
    feature_names = pipeline.named_steps['preprocessor'].named_steps['preproces

    # Get the SHAP-selected features
    shap_selected_features = [feature_names[i] for i in shap_selector.selected_

    # Apply RFE to refine SHAP-selected features
    rfe_mask = rfe_selector.support_
    refined_features = [shap_selected_features[i] for i in range(len(shap_selec

    print("\nSelected Features after SHAP and RFE:")
    for feature in refined_features:
        print(feature)

    return refined_features


# Extract and print the selected features
selected_features = print_selected_features(pipeline, X_train)
```

```
Selected Features after SHAP and RFE:
num__preventive_care_index_12
num__adl_iadl_progression
num__rafeduc_m
num__decis_famil_12
num__satis_excel_12
num__hincome_03
num__n_depr_03
num__hincome_change
num__memory_12
num__glob_hlth_03
num__n_living_child_03
num__n_depr_12
num__hincome_12
num__time_gap
num__insurance_coverage_depth_12
num__aggregate_income_12
num__rrfcntx_m_12
num__aggregate_income_03
num__rsocact_m_12
num__glob_hlth_12
num__n_living_child_12
num__rameduc_m
num__edu_gru_03
nom__j11_12_Wood, mosaic, or other covering 1
num__reads_12
num__bmi_12
num__age_03
num__social_engagement_12
num__age_12
num__edu_gru_12
```

In [ ]:

```
def interpret_stacked_model_with_shap_fixed(pipeline, X_train, y_train, selecte
    """
    Interprets the base models within a StackingRegressor using SHAP.

    Parameters:
    - pipeline: The fitted pipeline with the StackingRegressor.
    - X_train: The training data.
    - y_train: The target variable for training data.
    - selected_features: List of selected features after SHAP and RFE.
    """
    import shap
    from sklearn.pipeline import Pipeline

    # Extract preprocessing step
    preprocessor = pipeline.named_steps['preprocessor']
    X_train_preprocessed = preprocessor.transform(X_train)

    # Select only the features used in the model
    selected_feature_indices = [
        i for i, feature in enumerate(preprocessor.named_steps['preprocessing'
    ]
    X_train_selected = X_train_preprocessed[:, selected_feature_indices]

    # Extract base models
    stacked_model = pipeline.named_steps['model']
    base_models = stacked_model.estimators

    # Interpret each base model individually
    for name, base_model in base_models:
        print(f"\nInterpreting model: {name}")
        try:
            # Fit the base model on the training data
            fitted_model = base_model.fit(X_train_selected, y_train)

            # Create a SHAP explainer
            explainer = shap.Explainer(fitted_model, X_train_selected)
            shap_values = explainer(X_train_selected)

            # Print individual SHAP values for the top features
            print(f"Top SHAP values for {name}:")
            shap_values_df = pd.DataFrame(shap_values.values, columns=selected_
            print(shap_values_df.head())

            # Generate SHAP plots
            shap.summary_plot(shap_values, X_train_selected, feature_names=sele
            for feature in selected_features[:5]:
                shap.dependence_plot(feature, shap_values, X_train_selected, fe
        except Exception as e:
            print(f"SHAP could not interpret {name}: {e}")

    # Interpret final model if compatible
    print("\nInterpreting final model (Linear Regression)...")
    try:
        final_model = stacked_model.final_estimator
        explainer = shap.Explainer(final_model, X_train_selected)
        shap_values = explainer(X_train_selected)
        shap.summary_plot(shap_values, X_train_selected, feature_names=selecte
```

```
    except Exception as e:
        print(f"SHAP could not interpret the final estimator: {e}")

# Call the function
interpret_stacked_model_with_shap_fixed(pipeline, X_train, y_train, selected_f
```

```
Interpreting model: xgb
Top SHAP values for xgb:
   num__preventive_care_index_12  num__adl_iadl_progression  num__rafeduc_m
\
0                       1.236955                  -0.538216       -0.673425
1                       1.297447                  -0.971720       -0.506903
2                       1.023127                  -1.250926       -0.401129
3                       1.023127                  -1.240010       -0.307831
4                       1.051894                  -0.884451       -0.633253

   num__decis_famil_12  num__satis_excel_12  num__hincome_03  num__n_depr_0
3  \
0             0.370956            -0.007487        -0.804830         0.49612
8
1             0.508570            -1.087687        -1.046501         0.61102
1
2             0.497541            -1.467580        -1.066338        -0.48056
7
```

10.1 SHAP Explainability Report

The SHAP analysis highlights the following Social Determinants of Health (SDOH) features as influential in predicting composite cognitive scores:

- **Preventive Care Index** (num__preventive_care_index_12): Higher preventive care scores correlate positively with cognitive outcomes, underscoring the role of proactive healthcare engagement.
- **Progression in ADL/IADL Limitations** (num__adl_iadl_progression): Worsening limitations are associated with lower cognitive scores, highlighting the link between physical functionality and cognitive health.
- **Parental Education** (num__rafeduc_m, num__rameduc_m): Education levels of parents strongly influence cognitive outcomes, reflecting intergenerational effects.
- **Household Income** (num__hincome_03, num__hincome_change, num__aggregate_income_03, num__aggregate_income_12): Baseline and changes in income predict cognitive outcomes, emphasizing the importance of economic stability.
- **Global Health Ratings** (num__glob_hlth_03, num__glob_hlth_12): Self-reported health assessments are robust predictors, suggesting that overall health perception aligns with cognitive well-being.
- **Social Engagement** (num__social_engagement_12, num__reads_12): Activities like reading and social participation are positive predictors, underscoring the importance of social and intellectual stimulation.
- **Education Levels** (num__edu_gru_12, num__edu_gru_03): Higher education attainment remains a critical determinant of cognitive outcomes.

11 Insights from our model

The analysis identifies key actionable insights into factors influencing cognitive health:

1. **Preventive Care:**
    - Regular health check-ups and preventive care are strongly linked to better cognitive outcomes. Public health initiatives should prioritize access to preventive services, particularly for vulnerable populations.
2. **Physical and Functional Health:**
    - Declines in physical functionality, measured through ADL/IADL limitations, are early indicators of cognitive decline. Community-based programs to maintain physical health and mobility should be emphasized.
3. **Economic Stability:**
    - Household income levels and their progression over time significantly impact cognitive health. Policies supporting economic stability and addressing poverty could improve cognitive outcomes on a population level.
4. **Parental and Personal Education:**
    - Educational attainment of both individuals and their parents plays a crucial role. Strengthening education systems and supporting lifelong learning initiatives is

essential.

5. **Social and Intellectual Engagement:**

    - Participation in social and intellectual activities (e.g., reading) enhances cognitive resilience. Programs to foster community engagement and access to libraries and digital resources are recommended.

12 Recommendations

- Invest in universal preventive healthcare access.
- Develop mobility and exercise programs for aging populations.
- Address socioeconomic disparities through targeted financial assistance.
- Promote education and intellectual activities at all life stages.

12.1 A Guide for Patients and Healthcare Providers

1. **Preventive Healthcare:**

    - Regular check-ups and screenings are crucial. Encourage patients to prioritize preventive care visits.

2. **Physical Health Matters:**

    - Addressing mobility and physical health issues early can reduce the risk of cognitive decline. Engage patients in discussions about daily physical activity and exercises.

3. **Family and Social Support:**

    - Social engagement, such as participating in community events or reading, positively influences cognitive health. Encourage patients to stay socially active.

4. **Economic and Educational Factors:**

    - Economic stress and limited access to education can impact cognitive health. Connect patients to financial and educational resources when possible.

5. **Self-Reported Health Assessments:**

    - Use patients' global health ratings as a discussion point to address broader health issues impacting their cognitive function.

6. **Recommendations for Providers:**

    - Screening and Early Intervention: Regularly screen patients for physical and cognitive health risks, particularly among populations with limited preventive care access.
    - Patient Education: Discuss the importance of preventive care, social activities, and continuous learning with patients.
    - Community Resources: Connect patients to community programs promoting mobility, social engagement, and intellectual stimulation.

7. **For Patients:**

    - Engage in Preventive Care: Regular doctor visits help identify and address health concerns early.
    - Stay Active: Physical and social activities are vital for maintaining cognitive health.
    - Leverage Community Resources: Libraries, community centers, and local organizations offer programs to keep your body and mind engaged.

- Discuss Your Health Goals: Share your goals and concerns with your healthcare provider for personalized advice.

Creating a Test Submission

Loading the submission_format data

In [ ]:
```
# Load the submission format file
submission_format = pd.read_csv('Data/submission_format.csv')

# Display the first few rows to verify
print(submission_format.head())
```

```
    uid  year  composite_score
0  abxu  2016                0
1  aeol  2016                0
2  aeol  2021                0
3  afnb  2016                0
4  afnb  2021                0
```

Loading the test_features data

In [134]:
```
# Load the test features
test_features = pd.read_csv('Data/test_features.csv')
```

Prepare the Test Data for Prediction

We need to merge submission_format with test_features to ensure we have all the necessary features for the required uid and year combinations.

In [ ]:
```
# Merge submission_format with test_features to get the test data for predictio
test_data = submission_format[['uid', 'year']].merge(
    test_features, on=['uid'], how='left'
)
# Check for missing values after the merge
missing_values = test_data.isnull().sum()
print("Missing values in test data before pipeline:\n", missing_values)
```

```
Missing values in test data before pipeline:
 uid              0
year              0
age_03          359
urban_03        359
married_03      359
                ...
a21_12         1090
a22_12         1091
a33b_12        1090
a34_12          390
j11_12           29
Length: 185, dtype: int64
```

In [ ]:
```
# Prepare features for prediction
features_to_drop = ['uid']  # Do not drop 'year' since it's needed by the pipe
X_test = test_data.drop(columns=features_to_drop)

# Ensure that X_test has the same features as used in training
expected_features = X_train.columns.tolist()
# Remove 'uid' from expected_features if present
expected_features = [col for col in expected_features if col != 'uid']

# Reindex X_test to match the expected features
X_test = X_test.reindex(columns=expected_features)

# Confirm lengths before prediction
print(f"Length of submission_format: {len(submission_format)}")
print(f"Length of test_data: {len(test_data)}")
print(f"Length of X_test: {len(X_test)}")
```

```
Length of submission_format: 1105
Length of test_data: 1105
Length of X_test: 1105
```

Making Predictions

In [ ]:
```
# Make predictions using the trained pipeline
y_pred = pipeline.predict(X_test)

# Confirm the number of predictions
print(f"Length of y_pred: {len(y_pred)}")
```

```
Length of y_pred: 1105
```

Adding the predictions to the submission_format DataFrame.

In [ ]:
```
# Round the predictions to the nearest integer and convert to int64
y_pred_int = np.round(y_pred).astype(int)

# Assign integer predictions to submission DataFrame
submission = submission_format.copy()
submission['composite_score'] = y_pred_int

# Ensure the submission has the correct columns
submission = submission[['uid', 'year', 'composite_score']]

# Save the submission file
submission.to_csv('submission.csv', index=False)

print("Submission file 'submission.csv' created successfully!")
```

```
Submission file 'submission.csv' created successfully!
```

In [ ]:
```
end_time = time.time()
print(f'Time taken to run the notebook: {(end_time - start_time) / 60} minutes
```

```
Time taken to run the notebook: 38.90929878155391 minutes
```

In [ ]:
```
# Saving the pipeline as a pkl
import joblib

with open('stacked_model.pkl', 'wb') as f:
    joblib.dump(pipeline, f)
```

```
---------------------------------------------------------------------------
PicklingError                             Traceback (most recent call last)
Cell In[125], line 5
      2 import joblib
      4 with open('stacked_model.pkl', 'wb') as f:
----> 5     joblib.dump(pipeline, f)

File /opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/joblib/numpy
_pickle.py:555, in dump(value, filename, compress, protocol, cache_size)
    553         NumpyPickler(f, protocol=protocol).dump(value)
    554 else:
--> 555     NumpyPickler(filename, protocol=protocol).dump(value)
    557 # If the target container is a file object, nothing is returned.
    558 if is_fileobj:

File /opt/anaconda3/envs/learn-env/lib/python3.9/pickle.py:487, in _Pickle
r.dump(self, obj)
    485 if self.proto >= 4:
    486     self.framer.start_framing()
```

In [ ]:
```
print(f"Expected features: {expected_features}")
```

```
Expected features: ['year', 'age_03', 'urban_03', 'married_03', 'n_mar_03',
'edu_gru_03', 'n_living_child_03', 'migration_03', 'glob_hlth_03', 'adl_dress
_03', 'adl_walk_03', 'adl_bath_03', 'adl_eat_03', 'adl_bed_03', 'adl_toilet_0
3', 'n_adl_03', 'iadl_money_03', 'iadl_meds_03', 'iadl_shop_03', 'iadl_meals_
03', 'n_iadl_03', 'depressed_03', 'hard_03', 'restless_03', 'happy_03', 'lone
ly_03', 'enjoy_03', 'sad_03', 'tired_03', 'energetic_03', 'n_depr_03', 'cesd_
depressed_03', 'hypertension_03', 'diabetes_03', 'resp_ill_03', 'arthritis_0
3', 'hrt_attack_03', 'stroke_03', 'cancer_03', 'n_illnesses_03', 'exer_3xwk_0
3', 'alcohol_03', 'tobacco_03', 'test_chol_03', 'test_tuber_03', 'test_diab_0
3', 'test_pres_03', 'hosp_03', 'visit_med_03', 'out_proc_03', 'visit_dental_0
3', 'imss_03', 'issste_03', 'pem_def_mar_03', 'insur_private_03', 'insur_othe
r_03', 'insured_03', 'decis_personal_03', 'employment_03', 'age_12', 'urban_1
2', 'married_12', 'n_mar_12', 'edu_gru_12', 'n_living_child_12', 'migration_1
2', 'glob_hlth_12', 'adl_dress_12', 'adl_walk_12', 'adl_bath_12', 'adl_eat_1
2', 'adl_bed_12', 'adl_toilet_12', 'n_adl_12', 'iadl_money_12', 'iadl_meds_1
2', 'iadl_shop_12', 'iadl_meals_12', 'n_iadl_12', 'depressed_12', 'hard_12',
'restless_12', 'happy_12', 'lonely_12', 'enjoy_12', 'sad_12', 'tired_12', 'en
ergetic_12', 'n_depr_12', 'cesd_depressed_12', 'hypertension_12', 'diabetes_1
2', 'resp_ill_12', 'arthritis_12', 'hrt_attack_12', 'stroke_12', 'cancer_12',
'n_illnesses_12', 'bmi_12', 'exer_3xwk_12', 'alcohol_12', 'tobacco_12', 'test
_chol_12', 'test_tuber_12', 'test_diab_12', 'test_pres_12', 'hosp_12', 'visit
_med_12', 'out_proc_12', 'visit_dental_12', 'imss_12', 'issste_12', 'pem_def_
mar_12', 'insur_private_12', 'insur_other_12', 'insured_12', 'decis_famil_1
2', 'decis_personal_12', 'employment_12', 'vax_flu_12', 'vax_pneu_12', 'seg_p
op_12', 'care_adult_12', 'care_child_12', 'volunteer_12', 'attends_class_12',
'attends_club_12', 'reads_12', 'games_12', 'table_games_12', 'comms_tel_comp_
12', 'act_mant_12', 'tv_12', 'sewing_12', 'satis_ideal_12', 'satis_excel_12',
'satis_fine_12', 'cosas_imp_12', 'wouldnt_change_12', 'memory_12', 'ragende
r', 'rameduc_m', 'rafeduc_m', 'sgender_03', 'rearnings_03', 'searnings_03',
'hincome_03', 'hinc_business_03', 'hinc_rent_03', 'hinc_assets_03', 'hinc_cap
_03', 'rinc_pension_03', 'sinc_pension_03', 'rrelgimp_03', 'sgender_12', 'rjl
occ_m_12', 'rearnings_12', 'searnings_12', 'hincome_12', 'hinc_business_12',
'hinc_rent_12', 'hinc_assets_12', 'hinc_cap_12', 'rinc_pension_12', 'sinc_pen
sion_12', 'rrelgimp_12', 'rrfcntx_m_12', 'rsocact_m_12', 'rrelgwk_12', 'a34_1
2', 'j11_12']
```

In [ ]: