Final Project

Group Members:

Name / Cnetid / Github_username / section

1.Pei-Chin Lu / Peichin / Pei0504 / section 4

2. Yuan Qi / yuanqi / freyaqi / section 4
3. Huiting(Aurora) Zhang / zhanght / aurorazhang688 / section 4

Data File

Due to file size limitations, the necessary data file for this project is hosted on Google Drive. You can download it using the following link:

[Download Original Data File](https://drive.google.com/file/d/1TGlCQULvQDOUOzWxdpDD60O2lJmcQer0/

Make sure to download this file and place it in the appropriate directory before running the project.

Research Question and the approach we took

Research in various fields has shown that **residential mobility** influences key aspects of how individuals think about themselves, interact with others, and perceive public rules. Based on this, our project primarily investigates the relationship between social mobility (e.g., economic opportunities, migration patterns, and educational access) and a range of personal characteristics. Specifically, we will pay attention to residents' **Happiness (HAPPY), Trust (TRUST), and Fairness (FAIR)**. These variables were selected because they represent essential aspects of individual well-being, interpersonal dynamics, and societal norms. Establishing reliable connections between these factors is crucial for designing effective public policies that enhance social creativity and public well-being.

To achieve this, we conduct a series of regression analyses. Additionally, to ensure that our variable selection is free from selection bias or **"cherry-picking"**(hand-picking variables to show favorable insights), we have implemented additional measures. We developed a method to objectively verify the validity of our variable selection, including the use of Exploratory Graph Analysis (EGA) and statistical tests, to classify variables as "Interested," "Proximate," or "Distal." These measures and visualizations help enhance the credibility of our research and ensure that our conclusions are derived from objective data analysis. All these charts and visualizations will ultimately be presented in our Shiny app.

Literature Base

A growing body of literature highlights the significant impact of residential mobility on both individual and cultural dynamics. At the individual level, residential mobility has been linked to increased individualism, a heightened sense of freedom, optimism, and well-being. At the group level, it fosters broader but shallower social networks, higher relational mobility, and greater trust in strangers.

A study **Shifts in Residential Mobility Predict Shifts in Culture** by Buttrick, Cha, and Oishi used quantitative methods to examine the positive impact of residential mobility on cultural variables such as trust, fairness, and happiness.

Research Objective

Building on these studies, we aim to revisit this topic using the methodology of Buttrick, Cha, and Oishi to verify the impact of residential mobility on key cultural dimensions. The results will offer valuable insights for shaping mobility-related policies and understanding their broader cultural implications.

Setup

```
# Setup
import dask.dataframe as dd
import pandas as pd
import numpy as np
import pyreadstat
from sklearn.preprocessing import StandardScaler
from scipy.interpolate import interp1d
from dash import Dash, dcc, html, Input, Output, State
import plotly.graph_objects as go
import plotly.express as px
```

```python
from dash.dash_table import DataTable
from sklearn.impute import KNNImputer

from shiny import App, ui, reactive, render
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import statsmodels.api as sm
import tempfile
import matplotlib.pyplot as plt

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.impute import KNNImputer
from sklearn.cluster import KMeans
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns

import altair as alt
```

Load Dataset

```python
# Load data
gss_data, meta = pyreadstat.read_sav(r"F:\python2\Final-Project\Data\GSS7218_R3.sav")
labels_data = pd.read_excel(r"F:\python2\Final-Project\Data\suggestions_modified.xlsx")
```

We utilize publicly available datasets such as the General Social Survey (GSS) from 1978 to 2018, which provides comprehensive data on social trends, public happiness, and socio-economic factors in the United States. In addition, we will incorporate data on U.S. Immigration and GDP, sourced from government and financial databases like the Federal Reserve Economic Data (FRED). These datasets will be merged and preprocessed within our Shiny app to ensure consistency and compatibility.

Immigration Data (DHS)

The yearly immidration population in the United States, as one covariate of our regression model. Data source: https://www.dhs.gov/immigration-statistics

Gross Domestic Product (GDP) Data (FRED)

Yearly gross domestic product data in the United States as one covariate of our regression model. Data source: https://fred.stlouisfed.org/

General Social Survey (GSS)

All Likert scale variables related to culture comes from this dataset, including measurements about Social Issues, Political Views, Health and Well-being, Religious Beliefs in the United States, as dependent variables of our regression model. Data source: https://gss.norc.org/

Residential Mobility Data (ACS)

Yearly mobility level in the United States, as independent variables of our regression model Data source: https://www.census.gov/programs-surveys/acs

In labels_data, we categorized each personal characteristic variable into one of the following types for user selection in the Shiny app interface: Likert Scale Variables, Binary Variables, Continuous Variables, Multichoice Variables, Administration Variables.

Clean Data

```
# Define columns to recode
column_recode_3to2 = ["COURTS", "RELITEN", "HELPFUL", "FAIR", "TRUST", "AGED", "FINALTER", "
column_recode_4othertomissing = ["GETAHEAD"]
column_recode_5othertomissing = ["PREMARSX", "XMARSEX", "HOMOSEX"]

# Recode columns
gss_data[column_recode_4othertomissing] = gss_data[column_recode_4othertomissing].replace(4,
gss_data[column_recode_5othertomissing] = gss_data[column_recode_5othertomissing].replace(5,

# Recode values in column_recode_3to2
gss_data[column_recode_3to2] = gss_data[column_recode_3to2].replace(3, 9992)
gss_data[column_recode_3to2] = gss_data[column_recode_3to2].replace(2, 9993)
gss_data[column_recode_3to2] = gss_data[column_recode_3to2].replace(9992, 2)
gss_data[column_recode_3to2] = gss_data[column_recode_3to2].replace(9993, 3)
```

```
# Calculate the mean of each column by year
data_by_year = gss_data.groupby("YEAR").mean(numeric_only=True).reset_index()
print(data_by_year.columns)
```

```
Index(['YEAR', 'ID', 'WRKSTAT', 'HRS1', 'HRS2', 'EVWORK', 'OCC', 'PRESTIGE',
       'WRKSLF', 'WRKGOVT',
       ...
       'NEISAFE', 'RLOOKS', 'RGROOMED', 'RWEIGHT', 'RHLTHEND', 'WTSS',
       'WTSSNR', 'WTSSALL', 'VSTRAT', 'VPSU'],
      dtype='object', length=6110)
```

We cleaned the GSS dataset by recoding specific variables to ensure consistency and handle missing values. Columns with specific values (e.g., 4 or 5) were recoded to NaN, while others had their values swapped to align with our analysis needs. After preprocessing, we calculated the mean of each column grouped by year and saved the results to a CSV file for further analysis.

```
# Calculate missing values per column in 'data_by_year'
nan_count_per_column = data_by_year.isna().sum()

# Mark the missing data of each column in labels_data
labels_data['missing_count'] = labels_data['variable'].map(nan_count_per_column)
```

```
# Print the result
print(labels_data.head())
```

```
   Likert Scale Variables  Binary Variables  Continuous Variables  \
0                       0                 0                     0
1                       0                 0                     0
2                       0                 0                     0
3                       0                 0                     1
4                       0                 0                     1

   Multichoice variables  Administration variable variable  \
0                      0                        1     YEAR
1                      0                        1       ID
2                      1                        0  WRKSTAT
3                      0                        0     HRS1
4                      0                        0     HRS2

                                 label  \
0         GSS year for this respondent
1                 Respondent ID number
2                   Labor force status
3     Number of hours worked last week
4  Number of hours usually work a week
```

```
                                              labels Note Source  \
0                                                NaN  NaN    NaN
1                                                NaN  NaN    NaN
2  IAP = 0; WORKING FULLTIME = 1; WORKING PARTTIM...  NaN    NaN
3           IAP = -1; 89+ hrs = 89; DK = 98; NA = 99  NaN    NaN
4           IAP = -1; 89+ hrs = 89; DK = 98; NA = 99  NaN    NaN

   missing_count
0              0
1              0
2              0
3              1
4              1
```

We calculated the number of missing values for each column in data_by_year and mapped these counts to the corresponding variables in labels_data.

```python
# Get the current column names
current_colnames = gss_data.columns

# Convert column names to lowercase and then capitalize the first letter
new_colnames = [col.lower().capitalize() for col in current_colnames]

# Prepare for EGA
gss_data_plot = gss_data.copy()

# Assign the new column names to the data frame
gss_data_plot.columns = new_colnames

# Set the first column name to "year"
gss_data_plot.columns.values[0] = "year"
```

```python
# Print the new column names to verify
print(gss_data_plot.columns)
print(gss_data_plot.head())
```

```
Index(['year', 'Id', 'Wrkstat', 'Hrs1', 'Hrs2', 'Evwork', 'Occ', 'Prestige',
       'Wrkslf', 'Wrkgovt',
       ...
       'Neisafe', 'Rlooks', 'Rgroomed', 'Rweight', 'Rhlthend', 'Wtss',
       'Wtssnr', 'Wtssall', 'Vstrat', 'Vpsu'],
```

```
      dtype='object', length=6110)
     year   Id  Wrkstat  Hrs1  Hrs2  Evwork    Occ  Prestige  Wrkslf  Wrkgovt  \
0  1972.0  1.0      1.0   NaN   NaN     NaN  205.0      50.0     2.0      NaN
1  1972.0  2.0      5.0   NaN   NaN     1.0  441.0      45.0     2.0      NaN
2  1972.0  3.0      2.0   NaN   NaN     NaN  270.0      44.0     2.0      NaN
3  1972.0  4.0      1.0   NaN   NaN     NaN    1.0      57.0     2.0      NaN
4  1972.0  5.0      7.0   NaN   NaN     1.0  385.0      40.0     2.0      NaN

   ...  Neisafe  Rlooks  Rgroomed  Rweight  Rhlthend  Wtss  Wtssnr  Wtssall  \
0  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.4446
1  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.8893
2  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.8893
3  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.8893
4  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.8893

   Vstrat  Vpsu
0     NaN   NaN
1     NaN   NaN
2     NaN   NaN
3     NaN   NaN
4     NaN   NaN

[5 rows x 6110 columns]
```

```python
# Remove the column at index 2 (equivalent to removing ID column)
gss_data_cleaned = gss_data.drop(gss_data.columns[1], axis=1)

# Group by the 'YEAR' column and calculate the mean for each year, ignoring NaN values
data_mean_by_year = gss_data_cleaned.groupby('YEAR').mean(numeric_only=True).reset_index()
```

```python
# Get the current column names
current_colnames = data_mean_by_year.columns

# Convert column names to lowercase and then capitalize the first letter
new_colnames = [col.lower().capitalize() for col in current_colnames]

# Assign the new column names to the DataFrame
data_mean_by_year.columns = new_colnames
```

```python
# Display the updated DataFrame with new column names
print(data_mean_by_year.head())
```

```python
# Sve the dataframe as .csv
data_mean_by_year.to_csv(r"F:\python2\Final-Project\Data\data_mean_by_year.csv", index=False
```

```
     Year   Wrkstat       Hrs1       Hrs2    Evwork         Occ   Prestige  \
0  1972.0  3.455673        NaN        NaN  1.246201  489.826327  38.485142
1  1973.0  3.573803  39.882503  40.766667  1.226502  495.518055  38.623123
2  1974.0  3.585580  39.828610  38.977273  1.199688  475.305310  39.447076
3  1975.0  3.575168  38.967277  41.400000  1.210692  487.310472  38.483321
4  1976.0  3.625083  39.659973  39.892857  1.220365  483.449040  38.400888

     Wrkslf  Wrkgovt  Commute  ...  Neisafe  Rlooks  Rgroomed  Rweight  \
0  1.897099      NaN      NaN  ...      NaN     NaN       NaN      NaN
1  1.890126      NaN      NaN  ...      NaN     NaN       NaN      NaN
2  1.914328      NaN      NaN  ...      NaN     NaN       NaN      NaN
3  1.893917      NaN      NaN  ...      NaN     NaN       NaN      NaN
4  1.898669      NaN      NaN  ...      NaN     NaN       NaN      NaN

   Rhlthend  Wtss  Wtssnr   Wtssall       Vstrat      Vpsu
0       NaN   1.0     1.0  0.999827          NaN       NaN
1       NaN   1.0     1.0  0.999916          NaN       NaN
2       NaN   1.0     1.0  1.000213          NaN       NaN
3       NaN   1.0     1.0  0.999844  7026.524161  1.504027
4       NaN   1.0     1.0  1.000063  7126.418946  1.493662

[5 rows x 6109 columns]
```

```python
# Load the US Immigration data
us_immigration_data = pd.read_csv(r"F:\python2\Final-Project\Data\USImmigration.csv")

# Load the external datasets
us_gdp_data = pd.read_csv(r"F:\python2\Final-Project\Data\FREDGDP.csv")

# Load mobility data
mobility_data = pd.read_csv(r"F:\python2\Final-Project\Data\GSS level 2e.csv")

mobility_data = mobility_data[["year", "Mobility", "Mobilitystate"]]
```

```python
# Change the name of data_mean_by_year
gss_data1 = data_mean_by_year.copy()

# Rename the first column to "year"
```

```python
gss_data1.rename(columns={gss_data1.columns[0]: "year"}, inplace=True)

# Uniform the data type of year
gss_data1['year'] = gss_data1['year'].astype(int)
mobility_data['year'] = mobility_data['year'].astype(int)
```

```python
# Merge mobility data with the gss data on "year"
gss_data1 = pd.merge(mobility_data, gss_data1, on="year", how="left")
```

```python
# Format the date and filter for October in the GDP data
us_gdp_data['Date'] = pd.to_datetime(us_gdp_data['DATE'], format='%Y-%m-%d')
us_gdp_data['Year'] = us_gdp_data['Date'].dt.year
us_gdp_data['Month'] = us_gdp_data['Date'].dt.month
us_gdp_data = us_gdp_data[us_gdp_data['Month'] == 10]
```

```python
# Verifying by printing the first few rows of each dataset
print(gss_data.head())
print(us_gdp_data.head())
print(us_immigration_data.head())
```

```
     YEAR   ID  WRKSTAT  HRS1  HRS2  EVWORK    OCC  PRESTIGE  WRKSLF  WRKGOVT  \
0  1972.0  1.0      1.0   NaN   NaN     NaN  205.0      50.0     2.0      NaN
1  1972.0  2.0      5.0   NaN   NaN     1.0  441.0      45.0     2.0      NaN
2  1972.0  3.0      2.0   NaN   NaN     NaN  270.0      44.0     2.0      NaN
3  1972.0  4.0      1.0   NaN   NaN     NaN    1.0      57.0     2.0      NaN
4  1972.0  5.0      7.0   NaN   NaN     1.0  385.0      40.0     2.0      NaN

   ...  NEISAFE  RLOOKS  RGROOMED  RWEIGHT  RHLTHEND  WTSS  WTSSNR  WTSSALL  \
0  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.4446
1  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.8893
2  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.8893
3  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.8893
4  ...      NaN     NaN       NaN      NaN       NaN   1.0     1.0   0.8893

   VSTRAT  VPSU
0     NaN   NaN
1     NaN   NaN
2     NaN   NaN
3     NaN   NaN
4     NaN   NaN
```

```
[5 rows x 6110 columns]
          DATE      GDP       Date  Year  Month
3   1947-10-01  259.745 1947-10-01  1947     10
7   1948-10-01  280.366 1948-10-01  1948     10
11  1949-10-01  270.627 1949-10-01  1949     10
15  1950-10-01  319.945 1950-10-01  1950     10
19  1951-10-01  356.178 1951-10-01  1951     10
   Year   Number
0  2019  1031765
1  2018  1096611
2  2017  1127167
3  2016  1183505
4  2015  1051031
```

We standardized column names, cleaned data by removing unnecessary columns, and calculated yearly averages. External datasets like US Immigration, GDP, and mobility data were processed and merged with GSS data using the "year" column. GDP data was filtered for October observations, ensuring all datasets were aligned and ready for analysis.

```
# Define the columns to interpolate
columns_to_interpolate = [
    "Fair", "Trust", "Happy", "Helpful", "Mobility", "Mobilitystate", "Aged",
    "Attend", "Conarmy", "Conbus", "Conclerg", "Coneduc", "Confed", "Confinan",
    "Conjudge", "Conlabor", "Conlegis", "Conmedic", "Conpress", "Consci", "Contv",
    "Courts", "Divlaw", "Finalter", "Finrela", "Getahead", "Hapmar", "Health",
    "Homosex", "Life", "Nataid", "Natarms", "Natcity", "Natcrime", "Natdrug",
    "Nateduc", "Natenvir", "Natfare", "Natheal", "Natrace", "News", "Pornlaw",
    "Premarsx", "Reliten", "Satjob", "Incom16", "Income", "Rincome", "Partyid",
    "Polviews", "Natspac", "Fund", "Fund16", "Spfund", "Class", "Satfin", "Coop",
    "Comprend", "Xmarsex"
]
```

```
# Sort data by 'year' or the relevant column before interpolation
gss_data2 = gss_data1.sort_values(by='year')

# Check data types of columns to be interpolated
column_types = gss_data2[columns_to_interpolate].dtypes
print("Data types of columns to interpolate:")
print(column_types)

# Convert columns to numeric (force errors to NaN)
gss_data2[columns_to_interpolate] = gss_data2[columns_to_interpolate].apply(pd.to_numeric, e
```

```
Data types of columns to interpolate:
Fair              float64
Trust             float64
Happy             float64
Helpful           float64
Mobility           object
Mobilitystate      object
Aged              float64
Attend            float64
Conarmy           float64
Conbus            float64
Conclerg          float64
Coneduc           float64
Confed            float64
Confinan          float64
Conjudge          float64
Conlabor          float64
Conlegis          float64
Conmedic          float64
Conpress          float64
Consci            float64
Contv             float64
Courts            float64
Divlaw            float64
Finalter          float64
Finrela           float64
Getahead          float64
Hapmar            float64
Health            float64
Homosex           float64
Life              float64
Nataid            float64
Natarms           float64
Natcity           float64
Natcrime          float64
Natdrug           float64
Nateduc           float64
Natenvir          float64
Natfare           float64
Natheal           float64
Natrace           float64
News              float64
Pornlaw           float64
```

```
Premarsx         float64
Reliten          float64
Satjob           float64
Incom16          float64
Income           float64
Rincome          float64
Partyid          float64
Polviews         float64
Natspac          float64
Fund             float64
Fund16           float64
Spfund           float64
Class            float64
Satfin           float64
Coop             float64
Comprend         float64
Xmarsex          float64
dtype: object
```

```
# Apply interpolation using KNN imputer method
imputer = KNNImputer(n_neighbors=5)
gss_data2[columns_to_interpolate] = imputer.fit_transform(gss_data2[columns_to_interpolate])
```

```
# Scaling the data (using StandardScaler from sklearn)
scaler = StandardScaler()

# Apply scaling to the selected columns
gss_data2[columns_to_interpolate] = scaler.fit_transform(gss_data2[columns_to_interpolate])
```

```
# Merging datasets (left join)
gss_data3 = gss_data2.merge(us_immigration_data, how='left', left_on='year', right_on='Year')
                   .rename(columns={'Number': 'Immigration'}) \
                   .merge(us_gdp_data, how='left', left_on='year', right_on='Year')

# Creating lag columns
columns_to_lag = [
    "Fair", "Trust", "Happy", "Helpful", "Mobility", "Mobilitystate", "Aged", "Attend", "Cona
    "Conbus", "Conclerg", "Coneduc", "Confed", "Confinan", "Conjudge", "Conlabor", "Conlegis'
    "Conmedic", "Conpress", "Consci", "Contv", "Courts", "Divlaw", "Finalter", "Finrela", "Ge
    "Hapmar", "Health", "Homosex", "Life", "Nataid", "Natarms", "Natcity", "Natcrime", "Natd:
    "Nateduc", "Natenvir", "Natfare", "Natheal", "Natrace", "News", "Pornlaw", "Premarsx", "I
    "Satjob", "Incom16", "Income", "Rincome", "Partyid", "Polviews", "Natspac", "Fund", "Func
```

```python
    "Spfund", "Class", "Satfin", "Coop", "Comprend", "Xmarsex"
]

# Creating lag columns for each column in columns_to_lag
for column in columns_to_lag:
    gss_data3[f'{column}Lag'] = gss_data3[column].shift(1)

# Calculating statistics (can be adjusted as needed)
calculateStatistics_data = gss_data3
```

```python
print(calculateStatistics_data.head())
calculateStatistics_data.to_csv(r"F:\python2\Final-Project\Shiny app\calculateStatistics_data
```

```
   year  Mobility  Mobilitystate   Wrkstat       Hrs1       Hrs2    Evwork  \
0  1972  0.921235       0.697366  3.455673        NaN        NaN  1.246201
1  1973  0.795235       0.697366  3.573803  39.882503  40.766667  1.226502
2  1974  0.795235       0.697366  3.585580  39.828610  38.977273  1.199688
3  1975  0.795235       0.768203  3.575168  38.967277  41.400000  1.210692
4  1976  0.897610       0.945297  3.625083  39.659973  39.892857  1.220365

          Occ   Prestige    Wrkslf  ...  PolviewsLag  NatspacLag   FundLag  \
0  489.826327  38.485142  1.897099  ...          NaN         NaN       NaN
1  495.518055  38.623123  1.890126  ...    -0.724265    1.123140 -0.887056
2  475.305310  39.447076  1.914328  ...    -1.439654    1.976468 -1.089271
3  487.310472  38.483321  1.893917  ...    -2.160625    2.123863 -0.822573
4  483.449040  38.400888  1.898669  ...    -2.494316    1.936353 -0.779984

   Fund16Lag  SpfundLag  ClassLag  SatfinLag   CoopLag  ComprendLag  \
0        NaN        NaN       NaN        NaN       NaN          NaN
1  -0.785336  -0.941270 -0.788360  -1.881082 -0.881419     1.359171
2  -1.096949  -0.931674  0.836226  -1.188343 -0.997771     1.017964
3  -1.114826  -0.521741  0.836656  -1.451932 -1.475067     0.808359
4  -1.558210  -1.028737  0.040794  -0.538404 -1.054593     0.489306

   XmarsexLag
0         NaN
1    1.676054
2    2.222613
3    1.006868
4    1.600159

[5 rows x 6177 columns]
```

We prepared the data by interpolating missing values using KNN imputation, standardizing the variables for consistency, and merging external datasets like immigration and GDP data to add context. Lagged variables were created to capture temporal relationships, ensuring the dataset was ready for regression and statistical analysis.

One key difference between our analysis and the authors' methodology is how we handled missing data. While the authors used interpolation to address missing values—preserving more of the time-series structure in the data—we used KNN imputation in our analysis. This difference in data handling might explain some of the variations in the results, particularly in terms of the short-term impacts of mobility on happiness and trust.

Shiny app

To further help other researchers who are interested in studying effect of mobility on culture factors in GSS dataset, we build up a shiny app.

By using this Shiny App, users can freely choose any variables they are interested in, run the analysis, and visualize the results, ensuring the research process remains unbiased and transparent. This approach will allow future studies to validate their findings and ensure their analysis is not influenced by subjective variable selection.

```
# Load data paths
cleaned_data_path = r"C:\Users\Aurora\Desktop\final project\calculateStatistics_data_test.csv
labels_data_path = r"C:\Users\Aurora\Desktop\final project\labels_data.csv"

# Load cleaned data
def load_cleaned_data():
    data = pd.read_csv(cleaned_data_path)
    data.columns = data.columns.str.upper()  # Ensure all columns are uppercase to match labe
    data = data.apply(pd.to_numeric, errors="coerce")  # Ensure all columns are numeric
    return data

# Load labels data
def load_labels():
    labels = pd.read_csv(labels_data_path)
    return labels

# Load data
cleaned_data = load_cleaned_data()
labels_data = load_labels()
```

UI Design

```
# Define UI
app_ui = ui.page_fluid(
    ui.tags.style("""
        .output-separator {
            margin-top: 50px;
            margin-bottom: 300px;
        }
    """),
    ui.h2("Dynamic Variable Selection, PCA Clustering, and Regression Analysis"),
    ui.input_checkbox_group(
        "variable_types",
        "Select Variable Types:",
        {
            "Likert Scale Variables": "Likert Scale Variables",
            "Binary Variables": "Binary Variables",
            "Continuous Variables": "Continuous Variables",
            "Multichoice Variables": "Multichoice Variables",
            "Administration Variable": "Administration Variable",
        },
        selected=["Likert Scale Variables"],
    ),
    ui.input_slider(
        "missing_threshold",
        "Select Missing Value Threshold:",
        min=0,
        max=int(labels_data["missing_count"].max()),
        value=5,
        step=1,
    ),
    ui.output_text_verbatim("selected_variables"),
    ui.input_action_button("generate_plot", "Generate PCA Plot"),
    ui.div(ui.output_image("pca_plot"), class_="output-separator"),
    ui.div(ui.output_text_verbatim("cluster_summary"), class_="output-separator"),
    ui.input_action_button("generate_regression", "Generate Regression Results"),
    ui.div(ui.output_table("regression_results"), class_="output-separator"),
    ui.div(ui.output_image("regression_lines_filtered"), class_="output-separator"),

    # Add buttons for histogram and raincloud plot
    ui.input_action_button("generate_histogram", "Generate Histogram Plot"),
    ui.div(ui.output_image("histogram_plot"), class_="output-separator"),
```

```
    ui.input_action_button("generate_raincloud", "Generate Raincloud Plot"),
    ui.div(ui.output_image("raincloud_plot"), class_="output-separator"),
)
```

In the Shiny app interface, users select variables based on two main criteria: Variable Types and Missing Value Threshold for the period from 1972 to 2018. Users can filter variables by selecting specific types and adjusting the threshold slider. Based on the selected variables, the system generates three visualizations and a regression results table.

Server Logic

Altair–Static Plot

Before doing regression and cherry-picking test, let's draw some pictures using altair to observe the patterns and trends between mobility and Happy, Trust and Fair first.

Focused variables over Time

```
# Melt the DataFrame to long format for Altair
gss_melted = calculateStatistics_data.melt(
    id_vars=["year"],  # Ensure 'year' exists in your DataFrame
    value_vars=["Mobility", "Trust", "Fair", "Happy"],
    var_name="Variable",
    value_name="Value"  # Renamed to 'Value' to avoid duplication with 'Variable'
)

# Define colors for the variables
color_scale = alt.Scale(
    domain=["Mobility", "Trust", "Fair", "Happy"],  # Ensure these match the melted variable
    range=["#e41a1c", "#377eb8", "#4daf4a", "#984ea3"]
)

# Create the line chart
line_chart = alt.Chart(gss_melted).mark_line().encode(
    x=alt.X("year:O", title="Year"),  # Ensure 'year' exists in your DataFrame
    y=alt.Y("Value:Q", title="Variable Value"),  # Updated to use 'Value' for Y-axis
    color=alt.Color("Variable:N", scale=color_scale, title="Legend"),
    strokeDash=alt.StrokeDash("Variable:N")  # Different line styles for each variable
).properties(
    width=800,
```

```
    height=400,
    title="Cultural Factors Over Time"
)

# Add trend lines for each variable
trend_lines = line_chart.transform_regression(
    "year", "Value", groupby=["Variable"]  # Updated to use 'Value' instead of non-existent 
).mark_line(strokeDash=[5, 2], opacity=0.7)

# Combine the line chart and trend lines
final_chart = (line_chart + trend_lines).interactive()

# Display the chart
final_chart
```

```
alt.LayerChart(...)
```

Effects of mobility on Trust, Fair, Happy

```
# Create a new column for Mobility categorized into Low, Medium, High
calculateStatistics_data['Mobility_Category'] = pd.cut(calculateStatistics_data['Mobility'],
                                                     bins=[-float('inf'), calculateStatistic
                                                           calculateStatistics_data['Mobility
                                                     labels=["Low", "Medium", "High"])

# Boxplot: Show effect of Mobility on Trust, Fair, and Happy, grouped by Mobility categories
boxplot = alt.Chart(calculateStatistics_data).transform_fold(
    fold=['Trust', 'Fair', 'Happy'],  # Including Happy as a dependent variable
    as_=['Variable', 'Value']
).mark_boxplot().encode(
    x=alt.X('Variable:N', title='Dependent Variables'),  # X-axis: dependent variables
    y=alt.Y('Value:Q', title='Value'),  # Y-axis: values of dependent variables
    color=alt.Color('Variable:N', title='Legend'),  # Different colors for each variable
    column=alt.Column('Mobility_Category:N', title='Mobility Categories')  # Boxplot separat
).properties(
    width=300,
    height=300,
    title='Boxplot: Effect of Mobility on Trust, Fair, and Happy'
)

boxplot
```

```
alt.Chart(...)
```

```python
gss_melted2 = calculateStatistics_data.melt(
    id_vars=["Mobility"],  # Use Mobility as the x-axis
    value_vars=["Happy", "Trust", "Fair"],  # Variables to plot as separate lines
    var_name="Variable",  # Column name for variable names
    value_name="Value"  # Column name for variable values
)

# Create the combined line chart
chart = alt.Chart(gss_melted2).mark_line(point=True).encode(
    x=alt.X('Mobility:Q', title='Mobility'),  # Mobility on the x-axis
    y=alt.Y('Value:Q', title='Values'),  # Combined values of Happy, Trust, Fair
    color=alt.Color('Variable:N', title='Variable'),  # Color each line by Variable
    tooltip=[
        alt.Tooltip('Variable:N', title='Variable'),  # Show variable name
        alt.Tooltip('Mobility:Q', title='Mobility'),
        alt.Tooltip('Value:Q', title='Value')
    ]
).properties(
    width=800,
    height=400,
    title='Mobility vs Happy, Trust, and Fair'
).interactive()  # Enable interactivity for zoom and pan

chart
```

```
alt.Chart(...)
```

From these plots, we can generally conclude that mobility has a positive relationship with Fair as opposed to a negative relationship with Happy and Trust.

Regression and Dynamic Plots

Now, let's do the regressions and draw some dynamic plots.

```python
# Server logic
def server(input, output, session):
    # Reactive function to filter variables
    @reactive.Calc
    def filtered_variables():
```

```
        selected_types = input.variable_types()
        missing_threshold = input.missing_threshold()

        # Map variable types to columns in the labels_data
        type_column_map = {
            "Likert Scale Variables": "Likert Scale Variables",
            "Binary Variables": "Binary Variables",
            "Continuous Variables": "Continuous Variables",
            "Multichoice Variables": "Multichoice variables",
            "Administration Variable": "Administration variable",
        }

        selected_columns = [
            type_column_map[typ] for typ in selected_types if typ in type_column_map
        ]

        if not selected_columns:
            return []

        # Filter labels_data by selected types and missing value threshold
        filtered_data = labels_data.loc[
            (labels_data[selected_columns].sum(axis=1) > 0) &
            (labels_data["missing_count"] <= missing_threshold)
        ]

        final_variables = [var.upper() for var in filtered_data["variable"].tolist()]
        if len(final_variables) == 0:
            return None
        return final_variables

    @output
    @render.text
    def selected_variables():
        variables = filtered_variables()
        if variables is None:
            return "No variables selected due to filtering."
        return f"Selected Variables: {', '.join(variables)}"

    @reactive.event(input.generate_plot)
    def generate_pca_and_clusters():
        selected_vars = filtered_variables()
        if selected_vars is None or len(selected_vars) < 2:
```

```
        return None

    try:
        data_for_pca = cleaned_data[selected_vars]
        imputer = KNNImputer(n_neighbors=5)
        data_imputed = imputer.fit_transform(data_for_pca)

        scaler = StandardScaler()
        data_scaled = scaler.fit_transform(data_imputed)

        pca = PCA(n_components=2)
        pca_result = pca.fit_transform(data_scaled)

        kmeans = KMeans(n_clusters=4, random_state=42)
        clusters = kmeans.fit_predict(pca_result)

        return pca_result, clusters, selected_vars
    except Exception as e:
        print(f"Error in PCA or Clustering: {e}")
        return None

@output
@render.image
def pca_plot():
    pca_data = generate_pca_and_clusters()
    if not pca_data:
        return None

    pca_result, clusters, variable_names = pca_data
    try:
        with tempfile.NamedTemporaryFile(suffix=".png", delete=False) as tmpfile:
            plt.figure(figsize=(10, 7))
            for cluster in np.unique(clusters):
                cluster_indices = np.where(clusters == cluster)[0]
                plt.scatter(
                    pca_result[cluster_indices, 0],
                    pca_result[cluster_indices, 1],
                    label=f"Cluster {cluster + 1}",
                    alpha=0.7,
                )
```

```
                for idx in cluster_indices:
                    plt.annotate(
                        variable_names[idx],  #
                        (pca_result[idx, 0], pca_result[idx, 1]),  #    PCA
                        fontsize=8,  #
                        alpha=0.7,  #
                    )

            plt.title("PCA Result and Clustering")
            plt.xlabel("Principal Component 1")
            plt.ylabel("Principal Component 2")
            plt.legend()
            plt.savefig(tmpfile.name)
            plt.close()
            return {"src": tmpfile.name, "mime_type": "image/png"}
    except Exception as e:
        print(f"Error during plotting: {e}")
        return None

@output
@render.text
def cluster_summary():
    pca_data = generate_pca_and_clusters()
    if not pca_data:
        return "No clusters to summarize."

    _, clusters, variable_names = pca_data
    summary = []
    for cluster in np.unique(clusters):
        cluster_vars = [variable_names[idx] for idx in np.where(clusters == cluster)[0]]
        summary.append(f"Cluster {cluster + 1}: {', '.join(cluster_vars)}")
    return "\n".join(summary)

@reactive.event(input.generate_regression)
def generate_filtered_regression():
    selected_vars = filtered_variables()
    if selected_vars is None or len(selected_vars) < 2:
        return None, None

    regression_results = []
    regression_lines = []
```

```
    try:
        for var in selected_vars:
            lag_var = f"{var}Lag"
            if lag_var not in cleaned_data.columns:
                cleaned_data[lag_var] = cleaned_data[var].shift(1)

            if "MOBILITYLAG" not in cleaned_data.columns:
                return "Required variable 'MOBILITYLAG' not found in data.", None

            regression_data = cleaned_data[[var, lag_var, "MOBILITYLAG"]].dropna()

            if regression_data.empty:
                continue

            X = regression_data[["MOBILITYLAG", lag_var]]
            y = regression_data[var]
            X = sm.add_constant(X)
            model = sm.OLS(y, X).fit()

            regression_results.append({
                "Variable": var,
                "R-squared": model.rsquared,
                "MOBILITYLAG_coef": model.params["MOBILITYLAG"],
                "Lag_coef": model.params[lag_var],
            })

            x_vals = np.linspace(regression_data["MOBILITYLAG"].min(), regression_data["M
            y_vals = model.params["const"] + model.params["MOBILITYLAG"] * x_vals
            regression_lines.append((x_vals, y_vals, var))

        #       10
        top_results = sorted(regression_results, key=lambda x: abs(x["MOBILITYLAG_coef"])
        top_vars = [result["Variable"] for result in top_results]

        filtered_lines = [(x, y, var) for x, y, var in regression_lines if var in top_va
        return pd.DataFrame(regression_results), filtered_lines

    except Exception as e:
        return f"Regression error: {e}", None

@output
@render.table
```

```
def regression_results():
    results, _ = generate_filtered_regression()
    if isinstance(results, str):
        return pd.DataFrame({"Error": [results]})
    return results  #

@output
@render.image
def regression_lines_filtered():
    _, regression_lines = generate_filtered_regression()
    if not regression_lines:
        return None

    try:
        with tempfile.NamedTemporaryFile(suffix=".png", delete=False) as tmpfile:
            plt.figure(figsize=(10, 7))
            for x_vals, y_vals, var in regression_lines:
                plt.plot(x_vals, y_vals, label=var, linewidth=2)
            plt.title("Top Variables by Coefficient")
            plt.xlabel("MOBILITYLAG")
            plt.ylabel("Fitted Values")
            plt.legend()
            plt.savefig(tmpfile.name)
            plt.close()
            return {"src": tmpfile.name, "mime_type": "image/png"}
    except Exception as e:
        print(f"Error during plotting: {e}")
        return None

@reactive.event(input.generate_histogram)
@reactive.event(input.generate_raincloud)
def generate_selected_variable_clusters():
    selected_vars = filtered_variables()
    if selected_vars is None or len(selected_vars) < 2:
        return None

    try:
        data_for_pca = cleaned_data[selected_vars]
        imputer = KNNImputer(n_neighbors=5)
        data_imputed = imputer.fit_transform(data_for_pca)

        scaler = StandardScaler()
```

```
        data_scaled = scaler.fit_transform(data_imputed)

        pca = PCA(n_components=2)
        pca_result = pca.fit_transform(data_scaled)

        kmeans = KMeans(n_clusters=4, random_state=42)
        clusters = kmeans.fit_predict(pca_result)

        return pd.DataFrame({"variable": selected_vars, "cluster": clusters})
    except Exception as e:
        print(f"Error in PCA or Clustering for selected variables: {e}")
        return None

def generate_all_variable_clusters():
    try:
        data_for_pca = cleaned_data.iloc[:, 1:]  # Assuming the first column is metadata
        imputer = KNNImputer(n_neighbors=5)
        data_imputed = imputer.fit_transform(data_for_pca)

        scaler = StandardScaler()
        data_scaled = scaler.fit_transform(data_imputed)

        pca = PCA(n_components=2)
        pca_result = pca.fit_transform(data_scaled)

        kmeans = KMeans(n_clusters=4, random_state=42)
        clusters = kmeans.fit_predict(pca_result)

        variables = cleaned_data.columns[1:]  # Exclude the first column (metadata)
        return pd.DataFrame({"variable": variables, "cluster": clusters})
    except Exception as e:
        print(f"Error in PCA or Clustering for all variables: {e}")
        return None

def classify_variables():
    cluster_info = generate_all_variable_clusters()
    if cluster_info is None or cluster_info.empty:
        return None

    selected_clusters = generate_selected_variable_clusters()
    if selected_clusters is None or selected_clusters.empty:
        return None
```

```
    interested_vars = selected_clusters["variable"].tolist()
    selected_cluster_ids = selected_clusters["cluster"].unique()

    proximate_vars = cluster_info.loc[
        cluster_info["cluster"].isin(selected_cluster_ids) & ~cluster_info["variable"].is
        "variable"
    ].tolist()

    distal_vars = cluster_info.loc[
        ~cluster_info["cluster"].isin(selected_cluster_ids),
        "variable"
    ].tolist()

    return {
        "interested": interested_vars,
        "proximate": proximate_vars,
        "distal": distal_vars
    }

@output
@render.image
def histogram_plot():
    classified = classify_variables()
    if classified is None:
        return None

    grouped_stats = cleaned_data.melt(id_vars=["year"], var_name="variable", value_name="
    grouped_stats["category"] = grouped_stats["variable"].apply(
        lambda var: (
            "Interested" if var in classified["interested"] else
            "Proximate" if var in classified["proximate"] else
            "Distal"
        )
    )

    try:
        with tempfile.NamedTemporaryFile(suffix=".png", delete=False) as tmpfile:
            plt.figure(figsize=(12, 8))
            sns.barplot(
                data=grouped_stats,
                x="value",
                y="variable",
```

```
                hue="category",
                dodge=False,
                palette={"Interested": "#83CA55", "Proximate": "#F36F61", "Distal": "#345
            )
            plt.title("Mean Values of Variables by Category")
            plt.xlabel("Mean Value")
            plt.ylabel("Variable")
            plt.legend(title="Category")
            plt.savefig(tmpfile.name)
            plt.close()
            return {"src": tmpfile.name, "mime_type": "image/png"}
    except Exception as e:
        print(f"Error during histogram plot: {e}")
        return None

@output
@render.image
def raincloud_plot():
    classified = classify_variables()
    if classified is None:
        return None

    grouped_stats = cleaned_data.melt(id_vars=["year"], var_name="variable", value_name="
    grouped_stats["category"] = grouped_stats["variable"].apply(
        lambda var: (
            "Interested" if var in classified["interested"] else
            "Proximate" if var in classified["proximate"] else
            "Distal"
        )
    )

    try:
        with tempfile.NamedTemporaryFile(suffix=".png", delete=False) as tmpfile:
            plt.figure(figsize=(12, 8))
            sns.violinplot(
                data=grouped_stats,
                x="category",
                y="value",
                scale="width",
                inner="box",
                linewidth=1.2,
                palette={"Interested": "#83CA55", "Proximate": "#F36F61", "Distal": "#345
```

```
            )
            sns.stripplot(
                data=grouped_stats,
                x="category",
                y="value",
                size=4,
                jitter=True,
                alpha=0.7,
                palette={"Interested": "#83CA55", "Proximate": "#F36F61", "Distal": "#345
            )
            plt.title("Raincloud Plot of Values by Category")
            plt.xlabel("Category")
            plt.ylabel("Value")
            plt.savefig(tmpfile.name)
            plt.close()
            return {"src": tmpfile.name, "mime_type": "image/png"}
    except Exception as e:
        print(f"Error during raincloud plot: {e}")
        return None
```

The first visualization graph involves Exploratory Graph Analysis (EGA), where variables are clustered using a structured process. Selected variables are standardized and subjected to Principal Component Analysis (PCA) for dimensionality reduction, mapping the data onto two principal components for visualization. These components are then clustered using the K-Means algorithm, dividing the variables into four distinct groups. The results are displayed in a PCA plot, showing the distribution of variables across clusters. From this plot, we know clearly about which cluster our selected variables belong or don't belong to and what are the other variables in the same cluster as our selected variables so that we are able to do some further analysis.

The regression results table complements the visualizations by presenting coefficients, t-values, p-values, and R-squared values for the relationship between the selected variables and mobility. This dual approach enhances the analytical depth, offering users both statistical and visual insights into the data.

Then, we categorized variables into three distinct types for our analysis:

- Interested/focal Variables: These are user-selected variables that represent specific areas of focus. They are explicitly identified by the user and belong to a particular cluster of interest.
- Proximate Variables: These variables share the same cluster as the Interested Variables but are not directly chosen by the user. They are closely related but remain unselected.

- Distal Variables: These variables belong to clusters other than the one containing the Interested Variables. They represent less direct or more distant relationships to the primary focus.

Based on these three categories, we created the histogram chart and the raincloud chart.

The first chart is a histogram plot that visualizes the absolute t-values for regression coefficients, categorizing variables into the three types: Interested, Proximate-other, and Distal-other. Each bar represents a variable, with the t-value magnitude indicating the strength of the relationship between that variable and the independent variable–mobility. The color coding helps distinguish the variable types based on their clustering.

The second chart is a raincloud plot comparing the distribution of absolute t-values across the same three clusters: Interested, Proximate-other, and Distal-other, along with an additional cluster, All-other. The plot shows the spread, central tendency, and individual data points of t-values for each cluster. The annotated Z and p-values compare the statistical significance of differences between clusters, highlighting potential variations in regression performance among the clusters. This detailed visualization further underscores the robustness of our approach, demonstrating that our variable selection methodology maintains objectivity and credibility.

Cherry-picking test–Dynamic Plots

```python
# Load data paths
cleaned_data_path = r"F:\python2\Final-Project\Shiny app\calculateStatistics_data.csv"
labels_data_path = r"F:\python2\Final-Project\Shiny app\labels_data.csv"

# Load cleaned data
def load_cleaned_data():
    data = pd.read_csv(cleaned_data_path)
    data.columns = data.columns.str.upper()  # Ensure all columns are uppercase
    data = data.apply(pd.to_numeric, errors="coerce")  # Ensure all columns are numeric
    return data

# Load labels data
def load_labels():
    labels = pd.read_csv(labels_data_path)
    return labels

# Data loading
cleaned_data = load_cleaned_data()
labels_data = load_labels()
```

```
# Function to filter variables
def filter_variables(variable_types, missing_threshold):
    type_column_map = {
        "Likert Scale Variables": "Likert Scale Variables",
        "Binary Variables": "Binary Variables",
        "Continuous Variables": "Continuous Variables",
        "Multichoice Variables": "Multichoice variables",
        "Administration Variable": "Administration variable",
    }
    selected_columns = [type_column_map[typ] for typ in variable_types if typ in type_column_
    if not selected_columns:
        return []

    filtered_data = labels_data.loc[
        (labels_data[selected_columns].sum(axis=1) > 0) &
        (labels_data["missing_count"] <= missing_threshold)
    ]
    return [var.upper() for var in filtered_data["variable"].tolist()]
```

```
# Perform PCA and clustering
def perform_pca_and_clustering(selected_vars):
    if not selected_vars or len(selected_vars) < 2:
        print("Insufficient variables selected for PCA.")
        return None, None, None

    data_for_pca = cleaned_data[selected_vars]

    # Transpose data to cluster variables (columns)
    data_for_pca = data_for_pca.T

    # Fill missing values
    imputer = KNNImputer(n_neighbors=5)
    data_imputed = imputer.fit_transform(data_for_pca)

    # Standardize data
    scaler = StandardScaler()
    data_scaled = scaler.fit_transform(data_imputed)

    # Perform PCA
    pca = PCA(n_components=2)
    pca_result = pca.fit_transform(data_scaled)
```

```
    # Perform clustering
    kmeans = KMeans(n_clusters=4, random_state=42)
    clusters = kmeans.fit_predict(data_scaled)

    return pca_result, clusters, data_for_pca.index.tolist()

# Generate PCA plot
def generate_pca_plot(pca_result, clusters, variable_names):
    if pca_result is None or clusters is None:
        return

    plt.figure(figsize=(10, 7))
    for cluster in np.unique(clusters):
        cluster_indices = np.where(clusters == cluster)[0]
        plt.scatter(
            pca_result[cluster_indices, 0],
            pca_result[cluster_indices, 1],
            label=f"Cluster {cluster + 1}",
            alpha=0.7,
        )

        for idx in cluster_indices:
            plt.annotate(
                variable_names[idx],
                (pca_result[idx, 0], pca_result[idx, 1]),
                fontsize=8,
                alpha=0.7,
            )

    plt.title("PCA Result and Clustering")
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

```
# Perform regression
def perform_regression(selected_vars):
    if not selected_vars or len(selected_vars) < 2:
        print("Insufficient variables selected for regression.")
        return None
```

```python
    results = []
    for var in selected_vars:
        lag_var = f"{var}Lag"
        if lag_var not in cleaned_data.columns:
            cleaned_data[lag_var] = cleaned_data[var].shift(1)

        if "MOBILITYLAG" not in cleaned_data.columns:
            print("Required variable 'MOBILITYLAG' not found in data.")
            continue

        regression_data = cleaned_data[[var, lag_var, "MOBILITYLAG"]].dropna()

        if regression_data.empty:
            continue

        X = regression_data[["MOBILITYLAG", lag_var]]
        y = regression_data[var]
        X = sm.add_constant(X)
        model = sm.OLS(y, X).fit()

        results.append({
            "Variable": var,
            "R-squared": model.rsquared,
            "MOBILITYLAG_coef": model.params["MOBILITYLAG"],
            "Lag_coef": model.params[lag_var],
            "p-value (MOBILITYLAG)": model.pvalues["MOBILITYLAG"],
            "t-value": model.tvalues["MOBILITYLAG"]
        })

    return pd.DataFrame(results)

# Generate cluster information
def generate_cluster_info(clusters, variable_names):
    cluster_info = {f"Cluster {i + 1}": [] for i in np.unique(clusters)}
    for var, cluster in zip(variable_names, clusters):
        cluster_info[f"Cluster {cluster + 1}"].append(var)
    return cluster_info
```

```python
# Plot regression fit lines
def plot_regression_fit_lines(cleaned_data, top_vars):
    plt.figure(figsize=(12, 8))
    for var, lag_var, model in top_vars:
```

```
        x_vals = np.linspace(cleaned_data["MOBILITYLAG"].min(), cleaned_data["MOBILITYLAG"].m
        y_vals = model.params["const"] + model.params["MOBILITYLAG"] * x_vals
        plt.plot(x_vals, y_vals, label=f"{var} (Coef: {model.params['MOBILITYLAG']:.2f})", al

    plt.title("Regression Fit Lines for Top 10 Variables", fontsize=14)
    plt.xlabel("MOBILITYLAG", fontsize=12)
    plt.ylabel("Fitted Value", fontsize=12)
    plt.legend(loc="best", fontsize=10)
    plt.tight_layout()
    plt.show()
```

```
# Generate barplot and raincloud plot
def generate_plots(regression_results, clusters, variable_names, interested_vars):
    if regression_results is None:
        return

    # Assign categories
    results = []
    for var in variable_names:
        if var in interested_vars:
            category = "Interested"
        elif clusters[variable_names.index(var)] == clusters[variable_names.index(interested_
            category = "Proximate"
        else:
            category = "Distal"

        t_value = regression_results.loc[regression_results["Variable"] == var, "MOBILITYLAG_
        results.append({"variable": var, "abs_t_value": abs(t_value), "category": category})

    results_df = pd.DataFrame(results)

    # Define soft colors for categories
    color_palette = {
        "Interested": "#a6cee3",  # Soft blue
        "Proximate": "#b2df8a",  # Soft green
        "Distal": "#fb9a99",     # Soft pink
    }

    # Barplot
    plt.figure(figsize=(12, 8))
    sns.barplot(
        data=results_df,
```

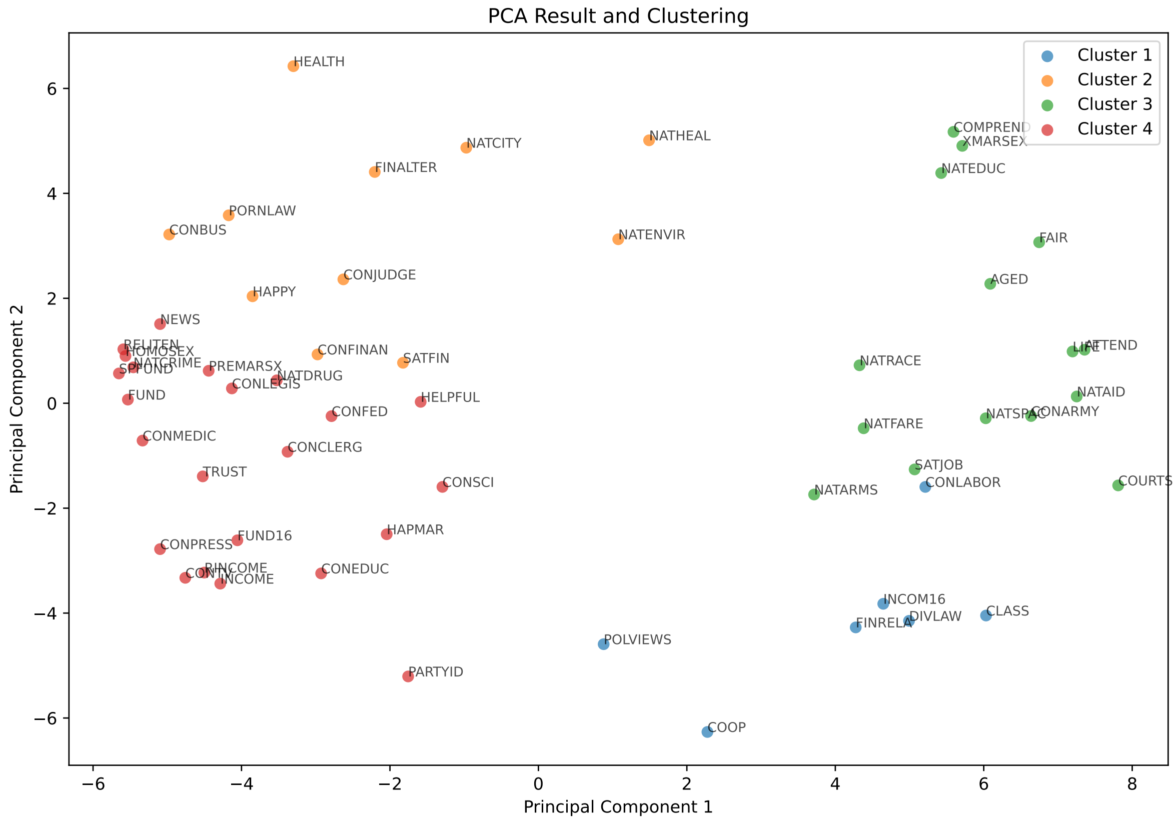
```
        x="abs_t_value",
        y="variable",
        hue="category",
        order=results_df.sort_values("abs_t_value", ascending=False)["variable"],
        palette=color_palette  # Use the soft color palette
    )
    plt.title("Variables by Absolute T-Value and Category")
    plt.xlabel("Absolute T-Value")
    plt.ylabel("Variables")
    plt.legend(title="Category")
    plt.tight_layout()
    plt.show()

    # Raincloud Plot
    plt.figure(figsize=(10, 6))
    sns.violinplot(
        data=results_df,
        x="category",
        y="abs_t_value",
        hue="category",
        inner="box",
        palette=color_palette  # Use the soft color palette
    )
    sns.stripplot(
        data=results_df,
        x="category",
        y="abs_t_value",
        color="gray",  # Use neutral color for strip points
        size=2,
        alpha=0.5,
        jitter=True,
        dodge=True
    )
    plt.title("Raincloud Plot: Absolute T-Values by Category")
    plt.ylabel("Absolute T-Value for Regression Coefficient")
    plt.xlabel("Category")
    plt.legend(title="Category")
    plt.tight_layout()
    plt.show()
```

```
# Main execution
if __name__ == "__main__":
```

```
variable_types = ["Likert Scale Variables"]
missing_threshold = 5

# Filter variables
selected_vars = filter_variables(variable_types, missing_threshold)
print(f"Selected Variables: {selected_vars}")

# Perform PCA and clustering
pca_result, clusters, variable_names = perform_pca_and_clustering(selected_vars)

# Display cluster information
cluster_info = generate_cluster_info(clusters, variable_names)
for cluster, vars_in_cluster in cluster_info.items():
    print(f"{cluster}: {vars_in_cluster}")

# Generate PCA plot
generate_pca_plot(pca_result, clusters, variable_names)

# Perform regression
regression_results = perform_regression(selected_vars)
print(regression_results)

# Select top 5 positive and 5 negative variables by MOBILITYLAG_coef
regression_results["abs_coef"] = regression_results["MOBILITYLAG_coef"].abs()
top_5_positive = regression_results.sort_values("MOBILITYLAG_coef", ascending=False).head
top_5_negative = regression_results.sort_values("MOBILITYLAG_coef", ascending=True).head
top_10_vars = pd.concat([top_5_positive, top_5_negative])

# Prepare data for fit-line plot
top_vars_for_fit = []
for _, row in top_10_vars.iterrows():
    var = row["Variable"]
    lag_var = f"{var}Lag"
    regression_data = cleaned_data[[var, lag_var, "MOBILITYLAG"]].dropna()

    X = regression_data[["MOBILITYLAG", lag_var]]
    y = regression_data[var]
    X = sm.add_constant(X)
    model = sm.OLS(y, X).fit()

    top_vars_for_fit.append((var, lag_var, model))
```

```python
    # Plot regression fit lines
    plot_regression_fit_lines(cleaned_data, top_vars_for_fit)

    # Generate barplot and raincloud plot
    generate_plots(regression_results, clusters, variable_names, ["HAPPY", "TRUST", "FAIR"])
```

```
Selected Variables: ['INCOM16', 'INCOME', 'RINCOME', 'PARTYID', 'POLVIEWS', 'NATSPAC', 'NATE

C:\Users\Aurora\Anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init

C:\Users\Aurora\Anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than a

Cluster 1: ['INCOM16', 'POLVIEWS', 'CONLABOR', 'CLASS', 'FINRELA', 'DIVLAW', 'COOP']
Cluster 2: ['NATENVIR', 'NATHEAL', 'NATCITY', 'HAPPY', 'HEALTH', 'CONFINAN', 'CONBUS', 'CONJ
Cluster 3: ['NATSPAC', 'NATEDUC', 'NATRACE', 'NATARMS', 'NATAID', 'NATFARE', 'COURTS', 'ATTE
Cluster 4: ['INCOME', 'RINCOME', 'PARTYID', 'NATCRIME', 'NATDRUG', 'FUND', 'RELITEN', 'FUND1
```



	Variable	R-squared	MOBILITYLAG_coef	Lag_coef	p-value (MOBILITYLAG)
0	INCOM16	0.349632	0.605932	0.021833	0.000327
1	INCOME	0.857104	-0.011820	0.897893	0.881496
2	RINCOME	0.852369	-0.016146	0.906253	0.851047
3	PARTYID	0.369032	-0.110403	0.560070	0.397747
4	POLVIEWS	0.256422	0.052881	0.506091	0.702383
5	NATSPAC	0.809107	0.261831	0.718413	0.013159
6	NATENVIR	0.291161	-0.121389	0.524541	0.373820
7	NATHEAL	0.359107	-0.132212	0.576263	0.310057
8	NATCITY	0.291323	-0.178681	0.483206	0.203033
9	NATCRIME	0.669806	-0.245629	0.626415	0.079623
10	NATDRUG	0.436751	-0.142801	0.577174	0.354858
11	NATEDUC	0.744171	0.034190	0.830840	0.720470
12	NATRACE	0.395229	0.259725	0.510744	0.082441
13	NATARMS	0.287628	0.449530	0.203132	0.003523
14	NATAID	0.752158	0.415447	0.563172	0.006596
15	NATFARE	0.399279	0.104576	0.591984	0.436708
16	COURTS	0.909077	0.413472	0.603531	0.004730

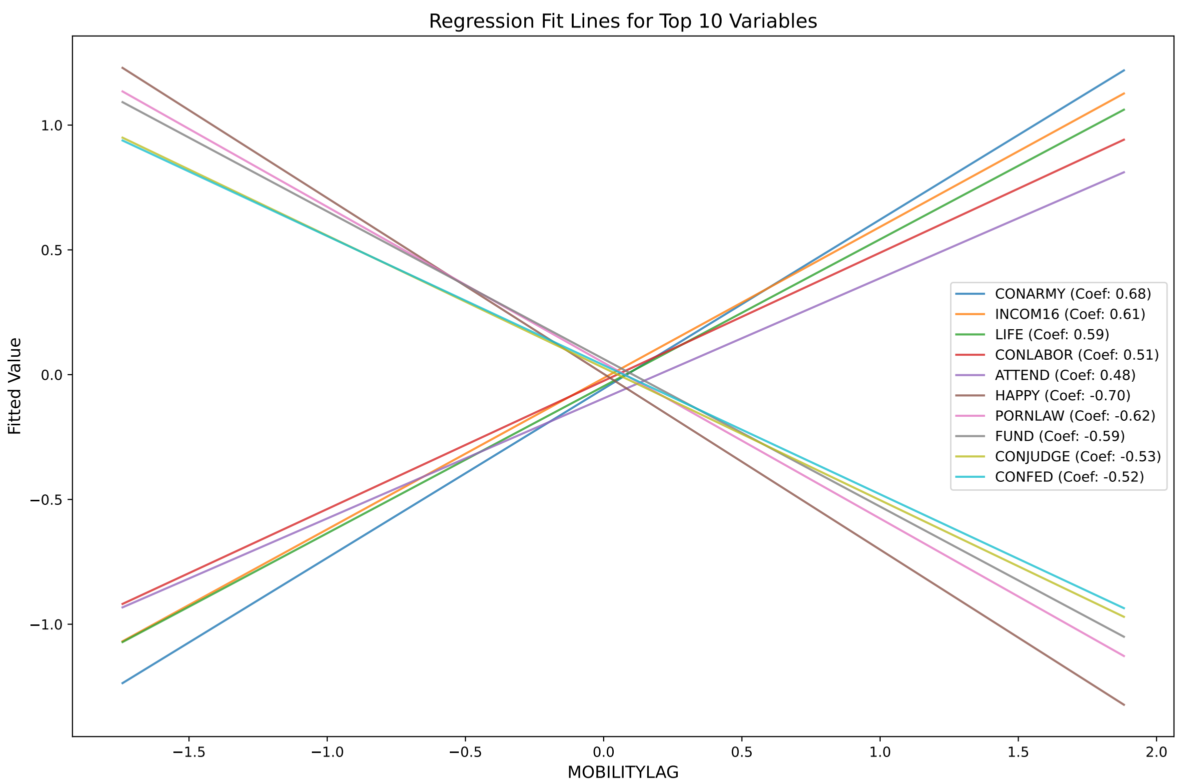
\

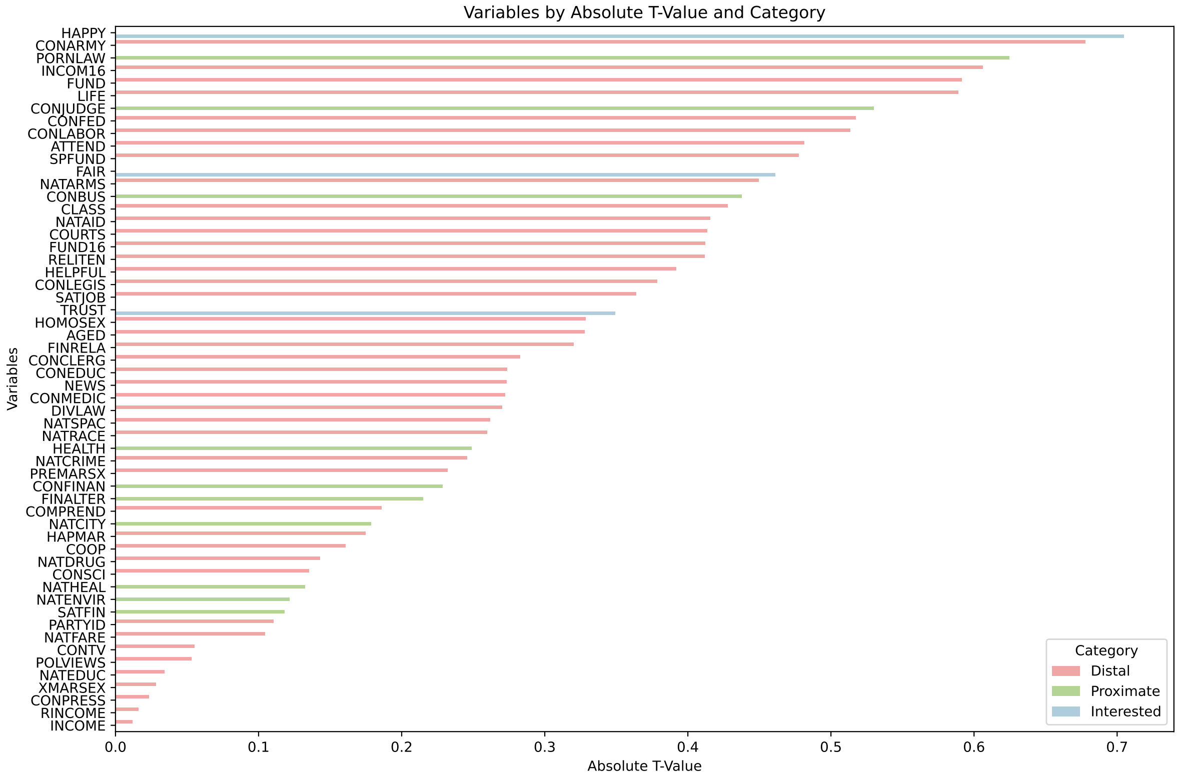
17	FUND	0.823022	-0.591361	0.375862	0.000417
18	ATTEND	0.823380	0.481285	0.459414	0.001223
19	RELITEN	0.851276	-0.411798	0.575858	0.011417
20	FUND16	0.466834	-0.412057	0.365759	0.008044
21	SPFUND	0.792352	-0.477446	0.467316	0.005218
22	HAPPY	0.545444	-0.704478	0.076012	0.000012
23	HAPMAR	0.237398	-0.174746	0.381176	0.239730
24	HEALTH	0.795703	-0.248964	0.736401	0.018568
25	LIFE	0.683288	0.588858	0.305798	0.000272
26	HELPFUL	0.129724	-0.391851	-0.136246	0.015426
27	FAIR	0.565535	0.460902	0.365767	0.004787
28	TRUST	0.581649	-0.348985	0.449649	0.031051
29	CONFINAN	0.593875	-0.228475	0.598375	0.101989
30	CONBUS	0.618772	-0.437587	0.407847	0.008682
31	CONCLERG	0.411215	-0.282758	0.428433	0.068023
32	CONEDUC	0.482008	-0.273609	0.502081	0.052554
33	CONFED	0.404906	-0.517333	0.213210	0.001382
34	CONLABOR	0.387771	0.513518	0.223179	0.000959
35	CONPRESS	0.851634	0.023306	0.925501	0.827054
36	CONMEDIC	0.762794	-0.272328	0.638587	0.061434
37	CONTV	0.841204	-0.055180	0.866860	0.551921
38	CONJUDGE	0.449008	-0.530020	0.233343	0.000886
39	CONSCI	0.017842	-0.135122	-0.073562	0.403979
40	CONLEGIS	0.809888	-0.378347	0.592758	0.002253
41	CONARMY	0.575449	0.677775	0.135499	0.000269
42	AGED	0.679948	0.327903	0.575656	0.017723
43	SATJOB	0.206936	0.363701	0.168566	0.033374
44	CLASS	0.745311	0.427836	0.495030	0.004023
45	SATFIN	0.426422	-0.118120	0.576344	0.385252
46	FINALTER	0.411221	-0.214999	0.502936	0.143049
47	FINRELA	0.391620	0.320080	0.411119	0.032022
48	DIVLAW	0.764618	0.270276	0.770787	0.021546
49	PREMARSX	0.879038	-0.231999	0.719013	0.025257
50	XMARSEX	0.563396	0.028309	0.711835	0.822504
51	HOMOSEX	0.919319	-0.328389	0.684967	0.053532
52	PORNLAW	0.677463	-0.624512	0.289227	0.000073
53	NEWS	0.899935	-0.273282	0.740811	0.062181
54	COOP	0.493158	0.160747	0.655171	0.171569
55	COMPREND	0.462921	0.186006	0.566009	0.167465

	t-value
0	3.905978
1	-0.149961

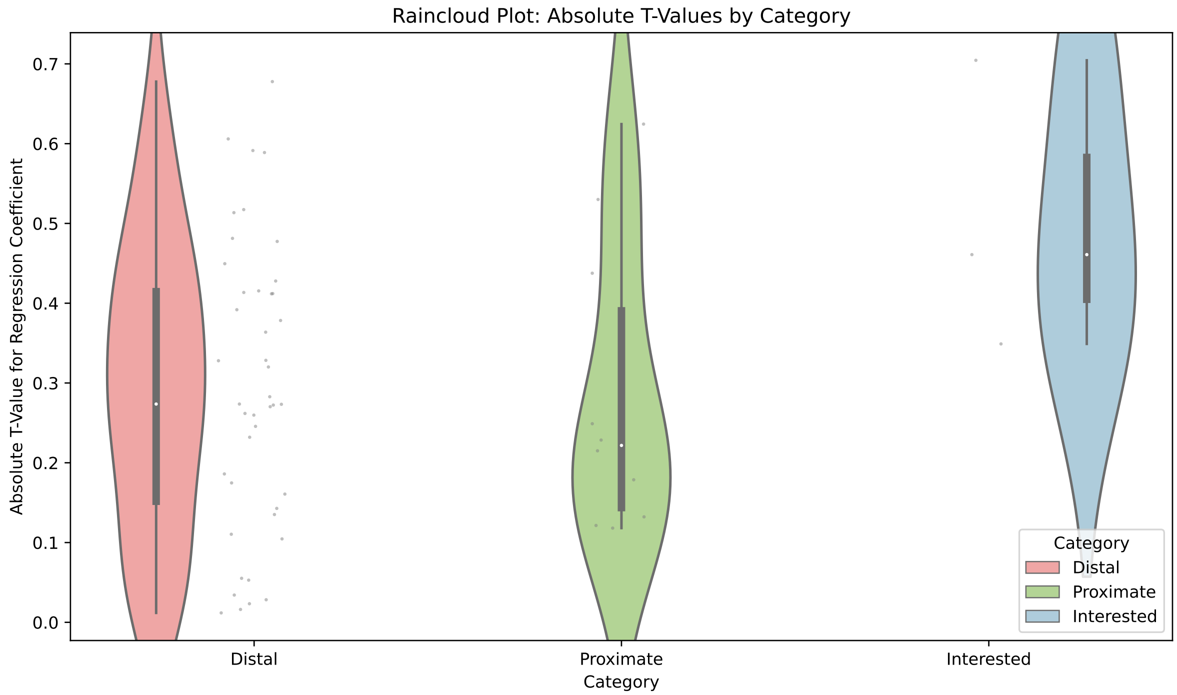
2	-0.188917
3	-0.854165
4	0.384663
5	2.586662
6	-0.898695
7	-1.027227
8	-1.292656
9	-1.795381
10	-0.935297
11	0.360188
12	1.778210
13	3.087961
14	2.855156
15	0.785086
16	2.979775
17	-3.825530
18	3.462603
19	-2.643054
20	-2.779450
21	-2.943280
22	-4.937239
23	-1.192177
24	-2.446972
25	3.965800
26	-2.522706
27	2.975296
28	-2.229571
29	-1.670982
30	-2.750034
31	-1.871967
32	-1.993683
33	-3.420145
34	3.545997
35	0.219817
36	-1.920562
37	-0.599598
38	-3.572728
39	-0.842846
40	-3.248674
41	3.969128
42	2.466126
43	2.198098
44	3.039451

```
45 -0.877202
46 -1.491813
47  2.216161
48  2.385236
49 -2.318277
50  0.225700
51 -1.985130
52 -4.387124
53 -1.914834
54  1.390379
55  1.404121
```






```
C:\Users\Aurora\Anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf value

C:\Users\Aurora\Anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf value
```




```python
# Create and run the app
import nest_asyncio
nest_asyncio.apply()
app = App(app_ui, server)
app.run(host="127.0.0.1", port=8273)
```

Output analysis

Based on the results, we observe that the p-values for happy, fair, and trust are all below or around 0.05, indicating that mobility has a statistically significant impact on these variables. Notably, the coefficient for fair is positive(0.25), suggesting that as mobility increases, individuals perceive a greater sense of fairness in societal interactions. This could indicate that greater residential mobility exposes individuals to more diverse environments, fostering a sense of equitable opportunity and fairness in society.

On the other hand, while the literature highlights a positive link between mobility and happiness, our results show that the coefficients for happy and trust are negative(-0.58 and -0.26), implying that increased residential mobility correlates with a decrease in reported happiness and trust. This finding reveals a potential social insight: while mobility may broaden individuals' exposure and opportunities, it can also disrupt social ties and create instability, leading to feelings of alienation or a loss of trust in others. These results suggest a dual-edged nature of mobility—it can promote perceptions of fairness while simultaneously eroding the emotional and social foundations of happiness and trust. This highlights the need for policymakers to balance efforts to increase mobility with initiatives that strengthen community cohesion and social support systems.

From the results of the histogram and raincloud plots, we uncover insights that diverge from those reported in the existing literature. While there is no cherry-picking issue in the literature, it does exist in our study.

Bar Chart

Blue bars represent our focal variables, showing the strongest effects, consistently ranking from 1st to 19th. In contrast, proximal-other variables (green bars), which are more strongly correlated with the focal variables, display a wider range of effect sizes, ranking from 2nd to near the bottom. Similarly, distal-other variables (red bars), which are less correlated with the focal variables, exhibit a similar wide range, ranking from 2nd to last.

Notably, the focal variables have a smaller range of effect sizes and higher t-values overall, suggesting a stronger and more consistent influence. However, this does not entirely rule out the possibility of cherry-picking.

Raincloud Plot

The raincloud plot further tests whether the mean effect-size distribution associated with our focal variables differs from that of other dependent variables.

A permutation-based comparison of effect sizes revealed a significant difference between the focal variables and all other groups. This finding suggests that cherry-picking might be present, as the observed differences indicate a non-random selection of variables.

Policy Implications

The findings suggest that residential mobility has a complex and dual-edged impact on individual and societal well-being. The positive correlation between mobility and perceived fairness indicates that mobility can create an environment where individuals experience greater equity and societal justice. Policymakers can leverage this by encouraging mobility through programs such as job relocation assistance, education grants, or housing subsidies that promote equitable access to opportunities.

However, the negative impact of mobility on happiness and trust raises concerns about the social and psychological costs of increased mobility. As individuals move more frequently, the erosion of trust and community ties can lead to feelings of alienation and lower overall life satisfaction. Policymakers must address these unintended consequences by fostering stronger social networks and support systems. For example:

- Community-Building Programs: Investing in initiatives that strengthen community bonds, such as local engagement programs, shared public spaces, or neighborhood events, can help mitigate the social disconnection caused by mobility.
- Mental Health Support: Introducing mental health resources and support systems for individuals experiencing mobility-related stress or loneliness could alleviate negative emotional outcomes.
- Trust-Building Campaigns: Promoting social trust through targeted campaigns, inclusive policies, and fostering shared cultural or civic values may counteract the erosion of interpersonal trust.

These findings highlight the importance of designing mobility-enhancing policies that are socially conscious and proactive in addressing the trade-offs between opportunity creation and emotional well-being.

Directions for Future Work

While this study sheds light on the impact of residential mobility on fairness, happiness, and trust, there are several areas for future exploration:

- Longitudinal Analysis: Since we only study patterns across 1978-2018, investigating a longer-term effects of mobility on well-being continuously could provide deeper insights into whether the observed impacts persist, diminish, or intensify over time.
- Geographical Variations: Analyzing mobility's effects in urban vs. rural settings or across different socioeconomic strata could uncover regional or demographic differences in outcomes.
- Mechanisms Behind Trust Decline: Future research could focus on understanding the mechanisms that drive the decline in trust due to mobility. For example, does it stem from weaker interpersonal relationships, cultural dissonance, or perceived competition?
- Experimental Approaches: Designing controlled experiments or natural experiments, for example, randomly assigning residents to move from one place to another to avoid pre-existing different baseline personal characteristics and validate causal relationships between mobility and personal characteristics could strengthen the robustness of these findings.

By addressing these areas, future work can provide more granular and actionable insights for policymakers, ensuring that residential mobility fosters not only equitable opportunities but also emotional and social well-being.