```
import pandas as pd
df= pd.read_excel(r"C:¥Users¥strat¥Downloads¥cln_06-26 - コピー.xlsx")
df
```

	city	year	Unnamed:_0	世帯数分布(抽出率調整)	集計世帯数	世帯人員_(人)	18歳未満人員_(人)	65歳以上人員_(人)	うち無職者人員_(人)	有業人員_(人)	...	光熱・水道.1	家具・家事用品.2	被服及び履物.2	保健医療.1	交通・通信.1
0	Number of households	2000	2.0	...	...	...	...	...	...	...	...	28.0	1465.0	668.0	225.0	228.0
1	Chukyo	2000	21.0	773	425	3.42	0.67	0.51	0.38	1.72	...	18.0	533.0	433.0	63.0	575.0
2	China	2000	16.0	619	668	3.29	0.73	0.62	0.48	1.52	...	9.0	460.0	502.0	69.0	204.0
3	medium city	2000	6.0	3192	4156	3.23	0.72	0.49	0.4	1.42	...	12.0	539.0	505.0	82.0	434.0
4	Kyushu	2000	18.0	1056	1060	3.25	0.81	0.55	0.45	1.42	...	10.0	534.0	803.0	102.0	160.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1773	Tokai region	2024	11.0	1187	693	2.95	0.56	0.82	0.64	1.38	...	NaN	NaN	NaN	NaN	NaN
1774	Okinawa Region	2024	16.0	108	217	3.02	0.72	0.76	0.59	1.23	...	NaN	NaN	NaN	NaN	NaN
1775	Kinki Region	2024	12.0	1669	972	2.85	0.54	0.85	0.68	1.28	...	NaN	NaN	NaN	NaN	NaN

```
exclude_values = ["That's all", "Nationwide", "All cities", "#VALUE!", "Number of households"]

# Drop those rows without changing order
df = df[~df["city"].isin(exclude_values)]
df
```

	city	year	Unnamed:_0	世帯数分布(抽出率調整)	集計世帯数	世帯人員_(人)	18歳未満人員_(人)	65歳以上人員_(人)	うち無職者人員_(人)	有業人員_(人)	...	光熱・水道.1	家具・家事用品.2	被服及び履物.2	保健医療.1	交通・通信.1	教育
1	Chukyo	2000	21.0	773	425	3.42	0.67	0.51	0.38	1.72	...	18.0	533.0	433.0	63.0	575.0	
2	China	2000	16.0	619	668	3.29	0.73	0.62	0.48	1.52	...	9.0	460.0	502.0	69.0	204.0	
3	medium city	2000	6.0	3192	4156	3.23	0.72	0.49	0.4	1.42	...	12.0	539.0	505.0	82.0	434.0	
4	Kyushu	2000	18.0	1056	1060	3.25	0.81	0.55	0.45	1.42	...	10.0	534.0	803.0	102.0	160.0	
5	Keihin Leaf	2000	20.0	2858	1392	3.23	0.67	0.46	0.37	1.49	...	25.0	670.0	619.0	94.0	612.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1773	Tokai region	2024	11.0	1187	693	2.95	0.56	0.82	0.64	1.38	...	NaN	NaN	NaN	NaN	NaN	N
1774	Okinawa Region	2024	16.0	108	217	3.02	0.72	0.76	0.59	1.23	...	NaN	NaN	NaN	NaN	NaN	N
1775	Kinki Region	2024	12.0	1669	972	2.85	0.54	0.85	0.68	1.28	...	NaN	NaN	NaN	NaN	NaN	N
1776	Kanto region	2024	9.0	3724	1813	2.88	0.52	0.82	0.65	1.34	...	NaN	NaN	NaN	NaN	NaN	N
1777	NaN	2024	NaN	10101	10101	10101	10101	10101	10101	10101	...	10101.0	10101.0	10101.0	10101.0	10101.0	1010

```
cities =df["city"].unique()
city_list = []
for x in cities:
    city_list.append(x)


city_list
```

```
['Chukyo',
 'China',
 'medium city',
 'Kyushu',
 'Keihin Leaf',
```

```
        'Kyoto City',
        'Keihanshin',
        'Sendai City',
        'Saga City',
        'Maebashi City',
        'Hokuriku',
        'Hokkaido',
        'Kitakyushu City',
        'Kitakyushu',
        'Chiba City',
        'Nagoya City',
        'Wakayama City',
        'Shikoku',
        'Oita City',
        'Otsu City',
        'big city',
        'Osaka City',
        'Nara City',
        'Utsunomiya City',
        'Miyazaki City',
        'Toyama City',
        'Small City A',
        'Small city B',
        'Yamaguchi City',
        'Yamagata City',
        'Gifu City',
        'Okayama City',
        'Kawasaki City',
        'Hiroshima City',
        'Tokushima City',
        'Niigata City',
        'Sapporo City',
        'Northeast',
        'Tokai',
        'Tokyo wards',
        'matsuyama city',
        'Matsue City',
        'Yokohama City',
        'Mito City',
        'Okinawa',
        'Tsu city',
        'Urawa City',
        'kumamoto city',
        'Kofu City',
        'Towns and Villages',
        'Morioka City',
        'Kobe City',
        'Fukui City',
        'Fukuoka City',
        'Fukushima city',
        'Akita City',
        'Kinki',
        'Naha City',

len(city_list)
```

```
137
```

```
import pandas as pd

# Example df for context:
# df = pd.DataFrame({'cities': [...your city list here...]})

# Mapping dictionary for normalization
city_map = {
    # Variants without codes to clean names
    'Kitakyushu': 'Kitakyushu City',
    'kumamoto city': 'Kumamoto City',
    'matsuyama city': 'Matsuyama City',
    'Fukushima city': 'Fukushima City',
    'Tsu city': 'Tsu City',
    'Small City B/Town/Village': 'Small City B',

    # Codes with names to just the city name
    '01100 Sapporo City': 'Sapporo City',
    '02201 Aomori City': 'Aomori City',
    '03201 Morioka City': 'Morioka City',
    '04100 Sendai City': 'Sendai City',
    '05201 Akita City': 'Akita City',
    '06201 Yamagata City': 'Yamagata City',
    '07201 Fukushima City': 'Fukushima City',
    '08201 Mito City': 'Mito City',
    '09201 Utsunomiya City': 'Utsunomiya City',
    '10201 Maebashi City': 'Maebashi City',
    '11100 Saitama City': 'Saitama City',
    '12100 Chiba City': 'Chiba City',
```

```
        '13100 Tokyo wards': 'Tokyo wards',
        '14100 Yokohama City': 'Yokohama City',
        '14130 Kawasaki City': 'Kawasaki City',
        '14150 Sagamihara City': 'Sagamihara City',
        '15100 Niigata City': 'Niigata City',
        '16201 Toyama City': 'Toyama City',
        '17201 Kanazawa City': 'Kanazawa City',
        '18201 Fukui City': 'Fukui City',
        '19201 Kofu City': 'Kofu City',
        '20201 Nagano City': 'Nagano City',
        '21201 Gifu City': 'Gifu City',
        '22100 Shizuoka City': 'Shizuoka City',
        '22130 Hamamatsu City': 'Hamamatsu City',
        '23100 Nagoya City': 'Nagoya City',
        '24201 Tsu City': 'Tsu City',
        '25201 Otsu City': 'Otsu City',
        '26100 Kyoto City': 'Kyoto City',
        '27100 Osaka City': 'Osaka City',
        '27140 Sakai City': 'Sakai City',
        '28100 Kobe City': 'Kobe City',
        '29201 Nara City': 'Nara City',
        '30201 Wakayama City': 'Wakayama City',
        '31201 Tottori City': 'Tottori City',
        '32201 Matsue City': 'Matsue City',
        '33100 Okayama City': 'Okayama City',
        '34100 Hiroshima City': 'Hiroshima City',
        '35203 Yamaguchi City': 'Yamaguchi City',
        '36201 Tokushima City': 'Tokushima City',
        '37201 Takamatsu City': 'Takamatsu City',
        '38201 Matsuyama City': 'Matsuyama City',
        '39201 Kochi City': 'Kochi City',
        '40100 Kitakyushu City': 'Kitakyushu City',
        '40130 Fukuoka City': 'Fukuoka City',
        '41201 Saga City': 'Saga City',
        '42201 Nagasaki City': 'Nagasaki City',
        '43100 Kumamoto City': 'Kumamoto City',
        '44201 Oita City': 'Oita City',
        '45201 Miyazaki City': 'Miyazaki City',
        '46201 Kagoshima City': 'Kagoshima City',
        '47201 Naha City': 'Naha City',

        # Regions to a normalized form (optional, since these are not cities)
        'Chugoku Region': 'Chugoku Region',
        'Kyushu region': 'Kyushu Region',
        'Hokkaido Region': 'Hokkaido Region',
        'Hokuriku Region': 'Hokuriku Region',
        'Shikoku Region': 'Shikoku Region',
        'Tohoku Region': 'Tohoku Region',
        'Tokai region': 'Tokai Region',
        'Okinawa Region': 'Okinawa Region',
        'Kinki Region': 'Kinki Region',
        'Kanto region': 'Kanto Region',

        # Big/small city categories
        'Big cities': 'Big cities',
        'Small City B/Town/Village': 'Small City B',

        # Some others without change
        'China': 'China',
        'medium city': 'medium city',
        'big city': 'big city',
        'Towns and Villages': 'Towns and Villages',
        'Tokyo wards': 'Tokyo wards',
        'Chukyo': 'Chukyo',
        'Keihin Leaf': 'Keihin Leaf',
        'Keihanshin': 'Keihanshin',
        'Northeast': 'Northeast',
        'Tokai': 'Tokai',
        'Kinki': 'Kinki',
        'Kanto': 'Kanto',
        'Kyushu': 'Kyushu',
        'Hokkaido': 'Hokkaido',
        'Hokuriku': 'Hokuriku',
        'Shikoku': 'Shikoku',
        'Okinawa': 'Okinawa',
    }

    # Replace values using map, keep original if no mapping
    df['city'] = df['city'].map(city_map).fillna(df['city'])
    df[["city", "year"]]
```

```
C:¥Users¥strat¥AppData¥Local¥Temp¥ipykernel_12948¥277811384.py:107: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['city'] = df['city'].map(city_map).fillna(df['city'])
```

	city	year
1	Chukyo	2000
2	China	2000
3	medium city	2000
4	Kyushu	2000
5	Keihin Leaf	2000
...	...	...
1773	Tokai Region	2024
1774	Okinawa Region	2024
1775	Kinki Region	2024
1776	Kanto Region	2024
1777	NaN	2024

1702 rows × 2 columns

```
len(city_list)
```

```
137
```

```
df = df.set_index(["city","year"])
df["const"]= 1
df
```

		Unnamed:_0	世帯数分布(抽出率調整)	集計世帯数	世帯人員_(人)	18歳未満人員_(人)	65歳以上人員_(人)	うち無職者人員_(人)	有業人員_(人)	世帯主の配偶者のうち女の有業率(%)	世帯主の年齢_(歳)	...	家具・家事用品.2	被服及び履物.2	保健医療.1	交通・通信.1	教育
city	**year**																
Chukyo	**2000**	21.0	773	425	3.42	0.67	0.51	0.38	1.72	44.7	52.9	...	533.0	433.0	63.0	575.0	
China	**2000**	16.0	619	668	3.29	0.73	0.62	0.48	1.52	40.6	53.6	...	460.0	502.0	69.0	204.0	
medium city	**2000**	6.0	3192	4156	3.23	0.72	0.49	0.4	1.42	32.8	52.4	...	539.0	505.0	82.0	434.0	
Kyushu	**2000**	18.0	1056	1060	3.25	0.81	0.55	0.45	1.42	37.9	52.8	...	534.0	803.0	102.0	160.0	
Keihin Leaf	**2000**	20.0	2858	1392	3.23	0.67	0.46	0.37	1.49	33.9	52.3	...	670.0	619.0	94.0	612.0	
...	**...**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
Tokai Region	**2024**	11.0	1187	693	2.95	0.56	0.82	0.64	1.38	44.5	59.8	...	NaN	NaN	NaN	NaN	N
Okinawa Region	**2024**	16.0	108	217	3.02	0.72	0.76	0.59	1.23	33.1	58.2	...	NaN	NaN	NaN	NaN	N
Kinki Region	**2024**	12.0	1669	972	2.85	0.54	0.85	0.68	1.28	40.2	60.5	...	NaN	NaN	NaN	NaN	N
Kanto Region	**2024**	9.0	3724	1813	2.88	0.52	0.82	0.65	1.34	42.3	60.2	...	NaN	NaN	NaN	NaN	N
NaN	**2024**	NaN	10101	10101	10101	10101	10101	10101	10101	10101	10101		10101.0	10101.0	10101.0	10101.0	1010

コーディングを開始するか、AI で生成します。

```
df
```

city	year	Unnamed: 0	世帯数分布(抽出率調整)	集計世帯数	世帯人員_(人)	18歳未満人員_(人)	65歳以上人員_(人)	うち無職者人員_(人)	有業人員_(人)	世帯主の配偶者のうち女の有業率(%)	世帯主の年齢_(歳)	...	家具・家事用品.2	被服及び履物.2	保健医療.1	交通・通信.1	教育
Chukyo	**2000**	21.0	773	425	3.42	0.67	0.51	0.38	1.72	44.7	52.9	...	533.0	433.0	63.0	575.0	
China	**2000**	16.0	619	668	3.29	0.73	0.62	0.48	1.52	40.6	53.6	...	460.0	502.0	69.0	204.0	
medium city	**2000**	6.0	3192	4156	3.23	0.72	0.49	0.4	1.42	32.8	52.4	...	539.0	505.0	82.0	434.0	
Kyushu	**2000**	18.0	1056	1060	3.25	0.81	0.55	0.45	1.42	37.9	52.8	...	534.0	803.0	102.0	160.0	
Keihin Leaf	**2000**	20.0	2858	1392	3.23	0.67	0.46	0.37	1.49	33.9	52.3	...	670.0	619.0	94.0	612.0	
...	**...**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
Tokai Region	**2024**	11.0	1187	693	2.95	0.56	0.82	0.64	1.38	44.5	59.8	...	NaN	NaN	NaN	NaN	N
Okinawa Region	**2024**	16.0	108	217	3.02	0.72	0.76	0.59	1.23	33.1	58.2	...	NaN	NaN	NaN	NaN	N
Kinki Region	**2024**	12.0	1669	972	2.85	0.54	0.85	0.68	1.28	40.2	60.5	...	NaN	NaN	NaN	NaN	N
Kanto Region	**2024**	9.0	3724	1813	2.88	0.52	0.82	0.65	1.34	42.3	60.2	...	NaN	NaN	NaN	NaN	N
NaN	**2024**	NaN	10101	10101	10101	10101	10101	10101	10101	10101	10101		10101.0	10101.0	10101.0	10101.0	1010

```
for x in df.columns:
    print(x)
```

家具・家事用品.2

```
家具・家事用品.2
被服及び履物.2
保健医療.1
交通・通信.1
教育.1
教養娯楽.2
その他の消費支出.1
エンゲル係数_(%)
調整集計世帯数
const
```

コーディングを開始するか、AI で生成します。

```
from linearmodels.panel import PanelOLS
import statsmodels.api as sm
model = PanelOLS(df['仕送り金'], df[['const', '世帯人員_(人)', 'エンゲル係数_(%)']], entity_effects=True, time_effects=True, check_rank=False)
results = model.fit(cov_type='clustered', cluster_entity=True)
print(results.summary)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[204], line 4
      2 import statsmodels.api as sm
      3 model = PanelOLS(df['仕送り金'], df[['const', '世帯人員_(人)', 'エンゲル係数_(%)']], entity_effects=True, time_effects=True,
check_rank=False)
----> 4 results = model.fit(cov_type='clustered', cluster_entity=True)
      5 print(results.summary)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥linearmodels¥panel¥model.py:1918, in PanelOLS.fit(self, use_lsdv,
use_lsmr, low_memory, cov_type, debiased, auto_df, count_effects, **cov_config)
   1914 low_memory = (
   1915     self._choose_twoway_algo() if low_memory is None else low_memory
   1916 )
   1917 if not weighted:
-> 1918     y, x, ybar = self._fast_path(low_memory=low_memory)
   1919     y_effects = np.array([0.0])
   1920     x_effects = np.zeros(x.shape)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥linearmodels¥panel¥model.py:1664, in PanelOLS._fast_path(self,
low_memory)
   1662     x = cast(PanelData, x.general_demean(groups))
   1663 elif self.entity_effects and self.time_effects:
-> 1664     y = cast(PanelData, y.demean("both", low_memory=low_memory))
   1665     x = cast(PanelData, x.demean("both", low_memory=low_memory))
   1666 elif self.entity_effects:

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥linearmodels¥panel¥data.py:584, in PanelData.demean(self, group,
weights, return_panel, low_memory)
    582 if group == "both":
    583     if not low_memory:
--> 584         return self._demean_both(weights)
    585     else:
    586         return self._demean_both_low_mem(weights)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥linearmodels¥panel¥data.py:416, in PanelData._demean_both(self,
weights)
    414 d_arr = d_pd.values2d
    415 e_arr = e.values2d
--> 416 resid_arr = e_arr - d_arr @ lstsq(d_arr, e_arr, rcond=None)[0]
    417 resid = DataFrame(
    418     resid_arr, index=self._frame.index, columns=self._frame.columns
    419 )
    421 return PanelData(resid)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥numpy¥linalg¥linalg.py:2326, in lstsq(a, b, rcond)
   2323 if n_rhs == 0:
   2324     # lapack can't handle n_rhs = 0 - so allocate the array one larger in that axis
   2325     b = zeros(b.shape[:-2] + (m, n_rhs + 1), dtype=b.dtype)
-> 2326 x, resids, rank, s = gufunc(a, b, rcond, signature=signature, extobj=extobj)
   2327 if m == 0:
   2328     x[...] = 0

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥numpy¥linalg¥linalg.py:124, in _raise_linalgerror_lstsq(err, flag)
```

コーディングを開始するか、AI で生成します。

```
⚠ Column '仕送り金' not found in df_diff
```

世帯数分布(抽出率調整) 集計世帯数 世帯人員*(人)* *18歳未満人員*(人) 65歳以上人員*(人)* *うち無職者人員*(人) 有業人員*(人)* *世帯主の配偶者のうち女の有業率(%)* *世帯主の年齢*(歳) 持家率(%) 平均畳数 家賃・地代を支払っている世帯の割合(%) 平均畳数.1 農林漁家世帯の割合(%) 消費支出 食料 穀類 米 パン めん類 他の穀類 魚介類 生鮮魚介 塩干魚介 魚肉練製品 他の魚介加工品 肉類 生鮮肉 加工肉 乳卵類 牛乳 乳製品 卵 野菜・海藻 生鮮野菜 乾物・海藻 大豆加工品 他の野菜・海藻加工品 果物 生鮮果物 果物加工品 油脂・調味料 油脂 調味料 菓子類 調理食品 主食

的調理食品 他の調理食品 飲料 茶類 コーヒー・ココア 他の飲料 酒類 外食 一般外食 学校給食 住居 家賃地代 設備修繕・維持 設備材料 工事その他のサービス 光熱・水道 電気代 ガス代 他の光熱 上下水道料 家具・家事用品 家庭用耐久財 家事用耐久財 冷暖房用器具 一般家具 室内装備・装飾品 寝具類 家事雑貨 家事用消耗品 家事サービス 被服及び履物 和服 洋服 男子用洋服 婦人用洋服 子供用洋服 シャツ・セーター類 男子用シャツ・セーター類 婦人用シャツ・セーター類 子供用シャツ・セーター類 下着類 男子用下着類 婦人用下着類 子供用下着類 生地・糸類 他の被服 履物類 被服関連サービス 保健医療 医薬品 健康保持用摂取品 保健医療用品・器具 保健医療サービス 交通・通信 交通 自動車等関係費 自動車等購入 自転車購入 自動車等維持 通信 教育 授業料等 教科書・学習参考教材 補習教育 教養娯楽 教養娯楽用耐久財 教養娯楽用品 書籍・他の印刷物 教養娯楽サービス 宿泊料 パック旅行費 月謝類 他の教養娯楽サービス その他の消費支出 諸雑費 理美容サービス 理美容用品 身の回り用品 たばこ その他の諸雑費 こづかい(使途不明) 交際費 食料.1 家具・家事用品.1 被服及び履物.1 教養娯楽.1 他の物品サービス 贈与金 他の交際費 仕送り金 (再掲)教育関係費 (再掲)*教養娯楽関係費 (再掲)移転支出(贈与金+仕送り金) (再掲)_経常消費支出 (再掲)_情報通信関係費 (再掲)_消費支出(除く住居等)_1) 現物総額 食料.2 自家産物 住居.1 光熱・水道.1 家具・家事用品.2 被服及び履物.2 保健医療.1 交通・通信.1 教育.1 教養娯楽.2 その他の消費支出.1 エンゲル係数*(%) 調整集計世帯数 const

```
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Ensure numeric and safe logging
df = df.apply(pd.to_numeric, errors='coerce')
for col in df.columns:
    if pd.api.types.is_numeric_dtype(df[col]) and (df[col] > 0).all():
        df[col] = np.log(df[col])

# Sort and first-difference
df = df.sort_values(['city', 'year'])
df_diff = df.groupby('city').diff()

# Define your regressors in one place
regressors = [    '18歳未満人員_(人)',                    # Number of children
    '65歳以上人員_(人)',                   # Number of elderly
    '有業人員_(人)',                       # Number of employed persons
    '世帯主の配偶者のうち女の有業率(%)',     # Female spouse employment rate
    '世帯主の年齢_(歳)','農林漁家世帯の割合(%)', 'エンゲル係数_(%)',
    "光熱・水道",
    "家賃・地代を支払っている世帯の割合(%)"]

# Ensure they and the target are numeric
for col in ['仕送り金'] + regressors:
    df_diff[col] = pd.to_numeric(df_diff[col], errors='coerce')

# Define target and X
y_diff = df_diff['仕送り金']
X_diff = df_diff[regressors]
X_diff = sm.add_constant(X_diff)

# Drop missing/infinite rows
df_model = pd.concat([y_diff, X_diff], axis=1).replace([np.inf, -np.inf], np.nan).dropna()
y_diff_clean = df_model['仕送り金']
X_diff_clean = df_model.drop(columns='仕送り金')

# Run regression
model_fd = sm.OLS(y_diff_clean, X_diff_clean).fit()
print(model_fd.summary())
```

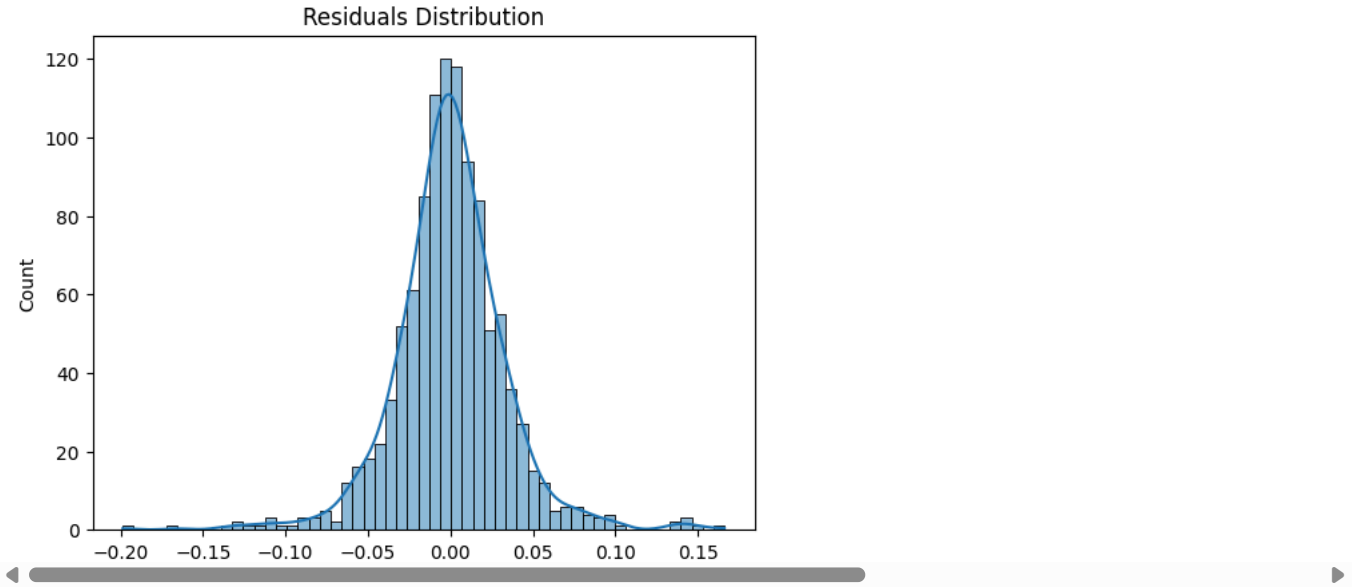
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  仕送り金   R-squared:                       0.093
Model:                            OLS   Adj. R-squared:                  0.086
Method:                 Least Squares   F-statistic:                     12.24
Date:                Sun, 06 Jul 2025   Prob (F-statistic):           1.37e-18
Time:                        13:04:56   Log-Likelihood:                 2129.0
No. Observations:                1083   AIC:                            -4238.
Df Residuals:                    1073   BIC:                            -4188.
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           -0.0011      0.001     -1.042      0.298      -0.003       0.001
18歳未満人員_(人)      -0.0024      0.010     -0.233      0.816      -0.022       0.018
65歳以上人員_(人)      -0.0285      0.014     -2.053      0.040      -0.056      -0.001
有業人員_(人)          0.0746      0.021      3.501      0.000       0.033       0.116
世帯主の配偶者のうち女の有業率(%)    -0.0167      0.007     -2.551      0.011      -0.029      -0.004
世帯主の年齢_(歳)        0.2993      0.141      2.117      0.034       0.022       0.577
農林漁家世帯の割合(%)      -0.0005      0.001     -0.580      0.562      -0.002       0.001
エンゲル係数_(%)       -0.0933      0.011     -8.154      0.000      -0.116      -0.071
光熱・水道             0.8524      0.417      2.043      0.041       0.034       1.671
家賃・地代を支払っている世帯の割合(%)    -0.0313      0.011     -2.788      0.005      -0.053      -0.009
==============================================================================
Omnibus:                      111.552   Durban-Watson:                   2.741
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              821.308
Skew:                          -0.024   Prob(JB):                   4.52e-179
Kurtosis:                       7.266   Cond. No.                         535.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
model_fd = sm.OLS(y_diff_clean, X_diff_clean).fit(cov_type='HC1')
```

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

sns.histplot(model_fd.resid, kde=True)
plt.title("Residuals Distribution")
```

```
Text(0.5, 1.0, 'Residuals Distribution')
```




```
target_cols = ['仕送り金', '農林漁家世帯の割合(%)', 'エンゲル係数_(%)']
for col in target_cols:
    if col in df_diff.columns:
        df_diff[col] = pd.to_numeric(df_diff[col], errors='coerce')
    else:
        print(f"⚠️ Column '{col}' not found in df_diff")
```

```
df_diff
```

		Unnamed: 0	世帯数分布(抽出率調整)	集計世帯数	世帯人員_(人)	18歳未満人員_(人)	65歳以上人員_(人)	うち無職者人員_(人)	有業人員_(人)	世帯主の配偶者のうち女の有業率(%)	世帯主の年齢_(歳)	...	家具・家事用品.2	被服及び履物.2
city	year													
Akita City	2000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	2001	0.0	0.000000	0.000000	-0.034869	-0.075035	0.022473	0.000000	0.000000	0.128874	0.011561	...	34.0	-77.0
	2003	0.0	0.039221	0.000000	-0.022839	-0.216671	0.287682	0.332706	-0.040491	-0.092373	0.068482	...	-343.0	-300.0
	2004	0.0	0.000000	0.000000	0.073165	0.121361	0.000000	-0.019048	0.079407	0.085816	-0.027200	...	163.0	-11.0
	2005	0.0	0.000000	-0.010471	-0.027996	0.095310	-0.105361	-0.101096	-0.071176	-0.043704	-0.045120	...	-194.0	28.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
NaN	2016	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	2017	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	2017	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	2022	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
	2024	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

1702 rows × 158 columns

```
# Create year dummies after dropping first obs per group (already done in df_diff)
df_diff['year'] = df['year']  # reattach 'year' column since it was removed in .diff()
year_dummies = pd.get_dummies(df_diff['year'], prefix='year', drop_first=True)
X_diff = pd.concat([X_diff, year_dummies], axis=1)

# Re-run regression
model_fd = sm.OLS(y_diff, X_diff).fit()
print(model_fd.summary())
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥pandas¥core¥indexes¥base.py:3805, in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas¥¥_libs¥¥hashtable_class_helper.pxi:7081, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas¥¥_libs¥¥hashtable_class_helper.pxi:7089, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'year'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[229], line 2
      1 # Create year dummies after dropping first obs per group (already done in df_diff)
----> 2 df_diff['year'] = df['year']  # reattach 'year' column since it was removed in .diff()
      3 year_dummies = pd.get_dummies(df_diff['year'], prefix='year', drop_first=True)
      4 X_diff = pd.concat([X_diff, year_dummies], axis=1)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥pandas¥core¥frame.py:4102, in DataFrame.__getitem__(self, key)
   4100 if self.columns.nlevels > 1:
   4101     return self._getitem_multilevel(key)
-> 4102 indexer = self.columns.get_loc(key)
   4103 if is_integer(indexer):
   4104     indexer = [indexer]

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥pandas¥core¥indexes¥base.py:3812, in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 'year'
```

```
import pandas as pd
from linearmodels.panel import ArellanoBond

# Assume your DataFrame is called df and already cleaned and indexed
# (i.e., df.index = ["city", "year"])
# If not yet set:
#df = df.set_index(['city', 'year'])
# "18歳未満人員_(人)"
# Drop missing values in relevant columns (important for Arellano-Bond)
df_model = df[['仕送り金', '18歳未満人員_(人)',"農林漁家世帯の割合(%)" ,'エンゲル係数_(%)']].dropna()

# Define dependent variable
dep_var = '仕送り金'

# Define exogenous regressors (these can be instrumented internally via lag structure)
exog_vars = ['18歳未満人員_(人)',"農林漁家世帯の割合(%)" ,'エンゲル係数_(%)']

# Arellano-Bond estimation
model = ArellanoBond(
    df_model,
    dep_var=dep_var,
    exog=exog_vars,
    lags=1,  # First lag of the dependent variable as regressor
    instruments=None,  # Use default instrument strategy
    max_instr=3,  # Max lag of instruments used
    demean=True  # Time-demeaning
)

results = model.fit(cov_type='robust')
print(results.summary)
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[198], line 2
      1 import pandas as pd
----> 2 from linearmodels.panel import ArellanoBond
      4 # Assume your DataFrame is called df and already cleaned and indexed
      5 # (i.e., df.index = ["city", "year"])
      6 # If not yet set:
      7 #df = df.set_index(['city', 'year'])
      8 # "18歳未満人員_(人)"
      9 # Drop missing values in relevant columns (important for Arellano-Bond)
     10 df_model = df[['仕送り金', '18歳未満人員_(人)',"農林漁家世帯の割合(%)" ,'エンゲル係数_(%)']].dropna()

ImportError: cannot import name 'ArellanoBond' from 'linearmodels.panel' (c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-
```

```
from sklearn.linear_model import LassoCV
from sklearn.preprocessing import StandardScaler

# Drop target and some known non-features
drop_cols = ['仕送り金','贈与金', '(再掲)教育関係費', '(再掲)移転支出(贈与金+仕送り金)']
features = df_diff.drop(columns=[col for col in drop_cols if col in df_diff.columns], errors='ignore')
features = features.select_dtypes(include='number')

# Drop rows where target or any feature is NaN
df_lasso = pd.concat([df_diff['仕送り金'], features], axis=1).dropna()

# Separate clean X and y
X = df_lasso.drop(columns='仕送り金')
y = df_lasso['仕送り金']

# Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit LassoCV
alphas = np.logspace(-1.7,1,50)
lasso = LassoCV(alphas=alphas, cv=5, random_state=0).fit(X_scaled, y)

# Get selected features
selected_features = X.columns[lasso.coef_ != 0].tolist()
rejected_features = X.columns[lasso.coef_ == 0].tolist()

print("✅ Selected features:")
print(selected_features)

print("¥n❌ Rejected features:")
print(rejected_features)
print("Chosen alpha:", lasso.alpha_)
```

```
✅ Selected features:
['他の穀類', '生鮮野菜', '果物加工品', '主食的調理食品', '他の飲料', '和服', '男子用洋服', '婦人用シャツ・セーター類', '子供用下着類', '保健医療
```

```
❌ Rejected features:
['Unnamed:_0', '世帯数分布(抽出率調整)', '集計世帯数', '世帯人員_(人)', '18歳未満人員_(人)', '65歳以上人員_(人)', 'うち無職者人員_(人)', '有業人
Chosen alpha: 0.0199526231496888
```

Features which seem to be significant "'消費支出', '住居', '自動車等購入', 'その他の消費支出', '諸雑費"

✅ Selected features: ['その他の消費支出', '諸雑費', 'その他の諸雑費', 'こづかい(使途不明)', '交際費']

❌ Rejected features: ['Unnamed:*0', '世帯数分布(抽出率調整)', '集計世帯数', '世帯人員*(人)', '18歳未満人員*(人)', '65歳以上人員*(人)', 'うち無職者人員*(人)', '有業人員*(人)', '世帯主の配偶者のうち女の有業率(%)', '世帯主の年齢*(歳)', '持家率*(%)', '平均畳数', '家賃・地代を支払っている世帯の割合(%)', '平均畳数.1', '農林漁家世帯の割合(%)', '消費支出', '食料', '穀類', '米', 'パン', 'めん類', '他の穀類', '魚介類', '生鮮魚介', '塩干魚介', '魚肉練製品', '他の魚介加工品', '肉類', '生鮮肉', '加工肉', '乳卵類', '牛乳', '乳製品', '卵', '野菜・海藻', '生鮮野菜', '乾物・海藻', '大豆加工品', '他の野菜・海藻加工品', '果物', '生鮮果物', '果物加工品', '油脂・調味料', '油脂', '調味料', '菓子類', '調理食品', '主食的調理食品', '他の調理食品', '飲料', '茶類', 'コーヒー・ココア', '他の飲料', '酒類', '外食', '一般外食', '学校給食', '住居', '家賃地代', '設備修繕・維持', '設備材料', '工事その他のサービス', '光熱・水道', '電気代', 'ガス代', '他の光熱', '上下水道料', '家具・家事用品', '家庭用耐久財', '家事用耐久財', '冷暖房用器具', '一般家具', '室内装備・装飾品', '寝具類', '家事雑貨', '家事用消耗品', '家事サービス', '被服及び履物', '和服', '洋服', '男子用洋服', '婦人用洋服', '子供用洋服', 'シャツ・セーター類', '男子用シャツ・セーター類', '婦人用シャツ・セーター類', '子供用シャツ・セーター類', '下着類', '男子用下着類', '婦人用下着類', '子供用下着類', '生地・糸類', '他の被服', '履物類', '被服関連サービス', '保健医療', '医薬品', '健康保持用摂取品', '保健医療用品・器具', '保健医療サービス', '交通・通信', '交通', '自動車等関係費', '自動車等購入', '自転車購入', '自動車等維持', '通信', '教育', '授業料等', '教科書・学習参考教材', '補習教育', '教養娯楽', '教養娯楽用耐久財', '教養娯楽用品', '書籍・他の印刷物', '教養娯楽サービス', '宿泊料', 'パック旅行費', '月謝類', '他の教養娯楽サービス', '理美容サービス', '理美容用品', '身の回り用品', 'たばこ', '食料.1', '家具・家事用品.1', '被服及び履物.1', '教養娯楽.1', '他の物品サービス', '他の交際費', '(再掲)_教養娯楽関係費', '(再掲)_経常消費支出', '(再掲)_情報通信関係費', '(再掲)_消費支出(除く住居等)_1)', '現物総額', '食料.2', '自家産物', '住

居.1', '光熱・水道.1', '家具・家事用品.2', '被服及び履物.2', '保健医療.1', '交通・通信.1', '教育.1', '教養娯楽.2', 'その他の消費支出.1', 'エンゲル係数(%)', '調整集計世帯数', 'const'] Chosen alpha: 0.05011872336272722

```
import pandas as pd
import statsmodels.api as sm

# Sort by entity and time for differencing
df = df.sort_values(['city', 'year'])

# First difference the variables (by group)
df_diff = df.groupby('city').diff().dropna()
df_diff['仕送り金'] = pd.to_numeric(df_diff['仕送り金'], errors='coerce')
cols = regressors
df_diff[cols] = df_diff[cols].apply(pd.to_numeric, errors='coerce')

#df_diff['農林漁家世帯の割合(%)'] = pd.to_numeric(df_diff['消費支出', '住居', '自動車等購入', 'その他の消費支出', '諸雑費"], errors='coerce')
#df_diff['エンゲル係数_(%)'] = pd.to_numeric(df_diff['エンゲル係数_(%)'], errors='coerce')

# You lose the first observation for each city due to differencing
# Make sure to align the dependent and independent variables after differencing
y_diff = df_diff['仕送り金']
X_diff = df_diff[regressors]  # include all regressors

# Add constant
X_diff = sm.add_constant(X_diff)

# Run OLS on first differences
model_fd = sm.OLS(y_diff, X_diff).fit()
print(model_fd.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   仕送り金   R-squared:                       0.067
Model:                            OLS   Adj. R-squared:                  0.048
Method:                 Least Squares   F-statistic:                     3.675
Date:                Sun, 06 Jul 2025   Prob (F-statistic):           0.000185
Time:                        13:13:04   Log-Likelihood:                 885.49
No. Observations:                 474   AIC:                            -1751.
Df Residuals:                     464   BIC:                            -1709.
Df Model:                           9
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                          0.0009      0.002      0.485      0.628      -0.003       0.005
18歳未満人員_(人)                  0.0097      0.016      0.597      0.551      -0.022       0.041
65歳以上人員_(人)                 -0.0246      0.026     -0.943      0.346      -0.076       0.027
有業人員_(人)                     0.0577      0.036      1.602      0.110      -0.013       0.128
世帯主の配偶者のうち女の有業率(%)       -0.0266      0.012     -2.142      0.033      -0.051      -0.002
世帯主の年齢_(歳)                  0.4947      0.251      1.975      0.049       0.002       0.987
農林漁家世帯の割合(%)                0.0003      0.001      0.200      0.842      -0.003       0.003
エンゲル係数_(%)                  -0.1117      0.022     -4.988      0.000      -0.156      -0.068
光熱・水道                        -0.1126      0.661     -0.170      0.865      -1.411       1.186
家賃・地代を支払っている世帯の割合(%)     -0.0226      0.016     -1.395      0.164      -0.054       0.009
==============================================================================
Omnibus:                       57.165   Durbin-Watson:                   2.631
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              301.031
Skew:                          -0.339   Prob(JB):                     4.28e-66
Kurtosis:                       6.845   Cond. No.                         469.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
df2 =df.copy()
df2 = df2.reset_index()
df2
```

	city	year	Unnamed: 0	世帯数分布(抽出率調整)	集計世帯数	世帯人員_(人)	18歳未満人員_(人)	65歳以上人員_(人)	うち無職者人員_(人)	有業人員_(人)	...	家具・家事用品.2	被服及び履物.2	保健医療.1	交通・通信.1	教育.1	教養娯楽.2
0	Akita City	2000	28.0	25	96	3.21	0.83	0.44	0.38	1.26	...	681.0	819.0	97.0	176.0	0.0	586.0
1	Akita City	2001	28.0	25	96	3.1	0.77	0.45	0.38	1.26	...	715.0	742.0	60.0	167.0	0.0	561.0
2	Akita City	2003	28.0	26	96	3.03	0.62	0.6	0.53	1.21	...	372.0	442.0	93.0	152.0	0.0	624.0
3	Akita City	2004	28.0	26	96	3.26	0.7	0.6	0.52	1.31	...	535.0	431.0	64.0	208.0	0.0	676.0
4	Akita City	2005	28.0	26	95	3.17	0.77	0.54	0.47	1.22	...	341.0	459.0	66.0	82.0	0.0	711.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1697	NaN	2016	18.0	2882	1360	2.96	0.53	0.81	0.65	1.3	...	140.0	183.0	28.0	267.0	0.0	378.0
1698	NaN	2017	20.0	1437	963	2.95	0.52	0.9	0.73	1.19	...	109.0	147.0	28.0	139.0	0.0	278.0
1699	NaN	2017	18.0	2889	1352	2.95	0.51	0.84	0.67	1.29	...	135.0	133.0	14.0	272.0	0.0	492.0
1700	NaN	2022	5.0	2460	957	2.96	0.55	0.9	0.7	1.36	...	NaN	NaN	NaN	NaN	NaN	NaN
1701	NaN	2024	NaN	10101	10101	10101	10101	10101	10101	10101		10101.0	10101.0	10101.0	10101.0	10101.0	10101.0

```
import pandas as pd
from linearmodels.panel import PanelOLS

# Sort and set MultiIndex (required for PanelOLS)
df2 =df.copy()
df2 = df2.reset_index()
df2 = df2.sort_values(['city', 'year'])
df2 = df2.set_index(['city', 'year'])

# Convert variables to numeric
df2['仕送り金'] = pd.to_numeric(df2['仕送り金'], errors='coerce')
#, 'その他の消費支出', '諸雑費', 'エンゲル係数_(%)'
cols = regressors
df2[cols] = df2[cols].apply(pd.to_numeric, errors='coerce')

# Drop rows with NA in any of the used columns
df2 = df2.dropna(subset=['仕送り金'] + cols)

# Define dependent and independent variables
y = df2['仕送り金']
X = df2[cols]

# Add a constant (PanelOLS drops it if using entity/time effects, but safe to include)
X = sm.add_constant(X)

# Run fixed effects regression with both entity (city) and time effects
model = PanelOLS(y, X, entity_effects=True, time_effects=True)
results = model.fit()

# Print full regression output
print(results.summary)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[231], line 28
     26 # Run fixed effects regression with both entity (city) and time effects
     27 model = PanelOLS(y, X, entity_effects=True, time_effects=True)
---> 28 results = model.fit()
     30 # Print full regression output
     31 print(results.summary)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥linearmodels¥panel¥model.py:1918, in PanelOLS.fit(self, use_lsdv,
use_lsmr, low_memory, cov_type, debiased, auto_df, count_effects, **cov_config)
   1914 low_memory = (
   1915     self._choose_twoway_algo() if low_memory is None else low_memory
   1916 )
   1917 if not weighted:
-> 1918     y, x, ybar = self._fast_path(low_memory=low_memory)
   1919     y_effects = np.array([0.0])
   1920     x_effects = np.zeros(x.shape)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥linearmodels¥panel¥model.py:1664, in PanelOLS._fast_path(self,
low_memory)
   1662     x = cast(PanelData, x.general_demean(groups))
   1663 elif self.entity_effects and self.time_effects:
-> 1664     y = cast(PanelData, y.demean("both", low_memory=low_memory))
   1665     x = cast(PanelData, x.demean("both", low_memory=low_memory))
   1666 elif self.entity_effects:

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥linearmodels¥panel¥data.py:584, in PanelData.demean(self, group,
weights, return_panel, low_memory)
    582 if group == "both":
    583     if not low_memory:
--> 584         return self._demean_both(weights)
    585     else:
    586         return self._demean_both_low_mem(weights)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥linearmodels¥panel¥data.py:416, in PanelData._demean_both(self,
weights)
    414 d_arr = d_pd.values2d
    415 e_arr = e.values2d
--> 416 resid_arr = e_arr - d_arr @ lstsq(d_arr, e_arr, rcond=None)[0]
    417 resid = DataFrame(
    418     resid_arr, index=self._frame.index, columns=self._frame.columns
    419 )
    421 return PanelData(resid)

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥numpy¥linalg¥linalg.py:2326, in lstsq(a, b, rcond)
   2323 if n_rhs == 0:
   2324     # lapack can't handle n_rhs = 0 - so allocate the array one larger in that axis
   2325     b = zeros(b.shape[:-2] + (m, n_rhs + 1), dtype=b.dtype)
-> 2326 x, resids, rank, s = gufunc(a, b, rcond, signature=signature, extobj=extobj)
   2327 if m == 0:
   2328     x[...] = 0

File c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥numpy¥linalg¥linalg.py:124, in _raise_linalgerror_lstsq(err, flag)
    123 def _raise_linalgerror_lstsq(err, flag):
```

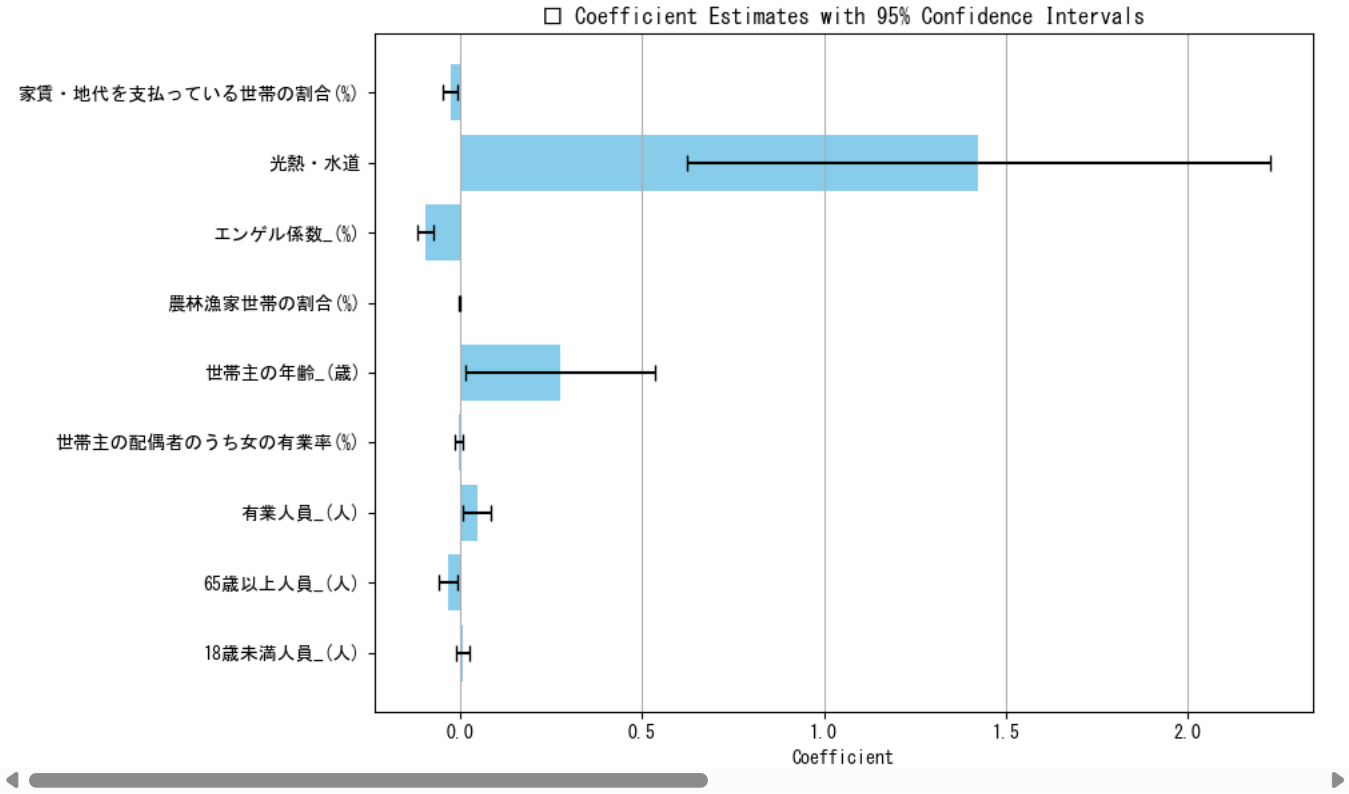
```
import pandas as pd
from linearmodels.panel import PanelOLS

# Copy and prep data
df2 = df.copy().reset_index()
df2 = df2.sort_values(['city', 'year'])

# Convert city and year to categorical and then to codes
df2['city_code'] = pd.Categorical(df2['city']).codes
df2['year_code'] = pd.Categorical(df2['year']).codes

# Set MultiIndex on the integer codes (zero-based)
df2 = df2.set_index(['city_code', 'year_code'])

# Convert to numeric
df2['仕送り金'] = pd.to_numeric(df2['仕送り金'], errors='coerce')
cols = regressors
df2[cols] = df2[cols].apply(pd.to_numeric, errors='coerce')

# Drop NA rows
df2 = df2.dropna(subset=['仕送り金'] + cols)

# Define dependent and independent variables
y = df2['仕送り金']
X = df2[cols]  # No constant added!

# Run PanelOLS with fixed effects
```

```
model = PanelOLS(y, X, entity_effects=True, time_effects=True)
results = model.fit()

print(results.summary)
```

```
                          PanelOLS Estimation Summary
================================================================================
Dep. Variable:                  仕送り金   R-squared:                        0.1028
Estimator:                   PanelOLS   R-squared (Between):              0.8506
No. Observations:                1240   R-squared (Within):              -35.415
Date:                Sun, Jul 06 2025   R-squared (Overall):              0.4206
Time:                        13:14:26   Log-likelihood                   2825.3
Cov. Estimator:            Unadjusted
                                        F-statistic:                      14.521
Entities:                          72   P-value                           0.0000
Avg Obs:                       17.222   Distribution:                  F(9,1141)
Min Obs:                       1.0000
Max Obs:                       36.000   F-statistic (robust):             14.521
                                        P-value                           0.0000
Time periods:                      19   Distribution:                  F(9,1141)
Avg Obs:                       65.263
Min Obs:                       1.0000
Max Obs:                       70.000

                                Parameter Estimates
==============================================================================================
                      Parameter  Std. Err.     T-stat    P-value    Lower CI    Upper CI
----------------------------------------------------------------------------------------------
18歳未満人員_(人)               0.0072      0.0096      0.7465      0.4555      -0.0116       0.0260
65歳以上人員_(人)              -0.0316      0.0130     -2.4265      0.0154      -0.0572      -0.0061
有業人員_(人)                  0.0464      0.0196      2.3644      0.0182       0.0079       0.0849
世帯主の配偶者のうち女の有業率(%)       -0.0042      0.0059     -0.7047      0.4812      -0.0158       0.0074
世帯主の年齢_(歳)               0.2753      0.1323      2.0811      0.0376       0.0158       0.5348
農林漁家世帯の割合(%)             -0.0006      0.0006     -0.9516      0.3415      -0.0018       0.0006
エンゲル係数_(%)               -0.0966      0.0112     -8.6295      0.0000      -0.1186      -0.0747
光熱・水道                     1.4246      0.4092      3.4815      0.0005       0.6218       2.2275
家賃・地代を支払っている世帯の割合(%)     -0.0270      0.0099     -2.7229      0.0066      -0.0465      -0.0076
==============================================================================================

F-test for Poolability: 15.568
P-value: 0.0000
Distribution: F(89,1141)

Included effects: Entity, Time
```

```
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams['font.family'] = 'MS Gothic'  # Japanese font

coefs = results.params.dropna()
errors = results.std_errors[coefs.index].dropna()

y_pos = np.arange(len(coefs))

plt.figure(figsize=(10, 6))
plt.barh(y_pos, coefs.values, xerr=1.96 * errors.values, align='center', color='skyblue', ecolor='black', capsize=4)
plt.yticks(y_pos, coefs.index)
plt.xlabel('Coefficient')
plt.title('📊 Coefficient Estimates with 95% Confidence Intervals')
plt.grid(axis='x')
plt.tight_layout()
plt.show()
```

```
C:¥Users¥strat¥AppData¥Local¥Temp¥ipykernel_12948¥2611541260.py:17: UserWarning: Glyph 128202 (¥N{BAR CHART}) missing from font(s) MS Gothic.
  plt.tight_layout()
c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥IPython¥core¥pylabtools.py:170: UserWarning: Glyph 128202 (¥N{BAR CHART}
  fig.canvas.print_figure(bytes_io, **kw)
```




```
print(type(results.params))
print(results.params.shape)
print(results.params.head())
print(type(results.std_errors))
print(results.std_errors.shape)
print(results.std_errors.head())
```

```
<class 'pandas.core.series.Series'>
(9,)
18歳未満人員_(人)             0.007153
65歳以上人員_(人)            -0.031621
有業人員_(人)                0.046419
世帯主の配偶者のうち女の有業率(%)    -0.004170
世帯主の年齢_(歳)              0.275254
Name: parameter, dtype: float64
<class 'pandas.core.series.Series'>
(9,)
18歳未満人員_(人)             0.009582
65歳以上人員_(人)             0.013031
有業人員_(人)                0.019633
世帯主の配偶者のうち女の有業率(%)     0.005918
世帯主の年齢_(歳)              0.132261
Name: std_error, dtype: float64
```

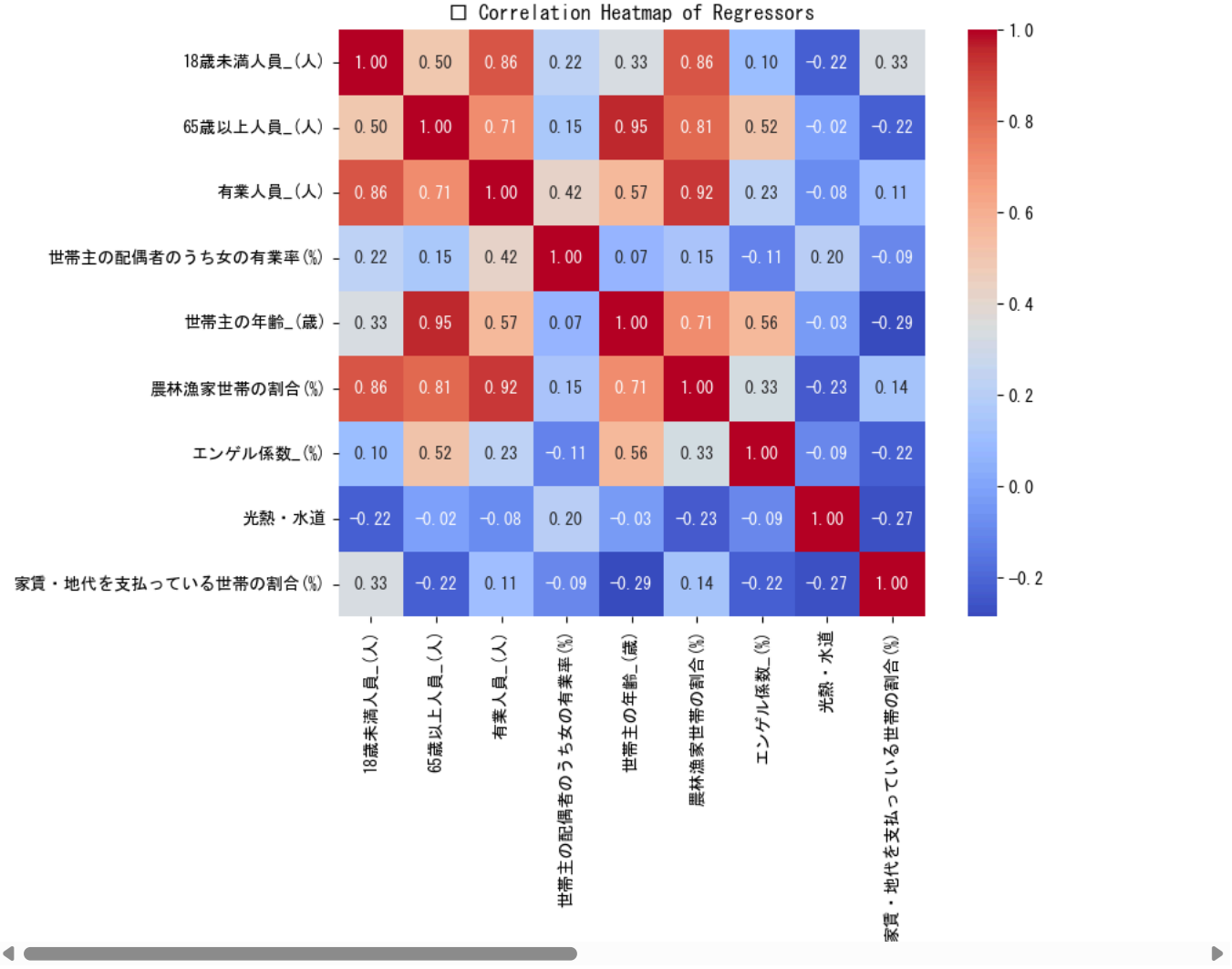
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

X_vif = add_constant(X)
vif_data = pd.DataFrame({
    "Variable": X_vif.columns,
    "VIF": [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.shape[1])]
})
print("📈 Variance Inflation Factors:")
print(vif_data)
```

```
📈 Variance Inflation Factors:
                 Variable           VIF
0                   const  19288.968980
1            18歳未満人員_(人)      9.737869
2            65歳以上人員_(人)     21.515949
3               有業人員_(人)     22.510995
4    世帯主の配偶者のうち女の有業率(%)      2.752929
5              世帯主の年齢_(歳)     21.123244
6            農林漁家世帯の割合(%)     44.080006
7              エンゲル係数_(%)      1.609743
```

```
8                光熱・水道       1.478535
9  家賃・地代を支払っている世帯の割合(%)       1.608123
```

```
plt.figure(figsize=(10, 8))
corr_matrix = X.corr()
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", square=True)
plt.title("🔍 Correlation Heatmap of Regressors")
plt.tight_layout()
plt.show()
```

```
C:¥Users¥strat¥AppData¥Local¥Temp¥ipykernel_12948¥3774100250.py:5: UserWarning: Glyph 128269 (¥N{LEFT-POINTING MAGNIFYING GLASS}) missing from fo
  plt.tight_layout()
c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥IPython¥core¥pylabtools.py:170: UserWarning: Glyph 128269 (¥N{LEFT-POIN
  fig.canvas.print_figure(bytes_io, **kw)
```




```
# Flatten if needed
residuals = results.resids.squeeze()
fitted_vals = results.fitted_values.squeeze()

plt.figure(figsize=(8, 6))
sns.scatterplot(x=fitted_vals, y=residuals, alpha=0.6)
plt.axhline(0, color='red', linestyle='--')
plt.title("⚙ Residuals vs. Fitted Values")
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:¥Users¥strat¥AppData¥Local¥Temp¥ipykernel_12948¥901330578.py:12: UserWarning: Glyph 9881 (¥N{GEAR}) missing from font(s) MS Gothic.
  plt.tight_layout()
C:¥Users¥strat¥AppData¥Local¥Temp¥ipykernel_12948¥901330578.py:12: UserWarning: Glyph 65039 (¥N{VARIATION SELECTOR-16}) missing from font(s) MS G
  plt.tight_layout()
c:¥Users¥strat¥AppData¥Local¥Programs¥Python¥Python312¥Lib¥site-packages¥IPython¥core¥pylabtools.py:170: UserWarning: Glyph 9881 (¥N{GEAR}) miss
```