1. Data-preprocessing

A. The role of borrower income on credit risk prediction

The borrower’s income plays a vital role in credit risk prediction. First of all, it is a key factor in assessing a borrower’s ability to repay loans. In general lending principles CAMPARI, borrower income is a part of the "Ability" criterion (University of Technology Sydney, 2025). Banks rely on the borrower’s income to ensure that the borrower has adequate cash flow to meet debt obligations. Secondly, bank lenders can evaluate a borrower’s financial capacity through their income, and its stability over time is essential in determining creditworthiness (University of Technology Sydney, 2025). In evaluating a personal credit, the central task is to ascertain the borrower’s capacity to repay. High, stable income levels reduce the likelihood of default, as they increase the probability of consistent repayment. Conversely, low or unstable income indicates a higher risk of default. Credit scoring models often incorporate income alongside other financial data to estimate the repayment capability (University of Technology Sydney, 2025). Moreover, in commercial banks, the logistic regression model effectively predicts digital loan defaults with income to loan ratio and credit score being critical variables (Barasa et al., 2025).

B. Borrower incomes by state in the US

Import data

The data of Median Household Income by State: 1984 to 2023 is devided into 2 dataset by Current dollars (`current.csv`) and 2023 dollars (`2023.csv`). Firstly, consider the dataset of current dollars.

In [1]:
```
cd "C:\Users\2017\Downloads"
```

```
C:\Users\2017\Downloads
```

In [2]:
```
import pandas as pd

df = pd.read_csv('current.csv')
df.head()
```

Out[2]:

	State	2023	Unnamed: 2	2022	Unnamed: 4	2021	Unnamed: 6	2020 (41)	Unnamed: 8	2019
0	United States	80,610	385	74,580	588	70,780	368	68,010	535	68,700
1	Alabama	60,660	3,993	59,910	1,934	56,930	2,294	54,690	2,563	56,200
2	Alaska	98,190	6,112	89,740	4,842	81,130	3,967	74,750	3,557	78,390
3	Arizona	82,660	2,723	73,450	4,123	70,820	3,394	67,090	3,409	70,670
4	Arkansas	63,250	2,451	53,980	2,376	50,780	1,440	50,780	1,837	54,540

5 rows × 85 columns

In [3]:

```
import matplotlib.pyplot as plt
```

In [4]:

```
year_columns = [col for col in df.columns if 'Unnamed' not in col and col != 'State
standard_error_columns = [col for col in df.columns if 'Unnamed' in col]
```

In [5]:

```
standard_error_columns
```

Out[5]:

```
['Unnamed: 2',
 'Unnamed: 4',
 'Unnamed: 6',
 'Unnamed: 8',
 'Unnamed: 10',
 'Unnamed: 12',
 'Unnamed: 14',
 'Unnamed: 16',
 'Unnamed: 18',
 'Unnamed: 20',
 'Unnamed: 22',
 'Unnamed: 24',
 'Unnamed: 26',
 'Unnamed: 28',
 'Unnamed: 30',
 'Unnamed: 32',
 'Unnamed: 34',
 'Unnamed: 36',
 'Unnamed: 38',
 'Unnamed: 40',
 'Unnamed: 42',
 'Unnamed: 44',
 'Unnamed: 46',
 'Unnamed: 48',
 'Unnamed: 50',
 'Unnamed: 52',
 'Unnamed: 54',
 'Unnamed: 56',
 'Unnamed: 58',
 'Unnamed: 60',
 'Unnamed: 62',
 'Unnamed: 64',
 'Unnamed: 66',
 'Unnamed: 68',
 'Unnamed: 70',
 'Unnamed: 72',
 'Unnamed: 74',
 'Unnamed: 76',
 'Unnamed: 78',
 'Unnamed: 80',
 'Unnamed: 82',
 'Unnamed: 84']
```

In [6]:

```
for i, year in enumerate(year_columns):
    df.rename(columns={standard_error_columns[i]: f"Standard Error {year}"}, inplac
```

In [7]:

```
df.head()
```

Out[7]:

	State	2023	Standard Error 2023	2022	Standard Error 2022	2021	Standard Error 2021	2020 (41)	Standard Error 2020 (41)	2019	...	
0	United States	80,610	385	74,580	588	70,780	368	68,010	535	68,700	...	2
1	Alabama	60,660	3,993	59,910	1,934	56,930	2,294	54,690	2,563	56,200	...	1
2	Alaska	98,190	6,112	89,740	4,842	81,130	3,967	74,750	3,557	78,390	...	3
3	Arizona	82,660	2,723	73,450	4,123	70,820	3,394	67,090	3,409	70,670	...	2
4	Arkansas	63,250	2,451	53,980	2,376	50,780	1,440	50,780	1,837	54,540	...	2

5 rows × 85 columns

In [8]:

```
year_columns = [col for col in df.columns if col != "State" and "Unnamed" not in co

median_income_columns = [col for col in year_columns if "Error" not in col]
standard_error_columns = [col for col in year_columns if "Error" in col]
```

In [9]:

```
standard_error_columns
```

Out[9]:
```
['Standard Error 2023',
 'Standard Error 2022',
 'Standard Error 2021',
 'Standard Error 2020 (41)',
 'Standard Error 2019',
 'Standard Error 2018',
 'Standard Error 2017 (40)',
 'Standard Error 2017',
 'Standard Error 2016',
 'Standard Error 2015',
 'Standard Error 2014',
 'Standard Error 2013 (39)',
 'Standard Error 2013 (38)',
 'Standard Error 2012',
 'Standard Error 2011',
 'Standard Error 2010 (37)',
 'Standard Error 2009 (36)',
 'Standard Error 2008',
 'Standard Error 2007',
 'Standard Error 2006',
 'Standard Error 2005',
 'Standard Error 2004 (revised)',
 'Standard Error 2003',
 'Standard Error 2002',
 'Standard Error 2001',
 'Standard Error 2000 (30)',
 'Standard Error 1999 (29)',
 'Standard Error 1998',
 'Standard Error 1997',
 'Standard Error 1996',
 'Standard Error 1995 (25)',
 'Standard Error 1994 (24)',
 'Standard Error 1993 (23)',
 'Standard Error 1992 (22)',
 'Standard Error 1991',
 'Standard Error 1990',
 'Standard Error 1989',
 'Standard Error 1988',
 'Standard Error 1987 (21)',
 'Standard Error 1986',
 'Standard Error 1985 (20)',
 'Standard Error 1984 (19)']
```

In [10]:
```
data_long = pd.melt(df, id_vars=["State"], value_vars=median_income_columns, var_na

error_long = pd.melt(df, id_vars=["State"], value_vars=standard_error_columns, var_
```

In [11]:
```
error_long.head()
```

Out[11]:

	State	Year	Standard Error
0	United States	Standard Error 2023	385
1	Alabama	Standard Error 2023	3,993
2	Alaska	Standard Error 2023	6,112
3	Arizona	Standard Error 2023	2,723
4	Arkansas	Standard Error 2023	2,451

In [12]:
```
data_long.head()
```

Out[12]:

	State	Year	Median Income
0	United States	2023	80,610
1	Alabama	2023	60,660
2	Alaska	2023	98,190
3	Arizona	2023	82,660
4	Arkansas	2023	63,250

Create a new dataset including 4 columns: State, Year, Median income and Standard Error

In [13]:

```
error_long['Year'] = error_long['Year'].str.replace("Standard Error ", "")

# Merge the median income data (data_long) and standard error data (error_long)
current = pd.merge(data_long, error_long, on=["State", "Year"])

# Display the final merged data
current.head()
```

Out[13]:

	State	Year	Median Income	Standard Error
0	United States	2023	80,610	385
1	Alabama	2023	60,660	3,993
2	Alaska	2023	98,190	6,112
3	Arizona	2023	82,660	2,723
4	Arkansas	2023	63,250	2,451

In [14]:

```
current = current[(current['State'] != 'United States')]
```

In [15]:

```
current
```

Out[15]:

	State	Year	Median Income	Standard Error
1	Alabama	2023	60,660	3,993
2	Alaska	2023	98,190	6,112
3	Arizona	2023	82,660	2,723
4	Arkansas	2023	63,250	2,451
5	California	2023	89,870	1,840
...	...	...	...	...
2179	Virginia	1984 (19)	26,530	874
2180	Washington	1984 (19)	25,020	823
2181	West Virginia	1984 (19)	16,840	608
2182	Wisconsin	1984 (19)	20,740	821
2183	Wyoming	1984 (19)	23,820	731

2142 rows × 4 columns

In [16]:
```
current.to_csv('data1.csv')
```

In [17]:
```
data = current.copy()
```

Collect time series from 2001 to 2015 of borrower incomes by state

In [18]:
```
data['Year'] = pd.to_numeric(data['Year'], errors='coerce')  # Ensure 'Year' is num
data = data[(data['Year'] >= 2001) & (data['Year'] <= 2015)]

# Select relevant columns (State, Year, Median Income)
data = data[['State', 'Year', 'Median Income']]

# Remove commas and convert 'Median Income' to numeric
data['Median Income'] = data['Median Income'].replace({',': ''}, regex=True)
data['Median Income'] = pd.to_numeric(data['Median Income'], errors='coerce')
```

In [19]:
```
data['Year'] = data['Year'].astype(int)
```

In [20]:
```
data
```

Out[20]:

	State	Year	Median Income
469	Alabama	2015	44510
470	Alaska	2015	75110
471	Arizona	2015	52250
472	Arkansas	2015	42800
473	California	2015	63640
...	...	...	...
1295	Virginia	2001	50240
1296	Washington	2001	42490
1297	West Virginia	2001	29670
1298	Wisconsin	2001	45350
1299	Wyoming	2001	39720

561 rows × 3 columns

Describe the data

In [21]:
```
data.describe()
```

Out[21]:

	Year	Median Income
count	561.000000	561.000000
mean	2007.636364	48979.055258
std	4.601983	8955.712762
min	2001.000000	29360.000000
25%	2003.000000	42440.000000
50%	2007.000000	47920.000000
75%	2012.000000	54780.000000
max	2015.000000	76170.000000

In [68]:

```
unique_years = data['Year'].unique()
print(unique_years)
```

```
[2015 2014 2012 2011 2008 2007 2006 2005 2003 2002 2001]
```

In [23]:

```
income = data.pivot(index='Year', columns='State', values='Median Income')
```

In [24]:

```
income
```

Out[24]:

State	Alabama	Alaska	Arizona	Arkansas	California	Colorado	Connecticut	Delaware	Distric o Columbi
Year									
2001	35160	57360	42700	33340	47260	49400	53350	49600	4117
2002	37600	52770	39730	32390	47440	48290	53390	49650	3907
2003	37260	51840	41170	32000	49300	49940	54970	49020	4504
2005	37150	55890	45250	36660	51760	50450	56840	51240	4499
2006	37950	56420	46660	37060	55320	55700	62400	52440	4848
2007	42210	62990	47220	40800	55730	61140	64140	54590	5078
2008	44480	63990	46910	39590	57010	60940	64680	50700	5559
2011	42590	57430	48620	41300	53370	58630	65420	54660	5525
2012	43460	63650	47040	39020	57020	57260	64250	48970	6525
2014	42280	67630	49250	44920	60490	60940	70160	57520	6828
2015	44510	75110	52250	42800	63640	66600	72890	57760	7007

11 rows × 51 columns

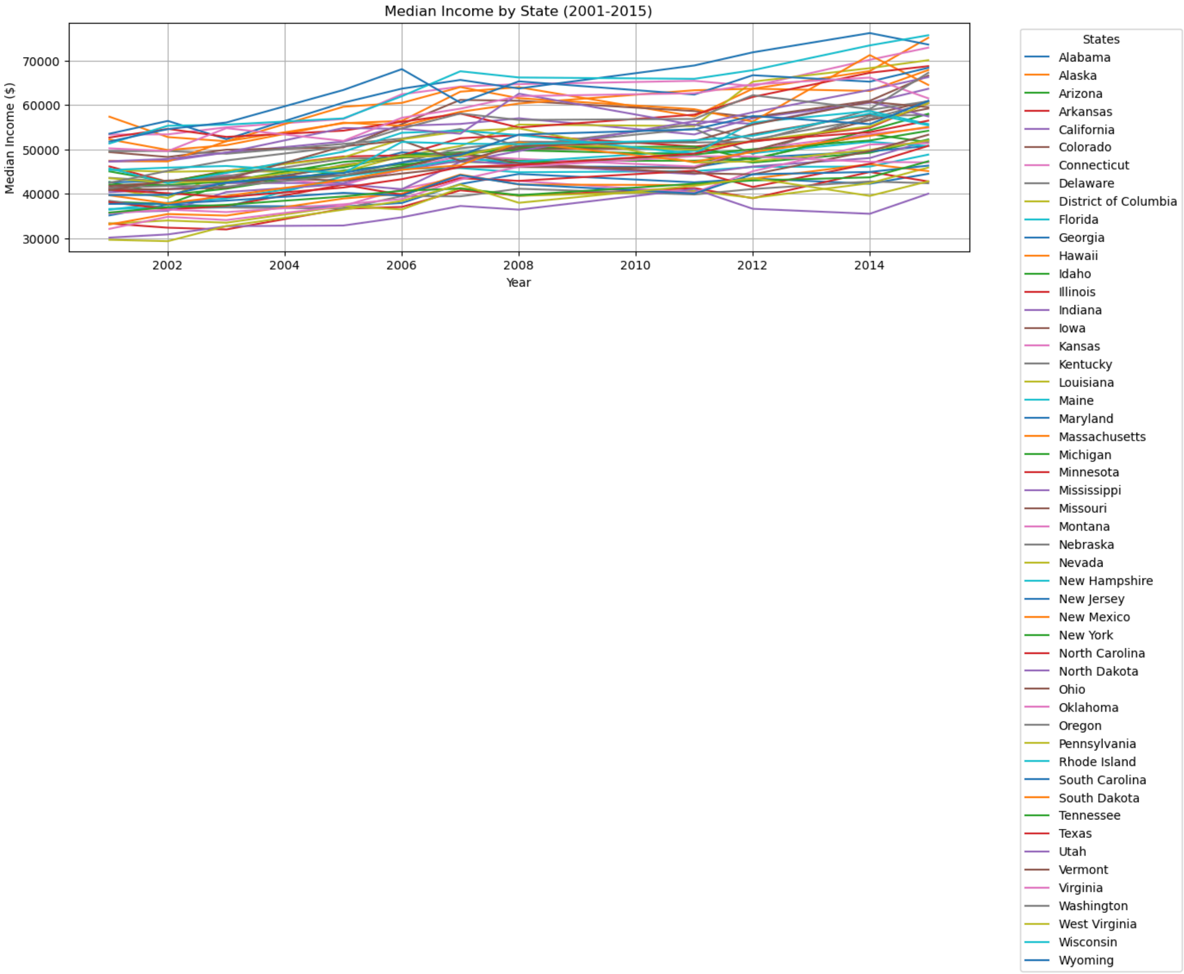
Plot time series by states

In [69]:

```
plt.figure(figsize=(12, 8))
income.plot(title='Median Income by State (2001-2015)', figsize=(14, 8))
plt.xlabel('Year')
plt.ylabel('Median Income ($)')
```

```python
plt.legend(title='States', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

<Figure size 1200x800 with 0 Axes>



In [26]:
```python
income_2015 = income.loc[2015]
```

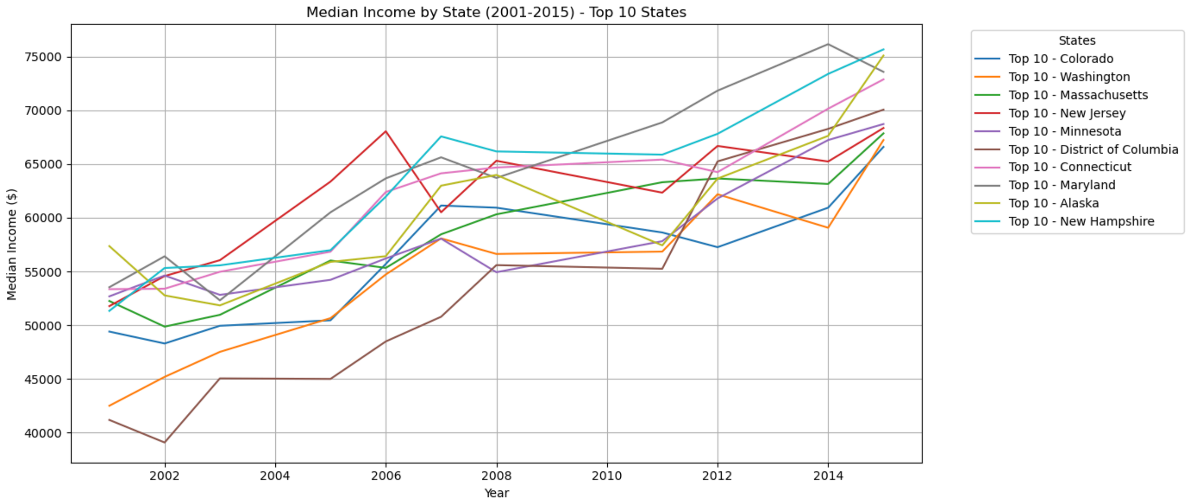
In [27]:
```python
sorted_states_2015 = income_2015.sort_values()
```

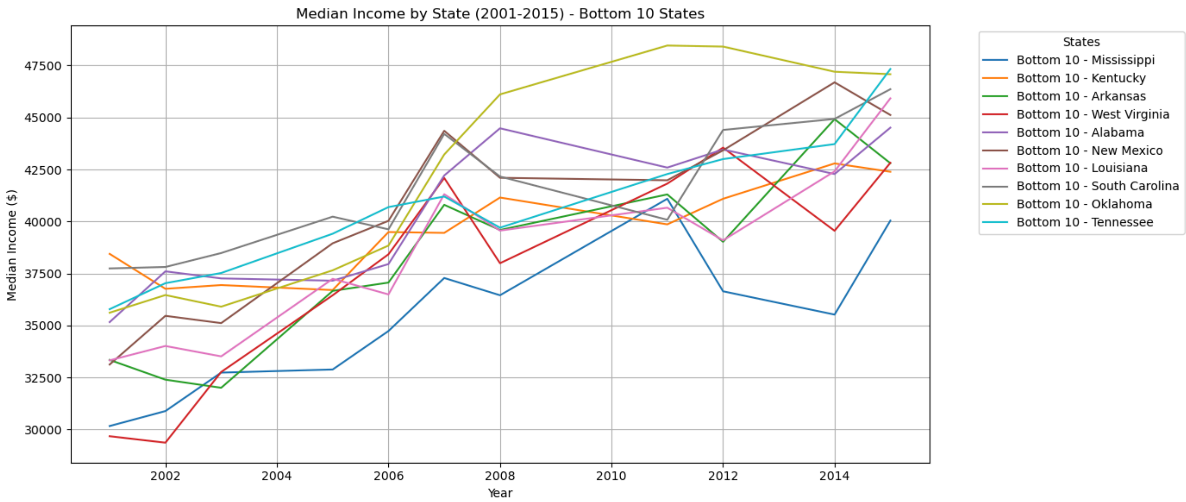
In [28]:
```python
top_10_states = sorted_states_2015.tail(10)
bottom_10_states = sorted_states_2015.head(10)
```

In [29]:
```python
colors = plt.cm.get_cmap('tab10', 10)

plt.figure(figsize=(14, 6))
for idx, state in enumerate(top_10_states.index):
    plt.plot(income.index, income[state], label=f'Top 10 - {state}', color=colors(i

plt.title('Median Income by State (2001-2015) - Top 10 States')
plt.xlabel('Year')
plt.ylabel('Median Income ($)')
plt.legend(title='States', loc='upper left', bbox_to_anchor=(1.05, 1))
plt.grid(True)
plt.tight_layout()
plt.show()


plt.figure(figsize=(14, 6))
for idx, state in enumerate(bottom_10_states.index):
    plt.plot(income.index, income[state], label=f'Bottom 10 - {state}', color=color
```

```
plt.title('Median Income by State (2001-2015) - Bottom 10 States')
plt.xlabel('Year')
plt.ylabel('Median Income ($)')
plt.legend(title='States', loc='upper left', bbox_to_anchor=(1.05, 1))
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\2017\AppData\Local\Temp\ipykernel_20252\3003657566.py:1: MatplotlibDeprec
ationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be r
emoved two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotli
b.colormaps.get_cmap(obj)`` instead.
  colors = plt.cm.get_cmap('tab10', 10)
```





When plotting data for all states, the graph becomes overcrowded and difficult to interpret, so plotting the **Top 10** and **Bottom 10** states simplifies the data and allows us to focus on the most meaningful trends.

Interpret outputs

Overall, there is a general upward movement in median income over the years. However, the rate of increase is different from state to state. In top 10 states, states such as **Colorado**, **Washington**, and **Massachusetts** show strong upward trajectories over the period, with median incomes increasing steadily over the years. This suggests economic growth in these states and positive impacts from some factors such as employment opportunities, stronger industries, etc. may push wages up. Moreover, these states could have benefited from policies or economic conditions, which also suggests a well-educated, high-skill labor force with higher wages.

On the other hand, for bottom 10 states, they show sluggish growth. This indicates slower economic growth or stagnation, which could be due to factors such as weaker industries, lower educational attainment or economic challenges. **Mississippi**, in particular, stands out with one of the lowest median incomes throughout the period, which may reflect persistent poverty or challenges in the local economy.

The significant gap between the top and bottom states highlights a growing regional income disparity. Economic strategies focusing on industry diversification, education, and higher wages could help narrow the gap between these regions.

Interpolate missing years

Because the data shows a consistent pattern or trend over time, **Linear interpolation** is used first to create an estimate of the missing data based on surrounding trends. Next, **Forward-fill** is used afterward to fill any gaps that still exist, ensuring that data continuity is maintained as well as no missing values persist for those years.

In [98]:
```
missing_years = [2004, 2009, 2010, 2013]
states = data['State'].unique()

# Create the missing data rows
missing_data_list = []

for state in states:
    for year in missing_years:
        missing_data_list.append({'State': state, 'Year': year, 'Median Income': No

# Convert the list to a DataFrame
missing_data = pd.DataFrame(missing_data_list)

# Concatenate the missing data with the existing data
data_with_missing_years = pd.concat([data, missing_data], ignore_index=True)

# Sort the values by 'State' and 'Year' after concatenating
data_with_missing_years = data_with_missing_years.sort_values(by=['State', 'Year'])
```

In [99]:
```
data_with_missing_years
```

Out[99]:

	State	Year	Median Income
510	Alabama	2001	35160
459	Alabama	2002	37600
408	Alabama	2003	37260
561	Alabama	2004	None
357	Alabama	2005	37150
...	...	...	...
203	Wyoming	2011	54510
152	Wyoming	2012	57510
764	Wyoming	2013	None
101	Wyoming	2014	55690
50	Wyoming	2015	60930

765 rows × 3 columns

In [100…

```
# Count the number of None/NaN values in the 'Median Income' column
missing_values_count = data_with_missing_years['Median Income'].isna().sum()

# Display the result
print(f"Number of missing ('None' or NaN) values in 'Median Income': {missing_value
```

```
Number of missing ('None' or NaN) values in 'Median Income': 204
```

In [101…

```
data_with_missing_years
```

Out[101]:

	State	Year	Median Income
510	Alabama	2001	35160
459	Alabama	2002	37600
408	Alabama	2003	37260
561	Alabama	2004	None
357	Alabama	2005	37150
...	...	...	...
203	Wyoming	2011	54510
152	Wyoming	2012	57510
764	Wyoming	2013	None
101	Wyoming	2014	55690
50	Wyoming	2015	60930

765 rows × 3 columns

In [563…

```
# Then interpolate remaining missing values with linear method
data_with_missing_years['Median Income'] = data_with_missing_years.groupby('State',

# Forward-fill missing values (use last known value)
data_with_missing_years['Median Income'] = data_with_missing_years.groupby('State',
```

```
# Display the updated data to verify
data_with_missing_years.head()
```

Out[563]:

	State	Year	Median Income	State_abbr
510	Alabama	2001	35160	AL
459	Alabama	2002	37600	AL
408	Alabama	2003	37260	AL
561	Alabama	2004	37260	AL
357	Alabama	2005	37150	AL

In [35]:
```
income_interpolate = data_with_missing_years.pivot(index='Year', columns='State', v
```

In [36]:
```
data_with_missing_years.to_csv('data1_interpolate.csv')
```

Import 2023 data

In [37]:
```
df2023 = pd.read_csv('2023.csv')
```

In [38]:
```
year_col= [col for col in df2023.columns if 'Unnamed' not in col and col != 'State'
standard_error = [col for col in df2023.columns if 'Unnamed' in col]
```

In [39]:
```
for i, year in enumerate(year_col):
    df2023.rename(columns={standard_error[i]: f"Standard Error {year}"}, inplace=Tr
```

In [40]:
```
df2023.head()
```

Out[40]:

	State	2023	Standard Error 2023	2022	Standard Error 2022	2021	Standard Error 2021	2020 (41)	Standard Error 2020 (41)	2019	...	
0	United States	80,610	385	77,540	612	79,260	412	79,560	626	81,210	...	6
1	Alabama	60,660	3,993	62,290	2,011	63,750	2,569	63,980	2,998	66,430	...	4
2	Alaska	98,190	6,112	93,310	5,034	90,850	4,442	87,440	4,162	92,670	...	7
3	Arizona	82,660	2,723	76,370	4,287	79,310	3,801	78,480	3,988	83,540	...	6
4	Arkansas	63,250	2,451	56,120	2,470	56,870	1,613	59,400	2,149	64,470	...	4

5 rows × 85 columns

In [41]:
```
year_col = [col for col in df2023.columns if col != "State" and "Unnamed" not in co

median_income = [col for col in year_columns if "Error" not in col]
standard_error = [col for col in year_columns if "Error" in col]
```

In [42]:
```
data2 = pd.melt(df2023, id_vars=["State"], value_vars=median_income, var_name="Year

error2 = pd.melt(df2023, id_vars=["State"], value_vars=standard_error, var_name="Ye
```

In [43]:
```
error2['Year'] = error2['Year'].str.replace("Standard Error ", "")

data2023 = pd.merge(data2, error2, on=["State", "Year"])

data2023.head()
```

Out[43]:

	State	Year	Median Income	Standard Error
0	United States	2023	80,610	385
1	Alabama	2023	60,660	3,993
2	Alaska	2023	98,190	6,112
3	Arizona	2023	82,660	2,723
4	Arkansas	2023	63,250	2,451

In [44]:
```
data2023 = data2023[(data2023['State'] != 'United States')]
```

In [45]:
```
data2023.to_csv('data2.csv')
```

In [46]:
```
data2 = data2023.copy()
```

In [47]:
```
data2['Year'] = pd.to_numeric(data2['Year'], errors='coerce')  # Ensure 'Year' is r
data2 = data2[(data2['Year'] >= 2001) & (data2['Year'] <= 2015)]

# Select relevant columns (State, Year, Median Income)
data2 = data2[['State', 'Year', 'Median Income']]

# Remove commas and convert 'Median Income' to numeric
data2['Median Income'] = data2['Median Income'].replace({',': ''}, regex=True)
data2['Median Income'] = pd.to_numeric(data2['Median Income'], errors='coerce')
```

In [48]:
```
data2['Year'] = data2['Year'].astype(int)
```

In [49]:
```
data2.describe()
```

Out[49]:

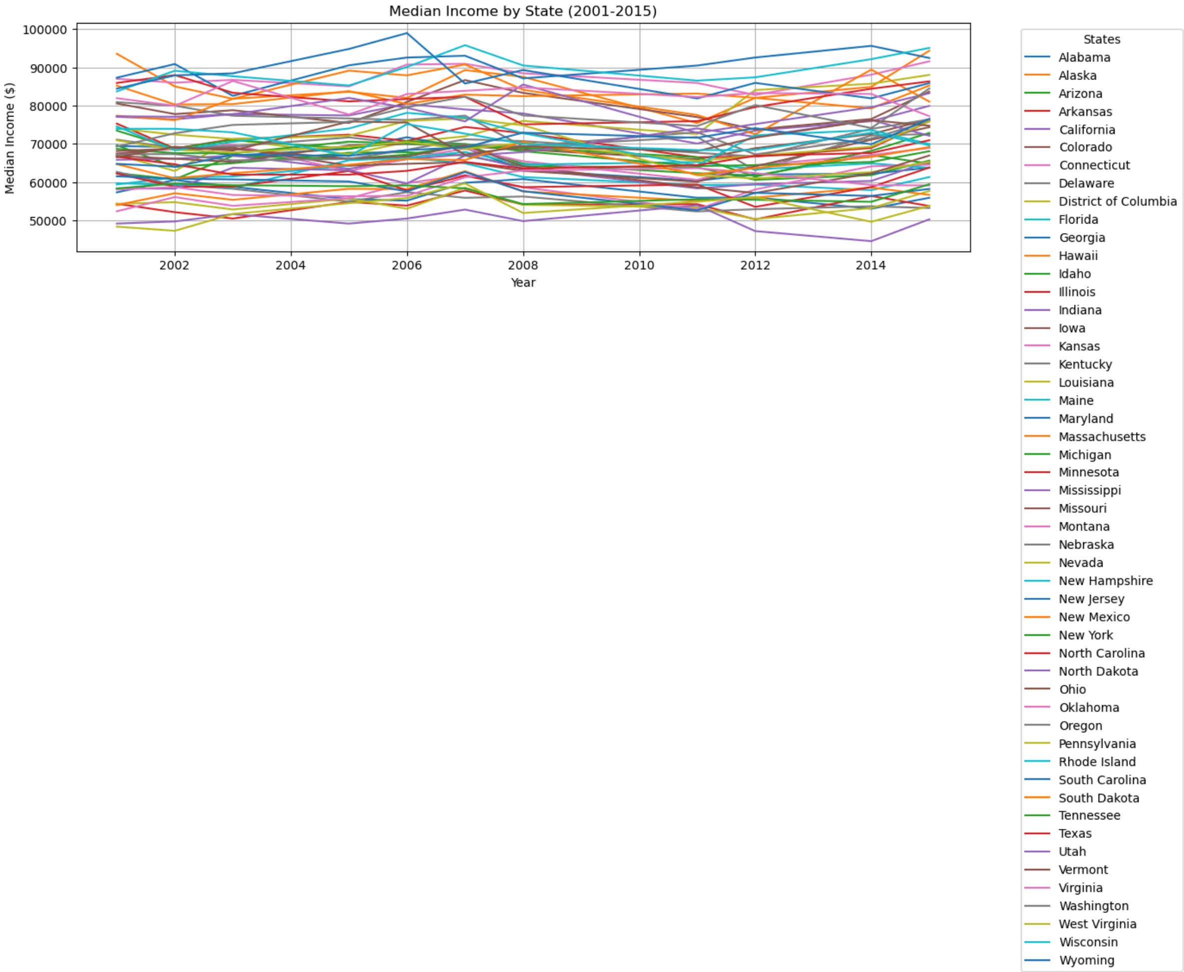
	Year	Median Income
count	561.000000	561.000000
mean	2007.636364	69145.614973
std	4.601983	10733.978477
min	2001.000000	44590.000000
25%	2003.000000	61260.000000
50%	2007.000000	68150.000000
75%	2012.000000	76210.000000
max	2015.000000	98950.000000

In [50]:
```
income2023 = data2.pivot(index='Year', columns='State', values='Median Income')
```

In [51]:
```
plt.figure(figsize=(12, 8))
income2023.plot(title='Median Income by State (2001-2015)', figsize=(14, 8))
plt.xlabel('Year')
plt.ylabel('Median Income ($)')
```

```
plt.legend(title='States', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```



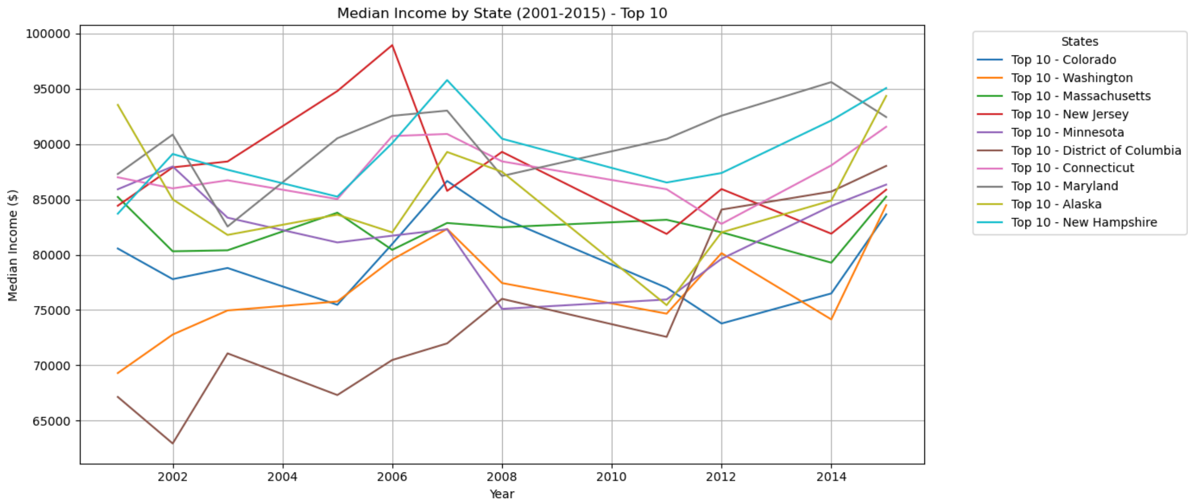
In [52]:
```
income2023_2015 = income2023.loc[2015]
```

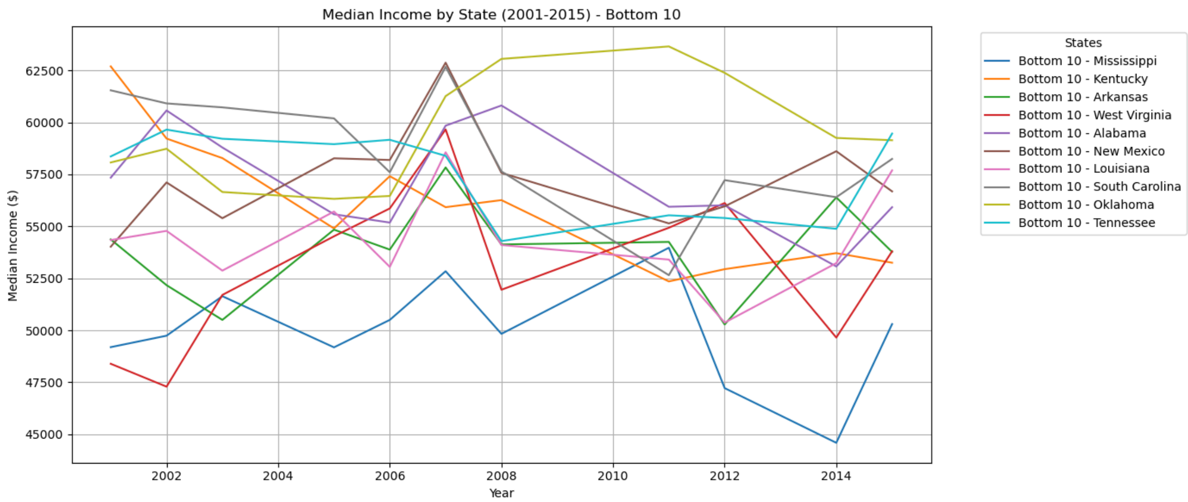
In [53]:
```
sorted_2015 = income2023_2015.sort_values()

top_10 = sorted_2015.tail(10)
bottom_10  = sorted_2015.head(10)
```

In [54]:
```
colors = plt.cm.get_cmap('tab10', 10)

plt.figure(figsize=(14, 6))
for idx, state in enumerate(top_10.index):
    plt.plot(income2023.index, income2023[state], label=f'Top 10 - {state}', color=

plt.title('Median Income by State (2001-2015) - Top 10')
plt.xlabel('Year')
plt.ylabel('Median Income ($)')
plt.legend(title='States', loc='upper left', bbox_to_anchor=(1.05, 1))
plt.grid(True)
plt.tight_layout()
plt.show()


plt.figure(figsize=(14, 6))
for idx, state in enumerate(bottom_10.index):
    plt.plot(income2023.index, income2023[state], label=f'Bottom 10 - {state}', col
```

```
plt.title('Median Income by State (2001-2015) - Bottom 10')
plt.xlabel('Year')
plt.ylabel('Median Income ($)')
plt.legend(title='States', loc='upper left', bbox_to_anchor=(1.05, 1))
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\2017\AppData\Local\Temp\ipykernel_20252\2053074969.py:1: MatplotlibDeprec
ationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be r
emoved two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotli
b.colormaps.get_cmap(obj)`` instead.
  colors = plt.cm.get_cmap('tab10', 10)
```





Interpolate

Similar to current dollars, it can be seen that in 2023 Dollar most states show an upward trend in median income. Some states have seen sharp increases in median income, while others show modest gains or stagnation. However, in the bottom 10, unlike the top states, which show clear and consistent growth, the bottom 10 states have relatively flat or declining lines. This suggests that these states have struggled with economic growth in terms of median income, even after adjusting for inflation. In particular, some states show sharp drops in median income around the 2008 financial crisis (e.g., Mississippi, Arkansas). In general, this reflects growing income inequality across the U.S. By adjusting for inflation to **2023 dollars**, it's clear that the nominal income in bottom states might have increased, but the actual purchasing power has remained stagnant or grown slowly, especially when compared to high-income states.

In [55]:
```
missing_years = [2004, 2009, 2010, 2013]
states = data['State'].unique()

# Create the missing data rows
missing_data_list = []

for state in states:
    for year in missing_years:
        missing_data_list.append({'State': state, 'Year': year, 'Median Income': No

# Convert the list to a DataFrame
missing_data = pd.DataFrame(missing_data_list)

# Concatenate the missing data with the existing data
data2_with_missing_years = pd.concat([data2, missing_data], ignore_index=True)

# Sort the values by 'State' and 'Year' after concatenating
data2_with_missing_years = data2_with_missing_years.sort_values(by=['State', 'Year'
```

In [56]:
```
# Interpolate remaining missing values with linear method
data2_with_missing_years['Median Income'] = data2_with_missing_years.groupby('State

# Forward-fill missing values (use last known value)
data2_with_missing_years['Median Income'] = data2_with_missing_years.groupby('State

# Display the updated data to verify
data2_with_missing_years.head(20)
```

Out[56]:

	State	Year	Median Income
510	Alabama	2001	57340
459	Alabama	2002	60570
408	Alabama	2003	58790
561	Alabama	2004	58790
357	Alabama	2005	55580
306	Alabama	2006	55180
255	Alabama	2007	59840
204	Alabama	2008	60810
562	Alabama	2009	60810
563	Alabama	2010	60810
153	Alabama	2011	55940
102	Alabama	2012	56010
564	Alabama	2013	56010
51	Alabama	2014	53070
0	Alabama	2015	55920
511	Alaska	2001	93550
460	Alaska	2002	85010
409	Alaska	2003	81790
565	Alaska	2004	81790
358	Alaska	2005	83620

In [57]:
```
data2_with_missing_years.to_csv('data2_interpolate.csv')
```

C. Growth rate of income

In [58]:
```
import numpy as np
```

In [59]:
```
data_with_missing_years['Growth Rate'] = data_with_missing_years.groupby('State')['
```

In [60]:
```
data_with_missing_years = data_with_missing_years.dropna(subset=['Growth Rate'])
```

In [61]:
```
growth_rate_summary = data_with_missing_years.groupby('State')['Growth Rate'].descr
```

In [62]:
```
print(growth_rate_summary)
```

State	count	mean	std	min	25% \
Alabama	14.0	1.774977	4.131877	-4.249101	-0.221417
Alaska	14.0	2.151485	6.715166	-10.251602	0.000000
Arizona	14.0	1.530276	4.128221	-6.955504	0.000000
Arkansas	14.0	1.994700	6.666770	-5.520581	-2.438091
California	14.0	2.211040	3.687767	-6.384845	0.000000
Colorado	14.0	2.258878	4.783895	-3.790614	-0.245339
Connecticut	14.0	2.306686	3.444876	-1.788444	0.000000
Delaware	14.0	1.282460	6.484064	-10.409806	0.000000
District of Columbia	14.0	4.056811	6.551097	-5.100802	0.000000
Florida	14.0	2.167287	3.378493	-2.031011	0.000000
Georgia	14.0	1.321702	3.552114	-4.954770	-0.421804
Hawaii	14.0	2.580432	9.235906	-9.421511	-3.002550
Idaho	14.0	2.275012	4.961976	-3.578691	0.000000
Illinois	14.0	2.049098	4.905422	-7.494044	0.000000
Indiana	14.0	1.874645	3.463229	-4.449699	0.000000
Iowa	14.0	2.929521	3.830046	0.000000	0.039888
Kansas	14.0	2.111343	4.259456	-4.974000	0.000000
Kentucky	14.0	0.745217	3.109554	-4.370447	-0.512600
Louisiana	14.0	2.455673	5.586251	-4.236262	-1.102617
Maine	14.0	2.476559	5.202829	-1.837169	0.000000
Maryland	14.0	2.443945	5.692840	-7.268215	0.000000
Massachusetts	14.0	1.952524	3.880892	-4.574163	0.000000
Michigan	14.0	1.369973	2.959585	-5.172031	0.000000
Minnesota	14.0	1.983666	3.794549	-5.390975	0.000000
Mississippi	14.0	2.224808	6.285740	-10.829886	0.000000
Missouri	14.0	2.675822	4.222260	-1.759598	0.000000
Montana	14.0	3.579847	6.122253	-6.107226	0.000000
Nebraska	14.0	2.456976	4.580281	-6.148867	0.000000
Nevada	14.0	1.109961	5.254026	-14.066496	0.000000
New Hampshire	14.0	2.880462	3.922953	-2.071619	0.000000
New Jersey	14.0	2.175660	6.067717	-11.093153	0.000000
New Mexico	14.0	2.344564	4.992873	-5.094680	-0.213777
New York	14.0	2.422395	4.944520	-5.845182	0.000000
North Carolina	14.0	2.258331	6.559622	-8.095554	-0.999770
North Dakota	14.0	3.611938	6.360125	-5.450354	0.000000
Ohio	14.0	1.844349	4.514617	-4.858300	0.000000
Oklahoma	14.0	2.077839	3.777931	-2.499484	-0.077383
Oregon	14.0	2.883256	4.104127	-0.386623	0.000000
Pennsylvania	14.0	2.439290	3.899370	-2.898833	0.000000
Rhode Island	14.0	1.611667	6.512159	-7.907588	-1.342003
South Carolina	14.0	1.582744	4.794788	-4.933586	0.000000
South Dakota	14.0	2.496461	5.233685	-8.488372	0.000000
Tennessee	14.0	2.060559	3.055636	-3.640777	0.000000
Texas	14.0	2.380759	3.066057	-2.191781	0.000000
Utah	14.0	2.630218	6.652265	-11.272785	0.000000
Vermont	14.0	2.898740	6.121342	-8.830319	0.000000
Virginia	14.0	1.560711	4.848753	-7.058646	0.000000
Washington	14.0	3.455165	5.237281	-5.016884	0.000000
West Virginia	14.0	2.879771	6.976794	-9.741031	0.000000
Wisconsin	14.0	1.559401	5.157508	-4.562672	-0.117005
Wyoming	14.0	3.171384	3.895262	-3.164667	0.000000

State	50%	75%	max
Alabama	0.000000	4.494129	11.225296
Alaska	0.474146	7.422611	11.644807
Arizona	0.600086	3.640075	9.910129
Arkansas	0.000000	3.512231	15.120451
California	1.518965	5.153069	6.877898
Colorado	0.000000	5.674333	10.406343
Connecticut	0.993001	3.291231	9.781844
Delaware	0.050403	3.660423	17.459669

```
District of Columbia  1.310779  7.004016  18.099548
Florida               0.399047  3.919564  10.315627
Georgia               0.000000  2.854760   8.223374
Hawaii                0.000000  4.772199  26.590828
Idaho                 0.042176  4.514102  12.327678
Illinois              0.983553  6.037823   9.996358
Indiana               0.841403  4.048843   8.156471
Iowa                  1.212254  4.833272  12.373127
Kansas                1.337949  5.801692   8.374970
Kentucky              0.000000  2.436766   7.602180
Louisiana             0.000000  6.902409  13.209098
Maine                 0.327779  4.676467  18.350849
Maryland              1.539186  5.340690  15.675779
Massachusetts         0.276418  4.513088   9.929356
Michigan              1.165339  3.566867   5.922055
Minnesota             2.433227  3.679517   8.802589
Mississippi           0.229148  5.899830  12.729767
Missouri              1.177997  3.644728  13.806270
Montana               0.293542  9.144681  13.328898
Nebraska              1.299173  5.540837   9.639267
Nevada                0.552910  4.053872   8.442232
New Hampshire         1.494628  6.606491   9.052767
New Jersey            1.356056  6.585532  13.059768
New Mexico            0.000000  6.156464  10.937055
New York              0.924937  2.880464  13.905201
North Carolina        0.000000  7.772805  12.821888
North Dakota          0.572786  7.951761  15.006090
Ohio                  0.781250  3.417039  11.852186
Oklahoma              0.000000  4.446162  11.277034
Oregon                0.884690  5.366860  13.711858
Pennsylvania          0.505882  5.760095   9.461664
Rhode Island          0.000000  5.190228  14.358556
South Carolina        0.092740  2.830051  11.585058
South Dakota          2.993453  5.127681  11.158983
Tennessee             1.498835  3.432156   8.257091
Texas                 2.355269  5.307933   6.326483
Utah                  0.549218  4.988050  16.831683
Vermont               1.436224  6.608772  17.198336
Virginia              0.508146  3.481031  10.376788
Washington            2.772818  6.539577  13.831048
West Virginia         2.068388  9.231241  11.580381
Wisconsin             0.000000  1.562963  15.767077
Wyoming               2.903711  5.424642   9.437833
```

Describe the heterogeneity (differences) in growth rates

The mean growth rate for each state provides an indication of the average income growth over the period. We can see that Iowa has the highest average growth rate of 2.93%, while Kentucky has the lowest average growth rate of 0.75%. In addition, std (Standard Deviation) measures the variability of income growth across years for each state. For example, Hawaii has a high standard deviation of 9.24%, indicating that income growth in Hawaii has been highly volatile, with large fluctuations between years. Some states like Wisconsin have very low variability (1.56%), suggesting that income growth has been relatively steady. Next, percentile distributions (25%, 50%, 75%) show the range of growth rates. 25% shows the lower range of growth rates, indicating where the bottom 25% of values fall. For instance, Georgia, with -0.42%, suggests that a significant portion of the years saw negative or stagnant growth. In contrast, 75% shows the higher range of growth rates. For example, California, with 5.15%, shows that in three-quarters of the years, the growth rate was above this threshold. Lastly, the minimum and maximum values represent the extremes of income growth for each state.

The heterogeneity is obvious when considering top-growth states and struggling states. In the first group (states such as Iowa, South Dakota, Washington, etc.), we can see high growth rates with minimal fluctuations, suggesting that these states may have seen steady economic expansion. On the other hand, states like Kentucky, Georgia, West Virginia show low or even negative growth at times, indicating economic struggles, stagnation or slow recovery after recessions. This heterogeneity reflects a significant economic disparity between regions. In reality, states like California, Texas, and Washington are benefiting from booming economies and are driven by high-wage sectors like technology and finance. Meanwhile, Mississippi, Kentucky and West Virginia are seeing stagnation. It seems that there are some industries that have faced decline. Moreover, states with larger negative growth in the years immediately following the 2008 financial crisis (e.g., Nevada, Arizona, Florida) may have been impacted by the downturn. However, by 2015, many of these states showed recovery. This could indicate that their local economies are beginning to rebound after the crisis. Meanwhile, states like Kentucky, Mississippi show lower income growth rates. This may show slower recovery or other economic issues. In summary, the heterogeneity in income growth rates highlights the disparities in economic performance.

In [103…
```
dcr_data = pd.read_csv('dcr.csv')
```

In [104…
```
dcr_data
```

Out[104]:

	id	time	orig_time	first_time	mat_time	res_time	balance_time	LTV_time	interest_
0	4	25	-2	25	119	NaN	61031.10	33.911009	
1	4	26	-2	25	119	NaN	60882.42	34.007232	
2	4	27	-2	25	119	NaN	60729.80	34.335349	
3	4	28	-2	25	119	NaN	60576.14	34.672545	
4	4	29	-2	25	119	NaN	60424.39	34.951639	
...	...	...	...	...	...	...	...	...	
62173	49972	52	25	52	145	NaN	180673.24	103.306966	
62174	49972	53	25	52	145	NaN	179944.95	95.736862	
62175	49972	54	25	52	145	NaN	179451.81	91.867079	
62176	49972	55	25	52	145	NaN	178952.48	91.560581	
62177	49972	56	25	52	145	NaN	178952.48	90.874242	

62178 rows × 28 columns

In [105…
```
dcr_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62178 entries, 0 to 62177
Data columns (total 28 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       62178 non-null  int64  
 1   time                     62178 non-null  int64  
 2   orig_time                62178 non-null  int64  
 3   first_time               62178 non-null  int64  
 4   mat_time                 62178 non-null  int64  
 5   res_time                 1160 non-null   float64
 6   balance_time             62178 non-null  float64
 7   LTV_time                 62153 non-null  float64
 8   interest_rate_time       62178 non-null  float64
 9   rate_time                62178 non-null  float64
 10  hpi_time                 62178 non-null  float64
 11  gdp_time                 62178 non-null  float64
 12  uer_time                 62178 non-null  float64
 13  REtype_CO_orig_time      62178 non-null  int64  
 14  REtype_PU_orig_time      62178 non-null  int64  
 15  REtype_SF_orig_time      62178 non-null  int64  
 16  investor_orig_time       62178 non-null  int64  
 17  balance_orig_time        62178 non-null  float64
 18  FICO_orig_time           62178 non-null  int64  
 19  LTV_orig_time            62178 non-null  float64
 20  Interest_Rate_orig_time  62178 non-null  float64
 21  state_orig_time          61828 non-null  object 
 22  hpi_orig_time            62178 non-null  float64
 23  default_time             62178 non-null  int64  
 24  payoff_time              62178 non-null  int64  
 25  status_time              62178 non-null  int64  
 26  lgd_time                 1525 non-null   float64
 27  recovery_res             1525 non-null   float64
dtypes: float64(14), int64(13), object(1)
memory usage: 13.3+ MB
```

In [106…

```
data_with_missing_years
```

Out[106]:

	State	Year	Median Income
510	Alabama	2001	35160
459	Alabama	2002	37600
408	Alabama	2003	37260
561	Alabama	2004	37260
357	Alabama	2005	37150
...	...	...	...
203	Wyoming	2011	54510
152	Wyoming	2012	57510
764	Wyoming	2013	57510
101	Wyoming	2014	55690
50	Wyoming	2015	60930

765 rows × 3 columns

In [107…

```
#Drop rows with missing 'state_orig_time' in dcr_data
dcr_data_cleaned = dcr_data.dropna(subset=['state_orig_time'])
```

```
dcr_data_cleaned
```

Out[107]:

	id	time	orig_time	first_time	mat_time	res_time	balance_time	LTV_time	interest_
0	4	25	-2	25	119	NaN	61031.10	33.911009	
1	4	26	-2	25	119	NaN	60882.42	34.007232	
2	4	27	-2	25	119	NaN	60729.80	34.335349	
3	4	28	-2	25	119	NaN	60576.14	34.672545	
4	4	29	-2	25	119	NaN	60424.39	34.951639	
...	...	...	...	...	...	...	...	...	
62173	49972	52	25	52	145	NaN	180673.24	103.306966	
62174	49972	53	25	52	145	NaN	179944.95	95.736862	
62175	49972	54	25	52	145	NaN	179451.81	91.867079	
62176	49972	55	25	52	145	NaN	178952.48	91.560581	
62177	49972	56	25	52	145	NaN	178952.48	90.874242	

61828 rows × 28 columns

In [108…

```
print(dcr_data_cleaned['state_orig_time'].unique())
```

```
['KY' 'CO' 'GA' 'TN' 'CA' 'AL' 'NJ' 'DC' 'NC' 'NY' 'FL' 'WA' 'MD' 'AZ'
 'SC' 'MN' 'TX' 'VA' 'OH' 'CT' 'ME' 'MI' 'WI' 'PA' 'OK' 'NV' 'MA' 'LA'
 'IL' 'NE' 'ND' 'MO' 'MT' 'AR' 'OR' 'NM' 'UT' 'IA' 'SD' 'ID' 'HI' 'RI'
 'IN' 'WV' 'VT' 'MS' 'NH' 'DE' 'KS' 'WY' 'PR' 'AK']
```

In [133…

```
# Create a mapping of full state names to abbreviations
state_map = {
    'Kentucky': 'KY', 'Colorado': 'CO', 'Georgia': 'GA', 'Tennessee': 'TN',
    'California': 'CA', 'Alabama': 'AL', 'New Jersey': 'NJ', 'District of Columbia'
    'North Carolina': 'NC', 'New York': 'NY', 'Florida': 'FL', 'Washington': 'WA',
    'Maryland': 'MD', 'Arizona': 'AZ', 'South Carolina': 'SC', 'Minnesota': 'MN',
    'Texas': 'TX', 'Virginia': 'VA', 'Ohio': 'OH', 'Connecticut': 'CT', 'Maine': 'M
    'Michigan': 'MI', 'Wisconsin': 'WI', 'Pennsylvania': 'PA', 'Oklahoma': 'OK',
    'Nevada': 'NV', 'Massachusetts': 'MA', 'Louisiana': 'LA', 'Illinois': 'IL',
    'Nebraska': 'NE', 'North Dakota': 'ND', 'Missouri': 'MO', 'Montana': 'MT',
    'Arkansas': 'AR', 'Oregon': 'OR', 'New Mexico': 'NM', 'Utah': 'UT', 'Iowa': 'IA
    'South Dakota': 'SD', 'Idaho': 'ID', 'Hawaii': 'HI', 'Rhode Island': 'RI',
    'Indiana': 'IN', 'West Virginia': 'WV', 'Vermont': 'VT', 'Mississippi': 'MS',
    'New Hampshire': 'NH', 'Delaware': 'DE', 'Kansas': 'KS', 'Wyoming': 'WY',
    'Puerto Rico': 'PR', 'Alaska': 'AK', 'Puerto Rico': 'PR'
}

# Map full state names to abbreviations in 'data_with_missing_years'
data_with_missing_years['State_abbr'] = data_with_missing_years['State'].map(state_
```

In [134…

```
data_with_missing_years
```

Out[134]:

	State	Year	Median Income	State_abbr
510	Alabama	2001	35160	AL
459	Alabama	2002	37600	AL
408	Alabama	2003	37260	AL
561	Alabama	2004	37260	AL
357	Alabama	2005	37150	AL
...	...	...	...	...
203	Wyoming	2011	54510	WY
152	Wyoming	2012	57510	WY
764	Wyoming	2013	57510	WY
101	Wyoming	2014	55690	WY
50	Wyoming	2015	60930	WY

765 rows × 4 columns

In [135…

```
data_with_missing_years.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 765 entries, 510 to 50
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   State          765 non-null    object
 1   Year           765 non-null    int64 
 2   Median Income  765 non-null    int64 
 3   State_abbr     765 non-null    object
dtypes: int64(2), object(2)
memory usage: 29.9+ KB
```

In [450…

```
dcr_data_cleaned['Year'] = (dcr_data['time'] - 1) // 4 + 2001
```

```
C:\Users\2017\AppData\Local\Temp\ipykernel_20252\1586149626.py:1: SettingWithCopyW
arning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  dcr_data_cleaned['Year'] = (dcr_data['time'] - 1) // 4 + 2001
```

In [137…

```
dcr_data_cleaned
```

Out[137]:

	id	time	orig_time	first_time	mat_time	res_time	balance_time	LTV_time	interest_
0	4	25	-2	25	119	NaN	61031.10	33.911009	
1	4	26	-2	25	119	NaN	60882.42	34.007232	
2	4	27	-2	25	119	NaN	60729.80	34.335349	
3	4	28	-2	25	119	NaN	60576.14	34.672545	
4	4	29	-2	25	119	NaN	60424.39	34.951639	
...	...	...	...	...	...	...	...	...	
62173	49972	52	25	52	145	NaN	180673.24	103.306966	
62174	49972	53	25	52	145	NaN	179944.95	95.736862	
62175	49972	54	25	52	145	NaN	179451.81	91.867079	
62176	49972	55	25	52	145	NaN	178952.48	91.560581	
62177	49972	56	25	52	145	NaN	178952.48	90.874242	

61828 rows × 29 columns

In [146…

```
merged_data = pd.merge(dcr_data_cleaned, data_with_missing_years[['Year', 'Median I
                       left_on=['Year', 'state_orig_time'],
                       right_on=['Year', 'State_abbr'], how='left')

merged_data
```

Out[146]:

	id	time	orig_time	first_time	mat_time	res_time	balance_time	LTV_time	interest_
0	4	25	-2	25	119	NaN	61031.10	33.911009	
1	4	26	-2	25	119	NaN	60882.42	34.007232	
2	4	27	-2	25	119	NaN	60729.80	34.335349	
3	4	28	-2	25	119	NaN	60576.14	34.672545	
4	4	29	-2	25	119	NaN	60424.39	34.951639	
...	...	...	...	...	...	...	...	...	
61823	49972	52	25	52	145	NaN	180673.24	103.306966	
61824	49972	53	25	52	145	NaN	179944.95	95.736862	
61825	49972	54	25	52	145	NaN	179451.81	91.867079	
61826	49972	55	25	52	145	NaN	178952.48	91.560581	
61827	49972	56	25	52	145	NaN	178952.48	90.874242	

61828 rows × 31 columns

In [147…

```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 61828 entries, 0 to 61827
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       61828 non-null  int64  
 1   time                     61828 non-null  int64  
 2   orig_time                61828 non-null  int64  
 3   first_time               61828 non-null  int64  
 4   mat_time                 61828 non-null  int64  
 5   res_time                 1155 non-null   float64
 6   balance_time             61828 non-null  float64
 7   LTV_time                 61803 non-null  float64
 8   interest_rate_time       61828 non-null  float64
 9   rate_time                61828 non-null  float64
 10  hpi_time                 61828 non-null  float64
 11  gdp_time                 61828 non-null  float64
 12  uer_time                 61828 non-null  float64
 13  REtype_CO_orig_time      61828 non-null  int64  
 14  REtype_PU_orig_time      61828 non-null  int64  
 15  REtype_SF_orig_time      61828 non-null  int64  
 16  investor_orig_time       61828 non-null  int64  
 17  balance_orig_time        61828 non-null  float64
 18  FICO_orig_time           61828 non-null  int64  
 19  LTV_orig_time            61828 non-null  float64
 20  Interest_Rate_orig_time  61828 non-null  float64
 21  state_orig_time          61828 non-null  object 
 22  hpi_orig_time            61828 non-null  float64
 23  default_time             61828 non-null  int64  
 24  payoff_time              61828 non-null  int64  
 25  status_time              61828 non-null  int64  
 26  lgd_time                 1520 non-null   float64
 27  recovery_res             1520 non-null   float64
 28  Year                     61828 non-null  int64  
 29  Median Income            61414 non-null  float64
 30  State_abbr               61414 non-null  object 
dtypes: float64(15), int64(14), object(2)
memory usage: 15.1+ MB
```

In [148…

```
missing_state = merged_data[merged_data['State_abbr'].isna()]
missing_state
```

Out[148]:

	id	time	orig_time	first_time	mat_time	res_time	balance_time	LTV_time	interest_ra
24357	20175	28	12	28	72	NaN	369356.80	32.489599	
24358	20175	29	12	28	72	NaN	363427.49	32.306296	
24359	20175	30	12	28	72	NaN	357414.41	32.464769	
24360	20175	31	12	28	72	NaN	351316.38	33.828672	
24361	20175	32	12	28	72	NaN	345132.20	35.831207	
...	...	...	...	...	...	...	...	...	
28001	22561	56	15	28	135	NaN	163100.38	60.407132	
28002	22561	57	15	28	135	NaN	162104.67	58.187812	
28003	22561	58	15	28	135	NaN	161092.70	57.465404	
28004	22561	59	15	28	135	NaN	160064.20	57.317598	
28005	22561	60	15	28	135	NaN	159018.89	56.292313	

414 rows × 31 columns

In [150…

```
merged_data
```

Out[150]:

	id	time	orig_time	first_time	mat_time	res_time	balance_time	LTV_time	interest_
0	4	25	-2	25	119	NaN	61031.10	33.911009	
1	4	26	-2	25	119	NaN	60882.42	34.007232	
2	4	27	-2	25	119	NaN	60729.80	34.335349	
3	4	28	-2	25	119	NaN	60576.14	34.672545	
4	4	29	-2	25	119	NaN	60424.39	34.951639	
...	...	...	...	...	...	...	...	...	
61823	49972	52	25	52	145	NaN	180673.24	103.306966	
61824	49972	53	25	52	145	NaN	179944.95	95.736862	
61825	49972	54	25	52	145	NaN	179451.81	91.867079	
61826	49972	55	25	52	145	NaN	178952.48	91.560581	
61827	49972	56	25	52	145	NaN	178952.48	90.874242	

61828 rows × 31 columns

In [151…

```
merged_data.to_csv('merged.csv')
```

Assumptions for data challenges

1. Missing values

- Rows with State_abbr = 'PR' (which stands for Puerto Rico) in the mortgage dataset don't have corresponding values in the median income dataset. We need to handle missing data for Puerto Rico in some ways such as filling with assumed median income for Puerto Rico (e.g., using the average of nearby years or an industry standard) or removing rows.
- There are 350 rows with missing values in the `state_orig_time` column (0.5%), clean up by removing these rows.
- For numerical columns, we can impute missing values using the mean, median, or interpolation methods. If missing values are for columns like `payoff_time` or `status_time`, they might indicate that no action was taken during that period, so we can fill them with 0.

2. State Matching
In merged data, the `state_orig_time` column corresponds to state abbreviations, while `State_abbr` is the same in the merged data. If any state abbreviation mismatch occurs, we would need to perform a clean-up to ensure they match correctly across datasets.

3. Data Consistency
Ensure that `Median Income` and `Year` match correctly for each row. If there are inconsistencies, we might have to filter out rows where the data doesn't correspond to the expected mapping.

4. Year-to-Time Mapping
As the time ranges from 1 to 60, which corresponds to years between 2001 and 2015, there could be edge cases where the mapping might not align precisely due to gaps in the data or incorrect time values. We can handle this by checking if there is any time outside the expected range and correcting them as needed.

2. PD modelling

A. Estimate a basic credit risk model for mortgage default probabilities

- Examples: credit ratings (AAA to C), FICO score (from 350 to 850)
- Scores/ratings are ordinal/rank on purpose: rank measures is easier to produce, less litigious

A default event is generally recorded if one of the following conditions are met:

- Payment delinquency of 30, 60 and 90 days or more;
- Bankruptcy of the borrower;
- Collateral owned by a bank, e.g., real estate owned after an unsuccessful sale at a foreclosure auction;
- Foreclosure of loan;
- Short sale of loan;
- Loss/write-down amount;
- Involuntary liquidation;

- Debt modification as a positive interest, expense, or principle forgiveness.

Here, default indicator is represented by the binary variable `default_time`:

$$D_{it} = \begin{cases} 1 & \text{borrower } i \text{ defaults at time } t \\ 0 & \text{otherwise} \end{cases}$$

In [152…

```
dt = merged_data.sort_values(by=['id', 'time'])

print(dt.loc[(dt.id==9)|(dt.id==47),['id', 'time', 'default_time']])
```

```
    id  time  default_time
35   9    25             0
36   9    26             0
37   9    27             0
38   9    28             0
39   9    29             0
40   9    30             0
41   9    31             0
42   9    32             0
43   9    33             0
44   9    34             0
45   9    35             0
46   9    36             0
47   9    37             1
48  47    25             0
49  47    26             0
50  47    27             0
```

Probabilities of default are expectation of default events:

$$E(D_{it} = 1) = PD_{it} * 1 + (1 - PD_{it}) * 0 = PD_{it}$$

We now build probability models that explain the drivers of default events.

In [155…

```
import statsmodels.formula.api as smf
```

In [157…

```
data_ols=smf.ols(formula='default_time ~ LTV_orig_time', data=dt).fit()
```

In [158…

```
dir(data_ols)
```

Out[158]:
```
['HC0_se',
 'HC1_se',
 'HC2_se',
 'HC3_se',
 '_HCCM',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_abat_diagonal',
 '_cache',
 '_data_attr',
 '_data_in_cache',
 '_get_robustcov_results',
 '_get_wald_nonlinear',
 '_is_nested',
 '_transform_predict_exog',
 '_use_t',
 '_wexog_singular_values',
 'aic',
 'bic',
 'bse',
 'centered_tss',
 'compare_f_test',
 'compare_lm_test',
 'compare_lr_test',
 'condition_number',
 'conf_int',
 'conf_int_el',
 'cov_HC0',
 'cov_HC1',
 'cov_HC2',
 'cov_HC3',
 'cov_kwds',
 'cov_params',
 'cov_type',
 'df_model',
 'df_resid',
 'eigenvals',
 'el_test',
 'ess',
```

```
 'f_pvalue',
 'f_test',
 'fittedvalues',
 'fvalue',
 'get_influence',
 'get_prediction',
 'get_robustcov_results',
 'info_criteria',
 'initialize',
 'k_constant',
 'llf',
 'load',
 'model',
 'mse_model',
 'mse_resid',
 'mse_total',
 'nobs',
 'normalized_cov_params',
 'outlier_test',
 'params',
 'predict',
 'pvalues',
 'remove_data',
 'resid',
 'resid_pearson',
 'rsquared',
 'rsquared_adj',
 'save',
 'scale',
 'ssr',
 'summary',
 'summary2',
 't_test',
 't_test_pairwise',
 'tvalues',
 'uncentered_tss',
 'use_t',
 'wald_test',
 'wald_test_terms',
 'wresid']
```

In [159…

```
data_ols.summary()
```

Out[159]:

OLS Regression Results

Dep. Variable:	default_time	**R-squared:**	0.001
Model:	OLS	**Adj. R-squared:**	0.001
Method:	Least Squares	**F-statistic:**	81.44
Date:	Sat, 03 May 2025	**Prob (F-statistic):**	1.86e-19
Time:	22:00:11	**Log-Likelihood:**	27636.
No. Observations:	61828	**AIC:**	-5.527e+04
Df Residuals:	61826	**BIC:**	-5.525e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-0.0186	0.005	-3.859	0.000	-0.028	-0.009
LTV_orig_time	0.0005	6.09e-05	9.024	0.000	0.000	0.001

Omnibus:	69092.412	**Durbin-Watson:**	2.020
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3652910.720
Skew:	6.128	**Prob(JB):**	0.00
Kurtosis:	38.606	**Cond. No.**	616.

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

In [160…

```
data_ols.fittedvalues
```

Out[160]:

```
0        0.026292
1        0.026292
2        0.026292
3        0.026292
4        0.026292
           ...
61823    0.025193
61824    0.025193
61825    0.025193
61826    0.025193
61827    0.025193
Length: 61828, dtype: float64
```

In [162…

```
data_ols.predict(dt)
```

Out[162]:
```
0        0.026292
1        0.026292
2        0.026292
3        0.026292
4        0.026292
           ...
61823    0.025193
61824    0.025193
61825    0.025193
61826    0.025193
61827    0.025193
Length: 61828, dtype: float64
```

In [163…
```
data_ols.fittedvalues.describe()
```

Out[163]:
```
count    61828.000000
mean         0.024584
std          0.005617
min          0.008879
25%          0.022557
50%          0.025303
75%          0.025303
max          0.047165
dtype: float64
```

Model with LTV-ratio

In [168…
```
data_ols2=smf.ols(formula='default_time ~ LTV_time', data=dt).fit()
```

In [169…
```
data_ols2.params
```

Out[169]:
```
Intercept   -0.011917
LTV_time     0.000437
dtype: float64
```

$$\hat{PD}_{it} = \hat{P}(D_{it} = 1|x_{it-1}) = --0.011917 + 0.000437' x_{it-1}$$

In [170…
```
data_ols2.fittedvalues.describe()
```

Out[170]:
```
count    61803.000000
mean         0.024578
std          0.012255
min         -0.011917
25%          0.017461
50%          0.024081
75%          0.032185
max          0.339452
dtype: float64
```

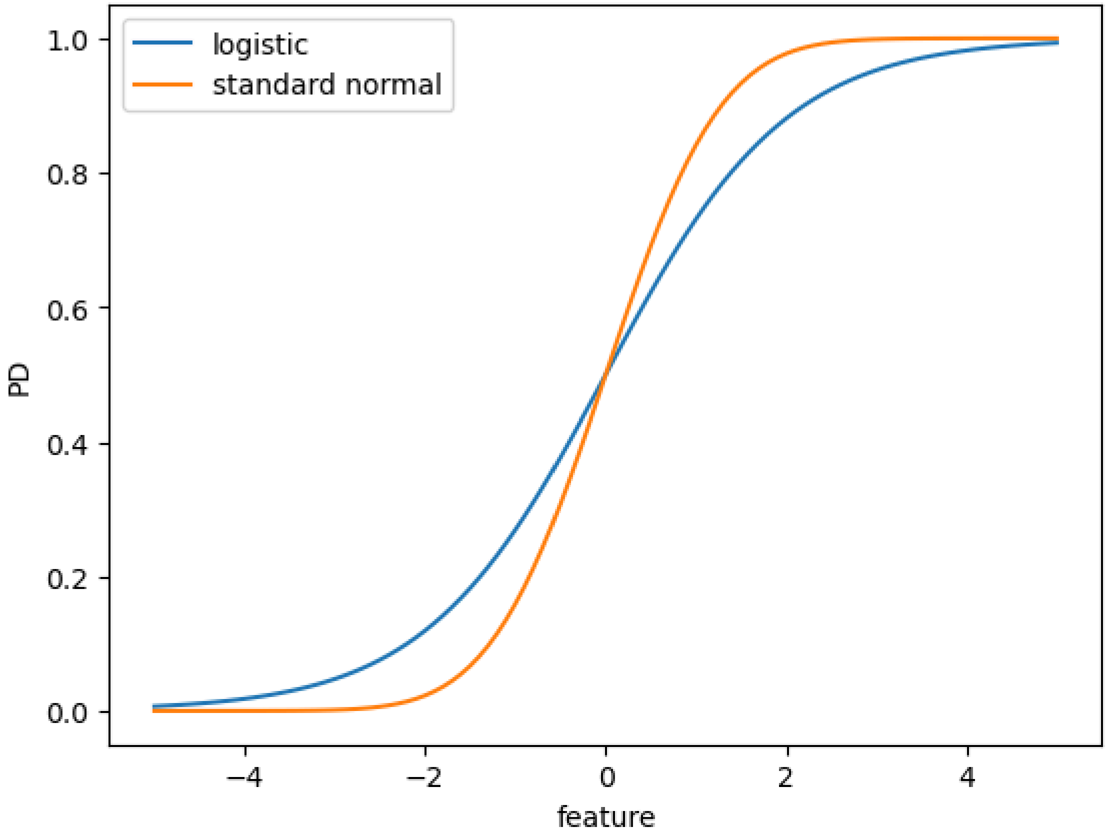
In [171…
```
PD_ols=pd.DataFrame(data_ols2.fittedvalues, columns=['PD_ols_model'])
```

Non-linear Regresion Models

In [187…
```
import statsmodels.api as sm
```

In [193…
```
x=np.arange(-5,5.1,0.1)
logistic= np.exp(x)/(1+np.exp(x))
standardnormal=scipy.stats.norm.cdf(x,0,1)

plt.plot(x,logistic,label='logistic')
```

```
plt.plot(x,standardnormal,label='standard normal')
plt.xlabel('feature')
plt.ylabel('PD')
plt.legend(loc='best')
plt.show()
```



In [426…

```
data_logistic=smf.glm('default_time ~ LTV_time', family=sm.families.Binomial(), dat
```

In [427…

```
data_logistic.fittedvalues.describe()
```

Out[427]:

```
count    61803.000000
mean         0.024578
std          0.017367
min          0.012303
25%          0.020969
50%          0.023628
75%          0.027334
max          0.890592
dtype: float64
```

In [428…

```
PD_logistic=pd.DataFrame(data_logistic.fittedvalues, columns=['PD_logistic_model'])
```

In [481…

```
data_logistic2=smf.glm('default_time ~ LTV_time + time + FICO_orig_time', family=sm

data_logistic2.summary()
```

Out[481]:

Generalized Linear Model Regression Results

Dep. Variable:	default_time	**No. Observations:**	61803
Model:	GLM	**Df Residuals:**	61799
Model Family:	Binomial	**Df Model:**	3
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-6925.2
Date:	Sun, 04 May 2025	**Deviance:**	13850.
Time:	10:32:40	**Pearson chi2:**	5.66e+04
No. Iterations:	7	**Pseudo R-squ. (CS):**	0.006587
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-0.9093	0.242	-3.753	0.000	-1.384	-0.434
LTV_time	0.0093	0.001	14.411	0.000	0.008	0.011
time	-0.0051	0.002	-2.179	0.029	-0.010	-0.001
FICO_orig_time	-0.0051	0.000	-14.220	0.000	-0.006	-0.004

In [482...

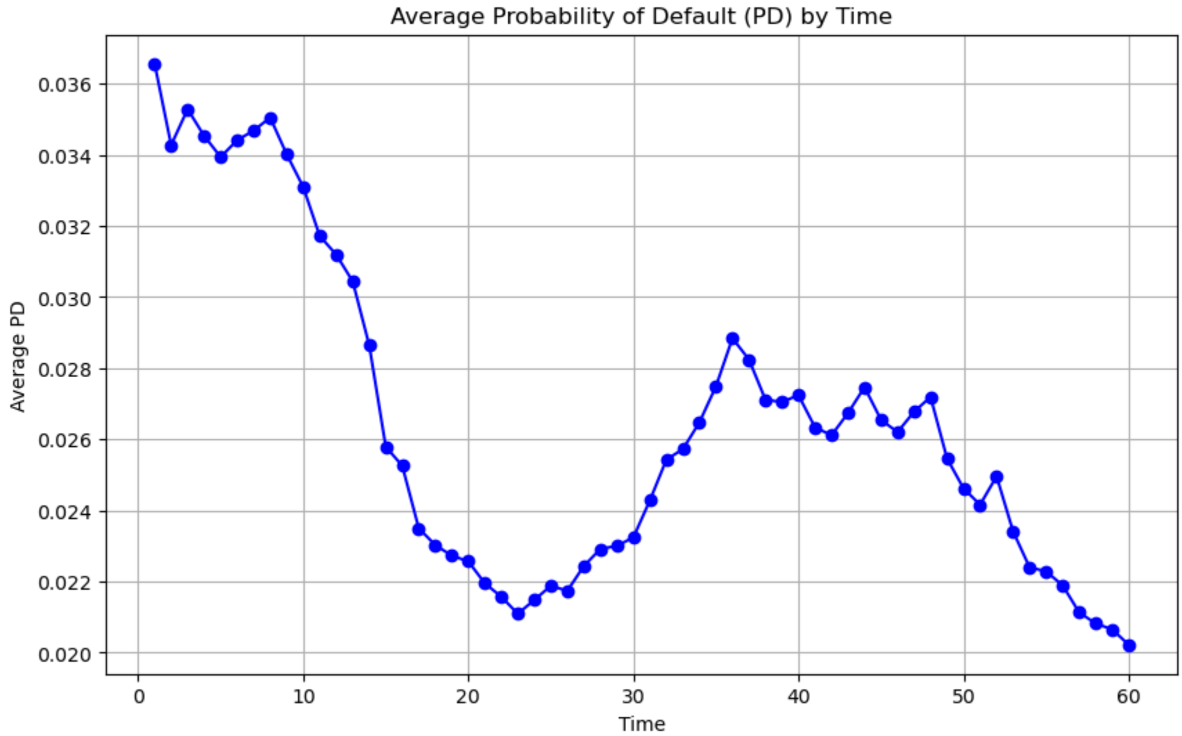
```
data_logistic_pred = data_logistic2.predict(dt)
```

In [483...

```
dt['predicted_PD_logistic'] = data_logistic_pred
```

In [484...

```
avg_PD_logistic_by_time = dt.groupby('time')['predicted_PD_logistic'].mean()

plt.figure(figsize=(10, 6))
plt.plot(avg_PD_logistic_by_time, marker='o', linestyle='-', color='b')
plt.title('Average Probability of Default (PD) by Time')
plt.xlabel('Time')
plt.ylabel('Average PD')
plt.grid(True)
plt.show()
```



Interpret the Plot

The plot shows that the average probability of default fluctuates over time, peaking around time period 20 and then decreasing towards the end of the time period. This might suggest that borrowers have a higher likelihood of default at certain points in the mortgage life cycle, possibly linked to external factors like market conditions or borrower behavior.

Early spikes in PD indicate the beginning of economic distress, where borrowers are more likely to default due to macroeconomic challenges (e.g., economic recessions, high inflation, interest rate increases). **Mid-periods** show stabilization, suggesting recovery in income levels, financial conditions and possibly government interventions. The **later periods** demonstrate continued recovery with lower PD values indicating improved borrower repayment ability and economic stability. The **sharp drop at the end** shows that, after the economic distress, the situation improved, leading to fewer defaults.

In [433…

```
data_train=dt.query('time<=27')
data_test=dt.query('time>27')
```

In [434…

```
dt
```

Out[434]:

	id	time	orig_time	first_time	mat_time	res_time	balance_time	LTV_time	interest_
0	4	25	-2	25	119	NaN	61031.10	33.911009	
1	4	26	-2	25	119	NaN	60882.42	34.007232	
2	4	27	-2	25	119	NaN	60729.80	34.335349	
3	4	28	-2	25	119	NaN	60576.14	34.672545	
4	4	29	-2	25	119	NaN	60424.39	34.951639	
...	...	...	...	...	...	...	...	...	
61823	49972	52	25	52	145	NaN	180673.24	103.306966	
61824	49972	53	25	52	145	NaN	179944.95	95.736862	
61825	49972	54	25	52	145	NaN	179451.81	91.867079	
61826	49972	55	25	52	145	NaN	178952.48	91.560581	
61827	49972	56	25	52	145	NaN	178952.48	90.874242	

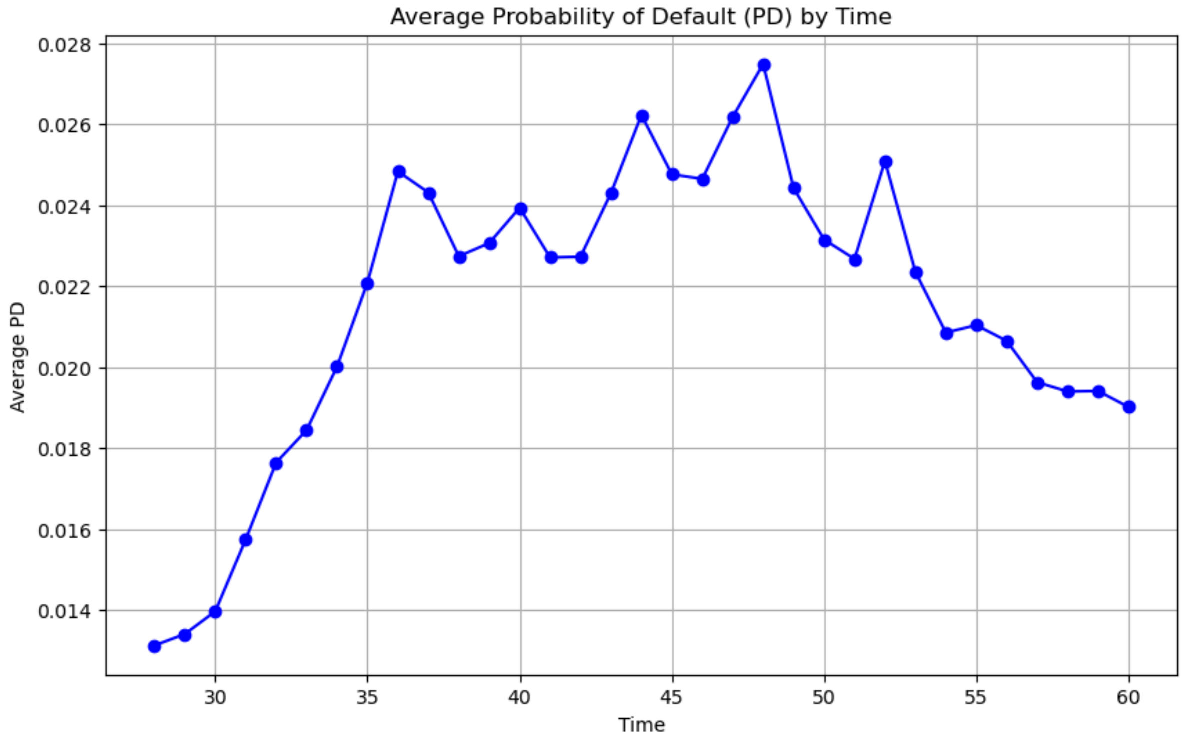
61803 rows × 33 columns

Backtesting using GLM models

For backtesting, we split the data along the feature `time` into a pre-crisis period training sample `data_train` and post-crisis test sample `data_test`. Training samples are used for model fitting and Testing samples are used for model testing

In [567…

```
data_logistic2=smf.glm('default_time ~ LTV_time + FICO_orig_time + time + LTV_time'

PD_logistic2=pd.DataFrame(data_logistic2.predict(data_test), columns=['PD_logistic2
data_test2=pd.merge(data_test, PD_logistic2, right_index=True, left_index=True)

data_test3=data_test2[['PD_logistic2_model', 'default_time', 'time']].dropna()

avg_PD_by_time = data_test3.groupby('time')['PD_logistic2_model'].mean()

plt.figure(figsize=(10, 6))
plt.plot(avg_PD_by_time, marker='o', linestyle='-', color='b')
plt.title('Average Probability of Default (PD) by Time')
plt.xlabel('Time')
plt.ylabel('Average PD')
plt.grid(True)
plt.show()
```



B. Estimate the PD model again by including explanatory variables in part (a), and the state-level income growth from Question 1

In [468…
```
growth_rate_summary.to_csv('growth_rate.csv')
```

In [469…
```
growth_rate = pd.read_csv('growth_rate.csv')
growth_rate
```

Out[469]:

	State	count	mean	std	min	25%	50%	75%	max
0	Alabama	14.0	1.774977	4.131877	-4.249101	-0.221417	0.000000	4.494129	11.225296
1	Alaska	14.0	2.151485	6.715166	-10.251602	0.000000	0.474146	7.422611	11.644807
2	Arizona	14.0	1.530276	4.128221	-6.955504	0.000000	0.600086	3.640075	9.910129
3	Arkansas	14.0	1.994700	6.666770	-5.520581	-2.438091	0.000000	3.512231	15.120451
4	California	14.0	2.211040	3.687767	-6.384845	0.000000	1.518965	5.153069	6.877898
5	Colorado	14.0	2.258878	4.783895	-3.790614	-0.245339	0.000000	5.674333	10.406343
6	Connecticut	14.0	2.306686	3.444876	-1.788444	0.000000	0.993001	3.291231	9.781844
7	Delaware	14.0	1.282460	6.484064	-10.409806	0.000000	0.050403	3.660423	17.459669
8	District of Columbia	14.0	4.056811	6.551097	-5.100802	0.000000	1.310779	7.004016	18.099548
9	Florida	14.0	2.167287	3.378493	-2.031011	0.000000	0.399047	3.919564	10.315627
10	Georgia	14.0	1.321702	3.552114	-4.954770	-0.421804	0.000000	2.854760	8.223374
11	Hawaii	14.0	2.580432	9.235906	-9.421511	-3.002550	0.000000	4.772199	26.590828
12	Idaho	14.0	2.275012	4.961976	-3.578691	0.000000	0.042176	4.514102	12.327678
13	Illinois	14.0	2.049098	4.905422	-7.494044	0.000000	0.983553	6.037823	9.996358
14	Indiana	14.0	1.874645	3.463229	-4.449699	0.000000	0.841403	4.048843	8.156471
15	Iowa	14.0	2.929521	3.830046	0.000000	0.039888	1.212254	4.833272	12.373127
16	Kansas	14.0	2.111343	4.259456	-4.974000	0.000000	1.337949	5.801692	8.374970
17	Kentucky	14.0	0.745217	3.109554	-4.370447	-0.512600	0.000000	2.436766	7.602180
18	Louisiana	14.0	2.455673	5.586251	-4.236262	-1.102617	0.000000	6.902409	13.209098
19	Maine	14.0	2.476559	5.202829	-1.837169	0.000000	0.327779	4.676467	18.350849
20	Maryland	14.0	2.443945	5.692840	-7.268215	0.000000	1.539186	5.340690	15.675779
21	Massachusetts	14.0	1.952524	3.880892	-4.574163	0.000000	0.276418	4.513088	9.929356
22	Michigan	14.0	1.369973	2.959585	-5.172031	0.000000	1.165339	3.566867	5.922055
23	Minnesota	14.0	1.983666	3.794549	-5.390975	0.000000	2.433227	3.679517	8.802589
24	Mississippi	14.0	2.224808	6.285740	-10.829886	0.000000	0.229148	5.899830	12.729767
25	Missouri	14.0	2.675822	4.222260	-1.759598	0.000000	1.177997	3.644728	13.806270
26	Montana	14.0	3.579847	6.122253	-6.107226	0.000000	0.293542	9.144681	13.328898
27	Nebraska	14.0	2.456976	4.580281	-6.148867	0.000000	1.299173	5.540837	9.639267
28	Nevada	14.0	1.109961	5.254026	-14.066496	0.000000	0.552910	4.053872	8.442232
29	New Hampshire	14.0	2.880462	3.922953	-2.071619	0.000000	1.494628	6.606491	9.052767
30	New Jersey	14.0	2.175660	6.067717	-11.093153	0.000000	1.356056	6.585532	13.059768
31	New Mexico	14.0	2.344564	4.992873	-5.094680	-0.213777	0.000000	6.156464	10.937055
32	New York	14.0	2.422395	4.944520	-5.845182	0.000000	0.924937	2.880464	13.905201
33	North Carolina	14.0	2.258331	6.559622	-8.095554	-0.999770	0.000000	7.772805	12.821888

	State	count	mean	std	min	25%	50%	75%	max
34	North Dakota	14.0	3.611938	6.360125	-5.450354	0.000000	0.572786	7.951761	15.006090
35	Ohio	14.0	1.844349	4.514617	-4.858300	0.000000	0.781250	3.417039	11.852186
36	Oklahoma	14.0	2.077839	3.777931	-2.499484	-0.077383	0.000000	4.446162	11.277034
37	Oregon	14.0	2.883256	4.104127	-0.386623	0.000000	0.884690	5.366860	13.711858
38	Pennsylvania	14.0	2.439290	3.899370	-2.898833	0.000000	0.505882	5.760095	9.461664
39	Rhode Island	14.0	1.611667	6.512159	-7.907588	-1.342003	0.000000	5.190228	14.358556
40	South Carolina	14.0	1.582744	4.794788	-4.933586	0.000000	0.092740	2.830051	11.585058
41	South Dakota	14.0	2.496461	5.233685	-8.488372	0.000000	2.993453	5.127681	11.158983
42	Tennessee	14.0	2.060559	3.055636	-3.640777	0.000000	1.498835	3.432156	8.257091
43	Texas	14.0	2.380759	3.066057	-2.191781	0.000000	2.355269	5.307933	6.326483
44	Utah	14.0	2.630218	6.652265	-11.272785	0.000000	0.549218	4.988050	16.831683
45	Vermont	14.0	2.898740	6.121342	-8.830319	0.000000	1.436224	6.608772	17.198336
46	Virginia	14.0	1.560711	4.848753	-7.058646	0.000000	0.508146	3.481031	10.376788
47	Washington	14.0	3.455165	5.237281	-5.016884	0.000000	2.772818	6.539577	13.831048
48	West Virginia	14.0	2.879771	6.976794	-9.741031	0.000000	2.068388	9.231241	11.580381
49	Wisconsin	14.0	1.559401	5.157508	-4.562672	-0.117005	0.000000	1.562963	15.767077

In [470…

```
state_map = {
    'Kentucky': 'KY', 'Colorado': 'CO', 'Georgia': 'GA', 'Tennessee': 'TN',
    'California': 'CA', 'Alabama': 'AL', 'New Jersey': 'NJ', 'District of Columbia'
    'North Carolina': 'NC', 'New York': 'NY', 'Florida': 'FL', 'Washington': 'WA',
    'Maryland': 'MD', 'Arizona': 'AZ', 'South Carolina': 'SC', 'Minnesota': 'MN',
    'Texas': 'TX', 'Virginia': 'VA', 'Ohio': 'OH', 'Connecticut': 'CT', 'Maine': 'M
    'Michigan': 'MI', 'Wisconsin': 'WI', 'Pennsylvania': 'PA', 'Oklahoma': 'OK',
    'Nevada': 'NV', 'Massachusetts': 'MA', 'Louisiana': 'LA', 'Illinois': 'IL',
    'Nebraska': 'NE', 'North Dakota': 'ND', 'Missouri': 'MO', 'Montana': 'MT',
    'Arkansas': 'AR', 'Oregon': 'OR', 'New Mexico': 'NM', 'Utah': 'UT', 'Iowa': 'IA
    'South Dakota': 'SD', 'Idaho': 'ID', 'Hawaii': 'HI', 'Rhode Island': 'RI',
    'Indiana': 'IN', 'West Virginia': 'WV', 'Vermont': 'VT', 'Mississippi': 'MS',
    'New Hampshire': 'NH', 'Delaware': 'DE', 'Kansas': 'KS', 'Wyoming': 'WY',
    'Puerto Rico': 'PR', 'Alaska': 'AK', 'Puerto Rico': 'PR'
}

# Map full state names to abbreviations in 'data_with_missing_years'
growth_rate['State_abbr'] = growth_rate['State'].map(state_map)
```

In [471…

```
growth_rate
```

Out[471]:

	State	count	mean	std	min	25%	50%	75%	max
0	Alabama	14.0	1.774977	4.131877	-4.249101	-0.221417	0.000000	4.494129	11.225296
1	Alaska	14.0	2.151485	6.715166	-10.251602	0.000000	0.474146	7.422611	11.644807
2	Arizona	14.0	1.530276	4.128221	-6.955504	0.000000	0.600086	3.640075	9.910129
3	Arkansas	14.0	1.994700	6.666770	-5.520581	-2.438091	0.000000	3.512231	15.120451
4	California	14.0	2.211040	3.687767	-6.384845	0.000000	1.518965	5.153069	6.877898
5	Colorado	14.0	2.258878	4.783895	-3.790614	-0.245339	0.000000	5.674333	10.406343
6	Connecticut	14.0	2.306686	3.444876	-1.788444	0.000000	0.993001	3.291231	9.781844
7	Delaware	14.0	1.282460	6.484064	-10.409806	0.000000	0.050403	3.660423	17.459669
8	District of Columbia	14.0	4.056811	6.551097	-5.100802	0.000000	1.310779	7.004016	18.099548
9	Florida	14.0	2.167287	3.378493	-2.031011	0.000000	0.399047	3.919564	10.315627
10	Georgia	14.0	1.321702	3.552114	-4.954770	-0.421804	0.000000	2.854760	8.223374
11	Hawaii	14.0	2.580432	9.235906	-9.421511	-3.002550	0.000000	4.772199	26.590828
12	Idaho	14.0	2.275012	4.961976	-3.578691	0.000000	0.042176	4.514102	12.327678
13	Illinois	14.0	2.049098	4.905422	-7.494044	0.000000	0.983553	6.037823	9.996358
14	Indiana	14.0	1.874645	3.463229	-4.449699	0.000000	0.841403	4.048843	8.156471
15	Iowa	14.0	2.929521	3.830046	0.000000	0.039888	1.212254	4.833272	12.373127
16	Kansas	14.0	2.111343	4.259456	-4.974000	0.000000	1.337949	5.801692	8.374970
17	Kentucky	14.0	0.745217	3.109554	-4.370447	-0.512600	0.000000	2.436766	7.602180
18	Louisiana	14.0	2.455673	5.586251	-4.236262	-1.102617	0.000000	6.902409	13.209098
19	Maine	14.0	2.476559	5.202829	-1.837169	0.000000	0.327779	4.676467	18.350849
20	Maryland	14.0	2.443945	5.692840	-7.268215	0.000000	1.539186	5.340690	15.675779
21	Massachusetts	14.0	1.952524	3.880892	-4.574163	0.000000	0.276418	4.513088	9.929356
22	Michigan	14.0	1.369973	2.959585	-5.172031	0.000000	1.165339	3.566867	5.922055
23	Minnesota	14.0	1.983666	3.794549	-5.390975	0.000000	2.433227	3.679517	8.802589
24	Mississippi	14.0	2.224808	6.285740	-10.829886	0.000000	0.229148	5.899830	12.729767
25	Missouri	14.0	2.675822	4.222260	-1.759598	0.000000	1.177997	3.644728	13.806270
26	Montana	14.0	3.579847	6.122253	-6.107226	0.000000	0.293542	9.144681	13.328898
27	Nebraska	14.0	2.456976	4.580281	-6.148867	0.000000	1.299173	5.540837	9.639267
28	Nevada	14.0	1.109961	5.254026	-14.066496	0.000000	0.552910	4.053872	8.442232
29	New Hampshire	14.0	2.880462	3.922953	-2.071619	0.000000	1.494628	6.606491	9.052767
30	New Jersey	14.0	2.175660	6.067717	-11.093153	0.000000	1.356056	6.585532	13.059768
31	New Mexico	14.0	2.344564	4.992873	-5.094680	-0.213777	0.000000	6.156464	10.937055
32	New York	14.0	2.422395	4.944520	-5.845182	0.000000	0.924937	2.880464	13.905201
33	North Carolina	14.0	2.258331	6.559622	-8.095554	-0.999770	0.000000	7.772805	12.821888

	State	count	mean	std	min	25%	50%	75%	max
34	North Dakota	14.0	3.611938	6.360125	-5.450354	0.000000	0.572786	7.951761	15.006090
35	Ohio	14.0	1.844349	4.514617	-4.858300	0.000000	0.781250	3.417039	11.852186
36	Oklahoma	14.0	2.077839	3.777931	-2.499484	-0.077383	0.000000	4.446162	11.277034
37	Oregon	14.0	2.883256	4.104127	-0.386623	0.000000	0.884690	5.366860	13.711858
38	Pennsylvania	14.0	2.439290	3.899370	-2.898833	0.000000	0.505882	5.760095	9.461664
39	Rhode Island	14.0	1.611667	6.512159	-7.907588	-1.342003	0.000000	5.190228	14.358556
40	South Carolina	14.0	1.582744	4.794788	-4.933586	0.000000	0.092740	2.830051	11.585058
41	South Dakota	14.0	2.496461	5.233685	-8.488372	0.000000	2.993453	5.127681	11.158983
42	Tennessee	14.0	2.060559	3.055636	-3.640777	0.000000	1.498835	3.432156	8.257091
43	Texas	14.0	2.380759	3.066057	-2.191781	0.000000	2.355269	5.307933	6.326483
44	Utah	14.0	2.630218	6.652265	-11.272785	0.000000	0.549218	4.988050	16.831683
45	Vermont	14.0	2.898740	6.121342	-8.830319	0.000000	1.436224	6.608772	17.198336
46	Virginia	14.0	1.560711	4.848753	-7.058646	0.000000	0.508146	3.481031	10.376788
47	Washington	14.0	3.455165	5.237281	-5.016884	0.000000	2.772818	6.539577	13.831048
48	West Virginia	14.0	2.879771	6.976794	-9.741031	0.000000	2.068388	9.231241	11.580381
49	Wisconsin	14.0	1.559401	5.157508	-4.562672	-0.117005	0.000000	1.562963	15.767077

In [472…

```
dt_merged = dt.merge(growth_rate[['State_abbr', 'mean']], on='State_abbr', how='lef

# Check the merged result
dt_merged.head()
```

Out[472]:

	id	time	orig_time	first_time	mat_time	res_time	balance_time	LTV_time	interest_rate_time
0	4	25	-2	25	119	NaN	61031.10	33.911009	10.500
1	4	26	-2	25	119	NaN	60882.42	34.007232	10.500
2	4	27	-2	25	119	NaN	60729.80	34.335349	10.500
3	4	28	-2	25	119	NaN	60576.14	34.672545	10.875
4	4	29	-2	25	119	NaN	60424.39	34.951639	10.875

5 rows × 34 columns

In [473…

```
data_with_growth_rate = smf.glm(
    'default_time ~ FICO_orig_time + time + LTV_time + mean',
    family=sm.families.Binomial(),
    data=dt_merged
).fit()
```

In [474…

```
data_with_growth_rate.summary()
```

Out[474]:

Generalized Linear Model Regression Results

Dep. Variable:	default_time	**No. Observations:**	61389
Model:	GLM	**Df Residuals:**	61384
Model Family:	Binomial	**Df Model:**	4
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-6897.9
Date:	Sun, 04 May 2025	**Deviance:**	13796.
Time:	10:31:52	**Pearson chi2:**	5.62e+04
No. Iterations:	7	**Pseudo R-squ. (CS):**	0.006706
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-0.6090	0.273	-2.235	0.025	-1.143	-0.075
FICO_orig_time	-0.0052	0.000	-14.260	0.000	-0.006	-0.004
time	-0.0051	0.002	-2.141	0.032	-0.010	-0.000
LTV_time	0.0093	0.001	14.430	0.000	0.008	0.011
mean	-0.1351	0.058	-2.346	0.019	-0.248	-0.022

In [475…

```
dt_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 61803 entries, 0 to 61802
Data columns (total 34 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   id                        61803 non-null  int64
 1   time                      61803 non-null  int64
 2   orig_time                 61803 non-null  int64
 3   first_time                61803 non-null  int64
 4   mat_time                  61803 non-null  int64
 5   res_time                  1155 non-null   float64
 6   balance_time              61803 non-null  float64
 7   LTV_time                  61803 non-null  float64
 8   interest_rate_time        61803 non-null  float64
 9   rate_time                 61803 non-null  float64
 10  hpi_time                  61803 non-null  float64
 11  gdp_time                  61803 non-null  float64
 12  uer_time                  61803 non-null  float64
 13  REtype_CO_orig_time       61803 non-null  int64
 14  REtype_PU_orig_time       61803 non-null  int64
 15  REtype_SF_orig_time       61803 non-null  int64
 16  investor_orig_time        61803 non-null  int64
 17  balance_orig_time         61803 non-null  float64
 18  FICO_orig_time            61803 non-null  int64
 19  LTV_orig_time             61803 non-null  float64
 20  Interest_Rate_orig_time   61803 non-null  float64
 21  state_orig_time           61803 non-null  object
 22  hpi_orig_time             61803 non-null  float64
 23  default_time              61803 non-null  int64
 24  payoff_time               61803 non-null  int64
 25  status_time               61803 non-null  int64
 26  lgd_time                  1519 non-null   float64
 27  recovery_res              1519 non-null   float64
 28  Year                      61803 non-null  int64
 29  Median Income             61389 non-null  float64
 30  State_abbr                61389 non-null  object
 31  predicted_PD              61803 non-null  float64
 32  predicted_PD_logistic     61803 non-null  float64
 33  mean                      61389 non-null  float64
dtypes: float64(18), int64(14), object(2)
memory usage: 16.5+ MB
```

In [476…

```
PD_logistic_with_growth_rate = data_with_growth_rate.predict(dt_merged)
dt_merged.loc[:, 'PD_logistic_with_growth_rate'] = PD_logistic_with_growth_rate
```

In [477…

```
dt_merged.info()
```

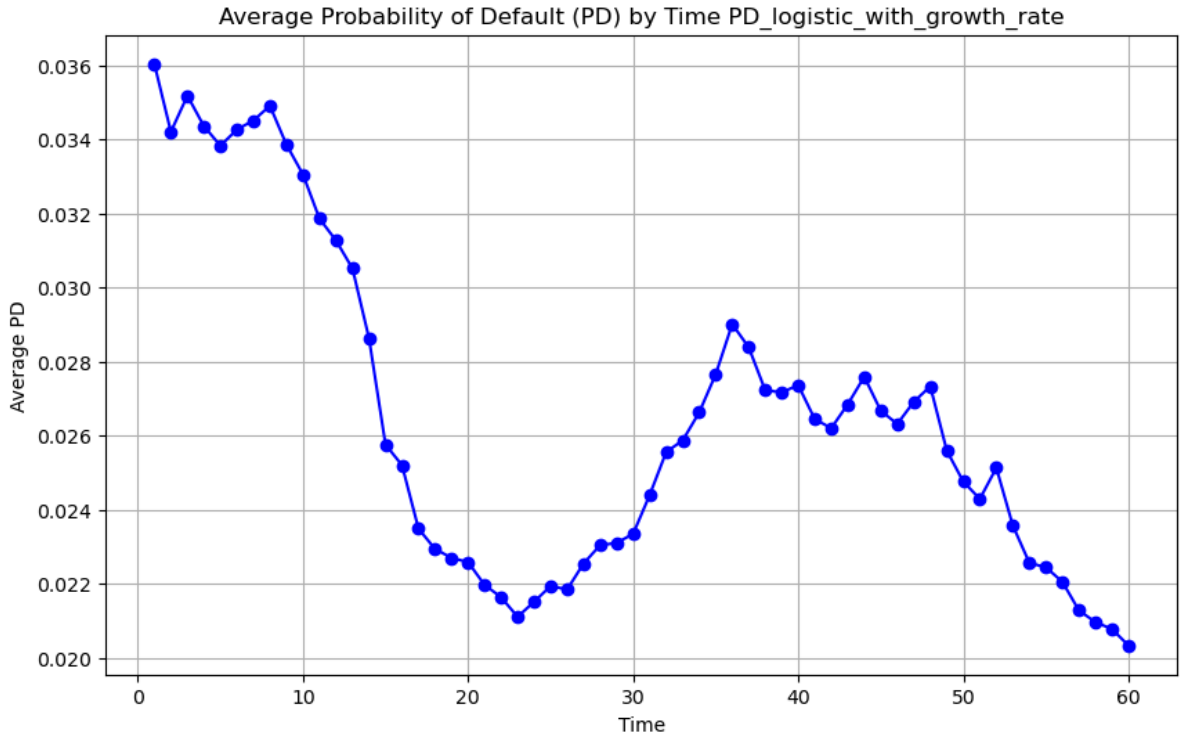
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 61803 entries, 0 to 61802
Data columns (total 35 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   id                            61803 non-null  int64
 1   time                          61803 non-null  int64
 2   orig_time                     61803 non-null  int64
 3   first_time                    61803 non-null  int64
 4   mat_time                      61803 non-null  int64
 5   res_time                      1155 non-null   float64
 6   balance_time                  61803 non-null  float64
 7   LTV_time                      61803 non-null  float64
 8   interest_rate_time            61803 non-null  float64
 9   rate_time                     61803 non-null  float64
 10  hpi_time                      61803 non-null  float64
 11  gdp_time                      61803 non-null  float64
 12  uer_time                      61803 non-null  float64
 13  REtype_CO_orig_time           61803 non-null  int64
 14  REtype_PU_orig_time           61803 non-null  int64
 15  REtype_SF_orig_time           61803 non-null  int64
 16  investor_orig_time            61803 non-null  int64
 17  balance_orig_time             61803 non-null  float64
 18  FICO_orig_time                61803 non-null  int64
 19  LTV_orig_time                 61803 non-null  float64
 20  Interest_Rate_orig_time       61803 non-null  float64
 21  state_orig_time               61803 non-null  object
 22  hpi_orig_time                 61803 non-null  float64
 23  default_time                  61803 non-null  int64
 24  payoff_time                   61803 non-null  int64
 25  status_time                   61803 non-null  int64
 26  lgd_time                      1519 non-null   float64
 27  recovery_res                  1519 non-null   float64
 28  Year                          61803 non-null  int64
 29  Median Income                 61389 non-null  float64
 30  State_abbr                    61389 non-null  object
 31  predicted_PD                  61803 non-null  float64
 32  predicted_PD_logistic         61803 non-null  float64
 33  mean                          61389 non-null  float64
 34  PD_logistic_with_growth_rate  61389 non-null  float64
dtypes: float64(19), int64(14), object(2)
memory usage: 17.0+ MB
```

In [478…

```
dt_merged = dt_merged.dropna(subset=['PD_logistic_with_growth_rate', 'default_time'
```

In [479…

```
avg_PD_logistic_by_time = dt_merged.groupby('time')['PD_logistic_with_growth_rate']

plt.figure(figsize=(10, 6))
plt.plot(avg_PD_logistic_by_time, marker='o', linestyle='-', color='b')
plt.title('Average Probability of Default (PD) by Time PD_logistic_with_growth_rate
plt.xlabel('Time')
plt.ylabel('Average PD')
plt.grid(True)
plt.show()
```



Interpret output

The plot showing the **Average Probability of Default (PD) by Time with State Income Growth** provides insights into the relationship between the time period and the default probability of mortgage loans across different states, while also factoring in the income growth within those states.

Initial Periods (Time 0-10):

Firstly, we see a spike in the average PD. This could signify a period of economic uncertainty or a time when the loans were more likely to default. During this time, many states may have been experiencing stagnation or slower growth, causing a higher likelihood of defaults. The income growth at the state level could be negatively impacted during these periods, making it harder for borrowers to meet repayment schedules.

Middle Periods (Time 10-30):

From Time 10 onward, we observe a sharp decline in PD, especially between **Time 15 to 20** (2004:Q3 to 2005:Q4). This could represent a recovery period where the economic environment improved, potentially driven by favorable income growth across several states.

Later Periods (Time 30-60):

Flatter PD Curve shows that after Time 30, the PD stabilizes at a lower level, with a more gradual decline as the time progresses. This might indicate that while some states have recovered from the economic downturn, the remaining loan defaults are mainly driven by other factors, such as changing interest rates or credit conditions.

Besides, there is a slight increase in PD after **Time 50** (2012:Q2). This could suggest some new economic challenges, possibly related to external shocks such as recessions, tightening of credit, or regional economic slowdowns.

Overall, the relationship between income growth and default probability is inverse. When state-level income grows, the probability of default tends to decline. Conversely, when income stagnates or declines, the PD tends to increase as borrowers may struggle to meet payment schedules. The plot also suggests that macroeconomic factors, such as income growth, recession periods, etc. can affect mortgage defaults.

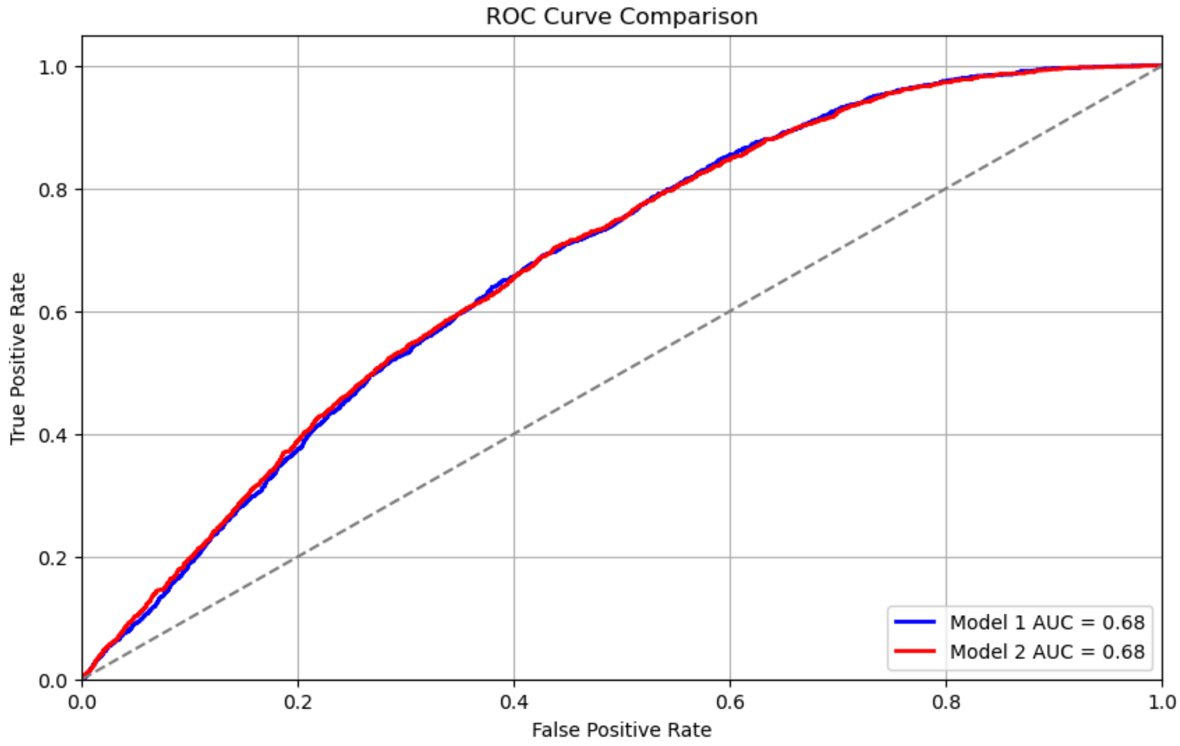
In [458…
```
dt = dt.dropna(subset=['predicted_PD_logistic', 'default_time'])
```

C. Compare the accuracy

In [485…
```
from sklearn.metrics import roc_auc_score, roc_curve

# Model 1: Predictions for 2a model (already fitted)
# Use the predicted probabilities for Model 1
y_true = dt_merged['default_time']  # Actual values (default)
y_pred_model1 = dt_merged['predicted_PD_logistic']  # Predicted probabilities from

# Calculate AUC for Model 1
roc_auc_model1 = roc_auc_score(y_true, y_pred_model1)
fpr_model1, tpr_model1, _ = roc_curve(y_true, y_pred_model1)
```

In [486…
```
# Model 2: Predictions for 2b model (including state-level income growth)
# Use the predicted probabilities for Model 2
y_true = dt_merged['default_time']
y_pred_model2 = dt_merged['PD_logistic_with_growth_rate']  # Predicted probabilitie

# Calculate AUC for Model 2
roc_auc_model2 = roc_auc_score(y_true, y_pred_model2)
fpr_model2, tpr_model2, _ = roc_curve(y_true, y_pred_model2)
```

In [487…
```
plt.figure(figsize=(10, 6))
plt.plot(fpr_model1, tpr_model1, color='blue', lw=2, label=f'Model 1 AUC = {roc_auc
plt.plot(fpr_model2, tpr_model2, color='red', lw=2, label=f'Model 2 AUC = {roc_auc_
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



In [488...

```
print(f"Model 1 AUC: {roc_auc_model1:.4f}")
print(f"Model 2 AUC: {roc_auc_model2:.4f}")
```

```
Model 1 AUC: 0.6775
Model 2 AUC: 0.6792
```

In [489...

```
from sklearn.metrics import accuracy_score, roc_auc_score, log_loss

predictions_model_1 = dt_merged['predicted_PD_logistic']  # For Model 1
predictions_model_2 = dt_merged['PD_logistic_with_growth_rate']  # For Model 2

true_labels = dt_merged['default_time']

# Model 1 Accuracy
accuracy_model_1 = accuracy_score(true_labels, (predictions_model_1 > 0.5))
log_loss_model_1 = log_loss(true_labels, predictions_model_1)

# Model 2 Accuracy
accuracy_model_2 = accuracy_score(true_labels, (predictions_model_2 > 0.5))
log_loss_model_2 = log_loss(true_labels, predictions_model_2)

print(f"Model 1 Accuracy: {accuracy_model_1:.4f}")
print(f"Model 1 Log-Loss: {log_loss_model_1:.4f}")

print(f"Model 2 Accuracy: {accuracy_model_2:.4f}")
print(f"Model 2 Log-Loss: {log_loss_model_2:.4f}")
```

```
Model 1 Accuracy: 0.9748
Model 1 Log-Loss: 0.1124
Model 2 Accuracy: 0.9748
Model 2 Log-Loss: 0.1124
```

Model 1 and Model 2 perform similarly. However, **Model 2**'s slight increase in AUC could be indicative of better classification performance at certain thresholds. The growth rate variable is potentially important but does not add significant value to the predictive power of the model.

Both models show a similar AUC score (Model 1: 0.6775 and Model 2: 0.6792), which

indicates a similar ability to differentiate. The slight increase in AUC in **Model 2** suggests that adding the growth rate variable slightly improves the model's ability to correctly classify the outcome, although the improvement is small.
Both models achieve an identical accuracy **of 0.9748, which suggests that the accuracy rate is quite high. Both models have the same** Log-Loss** value of 0.1124. Log-Loss measures the uncertainty of the model's predictions, with lower values indicating better model performance.

3. LGD modelling

A. Predict LGD

$$LGD_{it} = \frac{EAD_{it} - \sum_{\tau=1}^{T}(CF_{t+\tau}/(1+r_{t+\tau})^{t+\tau})}{EAD_{it}}$$

- EAD: outstanding loan amount at default
- $\sum_{\tau=1}^{T}(CF_{t+\tau}/(1+r_{t+\tau})^{t+\tau})$: present value of recoveries, these can include incoming and outgoing (cost) cashflows.

In [359…

```
data_default = dt_merged.query('default_time==1').copy()

data_default.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1515 entries, 47 to 61802
Data columns (total 35 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   id                           1515 non-null   int64  
 1   time                         1515 non-null   int64  
 2   orig_time                    1515 non-null   int64  
 3   first_time                   1515 non-null   int64  
 4   mat_time                     1515 non-null   int64  
 5   res_time                     1154 non-null   float64
 6   balance_time                 1515 non-null   float64
 7   LTV_time                     1515 non-null   float64
 8   interest_rate_time           1515 non-null   float64
 9   rate_time                    1515 non-null   float64
 10  hpi_time                     1515 non-null   float64
 11  gdp_time                     1515 non-null   float64
 12  uer_time                     1515 non-null   float64
 13  REtype_CO_orig_time          1515 non-null   int64  
 14  REtype_PU_orig_time          1515 non-null   int64  
 15  REtype_SF_orig_time          1515 non-null   int64  
 16  investor_orig_time           1515 non-null   int64  
 17  balance_orig_time            1515 non-null   float64
 18  FICO_orig_time               1515 non-null   int64  
 19  LTV_orig_time                1515 non-null   float64
 20  Interest_Rate_orig_time      1515 non-null   float64
 21  state_orig_time              1515 non-null   object 
 22  hpi_orig_time                1515 non-null   float64
 23  default_time                 1515 non-null   int64  
 24  payoff_time                  1515 non-null   int64  
 25  status_time                  1515 non-null   int64  
 26  lgd_time                     1515 non-null   float64
 27  recovery_res                 1515 non-null   float64
 28  Year                         1515 non-null   int64  
 29  Median Income                1515 non-null   float64
 30  State_abbr                   1515 non-null   object 
 31  predicted_PD                 1515 non-null   float64
 32  predicted_PD_logistic        1515 non-null   float64
 33  mean                         1515 non-null   float64
 34  PD_logistic_with_growth_rate 1515 non-null   float64
dtypes: float64(19), int64(14), object(2)
memory usage: 426.1+ KB
```

In [361…

```python
data_default[['orig_time', 'time', 'res_time', 'mat_time']]
```

Out[361]:

	orig_time	time	res_time	mat_time
47	18	37	NaN	138
75	25	37	NaN	141
91	21	40	NaN	141
133	21	31	NaN	142
164	23	31	38.0	143
...	...	...	...	...
61264	21	57	NaN	142
61267	18	54	NaN	139
61375	23	52	NaN	144
61566	23	53	NaN	201
61802	25	56	NaN	145

1515 rows × 4 columns

After loan origination `orig_time` a loan may default. The default time is indicated by `time` . Loans then enter into the resolution period which is finished after the last cash flow is received (indicate by `res_time`). Default time occurs prior to maturity time `mat_time` . THe resolution time be before or after the maturity time.

Time to default

In [363…

```
data_default.loc[:,'ttd_period'] = data_default.loc[:,'time'] - data_default.loc[:,
```

In [364…

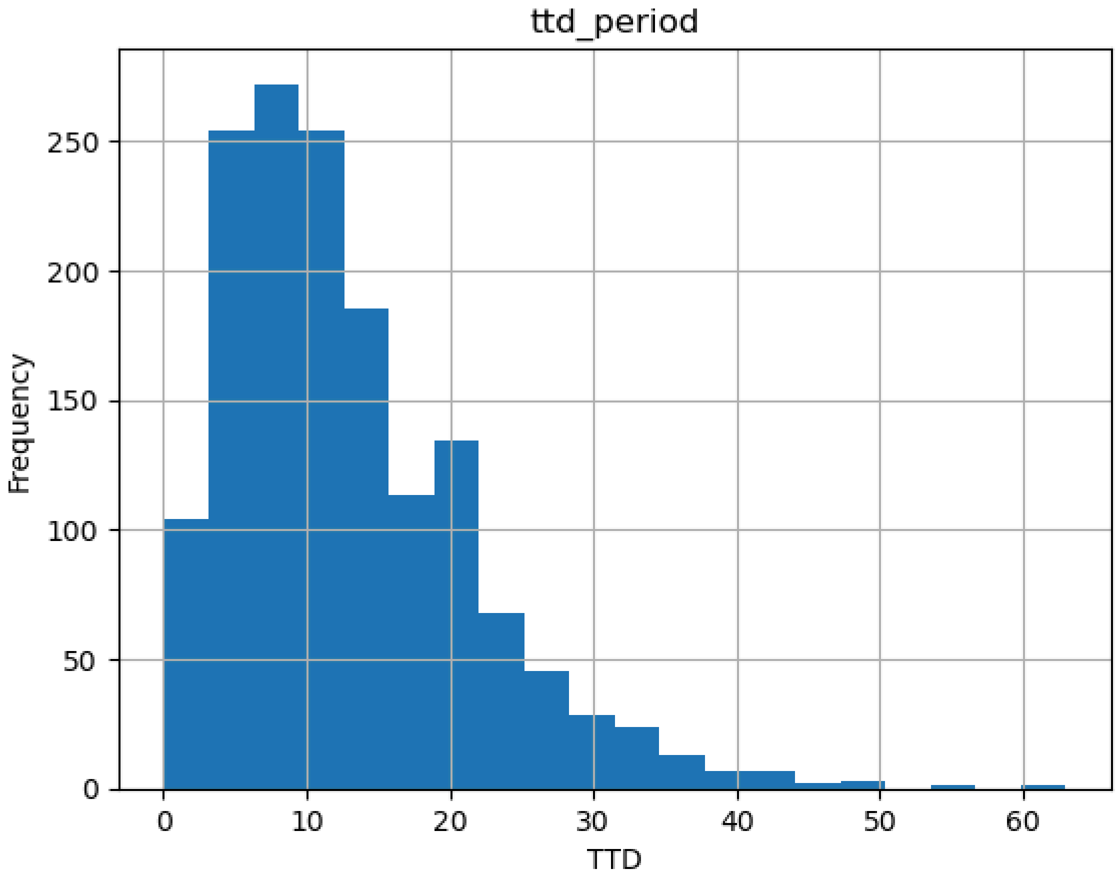
```
data_default.loc[:,'ttd_period'].describe()
```

Out[364]:

```
count    1515.000000
mean       12.935314
std         8.442034
min         0.000000
25%         7.000000
50%        11.000000
75%        17.000000
max        63.000000
Name: ttd_period, dtype: float64
```

In [365…

```
data_default[['ttd_period']].hist(bins=20)
plt.xlabel('TTD')
plt.ylabel('Frequency')
plt.show()
```



Time to resolution

In [367…

```
data_default.loc[:,'res_period']=data_default.loc[:,'res_time']-data_default.loc[:,
```

In [368…

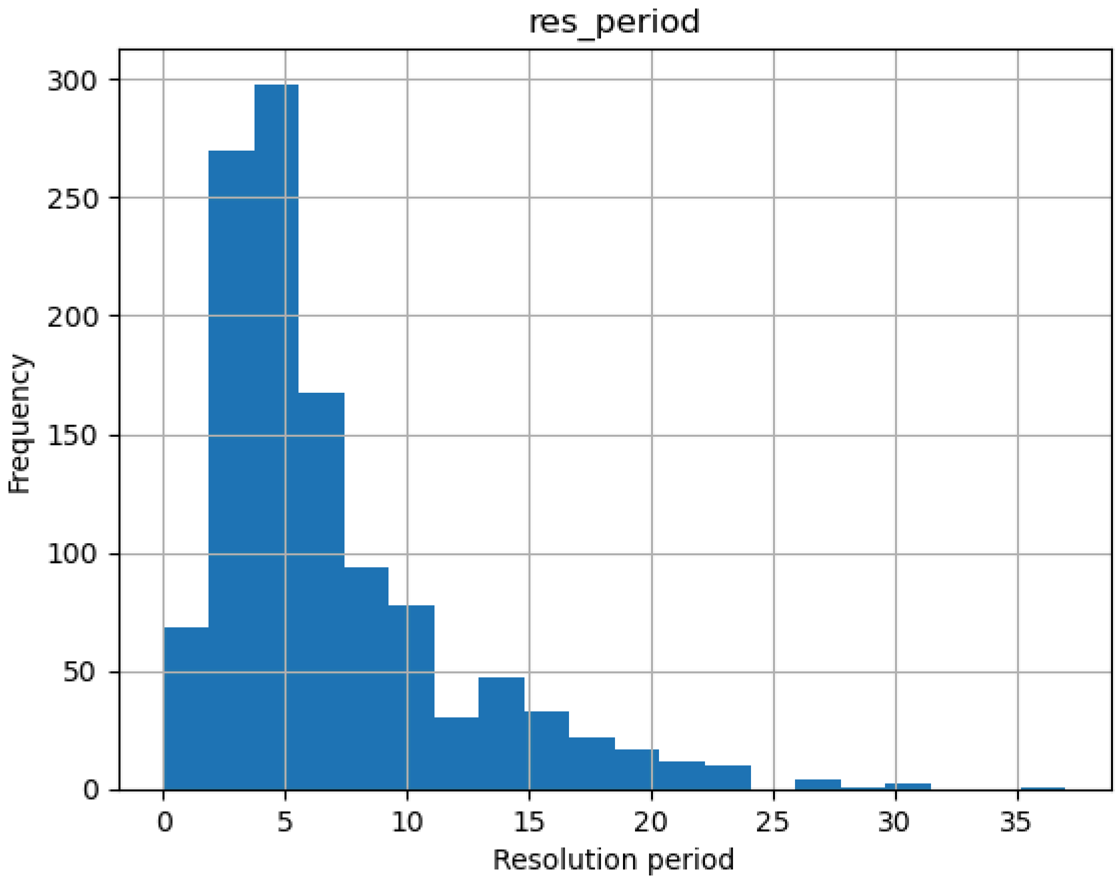
```
data_default[['res_period']].describe()
```

Out[368]:

	res_period
count	1154.000000
mean	6.670711
std	5.141440
min	0.000000
25%	3.000000
50%	5.000000
75%	8.000000
max	37.000000

In [369…

```
data_default[['res_period']].hist(bins=20)
plt.xlabel('Resolution period')
plt.ylabel('Frequency')
plt.show()
```



Calculation of LGD

In [371…

```
data_default.loc[:,'LGD'] = (data_default.loc[:,'balance_time'] - data_default.loc[
```

In [372…

```
data_default.loc[:,'LGD'].describe()
```

Out[372]:

```
count    1154.000000
mean        0.615219
std         0.329079
min        -0.033736
25%         0.388340
50%         0.636635
75%         0.842615
max         1.814265
Name: LGD, dtype: float64
```

In [373…

```
data_default[['res_period','LGD']].corr()
```

Out[373]:

	res_period	LGD
res_period	1.000000	0.384515
LGD	0.384515	1.000000

In [374…

```
LGD_mean=data_default.groupby('time')[['LGD']].mean().reset_index(drop=False)
LGD_mean
```

Out[374]:

	time	LGD
0	3	0.522505
1	4	0.148862
2	5	0.289206
3	6	0.194931
4	7	0.360203
5	8	0.130560
6	9	0.798333
7	10	0.154742
8	11	0.140276
9	12	0.095852
10	13	0.227658
11	14	0.094597
12	15	1.036597
13	16	0.223994
14	17	0.139593
15	18	0.110181
16	19	0.350341
17	20	0.507339
18	21	0.381462
19	22	0.313888
20	23	0.319299
21	24	0.474889
22	25	0.499424
23	26	0.355702
24	27	0.462395
25	28	0.581114
26	29	0.567227
27	30	0.625988
28	31	0.752460
29	32	0.614751
30	33	0.713401
31	34	0.709552
32	35	0.734929
33	36	0.649355
34	37	0.649468
35	38	0.668177

	time	LGD
36	39	0.590123
37	40	0.759899
38	41	0.777252
39	42	0.682999
40	43	0.700213
41	44	0.657948
42	45	0.681950
43	46	0.455970
44	47	0.472095
45	48	0.437895
46	49	0.567488
47	50	0.471155
48	51	0.516952
49	52	0.626856
50	53	0.483916
51	54	0.284591
52	55	0.169489
53	56	0.009143
54	57	NaN
55	58	0.053071
56	59	NaN
57	60	NaN

LGD with resolutionbias

In [375…

```
data_default2 = data_default.dropna(subset=['res_time']).copy()
data_default2.loc[data_default2['res_period'] >= 20, 'res_period'] = 20

data_LGD_sum = data_default2.groupby('res_period')[['LGD']].sum()

print(data_LGD_sum)
```

```
                  LGD
res_period
0.0          0.001408
1.0         16.535158
2.0         53.032449
3.0         78.972675
4.0         90.344237
5.0         83.495702
6.0         59.020499
7.0         53.579272
8.0         44.222353
9.0         20.515791
10.0        37.842277
11.0        25.861712
12.0        23.995524
13.0        19.321294
14.0        21.136334
15.0        12.728855
16.0        14.095370
17.0         8.155919
18.0         6.792510
19.0         7.392623
20.0        32.920982
```

$$LGD_{\text{unresolved}} = \frac{1}{\Sigma_{i=1}^{I} I(t_{R,i} - t_{D,i} \geq TEOP - t_D)} \Sigma_{i=1}^{I} LGD_{\text{resolved}, t_{R,i} - t_{D,i} \geq TEOP - t_D}$$

The charts with mean LGDs by time suggest declining LGD levels in the last periods — this may be due to the resolution bias, a recovery from previously high levels or a combination of both.

In [376…

```
data_default2 = data_default.dropna(subset=['res_time']).copy()
data_default2.loc[data_default2['res_period'] >= 20, 'res_period'] = 20

data_LGD_sum = data_default2.groupby('res_period')[['LGD']].sum()

print(data_LGD_sum)
```

```
                  LGD
res_period
0.0          0.001408
1.0         16.535158
2.0         53.032449
3.0         78.972675
4.0         90.344237
5.0         83.495702
6.0         59.020499
7.0         53.579272
8.0         44.222353
9.0         20.515791
10.0        37.842277
11.0        25.861712
12.0        23.995524
13.0        19.321294
14.0        21.136334
15.0        12.728855
16.0        14.095370
17.0         8.155919
18.0         6.792510
19.0         7.392623
20.0        32.920982
```

In [377…

```
data_LGD_count = data_default2.groupby('res_period')[['LGD']].count()

print(data_LGD_count)
```

```
            LGD
res_period
0.0          14
1.0          54
2.0         115
3.0         155
4.0         162
5.0         136
6.0          88
7.0          79
8.0          65
9.0          29
10.0         45
11.0         33
12.0         30
13.0         23
14.0         24
15.0         17
16.0         16
17.0         12
18.0         10
19.0          9
20.0         38
```

In [378…

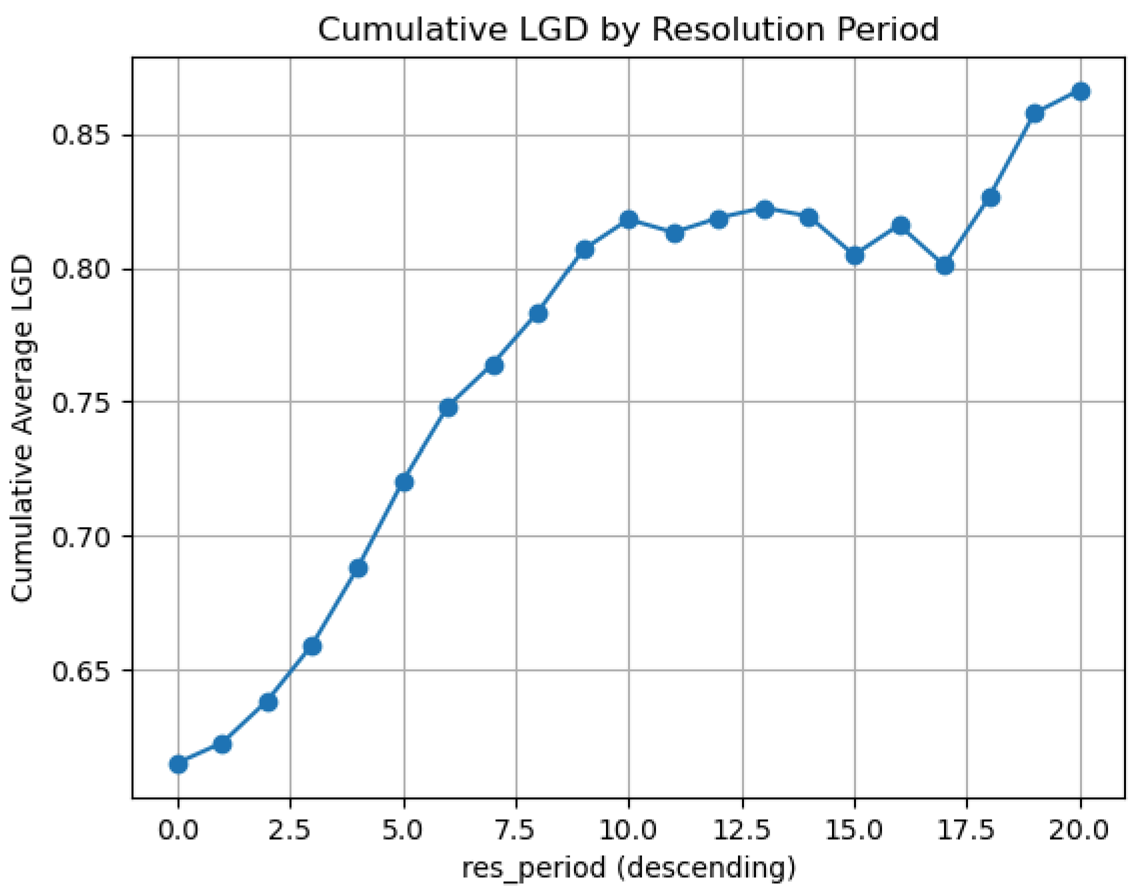
```
data_LGD_sum = data_LGD_sum.sort_values(by='res_period', ascending=False)
data_LGD_count = data_LGD_count.sort_values(by='res_period', ascending=False)

data_LGD_sum_cumsum = data_LGD_sum.cumsum()
data_LGD_count_cumsum = data_LGD_count.cumsum()
data_LGD_mean = data_LGD_sum_cumsum / data_LGD_count_cumsum

print(data_LGD_mean.round(4))
```

```
               LGD
res_period
20.0        0.8663
19.0        0.8577
18.0        0.8264
17.0        0.8009
16.0        0.8160
15.0        0.8048
14.0        0.8192
13.0        0.8224
12.0        0.8187
11.0        0.8132
10.0        0.8181
9.0         0.8069
8.0         0.7834
7.0         0.7641
6.0         0.7482
5.0         0.7203
4.0         0.6880
3.0         0.6595
2.0         0.6385
1.0         0.6228
0.0         0.6152
```

In [379…

```
plt.plot(data_LGD_mean.index, data_LGD_mean['LGD'], marker='o')
plt.xlabel('res_period (descending)')
plt.ylabel('Cumulative Average LGD')
```

```
plt.title('Cumulative LGD by Resolution Period')
plt.grid(True)
plt.show()
```



In [380…

```
data_LGD_mean = data_LGD_mean.iloc[:,0:4]
data_LGD_mean['time'] = 61 - data_LGD_mean.index
data_LGD_mean = data_LGD_mean.set_index('time')

data_LGD_mean2 = data_LGD_mean.iloc[np.full(41, 0)].reset_index(drop=True)
data_LGD_mean3 = pd.concat([data_LGD_mean2,data_LGD_mean]).reset_index(drop=False)
data_LGD_mean3 = data_LGD_mean3.rename(columns={'index': 'time'})

data_default_replace = data_default[data_default.loc[:,'res_time'].isnull()].drop([

data_default_replace2 = pd.merge(data_default_replace, data_LGD_mean3, on='time')

print(data_default_replace2.shape)
```

```
(361, 37)
```

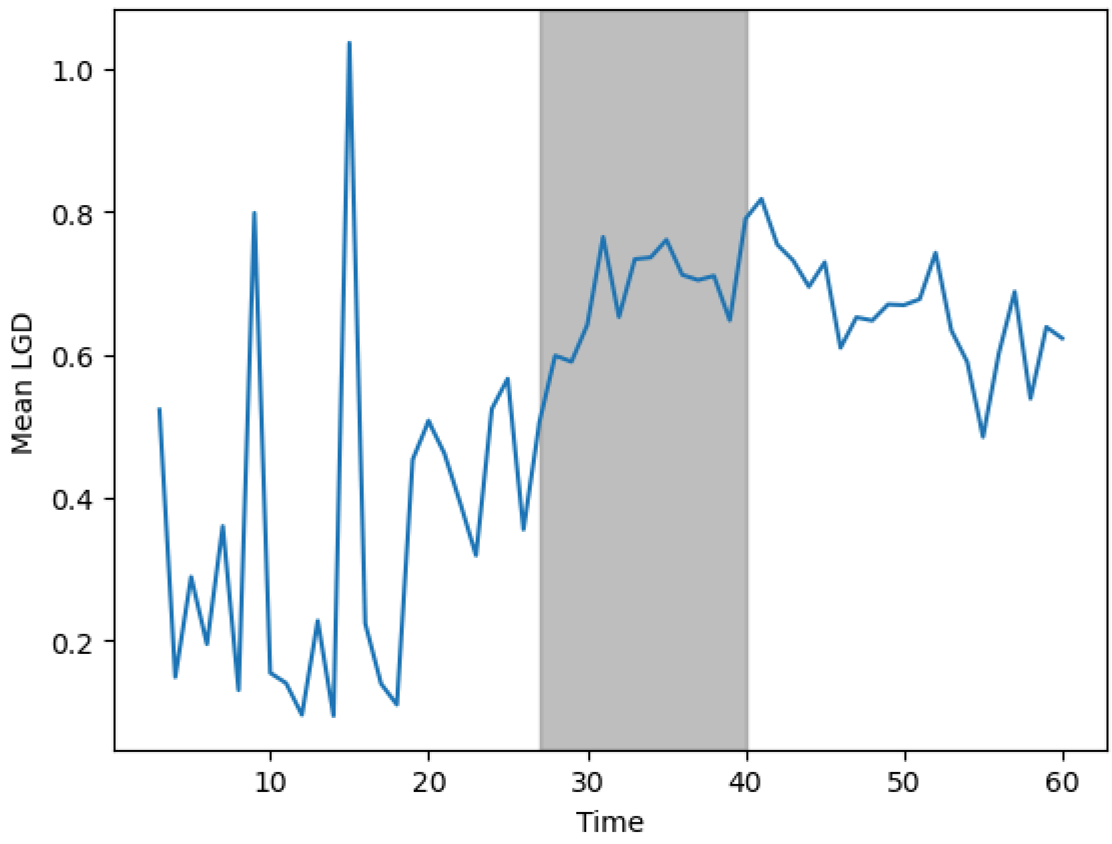
In [381…

```
data_default2 = data_default2[data_default_replace2.columns]
print(data_default2.shape)

data_default3 = pd.concat([data_default2,data_default_replace2]).reset_index(drop=F
print(data_default3.shape)
```

```
(1154, 37)
(1515, 38)
```

In [382…

```
data_default3_mean = data_default3.groupby('time')[['LGD']].mean().reset_index(drop

plt.plot('time', 'LGD', data=data_default3_mean)
plt.axvspan(27,40,color="grey",alpha=0.5)
plt.xlabel('Time')
plt.ylabel('Mean LGD')
plt.show()
```



tabulate moments of empirical LGD distribution

In [383…

```
print(data_default3[['LGD']].dropna().describe().round(decimals=3))
```

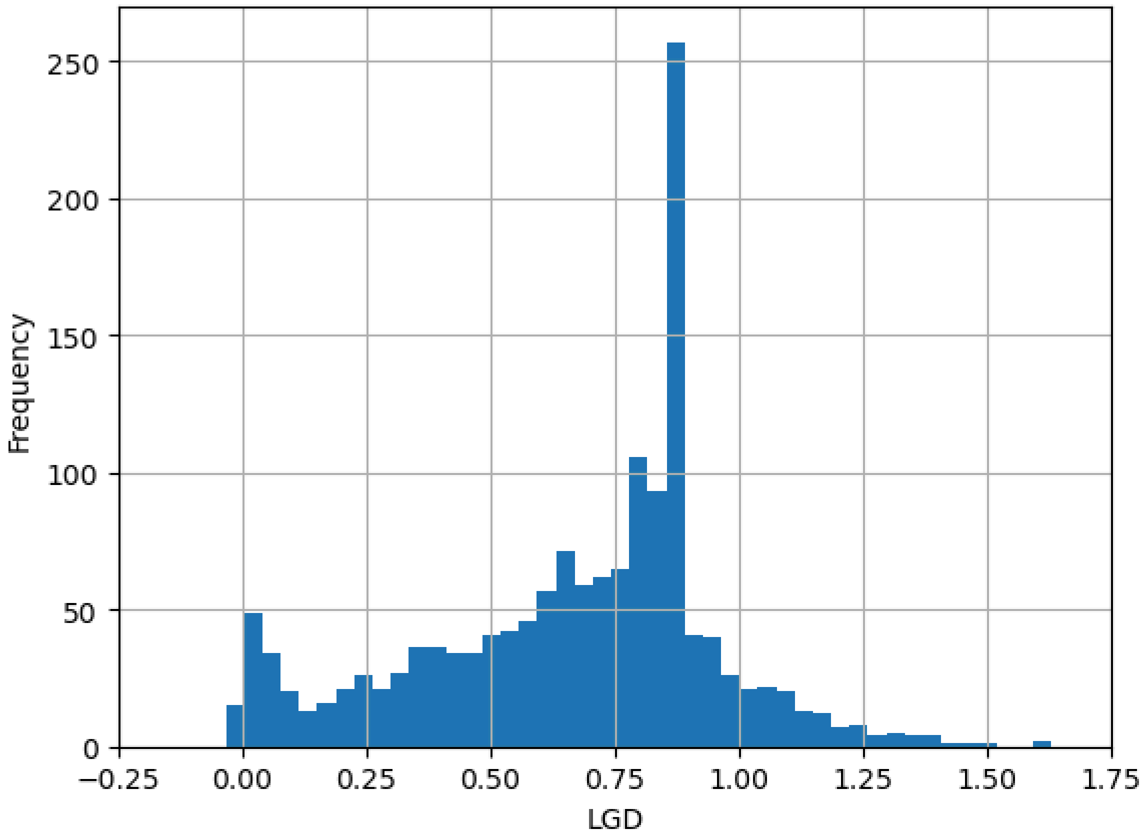
```
            LGD
count  1515.000
mean      0.667
std       0.303
min      -0.034
25%       0.481
50%       0.741
75%       0.866
max       1.814
```

The chart shows that the LGDs towards the end of the observation period no longer decrease as before and remain at comparable levels. The summary statistics show that the LGDs after correction for resolution bias are higher.

We generate a histogram for NLGD definition. The spike around 0.8 is due to the imputation of missing LGD values.

In [386…

```
data_default3.LGD.hist(bins=50)
plt.xlim((-0.25, 1.75))
plt.xlabel('LGD')
plt.ylabel('Frequency')
plt.show()
```



Function `resolutionbias()`

In [394…

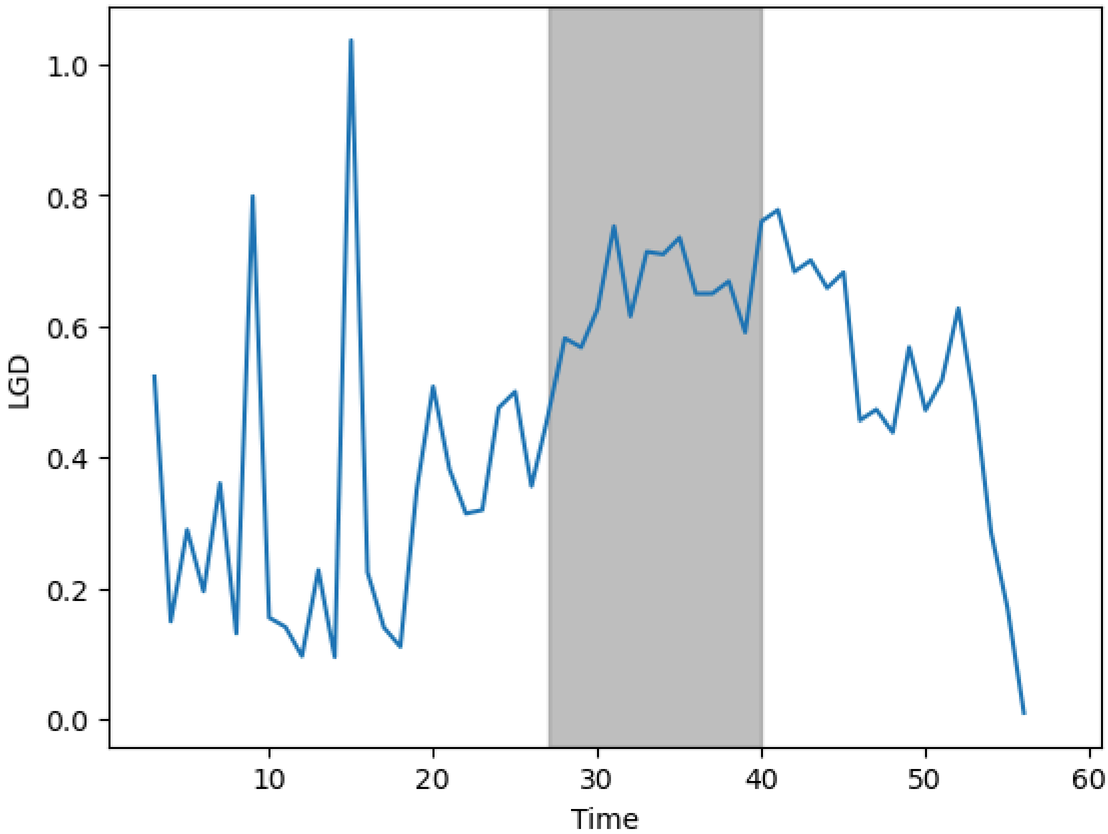
```
def resolutionbias(df, LGD_column, res_time_column, time_column):
    # Calculate the resolution period based on time and resolution time
    df['resolution_bias'] = np.where(
        (df[res_time_column] <= df[time_column]) & (df[time_column] <= df['mat_time
        1,  # Bias is present if within the resolution period
        0   # No bias outside the resolution period
    )

    # Compute the average LGD within the resolution period
    df['LGD'] = df[LGD_column]
    return df
```

In [395…

```
data_default = resolutionbias(data_default, 'LGD', 'res_time', 'time')
```

In [396…

```
LGD_mean=data_default.groupby('time')[['LGD']].mean().reset_index(drop=False)

plt.plot('time','LGD', data=LGD_mean)
plt.axvspan(27, 40, color="grey", alpha=0.5)
plt.xlabel('Time')
plt.ylabel('LGD')
plt.show()
```



LGD Model

In [467…

```
data_default2 =data_default[['LGD', 'LTV_time', 'FICO_orig_time', 'mean', 'time']].

model_ols = smf.ols(formula='LGD ~ LTV_time + FICO_orig_time + mean', data=data_def
```

In [490…

```
model_ols.summary()
```

Out[490]:

OLS Regression Results

Dep. Variable:	LGD	**R-squared:**	0.074
Model:	OLS	**Adj. R-squared:**	0.072
Method:	Least Squares	**F-statistic:**	30.62
Date:	Sun, 04 May 2025	**Prob (F-statistic):**	4.83e-19
Time:	10:34:32	**Log-Likelihood:**	-310.00
No. Observations:	1154	**AIC:**	628.0
Df Residuals:	1150	**BIC:**	648.2
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	0.6261	0.103	6.080	0.000	0.424	0.828
LTV_time	0.0039	0.000	8.357	0.000	0.003	0.005
FICO_orig_time	-0.0003	0.000	-2.036	0.042	-0.001	-1.11e-05
mean	-0.0902	0.020	-4.592	0.000	-0.129	-0.052

Omnibus:	20.854	**Durbin-Watson:**	1.892
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	21.569
Skew:	0.317	**Prob(JB):**	2.07e-05
Kurtosis:	3.216	**Cond. No.**	7.34e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.34e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [491…

```
fittedvalues=pd.DataFrame(model_ols.fittedvalues, columns=['LGD_fit'])
data_default3=pd.merge(data_default2, fittedvalues, right_index=True, left_index=Tr
```

In [492…

```
data_default3
```

Out[492]:

	LGD	LTV_time	FICO_orig_time	mean	time	LGD_fit
164	0.892458	87.176730	630	2.167287	31	0.577602
257	1.210119	98.272701	613	2.167287	33	0.625848
261	0.790201	89.452618	605	2.167287	31	0.594035
273	1.075569	121.463371	633	2.443945	42	0.684853
321	0.366617	122.611681	584	2.380759	39	0.709900
...	...	...	...	...	...	...
61050	1.110404	88.379868	638	2.167287	31	0.579843
61132	1.168885	123.957507	553	2.049098	49	0.754464
61165	0.648558	102.690954	605	2.167287	32	0.645433
61178	0.707012	87.173151	557	2.211040	40	0.595823
61195	0.676541	88.403443	655	2.175660	43	0.574013

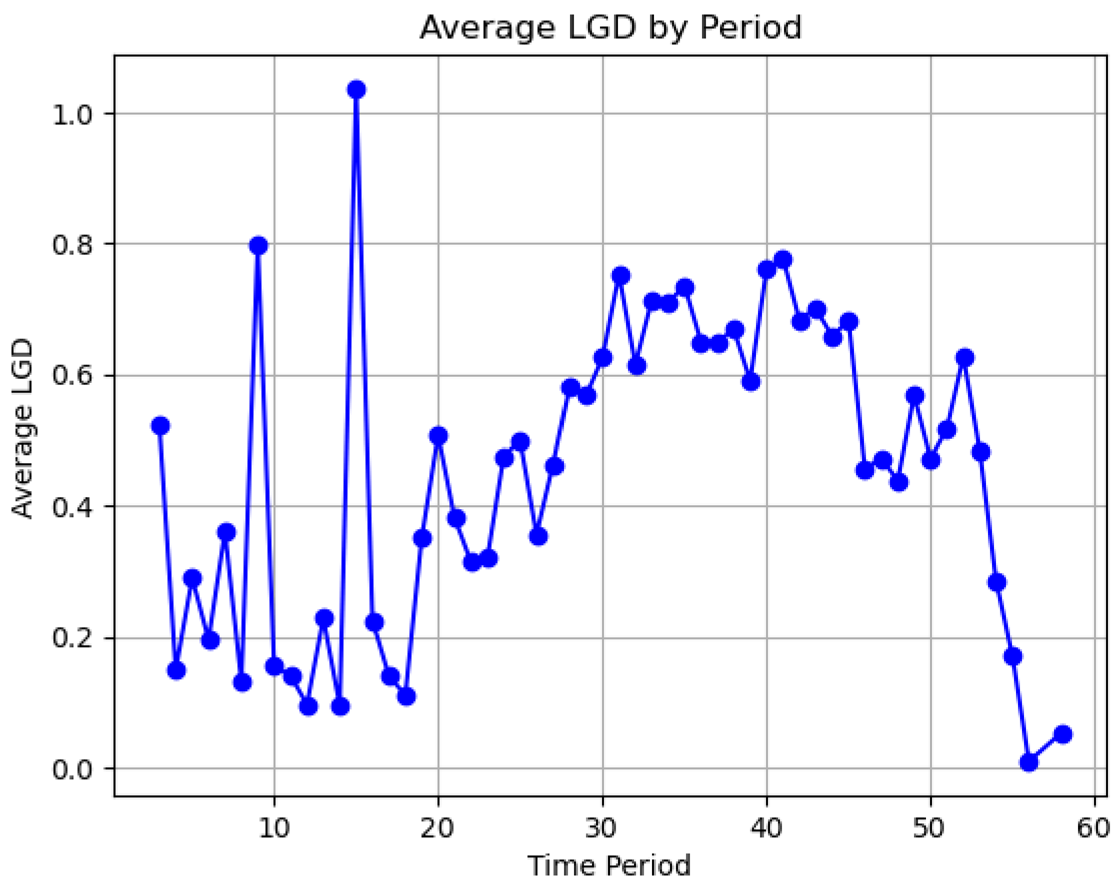
1154 rows × 6 columns

In [506…

```
LGD_mean_by_period = data_default3.groupby('time')['LGD'].mean().reset_index()
```

In [507…

```
plt.plot(LGD_mean_by_period['time'], LGD_mean_by_period['LGD'], marker='o', linesty
plt.title('Average LGD by Period')
plt.xlabel('Time Period')
plt.ylabel('Average LGD')
plt.grid(True)
plt.show()
```



Interpret

The plot of **Average LGD by Period** shows how the **Loss Given Default (LGD)** varies over time across different periods.
The **early periods (Time 1-10, namely 2001:Q1 - 2003:Q2)** of high LGD reflect economic distress, likely tied to external shocks or systemic risks in the housing or credit markets. This volatility could also be due to data imbalances in the early periods or a smaller number of loans during this time frame. High LGD values indicate that, during these periods, borrowers were likely unable to repay loans due to poor economic conditions such as a downturn in the housing market, higher interest rates**, or unfavorable income conditions at the state level.**

There's a stabilization in the average LGD, especially between Time 10 and Time 30 (2003:Q2 - 2008:Q2)**. It suggests an improved financial environment, lower default rates, and better borrower repayment capacity. This may also reflect a recovery phase of the economy.**

Although between Time 30 and Time 50 (2008:Q2 - 20013:Q2)**, the LGD remains relatively stable at lower levels, the** late-period increases (Time 50-60)** indicate that challenges such as rising inflation, credit tightening, or another economic downturn could be starting to affect borrowers' ability to repay loans.

4. Generative AI

A

Generative AI has promise for assisting the credit risk prediction process through increased productivity, prednisone and individualized services throughout the credit life cycle (McKinsey, 2024). There are various ways that generative AI might support including Data Augmentation, Feature, Anomaly Detection, Conditional Generation, Model Interpretation, Scenario Analysis and Fraud Detection. One additional area where generative AI can also greatly improve the process of predicting credit risk is portfolio optimization. According to a study by Moolchandani (2024), by simulating a variety of asset behaviors and market conditions, generative AI can help optimize portfolios. That helps financial institutions and banks to make more informed investment decisions, gain a better understanding of risk-return profiles and better manage credit portfolios. Particularly, generative AI can propose the best and optimal asset allocations throughout analyzing current and historical data. This assists in risk reduction strategy recommendation, real-time portfolio monitoring, default risk prediction and credit pricing setting (McKinsey, 2024).

In contrast, other generative AI applications focus on more specialized functions supporting the credit risk prediction process. First, data augmentation creates synthetic samples to counteract data scarcity and class imbalance to improve model robustness. Second, feature engineering generates new variables revealing hidden patterns in order to improve prediction accuracy without directly affecting asset allocations (Moolchandani, 2024). Third, anomaly detection using generative models to find outliers or unusual behavior is mainly used for fraud protection and data quality assurance. Conditional generation which easily simulates particular borrower outcomes under predetermined economic scenarios is mostly used for scenario testing rather than allocation. The next area generative AI supports in the credit risk prediction process is scenario analysis in which AI is used to stress test portfolios under simulated macroeconomic circumstances that provide information to guide but not carry out portfolio adjustments (Ajay, 2024). Lastly, fraud detection prioritizes financial security over optimization by using synthetic case generation and pattern learning to identify questionable activities (Stout, 2025). To sum up, portfolio optimization uses real-time intelligence to minimize risk across credit portfolios and maximize returns, in contrast, the other areas are supportive to enhance data quality, model performance, interpretability, and systemic risk awareness instead of directly controlling portfolio structure.

B

5. Stress testing

A. The change of PDs

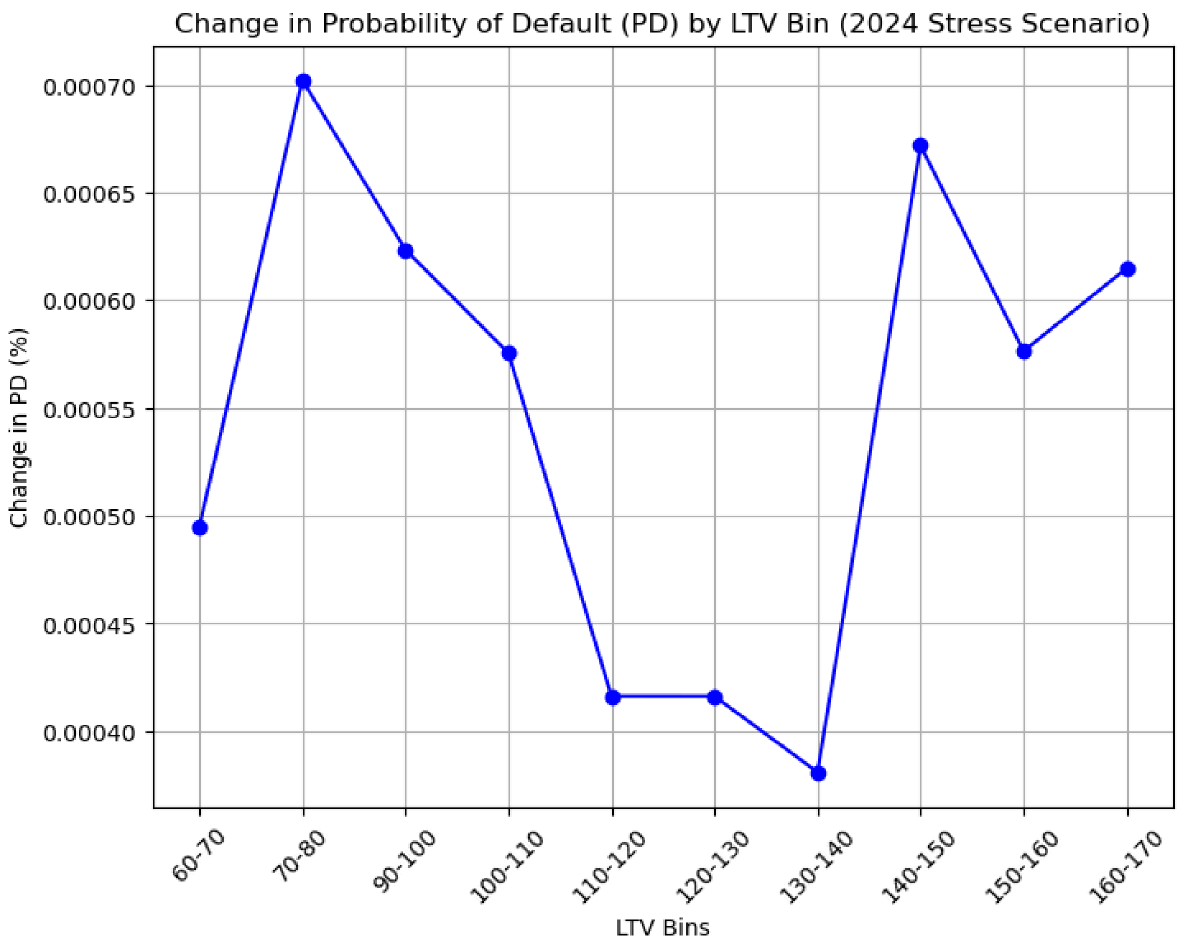
The selected scenario is 2024 with 1.8% projected income growth, combined with 10 bins of original LTV ratio.

- Baseline Period: This is the period where income growth was higher than the projected scenario (income growth of 2.5% or higher in earlier years like 2022–2023)
- Stress Test Period: This will be the 2024 scenario, where projected income growth is 1.8%, reflecting economic challenges.

In [562…

```
# Define 10 LTV bins with specified ranges
ltv_bins = [(60, 70), (70, 80), (90, 100), (100, 110), (110, 120), (120, 130), (13

# Simulate PD values before stress (Random values between 0.02 and 0.04 for each bi
np.random.seed(42)
pd_before_stress = np.random.uniform(0.02, 0.04, size=10)

# Apply 1.8% increase in PD for 2024 stress scenario
pd_after_stress = pd_before_stress * (1 + 0.018)

# Calculate the change in PD for each bin
change_in_pd = pd_after_stress - pd_before_stress

ltv_bin_labels = [f'{bin[0]}-{bin[1]}' for bin in ltv_bins]
bin_results_2024 = pd.DataFrame({
    'LTV Bin': ltv_bin_labels,
    'PD Before Stress': pd_before_stress,
    'PD After Stress': pd_after_stress,
    'Change in PD': change_in_pd
})

plt.figure(figsize=(8, 6))
plt.plot(bin_results_2024['LTV Bin'], bin_results_2024['Change in PD'], marker='o',
plt.title('Change in Probability of Default (PD) by LTV Bin (2024 Stress Scenario)'
plt.xlabel('LTV Bins')
plt.ylabel('Change in PD (%)')
```

```
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



The change in PD varies by LTV bins. The overall trend indicates greater sensitivity to economic conditions in certain LTV ranges, especially in the low and high ends.

B

The first condition refers to a borrower's inability to pay (often due to income loss). The macroeconomic factor corresponding with this hypothesis is rising unemployment. High unemployment leads to sudden income loss, the borrowers are unable to repay their loan. Borrowers facing job loss or reduced income are more likely to default if their home is also worth less than the mortgage balance (Pavan et al., 2020).

The second trigger is negative equity (when the mortgage exceeds the home's value). This is related to housing market downturn. Falling home prices reduce the value of the collateral securing mortgages. When home values drop below the outstanding mortgage, borrowers experiencing financial distress are more likely to default. This not only reduces the ability to sell or refinance, but also makes it impossible for the borrower to continue paying if they found hopeless to recover their equity.

Reference

1. Ajay, K. (2024). Leveraging Gen AI in reimagining Credit Risk Management | Risk &

Compliance Platform Europe. Risk & Compliance Platform Europe. Retrieved from: https://www.riskcompliance.biz/news/leveraging-gen-ai-in-reimagining-credit-risk-management/

2. Barasa, I., Wanyonyi, S., & Kololi, M. (2025). Application of Logistic Regression in Enhancing Digital Credit Risk Management in Commercial Banks. Asian Journal of Probability and Statistics. Retrieved from: https://doi.org/10.9734/ajpas/2025/v27i2710.

3. Stout, D. W. (2025). Generative AI for Financial Risk Prediction: Use cases. Magai. https://magai.co/generative-ai-for-financial-risk-prediction-use-cases/

4. McKinsey. (2024). Embracing generative AI in credit risk. McKinsey & Company. Retrieved from: https://www.mckinsey.com/capabilities/risk-and-resilience/our-insights/embracing-generative-ai-in-credit-risk

5. Moolchandani, S. (2024). The integration of generative AI in credit risk management. www.academia.edu. Retrieved from: https://www.academia.edu/124189188/The_Integration_of_Generative_AI_in_Credit_Risk_Manage

6. University of Technology Sydney (2025). 25751 Financial Institution Management. Lecture 1: Personal Lending.

7. University of Technology Sydney (2025). 25752 Bank Lending and Analytics. Lecture 12: Credit Control

8. Pavan, M., & Barreda-Tarrazona, I. (2020). Should I default on my mortgage even if I can pay? Experimental evidence. Journal of Economic Dynamics and Control. https://doi.org/10.1016/J.JEDC.2019.103733.