# SimpliLearn Capstone Project 1 : Real Estate

# Real Estate.

## Course-end Project 1

**DESCRIPTION**

**Problem Statement**

A banking institution requires actionable insights into mortgage-backed securities, geographic business investment, and real estate analysis. The mortgage bank would like to identify potential monthly mortgage expenses for each region based on monthly family income and rental of the real estate. A statistical model needs to be created to predict the potential demand in dollars amount of loan for each of the region in the USA. Also, there is a need to create a dashboard which would refresh periodically post data retrieval from the agencies. The dashboard must demonstrate relationships and trends for the key metrics as follows: number of loans, average rental income, monthly mortgage and owner's cost, family income vs mortgage cost comparison across different regions. The metrics described here do not limit the dashboard to these few. Dataset Description

Variables

Description Second mortgage Households with a second mortgage statistics Home equity Households with a home equity loan statistics Debt Households with any type of debt statistics Mortgage Costs Statistics regarding mortgage payments, home equity loans, utilities, and property taxes Home Owner Costs Sum of utilities, and property taxes statistics Gross Rent Contract rent plus the estimated average monthly cost of utility features High school Graduation High school graduation statistics Population Demographics Population demographics statistics Age Demographics Age demographic statistics Household Income Total income of people residing in the household Family Income Total income of people related to the householder

## Project Task: Week 1

## Data Import and Preparation:

1. Import data.
2. Figure out the primary key and look for the requirement of indexing.

3. Gauge the fill rate of the variables and devise plans for missing value treatment. Please explain explicitly the reason for the treatment chosen for each variable.
4. Perform debt analysis. You may take the following steps:

Exploratory Data Analysis (EDA):

a) Explore the top 2,500 locations where the percentage of households with a second mortgage is the highest and percent ownership is above 10 percent. Visualize using geo-map. You may keep the upper limit for the percent of households with a second mortgage to 50 percent

b) Use the following bad debt equation: Bad Debt = P (Second Mortgage ∩ Home Equity Loan) Bad Debt = second_mortgage + home_equity - home_equity_second_mortgage c) Create pie charts to show overall debt and bad debt

d) Create Box and whisker plot and analyze the distribution for 2nd mortgage, home equity, good debt, and bad debt for different cities

e) Create a collated income distribution chart for family income, house hold income, and remaining income

## Project Task: Week 2

## Exploratory Data Analysis (EDA):

1. Perform EDA and come out with insights into population density and age. You may have to derive new fields (make sure to weight averages for accurate measurements):

a) Use pop and ALand variables to create a new field called population density

b) Use male_age_median, female_age_median, male_pop, and female_pop to create a new field called median age c) Visualize the findings using appropriate chart type

2. Create bins for population into a new variable by selecting appropriate class interval so that the number of categories don't exceed 5 for the ease of analysis.

a) Analyze the married, separated, and divorced population for these population brackets

b) Visualize using appropriate chart type

3. Please detail your observations for rent as a percentage of income at an overall level, and for different states.
4. Perform correlation analysis for all the relevant variables by creating a heatmap. Describe your findings.

## Project Task: Week 3

## Data Pre-processing:

1. The economic multivariate data has a significant number of measured variables. The goal is to find where the measured variables depend on a number of smaller unobserved common factors or latent variables. 2. Each variable is assumed to be dependent upon a linear combination of the common factors, and the coefficients are known as loadings. Each measured variable also includes a component due to independent random variability, known as “specific variance” because it is specific to one variable. Obtain the common factors and then plot the loadings. Use factor analysis to find latent variables in our dataset and gain insight into the linear relationships in the data. Following are the list of latent variables:

• Highschool graduation rates

• Median population age

• Second mortgage statistics

• Percent own

• Bad debt expense

## Project Task: Week 4

## Data Modeling :

1. Build a linear Regression model to predict the total monthly expenditure for home mortgages loan. Please refer ‘deplotment_RE.xlsx’. Column hc_mortgage_mean is predicted variable. This is the mean monthly mortgage and owner costs of specified geographical location. Note: Exclude loans from prediction model which have NaN (Not a Number) values for hc_mortgage_mean.

a) Run a model at a Nation level. If the accuracy levels and R square are not satisfactory proceed to below step.

b) Run another model at State level. There are 52 states in USA.

c) Keep below considerations while building a linear regression model. Data Modeling :

• Variables should have significant impact on predicting Monthly mortgage and owner costs

• Utilize all predictor variable to start with initial hypothesis

• R square of 60 percent and above should be achieved

• Ensure Multi-collinearity does not exist in dependent variables

• Test if predicted variable is normally distributed

Data Reporting:

2. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

a) Box plot of distribution of average rent by type of place (village, urban, town, etc.).

b) Pie charts to show overall debt and bad debt.

c) Explore the top 2,500 locations where the percentage of households with a second mortgage is the highest and percent ownership is above 10 percent. Visualize using geo-map.

d) Heat map for correlation matrix.

e) Pie chart to show the population distribution across different types of places (village, urban, town etc.)

```
# connect to drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

# WEEK 1:

## 1. Import data.

```
#import both the datasets

df_train=pd.read_csv('/content/drive/MyDrive/CAPSTONE PROJECT/Project 1/train.csv')
df_test=pd.read_csv('/content/drive/MyDrive/CAPSTONE PROJECT/Project 1/test.csv')
```

```
df_train.shape
```

```
(27321, 80)
```

```
df_test.shape
```

```
(11709, 80)
```

```
#check the train dataset

df_train.head()
```

| | UID | BLOCKID | SUMLEVEL | COUNTYID | STATEID | state | state_ab | city | place |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 267822 | NaN | 140 | 53 | 36 | New York | NY | Hamilton | Hamilton |
| **1** | 246444 | NaN | 140 | 141 | 18 | Indiana | IN | South Bend | Roseland |
| **2** | 245683 | NaN | 140 | 63 | 18 | Indiana | IN | Danville | Danville |
| **3** | 279653 | NaN | 140 | 127 | 72 | Puerto Rico | PR | San Juan | Guaynabo |
| **4** | 247218 | NaN | 140 | 161 | 20 | Kansas | KS | Manhattan | Manhattan City |

5 rows × 80 columns

```
#check for null values for train data

df_train.isnull().sum()
```

```
UID               0
BLOCKID       27321
SUMLEVEL          0
COUNTYID          0
STATEID           0
              ...
pct_own         268
married         191
married_snp     191
separated       191
divorced        191
Length: 80, dtype: int64
```

```
#check the test dataset

df_test.head()
```

| | UID | BLOCKID | SUMLEVEL | COUNTYID | STATEID | state | state_ab | city | pla |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 255504 | NaN | 140 | 163 | 26 | Michigan | MI | Detroit | Dearbo Heigl C |
| **1** | 252676 | NaN | 140 | 1 | 23 | Maine | ME | Auburn | Aubu C |
| **2** | 276314 | NaN | 140 | 15 | 42 | Pennsylvania | PA | Pine City | Millert |
| **3** | 248614 | NaN | 140 | 231 | 21 | Kentucky | KY | Monticello | Montice C |
| **4** | 286865 | NaN | 140 | 355 | 48 | Texas | TX | Corpus Christi | Edı |

5 rows × 80 columns

```
#check the test data info and null values

df_test.info()
```

```
 24  rent_samples                11561 non-null  float64
 25  rent_gt_10                  11560 non-null  float64
```

```
 55  home_equity_second_mortgage  11489 non-null  float64
 56  second_mortgage              11489 non-null  float64
 57  home_equity                  11489 non-null  float64
 58  debt                         11489 non-null  float64
 59  second_mortgage_cdf          11489 non-null  float64
 60  home_equity_cdf              11489 non-null  float64
 61  debt_cdf                     11489 non-null  float64
 62  hs_degree                    11624 non-null  float64
 63  hs_degree_male               11620 non-null  float64
 64  hs_degree_female             11604 non-null  float64
 65  male_age_mean                11625 non-null  float64
 66  male_age_median              11625 non-null  float64
 67  male_age_stdev               11625 non-null  float64
 68  male_age_sample_weight       11625 non-null  float64
 69  male_age_samples             11625 non-null  float64
 70  female_age_mean              11613 non-null  float64
 71  female_age_median            11613 non-null  float64
 72  female_age_stdev             11613 non-null  float64
 73  female_age_sample_weight     11613 non-null  float64
 74  female_age_samples           11613 non-null  float64
 75  pct_own                      11587 non-null  float64
 76  married                      11625 non-null  float64
 77  married_snp                  11625 non-null  float64
 78  separated                    11625 non-null  float64
 79  divorced                     11625 non-null  float64
dtypes: float64(61), int64(13), object(6)
memory usage: 7.1+ MB
```

```
df_test.isnull().sum()
```

```
UID               0
BLOCKID       11709
SUMLEVEL          0
COUNTYID          0
STATEID           0
                ...
pct_own         122
married          84
married_snp      84
separated        84
divorced         84
Length: 80, dtype: int64
```

```
#total rows in train data

len(df_train)
```

```
27321
```

```
#total null values in column "BLOCKEDID" from train data

df_train['BLOCKID'].isnull().sum()
```

```
27321
```

```
#total rows in test data

len(df_test)
```

```
11709
```

```
#total null values in column "BLOCKEDID" from train data

df_test['BLOCKID'].isnull().sum()
```

```
11709
```

From above two cells it is clear that

All the rows are null from BlockedId deom both the dataset

So we can remove that column

```
# delete BLOCKEDID column from train data

df_train=df_train.drop('BLOCKID',axis=1)
```

```
# delete BLOCKEDID column from test data

df_test=df_test.drop('BLOCKID',axis=1)
```

## 2. Figure out the primary key and look for the requirement of indexing.

```
#caculating unique value fron train dataset

df_train.nunique()
```

```
UID            27161
SUMLEVEL           1
COUNTYID         296
STATEID           52
state             52
                 ...
pct_own        22302
married        20282
married_snp    10350
separated       6190
```

```
divorced       13688
Length: 79, dtype: int64
```

```
#caculating unique value fron test dataset
```

```
df_test.nunique()
```

```
UID            11677
SUMLEVEL           1
COUNTYID         246
STATEID           52
state             52
                 ...
pct_own        10578
married        10215
married_snp     6829
separated       4512
divorced        8273
Length: 79, dtype: int64
```

```
# Checking duplicate values From Train dataset
```

```
duplicate_train = df_train[df_train.duplicated()]
```

```
duplicate_train
```

| | UID | SUMLEVEL | COUNTYID | STATEID | state | state_ab | city | place |
|---|---|---|---|---|---|---|---|---|
| **1623** | 230058 | 140 | 73 | 6 | California | CA | Oceanside | Camp Pendleton North |
| **1907** | 292484 | 140 | 25 | 55 | Wisconsin | WI | Madison | Madison City |
| **2447** | 268401 | 140 | 61 | 36 | New York | NY | Long Island City | New York City |

```
# Checking duplicate values From Test dataset

duplicate_test = df_test[df_test.duplicated()]
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **5066** | 274254 | 140 | 109 | 40 | Oklahoma | OK | [illegible] | [illegible] |

```
duplicate_test
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **4225** | 266664 | 140 | 27 | 36 | New York | NY | Poughquag | H |
| **5278** | 278468 | 140 | 101 | 42 | Pennsylvania | PA | Philadelphia | |
| **5933** | 231120 | 140 | 85 | 6 | California | CA | Palo Alto | |
| **6288** | 279063 | 140 | 23 | 72 | Puerto Rico | PR | Cabo Rojo | P |
| **6675** | 267259 | 140 | 47 | 36 | New York | NY | Brooklyn | New |
| **7010** | 281345 | 140 | 99 | 46 | South Dakota | SD | Sioux Falls | A |
| **7186** | 278597 | 140 | 119 | 42 | Pennsylvania | PA | Lewisburg | L |
| **7500** | 253706 | 140 | 73 | 26 | Michigan | MI | Mount Pleasant | Plea |
| **8858** | 284740 | 140 | 141 | 48 | Texas | TX | El Paso | F |
| **9006** | 268339 | 140 | 61 | 36 | New York | NY | New York | New |
| **9137** | 268730 | 140 | 65 | 36 | New York | NY | Rome | R |
| **9231** | 265330 | 140 | 3 | 32 | Nevada | NV | Las Vegas | Wi |
| **9287** | 251576 | 140 | 5 | 24 | Maryland | MD | Baltimore | |
| **9396** | 238950 | 140 | 53 | 13 | Georgia | GA | Cusseta | C chatta |
| **9406** | 279895 | 140 | 5 | 44 | Rhode Island | RI | Middletown | |
| **9538** | 225615 | 140 | 37 | 6 | California | CA | Los Angeles | Ver |
| **9770** | 227404 | 140 | 37 | 6 | California | CA | Los Angeles | El S |
| **9801** | 283415 | 140 | 29 | 48 | Texas | TX | San Antonio | Lack |
| **10395** | 240129 | 140 | 179 | 13 | Georgia | GA | Fort Stewart | Fort |

From above visualization its clear that

there are 32 rows duplicate in test data

and 160 rows in train data

as the data is same for whole row

we will ignore this value and set UID as primary key

Salt Lake Salt L

```
#deleting duplicate values from train dataset

df_train.drop_duplicates(keep='first',inplace=True)
```

```
#deleting duplicate values from train dataset

df_test.drop_duplicates(keep='first',inplace=True)


# setting UID as index column in Train dataset
# and using key word setting it as primary key

df_train.set_index(keys='UID',inplace=True)


df_train.head()
```

| UID | SUMLEVEL | COUNTYID | STATEID | state | state_ab | city | place | type | prima |
|---|---|---|---|---|---|---|---|---|---|
| **267822** | 140 | 53 | 36 | New York | NY | Hamilton | Hamilton | City | tr |
| **246444** | 140 | 141 | 18 | Indiana | IN | South Bend | Roseland | City | tr |
| **245683** | 140 | 63 | 18 | Indiana | IN | Danville | Danville | City | tr |
| **279653** | 140 | 127 | 72 | Puerto Rico | PR | San Juan | Guaynabo | Urban | tr |
| **247218** | 140 | 161 | 20 | Kansas | KS | Manhattan | Manhattan City | City | tr |

5 rows × 78 columns

```
# setting UID as index column in Test dataset
# and using key word setting it as primary key

df_test.set_index(keys='UID',inplace=True)


df_test.head()
```

| UID | SUMLEVEL | COUNTYID | STATEID | state | state_ab | city | place | type |
|---|---|---|---|---|---|---|---|---|
| **255504** | 140 | 163 | 26 | Michigan | MI | Detroit | Dearborn Heights City | CDP |
| **252676** | 140 | 1 | 23 | Maine | ME | Auburn | Auburn City | City |
| **276314** | 140 | 15 | 42 | Pennsylvania | PA | Pine City | Millerton | Borough |
| **248614** | 140 | 231 | 21 | Kentucky | KY | Monticello | Monticello | City |

## 3. Gauge the fill rate of the variables and devise plans for missing value treatment. Please explain explicitly the reason for the treatment chosen for each variable.

```
#calculating percentage of missing values from Train dataset


missing_value_train=df_train.isnull().sum()*100/len(df_train)
df_missing_value_train=pd.DataFrame(missing_value_train,columns=['Percentage of missing value

df_missing_value_train
```

```
#sorting in descending order to check highest missing values

df_missing_value_train.sort_values(by=['Percentage of missing value'],ascending=False)
```

| | Percentage of missing value |
|---|---|
| **hc_stdev** | 1.759876 |
| **hc_mean** | 1.759876 |
| **hc_sample_weight** | 1.759876 |
| **hc_samples** | 1.759876 |
| **hc_median** | 1.759876 |
| **...** | ... |
| **pop** | 0.000000 |
| **male_pop** | 0.000000 |
| **female_pop** | 0.000000 |
| **universe_samples** | 0.000000 |
| **SUMLEVEL** | 0.000000 |

78 rows × 1 columns

```
#calculating percentage of missing values from Test dataset


missing_value_test=df_test.isnull().sum()*100/len(df_test)
df_missing_value_test=pd.DataFrame(missing_value_test,columns=['Percentage of missing value']


df_missing_value_test
```

| | Percentage of missing value |
|---|---|
| **SUMLEVEL** | 0.000000 |
| **COUNTYID** | 0.000000 |
| **STATEID** | 0.000000 |
| **state** | 0.000000 |

```
#sorting in descending order to check highest missing values

df_missing_value_test.sort_values(by=['Percentage of missing value'],ascending=False)
```

| | Percentage of missing value |
|---|---|
| **hc_sample_weight** | 2.286546 |
| **hc_samples** | 2.286546 |
| **hc_stdev** | 2.286546 |
| **hc_median** | 2.286546 |
| **hc_mean** | 2.286546 |
| **...** | ... |
| **pop** | 0.000000 |
| **male_pop** | 0.000000 |
| **female_pop** | 0.000000 |
| **universe_samples** | 0.000000 |
| **SUMLEVEL** | 0.000000 |

78 rows × 1 columns

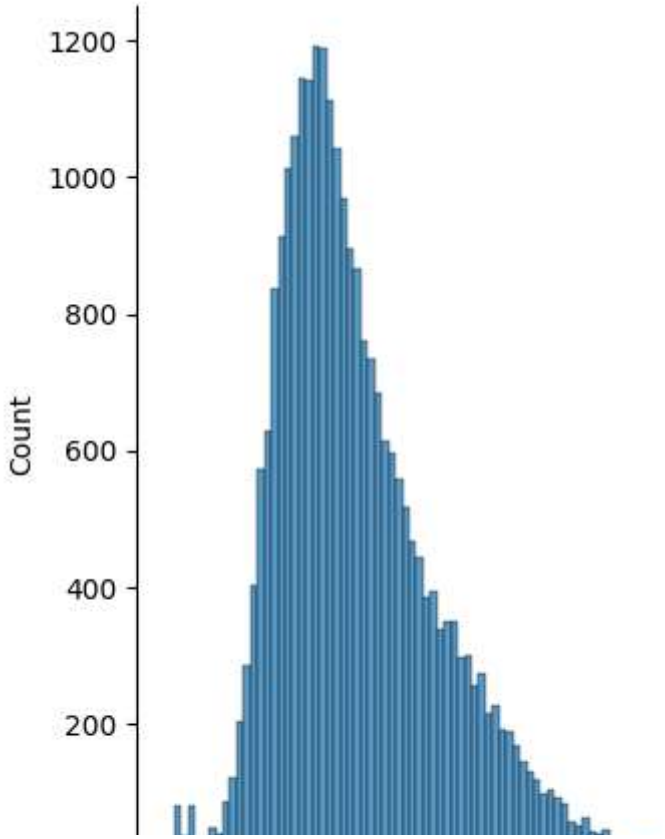
- As per above observation we found very less missing values in both the dataset

  So

  we will check the data for skewness

```
sns.displot(df_train['hc_stdev'])
```

```
<seaborn.axisgrid.FacetGrid at 0x7f953176c6d0>
```




- since we have skewness, we will use median to fill the missing values

```
#filling missing data with median in train dataset

df_train.fillna(df_train.median(),inplace=True)
```

```
<ipython-input-36-9b905a3ae00f>:3: FutureWarning: The default value of numeric_only in D
  df_train.fillna(df_train.median(),inplace=True)
```

```
#filling missing data with median in test dataset

df_test.fillna(df_test.median(),inplace=True)
```

```
<ipython-input-37-8d3c22968141>:3: FutureWarning: The default value of numeric_only in D
  df_test.fillna(df_test.median(),inplace=True)
```

# Exploratory Data Analysis (EDA):

## 4. Perform debt analysis. You may take the following steps:

## a) Explore the top 2,500 locations where the percentage of households with a second mortgage is the highest and percent ownership is above 10 percent. Visualize using geo-map. You may keep the upper limit for the percent of households with a second mortgage to 50 percent

## b) Use the following bad debt equation: Bad Debt = P (Second Mortgage ∩ Home Equity Loan) Bad Debt = second_mortgage + home_equity - home_equity_second_mortgage

```
#calculating given formula for train Dataset

df_train['Bad_Debt']=df_train['second_mortgage']+df_train['home_equity']-df_train['home_equit


#checking values

df_train['Bad_Debt']
```

```
     UID
     267822    0.09408
     246444    0.04274
     245683    0.09512
     279653    0.01086
     247218    0.05426
               ...
     279212    0.00000
     277856    0.20908
     233000    0.07857
     287425    0.14305
     265371    0.18362
     Name: Bad_Debt, Length: 27161, dtype: float64
```

```
#calculating given formula for test Dataset

df_test['Bad_Debt']=df_test['second_mortgage']+df_test['home_equity']-df_test['home_equity_se


#checking values

df_test['Bad_Debt']
```

```
     UID
     255504    0.07651
     252676    0.14375
     276314    0.06744
     248614    0.01741
     286865    0.03440
```

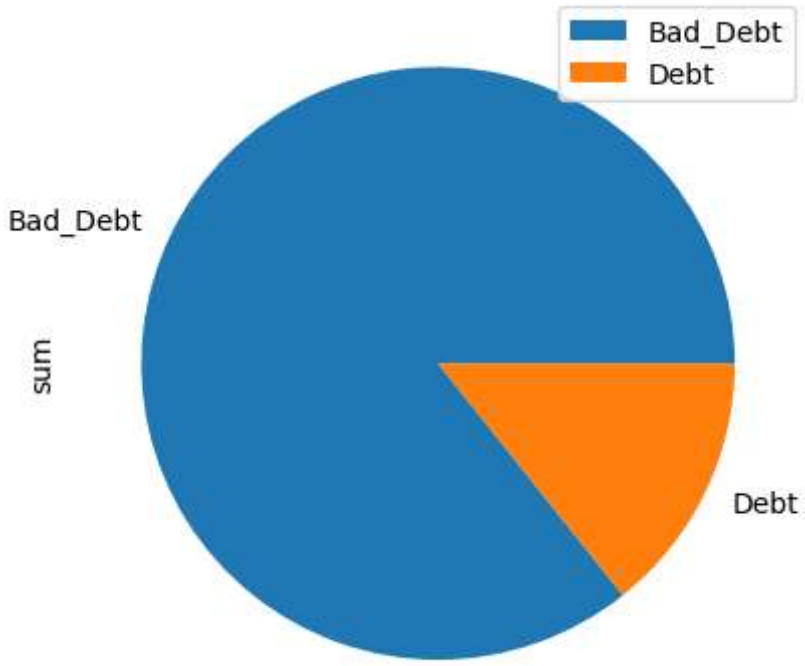
```
            ...
238088    0.05620
242811    0.08182
250127    0.13545
241096    0.07967
287763    0.05042
Name: Bad_Debt, Length: 11677, dtype: float64
```

## c) Create pie charts to show overall debt and bad debt

```
#import required libraries

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

```
# creating dataframe for debt and Bad debt

dataframe = pd.DataFrame({'Name': ['Debt', 'Bad_Debt'],
                          'sum': [df_train['Bad_Debt'].sum(), df_train['debt'].sum()]})

# Plotting the pie chart for above dataframe
dataframe.groupby(['Name']).sum().plot(kind='pie', y='sum')
```

```
<Axes: ylabel='sum'>
```



## d) Create Box and whisker plot and analyze the distribution for 2nd mortgage, home equity, good debt, and bad debt for different cities

```
# create lit of given column

colmn=['second_mortgage','home_equity','debt','Bad_Debt']


# take any two cities to show plot
# we will take Hamilton and Danville
# and create seprate df for each city

df_Danville=df_train.loc[df_train['city']=='Danville']
df_Hamilton=df_train.loc[df_train['city']=='Hamilton']


# now merge both the dataset

df_city=pd.concat([df_Danville,df_Hamilton])


# Now check the dataset

df_city.head(3)
```

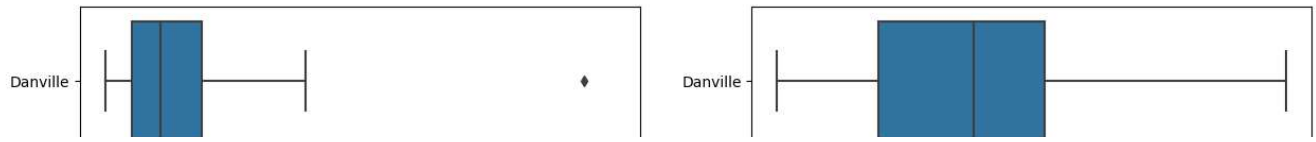
| UID | SUMLEVEL | COUNTYID | STATEID | state | state_ab | city | place | type | primary |
|---|---|---|---|---|---|---|---|---|---|
| **245683** | 140 | 63 | 18 | Indiana | IN | Danville | Danville | City | tract |
| **247609** | 140 | 21 | 21 | Kentucky | KY | Danville | Danville City | City | tract |
| **290135** | 140 | 590 | 51 | Virginia | VA | Danville | Danville City | Town | tract |

3 rows × 79 columns

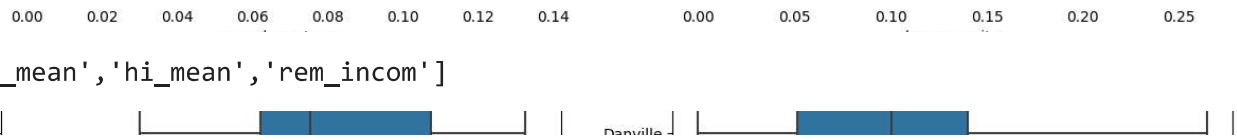
```
#check total cities value

df_city['city'].value_counts()
```

```
Danville    28
Hamilton    19
Name: city, dtype: int64
```

```
# create box plot using subplots of all given columns = colmn=['second_mortgage','home_equity

plt.figure(figsize=(15,8))
plt.subplot(2, 2, 1)
sns.boxplot(data=df_city,x='second_mortgage', y='city')
plt.subplot(2, 2, 2)
sns.boxplot(data=df_city,x='home_equity', y='city')
plt.subplot(2, 2, 3)
sns.boxplot(data=df_city,x='debt', y='city')
plt.subplot(2, 2, 4)
sns.boxplot(data=df_city,x='Bad_Debt', y='city')
plt.show()
```




## e) Create a collated income distribution chart for family income, house hold income, and remaining income

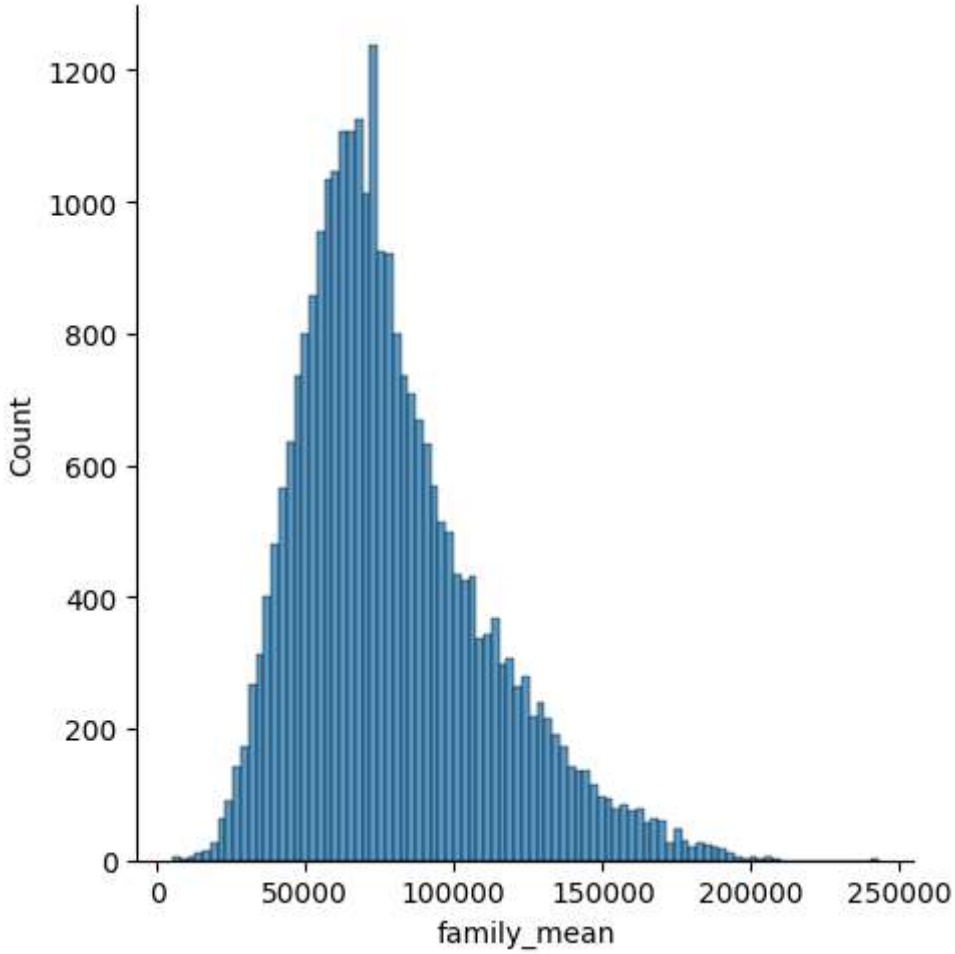



```
col=['family_mean','hi_mean','rem_incom']
```



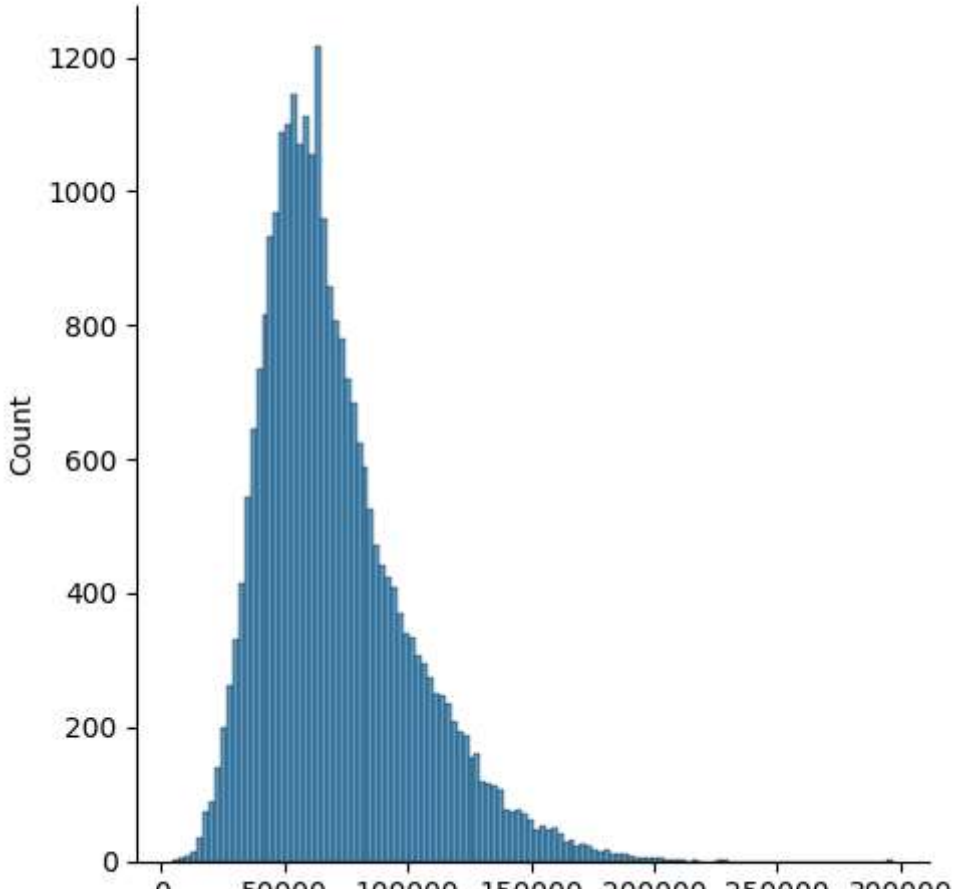
```
#CREATING DISPLOT OF FAMILY INCOME

sns.displot(df_train, x="family_mean")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f952ec0a6a0>
```




```
#CREATING DISPLOT OF houssehold INCOME

sns.displot(df_train, x="hi_mean")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f952e998280>
```




```
#CREATING DISPLOT OF remaining Income

sns.displot(df_train, x=df_train['family_mean']-df_train['hi_mean'])
```

```
<seaborn.axisgrid.FacetGrid at 0x7f952e76ef70>
```


# WEEK 2:


## Exploratory Data Analysis (EDA):

### 1. Perform EDA and come out with insights into population density and age. You may have to derive new fields (make sure to weight averages for accurate measurements):

### a) Use pop and ALand variables to create a new field called population density

### b) Use male_age_median, female_age_median, male_pop, and female_pop to create a new field called median age

### c) Visualize the findings using appropriate chart type

```
#calculating pop density for train data

df_train['pop_dens']=df_train['pop']/df_train['ALand']


#calculating pop density for test data

df_test['pop_dens']=df_test['pop']/df_test['ALand']


#calculating madian age
#train data

df_train['median_age'] = (df_train['male_age_median']*df_train['male_pop'] + df_train['female
```

```
#checking result

df_train['median_age']
```

```
UID
267822    44.667430
246444    34.722748
```

```
245683    41.774472
279653    49.879012
247218    21.965629
            ...
279212    40.904894
277856    39.160488
233000    44.089311
287425    45.029280
265371    31.132312
Name: median_age, Length: 27161, dtype: float64
```

```
#calculating madian age
#train data

df_test['median_age'] = (df_test['male_age_median']*df_test['male_pop'] + df_test['female_age
```

```
#checking result

df_test['median_age']
```

```
UID
255504    31.189053
252676    46.382991
276314    43.147420
248614    45.155104
286865    43.235983
            ...
238088    57.620624
242811    31.159118
250127    39.323630
241096    44.528597
287763    35.207171
Name: median_age, Length: 11677, dtype: float64
```

```
#check for null values
#train data
df_train['median_age'].isnull().sum()
```

```
142
```

```
#check for null values
#test data

df_test['median_age'].isnull().sum()
```

```
74
```

```
#treat null values with median

df_train['median_age'].fillna(df_train['median_age'].median(),inplace=True)
df_test['median_age'].fillna(df_test['median_age'].median(),inplace=True)
```

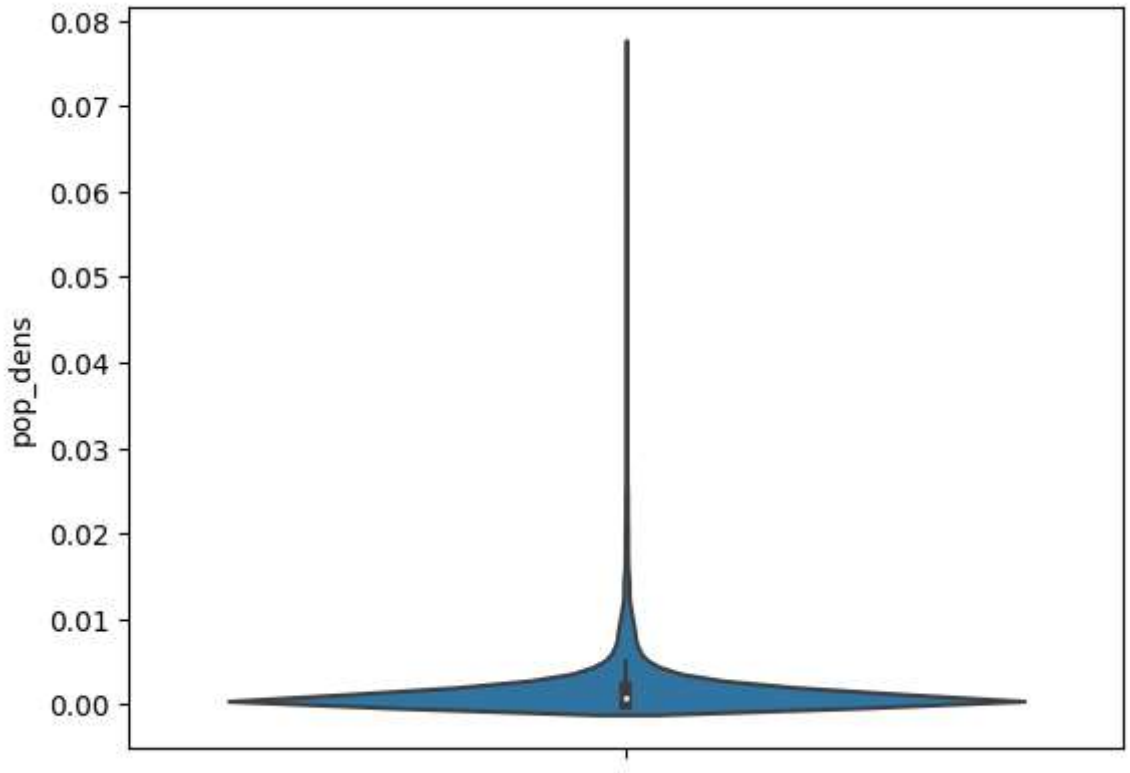
```
df_test['median_age'].fillna(df_test['median_age'].median(),inplace=True)
```

```
#plotting 4 different plots
```

```
#plot no 1
```

```
sns.violinplot(data=df_train,y='pop_dens')
```

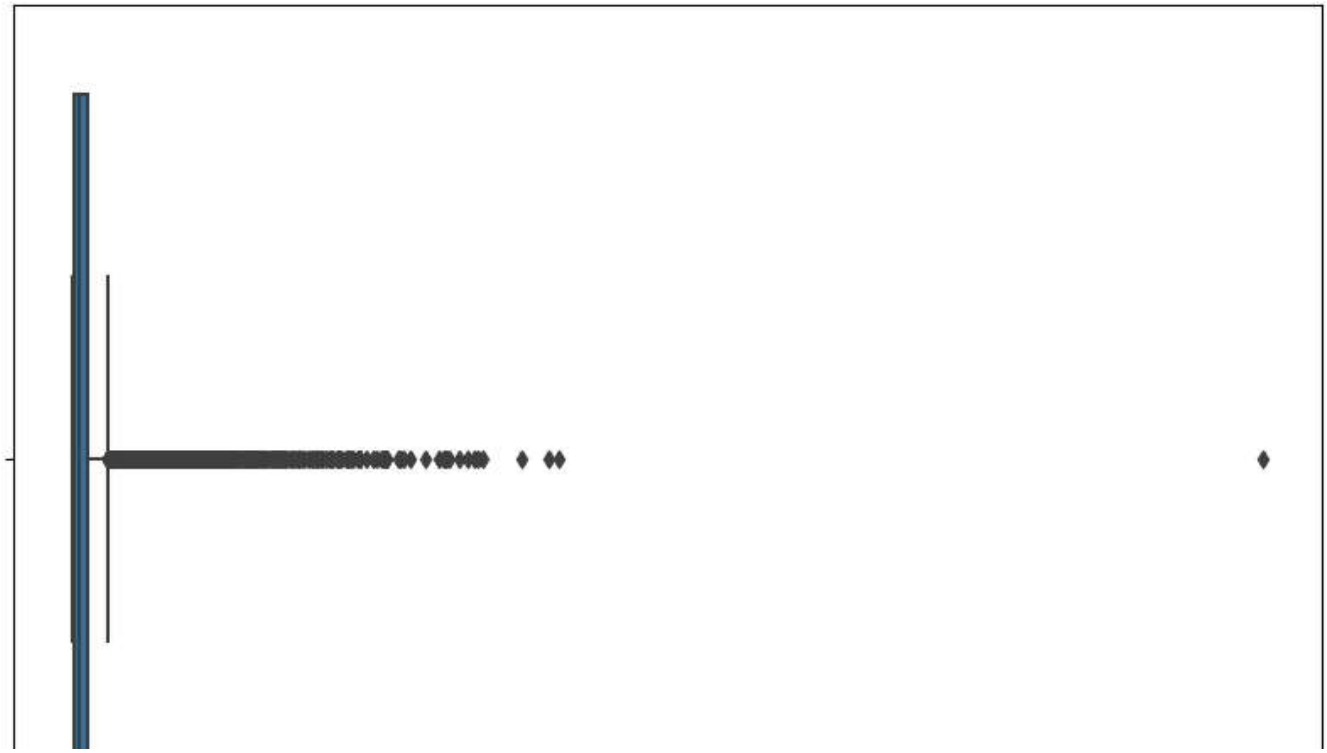
```
<Axes: ylabel='pop_dens'>
```




```
#plot no 2
```

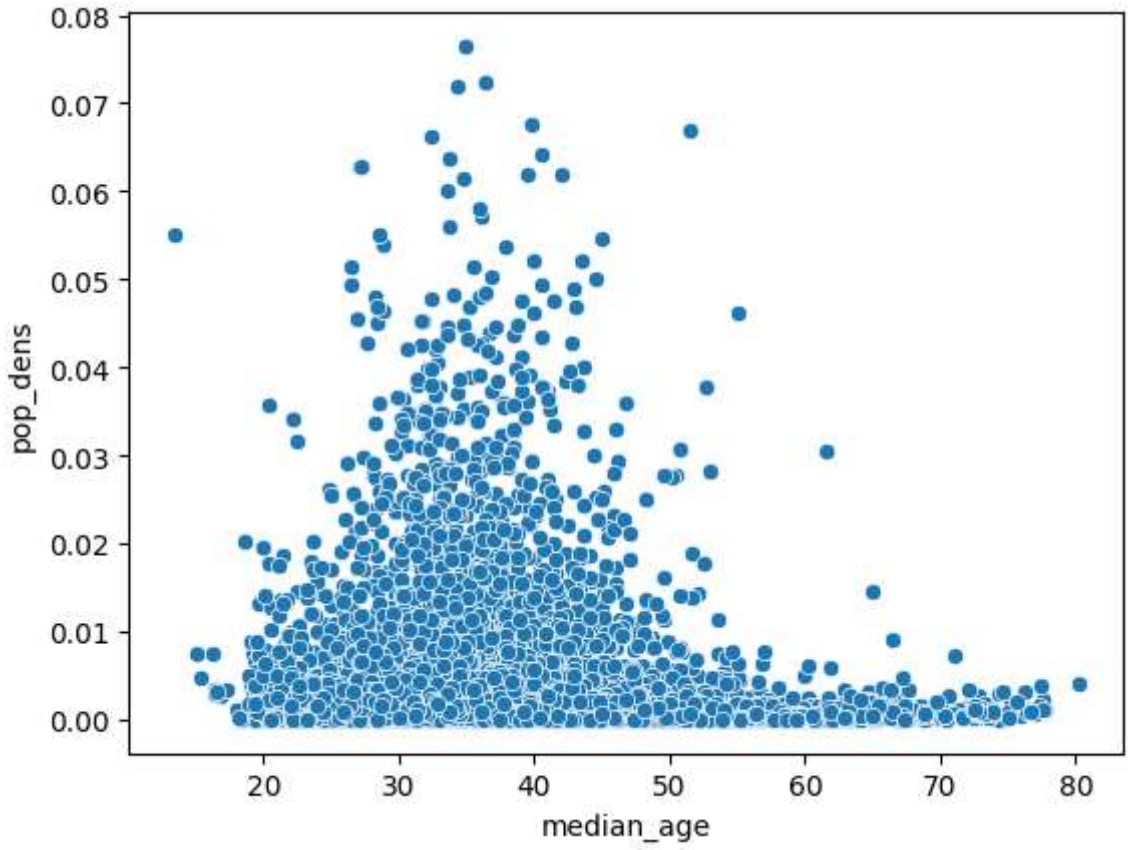
```
plt.figure(figsize=(10,7))
sns.boxplot(data=df_test,x='pop_dens')
```

```
<Axes: xlabel='pop_dens'>
```



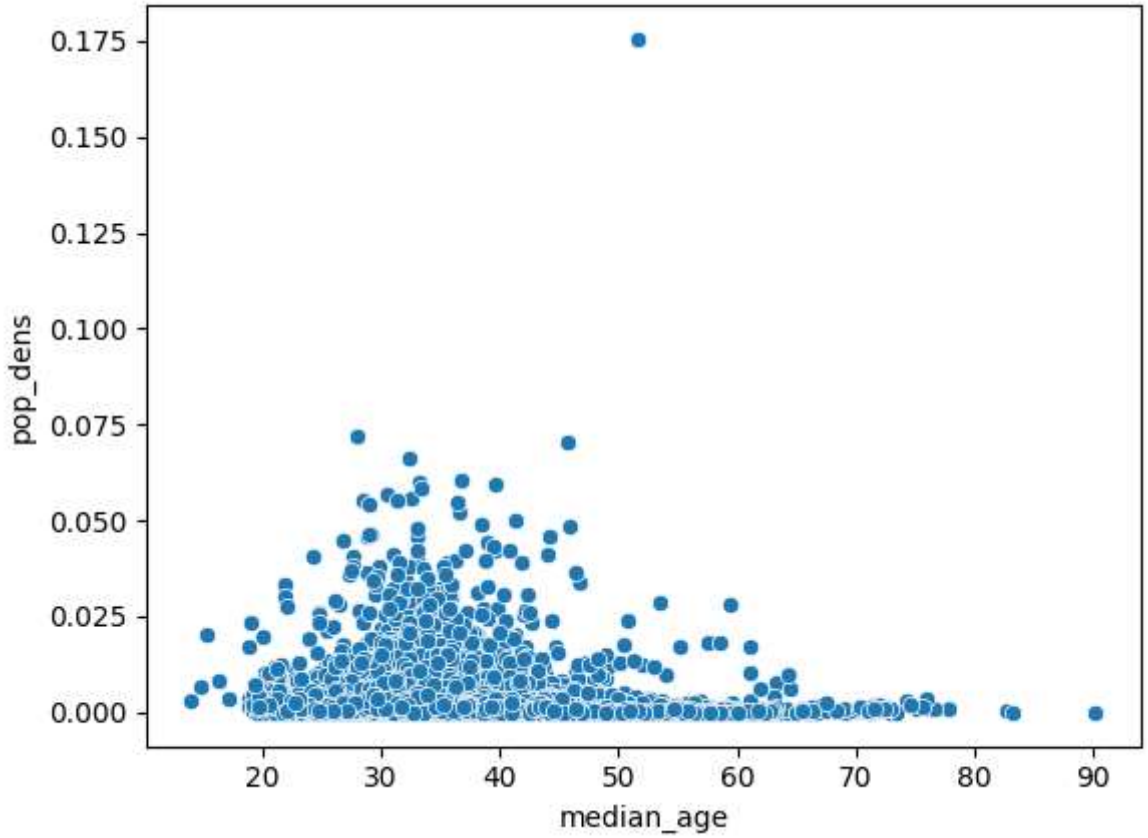
```
#plot 3


sns.scatterplot(data=df_train, x='median_age',y='pop_dens')
```

```
<Axes: xlabel='median_age', ylabel='pop_dens'>
```



```
#plot 4


sns.scatterplot(data=df_test, x='median_age',y='pop_dens')
```

```
<Axes: xlabel='median_age', ylabel='pop_dens'>
```




## 2. Create bins for population into a new variable by selecting appropriate class interval so that the number of categories don't exceed 5 for the ease of analysis.

### a) Analyze the married, separated, and divorced population for these population brackets

### b) Visualize using appropriate chart type

```
#creating 5 bins and naming it
#train data

df_train['pop_bin']=pd.cut(df_train['pop'],bins=5,labels=['very low pop','low pop','medium po
```

```
#checking result

df_train['pop_bin'].value_counts()
```

```
very low pop     26901
low pop            246
medium pop           9
high pop             4
very high pop        1
Name: pop_bin, dtype: int64
```

```
#creating 5 bins and naming it
#train data

df_test['pop_bin']=pd.cut(df_test['pop'],bins=5,labels=['very low pop','low pop','medium pop'
```

```
#checking result

df_test['pop_bin'].value_counts()
```

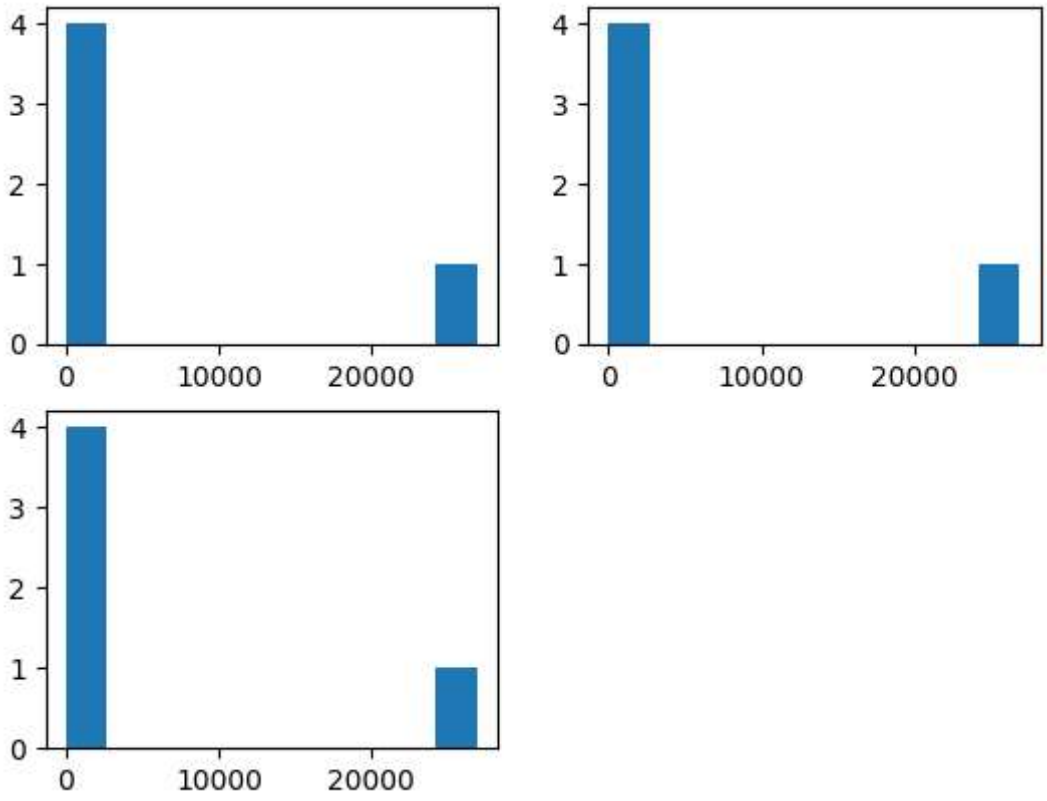
```
very low pop     11101
low pop            552
medium pop          22
high pop             1
very high pop        1
Name: pop_bin, dtype: int64
```

```
#checking ['married','separated','divorced'] data wrt bins
#train data

df_train.groupby(by='pop_bin')[['married','separated','divorced']].count()
```

| pop_bin | married | separated | divorced |
|---|---|---|---|
| **very low pop** | 26901 | 26901 | 26901 |
| **low pop** | 246 | 246 | 246 |
| **medium pop** | 9 | 9 | 9 |
| **high pop** | 4 | 4 | 4 |
| **very high pop** | 1 | 1 | 1 |

```
#saving result in another variable

df_data=df_train.groupby(by='pop_bin')[['married','separated','divorced']].count()
```

```
#ploting 3 different plots

plt.subplot(2,2,1)
plt.hist(data=df_data, x='married')
plt.subplot(2,2,2)
plt.hist(data=df_data, x='separated')
plt.subplot(2,2,3)
plt.hist(data=df_data, x='divorced')
plt.show()
```




## 3. Please detail your observations for rent as a percentage of income at an overall level, and for different states.

```
#calculating overall income
#train data

df_train['overall_income']=df_train['family_mean']+df_train['hi_mean']


#calculating overall income
#test data

df_test['overall_income']=df_test['family_mean']+df_test['hi_mean']


#checking result
```

```
df_train['overall_income']
```

```
UID
267822    131119.43196
246444     92602.02930
245683    180205.19748
279653    105135.35249
247218     85887.57862
              ...
279212     39404.81638
277856    238785.15150
233000    168768.82147
287425    332659.10880
265371    106534.26530
Name: overall_income, Length: 27161, dtype: float64
```

```
#setting result upto 2 decimal point

pd.set_option('display.precision',2)


#checking again

df_train['overall_income']
```

```
UID
267822    131119.43
246444     92602.03
245683    180205.20
279653    105135.35
247218     85887.58
              ...
279212     39404.82
277856    238785.15
233000    168768.82
287425    332659.11
265371    106534.27
Name: overall_income, Length: 27161, dtype: float64
```

```
#calculting rent percentage
#train data

df_train['rent_percent']=df_train['rent_mean']*100/df_train['overall_income']


#checking result

df_train['rent_percent']
```

```
UID
267822    0.59
246444    0.87
245683    0.41
```

```
279653    0.76
247218    1.09
          ...
279212    1.12
277856    0.76
233000    0.50
287425    0.59
265371    0.89
Name: rent_percent, Length: 27161, dtype: float64
```

```
#calculting rent percentage
#test data

df_test['rent_percent']=df_test['rent_mean']*100/df_test['overall_income']


#checking result

df_test['rent_percent']
```

```
UID
255504    0.84
252676    0.53
276314    0.66
248614    0.51
286865    0.43
          ...
238088    1.14
242811    0.94
250127    0.57
241096    0.52
287763    0.84
Name: rent_percent, Length: 11677, dtype: float64
```

```
#checking percentage with group by state , so taking aggrecated mean of percentage
#train data

df_train.groupby(by='state')['rent_percent'].agg('mean')
```

```
state
Alabama                0.65
Alaska                 0.69
Arizona                0.82
Arkansas               0.63
California             0.92
Colorado               0.75
Connecticut            0.71
Delaware               0.71
District of Columbia   0.77
Florida                0.89
Georgia                0.75
Hawaii                 0.95
Idaho                  0.65
```

```
Illinois                0.72
Indiana                 0.69
Iowa                    0.56
Kansas                  0.62
Kentucky                0.62
Louisiana               0.72
Maine                   0.64
Maryland                0.76
Massachusetts           0.69
Michigan                0.73
Minnesota               0.61
Mississippi             0.70
Missouri                0.66
Montana                 0.59
Nebraska                0.61
Nevada                  0.83
New Hampshire           0.66
New Jersey              0.77
New Mexico              0.69
New York                0.80
North Carolina          0.69
North Dakota            0.51
Ohio                    0.66
Oklahoma                0.66
Oregon                  0.74
Pennsylvania            0.68
Puerto Rico             0.87
Rhode Island            0.69
South Carolina          0.71
South Dakota            0.51
Tennessee               0.69
Texas                   0.72
Utah                    0.71
Vermont                 0.65
Virginia                0.75
Washington              0.73
West Virginia           0.58
Wisconsin               0.65
Wyoming                 0.58
Name: rent_percent, dtype: float64
```

## 4. Perform correlation analysis for all the relevant variables by creating a heatmap. Describe your findings.

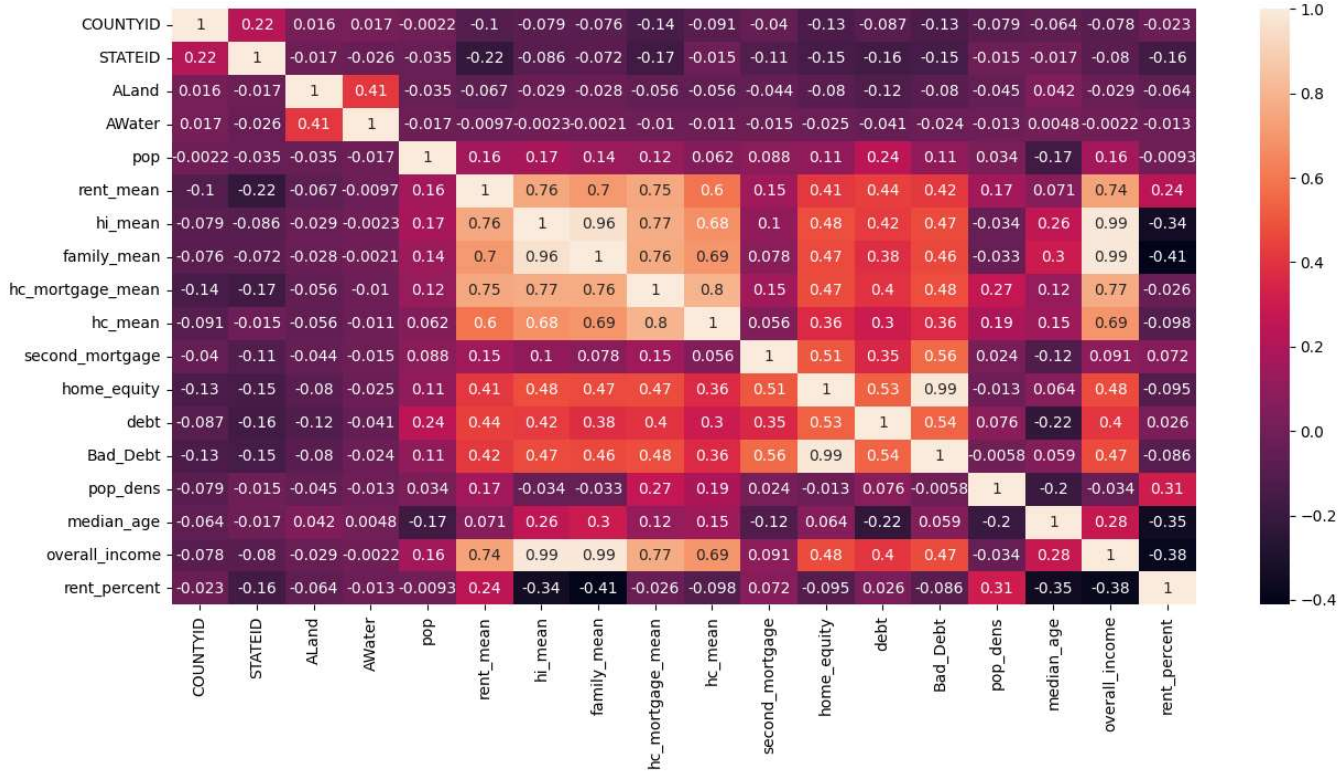
```
# check columns present in train data

df_train.columns
```

```
Index(['SUMLEVEL', 'COUNTYID', 'STATEID', 'state', 'state_ab', 'city', 'place',
       'type', 'primary', 'zip_code', 'area_code', 'lat', 'lng', 'ALand',
       'AWater', 'pop', 'male_pop', 'female_pop', 'rent_mean', 'rent_median',
       'rent_stdev', 'rent_sample_weight', 'rent_samples', 'rent_gt_10',
       'rent_gt_15', 'rent_gt_20', 'rent_gt_25', 'rent_gt_30', 'rent_gt_35',
```

```
       'rent_gt_40', 'rent_gt_50', 'universe_samples', 'used_samples',
       'hi_mean', 'hi_median', 'hi_stdev', 'hi_sample_weight', 'hi_samples',
       'family_mean', 'family_median', 'family_stdev', 'family_sample_weight',
       'family_samples', 'hc_mortgage_mean', 'hc_mortgage_median',
       'hc_mortgage_stdev', 'hc_mortgage_sample_weight', 'hc_mortgage_samples',
       'hc_mean', 'hc_median', 'hc_stdev', 'hc_samples', 'hc_sample_weight',
       'home_equity_second_mortgage', 'second_mortgage', 'home_equity', 'debt',
       'second_mortgage_cdf', 'home_equity_cdf', 'debt_cdf', 'hs_degree',
       'hs_degree_male', 'hs_degree_female', 'male_age_mean',
       'male_age_median', 'male_age_stdev', 'male_age_sample_weight',
       'male_age_samples', 'female_age_mean', 'female_age_median',
       'female_age_stdev', 'female_age_sample_weight', 'female_age_samples',
       'pct_own', 'married', 'married_snp', 'separated', 'divorced',
       'Bad_Debt', 'pop_dens', 'median_age', 'pop_bin', 'overall_income',
       'rent_percent'],
      dtype='object')


# check columns present in test data

df_test.columns

     Index(['SUMLEVEL', 'COUNTYID', 'STATEID', 'state', 'state_ab', 'city', 'place',
            'type', 'primary', 'zip_code', 'area_code', 'lat', 'lng', 'ALand',
            'AWater', 'pop', 'male_pop', 'female_pop', 'rent_mean', 'rent_median',
            'rent_stdev', 'rent_sample_weight', 'rent_samples', 'rent_gt_10',
            'rent_gt_15', 'rent_gt_20', 'rent_gt_25', 'rent_gt_30', 'rent_gt_35',
            'rent_gt_40', 'rent_gt_50', 'universe_samples', 'used_samples',
            'hi_mean', 'hi_median', 'hi_stdev', 'hi_sample_weight', 'hi_samples',
            'family_mean', 'family_median', 'family_stdev', 'family_sample_weight',
            'family_samples', 'hc_mortgage_mean', 'hc_mortgage_median',
            'hc_mortgage_stdev', 'hc_mortgage_sample_weight', 'hc_mortgage_samples',
            'hc_mean', 'hc_median', 'hc_stdev', 'hc_samples', 'hc_sample_weight',
            'home_equity_second_mortgage', 'second_mortgage', 'home_equity', 'debt',
            'second_mortgage_cdf', 'home_equity_cdf', 'debt_cdf', 'hs_degree',
            'hs_degree_male', 'hs_degree_female', 'male_age_mean',
            'male_age_median', 'male_age_stdev', 'male_age_sample_weight',
            'male_age_samples', 'female_age_mean', 'female_age_median',
            'female_age_stdev', 'female_age_sample_weight', 'female_age_samples',
            'pct_own', 'married', 'married_snp', 'separated', 'divorced',
            'Bad_Debt', 'pop_dens', 'median_age', 'pop_bin', 'overall_income',
            'rent_percent'],
           dtype='object')


# take only important or relavant columns and calulate correlation
#for train data

df_train_corr=df_train[['COUNTYID', 'STATEID','ALand',
       'AWater', 'pop','rent_mean','hi_mean','family_mean','hc_mortgage_mean','hc_mean','seco
       'rent_percent']].corr()


# take only important or relavant columns and calulate correlation
#for test data
```

```
df_test_corr=df_train[['COUNTYID', 'STATEID','ALand',
       'AWater', 'pop','rent_mean','hi_mean','family_mean','hc_mortgage_mean','hc_mean','seco
       'rent_percent']].corr()


# create heatmap for train data

plt.figure(figsize=(15,7))
sns.heatmap(df_train_corr,annot=True)
```

```
<Axes: >
```



```
# create heatmap for test data

plt.figure(figsize=(15,7))
sns.heatmap(df_test_corr,cmap="PiYG", annot=True)
```

```
<Axes: >
```


# Project Task: Week 3

## Data Pre-processing:

**1. The economic multivariate data has a significant number of measured variables. The goal is to find where the measured variables depend on a number of smaller unobserved common factors or latent variables.**

**2. Each variable is assumed to be dependent upon a linear combination of the common factors, and the coefficients are known as loadings. Each measured variable also includes a component due to independent random variability, known as "specific variance" because it is specific to one variable. Obtain the common factors and then plot the loadings. Use factor analysis to find latent variables in our dataset and gain insight into the linear relationships in the data. Following are the list of latent variables:**

- **Highschool graduation rates**
- **Median population age**
- **Second mortgage statistics**
- **Percent own**
- **Bad debt expense**

```
# Pip install factor analyser for factor analysis

!pip install factor_analyzer
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting factor_analyzer
  Downloading factor_analyzer-0.4.1.tar.gz (41 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41.8/41.8 kB 2.2 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from fac
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from fac
Collecting pre-commit
  Downloading pre_commit-3.2.2-py2.py3-none-any.whl (202 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 202.7/202.7 kB 7.5 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.9/dist-packages (f
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages (from fa
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (f
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-p
Collecting cfgv>=2.0.0
  Downloading cfgv-3.3.1-py2.py3-none-any.whl (7.3 kB)
Collecting virtualenv>=20.10.0
  Downloading virtualenv-20.21.0-py3-none-any.whl (8.7 MB)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 68.6 MB/s eta 0:00:00
Collecting nodeenv>=0.11.1
  Downloading nodeenv-1.7.0-py2.py3-none-any.whl (21 kB)
Collecting identify>=1.0.0
  Downloading identify-2.5.22-py2.py3-none-any.whl (98 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.8/98.8 kB 10.4 MB/s eta 0:00:00
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (fr
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from
Collecting distlib<1,>=0.3.6
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 468.5/468.5 kB 38.5 MB/s eta 0:00:00
Requirement already satisfied: platformdirs<4,>=2.4 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: filelock<4,>=3.4.1 in /usr/local/lib/python3.9/dist-packa
Building wheels for collected packages: factor_analyzer
  Building wheel for factor_analyzer (pyproject.toml) ... done
  Created wheel for factor_analyzer: filename=factor_analyzer-0.4.1-py2.py3-none-any.whl
  Stored in directory: /root/.cache/pip/wheels/6d/32/bd/460a71becd83f7d77152f437c2fd451f
Successfully built factor_analyzer
Installing collected packages: distlib, virtualenv, nodeenv, identify, cfgv, pre-commit,
Successfully installed cfgv-3.3.1 distlib-0.3.6 factor_analyzer-0.4.1 identify-2.5.22 no
```

```
#import factor analyzer and standard scalar

from factor_analyzer import FactorAnalyzer
from sklearn.preprocessing import StandardScaler


# Drop the column SUMLEVEL as it holds no significance
#train data

df_train= df_train.drop(columns='SUMLEVEL',axis=1)
df_train.head()
```

| UID | COUNTYID | STATEID | state | state_ab | city | place | type | primary | zip_co |
|---|---|---|---|---|---|---|---|---|---|
| **267822** | 53 | 36 | New York | NY | Hamilton | Hamilton | City | tract | 133 |

```
# Drop the column SUMLEVEL as it holds no significance
#test data

df_test= df_test.drop(columns='SUMLEVEL')
df_test.head()
```

| UID | COUNTYID | STATEID | state | state_ab | city | place | type | primary |
|---|---|---|---|---|---|---|---|---|
| **255504** | 163 | 26 | Michigan | MI | Detroit | Dearborn Heights City | CDP | tract |
| **252676** | 1 | 23 | Maine | ME | Auburn | Auburn City | City | tract |
| **276314** | 15 | 42 | Pennsylvania | PA | Pine City | Millerton | Borough | tract |
| **248614** | 231 | 21 | Kentucky | KY | Monticello | Monticello City | City | tract |
| **286865** | 355 | 48 | Texas | TX | Corpus Christi | Edroy | Town | tract |

5 rows × 83 columns

```
# we have prior knowledge that certain variables in our dataset are highly related to the lat
# In order to focus on the remaining measured variables and identify the underlying structure
# we removed the columns corresponding to the highly related variables using the drop() metho
#This allowed us to better isolate the effects of the latent variables and identify their rel

df_train_preproc = df_train.drop(['hs_degree','median_age', 'second_mortgage', 'pct_own', 'Ba
df_test_preproc = df_test.drop(['hs_degree','median_age', 'second_mortgage', 'pct_own', 'Bad_


#we only need numberic values so we are selecting only those data type

df_train_preproc=df_train_preproc.select_dtypes(include=('int','float'))
df_test_preproc=df_test_preproc.select_dtypes(include=('int','float'))
```

```
#check for missing values
#train data

df_train_preproc.isnull().sum().any()
```

```
False
```

```
#check for missing values
#test data

df_test_preproc.isnull().sum().any()
```

```
False
```

```
# Scale the data to have zero mean and unit variance
#train data

scaler = StandardScaler()
df_train_preproc_scaled = scaler.fit_transform(df_train_preproc)


# Scale the data to have zero mean and unit variance
#test data

df_test_preproc_scaled = scaler.fit_transform(df_test_preproc)


#Use the EFA (Exploratory factor analysis) method for factor analysis
#train data

n_factors=5
fa_train = FactorAnalyzer(n_factors,rotation ='varimax',method='principal')


#Use the EFA (Exploratory factor analysis) method for factor analysis
#test data

fa_test = FactorAnalyzer(n_factors,rotation ='varimax',method='principal')


#Fitting train dataset in the factor analyzer

fa_train.fit(df_train_preproc_scaled)
fa_test.fit(df_test_preproc_scaled)
```

```
FactorAnalyzer
FactorAnalyzer(method='principal', n_factors=5, rotation='varimax',
               rotation_kwargs={})
```

```
#interpret the train results

factors_train = fa_train.loadings_
factors_train
```

```
 -3.27543275e-02,  3.55899969e-02],
[ 1.85139921e-01, -5.68151500e-02, -8.11181781e-02,
  1.21383011e-01,  5.72777923e-01],
[ 9.09046267e-01,  7.63626165e-02, -1.42397243e-04,
 -2.09662111e-01, -2.57984558e-01],
[-1.46652018e-01, -9.49789502e-02, -2.76686221e-01,
  5.47111918e-01,  3.31827792e-01]])


#interpret the test results

factors_test = fa_test.loadings_
factors_test
```

```
[ 0.69476818,  0.02985963, -0.00267449,  0.14059784,  0.07708043],
[-0.32989419,  0.31415043, -0.03439165, -0.00598326,  0.67570488],
[ 0.01190232,  0.40103539, -0.13739549,  0.05803836,  0.7943229 ],
[-0.02207137,  0.06009799, -0.10902889,  0.49146801, -0.02437385],
[-0.03564451,  0.04114647, -0.10011218,  0.69407672,  0.04644313],
[-0.07765214,  0.00426697, -0.04018618,  0.81871545,  0.09280618],
[-0.12433699, -0.02046035,  0.00815302,  0.86644079,  0.13578903],
[-0.12063579, -0.02555179,  0.02765256,  0.88569834,  0.13811258],
[-0.12127907, -0.03498787,  0.0440778 ,  0.87533452,  0.13809323],
[-0.11333066, -0.04690975,  0.0525278 ,  0.84333899,  0.14345597],
[-0.10286028, -0.05419986,  0.05198579,  0.7683628 ,  0.15326231],
[-0.01012084,  0.42520578, -0.10922391,  0.05366542,  0.78908354],
[ 0.02015591,  0.40931624, -0.13298533,  0.05282115,  0.78543849],
[ 0.8948118 ,  0.0810927 , -0.05221141, -0.19961668, -0.28705755],
[ 0.84976662,  0.08171839, -0.09200142, -0.20465151, -0.3153945 ],
[ 0.8889024 ,  0.06347615,  0.08055399, -0.15010628, -0.15945115],
[-0.30764668,  0.82026658,  0.14300791,  0.05599942,  0.29191751],
[ 0.14386   ,  0.94052637,  0.07502517, -0.04666217,  0.1128192 ],
[ 0.90463907,  0.0621768 ,  0.01009702, -0.2125006 , -0.2417783 ],
[ 0.87821968,  0.05381625, -0.0048497 , -0.21359135, -0.24620381],
[ 0.82872226,  0.06337431,  0.09079917, -0.13878116, -0.10991278],
[-0.34015429,  0.85408851,  0.05063089,  0.08772226,  0.07077744],
[ 0.14792734,  0.94129716,  0.00850938, -0.02675889, -0.14918174],
[ 0.92303063, -0.02862215, -0.04716529,  0.07611448,  0.12190673],
[ 0.9093168 , -0.03715555, -0.05728808,  0.08258239,  0.12436366],
[ 0.78126086,  0.00701099,  0.10278207,  0.02479336,  0.02743174],
[-0.2171158 ,  0.74356699,  0.01841148, -0.11255043, -0.43286512],
[ 0.29616516,  0.76153064, -0.09233987, -0.05752159, -0.43792368],
[ 0.84816205, -0.05343041,  0.07750278,  0.00964797,  0.15498653],
```

```
[ 0.24914385, -0.05072875,  0.6713895 , -0.10888343, -0.43448901],
[-0.01259603,  0.07565604,  0.50305161, -0.03155574, -0.38041431],
[ 0.0660861 ,  0.84352226, -0.17489708,  0.03657774,  0.12335135],
[ 0.10351   ,  0.92669293, -0.1267735 ,  0.00785735,  0.06806036],
[ 0.15671321, -0.05781791,  0.75248021, -0.04474602, -0.32668186],
[ 0.1952718 , -0.03689248,  0.71373772, -0.06237224, -0.42351859],
[-0.06720565,  0.05478821,  0.47698729, -0.02621607, -0.27241687],
[ 0.08182227,  0.88788169, -0.14465053,  0.07539227,  0.13091448],
[ 0.118627  ,  0.95684323, -0.0857342 ,  0.04782331,  0.0645026 ],
[ 0.34824   ,  0.21893253,  0.23264862, -0.2079176 , -0.56809541],
[-0.19555321, -0.09319619, -0.00840998,  0.16255924,  0.38090182],
[-0.26352179, -0.08839383,  0.05826764,  0.13471559,  0.28392966],
[-0.4145198 , -0.06662999,  0.22158571, -0.03728646, -0.02190071],
[ 0.18415063, -0.06392691, -0.01946456,  0.1150759 ,  0.55477302],
[ 0.90802764,  0.07208657, -0.02058735, -0.20807296, -0.26634631],
[-0.16014657, -0.09960008, -0.24455571,  0.533865  ,  0.37066694]])
```

# Project Task: Week 4

## Data Modeling :

### 1. Build a linear Regression model to predict the total monthly expenditure for home mortgages loan. Please refer 'deplotment_RE.xlsx'. Column hc_mortgage_mean is predicted variable. This is the mean monthly mortgage and owner costs of specified geographical location. Note: Exclude loans from prediction model which have NaN (Not a Number) values for hc_mortgage_mean.

**a) Run a model at a Nation level. If the accuracy levels and R square are not satisfactory proceed to below step.**

**b) Run another model at State level. There are 52 states in USA.**

**c) Keep below considerations while building a linear regression model. Data Modeling :**

- **Variables should have significant impact on predicting Monthly mortgage and owner costs**
- **Utilize all predictor variable to start with initial hypothesis**
- **R square of 60 percent and above should be achieved**
- **Ensure Multi-collinearity does not exist in dependent variables**
- **Test if predicted variable is normally distributed**

```
# Select columns with categorical and object datatypes

df_cat_train=df_train.select_dtypes(include=('object','category')).columns


df_cat_train

     Index(['state', 'state_ab', 'city', 'place', 'type', 'primary', 'pop_bin'],
     dtype='object')


#import label encoder and apply to all dataset

from sklearn.preprocessing import LabelEncoder
for df_cat_train in df_train:
    encoder = LabelEncoder()
    df_train[df_cat_train] = encoder.fit_transform(df_train[df_cat_train])


# check for null values

df_train.isnull().sum().any()

     False
```

- as we can see on 'type' column is not a numeric value

  convert this to numberic data

  use label encoding

```
# select feature and target

featr=df_train.drop('hc_mortgage_mean',axis=1)
target = df_train['hc_mortgage_mean']


# Calculate correlation coefficient between 'feature' and 'target'

correlations = featr.corrwith(target)


# sort the correlations in descending order

correlations = correlations.sort_values(ascending=False)


# select the features with correlation between 0.6 to 1 or between -0.6 to -1

selected_features = correlations.loc[(correlations >= 0.5)].index.get_level_values(0).unique(
```

```
selected_features
```

```
Index(['hc_mortgage_median', 'hc_mortgage_stdev', 'hc_mean', 'rent_mean',
       'hi_stdev', 'hi_mean', 'overall_income', 'family_mean', 'family_stdev',
       'hc_median', 'hi_median', 'family_median', 'rent_median', 'rent_stdev',
       'hc_stdev', 'Bad_Debt', 'home_equity'],
      dtype='object')
```

```
#import required libraries

from sklearn.linear_model import LinearRegression
linereg=LinearRegression()
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error,accuracy_score



#select x and y
#split into train and test
# run the model

X = df_train[selected_features]
y = df_train['hc_mortgage_mean']
X_train1, X_test1, y_train1, y_test1 = train_test_split(X,y,test_size=0.3,random_state=23)
linereg.fit(X_train1,y_train1)
```

```
LinearRegression()
```

```
# Predicting on the basis of mortgage mean

y_pred1 = linereg.predict(X_test1)


#check r2 score and rmse

print("Overall R2 score of linear regression model",r2_score(y_test1,y_pred1))
print("Overall RMSE of linear regression model", np.sqrt(mean_squared_error(y_test1,y_pred1))
```

```
Overall R2 score of linear regression model 0.9451518142605312
Overall RMSE of linear regression model 1775.2389003642152
```

## as accuracy of r2 not good :

## b) Run another model at State level. There are 52 states in USA.

```
#check stateid available
```

```
df_train['STATEID'].value_counts()
```

```
4     2905
43    1928
32    1769
9     1594
38    1219
13    1110
35    1087
22    1037
33     824
10     770
30     696
46     690
14     572
21     549
47     542
2      537
49     536
20     529
42     527
23     499
25     495
5      477
18     428
0      412
17     399
40     394
36     369
6      325
37     322
51     313
16     301
15     288
3      268
24     244
28     243
44     230
31     197
27     188
48     168
19     130
11     125
26     120
29     112
12     101
39      90
41      89
34      80
1       75
7       71
8       66
45      63
50      58
Name: STATEID, dtype: int64
```

```
#take any state Id to check and create new dataset for that id
#fit into train test and run modle for this dataset
#check new r2 and rmse values
#using for loop


linreg2=LinearRegression()

state_ids=[43,4,32]
for i in state_ids:
  x_state_df=df_train[df_train['STATEID']==i][selected_features]
  y_state_df=df_train[df_train['STATEID']==i]['hc_mortgage_mean']
  X_train2, X_test2, y_train2, y_test2 = train_test_split(x_state_df,y_state_df,test_size=0.3
  linreg2.fit(X_train2,y_train2)
  y_pred2 = linreg2.predict(X_test2)
  print('Result of state id =',i)
  print("Overall R2 score of linear regression model",r2_score(y_test2,y_pred2))
  print("Overall RMSE of linear regression model", np.sqrt(mean_squared_error(y_test2,y_pred2
```

```
Result of state id = 43
Overall R2 score of linear regression model 0.9315211702059945
Overall RMSE of linear regression model 1735.4125777730935
Result of state id = 4
Overall R2 score of linear regression model 0.8964217232557066
Overall RMSE of linear regression model 1581.4999204135584
Result of state id = 32
Overall R2 score of linear regression model 0.9293587584996174
Overall RMSE of linear regression model 2031.3675858780744
```

```
#saving file

file_name = 'final_data.xlsx'
df_train.to_excel(file_name)
```

## Conclusion:

**As r2 values are good we can say our modle is good to test, we can test it on test data**