```
#%% md
# Real Estate - PGP - Capstone Project

## 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.

## Dataset 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
```

```
21 the householder
22 #%% md
23 ### 1. Import Data
24 #%%
25 #IMPORT PACKAGES AND DATA
26
27 import numpy as np
28 import pandas as pd
29 import os
30 import warnings
31 warnings.filterwarnings('ignore')
32
33 import matplotlib.pyplot as plt
34 import seaborn as sns
35 sns.set(style= 'white', color_codes = True)
36 sns.set(font_scale = 1.5)
37 from itertools import cycle
38 #%%
39 train_df = pd.read_csv('train.csv')
40 test_df = pd.read_csv('test.csv')
41
42 train_df.head()
43 #%%
44 train_df.columns
45 #%%
46 test_df.columns
47 #%%
48 len(train_df)
49 #%%
50 train_df.describe()
51 #%%
52 train_df.info()
53 #%% md
54 ### 2. Find the primary key and locate the indexing
   requirement
55 #%%
56 #UID is the unique identifier in the test and
   training set data. So an index can be created from
```

```
56 the UID Feature
57 train_df.set_index(keys = ['UID'], inplace=True)
58 test_df.set_index(keys = ['UID'], inplace= True)
59 #%%
60 train_df.head()
61 #%%
62 test_df.head()
63 #%% md
64 ### 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.
65 #%%
66 #PERCENTAGE TOTALS OF MISSING VALUES IN TEST AND
   TRAINING DATASETS
67
68 mvt_train = train_df.isnull().sum() *100/len(train_df
   )
69 mvt_train_df = pd.DataFrame(mvt_train, columns = ['
   Percentage of Values Missing'])
70 mvt_train_df.sort_values(by=['Percentage of Values
   Missing'],inplace=True, ascending=False)
71 mvt_train_df[mvt_train_df['Percentage of Values
   Missing']> 0][:10]
72 #%%
73 mvt_test = test_df.isnull().sum() *100/len(train_df)
74 mvt_test_df = pd.DataFrame(mvt_test, columns = ['
   Percentage of Values Missing'])
75 mvt_test_df.sort_values(by=['Percentage of Values
   Missing'],inplace=True, ascending=False)
76 mvt_test_df[mvt_test_df['Percentage of Values Missing
   ']> 0][:10]
77 #%%
78 train_df['SUMLEVEL'].unique() #SUM LEVEL HAS ONLY ONE
    VALUE AND WILL NOT HELP WITH DIFFERENTIATING RESULTS
79 #%% md
80 **Based on the missing value analysis, we will drop
   the BLOCKID value entirely since the TRAIN_DF sample
```

```
 80 has 100% NULL values. We will also impute the
    missing values for the other columns with the mean
    of the data since the values for the dataset are
    negligible. (Note: If the dataset were larger we
    would drop the missing queries as they would not
    impact the study from the models.**
 81 #%%
 82 train_df.drop(columns = ['BLOCKID','SUMLEVEL'],
    inplace=True)
 83 test_df.drop(columns = ['BLOCKID','SUMLEVEL'],
    inplace=True)
 84 #%%
 85 #IMPUTE THE MEAN
 86
 87 missing_train_cols = []
 88
 89 for col in train_df.columns:
 90     if train_df[col].isna().sum() !=0:
 91         missing_train_cols.append(col)
 92     print(missing_train_cols)
 93 #%%
 94 missing_test_cols = []
 95
 96 for col in test_df.columns:
 97     if test_df[col].isna().sum() !=0:
 98         missing_test_cols.append(col)
 99     print(missing_test_cols)
100 #%%
101 for col in train_df.columns:
102     if col in (missing_train_cols):
103         train_df[col].replace(np.nan, train_df[col].
    mean(), inplace=True)
104 #%%
105 for col in test_df.columns:
106     if col in (missing_test_cols):
107         test_df[col].replace(np.nan, test_df[col].
    mean(), inplace=True)
108 #%%
```

```
train_df.isna().sum().sum()
#%%
test_df.isna().sum().sum()
#%% md
**ALL NULL VALUES HAVE BEEN REMOVED**
#%% md
## Exploratory Data Analysis:

### 4. Perform Debt analysis and Study Population Density

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
#%%
#GETTING A SMALLER QUERIED DATASET WITH REQUIRED ELEMENTS TO EXPLORE ITEM (A)

from pandasql import sqldf
q1 = "select place,pct_own,second_mortgage,lat,lng from train_df where pct_own >0.10 and second_mortgage <0.5 order by second_mortgage DESC LIMIT 2500;"
pysql_df = lambda q: sqldf(q, globals())
train_df_location_mort_pct=pysql_df(q1)
#%%
train_df_location_mort_pct.head()
#%%
import plotly.express as px
import plotly.graph_objects as go
#%%
#VISUALIZE USING PLOTLY GEO-MAPPING PACKAGE. <USED REFERENCE CODE FROM GITHUB>

fig = go.Figure(data = go.Scattergeo(
```

```
136     lat = train_df_location_mort_pct['lat'],
137     lon = df_train_location_mort_pct['lng']
138 ),)
139 
140 fig.update_layout(
141     geo=dict(
142         scope = 'north america',
143         showland = True,
144         landcolor= 'rgb(212,212,212)',
145         subunitcolor ='rgb(255,255,255)',
146         showlakes = True,
147         lakecolor = 'rgb(255,255,255)',
148         showsubunits = True,
149         showcountries = True,
150         resolution = 50,
151         projection = dict(
152             type = 'conic conformal',
153             rotation_lon = -100
154         ),
155         lonaxis = dict(
156             showgrid = True,
157             gridwidth = 0.5,
158             range = [-140.0, -55.0],
159             dtick = 5
160         ),
161         lataxis = dict(
162             showgrid = True,
163             gridwidth = 0.5,
164             range = [20.0, 60.0],
165             dtick = 5
166         )
167     ),
168     title = 'Top 2,500 locations with Highest rate
    of Second Mortgages and Percent Ownership is Above
    10%'
169     )
170 fig.show()
171 #%% md
```

```
**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**
#%%
train_df['bad_debt'] = train_df['second_mortgage'] + train_df['home_equity'] - train_df['home_equity_second_mortgage']
#%%
train_df['bins'] = pd.cut(train_df['bad_debt'], bins = [0,0.1,1], labels = ['less than 50%', 'greater than 50%'])
train_df.groupby(['bins']).size().plot(kind='pie', subplots=True, startangle =90, autopct = '%1.1f%%')
plt.axis('equal')
plt.title('Bad Debt Distribution')

plt.show()
#%% md
**d) Create Box and whisker plot and analyze the distribution for 2nd mortgage, home equity, good debt, and bad debt for different cities**
#%%
#SELECT TWO CITIES TO COMPARE
train_df['city'].unique()

#PANTEGO and STANTON CITY
#%%
cols = ['second_mortgage', 'home_equity', 'debt', 'bad_debt']
box_pantego = train_df.loc[train_df['city'] == 'Trinity']
box_stantoncity = train_df.loc[train_df['city'] == 'Nocona']
box_cities = pd.concat([box_pantego,box_stantoncity])
box_cities.head()
```

```
195 #%%
196 plt.figure(figsize=(7,5))
197 sns.boxplot(data=box_cities,x='second_mortgage', y='
    city',width=0.5,palette="Set3")
198 plt.show()
199
200 plt.figure(figsize=(7,5))
201 sns.boxplot(data=box_cities,x='home_equity', y='city
    ',width=0.5,palette="Set3")
202 plt.show()
203
204 plt.figure(figsize=(7,5))
205 sns.boxplot(data=box_cities,x='debt', y='city',width
    =0.5,palette="Set3")
206 plt.show()
207
208 plt.figure(figsize=(7,5))
209 sns.boxplot(data=box_cities,x='bad_debt', y='city',
    width=0.5,palette="Set3")
210 plt.show()
211 #%% md
212 **e) Create a collated income distribution chart for
     family income, house hold income, and remaining
    income**
213 #%%
214 sns.displot(train_df['hi_mean'])
215 plt.title('Household Income Distribution Chart')
216 plt.show()
217
218 sns.distplot(train_df['family_mean'])
219 plt.title('Family Income Distribution Chart')
220 plt.show()
221
222 sns.distplot(train_df['family_mean']-train_df['
    hi_mean'])
223 plt.title('Remaining Income Distribution Chart')
224 plt.show()
225 #%% md
```

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

ALand - Land Square Footage
pop - Male and Female Populaton
#%%
train_df['pop_density'] = train_df['pop']/train_df['ALand']
test_df['pop_density'] = test_df['pop']/test_df['ALand']
#%%
sns.displot(train_df['pop_density'])
plt.title('Population Density')
plt.show()
#%% md
**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**
#%%
train_df['age_median']=(train_df['male_age_median']+train_df['female_age_median'])/2

test_df['age_median']=(test_df['male_age_median']+test_df['female_age_median'])/2

train_df[['male_age_median','female_age_median', 'male_pop','female_pop','age_median']]
#%%
sns.displot([train_df['age_median']])
plt.title('Median Age')
plt.show()
#%% md
**FINDINGS:**

- Median Age is concentrated between ages 20 and 60, with a majority around the age of 40.
- Median age distribution has a Gaussian
```

```
Distribution
- Slight Right Skewness is noted
#%%
train_df['pop_bins'] = pd.cut(train_df['pop'], bins
 = 5, labels = ['lowest','low','medium', 'high', '
highest'])

#%%
train_df[['pop','pop_bins']]
#%%
train_df['pop_bins'].value_counts()
#%% md
**Analyze the married, separated, and divorced
population for these population brackets. Visualize
using appropriate chart type**
#%%
train_df.groupby(by='pop_bins')[['married','
separated','divorced']].count()
#%%
train_df.groupby(by='pop_bins')[['married','
separated','divorced']].agg(['mean','median'])
#%%
plt.figure(figsize=(20,10))
pop_bin_married=train_df.groupby(by='pop_bins')[['
married','separated','divorced']].agg(["mean"])
pop_bin_married.plot(figsize = (20,10))
plt.legend(loc = 'best')
plt.show()
#%%
state_rent_mean = train_df.groupby(by = 'state')['
rent_mean'].agg(['mean'])
state_income_mean = train_df.groupby(by = 'state')['
family_mean'].agg(['mean'])

percentage_rent_income = state_rent_mean['mean']/
state_income_mean['mean']
#%%
percentage_rent_income.head(10)
```

```
281 #%%
282 #DEPICT LEVEL OF RENT AS AN OVERALL PERCENTAGE OF
    INCOME
283 print(round(sum(train_df['rent_mean']) / sum(
    train_df['family_mean']) *100),'%')
284 #%% md
285 #### 5. Perform correlation analysis for all the
    relevant variables by creating a heatmap. Describe
    your findings.
286 #%%
287 train_df.columns
288 #%%
289 #SELECT DATA TO CORRELATE USING HEATMAP
290 co = train_df[['COUNTYID','STATEID','zip_code','type
    ','pop', 'family_mean',
291                'second_mortgage', 'home_equity', '
    debt','rent_median',
292                'age_median','hs_degree','pct_own', '
    married','separated', 'divorced']].corr()
293 #%%
294 plt.figure(figsize= (20,10))
295 sns.heatmap(co, annot= True, cmap = 'plasma')
296 plt.show()
297
298 #%% md
299 **NOTES FROM HEATMAP**
300 - High correlation seen between hs_degree and
    family_mean
301 - High positive correlation seen between pct_down
    and married
302 #%% md
303 ## DATA PRE-PROCESSING
304
305 ### 1. The multivariate data has a significant
    number of measured variables, not all of which have
    an impact on the selected predicted variable. The
    goal is to find the measured variables that would be
     dependent on smaller unobserved common factors of
```

```
latent variables.

### See the common factors below and plotted loadings below. Use factor analysis to find the latent variables within the dataset and gain insights on the linear relationships within the data.

**Latent Variables:**
- High School graduation rates
- Median population age
- Second mortgage stats
- Percent own
- Bad debt expense
#%%
from sklearn.decomposition import _factor_analysis
from factor_analyzer import FactorAnalyzer
#%%
fa = FactorAnalyzer(n_factors=5)
fa.fit_transform(train_df.select_dtypes(exclude=('object', 'category')))
fa.loadings_
#%% md
## Linear Regression Model

### 1. This linear regression model serves to predict the total monthly expenditure for the home mortgages loan. This is the mean monthly mortgage and owner costs of a specified geographical location.

**NOTE: All NaN values have been removed from the dataset.**
#%%
train_df.columns
#%%
train_df['type'].unique()
type_dict = {'type_2':{'City':1,
```

```
333                         'Urban':2,
334                         'Town': 3,
335                         'CDP':4,
336                         'Village':5,
337                         'Borough':6}
338                 }
339 train_df['type_2'] = train_df[type_dict,inplace =
    True]
340 #%%
341 train_df['type_2'].unique()
342 #%%
343 test_df.replace(type_dict,inplace = True)
344 #%%
345 test_df['type'].unique()
346 #%%
347 feature_cols = ['COUNTYID','STATEID','zip_code','
    type','pop', 'family_mean','second_mortgage', '
    home_equity', 'debt','rent_median', 'age_median','
    hs_degree','pct_own', 'married','separated', '
    divorced']
348 #%%
349 X_train = train_df[feature_cols]
350 y_train = train_df['hc_mortgage_mean']
351
352 X_test = test_df[feature_cols]
353 y_test = test_df['hc_mortgage_mean']
354 #%%
355 from sklearn.preprocessing import StandardScaler
356 from sklearn.linear_model import LinearRegression
357 from sklearn.metrics import r2_score,
    mean_absolute_error,mean_squared_error,
    accuracy_score
358 #%%
359 X_train.head()
360 #%%
361 scaler = StandardScaler()
362 X_train_scaled = scaler.fit_transform(X_train)
363 X_test_scaled = scaler.fit_transform(X_test)
```

```
#%% md
**a)Run a model at a Nation level. If the accuracy levels and R square are not satisfactory proceed to below step.**
#%%
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled , y_train)
#%%
y_pred = lin_reg.predict(X_test_scaled)
#%%
print('Overall R2 Score of linear regression model:' , r2_score(y_test,y_pred))
print("Overall RMSE of linear regression model", np.sqrt(mean_squared_error(y_test,y_pred)))
#%% md
Model has run successfully and has good results but we will further investigate the model's performance at the state level.

**b) Run another model at State level.**
#%%
state = train_df['STATEID'].unique()
state[0:5]
#%%
for i in [20,1,45]:
    print('State-ID:',i)

    X_train_state = train_df[train_df['COUNTYID'] == i][feature_cols]
    y_train_state = train_df[train_df['COUNTYID'] == i]['hc_mortgage_mean']

    x_test_state = test_df[test_df['COUNTYID'] == i][feature_cols]
    y_test_state = test_df[test_df['COUNTYID'] == i]['hc_mortgage_mean']

    x_train_state_scaled = scaler.fit_transform(
```

```
X_train_state)
    x_test_state_scaled = scaler.fit_transform(x_test_state)

    lin_reg.fit(x_train_state_scaled, y_train_state)
    y_pred_nation = lin_reg.predict(x_test_state_scaled)

    print('Overall R2 Score of linear regression model for state,' , i, ':',r2_score(y_test_state, y_pred_nation) )
    print('Overall RMSE of linear regression model for state, ', i, ':', np.sqrt(mean_squared_error(y_test_state,y_pred_nation)))
    print('\n')
#%%
#checking error (residuals)
residuals = y_test - y_pred
residuals
#%%
plt.hist(residuals)
#%%
sns.displot(residuals)
#%%
plt.scatter(residuals, y_pred)
#%%
train_df.to_excel (r'/Users/Dishant/Documents/DATA SCIENCE/Machine Learning Projects/Real Estate Data.xlsx', index = None, header=True)
#%%
%pip install xgboost
#%%

```