FACULTY OF ENGINEERING, ENVIRONMENT & COMPUTING

(7082CEM)

Coursework
Demonstration of a Big Data Program

MODULE LEADER: Dr. Marwan Fuad

Student Name: Sivasharmini Ganeshamoorthy

SID: 9685011

BIG DATA ANALYSIS THROUGH MACHINE LEARNING USING PYSPARK

I can confirm that all work submitted is my own: Yes

Table of Contents

1. Introduction

In the fast growing world, Artificial Intelligence especially machine learning plays a major role in the field of Information Technology. This has been implemented in different ways to identify the varied and diverse resources to process fast and maintain the high standards in the products and services. However, incorporating machine learning strategies are challenging due to high cost and huge amount of space for files, CPU and memory. Sophisticated platforms have started to develop to handle to massive amount of data. Spark is one of the famous distributed computation platform for big data analysis which comprises outstanding functionalities such as performing faster in large dataset, ease of use, fault tolerance and overcoming memory latency. Although, Spark offers programming for different programming languages, Python has been considered for this study for its special qualities such as real time screen analytics, facilitates data visualisation and process faster on the framework. Following this PySpark which is a combination of Spark and Python is implemented in this study for Spark data processing through PySpark which easily integrate and collaborate with RDD through navigating Py4J library. In this study Resilient Distributed Dataset (RDD) which is a primary data structure of Spark is used to split the nodes and quickly perform the calculation for the given dataset through its functions: transformation and actions.

Data for this study is based on USA Economic Statistic Dataset which is from US Census Reports. However, due to time limitations a part of this huge data is extracted dataset from Kaggle dataset and used in this study for Big Data analysis and Data visualisation. The collected data contains 79 attributes but only 29 attributes were chosen which are best suited for this study. Income, age, marriage, mortgage, home equity loan and demographics are some of the attributes chosen for the study. After the selection of dataset PySpark was launched using number on installation steps. Next, preproessing for dataset was completed through loading data (Loading through RDD and Dataframe). Following this, duplications were removed and columns were dropped including handing missing values. Then Dataframe operations were completed through different methods: Groupby, Distinct, Orderby, Built-in-functions, describe function and check value for specific column. This then led to exploratory analysis. The explanation for visualisations and algorithms derived were provided in the discussion section.

Finally, the study concludes that there is a positive relationship between rent_mean and family_mean. It is also identified California is one of the largest State in US is an expensive place due to having high values in populations, debt, home equity and household and mortgage cost. More information about the study and results are discussed in details in the following sections.

2. Implementation

2.1 Background study

SPARK

SPARK which is a distributed computing platform acts as a common engine for big data analysis processing and computations. Key purpose of this open source is to overcome the drawbacks of MapReduce by performing faster in large dataset, ease of use, fault tolerance and overcoming memory latency. SPARK in its higher level has divided into two main parts: head node and workers. Spark driver which is a component in head node helps to run the code which is written by the developer whereas main execution of the code takes place in workers. SQL, Machine learning and Streaming are built within SPARK to enhance the big data analysis. SPARK provides programming API's for Python,Scala, Java and R. Among these high level programming languages Python allows wide range of libraries for machine learning and real time screen analytics, facilitates data visualisation and process faster on the framework. Considering all these advantages, Python has been chosen for this study to demonstrate the sound knowledge for the selected semi-structured dataset. The following section discuss about PySpark which is the combination of both Python and SPARK.

SparkContext

SparkContext is the gatekeeper for any spark developed applications or functionality. In other words, driver program which includes the main function begins for the SparkContext to get commenced while running a Spark application. Following this, driver program will allow operations to run within the executors on worker nodes. SparkConext in PySpark is in the form of sc by default therefore generating a new SparkContext will provide an error.

RDD

The primary data structure of Spark is called as Resilient Distributed Dataset (RDD). The function of this low level object is to split nodes according to some key among various nodes within the cluster and distributes to executor nodes. This will allows Spark to perform efficiently through quick calculations against the given dataset. Therefore, this distributed group of immutable JVM objects acts as the key pillar of the Apache Spark. RDD operations are mainly separated into two divisions: Transformation and Actions. Transformations are the RDD operations that brings back a new RDD whereas Actions are the operations that brings a result.

DataFrame

Similar to RDD, a DataFrame is a collection of data,where data cannot be changed after creating it (immutable distributed collection of data) . However, this differs from an RDD, by organising the data into named columns in a table format. This is especially used to easily handle large datasets through inserting a structure onto immutable distributed collection of data, allowing to large number of viewers, helping to abstract more data and giving more domain precise language API to manipulate.

PySpark

Spark which is developed in Scala language collects the program into type code for the JVM and capitalise HDFS to accomplish spark bug data processing. Spark and Python are combined together for the creation of PySpark. In this Python API for Spark helps to easily assimilate and collaborate with RDD through navigating Py4J library. In other words, Spark data processing is revealed to Python through PySpark. PySpark creates API connections around Spark to utilise the entire Python ecosystem among all the nodes within the cluster. In addition, it allows to use Python's rich libraries (eg: Scikit-Learn) and data processing (eg:Pandas).

In the beginning of coordinating a Spark programme the initial step will be creating a SparkContext object. This helps Spark to access the nodes within the cluster. On the other hand, PySpark Context are generated by Python. As mentioned previously Py4J acts as a mediator to combine Python program to JVM Spark Context. Following this, JVM Spark Context converts the object into smaller bites before transferring to the nodes in the cluster for execution. During this process cluster manager plays a vital role through assigning relevant resources and schedules and then send it to Spark workers. Finally, Python virtual machines are activated by the Spark workers in the cluster. Executors in Park workers control the machines through functions such as computation, storage and cache in.

The diagram below shows how both PySpark and Spark contexts are managed by Spark driver with the aid of local file system and through communicating with Spark worker through cluster manager.

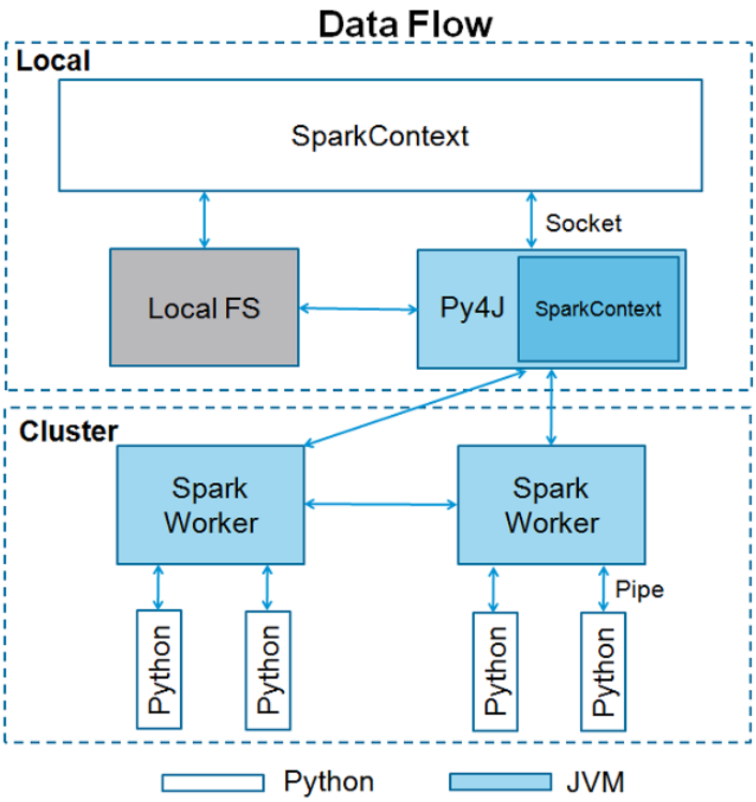


Image 2.1: Data flow of PySpark

2.2 Dataset

USA Economic Statistic Dataset which is from US Census Reports (2012-2016 ACS 5-Year Documentation) is a huge dataset. Therefore, a version of this dataset has been extracted from Kaggle dataset and used in this study for Big Data analysis and Data visualisation. This chosen dataset includes information on income, age, marriage, mortgage, home equity loan and demographics. There are 79 attributes in the dataset however unique values, constant values and inappropriate attributes for this specific goal was opt out and finally only 29 of them were chosen for this study. Moreover, it contains 39030 number of records. The main aim of this study is to analyse economic status of different locations of US through analysing the factors such income, debt, education, mortgage,etc.

The table below shows the chosen attributes from the dataset including its description and data types.

Attribute	Description	Data Types
state	State name of the specific geographic area	string
city	Closest city name of the specific geographic area	string
place	The place name of the specific geographic area	string
type	The place type of the specific geographic area	string
lat	The latitude of geographic location	double
lng	The longitude of geographic location	double
ALand	Area of the land in the location	long
Awater	Area of the water in the location	long
pop	Male or female population in the geographic location	integer
male_pop	Male population in the geographic location	integer
female_pop	Female population in the geographic location	integer
rent_mean	The mean of gross rent of the specific geographic location	double
hi_mean	The mean household income for specific geographic location	double
family_mean	The mean family income of the specified geographic location	double
hc_mortagage_mean	The mean monthly mortgage and owner costs of specific geographic location	double
hc_mean	The mean of monthly owner costs of specific geographic location	double
home_equity_second_mortgage	Percentage of homes with second mortgage and home equity loan	double
second_mortgage	Percentage of houses with a second mortgagae	double
home_equity	Percentage of homes with a home equity loan	double
debt	Percentage of home with some sort of debt	double
hs_degree	Percentage of people with high school degree	double
hs_degree_male	Percentage of male with high school degree	double
hs_degree_female	Percentage of female with high school degree	double
male_age_mean	The mean male age of specific geographic location	double
Female_age_mean	The mean female age of specific geographic location	double
pct_own	Percentage of ownership of houses	double
married	Percentage of person got married for specific geographic location	double
separated	Percentage of person got separated for specific geographic location	double
divorced	Percentage of person got divorced for specific geographic location	double

2.3 Installation Steps- Spark/Pyspark Local Mode

Machine used to perform this study has the hard disk drive with the capacity of 50 GB,RAM with 4GB and the used operating system is Ubuntu 18.04.3 LTS. The steps installation steps followed are explained below:

Step 1: Initially, it is important to check whether java is installed within the machine through looking at the current java version. If the java version is installed the following command is given:

java –version

And if the java is not installed the following command needs to be used:

sudo apt install default-jdk

Step 2:- Move the downloaded unzip spark file into Home folder. The command for this is given below:

tar -xzf spark-2.3.0-bin-hadoop2.7.tgz

Step 3:- Setting up the Spark Environment. The command for this shown below:

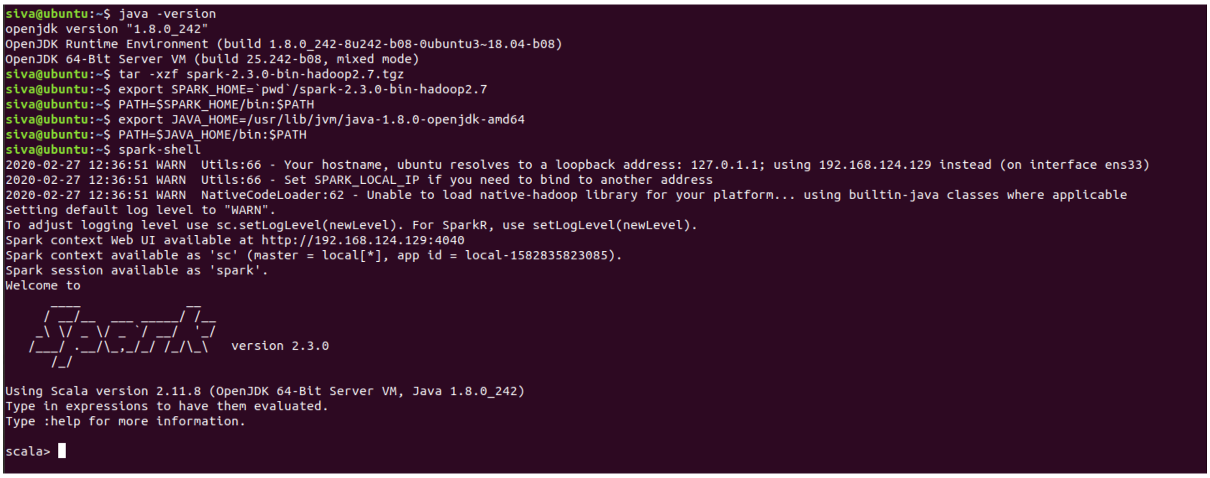
export SPARK_HOME=`pwd`/spark-2.3.0-bin-hadoop2.7
PATH=$SPARK_HOME/bin:$PATH

Step 4:- Setting up the java environment. The command for this is displayed below:

export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
PATH=$JAVA_HOME/bin:$PATH

Step 5:- It is ensured whether the Spark is installed correctly with the following command

spark-shell

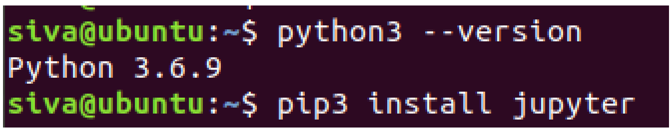


Step 6:- After Spark installation, to install PySpark it is important to check whether Python is installed in the machine. The following command will be used to check the Python version.

python3 --version

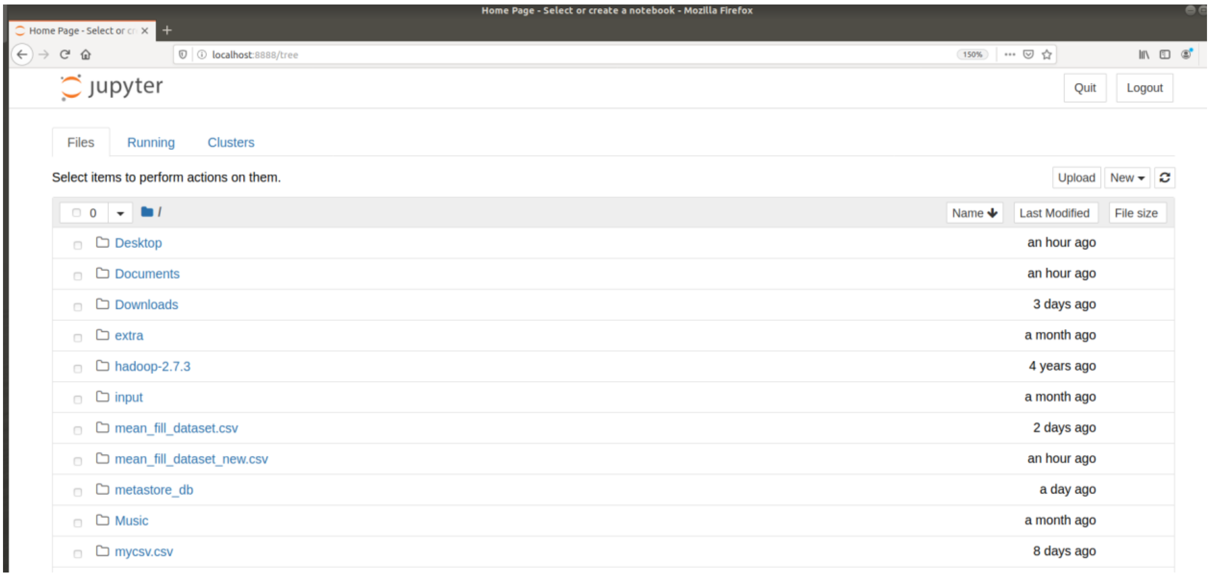
step7:- Once Python is installed Jupyter notebook can be installed with the following command. User interface within Jupyter notebook allows to easily write the Python programming.

pip3 install jupyter



Step 8:-To make sure whether Jupyter notebook is successfully installed the following command is used.

jupyter notebook

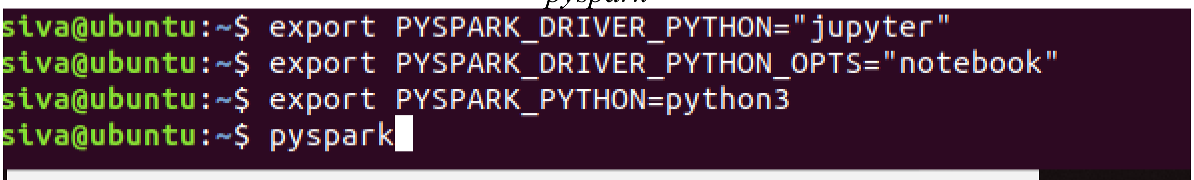


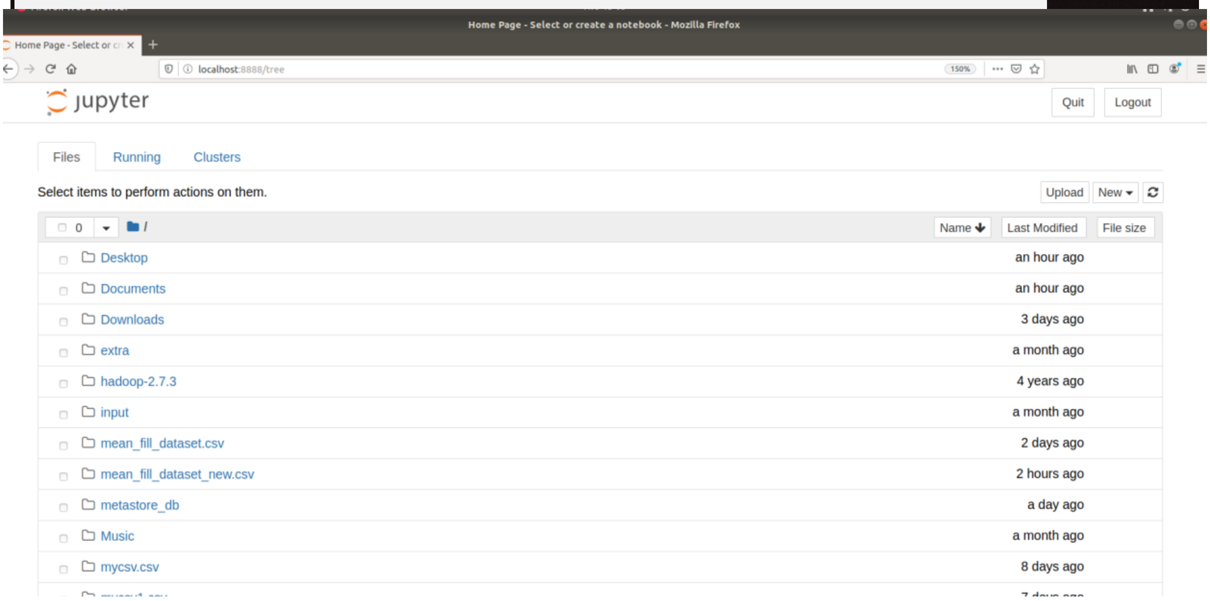
Step 9:- Setting up an environment for Python using following commands

export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
export PYSPARK_PYTHON=python3

Step 10:- Finally, to check the successful launch of PySpark the following commands need to be used. Alongside this SparkSession will be also launched at the same time. Following this SparkContext related to SparkSession will be launched to run the program.

pyspark





2.4 Preprocessing the Dataset

Dataset chosen for this machine learning study is not clean. In other words, it includes missing values, null values and duplications. Therefore, these inappropriate values are removed through prepossessing step.

2.4.1 Load the Data

There are two ways to load the data,

Method 1: Load Dataset through RDD

The dataset will be loaded through RDD. This is illustrated by the following code. In this, 'sc' refers to object of the SparkContext.

rdd1 = sc.textFile('real_estate_db.csv')

```
In [63]: rdd1 = sc.textFile('real_estate_db.csv')
         rdd1.first()

Out[63]: 'UID,BLOCKID,SUMLEVEL,COUNTYID,STATEID,state,state_ab,city,place,type,primary,zip_code,area_code,lat,lng,ALand,AWa
ter,pop,male_pop,female_pop,rent_mean,rent_median,rent_stdev,rent_sample_weight,rent_samples,rent_gt_10,rent_gt_1
5,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_media
n,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_med
ian,hc_stdev,hc_samples,hc_sample_weight,home_equity_second_mortgage,second_mortgage,home_equity,debt,second_mortg
age_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'
```

1.1 Operations with RDD

1.1.1 Filter function

This method enables to choose the appropriate dataset through setting up conditions to achieve the specific criteria. This is done through filtering both the head line (columns name) and creating a new RDD results without headlines. The code to achieve this task including the results are displayed below:

rdd_head= rdd1.first()

rdd2 = rdd1.filter(lambda line:line!=rdd_head)

Results

```
rdd_head= rdd1.first()
rdd2 = rdd1.filter(lambda line:line!=rdd_head)
rdd2.first()

'00220336,,140,016,02,Alaska,AK,Unalaska,Unalaska City,City,tract,99685,907,53.6210913,-166.7709793,2823180154,310
1986247,4619,2725,1894,1366.24657,1405,650.1638,131.50967,372,0.85676,0.65676,0.47838,0.35405,0.28108,0.21081,0.15
135,0.12432,661,370,107394.63092,92807,70691.05352,329.85389,874,114330.20465,101229,63955.77136,161.15239,519,226
6.22562,2283,768.53497,41.65644,155,840.67205,776,341.8558,58,29.74375,0.00469,0.01408,0.02817,0.7277,0.50216,0.77
143,0.30304,0.82841,0.82784,0.8294,38.45838,39.25,17.65453,709.06255,2725,32.78177,31.91667,19.31875,440.46429,189
4,0.25053,0.47388,0.30134,0.03443,0.09802'
```

1.1.2 Map Function

This is a transformation function which helps to split the entire data into separate elements. In this study it is separated through using the comma string. The code used is shown below:

rdd2.map(lambda line:line.split(',')).take(1)

```
In [66]: rdd2.map(lambda line:line.split(',')).take(1)

Out[66]: [['00220336',
  '',
  '140',
  '016',
  '02',
  'Alaska',
  'AK',
  'Unalaska',
  'Unalaska City',
  'City',
  'tract',
  '99685',
  '907',
  '53.6210913',
```

1.1.3 Combine Map and Filter and take

This is a combination of Map, Filter and Take functions. Take function helps to print the output results. The following example selected from the study depicts the filtering of three columns: BLOCKID, SUMLEVEL, type while state= Alaska, Georgia. Output for this study has printed out 5 records. The code used and output results displaying first 5 records are displayed below:

```
In [67]: (rdd2.filter(lambda line:line.split(',')[5] in ['Alaska','Georgia']).
          map(lambda line:(line.split(',')[0],
                           line.split(',')[1],
                           line.split(',')[2],
                           line.split(',')[5])).take(5))

Out[67]: [('00220336', '', '140', 'Alaska'),
          ('00220342', '', '140', 'Alaska'),
          ('00220343', '', '140', 'Alaska'),
          ('00220345', '', '140', 'Alaska'),
          ('00220347', '', '140', 'Alaska')]
```

```
(rdd2.filter(lambda line:line.split(',')[5] in ['Alaska','Georgia']).
 map(lambda line:(line.split(',')[0],
                  line.split(',')[1],
                  line.split(',')[2],
                  line.split(',')[5])).take(5))
```

1.1.4 Create DataFrame through RDD

In this RDD is used to generate the DataFrame. The code used is shown below:

```
df = spark.createDataFrame(rdd)
```

Method 2:- Load Dataset through DataFrame

In this method dataset is loaded through following steps:

2. 1.Load the csv file into Dataframe. The code chosen is given below:

df = spark.read.option('header','true').option('inferSchema','true').csv("real_estate_db.csv")

```
df = spark.read.option('header','true').option('inferSchema','true').csv("real_estate_db.csv")
```

2.2 .Print the columns. The code used is displayed below:

```
In [3]: print(df.columns)
```

```
['UID', 'BLOCKID', 'SUMLEVEL', 'COUNTYID', 'STATEID', 'state', 'state_ab', 'city', 'place', 'type', 'primary', 'zi
p_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', 're
nt_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_mo
rtgage', 'second_mortgage', 'home_equity', 'debt', 'second_mortgage_cdf', 'home_equity_cdf', 'debt_cdf', 'hs_degre
e', 'hs_degree_male', 'hs_degree_female', 'male_age_mean', 'male_age_median', 'male_age_stdev', 'male_age_sample_w
eight', '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']
```

2.3. Count the no of records. The code implemented for this shown below:

```
print(df.count())
```

```
39030
```

2.4. Display record in different ways

The final out puts can be displayed in several ways as discussed below:

2.4.1 take Funtion

```
df.take(1)
```

```
[Row(UID=220336, BLOCKID=None, SUMLEVEL=140, COUNTYID=16, STATEID=2, state='Alaska', state_ab='AK', city='Unalaska
', place='Unalaska City', type='City', primary='tract', zip_code=99685, area_code=907, lat=53.6210913, lng=-166.77
09793, ALand=2823180154, AWater=3101986247, pop=4619, male_pop=2725, female_pop=1894, rent_mean=1366.24657, rent_m
edian=1405.0, rent_stdev=650.1638, rent_sample_weight=131.50967, rent_samples=372.0, rent_gt_10=0.85676, rent_gt_1
5=0.65676, rent_gt_20=0.47838, rent_gt_25=0.35405, rent_gt_30=0.28108, rent_gt_35=0.21081, rent_gt_40=0.15135, ren
t_gt_50=0.12432, universe_samples=661, used_samples=370, hi_mean=107394.63092, hi_median=92807.0, hi_stdev=70691.0
5352, hi_sample_weight=329.85389, hi_samples=874.0, family_mean=114330.20465, family_median=101229.0, family_stdev
=63955.77136, family_sample_weight=161.15239, family_samples=519.0, hc_mortgage_mean=2266.22562, hc_mortgage_media
n=2283.0, hc_mortgage_stdev=768.53497, hc_mortgage_sample_weight=41.65644, hc_mortgage_samples=155.0, hc_mean=840.
67205, hc_median=776.0, hc_stdev=341.8558, hc_samples=58.0, hc_sample_weight=29.74375, home_equity_second_mortgage
=0.00469, second_mortgage=0.01408, home_equity=0.02817, debt=0.7277, second_mortgage_cdf=0.50216, home_equity_cdf=
```

2.4.2 Show Function – shows the results in a structured format

```
df.show(2)
```

```
+------+-------+--------+--------+-------+------+--------+-----------+-------------+----+-------+--------+--------
-+----------+------------+----------+----------+----+--------+----------+----------+----------+-----------+----------+-------
----------+------------+----------+----------+----------+----------+----------+----------+----------+----------+-
---------------+------------+------------+---------+-----------+----------------+----------+------------+---------
----+------------+--------------------+--------------+----------------+------------------+-----------------+------
------------------+-------------------+---------+---------+---------+----------+----------------+---------------
----------+---------------+-----------+-------+-------------------+---------------+--------+---------+----------
---+----------------+-------------+---------------+--------------+----------------------+----------------+--------
-------+----------------+----------------+-----------------------+-----------------+-------+-------+----------
+---------+--------+
|   UID|BLOCKID|SUMLEVEL|COUNTYID|STATEID| state|state_ab|       city|        place|type|primary|zip_code|area_cod
e|       lat|         lng|     ALand|    AWater| pop|male_pop|female_pop| rent_mean|rent_median|rent_stdev|rent_sa
mple_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|u
niverse_samples|used_samples|     hi_mean|hi_median|   hi_stdev|hi_sample_weight|hi_samples| family_mean|family_me
dian|family_stdev|family_sample_weight|family_samples|hc_mortgage_mean|hc_mortgage_median|hc_mortgage_stdev|hc_mor
tgage_sample_weight|hc_mortgage_samples|   hc_mean|hc_median|  hc_stdev|hc_samples|hc_sample_weight|home_equity_seco
nd_mortgage|second_mortgage|home_equity|   debt|second_mortgage_cdf|home_equity_cdf|debt_cdf|hs_degree|hs_degree_m
```

2.4.3 Head Function

```
df.head(2)
```

```
[Row(UID=220336, BLOCKID=None, SUMLEVEL=140, COUNTYID=16, STATEID=2, state='Alaska', state_ab='AK', city='Unalaska
', place='Unalaska City', type='City', primary='tract', zip_code=99685, area_code=907, lat=53.6210913, lng=-166.77
09793, ALand=2823180154, AWater=3101986247, pop=4619, male_pop=2725, female_pop=1894, rent_mean=1366.24657, rent_m
edian=1405.0, rent_stdev=650.1638, rent_sample_weight=131.50967, rent_samples=372.0, rent_gt_10=0.85676, rent_gt_1
5=0.65676, rent_gt_20=0.47838, rent_gt_25=0.35405, rent_gt_30=0.28108, rent_gt_35=0.21081, rent_gt_40=0.15135, ren
t_gt_50=0.12432, universe_samples=661, used_samples=370, hi_mean=107394.63092, hi_median=92807.0, hi_stdev=70691.0
5352, hi_sample_weight=329.85389, hi_samples=874.0, family_mean=114330.20465, family_median=101229.0, family_stdev
=63955.77136, family_sample_weight=161.15239, family_samples=519.0, hc_mortgage_mean=2266.22562, hc_mortgage_media
n=2283.0, hc_mortgage_stdev=768.53497, hc_mortgage_sample_weight=41.65644, hc_mortgage_samples=155.0, hc_mean=840.
67205, hc_median=776.0, hc_stdev=341.8558, hc_samples=58.0, hc_sample_weight=29.74375, home_equity_second_mortgage
=0.00469, second_mortgage=0.01408, home_equity=0.02817, debt=0.7277, second_mortgage_cdf=0.50216, home_equity_cdf=
0.77143, debt_cdf=0.30304, hs_degree=0.82841, hs_degree_male=0.82784, hs_degree_female=0.8294, male_age_mean=38.45
```

2.4.4 printSchema Function – This shows the column names and the data type they belongs to

```
In [26]: df.printSchema()

root
 |-- state: string (nullable = true)
 |-- city: string (nullable = true)
 |-- place: string (nullable = true)
 |-- type: string (nullable = true)
 |-- lat: double (nullable = true)
 |-- lng: double (nullable = true)
 |-- ALand: long (nullable = true)
 |-- AWater: long (nullable = true)
 |-- pop: integer (nullable = true)
 |-- male_pop: integer (nullable = true)
 |-- female_pop: integer (nullable = true)
 |-- rent_mean: double (nullable = true)
 |-- hi_mean: double (nullable = true)
 |-- family_mean: double (nullable = true)
 |-- hc_mortgage_mean: double (nullable = true)
 |-- hc_mean: double (nullable = true)
 |-- home_equity_second_mortgage: double (nullable = true)
 |-- second_mortgage: double (nullable = true)
 |-- home_equity: double (nullable = true)
 |-- debt: double (nullable = true)
 |-- hs_degree: double (nullable = true)
 |-- hs_degree_male: double (nullable = true)
 |-- hs_degree_female: double (nullable = true)
 |-- male_age_mean: double (nullable = true)
 |-- female_age_mean: double (nullable = true)
 |-- pct_own: double (nullable = true)
 |-- married: double (nullable = true)
 |-- separated: double (nullable = true)
 |-- divorced: double (nullable = true)
```

2.4.2 Remove Duplications

In this step all the duplications identified in the dataset will be removed. The code used is shown below:

```
df = df.dropDuplicates()
```

After removing the duplications, we have 38715 records. The code for this is displayed below:

```
df.count()

38715
```

2.4.3 Dropping Columns

In this stage unique id columns will be dropped due to having unique no in all the rows which is not useful for our analysis. Different scenarios for dropping columns are mentioned below:

- Removing the column which has distinct value.

To check the column the following code is used:

```
df.select('UID').distinct().count()

38715
```

When the distinct value is identified code shown below is used to drop UID column

```
df = df.drop('UID')
```

- Dropping BLOCKID since it is null for all the rows. The code used is shown below:

```
from pyspark.sql.functions import count
df.groupBy('BLOCKID').count().show()
```

```
+-------+-----+
|BLOCKID|count|
+-------+-----+
|   null|38715|
+-------+-----+
```

```
df = df.drop('BLOCKID')
```

- Dropping SUMLEVEL and primary because it is constant throughout the dataset. The code implemented for this displayed below:

```
from pyspark.sql.functions import count
df.groupBy('SUMLEVEL').count().show()
```

```
+--------+-----+
|SUMLEVEL|count|
+--------+-----+
|     140|38715|
+--------+-----+
```

```
from pyspark.sql.functions import count
df.groupBy('primary').count().show()
```

```
+-------+-----+
|primary|count|
+-------+-----+
|  tract|38715|
+-------+-----+
```

```
df = df.drop('SUMLEVEL','primary')
```

- Print the columns that are not relevant or used to the study. The code used is illustrated below:

```
df = df.drop('rent_median','rent_stdev','rent_samples','rent_sample_weight','rent_gt_10','rent_gt_15','rent_gt_20',
```

```
df = df.drop('hi_median','hi_stdev','hi_sample_weight','hi_samples')
```

```
df = df.drop('family_median','family_stdev','family_samples','family_sample_weight')
```

```
df = df.drop('hc_mortgage_median','hc_mortgage_stdev','hc_mortgage_sample_weight','hc_mortgage_samples')
```

```
df = df.drop('hc_median','hc_stdev','hc_sample_weight','hc_samples')
```

```
df = df.drop('second_mortgage_cdf','home_equity_cdf','debt_cdf')
```

```
df = df.drop('male_age_median','male_age_stdev','male_age_sample_weight','male_age_samples')
```

```
df = df.drop('female_age_median','female_age_stdev','female_age_sample_weight','female_age_samples')
```

```
df = df.drop('state_ab','zip_code','area_code')
```

```
df = df.drop('universe_samples','used_samples')
```

```
df = df.drop('married_snp','STATEID','COUNTYID')
```

2.4.4 Handling missing Values in the dataset

Handling missing values can be done through two approaches: Imputing missing values and removing rows which contains missing values.

Method 1: Imputing missing values

Step 1: Initially, columns names and the percentage of missed values will be checked.

```
total = df.count()
for col in df.columns:
    df_filter = (df.filter(df[col]=="0")).count()
    percen_filter = df_filter/total
    print(col, "\t", "with ‘0’ values: ", percen_filter)
```

```
state	with ‘0’ values:  0.0
city	with ‘0’ values:  0.0
place	with ‘0’ values:  0.0
type	with ‘0’ values:  0.0
lat	with ‘0’ values:  0.0
lng	with ‘0’ values:  0.0
ALand	with ‘0’ values:  0.0
AWater	with ‘0’ values:  0.38519953506392873
pop	with ‘0’ values:  0.004778509621593698
male_pop	with ‘0’ values:  0.005010977657238796
female_pop	with ‘0’ values:  0.00545008394679065
rent_mean	with ‘0’ values:  0.00826553015627018
hi_mean	with ‘0’ values:  0.006999870851091309
family_mean	with ‘0’ values:  0.007826423866718326
hc_mortgage_mean	with ‘0’ values:  0.015368720134314865
hc_mean	with ‘0’ values:  0.01653106031254036
home_equity_second_mortgage	with ‘0’ values:  0.2332945886607258
second_mortgage	with ‘0’ values:  0.19966421283740152
home_equity	with ‘0’ values:  0.0665116879762366
debt	with ‘0’ values:  0.015368720134314865
hs_degree	with ‘0’ values:  0.004985147875500452
hs_degree_male	with ‘0’ values:  0.005398424383313961
hs_degree_female	with ‘0’ values:  0.0059666795815575355
male_age_mean	with ‘0’ values:  0.005010977657238796
female_age_mean	with ‘0’ values:  0.00545008394679065
pct_own	with ‘0’ values:  0.01257910370657368
married	with ‘0’ values:  0.005889190236342503
separated	with ‘0’ values:  0.193697533255844
divorced	with ‘0’ values:  0.009040423608420509
```

Step 2: Next, missing columns will be printed.

```
missing_col =[]
for col in df.columns:
    df_filter = (df.filter(df[col]=="0")).count()
    if(df_filter > 0):
        missing_col.append(col)
print(missing_col)
```

```
['AWater', 'pop', 'male_pop', 'female_pop', 'rent_mean', 'hi_mean', 'family_mean', 'hc_mortgage_mean', 'hc_mean', 'home_equity_second_mortgage', 'second_mortgage', 'home_equity', 'debt', 'hs_degree', 'hs_degree_male', 'hs_degree_female', 'male_age_mean', 'female_age_mean', 'pct_own', 'married', 'separated', 'divorced']
```

List of missing columns will help to identify whether categorical or numerical columns have missing values. This will lead to come up with following two solutions which suitable to choose to handle the dataset.

I. Imputing mean values
II. Imputing KNN values

(i) Imputing mean values

Imputing mean values will be achieved through following steps:

➢ Identifying the mean value

- Null value with 0 will be filled for the missing numeric column. The code used is shown below:

```
df = df.fillna(0)
```

- Find the mean values for the missing columns which is explained by the following function During this mean for each column is identified through using the avg function. The code used for this displayed below:

```
#Find the avg of all numeric columns
from pyspark.sql.functions import avg

def find_columns_fill_mean(df, num_col, verbose=False):
    col_with_mean=[]
    for col in num_col:
        mean_value = df.select(avg(df[col]))
        avg_col = mean_value.columns[0]
        result = mean_value.rdd.map(lambda row : row[avg_col]).collect()

        if (verbose==True): print(mean_value.columns[0], "\t", result[0])
        col_with_mean.append([col, result[0]])
    return col_with_mean
```

```
print(col, "\t", "mean values: ",  find_columns_fill_mean(df,missing_col))
```

```
divorced         mean values:  [['AWater', 6144408.759834689], ['pop', 4344.563580007749], ['male_pop', 2136.82518
4037195], ['female_pop', 2207.738395970554], ['rent_mean', 1046.1010459418835], ['hi_mean', 69955.49093368076], ['
family_mean', 78370.33168289994], ['hc_mortgage_mean', 1606.0487613033706], ['hc_mean', 531.2880015376471], ['home
_equity_second_mortgage', 0.02537785122045718], ['second_mortgage', 0.029632937621077122], ['home_equity', 0.09983
200103319129], ['debt', 0.6223142820612165], ['hs_degree', 0.8538958439881182], ['hs_degree_male', 0.8471570233759
526], ['hs_degree_female', 0.8596712323388869], ['male_age_mean', 38.13408689913471], ['female_age_mean', 40.08259
130982822], ['pct_own', 0.6365622825778121], ['married', 0.5061182921348315], ['separated', 0.01901723053080203],
['divorced', 0.09960359731370264]]
```

➢ Filling the mean function
In this stage the missing values in each is column will be replaced by the mean of the each column. The code used is shown below:

```
#Fill missing values for mean
from pyspark.sql.functions import when, lit

def fill_missing_value_with_mean(df, numeric_cols):
    col_mean = mean_of_pyspark_columns(df, numeric_cols)

    for col, mean in col_mean:
        df = df.withColumn(col, when(df[col]== 0, lit(mean)).otherwise(df[col]))

    return df
```

```
df_mean_filter1 = fill_missing_value_with_mean(df, missing_col)
```

➢ Save the dataframe into a csv file.
After removing the missing value the new dataframe will be saved to csv file and will be used for exploratory data analysis.
The Line 1 code is used to collect the distributed data into single scv file whereas Line 2 code is used to create the a csv file with the name 'mean_fill_dataset_new'

```
df_mean_filter1 = df_mean_filter1.repartition(1)
df_mean_filter1.write.csv('mean_fill_dataset_new.csv',header=True)
```

(ii) KNN Imputation

- During this all the 0 value columns will be converted into Null values to impute the k-nearest value. The code use for this is shown below:

```
testDF1 = df.replace(0, None)
```

- Next is KNN imputation. Line 1 of the code shows loading the KNN imputer whereas Line 2 illustrates initialising KNN imputer with two nearest neighbour. Line 3 of code is used to fill the kNN values into dataset.

```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=2)
df_knn_filter = imputer.fit_transform(new_df[:3000])
```

- Since the output is an array this needs to be converted to dataframe and this is achieved through following code:

```
print(df_knn_filter)
```

```
[[ 3.44574952e+01 -8.56352140e+01  9.13714300e+07 ...  5.90020000e-01
   2.42100000e-02  1.38680000e-01]
 [ 3.12688350e+01 -8.52759901e+01  1.42938740e+08 ...  5.66780000e-01
   3.20400000e-02  1.12370000e-01]
 [ 3.34866689e+01 -8.68173266e+01  2.42284900e+06 ...  2.97580000e-01
   1.92200000e-02  6.63400000e-02]
 ...
 [ 3.88098033e+01 -7.68691578e+01  1.79066860e+07 ...  7.87350000e-01
   1.20000000e-02  8.72000000e-03]
 [ 4.30672253e+01 -8.45338429e+01  9.11074510e+07 ...  6.38170000e-01
   1.04200000e-02  5.03500000e-02]
 [ 4.30187659e+01 -8.37072465e+01  1.49190800e+06 ...  2.79190000e-01
   1.86100000e-02  8.62900000e-02]]
```

```
import pandas as pd
knn_fillet_data = pd.DataFrame({'lat': df_knn_filter[:, 0], 'lng': df_knn_filter[:, 1],'ALand': df_knn_filter[:, 2]
                        'pop': df_knn_filter[:, 4], 'male_pop': df_knn_filter[:, 5],'female_pop': df_knn_filter[:,
                        'hi_mean': df_knn_filter[:, 8], 'family_mean': df_knn_filter[:, 9],'hc_mortgage_mean': df_k
                        'home_equity_second_storage': df_knn_filter[:, 12], 'second_mortgage': df_knn_filter[:, 13]
                        'hs_degree': df_knn_filter[:, 16], 'hs_degree_male': df_knn_filter[:, 17],'hs_degree_female
                        'female_age_mean': df_knn_filter[:, 20], 'pct_own': df_knn_filter[:, 21],'married': df_knn_
```

- Finally this dataframe will be saved in cv file. The code used is shown below:

```
knn_fillet_data.to_csv('mycsv_knn.csv')
```

Method 2: Removing rows that contains missing values

Another approach to handle missing value is dropping missing value and the code for the process is shown below:

```
df_filtered = df[df['AWater'] != 0]
df_filtered = df_filtered[df_filtered['rent_mean'] != 0]
df_filtered = df_filtered[df_filtered['hi_mean'] != 0]
df_filtered = df_filtered[df_filtered['family_mean'] != 0]
df_filtered = df_filtered[df_filtered['female_pop'] != 0]
df_filtered = df_filtered[df_filtered['male_pop'] != 0]
df_filtered = df_filtered[df_filtered['pop'] != 0]
df_filtered = df_filtered[df_filtered['hc_mortgage_mean'] != 0]
df_filtered = df_filtered[df_filtered['hc_mean'] != 0]
df_filtered = df_filtered[df_filtered['home_equity_second_mortgage'] != 0]
df_filtered = df_filtered[df_filtered['second_mortgage'] != 0]
df_filtered = df_filtered[df_filtered['home_equity'] != 0]
df_filtered = df_filtered[df_filtered['debt'] != 0]
df_filtered = df_filtered[df_filtered['hs_degree'] != 0]
df_filtered = df_filtered[df_filtered['hs_degree_male'] != 0]
df_filtered = df_filtered[df_filtered['hs_degree_female'] != 0]
df_filtered = df_filtered[df_filtered['male_age_mean'] != 0]
df_filtered = df_filtered[df_filtered['female_age_mean'] != 0]
df_filtered = df_filtered[df_filtered['pct_own'] != 0]
df_filtered = df_filtered[df_filtered['separated'] != 0]
df_filtered = df_filtered[df_filtered['divorced'] != 0]
df_filtered = df_filtered[df_filtered['married'] != 0]
```

DataFrame Operations

DataFrame includes number of operations and selected operations for this study have been discussed below:

1. **GroupBy**

Dataset is grouped by state column and no of records in the dataset will be calculated. The code is displayed below:

```
df.groupBy('state').count().show()
```

```
+--------------------+-----+
|               state|count|
+--------------------+-----+
|                Utah|  326|
|              Hawaii|  170|
|           Minnesota|  696|
|                Ohio| 1530|
|            Arkansas|  361|
|              Oregon|  454|
|               Texas| 2733|
|        North Dakota|  110|
|        Pennsylvania| 1722|
|         Connecticut|  443|
|            Nebraska|  272|
|             Vermont|   94|
|              Nevada|  349|
|         Puerto Rico|  478|
|          Washington|  805|
|            Illinois| 1587|
|            Oklahoma|  550|
|District of Columbia|   96|
|            Delaware|  109|
|              Alaska|  105|
+--------------------+-----+
only showing top 20 rows
```

2. Distinct

The unique values in columns will be identified. The code implemented is shown below:

```
df.select('city').distinct().show()
```

```
+----------------+
|            city|
+----------------+
|       Worcester|
|         Jemison|
|      Prattville|
|         Hanover|
|       Fairbanks|
|    Harleysville|
|           Tyler|
|Kingsford Heights|
|         Palermo|
|         Newbern|
|        Fredonia|
| North Saint Paul|
|     Santa Paula|
|        Bluffton|
|     Middlefield|
|       Birchwood|
|    Belle Plaine|
|     Johnsonburg|
|  West Sand Lake|
|         Minster|
+----------------+
only showing top 20 rows
```

3. Orderby

In this a column will be sorted according a pattern of order. This study has ordered top 10 states in US, sorting by count. The code used are displayed below:

```
# Top 10 States
df.groupby('state').count().orderBy('count',ascending= False).show(10)
```

```
+--------------+-----+
|         state|count|
+--------------+-----+
|    California| 4150|
|         Texas| 2733|
|      New York| 2518|
|       Florida| 2272|
|  Pennsylvania| 1722|
|      Illinois| 1587|
|          Ohio| 1530|
|      Michigan| 1458|
|North Carolina| 1162|
|       Georgia| 1065|
+--------------+-----+
```

4. Built-in Function

There are plenty of in-build functions in PySpark and some of them were explored for this study. The code used is shown below:

```
from pyspark.sql import functions
print(dir(functions))
```

```
['AutoBatchedSerializer', 'Column', 'DataFrame', 'DataType', 'PandasUDFType', 'PickleSerializer', 'PythonEvalType
', 'SparkContext', 'StringType', 'UserDefinedFunction', '__all__', '__builtins__', '__cached__', '__doc__', '__fil
e__', '__loader__', '__name__', '__package__', '__spec__', '_binary_mathfunctions', '_collect_list_doc', '_collect
_set_doc', '_create_binary_mathfunction', '_create_function', '_create_udf', '_create_window_function', '_function
s', '_functions_1_4', '_functions_1_6', '_functions_2_1', '_functions_deprecated', '_lit_doc', '_message', '_strin
g_functions', '_test', '_to_java_column', '_to_seq', '_window_functions', '_wrap_deprecated_function', 'abs', 'aco
s', 'add_months', 'approxCountDistinct', 'approx_count_distinct', 'array', 'array_contains', 'asc', 'ascii', 'asin
', 'atan', 'atan2', 'avg', 'base64', 'bin', 'bitwiseNOT', 'blacklist', 'broadcast', 'bround', 'cbrt', 'ceil', 'coa
lesce', 'col', 'collect_list', 'collect_set', 'column', 'concat', 'concat_ws', 'conv', 'corr', 'cos', 'cosh', 'cou
nt', 'countDistinct', 'covar_pop', 'covar_samp', 'crc32', 'create_map', 'cume_dist', 'current_date', 'current_time
stamp', 'date_add', 'date_format', 'date_sub', 'date_trunc', 'datediff', 'dayofmonth', 'dayofweek', 'dayofyear', '
```

One of the in-build function of PySpark used for this study is Numeric Function. This is to calculate minimum and maximum of rent_mean column. The code implemented for this is displayed below:

```
from pyspark.sql.functions import min,max
df.select(min(col('rent_mean')),max(col('rent_mean'))).show(2)
```

```
+--------------+--------------+
|min(rent_mean)|max(rent_mean)|
+--------------+--------------+
|           0.0|    3962.34229|
+--------------+--------------+
```

5. Describe Function

This provides the summary about the column including minimum, maximum, count mean and stdev. The code adopted used for this shown below:

6. Check value for specific columns

During this stage only first three columns such as UID,BLOCKID,SUMLEVEL will be checked. The code implemented for this displayed below:

SQL Queries

The entry point for SQL functions is SQLContext. SQLContext is created through SparkContext. The steps followed are mentioned below:

Step 1: Creating a table called ‘data table’ and select all the records in the table. The code used is shown below:

```
df.registerTempTable('data_table')
sqlContext.sql('select * from data_table').show(1)
```

```
+-------+----------+----------------+----+----------+----------+--------+------+----+--------+----------+---------+
-----------+-----------+----------------+---------+--------------------------+---------------+-----------+-------
+---------+-------------+----------------+-------------+---------------+-------+-------+---------+--------+
|  state|      city|           place|type|       lat|       lng|   ALand|AWater| pop|male_pop|female_pop|rent_mean|
hi_mean|family_mean|hc_mortgage_mean|  hc_mean|home_equity_second_mortgage|second_mortgage|home_equity|    debt|hs_
degree|hs_degree_male|hs_degree_female|male_age_mean|female_age_mean|pct_own|married|separated|divorced|
+-------+----------+----------------+----+----------+----------+--------+------+----+--------+----------+---------+
-----------+-----------+----------------+---------+--------------------------+---------------+-----------+-------
+---------+-------------+----------------+-------------+---------------+-------+-------+---------+--------+
|Alabama|Fort Payne|Fort Payne City|Town|34.4574952|-85.635214|91371430| 86324|5671|    2754|      2917|489.98072|
43343.66173|54959.28042|      1103.68654|312.36929|                     0.026|          0.026|    0.07335|0.5626
7|  0.60773|       0.60821|          0.60722|     35.65068|       32.95663|0.49321|0.59002|  0.02421| 0.13868|
+-------+----------+----------------+----+----------+----------+--------+------+----+--------+----------+---------+
-----------+-----------+----------------+---------+--------------------------+---------------+-----------+-------
+---------+-------------+----------------+-------------+---------------+-------+-------+---------+--------+
only showing top 1 row
```

Step 2: Selecting distinct value of ‘type’ column. The code used is displayed below:

```
sqlContext.sql('select distinct(type) from data_table').show()
```

```
+-------+
|   type|
+-------+
|  Urban|
|Borough|
|    CDP|
|Village|
|   Town|
|   City|
+-------+
```

Step 3: Finding maximum value in 'rent_mean' column. The code used is shown below:

```
sqlContext.sql('select max(rent_mean) from data_table').show()
```

```
+--------------+
|max(rent_mean)|
+--------------+
|    3962.34229|
+--------------+
```

PySpark SQL

Another SQL approach to do SQL functions is through Pypark dataframe. Below example is calculating the percentage of city among all the other types. The code used is illustrated below:

```
#percentage of city in alltypes
from pyspark.sql.functions import count
from pyspark.sql.functions import col
df.filter(col('type') == 'City').count() / df.select('type').count()
```

Statistical Analysis with dataset

Two areas of statistical analysis within dataset is considered in this study.

1. Calculating skewness and kurtosis of column 'pop'(population). The code used is shown below:

```
from pyspark.sql.functions import col, skewness, kurtosis
df.select(skewness(df.pop),kurtosis(df.pop)).show()
```

```
+-----------------+----------------+
|    skewness(pop)|    kurtosis(pop)|
+-----------------+----------------+
|2.073522916279963|19.42071614442881|
+-----------------+----------------+
```

2. Calculating correlation between rent_mean and all the other numerical attributes. The cde implemented for this is displayed below:

```
numeric_features = [t[0] for t in df.dtypes if t[1] == 'int' or t[1] == 'double']
print(numeric_features)
```

```
['lat', 'lng', 'pop', 'male_pop', 'female_pop', 'rent_mean', 'hi_mean', 'family_mean', 'hc_mortgage_mean', 'hc_mea
n', 'home_equity_second_mortgage', 'second_mortgage', 'home_equity', 'debt', 'hs_degree', 'hs_degree_male', 'hs_de
gree_female', 'male_age_mean', 'female_age_mean', 'pct_own', 'married', 'separated', 'divorced']
```

```
import six
for i in df_filtered.columns:
    if not( isinstance(df_filtered.select(i).take(1)[0][0], six.string_types)):
        print( "Correlation to rent_mean for ", i, df_filtered.stat.corr('rent_mean',i))
```

```
Correlation to rent_mean for  lat 0.0030812775019517197
Correlation to rent_mean for  lng -0.16725953964770007
Correlation to rent_mean for  ALand -0.07159688914509205
Correlation to rent_mean for  AWater -0.009885280953300454
Correlation to rent_mean for  pop 0.1628590150595818
Correlation to rent_mean for  male_pop 0.15919802450750484
Correlation to rent_mean for  female_pop 0.16085487271966195
Correlation to rent_mean for  rent_mean 1.0
Correlation to rent_mean for  hi_mean 0.756261409241335
Correlation to rent_mean for  family_mean 0.7043059857618401
Correlation to rent_mean for  hc_mortgage_mean 0.7539786178578622
Correlation to rent_mean for  hc_mean 0.5989907501436637
Correlation to rent_mean for  home_equity_second_mortgage 0.09857716504146785
Correlation to rent_mean for  second_mortgage 0.12047017730816342
Correlation to rent_mean for  home_equity 0.39921346915161754
Correlation to rent_mean for  debt 0.44777012880650985
Correlation to rent_mean for  hs_degree 0.36196530766222007
Correlation to rent_mean for  hs_degree_male 0.3724633862540176
Correlation to rent_mean for  hs_degree_female 0.3278756779883243
Correlation to rent_mean for  male_age_mean 0.04340081020578103
Correlation to rent_mean for  female_age_mean 0.0023360753457670313
Correlation to rent_mean for  pct_own 0.1456335771096801
Correlation to rent_mean for  married 0.25908906438023255
Correlation to rent_mean for  separated -0.15281540063232774
Correlation to rent_mean for  divorced -0.38102549390421825
```

Exploratory Analysis

The purpose of this section is to visualise and analyse the dataset chosen for this study

Univariant Analysis

Univariant analysis for this study is investigated through following methods.

Histogram

This way easily helps to visualise the distribution of dataset. It enables to determine whether the distribution is normal or not through its visualisation. The code for debt attribute is given below:

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


#x = df_filtered['pop']
x = df_filtered.toPandas()['female_age_mean'].values.tolist()
bins = np.arange(0, 100, 1)

plt.figure(figsize=(10,8))
# the histogram of the data
plt.hist(x, bins, alpha=0.8, histtype='bar', color='blue',
         ec='blue')

plt.xlabel("Female age mean")
plt.ylabel('Percentage')
plt.xticks(bins)
plt.show()
```

Code 6.1: Code for histogram of female age mean

Result:-

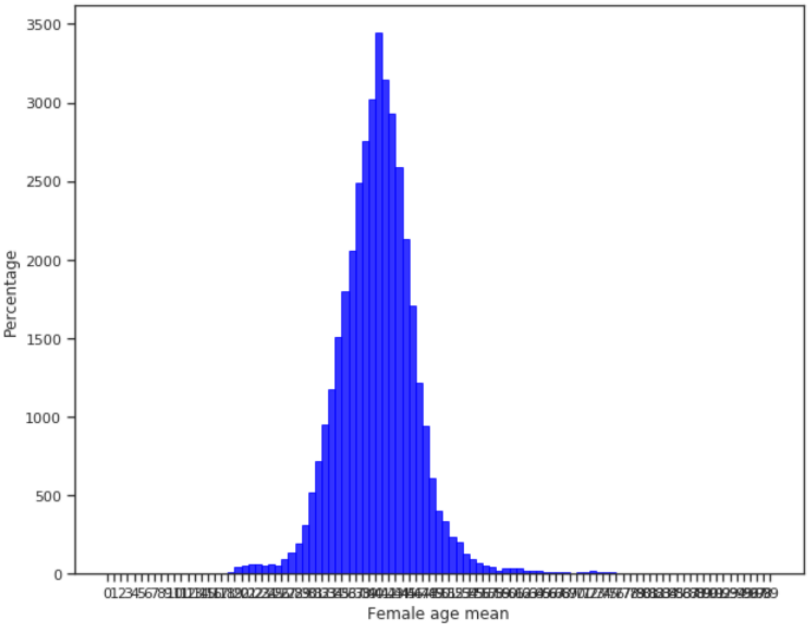


Image 6.1: Histogram of female age mean

Box plot and violin plot

The purpose of the boxplot is to give a summary description and identify the outliers and skewness. Violin plot describes the distribution of the data and its density distribution. The code for ‘pop’ (population) attribute is given below:

```
import seaborn as sns
import matplotlib.pyplot as plt
x = df_filtered.select('pop').toPandas()

fig = plt.figure(figsize=(20, 8))
ax = fig.add_subplot(1, 2, 1)
ax = sns.boxplot(data=x)

ax = fig.add_subplot(1, 2, 2)
ax = sns.violinplot(data=x)
```

Code 6.2: Code for box plot and violin plot

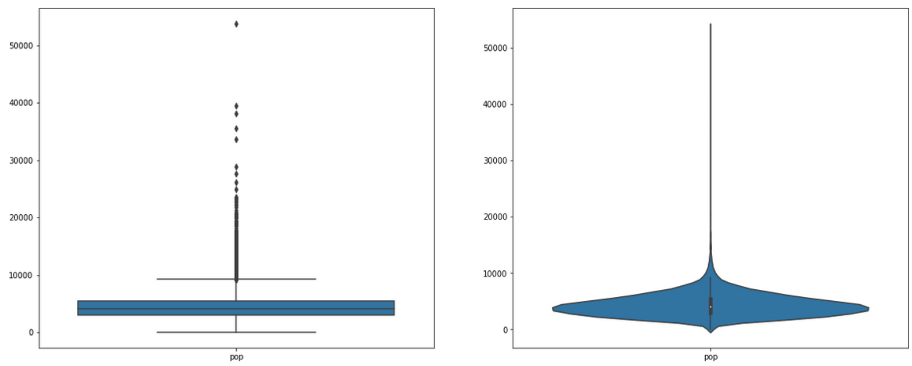


Image 6.2: Image for box plot and violin plot

Multivariant Analysis

Multivariant Analysis for this study is examined through following methods.

Scatter Plot

The main purpose of scatter diagram to find the relationship among two variables. The relationship code for rent_mean and family mean attribute are given below:

```
x1 = df_filtered.toPandas()['rent_mean'].values.tolist()
y1 = df_filtered.toPandas()['family_mean'].values.tolist()
plt.scatter(x1, y1, color='blue', s=30)
plt.xlabel('Rent Mean')
plt.ylabel('Family Income Mean')
plt.title('Scatter Plot')
```

Code 6.3: Code for scatter plot

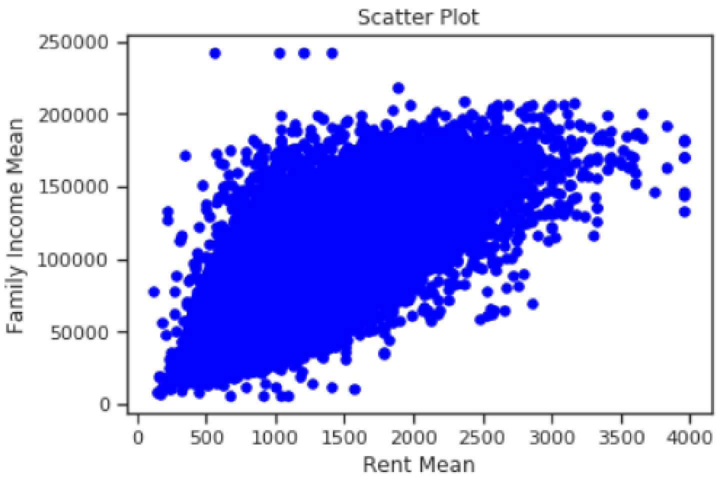


Image 6.3: Image for scatter plot

Pair plot

This describes the relationship between columns in pair. The following code describes the relationship between attributes such as rent_mean, family_mean, hi_mean, hc_mean,debt, pop in pairs.

```
: df_scatter_plot =df_filtered.select('rent_mean','family_mean','hi_mean','hc_mean','debt','pop')

  import seaborn as sns
  sns.set(style="ticks")

  #df = sns.load_dataset("iris")
  sns.pairplot(df_scatter_plot.toPandas())
  plt.show()
```

Code 6.4: Code for pair plot

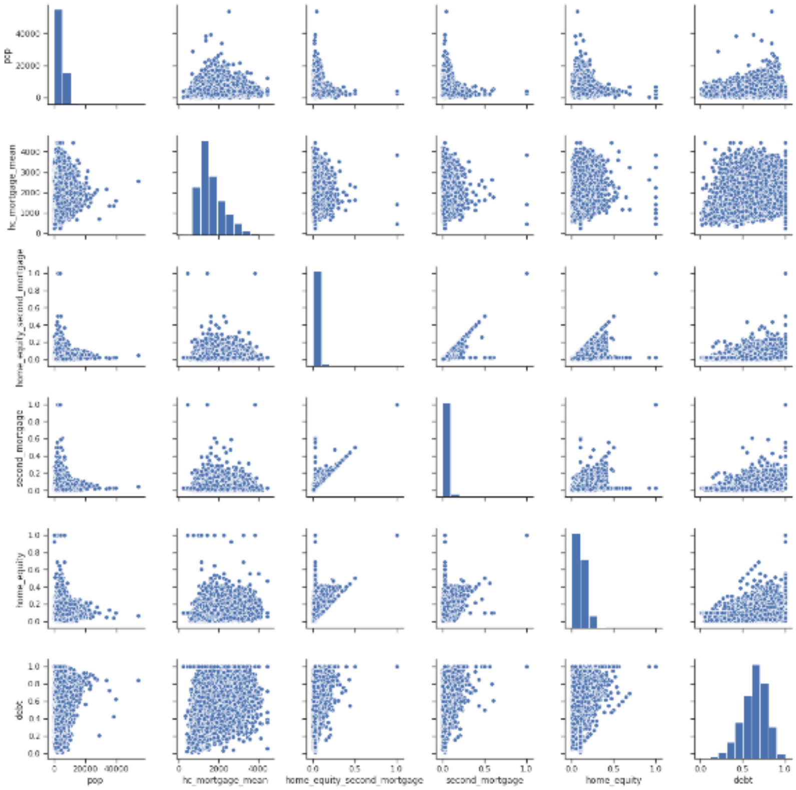


Image 6.4: Image for pair plot

Correlation Matrix

The purpose of adopting Correlation Matrix is to illustrate the correlation coefficients among attributes. The following code shows the correlation matrix used for the study.

```
df_filtered_num = df_filtered = df_filtered.drop('city','place','type','state')
from pyspark.mllib.stat import Statistics
import pandas as pd

corr_data = df_filtered_num.select(df_filtered.columns)

col_names = corr_data.columns
features = corr_data.rdd.map(lambda row: row[0:])
corr_mat=Statistics.corr(features, method="pearson")
corr_df = pd.DataFrame(corr_mat)
corr_df.index, corr_df.columns = col_names, col_names

print(corr_df.to_string())
```

Code 6.5: Code for Correlation Matrix

```
                              lat       lng     ALand    AWater       pop  male_pop  female_pop  rent_mean
hi_mean  family_mean  hc_mortgage_mean   hc_mean  home_equity_second_mortgage  second_mortgage  home_equity
debt  hs_degree  hs_degree_male  hs_degree_female  male_age_mean  female_age_mean   pct_own   married  separated
divorced
lat                          1.000000  0.016597  0.096512  0.068605 -0.084693 -0.078942   -0.087143   0.003081
0.135464     0.158540          0.103967  0.220807                     0.028534         0.031208      0.173697  0.1
87442   0.241920        0.230817           0.237396      -0.001090        -0.009688  0.059235  0.041529  -0.133976
-0.062394
lng                          0.016597  1.000000 -0.103957 -0.066076 -0.082556 -0.100318   -0.061673  -0.167260 -
0.059179    -0.030576         -0.093169  0.152501                    -0.092704        -0.096526     -0.001411 -0.1
21767   0.052967        0.032808           0.068969       0.083194         0.120893  0.092846 -0.029101   0.059554
-0.004055
ALand                        0.096512 -0.103957  1.000000  0.453650 -0.033301 -0.022235   -0.043189  -0.071597 -
0.030666    -0.030083         -0.059967 -0.059595                    -0.037598        -0.039967     -0.076836 -0.1
26573  -0.002365       -0.006210           0.003258       0.048304         0.020857  0.054824  0.032773  -0.015081
0.024463
AWater                       0.068605 -0.066076  0.453650  1.000000 -0.014546 -0.011122   -0.017432  -0.009885 -
0.004746    -0.004993         -0.010461 -0.010568                    -0.008577        -0.010282     -0.021858 -0.0
43135   0.003088        0.003302           0.002406       0.005508        -0.008449  0.006646 -0.003315   0.004945
0.004751
pop                         -0.084693 -0.082556 -0.033301 -0.014546  1.000000  0.980200    0.980708   0.162859
0.173757     0.135180          0.112578  0.057687                     0.040513         0.039337      0.082318  0.2
50429   0.049973        0.056771           0.040168      -0.187188        -0.189309  0.088790  0.172076  -0.110621
-0.173002
male_pop                    -0.078942 -0.100318 -0.022235 -0.011122  0.980200  1.000000    0.924482   0.159198
0.175758     0.134674          0.108919  0.048914                     0.036547         0.035607      0.079186  0.2
41752   0.032472        0.037597           0.032459      -0.200218        -0.192463  0.092628  0.141656  -0.106550
-0.158786
female_pop                  -0.087143 -0.061673 -0.043189 -0.017432  0.980708  0.924482    1.000000   0.160855
0.165231     0.131055          0.111925  0.064270                     0.042557         0.041232      0.082264  0.2
50154   0.064820        0.072779           0.046589      -0.167121        -0.178874  0.082111  0.193875  -0.109896
-0.178419
rent_mean                    0.003081 -0.167260 -0.071597 -0.009885  0.162859  0.159198    0.160855   1.000000
0.756261     0.704306          0.753979  0.598991                     0.098577         0.120470      0.399213  0.4
47770   0.361965        0.372463           0.327876       0.043401         0.002336  0.145634  0.259089  -0.152815
-0.381025
hi_mean                      0.135464 -0.059179 -0.030666 -0.004746  0.173757  0.175758    0.165231   0.756261
1.000000     0.962434          0.767821  0.678410                     0.004691         0.019627      0.431933  0.4
22547   0.583031        0.578457           0.553612       0.204826         0.134088  0.484489  0.539128  -0.272231
-0.401867
family_mean                  0.158540 -0.030576 -0.030083 -0.004993  0.135180  0.134674    0.131055   0.704306
0.962434     1.000000          0.762305  0.690572                    -0.015332        -0.001299      0.418693  0.3
81902   0.638509        0.630654           0.609573       0.259973         0.197151  0.452983  0.488418  -0.276927
-0.365943
hc_mortgage_mean             0.103967 -0.093169 -0.059967 -0.010461  0.112578  0.108919    0.111925   0.753979
0.767821     0.762305          1.000000  0.796640                     0.094534         0.126065      0.464798  0.4
02789   0.335707        0.350619           0.299612       0.094663         0.051857  0.069672  0.230679  -0.145905
-0.409700
```

Image 6.5: Image for Correlation Matrix

Location Analysis

Location Analysis is an exploration about different economic factors related to locations in a country. In this study, economic factors of different US locations are chosen to find the relationship among the factors and to predict the economic status of the country.

The study has consider specific cities from state and that is represented in the map (See the image 6.6 below). In this different colours used indicates states and the points show the cities.

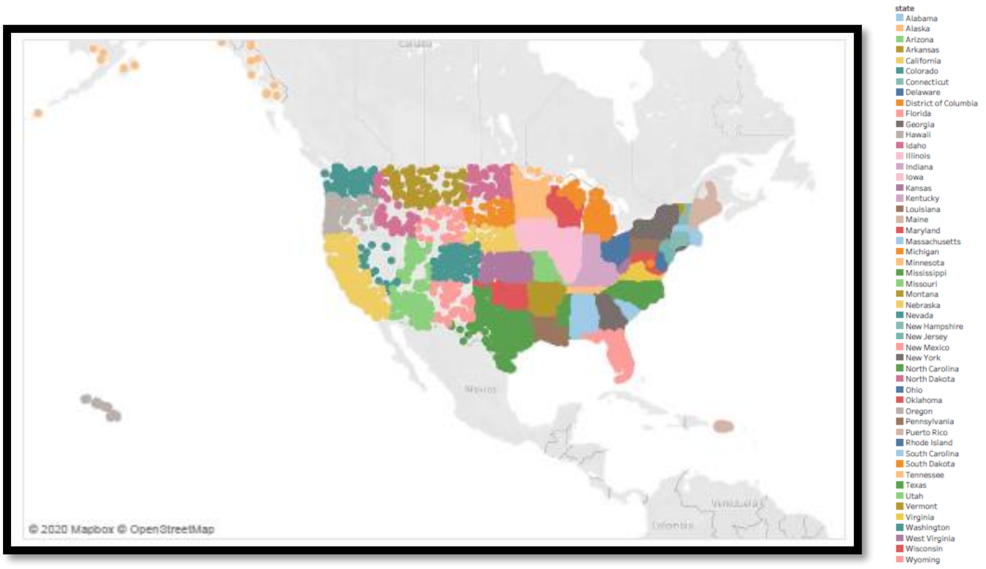


Image 6.6: Geographical location of state and cities in US

Link between state and population is researched in the geopraphical context. The different states in US are fitted in the map with the aid of actual longitude and Latitude. Shade of the colour illustrate the density of the popoulation from yellow to green.This is shown in the image 6.7 below:

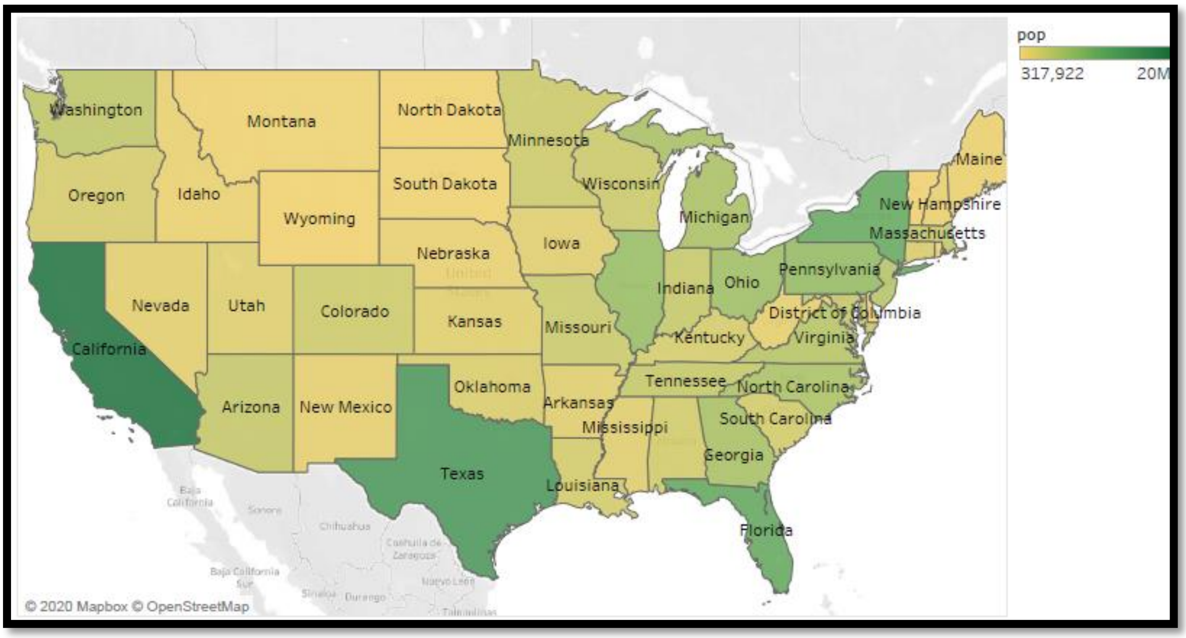
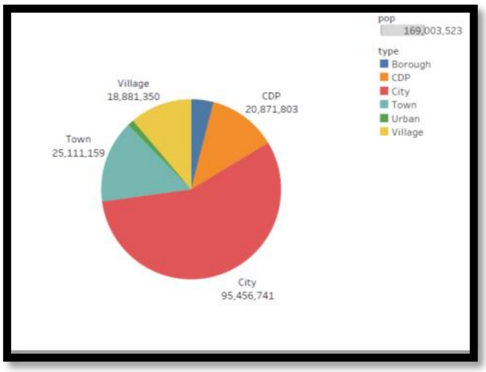


Image 6.7: State Vs Population

Later, the connection of male population and female population is analysed against the state. In this percentage of male or female population is represented by transparency of the dots. This is displayed in the **images** below:

Image 6.8 (b): State Vs male Population *Image 6.8 (b): State Vs female Population*

After this, the populations of US is analysed in different type of localities in US. This is shown below in the form of pie chart and bar chart.



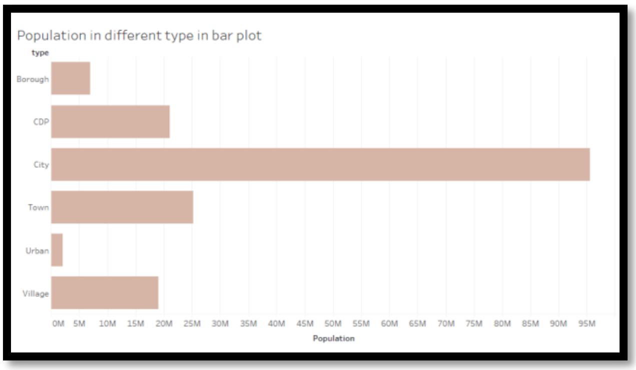


Image 6.9: Population in different type in US

Next, the connection between state and population attributes were explored and it is illustrated in the bar chart shown below:

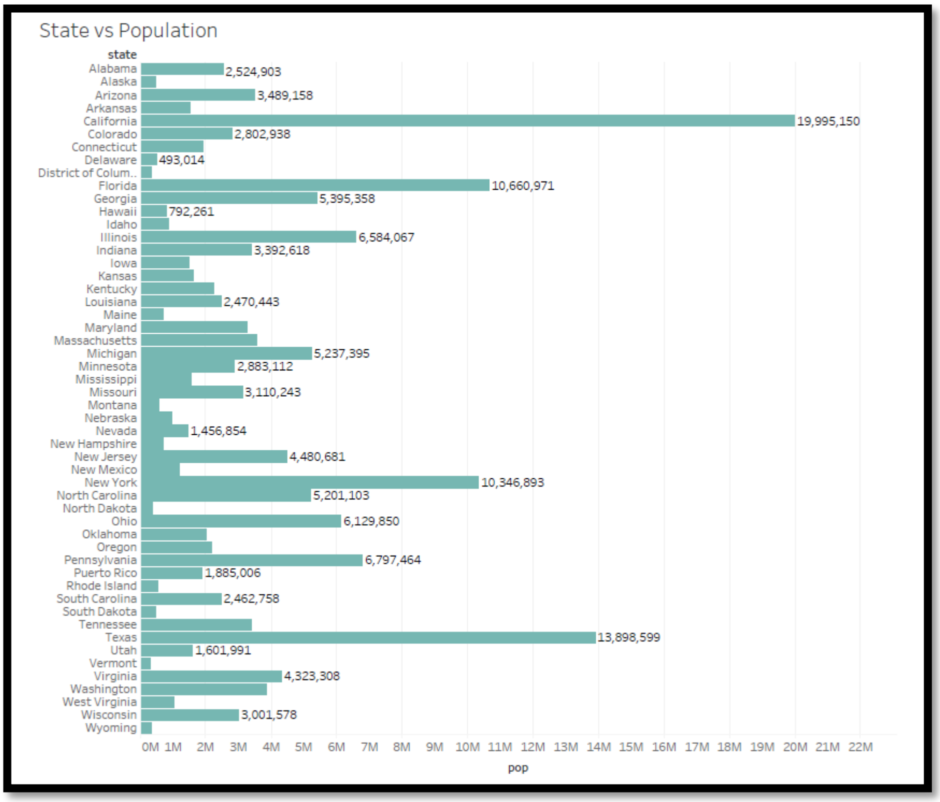


Image 6.10: State Vs Populations

Following this, linkage between type and city were analysed and this is shown in the packed bubble diagram below. In this different colours represents different types whereas the size of the bubble shows number of records of each city.

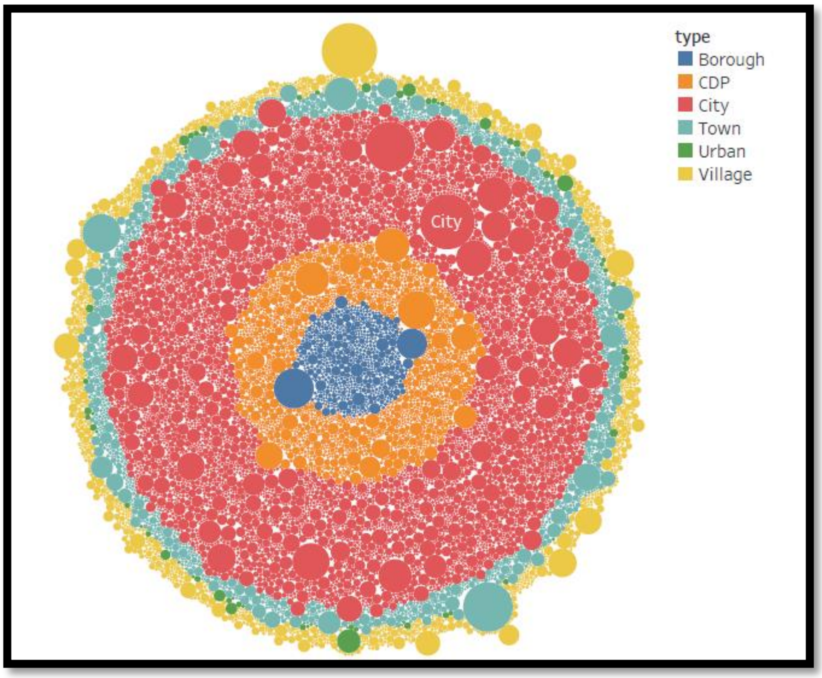


Image 6.11: Type Vs City

Then analysis has moved further advance through looking at the connections between three attributes which are state, pop and type. This is shown in the following bar chart where the x axis represents state, y axis represents population and the different colours illustrates the type.

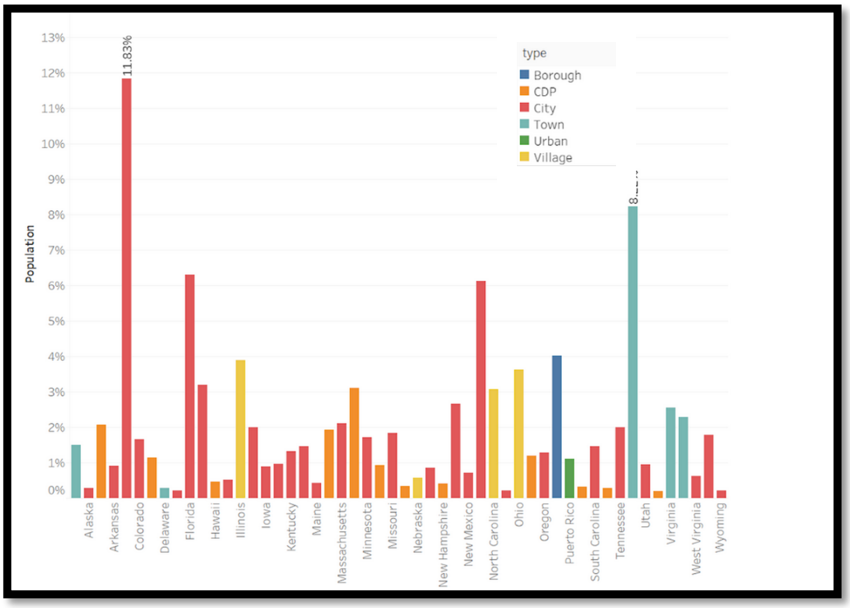


Image 6.12: State Vs Population Vs Type

Finally the study also explored number of records among the top 10 states. The image 6.13 shows the number of records against the state and image 6.14 illustrates the number if records against the city.

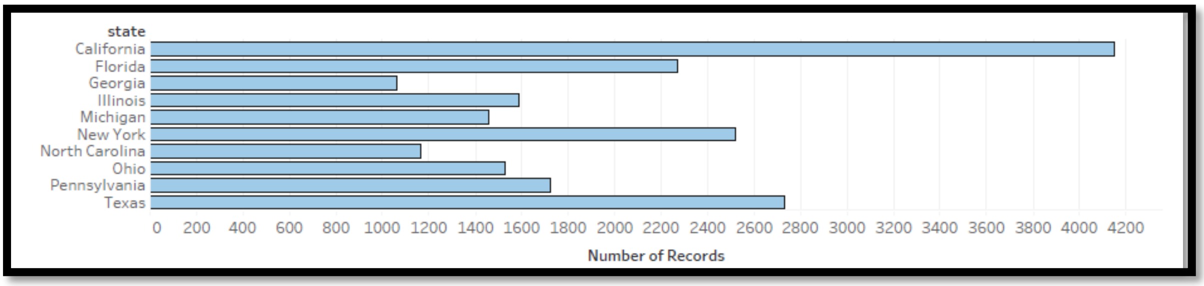


Image 6.13: State Vs Number of records

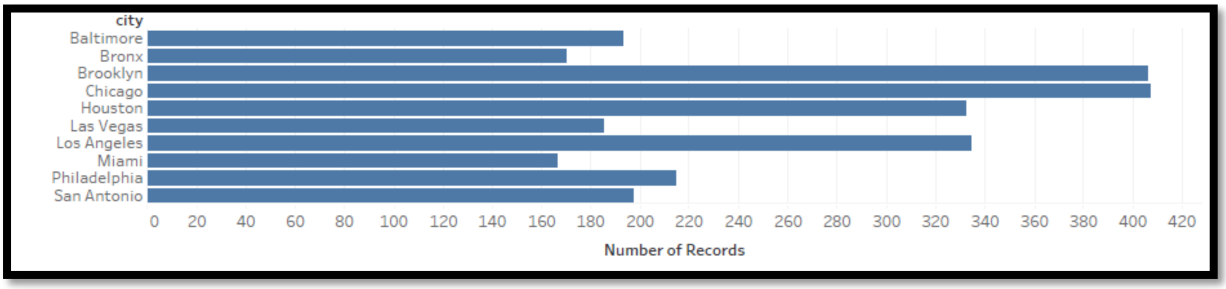


Image 6.14: City Vs Number of records

Debt Analysis

Generally families in a country purchase a house by getting mortgage or re mortgaging their existing properties. This put them under debt which will impact on economic growth. Analysing the factors that influence debt is done below visualisations.

Initially the density of debt in all US locations is explored in this study as shown in the image 6.15 below:

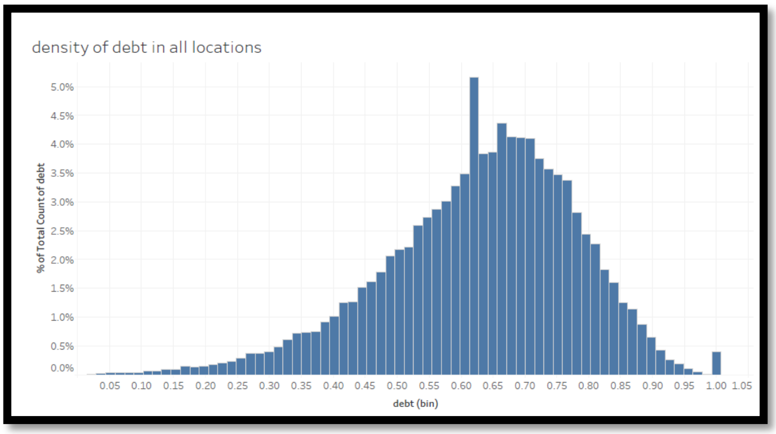


Image 6.15: Density of debt in US locations

Following that, debt against the top 10 cities were explored in this study. In this map the top 10 cities are represented by the squares and density of the debt is shown by colours.

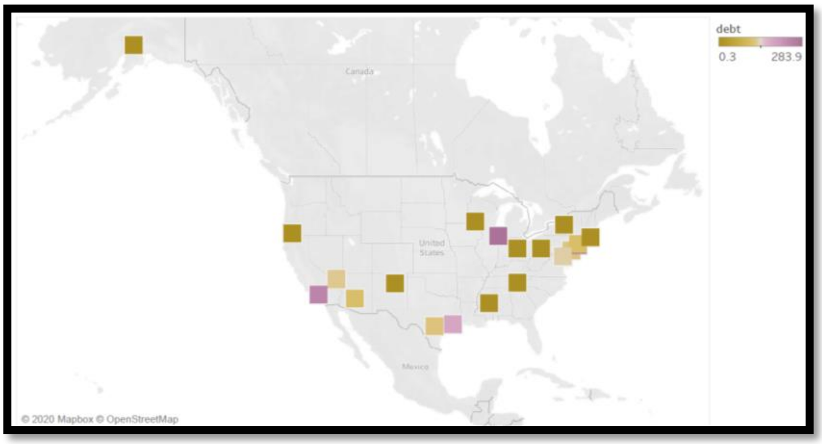


Image 6.16: Top 10 city Vs Debt

Similarly, debt against different states in US were analysed. In this map different states are represented by geographical locations and debt is illustrated by density of the colour.

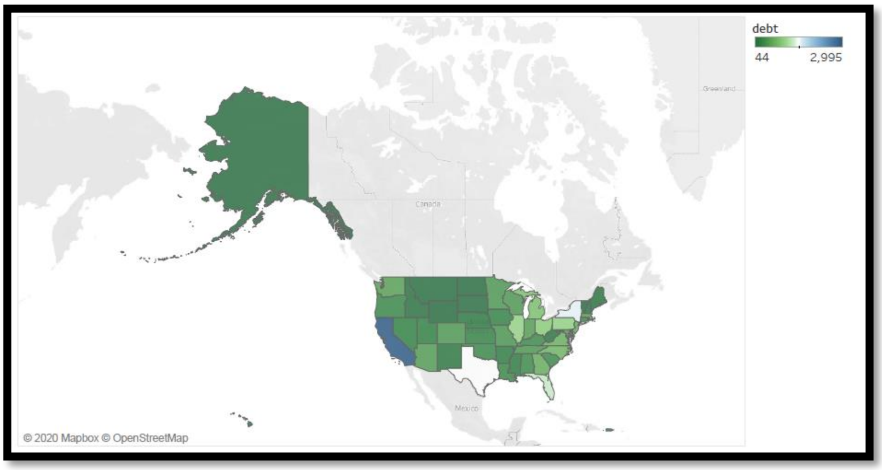


Image 6.17: State Vs Debt

Debt in this study represents the percentage of debt with is in each house. Hence, the connection between debt and types were analysed and the results produced is shown in the image 6.18 below:

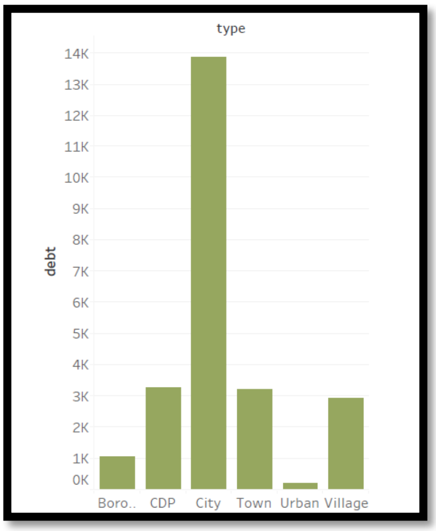


Image 6.18: Debt Vs Type

Later, the connections between family_mean, debt and states are explored through a scatter diagram. In this scatter diagram, family _mean is plotted in x-axis and debt values are plotted in y-axis. Moreover, different colours depicts the states.

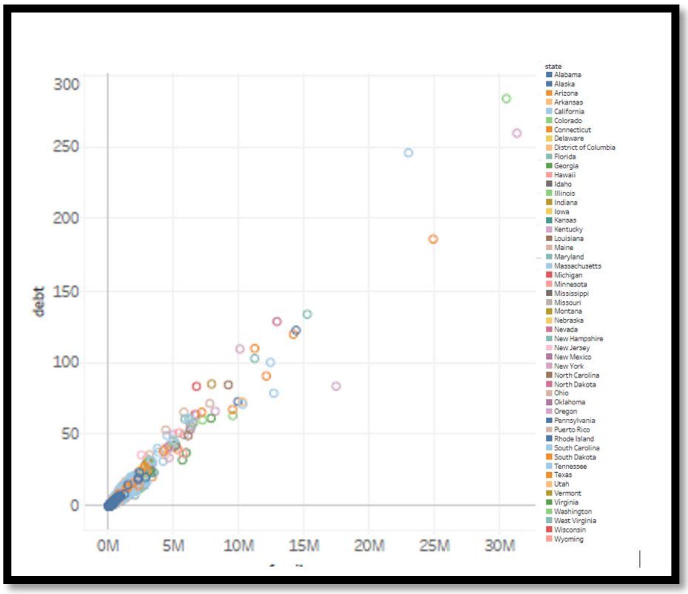


Image 6.19: Scatter diagram of family_mean and debt with states

Further the connection between state, home equity and type was researched. This is represented by the bar chart where x- axis and y- axis are depicted by the home equity and states accordingly. Types are denoted by different colours.

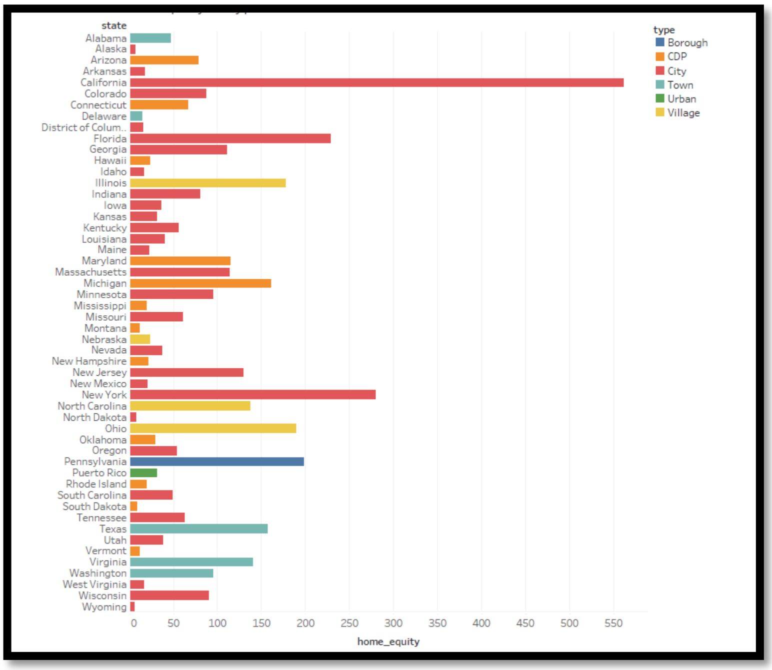


Image 6.20: State Vs Home equity Vs Type

Likewise, next the links between state, city and households incomes are analysed. In this top 10 cities that have higher household income values are adopted. In this x-axis and y-axis are shown by states and family income_mean. Moreover, colour illustrates the cities adopted for the study (See image 6.21 below).

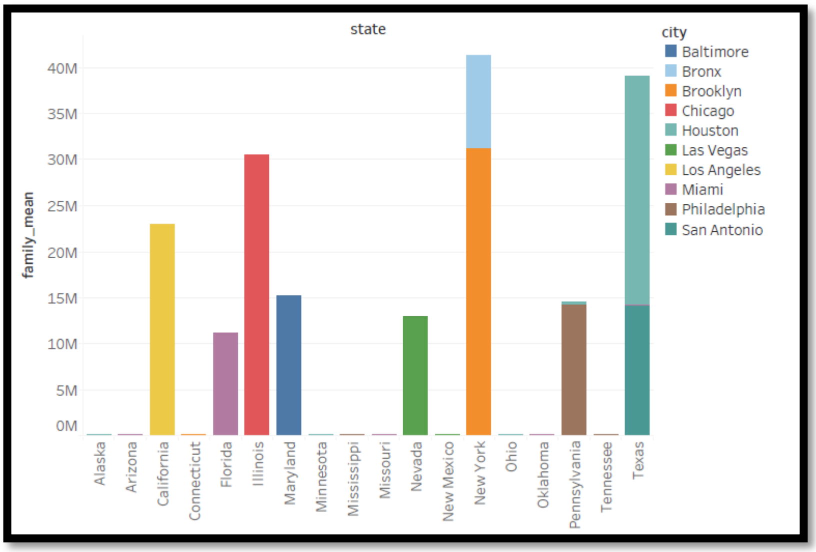


Image 6.21: State Vs City Vs Households Income

Next the relationship between state, city and family income was explored. Information collected are plotted on a bar chart where x-axis represents cities and y-axis represents family income_mean. Different states considered for the study is displayed through different colours (See the image 6.22 below).

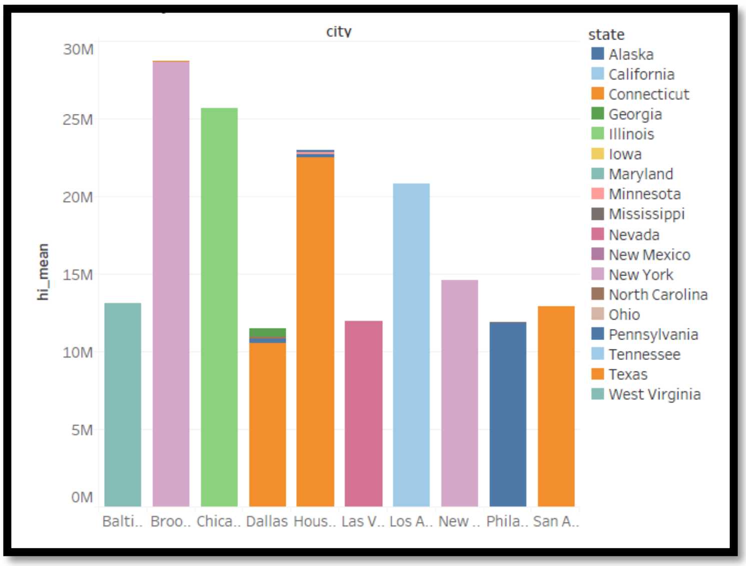


Image 6.22: State Vs City Vs Family Income

To explore further the study also has looked at the connection between state, city and home equity. The result is plotted in the bar chart where x-axis and y axis represent cities and home equity respectively. Different colours in the diagram shows the states considered for this investigation (See the image 6.23 below).

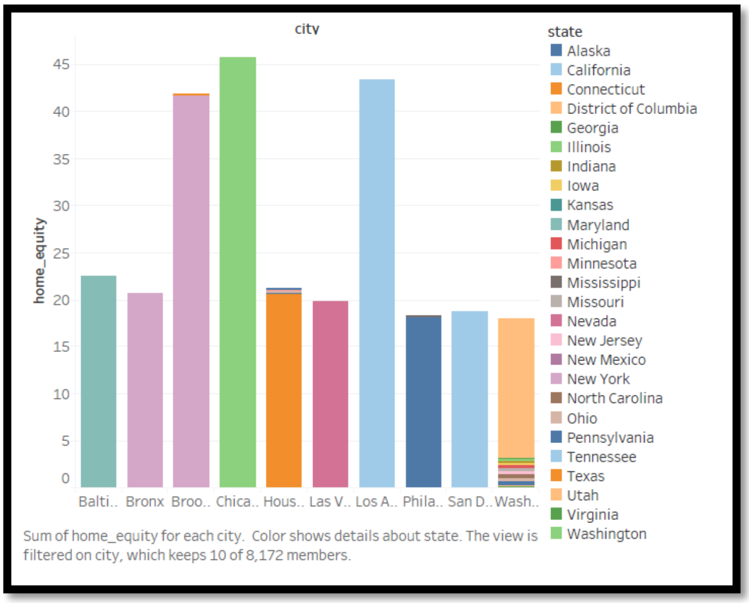


Image 6.23: State Vs City Vs Home equity

Next, to analyse more in detail the connection between state, city and percentage of owned houses are examined. As the above images this is also plotted in bar chart where x-axis and y-axis illustrates cities and percentage of owned houses accordingly. States are represented in different colours as shown in the image 6.24 below.

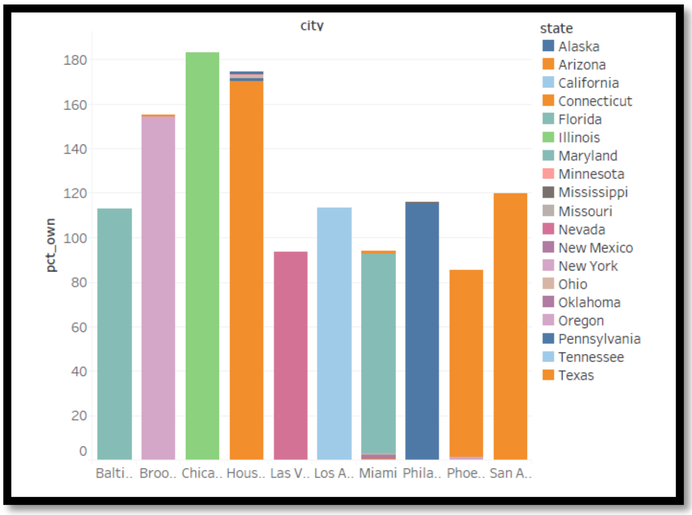


Image 6.24: State Vs City Vs Percentage of owned houses

Following this, the links between Higher school degree, city, population and state are analysed. In this, size of the circle shows the percentage of people completed high school and colour depicts the density of the populations. Moreover, the top 10 cities that have higher school completion is illustrted throug points. Furthermore, states of the country are shown through divisions in the map.

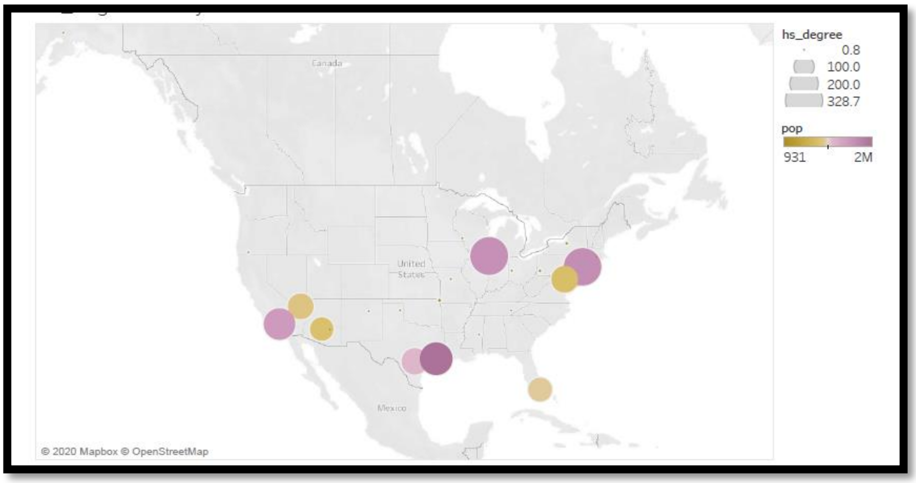


Image 6.25: State Vs City Vs high school degree Vs Populations

One of the other connections analysed in the study is between State and Percentage of both household costs and mortgage costs. States are illustrated through the geographical area and colour range shows the density of both household costs and mortgage costs (See the image 6.26 below).

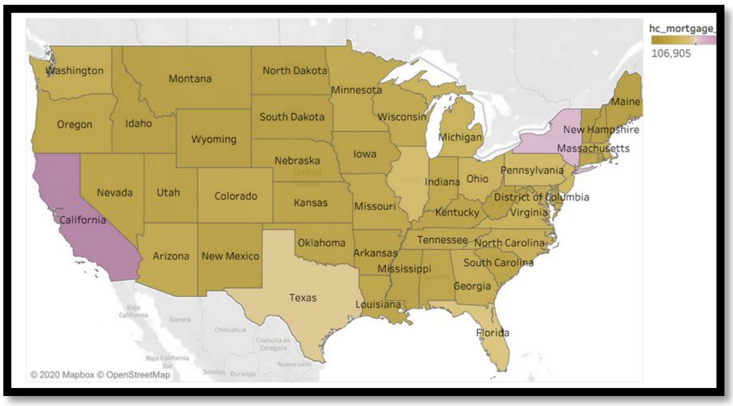


Image 6.26: State Vs Percentage of both household costs and Mortgage costs

Finally, the analysis among state, city and second mortgage were investigated for the study. In this results were plotted in a bar chart where x-axis and y-axis represent Cities and second mortgage accordingly. In addition, similar to other images states are illustrated through different colours.

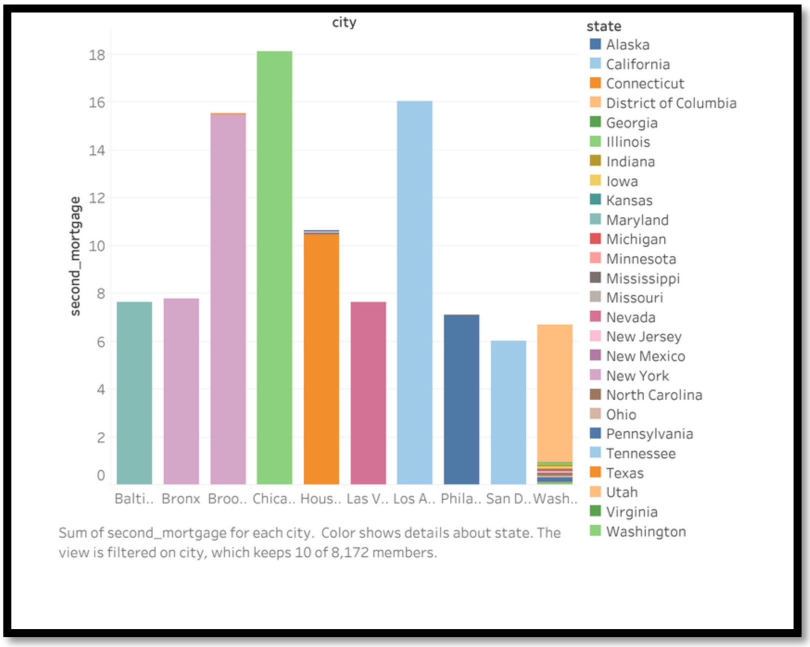


Image 6.27: State Vs City Vs Second Mortgage

Machine Learning -Linear Regression

Statistical Modelling is a wide area which comprises several analysis to achieve different goals. Regression Analysis is a one of them which is used to examine the relationship against a dependent variable and one or more independent variables. In this study pair plot and correlation matrix are adopted to find the linear regression between rent_mean and family_mean. To further confirm this Linear Regression which is a well-known machine learning technique is implemented to find the actual relationship between rent_mean and family_mean. The following code has been used for it.

In this study, the dataset has been split into 70% training dataset and 30% testing dataset. This is explained in Line 3 in the below code. Line 4, 5 and 6 illustrates the transformation through vector assembler to setup rent_mean, family_mean as features and labels. Only training dataset has been transformed.

```
from pyspark.ml.feature import VectorAssembler
data2 = df_filtered.select(df_filtered.rent_mean,df_filtered.family_mean.alias('label'))
train, test = data2.randomSplit([0.7,0.3])
assembler = VectorAssembler().setInputCols(['rent_mean',]).setOutputCol('features')
train01 = assembler.transform(train)
''' we only need features and label column '''
train02 = train01.select("features","label")
train02.show(truncate=False)

+-----------+------------+
|features   |label       |
+-----------+------------+
|[147.5481] |8725.80658  |
|[172.725]  |19414.83729 |
|[181.7723] |56197.76363 |
|[186.70202]|11950.5278  |
|[218.78415]|14311.34755 |
|[224.5]    |127218.56951|
|[232.18411]|18962.67058 |
|[233.43589]|10706.2618  |
|[236.80179]|11887.84042 |
|[236.943]  |12256.28786 |
|[237.90707]|31542.99902 |
|[244.81654]|10826.44645 |
|[245.61748]|13172.22162 |
|[255.05874]|27346.52613 |
|[261.66524]|22152.90647 |
|[263.51021]|23394.99095 |
|[263.55309]|26823.47547 |
|[268.25262]|15960.91515 |
|[268.56819]|28864.69211 |
|[268.82751]|21805.84134 |
+-----------+------------+
only showing top 20 rows
```

Code 7.1: Code for transforming training dataset into features and labels.

Following this, linear regression is fitted in the training dataset and model was created through the PySpark ML library (See Line 1 and 2). In addition, testing dataset was transformed to features and labels to apply the created model on it and to predict the output. This can be seen in the line 4,5 and 6 in the code below.

```
# Import LinearRegression class
from pyspark.ml.regression import LinearRegression
lr = LinearRegression()
model = lr.fit(train02)
test01 = assembler.transform(test)
test02 = test01.select('features', 'label')
test03 = model.transform(test02)
test03.show(truncate=False)

+-----------+-----------------+------------------+
|features   |label            |prediction        |
+-----------+-----------------+------------------+
|[117.15]   |78370.33168289994|31581.065553760563|
|[159.13436]|19242.02238      |33709.04918546362 |
|[166.08536]|7581.09788       |34061.36167096937 |
|[180.14139]|9663.8382        |34773.79367580943 |
|[205.38889]|48007.13653      |36053.46702896815 |
|[217.49822]|17785.85102      |36667.23025010314 |
|[224.5]    |133435.40993     |37022.116527583275|
```

Code 7.2: Code for creating model and transform testing dataset into features and labels.

Next the predicted values were evaluated through R^2 values and achieved approximately 0.5%. Higher the R^2 better the model.

```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator()
print(evaluator.evaluate(test03,{evaluator.metricName: "r2"}))

0.4997781471644356
```

Code 7.3: Code for evaluation through R^2 values

Later, this study has calculate the co-efficient and intercept value of the linear regression line.

```
print(model.coefficients)

[50.68515113016031]

print(model.intercept)

25643.300098862284
```

Code 7.4: Code for calculating co-efficient and intercept value

Discussion

The following discussions for this study are based on visualisations and machine learning algorithms derived. From the scatter plot (image 6.3) and pair plot (image 6.4) a positive linear relationship between rent_mean and family income_mean can be identified. Furthermore, this is conformed through the positive values of intercepts and co-efficient which are respectively 25643.3 and 50.68. Generally, when the R2 is high the linear regression model will perform well. However, in our study the value of R2 is 0.5 hence, it is an average performing linear regression model.

In the exploratory analysis the image 6.7 shows the relationship between state and population. From this is recognised that California is the state that has high populations. Interestingly, both female and male population are also high in California (Image 6.8 a and b). This study also analysed the connection between population and different types of locations. The result shows that more than 50% of the population lives in city whereas only around 3% of the population is living in the urban sides of the country (image 6.9). During the exploration among the localities types and locations, the results showed that there are more cities in US (Image 6.11).

After focusing in location and population the study has moved to investigate on factors related to debt. Firstly, the density of the debt in the entire US was analysed. Findings showed the distribution is skewed towards left and this shows that most of the parts in US are under debt (image 6.15). Moreover, images 6.16 and 6.17 depict that most debt city in US is Chicago and state is California. On the other hand, image 6.18 clearly shows that cities are the place where there is more debt. While exploring debt Vs family_mean a linear relationship was noticed which means when the debt is high the family_mean is also high. This is shown in image 6.19 which also indicates that debt and family_mean are greater in Illinosis state.

Another factor home equity was analysed against state and type which is shown in image 6.20 and 6.23. Findings from image 6.20 show home equity is greater in California State a lesser in Alaska. Surprisingly, both belongs to city locality. To further strengthen the study household income against cities and states were explored. From the image 6.21, it is identified that Broklyn city in New York State has the greater household income compare to the other states in US. Similarly, Broklyn city in New York State has greater family income (image 6.22). This study also looked into percentage of owned home versus state and city. The results showed that people in Chicago owned more homes in comparison to the other states (image 6.24).

Education is one of the key factors that contribute on the country's economy. In this study it was examined through looking at number of people who have completed the high school education. The findings shows Houston city in Texas State has more educated people. This has been identified through investigating the connection between State, city, high school degree and populations (image 6.25). Moreover, Chicago and Illinois have the second educated States in US. Furthermore, Brooklyn city in New York the next educated place.

During the analysis between State and percentage of household and mortgage cost, the findings depict that California has the highest cost rate whereas Texas has the lowest cost rates (image 6.26). Finally, as shown in the image 6.27 the relationship between percentage of secondary mortgage, city and state were analysed. The result shows that Chicago and Illinois have the highest percentage of secondary mortgages.

Conclusion

Every country needs to focus on their economic growth to survive among the other in the world. There are number of factors influences the economic growth. Due to the time and word limitation this study has only considered 29 factors that impacts on economic growth (eg: household income, debt, population, location, completion of high school etc).

The results from the visualisations and discussions show overall there is a positive relationship between rent_mean and family_mean. California which is one of the largest State in US has the highest populations, debt, home equity and household and mortgage cost. This clearly indicates that it is the most expensive place in US. On the other hand, Chicago is identified with highest debt, owned houses and secondary mortgages including second highest in secondary mortgages. This shows that people in this State are wealthy enough with the good education background. Conversely New York has greater household income and family. This illustrate that it’s wealthier in terms of assets.

Overall, this study concludes that each factors has its own impact on economic growth hence it is important to focus on all the factors individually.

References

Assefi.M, Behravesh.E, Liu.G and Tafti A. P,(2017) "*Big data machine learning using apache spark MLlib*," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, pp. 3492-3498

Drabas, T. and Lee, D (2017). ''*Learning PySpark''*. 1st Edition. Packt Publishing.

Feng.W. (2020). Learning Apache Spark with Python. Available: https://runawayhorse001.github.io/LearningApacheSpark/pyspark. pdf.. Last accessed 28th February 2020.

Frampton, M., (2014). ''*Big Data made easy''*, New York: Apress.

Gupta.A, Thakur H. K, Shrivastava.R, Kumar.P and Nag.S, (2017) "*A Big Data Analysis Framework Using Apache Spark and Deep Learning*," 2017 IEEE International Conference on Data Mining Workshops (ICDMW), New Orleans, LA, pp. 9-16.

Nandi.A.(2015). ''*Spark for Python Developers''*. Packt Publishing Ltd.

Sheshasaayee.A and Lakshmi. J. V. N,(2017) "An insight into tree based machine learning techniques for big data analytics using Apache Spark," *International Conference on Intelligent Computing*, Instrumentation and Control Technologies (ICICICT), Kannur, 2017, pp. 1740-1743.

Stančin.I and Jović.A,(2019) "*An overview and comparison of free Python libraries for data mining and big data analysis*," 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, pp. 977-982.

Appendix

Please see the link for to access the codes used for this study.

https://github.com/moorthysiva/BigDataAnalysisForPyspark