```python
import pandas as pd
from sklearn.linear_model import LinearRegression


df=pd.read_csv("loans-3.csv")

df
```

```
          paid_status  status  Age  gender      education
married_status  \
0                Paid       0   35    Male  Post Graduate
Married
1                Paid       0   32    Male  Self Employed
Unmarried
2                Paid       0   27    Male       Graduate
Unmarried
3                Paid       0   25    Male  Post Graduate
Unmarried
4                Paid       0   26    Male  Post Graduate
Unmarried
...               ...     ...  ...     ...            ...            .
..
3850             Paid       0   32    Male       Graduate
Married
3851  Pending Payment       1   31  Female  Self Employed
Married
3852             Paid       0   31    Male  Post Graduate
Unmarried
3853             Paid       0   25    Male  Post Graduate
Unmarried
3854             Paid       0   39    Male  Self Employed
Married

                         city        province agerange  salary  civil
\
0                      Daskroi         GUJARAT    25-35       6    569

1                       Guntur  ANDHRA PRADESH    25-35       4    734

2                       Rajkot         GUJARAT    25-35       5    641

3                     Vadodara         GUJARAT    25-35       5    800

4                       Bhokar     MAHARASHTRA    25-35       3    616

...                        ...             ...      ...     ...    ...

3850                    Dabhoi         GUJARAT    25-35       4    655
```

```
3851                    Mumbai     Maharashtra    25-35       5    505

3852  Chennai City Corporation      TAMIL NADU    25-35       4    777

3853                    Kalyan     MAHARASHTRA    25-35       4    739

3854                 Hyderabad       Telangana      >35       4    670


      appcount  phonegrade  simstrength
0            7           1            3
1            8           2            1
2            6           1            1
3            4           3            3
4            5           5            1
...        ...         ...          ...
3850         2           3            2
3851         4           3            5
3852         7           2            1
3853         6           4            2
3854         2           4            3

[3855 rows x 14 columns]
```

```
columns_to_drop=["city","province","paid_status","civil","agerange"]
data=df.drop(columns=columns_to_drop)

data
```

```
      status  Age  gender      education married_status  salary
appcount  \
0          0   35    Male  Post Graduate        Married       6
7
1          0   32    Male  Self Employed      Unmarried       4
8
2          0   27    Male       Graduate      Unmarried       5
6
3          0   25    Male  Post Graduate      Unmarried       5
4
4          0   26    Male  Post Graduate      Unmarried       3
5
...      ...  ...     ...            ...            ...     ...
...
3850       0   32    Male       Graduate        Married       4
2
3851       1   31  Female  Self Employed        Married       5
4
3852       0   31    Male  Post Graduate      Unmarried       4
7
3853       0   25    Male  Post Graduate      Unmarried       4
```

```
6
3854       0   39    Male  Self Employed        Married       4
2

      phonegrade  simstrength
0              1            3
1              2            1
2              1            1
3              3            3
4              5            1
...          ...          ...
3850           3            2
3851           3            5
3852           2            1
3853           4            2
3854           4            3

[3855 rows x 9 columns]
```

```
columns_to_convert=["gender","education","married_status",]
data=pd.get_dummies(data,columns=columns_to_convert)
data.head(20)
```

```
    status  Age  salary  appcount  phonegrade  simstrength
gender_Female  \
0        0   35       6         7           1            3
0
1        0   32       4         8           2            1
0
2        0   27       5         6           1            1
0
3        0   25       5         4           3            3
0
4        0   26       3         5           5            1
0
5        0   27       4         1           5            2
0
6        0   32       4         8           5            1
0
7        0   35       4         4           2            1
0
8        0   24       4         7           3            1
0
9        0   38       5         8           5            3
0
10       0   38       6         4           5            3
0
11       0   22       4         5           1            3
0
12       0   38       5         5           2            3
```

```
0
13       0   32       3         2           4            1
0
14       0   26       4         8           5            1
0
15       0   34       4         8           1            3
0
16       1   31       4         2           1            3
0
17       0   38       6         3           1            3
0
18       0   29       4         1           1            2
0
19       0   31       4         4           2            2
0

    gender_Male  education_10th-12th  education_Graduate  \
0             1                    0                   0
1             1                    0                   0
2             1                    0                   1
3             1                    0                   0
4             1                    0                   0
5             1                    0                   0
6             1                    0                   1
7             1                    0                   0
8             1                    1                   0
9             1                    0                   1
10            1                    0                   0
11            1                    0                   0
12            1                    0                   0
13            1                    0                   1
14            1                    0                   0
15            1                    0                   1
16            1                    1                   0
17            1                    0                   0
18            1                    0                   1
19            1                    0                   0

    education_Post Graduate  education_Self Employed
married_status_Married  \
0                         1                        0
1
1                         0                        1
0
2                         0                        0
0
3                         1                        0
0
4                         1                        0
0
```

```
5                           1                           0
0
6                           0                           0
0
7                           0                           1
0
8                           0                           0
0
9                           0                           0
1
10                          1                           0
0
11                          1                           0
0
12                          0                           1
1
13                          0                           0
1
14                          1                           0
0
15                          0                           0
0
16                          0                           0
1
17                          0                           1
1
18                          0                           0
0
19                          0                           1
0

    married_status_Unmarried
0                           0
1                           1
2                           1
3                           1
4                           1
5                           1
6                           1
7                           1
8                           1
9                           0
10                          1
11                          1
12                          0
13                          0
14                          1
15                          1
16                          0
17                          0
```

```
18                          1
19                          1
```

```
data.columns
```

```
Index(['status', 'Age', 'salary', 'appcount', 'phonegrade',
'simstrength',
       'gender_Female', 'gender_Male', 'education_10th-12th',
       'education_Graduate', 'education_Post Graduate',
       'education_Self Employed', 'married_status_Married',
       'married_status_Unmarried'],
      dtype='object')
```

```
x=data
y=df.civil
```

```
model = LinearRegression()
model.fit(x, y)
```

```
LinearRegression()
```

```
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Coefficients: [-100.03738565    0.23339197    3.37661757   -0.67993757
-0.34935567
   -1.20538408   -1.24512336    1.24512336   -6.1545514   -11.22846127
  -15.38336898   -8.72609276   -0.99801433    0.99801433]
Intercept: 672.1960890786604
```

```
x_new = [[0, 25, 4, 7, 3, 2, 0, 0, 1, 1, 1,1,1,0]]
y_new = model.predict(x_new)
print("Predicted value:", y_new)
```

```
Predicted value: [640.82847169]
```

```
c:\Users\ayann\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:420: UserWarning: X does not have valid
feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(data, df.civil,
test_size= 0.25, random_state=42)
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
```

```
X_train, X_test, y_train, y_test = train_test_split(x,y,
test_size=0.25, random_state=42)
```

```
model.fit(X_train, y_train)
```

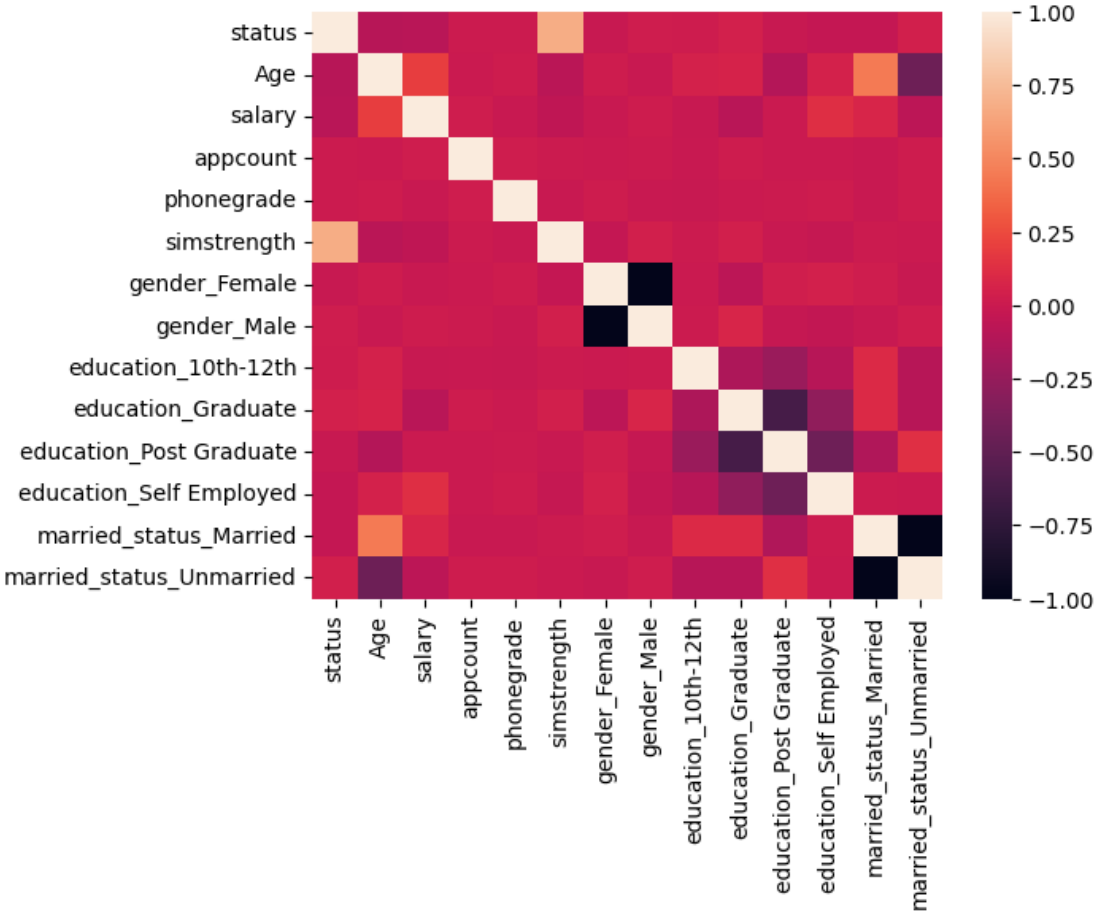
```
LinearRegression()
```

```
y_pred = model.predict(X_test)

# evaluate the performance of the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('Mean squared error:', mse)
print('Mean absolute error:', mae)
print('R-squared:', r2)
```

```
Mean squared error: 4670.214259441586
Mean absolute error: 58.61850262846409
R-squared: 0.21992055771651742
```

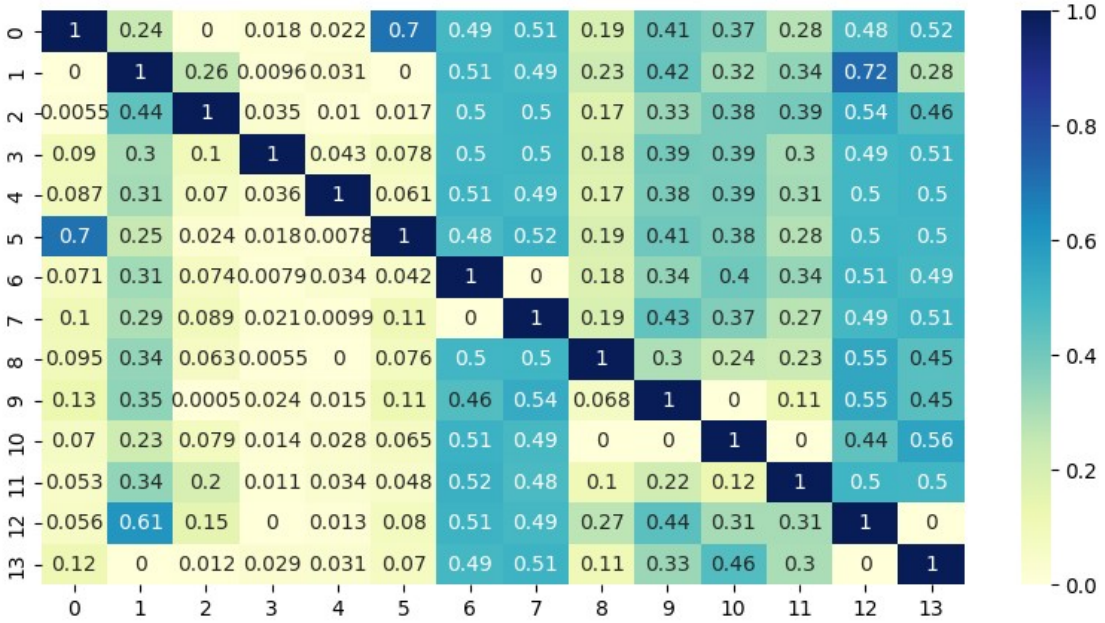
```
import seaborn as sns
sns.heatmap(x.corr())
```

```
<AxesSubplot: >
```



```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import MinMaxScaler
corr_matrix=x.corr()

# Scale the data to a range between 0 and 1
scaler = MinMaxScaler()
scaled_corr_matrix = scaler.fit_transform(corr_matrix)

# Plot the heatmap
fig, ax = plt.subplots(figsize=(10, 5))
sns.heatmap(scaled_corr_matrix, annot=True, cmap="YlGnBu")
```

```
<AxesSubplot: >
```



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

#importing datasets
df= pd.read_csv('loans-3.csv')
df.head(20)
# data = {'civil'}
# df = pd.DataFrame(data)
# bins = [300, 550, 650,750,900, float('inf')]
# labels = ['poor', 'average', 'good','excellent']
# df['credit_score'] = pd.cut(df['civil'], bins=bins, labels=labels)
```

```
        paid_status  status  Age gender      education married_status
\
0              Paid       0   35   Male  Post Graduate        Married

1              Paid       0   32   Male  Self Employed      Unmarried

2              Paid       0   27   Male       Graduate      Unmarried

3              Paid       0   25   Male  Post Graduate      Unmarried

4              Paid       0   26   Male  Post Graduate      Unmarried

5              Paid       0   27   Male  Post Graduate      Unmarried

6              Paid       0   32   Male       Graduate      Unmarried

7              Paid       0   35   Male  Self Employed      Unmarried

8              Paid       0   24   Male      10th-12th      Unmarried

9              Paid       0   38   Male       Graduate        Married

10             Paid       0   38   Male  Post Graduate      Unmarried

11             Paid       0   22   Male  Post Graduate      Unmarried

12             Paid       0   38   Male  Self Employed        Married

13             Paid       0   32   Male       Graduate        Married

14             Paid       0   26   Male  Post Graduate      Unmarried

15             Paid       0   34   Male       Graduate      Unmarried
```

16	Pending Payment	1	31	Male	10th-12th	Married	
17	Paid	0	38	Male	Self Employed	Married	
18	Paid	0	29	Male	Graduate	Unmarried	
19	Paid	0	31	Male	Self Employed	Unmarried	

	city	province	agerange	salary	civil	appcount
0	Daskroi	GUJARAT	25-35	6	569	7
1	Guntur	ANDHRA PRADESH	25-35	4	734	8
2	Rajkot	GUJARAT	25-35	5	641	6
3	Vadodara	GUJARAT	25-35	5	800	4
4	Bhokar	MAHARASHTRA	25-35	3	616	5
5	Pune	Maharashtra	25-35	4	707	1
6	Guntur	ANDHRA PRADESH	25-35	4	703	8
7	Dindigul	TAMIL NADU	25-35	4	797	4
8	Malad East	MAHARASHTRA	<25	4	565	7
9	Coimbatore North	TAMIL NADU	>35	5	595	8
10	Mumbai	Maharashtra	>35	6	584	4
11	Bhubaneswar	ODISHA	<25	4	612	5
12	Ahmadabad City	GUJARAT	>35	5	663	5
13	Ahmedabad	Gujarat	25-35	3	771	2
14	Bangalore South	KARNATAKA	25-35	4	597	8
15	Mettur	TAMIL NADU	25-35	4	722	8
16	Malad West	MAHARASHTRA	25-35	4	549	2
17	Thane	MAHARASHTRA	>35	6	642	3
18	Surat City	GUJARAT	25-35	4	587	1

```
19         Srirangam      TAMIL NADU    25-35       4    595         4


    phonegrade  simstrength
0            1            3
1            2            1
2            1            1
3            3            3
4            5            1
5            5            2
6            5            1
7            2            1
8            3            1
9            5            3
10           5            3
11           1            3
12           2            3
13           4            1
14           5            1
15           1            3
16           1            3
17           1            3
18           1            2
19           2            2
```

```
df.head(20)
```

```
        paid_status  status  Age gender      education married_status
\
0              Paid       0   35   Male  Post Graduate        Married

1              Paid       0   32   Male  Self Employed      Unmarried

2              Paid       0   27   Male       Graduate      Unmarried

3              Paid       0   25   Male  Post Graduate      Unmarried

4              Paid       0   26   Male  Post Graduate      Unmarried

5              Paid       0   27   Male  Post Graduate      Unmarried

6              Paid       0   32   Male       Graduate      Unmarried

7              Paid       0   35   Male  Self Employed      Unmarried

8              Paid       0   24   Male      10th-12th      Unmarried

9              Paid       0   38   Male       Graduate        Married
```

10	Paid	0	38	Male	Post Graduate	Unmarried	
11	Paid	0	22	Male	Post Graduate	Unmarried	
12	Paid	0	38	Male	Self Employed	Married	
13	Paid	0	32	Male	Graduate	Married	
14	Paid	0	26	Male	Post Graduate	Unmarried	
15	Paid	0	34	Male	Graduate	Unmarried	
16	Pending Payment	1	31	Male	10th-12th	Married	
17	Paid	0	38	Male	Self Employed	Married	
18	Paid	0	29	Male	Graduate	Unmarried	
19	Paid	0	31	Male	Self Employed	Unmarried	

	city	province	agerange	salary	civil	appcount
0	Daskroi	GUJARAT	25-35	6	569	7
1	Guntur	ANDHRA PRADESH	25-35	4	734	8
2	Rajkot	GUJARAT	25-35	5	641	6
3	Vadodara	GUJARAT	25-35	5	800	4
4	Bhokar	MAHARASHTRA	25-35	3	616	5
5	Pune	Maharashtra	25-35	4	707	1
6	Guntur	ANDHRA PRADESH	25-35	4	703	8
7	Dindigul	TAMIL NADU	25-35	4	797	4
8	Malad East	MAHARASHTRA	<25	4	565	7
9	Coimbatore North	TAMIL NADU	>35	5	595	8
10	Mumbai	Maharashtra	>35	6	584	4
11	Bhubaneswar	ODISHA	<25	4	612	5

12	Ahmadabad City	GUJARAT	>35	5	663	5
13	Ahmedabad	Gujarat	25-35	3	771	2
14	Bangalore South	KARNATAKA	25-35	4	597	8
15	Mettur	TAMIL NADU	25-35	4	722	8
16	Malad West	MAHARASHTRA	25-35	4	549	2
17	Thane	MAHARASHTRA	>35	6	642	3
18	Surat City	GUJARAT	25-35	4	587	1
19	Srirangam	TAMIL NADU	25-35	4	595	4

	phonegrade	simstrength
0	1	3
1	2	1
2	1	1
3	3	3
4	5	1
5	5	2
6	5	1
7	2	1
8	3	1
9	5	3
10	5	3
11	1	3
12	2	3
13	4	1
14	5	1
15	1	3
16	1	3
17	1	3
18	1	2
19	2	2

```
def convert_attribute(civil):
  if civil<=550:
    return 'POOR'
  elif civil<=650:
    return 'AVERAGE'
  elif civil<=750:
    return 'GOOD'
  else:
    return 'EXCELLENT'
```

```
df['credit']=df['civil'].apply(convert_attribute)
df.to_csv("update_loan.csv",index=False)

data= pd.read_csv('update_loan.csv')
data.head(20)
```

```
        paid_status  status  Age gender      education married_status
\
0              Paid       0   35   Male  Post Graduate        Married

1              Paid       0   32   Male  Self Employed      Unmarried

2              Paid       0   27   Male       Graduate      Unmarried

3              Paid       0   25   Male  Post Graduate      Unmarried

4              Paid       0   26   Male  Post Graduate      Unmarried

5              Paid       0   27   Male  Post Graduate      Unmarried

6              Paid       0   32   Male       Graduate      Unmarried

7              Paid       0   35   Male  Self Employed      Unmarried

8              Paid       0   24   Male      10th-12th      Unmarried

9              Paid       0   38   Male       Graduate        Married

10             Paid       0   38   Male  Post Graduate      Unmarried

11             Paid       0   22   Male  Post Graduate      Unmarried

12             Paid       0   38   Male  Self Employed        Married

13             Paid       0   32   Male       Graduate        Married

14             Paid       0   26   Male  Post Graduate      Unmarried

15             Paid       0   34   Male       Graduate      Unmarried

16  Pending Payment       1   31   Male      10th-12th        Married

17             Paid       0   38   Male  Self Employed        Married

18             Paid       0   29   Male       Graduate      Unmarried

19             Paid       0   31   Male  Self Employed      Unmarried
```

	city	province	agerange	salary	civil	appcount
0	Daskroi	GUJARAT	25-35	6	569	7
1	Guntur	ANDHRA PRADESH	25-35	4	734	8
2	Rajkot	GUJARAT	25-35	5	641	6
3	Vadodara	GUJARAT	25-35	5	800	4
4	Bhokar	MAHARASHTRA	25-35	3	616	5
5	Pune	Maharashtra	25-35	4	707	1
6	Guntur	ANDHRA PRADESH	25-35	4	703	8
7	Dindigul	TAMIL NADU	25-35	4	797	4
8	Malad East	MAHARASHTRA	<25	4	565	7
9	Coimbatore North	TAMIL NADU	>35	5	595	8
10	Mumbai	Maharashtra	>35	6	584	4
11	Bhubaneswar	ODISHA	<25	4	612	5
12	Ahmadabad City	GUJARAT	>35	5	663	5
13	Ahmedabad	Gujarat	25-35	3	771	2
14	Bangalore South	KARNATAKA	25-35	4	597	8
15	Mettur	TAMIL NADU	25-35	4	722	8
16	Malad West	MAHARASHTRA	25-35	4	549	2
17	Thane	MAHARASHTRA	>35	6	642	3
18	Surat City	GUJARAT	25-35	4	587	1
19	Srirangam	TAMIL NADU	25-35	4	595	4

	phonegrade	simstrength	credit
0	1	3	AVERAGE
1	2	1	GOOD
2	1	1	AVERAGE

```
3            3            3  EXCELLENT
4            5            1    AVERAGE
5            5            2       GOOD
6            5            1       GOOD
7            2            1  EXCELLENT
8            3            1    AVERAGE
9            5            3    AVERAGE
10           5            3    AVERAGE
11           1            3    AVERAGE
12           2            3       GOOD
13           4            1  EXCELLENT
14           5            1    AVERAGE
15           1            3       GOOD
16           1            3       POOR
17           1            3    AVERAGE
18           1            2    AVERAGE
19           2            2    AVERAGE
```

Shape of the dataset

```
data.shape
```

```
(3855, 15)
```

Looking at the datatypes of the data

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3855 entries, 0 to 3854
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   paid_status     3855 non-null   object
 1   status          3855 non-null   int64 
 2   Age             3855 non-null   int64 
 3   gender          3855 non-null   object
 4   education       3833 non-null   object
 5   married_status  3855 non-null   object
 6   city            3767 non-null   object
 7   province        3855 non-null   object
 8   agerange        3855 non-null   object
 9   salary          3855 non-null   int64 
 10  civil           3855 non-null   int64 
 11  appcount        3855 non-null   int64 
 12  phonegrade      3855 non-null   int64 
 13  simstrength     3855 non-null   int64 
 14  credit          3855 non-null   object
dtypes: int64(7), object(8)
memory usage: 451.9+ KB
```

```
data.paid_status = data.paid_status.astype('category')
data.gender = data.gender.astype('category')
data.education    = data.education     .astype('category')
data.married_status    = data.married_status.astype('category')
data.city = data.city.astype('category')
data.province = data.province.astype('category')
data.credit = data.credit.astype('category')


data.paid_status
```

```
0                  Paid
1                  Paid
2                  Paid
3                  Paid
4                  Paid
              ...
3850               Paid
3851    Pending Payment
3852               Paid
3853               Paid
3854               Paid
Name: paid_status, Length: 3855, dtype: category
Categories (2, object): ['Paid', 'Pending Payment']
```

```
### Looking at the modified datatypes of the data

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3855 entries, 0 to 3854
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   paid_status     3855 non-null   category
 1   status          3855 non-null   int64
 2   Age             3855 non-null   int64
 3   gender          3855 non-null   category
 4   education       3833 non-null   category
 5   married_status  3855 non-null   category
 6   city            3767 non-null   category
 7   province        3855 non-null   category
 8   agerange        3855 non-null   object
 9   salary          3855 non-null   int64
 10  civil           3855 non-null   int64
 11  appcount        3855 non-null   int64
 12  phonegrade      3855 non-null   int64
 13  simstrength     3855 non-null   int64
 14  credit          3855 non-null   category
```

```
dtypes: category(7), int64(7), object(1)
memory usage: 292.8+ KB
```

```
data.columns
```

```
Index(['paid_status', 'status', 'Age', 'gender', 'education',
'married_status',
       'city', 'province', 'agerange', 'salary', 'civil', 'appcount',
       'phonegrade', 'simstrength', 'credit'],
      dtype='object')
```

```
# columns_to_drop=["city","province","paid_status"]
# data=df.drop(columns=columns_to_drop)
# data.head()

# ### A function to remove the '_' in the data

# def removeUnderscore(value):
#     first_index = 0
#     last_index = len(value) - 1
#     while first_index <= last_index:
#         if value[first_index] == '_':
#             first_index += 1
#         if value[last_index] == '_':
#             last_index -= 1
#         if '_' not in value[first_index : last_index + 1]:
#             if value[first_index : last_index + 1] == '':
#                 return 0
#             else:
#                 return value[first_index : last_index + 1]


# def modifyData(columns):
#     for each_column in columns:
#         data1 = [str(value) for value in list(data[each_column])]
#         new_data = []
#         for value in data:
#             if value == 'nan':
#                 new_data.append(float('nan'))
#             else:
#                 new_data.append(float(removeUnderscore(value)))

#         data[each_column] = new_data

# modifyData([ 'Age', 'gender', 'education', 'married_status',
#        'city', 'province', 'agerange', 'salary', 'civil',
'appcount',
#        'phonegrade', 'simstrength', 'credit'])
```

```
### Missing data by columns in the dataset

data.isnull().sum().sort_values(ascending = False)
```

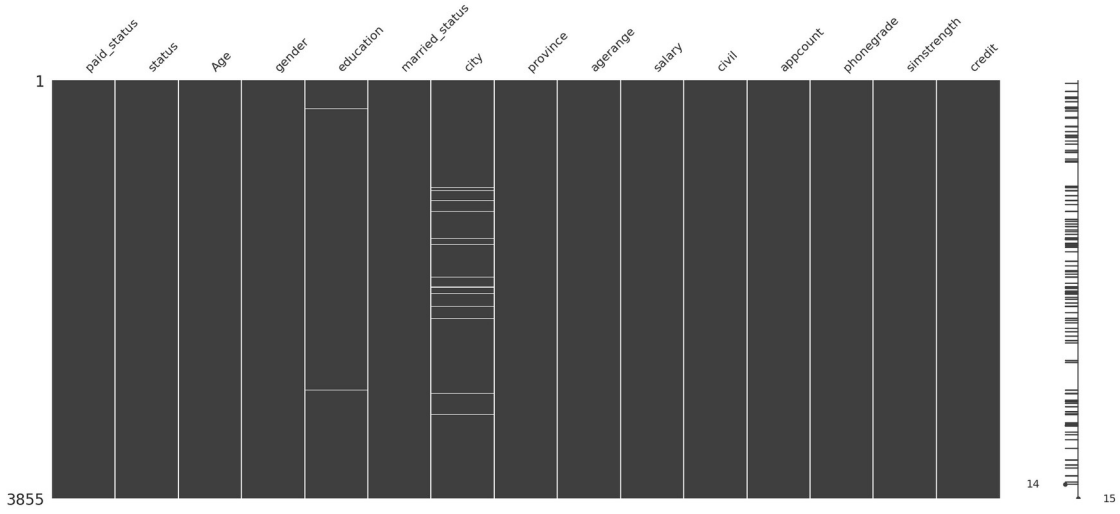
```
city              88
education         22
paid_status        0
status             0
Age                0
gender             0
married_status     0
province           0
agerange           0
salary             0
civil              0
appcount           0
phonegrade         0
simstrength        0
credit             0
dtype: int64
```

```
data.columns
```

```
Index(['paid_status', 'status', 'Age', 'gender', 'education',
'married_status',
       'city', 'province', 'agerange', 'salary', 'civil', 'appcount',
       'phonegrade', 'simstrength', 'credit'],
      dtype='object')
```

```
### Visual representation of the missing data in the dataset
import missingno
missingno.matrix(data)
```

```
<Axes: >
```



```
### Summary statistics of the numerical columns in the dataset

data.describe()
```

```
            status          Age       salary        civil     appcount
\
count  3855.000000  3855.000000  3855.000000  3855.000000  3855.000000

mean      0.169390    29.603891     4.238651   658.395850     4.471336

std       0.375145     4.503131     0.628513    78.510868     2.289083

min       0.000000    22.000000     2.000000   500.000000     1.000000

25%       0.000000    26.000000     4.000000   594.000000     2.000000

50%       0.000000    29.000000     4.000000   649.000000     4.000000

75%       0.000000    33.000000     4.000000   726.000000     6.000000

max       1.000000    41.000000     6.000000   800.000000     8.000000


        phonegrade  simstrength
count  3855.000000  3855.000000
mean      3.001038     2.337484
std       1.420072     1.100306
min       1.000000     1.000000
25%       2.000000     1.000000
50%       3.000000     2.000000
75%       4.000000     3.000000
max       5.000000     5.000000
```

Feature Analysis

```
### Value counts of the column - education

education_count = data['education'].value_counts(dropna = False)
education_count
```

```
Post Graduate    1914
Graduate         1101
Self Employed     623
10th-12th         195
NaN                22
Name: education, dtype: int64
```

```
# ### Distribution of Credit_Score for each Occupation
# from collections import Counter
# from collections import OrderedDict
```

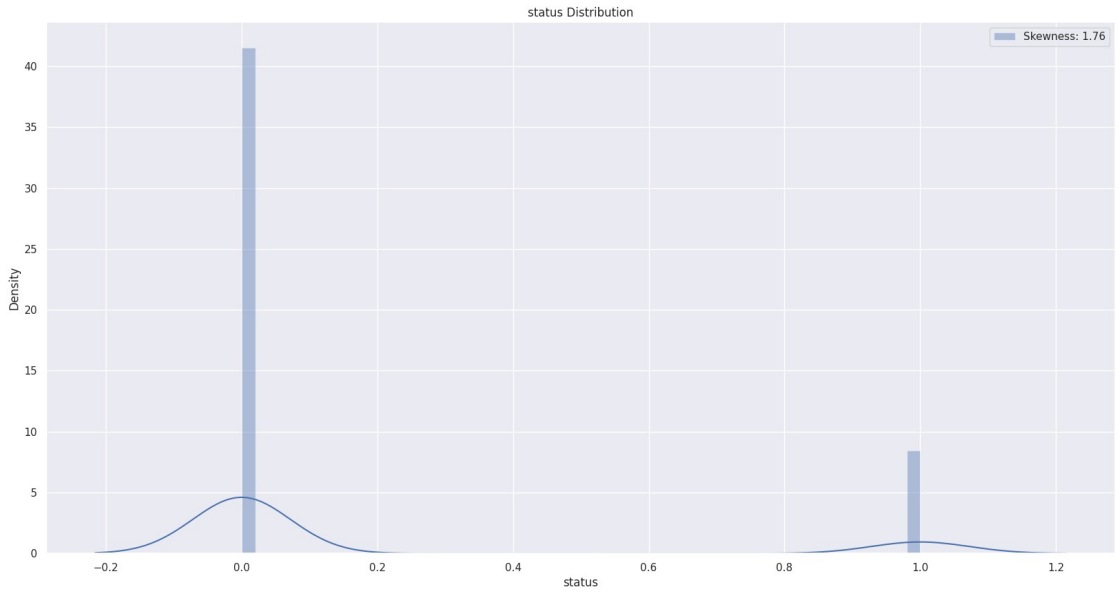
```
# ### Data Visualization

# import matplotlib.pyplot as plt
# sns.factorplot('Credit', col = 'education', data = data, kind =
'count', col_wrap = 4)


# sns.set(rc = {'figure.figsize': (20, 10)})
# sns.barplot(education_count.index, education_count.values)
# plt.title('Bar graph showing the value counts of the column -
education')
# plt.ylabel('Count', fontsize = 12)
# plt.xlabel('Occupation', fontsize = 12)

### Fetching the not null data of the column - Type of Data

index_values = data['education'].isnull().values
education_type_data = list(data['education'][index_values])
education_type_data

[nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan,
 nan]

### Understanding the distribution of the column - Age

sns.distplot(data['Age'], label = 'Skewness: %.2f'%
(data['Age'].skew()))
plt.legend(loc = 'best')
plt.title('Age Distribution')
```

```
<ipython-input-225-5c8354648af3>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Age'], label = 'Skewness: %.2f'%
(data['Age'].skew()))

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


```
### Understanding the distribution of the column - status

sns.distplot(data['status'], label = 'Skewness: %.2f'%
(data['status'].skew()))
plt.legend(loc = 'best')
plt.title('status Distribution')

<ipython-input-226-751f5c9c9878>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
```

```
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['status'], label = 'Skewness: %.2f'%
(data['status'].skew()))
```

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



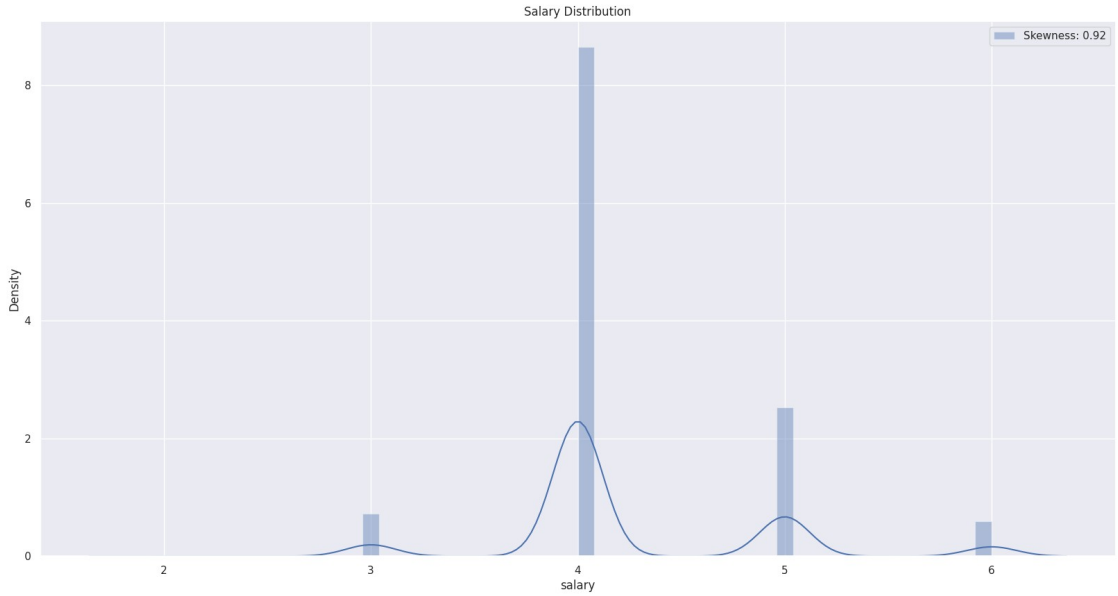

```
### Understanding the distribution of the column - Age
```

```
sns.distplot(data['salary'], label = 'Skewness: %.2f'%
(data['salary'].skew()))
plt.legend(loc = 'best')
plt.title('Salary Distribution')
```

```
<ipython-input-227-6b88c5e72c56>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
  sns.distplot(data['salary'], label = 'Skewness: %.2f'%
(data['salary'].skew()))
```

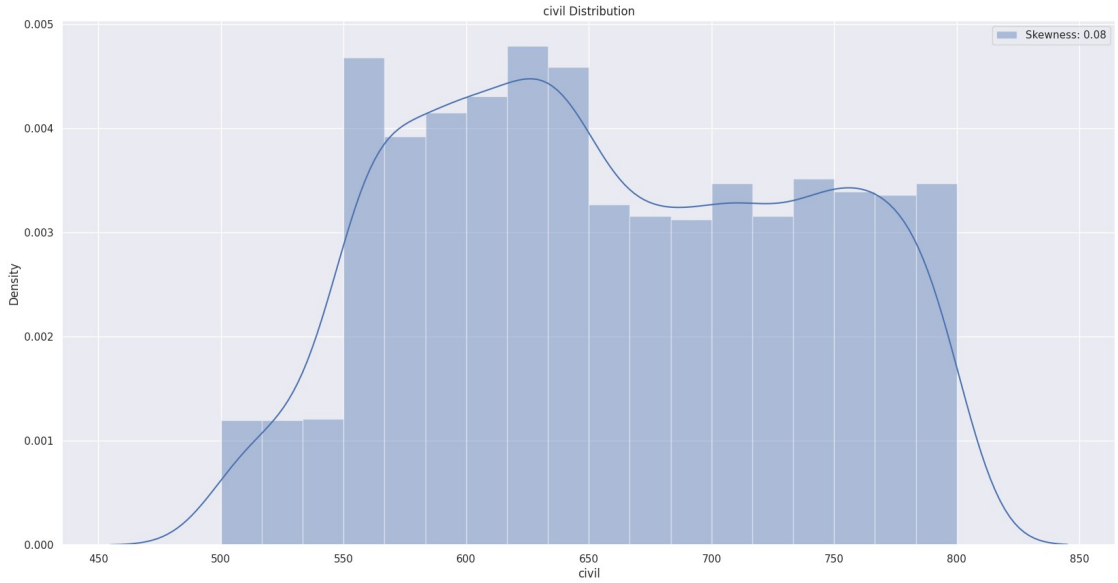
```
Text(0.5, 1.0, 'Salary Distribution')
```




```
sns.distplot(data['civil'], label = 'Skewness: %.2f'%
(data['civil'].skew()))
plt.legend(loc = 'best')
plt.title('civil Distribution')
```

```
<ipython-input-228-2a31e62a24dc>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['civil'], label = 'Skewness: %.2f'%
(data['civil'].skew()))
```

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



Monthly salary distribution by Credit Score

```
grid = sns.FacetGrid(data, col = 'credit')
grid.map(sns.distplot, 'salary')
```

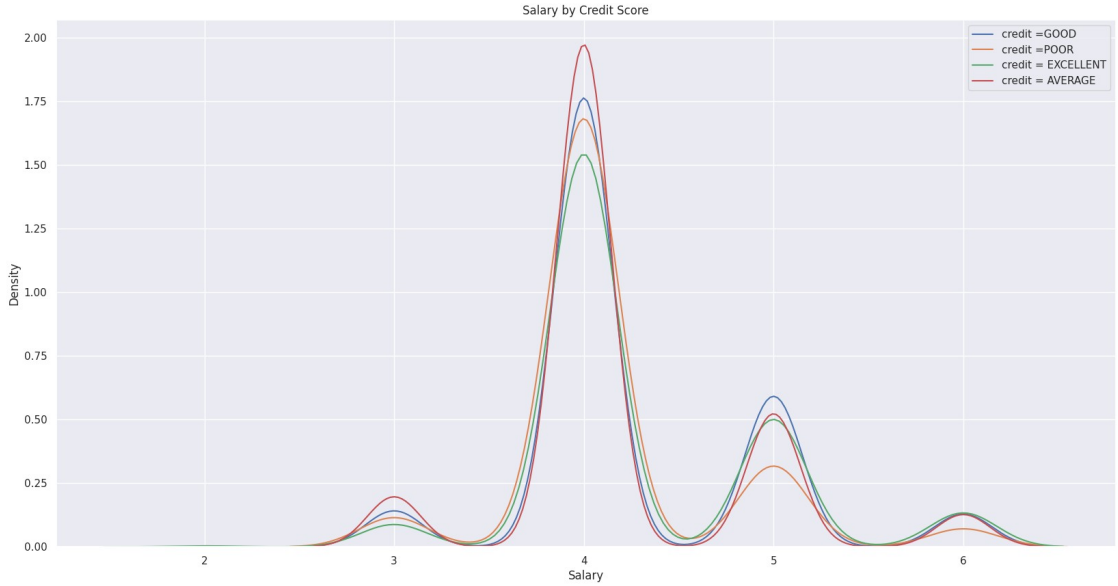
```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
```

```
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)

<seaborn.axisgrid.FacetGrid at 0x7f891607fb50>
```


```
### Merging the above graphs into one

sns.kdeplot(data['salary'][data['credit'] == 'GOOD'], label = 'credit
=GOOD')
sns.kdeplot(data['salary'][data['credit'] == 'POOR'], label = 'credit
```

```
=POOR')
sns.kdeplot(data['salary'][data['credit'] == 'EXCELLENT'], label=
'credit = EXCELLENT')
sns.kdeplot(data['salary'][data['credit'] == 'AVERAGE'], label=
'credit = AVERAGE')

plt.xlabel('Salary')
plt.legend()
plt.title('Salary by Credit Score')
```

```
Text(0.5, 1.0, 'Salary by Credit Score')
```

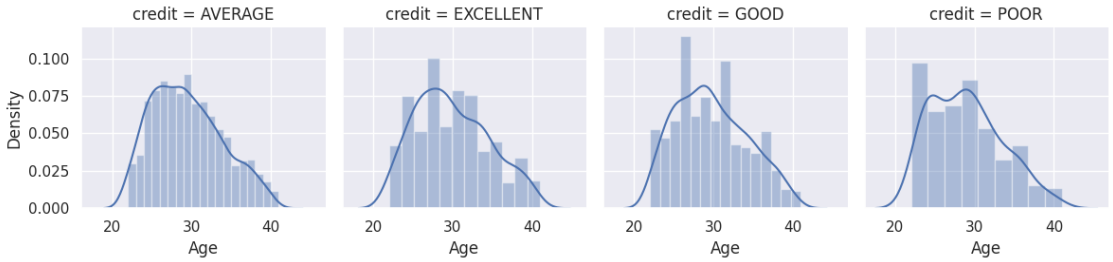


Outstanding Debt distribution by Credit Score

```
grid = sns.FacetGrid(data, col = 'credit')
grid.map(sns.distplot, 'Age')
```

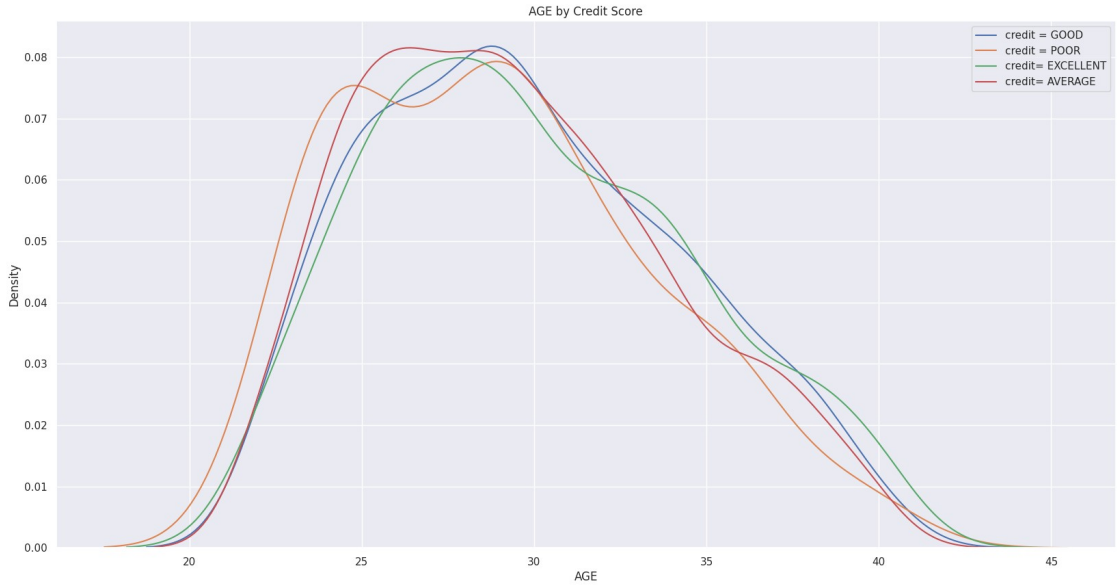
```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)

<seaborn.axisgrid.FacetGrid at 0x7f891576ee50>
```




```
### Merging the above graphs into o
sns.kdeplot(data['Age'][data['credit'] == 'GOOD'], label = 'credit =
GOOD')
sns.kdeplot(data['Age'][data['credit'] == 'POOR'], label = 'credit =
POOR')
sns.kdeplot(data['Age'][data['credit'] == 'EXCELLENT'], label =
'credit= EXCELLENT')
sns.kdeplot(data['Age'][data['credit'] == 'AVERAGE'], label = 'credit=
AVERAGE')

plt.xlabel("AGE")
plt.legend()
plt.title("AGE by Credit Score")
```

```
Text(0.5, 1.0, 'AGE by Credit Score')
```




```
from collections import Counter
def detect_outliers(df, n, features_list):
    outlier_indices = []
    for feature in features_list:
        Q1 = np.percentile(df[feature], 25)
        Q3 = np.percentile(df[feature], 75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR
```

```
        outlier_list_col = df[(df[feature] < Q1 - outlier_step) |
(df[feature] > Q3 + outlier_step)].index
        outlier_indices.extend(outlier_list_col)
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(key for key, value in
outlier_indices.items() if value > n)
    return multiple_outliers

numerical_columns = list(data.select_dtypes('number').columns)
print('Numerical columns: {}'.format(numerical_columns))
outliers_to_drop = detect_outliers(data, 2, numerical_columns)
print("We will drop these {} indices: ".format(len(outliers_to_drop)),
outliers_to_drop)
```

```
Numerical columns: ['status', 'Age', 'salary', 'civil', 'appcount',
'phonegrade', 'simstrength']
We will drop these 0 indices:  []
```

```
data.iloc[outliers_to_drop, :]
```

```
Empty DataFrame
Columns: [paid_status, status, Age, gender, education, married_status,
city, province, agerange, salary, civil, appcount, phonegrade,
simstrength, credit]
Index: []
```

```
### Drop outliers and reset index

print("Before: {} rows".format(len(data)))
data = data.drop(outliers_to_drop, axis = 0).reset_index(drop = True)
print("After: {} rows".format(len(data)))
```

```
Before: 3855 rows
After: 3855 rows
```

```
data
```

```
          paid_status  status  Age  gender      education
married_status  \
0                Paid       0   35    Male  Post Graduate
Married
1                Paid       0   32    Male  Self Employed
Unmarried
2                Paid       0   27    Male       Graduate
Unmarried
3                Paid       0   25    Male  Post Graduate
Unmarried
4                Paid       0   26    Male  Post Graduate
Unmarried
...               ...     ...  ...     ...            ...            .
..
```

```
3850             Paid       0   32    Male       Graduate
Married
3851  Pending Payment       1   31  Female  Self Employed
Married
3852             Paid       0   31    Male  Post Graduate
Unmarried
3853             Paid       0   25    Male  Post Graduate
Unmarried
3854             Paid       0   39    Male  Self Employed
Married

                          city        province agerange  salary  civil
\
0                       Daskroi         GUJARAT    25-35       6    569

1                        Guntur  ANDHRA PRADESH    25-35       4    734

2                        Rajkot         GUJARAT    25-35       5    641

3                      Vadodara         GUJARAT    25-35       5    800

4                        Bhokar     MAHARASHTRA    25-35       3    616

...                         ...             ...      ...     ...    ...

3850                     Dabhoi         GUJARAT    25-35       4    655

3851                     Mumbai     Maharashtra    25-35       5    505

3852  Chennai City Corporation      TAMIL NADU    25-35       4    777

3853                     Kalyan     MAHARASHTRA    25-35       4    739

3854                  Hyderabad       Telangana      >35       4    670


      appcount  phonegrade  simstrength     credit
0            7           1            3    AVERAGE
1            8           2            1       GOOD
2            6           1            1    AVERAGE
3            4           3            3  EXCELLENT
4            5           5            1    AVERAGE
...        ...         ...          ...        ...
3850         2           3            2       GOOD
3851         4           3            5       POOR
3852         7           2            1  EXCELLENT
3853         6           4            2       GOOD
3854         2           4            3       GOOD
```

```
[3855 rows x 15 columns]
```

```
data.columns
```

```
Index(['paid_status', 'status', 'Age', 'gender', 'education',
'married_status',
       'city', 'province', 'agerange', 'salary', 'civil', 'appcount',
       'phonegrade', 'simstrength', 'credit'],
      dtype='object')
```

```
### Dropping the columns from the dataset
```

```
data.drop(['paid_status', 'city',  'province', 'agerange',  'civil',],
axis = 1, inplace = True)
data
```

```
      status  Age  gender      education married_status  salary
appcount  \
0          0   35    Male  Post Graduate        Married       6
7
1          0   32    Male  Self Employed      Unmarried       4
8
2          0   27    Male       Graduate      Unmarried       5
6
3          0   25    Male  Post Graduate      Unmarried       5
4
4          0   26    Male  Post Graduate      Unmarried       3
5
...      ...  ...     ...            ...            ...     ...
...
3850       0   32    Male       Graduate        Married       4
2
3851       1   31  Female  Self Employed        Married       5
4
3852       0   31    Male  Post Graduate      Unmarried       4
7
3853       0   25    Male  Post Graduate      Unmarried       4
6
3854       0   39    Male  Self Employed        Married       4
2

      phonegrade  simstrength     credit
0              1            3    AVERAGE
1              2            1       GOOD
2              1            1    AVERAGE
3              3            3  EXCELLENT
4              5            1    AVERAGE
...          ...          ...        ...
3850           3            2       GOOD
3851           3            5       POOR
```

```
3852           2            1  EXCELLENT
3853           4            2       GOOD
3854           4            3       GOOD

[3855 rows x 10 columns]
```

```
### Looking at the missing values in the dataset
```

```
data.isnull().sum().sort_values(ascending = False)
```

```
education         22
status             0
Age                0
gender             0
married_status     0
salary             0
appcount           0
phonegrade         0
simstrength        0
credit             0
dtype: int64
```

```
### Finding the mean value of the column - Salary in the dataset using
Credit_Score
```

```
salary_good_mean = np.mean(data[data['credit'] == 'GOOD']['salary'])
salary_poor_mean = np.mean(data[data['credit'] == 'POOR']['salary'])
salary_excellent_mean = np.mean(data[data['credit'] == 'EXCELLENT']
['salary'])
salary_average_mean = np.mean(data[data['credit'] == 'AVERAGE']
['salary'])
```

```
(salary_good_mean, salary_poor_mean,
salary_average_mean,salary_excellent_mean)
```

```
(4.269565217391304, 4.157258064516129, 4.20543093270366,
4.296296296296297)
```

```
### Finding the indices of the rows where Salary is null
```

```
index_values = list(data['salary'].isnull())
index_values
```

```
[False,
 False,
 False,
 False,
 False,
 False,
 False,
```

```
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 ...]

### Replacing the missing values in the column Monthly_Inhand_Salary
using the decision logic

for index in range(len(data)):
    if index_values[index]:
```

```
        if data['credit'][index] == 'GOOD':
            data['salary'][index] = salary_good_mean
        elif data['credit'][index] == 'POOR':
            data['salary'][index] = salary_poor_mean
        elif data['credit'][index] == 'AVERAGE':
            data['salary'][index] = salary_average_mean
        else:

            data['salary'][index] = salary_excellent_mean
```

```
data['salary'].isnull().sum()
```

```
0
```

```
### Finding the median value of the column - salary in the dataset
```

```
salary_index = list(data['salary'].isnull())
median_sal = np.median(data['salary'].loc[salary_index])
median_sal
```

```
/usr/local/lib/python3.9/dist-packages/numpy/core/fromnumeric.py:3474:
RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/usr/local/lib/python3.9/dist-packages/numpy/core/_methods.py:189:
RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

```
nan
```

```
data.head()
```

```
   status  Age gender      education married_status  salary  appcount
\
0       0   35   Male  Post Graduate        Married       6         7

1       0   32   Male  Self Employed      Unmarried       4         8

2       0   27   Male       Graduate      Unmarried       5         6

3       0   25   Male  Post Graduate      Unmarried       5         4

4       0   26   Male  Post Graduate      Unmarried       3         5


   phonegrade  simstrength  Target
0           1            3       2
1           2            1       1
2           1            1       2
3           3            3       2
4           5            1       2
```

```
### One Hot Encoding the columns - Month, Occupation,
Payment_of_Min_Amount of the dataset

data = pd.get_dummies(data = data,
                                        columns = ['gender',
'education', 'married_status'])
data
```

```
      status  Age  salary  appcount  phonegrade  simstrength
credit  \
0          0   35       6         7           1            3
AVERAGE
1          0   32       4         8           2            1
GOOD
2          0   27       5         6           1            1
AVERAGE
3          0   25       5         4           3            3
EXCELLENT
4          0   26       3         5           5            1
AVERAGE
...      ...  ...     ...       ...         ...          ...        ..
.
3850       0   32       4         2           3            2
GOOD
3851       1   31       5         4           3            5
POOR
3852       0   31       4         7           2            1
EXCELLENT
3853       0   25       4         6           4            2
GOOD
3854       0   39       4         2           4            3
GOOD

      gender_Female  gender_Male  education_10th-12th
education_Graduate  \
0                 0            1                    0
0
1                 0            1                    0
0
2                 0            1                    0
1
3                 0            1                    0
0
4                 0            1                    0
0
...             ...          ...                  ...
...
3850              0            1                    0
1
3851              1            0                    0
```

```
0
3852              0            1                     0
0
3853              0            1                     0
0
3854              0            1                     0
0

      education_Post Graduate  education_Self Employed  \
0                           1                        0
1                           0                        1
2                           0                        0
3                           1                        0
4                           1                        0
...                       ...                      ...
3850                        0                        0
3851                        0                        1
3852                        1                        0
3853                        1                        0
3854                        0                        1

      married_status_Married  married_status_Unmarried
0                          1                        0
1                          0                        1
2                          0                        1
3                          0                        1
4                          0                        1
...                      ...                      ...
3850                       1                        0
3851                       1                        0
3852                       0                        1
3853                       0                        1
3854                       1                        0

[3855 rows x 15 columns]
```

Encoding the Credit Score (Target) column

```
credit_score_data = data['credit']
target = []

for each_credit_score in credit_score_data:
    if each_credit_score == 'GOOD':
        target.append(1)
    elif each_credit_score == 'AVERAGE':
        target.append(2)
    elif each_credit_score == 'EXCELLENT':
        target.append(2)
    else:
        target.append(0)
```

```
### Removing the Credit Score column

data.drop(['credit'], axis = 1, inplace = True)

### Adding the Target column

data['Target'] = target
```

Splitting the Training

```
### Splitting the data to the matrices X and Y using the training set.

X = data.iloc[:, : -1].values
Y = data.iloc[:, -1].values

X
```

```
array([[ 0, 35,  6, ...,  0,  1,  0],
       [ 0, 32,  4, ...,  1,  0,  1],
       [ 0, 27,  5, ...,  0,  0,  1],
       ...,
       [ 0, 31,  4, ...,  0,  0,  1],
       [ 0, 25,  4, ...,  0,  0,  1],
       [ 0, 39,  4, ...,  1,  1,  0]])
```

```
Y
```

```
array([2, 1, 2, ..., 2, 1, 1])
```

```
### Dividing the dataset into train and test in the ratio of 70 : 30
from sklearn.model_selection import train_test_split


X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
0.3, random_state = 27, shuffle = True)

X_train
```

```
array([[ 0, 28,  4, ...,  1,  0,  1],
       [ 0, 33,  4, ...,  0,  0,  1],
       [ 0, 29,  4, ...,  0,  1,  0],
       ...,
       [ 0, 31,  4, ...,  0,  0,  1],
       [ 0, 35,  4, ...,  0,  1,  0],
       [ 0, 24,  3, ...,  0,  0,  1]])
```

```
X_test
```

```
array([[ 0, 28,  4, ...,  0,  0,  1],
       [ 0, 29,  4, ...,  1,  1,  0],
       [ 0, 29,  4, ...,  0,  0,  1],
```

```
       ...,
       [ 0, 38,  4, ...,  0,  1,  0],
       [ 0, 25,  4, ...,  0,  0,  1],
       [ 0, 28,  4, ...,  0,  0,  1]])
```

```
Y_train
```

```
array([1, 2, 1, ..., 1, 2, 2])
```

```
Y_test
```

```
array([1, 2, 1, ..., 2, 1, 2])
```

Fit Model

```
from collections import OrderedDict
from sklearn.metrics import confusion_matrix, accuracy_score,
recall_score, precision_score, f1_score

### Dictionary to store model and its accuracy

model_accuracy = OrderedDict()
### Dictionary to store model and its precision

model_precision = OrderedDict()
### Dictionary to store model and its recall

model_recall = OrderedDict()
```

Logistic Regression

```
### Training the Logistic Regression model on the dataset
from sklearn.linear_model import LogisticRegression
logistic_classifier = LogisticRegression(random_state = 27)
logistic_classifier.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression(random_state=27)
```

```
### Predicting the Test set results

Y_pred = logistic_classifier.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))
```

```
[[2 1]
 [2 2]
 [2 1]
 ...
 [2 2]
 [2 1]
 [2 2]]
```

```
### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

logistic_accuracy = round(100 * accuracy_score(Y_test, Y_pred), 2)
model_accuracy['Logistic Regression'] = logistic_accuracy

logistic_precision = round(100 * precision_score(Y_test, Y_pred,
average = 'weighted'), 2)
model_precision['Logistic Regression'] = logistic_precision

logistic_recall = round(100 * recall_score(Y_test, Y_pred, average =
'weighted'), 2)
model_recall['Logistic Regression'] = logistic_recall

print('The accuracy of this model is {} %.'.format(logistic_accuracy))
print('The precision of this model is {}
%.'.format(logistic_precision))
print('The recall of this model is {} %.'.format(logistic_recall))
```

```
[[  0   0  80]
 [  0   4 368]
 [  0  14 691]]
The accuracy of this model is 60.07 %.
The precision of this model is 44.11 %.
The recall of this model is 60.07 %.

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

K-Nearest Neighbors (k = 5)

```
### Applying 5NN model
from sklearn.neighbors import KNeighborsClassifier

classifier_1nn = KNeighborsClassifier(n_neighbors = 5, algorithm =
'auto', p = 2, metric = 'minkowski')
classifier_1nn.fit(X_train, Y_train)
```

```
KNeighborsClassifier()
```

```
### Predicting the Test set results

Y_pred = classifier_1nn.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))
```

```
[[1 1]
 [2 2]
 [2 1]
 ...
 [1 2]
 [2 1]
 [2 2]]
```

```
### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

```
### Printing the accuracy, precision, and recall of the model

nn1_accuracy = round(100 * accuracy_score(Y_test, Y_pred), 2)
model_accuracy['1 - Nearest Neighbors'] = nn1_accuracy

nn1_precision = round(100 * precision_score(Y_test, Y_pred, average =
'weighted'), 2)
model_precision['1 - Nearest Neighbors'] = nn1_precision

nn1_recall = round(100 * recall_score(Y_test, Y_pred, average =
'weighted'), 2)
model_recall['1 - Nearest Neighbors'] = nn1_recall

print('The accuracy of this model is {} %.'.format(nn1_accuracy))
print('The precision of this model is {} %.'.format(nn1_precision))
print('The recall of this model is {} %.'.format(nn1_recall))
```

```
[[  8   3  69]
 [  0 110 262]
 [ 24 186 495]]
The accuracy of this model is 52.98 %.
```

```
The precision of this model is 50.07 %.
The recall of this model is 52.98 %.
```

```
### Applying 3NN model
```

```
classifier_3nn = KNeighborsClassifier(n_neighbors = 3, algorithm =
'auto', p = 2, metric = 'minkowski')
classifier_3nn.fit(X_train, Y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
### Predicting the Test set results
```

```
Y_pred = classifier_3nn.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))
```

```
### Making the confusion matrix
```

```
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

```
### Printing the accuracy, precision, and recall of the model
```

```
nn3_accuracy = round(100 * accuracy_score(Y_test, Y_pred), 2)
model_accuracy['3 - Nearest Neighbors'] = nn3_accuracy

nn3_precision = round(100 * precision_score(Y_test, Y_pred, average =
'weighted'), 2)
model_precision['3 - Nearest Neighbors'] = nn3_precision

nn3_recall = round(100 * recall_score(Y_test, Y_pred, average =
'weighted'), 2)
model_recall['3 - Nearest Neighbors'] = nn3_recall

print('The accuracy of this model is {} %.'.format(nn3_accuracy))
print('The precision of this model is {} %.'.format(nn3_precision))
print('The recall of this model is {} %.'.format(nn3_recall))
```

```
[[2 1]
 [2 2]
 [2 1]
 ...
 [1 2]
 [2 1]
 [2 2]]
[[ 14   2  64]
 [  6 127 239]
 [ 51 193 461]]
The accuracy of this model is 52.03 %.
```

```
The precision of this model is 50.81 %.
The recall of this model is 52.03 %.
```

```
### Applying 5NN model

classifier_5nn = KNeighborsClassifier(n_neighbors = 5, algorithm =
'auto', p = 2, metric = 'minkowski')
classifier_5nn.fit(X_train, Y_train)
KNeighborsClassifier()
### Predicting the Test set results

Y_pred = classifier_5nn.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))

### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

nn5_accuracy = round(100 * accuracy_score(Y_test, Y_pred), 2)
model_accuracy['5 - Nearest Neighbors'] = nn5_accuracy

nn5_precision = round(100 * precision_score(Y_test, Y_pred, average =
'weighted'), 2)
model_precision['5 - Nearest Neighbors'] = nn5_precision

nn5_recall = round(100 * recall_score(Y_test, Y_pred, average =
'weighted'), 2)
model_recall['5 - Nearest Neighbors'] = nn5_recall

print('The accuracy of this model is {} %.'.format(nn5_accuracy))
print('The precision of this model is {} %.'.format(nn5_precision))
print('The recall of this model is {} %.'.format(nn5_recall))
```

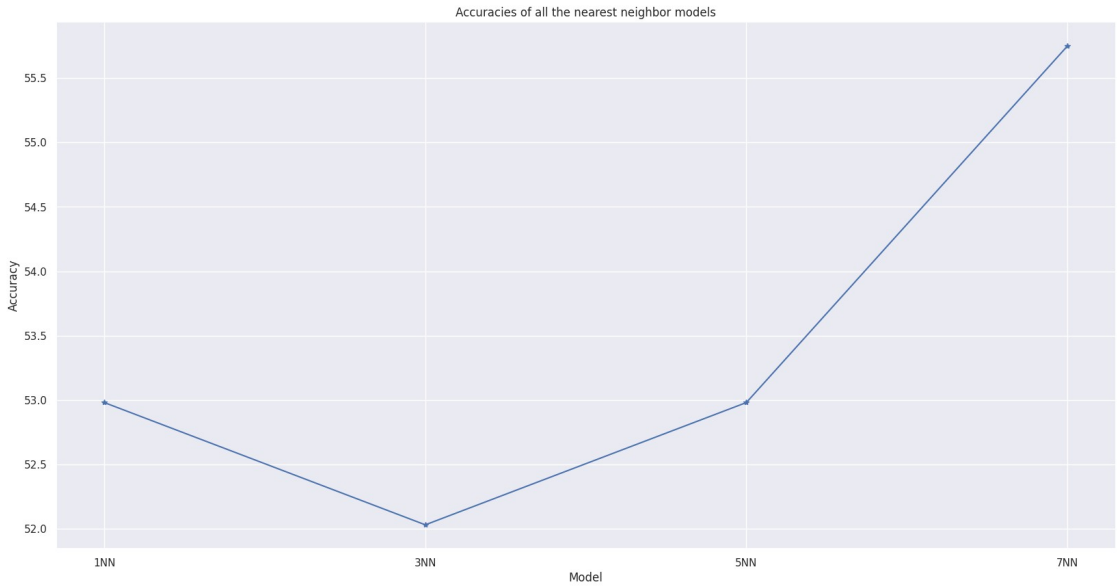
```
[[1 1]
 [2 2]
 [2 1]
 ...
 [1 2]
 [2 1]
 [2 2]]
[[  8   3  69]
 [  0 110 262]
 [ 24 186 495]]
The accuracy of this model is 52.98 %.
The precision of this model is 50.07 %.
The recall of this model is 52.98 %.
```

```
### Applying 7NN model

classifier_7nn = KNeighborsClassifier(n_neighbors = 7, algorithm =
'auto', p = 2, metric = 'minkowski')
classifier_7nn.fit(X_train, Y_train)
KNeighborsClassifier(n_neighbors=7)
### Predicting the Test set results

Y_pred = classifier_7nn.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))

### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

nn7_accuracy = round(100 * accuracy_score(Y_test, Y_pred), 2)
model_accuracy['7 - Nearest Neighbors'] = nn7_accuracy

nn7_precision = round(100 * precision_score(Y_test, Y_pred, average =
'weighted'), 2)
model_precision['7 - Nearest Neighbors'] = nn7_precision

nn7_recall = round(100 * recall_score(Y_test, Y_pred, average =
'weighted'), 2)
model_recall['7 - Nearest Neighbors'] = nn7_recall

print('The accuracy of this model is {} %.'.format(nn7_accuracy))
print('The precision of this model is {} %.'.format(nn7_precision))
print('The recall of this model is {} %.'.format(nn7_recall))
```

```
[[2 1]
 [2 2]
 [2 1]
 ...
 [1 2]
 [2 1]
 [2 2]]
[[  5   2  73]
 [  0 111 261]
 [ 11 165 529]]
The accuracy of this model is 55.75 %.
The precision of this model is 52.35 %.
The recall of this model is 55.75 %.
```

```
### Looking at the accuracy graph of all the nearest neighbors
```

```
labels = ['1NN', '3NN', '5NN', '7NN']
values = [nn1_accuracy, nn3_accuracy, nn5_accuracy, nn7_accuracy]

plt.title('Accuracies of all the nearest neighbor models')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.plot(labels, values, '*-')
plt.show()
```



Naive Bayes

```
### Applying Naive Bayes Classification model
from sklearn.naive_bayes import GaussianNB

naive_bayes_classifier = GaussianNB()
naive_bayes_classifier.fit(X_train, Y_train)

GaussianNB()

### Predicting the Test set results

Y_pred = naive_bayes_classifier.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))

[[1 1]
 [1 2]
 [1 1]
 ...
 [1 2]
 [1 1]
 [1 2]]
```

```
### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

naive_bayes_accuracy = round(100 * accuracy_score(Y_test, Y_pred), 2)
model_accuracy['Gaussian Naive Bayes'] = naive_bayes_accuracy

naive_bayes_precision = round(100 * precision_score(Y_test, Y_pred,
average = 'weighted'), 2)
model_precision['Gaussian Naive Bayes'] = naive_bayes_precision

naive_bayes_recall = round(100 * recall_score(Y_test, Y_pred, average
= 'weighted'), 2)
model_recall['Gaussian Naive Bayes'] = naive_bayes_recall

print('The accuracy of this model is {}
%.'.format(naive_bayes_accuracy))
print('The precision of this model is {}
%.'.format(naive_bayes_precision))
print('The recall of this model is {} %.'.format(naive_bayes_recall))
```

```
[[ 70   4   6]
 [  0 372   0]
 [113 579  13]]
The accuracy of this model is 39.33 %.
The precision of this model is 56.86 %.
The recall of this model is 39.33 %.
```

Decision Tree Classification

```
### Applying Decision Tree Classification model
from sklearn.tree import DecisionTreeClassifier

decision_tree_classifier = DecisionTreeClassifier(criterion =
'entropy', random_state = 27)
decision_tree_classifier.fit(X_train, Y_train)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=27)
```

```
### Predicting the Test set results

Y_pred = decision_tree_classifier.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))
```

```
[[2 1]
 [2 2]
 [1 1]
```

```
 ...
 [1 2]
 [2 1]
 [2 2]]
```

```
### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

decision_tree_accuracy = round(100 * accuracy_score(Y_test, Y_pred),
2)
model_accuracy['Decision Tree'] = decision_tree_accuracy

decision_tree_precision = round(100 * precision_score(Y_test, Y_pred,
average = 'weighted'), 2)
model_precision['Decision Tree'] = decision_tree_precision

decision_tree_recall = round(100 * recall_score(Y_test, Y_pred,
average = 'weighted'), 2)
model_recall['Decision Tree'] = decision_tree_recall

print('The accuracy of this model is {}
%.'.format(decision_tree_accuracy))
print('The precision of this model is {}
%.'.format(decision_tree_precision))
print('The recall of this model is {}
%.'.format(decision_tree_recall))
```

```
[[ 32   1  47]
 [  3 157 212]
 [ 44 233 428]]
The accuracy of this model is 53.33 %.
The precision of this model is 53.67 %.
The recall of this model is 53.33 %.
```

Random Forest Classification (10 trees)

```
### Applying Random Forest Classification model (10 trees)
from sklearn.ensemble import RandomForestClassifier
# from sklearn.ensemble import StackingClassifier
random_forest_10_classifier = RandomForestClassifier(n_estimators =
10, criterion = 'entropy', random_state = 27)
random_forest_10_classifier.fit(X_train, Y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10,
random_state=27)
```

```
### Predicting the Test set results

Y_pred = random_forest_10_classifier.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))
```

```
[[2 1]
 [1 2]
 [2 1]
 ...
 [2 2]
 [2 1]
 [2 2]]
```

```
### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

random_forest_10_accuracy = round(100 * accuracy_score(Y_test,
Y_pred), 2)
model_accuracy['Random Forest (10 trees)'] = random_forest_10_accuracy

random_forest_10_precision = round(100 * precision_score(Y_test,
Y_pred, average = 'weighted'), 2)
model_precision['Random Forest (10 trees)'] =
random_forest_10_precision

random_forest_10_recall = round(100 * recall_score(Y_test, Y_pred,
average = 'weighted'), 2)
model_recall['Random Forest (10 trees)'] = random_forest_10_recall

print('The accuracy of this model is {}
%.'.format(random_forest_10_accuracy))
print('The precision of this model is {}
%.'.format(random_forest_10_precision))
print('The recall of this model is {}
%.'.format(random_forest_10_recall))
```

```
[[ 24   0  56]
 [  0 126 246]
 [ 38 233 434]]
The accuracy of this model is 50.48 %.
The precision of this model is 49.89 %.
The recall of this model is 50.48 %.
```

```
### Applying Random Forest Classification model (25 trees)
```

```
random_forest_25_classifier = RandomForestClassifier(n_estimators =
25, criterion = 'entropy', random_state = 27)
random_forest_25_classifier.fit(X_train, Y_train)
RandomForestClassifier(criterion='entropy', n_estimators=25,
random_state=27)
### Predicting the Test set results

Y_pred = random_forest_25_classifier.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))

### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

random_forest_25_accuracy = round(100 * accuracy_score(Y_test,
Y_pred), 2)
model_accuracy['Random Forest (25 trees)'] = random_forest_25_accuracy

random_forest_25_precision = round(100 * precision_score(Y_test,
Y_pred, average = 'weighted'), 2)
model_precision['Random Forest (25 trees)'] =
random_forest_25_precision

random_forest_25_recall = round(100 * recall_score(Y_test, Y_pred,
average = 'weighted'), 2)
model_recall['Random Forest (25 trees)'] = random_forest_25_recall

print('The accuracy of this model is {}
%.'.format(random_forest_25_accuracy))
print('The precision of this model is {}
%.'.format(random_forest_25_precision))
print('The recall of this model is {}
%.'.format(random_forest_25_recall))

[[2 1]
 [2 2]
 [2 1]
 ...
 [2 2]
 [2 1]
 [2 2]]
[[ 19   0  61]
 [  1 113 258]
 [ 24 178 503]]
The accuracy of this model is 54.88 %.
```

```
The precision of this model is 52.76 %.
The recall of this model is 54.88 %.
```

```
### Applying Random Forest Classification model (50 trees)

random_forest_50_classifier = RandomForestClassifier(n_estimators =
50, criterion = 'entropy', random_state = 27)
random_forest_50_classifier.fit(X_train, Y_train)
RandomForestClassifier(criterion='entropy', n_estimators=50,
random_state=27)
### Predicting the Test set results

Y_pred = random_forest_50_classifier.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))

### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

random_forest_50_accuracy = round(100 * accuracy_score(Y_test,
Y_pred), 2)
model_accuracy['Random Forest (50 trees)'] = random_forest_50_accuracy

random_forest_50_precision = round(100 * precision_score(Y_test,
Y_pred, average = 'weighted'), 2)
model_precision['Random Forest (50 trees)'] =
random_forest_50_precision

random_forest_50_recall = round(100 * recall_score(Y_test, Y_pred,
average = 'weighted'), 2)
model_recall['Random Forest (50 trees)'] = random_forest_50_recall

print('The accuracy of this model is {}
%.'.format(random_forest_50_accuracy))
print('The precision of this model is {}
%.'.format(random_forest_50_precision))
print('The recall of this model is {}
%.'.format(random_forest_50_recall))
```

```
[[2 1]
 [1 2]
 [2 1]
 ...
 [2 2]
 [2 1]
 [2 2]]
```

```
[[ 23   0  57]
 [  1 101 270]
 [ 28 186 491]]
The accuracy of this model is 53.15 %.
The precision of this model is 50.95 %.
The recall of this model is 53.15 %.
```

```
### Preparing the Soft Voting Classifier
```

```
### Creating the list of estimators
```

```
estimators = list()
from sklearn.ensemble import VotingClassifier

estimators.append(('3nn', classifier_3nn))
estimators.append(('decision_tree', decision_tree_classifier))
estimators.append(('random_forest_50', random_forest_50_classifier))
```

```
### Applying Soft Voting Classification
```

```
soft_voting_classifier = VotingClassifier(estimators = estimators,
voting = 'soft')
soft_voting_classifier.fit(X_train, Y_train)
```

```
VotingClassifier(estimators=[('3nn',
KNeighborsClassifier(n_neighbors=3)),
                             ('decision_tree',

DecisionTreeClassifier(criterion='entropy',

random_state=27)),
                             ('random_forest_50',

RandomForestClassifier(criterion='entropy',
                                                    n_estimators=50,

random_state=27))],
                 voting='soft')
```

```
### Predicting the Test set results
```

```
Y_pred = soft_voting_classifier.predict(X_test)
print(np.concatenate((Y_pred.reshape(len(Y_pred), 1),
Y_test.reshape(len(Y_test), 1)), 1))
```

```
[[2 1]
 [2 2]
 [2 1]
 ...
 [1 2]
```

```
 [2 1]
 [2 2]]
```

```
### Making the confusion matrix

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

### Printing the accuracy, precision, and recall of the model

soft_voting_accuracy = round(100 * accuracy_score(Y_test, Y_pred), 2)
model_accuracy['Soft Voting'] = soft_voting_accuracy

soft_voting_precision = round(100 * precision_score(Y_test, Y_pred,
average = 'weighted'), 2)
model_precision['Soft Voting'] = soft_voting_precision

soft_voting_recall = round(100 * recall_score(Y_test, Y_pred, average
= 'weighted'), 2)
model_recall['Soft Voting'] = soft_voting_recall

print('The accuracy of this model is {}
%.'.format(soft_voting_accuracy))
print('The precision of this model is {}
%.'.format(soft_voting_precision))
print('The recall of this model is {} %.'.format(soft_voting_recall))
```

```
[[ 27   1  52]
 [  0 138 234]
 [ 32 217 456]]
The accuracy of this model is 53.67 %.
The precision of this model is 53.07 %.
The recall of this model is 53.67 %.
```

Model evaluation

```
### Looking at the model accuracy dictionary

model_accuracy
```

```
OrderedDict([('Logistic Regression', 60.07),
             ('1 - Nearest Neighbors', 52.98),
             ('3 - Nearest Neighbors', 52.03),
             ('5 - Nearest Neighbors', 52.98),
             ('7 - Nearest Neighbors', 55.75),
             ('Gaussian Naive Bayes', 39.33),
             ('Decision Tree', 53.33),
             ('Random Forest (10 trees)', 50.48),
             ('Random Forest (25 trees)', 54.88),
             ('Random Forest (50 trees)', 53.15),
             ('Soft Voting', 53.67)])
```

```
### Looking at the model precision dictionary

model_precision

OrderedDict([('Logistic Regression', 44.11),
             ('1 - Nearest Neighbors', 50.07),
             ('3 - Nearest Neighbors', 50.81),
             ('5 - Nearest Neighbors', 50.07),
             ('7 - Nearest Neighbors', 52.35),
             ('Gaussian Naive Bayes', 56.86),
             ('Decision Tree', 53.67),
             ('Random Forest (10 trees)', 49.89),
             ('Random Forest (25 trees)', 52.76),
             ('Random Forest (50 trees)', 50.95),
             ('Soft Voting', 53.07)])

### Looking at the model recall dictionary

model_recall

OrderedDict([('Logistic Regression', 60.07),
             ('1 - Nearest Neighbors', 52.98),
             ('3 - Nearest Neighbors', 52.03),
             ('5 - Nearest Neighbors', 52.98),
             ('7 - Nearest Neighbors', 55.75),
             ('Gaussian Naive Bayes', 39.33),
             ('Decision Tree', 53.33),
             ('Random Forest (10 trees)', 50.48),
             ('Random Forest (25 trees)', 54.88),
             ('Random Forest (50 trees)', 53.15),
             ('Soft Voting', 53.67)])

### Tabulating the results
from tabulate import tabulate

### Model Validation

from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

### Remove unnecessary warnings

import warnings
warnings.filterwarnings('ignore')

table = []
table.append(['S.No.', 'Classification Model', 'Model Accuracy',
'Model Precision', 'Model Recall'])
count = 1
```

```
for model in model_accuracy:
    row = [count, model, model_accuracy[model],
model_precision[model], model_recall[model]]
    table.append(row)
    count += 1

print(tabulate(table, headers = 'firstrow', tablefmt = 'fancy_grid'))
```

```
╒═══════╤═══════════════════════════╤═══════════════════╤═══════════
═══╤═══════════════════╕
│   S.No. │ Classification Model      │   Model Accuracy │   Model
Precision │   Model Recall │
╞═══════╪═══════════════════════════╪═══════════════════╪═══════════
═══╪═══════════════════╡
│       1 │ Logistic Regression       │             60.07 │
44.11 │           60.07 │
├───────┼───────────────────────────┼───────────────────┼───────────
───┼───────────────────┤
│       2 │ 1 - Nearest Neighbors     │             52.98 │
50.07 │           52.98 │
├───────┼───────────────────────────┼───────────────────┼───────────
───┼───────────────────┤
│       3 │ 3 - Nearest Neighbors     │             52.03 │
50.81 │           52.03 │
├───────┼───────────────────────────┼───────────────────┼───────────
───┼───────────────────┤
│       4 │ 5 - Nearest Neighbors     │             52.98 │
50.07 │           52.98 │
├───────┼───────────────────────────┼───────────────────┼───────────
───┼───────────────────┤
│       5 │ 7 - Nearest Neighbors     │             55.75 │
52.35 │           55.75 │
├───────┼───────────────────────────┼───────────────────┼───────────
───┼───────────────────┤
│       6 │ Gaussian Naive Bayes      │             39.33 │
56.86 │           39.33 │
├───────┼───────────────────────────┼───────────────────┼───────────
───┼───────────────────┤
│       7 │ Decision Tree             │             53.33 │
53.67 │           53.33 │
├───────┼───────────────────────────┼───────────────────┼───────────
───┼───────────────────┤
│       8 │ Random Forest (10 trees)  │             50.48 │
49.89 │           50.48 │
├───────┼───────────────────────────┼───────────────────┼───────────
───┼───────────────────┤
│       9 │ Random Forest (25 trees)  │             54.88 │
52.76 │           54.88 │
├───────┼───────────────────────────┼───────────────────┼───────────
```

10	Random Forest (50 trees)	53.15	50.95	53.15
11	Soft Voting	53.67	53.07	53.67

CREDIT SCORE CALCULATION USING MACHINE LEARNING APPROACH AND RISK CALCULATION USING MORAN'S I

Abstract:

Can money buy happiness? To examine this question, research in economics, psychology, and sociology has focused almost exclusively on examining the associations between income, spending or wealth and subjective well-being. Moving beyond this research, we provide the first empirical evidence that credit scores uniquely predict happiness. Across two samples, from the United Kingdom (N=615) and the United States (N=768), credit scores predicted life satisfaction even after controlling for a range of financial covariates, including income, spending, savings, debt, and home-ownership. Respondents with higher credit scores felt more optimistic about their future, promoting happiness. Further, the relationship between credit scores and wellbeing was moderated by participants' prior awareness of their score. Together, these results suggest that creditworthiness can plausibly increase well-being, either directly or indirectly, meaning that interventions to improve creditworthiness could improve consumer welfare.

Modern economies run on credit. From securing a mortgage for a first home, to simply opening a cellphone contract, most people are reliant on access to credit. Creditworthiness is defined by a person's credit score; a computationally derived metric of their financial decisions over time. While credit scores have traditionally enabled consumers to take on debt, their use has expanded into diverse settings, including insurance eligibility (Brockett & Golden, 2007), and even recruitment (Bernerth, Taylor, Walker, & Whitman, 2012). And yet, despite the growing role of credit in modern life, it remains unknown whether having 'good credit' is associated with greater subjective well-being. Understanding the relationship between credit scores and well-being contributes to the ongoing debate over whether money (or the lack of it) influences happiness. Research to date has focused almost exclusively on the role of income and wealth, demonstrating that those with higher incomes and greater wealth are, on average, more satisfied with their lives (Headey, Muffels, & Wooden, 2008; Stevenson & Wolfers, 2013). However, income and wealth are not the only, nor necessarily the best, indicators of an individual's financial circumstances: how a person uses their money may be just as strong a determinant of happiness as the total amount of money they have. For example, credit scores capture the extent to which long-term interests (e.g., budgeting to pay bills on time) are valued over short-term desires (e.g., impulse purchases leading to missed credit card payments). And because credit scores are not directly calculated based on a person's income, savings or spending1, this suggests credit scores may also indirectly measure important psychological characteristics, such as conscientiousness, which is known to be correlated with both credit scores (Bernerth, Taylor, Walker, & Whitman, 2012) and well-being (Hayes & Joseph, 2003).

In this research, we investigate whether credit scores are significantly correlated with well-being, and compare the strength of this association to a range of financial characteristics, including income, spending and savings. We also consider the psychological mechanisms through which credit scores could shape well-being. For example, credit allows people to achieve major milestones in life, as well as providing a buffer against financial shocks and uncertainty. On the basis that optimism, resilience and control are all known to predict wellbeing (Gallo & Matthews, 2003; Scheier & Carver, 1992), we test for three plausible explanations for why higher credit scores might shape well-being: (i) higher credit scores increase optimism about one's financial future, (ii) higher credit scores increase a sense of resilience, allowing one to capably handle economic shocks and reducing worry and anxiety, and (iii) higher credit scores increase perceived control. Finally, we also test if awareness of one's creditworthiness moderates the relationship between credit-score and wellbeing.

Introduction:

What Is a Loan?

The term loan refers to a type of credit vehicle in which a sum of money is lent to another party in exchange for future repayment of the value or principal amount. In many cases, the lender also adds interest or finance charges to the principal value which the borrower must repay in addition to the principal balance.

Loans may be for a specific, one-time amount, or they may be available as an open-ended line of credit up to a specified limit. Loans come in many different forms including secured, unsecured, commercial, and personal loans.

Understanding Loans

A loan is a form of debt incurred by an individual or other entity. The lender—usually a corporation, financial institution, or government—advances a sum of money to the borrower. In return, the borrower agrees to a certain set of terms including any finance charges, interest, repayment date, and other conditions.

In some cases, the lender may require collateral to secure the loan and ensure repayment. Loans may also take the form of bonds and certificates of deposit (CDs). It is also possible to take a loan from a 401(k) account.

The Loan Process

Here's how the loan process works. When someone needs money, they apply for a loan from a bank, corporation, government, or other entity. The borrower may be required to provide specific details such as the reason for the loan, their financial history, Social Security Number (SSN), and other information. The lender reviews the information including a person's debt-to-income (DTI) ratio to see if the loan can be paid back.

Based on the applicant's creditworthiness, the lender either denies or approves the application. The lender must provide a reason should the loan application be denied. If the application is approved, both parties sign a contract that outlines the details of the agreement. The lender advances the proceeds of the loan, after which the borrower must repay the amount including any additional charges such as interest.

The terms of a loan are agreed to by each party before any money or property changes hands or is disbursed. If the lender requires collateral, the lender outlines this in the loan documents. Most loans also have provisions regarding the maximum amount of interest, as well as other covenants such as the length of time before repayment is required.

Why Are Loans Used?

Loans are advanced for a number of reasons including major purchases, investing, renovations, debt consolidation, and business ventures. Loans also help existing companies expand their operations. Loans allow for growth in the overall money supply in an economy and open up competition by lending to new businesses.

The interest and fees from loans are a primary source of revenue for many banks, as well as some retailers through the use of credit facilities and credit cards.

If you're looking to take out a loan to pay for personal expenses, then a personal loan calculator can help you find the interest rate that best suits your needs.

Types of Loans

Loans come in many different forms. There are a number of factors that can differentiate the costs associated with them along with their contractual terms.

Secured vs. Unsecured Loan
Loans can be secured or unsecured. Mortgages and car loans are secured loans, as they are both backed or secured by collateral. In these cases, the collateral is the asset for which the loan is taken out, so the collateral for a mortgage is the home, while the vehicle secures a car loan. Borrowers may be required to put up other forms of collateral for other types of secured loans if required.

Credit cards and signature loans are unsecured loans. This means they are not backed by any collateral. Unsecured loans usually have higher interest rates than secured loans because the risk of default is higher than secured loans. That's because the lender of a secured loan can repossess the collateral if the borrower defaults. Rates tend to vary wildly on unsecured loans depending on multiple factors including the borrower's credit history.

Revolving vs. Term Loan
Loans can also be described as revolving or term. A revolving loan can be spent, repaid, and spent again, while a term loan refers to a loan paid off in equal monthly installments over a set period. A credit card is an unsecured, revolving loan, while a home equity line of credit (HELOC) is a secured, revolving loan. In contrast, a car loan is a secured, term loan, and a signature loan is an unsecured, term loan.

What Is a Loan Shark?
A loan shark is a slang term for predatory lenders who give informal loans at extremely high interest rates, often to people with little credit or collateral. Because these loan terms may not be legally enforceable, loan sharks have sometimes resorted to intimidation or violence in order to ensure repayment.

The Bottom Line
Loans are one of the basic building blocks of the financial economy. By giving out money with interest, lenders are able to provide funding for economic activity while being compensated for their risk. From small personal loans to billion-dollar corporate debts, lending money is an essential function of the modern economy.

In our program we have used two datasets ***1)Loans-3.csv*** ,here is the sample data of the dataset.
But from this dataframe we didn't use all columns for our work.we have considered the columns named **" status"** and **"province"** for our working where the earlier column gives the idea about whether a person has returned the loan amount or not. Values representing

	paid_status	status	Age	gender	education	married_status	city	province	agerange	salary	civil	appcount	phonegrade	simstrength
0	Paid	0	35	Male	Post Graduate	Married	Daskroi	GUJARAT	25-35	6	569	7	1	3
1	Paid	0	32	Male	Self Employed	Unmarried	Guntur	ANDHRA PRADESH	25-35	4	734	8	2	1
2	Paid	0	27	Male	Graduate	Unmarried	Rajkot	GUJARAT	25-35	5	641	6	1	1
3	Paid	0	25	Male	Post Graduate	Unmarried	Vadodara	GUJARAT	25-35	5	800	4	3	3
4	Paid	0	26	Male	Post Graduate	Unmarried	Bhokar	MAHARASHTRA	25-35	3	616	5	5	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3850	Paid	0	32	Male	Graduate	Married	Dabhoi	GUJARAT	25-35	4	655	2	3	2
3851	Pending Payment	1	31	Female	Self Employed	Married	Mumbai	Maharashtra	25-35	5	505	4	3	5
3852	Paid	0	31	Male	Post Graduate	Unmarried	Chennai City Corporation	TAMIL NADU	25-35	4	777	7	2	1
3853	Paid	0	25	Male	Post Graduate	Unmarried	Kalyan	MAHARASHTRA	25-35	4	739	6	4	2
3854	Paid	0	39	Male	Self Employed	Married	Hyderabad	Telangana						

History

In [13], it is noted that in 1941, David Durand was the first to recognize that one could use the same techniques that Fisher used in 1936 when he sought to solve the problem of differentiating between groups of a population. Durand carried out a research project for the U.S. National Bureau of Economic Research though it was not for predictive purposes [13]. In the 1950s Bill Fair and Earl Isaac established a consultancy in San Francisco, which used statistically derived models in lending decisions. Credit scoring's usefulness became more apparent in the 1960s when credit cards were introduced. Following the success in credit cards, banks and other lending institutions in the 1980s started using scoring in other products. It is important also to note that the objective of credit scoring is to predict risk not to explain why some customers default. Like any other model, credit scoring should be founded on sound methodology and the data used in modelling should be empirically derived. Statistical methods have been used since the 1950s and 1960s and are still popularly used today because they enable lenders to use concepts of sample estimators, confidence intervals and statistical inference, that is, hypothesis testing in credit scoring. This allows scorecard developers to evaluate discriminatory power of models and also to determine which borrower characteristics are more important in explaining borrower behaviour. Linear discriminant analysis was one of the earliest approaches to be used in credit scoring. Even though the scorecards it produced were very robust, the assumptions which were needed to ensure satisfactory discriminatory power were restrictive. According to [13], logistic regression has taken over from linear discriminant analysis and has been successful as the statistical method for credit scoring. According to [8], the methods used for credit scoring have increased in sophistication in recent years. They have evolved from traditional statistical techniques to innovative methods such as artificial intelligence, including machine 8 learning algorithms such as random forests, gradient boosting, and deep neural networks. With the ongoing discussions in the banking industry, the future of machine learning (ML) models will soon be more prevalent [4]. According to a recent discussion paper [3] by the European Banking Authority, Machine Learning models applied in Basel II regulation's Internal Rating Based (IRB) approach can be beneficial in the following ways; • Improving risk differentiation, both by improving the model discriminatory power and by providing useful tools for the identification of all relevant risk drivers or even relations among them. ML models might be used to optimise the portfolio segmentation, to build robust models across geographical and industry sectors/products and take data-driven decisions that balance data availability against the required model granularity. Moreover, ML models might help to confirm data features selected by expert judgement used for 'traditional' model development giving a data-driven perspective to the feature selection process • Improving data collection and preparation processes including, for example, cleaning of input data or by providing a tool for data treatment and data quality checks. ML models might also be used for performing outlier detection and for error correction. • Providing robust systems for validation and monitoring of the models. ML models might be used to generate model challengers or as a supporting analysis for alternative assumptions or approaches • Performing stress testing, by assessing the effect of certain specific conditions on the total capital requirements for credit risk and by identifying adverse scenarios

Global Moran's I

The Spatial Autocorrelation (Global Moran's I) tool measures spatial autocorrelation based on both feature locations and feature values simultaneously. Given a set of features and an associated attribute, it

evaluates whether the pattern expressed is clustered, dispersed, or random. The tool calculates the Moran's I Index value and both a a z-score and p-value to evaluate the significance of that Index. P-values are numerical approximations of the area under the curve for a known distribution, limited by the test statistic.

The Moran's I statistic for spatial autocorrelation is given as:

$$I = \frac{n}{S_0} \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} z_i z_j}{\sum_{i=1}^{n} z_i^2} \quad (1)$$

where z_i is the deviation of an attribute for feature i from its mean $(x_i - \bar{X})$, $w_{i,j}$ is the spatial weight between feature i and j, n is equal to the total number of features, and S_0 is the aggregate of all the spatial weights:

$$S_0 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} \quad (2)$$

The z_I-score for the statistic is computed as:

$$z_I = \frac{I - \mathrm{E}[I]}{\sqrt{\mathrm{V}[I]}} \quad (3)$$

where:

$$\mathrm{E}[I] = -1/(n-1) \quad (4)$$

$$\mathrm{V}[I] = \mathrm{E}[I^2] - \mathrm{E}[I]^2 \quad (5)$$

This tool measures spatial autocorrelation using feature locations and feature values simultaneously. The spatial autocorrelation tool utilizes multidimensional and multi-directional factors. The Moran's I index will be a value between -1 and 1. Positive spatial autocorrelation will show values that are clustered. Negative autocorrelation is dispersed. Random is close to zero. The tool generates a Z-score and p-value which helps evaluate the significance of the Moran's index.

The output of the Moran's I tool can be found in the results section of ArcGIS. Upon opening the HTML report for the Moran's I results you will see a graph showing how the tool calculated the data and whether or not the data is dispersed, random, or clustered. This report will also include the Moran's Index value, z-score, p-value. It will also provide a scale for the significance of the p-value and critical value for the z-score.

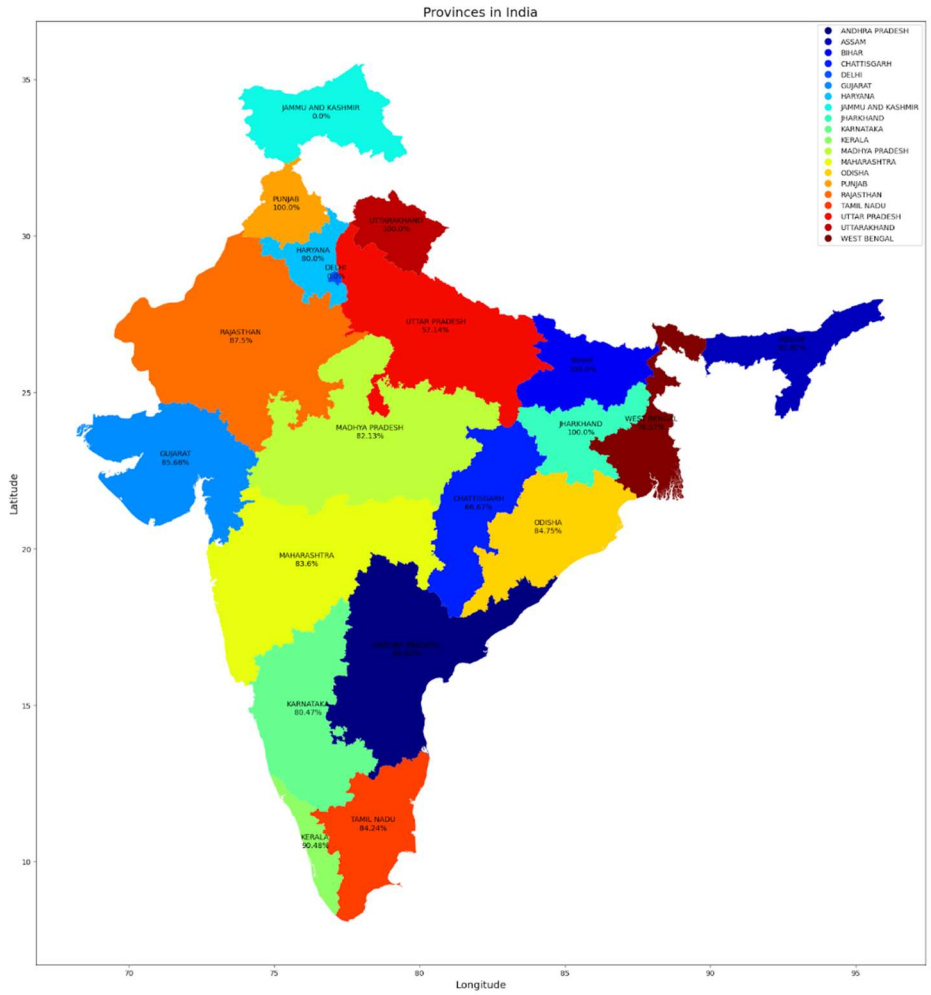
As decided in the hotspot analysis there is clustering of the data. The spatial autocorrelation tool indicates that clustering is occurring with regard to the percentage of returning bank loans in a state at their respective locations with regard to the concerned state.

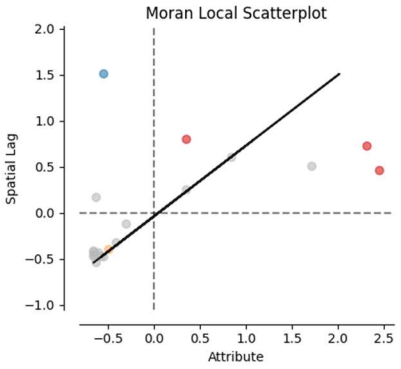
Review

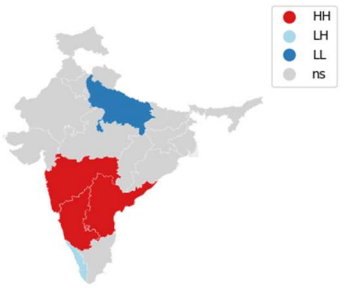
So we have 19 states of India from where people took loans . We tried to understand the percentage of returning the loan amount. That came out to be

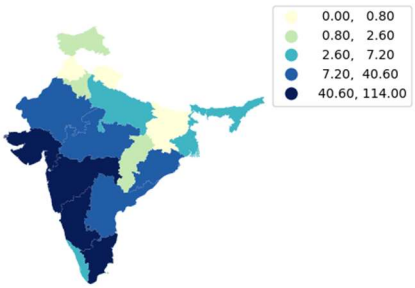
	province	0 state	1 state	percentage	geometry
0	GUJARAT	329	55	85.677083	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
1	ANDHRA PRADESH	289	37	88.650307	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
2	MAHARASHTRA	581	114	83.597122	MULTIPOLYGON (((73.45597 15.88986, 73.45597 15...
3	TAMIL NADU	465	87	84.239130	MULTIPOLYGON (((77.55596 8.07903, 77.55596 8.0...
4	ODISHA	50	9	84.745763	MULTIPOLYGON (((84.76986 19.10597, 84.76986 19...
5	KARNATAKA	449	109	80.465950	MULTIPOLYGON (((74.67097 13.19986, 74.67097 13...
6	ASSAM	18	4	81.818182	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
7	MADHYA PRADESH	170	37	82.125604	POLYGON ((78.36465 26.86884, 78.36688 26.86259...
8	RAJASTHAN	91	13	87.500000	POLYGON ((73.88944 29.97761, 73.89118 29.97007...
9	HARYANA	8	2	80.000000	POLYGON ((76.83715 30.87887, 76.85243 30.87069...
10	KERALA	38	4	90.476190	MULTIPOLYGON (((76.46736 9.54097, 76.46736 9.5...
11	UTTARAKHAND	4	0	100.000000	POLYGON ((79.19478 31.35362, 79.19817 31.35196...
12	UTTAR PRADESH	8	6	57.142857	POLYGON ((77.58468 30.40878, 77.58639 30.40801...
13	PUNJAB	4	0	100.000000	POLYGON ((75.86877 32.48868, 75.88712 32.47203...
14	WEST BENGAL	11	3	78.571429	MULTIPOLYGON (((88.01861 21.57278, 88.01889 21...
15	CHATTISGARH	2	1	66.666667	POLYGON ((83.32760 24.09965, 83.34575 24.09707...
16	BIHAR	1	0	100.000000	MULTIPOLYGON (((88.10548 26.53904, 88.10505 26...
17	JHARKHAND	1	0	100.000000	POLYGON ((87.59989 25.31466, 87.60688 25.31138...
18	DELHI	0	1	0.000000	POLYGON ((77.21732 28.83528, 77.21365 28.81350...
19	JAMMU AND KASHMIR	0	1	0.000000	POLYGON ((77.89957 35.42789, 77.90297 35.42759...

And then we showed that percentage of corresponding states on the map of India .
Out of those states with the help of spatial autocorrelation(Morans' I) we marked the hotspot states where the percentage of returning the loan is high;in our word the hotspot states which could be less damage prone to the Banks after circulating loans.









METHODOLOGY

1.Data collection:

Data collection is a systematic process of gathering observations or measurements. Whether you are performing research for business, governmental or academic purposes, data collection allows you to gain first-hand knowledge and original insights into your research problem.

In our project our main goal is to find out the risk zone from all province one the basis of our dataset.At first we collected the data from online website Kaggle.Then we verify this dataset from our institution teacher.The name of our dataset is “loans-3.csv” with total 14 column and 3855 rows.Here is the screenshot of our dataset.

	ID_0	ISO	NAME_0	ID_1	NAME_1	NL_NAME_1	VARNAME_1	TYPE_1	ENGTYPE_1	geometry
0	105	IND	India	1	Andaman and Nicobar	NaN	Andaman & Nicobar Islands|Andaman et Nicobar|I...	Union Territor	Union Territory	MULTIPOLYGON (((93.78773 6.85264, 93.78849 6.8...
1	105	IND	India	2	Andhra Pradesh	NaN	NaN	State	State	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
2	105	IND	India	3	Arunachal Pradesh	NaN	Agence de la Frontire du Nord-Est|French-obsol...	State	State	POLYGON ((96.15778 29.38310, 96.16380 29.37668...
3	105	IND	India	4	Assam	NaN	NaN	State	State	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
4	105	IND	India	5	Bihar	NaN	NaN	State	State	MULTIPOLYGON (((88.10548 26.53904, 88.10505 26...
5	105	IND	India	6	Chandigarh	NaN	NaN	Union Territor	Union Territory	POLYGON ((76.81981 30.68583, 76.81051 30.68495...
6	105	IND	India	7	Chhattisgarh	NaN	NaN	State	State	POLYGON ((83.32760 24.09965, 83.34575 24.09707...
7	105	IND	India	8	Dadra and Nagar Haveli	NaN	DAdra et Nagar Haveli|Dadra e Nagar Haveli	Union Territor	Union Territory	POLYGON ((72.99046 20.29209, 73.00357 20.29314...
8	105	IND	India	9	Daman and Diu	NaN	NaN	Union Territor	Union Territory	MULTIPOLYGON (((72.86014 20.47096, 72.86340 20...
9	105	IND	India	10	Delhi	NaN	NaN	Union Territor	Union Territory	POLYGON ((77.21732 28.83528, 77.21365 28.81350...
10	105	IND	India	11	Goa	NaN	Ga	State	State	MULTIPOLYGON (((73.78181 15.35569, 73.78181 15...
11	105	IND	India	12	Gujarat	NaN	Goudjerate|Gujerat|Gujerate	State	State	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
12	105	IND	India	13	Haryana	NaN	NaN	State	State	POLYGON ((76.83715 30.87887, 76.85243 30.87069...
13	105	IND	India	14	Himachal Pradesh	NaN	NaN	Union	Union	POLYGON ((76.80276 33.23666, 76.80630

	paid_status	status	Age	gender	education	married_status	city	province	agerange	salary	civil	appcount	phonegrade	simstrength
0	Paid	0	35	Male	Post Graduate	Married	Daskroi	GUJARAT	25-35	6	569	7	1	3
1	Paid	0	32	Male	Self Employed	Unmarried	Guntur	ANDHRA PRADESH	25-35	4	734	8	2	1
2	Paid	0	27	Male	Graduate	Unmarried	Rajkot	GUJARAT	25-35	5	641	6	1	1
3	Paid	0	25	Male	Post Graduate	Unmarried	Vadodara	GUJARAT	25-35	5	800	4	3	3
4	Paid	0	26	Male	Post Graduate	Unmarried	Bhokar	MAHARASHTRA	25-35	3	616	5	5	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3850	Paid	0	32	Male	Graduate	Married	Dabhoi	GUJARAT	25-35	4	655	2	3	2
3851	Pending Payment	1	31	Female	Self Employed	Married	Mumbai	Maharashtra	25-35	5	505	4	3	5
3852	Paid	0	31	Male	Post Graduate	Unmarried	Chennai City Corporation	TAMIL NADU	25-35	4	777	7	2	1
3853	Paid	0	25	Male	Post Graduate	Unmarried	Kalyan	MAHARASHTRA	25-35	4	739	6	4	2
3854	Paid	0	39	Male	Self Employed	Married	Hyderabad	Telangana	>35	4	670	2	4	3

3855 rows × 14 columns

From the above dataset we used “status” and “provine”.
We took another dataset for our project purpose named “india_state_geo.json.json” with 10 columns.Here is the screenshot of our dataset.

We used the geometry column for project purposes.

2.Preprocessing:

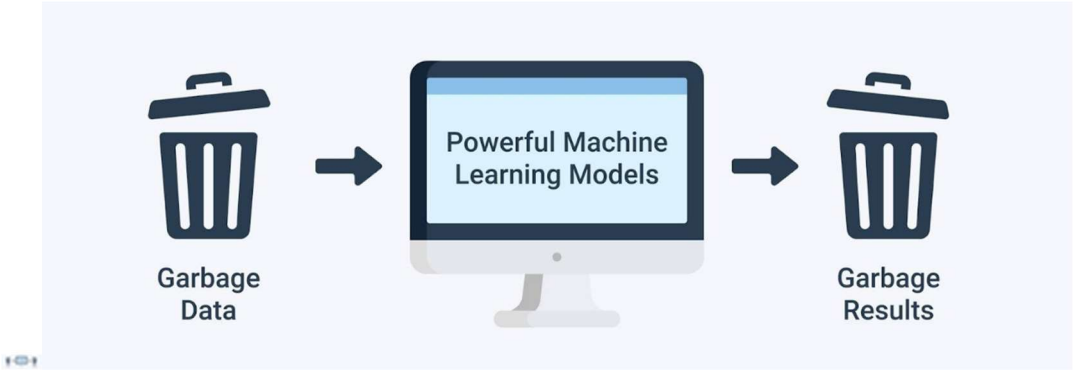
What Is Data Preprocessing?

Data preprocessing is a step in the data mining and data analysis process that takes raw data and transforms it into a format that can be understood and analyzed by computers and machine learning.
Raw, real-world data in the form of text, images, video, etc., is messy. Not only may it contain errors and inconsistencies, but it is often incomplete, and doesn’t have a regular, uniform design.
Machines like to process nice and tidy information – they read data as 1s and 0s. So calculating structured data, like whole numbers and percentages is easy. However, unstructured data, in the form of text and images must first be cleaned and formatted before analysis.

Data Preprocessing Importance

When using data sets to train machine learning models, you’ll often hear the phrase **“garbage in, garbage out”** This means that if you use bad or “dirty” data to train your model, you’ll end up with a bad, improperly trained model that won’t actually be relevant to your analysis.

Good, preprocessed data is even more important than the most powerful algorithms, to the point that machine learning models trained with bad data could actually be harmful to the analysis you’re trying to do – giving you “garbage” results.



Depending on your data gathering techniques and sources, you may end up with data that's out of range or includes an incorrect feature, like household income below zero or an image from a set of "zoo animals" that is actually a tree. Your set could have missing values or fields. Or text data, for example, will often have misspelled words and irrelevant symbols, URLs, etc.

When you properly preprocess and clean your data, you'll set yourself up for much more accurate downstream processes. We often hear about the importance of "data-driven decision making," but if these decisions are driven by bad data, they're simply bad decisions.

In our dataset at first we read the dataset("loans-3.csv" and "india_state_geo.json.json" and then find out the unique values from the "province " and "city" and "status" after we `iterate over each province and count the occurrences of 0 and 1` and then find out 0 means total payment and 1 means total pending payment for each province attributes.Then we calculate the percentage of 0 state on every province column attributes.From the india_state_geo_json dataset we extract the "geometry"and "city".If we merge this two dataset with some rules that if city and province state are matched then those row will create a new row with matching attributes.After

matched the dataset we save it in a new dataset.In the new dataset some values are not matched because of upper string and lower string letter case,then used merge function how=”outer”.If there some attributes is null in geometry columns then we filled this position with zeros.We plot choropleth map in india map with 0 state and corresponding their latitude and longitude.At last we calculate Moran's I test for global autocorrelation for a continuous attribute.Then we plot the spatial lag 1 state and 1 state in moran scatterplot.Then we distinguish the specific type of auto spatial correlation in High-High,High-Low,Low-High,Low-Low,after we visualize in indian map.

Approaches

We used a few approaches in our project ,mainly we focused on spatial autocorrelation global Moran's i.

Global Moran's I

The Spatial Autocorrelation (Global Moran's I) tool measures spatial autocorrelation based on both feature locations and feature values simultaneously. Given a set of features and an associated attribute, it evaluates whether the pattern expressed is clustered, dispersed, or random. The tool calculates the Moran's I Index value and both a a z-score and p-value to evaluate the significance of that Index. P-values are numerical approximations of the area under the curve for a known distribution, limited by the test statistic.

This tool measures spatial autocorrelation using feature locations and feature values simultaneously. The spatial autocorrelation tool utilizes multidimensional and multi-directional factors. The Moran's I index will be a value between -1 and 1. Positive spatial autocorrelation will show values that are clustered. Negative autocorrelation is dispersed. Random is close to zero. The tool generates a Z-score and p-value which helps evaluate the significance of the Moran's index.

The Moran's I statistic for spatial autocorrelation is given as:

$$I = \frac{n}{S_0} \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} z_i z_j}{\sum_{i=1}^{n} z_i^2} \quad (1)$$

where z_i is the deviation of an attribute for feature i from its mean $(x_i - \bar{X})$, $w_{i,j}$ is the spatial weight between feature i and j, n is equal to the total number of features, and S_0 is the aggregate of all the spatial weights:

$$S_0 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} \quad (2)$$

The z_I-score for the statistic is computed as:

$$z_I = \frac{I - \mathrm{E}[I]}{\sqrt{\mathrm{V}[I]}} \quad (3)$$

where:

$$\mathrm{E}[I] = -1/(n-1) \quad (4)$$

$$\mathrm{V}[I] = \mathrm{E}[I^2] - \mathrm{E}[I]^2 \quad (5)$$

The output of the Moran's I tool can be found in the results section of ArcGIS. Upon opening the HTML report for the Moran's I results you will see a graph showing how the tool calculated the data and whether or not the data is dispersed, random, or clustered. This report will also include the Moran's Index value, z-score, p-value. It will also provide a scale for the significance of the p-value and critical value for the z-score.

As decided in the hotspot analysis there is clustering of the data. The spatial autocorrelation tool indicates that clustering is occurring with regard to the percentage of returning bank loans in a state at their respective locations with regard to the concerned state.

Annexure

Code and Discussion

```
import pandas as pd

import geopandas as gpd
```

#“pandas” and “geopandas” libraries in Python both of these libraries are commonly used for data manipulation and analysis, with geopandas specifically designed for working with geospatial data.

#read the dataset

df= pd.read_csv("loans-3.csv")

df

	paid_status	status	Age	gender	education	married_status	city	province	agerange	salary	civil	appcount	phonegrade	simstrength
0	Paid	0	35	Male	Post Graduate	Married	Daskroi	GUJARAT	25-35	6	569	7	1	3
1	Paid	0	32	Male	Self Employed	Unmarried	Guntur	ANDHRA PRADESH	25-35	4	734	8	2	1
2	Paid	0	27	Male	Graduate	Unmarried	Rajkot	GUJARAT	25-35	5	641	6	1	1
3	Paid	0	25	Male	Post Graduate	Unmarried	Vadodara	GUJARAT	25-35	5	800	4	3	3
4	Paid	0	26	Male	Post Graduate	Unmarried	Bhokar	MAHARASHTRA	25-35	3	616	5	5	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3850	Paid	0	32	Male	Graduate	Married	Dabhoi	GUJARAT	25-35	4	655	2	3	2
3851	Pending Payment	1	31	Female	Self Employed	Married	Mumbai	Maharashtra	25-35	5	505	4	3	5
3852	Paid	0	31	Male	Post Graduate	Unmarried	Chennai City Corporation	TAMIL NADU	25-35	4	777	7	2	1
3853	Paid	0	25	Male	Post Graduate	Unmarried	Kalyan	MAHARASHTRA	25-35	4	739	6	4	2
3854	Paid	0	39	Male	Self Employed	Married	Hyderabad	Telangana	>35	4	670	2	4	3

#

see unique values in the 'paid_status' column

print(df['paid_status'].unique())

see unique values in the 'province' column

print(df['province'].unique())

```
['Paid' 'Pending Payment']
['GUJARAT' 'ANDHRA PRADESH' 'MAHARASHTRA' 'Maharashtra' 'TAMIL NADU'
 'ODISHA' 'Gujarat' 'KARNATAKA' 'Telangana' 'ASSAM' 'MADHYA PRADESH'
 'Karnataka' 'Andhra Pradesh' 'Tamil Nadu' 'RAJASTHAN' 'Madhya Pradesh'
 'HARYANA' 'KERALA' 'TELANGANA' 'UTTARAKHAND' 'UTTAR PRADESH' 'Odisha'
 'PUNJAB' 'WEST BENGAL' 'Rajasthan' 'CHATTISGARH' 'BIHAR' 'JHARKHAND'
 'Kerala' 'Assam' 'DELHI' 'Jammu and Kashmir']
```

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3855 entries, 0 to 3854
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   paid_status     3855 non-null   object
 1   status          3855 non-null   int64
 2   Age             3855 non-null   int64
 3   gender          3855 non-null   object
 4   education       3833 non-null   object
 5   married_status  3855 non-null   object
 6   city            3767 non-null   object
 7   province        3855 non-null   object
 8   agerange        3855 non-null   object
 9   salary          3855 non-null   int64
 10  civil           3855 non-null   int64
 11  appcount        3855 non-null   int64
 12  phonegrade      3855 non-null   int64
 13  simstrength     3855 non-null   int64
dtypes: int64(7), object(7)
memory usage: 421.8+ KB
```

#Count occurrence of 0(paid) and 1(pending payment) from status column and and display them

```
paid_status_count= df['paid_status'].value_counts(dropna = False)
```

paid_status_count

```
Paid              3202
Pending Payment    653
Name: paid_status, dtype: int64
```

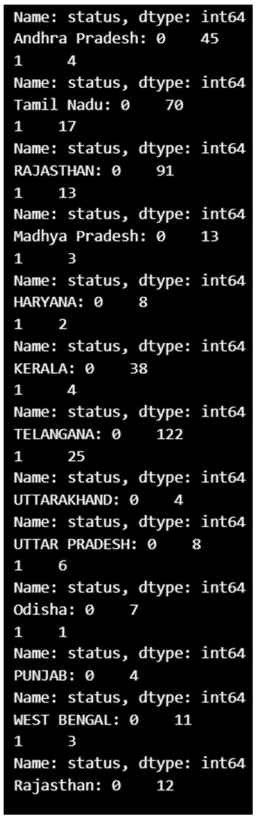
count the occurrences of 0 and 1 from the province column attributes

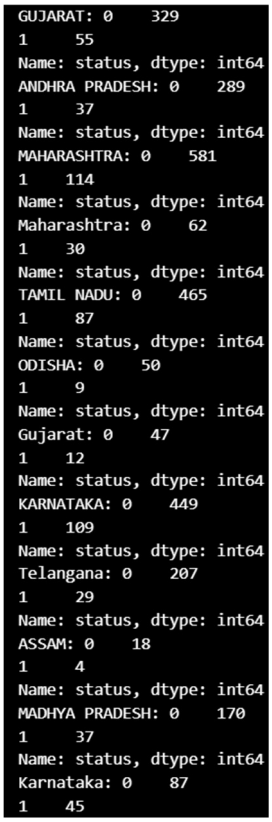
```
gujarat_df = df[df['province'] == 'GUJARAT']
counts = gujarat_df['status'].value_counts()
print(counts)
```

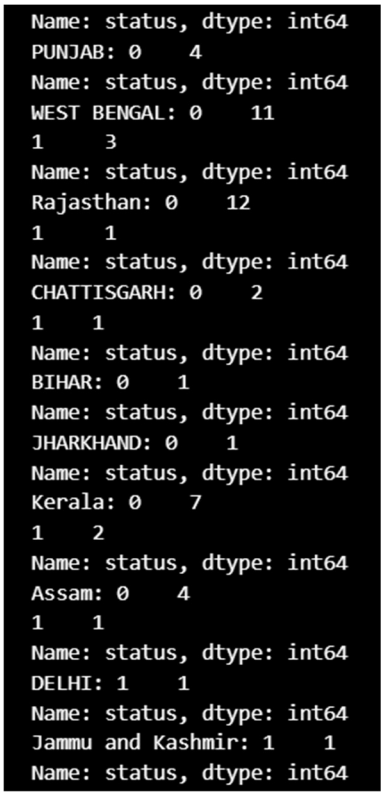
```
0    329
1     55
Name: status, dtype: int64
```

get all unique values of 'province'

```
provinces = df['province'].unique()
# iterate over each province and count the occurrences of 0 and 1 for every
attributes in province column and then display their corresponding values in op
for province in provinces:
    province_df = df[df['province'] == province]
counts = province_df['status'].value_counts()
    print(f"{province}: {counts}")
```








```
# create an empty dictionary to store the counts
counts_dict = {}
# We create a new dictionary and then store 0 and 1.After we create a data frame
and the columns are province ,0 state and 1 state and store them.
# iterate over each province and count the occurrences of 0 and 1
for province in df['province'].unique():
```

```
    province_df = df[df['province'] == province]
    counts = province_df['status'].value_counts()
    counts_dict[province] = {
        '0 state': counts.get(0, 0),
        '1 state': counts.get(1, 0)
    }

# create a new DataFrame from the counts dictionary
counts_df = pd.DataFrame.from_dict(counts_dict, orient='index')

# add a 'province' column to the new DataFrame
counts_df['province'] = counts_df.index

# reset the index of the new DataFrame
counts_df = counts_df.reset_index(drop=True)

# reorder the columns of the new DataFrame
counts_df = counts_df[['province', '0 state', '1 state']]

print(counts_df)


# We calculate the percentage of 0 state values for each province.Then we are
clearly known about how many people from each  state are paid or not.
counts_df['percentage'] = counts_df.apply(lambda row: row['0 state'] / (row['0 state']
+ row['1 state']) * 100, axis=1)

print(counts_df)
```

	province	0 state	1 state	percentage
0	GUJARAT	329	55	85.677083
1	ANDHRA PRADESH	289	37	88.650307
2	MAHARASHTRA	581	114	83.597122
3	Maharashtra	62	30	67.391304
4	TAMIL NADU	465	87	84.239130
5	ODISHA	50	9	84.745763
6	Gujarat	47	12	79.661017
7	KARNATAKA	449	109	80.465950
8	Telangana	207	29	87.711864
9	ASSAM	18	4	81.818182
10	MADHYA PRADESH	170	37	82.125604
11	Karnataka	87	45	65.909091
12	Andhra Pradesh	45	4	91.836735
13	Tamil Nadu	70	17	80.459770
14	RAJASTHAN	91	13	87.500000
15	Madhya Pradesh	13	3	81.250000
16	HARYANA	8	2	80.000000
17	KERALA	38	4	90.476190
18	TELANGANA	122	25	82.993197
19	UTTARAKHAND	4	0	100.000000
20	UTTAR PRADESH	8	6	57.142857
21	Odisha	7	1	87.500000
22	PUNJAB	4	0	100.000000
23	WEST BENGAL	11	3	78.571429
24	Rajasthan	12	1	92.307692
25	CHATTISGARH	2	1	66.666667
26	BIHAR	1	0	100.000000
27	JHARKHAND	1	0	100.000000
28	Kerala	7	2	77.777778
29	Assam	4	1	80.000000
30	DELHI	0	1	0.000000
31	Jammu and Kashmir	0	1	0.000000

```
# change the data frame name to 'dataset1'

dataset1 = counts_df

# print the new dataframe

print(dataset1.head())

#We read the second data set to find out every province and  their geometry(latitude
and longitude)

filename = "india_state_geo.json.json"

file = open(filename)

dataset2= gpd.read_file(file)

Dataset2
```

	ID_0	ISO	NAME_0	ID_1	NAME_1	NL_NAME_1	VARNAME_1	TYPE_1	ENGTYPE_1	geometry
0	105	IND	India	1	Andaman and Nicobar	NaN	Andaman & Nicobar Islands|Andaman et Nicobar|I...	Union Territor	Union Territory	MULTIPOLYGON (((93.78773 6.85264, 93.78849 6.8...
1	105	IND	India	2	Andhra Pradesh	NaN	NaN	State	State	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
2	105	IND	India	3	Arunachal Pradesh	NaN	Agence de la Frontire du Nord-Est(French-obsol...	State	State	POLYGON ((96.15778 29.38310, 96.16380 29.37668...
3	105	IND	India	4	Assam	NaN	NaN	State	State	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
4	105	IND	India	5	Bihar	NaN	NaN	State	State	MULTIPOLYGON (((88.10548 26.53904, 88.10505 26...
5	105	IND	India	6	Chandigarh	NaN	NaN	Union Territor	Union Territory	POLYGON ((76.81981 30.68583, 76.81051 30.68495...
6	105	IND	India	7	Chhattisgarh	NaN	NaN	State	State	POLYGON ((83.32760 24.09965, 83.34575 24.09707...
7	105	IND	India	8	Dadra and Nagar Haveli	NaN	DAdra et Nagar Haveli|Dadra e Nagar Haveli	Union Territor	Union Territory	POLYGON ((72.99046 20.29209, 73.00357 20.29314...
8	105	IND	India	9	Daman and Diu	NaN	NaN	Union Territor	Union Territory	MULTIPOLYGON (((72.86014 20.47096, 72.86340 20...

From the new dataset we don't take all columns ,we drop out all columns except city and geometry column

```
dataset2=dataset2.drop(['ID_0','ISO','NAME_0','ID_1','NL_NAME_1','VARNAME_1','TYPE_1','ENGTYPE_1'], axis=1)
```

print the updated dataframe

```
print(dataset2)
```

	NAME_1	geometry
0	Andaman and Nicobar	MULTIPOLYGON (((93.78773 6.85264, 93.78849 6.8...
1	Andhra Pradesh	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
2	Arunachal Pradesh	POLYGON ((96.15778 29.38310, 96.16380 29.37668...
3	Assam	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
4	Bihar	MULTIPOLYGON (((88.10548 26.53904, 88.10505 26...
5	Chandigarh	POLYGON ((76.81981 30.68583, 76.81051 30.68495...
6	Chhattisgarh	POLYGON ((83.32760 24.09965, 83.34575 24.09707...
7	Dadra and Nagar Haveli	POLYGON ((72.99046 20.29209, 73.00357 20.29314...
8	Daman and Diu	MULTIPOLYGON (((72.86014 20.47096, 72.86340 20...
9	Delhi	POLYGON ((77.21732 28.83528, 77.21365 28.81350...
10	Goa	MULTIPOLYGON (((73.78181 15.35569, 73.78181 15...
11	Gujarat	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
12	Haryana	POLYGON ((76.83715 30.87887, 76.85243 30.87069...
13	Himachal Pradesh	POLYGON ((76.80276 33.23666, 76.80630 33.23623...
14	Jammu and Kashmir	POLYGON ((77.89957 35.42789, 77.90297 35.42759...
15	Jharkhand	POLYGON ((87.59989 25.31466, 87.60688 25.31138...
16	Karnataka	MULTIPOLYGON (((74.67097 13.19986, 74.67097 13...

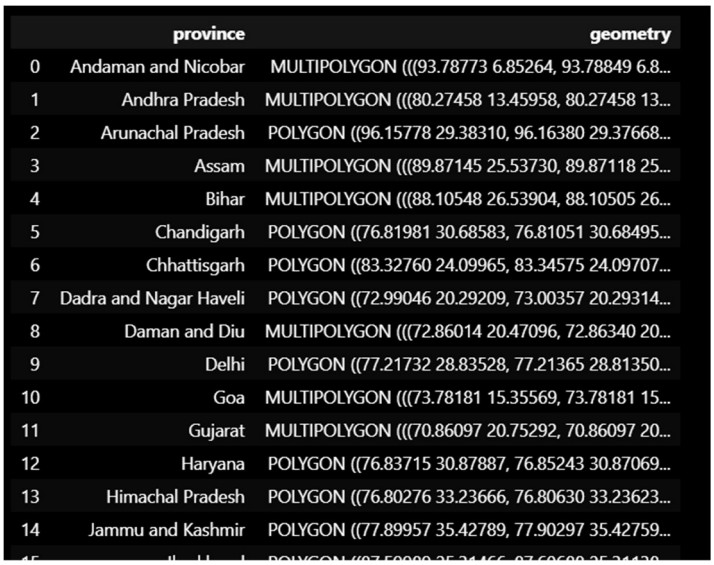
print the column names

```
print(dataset2.columns)
```

rename the 'NAME-1' column to 'province' because we want to match this two columns(province from dataset1 and NAME_1 from dataset1)

	province	geometry
0	Andaman and Nicobar	MULTIPOLYGON (((93.78773 6.85264, 93.78849 6.8...
1	Andhra Pradesh	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
2	Arunachal Pradesh	POLYGON ((96.15778 29.38310, 96.16380 29.37668...
3	Assam	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
4	Bihar	MULTIPOLYGON (((88.10548 26.53904, 88.10505 26...
5	Chandigarh	POLYGON ((76.81981 30.68583, 76.81051 30.68495...
6	Chhattisgarh	POLYGON ((83.32760 24.09965, 83.34575 24.09707...
7	Dadra and Nagar Haveli	POLYGON ((72.99046 20.29209, 73.00357 20.29314...
8	Daman and Diu	MULTIPOLYGON (((72.86014 20.47096, 72.86340 20...
9	Delhi	POLYGON ((77.21732 28.83528, 77.21365 28.81350...
10	Goa	MULTIPOLYGON (((73.78181 15.35569, 73.78181 15...
11	Gujarat	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
12	Haryana	POLYGON ((76.83715 30.87887, 76.85243 30.87069...
13	Himachal Pradesh	POLYGON ((76.80276 33.23666, 76.80630 33.23623...
14	Jammu and Kashmir	POLYGON ((77.89957 35.42789, 77.90297 35.42759...
15	Jharkhand	POLYGON ((87.59989 25.31466, 87.60688 25.31138...
16	Karnataka	MULTIPOLYGON (((74.67097 13.19986, 74.67097 13...
17	Kerala	MULTIPOLYGON (((76.46736 9.54097, 76.46736 9.5...
18	Lakshadweep	MULTIPOLYGON (((73.01014 8.28042, 73.01014 8.2...
19	Madhya Pradesh	POLYGON ((78.36465 26.86884, 78.36688 26.86259...
20	Maharashtra	MULTIPOLYGON (((73.45597 15.88986, 73.45597 15...
21	Manipur	POLYGON ((94.57723 25.64833, 94.57609 25.64470...
22	Meghalaya	POLYGON ((91.85384 26.10479, 91.86470 26.10035...
23	Mizoram	POLYGON ((92.80080 24.41905, 92.80370 24.41879...
24	Nagaland	POLYGON ((95.21445 26.93695, 95.21706 26.93420...
25	Orissa	MULTIPOLYGON (((84.76986 19.10597, 84.76986 19...
26	Puducherry	MULTIPOLYGON (((79.84486 10.82653, 79.84486 10...
27	Punjab	POLYGON ((75.86877 32.48868, 75.88712 32.47203...
28	Rajasthan	POLYGON ((73.88944 29.97761, 73.89118 29.97007...
29	Sikkim	POLYGON ((88.64526 28.09912, 88.65411 28.08984...
30	Tamil Nadu	MULTIPOLYGON (((77.55596 8.07903, 77.55596 8.0...

dataset2 = dataset2.rename(columns={'NAM



	province	geometry
0	Andaman and Nicobar	MULTIPOLYGON (((93.78773 6.85264, 93.78849 6.8...
1	Andhra Pradesh	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
2	Arunachal Pradesh	POLYGON ((96.15778 29.38310, 96.16380 29.37668...
3	Assam	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
4	Bihar	MULTIPOLYGON (((88.10548 26.53904, 88.10505 26...
5	Chandigarh	POLYGON ((76.81981 30.68583, 76.81051 30.68495...
6	Chhattisgarh	POLYGON ((83.32760 24.09965, 83.34575 24.09707...
7	Dadra and Nagar Haveli	POLYGON ((72.99046 20.29209, 73.00357 20.29314...
8	Daman and Diu	MULTIPOLYGON (((72.86014 20.47096, 72.86340 20...
9	Delhi	POLYGON ((77.21732 28.83528, 77.21365 28.81350...
10	Goa	MULTIPOLYGON (((73.78181 15.35569, 73.78181 15...
11	Gujarat	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
12	Haryana	POLYGON ((76.83715 30.87887, 76.85243 30.87069...
13	Himachal Pradesh	POLYGON ((76.80276 33.23666, 76.80630 33.23623...
14	Jammu and Kashmir	POLYGON ((77.89957 35.42789, 77.90297 35.42759...

E_1': 'province'})

print the column names again

print(dataset2.columns)

convert all the strings in the dataframe to uppercase

dataset2= dataset2.apply(lambda x: x.str.upper() if x.dtype == "province" else x)

print the resulting dataframe

print(dataset2)

#using merge function by setting how='inner'

dataset3=pd.merge(dataset1,dataset2,on='province',how='outer')

dataset3

	province	0 state	1 state	percentage	geometry
0	GUJARAT	329.0	55.0	85.677083	None
1	ANDHRA PRADESH	289.0	37.0	88.650307	None
2	MAHARASHTRA	581.0	114.0	83.597122	None
3	Maharashtra	62.0	30.0	67.391304	MULTIPOLYGON (((73.45597 15.88986, 73.45597 15...
4	TAMIL NADU	465.0	87.0	84.239130	None
5	ODISHA	50.0	9.0	84.745763	None
6	Gujarat	47.0	12.0	79.661017	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
7	KARNATAKA	449.0	109.0	80.465950	None
8	Telangana	207.0	29.0	87.711864	None
9	ASSAM	18.0	4.0	81.818182	None
10	MADHYA PRADESH	170.0	37.0	82.125604	None
11	Karnataka	87.0	45.0	65.909091	MULTIPOLYGON (((74.67097 13.19986, 74.67097 13...
12	Andhra Pradesh	45.0	4.0	91.836735	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
13	Tamil Nadu	70.0	17.0	80.459770	MULTIPOLYGON (((77.55596 8.07903, 77.55596 8.0...
14	RAJASTHAN	91.0	13.0	87.500000	None
15	Madhya Pradesh	13.0	3.0	81.250000	POLYGON ((78.36465 26.86884, 78.36688 26.86259...
16	HARYANA	8.0	2.0	80.000000	None
17	KERALA	38.0	4.0	90.476190	None
18	TELANGANA	122.0	25.0	82.993197	None
19	UTTARAKHAND	4.0	0.0	100.000000	None

#using merge function by setting how='inner'

dataset3=pd.merge(dataset1,dataset2,on='province',how='outer')

dataset3

	province	0 state	1 state	percentage	geometry
0	GUJARAT	329.0	55.0	85.677083	None
1	ANDHRA PRADESH	289.0	37.0	88.650307	None
2	MAHARASHTRA	581.0	114.0	83.597122	None
3	Maharashtra	62.0	30.0	67.391304	MULTIPOLYGON (((73.45597 15.88986, 73.45597 15...
4	TAMIL NADU	465.0	87.0	84.239130	None
5	ODISHA	50.0	9.0	84.745763	None
6	Gujarat	47.0	12.0	79.661017	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
7	KARNATAKA	449.0	109.0	80.465950	None
8	Telangana	207.0	29.0	87.711864	None
9	ASSAM	18.0	4.0	81.818182	None
10	MADHYA PRADESH	170.0	37.0	82.125604	None
11	Karnataka	87.0	45.0	65.909091	MULTIPOLYGON (((74.67097 13.19986, 74.67097 13...
12	Andhra Pradesh	45.0	4.0	91.836735	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
13	Tamil Nadu	70.0	17.0	80.459770	MULTIPOLYGON (((77.55596 8.07903, 77.55596 8.0...
14	RAJASTHAN	91.0	13.0	87.500000	None
15	Madhya Pradesh	13.0	3.0	81.250000	POLYGON ((78.36465 26.86884, 78.36688 26.86259...
16	HARYANA	8.0	2.0	80.000000	None
17	KERALA	38.0	4.0	90.476190	None
18	TELANGANA	122.0	25.0	82.993197	None
19	UTTARAKHAND	4.0	0.0	100.000000	None
20	UTTAR PRADESH	8.0	6.0	57.142857	None
21	Odisha	7.0	1.0	87.500000	None
22	PUNJAB	4.0	0.0	100.000000	None
23	WEST BENGAL	11.0	3.0	78.571429	None

missing_geometry = dataset2[dataset2['geometry'].isnull()]

print(missing_geometry)

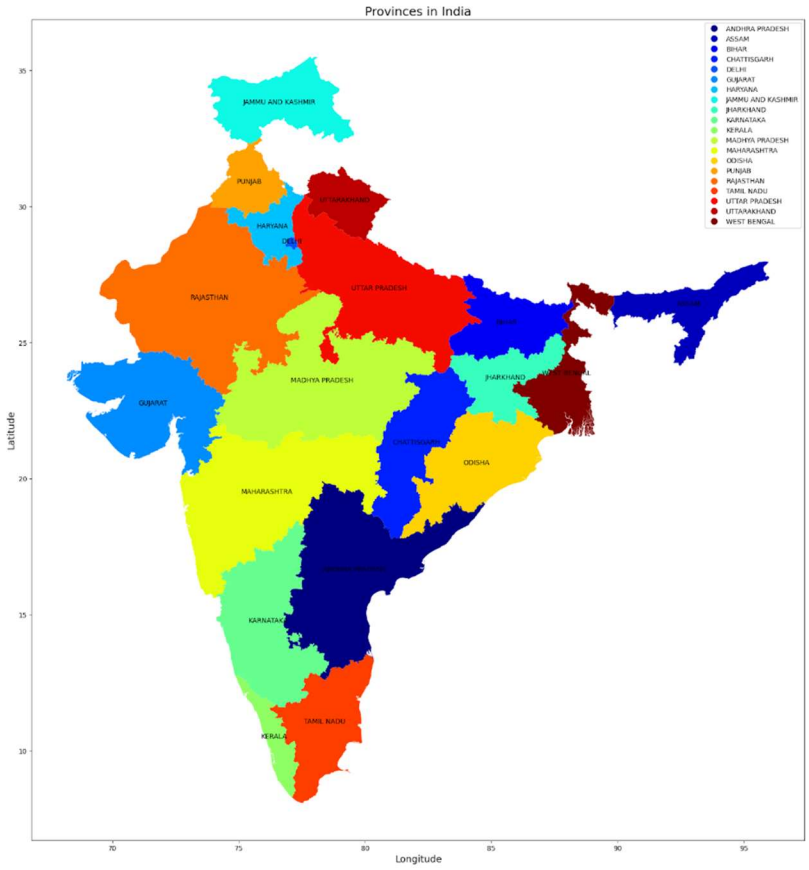
```
dataset1['province'] = dataset1['province'].str.upper()

dataset2['province'] = dataset2['province'].str.upper()

merged_df = pd.merge(dataset1,dataset2, on='province', how='outer')

merged_df
```

	province	0 state	1 state	percentage	geometry
0	GUJARAT	329.0	55.0	85.677083	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
1	GUJARAT	47.0	12.0	79.661017	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
2	ANDHRA PRADESH	289.0	37.0	88.650307	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
3	ANDHRA PRADESH	45.0	4.0	91.836735	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
4	MAHARASHTRA	581.0	114.0	83.597122	MULTIPOLYGON (((73.45597 15.88986, 73.45597 15...
5	MAHARASHTRA	62.0	30.0	67.391304	MULTIPOLYGON (((73.45597 15.88986, 73.45597 15...
6	TAMIL NADU	465.0	87.0	84.239130	MULTIPOLYGON (((77.55596 8.07903, 77.55596 8.0...
7	TAMIL NADU	70.0	17.0	80.459770	MULTIPOLYGON (((77.55596 8.07903, 77.55596 8.0...
8	ODISHA	50.0	9.0	84.745763	None
9	ODISHA	7.0	1.0	87.500000	None
10	KARNATAKA	449.0	109.0	80.465950	MULTIPOLYGON (((74.67097 13.19986, 74.67097 13...
11	KARNATAKA	87.0	45.0	65.909091	MULTIPOLYGON (((74.67097 13.19986, 74.67097 13...
12	TELANGANA	207.0	29.0	87.711864	None
13	TELANGANA	122.0	25.0	82.993197	None
14	ASSAM	18.0	4.0	81.818182	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
15	ASSAM	4.0	1.0	80.000000	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
16	MADHYA PRADESH	170.0	37.0	82.125604	POLYGON ((78.36465 26.86884, 78.36688 26.86259...
17	MADHYA PRADESH	13.0	3.0	81.250000	POLYGON ((78.36465 26.86884, 78.36688 26.86259...
18	RAJASTHAN	91.0	13.0	87.500000	POLYGON ((73.88944 29.97761, 73.89118 29.97007...
19	RAJASTHAN	12.0	1.0	92.307692	POLYGON ((73.88944 29.97761, 73.89118 29.97007...

```
# remove duplicates and keep first occurrence

dataset1 = dataset1.drop_duplicates(subset='province', keep='first')

dataset2 = dataset2.drop_duplicates(subset='province', keep='first')

# merge datasets on province

merged_df = pd.merge(dataset1, dataset2, on='province', how='inner')

df1=gpd.GeoDataFrame(merged_df,geometry='geometry')

df1
```

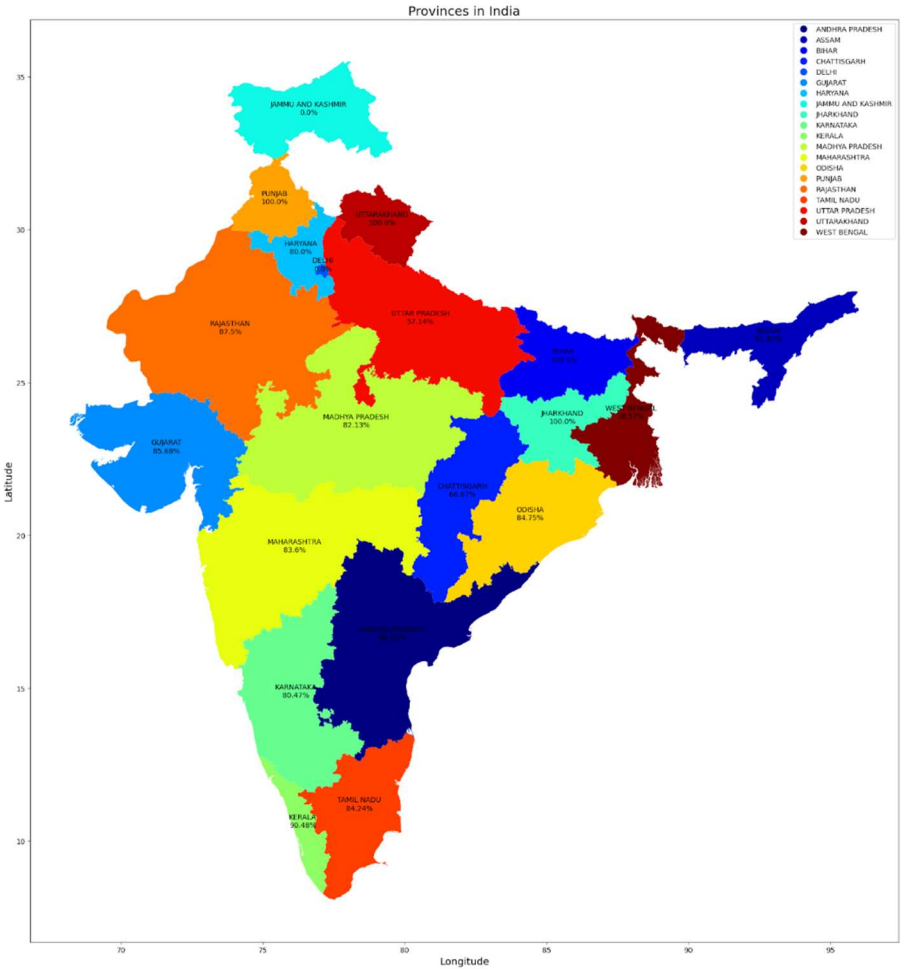
	province	0 state	1 state	percentage	geometry
0	GUJARAT	329	55	85.677083	MULTIPOLYGON (((70.86097 20.75292, 70.86097 20...
1	ANDHRA PRADESH	289	37	88.650307	MULTIPOLYGON (((80.27458 13.45958, 80.27458 13...
2	MAHARASHTRA	581	114	83.597122	MULTIPOLYGON (((73.45597 15.88986, 73.45597 15...
3	TAMIL NADU	465	87	84.239130	MULTIPOLYGON (((77.55596 8.07903, 77.55596 8.0...
4	ODISHA	50	9	84.745763	MULTIPOLYGON (((84.76986 19.10597, 84.76986 19...
5	KARNATAKA	449	109	80.465950	MULTIPOLYGON (((74.67097 13.19986, 74.67097 13...
6	ASSAM	18	4	81.818182	MULTIPOLYGON (((89.87145 25.53730, 89.87118 25...
7	MADHYA PRADESH	170	37	82.125604	POLYGON ((78.36465 26.86884, 78.36688 26.86259...
8	RAJASTHAN	91	13	87.500000	POLYGON ((73.88944 29.97761, 73.89118 29.97007...
9	HARYANA	8	2	80.000000	POLYGON ((76.83715 30.87887, 76.85243 30.87069...
10	KERALA	38	4	90.476190	MULTIPOLYGON (((76.46736 9.54097, 76.46736 9.5...
11	UTTARAKHAND	4	0	100.000000	POLYGON ((79.19478 31.35362, 79.19817 31.35196...
12	UTTAR PRADESH	8	6	57.142857	POLYGON ((77.58468 30.40878, 77.58639 30.40801...
13	PUNJAB	4	0	100.000000	POLYGON ((75.86877 32.48868, 75.88712 32.47203...
14	WEST BENGAL	11	3	78.571429	MULTIPOLYGON (((88.01861 21.57278, 88.01889 21...
15	CHATTISGARH	2	1	66.666667	POLYGON ((83.32760 24.09965, 83.34575 24.09707...
16	BIHAR	1	0	100.000000	MULTIPOLYGON (((88.10548 26.53904, 88.10505 26...
17	JHARKHAND	1	0	100.000000	POLYGON ((87.59989 25.31466, 87.60688 25.31138...
18	DELHI	0	1	0.000000	POLYGON ((77.21732 28.83528, 77.21365 28.81350...
19	JAMMU AND KASHMIR	0	1	0.000000	POLYGON ((77.89957 35.42789, 77.90297 35.42759...

```
# Plot the choropleth map with '0 state' values
ax = df1.plot(column='province', cmap='jet', figsize=(22, 22), legend=True)

# Add the province names as labels
df1.apply(lambda x: ax.annotate(text=x.province, xy=x.geometry.centroid.coords[0],
ha='center'),axis=1);

# Set plot title and axes labels
ax.set_title('Provinces with 0 state in  India', fontsize=18)
ax.set_xlabel('Longitude', fontsize=14)
ax.set_ylabel('Latitude', fontsize=14)

# Display the map
plt.show()
```




```
# Plot the choropleth map with '0 state' values
ax = df1.plot(column='province', cmap='jet', figsize=(22, 25), legend=True)

# Add the province names and percentage values as labels
df1.apply(lambda x: ax.annotate(text=f"{x.province}\n{round(x.percentage,2)}%",
                                xy=x.geometry.centroid.coords[0],
                                ha='center'),axis=1)

# Set plot title and axes labels
ax.set_title('Provinces in India', fontsize=18)
ax.set_xlabel('Longitude', fontsize=14)
ax.set_ylabel('Latitude', fontsize=14)

# Display the map
```

plt.show()



Create subplots

fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

Plot the first map on the left subplot

df1.plot(column='0 state', cmap='jet', ax=axs[0])

axs[0].set_title('Province-wise 0 State')

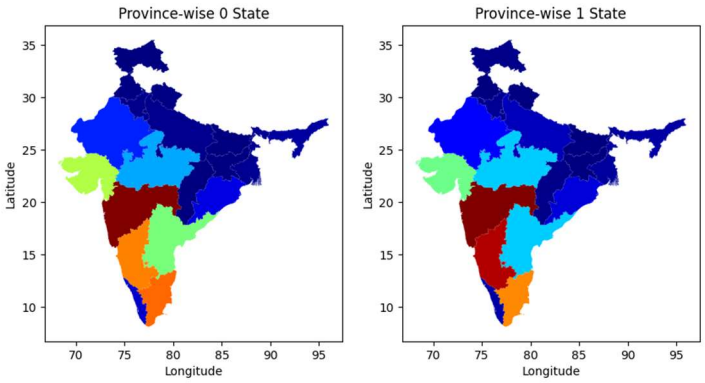
axs[0].set_xlabel('Longitude')

axs[0].set_ylabel('Latitude')

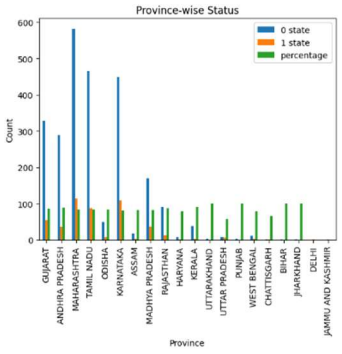
Plot the second map on the right subplot

df1.plot(column='1 state', cmap='jet', ax=axs[1])

```
axs[1].set_title('Province-wise 1 State')

axs[1].set_xlabel('Longitude')

axs[1].set_ylabel('Latitude')


# Display the map

plt.show()
```

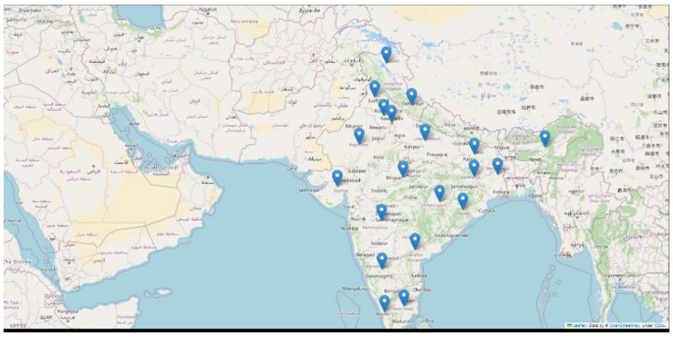



```
# plot bar graph

df1.plot(x='province', kind='bar')

plt.title('Province-wise Status')

plt.xlabel('Province')

plt.ylabel('Count')

plt.show()
```




```
import folium

# Create a new column with HTML code for the marker popups
```

```
df1['popup'] = df1['province'] + '<br>' + '0 State: ' + df1['0 state'].astype(str) + '%' +
'<br>' + '1 State: ' + df1['1 state'].astype(str) + '%'

# Create the Folium map
obj = folium.Map(location=[28.6600,77.2300], zoom_start=5)

# Add markers to the map
for _, row in df1.iterrows():
    folium.Marker(location=[row.geometry.centroid.y, row.geometry.centroid.x],
popup=row.popup).add_to(obj)

# Display the map
obj
```

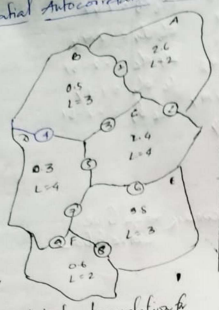


```
%matplotlib inline
import matplotlib.pyplot as plt
from libpysal.weights.contiguity import Queen
from libpysal import examples
import numpy as np
import pandas as pd
import geopandas as gpd
import os
```

```
import splot
# For this example we will focus on the Donatns (charitable donations per capita)
variable. We will
# calculate Contiguity weights w with libpysals Queen.from_dataframe(gdf) . Then
we
# transform our weights to be row-standardized.
y = df1['1 state'].values
w = Queen.from_dataframe(df1)
w.transform = 'r'
# Assessing Global Spatial Autocorrelation
# We calculate Moran's I. A test for global autocorrelation for a continuous attribute.
from esda.moran import Moran
w = Queen.from_dataframe(df1)
moran = Moran(y, w)
moran.I
0.4291954923163637
# Our value for the statistic is interpreted against a reference distribution under the
null
# hypothesis of complete spatial randomness. PySAL uses the approach of random
spatial
# permutations.
from splot.esda import moran_scatterplot
fig, ax = moran_scatterplot(moran, aspect_equal=True)
plt.show()
```

Spatial Autocorrelation Moran's coefficient

$\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij} = 18$



spatial autocorrelation for areas

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	1	0	1	1	0	0
C	1	1	0	1	1	0
D	0	1	1	0	1	1
E	0	0	1	1	0	1
F	0	0	0	1	1	0

calculating Moran's I :-

$$I = \left(\frac{N}{\sum_i \sum_j W_{ij}}\right) \cdot \frac{\sum_i \sum_j (x_i - \bar{x})(x_j - \bar{x})}{\sum (x_i - \bar{x})^2}$$

	x	$d = (x-\bar{x})$	$(x-\bar{x})^2$
A	2.6	d_1 0.9	0.81
B	0.5	d_2 -1.2	1.44
C	2.4	d_3 0.7	0.49
D	0.3	d_4 -1.4	1.96
E	3.8	d_5 2.1	4.41
F	0.6	d_6 -1.1	1.21

i,j	1/A	2/B	3/C	4/D	5/E	6/F
1/A	0	d_1d_2	d_1d_3	0	0	0
2/B	d_2d_1	0	d_2d_3	d_2d_4	0	0
3/C	d_3d_1	d_3d_2	0	d_3d_4	d_3d_5	0
4/D	0	d_4d_2	d_4d_3	0	d_4d_5	d_4d_6
5/E	0	0	d_5d_3	d_5d_4	0	d_5d_6
6/F	0	0	0	d_6d_4	d_6d_5	0

$$I = \left(\frac{6}{18}\right) \frac{\sum_i \sum_j (W_{ij}(x_i-\bar{x})(x_j-\bar{x}))}{10.32}$$

$d_1d_2 = -1.08$ $d_2d_1 = -0.98$ $d_3d_2 = 1.47$

$d_1d_3 = 0.63$ $d_3d_1 = 0.63$ $d_3d_4 = -2.94$

$d_2d_1 = -1.08$ $d_3d_2 = 1.47$ $d_4d_2 = -2.31$

$d_2d_3 = -0.84$ $d_4d_2 = 1.68$ $d_4d_3 = 1.54$

$d_3d_2 = -0.84$ $d_4d_3 = -0.98$ $d_4d_5 = -2.31$

$d_4d_5 = -2.94$

$d_5d_3 = 1.47$

$d_5d_4 = 1.68$

$d_5d_6 = -2.31$

$d_6d_4 = 1.54$

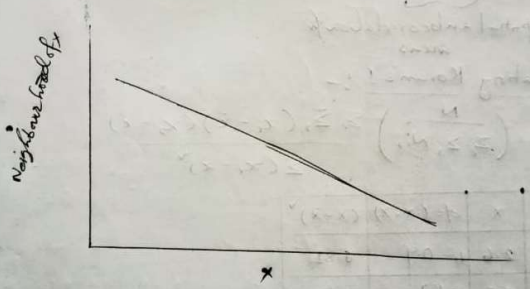
$d_6d_5 = -2.31$

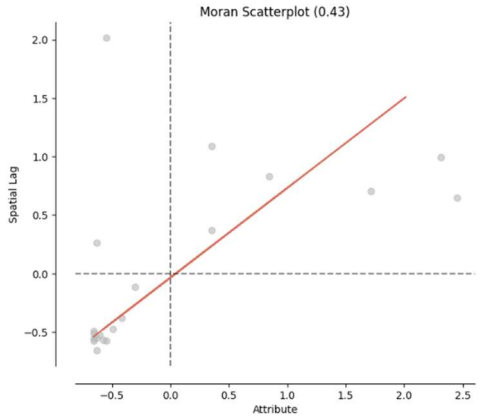
$$I = \left(\frac{6}{18}\right) \cdot \frac{-5.66}{10.32}$$

$$= -\left(\frac{1}{3}\right) 0.55$$

$$= -0.183$$

⇒ Some weak negative correlation

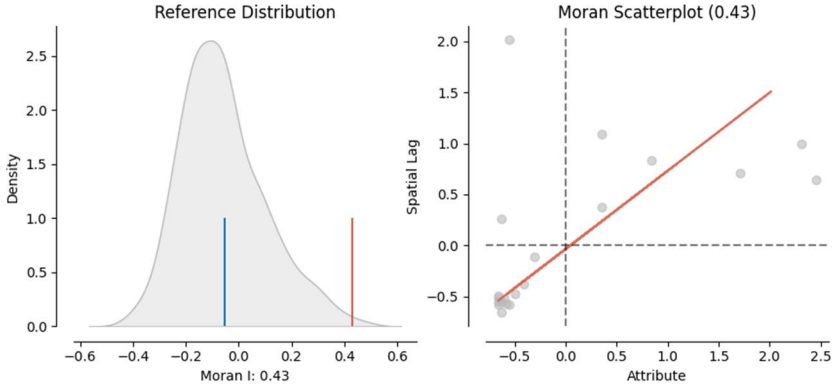




from splot.esda import plot_moran

plot_moran(moran, zstandard=True, figsize=(10,4))

plt.show()



Our observed value is statistically significant

moran.p_sim

0.006

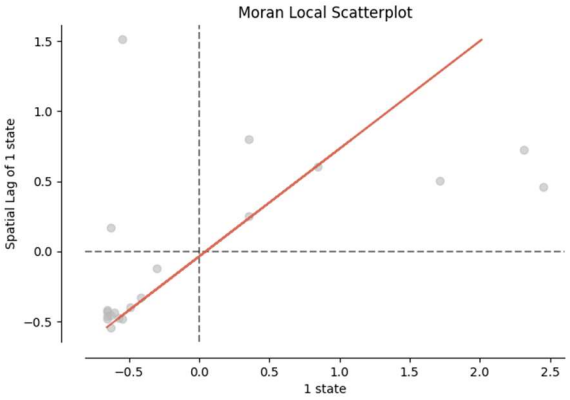
Visualizing Local Autocorrelation with splot - Hot Spots, Cold Spots and

Spatial Outliers

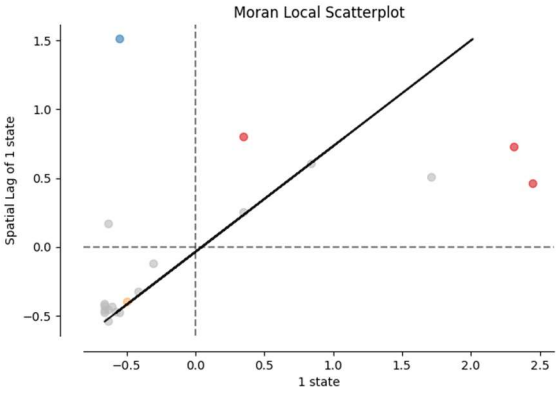
In addition to visualizing Global autocorrelation statistics, splot has options to visualize local

autocorrelation statistics. We compute the local Moran m . Then, we plot the spatial lag 1 state and the

```
#  Spatial Lag of 1 state variable in a Moran Scatterplot.
from splot.esda import moran_scatterplot
from esda.moran import Moran_Local
# calculate Moran_Local and plot
moran_loc = Moran_Local(y, w)
fig, ax = moran_scatterplot(moran_loc)
ax.set_xlabel('1 state')
ax.set_ylabel('Spatial Lag of 1 state')
plt.show()
```




```
fig, ax = moran_scatterplot(moran_loc, p=0.05)
ax.set_xlabel('1 state')
ax.set_ylabel('Spatial Lag of 1 state')
plt.show()
```



We can distinguish the specific type of local spatial autocorrelation in High-High, Low-Low,

High-Low, Low-High. Where the upper right quadrant displays HH, the lower left, LL, the upper

left LH and the lower left HL.

These types of local spatial autocorrelation describe similarities or dissimilarities between a

specific polygon with its neighboring polygons. The upper left quadrant for example indicates

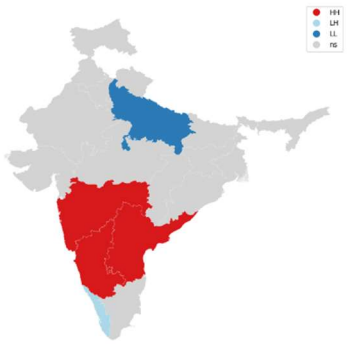
that polygons with low values are surrounded by polygons with high values (LH). The lower right

quadrant shows polygons with high values surrounded by neighbors with low values (HL). This

# indicates an association of dissimilar values.

Let's now visualize the areas we found to be significant on a map:

```
from splot.esda import lisa_cluster

lisa_cluster(moran_loc, df1, p=0.05, figsize = (9,9))

plt.show()
```



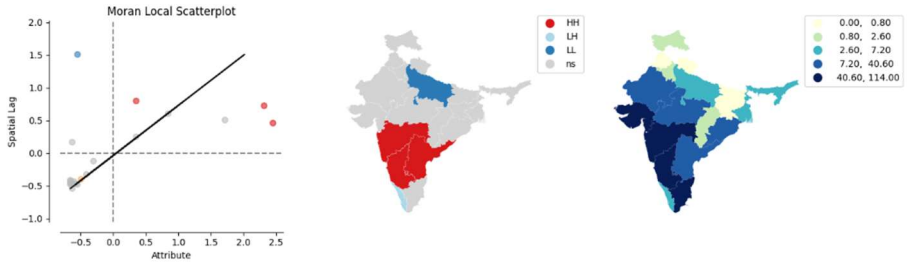
Combined visualizations

Often, it is easier to asses once statistical results or interpret these results comparing different

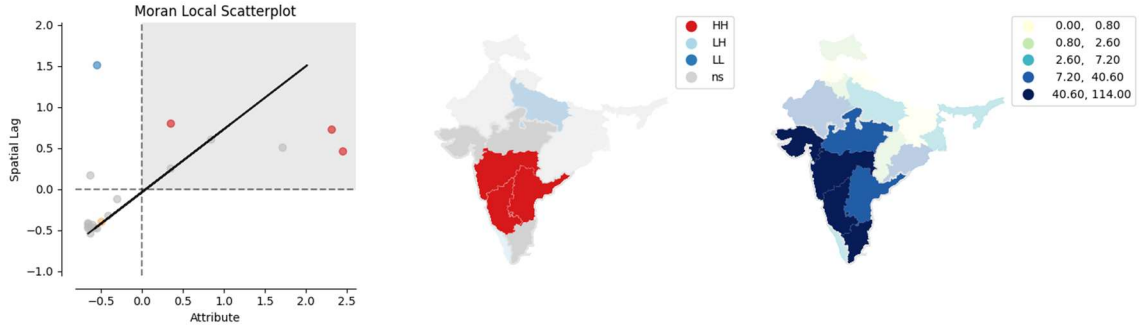
visualizations. Here we for example look at a static visualization of a Moran Scatterplot, LISA

cluster map and choropleth map.

```
from splot.esda import plot_local_autocorrelation

plot_local_autocorrelation(moran_loc, df1, '1 state')

plt.show()
```




```
plot_local_autocorrelation(moran_loc, df1, '1 state', quadrant=1)

plt.show()
```



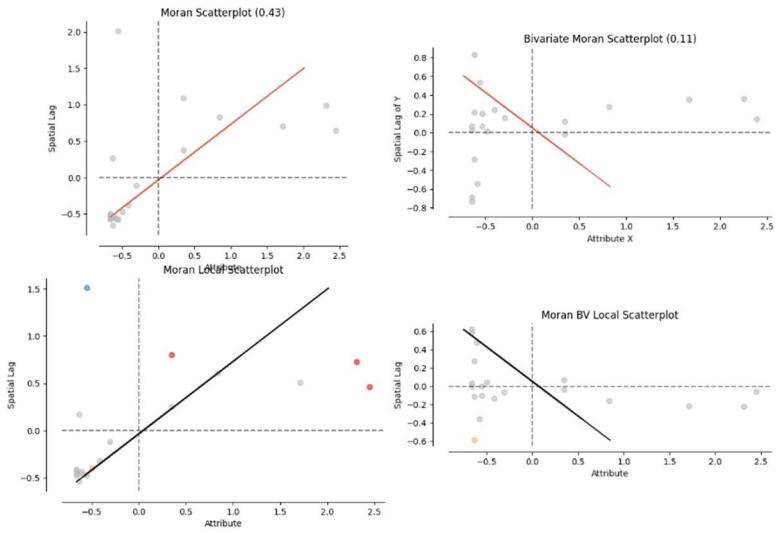
Bivariate Moran Statistics

Additionally, to assessing the correlation of one variable over space. It is possible to inspect the

relationwhip of two variables and their position in space with so called Bivariate Moran Statistics.

These can be found in esda.moran.Moran_BV .

```
from esda.moran import Moran_BV, Moran_Local_BV
from splot.esda import plot_moran_bv_simulation, plot_moran_bv
# Next to y we will also be looking at the suicide rate x .
x = df1['percentage'].values
# Before we dive into Bivariate Moran startistics, let's make a quick overview which
esda.moran
# objects are supported by moran_scatterplot :
moran = Moran(y,w)
moran_bv = Moran_BV(y, x, w)
moran_loc = Moran_Local(y, w)
moran_loc_bv = Moran_Local_BV(y, x, w)
fig, axs = plt.subplots(2, 2, figsize=(15,10),
                        subplot_kw={'aspect': 'equal'})

moran_scatterplot(moran, ax=axs[0,0])
moran_scatterplot(moran_loc, p=0.05, ax=axs[1,0])
moran_scatterplot(moran_bv, ax=axs[0,1])
moran_scatterplot(moran_loc_bv, p=0.05, ax=axs[1,1])
plt.show()
```



As you can see an easy moran_scatterplot call provides you with loads of options. Now what

are Bivariate Moran Statistics?

Bivariate Moran Statistics describe the correlation between one variable and the spatial lag of

another variable. Therefore, we have to be careful interpreting our results. Bivariate Moran

Statistics do not take the inherent correlation between the two variables at the same location

into account. They much more offer a tool to measure the degree one polygon with a specific

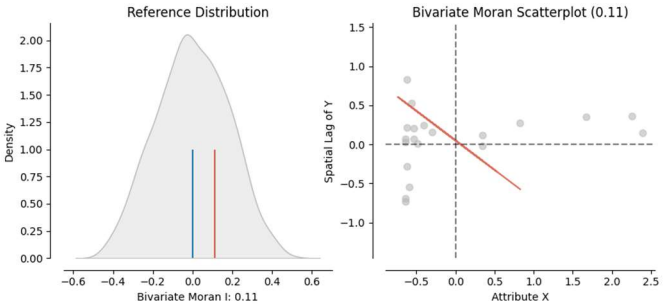
attribute is correlated with its neighboring polygons with a different attribute.

splot can offer help interpreting the results by providing visualizations of reference

distributions and a Moran Scatterplot:

plot_moran_bv(moran_bv)

plt.show()

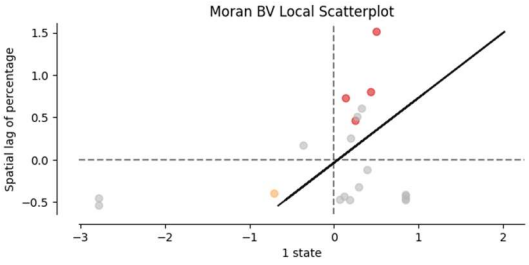


Local Bivariate Moran Statistics

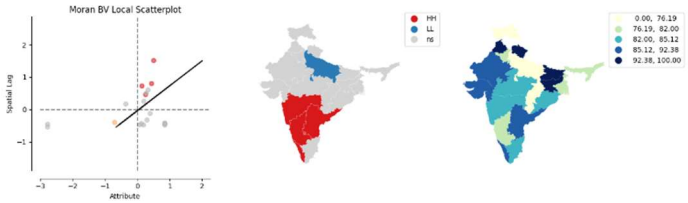
Similar to univariate local Moran statistics pysal and splot offer tools to asses local

autocorrelation for bivariate analysis:

```
from esda.moran import Moran_Local_BV

moran_loc_bv = Moran_Local_BV(x, y, w)

fig, ax = moran_scatterplot(moran_loc_bv, p=0.05)

ax.set_xlabel('1 state')

ax.set_ylabel('Spatial lag of percentage')

plt.show()
```




```
plot_local_autocorrelation(moran_loc_bv, df1, 'percentage')

plt.show()
```



Discussion

Advantages

Risk evaluation: To evaluate the risk involved with a loan application, a loan prediction model can analyze historical data as well as a number of other variables. This lowers the risk of defaults and potential losses by enabling lenders to make well-informed judgements about whether to approve or reject loan applications.

Greater efficiency: Lenders can work much more efficiently by automating the loan projection process. When compared to manual examination, the model can quickly analyze huge numbers of loan applications, saving time and resources. This enables lenders to process loan applications more quickly and respond to applicants more quickly.

Objectivity and consistency:

Human judgment is vulnerable to subjectivity and lacks objectivity. A loan prediction model ensures fairness and impartiality in the decision-making process by consistently applying established rules and algorithms to each loan application. By doing this, the possibility of bias or discrimination in lending practices is diminished.

heightened accuracy:

Machine learning models can examine enormous volumes of data and spot patterns that humans would miss. A loan prediction model that takes advantage of this capability can offer more accurate creditworthiness evaluations, assisting lenders in making more informed decisions and lowering the risk of granting loans for borrowers who have a higher chance of defaulting.

Reduced default rates: By precisely forecasting applicants' creditworthiness, a loan prediction model can assist lenders in reducing default rates. By identifying high-risk applications, lenders can avoid making loans to people or firms who are more likely to default, resulting in better portfolio performance and smaller financial losses.

Savings: Lenders can save money by automating the loan forecast process. Lenders can save operating expenses and utilize resources more efficiently by eliminating the need for manual underwriting and expediting the application review process.

Loan prediction models are easily **scalable** to handle huge quantities of loan applications. As loan demand grows, the model can handle the increased workload without sacrificing accuracy or efficiency, making it suited for both small-scale and large-scale lending operations.

Continuous learning and improvement: Machine learning models may learn and change in response to new data. A loan prediction model may be updated and adjusted on a regular basis, including new information and changing projections to enhance performance over time.

Overall, a loan prediction model may give lenders with useful insights and tools to help them make better lending decisions, resulting in lower risks, more efficiency, and better outcomes for both lenders and borrowers.

DISADVANTAGES

Data scarcity: Loan prediction algorithms rely largely on previous data for training and prediction. The model may not adequately depict the complexity and subtleties of the lending landscape if data is sparse or missing. As a result, forecasts may be less accurate and judgements may be erroneous.

Data bias: Bias in previous data can impact loan forecast models. If historical data contains prejudices based on race, gender, or other protected characteristics, the model may unintentionally perpetuate these biases in loan choices. This can lead to unfair lending practises and further exacerbate existing imbalances.

Interpretability: Some loan prediction models, particularly complicated machine learning algorithms such as deep neural networks, are difficult to comprehend. Understanding the

particular aspects or variables that lead to a loan application being granted or refused can be difficult. Because of this lack of transparency, it may be difficult to explain or defend the model's judgements, perhaps leading to mistrust or regulatory problems.

Lending is dynamic and evolving: The lending environment is always changing, impacted by economic conditions, industry developments, and regulatory laws. Loan prediction algorithms may struggle to adjust fast to these changing conditions, perhaps resulting in obsolete projections or failure to recognise rising dangers. Regular model updates and monitoring are required to guarantee that the model remains relevant and accurate.

Over Reliance on previous patterns: Loan prediction models create forecasts about future loan performance largely on historical data. However, previous patterns are not necessarily predictive of future behavior, particularly during times of major economic or market turmoil. Models may fail to account for unusual occurrences, resulting in erroneous projections and consequently increased risk exposure.

Incomplete evaluation of borrower characteristics: Loan prediction models often depend on quantitative data to assess applicants' creditworthiness, such as credit scores and financial indicators. These models, however, may not completely capture significant qualitative elements, such as personal circumstances, character, or entrepreneurial ability, which may be relevant in loan application evaluation. This can lead to lost opportunities to assist eligible consumers with minimal credit history or unusual financial characteristics.

Loan prediction algorithms, like any other machine learning model, are subject to adversarial assaults. Malicious actors may attempt to influence or confuse the model by giving purposefully false or misleading information in loan applications. If the model fails to identify these assaults, it may result in incorrect loan approvals and financial losses for lenders.

To avoid these drawbacks, it is critical to regularly monitor and analyze the performance of the loan prediction model, maintain the quality and diversity of data, solve bias and interpretability problems, and supplement the algorithm's forecasts with human judgment and knowledge.

Future Work

1.Our results here were obtained from training models on a single dataset. As suggested we find out thre risk zone in credit scoring be examined better with more data and using more datasets can help in this re-gard.

2.We can expand the project's dataset by integrating additional relevant data sources. This could include socio-economic data, demographic information, or even real-time financial data to improve the accuracy and robustness of risk assessment.

3. Create a web-based visualization interface to make the project's findings and risk zones accessible to a wider audience. This can involve designing interactive maps, charts, and dashboards that allow users to explore the data and understand the risk zones intuitively.

4. As the project deals with sensitive financial and personal information, it is crucial to ensure data privacy and comply with ethical guidelines. Future work should focus on implementing appropriate data anonymization techniques, obtaining necessary consent, and maintaining the highest standards of data security throughout the project.

Reference

1.Credit Scoring using Machine Learning Approaches:

Author: Bornvalue Chitambira

Supervisor: Christopher Engstr¨om

2. Good Credit and the Good Life: Credit Scores Predict Subjective Well-Being

Joe J. Gladstone

Ashley Whillans

3. THE AWARENESS ABOUT THE CIBIL SCORES AMONG THE VARIOUS CUSTOMERS OF COMMERCIAL BANKS IN CENTRAL KERALA (A STUDY WITH REFERENCE TOGOVT.EMPLOYEES) GREESHMA SAJAN, Assistant Professor, Calicut Adarsha Sanskrit Vidyapeetha, Balussery , Kerala

4. International Journal of Computing, Programming and Database Management 2020; 1(1): 22-25

Gandla Venkatesh Dhamodhar Department of Computer Science, Sri Venkateswara University, Tirupati, Andhra Pradesh, India

5. COVID19 IMPACT ON CIBIL REPORT AND LOAN REPAYMENT CAPACITY OF BORROWERS (Special reference to Loan Moratorium, Loan Restructuring & Loan default)

Rashmi Somani(Medi-Caps Institute of Technology and Management)

Insha Mohammad(Medi-Caps Institute of Technology and Management)

Kapil Kumar Tiwari(Medi-Caps University)

6. Analysis and Prediction of CIBIL Score using Machine Learning

November 2021 Journal of Information and Computational Science 10(10):248-253

Authors:Sunil Dhore(Army Institute of Technology)

7. Machine Learning based Cibil Verification System

1G Elizabeth Rani Assistant Professor, CSE, Kalasalingam Academy of Research and Education Anand Nagar, Krishnankoil, India. g.elizabeth@klu.ac.in

4A.Sai Sri Harsha Student, Department of CSE, Kalasalingam Academy of Research and Education Anand Nagar, Krishnankoil, India. saisriharsha100@gmail.com

2A.Tirumala Vikas Reddy Student, Department of CSE, Kalasalingam Academy of Research and Education Anand Nagar, Krishnankoil, India. vikas.sunny99@gmail.com

5M. Sakthimohan Assistant Professor, ECE, Kalasalingam Academy of Research and Education Anand Nagar, Krishnankoil, India. sakthimohan.phd@gmail.com

8. How Spatial Autocorrelation (Global Moran's I) works

9. Arthur Getis J. K. Ord The Analysis of Spatial Association by Use of Distance Statistics

10. US Drug Overdose: Spatial Autocorrelation, CRAIG CHILVERS