CS 5565

INTRODUCTION TO STATISTICAL LEARNING

US Household income statistics Project Report

Team Members:
Manaswini Vedula(16292800)
Mounika Rachamadugu(16297401)
Pujitha Sakhavarapu(16299783)
Sowmya Reddy Katukuri(16297667)

Description of the Problem: Based on the household income statistics dataset and the predictors provided we calculate the household income as well as taking up the household income and based on the geographic location like latitude and longitude we will observe the data distribution across the united states on a geographical map using the scatter plots and
We have taken the dataset from the following link:
https://www.kaggle.com/goldenoakresearch/us-household-income-stats-geo-locations

Description of the Dataset:

The dataset originally developed for real estate and business investment research. Income is a vital element when determining both quality and socioeconomic features of a given geographic location. The data was derived from over +36,000 files and covers 348,893 location records. The database contains 32,000 records on US Household Income Statistics & Geo Locations. To access, all 348,893 records on a scale roughly equivalent to a neighborhood. The dataset has 32527 rows and 19 columns. Below are the features that the dataset uses to classify the income.

Household & Geographic Statistics:

Mean Household Income (double)
Median Household Income (double)
Standard Deviation of Household Income (double)
Number of Households (double)
Square area of land at location (double)
Square area of water at location (double)

Geographic Location:

Longitude (double)
Latitude (double)
State Name (character)
State abbreviated (character)
State_Code (character)
County Name (character)
City Name (character)
Name of city, town, village or CPD (character)
Primary, Defines if the location is a track and block group.
Zip Code (character)
Area Code (character)

Importing all the required libraries and reading the dataset into a data frame. The data frame contains 1000 rows and 19 columns.

```
[1] from mpl_toolkits.mplot3d import Axes3D
    from sklearn.preprocessing import StandardScaler
    import matplotlib.pyplot as plt # plotting
    import numpy as np # linear algebra
    import os # accessing directory structure
    import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
[2] nRowsRead = 1000 # specify 'None' if want to read whole file
    # kaggle_income.csv may have more rows in reality, but we are only loading/previewing the first 1000 rows
    df1 = pd.read_csv('kaggle_income.csv', delimiter=',', nrows = nRowsRead)
    df1.dataframeName = 'kaggle_income.csv'
    nRow, nCol = df1.shape
    print(f'There are {nRow} rows and {nCol} columns')

    There are 1000 rows and 19 columns
```

Displaying the top 5 rows of the dataset.

```
[3] df1.head(5)
```

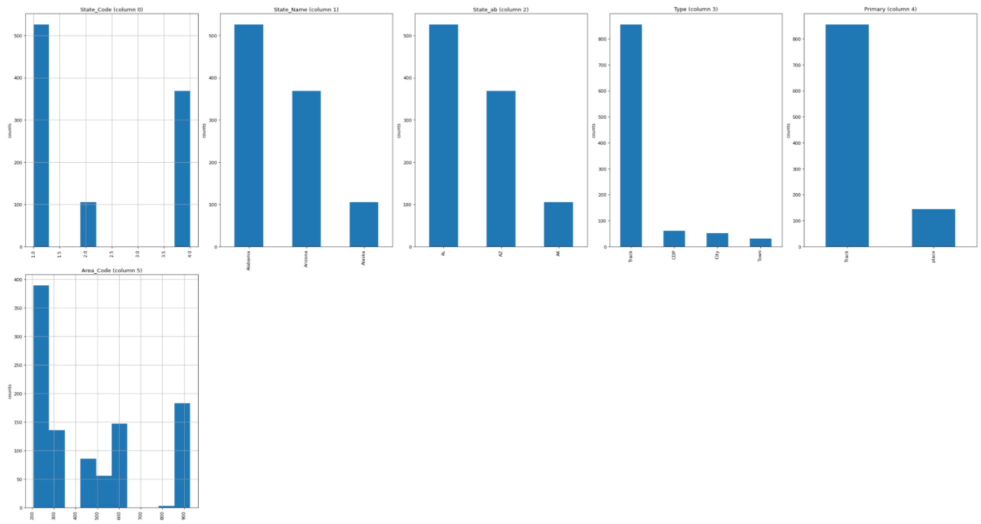
	id	State_Code	State_Name	State_ab	County	City	Place	Type	Primary	Zip_Code	Area_Code	ALand	AWater	Lat
0	1011000	1	Alabama	AL	Mobile County	Chickasaw	Chickasaw city	City	place	36611	251	10894952	909156	30.771450
1	1011010	1	Alabama	AL	Barbour County	Louisville	Clio city	City	place	36048	334	26070325	23254	31.708516
2	1011020	1	Alabama	AL	Shelby County	Columbiana	Columbiana city	City	place	35051	205	44835274	261034	33.191452
3	1011030	1	Alabama	AL	Mobile County	Satsuma	Creola city	City	place	36572	251	36878729	2374530	30.874343
4	1011040	1	Alabama	AL	Mobile County	Dauphin Island	Dauphin Island	Town	place	36528	251	16204185	413605152	30.250913

Plotting per column distribution plots if the column value is numerical then we are plotting the bar graph else we are plotting the histogram plots

```
# Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns that have between 1 and 50 ur
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()
```

Now calling the function and the output plots are displayed as below.

```
[7]  plotPerColumnDistribution(df1, 10, 5)
```

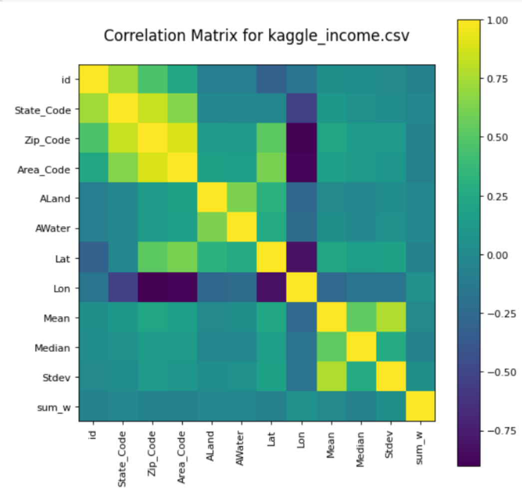


Plotting a correlation matrix dropping the Nan columns

```
[5]  # Correlation matrix
     def plotCorrelationMatrix(df, graphWidth):
         filename = df.dataframeName
         df = df.dropna('columns') # drop columns with NaN
         df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
         if df.shape[1] < 2:
             print(f'No correlation plots shown: The number of non-NaN or constant columns ({df.shape[1]}) is less than 2')
             return
         corr = df.corr()
         plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')
         corrMat = plt.matshow(corr, fignum = 1)
         plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
         plt.yticks(range(len(corr.columns)), corr.columns)
         plt.gca().xaxis.tick_bottom()
         plt.colorbar(corrMat)
         plt.title(f'Correlation Matrix for {filename}', fontsize=15)
         plt.show()
```

Now calling the function and the correlation matrix is as shown below

```
plotCorrelationMatrix(df1, 8)
```

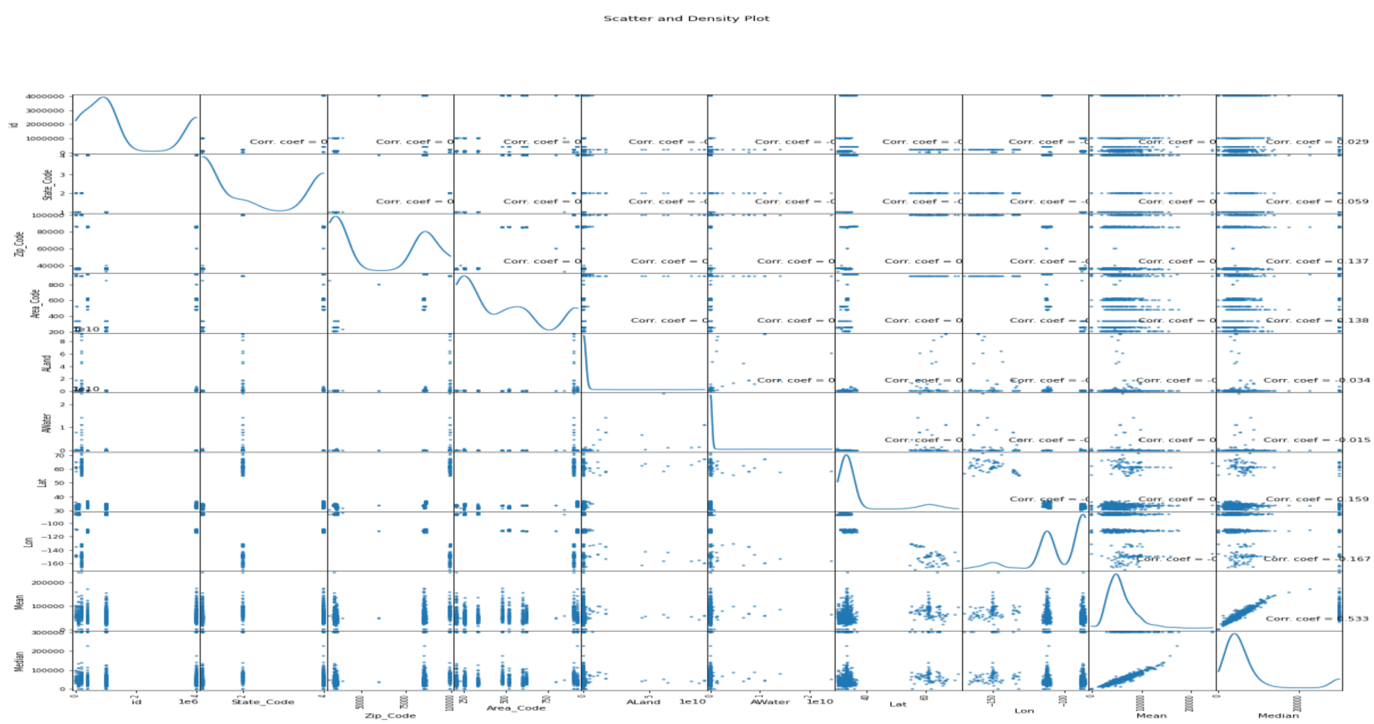


Now plotting the scatter and density plots and the plots are as shown below. These plots show the relation between each column to the other

```
[6] # Scatter and density plots
    def plotScatterMatrix(df, plotSize, textSize):
        df = df.select_dtypes(include =[np.number]) # keep only numerical columns
        # Remove rows and columns that would lead to df being singular
        df = df.dropna('columns')
        df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
        columnNames = list(df)
        if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
            columnNames = columnNames[:10]
        df = df[columnNames]
        ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
        corrs = df.corr().values
        for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
            ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center', va='center', size=textSize)
        plt.suptitle('Scatter and Density Plot')
        plt.show()
```

Plotting the scatter matrix plot

```
plotScatterMatrix(df1, 20, 10)
```



Again, reading the data using a particular encoding and looking at the summary of the data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
```

Loading the data and look at the data fields

```
df_income = pd.read_csv('kaggle_income.csv', encoding='ISO-8859-1')
df_income.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32526 entries, 0 to 32525
Data columns (total 19 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id          32526 non-null  int64
 1   State_Code  32526 non-null  int64
 2   State_Name  32526 non-null  object
 3   State_ab    32526 non-null  object
 4   County      32526 non-null  object
 5   City        32526 non-null  object
 6   Place       32526 non-null  object
 7   Type        32526 non-null  object
 8   Primary     32526 non-null  object
 9   Zip_Code    32526 non-null  int64
 10  Area_Code   32526 non-null  object
 11  ALand       32526 non-null  int64
 12  AWater      32526 non-null  int64
 13  Lat         32526 non-null  float64
 14  Lon         32526 non-null  float64
 15  Mean        32526 non-null  int64
 16  Median      32526 non-null  int64
 17  Stdev       32526 non-null  int64
 18  sum_w       32526 non-null  float64
dtypes: float64(3), int64(8), object(8)
memory usage: 4.7+ MB
```

Looking at statistical values of all the columns.

data column means:

```
In [5]: df_income.describe()
```

Out[5]:

	id	State_Code	Zip_Code	ALand	AWater	Lat	Lon	Mean	Median	Stdev	s
count	3.252600e+04	32526.000000	32526.000000	3.252600e+04	3.252600e+04	32526.000000	32526.000000	32526.000000	32526.000000	32526.000000	:
mean	6.203707e+07	28.624885	50182.648404	1.165893e+08	6.952054e+06	37.731983	-91.303844	66703.986042	85452.938818	47273.695321	!
std	1.115546e+08	16.297205	29410.122808	1.280894e+09	2.092093e+08	5.579450	16.227588	30451.194599	87810.895132	16555.486882	:
min	1.026000e+03	1.000000	601.000000	0.000000e+00	0.000000e+00	17.929085	-175.860041	0.000000	0.000000	0.000000	(
25%	8.021282e+06	13.000000	26362.000000	1.906991e+06	0.000000e+00	34.013469	-97.664034	46015.500000	36046.250000	36075.000000	:
50%	2.901168e+07	29.000000	48163.000000	5.022976e+06	2.703350e+04	38.925588	-87.139280	60738.000000	51874.500000	46179.000000	:
75%	4.802899e+07	42.000000	76712.000000	3.090984e+07	5.082078e+05	41.495793	-79.852969	82223.500000	80915.000000	58078.000000	!
max	4.802211e+08	72.000000	99950.000000	9.163267e+10	2.453228e+10	71.253500	-65.500823	242857.000000	300000.000000	113936.000000	(

Only taking the data with column sum_w>50000

Cleaning the data

```
df_income[df_income['sum_w']>50000]
```

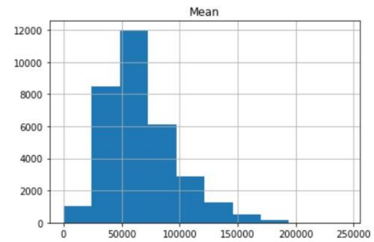
te_ab	County	City	Place	Type	Primary	Zip_Code	Area_Code	ALand	AWater	Lat	Lon	Mean	Median	Stdev	sum_
	Mobile County	Mobile	Mobile city	City	place	36606	251	361044263	105325210	30.668426	-88.100226	53330	38231	50855	51243
	Kern County	Bakersfield	Bakersfield city	City	place	93304	661	385366784	3809676	35.321213	-119.018291	71025	56986	57439	62475
	Sacramento County	Sacramento	Sacramento city	City	place	95816	916	253621861	5651667	38.566592	-121.468632	64615	49743	55451	10603
	District of	Washington	Washington city	City	place	20001	202	158364992	18633403	38.904103	-77.017229	92477	71044	76702	12936
	Cook County	Chicago	Chicago city	City	place	60608	773	588808396	17615206	41.837551	-87.681844	67450	48321	62043	61224
	East Baton Rouge Parish	Baton Rouge	Baton Rouge city	City	place	70806	225	222547923	5682680	30.442174	-91.130894	56178	39487	54542	57977
	Wayne County	Detroit	Detroit city	City	place	48206	313	359360867	10667144	42.383037	-83.102237	36811	25417	37336	19279
	Lubbock County	Lubbock	Lubbock city	City	place	79410	806	322793416	2955879	33.565636	-101.886671	58705	44459	52375	57937
	Richmond	Richmond	Richmond city	City	place	23220	804	154942138	6879776	37.531399	-77.476009	57579	40384	56721	56861

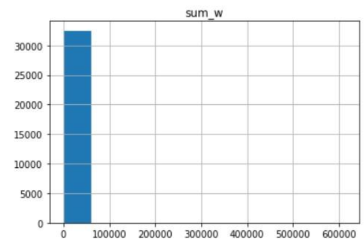
Plotting histograms for the Mean and sum_w to remove the outliers and data of less frequency.

Draw histogram for Mean and sum_w columns:

```
df_income.hist(column='Mean')
df_income.hist(column='sum_w')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000169B4C85448>]],
      dtype=object)
```



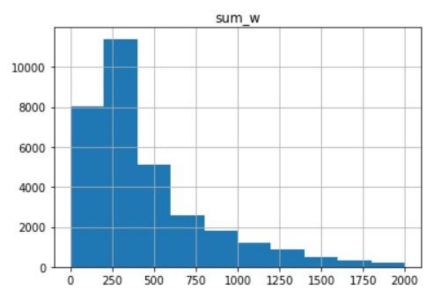


As we have seen that most data is below 2000 so we are taking the rows with sum_w values less than 2000 and plotting them to observe their distribution.

As we found there are very few sum_w greater than 2000 so we remove all the rows with huge sum_w value

```
df_income = df_income[df_income['sum_w'] < 2000]
df_income.hist(column='sum_w')
```

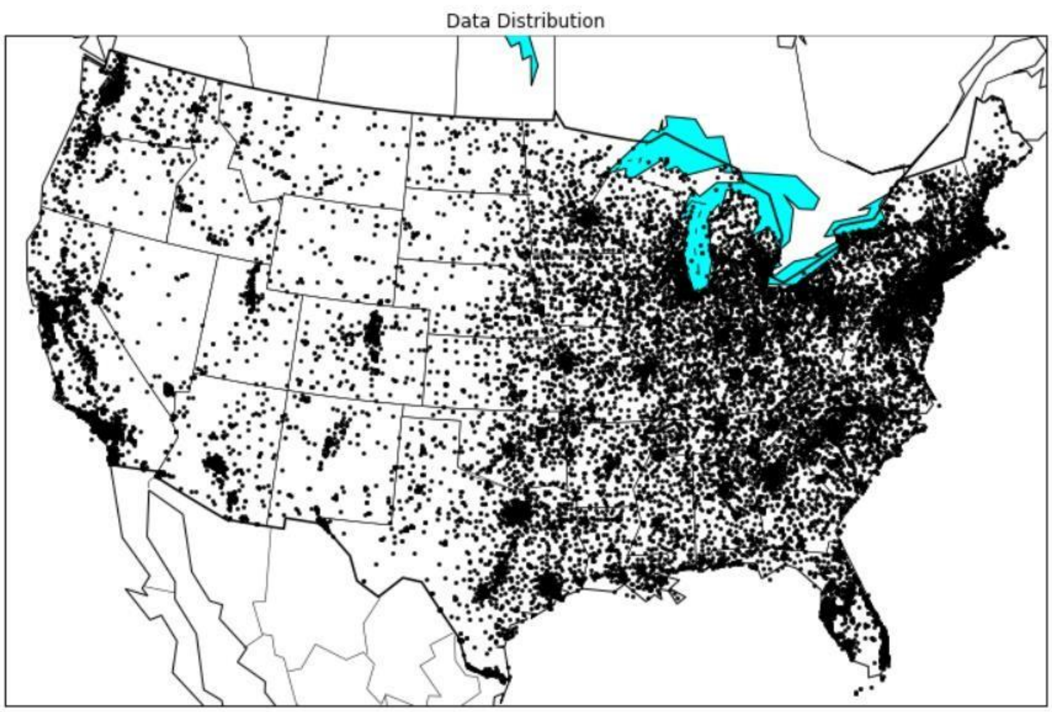
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000169AF41EB08>]],
      dtype=object)
```



Now Using the Lat Long values in the data set plotting the values to find out the data distribution across the US map.

```
os.environ['PROJ_LIB'] = r'C:\Users\bharg\anaconda3\pkgs\proj4-5.2.0-h6538335_1006\Library\share'
from mpl_toolkits.basemap import Basemap
```

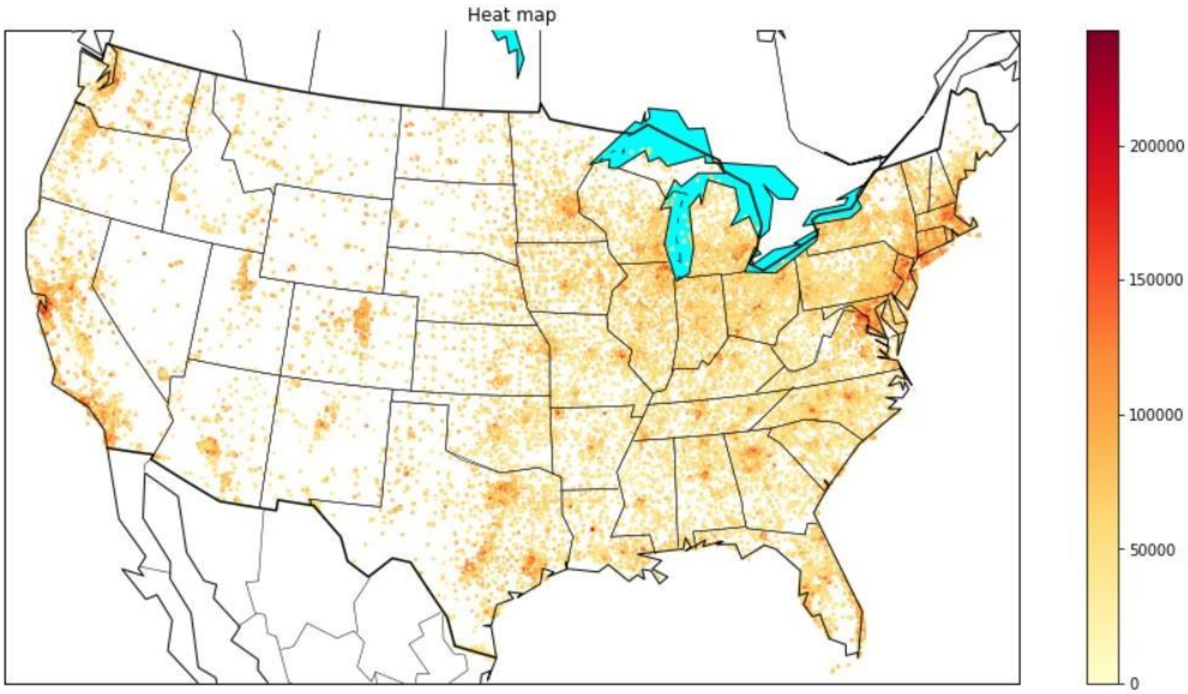
```
fig1 = plt.figure(figsize=(16,8))

m = Basemap(
    llcrnrlon=-119,
    llcrnrlat=22,
    urcrnrlon=-64,
    urcrnrlat=49,
    projection='lcc',
    lat_1=33,
    lat_2=45,
    lon_0=-95
)
m.drawmapboundary(zorder=0)
m.fillcontinents(color='#ffffff', lake_color='aqua', zorder=1)
m.drawcountries(linewidth=1.5)
m.drawcoastlines()
m.drawstates()

x, y = m(
    np.array(df_income['Lon']),
    np.array(df_income['Lat'])
)
# load data into map
ax = fig1.add_subplot(111)
ax.scatter(
    x,
    y,
    3,
    marker='o',
    color='k',
    zorder=1.5
)
# draw dots on map
plt.title('Data Distribution')
```



Now Using the sum_w values in the data set plotting the values to know the density of distribution using heat map across the US map.

```
fig2 = plt.figure(figsize=(16,8))

m = Basemap(llcrnrlon=-119, llcrnrlat=22, urcrnrlon=-64, urcrnrlat=49, projection='lcc', lat_1=33, lat_2=45, lon_0=-95)
m.drawmapboundary(zorder=0)
m.fillcontinents(color='#ffffff', lake_color='aqua', zorder=1)
m.drawcountries(linewidth=1.5)
m.drawcoastlines()
m.drawstates()

#normalize sum_w
df_income['sum_w_n'] = 20 * (1 + (df_income['sum_w'] - df_income['sum_w'].mean()) / df_income['sum_w'].std())

ax = fig2.add_subplot(111)
scatter1 = ax.scatter(
    x,
    y,
    3,
#     s=np.array(df_income['sum_w_n']), # dot size
    c=np.array(df_income['Mean']), # You can change here to ALand or AWater
    cmap=plt.cm.YlOrRd,
    marker='o',
    zorder=1.5
) # draw dots on map

plt.colorbar(scatter1)
plt.title('Heat map')
```

Cleani



Cleaning and splitting the data set like drop the categorical columns and fill the empty values with zeros.

Cleaning the data and Spliting the data

Cleaning the data and then we are spliting the data into training data and test data:

```
from sklearn.model_selection import train_test_split

# Convert string to NaN
df_income['Area_Code_Num'] = pd.to_numeric(df_income['Area_Code'], errors='coerce')
# Convert NaN to 0
df_income['Area_Code_Num'].fillna(0, inplace=True)
# Convert string to number
dummies_Type = pd.get_dummies(df_income['Type'], prefix= 'Type')
dummies_Primary = pd.get_dummies(df_income['Primary'], prefix= 'Primary')
# Add number columns
df_income_new = pd.concat([df_income, dummies_Type, dummies_Primary], axis=1)
# Drop string columns
df_income_new.drop(['id', 'State_Name', 'State_ab', 'County', 'City', 'Place', 'Type', 'Primary', 'Median', 'Stdev', 'Area_Code', 'Mea
n'], axis=1, inplace=True)
# Split data into training data and cross validation data
X = df_income_new
y = df_income[['Mean']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=3)
X_train.head(10)
```

	State_Code	Zip_Code	ALand	AWater	Lat	Lon	sum_w	sum_w_n	Area_Code_Num	Type_Borough	Type_CDP	Type_City
4324	6	95843	1905258	0	38.721477	-121.372725	1383.759579	71.288422	916.0	0	0	0
4961	8	80120	1852818	1620	39.596541	-104.996668	754.663244	36.902432	303.0	0	0	0
17873	34	7652	4521063	7133	40.940233	-74.054245	986.407272	49.569407	201.0	0	0	0
20030	36	14590	183852269	219965	43.171268	-76.832549	263.424794	10.051663	315.0	0	0	0
25234	42	15902	874032	36914	40.296098	-78.912390	462.085718	20.910337	814.0	0	0	0
1713	6	93292	1647669	0	36.485694	-119.224771	98.198801	1.020519	559.0	0	1	0
1297	5	71941	1693718	0	34.234145	-92.918641	77.430112	-0.114684	501.0	0	0	0
813	4	85345	2569200	0	33.588530	-112.228887	1607.183433	83.500622	623.0	0	0	0

Normalizing values of training and testing before applying them to a model.

Linear Regression Model

Before running linear regression, we have to normalize data.

```
from sklearn_pandas import DataFrameMapper
from sklearn.preprocessing import StandardScaler

mapper = DataFrameMapper([(X_train.columns, StandardScaler())])
scaled_features = mapper.fit_transform(X_train.copy())
X_train_scaled = pd.DataFrame(scaled_features, index=X_train.index, columns=X_train.columns)
scaled_features_test = mapper.fit_transform(X_test.copy())
X_test_scaled = pd.DataFrame(scaled_features_test, index=X_test.index, columns=X_test.columns)
X_train_scaled.head(10)
```

	State_Code	Zip_Code	ALand	AWater	Lat	Lon	sum_w	sum_w_n	Area_Code_Num	Type_Borough	Type_CDP	Type_City	Type_
4324	-1.393014	1.556447	-0.093160	-0.032725	0.174187	-1.856751	2.564539	2.564539	1.392686	-0.060799	-0.170924	-0.160521	-0.025
4961	-1.270040	1.021685	-0.093203	-0.032717	0.331065	-0.847246	0.844670	0.844670	-1.246068	-0.060799	-0.170924	-0.160521	-0.025
17873	0.328626	-1.443057	-0.091055	-0.032691	0.571957	1.060206	1.478229	1.478229	-1.685143	-0.060799	-0.170924	-0.160521	-0.025
20030	0.451600	-1.207086	0.053287	-0.031686	0.971929	0.888937	-0.498313	-0.498313	-1.194413	-0.060799	-0.170924	-0.160521	-0.025
25234	0.820523	-1.162463	-0.093990	-0.032550	0.456479	0.760725	0.044801	0.044801	0.953611	-0.060799	-0.170924	-0.160521	-0.025
1713	-1.393014	1.469684	-0.093368	-0.032725	-0.226636	-1.724340	-0.950020	-0.950020	-0.144076	-0.060799	5.850546	-0.160521	-0.025
1297	-1.454501	0.743505	-0.093331	-0.032725	-0.630286	-0.102694	-1.006799	-1.006799	-0.393747	-0.060799	-0.170924	-0.160521	-0.025
813	-1.515988	1.199395	-0.092626	-0.032725	-0.746029	-1.293078	3.175351	3.175351	0.131422	-0.060799	-0.170924	-0.160521	-0.025
3130	-1.393014	1.556447	-0.093107	-0.032725	0.171684	-1.856601	-0.165430	-0.165430	1.392686	-0.060799	-0.170924	-0.160521	-0.025
26190	1.004984	-0.706539	-0.087384	-0.032249	-0.504653	0.578739	0.211881	0.211881	1.168844	-0.060799	-0.170924	-0.160521	-0.025

Applying the Linear regression model on the dataset and calculated the Variance, MSE and r^2 values.

Linear regression is a Linear approach of modelling the relationship between a dependent variable and one or more explanatory variables.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import explained_variance_score, mean_squared_error, r2_score
from sklearn.metrics import accuracy_score

model = LinearRegression()
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

print('Explained variance score: %.2f' % explained_variance_score(y_test, y_pred))
print('Mean squared error: %.2f' % mean_squared_error(y_test, y_pred))
print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

```
Explained variance score: -474365370888521252864.00
Mean squared error: 44568914639895656238243066675 2.00
Variance score: -474365370888521252864.00
```

The result is not so good, Linear regression is not suit for this dataset.

Applying Random forest regressor for the data set.

Random forest fits several classifying decision trees on various sub samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting.

Random Forest Regressor Model

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import explained_variance_score, mean_squared_error, r2_score

model = RandomForestRegressor(random_state=0, n_jobs=-1)
model.fit(X_train, y_train.values.ravel())
y_pred = model.predict(X_test)

print('Explained variance score: %.2f' % explained_variance_score(y_test, y_pred))
print('Mean squared error: %.2f' % mean_squared_error(y_test, y_pred))
print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

```
Explained variance score: 0.52
Mean squared error: 452790907.35
Variance score: 0.52
```

We got a score of 0.52. Seems much better now.

Applying the KNN model on the dataset.

It is a supervised machine learning algorithm and is easy to implement that can be used to solve both classification and regression problems.

K Nearest Neighbors

```
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor(n_neighbors=18)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print('Explained variance score: %.2f' % explained_variance_score(y_test, y_pred))
print('Mean squared error: %.2f' % mean_squared_error(y_test, y_pred))
print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

```
Explained variance score: 0.05
Mean squared error: 894799775.82
Variance score: 0.05
```

I tried the nearest neighbour algorithm, but still did not get a better result.

Applying the Naïve Bayes algorithms using Gaussian kernel.

It is a classification technique based on Bayes Theorem with an assumption of independence among the predictors. In simple terms, a Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Naive Bayes algorithm

```
from sklearn.naive_bayes import GaussianNB

#Applying implement Naive Bayes algorithm
gnb_model = GaussianNB()
gnb_model.fit(X_train, y_train)
y_predict = gnb_model.predict(X_test)
print('Explained variance score: %.2f' % explained_variance_score(y_test, y_predict))
print('Mean squared error: %.2f' % mean_squared_error(y_test, y_predict))
print('Variance score: %.2f' % r2_score(y_test, y_predict))
```

```
C:\Users\bharg\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
Explained variance score: -0.63
Mean squared error: 1529959829.78
Variance score: -0.63
```

Applying the SVM model on the dataset.

SVM is a supervised machine learning algorithm, which can be used for classification or regression problems. SVM find the optimal boundary between the possible inputs.

```
[ ] from sklearn import svm

    #Create a svm Classifier
    mode = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    model.fit(X_train, y_train)

    #Predict the response for test dataset
    y_pred = model.predict(X_test)
    print('Explained variance score: %2f' % explained_variance_score(y_test,y_pred))
    print('Mean squared error: %2f' % mean_squared_error(y_test,y_pred))
    print('Variance score: %2f' % r2_score(y_test,y_pred))
```

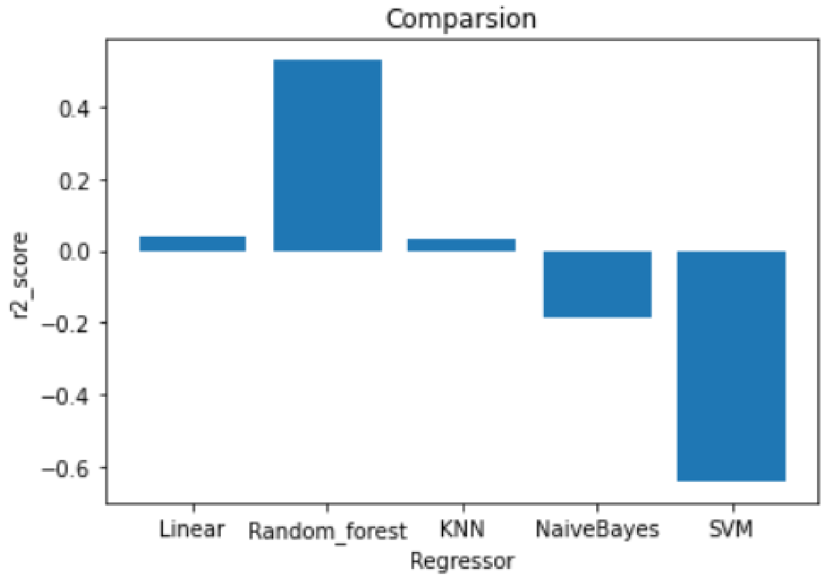
```
/usr/local/lib/python3.6/dist-packages/sklearn/naive_bayes.py:206: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_
  y = column_or_1d(y, warn=True)
Explained variance score: -0.642465
Mean squared error: 1524120048.178912
Variance score: -0.642469
```

Comparing the results:

Variance, MSE and R^2 values are calculated for each model. Compared the variance r2_score for all the models.

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

= np.array([r2_score(y_test,y_pred_linear),r2_score(y_test,y_pred_rf),r2_score(y_test,y_pred_knn),r2_score(y_test,y_pred_nb),r2_score(y_test,y_p
=['Linear', 'Random_forest', 'KNN','NaiveBayes','SVM']
lt.bar(x,y)
lt.title('Comparsion')
lt.xlabel('Regressor')
lt.ylabel('r2_score')
```

```
Text(0, 0.5, 'r2_score')
```



Conclusion:

We have applied regression as well as classification techniques for predicting the mean household income from the US income household dataset. From the above observation it is evident that Random forest regressor outperforms all the other applied models in this data set. Whereas SVM has high negative impact on the variable's correlation.

GitHub link: Click here for GitHub link for Code.