An-Najah National University

Faculty of Engineering and Information Technology Computer Engineering Department

Machine Learning Project

Afeef Moghaiar

Sameh Salama

Raghad Abo-Arqoub

Dr.Anas Toma

December ,2021

Contents

1 Introduction

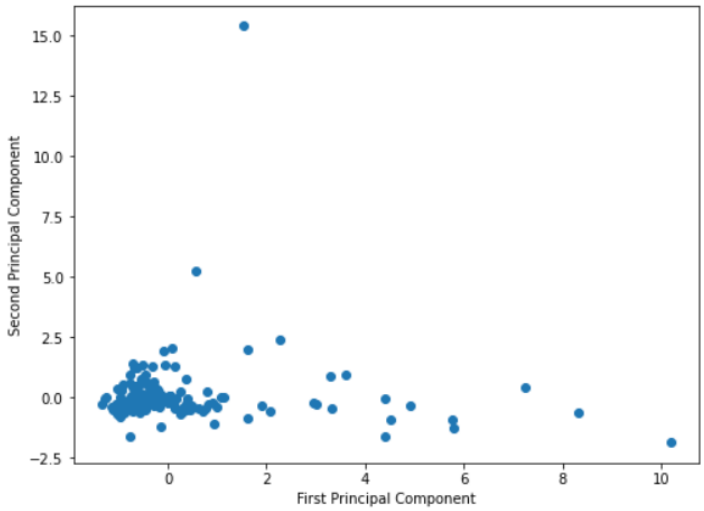
2 Part one

2.1 Data visualization and analysis:

We will use for the data visualization Principle Component Analysis (PCA) is a fast and flexible unsupervised method for dimensionality reduction in data, Its behavior is easiest to visualize by looking at a two-dimensional data set.
The Code :

```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
scalar.fit(df)
scaled_data = scalar.transform(df)
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
# fitting
pca.fit(scaled_data)
#to compute the mean and standard deviation on a training set so as to be able to later re-apply the same transformation on the testing set.
x_pca = pca.transform(scaled_data)
x_pca.shape
from sklearn.decomposition import PCA
# giving a larger plot
plt.figure(figsize =(8, 6))
plt.scatter(x_pca[:, 0], x_pca[:, 1])
# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

The Output of the code :



By eye, it is clear that there is NO linear relationship between the x and y variables.K-Nearest Neighbor is more suitable for this data because the points close to each other. So we don't use linear regression.
Before using the model we want to see the correlation between features and the target(CORONA__Ca) using correlation matrix.

```
X =df[["Population","PopDensity","AgingRatio","ServicesHi","HealthServ","Landuse","Commercial","RoadDensit","GreenAreas","Open_spave","CORONA__Ca"]]
new = pd.DataFrame(X)
cor_matrix = new.corr().abs()
print(cor_matrix)
```

```
            Population  PopDensity  ...  Open_spave  CORONA__Ca
Population    1.000000    0.023690  ...    0.024566    0.449980
PopDensity    0.023690    1.000000  ...    0.013193    0.019244
AgingRatio    0.062759    0.007668  ...    0.002115    0.023188
ServicesHi    0.463569    0.031345  ...    0.061326    0.570953
HealthServ    0.535200    0.035081  ...    0.043087    0.653498
Landuse       0.009145    0.219817  ...    0.066678    0.045853
Commercial    0.062794    0.116781  ...    0.033578    0.090260
RoadDensit    0.082177    0.011663  ...    0.010977    0.112087
GreenAreas    0.042760    0.589041  ...    0.010678    0.163252
Open_spave    0.024566    0.013193  ...    1.000000    0.007730
CORONA__Ca    0.449980    0.019244  ...    0.007730    1.000000

[11 rows x 11 columns]
```

From correlation matrix the (Population,ServicesHi,HealthServ) more correlation with the target and apply the model . using KNN (with k=7) for regression and split the data into training and testing, with test size = 0.2 from data the accuracy become 47.68 .

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
import numpy as np
X = df[["Population","ServicesHi","HealthServ"]]
y = df['CORONA__Ca']
from sklearn.model_selection import train_test_split
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
knn = KNeighborsRegressor(n_neighbors=7)
knn.fit(X_train, y_train)
# Calculate the accuracy of the model
print('R squared: {:.2f}'.format(knn.score(X_test, y_test)*100))
```

```
R squared: 47.86
```

3 Part 2

The dataset may suffer from noisy, missing, inconsistent or replicated data. It may also suffer from redundant or irrelevant features we want to handle these problem.

3.1 Data cleaning: Handle missing and noisy data

3.1.1 Missing data

In this Part we want to handle missing data . at first we convert all zeros to nan value and if we have more then 6 variable on the tuple nan drop the whole tuple .but there is no missing data.

```
# Marking missing values with nan values
from numpy import nan
df[["PopDensity","AgingRatio","ServicesHi","HealthServ","Commercial","RoadDensit","GreenAreas","CORONA__Ca"]]=df[["PopDensity","AgingRatio","ServicesHi","Health
# count the number of nan values in each column
#print(df.isnull().sum())
print (df)
df.dropna(axis=0,thresh=6,inplace=True)
print(len(df))
```

```
     CORONA__Ca  Population  PopDensity  ...  RoadDensit  GreenAreas  Open_spave
0        7299.0       15988       5.920  ...     0.04670    0.080100      0.8651
1         139.0        3930       4.200  ...     0.16400         NaN      0.8800
2          63.0        4966       3.000  ...     0.03200         NaN      0.9100
3         181.0       18534       3.700  ...     0.10506    0.002709      0.9201
4         418.0       18202       2.000  ...     0.07800    0.162000      0.8940
..          ...         ...         ...  ...         ...         ...         ...
186        22.0        4572       5.000  ...         NaN         NaN      0.8300
187        92.0       10049       6.500  ...     0.09500         NaN      0.8300
188        55.0       25773       5.400  ...     0.13600    0.001200      0.9600
189      3381.0       30233       5.400  ...     0.18000    0.020000      0.9000
190       317.0       14258       6.388  ...     0.48000         NaN      0.7100

[191 rows x 11 columns]
191
```

after that we return nan to zeros.

```
[ ] df[["PopDensity","AgingRatio","ServicesHi","HealthServ","Commercial","RoadDensit","GreenAreas","CORONA__Ca"]]=df[["PopDensity","AgingRatio","ServicesHi","Healt
    print(df)
```

```
     CORONA__Ca  Population  PopDensity  ...  RoadDensit  GreenAreas  Open_spave
0        7299.0       15988       5.920  ...     0.04670    0.080100      0.8651
1         139.0        3930       4.200  ...     0.16400    0.000000      0.8800
2          63.0        4966       3.000  ...     0.03200    0.000000      0.9100
3         181.0       18534       3.700  ...     0.10506    0.002709      0.9201
4         418.0       18202       2.000  ...     0.07800    0.162000      0.8940
..          ...         ...         ...  ...         ...         ...         ...
186        22.0        4572       5.000  ...     0.00000    0.000000      0.8300
187        92.0       10049       6.500  ...     0.09500    0.000000      0.8300
188        55.0       25773       5.400  ...     0.13600    0.001200      0.9600
189      3381.0       30233       5.400  ...     0.18000    0.020000      0.9000
190       317.0       14258       6.388  ...     0.48000    0.000000      0.7100

[191 rows x 11 columns]
```

But in feature (ServicesHi) the category from 1 to 5 but we have zeros , at first we drop it and find the accuracy is low, so we replace the zeros with middle value (3) and the accuracy remain the same but the correlation between the feature (ServicesHi)and target(CORONA__Ca) increase .

```
X =df[["Population","PopDensity","AgingRatio","ServicesHi","HealthServ","Landuse","Commercial","RoadDensit","GreenAreas","Open_spave","CORONA__Ca"]]
y =df['CORONA__Ca']
new = pd.DataFrame(X)
cor_matrix = new.corr().abs()
print(cor_matrix)
```

```
             Population  PopDensity  ...  Open_spave  CORONA__Ca
Population     1.000000    0.023690  ...    0.024566    0.449980
PopDensity     0.023690    1.000000  ...    0.013193    0.019244
AgingRatio     0.062759    0.007668  ...    0.002115    0.023188
ServicesHi     0.456944    0.036869  ...    0.052135    0.586829
HealthServ     0.535200    0.035081  ...    0.043087    0.653498
Landuse        0.009145    0.219817  ...    0.066678    0.045853
Commercial     0.062794    0.116781  ...    0.033578    0.090260
RoadDensit     0.082177    0.011663  ...    0.010977    0.112087
GreenAreas     0.042760    0.589041  ...    0.010678    0.163252
Open_spave     0.024566    0.013193  ...    1.000000    0.007730
CORONA__Ca     0.449980    0.019244  ...    0.007730    1.000000

[11 rows x 11 columns]
```

```
[ ] from sklearn.linear_model import LinearRegression
    from sklearn.neighbors import KNeighborsRegressor
    import numpy as np
    X = df[["Population","ServicesHi","HealthServ"]]
    y = df['CORONA__Ca']
    from sklearn.model_selection import train_test_split
    # Split into training and test set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
    knn = KNeighborsRegressor(n_neighbors=7)
    knn.fit(X_train, y_train)
    # Calculate the accuracy of the model
    print('R squared: {:.2f}'.format(knn.score(X_test, y_test)*100))
```

```
R squared: 47.86
```

3.1.2 Noisy data

We detect the noisy data by box plot for each feature .

```
from matplotlib import pyplot as plt
# Box Plot to detect outliers
def boxplot(df,ft):
    df.boxplot(column=ft)
    plt.grid(False)
    plt.show()
boxplot(df,'CORONA_Ca')
boxplot(df,'Population')
boxplot(df,'PopDensity')
boxplot(df,'AgingRatio')
boxplot(df,'ServicesHi')
boxplot(df,'HealthServ')
boxplot(df,'Landuse')
boxplot(df,'Commercial')
boxplot(df,'RoadDensit')
boxplot(df,'GreenAreas')
boxplot(df,'Open_spave')
```

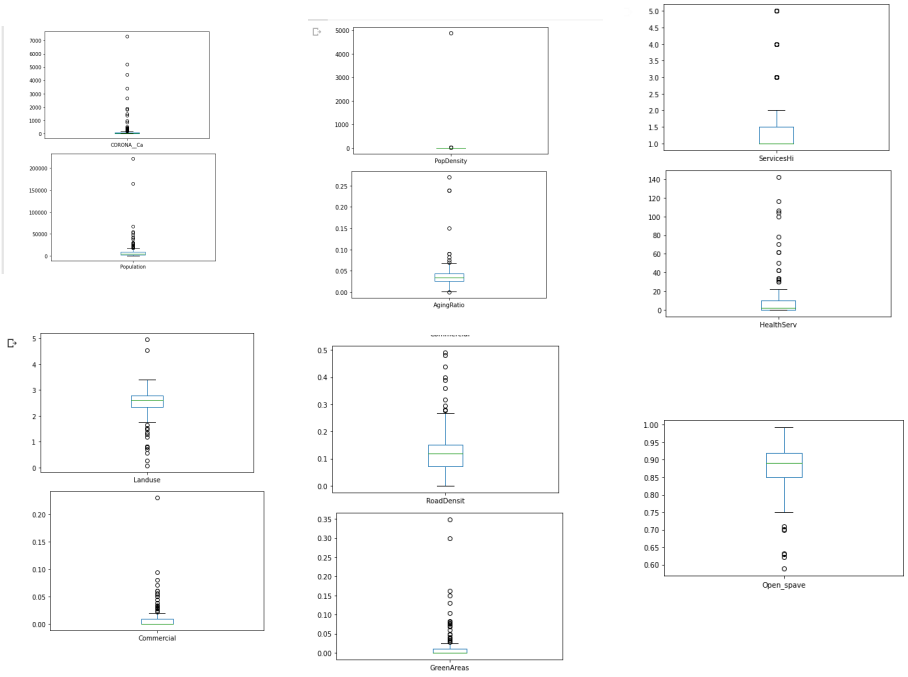


Figure 1: Box Plot for each feature to detect outlier

At first we sort the data ,then calculate the first ,second and third quartile for each feature, and we calculate the max and min(low_lim = Q1 - 1.5 * IQR, up_lim = Q3 + 1.5 * IQR) then if any data above the max then the outlier=up_lim,if the data below the low_lim then the outlier =low_lim. Then we plot the box plot.

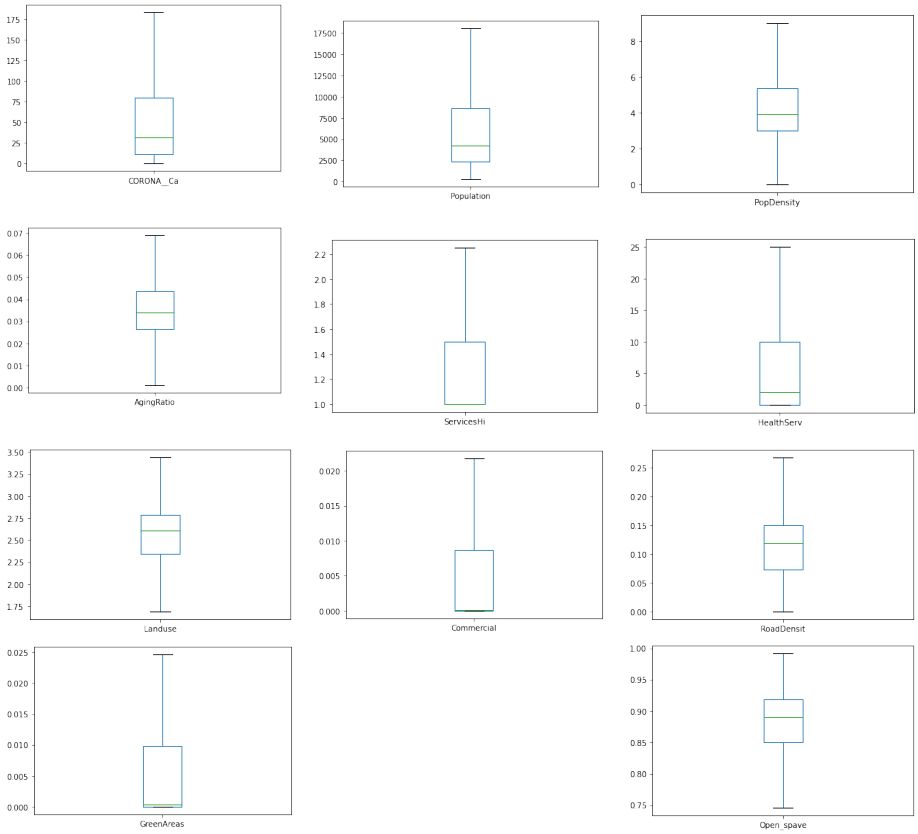


Figure 2: Box Plot for each feature after detect outlier

And we calculate the accuracy and it improves.

```
[ ] from sklearn.linear_model import LinearRegression
    from sklearn.neighbors import KNeighborsRegressor

    import numpy as np
    X = df[["Population","ServicesHi","HealthServ"]]
    y = df['CORONA__Ca']

    from sklearn.model_selection import train_test_split

    # Split into training and test set
    X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size = 0.2, random_state=42)
    knn = KNeighborsRegressor(n_neighbors=7)
    knn.fit(X_train, y_train)
    # Calculate the accuracy of the model
    print('R squared: {:.2f}'.format(knn.score(X_test, y_test)*100))
```

```
R squared: 78.47
```

3.2 Data Integration: Handle redundant data

In this data set there is no redundant tuples because after applying the function number of tuples remain the same.

```
[ ] df.duplicated
    df.drop_duplicates(keep="last")
```

	CORONA__Ca	Population	PopDensity	AgingRatio	ServicesHi	HealthServ	Landuse	Commercial	RoadDensit	GreenAreas	Open_spave
0	183.5	15988	5.920	0.02000	2.25	25.0	2.78610	0.001750	0.04670	0.024625	0.8651
1	139.0	3930	4.200	0.04700	1.00	8.0	2.33000	0.004900	0.16400	0.000000	0.8800
2	63.0	4966	3.000	0.05000	1.00	12.0	2.82000	0.000000	0.03200	0.000000	0.9100
3	181.0	18060	3.700	0.02579	1.00	25.0	2.69850	0.002074	0.10506	0.002709	0.9201
4	183.5	18060	2.000	0.03300	2.00	25.0	2.37000	0.003000	0.07800	0.024625	0.8940
...	...	...	...	...	...	...	...	...	...	...	...
186	22.0	4572	5.000	0.03000	1.00	10.0	3.00000	0.000000	0.00000	0.000000	0.8300
187	92.0	10049	6.500	0.02600	1.00	8.0	3.43825	0.000000	0.09500	0.000000	0.8300
188	55.0	18060	5.400	0.03400	2.25	12.0	2.40000	0.003400	0.13600	0.001200	0.9600
189	183.5	18060	5.400	0.06000	2.25	25.0	2.82000	0.021750	0.18000	0.020000	0.9000
190	183.5	14258	6.388	0.06900	1.00	20.0	1.68905	0.000000	0.26750	0.000000	0.7459

191 rows × 11 columns

3.3 Data reduction and feature selection: Eliminate the irrelevant features.

At first we want to recompute the correlation matrix between feature and the target ,then we put threshold (0.4) any value below this value it redundant feature and drop it from table .

```
X =df[["Population","PopDensity","AgingRatio","ServicesHi","HealthServ","Landuse","Commercial","RoadDensit","GreenAreas","Open_spave","CORONA__Ca"]]
new = pd.DataFrame(X)
cor_matrix = new.corr().abs()
print(cor_matrix)
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).astype(np.bool))
print(upper_tri['CORONA__Ca'])
to_drop=[]
R=[]
# Searching in whole column
for i in range(len(upper_tri['CORONA__Ca'])):
    if upper_tri['CORONA__Ca'][i] < 0.4 :

        to_drop.append(cor_matrix.columns[i])
print(to_drop)
# to_drop = [column for column in upper_tri.columns["CORONA__Ca"] if any(upper_tri[] < 0.4)]
df=new.drop(to_drop, axis=1)
df
```

```
             Population  PopDensity  ...  Open_spave  CORONA__Ca
Population     1.000000    0.228803  ...    0.055571    0.836053
PopDensity     0.228803    1.000000  ...    0.515227    0.158386
AgingRatio     0.169690    0.223604  ...    0.016911    0.052981
ServicesHi     0.565362    0.049586  ...    0.048425    0.493669
HealthServ     0.667848    0.041458  ...    0.062703    0.627877
Landuse        0.018993    0.057680  ...    0.108308    0.024859
Commercial     0.190751    0.055762  ...    0.040767    0.181240
RoadDensit     0.092219    0.030167  ...    0.001564    0.120320
GreenAreas     0.257779    0.044072  ...    0.037248    0.284953
Open_spave     0.055571    0.515227  ...    1.000000    0.075929
CORONA__Ca     0.836053    0.158386  ...    0.075929    1.000000
```

Then we print the feature will droped and print new dataset.

```
['PopDensity', 'AgingRatio', 'Landuse', 'Commercial', 'RoadDensit', 'GreenAreas', 'Open_spave']
```

	Population	ServicesHi	HealthServ	CORONA__Ca
0	15988	2.25	25.0	183.5
1	3930	1.00	8.0	139.0
2	4966	1.00	12.0	63.0
3	18060	1.00	25.0	181.0
4	18060	2.00	25.0	183.5
...	...	...	...	...
186	4572	1.00	10.0	22.0
187	10049	1.00	8.0	92.0
188	18060	2.25	12.0	55.0
189	18060	2.25	25.0	183.5
190	14258	1.00	20.0	183.5

191 rows × 4 columns

3.4 . Data transformation and discretization

3.4.1 Normalize the features

We normalize the input features using min max,The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range.

```
from sklearn import preprocessing
import numpy as np
X =df[["Population","ServicesHi","HealthServ"]]
# copy the data
df_min_max_scaled = X
# apply normalization techniques
for column in df_min_max_scaled.columns:
    df_min_max_scaled[column] = (df_min_max_scaled[column] - df_min_max_scaled[column].min()) / (df_min_max_scaled[column].max() - df_min_max_scaled[column].min())
# view normalized data
df[["Population","ServicesHi","HealthServ"]]=df_min_max_scaled
df
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  # Remove the CWD from sys.path while we load stuff.
```

	Population	ServicesHi	HealthServ	CORONA__Ca
0	0.883300	1.0	1.00	183.5
1	0.204168	0.0	0.32	139.0
2	0.262518	0.0	0.48	63.0
3	1.000000	0.0	1.00	181.0
4	1.000000	0.8	1.00	183.5
...	...	...	...	...
186	0.240327	0.0	0.40	22.0
187	0.548803	0.0	0.32	92.0
188	1.000000	1.0	0.48	55.0

3.4.2 Discretize the output numerical value and then build a classification model

We want to build classification model , at first we calculate the min and max value , after that we split the data into three categories (low,medium,high) using bins [-1,75,120,184].
we apply KNN classifier with K=7.

```
print(df['CORONA__Ca'].max())
print(df['CORONA__Ca'].min())
df['Class']=pd.cut(x = df['CORONA__Ca'],
                           bins = [-1,75,120,184],
                           labels = ['Low','Mediam','High'])
```

The final accuracy after data preprocessing.

```
from sklearn.neighbors import KNeighborsClassifier

import numpy as np
X = df[["Population","ServicesHi","HealthServ"]]
y = df['Class']

from sklearn.model_selection import train_test_split

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
# Calculate the accuracy of the model
print('R squared: {:.2f}'.format(knn.score(X_test, y_test)*100))
```

```
R squared: 89.74
```