Week 5: Project

This notebook presents the final project of the author in the **Design Thinking and Predictive Analytics for Data Products** course offered by Coursera.

The dataset analysed in this workbook was obtained from another Coursera course that the author is enrolled in, namely **Data Management and Visualization** by Wesleyan University (https://www.coursera.org/learn/data-visualization/supplement/F0UbG/course-data-sets (https://www.coursera.org/learn/data-visualization/supplement/F0UbG/course-data-sets)).

If the reader is interested, more information is available at https://www.gapminder.org/ (https://www.gapminder.org/).

GapMinder Dataset Description

GapMinder is a non-profit venture founded in Stockholm by *Ola Rosling*, *Anna Rosling Rönnlund* and *Hans Rosling*, which aims to **promote sustainable global development and achievement** of the *United Nations Millennium Development Goals*. It seeks to increase the use and understanding of statistics about social, economic, and environmental development at local, national, and global levels.

Since its conception in 2005, Gapminder has grown to include over 200 indicators, including gross domestic product, total employment rate, and estimated HIV prevalence. Gapminder contains data for all 192 UN members, aggregating data for Serbia and Montenegro. Additionally, it includes data for 24 other areas, generating a total of 215 areas.

GapMinder collects data from a handful of sources, including the *Institute for Health Metrics and Evaulation*, *US Census Bureau's International Database*, *United Nations Statistics Division*, and the *World Bank*.

This portion of the GapMinder data includes one year of numerous country-level indicators of health, wealth and development. Each entry is uniquely identified by their country name. The aim of the project is to predict the residential electricity consumption per person from other measurable variables, specifically alcohol consumption and urban population as these are the only two variables collected on the same year.

Below is the dataset description from the GapMinder Codebook (https://d396qusza40orc.cloudfront.net/phoenixassets/data-management-visualization/GapMinder%20Codebook%20.pdf (https://d396qusza40orc.cloudfront.net/phoenixassets/data-management-visualization/GapMinder%20Codebook%20.pdf)):

Name	Data Type	Measure	Description
incomeperperson	continuous	US$	2010 gross domestic product per capita with inflation taken into account
alcconsumption	continuous	litres	2008 alcohol consumption per adult (age 15+)
armedforcesrate	continuous	%	percentage of armed forces personnel of total labor force
breastcancerper100TH	continuous	rate	2002 breast cancer new cases per 100,000 female
co2emissions	continuous	metric tons	2006 cumulative CO2 emission since 1751
femaleemployrate	continuous	% of population	2007 percentage of adult female employed
HIVrate	continuous	%	2009 estimated HIV prevalence (Ages 15-49)
Internetuserate	continuous	rate	2010 Internet users per 100 people

Name	Data Type	Measure	Description
lifeexpectancy	continuous	years	2011 life expectancy at birth
oilperperson	continuous	tonnes	2010 oil consumption per capita
polityscore	integer	polity	2009 overall polity score from -10 to 10 (overall = autocracy - democracy)
relectricperperson	continuous	kWh	2008 residential electricity consumption per person
suicideper100TH	continuous	rate	2005 suicide per 100,000 people (age adjusted)
employrate	continuous	% of population	2007 percentage of adult employed
urbanrate	continuous	% of population	2008 urban population

Data Preparation and Cleaning

Some data are missing in the dataset. Since all of the variables are numerical, the missing data could be replaced by zeros to allow the calculation of statistics later on.

However, for the computation of correlation scores between the variables, the number of data points for each variable has to be the same. Therefore, the countries with missing data of interest, namely *alcconsumption*, *relectricperperson* and *urbanrate*, are eliminated from the dataset.

```
# Import useful libraries
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
from collections import defaultdict
from scipy.stats import pearsonr

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_
```

```
# Load the GapMinder dataset
f = open("./gapminder.csv")
all_lines = list(csv.reader(f, delimiter = ','))

headers = all_lines[0]  # Extract the headers
dataset = []
for line in all_lines[1:]:
    d = dict(zip(headers, line))
    for header in headers[1:]:
        if ' ' not in d[header]: d[header] = float(d[header])  # Cast the numerical v
        else: d[header] = np.nan
    if d['alcconsumption'] != np.nan and d['relectricperperson'] != np.nan and d['urb
        dataset.append(d)

# Create a dataframe from the dataset
variables_of_interest = ['country', 'alcconsumption', 'relectricperperson', 'urbanrat
data = pd.DataFrame(dataset)[variables_of_interest]
data.set_index('country', inplace=True)

# Check for missing values
print("Total number of entries: {0}\n".format(len(data)))
print("Number of missing data in variables:")
print("alcconsumption: {0}".format(data['alcconsumption'].isnull().sum()))
print("relectricperperson: {0}".format(data['relectricperperson'].isnull().sum()))
print("urbanrate: {0}".format(data['urbanrate'].isnull().sum()))

# First 5 rows
data.head()
```

```
Total number of entries: 213

Number of missing data in variables:
alcconsumption: 26
relectricperperson: 77
urbanrate: 10
```

country	alcconsumption	relectricperperson	urbanrate
Afghanistan	0.03	NaN	24.04
Albania	7.29	636.341383	46.72
Algeria	0.69	590.509814	65.22
Andorra	10.17	NaN	88.92
Angola	5.57	172.999227	56.70

Since all of the variables of interest contain missing values, it is important to figure out the

best way to handle them. My first step is to briefly examine their distribution using frequency tables. The continuous quantitative data are binned into equal quartiles for this purpose.

Since the urbanrate variable has the least missing data, I decided to discard the entries without urbanrate data and fill in the missing values in both consumption variables based on their quartile in urbanrate. The values that are used to fill in the missing spots are the medians of the other countries belonging to the same subgroup quartile as the country with missing data.

As the distribution of both consumption variables seem to be right-skewed, median is chosen instead of mean due to its greater resistence towards influence of outliers.

```
# Investigate frequency distributions of raw data
data['alcpa (litre)'] = pd.cut(data['alcconsumption'], 4, labels=[1, 2, 3, 4])
alc_val_count = data.groupby('alcpa (litre)').size()
alc_dist = data['alcpa (litre)'].value_counts(sort=False, dropna=True, normalize=True
alc_freq_tab = pd.concat([alc_val_count, alc_dist], axis=1)
alc_freq_tab.columns = ['value_count', 'frequency']
print("Frequency table of alcohol consumption per adult:\n{0}\n".format(alc_freq_tab)

data['relectricpp (kWh)'] = pd.cut(data['relectricperperson'], 4, labels=[1, 2, 3, 4]
elec_val_count = data.groupby('relectricpp (kWh)').size()
elec_dist = data['relectricpp (kWh)'].value_counts(sort=False, dropna=True, normalize
elec_freq_tab = pd.concat([elec_val_count, elec_dist], axis=1)
elec_freq_tab.columns = ['value_count', 'frequency']
print("Frequency table of residential electricity consumption per person:\n{0}\n".for

data['urbanr (%)'] = pd.cut(data['urbanrate'], 4, labels=[1, 2, 3, 4])
urb_val_count = data.groupby('urbanr (%)').size()
urb_dist = data['urbanr (%)'].value_counts(sort=False, dropna=True, normalize=True)
urb_freq_tab = pd.concat([urb_val_count, urb_dist], axis=1)
urb_freq_tab.columns = ['value_count', 'frequency']
print("Frequency table of urban population:\n{0}\n".format(urb_freq_tab))

# Code in valid data in place of missing data for each variable
data = data[data['urbanrate'].notnull()]

null_alc_data = data[data['alcconsumption'].isnull()].copy()
alc_map_dict = data.groupby('urbanr (%)').median()['alcconsumption'].to_dict()
print("Median values of alcconsumption corresponding to each urbanrate group:\n{0}\n'
null_alc_data['alcconsumption'] = null_alc_data['urbanr (%)'].map(alc_map_dict)
data = data.combine_first(null_alc_data)
data['alcpa (litre)'] = pd.cut(data['alcconsumption'], 4, labels=[1, 2, 3, 4])

null_elec_data = data[data['relectricperperson'].isnull()].copy()
elec_map_dict = data.groupby('urbanr (%)').median()['relectricperperson'].to_dict()
print("Median values of relectricperperson corresponding to each urbanrate group:\n{
null_elec_data['relectricperperson'] = null_elec_data['urbanr (%)'].map(elec_map_dict
data = data.combine_first(null_elec_data)
data['relectricpp (kWh)'] = pd.cut(data['relectricperperson'], 4, labels=[1, 2, 3, 4]

data.head()
```

```
Frequency table of alcohol consumption per adult:
   value_count  frequency
1           90   0.481283
2           64   0.342246
3           29   0.155080
4            4   0.021390
```

```
Frequency table of residential electricity consumption per person:
   value_count  frequency
1          122   0.897059
2           10   0.073529
3            3   0.022059
4            1   0.007353

Frequency table of urban population:
   value_count  frequency
1           42   0.206897
2           51   0.251232
3           68   0.334975
4           42   0.206897

Median values of alcconsumption corresponding to each urbanrate group:
{1: 3.91, 2: 4.96, 3: 8.35, 4: 9.17}

Median values of relectricperperson corresponding to each urbanrate group:
{1: 59.848274081904805, 2: 278.739635049004, 3: 753.209802163397, 4: 1741.4866504
7562}
```

country	alcconsumption	relectricperperson	urbanrate	alcpa (litre)	relectricp (kWh
Afghanistan	0.03	59.848274	24.04	1	1
Albania	7.29	636.341383	46.72	2	1
Algeria	0.69	590.509814	65.22	1	1
Andorra	10.17	1741.486650	88.92	2	1
Angola	5.57	172.999227	56.70	1	1

Simple Statistics

It is paramount to check out some simple statistics of the variables of interest prior to further analyses of the data. Now that the missing data have been handled and dataset is cleaned, some of the statistics are computed below.

The mean value and range of each variable has been explored. The countries with corresponding maximum and minimum values of each variable are also investigated. However, it is hard to deduce the relationships between these variables without visualizing them at this stage.

```
alc_desc = data['alcconsumption'].astype('float').describe()
elec_desc = data['relectricperperson'].describe()
urb_desc = data['urbanrate'].describe()

print(f"Total number of countries in dataset: {len(data)}\n")
print(f"Average alcohol consumption per person over all countries: {alc_desc['mean']]
print(f"Highest alcohol consumption per person: {alc_desc['max']} litres")
print(f"Country with the highest alcohol consumption per person: {data[data['alcconsu
print(f"Lowest alcohol consumption per person: {alc_desc['min']} litres")
print(f"Country with the lowest alcohol consumption per person: {data[data['alcconsur
print(f"Average residential electricity consumption per person over all countries: {e
print(f"Maximum residential electricity consumption per person: {elec_desc['max']} kW
print(f"Country with the highest residential electricity consumption per person: {dat
print(f"Minimum residential electricity consumption per person: {elec_desc['min']} kW
print(f"Country with the lowest residential electricity consumption per person: {data
print(f"Average urban population over all countries: {urb_desc['mean']} %")
print(f"Highest urban population: {urb_desc['max']} %")
print(f"Country with the highest urban population: {data[data['urbanrate'] == urb_des
print(f"Lowest urban population: {urb_desc['min']} %")
print(f"Country with the lowest urban population: {data[data['urbanrate'] == urb_desc
```

```
Total number of countries in dataset: 203

Average alcohol consumption per person over all countries: 6.843596059113301 litre
s
Highest alcohol consumption per person: 23.01 litres
Country with the highest alcohol consumption per person: Moldova
Lowest alcohol consumption per person: 0.03 litres
Country with the lowest alcohol consumption per person: Afghanistan
Average residential electricity consumption per person over all countries: 953.678
9368884695 kWh
Maximum residential electricity consumption per person: 11154.7550328078 kWh
Country with the highest residential electricity consumption per person: United Ar
ab Emirates
Minimum residential electricity consumption per person: 0.0 kWh
Country with the lowest residential electricity consumption per person: Iraq

Average urban population over all countries: 56.76935960591133 %
Highest urban population: 100.0 %
Country with the highest urban population: Bermuda
Lowest urban population: 10.4 %
Country with the lowest urban population: Burundi
```

Data Visualizations

The distributions of the data for the three variables are visualized using histograms as they are all numeric continuous variables. The histograms also enable the central tendency and skewness of the data to be shown clearly. Box plots are then used to mark the outliers.

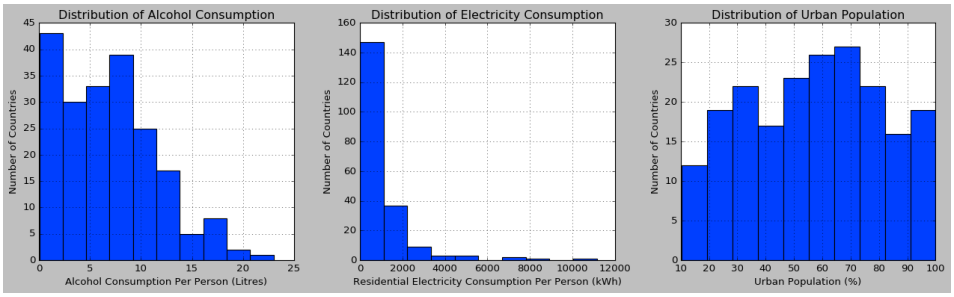
The relationships between the variables are then visualized using scatter plots. The correlations are calculated and displayed on the graphs for easier reference.

Data Distribution

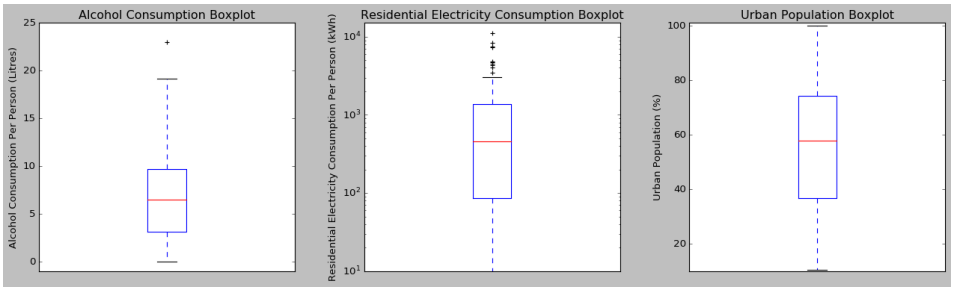
It is concluded from the histograms that the residential electricity consumption per person and alcohol consumption per person are strongly skewed to the right while the urban population roughly follows a normal distribution which peaks around urban population of 65%.

The boxplots allows the identification of the outliers in the residential electricity consumption per person and alcohol consumption per person data. No data point is found to lie outside 1.5 * the interquartile range for the urban population variable. Note that a log scale was used for the plotting of residential electricity consumption per person due to the large order of magnitude in differences between the minimum and maximum of the variable. This allows clearer presentation of all points in the boxplot.

```
# Plot the histograms
style.use('seaborn-bright')
plt.figure(figsize=(16, 5))
plt.subplot(1, 3, 1)
plt.gca().set(xlabel='Alcohol Consumption Per Person (Litres)', ylabel='Number of Cou
plt.grid()
plt.hist(data['alcconsumption'])
plt.subplot(1, 3, 2)
plt.gca().set(xlabel='Residential Electricity Consumption Per Person (kWh)', ylabel='
plt.grid()
plt.hist(data['relectricperperson'])
plt.subplot(1, 3, 3)
plt.gca().set(xlabel='Urban Population (%)', ylabel='Number of Countries', title='Dis
plt.grid()
plt.hist(data['urbanrate'])
plt.tight_layout()
plt.show()
```



```
# Plot the box plots
style.use('classic')
plt.figure(figsize=(16, 5))
plt.subplot(1, 3, 1)
plt.boxplot(data['alcconsumption'])
plt.xticks([])
plt.ylim(-1, None)
plt.gca().set(ylabel='Alcohol Consumption Per Person (Litres)', title='Alcohol Consur
plt.subplot(1, 3, 2)
plt.boxplot(data['relectricperperson'])
plt.xticks([])
plt.yscale('log')
plt.ylim(None, 15000)
plt.gca().set(ylabel='Residential Electricity Consumption Per Person (kWh)', title='F
plt.subplot(1, 3, 3)
plt.boxplot(data['urbanrate'])
plt.xticks([])
plt.ylim(None, 101)
plt.gca().set(ylabel='Urban Population (%)', title='Urban Population Boxplot')
plt.tight_layout()
plt.show()
```



Data Correlation

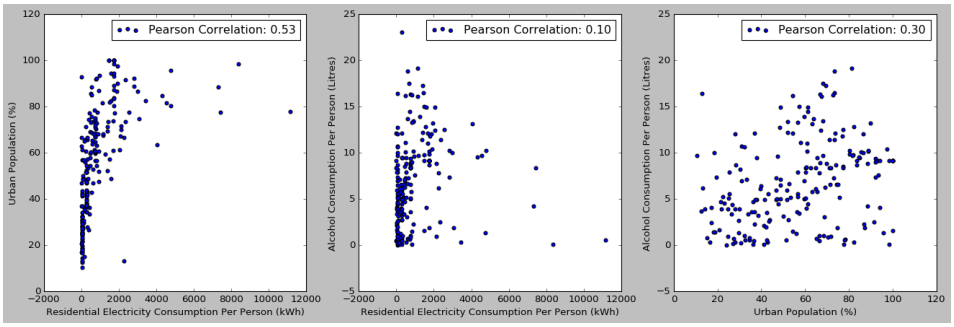
The scatterplots clearly show that both urban population and alcohol consumption are positively correlated with the residential electricity consumption, with the correlation between urban population and residential electricity consumption being much stronger. In fact, alcohol consumption is only slightly correlated with the electricity consumption, as seen from the small magnitude of the correlation.

On a side note, alcohol consumption and urban population are also found to be slightly positively correlated. This could perhaps be explained by greater need for alcohol and greater buying power of the residents in the urban area.

```
# Compute the Pearson's correlation scores
elec_urban_corr = pearsonr(data['relectricperperson'], data['urbanrate'])
elec_alc_corr = pearsonr(data['relectricperperson'], data['alcconsumption'])
urban_alc_corr = pearsonr(data['urbanrate'], data['alcconsumption'])

# Graph the scatter plots
plt.figure(figsize=(20, 6))
plt.subplot(1, 3, 1)
plt.gca().set(xlabel='Residential Electricity Consumption Per Person (kWh)', ylabel='
plt.scatter(data['relectricperperson'], data['urbanrate'], label='Pearson Correlatior
plt.legend()
plt.subplot(1, 3, 2)
plt.gca().set(xlabel='Residential Electricity Consumption Per Person (kWh)', ylabel='
plt.scatter(data['relectricperperson'], data['alcconsumption'], label='Pearson Corre]
plt.legend()
plt.subplot(1, 3, 3)
plt.gca().set(xlabel='Urban Population (%)', ylabel='Alcohol Consumption Per Person (
plt.scatter(data['urbanrate'], data['alcconsumption'], label='Pearson Correlation: {(
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fd3f225cc88>
```



Basic Operations

Regressions and classifications were performed on the variables. The classification models include logistic regression classifier, K-nearest neighbor classifier, and support vector classifier. On the other hand, only a linear regressor model was created to carry out the regression. Default values were chosen for this program as it is still in the exploratory phase. The only exception is that number of neighbors is provided as an input to the K-nearest neighbor classifier as it is mandatory.

The classifications were done on the binned frequency distribution variables created earlier while the regression was carried out on the continuous quantitative data that are first normalized.

The training set was set to 80% of the total number of entries, which means around 160 entries. The remainder entries were used as test set to evaluate the models.

Classification

The evaluation of the classification models reveals that all of them are performing equally well. This is due to the extremely imbalanced dataset that is currently used. Most of the values of the residential electricity consumption are centred at relatively low. Additionally, the amount of data used to train the model is rather little (less than 200). These issues had led the models to consistently predict new unseen data as belonging to the most populated quartile while maintaining their capability to achieve high accuracy.

However, when the classification reports were investigated closely, it could be seen that the models could only achieve less than 50% of F1 scores if macro averages are used. This is because while the models manage to obtain high accuracies in prediction of one of the four classes, they completely ignored the accuracy in other classes.

Regression

The R2 score of the linear regression model is very close to 0. This possibly indicates that the model is also consistently predicting the expected value of the data for the new unseen data, which again could be due to the lack of training data and imbalanced dataset.

However, the mean squared error seems to be pretty low. Further tuning of the hyperparameters of the model could potentially improve its performance.

```
# Scale the contiuous quantitative data
data['alcconsumption'] = MinMaxScaler().fit_transform(np.array(data['alcconsumption'
data['relectricperperson'] = MinMaxScaler().fit_transform(np.array(data['relectricper
data['urbanrate'] = MinMaxScaler().fit_transform(np.array(data['urbanrate']).reshape(

# Prepare training and testing set
X_variables_cls, y_variable_cls = ['urbanr (%)', 'alcpa (litre)'], 'relectricpp (kWh)
X_variables_reg, y_variable_reg = ['urbanrate', 'alcconsumption'], 'relectricperpers
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(data[X_variables_
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(data[X_variables_

# Creating classification and regression models
logreg_clf = LogisticRegression()
knn_clf = KNeighborsClassifier(n_neighbors=3)
svc_clf = SVC()
linreg = LinearRegression()  # The only regression model

# Training data
logreg_clf.fit(X_train_cls, y_train_cls)
knn_clf.fit(X_train_cls, y_train_cls)
svc_clf.fit(X_train_cls, y_train_cls)
linreg.fit(X_train_reg, y_train_reg)

# Predicting outcome on testing set
logreg_pred = logreg_clf.predict(X_test_cls)
knn_pred = knn_clf.predict(X_test_cls)
svc_pred = svc_clf.predict(X_test_cls)
linreg_pred = linreg.predict(X_test_reg)

# Evaluating the classification models
print("Logistic regression classifier accuracy: {0}".format(accuracy_score(y_test_cls
print("K-nearest neighbor classifier accuracy: {0}".format(accuracy_score(y_test_cls,
print("Support vector classifier accuracy: {0}\n".format(accuracy_score(y_test_cls, s

print("Logistic regression classifier precision: {0}".format(precision_score(y_test_c
print("K-nearest neighbor classifier precision: {0}".format(precision_score(y_test_cl
print("Support vector classifier precision: {0}\n".format(precision_score(y_test_cls,

print("Logistic regression classifier recall: {0}".format(recall_score(y_test_cls, lo
print("K-nearest neighbor classifier recall: {0}".format(recall_score(y_test_cls, knn
print("Support vector classifier recall: {0}\n".format(recall_score(y_test_cls, svc_p

print("Logistic regression classifier confusion matrix:\n{0}".format(confusion_matrix
print("K-nearest neighbor classifier confusion matrix:\n{0}".format(confusion_matrix(
print("Support vector classifier confusion matrix:\n{0}\n".format(confusion_matrix(y_

print("Logistic regression classifier classification report:\n{0}".format(classificat
print("K-nearest neighbor classifier classification report:\n{0}".format(classificati
```

```
print("Support vector classifier classification report:\n{0}".format(classification_r

# Evaluating the regression model
print("Linear regressor R2 score: {0}".format(r2_score(y_test_reg, linreg_pred)))
print("Linear regressor mean squared error: {0}".format(mean_squared_error(y_test_reg
```

```
Logistic regression classifier accuracy: 0.975609756097561
K-nearest neighbor classifier accuracy: 0.975609756097561
Support vector classifier accuracy: 0.975609756097561

Logistic regression classifier precision: 0.975609756097561
K-nearest neighbor classifier precision: 0.975609756097561
Support vector classifier precision: 0.975609756097561

Logistic regression classifier recall: 1.0
K-nearest neighbor classifier recall: 1.0
Support vector classifier recall: 1.0

Logistic regression classifier confusion matrix:
[[40  0]
 [ 1  0]]
K-nearest neighbor classifier confusion matrix:
[[40  0]
 [ 1  0]]
Support vector classifier confusion matrix:
[[40  0]
 [ 1  0]]

Logistic regression classifier classification report:
              precision    recall  f1-score   support

           1       0.98      1.00      0.99        40
           2       0.00      0.00      0.00         1

    accuracy                           0.98        41
   macro avg       0.49      0.50      0.49        41
weighted avg       0.95      0.98      0.96        41

K-nearest neighbor classifier classification report:
              precision    recall  f1-score   support

           1       0.98      1.00      0.99        40
           2       0.00      0.00      0.00         1

    accuracy                           0.98        41
   macro avg       0.49      0.50      0.49        41
weighted avg       0.95      0.98      0.96        41

Support vector classifier classification report:
              precision    recall  f1-score   support

           1       0.98      1.00      0.99        40
           2       0.00      0.00      0.00         1

    accuracy                           0.98        41
   macro avg       0.49      0.50      0.49        41
weighted avg       0.95      0.98      0.96        41
```

```
Linear regressor R2 score: 0.021524000041776237
Linear regressor mean squared error: 0.0046087414230685944


/home/jonting/.local/lib/python3.6/site-packages/sklearn/metrics/_classification.p
y:1221  UndefinedMetricWarning: Precision and F-score are ill-defined and being se
t to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```