

Machine Learning 1

by William Henry Baker

FILE	3611_WILLIAM_HENRY_BAKER_MACHINE_LEARNING_1_309231_167082498 9.DOCX (1.1M)		
TIME SUBMITTED	30-NOV-2020 11:10PM (UTC+0000)	WORD COUNT	5302
SUBMISSION ID	137914064	CHARACTER COUNT	29680



Machine Learning 1

Application of Machine Learning (ACCA7025)

William Baker – 1703131

Software Engineering and Artificial Intelligence

Gordon Dicker



Table of Contents

Table of Figures	2
<i>Abstract</i>	3
1. Introduction	3
2. Dataset	4
3. Machine Learning Methodology	4
4. Dataset Cleaning and Pre-processing	5
5. Chosen Algorithms	8
5.1. K-Means Clustering	8
5.2. Linear Regression	8
6. Implementation	8
6.1. Filtering the Data	8
6.2. Merging the Data	9
6.3. Principle Component Analysis	9
6.4. T-Distributed Stochastic Neighbour Embedding	10
6.5. Elbow Method	10
6.6. Clustering	11
6.7. Regression	11
7. Training, Testing, and Evaluation	12
8. Conclusion	14
Bibliography	15
Appendices	16
A. Jupyter Notebook Code	16

Table of Figures

Figure 1 - The results of the Principle Component Analysis algorithm	7
Figure 2 - The results of the t-Distributed Stochastic Neighbor Embedding Algorithm	7
Figure 3 - Filtering the data from the Diabetes dataset.	9
Figure 4 - Merging the datasets.	9
Figure 5 - The Principle Component Analysis function and the output result	10
Figure 6 - The t-Distributed Stochastic Neighbour Embedding algorithm and the results of running on the dataset.....	10
Figure 7 - The elbow method used for calculating the right number of clusters for the K-Means algorithm.....	11
Figure 8 - Running the clustering algorithm with the PCA reduced data set	11
Figure 9 - The Linear Regression algorithm	12
Figure 10 - The Linear Regression Algorithm using the diabetes prevalence and mean BMI.....	13
Figure 11 - Testing the K-Means model using test data run through Principle Component Analysis	14

Abstract

Diabetes is an increasingly common disease, with rates doubling in most countries in the last 15 years. This study aims to use machine learning to gain new insights into this disease and build a predictive model to help estimate future growth rates. Diabetes prevalence levels from several countries spread out across the world when combined with other common health data such as BMI (Body Mass Index), cholesterol level, and blood pressure can be used to create a model to predict the diabetes prevalence through the use of linear regression and machine learning techniques.

1. Introduction

Machine learning is being used in almost all aspects of modern life, although many people do not realise it. With more and more data becoming readily available, machine learning will continue to develop more and more into our lives and could help improve our health and well-being. [1] This paper aims to show how useful machine learning can be when used in a medical environment. Machine learning algorithms, such as clustering, can be used to create and model and find new insights into data by automatically grouping medical data into categories that may not have been thought of before. Regression algorithms can also be used to predict the growth of diseases globally, helping us think of a solution before it happens. [2]

Data containing several metrics such as BMI, blood pressure, and cholesterol can be used to predict the rate and likelihood of diabetes. Each of these metrics has a close relation to diabetes. For example, people with diabetes often have a higher than average BMI and can often have high blood pressure [3]. These correlations will be further investigated in the paper through the use of machine learning techniques. The aim is to use this medical data to create a model that could predict an area's diabetes prevalence by using the average of these metrics versus the currently known diabetes prevalence. This type of data is useful because it is often recorded at general health checkups, so the model could be used to predict if a patient is likely to have diabetes or not.

2. Dataset

The dataset selected for this research was a combination of multiple datasets, all from the same source. The datasets were all country and year based health data; they were diabetes, BMI, cholesterol, and blood pressure. The datasets are provided freely by NCDRisc.org [4] who is a worldwide collection of health scientists. They aim to provide data on the major risk factors for non-communicable diseases (NCDs) for all countries.

This dataset is perfect for our aim; it provides health data on all of the required metrics from all over the world. This should help the model more accurately predict rates of diabetes as the data won't be biased for any specific country. As these datasets are provided separately, they will require joining together based on their country and year. Some datasets cover more years than others; for example, the diabetes dataset only contains data up to 2014, whereas the BMI dataset has data up to 2016.

3. Machine Learning Methodology

Machine learning algorithms are used to find patterns in big datasets; due to the size of these datasets, there can be issues with the data, such as missing values and incorrectly entered or formatted data. This is just one of the issues that can run into when carrying out a machine learning project. For this reason, there needs to be a methodology and a set of steps to follow with each machine learning project. [5]

The first stage of a machine learning project is getting the data; this can be from a freely available source or could be collected by the researcher. If the researcher is collecting data, it should be automated where possible to save time, as a large amount of data will be needed. Multiple datasets can be joined or merged where appropriate.

Once the data has been obtained, the data needs to be explored, looking for insights and studying each attribute of the data. Any missing values or unusual data should be noted at this stage. The data should also be visualised with the use of charts and graphs. Looking for correlations in the data is also a key process at this stage; knowing there are correlations in the data helps understand it further, it helps show if the data will be useful to the model or not.

The next stage is to prepare the data. This will be guided by what was learned in the data exploration stage. Here, any bad data can be removed or replaced. If a value was missing, there are a few options:

- Remove the row – Completely delete this row of data from the set. This can be fine if there is plenty of data
- Replace the value – This could be with the mean, median, or mode.

At this point, the data should be split into two separate sets, one for training the model and one for testing it. The data is split now, before any normalization or standardization to prevent *data snooping*. If the normalisation or standardisation were done before the split, issues would arise as values could be set to a mean of the combination of the sets, values. Meaning when the model is trained, it will have already effectively seen values that will be used for testing. [6]

If needed, standardization, normalisation, and dimensionality reduction can now be carried out on the data. Functions can be created for each of these to make it easier to carry out on test data in the future. At this stage, these functions should only be carried out on the training data.

With the data prepared, the data can be visualised once again, looking at the prepared data compared to how it was when previously explored. The data will look somewhat different due to the functions carried out on it, but correlations should still be there.

Now the model(s) must be selected. Appropriate models should be selected based on the dataset and what the preferred outcome is. This could be an unsupervised algorithm like clustering or a supervised one such as classification or regression. The models should be set up here with any parameters that are required, such as random states – used to ensure that any random values are the same each time the algorithm is run.

With the models selected, the training of them can begin. Here the training data is passed to the model to be trained on. Some training results can also be recorded, such as t intercept and coefficient for in a regression algorithm.

Using the trained models, predictions can now be made. The test data can now be passed into the trained models to produce an outcome prediction. Results of the prediction(s) can be compared to their expected outcome to see how accurate they are.

4. Dataset Cleaning and Pre-processing

With the selected health dataset, a number of data cleaning and pre-processing functions had to be carried out. To begin with, the datasets had to be combined. This was done by merging the datasets based on the 'Country' and 'Year' values. For example, a row in the diabetes dataset with a country of Afghanistan and a year of 1980 would be merged with the matching row in the cholesterol dataset. This was carried out across all datasets until they had all been merged into a single dataset. When merging, only the required columns were

taken, anything that was deemed unnecessary such as 95% upper and lower uncertainty levels were dropped as they would not provide anything for the model.

The 'Country/Region/World', 'ISO', and 'Year' columns were dropped from the dataset as they did not provide any meaningful value to our model. It is not possible to use them to model the prevalence data unless an outcome where we could predict for specific years or countries was wanted. If that was the case, these values could have been converted to use one-hot encoding.

The 'Sex' column contained a String value of the sex, such as 'Male' and 'Female'. Machine learning models are not able to compute String values like this. Therefore, the 'Sex' value was converted into a binary value, using 0 to represent male, and 1 to represent female.

The dataset was then split to prevent any data snooping bias. Dimensionality reduction would be carried out on the dataset; therefore, the data had to be split *before* doing this; else, it would have created values based on the test set too, which would have skewed the results.

Two methods of dimensionality reduction were applied to the data set, PCA (Principal Component Analysis) and t-SNE (t-Distributed Stochastic Neighbor Embedding). Dimensionality reduction allows the data to be easily visualized, as it cuts the number of columns (dimensions) down from eight to just two. This means that it can be easily plotted on a 2D scatter graph. Both algorithms were chosen as although they are both capable of reducing the dimensions, they work in very different ways and can produce totally different outputs. T-SNE is often thought to be the superior algorithm, but it does not learn a function from the original space. Therefore, it takes longer to apply it to future datasets. However, this extra overhead can be worth it if the output is better [7]. Before running these algorithms, the dataset was copied once for each algorithm so they are both running on the same data and not affecting the original dataset.

Figure 1 shows the results of the PCA algorithms. Looking at the results, there are no obvious clusters in the data just yet.

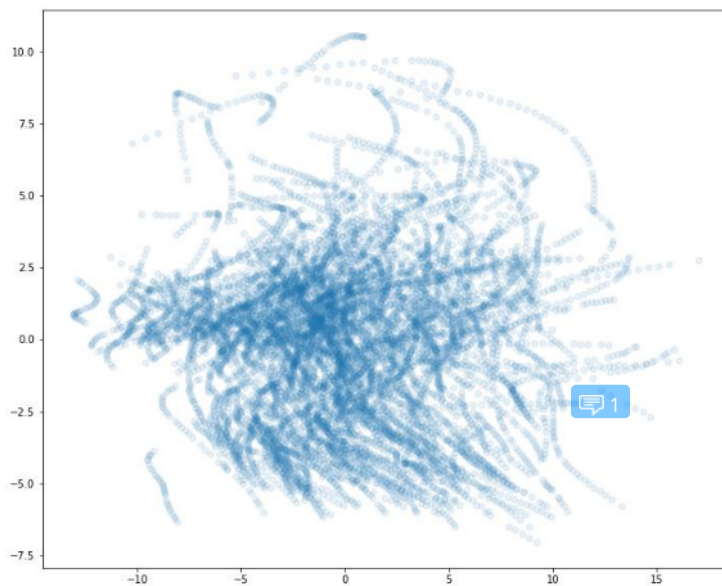


Figure 1 - The results of the Principal Component Analysis algorithm

In Figure 2, the results of t-SNE can be seen. This data seems a little more grouped, with some obvious clusters there. Although the results look very different, the general shape is actually very similar.

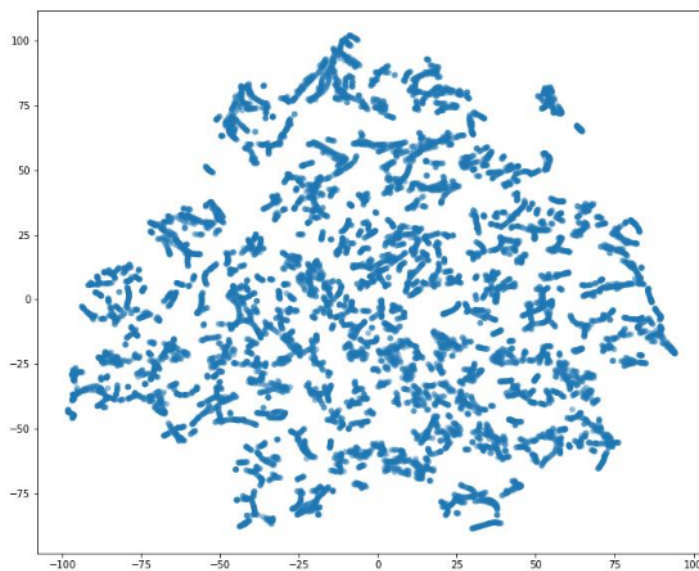


Figure 2 - The results of the t-Distributed Stochastic Neighbor Embedding Algorithm

The dataset contained almost no out of place values or issues, meaning the dataset was already clean; it was likely pre-cleaned by NCDRisC. A number of null values were found in the dataset, which was caused by the merging of multiple datasets, as discussed earlier in the report.

5. Chosen Algorithms

As previously mentioned, two algorithms were chosen to be used on this dataset. These were clustering and regression. Specifically, the aim was to use K-Means for clustering and *linear* regression.

5.1. K-Means Clustering

K-Means is an algorithm capable of clustering datasets quickly and easily. It requires an input number of clusters to look for and tries to group the data based on similarities. It does this by finding points that represent the centroid (cluster centers) and assigning each datapoint to the closest centroid. The centroid of each cluster is then recalculated based on the mean of the points within that cluster. This process is repeated until there is no longer any change in the clusters. [8]

5.2. Linear Regression

Linear regression is a basic regression algorithm that makes a prediction by computing a weighted sum of the input features. It also adds a constant known as the *bias term* to this value. Although a fairly basic algorithm, it can produce excellent results. [8]

6. Implementation

6.1. Filtering the Data

One of the project's first steps was combining the datasets from NCDRisC; before this could be done, the required columns had to be filtered out. This was done using the code shown in figure 3.

```
In [6]: diabetesDf = diabetesDf.filter(['Country/Region/World', 'Sex', 'Year', 'Age-standardised diabetes prevalence'])
diabetesDf.head(3)
```

```
Out[6]:
```

	Country/Region/World	Sex	Year	Age-standardised diabetes prevalence
0	Afghanistan	Men	1980	0.044712
1	Afghanistan	Men	1981	0.046114
2	Afghanistan	Men	1982	0.047601

Figure 3 - Filtering the data from the Diabetes dataset.

6.2. Merging the Data

With the stripped-down datasets, they could now be merged. The merge took place where the 'Country', 'Sex' and 'Year' values matched. The code for this operation can be seen in figure 4. The result of this was a single dataset containing only the required values.

```
In [10]: mergeCols = ['Country/Region/World', 'Sex', 'Year'];
allDf = diabetesDf.merge(bmiDf, on=mergeCols, how='outer')
allDf = allDf.merge(cholesterolDf, on=mergeCols, how='outer')
allDf = allDf.merge(bloodPressureDf, on=mergeCols, how='outer')
allDf.head(3)
```

```
Out[10]:
```

	Country/Region/World	Sex	Year	Age-standardised diabetes prevalence	Mean BMI	Mean total cholesterol (mmol/L)	Mean non-HDL cholesterol (mmol/L)	Mean HDL cholesterol (mmol/L)	Mean systolic blood pressure (mmHg)	Mean diastolic blood pressure (mmHg)
0	Afghanistan	Men	1980	0.044712	19.532923	4.263545	2.919134	1.221839	121.779978	76.662263
1	Afghanistan	Men	1981	0.046114	19.637998	4.266327	2.934723	1.214738	121.924842	76.753616
2	Afghanistan	Men	1982	0.047601	19.741670	4.269252	2.950451	1.207716	122.073346	76.846447

Figure 4 - Merging the datasets.

6.3. Principle Component Analysis

PCA only required a few lines to start working. The training set was copied, and the copy was used to carry out the function. The code and results of this can be seen in Figure 5.

```

In [114]: # Copy the training set
trainingSetPca = trainingSet.copy()

In [115]: from sklearn.decomposition import PCA

In [117]: pca = PCA(n_components=2)

# Here we store the new, two-dimensional dataset
trainingSetPca2d = pca.fit_transform(trainingSetPca)

print(trainingSetPca2d)

[[ -1.3013126  -0.29004662]
 [  0.07771809  -5.5287508 ]
 [-12.68177972   0.64090868]
 ...
 [  0.17568272   0.31074437]
 [ -5.77062519  -3.25482426]
 [  6.41063306   2.31136832]]

```

Figure 5 - The Principle Component Analysis function and the output result

6.4. T-Distributed Stochastic Neighbour Embedding

Once again, the training set was copied to prevent editing the original training set. A random state was added to the t-SNE function so that the randomization would be repeated in the same way on each run.

```

In [124]: # Copy the training set
trainingSetTsne = trainingSet.copy()

In [125]: from sklearn.manifold import TSNE

tsne = TSNE(random_state=17)

# Here we store the new, two-dimensional dataset
trainingSetTsne2d = tsne.fit_transform(trainingSet)

print(trainingSetTsne2d)

[[ -8.661454   36.478912 ]
 [ -9.064649  102.11396 ]
 [-98.18452  -42.859463 ]
 ...
 [ 14.291104   5.4900784]
 [-46.114864   69.29971 ]
 [ 37.507233  -53.707592 ]]

```

Figure 6 - The t-Distributed Stochastic Neighbour Embedding algorithm and the results of running on the dataset

6.5. Elbow Method

The elbow method was used to find the correct number of clusters to use with the K-Means algorithm. This was carried out on both the PCA and t-SNE version of the data. A code example for the PCA version can be seen in Figure 7.

```

In [129]: from sklearn.cluster import KMeans
          from sklearn import metrics
          from scipy.spatial.distance import cdist

          distortions = []

          K = range(2, 25)
          for k in K:
              kMeans = KMeans(n_clusters=k, random_state=42).fit(trainingSetPca2d)
              distortions.append(sum(np.min(cdist(trainingSetPca2d, kMeans.cluster_centers_, 'euclidean'), axis=1)) / trainingSetPca2d.shape[0])

In [130]: xLine = [K[0], K[-1]]
          yLine = [distortions[0], distortions[-1]]

          plot.plot(K, distortions, 'b-')
          plot.plot(xLine, yLine, 'r')
          plot.xlabel('K')
          plot.ylabel('Distortion')
          plot.title('The Elbow method showing the optimal number of clusters')
          plot.show()

```

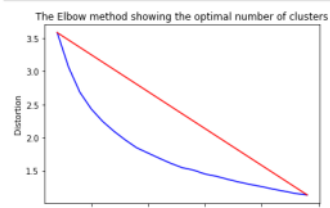


Figure 7 - The elbow method used for calculating the right number of clusters for the K-Means algorithm.

6.6. Clustering

Carrying out the K-Means algorithm only required a single call in the code, simple passing in the two-dimensional training set created with the PCA and t-SNE.

```

In [147]: yPredPca = kMeansPca.fit_predict(trainingSetPca2d)

In [148]: plot.scatter(trainingSetPca2d[:, 0], trainingSetPca2d[:, 1], c=yPredPca)

Out[148]: <matplotlib.collections.PathCollection at 0x1bae306e6d0>

```

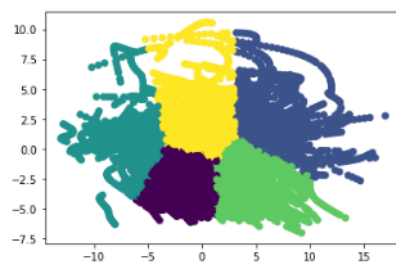


Figure 8 - Running the clustering algorithm with the PCA reduced data set

6.7. Regression

The linear regression algorithm required an x and y input. Y being the value to be predicted, in the case the column containing the diabetes prevalence, and x being the data. These were passed into the algorithm, and the predicted result of y was stored. The returned value was an array of predicted values in an order corresponding to the input. This can be seen in figure 9.

```
In [188]: yActual = testingSetLinReg['Age-standardised diabetes prevalence']
          predictData = testingSetLinReg.drop(columns=['Age-standardised diabetes prevalence'])
          yPred = linRegModel.predict(predictData)

          data = pd.DataFrame({
              'Actual': yActual,
              'Predicted': yPred
          })

          data

Out[188]:
```

Figure 9 - The Linear Regression algorithm

7. Training, Testing, and Evaluation

The previous section showed how the models were implemented; in this section, we will look into the results of the training and testing of these models.

7.1. Linear Regression

	Actual	Predicted
11756	0.114125	0.092990
3710	0.023107	0.002362
3571	0.065465	0.055585
6926	0.114046	0.077896
6224	0.075731	0.068887
...	...	...
5191	0.016594	0.023012
13418	0.082518	0.094200
5390	0.076612	0.089680
860	0.055022	0.047860
7270	0.081222	0.096169

The training of the linear regression algorithm was done on the whole dataset, not a dimensionality reduced or manually reduced dataset. This used a y (The value to be predicted) value of the diabetes prevalence column and the x value containing the rest of the data. When testing this model using the separate testing set, the results were incredibly good. The predicted and actual results were stored in a table to be compared; this can be seen in Figure 9.

The explained variance score for this model was 78%; this could be considered to be a very good score. The maximum error was only 12%.

Overall this model was incredibly good. Because of the correlation between mean bmi and diabetes prevalence, it was decided to try another linear regression model, but this time only using the diabetes prevalence for the y value and mean BMI for the x.

As can be seen in Figure 10, the results of running the linear regression algorithm using only the diabetes prevalence and mean BMI still produces a fairly accurate result, but still not close to the previous test using the full data. This shows that using the full data produces a much more accurate result.

Out[68]:

	Actual	Predicted
2900	0.070387	0.044379
3143	0.081211	0.061190
8231	0.027482	0.014741
3855	0.074238	0.094856
8045	0.036496	0.033989
...	...	...
5130	0.053634	0.077859
9657	0.065792	0.066008
11668	0.097130	0.112306
12065	0.122493	0.114328
10479	0.065853	0.088273

2800 rows x 2 columns

Figure 10 - The Linear Regression Algorithm using the diabetes prevalence and mean BMI

7.2. K-Means

The K-Means algorithm was trained on both PCA and t-SNE reduced datasets. Therefore the test data had also to be run through these functions before being passed into the model. First, the testing set was copied, then the copy was run through the PCA and t-SNE functions, respectively, before being passed into the K-means model.

```
In [93]: testingSetPca = testingSet.copy()
testingSetPca2d = pca.fit_transform(testingSetPca)
yPredPca = kMeansPca.predict(testingSetPca2d)

In [94]: plot.scatter(testingSetPca2d[:, 0], testingSetPca2d[:, 1], c=yPredPca)
Out[94]: <matplotlib.collections.PathCollection at 0x238a4760310>
```

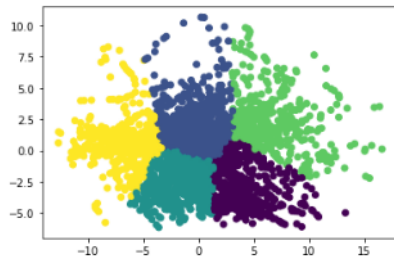


Figure 11 - Testing the K-Means model using test data run through Principle Component Analysis

As can be seen in figure 10, the clustering model produced nice, evenly distributed clusters. The meaning behind these clusters could be further investigated to find out exactly why they were grouped. With more time, the mean diabetes prevalence could have been calculated for each cluster, which could have provided more insight into the clusters.

8. Conclusion

Overall, the machine learning algorithms' results were successful; the linear regression model accurately reproduced results when comparing the predicted result to the actual result. The overall 'R' score – the score used to measure the accuracy of the regression outcome came out as 0.758, a very good result. The clustering model worked well, but some more work into deciphering what the clusters mean could be carried out. The clusters themselves were all evenly distributed, helped by the elbow method to choose the number of clusters.

Bibliography

- [1] S. V. Alex Smola, Introduction to Machine Learning, Cambridge: The Press Syndicated of the University of Cambridge, 2008.
- [2] Z. a. E. E. J. Obermeyer, "Predicting the Future — Big Data, Machine Learning, and Clinical Medicine," *New England Journal of Medicine*, vol. 375, pp. 1216-1219, 2016.
- [3] K. M. V. Narayan, J. P. Boyle, T. J. Thompson, E. W. Gregg and D. F. Williamson, "Effect of BMI on Lifetime Risk for Diabetes in the U.S.," *Diabetes Care*, vol. 30, no. 6, pp. 1562-1566, 2007.
- [4] N. RisC, "NCD RisC," [Online]. Available: <https://ncdrisc.org/>.
- [5] E. Alpaydin, Introduction to Machine Learning, The MIT Press, 2010.
- [6] Y. S. Abu-Mostafa, M. Magdon-Ismael and H.-T. Lin, Learning From Data, AMLBook, 2012.
- [7] H. Herik, E. Postma and L. van der Maaten, "Dimensionality Reduction: A Comparative Review," *Journal of Machine Learning Research - JMLR*, no. 10, 2007.
- [8] G. Aurélien, Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow, Canada: O'Reilly Media Inc, 2019.
- [9] S. C. a. B. Straub, Pro Git, Second Edition, Berkely: Apress, 2014.

Appendices

A. Jupyter Notebook Code

1. Explore the Data

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plot
```

#Set the numpy seed so when splitting the dataset, we get the same result on each run.

```
np.random.seed(99)
```

```
diabetesDf = pd.read_csv('diabetes.csv')
bmiDf = pd.read_csv('body_mass_index.csv', encoding = "ISO-8859-1")
cholesterolDf = pd.read_csv('cholesterol.csv')
bloodPressureDf = pd.read_csv('blood_pressure.csv')
```

1.1. Reading the Data

To start, we will look at some of the data in each of these datasets, so we can decide which columns/data we want to keep.

```
diabetesDf.head(3)
```

```
cholesterolDf.head(3)
```

```
bloodPressureDf.head(3)
```

In each of these datasets, there is a number of columns that we don't need. We can remove the ISO and Uncertainty levels as they won't provide anything to our model.

```
diabetesDf = diabetesDf.filter(['Country/Region/World', 'Sex', 'Year', 'Age-standardised diabetes prevalence'])
diabetesDf.head(3)
```

We will go ahead and remove the unwanted columns, and merge the datasets. This is because we can't properly look at the data till we have a single dataset.

```
bloodPressureDf = bloodPressureDf.filter(['Country/Region/World', 'Sex', 'Year', 'Mean systolic blood pressure (mmHg)', 'Mean diastolic blood pressure (mmHg)'])
bloodPressureDf.head(3)
```

Country/Region/World
 Sex
 Year
 Mean systolic blood pressure (mmHg)
 Mean diastolic blood pressure (mmHg)
 0

Afghanistan

Men

1975

121.078635

76.217903

1

Afghanistan

Men

1976

121.213135

76.302382

2

Afghanistan

Men

1977

121.352221

76.389730

```
cholesterolDf = cholesterolDf.filter(['Country/Region/World', 'Sex', 'Year',
  'Mean total cholesterol (mmol/L)', 'Mean non-HDL cholesterol (mmol/L)', 'Mean HDL cholesterol (mmol/L)'])
cholesterolDf.head(3)
```

Country/Region/World

Sex

Year

Mean total cholesterol (mmol/L)

Mean non-HDL cholesterol (mmol/L)

Mean HDL cholesterol (mmol/L)

0

Afghanistan

Men

1980

4.263545

```
2.919134
1.221839
1
Afghanistan
Men
1981
4.266327
2.934723
1.214738
2
Afghanistan
Men
1982
4.269252
2.950451
1.207716
```

```
bmiDf = bmiDf.filter(['Country/Region/World', 'Sex', 'Year', 'Mean BMI'])
bmiDf.head(3)
```

```
Country/Region/World
Sex
Year
Mean BMI
0
Afghanistan
Men
1975
18.999440
1
Afghanistan
Men
1976
19.105518
2
Afghanistan
Men
1977
19.212080
```

Now we have removed the unwanted column from each of the datasets, we can combine them all.

```
mergeCols = ['Country/Region/World', 'Sex', 'Year'];

allDf = diabetesDf.merge(bmiDf, on=mergeCols, how='outer')
allDf = allDf.merge(cholesterolDf, on=mergeCols, how='outer')
allDf = allDf.merge(bloodPressureDf, on=mergeCols, how='outer')

allDf.head(3)

allDf.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 17600 entries, 0 to 17599
Data columns (total 10 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Country/Region/World                     17600 non-null  object
1   Sex                                       17600 non-null  object
2   Year                                     17600 non-null  int64
3   Age-standardised diabetes prevalence    14000 non-null  float64
4   Mean BMI                                16800 non-null  float64
5   Mean total cholesterol (mmol/L)         15600 non-null  float64
6   Mean non-HDL cholesterol (mmol/L)       15600 non-null  float64
7   Mean HDL cholesterol (mmol/L)           15600 non-null  float64
8   Mean systolic blood pressure (mmHg)     16400 non-null  float64
9   Mean diastolic blood pressure (mmHg)    16400 non-null  float64
dtypes: float64(7), int64(1), object(2)
memory usage: 1.5+ MB
```

We can see here that there are a number of columns that include null values. We know this because the country column has 17600 values, whereas diabetes prevalence only has 14000.

```
allDf.describe()
```

We can create a histogram of each of the attributes to help us detect any outlier or anomalies in the data. As you can see below, there is seemingly no outliers or unexpected values with this data, meaning it has likely already been cleaned.

You would often expect to see entries that are impossible, such as a blood pressure, bmi, or cholesterol level which is not humanly possible, if these existed then we would have to remove or replace the values.

```
allDf.hist(bins=50, figsize=(20,15))

allDf.plot(kind='scatter', x='Mean total cholesterol (mmol/L)', y='Mean BMI',
           alpha=0.2,
           figsize=(15,10), c='Age-standardised diabetes prevalence',
           cmap=plot.get_cmap('jet'), colorbar=True)

<matplotlib.axes._subplots.AxesSubplot at 0x238944c7e20>

png

png
```

In the above scatter graph, we can see that diabetes prevalence increases with mean BMI. We will explore the correlation more in the next section.

1.3. Correlations

Now we have merged all the data and removed the unwanted columns, we can start exploring the data and any correlations that might exist.

```
correlations = allDf.corr()
correlations["Age-standardised diabetes prevalence"].sort_values()
```

Mean HDL cholesterol (mmol/L)	-0.216642
Mean systolic blood pressure (mmHg)	0.006596
Mean diastolic blood pressure (mmHg)	0.010809
Mean total cholesterol (mmol/L)	0.126876
Mean non-HDL cholesterol (mmol/L)	0.288011
Year	0.339923
Mean BMI	0.779119
Age-standardised diabetes prevalence	1.000000

Name: Age-standardised diabetes prevalence, dtype: float64

The above information shows us how closely each attribute correlates to the diabetes prevalence. The closer to 1 each attribute, the more the correlate. You can see that Mean BMI is very closely correlated to diabetes prevalence, with a value of 0.779.

```
from pandas.plotting import scatter_matrix
scatter_matrix(allDf, figsize = (32, 24), alpha=0.2)
```

The most interesting correlation of the above scatter matrix is the diabetes prevalence/mean BMI. We can take a closer look at that below..

```
allDf.plot(kind='scatter', x='Mean BMI', y='Age-standardised diabetes prevalence',
           alpha=0.1)

<matplotlib.axes._subplots.AxesSubplot at 0x2389552dfa0>
```

png

png

```
import seaborn as sns
corrmat = allDf.corr()
top_corr_features = corrmat.index
plot.figure(figsize=(15,15))

g= sns.heatmap(allDf[top_corr_features].corr(), annot=True, cmap="RdYlGn")
```

png

png

As we can see, there is a big correlation between BMI and Diabetes prevalence, and a smaller correlation between other attributes such as blood pressure and cholesterol.

We could add an additional column to determine, high/low blood pressure, or obesity, but because these values are averages, based on a survey of the population, we will avoid this for now.

2. Prepare the Data

Now we have explored the data, and looked at any correlations that exist, we can begin to prepare the data for our chosen Machine learning algorithms. To start, we will clean the data, removing any null values, and removing any attributes that we don't need or may negatively affect the model.

We have already removed a number of columns when we merged the data sets.

#We will make a copy of the data before we start making any changes

```
allDfCopy = allDf.copy()

#We can drop the Country/Region/World and Year columns as they won't provide
anything to our data.
allDfCopy = allDfCopy.drop(columns=['Country/Region/World', 'Year'])

# We can drop any rows that don't have a value for Age-standardised diabetes
prevalence
allDfCopy = allDfCopy.dropna(subset=['Age-standardised diabetes prevalence']
)

len(allDfCopy)

14000

allDfCopy.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 14000 entries, 0 to 13999
Data columns (total 8 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Sex                                         14000 non-null  object
1   Age-standardised diabetes prevalence      14000 non-null  float64
2   Mean BMI                                   14000 non-null  float64
3   Mean total cholesterol (mmol/L)           14000 non-null  float64
4   Mean non-HDL cholesterol (mmol/L)         14000 non-null  float64
5   Mean HDL cholesterol (mmol/L)             14000 non-null  float64
6   Mean systolic blood pressure (mmHg)        14000 non-null  float64
7   Mean diastolic blood pressure (mmHg)       14000 non-null  float64
dtypes: float64(7), object(1)
memory usage: 984.4+ KB

```

Looking at the info once again, we can see that there are no longer any null values.

We want to convert the Sex values from a String to a binary value to allow it to be used in our algorithms. We will simply convert all occurrences of 'Men' into 1, and 'Women' into 0.

```

gender = {'Men': 1, 'Women': 0}

# For each item, set the new Sex based on the set above.
allDfCopy['Sex'] = [gender[item] for item in allDfCopy['Sex']]

allDfCopy.head(2)

Sex
Age-standardised diabetes prevalence
Mean BMI
Mean total cholesterol (mmol/L)
Mean non-HDL cholesterol (mmol/L)
Mean HDL cholesterol (mmol/L)
Mean systolic blood pressure (mmHg)
Mean diastolic blood pressure (mmHg)
0
1
0.044712
19.532923
4.263545
2.919134
1.221839
121.779978

```

76.662263

1

1

0.046114

19.637998

4.266327

2.934723

1.214738

121.924842

76.753616

2.1. Split the Dataset

```
from sklearn.model_selection import train_test_split
```

We will split our dataset between training and testing, using 20% (0.2) for testing.

Random_state is setting a seed of 42 for the randomisation, this ensures our data will come out the same everytime

we run the application

```
trainingSet, testingSet = train_test_split (allDfCopy, test_size=0.2, random_state=42)
```

```
len(trainingSet)
```

11200

```
len(testingSet)
```

2800

2.2. Dimensionality Reduction

For the next step, we will run some dimensionality reduction algorithms to reduce the number of columns from eight to two. This will enable us to more easily visualise our data after we run the K-Means clustering algorithm.

We will test using both PCA (Principle Component Analysis) and t-SNE (t-Distributed Stochastic Neighbor Embedding).

2.2.1. Principle Component Analysis

We will make a copy of the training data set that we will use with PCA, and then for clustering in the future. The results can then be compared to those of the t-SNE algorithm.

```

# Copy the training set
trainingSetPca = trainingSet.copy()

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

# Here we store the new, two-dimensional dataset
trainingSetPca2d = pca.fit_transform(trainingSetPca)

print(trainingSetPca2d)

[[ -1.3013126  -0.29004662]
 [  0.07771809 -5.5287508 ]
 [-12.68177972  0.64090868]
 ...
 [  0.17568272  0.31074437]
 [ -5.77062519 -3.25482426]
 [  6.41063306  2.31136832]]

# We can draw a scatter graph of the result to view the data
plot.figure(figsize=(12,10))
plot.scatter(trainingSetPca2d[:, 0], trainingSetPca2d[:, 1], alpha = 0.1)

<matplotlib.collections.PathCollection at 0x1badb8df220>

png

png

```

2.2.2. t-Distributed Stochastic Neighbor Embedding

Once again, we'll make a copy of the training data to use for the t-SNE algorithm.

```

# Copy the training set
trainingSetTsne = trainingSet.copy()

from sklearn.manifold import TSNE

tsne = TSNE(random_state=17)

# Here we store the new, two-dimensional dataset
trainingSetTsne2d = tsne.fit_transform(trainingSet)

print(trainingSetTsne2d)

[[ -8.661454  36.478912 ]
 [ -9.064649 102.11396 ]
 [-98.18452  -42.859463 ]

```

```
...
[ 14.291104    5.4900784]
[-46.114864   69.29971 ]
[ 37.507233  -53.707592 ]]

plot.figure(figsize=(12,10))
plot.scatter(trainingSetTsne2d[:, 0], trainingSetTsne2d[:, 1],
             edgecolor='none', alpha=0.5)
```

```
<matplotlib.collections.PathCollection at 0x1badd89e160>
```

```
png
```

```
png
```

Although there are no incredibly obvious clusters, it's definitely easy to pickout more than the results of PCA.

3. Visualise the Data

#Put a chart here?

#Gender chart?

```
trainingSet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 11200 entries, 11756 to 7270
```

```
Data columns (total 8 columns):
```

#	Column	Non-Null Count	Dtype
0	Sex	11200 non-null	int64
1	Age-standardised diabetes prevalence	11200 non-null	float64
2	Mean BMI	11200 non-null	float64
3	Mean total cholesterol (mmol/L)	11200 non-null	float64
4	Mean non-HDL cholesterol (mmol/L)	11200 non-null	float64
5	Mean HDL cholesterol (mmol/L)	11200 non-null	float64
6	Mean systolic blood pressure (mmHg)	11200 non-null	float64
7	Mean diastolic blood pressure (mmHg)	11200 non-null	float64

```
dtypes: float64(7), int64(1)
```

```
memory usage: 787.5 KB
```

4. Select a Model

We will use a few models on this dataset, clustering and classification.

4.1. Clustering

We will start off by setting up the clustering model. We will be using K-Means. Before we can do anything, we must decide the number of clusters to use. To do this, we will use the elbow method.

4.1.1. Elbow Method

We will have to carry out the elbow method on both the PCA and t-SNE datasets to decide the correct number of clusters to use for each.

We will use the PCA dataset first..

```
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist

distortions = []

K = range(2, 25)
for k in K:
    kMeans = KMeans(n_clusters=k, random_state=42).fit(trainingSetPca2d)
    distortions.append(sum(np.min(cdist(trainingSetPca2d, kMeans.cluster_centers_, 'euclidean'), axis=1)) / trainingSetPca2d.shape[0])

xLine = [K[0], K[-1]]
yLine = [distortions[0], distortions[-1]]

plot.plot(K, distortions, 'b-')
plot.plot(xLine, yLine, 'r')
plot.xlabel('k')
plot.ylabel('Distortion')
plot.title('The Elbow method showing the optimal number of clusters')
plot.show()
```

png

png

We can see that the elbow is roughly at k=5 clusters.

Let's now do the same for the t-SNE dataset.

```
# Set the number of clusters to be used with K-Means
nClustersPca = 5

distortions = []

K = range(2, 25)
for k in K:
```

```

    kMeans = KMeans(n_clusters=k, random_state=42).fit(trainingSetTsne2d)
    distortions.append(sum(np.min(cdist(trainingSetTsne2d, kMeans.cluster_centers_,
    'euclidean'), axis=1)) / trainingSetTsne2d.shape[0])

xLine = [K[0], K[-1]]
yLine = [distortions[0], distortions[-1]]

plot.plot(K, distortions, 'b-')
plot.plot(xLine, yLine, 'r')
plot.xlabel('k')
plot.ylabel('Distortion')
plot.title('The Elbow method showing the optimal number of clusters')
plot.show()

```

png

png

The graph is very similar, but this time it looks as if the elbow is at k=6 clusters rather than 5.

```

# Set the number of clusters to be used with K-Means
nClustersTsne = 6

```

4.1.2. Creating the Model

Now that we know how many clusters we should be using, we can go ahead and create the K-Means model.

```

# Create the PCA K-Means clustering model
kMeansPca = KMeans(n_clusters=nClustersPca)

# Create the t-SNE K-Means clustering model
kMeansTsne = KMeans(n_clusters=nClustersTsne)

```

4.2. Classification

Next we will create the Model for our classifiers, we will try SVM to start.

```

from sklearn.linear_model import LinearRegression

# Instantiate the Linear Regression model
linRegModel = LinearRegression()

# Copy the training set
trainingSetLinReg = trainingSet.copy()

```

```
# Copy the testing set
```

```
testingSetLinReg = trainingSet.copy()
```

5. Train the Model

We can now train our model, we will start with the PCA data and K-Means

5.1. Clustering

```
yPredPca = kMeansPca.fit_predict(trainingSetPca2d)
```

```
plot.scatter(trainingSetPca2d[:, 0], trainingSetPca2d[:, 1], c=yPredPca)
```

```
<matplotlib.collections.PathCollection at 0x1bae306e6d0>
```

```
png
```

```
png
```

```
yPredTsne = kMeansTsne.fit_predict(trainingSetTsne2d)
```

```
plot.scatter(trainingSetTsne2d[:, 0], trainingSetTsne2d[:, 1], c=yPredTsne)
```

```
<matplotlib.collections.PathCollection at 0x1bae30c96d0>
```

```
png
```

```
png
```

5.2. Regression

```
trainingSetLinReg.head(2)
```

```
Sex
```

```
Age-standardised diabetes prevalence
```

```
Mean BMI
```

```
Mean total cholesterol (mmol/L)
```

```
Mean non-HDL cholesterol (mmol/L)
```

```
Mean HDL cholesterol (mmol/L)
```

```
Mean systolic blood pressure (mmHg)
```

```
Mean diastolic blood pressure (mmHg)
```

```
11756
```

```
0
```

```
0.114125
```

```
24.283264
```

```
4.369073
```

```
3.269678
```

```

1.161644
126.347605
81.726917
3710
1
0.023107
19.487161
3.588294
1.994407
1.025541
128.616151
75.219010

y = trainingSetLinReg['Age-standardised diabetes prevalence']

x = trainingSetLinReg.drop(columns=['Age-standardised diabetes prevalence'])

linRegModel.fit(x, y)

print(linRegModel.intercept_)
print(linRegModel.coef_)

-0.015849942574999804
[-0.00525428  0.01221534 -0.10893415  0.08556497  0.00340161 -0.00015466
  0.00029333]

6. Make a Prediction

6.1. Linear Regression

yActual = testingSetLinReg['Age-standardised diabetes prevalence']

predictData = testingSetLinReg.drop(columns=['Age-standardised diabetes prevalence'])

yPred = linRegModel.predict(predictData)

data = pd.DataFrame({
    'Actual': yActual,
    'Predicted': yPred
})

data
Actual

```

```
Predicted
11756
0.114125
0.092990
3710
0.023107
0.002362
3571
0.065465
0.055585
6926
0.114046
0.077896
6224
0.075731
0.068887
...
...
...
5191
0.016594
0.023012
13418
0.082518
0.094200
5390
0.076612
0.089680
860
0.055022
0.047860
7270
0.081222
0.096169
11200 rows × 2 columns
6.2. Clustering
```

```
testingSetPca = testingSet.copy()
```

```
testingSetPca2d = pca.fit_transform(testingSetPca)
yPredPca = kMeansPca.predict(kMeansPcaTest)
plot.scatter(testingSetPca2d[:, 0], testingSetPca2d[:, 1], c=yPredPca)
<matplotlib.collections.PathCollection at 0x1bae3185c10>
png
png
```

Machine Learning 1

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

58/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8



Comment 1

label your axes!!!

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14



Comment 2

avoid subject comments or follow immediately with figures.



Comment 3

describe the data - units?

Which data

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

GRADING FORM: SMU GENERIC_GJ

WILLIAM HENRY BAKER

58

RESEARCH-INFORMED LITERATURE



References a little light, though most texts of a good quality. Some relevant papers referenced.

6

KNOWLEDGE AND UNDERSTANDING



You apply regression and clustering. PCA carried out. Dimension choices could be better described. Relationship with correlation not discussed. Re clustering, elbow method used - good. Evaluation of regressor needs to be more clearly described. More on loss functions please. Could be clearer in explaining the purpose of the clustering here.

11.5

ANALYSIS



For regression - additional evaluation measures could have been used = - MSE, etc. Data split - good. More analysis of clustering would help. Elbow method for identifying number of clusters applied. Good.

18

PRACTICAL APPLICATION AND DEP.



Good effort here. Some additional methods could have been applied.

20

SKILLS FOR PROFESSIONAL PRAC.



Refs a little light. LABEL your diagrams clearly!

2.5