IFQ509 - Assignment 2

Exploration of Descriptive and Predictive Mining

Team 02G

Viet Huynh Nguyen

Subha Anne Pramudi Perera

Submitted 18/12/2024 with extension approval

Executive summary

This report presents the efforts of 2 team members to perform the data analysis and modelling for Assignment 2A, which focused on **Descriptive** (**Association and Clustering Mining**) and **Predictive Mining** (Classification and regression with **Decision tree**, **Regression Mining** and **Neural Network**) that applied to two provided data sets D1 and D2.

The purpose of Assignment 2A was to explore, pre-process, analyse and build some descriptive and predictive models to gain valuable insights and support decision-making. To show visualisations, we have done with S**eaborn** and **Matplotib** to show graphs and charts of patterns and trends identified in the dataset. To make an in-depth analysis, we have applied many different techniques required for each dataset as follows:

Part A: **Descriptive Mining**

Activity 5.9 analyses the D1.csv dataset consisting of IMDB, an online database of movies that records user ratings of movies on their liking. The **Apriori algorithm** was applied for association analysis to identify the most frequently watched movies and associations between movies that are commonly rated accordingly. The findings show several interesting associations between watched movies.

Activity 6.10 examines the D2.csv dataset consisting of demographic, behavioural and health-related data on individuals who reported being COVID-19 positive. Clustering techniques, specifically **K-means** for numerical data and **Kprototypes clustering** for numerical and categorical data, were applied to identify distinct groups within the dataset. The data needed to be normalised to produce and find the best k value. Visualisations for those clustering techniques used the **Seaborn** package “pairplot”.

Part B: **Predictive Mining**

Decision trees (with the target variable as 'covid19_positive') were built in activity 7.12 on the same D2.csv dataset as activity 6.10. Our aim was to build and evaluate the predictive models of the simple decision tree (**training** and **test** sets were 70:30 splits of the provided data set) with default settings using general **hyperparameters.** We also used **GridSearchCV** from **sklearn.model_selection** to tune parameters to build the best predictive model. The default and the new optimal decision trees can be were compared by checking tree size and accuracy metrics in the relevant classification reports, as well as determination of the area under the **ROC** curve for each model. The higher-performing and more accurate model was the decision tree with optimised parameters. To visualize the decision tree, we exported a PNG extension file with the support of **Pydot** and **Graphviz** modules. Besides, the extraction of the simpler decision tree rules (max-depth =3) was shown by using **export_text** from **sklearn.tree** with the first feature of “income_high”.

Next we used **regression modelling** techniques to predict COVID-19 positivity in the same provided dataset. The dataset was split into training and testing sets with a ratio of 70:30. To prevent issues that could arise from features with different scales, the dataset was **standardised**.

The performance of the regression models was evaluated by comparing the default logistic regression model with one that was tuned using **GridSearchCV**. In addition, further models were created using selected variables from the optimal decision tree derived from the previous effort, together with a model utilising **dimensionality reduction via recursive feature elimination (RFE)**. These models were also tuned with **GridSearchCV** to determine the best value for the regularisation parameter (C). The RFE model slightly outperformed the model based on selected variables. All models showed few signs of overfitting as the training data did not perfectly match the model predictions.

The regression models built using the selected features and RFE were evaluated using **ROC curves**, which compare the true positive rate and false positive rate for different thresholds. The **ROC curve** for the RFE model and the regression model with parameter tuning were very similar, with the RFE model performing marginally better. Interestingly, the optimal decision tree model outperformed the regression model based on selected features from the decision tree in predicting COVID-19 positivity.

From the analysis of the regression models, particularly the RFE model, the top features associated with COVID-19 positivity were identified. These features included being blood type B- (as opposed to A-), having a higher weight, having a medium income, identifying as 'Other' gender, and being shorter in height. Notably, blood type was not a significant factor in the decision tree models as opposed to the regression based models.

Lastly, we built **neural network models** to predict COVID-19 positivity using the same dataset as in previous tasks. We began by training a default neural network model using the **MLPClassifier** with its default parameters, before we used **GridSearchCV** in further models to sequentially tune two key parameters: **hidden_layer_sizes** and **alpha**.

Despite these adjustments, the model still failed to converge, leading to further attempts at fine-tuning the model using **dimensionality reduction** via **RFE** (recursive feature elimination), which reduced the number of features from 44 to 32. This modification showed a slight improvement in test accuracy.

Next, we built a neural network model using the features from the optimal decision tree and tuned the model again with **GridSearchCV** for **alpha** and **hidden_layer_sizes**. Although the test accuracy remained slightly lower compared to previous models, it was still noteworthy. However, convergence was still an issue across all models.

We attempted to solve this by tuning **max_iter**, which is the maximum number of iterations the neural network will perform. After tuning **max_iter** for the selected features neural network model, it finally converged, showing a significant improvement.

The **RFE model**, with 32 input features, was then examined to understand which features were most influential in predicting COVID-19 positivity. The top features, ordered by their importance (coefficients), included variables like **'contacts_count'**, **'smoking_never'**, and **various age categories** (e.g., **'age_0_10'**, **'age_40_50'**).

The performance of all the neural network models was finallyevaluated using **ROC curves**, which plot the true positive rate against the false positive rate. Most models performed similarly, with the **default neural network** showing the weakest performance. The best-performing model, though only marginally better, was the neural network that used more granular tuning of **hidden_layer_sizes**.

Table of contents

Introduction

The following analysis explores several machine learning techniques applied to two distinct datasets. The eventual aim is to evaluate the performance of association mining and clustering models as well as comparing and contrasting various predictive models, such as decision trees, regression models, and neural networks. We will also discuss their utility in identifying associations, predicting outcomes, and providing business value.

Task 1- Activity 5.9: Association mining to understand movie preferences

In order to complete association mining on the provided IMDb dataset, we needed to perform some pre-processing on the fields in the dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8000 entries, 0 to 7999
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   userId     8000 non-null   int64  
 1   movieId    7958 non-null   float64
 2   rating     8000 non-null   float64
 3   timestamp  8000 non-null   object 
 4   imdbId     7759 non-null   object 
 5   title      8000 non-null   object 
dtypes: float64(2), int64(1), object(3)
memory usage: 375.1+ KB
None
```

Attribute	Description	Data type
userID	Unique identifier of the user	ID
movieID	Unique identifier of the movie	ID
rating	Rating given by the user to the movie	Numeric
timestamp	Date and time when the user rated the movie	Datetime
imbdId	Unique identifier of the movie given by the IMDB	ID
title	Title of the movie	String

Table 1: D1.csv variable description (2024) QUT School of Computer Science

With respect to data types as the variable description provided:

- 'movieId' was originally float64 type: this was cast to int64
- 'title' was originally object type: this was cast to string
- 'imdbId' was originally object type: this was cast to string
- 'timestamp' was originally object type: this was cast to datetime64

All other fields remained the same.

There were null values for 'movieId' and 'imdbId' - these were replaced based on populated 'imdbId' and 'movieId' for the same movie, leaving 13 nulls for 'movieId' and 54 nulls for 'imdbId' that could not be inferred from the rest of the data.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8000 entries, 0 to 7999
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   userId     8000 non-null   int64
 1   movieId    7987 non-null   Int64
 2   rating     8000 non-null   float64
 3   timestamp  8000 non-null   object
 4   imdbId     7946 non-null   string
 5   title      8000 non-null   string
dtypes: Int64(1), float64(1), int64(1), object(1), string(2)
memory usage: 382.9+ KB
None
```

However, we are able to perform this replacement, exactly because the three fields 'title', 'movieId' and 'imdbId' all convey the same information. Since the 'title' field has the most complete data (no nulls), we will keep this field and drop the 'movieId'and 'imdbId' fields.

The 'title' field was also aggregated by 'userId' so that we could see all the titles a user had watched in one row per user.

The only fields included in the analysis were 'userId' and 'title' aggregated by 'userId'.

```
userId
2      [Interview with the Vampire, Ed Wood, Disclosu...
3      [The White Stripes: Under Great White Northern...
4      [Addams Family Values, Mystery Men, The Fly, F...
5      [Road to Perdition, Almost Famous, Miss Congen...
6                          [Judge Dredd, Spider-Man 2]
                           ...
667    [Pulp Fiction, The American President, Apollo ...
668                                        [Punchline]
669            [Me, Myself & Irene, Return of the Jedi]
670                      [Se7en, Contact, The Fugitive]
671    [Dolores Claiborne, Amadeus, The Wizard of Oz,...
Name: title, Length: 646, dtype: object
```

We decided on using 'min_support' 0.005 and 'min_confidence' 0.3 as this allowed for interesting rules. Rules with support above 0.01 tended to indicate that most movies were independent of others, and excluded rules that provided useful insights but were excluded by the higher support.

The five most prominent association rules according to these 'min_support' and 'min_confidence' values, ordered by lift were:

- **['Rebel Without A Cause'] -> ['Rear Window']**
    - (This means that users who watch 'Rebel Without A Cause' are more likely to have also seen 'Rear Window')
- **['The Mummy'] -> ['Ace Ventura: Pet Detective']**
    - (This means that users who watch 'The Mummy' are more likely to have also seen 'Ace Ventura: Pet Detective')
- **['Lock, Stock and Two Smoking Barrels'] -> ['Back to the Future']**
    - (This means that users who watch 'Lock, Stock and Two Smoking Barrels' are more likely to have also seen 'Back to the Future')
- **['Mission: Impossible II'] -> ['Twelve Monkeys']**
    - (This means that users who watch 'Mission: Impossible II" are more likely to have also seen 'Twelve Monkeys')
- **['Rocky'] -> ['The Usual Suspects']**
    - (This means that users who watch 'Rocky' are more likely to have also seen 'The Usual Suspects')

We then used the 'timestamp' field, together with the 'userId' and 'title' fields, to conduct sequential mining. This was done by converting the 'timestamp' field ordering the dataset's rows by the 'timestamp' field, dropping all fields except 'userId' and 'title', then aggregating all 'title' values by 'userId'. This results in a list of movies for each user, now ordered in the sequence they watched them. We then used Java's SPMF library via Python for the sequential mining process.

The rules obtained with a ‘min_support’ of 0.005, ordered by lift, were as follows:

	Left_rule	Right_rule	Support	Confidence	Lift
6	[Back to the Future]	[Lock, Stock and Two Smoking Barrels]	0.006192	0.160000	14.765714
5	[Twelve Monkeys]	[Mission: Impossible II]	0.006192	0.266667	14.355556
2	[Pulp Fiction]	[Ace Ventura: When Nature Calls]	0.006192	0.121212	9.787879
0	[Dumb and Dumber]	[Interview with the Vampire]	0.006192	0.210526	7.555556
4	[Dumb and Dumber]	[The Lion King]	0.007740	0.263158	7.083333
1	[Jurassic Park]	[The Mask]	0.006192	0.173913	7.021739
3	[Pulp Fiction]	[Ghost]	0.006192	0.121212	6.525253

Interpreting the rules obtained here via sequential mining, for each rule, if the user has watched the movie on the left side (‘left_rule’) then there is an association with the movie on the right side (‘right_rule’) at some point in time. As we have sorted by high lift values (>1), we can infer that there is a strong positive association between movies in the “left_rule” (precedent) and “right_rule” (antecedent) columns.
In order to obtain the five most watched movies by people who have watched “The Shawshank Redemption”, we obtained the subset of all users in the dataset provided who watched “The Shawshank Redemption. Then we found the top five movies that these users watched (22/646 users). The top five most-watched movies were as follows:

- 'Jurassic Park' (4 times)
- 'Stargate' (3 times)
- 'The Client' (3 times)
- 'Die Hard: With a Vengeance' (3 times)
- 'Muriel's Wedding' (3 times)

```
title
The Shawshank Redemption      22
Jurassic Park                  4
Stargate                       3
The Client                     3
Die Hard: With a Vengeance     3
Muriel's Wedding               3
Name: count, dtype: int64
```

Association analysis for this IMDb dataset is useful for providing users with movie recommendations in the IMDB UI (website), using their users’ movie watching activities to help keep users engaged with the website. Furthermore, this data, combined with other user information, can be used for ad targeting, showing the current users relevant ads based on their watch history. For example, the site could provide user-relevant ads regarding a screening of a movie based on their previous watch history.

Task 2 - Activity 6.10 : Clustering the COVID-19 data

As part of clustering analysis on the COVID-19 dataset provided, we needed to do some pre-processing. The following fields had their datatype adjusted:

- ‘alcohol’ was originally float64 type: this was cast to int64
- ‘contacts_count’ was originally float64 type: this was cast to int64
- ‘worried’ was originally float64 type: this was cast to int64

All other fields remained the same.

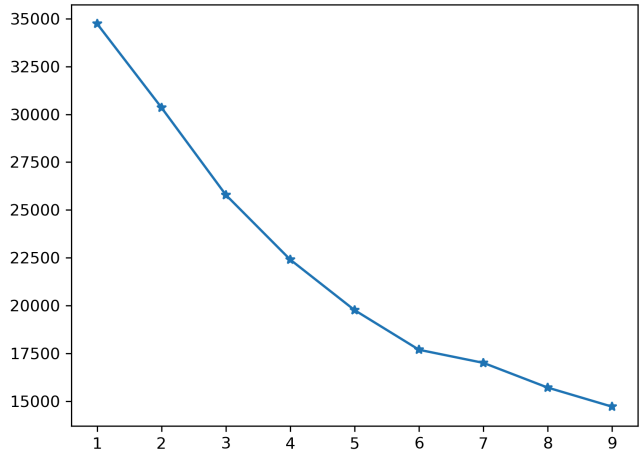
The following shows before and after pre-processing:

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            5789 non-null   object
 1   age               5789 non-null   object
 2   height            5789 non-null   int64
 3   weight            5789 non-null   int64
 4   blood_type        5789 non-null   object
 5   insurance         5789 non-null   object
 6   income            5789 non-null   object
 7   smoking           5789 non-null   object
 8   alcohol           5789 non-null   float64
 9   contacts_count    5789 non-null   float64
 10  working           5789 non-null   object
 11  worried           5789 non-null   float64
 12  covid19_positive  5789 non-null   int64
dtypes: float64(3), int64(3), object(7)
memory usage: 588.1+ KB
None
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5789 entries, 0 to 5788
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            5789 non-null   object
 1   age               5789 non-null   object
 2   height            5789 non-null   int64
 3   weight            5789 non-null   int64
 4   blood_type        5789 non-null   object
 5   insurance         5789 non-null   object
 6   income            5789 non-null   object
 7   smoking           5789 non-null   object
 8   alcohol           5789 non-null   int64
 9   contacts_count    5789 non-null   int64
 10  working           5789 non-null   object
 11  worried           5789 non-null   int64
 12  covid19_positive  5789 non-null   int64
dtypes: int64(6), object(7)
memory usage: 588.1+ KB
None
```

From the dataset, the numerical variables are: ‘height’, ‘weight’, ‘alcohol’, ‘contacts_count’, ‘worried’ and ‘covid19_positive’. We will use these variables for **Kmeans clustering analysis**, after normalisation, as clustering analysis methods are skewed by dominating variables.

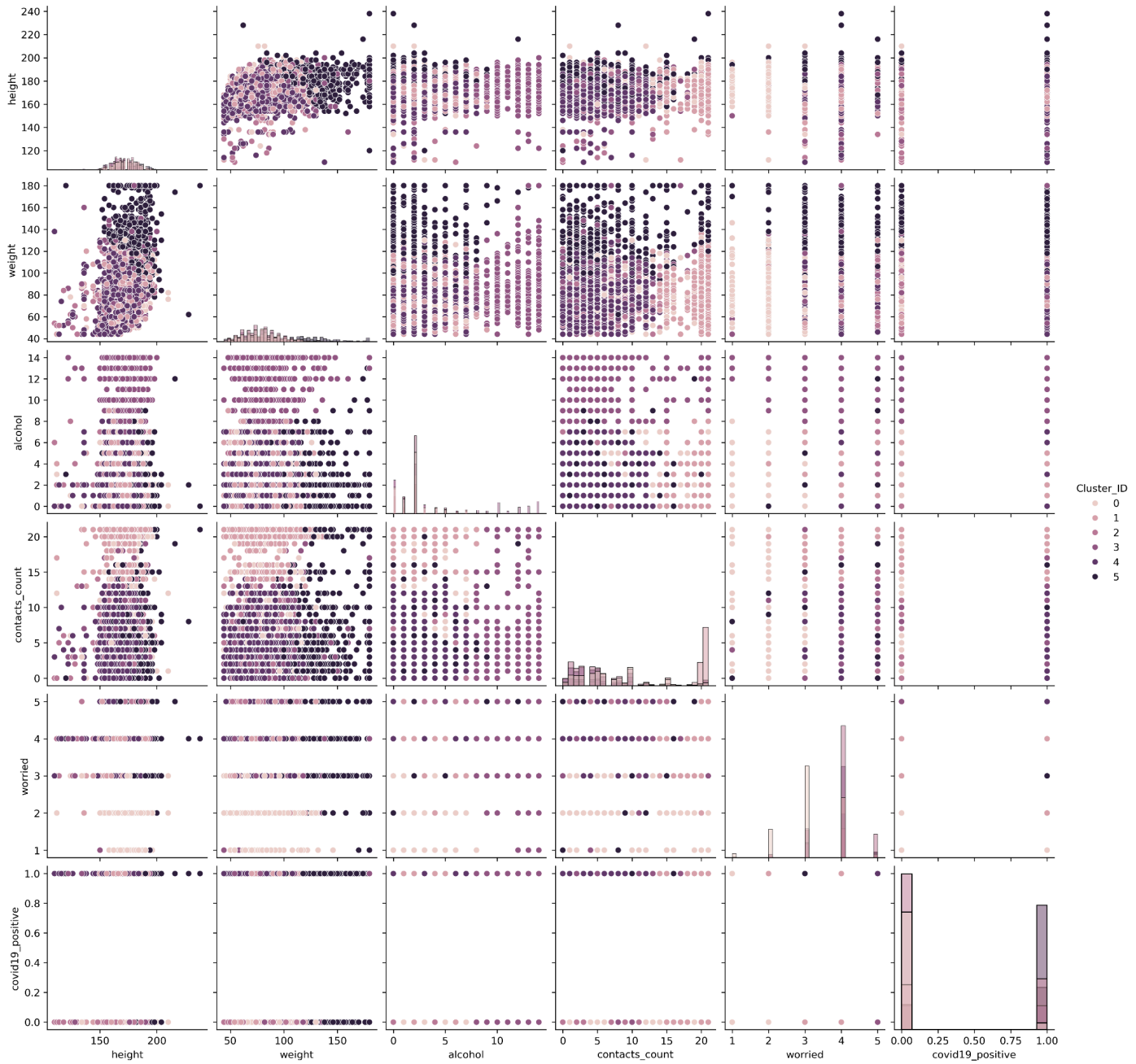
k number was determined by examining a series of silhouette scores for the generated kmeans model and determining which k value produced the greatest silhouette score as well as looking at the ‘elbow’ point, based on inertia and k-values plotted against each other. The normalised data yielded a best k=6.



Comparing normalised values to non-normalised values, the k value is 2 when values are not normalised - this is very different from k = 6 when the values are normalised. Kmeans algorithm is very sensitive to scale

variances between variables, thus it is important to have all variables on the same scale (e.g. 0-1) before conducting clustering analysis so that no one variable affects the clustering calculation too much.

Clustering results were then visualised via seaborn package's 'pairplot'.



Insights:

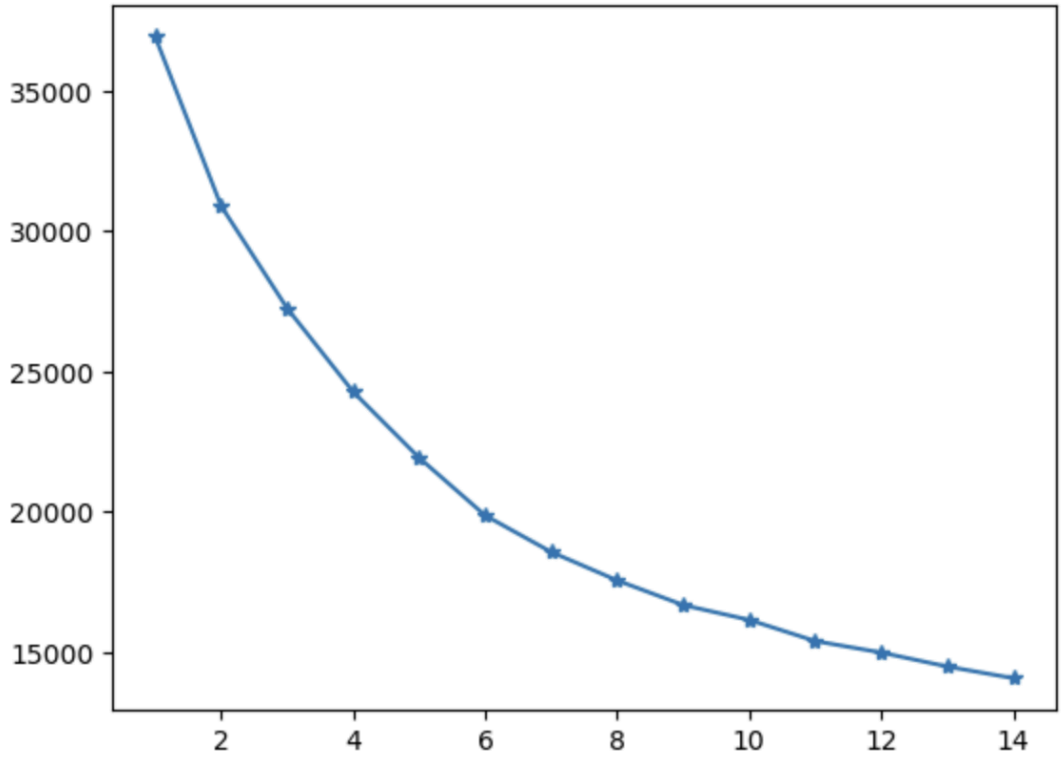
- There seems to be a noticeable positive correlation between height and weight. Clusters are spread across this trend, with certain clusters dominating specific height and weight ranges (e.g., one cluster for shorter and lighter individuals, and another for taller and heavier ones).
- Alcohol usage varies significantly across clusters, with certain clusters showing no usage and others showing higher consumption.
- The range of 'contacts_count' differs across clusters. Some clusters have fewer contacts, while others are spread across higher values. When paired with 'worried', it suggests that higher 'worried' levels might be correlated with lower 'contacts_count' values.

We then looked at characteristics specific to certain clusters, in comparison to the wider dataset.

- Cluster 0: People possibly taking moderate precautions
    - Overall greater numbers of lower ‘worried’ scores
    - Greater numbers of lower ‘contacts_count’ scores
    - Much higher proportion of COVID-19 negativity compared to wider dataset
    - Lower proportion of COVID-19 positivity compared to wider dataset
- Cluster 1: People with possibly unavoidable high contact levels
    - Overall greater numbers of higher ‘worried’ scores
    - Greater numbers of higher ‘contacts_count’ scores
    - Overall higher proportion of COVID-19 positivity
- Cluster 2: Worried people possibly actively avoiding infection
    - Slightly greater proportion of lower ‘weight’ scores
    - Slightly greater proportion of lower ‘height’ scores
    - Far greater numbers of lower ‘alcohol’ scores compared to wider dataset
    - Far greater numbers of lower ‘contacts_count’ scores compared to wider dataset
    - Far higher ‘worried’ scores compared to wider dataset
    - **No** COVID-19 **positive** individuals
- Cluster 3: Heavy drinkers
    - Greater proportion of higher ‘weight’ scores
    - Greater proportion of higher ‘height’ scores
    - Far greater numbers of higher ‘alcohol’ scores compared to wider dataset
    - Overall higher proportion of COVID-19 positivity
    - Overall higher proportion of COVID-19 negativity but COVID-19 positivity still dominant within cluster data
- Cluster 4:
    - Greater proportion of lower ‘weight’ scores
    - Greater proportion of lower ‘height’ scores
    - Greater numbers of lower ‘alcohol’ scores compared to wider dataset
    - Greater numbers of lower ‘contacts_count’ scores compared to wider dataset
    - Far higher ‘worried’=4 scores compared to wider dataset
    - **No** COVID-19 **negative** individuals
- Cluster 5: Larger, heavier people with higher contact levels
    - Far greater proportion of higher ‘weight’ scores
    - Far greater proportion of higher ‘height’ scores
    - Greater numbers of ‘contacts_count’ scores > 10 compared to wider dataset
    - Far higher ‘worried’=4 scores compared to wider dataset
    - Overall higher proportion of COVID-19 positivity

We then used **Kprototypes clustering** on the same fieldset as above, plus the ‘age’ field (that is, ‘age’, ‘height’, ‘weight’, ‘alcohol’, ‘contacts_count’, ‘worried’ and ‘covid19_positive’). The fields were normalised and ‘age’ converted to a numerical series representing the different categories.

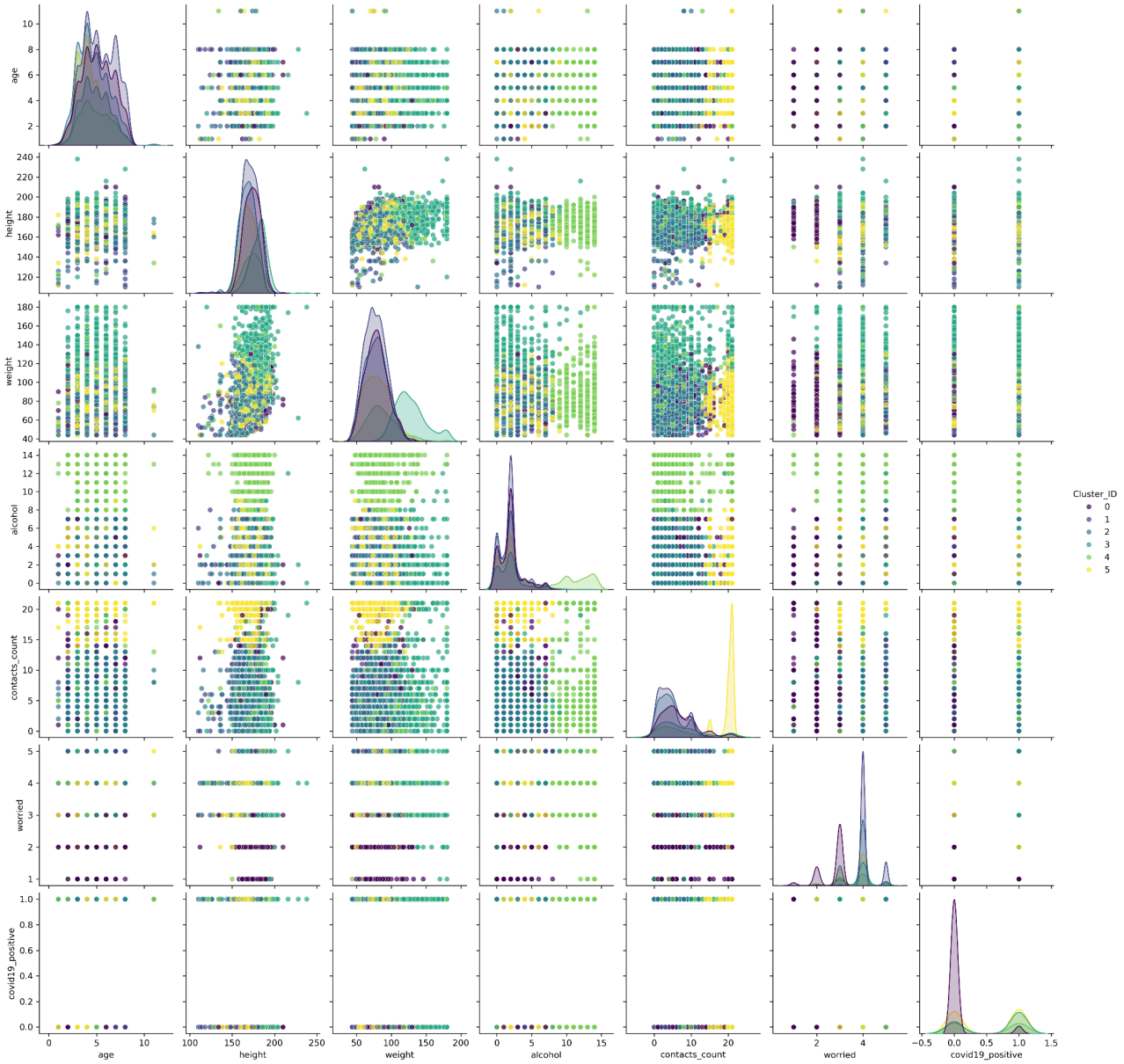
Best k was determined to be k = 6 based on the highest silhouette score.



Note that ‘age’ category was assigned as follows:

Age category	Representation during Kprototypes analysis
0_10	1
10_20	2
20_30	3
30_40	4
40_50	5
50_60	6
60_70	7
70_80	8
80_90	9
90_100	10
100_10	11

Visualising Kprototype clustering via ‘pairplot’:



Once again we examined characteristics specific to certain clusters in this Kprototypes analysis, in comparison to the wider dataset.

- **Cluster 0:**
    - Greater numbers of patients aged under 30 compared to other clusters
    - Overall greater numbers of lower ‘worried’ scores
    - Much higher proportion of COVID-19 negativity compared to wider dataset
    - Greater numbers of lower ‘alcohol’ scores compared to wider dataset
    - Lower proportion of COVID-19 positivity compared to wider dataset
- **Cluster 1**

    - Greater numbers of middle aged patients compared to other clusters
    - Overall greater numbers of higher ‘worried’ scores
    - Much higher proportion of COVID-19 negativity compared to wider dataset
    - Greater numbers of lower ‘alcohol’ scores compared to wider dataset
    - Greater numbers of lower ‘contacts_count’ scores compared to wider dataset
    - **No** COVID-19 **positive** individuals
- **Cluster 2:**
    - Greater numbers of middle aged patients compared to other clusters
    - Overall greater numbers of higher ‘worried’ scores
    - Greater numbers of lower ‘contacts_count’ scores compared to wider dataset
    - Greater numbers of lower ‘alcohol’ scores compared to wider dataset
    - **No** COVID-19 **negative** individuals
- **Cluster 3:**
    - Greater numbers of young adults and early middle aged patients compared to other clusters
    - Taller and heavier compared to the wider dataset
    - Greater numbers of ‘contacts_count’ scores > 10 compared to wider dataset
    - Slightly greater numbers of lower ‘worried’ scores compared to other clusters
    - Greater numbers of lower ‘alcohol’ scores compared to wider dataset
    - Higher proportion of COVID-19 positivity compared to wider dataset
- **Cluster 4:**
    - Greater numbers of young adults compared to other clusters
    - Slightly greater numbers of taller and heavier patients compared to the wider dataset
    - Far greater numbers of higher ‘alcohol’ scores compared to wider dataset
    - Slightly greater numbers of lower ‘worried’ scores compared to other clusters
    - Higher proportion of COVID-19 positivity **and** negativity compared to wider dataset
- **Cluster 5:**
    - Greater numbers of children and young adults compared to other clusters
    - Far greater numbers of ‘contacts_count’ scores > 20 compared to wider dataset
    - Slightly greater numbers of lower ‘worried’ scores compared to other clusters
    - Greater numbers of lower ‘alcohol’ scores compared to wider dataset
    - Higher proportion of COVID-19 positivity **and** negativity compared to wider dataset - positivity dominates

It’s difficult to compare between the Kmeans and Kprototypes clustering of data, as although we have the same number of clusters, the addition of the ‘age’ field into the Kprototypes clustering analysis has changed the cluster split. In general, it seems that higher ‘alcohol’ counts (that is, drinking over more days) and higher ‘contacts_counts’ (that is, being in contact with more people over the last week) seem to be associated with greater proportions of COVID-19 positivity.

Performing clustering analysis on COVID-19 data can provide valuable insights for many relevant decision-makers in the health industry and government sector, for example in evaluating COVID-19 patients

based on symptoms or demographic features such as age, and gender to prioritize treatment or vaccinations. Government bodies can also use such analyses to make decisions on public communication, enact data-driven policies, and support socioeconomic recovery.

Task 3: Building and Evaluating Predictive Models

Taking the same COVID-19 dataset provided for Task 2, we now look to perform predictive analysis.

7.12 Activity 2: Making a decision tree

We began with building a simple decision tree, based on the provided COVID-19 dataset, with the default settings. In this pre-processing, a Python file that we used to adjust the data types and perform the data partition before building a decision tree. This pre-processing is considered as data preparation. We noticed that the data set has no missing values. The dataset was split 70:30 training data to test data. Classification accuracy was 1.0 on the training data (i.e. it matched the dataset perfectly, which makes sense as this is the portion of the dataset it was trained on), and test accuracy was 0.64 (rounded to 2dp).

```
print("Train accuracy:", model.score(X_train, y_train))
```

```
Train accuracy: 1.0
```

```
print("Test accuracy:", model.score(X_test, y_test))
```

```
Test accuracy: 0.6436384571099597
```

The following list represents the original data types for the provided COVID-19 dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5789 entries, 0 to 5788
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            5789 non-null   object
 1   age               5789 non-null   object
 2   height            5789 non-null   int64
 3   weight            5789 non-null   int64
 4   blood_type        5789 non-null   object
 5   insurance         5789 non-null   object
 6   income            5789 non-null   object
 7   smoking           5789 non-null   object
 8   alcohol           5789 non-null   float64
 9   contacts_count    5789 non-null   float64
 10  working           5789 non-null   object
 11  worried           5789 non-null   float64
 12  covid19_positive  5789 non-null   int64
dtypes: float64(3), int64(3), object(7)
memory usage: 588.1+ KB
None
```

The following fields had their datatype adjusted:

- ‘gender’ was originally object type: this was cast to category
- ‘age’ was originally object type: this was cast to category
- ‘blood_type’ was originally object type: this was cast to category
- ‘insurance’ was originally object type: this was cast to int32
- ‘income’ was originally object type: this was cast to category

- ‘smoking’ was originally object type: this was cast to category
- ‘alcohol’ was originally float64 type: this was cast to int64
- ‘contacts_count’ was originally float64 type: this was cast to int64
- ‘working’ was originally object type: this was cast to category
- ‘worried’ was originally float64 type: this was cast to int64

The general hyperparameters used to build the tree are as follows (Scikit-learn developers, n.d.-a):

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'random_state': 59,
 'splitter': 'best'}
```

ccp_alpha: This is a parameter used for Minimal Cost-Complexity Pruning (MCCP). It causes branches of the decision tree to be pruned based on increasing complexity. The higher the value for ‘ccp_alpha’, the more aggressive the pruning. Increasing tree complexity can result from overfitting, and reducing the complexity can help the tree perform better with unknown data as well as data from the test set (Madaan, 2020). In the default tree, we are not performing any MCCP, and ‘ccp_alpha’ is set to 0.0 (Scikit-learn developers, n.d.-b).

class_weight: This parameter allows for variable weighting of classes based on frequency. We can pass the keyword ‘balanced’ to this parameter, which results in classes being weighted in inverse proportion to their frequency. This helps minimise the effect of classes with higher frequency dominating the dataset. This is currently set to ‘None’, meaning that all classes are weighted equally (Scikit-learn developers, n.d.-b).

criterion: Here we can set the function used for measuring the quality of a split. The default value is ‘gini’, but other possible inputs are ‘log_loss’/’entropy’ (Scikit-learn developers, n.d.-b).

max_depth: The number of levels a decision tree can consist of. This is currently set to the default ‘None’, meaning that the tree will branch until each leaf hits the value set for ‘min_samples_split’ (Scikit-learn developers, n.d.-b).

max_features: The number of features the function will analyse when looking for the best split. There are a number of settings for this parameter to allow max_features to be decided based on various rules-of-thumb (e.g. ‘sqrt’, ‘log2’). The current setting is ‘None’, meaning that all features are used to decide the best split.

max_leaf_nodes: The maximum number of leaf nodes in the tree. The current setting is 'None', meaning that there is no limit on 'max_leaf_nodes'. If a number is passed in, the function limits the number of leaf nodes to the number specified based on best split (Scikit-learn developers, n.d.-b).

min_impurity_decrease: This parameter sets the reduction in impurity level when a node branches. If the impurity doesn't decrease by at least this amount, then the node will not branch. The current setting is 0.0, meaning that nodes are not limited in their branching by impurity reduction (Scikit-learn developers, n.d.-b).

min_samples_leaf: The minimum number of rows of data supporting a leaf node. This is currently set to 1, meaning as long as there is one row with data supporting the leaf, then the leaf will exist in the decision tree.

min_samples_split: The minimum number of rows of data for a node to be split. This is currently set to 2, meaning that as long as a node contains at least 2 rows worth of data that support a split, then that node can be split into two leaves, one row pertaining to each leaf (Scikit-learn developers, n.d.-b).

min_weight_fraction_leaf: Default is set to 0.0, meaning that we do not care about the proportion of the total weight at any leaf in comparison to the others, leading to greater numbers of smaller leaves. As this number increases, the leaf is only supported if the total weight meets or exceeds this value (Scikit-learn developers, n.d.-b).

monotonic_cst: Allows for assignment of trends to features - for example, predictions should increase or remain constant as a feature increases if 'monotonic_cst' is 1 for that feature, or predictions should decrease or remain constant as a feature increases if 'monotonic_cst' is -1 for that feature. The currently-used default setting is 'None' (no constraints applied) (Scikit-learn developers, n.d.-b).

random_state: a random integer provided to determine features used at each split (Scikit-learn developers, n.d.-b). The random integer in our case has been decided randomly at the data preparation level.

splitter: This parameter informs the function how to decide on splits. The default is 'best', based purely on the 'criterion' function. There is also a 'random' state, which causes splits to be chosen randomly from a selection of best splits (Scikit-learn developers, n.d.-b).

As we have used default parameters for this first decision tree, the tree produced had no limit on 'max_depth' and a default value of 1 for 'min_samples_leaf', meaning that the tree produced in the above process is very large, having 2015 nodes and 1008 rules (MLJAR, n.d.). Moreover, the first split used in the default tree is 'income_high', which is the fourth most important variable in building the tree, based on feature importance.

```
model.tree_.n_leaves # equivalent to number of rules
```

```
1008
```

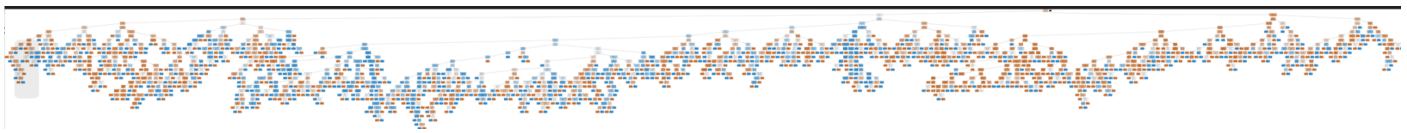
```
model.tree_.node_count # number of nodes
```

```
2015
```

The five most important variables used to build the default tree are:

```
weight : 0.13941453768669052
height : 0.11377082940329641
contacts_count : 0.10035236503146555
income_high : 0.09319573009185216
worried : 0.07600402016795169
```

Decision tree visualisation exported to PNG extension to show that the model is very complex and deep:



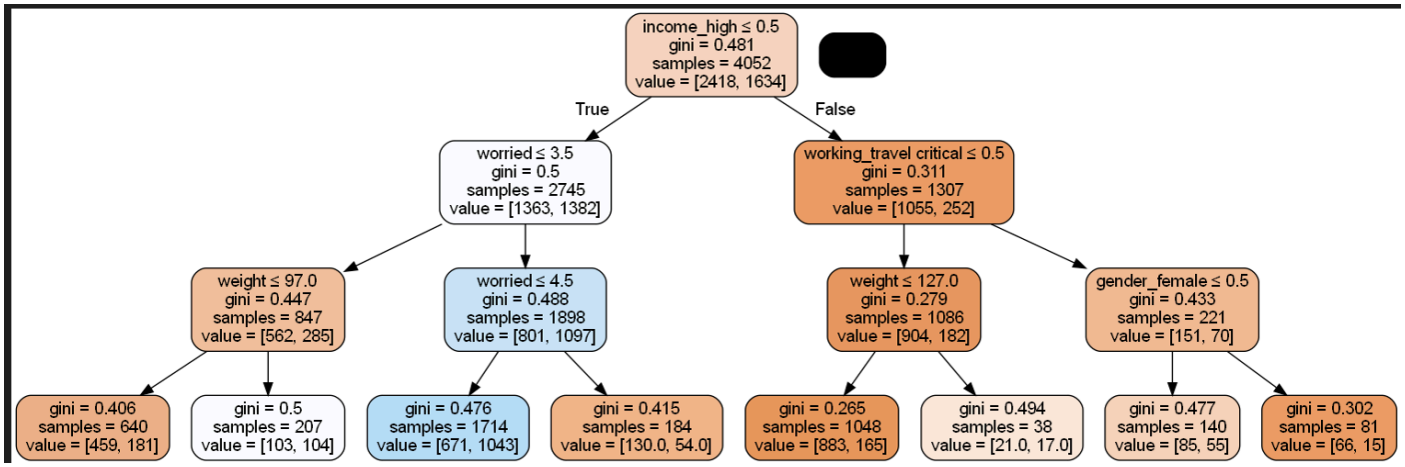
Again, we had to limit the complexity of the model by setting the max-depth equal to 3, and retrain the model:

```
Train accuracy: 0.6932379072063178
Test accuracy: 0.6954519286125503
              precision    recall  f1-score   support

           0       0.75      0.73      0.74      1036
           1       0.62      0.64      0.63       701

    accuracy                           0.70      1737
   macro avg       0.68      0.69      0.69      1737
weighted avg       0.70      0.70      0.70      1737
```

It is noted that the simpler model with a smaller value max-depth performs better on the test sets and performs poorly on the training sets. Again, The five most important variables used to build the default tree are:

```
income_high : 0.5782050326572803
worried : 0.3247093022542382
weight : 0.040748662456435755
working_stopped : 0.0391566336528118
smoking_yesheavy : 0.01718036897923404
```

Visualisation of this simpler decision tree with the max-depth value of 3 is as follows:



Next, we rebuilt the decision tree based on the same dataset, but we used GridSearchCV to determine the best hyperparameters to pass into DecisionTreeClassifier. We began with ‘criterion’, ‘max_depth’ and ‘min_samples_leaf’ (attempted to input further at this level but this caused the notebook to freeze), and GridSearchCV determined that the best parameters to use were {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 5}. We then ran GridSearchCV again with the optimal values for ‘criterion’, ‘max_depth’ and ‘min_samples_leaf’, adding in class_weight', 'max_features','max_leaf_nodes’, 'min_impurity_decrease', and 'splitter'. GridSearchCV produced the following best parameters:

```
{'class_weight': None, 'criterion': 'gini', 'max_depth': 6, 'max_features': None, 'max_leaf_nodes':
10, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'splitter': 'random'}
```

On creating a decision tree based on these parameters, the classification accuracy of this optimal tree was 0.69 (rounded to 2dp) on the training data, but test accuracy was improved to 0.69 (rounded to 2dp). The new decision tree had 10 rules and 19 nodes, with the initial split still based on ‘income_high’.

```
get_rule_count(optimal_model, X.columns, None)
```

```
10
```

```
optimal_model.tree_.node_count # number of nodes
```

```
19
```

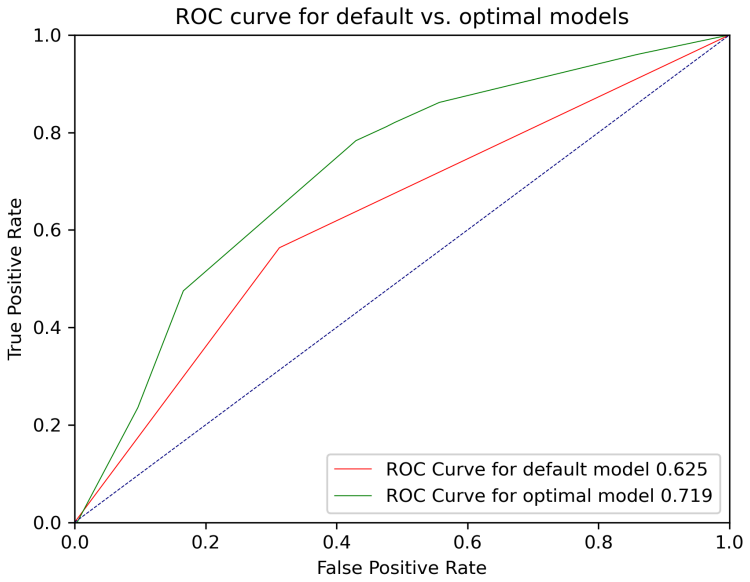
The five most important variables used to build the optimal tree are:

```
income_high : 0.5791998451589087
worried : 0.16450489603474816
working_travel critical : 0.1103853462613169
working_stopped : 0.0738059191836583
income_med : 0.02956475619768735
```

There is little evidence of overfitting, as the model doesn’t seem to have an overly high accuracy with respect to the training dataset in comparison to the test dataset.

This new optimal decision tree model is far smaller than the default tree, with 998 less rules and 1996 less nodes.

The ROC curve for these two models are as follows:

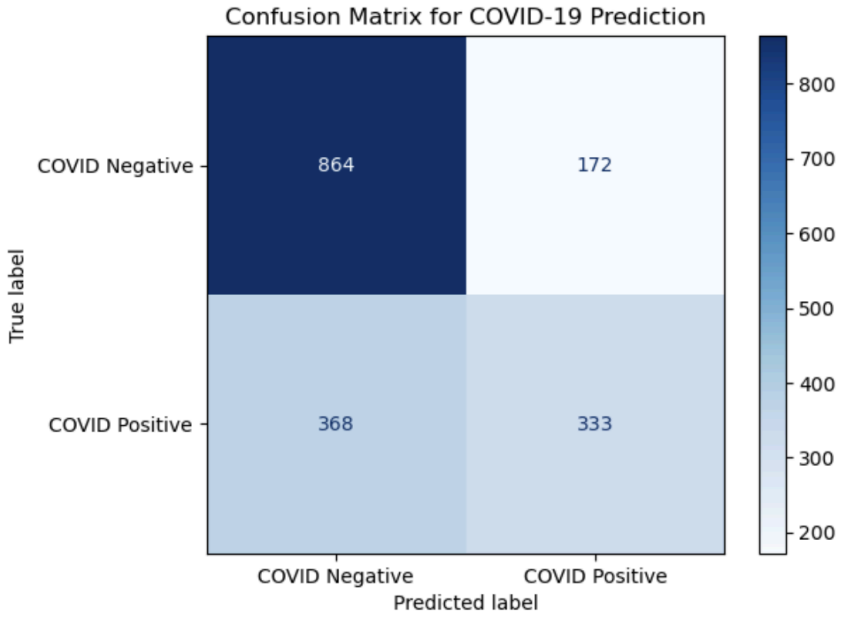


Here we can easily see that the new optimal model has a higher true positive rate and a lower false positive rate than the default model. By generalising the decision tree in limiting nodes, leaves and depth, the tree is less fitted to the training data and now performs better in predictions against the test data.

Visualising the optimal tree:

```
|--- income_high <= 0.41
|   |--- working_travel critical <= 0.56
|   |   |--- worried <= 4.42
|   |   |   |--- working_stopped <= 0.29
|   |   |   |   |--- class: 0
|   |   |   |--- working_stopped >  0.29
|   |   |   |   |--- worried <= 3.99
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- worried >  3.99
|   |   |   |   |   |--- income_low <= 0.54
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- income_low >  0.54
|   |   |   |   |   |   |--- class: 0
|   |   |--- worried >  4.42
|   |   |   |--- class: 0
|   |--- working_travel critical >  0.56
|   |   |--- income_med <= 0.32
|   |   |   |--- class: 0
|   |   |--- income_med >  0.32
|   |   |   |--- class: 1
|--- income_high >  0.41
|   |--- working_stopped <= 0.27
|   |   |--- smoking_yesheavy <= 0.22
|   |   |   |--- class: 0
|   |   |--- smoking_yesheavy >  0.22
|   |   |   |--- class: 1
|   |--- working_stopped >  0.27
|   |   |--- class: 0
```

The confusion matrix for the optimal model looks like this:



This means that the optimal model correctly predicted 864/1036 COVID-19 negative individuals, and 333/701 COVID-19 positive individuals.

According to the optimal tree, COVID-19 positive individuals are likely to have the following characteristics:

- Heavy smokers (in conjunction with higher income)
- Have not stopped working
- Are not as worried
- Have lower income in general

8.10 Activity 1: Making a regression model for Assignment 2A: Project

We will now look at predictive analysis of the provided COVID-19 dataset via regression modelling.
We continued with the same training/testing split (70:30). We also standardised the original pre-processed dataset used in the decision tree analysis before using it for logistic regression modelling, as features that are on different scales can skew the model based on the scale instead of the actual data. Larger scales end up affecting the model disproportionately - hence we must ensure all features use the same scale. Furthermore, features on different scales used in the same regression model may cause convergence failures, and will be affected differently by the penaliser ('l1'/'l2'). Standardisation ensures that any penalties are applied uniformly across all features.
The regression model with C parameter tuning, solver = 'newton-cholesky' and penalty='l2' via GridSearchCV is very slightly better with respect to test data accuracy than the default regression model (0.69315 test accuracy in the default regression model in comparison to 0.69430 test accuracy in the regression model with parameter tuning).
The model with higher test accuracy contains all features from the original dataset. Here they are, ordered by absolute coefficient (weighting) descending:

```
income_high                    : -0.3228880415180645 abs: 0.3228880415180645
income_med                     : 0.30723264584212845 abs: 0.30723264584212845
weight                         : 0.25113490624686535 abs: 0.25113490624686535
working_travel critical        : 0.20600591201101126 abs: 0.20600591201101126
age_70_80                      : -0.1221593636281359 abs: 0.1221593636281359
worried                        : 0.12168939802509429 abs: 0.12168939802509429
working_home                   : -0.12063405715384569 abs: 0.12063405715384569
height                         : -0.11440775381404908 abs: 0.11440775381404908
age_60_70                      : -0.1140077741592616 abs: 0.1140077741592616
age_20_30                      : 0.10607971456787195 abs: 0.10607971456787195
age_100_110                    : 0.09760477059414664 abs: 0.09760477059414664
insurance                      : 0.0953625700357334 abs: 0.0953625700357334
smoking_yesheavy               : 0.08830886652475246 abs: 0.08830886652475246
working_travel non critical    : -0.07921461291617266 abs: 0.07921461291617266
smoking_quit5                  : -0.06588847665609958 abs: 0.06588847665609958
working_stopped                : -0.06130238890852926 abs: 0.06130238890852926
smoking_quit10                 : 0.057200305157857304 abs: 0.057200305157857304
income_gov                     : 0.05684896010715584 abs: 0.05684896010715584
blood_type_abp                 : 0.04469160861337181 abs: 0.04469160861337181
gender_other                   : 0.03787096286915727 abs: 0.03787096286915727
alcohol                        : 0.03742075061702609 abs: 0.03742075061702609
age_50_60                      : 0.036904217445128826 abs: 0.036904217445128826
smoking_never                  : -0.03561094240055872 abs: 0.03561094240055872
smoking_vape                   : 0.03407898740122742 abs: 0.03407898740122742
blood_type_on                  : -0.03198500408697733 abs: 0.03198500408697733
age_10_20                      : 0.031743140710122054 abs: 0.031743140710122054
age_30_40                      : 0.03124818039729284 abs: 0.03124818039729284
blood_type_unknown             : 0.030129694292459496 abs: 0.030129694292459496
contacts_count                 : 0.029818935056045246 abs: 0.029818935056045246
gender_female                  : -0.0280098552015366 abs: 0.0280098552015366
income_low                     : -0.025794547598802443 abs: 0.025794547598802443
gender_male                    : 0.02464164873491532 abs: 0.02464164873491532
age_40_50                      : -0.023432399255774543 abs: 0.023432399255774543
blood_type_op                  : -0.022713612762672565 abs: 0.022713612762672565
blood_type_an                  : -0.02188938161438303 abs: 0.02188938161438303
blood_type_bp                  : 0.02093891382734921 abs: 0.02093891382734921
smoking_yeslight               : -0.018520647254809085 abs: 0.018520647254809085
smoking_quit0                  : 0.01674282040343572 abs: 0.01674282040343572
smoking_yesmedium              : 0.014760548753976695 abs: 0.014760548753976695
blood_type_ap                  : -0.00803333745935436 abs: 0.00803333745935436
blood_type_bn                  : -0.008003487346143924 abs: 0.008003487346143924
blood_type_abn                 : -0.007795900356880461 abs: 0.007795900356880461
age_0_10                       : 0.002297325548235284 abs: 0.002297325548235284
working_never                  : 0.0015972613564896004 abs: 0.0015972613564896004
```

The top five most important variables, in order:

```
income_high                  : -0.3228880415180645 abs: 0.3228880415180645
income_med                   : 0.30723264584212845 abs: 0.30723264584212845
weight                       : 0.25113490624686535 abs: 0.25113490624686535
working_travel critical      : 0.20600591201101126 abs: 0.20600591201101126
age_70_80                    : -0.1221593636281359 abs: 0.1221593636281359
```

Default model:

```
Train accuracy: 0.6853405725567621
Test accuracy: 0.6931491076568796
              precision    recall  f1-score   support

           0       0.72      0.79      0.75      1036
           1       0.64      0.55      0.59       701

    accuracy                           0.69      1737
   macro avg       0.68      0.67      0.67      1737
weighted avg       0.69      0.69      0.69      1737
```

Model with parameter tuning:

```
Train accuracy: 0.6875616979269497
Test accuracy: 0.694300518134715
              precision    recall  f1-score   support

           0       0.72      0.80      0.76      1036
           1       0.65      0.54      0.59       701

    accuracy                           0.69      1737
   macro avg       0.68      0.67      0.67      1737
weighted avg       0.69      0.69      0.69      1737
```

There does not seem to be obvious overfitting between the default regression model and the regression model with parameter tuning - we are not seeing a 100% match between the model and the training data in either instance.

We then went on to build further regression models, one based on the selected variables from the optimal decision tree built in the previous section, and one utilising dimensionality reduction via recursive feature elimination (RFE). These models were both tuned with GridSearchCV to determine their best C value.

Accuracy and classification scores for the selected variables model were as follows:

```
Train accuracy: 0.6762092793682132
Test accuracy: 0.6747265400115141
              precision    recall  f1-score   support

           0       0.70      0.79      0.74      1036
           1       0.62      0.50      0.55       701

    accuracy                           0.67      1737
   macro avg       0.66      0.65      0.65      1737
weighted avg       0.67      0.67      0.67      1737

{'C': 10}
```

Accuracy and classification scores for the RFE model were as follows:

```
Train accuracy: 0.6875616979269497
Test accuracy: 0.697754749568221
              precision    recall  f1-score   support

           0       0.72      0.80      0.76      1036
           1       0.65      0.55      0.60       701

    accuracy                           0.70      1737
   macro avg       0.69      0.67      0.68      1737
weighted avg       0.69      0.70      0.69      1737

{'C': 0.1}
```

Based on these reports, the RFE model performs marginally better than the selected variables regression model.

The RFE model (with parameter tuning) has the following variables, ordered by coefficient value descending:

```
blood_type_an                : -0.34748460786444246 abs: 0.34748460786444246
blood_type_bn                : 0.32842512946011204 abs: 0.32842512946011204
weight                       : 0.2912121155412531 abs: 0.2912121155412531
income_med                   : 0.22787769509912148 abs: 0.22787769509912148
gender_other                 : 0.21963471545791546 abs: 0.21963471545791546
height                       : -0.15814620418342962 abs: 0.15814620418342962
age_50_60                    : -0.14522041339110212 abs: 0.14522041339110212
income_high                  : -0.14231939718430311 abs: 0.14231939718430311
contacts_count               : 0.13080851299729113 abs: 0.13080851299729113
age_40_50                    : -0.1299644071186225 abs: 0.1299644071186225
age_100_110                  : 0.1218682214724742 abs: 0.1218682214724742
insurance                    : 0.1111606498444884 abs: 0.1111606498444884
blood_type_op                : -0.09884488368471374 abs: 0.09884488368471374
blood_type_unknown           : 0.09167870069626095 abs: 0.09167870069626095
smoking_never                : -0.08949261830012778 abs: 0.08949261830012778
blood_type_bp                : -0.07944116944224504 abs: 0.07944116944224504
income_low                   : -0.07800035875643192 abs: 0.07800035875643192
blood_type_abp               : 0.06468035279553358 abs: 0.06468035279553358
age_60_70                    : 0.06041588827662756 abs: 0.06041588827662756
blood_type_abn               : 0.05951523423184628 abs: 0.05951523423184628
worried                      : -0.044131684537147296 abs: 0.044131684537147296
blood_type_on                : 0.043658202194834836 abs: 0.043658202194834836
age_0_10                     : 0.04140168416494834 abs: 0.04140168416494834
gender_male                  : 0.04113603532463754 abs: 0.04113603532463754
alcohol                      : 0.040496053282036815 abs: 0.040496053282036815
age_70_80                    : 0.040492907882447846 abs: 0.040492907882447846
gender_female                : 0.040471106314432674 abs: 0.040471106314432674
age_30_40                    : 0.039383214942463586 abs: 0.039383214942463586
income_gov                   : -0.0378951929229421 abs: 0.0378951929229421
age_10_20                    : 0.03441051755053889 abs: 0.03441051755053889
age_20_30                    : -0.02546826377045075 abs: 0.02546826377045075
blood_type_ap                : -0.02514412076702498 abs: 0.02514412076702498
```

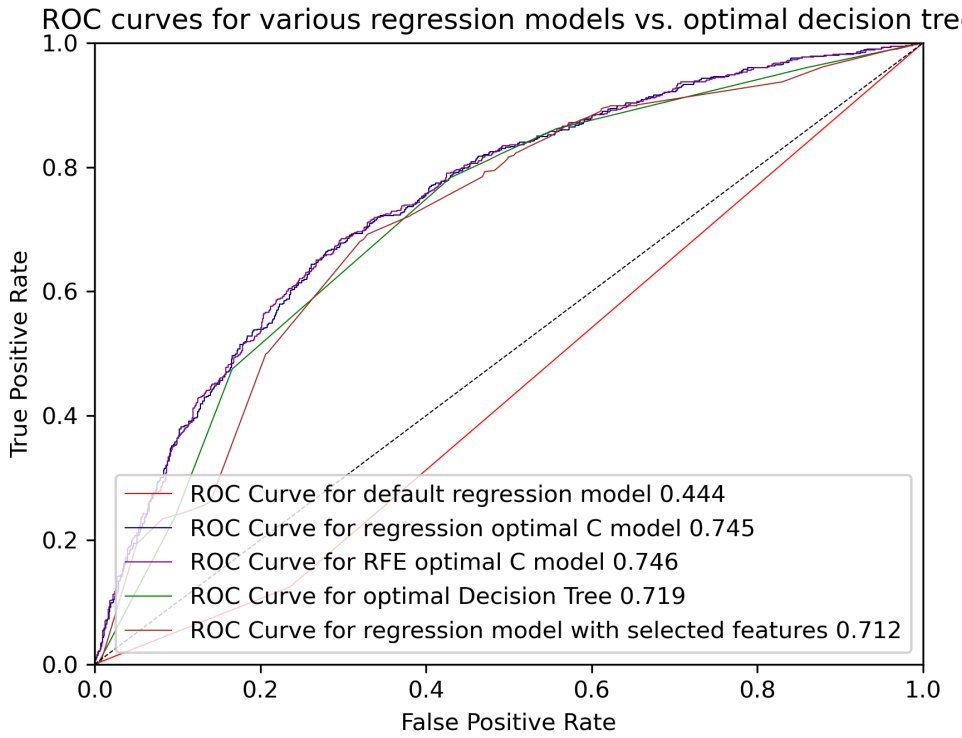
The top three important variables are thus:

```
blood_type_an                : -0.34748460786444246 abs: 0.34748460786444246
blood_type_bn                : 0.32842512946011204 abs: 0.32842512946011204
weight                       : 0.2912121155412531 abs: 0.2912121155412531
```

Interestingly, no ‘blood_type’ variables made it into the top five important variables when building either the default or optimal decision trees.

Again, there does not seem to be obvious overfitting in the selected variables model and the RFE model - we are not seeing a 100% match between the model and the training data in either instance.

The ROC curve for all four regression-based models, plus the ROC curve for the optimal decision tree are as follows:



The RFE regression model and the regression model with parameter tuning are similar, with the RFE model very narrowly beating out the regression model with parameter tuning. Interestingly, the optimal decision tree is better at predicting true positivity than a regression model built on the selected features in the base decision tree.

Based on the coefficient ordering for the RFE regression model, COVID-19 positive patients are likely to have or be:

- **Not** blood type A-
- Blood type B-
- Heavier in weight
- Medium income
- 'Other' gender
- **Not** being taller in height

8.11 Activity 2: Making a neural network for Assignment 2A: Project

Finally, we perform predictive analysis via neural networks with the same COVID-19 dataset used across Task 3. (Note that training-testing split, preprocessing and First, we built a default neural network using MLPClassifier's default parameter list. The accuracy scores and classification report for this default model is as follows:

```
Train accuracy: 0.8877097729516288
Test accuracy: 0.6689694876223373
              precision    recall  f1-score   support

           0       0.72      0.73      0.73      1036
           1       0.59      0.57      0.58       701

    accuracy                           0.67      1737
   macro avg       0.66      0.65      0.65      1737
weighted avg       0.67      0.67      0.67      1737

MLPClassifier(random_state=61)
/opt/anaconda3/lib/python3.12/site-packages/sklearn/neural_network/_multilayer_perceptron.py:691: C
onvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  warnings.warn(
```

Note that the model has an associated convergence warning. Lack of convergence in a neural network describes the situation when the model fails to find a solution after several iterations of training, and the model's loss function (which measures how well the model is performing) does not reduce in a meaningful way over time. This points towards the model's weights and biases not being updated effectively, and the network not learning as intended as a result.

First, we'll examine the parameters associated with the default neural network.

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 200,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': 61,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
```

activation: Defines the activation function for the hidden layers. The default value, 'relu' (Rectified Linear Unit), is a widely used function that outputs the input directly if it's positive; otherwise, it outputs zero. Other possible inputs are 'identity', 'logistic' and 'tanh' (Scikit-learn developers, n.d.-c).

alpha: This parameter helps prevent overfitting by penalising features with large weights. A higher value increases regularisation strength. The default value is 0.0001 (Scikit-learn developers, n.d.-c).

batch_size: Defines the size of mini-batches used in training. The default value, 'auto', sets it to the size of the training dataset for the 'adam' solver, or to 200 for 'sgd' solver (Scikit-learn developers, n.d.-c).

beta_1: Used in the Adam optimiser, this parameter represents the exponential decay rate for the first moment estimate. It controls the momentum for updating the model's weights. 0.9 is the default value.

beta_2: The exponential decay rate for the second moment estimate in the Adam optimiser. It helps correct the bias in the moment estimates. 0.999 is the default value (Scikit-learn developers, n.d.-c).

early_stopping: If True, the training stops early if the validation score doesn't improve for a given number of iterations (i.e. 'n_iter_no_change'). It prevents overfitting by stopping training before the model starts to memorize the data. The default value is False (Scikit-learn developers, n.d.-c).

epsilon: Used in the Adam optimiser, this parameter represents a small constant added to avoid division by zero in calculations. The default value is 1e-08 (Scikit-learn developers, n.d.-c).

hidden_layer_sizes: This specifies the number of neurons in each hidden layer. (100,) means a single hidden layer with 100 neurons. You can specify more hidden layers like (100, 50) for two layers with 100 and 50 neurons, respectively. The default value for this parameter, (100,) represents one hidden layer with 100 neurons (Scikit-learn developers, n.d.-c)

learning_rate: Defines the learning rate schedule. 'constant', the default value, means that the learning rate remains the same throughout training. Other options are 'invscaling' and 'adaptive'.
'learning_rate_init': The initial learning rate. This controls how much to change the model's weights in response to the estimated error. The default value is 0.001 (Scikit-learn developers, n.d.-c).

max_fun: This represents the number of function evaluations during optimization. This is specific to certain solvers like 'lbfgs' and 'sgd'. It may not be used if you're using the 'adam' solver. 15000 is the default value (Scikit-learn developers, n.d.-c).

max_iter: The maximum number of iterations (epochs) to train the model. If early_stopping=True, training may stop earlier if performance on the validation set doesn't improve. 'max_iter' is 200 by default; we are already getting a convergence warning with the default neural network build (Scikit-learn developers, n.d.-c).

momentum: Momentum factor used in the optimization algorithm. Helps accelerate gradient descent in the right direction. The default value used with the 'sgd' solver is 0.9 (Scikit-learn developers, n.d.-c).

n_iter_no_change: Number of iterations with no improvement on the validation score before stopping. This is used when early_stopping=True. The default value for this parameter is 10 iterations (Scikit-learn developers, n.d.-c).

nesterovs_momentum: If True (the default value when using the 'sgd' solver), the model will uses Nesterov's Accelerated Gradient (NAG), a modification of momentum that can lead to faster convergence (Scikit-learn developers, n.d.-c).

power_t: The exponent for the inverse scaling of the learning rate, which is set to 0.5 by default. This is relevant when the learning_rate is set to 'invscaling'. It controls how the learning rate changes over time (Scikit-learn developers, n.d.-c).

random_state: A random seed to ensure reproducibility of the model's training. Using the same random_state will generate the same results. The default is None, but we need to use the same random state as the decision tree and the regression modelling tasks for reproducibility purposes (Scikit-learn developers, n.d.-c).

shuffle: If True (the default value), the training data is shuffled before each epoch. This helps improve convergence.

solver: This parameter defines the optimisation algorithm to use. 'Adam' (adaptive moment estimation) is a popular choice (and the default value) due to its efficiency and ability to adapt the learning rate. 'lbfgs' (limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (Liu & Nocedal, 1989)) and 'sgd' (stochastic gradient descent) are other options for the weighting algorithm (Scikit-learn developers, n.d.-c).

tol: Tolerance for optimization. Training stops if the improvement in the loss function is less than this value, which is set to 0.0001 by default (Scikit-learn developers, n.d.-c).

validation_fraction: This parameter represents the fraction of training data to set aside as validation data for early stopping, if 'early_stopping' is set to True. The default is 0.1 (10%) of the training data (Scikit-learn developers, n.d.-c).

verbose: This parameter mediates printing out detailed progress information during training, which is useful for debugging or tracking progress. This is set to False by default (Scikit-learn developers, n.d.-c).

warm_start: If True, the model will reuse the solution of the previous fit as initialization for the next fit. This can save time if you want to add more iterations or layers. This parameter is set to false by default (Scikit-learn developers, n.d.-c).

Let's tune this model with GridSearchCV. The parameters we are aiming to tune are 'hidden_layer_sizes' and 'alpha'. (We will leave 'max_iter' for now as increasing it can blow out model build time.)

We first tuned 'hidden_layer_sizes' with neuron counts of 5-45, going up in increments of 5. This yielded the following best_params and classification report:

```
Train accuracy: 0.748025666337611
Test accuracy: 0.7006332757628094
              precision    recall  f1-score   support

           0       0.73      0.78      0.76      1036
           1       0.64      0.58      0.61       701

    accuracy                           0.70      1737
   macro avg       0.69      0.68      0.68      1737
weighted avg       0.70      0.70      0.70      1737

{'hidden_layer_sizes': (10,)}
```

Drilling down into even numbers of neurons around this value, we performed GridSearchCV parameter tuning again with ‘hidden_layer_sizes’ set as [(4,), (6,), (8,), (12,), (14,), (16,),]. This yielded a slightly better test accuracy score and a best_param of (6,).

```
Train accuracy: 0.7322309970384995
Test accuracy: 0.7023603914795624
              precision    recall  f1-score   support

           0       0.73      0.80      0.76      1036
           1       0.66      0.55      0.60       701

    accuracy                           0.70      1737
   macro avg       0.69      0.68      0.68      1737
weighted avg       0.70      0.70      0.70      1737

{'hidden_layer_sizes': (6,)}
```

We then also completed ‘alpha’ parameter tuning using ‘hidden_layer_sizes’ set as [(4,), (6,), (8,), (12,), (14,), (16,),], and ‘alpha’ as [0.01, 0.001, 0.0001, 0.00001]. This yielded a best_param of 0.00001, but test accuracy score dipped slightly. We will retain ‘alpha’ = 0.0001, which is the default value.

```
Train accuracy: 0.7319842053307009
Test accuracy: 0.7000575705238917
              precision    recall  f1-score   support

           0       0.72      0.80      0.76      1036
           1       0.65      0.55      0.60       701

    accuracy                           0.70      1737
   macro avg       0.69      0.68      0.68      1737
weighted avg       0.70      0.70      0.69      1737

{'alpha': 1e-05, 'hidden_layer_sizes': (6,)}
```

Note that the model still hasn’t converged yet.

Implementing a neural network model with dimensionality reduction via RFE (a reduction in features from 44 to 32, much like the other two methods of predictive analysis completed) shows a very slight improvement in test accuracy. Note that best_params have changed again.

```
Train accuracy: 0.7532082922013821
Test accuracy: 0.7040875071963155
              precision    recall  f1-score   support

           0       0.74      0.78      0.76      1036
           1       0.65      0.58      0.61       701

    accuracy                           0.70      1737
   macro avg       0.69      0.68      0.69      1737
weighted avg       0.70      0.70      0.70      1737

{'alpha': 1e-05, 'hidden_layer_sizes': (16,)}
```

A further neural network model was created, using the features from the optimal decision tree, and GridSearchCV used to determine optimal ‘alpha’ and ‘hidden_layer_sizes’. The test accuracy remains slightly lower than previous, however.

None of these models are converging.

There does not seem to be obvious overfitting in the selected variables neural network - we are not seeing a 100% match between the model and the training data in either instance.

Tuning ‘max_iter’ for this selected features neural network model:

```
Train accuracy: 0.6971865745310958
Test accuracy: 0.7000575705238917
              precision    recall  f1-score   support

           0       0.76      0.73      0.74      1036
           1       0.62      0.65      0.64       701

    accuracy                           0.70      1737
   macro avg       0.69      0.69      0.69      1737
weighted avg       0.70      0.70      0.70      1737

{'alpha': 1e-05, 'hidden_layer_sizes': (9,), 'max_iter': 300}
```

This model is now converging.

Tuning ‘max_iter’ for the RFE neural network:

```
Train accuracy: 0.7268015794669299
Test accuracy: 0.7023603914795624
              precision    recall  f1-score   support

           0       0.74      0.78      0.76      1036
           1       0.64      0.59      0.61       701

    accuracy                           0.70      1737
   macro avg       0.69      0.68      0.69      1737
weighted avg       0.70      0.70      0.70      1737

{'alpha': 0.01, 'hidden_layer_sizes': (8,), 'max_iter': 600}
```

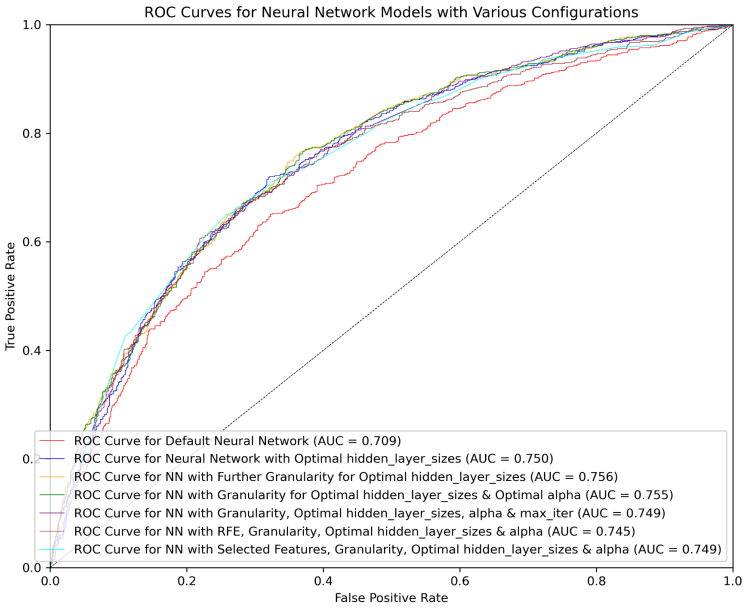
This has reduced test accuracy slightly, so we will use the original RFE neural network model.

The 32 inputs for the RFE model in order of importance are as follows:

'contacts_count': 6.360312421773421
'smoking_never': 4.9199120009527935
'age_0_10': 4.813758836678143
'age_40_50': 4.387861759007465
'age_60_70': 4.176566401043816
'age_70_80': 4.169100623940956
'height': 3.9105875045382517

'blood_type_abn': 3.8503374510143358
'income_gov': 3.8008113388025535
'blood_type_abp': 3.763315509350938
'insurance': 3.7582415978812542
'age_20_30': 3.728983430342602
'blood_type_unknown': 3.5924927718572457
'blood_type_bn': 3.512667381774999
'gender_female': 3.3724190968409165
'blood_type_an': 3.3677133073633883
'income_med': 3.317385277713913
'income_low': 3.2926508922380333
'income_high': 3.241886861792721
'blood_type_ap': 3.1229987620244835
'weight': 3.043854289480673
'gender_male': 3.005207099300279
'blood_type_op': 2.9885558655790634
'worried': 2.7791204260968256
'age_30_40': 2.7633718362220723
'blood_type_bp': 2.6951347722029215
'age_100_110': 2.525011552388502
'alcohol': 2.497156814735663
'blood_type_on': 2.4695812659820993
'age_50_60': 2.4581453199950674
'age_10_20': 2.3560380378232333
'gender_other': 2.0279116946805127

Developing the ROC curve for the generated neural networks:



Most models hover around the same area, with the exception of the default NN model, which makes sense. The best model by a tiny margin is the neural network with further granularity for optimal 'hidden_layer_sizes'.

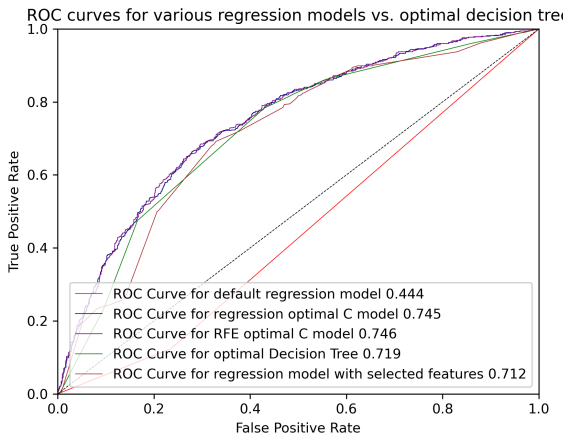
Characteristics of COVID-19 positive individuals in this model can be determined by looking at the list of coefficients for variables, ordered by feature importance:

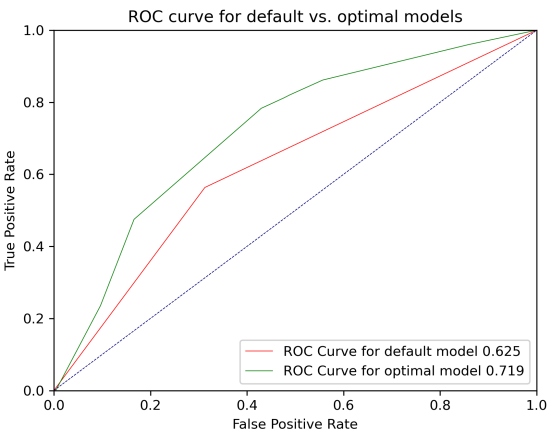
```
Feature: income_med, Importance: 0.04242947610823254
Feature: weight, Importance: 0.023028209556706904
Feature: worried, Importance: 0.021013241220495072
Feature: height, Importance: 0.015486470926885399
Feature: working_never, Importance: 0.015371329879101859
Feature: income_high, Importance: 0.013413932066781754
Feature: insurance, Importance: 0.01151410477835343
Feature: smoking_never, Importance: 0.010477835348301623
Feature: age_70_80, Importance: 0.008059873344847378
Feature: gender_male, Importance: 0.007656879677605022
Feature: age_60_70, Importance: 0.006044905008635503 5
Feature: contacts_count, Importance: 0.005929763960852019
Feature: smoking_quit5, Importance: 0.005469199769717858
Feature: income_low, Importance: 0.005354058721934318
Feature: age_20_30, Importance: 0.00506620610247549
Feature: working_home, Importance: 0.004663212435233122
Feature: working_travel non critical, Importance: 0.004490500863557812
Feature: gender_female, Importance: 0.0038572251007483517
Feature: blood_type_bp, Importance: 0.003223949337938936
Feature: alcohol, Importance: 0.003166378814047166
Feature: working_stopped, Importance: 0.003108808290155374
Feature: blood_type_ap, Importance: 0.0025906735751294874
Feature: age_30_40, Importance: 0.002590673575129476
Feature: blood_type_unknown, Importance: 0.0024179620034541992
Feature: smoking_yesheavy, Importance: 0.002360391479562407
Feature: blood_type_bn, Importance: 0.0020725388601035787
Feature: smoking_yeslight, Importance: 0.00201496833621182
Feature: blood_type_op, Importance: 0.0016119746689694403
Feature: income_gov, Importance: 0.0014392630972941522
Feature: blood_type_abp, Importance: 0.0012089810017270719
Feature: blood_type_abn, Importance: 0.0009786989061599915
Feature: age_10_20, Importance: 0.0008635578583764513
Feature: age_50_60, Importance: 0.0008059873344847146
Feature: age_0_10, Importance: 0.0008059873344846702
Feature: gender_other, Importance: 5.757052389173678e-05
Feature: age_100_110, Importance: -3.3306690738754695e-17
Feature: smoking_quit0, Importance: -0.00011514104778357349
Feature: blood_type_an, Importance: -0.00011514104778359568
Feature: smoking_vape, Importance: -0.00034542314335064275
Feature: smoking_yesmedium, Importance: -0.0009211283822683103
Feature: working_travel critical, Importance: -0.0017271157167530692
Feature: smoking_quit10, Importance: -0.0017846862406448282
Feature: age_40_50, Importance: -0.0017846862406448282
Feature: blood_type_on, Importance: -0.0020149683362118975
```

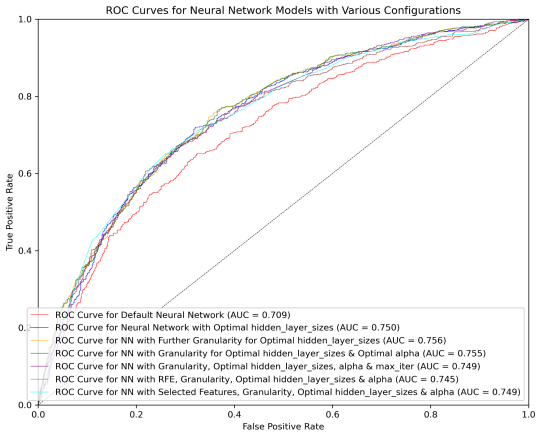
Highly positive values for a feature correlate with likelihood of Covid positivity whereas lower values may be indicative of higher likelihood of Covid-19 negativity. It is not possible to determine exactly which features are which.

Task 4: Final Remarks

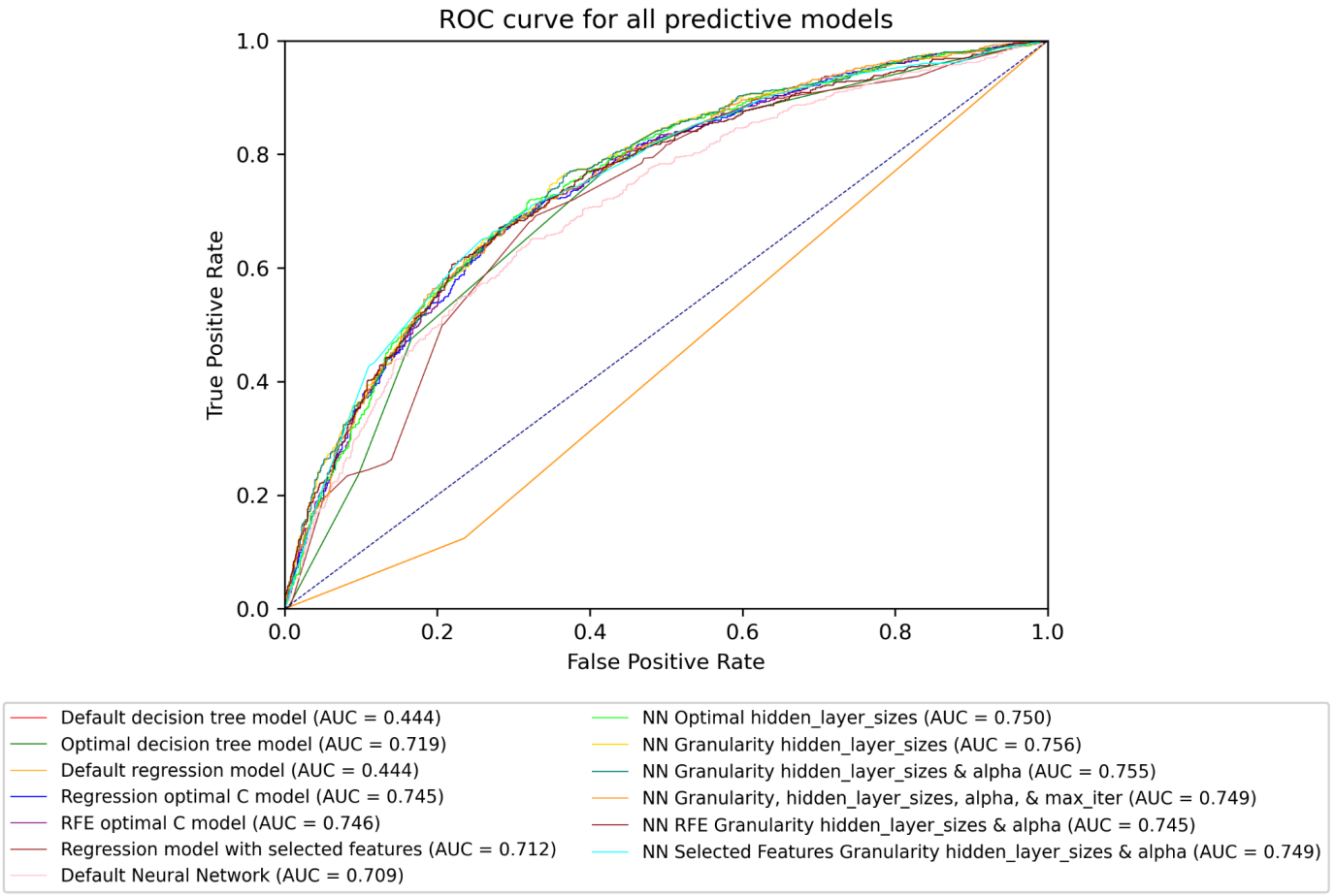
After obtaining the 3 optimal predictive models (Decision tree, Regression model and Neural network), we are going to compare their results. Again, here is the ROC curve for 3 types accordingly:







Here are all of the predictive models we built plotted on the same axes:



The best model out of these remains the neural network model with the further granularity in 'hidden_layer_sizes'.

	Decision Trees (optimal)	Regression Models (optimal)	Neural Networks (optimal)
Accuracy score	0.64	0.69	0.69
ROC index	0.719	0.746	0.756

The above table aggregated the metric information of the 3 types of predictive models we did in Task 3. In normal observation, we can see that the Regression Model and Neural Network models have an approximate accuracy (~0.69), and outperform the Decision Tree with a value of 0.64. For the ROC AUC index, the Neural Network model showed it has the highest value (0.756) while the Decision Tree and Regression models have lower values. As a result, we will use the Neural Network model in decision-making, because it remained the best model overall in terms of two metrics (Accuracy core and ROC index). However, the ROC index is a much preferred and better metric for comparison of model performance (Shiksha, n.d.).

Factor	Decision Trees	Regression Models	Neural Networks
Interpretability	High (easy to visualize and understand)	Moderate (coefficients can be interpreted but may require domain knowledge)	Low (often seen as a "black-box" model)
Handling of Non-linearity	Handles non-linear relationships well	Struggles with non-linear data unless using polynomial terms or transformations	Excellent
Data Type	Works with both categorical and continuous data	Primarily used for continuous data	Works well with both categorical and continuous data
Feature Scaling	Not required	Required	Required
Overfitting Risk	High (especially without pruning)	Low to moderate	High
Computation Time	Fast to train, especially on smaller datasets	Fast to train (especially linear regression)	Often computationally expensive
Model Complexity	Can become very complex and hard to interpret with deep trees	Generally simpler, but may need multiple models or transformations for complex data	Can be very complex as the number of internal layers increase, making it

			flexible but harder to fine-tune
Hyperparameter Tuning	Requires tuning of tree depth, min samples, and pruning options	Requires some parameter tuning	Requires some parameter tuning
Outliers Sensitivity	Sensitive to outliers, as they can significantly alter the tree structure	Sensitive to outliers	Less sensitive to outliers
Training Time	Generally fast for smaller datasets, but slower for larger datasets	Fast for small to medium-sized datasets, but can slow with larger datasets	Can be slow to train, especially for deep neural networks and large datasets
Prediction Speed	Fast (once the tree is built)	Fast	Slower
Feature Importance	Clear and easy to interpret (based on tree structure and split criteria)	Interpretable via coefficients (though may require domain-specific understanding)	Difficult to interpret directly (weights and activations are not easily interpretable)
Handling Missing Data	Handles missing values well with imputation or splitting on missing values	Requires preprocessing to handle missing data (e.g. imputation)	Can handle missing data if the network is trained to do so or with appropriate preprocessing
Performance on Large Datasets	Performs well with small to medium-sized datasets; struggles with very large datasets without optimization	Performs well on medium-sized datasets but can struggle with very large datasets	Performs well with large datasets, especially with higher computational power
Applicability	Great for classification, especially when feature importance is needed	Primarily used for regression tasks, good for simple models and baseline comparisons	Best for complex tasks like image recognition, natural language processing, and highly non-linear problems

Based on the above factors mentioned, we can make the summary of both positive and negative sides for each predictive model as follows:

	Decision Trees	Regression Models	Neural Networks
Positive	• Interpretability • Handling of Non-linearity • Data Type • Handling Missing Data • Computation Time • Feature Importance • Training Time • Prediction Speed	• Interpretability • Computation Time • Training Time • Prediction Speed • Outliers Sensitivity • Good Accuracy • Model Complexity	• Data Type • Performance on Large Datasets • Outliers Sensitivity • ROC index • Good Accuracy • Applicability
Negative	• Overfitting Risk • Performance on Large Datasets • Outliers Sensitivity • Applicability • ROC index • Good Accuracy	• Feature Importance • Applicability • Handling of Non-linearity	• Overfitting Risk • Feature Importance • Training Time • Prediction Speed • Hyperparameter Tuning • Computation Time

Conclusion

There are benefits and drawbacks to using predictive models like decision trees, regression modelling and neural networks - it is important to choose the most appropriate analysis method for your usecase, to pre-process the data appropriately, and to tune the parameters passed into the various predictive model functions instead of using the default version.

References

MLJAR. (n.d.). Extract rules from decision tree. https://mljar.com/blog/extract-rules-decision-tree/

Scikit-learn developers. (n.d.). Sklearn tree Decision Tree Classifier Scikit-learn documentation. https://scikit-learn.org/1.5/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.get_params

Madaan, A. (2020, October 8). Cost complexity pruning in decision trees. https://www.analyticsvidhya.com/blog/2020/10/cost-complexity-pruning-decision-trees/

Scikit-learn developers. (n.d.). Sklearn tree Decision Tree Classifier Scikit-learn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Scikit-learn developers. (n.d.). MLPClassifier — scikit-learn 1.3.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Liu, D. C., & Nocedal, J. (1989). On the limited memory method for large scale optimization. Mathematical Programming, B, 45(3), 503–528. https://doi.org/10.1007/BF01589116

Shiksha. (n.d.). ROC-AUC vs accuracy. https://www.shiksha.com/online-courses/articles/roc-auc-vs-accuracy/