Task 1 – Implementing Forward Selection Method

The forward selection method simply compares a column at a time to see which could contribute to the final score the better, in each iteration a column is added to the list of the selected features until the desired amount of iterations is reached

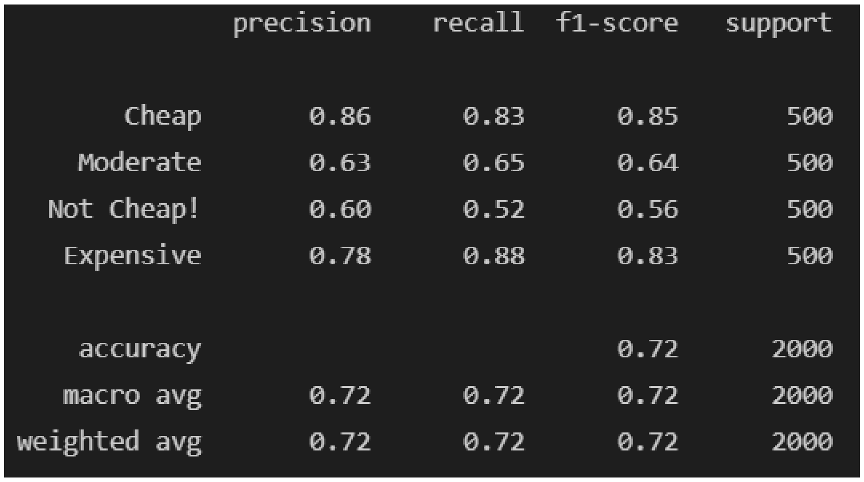
How to implement it is self-explanatory, a for loop to iterate over features and a function to evaluate the score of a Random Forest Classifier (I used a random forest as it requires almost no data preprocessing) stores the results and finds the feature that could improve the model the best, this process is repeated next time using the selected features so far plus the feature we're evaluating. It is implemented using 2 functions, one for the iterative part, the other to evaluate each feature contribution using an AUC metric (Area Under The Curve Of The ROC)

We'll use NUM_ITERATION of 5, this is the 5 features selected in order (also I didn't bin the price range into 2 ranges, which would yield almost 100 accuracies and make it harder for future comparisons of the models)

1. ram
2. px_height
3. blue
4. dual_sim
5. touch_screen

Task 2 – Evaluating Selected Features of the Forward Selection Method

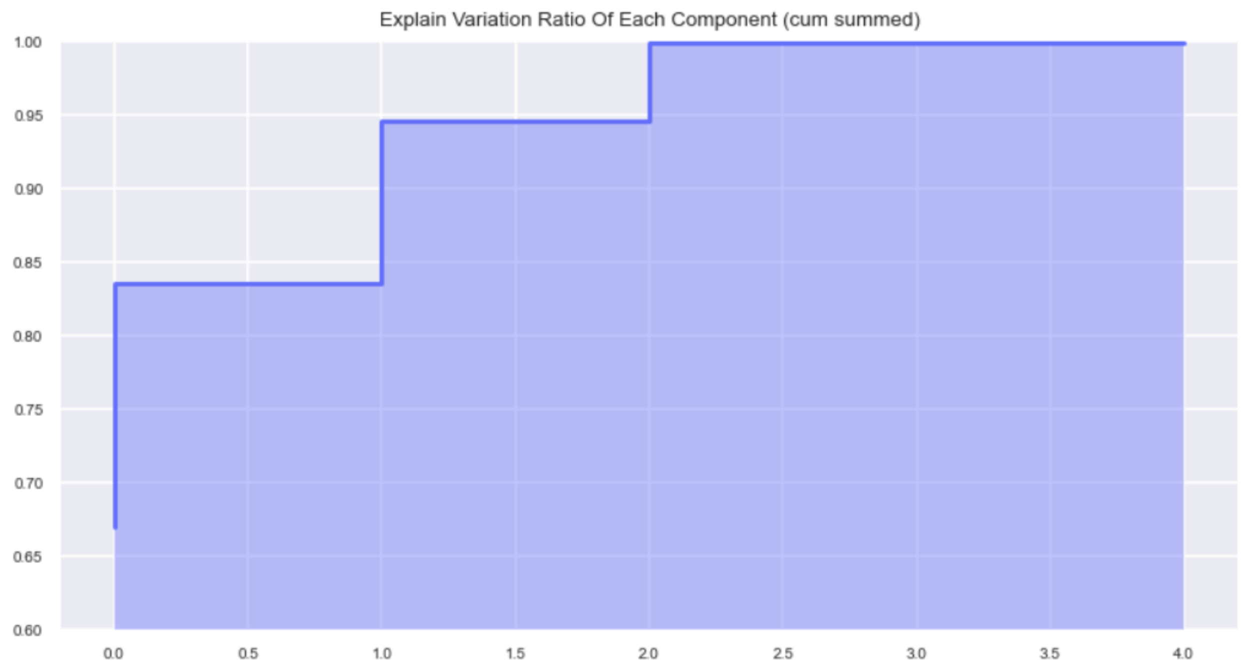
This is the final output of the logistic classifier model using the 5 features seen above, later on, we'll see comparisons towards other models such as SVMs or Decision Trees



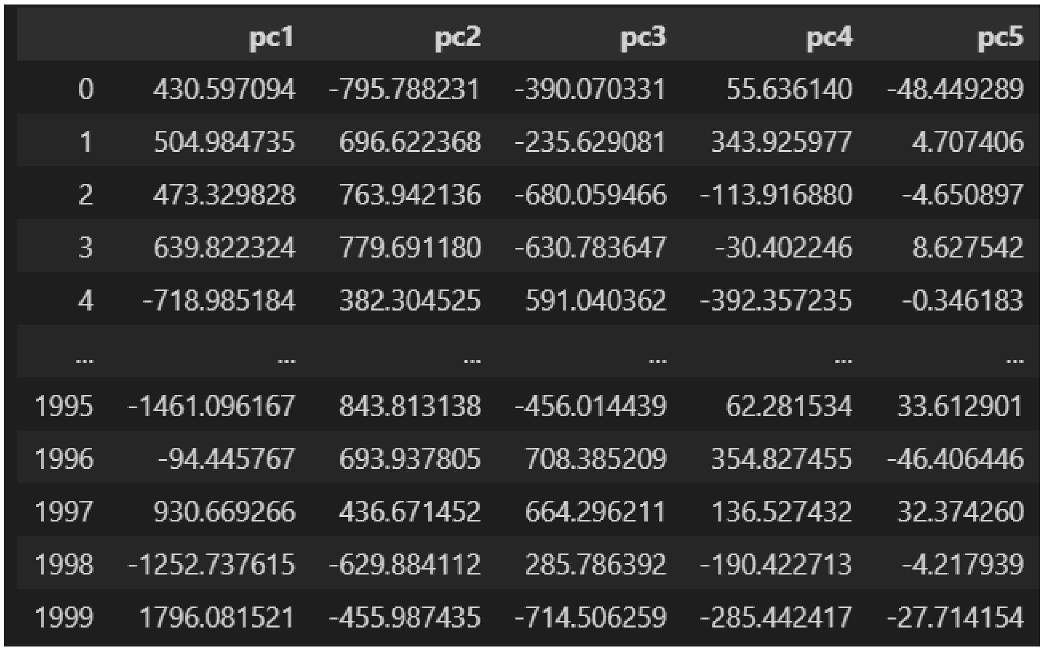
```
              precision    recall  f1-score   support

       Cheap       0.86      0.83      0.85       500
    Moderate       0.63      0.65      0.64       500
  Not Cheap!       0.60      0.52      0.56       500
   Expensive       0.78      0.88      0.83       500

    accuracy                           0.72      2000
   macro avg       0.72      0.72      0.72      2000
weighted avg       0.72      0.72      0.72      2000
```

Task 3 – Using PCA to Extract 5 components

It seems only 5 components could capture almost all the explainability of our data, here is the cumulative sum of the explain-variance-ratio of each component

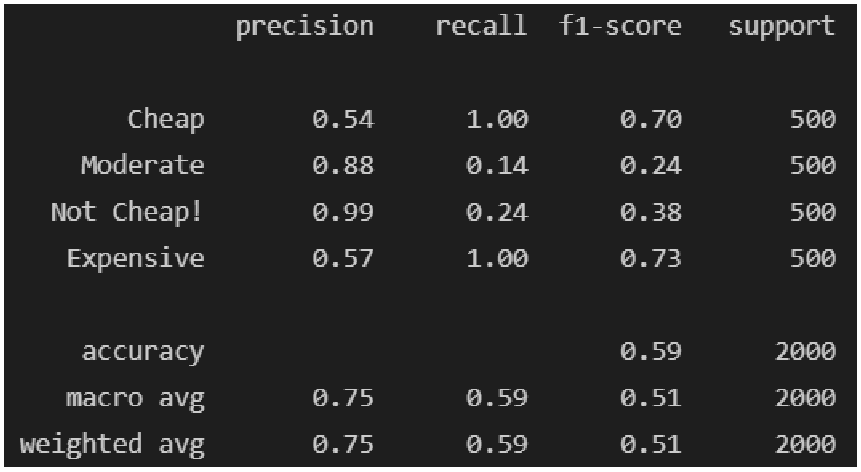


This is the final look at the data



	pc1	pc2	pc3	pc4	pc5
0	430.597094	-795.788231	-390.070331	55.636140	-48.449289
1	504.984735	696.622368	-235.629081	343.925977	4.707406
2	473.329828	763.942136	-680.059466	-113.916880	-4.650897
3	639.822324	779.691180	-630.783647	-30.402246	8.627542
4	-718.985184	382.304525	591.040362	-392.357235	-0.346183
...	...	...	...	...	...
1995	-1461.096167	843.813138	-456.014439	62.281534	33.612901
1996	-94.445767	693.937805	708.385209	354.827455	-46.406446
1997	930.669266	436.671452	664.296211	136.527432	32.374260
1998	-1252.737615	-629.884112	285.786392	-190.422713	-4.217939
1999	1796.081521	-455.987435	-714.506259	-285.442417	-27.714154

Task 4 – Using Logistic Regression on the PCA data



```
              precision    recall  f1-score   support

       Cheap       0.54      1.00      0.70       500
    Moderate       0.88      0.14      0.24       500
  Not Cheap!       0.99      0.24      0.38       500
   Expensive       0.57      1.00      0.73       500

    accuracy                           0.59      2000
   macro avg       0.75      0.59      0.51      2000
weighted avg       0.75      0.59      0.51      2000
```

Task 5 – SVM Kernels, What they are and how do they do it

The function of a kernel is to take data as input and transform it into the required form.

Essentially a kernel refers to a method that allows us to apply linear classifiers to non-linear problems by mapping non-linear data into a higher-dimensional space without the need to visit or understand that higher-dimensional space.

These functions can be different types. For example

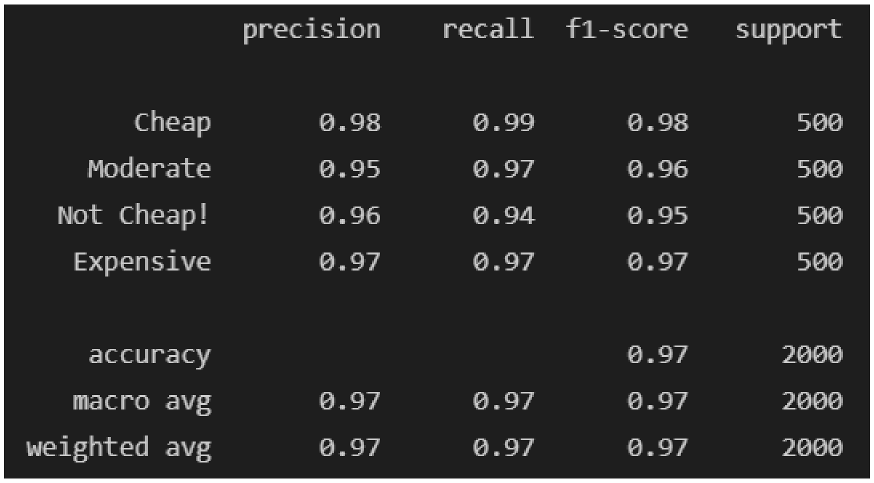
a) linear
b) sigmoid
$$k(x, y) = \tanh(\alpha x^T y + c)$$
c) polynomial
$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$
d) radial basis function (RBF)
$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$
e) hyperbolic tangent kernel
$$k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa \mathbf{x_i} \cdot \mathbf{x_j} + c)$$

The most used type of kernel function is RBF. Because it has localized and finite responses along the entire x-axis.

The kernel functions return the inner product between two points in a suitable feature space. Thus by defining a notion of similarity, with a little computational cost even in very high-dimensional spaces.

Task6 – Implementing SVM:

For this part, a simple SVM model without any hyper-parameter-tuning is used, though we'll put it inside a pipeline containing a Standard Scaler as the SVM model wouldn't work properly with unscaled data, from now on we'll use this pipeline for Tasks 7, 8 and 9.

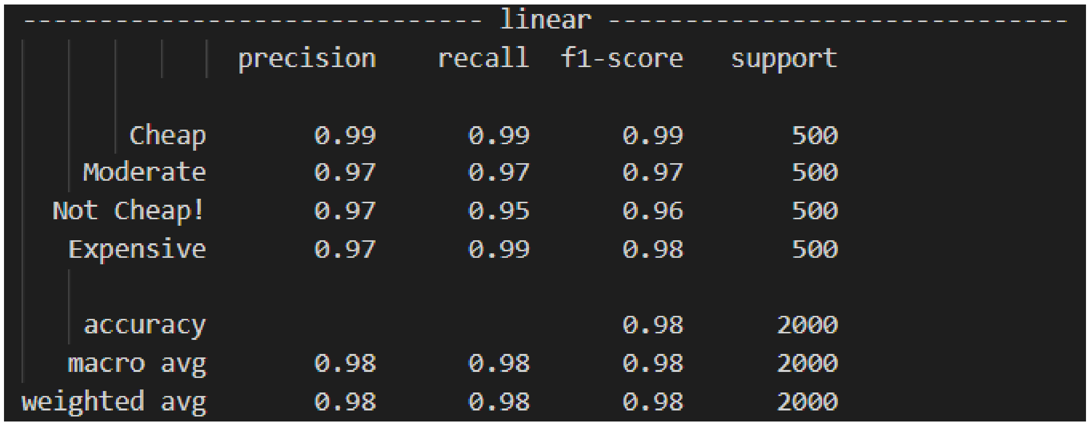


```
              precision    recall  f1-score   support

       Cheap       0.98      0.99      0.98       500
    Moderate       0.95      0.97      0.96       500
  Not Cheap!       0.96      0.94      0.95       500
   Expensive       0.97      0.97      0.97       500

    accuracy                           0.97      2000
   macro avg       0.97      0.97      0.97      2000
weighted avg       0.97      0.97      0.97      2000
```

Task 7 – Using Different SVM and Parameters kernels

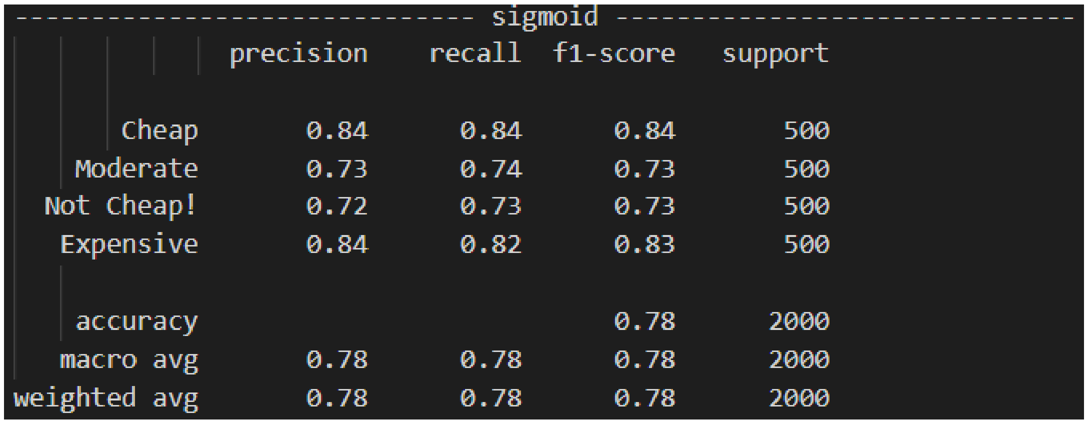
I have used this set of parameters for the comparisons

a) Kernel = linear



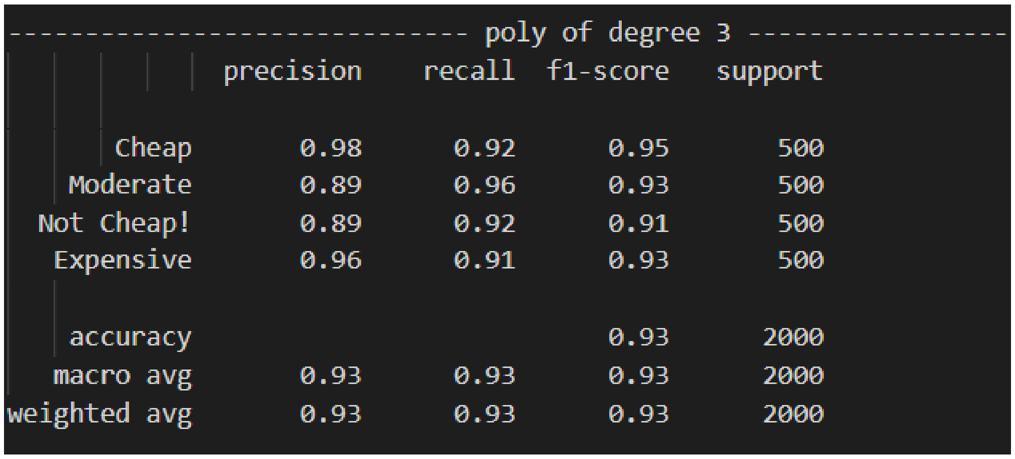
```
------------------------------ linear ------------------------------
              precision    recall  f1-score   support

       Cheap       0.99      0.99      0.99       500
    Moderate       0.97      0.97      0.97       500
  Not Cheap!       0.97      0.95      0.96       500
   Expensive       0.97      0.99      0.98       500

    accuracy                           0.98      2000
   macro avg       0.98      0.98      0.98      2000
weighted avg       0.98      0.98      0.98      2000
```

b) Kernel = sigmoid



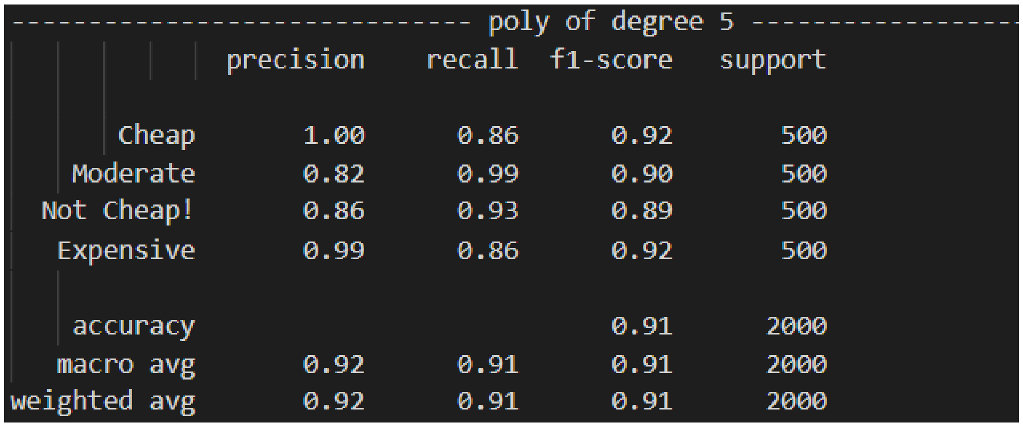
```
------------------------------ sigmoid ------------------------------
              precision    recall  f1-score   support

       Cheap       0.84      0.84      0.84       500
    Moderate       0.73      0.74      0.73       500
  Not Cheap!       0.72      0.73      0.73       500
   Expensive       0.84      0.82      0.83       500

    accuracy                           0.78      2000
   macro avg       0.78      0.78      0.78      2000
weighted avg       0.78      0.78      0.78      2000
```

c) Kernel = poly, degree of polynomiality = 3



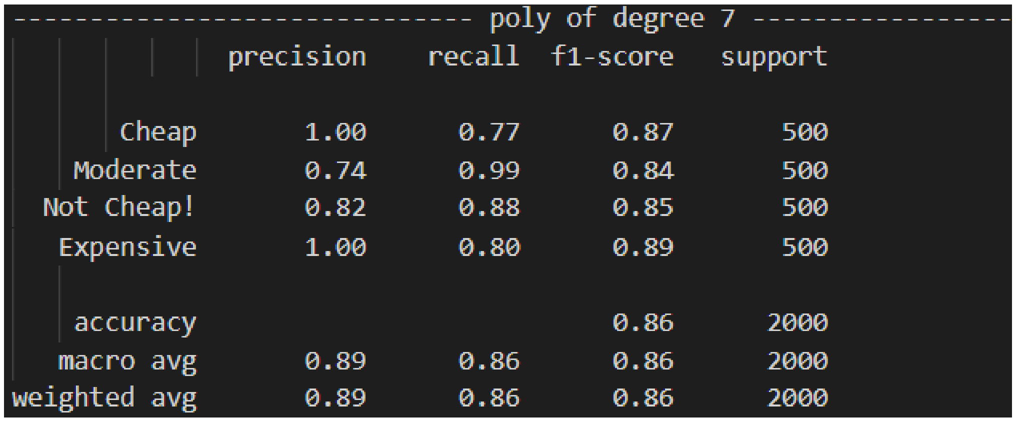
```
------------------------------ poly of degree 3 -----------------
              precision    recall  f1-score   support

       Cheap       0.98      0.92      0.95       500
    Moderate       0.89      0.96      0.93       500
  Not Cheap!       0.89      0.92      0.91       500
   Expensive       0.96      0.91      0.93       500

    accuracy                           0.93      2000
   macro avg       0.93      0.93      0.93      2000
weighted avg       0.93      0.93      0.93      2000
```

d) Kernel = poly, degree of polynomiality = 5



```
------------------------------ poly of degree 5 -----------------
              precision    recall  f1-score   support

       Cheap       1.00      0.86      0.92       500
    Moderate       0.82      0.99      0.90       500
  Not Cheap!       0.86      0.93      0.89       500
   Expensive       0.99      0.86      0.92       500

    accuracy                           0.91      2000
   macro avg       0.92      0.91      0.91      2000
weighted avg       0.92      0.91      0.91      2000
```

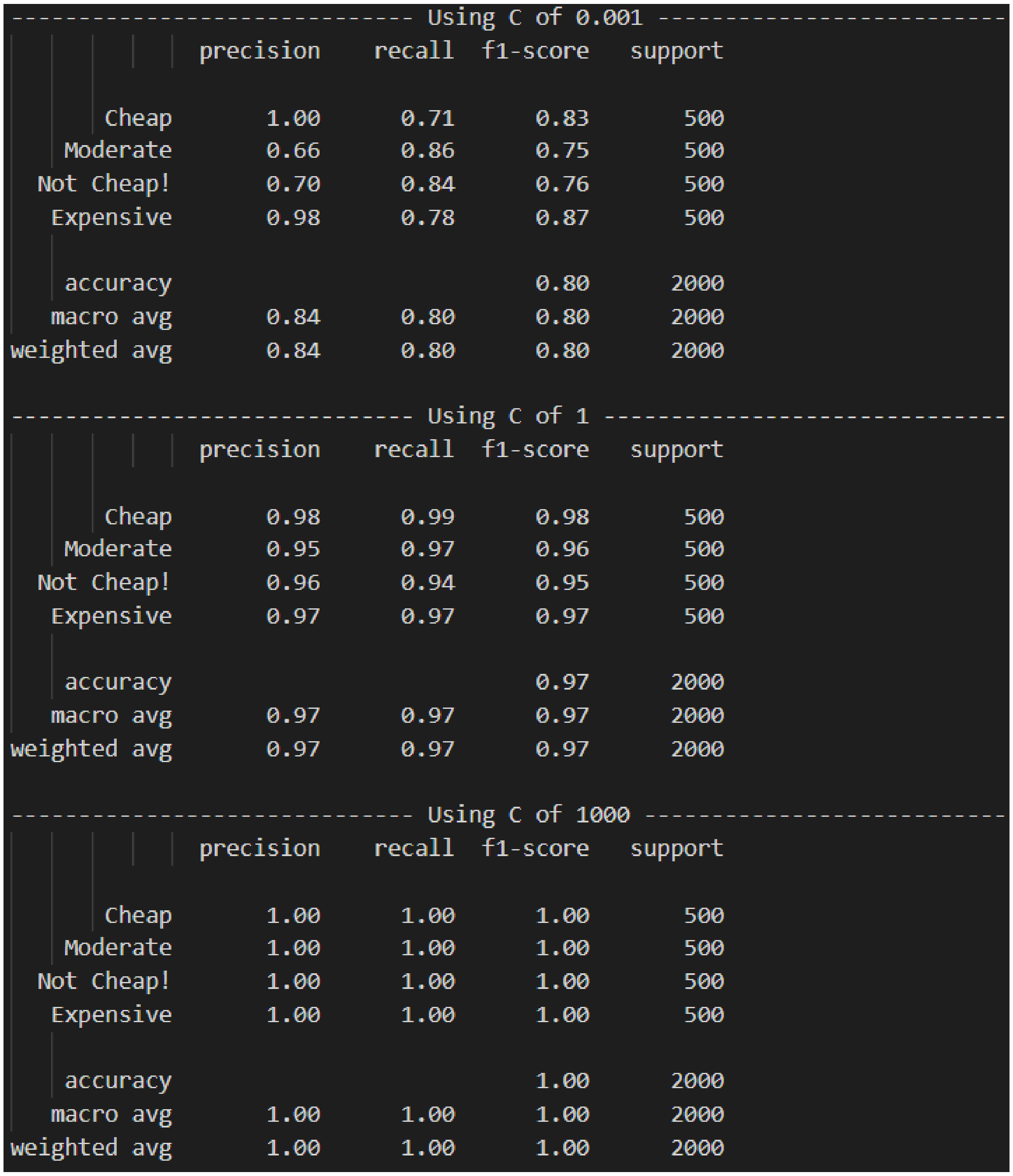
e) Kernel = poly, degree of polynomiality = 7



```
------------------------------ poly of degree 7 -----------------
              precision    recall  f1-score   support

       Cheap       1.00      0.77      0.87       500
    Moderate       0.74      0.99      0.84       500
  Not Cheap!       0.82      0.88      0.85       500
   Expensive       1.00      0.80      0.89       500

    accuracy                           0.86      2000
   macro avg       0.89      0.86      0.86      2000
weighted avg       0.89      0.86      0.86      2000
```

Task 8 - soft hard margins:

we can set the C parameter in higher numbers to achieve what we call harder margins (eventually some value like 1e10 could be the theoretical hard margin)



```
------------------------------ Using C of 0.001 --------------------------
              precision    recall  f1-score   support

       Cheap       1.00      0.71      0.83       500
    Moderate       0.66      0.86      0.75       500
  Not Cheap!       0.70      0.84      0.76       500
   Expensive       0.98      0.78      0.87       500

    accuracy                           0.80      2000
   macro avg       0.84      0.80      0.80      2000
weighted avg       0.84      0.80      0.80      2000

------------------------------ Using C of 1 ------------------------------
              precision    recall  f1-score   support

       Cheap       0.98      0.99      0.98       500
    Moderate       0.95      0.97      0.96       500
  Not Cheap!       0.96      0.94      0.95       500
   Expensive       0.97      0.97      0.97       500

    accuracy                           0.97      2000
   macro avg       0.97      0.97      0.97      2000
weighted avg       0.97      0.97      0.97      2000

------------------------------ Using C of 1000 ---------------------------
              precision    recall  f1-score   support

       Cheap       1.00      1.00      1.00       500
    Moderate       1.00      1.00      1.00       500
  Not Cheap!       1.00      1.00      1.00       500
   Expensive       1.00      1.00      1.00       500

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

Task9 – Feature Engineering

a) For the binning section, I used 4 bins with this value distribution

Bin 1 contains values that fall below the 40th percentile and the rest of the bins are equally spaced using 20th percentile each

b) There were no categorical features in the dataset!

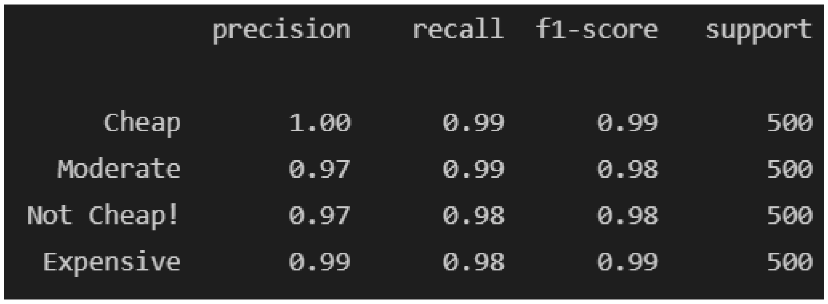
c) Aside from log transformation which could come in handy when we are dealing with a feature that contains very large values (and sparse) so the log transformation could smooth the values out and reduce the scarcity even, we can use a transformation called box-cox transformation, this transformation comes in handy when we're using linear models, it also changes the distribution of the feature, making it more Gaussian-like as some models could benefit from such transformation generally we'd want to use this transformation when we're dealing with non-homoscedastic (heteroskedasticity) data. homoscedasticity means a situation in which the variance of the dependent variable is the same for all the data.

d) For creating the area of the screen we'll use the product of px-width and px-height

Task10 – SVM on Feature Engineered Data

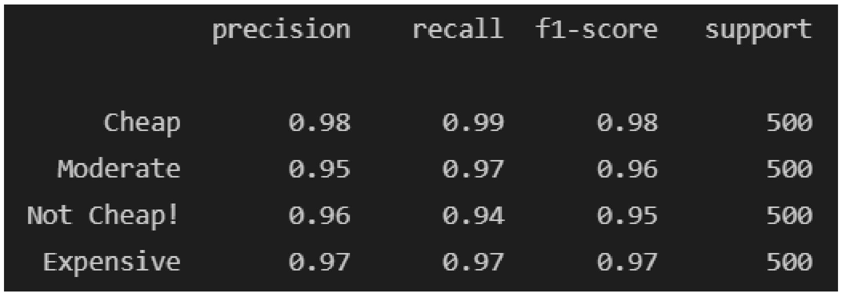
An improvement is clearly visible in all the classes

Using feature engineered data:



```
              precision    recall  f1-score   support

       Cheap       1.00      0.99      0.99       500
    Moderate       0.97      0.99      0.98       500
  Not Cheap!       0.97      0.98      0.98       500
   Expensive       0.99      0.98      0.99       500
```

Using Normal data:



```
              precision    recall  f1-score   support

       Cheap       0.98      0.99      0.98       500
    Moderate       0.95      0.97      0.96       500
  Not Cheap!       0.96      0.94      0.95       500
   Expensive       0.97      0.97      0.97       500
```

Task11 – Decision Trees Algorithms

ID3 (Iterative Dichotomies 3)

The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalize to unseen data.

C4.5

C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. The accuracy of each rule is then evaluated to determine the order in which it should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.

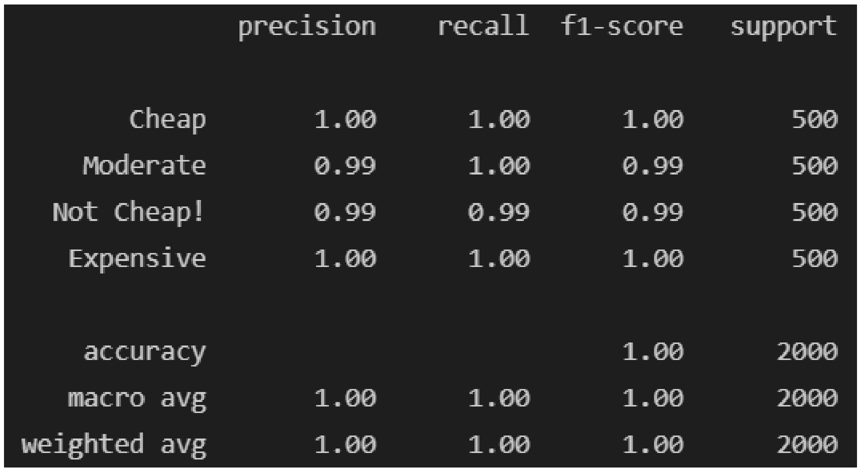
CART (Classification and Regression Trees)

CART is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yields the largest information gain at each node.

Task 12 – Implementing a Decision Tree

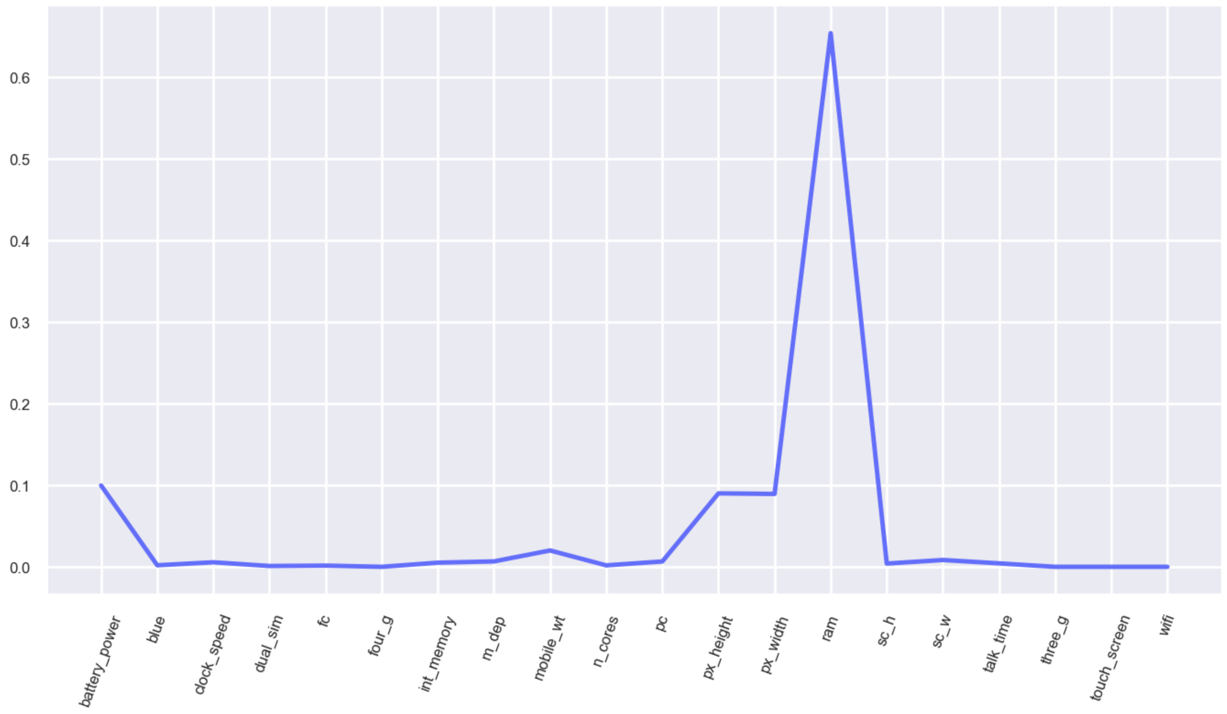
For this cause we won't need the scaled data so we can skip the standard scaler in the pipeline, here are the final results which show improvement on the previous models meaning the decision trees came in handy the best in our dataset (this usually happens when a couple of features contribute to the final result more than the sum of the contribution of other models if we're given a data set which all the feature contribute to the score to some mediocre scale, other models could prove to be more accurate than trees)

Here are the results



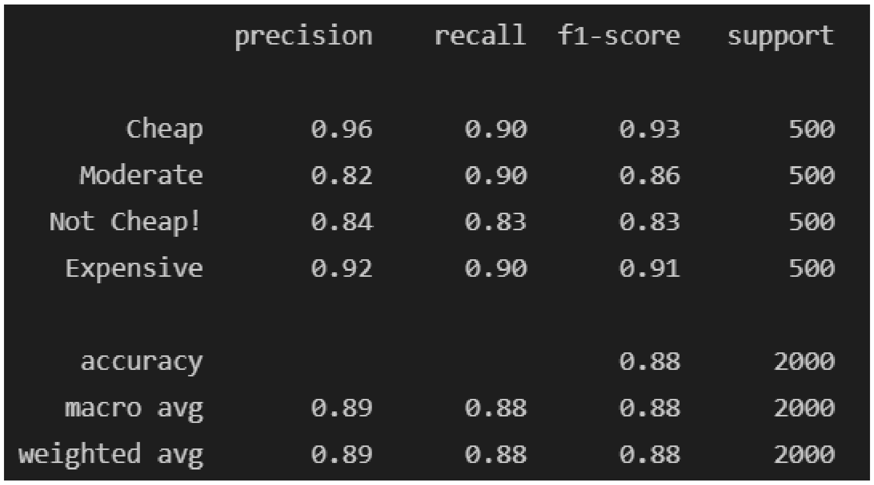
```
              precision    recall  f1-score   support

       Cheap       1.00      1.00      1.00       500
    Moderate       0.99      1.00      0.99       500
  Not Cheap!       0.99      0.99      0.99       500
   Expensive       1.00      1.00      1.00       500

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

And here's the feature importance of each feature



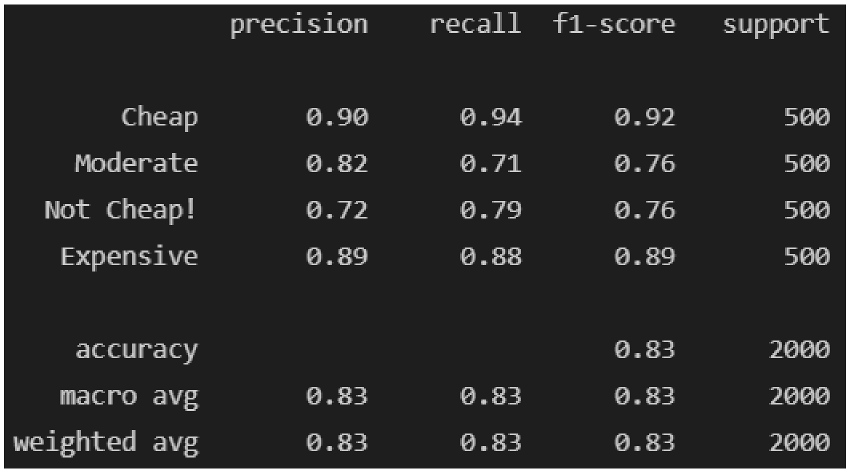
Task 13 – Decision Tree Parameters Comparison

The most optimal values for the tree were the default ones on the min_samples_leaf = 2 and min_sample_split = 1, though to get more accurate results we can set the depth to a high number (though the default version is -1 which refers to unlimited depth) so we can lower the depth to see it's the effect on how it reduces performance, the first plot still uses 10 depth (like the results on the previous task) but with the min_samples_leaf of 20 and min_sample_split of 10, which definitely reduced the accuracy, the second scores have the depth of 4 in advance



```
              precision    recall  f1-score   support

       Cheap       0.96      0.90      0.93       500
    Moderate       0.82      0.90      0.86       500
  Not Cheap!       0.84      0.83      0.83       500
   Expensive       0.92      0.90      0.91       500

    accuracy                           0.88      2000
   macro avg       0.89      0.88      0.88      2000
weighted avg       0.89      0.88      0.88      2000
```

Depth = 4



```
              precision    recall  f1-score   support

       Cheap       0.90      0.94      0.92       500
    Moderate       0.82      0.71      0.76       500
  Not Cheap!       0.72      0.79      0.76       500
   Expensive       0.89      0.88      0.89       500

    accuracy                           0.83      2000
   macro avg       0.83      0.83      0.83      2000
weighted avg       0.83      0.83      0.83      2000
```

Task 14 – Pruning

Pruning a decision tree helps to prevent overfitting the training data so that our model generalizes well to unseen data. Pruning a decision tree means removing a redundant subtree and not a useful split and replacing it with a leaf node.

Decision tree pruning can be divided into two types: pre-pruning and post-pruning.

Pre pruning is when we specify a limit to which we could go deeper according to the minimum number of samples found in each leaf, also we could limit the minimum number of samples that could divide a node to its children, these are implemented using the min_sample_leaf & min_sample_split

Post pruning is when we'll prune the tree after it has been created with no limitation on those discussed above, usually, this approach is not efficient as for specific datasets the tree could be so much costly to build without using pre pruning

Task 15 – Bootstrapping vs Cross-Validation

Both cross-validation and bootstrapping are *resampling* methods.

- bootstrap resamples with replacement (and usually produces new "surrogate" data sets with the same number of cases as the original data set). Due to the drawing with replacement, a bootstrapped data set may contain multiple instances of the same original cases, and may completely omit other original cases.
- Cross-validation resamples without replacement and thus produces surrogate data sets that are smaller than the original.
- As the name cross-validation suggests, its primary purpose is measuring the (generalization) performance of a model. In the contrast, bootstrapping is primarily used to establish empirical distribution functions for a wider range of statistics (widespread as in ranging from, say, the variation of the mean to the variation of models in bagged ensemble models).
- The leave-one-out analog of the bootstrap procedure is called *jackknifing* (and is older than bootstrapping).
- The bootstrap analog to cross-validation estimates of generalization error is called the *out-of-bootstrap* estimate (because the test cases are those that were left out of the bootstrap resampled training set).

With a similar total number of evaluated surrogate models,

total error [of the model prediction error measurement] is similar, although bootstrapping typically has more bias and less variance than the corresponding CV estimates.

To Sum up, cross-validation and bootstrap tell us different things. Cross-validation gives the cross-validation estimate for assessing how well a model can perform on new data. Also, you can evaluate which model is superior to others with the cross-validation estimate. On the other hand, the bootstrap tells how accurate a sample statistic is compared to the true population statistic. You could also set coefficients as sample statistics to assess model accuracy with the bootstrap. Therefore, choosing which method to use between cross-validation and bootstrap depends on the purpose of your study.

Task 16 – 5*2 Cross-Validation

It refers to a 5 repetition of a 2-fold.

2 repetitions in the outer loop mean that you repeat your 5-fold CV 2 times on the whole train set. Each time subdivision into folds will be different.

This is mainly used for better estimations of model performance, like running statistical tests on whether one model performs statistically significantly better than another.

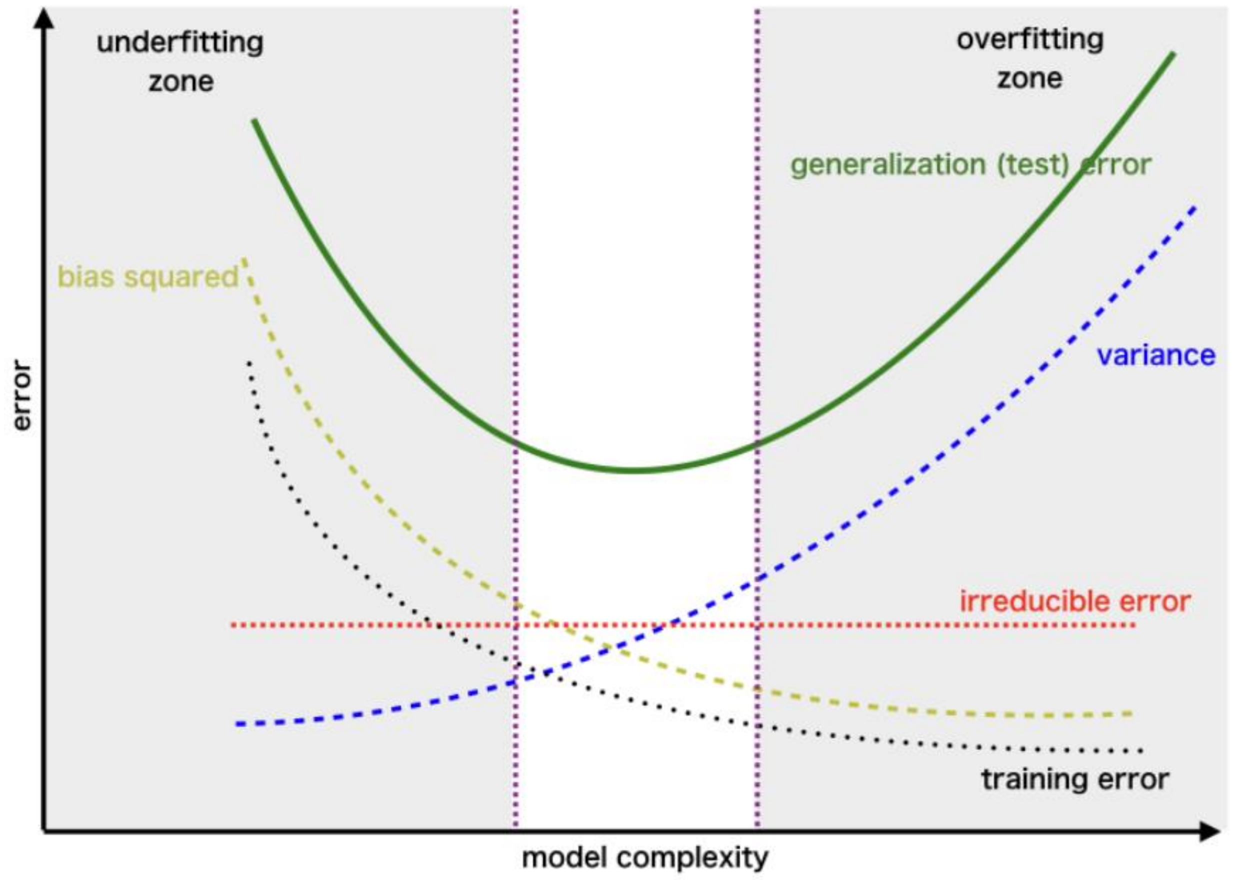
A nested CV is not critically important if your data set is large and without outliers. If your data do have outliers, then cross-validation performance may be drastically different depending on what fold/folds these outliers are in. Therefore you repeat CV several times.

a way of obtaining not only a good estimate of the generalization error but also a good estimate of the variance of that error.

Task 17 – Usage of Bias-Variance Plot To Find the optimal poly-degree

to my understanding when we talk about the bias-variance trade-off, we're talking about how balanced our model is according to being underfitted vs being overfitted. As every model should essentially choose a point between but we'd want a good balance for that cause, now a 'good' balance could mean different aspects and what we'd want to achieve by the modeling, so maybe using an elbow method wouldn't always yield the most optimal results, though it's usually the best we can get to the optimal results, for example when dealing with clustering, the degree of complexity would roughly mean how many centroids we use (in case of those algorithms which take in the argument) and in these cases using an elbow method on the bias-variance trade-off is the best we can get to the optimal number of centroids

though I could imagine a case where we don't really care about the overfitting as much as we would care about not having underfitting (imagine we have a lot of data which would almost always capture what new data could tell us, in this case, we could overlook the overfitting part in favor of not getting underfit.


underfitting zone
overfitting zone
generalization (test) error
bias squared
variance
irreducible error
training error
error
model complexity