Census Income Report

a) Explore basic information of the dataset and check if there are any issues.

First we import both training and test data, specifying the columns as those given in the assignment brief, done using the 'names' argument of the pandas.read_csv() function. Immediately the first row of the test data is dropped using pandas.drop(), as it is stated to not be relevant. For exploration of the data a combined dataframe is created, *all_data*, and its basic information displayed using pandas.info().

```
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  object
 1   workclass       48842 non-null  object
 2   fnlwgt          48842 non-null  float64
 3   education       48842 non-null  object
 4   education-num   48842 non-null  float64
 5   marital-status  48842 non-null  object
 6   occupation      48842 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  float64
 11  capital-loss    48842 non-null  float64
 12  hours-per-week  48842 non-null  float64
 13  native-country  48842 non-null  object
 14  label           48842 non-null  object
dtypes: float64(5), object(10)
```

We can see that no columns have any null values, which is good, but we see that the *age* column is given an object Dtype. We also note that the other numeric columns are floats, but could be simplified to integers, as in the training set. This is completed using pd.apply() in combination with pd.to_numeric().

Although there are no null values listed, we need to check that the data is sensical. This is checked first on the numerical columns using pd.describe().

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	48842.0	48842.0	48842.0	48842.0	48842.0	48842.0
mean	39.0	189664.0	10.0	1079.0	88.0	40.0
std	14.0	105604.0	3.0	7452.0	403.0	12.0
min	17.0	12285.0	1.0	0.0	0.0	1.0
25%	28.0	117550.0	9.0	0.0	0.0	40.0
50%	37.0	178144.0	10.0	0.0	0.0	40.0
75%	48.0	237642.0	12.0	0.0	0.0	45.0
max	90.0	1490400.0	16.0	99999.0	4356.0	99.0

We can see that all the numerical data seems reasonable - although the multiple 0.0 values in *capital-gain* and *capital-loss* initially give the impression something might be wrong, this is

simply a result of the majority of people in the dataset not buying or selling capital, which does not seem unreasonable.

Next we look at the categorical variables. Again, we can use the pd.describe() function to get an overview of each feature.

	workclass	education	marital-status	occupation	relationship	race	sex	native-country	label
count	48842	48842	48842	48842	48842	48842	48842	48842	48842
unique	9	16	7	15	6	5	2	42	4
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-States	<=50K
freq	33906	15784	22379	6172	19716	41762	32650	43832	24720

One thing that attracts attention from this output is that *label* has 4 unique values, when we'd only expect 2 as we want to create a binary classifier. To inspect further we print the output of pd.unique() for each of these features.

```
CAT COLS UNIQUE VALUES:

workclass  :
 [' State-gov' ' Self-emp-not-inc' ' Private' ' Federal-gov' ' Local-gov'
 ' ?' ' Self-emp-inc' ' Without-pay' ' Never-worked']
education  :
 [' Bachelors' ' HS-grad' ' 11th' ' Masters' ' 9th' ' Some-college'
 ' Assoc-acdm' ' Assoc-voc' ' 7th-8th' ' Doctorate' ' Prof-school'
 ' 5th-6th' ' 10th' ' 1st-4th' ' Preschool' ' 12th']
marital-status  :
 [' Never-married' ' Married-civ-spouse' ' Divorced'
 ' Married-spouse-absent' ' Separated' ' Married-AF-spouse' ' Widowed']
occupation  :
 [' Adm-clerical' ' Exec-managerial' ' Handlers-cleaners' ' Prof-specialty'
 ' Other-service' ' Sales' ' Craft-repair' ' Transport-moving'
 ' Farming-fishing' ' Machine-op-inspct' ' Tech-support' ' ?'
 ' Protective-serv' ' Armed-Forces' ' Priv-house-serv']
relationship  :
 [' Not-in-family' ' Husband' ' Wife' ' Own-child' ' Unmarried'
 ' Other-relative']
race  :
 [' White' ' Black' ' Asian-Pac-Islander' ' Amer-Indian-Eskimo' ' Other']
sex  :
 [' Male' ' Female']
native-country  :
 [' United-States' ' Cuba' ' Jamaica' ' India' ' ?' ' Mexico' ' South'
 ' Puerto-Rico' ' Honduras' ' England' ' Canada' ' Germany' ' Iran'
 ' Philippines' ' Italy' ' Poland' ' Columbia' ' Cambodia' ' Thailand'
 ' Ecuador' ' Laos' ' Taiwan' ' Haiti' ' Portugal' ' Dominican-Republic'
 ' El-Salvador' ' France' ' Guatemala' ' China' ' Japan' ' Yugoslavia'
 ' Peru' ' Outlying-US(Guam-USVI-etc)' ' Scotland' ' Trinadad&Tobago'
 ' Greece' ' Nicaragua' ' Vietnam' ' Hong' ' Ireland' ' Hungary'
 ' Holand-Netherlands']
label  :
 [' <=50K' ' >50K' ' <=50K.' ' >50K.']
```

We see that the 4 entries of *label* are ' <=50K', ' >50K', ' <=50K.' and ' >50K'. We want to narrow these down to 2 columns. We will convert the values of ' <=50K.' and ' >50K.' to '

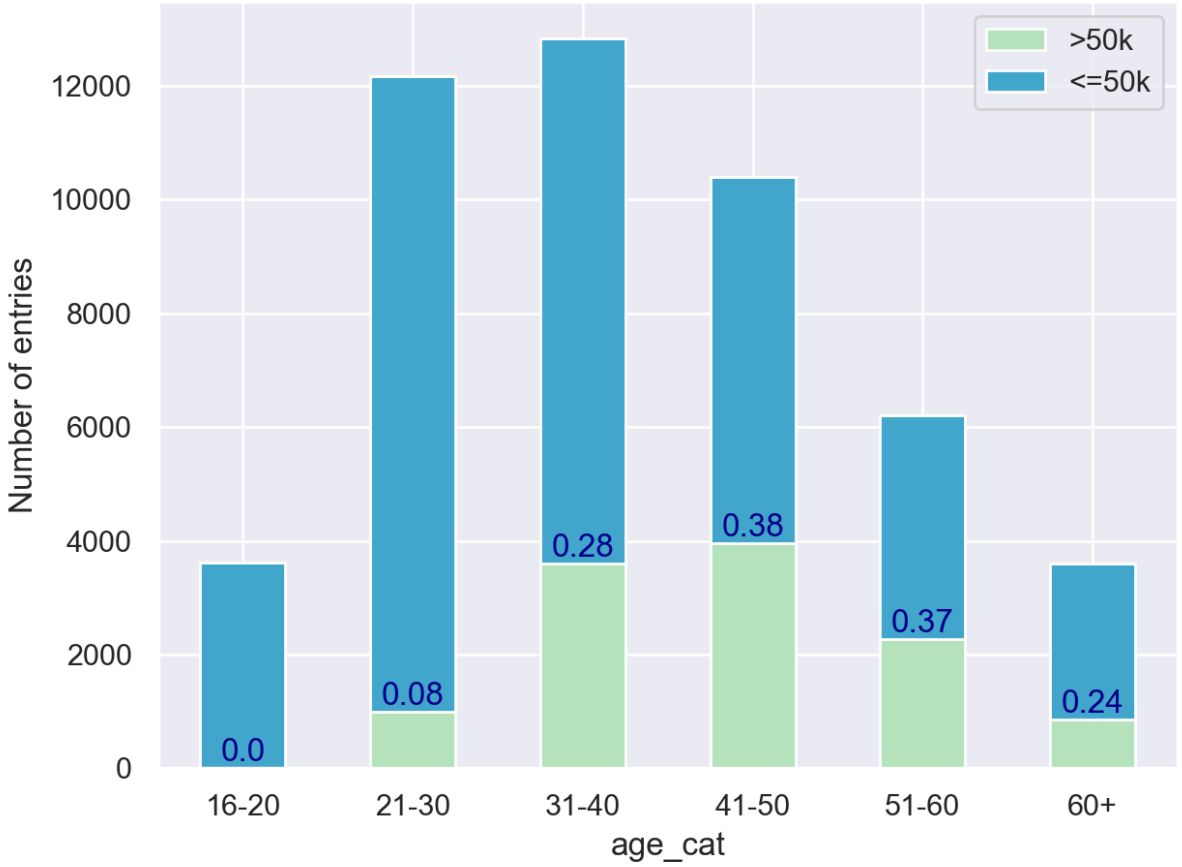
<=50K' and ' >50K' so that we have consistency throughout. Also the space at the beginning is removed on all values.

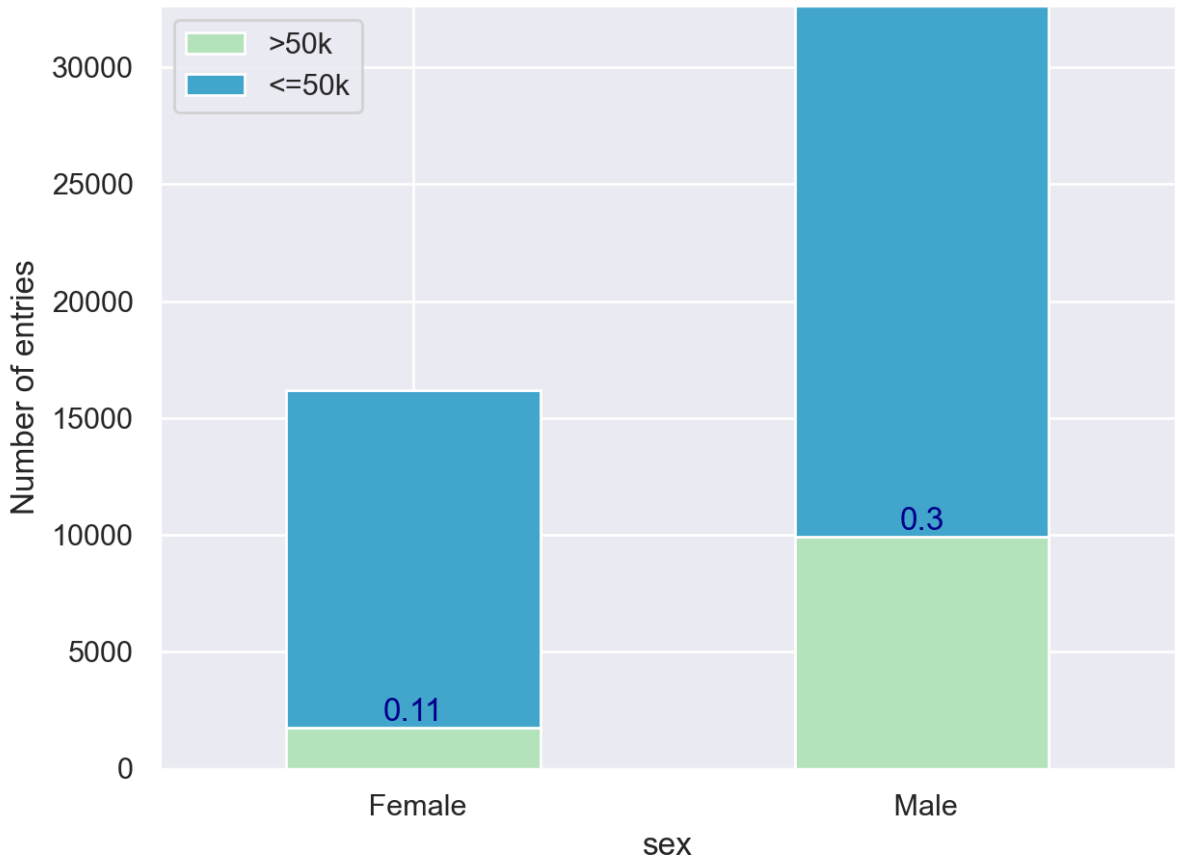
Looking at the unique values for the other features, it appears like there are no bad entries or typos of the entries etc. However, in the *workclass*, *occupation* and *native-country* features there is a value '?', indicating a lack of data. For now we shall replace this with null values in each of those columns, using pd.replace(). What's more, almost all of the categorical data contains whitespace before it, so this is removed using pd.str.strip().

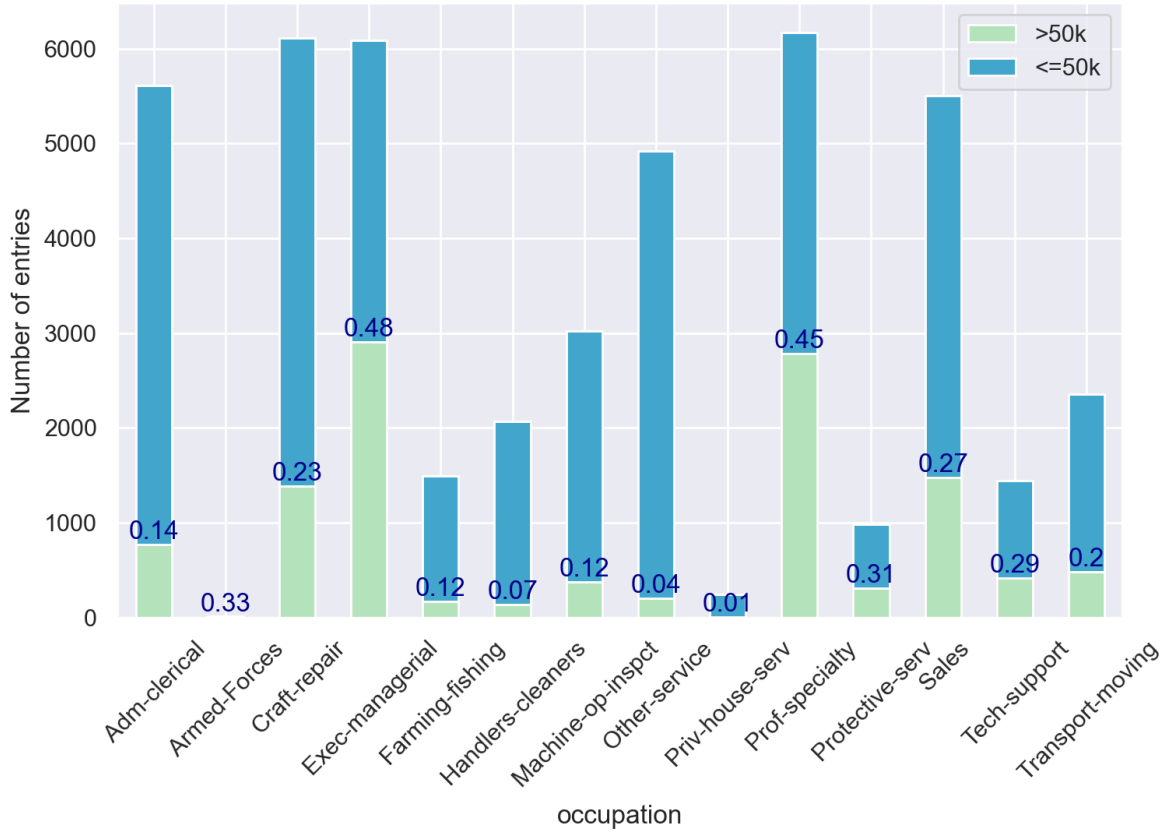
Finally, we check the data for entire duplicate rows using the pandas duplicated() function, which finds that there are 52 duplicate entries. However, given the size of the dataset (almost 50,000 entries) and the lack of specificity in the features, it is possible that this many duplicate entries could occur as simply different people with the same details. There is no way of knowing if that is the case or not for these duplicate entries, so they are left in the data.

b) Explain your findings when comparing total income (labels)

The income labels were compared against the features of age group, gender and occupation through the construction of stacked bar graphs, shown below, that depict the proportion of people in each category whose salary is above 50k.




>50k
<=50k
30000
25000
20000
15000
10000
5000
0
Number of entries
0.11
0.3
Female
Male
sex


>50k
<=50k
6000
5000
4000
3000
2000
1000
0
Number of entries
0.14
0.33
0.23
0.48
0.12
0.07
0.12
0.04
0.01
0.45
0.31
0.27
0.29
0.2
Adm-clerical
Armed-Forces
Craft-repair
Exec-managerial
Farming-fishing
Handlers-cleaners
Machine-op-inspct
Other-service
Priv-house-serv
Prof-specialty
Protective-serv
Sales
Tech-support
Transport-moving
occupation

We can see in the first of the above graphs that the fraction of people with incomes above 50k increases with age from teenagers up to people in their 50s, when it starts to decrease. This is logical as salaries increase throughout one's career, and people's incomes decrease from around 60 as they start to retire.

In the second graph, we can see the clear difference in the proportion of women earning over 50k with the number of men, at 0.11 to 0.3. Finally in the third graph, we observe how almost half of all people listed as *Exec-managerial* or *Prof-speciality* are above the threshold, whilst for *Handlers-cleaners* and *Priv-house-serv* the proportions are very low at 0.07 and 0.01 respectively. However, for the latter the sample size is clearly rather small.

c) Build a classifier for income (labels) with your selected features

Use appropriate methods to handle categorical data

In order to create any machine learning models we need to handle the categorical data in the dataframe. This is undertaken on our joint dataframe, *all_data*, and later this will be split back into the test and training data. The easiest first step in this case is to remove the categorical features that don't contribute information; *age_cat* is no longer needed, and *education* represents the same information as *education-num*, so they are both removed.

Next we look for categorical columns that can be binary encoded, i.e. they only have two categorical values. These are the *sex* and *label* columns. Encoding is completed manually with dictionaries, (although this was done earlier with *label*).

The rest of the categorical variables are nominal variables that need to be represented with dummy variables, which is completed using pd.get_dummies(), with the drop_first parameter set to true, so that we use k-1 features to represent k variables. These columns are *workclass, marital-status, occupation, relationship, race,* and *native-country*.

Now the data is split back into test and training sets using .iloc.

Investigate and train at least 5 classification methods by (adult.data)

In order to train and test the chosen method, first both the test and training data is separated into the features, X, and the response, y, which is in this case the *label* column. All features are left in to start with as feature selection will be investigated later.

The chosen classifiers are taken from the sklearn library, in this case we use K Nearest Neighbours - KNeighborsClassifier(), Logistic Regression - LogisticRegression(), Decision Tree - DecisionTreeClassifier(), Random Forest - RandomForestClassifier(), Gaussian Naive Bayes - GaussianNB() and Support Vector Classifier - LinearSVC. The latter is specified as the linear version of the function because the standard SVC's computational time scales quadratically

with the number of data entries and becomes impractical to use on this big a dataset. This function is also specified with the parameter 'dual=False' as it is stated in the documentation to be preferred when n_samples > n_features, as is the case with our dataset. The maximum number of iterations are also increased to 1000 so that the function can converge, which is also done for Logistic Regression.

The models are evaluated on the training set through cross validation using sklearn's cross_validate() function, using 5 folds. The metrics chosen for evaluation are accuracy and F-score, which are averaged across all folds of the training data and evaluated on the test set after being fit.

Name	Training Accuracy	Training F-Score	Test Accuracy	Test F-Score
KNN	0.776481	0.410558	0.776611	0.404844
LR	0.797641	0.385413	0.797801	0.377929
DT	0.813396	0.617373	0.811068	0.607353
RF	0.856792	0.676505	0.855967	0.666477
L-SVC	0.799208	0.381119	0.801794	0.380971
NB	0.795184	0.422149	0.795651	0.413538

It is seen here that, in terms of both metrics, the Random Forest is the best performing classifier, at least with its default hyperparameters and all features, closely followed by the Decision Tree. While the worst performers in terms of F-score are Logistic Regression and Linear Support Vector Classifier, in terms of accuracy the worst is the K Nearest Neighbours classifier.

There is very little difference in the metrics between the test data and the training data, meaning that the models generalise well and the numbers obtained in this table are reliable indicators of performance.

The models are saved using the joblib library.

d) Improve individual classifiers
Fine tune hyperparameters

It is difficult to narrow down too much the range of hyperparameters used for each classifier as we don't have a reference for roughly where we want to look. Therefore we have a large list of hyperparameters to test for each model, so make use of sklearn's RandomizedSearchCV function. This is because to undertake a full grid search with cross validation would be quite time and computationally intensive. Obviously this means we may miss some optimum parameters, but it is a necessary compromise.

The hyperparameters chosen to be investigated are centered on the default values of the function, i.e. in the case of KNN leaf size, the default value is 30, so we try out 10,20,30,40

and 50. The same is done for as many of the models' parameters as possible. Each model undergoes search in turn and the combination of hyperparameters that produces the best score, along with the score itself are returned. This metric is specified as F-score, to enable easy comparison with our original models. We see that there is an increase in performance across all the models (we are comparing its score to our original *training* score). Especially noticeable are the jumps in score of the Linear Support Vector Classifier and the Gaussian Naive Bayes, both with an increase of around 0.2.

Name	Best Hyperparameters	Score
KNN	{'p': 1, 'n_neighbors'…	0.426449
LR	{'penalty': 'l2', 'C': 25}	0.426037
DR	{'splitter': 'best', '…	0.667879
RF	{'n_estimators': 250, …	0.683662
L-SVC	{'tol': 1e-05, 'C': 25}	0.61703
NB	{'var_smoothing': 1e-12}	0.643033

Feature Selection

Two methods of feature selection are investigated, one being recursive feature elimination using sklearn's RFECV function, and the other its SequentialFeatureSelector function. The former requires the estimator to provide either a coef_ or feature_importances_ attribute, which neither KNN nor Gaussian Naive Bayes does, so they are excluded from the analysis. As for the hyperparameters investigation, the score used to compare results is the F-Score. It can be seen that reducing the features makes a significant difference in F-Score for L-SVC and Logistic Regression in comparison to the default of all features, and for RF and DT there is still a difference made, but it is very small.

Name	Number Selected Columns	Score	Selected Columns
L-SVC	42	0.593872	['sex', 'workclass_…
LR	37	0.618704	['age', 'fnlwgt', '…
RF	77	0.679106	['age', 'fnlwgt', '…
DT	73	0.621238	['age', 'fnlwgt', '…

The other feature selection method is SequentialFeatureSelector. The downside of using this method is that the number of final features has to be chosen with the initialisation of the selector, unlike for RFECV. As a balance between amount of information and runtime, this parameter is set to find the 10 best features (scored again using F-score). However, it does function on all of the chosen models so that is certainly an advantage over RFECV. The results are shown below and once again we see an improvement in the F-score across all the models compared with the default.

Name	Score	Selected Cols
KNN	0.51683	'native-country_Nicaragua', 'native-country_Outlying-US(Gu… 'native-country_Peru', 'native-country_Portugal',
LR	0.647755	'occupation_Exec-managerial', 'occupation_Sales',
DT	0.683252	'occupation_Exec-managerial', 'occupation_Other-service',
RF	0.682002	'marital-status_Never-married', 'occupation_Exec-manageria…
L-SVC	0.633445	'occupation_Farming-fishing', 'occupation_Tech-support',
NB	0.648852	'occupation_Other-service', 'occupation_Sales', 'native-co…

Test combination of selected features and hyperparameters

As for all of the models, the scores were better when using the forward feature selection, the features chosen with this are combined with the highest performing hyperparameters above and tested on both the training and the test data to see what the improvements made over the default functions are.

Name	Training Accuracy	Training F-Score	Test Accuracy	Test F-Score
KNN	0.586622	0.51683	0.763774	0
LR	0.843832	0.647524	0.849149	0.651037
DR	0.859832	0.6777	0.860021	0.672699
RF	0.860201	0.681342	0.861188	0.678338
L-SVC	0.843494	0.633507	0.84614	0.632859
NB	0.799484	0.648852	0.802223	0.645998

We now look at the test metrics and compare to our original models. All have improved except for KNN, so these are saved again with the new hyperparameters and selected features. KNN is left with its default hyperparameters and features because there is a decrease in both accuracy and F-score. The fact the latter is 0 suggests something is significantly wrong with the new model.

e) Investigate further by using ensemble methods

Sklearn's VotingClassifier and StackingClassifier are utilised to implement ensemble methods. Both of these functions require each of the individual estimators in the ensemble to be fitted on the same features. Therefore we use the full original training X when training the ensembles, with no feature selection. The estimators though were all shown to perform better when using the specific hyperparameters investigated earlier, so these are used for initialisation.

For voting, all the models are inputted as estimators into the function. It's then fitted on the training data and tested on the test data. The scores are nearly at those of the initial Random Forest model, but not quite, so it's not worth the extra calculation necessary with this ensemble method.

For the stacking method, the meta-classifier is chosen as Logistic Regression, and so the rest of the models make up the input estimators. Again, all are initialised with their earlier found 'optimal' hyperparameters. It achieves the best accuracy of the models tested so far, but with an F-score very slightly inferior to that obtained with the optimised Random Forest method. This model is saved because of its high accuracy.