Irene(Jiaying) Xu, UAL Creative Computing Institute

23/24 Introduction to Data Science
In-Class Assignment Portfolio - Exercise 3: Classification

0 Introduction

**Related Course Material: Week 6 Classification, Week 7 Random Forests and Feature Engineering*

In Week 6 & Week 7, we looked at Decision Trees and Random Forest Algorithms for classification tasks. The same dataset used in Exercise 3 (Regression) was also used here to explore how classification and regression differ, as well as to build a model with suitable features.

1 Decision Tree

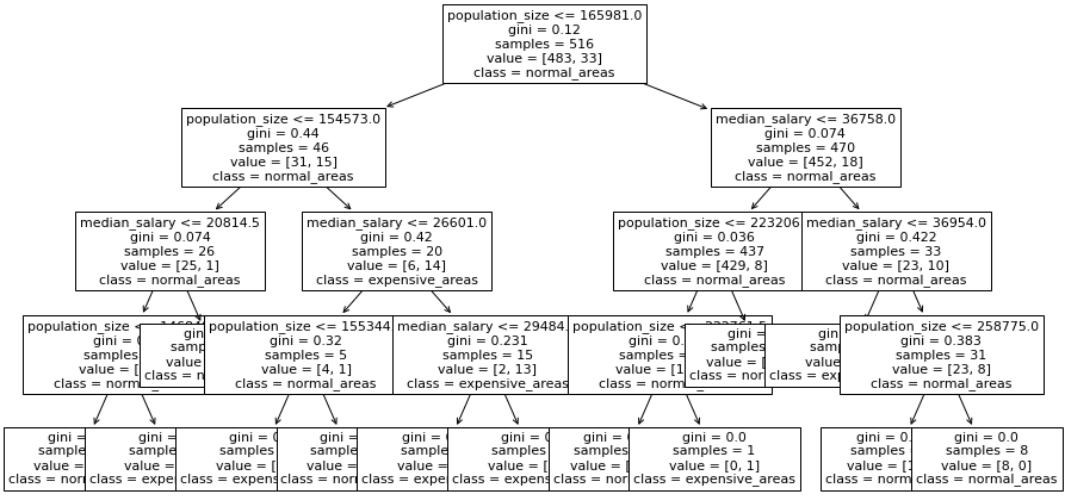
The dataset ("*Housing in London*" created by Justinas Cirtautas) has already been pre-processed in Exercise 3. However, to make categorical predictions on the same dataset, appropriate labels need to be encoded into the df. Drawing from insights gained in Exercise 3, I have decided to first categorise all unique areas into two types: expensive areas (as identified in the regression analysis, Kensington and Chelsea, and Westminster) and normal areas (other areas), as Labelling Method 1.

The updated df, shown below, will be used to conduct analyses with decision trees and random forests, allowing for a comparison between the two methods.

	date	average_price	median_salary	mean_salary	population_size	type	types
0	1999	171300.08333	33020.00000	48922.00000	6581.00000	normal_areas	0.00000
1	1999	65320.83333	21480.00000	23620.00000	162444.00000	normal_areas	0.00000
2	1999	136004.41667	19568.00000	23128.00000	313469.00000	normal_areas	0.00000
3	1999	86777.66667	18621.00000	21386.00000	217458.00000	normal_areas	0.00000
4	1999	112157.41667	18532.00000	20911.00000	260317.00000	normal_areas	0.00000
...	...	...	...	...	...	...	...
655	2018	379262.58333	28853.00000	32442.00000	204525.00000	normal_areas	0.00000
656	2018	446500.41667	49237.00000	69806.00000	317705.00000	normal_areas	0.00000
657	2018	440859.41667	30298.00000	32875.00000	276700.00000	normal_areas	0.00000
658	2018	596649.16667	34501.00000	45317.00000	326474.00000	normal_areas	0.00000
659	2018	1020025.50000	43015.00000	63792.00000	255324.00000	expensive_areas	1.00000

1) Decision Tree with 2 Features – `Median_Salary`, `Population_size`

   To begin, I have selected two features (the same features used to test linear regression in Exercise 3) and split the dataset into 80% training data and 20% testing data. After experimenting with various max_depth settings, I found that accuracy does not really improve beyond a max_depth of 4, making a max_depth of 4 the most appropriate depth for this model - when setting a mex_depth of 4, we have a 95.35% accuracy.



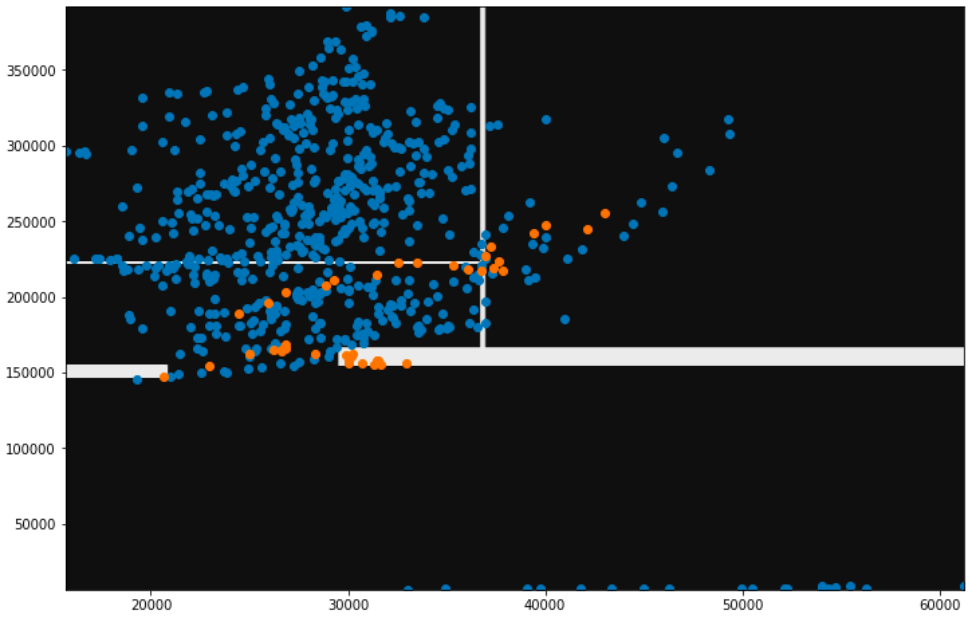

```
Accuracy: 95.34883720930233
Confusion Matrix: [[120   2]
 [  4   3]]
Class Report:               precision    recall  f1-score

         0.0       0.97      0.98      0.98       122
         1.0       0.60      0.43      0.50         7

    accuracy                           0.95       129
   macro avg       0.78      0.71      0.74       129
weighted avg       0.95      0.95      0.95       129
```

It is interesting to note that the same features appear to be effective in building models for classification tasks but perform poorly in regression tasks (refer to Exercise 3 – Part 3.1 Linear Regression). This could suggest that non-linear models (e.g., decision trees) can handle much more complex relationships between features and a categorical target.

However, while the accuracy suggests the model performs well on the test dataset, the precision, recall, and f1-score all suggest that the model has difficulty in correctly classifying most of class 1 (expensive areas). This might be a result of the imbalanced class – as the most of areas are labelled as normal areas.



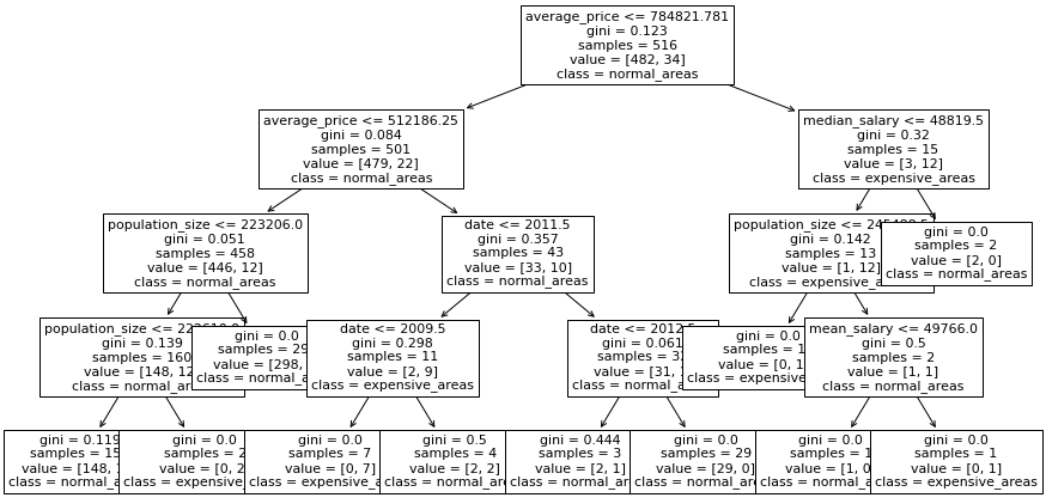
The decision boundary above also suggests a very imbalance, with a large number of blue points compared to orange points. (note that when plotting the decision boundary here, the size of the step needs to be modified based on the dataset, or it will crash the kernel if the data is too big...)

```
# https://hackernoon.com/how-to-plot-a-decision-boundary-for-machine-learning-algorithm
# plot decision boundarys
def plot_decision(X,y,model,n_classes=2):
    min1, max1 = X[:, 0].min()-1, X[:, 0].max()+1
    min2, max2 = X[:, 1].min()-1, X[:, 1].max()+1
    x1grid = np.arange(min1, max1, 100) # need to change the size of step based on the
    x2grid = np.arange(min2, max2, 100)
    xx, yy = np.meshgrid(x1grid, x2grid)
```

2) Decision Tree with Multiple (All) Features

Then, with the same max depth of 4, including not just two features but all available features for input will give us a model with an accuracy of 98.45%. This seems to be a good result and it is better than the previous attempt. However, similar to the previous model, the performance metrics for class 1 (expensive areas) are not as good as those for class (normal areas), suggesting an imbalanced class.

Generally, more features can provide the model with additional information, potentially increasing its predictive power (but not always!! — refer to the case with random forests in the latter part of this exercise).




```
Accuracy: 98.44961240310077
Confusion Matrix: [[122   1]
 [  1   5]]
Class Report:               precision    recall  f1-score

         0.0       0.99      0.99      0.99       123
         1.0       0.83      0.83      0.83         6

    accuracy                           0.98       129
   macro avg       0.91      0.91      0.91       129
weighted avg       0.98      0.98      0.98       129
```

```
# get feature importances
feature_importances = model.feature_importances_
importances = pd.Series(feature_importances, index=x_labels)
sorted_importances = importances.sort_values(ascending=False)
sorted_importances
```

```
average_price     0.48247
date              0.28987
population_size   0.13228
median_salary     0.07126
mean_salary       0.02412
dtype: float64
```

Additionally, by using the *.feature_importances_* attribute from the sklearn library, we can see that 'average_price' is the most significant predictor, with an importance score of approximately 0.48247. Meanwhile, 'mean_salary' and 'median_salary' only

have an importance score of around 0.05, suggesting that they may not have much predictive power.

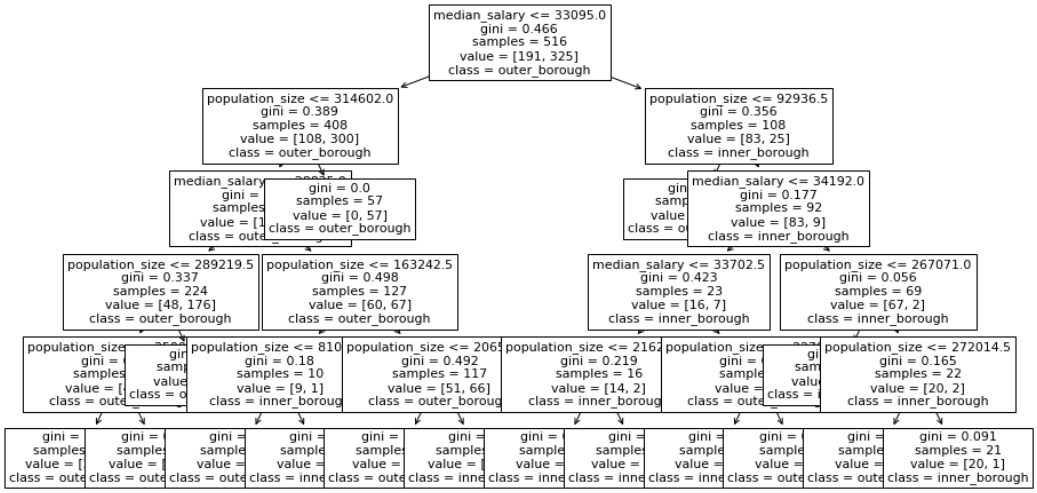
3) Alternative Labelling Method - Attempt to Fix Class Imbalance

Based on the above models and metrics, Labelling Method 1 created a very imbalanced class (labelling areas as expensive or normal based solely on my perception might be too intuitive). Therefore, I also explored an alternative labelling method (Labelling Method 2) that categorises boroughs into inner and outer boroughs, according to Wikipedia page of London boroughs.

> "camden", "greenwich", "hackney", "hammersmith and fulham", "islington", "kensington and chelsea", "lambeth", "lewisham", "southwark", "tower hamlets", "wandsworth", "westminster" are categorieses into inner boroughs, with a labelling number of 0. The rest of the boroughs are labelled with 1 – outer boroughs.

The following are the results of the created models based on the new labelling method, with a max depth of 5:

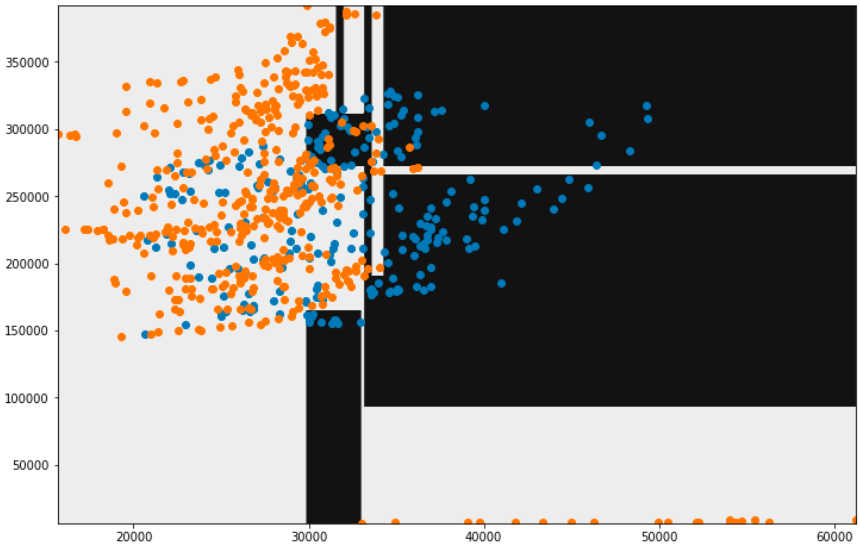
[Decision Tree with 2 features – Median_Salary`, `Population_size`]



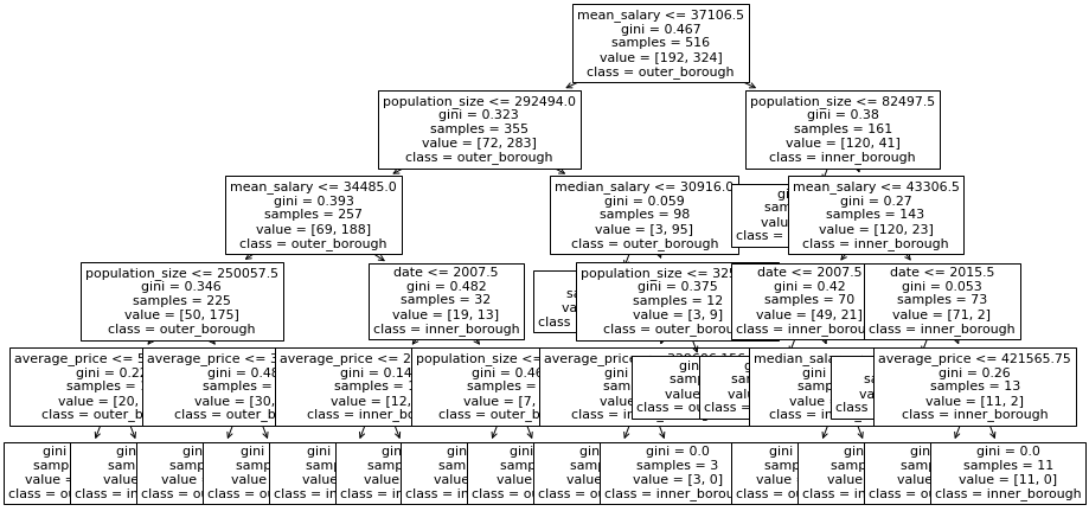

```
Accuracy: 74.4186046511628
Confusion Matrix: [[26 20]
 [13 70]]
Class Report:                precision    recall  f1-score

         0.0       0.67      0.57      0.61        46
         1.0       0.78      0.84      0.81        83

    accuracy                           0.74       129
   macro avg       0.72      0.70      0.71       129
weighted avg       0.74      0.74      0.74       129
```



[Decision Tree with All features]




```
Accuracy: 79.84496124031007
Confusion Matrix: [[21 24]
 [ 2 82]]
Class Report:               precision    recall  f1-score
```

0.0	0.91	0.47	0.62	45
1.0	0.77	0.98	0.86	84
accuracy			0.80	129
macro avg	0.84	0.72	0.74	129
weighted avg	0.82	0.80	0.78	129

```
# get feature importances
feature_importances = model.feature_importances_
importances = pd.Series(feature_importances, index=x_labels)
sorted_importances = importances.sort_values(ascending=False)
sorted_importances
```

✓ 0.0s

```
mean_salary       0.50891
population_size   0.29860
average_price     0.07667
date              0.06471
median_salary     0.05112
dtype: float64
```

Reflection - A model with a high accuracy rate does not necessarily indicate a better model for prediction, especially when it has a very imbalanced class. With the new labelling method, despite the accuracy of the model (if we look at models with all features) has significantly dropped from around 98% to 79%, the evaluation metrics actually give us a more balanced result, and the improved decision boundary also suggested that Labelling Method 2, while not perfectly resolving class imbalance, has indeed shown a significant improvement.

It is also worth noting that with different labels, the most significant predictor has changed from 'average_price' to 'mean_salary', suggesting that 'mean_salary' might be better in capturing the variance between these new classes.

All in all, from the above explorations, it's crucial to understand that when building a predictive model, multiple evaluation metrics and factors need to be considered when assessing the performance of a model; relying solely on one metric, such as accuracy, can be misleading.

3 Random Forests and Feature Engineering

Would an ensemble of Decision Trees give us a better model?

The following screenshot shows that with Labelling Method 2, ensemble models perform better than decision trees, proving that ensemble techniques that combine multiple models can indeed improve prediction accuracy.

**Random Forest and AdaBoost - the accuracy of the random forest (bagging) model is slightly higher than the accuracy of Adaboost (boosting) model – is it affected by noisy data?*

```
# fit the random forest model
model = RandomForestClassifier(oob_score=True, random_state=42, n_estimators=500, n_jobs=-1)
model.fit(x, y)
print("OOB accuracy", model.oob_score_)
```

✓ 0.4s Python

```
OOB accuracy 0.8635658914728682
```

```
# fit the adaboost model
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model = AdaBoostClassifier(n_estimators=500, random_state=42)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy",accuracy)
```

✓ 0.3s Python

```
accuracy 0.8294573643410853
```

Both labelling methods 1 and 2 categorise the area into different categories, but I guess it's also interesting to see if we can build a model that accurately predicts the area based on the information available in the dataset?

1) <u>Further Exploration on Ensemble Models – Housing in London Dataset</u>

   First, as the 'area' column contains only categorical variable (the name of each unique area is quite straightforward already and needs no further feature engineering), we need to use LabelEncoder object from sklearn to assign each unique area with a number. With a total of 33 unique areas, we have 33 encoded numbers ranging from 0 to 32, which will be included in a new column named 'area_label'.

```
features = ["mean_salary", "population_size"]
train(get_dataset(features))
```

✓ 0.4s

```
OOB accuracy 0.4232558139534884
Feature Importances: [0.46210708 0.53789292]
```

```
features = features + ["median_salary", "date", "average_price"]
train(get_dataset(features))
```

✓ 0.4s

```
OOB accuracy 0.7116279069767442
Feature Importances: [0.19979877 0.34013924 0.1674578  0.10768326 0.18492093]
```

By selecting the two features with the highest importance scores from the previous decision tree ('mean_salary' and 'population_size') and fitting them to the Random Forest Model will give us a model accuracy of 0.423 accuracy. Adding in more features about date, median salary, average price significantly improves accuracy (0.711) than using just 'mean_salary' and 'population_size' alone.

Interestingly, in contrast to the decision tree model, the most significant predictor in this random forest model has changed again. Now, 'population_size' has the highest importance score, followed by 'mean_salary'. This suggests that if we aim to predict a specific area rather than a broader one, population size and area mean salary are the better features.

So, despite having similar classification objectives (e.g., inner or outer boroughs, expensive or normal boroughs, or each specific borough), different labels or labelling methods will require different features. This highlights the importance of feature engineering in model development. Carefully considered and tailored feature selection and engineering are essential to meet the specific goals of each classification task.

2) New Dataset – Titanic Dataset

Since the Housing in London dataset has limited features and its only categorical values require no further feature engineering, I sought out another well-known dataset on Kaggle – [Titanic Dataset - Titanic - Machine Learning from Disaster Competition on Kaggle] to further practice on feature engineering, and hopefully, build a predictive model to determine which types of passengers have the greatest chance of survival.

After removing/replacing missing values and uninformative columns, the dataset has 891 rows × 9 columns, providing information on survival status (0 = No, 1 = Yes), ticket class (1 = 1st, 2 = 2nd, 3 = 3rd), passenger name, gender, age, No. of siblings/spouses aboard the Titanic, No. of parents/children aboard the Titanic, ticket fare, and embarkation point.

- Most of the 'cabin' information is missing – the whole column has been dropped from the dataset.
- Missing 'gender' information replaced with a random number (mean +-10)
- Missing 'embarked' information replaced with mode

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	Braund, Mr. Owen Harris	male	22	1	0	7.25000	S
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	71.28330	C
2	1	3	Heikkinen, Miss. Laina	female	26	0	0	7.92500	S
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	53.10000	S
4	0	3	Allen, Mr. William Henry	male	35	0	0	8.05000	S
...	...	...	...	...	...	...	...	...	...
886	0	2	Montvila, Rev. Juozas	male	27	0	0	13.00000	S
887	1	1	Graham, Miss. Margaret Edith	female	19	0	0	30.00000	S
888	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	27	1	2	23.45000	S
889	1	1	Behr, Mr. Karl Howell	male	26	0	0	30.00000	C
890	0	3	Dooley, Mr. Patrick	male	32	0	0	7.75000	Q

891 rows × 9 columns

At first glance, I think that the survival rate would be closely associated with ticket class and fare; by selecting those two features, the Random Forest model yields an accuracy of 0.693. Surprisingly, incorporating additional features such as age, number of siblings/spouses aboard (sibsp), and the number of parents/children aboard (parch) doesn't increase but slightly decreases the model's accuracy.

```
features = ["Pclass", "Fare"]
train(get_dataset(features))
```
✓ 0.3s Python

```
OOB accuracy 0.6936026936026936
Feature Importances: [0.13929261 0.86070739]
```

```
features = features + ["Age"]
train(get_dataset(features))
```
✓ 0.4s Python

```
OOB accuracy 0.67003367003367
Feature Importances: [0.08894585 0.4091549  0.19961045 0.06043554 0.04341999 0.19843326]
```

```
features = features + ["SibSp"]
train(get_dataset(features))
```
✓ 0.4s Python

```
OOB accuracy 0.6644219977553311
Feature Importances: [0.08713293 0.40219258 0.20038452 0.03313942 0.04338799 0.20142462
 0.03233794]
```

```
features = features + ["Parch"]
train(get_dataset(features))
```
✓ 0.4s Python

```
OOB accuracy 0.6632996632996633
Feature Importances: [0.08419264 0.39860595 0.20245729 0.03522492 0.02377732 0.20001336
 0.03288533 0.02284319]
```

I am also curious about whether gender affects the chance of survival. By encoding all females as 0 and all males as 1 in the 'gender' column, I found that the model's accuracy, after including 'gender', was indeed much higher (0.82), indicating that a passenger's gender seems to have a high effect on their chance of survival.

```
features = ["Pclass", "Fare"]
features = features + ["gender"]
train(get_dataset(features))
```
✓ 0.4s Python

```
OOB accuracy 0.8204264870931538
Feature Importances: [0.10834905 0.54614096 0.34550999]
```

I also tried to add the number of siblings/spouses aboard (sibsp), the number of parents/children aboard (parch), and age to the model, but the overall accuracy did not significantly change or even slightly decreased, which might indicate that these features might not significantly contribute to the model's predictive power.

This is the same situation when I try to add the 'embarkation point' feature to the model (by encoding the 3 different embarkation points as 0, 1, and 2 in a new column). The accuracy dropped after adding this new feature, indicating that a passenger's embarkation point also seems to be irrelevant to their survival rate (more features don't necessarily mean it will increase predictive power!)

Adding all features together – the feature importance shows that fare, age, gender are the most important features related to the survival of the passengers.

```
features = ["Pclass", "Fare", "gender", "Age", "SibSp", "Parch"]
train(get_dataset(features))
```

✓ 0.4s Python

```
OOB accuracy 0.7901234567901234
Feature Importances: [0.08534921 0.29219749 0.26452718 0.27454449 0.04861266 0.03476897]
```

[Reflection]
While the model shows some reasonable results, there are some problems, especially regarding the dataset itself.

- The handle of missing data
  In the above exercise, a full column has been removed from the dataset and some of the other missing data is replaced with mean/mode – easier to compute a model but it also leads to biased estimates and a decrease in the model's predictive power.

  Removing data also leads to a potential ethical problem – especially in such cases that involve individuals/perspectives in real life, incomplete dataset will only lead to incomplete and flawed analysis, missing data needs to be handled very carefully, and consider not only its numeric value but also its social and cultural meanings.
  **reference posts: LinkedIn community - What methods can you use to manage missing data in your analyses?*

- Other features
  The model only considers some of the most obvious features – other features such as age group, last name, cabin floor (which depends on the cabin number), and whether they were travelling alone also need to be considered, when possible, to improve the model's performance.

  While more features are NOT necessarily always better – having a comprehensive dataset or a broad set of information at the very beginning is beneficial and can help us better understand and identify features.

- Model Selection
  It seems that ensemble models (such as the random forests here) that combine multiple models are often better than individual models, then why not always use ensemble modelling?

  – in the case of housing prices in this exercise, some relationships (e.g., `date` & `avg. housing price`, refer to exercise 2) can already be explained well by linear correlation/regression, ensemble models may be able to increase accuracy but may also doesn't really worth the hassle, it would be more obvious if we have a very large dataset, as ensemble models are more complicated (more computational resources) and harder to tune and to explain – accuracy is not the only parameter, multiple factors related to the context of the problem needs to be considered together when selecting a model, especially in solving real life problems.

  – ensemble models, with no careful training, can also very easily produce overfitting models (Alhamid (2022)), which is something that needs to be kept in mind in future exploration of ensemble models.

4 Code, Supporting Document, and LLM Disclaimer

All datasets, code, Jupyter Notebooks, and GIF files are available to access via the GitHub repository: https://git.arts.ac.uk/23001934/ds-portfolio

ChatGPT 3.5 has been used in this exercise for code debugging and proofreading.

5 References and External Resources

Reference:

[1] Alhamid, M. (2022). *Ensemble Models: What Are They and When Should You Use Them? | Built In*. [online] builtin.com. Available at: https://builtin.com/machine-learning/ensemble-model [Accessed 1 Mar. 2024].

External Resources (Blogs, Posts, etc.):

[1] https://stackoverflow.com/questions/60481193/how-to-select-rows-in-a-dataframe-based-on-an-or-operator-of-values-of-a-colum

[2] https://stackoverflow.com/questions/39078781/filter-a-pandas-data-frame-on-all-rows-that-do-not-meet-a-condition

[3] https://scikit-learn.org/stable/modules/model_evaluation.html

[4] https://www.geeksforgeeks.org/python-pandas-dataframe-isin/

[5] https://www.statology.org/pandas-pie-chart/

[6] https://stackoverflow.com/questions/6736590/fast-check-for-nan-in-numpy

[7] https://en.wikipedia.org/wiki/London_boroughs

[8] https://www.linkedin.com/advice/3/what-methods-can-you-use-manage-missing-data-your-iswce

**Detailed annotations were included in the Jupyter Notebook along with the code to indicate where ChatGPT and external resources, such as StackOverflow posts and other Python study materials were used.*