Portfolio assignment 16

30 min: Train a decision tree to predict one of the categorical columns of your own dataset.

- Split your dataset into a train (70%) and test (30%) set.
- Use the train set to fit a DecisionTreeClassifier. You are free to to choose which columns you want to use as feature variables and you are also free to choose the max_depth of the tree.
- Use your decision tree model to make predictions for both the train and test set.
- Calculate the accuracy for both the train set predictions and test set predictions.
- Is the accurracy different? Did you expect this difference?
- Use the plot_tree function above to create a plot of the decision tree. Take a few minutes to analyse the decision tree. Do you understand the tree?

In [1]:
```
import pandas as pd
import seaborn as sns
```

In [2]:
```
steam = pd.read_csv('../week 1/steam_games.csv')
steam.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-2893af2ee9ca> in <module>
----> 1 steam = pd.read_csv('../week 1/steam_games.csv')
      2 steam.head()

~\anaconda3\lib\site-packages\pandas\io\parsers.py in read_csv(filepath_or_buffer, se
p, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dt
ype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfo
oter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_d
ates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterat
or, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, do
ublequote, escapechar, comment, encoding, dialect, error_bad_lines, warn_bad_lines, de
lim_whitespace, low_memory, memory_map, float_precision)
    684     )
    685 
--> 686     return _read(filepath_or_buffer, kwds)
    687 
    688 

~\anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filepath_or_buffer, kwds)
    450 
    451     # Create the parser.
--> 452     parser = TextFileReader(fp_or_buf, **kwds)
    453 
    454     if chunksize or iterator:

~\anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, f, engine, **kwd
s)
    944             self.options["has_index_names"] = kwds["has_index_names"]
    945 
--> 946         self._make_engine(self.engine)
    947 
    948     def close(self):

~\anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engine(self, engine)
   1176     def _make_engine(self, engine="c"):
   1177         if engine == "c":
-> 1178             self._engine = CParserWrapper(self.f, **self.options)
   1179         else:
   1180             if engine == "python":

~\anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, src, **kwds)
   2006         kwds["usecols"] = self.usecols
   2007 
-> 2008         self._reader = parsers.TextReader(src, **kwds)
   2009         self.unnamed_cols = self._reader.unnamed_cols
   2010 

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_source()

FileNotFoundError: [Errno 2] No such file or directory: '../week 1/steam_games.csv'
```

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
len(steam)
```

In [ ]:
```
len(steam.dropna())
```

that's a lot less rows... let's fill up the empty spaces.
we'll start by filling up the numerical values

In [ ]:
```
steam = steam.fillna(value={'achievements': 0, 'discount_price': steam.original_price
```

In [ ]:
```
len(steam.dropna(subset=['developer']))
```

This is still a fine size, plus it's weird to have games without a developer.

In [ ]:
```
steam = steam.dropna(subset=['developer'])
```

In [ ]:
```
features= ['achievements']
dt = DecisionTreeClassifier(max_depth = 1) # Increase max_depth to see effect in the p
dt.fit(steam[features], steam['developer'])
```

In [ ]:
```
from sklearn import tree
import graphviz

def plot_tree_classification(model, features, class_names):
    # Generate plot data
    dot_data = tree.export_graphviz(model, out_file=None,
                          feature_names=features,
                          class_names=class_names,
                          filled=True, rounded=True,
                          special_characters=True)

    # Turn into graph using graphviz
    graph = graphviz.Source(dot_data)

    # Write out a pdf
    graph.render("decision_tree")

    # Display in the notebook
    return graph
```

In [ ]:
```
plot_tree_classification(dt, features, steam.developer.unique())
```

Here it asks if there are less than 0.5 achievements, it'll be from Tokiwa Graphics.

In [ ]:
```
predictions = dt.predict(steam[features])
```

In [ ]:
```
def calculate_accuracy(predictions, actuals):
    if(len(predictions) != len(actuals)):
        raise Exception("The amount of predictions did not equal the amount of actuals

    return (predictions == actuals).sum() / len(actuals)
```

In [ ]:
```
calculate_accuracy(predictions, steam.developer)
```

I guess this has a very low accuracy.

In [ ]:
```
len(steam.developer.unique())
```

This might be why

In [ ]:
```
top10 = steam.developer.value_counts().sort_values(ascending=False).index[:10]
```

In [ ]:
```
steam.loc[~steam.developer.isin(top10),'developer'] = 'Other'
```

In [ ]:
```
len(steam.developer.unique())
```

In [ ]:
```
steam.developer.value_counts()
```

In [ ]:
```
features= ['achievements']
dt = DecisionTreeClassifier(max_depth = 10) # Increase max_depth to see effect in the
dt.fit(steam[features], steam['developer'])
```

In [ ]:
```
plot_tree_classification(dt, features, steam.developer.unique())
```

Apparently now the majority is Ubisoft, so we're going to have to check per class.