Step 1: Download repo, and open '*guided_notebook_(loans).ipynb.'*

Step 2: State brief/problem statement

Today we are a bank, and we'll be building our own machine learning model to predict whether a customer should be given a loan or not.

This is entirely based on the decisions our employees have made in the past.

Step 3: Load the Pandas library

Explain what Pandas does -- perhaps show documentation briefly (https://pandas.pydata.org/)

Please note, that loading one library at a time was found to be more useful and informative than loading altogether. I.e. load each library *only when you need to use its functionality.*

Step 4: Read the data

Step 5: Run useful pandas functions to 'get a feel' for our loans data.

Specifically, *.shape, .describe(), .info().*

Note: No need to go into more functions, just be sure to explain what each function does, and why its useful.

Step 6: Data Cleaning

6.1 Where are our NAs?

Run *loan.isnull().head()* -- shows True/False. This is good, but we need an aggregate view.

Therefore, we then replace the .head() with a *.null(),* which gives us the number of Null/NA values in each of the columns.

6.2 Deal with one column's NAs at a time

For each column:

1) Run the ***loan['COLUMN_NAME'].value_counts()*** function to see the unique values this column holds.

2) **ASK/OPEN TO FLOOR:** What do you think NA could mean here? What do we do? One option that is always possible is to remove those rows. But maybe we have a better way to deal with NAs for this particular column? Maybe we'll end up deleting lots of rows (e.g. Credit History has 49 NAs), and since we only have ~1000 rows (tiny data!), those rows could affect our predictive ability later on as they just might hold some useful information in the other columns.

3) Coding facilitator to make their decision, whilst giving their reasons -- e.g. I think it's best if I remove the rows for Gender = NA. (Specific possible justifications are given for each variable in the ***teacher_notebook_(loans).ipynb*** notebook.

4) Encourage learners to make their own decisions -- but they need to give a valid reason. E.g. "Anyone think we should replace them with 0 (if applicable, e.g. for Dependents variable)? Why do you think that?" Etc.

   Point to emphasise: They're data scientists and should be responsible for their models when doing this in the real world. Makes things more personal, makes them feel responsible for people getting or not getting loans in the end. Note, this will obviously mean people will end up with different results, because they have different models (but that's completely fine!)

Step 7: Exploratory data analysis

(NOTE: Forget PANDAS for this one, stick to Seaborn!)

1) Go up to the first cell and import seaborn library -- we use *pd* for pandas, and *sns* for seaborn. This is just common practice, so when I send you my code, or when you see some code online, you know what these aliases stand for.

2) Run **%matplotlib inline** -- why? This is so our graphs can be displayed here in our notebook!

3) Create first seaborn plot (e.g. countplot of Yes vs. No for Loan_Status)

4) **ASK:** What else might be interesting to plot? E.g. Maybe the relationship between *ApplicantIncome* and *LoanAmount?* To do this use, *sns.scatterplot().*

5) Refer to seaborn documentation on counttplot: https://seaborn.pydata.org/generated/seaborn.countplot.html, and if deem it appropriate, scatterplot documentation https://seaborn.pydata.org/generated/seaborn.scatterplot.html.

6) Give them 5 mins (**IN PAIRS)** to make whatever graphs they want. Support all round. Pairs works better because it feels less like a test, and they can always lean on each other and ask each other what they may think are stupid questions.

7) **EMPHASISE** that it's okay if they didn't have plenty of time to visualise what they wanted to visualise/explore further -- after all, you can spend an entire day on this alone! We also have a module dedicated to this (called Visualisation) in sprint 1, and they have the entire data academy to practice this, so no worries!

We're here to build a predictive model and go through the entire pipeline, so moving on...

Step 8: Modelling

8.1 Necessary transformation pre-modelling

Before we model, we need to make a couple of transformations in order for machine learning algorithm to work:

1) **Drop the ID column --** as it adds no value at all. Different value for each row, and hence, won't add value in terms of predicting whether a person should be given a loan or not.

2) **Split vertically --** i.e. into features (all variables but our target variable) and our target/class variable (what we're predicting: loan status).

3) **Split horizontally --** i.e. into a training set (what we're training our model on), and a testing set (data our model will not have seen -- so we know how well our model has done! Or will do, given new data in the future).

a) We'll be using the *sklearn* library (https://scikit-learn.org/stable/) -- a very comprehensive machine learning library that we'll be using just as much as pandas for the rest of the course.

b) Run in top cell "*From sklearn.model_selection import train_test_split"*

What is this? Same as before, but we won't be importing the whole library since it's huge! We only need the function required to split the dataset horizontally. This we get from the 'sublibrary' called model_selection

Note: Explain parameters in the *train_test_split()* function as in the teacher notebook.

4) One-hot encoding

a) The decision tree we'll be using, uses a binary split (remember Maria?)

b) However, many of our variables have more than two (binary) values -- e.g. Dependents has "0", "1", "2", "3+". How do we binarize that?

c) Answer: one-hot encoding. Fancy name for making each column value a column in its own right

d) Draw on a flipchart how this looks (simple example) -- i.e. Dependents_0, Dependents_1, Dependents_2, and Dependents_3+ are all columns now, and the values they hold are either 0 or 1 (binary!)

e) To do this in pandas, we use the neat function, *pd.get_dummies().* We need to apply this to both the training and testing **features** sets of course!

Note: if you apply this to the whole dataset, this will create two columns for the Loan status column (i.e. *Loan_Status_Y*, and *Loan_Status_N*). Worth emphasising, to clarify point further.

f) See what this looks like now by running the *shape* and *columns* functions. See the changes?

Note: no need to convert the Loan Status values to 0 or 1 from 'Y' and "N". Can keep as is.

8.2 Train the decision tree model

1) Import the decision tree function:

"*From sklearn.tree import DecisionTreeClassifier"*

Again, we don't need the entire library, just the Decision Tree classifier function, and this we get from the sublibrary "tree".

We get this as before, from the documentation:
https://scikit-learn.org/stable/modules/tree.html

Note, there is a DecisionTreeRegressor and a DecisionTreeClassifier function. The former is concerned with predicting numbers (decimals), and the latter is concerned with predicting categories (1/0, Yes/No). Therefore, we'll be using the latter.

Signpost: Don't worry too much about details here. Remind them they'll have a module just for regression, and another just for classification, where they'll be exploring decision trees in more detail!

2) Import pydotplus
   Import graphviz
   From sklearn.tree import export_graphviz

   Take those at face value: just the libraries and the functions we need to plot and display our decision tree. Nothing important!

3) **Copy/Paste the code on plotting using Graphviz.** (from the ***notes*** folder in the repo)

4) **Plot --** What's the first thing you see? It's so big! So complex! Kind of defeats the purpose of a decision tree. *Refer back to slide where you discussed types of supervised learning models.*

5) Can we deal with this somehow? Yes! It's through a parameter called *max_depth.*

6) Train another model. Same thing, but now we are setting a max_depth of 3.

7) Plot second tree.

8) Any better? Yes! Now it's readable, and we can understand it.

   Could be useful to traverse tree.

Step 9: Making predictions

Remember we built two models, so we'll make two sets of predictions and compare.

For each model, run the *.predict()* function. Output results.

Evaluate using the *classification_report()* function. This we import again from sklearn using: *"from sklearn.metrics import classification_report"*

Step 10: Evaluating our models

When evaluating, we need to look at "**F1 Score". <u>No need to worry about the others!</u>**

Looking at F1-score, it seems to be higher for our second model than in the first. I.e.

- Which model performed better? The second.
- Weird! The first was far more complex -- surely this will mean it should do better?
- Not necessarily! The more complex model *overfit* to very small, menial, noisy, maybe wrong patterns in our training data.

  Therefore, when we applied this to our testing data, which obviously our model hadn't seen before, it assumed that those noisy, menial patterns also applied here as well -- but they didn't!

  Therefore, the simpler (and clearer to understand) model performed better.

  In technical terms, this is what's called the **Bias-Variance tradeoff**. Some machine learning models are high in variance (like decision trees) and are prone to overfitting.

  Some others are higher in bias -- meaning they are better at generalising. The simplest, high bias model could be just calculating the *mean.*

10.1: Optional, but potentially useful

IF you have time and/or you feel the need to explain the rest of the metrics:

1) Draw the following matrix on a flipchart:

	Actual	
Predicted	Positive (Yes)	Negative (No)
Positive (Yes)	**True Positive**	**False Positive**
Negative (No)	**False Negative**	**True Negative**

True Positive -- Rows that were predicted to be Yes, and were in fact Yes

True Negative -- Rows that were predicted to be No, and were in fact No

False Positive -- Rows that were (falsely) predicted to be Yes, and were in fact No

False Negative -- Rows that were (falsely) predicted to be No, and were in fact Yes

$$\textbf{Precision} = \frac{tp}{tp + fp}$$

In other words, out of all positives we predicted, how many were correct?

$$\textbf{Recall} = \frac{tp}{tp + fn}$$

In other words, out of all positives in the dataset, how many did we predict to be positive?

F1 - score -- average of both.