In [23]:
```
%%capture
!pip install fastbook
```

In [24]:
```
import pandas as pd, fastai, numpy as np
from fastai.tabular.all import *
```

Using a Framework on the Titanic Dataset

Introduction

In the previous notebook we manually built a deep learning (DL) model to demonstrate the speci how neural networks (NNs) work. However, in practice and when not learning, building NNs from is generally time-consuming and will yield worse results. By using pre-made architectures and pı models, we can get better results as the models have been optimised by experts and with exten: research.

In this notebook, we will be using fastai to reproduce the results of the last notebook, where we (model that predicts whether a person survived the Titanic disaster. We will also employ a more a technique called ensembling to improve on our model further.

Prepare Data

In [25]:
```
df = pd.read_csv('train.csv')
```

In the last notebook, because we were building the NN from scratch, we had to carefully feature as the variables all required a lot of work to prepare for the model. However, since we are using of this is done for us! So, we will use some amazing features from this notebook Titanic - Advanc Feature Engineering Tutorial to better understand Pandas and to improve our model.

The feature will be shown in code and explained *afterwards*.

In [26]:
```
df['LogFare'] = np.log1p(df.Fare)
```

`log1p` is a function that takes the logarithm of `Fare + 1`. Instead of having to add , to p `log(0)` this function does that for us!

We use the 'LogFare' as this column has many really big values and lots of smaller values. This cause a skewed distribution - generally, machine learning (ML) models work better with more no distributions.

In [27]:
```
df['Deck'] = df.Cabin.str[0].map(dict(A='ABC', B='ABC', C='ABC', D='DE',
```

`df.Cabin.str[0]` extracts the first letter of the 'Cabin' column, which represents the deck wl passenger's cabin is locate. We use `map` to group certain decks together.

This line reduces the number of unique categories and may improve the predictive power by sim the model's input data.

In [28]:
```python
df['Family'] = df.SibSp + df.Parch
```

The 'Family' column is created by adding siblings and spouses and parents and children.

This column indicates that the sum of family members on board could be a predictor of a passer likelihood of survival.

In [29]:
```python
df['Alone'] = df.Family == 0
```

Similar to above, this line indicates that travelling alone is a predictor of survival. This will produc boolean value that fastai can automatically handle.

In [30]:
```python
df['TicketFreq'] = df.groupby('Ticket')['Ticket'].transform('count')
```

'TicketFreq' represents how many passesngers shared a ticket number. `groupby` allows us to rows with the same ticket number as a group so we can perform operations on these groups. `transform` is applied after grouping and is used to count the occurences of each ticket numb

`transform` is very useful for performing calculations within groups while retaining the original the DataFrame.

This feature indicates that people travelling on the same ticket may have had a similar experienc the journey impacting their likelihood of survival.

In [31]:
```python
df['Title'] = df.Name.str.split(', ', expand=True)[1].str.split('.', expa
df['Title'] = df.Title.map(dict(Mr='Mr', Miss='Miss', Mrs='Mrs', Master='
```

The first line takes the common format of each name to extract the Title and save it into the Data Each name has the format:

```
Surname, Title. Name
```

The second line maps the four valid titles (for the purposes of this model). This will remove titles 'Dr' or 'Rev' in case of invalid titles and replace them with `NaN`.

This feature indicates that a person's title, which is indicative of their age, gender and family stat predictor of their likelihood of survival.

We can inspect our final dataframe below with many added features to optimise our model. Also their has been no need to convert values into specific types or deal with `NaN` values as fastai w this for us!

In [32]:
```python
df.head()
```

Out[32]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
			Braund, Mr. Owen Harris	male	.			A
			Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	.			
			Heikkinen, Miss. Laina	female	.			STON/O
			Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	.			
			Allen, Mr. William Henry	male	.			

We will put all of above into a function, so that we can also call it on the test data.

In [33]:

```
def add_features(df):
    df['LogFare'] = np.log1p(df['Fare'])
    df['Deck'] = df.Cabin.str[0].map(dict(A="ABC", B="ABC", C="ABC", D="D
    df['Family'] = df.SibSp+df.Parch
    df['Alone'] = df.Family==0
    df['TicketFreq'] = df.groupby('Ticket')['Ticket'].transform('count')
    df['Title'] = df.Name.str.split(', ', expand=True)[1].str.split('.',
    df['Title'] = df.Title.map(dict(Mr="Mr",Miss="Miss",Mrs="Mrs",Master=
```

Now, we can split the data using the methods provided by fastai and load our data into `TabularPandas` which works with well... Tabular data from Pandas!

In [34]:

```
splits = RandomSplitter(seed=42)(df) # like this last time, we will use a
splits
```

Out[34]:

```
((#713)
[788,525,821,253,374,98,215,313,281,305,701,812,76,50,387,47,516,564,434,117
 (#178)
[303,778,531,385,134,476,691,443,386,128,579,65,869,359,202,187,456,880,705,
)
```

In [35]:

```
%%capture
dls = TabularPandas(df,
                    splits = splits,
                    procs = [Categorify, FillMissing, Normalize],
                    cat_names = ["Sex","Pclass","Embarked","Deck", "Title
```

```
                     cont_names = ['Age', 'SibSp', 'Parch', 'LogFare', 'Al
                     y_names = "Survived",
                     y_block = CategoryBlock()
).dataloaders(path='.')
```

- `splits` defines the indices of the training and validation sets.
- `procs` turns strings to categories, fills missing values with the median and nomralises nur columns.
- `cat_names` are the categorical variables.
- `cont_names` are the continuous variables.
- `y_names` is dependent variable.
- `y_block` defines the dependent variable as categorical, so we build a classification mode regression model.

`dataloaders` batches the data and applies any necessary transformations. `path` is where a temporary files or processed data will be saved.

Training the Model

Now that we have our `dataloaders` object it is *very* easy to create our model. We will make a `Learner` which is the data and the model (in this case, `tabular_learner` - it will pick us a one!). While we don't strictly have to define the `layers`, we can and we will. This means it will same as the last notebook, when we had hidden layers with activations each. We give it the `dataloaders` object and any metrics we want it to print on the way.

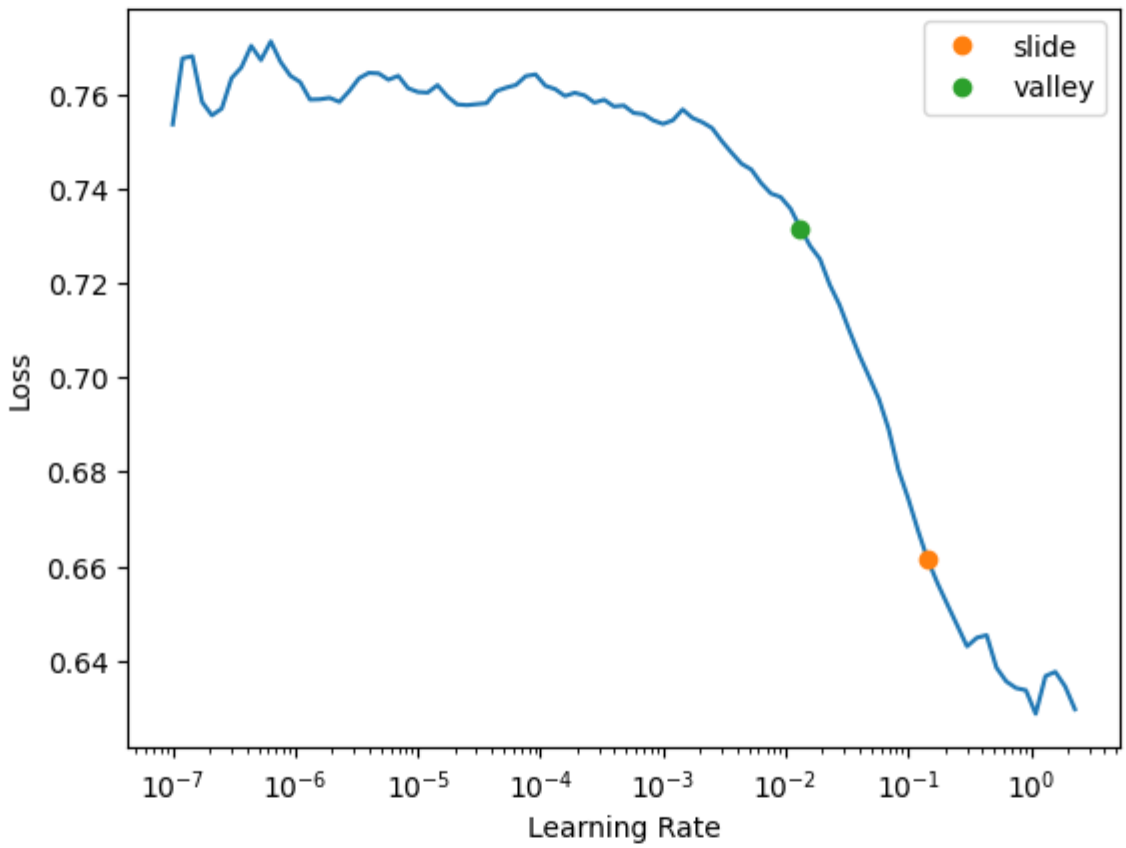
In [36]:
```
learn = tabular_learner(dls, metrics=accuracy, layers=[10,10])
```

We can also use some fastai's advanced features to help us select a learning rate, which would take experimentation.

In [37]:
```
learn.lr_find(suggest_funcs=(slide, valley))
```

```
/usr/local/lib/python3.11/dist-packages/fastai/learner.py:53: FutureWarning:
are using `torch.load` with `weights_only=False` (the current default value),
uses the default pickle module implicitly. It is possible to construct malici
pickle data which will execute arbitrary code during unpickling (See https://
github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more
details). In a future release, the default value for `weights_only` will be f
to `True`. This limits the functions that could be executed during unpickling
Arbitrary objects will no longer be allowed to be loaded via this mode unless
are explicitly allowlisted by the user via `torch.serialization.add_safe_glob
We recommend you start setting `weights_only=True` for any use case where you
have full control of the loaded file. Please open an issue on GitHub for any
related to this experimental feature.
  state = torch.load(file, map_location=device, **torch_load_kwargs)
```

Out[37]:
```
SuggestedLRs(slide=0.14454397559165955, valley=0.013182567432522774)
```



A good recommendation is to choose a value between slide and valley. In this case, we will use

. .

In [38]:
```
learn.fit(5, lr=0.01)
```

epoch	train_loss	valid_loss	accuracy	time
	.	.	.	:
	.	.	.	:
	.	.	.	:
	.	.	.	:
	.	.	.	:

We have achieved a similar accuracy to our 'from-scratch' model at `0.83` ! This is expected, co the dataset is small and a simple linear model was already working very well. Let's use some mc advanced techniques to improve this - even just a little bit.

Ensembling

Ensembling is the process of creating several separate models, each trained from different ranc starting points, and averaging their predictions. Since its very easy to make model, makin will not be hard either!

We use `learn.no_bar()` and `learn.no_logging()` to supress the training output for cle execution.

We use `[0]` as `get_preds()` also outputs target labels, which we don't need.

In [39]:
```
def ensemble():
  learn = tabular_learner(dls, metrics=accuracy, layers=[10,10])
  with learn.no_bar(), learn.no_logging():
    learn.fit(5, lr=0.001)
  return learn.get_preds(dl=tst_dl)[0]
```

Let's prepare our test data so that we can generate predictions using ensembling. While we do r
apply the same feature engineering to the test dataframe, all the preprocessing steps for the dat
saved in the `Learner` so we can pass it directly in usin `test_dl()` so it applies those steps

In [40]:
```
%%capture
tst_df = pd.read_csv('test.csv')
tst_df['Fare'] = tst_df.Fare.fillna(0) # account for some missing columns
add_features(tst_df)

tst_dl = learn.dls.test_dl(tst_df)
```

Now we can call our `ensemble()` function times, collecting the results into a list. As you see, the learns will contain tensors, each with the probability that the person survived and probability they did not - for each learner model.

In [41]:
```
learns = [ensemble() for _ in range(5)]
learns[0][:5]
```

Out[41]:
```
tensor([[0.7813, 0.2187],
        [0.6132, 0.3868],
        [0.8190, 0.1810],
        [0.7573, 0.2427],
        [0.5118, 0.4882]])
```

Using PyTorch, we will stack these preditions and take the mean across the th dimension. Tl
dimensions represents the number of models, the st dimension represents the number of pre
made for each model, and the nd dimension represents the number of categories.

By averaging across the th dimension (the different model), these represent our **final predictions**!

In [42]:
```
preds = torch.stack(learns)
print(preds.shape)
preds = preds.mean(0)
preds[:5]
```

```
torch.Size([5, 418, 2])
```

Out[42]:
```
tensor([[0.7468, 0.2532],
        [0.6302, 0.3698],
        [0.7439, 0.2561],
        [0.7272, 0.2728],
        [0.4543, 0.5457]])
```

If we wanted to export this to submit to the Kaggle competition, we would use the following code

In [43]:

```
tst_df['Survived'] = (preds[:,1]>0.5).int()
sub_df = tst_df[['PassengerId','Survived']]
sub_df.to_csv('ens_sub.csv', index=False)
```

Multi-target model

Now, to supplement our learning even further, we will create a Titanic model that can predict whether a person survived and where they debarked from. This is an example of a multi-target n

To build a model that can predict multiple things, we need a DataLoaders with dependent variables. We will use the `DataBlock` API which is one level down compared to the high level `TabularPandas` API. This allows more flexibility.

- `blocks` defines what kind of information is the

In [48]:
```
dls = DataBlock(
    blocks=(MultiCategoryBlock, CategoryBlock, CategoryBlock),
    n_inp=1,
    get_items=lambda: df,
    get_y=[lambda r: r.Survived, lambda r: r.Embarked],
)
dls =
dls.show_batch()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-48-db44ca7ce0ef> in <cell line: 0>()
      5     get_y=[lambda r: r.Survived, lambda r: r.Embarked],
      6 )
----> 7 dls.show_batch()

AttributeError: 'DataBlock' object has no attribute 'show_batch'
```

Conclusion

We've learnt some advanced deep learning techniques to build an optimal model that can predic person's survival of the Titanic based on statistics.

While this approach doesn't allow us to see the underlying code behind creating a model, it prod quicker and better results and makes use of the the extensive research and expertise that has g deep learning optimisation.