In [ ]:
```
%%capture
!pip install fastai
!pip install fastai.structured
```

In [ ]:
```
%%capture
import pandas as pd
from fastai.tabular.all import *
from fastai.data.transforms import IndexSplitter
```

Predicting Sticker Sales

Introduction

As I continue my machine learning (ML) journey, it seemed like I had accumulated sufficient kno
to participate in a Kaggle competition. As such, I decided to produce a model to submit to this
competition: Forecasting Sticker Sales.

Since I've spent a lot of time working through the examples provided in the wonderful fastai cour
decided it was time to produce some of my own work to demonstrate my knowledge acquisition
By using the course material on different data and using my own research, I have a chance to ch
knowledge and perform some creative problem-solving to make a solid submission.

In this Kaggle competition, the goal is to predict the number of sticker sales based on training da
includes past sticker sales as well as:

- `date`
- `country`
- `store`
- `product`

This competition is aimed to be approachable, for people to practice their ML skills. But, it still pr
real-world data and hidden test sets. I'm excited to demonstrate my ML knowledge through this
challenge!

So far, I've learnt **deep learning** as part of my machine learning journey, so this is the approach
take! However, it would also be worth attempting an **ensemble of decision trees** based approa
we are working with **structured data** (i.e., tabular data).

Data Extraction

I've manually uploaded `train.csv` and `test.csv` to the folder for this notebook to use for
the model. We will use Pandas, which is great for handling csv files and inspect the contents of t
dataframe.

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

Out[ ]:

	id	date	country	store	product	num
		- -	Canada	Discount Stickers	Holographic Goose	
		- -	Canada	Discount Stickers	Kaggle	
		- -	Canada	Discount Stickers	Kaggle Tiers	
		- -	Canada	Discount Stickers	Kerneler	
		- -	Canada	Discount Stickers	Kerneler Dark Mode	
...	...	...	...	...	...	
		- -	Singapore	Premium Sticker Mart	Holographic Goose	
		- -	Singapore	Premium Sticker Mart	Kaggle	
		- -	Singapore	Premium Sticker Mart	Kaggle Tiers	
		- -	Singapore	Premium Sticker Mart	Kerneler	
		- -	Singapore	Premium Sticker Mart	Kerneler Dark Mode	

rows × columns

In [ ]:
```
df.isnull().sum()
```

Out[ ]:

id	
date	
country	
store	
product	
num_sold	

dtype: int

The first thing that I've noticed is that the training dataset contains a number of `NaN` values. Ho
unusually, these values are only in the dependent variable.

While this is a situation I have not yet come across, I can immediately identify that unlike `NaN` v
our features, we can not fill these with the *mean* or the *mode*. While this technique is suitable for
features, as it allows the model to use the remaining data in the row, by filling our dependent var
with meaningless values, we will essentially be training it on incorrect values. This will not help o
model.

At this stage, I'm not sure how I can use this data to improve my model, so I will use only the lab
data for now. We have over , labelled pieces of data so we will not drastically im
model by removing an odd , rows (representing only ~ % of the dataset).

In [ ]:
```
df.dropna(axis=0, inplace=True)
df
```

Out[ ]:

	id	date	country	store	product	num
		- -	Canada	Discount Stickers	Kaggle	
		- -	Canada	Discount Stickers	Kaggle Tiers	
		- -	Canada	Discount Stickers	Kerneler	
		- -	Canada	Discount Stickers	Kerneler Dark Mode	
		- -	Canada	Stickers for Less	Holographic Goose	
...	...	...	...	...	...	
		- -	Singapore	Premium Sticker Mart	Holographic Goose	
		- -	Singapore	Premium Sticker Mart	Kaggle	
		- -	Singapore	Premium Sticker Mart	Kaggle Tiers	
		- -	Singapore	Premium Sticker Mart	Kerneler	
		- -	Singapore	Premium Sticker Mart	Kerneler Dark Mode	

rows × columns

In [ ]:
```
df.isnull().sum()
```

Out[ ]:

id	
date	
country	
store	
product	
num_sold	

dtype: int

Feature Engineering

Perfect! The `NaN` values are removed. Now let's take a deeper dive into our data using the Par
`describe()` method.

In [ ]:
```
df.describe(include='all')
```

Out[ ]:

	id	date	country	store	prod
count	.				
unique	NaN				
top	NaN	- -	Finland	Premium Sticker Mart	Kag
freq	NaN				
mean	.	NaN	NaN	NaN	N
std	.	NaN	NaN	NaN	N
min	.	NaN	NaN	NaN	N
%	.	NaN	NaN	NaN	N
%	.	NaN	NaN	NaN	N
%	.	NaN	NaN	NaN	N
max	.	NaN	NaN	NaN	N

At the moment, the `date` field cannot be fed into the model. Furthermore, in its current format,
provides insight into whether one sale occured before or after another sale. But dates can provic
more detail than this and fastai provides a suite of methods to deal with `datetime` features.

We will use the `add_datepart()` function I found online that is provided by fastai. This functi
several useful features to the dataset including whether it is the start or end of the year or quarte
more.

It is important to perform **feature engineering** on the date feature as the date that the sales orig
likely to have the biggest impact on sales. For example, people may be more likely to make stick
purchases towards the end of the year when Christmas is. This means we're letting the model kr
more than just whether a date is more or less recent than another!

In [ ]:
```
add_datepart(df, 'date')
df
```

```
/usr/local/lib/python3.11/dist-packages/fastai/tabular/core.py:25: UserWarnin
argument 'infer_datetime_format' is deprecated and will be removed in a futur
version. A strict version of it is now the default, see https://pandas.pydata
pdeps/0004-consistent-to-datetime-parsing.html. You can safely remove this
argument.
  df[date_field] = pd.to_datetime(df[date_field], infer_datetime_format=True)
```

Out[ ]:

	id	country	store	product	num_sold	Year	Month
		Canada	Discount Stickers	Kaggle	.		
		Canada	Discount Stickers	Kaggle Tiers	.		
		Canada	Discount Stickers	Kerneler	.		
		Canada	Discount Stickers	Kerneler Dark Mode	.		
		Canada	Stickers for Less	Holographic Goose	.		
...	...	...	...	...	...	...	...
		Singapore	Premium Sticker Mart	Holographic Goose	.		
		Singapore	Premium Sticker Mart	Kaggle	.		
		Singapore	Premium Sticker Mart	Kaggle Tiers	.		
		Singapore	Premium Sticker Mart	Kerneler	.		
		Singapore	Premium Sticker Mart	Kerneler Dark Mode	.		

rows × columns

We've easily generated several useful features from the date column to feed into our model!

Let's have a look at the variables now and see how many distinct values they have. This will hel
determining what is a **continuous** vs a **categorical** variable.

```
In [ ]: df.nunique()
```

```
Out[ ]:
```

id
country
store
product
num_sold
Year
Month
Week
Day
Dayofweek
Dayofyear
Is_month_end
Is_month_start
Is_quarter_end
Is_quarter_start
Is_year_end
Is_year_start
Elapsed

dtype: int

It seems like our only continuous variables will be `Dayofyear` and `Elapsed`. It is possible th
of our other variables like `Week` or `Day` could be passed as continuous variables, but since th
a smaller number of distinct values we will attempt to use them as categorical variables.

In my first attempt, I performed no further feature engineering at this point.

However, the model was scoring extremely poorly, with a Mean Absolute Percentage Error (MA
metric defined by the Kaggle competition) of approximately `1.0`. After doing some researching
and relying on my own intuition, I determined that the extremely large (and sometimes extremely
values of `num_score` were likely impacting the model. A value like MAPE will be very sensitive
discrepancies and therefore outliers.

In [ ]:
```
df['num_sold'].min(), df['num_sold'].max()
```

Out[ ]:
```
(5.0, 5939.0)
```

Having a look here, we can see that in one row of the dataframe only stickers were sold! E maximum values is almost stickers. This big difference was causing significant failir model and metric.

As such, I performed the following **log transformation** to the `num_sold` column. The indepen variables (or features) will be automatically normalised by the fastai `dataloaders` object so tl not require log transformations.

In [ ]:
```
df['num_sold'] = np.log1p(df['num_sold'])
```

By taking the log of `num_sold` now and using the model to predict `log(num_sold)`, we car transform out predictions back into `num_sold` using the exponential function.

This simple feature engineering improved our MAPE from `1.0` to `0.1`!

At this stage our tabular data is ready, we will just drop the `id` column since it is not relevant to model. It is a unique value from `0` to `n` and in the same order as `date`.

In [ ]:
```
df.drop('id', axis=1, inplace=True)
```

In [ ]:
```
df
```

Out[ ]:

	country	store	product	num_sold	Year	Month	Week	D
	Canada	Discount Stickers	Kaggle	.				
	Canada	Discount Stickers	Kaggle Tiers	.				
	Canada	Discount Stickers	Kerneler	.				
	Canada	Discount Stickers	Kerneler Dark Mode	.				
	Canada	Stickers for Less	Holographic Goose	.				
...	...	...	...	...	...	...	...	
	Singapore	Premium Sticker Mart	Holographic Goose	.				
	Singapore	Premium Sticker Mart	Kaggle	.				
	Singapore	Premium Sticker Mart	Kaggle Tiers	.				
	Singapore	Premium Sticker Mart	Kerneler	.				
	Singapore	Premium Sticker Mart	Kerneler Dark Mode	.				

rows × columns

Validation Set

An important learning that I've made so far is that choosing a suitable and meaningful validation
extremely important when training a model. While it is tempting to use a random split of the data
situation, it is likely unideal. Since we will be using the model to predict **future** sales, it makes se
the same with the validation set.

As such, I intend on trimming the later sales in the dataset to use as the validation set and the sa
were made before then can be used to fit the parameters.

In [ ]:

```
df.groupby("Year").count()
```

Out[ ]:

	country	store	product	num_sold	Month	Week	Da
Year							

Having a look at the number of rows for each year, it makes sense to split the model using:

```
sold in 2016 vs ~sold in 2016
```

This means of our , rows of data, our validation set will represent ~ % of ou at , values. This is a reasonable split, especially given the reasonably large amou we have access to.

In [ ]:
```
df = df.reset_index(drop=True)  # Ensure the index is sequential

val_i = df[df['Year'] == 2016].index # indices of validation data

splitter = IndexSplitter(val_i)
splits = splitter(df)

len(splits[0]), len(splits[1])
```

Out[ ]:
```
(189492, 31767)
```

Seems like a great split!

Let's make our `dataloaders` object now! We don't need to specify `y_block` as fastai infers automatically. However, we will need to specify our continuous and categorical variables (which already discussed above). We will also need to specify our dependent variable.

In [ ]:
```
cont_names = ['Dayofyear', 'Elapsed']
cat_names = [i for i in df.columns if i not in cont_names + ['num_sold']]
cont_names, cat_names
```

Out[ ]:
```
([ 'Dayofyear', 'Elapsed'],
 ['country',
  'store',
  'product',
  'Year',
  'Month',
  'Week',
  'Day',
  'Dayofweek',
  'Is_month_end',
  'Is_month_start',
  'Is_quarter_end',
  'Is_quarter_start',
  'Is_year_end',
  'Is_year_start'])
```

In [ ]:
```
dls = TabularPandas(df,
                    splits=splits,
                    procs=[Categorify,Normalize],
                    cont_names=cont_names,
                    cat_names=cat_names,
                    y_names='num_sold'
).dataloaders(path='.')
```

Upon analysing the specificities of the Kaggle competition, it is noted that the metric with with the are evaluated is the **Mean Absolute Percentage Error** (MAPE).

Since this is not provided by default by fastai, I have created my own function to deal with this.

In [ ]:
```
from fastai.metrics import mae

# Define MAPE as a simple function
def mape(preds, targs):
    epsilon = 1e-7  # To avoid division by zero
    return ((targs - preds).abs() / (targs.abs() + epsilon)).mean()
```

Fit the Model

Now that we have both:

- a metric, and,
- a dataloaders object with training and validation sets,

we can create a `learner` using `tabular_learner` provided by fastai. This will choose a su architecture depending on our data.

In [ ]:
```
learn = tabular_learner(dls, metrics=[mape])
```

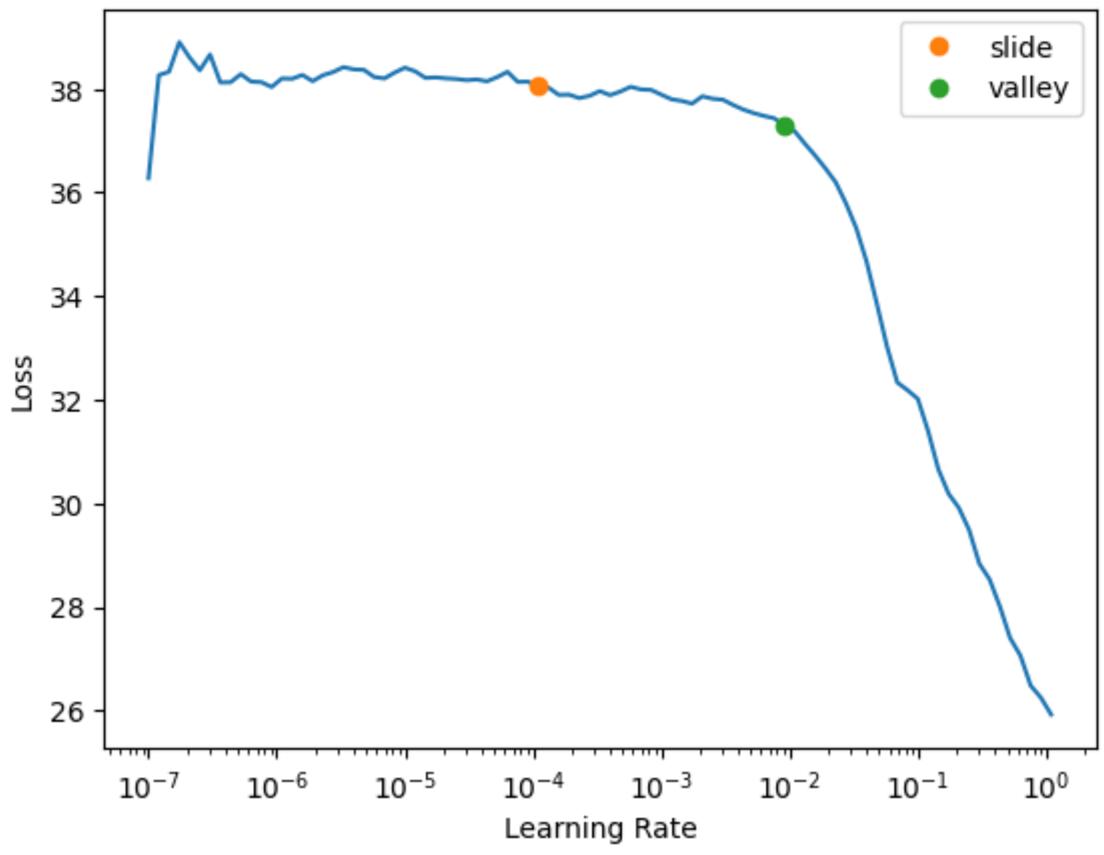
Now that we have a `learner` let's use some of fastai's advanced features to choose the best l rate.

In [ ]:
```
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[ ]:

```
SuggestedLRs(slide=0.00010964782268274575, valley=0.009120108559727669)
```



I'm choosing a **learning rate** around the value of *valley*. This is generally a conservative estimat suitable learning rate, which is supposed to prevent **overfitting**.

In [ ]:

```
learn.fit(5, lr=0.01)
```

epoch	train_loss	valid_loss	mape	time
.	.	.	:	
.	.	.	:	
.	.	.	:	
.	.	.	:	
.	.	.	:	

Our `learner` model is performing well. We are achieving a **MAPE** of around . whi
excellent and the model is training in around seconds per epoch.

However, we can definitely see overfitting occuring with this model, as our loss function begins g
worse after epoch . This is something we will address with **weight decay** in the next model.

Pseudo-Labelling

While the model is already performing very well, why not aim even higher?!

When considering ways of improving my model, it is clear that I could experiment with a range o
hyperparameters and **architectures**. However, I think the single biggest improvement I could m
straight away is to use the remaining data that I removed from the dataframe at the beginning! W
data was unlabelled, it still contained useful information to fit the model and using **semi-supervi
learning** we can make the most of **all** the data that was provided.

To perform **pseudo-labelling** we will use our trained model to predict `num_sold` for the rows w
this data is missing. These preditions will be treated as pseduo-labels and combined back into th
training dataset.

Then, we will retrain the model on the combined dataset.

First, we will extract the unlabelled data from the `train.csv` file and perform the same transfo
on the `date` column.

In [ ]:
```
unlabelled = pd.read_csv('train.csv')
unlabelled = unlabelled[unlabelled['num_sold'].isnull()]
unlabelled.drop(['num_sold','id'], axis=1, inplace=True)
add_datepart(unlabelled, 'date')
unlabelled
```

```
/usr/local/lib/python3.11/dist-packages/fastai/tabular/core.py:25: UserWarnin
argument 'infer_datetime_format' is deprecated and will be removed in a futur
version. A strict version of it is now the default, see https://pandas.pydata
pdeps/0004-consistent-to-datetime-parsing.html. You can safely remove this
argument.
  df[date_field] = pd.to_datetime(df[date_field], infer_datetime_format=True)
```

Out[ ]:

	country	store	product	Year	Month	Week	Day	Dayofweek	Da
	Canada	Discount Stickers	Holographic Goose						
	Kenya	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Kenya	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
...	...	...	...	...	...	...	...	...	
	Kenya	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Kenya	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Kenya	Discount Stickers	Holographic Goose						

rows × columns

Now we will make predictions using this data.

In [ ]:
```
nolabel_dl = learn.dls.test_dl(unlabelled)

preds,_ = learn.get_preds(dl=nolabel_dl)
preds
```

Out[ ]:
```
tensor([[5.0238],
        [1.7283],
        [5.0338],
        ...,
        [1.8684],
        [5.0092],
        [1.9257]])
```

In the next step, we are going to sort through our pseudo-labelled data and only choose the valu the highest **confidence**. We will select a **threshold** value of `0.7` and any prediction which the makes which is less confident then this will not be used in training the new model.

> The "confidence" measures how close the prediction is to the baseline, which is the mea
> value of the `num_sold` column. The closer the prediction is to the baseline, the higher
> the confidence.

In [ ]:
```
baseline_value = df['num_sold'].mean()
confidence = 1 - (abs(preds.squeeze() - baseline_value) / baseline_value)
unlabelled['num_sold'] = preds.squeeze().numpy() # removes the dimensions
unlabelled['confidence'] = confidence.numpy() # create new confidence col
```

We can see below that our `unlabelled` dataframe contains a column specifying the confiden
model's prediction.

In [ ]:
```
unlabelled
```

Out[ ]:

	country	store	product	Year	Month	Week	Day	Dayofweek	Da
	Canada	Discount Stickers	Holographic Goose						
	Kenya	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Kenya	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
...	...	...	...	...	...	...	...	...	
	Kenya	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Kenya	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Kenya	Discount Stickers	Holographic Goose						

rows × columns

Now we can sort through our dataframe, removing values that are below the predefined threshol

In [ ]:
```
threshold = 0.7
high_conf_df = unlabelled[unlabelled['confidence'] >= threshold]
high_conf_df
```

Out[ ]:

	country	store	product	Year	Month	Week	Day	Dayofweek	Da
	Canada	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
...	...	...	...	...	...	...	...	...	
	Canada	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						
	Canada	Discount Stickers	Holographic Goose						

rows × columns

This data is ready to fed back into the model to as **pseudo-labelled** data. We will combine all ou
into a single dataframe and make the same validation split ready to update our dataloaders obje

In [ ]:

```
combined = pd.concat([df, high_conf_df])
combined.drop("confidence", axis=1, inplace=True)
combined = combined.reset_index(drop=True)
val_i = combined[combined['Year'] == 2016].index
splits = IndexSplitter(val_i)(combined)
splits
```

Out[ ]:

```
((#192959) [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19...],
 (#32546)
[189492,189493,189494,189495,189496,189497,189498,189499,189500,189501,1895C
3,189504,189505,189506,189507,189508,189509,189510,189511...])
```

Now we can create our `dataloaders` object with all of our data (labelled and pseudo-labelled
will update the `dls` in our `learn` model.

In [ ]:

```
dls_combined = TabularPandas(
    combined,
    splits=splits,
    procs=[Categorify, Normalize],
    cont_names=cont_names,
    cat_names=cat_names,
    y_names='num_sold'
).dataloaders(path='.')

learn = tabular_learner(dls_combined, metrics=[mape])
```

Since our last model experienced overfitting, I am taking extra precautions to avoid overfitting, re
the **learning rate** to . .

In [ ]:
```
learn.fit(5, 0.001, wd=0.01)
```

epoch	train_loss	valid_loss	mape	time
	.	.	.	:
	.	.	.	:
	.	.	.	:
	.	.	.	:
	.	.	.	:

By using pseudo-labelled data, we have improved our MAPE from . to .
learning rate has a much more consistent decline.

However, we've also made use of an extra hyperparameter called **weight decay** or **L** **regula**
This hyperparameter is discussed further in my collaborative filtering notebook but essentially:

- a higher weight decay reduces overfitting but may make the model struggle to capture mear patterns.
- a lower weight decay could cause the model to overfit, but it will be better at capturing patte data.

FastAI generally computes a suitable weight decay for us, but with tabular data it can struggle to understand our data and make a good prediction. As such, I experimented with a few different w decays prior to find a good one.

Let's make our predictions and submit to the Kaggle competition.

Making Predictions

First, we need to extract our **test data** and perform the same preprocessing. Then we can create dataloaders object for the test data.

In [ ]:
```
%%capture
test_df = pd.read_csv('test.csv')
add_datepart(test_df, 'date')
test_dl = learn.dls.test_dl(test_df)
```

Finally, we can generate predictions using our `learn` model. We mustn't forget that the model outputting the predictions as a log, as we took a **log transformation** of the labels in the training Before submitting our predictions, we must take the exponent of our outputs.

In [ ]:
```
preds = learn.get_preds(dl=test_dl)
test_df['num_sold'] = preds[0].numpy()
test_df['num_sold'] = np.expm1(test_df['num_sold'])
```

In [ ]:
```
sub_df = test_df[['id','num_sold']]
sub_df.to_csv('submission.csv', index=False)
```

Conclusion

After making our Kaggle submission our **MAPE on the test set was .** , which is high our training data but expected, since it was generated on the `num_sold` predictions without th transformation.

This score put us in the top % of Kaggle submissions, which is a terrific effort for our first competition. It was also terrific to independently learn a new technique like pseudo-labelling. I'm to continue to learn new techniques so I can score even higher next time!

UPDATE: After the competition ended and the final results were calculated, our submission mov places in the leaderboard. We ended up in the top % of submissions in position out of , with a MAPE of . .