Assignment 2: Jane Street Market Prediction

> IL181.025 | **Prof. Ribeiro**

Sho Hihara and Anirudh Nair

Link to Notion Document: https://www.notion.so/nairspace42/Assignment-2-Jane-Street-Market-Prediction-d849525256934b5b950ab36be832e2f4

Introduction

For this assignment we tackled a classification task provided by the Jane Street Market Prediction Kaggle Competition.

The aim is to build a quantitative trading model which maximizes returns using market data from a major global stock exchange (this dataset was provided by the competition). The model must decide whether to accept or reject a potential trading opportunity, making this a binary classification task.

Data

Adapted from competition website:

> """
>
> This dataset contains an anonymized set of features, `feature_{0...129}`, representing real stock market data. Each row in the dataset represents a trading opportunity, for which you will be predicting an `action` value: 1 to make the trade and 0 to pass on it. Each trade has an associated `weight` and `resp`, which together represents a return on the trade. The `date` column is an integer which represents the day of the trade, while `ts_id` represents a time ordering. In addition to anonymized feature values, you are provided with metadata about the features in **features.csv**.
>
> In the training set, **train.csv**, you are provided a `resp` value, as well as several other `resp_{1,2,3,4}` values that represent returns over different time horizons. These variables are not included in the test set. Trades with `weight`

`= 0` were intentionally included in the dataset for completeness, although such trades will not contribute towards the scoring evaluation.

"""

Files

- **train.csv** - the training set, contains historical data and returns
- **example_test.csv** - a mock test set which represents the structure of the unseen test set. *You will not be directly using the test set or sample submission in this competition, as the time-series API will get/set the test set and predictions.*
- **example_sample_submission.csv** - a mock sample submission file in the correct format
- **features.csv** - metadata pertaining to the anonymized features

Exploratory Analysis

The main training dataset was formatted in the following order, and using the df.head() function, we receive the following subset of the dataset.

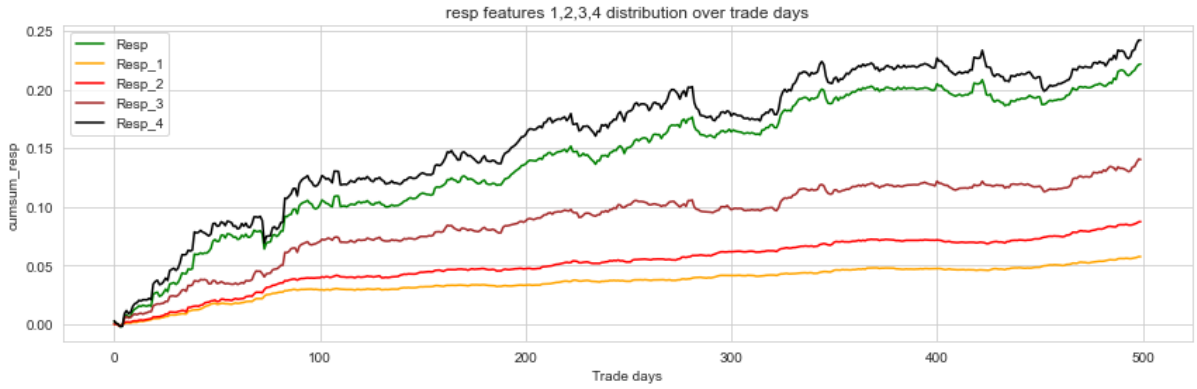
	date	weight	resp_1	resp_2	resp_3	resp_4	resp	feature_0	feature_1	feature_2	...	feature_122	feature_123	feature_124	feature_125	feature_126	feature_127	feature_128	feature_129	ts_id
1	0	16.671875	-0.002829	-0.003227	-0.007320	-0.011116	-0.009789	-1	-1.349609	-1.705078	...	-1.178711	1.777344	-0.915527	2.832031	-1.416992	2.296875	-1.304688	1.898438	1
4	0	0.138550	0.001252	0.002165	-0.001216	-0.006218	-0.002604	1	-3.171875	-3.093750	...	0.344971	4.101562	0.614258	6.625000	0.800293	5.234375	0.362549	3.925781	4
6	0	0.190552	-0.001939	-0.002300	0.001088	0.005962	0.000709	-1	-3.171875	-3.093750	...	0.336914	4.078125	0.614746	6.621094	0.800781	5.230469	0.361572	3.921875	6
7	0	3.820312	0.017395	0.021362	0.031158	0.036957	0.033478	-1	0.446045	-0.466309	...	2.101562	4.847656	1.479492	5.261719	2.304688	4.570312	2.201172	4.429688	7
8	0	0.116577	-0.005459	-0.007301	-0.009087	-0.003546	-0.001678	1	-3.171875	-3.093750	...	1.538086	4.785156	1.637695	6.968750	2.353516	5.824219	1.778320	4.742188	8

Figure 1. Overview of **train.csv**. The `date` column, which takes value from 0 to 500, represents an anonymized record of the day of each trade. The 5 `resp` columns represent the returns on that trade and adjusted using the `weight` values for final scoring. 130 anonymized `feature` columns characterize each trade opportunity.

Resp Values

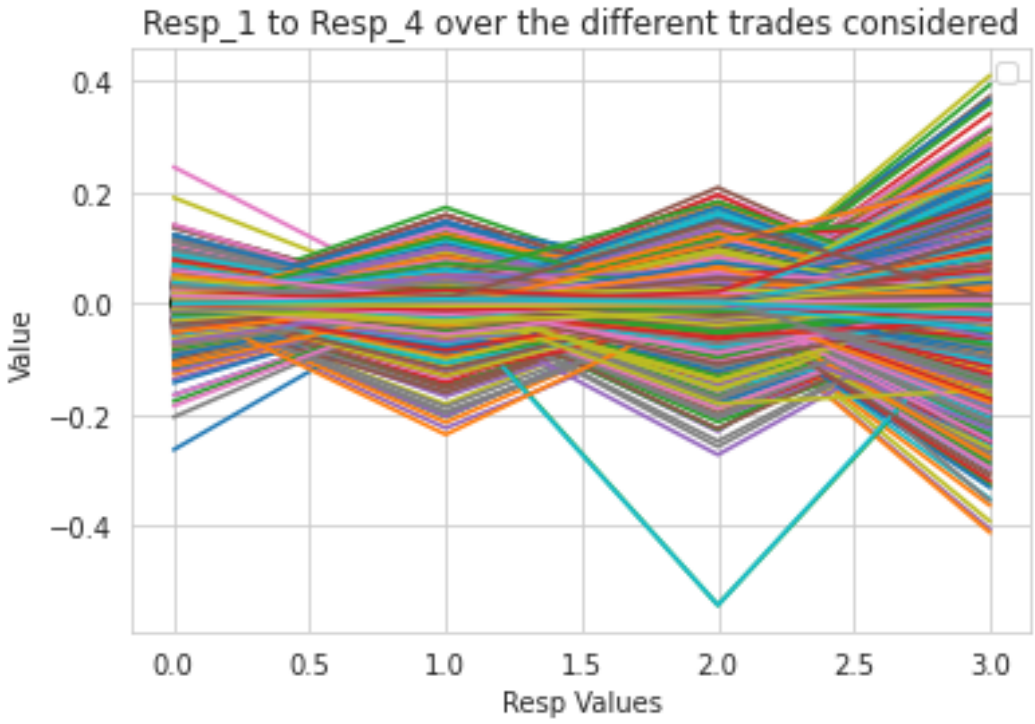
The competition defined the resp values as "values that represent returns over different time horizons" and thus it was immedietely clear to us that predicting the action of trading or not would be hinged on the behaviour that is shown by the Resp values. We hence first plotted the progression of different Resp

values over time to see if there was some kind of inference we could draw from that behavior.



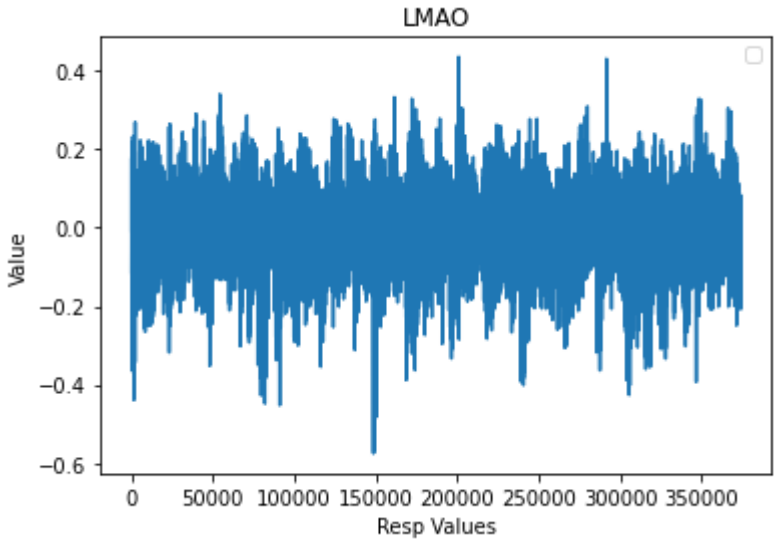
Through a quick analysis of the figure, we can observe that `resp` and `resp_4` values are very closely related and since all the Resp values stand to represent returns of different time horizons, it would be important to consider the trajectory of `resp_1` to `resp_4` for each trade that is considered to inspect for interesting patterns that the progression of these values show for different considerations.

We hence produce the following plot to further analyse Resp behavior.



From the above plot, we see that the variance of the Resp values increases over every iteration, and `resp_4`, which is closely tracking Resp, is the most variant while `resp_1` is the least variant. We infer from this that `resp_1` values are returns predicted over the nearest time frame while `resp_4` represents the returns over the farthest timeframe, and hence hypothesize a mechanism to classify trading actions based on the difference between these two values. A high positive difference between `resp_4` and `resp_1` indicates procurement of a higher return over time, and hence, we explore the distribution of this difference over the different trades considered to come up with a threshold that will allow us to only execute highly profitable trades.

We thus produce the following visualization that plots the required difference for each of the trade considered.



Through this distribution, we choose a difference of 0.1 for the `resp` values to qualify a trade for execution. Using this threshold, we create an additional column in our training dataset called action which indicates if a trade was executed using 1 or not executed using 0. Hence, for each of the trades considered, the value is set to 1 if the difference is above 0.1 and 0 otherwise. This action column will now be used to conduct relevant feature engineering for the 130 features provided to us and build our final supervised classification model.

Features

We had to "play" with the features for some time before really doing anything with it. Here we provide some key findings that were instrumental to the

feature engineering process which we discuss later.

As a preliminary analysis, we wanted to see how many trade opportunities were available each day. Below is a plot showing the different number of transactions provided per day, in the dataset.

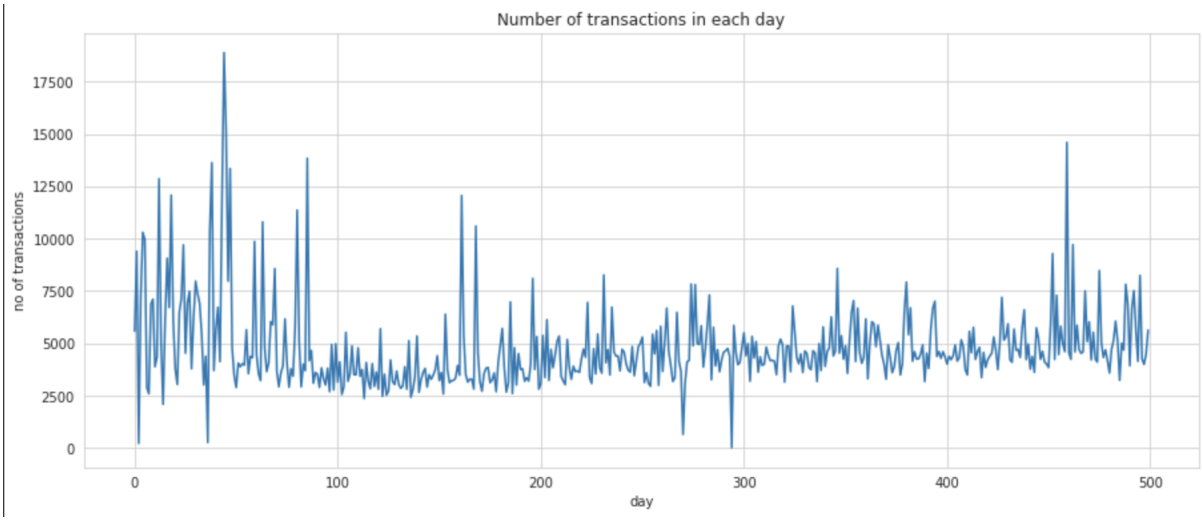


Figure 3. The number of trade opportunities provided per day. It seems quite unstable in the first 100 days or so, but maintains overall stability for the rest of the time period, albeit with occasional spikes and drops.

While the transaction count remains somewhere around 5000 per day, there is also very clear short-term volatility; some days go as high as 12500, and some other days go as low as 2500 or even less.

Let's take a look at the distribution of the transaction count:

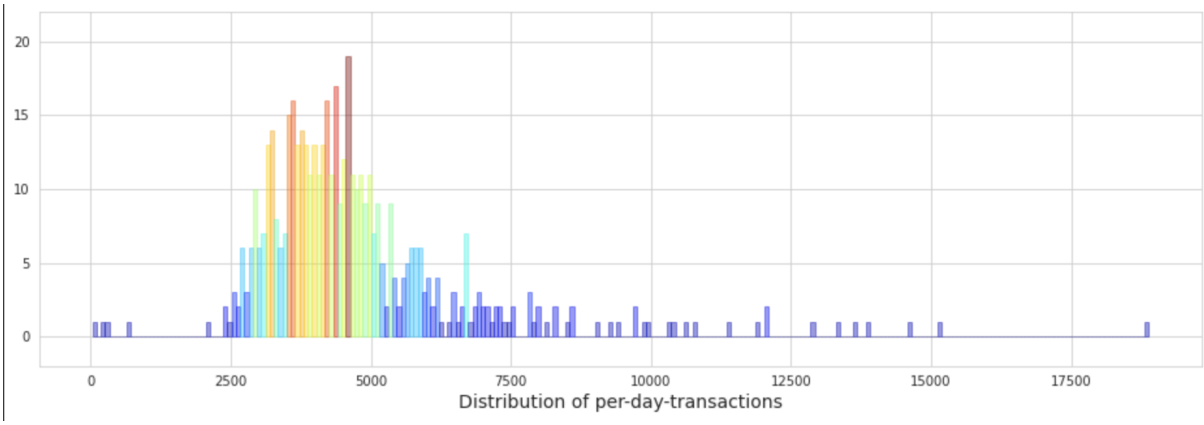


Figure 3. Distribution of the number of trade opportunities provided each day. The range is quite wide, and there are some extreme outliers, but still maintains a fairly normal distribution.

Taking three examples, one from a day with around 2500 trades, one from a day with around 4000 trades, and one from a day with around 7500 trades; we examine whether there is some sort of pattern that is consistent across all dates.

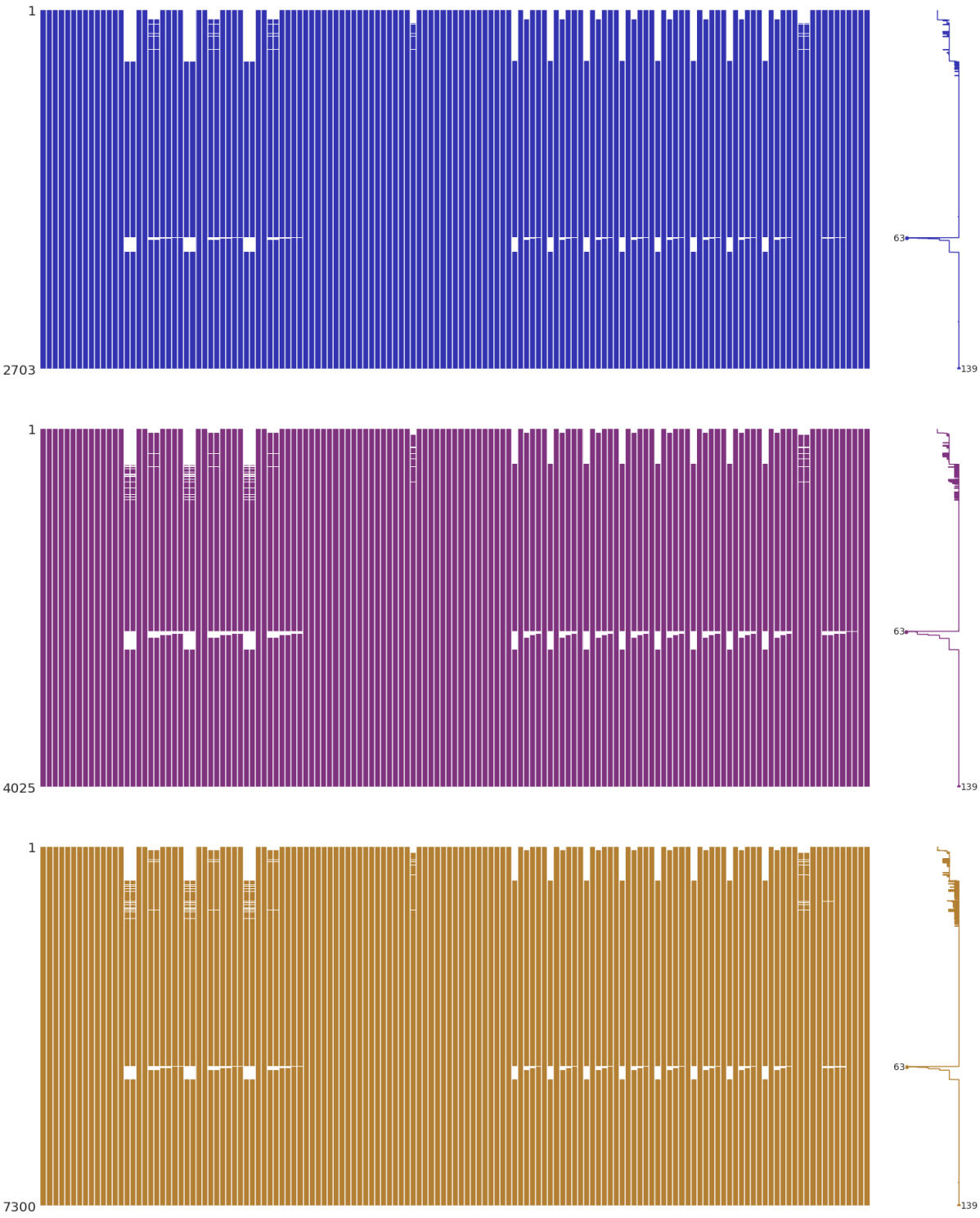


Figure 4. Presence/absence of data for opportunities presented on day 96, day 242, day 454, respectively. Day 96 has 2703 transactions, in the lower end of the spectrum; day 242 has 4025, around median; and day 454 has 7300 transactions, on the higher end.

These three figures plot the presence/absence of data for three different dates, each with low/medium/high volume of trading during that day. It shows us how there is a consistent pattern in the missing data regardless of the trading volume. This means that we should deal with these missing values with intention, rather than some average value of the entire column, or a convenient 0, or something of the kind. Specifically, they shouldn't be:

(a) mean values or 0

(b) any kind of uniform ffill method or something

(c) dropped altogether.

To devise a good filler value that meets the above requirements, we further examined the distribution of values at each feature. The box plots below show such distribution for all features with less than 5000 values.

Features having less than 5,000 missing values

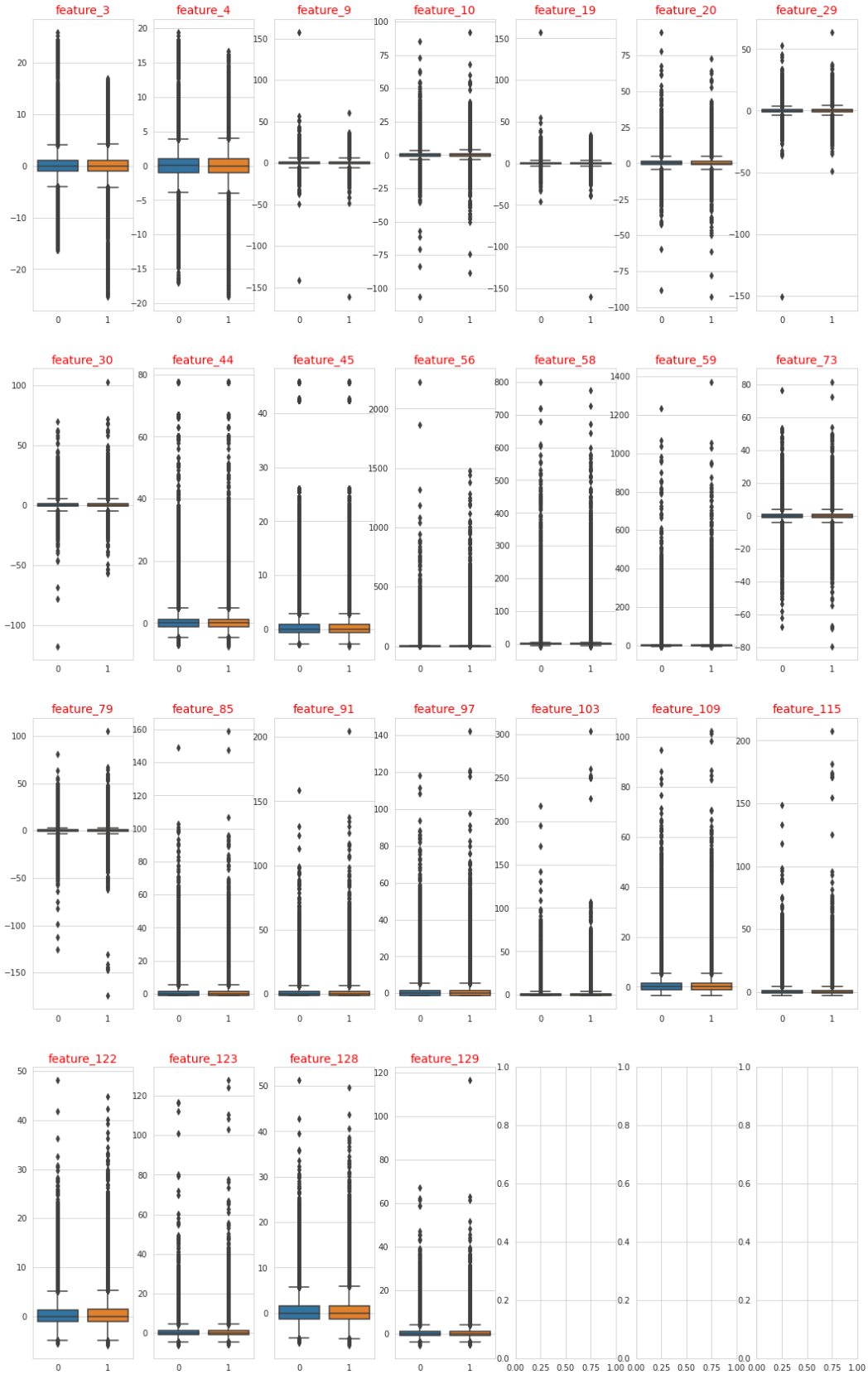


Figure 5. Box plot depicting the numerical distribution of values under each feature. The same process was repeated for features with 5000<missing values<10000, and 10000<missing values<50000.

From these plots, we see that:

- the mean of pretty much all features lies at/around zero.
- Many of the features do not have negative outliers. (all of them have positive outliers)

Therefore, here's an idea which we implemented for a filler value that would probably work better than simple fillers:

```
val_range = train_file[features].max() - train_file[features].min()
train_file[features].min()-0.01*val_range
```

Feature Engineering

At this point, we have some understanding of the data, as well as having filled the missing data appropriately. The next step is to understand the anonymized 130 `feature`s better so that we can clean the dataset further for optimal modeling. At this stage, we wish to reduce/combine redundant columns, mainly to reduce noise and the dimensionality of the dataset. To do this, we use a toy model to get an overview of how different features influence the decision that a classifier would make. In our case, we use a Random Forest classifier to identify the simple correlation between each feature to the transaction return. Random Forest classifier was chosen for this task as it is one of the most basic classification algorithms that can provide a baseline insight into the information of our interest; which feature seems to be influencing the returns more than others.

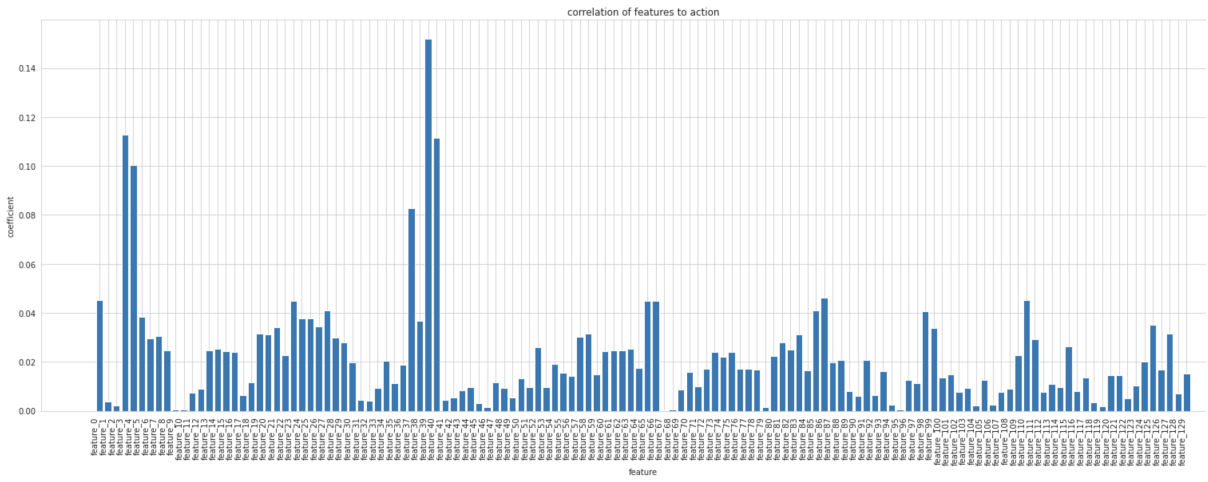


Figure 6. A simple bar graph illustrating the correlation between each feature and the `action` value, which is defined as 1 when `resp` is positive and 0 when `resp` is negative. This provides preliminary insight into which features we should be paying particular attention to.

SHAP Values

Before we get to the depth of how we engineered our features for optimized results, we need to go over the concept of Shap values, which was an important deciding factor for us to know which of the features and feature combinations are relevant to consider for our final model.

SHAP Values, which stand for SHapley Additive exPlanations, break down a prediction made by the model to show the contribution of each feature on the prediction. They do this by computing the impact of having a certain value for a given feature in comparison to the prediction the model would make if that feature had another baseline value.

Mathematically, the Shapley value $\phi_i(p)$ for a particular feature `i` out of `n` total features, for a model prediction `p` is calculated using the following equation:

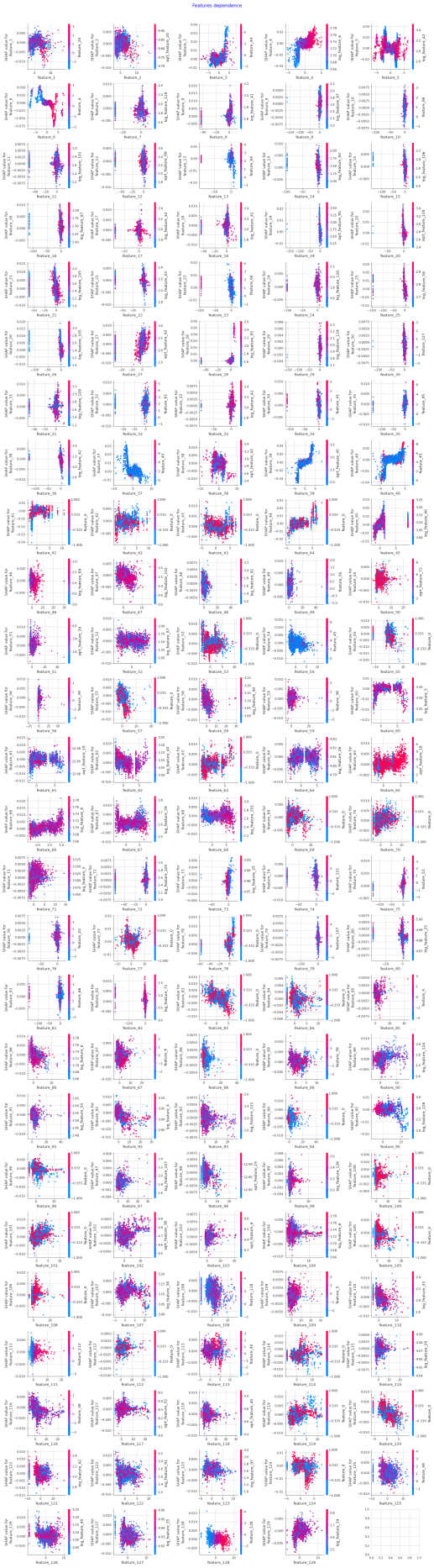
$$\phi_i(p) = \sum_{S \subseteq N/i} \frac{|S|!(n - |S| - 1)!}{n!} (p(S \cup i) - p(S))$$

Wherein S is the set of all possible combinations of features without `i` . Hence, even though the equation seems complicated, what it fundamentally does is

calculate the weighted average of the effect removing i from the list of features that the model uses to predict will have on the accuracy of the prediction and repeat this process for all possible combinations of features (including removing different features) to provide an importance score for the feature. In simple words:

$$importance\ of\ feature\ I = accuracy\ of\ model\ p\ with\ I - accuracy\ of\ model\ p\ without\ I$$

Now we move onto an analysis of the contributions that the different anonymized features make to our toy exploratory model using SHAP values to determine which features our model should focus on.


Features dependence

Using SHAP scores for each feature against our created "action" column, we take a look at feature dependence. With this information, we identify which features can be reduced/combined/multiplied with each other. The process of identifying from the plot which features could be synthesized was entirely manual, with a few heuristics utilized.

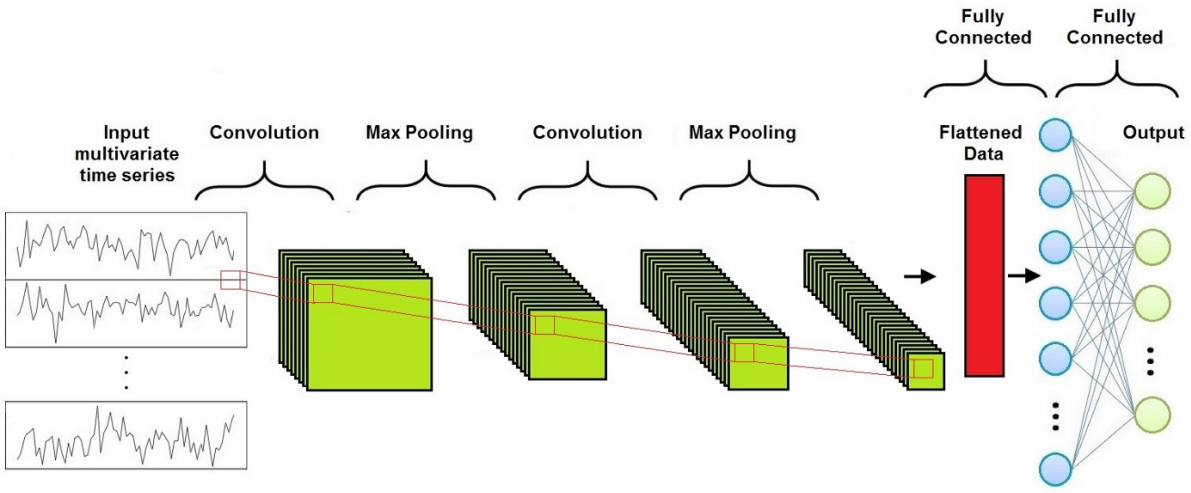
First is the scale SHAP values provided per feature. Some features have SHAP values ranging from 0.05 to -0.075, which indicate a strong presence affecting the final return. On the other hand, some other features only range from 0.005 to -0.0075, which are effectively just noise, in comparison to the more relevant features.

Another heuristic is the separation of color in each plot. Two features with very well separated colors indicate a strong relationship between the two variables. Take a look at feature_128 against feature_126 for a clear example of this separation.

The synthesized features were then added as columns accordingly, to provide a better signal to the model later.

Model Selection and Implementation

Given the exploratory analysis and feature engineering, we tested several different machine learning approaches to perform this binary classification, and after comparing the accuracy of different models (we considered simple models like Random Forest Classifiers, SVM classifiers and different Neural Network based classifiers such as Long Short Term Memory Recurrent Neural Networks (LSTMs) and Convolutional Neural Networks) we settled down with a CNN.



A diagramatic representation of how CNNs would work with multivariate time series inputs(Marco Del Pra, 2020).

Convolutional Neural Networks are deep learning algorithms that usually takes images as input, but in our case it used multivariate time series data to isolate and exploit the various spatial and temporal patterns it identifies using trainable filters on the series' image, and then it assigns importance to each of the pattern it recognises using trainable weights to provide us with a strong classification . Many different neural networks use filters to identify and isolate patterns, but these filters have to be tweaked and engineered case by case. Instead, Convolutional Neural Networks have the ability to learn and weigh the filters as required on their own without much supervision. Hence, CNNs can themselves detect which features are important in the case of our anonymized dataset. Also, the amount of pre-processing required in a Convolutional Neural Network is much lower as compared to other classification algorithms. These two factors were particularly important to our choice of using CNN as our final model given the anonymized dataset.

It is important to note that we provided our CNN with an optimized dataset in which several features were removed or combined as per our prior analysis, and then our CNN conducted further feature engineering internally to optimize the prediction of the model.

Two metrics were computed for evaluating the model; standard accuracy and the area under the ROC curve. The standard accuracy computes, well, how much of the prediction was correct and how much of them are not. However,

AUC is also relevant because it is a more nuanced metric that provides insight into the tradeoff between true positives and false positives. This is particularly important when we do not know the true ratio between accept and reject; for example if 99% of trade should be rejected, a model that blindly rejects all opportunities will still yield 99% accuracy. Hence, both of these evalulation metrics helped us fine tune the different hyperparameters that was required to run the model.

```
def build_model():
    model = keras.models.Sequential()
    model.add(Conv1D(180, 2, input_shape=x_train.shape[1:]))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=leaky_relu_alpha))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.15))
    model.add(Flatten())

    model.add(Dense(180))
    model.add(LeakyReLU(alpha=leaky_relu_alpha))
    model.add(Dropout(0.15))
    model.add(Dense(5))
    model.add(Activation("sigmoid"))

    model.compile(optimizer=keras.optimizers.Adam(lr=1e-3),
                  loss=BinaryCrossentropy(label_smoothing=0.095),
                  metrics=[tf.keras.metrics.AUC(name = 'auc'), "accuracy"])

    return model
```

```
Epoch 1/10
366/366 [==============================] - 395s 1s/step - loss: 0.8017 - auc: 0.5111 - accuracy: 0.1859
Epoch 2/10
366/366 [==============================] - 392s 1s/step - loss: 0.6952 - auc: 0.5290 - accuracy: 0.2024
Epoch 3/10
366/366 [==============================] - 380s 1s/step - loss: 0.6930 - auc: 0.5332 - accuracy: 0.2191
Epoch 4/10
366/366 [==============================] - 392s 1s/step - loss: 0.6919 - auc: 0.5370 - accuracy: 0.2324
Epoch 5/10
366/366 [==============================] - 383s 1s/step - loss: 0.6909 - auc: 0.5426 - accuracy: 0.2259
Epoch 6/10
366/366 [==============================] - 383s 1s/step - loss: 0.6904 - auc: 0.5443 - accuracy: 0.2273
Epoch 7/10
366/366 [==============================] - 384s 1s/step - loss: 0.6896 - auc: 0.5486 - accuracy: 0.2370
Epoch 8/10
366/366 [==============================] - 386s 1s/step - loss: 0.6892 - auc: 0.5493 - accuracy: 0.2506
Epoch 9/10
366/366 [==============================] - 384s 1s/step - loss: 0.6889 - auc: 0.5509 - accuracy: 0.2547
Epoch 10/10
366/366 [==============================] - 374s 1s/step - loss: 0.6884 - auc: 0.5521 - accuracy: 0.2437
<tensorflow.python.keras.callbacks.History at 0x7fa7b3c840d0>
```

Based on the training data, a train_test split ratio of 4:1, and our chosen method for classifying a trade as executable or not, we achieve an accuracy of

24% over ten different lifecycles of running our Neural Network. Unfortunately, there is no way for us to know the true accuracy of this model on a test set because we were past the Kaggle competition's submission deadline.

Conclusion

One thing that was very unique about this task was that it was exploratory in nature. All data columns were anonymized, and despite being a classification task, we were not provided the output labels in the training set. Coming up with a translation from `resp` values to an appropriate binary output was a part of what we had to do and define ourselves; in this regard, this task was particularly challenging. We put significant effort into breaking down and interpreting the millions of rows that were stripped away from any semantic meaning, but we think we did a fair job at it, explaining each step of the exploratory process.

Appendix

Possible Changes

Given the large dataset that we were dealing with and the time constraints we had, it was hard to run all the desired analyses and models on the dataset. Even though our script uses size optimization techniques on the data (converting float64 values to float16 and float32 wherever possible and converting int32 values to int8 and int16 values wherever possible to compress the size), given the fact that the dataset contains more than a million rows and 130 features and 4 resp values for each of these data points, the average time it took to train the model was more than an hour and hence we had to make certain tradeoffs when making decisions regarding analyzing the data for model choice.

If we had more time and computational resources, we would

1. Expand the scope of the toy models used for the calcualtion of Shap values, and compute an average of the SHAP values produced by all the models to determine feature importance and engineer the features accordingly into our final model.

2. Consider different techniques and logics to determine and tag the execution of a trade in our training set. This would have provided us with multiple training sets, and we could have engineered a model for each of those training set and compare their performance on the competition.
3. Create a predictive model for the resp values rather than a classifier on the action tag and test out different deep learning methods to achieve the prediction and measure how that approach influences the final model accuracy.
4. We would also have to liked to use more hyperparameter optimization techniques, such as GridSearchCV or Bayesian Optimization, to tweak our model as compared to what we were able to achieve.

Requirements

```
import sys
print('Python\t\ttested on 3.6.3,3.6.5\t\tcurrent:',sys.version)
import numpy
print('numpy\t\ttested on 1.14.3,1.14.5\t\tcurrent:',numpy.__version__)
import pandas
print('pandas\t\ttested on 0.22.0,0.23.0\t\tcurrent:',pandas.__version__)
import matplotlib
print('matplotlib\ttested on 2.2.1,2.2.2\t\tcurrent:',matplotlib.__version__)
import sklearn
print('sklearn\t\ttested on 0.19.1,0.19.2\t\tcurrent:',sklearn.__version__)
import tensorflow
print('tensorflow\ttested on 1.10.0,1.10.1\t\tcurrent:',tensorflow.__version__)
import logging
print('logging\t\ttested on 0.5.1.2\t\tcurrent:',logging.__version__)
import shap
print('shap\t\ttested on 0.38.1\t\tcurrent:',shap.__version__)
```

```
Python          tested on 3.6.3,3.6.5           current: 3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ]
numpy           tested on 1.14.3,1.14.5         current: 1.19.2
pandas          tested on 0.22.0,0.23.0         current: 1.1.3
matplotlib      tested on 2.2.1,2.2.2           current: 3.3.2
sklearn         tested on 0.19.1,0.19.2         current: 0.23.2
tensorflow      tested on 1.10.0,1.10.1         current: 2.4.1
logging         tested on 0.5.1.2               current: 0.5.1.2
shap            tested on 0.38.1                current: 0.38.1
```

LOs Applied

#QuantitativeToolkit:
Identify appropriate tools and resources to tackle a specific quantitative problem in the context
of quantitative finance, and present quantitative work products professionally.

- We used a variety of computational tools that were useful in tearing apart the data. For instance, the data provided by Jane Street was too big to conveniently handle with pandas, so we adopted a tool called datatable that can accelerate the retrieval process by only querying and processing relevant sections of the data.

#QuantitativeProgramming:
Implement well-defined strategies and/or algorithms to analyze financial data and/or make
predictions of financial signals into the future.

- code notebook 1 (exploring the data)
- code notebook 2 (final model, JaneStreet format)

#QuantitativeInterpretation:
Describe, analyze and provide a critical interpretation of the results of quantitative simulations
involving financial data.

- This entire task required numerous interpretations of data on top of one another, due to how cluttered as well as anonymized the data was. The bulk of the effort was experimenting with data to find consistencies and patterns to infer what they could imply, or at least how they could be useful to build our model.

References

1. Dansbecker. (2020, October). SHAP Values. Kaggle.com; Kaggle. https://www.kaggle.com/dansbecker/shap-values
2. Jane Street Market Prediction | Kaggle. (2020). Kaggle.com. https://www.kaggle.com/c/jane-street-market-prediction/overview
3. Marco Del Pra. (2020, September 8). Time Series Classification with Deep Learning - Towards Data Science. Medium; Towards Data Science. https://towardsdatascience.com/time-series-classification-with-deep-learning-d238f0147d6f
4. Tseng, G. (2018, June 21). Interpreting complex models with SHAP values - Gabriel Tseng - Medium. Medium; Medium. https://medium.com/@gabrieltseng/interpreting-complex-models-with-shap-values-1c187db6ec83