Unit 4 - Modeling

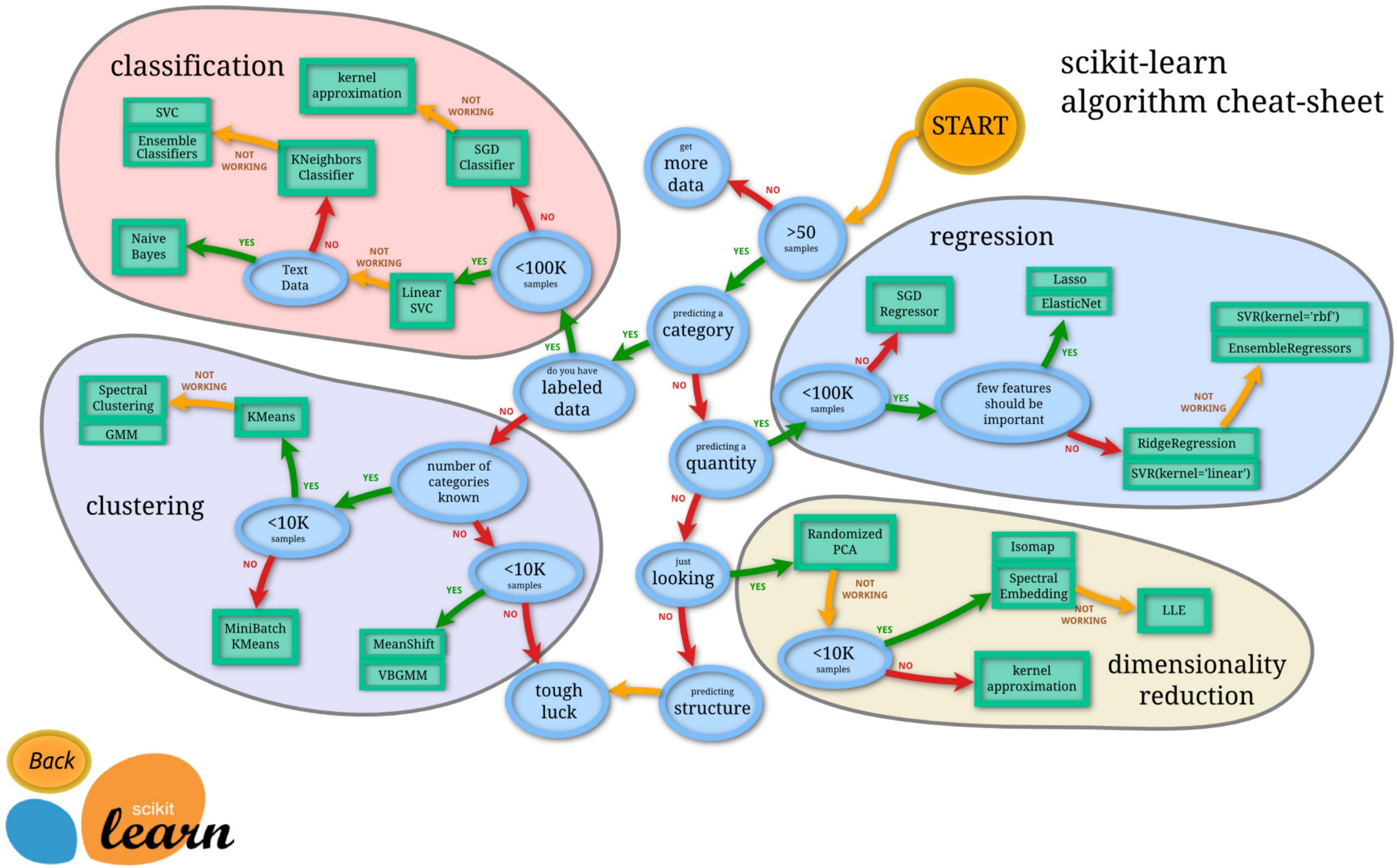
In this notebook we will cover:

1. How to choose a machine learning model?
   A. What to choose from?
   B. What to test for?

Model Selection

- Many different algorithms to chose from
- First 3 factors to consider when chosing an algorithm:
  - Task (Classification, Regression, Clustering, DR)
  - Type of data (Labeled, unlabeled)

- Amount of data




```python
In [1]: import matplotlib
        import numpy as np
        import pandas as pd
        import random
        import sklearn
        import lightgbm as lgb
```

```
import matplotlib.pyplot as plt
from scipy.stats import spearmanr
%matplotlib inline

#!pip install numerapi
from pathlib import Path
import dask.dataframe as dd
from dask.array import from_array
import numerapi
import matplotlib.pyplot as plt

from sklearn import (
    feature_extraction, feature_selection, decomposition, linear_model,
    model_selection, metrics, svm
)
```

In [2]:

```
#Create instance of NumerAPI
napi = numerapi.NumerAPI()

#Use numerAPI to download a single file
train_pq_path = "numerai_training_data_int8.parquet"
val_pq_path = "numerai_validation_data_int8.parquet"


napi.download_dataset("numerai_training_data_int8.parquet", train_pq_path)
napi.download_dataset("numerai_validation_data_int8.parquet", val_pq_path)
```

```
2021-11-17 14:36:20,700 INFO numerapi.utils: target file already exists
2021-11-17 14:36:20,702 INFO numerapi.utils: download complete
2021-11-17 14:36:22,191 INFO numerapi.utils: target file already exists
2021-11-17 14:36:22,192 INFO numerapi.utils: download complete
```

In [3]:

```
#Read parquet files into DataFrames
df_train = dd.read_parquet('numerai_training_data_int8.parquet')
df_val = dd.read_parquet('numerai_validation_data_int8.parquet')
```

In [4]:

```
features = [c for c in df_train if c.startswith("feature")]
features_erano = features + ["erano"]
```

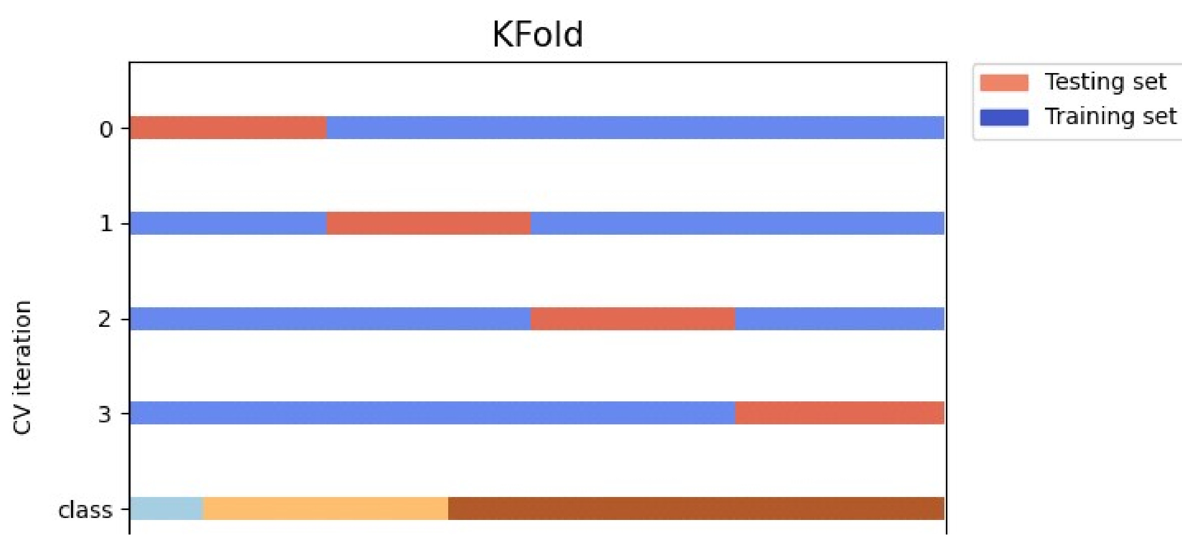
```
targets = [c for c in df_train if c.startswith("target")]

df_train["erano"] = df_train.era.astype(int)
eras = df_train.erano
target = "target"
```

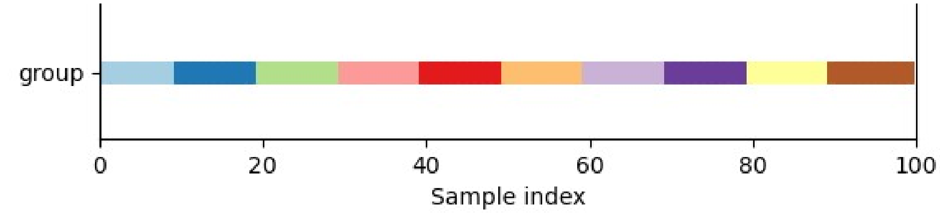
In [5]:
```
df_val["erano"] = df_val.era.astype(int)
```

In [19]:
```
#Create variables with just feature or target data
X_train = df_train.reset_index()[features].to_dask_array(lengths=True)

X_train_erano = df_train.reset_index()[features_erano].to_dask_array(lengths=True)

y_train = df_train.reset_index()["target"].to_dask_array(lengths=True)
```

K-Fold Cross Validation

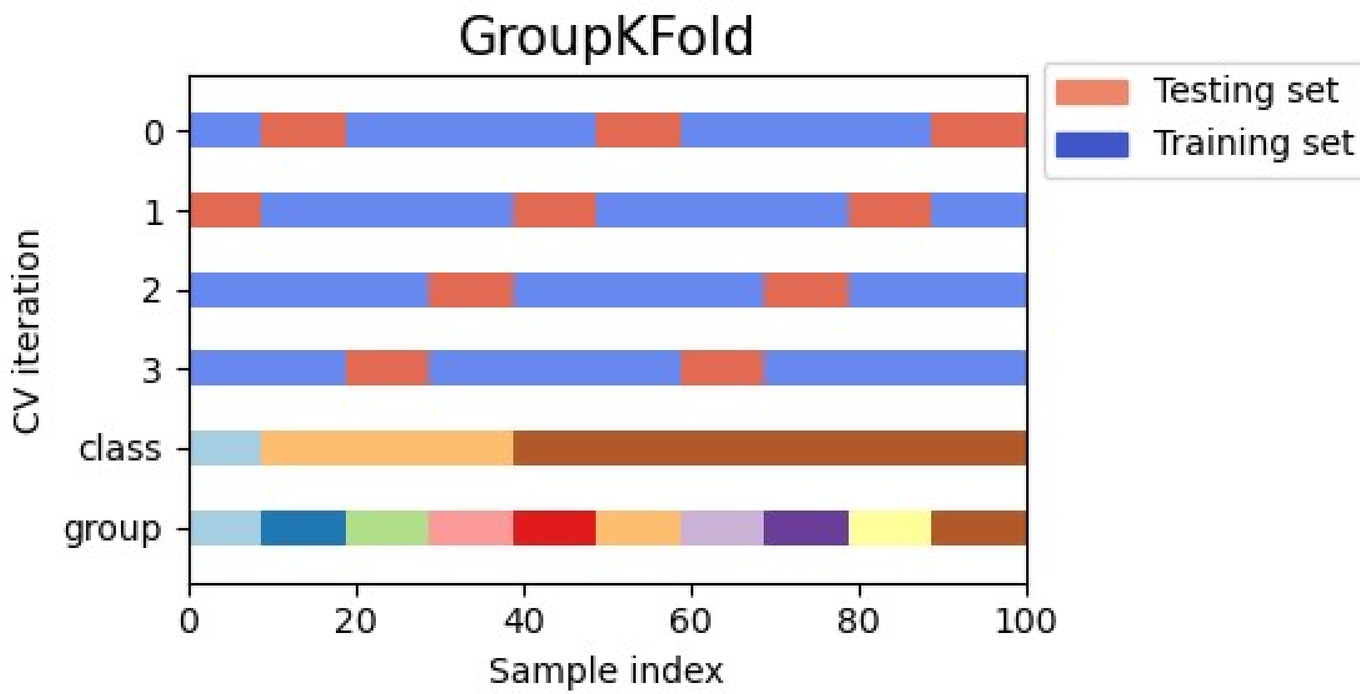
- K-fold cross-validation is a statistical method used to estimate the skill of machine learning models.
- Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).



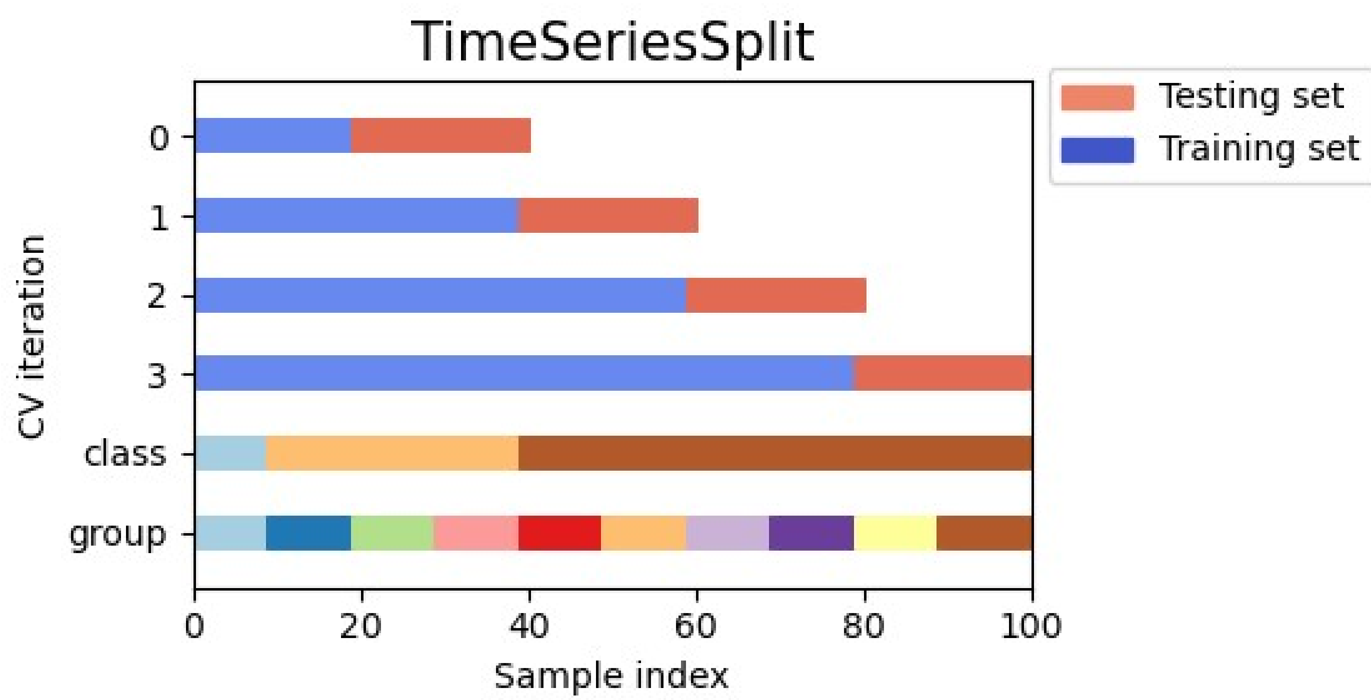


Group K-Fold Cross Validation

- Group K-fold is a K-fold iterator variant with non-overlapping groups.



Era-wise Time-series Cross Validation



In [7]:
```
from sklearn.model_selection._split import _BaseKFold, indexable, _num_samples
from sklearn import model_selection, metrics
import csv

class TimeSeriesSplitGroups(_BaseKFold):
    def __init__(self, n_splits=5):
        super().__init__(n_splits, shuffle=False, random_state=None)

    def split(self, X, y=None, groups=None):
        X, y, groups = indexable(X, y, groups)
        n_samples = _num_samples(X)
        n_splits = self.n_splits
        n_folds = n_splits + 1
        group_list = np.unique(groups)
        n_groups = len(group_list)
        if n_folds > n_groups:
```

```
            raise ValueError(
                ("Cannot have number of folds ={0} greater"
                 " than the number of samples: {1}.").format(n_folds,
                                                              n_groups))
        indices = np.arange(n_samples)
        test_size = (n_groups // n_folds)
        test_starts = range(test_size + n_groups % n_folds,
                            n_groups, test_size)
        #test_starts = list(test_starts)[::-1]
        for test_start in test_starts:

            yield (indices[groups.isin(group_list[:test_start])],
                   indices[groups.isin(group_list[test_start:test_start + test_size])])
```

In [9]:
```
cvGen=TimeSeriesSplitGroups(n_splits=5) # purged cv

for i,(train,test) in enumerate(cvGen.split(X=X_train_erano.compute(), y=y_train, groups=eras)):

    print(f"train: {train[0]}, {train[1]}")
    print(f"test: {test[0]}, {test[1]}")

    X0, y0 = X_train_erano.loc[train[0]:train[-1]], y_train.loc[train[0]:train[-1]]
    X1, y1 = X_train_erano.loc[test[0]:test[-1]], y_train.loc[test[0]:test[-1]]

    print(f"X0:{X0.shape[0].compute(), X0.shape[1]}, y0: {y0.shape}") #y0:{y0.shape[0].compute(), y0.s
    print(f"X1:{X1.shape[0].compute(), X1.shape[1]}, y1: {y1.shape}") #y1:{y1.shape[0].compute(), y1.s
```

```
train: 0, 1
test: 312079, 312080
X0:(312079, 1051), y0: (dd.Scalar<size-ag..., dtype=int32>,)
X1:(384953, 1051), y1: (dd.Scalar<size-ag..., dtype=int32>,)
train: 0, 1
test: 697032, 697033
X0:(697032, 1051), y0: (dd.Scalar<size-ag..., dtype=int32>,)
```

Loss Function 📉

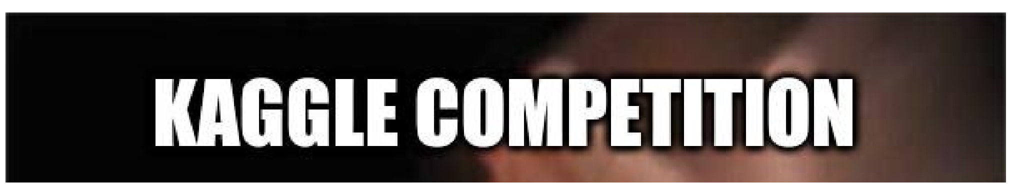
- We will be using a correlation based loss function
- MSE looks worse than correlation out of sample

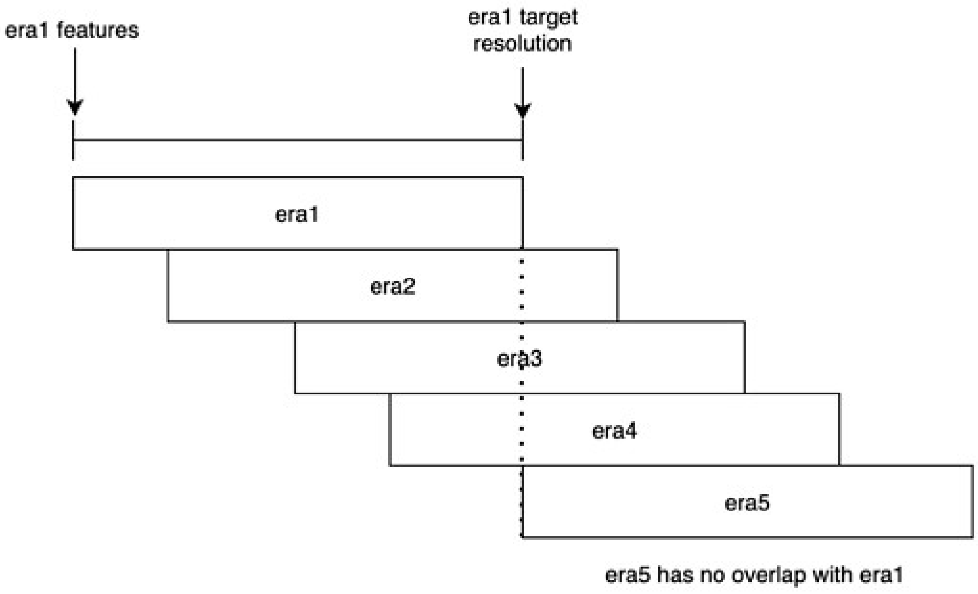
In [11]:

```
# The models should be scored based on the rank-correlation (spearman) with the target
def numerai_score(y_true, y_pred, eras):
    rank_pred = y_pred.groupby(eras).apply(lambda x: x.rank(pct=True, method="first"))
    return np.corrcoef(y_true, rank_pred)[0,1]

# It can also be convenient while working to evaluate based on the regular (pearson) correlation
def correlation_score(y_true, y_pred):
    return numpy.corrcoef(y_true, y_pred)[0,1]

def spearman(y_true, y_pred):
    return spearmanr(y_pred, y_true).correlation
```

The Meta(verse?)

- Gradient Boosting Decision Trees (GBDT) are a great starting point, and overall very well rounded algorithm
- Several popular implementations of GBDT (Light GBM vs XGBoost vs. CatBoost)
- We will be using LGBM for this series as it is very memory efficient





In [12]:
```
# train models on subsamples eras in df_train
lgb1 = lgb.LGBMRegressor()
lgb1.fit(df_train[eras.isin(np.arange(1, 304, 4))][features], df_train[eras.isin(np.arange(1, 304, 4))
```

Out[12]:
```
LGBMRegressor()
```

In [13]:
```
lgb1.predict(df_val[features])
```

Out[13]:
```
array([0.48882739, 0.49161944, 0.50468703, ..., 0.48715594, 0.49587954,
       0.49303782])
```

Putting it all together

- The following code sample will fit and perform Time series split cross validation on a LGBM Regressor, and calculate the average error across the 5 splits
- We will cover how to wrap this in a function and perform hyperparameter tuning in the next video! 👀

In [14]:
```
from sklearn import model_selection, metrics
```

In [21]:
```
cv_score = []
score = np.mean(model_selection.cross_val_score(
                lgb1,
                X_train,
                y_train,
                cv=TimeSeriesSplitGroups(5),
                n_jobs=1,
                groups=eras,
                scoring=metrics.make_scorer(spearman, greater_is_better=True)))

cv_score.append(score)
print(cv_score)
```

```
C:\Users\peter\Anaconda3\envs\numerai\lib\site-packages\sklearn\utils\__init__.py:202: PerformanceW
arning: Slicing is producing a large chunk. To accept the large
chunk and silence this warning, set the option
    >>> with dask.config.set(**{'array.slicing.split_large_chunks': False}):
    ...     array[indexer]

To avoid creating the large chunks, set the option
    >>> with dask.config.set(**{'array.slicing.split_large_chunks': True}):
    ...     array[indexer]
  return array[key] if axis == 0 else array[:, key]
C:\Users\peter\Anaconda3\envs\numerai\lib\site-packages\sklearn\utils\__init__.py:202: PerformanceW
arning: Slicing is producing a large chunk. To accept the large
chunk and silence this warning, set the option
    >>> with dask.config.set(**{'array.slicing.split_large_chunks': False}):
    ...     array[indexer]

To avoid creating the large chunks, set the option
    >>> with dask.config.set(**{'array.slicing.split_large_chunks': True}):
    ...     array[indexer]
  return array[key] if axis == 0 else array[:, key]
C:\Users\peter\Anaconda3\envs\numerai\lib\site-packages\sklearn\utils\__init__.py:202: PerformanceW
arning: Slicing is producing a large chunk. To accept the large
chunk and silence this warning, set the option
    >>> with dask.config.set(**{'array.slicing.split_large_chunks': False}):
    ...     array[indexer]

To avoid creating the large chunks, set the option
    >>> with dask.config.set(**{'array.slicing.split_large_chunks': True}):
    ...     array[indexer]
  return array[key] if axis == 0 else array[:, key]
C:\Users\peter\Anaconda3\envs\numerai\lib\site-packages\sklearn\utils\__init__.py:202: PerformanceW
arning: Slicing is producing a large chunk. To accept the large
chunk and silence this warning, set the option
    >>> with dask.config.set(**{'array.slicing.split_large_chunks': False}):
```

```
[0.04567694301958335]
```

Thank You and Good Luck!

- Like & Subscribe for more!

- Github (https://github.com/peterling7710/NumeraiStarterPack) with the notebooks for this series
- Find my socials here (https://linktr.ee/peterling) for more numer.ai related content

In [ ]:

In [ ]: