In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
d as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session
```

```
/kaggle/input/30-days-of-ml/sample_submission.csv
/kaggle/input/30-days-of-ml/train.csv
/kaggle/input/30-days-of-ml/test.csv
```

Loading All Datasets

In [2]:

```
# Use the column 'id' as the index for train and test datasets
train = pd.read_csv('/kaggle/input/30-days-of-ml/train.csv', index_col = 'id')
test = pd.read_csv('/kaggle/input/30-days-of-ml/test.csv', index_col = 'id')
sample = pd.read_csv('/kaggle/input/30-days-of-ml/sample_submission.csv')
```

In [3]:

```
# View train df
print(train.head())
```

```
   cat0 cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9  ...     cont5     cont6  \
id                                                    ...                       
1     B    B    B    C    B    B    A    E    C    N  ...  0.400361  0.160266   
2     B    B    A    A    B    D    A    F    A    O  ...  0.533087  0.558922   
3     A    A    A    C    B    D    A    D    A    F  ...  0.650609  0.375348   
4     B    B    A    C    B    D    A    E    C    K  ...  0.668980  0.239061   
6     A    A    A    C    B    D    A    E    A    N  ...  0.686964  0.420667   

       cont7     cont8     cont9    cont10    cont11    cont12    cont13  \
id                                                                         
1   0.310921  0.389470  0.267559  0.237281  0.377873  0.322401  0.869850   
2   0.516294  0.594928  0.341439  0.906013  0.921701  0.261975  0.465083   
3   0.902567  0.555205  0.843531  0.748809  0.620126  0.541474  0.763846   
4   0.732948  0.679618  0.574844  0.346010  0.714610  0.540150  0.280682   
6   0.648182  0.684501  0.956692  1.000773  0.776742  0.625849  0.250823   

      target  
id            
1   8.113634  
2   8.481233  
3   8.364351  
4   8.049253  
6   7.972260  

[5 rows x 25 columns]
```

In [4]:

```
# View test df
print(test.head())
```

```
   cat0 cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9  ...     cont4     cont5  \
id                                                    ...
0     B    B    B    C    B    B    A    E    E    I  ...  0.476739  0.376350
5     A    B    A    C    B    C    A    E    C    H  ...  0.285509  0.860046
15    B    A    A    A    B    B    A    E    D    K  ...  0.697272  0.683600
16    B    B    A    C    B    D    A    E    A    N  ...  0.719306  0.777890
17    B    B    A    C    B    C    A    E    C    F  ...  0.313032  0.431007

       cont6     cont7     cont8     cont9    cont10    cont11    cont12  \
id
0   0.337884  0.321832  0.445212  0.290258  0.244476  0.087914  0.301831
5   0.798712  0.835961  0.391657  0.288276  0.549568  0.905097  0.850684
15  0.404089  0.879379  0.275549  0.427871  0.491667  0.384315  0.376689
16  0.730954  0.644315  1.024017  0.391090  0.988340  0.411828  0.393585
17  0.390992  0.408874  0.447887  0.390253  0.648932  0.385935  0.370401

      cont13
id
0   0.845702
5   0.693940
15  0.508099
16  0.461372
17  0.900412

[5 rows x 24 columns]
```

In [5]:

```
# View sample submission
print(sample.head())
```

```
   id  target
0   0     0.5
1   5     0.5
2  15     0.5
3  16     0.5
4  17     0.5
```

In [6]:

```
# Separate target and features
y = train['target']
features = train.drop(['target'], axis = 1)
```

Ordinal Encode Categorical Columns In Given Features

In [7]:

```
from sklearn.preprocessing import OrdinalEncoder

object_cols = [col for col in features.columns if 'cat' in col]

X = features.copy()
X_test = test.copy()

# ordinal-encode the categorical columns in X and X_test datasets.
ordinal_encoder = OrdinalEncoder()
X[object_cols] = ordinal_encoder.fit_transform(features[object_cols])
X_test[object_cols] = ordinal_encoder.transform(test[object_cols])
```

In [8]:

```
# Preview
print(X.head())
```

```
    cat0  cat1  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  ...     cont4  \
id                                                              ...
1    1.0   1.0   1.0   2.0   1.0   1.0   0.0   4.0   2.0  13.0  ...  0.610706
2    1.0   1.0   0.0   0.0   1.0   3.0   0.0   5.0   0.0  14.0  ...  0.276853
3    0.0   0.0   0.0   2.0   1.0   3.0   0.0   3.0   0.0   5.0  ...  0.285074
4    1.0   1.0   0.0   2.0   1.0   3.0   0.0   4.0   2.0  10.0  ...  0.284667
6    0.0   0.0   0.0   2.0   1.0   3.0   0.0   4.0   0.0  13.0  ...  0.287595

       cont5     cont6     cont7     cont8     cont9    cont10    cont11  \
id
1   0.400361  0.160266  0.310921  0.389470  0.267559  0.237281  0.377873
2   0.533087  0.558922  0.516294  0.594928  0.341439  0.906013  0.921701
3   0.650609  0.375348  0.902567  0.555205  0.843531  0.748809  0.620126
4   0.668980  0.239061  0.732948  0.679618  0.574844  0.346010  0.714610
6   0.686964  0.420667  0.648182  0.684501  0.956692  1.000773  0.776742

      cont12    cont13
id
1   0.322401  0.869850
2   0.261975  0.465083
3   0.541474  0.763846
4   0.540150  0.280682
6   0.625849  0.250823

[5 rows x 24 columns]
```

In [9]:

```
# Split the encoded features df into training and validation datasets
from sklearn.model_selection import train_test_split

X_train, X_valid, y_train, y_valid = train_test_split(X, y, random_state = 0)
```

Use A XGBRegressor Model And Train It To Provide Better Performance

In [10]:

```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
```

In [11]:

```
# Define the model
model = XGBRegressor(learning_rate = 0.02, random_state = 42, tree_method = 'gpu_hist')

# Train the model(will take some time to execute)
model.fit(X_train, y_train, early_stopping_rounds = 5, eval_set = [(X_valid, y_valid)],
verbose = False)
preds_valid = model.predict(X_valid)
print(mean_squared_error(y_valid, preds_valid, squared = False))
```

```
1.262424849715232
```

In [12]:

```
# Get the first 5 prediction results
print(list(preds_valid)[0:6:1])
```

```
[7.189719, 7.0716166, 7.257436, 7.1754956, 7.313472, 7.2468147]
```

Submission

In [13]:

```
# Use the model to generate predictions on the given testing dataset
test_predictions = model.predict(X_test)
```

```
# Save the predictions to a CSV file
output = pd.DataFrame({'Id': X_test.index, 'target': test_predictions})
output.to_csv('submission.csv', index = False)
```

In [14]:

```
# Preview predicted output
print(output.head(10))
```

```
   Id    target
0   0  7.033919
1   5  7.255158
2  15  7.293473
3  16  7.254006
4  17  7.162273
5  19  7.237772
6  20  7.246070
7  21  7.082504
8  23  7.296534
9  29  7.266431
```