Starter Notebook: AIMS Data Science Hackathon by Microsoft

Welcome! This starter notebook is designed to get you started on the AIMS Data Science Hackathon, where you will be attempting to predict a measure of wealth for different locations across Africa. We will take a look at the data, create a model and then use that to make our first submission. After that we will briefly look at some ways to improve. Let's get started.

Loading the Data

We're using the pandas library to load the data into dataframes - a tabular data structure that is perfect for this kind of work. Each of the three CSV files from Zindi is loaded into a dataframe and we take a look at the shape of the data (number of rows and columns) as well as a preview of the first 5 rows to get a feel for what we're working with.

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
train = pd.read_csv('/content/drive/MyDrive/zindi_hackathon/Train.csv')
test = pd.read_csv('/content/drive/MyDrive/zindi_hackathon/Test.csv')
print(train.shape)
train.head()
```

```
(21454, 19)
```

	ID	country	year	urban_or_rural	ghsl_water_surface	ghsl_built_pr
0	ID_AAIethGy	Ethiopia	2016	R	0.0	0.(
1	ID_AAYiaCeL	Ethiopia	2005	R	0.0	0.(
2	ID_AAdurmKj	Mozambique	2009	R	0.0	0.(
3	ID_AAgNHles	Malawi	2015	R	0.0	0.(
4	ID_AAishfND	Guinea	2012	U	0.0	0.(

In train, we have a set of inputs (like 'urban_or_rural' or 'ghsl_water_surface') and our desired output variable, 'Target'. There are 21454 rows - lots of juicy data!

```
test = pd.read_csv('/content/drive/MyDrive/zindi_hackathon/Test.csv')
print(test.shape)
test.head()
```

```
(7194, 18)
```

	ID	country	year	urban_or_rural	ghsl_water_surface	ghsl_built_pı
0	ID_AAcismbB	Democratic Republic of Congo	2007	R	0.000000	0
1	ID_AAeBMsji	Democratic Republic of Congo	2007	U	0.000000	0
2	ID_AAjFMjzy	Uganda	2011	U	0.007359	0
3	ID_AAmMOEEC	Burkina Faso	2010	U	0.000000	0
4	ID_ABguzDxp	Zambia	2007	R	0.000000	0

```
train.columns
```

```
Index(['ID', 'country', 'year', 'urban_or_rural', 'ghsl_water_surface',
       'ghsl_built_pre_1975', 'ghsl_built_1975_to_1990',
       'ghsl_built_1990_to_2000', 'ghsl_built_2000_to_2014',
       'ghsl_not_built_up', 'ghsl_pop_density', 'landcover_crops_fraction',
       'landcover_urban_fraction', 'landcover_water_permanent_10km_fraction',
       'landcover_water_seasonal_10km_fraction', 'nighttime_lights',
       'dist_to_capital', 'dist_to_shoreline', 'Target'],
      dtype='object')
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21454 entries, 0 to 21453
Data columns (total 19 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   ID                                      21454 non-null  object 
 1   country                                 21454 non-null  object 
 2   year                                    21454 non-null  int64  
 3   urban_or_rural                          21454 non-null  object 
 4   ghsl_water_surface                      21454 non-null  float64
 5   ghsl_built_pre_1975                     21454 non-null  float64
 6   ghsl_built_1975_to_1990                 21454 non-null  float64
 7   ghsl_built_1990_to_2000                 21454 non-null  float64
 8   ghsl_built_2000_to_2014                 21454 non-null  float64
 9   ghsl_not_built_up                       21454 non-null  float64
 10  ghsl_pop_density                        21454 non-null  float64
 11  landcover_crops_fraction                21454 non-null  float64
```

```
 12  landcover_urban_fraction                  21454 non-null  float64
 13  landcover_water_permanent_10km_fraction   21454 non-null  float64
 14  landcover_water_seasonal_10km_fraction    21454 non-null  float64
 15  nighttime_lights                          21454 non-null  float64
 16  dist_to_capital                           21454 non-null  float64
 17  dist_to_shoreline                         21454 non-null  float64
 18  Target                                    21454 non-null  float64
dtypes: float64(15), int64(1), object(3)
memory usage: 3.1+ MB
```

```
test
```

	ID	country	year	urban_or_rural	ghsl_water_surface	ghsl_buil
0	ID_AAcismbB	Democratic Republic of Congo	2007	R	0.000000	
1	ID_AAeBMsji	Democratic Republic of Congo	2007	U	0.000000	
2	ID_AAjFMjzy	Uganda	2011	U	0.007359	
3	ID_AAmMOEEC	Burkina Faso	2010	U	0.000000	
4	ID_ABguzDxp	Zambia	2007	R	0.000000	
...	...	...	...	...	...	
7189	ID_zxzKJCMI	Zimbabwe	2010	R	0.000000	
7190	ID_zyBrpgRp	Uganda	2011	U	0.000000	
7191	ID_zyMafcYq	Burkina Faso	2010	U	0.002683	
7192	ID_zyfMsHMG	Zimbabwe	2011	R	0.000332	
7193	ID_zytUOqJv	Democratic Republic of Congo	2007	U	0.076950	

7194 rows × 18 columns

Test looks just like train but without the 'Target' column and with fewer rows.

```
train['country'].describe()
```

```
count       21454
unique         18
top       Nigeria
freq         2695
Name: country, dtype: object
```

```
ss = pd.read_csv('/content/drive/MyDrive/zindi_hackathon/SampleSubmission.csv')
print(ss.shape)
ss.head()
```

```
(7194, 2)
```

	ID	Target
0	ID_AAcismbB	0
1	ID_AAeBMsji	0
2	ID_AAjFMjzy	0
3	ID_AAmMOEEC	0
4	ID_ABguzDxp	0

The sample submission is just the ID column from test with a 'Target' column where we will put out predictions.

Now that we have the data loaded, we can start exploring.

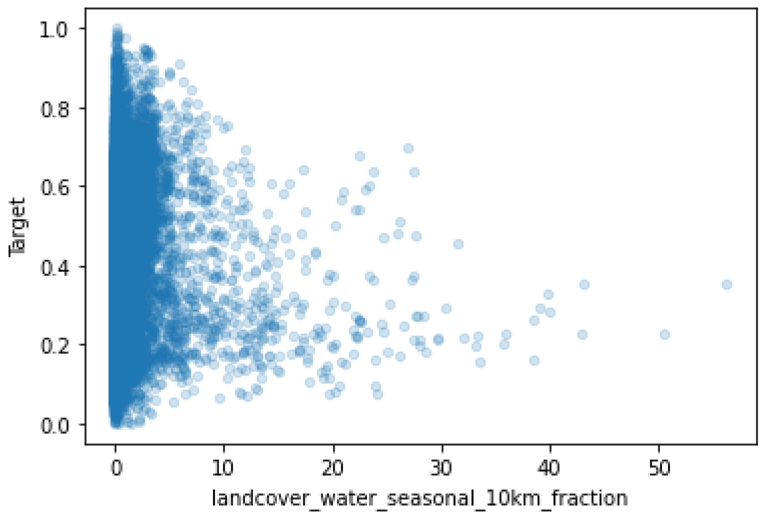
EDA

We will explore some trends in the data and look for any anomalies such as missing data. A few examples are done here but you can explore much further yourself and get to know the data better.

First up: let's see how an input like 'nighttime lights' relates to the target column:

```
# Plotting the relationship between an input column and the target
train.plot(x='nighttime_lights', y='Target', kind='scatter', alpha=0.2)
plt.show()
```


```
# Exercise: Try this with different inputs. Any unexpected trends?
train.plot(x='landcover_water_seasonal_10km_fraction', y='Target', kind='scatter', alpha=0
plt.show()
```

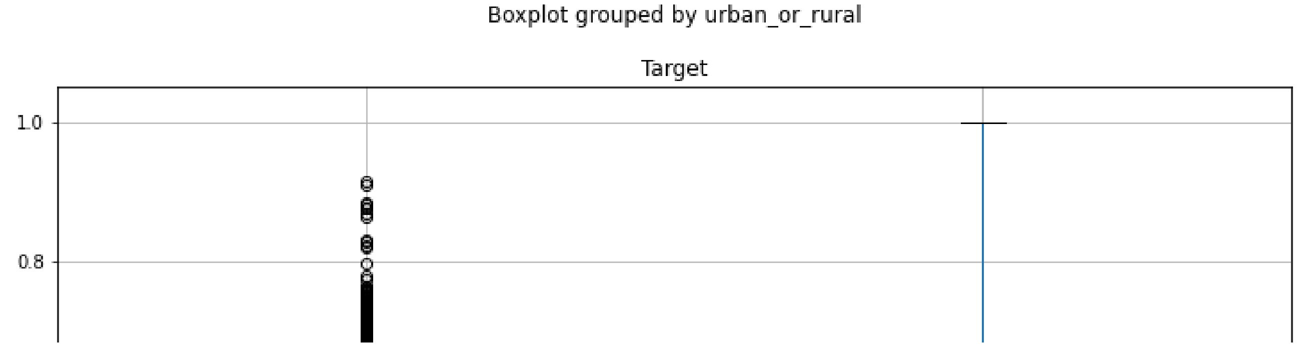


As you might have guessed, places that emit more light tend to be wealthier, but there is a lot of variation.

We can also look at categorical columns like 'country' or 'urban_vs_rural' and see the distribution of the target for each group:

```
# Looking at the wealth distribution for urban vs rural
train.boxplot(by='urban_or_rural', column='Target', figsize=(12, 8))
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationW
  return array(a, dtype, copy=False, order=order)
```




```
# Exercise: which is the country with the higest average wealth_index according to this da
```

Again, not unexpected. Rural areas tend to be less wealthy than urban areas.

Now the scary question: do we have missing data to deal with?

```
train.isna().sum() # Hooray - no missing data!
```

```
ID                                      0
country                                 0
year                                    0
urban_or_rural                          0
ghsl_water_surface                      0
ghsl_built_pre_1975                     0
ghsl_built_1975_to_1990                 0
ghsl_built_1990_to_2000                 0
ghsl_built_2000_to_2014                 0
ghsl_not_built_up                       0
ghsl_pop_density                        0
landcover_crops_fraction                0
landcover_urban_fraction                0
landcover_water_permanent_10km_fraction 0
landcover_water_seasonal_10km_fraction  0
nighttime_lights                        0
dist_to_capital                         0
dist_to_shoreline                       0
Target                                  0
dtype: int64
```

See what other trends you can uncover - we have only scratched the surface here.

```
# Exercise: explore the data further
```

Modelling

We've had a look at our data and it looks good! Let's see if we can create a model to predict the Target given some of our inputs. To start with we will use only the numeric columns, so that we can fit a model right away.

```
in_cols = list(train.columns[4:-1])
print('Input columns:', in_cols)
```

```
_water_surface', 'ghsl_built_pre_1975', 'ghsl_built_1975_to_1990', 'ghsl_built_1990_tc
```

To evaluate our model, we need to keep some data separate. We will split out data into X (inputs) and y (output) and then further split into train and test sets with the following code:

```
from sklearn.model_selection import train_test_split

X, y = train[in_cols], train['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=58)
print(X_train.shape, X_test.shape)
```

```
(19308, 14) (2146, 14)
```

```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install catboost
```

```
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/1e/21/d1718eb4c93d6bacdd540b379
     |████████████████████████████████| 67.3MB 46kB/s
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from ca
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (fror
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Installing collected packages: catboost
Successfully installed catboost-0.25
```

```
from catboost import CatBoostRegressor

model = CatBoostRegressor() # Create the model
model.fit(X_train, y_train) # Train it (this syntax looks the same for all sklearn models)
model.score(X_test, y_test) # Show a score
```

```
768:	learn: 0.0916093	total: 5.6s	remaining: 1.68s
769:	learn: 0.0915831	total: 5.61s	remaining: 1.68s
770:	learn: 0.0915695	total: 5.62s	remaining: 1.67s
771:	learn: 0.0915594	total: 5.62s	remaining: 1.66s
772:	learn: 0.0915443	total: 5.63s	remaining: 1.65s
773:	learn: 0.0915263	total: 5.64s	remaining: 1.65s
774:	learn: 0.0915068	total: 5.64s	remaining: 1.64s
775:	learn: 0.0914877	total: 5.65s	remaining: 1.63s
```

```
776:	learn: 0.0914770	total: 5.66s	remaining: 1.62s
777:	learn: 0.0914665	total: 5.66s	remaining: 1.62s
778:	learn: 0.0914556	total: 5.67s	remaining: 1.61s
779:	learn: 0.0914404	total: 5.68s	remaining: 1.6s
780:	learn: 0.0914217	total: 5.68s	remaining: 1.59s
781:	learn: 0.0914143	total: 5.69s	remaining: 1.59s
782:	learn: 0.0914055	total: 5.7s	remaining: 1.58s
783:	learn: 0.0913824	total: 5.71s	remaining: 1.57s
784:	learn: 0.0913712	total: 5.72s	remaining: 1.56s
785:	learn: 0.0913520	total: 5.72s	remaining: 1.56s
786:	learn: 0.0913286	total: 5.73s	remaining: 1.55s
787:	learn: 0.0913182	total: 5.74s	remaining: 1.54s
788:	learn: 0.0913017	total: 5.75s	remaining: 1.54s
789:	learn: 0.0912952	total: 5.76s	remaining: 1.53s
790:	learn: 0.0912629	total: 5.76s	remaining: 1.52s
791:	learn: 0.0912428	total: 5.77s	remaining: 1.51s
792:	learn: 0.0912207	total: 5.79s	remaining: 1.51s
793:	learn: 0.0912093	total: 5.79s	remaining: 1.5s
794:	learn: 0.0911898	total: 5.8s	remaining: 1.5s
795:	learn: 0.0911822	total: 5.81s	remaining: 1.49s
796:	learn: 0.0911748	total: 5.81s	remaining: 1.48s
797:	learn: 0.0911598	total: 5.82s	remaining: 1.47s
798:	learn: 0.0911484	total: 5.83s	remaining: 1.47s
799:	learn: 0.0911283	total: 5.83s	remaining: 1.46s
800:	learn: 0.0911070	total: 5.84s	remaining: 1.45s
801:	learn: 0.0910861	total: 5.85s	remaining: 1.44s
802:	learn: 0.0910705	total: 5.86s	remaining: 1.44s
803:	learn: 0.0910592	total: 5.86s	remaining: 1.43s
804:	learn: 0.0910467	total: 5.87s	remaining: 1.42s
805:	learn: 0.0910336	total: 5.88s	remaining: 1.41s
806:	learn: 0.0910130	total: 5.88s	remaining: 1.41s
807:	learn: 0.0909935	total: 5.89s	remaining: 1.4s
808:	learn: 0.0909668	total: 5.9s	remaining: 1.39s
809:	learn: 0.0909587	total: 5.91s	remaining: 1.39s
810:	learn: 0.0909437	total: 5.92s	remaining: 1.38s
811:	learn: 0.0909372	total: 5.92s	remaining: 1.37s
812:	learn: 0.0909176	total: 5.93s	remaining: 1.36s
813:	learn: 0.0909137	total: 5.94s	remaining: 1.36s
814:	learn: 0.0908944	total: 5.95s	remaining: 1.35s
815:	learn: 0.0908855	total: 5.95s	remaining: 1.34s
816:	learn: 0.0908659	total: 5.96s	remaining: 1.33s
817:	learn: 0.0908480	total: 5.97s	remaining: 1.33s
818:	learn: 0.0908423	total: 5.97s	remaining: 1.32s
819:	learn: 0.0908264	total: 5.98s	remaining: 1.31s
820:	learn: 0.0908159	total: 5.99s	remaining: 1.3s
821:	learn: 0.0908112	total: 6s	remaining: 1.3s
822:	learn: 0.0908042	total: 6s	remaining: 1.29s
823:	learn: 0.0907853	total: 6.01s	remaining: 1.28s
824:	learn: 0.0907747	total: 6.01s	remaining: 1.28s
825:	learn: 0.0907635	total: 6.02s	remaining: 1.27s
826:	learn: 0.0907584	total: 6.03s	remaining: 1.26s
```

We now have a nice test set of ~4200 rows. We will train our model and then use this test set to calculate our score.

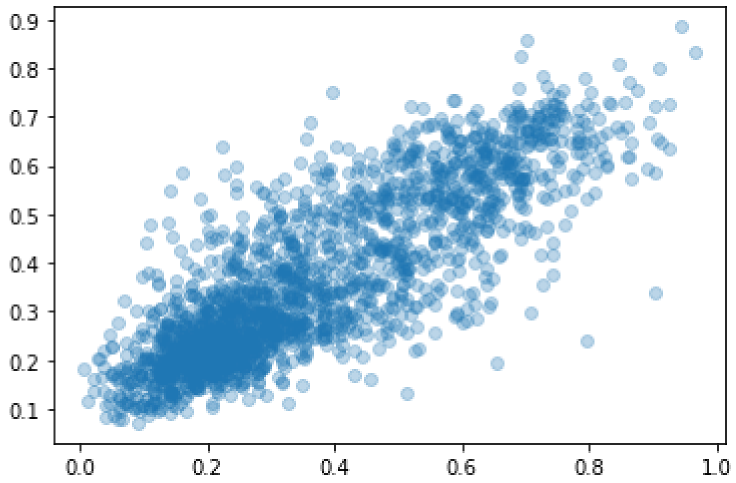
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor() # Create the model
model.fit(X_train, y_train) # Train it (this syntax looks the same for all sklearn models)
```

```
model.fit(X_train, y_train) # Train it (this syntax looks the same for all sklearn models)
model.score(X_test, y_test) # Show a score
```

    0.7037798932775707

What is the score above? The default for regression models is the R^2 score, a measure of how well the mode does at predicting the target. 0.69 is pretty good - let's plot the predictions vs the actual values and see how close it looks to a straight line:

```
from matplotlib import pyplot as plt
plt.scatter(y_test, model.predict(X_test), alpha=0.3)
plt.show()
```



This looks great - most predictions are nice and close to the true value! But we still don't have a way to link this to the leaderboard score on Zindi. Let's remedy that by calculating the Root Mean Squared Error, the same metric Zindi uses.

```
from sklearn.metrics import mean_squared_error

# The `squared=False` bit tells this function to return the ROOT mean squared error
mean_squared_error(y_test, model.predict(X_test), squared=False)
```

    0.10815594201379145

Great stuff. Let's make a submission and then move on to looking for ways to improve.

```
# Copying our predictions into the submission dataframe - make sure the rows are in the sa
ss['Target'] = model.predict(test[in_cols])
ss.head()
```

	ID	Target
0	ID_AAcismbB	0.221479
1	ID_AAeBMsji	0.158832
2	ID_AAmMOEFC	0.314846

We now have our predictions in the right format to submit. The following line saves this to a file that you can then upload to get a score:

```
ss.to_csv('second3_submission.csv', index=False)
```

Getting Better

You might have noticed that your score on Zindi wasn't as good as the one you got above. This is because the test set comes from different countries to the train set. When we did a random split, we ended up with our local train and test both coming from the same countries - and it's easier for a model to extrapolate within countries than it is for it to make predictions for a new location.

So our first step might be to make a scoring function that splits the data according to country, and measures the model performance on unseen countries. Try it and share your testing methods in the discussions. And look at the following questions:

- Does your score drop when you score your model on countries it wasn't trained with?
- Does the new score more accurately match the leaderboard score?
- Are any countries particularly 'hard' to make predictions in?

```
# You code for a new model evaluation method here

train_per_countries
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-67b62dacb722> in <module>()
      1 # You code for a enw model evaluation method here
      2
----> 3 train_per_countries

NameError: name 'train_per_countries' is not defined
```

SEARCH STACK OVERFLOW

Knowing how well our model is doing is useful, but however you measure that we also need ways to improve this performance! There are a few ways to do this:

- Feed the model better data. How? Feature engineering! If we can add meaningful features the model will have more data to work with.
- Tune your models. We used the default parameters - perhaps we can tweak some hyperparameters to make our models better
- Try fancier models. Perhaps XGBoost or a neural network is better than Random Forest at this task

Let's do a little of each. First up, let's create a numeric feature that encodes the 'urban_or_rural' column as something the model can use:

```
# Turning a categorical column into a numeric feature
train['is_urban'] = (train['urban_or_rural'] == 'U').astype(int)
test['is_urban'] = (test['urban_or_rural'] == 'U').astype(int)
train.head()
```

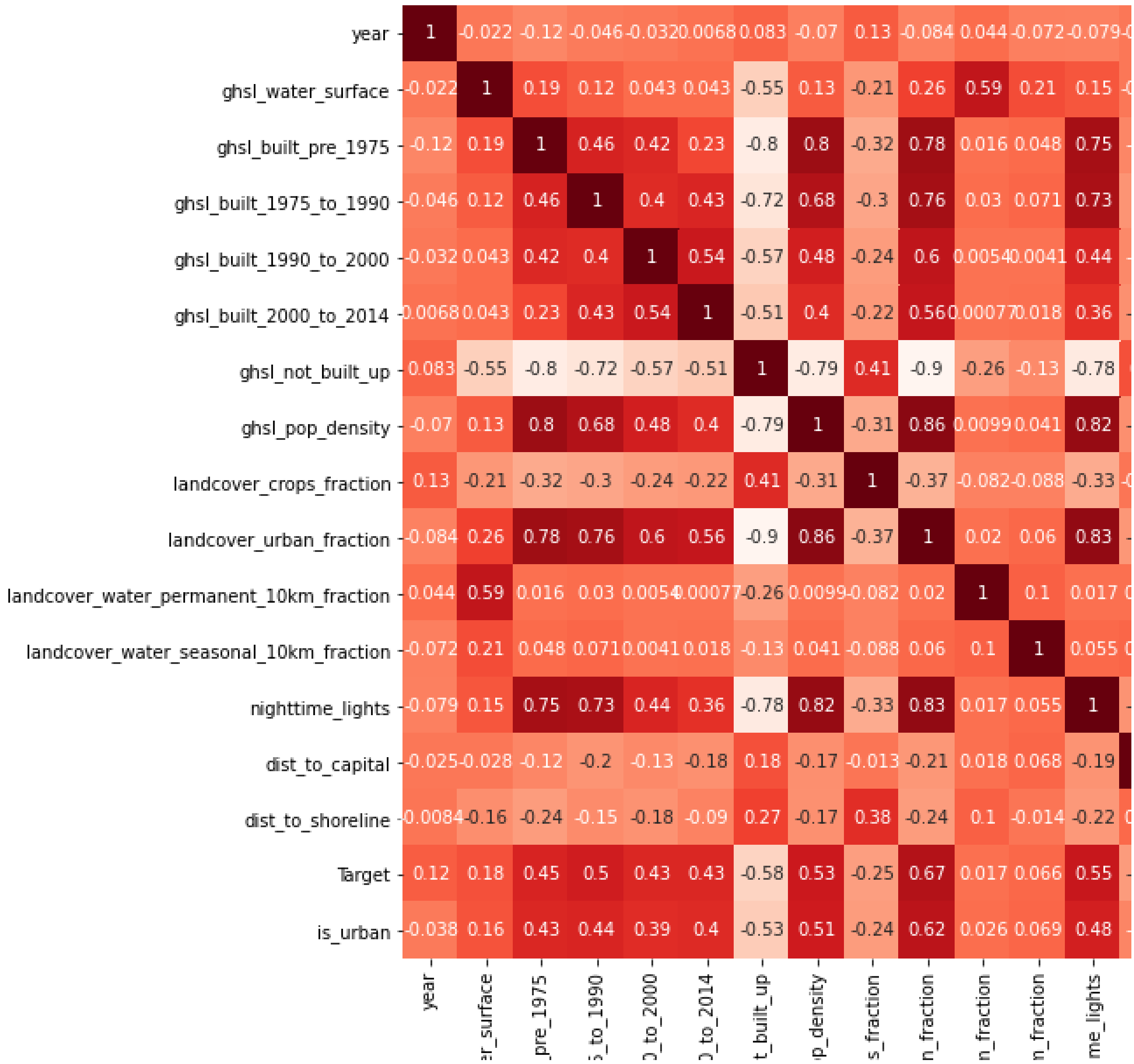
	ID	country	year	urban_or_rural	ghsl_water_surface	ghsl_built_pr
0	ID_AAIethGy	Ethiopia	2016	R	0.0	0.(
1	ID_AAYiaCeL	Ethiopia	2005	R	0.0	0.(
2	ID_AAdurmKj	Mozambique	2009	R	0.0	0.(
3	ID_AAgNHles	Malawi	2015	R	0.0	0.(
4	ID_AAishfND	Guinea	2012	U	0.0	0.(

Note that whenever we add features to train, *we also need to add them to test* otherwise we won't be able to make our predictions.

With this extra feature, we can fit a new model:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#Using Pearson Correlation
plt.figure(figsize=(12,10))
cor = train.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```




```
# selecting best features

from sklearn.feature_selection import SelectKBest , mutual_info_classif

df_input.columns
```

```
Index(['ID', 'country', 'year', 'urban_or_rural', 'ghsl_water_surface',
       'ghsl_built_pre_1975', 'ghsl_built_1975_to_1990',
       'ghsl_built_1990_to_2000', 'ghsl_built_2000_to_2014',
       'ghsl_not_built_up', 'ghsl_pop_density', 'landcover_crops_fraction',
       'landcover_urban_fraction', 'landcover_water_permanent_10km_fraction',
       'landcover_water_seasonal_10km_fraction', 'nighttime_lights',
       'dist_to_capital', 'dist_to_shoreline'],
      dtype='object')
```

```
df_input = train.drop('ID', axis=1, inplace=True)
df_input = train.drop('Target',axis=1)
y_output = train['Target']

df_input
```

	country	year	urban_or_rural	ghsl_water_surface	ghsl_built_pre_1975	gl
0	Ethiopia	2016	R	0.0	0.000000	
1	Ethiopia	2005	R	0.0	0.000000	
2	Mozambique	2009	R	0.0	0.000000	
3	Malawi	2015	R	0.0	0.000141	
4	Guinea	2012	U	0.0	0.011649	
...	...	...	...	...	...	
21449	Nigeria	2013	R	0.0	0.002961	
21450	Senegal	2011	R	0.0	0.000000	
21451	Ghana	2014	R	0.0	0.000536	
21452	Ghana	2014	R	0.0	0.000000	
21453	Mozambique	2011	R	0.0	0.000035	

21454 rows × 17 columns

```
in_cols = list(train.columns[4:-1])
print('Input columns:', in_cols)
```

```
990', 'ghsl_built_1990_to_2000', 'ghsl_built_2000_to_2014', 'ghsl_not_built_up', 'ghs
```

```
selector = SelectKBest(mutual_info_classif, k=5)
selector.fit_transform(df_input, y_output)
df_input.columns[selector.get_support(indices=True)]

#To get the list
vector_names = list(df_input.columns[selector.get_support(indices=True)])
print('The 5 features are ' , vector_names)
```

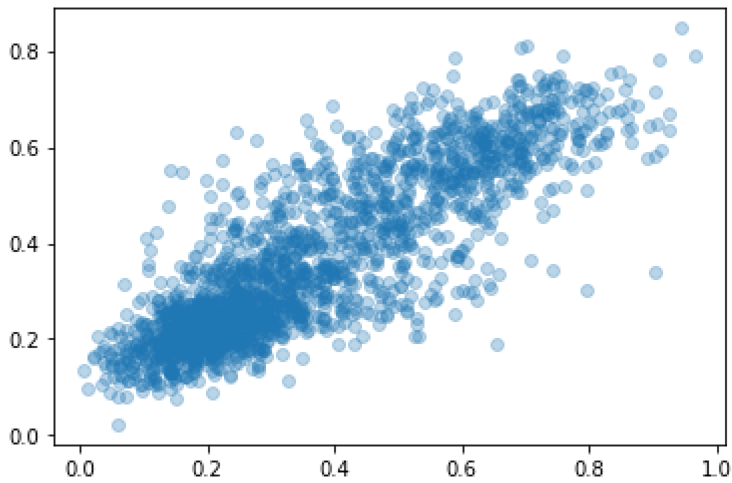
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-58-9541a820792c> in <module>()
      1 selector = SelectKBest(mutual_info_classif, k=5)
----> 2 selector.fit_transform(df_input, y_output)
      3 df_input.columns[selector.get_support(indices=True)]
```

```
in_cols.append('is_urban') # Adding the new features to our list of input columns

# Replace this with your chosen method for evaluating a model:
X, y = train[in_cols], train['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=58)
model = CatBoostRegressor()
result= model.fit(X_train, y_train)
mean_squared_error(y_test, model.predict(X_test), squared=False)
```

```
Learning rate set to 0.066109
0:	learn: 0.1859144	total: 7.86ms	remaining: 7.85s
1:	learn: 0.1789597	total: 15ms	remaining: 7.48s
2:	learn: 0.1722036	total: 22.1ms	remaining: 7.33s
3:	learn: 0.1662579	total: 34ms	remaining: 8.45s
4:	learn: 0.1606806	total: 41.7ms	remaining: 8.3s
5:	learn: 0.1555844	total: 49.1ms	remaining: 8.13s
6:	learn: 0.1510981	total: 56.6ms	remaining: 8.03s
7:	learn: 0.1469668	total: 63.9ms	remaining: 7.92s
8:	learn: 0.1430896	total: 72.6ms	remaining: 7.99s
9:	learn: 0.1397215	total: 79.8ms	remaining: 7.9s
10:	learn: 0.1365198	total: 87.1ms	remaining: 7.83s
11:	learn: 0.1335800	total: 94.3ms	remaining: 7.76s
12:	learn: 0.1311848	total: 113ms	remaining: 8.55s
13:	learn: 0.1289651	total: 124ms	remaining: 8.77s
14:	learn: 0.1268336	total: 137ms	remaining: 8.96s
15:	learn: 0.1249846	total: 144ms	remaining: 8.87s
16:	learn: 0.1233813	total: 152ms	remaining: 8.79s
17:	learn: 0.1219870	total: 159ms	remaining: 8.69s
18:	learn: 0.1206980	total: 167ms	remaining: 8.61s
19:	learn: 0.1194964	total: 174ms	remaining: 8.54s
20:	learn: 0.1183936	total: 184ms	remaining: 8.57s
21:	learn: 0.1173899	total: 196ms	remaining: 8.7s
22:	learn: 0.1164564	total: 204ms	remaining: 8.66s
23:	learn: 0.1156135	total: 211ms	remaining: 8.57s
24:	learn: 0.1148515	total: 218ms	remaining: 8.52s
25:	learn: 0.1141655	total: 226ms	remaining: 8.46s
26:	learn: 0.1135400	total: 233ms	remaining: 8.4s
27:	learn: 0.1130333	total: 240ms	remaining: 8.34s
28:	learn: 0.1124771	total: 248ms	remaining: 8.3s
29:	learn: 0.1120074	total: 255ms	remaining: 8.25s
30:	learn: 0.1115439	total: 262ms	remaining: 8.19s
31:	learn: 0.1111979	total: 269ms	remaining: 8.14s
32:	learn: 0.1108222	total: 276ms	remaining: 8.09s
33:	learn: 0.1104858	total: 283ms	remaining: 8.05s
34:	learn: 0.1100799	total: 290ms	remaining: 8.01s
35:	learn: 0.1097890	total: 298ms	remaining: 7.98s
36:	learn: 0.1095544	total: 305ms	remaining: 7.94s
37:	learn: 0.1092934	total: 313ms	remaining: 7.91s
38:	learn: 0.1090950	total: 320ms	remaining: 7.88s
39:	learn: 0.1088830	total: 327ms	remaining: 7.85s
40:	learn: 0.1086394	total: 335ms	remaining: 7.83s
41:	learn: 0.1083860	total: 343ms	remaining: 7.82s
42:	learn: 0.1081333	total: 350ms	remaining: 7.79s
43:	learn: 0.1079572	total: 358ms	remaining: 7.77s
```

```
44:     learn: 0.1077810        total: 365ms    remaining: 7.74s
45:     learn: 0.1076184        total: 372ms    remaining: 7.71s
46:     learn: 0.1074792        total: 379ms    remaining: 7.69s
47:     learn: 0.1073566        total: 386ms    remaining: 7.66s
48:     learn: 0.1071477        total: 399ms    remaining: 7.74s
49:     learn: 0.1070510        total: 406ms    remaining: 7.71s
50:     learn: 0.1068645        total: 413ms    remaining: 7.69s
51:     learn: 0.1067240        total: 421ms    remaining: 7.67s
52:     learn: 0.1066050        total: 428ms    remaining: 7.65s
53:     learn: 0.1065029        total: 435ms    remaining: 7.63s
54:     learn: 0.1063895        total: 443ms    remaining: 7.61s
55:     learn: 0.1063116        total: 450ms    remaining: 7.58s
56:     learn: 0.1062127        total: 457ms    remaining: 7.57s
57:     learn: 0.1061188        total: 464ms    remaining: 7.54s
```

```
plt.scatter(y_test, model.predict(X_test), alpha=0.3)
plt.show()
```




```
model.score(X_test, y_test) # Show a score
```

```
0.7313310611361041
```

Did your score improve?

Next, let's tune our model by adjusting the maximum depth. This is one of many hyperparameters that can be tweaked on a Random Forest model. Here I just try a few randomly chosen values, but you could also use a grid search to try values more methodically.

```
for max_depth in [3, 5, 8, 10, 14, 18]:
    model = RandomForestRegressor()
    # Again, you van use a better method to evaluate the model here...
    model.fit(X_train, y_train)
    print(max_depth, mean_squared_error(y_test, model.predict(X_test), squared=False))
```

```
3 0.10426857237585264
5 0.1042253627004492
8 0.10464683740800874
10 0.10442331242318882
14 0.1044716501340744
18 0.10435400766492799
```

```
for max_depth in [3, 5, 8, 10, 14, 18]:
    model = CatBoostRegressor()
    # Again, you van use a better method to evaluate the model here...
    model.fit(X_train, y_train)
    print(max_depth, mean_squared_error(y_test, model.predict(X_test), squared=False))
```

```
34:     learn: 0.1100799        total: 272ms    remaining: 7.49s
35:     learn: 0.1097890        total: 279ms    remaining: 7.47s
36:     learn: 0.1095544        total: 286ms    remaining: 7.45s
37:     learn: 0.1092934        total: 294ms    remaining: 7.43s
38:     learn: 0.1090950        total: 301ms    remaining: 7.41s
39:     learn: 0.1088830        total: 308ms    remaining: 7.39s
40:     learn: 0.1086394        total: 316ms    remaining: 7.39s
41:     learn: 0.1083860        total: 324ms    remaining: 7.4s
42:     learn: 0.1081333        total: 332ms    remaining: 7.39s
43:     learn: 0.1079572        total: 340ms    remaining: 7.38s
44:     learn: 0.1077810        total: 347ms    remaining: 7.37s
45:     learn: 0.1076184        total: 355ms    remaining: 7.36s
46:     learn: 0.1074792        total: 362ms    remaining: 7.34s
47:     learn: 0.1073566        total: 370ms    remaining: 7.33s
48:     learn: 0.1071477        total: 377ms    remaining: 7.31s
49:     learn: 0.1070510        total: 384ms    remaining: 7.3s
50:     learn: 0.1068645        total: 392ms    remaining: 7.3s
51:     learn: 0.1067240        total: 400ms    remaining: 7.29s
52:     learn: 0.1066050        total: 408ms    remaining: 7.29s
53:     learn: 0.1065029        total: 415ms    remaining: 7.27s
54:     learn: 0.1063895        total: 423ms    remaining: 7.26s
55:     learn: 0.1063116        total: 430ms    remaining: 7.24s
56:     learn: 0.1062127        total: 437ms    remaining: 7.23s
57:     learn: 0.1061188        total: 446ms    remaining: 7.24s
58:     learn: 0.1060560        total: 459ms    remaining: 7.32s
59:     learn: 0.1059764        total: 472ms    remaining: 7.39s
60:     learn: 0.1059254        total: 479ms    remaining: 7.37s
61:     learn: 0.1058188        total: 488ms    remaining: 7.38s
62:     learn: 0.1057452        total: 495ms    remaining: 7.36s
63:     learn: 0.1056768        total: 502ms    remaining: 7.34s
64:     learn: 0.1056312        total: 509ms    remaining: 7.32s
65:     learn: 0.1055690        total: 516ms    remaining: 7.3s
66:     learn: 0.1054824        total: 523ms    remaining: 7.29s
67:     learn: 0.1053399        total: 531ms    remaining: 7.28s
68:     learn: 0.1052663        total: 539ms    remaining: 7.27s
69:     learn: 0.1051602        total: 546ms    remaining: 7.26s
70:     learn: 0.1051071        total: 553ms    remaining: 7.24s
71:     learn: 0.1050672        total: 561ms    remaining: 7.23s
72:     learn: 0.1049650        total: 568ms    remaining: 7.21s
73:     learn: 0.1049101        total: 575ms    remaining: 7.2s
74:     learn: 0.1048532        total: 582ms    remaining: 7.18s
75:     learn: 0.1047954        total: 590ms    remaining: 7.17s
76:     learn: 0.1047541        total: 597ms    remaining: 7.16s
77:     learn: 0.1046917        total: 605ms    remaining: 7.15s
78:     learn: 0.1046285        total: 612ms    remaining: 7.13s
79:     learn: 0.1045762        total: 619ms    remaining: 7.12s
80:     learn: 0.1045120        total: 626ms    remaining: 7.11s
81:     learn: 0.1044563        total: 633ms    remaining: 7.09s
82:     learn: 0.1044181        total: 640ms    remaining: 7.08s
83:     learn: 0.1043875        total: 647ms    remaining: 7.06s
84:     learn: 0.1043277        total: 654ms    remaining: 7.04s
85:     learn: 0.1042808        total: 673ms    remaining: 7.15s
86:     learn: 0.1042176        total: 681ms    remaining: 7.14s
87:     learn: 0.1041579        total: 688ms    remaining: 7.13s
```

```
87:     learn: 0.1041379        total: 688ms    remaining: 7.13s
88:     learn: 0.1040936        total: 696ms    remaining: 7.12s
89:     learn: 0.1040500        total: 706ms    remaining: 7.14s
90:     learn: 0.1040069        total: 713ms    remaining: 7.12s
91:     learn: 0.1039391        total: 721ms    remaining: 7.11s
92:     learn: 0.1038705        total: 728ms    remaining: 7.1s
93:     learn: 0.1038233        total: 735ms    remaining: 7.08s
```

```
ss['Target'] = model.predict(test[in_cols])
ss.head()
```

	ID	Target
0	ID_AAcismbB	0.130702
1	ID_AAeBMsji	0.199215
2	ID_AAjFMjzy	0.632204
3	ID_AAmMOEEC	0.414034
4	ID_ABguzDxp	0.259061

```
ss.to_csv('second4_submission.csv', index=False)
```

In this case, it looks like we can improve our performance by specifying a max_depth to limit model complexity.

Finally, let's try a different model out of curiosity:

```
from catboost import CatBoostRegressor

model = CatBoostRegressor()
# Exercise: fit and score the model. Does it beat your other scores? Can you use it to mak
```

Remember, you can ask questions and share ideas in the discussions.

GOOD LUCK!

```
# new attempt
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
import keras
keras.__version__
```

```
'2.4.3'
```

```
train = pd.read_csv('/content/drive/MyDrive/zindi_hackathon/Train.csv')
test = pd.read_csv('/content/drive/MyDrive/zindi_hackathon/Test.csv')
```

```
test = pd.read_csv('/content/drive/MyDrive/zindi_hackathon/Test.csv')
# ss.to_csv('second4_submission.csv', index=False)
```

```
train
```

0	ghsl_built_1990_to_2000	ghsl_built_2000_to_2014	ghsl_not_built_up	ghsl_pop_densi
0	0.000055	0.000536	0.999408	12.1461
0	0.000000	0.000018	0.999872	113.8067
0	0.000000	0.000000	1.000000	0.0000
1	0.000254	0.000228	0.999195	5.2133
0	0.017383	0.099875	0.853533	31.7346
..	...	...	...	
0	0.002313	0.008068	0.978418	44.0443
0	0.000000	0.000000	1.000000	0.0000
2	0.000018	0.000074	0.999279	0.4581
0	0.000000	0.000000	1.000000	0.0000
0	0.002073	0.000337	0.997556	15.2236

```
import numpy as np
np.random.seed(123)

import matplotlib.pyplot as plt
import pandas as pd
import math

import tensorflow as tf
#tf.set_random_seed(1234)

import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN, GRU, LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

%matplotlib inline
```

```
import math
import matplotlib.pyplot as plt
import numpy as np
from numpy.random import seed
seed(1)
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

```
import tensorflow
tensorflow.random.set_seed(1)
from tensorflow.python.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.wrappers.scikit_learn import KerasRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21454 entries, 0 to 21453
Data columns (total 19 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   ID                                       21454 non-null  object
 1   country                                  21454 non-null  object
 2   year                                     21454 non-null  int64
 3   urban_or_rural                           21454 non-null  object
 4   ghsl_water_surface                       21454 non-null  float64
 5   ghsl_built_pre_1975                      21454 non-null  float64
 6   ghsl_built_1975_to_1990                  21454 non-null  float64
 7   ghsl_built_1990_to_2000                  21454 non-null  float64
 8   ghsl_built_2000_to_2014                  21454 non-null  float64
 9   ghsl_not_built_up                        21454 non-null  float64
 10  ghsl_pop_density                         21454 non-null  float64
 11  landcover_crops_fraction                 21454 non-null  float64
 12  landcover_urban_fraction                 21454 non-null  float64
 13  landcover_water_permanent_10km_fraction  21454 non-null  float64
 14  landcover_water_seasonal_10km_fraction   21454 non-null  float64
 15  nighttime_lights                         21454 non-null  float64
 16  dist_to_capital                          21454 non-null  float64
 17  dist_to_shoreline                        21454 non-null  float64
 18  Target                                   21454 non-null  float64
dtypes: float64(15), int64(1), object(3)
memory usage: 3.1+ MB
```

```
# encoding categorical feature using OneHotEncoding
encoded_data = pd.get_dummies(train, prefix_sep = "_",columns =["urban_or_rural"]) #Data E
```

```
'ghsl_built_2000_to_2014', 'ghsl_not_built_up', 'ghsl_pop_density', 'landcover_crops_fracti
```

```
train["country"].unique()
```

```
array(['Ethiopia', 'Mozambique', 'Malawi', 'Guinea', 'Cameroon', 'Ghana',
       'Senegal', 'Kenya', 'Tanzania', 'Mali', 'Swaziland', 'Rwanda',
```

```
           'Nigeria', 'Lesotho', 'Sierra Leone', 'Central African Republic',
           "Cote d'Ivoire", 'Togo'], dtype=object)
```

```
# Applying standardization function earlier defined
categ = encoded_data[['urban_or_rural_R','urban_or_rural_U', "year", "country", "ID"]]


standard_data = pd.concat([data, categ], axis=1)
#return standard_data, data,encoded_data
```

```
standard_data
```

ɔ	ghsl_pop_density	landcover_crops_fraction	landcover_urban_fraction	landcover_wat
2	-0.398706	0.265976	-0.553207	
)	0.086073	2.573071	-0.564932	
5	-0.456627	-0.993019	-0.584730	
3	-0.431766	0.259392	-0.505236	
2	-0.305296	-0.952334	0.371779	
.	...	...	...	
3	-0.246596	-0.506372	-0.324520	
5	-0.456627	0.406797	-0.574595	
1	-0.454442	-1.036011	-0.518097	
5	-0.456627	-0.519518	-0.549805	
)	-0.384031	-0.348767	-0.524288	

```
train.columns
```

```
Index(['ID', 'country', 'year', 'urban_or_rural', 'ghsl_water_surface',
       'ghsl_built_pre_1975', 'ghsl_built_1975_to_1990',
       'ghsl_built_1990_to_2000', 'ghsl_built_2000_to_2014',
       'ghsl_not_built_up', 'ghsl_pop_density', 'landcover_crops_fraction',
       'landcover_urban_fraction', 'landcover_water_permanent_10km_fraction',
       'landcover_water_seasonal_10km_fraction', 'nighttime_lights',
       'dist_to_capital', 'dist_to_shoreline', 'Target'],
      dtype='object')
```

```
# Data Splitting


X = standard_data.drop('Target', axis=1); y = standard_data['Target']

# we are using 80% data for testing
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20, random_state=10)
```

```
from keras import models  # model for building newtork
from keras import layers  # layers of network

def build_model():
    # Because we will need to instantiate
    # the same model multiple times,
    # we use a function to construct it.
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
                           input_shape=(X_train.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1, activation='linear'))
    model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['accuracy'])
    return model


# Model Validation: K-fold Cross-validation


import numpy as np

k = 5
num_val_samples = len(X_train) // k
num_epochs = 100
all_loss_histories = []
all_accuracies = []
for i in range(k):
    print('processing fold #', i)
    # Prepare the validation data: data from partition # k
    val_data = X_train[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = y_train[i * num_val_samples: (i + 1) * num_val_samples]

    # Prepare the training data: data from all other partitions
    partial_train_data = tf.convert_to_tensor.concatenate(
        [X_train[:i * num_val_samples],
         X_train[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = tf.convert_to_tensor.concatenate(
        [y_train[:i * num_val_samples],
         y_train[(i + 1) * num_val_samples:]],
        axis=0)

    # Build the Keras model (already compiled)
    model = build_model()
    # Train the model (in silent mode, verbose=0)
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=25, verbose=0)
    # Evaluate the model on the validation data
    loss, accuracy = model.evaluate(val_data, val_targets, verbose=0)
    all_accuracies.append(accuracy)

    # Train the model (in silent mode, verbose=0)
    history = model.fit(partial_train_data, partial_train_targets,
```

```
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=25, verbose=0)
    loss_history = history.history['val_loss']
    all_loss_histories.append(loss_history)
```

```
processing fold # 0
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-161-5fc62a41da20> in <module>()
     13
     14     # Prepare the training data: data from all other partitions
---> 15     partial_train_data = tf.convert_to_tensor.concatenate(
     16         [X_train[:i * num_val_samples],
     17          X_train[(i + 1) * num_val_samples:]],

AttributeError: 'function' object has no attribute 'concatenate'
```

SEARCH STACK OVERFLOW

```
mean = train_data.mean(axis=0)
# Note that "train_data -= mean" is the same as "train_data = train_data - mean"
# The "/=" operation is the same but with division.
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std


country = df_input.country.astype("category").cat.codes
country = pd.Series(country)


year = df_input.year.astype("category").cat.codes
year = pd.Series(year)


urban_or_rural = df_input.urban_or_rural.astype("category").cat.codes
urban_or_rural = pd.Series(urban_or_rural)


# values = .reshape(-1, 1)
# scaler = MinMaxScaler(feature_range=(-1, 1))
# values = scaler.fit_transform(values)
X = df_input
```

```
y = n_y_output
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1, random_state=10)
```

```
train.columns
```

```
Index(['country', 'year', 'urban_or_rural', 'ghsl_water_surface',
       'ghsl_built_pre_1975', 'ghsl_built_1975_to_1990',
       'ghsl_built_1990_to_2000', 'ghsl_built_2000_to_2014',
       'ghsl_not_built_up', 'ghsl_pop_density', 'landcover_crops_fraction',
       'landcover_urban_fraction', 'landcover_water_permanent_10km_fraction',
       'landcover_water_seasonal_10km_fraction', 'nighttime_lights',
       'dist_to_capital', 'dist_to_shoreline', 'Target'],
      dtype='object')
```

```
#X1_train = X_train.drop('country', axis=1, inplace=True)
X1_train = X_train.drop('year', axis=1)
X1_train = X_train.drop('urban_or_rural', axis=1)

#X1_val = X_val.drop('country', axis=1, inplace=True)
X1_val = X_val.drop('year', axis=1)
X1_val = X_val.drop('urban_or_rural', axis=1)

y_train=np.reshape(y_train, (-1,1))
y_val=np.reshape(y_val, (-1,1))
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()
print(scaler_x.fit(X1_train))
xtrain_scale=scaler_x.transform(X1_train)
print(scaler_x.fit(X1_val))
xval_scale=scaler_x.transform(X1_val)
print(scaler_y.fit(y_train))
ytrain_scale=scaler_y.transform(y_train)
print(scaler_y.fit(y_val))
yval_scale=scaler_y.transform(y_val)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

```
model = Sequential()
model.add(LSTM(10, input_shape=(21454, 19)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

```
     17 print(scaler_y.fit(y_train))
```

```
model = Sequential()
model.add(Dense(10, input_dim=8, kernel_initializer='normal', activation='relu'))
model.add(Dense(2670, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 10)                90
_________________________________________________________________
dense_8 (Dense)              (None, 2670)              29370
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 2671
=================================================================
Total params: 32,131
Trainable params: 32,131
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
history=model.fit(X_train, y_train, epochs=30, batch_size=150, verbose=1, validation_split
predictions = model.predict(xval_scale)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/util/structure.py in
normalize_element(element, element_signature)
    105         if spec is None:
--> 106           spec = type_spec_from_value(t, use_fallback=False)
    107       except TypeError:

15 frames

TypeError: Could not build a TypeSpec for        year urban_or_rural  ...
dist_to_capital  dist_to_shoreline
19138  1995              R  ...       120.626057         846.017440
1810   2014              R  ...        93.953261         258.355387
14399  2011              R  ...       605.340665          81.199135
16840  2014              U  ...       141.147908         386.968970
4535   2009              R  ...       796.093821         161.390197
```

```
nb_epoch = 50

model.fit(df_input, y_output, epochs=nb_epoch)
```

```
---------------------------------------------------------------------------

normalize element(element, element signature)

    107       except TypeError:

TypeError: Could not build a TypeSpec for          country  year  ...
dist_to_capital  dist_to_shoreline
0         Ethiopia  2016  ...      278.788451         769.338378
1         Ethiopia  2005  ...      200.986978         337.135243
2       Mozambique  2009  ...      642.594208         169.913773
3           Malawi  2015  ...      365.349451         613.591610
4           Guinea  2012  ...      222.867189         192.926363
...            ...   ...  ...             ...                ...
21449      Nigeria  2013  ...      283.861037         159.790057
21450      Senegal  2011  ...      295.307249         122.976960
21451        Ghana  2014  ...      166.405249         155.365355
21452        Ghana  2014  ...      568.759665         534.638628
21453   Mozambique  2011  ...     1486.151015         216.519408

[21454 rows x 17 columns] with type DataFrame

During handling of the above exception, another exception occurred:

ValueError                                 Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/constant_op.py in
convert_to_eager_tensor(value, ctx, dtype)
     96       dtype = dtypes.as_dtype(dtype).as_datatype_enum
     97   ctx.ensure_initialized()
---> 98   return ops.EagerTensor(value, ctx.device_name, dtype)
```

0s completed at 17:37