



Ho Chi Minh City University of Technology

Decision Making Techniques Project Semester 211

Trần Đình Trung 1814526

Predicting the Price of Electricity with Machine Learning

"Real-time electricity pricing models can potentially lead to economic and environmental advantages compared to the current common flat rates. In particular, they can provide end users with the opportunity to reduce their electricity expenditures by responding to pricing that varies with different times of the day. However, recent studies have revealed that the lack of knowledge among users about how to respond to time-varying prices as well as the lack of effective building automation systems are two major barriers for fully utilizing the potential benefits of real-time pricing tariffs. We tackle these problems by proposing an optimal and automatic residential energy consumption scheduling framework which attempts to achieve a desired trade-off between minimizing the electricity payment and minimizing the waiting time for the operation of each appliance in household in presence of a real-time pricing tariff combined with inclining block rates. Our design requires minimum effort from the users and is based on simple linear programming computations. Moreover, we argue that any residential load control strategy in real-time electricity pricing environments requires price prediction capabilities. This is particularly true if the utility companies provide price information only one or two hours ahead of time. Simulation results show that the combination of the proposed energy consumption scheduling design and the price predictor filter leads to significant reduction not only in users' payments but also in the resulting peak-to-average ratio in load demand for various load scenarios. Therefore, the deployment of the

proposed optimal energy consumption scheduling schemes is beneficial for both end users and utility companies." -[Amir-Hamed Mohsenian-Rad](#)

Data Contents -

This dataset contains four years of electrical consumption, generation, pricing, and weather data for Spain. Consumption and generation data was retrieved from ENTSOE a public portal for Transmission Service Operator (TSO) data. Settlement prices were obtained from the Spanish TSO Red Electric España.

Column Meanings -

- Time: Datetime index localized to CET
- Generation biomass: biomass generation in MW
- Generation fossil brown coal/lignite: coal/lignite generation in MW
- Generation fossil coal-derived gas: coal gas generation in MW
- Generation fossil gas: gas generation in MW
- Generation fossil hard coal: coal generation in MW
- Generation fossil oil: oil generation in MW
- Generation fossil oil shale: shale oil generation in MW
- Generation fossil peat: peat generation in MW
- Generation geothermal: geothermal generation in MW
- Generation hydro pumped storage aggregated: hydro1 generation in MW
- Generation hydro pumped storage consumption: hydro2 generation in MW
- Generation hydro run-of-river and poundage: hydro3 generation in MW
- Generation hydro water reservoir: hydro4 generation in MW
- Generation marine: sea generation in MW
- Generation nuclear: nuclear generation in MW
- Generation other: other generation in MW
- Generation other renewable: other renewable generation in MW
- Generation solar: solar generation in MW
- Generation waste: waste generation in MW
- Generation wind offshore: wind offshore generation in MW
- Generation wind onshore: wind onshore generation in MW
- Gorecast solar day ahead: forecasted solar generation
- Gorecast wind offshore eday ahead: forecasted offshore wind generation
- Gorecast wind onshore day ahead: forecasted onshore wind generation
- Total load forecast: forecasted electrical demand
- Total load actual: actual electrical demand
- Price day ahead: forecasted price EUR/MWh
- Price actual: price in EUR/MWh

```
In [87]: !pip install eli5
```

```
!pip install category_encoders
!pip install voila
!pip install shap
```

```
Requirement already satisfied: eli5 in /usr/local/lib/python3.7/dist-packages (0.11.0)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from eli5) (1.19.5)
Requirement already satisfied: attrs>16.0.0 in /usr/local/lib/python3.7/dist-packages (from eli5) (21.2.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from eli5) (1.15.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from eli5) (2.11.3)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist-packages (from eli5) (1.0.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (from eli5) (0.10.1)
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.7/dist-packages (from eli5) (0.8.9)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from eli5) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20->eli5) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20->eli5) (3.0.0)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->eli5) (2.0.1)
Requirement already satisfied: category_encoders in /usr/local/lib/python3.7/dist-packages (2.3.0)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (0.10.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.4.1)
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.1.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.0.1)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.19.5)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (0.5.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1->category_encoders) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1->category_encoders) (2.8.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy>=0.5.1->category_encoders) (1.15.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->category_encoders) (3.0.0)
Requirement already satisfied: voila in /usr/local/lib/python3.7/dist-packages (0.2.16)
Requirement already satisfied: jupyter-server<2.0.0,>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from voila) (1.12.0)
Requirement already satisfied: nbconvert<7,>=6.0.0 in /usr/local/lib/python3.7/dist-packages (from voila) (6.3.0)
Requirement already satisfied: nbclient<0.6,>=0.4.0 in /usr/local/lib/python3.7/dist-packages
```

kages (from voila) (0.5.9)
Requirement already satisfied: jupyter-client<7,>=6.1.3 in /usr/local/lib/python3.7/dist-packages (from voila) (6.1.12)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.7/dist-packages (from jupyter-client<7,>=6.1.3->voila) (6.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from jupyter-client<7,>=6.1.3->voila) (2.8.2)
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.7/dist-packages (from jupyter-client<7,>=6.1.3->voila) (4.9.1)
Requirement already satisfied: traitlets in /usr/local/lib/python3.7/dist-packages (from jupyter-client<7,>=6.1.3->voila) (5.1.1)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.7/dist-packages (from jupyter-client<7,>=6.1.3->voila) (22.3.0)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (21.1.0)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (0.12.0)
Requirement already satisfied: websocket-client in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (1.2.1)
Requirement already satisfied: nbformat in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (5.1.3)
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (3.4.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (2.11.3)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (0.12.1)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (0.2.0)
Requirement already satisfied: Send2Trash in /usr/local/lib/python3.7/dist-packages (from jupyter-server<2.0.0,>=0.3.0->voila) (1.8.0)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.7/dist-packages (from anyio<4,>=3.1.0->jupyter-server<2.0.0,>=0.3.0->voila) (2.10)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from anyio<4,>=3.1.0->jupyter-server<2.0.0,>=0.3.0->voila) (3.10.0.2)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.7/dist-packages (from anyio<4,>=3.1.0->jupyter-server<2.0.0,>=0.3.0->voila) (1.2.0)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.7/dist-packages (from nbclient<0.6,>=0.4.0->voila) (1.5.1)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.7/dist-packages (from nbconvert<7,>=6.0.0->voila) (0.1.2)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from nbconvert<7,>=6.0.0->voila) (0.8.4)
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packages (from nbconvert<7,>=6.0.0->voila) (4.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-packages (from nbconvert<7,>=6.0.0->voila) (0.7.1)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.7/dist-packages (from nbconvert<7,>=6.0.0->voila) (1.5.0)
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-packages (from nbconvert<7,>=6.0.0->voila) (0.5.0)
Requirement already satisfied: pygments>=2.4.1 in /usr/local/lib/python3.7/dist-packages (from nbconvert<7,>=6.0.0->voila) (2.6.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.7/dist-packages (from nbconvert<7,>=6.0.0->voila) (0.3)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->jupyter-server<2.0.0,>=0.3.0->voila) (2.0.1)

```

Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /usr/local/lib/python3.7/dist-packages (from nbformat->jupyter-server<2.0.0,>=0.3.0->voila) (2.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->jupyter-client<7,>=6.1.3->voila) (1.15.0)
Requirement already satisfied: pyprocess in /usr/local/lib/python3.7/dist-packages (from terminado>=0.8.3->jupyter-server<2.0.0,>=0.3.0->voila) (0.7.0)
Requirement already satisfied: cffi>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from argon2-cffi->jupyter-server<2.0.0,>=0.3.0->voila) (1.15.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from cffi>=1.0.0->argon2-cffi->jupyter-server<2.0.0,>=0.3.0->voila) (2.21)
Requirement already satisfied: webencodings in /usr/local/lib/python3.7/dist-packages (from bleach->nbconvert<7,>=6.0.0->voila) (0.5.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from bleach->nbconvert<7,>=6.0.0->voila) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->bleach->nbconvert<7,>=6.0.0->voila) (3.0.6)
Requirement already satisfied: shap in /usr/local/lib/python3.7/dist-packages (0.40.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from shap) (1.0.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from shap) (1.19.5)
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-packages (from shap) (0.51.2)
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.7/dist-packages (from shap) (0.0.7)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from shap) (1.1.5)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.7/dist-packages (from shap) (1.3.0)
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.7/dist-packages (from shap) (4.62.3)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.7/dist-packages (from shap) (21.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from shap) (1.4.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>20.9->shap) (3.0.6)
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in /usr/local/lib/python3.7/dist-packages (from numba->shap) (0.34.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from numba->shap) (57.4.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas->shap) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas->shap) (1.15.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->shap) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->shap) (3.0.0)

```

In [88]:

```

# Import Libraries -
import numpy as np # Linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
# Visualization
#import shap

```

```

%matplotlib inline
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import eli5
from eli5.sklearn import PermutationImportance

# Data Wrangling
import numpy as np
import pandas as pd
pd.set_option('display.max_columns',50)

# Model Creation
from xgboost import XGBRegressor
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
from sklearn.linear_model import Ridge, LinearRegression
from category_encoders import OneHotEncoder, OrdinalEncoder
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, RandomizedSearchC
from sklearn.model_selection import train_test_split, cross_val_score, validation_curve
from sklearn.metrics import roc_curve, plot_roc_curve, mean_absolute_error, mean_square

# Warnings Ignore
import warnings
warnings.filterwarnings("ignore")

```

1. Data Exploration / Clean -

In [89]:

```

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

```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

In [90]:

```

# Read in data
df = pd.read_csv('/content/drive/My Drive/BTL_RQD/energy_dataset.csv')

# Preview of data
df.head(5)

```

Out[90]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	4821.0	162.0	0.0
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	4755.0	158.0	0.0

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	4581.0	157.0	0.0
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	4131.0	160.0	0.0
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	3840.0	156.0	0.0

1.1. Data Exploration -

In [91]:

```
# .Describe to show overview of data
df.describe()
```

Out[91]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale
count	35045.000000	35046.000000	35046.0	35046.000000	35046.000000	35045.000000	35046.0
mean	383.513540	448.059208	0.0	5622.737488	4256.065742	298.319789	0.0
std	85.353943	354.568590	0.0	2201.830478	1961.601013	52.520673	0.0
min	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.0
25%	333.000000	0.000000	0.0	4126.000000	2527.000000	263.000000	0.0
50%	367.000000	509.000000	0.0	4969.000000	4474.000000	300.000000	0.0
75%	433.000000	757.000000	0.0	6429.000000	5838.750000	330.000000	0.0
max	592.000000	999.000000	0.0	20034.000000	8359.000000	449.000000	0.0

In [92]:

```
# .Info to show datatype, nulls, and count
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 29 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   time                                35064 non-null  object
1   generation biomass                  35045 non-null  float64
2   generation fossil brown coal/lignite 35046 non-null  float64
3   generation fossil coal-derived gas    35046 non-null  float64
4   generation fossil gas                35046 non-null  float64
```

```

                                final
5  generation fossil hard coal    35046 non-null float64
6  generation fossil oil          35045 non-null float64
7  generation fossil oil shale    35046 non-null float64
8  generation fossil peat         35046 non-null float64
9  generation geothermal          35046 non-null float64
10 generation hydro pumped storage aggregated 0 non-null float64
11 generation hydro pumped storage consumption 35045 non-null float64
12 generation hydro run-of-river and poundage 35045 non-null float64
13 generation hydro water reservoir 35046 non-null float64
14 generation marine              35045 non-null float64
15 generation nuclear             35047 non-null float64
16 generation other               35046 non-null float64
17 generation other renewable     35046 non-null float64
18 generation solar               35046 non-null float64
19 generation waste               35045 non-null float64
20 generation wind offshore       35046 non-null float64
21 generation wind onshore        35046 non-null float64
22 forecast solar day ahead       35064 non-null float64
23 forecast wind offshore eday ahead 0 non-null float64
24 forecast wind onshore day ahead 35064 non-null float64
25 total load forecast            35064 non-null float64
26 total load actual              35028 non-null float64
27 price day ahead                35064 non-null float64
28 price actual                   35064 non-null float64
dtypes: float64(28), object(1)
memory usage: 7.8+ MB

```

```

In [93]: # Null % check amongst columns
         round((df.isnull().sum()/len(df)*100),2)

```

```

Out[93]: time                0.00
         generation biomass    0.05
         generation fossil brown coal/lignite 0.05
         generation fossil coal-derived gas    0.05
         generation fossil gas                 0.05
         generation fossil hard coal           0.05
         generation fossil oil                 0.05
         generation fossil oil shale           0.05
         generation fossil peat                0.05
         generation geothermal                 0.05
         generation hydro pumped storage aggregated 100.00
         generation hydro pumped storage consumption 0.05
         generation hydro run-of-river and poundage 0.05
         generation hydro water reservoir       0.05
         generation marine                   0.05
         generation nuclear                   0.05
         generation other                     0.05
         generation other renewable            0.05
         generation solar                     0.05
         generation waste                     0.05
         generation wind offshore              0.05
         generation wind onshore               0.05
         forecast solar day ahead              0.00
         forecast wind offshore eday ahead     100.00
         forecast wind onshore day ahead        0.00
         total load forecast                   0.00
         total load actual                     0.10

```


price day ahead0.00

price actual0.00

dtype: float64

```
In [94]: # DF.corr to show correlation of values
df.corr()
```

Out[94]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	gen fos
generation biomass	1.000000	0.229809	NaN	-0.021660	0.433522	0.459530	NaN	
generation fossil brown coal/lignite	0.229809	1.000000	NaN	0.499808	0.768710	0.314869	NaN	
generation fossil coal- derived gas	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
generation fossil gas	-0.021660	0.499808	NaN	1.000000	0.541635	0.309623	NaN	
generation fossil hard coal	0.433522	0.768710	NaN	0.541635	1.000000	0.440837	NaN	
generation fossil oil	0.459530	0.314869	NaN	0.309623	0.440837	1.000000	NaN	
generation fossil oil shale	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
generation fossil peat	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
generation geothermal	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
generation hydro pumped storage aggregated	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
generation hydro pumped storage consumption	-0.044898	-0.323771	NaN	-0.420646	-0.406116	-0.331011	NaN	
generation hydro run- of-river and poundage	-0.284877	-0.525005	NaN	-0.271527	-0.497940	-0.106753	NaN	

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	gen fos
generation hydro water reservoir	-0.033675	-0.229455	NaN	0.060173	-0.157677	0.160465	NaN	
generation marine	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
generation nuclear	-0.021279	-0.008440	NaN	-0.112904	-0.023930	0.015619	NaN	
generation other	0.658488	0.097600	NaN	-0.066279	0.264383	0.375046	NaN	
generation other renewable	-0.560588	0.104552	NaN	0.334880	-0.019426	-0.115087	NaN	
generation solar	-0.004687	0.040447	NaN	0.074716	0.046185	0.100211	NaN	
generation waste	-0.346343	0.282810	NaN	0.275053	0.170235	-0.175741	NaN	
generation wind offshore	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
generation wind onshore	-0.068658	-0.434118	NaN	-0.397298	-0.441853	-0.051787	NaN	
forecast solar day ahead	-0.008713	0.042306	NaN	0.080171	0.047356	0.096435	NaN	
forecast wind offshore eday ahead	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
forecast wind onshore day ahead	-0.072368	-0.436031	NaN	-0.397303	-0.444490	-0.058244	NaN	
total load forecast	0.085216	0.278503	NaN	0.543711	0.394291	0.498637	NaN	
total load actual	0.083288	0.280461	NaN	0.548913	0.396564	0.497089	NaN	
price day ahead	0.108945	0.567905	NaN	0.640895	0.671596	0.292793	NaN	
price actual	0.142369	0.364088	NaN	0.461706	0.465641	0.284679	NaN	

```
In [95]: # Correlation of columns to target variable
correlations = df.corr(method='pearson')
print(correlations['price actual'].sort_values(ascending=False).to_string())
```

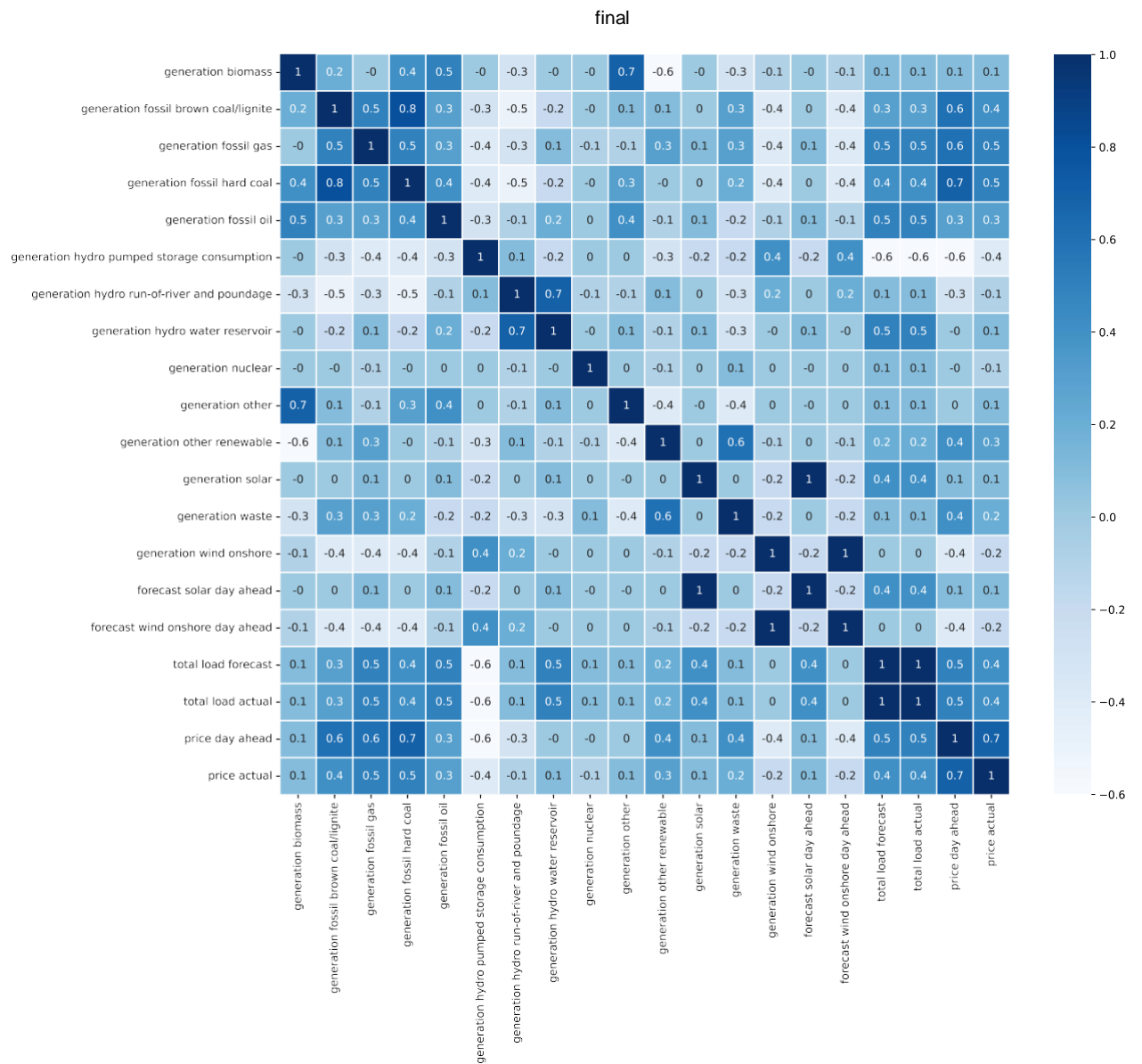
price actual	1.000000
price day ahead	0.732155
generation fossil hard coal	0.465641
generation fossil gas	0.461706
total load actual	0.436127
total load forecast	0.435864
generation fossil brown coal/lignite	0.364088
generation fossil oil	0.284679
generation other renewable	0.256181
generation waste	0.169605
generation biomass	0.142369
forecast solar day ahead	0.101402
generation other	0.100048
generation solar	0.098488
generation hydro water reservoir	0.071549
generation nuclear	-0.052596
generation hydro run-of-river and poundage	-0.137106
generation wind onshore	-0.220830
forecast wind onshore day ahead	-0.221706
generation hydro pumped storage consumption	-0.426417
generation fossil coal-derived gas	NaN
generation fossil oil shale	NaN
generation fossil peat	NaN
generation geothermal	NaN
generation hydro pumped storage aggregated	NaN
generation marine	NaN
generation wind offshore	NaN
forecast wind offshore eday ahead	NaN

```
In [96]: # Assign Variable to drop columns
zero_val_cols = ['generation marine',
                  'generation geothermal',
                  'generation fossil peat',
                  'generation wind offshore',
                  'generation fossil oil shale',
                  'forecast wind offshore eday ahead',
                  'generation fossil coal-derived gas',
                  'generation hydro pumped storage aggregated']

# Drop Columns with zero values
heat_map_features = df.drop(columns=zero_val_cols,axis=1)

# Set Figure Size
plt.figure(figsize=(15,12.5))

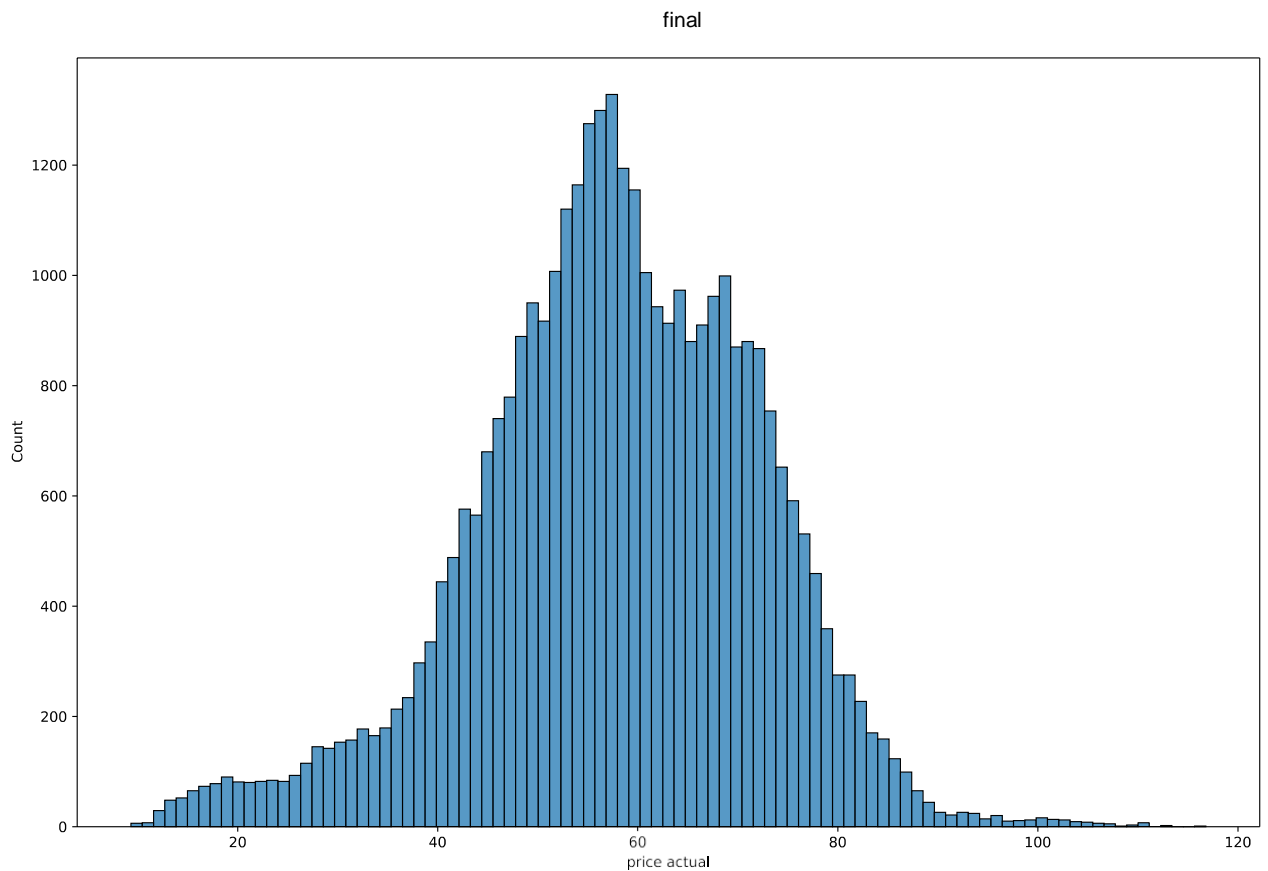
# .corr heatmap of df to visualize correlation & show plot
sns.heatmap(round(heat_map_features.corr(),1),annot=True,cmap='Blues',linewidth=0.9)
plt.show();
```



In [97]:

```
# Figure Size
plt.figure(figsize=(15,10))

# Hist graph to show distribution of target variable
sns.histplot(df,x='price actual');
```



1.2. Data Exploration Findings -

- ♦ Normal distribution of target variable.
- ♦ Low percentage of nan values in columns without 100% nan.
- ♦ Lots of columns with 0 values for certain energy generation types.
- ♦ Found very high correlation between some columns.
- ♦ All data seems to be numeric.

1.3. Data Cleaning -

In [98]:

```
# Wrangle function to clean data
def wrangle(filepath):

    # Read in the data, parse dates, and set the index
    df = pd.read_csv(filepath, parse_dates=['time'], index_col='time')

    # Rename columns by replacing all - or blank space with _
    df.columns = df.columns.str.replace(' ', '_').str.replace('-', '_')

    # Make the index DT
    df.index = pd.to_datetime(df.index, utc=True)

    # Drop all columns with data leakage, or 90% + null
    df.drop(columns=['price_day_ahead',
                    'generation_marine',
                    'total_load_forecast',
                    'generation_geothermal',
                    'generation_fossil_peat',
```

```

final
'generation_wind_offshore',
'forecast_solar_day_ahead',
'generation_fossil_oil_shale',
'forecast_wind_onshore_day_ahead',
'forecast_wind_offshore_eday_ahead',
'generation_fossil_coal_derived_gas',
'generation_hydro_pumped_storage_aggregated'], inplace=True)

# Drop Outlier row 2014 for plotting
df = df.drop(pd.Timestamp('2014-12-31 23:00:00+00:00'))

# Sort index
df = df.sort_index()
# Rename columns
df = df.rename(columns={'generation_fossil_brown_coal/lignite': 'generation_fossil_b
# Set conditional statements for filtering times of month to season value
condition_winter = (df.index.month>=1)&(df.index.month<=3)
condition_spring = (df.index.month>=4)&(df.index.month<=6)
condition_summer = (df.index.month>=7)&(df.index.month<=9)
condition_autumn = (df.index.month>=10)&(df.index.month<=12)

# Create column in dataframe that inputs the season based on the conditions created
df['season'] = np.where(condition_winter, 'winter',
                        np.where(condition_spring, 'spring',
                        np.where(condition_summer, 'summer',
                        np.where(condition_autumn, 'autumn', np.nan

return df

# Applying the wrangle function to the dataset
df=wrangle('/content/drive/My Drive/BTL_RQD/energy_dataset.csv')
df

```

Out[98]:

	generation_biomass	generation_fossil_brown_coal	generation_fossil_gas	generation_fossi
time				
2015-01-01 00:00:00+00:00	449.0	328.0	5196.0	
2015-01-01 01:00:00+00:00	448.0	323.0	4857.0	
2015-01-01 02:00:00+00:00	438.0	254.0	4314.0	
2015-01-01 03:00:00+00:00	428.0	187.0	4130.0	
2015-01-01 04:00:00+00:00	410.0	178.0	4038.0	
...	
2018-12-31 18:00:00+00:00	297.0	0.0	7634.0	
2018-12-31 19:00:00+00:00	296.0	0.0	7241.0	

	generation_biomass	generation_fossil_brown_coal	generation_fossil_gas	generation_fossi
time				
2018-12-31 20:00:00+00:00	292.0	0.0	7025.0	
2018-12-31 21:00:00+00:00	293.0	0.0	6562.0	
2018-12-31 22:00:00+00:00	290.0	0.0	6926.0	

35063 rows × 17 columns

1.4. Data Cleaning Notes -

- ♦ Removed forecasted columns to prevent data leakage.
- ♦ Removed all columns with 0 fill for all values.
- ♦ Removed only row from 2014 for plotting purposes.
- ♦ Feature engineered seasons using time of year.

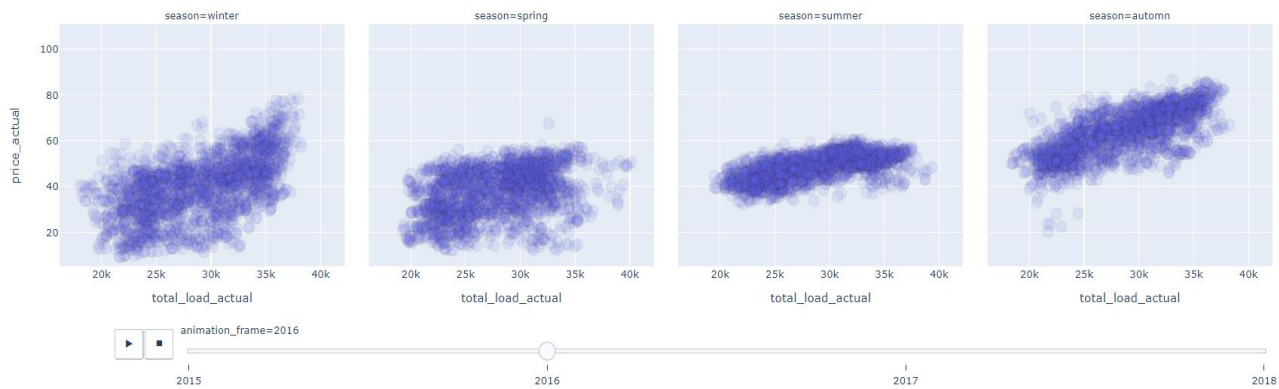
In [99]:

```
# Figure showing Price per total Load
fig = px.scatter(df,x='total_load_actual',
                 y='price_actual',
                 facet_col='season',
                 opacity=0.1,
                 title='Price Per KW Hour Compaired To Total Energy Genereated Per Seas

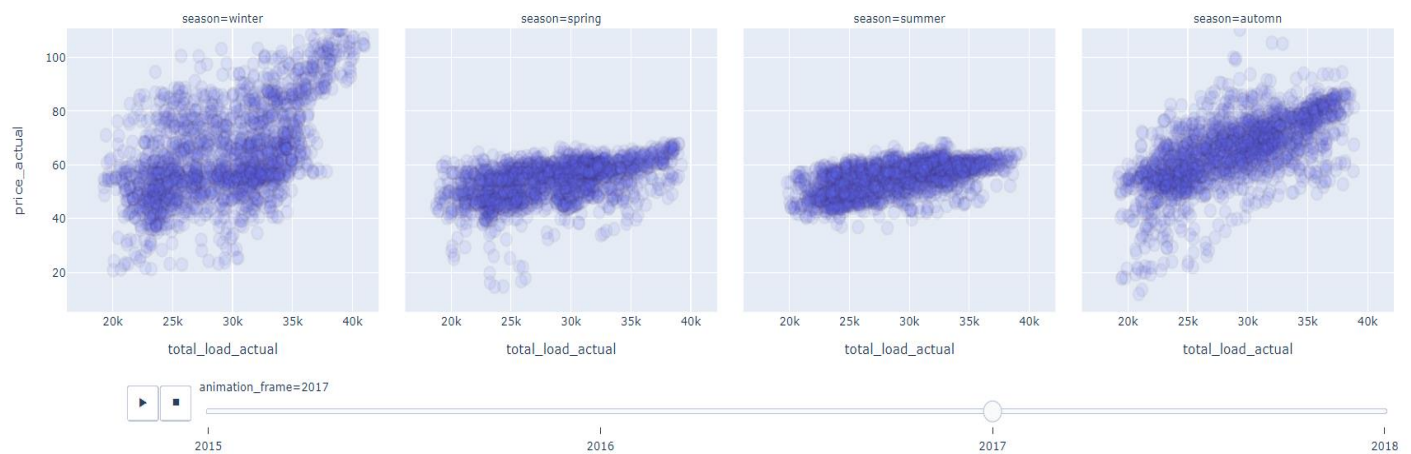
# Figure customizations
fig.update_traces(marker=dict(size=12,
                              line=dict(width=2,
                                          color='darkslateblue')),
                  selector=dict(mode='markers'))
```



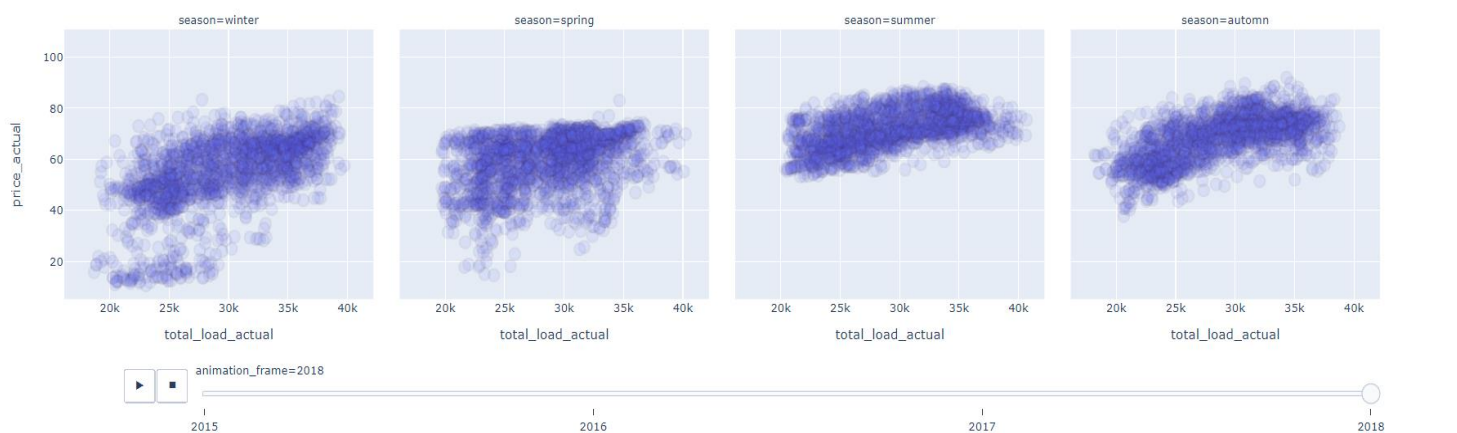
Price Per KW Hour Compared To Total Energy Generated Per Season



Price Per KW Hour Compared To Total Energy Generated Per Season



Price Per KW Hour Compared To Total Energy Generated Per Season



2. ML Model Creation -

2.1. Split Data -

In [100...

```
# Create Target variable
target='price_actual'

# Split data into feature matrix and target vector
y,X=df[target],df.drop(columns=target)
```


2.2. Baselines -

```
In [101...  # Assign variables for baselines and calculate baselines
y_pred = [y_train.mean()*len(y_train)]
mean_baseline_pred = y_train.mean()
baseline_mae = mean_absolute_error(y_train,y_pred)
baseline_rmse = mean_squared_error(y_train,y_pred,squared=False)

# Print statement to show all baseline values
print('Mean Price Per KW/h Baseline Pred:', mean_baseline_pred)
print('-----')
print('Baseline Mae:',baseline_mae)
print('-----')
print('Baseline RMSE:',baseline_rmse)
```

Mean Price Per KW/h Baseline Pred: 57.92005454545454

Baseline Mae: 11.08589652244369

Baseline RMSE: 14.21051579246629

2.3. Linear Regression Model Creation -

```
In [102...  # One Hot Encoder to transform Seasons column
onehot = OneHotEncoder(use_cat_names=True)
```

```

onehot_fit = onehot.fit(X_train)
XT_train = onehot.transform(X_train)
XT_val = onehot.transform(X_val)

# Simple imputer to fill nan values, then transform sets
simp = SimpleImputer(strategy='mean')
simp_fit = simp.fit(XT_train)
XT_train = simp.transform(XT_train)
XT_val = simp.transform(XT_val)

# Assigning model variables
model_lr=LinearRegression()

# Fitting models
model_lr.fit(XT_train,y_train);

# Def to check model metrics of baseline performance
print(model_lr)
print('=====')
print('Training MAE:', mean_absolute_error(y_train,model_lr.predict(XT_train)))
print('-----')
print('Validation MAE:', mean_absolute_error(y_val,model_lr.predict(XT_val)))
print('-----')
print('Validation R2 score:', model_lr.score(XT_val,y_val))
print('=====')

```

```

LinearRegression()
=====
Training MAE: 8.064818613080801
-----
Validation MAE: 8.12846659640155
-----
Validation R2 score: 0.4265793720525829
=====

```

2.4 XGB RandomSearch -

In [103...

```

# Pipeline variable for RandomSearch
pipe_rs_xgb = make_pipeline(OrdinalEncoder(),
                             SimpleImputer(),
                             XGBRegressor(random_state=42,
                                           n_jobs=-1))

# Params for RandomSearch
paramajama = {'simpleimputer_strategy':['meadian','mean'],
              'xgbregressor_max_depth':range(5,35,5),
              'xgbregressor_learning_rate':np.arange(0.2,1,0.1),
              'xgbregressor_booster':['gbtree','gblinear','dart'],
              'xgbregressor_min_child_weight':range(1,10,1),
              'xgbregressor_gamma':np.arange(0,1,0.1),
              'xgbregressor_max_delta_step':np.arange(0,1,0.1),
              'xgbregressor_subsample':np.arange(0.5,1,0.1)}

# RandomSearch Model
model_rs_xgbr = RandomizedSearchCV(pipe_rs_xgb,
                                    param_distributions = paramajama,

```

```

n_iter=20,
n_jobs=-1)

# Model fit
model_rs_xgbr.fit(X_train,y_train);

# Check model metrics
print('Training MAE:', mean_absolute_error(y_train,model_rs_xgbr.predict(X_train)))
print('-----')
print('Validation MAE:', mean_absolute_error(y_val,model_rs_xgbr.predict(X_val)))
print('-----')
print('R2 score:', model_rs_xgbr.score(X_val,y_val))
print('=====')
model_rs_xgbr.best_params_

```

[11:35:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

Training MAE: 2.1292366572089376

Validation MAE: 4.063467392107568

R2 score: 0.8315261170197177

=====

```

{'simpleimputer_strategy': 'mean',
 'xgbregressor_booster': 'gbtree',
 'xgbregressor_gamma': 0.5,
 'xgbregressor__learning_rate': 0.9000000000000001,
 'xgbregressor__max_delta_step': 0.9,
 'xgbregressor__max_depth': 15,
 'xgbregressor__min_child_weight': 1,
 'xgbregressor__subsample': 0.5}

```

Out[103]...

2.4.1. XGB RandomSearch Conclusions -

(20 random searches with 100 iter. Scaled down for kaggle notebook import)

- ♦ RandomSearch model out performed the default with tuning.
- ♦ Highest score listed - .8991
 - Highest scoring model.
- ♦ I'm interested in grid search performance post project to see if the model can outperform the random searches.

Best Score -

- ♦ Training MAE: 0.18182096506754555
- ♦ Validation MAE: 3.143150404572878
- ♦ R2 score: 0.8991961935947261

Params -

- ♦ 'xgbregressor_subsample': 0.7,
- ♦ 'xgbregressor_min_child_weight': 4,

- 'xgbregressor__max_depth': 25,
- 'xgbregressor__max_delta_step': 0.0,
- 'xgbregressor__learning_rate': 0.2,
- 'xgbregressor__gamma': 0.5,
- 'xgbregressor__booster': 'dart',
- 'simpleimputer_strategy': 'mean'}

Second Best Score -

- Training MAE: 0.22435663695989874
- Validation MAE: 3.6606957501289457
- R2 score: 0.8619067640247029

Creat Best XGB Regressor model

In [104...

```
# Ordinal Encoder to transform Seasons column
ordinal = OrdinalEncoder()
ordinal_fit = ordinal.fit(X_train)
XT_train = ordinal.transform(X_train)
XT_val = ordinal.transform(X_val)

# Simple imputer to fill nan values, then transform sets
simp = SimpleImputer(strategy='mean')
simp_fit = simp.fit(XT_train)
XT_train = simp.transform(XT_train)
XT_val = simp.transform(XT_val)

# Assigning model variables
model_xgbr=XGBRegressor(learning_rate = 0.2,
                        subsample = 0.7,
                        min_weight_fraction_leaf = 4,
                        max_depth = 25,
                        max_delta_step = 0.0,
                        gamma = 0.5,
                        strategy = 'mean')

# Fitting models
model_xgbr.fit(XT_train,y_train)

# Def to check model metrics of baseline performance

print(model_xgbr)
print('=====')
print('Training MAE:', mean_absolute_error(y_train,model_xgbr.predict(XT_train)))
print('-----')
print('Validation MAE:', mean_absolute_error(y_val,model_xgbr.predict(XT_val)))
print('-----')
print('Validation R2 score:', model_xgbr.score(XT_val,y_val))
print('=====')
```

[11:35:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

```
XGBRegressor(gamma=0.5, learning_rate=0.2, max_delta_step=0.0, max_depth=25,
              min_weight_fraction_leaf=4, strategy='mean', subsample=0.7)
=====
Training MAE: 0.1702937561551942
-----
Validation MAE: 3.271356720918258
-----
Validation R2 score: 0.8910092610762533
=====
```

Model Results

LinearRegression -

- Training MAE: 8.064818613080801
- Validation MAE: 8.12846659640155
- R2 score: 0.4265793720525829

XGBRegressor -

- Training MAE: 0.1702937561551942
- Validation MAE: 3.271356720918258
- R2 score: 0.8910092610762533

3. Model Visualizations -

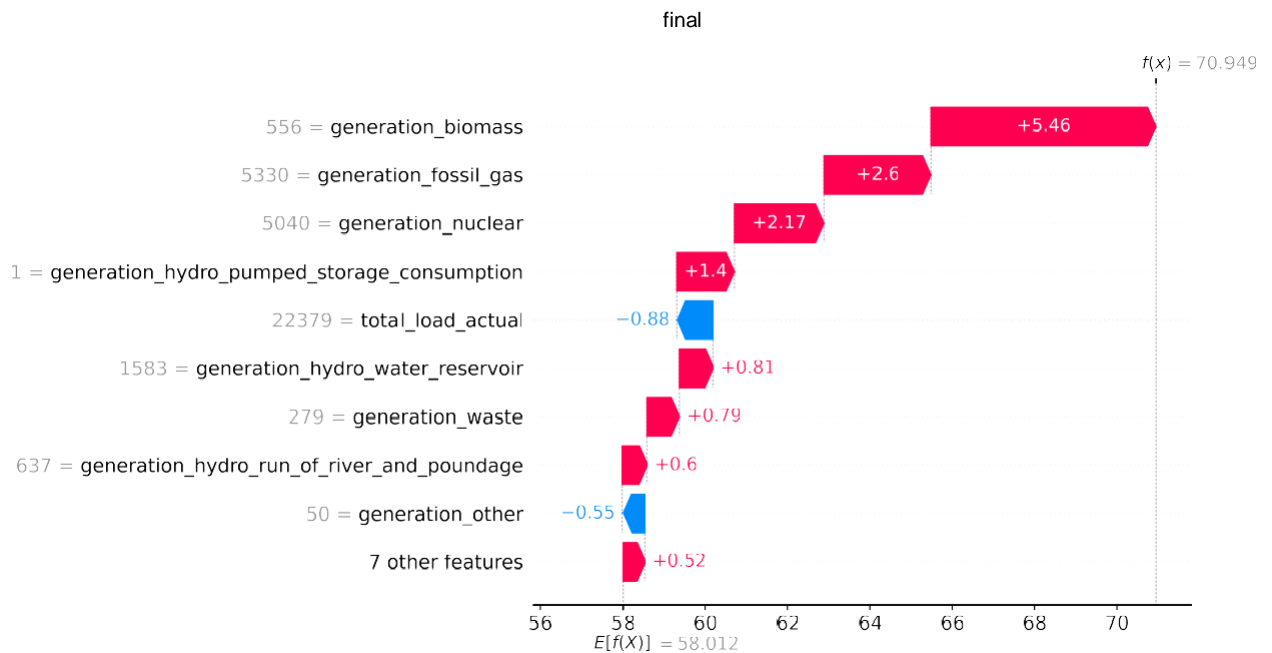
3.1 Feature Importance -

In [105...

```
#Set samp variable to show features when plotting
samp = pd.DataFrame(XT_val, columns=ordinal_fit.get_feature_names())
```

In [106...

```
import shap
# Shap waterfall plot showing feature importance
explainer = shap.TreeExplainer(model_xgbr)
shap_values=explainer(samp.head(1))
shap.plots.waterfall(shap_values[0])
```



In [107...

```
# Shap force plot also showing feature importance
explainer = shap.TreeExplainer(model_xgbr)
shap_values = explainer.shap_values(samp.head(1))
shap.initjs()
shap.force_plot(base_value = explainer.expected_value,
                shap_values=shap_values,
                features=samp.head(1))
```



In []:

```
# Permutation importance for features used in XGBR model
perm = PermutationImportance(model_xgbr, random_state=42).fit(XT_val, y_val)
eli5.show_weights(perm, feature_names = samp.columns.tolist())
```

3.2 Feature/Permutation Importance Conclusions -

- Generation biomass seems to have the highest importance, followed by generation_fossil_gas and generation nuclear.
- The feature engineered column seems to have the highest permutation importance value, followed by generation fossil gas.

4 Model Deployment

In fact, the model will be integrated into the electricity tariff system to receive real data in real time.

But because objective conditions do not allow, in this scale of exercise I can only implement a manual adjustment system so that everyone understands how the model works.

```
In [ ]: def model_make_prediction(season,generation_fossil_gas,generation_fossil_hard_coal,gene
        total_load_actual,generation_hydro_run_of_river_and_poundage
        ,generation_nuclear,generation_waste,generation_hydro_water_rese
        generation_biomass,generation_hydro_pumped_storage_consumption,
        generation_fossil_oil,generation_other,generation_solar,generati
        generation_wind_onshore):

    X_new = pd.DataFrame({
        'generation_biomass': [generation_biomass],
        'generation_fossil_brown_coal or lignite': [generation_fossil_brown_coal],
        'generation_fossil_gas': [generation_fossil_gas],
        'generation_fossil_hard_coal': [generation_fossil_hard_coal],
        'generation_fossil_oil': [generation_fossil_oil],
        'generation_hydro_pumped_storage_consumption': [generation_hydro_pumped_storage_con
        'generation_hydro_run_of_river_and_poundage': [generation_hydro_run_of_river_and_po
        'generation_hydro_water_reservoir': [generation_hydro_water_reservoir],
        'generation_nuclear': [generation_nuclear],
        'generation_other': [generation_other],
        'generation_other_renewable': [generation_other_renewable],
        'generation_solar': [generation_solar],
        'generation_waste': [generation_waste],
        'generation_wind_onshore': [generation_wind_onshore],
        'total_load_actual': [total_load_actual],
        'Season': [season]})
    y_new = model_xgbr.predict(X_new.values)
    print("The Price could be : ", y_new,"EUR/MWh")
```

```
In [ ]: from ipywidgets import interactive, HBox,Layout,Button,GridspecLayout
```

```
In [ ]: widget = interactive(model_make_prediction,
        generation_biomass=(0,1000,1),
        generation_fossil_brown_coal=(0,1500,1),
        generation_fossil_gas=(0,25000,1),
        generation_fossil_hard_coal=(0,10000,1),
        generation_fossil_oil=(0,1000,1),
        generation_hydro_pumped_storage_consumption=(0,10000,1),
        generation_hydro_run_of_river_and_poundage=(0,3000,1),
        generation_hydro_water_reservoir=(0,15000,1),
        generation_nuclear=(0,10000),
        generation_other=(0,200,1),
        generation_other_renewable=(0,200,1),
        generation_solar=(0,1000,1),
        generation_waste=(0,500,1),
        generation_wind_onshore=(0,20000,1),
        total_load_actual=(15000,45000,1),
        season=(1,4,1))
```

```
In [ ]: controls = HBox(widget.children[:-1], layout = Layout(flex_flow='column wrap'),width='a
        output = widget.children[-1]
```

```
In [ ]: def create_expanded_button(description, button_style):
        return Button(description=description, button_style=button_style, layout=Layout(hei
```

```
In [ ]: grid = GridspecLayout(18, 6, height='600px')
        grid[0,:] = create_expanded_button('Predict the Price of Energy', 'success')
        grid[2:15, 0:2] = controls
        grid[2:15, 2:] = output
        grid[16,:] = create_expanded_button('\xa9 2021 Copyright Trung_dz', 'info')

        grid
```

```
In [ ]: !jupyter nbconvert --to html /content/drive/MyDrive/BTL_RQD/final.ipynb
```



XGBoost Regression: Giải thích cho tôi như thể tôi 10 tuổi

Hãy bắt đầu với tập dữ liệu đào tạo của chúng tôi bao gồm năm người. Chúng tôi ghi lại tuổi của họ, cho dù họ có bằng thạc sĩ hay không và mức lương của họ (tính bằng hàng nghìn). Mục tiêu của chúng tôi là dự đoán *mức lương* bằng Thuật toán XGBoost.

AGE	MASTER'S DEGREE	SALARY
23	No	50
24	Yes	70
26	Yes	80
26	No	65
27	Yes	85

Hình ảnh của Tác giả

Bước 1: Dự đoán ban đầu và tính toán phần dư

Dự đoán này có thể là bất cứ điều gì. Nhưng giả sử dự đoán ban đầu của chúng tôi là giá trị trung bình của các biến mà chúng tôi muốn dự đoán.

$$\frac{50 + 70 + 80 + 65 + 85}{5} = 70$$

Chúng ta có thể tính toán lượng dư bằng công thức sau:

$$\text{Residuals} = \text{Observed values} - \text{Predicted values}$$

Ở đây, Giá trị quan sát của chúng tôi là các giá trị trong cột *Lương* và tất cả các Giá trị được dự đoán đều bằng 70 vì đó là giá trị mà chúng tôi đã chọn dự đoán ban đầu của mình.

AGE	MASTER'S DEGREE	SALARY	RESIDUALS
23	No	50	-20
24	Yes	70	0
26	Yes	80	10
26	No	65	-5
27	Yes	85	15

Hình ảnh của Tác giả


Bước 2: Xây dựng cây XGBoost

Mỗi cây bắt đầu với một lá duy nhất và tất cả phần còn lại sẽ đi vào lá đó.

-20, 0, 10, -5, 15

Bây giờ chúng ta cần tính toán một thứ gọi là **Điểm giống nhau** của lá này.

$$\text{Similarity Score} = \frac{(\text{Sum of Residuals})^2}{\text{Number of Residuals} + \lambda}$$

Regularization Parameter 

Hình ảnh của Tác giả

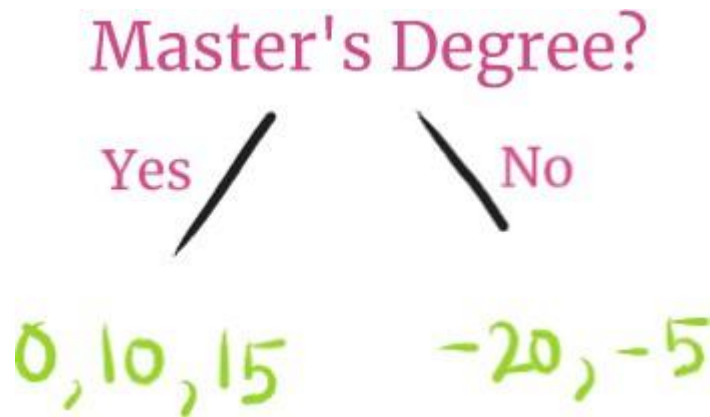
λ (lambda) là một tham số chính quy làm giảm độ nhạy của dự đoán đối với các quan sát riêng lẻ và ngăn việc trang bị quá nhiều dữ liệu (đây là khi một mô hình khớp chính xác với tập dữ liệu huấn luyện). Giá trị mặc định của λ là 1 vì vậy chúng ta sẽ đặt $\lambda = 1$ trong ví dụ này.

$$\frac{(-20 + 0 + 10 - 5 + 15)^2}{5 + 1}$$

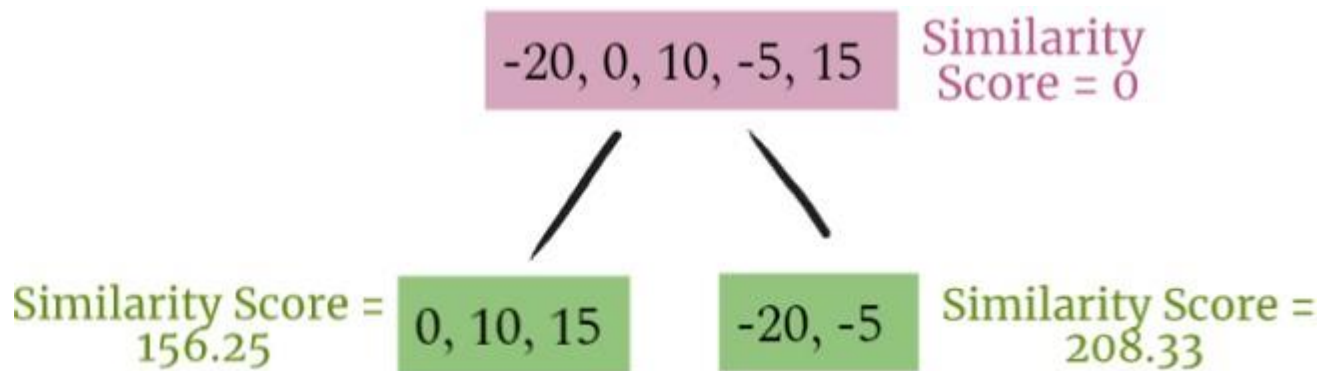
Bây giờ chúng ta nên xem liệu chúng ta có thể làm tốt hơn công việc phân cụm các phần còn lại hay không nếu chúng ta chia chúng thành hai nhóm bằng cách sử dụng các ngưỡng dựa trên các yếu tố dự đoán của chúng ta - *Tuổi* và *Bằng Thạc sĩ*?. Tách các phần còn lại về cơ bản có nghĩa là chúng ta đang thêm các nhánh vào cây của

mình.

Đầu tiên, chúng ta hãy thử tách chiếc lá bằng cách sử dụng *Bảng Thạc sĩ*?



Và sau đó tính **Điểm giống nhau** cho các lá bên trái và bên phải của phần chia trên:



Bây giờ chúng ta cần phải định lượng xem những chiếc lá có phần *Thặng dư* tương tự tốt hơn bao nhiêu so với phần rế. Chúng ta có thể làm điều này bằng cách tính toán **Lợi nhuận** khi chia *Phần còn lại* thành hai nhóm. Nếu **Gain** là dương, thì bạn nên tách ra, nếu không thì không.

$$\text{Gain} = \text{LeftSimilarity} + \text{RightSimilarity} - \text{RootSimilarity}$$
$$= 156.25 + 208.33 - 0 = 364.5833$$

Sau đó, tôi cần so sánh Mức **tăng** này với mức **tăng** trong *Age*. Vì *Tuổi* là một biến số liên tục, nên quá trình để tìm ra các phần khác nhau có liên quan nhiều hơn một chút. Đầu tiên, cần sắp xếp các hàng trong tập dữ liệu của chúng ta theo thứ tự tăng dần của *Tuổi*. Sau đó, chúng tôi tính giá trị trung bình của các giá trị liền kề trong *Tuổi*.

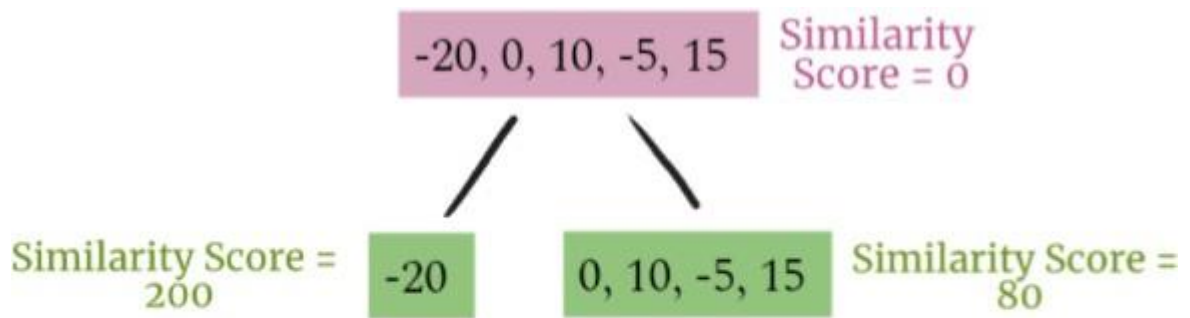
AGE
23
24
26
26
27

$\left. \begin{array}{c} 23 \\ 24 \end{array} \right\} 23.5$
 $\left. \begin{array}{c} 26 \\ 26 \end{array} \right\} 26$
 $\left. \begin{array}{c} 26 \\ 27 \end{array} \right\} 26.5$

Bây giờ chúng tôi chia *Phần còn lại* bằng cách sử dụng bốn mức trung bình làm ngưỡng và tính toán Mức **tăng** cho mỗi phần tách. Phần tách đầu tiên sử dụng *Tuổi* < 23,5 :

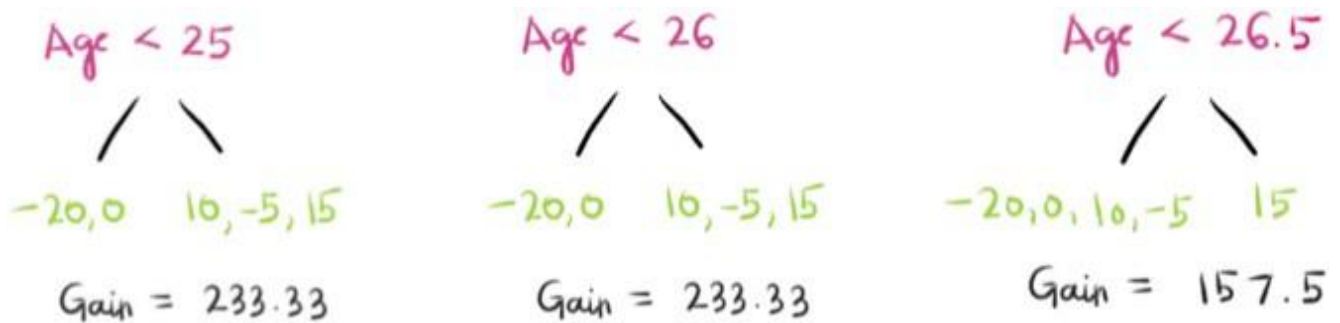
$Age < 23.5$
 $\swarrow \quad \searrow$
 $-20 \quad 0, 10, -5, 15$

Đối với sự phân chia này, chúng tôi tìm thấy **Điểm giống nhau** và **Đạt** được giống như cách chúng tôi đã làm đối với *Bảng Thạc sĩ*?



$$\text{Gain} = 200 + 80 - 0 = 280$$

Làm điều tương tự cho phần còn lại của *Thời đại* :



Ra khỏi một *Bảng cấp của Mater*? chia và bốn *Tuổi* chia tách, các *Bảng thực sĩ* chia có vĩ đại nhất **Gain** giá trị, vì vậy chúng tôi sẽ sử dụng như là phân chia ban đầu của chúng tôi. Bây giờ chúng ta có thể thêm nhiều cành vào cây bằng cách tách *Bảng Thực sĩ* của chúng ta ? lại sử dụng quy trình tương tự được mô tả ở trên. Nhưng, chỉ lần này, chúng ta sử dụng *Bảng Thực sĩ* ban đầu ? rồi làm nút gốc của chúng tôi và thử tách chúng bằng cách nhận giá trị **Gain** lớn nhất lớn hơn 0.

Hãy bắt đầu với nút bên trái. Đối với nút này, chúng tôi chỉ xem xét các quan sát có giá trị 'Có' trong *Bảng Thực sĩ*? bởi vì chỉ những quan sát đó mới hạ cánh ở nút bên trái.

AGE	MASTER'S DEGREE	SALARY	RESIDUALS
23	No	50	-20
24	Yes	70	0
26	Yes	80	10
26	No	65	-5
27	Yes	85	15

Hình ảnh của Tác giả

Vì vậy, chúng tôi tính toán phân tách **Gain** of the *Age* bằng cách sử dụng quy trình tương tự như trước đây, nhưng lần này chỉ sử dụng *Phần dư* trong các hàng được đánh dấu.

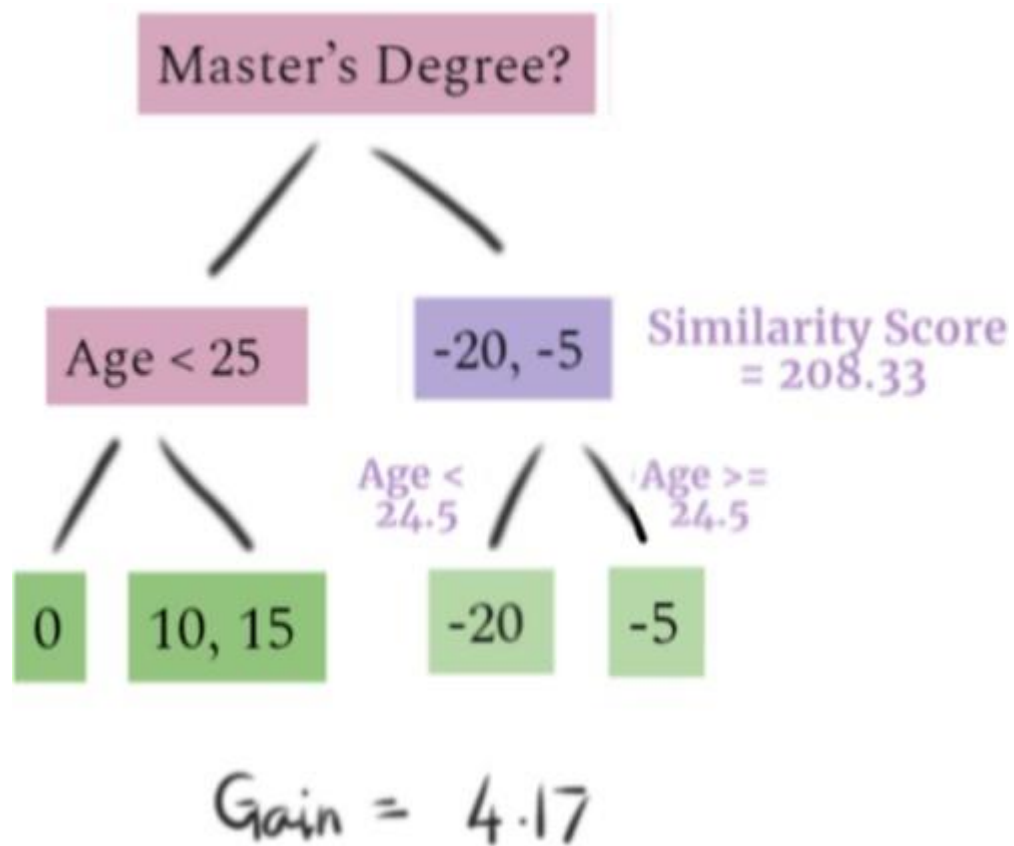


Vì chỉ *Tuổi* <25 mới cho chúng ta Mức **tăng** dương , nên chúng ta chia nút bên trái bằng cách sử dụng ngưỡng này. Chuyển sang nút bên phải của chúng tôi, chúng tôi chỉ xem xét các giá trị có giá trị "Không" trong *Bảng Thạc sĩ*?

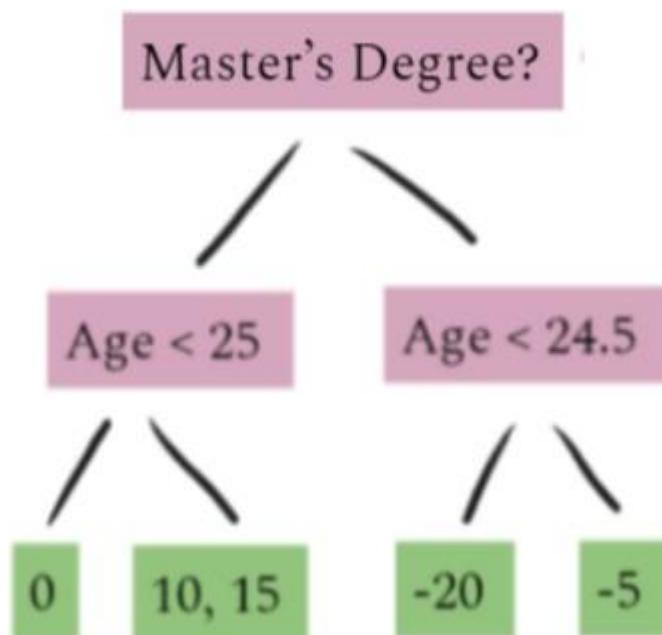
AGE	MASTER'S DEGREE	SALARY	RESIDUALS
23	No	50	-20
24	Yes	70	0
26	Yes	80	10
26	No	65	-5
27	Yes	85	15

Hình ảnh của Tác giả

Chúng tôi chỉ có hai quan sát trong nút bên phải của chúng tôi, vì vậy sự phân tách duy nhất có thể là *Tuổi* <24,5 vì đó là giá trị trung bình của hai giá trị *Tuổi* trong các hàng được đánh dấu.

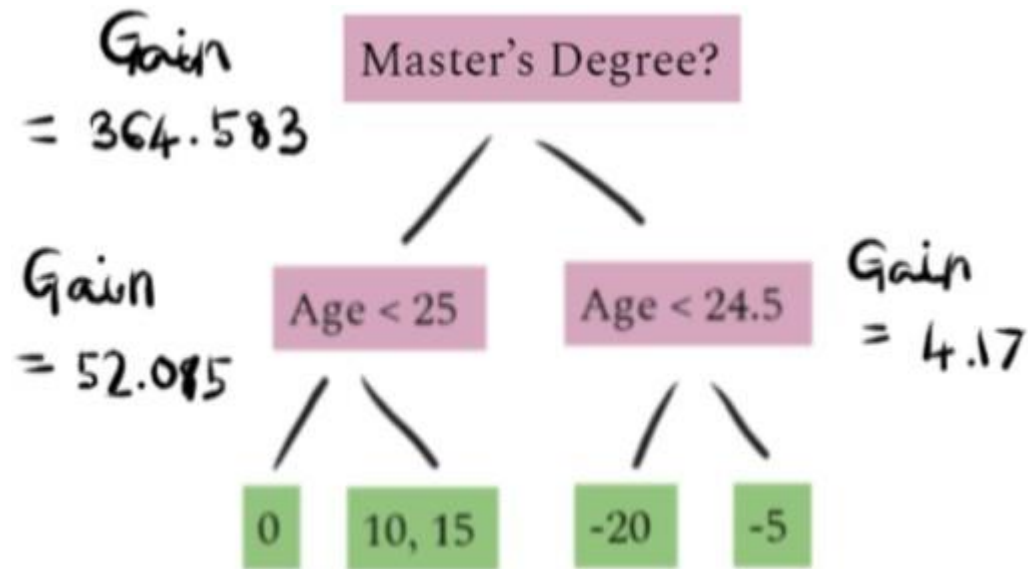


Các **Gain** của chia này là tích cực, vì vậy cây thức của chúng tôi là:

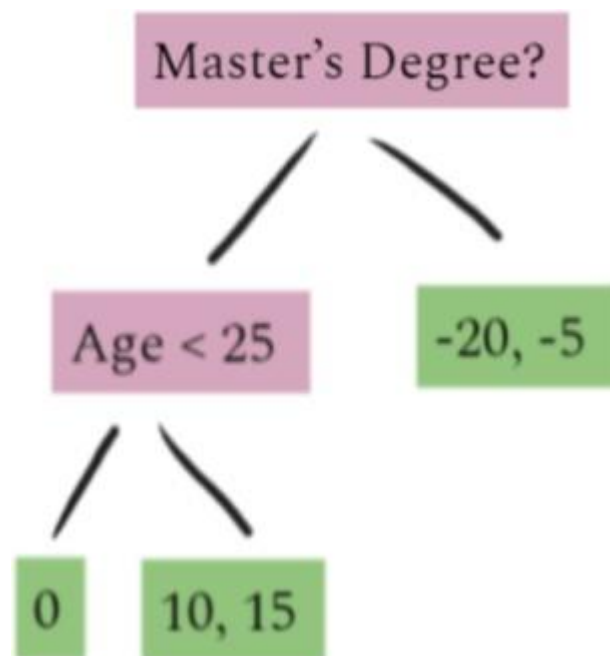


Bước 3: Tỉa cây

Cắt tỉa là một cách khác để chúng ta có thể tránh trang bị quá nhiều dữ liệu. Để làm điều này, chúng tôi bắt đầu từ dưới cùng của cây của chúng tôi và làm việc theo cách của chúng tôi để xem liệu sự phân chia có hợp lệ hay không. Để thiết lập tính hợp lệ, chúng tôi sử dụng γ (gamma). Nếu **Gain** - γ là dương thì chúng ta giữ nguyên phân tách, nếu không, chúng ta loại bỏ nó. Giá trị mặc định của γ là 0, nhưng với mục đích minh họa, hãy đặt γ của chúng ta thành 50. Từ các phép tính trước đó, chúng ta biết các giá trị **Gain** :



Vì **Gain** - γ là dương đối với tất cả các phân tách ngoại trừ Age < 24,5, chúng ta có thể loại bỏ nhánh đó. Vì vậy, cây kết quả là:



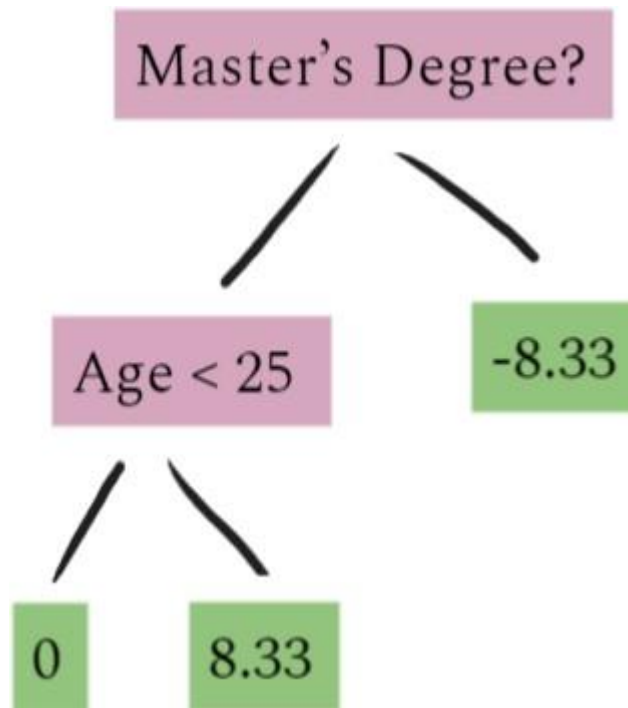
Bước 4: Tính giá trị đầu ra của lá

Chúng tôi gần như ở đó! Tất cả những gì chúng ta phải làm bây giờ là tính toán một giá trị duy nhất trong các nút lá của chúng ta bởi vì chúng ta không thể có một nút lá cung cấp cho chúng ta nhiều đầu ra.

$$\text{Output Value} = \frac{\text{Sum of Residuals}}{\text{Number of Residuals} + \lambda}$$

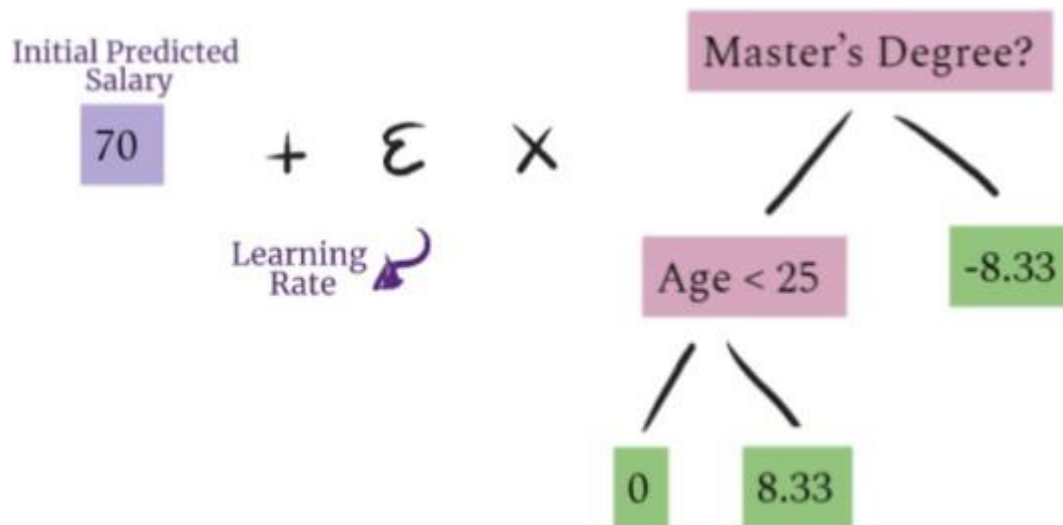
Hình ảnh của Tác giả

Điều này tương tự như công thức tính **Điểm Tương tự** ngoại trừ chúng tôi không bình phương *Phần còn lại*. Sử dụng công thức và $\lambda = 1$, *cuộn trống* cây cuối cùng của chúng ta là:



Bước 5: Đưa ra dự đoán mới

Bây giờ tất cả việc xây dựng mô hình khó khăn đó đang ở phía sau chúng ta, chúng ta đến với phần thú vị và xem các dự đoán của chúng ta cải thiện bao nhiêu khi sử dụng mô hình mới của chúng ta. Chúng tôi có thể đưa ra dự đoán bằng cách sử dụng công thức này:



Tỷ lệ học tập XGBoost là ϵ (eta) và giá trị mặc định là 0,3. Vì vậy, giá trị dự đoán của quan sát đầu tiên của chúng tôi sẽ là:

$$70 + 0.3 \times -8.33 = 67.501$$

Tương tự, chúng ta có thể tính toán phần còn lại của các giá trị dự đoán:

AGE	MASTER'S DEGREE	SALARY	Predicted Values
23	No	50	67.501
24	Yes	70	70
26	Yes	80	72.499
26	No	65	67.501
27	Yes	85	72.499

Hình ảnh của Tác giả

Bước 6: Tính toán phần dư bằng cách sử dụng các dự đoán mới

AGE	MASTER'S DEGREE	SALARY	RESIDUALS
23	No	50	-17.501
24	Yes	70	0
26	Yes	80	7.501
26	No	65	-2.501
27	Yes	85	12.501

Hình ảnh của Tác giả

Chúng tôi thấy rằng *Phần dư* mới nhỏ hơn so với trước đây, điều này cho thấy rằng chúng tôi đã thực hiện một bước nhỏ đúng hướng. Khi chúng tôi lặp lại quá trình này, *Phần dư* của chúng tôi sẽ ngày càng nhỏ hơn cho thấy rằng các giá trị dự đoán của chúng tôi đang tiến gần hơn đến các giá trị quan sát.

Bước 7: Lặp lại các bước 2–6

Bây giờ chúng ta chỉ cần lặp đi lặp lại quy trình tương tự, xây dựng một cây mới, đưa ra dự đoán và tính toán *Phần dư* ở mỗi lần lặp. Chúng tôi làm điều này cho đến khi *Phần dư* là siêu nhỏ hoặc chúng tôi đã đạt đến số lần lặp tối đa mà chúng tôi đặt cho thuật toán của mình. Nếu cây mà chúng ta đã xây dựng ở mỗi lần lặp được biểu thị bằng T_i , trong đó i là lần lặp hiện tại, thì công thức để tính toán các dự đoán là:

$$\begin{array}{c} \text{Initial Predicted} \\ \text{Salary} \\ 70 \end{array} + \epsilon \times T_1 + \epsilon \times T_2 + \epsilon \times T_3 + \dots + \epsilon \times T_i$$

Và đó là nó.