INTERNSHIP - I (ELECTRICITY PRICE PREDICTION)

Machine Learning (ML) Model Creation

Workflow

1) Importing Necessary Packages 2) Splitting Data 3) Baselines 4) Model Creation 5) Pickling 6) Model Visualization

Importing Necessary Packages

In [15]:
```
import numpy as np # linear algebra
import pandas as pd # data processing

# 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, RandomizedSearchCV
from sklearn.model_selection import train_test_split, cross_val_score, validation_curve, GridSearchCV
from sklearn.metrics import roc_curve, mean_absolute_error, mean_squared_error, accuracy_score

# Deployment
import pickle

# 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
```

Splitting Data

In [2]:
```
# Reading cleaned dataset
df=pd.read_csv("C:/Users/Arul Selvaraj/Desktop/EPP_Intern_1/energy_cleaned_dataset.csv")
```

In [3]:
```
df.fillna(method='ffill', inplace=True)  # Forward fill
```

In [4]:
```
df.isna().sum() # Checking for non-null values
```

Out[4]:
```
time                                          0
generation_biomass                            0
generation_fossil_brown_coal/lignite          0
generation_fossil_gas                         0
generation_fossil_hard_coal                   0
generation_fossil_oil                         0
generation_hydro_pumped_storage_consumption   0
generation_hydro_run_of_river_and_poundage    0
generation_hydro_water_reservoir              0
generation_nuclear                            0
generation_other                              0
generation_other_renewable                    0
generation_solar                              0
generation_waste                              0
generation_wind_onshore                       0
total_load_actual                             0
price_actual                                  0
season                                        0
dtype: int64
```

In [5]:
```
# Create Target variable
target='price_actual'

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

# Split data into train / validation sets
X_train,X_val,y_train,y_val = train_test_split(X,y,test_size=0.2,random_state=42)
```

Baselines using Evaluation Metrics

In [6]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
# 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)
```

In [7]:

```
# 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: 57.92005454545455
------------------------------------------------------------------
Baseline Mae: 11.08589652244369
------------------------------------------------------------------
Baseline RMSE: 14.21051579246629
```

Model Creation

Ordinal Encoder & Simple Imputer

In [8]:

```
# 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)
```

Regression Models

In [9]:

```
# Assigning model variables
model_lr=LinearRegression()
model_r=Ridge()
model_rfr=RandomForestRegressor()
model_xgbr=XGBRegressor()

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

# Def to check model metrics of baseline performance
def check_metrics(model):
    print(model)
    print('==================================================================')
    print('Training MAE:', mean_absolute_error(y_train,model.predict(XT_train)))
    print('------------------------------------------------------------------')
    print('Validation MAE:', mean_absolute_error(y_val,model.predict(XT_val)))
    print('------------------------------------------------------------------')
    print('Validation R2 score:', model.score(XT_val,y_val))
    print('==================================================================')
model = [model_r,model_lr,model_rfr,model_xgbr]
for m in model:
  check_metrics(m)
```

```
Ridge()
==================================================================
Training MAE: 8.310365252580741
------------------------------------------------------------------
Validation MAE: 8.38458763301881
------------------------------------------------------------------
Validation R2 score: 0.3946567309426182
==================================================================
LinearRegression()
==================================================================
Training MAE: 8.310367207097688
------------------------------------------------------------------
Validation MAE: 8.384586702906535
------------------------------------------------------------------
Validation R2 score: 0.39465689123077774
==================================================================
RandomForestRegressor()
==================================================================
Training MAE: 1.2311580855614976
------------------------------------------------------------------
Validation MAE: 3.384005004990732
------------------------------------------------------------------
Validation R2 score: 0.879435460817829
==================================================================
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
==================================================================
Training MAE: 3.1827855589324354
------------------------------------------------------------------
Validation MAE: 6.526469036531095
------------------------------------------------------------------
Validation R2 score: 0.6504817830734257
==================================================================
```

Pickling the Model for Deployment

Linear Regression

In [10]:
```
filename = 'linearregression.sav'
pickle.dump(model_lr, open(filename, 'wb'))
```

Ridge Regression

In [11]:
```
filename = 'ridgeregression.sav'
pickle.dump(model_r, open(filename, 'wb'))
```

Random Forest Regressor

In [12]:
```
filename = 'randomforest.sav'
pickle.dump(model_rfr, open(filename, 'wb'))
```

XGBoost Regressor

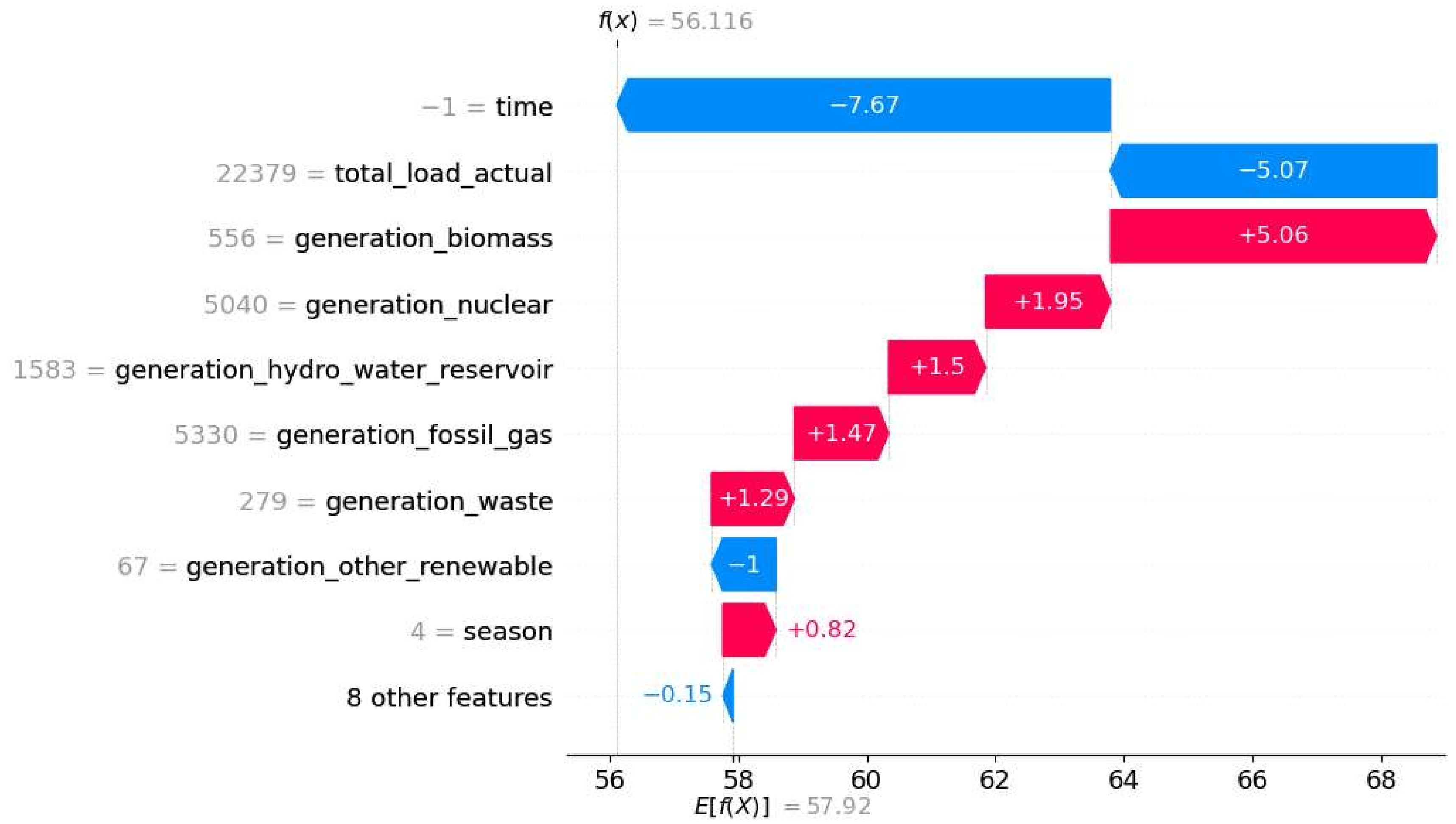
In [13]:
```
filename = 'xgboost.sav'
pickle.dump(model_xgbr, open(filename, 'wb'))
```

Model Visualization

In [21]:
```
# Set samp variable to show features when plotting
samp = pd.DataFrame(XT_val,columns=ordinal_fit.get_feature_names())
```

```
`get_feature_names` is deprecated in all of sklearn. Use `get_feature_names_out` instead.
```

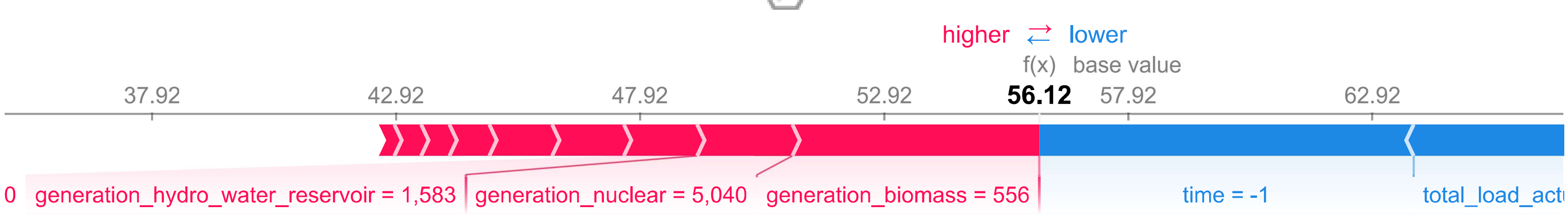
In [20]:
```
# Shap waterfall plot showing feature importance
explainer = shap.TreeExplainer(model_xgbr)
shap_values=explainer(samp.head(1))
shap.plots.waterfall(shap_values[0])
```



In [22]:
```python
# 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))
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

Out[22]:



In [23]:
```python
# 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())
```

Out[23]:

Weight	Feature
0.3428 ± 0.0126	season
0.1533 ± 0.0060	generation_other_renewable
0.1365 ± 0.0157	generation_fossil_gas
0.1314 ± 0.0015	generation_nuclear
0.1015 ± 0.0082	generation_fossil_hard_coal
0.0917 ± 0.0075	total_load_actual
0.0809 ± 0.0046	generation_hydro_run_of_river_and_poundage
0.0669 ± 0.0063	generation_waste
0.0615 ± 0.0016	generation_biomass
0.0494 ± 0.0054	generation_other
0.0352 ± 0.0020	generation_hydro_water_reservoir
0.0343 ± 0.0038	generation_fossil_brown_coal/lignite
0.0332 ± 0.0048	generation_fossil_oil
0.0150 ± 0.0038	generation_wind_onshore
0.0013 ± 0.0042	generation_solar
0 ± 0.0000	time
-0.0070 ± 0.0030	generation_hydro_pumped_storage_consumption