Machine Learning: Optimizing Model Prediction

November 4, 2023

In this project, I will use a dataset from the UCI Machine Learning Repository (https://archive.ics.uci.edu/dataset/162/forest+firesabout) about Forest Fires to predict the extent of fire damage to a forest. I will use a reference model which will be a standard linear regression model and then iterate on it using techniques to optimize it.

These are the features in the dataset: - `X` - x-axis spatial coordinate within the Montesinho park map: 1 to 9 - `Y` - y-axis spatial coordinate within the Montesinho park map: 2 to 9 - `month` - month of the year: 'jan' to 'dec' - `day` - day of the week: 'mon' to 'sun' - `FFMC` - FFMC index from the FWI system: 18.7 to 96.20 - `DMC` - DMC index from the FWI system: 1.1 to 291.3 - `DC` - DC index from the FWI system: 7.9 to 860.6 - `ISI` - ISI index from the FWI system: 0.0 to 56.10 - `temp` - temperature in Celsius degrees: 2.2 to 33.30 - `RH` - relative humidity in %: 15.0 to 100 - `wind` - wind speed in km/h: 0.40 to 9.40 - `rain` - outside rain in mm/m2 : 0.0 to 6.4 - `area` - the burned area of the forest (in ha): 0.00 to 1090.84 (this output variable is very skewed towards 0.0, thus it may make sense to model with the logarithm transform).

1 Exploring the Data and Creating Reference Model

Now, I will upload the explore the data and check for missing values. Then, I will create the reference model with only the `temp` and `wind` features.

```
import pandas as pd
import numpy as np
```

[2]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

fires = pd.read_csv('fires.csv')
```

[3]:
```
fires.head()
```

[4]:
```

```

[4]:
```
   Unnamed: 0  X  Y month  day  FFMC   DMC     DC  ISI  temp    RH  wind  \
0           1  7  5   mar  fri  86.2  26.2   94.3  5.1   NaN  51.0   6.7
1           2  7  4   oct  tue  90.6   NaN  669.1  6.7  18.0  33.0   0.9
2           3  7  4   oct  sat  90.6  43.7    NaN  6.7  14.6  33.0   1.3
3           4  8  6   mar  fri  91.7  33.3   77.5  9.0   8.3  97.0   4.0
4           5  8  6   mar  sun  89.3  51.3  102.2  9.6  11.4  99.0   NaN
```

```
   rain  area
0   0.0   0.0
1   0.0   0.0
2   0.0   0.0
3   0.2   0.0
4   0.0   0.0
```

Looking at a preview of the dataframe, I can see that there are some missing values in the dataset.

[5]:
```python
fires.describe()
```

[5]:
```
       Unnamed: 0           X           Y        FFMC         DMC          DC  \
count  517.000000  517.000000  517.000000  469.000000  496.000000  474.000000
mean   259.000000    4.669246    4.299807   90.580384  111.195363  550.673418
std    149.389312    2.313778    1.229900    5.698137   64.008450  246.061309
min      1.000000    1.000000    2.000000   18.700000    1.100000    7.900000
25%    130.000000    3.000000    4.000000   90.200000   70.800000  441.200000
50%    259.000000    4.000000    4.000000   91.600000  108.300000  664.500000
75%    388.000000    7.000000    5.000000   92.800000  141.575000  713.900000
max    517.000000    9.000000    9.000000   96.200000  291.300000  860.600000

              ISI        temp          RH        wind        rain         area
count  515.000000  496.000000  487.000000  482.000000  485.000000   517.000000
mean     9.018835   18.884677   44.381930    4.021784    0.023093    12.847292
std      4.564890    5.748318   16.180372    1.794460    0.305532    63.655818
min      0.000000    2.200000   15.000000    0.400000    0.000000     0.000000
25%      6.450000   15.475000   33.000000    2.700000    0.000000     0.000000
50%      8.400000   19.300000   42.000000    4.000000    0.000000     0.520000
75%     10.750000   22.725000   53.500000    4.900000    0.000000     6.570000
max     56.100000   33.300000  100.000000    9.400000    6.400000  1090.840000
```

In the dataset, I can see that there are 517 instances, and each instance represents a different observation related to forest fires. The 'X' and 'Y' columns indicate the spatial coordinates within the Montesinho park map, ranging from 1 to 9. The FFMC (Fine Fuel Moisture Code) values range from 18.7 to 96.2, with a mean of 90.58. The temperature ('temp') varies from 2.2 to 33.3 degrees Celsius, with a mean of 18.88. The 'area' variable, representing the burned area of the forest in hectares, has a skewed distribution, with the minimum being 0.0 and the maximum reaching 1090.84. Dealing with missing values is essential, especially in columns like 'temp' and 'wind,' and I'll need to consider strategies such as imputation.

[6]:
```python
fires.dtypes
```

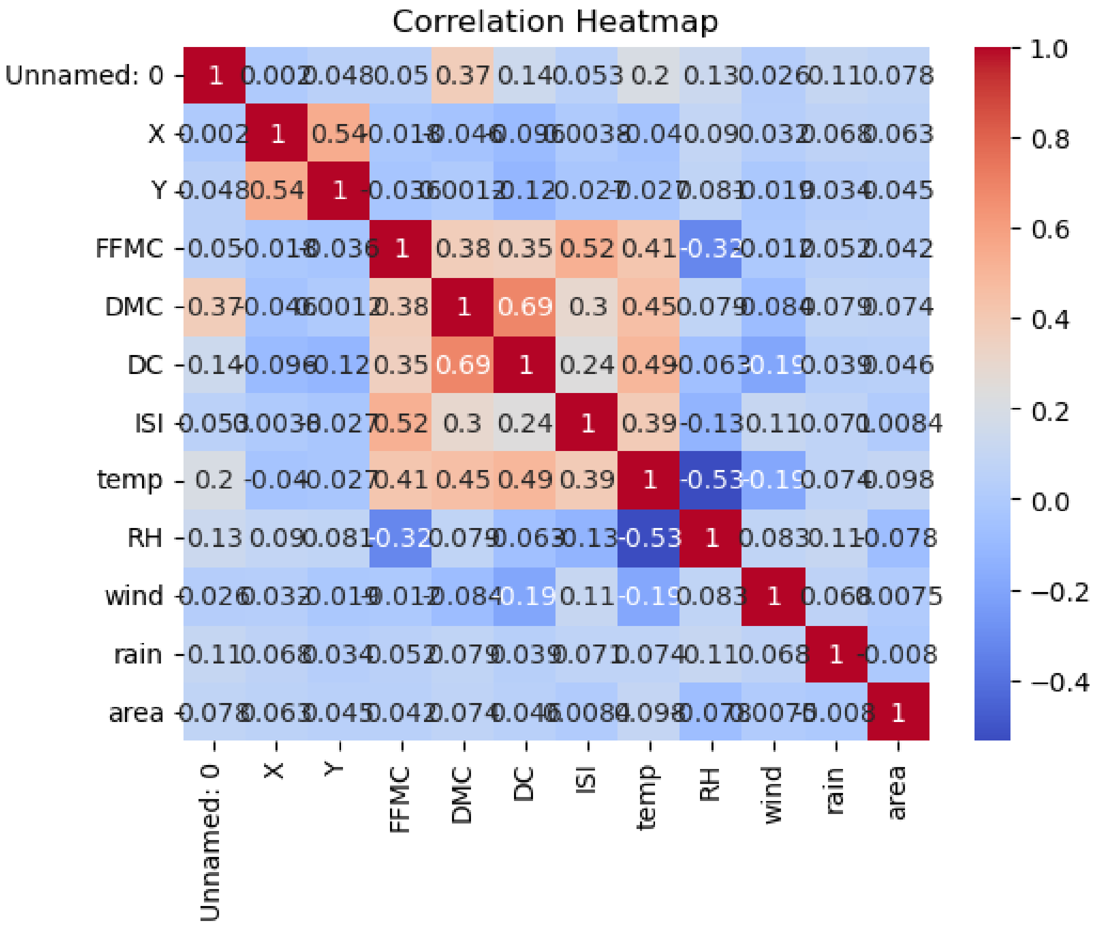
[6]:
```
Unnamed: 0      int64
X               int64
Y               int64
month          object
day            object
```

```
FFMC          float64
DMC           float64
DC            float64
ISI           float64
temp          float64
RH            float64
wind          float64
rain          float64
area          float64
dtype: object
```

[7]:
```
fires.isnull().sum()
```

[7]:
```
Unnamed: 0     0
X              0
Y              0
month          0
day            0
FFMC          48
DMC           21
DC            43
ISI            2
temp          21
RH            30
wind          35
rain          32
area           0
dtype: int64
```

[8]:
```
import matplotlib.pyplot as plt
import seaborn as sns
correlation_matrix = fires.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```




```
[9]: fires_reference = fires[["wind", "temp", "area"]].dropna()
     reference_X = fires[["wind", "temp"]]

     reference = LinearRegression()
```

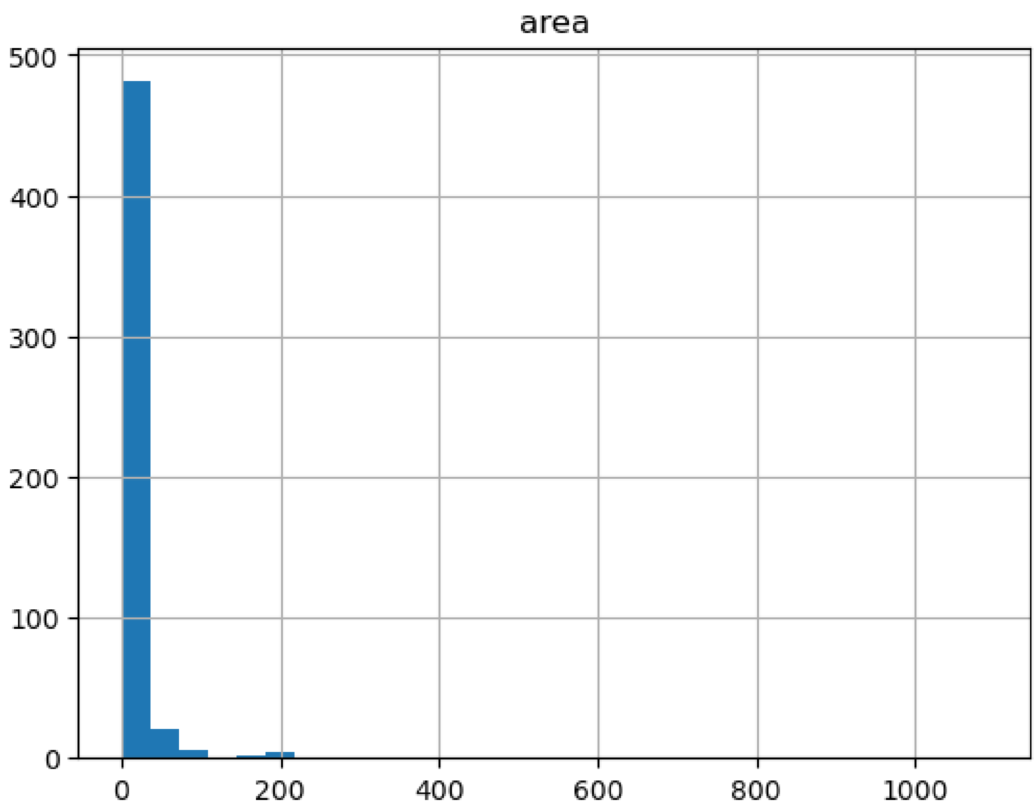
2 Data Processing

First, I will convert the `month` column into a categorical feature. Instead of using the strings, I will convert it into an indicator for the summer months in the northern hemisphere.

For the sake of completion, I will impute all of the features so that I can have the biggest set to choose from for sequential feature selection. I will use K-nearest neighbors imputation since I expect area damage to be similar among similar fires.

```
[10]: fires.hist("area", bins=30)
```

```
[10]: array([[<AxesSubplot:title={'center':'area'}>]], dtype=object)
```

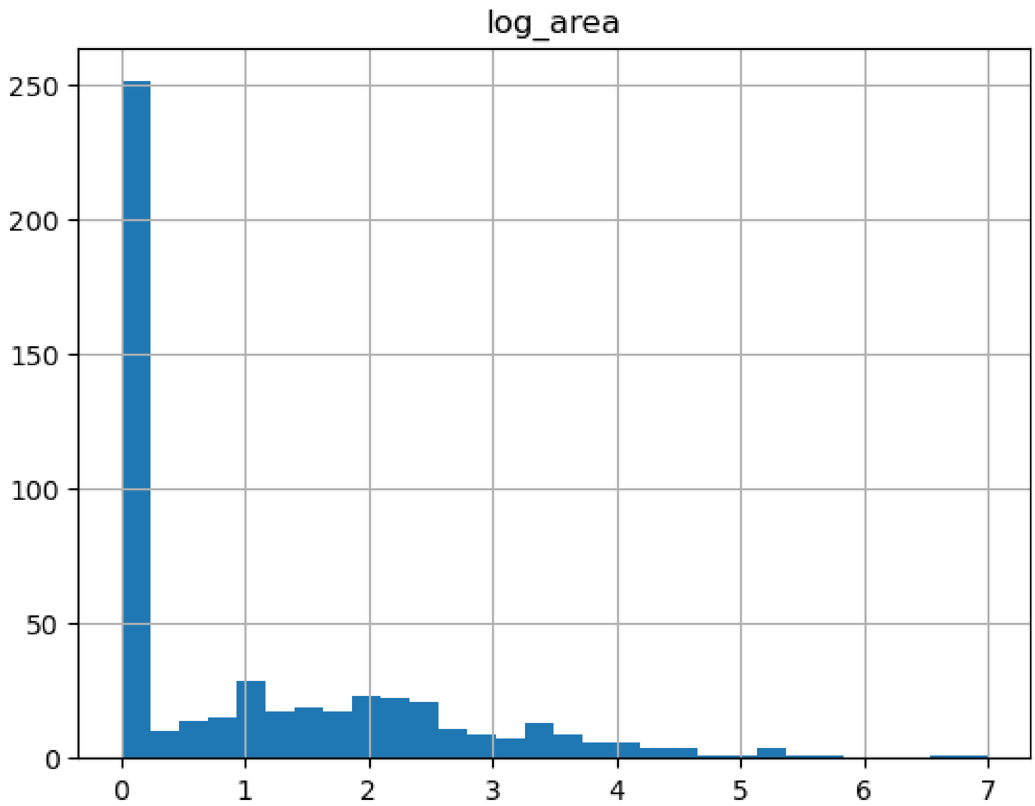


The outcome is highly right-skewed with extremely damaging fires. Furthermore, many of the rows have outcome values that are zero or near-zero. It might be worth it to log-transform the data. Some of the outcomes are actually 0, so I will add 1 to prevent any errors.

[11]:
```
fires["log_area"] = np.log(fires["area"] + 1)

fires.hist("log_area", bins=30)
```

[11]:
```
array([[<AxesSubplot:title={'center':'log_area'}>]], dtype=object)
```



Performing a log-transformation doesn't produce a bell-shaped distribution, but it does spread out the data a bit more than without the transformation. It's probably the case that most fires do not appreciably damage the forest, so I would be mistaken in removing all of these rows.

Instead of using month directly, I will derive another feature called `summer` that takes a value of 1 when the fire occurred during the summer. The idea here is that summer months are typically hotter, so fires are more likely.

[12]:
```
def is_summer_month(month):
    if month in ["jun", "jul", "aug"]:
        return 1
    else:
        return 0

fires["summer"] = [is_summer_month(m) for m in fires["month"]]
```

[13]:
```
from sklearn.impute import KNNImputer

imp = KNNImputer(missing_values = np.nan, n_neighbors=3)

fires_missing = fires[fires.columns[5:13]] # FFMC to rain
imputed = pd.DataFrame(imp.fit_transform(fires_missing),
```

```
                          columns = fires.columns[5:13])
imputed
```

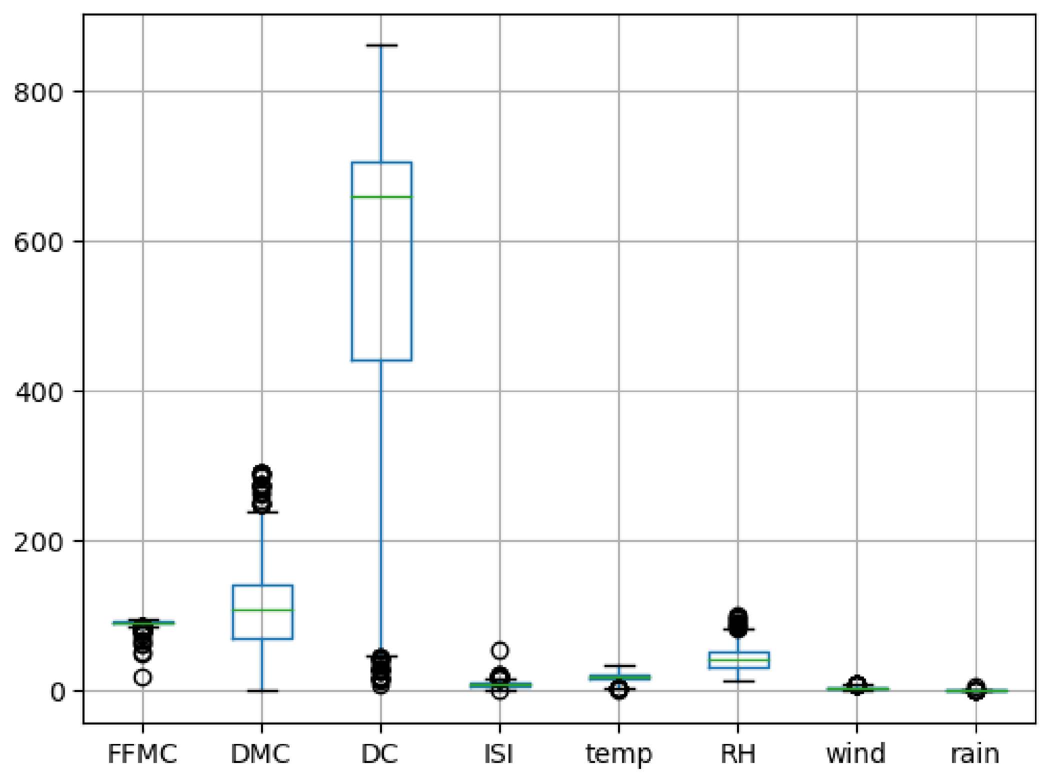
[13]:
```
     FFMC         DMC          DC   ISI  temp    RH      wind  rain
0    86.2   26.200000   94.300000   5.1  16.6  51.0  6.700000   0.0
1    90.6   56.433333  669.100000   6.7  18.0  33.0  0.900000   0.0
2    90.6   43.700000  470.833333   6.7  14.6  33.0  1.300000   0.0
3    91.7   33.300000   77.500000   9.0   8.3  97.0  4.000000   0.2
4    89.3   51.300000  102.200000   9.6  11.4  99.0  4.333333   0.0
..    …          …           …    …    …     …       …     …
512  81.6   56.700000  665.600000   1.9  27.8  32.0  2.700000   0.0
513  81.6   56.700000  665.600000   1.9  21.9  71.0  5.800000   0.0
514  81.6   56.700000  665.600000   1.9  21.2  70.0  6.700000   0.0
515  94.4  146.000000  614.700000  11.3  25.6  42.0  4.000000   0.0
516  79.5    3.000000  106.700000   1.1  11.8  31.0  4.500000   0.0

[517 rows x 8 columns]
```

Now, I will examine the data for outliers using boxplots:

[14]:
```
imputed.boxplot(column=["FFMC", "DMC", "DC", "ISI", "temp", "RH", "wind",␣
  ↪"rain"])
```

[14]: <AxesSubplot:>



The dots indicate that there are some outliers in the data. I will examine the number of outliers in each of the columns.

[15]:
```
for col in imputed:

    quartiles = np.percentile(fires[col], [25, 50, 75])
    iqr = quartiles[2] - quartiles[0]
    lower_bound = quartiles[0] - (1.5 * iqr)
    upper_bound = quartiles[2] + (1.5 * iqr)
    num_outliers =sum((imputed[col] < lower_bound) | (imputed[col] >␣
 ↪upper_bound))

    print(f"The {col} column has {num_outliers} according to the boxplot method.
 ↪")
```

```
The FFMC column has 0 according to the boxplot method.
The DMC column has 0 according to the boxplot method.
The DC column has 0 according to the boxplot method.
The ISI column has 0 according to the boxplot method.
The temp column has 0 according to the boxplot method.
The RH column has 0 according to the boxplot method.
The wind column has 0 according to the boxplot method.
The rain column has 0 according to the boxplot method.
```

Despite the visual cue in the boxplots, based on the actual calculations, there don't seem to be any outliers. In this case, I will leave the dataset as-is.

Now that the dataset has been inspected for missing values and outliers, I will proceed to standardize it. These standardized values will help for standardization. Afterwards, I will append the `summmer` feature back into the dataset.

[16]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled = scaler.fit_transform(imputed)
scaled = pd.DataFrame(scaled, columns = fires.columns[5:13])

final = pd.concat([fires["summer"], scaled], axis=1)

final
```

[16]:
```
     summer      FFMC       DMC        DC       ISI      temp        RH  \
0         0 -0.812283 -1.335942 -1.846711 -0.860187 -0.398187  0.418726
1         0 -0.010735 -0.859009  0.509582 -0.508736 -0.155493 -0.715565
2         0 -0.010735 -1.059878 -0.303178 -0.508736 -0.744894 -0.715565
3         0  0.189652 -1.223939 -1.915580 -0.003526 -1.837021  3.317471
```

```
4         0 -0.247556 -0.939988 -1.814327  0.128267 -1.299625  3.443503
..       …         …         …         …         …         …         …
512       1 -1.650265 -0.854803  0.495235 -1.563087  1.543370 -0.778581
513       1 -1.650265 -0.854803  0.495235 -1.563087  0.520585  1.679050
514       1 -1.650265 -0.854803  0.495235 -1.563087  0.399238  1.616034
515       1  0.681511  0.553912  0.286579  0.501683  1.161993 -0.148419
516       0 -2.032821 -1.701924 -1.795880 -1.738812 -1.230284 -0.841597

         wind      rain
0    1.514159 -0.073268
1   -1.761003 -0.073268
2   -1.535130 -0.073268
3   -0.010485  0.603155
4    0.177742 -0.073268
..        …         …
512 -0.744573 -0.073268
513  1.005944 -0.073268
514  1.514159 -0.073268
515 -0.010485 -0.073268
516  0.271856 -0.073268

[517 rows x 9 columns]
```

3 Subset Selection

[17]:
```python
from sklearn.feature_selection import SequentialFeatureSelector

y = fires["log_area"]

sfs_model = LinearRegression()
sfs_model2 = LinearRegression()
sfs_model3 = LinearRegression()

forward2 = SequentialFeatureSelector(estimator=sfs_model,
                                     n_features_to_select=2,
                                     direction="forward")

forward4 = SequentialFeatureSelector(estimator=sfs_model2,
                                     n_features_to_select=4,
                                     direction="forward")

forward6 = SequentialFeatureSelector(estimator=sfs_model3,
                                       n_features_to_select=6,
                                       direction="forward")

forward2.fit(final, y)
```

```
forward4.fit(final, y)
forward6.fit(final, y)

print("Features selected in 2 feature model:", forward2.get_feature_names_out())
print("Features selected in 4 feature model:", forward4.get_feature_names_out())
print("Features selected in 6 feature model:", forward6.get_feature_names_out())
```

```
Features selected in 2 feature model: ['FFMC' 'DC']
Features selected in 4 feature model: ['FFMC' 'DC' 'RH' 'wind']
Features selected in 6 feature model: ['summer' 'FFMC' 'DC' 'ISI' 'RH' 'wind']
```

[18]:
```
backward2 = SequentialFeatureSelector(estimator=sfs_model,
                                     n_features_to_select=2,
                                     direction="backward")

backward4 = SequentialFeatureSelector(estimator=sfs_model,
                                     n_features_to_select=4,
                                     direction="backward")

backward6 = SequentialFeatureSelector(estimator=sfs_model,
                                       n_features_to_select=6,
                                       direction="backward")

backward2.fit(final, y)
backward4.fit(final, y)
backward6.fit(final, y)

print("Features selected in 2 feature model:", backward2.
 ↪get_feature_names_out())
print("Features selected in 4 feature model:", backward4.
 ↪get_feature_names_out())
print("Features selected in 6 feature model:", backward6.
 ↪get_feature_names_out())
```

```
Features selected in 2 feature model: ['DC' 'wind']
Features selected in 4 feature model: ['FFMC' 'DC' 'RH' 'wind']
Features selected in 6 feature model: ['summer' 'FFMC' 'DC' 'ISI' 'RH' 'wind']
```

Based on the features chosen by forward and backward selection, it seems like `DC`, `wind` and `FFMC` seem to be the most impactful on predicting `log_area`.

[19]:
```
fw2_model = LinearRegression() # .fit(final[forward2.get_feature_names_out()],␣
 ↪y)
fw4_model = LinearRegression() # .fit(final[forward4.get_feature_names_out()],␣
 ↪y)
fw6_model = LinearRegression() # .fit(final[forward6.get_feature_names_out()],␣
 ↪y)
```

```
bw2_model = LinearRegression() # .fit(final[backward2.get_feature_names_out()],␣
 ↪y)
bw4_model = LinearRegression() # .fit(final[backward4.get_feature_names_out()],␣
 ↪y)
bw6_model = LinearRegression() # .fit(final[backward6.get_feature_names_out()],␣
 ↪y)
```

4 More Candidate Models

Another approach I might consider taking is using regularized versions of linear regression. Fires have many factors that can increase the damage they have, so it seems unhelpful to restrict the model to a univariate, non-linear model.

[21]:
```
from sklearn.linear_model import LassoCV, RidgeCV

ridge = RidgeCV(alphas = np.linspace(1, 10000, num=1000))
lasso = LassoCV(alphas = np.linspace(1, 10000, num=1000))

ridge.fit(final, y)
lasso.fit(final, y)

print("Ridge tuning parameter: ", ridge.alpha_)
print("LASSO tuning parameter: ", lasso.alpha_)

print("Ridge coefficients: ", ridge.coef_)
print("LASSO coefficients: ", lasso.coef_)
```

```
Ridge tuning parameter:  1372.2342342342342
LASSO tuning parameter:  10000.0
Ridge coefficients:  [-0.01455017  0.01311215  0.02006457  0.02004741
-0.01073465  0.01297049
 -0.01489714  0.02670554  0.00816103]
LASSO coefficients:  [-0.  0.  0.  0. -0.  0. -0.  0.  0.]
```

The LASSO tuning parameter always seems to be on the extreme. Given that the outcome has many small values, it suggests that having no features at all is better than having any. I will try to home in on a better tuning parameter value below by choosing a smaller range to pick from.

[22]:
```
ridge = RidgeCV(alphas = np.linspace(1000, 1500, num=1000))
ridge.fit(final, y)
print("Ridge tuning parameter: ", ridge.alpha_)
```

```
Ridge tuning parameter:  1371.3713713713714
```

I will use this value in k-fold cross-validation, rounded to the hundredths place. I will use a ridge regression and choose not to use a LASSO model here since the regularization results aren't helpful.

5 K-Fold Cross Validation

[23]:
```
from sklearn.model_selection import cross_val_score

reference_cv = cross_val_score(reference, final[["wind", "temp"]], y, cv = 5,␣
↪scoring = "neg_mean_squared_error")
fw2_cv = cross_val_score(fw2_model, final[forward2.get_feature_names_out()], y,␣
↪cv = 5, scoring = "neg_mean_squared_error")
fw4_cv = cross_val_score(fw4_model, final[forward4.get_feature_names_out()], y,␣
↪cv = 5, scoring = "neg_mean_squared_error")
fw6_cv = cross_val_score(fw6_model, final[forward6.get_feature_names_out()], y,␣
↪cv = 5, scoring = "neg_mean_squared_error")
bw2_cv = cross_val_score(bw2_model, final[backward2.get_feature_names_out()],␣
↪y, cv = 5, scoring = "neg_mean_squared_error")
bw4_cv = cross_val_score(bw4_model, final[backward4.get_feature_names_out()],␣
↪y, cv = 5, scoring = "neg_mean_squared_error")
bw6_cv = cross_val_score(bw6_model, final[backward6.get_feature_names_out()],␣
↪y, cv = 5, scoring = "neg_mean_squared_error")
ridge_cv = cross_val_score(ridge, final, y, cv = 5, scoring =␣
↪"neg_mean_squared_error")
```

[24]:
```
print("Reference Model, Avg Test MSE: ", np.mean(reference_cv), " SD: ", np.
↪std(reference_cv))
print("Forward-2 Model, Avg Test MSE: ", np.mean(fw2_cv), " SD: ", np.
↪std(fw2_cv))
print("Forward-4 Model, Avg Test MSE: ", np.mean(fw4_cv), " SD: ", np.
↪std(fw4_cv))
print("Forward-6 Model, Avg Test MSE: ", np.mean(fw6_cv), " SD: ", np.
↪std(fw6_cv))
print("Backward-2 Model, Avg Test MSE: ", np.mean(bw2_cv), " SD: ", np.
↪std(bw2_cv))
print("Backward-4 Model, Avg Test MSE: ", np.mean(bw4_cv), " SD: ", np.
↪std(bw4_cv))
print("Backward-6 Model, Avg Test MSE: ", np.mean(bw6_cv), " SD: ", np.
↪std(bw6_cv))
print("Ridge Model, Avg Test MSE: ", np.mean(bw6_cv), " SD: ", np.std(bw6_cv))
```

```
Reference Model, Avg Test MSE:  -2.204650013004116  SD:  1.060040355378637
Forward-2 Model, Avg Test MSE:  -2.1735431721198535  SD:  1.0208083278697586
Forward-4 Model, Avg Test MSE:  -2.193528106772711  SD:  1.0004774710977677
Forward-6 Model, Avg Test MSE:  -2.2397225539348753  SD:  1.0123323877770343
Backward-2 Model, Avg Test MSE:  -2.173357302739327  SD:  1.0038109503795958
Backward-4 Model, Avg Test MSE:  -2.193528106772711  SD:  1.0004774710977677
Backward-6 Model, Avg Test MSE:  -2.2397225539348753  SD:  1.0123323877770343
Ridge Model, Avg Test MSE:  -2.2397225539348753  SD:  1.0123323877770343
```

6 Conclusion

In this project, I started by exploring a dataset from the UCI Machine Learning Repository containing information about forest fires. The dataset included various spatial, weather, and fire-related features, with the target variable being the burned area of the forest.

Data Exploration and Preprocessing: - Explored the dataset's structure and identified missing values. - Imputed missing values using the KNN imputation method to retain data integrity. - Transformed the target variable by taking the logarithm to handle skewness. - Created a binary variable to capture summer months, potentially influencing forest fires.

Feature Engineering: - Utilized Sequential Feature Selector to identify optimal feature subsets for different models. - Engineered features to represent summer months, enhancing the model's ability to capture seasonal patterns.

Modeling: - Trained linear regression models using different feature sets derived from feature selection methods (forward and backward). - Incorporated regularization techniques (Ridge) to handle potential multicollinearity and overfitting. - Employed cross-validation to assess model performance and avoid overfitting.

Findings: - Multiple models (reference, feature selection, Ridge) yielded similar performance. - Feature selection techniques did not significantly outperform the reference model. - Ridge regularization did not substantially improve model performance.

Recommendations/ Next Steps: - Consider exploring more advanced algorithms or ensemble methods. - Investigate additional feature engineering possibilities to capture nuanced patterns in forest fire occurrence. - Evaluate the impact of other external factors that may influence forest fires but were not included in the current dataset.