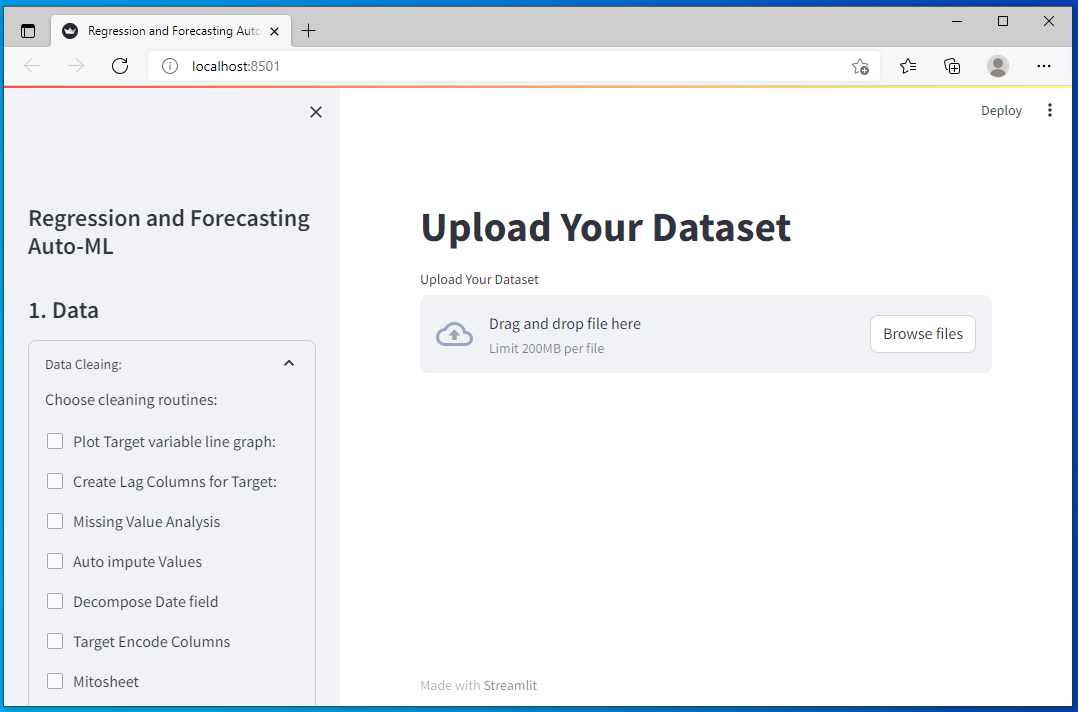
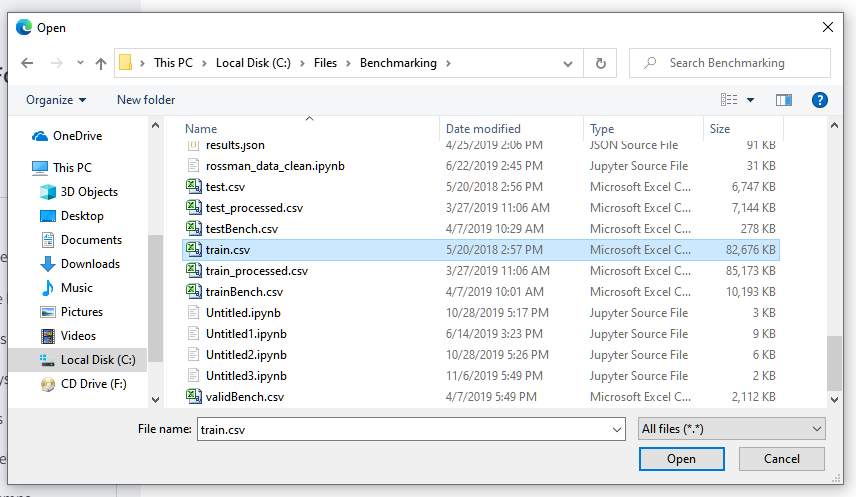
**Regression Auto-ML and Forecasting**

Regression problems are of key importance to business. From forecasting sales, materials consumption and even improving any kind of KPI, all start with the recording of some type of time series. With the intent of speeding up the analysis and forecasting of some key metrics that re-occur frequently in the field of supply chain I created this Auto-ML framework.

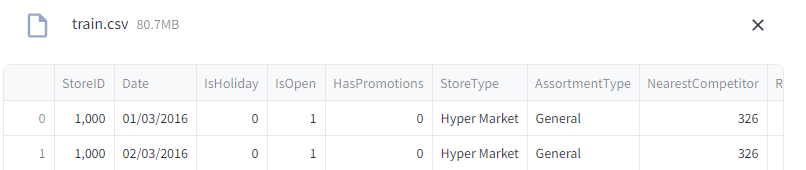
To demonstrate the capabilities of the framework I am going to use some real world sales data used in a competition held while I was doing my post-graduation in Artificial Intelligence at USI in Switzerland. The data set is composed of sales for 145 stores of a retail company in Italy.

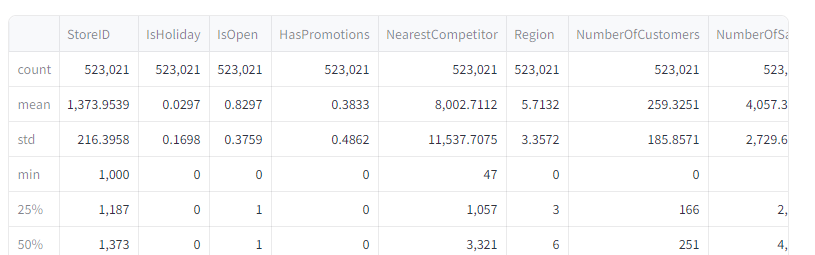
The process begins by uploading the dataset by clicking on the “Browse Files” button:

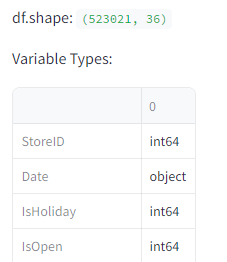




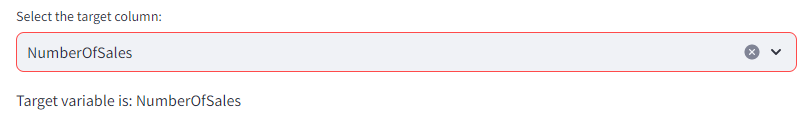
The dashboard then displays a sample of the dataset and some summary statistics



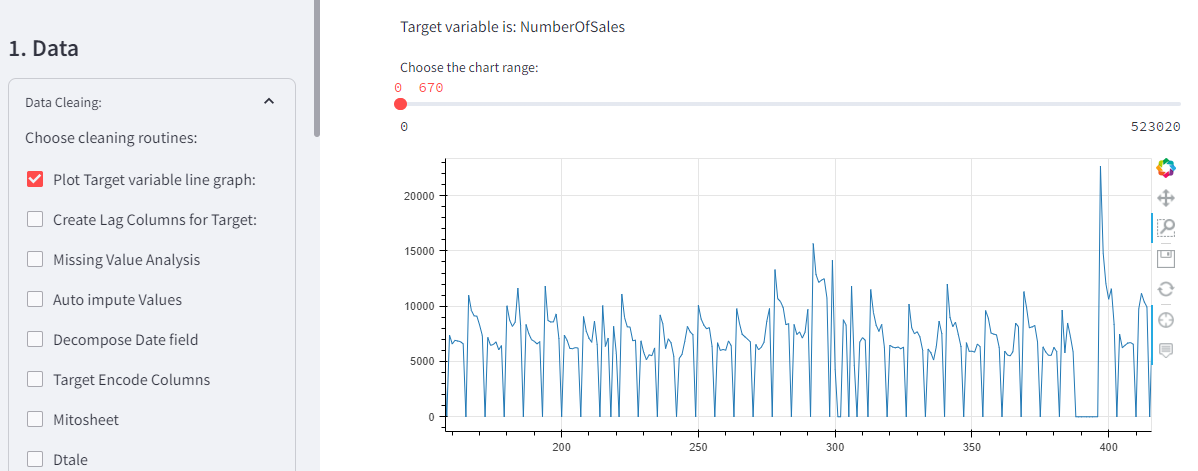


Then it asks you to select the target variable (the variable you want to forecast or your ‘Y’). All correlations and feature selection will be made with respect to this variable:

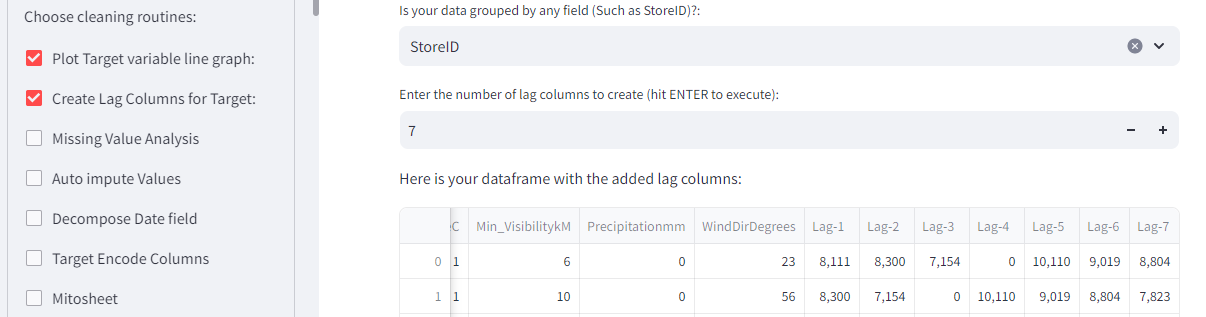


You can then choose to plot a quick line graph of the data to see if there is any trend or repeating pattern:

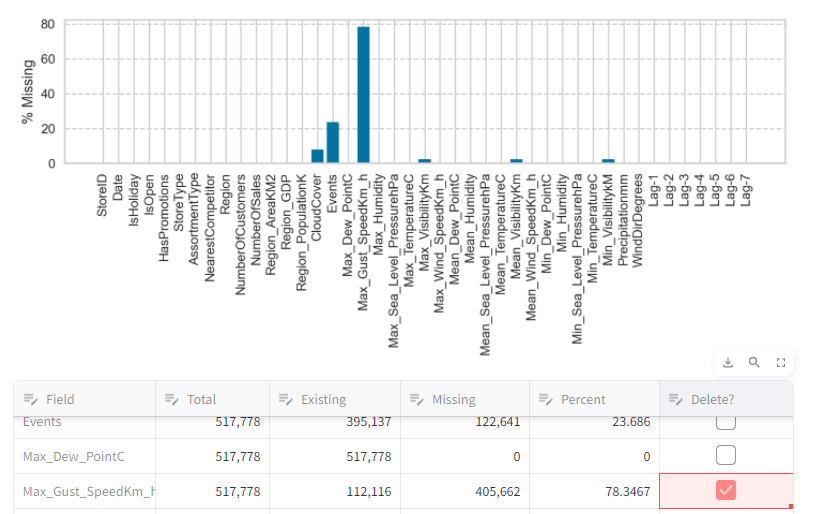


You can see that the data is cyclical going to 0 every Sunday.

Next you can create lag columns automatically, which usually are a useful feature in forecasting regression problems:

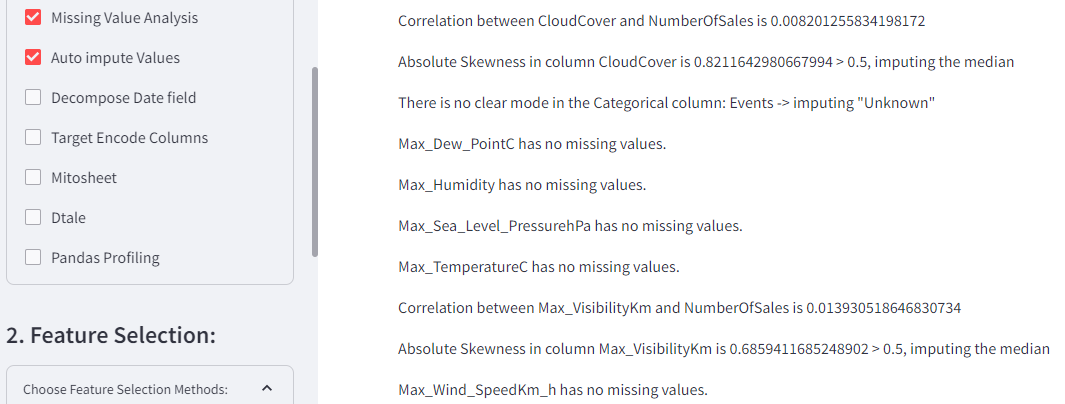


You can then perform missing value analysis. The dashboard gives you a bar chart of the % of rows missing and an opportunity to exclude any columns that you would like in case they have a high percentage missing:



We are going to exclude Max\_Gust\_speed because it has over 70% missing and the number of customers because it is something you would also have to forecast since these numbers are not available in the test set. To confirm the deletion you must confirm it by clicking on the “Delete marked columns” button.

In the next section you have an opportunity to ask the dashboard to automatically impute the missing values for the remaining variables. The system uses a series of statistical tests to determine which value should be imputed:



Here you can see the code o how this is determined:

AutoImpute = DataClean.checkbox("Auto impute Values")

if AutoImpute:

# Define Auto Impute function:

def impute\_column(df, col, target):

# Check if the column has any missing values

if df[col].isnull().any():

# Get the percentage of missing values in the column

percent = df[col].isnull().mean() \* 100

# If the percentage is more than 50, drop the column and return the dataframe

if percent > 50:

df.drop(col, axis=1, inplace=True)

return pd.DataFrame(data=None)

# Otherwise, proceed with the imputation process

else:

# Get the data type of the column

dtype = df[col].dtype

# If the column is numeric

if dtype in ["int", "float"]:

# Check if there is a strong correlation between the column and the target variable

corr = df[[col, target]].corr().iloc[0, 1]

st.write('Correlation between '+col+' and '+target+' is '+ str(corr))

# If the correlation coefficient is above 0.5 or below -0.5, use random forest imputation

if abs(corr) > 0.5:

# Import the RandomForestRegressor or RandomForestClassifier class from scikit-learn

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

imp\_model = RandomForestRegressor()

st.write("Correlation is greater than 0.5 -> Using Random Forest regressor")

# Fit the model on the rows that have no missing values in the column, using all other columns as predictors

imp\_model.fit(df.dropna()[df.columns.drop(col)], df.dropna()[col])

# Predict the missing values in the column using the fitted model and the other columns as inputs

df.loc[df[col].isnull(), col] = imp\_model.predict(df[df[col].isnull()][df.columns.drop(col)])

else:

# Get the distribution of the column

skewness = df[col].skew()

# If the column is normally distributed, use mean imputation

if abs(skewness) < 0.5:

st.write('Absolute Skewness in column '+col+' is '+ str(abs(skewness)) + ' < 0.5, imputing the mean')

imp\_mean = SimpleImputer(missing\_values=np.nan, strategy="mean")

df[col] = imp\_mean.fit\_transform(df[[col]])

# If the column is skewed, use median imputation

else:

st.write('Absolute Skewness in column '+col+' is ' + str(abs(skewness)) + ' > 0.5, imputing the median')

imp\_median = SimpleImputer(missing\_values=np.nan, strategy="median")

df[col] = imp\_median.fit\_transform(df[[col]])

# If the column is categorical

elif dtype == "object" or dtype == "category":

# Get the frequency of each category

mode = df[col].mode()[0]

count = df[col].value\_counts()[mode]

# If there is a clear mode, use mode imputation

if count > df.shape[0] \* 0.5:

st.write("There is a clear mode in column: '+col+' -> Imputing the most frequent")

imp\_mode = SimpleImputer(missing\_values=np.nan, strategy="most\_frequent")

df[col] = imp\_mode.fit\_transform(df[[col]])

# If there is no clear mode, use constant value imputation

else:

st.write('There is no clear mode in the Categorical column: '+col+' -> imputing "Unknown"')

imp\_constant = SimpleImputer(missing\_values=np.nan, strategy="constant", fill\_value="Unknown")

df[col] = imp\_constant.fit\_transform(df[[col]])

# If the column is neither numeric nor categorical, raise an error

else:

raise ValueError(f"Unsupported data type: {dtype}")

else:

st.write(col + " has no missing values.")

return df[col]

# Loop over all columns of df and impute values

for col in df.columns.drop(st.session\_state['TargetColStr']):

# Call the impute\_column function with each column name

df\_column = impute\_column(df, col, st.session\_state["TargetColStr"])

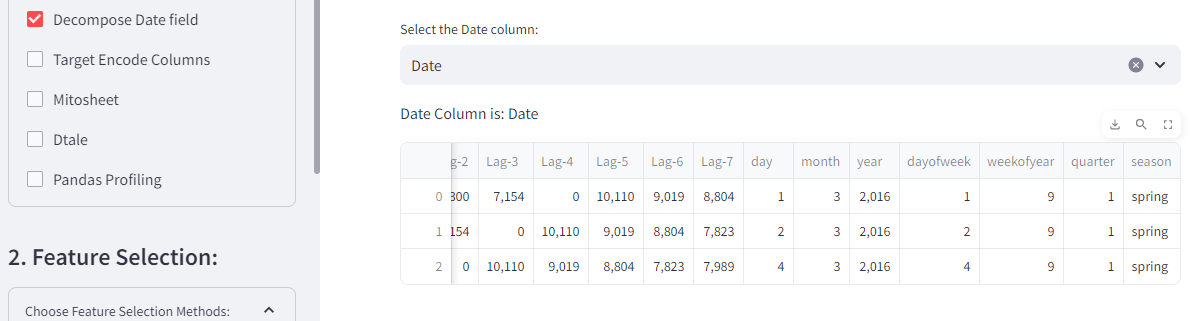
if not df\_column.empty:

df[col] = df\_column

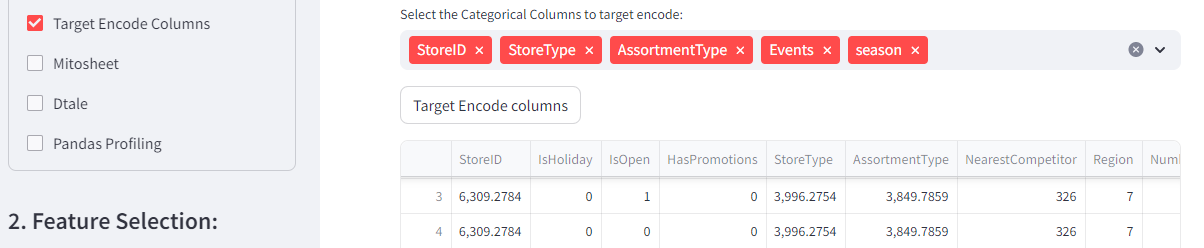
else:

st.write(col+ " has over 50% missing velues column DROPPED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<-----")

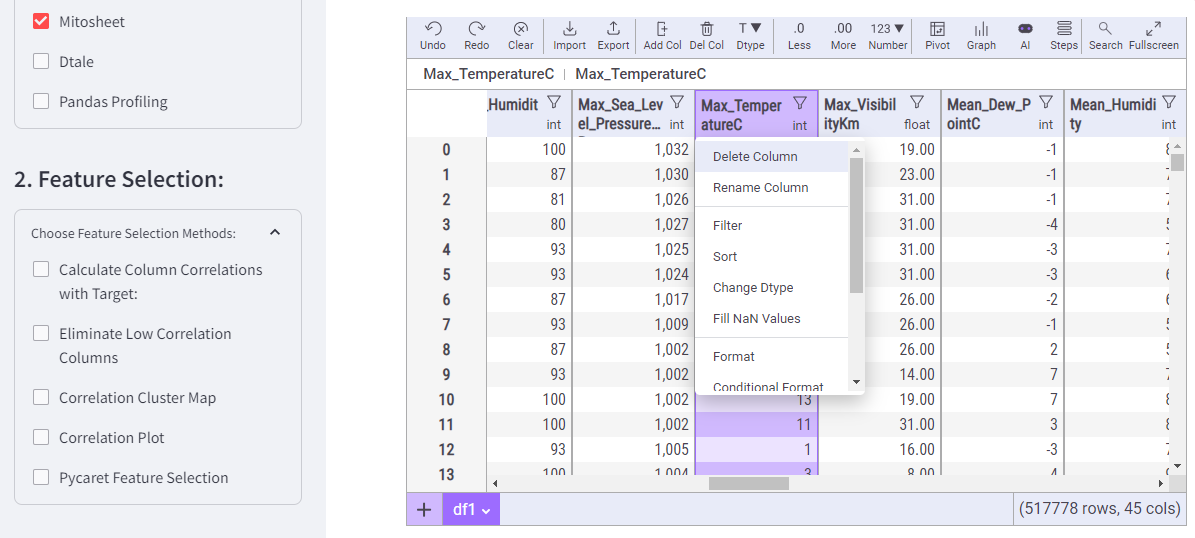
Next, you have the opportunity of decomposing the date into its components:



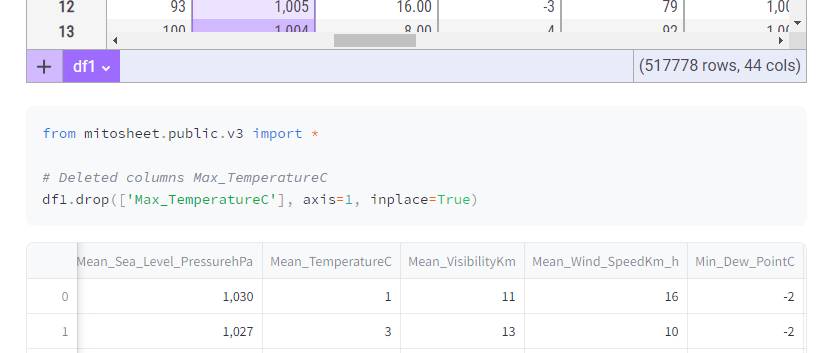
You can then select which categorical columns to target enconde:



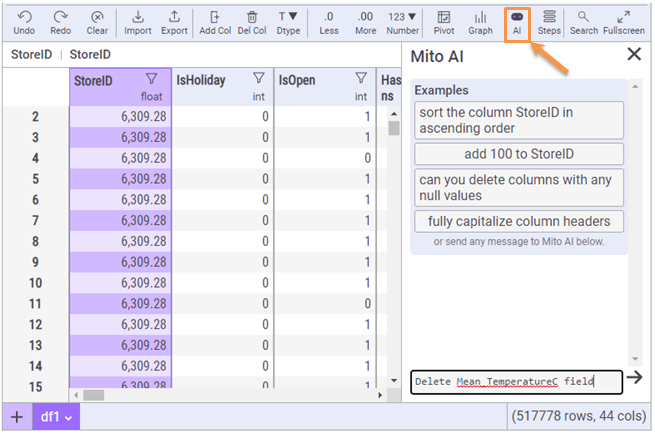
You can use Mitosheet to further transform the data:



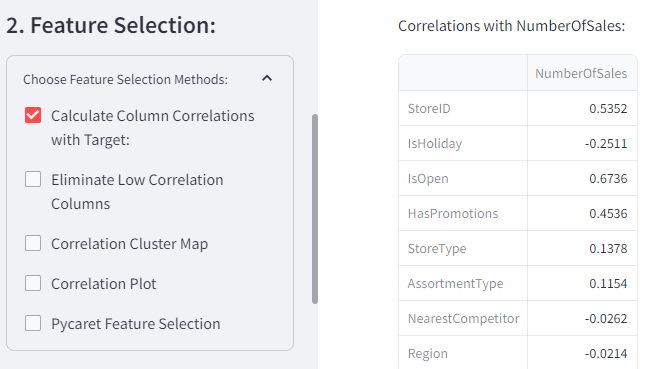
After performing each action, the code for the transformation is displayed followed by the transformed dataset:



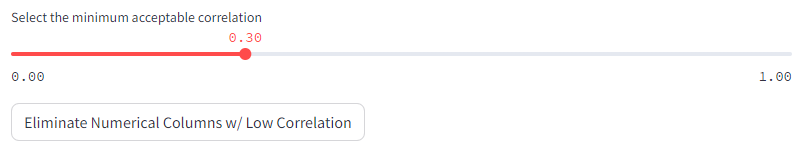
Mitosheet now has a natural language assistant powered by ChatGPT. You can send natural language commands to it and it will transform the data accordingly:



Section 2 has items to help with feature selection. You can start by calculating the textual correlations between each column and the target column;



The table generated can help you set a threshold to use the next feature, which is eliminating columns with correlations below a certain value. You can use the slider to set the value of the correlation, below which, the column should be deleted:

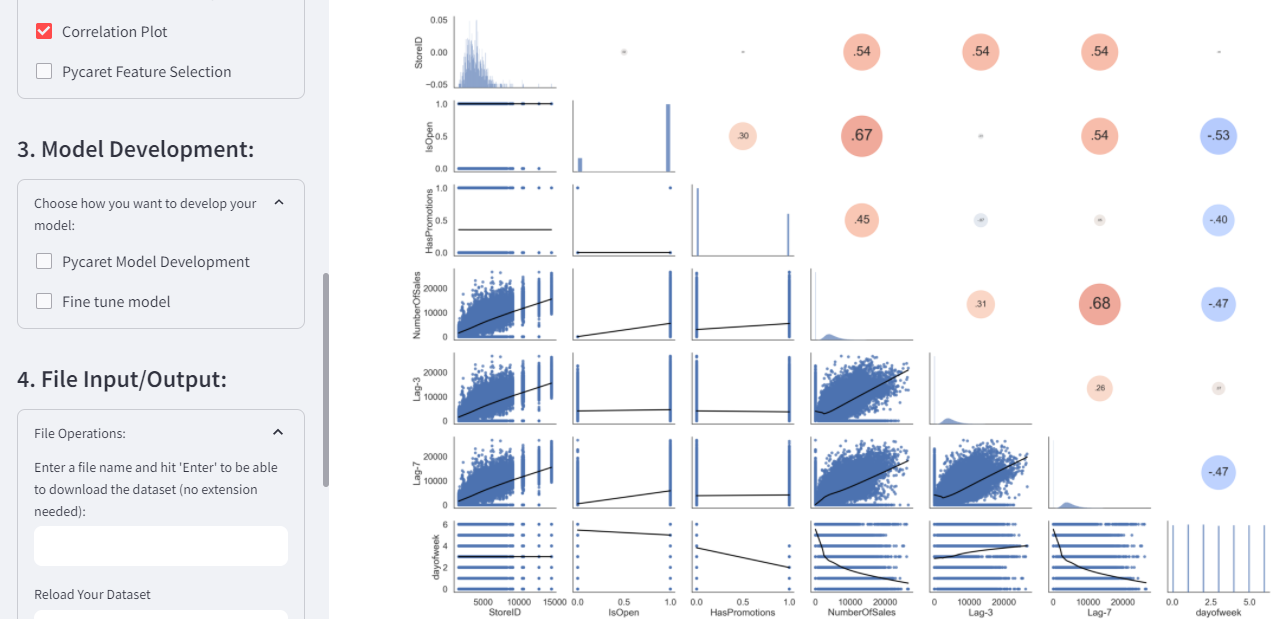


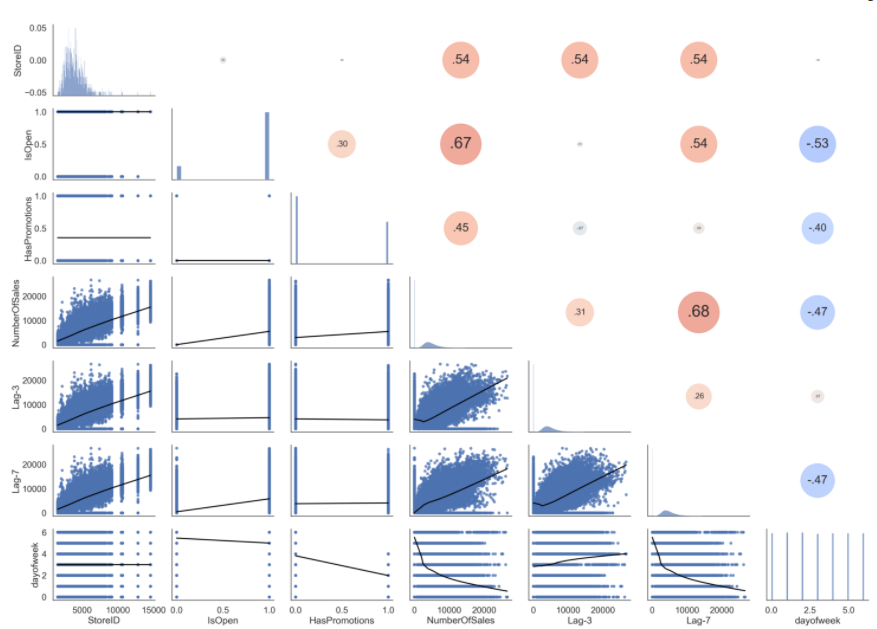
After clicking the button, the dashboard will keep all columns with absolute correlation values greater than 0.3 (in this case). Absolute values are compared because high negative correlations can also be good predictive features.

You can now create a correlation cluster map with a dendrogram:

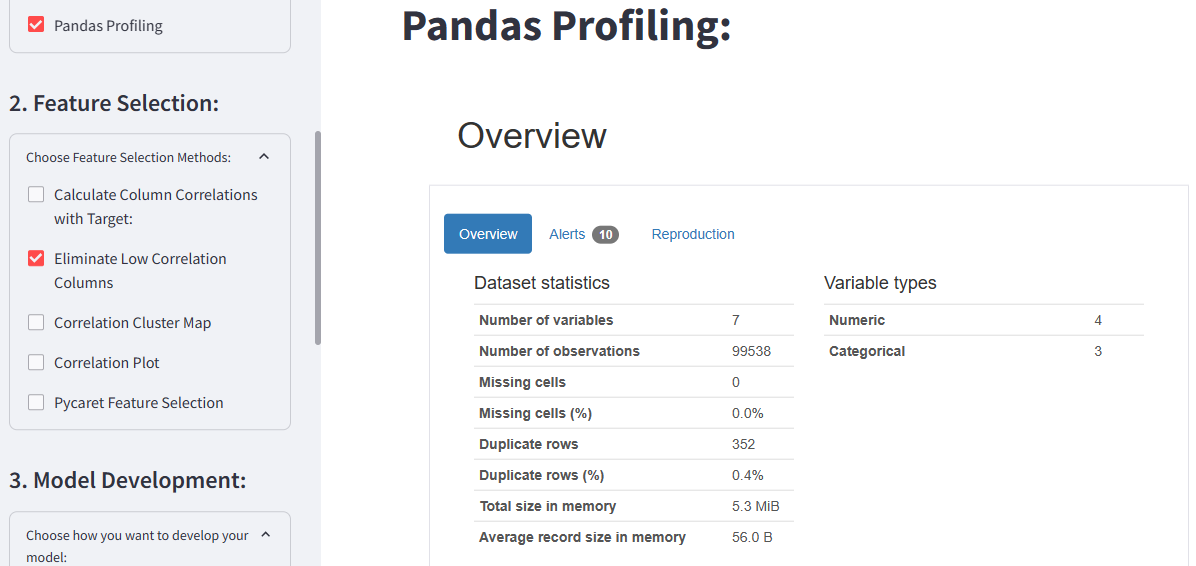


The correlation plot shows the pairwise scatter plots between variables on the bottom left and the correlation values in the upper right:

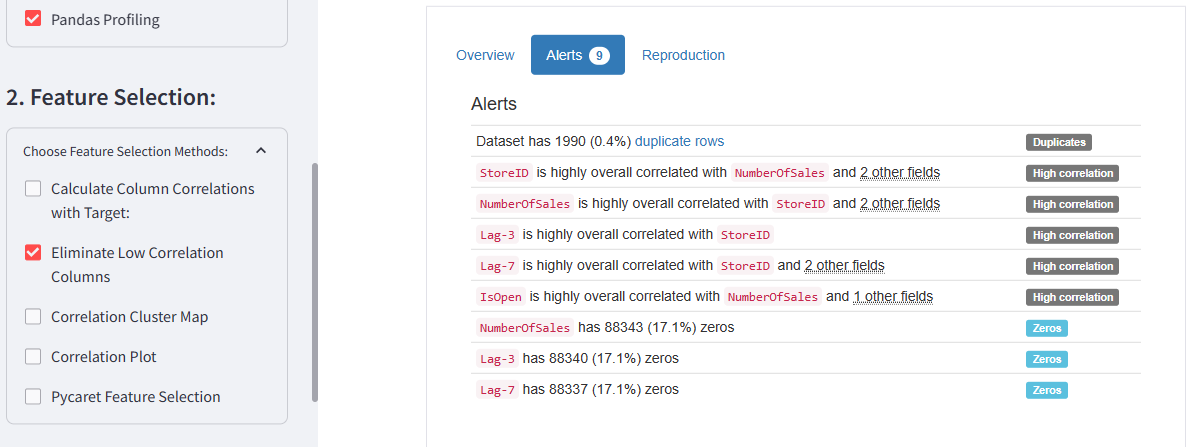




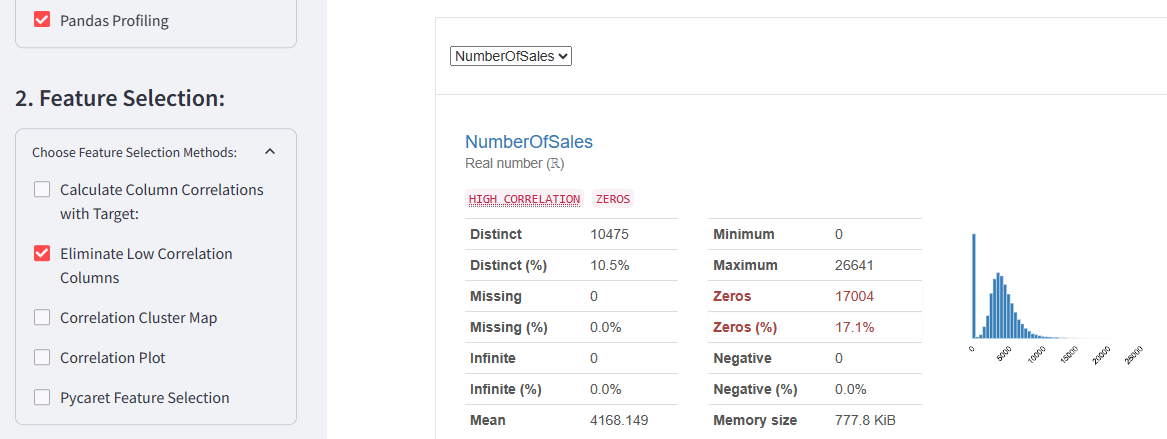
After cleaning the dataset, optionally, you can do further exploration with Pandas Profiling. It is ideal to use Pandas Profiling after you have eliminated most unneeded columns because calculating all graphs for a dataset with many columns can take a long time:



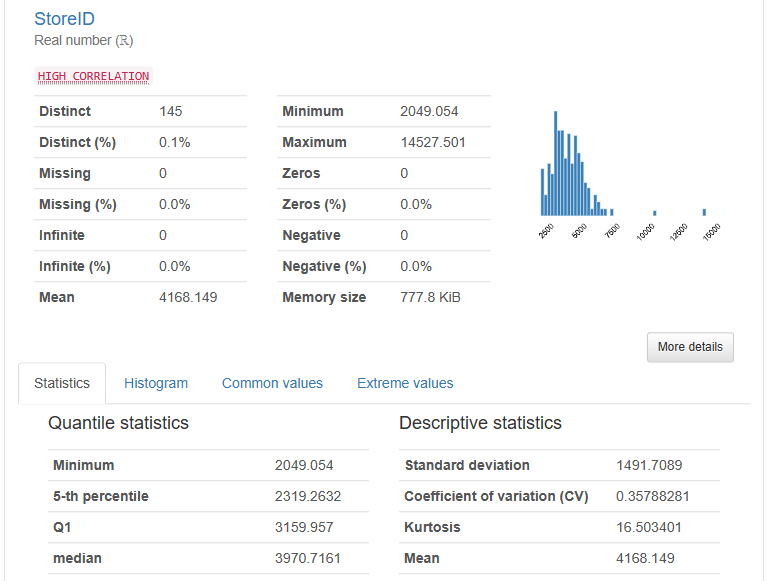
The alerts section of the overview can be particularly useful in calling out facts that might need further exploration:



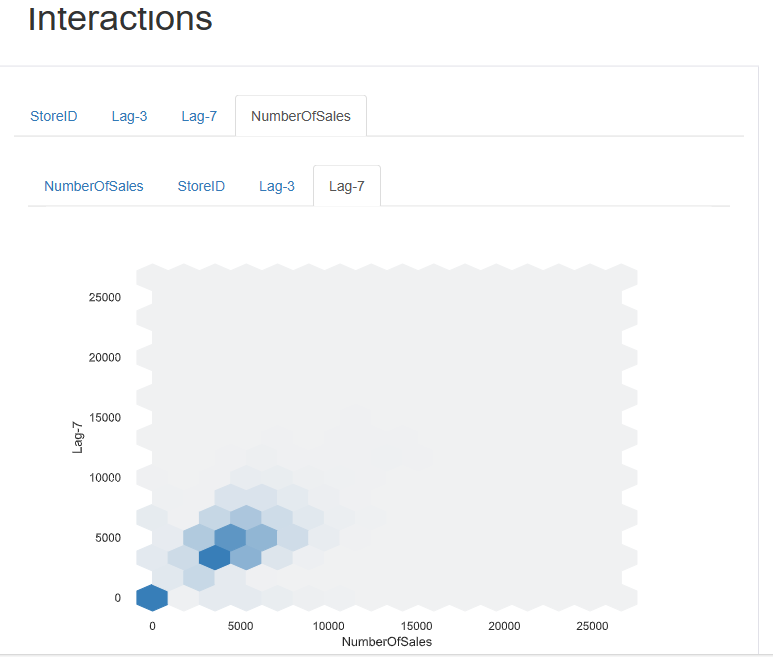
You can view the value distribution of any column by selecting the column name in the drop down. This will also give you several statistics about that column:



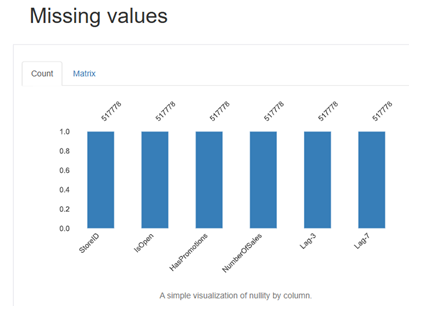
If you click on the “More details” button you get further summary statistics:



You can also perform individual scatter plots between any two columns:

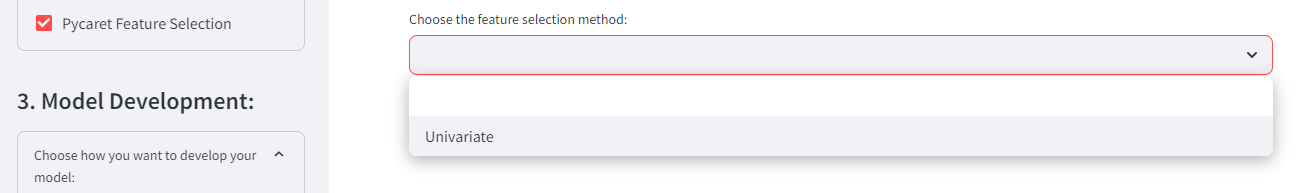


Pandas Profiling will also perform missing value analysis:

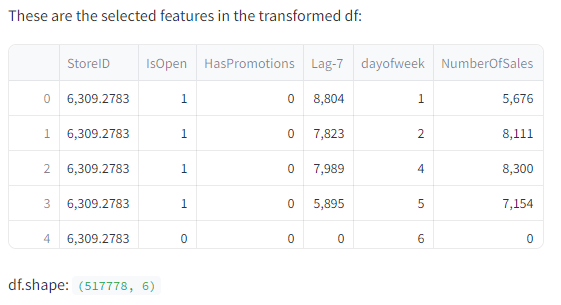


Like it has been mentioned the disadvantage of Pandas Profiling is that it will perform all of these analyses and it will take a long time depending on you dataset size. To choose which graphs to execute you have to change a config file.

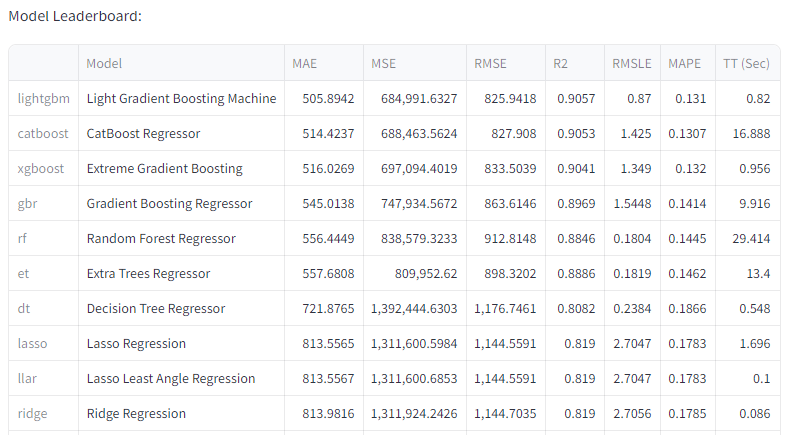
You can use Pycaret to perform feature selection. You will be able to choose different selection methods, but right now “Univariate” is the only one implemented:



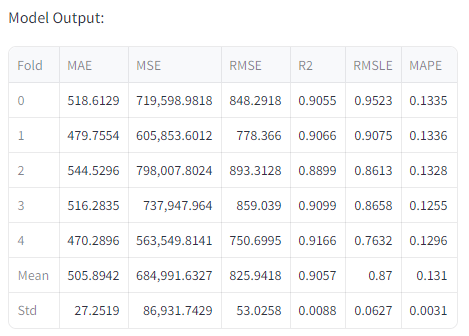
After selecting “Univariate”, pycaret shows the setup table and then the selected features:



For this problem, you get better results by selecting the features with the correlation equal to or above 0.3. With the features selected you can ask Pycaret to develop several models and create a leadearboard:



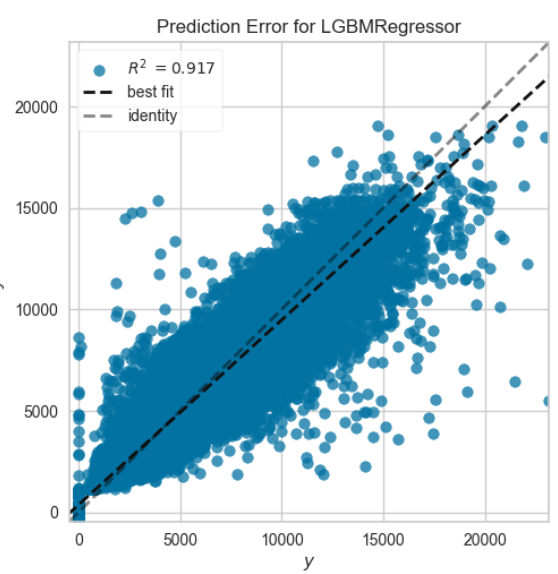
With the selected features, lightgbm was the best model. You can see the result for each of the 5 cross-validations:



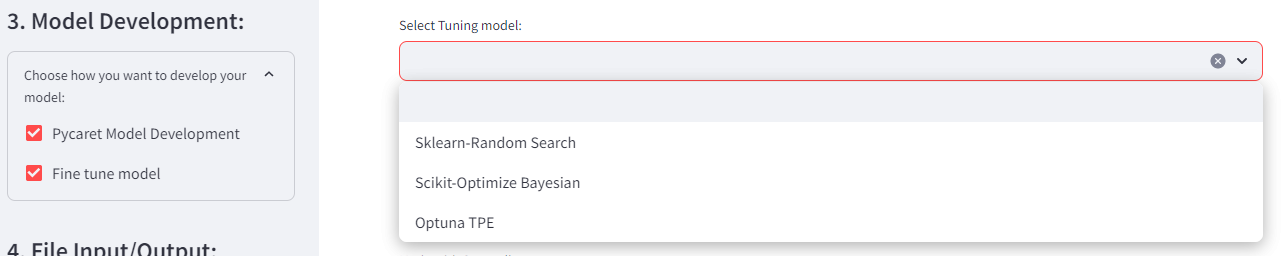
For this competition mean absolute error was the judging criteria and the dashboard is setup to try to minimize MAE. Without hyper parameter optimization we can see in the table above that a mean MAE of 534 was reached. On the test set these were the results:

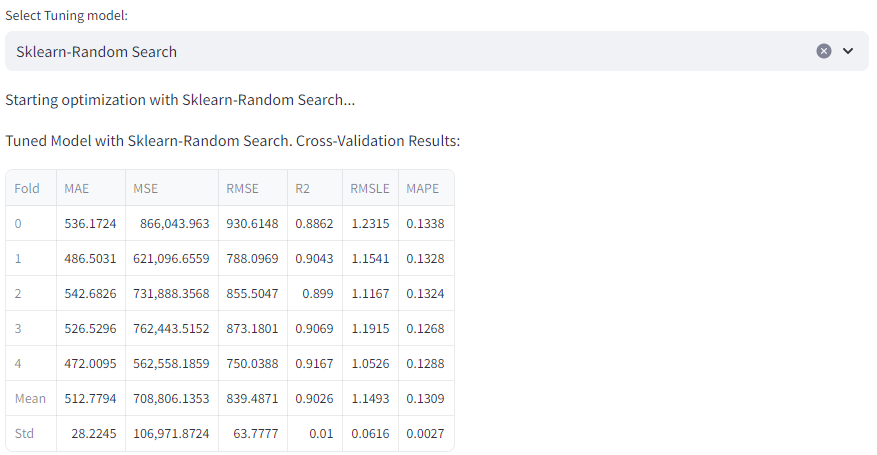


We can see that we reached an R2 of 0.92:



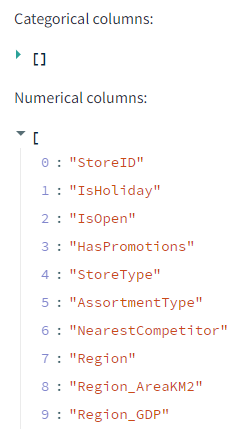
We can then choose a hyper-parameter optimization method:





Which not always improve the results because we are only running the parameter optimization for 5 iterations.

With and automated dashboard like this it becomes very easy to explore and try different feature selections. I was able to get great results with the following features:

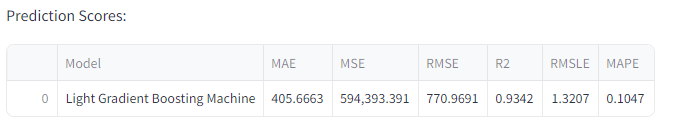
 

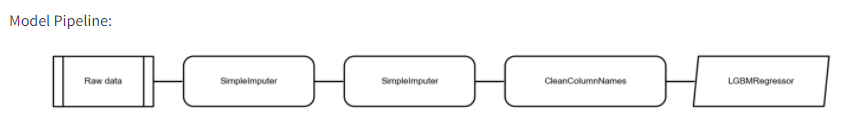
The StoreID, StoreType and AssortmentType, categorical columns were all target encoded.

This is what the model leaderboard looked like:

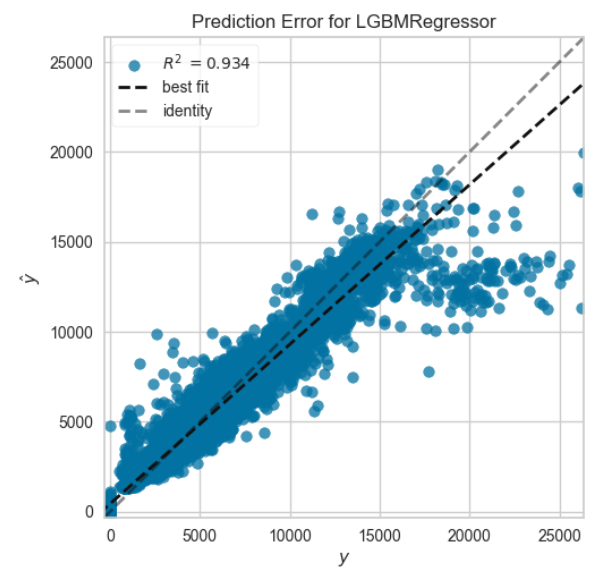


These were the un-optimized model prediction scores:

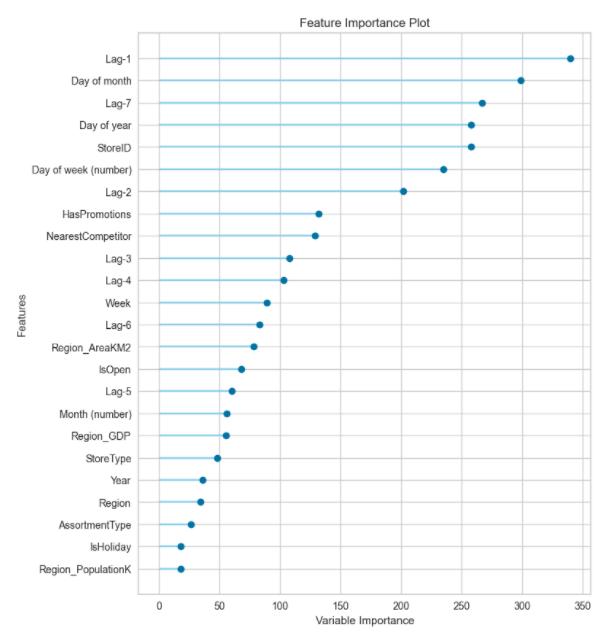




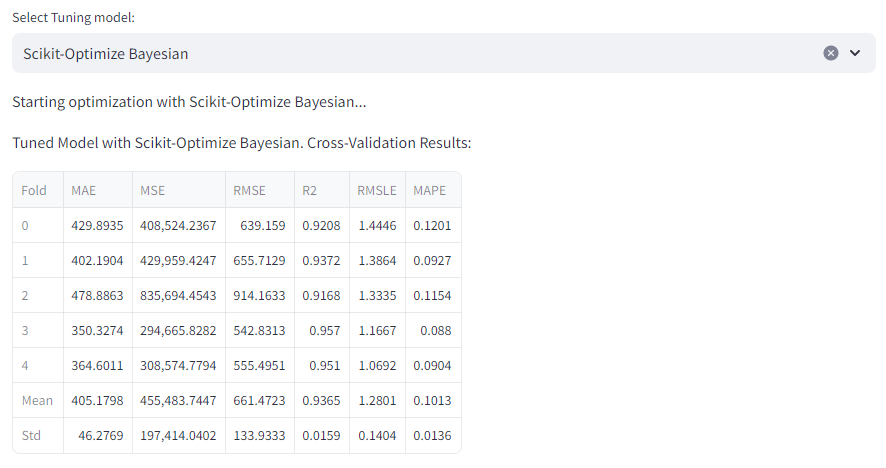
Prediction error graph:

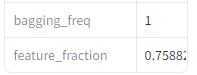
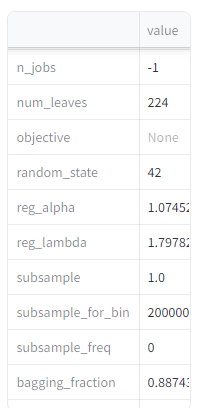
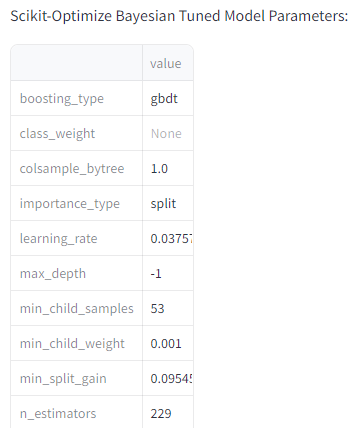


Feature importance plot:

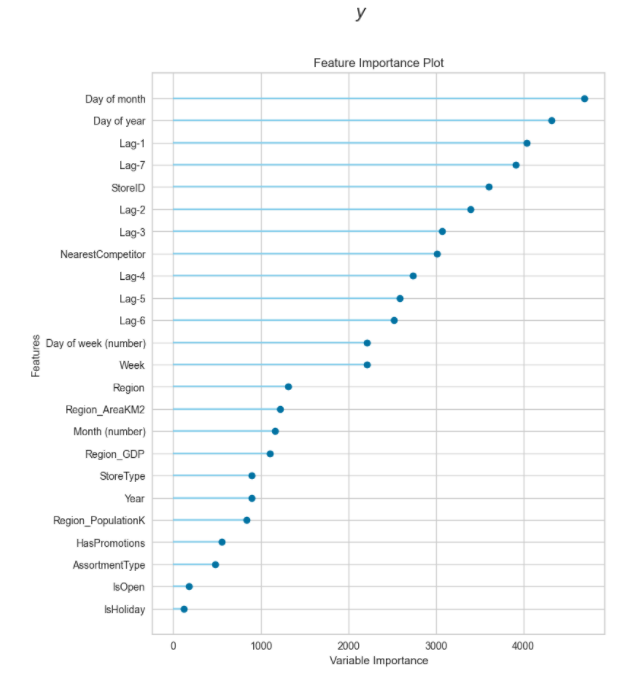


Results were able to be improved by using Bayesian hyper-parameter tuning:





In the next page we can see how feature importance has changed for the optimized model:

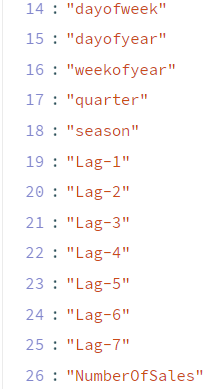


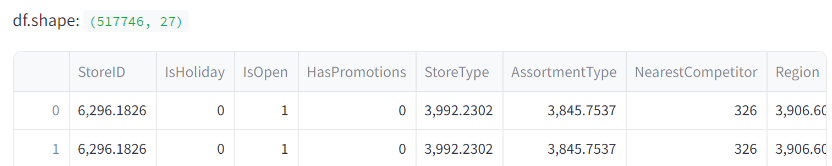
With these settings an MAE of 395.4 was achieved on the test set, which is at the same level of the first place in the competition:

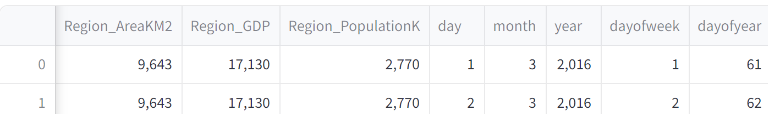


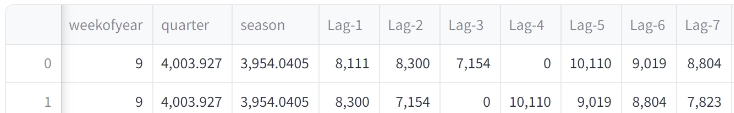
This was even better than the result achieved with the Fast.ai neural network library (subject of a previous study) which had reached 396.4 on the test set.

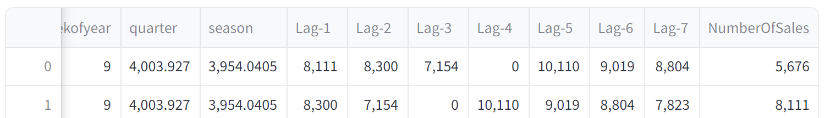
If we target encode the region, quarter and season beyond the StoreID and StoreType, the results can be further improved. This is the new best performing data set:



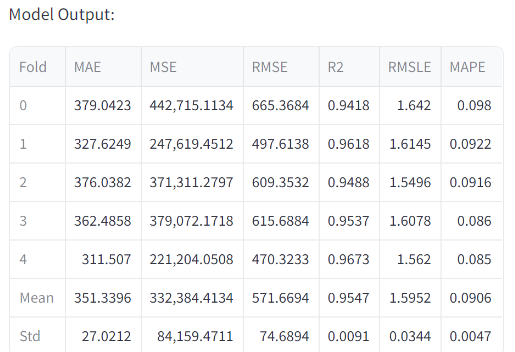




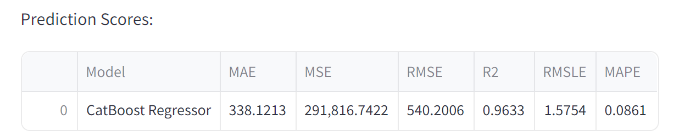


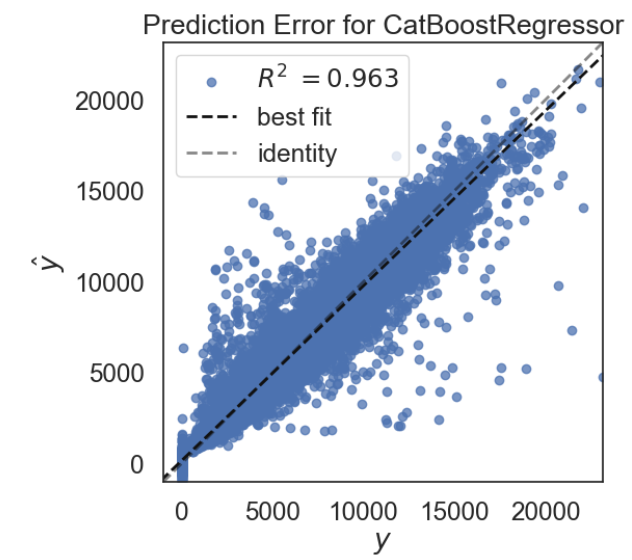


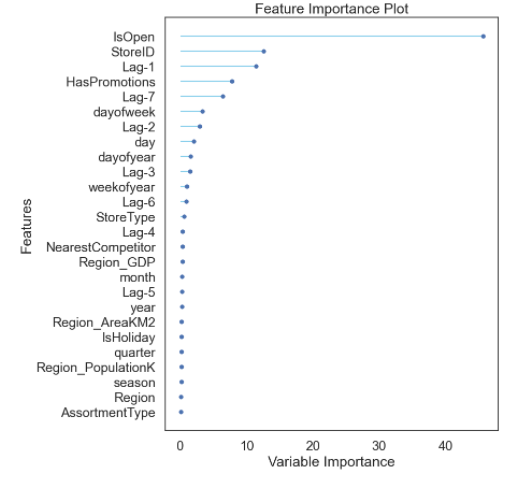
Cross validation scores were:

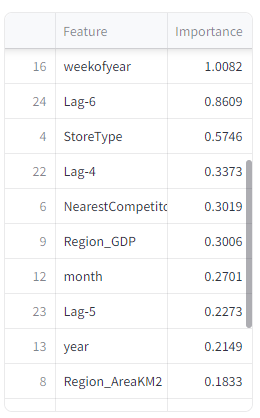
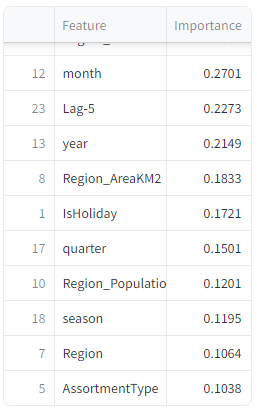


So for the cross validation set an MAE of 351.3 and an R2 of 95.5% was achieved. Prediction scores on the test set were:







This yields an R2 of 96.3% on the test set! Dataset is saved in file TrainBench\_V4.1.csv.