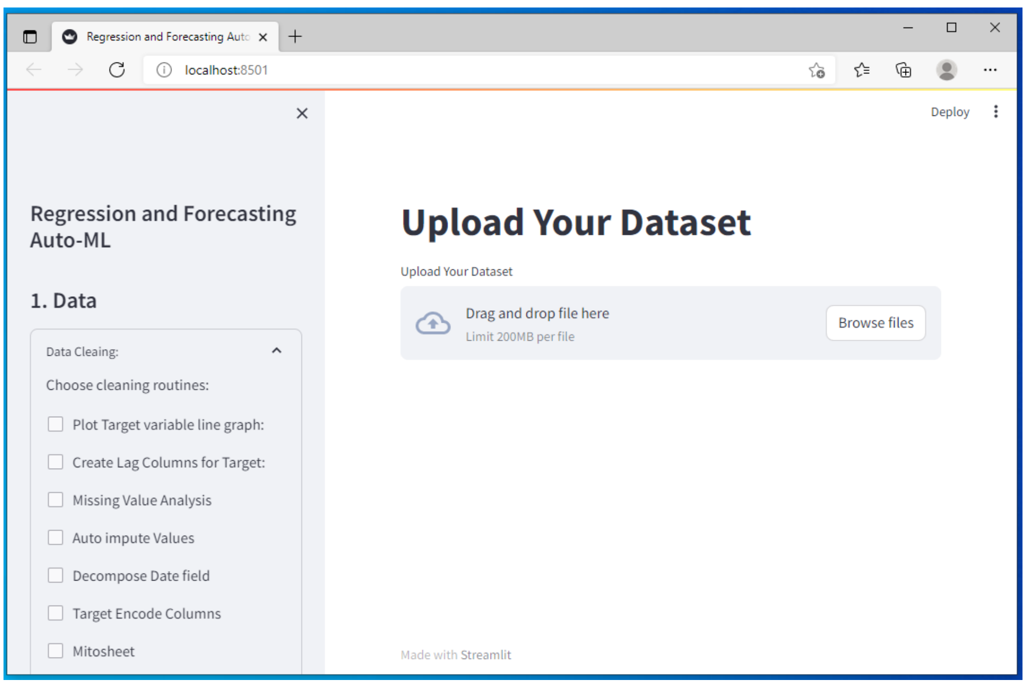
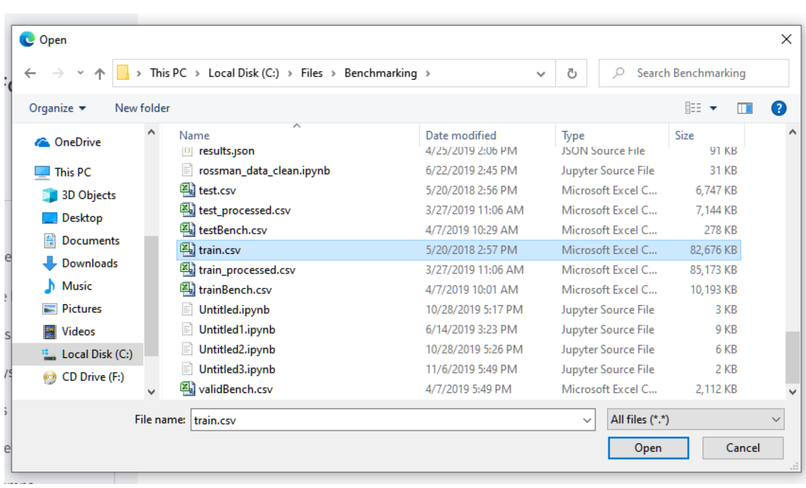
Regression Auto-ML and Forecasting

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

train.csv 80.7MB

	StoreID	Date	IsHoliday	IsOpen	HasPromotions	StoreType	AssortmentType	NearestCompetitor	R
0	1,000	01/03/2016	0	1	0	Hyper Market	General	326	
1	1,000	02/03/2016	0	1	0	Hyper Market	General	326	

	StoreID	IsHoliday	IsOpen	HasPromotions	NearestCompetitor	Region	NumberOfCustomers	NumberOfSa
count	523,021	523,021	523,021	523,021	523,021	523,021	523,021	523,
mean	1,373.9539	0.0297	0.8297	0.3833	8,002.7112	5.7132	259.3251	4,057.3
std	216.3958	0.1698	0.3759	0.4862	11,537.7075	3.3572	185.8571	2,729.6
min	1,000	0	0	0	47	0	0	
25%	1,187	0	1	0	1,057	3	166	2,
50%	1,373	0	1	0	3,321	6	251	4,

Categorical columns:

```
[
  0 : "Date"
  1 : "StoreType"
  2 : "AssortmentType"
  3 : "Events"
]
```

Numerical columns:

```
[
  0 : "StoreID"
  1 : "IsHoliday"
```

df.shape: `(523021, 36)`

Variable Types:

	0
StoreID	int64
Date	object
IsHoliday	int64
IsOpen	int64

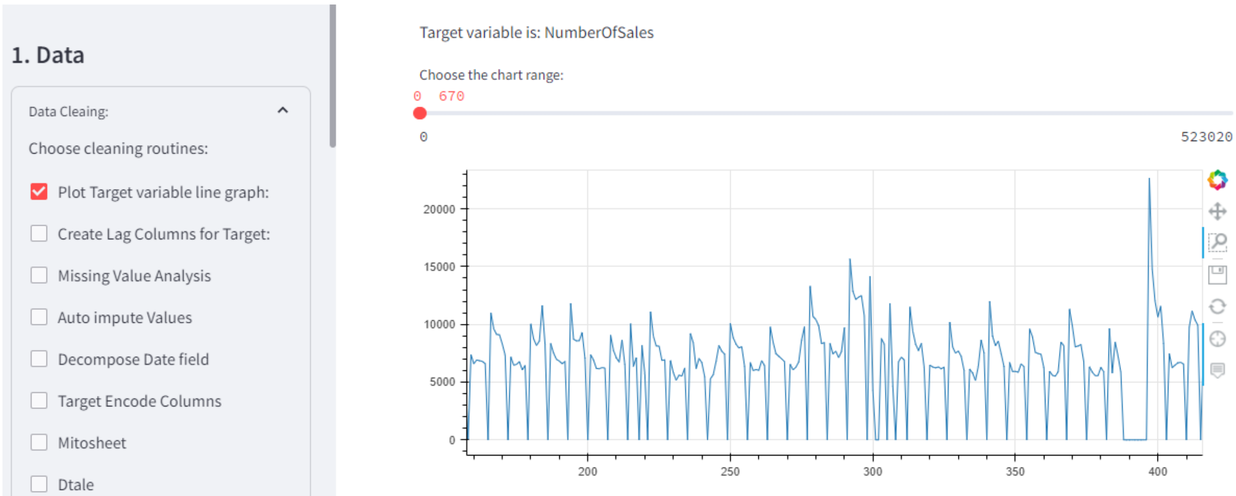
Then it asks you to select the target variable (the variable you want to forecast or your 'Y'). All correlations and feature selection will be made with respect to this variable:

Select the target column:

NumberOfSales

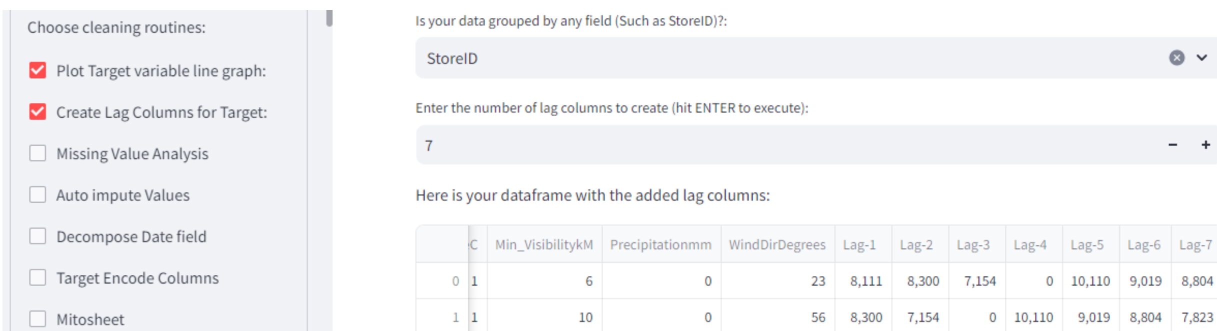
Target variable is: NumberOfSales

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

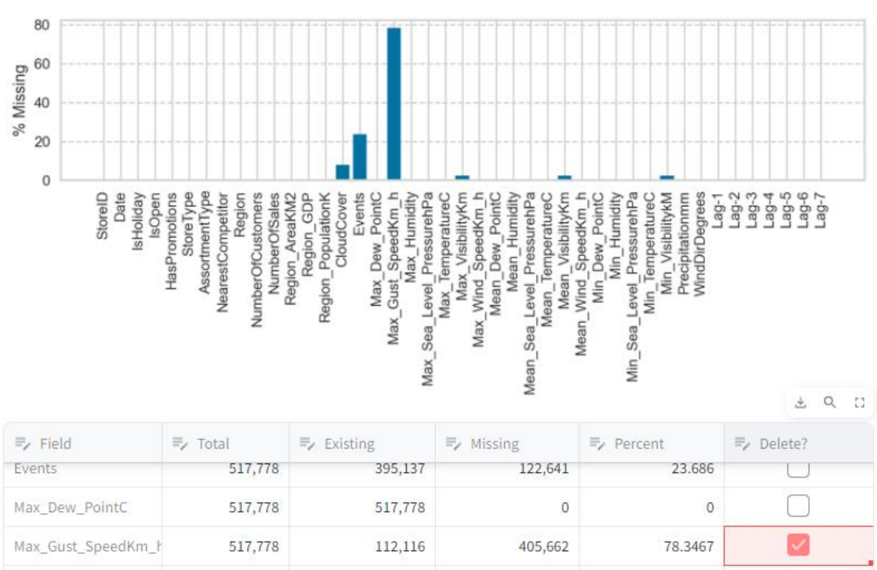


You can see that the data is cyclical going to 0 every Sunday.

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

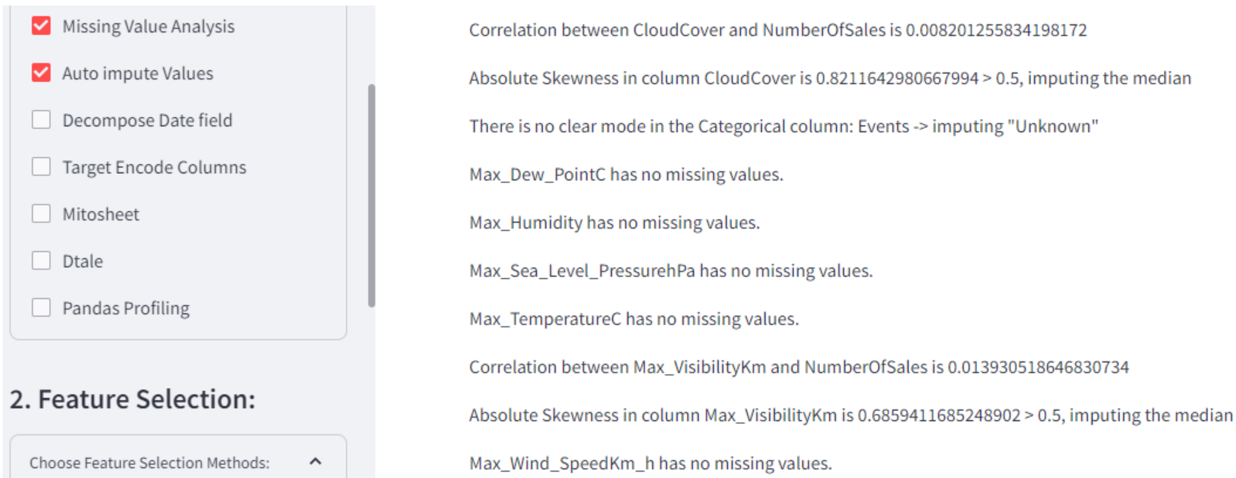


You can then perform missing value analysis. The dashboard gives you a bar chart of the % of rows missing and an opportunity to exclude any columns that you would like in case they have a high percentage missing:



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:

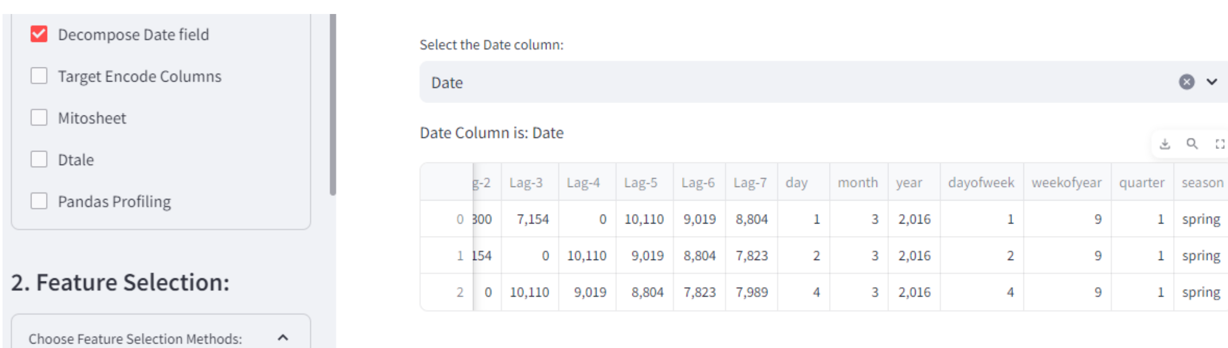


Here you can see the code o how this is determined:

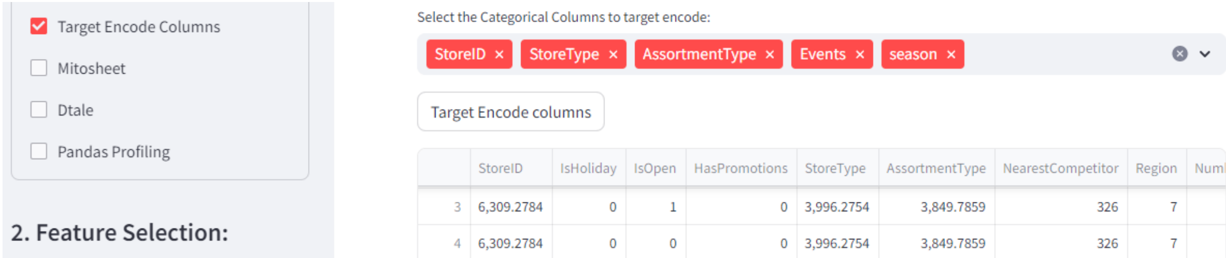
```
AutoImpute = DataClean.checkbox("Auto impute Values")
if AutoImpute:
  # Define Auto Impute function:
  def impute_column(df, col, target):
    # Check if the column has any missing values
    if df[col].isnull().any():
      # Get the percentage of missing values in the column
      percent = df[col].isnull().mean() * 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 ***********************<-----")
```

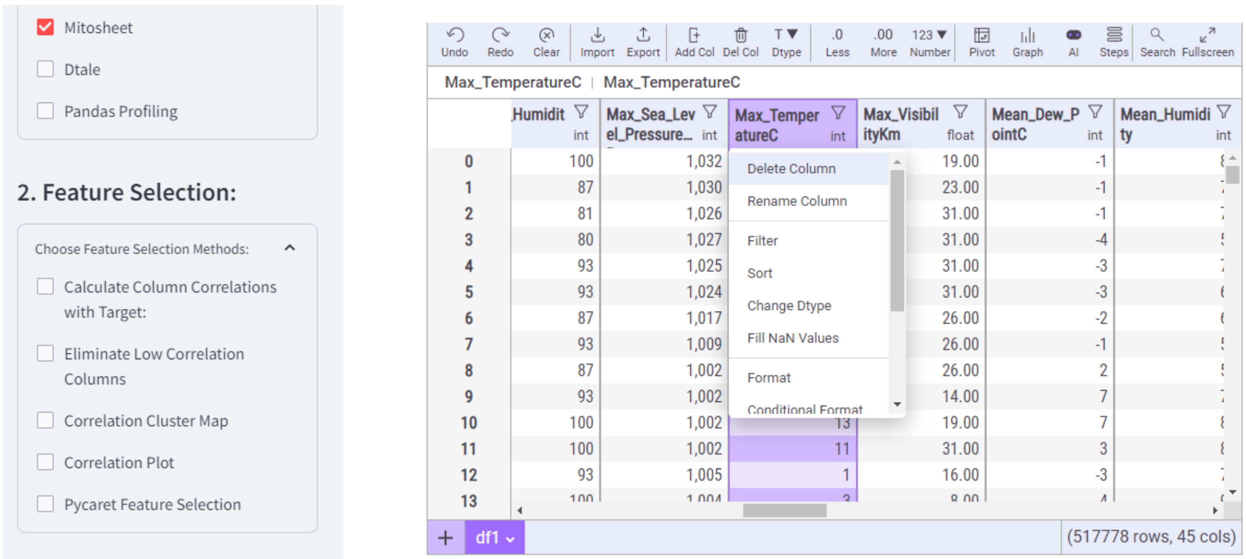
Next, you have the opportunity of decomposing the date into its components:



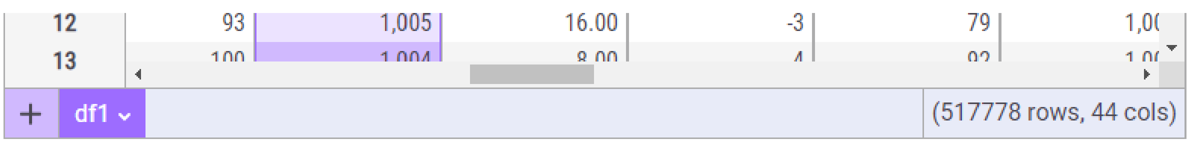
You can then select which categorical columns to target enconde:



You can use Mitosheet to further transform the data:



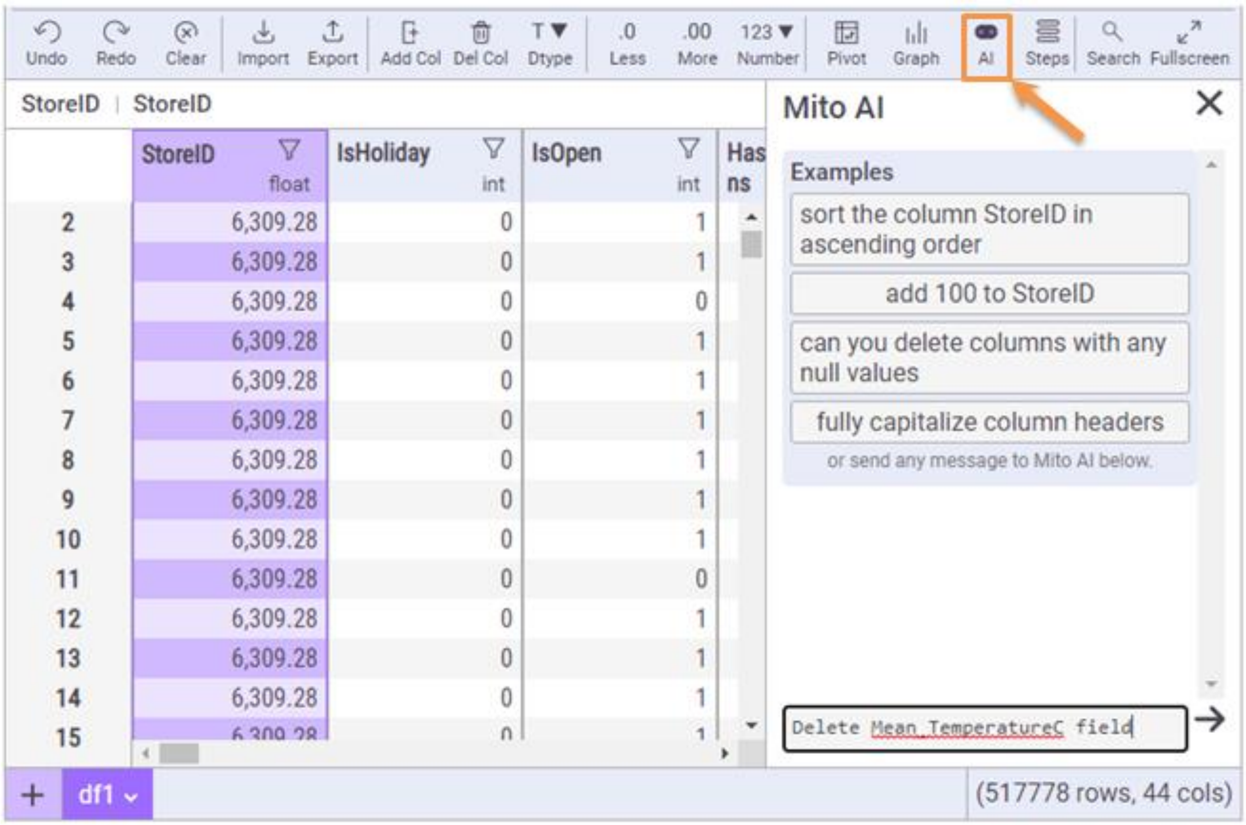
After performing each action, the code for the transformation is displayed followed by the transformed dataset:



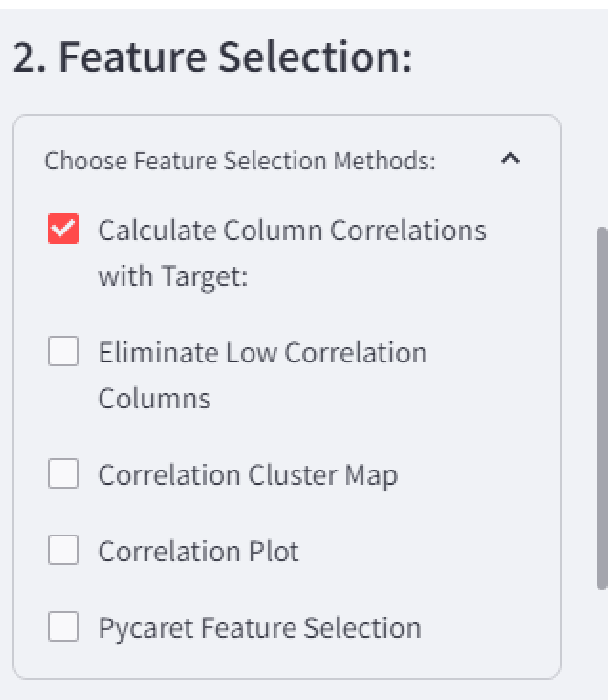

```
from mitosheet.public.v3 import *

# Deleted columns Max_TemperatureC
df1.drop(['Max_TemperatureC'], axis=1, inplace=True)
```

	Mean_Sea_Level_PressurehPa	Mean_TemperatureC	Mean_VisibilityKm	Mean_Wind_SpeedKm_h	Min_Dew_PointC
0	1,030	1	11	16	-2
1	1,027	3	13	10	-2

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:



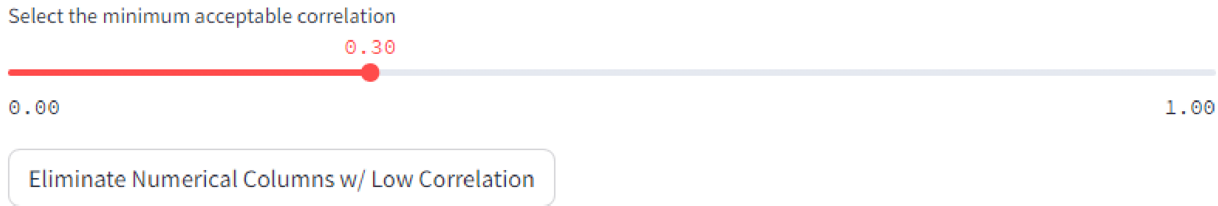
Section 2 has items to help with feature selection. You can start by calculating the textual correlations between each column and the target column;



Correlations with NumberOfSales:

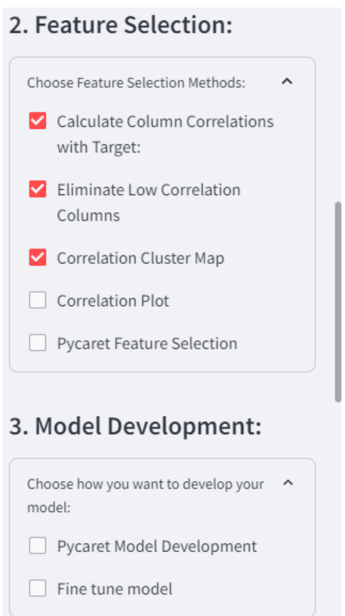
	NumberOfSales
StoreID	0.5352
IsHoliday	-0.2511
IsOpen	0.6736
HasPromotions	0.4536
StoreType	0.1378
AssortmentType	0.1154
NearestCompetitor	-0.0262
Region	-0.0214

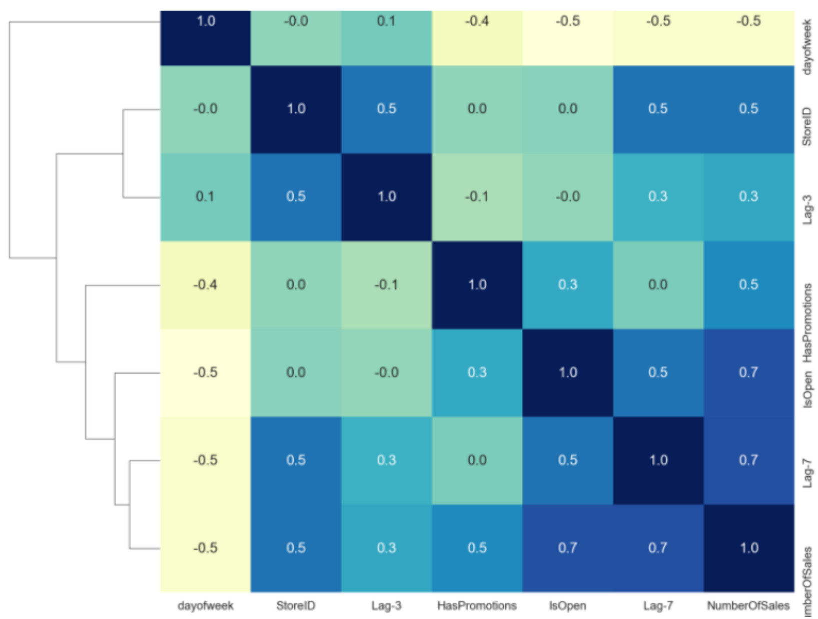
The table generated can help you set a threshold to use the next feature, which is eliminating columns with correlations below a certain value. You can use the slider to set the value of the correlation, below which, the column should be deleted:



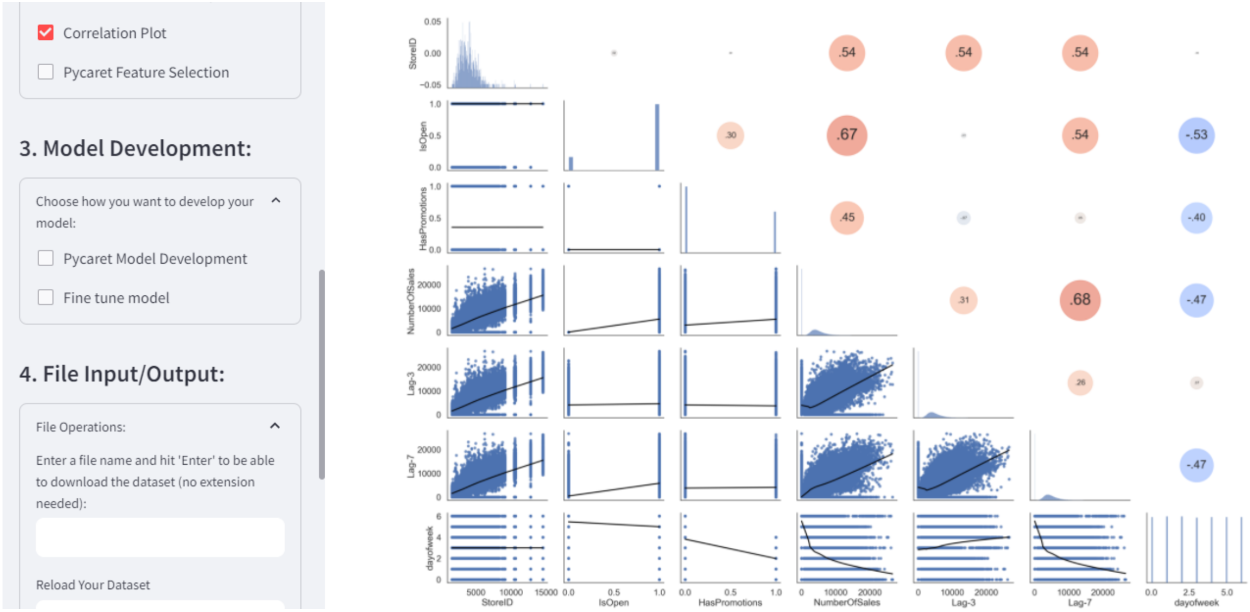
After clicking the button, the dashboard will keep all columns with absolute correlation values greater than 0.3 (in this case). Absolute values are compared because high negative correlations can also be good predictive features.

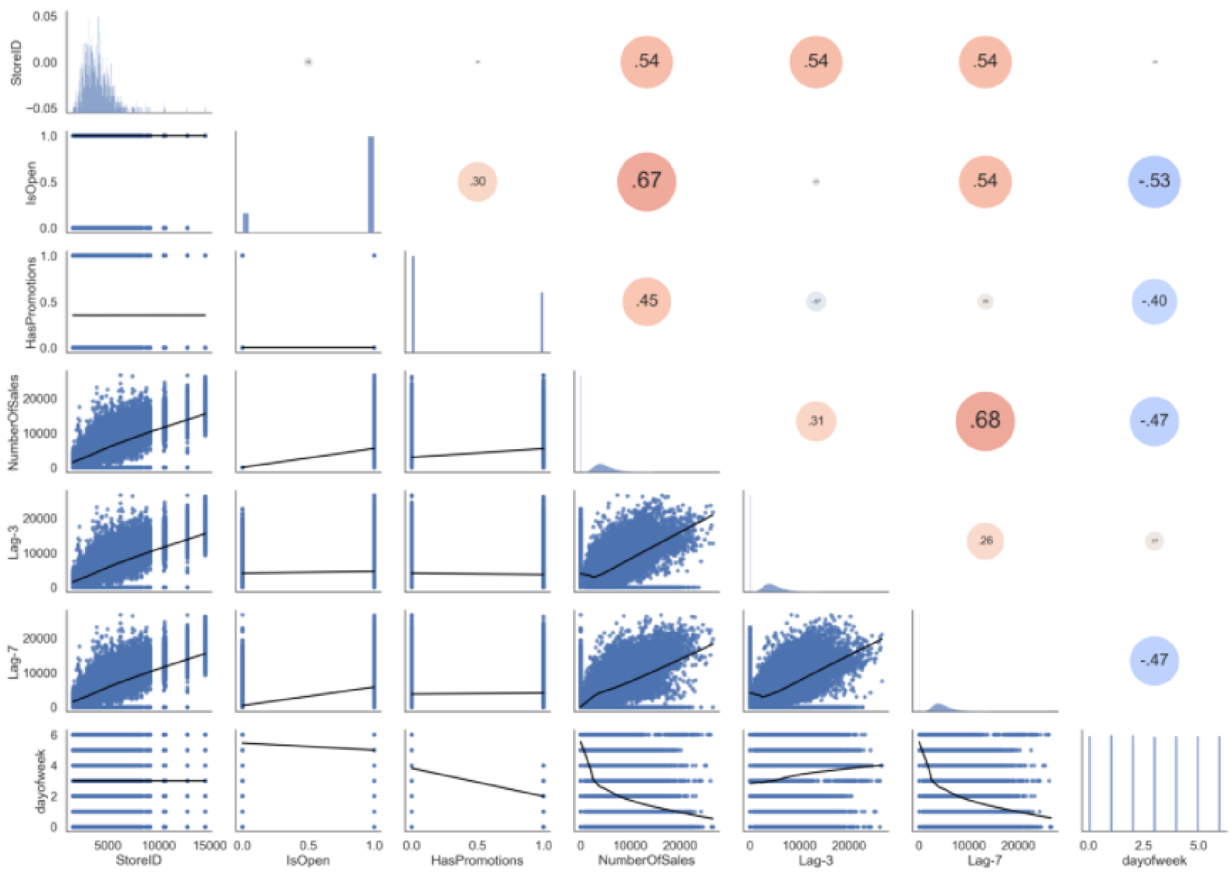
You can now create a correlation cluster map with a dendrogram:



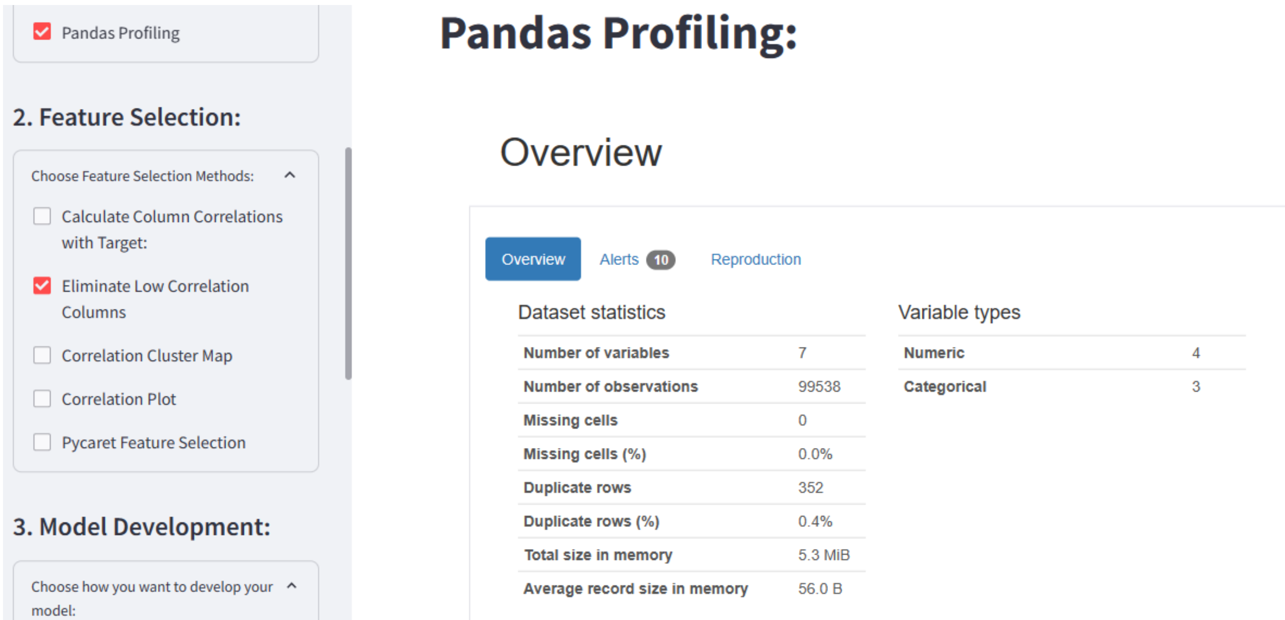


The correlation plot shows the pairwise scatter plots between variables on the bottom left and the correlation values in the upper right:

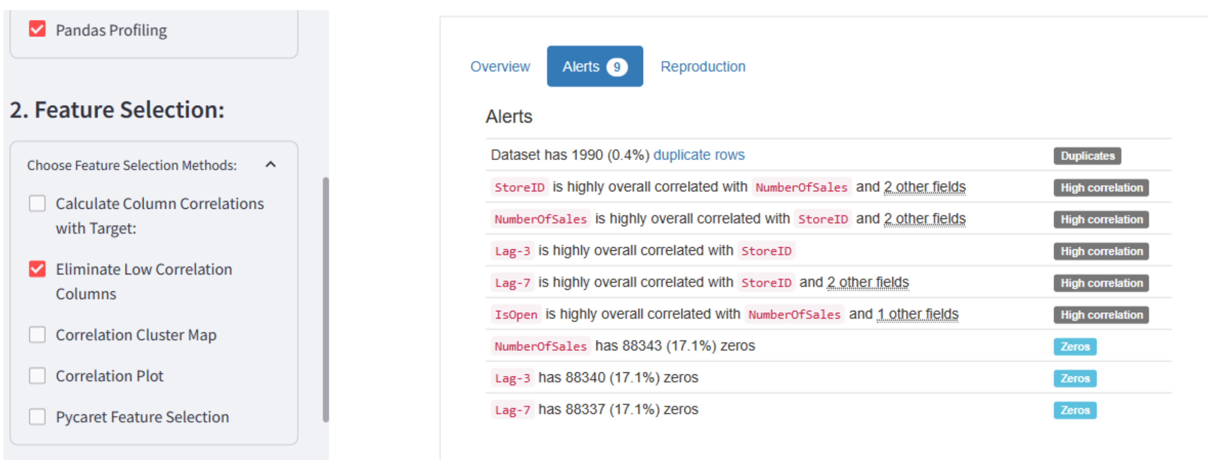




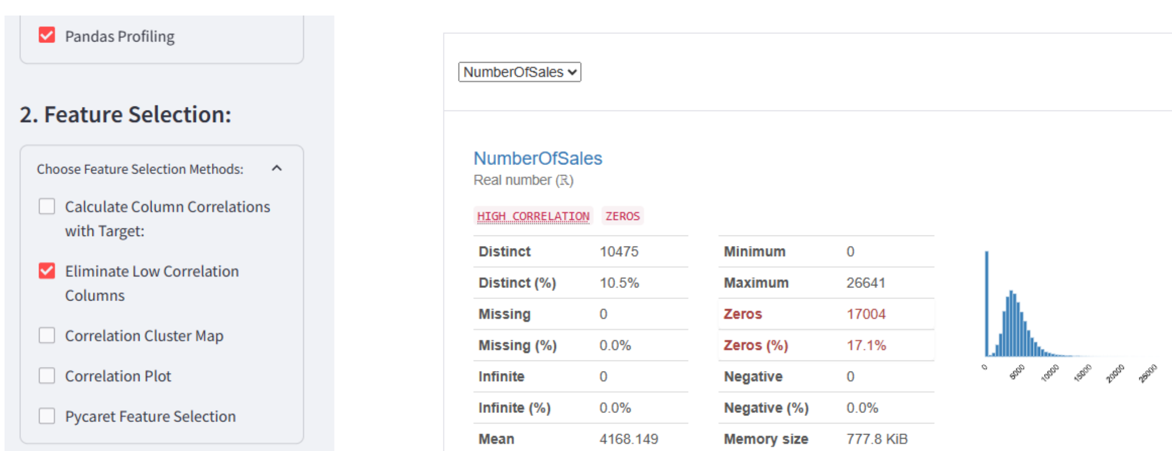
After cleaning the dataset, optionally, you can do further exploration with Pandas Profiling. It is ideal to use Pandas Profiling after you have eliminated most unneeded columns because calculating all graphs for a dataset with many columns can take a long time:



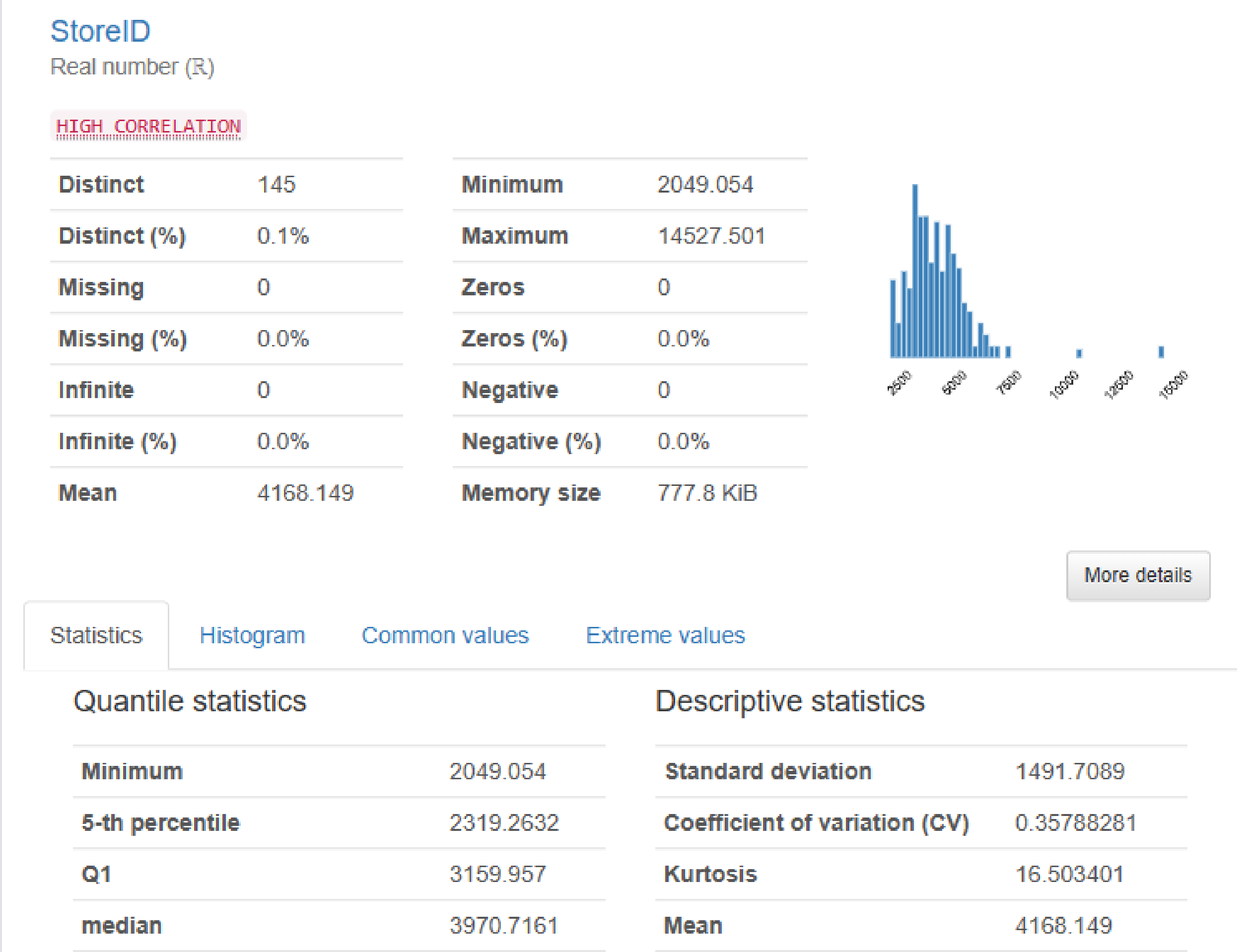
The alerts section of the overview can be particularly useful in calling out facts that might need further exploration:



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:



StoreID

Real number (ℝ)

HIGH CORRELATION

Distinct	145
Distinct (%)	0.1%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	4168.149

Minimum	2049.054
Maximum	14527.501
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	777.8 KiB

More details

Statistics | Histogram | Common values | Extreme values

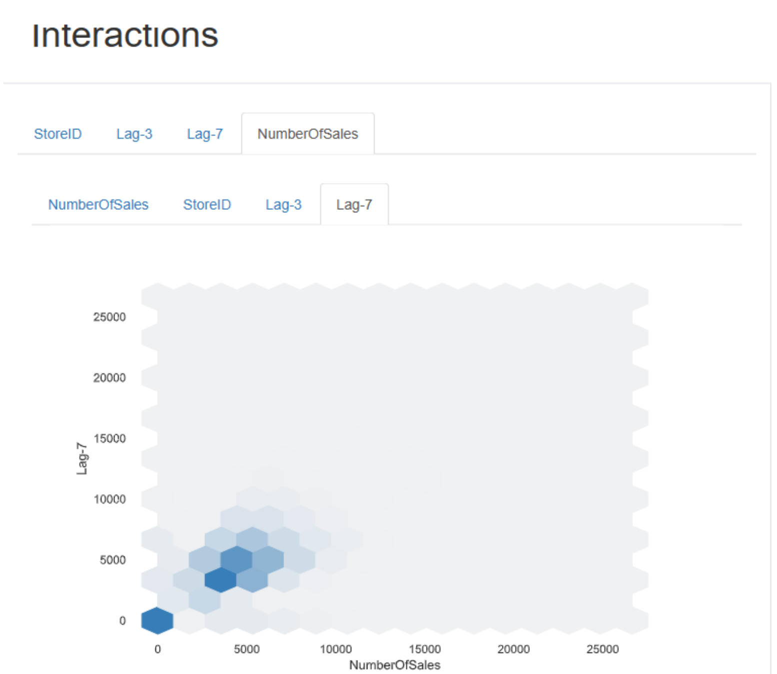
Quantile statistics

Minimum	2049.054
5-th percentile	2319.2632
Q1	3159.957
median	3970.7161

Descriptive statistics

Standard deviation	1491.7089
Coefficient of variation (CV)	0.35788281
Kurtosis	16.503401
Mean	4168.149

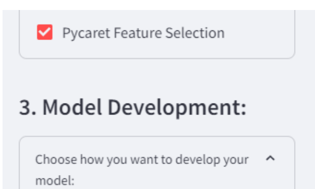
You can also perform individual scatter plots between any two columns:

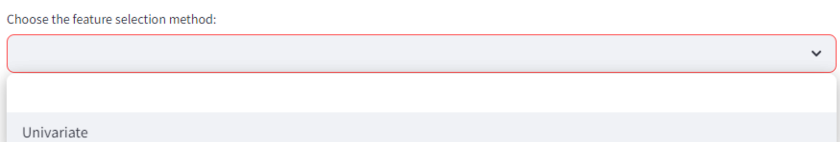


Pandas Profiling will also perform missing value analysis:

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:

These are the selected features in the transformed df:

	StoreID	IsOpen	HasPromotions	Lag-7	dayofweek	NumberOfSales
0	6,309.2783	1	0	8,804	1	5,676
1	6,309.2783	1	0	7,823	2	8,111
2	6,309.2783	1	0	7,989	4	8,300
3	6,309.2783	1	0	5,895	5	7,154
4	6,309.2783	0	0	0	6	0

df.shape: `(517778, 6)`

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:

Model Leaderboard:

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lightgbm	Light Gradient Boosting Machine	505.8942	684,991.6327	825.9418	0.9057	0.87	0.131	0.82
catboost	CatBoost Regressor	514.4237	688,463.5624	827.908	0.9053	1.425	0.1307	16.888
xgboost	Extreme Gradient Boosting	516.0269	697,094.4019	833.5039	0.9041	1.349	0.132	0.956
gbr	Gradient Boosting Regressor	545.0138	747,934.5672	863.6146	0.8969	1.5448	0.1414	9.916
rf	Random Forest Regressor	556.4449	838,579.3233	912.8148	0.8846	0.1804	0.1445	29.414
et	Extra Trees Regressor	557.6808	809,952.62	898.3202	0.8886	0.1819	0.1462	13.4
dt	Decision Tree Regressor	721.8765	1,392,444.6303	1,176.7461	0.8082	0.2384	0.1866	0.548
lasso	Lasso Regression	813.5565	1,311,600.5984	1,144.5591	0.819	2.7047	0.1783	1.696
llar	Lasso Least Angle Regression	813.5567	1,311,600.6853	1,144.5591	0.819	2.7047	0.1783	0.1
ridge	Ridge Regression	813.9816	1,311,924.2426	1,144.7035	0.819	2.7056	0.1785	0.086

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

Model Output:

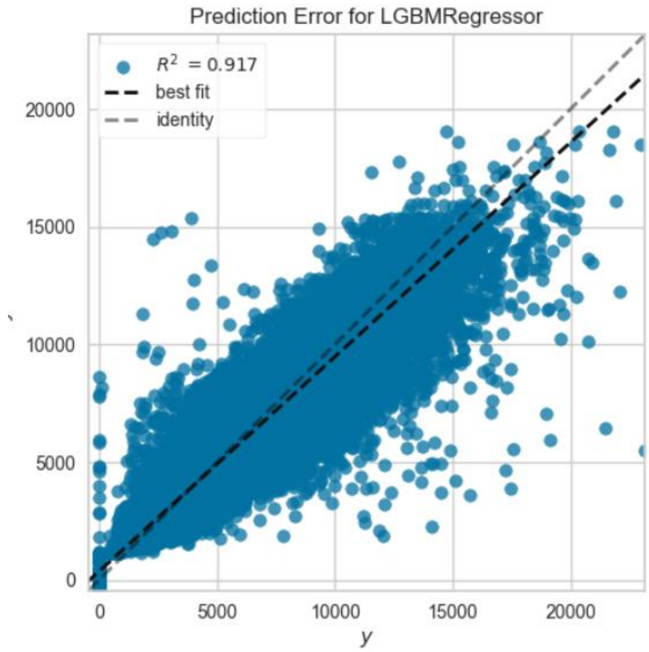
Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	518.6129	719,598.9818	848.2918	0.9055	0.9523	0.1335
1	479.7554	605,853.6012	778.366	0.9066	0.9075	0.1336
2	544.5296	798,007.8024	893.3128	0.8899	0.8613	0.1328
3	516.2835	737,947.964	859.039	0.9099	0.8658	0.1255
4	470.2896	563,549.8141	750.6995	0.9166	0.7632	0.1296
Mean	505.8942	684,991.6327	825.9418	0.9057	0.87	0.131
Std	27.2519	86,931.7429	53.0258	0.0088	0.0627	0.0031

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:

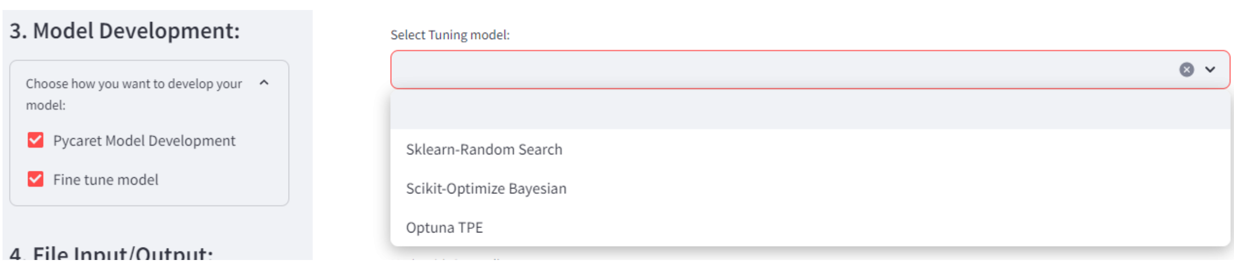
Prediction Scores:

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Light Gradient Boosting Machine	503.8344	660,286.4383	812.5801	0.917	0.7974	0.1291

We can see that we reached an R^2 of 0.92:



We can then choose a hyper-parameter optimization method:



Select Tuning model:

Sklearn-Random Search

Starting optimization with Sklearn-Random Search...

Tuned Model with Sklearn-Random Search. Cross-Validation Results:

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	536.1724	866,043.963	930.6148	0.8862	1.2315	0.1338
1	486.5031	621,096.6559	788.0969	0.9043	1.1541	0.1328
2	542.6826	731,888.3568	855.5047	0.899	1.1167	0.1324
3	526.5296	762,443.5152	873.1801	0.9069	1.1915	0.1268
4	472.0095	562,558.1859	750.0388	0.9167	1.0526	0.1288
Mean	512.7794	708,806.1353	839.4871	0.9026	1.1493	0.1309
Std	28.2245	106,971.8724	63.7777	0.01	0.0616	0.0027

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:

Categorical columns:

```
[]
```

Numerical columns:

```
[
0 : "StoreID"
1 : "IsHoliday"
2 : "IsOpen"
3 : "HasPromotions"
4 : "StoreType"
5 : "AssortmentType"
6 : "NearestCompetitor"
7 : "Region"
8 : "Region_AreaKM2"
9 : "Region_GDP"
10 : "Region_PopulationK"
11 : "Year"
12 : "Month (number)"
13 : "Week"
14 : "Day of year"
15 : "Day of month"
16 : "Day of week (number)"
17 : "Lag-1"
18 : "Lag-2"
19 : "Lag-3"
20 : "Lag-4"
21 : "Lag-5"
22 : "Lag-6"
23 : "Lag-7"
24 : "NumberOfSales"
]
```

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

This is what the model leaderboard looked like:

Model Leaderboard:

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lightgbm	Light Gradient Boosting Machine	412.1304	471,579.1011	676.8232	0.9342	1.3266	0.1031	0.602
catboost	CatBoost Regressor	418.1211	463,528.1383	667.1304	0.9359	1.7192	0.1035	8.192
et	Extra Trees Regressor	431.2827	528,381.0186	712.8857	0.9267	0.1571	0.112	7.222
rf	Random Forest Regressor	435.3737	545,910.6699	728.024	0.9234	0.1569	0.1116	18.458
xgboost	Extreme Gradient Boosting	437.0102	510,236.9396	700.4904	0.9295	1.7227	0.1044	0.514
gbr	Gradient Boosting Regressor	478.6828	568,364.9982	745.2483	0.9201	1.729	0.1192	9.142
dt	Decision Tree Regressor	621.2276	1,091,367.0198	1,030.5009	0.8473	0.2664	0.1557	0.326
llar	Lasso Least Angle Regression	773.8001	1,202,665.327	1,089.6128	0.8291	2.6942	0.1604	0.08
ridge	Ridge Regression	774.6679	1,186,973.0383	1,083.5738	0.8306	2.6989	0.1614	0.074

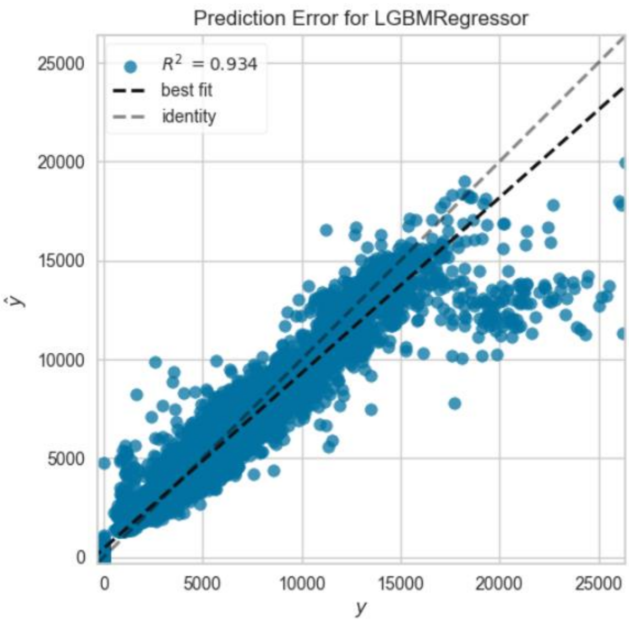
These were the un-optimized model prediction scores:

Prediction Scores:

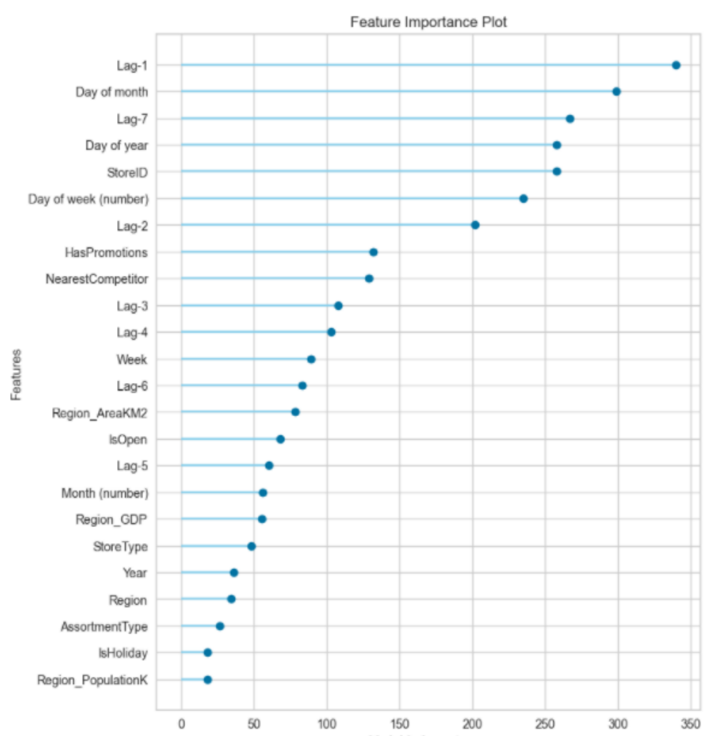
	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Light Gradient Boosting Machine	405.6663	594,393.391	770.9691	0.9342	1.3207	0.1047

Model Pipeline:

Prediction error graph:



Feature importance plot:



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

Select Tuning model:

Scikit-Optimize Bayesian

Starting optimization with Scikit-Optimize Bayesian...

Tuned Model with Scikit-Optimize Bayesian. Cross-Validation Results:

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	429.8935	408,524.2367	639.159	0.9208	1.4446	0.1201
1	402.1904	429,959.4247	655.7129	0.9372	1.3864	0.0927
2	478.8863	835,694.4543	914.1633	0.9168	1.3335	0.1154
3	350.3274	294,665.8282	542.8313	0.957	1.1667	0.088
4	364.6011	308,574.7794	555.4951	0.951	1.0692	0.0904
Mean	405.1798	455,483.7447	661.4723	0.9365	1.2801	0.1013
Std	46.2769	197,414.0402	133.9333	0.0159	0.1404	0.0136

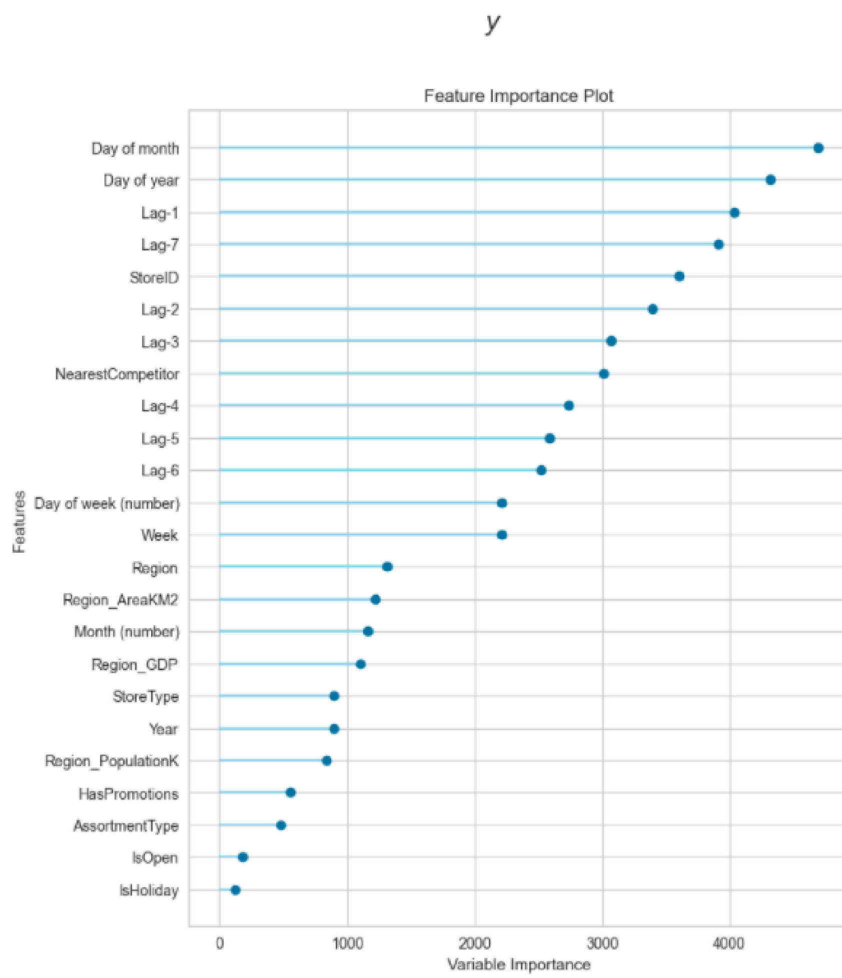
Scikit-Optimize Bayesian Tuned Model Parameters:

	value
boosting_type	gbdt
class_weight	None
colsample_bytree	1.0
importance_type	split
learning_rate	0.03757
max_depth	-1
min_child_samples	53
min_child_weight	0.001
min_split_gain	0.09545
n_estimators	229

	value
n_jobs	-1
num_leaves	224
objective	None
random_state	42
reg_alpha	1.07452
reg_lambda	1.79782
subsample	1.0
subsample_for_bin	200000
subsample_freq	0
bagging_fraction	0.88743

bagging_freq	1
feature_fraction	0.75882

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:

Prediction Scores with Scikit-Optimize Bayesian:

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Light Gradient Boosting Machine	395.3963	822,248.5692	906.7792	0.9089	1.1014	0.0965

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:

```
0 : "StoreID"
1 : "IsHoliday"
2 : "IsOpen"
3 : "HasPromotions"
4 : "StoreType"
5 : "AssortmentType"
6 : "NearestCompetitor"
7 : "Region"
8 : "Region_AreaKM2"
9 : "Region_GDP"
10 : "Region_PopulationK"
11 : "day"
12 : "month"
13 : "year"
14 : "dayofweek"
15 : "dayofyear"
16 : "weekofyear"
17 : "quarter"
18 : "season"
19 : "Lag-1"
20 : "Lag-2"
21 : "Lag-3"
22 : "Lag-4"
23 : "Lag-5"
24 : "Lag-6"
25 : "Lag-7"
26 : "NumberOfSales"
```

df.shape: `(517746, 27)`

	StoreID	IsHoliday	IsOpen	HasPromotions	StoreType	AssortmentType	NearestCompetitor	Region
0	6,296.1826	0	1	0	3,992.2302	3,845.7537	326	3,906.60
1	6,296.1826	0	1	0	3,992.2302	3,845.7537	326	3,906.60

	Region_AreaKM2	Region_GDP	Region_PopulationK	day	month	year	dayofweek	dayofyear
0	9,643	17,130	2,770	1	3	2,016	1	61
1	9,643	17,130	2,770	2	3	2,016	2	62

	weekofyear	quarter	season	Lag-1	Lag-2	Lag-3	Lag-4	Lag-5	Lag-6	Lag-7
0	9	4,003.927	3,954.0405	8,111	8,300	7,154	0	10,110	9,019	8,804
1	9	4,003.927	3,954.0405	8,300	7,154	0	10,110	9,019	8,804	7,823

	kofyear	quarter	season	Lag-1	Lag-2	Lag-3	Lag-4	Lag-5	Lag-6	Lag-7	NumberOfSales
0	9	4,003.927	3,954.0405	8,111	8,300	7,154	0	10,110	9,019	8,804	5,676
1	9	4,003.927	3,954.0405	8,300	7,154	0	10,110	9,019	8,804	7,823	8,111

Cross validation scores were:

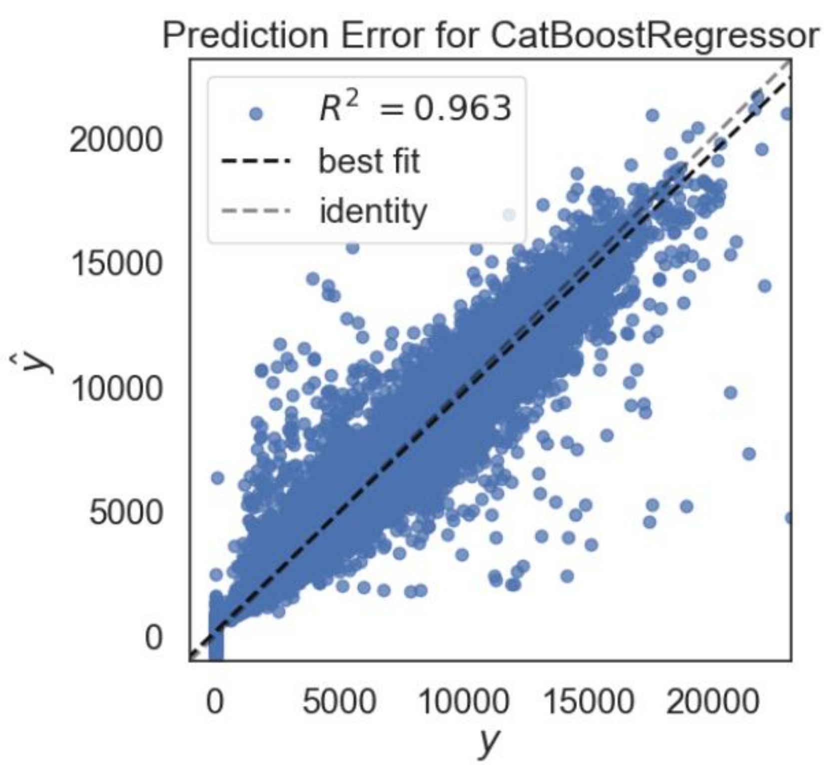
Model Output:

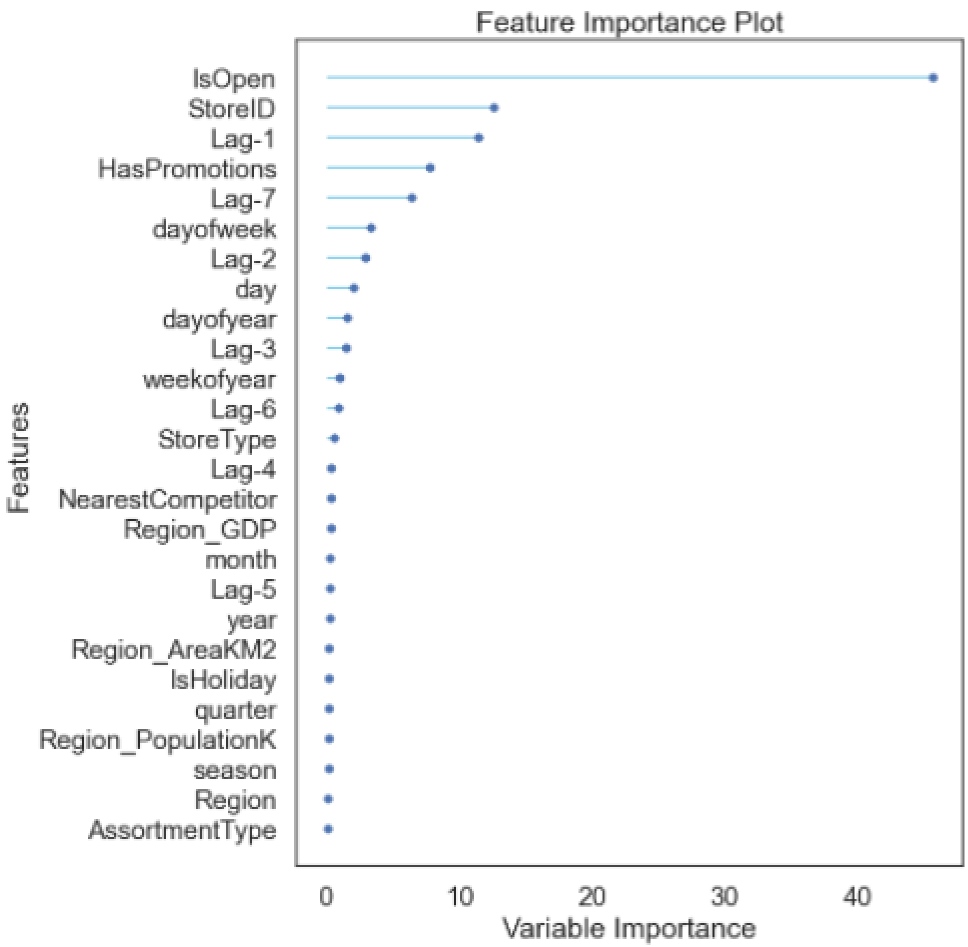
Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	379.0423	442,715.1134	665.3684	0.9418	1.642	0.098
1	327.6249	247,619.4512	497.6138	0.9618	1.6145	0.0922
2	376.0382	371,311.2797	609.3532	0.9488	1.5496	0.0916
3	362.4858	379,072.1718	615.6884	0.9537	1.6078	0.086
4	311.507	221,204.0508	470.3233	0.9673	1.562	0.085
Mean	351.3396	332,384.4134	571.6694	0.9547	1.5952	0.0906
Std	27.0212	84,159.4711	74.6894	0.0091	0.0344	0.0047

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:

Prediction Scores:

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	CatBoost Regressor	338.1213	291,816.7422	540.2006	0.9633	1.5754	0.0861





	Feature	Importance
2	IsOpen	45.6439
0	StoreID	12.5217
19	Lag-1	11.4098
3	HasPromotions	7.7681
25	Lag-7	6.402
14	dayofweek	3.342
20	Lag-2	2.8731
11	day	1.9876
15	dayofyear	1.5458
21	Lag-3	1.455

	Feature	Importance
16	weekofyear	1.0082
24	Lag-6	0.8609
4	StoreType	0.5746
22	Lag-4	0.3373
6	NearestCompetitc	0.3019
9	Region_GDP	0.3006
12	month	0.2701
23	Lag-5	0.2273
13	year	0.2149
8	Region_AreaKM2	0.1833

	Feature	Importance
12	month	0.2701
23	Lag-5	0.2273
13	year	0.2149
8	Region_AreaKM2	0.1833
1	IsHoliday	0.1721
17	quarter	0.1501
10	Region_Populatio	0.1201
18	season	0.1195
7	Region	0.1064
5	AssortmentType	0.1038

This yields an R^2 of 96.3% <u>on the test set</u>! Dataset is saved in file TrainBench_V4.1.csv.