Data Exploration and Inference

Walmart Store Sales Prediction

Haiyu Zhang, Zhouyuan Yuan

1. Introduction: Problem & why it is interesting

Walmart is a popular brand of retail chain stores that provide a large variety of daily life necessities. If it is possible to predict Holiday sales by analyzing the historical data, Walmart would be able to come up with sales' plans that not only give the company more strategic advantage to its competitors, but also better satisfy its customers.

The task we are facing is to predict weekly sales given date, store, department, and additional features. However, there are many different features, both numerical and categorical, that might need to be preprocessed or discarded in order to improve the model performance. In this project, we will perform statistical analysis on dataset features and develop a machine learning model based on the insights gained from our feature analysis.

2. Dataset: Feature analysis

The dataset includes historical sales data for different regions including 45 Walmart stores, each of which contains a number of departments. We will try to predict department-wide sales for each store.

Also, the dataset include promotional markdown events before holidays, including Super Bowl, Labor Day, Thanksgiving and Christmas. The sales predictions on these holidays will be weighted more in evaluation, as Walmart cares more about holiday sales.

2.1 Overall Statistics

The dataset has 421,570 rows and 15 features, which consist of 10 numeric features and 5 categorical features.

The 10 numeric features are: Temperature, Fuel Price, Markdown 1~5, CPI (consumer price index), and Unemployment rate. And the 5 categorical features are Store, Dept, Date, IsHoliday and Type.

Table 1 shows the basic statistics of weekly sales and the 10 numeric features. One thing to notice is that weekly sales have a large standard deviation and range from around -5k to 700k. Also, ~1.3k of them are negative, which is very strange and is likely caused by the refunds. Thus, we would suspect the sales prediction is challenging.

And MarkDowns have lower counts than other features, because most of the markdown values are NaN.

	Weekly_Sales	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Size
count	421570.000000	421570.000000	421570.000000	150681.000000	111248.000000	137091.000000	134967.000000	151432.000000	421570.000000	421570.000000	421570.000000
mean	15981.258123	60.090059	3.361027	7246.420196	3334.628621	1439.421384	3383.168256	4628.975079	171.201947	7.960289	136727.915739
std	22711.183519	18.447931	0.458515	8291.221345	9475.357325	9623.078290	6292.384031	5962.887455	39.159276	1.863296	60980.583328
min	-4988.940000	-2.060000	2.472000	0.270000	-265.760000	-29.100000	0.220000	135.160000	126.064000	3.879000	34875.000000
25%	2079.650000	46.680000	2.933000	2240.270000	41.600000	5.080000	504.220000	1878.440000	132.022667	6.891000	93638.000000
50%	7612.030000	62.090000	3.452000	5347.450000	192.000000	24.600000	1481.310000	3359.450000	182.318780	7.866000	140167.000000
75%	20205.852500	74.280000	3.738000	9210.900000	1926.940000	103.990000	3595.040000	5563.800000	212.416993	8.572000	202505.000000
max	693099.360000	100.140000	4.468000	88646.760000	104519.540000	141630.610000	67474.850000	108519.280000	227.232807	14.313000	219622.000000

Table 1

As for categorical features, Store ID ranges from 1-45, Department ID ranges from 1-99, Date ranges from 2010-2-5 to 2012-10-26, IsHoliday is in [True, False], and Type is in [A, B, C]. Each bar in Figure 1 corresponds to how departments are distributed in each store.

The data consist of 51% of Type A, 39% of Type B and 10% of Type C. And 93% of the data is not holiday and the other 7% is.

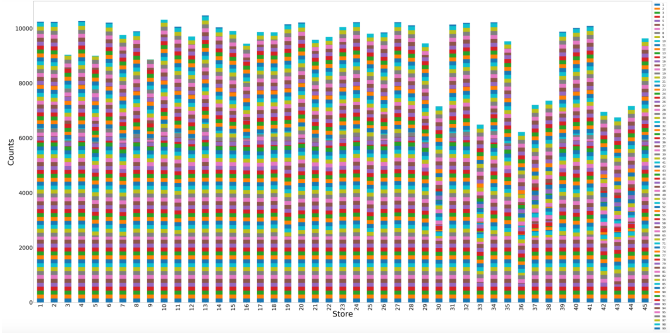


Figure 1

2.2 Categorical Feature: Relation with Weekly Sales

In this subsection, we investigate how weekly sales are distributed under each categorical feature.

As shown in Figure 2a, weekly sales is dependent on different types of store, as Type A has noticeable higher sales. And from Figure 2b, we can find that the data exists copious outliers which make the prediction task more challenging.

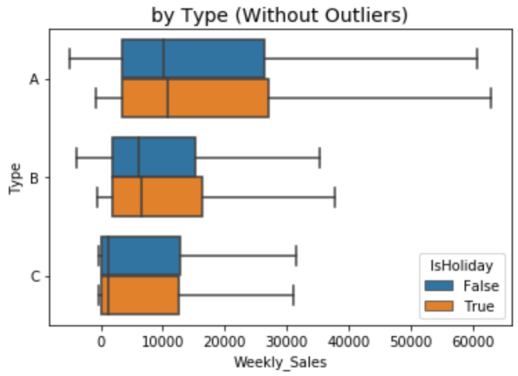


Figure 2a

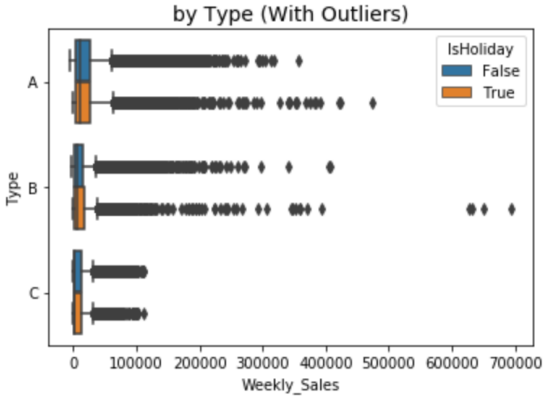


Figure 2b

As shown in Figure 3a, we can see that weekly sales are highly dependent on which types of Stores and Departments are. Thus, Store and department are two key features that we must add to our model. Also, as shown in Figure 3b at Appendix, the outliers are wild in this data so that simply regression would do poorly with the existence of these outliers.

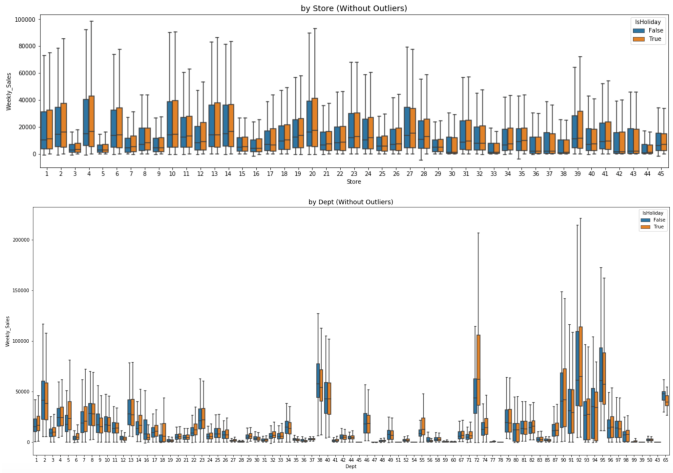


Figure 3a

Similarly Figure 4a shows month, year and week's relation with weekly sales. Although year does not show much differences, as month and week goes near the Christmas season, the weekly sales increase.

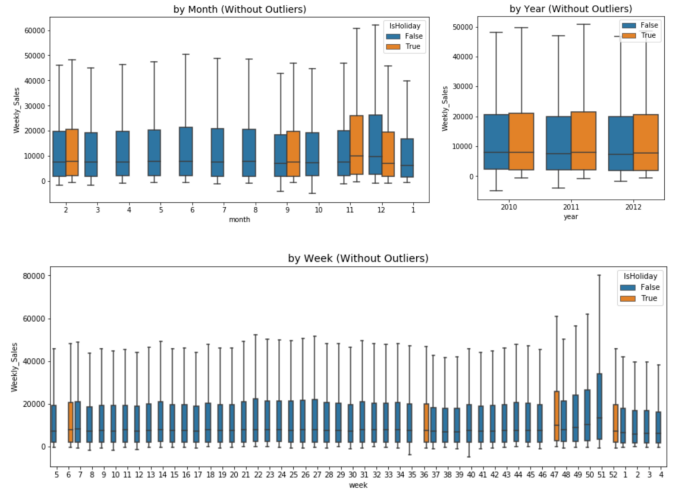


Figure 4a

2.3 Numeric Feature: Correlation & Chi-Squared Test

Correlation Heatmap (Figure 5) is a popular plot used to help selecting relevant features. Since we are predicting weekly sales, we are only interested in the last row or column in the plot. And among all numeric features, size is the most correlated feature with weekly sales.

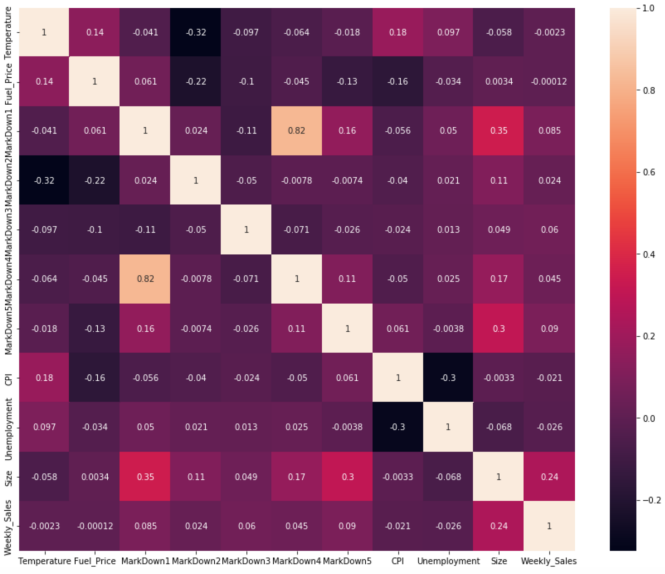


Figure 5

We also explore some further topics in the hypothesis test about the independence by chi-square test of good fitness. We do this test in the very first step in order to choose the correct model to analyze the data and predict the future data.

To be simple, the chi-square test compare the observed data with the expected data according to our hypothesized distribution under null hypothesis:

$$H_0: two\ \ random\ variables\ are\ independent$$

and get the relative difference:

$$\frac{(O-E)^2}{E}$$

where O is the observed value and E is the expected value.

And according to chi-square test theory, the sum of all of these value for each bin should be in a chi-square distribution

$$\frac{(O-E)^2}{E} \sim \chi_d^2$$

, where d is the degree of freedom, by which with chi-square value, we can get the p-value. If the p-value is very small, we reject the null hypothesis and accept the alternative hypothesis that the values are dependent on each other.

For continuous value, we classify the value into 3 different ranges by percentile to make it discrete. Here is the result:

Feature name	Chi-square value	P-value
Dept	320018	0
Store	48597	0
Size	47640.3	0
Type	22457.8	0
MarkDown5	5071.61	0
MarkDown1	4349.62	0
MarkDown3	1648.91	0
week	1574.32	3.08173e-262
MarkDown4	1410.66	3.37059e-304
CPI	749.66	6.14442e-161
Unemployment	679.53	9.42736e-146
MarkDown2	316.086	3.66748e-67
Temperature	227.275	5.09444e-48
month	139.271	4.04375e-29
year	139.271	4.04375e-29
Fuel_Price	112.397	2.24213e-23
IsHoliday	28.5711	6.24974e-07

Table 2

According to the result, we can conclude that almost all features are not independent from the weekly sale, which means we cannot exclude any features at all.

3. Model

According to the result, we can conclude that almost all features are not independent from the weekly sale, which means we cannot exclude any features at all. However, according to the data analysis and visualization in several previous sections, the relationship between those features and weekly sales is too complicated to deal with any regular models. That is the reason why we need to adapt machine learning algorithms.

Thus, we choose XGboost as our model since it is good at dealing with this type of complicated relation and multiple features.

In eXtreme Gradient Boosting (XGboost) algorithm, we try to combine a group of weak learner linearly to form a strong classifier for the classification problem:

$$H(\vec{x}) = \frac{1}{n}\sum_{t=1}^{T}\alpha_t h_t(\vec{x})$$

In term of a loss function to evaluate the overall performance over the whole available dataset by the loss function as:

$$l(H) = \frac{1}{n}\sum_{i=1}^{n} l(H(\vec{x_i}), y_i)$$

where y_i is the ground truth label.

Our goal is to tune the coefficient \alpha_t to get the best classifier:

$$H^* = \underset{H}{\operatorname{argmin}}\, l(H) = \underset{H}{\operatorname{argmin}}\, l(H)$$

In order to do this, we employ a gradient descent algorithm to do it. Since, we have the approximation when h is small:

$$l(H + \alpha h) \approx l(H) + \alpha \nabla l(H) * \vec{h} + \frac{1}{2}\alpha^2 \vec{h}^T * \nabla^2 l(H) * \vec{h}]$$

where

$$l(H) = \sum_{i=1}^{n} l(H(x_i)) = l(H(x_1), ..., H(x_n))$$

thus, we can get

$$l(H + \alpha h) \approx l(H) + \sum_{i=1}^{n}[\alpha \frac{\partial l}{\partial [H(\vec{x_i})]} h(x_i) + \frac{\alpha^2}{2}\frac{\partial^2 l}{\partial [H(\vec{x_i})]^2} h^2(x_i)]$$

, which is the total loss. So we can do gradient descent by choose

$$h_{t+1} = \underset{h}{\operatorname{argmin}} \sum_{i=1}^{n}[\frac{\partial l}{\partial [H(\vec{x_i})]} h(x_i) + \frac{1}{2}\frac{\partial^2 l}{\partial [H(\vec{x_i})]^2} h^2(x_i)]$$

$$h_{t+1} = \underset{h}{\operatorname{argmin}} \sum_{i=1}^{n} \frac{1}{2}\frac{\partial^2 l}{\partial [H(\vec{x_i})]^2}[2\frac{\frac{\partial l}{\partial [H(\vec{x_i})]}}{\frac{\partial^2 l}{\partial [H(\vec{x_i})]^2}} h(x_i) + h^2(x_i)]$$

since the argmin is about h, we can get:

$$h_{t+1} = \underset{h}{\operatorname{argmin}} \sum_{i=1}^{n} \frac{1}{2}\frac{\partial^2 l}{\partial [H(\vec{x_i})]^2}[\frac{\frac{\partial l}{\partial [H(\vec{x_i})]}}{\frac{\partial^2 l}{\partial [H(\vec{x_i})]^2}} + h(x_i)]^2$$

then add to H, multiplied by learning rate alpha:

$$H_{t+1} = H_t + \alpha * h_{t+1}$$

4. Implementation & Result

4.1 Evaluation Metrics

To evaluate the mode, we use three metrics: R-Squared, MAE (Mean Absolute Error), WMAE (Weighted Mean Absolute Error).

R-Squared value ranges from 0-1.0 or 0-100%. Higher the R-squared value, better the performance. R-squared is calculated as 1-RSS/TSS, where RSS is the sum of squared errors and TSS is (n-1)*sample variance of the ground truth values.

MAE is just the mean of the absolute errors between prediction values and ground truth values.

WMAE, used by Walmart, is a weighted version of MAE. It is calculated as the formula:

$\frac{1}{\sum w_i}\sum w_i|y_i - \hat{y}_i|$, where w_i = 5 if isHoliday is True, otherwise 1.

4.2 Data Preprocessing

We have two files: train.csv and test.csv. Since Walmart does not provide Weekly Sales for test.csv, we can not analyze our model performance on test.csv thoroughly. Thus, we decided to only use the data in train.csv. We do an 80/20 train/test split on the data of train.csv.

For numeric features, we fill MarkDown's NaN values as 0, and keep other values the same as the original.

For categorical data, we use one-hot encoding for Year, Type and IsHoliday which only have 8 unique values total.

However, for Store, Dept, Month, and Week, we decided to keep them as numeric, meaning that XGBoost

will do value < threshold split, instead of value == category on these 4 features. Here are the reasons why:

I. If one-hot-encoding is used, our model will run incredibly slow due to limited computation power.

II. Since XGBoost is a non-linear model, encoding with numeric integers does not seem to have influence in this dataset. We experimented with running the model using Type, IsHoliday, Store, Dept, and Week with max_depth=14, n_estimatos=300. It turns out the performances with and without one-hot-encoding is within 0.9% difference in MAE, 1% difference in WMAE and 0.03% difference in R-squared at test set.

4.3 Hyper-parameter tuning

Firstly, we took all the features in accordance with the result from the chi-square independence test with a max depth of 10 and 300 weaker learners . It reaches WMAE of 1154 and 1587 in training data and evaluation data respectively. And the importance of each feature used by the XGboost is as following:

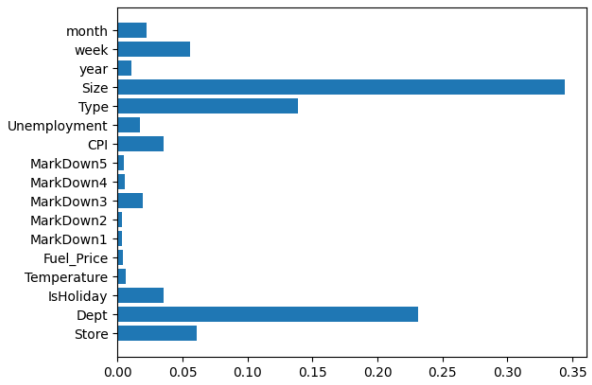


Figure 6

The result is relatively consistent with the Chi-square test that Size, Type, Dept, Store rank as the top features for the result in both independent test and XGboost test.

However, there is also some contrast between these two results. MarkDown5, MarkDown1, MarkDown3, week are considered not independent but turn out to be not important by XGBoost. IsHoliday is considered most independent with weekly sales but turns out to be more important than many other features.

For tuning the features, we first chose these most important features by XGboost, namely 'Store', 'Dept', 'Size', 'week', 'Type', 'Date', 'year', ‘isHoliday’, reaching a WMAE performance of 1126 and 1426 which is better than all the features.

Then according to the result of the chi-square independent test, we dropped the most dependent feature, ‘isHoliday’ and got a performance of 1143 and 1427. It seemed that the chi-square independence test was not accurate but when we tested it on the Kaggle leaderboard. The former had a WMAE performance of 2874 which is slightly worse than the latter, 2867. Also both pairs of these parameters worked worse on the leaderboard than on local data, training or validation, which was a sign of overfitting. According to the result, we might conclude that the chi-square independence test might reduce overfitting to some extent.
Next, we tuned the parameter depth.

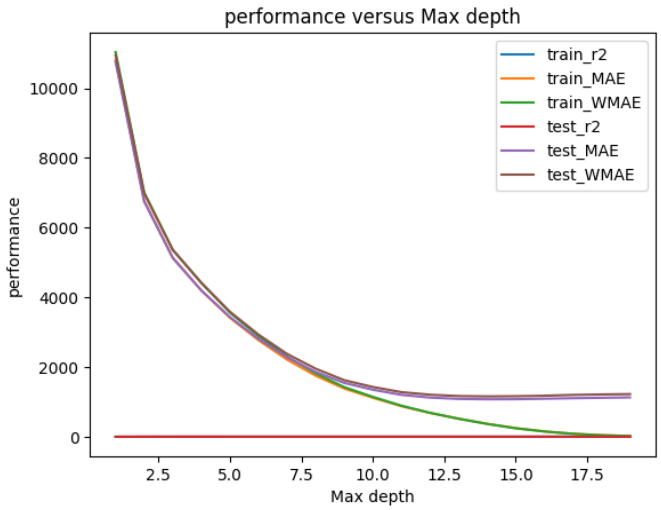


Figure 7

We searched and found that the best depth by WMAE on the evaluation dataset is 14.

Last, we tuned the number of weaker learners. As the program involves huge tons of computation, we start by roughly searching by step size of 40 then fine searching by step size of 1.

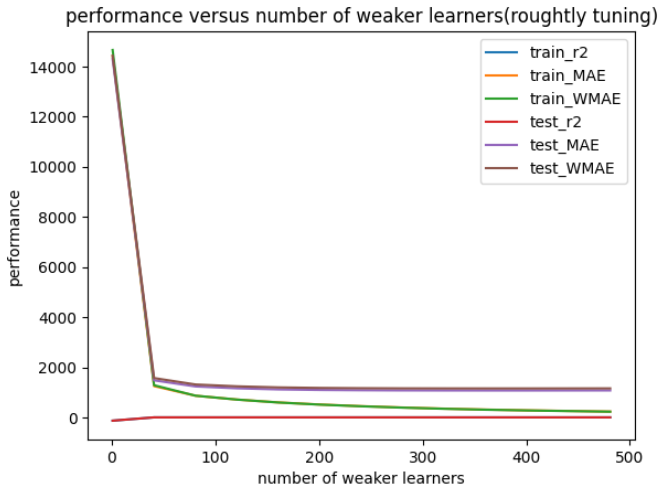


Figure 8

From this rough searching, we determine that the best number of weaker learners is between 361 and 401.

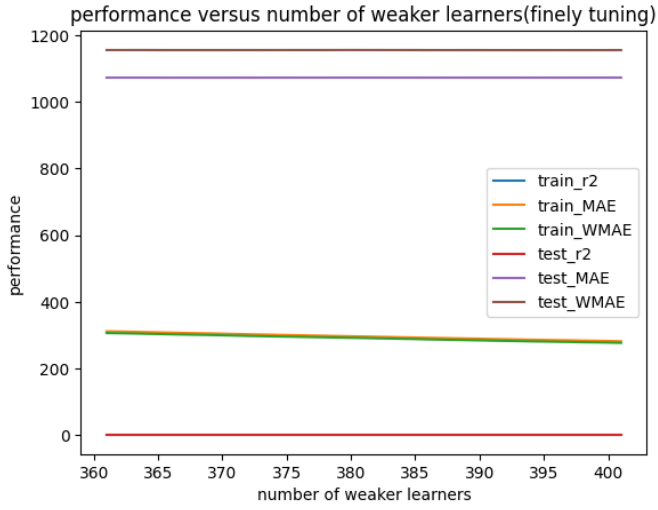


Figure 9

Then we finely searched it within that range, we got the best number of learners as 394.

4.4 Final Model Result

Thus, according to the parameter tuning above, we choose the features as 'Store', 'Dept', 'Size', 'week', 'Type', 'Date', 'year' with Max_depth of 14 and 394 weaker learners. With those settings, we reached a WMAE performance of 281 and 1155 on the training and evaluation dataset respectively. The importance of each feature by XGboost was shown below.

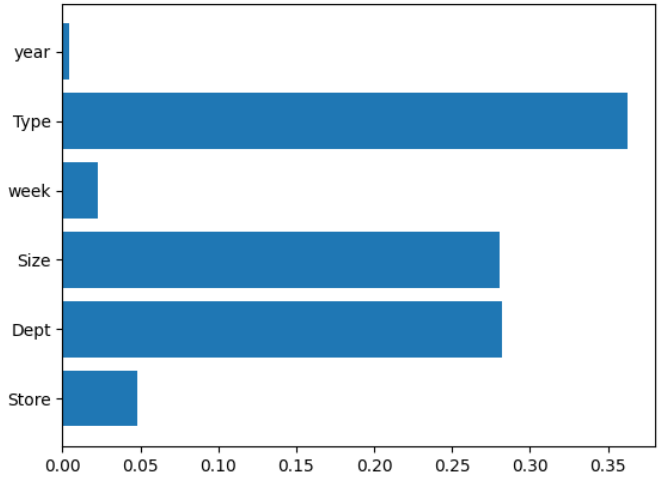


Figure 10

5. Conclusion

We concluded that the main features that affected the prediction by XGboost are 'Store', 'Dept', 'Size', 'week', 'Type', 'Date', 'year', ‘isHoliday’. Also, Chi-square independence test slightly helps reduce the overfitting issues on our XGboost by dropping the most independent feature, IsHoliday. This is amazing as it is counter-intuitive but supported by the evaluation score on the leaderboard. By carefully tuning the parameters, we chose Max_depth of 14 and 394 weaker learners and reached a WMAE performance of 281 and 1155 on the training and evaluation dataset respectively.

References

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939785

XGBoost Documentation. XGBoost Documentation - Xgboost 1.7.1 Documentation, https://xgboost.readthedocs.io/en/stable/index.html#.

Walmart. (2014). Walmart Recruiting - Store Sales Forecasting. Retrieved from https://www.kaggle.com/competitions/walmart-recruiting-store-sales-forecasting/overview/description

Pearson, K. (1900). X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *50*(302), 157–175. https://doi.org/10.1080/14786440009463897

Appendix

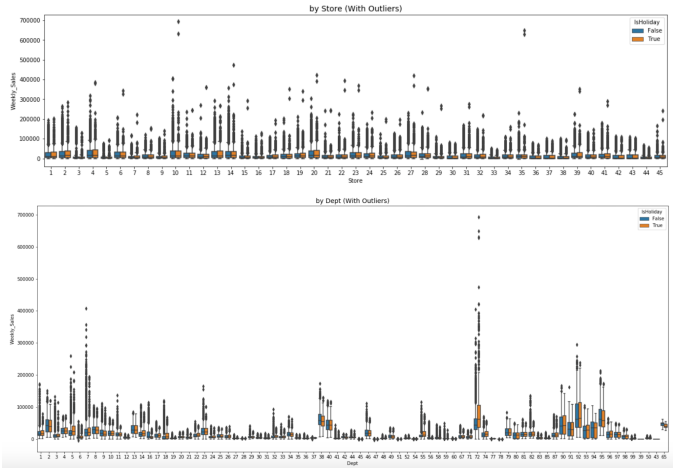


Figure 3b

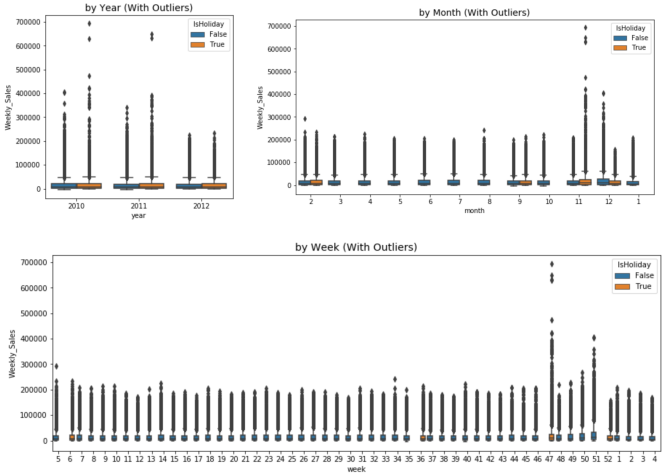


Figure 4b

Code (ipynb Version):

https://github.com/zhouyuanyuan1999/walmart/blob/eda/datasets.ipynb
https://github.com/zhouyuanyuan1999/walmart/blob/tuning/indepdence%20test.ipynb
https://github.com/zhouyuanyuan1999/walmart/blob/tuning/walmart_tuning.ipynb

EDA (Exploratory Data Analysis on Walmart dataset)

In [178…

```
import pandas as pd
import math

from sklearn.preprocessing import LabelEncoder
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

df_train = pd.read_csv('./csv/train.csv')
df_store = pd.read_csv('./csv/stores.csv')
df_feat = pd.read_csv('./csv/features.csv')
```

In [179…

```
df_train['Date'] = pd.to_datetime(df_train['Date'])
df_feat['Date'] = pd.to_datetime(df_feat['Date'])
```

In [180…

```
df = pd.merge(df_train, df_feat, on=['Store', 'Date', 'IsHoliday'], how='left' )
df = pd.merge(df, df_store, on=['Store'], how='left' )
df
```

Out[180]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDov
0	1	1	2010-02-05	24924.50	False	42.31	2.572	NaN	N
1	1	1	2010-02-12	46039.49	True	38.51	2.548	NaN	N
2	1	1	2010-02-19	41595.55	False	39.93	2.514	NaN	N
3	1	1	2010-02-26	19403.54	False	46.63	2.561	NaN	N
4	1	1	2010-03-05	21827.90	False	46.50	2.625	NaN	N
...	...	...	...	...	...	...	...	...	
421565	45	98	2012-09-28	508.37	False	64.88	3.997	4556.61	2(
421566	45	98	2012-10-05	628.10	False	64.89	3.985	5046.74	N
421567	45	98	2012-10-12	1061.02	False	54.47	4.000	1956.28	N
421568	45	98	2012-10-19	760.01	False	56.47	3.969	2004.02	N
421569	45	98	2012-10-26	1076.80	False	58.85	3.882	4018.91	5(

421570 rows × 16 columns

In [181…
```
df['year'] = df['Date'].dt.year
df['week'] = df['Date'].dt.week
df['month'] = df['Date'].dt.month
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: FutureWarning: Serie
s.dt.weekofyear and Series.dt.week have been deprecated.  Please use Series.dt.isocal
endar().week instead.
```

In [182…
```
num_feat = ['Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkD
            'CPI', 'Unemployment', 'Size']
cat_feat = ['Store', 'Dept', 'IsHoliday', 'Type', 'year', 'month', 'week']
target = ['Weekly_Sales']
```

Stats and Feature Counts

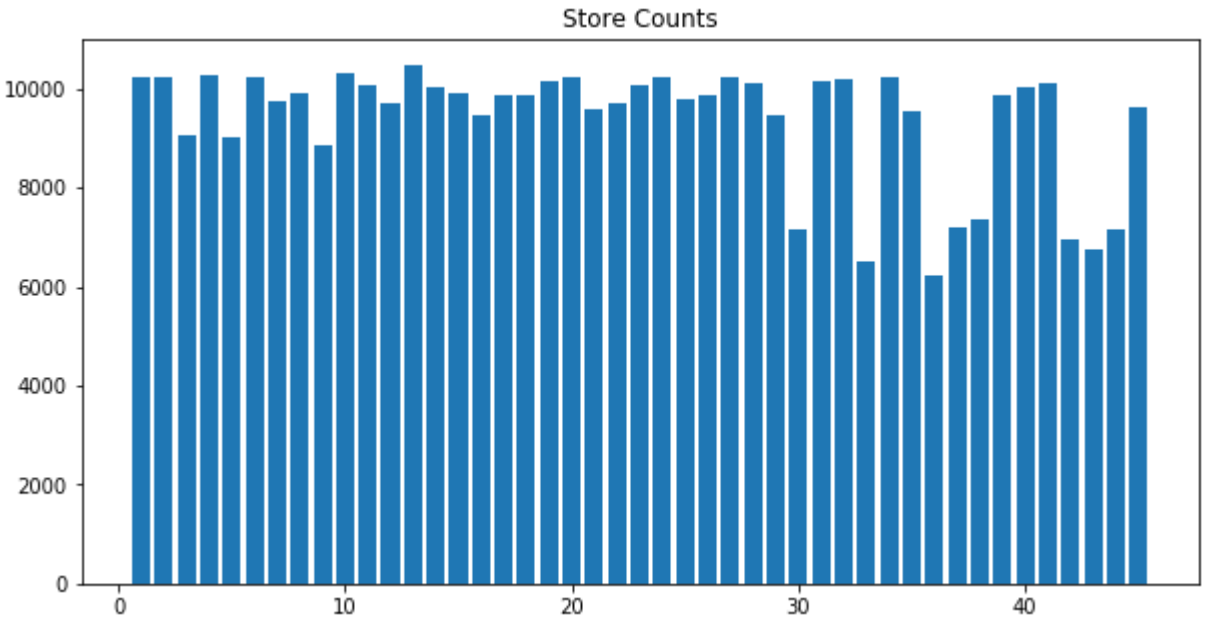
In [183…
```
df[target + num_feat].describe()
```

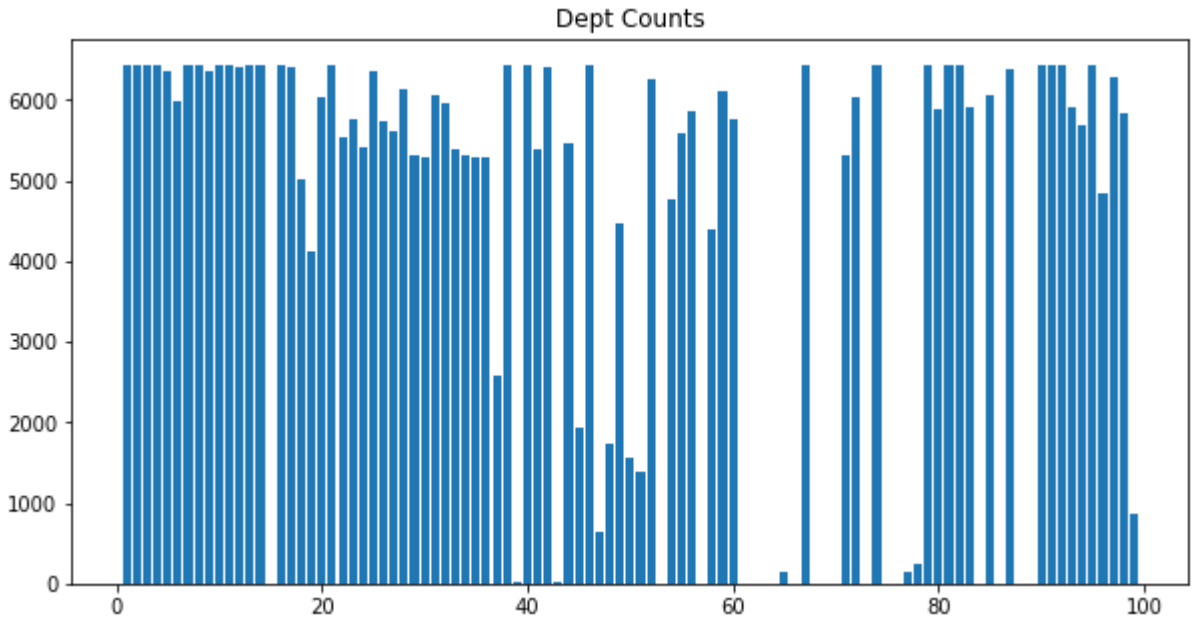
Out[183]:

	Weekly_Sales	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	M
count	421570.000000	421570.000000	421570.000000	150681.000000	111248.000000	137091.000000	13
mean	15981.258123	60.090059	3.361027	7246.420196	3334.628621	1439.421384	
std	22711.183519	18.447931	0.458515	8291.221345	9475.357325	9623.078290	
min	-4988.940000	-2.060000	2.472000	0.270000	-265.760000	-29.100000	
25%	2079.650000	46.680000	2.933000	2240.270000	41.600000	5.080000	
50%	7612.030000	62.090000	3.452000	5347.450000	192.000000	24.600000	
75%	20205.852500	74.280000	3.738000	9210.900000	1926.940000	103.990000	
max	693099.360000	100.140000	4.468000	88646.760000	104519.540000	141630.610000	6

In [184…

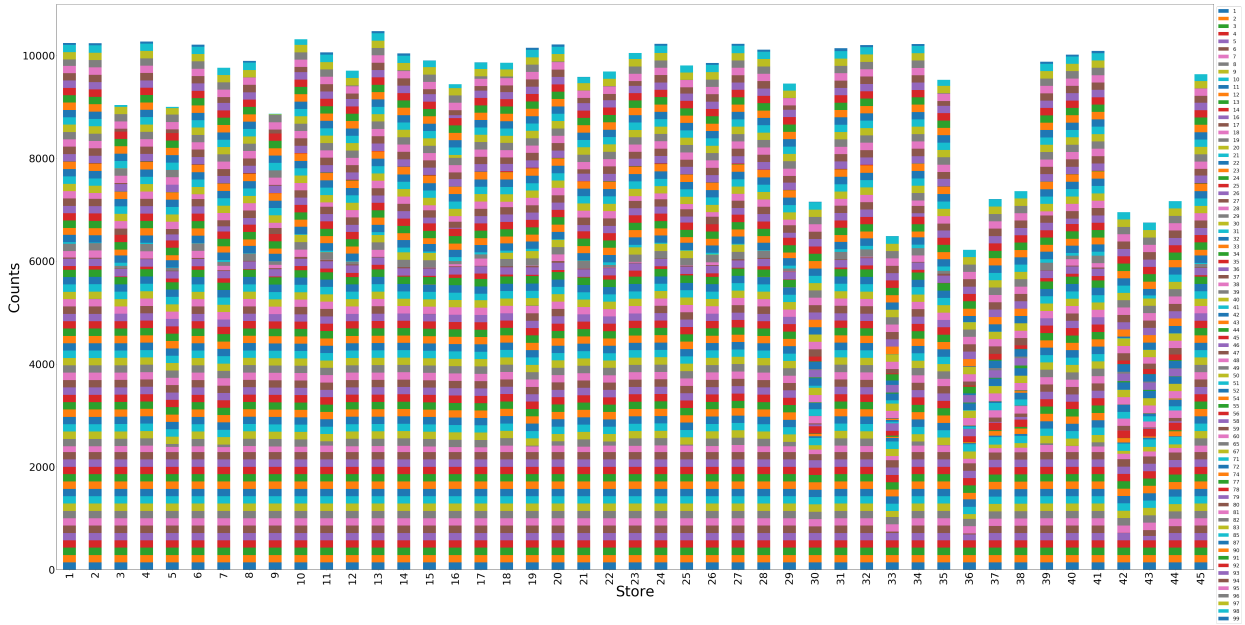
```
plt.figure(figsize=(10, 5))
plt.title('Store Counts')
plt.bar(df['Store'].value_counts().index, np.array( df['Store'].value_counts()))
plt.show()

plt.figure(figsize=(10, 5))
plt.title('Dept Counts')
plt.bar(df['Dept'].value_counts().index, np.array( df['Dept'].value_counts()))
plt.show()
```



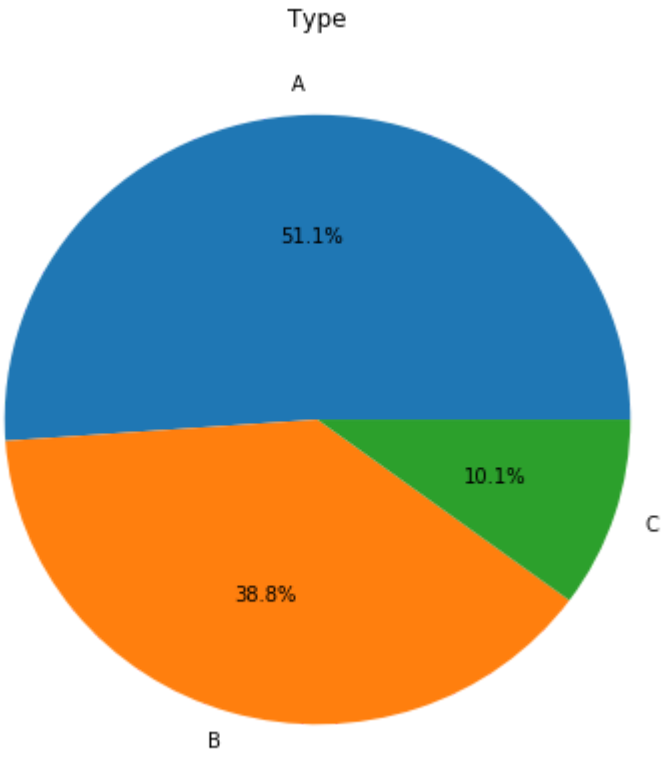


In [185…
```
pivot = df.pivot_table(index=['Store'], columns='Dept', aggfunc='size', fill_value=0)
cols = np.array(pivot.columns)
tmp = pd.DataFrame(pivot.to_numpy(), columns = cols)
tmp['Store'] = np.array( pivot.index )
ax = tmp.plot.bar(x='Store', stacked=True, figsize=(100,50), fontsize=50)
ax.legend(bbox_to_anchor=(1, 1), fontsize=25)
ax.set_xlabel('Store', fontsize=70)
ax.set_ylabel('Counts', fontsize=70)
```

Out[185]:
```
Text(0,0.5,'Counts')
```

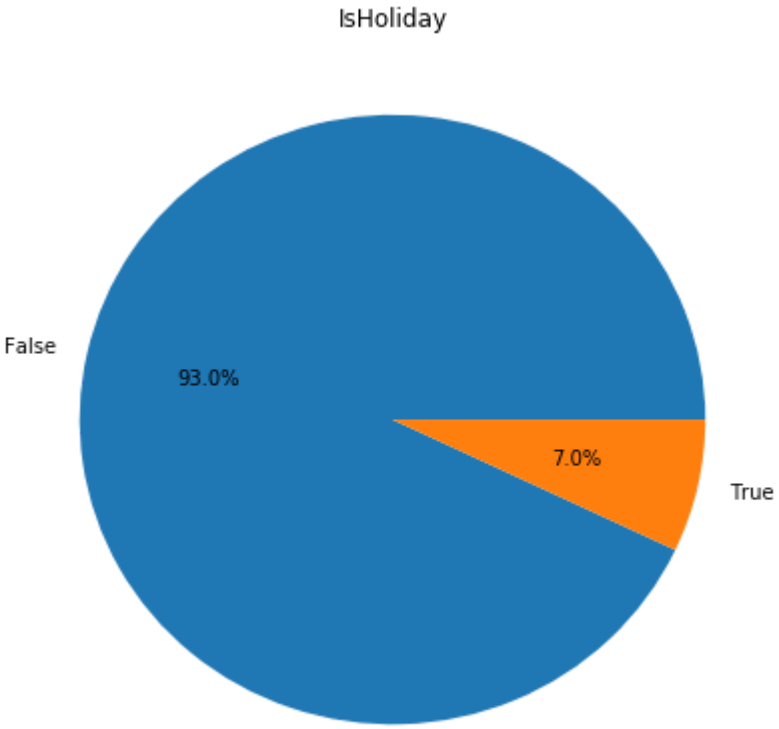


In [186…
```
plt.figure(figsize=(7, 7))
plt.title('Type')
plt.pie(x=df['Type'].value_counts(), labels=df['Type'].value_counts().index,  autopct=
plt.show()
```



In [187…

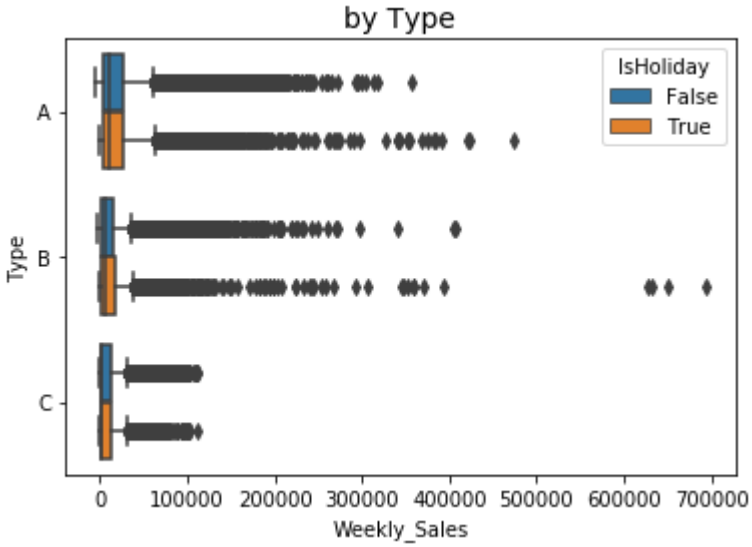
```
plt.figure(figsize=(7, 7))
plt.title('IsHoliday')
plt.pie(x=df['IsHoliday'].value_counts(), labels=['False', 'True'],  autopct="%1.1f%%"
plt.show()
```

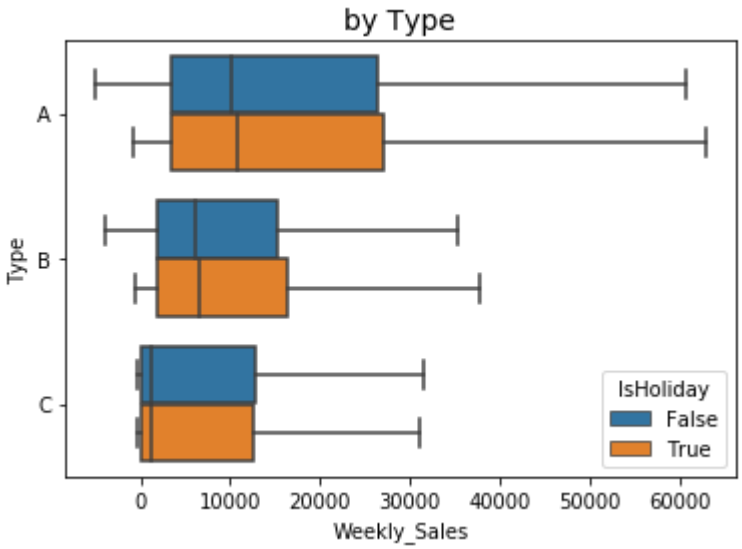


Categorical Feature: Relation with Weekly_Sales

In [188…

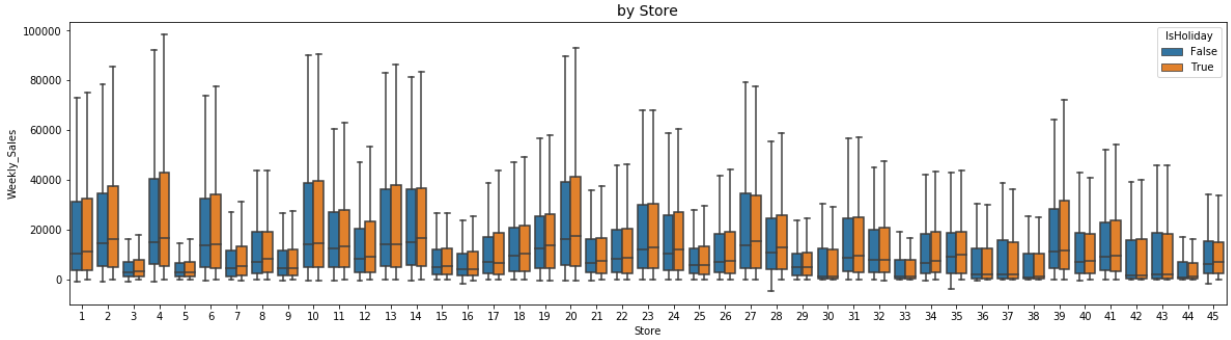
```
ax = sns.boxplot(y=df['Type'], x=df['Weekly_Sales'], hue=df['IsHoliday'], showfliers =
ax.set_title('by Type', size=14)
plt.show()

ax = sns.boxplot(y=df['Type'], x=df['Weekly_Sales'], hue=df['IsHoliday'], showfliers =
ax.set_title('by Type', size=14)
plt.show()
```

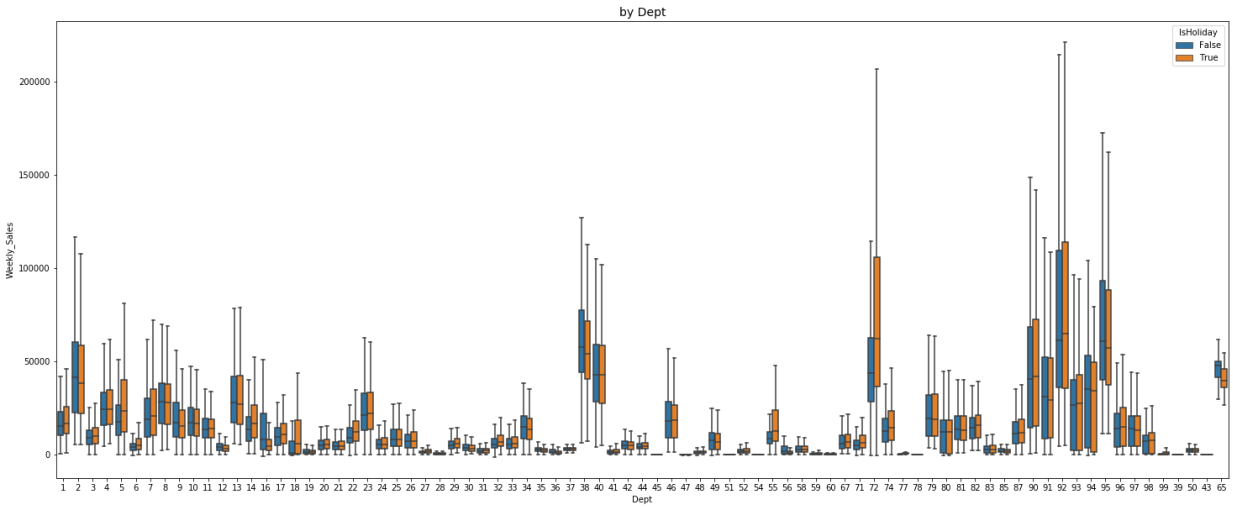




In [189…

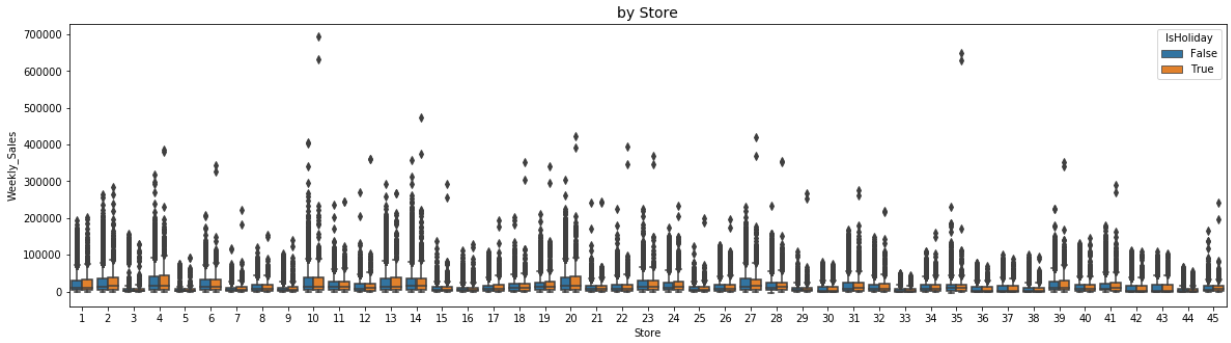
```
plt.figure(figsize=(20,5))
ax = sns.boxplot(x=df['Store'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Store', size=14)
plt.show()

plt.figure(figsize=(25,10))
ax = sns.boxplot(x=df['Dept'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Dept', size=14)
plt.show()
```

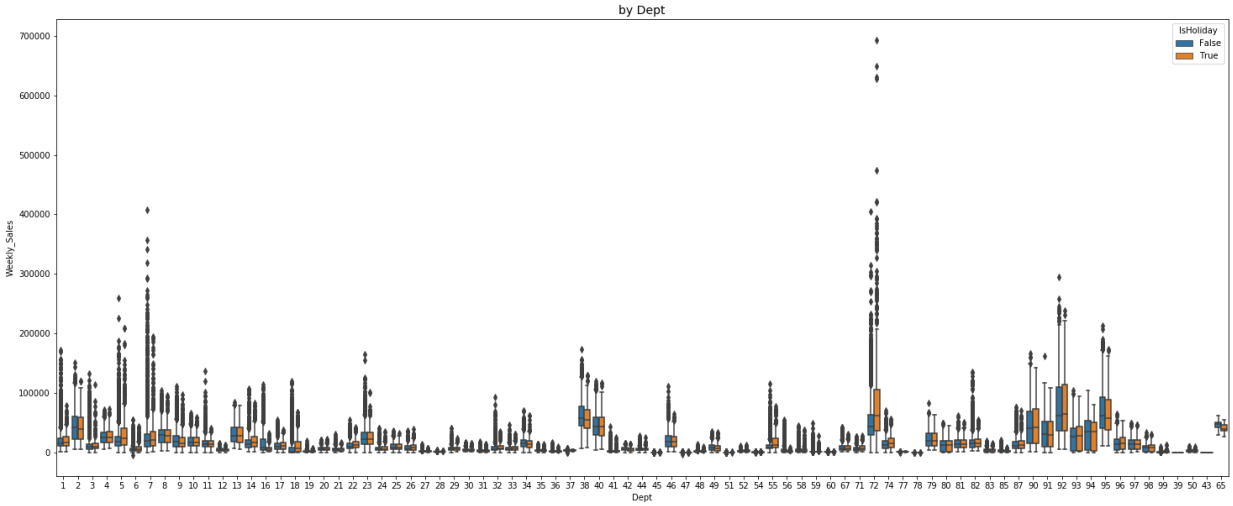




In [190…

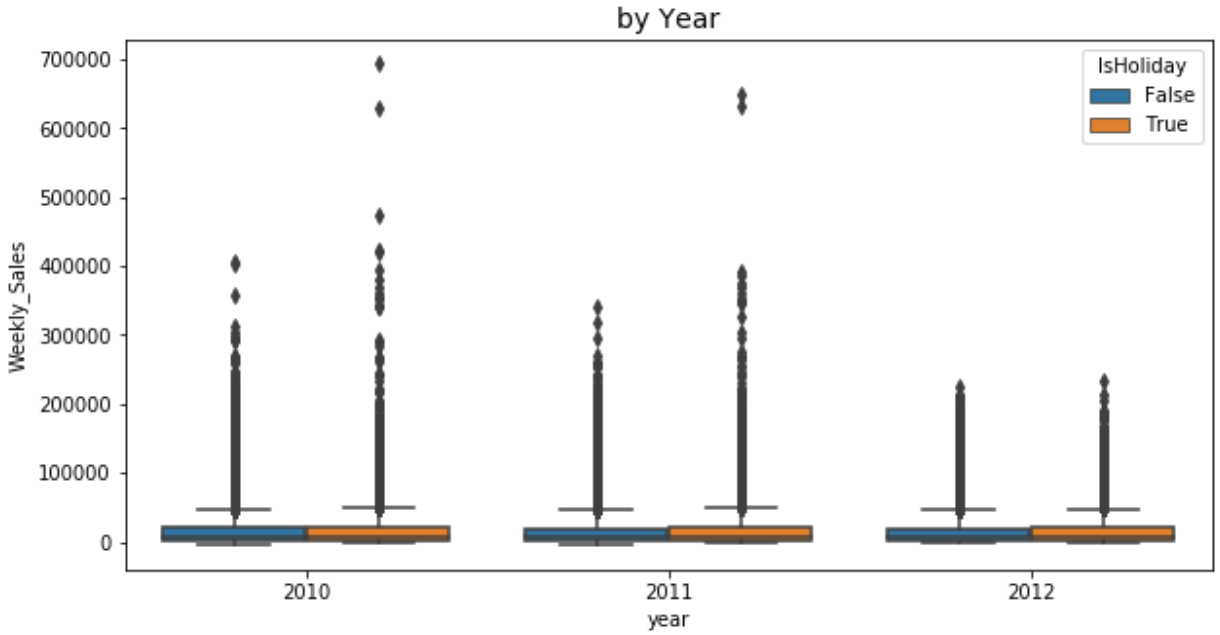
```
plt.figure(figsize=(20,5))
ax = sns.boxplot(x=df['Store'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Store', size=14)
plt.show()

plt.figure(figsize=(25,10))
ax = sns.boxplot(x=df['Dept'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Dept', size=14)
plt.show()
```

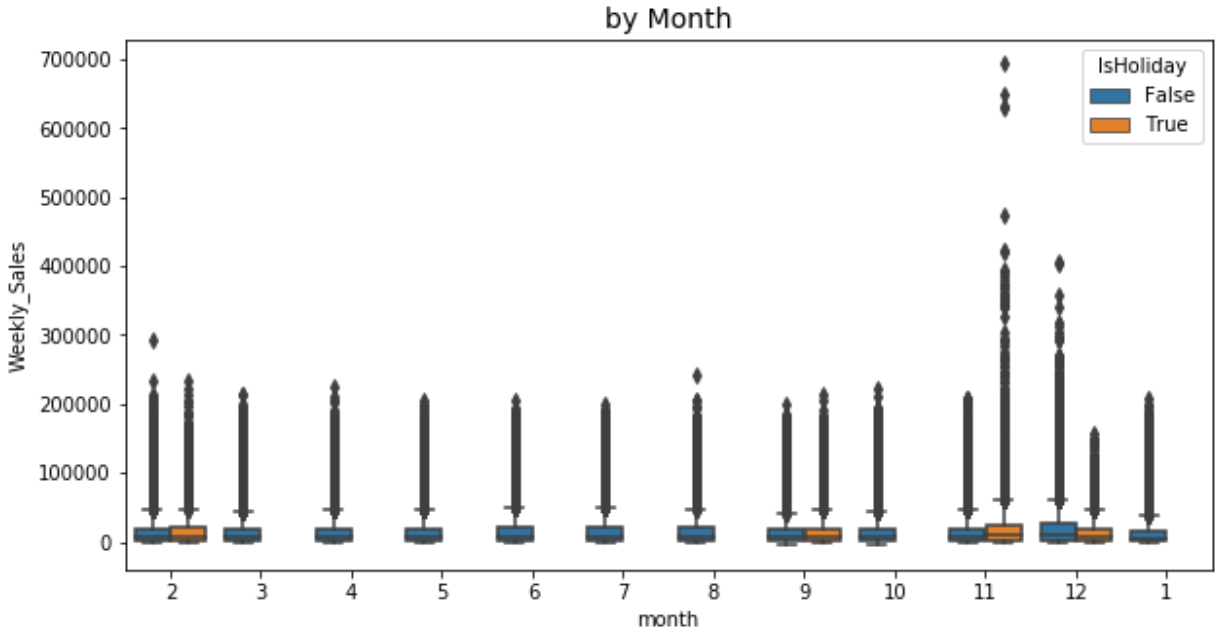


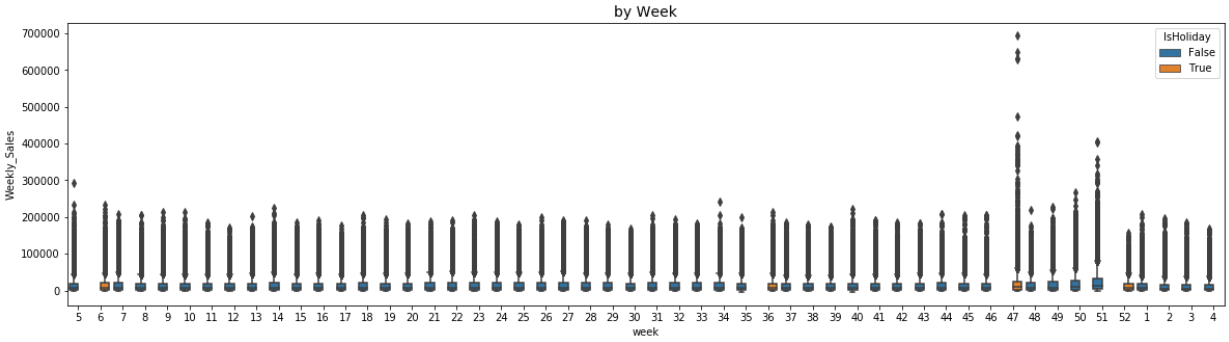


In [191…

```
plt.figure(figsize=(10,5))
ax = sns.boxplot(x=df['year'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Year', size=14)
plt.show()

plt.figure(figsize=(10,5))
ax = sns.boxplot(x=df['month'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Month', size=14)
plt.show()

plt.figure(figsize=(20,5))
ax = sns.boxplot(x=df['week'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Week', size=14)
plt.show()
```

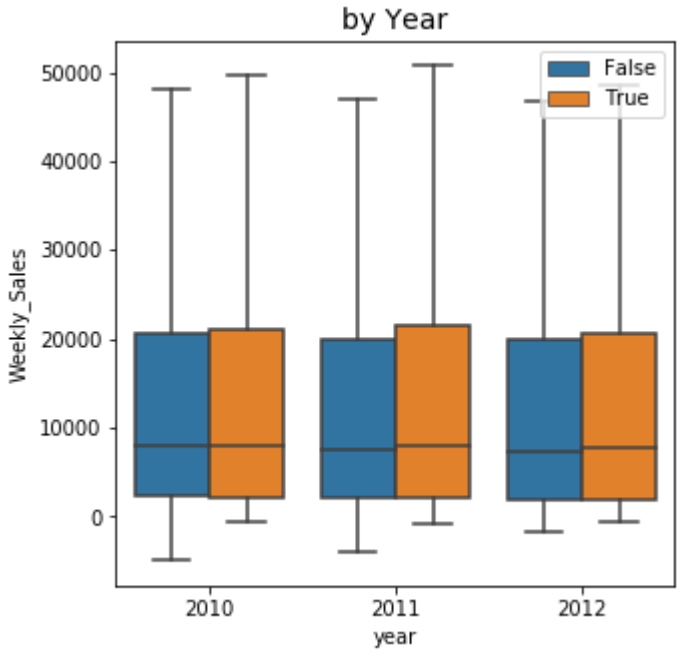


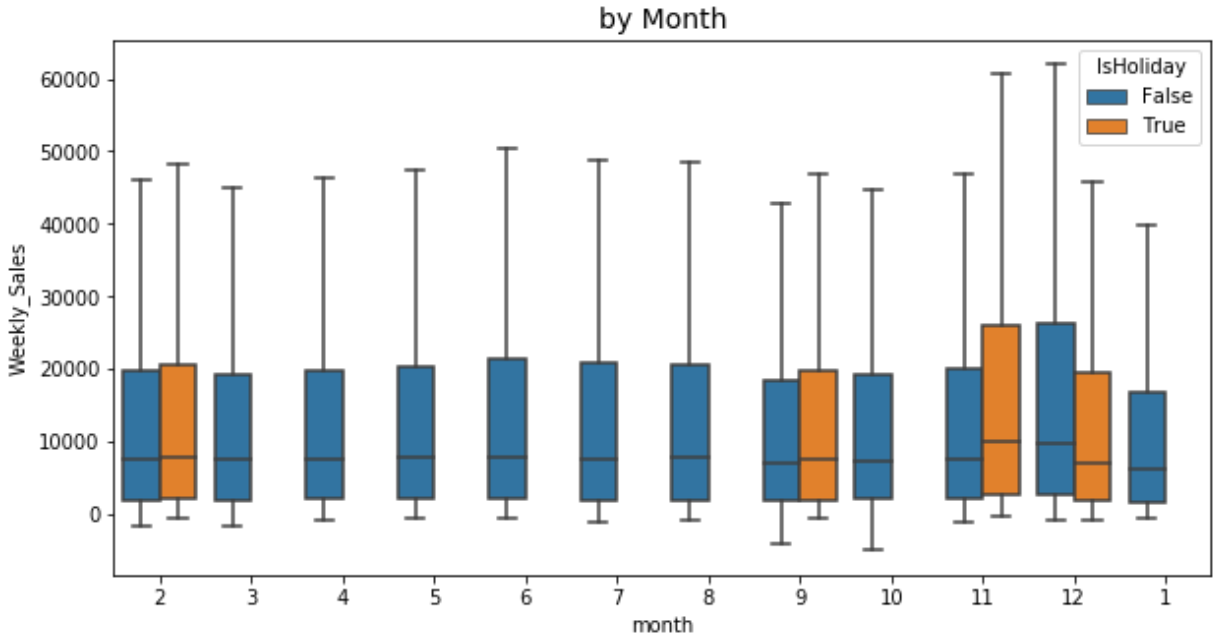


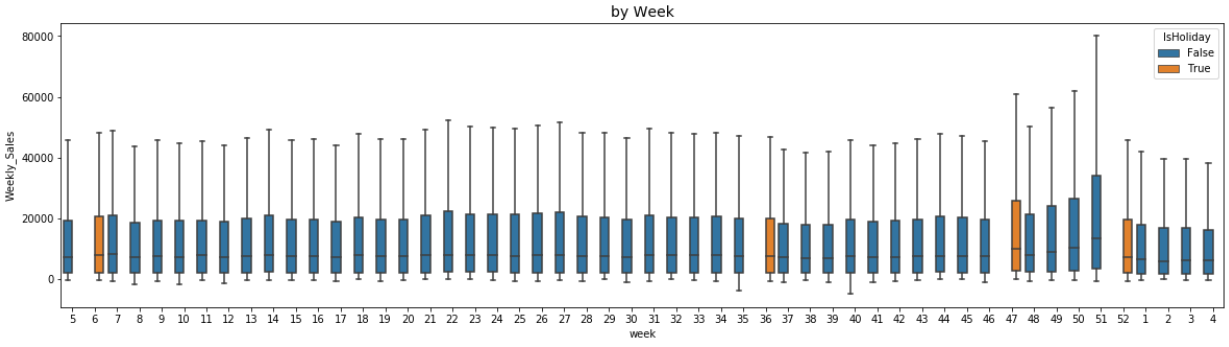


In [192…

```
plt.figure(figsize=(5,5))
ax = sns.boxplot(x=df['year'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Year', size=14)
ax.legend(loc='upper right')
plt.show()

plt.figure(figsize=(10,5))
ax = sns.boxplot(x=df['month'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Month', size=14)
plt.show()

plt.figure(figsize=(20,5))
ax = sns.boxplot(x=df['week'].astype(str), y=df['Weekly_Sales'], hue=df['IsHoliday'],
ax.set_title('by Week', size=14)
plt.show()
```





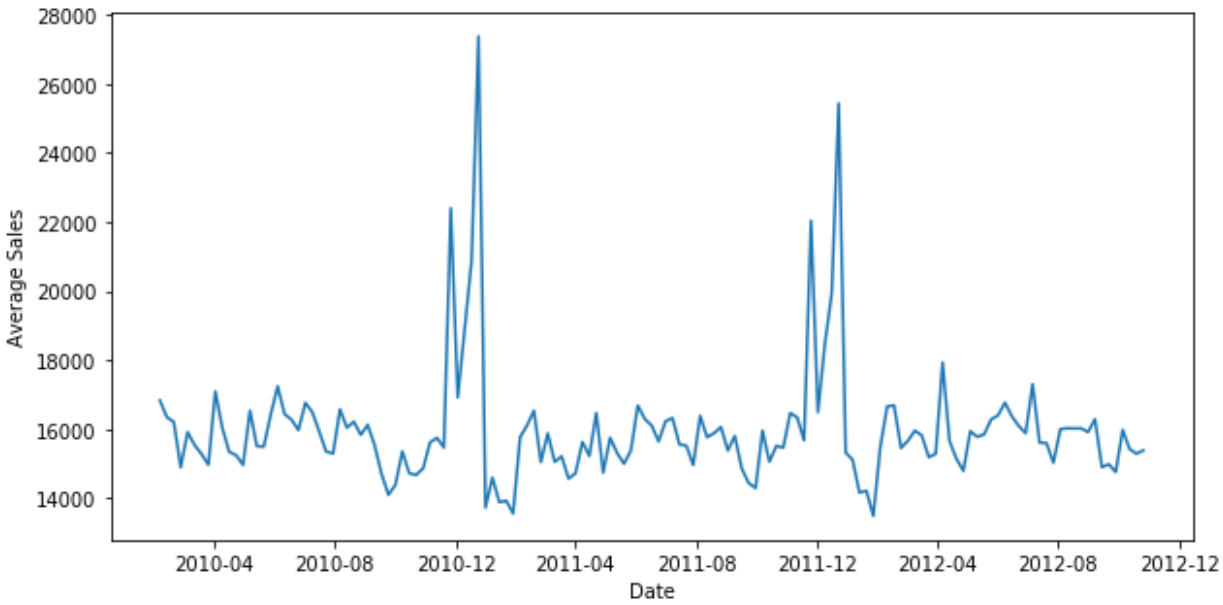


Time Series

In [193…

```
tmp = df.groupby('Date')['Weekly_Sales'].mean()
plt.figure(figsize=(10,5))
```

```
ax = sns.lineplot(y =  tmp, x = tmp.index )
ax.set_ylabel('Average Sales')
plt.show()
```



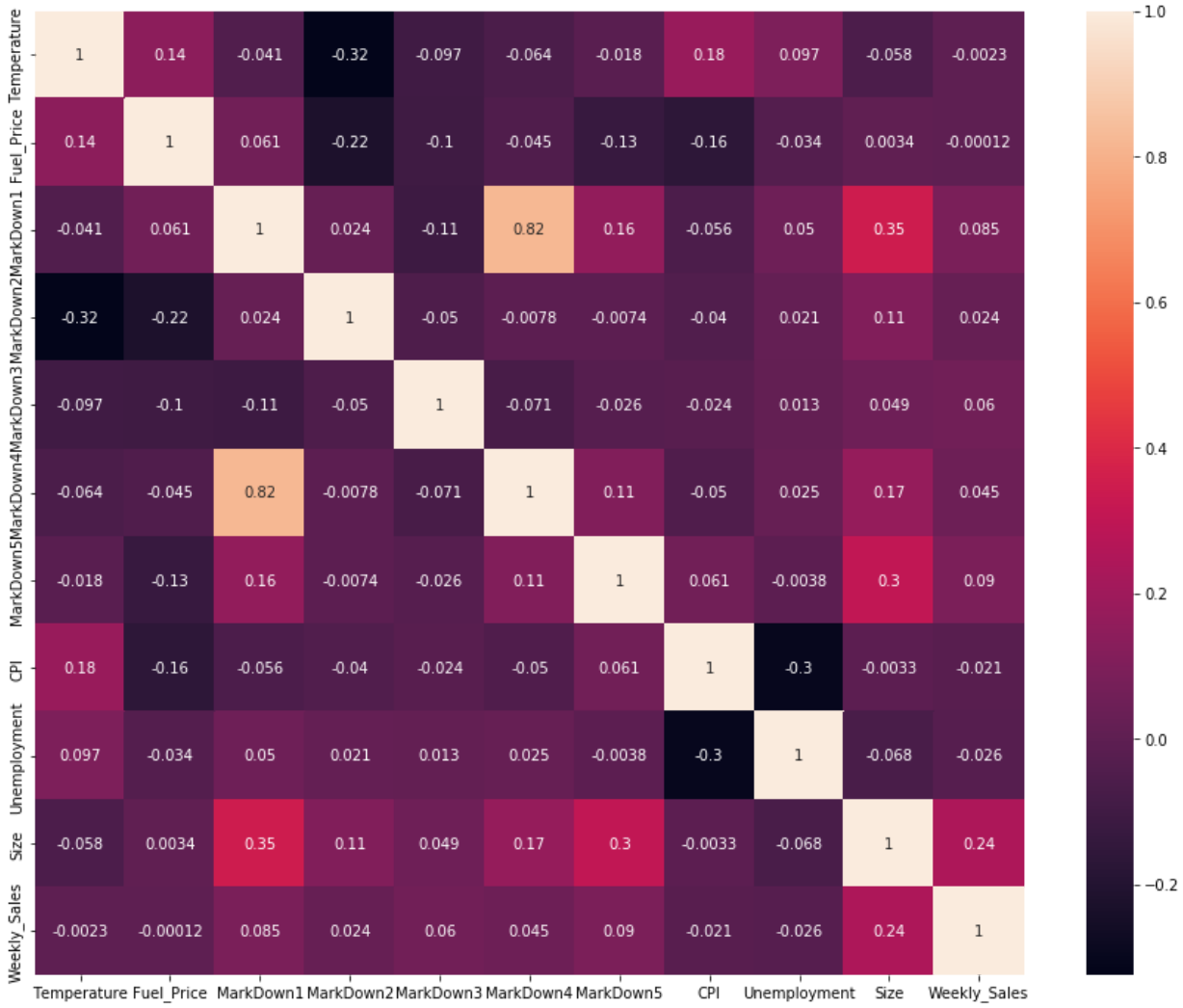
Numeric Feature: Correlations with Weekly Sales

In [203…

```
plt.figure(figsize=(15,12))
sns.heatmap(df[num_feat+target].corr(),annot=True)
```

Out[203]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a6aedaa90>
```


Temperature
Fuel_Price
MarkDown1
MarkDown2
MarkDown3
MarkDown4
MarkDown5
CPI
Unemployment
Size
Weekly_Sales
1 0.14 -0.041 -0.32 -0.097 -0.064 -0.018 0.18 0.097 -0.058 -0.0023
0.14 1 0.061 -0.22 -0.1 -0.045 -0.13 -0.16 -0.034 0.0034 -0.00012
-0.041 0.061 1 0.024 -0.11 0.82 0.16 -0.056 0.05 0.35 0.085
-0.32 -0.22 0.024 1 -0.05 -0.0078 -0.0074 -0.04 0.021 0.11 0.024
-0.097 -0.1 -0.11 -0.05 1 -0.071 -0.026 -0.024 0.013 0.049 0.06
-0.064 -0.045 0.82 -0.0078 -0.071 1 0.11 -0.05 0.025 0.17 0.045
-0.018 -0.13 0.16 -0.0074 -0.026 0.11 1 0.061 -0.0038 0.3 0.09
0.18 -0.16 -0.056 -0.04 -0.024 -0.05 0.061 1 -0.3 -0.0033 -0.021
0.097 -0.034 0.05 0.021 0.013 0.025 -0.0038 -0.3 1 -0.068 -0.026
-0.058 0.0034 0.35 0.11 0.049 0.17 0.3 -0.0033 -0.068 1 0.24
-0.0023 -0.00012 0.085 0.024 0.06 0.045 0.09 -0.021 -0.026 0.24 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

In [ ]:

In [ ]:

In [ ]:

Chi-Squared Test

In [6]:
```
df_sum.to_csv('csv/combined.csv')
```

In [56]:
```
import pandas as pd
import numpy as np
```

In [17]:
```
train = pd.read_csv("csv/train.csv")
stores = pd.read_csv("csv/stores.csv")
features = pd.read_csv("csv/features.csv")
```

In [19]:
```
train["Date"] = pd.to_datetime(train["Date"])
train.dtypes
```

Out[19]:
```
Store                    int64
Dept                     int64
Date            datetime64[ns]
Weekly_Sales           float64
IsHoliday                 bool
dtype: object
```

In [23]:
```
features["Date"] = pd.to_datetime(features["Date"])
features.dtypes
```

Out[23]:
```
Store                    int64
Date            datetime64[ns]
Temperature            float64
Fuel_Price             float64
MarkDown1              float64
MarkDown2              float64
MarkDown3              float64
MarkDown4              float64
MarkDown5              float64
CPI                    float64
Unemployment           float64
IsHoliday                 bool
dtype: object
```

In [24]:
```
df_sum = pd.merge(train, features, on =["Store","Date","IsHoliday"])
df_sum = pd.merge(df_sum, stores, on = ["Store"])
df_sum
```

Out[24]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDov
0	1	1	2010-02-05	24924.50	False	42.31	2.572	NaN	N
1	1	2	2010-02-05	50605.27	False	42.31	2.572	NaN	N
2	1	3	2010-02-05	13740.12	False	42.31	2.572	NaN	N
3	1	4	2010-02-05	39954.04	False	42.31	2.572	NaN	N
4	1	5	2010-02-05	32229.38	False	42.31	2.572	NaN	N
...	...	...	...	...	...	...	...	...	
421565	45	93	2012-10-26	2487.80	False	58.85	3.882	4018.91	58
421566	45	94	2012-10-26	5203.31	False	58.85	3.882	4018.91	58
421567	45	95	2012-10-26	56017.47	False	58.85	3.882	4018.91	58
421568	45	97	2012-10-26	6817.48	False	58.85	3.882	4018.91	58
421569	45	98	2012-10-26	1076.80	False	58.85	3.882	4018.91	58

421570 rows × 16 columns

In [25]:
```
df_sum.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 421570 entries, 0 to 421569
Data columns (total 16 columns):
 #   Column        Non-Null Count   Dtype         
---  ------        --------------   -----         
 0   Store         421570 non-null  int64         
 1   Dept          421570 non-null  int64         
 2   Date          421570 non-null  datetime64[ns]
 3   Weekly_Sales  421570 non-null  float64       
 4   IsHoliday     421570 non-null  bool          
 5   Temperature   421570 non-null  float64       
 6   Fuel_Price    421570 non-null  float64       
 7   MarkDown1     150681 non-null  float64       
 8   MarkDown2     111248 non-null  float64       
 9   MarkDown3     137091 non-null  float64       
 10  MarkDown4     134967 non-null  float64       
 11  MarkDown5     151432 non-null  float64       
 12  CPI           421570 non-null  float64       
 13  Unemployment  421570 non-null  float64       
 14  Type          421570 non-null  object        
 15  Size          421570 non-null  int64         
dtypes: bool(1), datetime64[ns](1), float64(10), int64(3), object(1)
memory usage: 51.9+ MB
```

In [26]:
```
df_sum[df_sum['MarkDown1'].isnull() | df_sum['MarkDown2'].isnull() | df_sum['MarkDown3
# xgboost can run on GPU
# adaboost not GPU package
```

Out[26]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDov
0	1	1	2010-02-05	24924.50	False	42.31	2.572	NaN	N
1	1	2	2010-02-05	50605.27	False	42.31	2.572	NaN	N
2	1	3	2010-02-05	13740.12	False	42.31	2.572	NaN	N
3	1	4	2010-02-05	39954.04	False	42.31	2.572	NaN	N
4	1	5	2010-02-05	32229.38	False	42.31	2.572	NaN	N
...	...	...	...	...	...	...	...	...	
421498	45	93	2012-10-19	2270.50	False	56.47	3.969	2004.02	N
421499	45	94	2012-10-19	4655.65	False	56.47	3.969	2004.02	N
421500	45	95	2012-10-19	48434.97	False	56.47	3.969	2004.02	N
421501	45	97	2012-10-19	5575.90	False	56.47	3.969	2004.02	N
421502	45	98	2012-10-19	760.01	False	56.47	3.969	2004.02	N

324514 rows × 16 columns

In [39]:
```
df_sum["year"] = df_sum["Date"].dt.year
df_sum["month"] = df_sum["Date"].dt.year
df_sum["week"] = df_sum["Date"].dt.week
```

```
/tmp/ipykernel_14774/3342123843.py:3: FutureWarning: Series.dt.weekofyear and Series.
dt.week have been deprecated. Please use Series.dt.isocalendar().week instead.
  df_sum["week"] = df_sum["Date"].dt.week
```

In [40]:
```
df_sum.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 421570 entries, 0 to 421569
Data columns (total 19 columns):
 #   Column        Non-Null Count   Dtype         
---  ------        --------------   -----         
 0   Store         421570 non-null  int64         
 1   Dept          421570 non-null  int64         
 2   Date          421570 non-null  datetime64[ns]
 3   Weekly_Sales  421570 non-null  float64       
 4   IsHoliday     421570 non-null  bool          
 5   Temperature   421570 non-null  float64       
 6   Fuel_Price    421570 non-null  float64       
 7   MarkDown1     150681 non-null  float64       
 8   MarkDown2     111248 non-null  float64       
 9   MarkDown3     137091 non-null  float64       
 10  MarkDown4     134967 non-null  float64       
 11  MarkDown5     151432 non-null  float64       
 12  CPI           421570 non-null  float64       
 13  Unemployment  421570 non-null  float64       
 14  Type          421570 non-null  object        
 15  Size          421570 non-null  int64         
 16  year          421570 non-null  int64         
 17  month         421570 non-null  int64         
 18  week          421570 non-null  int64         
dtypes: bool(1), datetime64[ns](1), float64(10), int64(6), object(1)
memory usage: 61.5+ MB
```

In [178…

```python
# for different value type, we have different method to make it discrete
def is_discrete(col_name, df=df_sum ):
    first = df[col_name].iloc[0]
    if isinstance(first,str):
        return 0 # 0--discrete, 1--continue, -1-- just skip
    elif isinstance(first,np.bool_):
        return 0
    elif isinstance(first,np.int64):
        if (len(df[col_name].unique())>100):
            return 1
        else:
            return 0
    elif isinstance(first,np.float64):
        return 1
    elif isinstance(first,pd._libs.tslibs.timestamps.Timestamp):
        return -1

    print("error___",type(first))
    assert False
```

In [179…

```python
for col_name in df_sum.columns:
    print(is_discrete(col_name))
```

```
0
0
-1
1
0
1
1
1
1
1
1
1
1
1
0
0
0
0
0
```

In [224…

```
def percent_range(col_name, df_=df_sum, N = 3):
    percent = [(i+1) * (1.0/N) for i in range(N-1)]
    x_percentile = df_[col_name].describe(percentiles=percent)

    if N == 3:
        x_percentile = x_percentile.drop(index=['50%'])

    ranges = x_percentile[3:].values
    return ranges
ranges = percent_range("Weekly_Sales")
ranges
```

Out[224]:

```
array([ -4988.94,   3521.78,  14599.76, 693099.36])
```

In [226…

```
from scipy import stats

def indepedence_test(col_name):
    df_tmp = df_sum[['Weekly_Sales', col_name]].dropna()

    # separate to grid
    df_tmp['range'] = pd.cut(df_tmp['Weekly_Sales'], ranges)

    if is_discrete(col_name,df_tmp)==1: # considered as continueous
        # get range
        temp_range = percent_range(col_name, df_tmp) # min, max included
        df_tmp['temp_range'] = pd.cut(df_tmp[col_name], temp_range)
        pivot = df_tmp.pivot_table(index=['range'], columns=['temp_range'], aggfunc='s
    elif is_discrete(col_name,df_tmp)==0: # considered as discrete
        pivot = df_tmp.pivot_table(index=['range'], columns=[col_name], aggfunc='size'
    elif is_discrete(col_name,df_tmp)==-1:
        # just skip
        print(col_name,"   skip")
        return None
    else:
        assert False # not implemented

    cols = np.array(pivot.columns)
    tmp = pd.DataFrame(pivot.to_numpy(), columns = cols, index=np.array( pivot.index )
    # display(tmp)
```

```
    # independence test of chi-square
    res = stats.chi2_contingency(tmp.to_numpy())
    # display(res)
    print(col_name, "    chi(x^2)=", res[0], "    p-value=", res[1])
    return [res[0], res[1]]


res = indepedence_test('MarkDown1')
```

```
MarkDown1     chi(x^2)= 4349.615890310006     p-value= 0.0
```

In [227…

```
independence = []
for col_name in df_sum.columns:
    if col_name != "Weekly_Sales": # not itself
        print(col_name)
        res = indepedence_test(col_name)
        if res is not None:
            independence.append(res+[col_name])
```

```
Store
Store     chi(x^2)= 48596.99536355429     p-value= 0.0
Dept
Dept     chi(x^2)= 320017.66711396724     p-value= 0.0
Date
Date    skip
IsHoliday
IsHoliday     chi(x^2)= 28.571112678394925     p-value= 6.249736555630811e-07
Temperature
Temperature     chi(x^2)= 227.27542294728656     p-value= 5.0944403144010454e-48
Fuel_Price
Fuel_Price     chi(x^2)= 112.3971189664625     p-value= 2.2421301410949232e-23
MarkDown1
MarkDown1     chi(x^2)= 4349.615890310006     p-value= 0.0
MarkDown2
MarkDown2     chi(x^2)= 316.08574153813305     p-value= 3.667478082554465e-67
MarkDown3
MarkDown3     chi(x^2)= 1648.9108706171419     p-value= 0.0
MarkDown4
MarkDown4     chi(x^2)= 1410.6617263063802     p-value= 3.370593661075429e-304
MarkDown5
MarkDown5     chi(x^2)= 5071.608680845734     p-value= 0.0
CPI
CPI     chi(x^2)= 749.659584860178     p-value= 6.144417236613282e-161
Unemployment
Unemployment     chi(x^2)= 679.5300009791179     p-value= 9.427360747507886e-146
Type
Type     chi(x^2)= 22457.84730756763     p-value= 0.0
Size
Size     chi(x^2)= 47640.31984398991     p-value= 0.0
year
year     chi(x^2)= 139.2706501011192     p-value= 4.0437521884291577e-29
month
month     chi(x^2)= 139.2706501011192     p-value= 4.0437521884291577e-29
week
week     chi(x^2)= 1574.3224478878144     p-value= 3.08173423983008e-262
```

In [7]:

```
for i in range(len(independence)):
    tmp = independence[i]
    independence[i] = [tmp[2], tmp[0], tmp[1]]
```

```
independence
```

Out[7]:
```
[['Dept', 320017.66711396724, 0.0],
 ['Store', 48596.99536355429, 0.0],
 ['Size', 47640.31984398991, 0.0],
 ['Type', 22457.84730756763, 0.0],
 ['MarkDown5', 5071.608680845734, 0.0],
 ['MarkDown1', 4349.615890310006, 0.0],
 ['MarkDown3', 1648.9108706171419, 0.0],
 ['week', 1574.3224478878144, 3.08173423983008e-262],
 ['MarkDown4', 1410.6617263063802, 3.370593661075429e-304],
 ['CPI', 749.659584860178, 6.144417236613282e-161],
 ['Unemployment', 679.5300009791179, 9.427360747507886e-146],
 ['MarkDown2', 316.08574153813305, 3.667478082554465e-67],
 ['Temperature', 227.27542294728656, 5.0944403144010454e-48],
 ['month', 139.2706501011192, 4.0437521884291577e-29],
 ['year', 139.2706501011192, 4.0437521884291577e-29],
 ['Fuel_Price', 112.3971189664625, 2.2421301410949232e-23],
 ['IsHoliday', 28.571112678394925, 6.249736555630811e-07]]
```

In [8]:
```
from tabulate import tabulate
table =[["Feature name", "Chi-square value", "P-value"]] + independence
print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
```

Feature name	Chi-square value	P-value
Dept	320018	0
Store	48597	0
Size	47640.3	0
Type	22457.8	0
MarkDown5	5071.61	0
MarkDown1	4349.62	0
MarkDown3	1648.91	0
week	1574.32	3.08173e-262
MarkDown4	1410.66	3.37059e-304
CPI	749.66	6.14442e-161
Unemployment	679.53	9.42736e-146
MarkDown2	316.086	3.66748e-67
Temperature	227.275	5.09444e-48
month	139.271	4.04375e-29
year	139.271	4.04375e-29
Fuel_Price	112.397	2.24213e-23
IsHoliday	28.5711	6.24974e-07

In [ ]:

In [ ]:

XGBoost

In [1]:
```
import pandas as pd
import math

import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
import numpy as np
from xgboost import plot_tree
from sklearn.metrics import r2_score

df_train = pd.read_csv('./csv/train.csv')
df_store = pd.read_csv('./csv/stores.csv')
df_feat = pd.read_csv('./csv/features.csv')
```

In [2]:
```
df_train['Date'] = pd.to_datetime(df_train['Date'])
df_feat['Date'] = pd.to_datetime(df_feat['Date'])
```

In [3]:
```
df = pd.merge(df_train, df_feat, on=['Store', 'Date', 'IsHoliday'], how='left' )
df = pd.merge(df, df_store, on=['Store'], how='left' )
```

In [4]:
```
fill_na_cols = [ 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5' ]

for c in fill_na_cols:
    df[c] = df[c].fillna(0)
```

In [5]:
```
df['year'] = df['Date'].dt.year
df['week'] = df['Date'].dt.week
df['month'] = df['Date'].dt.month

# df = df.drop(columns = ['Date'])
```

```
/tmp/ipykernel_2157/3234748729.py:2: FutureWarning: Series.dt.weekofyear and Series.dt.week have been deprecated. Please use Series.dt.isocalendar().week instead.
  df['week'] = df['Date'].dt.week
```

In [6]:
```
df.head()
```

Out[6]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	M
0	1	1	2010-02-05	24924.50	False	42.31	2.572	0.0	0.0	
1	1	1	2010-02-12	46039.49	True	38.51	2.548	0.0	0.0	
2	1	1	2010-02-19	41595.55	False	39.93	2.514	0.0	0.0	
3	1	1	2010-02-26	19403.54	False	46.63	2.561	0.0	0.0	
4	1	1	2010-03-05	21827.90	False	46.50	2.625	0.0	0.0	

In [7]:
```
label=LabelEncoder() # True-False --> 0-1

cols = ['Type', 'IsHoliday']

for c in cols:
    df[c] = label.fit_transform(df[c])
    df[c] = df[c].astype('category')
```

In [8]:
```
df = df.sample(frac=1, random_state=1) # shuffle
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 421570 entries, 367951 to 128037
Data columns (total 19 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Store         421570 non-null  int64
 1   Dept          421570 non-null  int64
 2   Date          421570 non-null  datetime64[ns]
 3   Weekly_Sales  421570 non-null  float64
 4   IsHoliday     421570 non-null  category
 5   Temperature   421570 non-null  float64
 6   Fuel_Price    421570 non-null  float64
 7   MarkDown1     421570 non-null  float64
 8   MarkDown2     421570 non-null  float64
 9   MarkDown3     421570 non-null  float64
 10  MarkDown4     421570 non-null  float64
 11  MarkDown5     421570 non-null  float64
 12  CPI           421570 non-null  float64
 13  Unemployment  421570 non-null  float64
 14  Type          421570 non-null  category
 15  Size          421570 non-null  int64
 16  year          421570 non-null  int64
 17  week          421570 non-null  int64
 18  month         421570 non-null  int64
dtypes: category(2), datetime64[ns](1), float64(10), int64(6)
memory usage: 58.7 MB
```

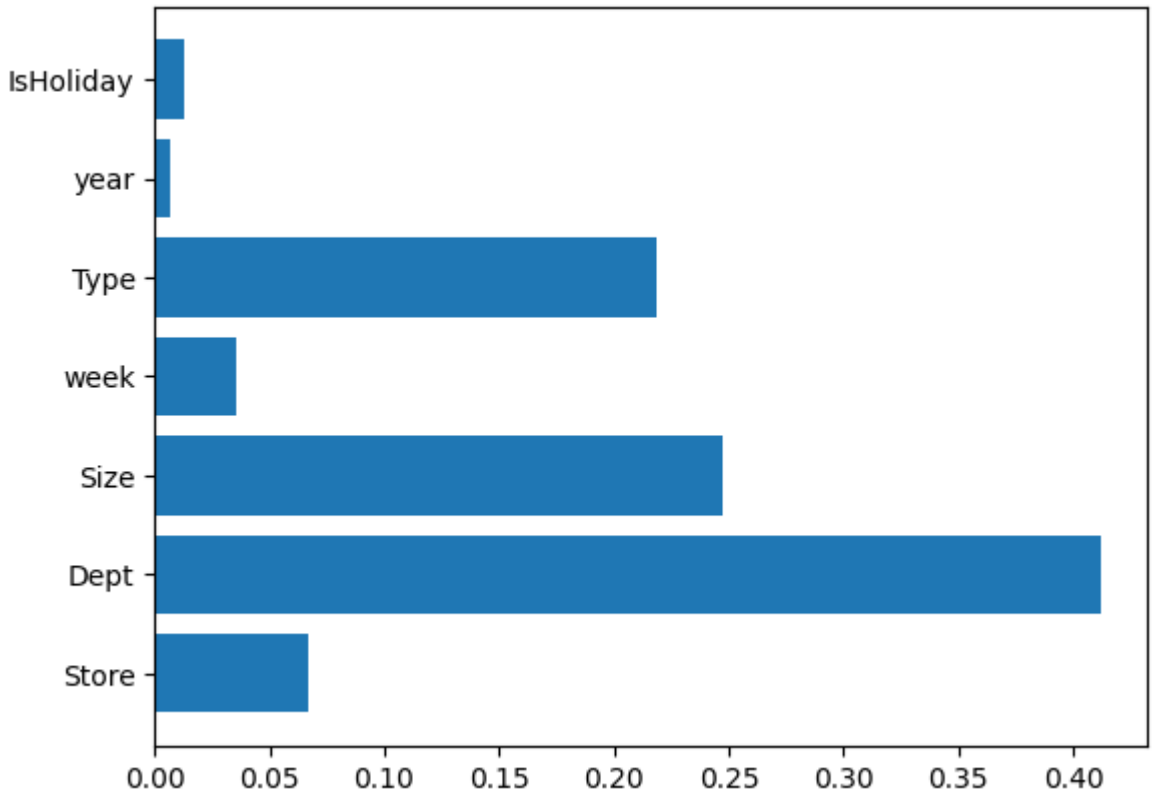
In [81]:
```
import matplotlib.pyplot as plt

def WMAE(pred, gtruth, is_holiday):
    weights = is_holiday.astype('int').apply(lambda x: 5 if x==1 else 1)
    tmp = weights * abs(pred-gtruth)
    return tmp.sum()/weights.sum()

def train_boost(df_t, df_v,
                features = None,
                sumb = False,
                depth = 6,
                N_ = 300):
    # feature selection
    if features is None:
        features = ['Weekly_Sales', 'Store','Dept','IsHoliday','Size','week','Type', '
    # df = df_0[ features ] #
    # split train and evaluation
    """
    train_num = int( 0.8 * len(df) )
    df_train = df[:train_num]
    df_val = df[train_num:]
    """
    df_train = df_t[features]
    df_val = df_v


    print("#(train)=",len(df_train), "#(val)", len(df_val))

    # train
    X_train = df_train.drop(columns = ['Weekly_Sales', 'Date'])
    Y_train = df_train['Weekly_Sales']


    reg = xgb.XGBRegressor(tree_method = 'gpu_hist', objective='reg:squarederror', max
                       enable_categorical=True) # n_estimators -- #(tree)

    reg.fit(X_train, Y_train ) #saved in reg


    # train evaluation
    print("performance on training")
    pred = pd.Series( reg.predict(X_train) ).apply(lambda x: 0 if x < 0 else x )
    #pred = pd.Series( reg.predict(X_train) )

    r2 = r2_score( pred,Y_train.reset_index(drop=True) )
    mae = np.mean( np.absolute( np.array( Y_train.tolist() ) - pred ))
    wmae = WMAE(pred, Y_train.reset_index(drop=True), df_t['IsHoliday'].reset_index(dr

    print("r2= ", r2)
    print("MAE= ", mae)
    print("WMAE= ", wmae)

    # plt.figure(figsize=(15, 10))
    plt.barh(X_train.columns, reg.feature_importances_)
    plt.show()
    # plt.savefig('./tmp.png', dpi=100)


    # test evaluation
```

```
    if not sumb: # not sumbmission, just evaluation
        print("performance on evaluation")
        df_val = df_v[features]
        X_val = df_val.drop(columns = ['Weekly_Sales', 'Date'])
        Y_val = df_val['Weekly_Sales']

        pred = pd.Series( reg.predict(X_val) ).apply(lambda x: 0 if x < 0 else x )
        #pred = pd.Series( reg.predict(X_val) )
        r2_v = r2_score( pred,Y_val.reset_index(drop=True) )
        mae_v = np.mean( np.absolute( np.array( Y_val.tolist() ) - pred ) )
        wmae_v = WMAE(pred, Y_val.reset_index(drop=True),
             df_v['IsHoliday'].reset_index(drop=True))

        print("r2=", r2_v)
        print("MAE=", mae_v)
        print("WMAE=", wmae_v)

        return [r2,mae,wmae,r2_v, mae_v, wmae_v]
    else: # submission

        X_test = df_v[features[1:]].drop(columns = ['Date'])
        df_test_0 = df_v[features[1:]].copy()
        print(df_test_0.columns)
        pred = pd.Series( reg.predict(X_test) ).apply(lambda x: 0 if x < 0 else x )

        df_test_0['Weekly_Sales'] = pred
        def concat(x):
            return str( f"{x['Store']}_{x['Dept']}_{str(x['Date']).split(' ')[0]}" )
        df_test_0['Id'] = df_test_0.apply(lambda x: concat(x), axis=1)

        submission = df_test_0[['Id', 'Weekly_Sales']]
        submission.to_csv('submission.csv', index=False)

    return reg
```

In [73]:
```
# test
train_num = int( 0.8 * len(df) )
df_train = df[:train_num]
df_val = df[train_num:]
feat= ['Weekly_Sales','Store', 'Dept', 'Size', 'week', 'Type', 'Date', 'year'] #, 'IsH
train_boost(df_train, df_val, sumb = False, depth= 10, N_ = 300,
           features = feat )
```

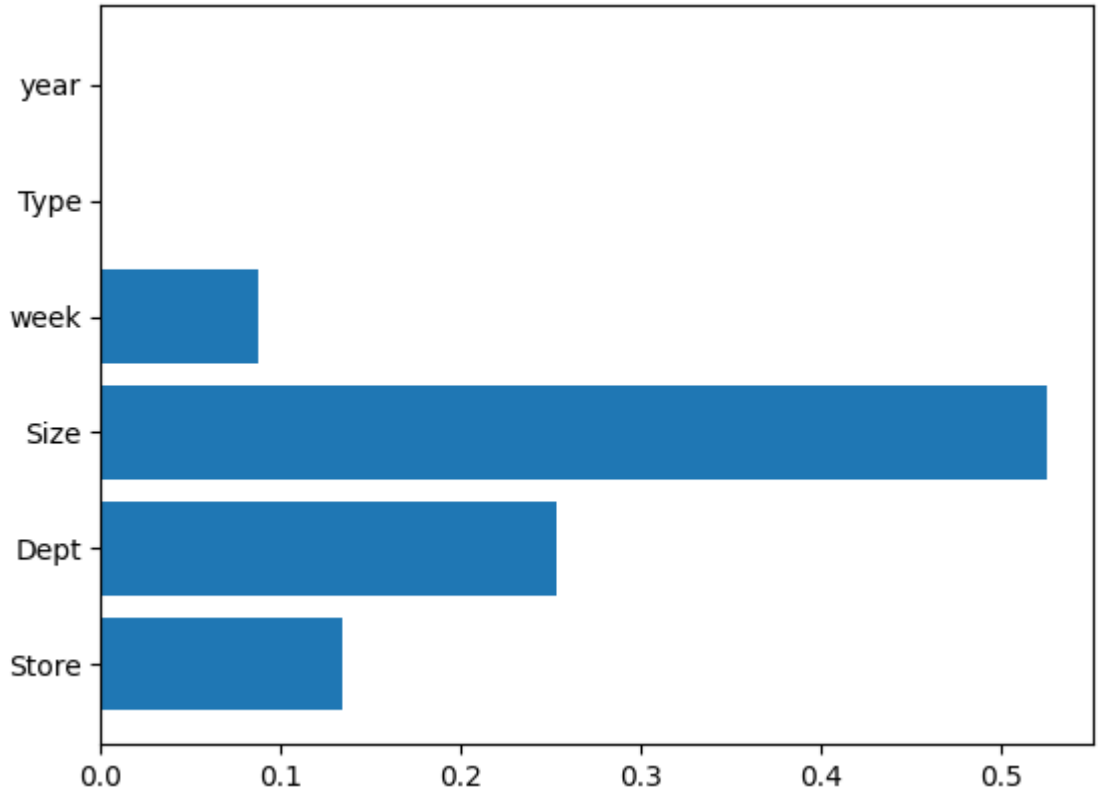
```
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9921148482837642
MAE=  1114.1361855614055
WMAE=  1126.8791124204918
performance on evaluation
r2= 0.9852344047533437
MAE= 1348.5775160095172
WMAE= 1426.541863817565
```

Out[73]:
```
[[0.9921148482837642, 1114.1361855614055, 1126.8791124204918],
 [0.9852344047533437, 1348.5775160095172, 1426.541863817565]]
```

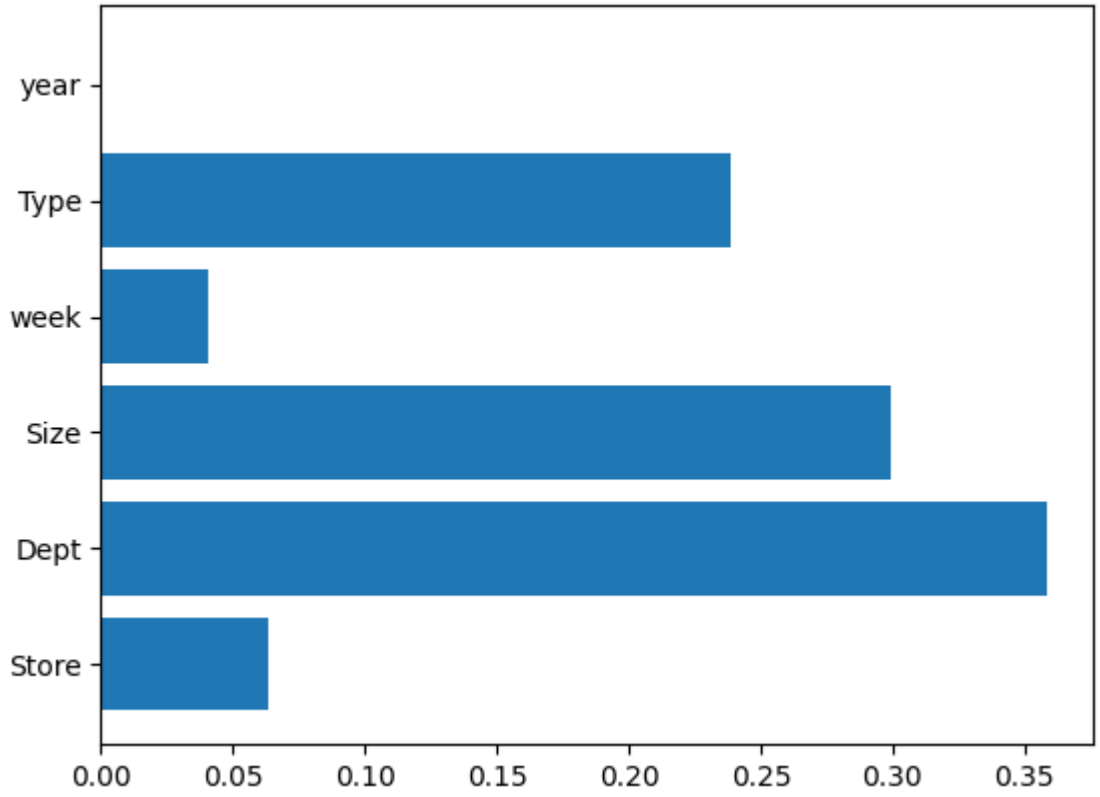


In [89]:
```
d_max = 20
feat= ['Weekly_Sales','Store', 'Dept', 'Size', 'week', 'Type', 'Date', 'year'] #, 'IsH
res = {}
serie = [[] for i in range(6)]
for d in range(1,d_max):
    print('============================')
    print("depth=",d)
    tmp = train_boost(df_train, df_val, sumb = False, depth= d, N_ = 300, features = f
    res[d] = tmp
    for i in range(6):
        serie[i].append(tmp[i])
```

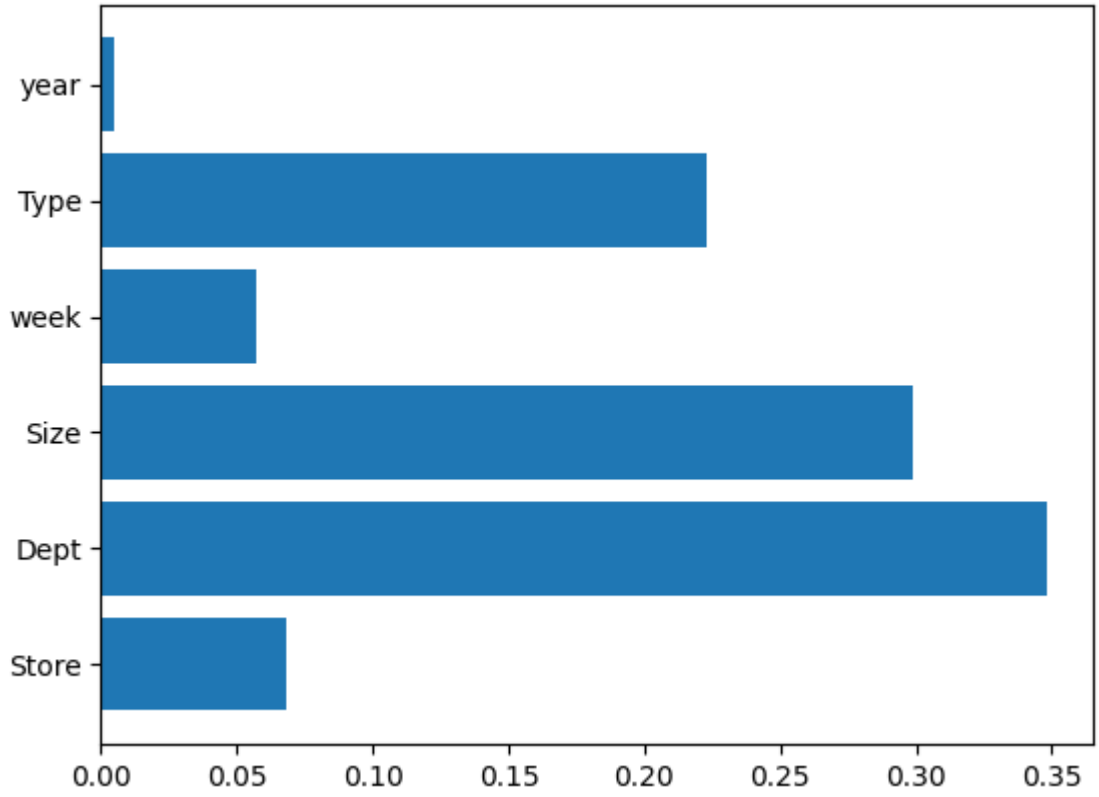
```
============================
depth= 1
#(train)= 337256 #(val) 84314
performance on training
r2=  -1.6021988477811813
MAE=  10807.40243662833
WMAE=  11039.43109064034
```



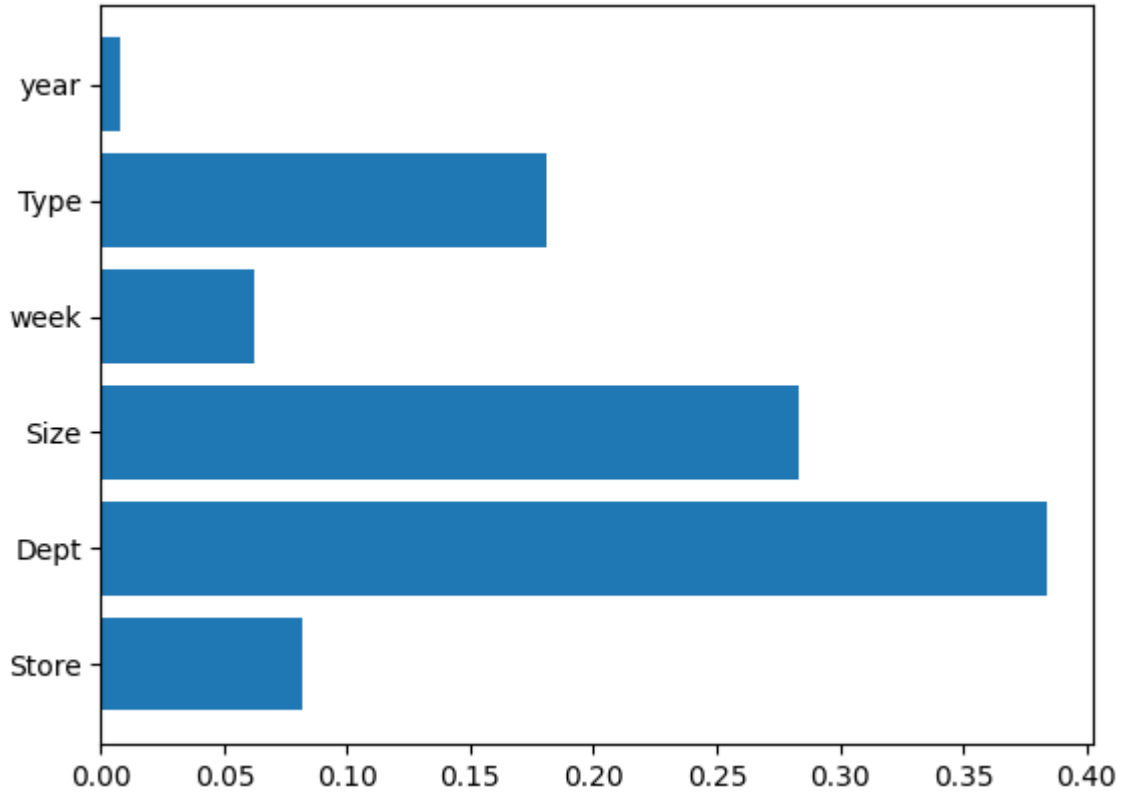

```
performance on evaluation
r2= -1.6246026747934126
MAE= 10758.20301880914
WMAE= 10949.51413880973
============================
depth= 2
#(train)= 337256 #(val) 84314
performance on training
r2=  0.5218889097070569
MAE=  6772.138301852728
WMAE=  7013.81596770443
```



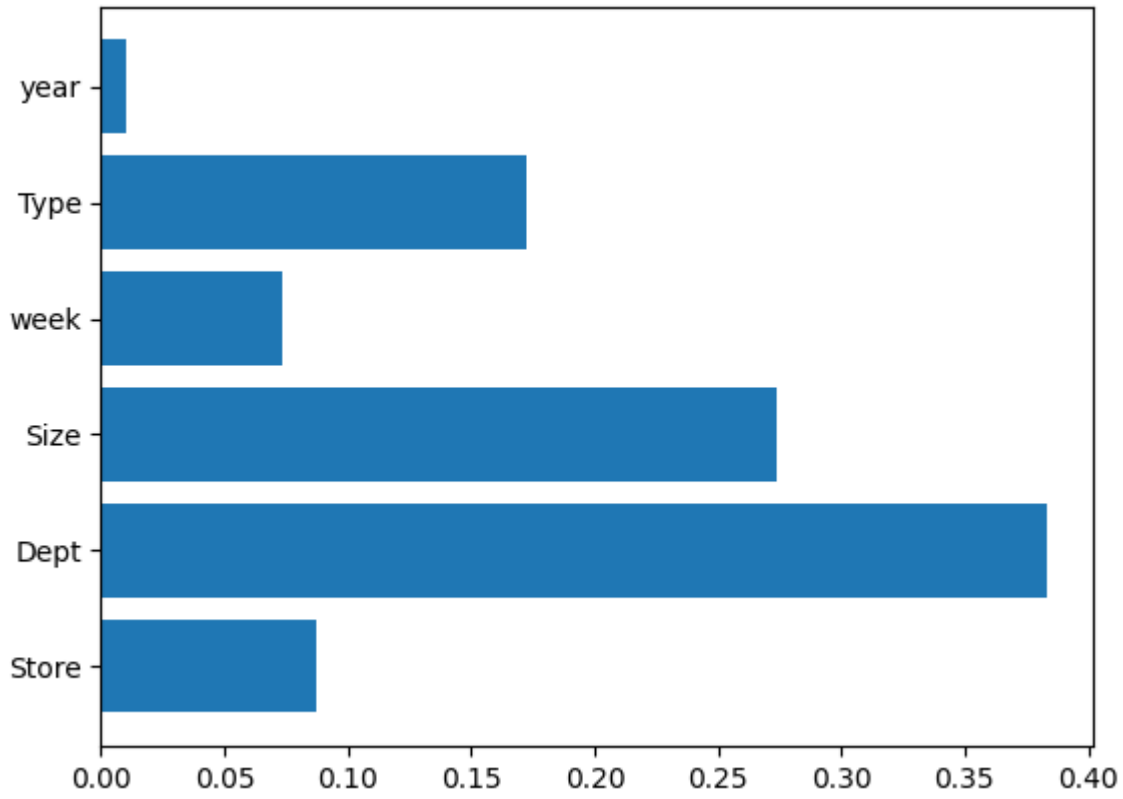

```
performance on evaluation
r2= 0.5046291964359069
MAE= 6749.893161601349
WMAE= 6982.687450740108
==============================
depth= 3
#(train)= 337256 #(val) 84314
performance on training
r2=  0.7660644104394211
MAE=  5142.203372820436
WMAE=  5365.173375192148
```



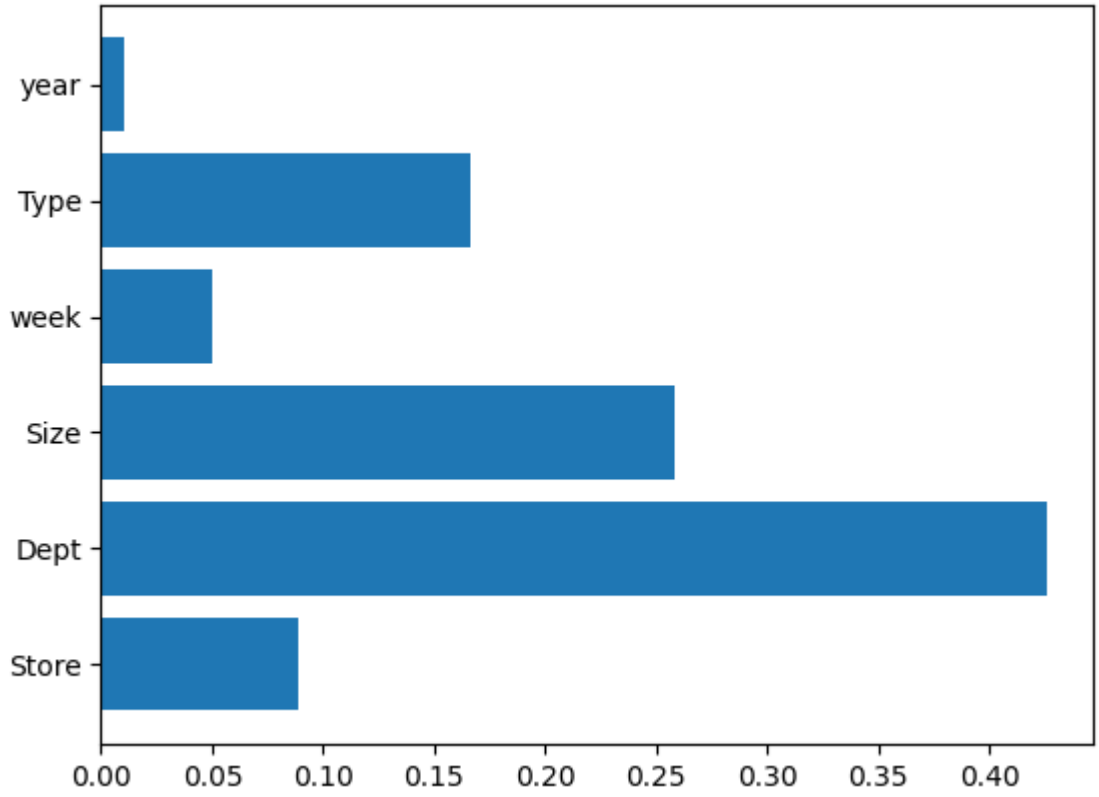

```
performance on evaluation
r2= 0.7551330955975125
MAE= 5127.444553527481
WMAE= 5361.737126055145
============================
depth= 4
#(train)= 337256 #(val) 84314
performance on training
r2=  0.8547619209770042
MAE=  4195.578715636453
WMAE=  4413.043644517702
```



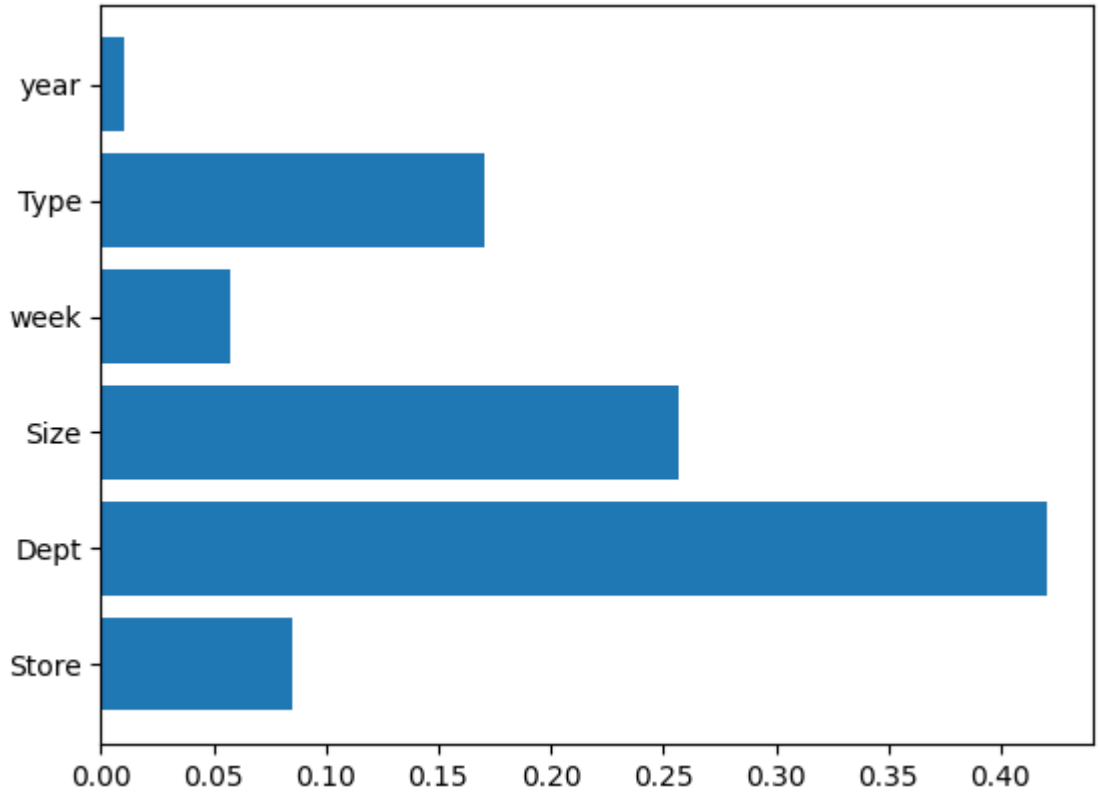

```
performance on evaluation
r2= 0.8464814355883463
MAE= 4197.774128399022
WMAE= 4419.060580064331
============================
depth= 5
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9127661384743316
MAE=  3413.861346885805
WMAE=  3561.190747645296
```



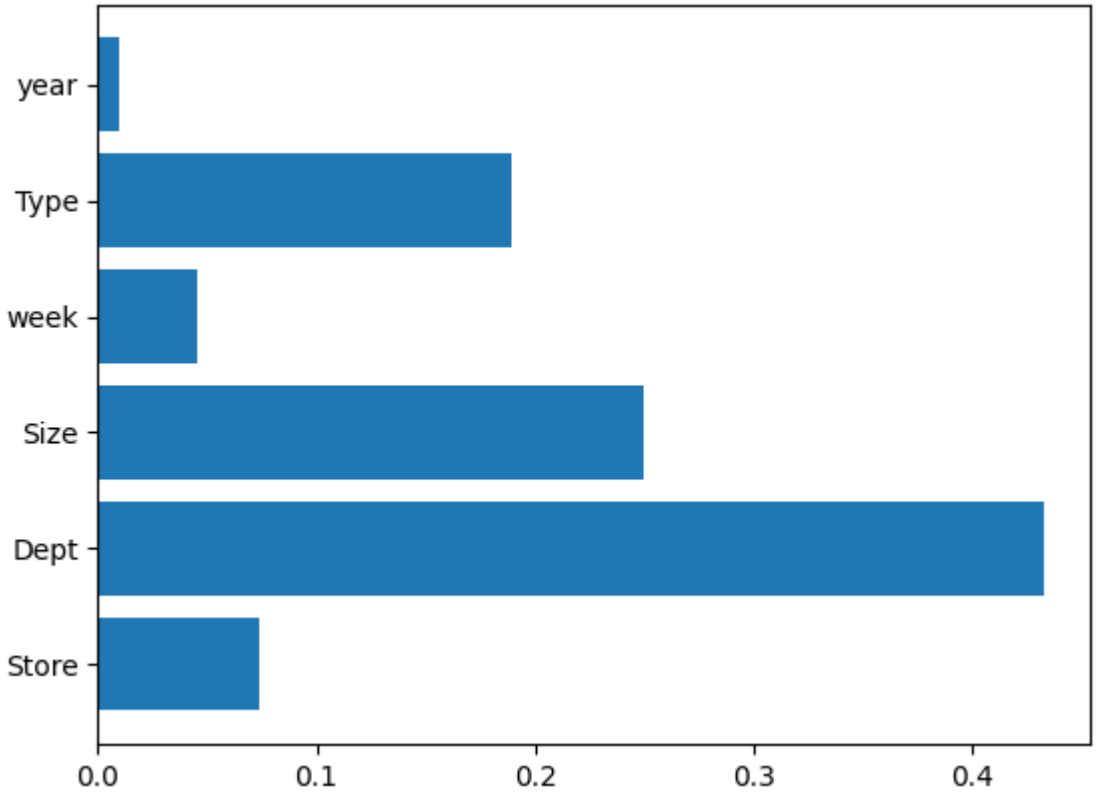

```
performance on evaluation
r2= 0.9079138847761882
MAE= 3433.409700094147
WMAE= 3587.3312591463127
============================
depth= 6
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9453022978693839
MAE=  2769.2122129395675
WMAE=  2885.6580984990333
```



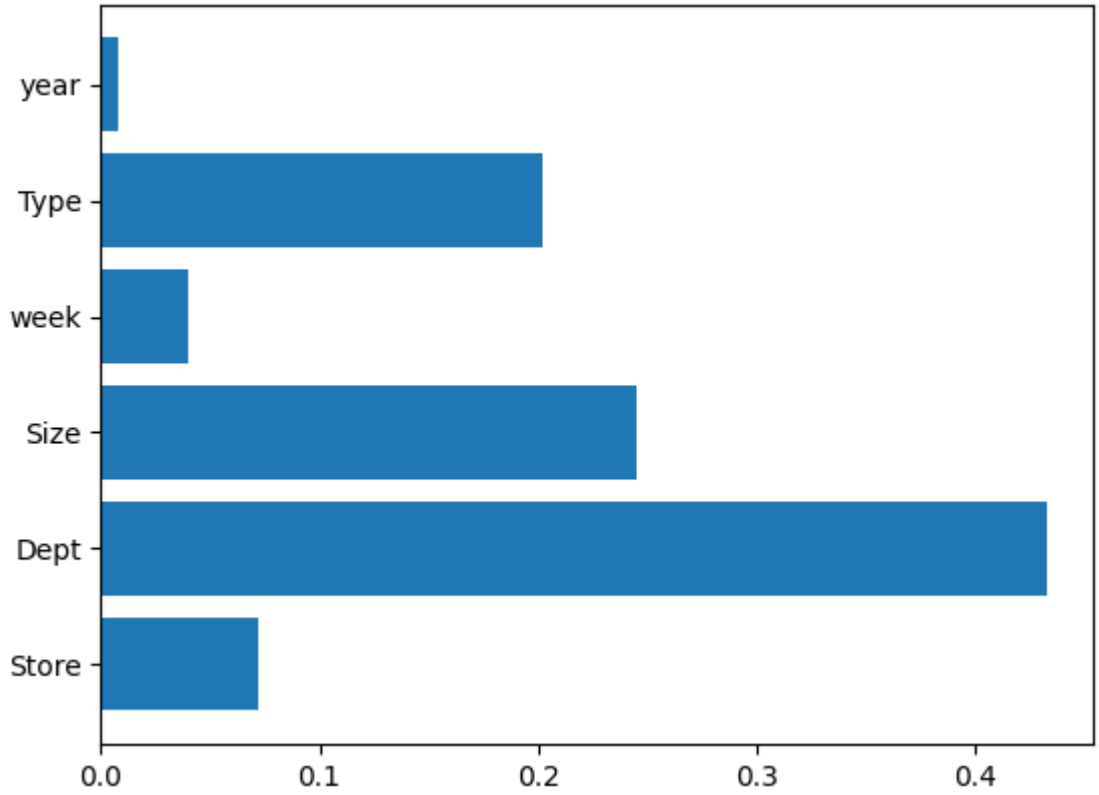

```
performance on evaluation
r2= 0.9411815136218987
MAE= 2805.500876047611
WMAE= 2929.5482482529446
============================
depth= 7
#(train)= 337256 #(val) 84314
performance on training
r2=  0.967009732882056
MAE=  2212.13433381156
WMAE=  2296.047501918744
```



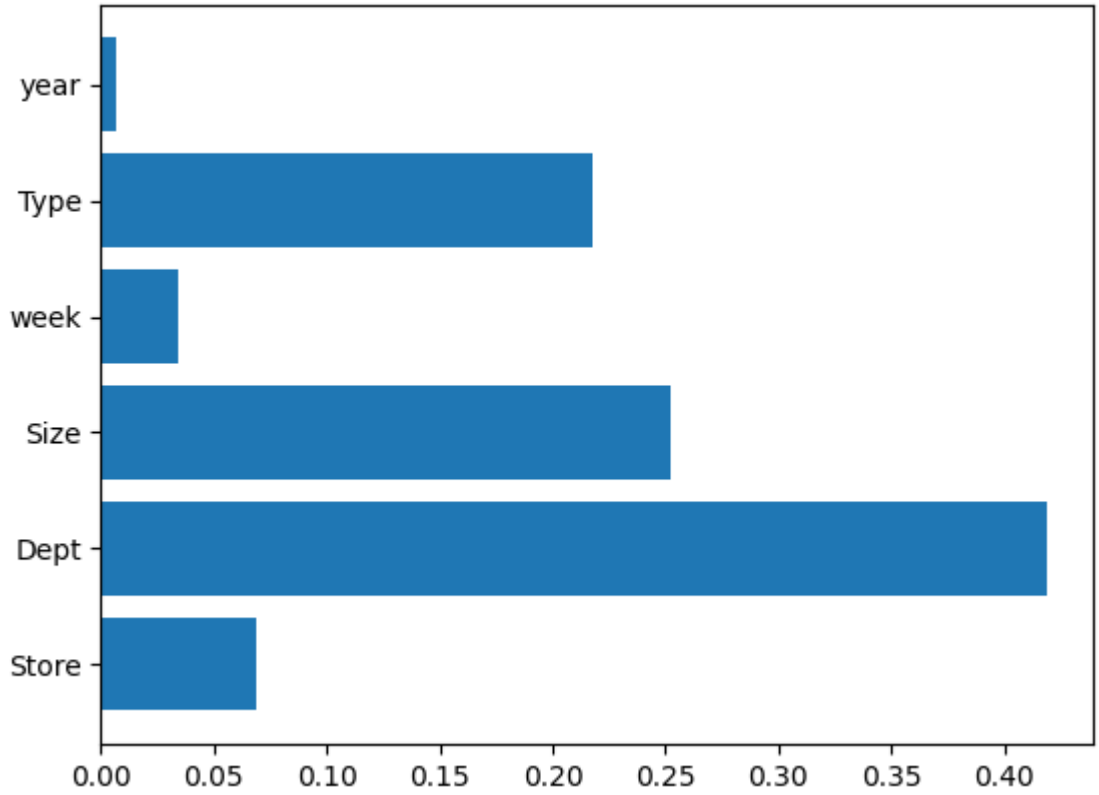

```
performance on evaluation
r2= 0.9625494513555661
MAE= 2277.0227923495468
WMAE= 2375.5316163193297
============================
depth= 8
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9795603946180931
MAE=  1750.21592571601
WMAE=  1816.5566195497058
```



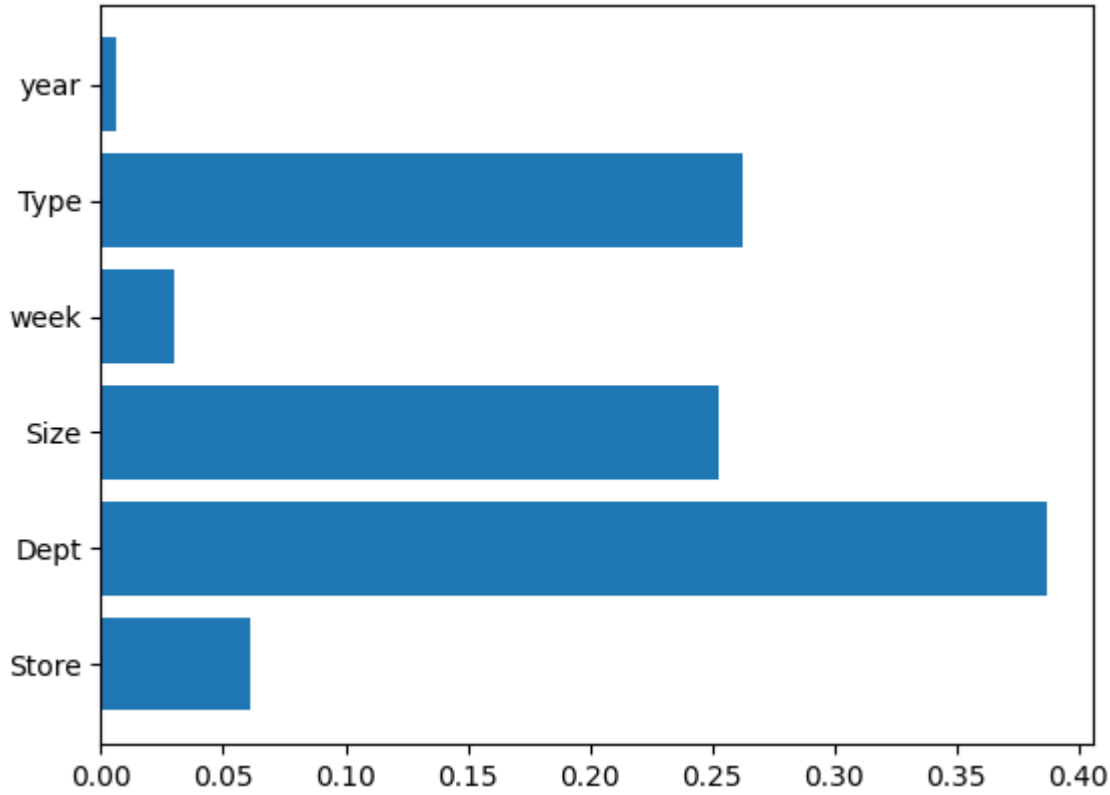

```
performance on evaluation
r2= 0.9743993033679887
MAE= 1857.9005195091777
WMAE= 1951.563464597922
============================
depth= 9
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9875227049986057
MAE=  1382.2849716514534
WMAE=  1422.7130849016075
```



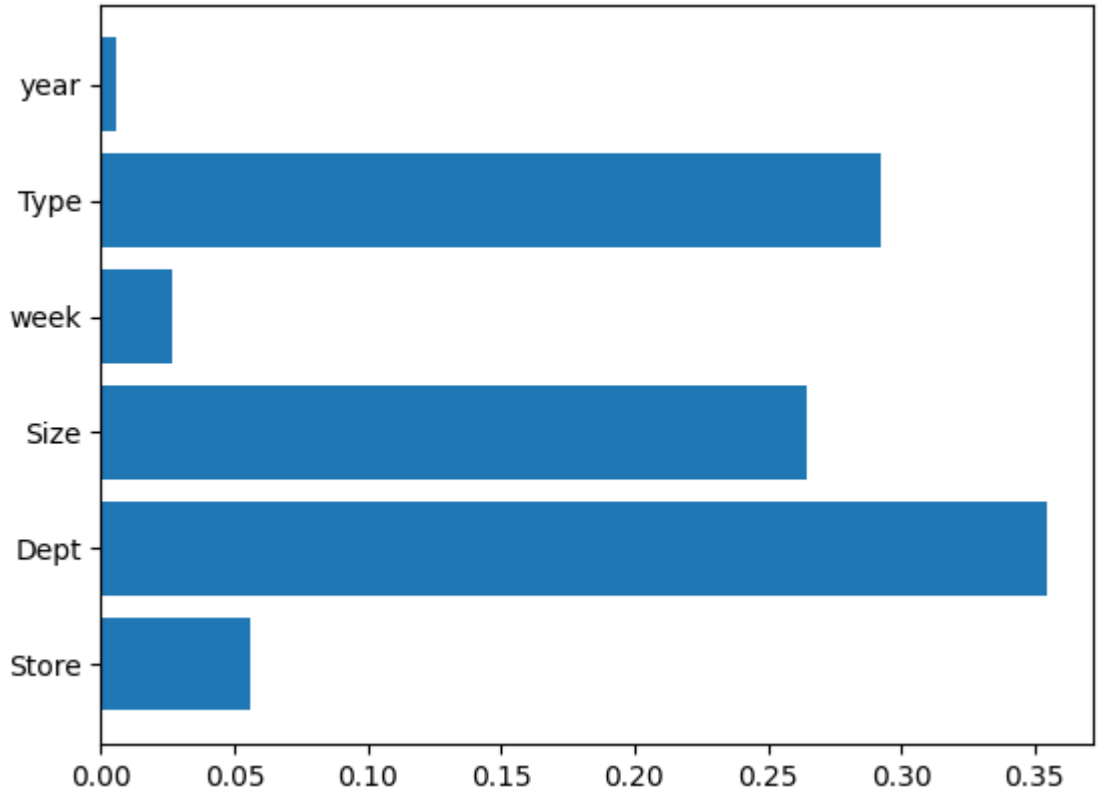

```
performance on evaluation
r2= 0.9813869302364139
MAE= 1543.831955944529
WMAE= 1621.2474875547705
============================
depth= 10
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9920263482285888
MAE=  1116.6331379254916
WMAE=  1143.5314854428502
```



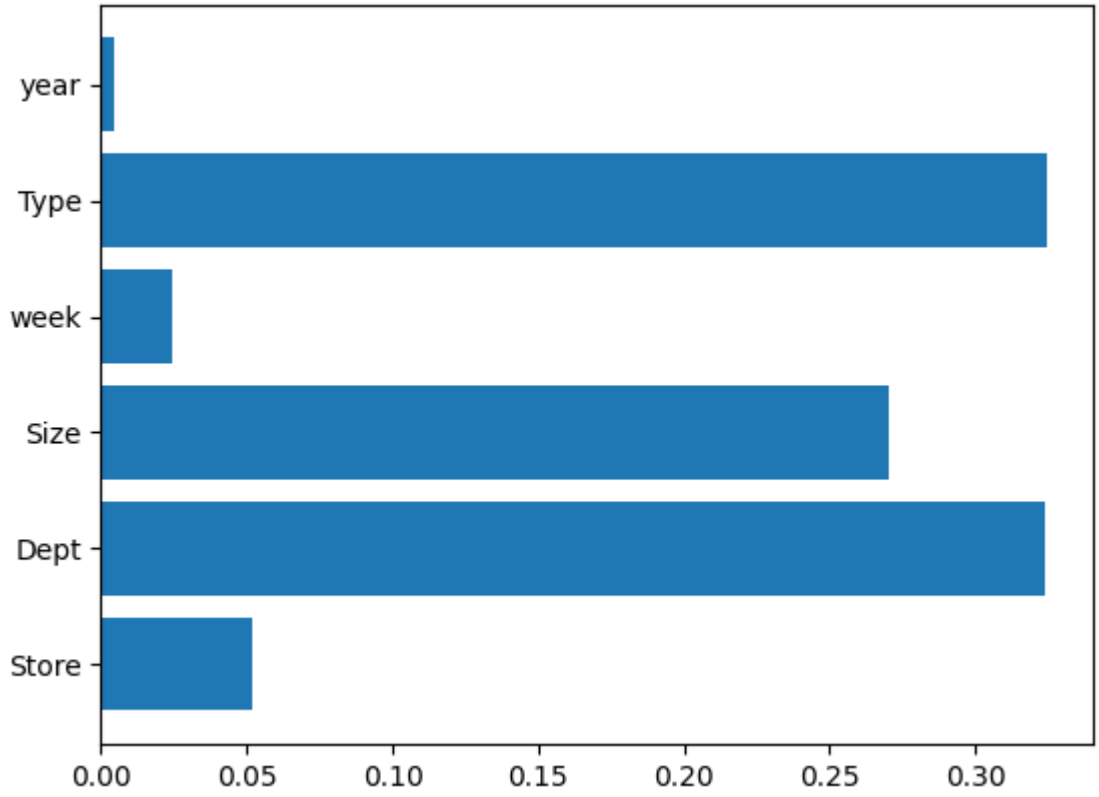

```
performance on evaluation
r2= 0.9852910884767765
MAE= 1348.4185556912348
WMAE= 1427.35127721853
============================
depth= 11
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9952412925227473
MAE=  875.0154997046584
WMAE=  889.8644491079411
```



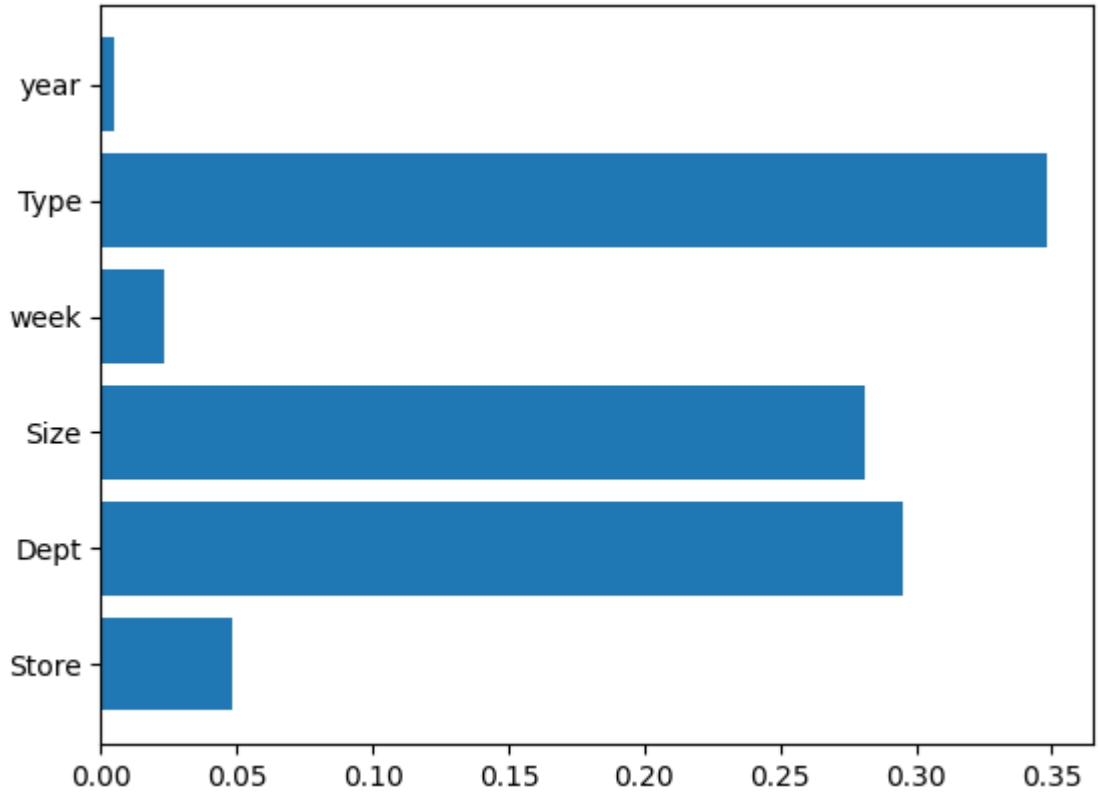

```
performance on evaluation
r2= 0.9875059755016623
MAE= 1200.6991400014701
WMAE= 1279.1652004939497
============================
depth= 12
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9972595504338009
MAE=  681.4463064331098
WMAE=  686.5273615344045
```



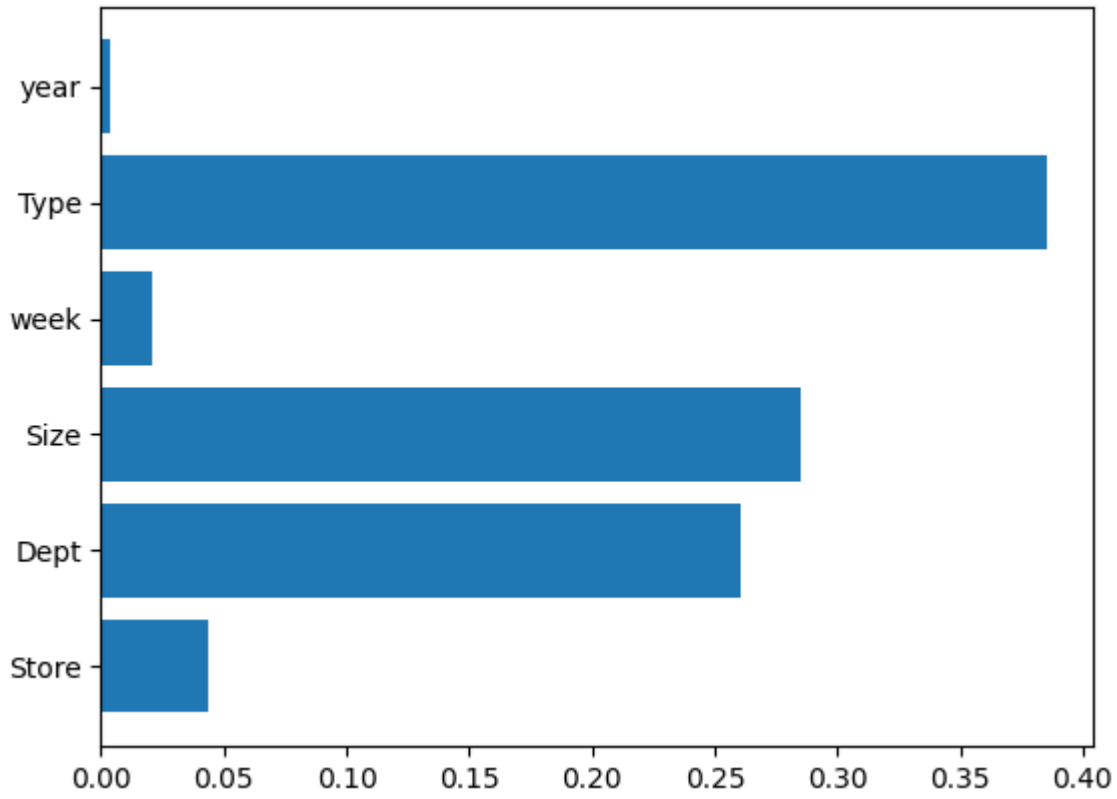

```
performance on evaluation
r2= 0.988256175292737
MAE= 1122.5356089638499
WMAE= 1204.938183495848
============================
depth= 13
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9985153621022143
MAE=  517.0213411928972
WMAE=  515.4944329630791
```



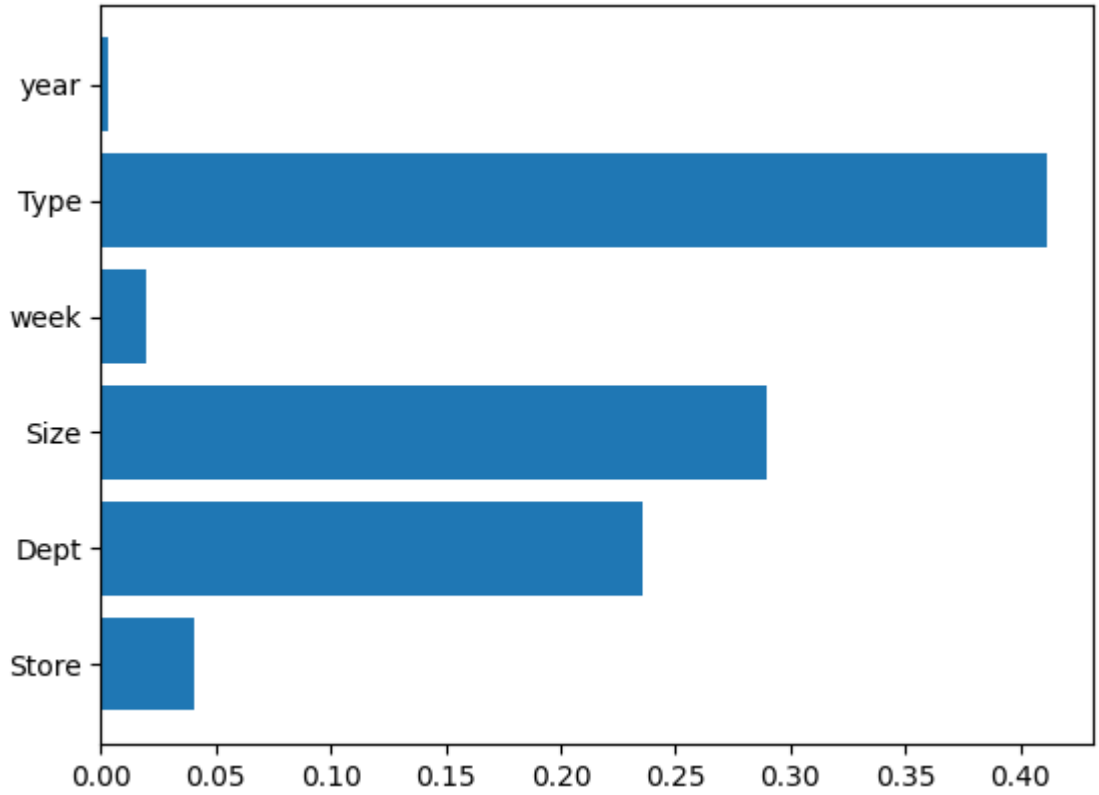

```
performance on evaluation
r2= 0.9886345698245854
MAE= 1084.6598483172127
WMAE= 1166.768783402921
============================
depth= 14
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9992840191472654
MAE=  371.1137170881647
WMAE=  366.201382300487
```



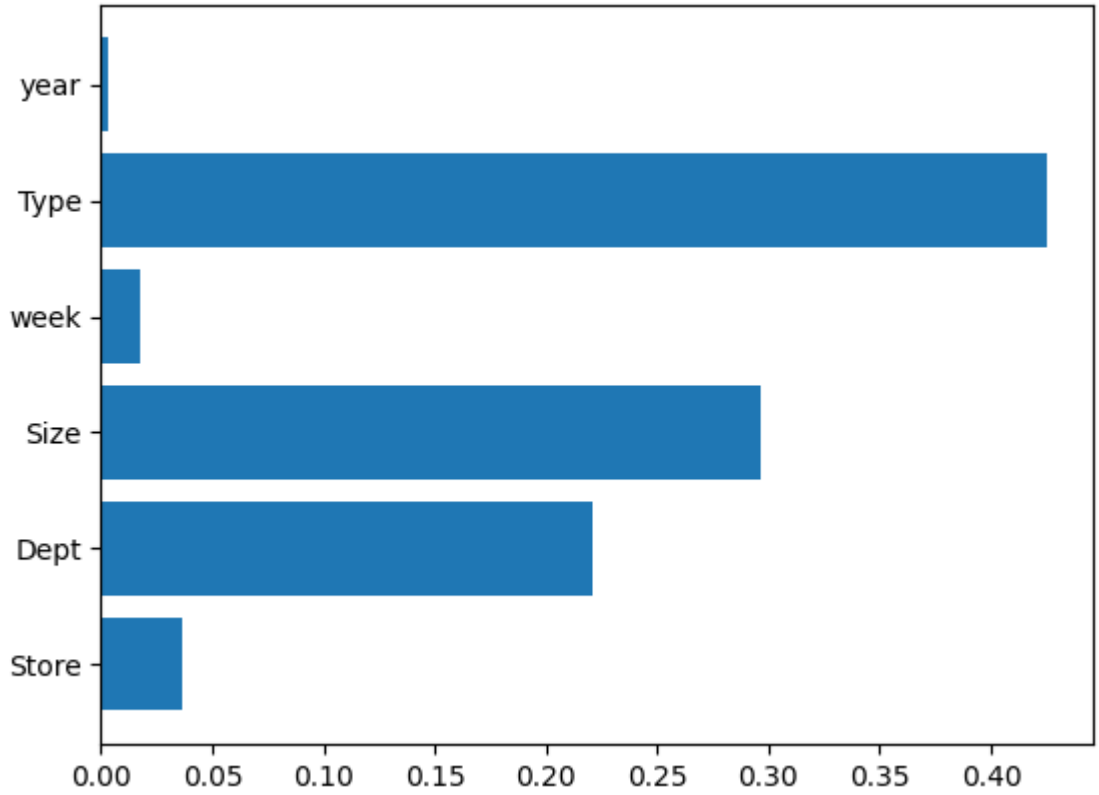

```
performance on evaluation
r2= 0.9882426080143639
MAE= 1074.6378398862844
WMAE= 1157.30707669466
============================
depth= 15
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9997089420943952
MAE=  246.40984414745233
WMAE=  242.20338827670946
```



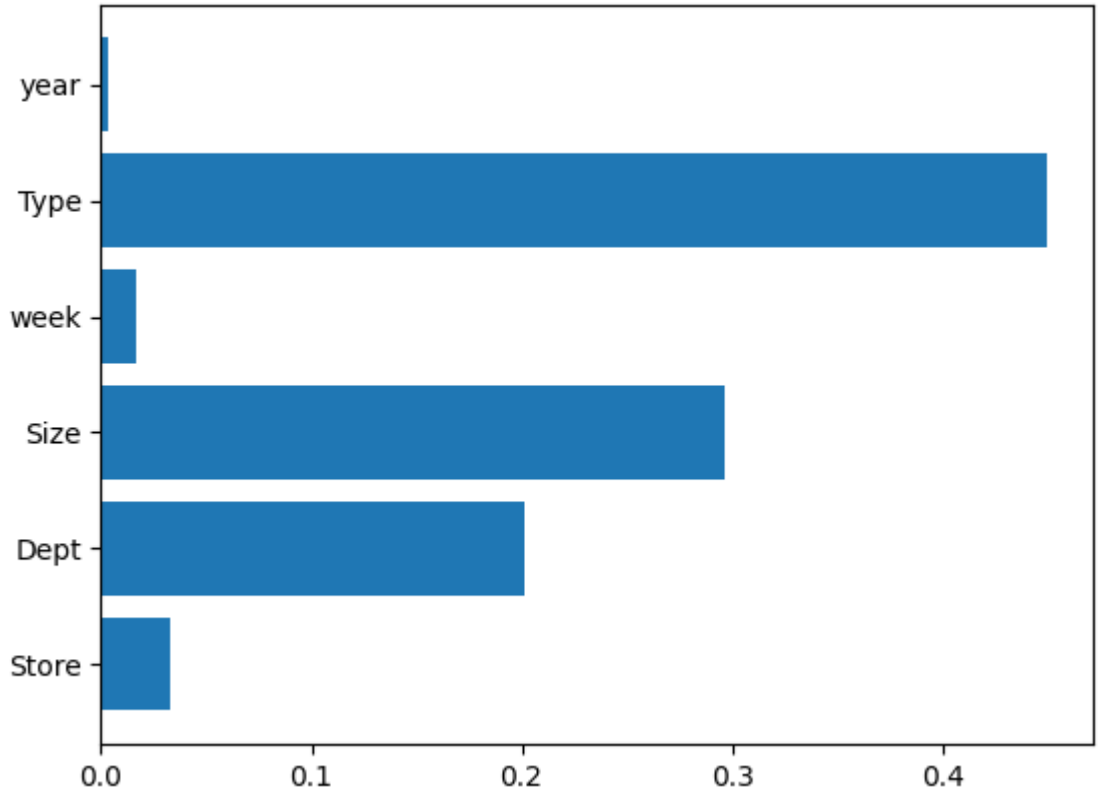

```
performance on evaluation
r2= 0.9877960176447831
MAE= 1077.0569969713174
WMAE= 1161.8254080831325
==============================
depth= 16
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9998949484803834
MAE=  152.44202933178553
WMAE=  149.0007667841276
```



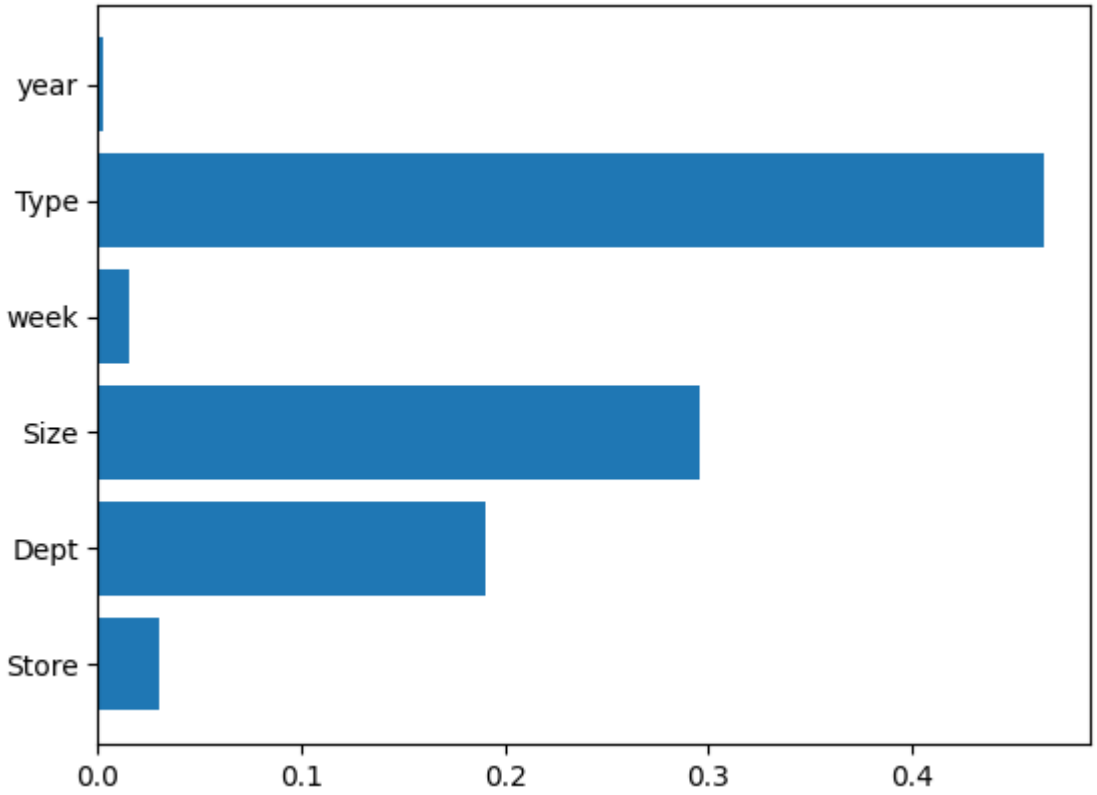

```
performance on evaluation
r2= 0.9871205325416866
MAE= 1089.6223891471404
WMAE= 1175.960541740415
============================
depth= 17
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9999681108025313
MAE=  85.54300833031643
WMAE=  82.55643712479909
```




```
performance on evaluation
r2= 0.9866016048393839
MAE= 1104.6389929583434
WMAE= 1199.5425722949897
============================
depth= 18
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9999919974277246
MAE=  42.776030804133725
WMAE=  41.27517293701089
```



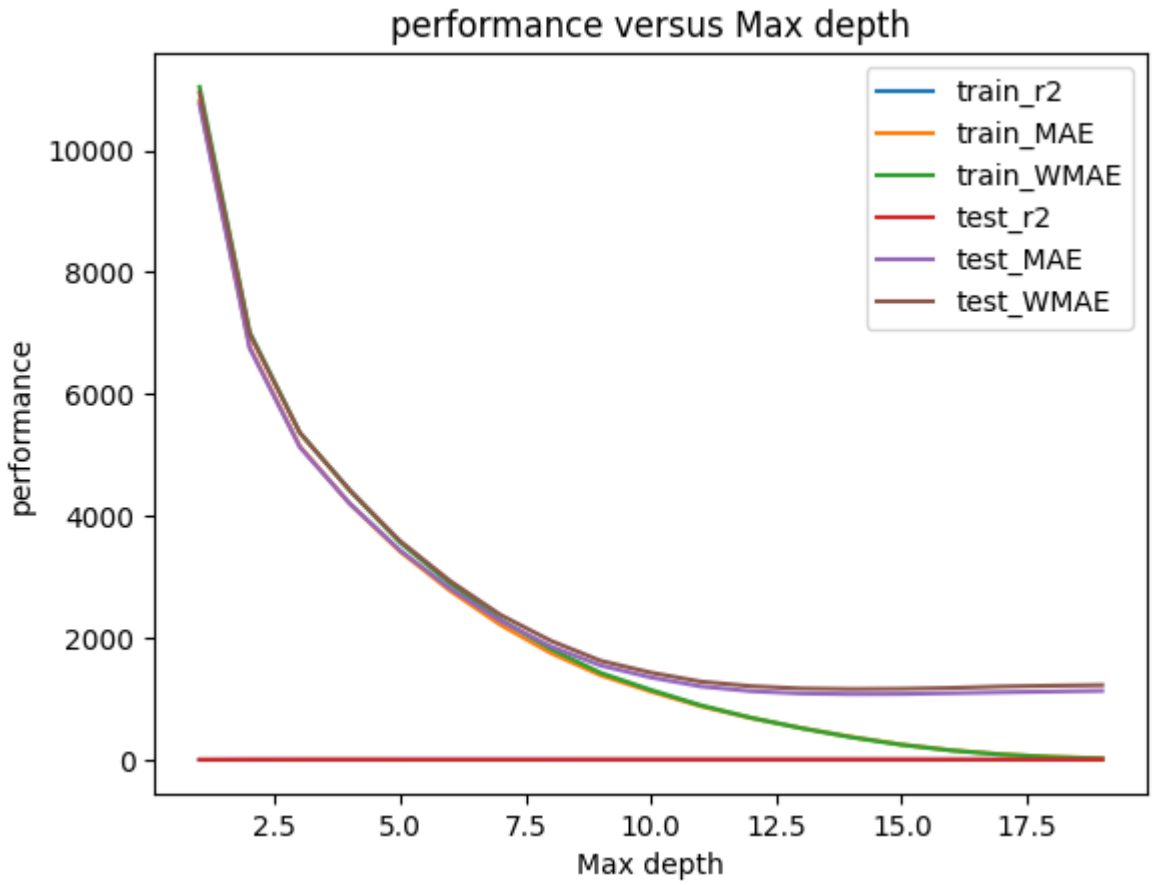

```
performance on evaluation
r2= 0.9861681884869635
MAE= 1116.6077207113788
WMAE= 1212.6969304240408
============================
depth= 19
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9999981398045825
MAE=  19.437524006658574
WMAE=  18.662416695585353
```




```
performance on evaluation
r2= 0.9858496070743521
MAE= 1128.1136282625876
WMAE= 1222.7592471347518
```

In [90]:
```
d_max = 20
```

In [125…
```
def res_plot(serie,y, title, xlabel = None, ylabel = "performance"):
    labelname = ["train_r2","train_MAE","train_WMAE","test_r2","test_MAE","test_WMAE"]

    for i in range(6):
        plt.plot(y, serie[i], label = labelname[i])
    plt.legend()
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.show()

res_plot(serie,range(1,d_max),
        title= "performance versus Max depth",
        xlabel = 'Max depth')
```

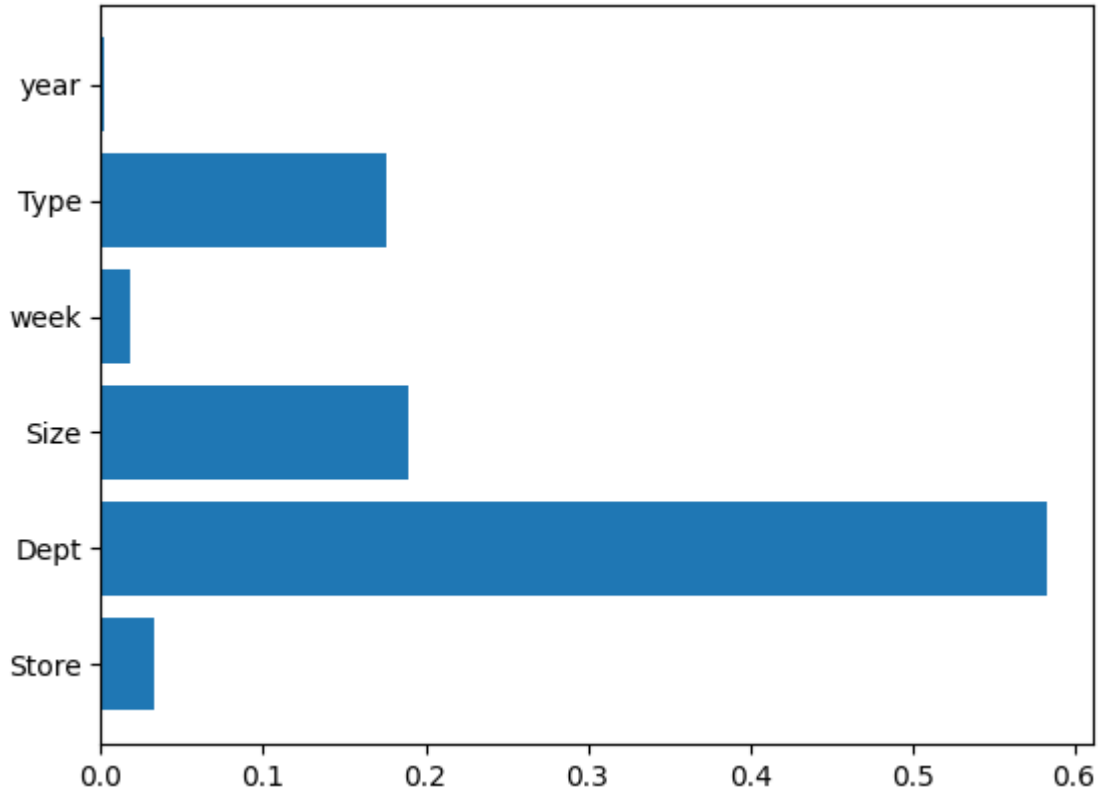


In [96]:
```
# get best depth by WMAE on evaluation dataset
d_best = serie[5].index(min(serie[5])) +1 # from 1
d_best
```

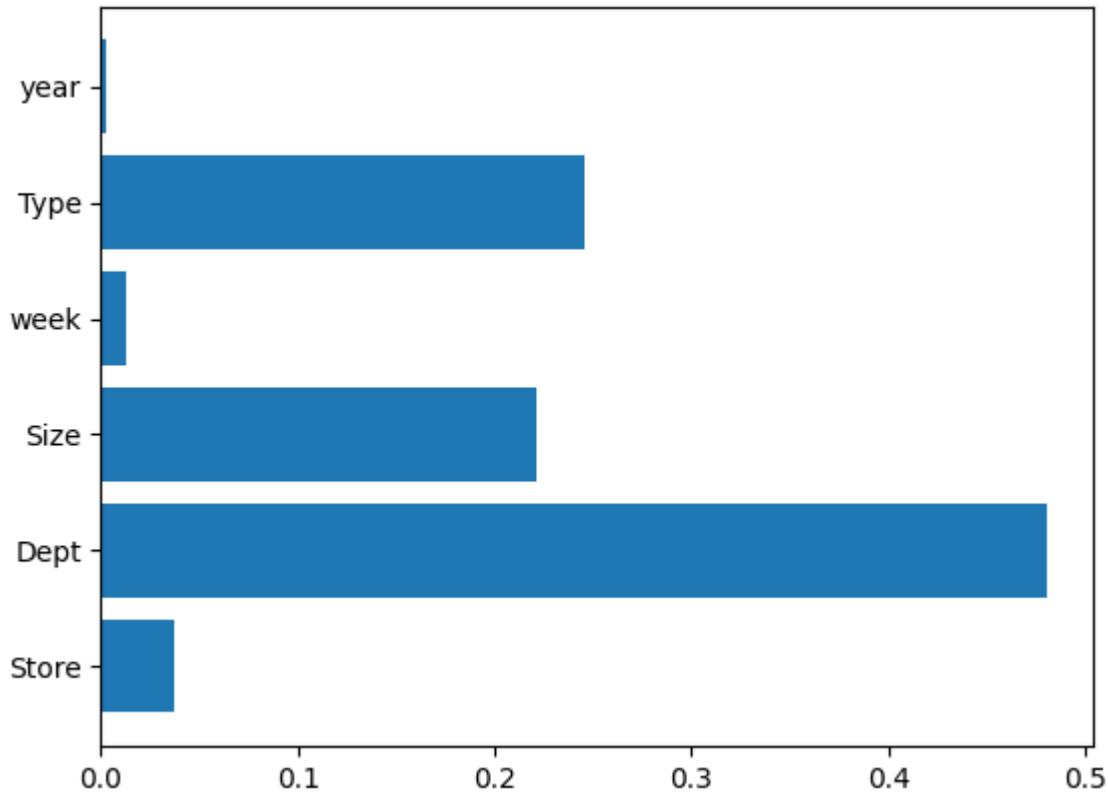
Out[96]:
```
14
```

In [102…
```
# roughly search for N
N_max = 500
feat= ['Weekly_Sales','Store', 'Dept', 'Size', 'week', 'Type', 'Date', 'year'] #, 'IsH
res_n = {}
serie_n = [[] for i in range(6)]
y = range(1,N_max,40)
for n in y:
    print('============================')
    print("N=",n)
    tmp = train_boost(df_train, df_val, sumb = False, depth= d_best , N_ = n, features
    res_n[n] = tmp
    for i in range(6):
        serie_n[i].append(tmp[i])
```

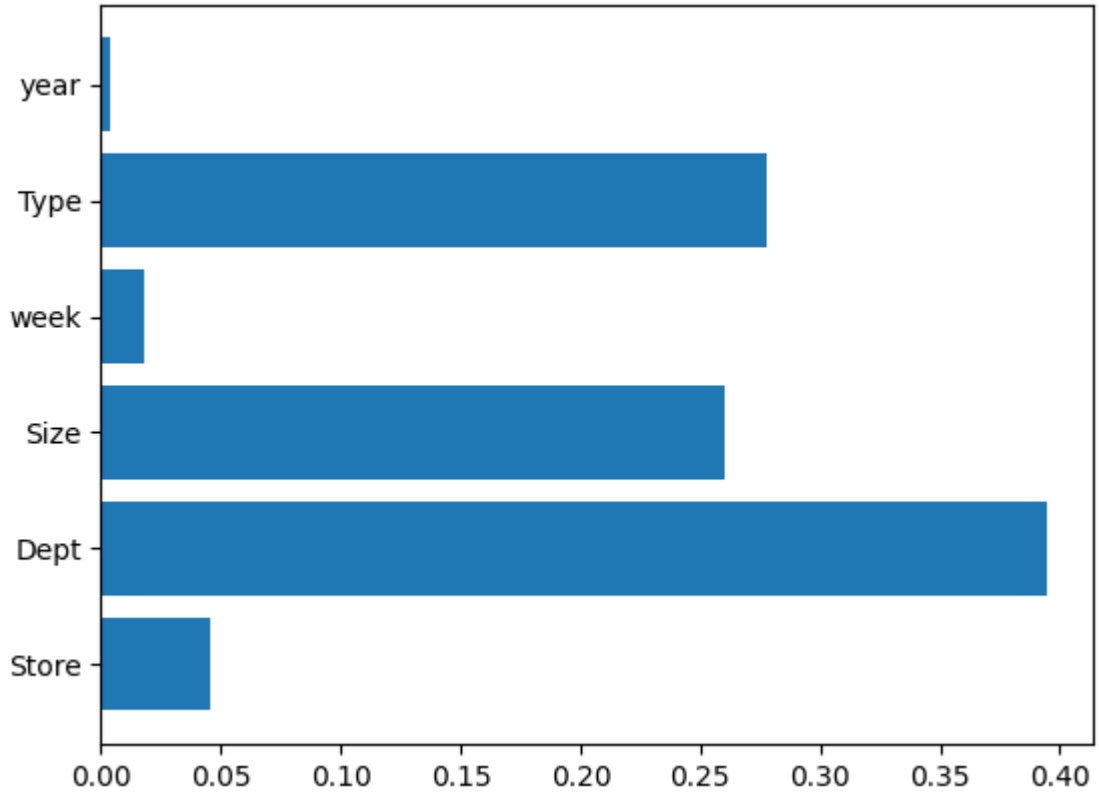
```
============================
N= 1
#(train)= 337256 #(val) 84314
performance on training
r2=  -131.3040082855635
MAE=  14426.244361016692
WMAE=  14659.27446571343
```



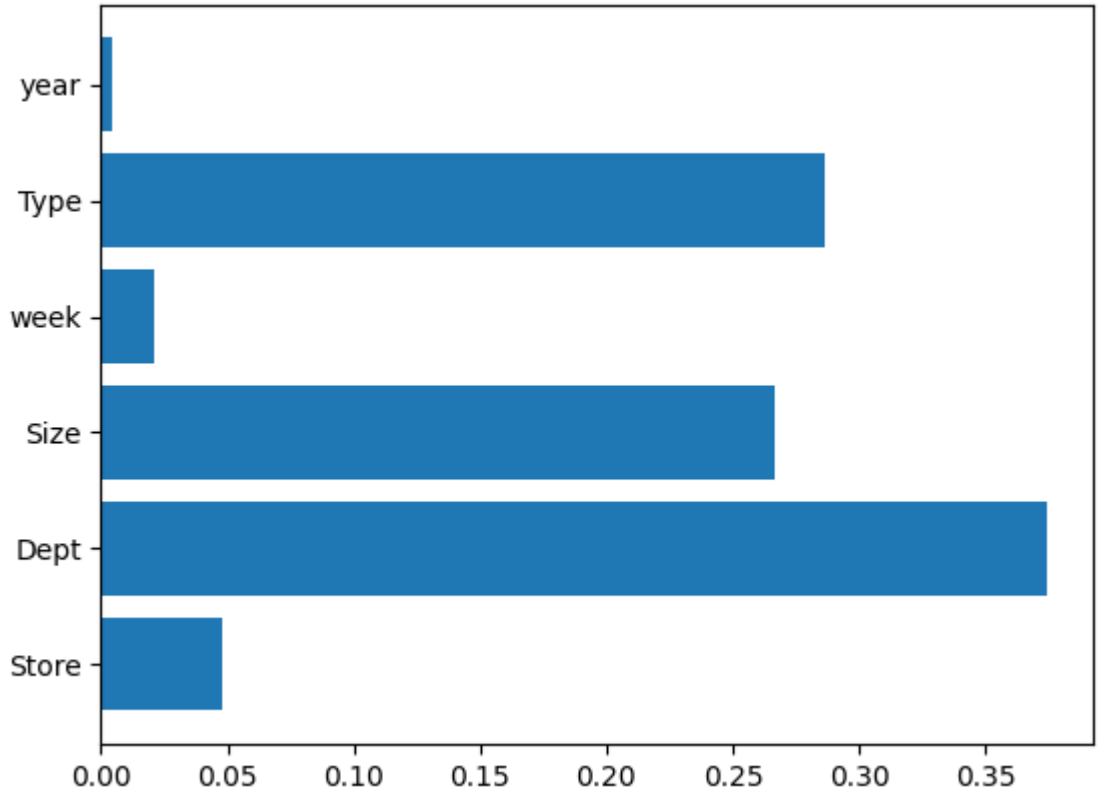

```
performance on evaluation
r2= -131.28822442783667
MAE= 14296.051369047093
WMAE= 14428.886704298407
============================
N= 41
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9874101604962308
MAE=  1248.2140294309643
WMAE=  1283.1720585313824
```



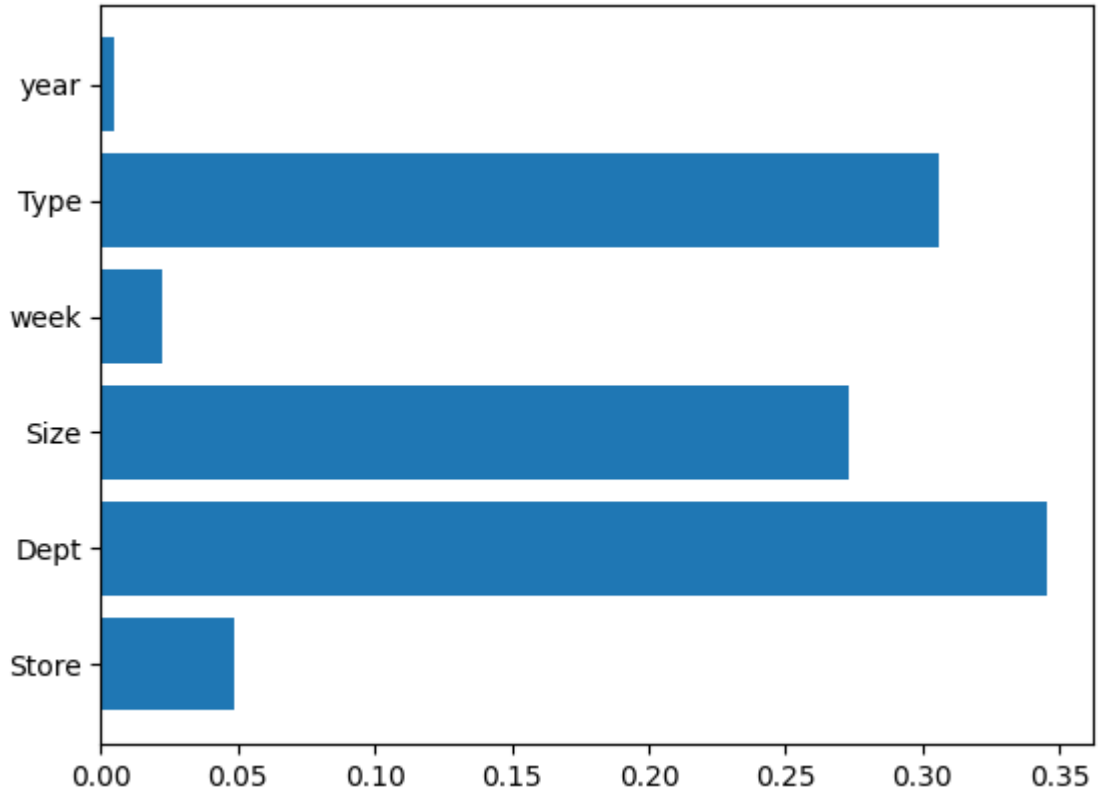

```
performance on evaluation
r2= 0.9782037052198331
MAE= 1478.3142339853537
WMAE= 1569.3846496108868
==============================
N= 81
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9950182844904008
MAE=  863.4675160479705
WMAE=  865.4472691871968
```



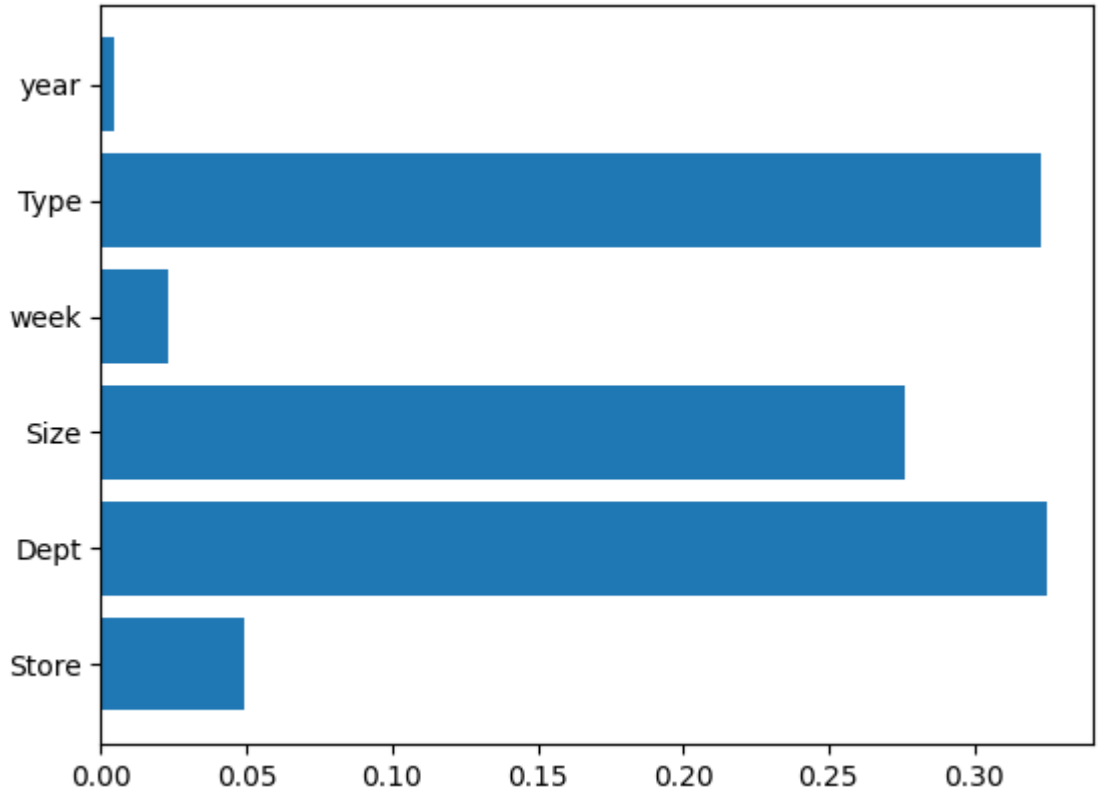

```
performance on evaluation
r2= 0.9855189823174917
MAE= 1234.6522334722463
WMAE= 1312.701196778275
============================
N= 121
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9968192173496607
MAE=  711.8679906706291
WMAE=  709.784649303903
```



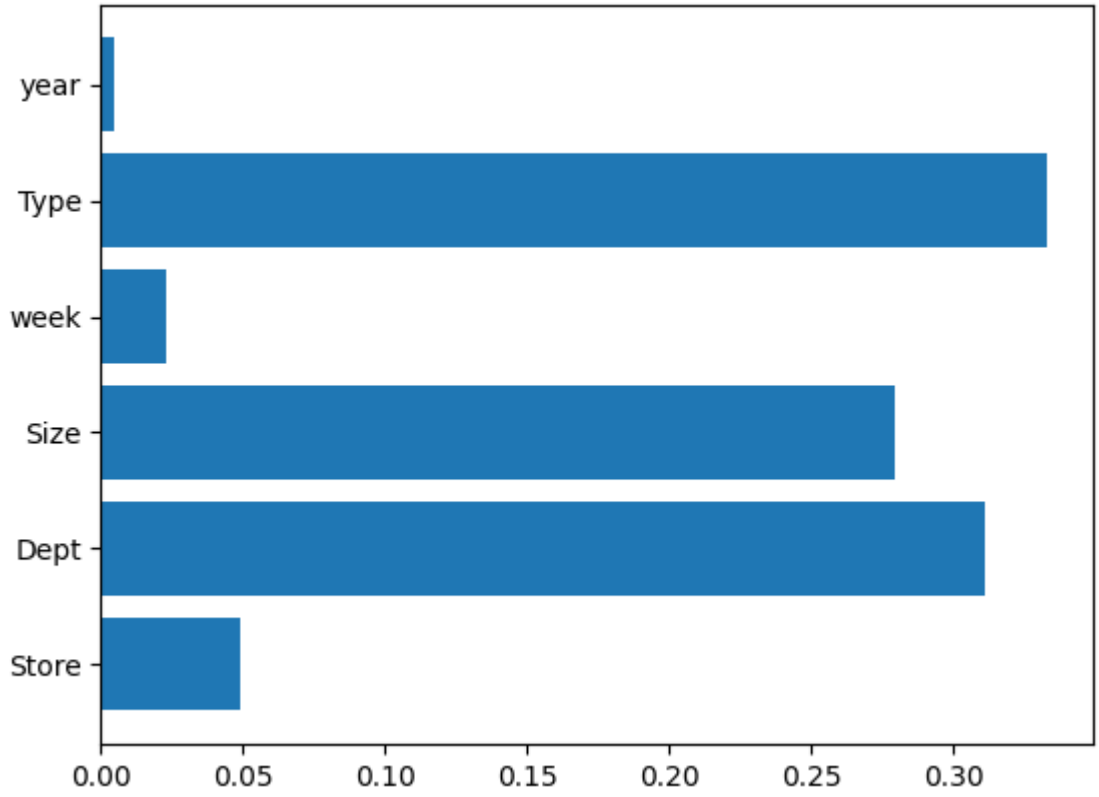

```
performance on evaluation
r2= 0.9869433542935641
MAE= 1159.9531312090444
WMAE= 1239.3689593843108
============================
N= 161
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9978736177144083
MAE=  596.6234423304109
WMAE=  594.2756920403543
```



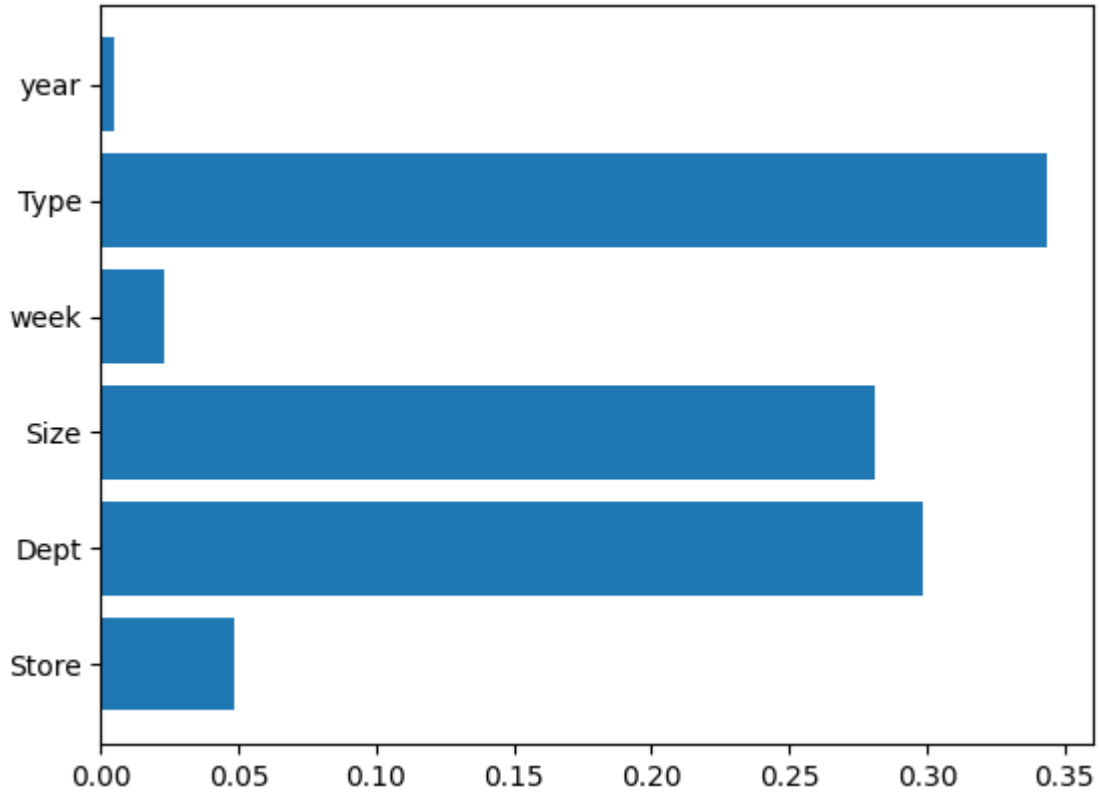

```
performance on evaluation
r2= 0.9876725972775445
MAE= 1115.8099933247133
WMAE= 1196.4554551586805
==============================
N= 201
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9984845811291098
MAE=  513.0788381434048
WMAE=  509.40537942219953
```



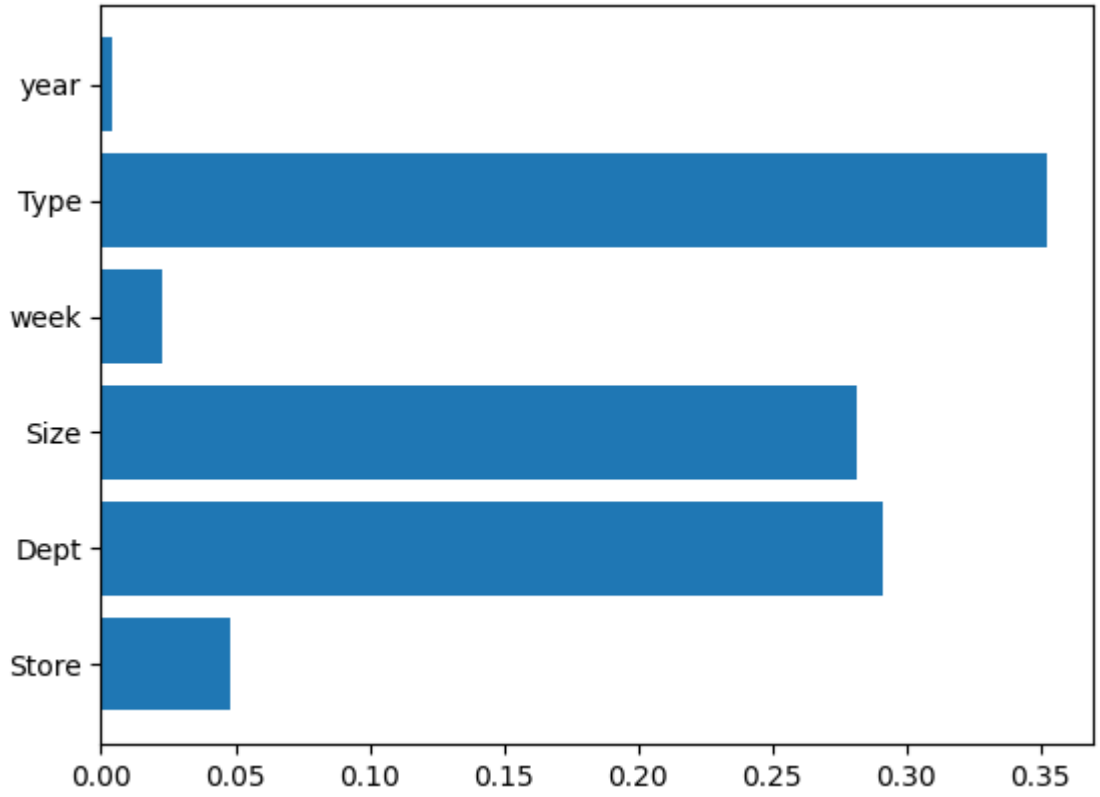

```
performance on evaluation
r2= 0.9879896619612225
MAE= 1093.7575683765833
WMAE= 1175.427071272427
============================
N= 241
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9988937592817378
MAE=  447.4917302568225
WMAE=  443.1467109219653
```



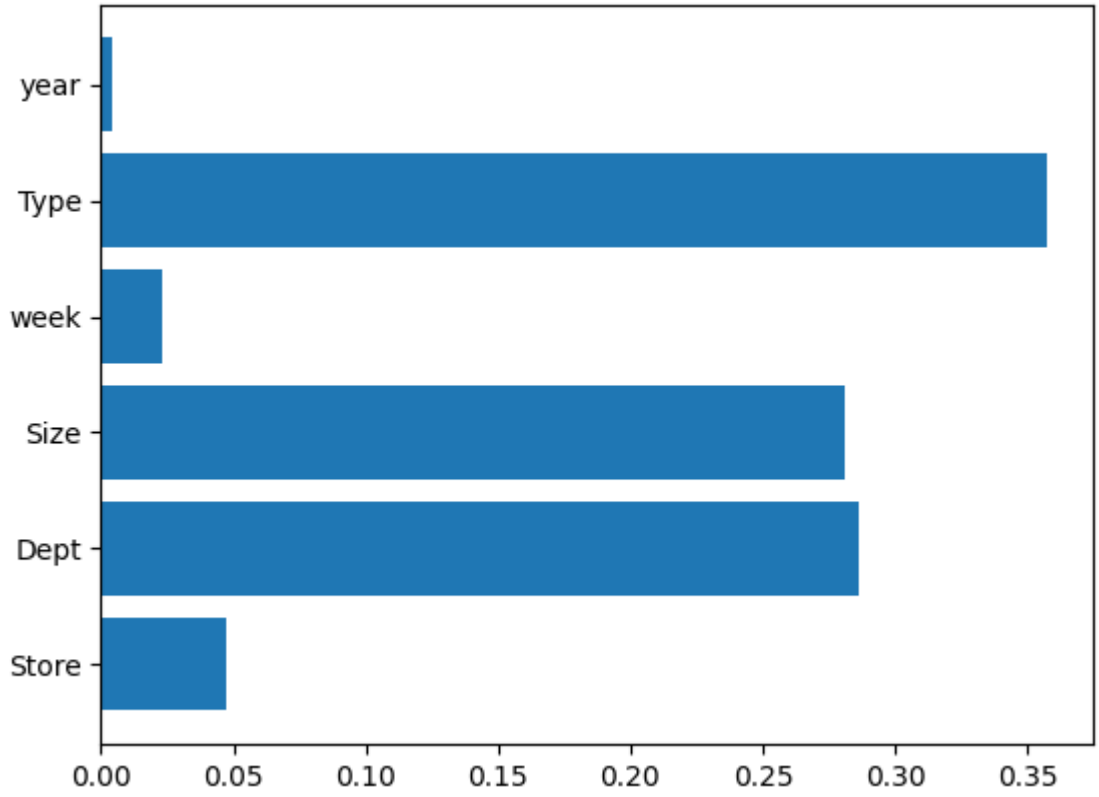

```
performance on evaluation
r2= 0.9881451957768786
MAE= 1082.3823505158414
WMAE= 1164.565223497246
============================
N= 281
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9991761858776189
MAE=  393.9522361044508
WMAE=  388.5671228711775
```



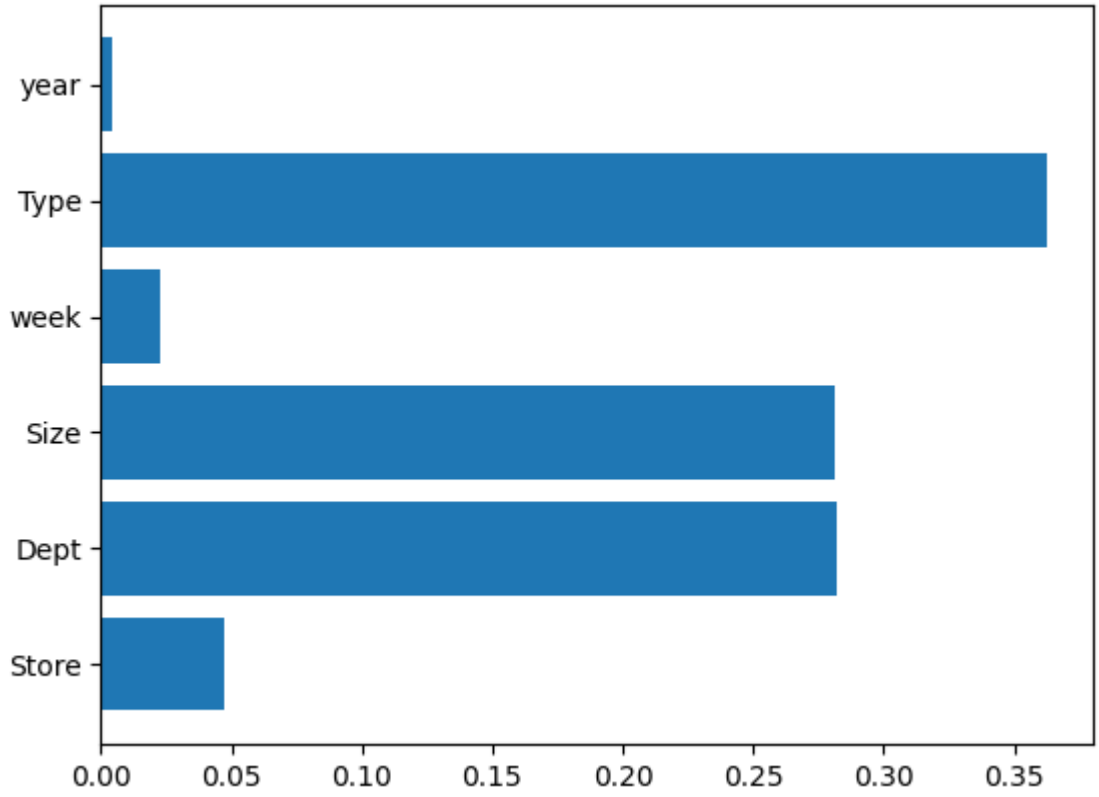

```
performance on evaluation
r2= 0.9882422903762959
MAE= 1075.9781205793313
WMAE= 1158.3844379887405
============================
N= 321
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9993825265481676
MAE=  348.5317787804474
WMAE=  343.6175272806133
```



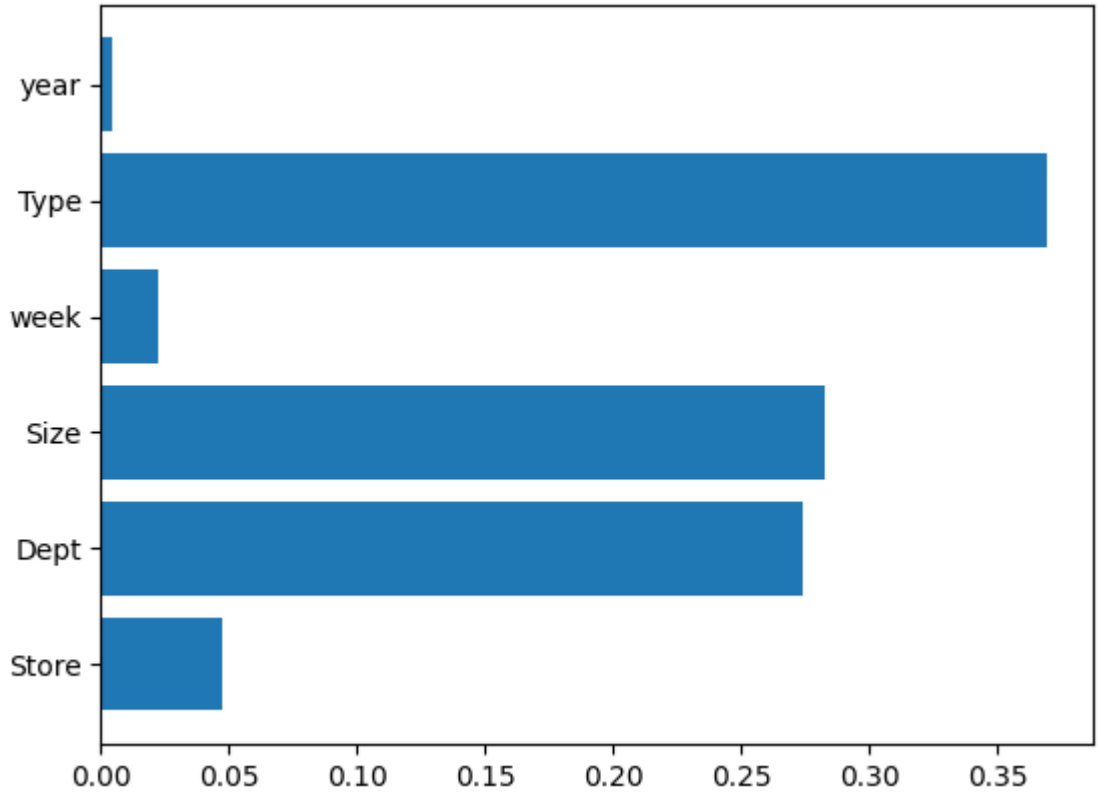

```
performance on evaluation
r2= 0.9882502620300109
MAE= 1073.5823989470925
WMAE= 1156.264898560848
==============================
N= 361
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995258102829883
MAE=  311.37757893631135
WMAE=  306.34507127170724
```



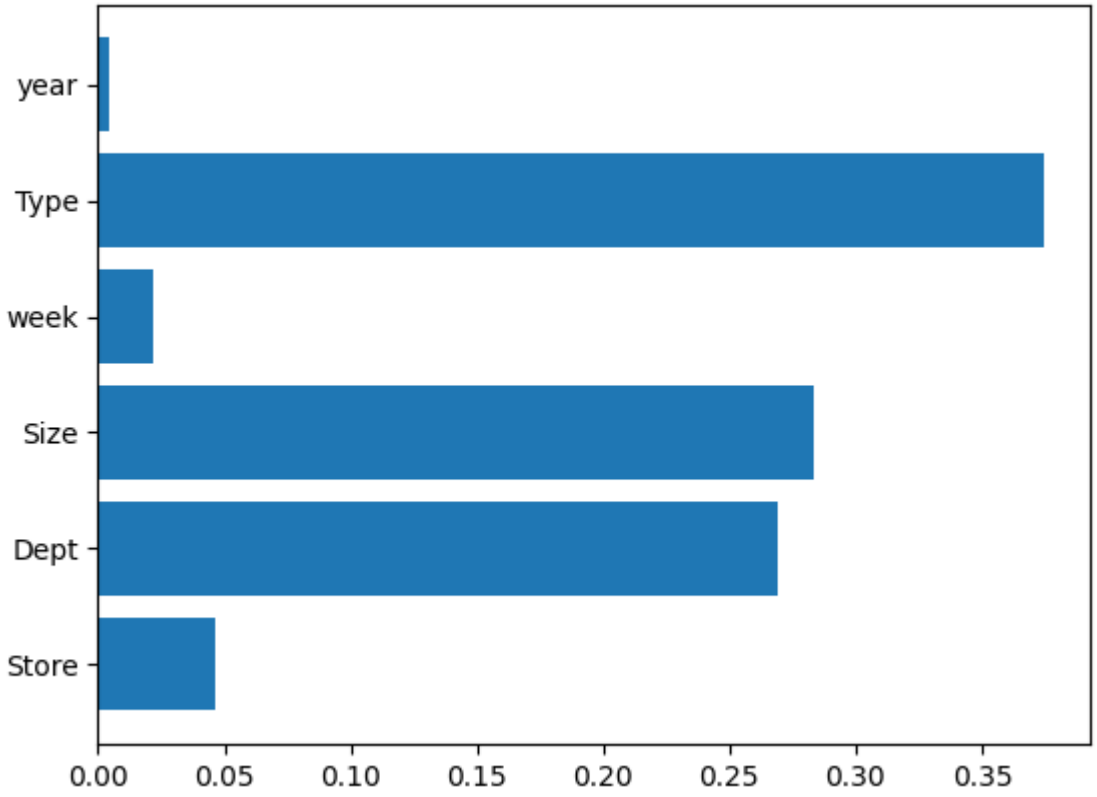

```
performance on evaluation
r2= 0.9882591527821029
MAE= 1072.6098133629455
WMAE= 1155.3387685946914
==============================
N= 401
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996226159666198
MAE=  281.549842100295
WMAE=  276.77672035179677
```




```
performance on evaluation
r2= 0.9882582303936814
MAE= 1072.72605945209
WMAE= 1155.1752618236108
==============================
N= 441
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996941806113746
MAE=  256.41756462703745
WMAE=  251.80531622493683
```



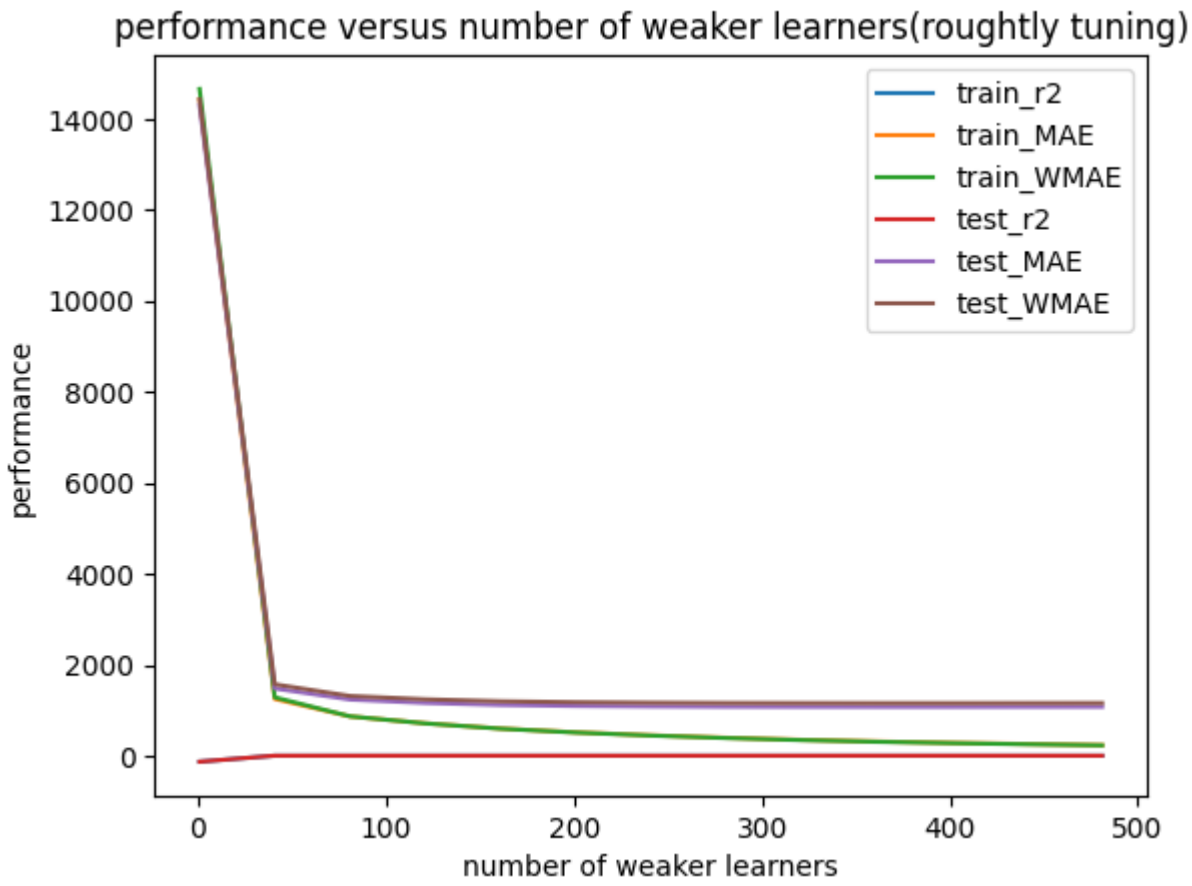

```
performance on evaluation
r2= 0.9882431572418455
MAE= 1073.8675287162073
WMAE= 1156.2968098538654
============================
N= 481
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9997538504172737
MAE=  232.69070165044022
WMAE=  228.46465253874226
```




```
performance on evaluation
r2= 0.9882331659886047
MAE= 1075.1107285099822
WMAE= 1157.4008501385788
```

In [127…

```
res_plot(serie_n,y,
        title= "performance versus number of weaker learners(roughtly tuning)",
        xlabel = 'number of weaker learners')
```



In [109…
```
N_best = serie_n[5].index(min(serie_n[5]))
print(len(serie_n[5]))
print(serie_n[5])
N_best
```

```
13
[14428.886704298407, 1569.3846496108868, 1312.701196778275, 1239.3689593843108, 1196.
4554551586805, 1175.427071272427, 1164.565223497246, 1158.3844379887405, 1156.2648985
60848, 1155.3387685946914, 1155.1752618236108, 1156.2968098538654, 1157.400850138578
8]
```

Out[109]:
```
10
```

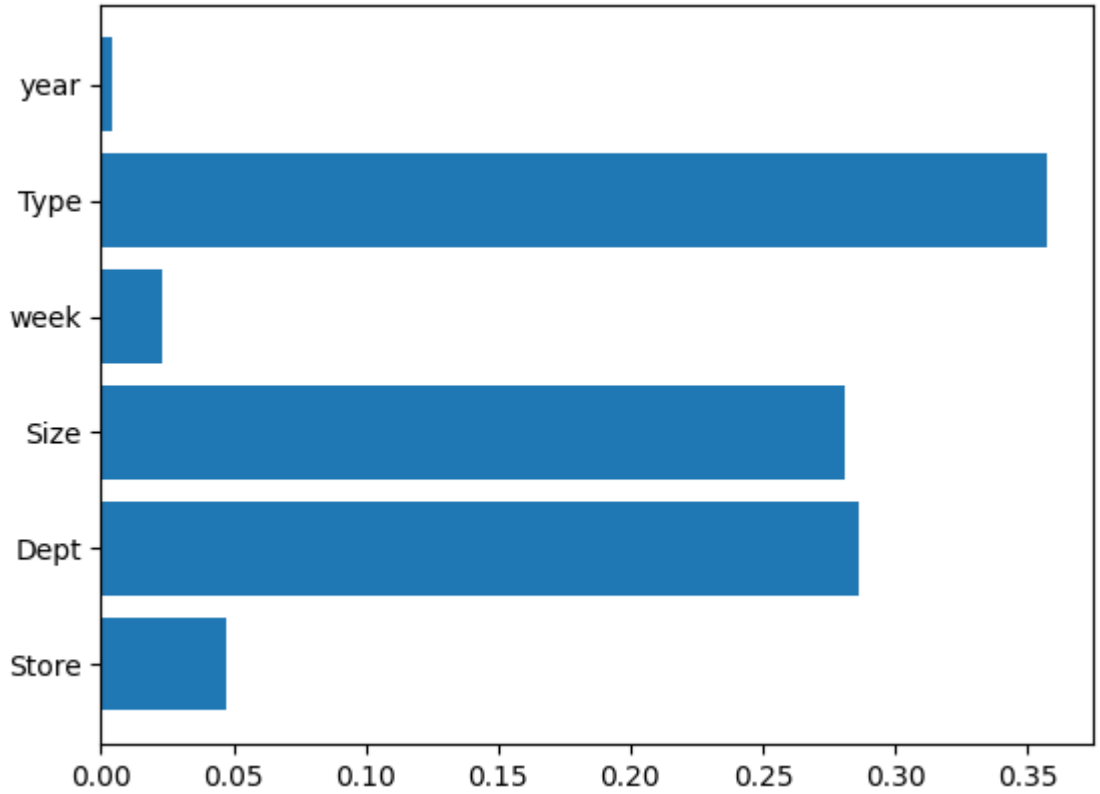
In [110…
```
# best N range
N_best = serie_n[5].index(min(serie_n[5]))
a,b=0,0
N_list = list(y)
if (serie_n[5][N_best-1] < serie_n[5][N_best+1]):
    a,b=N_list[N_best-1],N_list[N_best]
else:
    a,b=N_list[N_best],N_list[N_best+1]
a,b
```

Out[110]:
```
(361, 401)
```

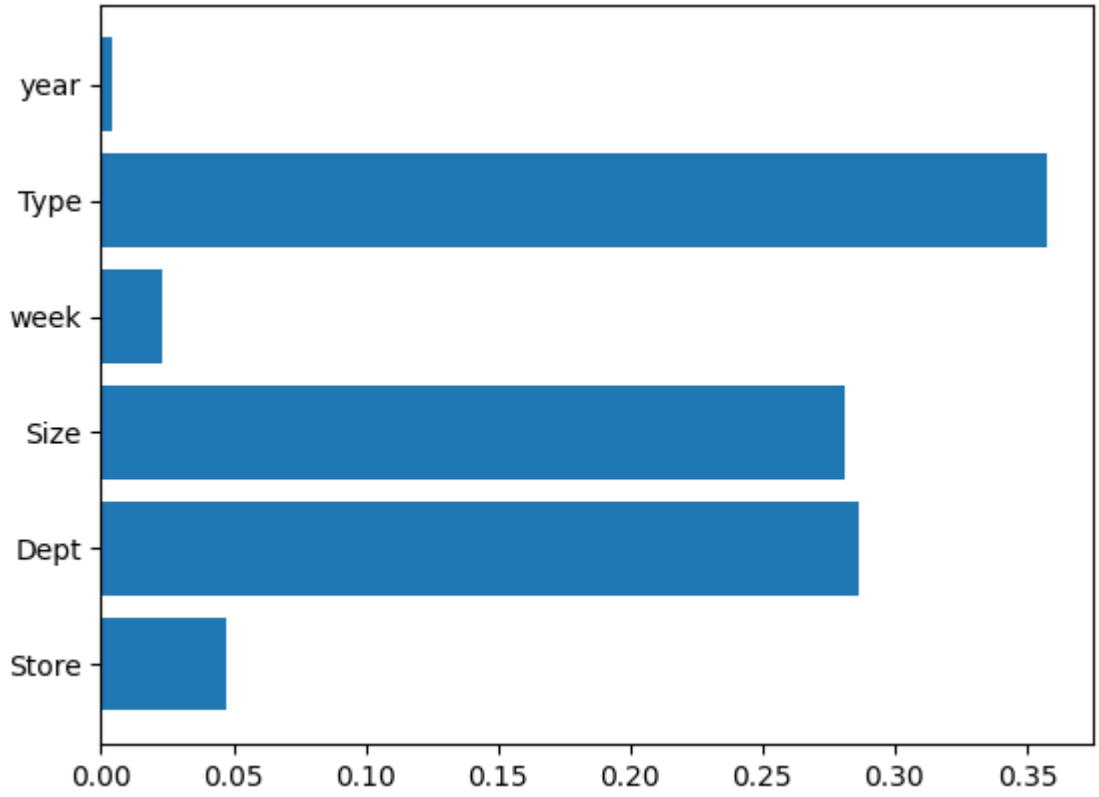
In [111…
```
# fine search for N
N_max = 500
feat= ['Weekly_Sales','Store', 'Dept', 'Size', 'week', 'Type', 'Date', 'year'] #, 'Is
res_n_fine = {}
serie_n_fine = [[] for i in range(6)]
y_fine = range(a,b+1)
```

```
for n in y_fine:
    print('============================')
    print("N=",n)
    tmp = train_boost(df_train, df_val, sumb = False, depth= d_best , N_ = n, features
    res_n_fine[n] = tmp
    for i in range(6):
        serie_n_fine[i].append(tmp[i])
```

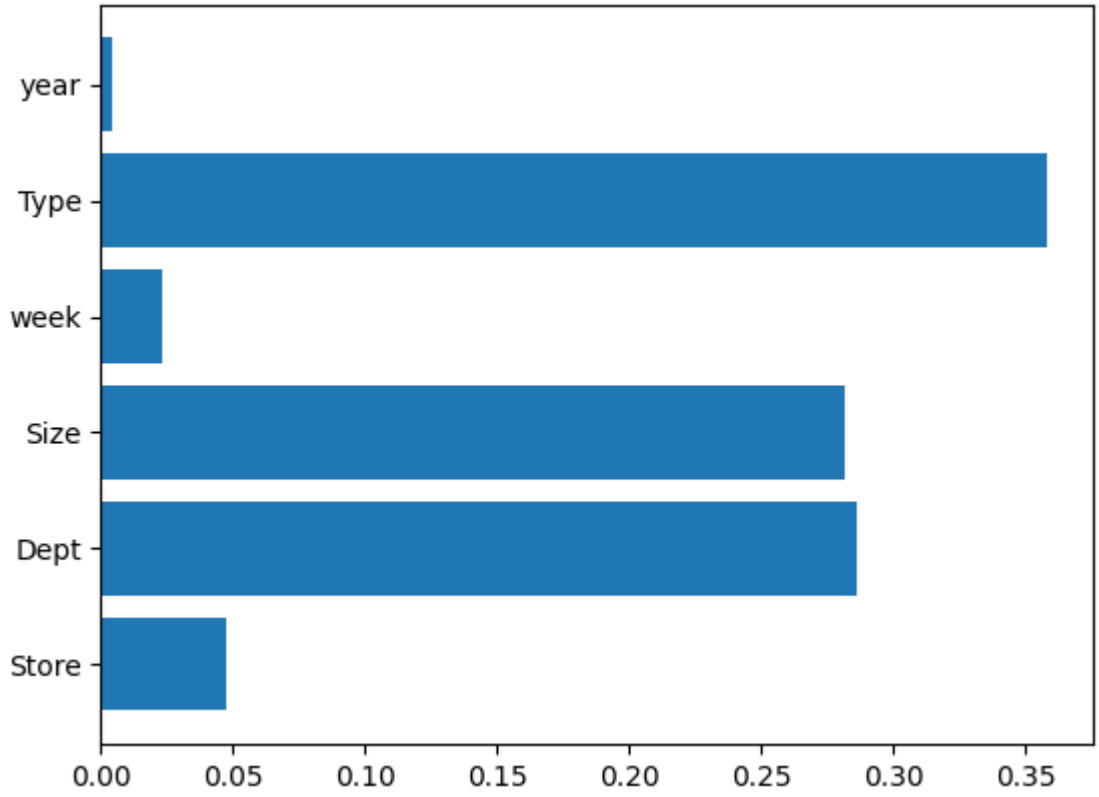
```
============================
N= 361
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995258102829883
MAE=  311.37757893631135
WMAE=  306.34507127170724
```



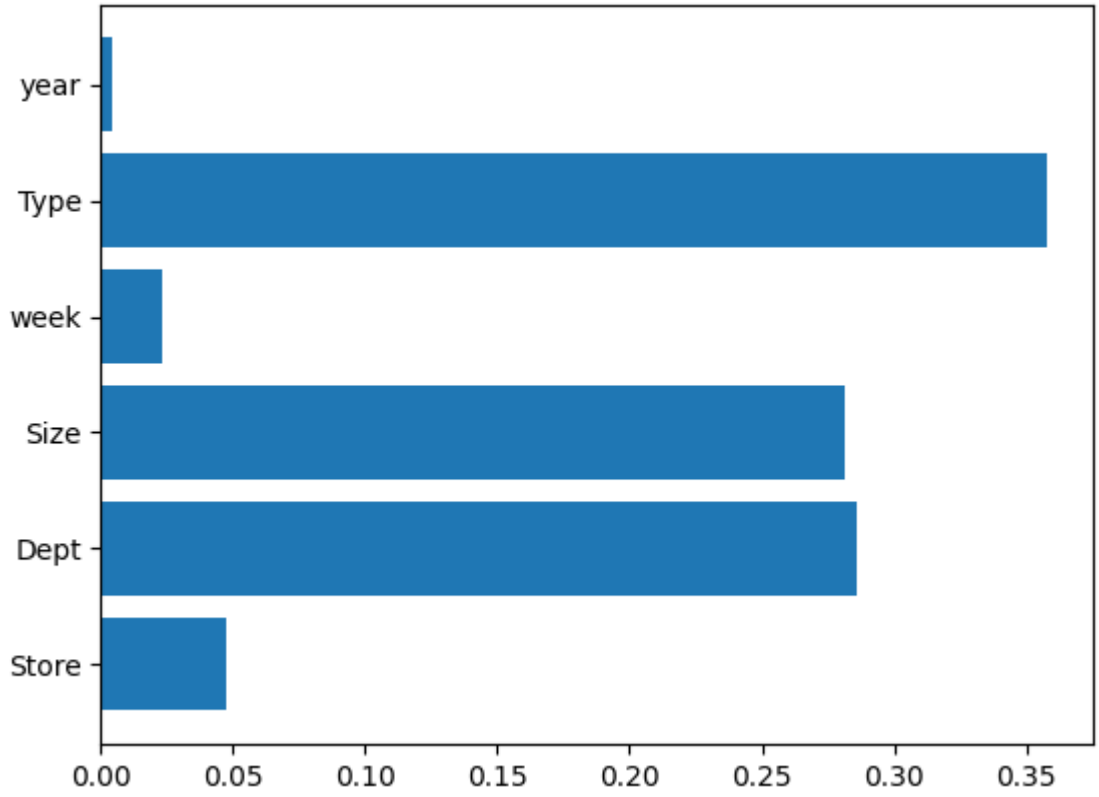

```
performance on evaluation
r2= 0.9882591527821029
MAE= 1072.6098133629455
WMAE= 1155.3387685946914
============================
N= 362
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995276596881608
MAE=  310.7873017391883
WMAE=  305.6229065031985
```



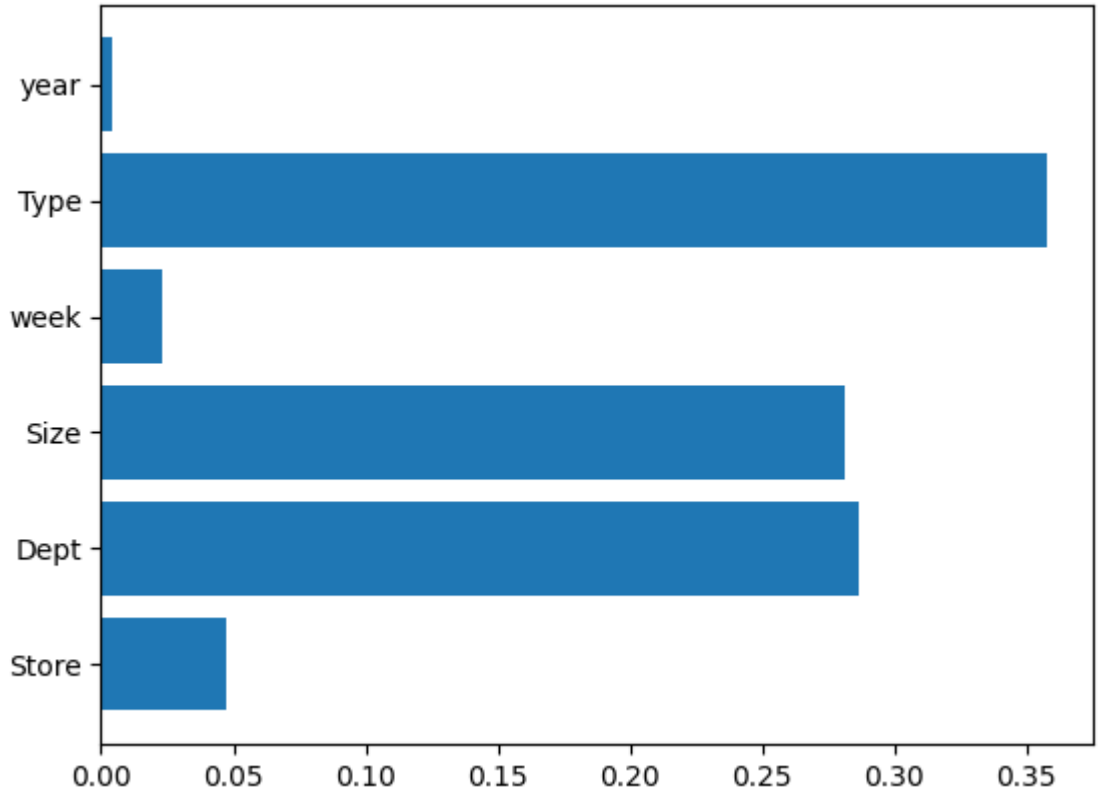

```
performance on evaluation
r2= 0.9882603654610084
MAE= 1072.6005501871057
WMAE= 1155.328937689624
==============================
N= 363
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995303587756105
MAE=  310.007491816902
WMAE=  304.93136037333477
```



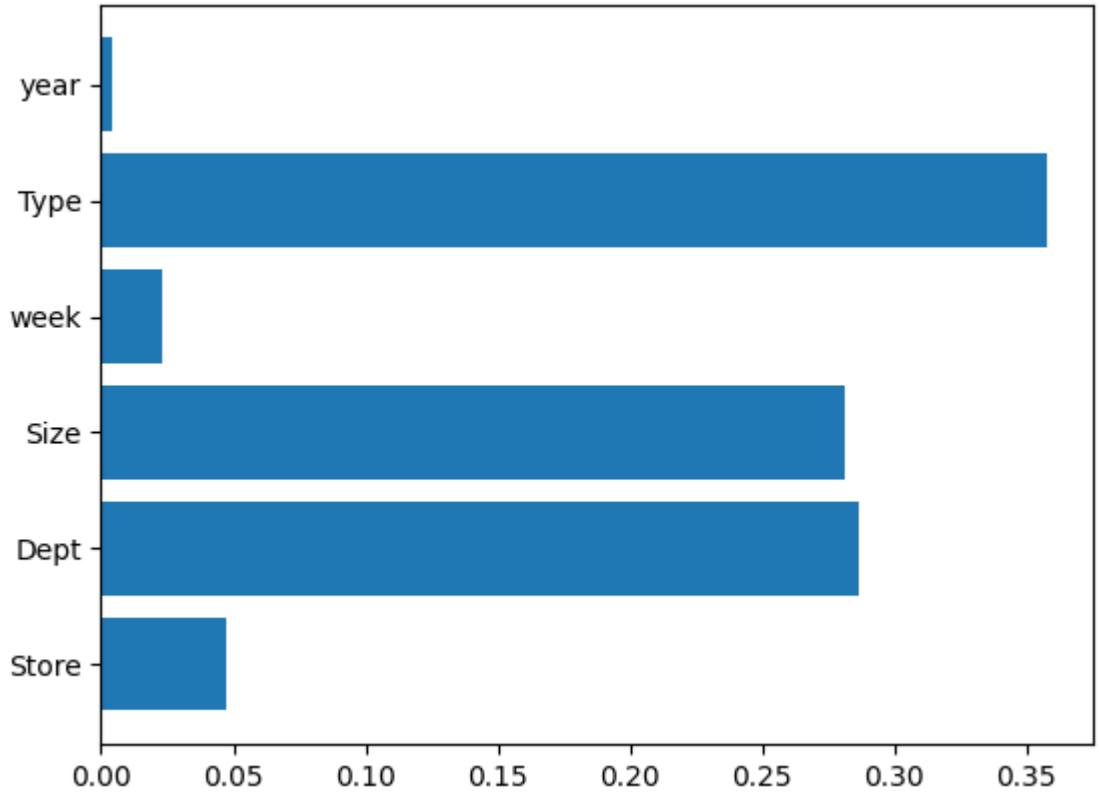

```
performance on evaluation
r2= 0.9882608067685817
MAE= 1072.607298889003
WMAE= 1155.3762153280306
==============================
N= 364
#(train)= 337256 #(val) 84314
performance on training
r2=  0.999533408458645
MAE=  309.09772098399924
WMAE=  304.0395446299404
```



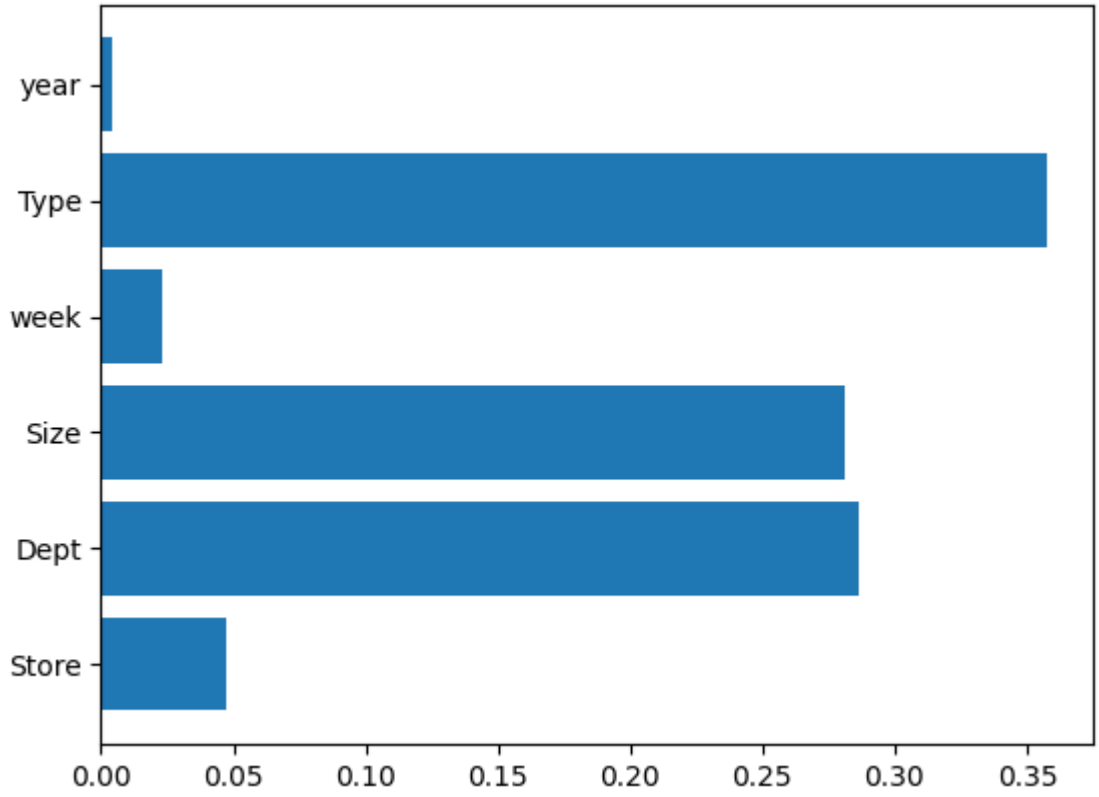

```
performance on evaluation
r2= 0.9882605504089335
MAE= 1072.5838322722561
WMAE= 1155.355367194814
============================
N= 365
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995361240093464
MAE=  308.3017200478813
WMAE=  303.24978341547194
```



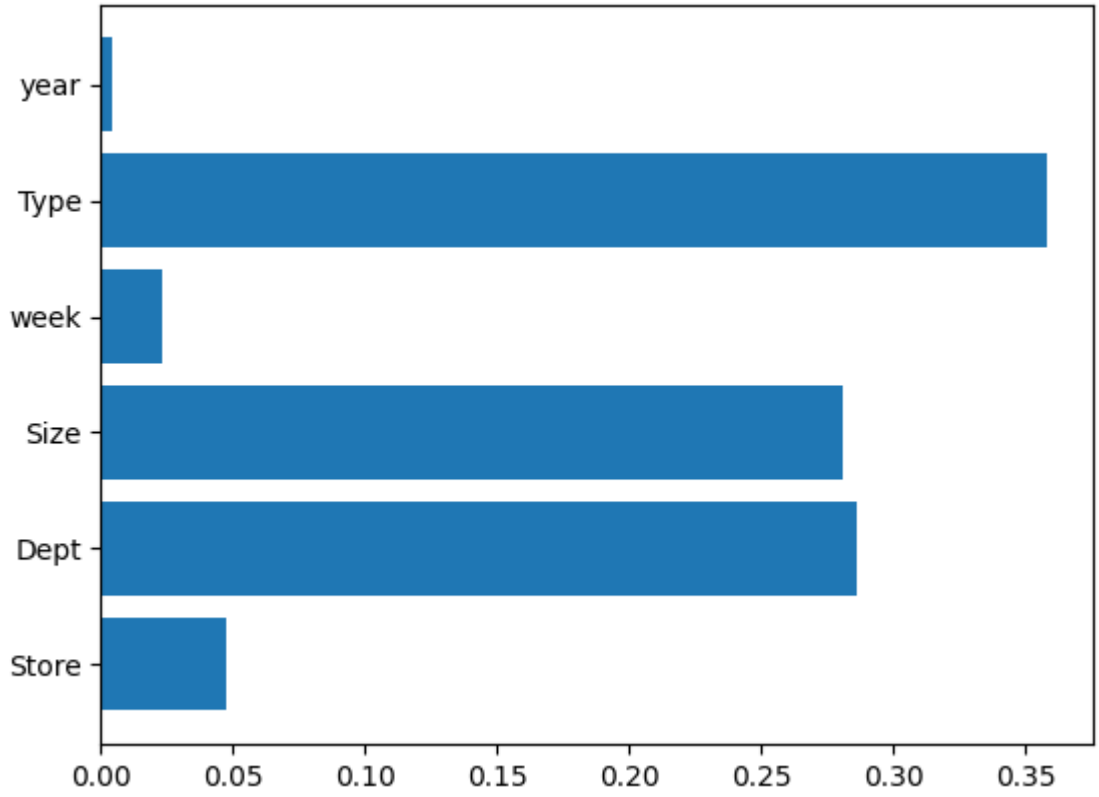

```
performance on evaluation
r2= 0.9882615025201856
MAE= 1072.4742377796313
WMAE= 1155.206974720701
============================
N= 366
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995394511000629
MAE=  307.31979030916307
WMAE=  302.3650213754822
```



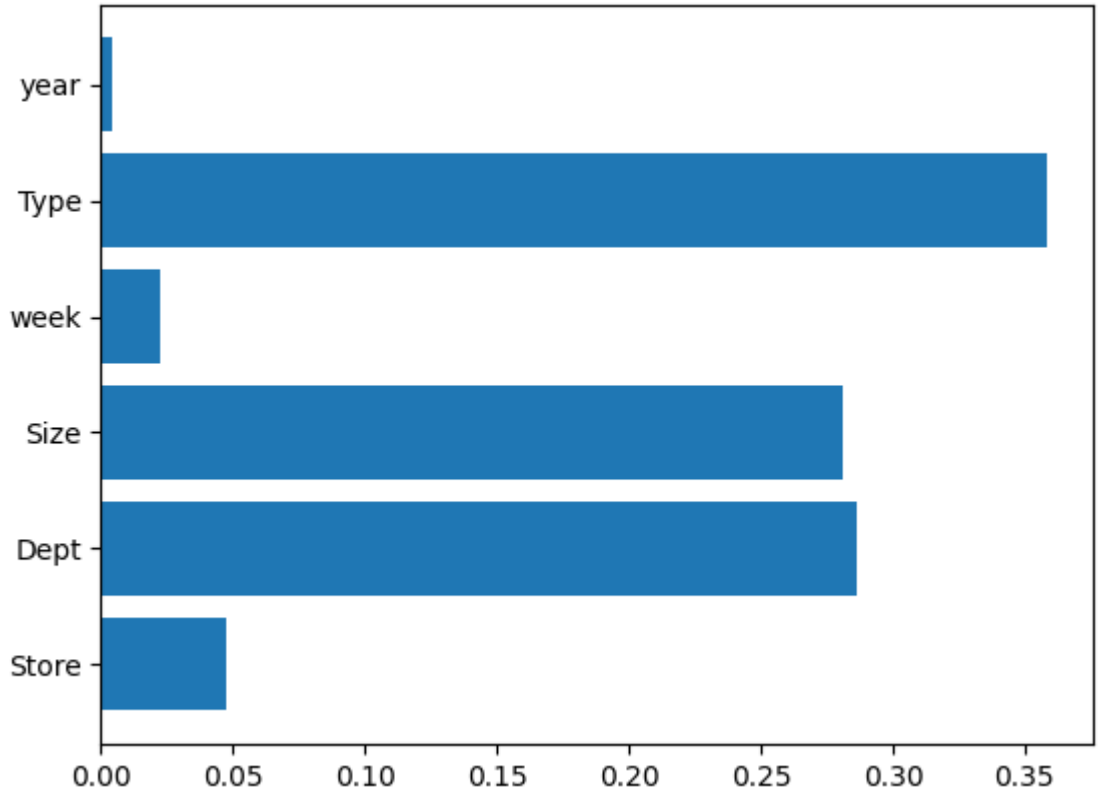

```
performance on evaluation
r2= 0.98826124102411
MAE= 1072.4426021990018
WMAE= 1155.1959180992983
============================
N= 367
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995418907009787
MAE=  306.5641802082296
WMAE=  301.6721346915184
```



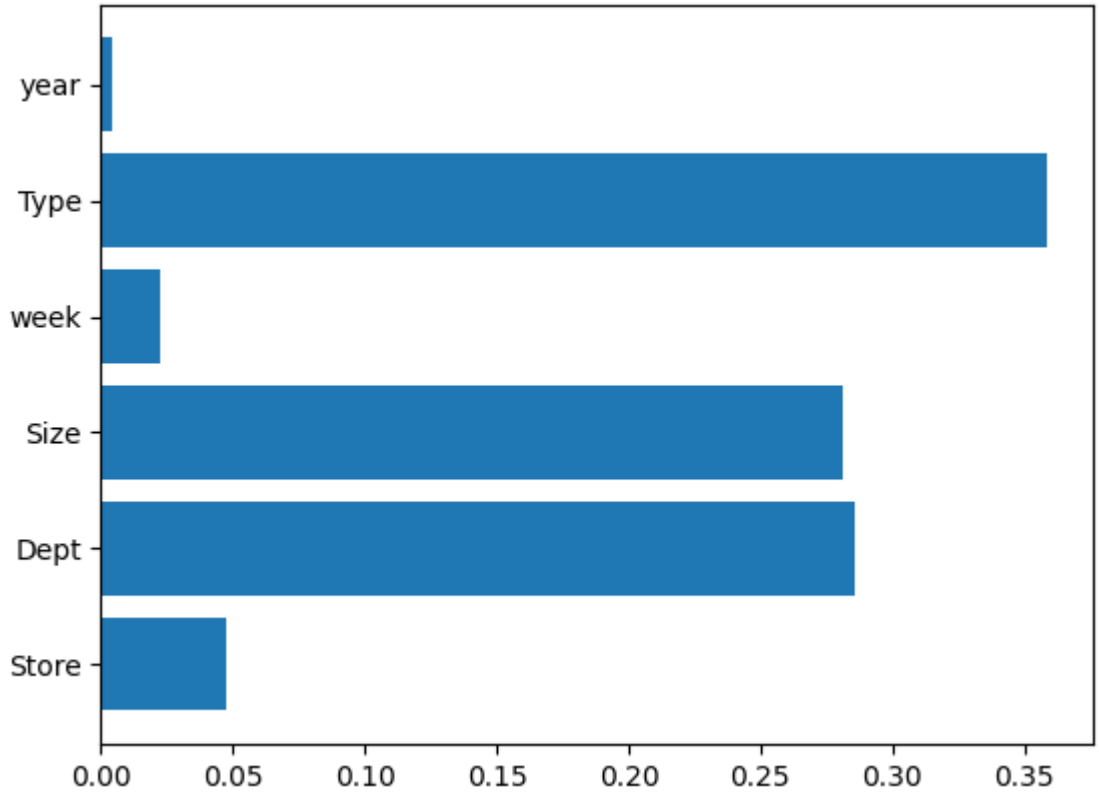

```
performance on evaluation
r2= 0.9882608011006974
MAE= 1072.4563954917703
WMAE= 1155.1932052233751
============================
N= 368
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995441185416519
MAE=  305.8341220670407
WMAE=  300.95491138626926
```



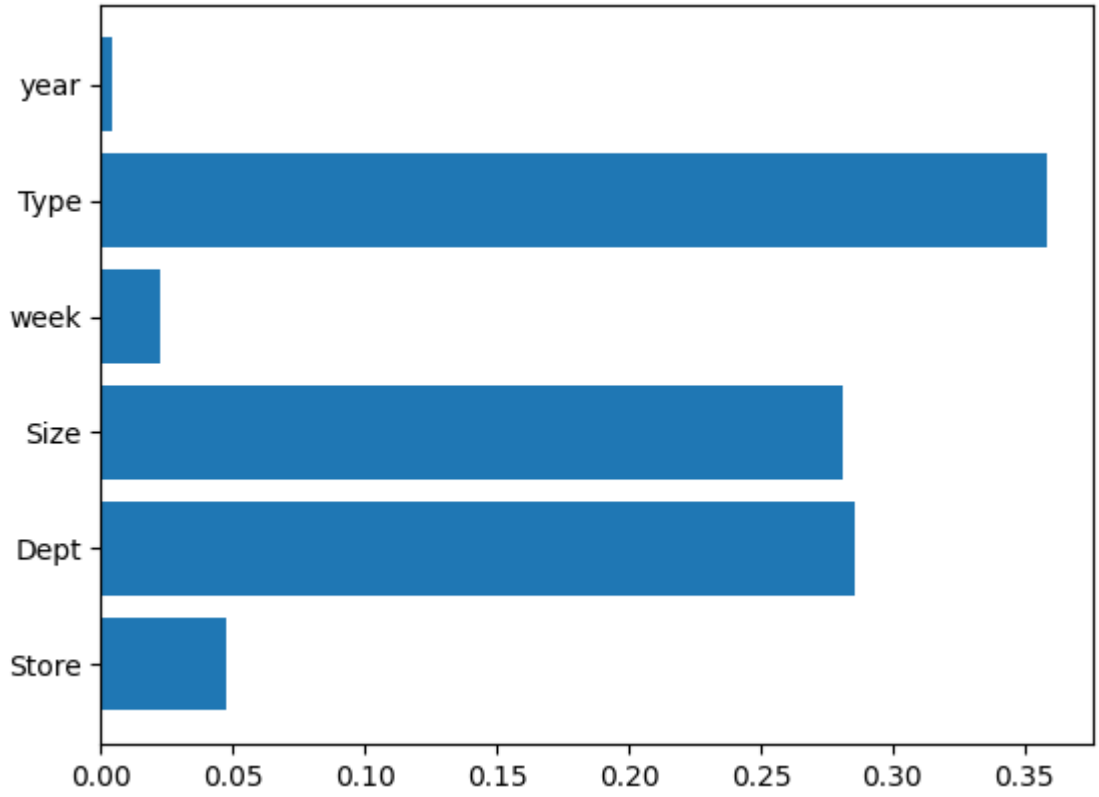

```
performance on evaluation
r2= 0.9882610755052794
MAE= 1072.4521701242
WMAE= 1155.201880532132
==============================
N= 369
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995462707082212
MAE=  305.1247254321656
WMAE=  300.26842939037857
```



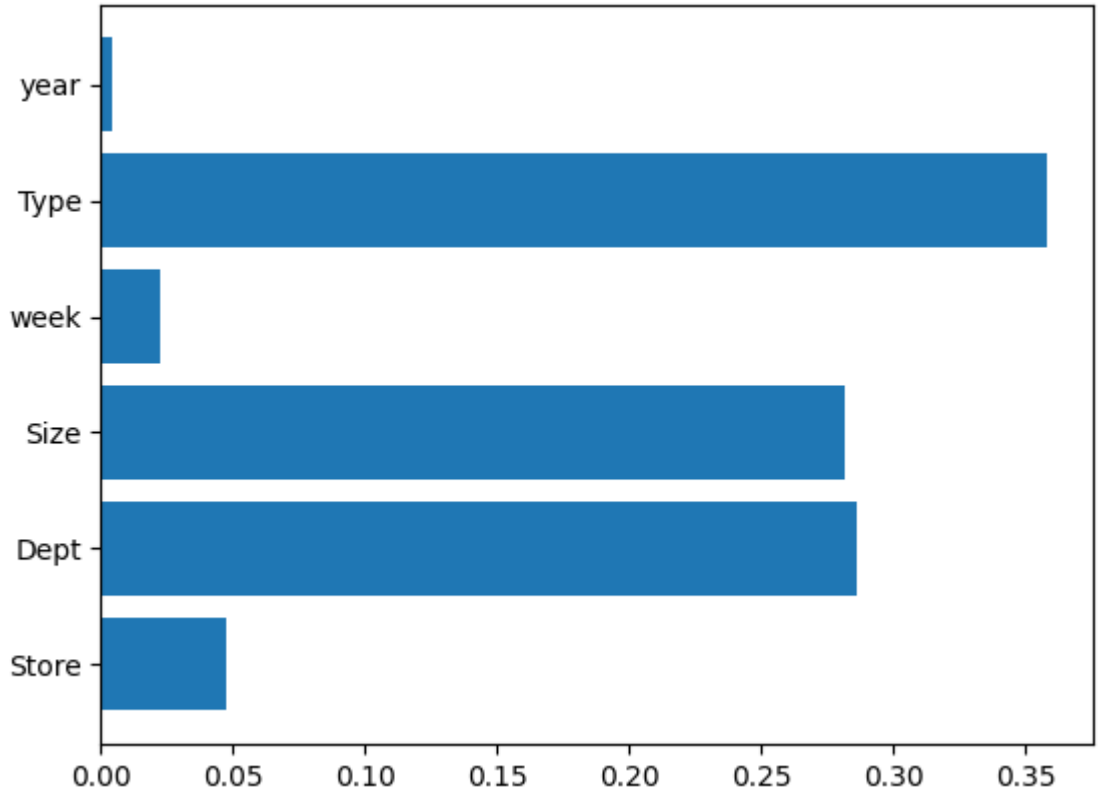

```
performance on evaluation
r2= 0.9882612069961492
MAE= 1072.4645637598157
WMAE= 1155.2157403723622
==============================
N= 370
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995490974160407
MAE=  304.35252374083024
WMAE=  299.46880390272025
```



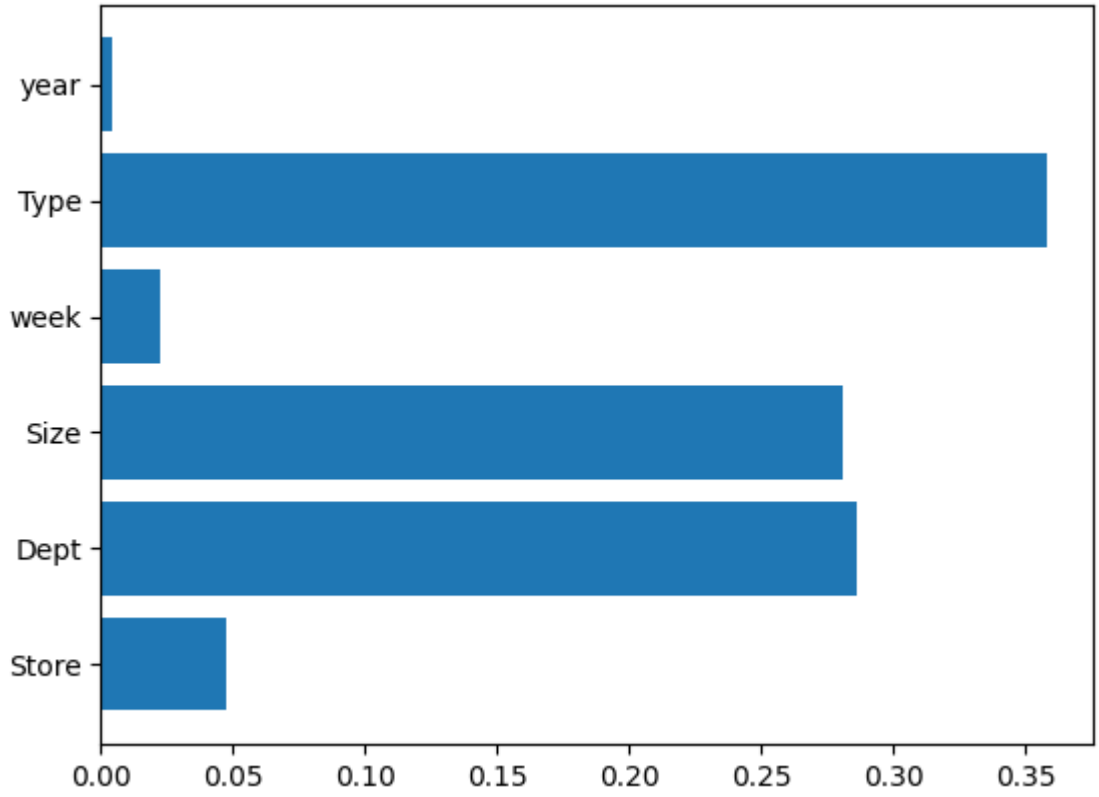

```
performance on evaluation
r2= 0.988261720500447
MAE= 1072.44949117656
WMAE= 1155.205966602962
==============================
N= 371
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995521518771199
MAE=  303.47016136657055
WMAE=  298.5077811772284
```



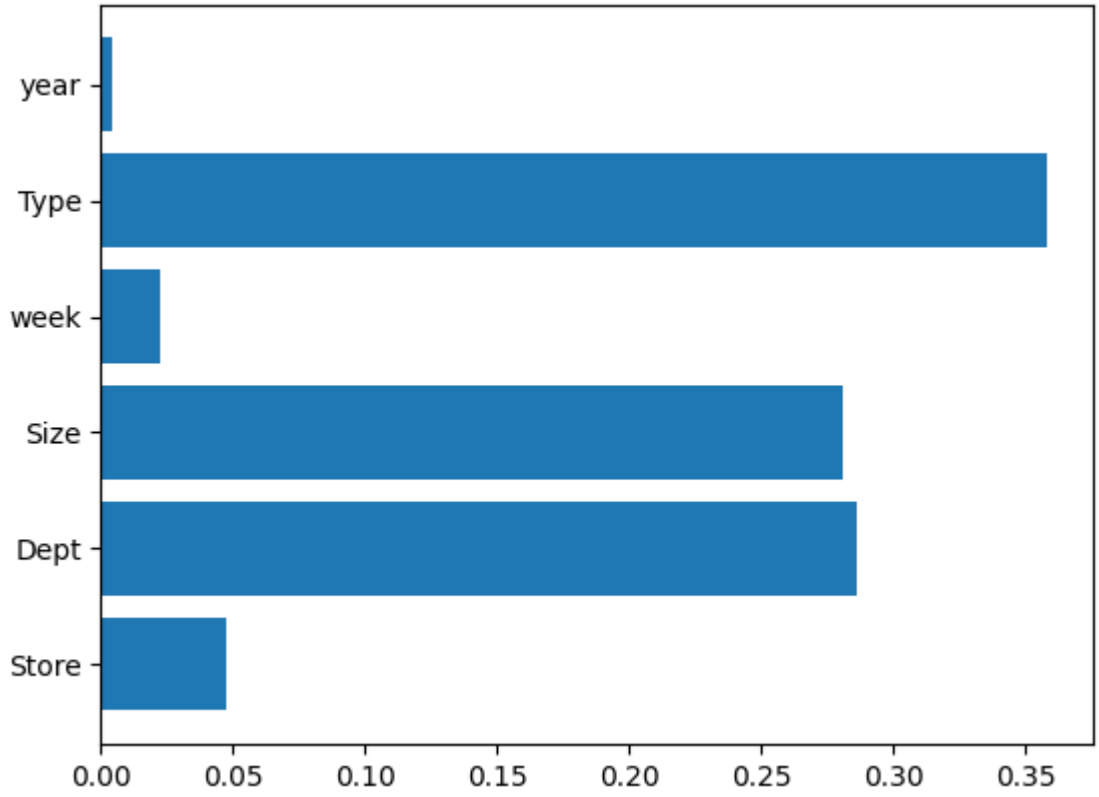

```
performance on evaluation
r2= 0.988259496750306
MAE= 1072.5014451630573
WMAE= 1155.2375156722096
============================
N= 372
#(train)= 337256 #(val) 84314
performance on training
r2=  0.999555510031197
MAE=  302.5299464993045
WMAE=  297.5440065045711
```



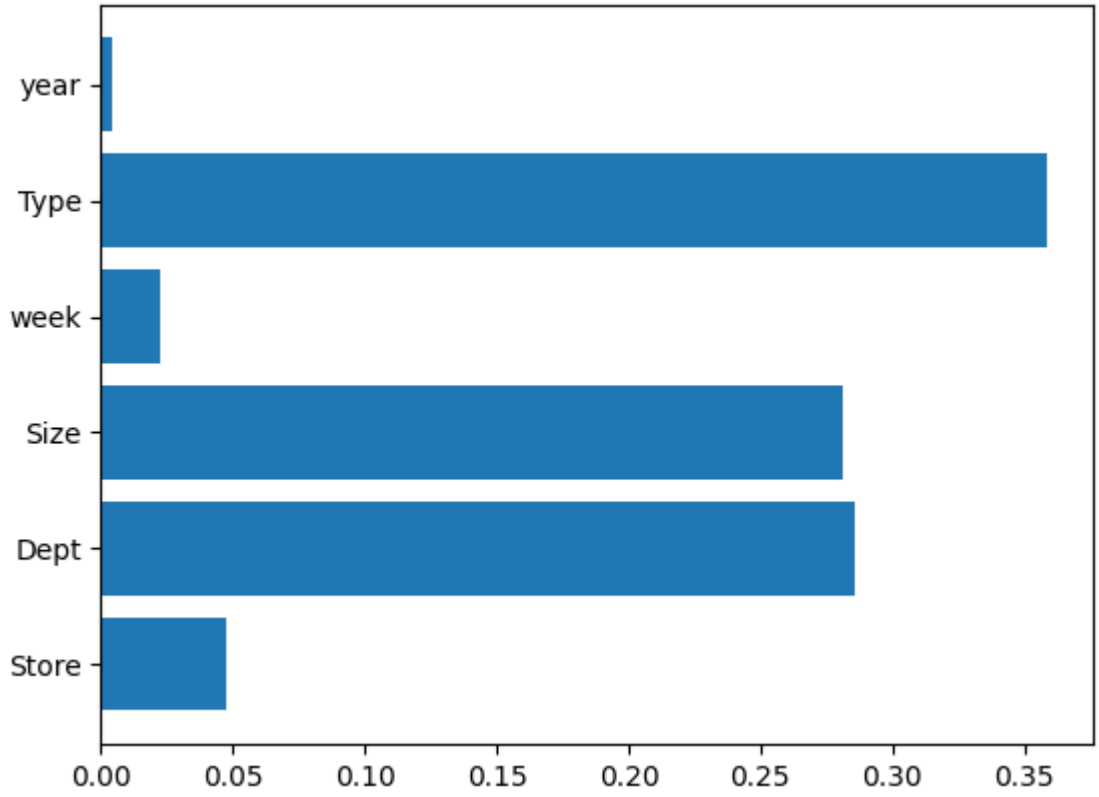

```
performance on evaluation
r2= 0.9882583888440176
MAE= 1072.5020707516965
WMAE= 1155.2490935000033
==============================
N= 373
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995579758774578
MAE=  301.75753039291556
WMAE=  296.85318723465986
```



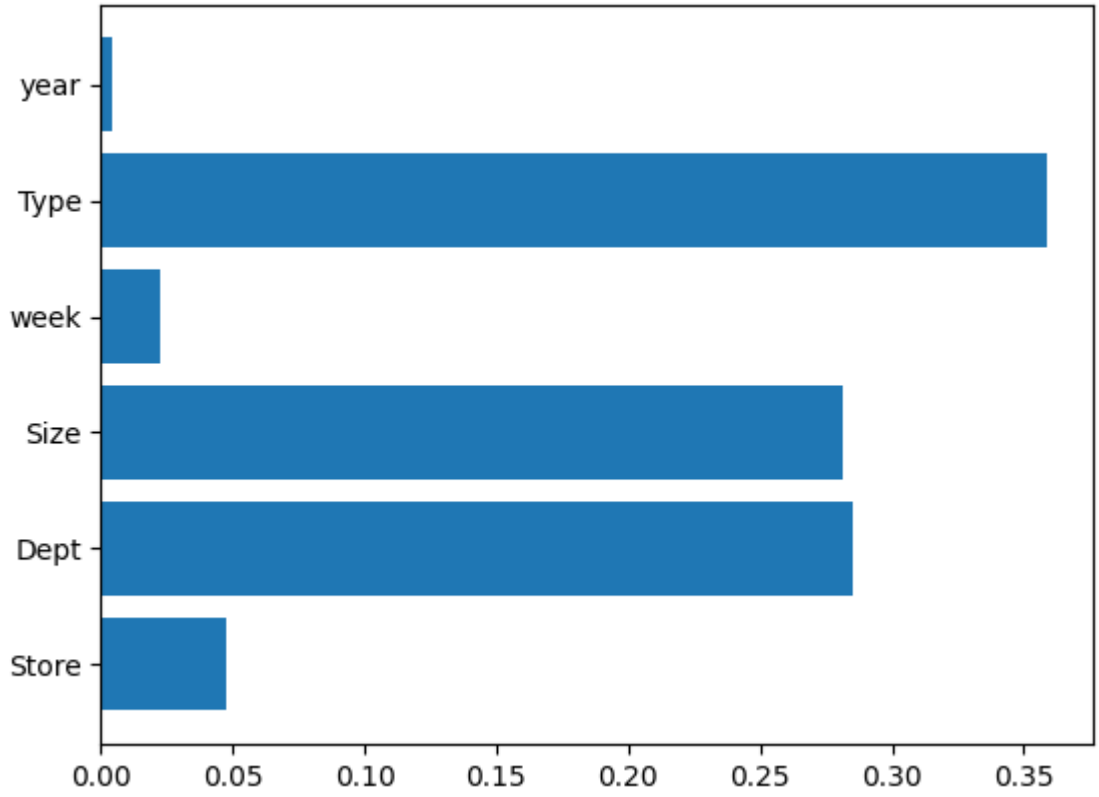

```
performance on evaluation
r2= 0.9882589864882663
MAE= 1072.4749736978797
WMAE= 1155.201626241891
==============================
N= 374
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995603704436681
MAE=  301.09509624451346
WMAE=  296.2277636683148
```



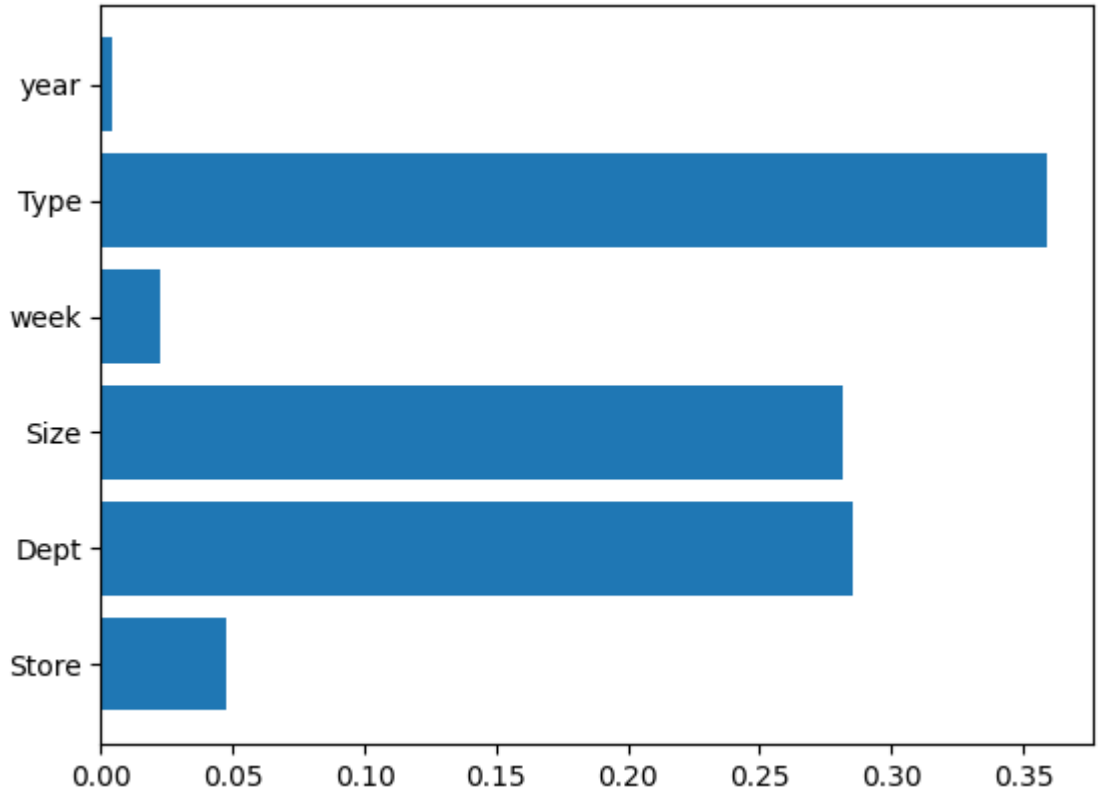

```
performance on evaluation
r2= 0.9882576386096829
MAE= 1072.5161635782194
WMAE= 1155.28108803457
============================
N= 375
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995629857355105
MAE=  300.30640209653734
WMAE=  295.48823491344535
```



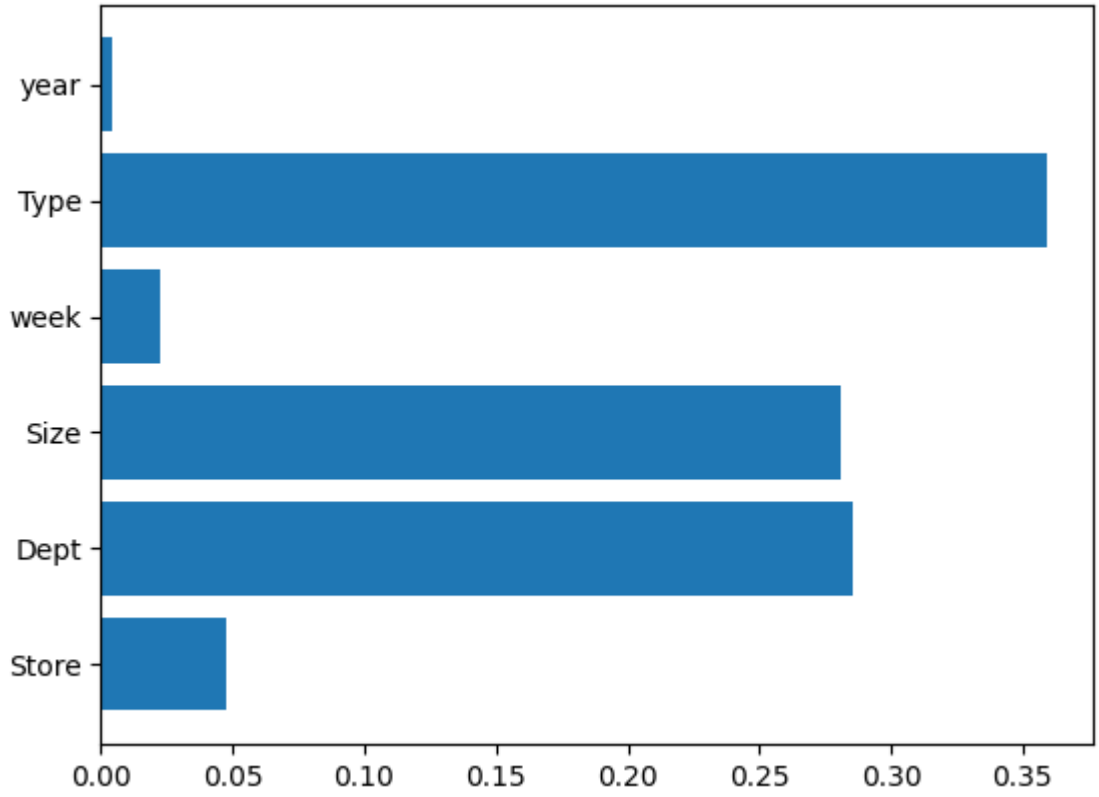

```
performance on evaluation
r2= 0.9882566387505466
MAE= 1072.5291589484993
WMAE= 1155.3016881137437
============================
N= 376
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995654957385393
MAE=  299.617888523538
WMAE=  294.7889862959851
```



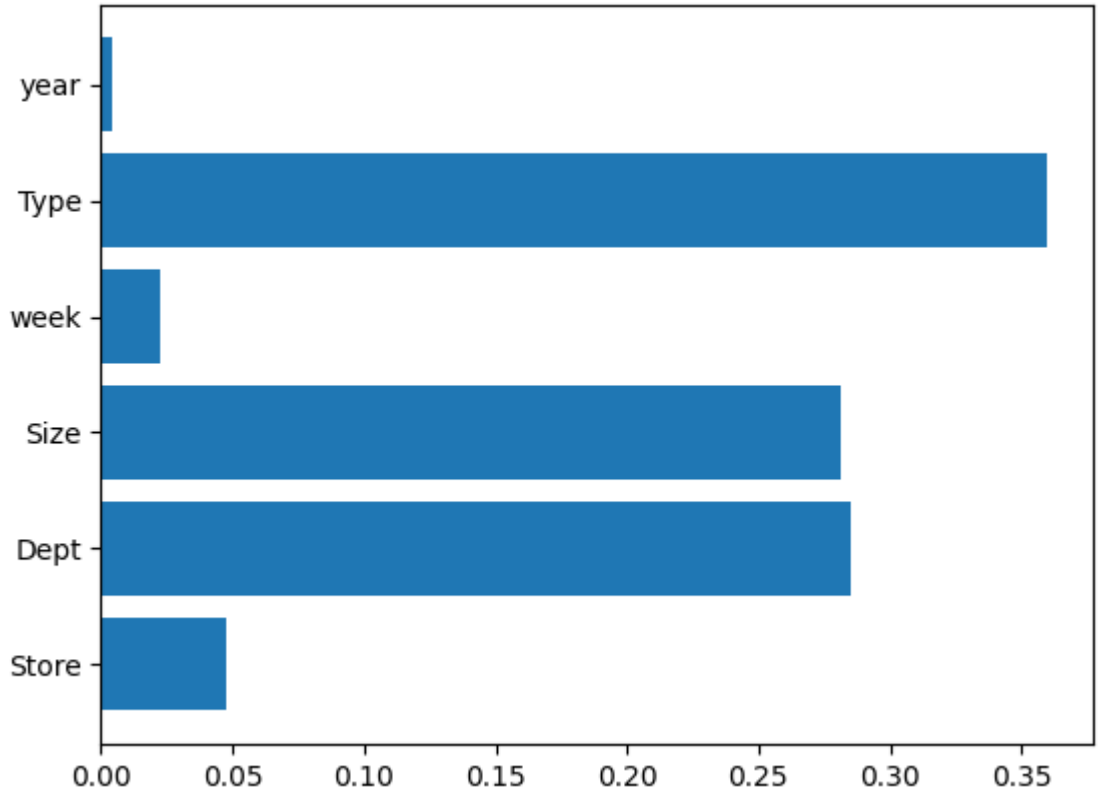

```
performance on evaluation
r2= 0.9882555308849752
MAE= 1072.5002338320592
WMAE= 1155.297960062153
============================
N= 377
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995683027376498
MAE=  298.77783871745777
WMAE=  294.03402225701564
```



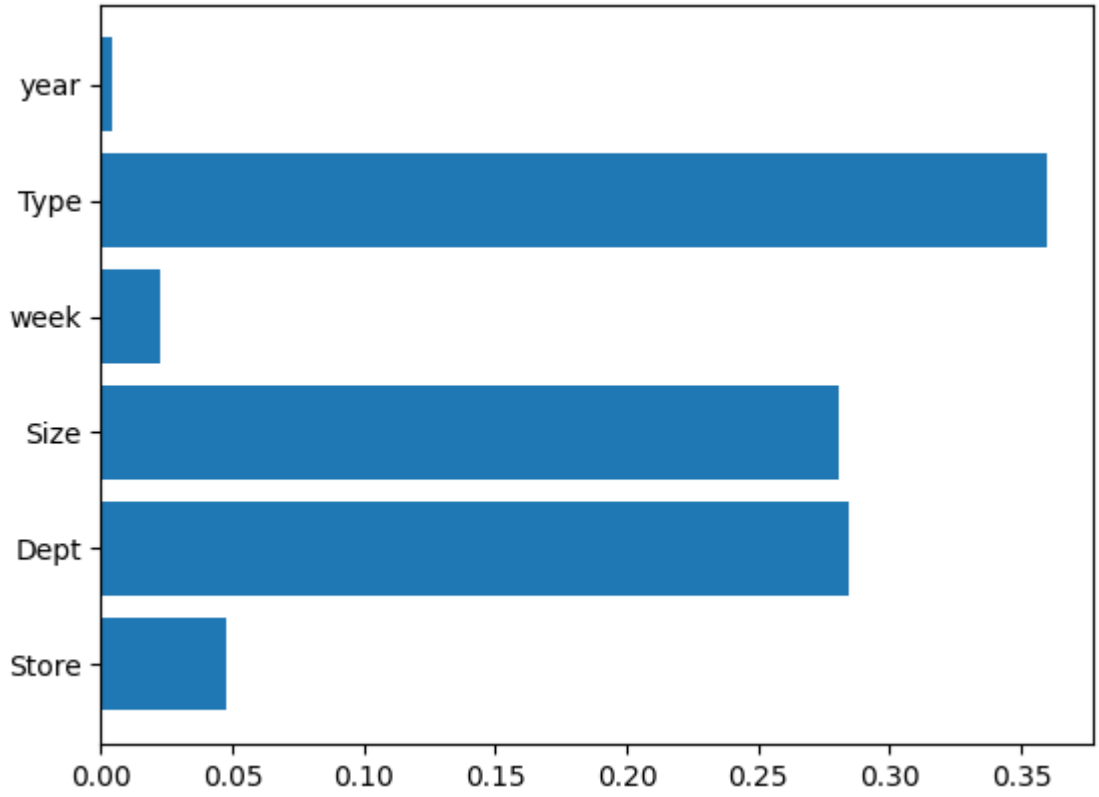

```
performance on evaluation
r2= 0.9882563195126627
MAE= 1072.5059877184565
WMAE= 1155.2704382333513
============================
N= 378
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995707026336966
MAE=  298.0278595547049
WMAE=  293.3609148478608
```



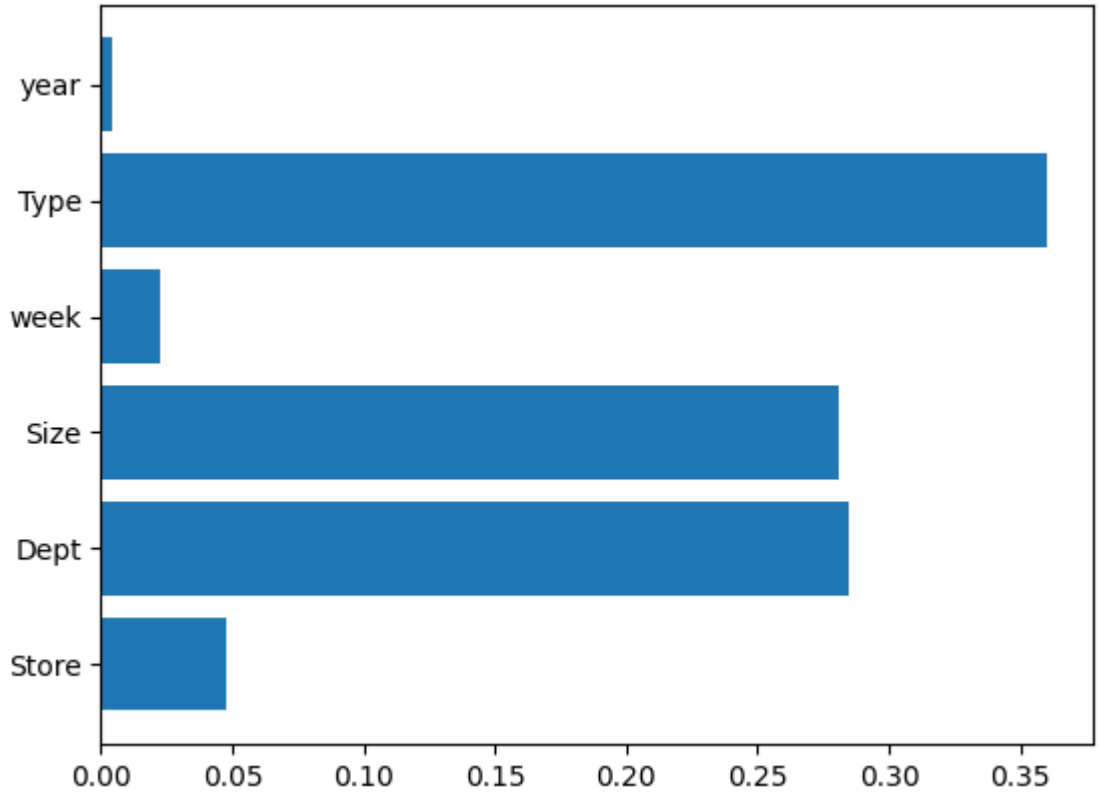

```
performance on evaluation
r2= 0.9882566640451755
MAE= 1072.5208272816049
WMAE= 1155.3041699489374
============================
N= 379
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995733389183239
MAE=  297.2533668042888
WMAE=  292.53986736265597
```



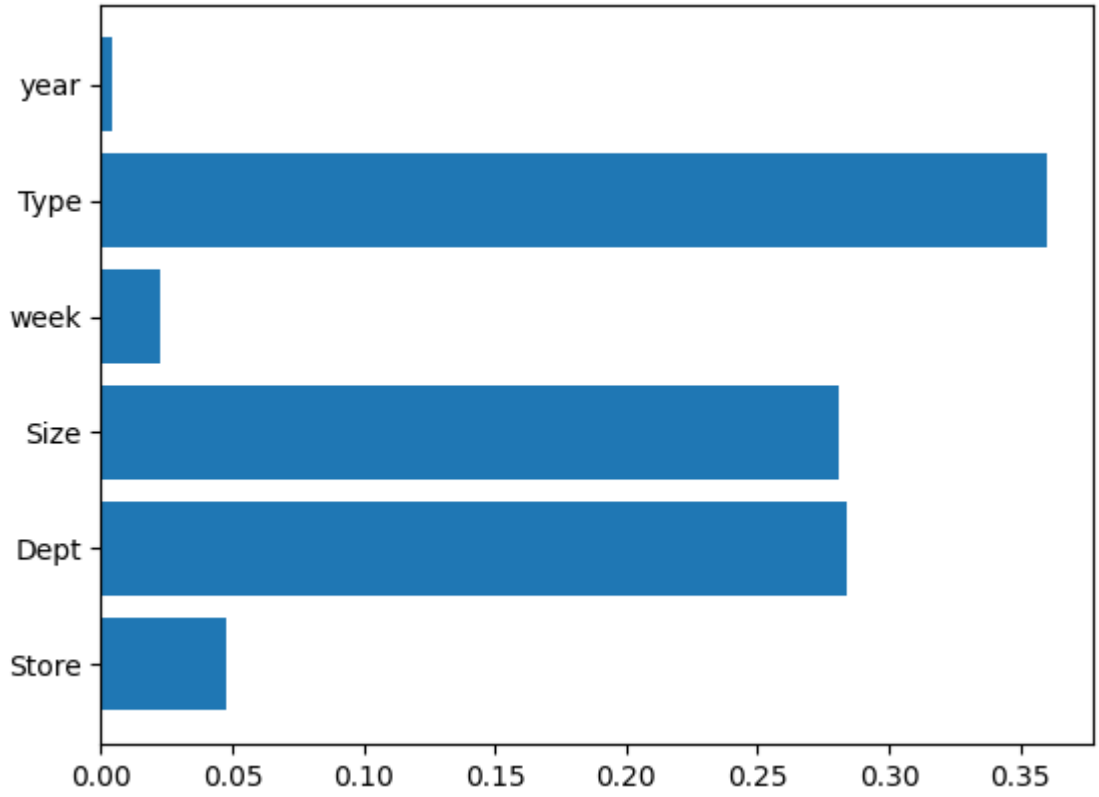

```
performance on evaluation
r2= 0.9882568443322468
MAE= 1072.6004771310288
WMAE= 1155.4123879741283
============================
N= 380
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995757095445442
MAE=  296.5171062136304
WMAE=  291.8967727839516
```



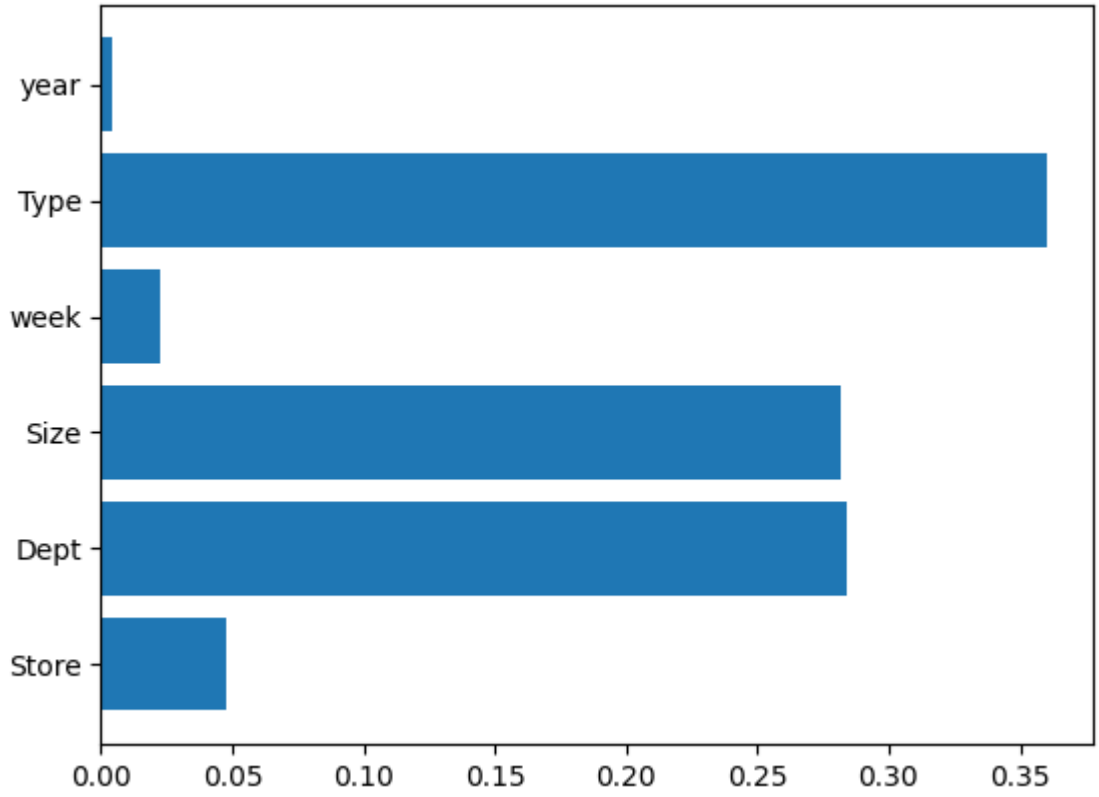

```
performance on evaluation
r2= 0.9882560331854363
MAE= 1072.6553722233434
WMAE= 1155.445462941421
============================
N= 381
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995775297553522
MAE=  295.88355551254904
WMAE=  291.29120150399746
```



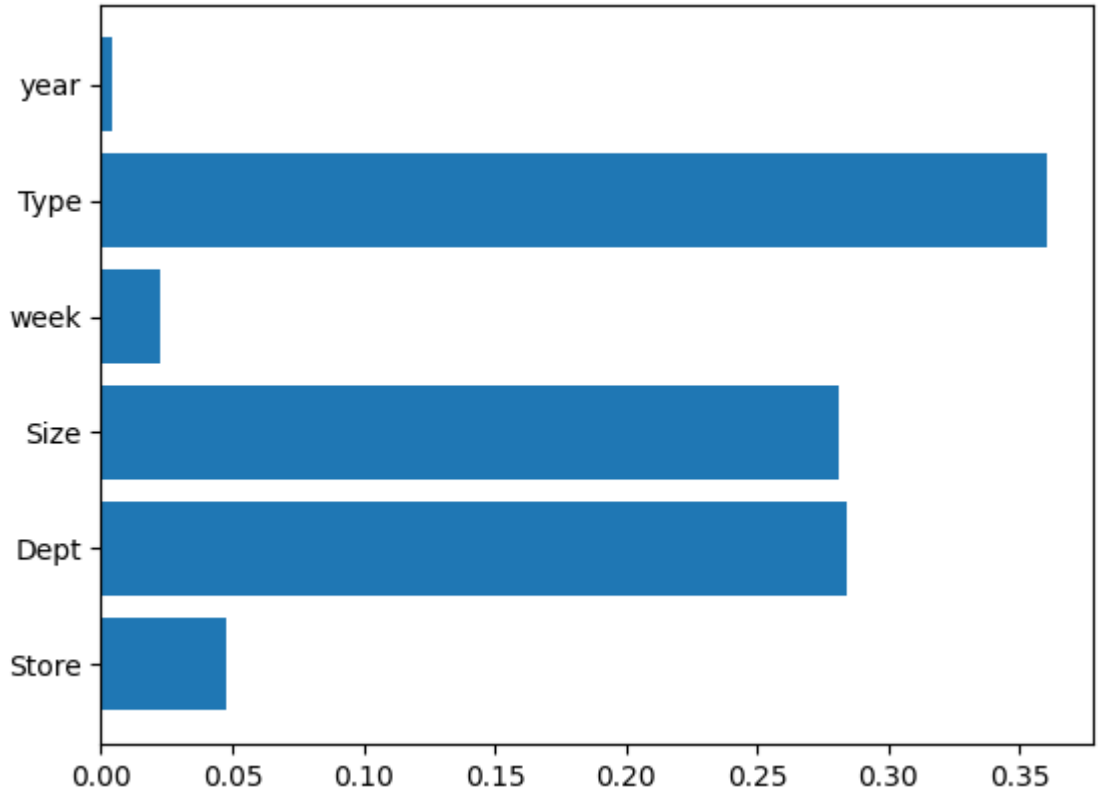

```
performance on evaluation
r2= 0.9882565518647185
MAE= 1072.6062261150878
WMAE= 1155.4074089119233
==============================
N= 382
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995803444123109
MAE=  295.00730816217657
WMAE=  290.4196952716948
```



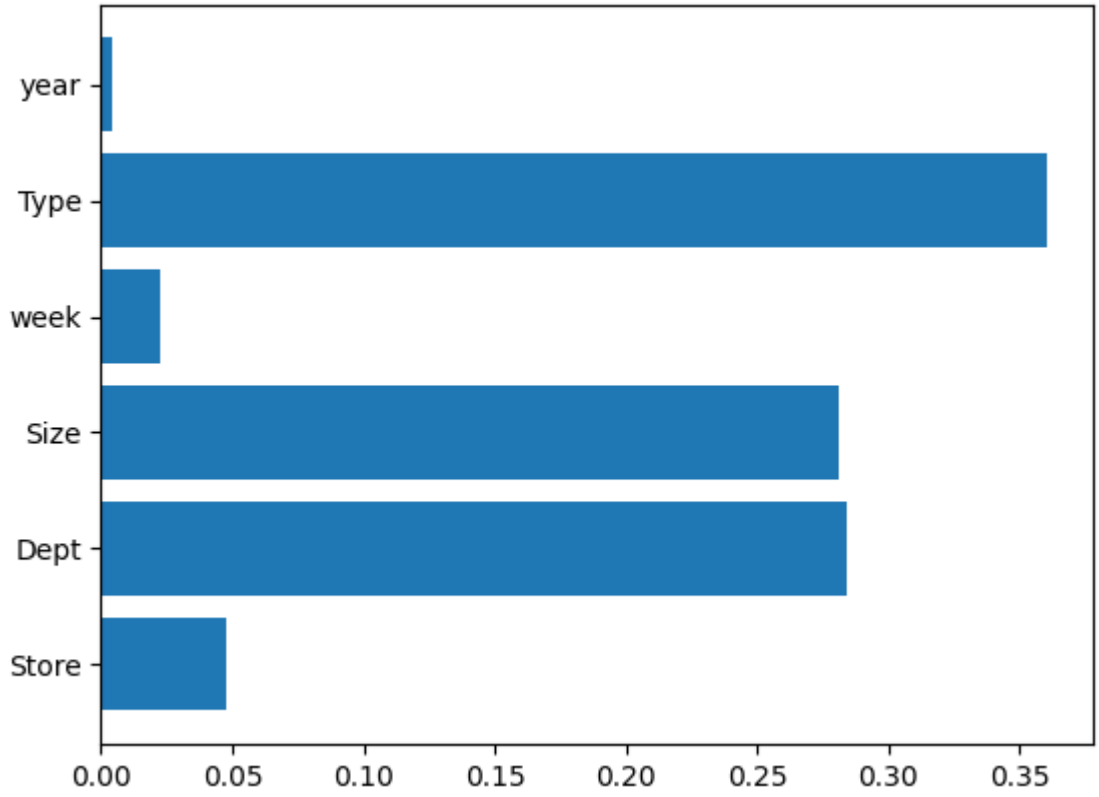

```
performance on evaluation
r2= 0.9882565604499985
MAE= 1072.5907603488188
WMAE= 1155.3666745627963
==============================
N= 383
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995825935026197
MAE=  294.35493031141175
WMAE=  289.71821938806096
```



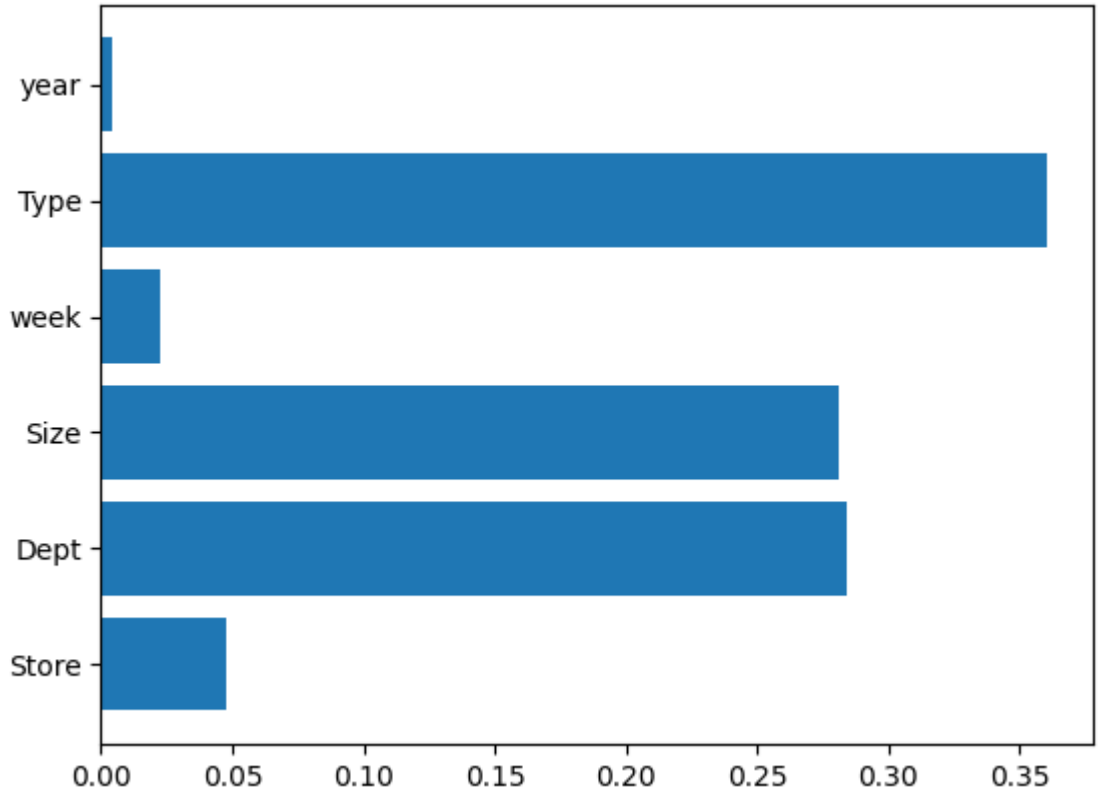

```
performance on evaluation
r2= 0.988256618951566
MAE= 1072.5535534115163
WMAE= 1155.3457088589487
============================
N= 384
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995854808259725
MAE=  293.46756664398845
WMAE=  288.86898092196816
```



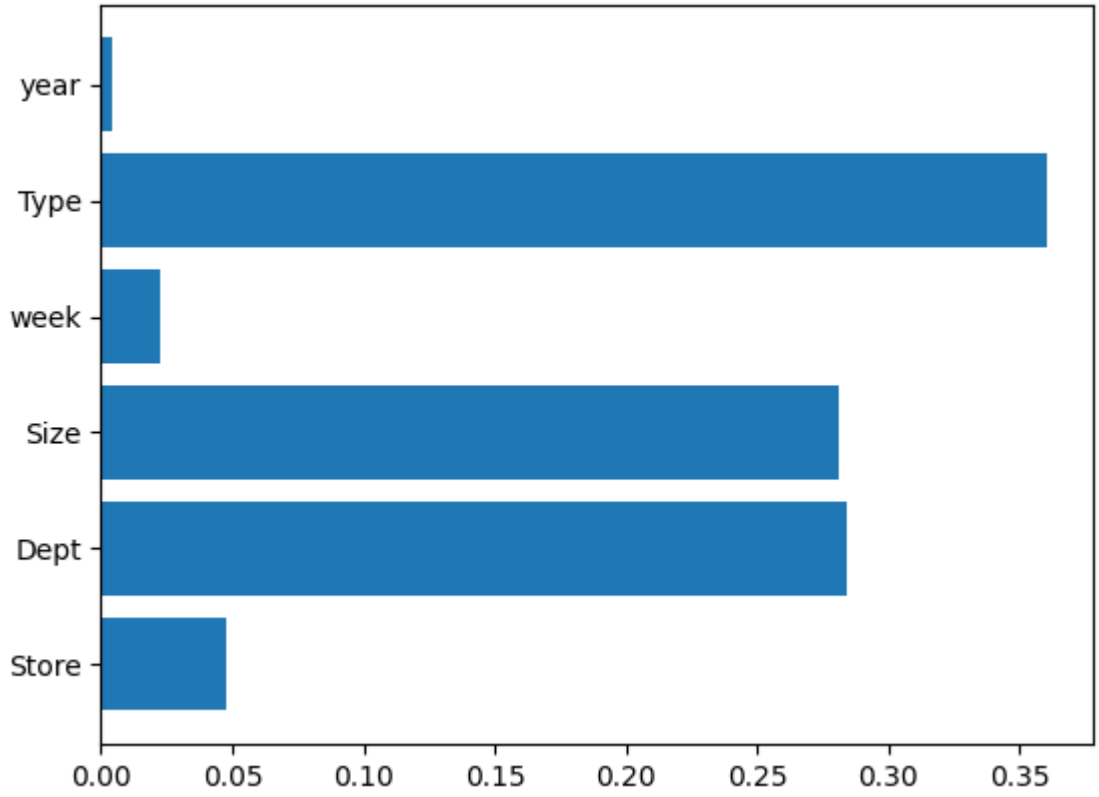

```
performance on evaluation
r2= 0.9882572258012737
MAE= 1072.5639133810441
WMAE= 1155.3371409124823
============================
N= 385
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995880641650003
MAE=  292.76651467887393
WMAE=  288.15719613515057
```



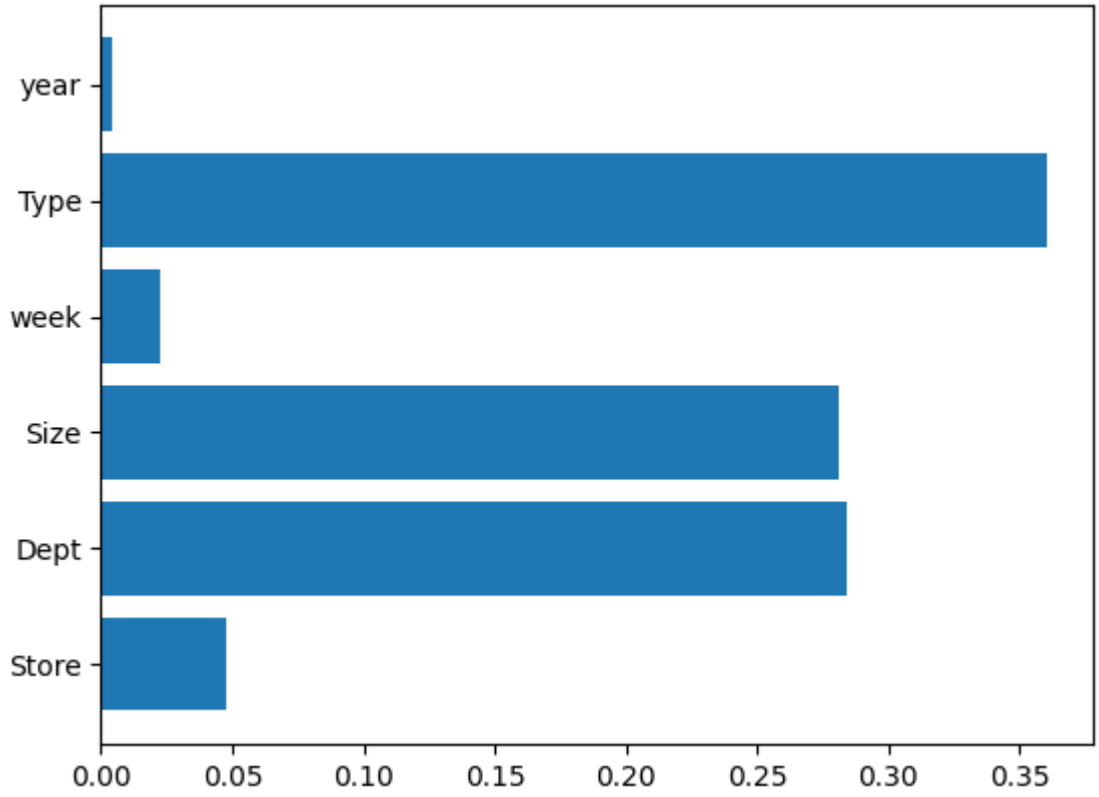

```
performance on evaluation
r2= 0.9882580149611124
MAE= 1072.5403375616474
WMAE= 1155.2531658538817
==============================
N= 386
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995907470026979
MAE=  291.96676244533035
WMAE=  287.1057427703986
```



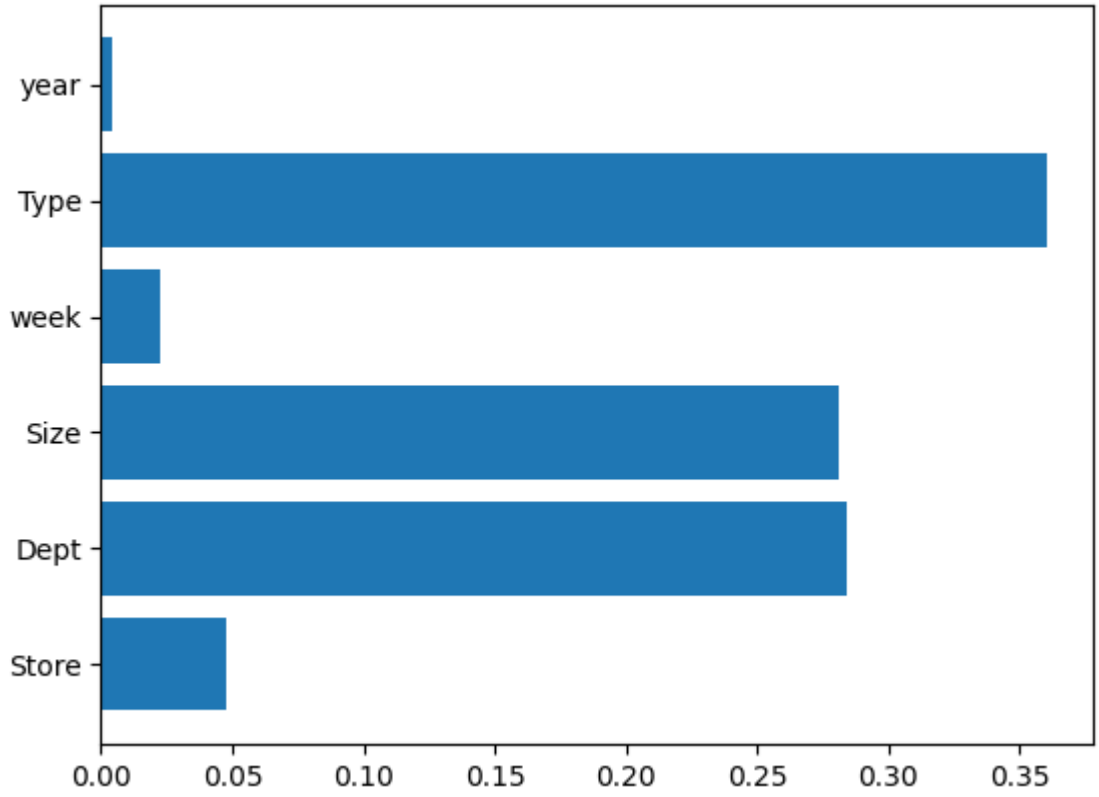

```
performance on evaluation
r2= 0.9882579331572432
MAE= 1072.5128990261044
WMAE= 1155.1570084579446
============================
N= 387
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995927233281082
MAE=  291.2505467306635
WMAE=  286.3786683161304
```



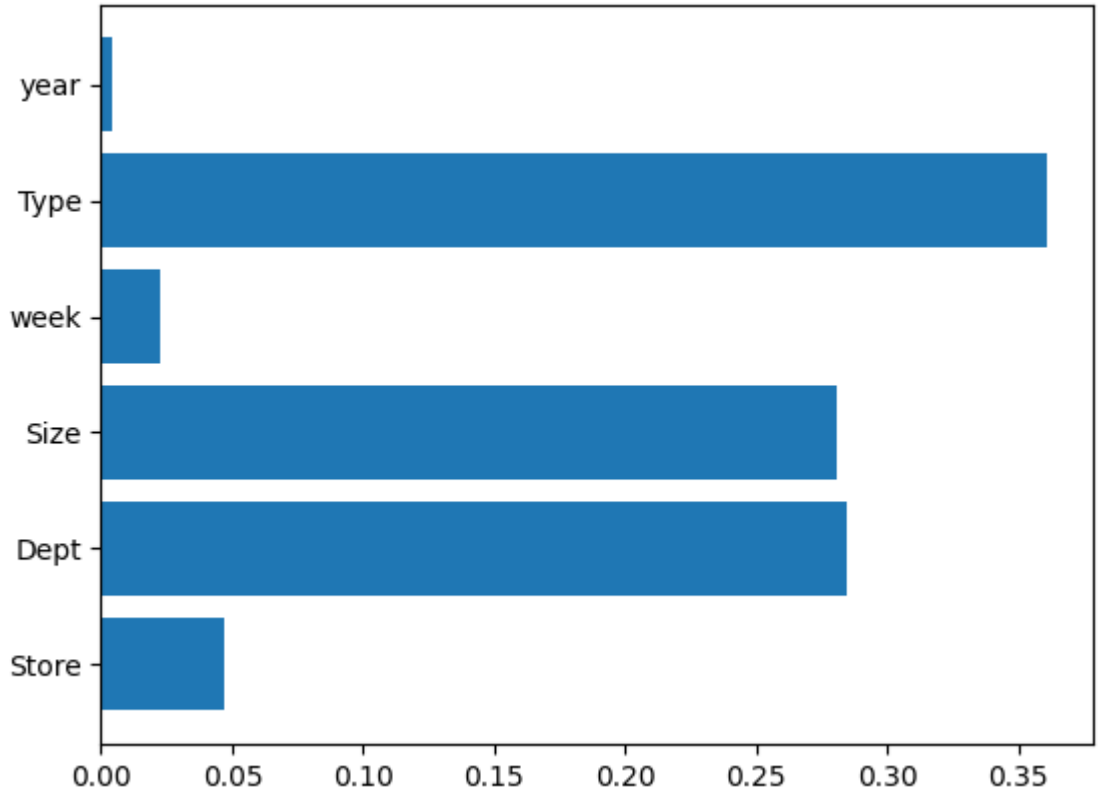

```
performance on evaluation
r2= 0.9882584773872195
MAE= 1072.4948103967677
WMAE= 1155.152640774108
============================
N= 388
#(train)= 337256 #(val) 84314
performance on training
r2=  0.999594953167064
MAE=  290.52884611268775
WMAE=  285.63073766171203
```



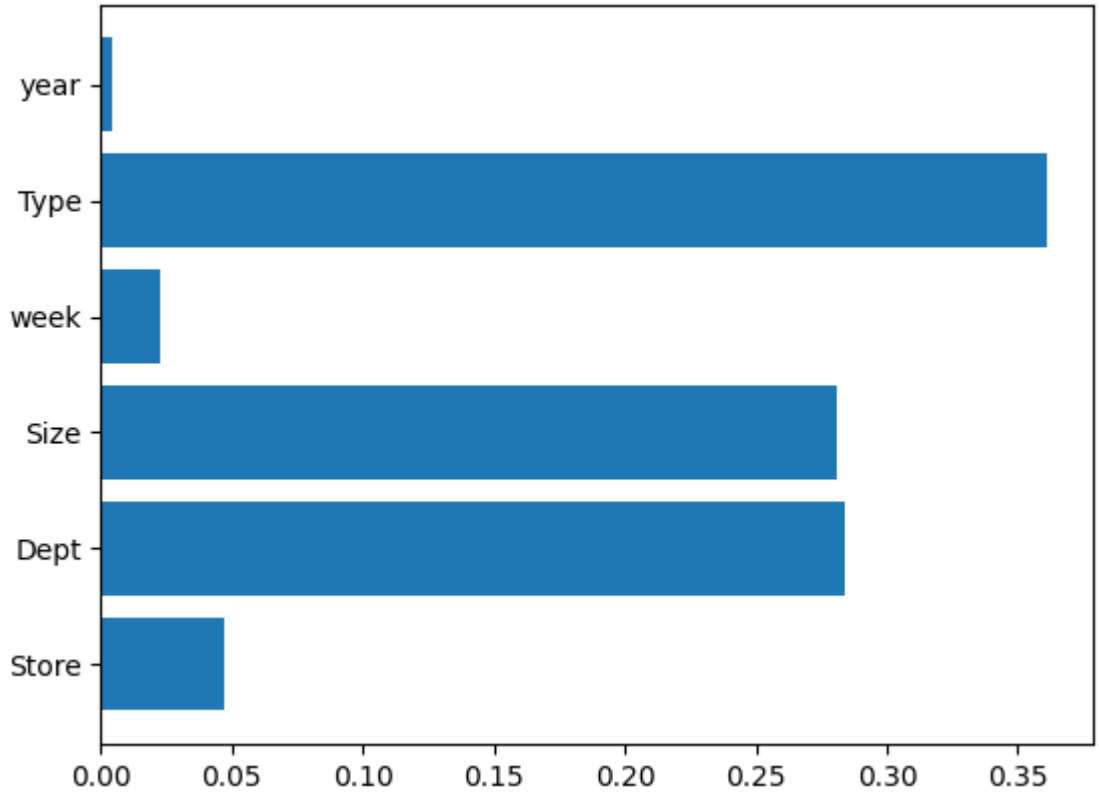

```
performance on evaluation
r2= 0.9882575150529068
MAE= 1072.5066481820102
WMAE= 1155.1459995986559
==============================
N= 389
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995970546428712
MAE=  289.8411596497167
WMAE=  284.9123732378352
```



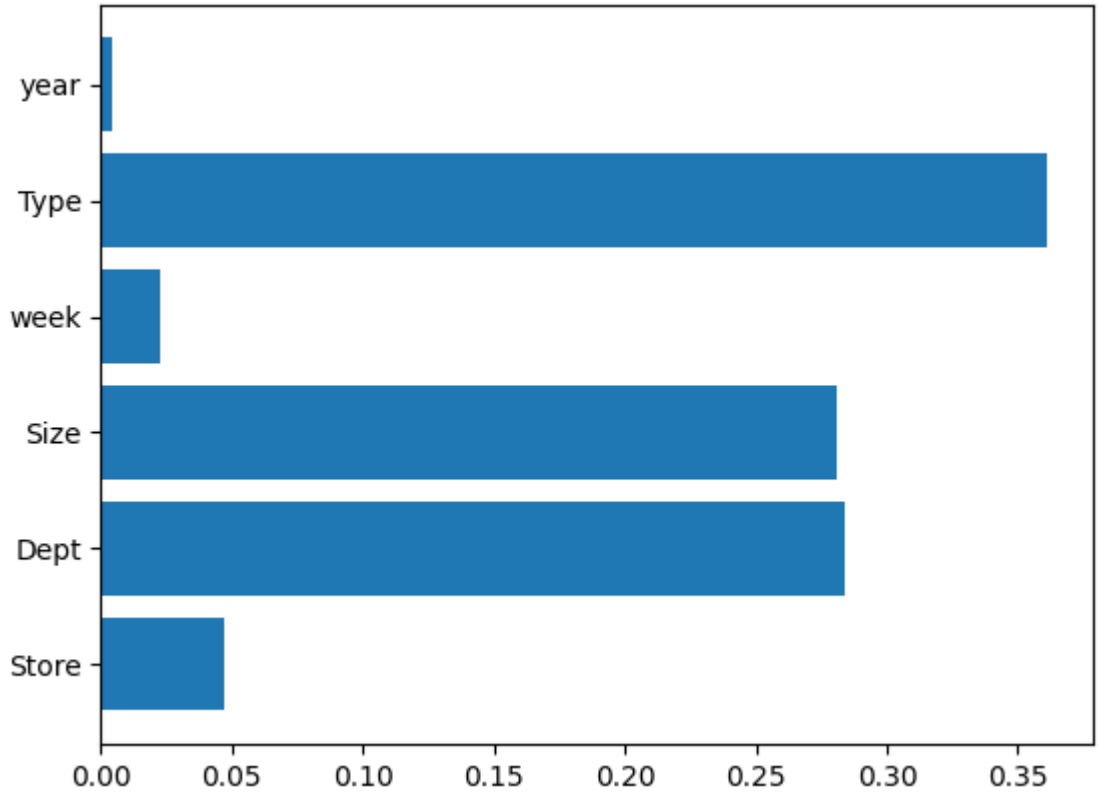

```
performance on evaluation
r2= 0.9882571021466088
MAE= 1072.5333757344124
WMAE= 1155.184715613933
==============================
N= 390
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9995990305662273
MAE=  289.2146807259307
WMAE=  284.24144312345135
```



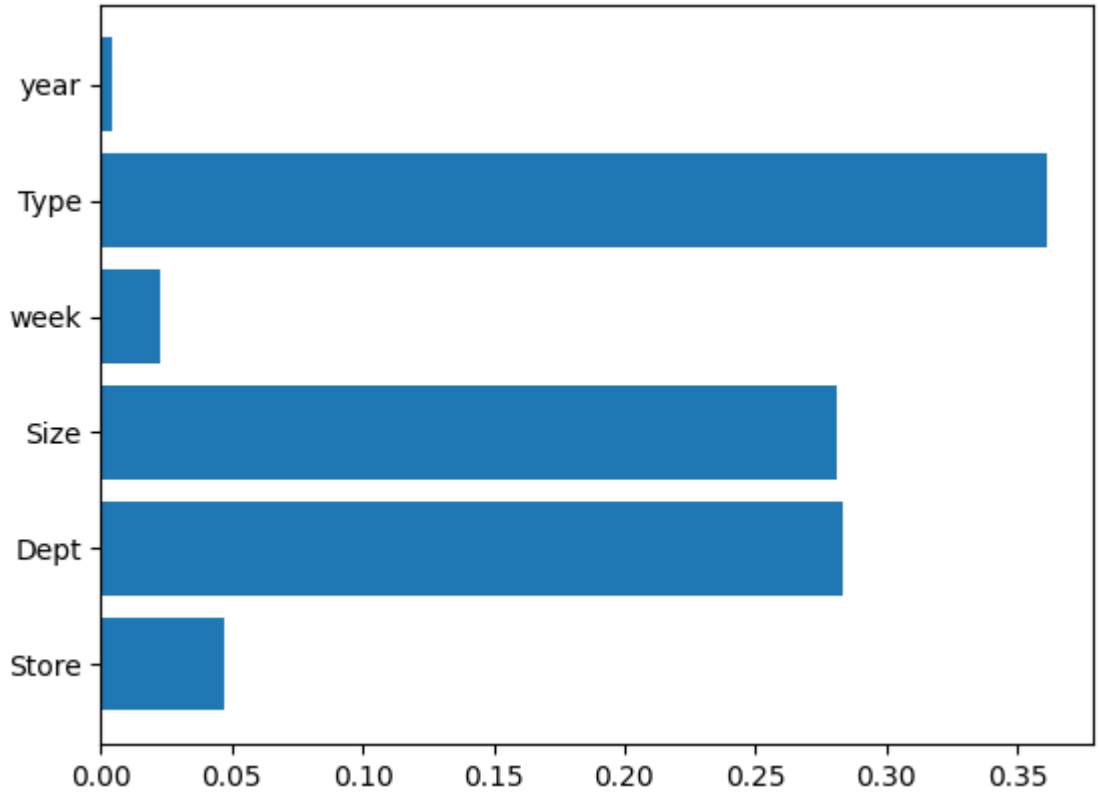

```
performance on evaluation
r2= 0.9882574837446878
MAE= 1072.5357938877742
WMAE= 1155.1645111156442
==============================
N= 391
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996020930131069
MAE=  288.31089581814166
WMAE=  283.3782003209255
```



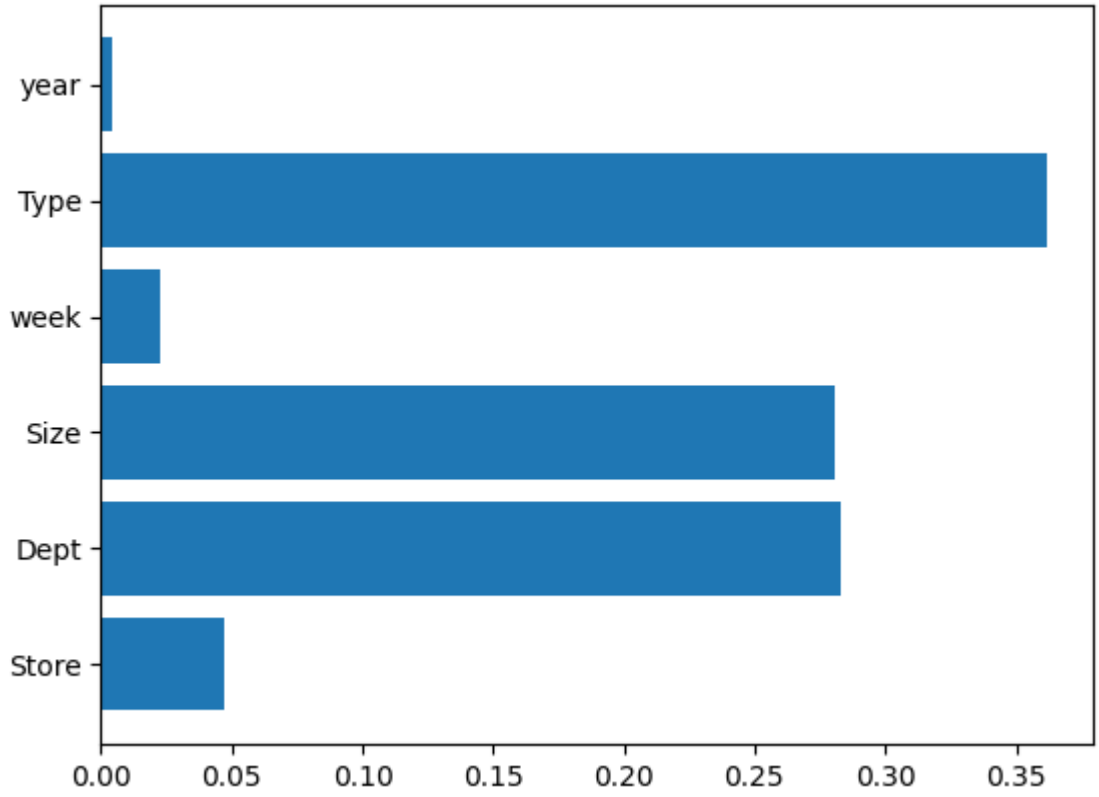

```
performance on evaluation
r2= 0.9882573877637099
MAE= 1072.521102495435
WMAE= 1155.1322894908672
==============================
N= 392
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996040985199817
MAE=  287.6703277377553
WMAE=  282.69053101707476
```



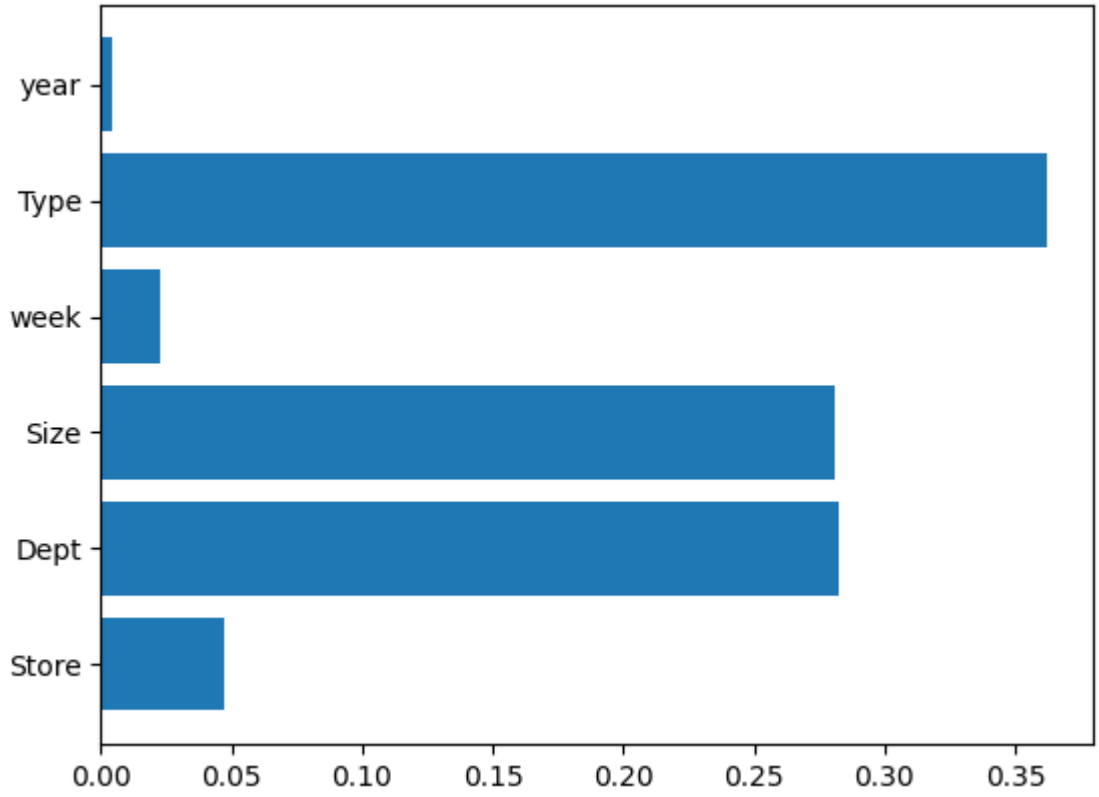

```
performance on evaluation
r2= 0.9882578494500394
MAE= 1072.5391482840855
WMAE= 1155.1245633523622
==============================
N= 393
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996059604129399
MAE=  286.999970758385
WMAE=  282.0453364789112
```



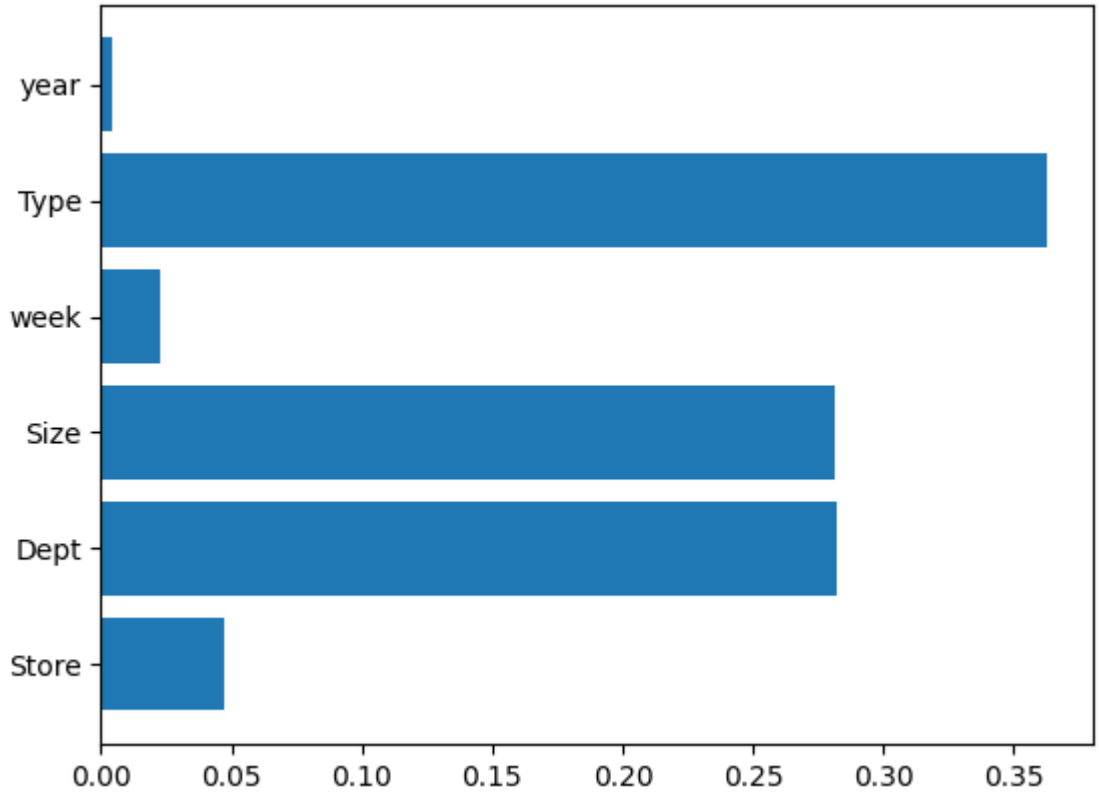

```
performance on evaluation
r2= 0.9882589914528744
MAE= 1072.514131443483
WMAE= 1155.081883924578
==============================
N= 394
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996087262711939
MAE=  286.15565038719745
WMAE=  281.21008386152164
```



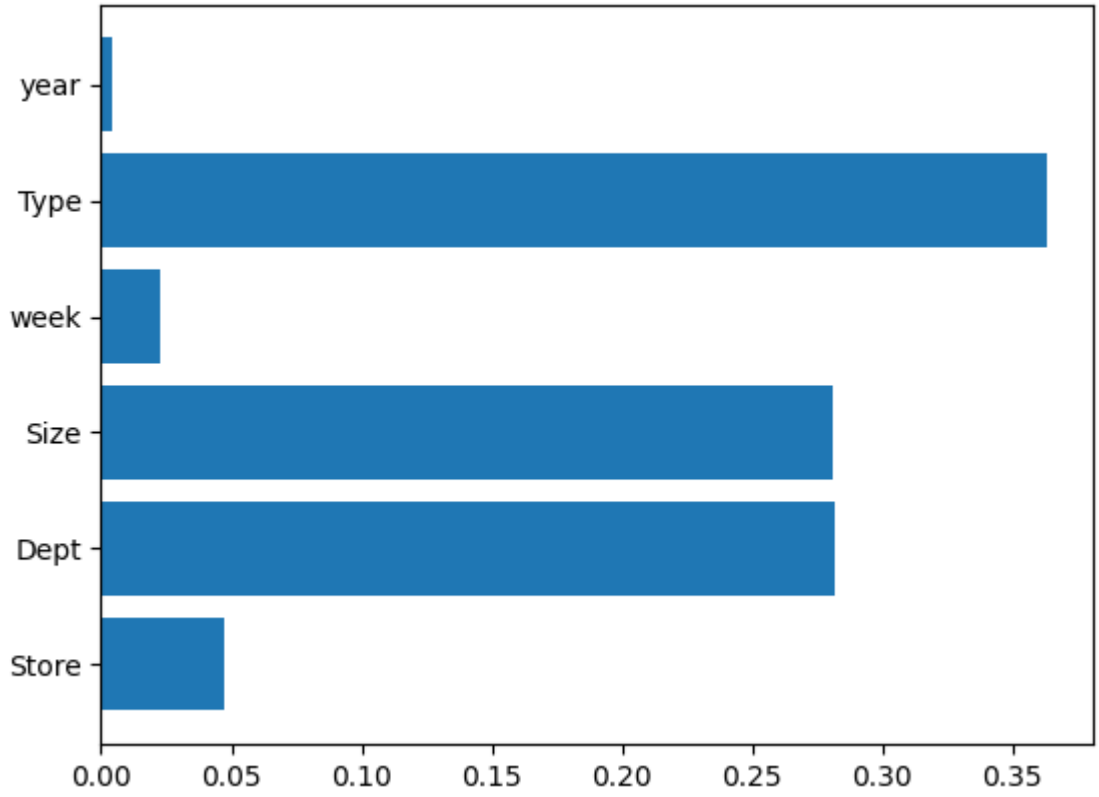

```
performance on evaluation
r2= 0.9882590368306013
MAE= 1072.4931636275542
WMAE= 1155.0488873329787
============================
N= 395
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996103644266034
MAE=  285.59680745599644
WMAE=  280.64502225017316
```



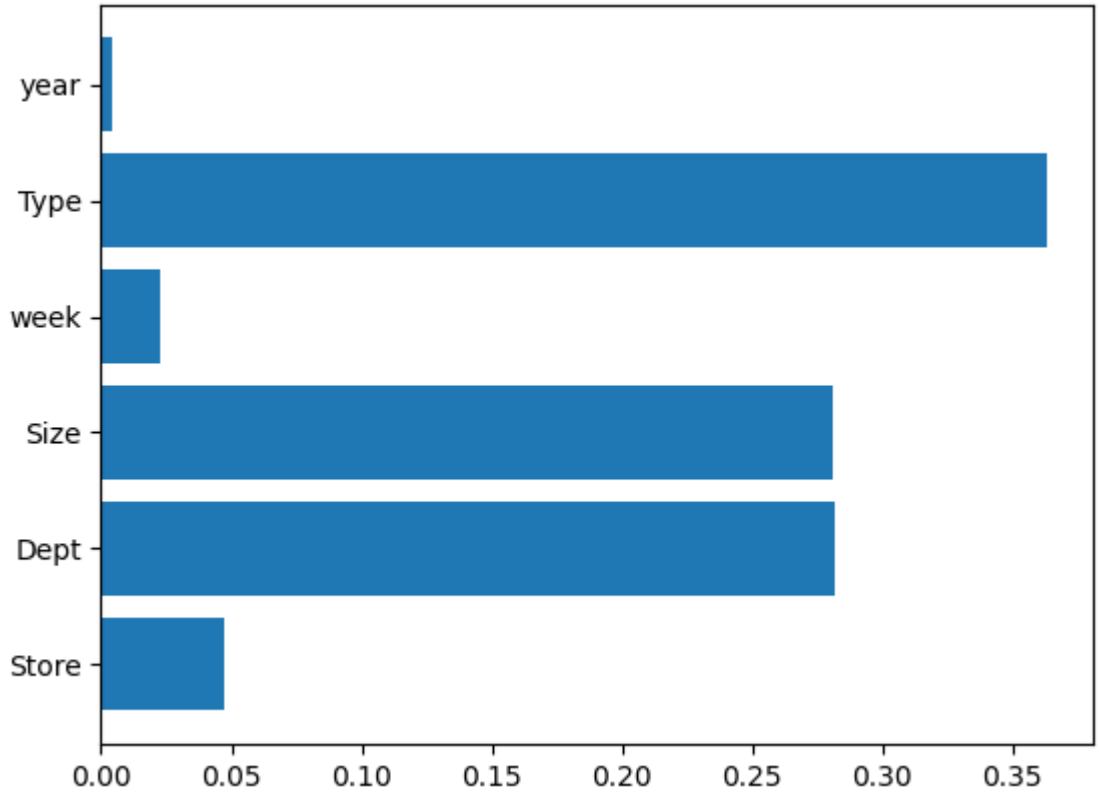

```
performance on evaluation
r2= 0.9882593138846024
MAE= 1072.5072942816687
WMAE= 1155.0741534378112
============================
N= 396
#(train)= 337256 #(val) 84314
performance on training
r2=  0.999612524685167
MAE=  284.89821244438036
WMAE=  279.97300338367006
```



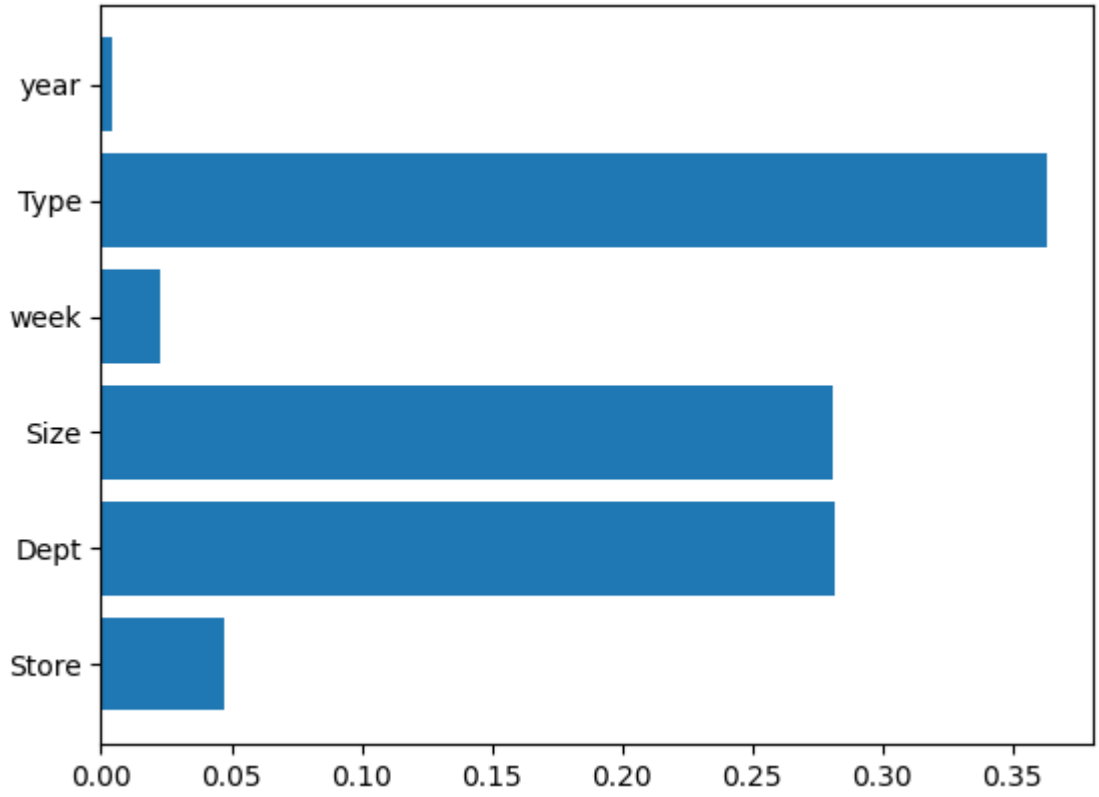

```
performance on evaluation
r2= 0.9882591230217341
MAE= 1072.5436404537918
WMAE= 1155.1000429074354
============================
N= 397
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996139088883421
MAE=  284.3924971436461
WMAE=  279.48926581140995
```



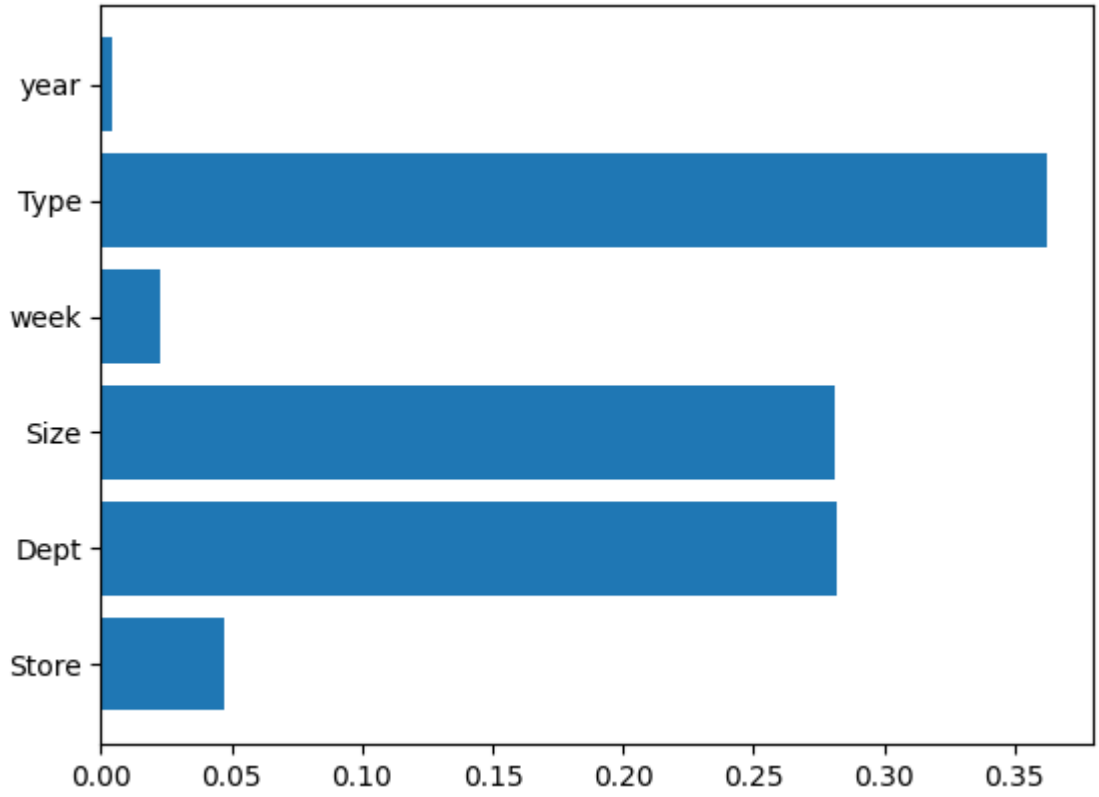

```
performance on evaluation
r2= 0.9882588628026646
MAE= 1072.5844234804697
WMAE= 1155.0998829655668
============================
N= 398
#(train)= 337256 #(val) 84314
performance on training
r2=  0.99961669633298
MAE=  283.5926302073758
WMAE=  278.7115917226024
```



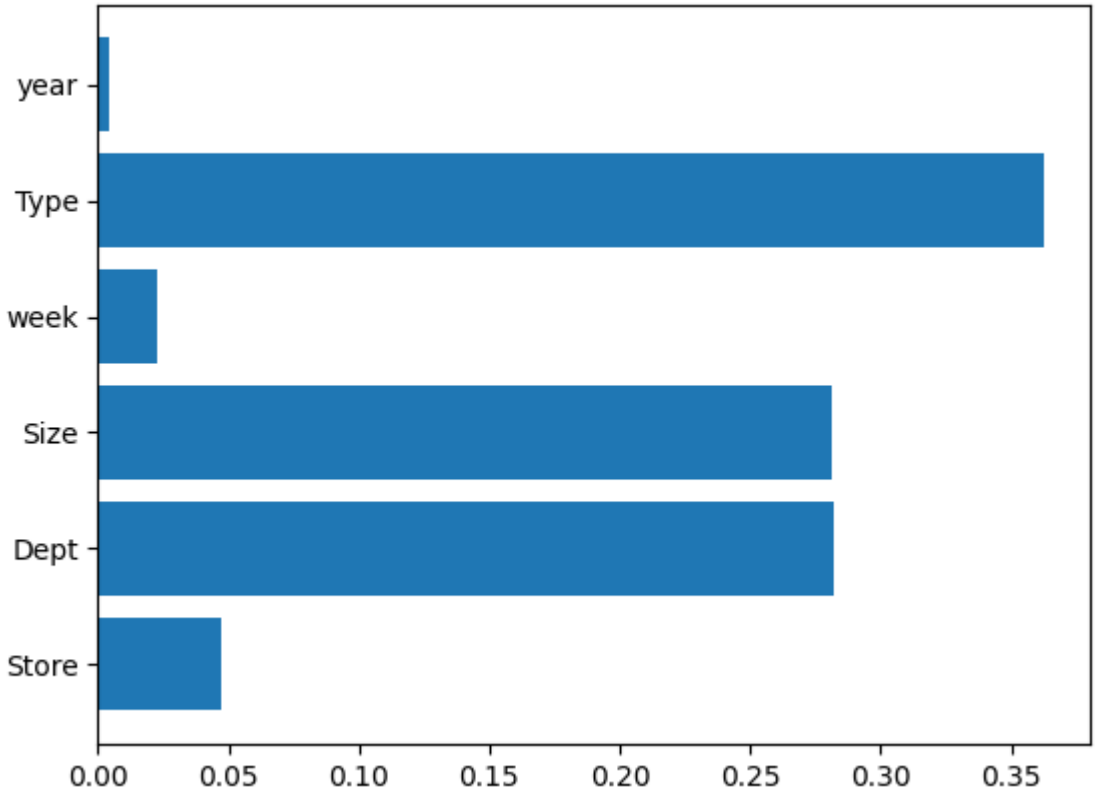

```
performance on evaluation
r2= 0.9882590319000975
MAE= 1072.5934002742774
WMAE= 1155.0805130811186
==============================
N= 399
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996183631724778
MAE=  282.9895608292196
WMAE=  278.1389055831136
```




```
performance on evaluation
r2= 0.9882582190528776
MAE= 1072.6445901154796
WMAE= 1155.110377473188
============================
N= 400
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996201172917372
MAE=  282.34589148381025
WMAE=  277.4900938022944
```



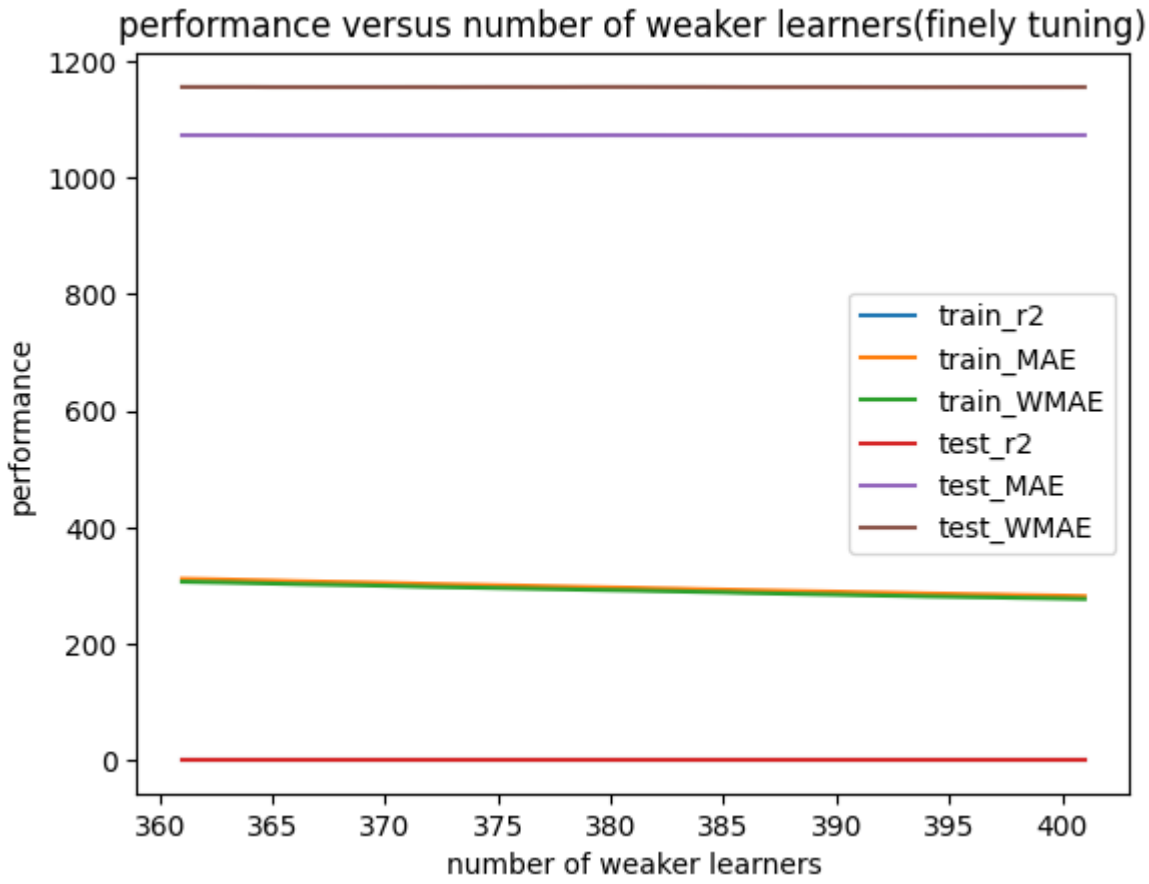

```
performance on evaluation
r2= 0.9882587093483485
MAE= 1072.6833860678407
WMAE= 1155.128009800659
============================
N= 401
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996226159666198
MAE=  281.549842100295
WMAE=  276.77672035179677
```




```
performance on evaluation
r2= 0.9882582303936814
MAE= 1072.72605945209
WMAE= 1155.1752618236108
```

In [128…

```
res_plot(serie_n_fine,y_fine,
        title= "performance versus number of weaker learners(finely tuning)",
        xlabel = 'number of weaker learners')
```



In [116…
```
N_best = serie_n_fine[5].index(min(serie_n_fine[5])) + a
N_best
```

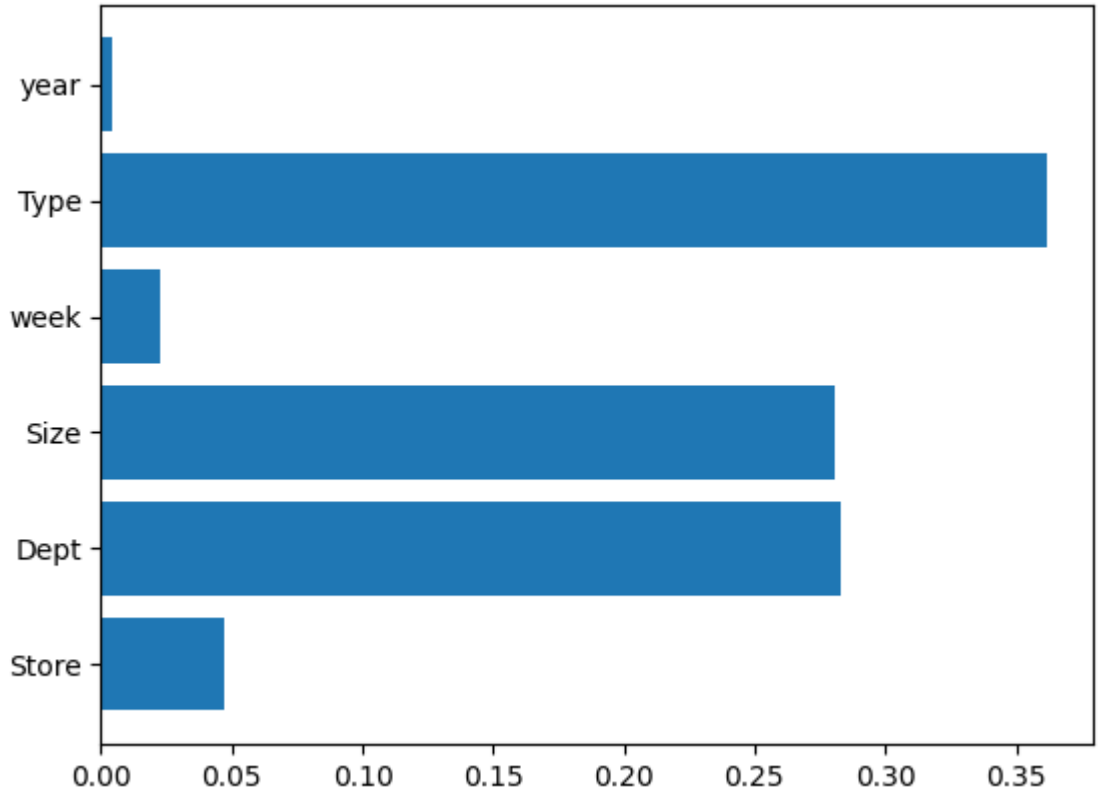
Out[116]:
```
394
```

In [117…
```
d_best, N_best
```

Out[117]:
```
(14, 394)
```

In [118…
```
tmp = train_boost(df_train, df_val, sumb = False, depth= d_best , N_ = N_best, feature
```

```
#(train)= 337256 #(val) 84314
performance on training
r2=  0.9996087262711939
MAE=  286.15565038719745
WMAE=  281.21008386152164
```

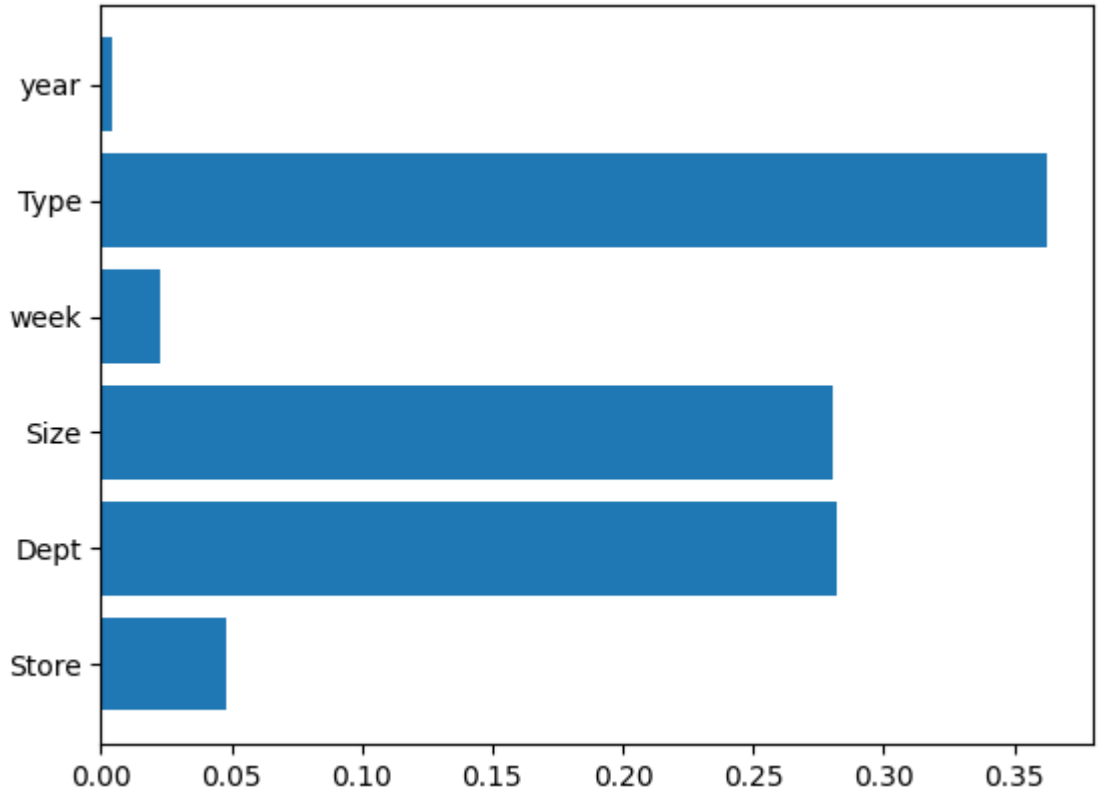



```
performance on evaluation
r2= 0.9882590368306013
MAE= 1072.4931636275542
WMAE= 1155.0488873329787
```

In [119…

```
tmp = train_boost(df, df_test, sumb = True, depth= d_best , N_ = N_best, features = fe
```

```
#(train)= 421570 #(val) 115064
performance on training
r2=  0.9994843687887396
MAE=  320.8731956979353
WMAE=  315.8635468640063
```




```
Index(['Store', 'Dept', 'Size', 'week', 'Type', 'Date', 'year'], dtype='object')
```

In [ ]:

In [ ]:

Test preprocess

In [ ]:
```
df_train_all = df.copy()

df_train_all = df_train_all[ df_train_all['Weekly_Sales'].apply(lambda x: x >= 0) ]

min_sales = df_train_all['Weekly_Sales'].min()
min_sales
```

In [50]:
```
# df_test preprocess
df_test = pd.read_csv('./csv/test.csv')
df_test['Date'] = pd.to_datetime(df_test['Date'])


df_test = pd.merge(df_test, df_feat, on=['Store', 'Date', 'IsHoliday'], how = 'left')
df_test = pd.merge(df_test, df_store, on=['Store'], how = 'left' )

fill_na_cols = [ 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5' ]

for c in fill_na_cols:
    df_test[c] = df_test[c].fillna(0)

df_test['MarkDown'] = df_test[fill_na_cols].sum(axis=1)
# df['MarkDown'] = df['MarkDown3'].copy()
# df_test = df_test.drop(columns = fill_na_cols)
```

```
df_test['year'] = df_test['Date'].dt.year
df_test['week'] = df_test['Date'].dt.week
df_test['month'] = df_test['Date'].dt.month

label=LabelEncoder()

cols = ['Type', 'IsHoliday']

for c in cols:
    df_test[c] = label.fit_transform(df_test[c])
    df_test[c] = df_test[c].astype('category')

# df_test = df_test[ ['Date', 'Store','Dept','IsHoliday','Size','week','Type'] ] # ,'y
```

```
/tmp/ipykernel_2157/1757414661.py:19: FutureWarning: Series.dt.weekofyear and Series.
dt.week have been deprecated. Please use Series.dt.isocalendar().week instead.
  df_test['week'] = df_test['Date'].dt.week
```