MOHAMMED VI POLYTECHNIC UNIVERSITY

EMINES School of Industrial Management

UNIVERSITÉ MOHAMMED VI POLYTECHNIQUE

ANALYTICS EDGE

WALMART SALES-FORECASTING

TEAM:

- Soukaina CHAKOUR
- Hajiba MOUTIA
- Abdelmoula ELBOUHALI
- Walid CHOUKRI
- Dao Pascal SOUMAHORO
- Taha LAKHDARI

SUPERVISOR:

Pr. Maryam GUESSOUS

INTRODUCTION

The world of large-scale distribution is very sensitive to a good understanding of consumer purchasing habits, and to the identification of seasonal trends, therefore in order to meet these needs, this sector adopts in its supply chain strategy an essential part based on the sales Forecasting, indeed this forecast makes it possible to manipulate the stock management and the purchase orders, especially as the biggest nightmare for this commerce is to have empty shelf product, or conversely to have perishable overstock.

This is exactly the case of Walmart being one of the biggest multinational retail corporation that operates a chain hypermarkets and stores that are located in different regions and with variant departments. Therefore, an accurate Sales Forecasting is crucial for a better management of inventory planning of each store.

In our project, we will use historical datasets of Walmart that contain :

- Sales data of 45 stores based on date, department, type and size
- Holidays and Markdowns data
- Macro-indicators like CPI (Consumer Price Index), Fuel price, temperature and unemployement
- Based on these different datasets, our main objectif is to project the sales for each department in each store.

SUMMARY

I-DATASET OVERVIEW

Four datasets are provided:

Stores.csv:

This file contains columns indicating the type and size of each store.

train.csv:

This is the training data, which starts from 2010-02-05 to 2012-11-01. Within this file, we have the following columns:

- Store : the store number from 1 to 45
- Dept : the department number
- Date : the week in format dd/mm/yyyy
- Weekly_Sales : sales for the given department in the given store
- IsHoliday : indicates whether the week is a special holiday or not

test.csv:

This is the testing data. It is same as train without the weekly sales column; therefore, we cannot use this file, as we cannot compare our predictions to the actuals.

features.csv:

This file contains additional data related to macro-indicators with common columns with train.csv and features.csv, the columns are:

- Store : the store number
- Date : the week
- Temperature : average temperature in the region
- Fuel_Price : cost of fuel in the region
- MarkDown1-5: anonymized data related to promotional markdowns that Walmart is running.
- CPI : the consumer price index
- Unemployment : the unemployment rate
- IsHoliday : whether the week is a special holiday week

II-DATA PREPARATION:

First, we will import all the required libraries:

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as seabornInstance
import statsmodels.api as sm
from datetime import datetime
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error as MSE
import seaborn as sns
import optuna
from catboost import Pool, CatBoostRegressor
from sklearn.tree import  DecisionTreeRegresso
from sklearn.ensemble import RandomForestRegressorr
import xgboost
import ray
from ray import tune
from ray.tune.suggest import ConcurrencyLimiter
from ray.tune.schedulers import AsyncHyperBandScheduler
from ray.tune.suggest.hyperopt import HyperOptSearch
from ray.tune.suggest.optuna import OptunaSearch
from ray.tune.logger import DEFAULT_LOGGERS
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict, \
    GridSearchCV, RandomizedSearchCV, KFold
```

Next, we load and read the datasets using the « read.csv » function:

```
features=pd.read_csv('features.csv')
train=pd.read_csv('train.csv')
stores=pd.read_csv('stores.csv')
```

And after «head()» function we get :

train:

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

stores:

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

features:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

After, it was necessary to merge the datasets on key (common) columns with ‘inner’ method, in order to guarantee that the merge will be based on the intersection of rows.

```
stores_features=stores.merge(features, how='inner', on = "Store")
train=train.merge(stores_features, how='inner', on = ["Store","Date","IsHoliday"])
```

Therefore, we have the final dataset ‘train’ below:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment
0	1	1	2010-02-05	24924.50	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106
1	1	2	2010-02-05	50605.27	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106
2	1	3	2010-02-05	13740.12	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106
3	1	4	2010-02-05	39954.04	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106
4	1	5	2010-02-05	32229.38	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106

Now we extract ‘Year’, ’Month’ and ‘Week’ from the ‘Date’ feature

```
train = train.copy()
train['Date'] = pd.to_datetime(train['Date'])
train['Year'] = pd.to_datetime(train['Date']).dt.year
train['Month'] = pd.to_datetime(train['Date']).dt.month
train['Day'] = pd.to_datetime(train['Date']).dt.day
train['Week'] = pd.to_datetime(train['Date']).dt.week
train['Weekofyear'] = pd.to_datetime(train['Date']).dt.weekofyear
```

III-EXPLORATORY DATA ANALYSIS:

Now we should visualize the columns types:

```
Store                    int64
Dept                     int64
Date            datetime64[ns]
Weekly_Sales           float64
IsHoliday                 bool
Type                    object
Size                     int64
Temperature            float64
Fuel_Price             float64
MarkDown1              float64
MarkDown2              float64
MarkDown3              float64
MarkDown4              float64
MarkDown5              float64
CPI                    float64
Unemployment           float64
Year                     int64
Month                    int64
Week                     int64
Day                      int64
dtype: object
```

All variables are numeric exept 'IsHoliday' that is booelan and 'Type' (of the store) that is object

Now we describe the dataset:

	Store	Dept	Weekly_Sales	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2
count	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	150681.000000	111248.000000
mean	22.200546	44.260317	15981.258123	136727.915739	60.090059	3.361027	7246.420196	3334.628621
std	12.785297	30.492054	22711.183519	60980.583328	18.447931	0.458515	8291.221345	9475.357325
min	1.000000	1.000000	-4988.940000	34875.000000	-2.060000	2.472000	0.270000	-265.760000
25%	11.000000	18.000000	2079.650000	93638.000000	46.680000	2.933000	2240.270000	41.600000
50%	22.000000	37.000000	7612.030000	140167.000000	62.090000	3.452000	5347.450000	192.000000
75%	33.000000	74.000000	20205.852500	202505.000000	74.280000	3.738000	9210.900000	1926.940000
max	45.000000	99.000000	693099.360000	219622.000000	100.140000	4.468000	88646.760000	104519.540000

MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Year	Month	Week	Day
137091.000000	134967.000000	151432.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000
1439.421384	3383.168256	4628.975079	171.201947	7.960289	2010.968591	6.449510	25.826762	15.673131
9623.078290	6292.384031	5962.887455	39.159276	1.863296	0.796876	3.243217	14.151887	8.753549
-29.100000	0.220000	135.160000	126.064000	3.879000	2010.000000	1.000000	1.000000	1.000000
5.080000	504.220000	1878.440000	132.022667	6.891000	2010.000000	4.000000	14.000000	8.000000
24.600000	1481.310000	3359.450000	182.318780	7.866000	2011.000000	6.000000	26.000000	16.000000
103.990000	3595.040000	5563.800000	212.416993	8.572000	2012.000000	9.000000	38.000000	23.000000
141630.610000	67474.850000	108519.280000	227.232807	14.313000	2012.000000	12.000000	52.000000	31.000000

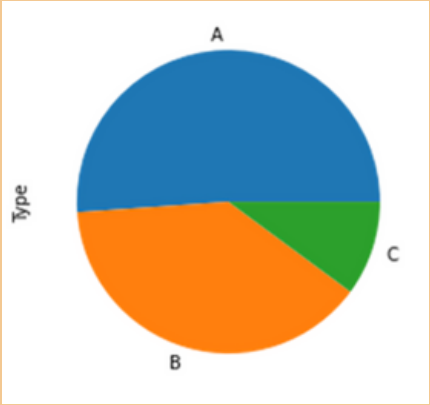
Appearance of negative values on 'Weekly_Sales' which is abnormal, we can say that it is probably about typing error.

So to adapt this column we replace 'Weekly_Sales' by its absolute values:

```
train['Weekly_Sales']=(train.Weekly_Sales).abs()
```

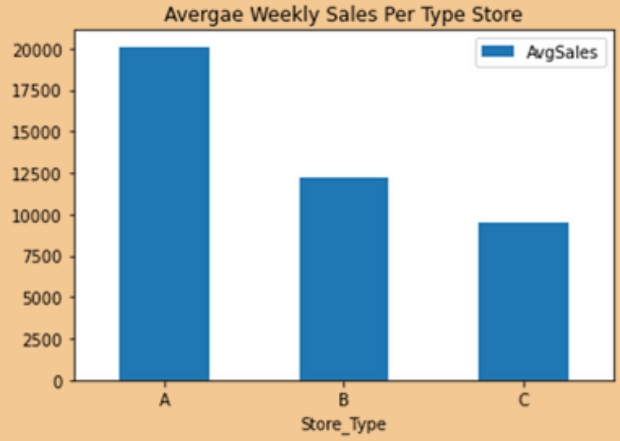
- **Average Weekly_Sales per store Type:**

Firstly, we visualize the frequency of appearance of each type of stores in the dataset:



Type A Stores are the most dominant

Now we visualize the average Weekly sales for each store type:

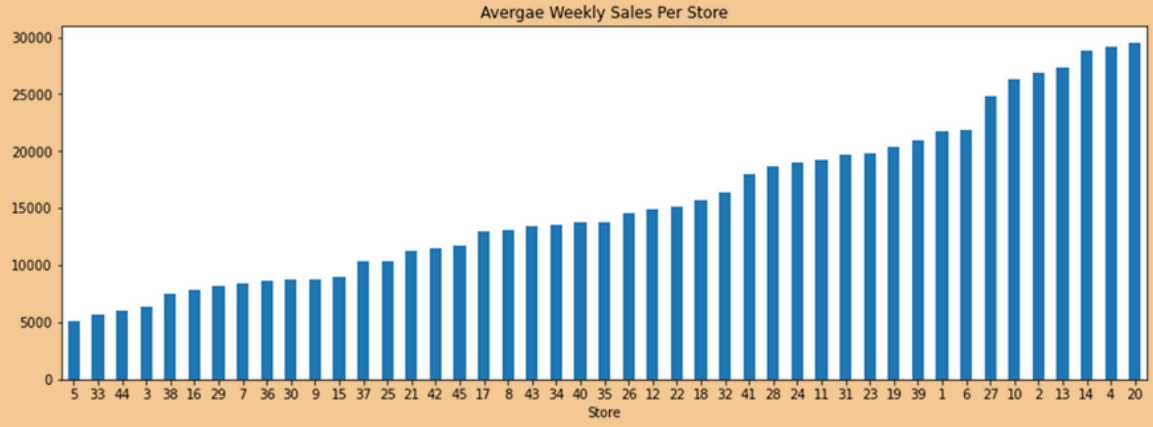


We observe that the average Sales per type 'A' stores are higher than ones of the other types.

- **Average Weekly Sales per Store:**

Here we visualize the average weekly sales for each one of the 45 stores

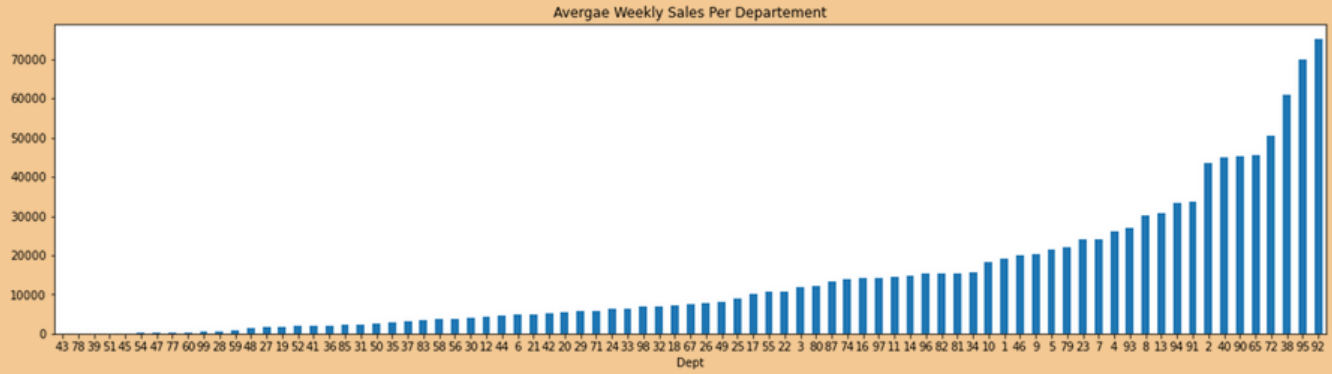
```
df = train.groupby('Store')['Weekly_Sales'].mean().sort_values()
plt.figure(figsize=(15,5))
df.plot.bar(x=df.index, y=df.values, title="Avergae Sales Per Store",rot=0)
```



We observe that the sales vary "hugely" among the 45 stores, so we ca say that the Weekly Sales are highly dependent on variable' Store'.

- **Average Weekly Sales per Department:**

The sales vary "hugely" among the different departments so the variable ‘Dept’ is also a relevant feature for the prediction.

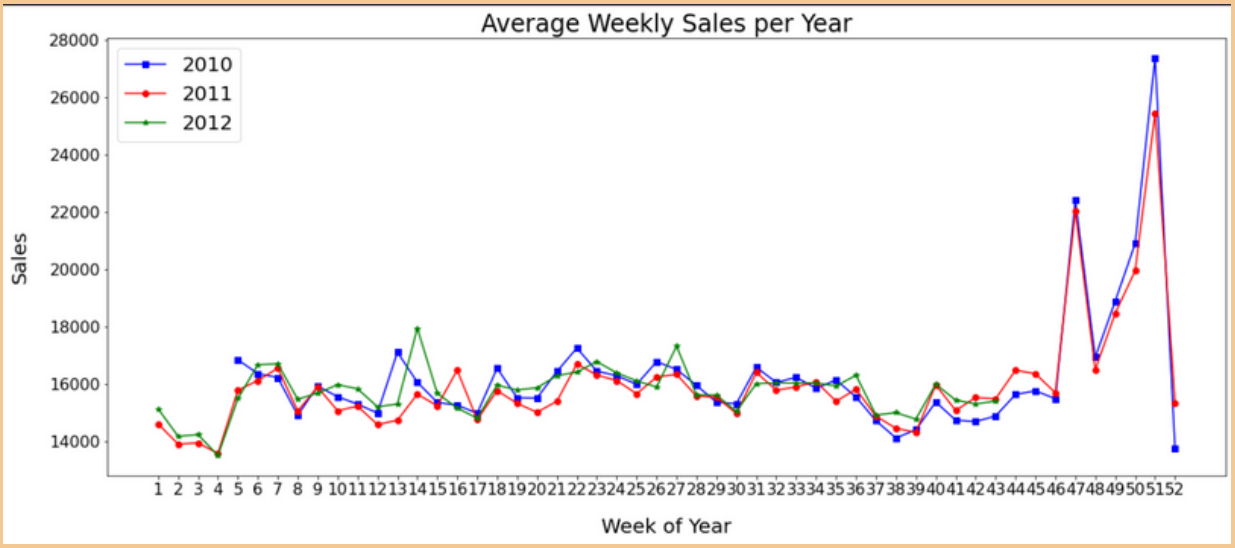


The departements 38, 95 and 92 have the heighest levels of Sales while 47, 43, 78, 39, 51, 45 and 45 departements do not have any Sales in all Stores.

- **Average Weekly Sales per Year:**

Here we see the evolution of Average Weekly_Sales throughout weeks of three given years.

```
weekly_sales_2010 = train[train.Year==2010].groupby('Week')['Weekly_Sales'].mean()
weekly_sales_2011 = train[train.Year==2011].groupby('Week')['Weekly_Sales'].mean()
weekly_sales_2012 = train[train.Year==2012].groupby('Week')['Weekly_Sales'].mean()

plt.figure(figsize=(20,8))
plt.plot(weekly_sales_2010.index, weekly_sales_2010.values, 's-b')
plt.plot(weekly_sales_2011.index, weekly_sales_2011.values, 'o-r')
plt.plot(weekly_sales_2012.index, weekly_sales_2012.values, '*-g')

plt.xticks(np.arange(1, 53, step=1), fontsize=16)
plt.yticks( fontsize=16)
plt.xlabel('Week of Year', fontsize=20, labelpad=20)
plt.ylabel('Sales', fontsize=20, labelpad=20)
plt.title("Average Weekly Sales per Year", fontsize=24)
plt.legend(['2010', '2011', '2012'], fontsize=20)
```



Walmart Weekly Sales conserve almost the same trend over the 3 years given.

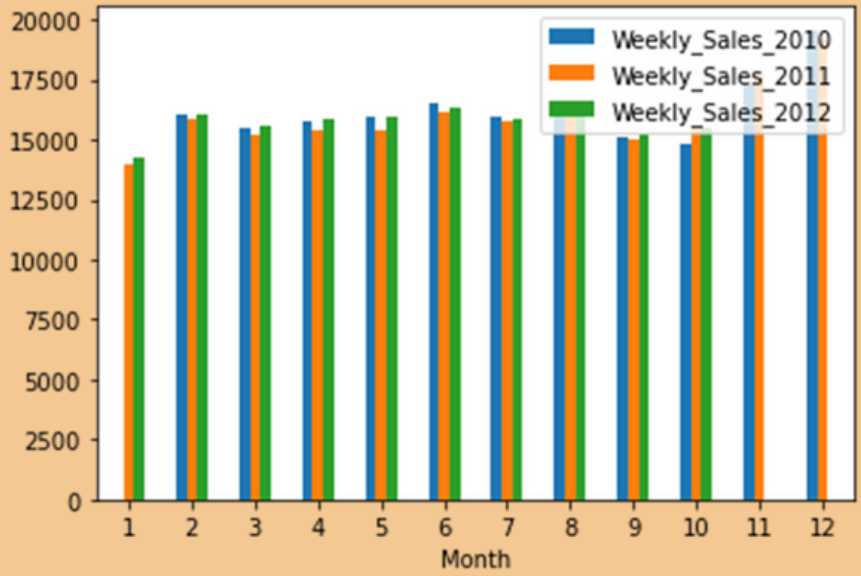
The highest amounts of Sales are recorded on the weeks 47 and 51 which corresponds exactly to one week before the Christmas and the week of Thanksgiving holiday

- Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13
- Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

- **Average Monthly Sales per Year:**

The bar chart below shows the average of sales per month.

```
Month_sales_2010 = pd.DataFrame(train[train.Year==2010].groupby('Month')['Weekly_Sales'].mean())
Month_sales_2011 = pd.DataFrame(train[train.Year==2011].groupby('Month')['Weekly_Sales'].mean())
Month_sales_2012 = pd.DataFrame(train[train.Year==2012].groupby('Month')['Weekly_Sales'].mean())
m2010_2011=Month_sales_2010.merge(Month_sales_2011, how='outer',on="Month",sort=True, suffixes=('_2010', '_
Month_sales=m2010_2011.merge(Month_sales_2012, how='outer',on="Month",sort=True)
Month_sales.rename(columns={"Weekly_Sales": "Weekly_Sales_2012"}, inplace=True)

plt.figure(figsize=(60,80))
Month_sales.plot.bar(rot=0)
```



The Average Monthly Sales remains almost in the same level of 15000 for the 3 years given, excepet for the November and December that record a high amount of Sales while Junuary represents the lowest Sales.

- **Missing values:**

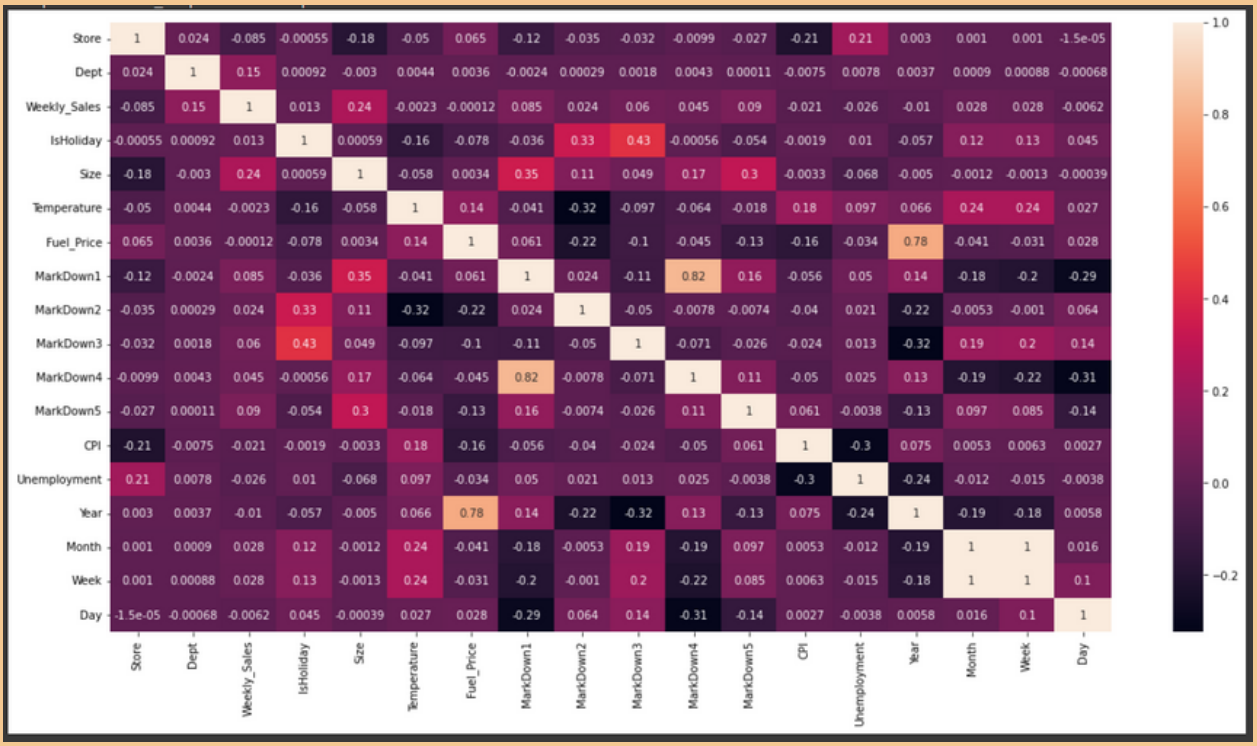
Now we will detect the missing values in our dataset:

```
Store           0.000000
Month           0.000000
Year            0.000000
Unemployment    0.000000
CPI             0.000000
Week            0.000000
Fuel_Price      0.000000
Day             0.000000
Size            0.000000
Type            0.000000
IsHoliday       0.000000
Weekly_Sales    0.000000
Date            0.000000
Dept            0.000000
Temperature     0.000000
MarkDown5       0.640790
MarkDown1       0.642572
MarkDown3       0.674808
MarkDown4       0.679847
MarkDown2       0.736110
```

MarkDowns present a considerable percentage of Nan, it is necessery to study their relationship with the target variable in order to decide if it's relevant to conserve them or not.

- **Correlation Matrix:**

In order to explore the multicoleniarity between features, we use the correlation matrix:

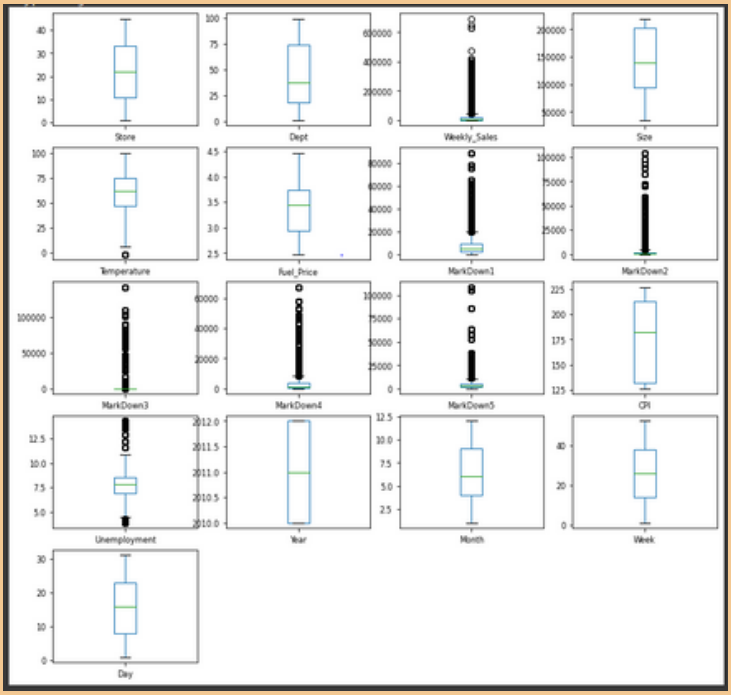


Interpretation:

• Very low correlations between the target 'Weekly Sales' and Temperature, Fuel_Price, CPI, Unemployment → these columns can be dropped.

• There is a moderate correlation between the Weekly Sales and each of the features Store, Dept and Size.

- **Outliers:**



Interpretation:

- For Weekly_Sales we can say that the high values correspond to the pics of sales.
- For the five Markdowns and the employment features, it seems that they contain so much outliers.

III-PREPROCESSING:

- **Encode categorical feature using LabelEncoder:**

```
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
def encode(data):
    #retains only non-null values
    nonulls = np.array(data.dropna())
    #reshapes the data for encoding
    impute_reshape = nonulls.reshape(-1,1)
    #encode data
    impute_ordinal = encoder.fit_transform(impute_reshape)
    #Assign back encoded values to non-null values
    data.loc[data.notnull()] = np.squeeze(impute_ordinal)
    return data

train['IsHoliday']=encode(train['IsHoliday']).astype(int)
train['Type']=encode(train['Type']).astype(int)
```

- **Feature selection:**

Based on the exploratory data analysis and the correlation study, we decide to drop the following features:

```
train = train.drop(['Date', 'Temperature','Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3',
            'MarkDown4', 'MarkDown5', 'CPI', 'Unemployment','Month','Week' ], axis=1)
```

- **Non-scaling justification:**

Despite of having different scales in our dataset, we chose not to scale it, because we intend to use models based on trees.

- **Splitting on training test:**

We chose to split our dataset, such that the trainset contains all the data of the year 2010 while the testset contains the data of the year 2011.

```
train_model = train[train.Year==2010]
test=train[train.Year==2011]

x_train=train_model.drop(columns=['Weekly_Sales'],axis=1)
y_train=train_model['Weekly_Sales']

x_test=test.drop(columns=['Weekly_Sales'],axis=1)
y_test=test['Weekly_Sales']
```

This choice is justified by the fact that the data for the months of December and November of 2012 are not recorded, while these two months are marked by the holidays that affect sales the most, and this is what we ultimately want to model.

IV-MODELING:

- **Decision Tree Regressor :**

The first model we use is Decision tree model:

```
from sklearn.tree import  DecisionTreeRegressor
dtr = DecisionTreeRegressor(random_state=0)

dtr.fit(x_train, y_train)
y_pred =dtr.predict(x_test)
y_pred1 =dtr.predict(x_train)
```

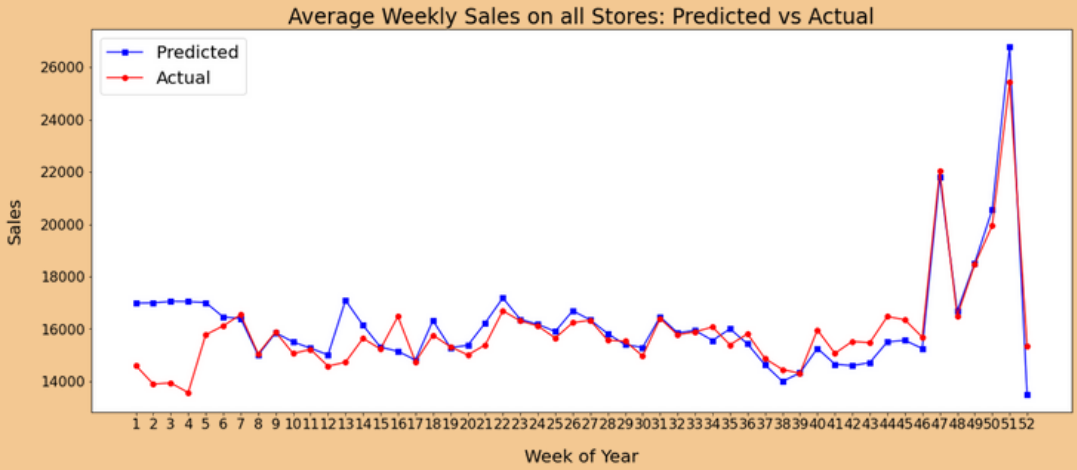
We evaluate the performance of the prediction on the training and testing set, by the RMSE metric and the accuracy.

```
Train score: 1.0
Test score: 0.9559409005225454
Train RMSE: 3.818668965874127e-17
Test RMSE: 4799.658853346945
```

- Actual vs Predicted Sales:

Here we compare the predicted and actual average of the Weekly sales throughout weeks on all departments, we conclude that generaly the predictions are good except for the four first weeks of the year, this can be justified by the fact that the train dataset tahat correspond to the year 2010 does not contain data of January.

```
comparison_dtr = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred, 'Delta': y_pred-y_test})
comparison_dtr=pd.concat([comparison_dtr, x_test],axis=1)

predicted=comparison_dtr.groupby('Weekofyear')['Predicted'].mean()
actual=comparison_dtr.groupby('Weekofyear')['Actual'].mean()
plt.figure(figsize=(20,8))
plt.plot(predicted.index, predicted.values, 's-b')
plt.plot(actual.index, actual.values, 'o-r')

plt.xticks(np.arange(1, 53, step=1), fontsize=16)
plt.yticks( fontsize=16)
plt.xlabel('Week of Year', fontsize=20, labelpad=20)
plt.ylabel('Sales', fontsize=20, labelpad=20)

plt.title("Average Weekly Sales on all Stores: Predicted vs Actual", fontsize=24)
plt.legend(['Predicted', 'Actual'], fontsize=20)
```



- *Hyperparameter tunning:*

We use GridSearch method in order to optimize our decision tree model:

```
from sklearn.model_selection import GridSearchCV
# Define the grid of hyperparameters 'params_dt'
dtr=DecisionTreeRegressor()
params_dtr = {'max_depth': [14,20,30,15,None], 'min_samples_leaf':[1,2,4,5,3,6],'random_state': [0,1,2,3]}
# Instantiate a 10-fold CV grid search object 'grid_dt'
grid_dtr = GridSearchCV(estimator=dtr,param_grid=params_dtr ,cv=10,n_jobs=-1)
# Fit 'grid_dtr' to the training data
grid_dtr.fit(x_train, y_train)
```

```
GridSearchCV(cv=10, estimator=DecisionTreeRegressor(), n_jobs=-1,
             param_grid={'max_depth': [14, 20, 30, 15, None],
                         'min_samples_leaf': [1, 2, 4, 5, 3, 6],
                         'random_state': [0, 1, 2, 3]})
```

```
best_hyperparams = grid_dtr.best_params_
print('Best hyerparameters:\n', best_hyperparams)
```

```
Best hyerparameters:
 {'max_depth': 20, 'min_samples_leaf': 3, 'random_state': 0}
```

We notice after using GridSearch, the metrics was not improved.

```
Train set accuracy of best model: 0.982
Train Error: 3069.9367075735317
Test set accuracy of best model: 0.952
Test Error: 4997.042312458638
```

- **Random Forest:**

The second model used is Random Forest:

```
from sklearn.ensemble import RandomForestRegressor
Model = RandomForestRegressor(criterion='mse',min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100,random_state=None)
Model.fit(x_train, y_train )
y_pred = Model.predict(x_test)
y_pred1 =Model.predict(x_train)
```

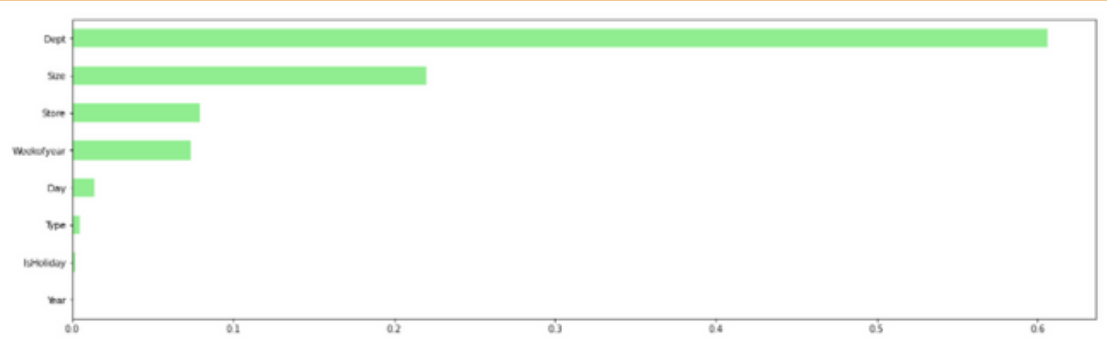
We check the performance of Random Forest prediction using the same metrics.

```
Train score: 0.9946243805128315
Test score: 0.9612755462396578
Train RMSE: 1690.9333571946195
Test RMSE: 4499.717118115292
```

- Features importance:

The graph below shows that the features ‘Dept’, ‘Size’, 'Store' and ‘Weekofyear’and are the most important features for this model.

```
importances = pd.Series(Model.feature_importances_, index = x_train.columns)
sorted_importances = importances.sort_values()
plt.figure(figsize = (20,6))
sorted_importances.plot(kind='barh', color='lightgreen')
plt.show()
```



- *Hyperparameter tunning:*

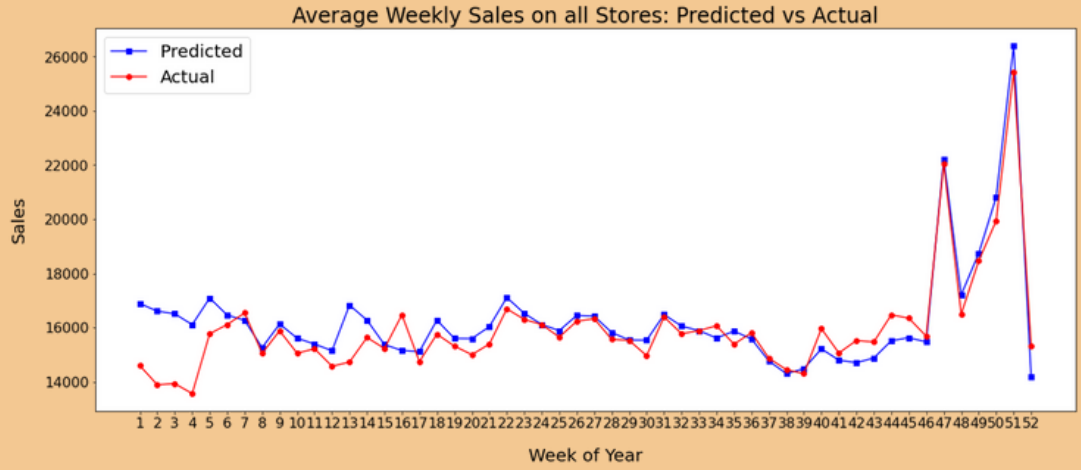
We use RandomizedSearchCV method in order to optimize our model:

```
from sklearn.model_selection import RandomizedSearchCV
from pprint import pprint
n_estimators = [int(x) for x in np.linspace(start = 20, stop = 200, num = 5)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(1, 45, num = 3)]
min_samples_split = [5, 10]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split}

pprint(random_grid)
```

```
The best model from the randomized search has a RMSE of 4753.83
```

We notice after using GridSearch, the metrics was not improved.

- Actual vs Predicted Sales:



Comparing the predicted and actual average of the Weekly sales throughout weeks on all departments, we can say that Random Forest prediction are as good as those of Decision tree .

- **KNeighbors Regressor :**

The third model used is KNeighbors Regressor :

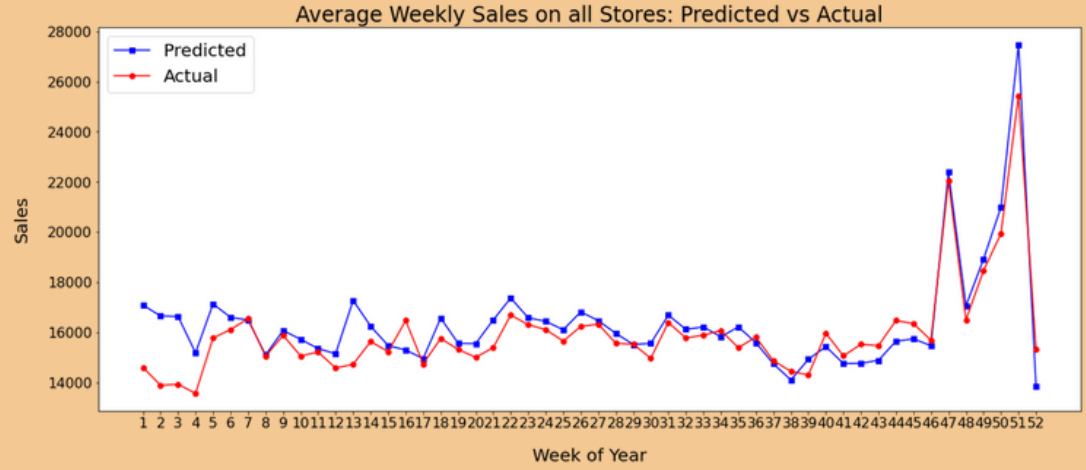
```
from sklearn.neighbors import KNeighborsRegressor
Regressor = KNeighborsRegressor(n_neighbors=1)
Regressor.fit(x_train, y_train)
y_pred = Regressor.predict(x_test)
```

The performance of this model was :

```
Train score: 0.9946760552826441
Test score: 0.9487824400908259
Train RMSE: 1682.7864267900409
Test RMSE: 5174.900500304572
```

This model is less performant than other two first models

- Actual vs Predicted Sales:

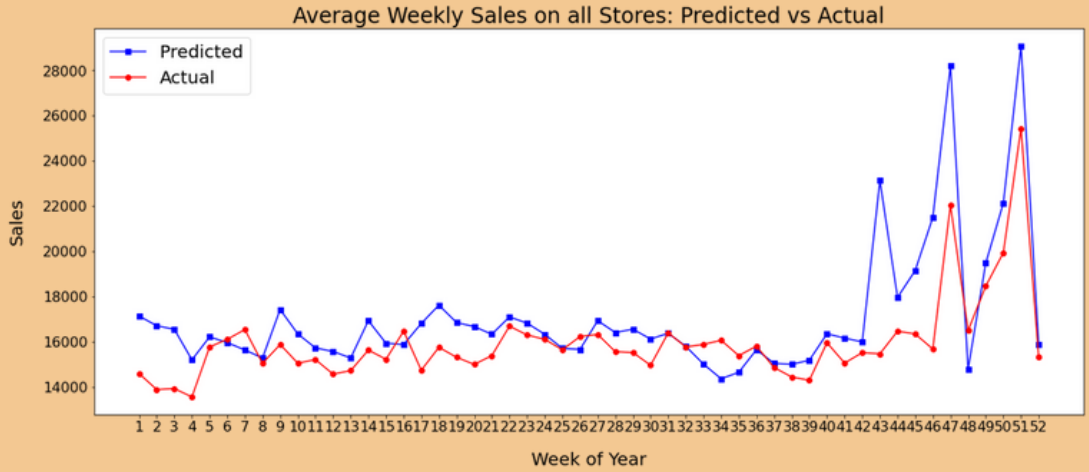


- **Time series :**

The next model used is Time series :

```
from statsmodels.tsa.ar_model import AutoReg
from sklearn.metrics import mean_squared_error
from math import sqrt

model = AutoReg(weekly_sales, lags=[1,4])
trained_model = model.fit()
```

- Actual vs Predicted Sales:



The graph shows that this model is not performant, we can say that this is because time series is based on one feature which is date, this is why it is week in prediction

- **Catboost :**

The next model used is Catboost :

```
model = CatBoostRegressor(n_estimators=5000,learning_rate=0.018000000000000002,
                          depth= 10, l2_leaf_reg= 4.0, min_child_samples=16)
```

```
Train score: 0.972634494610632
Test score: 0.950846054021616
Train RMSE: 3797.2712389916915
Test RMSE: 5069.577373010192
```

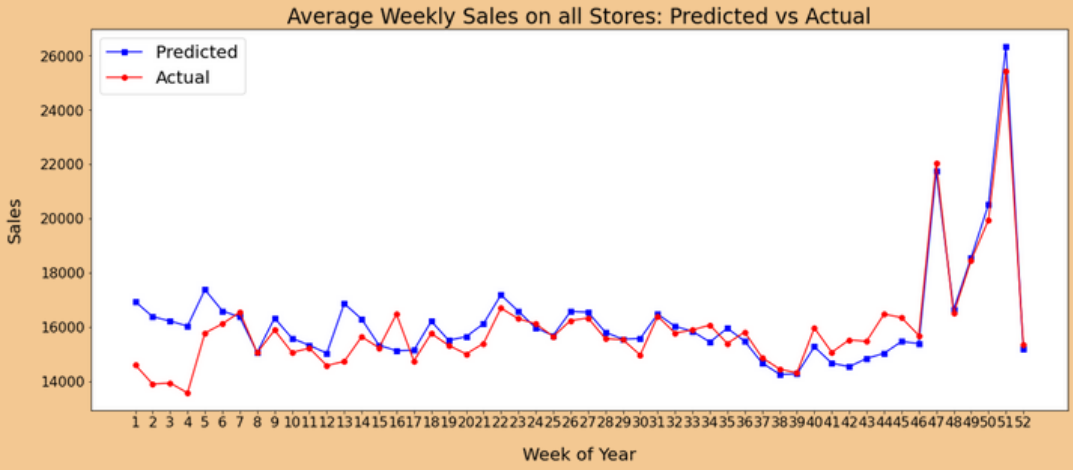
- *Hyperparameter tunning:*

We used Optuna method in order to optimise the model:

```
import optuna
def objective(trial):
    param = {}
    param['learning_rate'] = trial.suggest_discrete_uniform("learning_rate", 0.001, 0.02, 0.001)
    param['depth'] = trial.suggest_int('depth', 9, 16)
    param['l2_leaf_reg'] = trial.suggest_discrete_uniform('l2_leaf_reg', 1.0, 5.5, 0.5)
    param['min_child_samples'] = trial.suggest_categorical('min_child_samples', [4, 6, 8, 16, 20])
    param['grow_policy'] = 'Depthwise'
    param['iterations'] = 10000
    param['use_best_model'] = True
    param['eval_metric'] = 'RMSE'
    param['od_type'] = 'iter'
    param['od_wait'] = 20
    param['random_state'] = RANDOM_SEED
    param['logging_level'] = 'Silent'
```

This method did not give improvement, moreover it takes much time to be ran.

- Actual vs Predicted Sales:



- **XGBOOST**

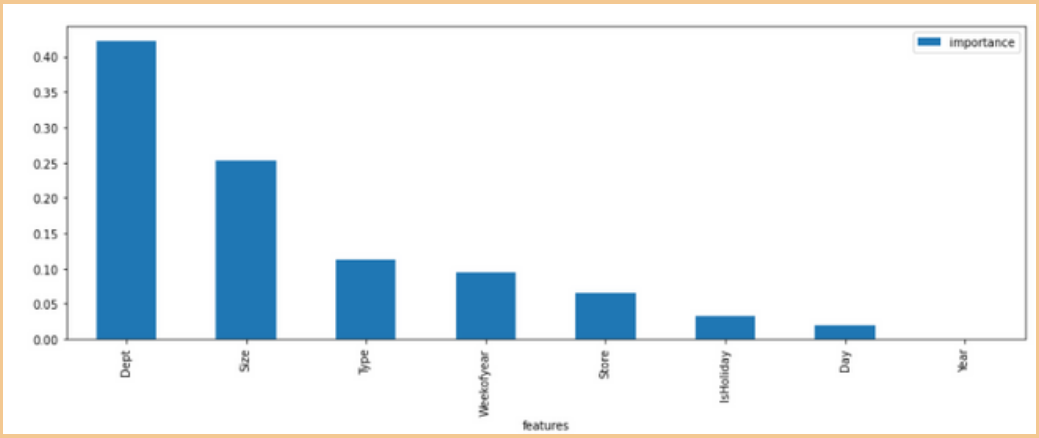
The last model used is the XGboost Regressor

```
model = XGBRegressor(n_estimators=2162, max_depth= 10, subsample=0.5,
                     colsample_bytree= 0.48, colsample_bylevel= 0.48, learning_rate= 0.4)
```

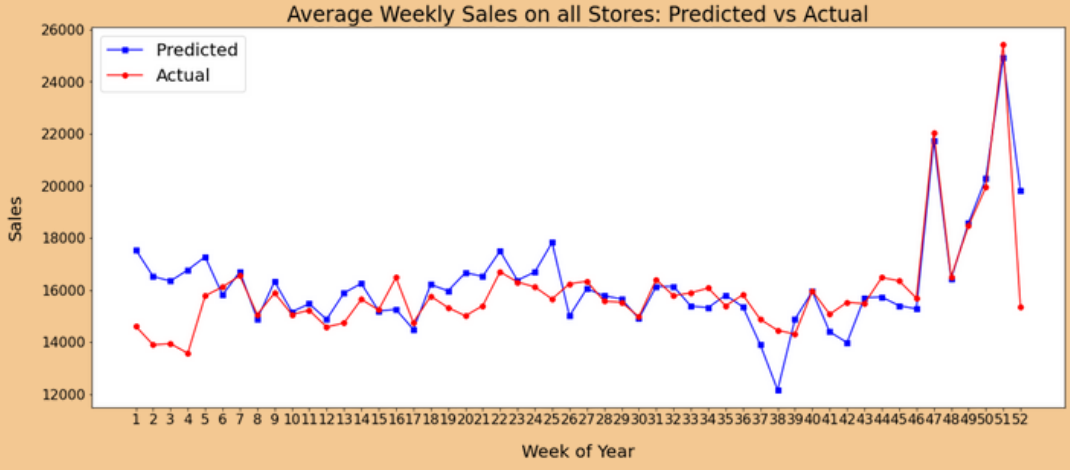
```
Train score: 0.9820443397692741
Train RMSE: 3090.387376622406
Test RMSE: 5329.381307793627
```

- Features importance:

Always we find that the features ‘Dept’, ‘Size’, 'Store' and ‘Weekofyear’and are the most important features.



- Actual vs Predicted Sales:



We can mention here that we used optuna method too, to improve the MSE but it doesn't give good results .

V-BEST MODEL PREDICTION

After using all these models, we compare between them in term of MSE and its approximation to the actual model

comparison_rf2

	Actual	Predicted	Delta	Store	Dept	IsHoliday	Type	Size	Year	Day	Weekofyear
3443	15984.24	27946.5611	11962.3211	1	1	0	0	151315	2011	7	1
3444	43202.29	49041.1227	5838.8327	1	2	0	0	151315	2011	7	1
3445	15808.15	12989.9820	-2818.1680	1	3	0	0	151315	2011	7	1
3446	37947.80	38347.6859	399.8859	1	4	0	0	151315	2011	7	1
3447	22699.69	30480.7357	7781.0457	1	5	0	0	151315	2011	7	1
...	...	...	...	...	...	...	...	...	...	...	...
418660	1426.52	2140.2614	713.7414	45	93	1	1	118221	2011	30	52
418661	2979.03	4141.4772	1162.4472	45	94	1	1	118221	2011	30	52
418662	42084.36	44600.2729	2515.9129	45	95	1	1	118221	2011	30	52
418663	5569.82	5989.1197	419.2997	45	97	1	1	118221	2011	30	52
418664	553.21	120.1896	-433.0204	45	98	1	1	118221	2011	30	52

153453 rows × 11 columns

THE BEST MODEL THEN IS RANDOM FOREST.

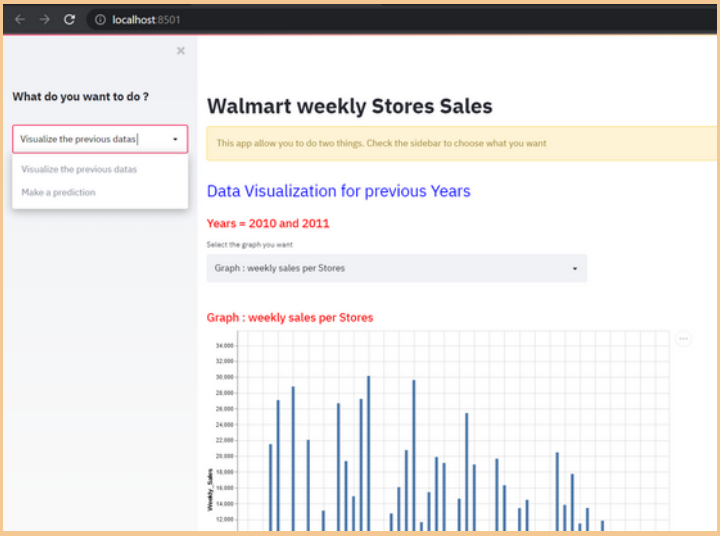
The code below generate graphs that shows predicted Weekly sales Vs actual for each departement in each stores for the Random Forest model.

```
for i in comparison_rf2.Store.unique():
    for j in comparison_rf2[comparison_rf2['Store']==i]['Dept'].unique():
        p=comparison_cat[(comparison_rf2['Store']==i) & (comparison_rf2['Dept']==j)]#['Predicted']
        plt.figure(figsize=(20,8))
        plt.plot(p.Weekofyear.values, p.Predicted.values, 's-b')
        plt.plot(p.Weekofyear.values, p.Actual.values, 'o-r')

        plt.xticks(np.arange(1, 53, step=1), fontsize=16)
        plt.yticks( fontsize=16)
        plt.xlabel('Week of Year', fontsize=20, labelpad=20)
        plt.ylabel('Sales', fontsize=20, labelpad=20)
        plt.title(f"Average Weekly Sales on Store {i} Dept {j}: Predicted vs Actual", fontsize=24)

plt.legend(['Predicted', 'Actual'], fontsize=20)
```

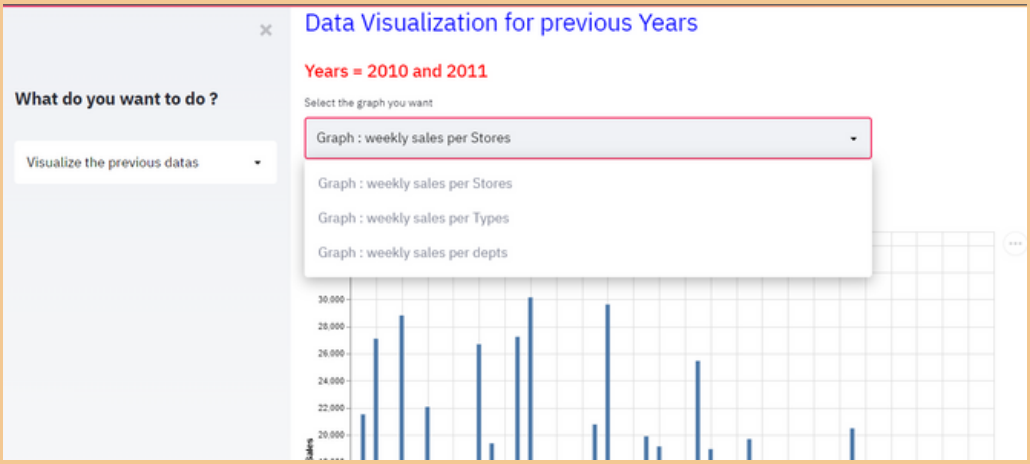
VI-Graphical User Interface

We choose to work on a Graphical User Interface, in order to make predictions easier for the users.

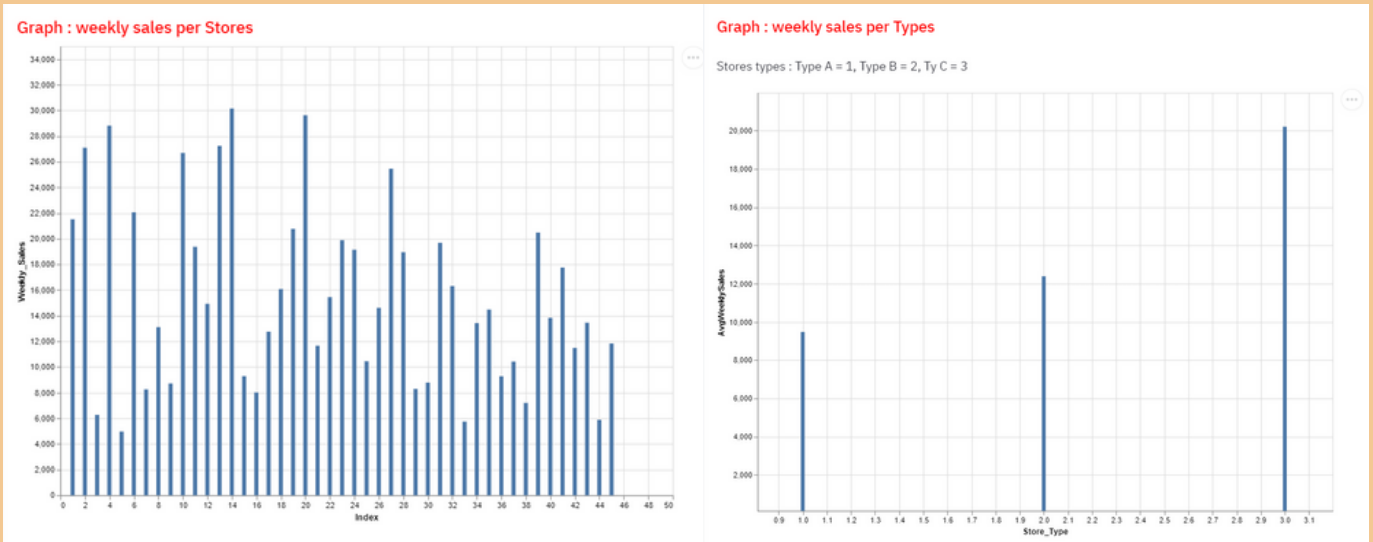
This Interface functions are explained below:

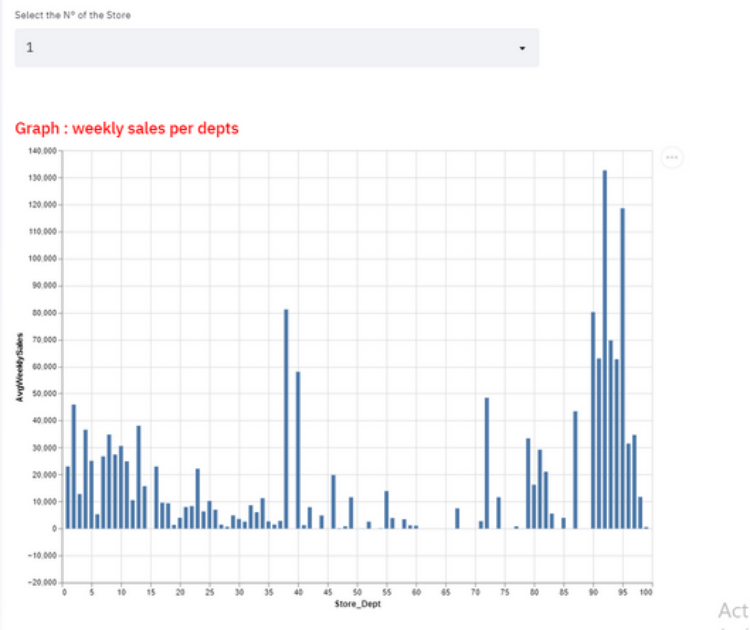


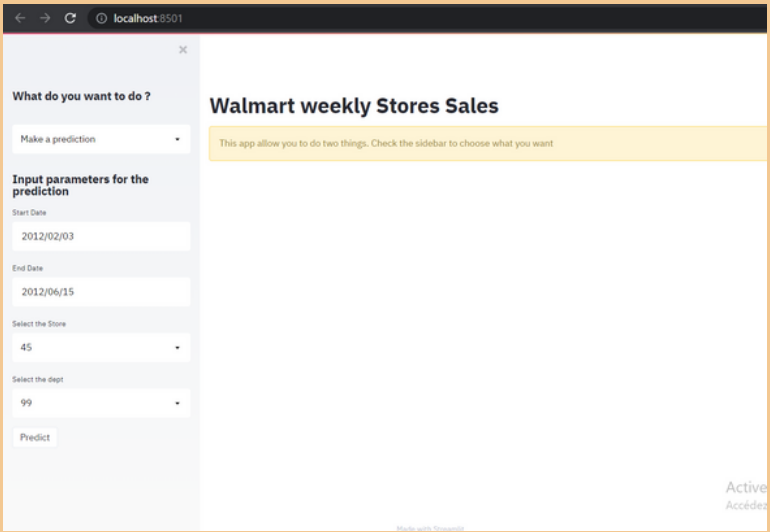
Our interface gives us access mainly to 2 functionalities, namely the visualization of data and the results of previous years for the company and making a prediction for the coming year.



With regard to the visualization of the results of the previous years of the company, the interface gives us the possibility of doing it in several forms through different graphs which are: a graph of the weekly averages of sales by stores, types of blinds and by department. See figure 3, 4 and 5 below:



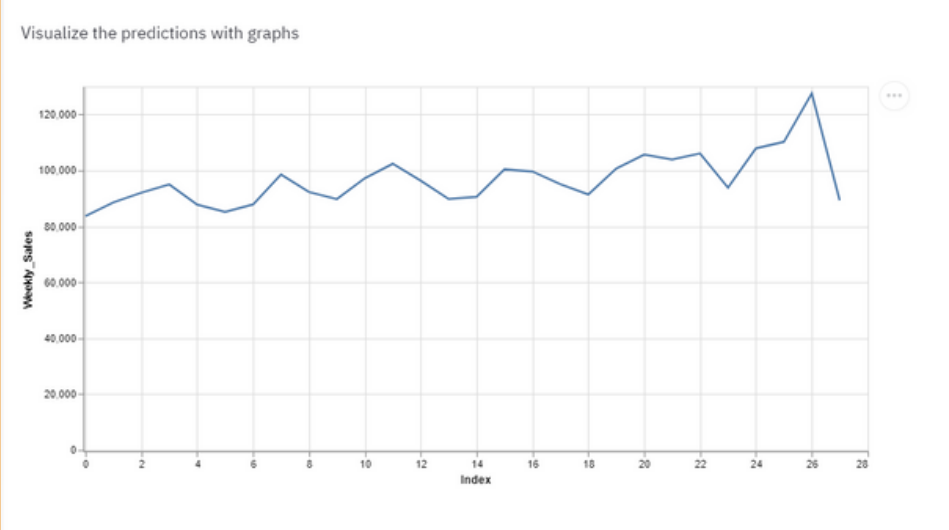


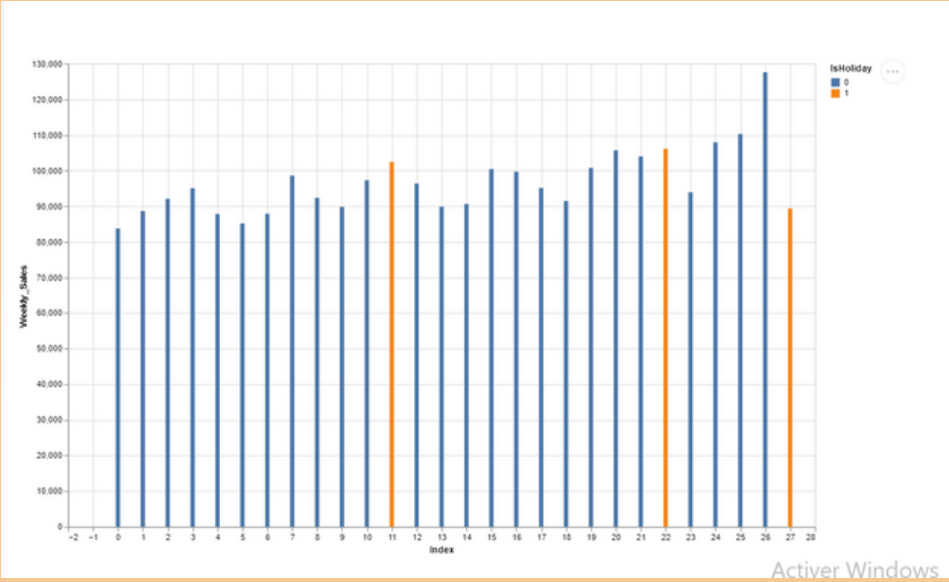


For the prediction part, the user enters the different useful parameters for the execution of our machine learning model. Thanks to these we generate a dataframe which will represent the data for which it will be necessary to predict the weekly sales. Knowing that our model was trained on data from the years 2010 and 2011, it would be preferable to enter dates later than the latter.

	Date	IsHoliday	Weekly_Sales
0	22-06-2012	0	83,746.6794
1	29-06-2012	0	88,623.6520
2	06-07-2012	0	92,069.1084
3	13-07-2012	0	95,027.0954
4	20-07-2012	0	87,803.9885
5	27-07-2012	0	85,162.1401
6	03-08-2012	0	87,869.8917
7	10-08-2012	0	98,587.9409
8	17-08-2012	0	92,322.6161
9	24-08-2012	0	89,786.7181
10	31-08-2012	0	97,254.6737

After running our prediction model, it displays on the interface a table representing the prediction of weekly sales per week over the period entered as a parameter.





The interface also allows us to visualize the results of our model through different graphs according to our preferences

VII-Conclusion

Our best model will be very beneficial for Walmart company, because it can manage and distribute its different stocks according to its different stores, this model prediction also allow the company to favor a department in such a store where sales are higher than in other departement, and this has finally a direct effect on the sales of this company.
The interface also plays a very important role in facilitating the use of this model for users that have no notions in machine learning