FORECASTING SALES USING THE STORE , PROMOTION AND COMPITITOR DATA

**ROSSMANN STORE SALE

BUSINESS PROBLEM

Rossmann operates over 3,000 drug stores in 7 European countries. Currently, Rossmann store managers are tasked with predicting their daily sales for up to six weeks in advance. Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied.

link for the dataset: https://www.kaggle.com/c/rossmann-store-sales/data.

BUSINESS OBJECTIVE

Business objective of this study is to predict daily sales for up to six weeks.

Files

- train.csv - historical data including Sales
- test.csv - historical data excluding Sales
- sample_submission.csv - a sample submission file in the correct format
- store.csv - supplemental information about the stores

Data fields --> Most of the fields are self-explanatory. The following are descriptions for those that aren't

- **Id** an Id that represents a (Store, Date) duple within the test set
- **Store** a unique Id for each store
- **Sales** the turnover for any given day (this is what you are predicting)
- **Customers** the number of customers on a given day
- **Open** an indicator for whether the store was open: 0 = closed, 1 = open
- **StateHoliday** indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
- **SchoolHoliday** indicates if the (Store, Date) was affected by the closure of public schools
- **StoreType** differentiates between 4 different store models: a, b, c, d
- **Assortment** describes an assortment level: a = basic, b = extra, c = extended
- **CompetitionDistance** distance in meters to the nearest competitor store
- **CompetitionOpenSince[Month/Year]** gives the approximate year and month of the time the nearest competitor was opened
- **Promo** indicates whether a store is running a promo on that day
- **Promo2** Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating
- **Promo2Since[Year/Week]** describes the year and calendar week when the store started participating in Promo2
- **PromoInterval** describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. **{**Feb, May, Aug, Nov**}** means each round starts in February, May, August, November of any given year for that store

However, we'll load each dataset and merge the "train.csv" with "store.csv", to build a dataframe that we'll call "df_raw". It's important to know that "test.csv" will be use just in the final sections.

```
### Importing Libraries

import numpy as np
import pandas as pd


!pip install inflection
import inflection
import math
import datetime

from scipy import stats
import pickle
import warnings
warnings.filterwarnings('ignore')
```

```
warnings.warn('DelftStack')
warnings.warn('Do not show this message')

#!pip install HTML
#from boruta import BorutaPy
from html import *
#plotting
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from IPython.display import Image
from tabulate import tabulate
from scipy import stats

#Xgboost
import xgboost             as xgb

#Moudles sklearn
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.preprocessing import RobustScaler, MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import RFE, RFECV, SelectKBest, chi2, SelectPercentile, f_c
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: inflection in /usr/local/lib/python3.7/dist-packages (
```

```
# Helper function for continious variable summary

def continuous_var_summary( x ):
    # freq and missings
    n_total = x.shape[0]
    n_miss = x.isna().sum()
    perc_miss = n_miss * 100 / n_total

    # outliers - iqr
    q1 = x.quantile(0.25)
    q3 = x.quantile(0.75)
    iqr = q3 - q1
    lc_iqr = q1 - 1.5 * iqr
    uc_iqr = q3 + 1.5 * iqr
    return pd.Series( [ x.dtype, x.nunique(), n_total, x.count(), n_miss, perc_miss,
                        x.sum(), x.mean(), x.std(), x.var(),
                        lc_iqr, uc_iqr,
                        x.min(), x.quantile(0.01), x.quantile(0.05), x.quantile(0.10),
                        x.quantile(0.25), x.quantile(0.5), x.quantile(0.75),
                        x.quantile(0.90), x.quantile(0.95), x.quantile(0.99), x.max(),
                        x.skew(),x.kurtosis() ],

                      index = ['dtype', 'cardinality', 'n_tot', 'n', 'nmiss', 'perc_miss',
                               'sum', 'mean', 'std', 'var',
                          'lc_iqr', 'uc_iqr',
                          'min', 'p1', 'p5', 'p10', 'p25', 'p50', 'p75', 'p90', 'p95', 'p99'
```

```
# Helper function for continious variable summary
def cat_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.value_counts()],
                  index=['N', 'NMISS', 'ColumnsNames'])


##Helper function for plotting
def jupyter_settings():
    %matplotlib inline
    %pylab inline

    plt.style.use( 'bmh' )
    plt.rcParams['figure.figsize'] = [22, 10]
    plt.rcParams['font.size'] = 24

    #display(HTML( '<style>.container { width:100% !important; }</style>') )
    pd.options.display.max_columns = None
    pd.options.display.max_rows = None
    pd.set_option( 'display.expand_frame_repr', False )

    sns.set()

def mean_absolute_percentage_error(y, yhat):
    return np.mean(np.abs((y - yhat)/y))

def ml_error(model_name,y,yhat):
    mae = mean_absolute_error(y, yhat)
    mape = mean_absolute_percentage_error(y,yhat)
    rmse = np.sqrt(mean_squared_error(y,yhat))

    return pd.DataFrame({'Model Name': model_name,
                         'MAE': mae,
                         'MAPE': mape,
                         'RMSE': rmse}, index = [0])

def cross_validation( x_training, kfold, model_name, model, verbose=False ):
    mae_list = []
    mape_list = []
    rmse_list = []
    for k in reversed( range( 1, kfold+1 ) ):
        if verbose:
            print( '\nKFold Number: {}'.format( k ) )
        # start and end date for validation
        validation_start_date = x_training['date'].max() - datetime.timedelta( days=k*6*7)
        validation_end_date = x_training['date'].max() - datetime.timedelta( days=(k-1)*6*

        # filtering dataset
        training = x_training[x_training['date'] < validation_start_date]
        validation = x_training[(x_training['date'] >= validation_start_date) & (x_trainin

        # training and validation dataset
        # training
        xtraining = training.drop( ['date', 'sales'], axis=1 )
```

```
        ytraining = training['sales']

        # validation
        xvalidation = validation.drop( ['date', 'sales'], axis=1 )
        yvalidation = validation['sales']

        # model
        m = model.fit( xtraining, ytraining )

        # prediction
        yhat = m.predict( xvalidation )

        # performance
        m_result = ml_error( model_name, np.expm1( yvalidation ), np.expm1( yhat ) )

        # store performance of each kfold iteration
        mae_list.append(  m_result['MAE'] )
        mape_list.append( m_result['MAPE'] )
        rmse_list.append( m_result['RMSE'] )

    return pd.DataFrame( {'Model Name': model_name,
                          'MAE CV': np.round( np.mean( mae_list ), 2 ).astype( str ) + ' +
                          'MAPE CV': np.round( np.mean( mape_list ), 2 ).astype( str ) + '
                          'RMSE CV': np.round( np.mean( rmse_list ), 2 ).astype( str ) + '


#Helper function for outlier treatment

def outlier_capping(x):
    x = x.clip_upper(x.quantile(0.95))
    x = x.clip_lower(x.quantile(0.01))
    return x

jupyter_settings()


     Populating the interactive namespace from numpy and matplotlib


#Mount drive
from google.colab import drive
drive.mount('/content/drive')

     Mounted at /content/drive
```

Reading Data

```
path = "/content/drive/MyDrive/Data Sets/rossmann_store_sales/"
df_train = pd.read_csv(path + "train.csv")

df_store = pd.read_csv(path + "store.csv")
```

```
df_test = pd.read_csv(path + "test.csv")


#Train Data Description
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 9 columns):
 #   Column         Non-Null Count    Dtype 
---  ------         --------------    ----- 
 0   Store          1017209 non-null  int64 
 1   DayOfWeek      1017209 non-null  int64 
 2   Date           1017209 non-null  object
 3   Sales          1017209 non-null  int64 
 4   Customers      1017209 non-null  int64 
 5   Open           1017209 non-null  int64 
 6   Promo          1017209 non-null  int64 
 7   StateHoliday   1017209 non-null  object
 8   SchoolHoliday  1017209 non-null  int64 
dtypes: int64(7), object(2)
memory usage: 69.8+ MB
```

```
df_store.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Store                      1115 non-null   int64  
 1   StoreType                  1115 non-null   object 
 2   Assortment                 1115 non-null   object 
 3   CompetitionDistance        1112 non-null   float64
 4   CompetitionOpenSinceMonth  761 non-null    float64
 5   CompetitionOpenSinceYear   761 non-null    float64
 6   Promo2                     1115 non-null   int64  
 7   Promo2SinceWeek            571 non-null    float64
 8   Promo2SinceYear            571 non-null    float64
 9   PromoInterval              571 non-null    object 
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

```
#merge
df_raw = pd.merge(df_train, df_store, how = 'left', on = 'Store')
df_raw.sample()
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolH
269171	79	3	2014-11-26	5967	577	1	1	0	

DATA PREPARATION

- **Rename Columns in proper naming convention**
- **Check Dimensions of data**
- **Missing values check**
- **Fill missing values**
- **Change Types**
- **Check Data Types**
- **Check Descriptive Statistics**

```
#Shape of data
print("Merged data has ", df_raw.shape[0], " rows")
print("Merged data has ", df_raw.shape[0], " columns")
```

```
Merged data has  1017209  rows
Merged data has  1017209  columns
```

Data Description

```
df_raw.columns
```

```
Index(['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open', 'Promo',
       'StateHoliday', 'SchoolHoliday', 'StoreType', 'Assortment',
       'CompetitionDistance', 'CompetitionOpenSinceMonth',
       'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWeek',
       'Promo2SinceYear', 'PromoInterval'],
      dtype='object')
```

Renaming Columns

Renaming columns in proper naming convention format

```
col_name = df_raw.columns

snakecase = lambda x:inflection.underscore(x)

cols_new = list(map(snakecase, col_name))
cols_new

df_raw.columns = cols_new


cols_new
```

```
['store',
 'day_of_week',
 'date',
 'sales',
 'customers',
```

```
        'open',
        'promo',
        'state_holiday',
        'school_holiday',
        'store_type',
        'assortment',
        'competition_distance',
        'competition_open_since_month',
        'competition_open_since_year',
        'promo2',
        'promo2_since_week',
        'promo2_since_year',
        'promo_interval']
```

Data Dimenssion

Exploring the dimession of data to take a look how large our data is:

```
print("Number of rows " , df_raw.shape[0])
print("Number of columns ", df_raw.shape[1])
```

```
Number of rows  1017209
Number of columns  18
```

Checking Data Types

```
df_raw.dtypes
```

```
store                            int64
day_of_week                      int64
date                            object
sales                            int64
customers                        int64
open                             int64
promo                            int64
state_holiday                   object
school_holiday                   int64
store_type                      object
assortment                      object
competition_distance           float64
competition_open_since_month   float64
competition_open_since_year    float64
promo2                           int64
promo2_since_week              float64
promo2_since_year              float64
promo_interval                  object
dtype: object
```

```
#Converting datatype for date variable
df_raw['date'] = pd.to_datetime(df_raw['date'])
df_raw.dtypes
```

```
store                                    int64
```

```
day_of_week                          int64
date                        datetime64[ns]
sales                                int64
customers                            int64
open                                 int64
promo                                int64
state_holiday                       object
school_holiday                       int64
store_type                          object
assortment                          object
competition_distance               float64
competition_open_since_month       float64
competition_open_since_year        float64
promo2                               int64
promo2_since_week                  float64
promo2_since_year                  float64
promo_interval                      object
dtype: object
```

Checking Missing Values(NAs)

Handling Missing Values is a very important aspect of machine learning there is no specific approach to handle missing data. It largely depends upon business context and importence of a variable

- **Exclude variable with null values :** different literature have different opinion about this while some literature suggest to drop a variable if more than 50% of data is missing while some suggest it to be more than 60 percent.
- **Exclude records with missing value :** if amount of missing values in a column is large than excluding records will reduce the size of our data and for a good model we need maximum amount of data.
- **Replace with the average or median**
- Change according to the business context: This choice is very important because, if we choose to exclude the rows from our data set, depending on the amount of null values, we will eliminate a considerable amount of data so that our model could train.

```
#Percentage of missing values for each variable
print("Percentage of missings in each variable","\n", (df_raw.isna().sum()/df_raw.shape[0]
```

```
Percentage of missings in each variable
 store                          0.000000
day_of_week                    0.000000
date                           0.000000
sales                          0.000000
customers                      0.000000
open                           0.000000
promo                          0.000000
state_holiday                  0.000000
school_holiday                 0.000000
store_type                     0.000000
assortment                     0.000000
competition_distance           0.259730
```

```
competition_open_since_month    31.787764
competition_open_since_year     31.787764
promo2                           0.000000
promo2_since_week               49.943620
promo2_since_year               49.943620
promo_interval                  49.943620
dtype: float64
```

```
df_raw.head()
```

	store	day_of_week	date	sales	customers	open	promo	state_holiday	school_ho
0	1	5	2015-07-31	5263	555	1	1	0	
1	2	5	2015-07-31	6064	625	1	1	0	
2	3	5	2015-07-31	8314	821	1	1	0	
3	4	5	2015-07-31	13995	1498	1	1	0	
4	5	5	2015-07-31	4822	559	1	1	0	

```
df_raw.groupby(["store", "promo2_since_year"])["competition_open_since_month"].nunique({"u
```

```
store  promo2_since_year
2      2010.0               1
434    2014.0               1
Name: competition_open_since_month, dtype: int64
```

Missing Value Treatment

- **competition_distance**

This variable tells us how far the nearest competitor is from our store. I chose to replace it with 100000, as this will be big enough to tell us that there is no competitor close to our store. (max = 75860)

- **competition_open_since_month**

This column tells us how long in months the competing store opened. We will replace it with the month that was filled in the date column.

- **competition_open_since_year**

This column tells us how long in years the competing store has opened. We will replace it with the year that was filled in the date column.

- **promo2_since_week**

describes the calendar week when the store started participating in Promo2. We will extract the week from the date column and replace the values that are null.

- **promo2_since_year**

describes the year when the store started participating in Promo2. We will extract the year from the date column and replace the values that are null.

- **promo_interval**

describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store. We replaced the null values with 0 and checked if the store participated in promo2 or not represented by 0 and 1.

```
#competition_distance
df_raw['competition_distance'] = df_raw['competition_distance'].apply( lambda x: 200000.0

#competition_open_since_month
df_raw['competition_open_since_month'] = df_raw.apply( lambda x: x['date'].month if math.i

#competition_open_since_year
df_raw['competition_open_since_year'] = df_raw.apply( lambda x: x['date'].year if math.isn

#promo2_since_week
df_raw['promo2_since_week'] = df_raw.apply( lambda x: x['date'].week if math.isnan( x['pro

#promo2_since_year
df_raw['promo2_since_year'] = df_raw.apply( lambda x: x['date'].year if math.isnan( x['pro

#promo_interval
month_map = {1: 'Jan',  2: 'Fev',  3: 'Mar',  4: 'Apr',  5: 'May',  6: 'Jun',  7: 'Jul',


df_raw['promo_interval'].fillna(0, inplace=True )

df_raw['month_map'] = df_raw['date'].dt.month.map( month_map )


df_raw['is_promo'] = df_raw[['promo_interval', 'month_map']].apply( lambda x: 0 if x['prom
```

```
#Checking missings
df_raw.isnull().sum()
```

```
store                         0
day_of_week                   0
date                          0
sales                         0
```

```
customers                       0
open                            0
promo                           0
state_holiday                   0
school_holiday                  0
store_type                      0
assortment                      0
competition_distance            0
competition_open_since_month    0
competition_open_since_year     0
promo2                          0
promo2_since_week               0
promo2_since_year               0
promo_interval                  0
month_map                       0
is_promo                        0
dtype: int64
```

Change Datatypes

```
#Checking datatypes
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1017209 entries, 0 to 1017208
Data columns (total 20 columns):
 #   Column                        Non-Null Count    Dtype         
---  ------                        --------------    -----         
 0   store                         1017209 non-null  int64         
 1   day_of_week                   1017209 non-null  int64         
 2   date                          1017209 non-null  datetime64[ns]
 3   sales                         1017209 non-null  int64         
 4   customers                     1017209 non-null  int64         
 5   open                          1017209 non-null  int64         
 6   promo                         1017209 non-null  int64         
 7   state_holiday                 1017209 non-null  object        
 8   school_holiday                1017209 non-null  int64         
 9   store_type                    1017209 non-null  object        
 10  assortment                    1017209 non-null  object        
 11  competition_distance          1017209 non-null  float64       
 12  competition_open_since_month  1017209 non-null  float64       
 13  competition_open_since_year   1017209 non-null  float64       
 14  promo2                        1017209 non-null  int64         
 15  promo2_since_week             1017209 non-null  float64       
 16  promo2_since_year             1017209 non-null  float64       
 17  promo_interval                1017209 non-null  object        
 18  month_map                     1017209 non-null  object        
 19  is_promo                      1017209 non-null  int64         
dtypes: datetime64[ns](1), float64(5), int64(9), object(5)
memory usage: 163.0+ MB
```

```
# competiton
df_raw['competition_open_since_month'] = df_raw['competition_open_since_month'].astype( in
df_raw['competition_open_since_year'] = df_raw['competition_open_since_year'].astype( int

# promo2
```

```
df_raw['promo2_since_week'] = df_raw['promo2_since_week'].astype( int )
df_raw['promo2_since_year'] = df_raw['promo2_since_year'].astype( int )
```

Segregating Numerical and Categorical data for statistical analysis

```
num_data = df_raw.select_dtypes(include = ['int32','int64','float64'])
cat_data = df_raw.select_dtypes(exclude = ['int32','int64','float64','datetime64[ns]'])
```

```
num_data.columns
```

```
Index(['store', 'day_of_week', 'sales', 'customers', 'open', 'promo',
       'school_holiday', 'competition_distance',
       'competition_open_since_month', 'competition_open_since_year', 'promo2',
       'promo2_since_week', 'promo2_since_year', 'is_promo'],
      dtype='object')
```

```
cat_data.columns
```

```
Index(['state_holiday', 'store_type', 'assortment', 'promo_interval',
       'month_map'],
      dtype='object')
```

Numerical Variable Summary

For the numerical variables we use two types of analysis.

- **Central Tendency**
  - average
  - median
- **Dispersion**
  - std - standard deviation
  - min
  - max
  - range
  - skew
  - kurtosis

```
num_data.apply(continuous_var_summary).T
```

	dtype	cardinality	n_tot	n	nmiss	perc_miss
store	int64	1115	1017209	1017209	0	0.0
day_of_week	int64	7	1017209	1017209	0	0.0
sales	int64	21734	1017209	1017209	0	0.0
customers	int64	4086	1017209	1017209	0	0.0
open	int64	2	1017209	1017209	0	0.0
promo	int64	2	1017209	1017209	0	0.0
school_holiday	int64	2	1017209	1017209	0	0.0
competition_distance	float64	655	1017209	1017209	0	0.0
competition_open_since_month	int64	12	1017209	1017209	0	0.0
competition_open_since_year	int64	23	1017209	1017209	0	0.0
promo2	int64	2	1017209	1017209	0	0.0
promo2_since_week	int64	52	1017209	1017209	0	0.0

Let's take a look at sales feature

- Min = 0, means that on that day the store was closed.
- Max = 41551
- Range = 451151 - 0
- Mean = It tells us that on average, 5773 sales are made per day.
- Median = Median very close to the average
- Std = Tell us that our sales may vary by +/- 3849, that is, there are days that total sales are (5773 + 3849) and there are days that total sales are (5773 - 3849)
- Skew = Informs us how shifted our graph is in relation to the origin.
- Kurtosis = Tells us how "pointy" our distribution is, or how close to a normal distribution it is.

Categorical Data Summary

For categorical variables we will use the boxplot, which provides a visual analysis of the position, dispersion, symmetry, tails and outliers of the data set.

From the **boxplot** below we can get some conclusions:

- **state_holiday** --> Holiday b has a median higher than A but very similar to C.
- **store_type** --> type of store b has a lot more outliers and a larger amount of sales. Store b has a very large amount of data around the median. A and D have a similar dispersion around the median.
- **assortment** --> b has more sales compared to other types of assortment

```
cat_data.nunique()
```

```
state_holiday      5
store_type         4
assortment         3
promo_interval     4
month_map         12
dtype: int64
```

```
#Box Plots
aux1 = df_raw[(df_raw['state_holiday'] != '0') & (df_raw['sales'] > 0 )]

#StateHoliday vs Sales
plt.subplot(1,3,1)
sns.boxplot(x = 'state_holiday', y = 'sales', data = aux1)

#Store type vs Sales
plt.subplot(1,3,2)
sns.boxplot(x = 'store_type',y = 'sales', data = aux1)

#assortment vs Sales
plt.subplot(1,3,3)
sns.boxplot(x = 'assortment', y = 'sales', data = aux1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4268731ed0>
```

Feature Engineering

To get a better understanding of data and to find out insights about data we will create some addi

-year
-month
-day
-week of day
-year week = Ex: 2019(year)-22(week)
Some variables have been modified to suit our questions. We replaced the rows in the "state_holida
column with "public_holiday", "easter_holiday" and "christmas". We did the same thing to our assor

```
# year
df_raw['year'] = df_raw['date'].dt.year

# month
df_raw['month'] = df_raw['date'].dt.month

# day
df_raw['day'] = df_raw['date'].dt.day

# week of year
df_raw['week_of_year'] = df_raw['date'].dt.weekofyear

# year week
df_raw['year_week'] = df_raw['date'].dt.strftime( '%Y-%W' )

# competition since
df_raw['competition_since'] = df_raw.apply( lambda x: datetime.datetime( year=x['competiti
df_raw['competition_time_month'] = ( ( df_raw['date'] - df_raw['competition_since'] )/30 )

#promo since
df_raw['promo_since'] = df_raw['promo2_since_year'].astype(str) + '-' + df_raw['promo2_sin
df_raw['promo_since'] = df_raw['promo_since'].apply(lambda x: datetime.datetime.strptime(x
df_raw['promo_time_week'] = ((df_raw['date'] - df_raw['promo_since'])/7).apply(lambda x: x

#assortment
#a = basic, b = extra, c = extended

df_raw['assortment'] = df_raw['assortment'].apply(lambda x: 'basic' if x == 'a' else 'extr

#state_holiday
#a = public_holiday, b = easter holiday c = christmas

df_raw['state_holiday'] = df_raw['state_holiday'].apply(lambda x: 'public_holiday' if x ==
```

Filtering rows

```
#we will remove the data whose column "open" is equal to "0", as it will mean that the sto
#that is, there were no sales on that day. We will also remove the lines that have in the
df_raw = df_raw[(df_raw['open'] != 0) & (df_raw['sales'] > 0)]
```

Filtering Columns

In a similar way, we will remove the columns. These will be:

- **Costumer** - To use this column, we would apparently have to create a model to predict customers. Therefore,right now, it will not help us.
- **Open** - As we've removed all the lines that have the value "0", so we will only have those that have the value "1". Since this column will have no variation it will not influence our model or the analysis.
- **Promo_interval** / **month_map** - We will remove it because we used it to build columns that will give to us the same information,so, from the column "promo_interval" and the column "month_map" we've built the column " is_promo "in the Fillout NA.

```
df_raw.drop(columns = ['customers','open','promo_interval','month_map'], axis = 1, inplace
df_raw.columns
```

```
Index(['store', 'day_of_week', 'date', 'sales', 'promo', 'state_holiday',
       'school_holiday', 'store_type', 'assortment', 'competition_distance',
       'competition_open_since_month', 'competition_open_since_year', 'promo2',
       'promo2_since_week', 'promo2_since_year', 'is_promo', 'year', 'month',
       'day', 'week_of_year', 'year_week', 'competition_since',
       'competition_time_month', 'promo_since', 'promo_time_week'],
      dtype='object')
```

EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is the process of visualizing and analyzing data to extract insights from it. In other words, EDA is the process of summarizing important characteristics of data in order to gain better understanding of the dataset. Therefore, this part will be done in the following three steps:

- Univariate Analysis
- Bivariate Analysis
- Multivariate Analysis

Univariate Analysis

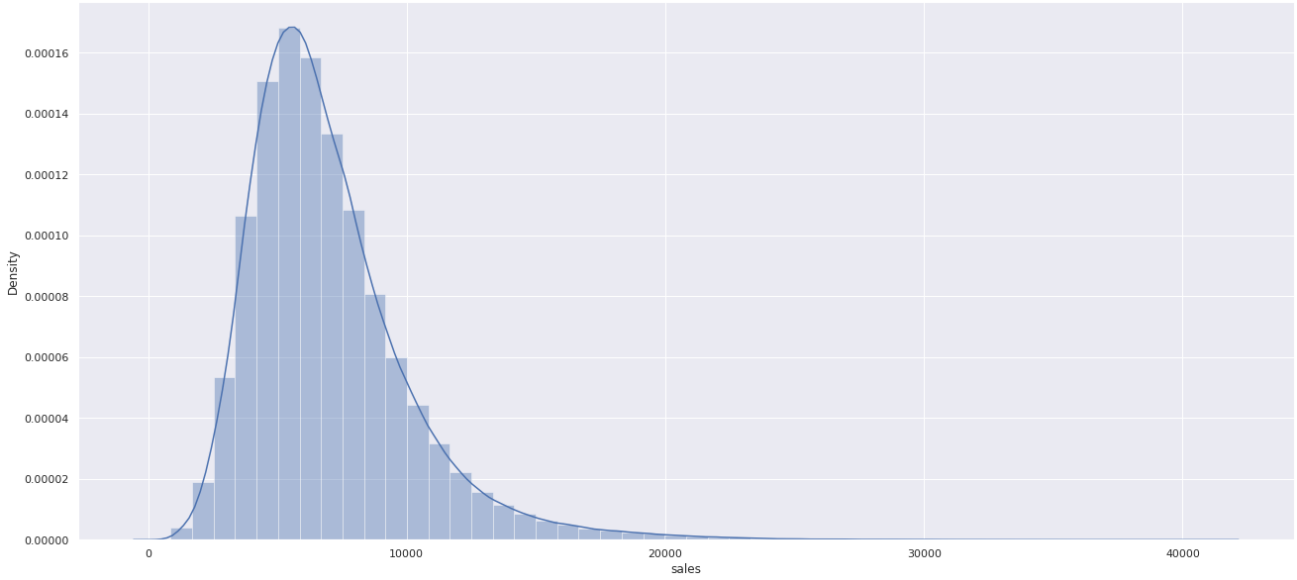
Univariate analysis is the simplest form of analyzing data. “Uni” means “one”, so in other words your data has only one variable. It doesn’t deal with causes or relationships (unlike regression)

and it's major purpose is to describe; It takes data, summarizes that data and finds patterns in the data.

Target Variable(sales)

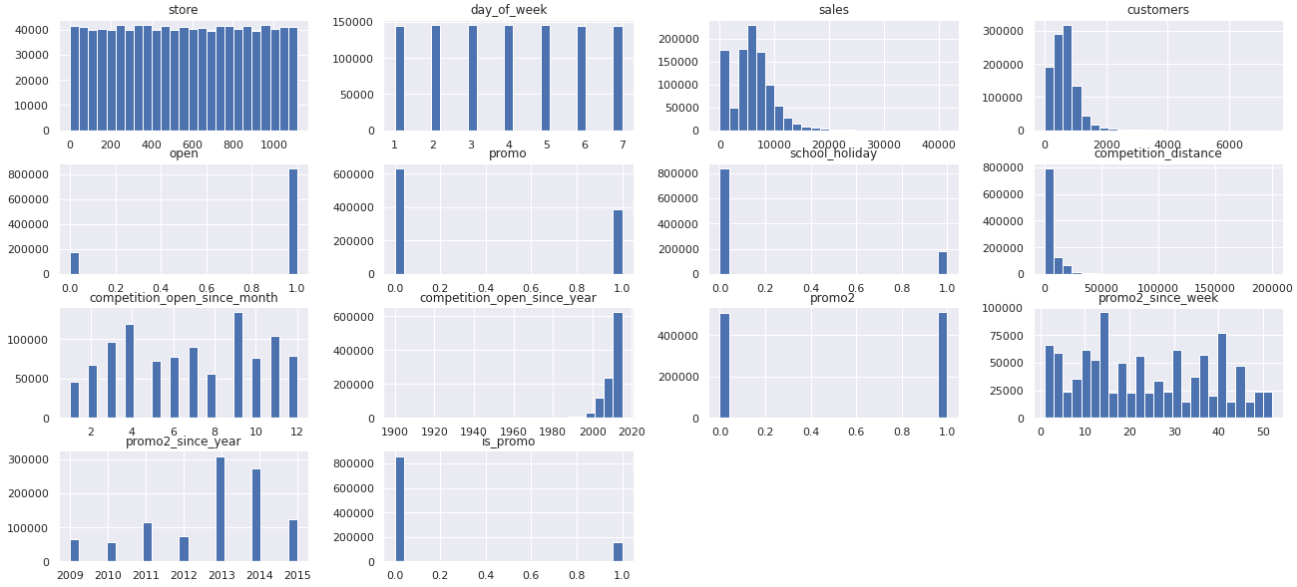
```
#Distplot for sales feature
sns.distplot(df_raw['sales'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4269e64c90>
```



Univariate Analysis for Numerical Variables

```
num_data.hist(bins = 25);
```



Insights from univariate Analysis:

- **competition_distance** = there are more number of stores at small distance.
- **competition_open_since_month** -> has an increase until the fourth month and reaches the maximum. From that there is a fall. Therefore, this feature has a certain variation.
- **day_of_week** -> There is no variation, so, the day of the week will not influence sales. There is no variation.
- **is_promo** -> we have a lot more sales when there are no promotions.
- promo2_since_year -> there is a very high peak in 2013, we need to check what happened that year.

Univariate Analysis for Categorical Variables

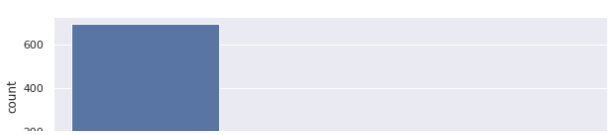
```
df_raw.select_dtypes(include = 'object').columns
```

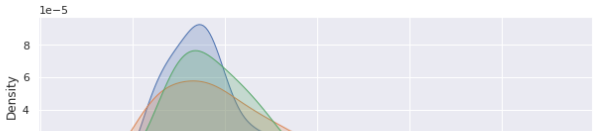
```
Index(['state_holiday', 'store_type', 'assortment', 'year_week'], dtype='object')
```

```
#state_holiday
a = df_raw[df_raw['state_holiday'] != 'regular_day']
plt.subplot(3,2,1)
```

```
sns.countplot(a['state_holiday'])

plt.subplot(3,2,2)
sns.kdeplot(df_raw[df_raw['state_holiday'] == 'public_holiday']['sales'],label = 'public_h
sns.kdeplot(df_raw[df_raw['state_holiday'] == 'easter_holiday']['sales'],label = 'easter_h
sns.kdeplot(df_raw[df_raw['state_holiday'] == 'christmas']['sales'],label = 'christmas',sh

#store_type
plt.subplot(3,2,4)
sns.kdeplot(df_raw[df_raw['store_type'] == 'a']['sales'],label = 'a',shade = True)
sns.kdeplot(df_raw[df_raw['store_type'] == 'b']['sales'],label = 'b',shade = True)
sns.kdeplot(df_raw[df_raw['store_type'] == 'c']['sales'],label = 'c',shade = True)
sns.kdeplot(df_raw[df_raw['store_type'] == 'd']['sales'],label = 'd',shade = True)

plt.subplot(3,2,3)
sns.countplot(df_raw['store_type'])

#assortment
plt.subplot(3,2,5)
sns.countplot(df_raw['assortment'])

plt.subplot(3,2,6)
sns.kdeplot(df_raw[df_raw['assortment'] == 'basic']['sales'], label  = 'basic', shade = Tr
sns.kdeplot(df_raw[df_raw['assortment'] == 'extended']['sales'], label = 'extended', shade
sns.kdeplot(df_raw[df_raw['assortment'] == 'extra']['sales'], label = 'extra', shade = Tru
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427debde50>
```





From all the plots above, we can get some conclusions:

- **state_holiday** -> We have a much larger amount of sales on public holidays, but at Christmas, which has a smaller amount of sales than easter_holiday, it has a higher peak.
- **store_type** -> The store_type "a" that sells more, does not have such a peak compared to the others.
- **assortment** -> We see that stores with the "extra" type assortment sell less, but have a higher distribution. So, there are stores that sell more with the "extra" assortment and stores that sell less.

Bivariate Analysis

- Assumptions
  - 1. Stores with a larger assortment should sell more.
  - 2. Stores with closer competitors should sell less
  - 3. Stores with longer competitors should sell more
  - 4. Stores with active promotions for longer should sell more.
  - 5. Stores with more promotion days should sell more
  - 6. Stores with more consecutive promotions should sell more.
  - 7. Stores open during the holiday should sell more
  - 8. Stores should sell more over the years.
  - 9. Stores should sell more in the second half of the year.
  - 10. Stores should sell more after the 10th of each month
  - 11. Stores should sell less on weekends.
  - 12. Stores should sell less during school holidays.

Hypothesis 1 Stores with a larger assortment should sell more.

Result

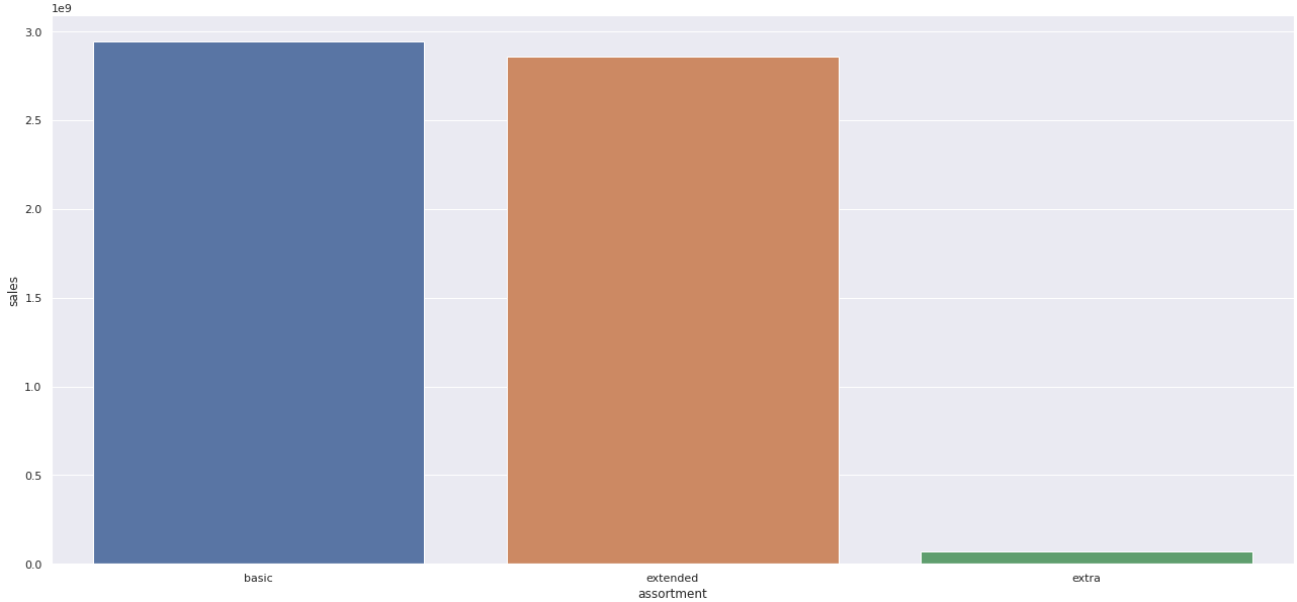
False Stores with a larger assortment sell less.

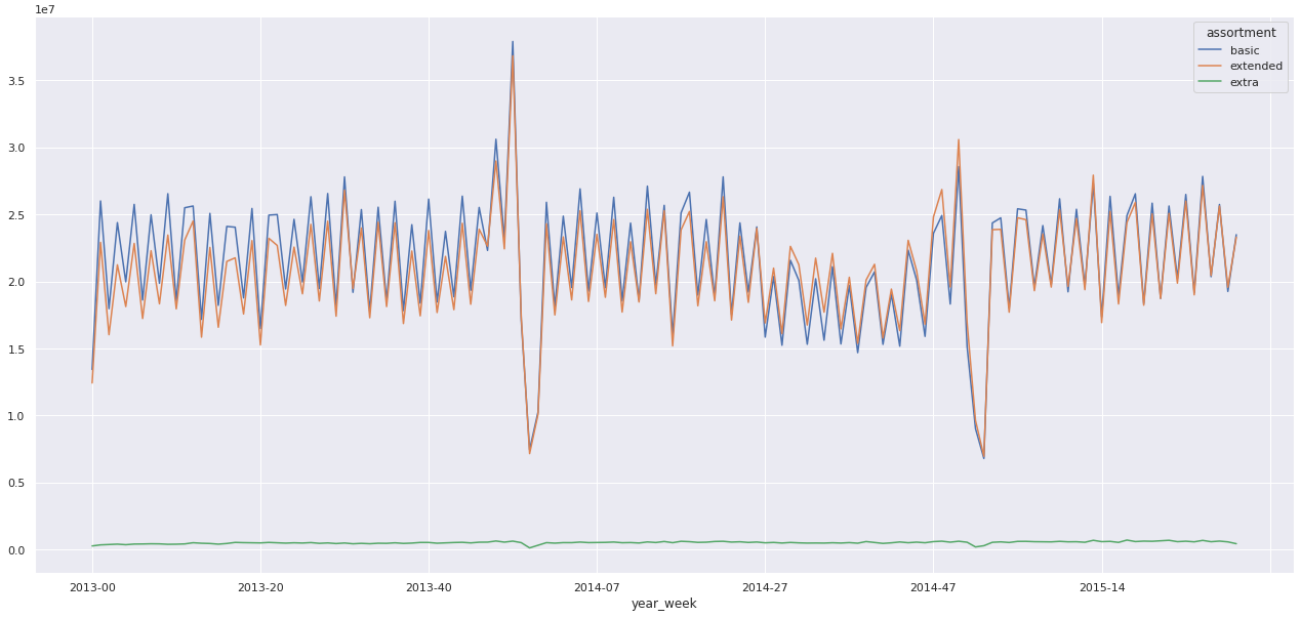
```
aux1 = df_raw[['assortment','sales']].groupby('assortment').sum().reset_index()
sns.barplot(x = 'assortment', y = 'sales', data = aux1);
```

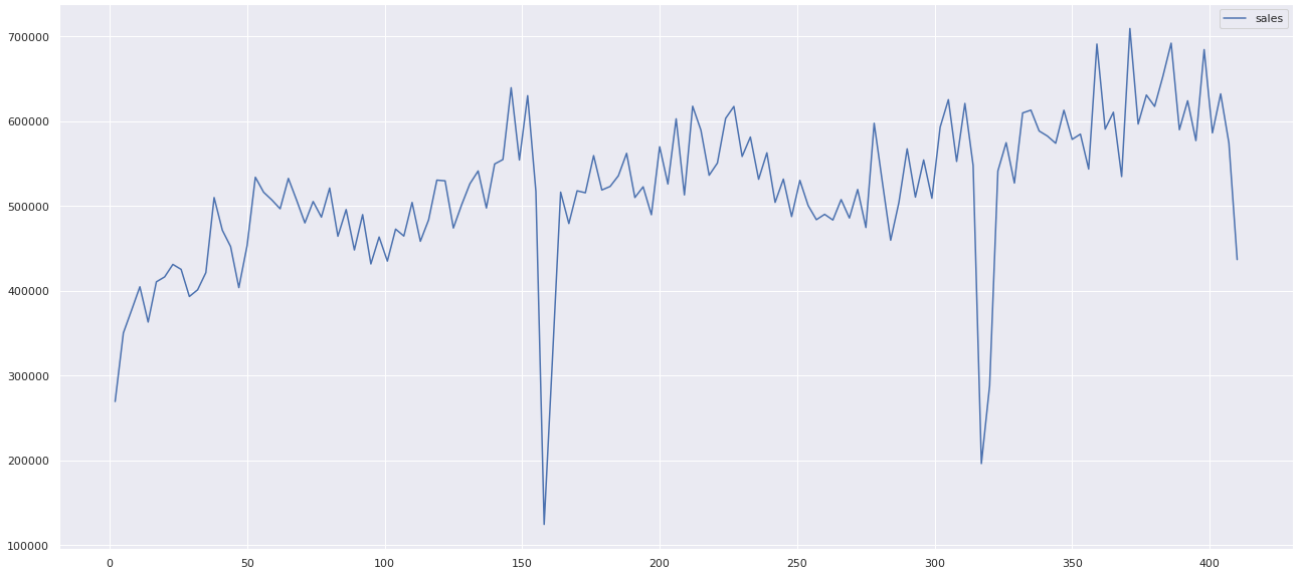
```
aux2 = df_raw[['assortment','sales','year_week']].groupby(['year_week','assortment']).sum(
aux3 = aux2.pivot(index = 'year_week', columns = 'assortment', values = 'sales')
aux3.plot()
```

```
aux4 = aux2[aux2['assortment'] == 'extra']
aux4.pivot(index = 'year_week',columns = 'assortment', values = 'sales')
aux4.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427df61f90>
```

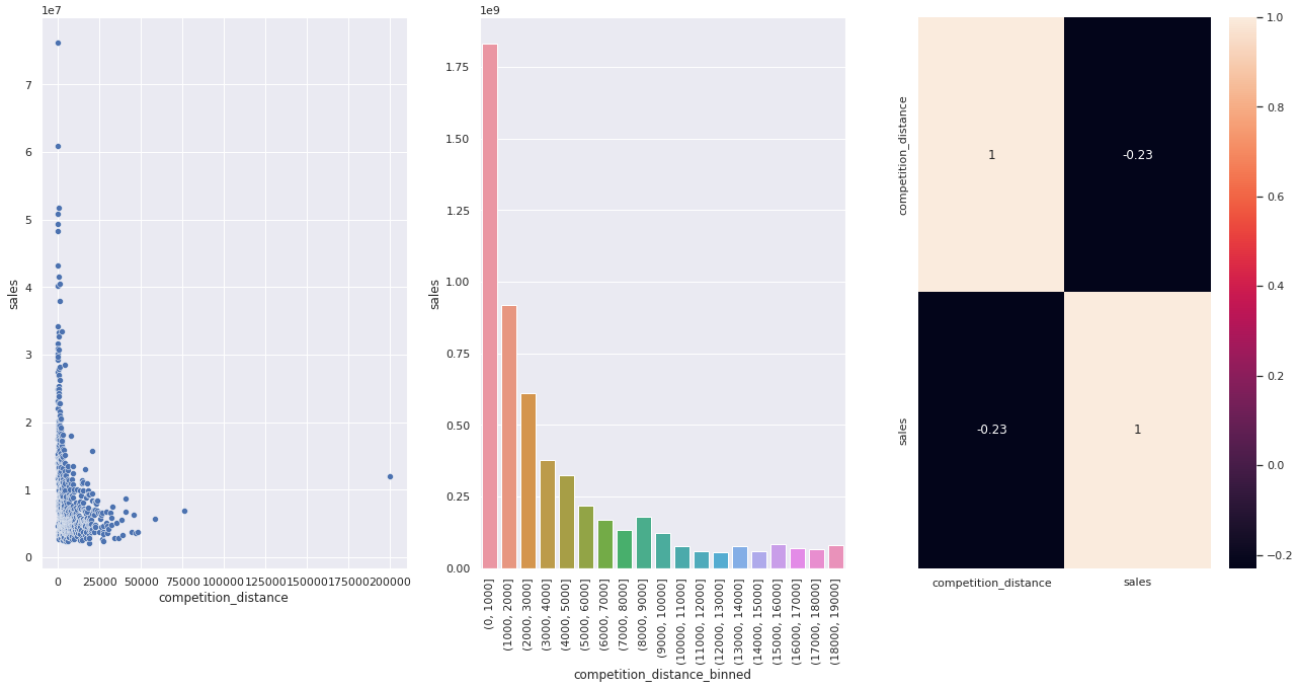






H2 Stores with closer competitors sell less False Stores with closer competitors sell more.

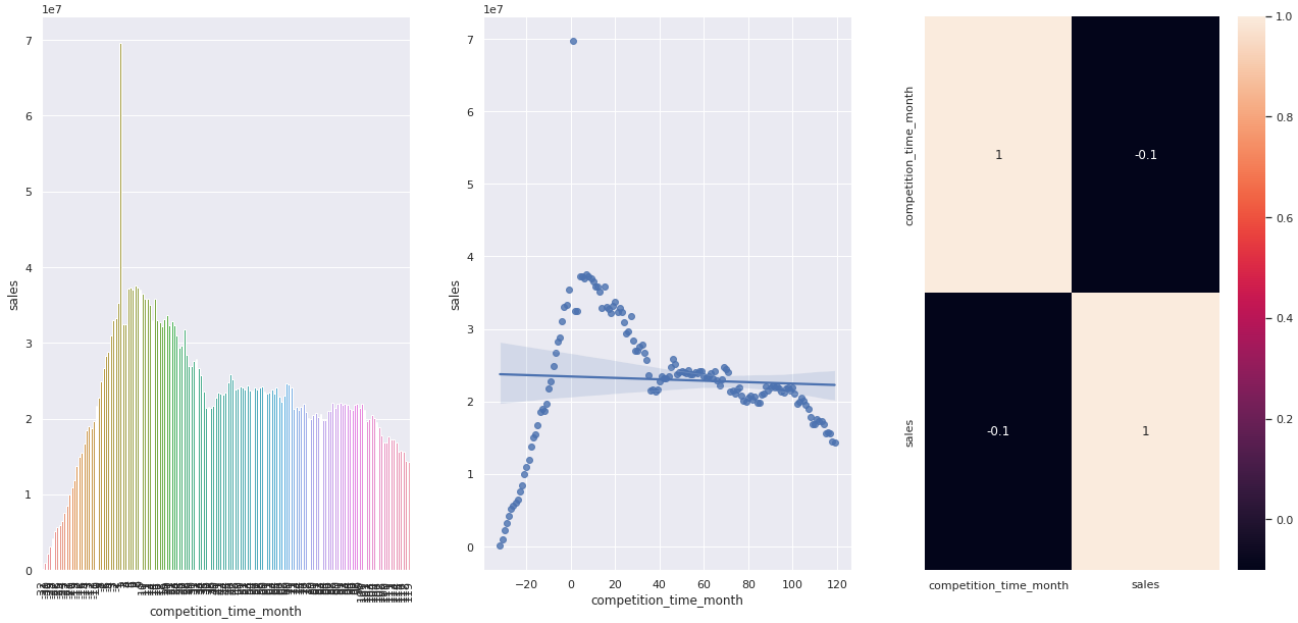
```
plt.subplot(1,3,1)
aux1 = df_raw[['competition_distance','sales']].groupby('competition_distance').sum().rese
sns.scatterplot(x = 'competition_distance', y = 'sales', data = aux1);

plt.subplot(1,3,2)
bins = list(np.arange(0,20000,1000))
aux1['competition_distance_binned'] = pd.cut(aux1['competition_distance'], bins = bins)
aux2 = aux1[['competition_distance_binned','sales']].groupby('competition_distance_binned'
sns.barplot(x = 'competition_distance_binned', y = 'sales', data = aux2);
plt.xticks(rotation = 90)

plt.subplot(1,3,3)
sns.heatmap(aux1.corr(method = 'pearson'), annot = True);
```



H3. Stores with longer competitors should sell more. False Stores with longer competitors sell less.

```
plt.subplot(1,3,1)
aux1 = df_raw[['competition_time_month','sales']].groupby('competition_time_month').sum().
aux2 = aux1[(aux1['competition_time_month'] < 120) & (aux1['competition_time_month'] != 0
sns.barplot(x = 'competition_time_month', y = 'sales', data = aux2);
plt.xticks(rotation = 90);

plt.subplot(1,3,2)
sns.regplot(x = 'competition_time_month', y = 'sales', data = aux2);

plt.subplot(1,3,3)
sns.heatmap(aux1.corr(method = 'pearson'), annot = True)
```

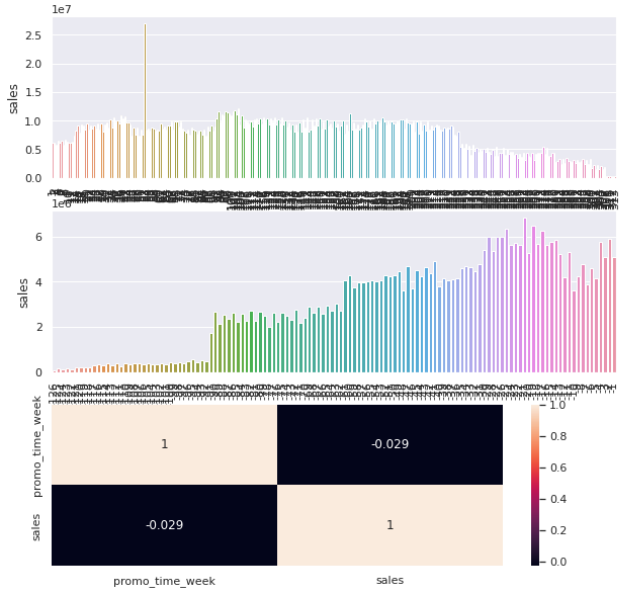
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427e1e8ed0>
```

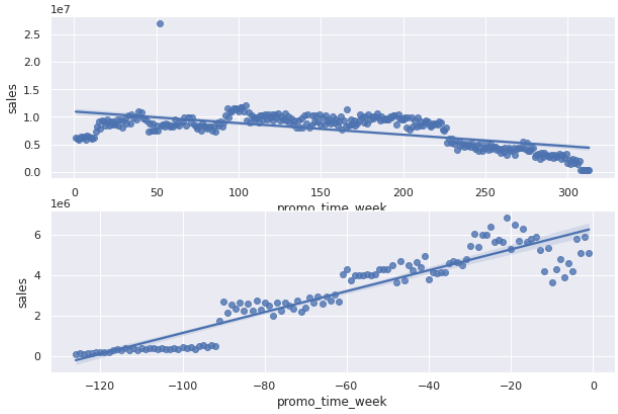


H4. Stores with active promotions for longer should sell more. False Stores with active promotions sell less after a certain duration.

```
aux1 = df_raw[['promo_time_week','sales']].groupby('promo_time_week').sum().reset_index()

plt.subplot(3,2,1)
```

```
aux2 = aux1[aux1['promo_time_week'] > 0] #periodo extendido
sns.barplot(x = 'promo_time_week',y = 'sales', data = aux2);
plt.xticks(rotation = 90);

plt.subplot(3,2,2)
sns.regplot(x = 'promo_time_week',y = 'sales', data = aux2);

plt.subplot(3,2,3)
aux3 = aux1[aux1['promo_time_week'] < 0] #promo regular
sns.barplot(x = 'promo_time_week',y = 'sales', data = aux3);
plt.xticks(rotation = 90);### H4. Lojas com promoções ativas por mais tempo deveriam vende

plt.subplot(3,2,4)
sns.regplot(x = 'promo_time_week',y = 'sales', data = aux3);

plt.subplot(3,2,5)
sns.heatmap(aux1.corr(method = 'pearson'), annot = True );
```





H6. Stores with more consecutive promotions should sell more. False Stores with more consecutive promotions sell less.

```
df_raw[['promo','promo2','sales']].groupby(['promo','promo2']).sum().reset_index()
```

	promo	promo2	sales
0	0	0	1482612096
1	0	1	1289362241
2	1	0	1628930532
3	1	1	1472275754

```
aux1 = df_raw[(df_raw['promo'] == 1) & (df_raw['promo2'] == 1)][['year_week','sales']].gro
ax = aux1.plot()
```

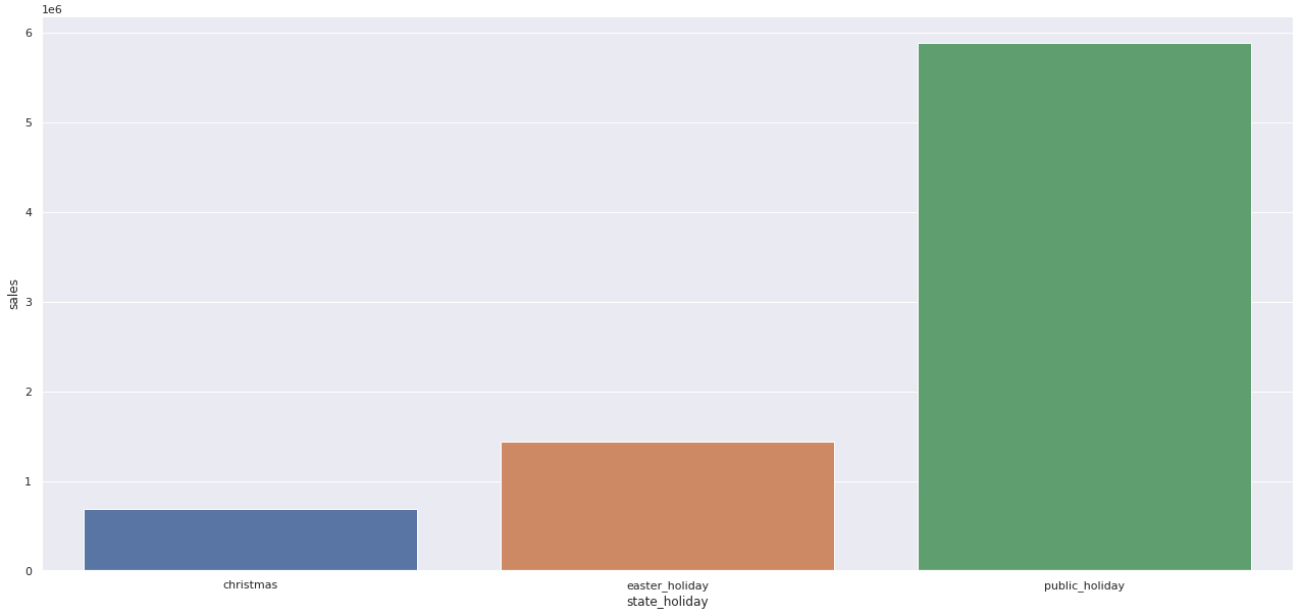
```
aux2 = df_raw[(df_raw['promo'] == 1) & (df_raw['promo2'] == 0)][['year_week','sales']].gro
aux2.plot(ax = ax)
ax.legend(labels = ['Tradicional & Extendida', 'Extendida'])
```

```
<matplotlib.legend.Legend at 0x7f427e82cfd0>
```

H7. Stores that were opened during the Christmas holiday should sell more. False Stores open during the Christmas holiday sell less.

```
aux1 = df_raw[['state_holiday','sales']].groupby('state_holiday').sum().reset_index()

aux2 = aux1[aux1['state_holiday'] != 'regular_day']
sns.barplot(x = 'state_holiday', y = 'sales', data = aux2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427daaaa90>
```

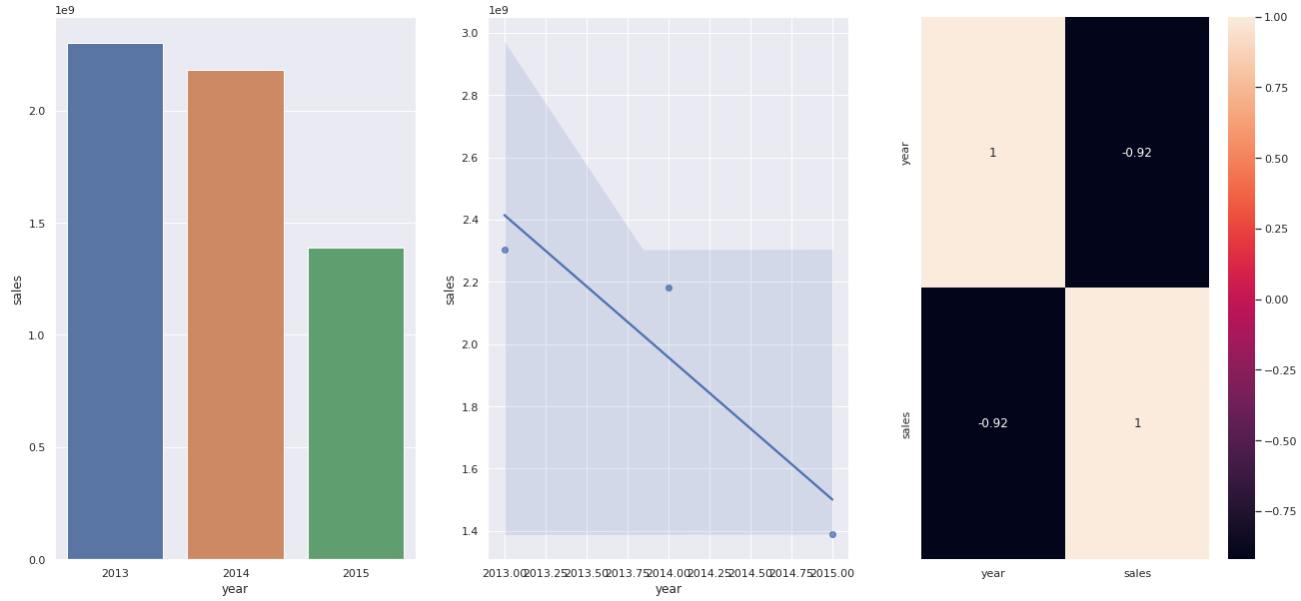


H8. Stores should sell more over the years. False Stores sell less over the years.

```
aux1 = df_raw[['year','sales']].groupby('year').sum().reset_index()

plt.subplot(1,3,1)
sns.barplot(x = 'year',y = 'sales', data = aux1)

plt.subplot(1,3,2)
sns.regplot(x = 'year',y = 'sales', data = aux1)
```

```
plt.subplot(1,3,3)
sns.heatmap(aux1.corr(method = 'pearson'), annot = True)
```

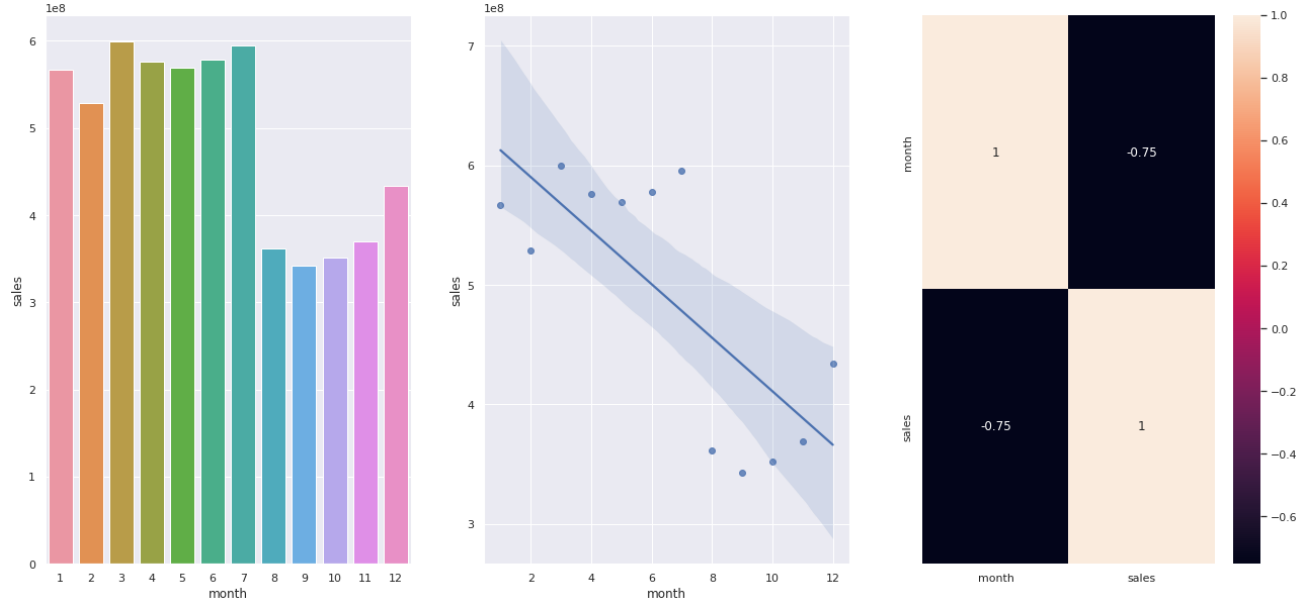
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427dcbf290>
```



H9. Stores should sell more in the second half of the year. False Stores sell less in the second half of the year.

```
aux1 = df_raw[['month','sales']].groupby('month').sum().reset_index()

plt.subplot(1,3,1)
sns.barplot(x = 'month',y = 'sales', data = aux1)

plt.subplot(1,3,2)
sns.regplot(x = 'month',y = 'sales', data = aux1)

plt.subplot(1,3,3)
sns.heatmap(aux1.corr(method = 'pearson'), annot = True)
```

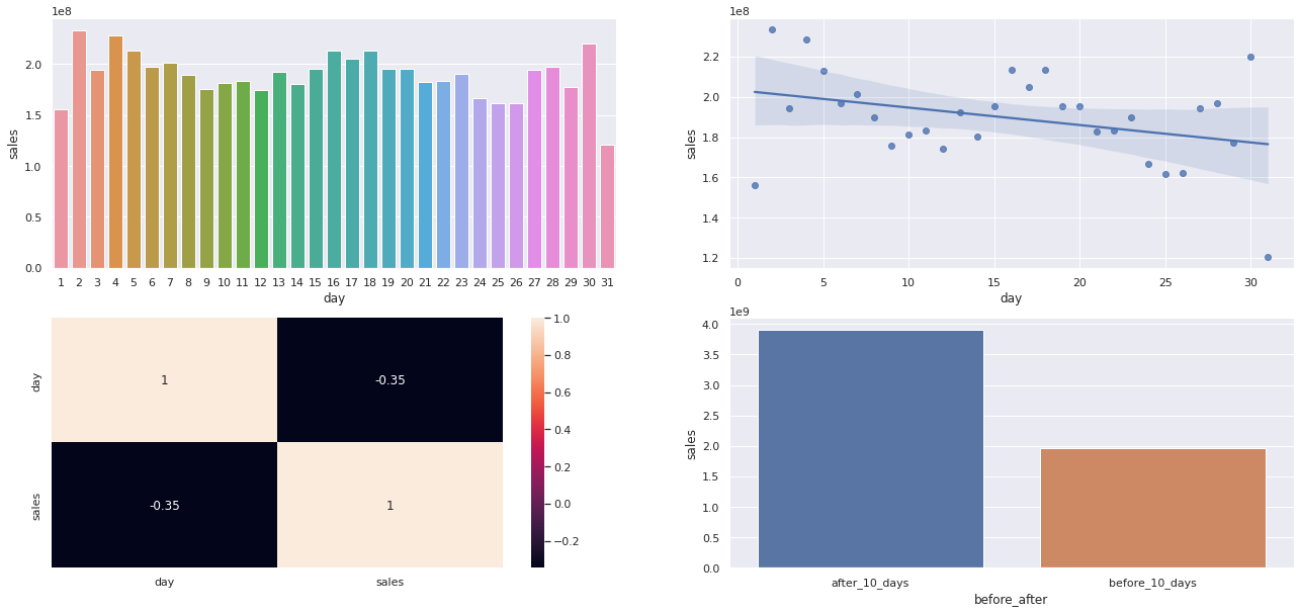
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427dad6710>
```



H10. Stores should sell more after the 10th of each month. True Stores sell more after the 10th of each month.

```
aux1 = df_raw[['day','sales']].groupby('day').sum().reset_index()

plt.subplot(2,2,1)
sns.barplot(x = 'day',y = 'sales', data = aux1)

plt.subplot(2,2,2)
sns.regplot(x = 'day',y = 'sales', data = aux1)

plt.subplot(2,2,3)
sns.heatmap(aux1.corr(method = 'pearson'), annot = True)

plt.subplot(2,2,4)
aux1['before_after'] = aux1['day'].apply(lambda x:'before_10_days' if x <= 10 else 'after_
aux2 = aux1[['before_after','sales']].groupby('before_after').sum().reset_index()
sns.barplot(x = 'before_after', y = 'sales', data = aux2)
```

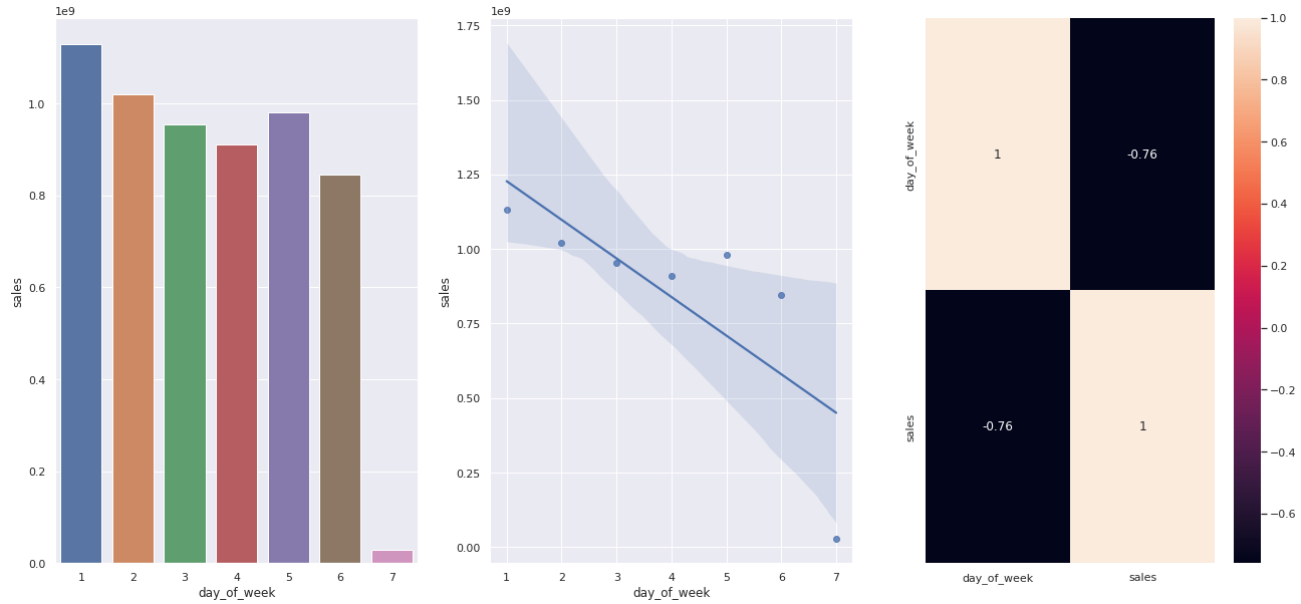
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427d89f850>
```



H11. Stores should sell less on weekends. True Stores sell less on weekends.

```
aux1 = df_raw[['day_of_week','sales']].groupby('day_of_week').sum().reset_index()

plt.subplot(1,3,1)
sns.barplot(x = 'day_of_week',y = 'sales', data = aux1)

plt.subplot(1,3,2)
sns.regplot(x = 'day_of_week',y = 'sales', data = aux1)

plt.subplot(1,3,3)
sns.heatmap(aux1.corr(method = 'pearson'), annot = True)
```

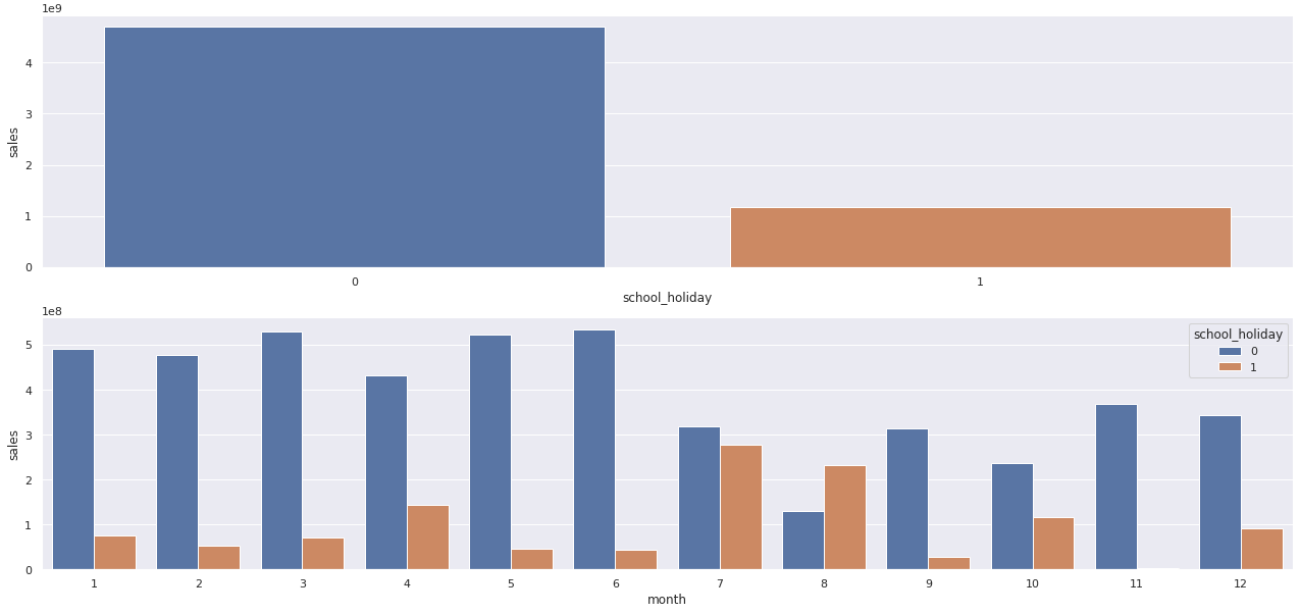
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427df0aa90>
```



H12. Stores should sell less during school holidays True Stores sell less during school holidays except in July and August.

```
plt.subplot(2,1,1)
aux1 = df_raw[['school_holiday','sales']].groupby('school_holiday').sum().reset_index()
sns.barplot(x = 'school_holiday',y = 'sales', data = aux1)

plt.subplot(2,1,2)
aux2 = df_raw[['month','school_holiday','sales']].groupby(['month','school_holiday']).sum(
sns.barplot(x = 'month',y = 'sales',hue = 'school_holiday', data = aux2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427eabfd90>
```



Multivariate Analysis

Multivariate analysis (MVA) is based on the principles of multivariate statistics, which involves observation and analysis of more than one statistical outcome variable at a time. Typically, MVA is used to address the situations where multiple measurements are made on each experimental unit and the relations among these measurements and their structures are important.

This step will be done because the machine learning algorithms assume some premises, among them, the Occam's razor principle. Ockham's rule is associated with the requirement to recognize, for each object analyzed, only one sufficient explanation.

We use the concepts of linear dependency, that is, let's assume that we have two columns that are linearly dependent, that is, whose influence on the problem is similar, we can take one that the effect will be maintained. To find this, we can look at the correlation between variables as an alternative so that we can reduce the dimensionality of our dataset.

This stage will be divided into two:

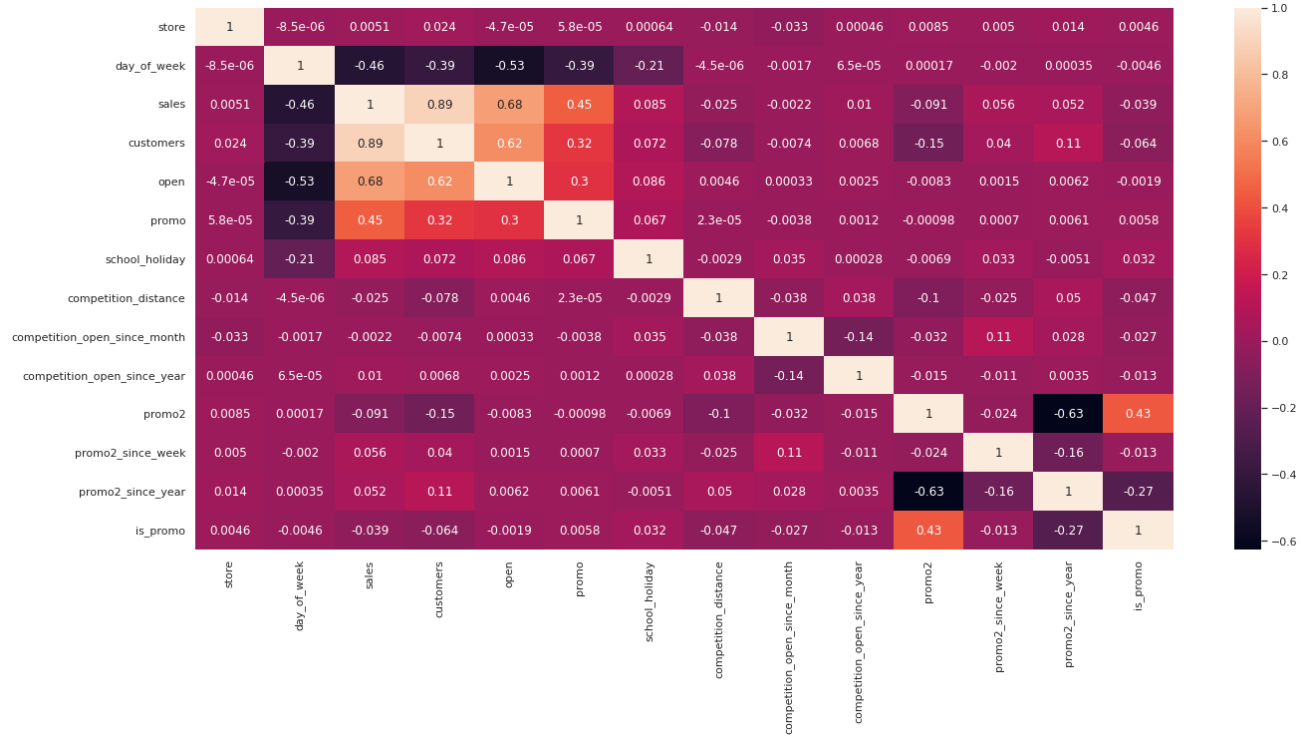
Numeric attributes Categorical attributes Let's use this strategy because the multivariate analysis of our variables will be different. That is, to check the correlation between numerical variables, we will use the Pearson method and for categorical variables, Cramér's V method.

Multivariate Analysis for Numerical Attributes

Correlation

```
#separating numerical variables and building the heat map
correlation = num_data.corr(method = 'pearson')
sns.heatmap(correlation, annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427ebc0e10>
```



Multivariate Analysis for Categorical Attributes

Cramer's V

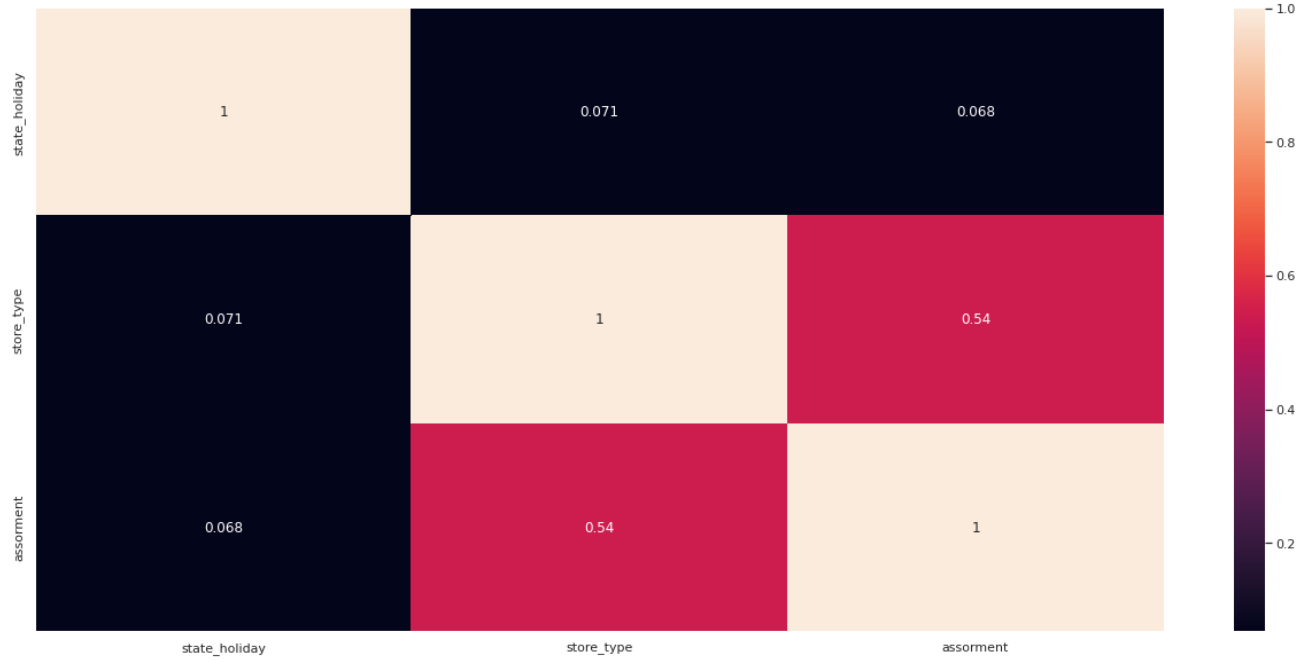
- It is used to understand the strength of the relationship between two variables. To use it, your variables of interest should be categorical with two or more unique values per

```
#only categorical data
a = df_raw.select_dtypes(include = 'object')
a.head()
```

	state_holiday	store_type	assortment	year_week
0	regular_day	c	basic	2015-30
1	regular_day	a	basic	2015-30
2	regular_day	a	basic	2015-30
3	regular_day	c	extended	2015-30
4	regular_day	a	basic	2015-30

```
def cramer_v(x,y):
    cm = pd.crosstab(x,y).values
    chi2 = stats.chi2_contingency( cm )[0]
    n = cm.sum()
    r, k = cm.shape
    chi2corr = max(0,chi2 - (k-1)*(r-1)/(n-1))
    kcorr = k - (k-1)**2/(n-1)

    rcorr = r - (r-1)**2/(n-1)

    return np.sqrt((chi2/n)/ (min(kcorr-1,rcorr-1)))


#calculate cramer v
a1 = cramer_v(a['state_holiday'],a['state_holiday'])
a2 = cramer_v(a['state_holiday'],a['store_type'])
a3 = cramer_v(a['state_holiday'],a['assortment'])

a4 = cramer_v(a['store_type'],a['state_holiday'])
a5 = cramer_v(a['store_type'],a['store_type'])
a6 = cramer_v(a['store_type'],a['assortment'])

a7 = cramer_v(a['assortment'],a['state_holiday'])
a8 = cramer_v(a['assortment'],a['store_type'])
a9 = cramer_v(a['assortment'],a['assortment'])

d = pd.DataFrame({'state_holiday': [a1,a2,a3],
              'store_type': [a4,a5,a6],
              'assorment': [a7,a8,a9]
             })

#final dataset
d = d.set_index(d.columns)

sns.heatmap(d , annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427dfd9090>
```



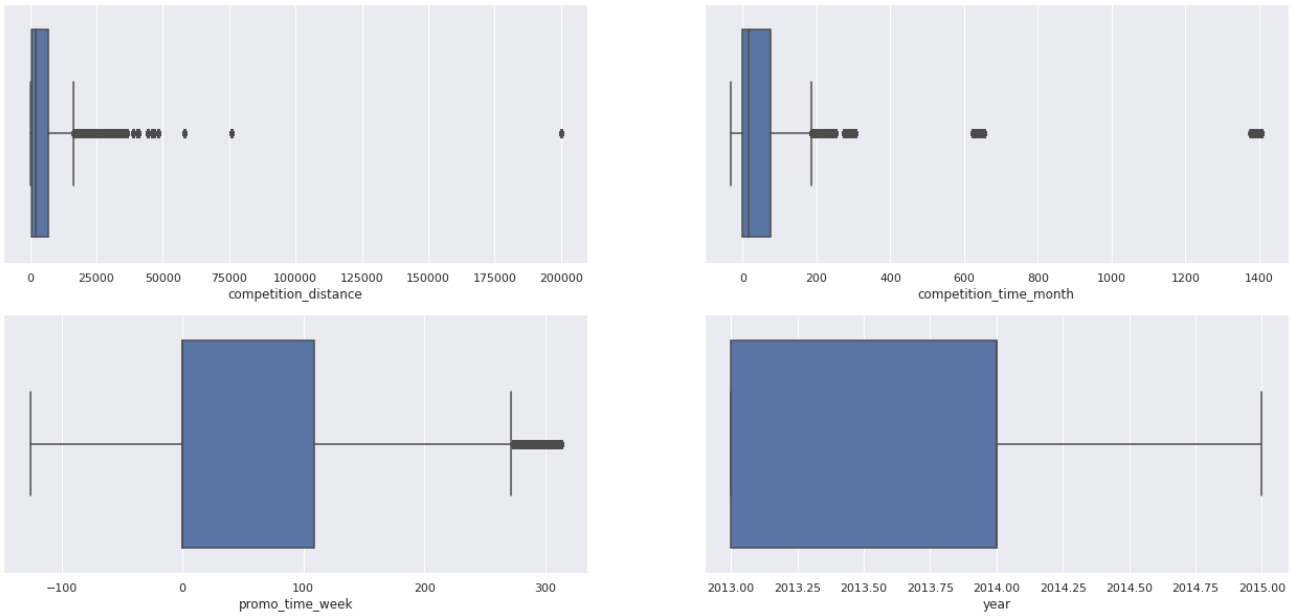
Rescaling

```
#selecting numerical features
num_var = df_raw.select_dtypes(include = ['int64','float64','int32'])
num_var.head()
```

	store	day_of_week	sales	promo	school_holiday	competition_distance	competitio
0	1	5	5263	1	1	1270.0	
1	2	5	6064	1	1	570.0	
2	3	5	8314	1	1	14130.0	
3	4	5	13995	1	1	620.0	
4	5	5	4822	1	1	29910.0	

```
#boxplots
plt.subplot(2,2,1)
sns.boxplot(df_raw['competition_distance'])

plt.subplot(2,2,2)
sns.boxplot(df_raw['competition_time_month'])

plt.subplot(2,2,3)
sns.boxplot(df_raw['promo_time_week'])

plt.subplot(2,2,4)
sns.boxplot(df_raw['year'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427e92c190>
```



competition_distance -> There are many outliers, so we will use RobustScaler.

year -> There are no outliers, so we will use MinMaxScaler.

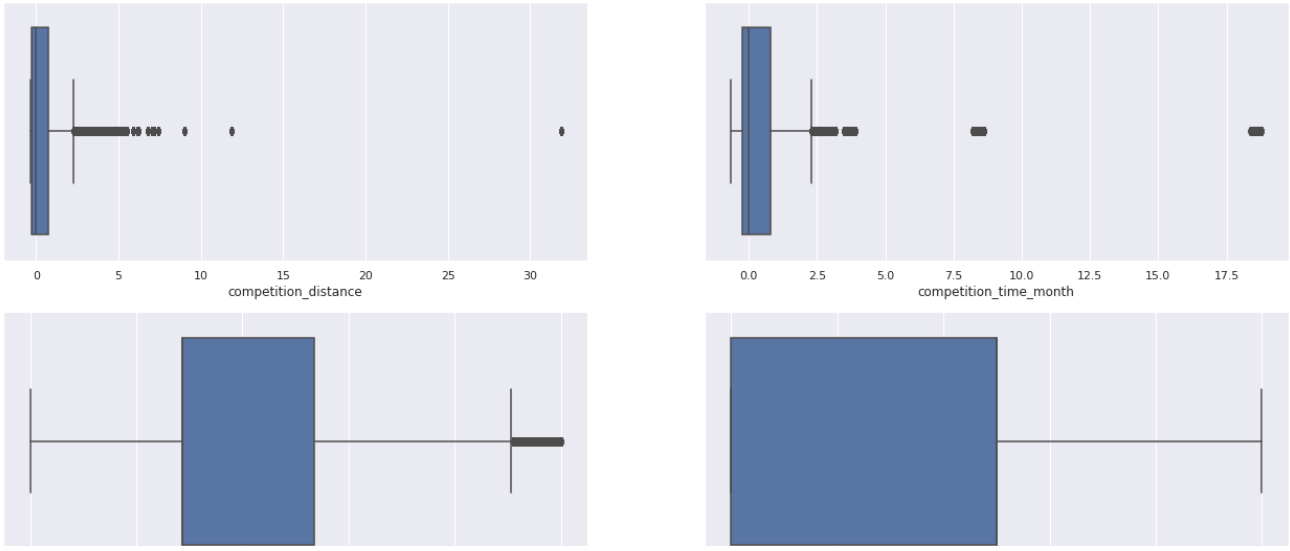
competition_time_month -> There are many outliers, so we will use RobustScaler.

promo_time_week -> There aren't many outliers, so we will use MinMaxScaler.

Outlier Treatment

```
rs = RobustScaler()
mms = MinMaxScaler()

#competition_distance - RobustScaler
df_raw['competition_distance'] = rs.fit_transform(df_raw[['competition_distance']].values)
pickle.dump(rs, open("/content/drive/MyDrive/Google Collab/pickle_folder/competition_dista

#year - MinMaxScaler
df_raw['year'] = mms.fit_transform(df_raw[['year']].values)
pickle.dump(mms, open("/content/drive/MyDrive/Google Collab/pickle_folder/year_scaler.pkl"

#competition_time_month - RobustScaler
df_raw['competition_time_month'] = rs.fit_transform(df_raw[['competition_time_month']].val
pickle.dump(rs, open("/content/drive/MyDrive/Google Collab/pickle_folder/competition_time_

#promo_time_week - MinMaxScaler
df_raw['promo_time_week'] = mms.fit_transform(df_raw[['promo_time_week']].values)
pickle.dump(rs, open("/content/drive/MyDrive/Google Collab/pickle_folder/promo_time_week_s


#boxplots
plt.subplot(2,2,1)
sns.boxplot(df_raw['competition_distance'])

plt.subplot(2,2,2)
sns.boxplot(df_raw['competition_time_month'])

plt.subplot(2,2,3)
sns.boxplot(df_raw['promo_time_week'])

plt.subplot(2,2,4)

sns.boxplot(df_raw['year'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427e7d2550>
```



Transformation

In this stage of data transformation, we will verify three techniques:

- Encoding
- Transformation of the variable response
- Transformation of nature.

Encoding

- **Label Enconder**

LabelEncoder is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1. This is sometimes useful for writing efficient Cython routines.It can also be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.

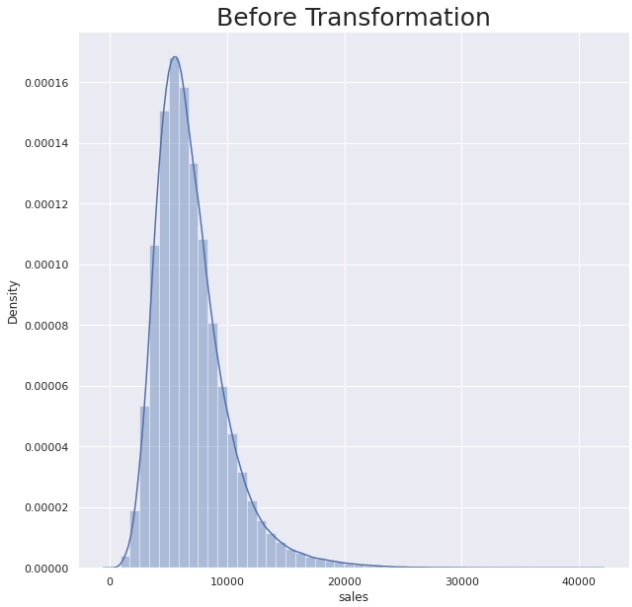
- **Ordinal Encoder** To convert categorical features to such integer codes, we can use the OrdinalEncoder. This estimator transforms each categorical feature to one new feature of integers (0 to n_categories - 1) as being ordered.

```
#state_holiday - OneHotEncoding
df_raw = pd.get_dummies(df_raw, prefix = ['state_holiday'], columns = ['state_holiday'])


#state_holiday - OneHotEncoding
df_raw = pd.get_dummies(df_raw, prefix = ['store_type'], columns = ['store_type'])


#assortment - OrdinalEncoder
assortment_dict = {'basic': 1,
                   'extra': 2,
                   'extended': 3}
df_raw['assortment'] = df_raw['assortment'].map(assortment_dict)
```

Response Variable Transformation

In transforming the response variable, we have to approximate the distribution of the response variable to a normal distribution. This is necessary because machine learning algorithms are built on the basis of assumptions, one of which is that the data is normally distributed.For that to happen, we will use the logarithmic transformation.

```
plt.subplot(1,2,1)
sns.distplot(df_raw['sales'])
plt.title('Before Transformation', fontsize = 25)

plt.subplot(1,2,2)
sns.distplot(np.log1p(df_raw['sales']))
plt.title("After Transformation", fontsize = 25)

#logarithmic transformation
df_raw['sales'] = np.log1p(df_raw['sales'])
```



** Nature Transformation**

In nature transformation, we have to bring the true nature of the data to the dataset. In this case, the variable 'month' is cyclical and has to be transformed, as the months are repeated each year that begins. But for that, we don't just list the months 1 to 12 because we lost the sense of cycle due to the different distances.

For example, we take the month of January 2018, we find that there is a long distance until December 2018, but not necessarily, December 2018 is far from the month of January 2019. What happens is that December 2018 has the same distance as January 2019 than March 2018 has April 2018. And the same distance keeps a cycle.

Therefore, I will use the trigonometric circle by placing the months as arcs and separating them at equal distances. This process will be done for the variables 'day_of_week', 'month', 'day', 'week_of_year'.

```
#month
df_raw['month_sin'] = df_raw['month'].apply(lambda x: np.sin(x * (2* np.pi/12) ))
df_raw['month_cos'] = df_raw['month'].apply(lambda x: np.cos(x * (2* np.pi/12) ))

#day
df_raw['day_sin'] = df_raw['day'].apply(lambda x: np.sin(x * (2* np.pi/30) ))
df_raw['day_cos'] = df_raw['day'].apply(lambda x: np.cos(x * (2* np.pi/30) ))

#week_of_year
df_raw['week_of_year_sin'] = df_raw['week_of_year'].apply(lambda x: np.sin(x * (2* np.pi/5
df_raw['week_of_year_cos'] = df_raw['week_of_year'].apply(lambda x: np.cos(x * (2* np.pi/5

#day_of_week
df_raw['day_of_week_sin'] = df_raw['day_of_week'].apply(lambda x: np.sin(x * (2* np.pi/7)
df_raw['day_of_week_cos'] = df_raw['day_of_week'].apply(lambda x: np.cos(x * (2* np.pi/7)


#Deleting those variables which were used to create other variables

cols_drop = ['month','week_of_year','day','day_of_week','promo_since','competition_since',
df_raw = df_raw.drop(cols_drop,axis = 1)


df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 844338 entries, 0 to 1017190
Data columns (total 32 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   store                         844338 non-null  int64
 1   date                          844338 non-null  datetime64[ns]
 2   sales                         844338 non-null  float64
 3   promo                         844338 non-null  int64
 4   school_holiday                844338 non-null  int64
 5   assortment                    844338 non-null  int64
 6   competition_distance          844338 non-null  float64
```

```
 7   competition_open_since_month  844338 non-null  int64
 8   competition_open_since_year   844338 non-null  int64
 9   promo2                        844338 non-null  int64
 10  promo2_since_week             844338 non-null  int64
 11  promo2_since_year             844338 non-null  int64
 12  is_promo                      844338 non-null  int64
 13  year                          844338 non-null  float64
 14  competition_time_month        844338 non-null  float64
 15  promo_time_week               844338 non-null  float64
 16  state_holiday_christmas       844338 non-null  uint8
 17  state_holiday_easter_holiday  844338 non-null  uint8
 18  state_holiday_public_holiday  844338 non-null  uint8
 19  state_holiday_regular_day     844338 non-null  uint8
 20  store_type_a                  844338 non-null  uint8
 21  store_type_b                  844338 non-null  uint8
 22  store_type_c                  844338 non-null  uint8
 23  store_type_d                  844338 non-null  uint8
 24  month_sin                     844338 non-null  float64
 25  month_cos                     844338 non-null  float64
 26  day_sin                       844338 non-null  float64
 27  day_cos                       844338 non-null  float64
 28  week_of_year_sin              844338 non-null  float64
 29  week_of_year_cos              844338 non-null  float64
 30  day_of_week_sin               844338 non-null  float64
 31  day_of_week_cos               844338 non-null  float64
dtypes: datetime64[ns](1), float64(13), int64(10), uint8(8)
memory usage: 199.7 MB
```

Feature Selection

Feature selection methods are intended to reduce the number of input variables to those that are believed to be most useful to a model in order to predict the target variable.

Techniques

- Based on the data, you may drop variables based on below reasons
  - If the variable have lots of missings (>25%)
  - if categorical Variable with lots categories(>20) - nominal variables
  - Variable with near zero variance (CV<0.05)
  - Unique variables/Keys/Names/Emaiid's/Phone number
  - Using Business Logic (By keeping in implementation perspective)
  - Variable Selection/Reduction using relationships
  - Supervised Learning (Based on Y & X Relationships)
  - Using statistical methods
  - Univariate Regression (F- Regression)
  - RFE (Recursive Feature Elimination)
  - SelectKBest
  - DT/RF
  - Regularization
  - WOE (relationship between Y & Log(odds) - Binary Classification

- Any technique (Based on relationship with in X's variables)
  - Correlation metrics
  - PCA/SVD
  - VIF (Variance Inflation factor >5)

```
#making a copy of our dataset
df1 = df_raw.copy()
```

TRAIN TEST SPLIT

```
#Grouping stores by date
df1[['date','store']].groupby('store').max().reset_index()['date'][0] - datetime.timedelta
```

```
Timestamp('2015-06-19 00:00:00')
```

```
#training dataset
X_train = df1[df1['date'] < '2015-06-19']
y_train = X_train['sales']

#test dataset
X_test = df1[df1['date'] > '2015-06-19']
y_test = X_test['sales']

print("Training min date: {}".format(X_train['date'].min()))
print("Training max date: {}".format(X_train['date'].max()))

print('\n')

print("Test Min date: {}".format(X_test['date'].min()))
print("Test Max Date: {}".format(X_test['date'].max()))
```

```
Training min date: 2013-01-01 00:00:00
Training max date: 2015-06-18 00:00:00


Test Min date: 2015-06-20 00:00:00
Test Max Date: 2015-07-31 00:00:00
```

Select K best

```
#### Select K best

SKB = SelectKBest(f_regression, k=15).fit(X_train[X_train.columns.difference(['date', 'sal

#SKB.get_support()
#imp_vars_SKB = list(X_train[X_train.columns.difference(['date', 'sale'])].columns[SKB.get
```

```
#imp_vars_SKB
imp_vars_SKB =  ['assortment',
 'competition_distance',
 'day_cos',
 'day_of_week_sin',
 'day_sin',
 'is_promo',
 'promo',
 'promo2',
 'promo2_since_week',
 'promo2_since_year',
 'promo_time_week',
 'school_holiday',
 'store_type_a',
 'store_type_b',
 'year']
```

RFE (Reverse Feature Elimination)

```
#### RFE

#Regressor = RandomForestRegressor(n_estimators=100, max_depth=10)
#rfe = RFE(Regressor,n_features_to_select=15,step=10,verbose=True)
#rfe = rfe.fit(X_train, y_train )


#imp_vars_RFE = list(X_train.columns[rfe.support_])


#Final List (RFE, SelectKbest)
#imp_features = list(set(imp_vars_SKB + imp_vars_RFE))
#imp_features

imp_features = ['state_holiday_regular_day',
 'day_of_week_cos',
 'competition_open_since_year',
 'competition_distance',
 'store',
 'competition_open_since_month',
 'store_type_d',
 'day_cos',
 'state_holiday_public_holiday',
 'promo2',
 'store_type_b',
 'is_promo',
 'month_cos',
 'promo2_since_year',
 'competition_time_month',
 'promo2_since_week',
 'assortment',
 'promo',
 'promo_time_week',
 'school_holiday',
 'year']
```

```
imp_features
```

```
['state_holiday_regular_day',
 'day_of_week_cos',
 'competition_open_since_year',
 'competition_distance',
 'store',
 'competition_open_since_month',
 'store_type_d',
 'day_cos',
 'state_holiday_public_holiday',
 'promo2',
 'store_type_b',
 'is_promo',
 'month_cos',
 'promo2_since_year',
 'competition_time_month',
 'promo2_since_week',
 'assortment',
 'promo',
 'promo_time_week',
 'school_holiday',
 'year']
```

```
imp_features_full= ['state_holiday_regular_day',
 'day_of_week_cos',
 'competition_open_since_year',
 'competition_distance',
 'store',
 'competition_open_since_month',
 'store_type_d',
 'day_cos',
 'state_holiday_public_holiday',
 'promo2',
 'store_type_b',
 'is_promo',
 'month_cos',
 'promo2_since_year',
 'competition_time_month',
 'promo2_since_week',
 'assortment',
 'promo',
 'promo_time_week',
 'school_holiday',
 'year','date','sales']
```

Base Model

MACHINE LEARNING MODELLING

In this section, we will finally build our predictive models. Therefore, we will use 5 machine learning algorithms, which will be:

- Average Model
- Linear Regression
- Lasso
- Random Forest Regressor
- XGBoost Regressor For each algorithm, we will build what we call the cross-validation technique. Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train / test split.

For this, we will separate our predictor variables from our target variable and then separate them in training and testing.

```
x_train = X_train[imp_features]
x_test = X_test[imp_features]

# Time Series Data Preparation
x_training = X_train[imp_features_full]
```

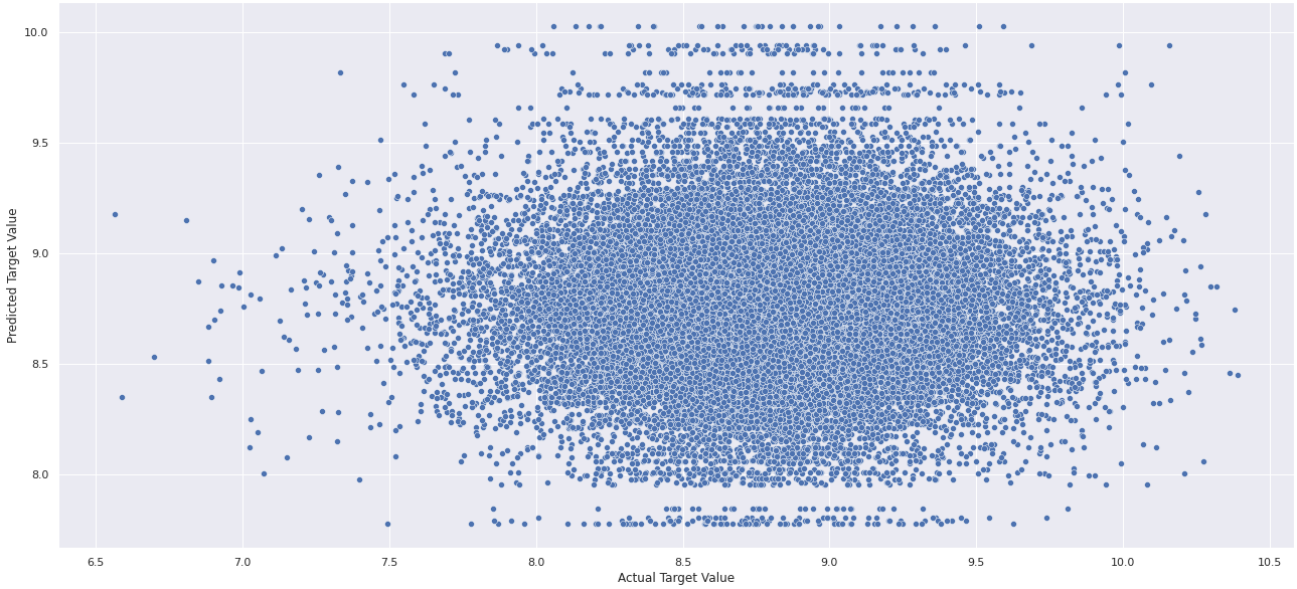
Average Model

```
aux1 = x_test.copy()
aux1['sales'] = y_test.copy()

#prediction
aux2 = aux1[['store','sales']].groupby("store").mean().reset_index().rename(columns = {'sa
aux1 = pd.merge(aux1,aux2, how = 'left', on = 'store')

yhat_baseline = aux1['predictions']

#performance
baseline_result = ml_error("Average Model", np.expm1(y_test), np.expm1(yhat_baseline))
baseline_result
```

	Model Name	MAE	MAPE	RMSE
0	Average Model	1366.128351	0.452087	1854.263316

```
#Plot y actual vs y predicted
plt.xlabel("Actual Target Value")
plt.ylabel("Predicted Target Value")
sns.scatterplot(x=y_test,y=yhat_baseline)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427e8a6a10>
```



Linear Regression Model

```
#model
lr = LinearRegression().fit(x_train,y_train)

#prediction
yhat_lr = lr.predict(x_test)

#performance
lr_result = ml_error('Linear Regression',np.expm1(y_test),np.expm1(yhat_lr))
lr_result
```

	Model Name	MAE	MAPE	RMSE
0	Linear Regression	1869.447873	0.301006	2630.525832

```
#Plot y actual vs y predicted
plt.xlabel("Actual Target Value")
```

```
plt.ylabel("Predicted Target Value")
sns.scatterplot(x=y_test,y=yhat_lr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427e169510>
```

Linear Regression Model - Cross Validation

```
lr_result_cv = cross_validation( x_training, 5, 'Linear Regression', lr, verbose=False )
lr_result_cv
```

	Model Name	MAE CV	MAPE CV	RMSE CV
0	Linear Regression	2060.86 +/- 310.79	0.29 +/- 0.01	2938.6 +/- 470.61

Linear Regression Regularized Model - Lasso

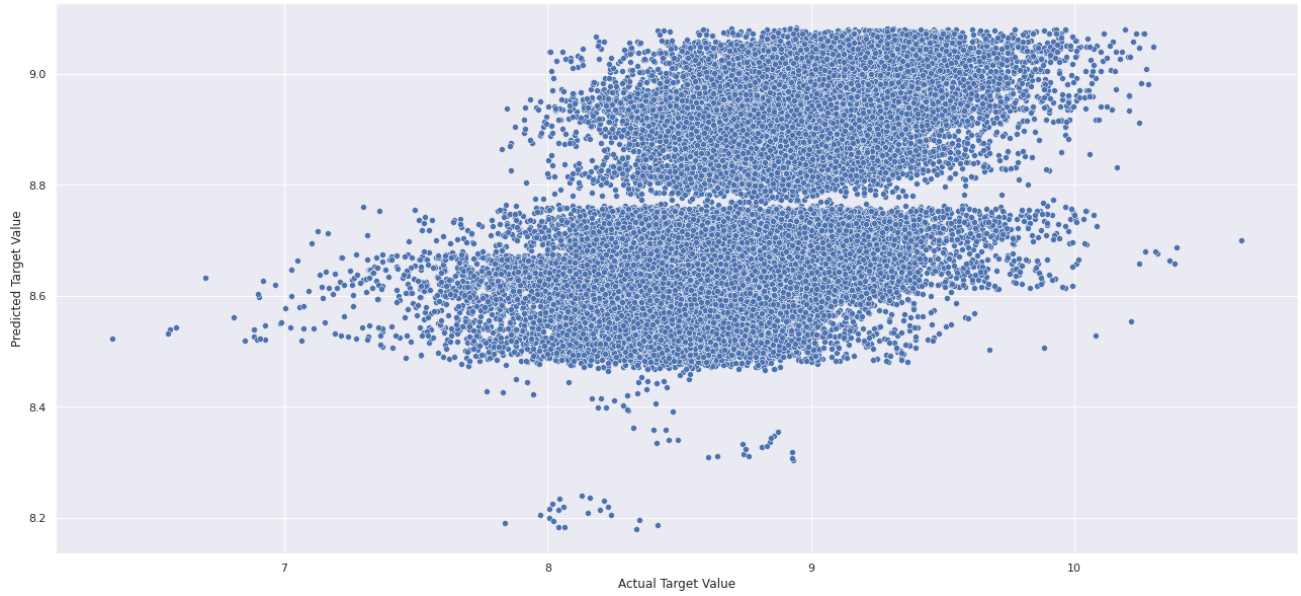
```
#model
lrr = Lasso(alpha = 0.01).fit(x_train,y_train)

#prediction
```

```
yhat_lrr = lrr.predict(x_test)

#performance
lrr_result = ml_error('Lasso ',np.expm1(y_test),np.expm1(yhat_lrr))
lrr_result
```

	Model Name	MAE	MAPE	RMSE
0	Lasso	1899.345657	0.292268	2754.328402

```
#Plot y actual vs y predicted
plt.xlabel("Actual Target Value")
plt.ylabel("Predicted Target Value")
sns.scatterplot(x=y_test,y=yhat_lrr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427e16a050>
```



Lasso - Cross Validation

```
lrr_result_cv = cross_validation( x_training, 5, 'Lasso', lrr, verbose=False )
```

```
lrr_result_cv
```

	Model Name	MAE CV	MAPE CV	RMSE CV
0	Lasso	2118.85 +/- 341.57	0.29 +/- 0.01	3062.94 +/- 503.47

```
#model
#rf = RandomForestRegressor(n_estimators = 100, n_jobs = -1, random_state = 42).fit(x_trai
#yhat_rf = rf.predict(y_test).resha

#performance
#rf_result = ml_error('Random Forest Regressor',np.expm1(y_test),np.expm1(yhat_rf))
#rf_result
```

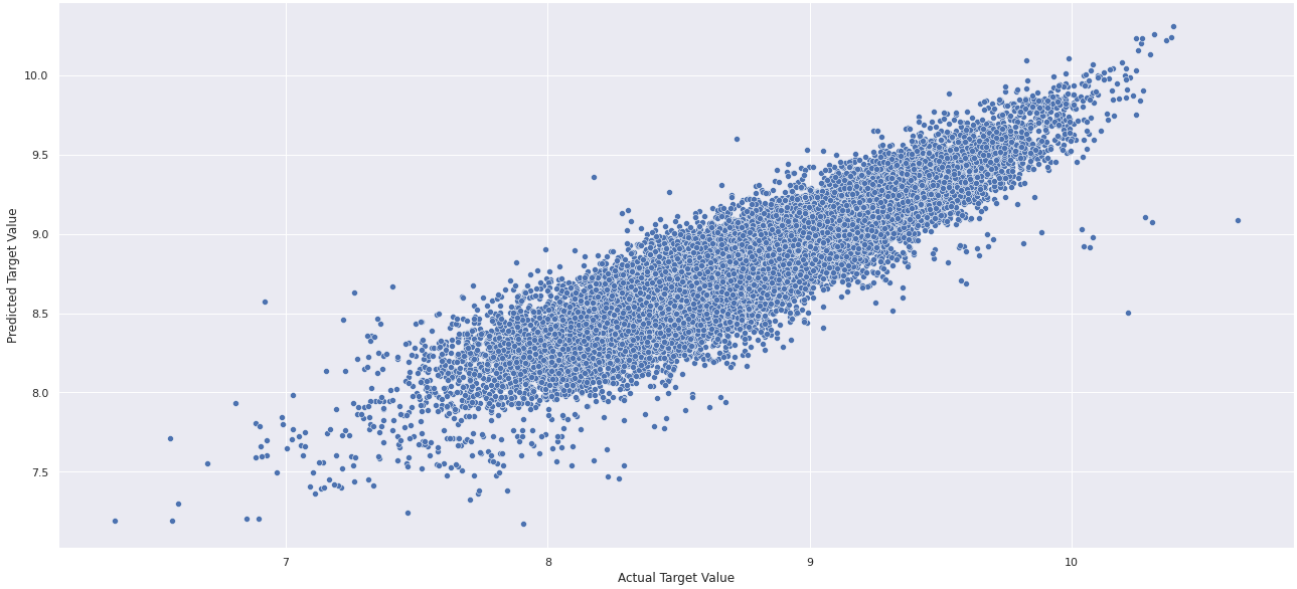
XGBoost Regressor

```
# model
model_xgb = xgb.XGBRegressor( objective='reg:squarederror',
                              n_estimators=100,
                              eta=0.01,
                              max_depth=10,
                              subsample=0.7,
                              colsample_bytee=0.9 ).fit( x_train, y_train )

# prediction
yhat_xgb = model_xgb.predict( x_test )

# performance
xgb_result = ml_error( 'XGBoost Regressor', np.expm1( y_test ), np.expm1( yhat_xgb ) )
xgb_result
```

	Model Name	MAE	MAPE	RMSE
0	XGBoost Regressor	956.737556	0.152953	1329.954186

```
#Plot y actual vs y predicted
plt.xlabel("Actual Target Value")
plt.ylabel("Predicted Target Value")
sns.scatterplot(x=y_test,y=yhat_xgb)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427e1f83d0>
```



XGBoost Regressor - Cross Validation

```
xgb_result_cv = cross_validation( x_training, 5, 'XGBoost Regressor', model_xgb, verbose=T
xgb_result_cv
```

```
KFold Number: 5

KFold Number: 4

KFold Number: 3

KFold Number: 2

KFold Number: 1
```

	Model Name	MAE CV	MAPE CV	RMSE CV
0	XGBoost Regressor	1088.43 +/- 149.04	0.15 +/- 0.01	1563.87 +/- 240.39

Compare Model Performance

Single Performance

```
modelling_result = pd.concat([baseline_result, lr_result, lrr_result,xgb_result])
modelling_result.sort_values('RMSE')
```

	Model Name	MAE	MAPE	RMSE
0	XGBoost Regressor	956.737556	0.152953	1329.954186
0	Average Model	1366.128351	0.452087	1854.263316
0	Linear Regression	1869.447873	0.301006	2630.525832
0	Lasso	1899.345657	0.292268	2754.328402

Real Performance - Cross Validation

```
modelling_result_cv = pd.concat( [lr_result_cv, lrr_result_cv, xgb_result_cv] )
modelling_result_cv.sort_values("RMSE CV")
```

	Model Name	MAE CV	MAPE CV	RMSE CV
0	XGBoost Regressor	1088.43 +/- 149.04	0.15 +/- 0.01	1563.87 +/- 240.39
0	Linear Regression	2060.86 +/- 310.79	0.29 +/- 0.01	2938.6 +/- 470.61
0	Lasso	2118.85 +/- 341.57	0.29 +/- 0.01	3062.94 +/- 503.47

As we can see in the table above, our model that best suited our problem was the XGBoost Regressor. We can interpret the metrics as follows:

- MAE = On average, our model is wrong about 1039 with a standard deviation of 178.
- MAPE = On average, our model has an error of 15% and a standard deviation of 2%. That is, it can make mistakes between 13% and 17%.
- RMSE = It will be useful to guide the performance of the model, that is, the RMSE will be more rigid with errors.

HYPERPARAMETER FINE TUNING

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

Approaches

- GridSearch -> Is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.

- RandomSearch -> The algorithm described herein is a type of local random search, where every iteration is dependent on the prior iteration's candidate solution.
- Bayesian optimization -> It builds a probabilistic model of mapping the function of hyperparameter values to the evaluated objective in a validation set

```
#param = {
#    'n_estimators': [1500, 1700, 2500, 3000, 3500],
#    'eta': [0.01, 0.03],
#    'max_depth': [3, 5, 9],
#    'subsample': [0.1, 0.5, 0.7],
#    'colsample_bytree': [0.3, 0.7, 0.9],
#    'min_child_weight': [3, 8, 15]
#        }
#
#MAX_EVAL = 5


#final_result = pd.DataFrame()
#
#for i in range( MAX_EVAL ):
#    # choose values for parameters randomly
#    hp = { k: random.sample( v, 1 )[0] for k, v in param.items() }
#    print( hp )
#
#    # model
#    model_xgb = xgb.XGBRegressor( objective='reg:squarederror',
#                                  n_estimators=hp['n_estimators'],
#                                  eta=hp['eta'],

#                                  max_depth=hp['max_depth'],
#                                  subsample=hp['subsample'],
#                                  colsample_bytee=hp['colsample_bytree'],
#                                  min_child_weight=hp['min_child_weight'] )
#
#    # performance
#    result = cross_validation( x_training, 5, 'XGBoost Regressor', model_xgb, verbose=Tru
#    final_result = pd.concat( [final_result, result] )
#
#final_result


#final_result


#final_resul = param_tuned
param_tuned = {
    'n_estimators': 3000,
    'eta': 0.03,
    'max_depth': 5,
    'subsample': 0.7,
    'colsample_bytree': 0.7,
    'min_child_weight': 3
        }
```

```
# model
model_xgb_tuned = xgb.XGBRegressor( objective='reg:squarederror',
                                    n_estimators=param_tuned['n_estimators'],
                                    eta=param_tuned['eta'],
                                    max_depth=param_tuned['max_depth'],
                                    subsample=param_tuned['subsample'],
                                    colsample_bytee=param_tuned['colsample_bytree'],
                                    min_child_weight=param_tuned['min_child_weight'] ).fit

# prediction
yhat_xgb_tuned = model_xgb_tuned.predict( x_test )

# performance
xgb_result_tuned = ml_error( 'XGBoost Regressor', np.expm1( y_test ), np.expm1( yhat_xgb_t
xgb_result_tuned
```

	Model Name	MAE	MAPE	RMSE
0	XGBoost Regressor	764.051411	0.118522	1074.639594

```
#Plot y actual vs y predicted
plt.xlabel("Actual Target Value")
plt.ylabel("Predicted Target Value")
sns.scatterplot(x=y_test,y=yhat_xgb)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427ee5d690>
```

ERROR INTERPRETATION

As previously mentioned in the machine learning stage, we will explain each metric to better understand business performance and model performance.

- **MAE**
  - It assigns equal weight to all errors.
  - Robust in the presence of Outliers.
  - Easy understanding by the business team.
- **MAPE**
  - Shows how far the prediction is from the actual value, on average, as a percentage.
  - Widely used to report results.
  - It cannot be used if the response variable contains zero. If you have to predict zero.
- **RMSE**
  - It gives a lot of weight to big mistakes.
  - Sensitive in the presence of outliers.
  - Ideal for measuring the performance of the machine learning model.
  - As we have applied variable transformation, we need to transform them back to their original scale.

Business Performance

```
df9 = X_test[ imp_features_full ]

# rescale
df9['sales'] = np.expm1( df9['sales'] )
df9['predictions'] = np.expm1( yhat_xgb_tuned )


#sum of predictions
df91 = df9[['store','predictions']].groupby('store').sum().reset_index()

#MAE e MAPE
df9_aux1 = df9[['store','sales','predictions']].groupby('store').apply(lambda x: mean_abso
df9_aux2 = df9[['store','sales','predictions']].groupby("store").apply(lambda x: mean_abso




#Merge
df9_aux3 = pd.merge(df9_aux1, df9_aux2, how = 'inner', on = 'store')
```

```
df92 = pd.merge(df91, df9_aux3, how = 'inner', on = 'store')


#scenarios

df92['worst_scenario'] = df92['predictions'] - df92['MAE']
df92['best_scenario'] = df92['predictions'] + df92['MAE']

#order columns
df92 = df92[['store','predictions','worst_scenario','best_scenario','MAE','MAPE']]


df92.sort_values("MAPE", ascending = False).head()
```

	store	predictions	worst_scenario	best_scenario	MAE	MAPE
291	292	102227.203125	98494.458876	105959.947374	3732.744249	0.675755
908	909	234252.187500	226270.662797	242233.712203	7981.524703	0.556086
875	876	193750.890625	189266.034779	198235.746471	4484.855846	0.400588
721	722	362442.156250	359937.901489	364946.411011	2504.254761	0.337728
482	483	182041.953125	181057.734056	183026.172194	984.219069	0.312803

As observed in the results, we need to report to the business team, that there are stores that are more difficult to make the predictions.

```
plt.figure(figsize = (15,7))
sns.scatterplot(x = 'store',y = 'MAPE', data = df92)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427edd62d0>
```


Total Performance

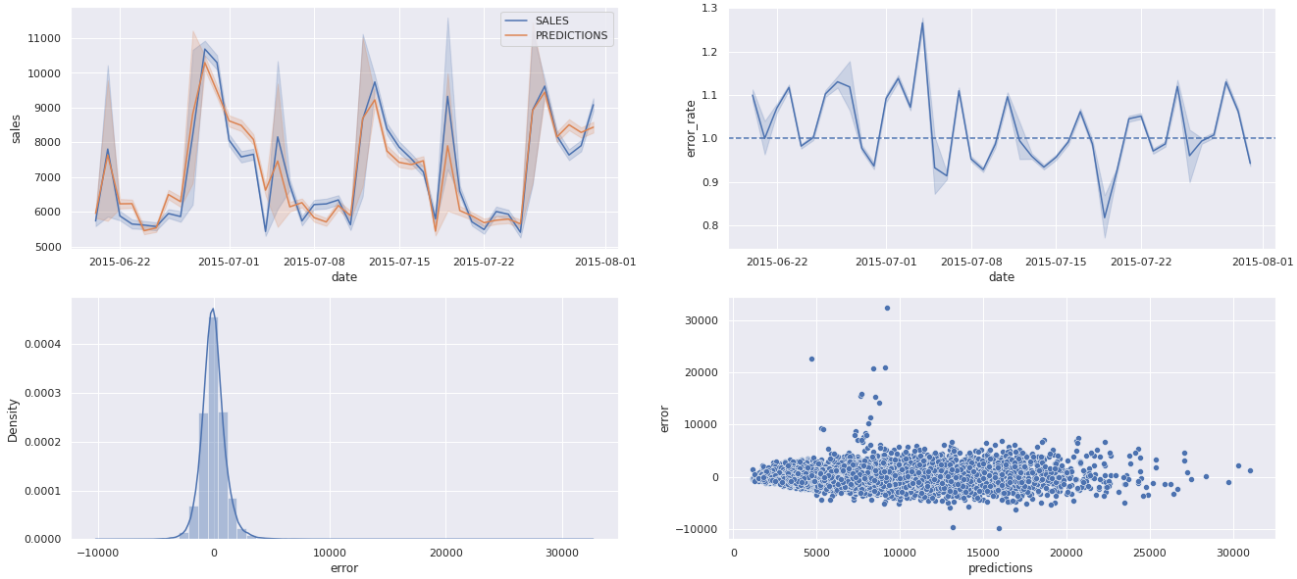
```
df93 = df92[['predictions','worst_scenario','best_scenario']].apply(lambda x: np.sum(x), a

df93['Values'] = df93['Values'].map("R${:,.2f}".format)
df93
```

	Scenario	Values
0	predictions	R$282,351,424.00
1	worst_scenario	R$281,494,899.38
2	best_scenario	R$283,207,996.42

Machine Learning Performance

```
df9['error'] = df9['sales'] - df9['predictions']
df9['error_rate'] = df9['predictions'] / df9['sales']


plt.subplot( 2, 2, 1 )
sns.lineplot( x='date', y='sales', data=df9, label='SALES' )
sns.lineplot( x='date', y='predictions', data=df9, label='PREDICTIONS' )

plt.subplot( 2, 2, 2 )
sns.lineplot( x='date', y='error_rate', data=df9 )
plt.axhline( 1, linestyle='--')

plt.subplot( 2, 2, 3 )
sns.distplot( df9['error'] )

plt.subplot( 2, 2, 4 )
sns.scatterplot( df9['predictions'], df9['error'] )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f427f02c050>
```



Model Deployment

Colab paid products - Cancel contracts here

✓ 0s completed at 21:29