# 03_dsemproducao_rossmann_store_sales_prediction

September 2, 2022

[1]:
```
# from IPython.display import display, HTML
# display(HTML("<style>.container { width:75% !important; }</style>"))
```

## 1 IMPORTS

### 1.1 HELPER FUNCTIONS

[1]:
```
import math
import inflection
import datetime
import random
import warnings
import pickle

import numpy   as np
import pandas  as pd
import seaborn as sns
import xgboost as xgb


from flask                   import Flask, request, Response
from scipy                   import stats
from boruta                  import BorutaPy
from tabulate                import tabulate
from matplotlib              import gridspec
from matplotlib              import pyplot as plt
from IPython.display         import Image
from IPython.core.display    import HTML
from sklearn.metrics         import mean_absolute_error,␣
 ↪mean_absolute_percentage_error, mean_squared_error
from sklearn.ensemble        import RandomForestRegressor
from sklearn.linear_model    import LinearRegression, Lasso
from sklearn.preprocessing   import LabelEncoder
from sklearn.preprocessing   import RobustScaler, MinMaxScaler
```

[2]:
```
warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.filterwarnings('ignore')
```

```
[3]: 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*7)

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

        # 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) +␣
 ↪' +/- ' + np.round(np.std(mae_list), 2).astype(str),
                         'MAPE CV': np.round(np.mean(mape_list), 2).astype(str)␣
 ↪+ ' +/- ' + np.round(np.std(mape_list), 2).astype(str),
```

```
                         'RMSE CV': np.round(np.mean(rmse_list), 2).astype(str)␣
↪+ ' +/- ' + np.round(np.std(rmse_list), 2).astype(str)}, index=[0])


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


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 cramer_v (x, y):
    cm = pd.crosstab(x, y).values
    n = cm.sum()
    r, k = cm.shape

    chi2 = stats.chi2_contingency(cm)[0]
    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((chi2corr/n)/(min(kcorr-1, rcorr-1)))


def jupyter_settings():
    %matplotlib notebook
    %matplotlib inline

    plt.style.use( 'bmh' )
    plt.rcParams['figure.figsize'] = [25, 12]
    plt.rcParams['font.size'] = 24
    display( HTML( '<style>.container { width:75% !important; }</style>') )

    pd.options.display.max_columns = None
    pd.options.display.max_rows = None
```

```
    pd.set_option( 'display.expand_frame_repr', False )

    sns.set()
```

[4]:
```
jupyter_settings()
```

```
<IPython.core.display.HTML object>
```

### 1.2 LOADING DATA

[6]:
```
df_sales_raw = pd.read_csv('data/train.csv', low_memory=False)
df_store_raw = pd.read_csv('data/store.csv', low_memory=False)

# merge
df_raw = pd.merge(df_sales_raw, df_store_raw, how='left', on='Store')
```

## 2 DATA OVERVIEW

[7]:
```
df1 = df_raw.copy()
```

### 2.1 RENAME COLUMNS

[8]:
```
cols_old = ['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open',␣
 ↪'Promo', 'StateHoliday', 'SchoolHoliday',
            'StoreType', 'Assortment', 'CompetitionDistance',␣
 ↪'CompetitionOpenSinceMonth',
            'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWeek',␣
 ↪'Promo2SinceYear', 'PromoInterval']

snakecase = lambda x: inflection.underscore(x)

cols_new = list(map(snakecase, cols_old))

# rename

df1.columns = cols_new
```

### 2.2 DATA DIMENSION

[9]:
```
print('Number of rows: {}'.format(df1.shape[0]))
print('Number of columns: {}'.format(df1.shape[1]))
```

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

## 2.3 DATA TYPES

[10]:
```
df1['date'] = pd.to_datetime(df1['date'])
df1.dtypes
```

[10]:
```
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
```

## 2.4 CHECK NA

[11]:
```
df1.isna().sum()
```

[11]:
```
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              2642
competition_open_since_month    323348
competition_open_since_year     323348
promo2                               0
promo2_since_week               508031
promo2_since_year               508031
promo_interval                  508031
```

dtype: int64

## 2.5 FILLOUT NA

[12]:
```
# competition_distance
df1['competition_distance'] = df1['competition_distance'].apply(lambda x:␣
↪200000.0 if math.isnan(x) else x)

# competition_open_since_month
df1['competition_open_since_month'] = df1.apply(lambda x: x['date'].month if␣
↪math.isnan(x['competition_open_since_month']) else␣
↪x['competition_open_since_month'], axis=1)

# competition_open_since_year
df1['competition_open_since_year'] = df1.apply(lambda x: x['date'].year if math.
↪isnan(x['competition_open_since_year']) else␣
↪x['competition_open_since_year'], axis=1)

# promo2_since_week
df1['promo2_since_week'] = df1.apply(lambda x: x['date'].week if math.
↪isnan(x['promo2_since_week']) else x['promo2_since_week'], axis=1)

# promo2_since_year
df1['promo2_since_year'] = df1.apply(lambda x: x['date'].year if math.
↪isnan(x['promo2_since_year']) else x['promo2_since_year'], axis=1)

# promo_interval
month_map = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7:␣
↪'Jul', 8: 'Aug', 9: 'Sept', 10: 'Oct', 11: 'Nov', 12: 'Dec'}

df1['promo_interval'].fillna(0, inplace=True)

df1['month_map'] = df1['date'].dt.month.map(month_map)

df1['is_promo'] = df1[['promo_interval', 'month_map']].apply(lambda x: 0 if␣
↪x['promo_interval'] == 0 else 1 if x['month_map'] in x['promo_interval'].
↪split(',') else 0, axis=1)
```

## 2.6 CHANGE DATA TYPES

[13]:
```
df1['competition_open_since_month'] = df1['competition_open_since_month'].
↪astype(int)
df1['competition_open_since_year'] = df1['competition_open_since_year'].
↪astype(int)

df1['promo2_since_week'] = df1['promo2_since_week'].astype(int)
```

```
df1['promo2_since_year'] = df1['promo2_since_year'].astype(int)
```

## 2.7 DESCRIPTIVE STATISTICS

[14]:
```
num_attributes = df1.select_dtypes(include=['int64', 'float64'])
cat_attributes = df1.select_dtypes(exclude=['int64', 'float64',␣
↪'datetime64[ns]'])
```

### 2.7.1 NUMERICAL ATTRIBUTES

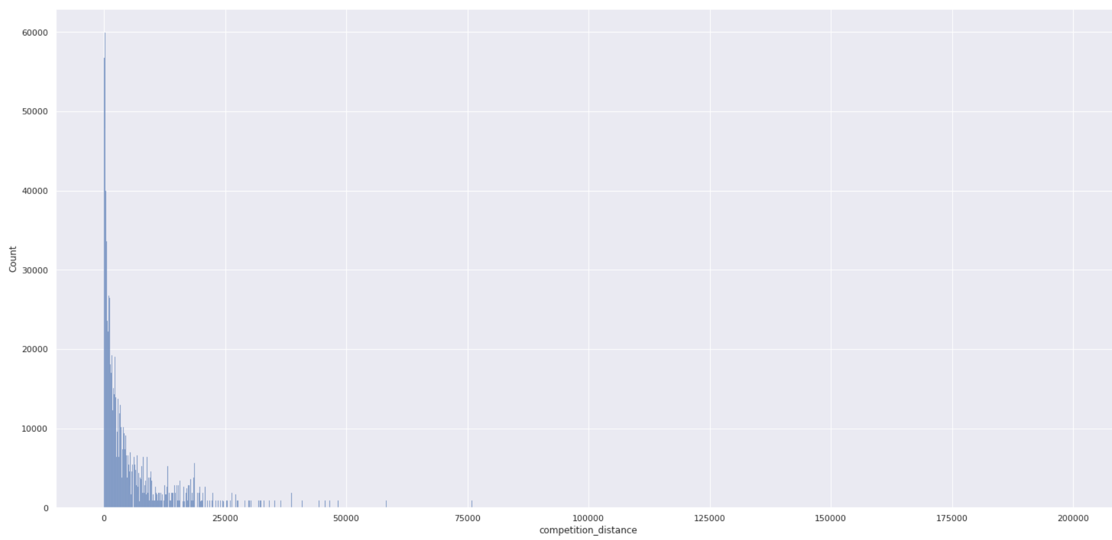
[15]:
```
# central tendency - mean, median
ct1 = pd.DataFrame(num_attributes.apply(np.mean)).T
ct2 = pd.DataFrame(num_attributes.apply(np.median)).T

# dispersion - std, min, max, range, skew, kurtosis
d1 = pd.DataFrame(num_attributes.apply(np.std)).T
d2 = pd.DataFrame(num_attributes.apply(min)).T
d3 = pd.DataFrame(num_attributes.apply(max)).T
d4 = pd.DataFrame(num_attributes.apply(lambda x: x.max() - x.min())).T
d5 = pd.DataFrame(num_attributes.apply(lambda x: x.skew())).T
d6 = pd.DataFrame(num_attributes.apply(lambda x: x.kurtosis())).T

# concatenate
m = pd.concat([d2, d3, d4, ct1, ct2, d1, d5, d6]).T.reset_index()
m.columns = ['attributes', 'min', 'max', 'range', 'mean', 'median',␣
↪'std','skew', 'kurtosis']
m
```

[15]:
```
                        attributes     min       max     range         mean  \
median           std        skew    kurtosis
0                            store     1.0    1115.0    1114.0   558.429727
558.0    321.908493  -0.000955   -1.200524
1                      day_of_week     1.0       7.0       6.0     3.998341
4.0      1.997390   0.001593   -1.246873
2                            sales     0.0   41551.0   41551.0  5773.818972
5744.0   3849.924283    0.641460    1.778375
3                        customers     0.0    7388.0    7388.0   633.145946
609.0    464.411506    1.598650    7.091773
4                             open     0.0       1.0       1.0     0.830107
1.0      0.375539   -1.758045    1.090723
5                            promo     0.0       1.0       1.0     0.381515
0.0      0.485758    0.487838   -1.762018
6                   school_holiday     0.0       1.0       1.0     0.178647
0.0      0.383056    1.677842    0.815154
7             competition_distance    20.0  200000.0  199980.0  5935.442677
2330.0  12547.646829   10.242344  147.789712
```

```
8   competition_open_since_month     1.0      12.0      11.0     6.786849
7.0      3.311085  -0.042076   -1.232607
9    competition_open_since_year  1900.0    2015.0     115.0  2010.324840
2012.0      5.515591  -7.235657  124.071304
10                        promo2     0.0       1.0       1.0     0.500564
1.0      0.500000  -0.002255   -1.999999
11             promo2_since_week     1.0      52.0      51.0    23.619033
22.0     14.310057   0.178723   -1.184046
12             promo2_since_year  2009.0    2015.0       6.0  2012.793297
2013.0      1.662657  -0.784436   -0.210075
13                      is_promo     0.0       1.0       1.0     0.171835
0.0      0.377237   1.739838    1.027039
```

[16]:
```python
sns.histplot(df1['competition_distance'])
```

[16]: <AxesSubplot:xlabel='competition_distance', ylabel='Count'>



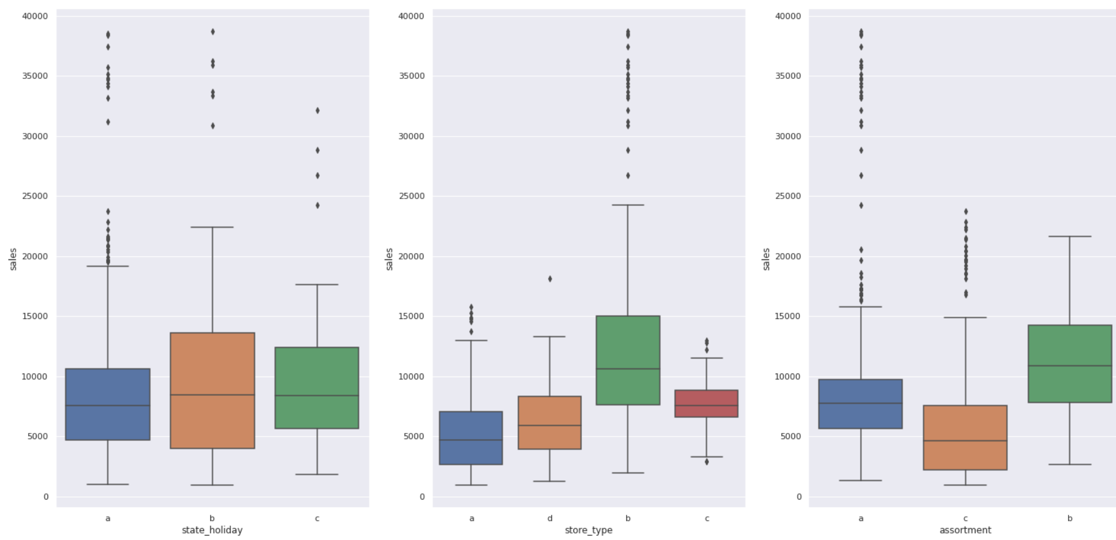

### 2.7.2 CATEGORICAL ATTRIBUTES

[17]:
```python
cat_attributes.apply(lambda x: x.unique().shape[0])
```

[17]:
```
state_holiday      4
store_type         4
assortment         3
promo_interval     4
month_map         12
dtype: int64
```

[18]:
```
aux1 = df1[(df1['state_holiday'] != '0') & (df1['sales'] > 0)]
plt.subplot(1, 3, 1)
sns.boxplot(x='state_holiday', y='sales', data=aux1)

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

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

[18]: <AxesSubplot:xlabel='assortment', ylabel='sales'>

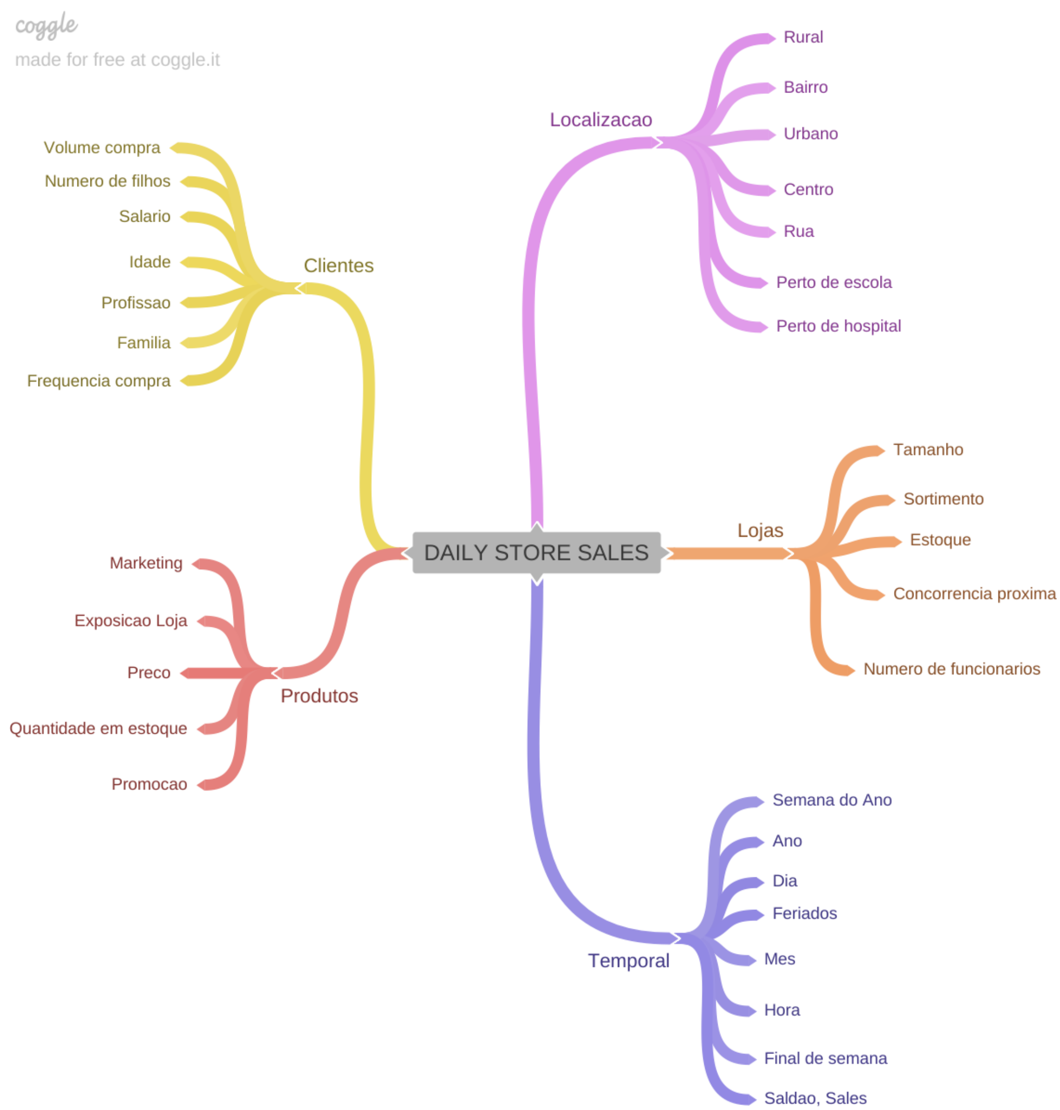



# 3 FEATURE ENGINEERING

## 3.1 MINDMAP HYPOTHESIS

[19]:
```
Image('img/mindmaphipotesis.png', width='600', height='400')
```

[19]:




## 3.2 HYPOTESIS DEVELOPMENT

[20]:
```
df2 = df1.copy()
```

### 3.2.1 STORE HYPOTHESIS

**1.** Lojas com número maior de funcionários deveriam vender mais.

**2.** Lojas com maior capacidade de estoque deveriam vender mais.

**3.** Lojas com maior porte deveriam vender mais.

**4.** Lojas com maior sortimentos deveriam vender mais.

**5.** Lojas com competidores mais próximos deveriam vender menos.

**6.** Lojas com competidores à mais tempo deveriam vendem mais.

#### 3.2.2 PRODUCT HYPOTHESIS

**1.** Lojas que investem mais em Marketing deveriam vender mais.

**2.** Lojas com maior exposição de produto deveriam vender mais.

**3.** Lojas com produtos com preço menor deveriam vender mais.

**5.** Lojas com promoções mais agressivas ( descontos maiores ), deveriam vender mais.

**6.** Lojas com promoções ativas por mais tempo deveriam vender mais.

**7.** Lojas com mais dias de promoção deveriam vender mais.

**8.** Lojas com mais promoções consecutivas deveriam vender mais.

#### 3.2.3 TIME HYPOTHESIS

**1.** Lojas abertas durante o feriado de Natal deveriam vender mais.

**2.** Lojas deveriam vender mais ao longo dos anos.

**3.** Lojas deveriam vender mais no segundo semestre do ano.

**4.** Lojas deveriam vender mais depois do dia 10 de cada mês.

**5.** Lojas deveriam vender menos aos finais de semana.

**6.** Lojas deveriam vender menos durante os feriados escolares.

### 3.3 HYPOTHESIS FINAL LIST

**1.** Lojas com maior sortimentos deveriam vender mais.

**2.** Lojas com competidores mais próximos deveriam vender menos.

**3.** Lojas com competidores à mais tempo deveriam vendem mais.

**4.** Lojas com promoções ativas por mais tempo deveriam vender mais.

**5.** Lojas com mais dias de promoção deveriam vender mais.

**6.** Lojas com mais promoções consecutivas deveriam vender mais.

**7.** Lojas abertas durante o feriado de Natal deveriam vender mais.

**8.** Lojas deveriam vender mais ao longo dos anos.

**9.** Lojas deveriam vender mais no segundo semestre do ano.

**10.** Lojas deveriam vender mais depois do dia 10 de cada mês.

**11.** Lojas deveriam vender menos aos finais de semana.

**12.** Lojas deveriam vender menos durante os feriados escolares.

### 3.4 FEATURE ENGINEERING

[21]:
```python
# year
df2['year'] = df2['date'].dt.year

# month
df2['month'] = df2['date'].dt.month

# day
df2['day'] = df2['date'].dt.day

# week of year
df2['week_of_year'] = df2['date'].dt.weekofyear

# year week
df2['year_week'] = df1['date'].dt.strftime('%Y-%W')

# competition since
df2['competition_since'] = df2.apply(lambda x: datetime.datetime(year=x['competition_open_since_year'], month=x['competition_open_since_month'],day=1), axis=1)
df2['competition_time_month'] =  ((df2['date'] - df2['competition_since'])/30).apply(lambda x: x.days).astype(int)

# promo since
df2['promo_since'] =  df2['promo2_since_year'].astype(str) + '-' + df2['promo2_since_week'].astype(str)
df2['promo_since'] = df2['promo_since'].apply(lambda x: datetime.datetime.strptime(x + '-1', '%Y-%W-%w') - datetime.timedelta(days=7))
df2['promo_time_week'] = ((df2['date'] - df2['promo_since'])/7).apply(lambda x: x.days).astype(int)

# assortment
df2['assortment'] = df2['assortment'].apply(lambda x: 'basic' if x == 'a' else 'extra' if x == 'b' else 'extended')

# state holiday
df2['state_holiday'] = df2['state_holiday'].apply(lambda x: 'public_holiday' if x == 'a' else 'easter_holiday' if x == 'b' else 'christmas' if x == 'c' else 'regular_day')
```

[22]:
```python
df2.sample(5).T
```

[22]:
```
                                                386248               365363
671113                  460309                 530898
store                                              326                 1060
669                     600                    944
```

```
day_of_week                                     4                    6
4                    4                    4
date                          2014-07-24 00:00:00  2014-08-16 00:00:00
2013-11-07 00:00:00  2014-05-15 00:00:00  2014-03-13 00:00:00
sales                                        4530                 5486
4794                 5196                 5562
customers                                     378                  669
409                  503                 1005
open                                            1                    1
1                    1                    1
promo                                           0                    0
1                    0                    0
state_holiday                         regular_day          regular_day
regular_day          regular_day          regular_day
school_holiday                                  0                    0
0                    0                    0
store_type                                      d                    a
d                    d                    c
assortment                                  basic             extended
basic             extended                basic
competition_distance                      10070.0               3430.0
17080.0              17340.0               1670.0
competition_open_since_month                    5                    8
7                    6                    7
competition_open_since_year                  2015                 2014
2012                 2010                 2015
promo2                                          1                    1
1                    1                    0
promo2_since_week                              31                   31
31                    9                   11
promo2_since_year                            2013                 2013
2013                 2011                 2014
promo_interval                    Feb,May,Aug,Nov      Feb,May,Aug,Nov
Jan,Apr,Jul,Oct      Feb,May,Aug,Nov                    0
month_map                                     Jul                  Aug
Nov                  May                  Mar
is_promo                                        0                    1
0                    1                    0
year                                         2014                 2014
2013                 2014                 2014
month                                           7                    8
11                    5                    3
day                                            24                   16
7                   15                   13
week_of_year                                   30                   33
45                   20                   11
year_week                                 2014-29              2014-32
```

```
2013-44                  2014-19                   2014-10
competition_since                  2015-05-01 00:00:00  2014-08-01 00:00:00
2012-07-01 00:00:00  2010-06-01 00:00:00  2015-07-01 00:00:00
competition_time_month                            -10                    0
16                   48                      -16
promo_since                        2013-07-29 00:00:00  2013-07-29 00:00:00
2013-07-29 00:00:00  2011-02-21 00:00:00  2014-03-10 00:00:00
promo_time_week                                     51                   54
14                  168                       0
```

# 4 FILTRAGEM DE VARIAVEIS

[23]:
```
df3 = df2.copy()
```

[24]:
```
df3.head()
```

[24]:
```
   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 month_map  is_promo  year
month  day  week_of_year year_week competition_since  competition_time_month
promo_since  promo_time_week
0      1            5 2015-07-31   5263        555      1      1   regular_day
1          c      basic                1270.0                             9
2008       0                 31                2015                0       Jul
0  2015      7   31            31   2015-30        2008-09-01
84  2015-07-27                 0
1      2            5 2015-07-31   6064        625      1      1   regular_day
1          a      basic                 570.0                            11
2007       1                 13                2010  Jan,Apr,Jul,Oct       Jul
1  2015      7   31            31   2015-30        2007-11-01
94  2010-03-22               279
2      3            5 2015-07-31   8314        821      1      1   regular_day
1          a      basic               14130.0                            12
2006       1                 14                2011  Jan,Apr,Jul,Oct       Jul
1  2015      7   31            31   2015-30        2006-12-01
105  2011-03-28              226
3      4            5 2015-07-31  13995       1498      1      1   regular_day
1          c   extended                 620.0                             9
2009       0                 31                2015                0       Jul
0  2015      7   31            31   2015-30        2009-09-01
71  2015-07-27                 0
4      5            5 2015-07-31   4822        559      1      1   regular_day
1          a      basic               29910.0                             4
2015       0                 31                2015                0       Jul
0  2015      7   31            31   2015-30        2015-04-01
```

```
4   2015-07-27                0
```

### 4.1 LINE FILTER

[25]:
```python
df3 = df3[(df3['open'] != 0) & (df3['sales'] > 0)]
```

### 4.2 COLUMNS FILTER

[26]:
```python
cols_drop = ['customers', 'open', 'promo_interval', 'month_map']
df3 = df3.drop(cols_drop, axis=1)
```

[27]:
```python
df3.columns
```

[27]:
```
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')
```

## 5 ANALISE EXPLORATORIA DOS DADOS

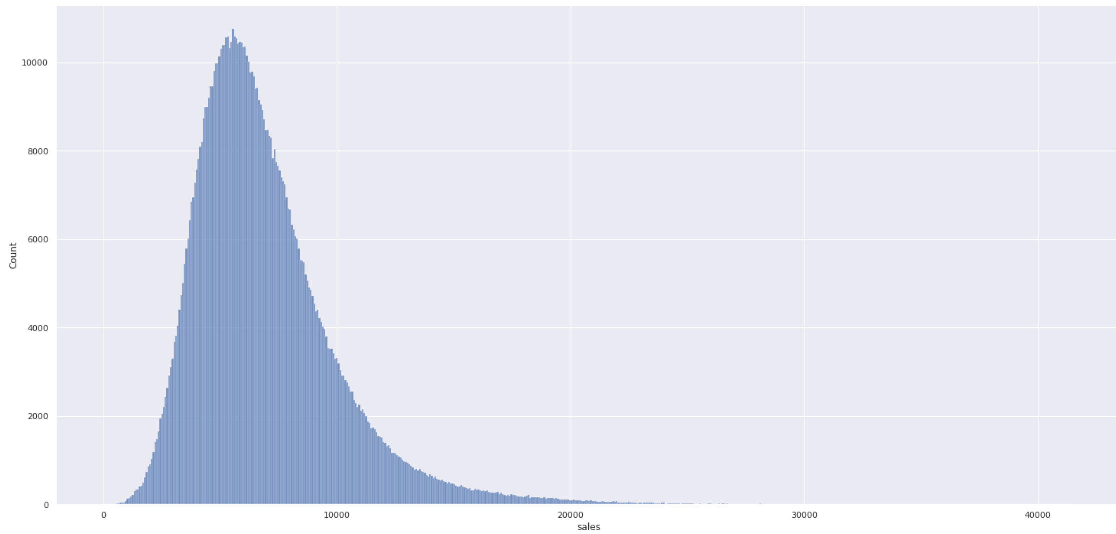
[28]:
```python
df4 = df3.copy()
```

### 5.1 ANALISE UNIVARIADA

#### 5.1.1 RESPONSE VARIABLE

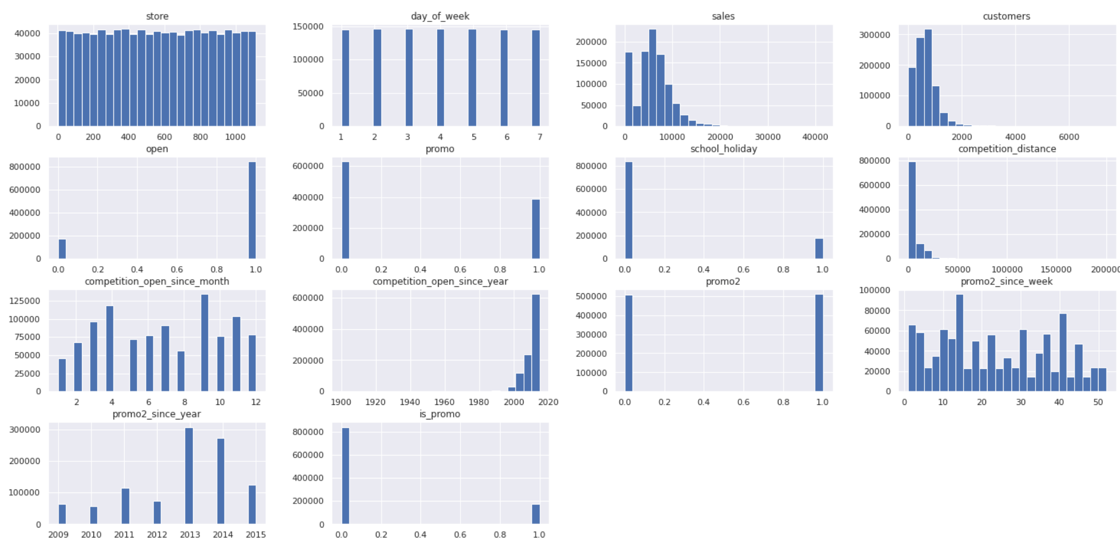
[29]:
```python
sns.histplot(df4['sales'], kde=False)
```

[29]:
```
<AxesSubplot:xlabel='sales', ylabel='Count'>
```




### 5.1.2 NUMERICAL VARIABLES

[30]:
```
num_attributes.hist(bins=25);
```

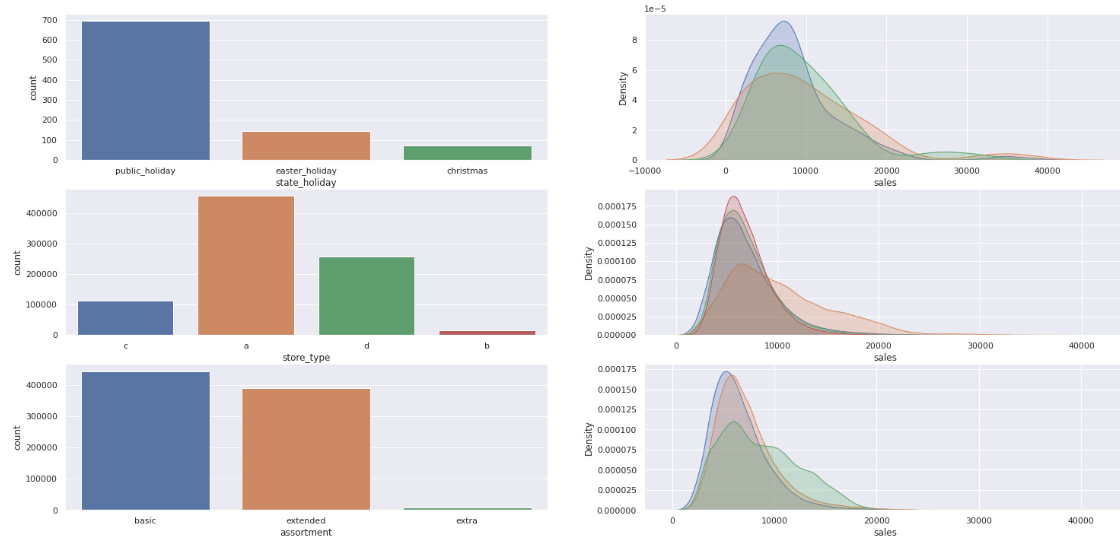



### 5.1.3 CATEGORICAL VARIABLES

[31]:
```
# state_holiday
plt.subplot(3, 2, 1)
a = df4[df4['state_holiday'] != 'regular_day']
sns.countplot(a['state_holiday'])
```

```
plt.subplot(3, 2, 2)
sns.kdeplot(df4[df4['state_holiday'] == 'public_holiday']['sales'],␣
 ↪label='public_holiday', shade=True)
sns.kdeplot(df4[df4['state_holiday'] == 'easter_holiday']['sales'],␣
 ↪label='easter_holiday', shade=True)
sns.kdeplot(df4[df4['state_holiday'] == 'christmas']['sales'],␣
 ↪label='christmas', shade=True)

# store_type
plt.subplot(3, 2, 3)
sns.countplot(df4['store_type'])

plt.subplot(3, 2, 4)
sns.kdeplot(df4[df4['store_type'] == 'a']['sales'], label='a', shade=True)
sns.kdeplot(df4[df4['store_type'] == 'b']['sales'], label='b', shade=True)
sns.kdeplot(df4[df4['store_type'] == 'c']['sales'], label='c', shade=True)
sns.kdeplot(df4[df4['store_type'] == 'd']['sales'], label='d', shade=True)


# assortment
plt.subplot(3, 2, 5)
sns.countplot(df4['assortment'])

plt.subplot(3, 2, 6)
sns.kdeplot(df4[df4['assortment'] == 'basic']['sales'], label='basic',␣
 ↪shade=True)
sns.kdeplot(df4[df4['assortment'] == 'extended']['sales'], label='extended',␣
 ↪shade=True)
sns.kdeplot(df4[df4['assortment'] == 'extra']['sales'], label='extra',␣
 ↪shade=True)
```

[31]: <AxesSubplot:xlabel='sales', ylabel='Density'>



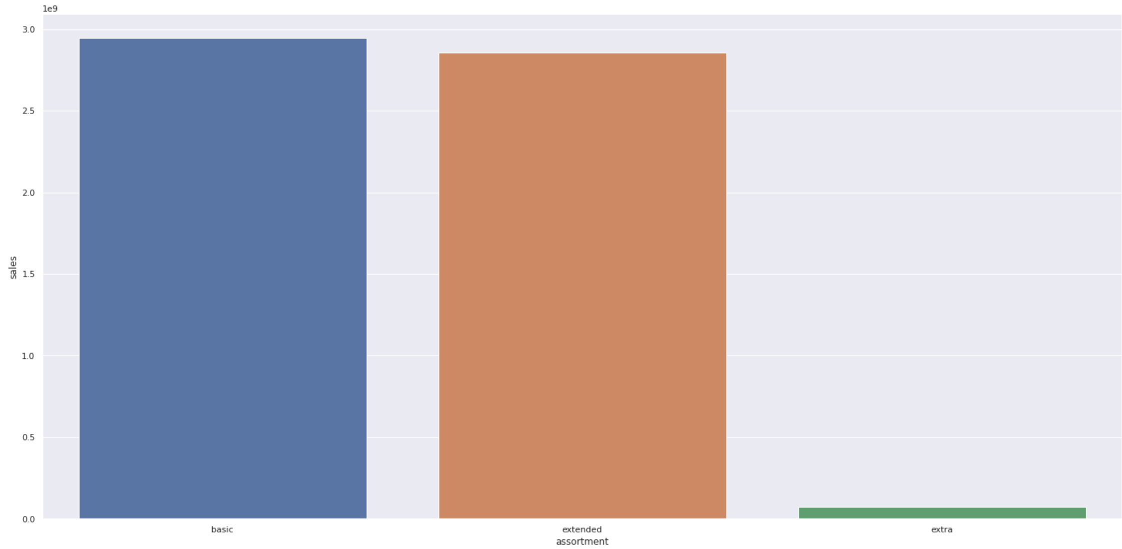

## 5.2 ANALISE BIVARIADA

### 5.2.1 H1 Lojas com maior sortimento deveriam vender mais.

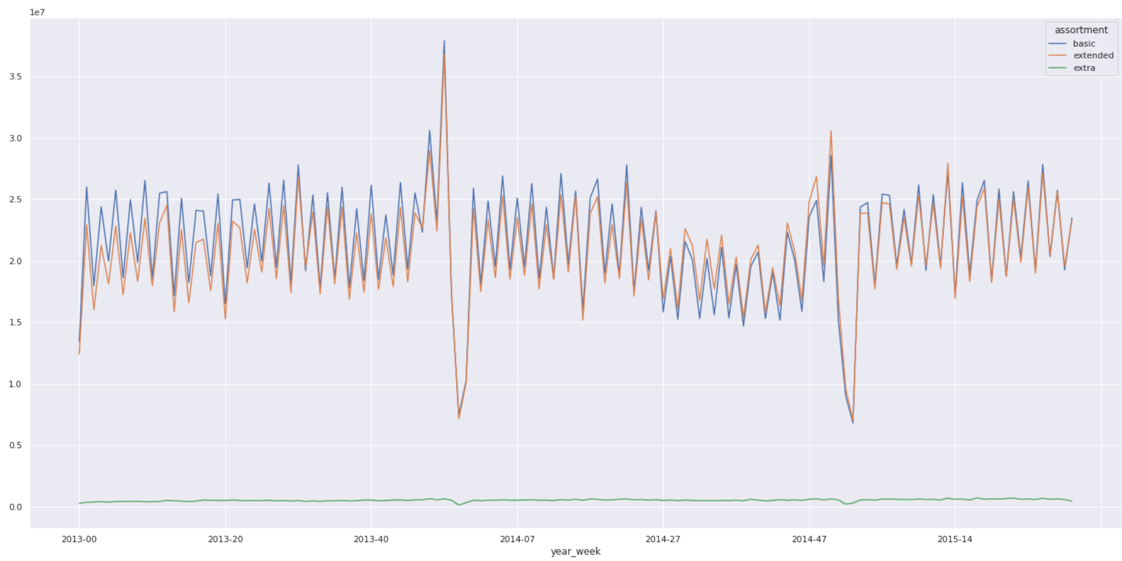
**FALSA** Lojas com MAIOR SORTIMENTO vendem MENOS

[32]:
```python
aux1 = df4[['assortment', 'sales']].groupby('assortment').sum().reset_index()
sns.barplot(x='assortment', y='sales', data=aux1);

aux2 = df4[['year_week', 'assortment', 'sales']].groupby(['year_week',␣
↪'assortment']).sum().reset_index()
aux2.pivot(index='year_week', columns='assortment', values='sales').plot()

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

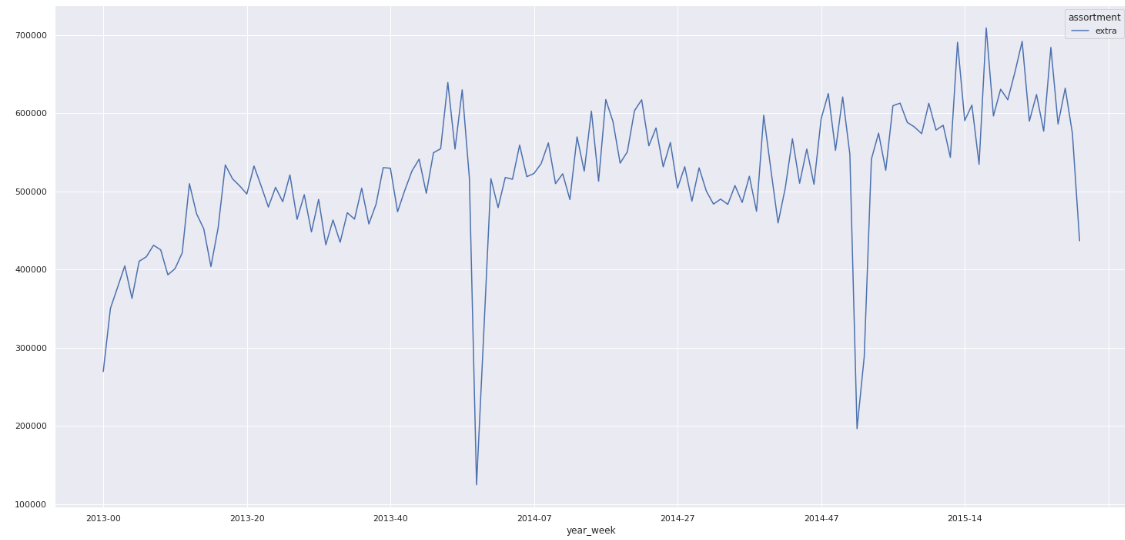
[32]:
```
<AxesSubplot:xlabel='year_week'>
```



1e9
3.0
2.5
2.0
1.5
1.0
0.5
0.0
sales
basic
extended
extra
assortment

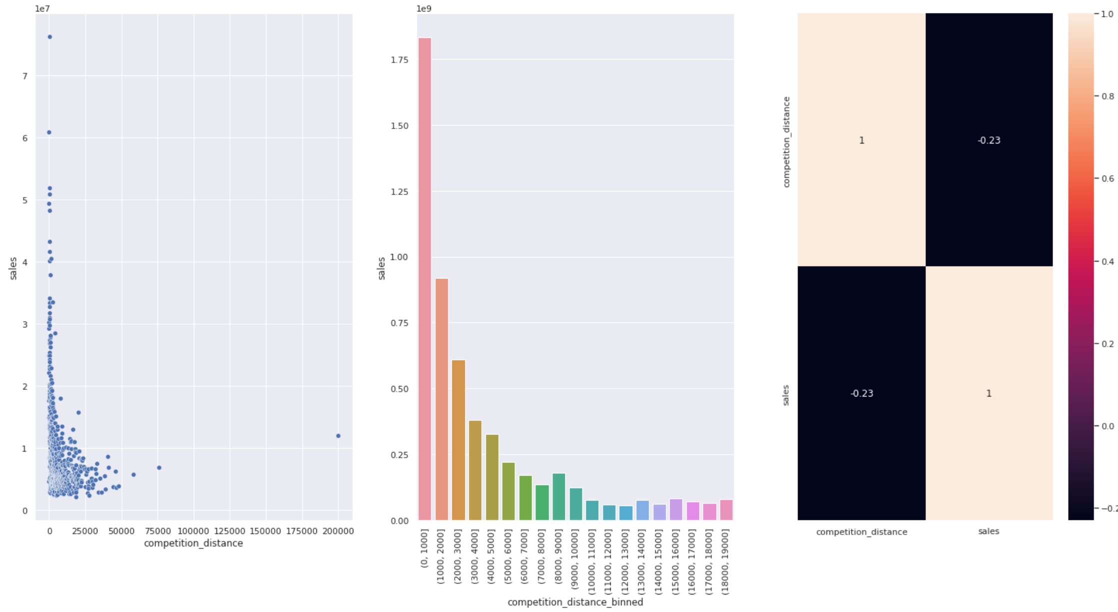
1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
assortment
basic
extended
extra
2013-00
2013-20
2013-40
2014-07
2014-27
2014-47
2015-14
year_week




### 5.2.2 H2 Lojas com competidores mais próximos deveriam vender menos.

**FALSA** Lojas com COMPETIDORES MAIS PROXIMOS vendem MAIS

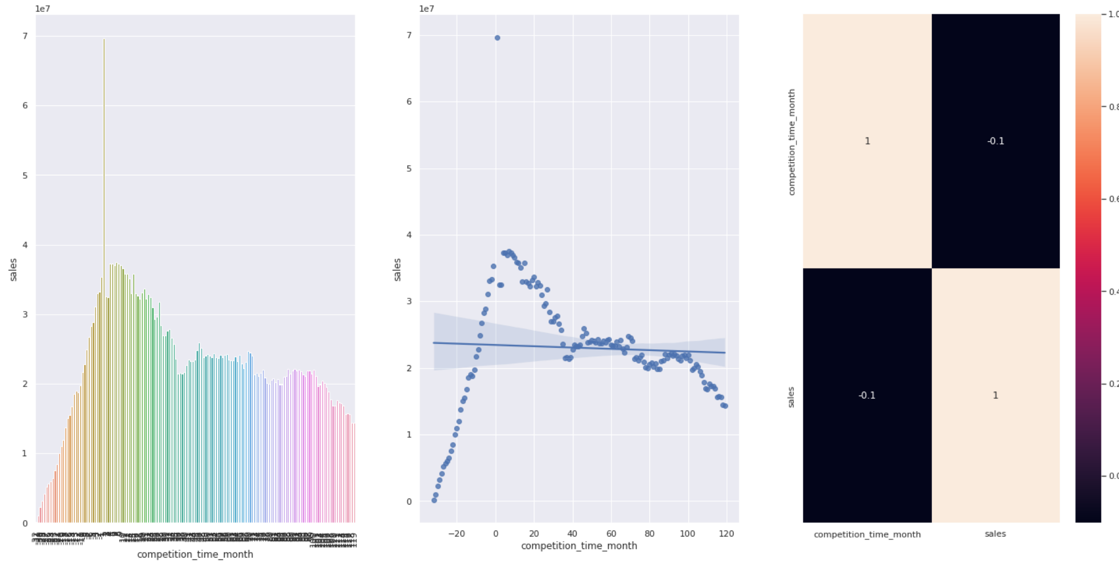
[33]:
```
aux1 = df4[['competition_distance', 'sales']].groupby('competition_distance').
    ↪sum().reset_index()

plt.subplot(1,3,1)
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').sum().reset_index()
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);
```




### 5.2.3 H3 Lojas com competidores à mais tempo deveriam vendem mais.

**FALSA** Lojas com COMPETIDORES A MAIS TEMPO vendem MENOS

[34]:
```
plt.subplot(1,3,1)
aux1 = df4[['competition_time_month', 'sales']].groupby('competition_time_month').sum().reset_index()
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);
```

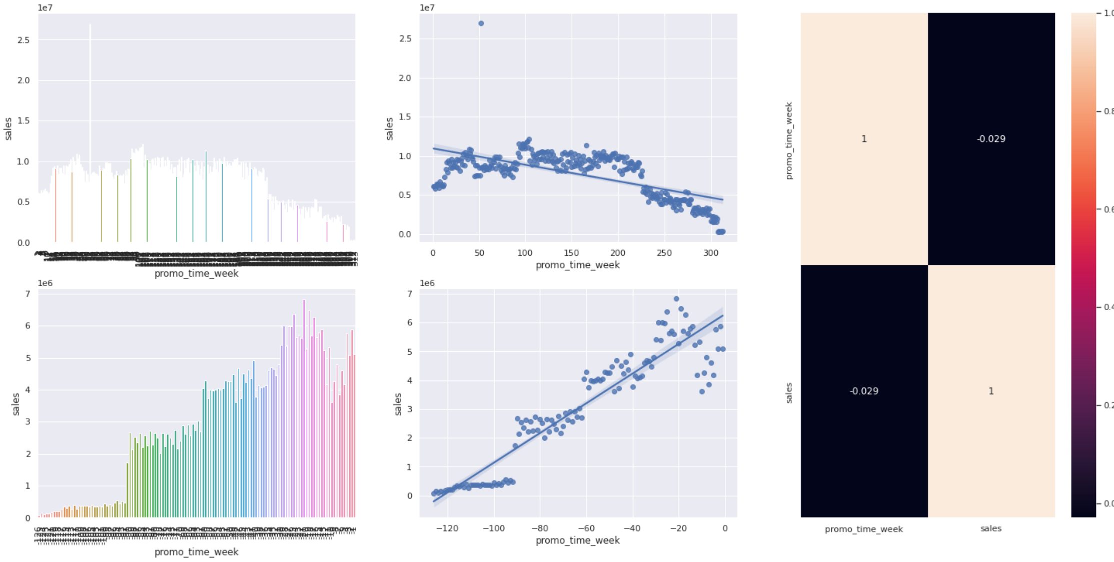



#### 5.2.4 H4 Lojas com promoções ativas por mais tempo deveriam vender mais.

**FALSA** Lojas com promocoes ativas por mais tempo vendem menos, depois de um certo periodo de promocao

[35]:
```
aux1 = df4[['promo_time_week', 'sales']].groupby('promo_time_week').sum().
 ↪reset_index()
#sns.barplot(x='promo_time_week', y='sales', data=aux1);

grid = gridspec.GridSpec(2, 3)

plt.subplot(grid[0,0])
aux2 = aux1[aux1['promo_time_week'] > 0] # extended promo
sns.barplot(x='promo_time_week', y='sales', data=aux2);
plt.xticks(rotation=90);

plt.subplot(grid[0,1])
sns.regplot(x='promo_time_week', y='sales', data=aux2);

plt.subplot(grid[1,0])
aux3 = aux1[aux1['promo_time_week'] < 0] # regular promo
sns.barplot(x='promo_time_week', y='sales', data=aux3);
plt.xticks(rotation=90);

plt.subplot(grid[1,1])
sns.regplot(x='promo_time_week', y='sales', data=aux3);

plt.subplot(grid[:,2])
```

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




### 5.2.5 H5 Lojas com mais dias de promoção deveriam vender mais.

### 5.2.6 H6 Lojas com mais promoções consecutivas deveriam vender mais.

**FALSA** Lojas com mais promocoes consecutivas vendem menos

[36]:
```
df4[['promo', 'promo2', 'sales']].groupby(['promo', 'promo2']).sum().
↪reset_index()
```

[36]:

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

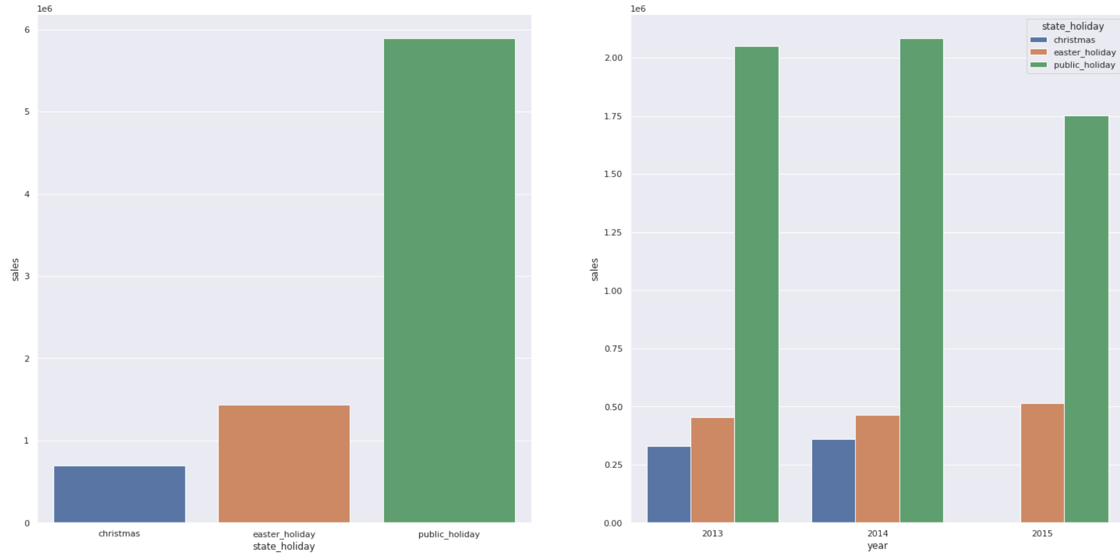
[37]:
```
aux1 = df4[(df4['promo'] == 1) & (df4['promo2'] == 1)][['year_week', 'sales']].
↪groupby('year_week').sum().reset_index()
ax = aux1.plot()

aux2 = df4[(df4['promo'] == 1) & (df4['promo2'] == 0)][['year_week', 'sales']].
↪groupby('year_week').sum().reset_index()
aux2.plot(ax=ax)

ax.legend(labels=['Tradicional & Extendida', 'Tradicional']);
```


### 5.2.7 H7 Lojas abertas durante o feriado de Natal deveriam vender mais.

**FALSA** Lojas abertas durante o feriado de natal vendem menos

[38]:
```
aux = df4[df4['state_holiday'] != 'regular_day']

plt.subplot(1,2,1)
aux1 = aux[['state_holiday', 'sales']].groupby('state_holiday').sum().
↪reset_index()
sns.barplot(x='state_holiday', y='sales', data=aux1);

plt.subplot(1,2,2)
aux2 = aux[['year', 'state_holiday', 'sales']].groupby(['year',␣
↪'state_holiday']).sum().reset_index()
sns.barplot(x='year', y='sales', hue='state_holiday', data=aux2)
```

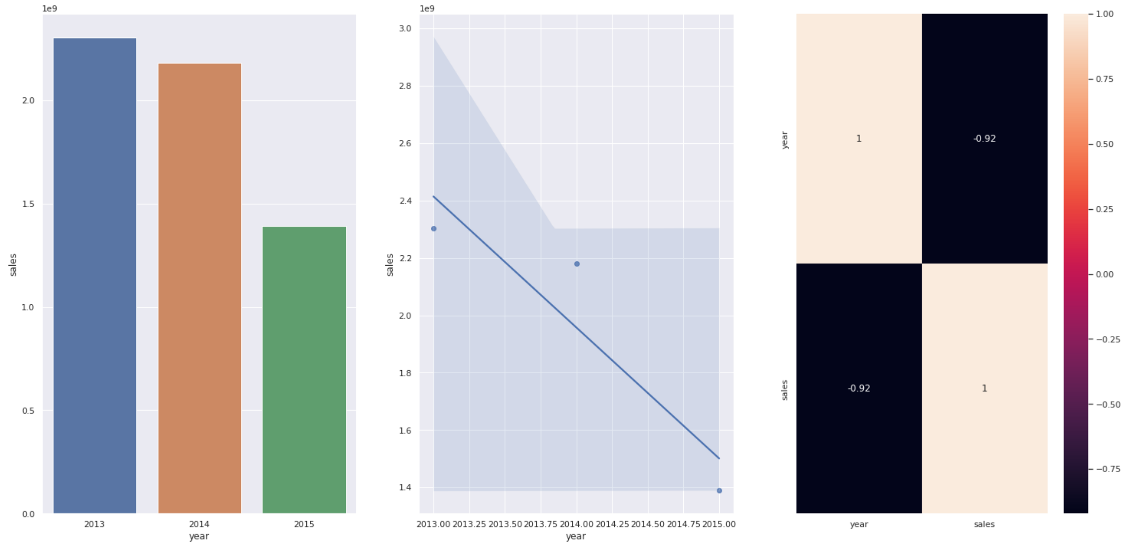
[38]: <AxesSubplot:xlabel='year', ylabel='sales'>




#### 5.2.8 H8 Lojas deveriam vender mais ao longo dos anos.

**FALSA** Lojas vendem menos ao longo dos anos (2015 em andamento)

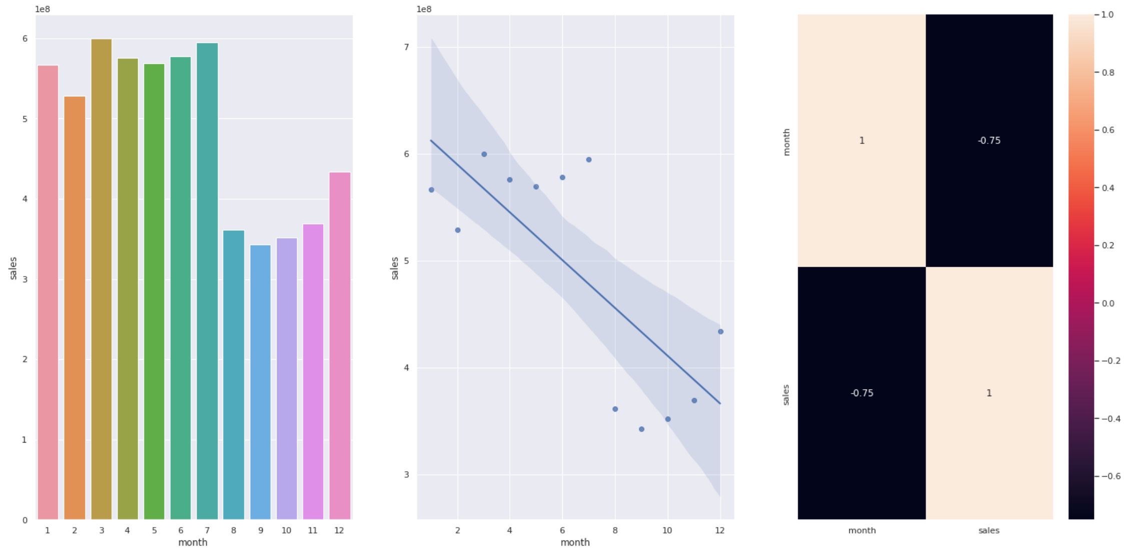
[39]:
```
aux1 = df4[['year', 'sales']].groupby('year').sum().reset_index()

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);
```




#### 5.2.9 H9 Lojas deveriam vender mais no segundo semestre do ano.

**FALSA** Lojas vendem menos no segundo semestre do ano

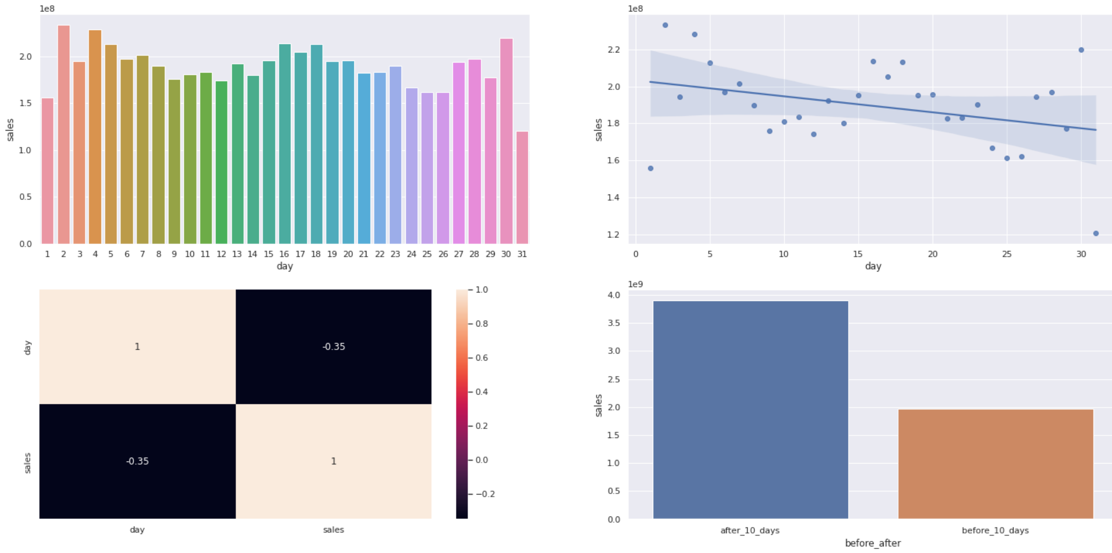
[40]:
```
aux1 = df4[['month', 'sales']].groupby('month').sum().reset_index()

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);
```




### 5.2.10 H10 Lojas deveriam vender mais depois do dia 10 de cada mês.

**VERDADEIRA** Lojas vendem mais depois do dia 10 de cada mes

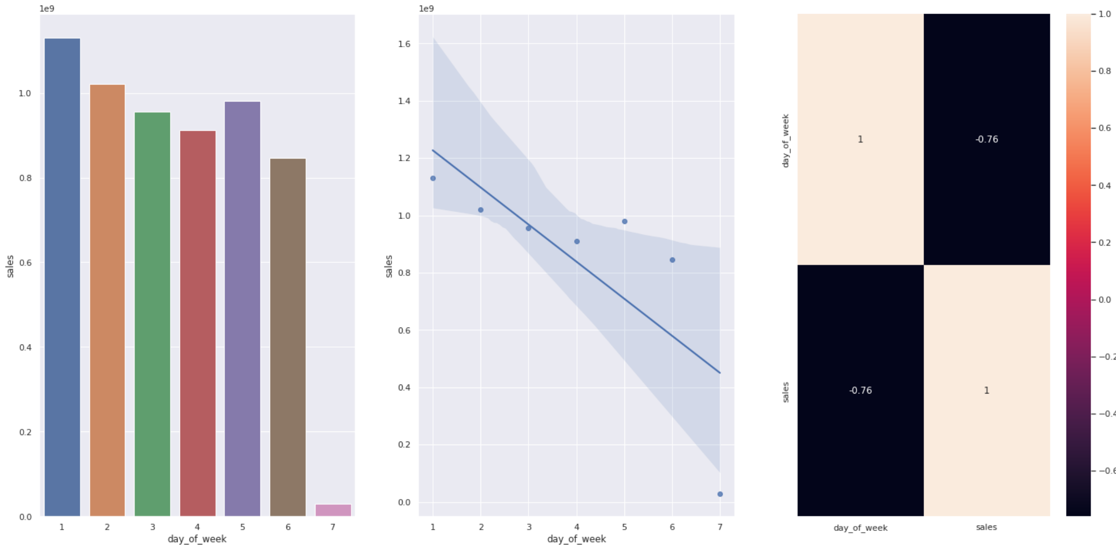
[41]:
```
aux1 = df4[['day', 'sales']].groupby('day').sum().reset_index()

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);

aux1['before_after'] = aux1['day'].apply(lambda x: 'before_10_days' if x <= 10␣
↪else 'after_10_days')
aux2 = aux1[['before_after', 'sales']].groupby('before_after').sum().
↪reset_index()

plt.subplot(2,2,4)
sns.barplot(x='before_after', y='sales', data=aux2);
```




#### 5.2.11 H11 Lojas deveriam vender menos aos finais de semana.

**VERDADEIRA** Lojas vendem menos nos finais de semana

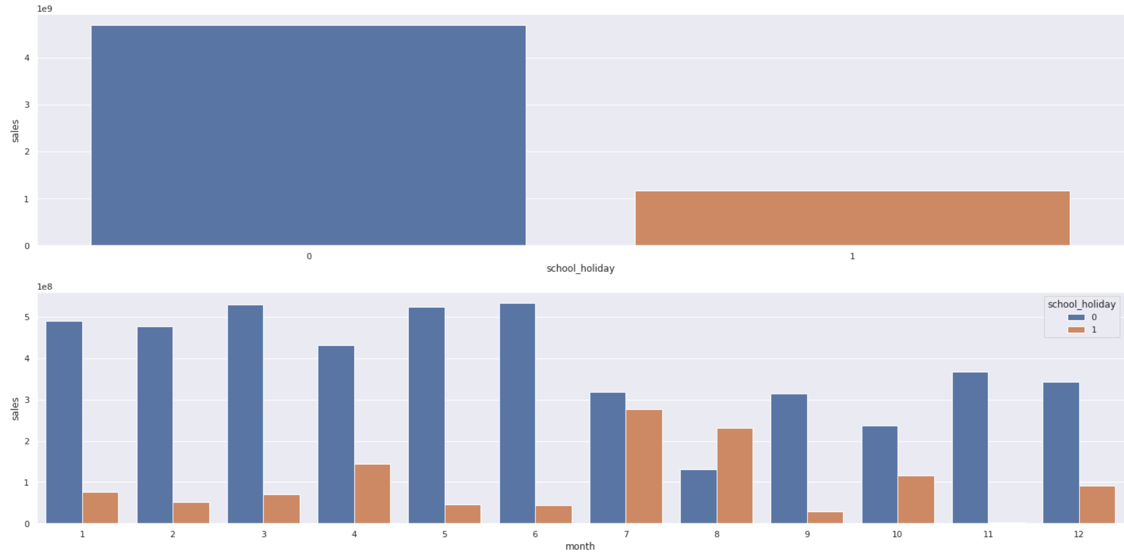
[42]:
```
aux1 = df4[['day_of_week', 'sales']].groupby('day_of_week').sum().reset_index()

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);
```




#### 5.2.12 H12 Lojas deveriam vender menos durante os feriados escolares.

**VERDADEIRA** Lojas vendem menos durante os feriados escolares, exceto os meses de Julho e Agosto

[43]:
```
aux1 = df4[['school_holiday', 'sales']].groupby('school_holiday').sum().
  ↪reset_index()
plt.subplot(2,1,1)
sns.barplot(x='school_holiday', y='sales', data=aux1);

aux2 = df4[['month', 'school_holiday', 'sales']].groupby(['month',␣
  ↪'school_holiday']).sum().reset_index()
plt.subplot(2,1,2)
sns.barplot(x='month', y='sales', hue='school_holiday', data=aux2);
```




#### 5.2.13 Resumo das hipoteses

[44]:
```
tab =[['Hipoteses', 'Conclusao', 'Relevancia'],
      ['H1', 'Falsa', 'Baixa'],
      ['H2', 'Falsa', 'Media'],
      ['H3', 'Falsa', 'Media'],
      ['H4', 'Falsa', 'Baixa'],
      ['H5', '-', '-'],
      ['H7', 'Falsa', 'Baixa'],
      ['H8', 'Falsa', 'Media'],
      ['H9', 'Falsa', 'Alta'],
      ['H10', 'Falsa', 'Alta'],
      ['H11', 'Verdadeira', 'Alta'],
      ['H12', 'Verdadeira', 'Alta'],
      ['H13', 'Verdadeira', 'Baixa'],
     ]
print(tabulate( tab, headers='firstrow' ))
```

```
Hipoteses    Conclusao    Relevancia
-----------  -----------  ------------
H1           Falsa        Baixa
H2           Falsa        Media
H3           Falsa        Media
H4           Falsa        Baixa
H5           -            -
H7           Falsa        Baixa
H8           Falsa        Media
H9           Falsa        Alta
H10          Falsa        Alta
```

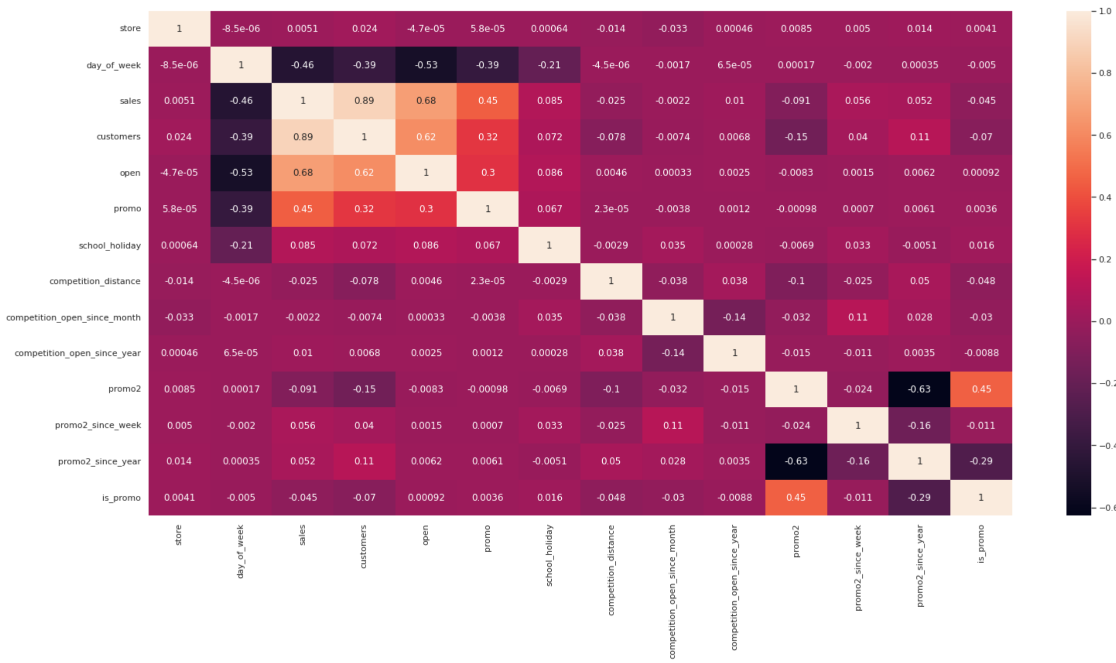
```
H11          Verdadeira   Alta
H12          Verdadeira   Alta
H13          Verdadeira   Baixa
```

## 5.3 ANALISE MULTIVARIADA

### 5.3.1 Numerical Attributes

[45]:
```python
correlation = num_attributes.corr(method='pearson')
sns.heatmap(correlation, annot=True)
```

[45]: <AxesSubplot:>

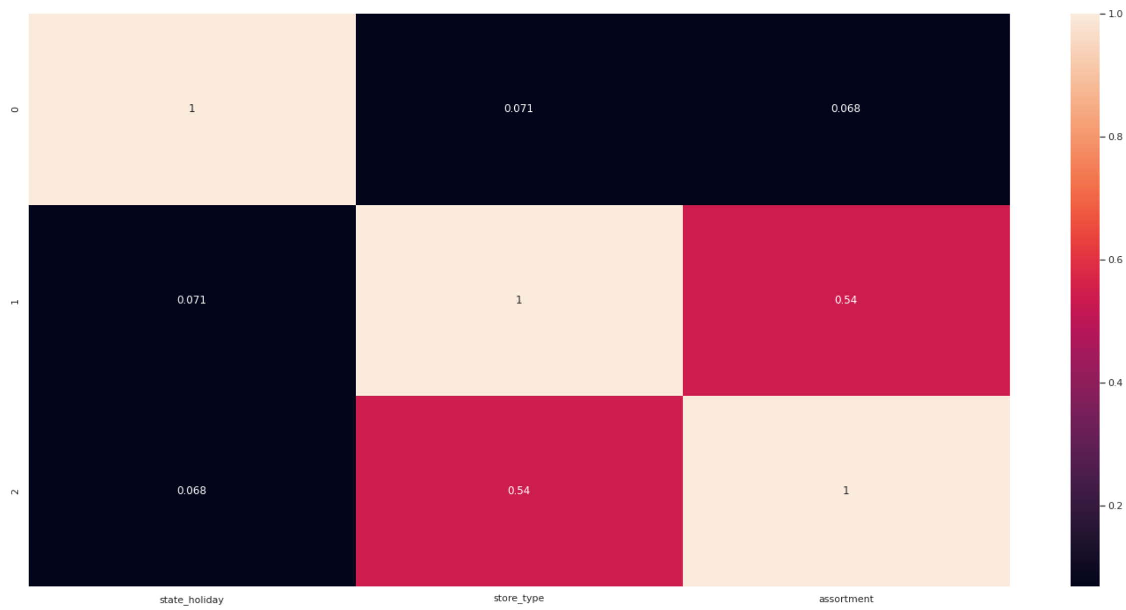


### 5.3.2 Categorical Attributes

[46]:
```python
# only categorical data
a = df4.select_dtypes(include='object')

# 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'])

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

d.set_index(d.columns)

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

[46]: <AxesSubplot:>




# 6 DATA PREPARATION

[98]:
```
df5 = df4.copy()
```

## 6.1 Nomalizacao

- Nao ha dados com distribuicao normal para aplicar esta tecnica

## 6.2 Rescaling

[99]:
```
a = df5.select_dtypes(include=['int64', 'float64'])
```

[100]:
```
rs = RobustScaler()
mms = MinMaxScaler()
# competition_distance
df5['competition_distance'] = rs.fit_transform(df5[['competition_distance']].
 ↪values)
pickle.dump(rs, open('parameter/competition_distance_scaler.pkl', 'wb'))

# competition_time_month
df5['competition_time_month'] = rs.
 ↪fit_transform(df5[['competition_time_month']].values)
pickle.dump(rs, open('parameter/competition_time_month_scaler.pkl', 'wb'))

# promo_time_week
df5['promo_time_week'] = mms.fit_transform(df5[['promo_time_week']].values)
pickle.dump(mms, open('parameter/promo_time_week_scaler.pkl', 'wb'))

# year
df5['year'] = mms.fit_transform(df5[['year']].values)
pickle.dump(mms, open('parameter/year_scaler.pkl', 'wb'))
```

## 6.3 Transformacao

### 6.3.1 Encoding

[96]:
```
# state_holiday - One Hot Encoding
df5 = pd.get_dummies(df5, prefix=['state_holiday'], columns=['state_holiday'])

# store_type - Label Encoding
le = LabelEncoder()
df5['store_type'] = le.fit_transform(df5['store_type'])
pickle.dump(le, open('parameter/store_type_scaler.pkl', 'wb'))

# assortment - Ordinal Encoding
assortment_dict = {'basic': 1, 'extra': 2, 'extended':3}
df5['assortment'] = df5['assortment'].map(assortment_dict)
```

### 6.3.2 Response Variable Transformation

[91]:
```
df5['sales'] = np.log1p(df5['sales'])
```

### 6.3.3 Nature Transformation

[52]:
```
# day_of_week
df5['day_of_week_sin'] = df5['day_of_week'].apply(lambda x: np.sin(x*(2*np.pi/
↪7)))
df5['day_of_week_cos'] = df5['day_of_week'].apply(lambda x: np.cos(x*(2*np.pi/
↪7)))

# month
df5['month_sin'] = df5['month'].apply(lambda x: np.sin(x*(2*np.pi/12)))
df5['month_cos'] = df5['month'].apply(lambda x: np.cos(x*(2*np.pi/12)))

# day
df5['day_sin'] = df5['day'].apply(lambda x: np.sin(x*(2*np.pi/30)))
df5['day_cos'] = df5['day'].apply(lambda x: np.cos(x*(2*np.pi/30)))

# week of year
df5['week_of_year_sin'] = df5['week_of_year'].apply(lambda x: np.sin(x*(2*np.pi/
↪52)))
df5['week_of_year_cos'] = df5['week_of_year'].apply(lambda x: np.cos(x*(2*np.pi/
↪52)))
```

# 7 FEATURE SELECTION

[53]:
```
df6 = df5.copy()
```

## 7.1 Split Dataframe into training and test dataset

[54]:
```
cols_drop = ['week_of_year', 'day', 'month', 'day_of_week', 'promo_since',␣
↪'competition_since', 'year_week']
df6= df6.drop(cols_drop, axis=1)
```

[55]:
```
df6[['store', 'date']].groupby('store').max().reset_index()['date'][0] -␣
↪datetime.timedelta(days=6*7)
```

[55]:
```
Timestamp('2015-06-19 00:00:00')
```

[56]:
```
#training dataset
X_train = df6[df6['date'] < '2015-06-19']
y_train = X_train['sales']

#test dataset
X_test = df6[df6['date'] >= '2015-06-19']
y_test = X_test['sales']

print('Training Min Date: {}'.format(X_train['date'].min()))
```

```python
print('Training Max Date: {}'.format(X_train['date'].max()))

print('\nTest 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: 2015-06-19 00:00:00
Test Max Date: 2015-07-31 00:00:00
```

### 7.2 Boruta as Feature Selector

[57]:
```python
# # training and test datase for Boruta
# X_train_n = X_train.drop(['date', 'sales'], axis=1).values
# y_train_n = y_train.values.ravel()

# # define RandomForestRegressor
# rf = RandomForestRegressor(n_jobs=-1)

# # define Boruta
# boruta = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=42).
↪fit(X_train_n, y_train_n)
```

#### 7.2.1 Best Features from Boruta

[58]:
```python
# cols_selected = boruta.support_.tolist()

# # best features
# X_train_fs = X_train.drop(['date', 'sales'], axis=1)
# cols_selected_boruta = X_train_fs.iloc[:, cols_selected].columns.to_list()

# cols_not_selected_boruta = list(np.setdiff1d(X_train_fs.columns,␣
↪cols_selected_boruta))
```

### 7.3 Manual Feature Selection

[59]:
```python
cols_selected_boruta = [
 'store',
 'promo',
 'store_type',
 'assortment',
 'competition_distance',
 'competition_open_since_month',
 'competition_open_since_year',
 'promo2',
 'promo2_since_week',
```

```
 'promo2_since_year',
 'competition_time_month',
 'promo_time_week',
 'day_of_week_sin',
 'day_of_week_cos',
 'month_sin',
 'month_cos',
 'day_sin',
 'day_cos',
 'week_of_year_sin',
 'week_of_year_cos']

# columns to add
feat_to_add = ['date', 'sales']

# final features
cols_boruta_full = cols_selected_boruta.copy()
cols_boruta_full.extend(feat_to_add)
```

[60]:
```
cols_not_selected_boruta = [
 'is_promo',
 'school_holiday',
 'state_holiday_christmas',
 'state_holiday_easter_holiday',
 'state_holiday_public_holiday',
 'state_holiday_regular_day',
 'year']
```

# 8 MACHINE LEARNING MODELLING

[61]:
```
x_train = X_train[cols_selected_boruta]
x_test = X_test[cols_selected_boruta]

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

## 8.1 Average Model

[62]:
```
aux1 = x_test.copy()
aux1['sales'] = y_test.copy()

# prediction
aux2 = aux1[['store', 'sales']].groupby('store').mean().reset_index().
 ↪rename(columns={'sales':'predictions'})
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
```

[62]:

| | Model Name | MAE | MAPE | RMSE |
|---|---|---|---|---|
| 0 | Average Model | 1354.800353 | 0.455051 | 1835.135542 |

### 8.2 Linear Regression Model

[63]:
```
# 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
```

[63]:

| | Model Name | MAE | MAPE | RMSE |
|---|---|---|---|---|
| 0 | Linear Regression | 1867.089774 | 0.292694 | 2671.049215 |

### 8.3 Linear Regression Model - Cross Validation

[64]:
```
lr_result_cv = cross_validation(x_training, 5, 'Linear Regression', lr,␣
 ↪verbose=False)
lr_result_cv
```

[64]:

| | Model Name | MAE CV | MAPE CV | RMSE CV |
|---|---|---|---|---|
| 0 | Linear Regression | 2081.73 +/- 295.63 | 0.3 +/- 0.02 | 2952.52 +/- 468.37 |

### 8.4 Linear Regression Regularized Model - Lasso

[65]:
```
# model
lrr = Lasso(alpha=0.01).fit(x_train, y_train)

# prediction
yhat_lrr = lrr.predict(x_test)

# performance
lrr_result = ml_error('Linear Regression Regularized - Lasso', np.
 ↪expm1(y_test), np.expm1(yhat_lrr))
lrr_result
```

[65]:

| | Model Name | MAE | MAPE | RMSE |
|---|---|---|---|---|
| 0 | Linear Regression Regularized - Lasso | 1891.704881 | 0.289106 | 2744.451737 |

### 8.5 Lasso - Cross Validation

[66]:
```
lrr_result_cv = cross_validation(x_training, 5, 'Lasso', lrr, verbose=False)
lrr_result_cv
```

[66]:

| | Model Name | MAE CV | MAPE CV | RMSE CV |
|---|---|---|---|---|
| 0 | Lasso | 2116.38 +/- 341.5 | 0.29 +/- 0.01 | 3057.75 +/- 504.26 |

### 8.6 Random Forest Regressor Model

[67]:
```
# model
rf = RandomForestRegressor(n_estimators=100, n_jobs=-1, random_state=42).
 ↪fit(x_train, y_train)

# prediction
yhat_rf = rf.predict(x_test)

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

[67]:

| | Model Name | MAE | MAPE | RMSE |
|---|---|---|---|---|
| 0 | Random Forest Regressor | 680.310758 | 0.100045 | 1011.960854 |

### 8.7 Random Forest Regressor Model - Cross Validation

[68]:
```
rf_result_cv = cross_validation(x_training, 5, 'Random Forest Regressor', rf,␣
 ↪verbose=False)
rf_result_cv
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [68], in <cell line: 1>()
----> 1 rf_result_cv =␣
 ↪cross_validation(x_training, 5, 'Random Forest Regressor', rf, verbose=False)
      2 rf_result_cv

Input In [4], in cross_validation(x_training, kfold, model_name, model, verbose
     24 yvalidation = validation['sales']
     26 # model
---> 27 m = model.fit(xtraining, ytraining)
     29 # prediction
```

```
     30 yhat = m.predict(xvalidation)

File ~/.pyenv/versions/3.9.5/envs/dsemproducao/lib/python3.9/site-packages/
↪sklearn/ensemble/_forest.py:476, in BaseForest.fit(self, X, y, sample_weight)
    465 trees = [
    466     self._make_estimator(append=False, random_state=random_state)
    467     for i in range(n_more_estimators)
    468 ]
    470 # Parallel loop: we prefer the threading backend as the Cython code
    471 # for fitting the trees is internally releasing the Python GIL
    472 # making threading more efficient than multiprocessing in
    473 # that case. However, for joblib 0.12+ we respect any
    474 # parallel_backend contexts set at a higher level,
    475 # since correctness does not rely on using threads.
--> 476 trees = Parallel(
    477     n_jobs=self.n_jobs,
    478     verbose=self.verbose,
    479     prefer="threads",
    480 )(
    481     delayed(_parallel_build_trees)(
    482         t,
    483         self.bootstrap,
    484         X,
    485         y,
    486         sample_weight,
    487         i,
    488         len(trees),
    489         verbose=self.verbose,
    490         class_weight=self.class_weight,
    491         n_samples_bootstrap=n_samples_bootstrap,
    492     )
    493     for i, t in enumerate(trees)
    494 )
    496 # Collect newly grown trees
    497 self.estimators_.extend(trees)

File ~/.pyenv/versions/3.9.5/envs/dsemproducao/lib/python3.9/site-packages/
↪joblib/parallel.py:1056, in Parallel.__call__(self, iterable)
   1053     self._iterating = False
   1055 with self._backend.retrieval_context():
-> 1056     self.retrieve()
   1057 # Make sure that we get a last message telling us we are done
   1058 elapsed_time = time.time() - self._start_time

File ~/.pyenv/versions/3.9.5/envs/dsemproducao/lib/python3.9/site-packages/
↪joblib/parallel.py:935, in Parallel.retrieve(self)
    933 try:
    934     if getattr(self._backend, 'supports_timeout', False):
```

```
--> 935         self._output.extend(job.get(timeout=self.timeout))
    936     else:
    937         self._output.extend(job.get())

File ~/.pyenv/versions/3.9.5/lib/python3.9/multiprocessing/pool.py:765, in␣
↪ApplyResult.get(self, timeout)
    764 def get(self, timeout=None):
--> 765     self.wait(timeout)
    766     if not self.ready():
    767         raise TimeoutError

File ~/.pyenv/versions/3.9.5/lib/python3.9/multiprocessing/pool.py:762, in␣
↪ApplyResult.wait(self, timeout)
    761 def wait(self, timeout=None):
--> 762     self._event.wait(timeout)

File ~/.pyenv/versions/3.9.5/lib/python3.9/threading.py:574, in Event.wait(self ␣
↪timeout)
    572 signaled = self._flag
    573 if not signaled:
--> 574     signaled = self._cond.wait(timeout)
    575 return signaled

File ~/.pyenv/versions/3.9.5/lib/python3.9/threading.py:312, in Condition.
↪wait(self, timeout)
    310 try:    # restore state no matter what (e.g., KeyboardInterrupt)
    311     if timeout is None:
--> 312         waiter.acquire()
    313         gotit = True
    314     else:

KeyboardInterrupt:
```

## 8.8 XGBoost Regressor Model

[69]:
```
# model
model_xgb = xgb.XGBRegressor(objective='reg:squarederror',
                             n_estimators=100,
                             eta=0.01,
                             max_depth=10,
                             subsample=0.7,
                             colsample_bytree=0.9).fit(x_train, y_train)

# prediction
yhat_xgb = model_xgb.predict(x_test)
```

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

[69]:

| | Model Name | MAE | MAPE | RMSE |
|---|---|---|---|---|
| 0 | XGBoost Regressor | 6683.705545 | 0.949492 | 7330.988585 |

### 8.9 XGBoost Regressor Model - Cross Validation

[70]:

```python
xgb_result_cv = cross_validation(x_training, 5, 'XGBoost Regressor', model_xgb,␣
↪verbose=False)
xgb_result_cv
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [70], in <cell line: 1>()
----> 1 xgb_result_cv =␣
↪cross_validation(x_training, 5, 'XGBoost Regressor', model_xgb, verbose=False
      2 xgb_result_cv

Input In [4], in cross_validation(x_training, kfold, model_name, model, verbose
     24 yvalidation = validation['sales']
     26 # model
---> 27 m = model.fit(xtraining, ytraining)
     29 # prediction
     30 yhat = m.predict(xvalidation)

File ~/.pyenv/versions/3.9.5/envs/dsemproducao/lib/python3.9/site-packages/
↪xgboost/core.py:532, in _deprecate_positional_args.<locals>.inner_f(*args,␣
↪**kwargs)
    530 for k, arg in zip(sig.parameters, args):
    531     kwargs[k] = arg
--> 532 return f(**kwargs)

File ~/.pyenv/versions/3.9.5/envs/dsemproducao/lib/python3.9/site-packages/
↪xgboost/sklearn.py:961, in XGBModel.fit(self, X, y, sample_weight,␣
↪base_margin, eval_set, eval_metric, early_stopping_rounds, verbose, xgb_model␣
↪sample_weight_eval_set, base_margin_eval_set, feature_weights, callbacks)
    956     obj = None
    958 model, metric, params, early_stopping_rounds, callbacks = self.
↪_configure_fit(
    959     xgb_model, eval_metric, params, early_stopping_rounds, callbacks
    960 )
--> 961 self._Booster = train(
    962     params,
    963     train_dmatrix,
    964     self.get_num_boosting_rounds(),
```

```
    965     evals=evals,
    966     early_stopping_rounds=early_stopping_rounds,
    967     evals_result=evals_result,
    968     obj=obj,
    969     custom_metric=metric,
    970     verbose_eval=verbose,
    971     xgb_model=model,
    972     callbacks=callbacks,
    973 )
    975 self._set_evaluation_result(evals_result)
    976 return self

File ~/.pyenv/versions/3.9.5/envs/dsemproducao/lib/python3.9/site-packages/xgboost/core.py:532, in _deprecate_positional_args.<locals>.inner_f(*args, **kwargs)
    530 for k, arg in zip(sig.parameters, args):
    531     kwargs[k] = arg
--> 532 return f(**kwargs)

File ~/.pyenv/versions/3.9.5/envs/dsemproducao/lib/python3.9/site-packages/xgboost/training.py:181, in train(params, dtrain, num_boost_round, evals, obj, feval, maximize, early_stopping_rounds, evals_result, verbose_eval, xgb_model, callbacks, custom_metric)
    179 if cb_container.before_iteration(bst, i, dtrain, evals):
    180     break
--> 181 bst.update(dtrain, i, obj)
    182 if cb_container.after_iteration(bst, i, dtrain, evals):
    183     break

File ~/.pyenv/versions/3.9.5/envs/dsemproducao/lib/python3.9/site-packages/xgboost/core.py:1733, in Booster.update(self, dtrain, iteration, fobj)
   1730 self._validate_features(dtrain)
   1732 if fobj is None:
-> 1733     _check_call(_LIB.XGBoosterUpdateOneIter(self.handle,
   1734                                             ctypes.c_int(iteration),
   1735                                             dtrain.handle))
   1736 else:
   1737     pred = self.predict(dtrain, output_margin=True, training=True)

KeyboardInterrupt:
```

### 8.10 Single Performance

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

### 8.11 Real Performance - Cross Validation

[ ]:
```
modelling_result_cv = pd.concat([lr_result_cv, lrr_result_cv, rf_result_cv,␣
↪xgb_result_cv])
modelling_result_cv.sort_values('RMSE CV')
```

# 9 HYPERPARAMETER FINE TUNING

## 9.1 Random Search

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

MAX_EVAL = 10
```

[ ]:
```
# 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['sub_sample'],
#                                  colsample_bytree=hp['colsample_bytree'],␣
↪min_child_weight=hp['min_child_weight'])

#     # performance
#     result = cross_validation(x_training, 5, 'XGBoost Regressor', model_xgb,␣
↪verbose=False)
#     final_result = pd.concat([final_result, result])

# final_result
```

## 9.2 Final model

{'n_estimators': 3500, 'eta': 0.03, 'max_depth': 9, 'sub_sample': 0.5, 'colsample_bytree': 0.9, 'min_child_weight': 8}

[71]:
```
param_tuned={
    'n_estimators': 3000,
    'eta': 0.03,
    'max_depth': 9,
    'sub_sample': 0.5,
    'colsample_bytree': 0.9,
    'min_child_weight': 8
       }
```

[72]:
```
# 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['sub_sample'],
                                    ␣
 ↪colsample_bytree=param_tuned['colsample_bytree'],
                                    ␣
 ↪min_child_weight=param_tuned['min_child_weight']).fit(x_train, y_train)
# 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_tuned))
xgb_result_tuned
```

[72]:

| | Model Name | MAE | MAPE | RMSE |
|---|---|---|---|---|
| 0 | XGBoost Regressor | 620.067026 | 0.089582 | 910.543822 |

[74]:
```
mpe = mean_percentage_error(np.expm1(y_test), np.expm1(yhat_xgb_tuned))
mpe
```

[74]: 0.006003645581086666

# 10 TRADUCAO E INTERPRETACAO DO ERRO

[2]:
```
df9 = X_test[cols_boruta_full]

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [2], in <cell line: 1>()
----> 1 df9 = X_test[cols_boruta_full]
      3 # rescale
      4 df9['sales'] = np.expm1(df9['sales'])

NameError: name 'X_test' is not defined
```

## 10.1 BUSINESS PERFORMANCE

[76]:

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

# MAE and MAPE
df9_aux1 = df9[['store', 'sales', 'predictions']].groupby('store').apply(lambda␣
↪x: mean_absolute_error(x['sales'], x['predictions'])).reset_index().
↪rename(columns={0:'MAE'})
df9_aux2 = df9[['store', 'sales', 'predictions']].groupby('store').apply(lambda␣
↪x: mean_absolute_percentage_error(x['sales'], x['predictions'])).
↪reset_index().rename(columns={0:'MAPE'})

# 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']]
```

[77]:

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

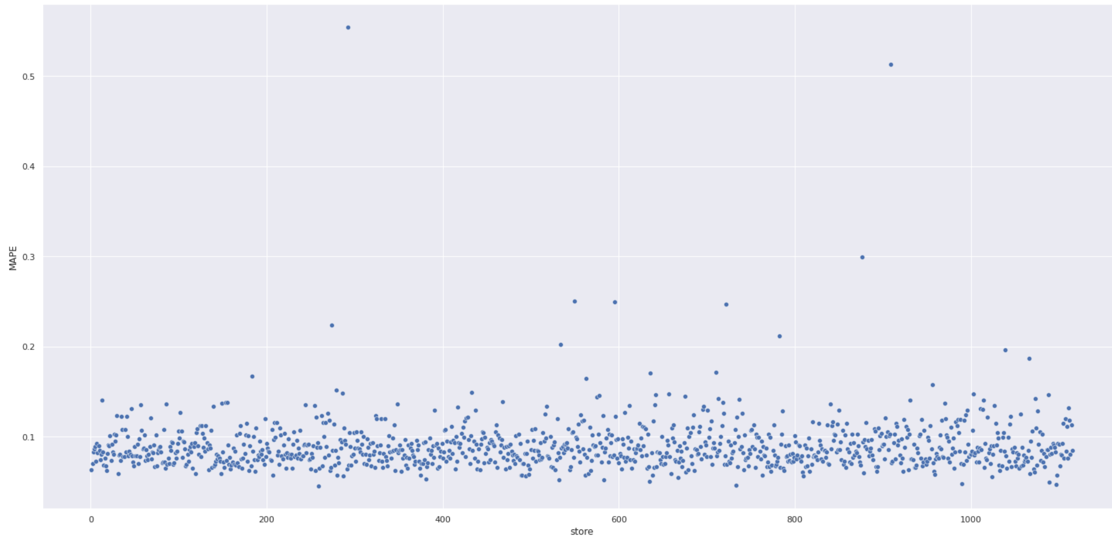
[77]:

| | store | predictions | worst_scenario | best_scenario | MAE | MAPE |
|---|---|---|---|---|---|---|
| 291 | 292 | 104080.976562 | 100754.290425 | 107407.662700 | 3326.686138 | 0.554840 |
| 908 | 909 | 237274.953125 | 229721.995436 | 244827.910814 | 7552.957689 | 0.512914 |
| 875 | 876 | 202620.781250 | 198622.608080 | 206618.954420 | 3998.173170 | 0.299266 |
| 549 | 550 | 240955.656250 | 239638.594436 | 242272.718064 | 1317.061814 | 0.250513 |
| 594 | 595 | 393923.781250 | 390267.315535 | 397580.246965 | 3656.465715 | 0.249605 |

[78]:

```python
sns.scatterplot(x='store', y='MAPE', data=df92)
```

```
[78]: <AxesSubplot:xlabel='store', ylabel='MAPE'>
```




## 10.2 TOTAL PERFORMANCE

```
[79]: df93 = df92[['predictions', 'worst_scenario', 'best_scenario']].apply(lambda x:␣
        ↪np.sum(x), axis=0).reset_index().rename(columns={'index':'Scenario', 0:
        ↪'Values'})
      df93['Values']  =df93['Values'].map('R${:,.2f}'.format)
      df93
```

[79]:

| | Scenario | Values |
|---|---|---|
| 0 | predictions | R$283,484,000.00 |
| 1 | worst_scenario | R$282,788,590.26 |
| 2 | best_scenario | R$284,179,403.04 |

## 10.3 MACHINE LEARNING PERFORMANCE

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

```
[81]: 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'])
```

[81]: <AxesSubplot:xlabel='predictions', ylabel='error'>


# 11 DEPLOY MODEL TO PRODUCTION

[82]:
```
# Save Trained Model
pickle.dump(model_xgb_tuned, open('/home/gabrielpastega/repos/ds_em_producao/
 ↪dsemproducao/model/model_rossmann.pkl', 'wb'))
```

## 11.1 Rossmann Class

[ ]:
```
class Rossmann(object):
    def __init__(self):
        self.competition_distance_scaler   = pickle.load(open('parameter/
 ↪competition_distance_scaler.pkl', 'rb'))
        self.promo_time_week_scaler        = pickle.load(open('parameter/
 ↪promo_time_week_scaler.pkl', 'rb'))
        self.competition_time_month_scaler = pickle.load(open('parameter/
 ↪competition_time_month_scaler.pkl', 'rb'))
        self.year_scaler                   = pickle.load(open('parameter/
 ↪year_scaler.pkl', 'rb'))
        self.store_type_scaler             = pickle.load(open('parameter/
 ↪store_type_scaler.pkl', 'rb'))
```

```python
    def data_cleaning(self, df1):

        ## RENAME COLUMNS

        cols_old = ['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open',␣
↪'Promo', 'StateHoliday', 'SchoolHoliday',
                    'StoreType', 'Assortment', 'CompetitionDistance',␣
↪'CompetitionOpenSinceMonth',
                    'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWeek',␣
↪'Promo2SinceYear', 'PromoInterval']

        snakecase = lambda x: inflection.underscore(x)

        cols_new = list(map(snakecase, cols_old))

        # rename
        df1.columns = cols_new

        ## DATA TYPES

        df1['date'] = pd.to_datetime(df1['date'])

        ## FILLOUT NA
        # competition_distance
        df1['competition_distance'] = df1['competition_distance'].apply(lambda␣
↪x: 200000.0 if math.isnan(x) else x)

        # competition_open_since_month
        df1['competition_open_since_month'] = df1.apply(lambda x: x['date'].
↪month if math.isnan(x['competition_open_since_month']) else␣
↪x['competition_open_since_month'], axis=1)

        # competition_open_since_year
        df1['competition_open_since_year'] = df1.apply(lambda x: x['date'].year␣
↪if math.isnan(x['competition_open_since_year']) else␣
↪x['competition_open_since_year'], axis=1)

        # promo2_since_week
        df1['promo2_since_week'] = df1.apply(lambda x: x['date'].week if math.
↪isnan(x['promo2_since_week']) else x['promo2_since_week'], axis=1)

        # promo2_since_year
        df1['promo2_since_year'] = df1.apply(lambda x: x['date'].year if math.
↪isnan(x['promo2_since_year']) else x['promo2_since_year'], axis=1)
```

```
        # promo_interval
        month_map = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug', 9: 'Sept', 10: 'Oct', 11: 'Nov', 12: 'Dec'}

        df1['promo_interval'].fillna(0, inplace=True)

        df1['month_map'] = df1['date'].dt.month.map(month_map)

        df1['is_promo'] = df1[['promo_interval', 'month_map']].apply(lambda x: 0 if x['promo_interval'] == 0 else 1 if x['month_map'] in x['promo_interval'].split(',') else 0, axis=1)


        ## CHANGE DATA TYPES
        # competition
        df1['competition_open_since_month'] = df1['competition_open_since_month'].astype(int)
        df1['competition_open_since_year'] = df1['competition_open_since_year'].astype(int)

        # promo2
        df1['promo2_since_week'] = df1['promo2_since_week'].astype(int)
        df1['promo2_since_year'] = df1['promo2_since_year'].astype(int)

        return df1

        def feature_engineering(self, df2):

        ## FEATURE ENGINEERING
        # year
        df2['year'] = df2['date'].dt.year

        # month
        df2['month'] = df2['date'].dt.month

        # day
        df2['day'] = df2['date'].dt.day

        # week of year
        df2['week_of_year'] = df2['date'].dt.weekofyear

        # year week
        df2['year_week'] = df1['date'].dt.strftime('%Y-%W')

        # competition since
```

```
        df2['competition_since'] = df2.apply(lambda x: datetime.
↪datetime(year=x['competition_open_since_year'],␣
↪month=x['competition_open_since_month'],day=1), axis=1)
        df2['competition_time_month'] =  ((df2['date'] -␣
↪df2['competition_since'])/30).apply(lambda x: x.days).astype(int)

        # promo since
        df2['promo_since'] =  df2['promo2_since_year'].astype(str) + '-' +␣
↪df2['promo2_since_week'].astype(str)
        df2['promo_since'] = df2['promo_since'].apply(lambda x: datetime.
↪datetime.strptime(x + '-1', '%Y-%W-%w') - datetime.timedelta(days=7))
        df2['promo_time_week'] = ((df2['date'] - df2['promo_since'])/7).
↪apply(lambda x: x.days).astype(int)

        # assortment
        df2['assortment'] = df2['assortment'].apply(lambda x: 'basic' if x ==␣
↪'a' else 'extra' if x == 'b' else 'extended')

        # state holiday
        df2['state_holiday'] = df2['state_holiday'].apply(lambda x:␣
↪'public_holiday' if x == 'a' else 'easter_holiday' if x == 'b' else␣
↪'christmas' if x == 'c' else 'regular_day')

        ## LINE FILTER

        df2 = df2[(df2['open'] != 0) & (df2['sales'] > 0)]

        ## COLUMNS FILTER

        cols_drop = ['customers', 'open', 'promo_interval', 'month_map']
        df2 = df2.drop(cols_drop, axis=1)

        return df2

        def data_preparation(self, df3):

        ## RESCALING
        # competition_distance
        df3['competition_distance'] = self.competition_distance_scaler.
↪transform(df3[['competition_distance']].values)

        # competition_time_month
        df3['competition_time_month'] = self.competition_time_month_scaler.
↪transform(df3[['competition_time_month']].values)

        # promo_time_week
```

```
        df3['promo_time_week'] = self.promo_time_week_scaler.
↪transform(df3[['promo_time_week']].values)

        # year
        df3['year'] = self.year_scaler.fit_transform(df3[['year']].values)

        ## TRANSFORMATION
        ### ENCODING
        # state_holiday - One Hot Encoding
        df3 = pd.get_dummies(df3, prefix=['state_holiday'],␣
↪columns=['state_holiday'])

        # store_type - Label Encoding
        df3['store_type'] = self.store_type_scaler.
↪fit_transform(df3['store_type'])

        # assortment - Ordinal Encoding
        assortment_dict = {'basic': 1, 'extra': 2, 'extended':3}
        df3['assortment'] = df3['assortment'].map(assortment_dict)

        ### NATURE TRANSFORMATION
        # day_of_week
        df3['day_of_week_sin'] = df3['day_of_week'].apply(lambda x: np.
↪sin(x*(2*np.pi/7)))
        df3['day_of_week_cos'] = df3['day_of_week'].apply(lambda x: np.
↪cos(x*(2*np.pi/7)))

        # month
        df3['month_sin'] = df3['month'].apply(lambda x: np.sin(x*(2*np.pi/12)))
        df3['month_cos'] = df3['month'].apply(lambda x: np.cos(x*(2*np.pi/12)))

        # day
        df3['day_sin'] = df3['day'].apply(lambda x: np.sin(x*(2*np.pi/30)))
        df3['day_cos'] = df3['day'].apply(lambda x: np.cos(x*(2*np.pi/30)))

        # week of year
        df3['week_of_year_sin'] = df3['week_of_year'].apply(lambda x: np.
↪sin(x*(2*np.pi/52)))
        df3['week_of_year_cos'] = df3['week_of_year'].apply(lambda x: np.
↪cos(x*(2*np.pi/52)))

        cols_selected = [ 'store', 'promo', 'store_type', 'assortment',␣
↪'competition_distance', 'competition_open_since_month',
                                'competition_open_since_year', 'promo2',␣
↪'promo2_since_week', 'promo2_since_year', 'competition_time_month',
```

```
                                 'promo_time_week', 'day_of_week_sin',␣
↪'day_of_week_cos', 'month_sin', 'month_cos', 'day_sin', 'day_cos',
                                 'week_of_year_sin', 'week_of_year_cos']

        return df3[cols_selected]
```

### 11.2 API Handler

[ ]:
```
import pickle
import pandas as pd
from flask              import Flask, request, Response
from rossmann.Rossmann import Rossmann

# loading model
model = pickle.load(open('/home/gabrielpastega/repos/ds_em_producao/
↪dsemproducao/model/model_rossmann.pkl', 'rb'))

# initialize API
app = Flask(__name__)

@app.route('/rossmann/predict', methods=['POST'])
def rossmann_predict():
    test_json = request.get_json()

    if test_json: # there is data
        if isinstance(test_json, dict):# Unique example
            test_raw = pd.DataFrame(test_json, index=[0])

        else: # Multiple examples
            test_raw = pd.DataFrame(test_json, columns=test_json[0].keys())

        # Instantiate Rossmann Class
        pipeline = Rossmann()

        # data cleaning
        df1 = pipeline.data_cleaning(test_raw)

        # feature engineering
        df2 = pipeline.feature_engineering(df1)

        # data preparation
        df3 = pipeline.data_preparation(df2)

        # prediction
        df_response = pipeline.get_prediction(model, test_raw, df3)

        return df_response
```

```python
    else:
        return Response('{}', status=200, mimetype='application/json')

if __name__ == '__main__':
    app.run('0.0.0.0')
```

### 11.3 API Tester

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: