m10_v01_store_sales_prediction

September 12, 2021

1 0.0. IMPORTS

[1]:
```
import math
import numpy  as np
import pandas as pd
import random
import pickle
import requests
import warnings
import inflection
import seaborn as sns
import xgboost as xgb

from scipy                 import stats  as ss
from boruta                import BorutaPy
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_squared_error
from sklearn.ensemble      import RandomForestRegressor
from sklearn.linear_model  import LinearRegression, Lasso
from sklearn.preprocessing import RobustScaler, MinMaxScaler, LabelEncoder

warnings.filterwarnings( 'ignore' )
```

1.1 0.1. Helper Functions

[2]:
```
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 ).as_matrix()
    n = cm.sum()
    r, k = cm.shape

    chi2 = ss.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 inline
    %pylab inline

    plt.style.use( 'bmh' )
    plt.rcParams['figure.figsize'] = [25, 12]
    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()
```

[3]:
```
jupyter_settings()
```

```
Populating the interactive namespace from numpy and matplotlib
```

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

1.2 0.2. Loading data

[4]:
```
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 1.0. PASSO 01 - DESCRICAO DOS DADOS

[5]:
```
df1 = df_raw.copy()
```

2.1 1.1. Rename Columns

[6]:
```
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 1.2. Data Dimensions

[7]:
```
print( 'Number of Rows: {}'.format( df1.shape[0] ) )
print( 'Number of Cols: {}'.format( df1.shape[1] ) )
```

```
Number of Rows: 1017209
Number of Cols: 18
```

2.3 1.3. Data Types

[8]:
```
df1['date'] = pd.to_datetime( df1['date'] )
df1.dtypes
```

[8]:
```
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 1.4. Check NA

[9]:
```
df1.isna().sum()
```

[9]:
```
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 1.5. Fillout NA

[10]:
```
df1.sample()
```

[10]:
```
         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
1010793    274            7 2013-01-06   3802        932     1      0
0                1          b          b                3640.0
NaN                           NaN       1                10.0             2013.0
Jan,Apr,Jul,Oct
```

[11]:
```
#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: 'Fev',  3: 'Mar',  4: 'Apr',  5: 'May',  6: 'Jun', ␣
↪7: 'Jul',  8: 'Aug',  9: 'Sep',  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 )
```

[12]:
```
df1.isna().sum()
```

[12]:
```
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
```

2.6 1.6. Change Data Types

[13]:
```
# competiton
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 )
```

2.7 1.7. Descriptive Statistics

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

2.7.1 1.7.1. Numerical Atributes

[15]:
```
# Central Tendency - mean, meadina
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
```

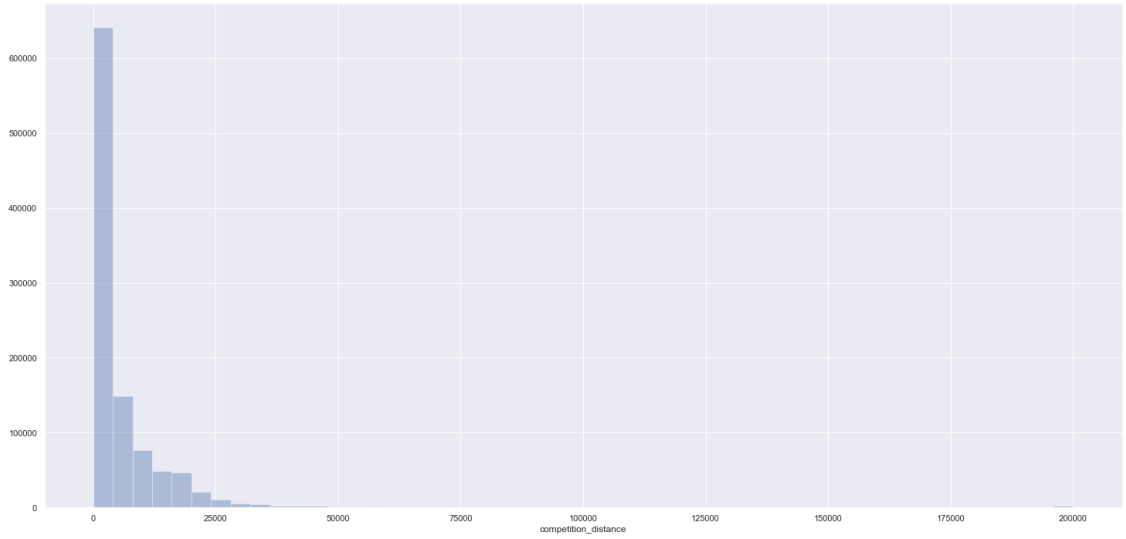
```
d5 = pd.DataFrame( num_attributes.apply( lambda x: x.skew() ) ).T
d6 = pd.DataFrame( num_attributes.apply( lambda x: x.kurtosis() ) ).T

# concatenar
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	0.155231	0.0	0.362124	1.904152	1.625796

[16]:
```
sns.distplot( df1['competition_distance'], kde=False )
```

[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1098a77f0>
```



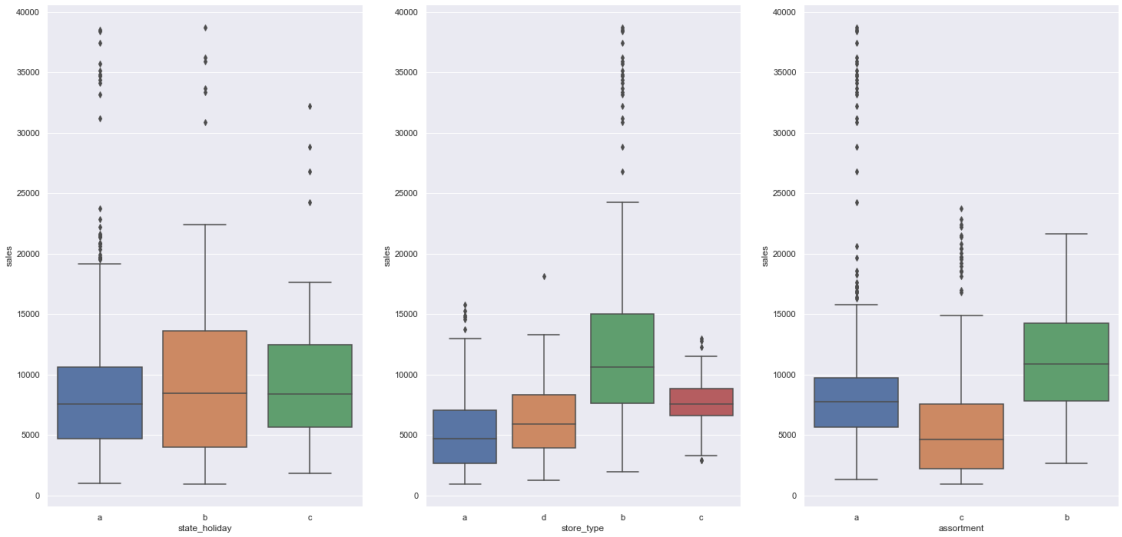
2.7.2 1.7.2. Categorical Atributes

[17]:
```
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]:
```
aux = df1[(df1['state_holiday'] != '0') & (df1['sales'] > 0)]

plt.subplot( 1, 3, 1 )
sns.boxplot( x='state_holiday', y='sales', data=aux )

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

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

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x109917760>
```



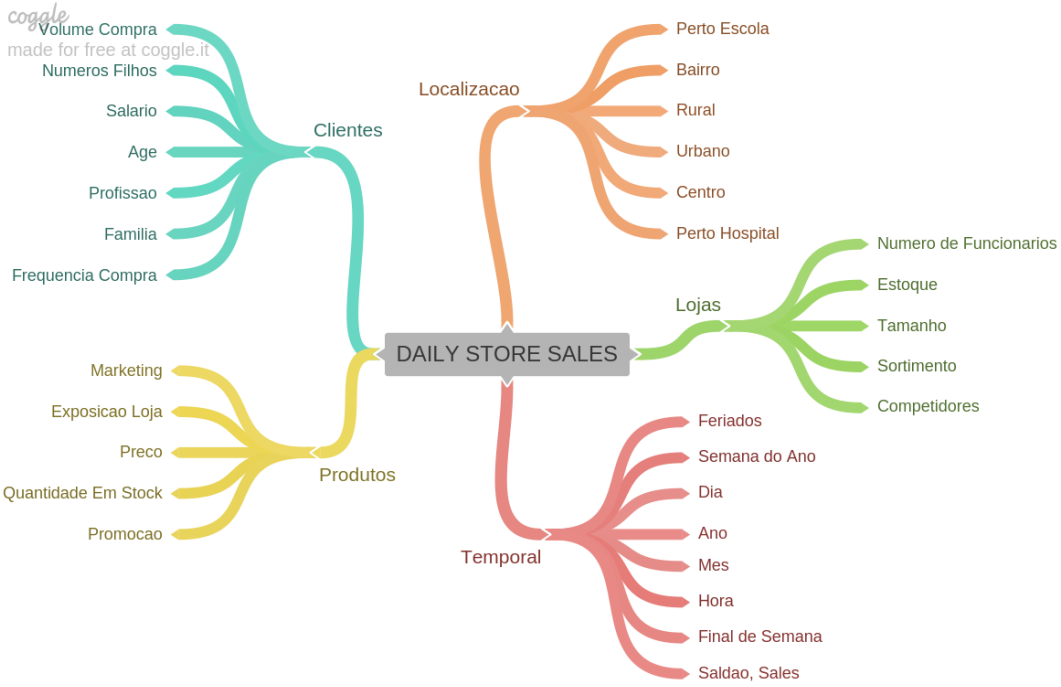
3 2.0. PASSO 02 - FEATURE ENGINEERING

[19]:
```
df2 = df1.copy()
```

3.1 2.1. Mapa Mental de Hipoteses

[20]:
```
Image( 'img/MindMapHypothesis.png' )
```

[20]:



3.2 2.2. Criacao das Hipoteses

3.2.1 2.2.1. Hipoteses Loja

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 2.2.2. Hipoteses Produto

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 2.2.3. Hipoteses Tempo

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 2.3. Lista Final de Hipóteses

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.

7. Lojas com mais promoções consecutivas deveriam vender mais.

8. Lojas abertas durante o feriado de Natal deveriam vender mais.

9. Lojas deveriam vender mais ao longo dos anos.

10. Lojas deveriam vender mais no segundo semestre do ano.

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

12. Lojas deveriam vender menos aos finais de semana.

13. Lojas deveriam vender menos durante os feriados escolares.

3.4 2.4. Feature Engineering

[21]:
```
# 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'] = df2['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' )
```

4 3.0. PASSO 03 - FILTRAGEM DE VARIÁVEIS

[22]:
```
df3 = df2.copy()
```

4.1 3.1. Filtragem das Linhas

[23]:
```
df3 = df3[(df3['open'] != 0) & (df3['sales'] > 0)]
```

4.2 3.2. Selecao das Colunas

[24]:
```
cols_drop = ['customers', 'open', 'promo_interval', 'month_map']
df3 = df3.drop( cols_drop, axis=1 )
```

5 4.0. PASSO 04 - ANALISE EXPLORATORIA DOS DADOS

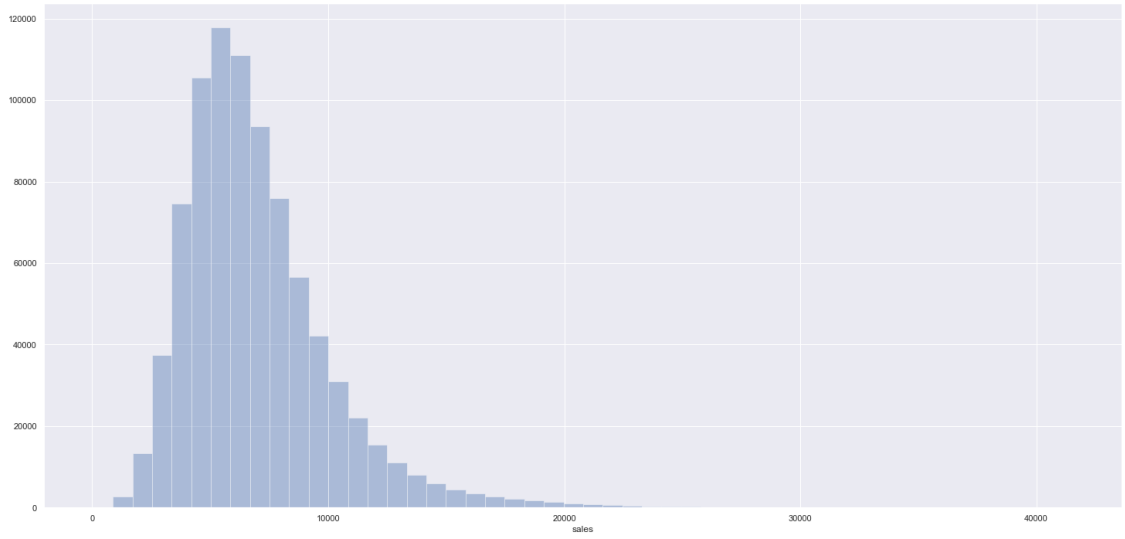
[25]:
```
df4 = df3.copy()
```

5.1 4.1. Analise Univariada

5.1.1 4.1.1. Response Variable

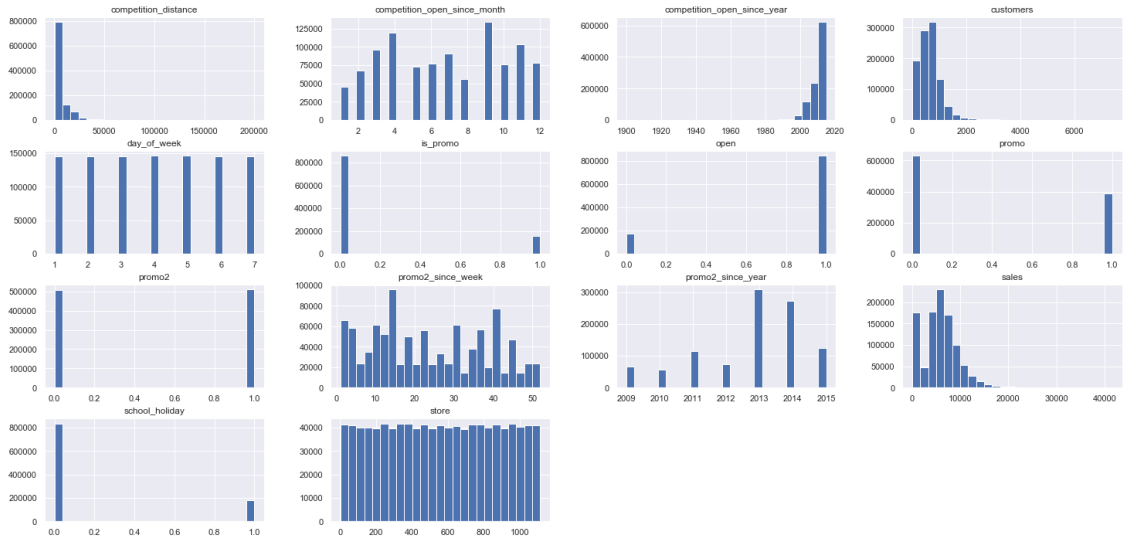
[26]:
```
sns.distplot( df4['sales'], kde=False  )
```

[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11f7a3910>
```



5.1.2 4.1.2. Numerical Variable

[27]:
```
num_attributes.hist( bins=25 );
```

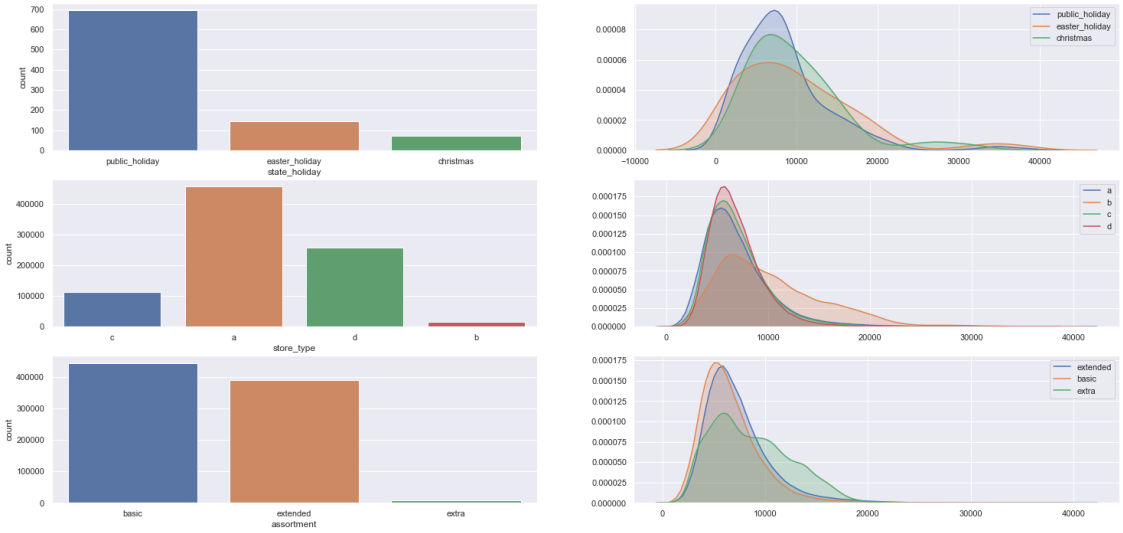


5.1.3 4.1.3. Categorical Variable

[28]:
```
# 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'] == 'extended']['sales'], label='extended',␣
 ↪shade=True )
sns.kdeplot( df4[df4['assortment'] == 'basic']['sales'], label='basic',␣
 ↪shade=True )
sns.kdeplot( df4[df4['assortment'] == 'extra']['sales'], label='extra',␣
 ↪shade=True )
```

[28]: <matplotlib.axes._subplots.AxesSubplot at 0x15bf1af40>



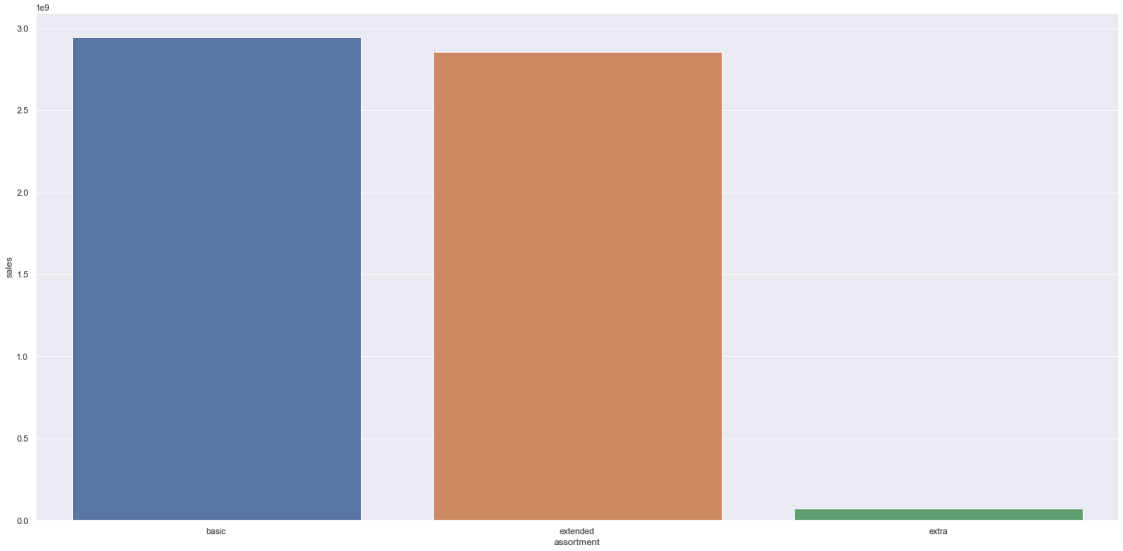
5.2 4.2. Analise Bivariada

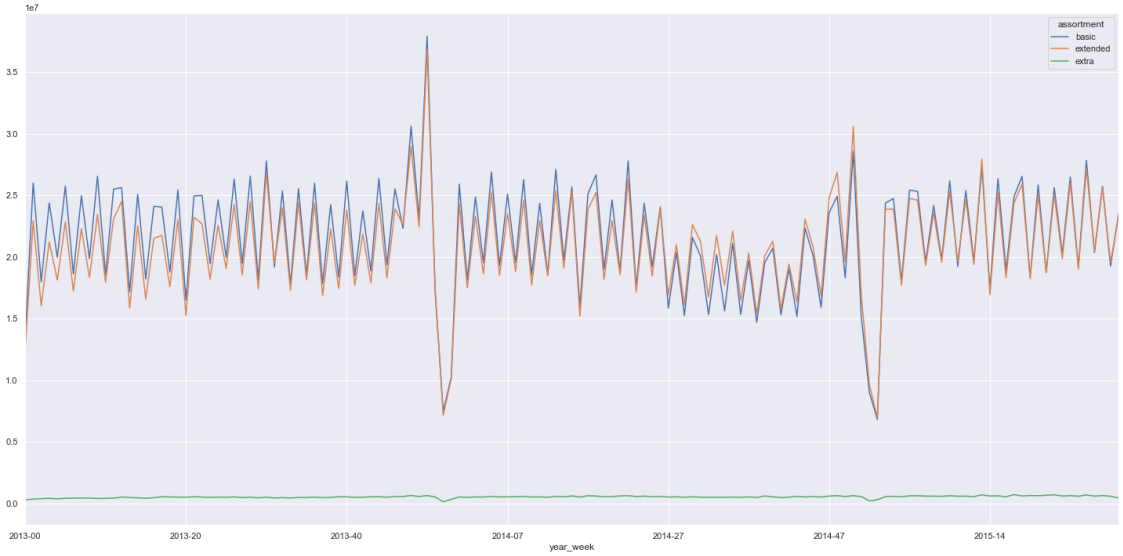
5.2.1 H1. Lojas com maior sortimentos deveriam vender mais.

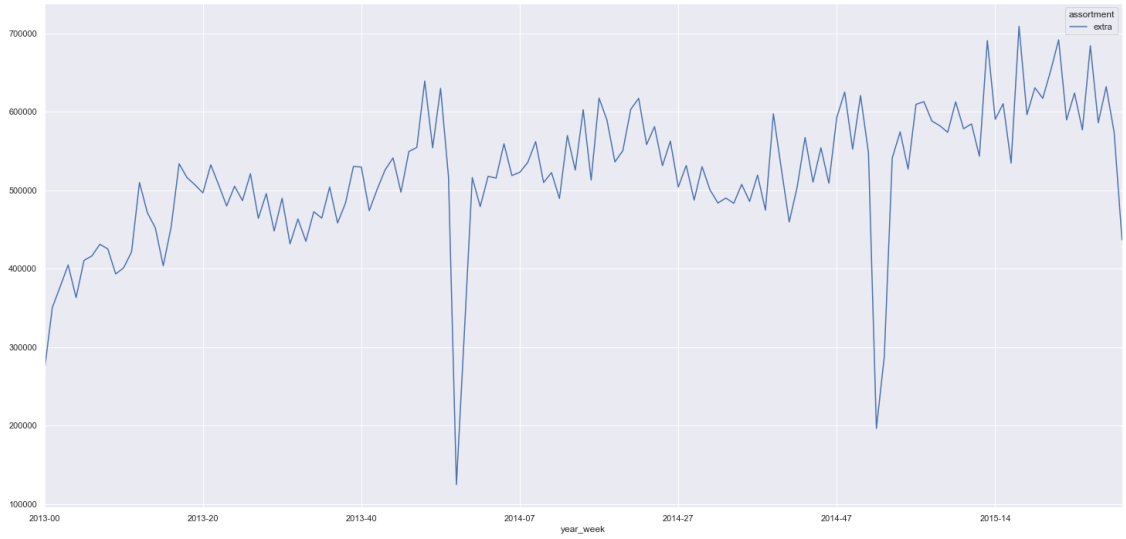
FALSA Lojas com MAIOR SORTIMENTO vendem MENOS.

[29]:
```
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()
```

[29]: <matplotlib.axes._subplots.AxesSubplot at 0x171f91a30>



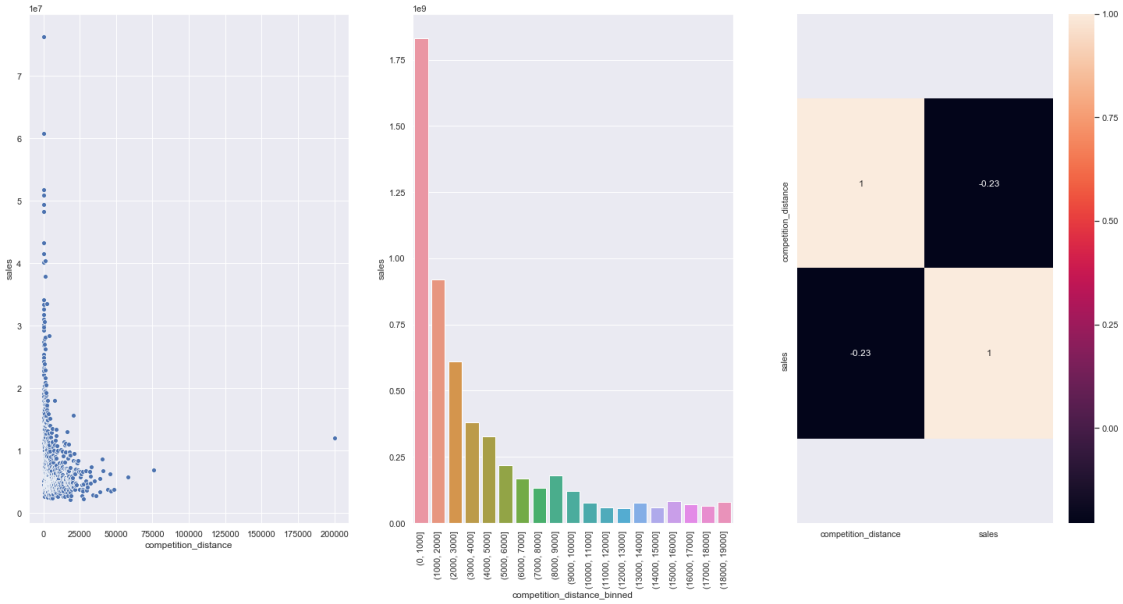




5.2.2 H2. Lojas com competidores mais próximos deveriam vender menos.

FALSA Lojas com COMPETIDORES MAIS PROXIMOS vendem MAIS.

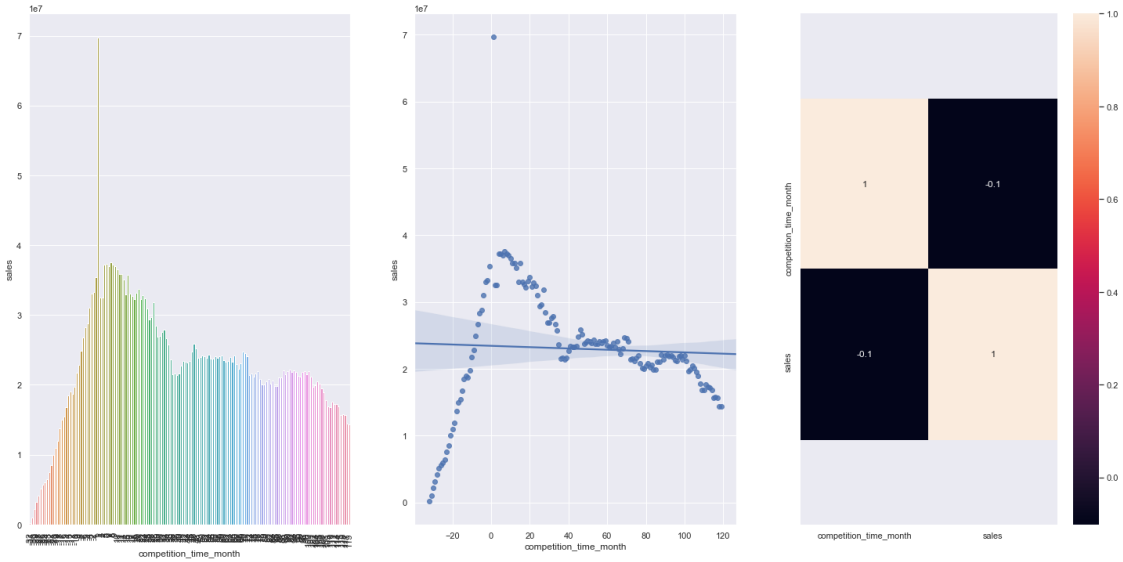
[30]:
```
aux1 = df4[['competition_distance', 'sales']].groupby( 'competition_distance' ).
 ↪sum().reset_index()

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 )
x = sns.heatmap( aux1.corr( method='pearson' ), annot=True );
bottom, top = x.get_ylim()
x.set_ylim( bottom+0.5, top-0.5 );
```



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

FALSE Lojas com COMPETIDORES À MAIS TEMPO vendem MENOS.

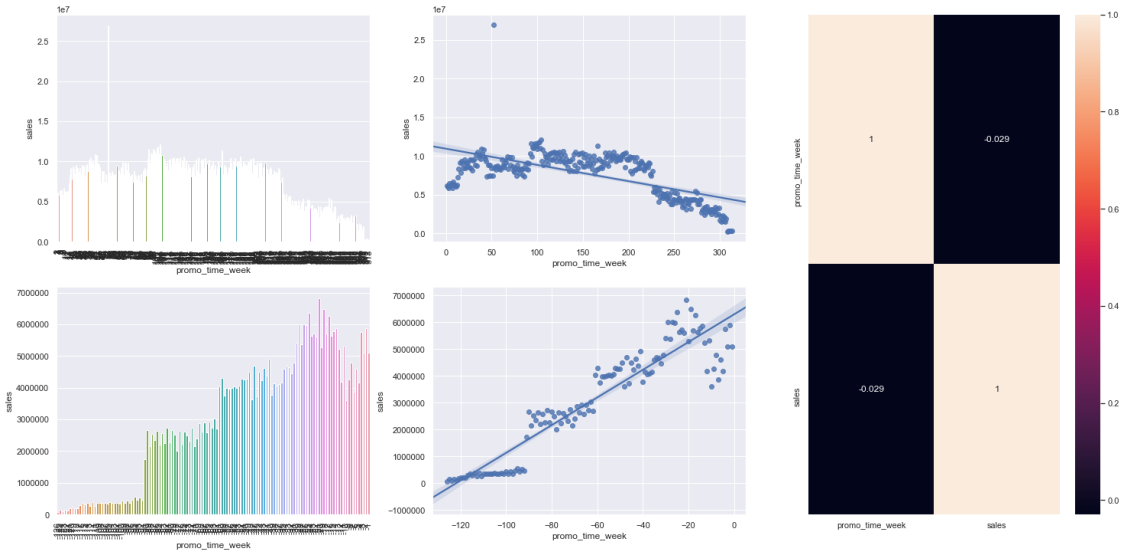
[31]:
```
plt.subplot( 1, 3, 1 )
aux1 = df4[['competition_time_month', 'sales']].groupby(␣
 ↪'competition_time_month' ).sum().reset_index()
aux2 = aux1[( aux1['competition_time_month'] < 120 ) & (␣
 ↪aux1['competition_time_month'] != 0 )]
sns.barplot( x='competition_time_month', y='sales', data=aux2 );
plt.xticks( rotation=90 );

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

plt.subplot( 1, 3, 3 )
x = sns.heatmap( aux1.corr( method='pearson'), annot=True );
bottom, top = x.get_ylim()
x.set_ylim( bottom+0.5, top-0.5);
```



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

[32]:
```
aux1 = df4[['promo_time_week', 'sales']].groupby( 'promo_time_week').sum().
↪reset_index()

grid = GridSpec( 2, 3 )

plt.subplot( grid[0,0] )
aux2 = aux1[aux1['promo_time_week'] > 0] # promo extendido
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] # promo regular
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 H7. Lojas com mais promoções consecutivas deveriam vender mais.

FALSA Lojas com mais promocoes consecutivas vendem menos

[33]:
```
df4[['promo', 'promo2', 'sales']].groupby( ['promo', 'promo2'] ).sum().
↪reset_index()
```

[33]:

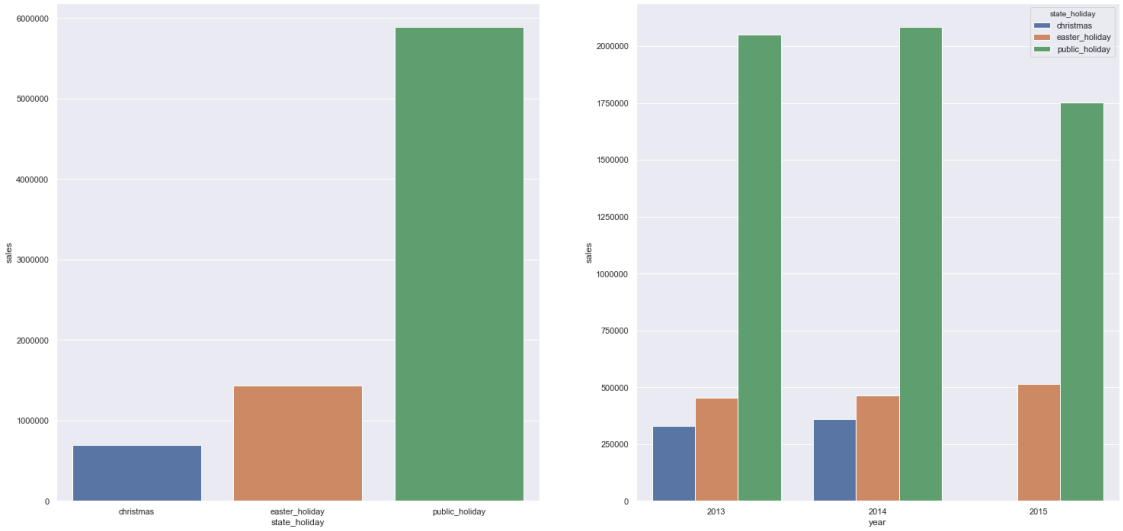
	promo	promo2	sales
0	0	0	1482612096
1	0	1	1289362241
2	1	0	1628930532
3	1	1	1472275754

[34]:
```
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', 'Extendida']);
```

5.2.7 H8. Lojas abertas durante o feriado de Natal deveriam vender mais.

FALSA Lojas abertas durante o feriado do Natal vendem menos.

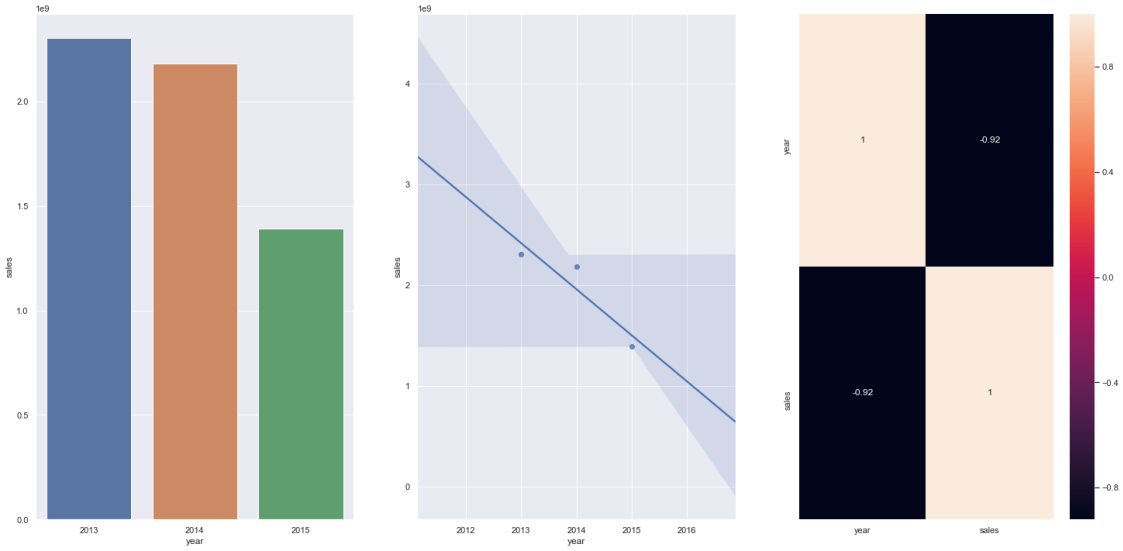
[35]:
```
aux = df4[df4['state_holiday'] != 'regular_day']

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 );
```



5.2.8 H9. Lojas deveriam vender mais ao longo dos anos.

FALSA Lojas vendem menos ao longo dos anos

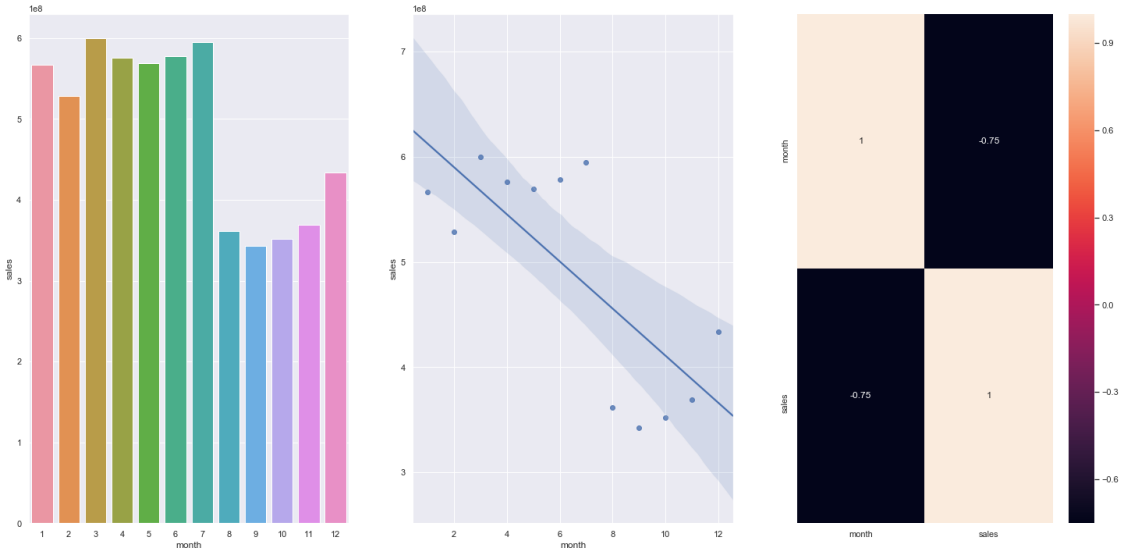
[36]:
```
aux1 = df4[['year', 'sales']].groupby( 'year' ).sum().reset_index()

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 H10. Lojas deveriam vender mais no segundo semestre do ano.

FALSA Lojas vendem menos no segundo semestre do ano

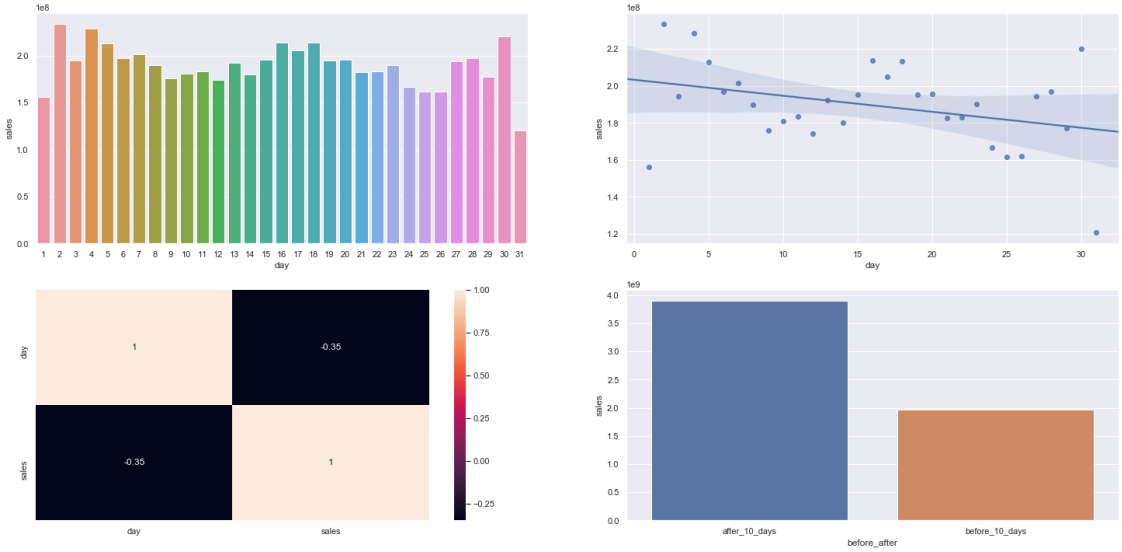
[37]:
```
aux1 = df4[['month', 'sales']].groupby( 'month' ).sum().reset_index()

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 H11. Lojas deveriam vender mais depois do dia 10 de cada mês.

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

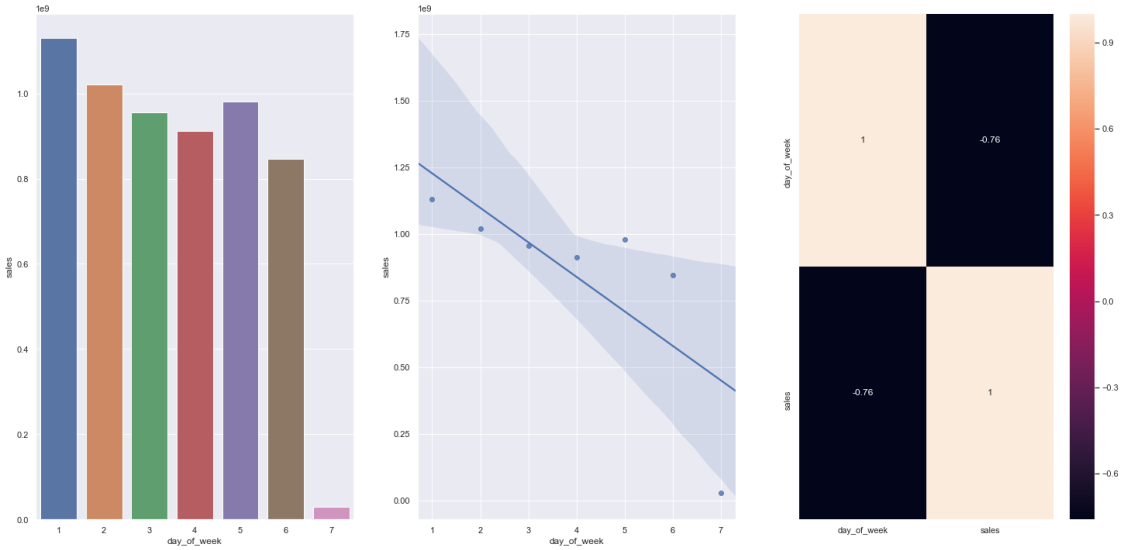
[38]:
```
aux1 = df4[['day', 'sales']].groupby( 'day' ).sum().reset_index()

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 H12. Lojas deveriam vender menos aos finais de semana.

VERDADEIRA Lojas vendem menos nos final de semana

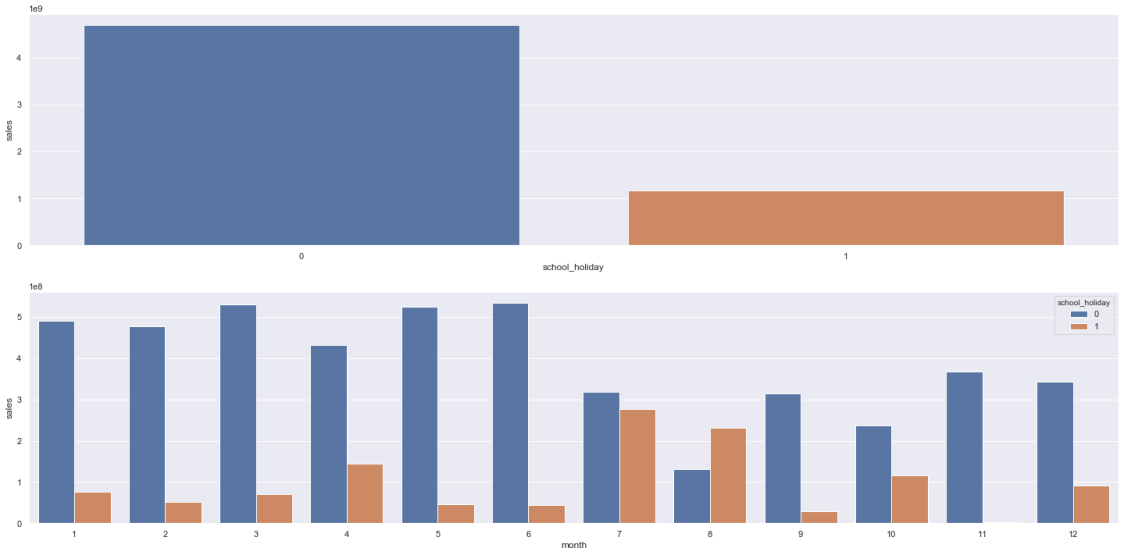
[39]:
```
aux1 = df4[['day_of_week', 'sales']].groupby( 'day_of_week' ).sum().
 ↪reset_index()

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 H13. Lojas deveriam vender menos durante os feriados escolares.

VERDADEIRA Lojas vendem menos durante os feriadso escolares, except os meses de Julho e Agosto.

[40]:
```
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 4.2.1. Resumo das Hipoteses

[41]:
```
from tabulate import tabulate
```

[42]:
```
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' ) )
```

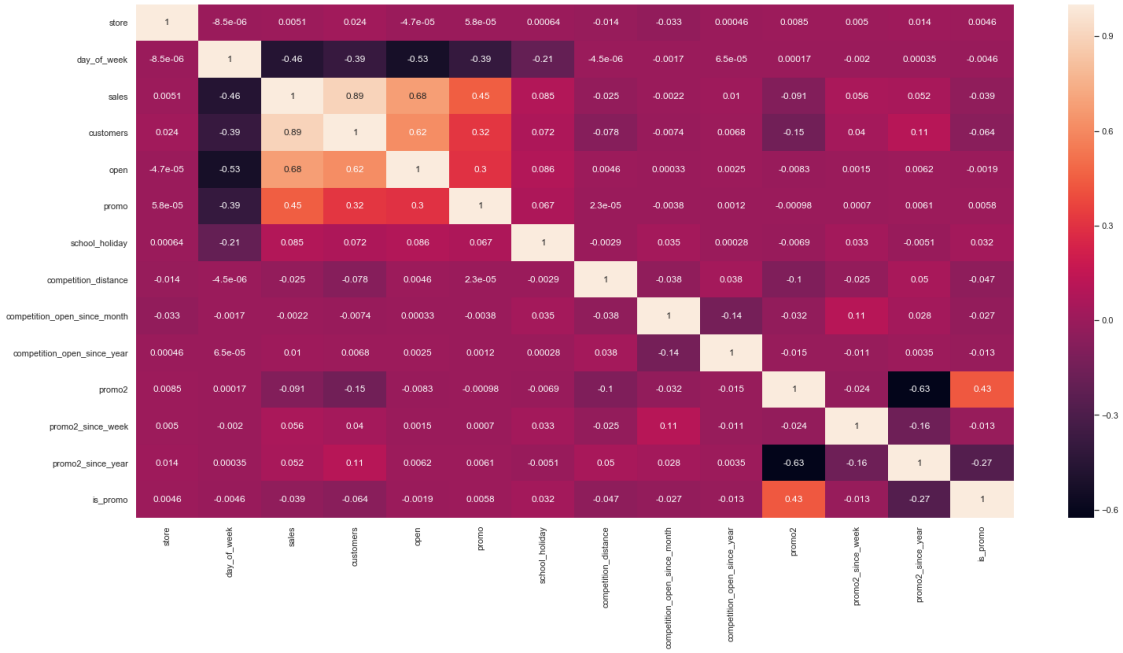
```
Hipoteses    Conclusao    Relevancia
-----------  -----------  ------------
H1           Falsa        Baixa
H2           Falsa        Media
H3           Falsa        Media
H4           Falsa        Baixa
H5           -            -
H7           Falsa        Baixa
H8           Falsa        Media
```

H9	Falsa	Alta
H10	Falsa	Alta
H11	Verdadeira	Alta
H12	Verdadeira	Alta
H13	Verdadeira	Baixa

5.3 4.3. Analise Multivariada

5.3.1 4.3.1. Numerical Attributes

[43]:
```
correlation = num_attributes.corr( method='pearson' )
sns.heatmap( correlation, annot=True );
```

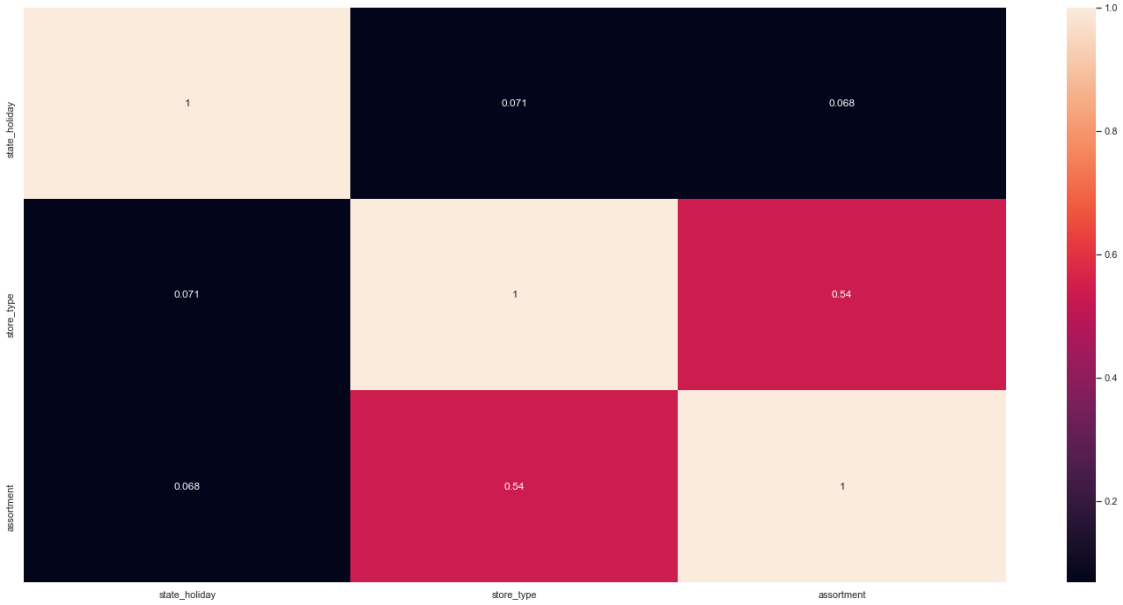


5.3.2 4.3.2. Categorical Attributes

[44]:
```
# 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 = d.set_index( d.columns )

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

[44]: <matplotlib.axes._subplots.AxesSubplot at 0x122d1ad30>



[ ]:

6 5.0. PASSO 05 - DATA PREPARATION

[476]:
```
df5 = df4.copy()
```

6.1 5.1. Normalizacao

[ ]:

6.2 5.2. Rescaling

[477]:
```
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( rs, 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 5.3. Transformacao

6.3.1 5.3.1. Encoding

[475]:
```
# 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 5.3.2. Response Variable Transformation

[48]:
```
df5['sales'] = np.log1p( df5['sales'] )
```

6.3.3 5.3.3. Nature Transformation

[49]:
```
# 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 6.0. PASSO 06 - FEATURE SELECTION

[50]:
```
df6 = df5.copy()
```

7.1 6.1. Split dataframe into training and test dataset

[51]:
```
cols_drop = ['week_of_year', 'day', 'month', 'day_of_week', 'promo_since',␣
↪'competition_since', 'year_week' ]
df6 = df6.drop( cols_drop, axis=1 )
```

[52]:
```
# 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() ) )
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 6.2. Boruta as Feature Selector

[53]:
```
## training and test dataset 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 6.2.1. Best Features from Boruta

[54]:
```
#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()
#
## not selected boruta
#cols_not_selected_boruta = list( np.setdiff1d( X_train_fs.columns,␣
↪cols_selected_boruta ) )
```

7.3 6.3. Manual Feature Selection

[55]:
```
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']

cols_selected_boruta_full = cols_selected_boruta.copy()
cols_selected_boruta_full.extend( feat_to_add )
```

8 7.0. PASSO 07 - MACHINE LEARNING MODELLING

[56]:
```
x_train = X_train[ cols_selected_boruta ]
x_test = X_test[ cols_selected_boruta ]

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

8.1 7.1. Average Model

[57]:
```
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
```

[57]:

	Model Name	MAE	MAPE	RMSE
0	Average Model	1354.800353	0.455051	1835.135542

8.2 7.2. Linear Regression Model

[58]:
```
# 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
```

[58]:

	Model Name	MAE	MAPE	RMSE
0	Linear Regression	1867.089774	0.292694	2671.049215

8.2.1 7.2.1. Linear Regression Model - Cross Validation

[59]:
```
lr_result_cv = cross_validation( x_training, 5, 'Linear Regression', lr,␣
 ↪verbose=False )
lr_result_cv
```

[59]:

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

8.3 7.3. Linear Regression Regularized Model - Lasso

[60]:
```
# 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 - Lasso', np.expm1( y_test ), np.
 ↪expm1( yhat_lrr ) )
lrr_result
```

[60]:

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

8.3.1 7.3.1. Lasso - Cross Validation

[61]:
```
lrr_result_cv = cross_validation( x_training, 5, 'Lasso', lrr, verbose=False )
lrr_result_cv
```

[61]:

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

8.4 7.4. Random Forest Regressor

[62]:
```
# 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
```

[62]:

	Model Name	MAE	MAPE	RMSE
0	Random Forest Regressor	679.622763	0.09996	1011.191561

8.4.1 7.4.1. Random Forest Regressor - Cross Validation

[63]:
```
rf_result_cv = cross_validation( x_training, 5, 'Random Forest Regressor', rf,␣
↪verbose=True )
rf_result_cv
```

```
KFold Number: 5

KFold Number: 4

KFold Number: 3

KFold Number: 2

KFold Number: 1
```

[63]:

	Model Name	MAE CV	MAPE CV	RMSE CV
0	Random Forest Regressor	837.68 +/- 219.1	0.12 +/- 0.02	1256.08 +/- 320.36

8.5 7.5. XGBoost Regressor

[64]:
```
# 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
```

[64]:

	Model Name	MAE	MAPE	RMSE
0	XGBoost Regressor	843.112292	0.122609	1250.952634

8.5.1 7.5.1. XGBoost Regressor - Cross Validation

[65]:
```
xgb_result_cv = cross_validation( x_training, 5, 'XGBoost Regressor',␣
↪model_xgb, verbose=True )
xgb_result_cv
```

```
KFold Number: 5

KFold Number: 4

KFold Number: 3

KFold Number: 2

KFold Number: 1
```

[65]:

	Model Name	MAE CV	MAPE CV	RMSE CV
0	XGBoost Regressor	1030.28 +/- 167.19	0.14 +/- 0.02	1478.26 +/- 229.79

8.6 7.6. Compare Model's Performance

8.6.1 7.6.1. Single Performance

[66]:
```
modelling_result = pd.concat( [baseline_result, lr_result, lrr_result,␣
↪rf_result, xgb_result] )
modelling_result.sort_values( 'RMSE' )
```

[66]:

	Model Name	MAE	MAPE	RMSE
0	Random Forest Regressor	679.622763	0.099960	1011.191561
0	XGBoost Regressor	843.112292	0.122609	1250.952634
0	Average Model	1354.800353	0.455051	1835.135542
0	Linear Regression	1867.089774	0.292694	2671.049215
0	Linear Regression - Lasso	1891.704881	0.289106	2744.451737

8.6.2 7.6.2. Real Performance - Cross Validation

[67]:
```
modelling_result_cv = pd.concat( [lr_result_cv, lrr_result_cv, rf_result_cv,␣
↪xgb_result_cv] )
modelling_result_cv
```

[67]:
```
                Model Name              MAE CV        MAPE CV              RMSE
CV
0        Linear Regression  2081.73 +/- 295.63   0.3 +/- 0.02  2952.52 +/-
468.37
0                    Lasso   2116.38 +/- 341.5  0.29 +/- 0.01  3057.75 +/-
504.26
0  Random Forest Regressor    837.68 +/- 219.1  0.12 +/- 0.02  1256.08 +/-
320.36
0        XGBoost Regressor  1030.28 +/- 167.19  0.14 +/- 0.02  1478.26 +/-
229.79
```

9 8.0. PASSO 08 - HYPERPARAMETER FINE TUNING

9.1 8.1. Random Search

[68]:
```
#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
```

[69]:
```
#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=True )
#    final_result = pd.concat( [final_result, result] )
#
#final_result
```

[70]:
```
#final_result
```

9.2 8.2. Final Model

[71]:
```
param_tuned = {
    'n_estimators': 3000,
    'eta': 0.03,
    'max_depth': 5,
    'subsample': 0.7,
    'colsample_bytree': 0.7,
    'min_child_weight': 3
        }
```

[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['subsample'],
                                   ␣
 ↪colsample_bytee=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	664.974996	0.097529	957.774225

[73]:
```
mpe = mean_percentage_error( np.expm1( y_test ), np.expm1( yhat_xgb_tuned ) )
mpe
```

[73]:
```
-0.0035453341443739675
```

10 9.0. PASSO 09 - TRADUCAO E INTERPRETACAO DO ERRO

[435]:
```
df9 = X_test[ cols_selected_boruta_full ]

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

10.1 9.1. Business Performance

[443]:
```
# 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']]
```

[446]:
```
df92.sort_values( 'MAPE', ascending=False ).head()
```

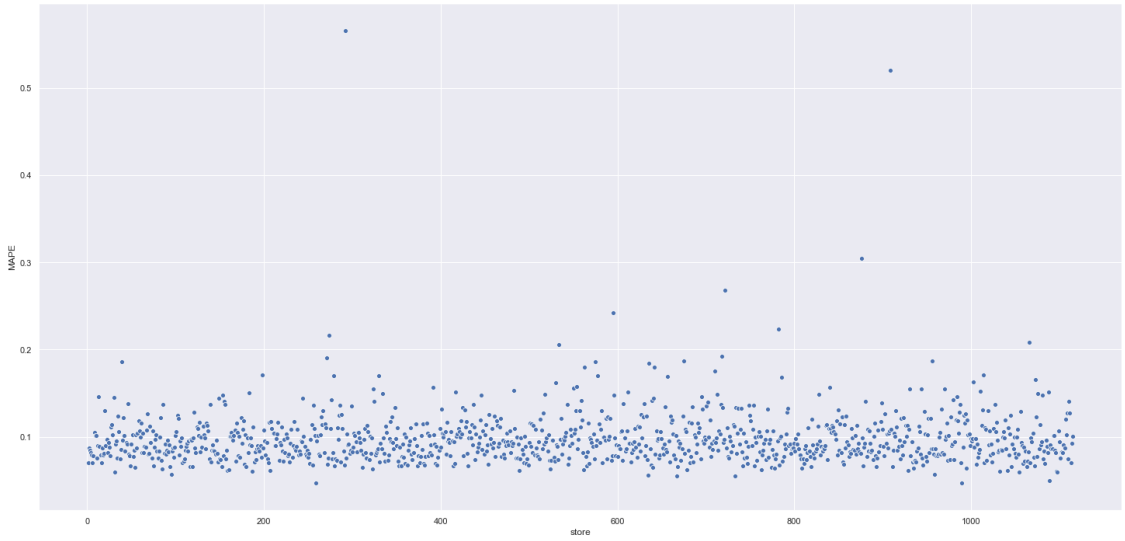
[446]:

	store	predictions	worst_scenario	best_scenario	MAE	MAPE
291	292	104033.078125	100714.973723	107351.182527	3318.104402	0.565828
908	909	238233.875000	230573.337190	245894.412810	7660.537810	0.520433
875	876	203030.156250	199110.952435	206949.360065	3919.203815	0.305099
721	722	353005.781250	351013.625224	354997.937276	1992.156026	
594	595	400883.625000	397415.263170	404351.986830	3468.361830	

```
721  0.268338
594  0.242192
```

[448]:
```
sns.scatterplot( x='store', y='MAPE', data=df92 )
```

[448]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x16a890280>
```



10.2 9.2. Total Performance

[455]:
```
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
```

[455]:

	Scenario	Values
0	predictions	R$285,860,497.77
1	worst_scenario	R$285,115,015.71
2	best_scenario	R$286,605,979.84

10.3 9.3. Machine Learning Performance

[457]:
```
df9['error'] = df9['sales'] - df9['predictions']
df9['error_rate'] = df9['predictions'] / df9['sales']
```

[459]:
```
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'] )
```

[459]: <matplotlib.axes._subplots.AxesSubplot at 0x1689cf700>

11 10.0. PASSO 10 - DEPLOY MODEL TO PRODUCTION

[ ]:
```
# Save Trained Model
pickle.dump( model_xgb_tuned, open( '/Users/meigarom/repos/
↪DataScience_Em_Producao/model/model_rossmann.pkl', 'wb' ) )
```

11.1 10.1. Rossmann Class

[ ]:
```
```

[17]:
```
import pickle
import inflection
import pandas as pd
import numpy as np
import math
import datetime
```

```
class Rossmann( object ):
    def __init__( self ):
        self.home_path='/Users/meigarom/repos/DataScience_Em_Producao/'
        self.competition_distance_scaler   = pickle.load( open( self.home_path␣
↪+ 'parameter/competition_distance_scaler.pkl', 'rb') )
        self.competition_time_month_scaler = pickle.load( open( self.home_path␣
↪+ 'parameter/competition_time_month_scaler.pkl', 'rb') )
        self.promo_time_week_scaler        = pickle.load( open( self.home_path␣
↪+ 'parameter/promo_time_week_scaler.pkl', 'rb') )
        self.year_scaler                   = pickle.load( open( self.home_path␣
↪+ 'parameter/year_scaler.pkl', 'rb') )
        self.store_type_scaler             = pickle.load( open( self.home_path␣
↪+ 'parameter/store_type_scaler.pkl', 'rb') )


    def data_cleaning( self, df1 ):

        ## 1.1. Rename Columns
        cols_old = ['Store', 'DayOfWeek', 'Date', '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

        ## 1.3. Data Types
        df1['date'] = pd.to_datetime( df1['date'] )

        ## 1.5. 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: 'Fev',  3: 'Mar',  4: 'Apr',  5: 'May',  6:␣
↪'Jun',  7: 'Jul',  8: 'Aug',  9: 'Sep',  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 )

        ## 1.6. Change Data Types
        # competiton
        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 ):

        # 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'] = df2['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' )

        # 3.0. PASSO 03 - FILTRAGEM DE VARIÁVEIS
        ## 3.1. Filtragem das Linhas
        df2 = df2[df2['open'] != 0]

        ## 3.2. Selecao das Colunas
        cols_drop = ['open', 'promo_interval', 'month_map']
        df2 = df2.drop( cols_drop, axis=1 )

        return df2


    def data_preparation( self, df5 ):
```

```
        ## 5.2. Rescaling
        # competition distance
        df5['competition_distance'] = self.competition_distance_scaler.
↪fit_transform( df5[['competition_distance']].values )

        # competition time month
        df5['competition_time_month'] = self.competition_time_month_scaler.
↪fit_transform( df5[['competition_time_month']].values )

        # promo time week
        df5['promo_time_week'] = self.promo_time_week_scaler.fit_transform(␣
↪df5[['promo_time_week']].values )

        # year
        df5['year'] = self.year_scaler.fit_transform( df5[['year']].values )

        ### 5.3.1. Encoding
        # state_holiday - One Hot Encoding
        df5 = pd.get_dummies( df5, prefix=['state_holiday'],␣
↪columns=['state_holiday'] )

        # store_type - Label Encoding
        df5['store_type'] = self.store_type_scaler.fit_transform(␣
↪df5['store_type'] )

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


        ### 5.3.3. Nature Transformation
        # 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 ) ) )


        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 df5[ cols_selected ]


    def get_prediction( self, model, original_data, test_data ):
        # prediction
        pred = model.predict( test_data )

        # join pred into the original data
        original_data['prediction'] = np.expm1( pred )

        return original_data.to_json( orient='records', date_format='iso' )
```

11.2 10.2. API Handler

[18]:
```
import pickle
import pandas as pd
from flask              import Flask, request, Response
from rossmann.Rossmann import Rossmann

# loading model
model = pickle.load( open( '/Users/meigarom/repos/DataScience_Em_Producao/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 example
            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


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

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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-18-202fd353a2d0> in <module>
      2 import pandas as pd
      3 from flask             import Flask, request, Response
----> 4 from rossmann.Rossmann import Rossmann
      5
      6 # loading model

ModuleNotFoundError: No module named 'rossmann'
```

11.3 10.3. API Tester

[ ]:
```
# loading test dataset
df10 = pd.read_csv( '/Users/meigarom/repos/DataScience_Em_Producao/data/test.
 ↪csv' )
```

[ ]:
```
# merge test dataset + store
df_test = pd.merge( df10, df_store_raw, how='left', on='Store' )

# choose store for prediction
df_test = df_test[df_test['Store'].isin( [20, 23, 22] )]

# remove closed days
df_test = df_test[df_test['Open'] != 0]
df_test = df_test[~df_test['Open'].isnull()]
df_test = df_test.drop( 'Id', axis=1 )
```

[ ]:
```
# convert Dataframe to json
data = json.dumps( df_test.to_dict( orient='records' ) )
```

[ ]:
```
# API Call
#url = 'http://0.0.0.0:5000/rossmann/predict'
url = 'https://rossmann-model-test.herokuapp.com/rossmann/predict'
header = {'Content-type': 'application/json' }
data = data

r = requests.post( url, data=data, headers=header )
print( 'Status Code {}'.format( r.status_code ) )
```

[ ]:
```
d1 = pd.DataFrame( r.json(), columns=r.json()[0].keys() )
```

[ ]:
```
d2 = d1[['store', 'prediction']].groupby( 'store' ).sum().reset_index()

for i in range( len( d2 ) ):
    print( 'Store Number {} will sell R${:,.2f} in the next 6 weeks'.format(
            d2.loc[i, 'store'],
            d2.loc[i, 'prediction'] ) )
```