Storytelling_Documentation_Rossmann

February 25, 2022

[63]:
```
Image("../img/ross.png")
```

[63]:

Rossmann History.

As lojas da Rossmann são uma rede de farmacias localizadas na europa, principalmente na Alemanha, com aproximadamente 56,200 funcionários e com mais de 4000 lojas. A empresa foi fundada por Dirk Rossmann com a sede em Burgwedel próximo de Hanover na Alemanha. ~ Wiki.

1 Contexto de Negócio

Diretor Financeiro da Rossmann em uma reunião mensal de resultados pediu a todos os gerentes de loja uma previsão de vendas para as próximas seis semanas para futuras reformas das lojas. 1. Não vai ser utilizado o E-Commerce da Rs para a previsão de vendas. 2. O modelo de negócio das lojas da Rossmann é um Varejo, logo existem vários fatores que influenciam as vendas, desdê a qualidade física da loja (pintura, detalhes...) até marketing relacionado a loja, muitas hipóteses!

Modelo de Negócio de Farmácia A Rossmann esta presente com um E-Commerce e com lojas físicas disponível para venda de diversos items, desdê maquiagens e é claro, remédios. Sobre as lojas físicas esta caracterizado em uma rede de farmácias, que estão espalhadas por diversas partes da

Europa, Podendo assim selecionar regiões com menor risco de competidores e espalhando a marca por toda a Europa.

Oque é uma Rede de Farmácia? Basicamente, a rede começa com uma loja aberta chamada matriz, a próxima loja aberta é a filial. Existem dois tipos de rede, a cadeia associada é quando várias farmácias com proprietários diferentes se juntam e cadeia privada que começa com a empresa matriz e demais filiais e tem um proprietário geral, em outras palavras o dinheiro entra seguindo um modelo de negócio do tipo Varejo.

'Primeiras Hipóteses' - **Market Size**: Todas as pessoas maiores de 18 anos com preferência em pessoas mais idosas. - **Marketing Channels**: Rossmann E-Commerce e Lojas. - **Principal Metrics**: - Channel Offline: Vendas em lojas Físicas. - Recency: Compras ao longo do tempo. - Frequency: Frequencia de vendas das lojas para uma previsão mais acurada. - Market Share: Competidores Proximos.

1. Os consumidores mais velhos compram nas lojas físicas com mais frequencia que no E-commerce ou nos competidores ?
2. Qual é o Investimento em marketing em relação a lojas físicas em comparação com o E-Commerce ?
3. Quais são os novos produtos que fazem os clientes comprarem nas lojas Rossmann em vez das lojas concorrentes ?
4. Como essas lojas se comportam em termos de recebimento de novas mercadorias para vender a novos clientes ?
5. Os produtos vendidos tem realmente um fácil acesso ?
6. Como são distribuidos os preços desses produtos em relação a localidade da loja física ?
7. Como os produtos da rossmann estão sendo avaliados ?
8. Como é o processo de compra para esses clientes ?
9. Quantos desses clientes que compraram uma vez, vão voltar a comprar novamente ?
10. Quanto custa um clinete novo em relação as lojas físicas ?
11. Quais são as principais marcas parceiras da Rossmann ? ...

Dataset: https://www.kaggle.com/c/rossmann-store-sales

`Informações dos Dados (Retirados do site do Kaggle)`

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

year for that store

1.1 Documentação Resumo

Descrição Estatística Renomear as colunas para snake_case, nesse formato unitário as colunas estão em um formato mais fácil para manipulação, seleção e desenvolvimento da solução, dimensão dos dados é uma forma visual de contar a quantidade de dados para checar que os requisitos do sistema suportam tais quantidades de dados, checar os dados serve basicamente para localizar eventuais erros de consulta a bancos de dados ou até inputs que estavam sendo armazenados até então, já a parte de checagem e preenchimento de valores Na vem muito da consulta da base de dados para a coleta dos dados, eventuais inputs no sistemas ou ate mesmo erros de digitação e armazenamento desses dados, após toda essa transformação vem a parte de mudar os tipos dos dados para uma melhor compreensão e realmente manter a natureza dos dados, e por fim um resumo estatístico para analisar possíveis *Outliers* ou até inputs errados no sistema, sempre fazendo a pergunta se esse dado realmente deveria estar sendo armazenado dessa forma.

Engenharia de Features Desenvolver novas features baseadas no negócio, em hipóteses e é claro, features em relação à o tempo até alguma coisa, e features derivadas a partir do tempo e datas. Também a limpeza e seleção de features que não vão ser relevantes para o desenvolvimento da solução e features que não vão estar disponíveis na produção.

Análise Exploratória de Dados Etapa onde o objetivo é analisar profundamente os dados buscando validar hipóteses e gerar novos insight's para o negócio, nessa análise profunda outro objetivo é localizar correlações, análise das variáveis categóricas, variáveis que "separam" ou aumentam o fenômeno de vendas que é o principal objetivo desse projeto.

Preparação dos Dados Objetivo é preparar os dados para os modelos matemáticos aprenderem o comportamento e assim conseguirem generalizar para o futuro o aprendizado de dados passados, pois a maioria desses modelos não entende uma variável categóricas ou outros dão uma importância muito maior a valores altos, por exemplo comparar a distância em quilometros com a idade dos consumidores. Probelma com vazamento de dados está presente nesse projeto, logo foram feitas duas preparações e separações em treino e teste para análise.

Seleção de Variáveis Selecionar possíveis variáveis que não ajudam a expandir o conhecimento desses modelos, apenas são variáveis correlacionadas que apenas aumentam a dimensionalidade causando o fenômeno da maldição da dimensionalidade, porém não se pode confiar totalmente nesses modelos e sempre trabalhar em conjunto com o conhecimento de negócio e o conhecimento adquirido da análise exploratória dos dados. Sendo a variável promo a mais importante seguindo a importância pelo XGBoost.

Algoritimos de Ml São os modelos que irão generalizar para o futuro o aprendizado de toda essa preparação e análise dos dados, com o cross validation para recolher a melhor e verdadeira performace dos modelos. 1. Support Vector Regression (Modelo de Regressão com Kernel). 2. XGBoost (Modelo de Boosting baseado em Arvores). 3. Random Forest. 5. Neural Network One Hidden Layer n Units.

Performace do Algoritimo em Resultados Com o Algoritimo XGBoost:

2 Desafio

Previsão Acurada de cada uma das n lojas da Rossmann, onde o Diretor pode checar essas previsões no celular com a aplicação do Telegram e acessar essas previsões em um aplicativo executável em seu computador caso não estiver utilizando o celular.

[28]:
```
anos = set( [int(x[:4]) for x in df_raw['Date']] )
print(f"Total de Lojas Físicas: {len(df_raw['Store'].unique())}\nNos anos de:␣
 ↪{anos}")
```

```
Total de Lojas Físicas: 1115
Nos anos de: {2013, 2014, 2015}
```

2.1 0.0. Imports & Functions

[1]:
```
import pickle
import warnings
import datetime
import inflection
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as api
import matplotlib.gridspec as gridspec

from random           import sample
from scipy            import stats
from sklearn.svm      import SVR
from IPython.display  import Image
from matplotlib       import pyplot as plt
from sklearn          import preprocessing as pp
from sklearn.ensemble import RandomForestRegressor
from Utils            import metricsAndPlots as mkk
from xgboost          import XGBRegressor, plot_importance

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

warnings.filterwarnings('ignore')
mp = mkk()
```

2.2 0.2. Loading Data

[16]:
```
df_store = pd.read_csv("../data/store.csv", low_memory=False )
df_train = pd.read_csv("../data/train.csv", low_memory=False )
```

```
df_raw = pd.merge( df_train, df_store, on="Store", how="left" )
```

3 Desenvolvimento da Solução

3.1 1.0. Statistical Description

[17]:
```
df1 = df_raw.copy()
```

3.1.1 1.1. Rename Columns

[18]:
```
df1.columns = [inflection.underscore( p ) for p in df1.columns.tolist()]
```

3.1.2 1.2. Data Dimension

[19]:
```
print(f'Number of Rows: {df1.shape[0]}' )
print(f'Number of Columns: {df1.shape[1]}' )
```

```
Number of Rows: 1017209
Number of Columns: 18
```

3.1.3 1.3. Data Types

[20]:
```
df1["date"] = pd.to_datetime( df1["date"] )

df1.dtypes
```

[20]:
```
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
```

3.1.4 1.4. Check Na

[8]:
```
df1.isna().sum()
```

[8]:
```
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
```

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

[9]:
```
store                         0.000000
day_of_week                   0.000000
date                          0.000000
sales                         0.000000
customers                     0.000000
open                          0.000000
promo                         0.000000
state_holiday                 0.000000
school_holiday                0.000000
store_type                    0.000000
assortment                    0.000000
competition_distance          0.002597
competition_open_since_month  0.317878
competition_open_since_year   0.317878
promo2                        0.000000
promo2_since_week             0.499436
promo2_since_year             0.499436
promo_interval                0.499436
dtype: float64
```

3.1.5 1.5. Fillout Na

Fillna Assumptions (1° Cicle) - Competition Distance: If don't have close distance competition, dont have competition or competitor is too far. - Competition Open Since Month: Using the Month of Date Column. - Competition Open Since Year: Using the Year of Date Column. - Promo2 Since Week: If shop dont started promo 2 extension then 0. - Promo2 Since Year: If shop dont started promo 2 extension then 0. - Promo Interval: If shop dont started promo 2 extension then 0. - Is Promo: New Column to check if current sale is in promo.

[21]:
```
df1["competition_distance"]         = df1["competition_distance"].fillna(␣
↪200000 )
df1["competition_open_since_month"] = df1.apply( lambda x: x["date"].month if␣
↪pd.isnull( x["competition_open_since_month"] ) else␣
↪x["competition_open_since_month"], axis=1 )
df1["competition_open_since_year"]  = df1.apply( lambda x: x["date"].year if pd.
↪isnull( x["competition_open_since_year"] ) else␣
↪x["competition_open_since_year"], axis=1 )
df1["promo2_since_week"].fillna( 0, inplace=True )
df1["promo2_since_year"].fillna( 0, inplace=True )

month = {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 )

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

3.1.6 1.6. Change Dtypes

[22]:
```
df1["promo2_since_week"] = df1["promo2_since_week"].astype('int64')
df1["promo2_since_year"] = df1["promo2_since_year"].astype('int64')
df1["competition_distance"] = df1["competition_distance"].astype('int64')
df1["competition_open_since_year"] = df1["competition_open_since_year"].
↪astype('int64')
df1["competition_open_since_month"] = df1["competition_open_since_month"].
↪astype('int64')
```

3.1.7 1.7. Descriptive Statistical

1.7.1. Num Attributes

[73]:
```
metrics(df1, stats=True)
```

[73]:

	att	min	max	range	mean	\
0	store	1.0	1115.0	1114.0	558.429727	
1	day_of_week	1.0	7.0	6.0	3.998341	

2	sales	0.0	41551.0	41551.0	5773.818972
3	customers	0.0	7388.0	7388.0	633.145946
4	open	0.0	1.0	1.0	0.830107
5	promo	0.0	1.0	1.0	0.381515
6	school_holiday	0.0	1.0	1.0	0.178647
7	competition_distance	20.0	200000.0	199980.0	5935.442677
8	competition_open_since_month	1.0	12.0	11.0	6.786849
9	competition_open_since_year	1900.0	2015.0	115.0	2010.324840
10	promo2	0.0	1.0	1.0	0.500564
11	promo2_since_week	0.0	50.0	50.0	11.647665
12	promo2_since_year	0.0	2015.0	2015.0	1007.010608
13	is_promo	0.0	1.0	1.0	0.171835

	median	std	skew	kurtosis
0	558.0	321.908493	-0.000955	-1.200524
1	4.0	1.997390	0.001593	-1.246873
2	5744.0	3849.924283	0.641460	1.778375
3	609.0	464.411506	1.598650	7.091773
4	1.0	0.375539	-1.758045	1.090723
5	0.0	0.485758	0.487838	-1.762018
6	0.0	0.383056	1.677842	0.815154
7	2330.0	12547.646829	10.242344	147.789712
8	7.0	3.311085	-0.042076	-1.232607
9	2012.0	5.515591	-7.235657	124.071304
10	1.0	0.500000	-0.002255	-1.999999
11	1.0	15.323921	1.003390	-0.498322
12	2009.0	1005.876436	-0.002251	-1.999993
13	0.0	0.377237	1.739838	1.027039

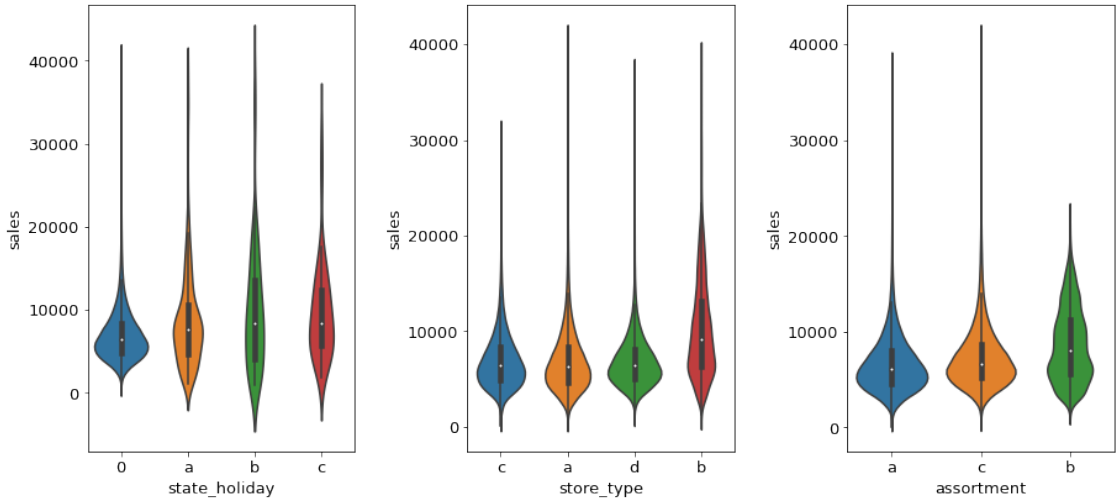
1.7.1. Catt Attributes

[67]:
```
pd.DataFrame( cat_att.apply( lambda x: x.unique().shape[0] )[:3] ).T
```

[67]:

	state_holiday	store_type	assortment
0	4	4	3

[159]:
```
df_aux = df1[(df1["state_holiday"] != 0) & (df1["sales"] > 0)]
for i in zip( range(1, 4), ["state_holiday", "store_type", "assortment"] ):
    plt.subplot( 1, 3, i[0] )
    plt.tight_layout(w_pad=2.0, h_pad=2.0)
    sns.violinplot( x=i[1], y='sales', data=df_aux )
```

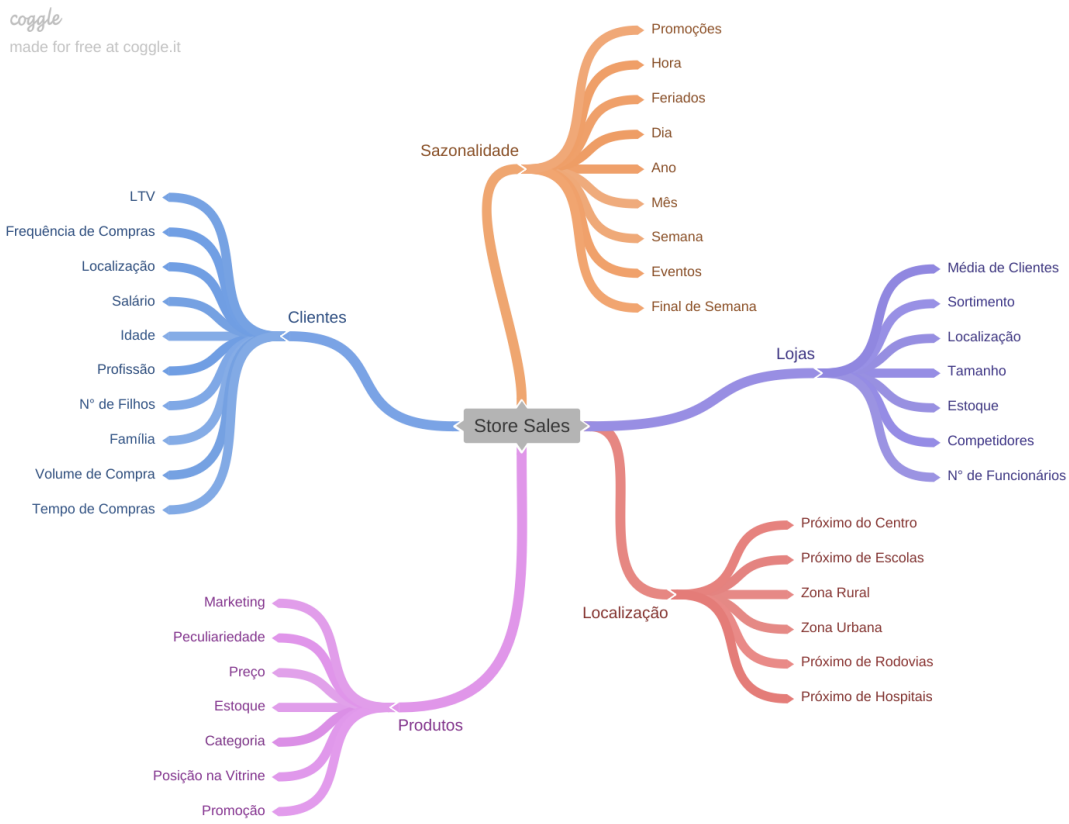


3.2 2.0. Feature Engineering

3.2.1 2.1. Mind Map

[79]:
```
Image('../img/mind.png')
```

[79]:



3.2.2 2.2. Hypothesis List

2.2.1. Shop Hypothesis **1.** Lojas com maior quadro de funcionários deverian vender mais.

2. Lojas com maior estoque deveriam vender mais.

3. Lojas com maior porte deveriam vender mais.

4. Lojas com menor porte deveriam vender menos.

5. Lojas com maior sortimento deveriam vender mais.

2.2.2. Product Hypothesis **1.** Lojas que investem mais em marketing deveriam vender mais.

2. Lojas que expoem mais o produto nas vitrines deveriam vender mais.

3. Lojas que tem um preço menor deveriam vender mais.

4. Lojas que tem mais variedades de produtos deveriam vender mais.

2.2.3. Time Hypothesis **1.** Lojas que abrem em feriados natalinos deveriam vender mais.

2. Lojas que abrem no finla de semana deveriam vender mais.

3. Lojas que entram mais em feriados deveriam vender menos.

2.2.4. Final Hypothesis List **1.** Lojas com maior sortimento deveriam vender mais.

2. Lojas com competidores mais próximos deveriam vender menos.

4. Lojas com competidores a mais tempo deveriam vender mais.

5. Lojas com promoções ativas a mais tempo deveriam vender mais.

6. 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 nos finais de semana.

13. Lojas deveriam vender menos nos feriados escolares.

3.2.3 2.3. Feature Engineering

[23]:
```
df2 = df1.copy()
```

[24]:
```
# Year
df2['year'] = df2["date"].dt.year

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

# Week of Year
df2['week_of_year'] = df2["date"].dt.weekofyear

# Year Week
df2['year_week'] = df2["date"].dt.strftime( '%Y-%W' )

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

# 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 )

# Competition Time Month
df2['competition_time_month'] = ((df2["date"] - df2["competition_since"] ) /␣
 ↪30).apply( lambda x: x.days ).astype('int64')

# 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 ) if x !=␣
 ↪'0-0' else x.replace('0-0', '0') )
df2['promo_time_week'] = ((df2['date'] - df2[~pd.notnull(df2['promo_since'].str.
 ↪contains('0-0'))]['promo_since'])/7).apply( lambda x: x.days ).fillna(0).
 ↪astype('int64')

# Assortment
df2['assortment'] = df2["assortment"].map({"a":"basic", "b":"extra", "c":
 ↪"extended"})

# State Holiday
df2['state_holiday'] = df2['state_holiday'].map({'a':'public_holiday', 'b':
 ↪'easter_holiday', 'c':'christmas', '0':'regular_day'})
```

3.3 3.0. Data Filtering

[25]:
```
df3 = df2.copy()
```

3.3.1 3.1. Row Features

[26]:
```
df3[(df3['open'] == 0) & (df3['sales'] > 0)]
```

[26]:
```
Empty DataFrame
Columns: [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, week_of_year, year_week, day, competition_since, competition_time_month,
promo_since, promo_time_week]
Index: []

[0 rows x 29 columns]
```

[27]:
```
df3 = df3[(df3['open'] != 0) & (df3['sales'] > 0)]
```

3.3.2 3.2. Columns Selection

[28]:
```
df3 = df3.drop( ['customers', 'open', 'month_map', 'promo_interval'], axis=1 )
df3 = df3.reset_index( drop=True )

df3.to_csv("../data_backup/df3.csv")
```

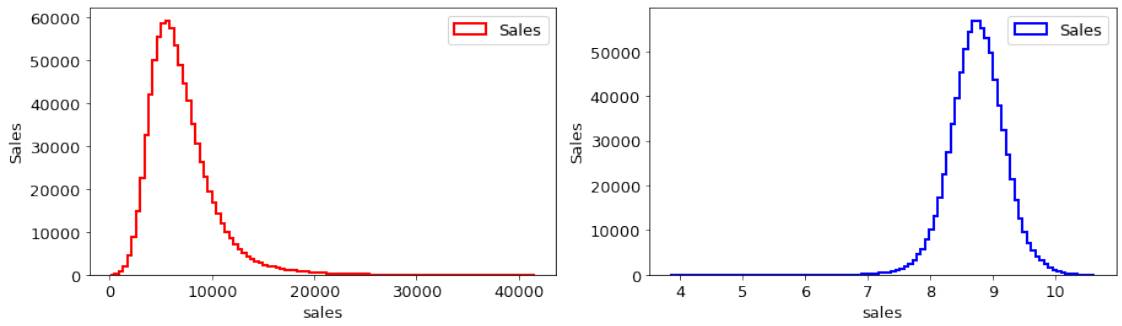
3.4 4.0. Exploratory Data Analysis

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

3.4.1 4.1. Univariable Analysis

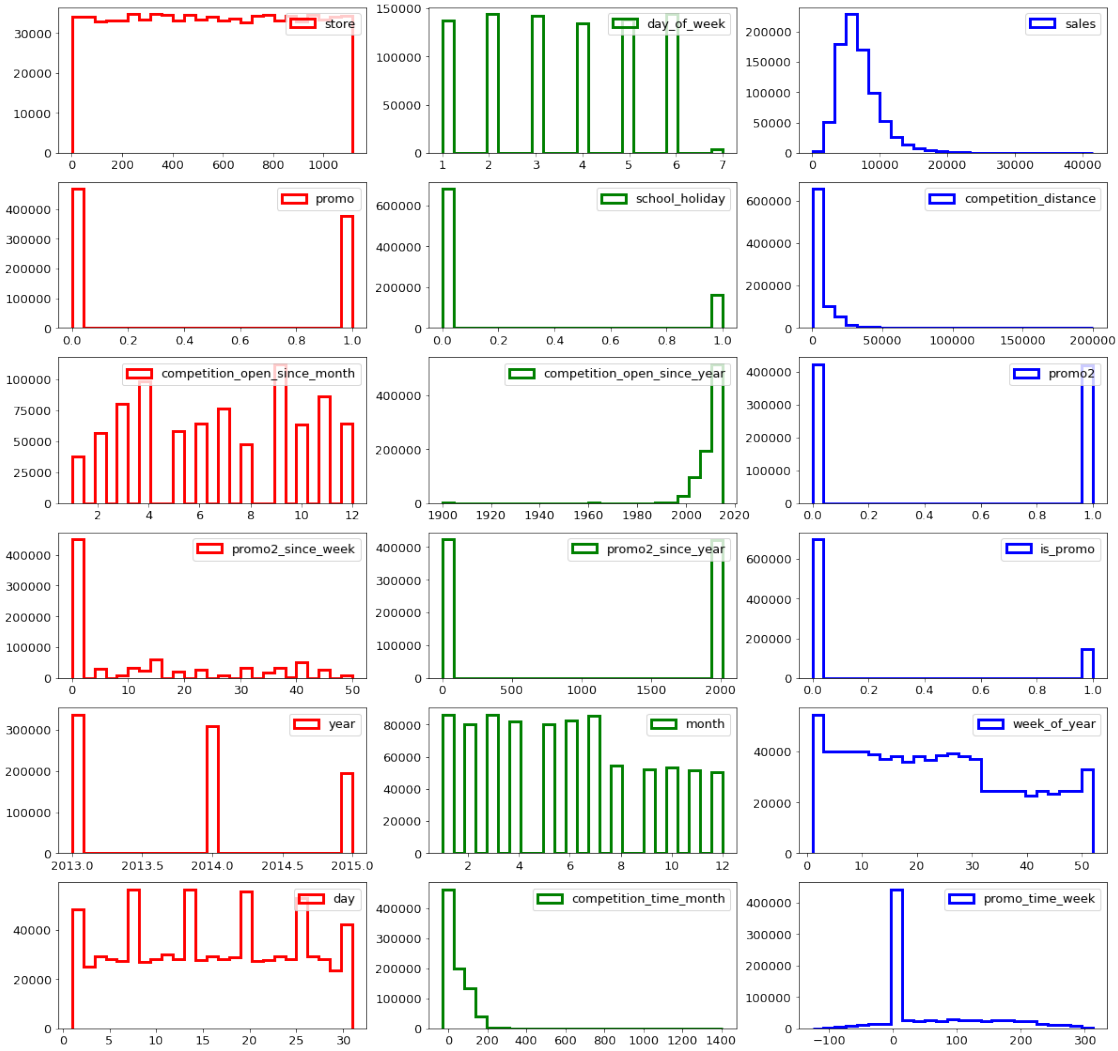
4.1.1. Response Variable

[58]:
```
plot_target( [df4['sales'], np.log1p( df4['sales'] )], 'Sales' );
```



4.1.2. Numerical Variable

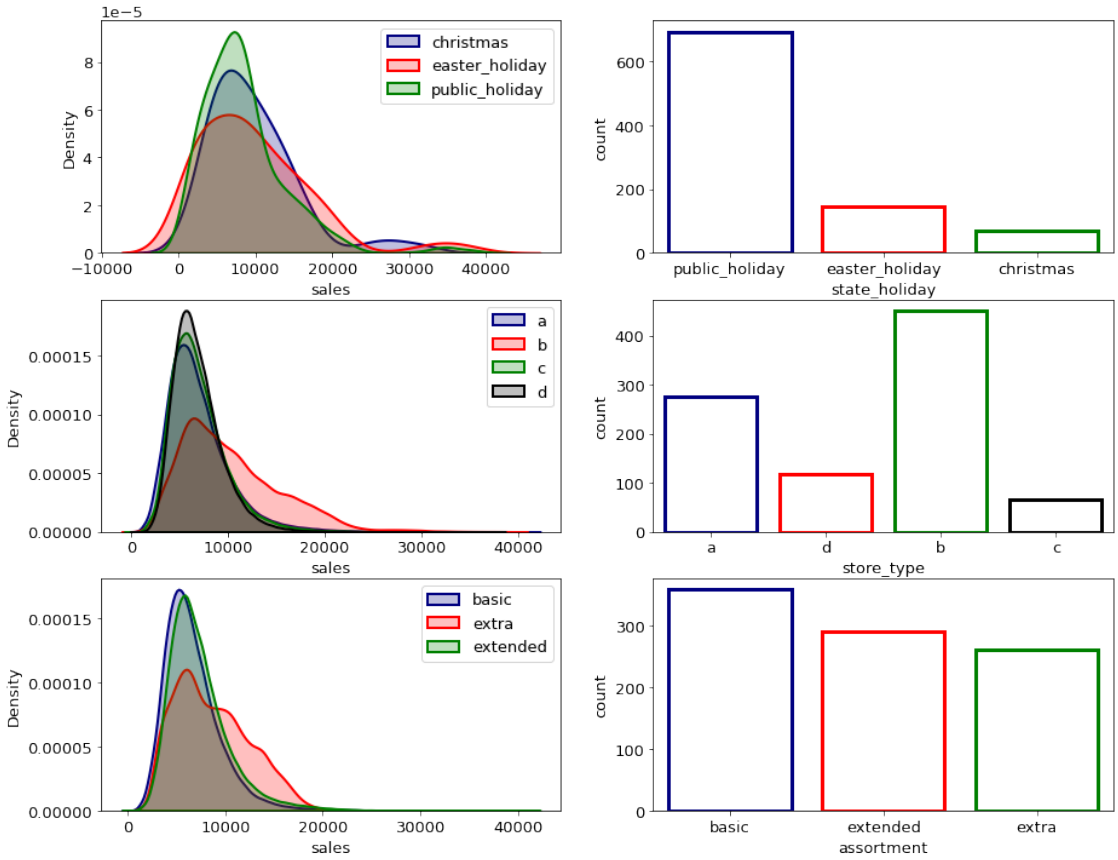
[161]:
```
num_att, catt_att = metrics( df4 )
plot_num_att( num_att, num_att.columns.tolist() )
```



4.1.3. Categorical Variable

[415]:
```
color = ['navy', 'r', 'g', 'k']

plot_categorical( df4, color );
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

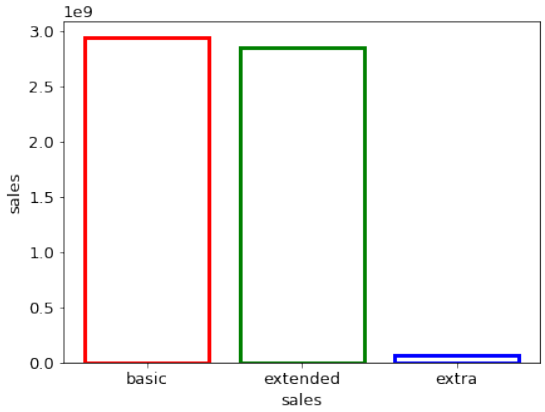


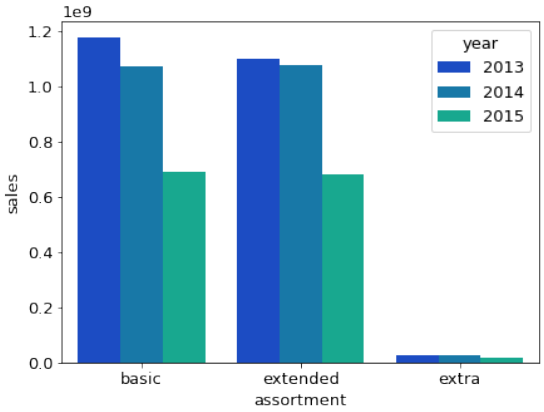
3.4.2 4.2. Bivariable Analysis

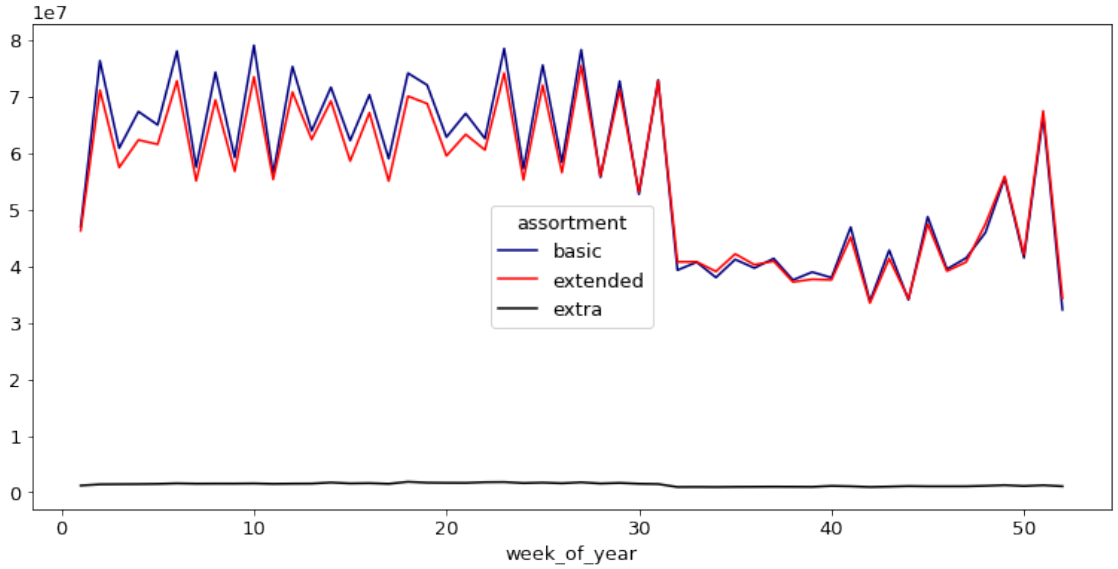
H1. Lojas com maior sortimento deveriam vender mais.

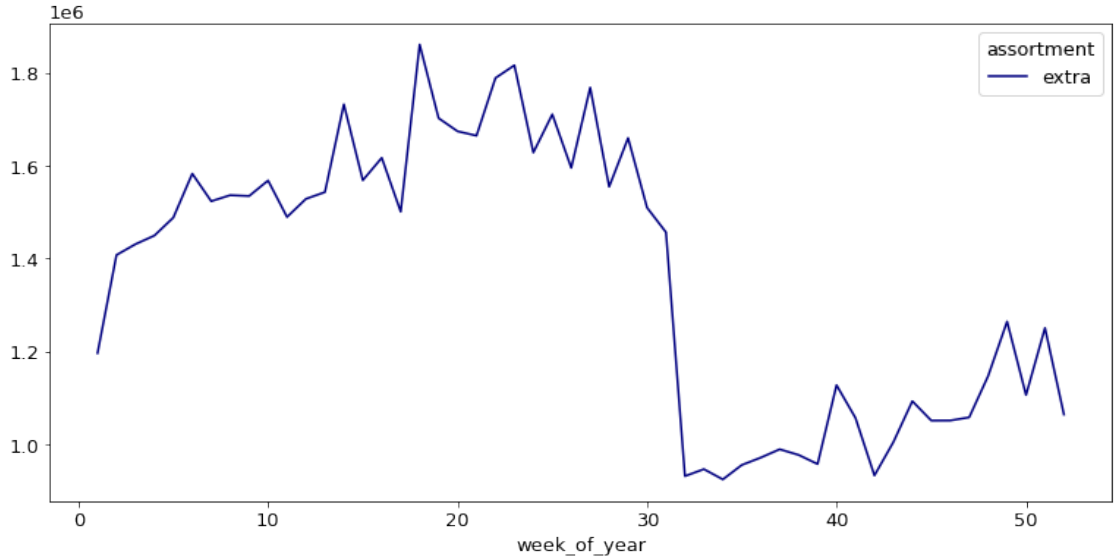
- Assumindo Extra como a maior a hipótese é **Falsa**

[524]:
```
aux0 = df4[['assortment', 'sales']].groupby('assortment').sum()
aux1 = df4[['year', 'assortment', 'sales']].groupby( ['assortment', 'year'] ).
 ↪sum().reset_index()
aux2 = df4[['week_of_year', 'assortment', 'sales']].groupby( ['assortment',␣
 ↪'week_of_year'] ).sum().reset_index()
aux3 = aux2[aux2['assortment'] == 'extra']

plot_bar( aux0, 'sales' )
plot_bar( aux0, 'sales', 'assortment', hue='year' )

aux2.pivot( index='week_of_year', columns='assortment', values='sales' ).plot(␣
 ↪color=['navy', 'r', 'k'] );
aux3.pivot( index='week_of_year', columns='assortment', values='sales' ).plot(␣
 ↪color='navy');
```


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


1e9
1.2
1.0
0.8
0.6
0.4
0.2
0.0
sales
year
2013
2014
2015
basic
extended
extra
assortment


1e7
8
7
6
5
4
3
2
1
0
assortment
basic
extended
extra
0
10
20
30
40
50
week_of_year



Teste de Hipótese Anova.

[87]:
```
anova_scipy( df4[df4['assortment'] == 'basic']['sales'], df4[df4['assortment']␣
↪== 'extended']['sales'],df4[df4['assortment'] == 'extra']['sales'] )
```

```
P-Valor: 0.0
Estatística F: 6302.467300859967
```

[83]:
```
anova_stats( df4, 'sales ~ assortment' )
```

[83]:

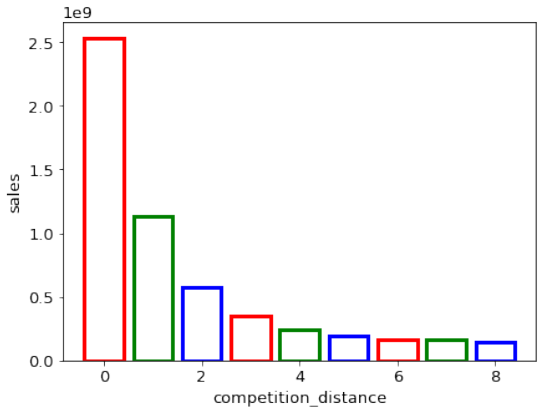
	sum_sq	df	F	PR(>F)
assortment	1.196459e+11	2.0	6302.467301	0.0
Residual	8.014419e+12	844335.0	NaN	NaN

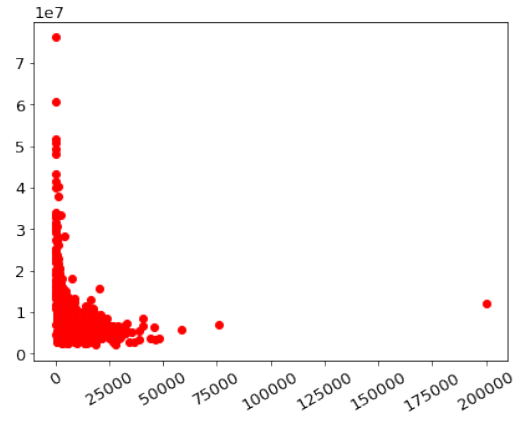
H2. Lojas com competidores mais próximos deveriam vender menos.

- Análise do grafico responde a hipótese como **Falsa**

[203]:
```
aux = df4[['competition_distance', 'sales']].groupby('competition_distance').
↪sum().reset_index()
aux['competition_binned'] = pd.cut( df4['competition_distance'], bins=np.
↪arange( 0, 20000, 2000 ) )
aux1 = aux[['competition_binned', 'sales']].groupby('competition_binned').sum().
↪reset_index()
c = ['r', 'gold', 'b', 'k', 'm', 'navy']

plt.subplot( 2, 1, 1 )
sns.heatmap( aux.corr( method='pearson' ), annot=True );
```

```
plot_bar( aux1, 'competition_binned', aux['competition_distance'], aux,␣
 ↪xl='competition_distance', scatter=True )
```

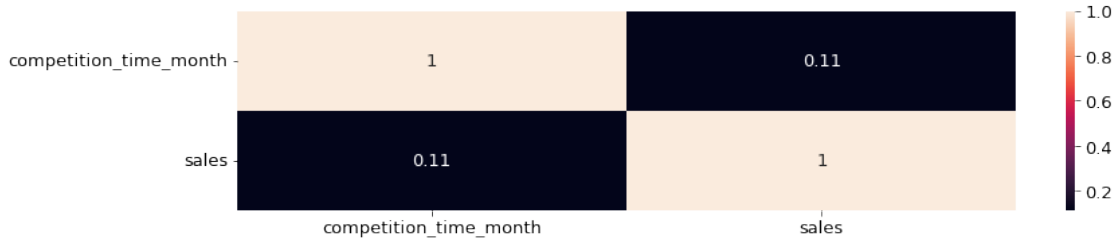


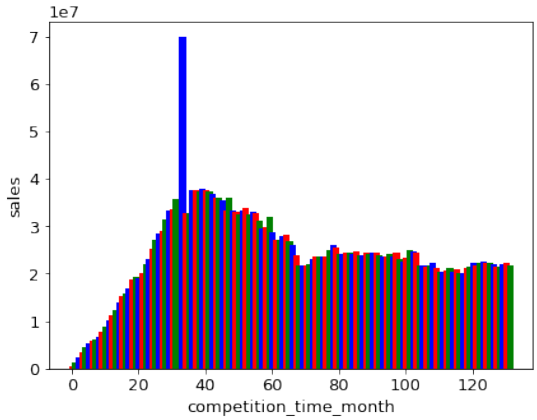


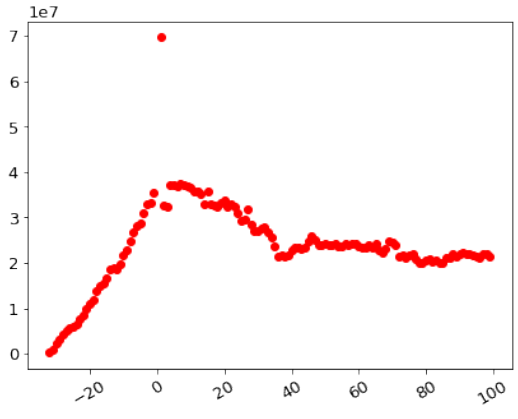
H4. Lojas com competidores a mais tempo deveriam vender mais.

- Quando mais proximo da abertura do competidor maior as vendas, logo a hipótese é **Falsa**

[307]:
```
aux = df4[['competition_time_month', 'sales']].groupby(␣
 ↪'competition_time_month' ).sum().reset_index()
aux = aux[(aux['competition_time_month'] < 100) &␣
 ↪(aux['competition_time_month'] != 0)]

plt.subplot( 2, 1, 1 )
sns.heatmap( aux.corr( method='pearson' ), annot=True );
plot_bar( aux, 'sales', 'competition_binned', aux['competition_time_month'],␣
 ↪aux, xl='competition_time_month', scatter=True )
```



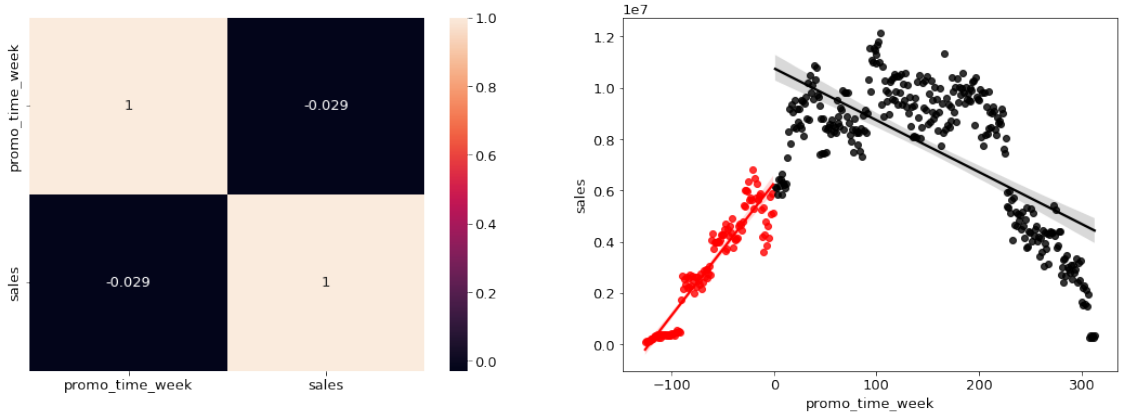


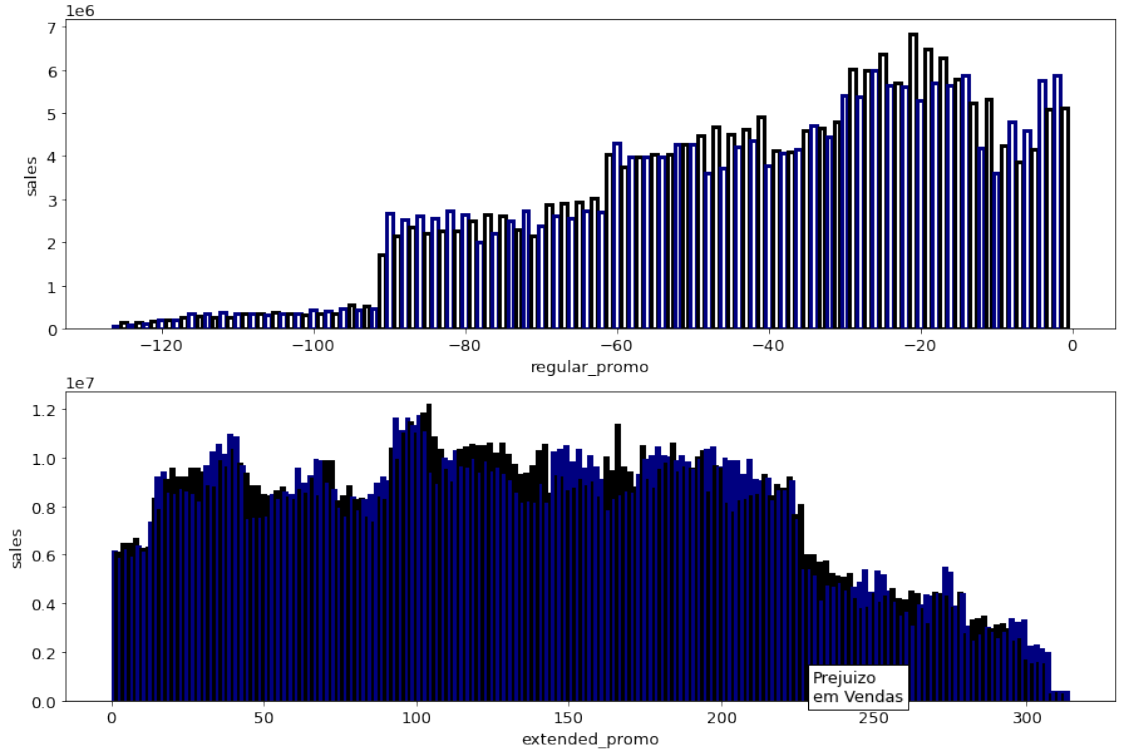


H5. Lojas com promoções ativas a mais tempo deveriam vender mais.

- Dependendo da quantidade de dias de promoções extendidas, é um prejuizo para a loja, logo a hipótese é **Falsa**

[417]:
```
aux = df4[['promo_time_week', 'sales']].groupby('promo_time_week').sum().
 ↪reset_index()
aux1 = aux[aux['promo_time_week'] < 0] # Regular Promo
aux2 = aux[aux['promo_time_week'] > 0] # Extended Promo

plt.subplot( 1, 2, 1 )
sns.heatmap( aux.corr( method='pearson' ), annot=True );

plt.subplot( 1, 2, 2 )
ax = sns.regplot( aux1['promo_time_week'], aux1['sales'], color='r' )
ax = sns.regplot( aux2['promo_time_week'], aux2['sales'], color='k' )

progression_bar( aux1, aux2, 'sales', 'promo_time_week', 'regular_promo',␣
 ↪'extended_promo', text_dims0=230, text_dims1=3*100 )
```





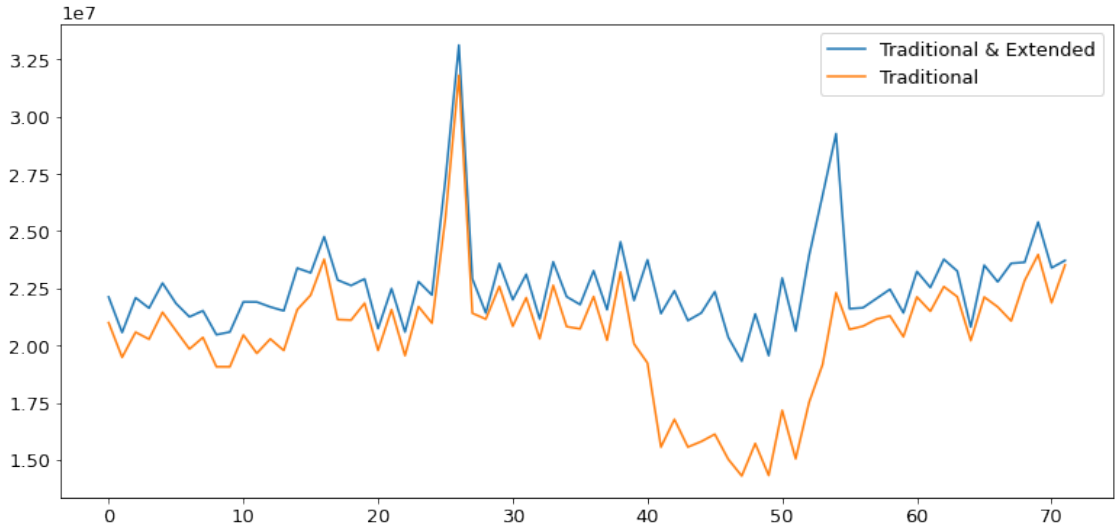
H6. Lojas com mais dias de promoção deveriam vender mais.

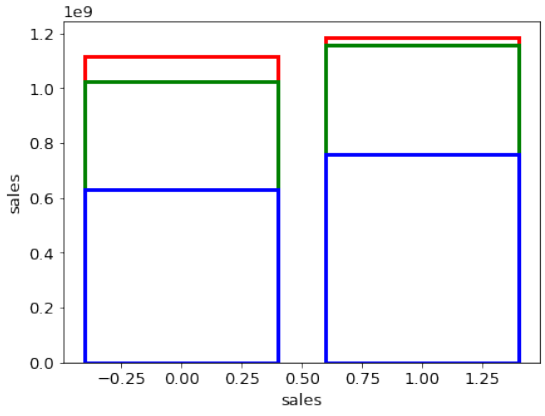
H7. Lojas com mais promoções consecutivas deveriam vender mais.

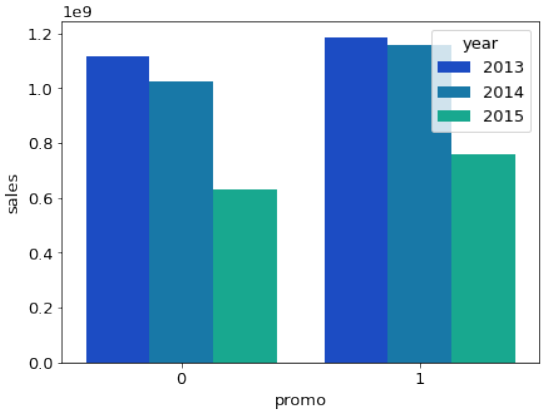
- Falsa, de acordo com os gráficos e o teste de hipóteses.

[494]:
```
aux1 = df4[['promo', 'sales', 'year']].groupby( ['promo', 'year'] ).sum().
↪reset_index()
```

```
trad = df4[( df4['promo']==1 ) & ( df4['promo2']==0 )][['sales', 'year_week']]
trad_extended = df4[( df4['promo']==1 ) & ( df4['promo2']==1 )][['sales',␣
↪'year_week']]

ax = trad.groupby( ['year_week'] ).sum().reset_index().plot()
trad_extended.groupby( ['year_week'] ).sum().reset_index().plot( ax=ax )
ax.legend( labels=['Traditional & Extended', 'Traditional'] )

plot_bar( aux1, 'promo', hue='year', xl='promo' )
```





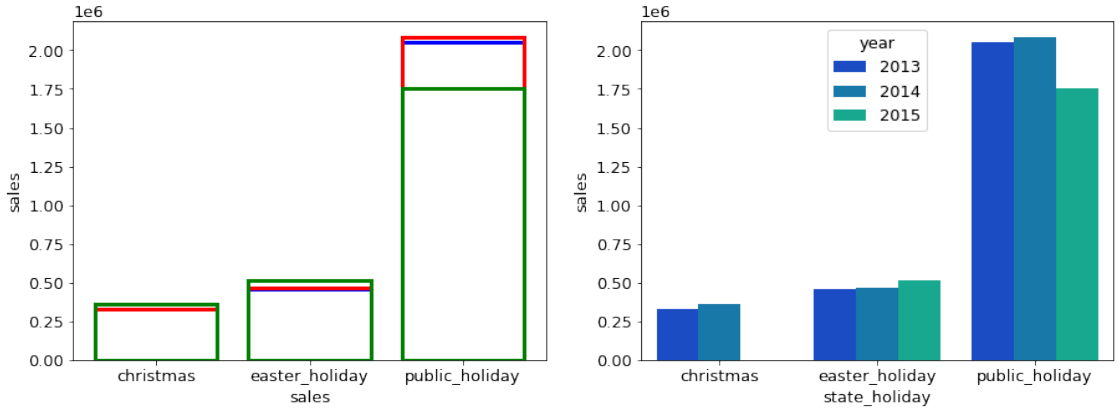


H8. Lojas abertas durante o feriado de natal deveriam vender mais.

- De acordo com os gráficos, nem o dado do ano de 2015 temos armazenado, poderia fazer uma

previsão para esses dados ou coletar eles, logo a hipótese é **Verdadeira** (Em relação a 2013 - 2014), pois as lojas vendem mais acada ano nos feriados de natal, mas em relação a outras variáveis não vende mais.

[44]:
```
aux = df4[df4['state_holiday'] != 'regular_day']
aux1 = aux[['state_holiday', 'sales', 'year']].groupby(['state_holiday',␣
 ↪'year']).sum().reset_index()

plot_bar( aux1, 'state_holiday', xl='state_holiday', hue='year' )
```



[85]:
```
anova_scipy( df4[df4['state_holiday'] == 'christmas']['sales'],␣
 ↪df4[df4['state_holiday'] ==␣
 ↪'easter_holiday']['sales'],df4[df4['state_holiday'] ==␣
 ↪'public_holiday']['sales'] )
```

```
P-Valor: 0.01654142394072274
Estatística F: 4.120494254925124
```

[86]:
```
anova_stats( df4, 'sales ~ state_holiday' )
```

[86]:

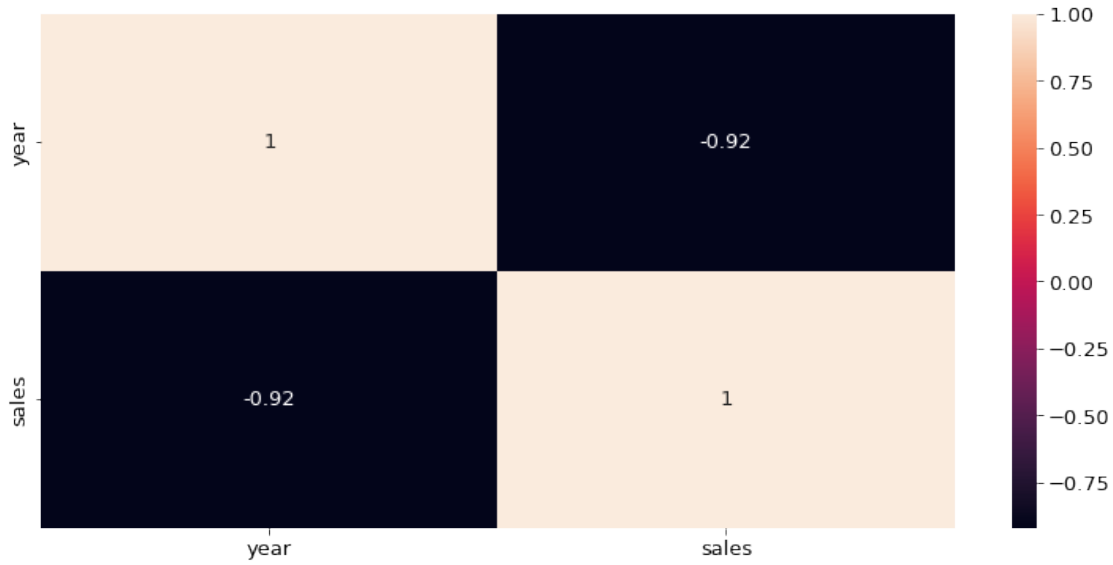
	sum_sq	df	F	PR(>F)
state_holiday	3.429413e+09	3.0	118.710298	7.283351e-77
Residual	8.130635e+12	844334.0	NaN	NaN

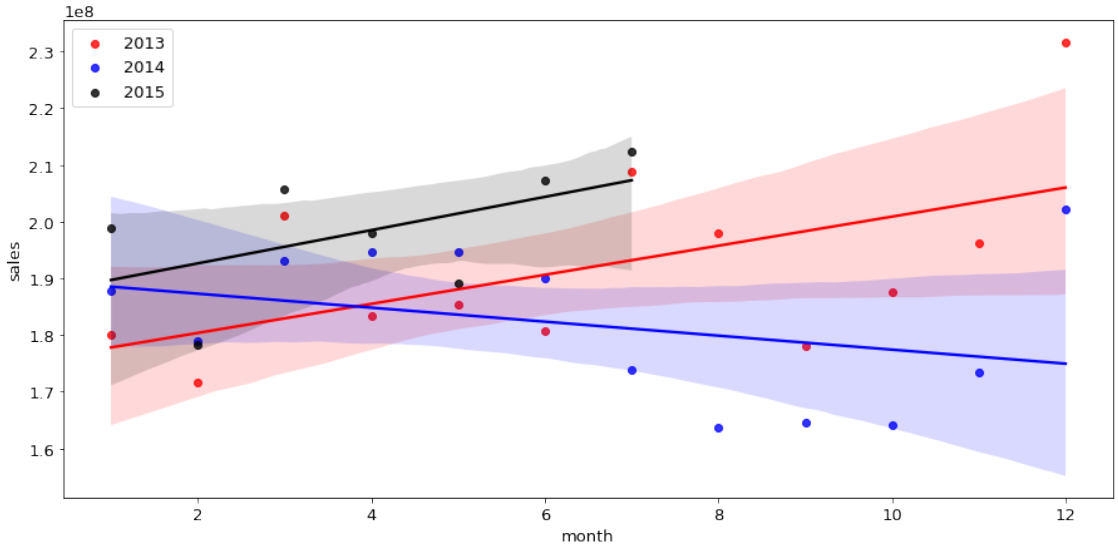
H9. Lojas deveriam vender mais ao longo dos anos.

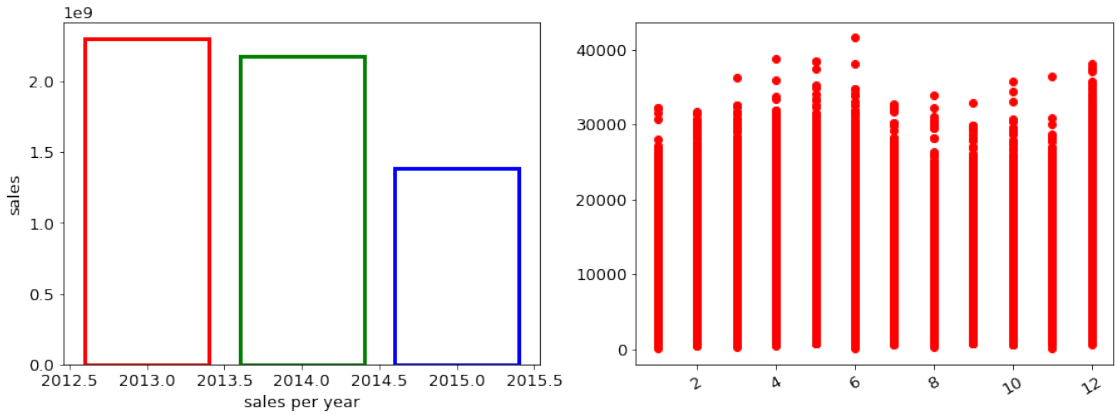
- Como não temos todos os dados, apriori poderiamos dizer que realmente as lojas poderiam vender menos ao longo dos anos, mas como o ano de 2015 não esta concluido, pode ser que as vendas em 2015 aumentem, logo essa hipótese é **Verdadeira**

[120]:
```
aux = df4[['sales', 'year']].groupby(['year']).sum().reset_index()

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

```
fig, ax = plt.subplots( figsize=(15, 7) )
for i in zip( [df4[df4['year'] == 2013], df4[df4['year'] == 2014],␣
 ↪df4[df4['year'] == 2015]], ['r', 'b', 'k'], ['2013', '2014', '2015']):
    aux2 = i[0][['sales', 'month']].groupby('month').sum().reset_index()
    ax = sns.regplot( aux2['month'], aux2['sales'], color=i[1], label=i[2] )
    ax.legend()

plot_bar( aux, 'year', xl='sales per year', scatter=True,␣
 ↪df_scatter_x=df4['month'], df_scatter_y=df4['sales'] )
```



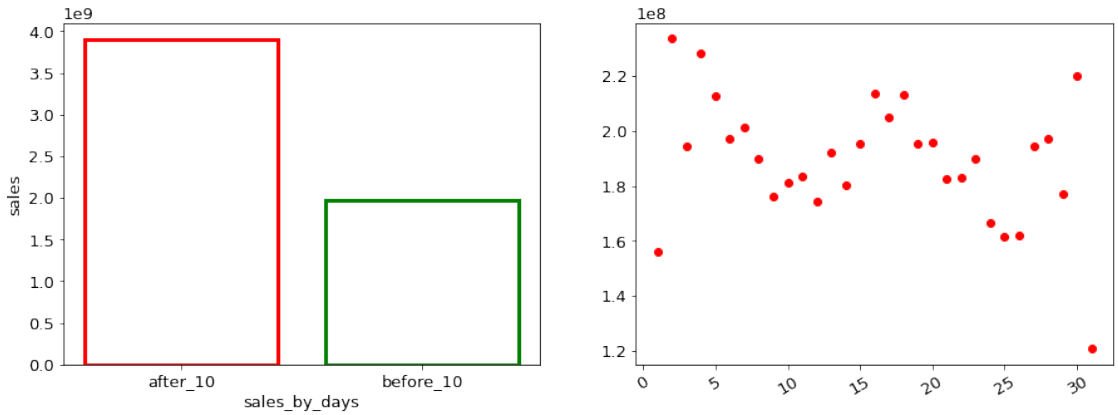




H11. Lojas deveriam vender mais depois do dia 10 de cada mês.

- Devido ao dobro do periodo não há uma comparação justa, logo é **Verdadeira**

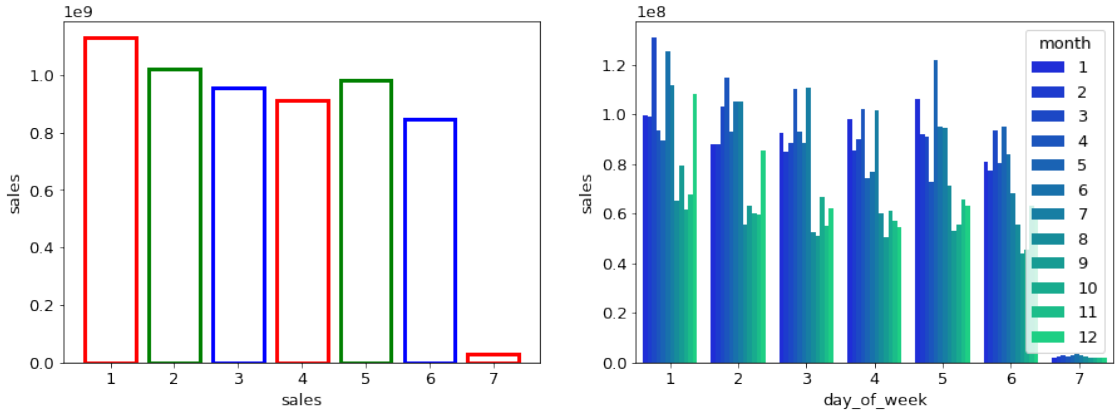
[134]:
```
aux = df4[['day', 'sales']].groupby('day').sum().reset_index()
aux['before_10'] = aux['day'].apply( lambda x: 'before_10' if x <= 10 else␣
↪'after_10' )
aux1 = aux[['before_10', 'sales']].groupby('before_10').sum().reset_index()

plot_bar( aux1, 'before_10', xl='sales_by_days', scatter=True,␣
↪df_scatter_x=aux['day'] , df_scatter_y=aux['sales'] )
```



H12. Lojas deveriam vender menos nos finais de semana.

- Dado os gráficos, as lojas realmente vendem menos.

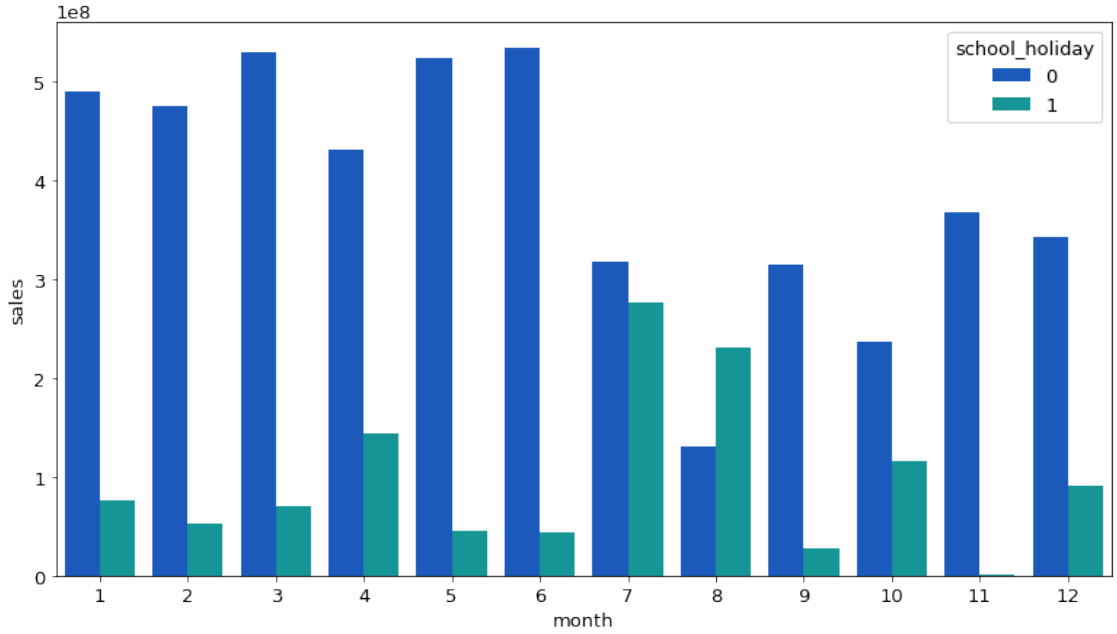
[151]:
```
aux = df4[['day_of_week', 'sales']].groupby('day_of_week').sum().reset_index()
aux2 = df4[['day_of_week', 'month', 'sales']].groupby(['day_of_week', 'month']).
 ↪sum().reset_index()

plot_bar( aux, 'day_of_week', xl='sales_by_days', hue='month', df_hue=aux2 )
```



H13. Lojas deveriam vender menos nos feriados escolares.

- Em comparação com os demais meses e lembrando que não esta fechado o dataset, logo essa hipótese é **falsa**.

[292]:
```
aux = df4[['school_holiday', 'sales', 'month']].groupby(['school_holiday',␣
 ↪'month']).sum().reset_index()

fig, ax = plt.subplots( figsize=(13, 7) )
ax = sns.barplot( aux['month'], aux['sales'], hue=aux['school_holiday'],␣
 ↪palette='winter')
```

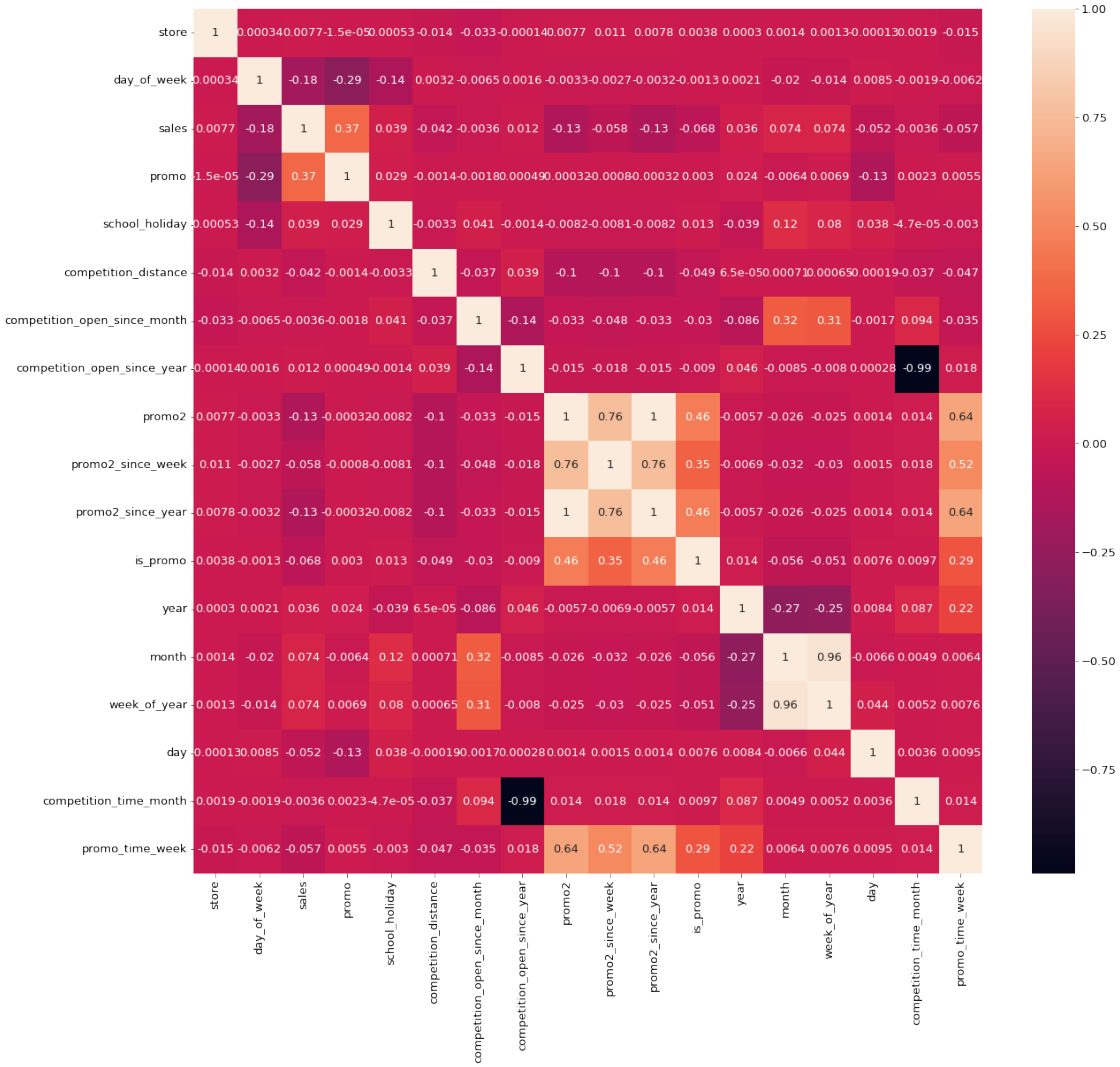


3.4.3 4.3. Multivariable Analysis

[316]:
```
num_att, catt_att = mp.metrics( df4 )
```

4.3.1. Numerical Attributes

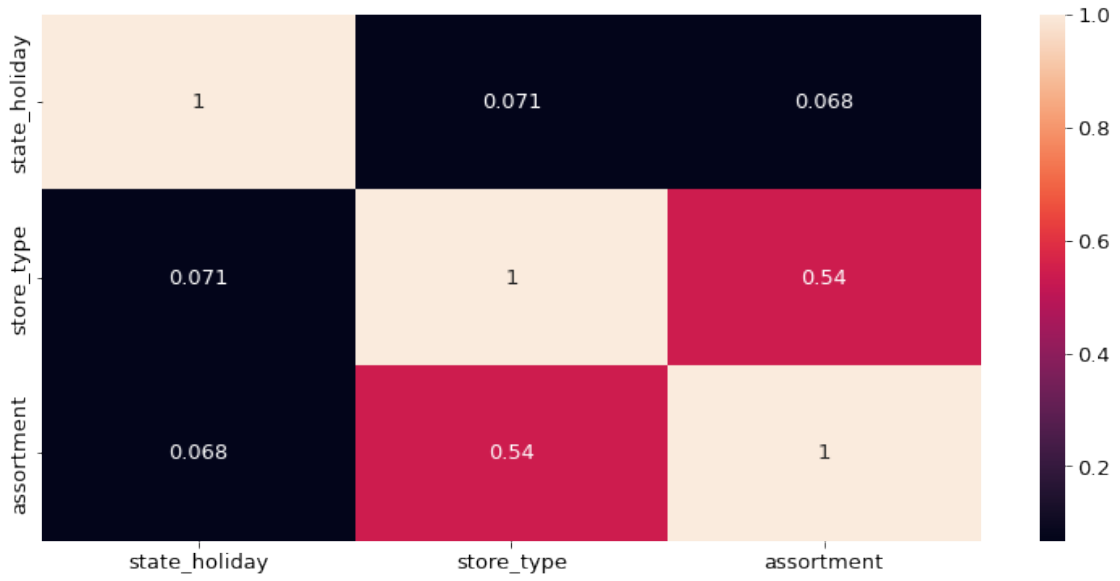
[312]:
```
plt.rcParams['figure.figsize'] = [20, 18]
sns.heatmap( num_att.corr( method='pearson' ), annot=True );
```



4.3.2. Categorical Attributes

[327]:
```
a = catt_att.iloc[:, :-2]

cols = []
for i in a.columns.tolist():
    for j in a.columns.tolist():
        a1 = cramer_v( a[i], a[j] )
        cols.append(a1)

d = pd.DataFrame( {'state_holiday': cols[0:3], 'store_type': cols[3:6],␣
 ↪'assortment': cols[6:9]} )
d = d.set_index( d.columns )
```

[380]:
```
sns.heatmap( d, annot=True );
```



3.5 5.0. Data Preparation

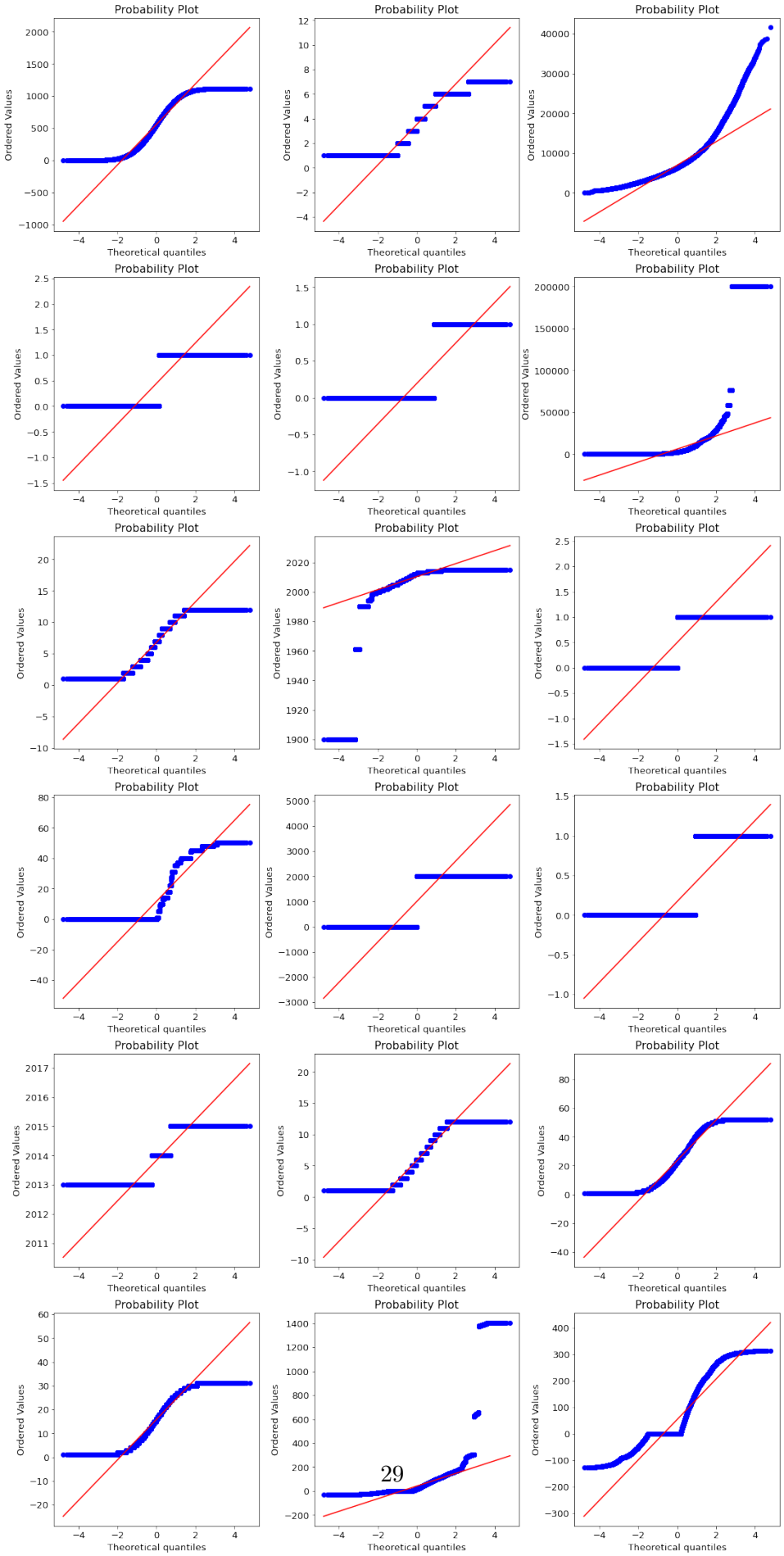
[228]:
```
df5 = df3.copy()
```

3.5.1 5.1. Normalization

[156]:
```
cols = num_att.columns.tolist()

# Hipótese Nula (Ho): As Distribuições vem de uma pop normal.
# Hipótese Alte (Ha): As distribuições não vem de uma pop normal.
# alpha -> 0.05

fig, ax = plt.subplots( 6, 3, figsize=(15, 30))
plt.tight_layout(w_pad=3.0, h_pad=3.0)
for i in range( len( cols ) ):
    ax = ax.flatten()
    print("T Stats: {:.4f} for {}".format( stats.kstest( df5[cols[i]], 'norm'␣
 ↪)[1], cols[i] ) )
    stats.probplot( df5[cols[i]], plot=ax[i] )
```

```
T Stats: 0.0000 for store
T Stats: 0.0000 for day_of_week
T Stats: 0.0000 for sales
T Stats: 0.0000 for promo
T Stats: 0.0000 for school_holiday
```

```
T Stats: 0.0000 for competition_distance
T Stats: 0.0000 for competition_open_since_month
T Stats: 0.0000 for competition_open_since_year
T Stats: 0.0000 for promo2
T Stats: 0.0000 for promo2_since_week
T Stats: 0.0000 for promo2_since_year
T Stats: 0.0000 for is_promo
T Stats: 0.0000 for year
T Stats: 0.0000 for month
T Stats: 0.0000 for week_of_year
T Stats: 0.0000 for day
T Stats: 0.0000 for competition_time_month
T Stats: 0.0000 for promo_time_week
```


Probability Plot
Ordered Values
Theoretical quantiles

3.5.2 5.2. Rescaling

[229]:
```
rs = pp.RobustScaler()
mms = pp.MinMaxScaler()

# competition_distance
df5['competition_distance'] = rs.fit_transform( df5[['competition_distance']].
 ↪values )

# competition_time_month
df5['competition_time_month'] = rs.fit_transform(␣
 ↪df5[['competition_time_month']].values )

# promo_time_week
df5['promo_time_week'] = mms.fit_transform( df5[['promo_time_week']].values )

# year
df5['year'] = mms.fit_transform( df5[['year']].values )
```

3.5.3 5.3. Transformation

5.3.1. Encoding

[230]:
```
# Frequency Encoding
f_store_type = df5.groupby('store_type').size() / len(df5)
f_assortment = df5.groupby('assortment').size() / len(df5)
f_state_holiday = df5.groupby('state_holiday').size() / len(df5)

df5['store_type'] = df5['store_type'].apply( lambda x: f_store_type[x] )
df5['assortment'] = df5['assortment'].apply( lambda x: f_assortment[x] )
df5['state_holiday'] = df5['state_holiday'].apply( lambda x: f_state_holiday[x]␣
 ↪)
```

5.3.2. Response Transformation

[243]:
```
df5['sales'] = np.log1p(df5['sales'])
```

5.3.3. Nature Transformation

[244]:
```
# 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. * ( 2. * np.pi /
↪30 ) ) ) )
df5['day_cos'] = df5['day'].apply( lambda x: np.cos( x * ( 2. * ( 2. * np.pi /
↪30 ) ) ) )

# 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 ) ) )

# 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 ) ) )
```

[246]:
```
df5.head()
```

[246]:

	store	day_of_week	date	sales	promo	state_holiday	\
0	1	5	2015-07-31	8.568646	1	0.998922	
1	2	5	2015-07-31	8.710290	1	0.998922	
2	3	5	2015-07-31	9.025816	1	0.998922	
3	4	5	2015-07-31	9.546527	1	0.998922	
4	5	5	2015-07-31	8.481151	1	0.998922	

	school_holiday	store_type	assortment	competition_distance	…	\
0	1	0.133795	0.526892	-0.170968	…	
1	1	0.541302	0.526892	-0.283871	…	
2	1	0.541302	0.526892	1.903226	…	
3	1	0.133795	0.463386	-0.275806	…	
4	1	0.541302	0.526892	4.448387	…	

	promo_since	promo_time_week	month_sin	month_cos	day_sin	\
0	0	0.287016	-0.5	-0.866025	0.406737	
1	2010-03-22 00:00:00	0.922551	-0.5	-0.866025	0.406737	
2	2011-03-28 00:00:00	0.801822	-0.5	-0.866025	0.406737	
3	0	0.287016	-0.5	-0.866025	0.406737	
4	0	0.287016	-0.5	-0.866025	0.406737	

	day_cos	day_of_week_sin	day_of_week_cos	week_of_year_sin	\
0	0.913545	-0.974928	-0.222521	-0.568065	
1	0.913545	-0.974928	-0.222521	-0.568065	
2	0.913545	-0.974928	-0.222521	-0.568065	
3	0.913545	-0.974928	-0.222521	-0.568065	
4	0.913545	-0.974928	-0.222521	-0.568065	

```
   week_of_year_cos
0         -0.822984
1         -0.822984
2         -0.822984
3         -0.822984
4         -0.822984

[5 rows x 33 columns]
```

[247]:
```
df5.to_csv("../data_backup/df5.csv")
```

3.6 6.0. Feature Selection

[3]:
```
df6 = pd.read_csv("../data_backup/df5.csv")
df6['date'] = pd.to_datetime( df6['date'] )

cols = ['Unnamed: 0', 'week_of_year', 'day', 'month', 'promo_since',␣
 ↪'competition_since', 'day_of_week', 'year_week']
df6 = df6.drop( columns=cols, axis=1 )
```

3.6.1 6.1. Split Into Train and Test

- After 06-19 Test Dataset
- Before 06-16 Train Dataset

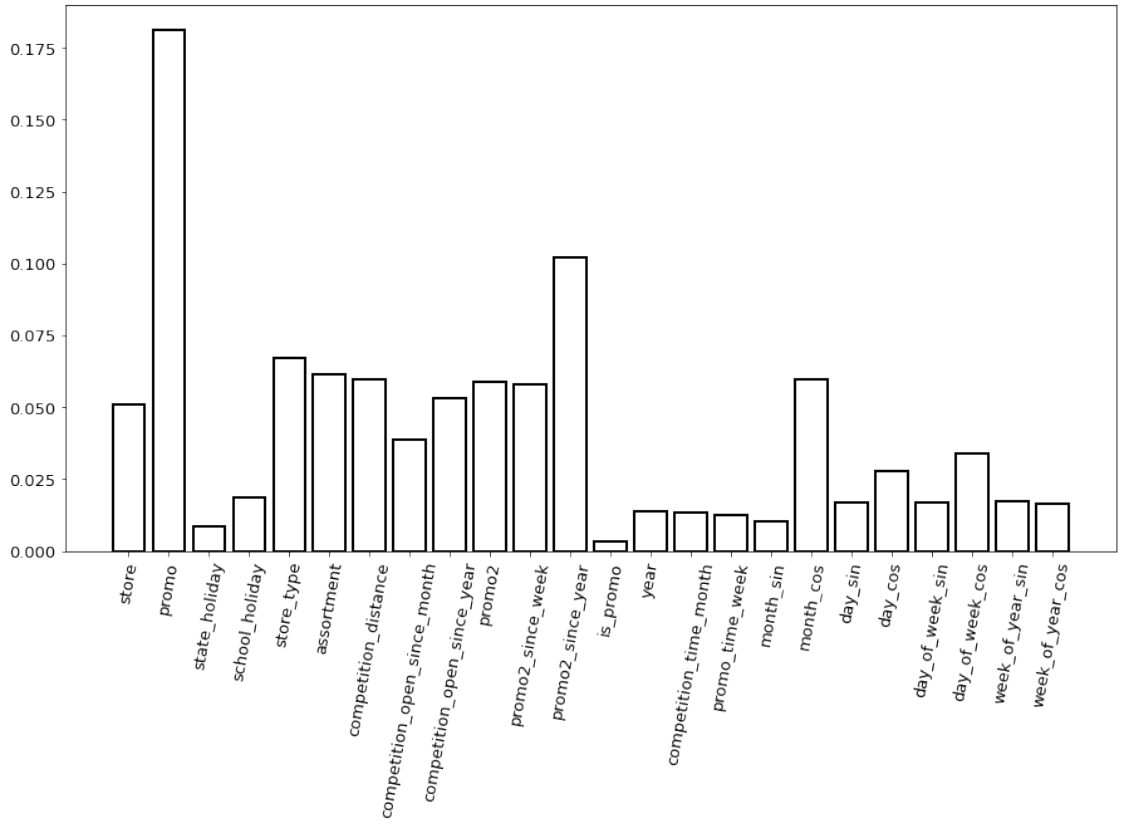
[4]:
```
X_train = df6[df6['date'] < df6['date'].max() - datetime.timedelta( days=42 )]
y_train = X_train['sales']

X_test = df6[df6['date'] >= df6['date'].max() - datetime.timedelta( days=42 )]
y_test = X_test['sales']

print(f"Train Min Date: {X_train['date'].min()}")
print(f"Test Min Date: {X_test['date'].min()}")

print(f"\nTrain Max Date: {X_train['date'].max()}")
print(f"Test Max Date: {X_test['date'].max()}")
```

```
Train Min Date: 2013-01-01 00:00:00
Test Min Date: 2015-06-19 00:00:00

Train Max Date: 2015-06-18 00:00:00
Test Max Date: 2015-07-31 00:00:00
```

3.6.2 6.2. XGBoost Feature Importance

[131]:
```
X_train_xg = X_train.drop( columns=['sales', 'date'], axis=1 ).values
y_train_xg = y_train.values

xg = XGBRegressor( n_jobs=-1 ).fit( X_train_xg, y_train_xg )

importances = {}
for i in range( 0, 24 ):
    importances[X_train.drop( columns=['date', 'sales'], axis=1 ).columns[i]] =␣
 ↪xg.feature_importances_[i]
```

[132]:
```
fig, ax = plt.subplots( figsize=(15, 8))
ax.bar( importances.keys(), importances.values(), **mp.args_b(edgecolor='k') )
plt.xticks( rotation=80 );
```



[119]:
```
df3['state_holiday'].value_counts()
```

[119]:
```
regular_day       595478
public_holiday       483
easter_holiday        98
```

```
christmas              71
Name: state_holiday, dtype: int64
```

3.6.3 6.3. Boruta Feature Selection

[ ]:
```
rf = RandomForestRegressor( n_jobs=-1 )

bp = BorutaPy( rf, n_estimators='auto', random_state=42, verbose=2 ).fit(␣
 ↪X_train_xg, y_train_xg )

cols = bp.support_.tolist()
cols_boruta = X_train.iloc[:, cols].columns.tolist()
```

3.6.4 6.4. Manual Feature Selection

[7]:
```
cols_selected = ['store', 'promo2',
                 'promo', 'store_type', 'assortment',
                 'competition_distance', 'competition_open_since_month',
                 'competition_open_since_year', 'promo2_since_week',
                 'promo2_since_year', 'competition_time_month',␣
 ↪'promo_time_week',
                 'month_sin', 'month_cos', 'day_sin', 'day_cos',
                 'day_of_week_sin', 'day_of_week_cos',
                 'week_of_year_sin', 'week_of_year_cos']

cols_full = cols_selected+['date', 'sales']
```

3.7 7.0. Machine Learning Models

[11]:
```
x_train_ = X_train[cols_selected]
x_test_ = X_test[cols_selected]

mms = pp.MinMaxScaler()

x_test_['competition_open_since_year'] = mms.fit_transform(␣
 ↪x_test_[['competition_open_since_year']].values )
x_test_['competition_open_since_month'] = mms.fit_transform(␣
 ↪x_test_[['competition_open_since_month']].values )

x_train_['competition_open_since_year'] = mms.fit_transform(␣
 ↪x_train_[['competition_open_since_year']].values )
x_train_['competition_open_since_month'] = mms.fit_transform(␣
 ↪x_train_[['competition_open_since_month']].values )
```

3.7.1 7.1. Average Model

[203]:
```
aux = x_test_.copy()
aux['sales'] = y_test.copy()

# Predict
aux1 = aux[['store', 'sales']].groupby('store').mean().reset_index().rename(␣
 ↪columns={'sales':'prediction'} )
aux  = pd.merge( aux, aux1, how='left', on='store' )
yhat_base = aux['prediction']

# Performace
base_result = mp.ml_error( 'Avg Model', np.expm1( y_test ), np.expm1( yhat_base␣
 ↪) )
base_result
```

[203]:

	Model Name	MAE	MAPE	RMSE
0	Avg Model	1354.800353	0.2064	1835.135542

3.7.2 7.2. SVR

[204]:
```
# Definition
svr = SVR(max_iter=100).fit( x_train_, y_train )

# Predict
yhat_svr = svr.predict( x_test_ )

# Performace
svr_result = mp.ml_error( 'SVR Model', np.expm1( y_test ), np.expm1( yhat_svr )␣
 ↪)
svr_result
```

[204]:

	Model Name	MAE	MAPE	RMSE
0	SVR Model	5786.217815	0.794773	6537.894802

3.7.3 7.3. Random Forest Regression

[205]:
```
# Definition
rf = RandomForestRegressor( n_jobs=-1, random_state=42 ).fit( x_train_, y_train␣
 ↪)

# Predict
yhat_rf = rf.predict( x_test_ )

# performace
rf_result = mp.ml_error( 'Random Forest Model', np.expm1( y_test ), np.expm1(␣
 ↪yhat_rf ) )
```

```
rf_result
```

[205]:

	Model Name	MAE	MAPE	RMSE
0	Random Forest Model	682.975548	0.100779	1013.700116

3.7.4 7.4. XGBoost Regression

[82]:
```
# Definition
#xg = XGBRegressor( n_jobs=-1, random_state=42 ).fit( x_train_, y_train )

# Predict
yhat_xg = xg.predict( x_test_ )

# Performace
xg_result = ml_error2( 'XGBoost Model', np.expm1( y_test ), np.expm1( yhat_xg )␣
↪)
xg_result
```

[82]:

	Model Name	MAE	MAPE	RMSE
0	XGBoost Model	0.131237	0.131237	1277.698002

3.7.5 7.5. Model Performace

[208]:
```
model_performace_0 = pd.concat( [xg_result, rf_result, svr_result,␣
↪base_result], axis=0 ).sort_values( 'RMSE' ).reset_index( drop=True )
model_performace_0
```

[208]:

	Model Name	MAE	MAPE	RMSE
0	Random Forest Model	682.975548	0.100779	1013.700116
1	XGBoost Model	888.718076	0.131237	1277.698002
2	Avg Model	1354.800353	0.206400	1835.135542
3	SVR Model	5786.217815	0.794773	6537.894802

3.8 7.x. Data Preparation After Split

[11]:
```
#df7x = df3.copy()
df7x = pd.read_csv("../data_backup/df3.csv")
df7x = df7x.drop( columns=['Unnamed: 0'], axis=1 )

df7x['date'] = pd.to_datetime( df7x['date'] )
```

3.8.1 7.1. Split Dataset

[12]:
```
X_train = df7x[df7x['date'] < df7x['date'].max() - datetime.timedelta( days=42␣
 ↪)]
y_train = X_train['sales']

X_test = df7x[df7x['date'] >= df7x['date'].max() - datetime.timedelta( days=42␣
 ↪)]
y_test = X_test['sales']

print(f"Train Min Date: {X_train['date'].min()}")
print(f"Test Min Date: {X_test['date'].min()}")

print(f"\nTrain Max Date: {X_train['date'].max()}")
print(f"Test Max Date: {X_test['date'].max()}")
```

```
Train Min Date: 2013-01-01 00:00:00
Test Min Date: 2015-06-19 00:00:00

Train Max Date: 2015-06-18 00:00:00
Test Max Date: 2015-07-31 00:00:00
```

3.8.2 7.2. Data Preparation

[13]:
```
rs = pp.RobustScaler()
mms = pp.MinMaxScaler()

y_test = np.log1p( y_test )
y_train = np.log1p( y_train )

X_train, X_test = mp.train_test_prep( X_train, X_test )
```

3.8.3 7.3. Machine Learning Models

[14]:
```
x_train = X_train[cols_selected]
x_test = X_test[cols_selected]
```

7.3.1. Average Model

[144]:
```
aux = x_test.copy()
aux['sales'] = y_test.copy()

# Predict
aux1 = aux[['store', 'sales']].groupby('store').mean().reset_index().rename(␣
 ↪columns={'sales':'prediction'} )
aux  = pd.merge( aux, aux1, how='left', on='store' )
yhat_base = aux['prediction']
```

```
# Performace
base_result = mp.ml_error( 'Avg Model', np.expm1( y_test ), np.expm1( yhat_base␣
↪) )
base_result
```

[144]:

	Model Name	MAE	MAPE	RMSE
0	Avg Model	1354.800353	0.2064	1835.135542

7.3.2. SVR

[49]:
```
# Definition
svr = SVR(max_iter=100).fit( x_train, y_train )

# Predict
yhat_svr = svr.predict( x_test )

# Performace
svr_result = mp.ml_error( 'SVR Model', np.expm1( y_test ), np.expm1( yhat_svr )␣
↪)
svr_result
```

[49]:

	Model Name	MAE	MAPE	RMSE
0	SVR Model	5786.217647	0.794773	6537.894648

7.3.3. Random Forest Regression

[186]:
```
# fit
rf = RandomForestRegressor( n_jobs=-1, random_state=42 ).fit( x_train, y_train )

# predict
yhat_rf = rf.predict( x_test )

# performace
rf_result = mp.ml_error( 'Random Forest Model', np.expm1( y_test ), np.expm1(␣
↪yhat_rf ) )
rf_result
```

[186]:

	Model Name	MAE	MAPE	RMSE
0	Random Forest Model	912.069229	0.125143	1299.120407

7.3.4. XGBoost Regression

[185]:
```
# Definition
xg = XGBRegressor( n_jobs=-1, random_state=42 ).fit( x_train, y_train )

# Predict
yhat_xg = xg.predict( x_test )
```

```
# Performace
xg_result = mp.ml_error( 'XGBoost Model', np.expm1( y_test ), np.expm1( yhat_xg␣
↪) )
xg_result
```

[185]:

	Model Name	MAE	MAPE	RMSE
0	XGBoost Model	1490.819553	0.186662	2081.622564

3.8.4 7.4. Model Results

[197]:
```
pd.concat( [base_result, svr_result, rf_result, xg_result], axis=0 ).
↪reset_index( drop=True ).sort_values( 'RMSE' )
```

[197]:

	Model Name	MAE	MAPE	RMSE
2	Random Forest Model	912.069229	0.125143	1299.120407
0	Avg Model	1339.842213	0.206209	1740.751135
3	XGBoost Model	1490.819553	0.186662	2081.622564
1	SVR Model	6478.753549	0.813506	7191.242927

[209]:
```
model_performace_0
```

[209]:

	Model Name	MAE	MAPE	RMSE
0	Random Forest Model	682.975548	0.100779	1013.700116
1	XGBoost Model	888.718076	0.131237	1277.698002
2	Avg Model	1354.800353	0.206400	1835.135542
3	SVR Model	5786.217815	0.794773	6537.894802

[199]:
```
x_train.to_csv('../data_backup/x_train_after.csv')
x_test.to_csv('../data_backup/x_test_after.csv')
```

3.8.5 7.5. Cross Validation

[114]:
```
# Feature Selecion + Store and Sales
x_training = X_train[ cols_full ]
```

[9]:
```
a0 = mp.cross_val( x_training, 4, 'SVR Model', SVR( max_iter=4 ) )
a1 = mp.cross_val( x_training, 4, 'Random Forest Model', RandomForestRegressor(␣
↪n_jobs=-1, random_state=42 ) )
a2 = mp.cross_val( x_training, 4, 'XGBoost Model', XGBRegressor( n_jobs=-1,␣
↪random_state=42 ) )
```

[10]:
```
models_cross_val = pd.concat( [a0, a1, a2], axis=0 ).reset_index( drop=True )
models_cross_val
```

[10]:
```
              Model Name                MAE Cv          MAPE Cv  \
0            SVR Model  5780.784 +/- 338.408  0.779 +/- 0.009
1  Random Forest Model   792.646 +/- 170.059  0.115 +/- 0.023
2        XGBoost Model    990.471 +/- 94.369  0.142 +/- 0.007

                 RMSE Cv
0  6513.706 +/- 394.442
1  1175.622 +/- 241.843
2  1406.819 +/- 145.715
```

3.9 8.0. Hyperparameter Fine Tuning

[87]:
```
#from sklearn.model_selection import GridSearchCV
# Param Info: https://xgboost.readthedocs.io/en/stable/parameter.html

params = {
    'n_estimators': [2500, 2900, 3000, 3100, 3200, 3500],
    'eta':   [.02, .03, .033, .035],
    'gamma': [1., 1.5],
    'max_depth': [5, 6, 7],
    'subsample': [.6, .7, .8, .9],
    'min_child_weight': [3, 4, 5, 6, 7],
    'colsample_bytree': [.5, .7, .9, 1.]
}
```

[19]:
```
# for i in range( 50 ):
#     hp = {k: sample( v, 1 ) for k, v in params.items()}
#     print(hp, file=open('parameters.txt', 'a'))

#     # Definition
#     xg = XGBRegressor( n_jobs = -1,
#                        n_estimators = hp['n_estimators'][0],
#                        eta    = hp['eta'][0],
#                        gamma = hp['gamma'][0],
#                        max_depth = hp['max_depth'][0],
#                        subsample = hp['subsample'][0],
#                        min_child_weight = hp['min_child_weight'][0],
#                        colsample_bytree = hp['colsample_bytree'][0])

#     # Performace
#     xg_result = mp.cross_val( x_training, 2, 'XGBoost Model', xg )
#     print( f"{ xg_result['RMSE Cv'].tolist()[0] } \n", file=open('parameters.
 ↪txt', 'a') )
```

3.9.1 8.1. Final Model

[88]:
```
param_tunned = {
    'n_estimators': 3500,
    'eta': .035,
    'gamma': 1,
    'max_depth': 6,
    'subsample': .7,
    'min_child_weight': 7,
    'colsample_bytree': 1
}
```

3.9.2 8.2. Model Performace (Prepare -> Split)

[90]:
```
xg_tnn = XGBRegressor( n_jobs = -1,
                       n_estimators = param_tunned['n_estimators'],
                       eta    = param_tunned['eta'],
                       gamma = param_tunned['gamma'],
                       max_depth = param_tunned['max_depth'],
                       subsample = param_tunned['subsample'],
                       min_child_weight = param_tunned['min_child_weight'],
                       colsample_bytree = param_tunned['colsample_bytree']).
 ↪fit( x_train, y_train )

# Predict
yhat = xg_tnn.predict( x_test )

# Performace
xg_result_t = mp.ml_error( "XGBoost Tunned", np.expm1( y_test ), np.expm1( yhat␣
 ↪) )
xg_result_t
```

[90]:

	Model Name	MAE	MAPE	RMSE
0	XGBoost Tunned	1251.518232	0.170826	1768.033845

[16]:
```
y_test.head().tolist()
```

[16]:
```
[8.568646473005153,
 8.71028982137815,
 9.025816391627028,
 9.54652685348758,
 8.481151420068972]
```

[17]:
```
yhat[:5].tolist()
```

[17]:
```
[8.429130554199219,
 8.647663116455078,
```

```
 9.102029800415039,
 8.884071350097656,
 8.493669509887695]
```

[26]:
```
print(f'Diferença entre o Máximo Valor: {np.expm1( y_test.max() ) - np.expm1( yhat.max() ):.2f}' )
```

```
Diferença entre o Máximo Valor: 14998.21
```

[34]:
```
print(f'MPE: {mean_percentage_error( y_test, yhat ):.5f}' )
```

```
MPE: 0.00113
```

[103]:
```
# Save best model on Pickle
pickle.dump(xg_tnn, open("model_xgb_t.pkl", "wb"))
```

3.9.3 8.3. Model Performace (Split -> Prepare)

[116]:
```
xg_tnn = XGBRegressor( n_jobs = -1,
                       n_estimators = param_tunned['n_estimators'],
                       eta    = param_tunned['eta'],
                       gamma = param_tunned['gamma'],
                       max_depth = param_tunned['max_depth'],
                       subsample = param_tunned['subsample'],
                       min_child_weight = param_tunned['min_child_weight'],
                       colsample_bytree = param_tunned['colsample_bytree']).
 ↪fit( x_train_, y_train )

# Predict
yhat = xg_tnn.predict( x_test_ )

# Performace
xg_result_t = mp.ml_error( "XGBoost Tunned", np.expm1( y_test ), np.expm1( yhat
 ↪) )
xg_result_t
```

[116]:

	Model Name	MAE	MAPE	RMSE
0	XGBoost Tunned	711.385696	0.103529	1032.75967

[117]:
```
y_test.head().tolist()
```

[117]:
```
[8.568646473005153,
 8.71028982137815,
 9.025816391627028,
 9.54652685348758,
 8.481151420068972]
```

[118]:
```
yhat[:5].tolist()
```

[118]:
```
[8.602458953857422,
 8.686539649963379,
 9.136541366577148,
 9.300227165222168,
 8.644275665283203]
```

[18]:
```
pickle.dump(xg_tnn, open("model_xgb_t2.pkl", "wb"))
```

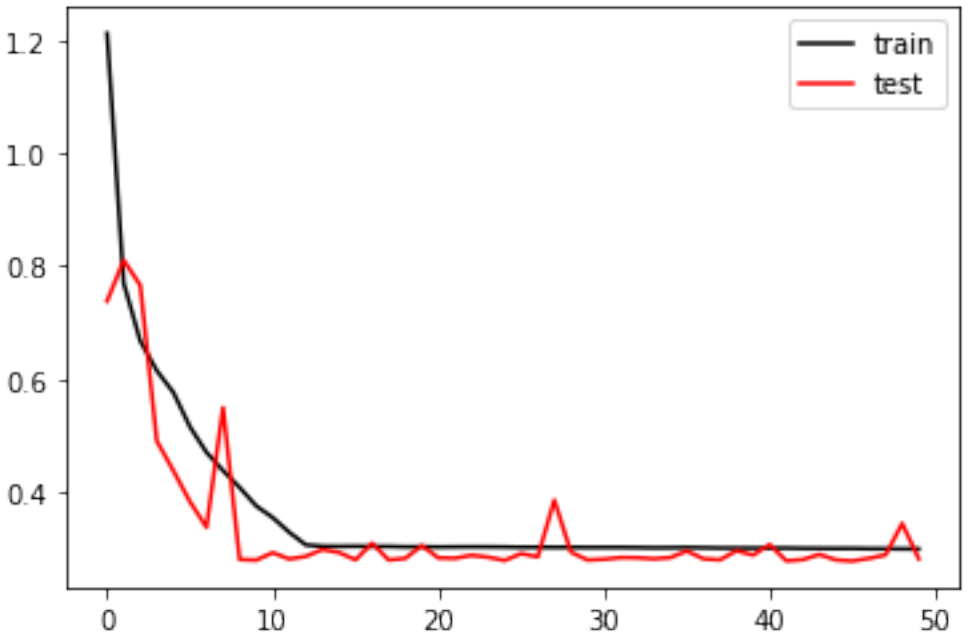
3.9.4 8.4. NN Regressor

[15]:
```
# NN Dataset Preparation
# Comment Below ----------
# x_test['promo2_since_week']  = mms.fit_transform(␣
 ↪x_test[['promo2_since_week']].values )
# x_test['promo2_since_week']  = mms.fit_transform(␣
 ↪x_test[['promo2_since_week']].values )
# x_train['promo2_since_year'] = mms.fit_transform(␣
 ↪x_train[['promo2_since_year']].values )
# x_train['promo2_since_year'] = mms.fit_transform(␣
 ↪x_train[['promo2_since_year']].values )

y_test_n = np.array( y_test ).reshape( -1, 1 )
y_train_n = np.array( y_train ).reshape(-1, 1 )

x_test_n = x_test.values
x_train_n = x_train.values
```

[33]:
```
# Model Definition
model = Sequential()
model.add( Dense( 25, input_dim=20, activation='relu' ) )
model.add( Dense( 1, activation='linear') )

# Model Compile
model.compile( loss='mean_absolute_error', optimizer='adam', metrics=['mse'] )␣
 ↪# Test RMSE for metric ***

# Model Training
history = model.fit( x_train_n, y_train_n, validation_data=(x_test_n,␣
 ↪y_test_n), epochs=50 )

# Model Validation
_, train_mse = model.evaluate( x_train_n, y_train_n )
_, test_mse  = model.evaluate( x_test_n, y_test_n )
```

```
# Model Performace
yhat_nn = model.predict( x_test_n )
yhat_nn = [p[0] for p in yhat_nn]
```

[17]:
```
plt.plot( history.history['loss'], label='train', color='k'  )
plt.plot( history.history['val_loss'], label='test', color='r'  )
plt.legend();
```



[19]:
```
mp.ml_error( "Neural Network Regressor", np.expm1( [p[0] for p in y_test_n] ),␣
↪np.expm1( yhat_nn ) )
```

[19]:

	Model Name	MAE	MAPE	RMSE
0	Neural Network Regressor	1913.687997	0.282804	2787.561813

[34]:
```
# Weights
model.save_weights( 'model_nn.h5' )

# Json Model File
model_jns = model.to_json()
with open('model_nn.json', 'w') as js_m:
    js_m.write( model_jns )
js_m.close()
```

3.10 9.0. Bussiness Model Performace

[40]:
```
df9 = X_test.copy()

df9['prediction'] = np.expm1( yhat )
```

3.10.1 9.1. Bussiness Performace

[267]:
```
# Sum of Predictions
df91 = df9[['store', 'prediction']].groupby( 'store' ).sum().reset_index()

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

df9_aux3 = pd.merge( df9_aux1, df9_aux2, on='store', how='inner' )
df92 = pd.merge( df91, df9_aux3, on='store', how='inner' )

# Ssenarios
df92 = mp.create_scenarios( df92 )

df92 = df92[['store', 'prediction', 'worst_scenario', 'best_scenario', 'MAE',␣
 ↪'MAPE']]
```

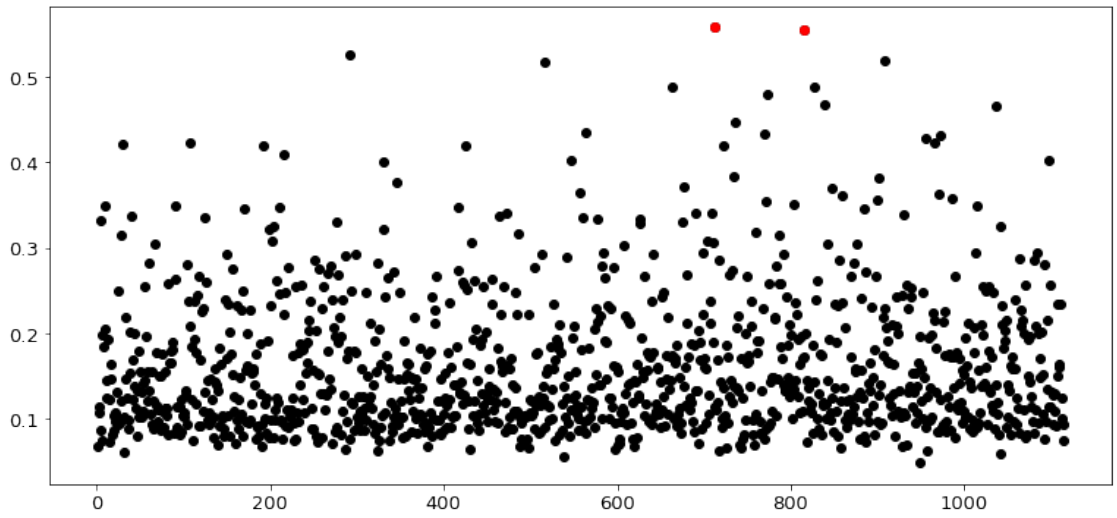
[272]:
```
df92[df92['MAPE'] > 0.55]
```

[272]:

	store	prediction	worst_scenario	best_scenario	MAE	MAPE
712	713	502075.91	497508.61	506643.2	4567.294948	0.557384
814	815	359472.34	356110.39	362834.3	3361.955593	0.555540

Shops harder to forecast sales: - Store 713 with MAPE: 0.557 - Store 815 with MAPE: 0.555

[271]:
```
plt.scatter( df92['store'], df92['MAPE'], c='k' )
for i in [713, 815]:
    plt.scatter( df92[df92['store'] == i]['store'], df92[df92['store'] ==␣
 ↪i]['MAPE'], c='r' )
```



3.10.2 9.2. Total Performace

[273]:
```
df93 = df92[['prediction', 'worst_scenario', 'best_scenario']].apply( lambda x:␣
↪np.sum( x ), axis=0 ).reset_index().rename( columns={'index': 'Scenario', 0:␣
↪'Total Sales'} )
df93['Total Sales'] = df93['Total Sales'].map( 'R$ {:,.2f}'.format )
df93['Scenario'] = ['Predictions', 'Worst Scenario', 'Best Scenario']

df93
```

[273]:

	Scenario	Total Sales
0	Predictions	R$ 276,712,777.60
1	Worst Scenario	R$ 275,455,320.37
2	Best Scenario	R$ 277,970,234.33

3.10.3 9.3. ML Performace

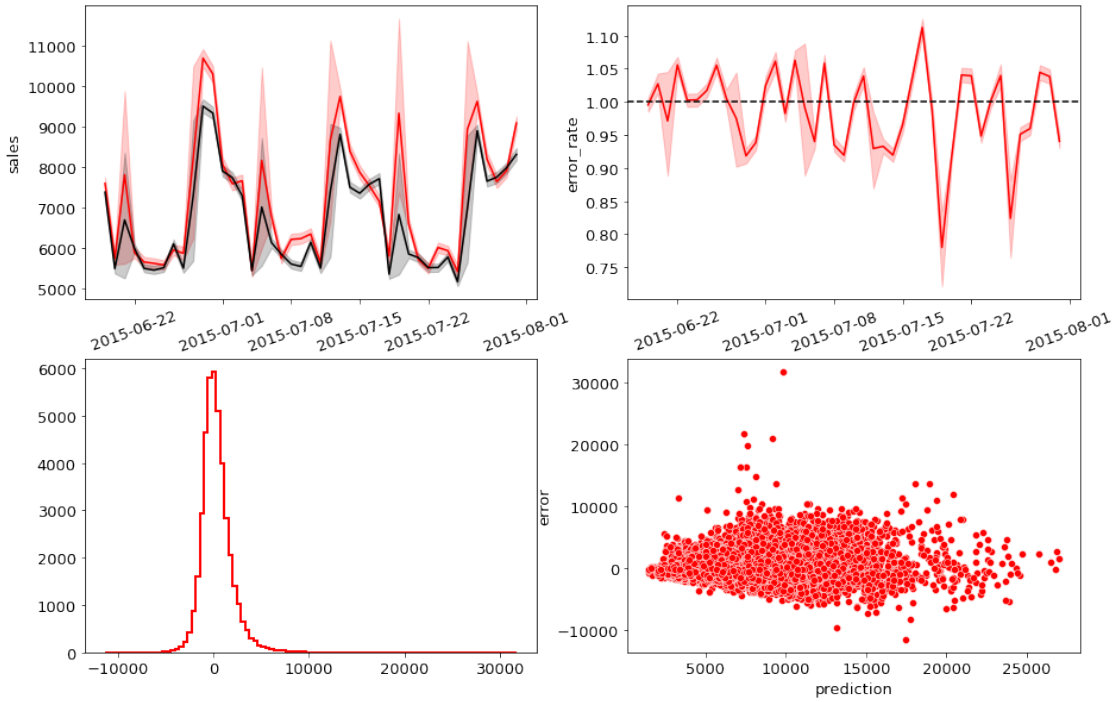
[274]:
```
df9['error'] = df9['sales'] - df9['prediction']
df9['error_rate'] = df9['prediction'] / df9['sales']
```

[353]:
```
df9['error_nn'] = df9['sales'] - np.expm1(yhat_nn)
df9['error_rate_nn'] = np.expm1(yhat_nn) / df9['sales']
```

9.3.1. XGBoost Performace

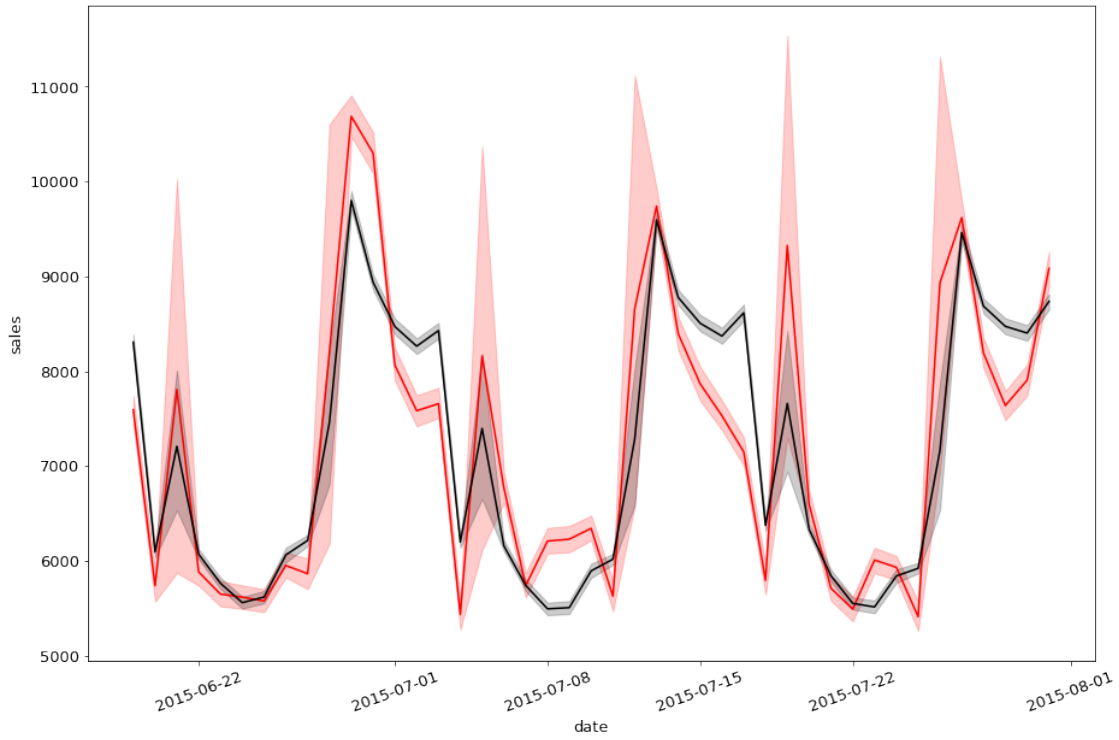
[329]:
```
plt.subplot( 2, 2, 1 )
mp.plot_pred_sales( df9['date'], df9['sales'], df9['prediction'] )

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

```
mp.plot_error_rate( df9['date'], df9['error_rate'] )

plt.subplot( 2, 2, 3 )
plt.hist( df9['error'], **mp.args(bins=100) )

plt.subplot( 2, 2, 4 )
sns.scatterplot( df9['prediction'], df9['error'], color='r' );
```



9.3.2. Neural Network Performace

[365]:
```
plot_pred_sales( df9['date'], df9['sales'], np.expm1( yhat_nn ) )
```



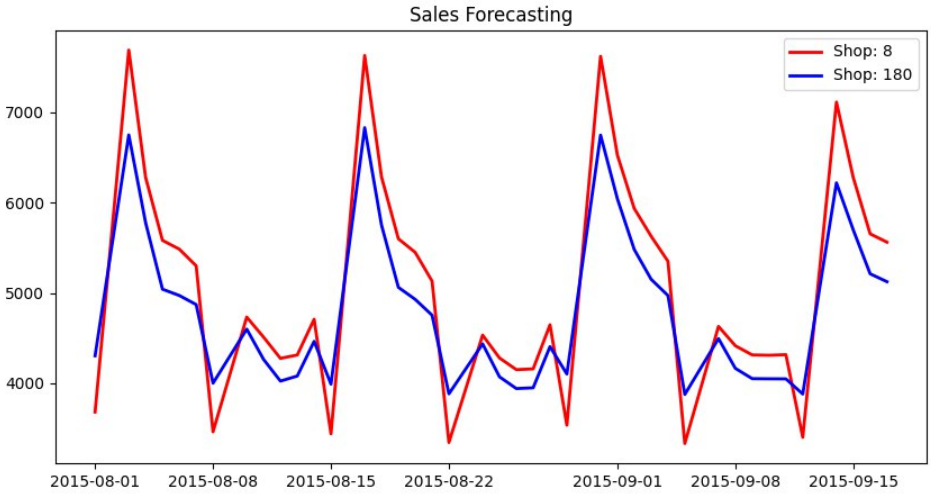
4 Conclusão e Demonstração

[33]:
```
Image( 'Img1.png' )
```

[33]:

[36]:
```
Image( 'img2.jpg' )
```

[36]:



5 Próximos Passos

0. Encinar o Diretor a Utilizar a Aplicação.
1. Entender melhor o negócio para criar novas Features.
2. Utilizar outras técnicas de preparação das Features.
3. Tunar melhor os Modelos de ML.
4. Desenvolver o App Executável.