

m03_v01_store_sales_prediction

September 12, 2021

1 0.0. IMPORTS

```
[1]: import math
import numpy as np
import pandas as pd
import inflection

import seaborn as sns

from matplotlib import pyplot as plt
from IPython.core.display import HTML
from IPython.display import Image
```

1.1 0.1. Helper Functions

```
[2]: 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
575825      156           5  2014-01-31    6454          714      1      0

```

0	0	a	a	2020.0	
2.0		2011.0	1	14.0	2011.0
Mar, Jun, Sept, Dec					

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

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

```

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 Attributes

```

[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

```

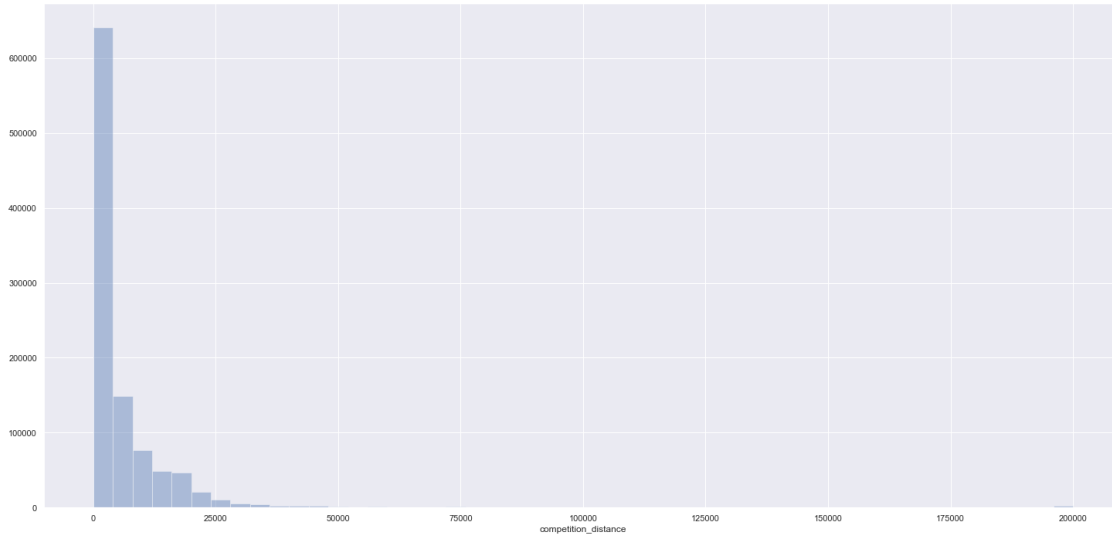
```
# concatenar
m = pd.concat( [d2, d3, d4, ct1, ct2, d1, d5, d6] ).T.reset_index()
m.columns = ['attributes', 'min', 'max', 'range', 'mean', 'median', 'std', 'skew', 'kurtosis']
m
```

```
[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 0x162de2370>
```



2.7.2 1.7.2. Categorical Attributes

```
[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
```

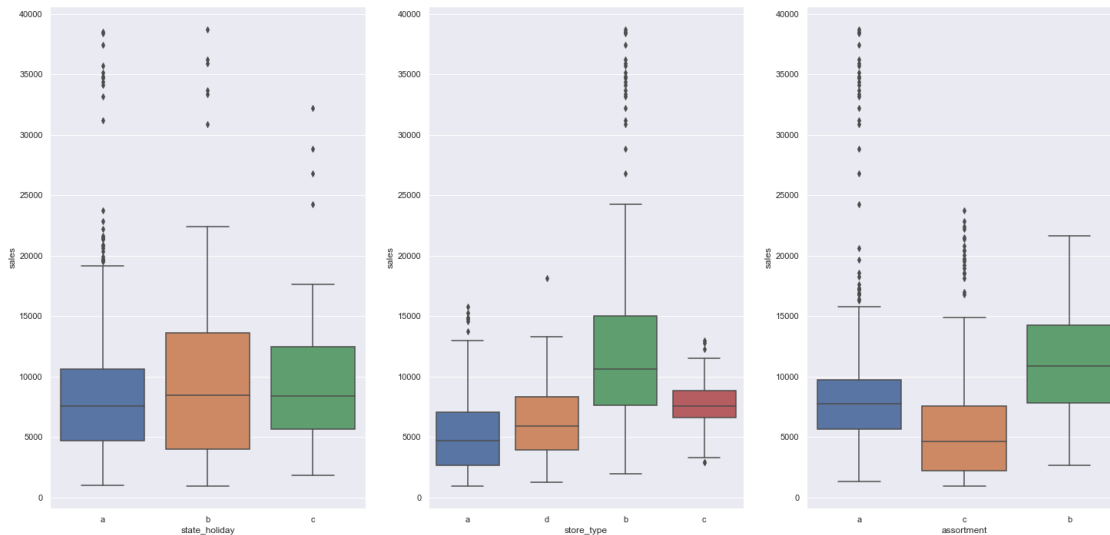
```
[18]: aux = df1[(df1['state_holiday'] != '0') & (df1['sales'] > 0)]

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 0x106f06dc0>
```



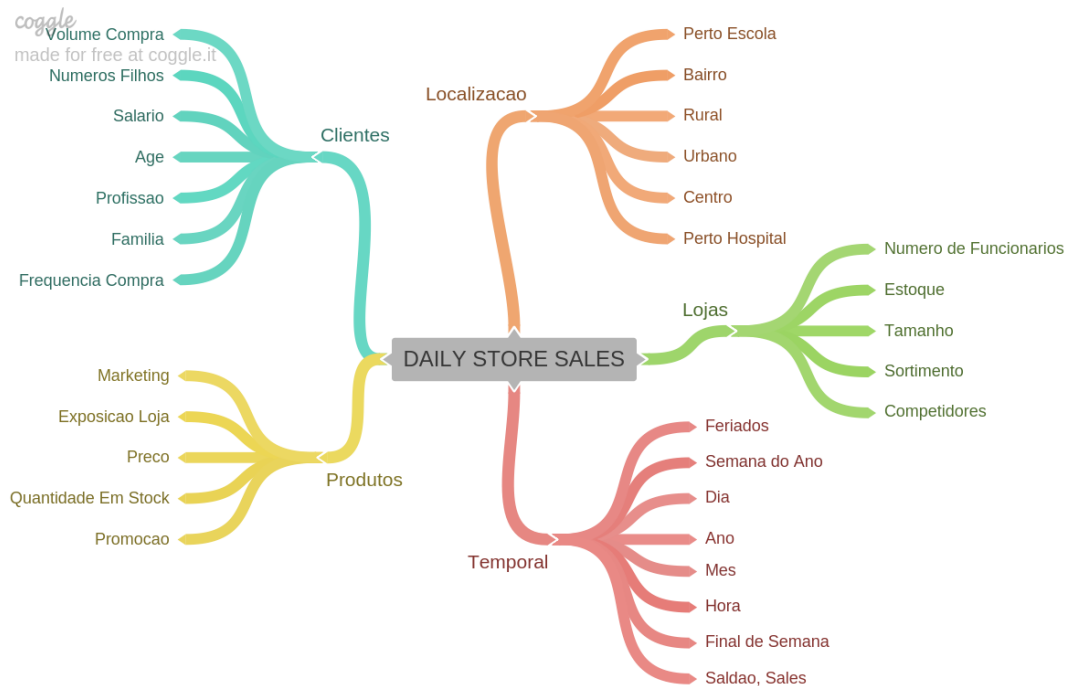
3 2.0. PASSO 02 - FEATURE ENGINEERING

```
[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.
    ↳strftime( 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 )
```