

# 1 Introduction

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

The idea here was to create a centralized pattern method to predict the next six weeks, using data science tools, like machine learning to read the variables and get their behavior comparing their correlation, and historic, then creating a sales prediction up to six weeks ahead, with the best and worst scenario furthermore the percentage that how accurate is the prediction, because it depends on each store.

Having the final result, the CEO could choose which store to invest more based on how much and precise the results about the store would be.

## 1.1 Imports

```
In [113]: 1 # List of all libraries used in the project
          2
          3 import math
          4 import numpy as np
          5 import pandas as pd
          6 import random
          7 import pickle
          8 import requests
          9 import warnings
         10 import inflection
         11 import seaborn as sns
         12 import xgboost as xgb
         13 import datetime
         14
         15 from tabulate import tabulate
         16 from scipy import stats as ss
         17 from boruta import BorutaPy
         18 from matplotlib import pyplot as plt
         19 from IPython.display import Image
         20 from IPython.core.display import HTML
         21
         22 from sklearn.metrics import mean_absolute_error, mean_absolute_per
         23 from sklearn.ensemble import RandomForestRegressor
         24 from sklearn.linear_model import LinearRegression, Lasso
         25 from sklearn.preprocessing import RobustScaler, MinMaxScaler, LabelEncod
         26
```

executed in 456ms, finished 10:38:55 2023-04-16

## 1.2 Understanding the Libraries

1. **math**: This library provides mathematical functions and constants in Python. It can be useful for performing mathematical operations in a program.
2. **numpy**: This library provides functions for working with arrays and matrices in Python. It can be useful for numerical computations and data analysis.
3. **pandas**: This library provides data structures and functions for working with tabular data in Python. It can be useful for data cleaning, manipulation, and analysis.
4. **random**: This library provides functions for generating random numbers and sequences. It can be useful for generating random inputs to test functions or simulating random events in a program.
5. **pickle**: This library provides functions for serializing and deserializing Python objects. It can be used to save objects to a file or send them over a network connection.
6. **requests**: This library provides functions for making HTTP requests from Python. It can be used to interact with web APIs or download files from the internet.
7. **warnings**: This library provides a way to issue warnings from Python code. It can be used to alert users of potential issues or deprecated functionality in a library.
8. **inflection**: This library provides functions for transforming strings to different cases. It can be useful for formatting column names in pandas dataframes or other data cleaning tasks.
9. **seaborn**: This library is used for creating visualizations in Python. It provides a high-level interface for creating statistical graphics, such as heatmaps, scatter plots, and bar charts.
10. **xgboost**: This library is used for building gradient boosted trees, which are a popular machine learning algorithm for regression and classification tasks.
11. **datetime**: This library provides classes for working with dates and times in Python.
12. **tabulate**: This library is used for creating tables in Python. It can create tables from various data sources, including lists, dictionaries, and pandas dataframes.
13. **scipy**: This library contains a wide range of scientific computing functions, including algorithms for optimization, signal processing, linear algebra, and more.
14. **boruta**: This library is used for feature selection in machine learning. It employs a random forest algorithm to evaluate the importance of each feature and determine whether it should be included in the final model.
15. **matplotlib**: This library is used for creating visualizations in Python. It provides a wide range of plotting functions for creating line plots, scatter plots, bar charts, and more.
16. **IPython.display**: This module provides a way to display rich media in the Jupyter Notebook environment.

17. **IPython.core.display:** This module contains the same functions as IPython.display, but is intended for use in IPython extensions and other low-level code.
18. **sklearn.metrics:** This library contains various metrics and evaluation techniques for machine learning models.
19. **sklearn.ensemble:** This library contains ensemble learning algorithms such as random forests, bagging, and boosting.
20. **sklearn.linear\_model:** This library contains various linear regression and classification models.
21. **sklearn.preprocessing:** This library contains various data preprocessing techniques such as scaling, normalization, and imputation.

## 1.3 Help Functions

```

In [2]: 1 # This function was created to validate the model, checking the differer
        2
        3 '''
        4 x_training: a pandas dataframe containing the training dataset
        5 kfold: an integer value representing the number of folds to be used in k
        6 model_name: a string representing the name of the machine learning model
        7 model: a machine learning model object that can be fitted to the data
        8 verbose: a boolean indicating whether or not to display progress updates
        9 '''
       10
       11 def cross_validation( x_training, kfold, model_name, model, verbose=False):
       12     mae_list = []
       13     mape_list = []
       14     rmse_list = []
       15
       16     '''
       17     The function first creates empty lists to store the performance metrics
       18     k-fold values in reverse order, starting from the highest value and
       19     validation datasets based on the start and end dates of the validation
       20     '''
       21
       22     for k in reversed( range( 1, kfold+1 ) ):
       23         if verbose:
       24             print( '\nKFold Number: {}'.format( k ) )
       25
       26         # start and end date for validation
       27         validation_start_date = x_training['date'].max() - datetime.timedelta(days=k)
       28         validation_end_date = x_training['date'].max() - datetime.timedelta(days=0)
       29
       30         '''
       31         Next, it separates the features and target variables for both the training
       32         on the training set and predicts the target variable for the validation set
       33         the ml_error function.
       34         '''
       35
       36         # filtering dataset
       37         training = x_training[x_training['date'] < validation_start_date]
       38         validation = x_training[( x_training['date'] >= validation_start_date) & ( x_training['date'] < validation_end_date)]
       39
       40         # training and validation dataset
       41         xtraining = training.drop( ['date', 'sales'], axis=1 )
       42         ytraining = training['sales']
       43
       44         # validation
       45         xvalidation = validation.drop( ['date', 'sales'], axis=1 )
       46         yvalidation = validation['sales']
       47
       48         # model
       49         m = model.fit( xtraining, ytraining )
       50
       51         # prediction
       52         yhat = m.predict( xvalidation )
       53
       54         # performance
       55         m_result = ml_error( model_name, np.exp(1)( yvalidation ), np.exp(1)( yhat ))

```

```

56
57     # store performance of each kfold iteration
58     mae_list.append( m_result['MAE'] )
59     mape_list.append( m_result['MAPE'] )
60     rmse_list.append( m_result['RMSE'] )
61
62     '''
63     The mean and standard deviation of the performance metrics for a
64     dataframe. The performance metrics include the mean absolute error
65     root mean squared error (RMSE)
66     '''
67
68     return pd.DataFrame( {'Model Name': model_name,
69                           'MAE CV': np.round( np.mean( mae_list ), 2 ).asf
70                           ' +/- ' + np.round( np.std( mae_list ), 2 ).asf
71                           'MAPE CV': np.round( np.mean( mape_list ), 2 )
72                           ' +/- ' + np.round( np.std( mape_list ), 2 ).asf
73                           'RMSE CV': np.round( np.mean( rmse_list ), 2 )
74                           ' +/- ' + np.round( np.std( rmse_list ), 2 ).asf
75
76     # Today we have all the 3 functions insides the module sklearn.metrics
77
78     '''
79     mae = mean_absolute_error(y_true, y_pred)
80     rmse = np.sqrt(mean_squared_error(y_true, y_pred))
81     '''
82
83     def mean_percentage_error( y, yhat ):
84         return np.mean( ( y - yhat ) / y )
85
86     def mean_absolute_percentage_error( y, yhat ):
87         return np.mean( np.abs( ( y - yhat ) / y ) )
88
89     def ml_error( model_name, y, yhat ):
90         mae = mean_absolute_error( y, yhat )
91         mape = mean_absolute_percentage_error( y, yhat )
92         rmse = np.sqrt( mean_squared_error( y, yhat ) )
93         return pd.DataFrame( { 'Model Name': model_name,
94                               'MAE': mae,
95                               'MAPE': mape,
96                               'RMSE': rmse }, index=[0] )
97
98     '''
99     The function cramer_v() calculates the Cramer's V correlation coefficient
100     Cramer's V is a measure of association between two nominal variables the
101     where 0 indicates no association and 1 indicates complete association.
102     '''
103
104     def cramer_v( x, y ):
105
106         '''
107         The function first creates a contingency table cm using pd.crosstab()
108         occurrence of each combination of categories for the two variables.
109         '''
110
111         cm = pd.crosstab( x, y ).values

```

```

112     n = cm.sum()
113     r, k = cm.shape
114
115     '''
116     Then it calculates the chi-square statistic using ss.chi2_contingency
117     for small sample sizes.
118     '''
119
120     chi2 = ss.chi2_contingency( cm )[0]
121     chi2corr = max(0, chi2 - (k-1) * (r-1) / (n-1))
122
123     '''
124     Finally, it calculates the Cramer's V coefficient by dividing the chi2corr
125     the corrected number of rows and columns minus 1. The result is returned.
126     '''
127
128     kcorr = k - (k-1) ** 2 / (n-1)
129     rcorr = r - (r-1) ** 2 / (n-1)
130
131     return np.sqrt((chi2corr/n) / (min(kcorr-1, rcorr - 1)))
132
133     # This Function create a better visual set to Jupyter
134
135     def jupyter_settings():
136         %matplotlib inline
137         %pylab inline
138
139         plt.style.use( 'bmh' )
140         plt.rcParams['figure.figsize'] = [25, 12]
141         plt.rcParams['font.size'] = 24
142
143         display( HTML( '<style>.container { width:100% !important; }</style>' ) )
144         pd.options.display.max_columns = None
145         pd.options.display.max_rows = None
146         pd.set_option( 'display.expand_frame_repr', False )
147
148         sns.set()
149
150     jupyter_settings()

```

executed in 312ms, finished 05:18:00 2023-04-12  
 %pylab is deprecated, use %matplotlib inline and import the required libraries.

Populating the interactive namespace from numpy and matplotlib

## 1.4 Load the Data

```

In [3]: 1 # Read the archive and save in the memory
        2
        3 df_sales_raw = pd.read_csv(r'C:\train.csv', low_memory=False)
        4 df_store_raw = pd.read_csv(r'C:\store.csv', low_memory=False)
        5
        6 # Low memory indicates if you have enough memory to read all the data by
        7

```

## 1.5 Checking how the both datasets are to do a Merge based on their columns

In [4]: executed in 242ms, finished 05:18:01 2023-04-12

Out[4]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1
3	4	5	2015-07-31	13995	1498	1	1	0	1
4	5	5	2015-07-31	4822	559	1	1	0	1

In [5]: executed in 60ms, finished 05:18:01 2023-04-12

Out[5]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear
0	1	c	a	1270.0	9.0	2013
1	2	a	a	570.0	11.0	2013
2	3	a	a	14130.0	12.0	2013
3	4	c	c	620.0	9.0	2013
4	5	a	a	29910.0	4.0	2013

In [6]:

```

1 # Merging
2
3 df_raw = pd.merge( df_sales_raw, df_store_raw, how='left', on='Store')

```

executed in 553ms, finished 05:18:02 2023-04-12

In [7]:

```

1 # Checking the result
2
3 df_raw.head()

```

executed in 44ms, finished 05:18:02 2023-04-12

Out[7]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1
3	4	5	2015-07-31	13995	1498	1	1	0	1
4	5	5	2015-07-31	4822	559	1	1	0	1

## 2 Describing the Data (Step One)

```
In [8]: 1 # Always create a checkpoint to not lose the entire progress, and avoid
        2
        3 df1 = df_raw.copy()
        4
        executed in 188ms, finished 05:18:02 2023-04-12
```

## 2.1 Rename Columns

```
In [9]: 1 # Get the name of all columns
        2
        3 df1.columns
        4
        executed in 12ms, finished 05:18:02 2023-04-12
```

```
Out[9]: Index(['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open', 'Promo',
              'StateHoliday', 'SchoolHoliday', 'StoreType', 'Assortment',
              'CompetitionDistance', 'CompetitionOpenSinceMonth',
              'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWeek',
              'Promo2SinceYear', 'PromoInterval'],
             dtype='object')
```

```
In [10]: 1 # Rename columns makes it easier to us to access the data later.
        2
        3 cols_old = ['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open',
        4           'StateHoliday', 'SchoolHoliday', 'StoreType', 'Assortment',
        5           'CompetitionDistance', 'CompetitionOpenSinceMonth',
        6           'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWeek',
        7           'Promo2SinceYear', 'PromoInterval']
        8
        9 # The function inflection will create the snakecase pattern in the column
        10
        11 snakecase = lambda x: inflection.underscore( x )
        12
        13 # Now we use map to apply the function in all columns and then add it to
        14
        15 cols_new = list( map( snakecase, cols_old ) )
        16
        17 # Rename columns
        18
        19 df1.columns = cols_new
        20
        21
        22
        23
        24
        25
        26
        27
        28
        29
        30
        31
        32
        33
        34
        35
        36
        37
        38
        39
        40
        41
        42
        43
        44
        45
        46
        47
        48
        49
        50
        51
        52
        53
        54
        55
        56
        57
        58
        59
        60
        61
        62
        63
        64
        65
        66
        67
        68
        69
        70
        71
        72
        73
        74
        75
        76
        77
        78
        79
        80
        81
        82
        83
        84
        85
        86
        87
        88
        89
        90
        91
        92
        93
        94
        95
        96
        97
        98
        99
        100
        101
        102
        103
        104
        105
        106
        107
        108
        109
        110
        111
        112
        113
        114
        115
        116
        117
        118
        119
        120
        121
        122
        123
        124
        125
        126
        127
        128
        129
        130
        131
        132
        133
        134
        135
        136
        137
        138
        139
        140
        141
        142
        143
        144
        145
        146
        147
        148
        149
        150
        151
        152
        153
        154
        155
        156
        157
        158
        159
        160
        161
        162
        163
        164
        165
        166
        167
        168
        169
        170
        171
        172
        173
        174
        175
        176
        177
        178
        179
        180
        181
        182
        183
        184
        185
        186
        187
        188
        189
        190
        191
        192
        193
        194
        195
        196
        197
        198
        199
        200
        201
        202
        203
        204
        205
        206
        207
        208
        209
        210
        211
        212
        213
        214
        215
        216
        217
        218
        219
        220
        221
        222
        223
        224
        225
        226
        227
        228
        229
        230
        231
        232
        233
        234
        235
        236
        237
        238
        239
        240
        241
        242
        243
        244
        245
        246
        247
        248
        249
        250
        251
        252
        253
        254
        255
        256
        257
        258
        259
        260
        261
        262
        263
        264
        265
        266
        267
        268
        269
        270
        271
        272
        273
        274
        275
        276
        277
        278
        279
        280
        281
        282
        283
        284
        285
        286
        287
        288
        289
        290
        291
        292
        293
        294
        295
        296
        297
        298
        299
        300
        301
        302
        303
        304
        305
        306
        307
        308
        309
        310
        311
        312
        313
        314
        315
        316
        317
        318
        319
        320
        321
        322
        323
        324
        325
        326
        327
        328
        329
        330
        331
        332
        333
        334
        335
        336
        337
        338
        339
        340
        341
        342
        343
        344
        345
        346
        347
        348
        349
        350
        351
        352
        353
        354
        355
        356
        357
        358
        359
        360
        361
        362
        363
        364
        365
        366
        367
        368
        369
        370
        371
        372
        373
        374
        375
        376
        377
        378
        379
        380
        381
        382
        383
        384
        385
        386
        387
        388
        389
        390
        391
        392
        393
        394
        395
        396
        397
        398
        399
        400
        401
        402
        403
        404
        405
        406
        407
        408
        409
        410
        411
        412
        413
        414
        415
        416
        417
        418
        419
        420
        421
        422
        423
        424
        425
        426
        427
        428
        429
        430
        431
        432
        433
        434
        435
        436
        437
        438
        439
        440
        441
        442
        443
        444
        445
        446
        447
        448
        449
        450
        451
        452
        453
        454
        455
        456
        457
        458
        459
        460
        461
        462
        463
        464
        465
        466
        467
        468
        469
        470
        471
        472
        473
        474
        475
        476
        477
        478
        479
        480
        481
        482
        483
        484
        485
        486
        487
        488
        489
        490
        491
        492
        493
        494
        495
        496
        497
        498
        499
        500
        501
        502
        503
        504
        505
        506
        507
        508
        509
        510
        511
        512
        513
        514
        515
        516
        517
        518
        519
        520
        521
        522
        523
        524
        525
        526
        527
        528
        529
        530
        531
        532
        533
        534
        535
        536
        537
        538
        539
        540
        541
        542
        543
        544
        545
        546
        547
        548
        549
        550
        551
        552
        553
        554
        555
        556
        557
        558
        559
        560
        561
        562
        563
        564
        565
        566
        567
        568
        569
        570
        571
        572
        573
        574
        575
        576
        577
        578
        579
        580
        581
        582
        583
        584
        585
        586
        587
        588
        589
        590
        591
        592
        593
        594
        595
        596
        597
        598
        599
        600
        601
        602
        603
        604
        605
        606
        607
        608
        609
        610
        611
        612
        613
        614
        615
        616
        617
        618
        619
        620
        621
        622
        623
        624
        625
        626
        627
        628
        629
        630
        631
        632
        633
        634
        635
        636
        637
        638
        639
        640
        641
        642
        643
        644
        645
        646
        647
        648
        649
        650
        651
        652
        653
        654
        655
        656
        657
        658
        659
        660
        661
        662
        663
        664
        665
        666
        667
        668
        669
        670
        671
        672
        673
        674
        675
        676
        677
        678
        679
        680
        681
        682
        683
        684
        685
        686
        687
        688
        689
        690
        691
        692
        693
        694
        695
        696
        697
        698
        699
        700
        701
        702
        703
        704
        705
        706
        707
        708
        709
        710
        711
        712
        713
        714
        715
        716
        717
        718
        719
        720
        721
        722
        723
        724
        725
        726
        727
        728
        729
        730
        731
        732
        733
        734
        735
        736
        737
        738
        739
        740
        741
        742
        743
        744
        745
        746
        747
        748
        749
        750
        751
        752
        753
        754
        755
        756
        757
        758
        759
        760
        761
        762
        763
        764
        765
        766
        767
        768
        769
        770
        771
        772
        773
        774
        775
        776
        777
        778
        779
        780
        781
        782
        783
        784
        785
        786
        787
        788
        789
        790
        791
        792
        793
        794
        795
        796
        797
        798
        799
        800
        801
        802
        803
        804
        805
        806
        807
        808
        809
        810
        811
        812
        813
        814
        815
        816
        817
        818
        819
        820
        821
        822
        823
        824
        825
        826
        827
        828
        829
        830
        831
        832
        833
        834
        835
        836
        837
        838
        839
        840
        841
        842
        843
        844
        845
        846
        847
        848
        849
        850
        851
        852
        853
        854
        855
        856
        857
        858
        859
        860
        861
        862
        863
        864
        865
        866
        867
        868
        869
        870
        871
        872
        873
        874
        875
        876
        877
        878
        879
        880
        881
        882
        883
        884
        885
        886
        887
        888
        889
        890
        891
        892
        893
        894
        895
        896
        897
        898
        899
        900
        901
        902
        903
        904
        905
        906
        907
        908
        909
        910
        911
        912
        913
        914
        915
        916
        917
        918
        919
        920
        921
        922
        923
        924
        925
        926
        927
        928
        929
        930
        931
        932
        933
        934
        935
        936
        937
        938
        939
        940
        941
        942
        943
        944
        945
        946
        947
        948
        949
        950
        951
        952
        953
        954
        955
        956
        957
        958
        959
        960
        961
        962
        963
        964
        965
        966
        967
        968
        969
        970
        971
        972
        973
        974
        975
        976
        977
        978
        979
        980
        981
        982
        983
        984
        985
        986
        987
        988
        989
        990
        991
        992
        993
        994
        995
        996
        997
        998
        999
        1000
        1001
        1002
        1003
        1004
        1005
        1006
        1007
        1008
        1009
        1010
        1011
        1012
        1013
        1014
        1015
        1016
        1017
        1018
        1019
        1020
        1021
        1022
        1023
        1024
        1025
        1026
        1027
        1028
        1029
        1030
        1031
        1032
        1033
        1034
        1035
        1036
        1037
        1038
        1039
        1040
        1041
        1042
        1043
        1044
        1045
        1046
        1047
        1048
        1049
        1050
        1051
        1052
        1053
        1054
        1055
        1056
        1057
        1058
        1059
        1060
        1061
        1062
        1063
        1064
        1065
        1066
        1067
        1068
        1069
        1070
        1071
        1072
        1073
        1074
        1075
        1076
        1077
        1078
        1079
        1080
        1081
        1082
        1083
        1084
        1085
        1086
        1087
        1088
        1089
        1090
        1091
        1092
        1093
        1094
        1095
        1096
        1097
        1098
        1099
        1100
        1101
        1102
        1103
        1104
        1105
        1106
        1107
        1108
        1109
        1110
        1111
        1112
        1113
        1114
        1115
        1116
        1117
        1118
        1119
        1120
        1121
        1122
        1123
        1124
        1125
        1126
        1127
        1128
        1129
        1130
        1131
        1132
        1133
        1134
        1135
        1136
        1137
        1138
        1139
        1140
        1141
        1142
        1143
        1144
        1145
        1146
        1147
        1148
        1149
        1150
        1151
        1152
        1153
        1154
        1155
        1156
        1157
        1158
        1159
        1160
        1161
        1162
        1163
        1164
        1165
        1166
        1167
        1168
        1169
        1170
        1171
        1172
        1173
        1174
        1175
        1176
        1177
        1178
        1179
        1180
        1181
        1182
        1183
        1184
        1185
        1186
        1187
        1188
        1189
        1190
        1191
        1192
        1193
        1194
        1195
        1196
        1197
        1198
        1199
        1200
        1201
        1202
        1203
        1204
        1205
        1206
        1207
        1208
        1209
        1210
        1211
        1212
        1213
        1214
        1215
        1216
        1217
        1218
        1219
        1220
        1221
        1222
        1223
        1224
        1225
        1226
        1227
        1228
        1229
        1230
        1231
        1232
        1233
        1234
        1235
        1236
        1237
        1238
        1239
        1240
        1241
        1242
        1243
        1244
        1245
        1246
        1247
        1248
        1249
        1250
        1251
        1252
        1253
        1254
        1255
        1256
        1257
        1258
        1259
        1260
        1261
        1262
        1263
        1264
        1265
        1266
        1267
        1268
        1269
        1270
        1271
        1272
        1273
        1274
        1275
        1276
        1277
        1278
        1279
        1280
        1281
        1282
        1283
        1284
        1285
        1286
        1287
        1288
        1289
        1290
        1291
        1292
        1293
        1294
        1295
        1296
        1297
        1298
        1299
        1300
        1301
        1302
        1303
        1304
        1305
        1306
        1307
        1308
        1309
        1310
        1311
        1312
        1313
        1314
        1315
        1316
        1317
        1318
        1319
        1320
        1321
        1322
        1323
        1324
        1325
        1326
        1327
        1328
        1329
        1330
        1331
        1332
        1333
        1334
        1335
        1336
        1337
        1338
        1339
        1340
        1341
        1342
        1343
        1344
        1345
        1346
        1347
        1348
        1349
        1350
        1351
        1352
        1353
        1354
        1355
        1356
        1357
        1358
        1359
        1360
        1361
        1362
        1363
        1364
        1365
        1366
        1367
        1368
        1369
        1370
        1371
        1372
        1373
        1374
        1375
        1376
        1377
        1378
        1379
        1380
        1381
        1382
        1383
        1384
        1385
        1386
        1387
        1388
        1389
        1390
        1391
        1392
        1393
        1394
        1395
        1396
        1397
        1398
        1399
        1400
        1401
        1402
        1403
        1404
        1405
        1406
        1407
        1408
        1409
        1410
        1411
        1412
        1413
        1414
        1415
        1416
        1417
        1418
        1419
        1420
        1421
        1422
        1423
        1424
        1425
        1426
        1427
        1428
        1429
        1430
        1431
        1432
        1433
        1434
        1435
        1436
        1437
        1438
        1439
        1440
        1441
        1442
        1443
        1444
        1445
        1446
        1447
        1448
        1449
        1450
        1451
        1452
        1453
        1454
        1455
        1456
        1457
        1458
        1459
        1460
        1461
        1462
        1463
        1464
        1465
        1466
        1467
        1468
        1469
        1470
        1471
        1472
        1473
        1474
        1475
        1476
        1477
        1478
        1479
        1480
        1481
        1482
        1483
        1484
        1485
        1486
        1487
        1488
        1489
        1490
        1491
        1492
        1493
        1494
        1495
        1496
        1497
        1498
        1499
        1500
        1501
        1502
        1503
        1504
        1505
        1506
        1507
        1508
        1509
        1510
        1511
        1512
        1513
        1514
        1515
        1516
        1517
        1518
        1519
        1520
        1521
        1522
        1523
        1524
        1525
        1526
        1527
        1528
        1529
        1530
        1531
        1532
        1533
        1534
        1535
        1536
        1537
        1538
        1539
        1540
        1541
        1542
        1543
        1544
        1545
        1546
        1547
        1548
        1549
        1550
        1551
        1552
        1553
        1554
        1555
        1556
        1557
        1558
        1559
        1560
        1561
        1562
        1563
        1564
        1565
        1566
        1567
        1568
        1569
        1570
        1571
        1572
        1573
        1574
        1575
        1576
        1577
        1578
        1579
        1580
        1581
        1582
        1583
        1584
        1585
        1586
        1587
        1588
        1589
        1590
        1591
        1592
        1593
        1594
        1595
        1596
        1597
        1598
        1599
        1600
        1601
        1602
        1603
        1604
        1605
        1606
        1607
        1608
        1609
        1610
        1611
        1612
        1613
        1614
        1615
        1616
        1617
        1618
        1619
        1620
        1621
        1622
        1623
        1624
        1625
        1626
        1627
        1628
        1629
        1630
        1631
        1632
        1633
        1634
        1635
        1636
        1637
        1638
        1639
        1640
        1641
        1642
        1643
        1644
        1645
        1646
        1647
        1648
        1649
        1650
        1651
        1652
        1653
        1654
        1655
        1656
        1657
        1658
        1659
        1660
        1661
        1662
        1663
        1664
        1665
        1666
        1667
        1668
        1669
        1670
        1671
        1672
        1673
        1674
        1675
        1676
        1677
        1678
        1679
        1680
        1681
        1682
        1683
        1684
        1685
        1686
        1687
        1688
        1689
        1690
        1691
        1692
        1693
        1694
        1695
        1696
        1697
        1698
        1699
        1700
        1701
        1702
        1703
        1704
        1705
        1706
        1707
        1708
        1709
        1710
        1711
        1712
        1713
        1714
        1715
        1716
        1717
        1718
        1719
        1720
        1721
        1722
        1723
        1724
        1725
        1726
        1727
        1728
        1729
        1730
        1731
        1732
        1733
        1734
        1735
        1736
        1737
        1738
        1739
        1740
        1741
        1742
        1743
        1744
        1745
        1746
        1747
        1748
        1749
        1750
        1751
        1752
        1753
        1754
        1755
        1756
        1757
        1758
        1759
        1760
        1761
        1762
        1763
        1764
        1765
        1766
        1767
        1768
        1769
        1770
        1771
        1772
        1773
        1774
        1775
        1776
        1777
        1778
        1779
        1780
        1781
        1782
        1783
        1784
        1785
        1786
        1787
        1788
        1789
        1790
        1791
        1792
        1793
        1794
        1795
        1796
        1797
        1798
        1799
        1800
        1801
        1802
        1803
        1804
        1805
        1806
        1807
        1808
        1809
        1810
        1811
        1812
        1813
        1814
        1815
        1816
        1817
        1818
        1819
        1820
        1821
        1822
        1823
        1824
        1825
        1826
        1827
        1828
        1829
        1830
        1831
        1832
        1833
       
```



## 2.2 Data Dimensions

```
In [12]: 1 # Find out how big the dataset is
          2
          3 print(f'Number of Rows: {df1.shape[0]}')
          4 print(f'Number of Columns: {df1.shape[1]}')
```

executed in 28ms, finished 05:18:02 2023-04-12

Number of Rows: 1017209

Number of Columns: 18

## 2.3 Data Types

```
In [13]: 1 # Checking the type of each column to identify possible changes
          2
          3 df1.dtypes
```

executed in 12ms, finished 05:18:02 2023-04-12

```
Out[13]: store                int64
          day_of_week         int64
          date                object
          sales               int64
          customers           int64
          open                int64
          promo               int64
          state_holiday       object
          school_holiday      int64
          store_type          object
          assortment          object
          competition_distance float64
          competition_open_since_month float64
          competition_open_since_year float64
          promo2              int64
          promo2_since_week   float64
          promo2_since_year   float64
          promo_interval      object
          dtype: object
```

```
In [14]: 1 # Change the columns with date, to a datetime type instead of object
          2
          3 df1['date'] = pd.to_datetime( df1['date'] )
          4
          5 # Checking if the change really happended
          6
          7 df1.dtypes
```

executed in 252ms, finished 05:18:03 2023-04-12

```
Out[14]: 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 Checking NA

```
In [15]: 1 # Sum how many NAs we have in each column
         2
         3 df1.isna().sum()
```

executed in 694ms, finished 05:18:03 2023-04-12

```
Out[15]: 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
```

## 2.4.1 What can we do with the NAs?

Each situation will bring a certain need, but most of all the times, we can simply:

1. discard the lines, if there aren't a lot of them, or if the columns isn't important.
2. Use a ML to fulfill the empty values based on a learning behavior inside the dataset
3. Use the mean or median as a pattern value
4. Or use a simple number that doesn't interfere in the analyze

## 2.5 Fill out NA

In [16]:

```
1 # Take a Sample of the dataset
2
3 df1.sample(20)
```

executed in 123ms, finished 05:18:03 2023-04-12

Out[16]:

	store	day_of_week	date	sales	customers	open	promo	state_holiday	school_
<b>255490</b>	487	4	2014-12-11	6413	641	1	0	0	
<b>274804</b>	106	4	2014-11-20	7465	755	1	0	0	
<b>477844</b>	295	2	2014-04-29	6785	797	1	1	0	
<b>798948</b>	279	1	2013-07-15	14059	890	1	1	0	
<b>64610</b>	1056	4	2015-06-04	0	0	0	1	a	
<b>931541</b>	187	1	2013-03-18	7620	846	1	1	0	
<b>575136</b>	582	6	2014-02-01	5548	688	1	0	0	
<b>527119</b>	510	7	2014-03-16	0	0	0	0	0	
<b>335501</b>	12	2	2014-09-16	8632	966	1	1	0	
<b>835419</b>	1070	4	2013-06-13	5657	690	1	0	0	
<b>122870</b>	221	7	2015-04-12	0	0	0	0	0	
<b>837893</b>	199	1	2013-06-10	6085	555	1	0	0	
<b>181668</b>	1039	4	2015-02-19	8406	989	1	1	0	
<b>801967</b>	1068	6	2013-07-13	4240	319	1	0	0	
<b>815953</b>	559	7	2013-06-30	0	0	0	0	0	
<b>773115</b>	91	3	2013-08-07	4346	512	1	0	0	
<b>515282</b>	938	4	2014-03-27	7302	833	1	0	0	
<b>820714</b>	860	3	2013-06-26	3570	503	1	0	0	
<b>685665</b>	726	5	2013-10-25	9619	986	1	1	0	
<b>600478</b>	279	4	2014-01-09	9295	709	1	1	0	

```
In [17]: 1 # Competition distance, let's consider a huge distance that couldn't cre
2
3 df1['competition_distance'] = df1['competition_distance'].apply( lambda
4
5 # Competition open_since_month and since_year, assume the sale date if com
6
7 df1['competition_open_since_month'] = df1.apply(lambda x: x['date'].mont
8                                     else x['competition_open
9 df1['competition_open_since_year'] = df1.apply(lambda x: x['date'].year
10                                     else x['competition_open_
11
12 # Promo2_since_week and promo2_since_year use the same concept as above
13
14 df1['promo2_since_week'] = df1.apply(lambda x: x['date'].week if math.is
15 df1['promo2_since_year'] = df1.apply(lambda x: x['date'].year if math.is
16
17 # Promo_interval, create a Dictionary to relate months with their respec
18
19 month_map = {1: 'Jan', 2: 'Fev', 3: 'Mar', 4: 'Apr', 5: 'May', 6: '
20
21 # Fulfill the NAs with 0, no promo
22
23 df1['promo_interval'].fillna( 0, inplace=True )
24
25 # Use the map to create the month_map column based on which month the sa
26
27 df1['month_map'] = df1['date'].dt.month.map( month_map )
28
29 # Identify if the sale was done within the promo interval
30
31 df1['is_promo'] = df1[['promo_interval', 'month_map']].apply( lambda x:
32                                     else 1 if x
33                                     else 0, axi
34
```

executed in 1m 30.8s, finished 05:19:34 2023-04-12

```
In [18]: 1 # Looking into if we could treat the NAs
         2
         3 df1.isna().sum()
```

executed in 792ms, finished 05:19:35 2023-04-12

```
Out[18]: 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 Change the Data types

```
In [19]: 1 # After modifying the dataset, maybe the columns type has chosen
          2
          3 df1.dtypes
```

executed in 12ms, finished 05:19:35 2023-04-12

```
Out[19]: 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
          month_map                          object
          is_promo                           int64
          dtype: object
```

```
In [20]: 1 # Now Let's reorganize the types as they must be
          2
          3 # Competition_open
          4
          5 df1['competition_open_since_month'] = df1['competition_open_since_month']
          6 df1['competition_open_since_year'] = df1['competition_open_since_year'].
          7
          8 # Promo 2
          9
          10 df1['promo2_since_week'] = df1['promo2_since_week'].astype( 'int64' )
          11 df1['promo2_since_year'] = df1['promo2_since_year'].astype( 'int64' )
```

executed in 58ms, finished 05:19:35 2023-04-12

## 2.7 Descriptive Statistics

```
In [21]: 1 # We are going to divide the columns into two datasets with different ki
          2
          3 num_attributes = df1.select_dtypes( include=['float64', 'int64'] )
          4 cat_attributes = df1.select_dtypes( exclude=['float64', 'int64', 'dateti
```

executed in 92ms, finished 05:19:35 2023-04-12

The idea is to create a dataset to analyze the data we have, and seek for insights

## 2.7.1 Numerical Attributes

```
In [22]: 1 # Central tendency - mean and median
2
3 ct1 = pd.DataFrame( num_attributes.apply( np.mean ) ).T
4 ct2 = pd.DataFrame( num_attributes.apply( np.median ) ).T
5
6 # Dispersion - std, min, max, range, skew, kurtosis
7
8 d1 = pd.DataFrame( num_attributes.apply( np.std ) ).T
9 d2 = pd.DataFrame( num_attributes.apply( min ) ).T
10 d3 = pd.DataFrame( num_attributes.apply( max ) ).T
11 d4 = pd.DataFrame( num_attributes.apply( lambda x: x.max() - x.min() ) ).T
12 d5 = pd.DataFrame( num_attributes.apply( lambda x: x.skew() ) ).T
13 d6 = pd.DataFrame( num_attributes.apply( lambda x: x.kurtosis() ) ).T
14
15 # Concatenate
16
17 m = pd.concat( [d2, d3, d4, ct1, ct2, d1, d5, d6] ).T.reset_index()
18 m.columns = ['attributes', 'min', 'max', 'range', 'mean', 'median', 'std', 'ske']
19 m
20
executed in 3.83s, finished 05:19:39 2023-04-12
```

Out[22]:

	attributes	min	max	range	mean	median	std	ske
0	store	1.0	1115.0	1114.0	558.429727	558.0	321.908493	-0.00095
1	day_of_week	1.0	7.0	6.0	3.998341	4.0	1.997390	0.00159
2	sales	0.0	41551.0	41551.0	5773.818972	5744.0	3849.924283	0.64146
3	customers	0.0	7388.0	7388.0	633.145946	609.0	464.411506	1.59865
4	open	0.0	1.0	1.0	0.830107	1.0	0.375539	-1.75804
5	promo	0.0	1.0	1.0	0.381515	0.0	0.485758	0.48783
6	school_holiday	0.0	1.0	1.0	0.178647	0.0	0.383056	1.67784
7	competition_distance	20.0	200000.0	199980.0	5935.442677	2330.0	12547.646829	10.24234
8	promo2	0.0	1.0	1.0	0.500564	1.0	0.500000	-0.00225
9	promo2_since_week	1.0	52.0	51.0	23.619033	22.0	14.310057	0.17872
10	promo2_since_year	2009.0	2015.0	6.0	2012.793297	2013.0	1.662657	-0.78443
11	is_promo	0.0	1.0	1.0	0.155231	0.0	0.362124	1.90415

**Seaborn Distplot** represents the overall distribution of continuous data variables, is a convenient way to visualize the distribution of a variable in a Pandas DataFrame using Seaborn library.

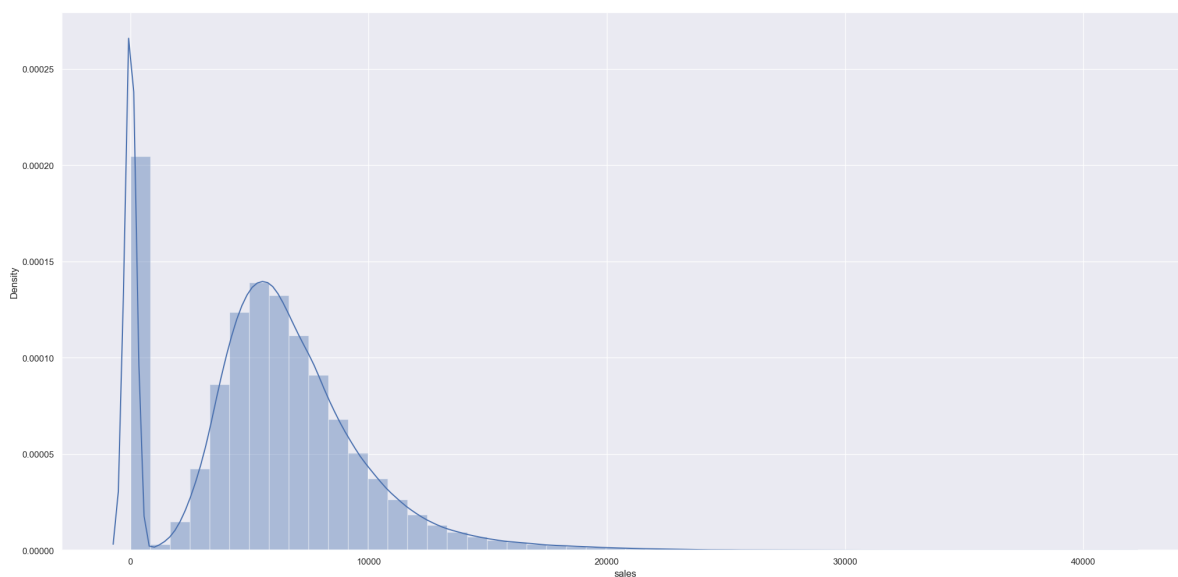
Overall, a Distplot chart provides a useful visual representation of the distribution of a dataset, allowing you to quickly identify the range of the data, the most common values, and any outliers or unusual patterns.



In [23]:

executed in 11.3s, finished 05:19:50 2023-04-12

Out[23]: &lt;AxesSubplot: xlabel='sales', ylabel='Density'&gt;



## 2.7.2 Categorical Attributes

In [24]:

```
1 # Check the range of the variables
2
3 cat_attributes.apply( lambda x: x.unique().shape[0] )
```

executed in 267ms, finished 05:19:51 2023-04-12

```
Out[24]: state_holiday      4
store_type      4
assortment      3
competition_open_since_month  12
competition_open_since_year   23
promo_interval   4
month_map        12
dtype: int64
```

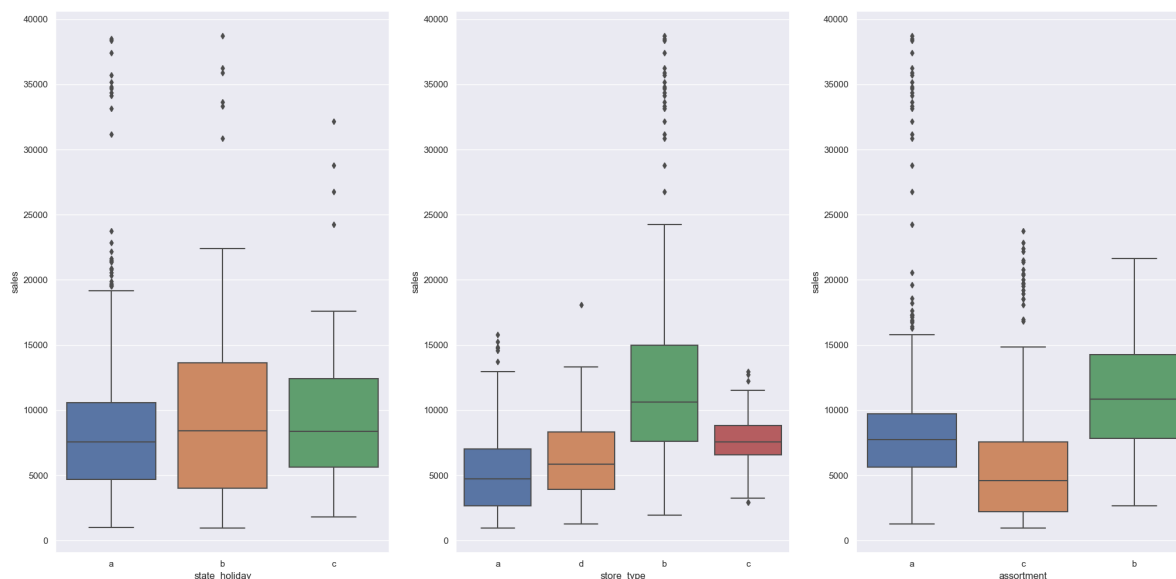
```

In [25]: 1 # Use seaborn to analyze how the categorical variables act
          2
          3 aux = df1[(df1['state_holiday'] != '0') & (df1['sales'] > 0)]
          4
          5 plt.subplot(1, 3, 1)
          6 sns.boxplot( x='state_holiday', y='sales', data=aux )
          7
          8 plt.subplot(1, 3, 2)
          9 sns.boxplot( x='store_type', y='sales', data=aux )
         10
         11 plt.subplot(1, 3, 3)
         12 sns.boxplot( x='assortment', y='sales', data=aux )
         13

```

executed in 1.02s, finished 05:19:52 2023-04-12

Out[25]: <AxesSubplot: xlabel='assortment', ylabel='sales'>



### 3 Feature Engineering (Second Step)

In [26]:

```

1 # Show the brainstorm done regarding some hypothesis
2
3 Image( r'C:Brainstorm.jpg' )

```

executed in 28ms, finished 05:19:52 2023-04-12

Out[26]:



## 3.1 Creating Hypothesis

### 3.1.1 Store Hypothesis

1. Stores with the most number of employee should sell more;
2. Stores with the most capacity of warehouse should sell more;
3. Big Stores should sell more;
4. Stores with the most assortment should sell more;
5. Stores with the closest competitors should sell less;
6. Stores with the oldest competitors should sell less.

### 3.1.2 Product Hypothesis

1. Stores which invest the most in marketing should sell more;
2. Stores with the most showcase of products should sell more;

3. Stores with the cheapest products should sell more;
5. Stores with the most aggressive discounts should sell more;
6. Stores with active discount for a longer time should sell more;
7. Stores with the most days on sale should sell more;

### **3.1.3 Time Hypothesis**

1. Stores which open at holidays should sell more;
2. Stores should sell more throughout the years;
3. Stores should sell more in the second semester;
4. Stores should sell more after the 10th of the each month;
5. Stores should sell less in the weekends;
6. Stores should sell less during scholars holidays

## **3.2 Final Hypothesis List**

1. Stores with the most assortment should sell more;
2. Stores with the closest competitors should sell less;
3. Stores with the oldest competitors should sell more;
4. Stores with active discount for a longer time should sell more;
5. Stores with the most days on sale should sell more;
6. Stores with frequent discounts should sell more;
7. Stores should sell more during Christmas;
8. Stores should sell more throughout the years;
9. Stores should sell more in the second semester;
10. Stores should sell more after the 10th of the each month;
11. Stores should sell less in the weekends;
12. Stores should sell less during scholars holidays.

## **3.3 Adding Features**

The Idea is to create new columns (features) before starting the data analyze.

```
In [27]: 1 # Checkpoint 2
2
3 df2 = df1.copy()
4
5 # Year, month, year, Year of week, Year week ( Creating Columns related
6
7 df2['year'] = df2['date'].dt.year
8 df2['month'] = df2['date'].dt.month
9 df2['day'] = df2['date'].dt.day
10 df2['week_of_year'] = df2['date'].dt.weekofyear
11 df2['year_week'] = df2['date'].dt.strftime( '%Y-%W' )
12
13 # Competition since
14
15 df2['competition_since'] = df2.apply( lambda x: datetime.datetime( year=
16                                     month=
17 df2['competition_time_month'] = ( ( df2['date'] - df2['competition_since
18
19 # Promo Since
20
21 df2['promo_since'] = df2['promo2_since_year'].astype( str ) + '-' + df2[
22 df2['promo_since'] = df2['promo_since'].apply( lambda x: datetime.dateti
23 df2['promo_time_week'] = ( ( df2['date'] - df2['promo_since'] ) / 7 ).ap
24
25 # Assortment
26
27 df2['assortment'] = df2['assortment'].apply( lambda x: 'basic' if x == '
28
29 # State Holiday
30
31 df2['state_holiday'] = df2['state_holiday'].apply( lambda x: 'public_hol
32                                     if x == 'c' else 'regul
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

executed in 59.9s, finished 05:20:52 2023-04-12

```
In [28]: 1 df2.head(10)
```

executed in 34ms, finished 05:20:52 2023-04-12

Out[28]:

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

## 4 Variable Filtering (Third Step)

In [29]:

```
1 # Checkpoint 3
2
3 df3 = df2.copy()
```

executed in 6.35s, finished 05:20:58 2023-04-12

## 4.1 Line Filtering

In [30]:

```
1 # Take out all the lines the stores were closed
2
3 df3 = df3[( df3['open'] != 0 ) & ( df3['sales'] > 0 )]
```

executed in 249ms, finished 05:20:58 2023-04-12

## 4.2 Column Selection

In [31]:

```
1 # Take out the columns we won't use or we can't use their information
2
3 cols_drop = ['customers', 'open', 'promo_interval', 'month_map']
4 df3 = df3.drop( cols_drop, axis=1 )
```

executed in 156ms, finished 05:20:58 2023-04-12

# 5 Data Analysis (Fourth Step)

In [32]:

```
1 # Checkpoint 4
2
3 df4 = df3.copy()
```

executed in 60ms, finished 05:20:58 2023-04-12

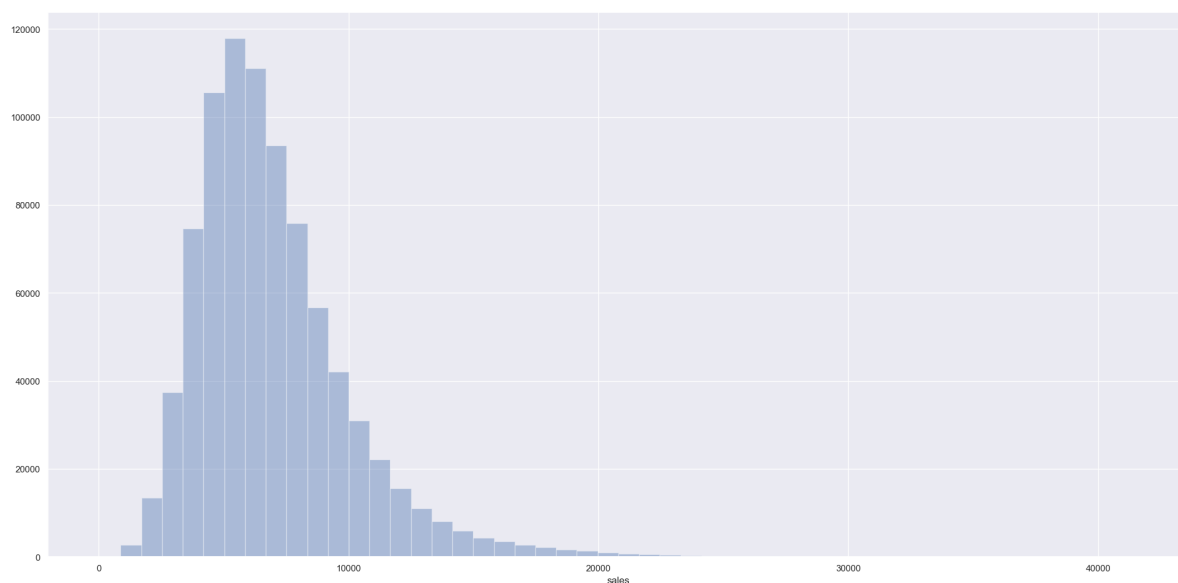
## 5.1 Univariable Analyse

### 5.1.1 Response Variable

```
In [33]: 1 # Check the behavior of our response variable
         2
         3 sns.distplot( df4['sales'], kde=False )
```

executed in 708ms, finished 05:20:59 2023-04-12

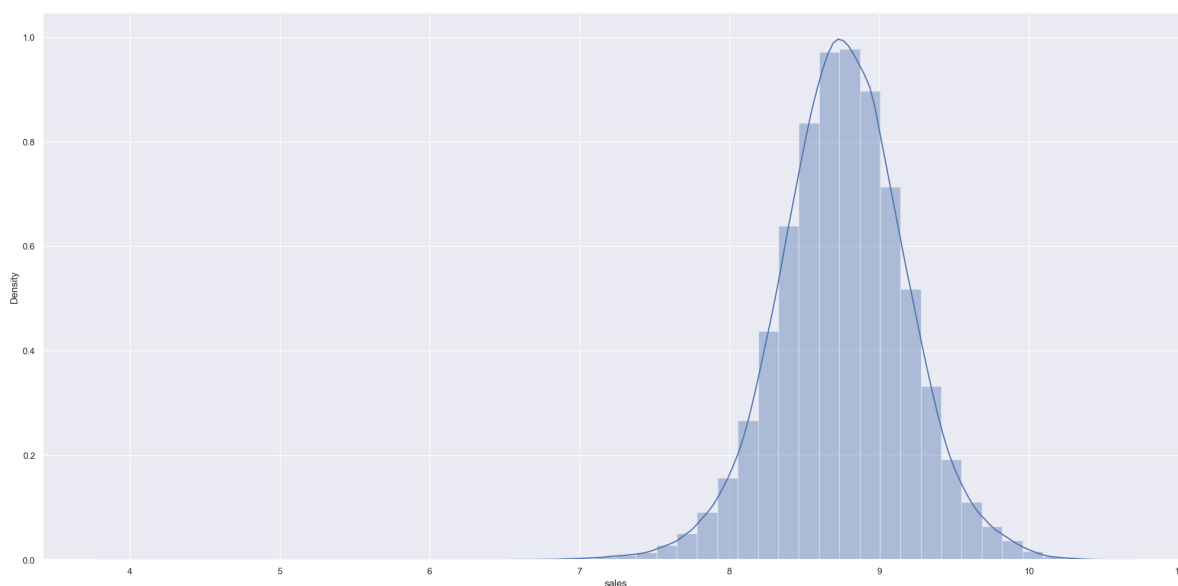
Out[33]: <AxesSubplot: xlabel='sales'>



```
In [34]: 1 # One of the requirements is to make the variable more normal as possible
         2
         3 sns.distplot( np.log1p( df4['sales'] ) )
```

executed in 9.29s, finished 05:21:08 2023-04-12

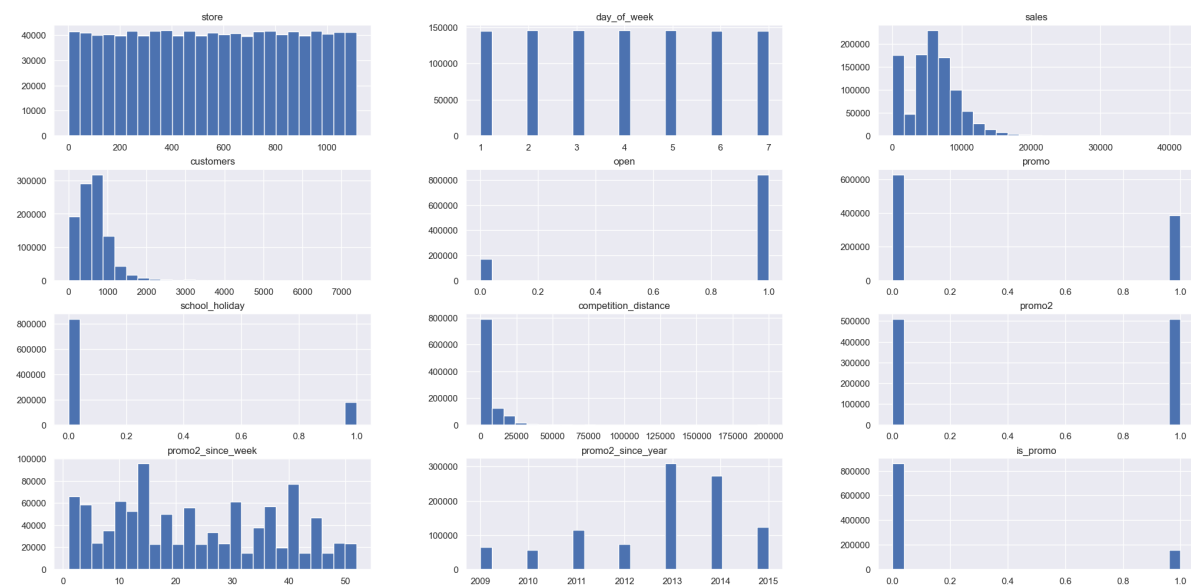
Out[34]: <AxesSubplot: xlabel='sales', ylabel='Density'>



## 5.1.2 Numerical Variables

```
In [35]: 1 # Analyse each variable to see their behaviors individually
          2
          3 num_attributes.hist( bins=25 );
          4
          5 # This can be useful for visualizing the distribution of the data and id
```

executed in 4.80s, finished 05:21:13 2023-04-12



### 5.1.3 Categorical Variable

```
In [36]: 1 df4['state_holiday'].drop_duplicates()
          2 df4['store_type'].drop_duplicates()
```

executed in 171ms, finished 05:21:13 2023-04-12

```
Out[36]: 0      basic
          3      extended
          258     extra
          Name: assortment, dtype: object
```

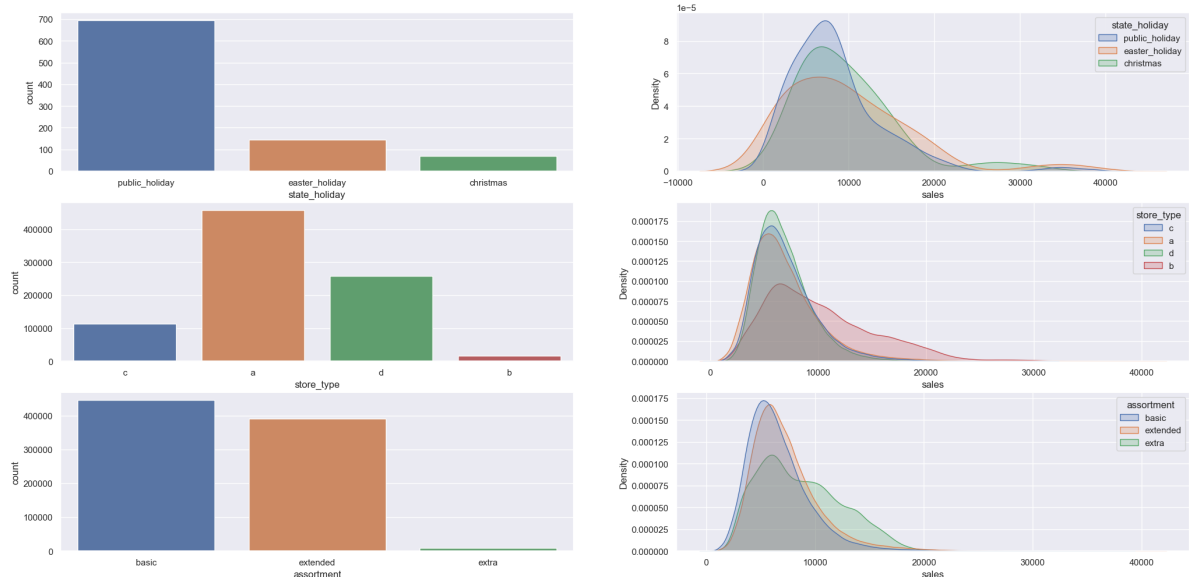


```

In [37]: 1 # Check the performance among the variables like holiday, store type and
2
3 # State_holiday
4
5 plt.subplot(3, 2, 1)
6 a = df4[ df4['state_holiday'] != 'regular_day' ]
7 sns.countplot( x=a['state_holiday'] )
8
9 plt.subplot(3, 2, 2)
10 sns.kdeplot( data=a, x='sales', hue='state_holiday', fill=True, common_n
11
12 # Store_type
13
14 plt.subplot(3, 2, 3)
15 sns.countplot( x=df4['store_type'] );
16
17 plt.subplot(3, 2, 4)
18 sns.kdeplot( data=df4, x='sales', hue='store_type', fill=True, common_no
19
20 # Assortment
21
22 plt.subplot(3, 2, 5)
23 sns.countplot( x=df4['assortment'] )
24
25 plt.subplot(3, 2, 6)
26 sns.kdeplot( data=df4, x='sales', hue='assortment', fill=True, common_no
27

```

executed in 20.1s, finished 05:21:34 2023-04-12



## 5.2 Bivariate Analyse

### 5.2.1 H1. Stores with the most assortment should sell more

**False** Stores with bigger assortment sell less

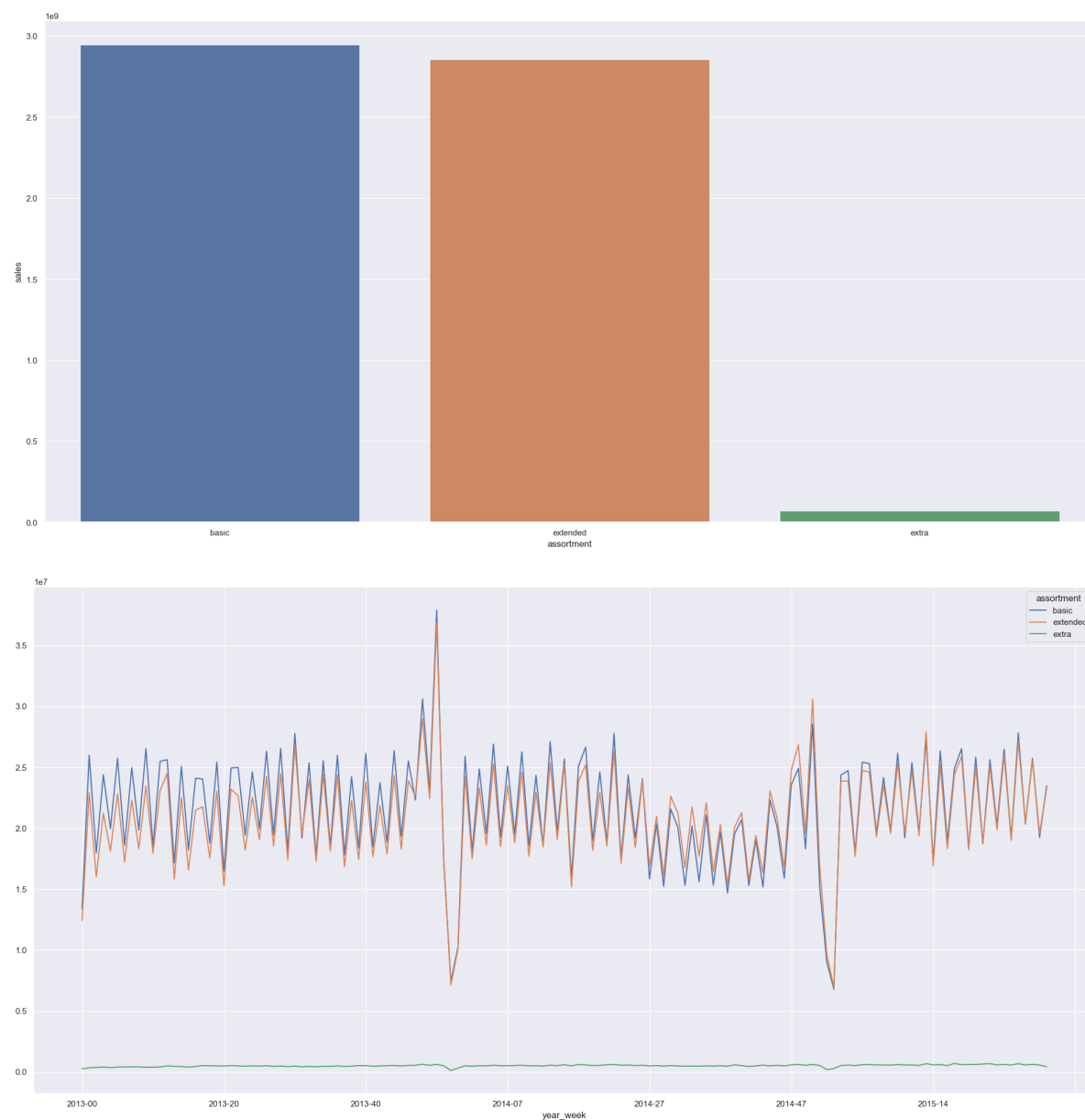
```

In [38]: 1 # Test if the Stores with bigger assortment sell more
          2
          3 # Group the segments, then sum their sales
          4
          5 aux1 = df4[['assortment', 'sales']].groupby( 'assortment' ).sum().reset_
          6 sns.barplot( x='assortment', y='sales', data=aux1 );
          7
          8 # Now let's see the behavior throughout the years
          9
         10 aux2 = df4[['year_week', 'assortment', 'sales']].groupby( ['year_week',
         11 aux2.pivot( index='year_week', columns='assortment', values='sales' ).pl
         12
         13 # We need to create one more chart to check the assortment (extra), beca
         14
         15 aux3 = aux2[aux2['assortment'] == 'extra']
         16 aux3.pivot( index='year_week', columns='assortment', values='sales' ).pl

```

executed in 1.74s, finished 05:21:35 2023-04-12

Out[38]: <AxesSubplot: xlabel='year\_week'>



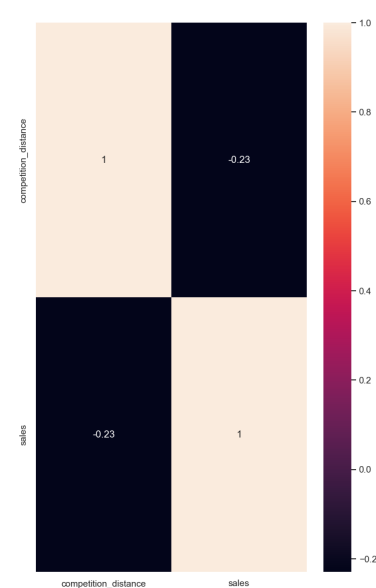
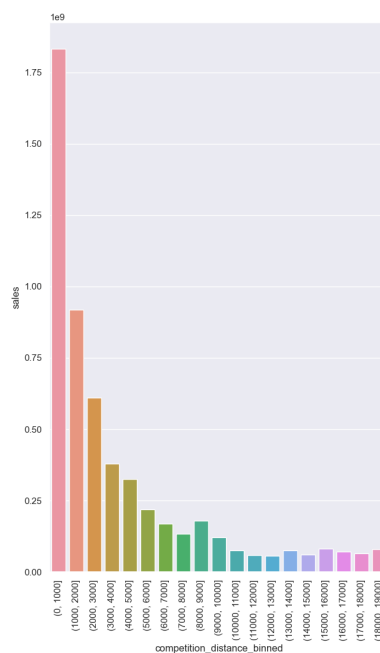
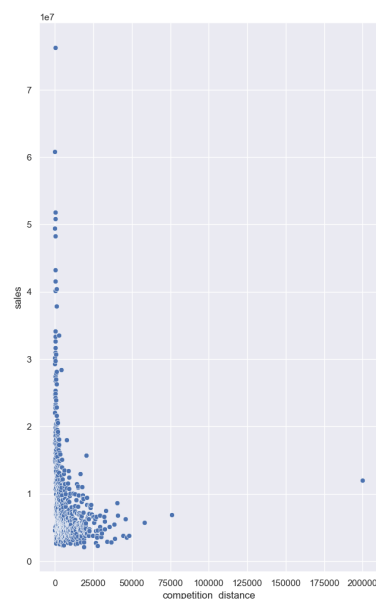
In [39]:

```

1 # Check the relation between sales and competition distance
2
3 aux1 = df4[['competition_distance', 'sales']].groupby( 'competition_dist
4
5 # Scatterplot help us to show the distribution of sales in the various d
6
7 plt.subplot(1, 3, 1)
8 sns.scatterplot( x='competition_distance', y='sales', data=aux1 );
9
10 # Let's create a more visible chart, Bins make the data divided in group
11
12 plt.subplot(1, 3, 2)
13 bins = list( np.arange(0, 20000, 1000) )
14
15 aux1['competition_distance_binned'] = pd.cut( aux1['competition_distance
16 aux2 = aux1[['competition_distance_binned', 'sales']].groupby( 'competit
17 sns.barplot( x='competition_distance_binned', y='sales', data=aux2 );
18
19 # Rotate the x label to be able to see the subtitles
20
21 plt.xticks( rotation=90 );
22
23 # Correlation Pearson
24
25 plt.subplot(1, 3, 3)
26 x = sns.heatmap( aux1.corr( method='pearson' ), annot=True );

```

executed in 1.98s, finished 05:21:37 2023-04-12

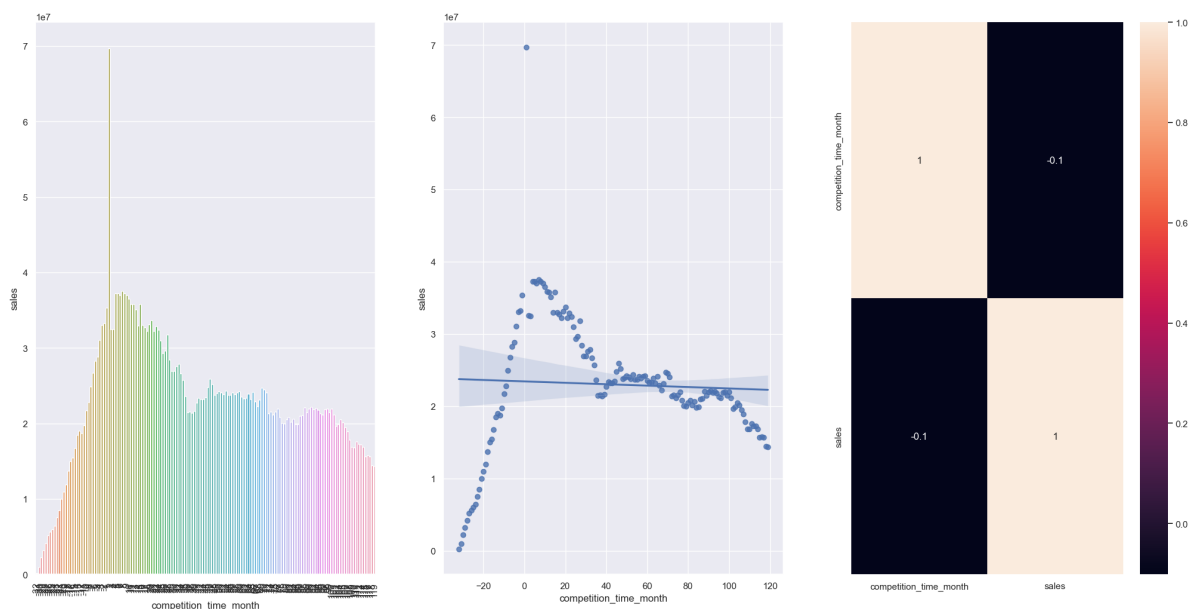


### 5.2.3 H3. Stores with the oldest competitors should sell more

**False** Stores with oldest competitors sell less

```
In [40]: 1 # Group the competition by time to its influence in the sales
2
3 plt.subplot(1, 3, 1)
4 aux1 = df4[['competition_time_month', 'sales']].groupby( 'competition_ti
5
6 # Make a filter competition time that returns a max 120 months and differ
7
8 aux2 = aux1[( aux1['competition_time_month'] < 120 ) & ( aux1['competiti
9 sns.barplot( x='competition_time_month', y='sales', data=aux2 );
10 plt.xticks( rotation=90 );
11
12 plt.subplot(1, 3, 2)
13 sns.regplot( x='competition_time_month', y='sales', data=aux2 );
14
15 plt.subplot(1, 3, 3)
16 x = sns.heatmap( aux1.corr( method='pearson' ), annot=True );
```

executed in 6.64s, finished 05:21:44 2023-04-12



### 5.2.4 H4. Stores with active discount for a longer time should sell more

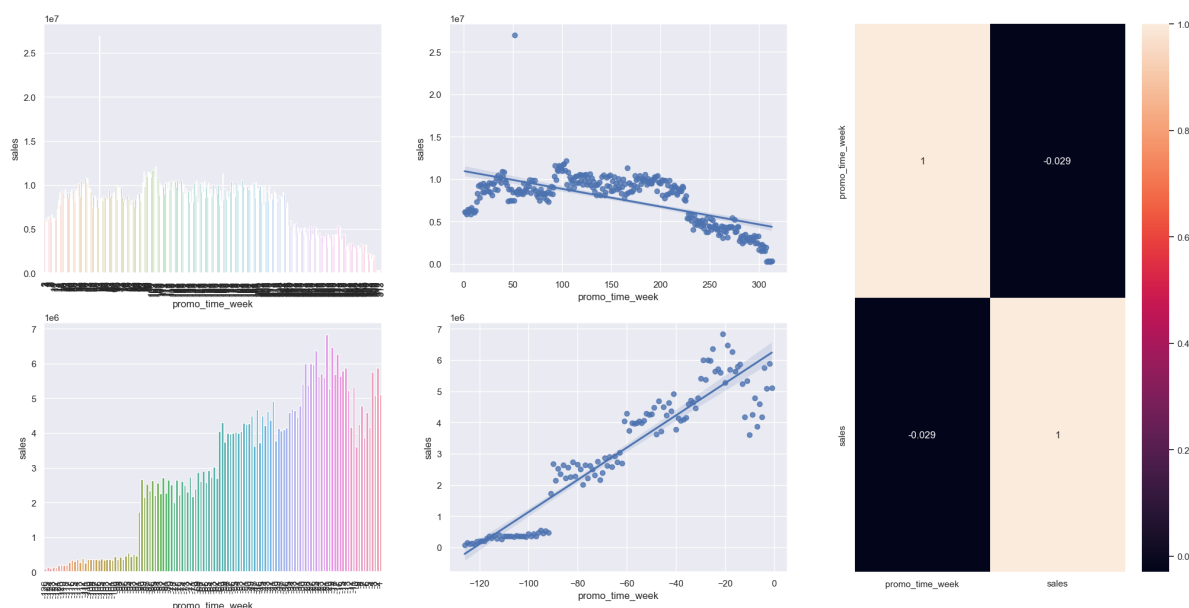
**False** Stores with active discount for a longer time sell less after a certain time

```

In [41]: 1 # check how the promo time affects the sales
2
3 aux1 = df4[['promo_time_week', 'sales']].groupby( 'promo_time_week' ).su
4
5 # Create a Grid
6
7 grid = GridSpec(2, 3)
8
9 plt.subplot( grid[0,0] )
10
11 aux2 = aux1[aux1['promo_time_week'] > 0] # Extended Promo
12 sns.barplot( x='promo_time_week', y='sales', data=aux2 );
13 plt.xticks( rotation=90 );
14
15 plt.subplot( grid[0, 1] )
16
17 sns.regplot( x='promo_time_week', y='sales', data=aux2 );
18
19 plt.subplot( grid[1, 0] )
20 aux3 = aux1[aux1['promo_time_week'] < 0] # Regular Promo
21 sns.barplot( x='promo_time_week', y='sales', data=aux3 );
22 plt.xticks( rotation=90 );
23
24 plt.subplot( grid[1,1] )
25 sns.regplot( x='promo_time_week', y='sales', data=aux3 );
26
27 plt.subplot( grid[:,2] )
28 sns.heatmap( aux1.corr( method='pearson'), annot=True );
29

```

executed in 11.9s, finished 05:21:56 2023-04-12



**5.2.5 H5. Stores with the most days on sale should sell more**

**5.2.6 H6. Stores with frequent discounts should sell more**

```
In [42]: 1 # Check how much the stores with and without discount have sold
2
3 df4[['promo', 'promo2', 'sales']].groupby( ['promo', 'promo2'] ).sum().s
```

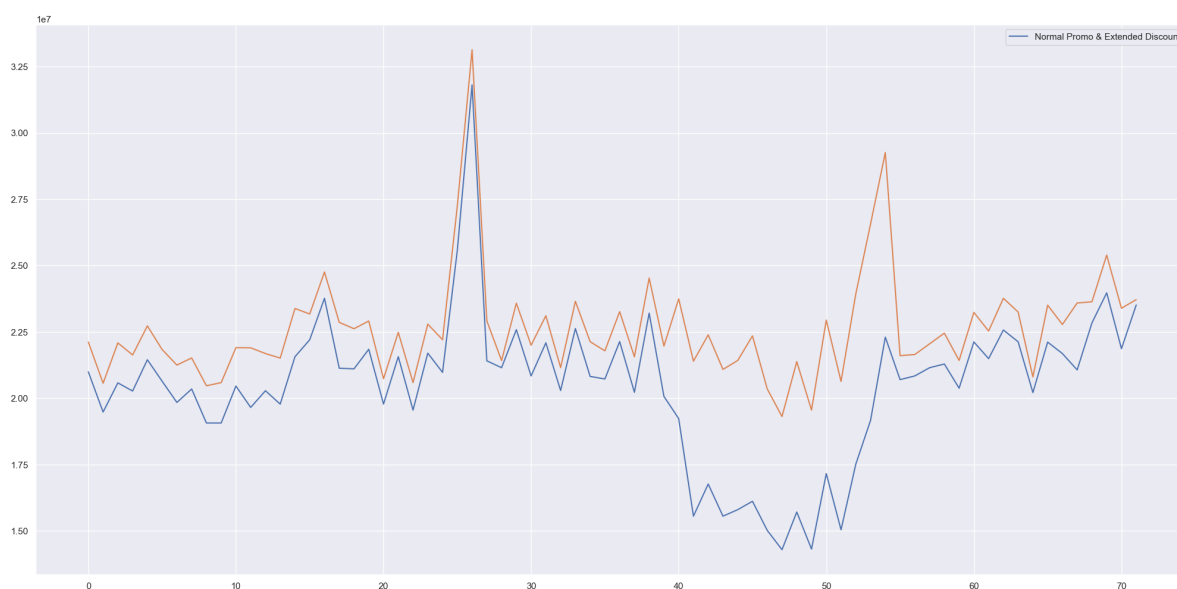
executed in 105ms, finished 05:21:56 2023-04-12

Out[42]:

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

```
In [43]: 1 # Nos Let's check the performance thoroughout time
2
3 aux1 = df4[( df4['promo'] == 1 ) & ( df4['promo2'] == 1 )][['year_week',
4 ax = aux1.plot()
5
6 aux2 = df4[( df4['promo'] == 1 ) & ( df4['promo2'] == 0 )][['year_week',
7 aux2.plot( ax=ax )
8
9 ax.legend( labels=['Normal Promo & Extended Discount'] );
```

executed in 792ms, finished 05:21:57 2023-04-12



## 5.2.7 H7. Stores should sell more during Christmas

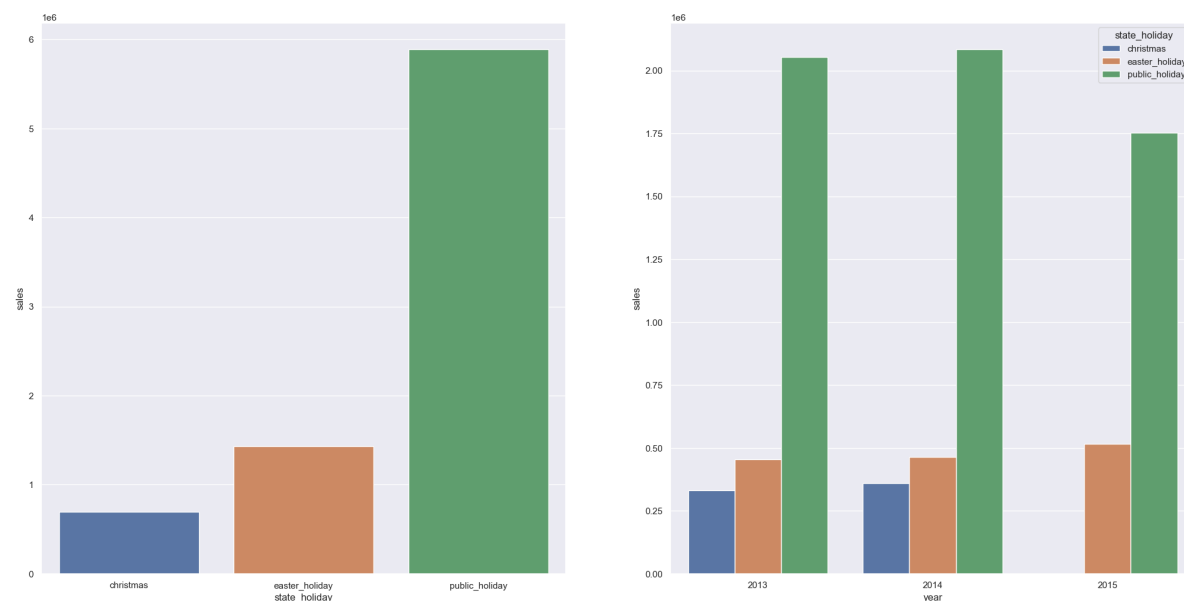
False Store sell less on Christmas

```

In [44]: 1 # Filter the Holiday
          2
          3 aux = df4[df4['state_holiday'] != 'regular_day']
          4
          5 plt.subplot(1, 2, 1)
          6 aux1 = aux[['state_holiday', 'sales']].groupby( 'state_holiday' ).sum().
          7 sns.barplot( x='state_holiday', y='sales', data=aux1 );
          8
          9 plt.subplot(1, 2, 2)
         10 aux2 = aux[['year', 'state_holiday', 'sales']].groupby(['year', 'state_h
         11 sns.barplot( x='year', y='sales', hue='state_holiday', data=aux2 );
         12

```

executed in 774ms, finished 05:21:57 2023-04-12



## 5.2.8 H8. Stores should sell more throughout the years

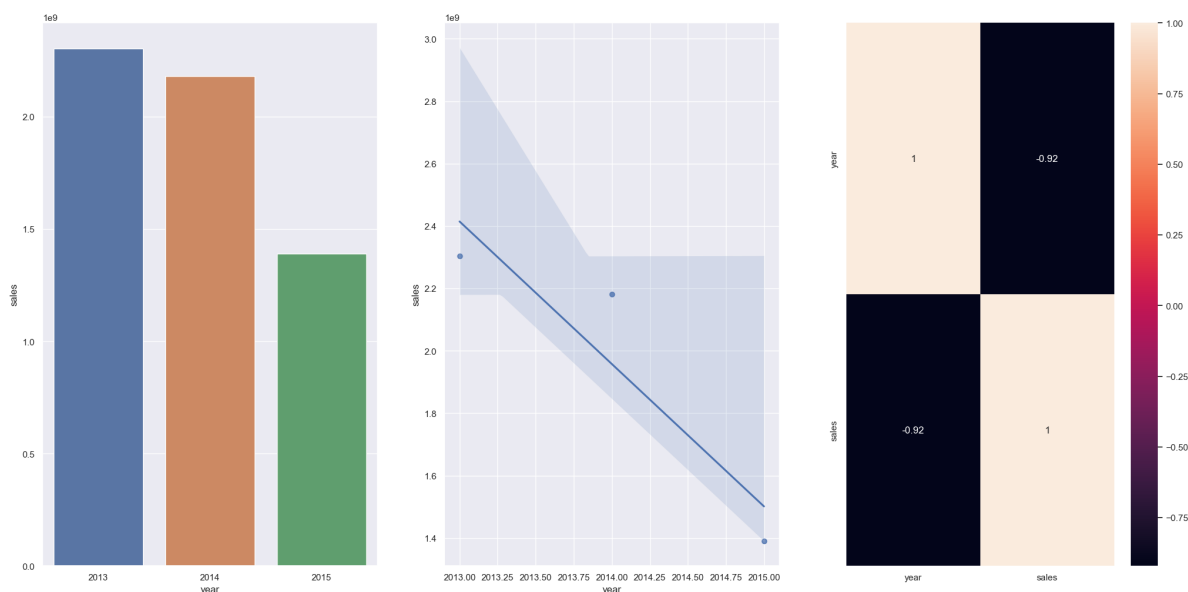
**False** Stores sell less throughout the years

```

In [45]: 1 # Group the sales in the years, (Note that the last year wasn't complete
          2
          3 aux1 = df4[['year', 'sales']].groupby( 'year' ).sum().reset_index()
          4
          5 plt.subplot(1, 3, 1)
          6 sns.barplot( x='year', y='sales', data=aux1 );
          7
          8 plt.subplot(1, 3, 2)
          9 sns.regplot( x='year', y='sales', data=aux1 );
         10
         11 plt.subplot(1, 3, 3)
         12 sns.heatmap( aux1.corr( method='pearson' ), annot=True );
         13

```

executed in 1.34s, finished 05:21:59 2023-04-12



### 5.2.9 H9. Stores should sell more in the second semester

**False** Stores sell less in the second semester

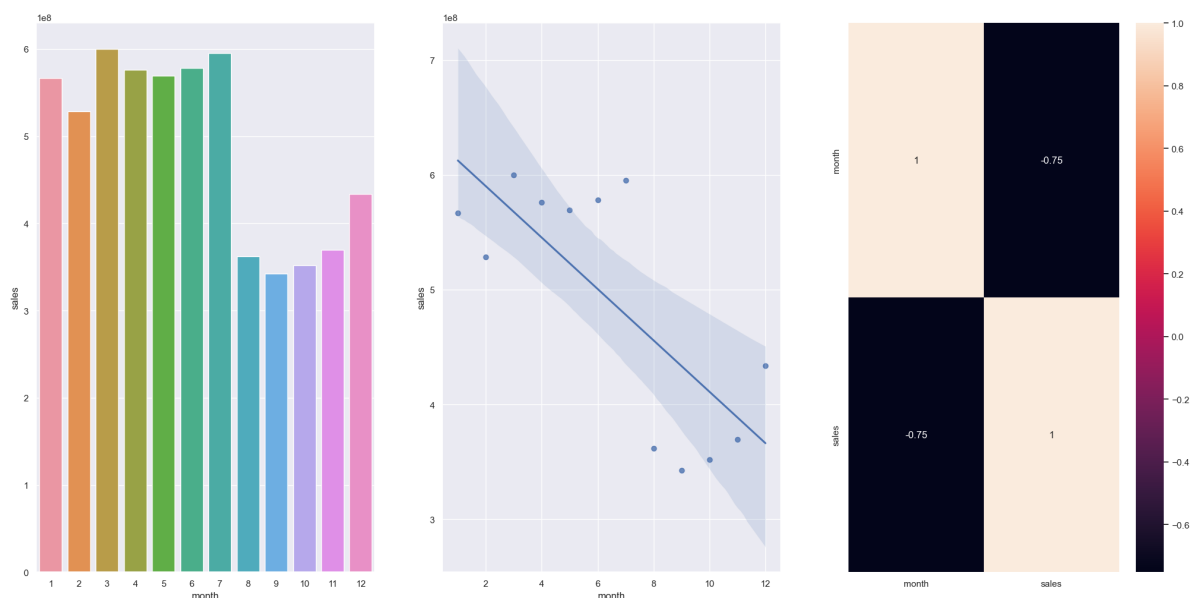


```

In [46]: 1 # Let's Group the sales in months, then we can see the behavior trougou
          2
          3 aux1 = df4[['month', 'sales']].groupby( 'month' ).sum().reset_index()
          4
          5 plt.subplot(1, 3, 1)
          6 sns.barplot( x='month', y='sales', data=aux1 );
          7
          8 plt.subplot(1, 3, 2)
          9 sns.regplot( x='month', y='sales', data=aux1 );
         10
         11 plt.subplot(1, 3, 3)
         12 sns.heatmap( aux1.corr( method='pearson' ), annot=True );
         13

```

executed in 1.47s, finished 05:22:00 2023-04-12



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

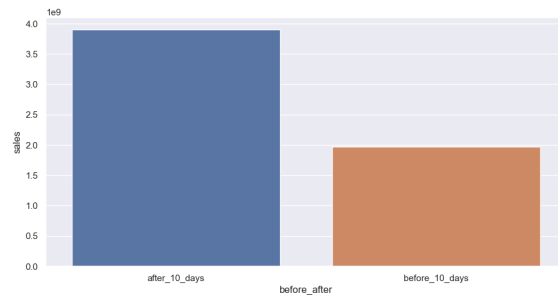
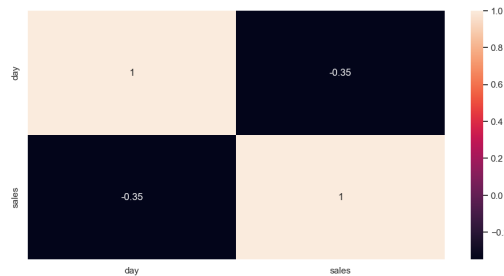
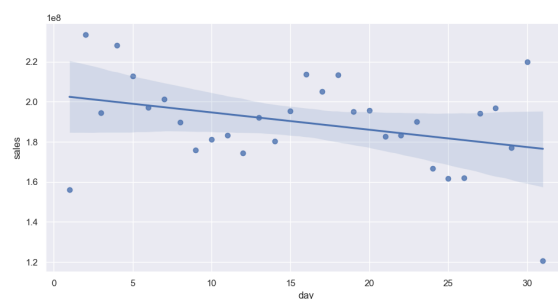
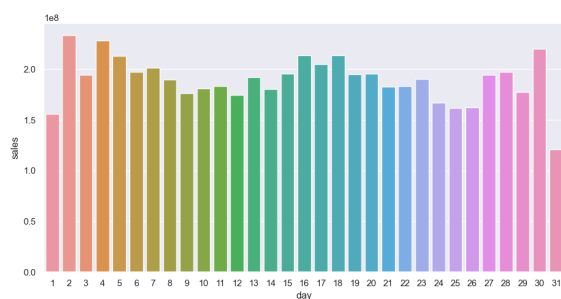
**True** Stores sell more after the 10th day of each month

```

In [47]: 1 # Group the sales in each day of the month
2
3 aux1 = df4[['day', 'sales']].groupby( 'day' ).sum().reset_index()
4
5 plt.subplot(2, 2, 1)
6 sns.barplot( x='day', y='sales', data=aux1 );
7
8 plt.subplot(2, 2, 2)
9 sns.regplot( x='day', y='sales', data=aux1 );
10
11 plt.subplot(2, 2, 3)
12 sns.heatmap( aux1.corr( method='pearson' ), annot=True );
13
14 # Filter in two groups, before day 10 and after day 10
15
16 aux1['before_after'] = aux1['day'].apply( lambda x: 'before_10_days' if
17 aux2 = aux1[['before_after', 'sales']].groupby( 'before_after' ).sum().r
18
19 plt.subplot(2, 2, 4)
20 sns.barplot( x='before_after', y='sales', data=aux2 );
21

```

executed in 2.31s, finished 05:22:03 2023-04-12



### 5.2.11 H11. Stores should sell less in the weekends

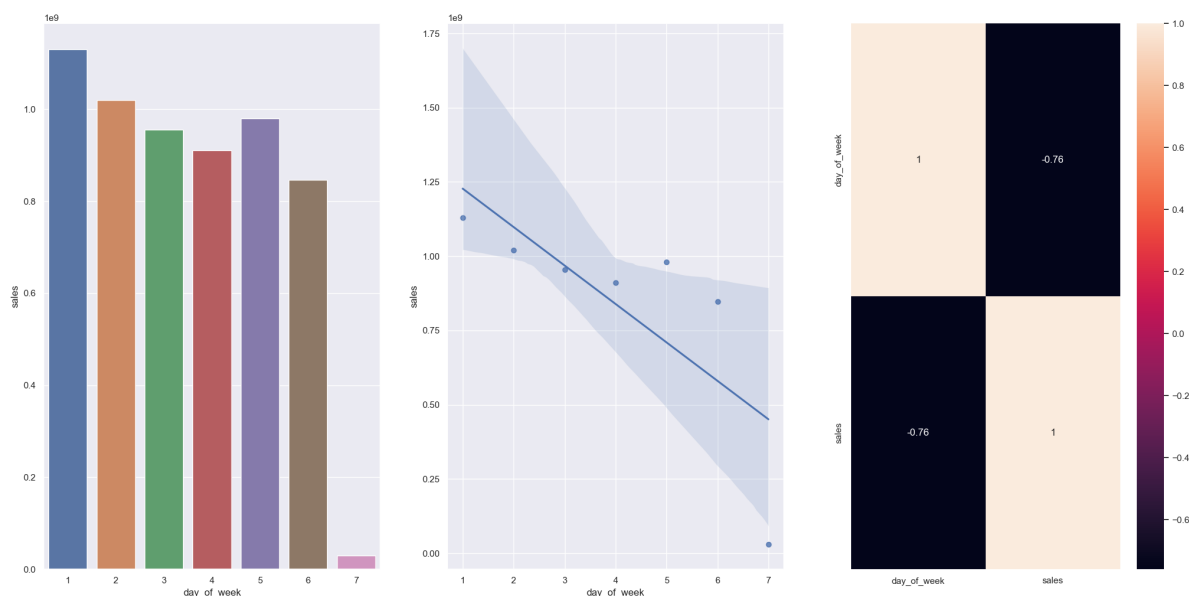
**True** Stores sell less in the weekend

```

In [48]: 1 # Group the sales by day
          2
          3 aux1 = df4[['day_of_week', 'sales']].groupby( 'day_of_week' ).sum().rese
          4
          5 plt.subplot(1, 3, 1)
          6 sns.barplot( x='day_of_week', y='sales', data=aux1 );
          7
          8 plt.subplot(1, 3, 2)
          9 sns.regplot( x='day_of_week', y='sales', data=aux1 );
         10
         11 plt.subplot(1, 3, 3)
         12 sns.heatmap( aux1.corr( method='pearson'), annot=True );
         13

```

executed in 1.38s, finished 05:22:04 2023-04-12



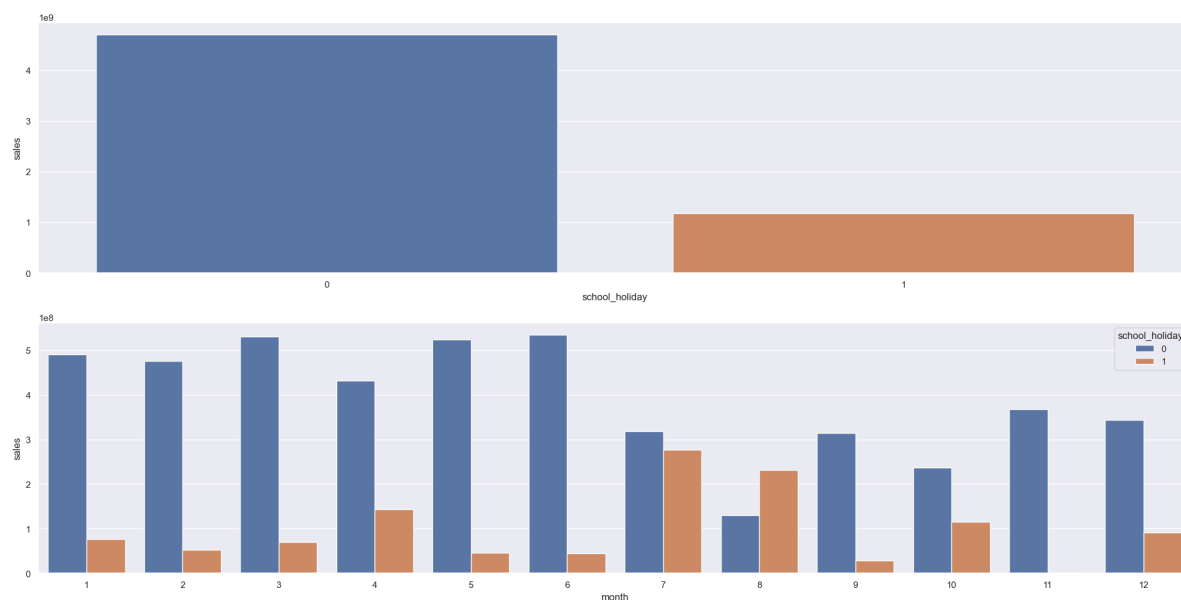
## 5.2.12 H12. Stores should sell less during scholar holidays

**True** Stores sell less during the scholar holidays but in August, and july we have almost the same number

```

In [49]: 1 # Group by school holiday yes or not
          2
          3 aux1 = df4[['school_holiday', 'sales']].groupby( 'school_holiday' ).sum(
          4
          5 plt.subplot(2, 1, 1)
          6 sns.barplot( x='school_holiday', y='sales', data=aux1 );
          7
          8 # filter deeper to see the diference throughout the months
          9
         10 aux2 = df4[['month', 'school_holiday', 'sales']].groupby( ['month', 'sch
         11
         12 plt.subplot(2, 1, 2)
         13 sns.barplot( x='month', y='sales', hue='school_holiday', data=aux2 );
         14
executed in 890ms, finished 05:22:05 2023-04-12

```



### 5.2.13 Hypothesis Results

```

In [50]: 1 # Check the charts we made, and see how influent is each variable regard
          2
          3 tab = [['Hipoteses', 'Conclusao', 'Relevancia'],
          4             ['H1', 'Falsa', 'Baixa'],
          5             ['H2', 'Falsa', 'Media'],
          6             ['H3', 'Falsa', 'Media'],
          7             ['H4', 'Falsa', 'Baixa'],
          8             ['H5', '-', '-'],
          9             ['H6', 'Falsa', 'Baixa'],
         10             ['H7', 'Falsa', 'Media'],
         11             ['H8', 'Falsa', 'Alta'],
         12             ['H9', 'Falsa', 'Alta'],
         13             ['H10', 'Verdadeira', 'Alta'],
         14             ['H11', 'Verdadeira', 'Alta'],
         15             ['H12', 'Verdadeira', 'Baixa'],
         16         ]
         17 print( tabulate( tab, headers='firstrow' ) )

```

executed in 28ms, finished 05:22:05 2023-04-12

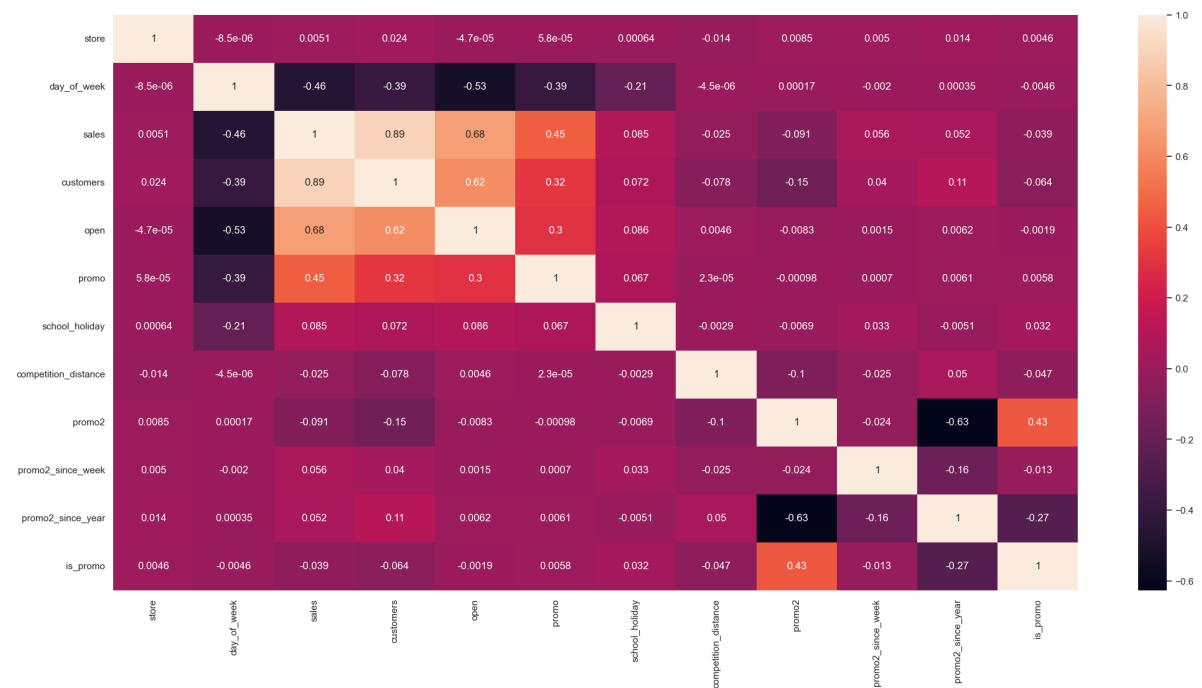
Hipoteses	Conclusao	Relevancia
H1	Falsa	Baixa
H2	Falsa	Media
H3	Falsa	Media
H4	Falsa	Baixa
H5	-	-
H6	Falsa	Baixa
H7	Falsa	Media
H8	Falsa	Alta
H9	Falsa	Alta
H10	Verdadeira	Alta
H11	Verdadeira	Alta
H12	Verdadeira	Baixa

## 5.3 Multiple Analyse

### 5.3.1 Numerical Attributes

```
In [51]: 1 # Check all the correlations among the Numerical Variables
2
3 correlation = num_attributes.corr( method='pearson' )
4 sns.heatmap( correlation, annot=True );
```

executed in 1.61s, finished 05:22:06 2023-04-12



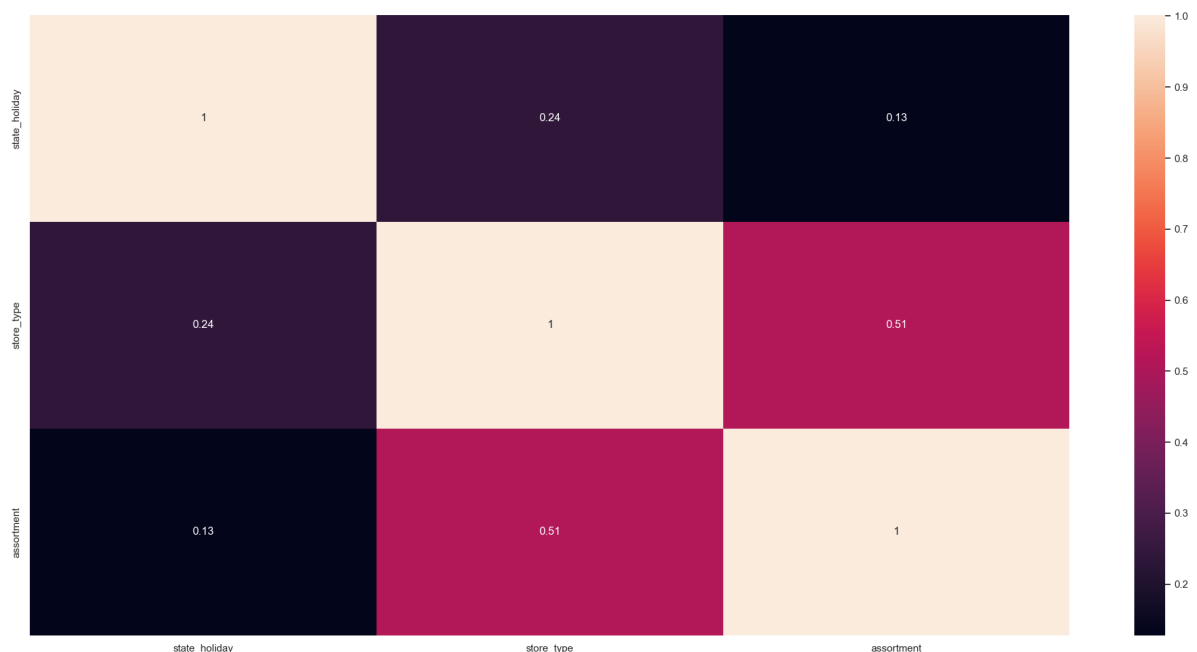
### 5.3.2 Categorical Attributes

```

In [52]: 1 # Now check all the correlations among the Categorical Variables
2
3 # Calculate Cramer V
4
5 a1 = cramer_v( a['state_holiday'], a['state_holiday'] )
6 a2 = cramer_v( a['state_holiday'], a['store_type'] )
7 a3 = cramer_v( a['state_holiday'], a['assortment'] )
8
9 a4 = cramer_v( a['store_type'], a['state_holiday'] )
10 a5 = cramer_v( a['store_type'], a['store_type'] )
11 a6 = cramer_v( a['store_type'], a['assortment'] )
12
13 a7 = cramer_v( a['assortment'], a['state_holiday'] )
14 a8 = cramer_v( a['assortment'], a['store_type'] )
15 a9 = cramer_v( a['assortment'], a['assortment'] )
16
17 # Creating the Matrix
18
19 d = pd.DataFrame( {'state_holiday': [a1, a2, a3],
20                    'store_type': [a4, a5, a6],
21                    'assortment': [a7, a8, a9]} )
22 d = d.set_index( d.columns )
23
24 sns.heatmap( d, annot=True );

```

executed in 820ms, finished 05:22:07 2023-04-12



## 6 Data Preparation (Fifth Step)

```

In [53]:

```

executed in 74ms, finished 05:22:07 2023-04-12

### 6.1 Standarization

We checked in (5.1.2 Numerical Variable) if we had already any normalized variable, Normal variable is when we have a variable without outliers

## 6.2 Rescaling

```
In [54]: 1 # Using methods from sklearn to rescale the variables, bringing them nex
2
3 # RobustScaler is used when we have too many outliers
4
5 rs = RobustScaler()
6
7 # MinMaxScaler is used when the variable has a pattern, with a little ou
8
9 mms = MinMaxScaler()
10
11 # competition distance
12
13 df5['competition_distance'] = rs.fit_transform( df5[['competition_distan
14
15 # This part is going to be used in the exploitation part as API
16 #pickle.dump( rs, open('C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo
17
18 # competition time month
19
20 df5['competition_time_month'] = rs.fit_transform( df5[['competition_time
21
22 # This part is going to be used in the exploitation part as API
23 #pickle.dump( rs, open('C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo
24
25 # promo time week
26
27 df5['promo_time_week'] = mms.fit_transform( df5[['promo_time_week']].val
28
29 # This part is going to be used in the exploitation part as API
30 #pickle.dump( rs, open('C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo
31
32 # year
33
34 df5['year'] = mms.fit_transform( df5[['year']].values )
35
36 # This part is going to be used in the exploitation part as API
37 #pickle.dump( mms, open('C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo
38
39
```

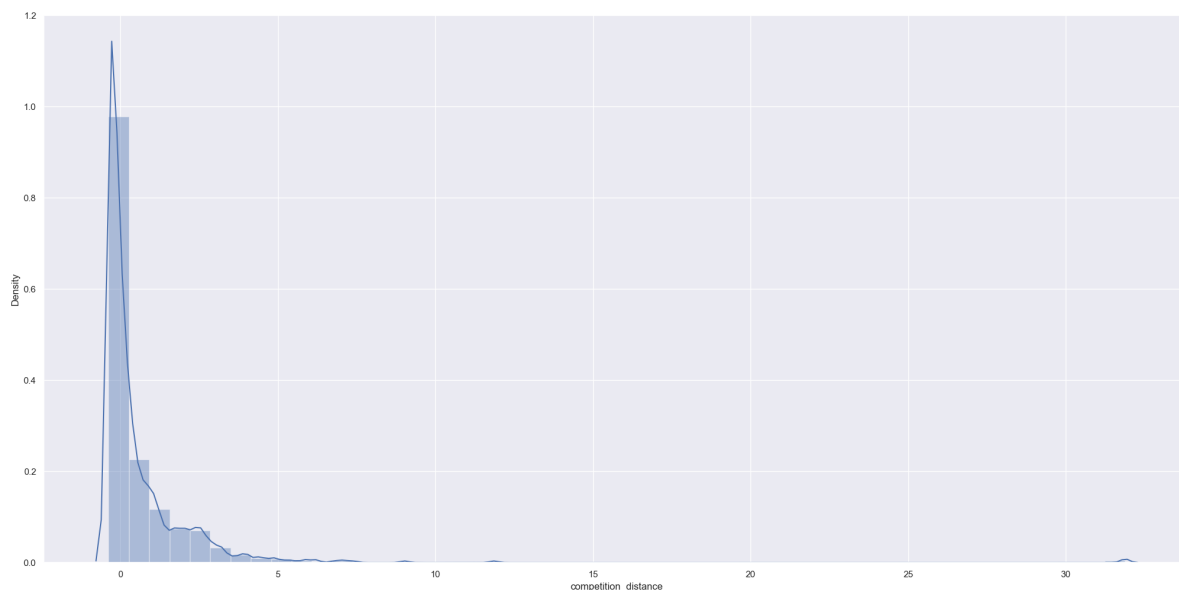
executed in 268ms, finished 05:22:08 2023-04-12



In [55]:

```
1 sns.distplot( df5['competition_distance'] );
```

executed in 10.6s, finished 05:22:18 2023-04-12



In [56]:

```
1 df5.head()
```

executed in 44ms, finished 05:22:18 2023-04-12

Out[56]:

	store	day_of_week	date	sales	promo	state_holiday	school_holiday	store_type	ass
0	1	5	2015-07-31	5263	1	regular_day	1	c	
1	2	5	2015-07-31	6064	1	regular_day	1	a	
2	3	5	2015-07-31	8314	1	regular_day	1	a	
3	4	5	2015-07-31	13995	1	regular_day	1	c	€
4	5	5	2015-07-31	4822	1	regular_day	1	a	

## 6.3 Transformation

### 6.3.1 Encoding

```

In [57]: 1 # State holiday - One Hot Encoding
          2
          3 df5 = pd.get_dummies( df5, prefix=['state_holiday'], columns=['state_hol
          4
          5 # Store type - Label Encoding
          6
          7 le = LabelEncoder()
          8 df5['store_type'] = le.fit_transform( df5['store_type'] )
          9
          10 # This code will be used in the exploitation step as API
          11 #pickle.dump( le, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Cicl
          12
          13 assortment_dict = {'basic': 1, 'extra': 2, 'extended': 3}
          14 df5['assortment'] = df5['assortment'].map( assortment_dict )
          15

```

executed in 696ms, finished 05:22:19 2023-04-12

```

In [58]: 1 df5[['store', 'day_of_week', 'date', 'sales', 'promo', 'school_holiday', 'store_type', 'assortment', 'comp']]
          2
          3
          4
          5
          6
          7
          8
          9
          10
          11
          12
          13
          14
          15
          16
          17
          18
          19
          20
          21
          22
          23
          24
          25
          26
          27
          28
          29
          30
          31
          32
          33
          34
          35
          36
          37
          38
          39
          40
          41
          42
          43
          44
          45
          46
          47
          48
          49
          50
          51
          52
          53
          54
          55
          56
          57
          58
          59
          60
          61
          62
          63
          64
          65
          66
          67
          68
          69
          70
          71
          72
          73
          74
          75
          76
          77
          78
          79
          80
          81
          82
          83
          84
          85
          86
          87
          88
          89
          90
          91
          92
          93
          94
          95
          96
          97
          98
          99
          100
          101
          102
          103
          104
          105
          106
          107
          108
          109
          110
          111
          112
          113
          114
          115
          116
          117
          118
          119
          120
          121
          122
          123
          124
          125
          126
          127
          128
          129
          130
          131
          132
          133
          134
          135
          136
          137
          138
          139
          140
          141
          142
          143
          144
          145
          146
          147
          148
          149
          150
          151
          152
          153
          154
          155
          156
          157
          158
          159
          160
          161
          162
          163
          164
          165
          166
          167
          168
          169
          170
          171
          172
          173
          174
          175
          176
          177
          178
          179
          180
          181
          182
          183
          184
          185
          186
          187
          188
          189
          190
          191
          192
          193
          194
          195
          196
          197
          198
          199
          200
          201
          202
          203
          204
          205
          206
          207
          208
          209
          210
          211
          212
          213
          214
          215
          216
          217
          218
          219
          220
          221
          222
          223
          224
          225
          226
          227
          228
          229
          230
          231
          232
          233
          234
          235
          236
          237
          238
          239
          240
          241
          242
          243
          244
          245
          246
          247
          248
          249
          250
          251
          252
          253
          254
          255
          256
          257
          258
          259
          260
          261
          262
          263
          264
          265
          266
          267
          268
          269
          270
          271
          272
          273
          274
          275
          276
          277
          278
          279
          280
          281
          282
          283
          284
          285
          286
          287
          288
          289
          290
          291
          292
          293
          294
          295
          296
          297
          298
          299
          300
          301
          302
          303
          304
          305
          306
          307
          308
          309
          310
          311
          312
          313
          314
          315
          316
          317
          318
          319
          320
          321
          322
          323
          324
          325
          326
          327
          328
          329
          330
          331
          332
          333
          334
          335
          336
          337
          338
          339
          340
          341
          342
          343
          344
          345
          346
          347
          348
          349
          350
          351
          352
          353
          354
          355
          356
          357
          358
          359
          360
          361
          362
          363
          364
          365
          366
          367
          368
          369
          370
          371
          372
          373
          374
          375
          376
          377
          378
          379
          380
          381
          382
          383
          384
          385
          386
          387
          388
          389
          390
          391
          392
          393
          394
          395
          396
          397
          398
          399
          400
          401
          402
          403
          404
          405
          406
          407
          408
          409
          410
          411
          412
          413
          414
          415
          416
          417
          418
          419
          420
          421
          422
          423
          424
          425
          426
          427
          428
          429
          430
          431
          432
          433
          434
          435
          436
          437
          438
          439
          440
          441
          442
          443
          444
          445
          446
          447
          448
          449
          450
          451
          452
          453
          454
          455
          456
          457
          458
          459
          460
          461
          462
          463
          464
          465
          466
          467
          468
          469
          470
          471
          472
          473
          474
          475
          476
          477
          478
          479
          480
          481
          482
          483
          484
          485
          486
          487
          488
          489
          490
          491
          492
          493
          494
          495
          496
          497
          498
          499
          500
          501
          502
          503
          504
          505
          506
          507
          508
          509
          510
          511
          512
          513
          514
          515
          516
          517
          518
          519
          520
          521
          522
          523
          524
          525
          526
          527
          528
          529
          530
          531
          532
          533
          534
          535
          536
          537
          538
          539
          540
          541
          542
          543
          544
          545
          546
          547
          548
          549
          550
          551
          552
          553
          554
          555
          556
          557
          558
          559
          560
          561
          562
          563
          564
          565
          566
          567
          568
          569
          570
          571
          572
          573
          574
          575
          576
          577
          578
          579
          580
          581
          582
          583
          584
          585
          586
          587
          588
          589
          590
          591
          592
          593
          594
          595
          596
          597
          598
          599
          600
          601
          602
          603
          604
          605
          606
          607
          608
          609
          610
          611
          612
          613
          614
          615
          616
          617
          618
          619
          620
          621
          622
          623
          624
          625
          626
          627
          628
          629
          630
          631
          632
          633
          634
          635
          636
          637
          638
          639
          640
          641
          642
          643
          644
          645
          646
          647
          648
          649
          650
          651
          652
          653
          654
          655
          656
          657
          658
          659
          660
          661
          662
          663
          664
          665
          666
          667
          668
          669
          670
          671
          672
          673
          674
          675
          676
          677
          678
          679
          680
          681
          682
          683
          684
          685
          686
          687
          688
          689
          690
          691
          692
          693
          694
          695
          696
          697
          698
          699
          700
          701
          702
          703
          704
          705
          706
          707
          708
          709
          710
          711
          712
          713
          714
          715
          716
          717
          718
          719
          720
          721
          722
          723
          724
          725
          726
          727
          728
          729
          730
          731
          732
          733
          734
          735
          736
          737
          738
          739
          740
          741
          742
          743
          744
          745
          746
          747
          748
          749
          750
          751
          752
          753
          754
          755
          756
          757
          758
          759
          760
          761
          762
          763
          764
          765
          766
          767
          768
          769
          770
          771
          772
          773
          774
          775
          776
          777
          778
          779
          780
          781
          782
          783
          784
          785
          786
          787
          788
          789
          790
          791
          792
          793
          794
          795
          796
          797
          798
          799
          800
          801
          802
          803
          804
          805
          806
          807
          808
          809
          810
          811
          812
          813
          814
          815
          816
          817
          818
          819
          820
          821
          822
          823
          824
          825
          826
          827
          828
          829
          830
          831
          832
          833
          834
          835
          836
          837
          838
          839
          840
          841
          842
          843
          844
          845
          846
          847
          848
          849
          850
          851
          852
          853
          854
          855
          856
          857
          858
          859
          860
          861
          862
          863
          864
          865
          866
          867
          868
          869
          870
          871
          872
          873
          874
          875
          876
          877
          878
          879
          880
          881
          882
          883
          884
          885
          886
          887
          888
          889
          890
          891
          892
          893
          894
          895
          896
          897
          898
          899
          900
          901
          902
          903
          904
          905
          906
          907
          908
          909
          910
          911
          912
          913
          914
          915
          916
          917
          918
          919
          920
          921
          922
          923
          924
          925
          926
          927
          928
          929
          930
          931
          932
          933
          934
          935
          936
          937
          938
          939
          940
          941
          942
          943
          944
          945
          946
          947
          948
          949
          950
          951
          952
          953
          954
          955
          956
          957
          958
          959
          960
          961
          962
          963
          964
          965
          966
          967
          968
          969
          970
          971
          972
          973
          974
          975
          976
          977
          978
          979
          980
          981
          982
          983
          984
          985
          986
          987
          988
          989
          990
          991
          992
          993
          994
          995
          996
          997
          998
          999
          1000
          1001
          1002
          1003
          1004
          1005
          1006
          1007
          1008
          1009
          1010
          1011
          1012
          1013
          1014
          1015
          1016
          1017
          1018
          1019
          1020
          1021
          1022
          1023
          1024
          1025
          1026
          1027
          1028
          1029
          1030
          1031
          1032
          1033
          1034
          1035
          1036
          1037
          1038
          1039
          1040
          1041
          1042
          1043
          1044
          1045
          1046
          1047
          1048
          1049
          1050
          1051
          1052
          1053
          1054
          1055
          1056
          1057
          1058
          1059
          1060
          1061
          1062
          1063
          1064
          1065
          1066
          1067
          1068
          1069
          1070
          1071
          1072
          1073
          1074
          1075
          1076
          1077
          1078
          1079
          1080
          1081
          1082
          1083
          1084
          1085
          1086
          1087
          1088
          1089
          1090
          1091
          1092
          1093
          1094
          1095
          1096
          1097
          1098
          1099
          1100
          1101
          1102
          1103
          1104
          1105
          1106
          1107
          1108
          1109
          1110
          1111
          1112
          1113
          1114
          1115
          1116
          1117
          1118
          1119
          1120
          1121
          1122
          1123
          1124
          1125
          1126
          1127
          1128
          1129
          1130
          1131
          1132
          1133
          1134
          1135
          1136
          1137
          1138
          1139
          1140
          1141
          1142
          1143
          1144
          1145
          1146
          1147
          1148
          1149
          1150
          1151
          1152
          1153
          1154
          1155
          1156
          1157
          1158
          1159
          1160
          1161
          1162
          1163
          1164
          1165
          1166
          1167
          1168
          1169
          1170
          1171
          1172
          1173
          1174
          1175
          1176
          1177
          1178
          1179
          1180
          1181
          1182
          1183
          1184
          1185
          1186
          1187
          1188
          1189
          1190
          1191
          1192
          1193
          1194
          1195
          1196
          1197
          1198
          1199
          1200
          1201
          1202
          1203
          1204
          1205
          1206
          1207
          1208
          1209
          1210
          1211
          1212
          1213
          1214
          1215
          1216
          1217
          1218
          1219
          1220
          1221
          1222
          1223
          1224
          1225
          1226
          1227
          1228
          1229
          1230
          1231
          1232
          1233
          1234
          1235
          1236
          1237
          1238
          1239
          1240
          1241
          1242
          1243
          1244
          1245
          1246
          1247
          1248
          1249
          1250
          1251
          1252
          1253
          1254
          1255
          1256
          1257
          1258
          1259
          1260
          1261
          1262
          1263
          1264
          1265
          1266
          1267
          1268
          1269
          1270
          1271
          1272
          1273
          1274
          1275
          1276
          1277
          1278
          1279
          1280
          1281
          1282
          1283
          1284
          1285
          1286
          1287
          1288
          1289
          1290
          1291
          1292
          1293
          1294
          1295
          1296
          1297
          1298
          1299
          1300
          1301
          1302
          1303
          1304
          1305
          1306
          1307
          1308
          1309
          1310
          1311
          1312
          1313
          1314
          1315
          1316
          1317
          1318
          1319
          1320
          1321
          1322
          1323
          1324
          1325
          1326
          1327
          1328
          1329
          1330
          1331
          1332
          1333
          1334
          1335
          1336
          1337
          1338
          1339
          1340
          1341
          1342
          1343
          1344
          1345
          1346
          1347
          1348
          1349
          1350
          1351
          1352
          1353
          1354
          1355
          1356
          1357
          1358
          1359
          1360
          1361
          1362
          1363
          1364
          1365
          1366
          1367
          1368
          1369
          1370
          1371
          1372
          1373
          1374
          1375
          1376
          1377
          1378
          1379
          1380
          1381
          1382
          1383
          1384
          1385
          1386
          1387
          1388
          1389
          1390
          1391
          1392
          1393
          1394
          1395
          1396
          1397
          1398
          1399
          1400
          1401
          1402
          1403
          1404
          1405
          1406
          1407
          1408
          1409
          1410
          1411
          1412
          1413
          1414
          1415
          1416
          1417
          1418
          1419
          1420
          1421
          1422
          1423
          1424
          1425
          1426
          1427
          1428
          1429
          1430
          1431
          1432
          1433
          1434
          1435
          1436
          1437
          1438
          1439
          1440
          1441
          1442
          1443
          1444
          1445
          1446
          1447
          1448
          1449
          1450
          1451
          1452
          1453
          1454
          1455
          1456
          1457
          1458
          1459
          1460
          1461
          1462
          1463
          1464
          1465
          1466
          1467
          1468
          1469
          1470
          1471
          1472
          1473
          1474
          1475
          1476
          1477
          1478
          1479
          1480
          1481
          1482
          1483
          1484
          1485
          1486
          1487
          1488
          1489
          1490
          1491
          1492
          1493
          1494
          1495
          1496
          1497
          1498
          1499
          1500
          1501
          1502
          1503
          1504
          1505
          1506
          1507
          1508
          1509
          1510
          1511
          1512
          1513
          1514
          1515
          1516
          1517
          1518
          1519
          1520
          1521
          1522
          1523
          1524
          1525
          1526
          1527
          1528
          1529
          1530
          1531
          1532
          1533
          1534
          1535
          1536
          1537
          1538
          1539
          1540
          1541
          1542
          1543
          1544
          1545
          1546
          1547
          1548
          1549
          1550
          1551
          1552
          1553
          1554
          1555
          1556
          1557
          1558
          1559
          1560
          1561
          1562
          1563
          1564
          1565
          1566
          1567
          1568
          1569
          1570
          1571
          1572
          1573
          1574
          1575
          1576
          1577
          1578
          1579
          1580
          1581
          1582
          1583
          1584
          1585
          1586
          1587
          1588
          1589
          1590
          1591
          1592
          1593
          1594
          1595
          1596
          1597
          1598
          1599
          1600
          1601
          1602
          1603
          1604
          1605
          1606
          1607
          1608
          1609
          1610
          1611
          1612
          1613
          1614
          1615
          1616
          1617
          1618
          1619
          1620
          1621
          1622
          1623
          1624
          1625
          1626
          1627
          1628
          1629
          1630
          1631
          1632
          1633
          1634
          1635
          1636
          1637
          1638
          1639
          1640
          1641
          1642
          1643
          1644
          1645
          1646
          1647
          1648
          1649
          1650
          1651
          1652
          1653
          1654
          1655
          1656
          1657
          1658
          1659
          1660
          1661
          1662
          1663
          1664
          1665
          1666
          1667
          1668
          1669
          1670
          1671
          1672
          1673
          1674
          1675
          1676
          1677
          1678
          1679
          1680
          1681
          1682
          1683
          1684
          1685
          1686
          1687
          1688
          1689
          1690
          1691
          1692
          1693
          1694
          1695
          1696
          1697
          1698
          1699
          1700
          1701
          1702
          1703
          1704
          1705
          1706
          1707
          1708
          1709
          1710
          1711
          1712
          1713
          1714
          1715
          1716
          1717
          1718
          1719
          1720
          1721
          1722
          1723
          1724
          1725
          1726
          1727
          1728
          1729
          1730
          1731
          1732
          1733
          1734
          1735
          1736
          1737
          1738
          1739
          1740
          1741
          1742
          1743
          1744
          1745
          1746
          1747
          1748
          1749
          1750
          1751
          1752
          1753
          1754
          1755
          1756
          1757
          1758
          1759
          1760
          1761
          1762
          1763
          1764
          1765
          1766
          1767
          1768
          1769
          1770
          1771
          1772
          1773
          1774
          1775
          1776
          1777
          1778
          1779
          1780
          1781
          1782
          1783
          1784
          1785
          1786
          1787
          1788
          1789
          1790
          1791
          1792
          1793
          1794
          1795
          1796
          1797
          1798
          1799
          1800
          1801
          1802
          1803
          1804
          1805
          1806
          1807
          1808
          1809
          1810
          1811
          1812
          1813
          1814
          1815
          1816
          1817
          1818
          1819
          1820
          1821
          1822
          1823
          1824
          1825
          1826
          1827
          1828
          1829
          1830
          1831
          1832
          1833
          1834
          1835
          1836
          1837
          1838
          1839
          1840
          1841
          1842
          1843
          1844
          1845
          1846
          1847
          1848
          1849
          1850
          1851
          1852
          1853
          1854
          1855
          1856
          1857
          1858
          18
```

In [60]:

```

1  # Day of Week
2
3  df5['day_of_week_sin'] = df5['day_of_week'].apply( lambda x: np.sin( x *
4  df5['day_of_week_cos'] = df5['day_of_week'].apply( lambda x: np.cos( x *
5
6  # Month
7
8  df5['month_sin'] = df5['month'].apply( lambda x: np.sin( x * ( 2 * np.pi
9  df5['month_cos'] = df5['month'].apply( lambda x: np.cos( x * ( 2 * np.pi
10
11 # Day
12
13 df5['day_sin'] = df5['day'].apply( lambda x: np.sin( x * ( 2 * np.pi/30
14 df5['day_cos'] = df5['day'].apply( lambda x: np.cos( x * ( 2 * np.pi/30
15
16 # Week of Year
17
18 df5['week_of_year_sin'] = df5['week_of_year'].apply( lambda x: np.sin( x
19 df5['week_of_year_cos'] = df5['week_of_year'].apply( lambda x: np.cos( x
20

```

executed in 13.9s, finished 05:22:33 2023-04-12

In [61]:

```

1  df5.head()

```

executed in 45ms, finished 05:22:33 2023-04-12

Out[61]:

	store	day_of_week	date	sales	promo	school_holiday	store_type	assortment	co
0	1	5	2015-07-31	8.568646	1	1	2	1	
1	2	5	2015-07-31	8.710290	1	1	0	1	
2	3	5	2015-07-31	9.025816	1	1	0	1	
3	4	5	2015-07-31	9.546527	1	1	2	3	
4	5	5	2015-07-31	8.481151	1	1	0	1	

## 7 Feature Selection (Sixth Step)

In [62]:

```

1  df6 = df5[['store', 'day_of_week', 'date', 'sales', 'promo', 'school_holiday', 'store_type', 'assortment', 'co']]

```

executed in 312ms, finished 05:22:33 2023-04-12

### 7.1 Split the Dataframe into Training and Test

In [63]:

```

1  # Delete the columns that were used to create the new columns
2
3  cols_drop = ['week_of_year', 'day', 'month', 'day_of_week', 'promo_since
4  df6 = df6.drop( cols_drop, axis=1 )

```

executed in 123ms, finished 05:22:33 2023-04-12

```
In [64]: 1 # Training dataset
2
3 X_train = df6[df6['date'] < '2015-06-19']
4 y_train = X_train['sales']
5
6 # Test dataset
7
8 X_test = df6[df6['date'] >= '2015-06-19']
9 y_test = X_test['sales']
10
11 print( f"Training Min Date { X_train['date'].min() }" )
12 print( f"Training Max Date { X_train['date'].max() }" )
13
14 print( f"\nTest Min date {X_test['date'].min()}" )
15 print( f"Test Max date {X_test['date'].max()}" )
```

executed in 217ms, finished 05:22:34 2023-04-12

Training Min Date 2013-01-01 00:00:00

Training Max Date 2015-06-18 00:00:00

Test Min date 2015-06-19 00:00:00

Test Max date 2015-07-31 00:00:00

## 7.2 Boruta as Feature Selector

```
In [65]: 1 # training and test dataset for Boruta
2
3 #X_train_n = X_train.drop( ['date', 'sales'], axis=1 ).values
4 #y_train_n = y_train.values.ravel()
5
6 # define RandomForestRegressor
7
8 #rf = RandomForestRegressor( n_jobs=-1 )
9
10 # define Boruta
11
```

executed in 11ms, finished 05:22:34 2023-04-12

### 7.2.1 Best Features Chosen by Boruta

```
In [66]: 1 #cols_selected = boruta.support_.tolist()
2
3 # best features
4 #X_train_fs = X_train.drop( ['date', 'sales'], axis=1 )
5 #cols_selected_boruta = X_train_fs.iloc[:, cols_selected].columns.to_list()
6
7 # not selected boruta
```

executed in 72ms, finished 05:22:34 2023-04-12

## 7.3 Manually Feature Adding

```
In [67]: 1 cols_selected_boruta = [  
2         'store',  
3         'promo',  
4         'store_type',  
5         'assortment',  
6         'competition_distance',  
7         'competition_open_since_month',  
8         'competition_open_since_year',  
9         'promo2',  
10        'promo2_since_week',  
11        'promo2_since_year',  
12        'competition_time_month',  
13        'promo_time_week',  
14        'day_of_week_sin',  
15        'day_of_week_cos',  
16        'month_sin',  
17        'month_cos',  
18        'day_sin',  
19        'day_cos',  
20        'week_of_year_sin',  
21        'week_of_year_cos']  
22  
23 # columns to add  
24 feat_to_add = ['date', 'sales']  
25  
26 cols_selected_boruta_full = cols_selected_boruta.copy()
```

executed in 42ms, finished 05:22:34 2023-04-12

## 8 Machine Learning Modelling (Seventh Step)

```
In [68]: 1 x_train = X_train[ cols_selected_boruta ]  
2         x_test = X_test[ cols_selected_boruta ]  
3  
4         # Time Series Data Preparation
```

executed in 217ms, finished 05:22:34 2023-04-12

### 8.1 Average Model

In [69]:

```

1 aux1 = x_test.copy()
2 aux1['sales'] = y_test.copy()
3
4 # Prediction
5
6 aux2 = aux1[['store', 'sales']].groupby( 'store' ).mean().reset_index().
7 aux1 = pd.merge( aux1, aux2, how='left', on='store' )
8 yhat_baseline = aux1['predictions']
9
10 # Performance
11
12 baseline_result = ml_error( 'Average Model', np.expm1( y_test ), np.expm1
13 baseline_result

```

executed in 106ms, finished 05:22:34 2023-04-12

Out[69]:

	Model Name	MAE	MAPE	RMSE
0	Average Model	1354.800353	0.455051	1835.135542

## 8.2 Linear Regression

In [70]:

```

1 # Model
2
3 lr = LinearRegression().fit( x_train, y_train )
4
5 # Prediction
6
7 yhat_lr = lr.predict( x_test )
8
9 # Performance
10
11 lr_result = ml_error( 'Linear Regression', np.expm1( y_test ), np.expm1(
12 lr_result

```

executed in 2.35s, finished 05:22:36 2023-04-12

Out[70]:

	Model Name	MAE	MAPE	RMSE
0	Linear Regression	1867.089774	0.292694	2671.049215

### 8.2.1 Linear Regression Model - Cross Validation

In [71]:

```

1 lr_result_cv = cross_validation( x_training, 5, 'Linear Regression', lr,
2 lr_result_cv

```

executed in 5.16s, finished 05:22:42 2023-04-12

Out[71]:

	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 Linear Regression Regularized Model - Lasso

```
In [72]: 1 # Model
          2
          3 lrr = Lasso( alpha=0.01 ).fit( x_train, y_train )
          4
          5 # Prediction
          6
          7 yhat_lrr = lrr.predict( x_test )
          8
          9 # Performance
         10
         11 lrr_result = ml_error( 'Linear Regression - Lasso', np.expm1( y_test ),
         12 lrr_result
          13
executed in 1.03s, finished 05:22:43 2023-04-12
```

Out[72]:

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

### 8.3.1 Lasso - Cross Validation

```
In [73]: 1 lrr_result_cv = cross_validation( x_training, 5, 'Lasso', lrr, verbose=F
          2 lrr_result_cv
          3
executed in 3.49s, finished 05:22:46 2023-04-12
```

Out[73]:

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

## 8.4 Random Forest Regressor

```

In [74]: 1 # Model
          2
          3 rf = RandomForestRegressor( n_estimators=100, n_jobs=-1, random_state=42
          4
          5 # Prediction
          6
          7 yhat_rf = rf.predict( x_test )
          8
          9 # Performance
         10
         11 rf_result = ml_error( 'Random Forest Regressor', np.expm1( y_test ), np.
         12 rf_result
         13
executed in 4m 51s, finished 05:27:37 2023-04-12

```

Out[74]:

	Model Name	MAE	MAPE	RMSE
0	Random Forest Regressor	679.598831	0.099913	1011.119437

### 8.4.1 Random Forest - Cross Validation

```

In [75]: 1 rf_result_cv = cross_validation( x_training, 5, 'Random Forest Regressor
          2 rf_result_cv
          3
executed in 18m 48s, finished 05:46:25 2023-04-12

```

KFold Number: 5

KFold Number: 4

KFold Number: 3

KFold Number: 2

KFold Number: 1

Out[75]:

	Model Name	MAE CV	MAPE CV	RMSE CV
0	Random Forest Regressor	836.61 +/- 217.1	0.12 +/- 0.02	1254.3 +/- 316.17

## 8.5 XGBoost Regressor



```

In [76]: 1 # Model
          2
          3 model_xgb = xgb.XGBRegressor( objective='reg:squarederror',
          4                               n_estimators=100,
          5                               eta=0.01,
          6                               max_depth=10,
          7                               subsample=0.7,
          8                               colsample_bytree=0.9 ).fit( x_train, y_train )
          9
          10 # Prediction
          11
          12 yhat_xgb = model_xgb.predict( x_test )
          13
          14 # Performance
          15
          16 xgb_result = ml_error( 'XGBoost Regressor', np.expm1( y_test ), np.expm1
          17 xgb_result
          18
          executed in 1m 38.9s, finished 05:48:04 2023-04-12

```

Out[76]:

	Model Name	MAE	MAPE	RMSE
0	XGBoost Regressor	6683.544086	0.949457	7330.812159

## 8.5.1 XGBoost - Cross Validation

```

In [77]: 1 xgb_result_cv = cross_validation( x_training, 5, 'XGBoost Regressor', mo
          2 xgb_result_cv
          3
          executed in 6m 41s, finished 05:54:46 2023-04-12

```

KFold Number: 5

KFold Number: 4

KFold Number: 3

KFold Number: 2

KFold Number: 1

Out[77]:

	Model Name	MAE CV	MAPE CV	RMSE CV
0	XGBoost Regressor	7049.17 +/- 588.63	0.95 +/- 0.0	7715.17 +/- 689.51

## 8.6 Compare the Performance

### 8.6.1 Single Performance

```
In [78]: 1 modelling_result = pd.concat( [baseline_result, lr_result, lrr_result, r
2 modelling_result.sort_values( 'RMSE' )
```

executed in 1.16s, finished 05:54:47 2023-04-12

Out[78]:

	Model Name	MAE	MAPE	RMSE
0	Random Forest Regressor	679.598831	0.099913	1011.119437
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
0	XGBoost Regressor	6683.544086	0.949457	7330.812159

## 8.6.2 Real Performance (Cross Validation)

```
In [79]: 1 modelling_result_cv = pd.concat( [lr_result_cv, lrr_result_cv, rf_result
2 modelling_result_cv
```

executed in 75ms, finished 05:54:47 2023-04-12

Out[79]:

	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	836.61 +/- 217.1	0.12 +/- 0.02	1254.3 +/- 316.17
0	XGBoost Regressor	7049.17 +/- 588.63	0.95 +/- 0.0	7715.17 +/- 689.51

# 9 Hyperparameter Fine Tuning (Eitghth Step)

## 9.1 Random Search

```
In [80]: 1 '''param = {'n_estimators': [1000, 1500, 2000, 2500, 3000],
2         'eta': [0.01, 0.03],
3         'max_depth': [3, 6, 9],
4         'subsample': [0.1, 0.4, 0.7],
5         'colsample_bytree': [0.3, 0.6, 0.9],
6         'min_child_weight': [3, 8, 12]}
7
```

executed in 59ms, finished 05:54:47 2023-04-12

Out[80]: "param = {'n\_estimators': [1000, 1500, 2000, 2500, 3000],\n 'eta': [0.01, 0.03],\n 'max\_depth': [3, 6, 9],\n 'subsample': [0.1, 0.4, 0.7],\n 'colsample\_bytree': [0.3, 0.6, 0.9],\n 'min\_child\_weight': [3, 8, 12]}\n\nMX\_EVAL = 10"

```

In [81]: 1 '''final_result = pd.DataFrame()
2
3 for i in range( MAX_EVAL ):
4     # Choose the values for parameters randomly
5     hp = { k: np.random.choice( v, 1 )[0] for k, v in param.items() }
6     print( hp )
7
8     # model
9     model_xgb = xgb.XGBRegressor( objective='reg:squarederror',
10                                n_estimators=hp['n_estimators'],
11                                eta=hp['eta'],
12                                max_depth=hp['max_depth'],
13                                subsample=hp['subsample'],
14                                colsample_bytree=hp['colsample_bytree'],
15                                min_child_weight=hp['min_child_weight']
16
17     # performance
18     result = cross_validation( x_training, 10, 'XGBoost Regressor', mode
19     final_result = pd.concat( [final_result, result] )
20
21
executed in 21ms, finished 05:54:47 2023-04-12

```

```

Out[81]: "final_result = pd.DataFrame()\n\nfor i in range( MAX_EVAL ):\n    # Choose the values for parameters randomly\n    hp = { k: np.random.choice( v, 1 )[0]\nfor k, v in param.items() }\n    print( hp )\n    \n    # model\n    model_xgb = xgb.XGBRegressor( objective='reg:squarederror',\n                                n_estimators=hp['n_estimators'], \n                                eta=hp['eta'], \n                                max_depth=hp['max_depth'], \n                                subsample=hp['subsample'],\n                                colsample_bytree=hp['colsample_bytree'],\n                                min_child_weight=hp['min_child_weight'] )\n    # performance\n    result = cross_validation( x_training, 10, 'XGBoost Regressor', model_xgb, verbose=True )\n    final_result = pd.concat( [final_result, result] )\n\nfinal_result"

```

```

In [82]:
executed in 10.9s, finished 05:55:00 2023-04-12

NameError: name 'final_result' is not defined

```

## 9.2 Final Model

```

In [87]: 1 # After testing the parameters, The most precise one was chosen below:
2
3 # {'n_estimators': 3000, 'eta': 0.03, 'max_depth': 9, 'subsample': 0.1,
4
5 param_tuned = {
6     'n_estimators': 3000,
7     'eta': 0.03,
8     'max_depth': 9,
9     'subsample': 0.1,
10    'colsample_bytree': 0.6,
11    'min_child_weight': 12
12 }

```

```

In [88]: 1 # model
          2 model_xgb_tuned = xgb.XGBRegressor( objective='reg:squarederror',
          3                                     n_estimators=param_tuned['n_estimators'],
          4                                     eta=param_tuned['eta'],
          5                                     max_depth=param_tuned['max_depth'],
          6                                     subsample=param_tuned['subsample'],
          7                                     colsample_bytree=param_tuned['colsample_bytree'],
          8                                     min_child_weight=param_tuned['min_child_weight'],
          9
         10 # prediction
         11 yhat_xgb_tuned = model_xgb_tuned.predict( x_test )
         12
         13 # performance
         14 xgb_result_tuned = ml_error( 'XGBoost Regressor', np.exp( y_test ), np.exp( yhat_xgb_tuned ) )

```

executed in 29m 35s, finished 06:34:13 2023-04-12

Out[88]:

	Model Name	MAE	MAPE	RMSE
0	XGBoost Regressor	640.526121	0.093286	934.833866

```

In [89]: 1 mpe = mean_percentage_error( np.exp( y_test ), np.exp( yhat_xgb_tuned ) )

```

executed in 265ms, finished 18:53:02 2023-04-12

Out[89]: 0.0015817572750591092

## 10 Interpreting the Errors (Nineth Step)

MAE (Mean Absolute Error) - equal weight to all errors

MAPE (Mean Absolute Percentage Error) - shows how far the prediction is from the real value in percentage

RMSE (Root mean square Error) - Shows a more precisely result than MAE

MPE ( Mean percentage Error) - Most used to increase the precise of the model, and idicates if the model is underestimating or superestimating

```

In [90]: 1 df7 = X_test[ cols_selected_boruta_full ]
          2
          3 # Rescale
          4
          5 df7['sales'] = np.exp( df7['sales'] )
          6 df7['predictions'] = np.exp( yhat_xgb_tuned )

```

executed in 254ms, finished 18:53:07 2023-04-12

### 10.1 Business Performance

```

In [91]: 1 # Sum predictions
2
3 df71 = df7[['store', 'predictions']].groupby( 'store' ).sum().reset_index
4
5 # MAE and MAPE
6
7 df7_aux1 = df7[['store', 'sales', 'predictions']].groupby( 'store' ).agg
8 df7_aux2 = df7[['store', 'sales', 'predictions']].groupby( 'store' ).agg
9     lambda x: mean_absolute_percentage_error( x['sales'], x['predictions'] )
10
11 # Merge
12
13 df7_aux3 = pd.merge( df7_aux1, df7_aux2, how='inner', on='store' )
14 df72 = pd.merge( df71, df7_aux3, how='inner', on='store' )
15
16 # Scenariosdf91
17
18 df72['worst_scenario'] = df72['predictions'] - df72['MAE']
19 df72['best_scenario'] = df72['predictions'] + df72['MAE']
20 7
21 # order columns
22
23 df72 = df72[['store', 'predictions', 'worst_scenario', 'best_scenario',
24

```

executed in 1.67s, finished 18:53:11 2023-04-12

```

In [92]: 1 df72
2
3 df72
4
5 df72
6
7 df72
8
9 df72
10
11 df72
12
13 df72
14
15 df72
16
17 df72
18
19 df72
20
21 df72
22
23 df72
24

```

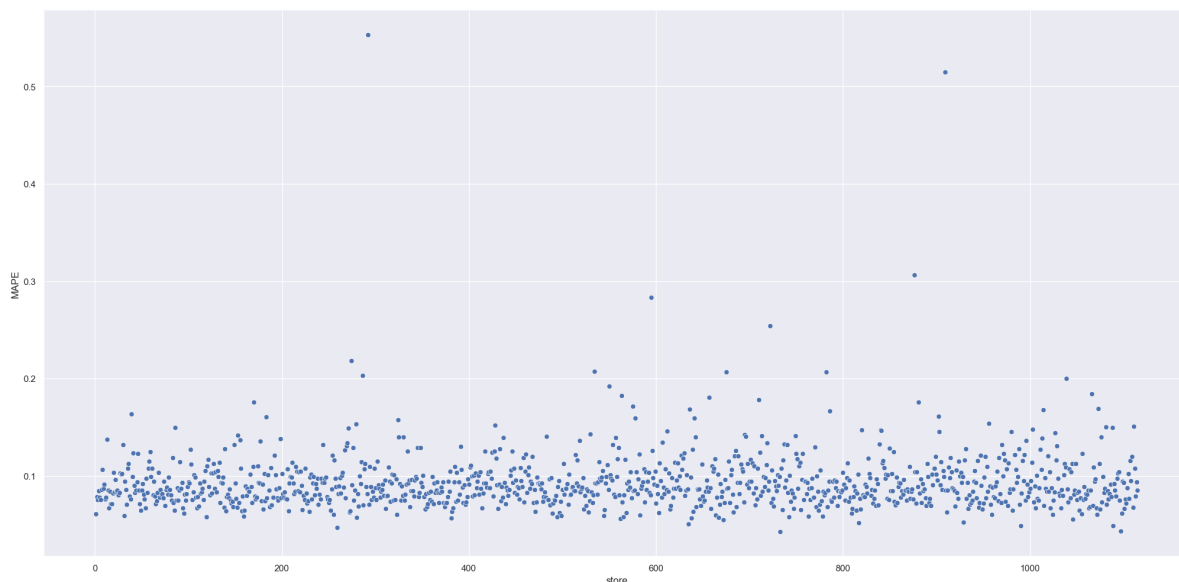
executed in 210ms, finished 18:53:14 2023-04-12

Out[92]:

	store	predictions	worst_scenario	best_scenario	MAE	MAPE
<b>291</b>	292	104620.882812	101291.098479	107950.667146	3329.784334	0.553023
<b>908</b>	909	231406.000000	223667.388799	239144.611201	7738.611201	0.515073
<b>875</b>	876	197739.859375	193747.744894	201731.973856	3992.114481	0.306496
<b>594</b>	595	376960.625000	372845.696223	381075.553777	4114.928777	0.283111
<b>721</b>	722	350205.156250	348288.692185	352121.620315	1916.464065	0.254349

In [93]:

executed in 4.57s, finished 18:53:21 2023-04-12



## 10.2 Total Performance

In [94]:

```
1 df73 = df72[['predictions', 'worst_scenario', 'best_scenario']].apply(
2     lambda x: np.sum( x ), axis=0 ).reset_index().rename( columns={'inde
3 df73['Values'] = df73['Values'].map( 'R$ {:.2f}'.format )
4 df73
```

executed in 38ms, finished 18:53:31 2023-04-12

Out[94]:

	Scenario	Values
0	predictions	R\$ 284,342,400.00
1	worst_scenario	R\$ 283,623,991.55
2	best_scenario	R\$ 285,060,814.20

## 10.3 Machine Learning Performance

In [95]:

```
1 df7['error'] = df7['sales'] - df7['predictions']
2 df7['error_rate'] = df7['predictions'] / df7['sales']
```

executed in 36ms, finished 18:53:33 2023-04-12

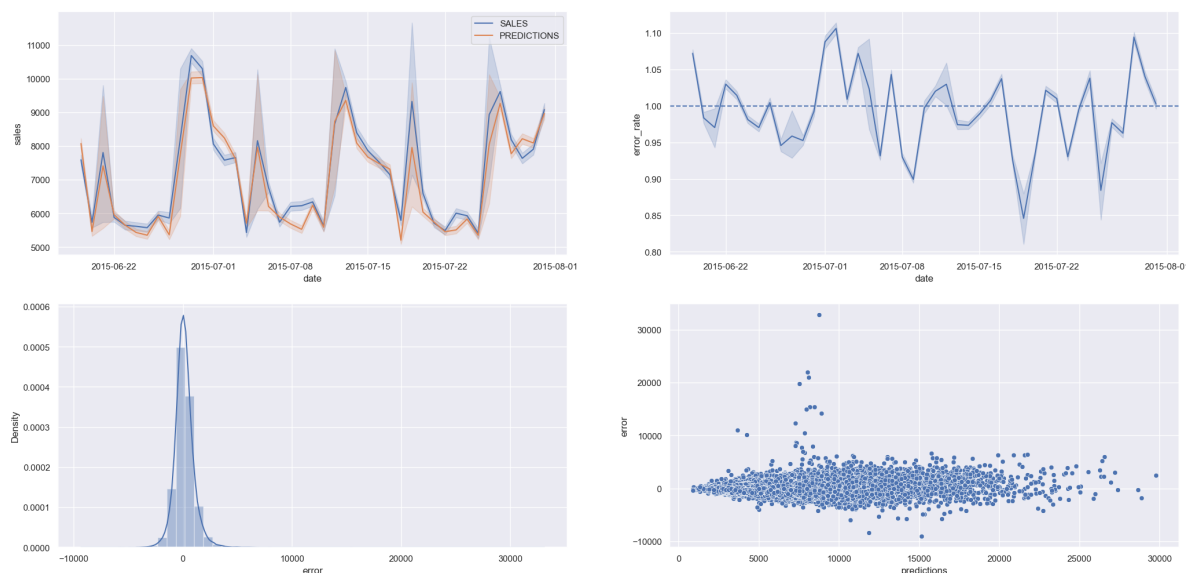
In [96]:

```

1 plt.subplot(2, 2, 1)
2 sns.lineplot( x='date', y='sales', data=df7, label='SALES' );
3 sns.lineplot( x='date', y='predictions', data=df7, label='PREDICTIONS' )
4
5 plt.subplot(2, 2, 2)
6 sns.lineplot( x='date', y='error_rate', data=df7 );
7 plt.axhline( 1, linestyle='--' )
8
9 plt.subplot(2, 2, 3)
10 sns.distplot( df7['error'] );
11
12 plt.subplot(2, 2, 4)
13 sns.scatterplot( x='predictions', y='error', data=df7 );

```

executed in 10.3s, finished 18:53:45 2023-04-12



## 11 Deploy Model to Production ( With Tester Local API ) (Tenth Step)

In [ ]:

```

1 # Save Trained Model
2
3 #pickle.dump( model_xgb_tuned, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\
4

```

executed in 0ms, finished 06:01:43 2023-04-12

### 11.1 Rossmann Class

In [97]:

```

1 import pickle
2 import inflection
3 import pandas as pd
4 import numpy as np
5 import math
6 import datetime
7
8
9 class Rossmann(object):
10     def __init__(self):
11         self.home_path = 'C:\\\\Users\\\\gabre\\DS IN PROGRESS\\DS_2023\\Cic
12         self.competition_distance_scaler = pickle.load( open( self.home_
13         self.competition_time_month_scaler = pickle.load( open( self.hor
14         self.promo_time_week_scaler = pickle.load( open( self.home_path
15         self.year_scaler = pickle.load( open( self.home_path + 'paramete
16         self.store_type_scaler = pickle.load( open( self.home_path + 'pa
17
18     def data_cleaning(self, df1):
19         ## 1.1. Rename Columns
20         cols_old = ['Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'Stat
21                     'StoreType', 'Assortment', 'CompetitionDistance', '(
22                     'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWe
23
24         snakecase = lambda x: inflection.underscore(x)
25
26         cols_new = list(map(snakecase, cols_old))
27
28         # rename
29         df1.columns = cols_new
30
31         ## 1.3. Data Types
32         df1['date'] = pd.to_datetime(df1['date'])
33
34         ## 1.5. Fillout NA
35         # competition_distance
36         df1['competition_distance'] = df1['competition_distance'].apply(
37
38         # competition_open_since_month
39         df1['competition_open_since_month'] = df1.apply(
40             lambda x: x['date'].month if math.isnan(x['competition_open_
41                 'competition_open_since_month'], axis=1)
42
43         # competition_open_since_year
44         df1['competition_open_since_year'] = df1.apply(
45             lambda x: x['date'].year if math.isnan(x['competition_open_s
46                 'competition_open_since_year'], axis=1)
47
48         # promo2_since_week
49         df1['promo2_since_week'] = df1.apply(
50             lambda x: x['date'].week if math.isnan(x['promo2_since_week
51
52         # promo2_since_year
53         df1['promo2_since_year'] = df1.apply(
54             lambda x: x['date'].year if math.isnan(x['promo2_since_year'
55

```



```
56     # promo_interval
57     month_map = {1: 'Jan', 2: 'Fev', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug', 9: 'Sep',
58                  10: 'Oct', 11: 'Nov', 12: 'Dec'}
59
60     df1['promo_interval'].fillna(0, inplace=True)
61
62     df1['month_map'] = df1['date'].dt.month.map(month_map)
63
64     df1['is_promo'] = df1[['promo_interval', 'month_map']].apply(
65         lambda x: 0 if x['promo_interval'] == 0 else 1 if x['month_map'] in month_map else 0,
66         axis=1)
67
68     ## 1.6. Change Data Types
69     # competition
70     df1['competition_open_since_month'] = df1['competition_open_since'].dt.month
71     df1['competition_open_since_year'] = df1['competition_open_since'].dt.year
72
73     # promo2
74     df1['promo2_since_week'] = df1['promo2_since_week'].astype(int)
75     df1['promo2_since_year'] = df1['promo2_since_year'].astype(int)
76
77     return df1
78
79     def feature_engineering(self, df2):
80         # year
81         df2['year'] = df2['date'].dt.year
82
83         # month
84         df2['month'] = df2['date'].dt.month
85
86         # day
87         df2['day'] = df2['date'].dt.day
88
89         # week of year
90         df2['week_of_year'] = df2['date'].dt.weekofyear
91
92         # year week
93         df2['year_week'] = df2['date'].dt.strftime('%Y-%W')
94
95         # competition since
96         df2['competition_since'] = df2.apply(
97             lambda x: datetime.datetime(year=x['competition_open_since_year'],
98                                         month=x['competition_open_since_month'],
99                                         day=1), axis=1)
100         df2['competition_time_month'] = ((df2['date'] - df2['competition_since']).dt.days // 30).astype(int)
101
102         # promo since
103         df2['promo_since'] = df2['promo2_since_year'].astype(str) + '-' + df2['promo2_since_week'].astype(str)
104         df2['promo_since'] = df2['promo_since'].apply(
105             lambda x: datetime.datetime.strptime(x + '-1', '%Y-%W-%w').date(), axis=1)
106         df2['promo_time_week'] = ((df2['date'] - df2['promo_since']).dt.days // 7).astype(int)
107
108         # assortment
109         df2['assortment'] = df2['assortment'].apply(
110             lambda x: 'basic' if x == 'a' else 'extra' if x == 'b' else 'normal' if x == 'c' else 'other')
111
```

```

112     # state holiday
113     df2['state_holiday'] = df2['state_holiday'].apply(lambda
114                                                         x: 'public
115
116     # 3.0. PASSO 03 - FILTRAGEM DE VARIÁVEIS
117     ## 3.1. Filtragem das Linhas
118     df2 = df2[df2['open'] != 0]
119
120     ## 3.2. Selecao das Colunas
121     cols_drop = ['open', 'promo_interval', 'month_map']
122     df2 = df2.drop(cols_drop, axis=1)
123
124     return df2
125
126     def data_preparation(self, df5):
127         ## 5.2. Rescaling
128         # competition distance
129         df5['competition_distance'] = self.competition_distance_scaler.
130             df5[['competition_distance']].values)
131
132         # competition time month
133         df5['competition_time_month'] = self.competition_time_month_scal
134             df5[['competition_time_month']].values)
135
136         # promo time week
137         df5['promo_time_week'] = self.promo_time_week_scaler.fit_transfo
138
139         # year
140         df5['year'] = self.year_scaler.fit_transform(df5[['year']].value
141
142         ### 5.3.1. Encoding
143         # state_holiday - One Hot Encoding
144         df5 = pd.get_dummies(df5, prefix=['state_holiday'], columns=['st
145
146         # store_type - Label Encoding
147         df5['store_type'] = self.store_type_scaler.fit_transform(df5['st
148
149         # assortment - Ordinal Encoding
150         assortment_dict = {'basic': 1, 'extra': 2, 'extended': 3}
151         df5['assortment'] = df5['assortment'].map(assortment_dict)
152
153         ### 5.3.3. Nature Transformation
154         # day of week
155         df5['day_of_week_sin'] = df5['day_of_week'].apply(lambda x: np.s
156         df5['day_of_week_cos'] = df5['day_of_week'].apply(lambda x: np.c
157
158         # month
159         df5['month_sin'] = df5['month'].apply(lambda x: np.sin(x * (2. *
160         df5['month_cos'] = df5['month'].apply(lambda x: np.cos(x * (2. *
161
162         # day
163         df5['day_sin'] = df5['day'].apply(lambda x: np.sin(x * (2. * np.
164         df5['day_cos'] = df5['day'].apply(lambda x: np.cos(x * (2. * np.
165
166         # week of year
167         df5['week_of_year_sin'] = df5['week_of_year'].apply(lambda x: np

```

```
168         df5['week_of_year_cos'] = df5['week_of_year'].apply(lambda x: np
169
170     cols_selected = ['store', 'promo', 'store_type', 'assortment',
171                     'competition_open_since_month',
172                     'competition_open_since_year', 'promo2', 'promo
173                     'competition_time_month', 'promo_time_week',
174                     'day_of_week_sin', 'day_of_week_cos', 'month_si
175                     'week_of_year_sin', 'week_of_year_cos']
176
177     return df5[cols_selected]
178
179     def get_prediction(self, model, original_data, test_data):
180         # prediction
181
182         pred = model.predict(test_data)
183
184         # join pred into the original data
185
186         original_data['prediction'] = np.expm1(pred)
187
188         return original_data.to_json(orient='records', date_format='iso
189
```

executed in 156ms, finished 18:53:58 2023-04-12

## 11.2 API Handler

In [ ]:

```
1  '''import os
2  import pickle
3  import pandas as pd
4  from flask import Flask, request, Response
5  from rossmann.Rossmann import Rossmann
6
7  # loading model local Test
8
9  model = pickle.load(open(
10      'C:\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de_Preparacao\\Dat
11      'rb'))
12
13  # initialize API
14  app = Flask(__name__)
15
16
17  @app.route('/rossmann/predict', methods=['POST'])
18  def rossmann_predict():
19      test_json = request.get_json()
20
21      if test_json: # there is data
22          if isinstance(test_json, dict): # unique example
23              test_raw = pd.DataFrame(test_json, index=[0])
24
25          else: # multiple example
26              test_raw = pd.DataFrame(test_json, columns=test_json[0].keys
27
28          # Instantiate Rossmann class
29          pipeline = Rossmann()
30
31          # data cleaning
32          df1 = pipeline.data_cleaning(test_raw)
33
34          # feature engineering
35          df2 = pipeline.feature_engineering(df1)
36
37          # data preparation
38          df3 = pipeline.data_preparation(df2)
39
40          # prediction
41          df_response = pipeline.get_prediction(model, test_raw, df3)
42
43          return df_response
44
45      else:
46          return Reponse('{}', status=200, mimetype='application/json')
47
48  if __name__ == '__main__':
49      app.run( '192.168.1.104' )'''
50
```

executed in 0ms, finished 06:01:43 2023-04-12

## 11.3 API Tester

```
In [107]: 1 # Loading test dataset
          2
          3 df10 = pd.read_csv( 'C:\\\\Users\\gabre\\DS IN PROGRESS\\DS_2023\\Ciclo_de
          4
          executed in 178ms, finished 06:07:10 2023-04-13
```

```
In [119]: 1 # merge test dataset + store
          2
          3 df_test = pd.merge( df10, df_store_raw, how='left', on='Store' )
          4
          5 # Choosing randoly stores just to test
          6
          7 random_stores_test = []
          8
          9 for i in range(10):
         10     random_stores_test.append( random.randint( 1, 500 ) )
         11
         12 # choose store for prediction
         13
         14 df_test = df_test[df_test['Store'].isin( random_stores_test )]
         15
         16 # remove closed days
         17
         18 df_test = df_test[df_test['Open'] != 0]
         19 df_test = df_test[~df_test['Open'].isnull()]
         20
          executed in 133ms, finished 22:59:06 2023-04-17
```

```
In [120]: 1 # convert Dataframe to json
          2
          3 data = json.dumps( df_test.to_dict( orient='records' ) )
          4
          executed in 80ms, finished 22:59:08 2023-04-17
```

```
In [121]: 1 # API Call
          2
          3 #url = 'http://192.168.1.104:5000/rossmann/predict'
          4 url = 'https://teste-rossmann-prediction-api.onrender.com/rossmann/predi
          5 header = { 'Content-type': 'application/json' }
          6 data = data
          7
          8 r = requests.post( url, data=data, headers=header )
          9
         10 print( 'Status Code {}'.format( r.status_code ) )
         11
          executed in 1m 17.2s, finished 23:00:26 2023-04-17
```

Status Code 200

In [122]:

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

executed in 119ms, finished 23:01:02 2023-04-17

Out[122]:

	store	day_of_week	date	open	promo	state_holiday	school_holiday	store_sales
0	66	4	2015-09-17T00:00:00.000	1.0	1	regular_day	0	257975.43
1	190	4	2015-09-17T00:00:00.000	1.0	1	regular_day	0	221698.85
2	215	4	2015-09-17T00:00:00.000	1.0	1	regular_day	0	268233.69
3	229	4	2015-09-17T00:00:00.000	1.0	1	regular_day	0	252852.35
4	289	4	2015-09-17T00:00:00.000	1.0	1	regular_day	0	233272.34

In [123]:

```
1 d2 = d1[['store', 'prediction']].groupby( 'store' ).sum().reset_index()
2
3 for i in range( len( d2 ) ):
4     print( 'Store Number {} will sell R${:,.2f} in the next 6 weeks'.format(
5         d2.loc[i, 'store'],
6         d2.loc[i, 'prediction'] ) )
```

executed in 25ms, finished 23:01:04 2023-04-17

Store Number 66 will sell R\$257,975.43 in the next 6 weeks  
 Store Number 190 will sell R\$221,698.85 in the next 6 weeks  
 Store Number 215 will sell R\$268,233.69 in the next 6 weeks  
 Store Number 229 will sell R\$252,852.35 in the next 6 weeks  
 Store Number 289 will sell R\$233,272.34 in the next 6 weeks  
 Store Number 373 will sell R\$201,926.03 in the next 6 weeks  
 Store Number 485 will sell R\$316,660.10 in the next 6 weeks

## 12 Deploy Model to Production ( Online )

In [ ]:

```
1 # Save Trained Model
2
3 #pickle.dump( model_xgb_tuned, open( 'C:\\Users\\gabre\\DS IN PROGRESS\\
4
```

executed in 0ms, finished 06:01:43 2023-04-12

### 12.1 Rossmann Class

In [ ]:

```
1  '''import pickle
2  import inflection
3  import pandas as pd
4  import numpy as np
5  import math
6  import datetime
7
8
9  class Rossmann(object):
10     def __init__(self):
11         self.home_path = ''
12         self.competition_distance_scaler = pickle.load( open( self.home_
13         self.competition_time_month_scaler = pickle.load( open( self.hon
14         self.promo_time_week_scaler = pickle.load( open( self.home_path
15         self.year_scaler = pickle.load( open( self.home_path + 'paramete
16         self.store_type_scaler = pickle.load( open( self.home_path + 'pa
17
18     def data_cleaning(self, df1):
19         ## 1.1. Rename Columns
20         cols_old = ['Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'Stat
21                     'StoreType', 'Assortment', 'CompetitionDistance', '(
22                     'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWe
23
24         snakecase = lambda x: inflection.underscore(x)
25
26         cols_new = list(map(snakecase, cols_old))
27
28         # rename
29         df1.columns = cols_new
30
31         ## 1.3. Data Types
32         df1['date'] = pd.to_datetime(df1['date'])
33
34         ## 1.5. Fillout NA
35         # competition_distance
36         df1['competition_distance'] = df1['competition_distance'].apply(
37
38         # competition_open_since_month
39         df1['competition_open_since_month'] = df1.apply(
40             lambda x: x['date'].month if math.isnan(x['competition_open_
41             'competition_open_since_month'], axis=1)
42
43         # competition_open_since_year
44         df1['competition_open_since_year'] = df1.apply(
45             lambda x: x['date'].year if math.isnan(x['competition_open_s
46             'competition_open_since_year'], axis=1)
47
48         # promo2_since_week
49         df1['promo2_since_week'] = df1.apply(
50             lambda x: x['date'].week if math.isnan(x['promo2_since_week'
51
52         # promo2_since_year
53         df1['promo2_since_year'] = df1.apply(
54             lambda x: x['date'].year if math.isnan(x['promo2_since_year'
55
```

```
56         # promo_interval
57         month_map = {1: 'Jan', 2: 'Fev', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug', 9: 'Sep',
58                       10: 'Oct', 11: 'Nov', 12: 'Dec'}
59
60         df1['promo_interval'].fillna(0, inplace=True)
61
62         df1['month_map'] = df1['date'].dt.month.map(month_map)
63
64         df1['is_promo'] = df1[['promo_interval', 'month_map']].apply(
65             lambda x: 0 if x['promo_interval'] == 0 else 1 if x['month_map'] in month_map else 0,
66             axis=1)
67
68         ## 1.6. Change Data Types
69         # competition
70         df1['competition_open_since_month'] = df1['competition_open_since_month'].astype(int)
71         df1['competition_open_since_year'] = df1['competition_open_since_year'].astype(int)
72
73         # promo2
74         df1['promo2_since_week'] = df1['promo2_since_week'].astype(int)
75         df1['promo2_since_year'] = df1['promo2_since_year'].astype(int)
76
77         return df1
78
79     def feature_engineering(self, df2):
80         # year
81         df2['year'] = df2['date'].dt.year
82
83         # month
84         df2['month'] = df2['date'].dt.month
85
86         # day
87         df2['day'] = df2['date'].dt.day
88
89         # week of year
90         df2['week_of_year'] = df2['date'].dt.weekofyear
91
92         # year week
93         df2['year_week'] = df2['date'].dt.strftime('%Y-%W')
94
95         # competition since
96         df2['competition_since'] = df2.apply(
97             lambda x: datetime.datetime(year=x['competition_open_since_year'], month=x['competition_open_since_month'],
98                                           day=1), axis=1)
99         df2['competition_time_month'] = ((df2['date'] - df2['competition_since']).dt.days // 30).astype(int)
100
101         # promo since
102         df2['promo_since'] = df2['promo2_since_year'].astype(str) + '-' + df2['promo2_since_week'].astype(str)
103         df2['promo_since'] = df2['promo_since'].apply(
104             lambda x: datetime.datetime.strptime(x + '-1', '%Y-%W-%w')
105         )
106         df2['promo_time_week'] = ((df2['date'] - df2['promo_since']).dt.days // 7).astype(int)
107
108         # assortment
109         df2['assortment'] = df2['assortment'].apply(
110             lambda x: 'basic' if x == 'a' else 'extra' if x == 'b' else 'other'
111         )
```



```
112         # state holiday
113         df2['state_holiday'] = df2['state_holiday'].apply(lambda
114                                                         x: 'public
115
116         # 3.0. PASSO 03 - FILTRAGEM DE VARIÁVEIS
117         ## 3.1. Filtragem das Linhas
118         df2 = df2[df2['open'] != 0]
119
120         ## 3.2. Selecao das Colunas
121         cols_drop = ['open', 'promo_interval', 'month_map']
122         df2 = df2.drop(cols_drop, axis=1)
123
124         return df2
125
126     def data_preparation(self, df5):
127         ## 5.2. Rescaling
128         # competition distance
129         df5['competition_distance'] = self.competition_distance_scaler.fit_transform(
130             df5[['competition_distance']].values)
131
132         # competition time month
133         df5['competition_time_month'] = self.competition_time_month_scaler.fit_transform(
134             df5[['competition_time_month']].values)
135
136         # promo time week
137         df5['promo_time_week'] = self.promo_time_week_scaler.fit_transform(df5[['promo_time_week']].values)
138
139         # year
140         df5['year'] = self.year_scaler.fit_transform(df5[['year']].values)
141
142         ### 5.3.1. Encoding
143         # state_holiday - One Hot Encoding
144         df5 = pd.get_dummies(df5, prefix=['state_holiday'], columns=['state_holiday'])
145
146         # store_type - Label Encoding
147         df5['store_type'] = self.store_type_scaler.fit_transform(df5['store_type'].values)
148
149         # assortment - Ordinal Encoding
150         assortment_dict = {'basic': 1, 'extra': 2, 'extended': 3}
151         df5['assortment'] = df5['assortment'].map(assortment_dict)
152
153         ### 5.3.3. Nature Transformation
154         # day of week
155         df5['day_of_week_sin'] = df5['day_of_week'].apply(lambda x: np.sin(x * (2. * np.pi / 7)))
156         df5['day_of_week_cos'] = df5['day_of_week'].apply(lambda x: np.cos(x * (2. * np.pi / 7)))
157
158         # month
159         df5['month_sin'] = df5['month'].apply(lambda x: np.sin(x * (2. * np.pi / 12)))
160         df5['month_cos'] = df5['month'].apply(lambda x: np.cos(x * (2. * np.pi / 12)))
161
162         # day
163         df5['day_sin'] = df5['day'].apply(lambda x: np.sin(x * (2. * np.pi / 31)))
164         df5['day_cos'] = df5['day'].apply(lambda x: np.cos(x * (2. * np.pi / 31)))
165
166         # week of year
167         df5['week_of_year_sin'] = df5['week_of_year'].apply(lambda x: np.sin(x * (2. * np.pi / 52)))
```

```
168         df5['week_of_year_cos'] = df5['week_of_year'].apply(lambda x: np
169
170     cols_selected = ['store', 'promo', 'store_type', 'assortment', '
171                     'competition_open_since_month',
172                     'competition_open_since_year', 'promo2', 'promoc
173                     'competition_time_month', 'promo_time_week',
174                     'day_of_week_sin', 'day_of_week_cos', 'month_si
175                     'week_of_year_sin', 'week_of_year_cos']
176
177     return df5[cols_selected]
178
179     def get_prediction(self, model, original_data, test_data):
180         # prediction
181
182         pred = model.predict(test_data)
183
184         # join pred into the original data
185
186         original_data['prediction'] = np.expm1(pred)
187
188         return original_data.to_json(orient='records', date_format='iso'
189
```

executed in 0ms, finished 06:01:43 2023-04-12

## 12.2 API Handler

In [ ]:

```
1  '''import os
2  import pickle
3  import pandas as pd
4  from flask import Flask, request, Response
5  from rossmann.Rossmann import Rossmann
6
7  # Loading in web
8
9  model = pickle.load(open( 'model\\model_rossmann.pkl', 'rb'))
10
11 # initialize API
12 app = Flask(__name__)
13
14
15 @app.route('/rossmann/predict', methods=['POST'])
16 def rossmann_predict():
17     test_json = request.get_json()
18
19     if test_json: # there is data
20         if isinstance(test_json, dict): # unique example
21             test_raw = pd.DataFrame(test_json, index=[0])
22
23         else: # multiple example
24             test_raw = pd.DataFrame(test_json, columns=test_json[0].keys)
25
26         # Instantiate Rossmann class
27         pipeline = Rossmann()
28
29         # data cleaning
30         df1 = pipeline.data_cleaning(test_raw)
31
32         # feature engineering
33         df2 = pipeline.feature_engineering(df1)
34
35         # data preparation
36         df3 = pipeline.data_preparation(df2)
37
38         # prediction
39         df_response = pipeline.get_prediction(model, test_raw, df3)
40
41         return df_response
42
43     else:
44         return Reponse('{}', status=200, mimetype='application/json')
45
46 if __name__ == '__main__':
47     port = os.environ.get( 'PORT', 5000 )
48     app.run( host='192.168.1.104', port = port)'''
```

executed in 0ms, finished 06:01:43 2023-04-12

In [ ]:

executed in 1ms, finished 06:01:43 2023-04-12

In [ ]:


```
1  #pip list --format=freeze > requirements.txt
```

In [ ]: 1 `#import sys`

executed in 0ms, finished 06:01:43 2023-04-12

In [85]:

executed in 42ms, finished 06:02:38 2023-04-12

 `NameError: name 'data' is not defined ▶`

In [ ]: