None

1. Introduction

Rossmann, brand that operates over 3,000 drug stores in 7 European countries, want to predict sales in the future based on historical data from January 2013 to July 2015.

1.1. Goal

- Explore the data and perform correlational analysis between sales and many related features of stores and time series.
- Build sales prediction model using Linear Regression

1.2. Data description

Store data

- *StoreType* - differentiates between 4 different store models: a, b, c, d
- *Assortment* - describes an assortment level: a = basic, b = extra, c = extended
- *CompetitionDistance* - distance in meters to the nearest competitor store
- *CompetitionOpenSince[Month/Year]* - gives the approximate year and month of the time the nearest competitor was opened
- *Promo* - indicates whether a store is running a promo on that day
- *Promo2* - Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating
- *Promo2Since[Year/Week]* - describes the year and calendar week when the store started participating in Promo2
- *PromoInterval* - describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

Sales data

- *Id* - an Id that represents a (Store, Date) duple within the test set
- *Store* - a unique Id for each store
- *Sales* (Target) - the turnover for any given day (this is what you are predicting)
- *Customers* - the number of customers on a given day
- *Open* - an indicator for whether the store was open: 0 = closed, 1 = open
- *StateHoliday* - indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
- *SchoolHoliday* - indicates if the (Store, Date) was affected by the closure of public schools

2. Data processing

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import datetime

# ignore warnings
import warnings
warnings.filterwarnings("ignore")

# for visualization
import seaborn as sns
import matplotlib.pyplot as plt

#set default parameters of charts
plt.style.use('seaborn')
plt.rcParams['figure.figsize'] = [20, 12]
plt.rc('font', size=15)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=15)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=12)    # fontsize of the tick labels
plt.rc('ytick', labelsize=12)    # fontsize of the tick labels
plt.rc('legend', fontsize=15)    # legend fontsize

pd.set_option('display.float_format', '{:.5f}'.format)
```

In [2]:
```
# import data

df_sales = pd.read_csv("Sales_data.csv")
df_store = pd.read_csv("Store_data.csv")
#testdata
```

In [3]:
```
print("Number of rows of Sales data: {}, number of columns of Sales data: {}".format(df_sales.shape[0],df_
print("Number of rows of Store data: {}, number of columns of Store data: {}".format(df_store.shape[0],df_
```

Number of rows of Sales data: 91256, number of columns of Sales da
ta: 9
Number of rows of Store data: 100, number of columns of Store data
: 10

In [4]:
```
df_sales.head(5)
```

Out[4]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	7/31/2015	5263	555	1	1	0	1
1	2	5	7/31/2015	6064	625	1	1	0	1
2	3	5	7/31/2015	8314	821	1	1	0	1
3	4	5	7/31/2015	13995	1498	1	1	0	1
4	5	5	7/31/2015	4822	559	1	1	0	1

In [5]:
```
df_store.head(5)
```

Out[5]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinc
0	1	c	a	1270	9.00000	2008.C
1	2	a	a	570	11.00000	2007.C
2	3	a	a	14130	12.00000	2006.C
3	4	c	c	620	9.00000	2009.C
4	5	a	a	29910	4.00000	2015.C

In [6]:
```
# Merge data
df = pd.merge(df_sales, df_store, how="left", on="Store")
df.sample()
```

Out[6]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	Store
12802	3	3	3/25/2015	5538	679	1	0	0	0	

In [7]:
```
df.describe(include='all')
```

Out[7]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHo
count	91256.00000	91256.00000	91256	91256.00000	91256.00000	91256.00000	91256.00000	(
unique	NaN	NaN	942	NaN	NaN	NaN	NaN	
top	NaN	NaN	7/31/2015	NaN	NaN	NaN	NaN	
freq	NaN	NaN	100	NaN	NaN	NaN	NaN	ε
mean	50.33870	3.99833	NaN	5545.36234	579.65725	0.82841	0.38152	
std	28.90536	1.99740	NaN	3466.08156	362.83595	0.37703	0.48576	
min	1.00000	1.00000	NaN	0.00000	0.00000	0.00000	0.00000	
25%	25.00000	2.00000	NaN	3755.00000	406.00000	1.00000	0.00000	
50%	50.00000	4.00000	NaN	5686.00000	587.00000	1.00000	0.00000	
75%	75.00000	6.00000	NaN	7623.00000	786.00000	1.00000	1.00000	
max	100.00000	7.00000	NaN	38037.00000	2849.00000	1.00000	1.00000	

Duplicate values

In [8]:
```
# check duplicate
df[df.duplicated()].shape[0]
```

Out[8]:

The dataset doesn't have duplicate values.

Missing values

In [9]:
```
pd.concat([pd.DataFrame(df.dtypes, columns=["Type"]),
           pd.DataFrame(df.isna().sum(), columns=["NA"]),
           pd.DataFrame(df.isna().mean(), columns=["%NA"])], axis=1)
```

Out[9]:

	Type	NA	%NA
Store	int64	0	0.00000
DayOfWeek	int64	0	0.00000
Date	object	0	0.00000
Sales	int64	0	0.00000
Customers	int64	0	0.00000
Open	int64	0	0.00000
Promo	int64	0	0.00000
StateHoliday	object	0	0.00000
SchoolHoliday	int64	0	0.00000
StoreType	object	0	0.00000
Assortment	object	0	0.00000
CompetitionDistance	int64	0	0.00000
CompetitionOpenSinceMonth	float64	25456	0.27895
CompetitionOpenSinceYear	float64	25456	0.27895
Promo2	int64	0	0.00000
Promo2SinceWeek	float64	42964	0.47081
Promo2SinceYear	float64	42964	0.47081
PromoInterval	object	42964	0.47081

Since NA values all come from Store data (*CompetitionOpenSinceMonth*, *CompetitionOpenSinceYear*, *Promo2SinceWeek*, *Promo2SinceYear*, *PromoInterval*), the fillout NA process will be done with Store data seperately and merged again after that.

In [14]:

There is no big difference between mean and median, so NA values will be replaced by mean.

In [15]:
```
# replace NA with mean
df_store['CompetitionOpenSinceMonth'].fillna(df_store['CompetitionOpenSinceMonth'].mean(), inplace = True
df_store['CompetitionOpenSinceYear'].fillna(df_store['CompetitionOpenSinceYear'].mean(), inplace = True)
```

In [16]:
```
# Promo2SinceWeek, Promo2SinceYear & PromoInterval NA
df_store[pd.isnull(df_store.Promo2SinceWeek)].sample(10)
```

Out[16]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSin
48	49	d	c	18010	9.00000	2007
87	88	a	a	10690	10.00000	2005
33	34	c	a	2240	9.00000	2009
54	55	a	a	720	11.00000	2004
36	37	c	a	4230	12.00000	2014
43	44	a	a	540	6.00000	2011
49	50	d	a	6260	11.00000	2009
22	23	d	a	4060	8.00000	2005
59	60	d	c	5540	10.00000	2009
9	10	a	a	3160	9.00000	2009

In [17]:
```
df_store['Promo2'][pd.isnull(df_store.Promo2SinceWeek)].sum()
```

Out[17]:

All the missing values comes from fields where Promo2 = 0 which also means that there are no continuous promotional activities for those stores. That leads to all the next columns will truely be all '0' values.

In [18]:
```
# replace NA with 0
```

```
df_store.Promo2SinceWeek.fillna(0,inplace=True)
df_store.Promo2SinceYear.fillna(0,inplace=True)
df_store.PromoInterval.fillna(0,inplace=True)
```

In [19]:
```
# Change Data Types
df_sales["Date"] = pd.to_datetime(df_sales["Date"])

df_store["CompetitionOpenSinceMonth"] = df_store["CompetitionOpenSinceMonth"].astype(int)
df_store["CompetitionOpenSinceYear"] = df_store["CompetitionOpenSinceYear"].astype(int)

df_store["Promo2SinceWeek"] = df_store["Promo2SinceWeek"].astype(int)
df_store["Promo2SinceYear"] = df_store["Promo2SinceYear"].astype(int)
```

In [20]:
```
# merge data again
df = pd.merge(df_sales, df_store, how="left", on="Store")
df.sample()
```

Out[20]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreTyp
14452	53	1	2015-03-09	5886	554	1	0	0	0	a

In [21]:
```
pd.concat([pd.DataFrame(df.dtypes, columns=["Type"]),
           pd.DataFrame(df.isna().sum(), columns=["NA"])], axis=1)
```

Out[21]:

	Type	NA
Store	int64	0
DayOfWeek	int64	0
Date	datetime64[ns]	0
Sales	int64	0
Customers	int64	0
Open	int64	0
Promo	int64	0
StateHoliday	object	0
SchoolHoliday	int64	0
StoreType	object	0
Assortment	object	0
CompetitionDistance	int64	0
CompetitionOpenSinceMonth	int32	0
CompetitionOpenSinceYear	int32	0
Promo2	int64	0
Promo2SinceWeek	int32	0
Promo2SinceYear	int32	0
PromoInterval	object	0

Feature Engineering

In [22]:
```
df["Year"] = df["Date"].dt.year # year
df["Month"] = df["Date"].dt.month # month
df["Day"] = df["Date"].dt.day # day
df["WeekOfYear"] = df["Date"].dt.isocalendar().week # week of year
```

In [23]:
```
# caculate the length of time since promo2 has first started in each store each day

df["Promo2Since"] = df["Promo2SinceYear"].astype(str) + "-" + df["Promo2SinceWeek"].astype(str)
df["Promo2Since"] = df[(df.Promo2 == 1)]["Promo2Since"].apply(lambda x: datetime.datetime.strptime(x + "-
df['Promo2SinceTime'] = ((df[(df.Promo2 == 1)]["Date"] - df[(df.Promo2 == 1)]["Promo2Since"]) / 7).apply(
df['Promo2SinceTime'].loc[df['Promo2SinceTime'] < 0] = 0
df.Promo2SinceTime.fillna(0,inplace=True)
```

In [24]:
```
# caculate the length of time since nearest competitor has first opend in each store each day
df["CompetitionSince"] = df.apply(lambda x: datetime.datetime(year=x["CompetitionOpenSinceYear"],
                                                               month=x["CompetitionOpenSinceMonth"],
                                                               day=1), axis=1)
df["CompetitionOpenSinceTime"] = ((df["Date"] - df["CompetitionSince"]) / 30).apply(lambda x: x.days).ast

# df['CompetitionOpenSinceLengthOfMonth'] = pd.Series()
# for i, v in df['CompetitionOpenSinceYear'].items():
#     if v == df.Year[i]:
#         if df['CompetitionOpenSinceMonth'][i] <= df['Month'][i]:
#             df['CompetitionOpenSinceLengthOfMonth'].loc[i] = df['Month'][i] - df['CompetitionOpenSinceM
#         else: df['CompetitionOpenSinceLengthOfMonth'].loc[i] = 0
#     if v < df.Year[i]:
#         if df['CompetitionOpenSinceMonth'][i] <= df['Month'][i]:
#             df['CompetitionOpenSinceLengthOfMonth'].loc[i] = df['Month'][i] - df['CompetitionOpenSinceM
#         else:
#             df['CompetitionOpenSinceLengthOfMonth'].loc[i] = df['Month'][i] - df['CompetitionOpenSinceM
```

New variables for analysis:

In [25]:
```
df.iloc[:,18:28].sample(5)
```

Out[25]:

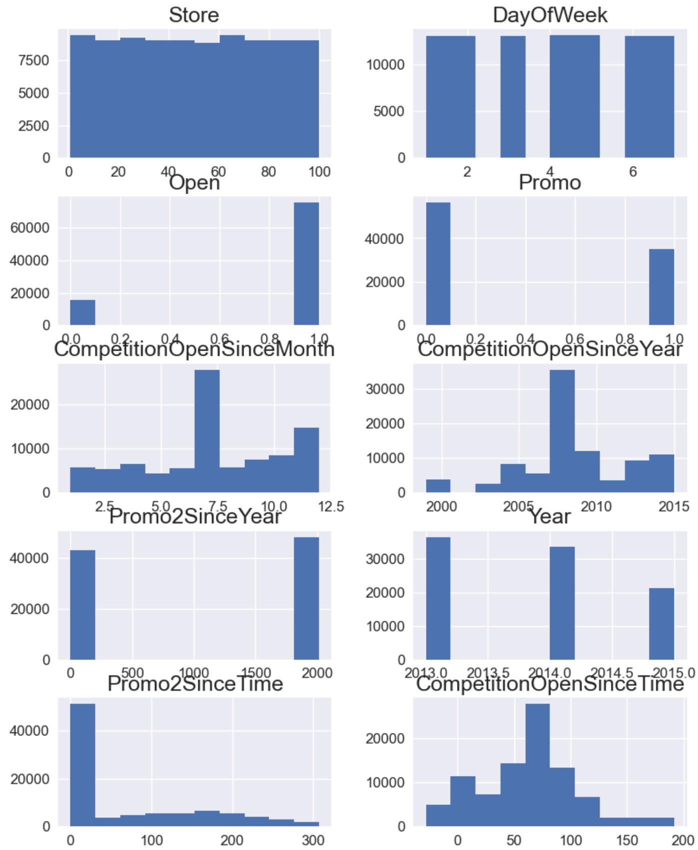
	Year	Month	Day	WeekOfYear	Promo2Since	Promo2SinceTime	CompetitionSince	CompetitionO
78243	2013	5	11	19	NaT	0.00000	2005-10-01	
19624	2015	1	16	3	NaT	0.00000	2003-04-01	
40305	2014	5	25	21	NaT	0.00000	2009-11-01	
87638	2013	2	6	6	NaT	0.00000	2008-07-01	
17744	2015	2	4	6	NaT	0.00000	2014-02-01	

In [26]:
```
# drop redundant column and create new dataset for EDA
dfa = df.drop(['Promo2Since', 'CompetitionSince', 'WeekOfYear'], axis=1)
```

3. Exploratory Data Analysis

In [27]:
```
df_num = dfa.select_dtypes(exclude=["object", "datetime64[ns]"])
df_cat = dfa.select_dtypes(include=["object"])
```

In [28]:
```
df_num.hist();
```



In [29]:
```
df_cat.apply(lambda x: x.unique())
```

Out[29]:

StateHoliday [0, a, b, c]
StoreType [c, a, d, b]
Assortment [a, c]
PromoInterval [0, Jan,Apr,Jul,Oct, Feb,May,Aug,Nov, Mar,Jun,...
dtype: object

Open, *SchoolHoliday* & *Promo2* are binary variables with 0 = no & 1 = yes.

Assortment only has 2 type a & c.

PromoInterval are intermittent period: one every 3 months.

In [30]:
```
print("Rossman stores earned nothing in {} days during the period, and there are {} days the stores had c
```

Rossman stores earned nothing in 15665.0 days during the period, and there are 15659.0 days the stores had closed.

The time stores didn't generate sales revenue was due to they didn't open that days, and the reason for closing are holidays or refurbishment, so it's no need to be analyzed and will be dropped out of the dataset.

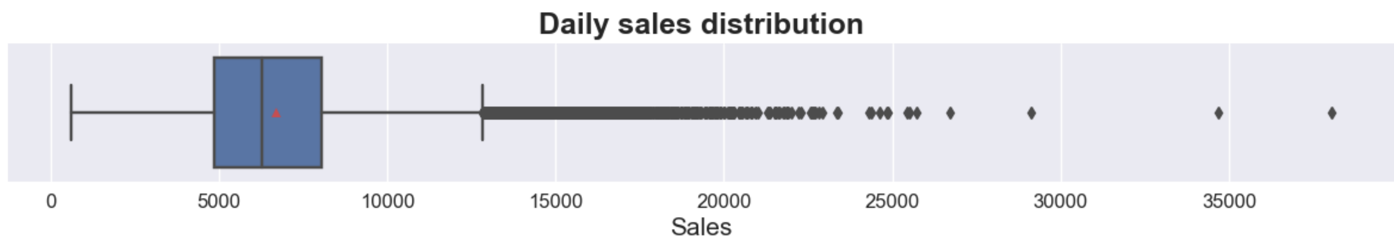
In [31]:
```
# drop Sales = 0 & Open Column
dfa=dfa.drop(dfa[(dfa.Sales == 0)].index)
dfa=dfa.drop('Open', axis = 1)
dfa = dfa.reset_index(drop=True) # reset index

print("New dataset now has {} rows.".format(dfa.shape[0]))
```

New dataset now has 75591 rows.

3.1. Sales

In [32]:



In [33]:
```
dfa["Sales"].describe()
```

Out[33]:

```
count    75591.00000
mean      6694.54811
std       2609.60164
min        612.00000
25%       4865.00000
50%       6282.00000
75%       8058.00000
max      38037.00000
Name: Sales, dtype: flo
at64
```

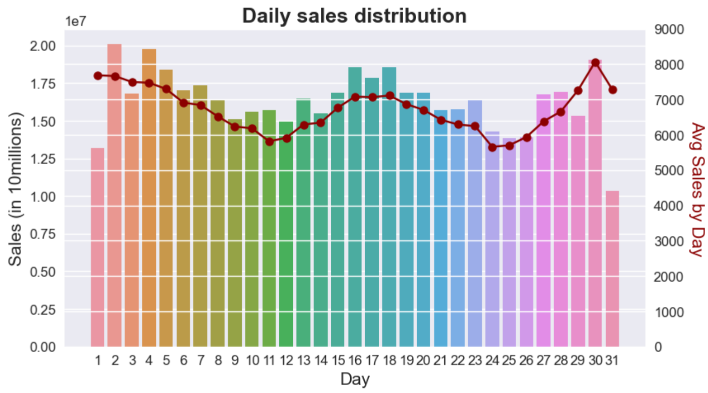
The sales distribution has many outliners, varing from 612 to 38037.

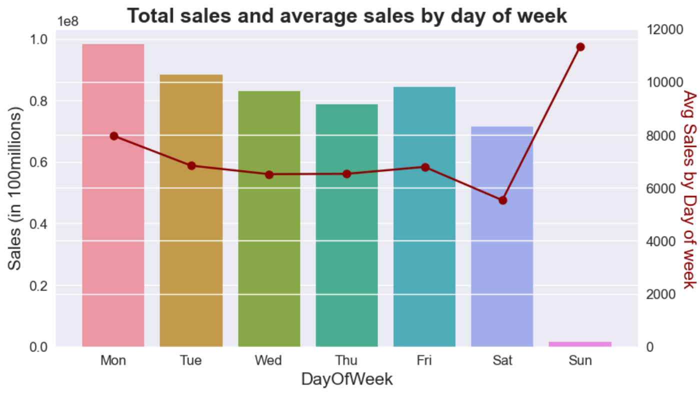
In [34]:

Overall, the total sales exhibited an upward trend, reaching their highest values in all months during 2015. However, the monthly sales in 2014 were inconsistent and often fell below the figures recorded of that in the last year.

Despite, over the course of the three-year period, there was no ditermined seasonal trend in sales. However, sales tend to ramain high in January and March in the first quater. Lastly, significant revenue can be expected during the final months of the year, especially December.

In [35]:





In terms of total sales and average sales, high sales figures were consistently observed at the beginning, middle, and end of each month.

While there was no apparent difference in sales between the days of the week, it was notable that Sundays, which were typically closed, recorded the biggest average sales figures despite not having special promotions or events.

In [36]:

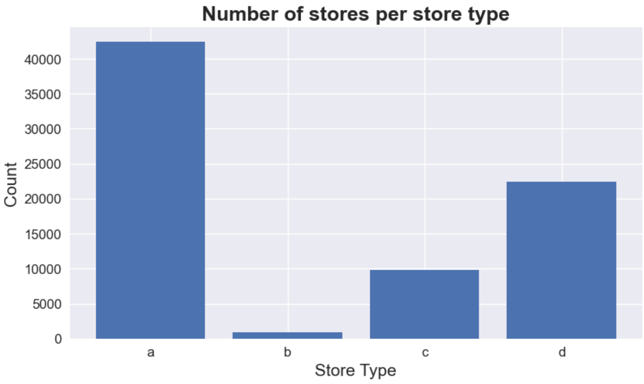
```
# check whether Sundays have promo or special events or not
dfa[['StateHoliday', 'SchoolHoliday', 'Promo', 'Promo2']][(dfa.DayOfWeek == 7)].apply(lambda x: x.unique(
```

Out[36]:

	StateHoliday	SchoolHoliday	Promo	Promo2
0	0	0	0	0

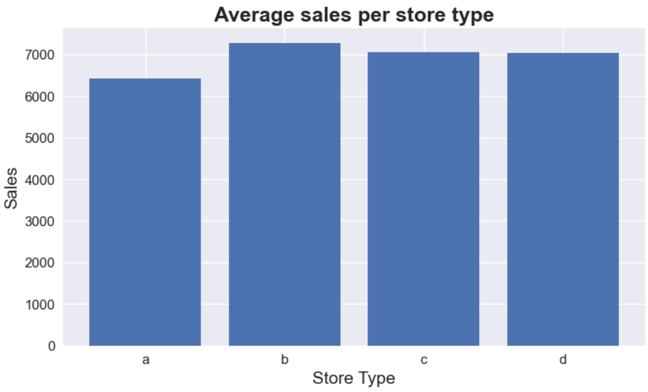
3.2. Sales vs Store type vs Customers

In [37]:



It can be seen that most of the Rossman stores are type *a*. Number of stores are directly proportional to number of customer in all store type.

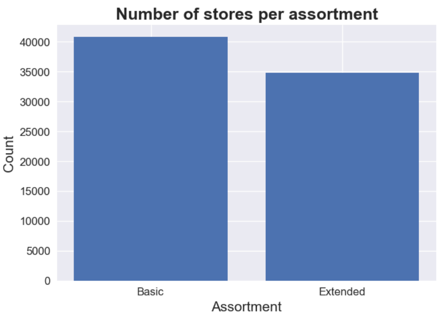
In [38]:

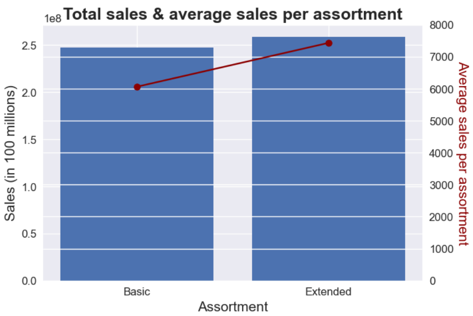


Averages sales in 4 types of stores were not much different, still, store type a had the smallest figures the rest of the store types. Store type b had the highest average customers but the lowest average spending, in contrast with store type d.

3.2. Sales vs Assortment vs Store Type

In [39]:





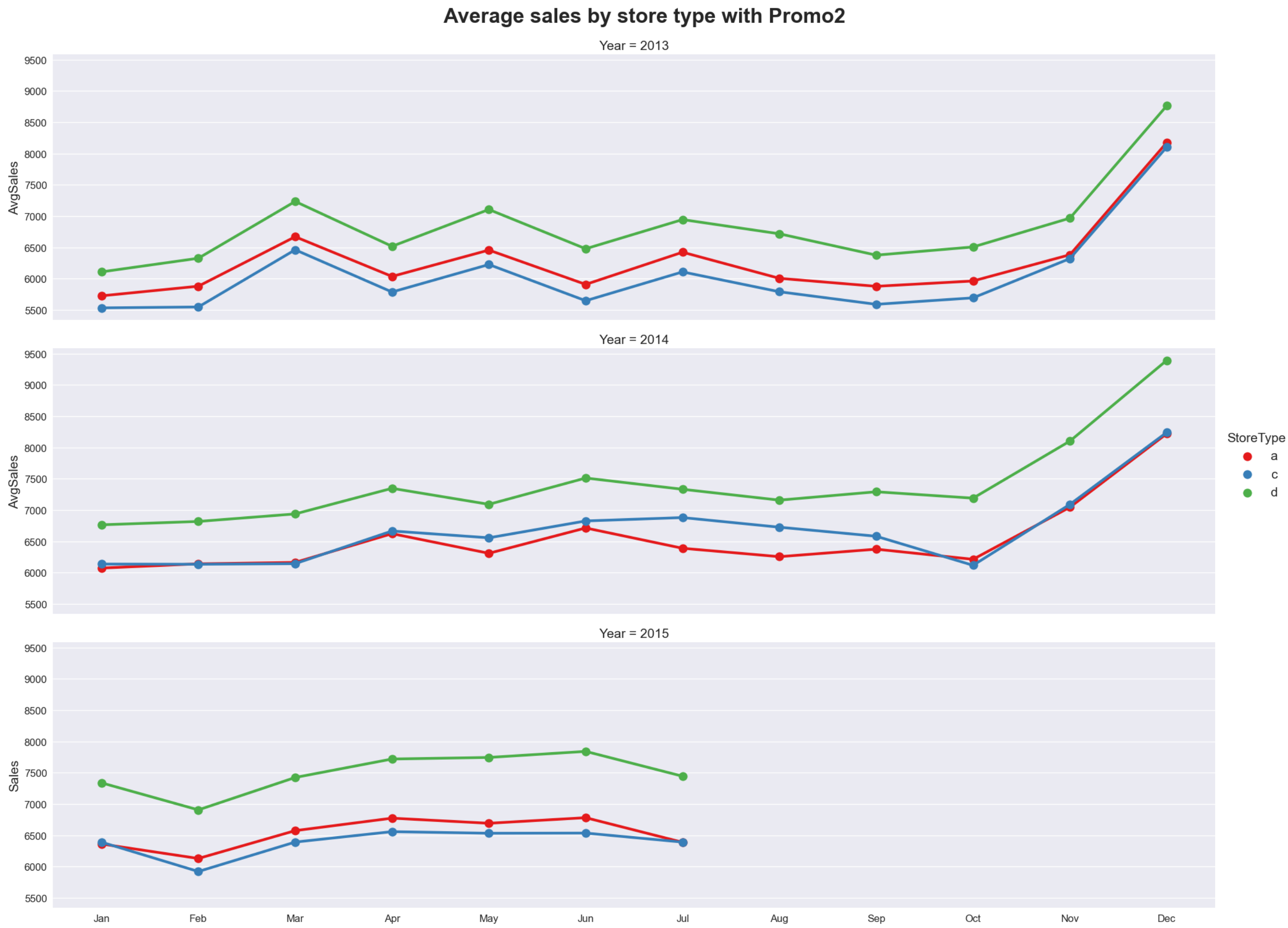
Despite being fewer in number, the extended assortment stores accounted for higher figures in both total and average sales. This helps to explain why store type *a*, which primarily used basic assortments, did not generate as much revenue on average compared to other store types, especially store type *d*, greater included extended assortments.

3.3. Sales vs Promotion

In [40]:

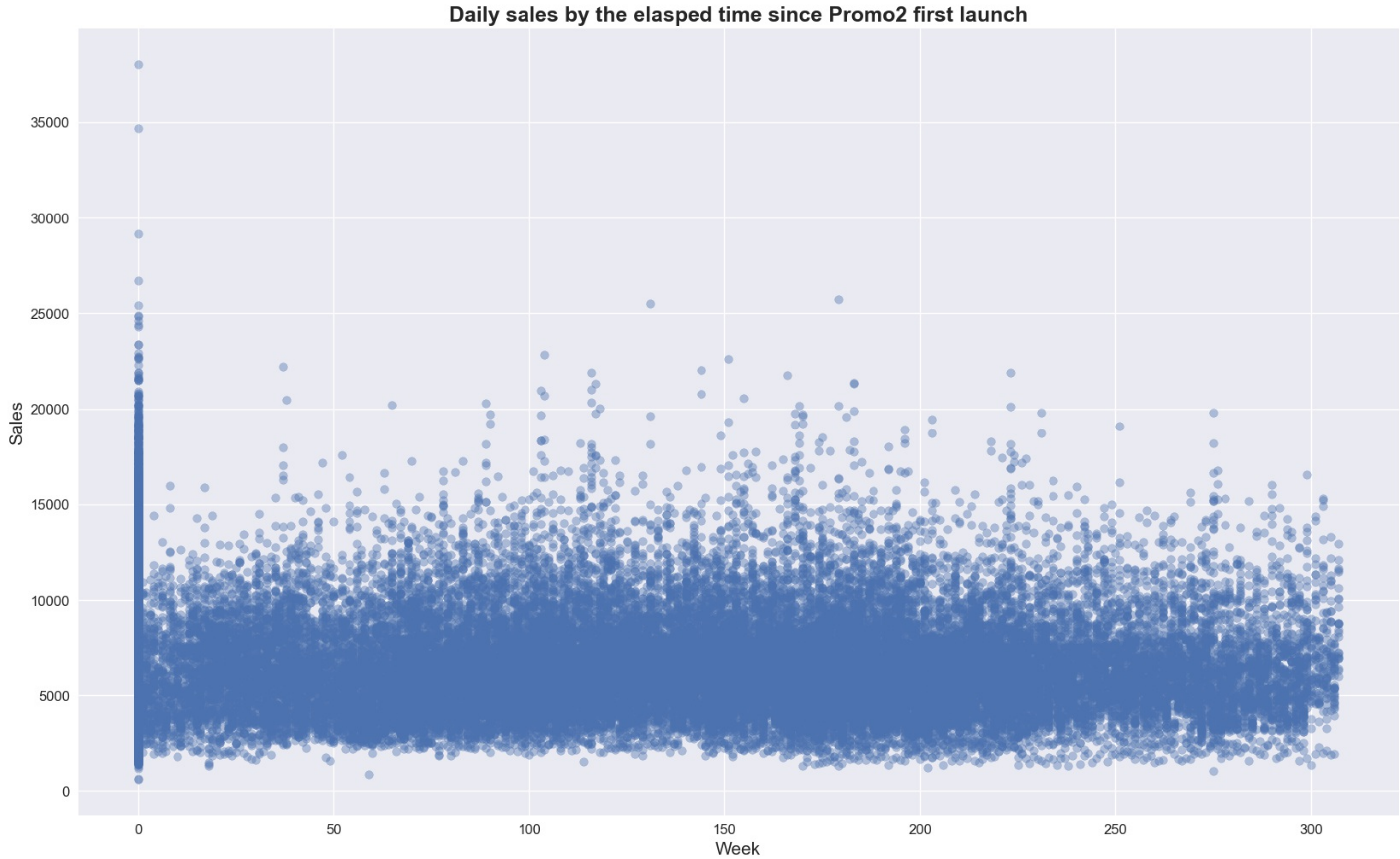
In general, stores that run individual promotions tend to have higher sales figures. Furthermore, co-operative promotional campaigns between stores (Promo2) also lead to increased sales, except for the last six months of 2014.

In [41]:



Co-operative promotional campaigns (Promo2) had a similar pattern across all store types, except for store type *b*, which did not participate. However, despite the standard impact across most store types, store type *a* experienced significantly higher sales.

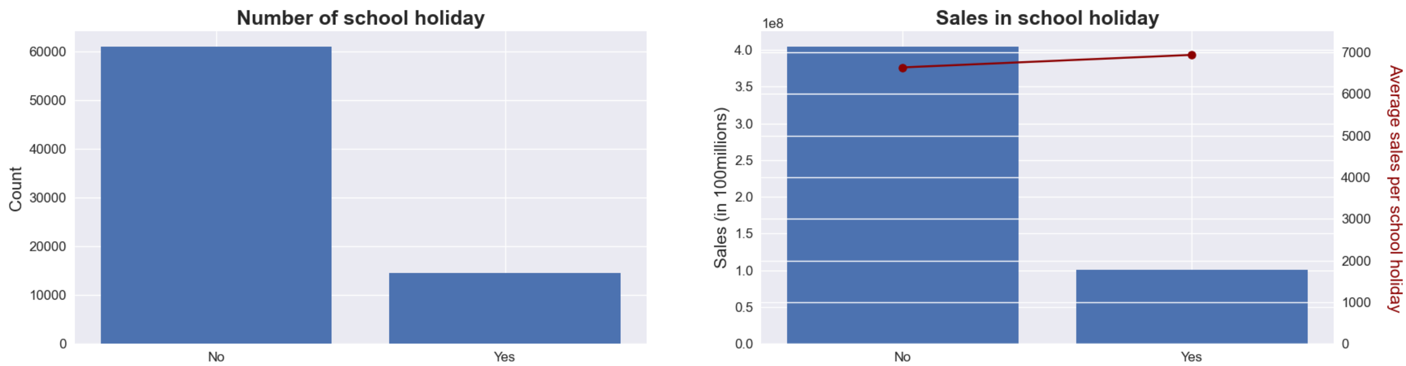
In [42]:



No significant correlation exists between the passage of time since the initial launch of Promo2 and the daily sales figures.

3.4. Sales vs Holiday

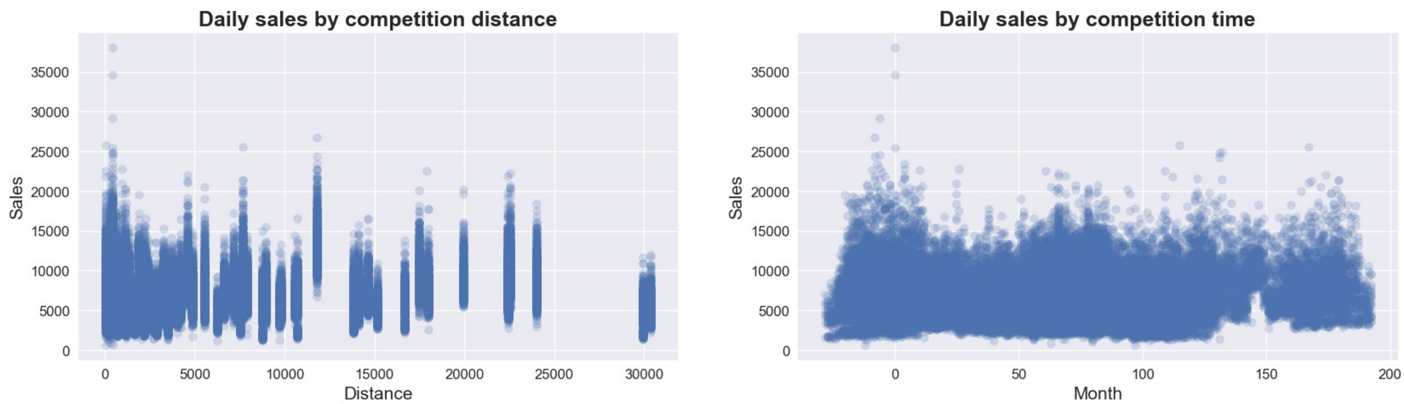
In [43]:



According to the graphs, the closure of public schools did not have a significant impact on the sales figures.

3.5. Sales vs Competition

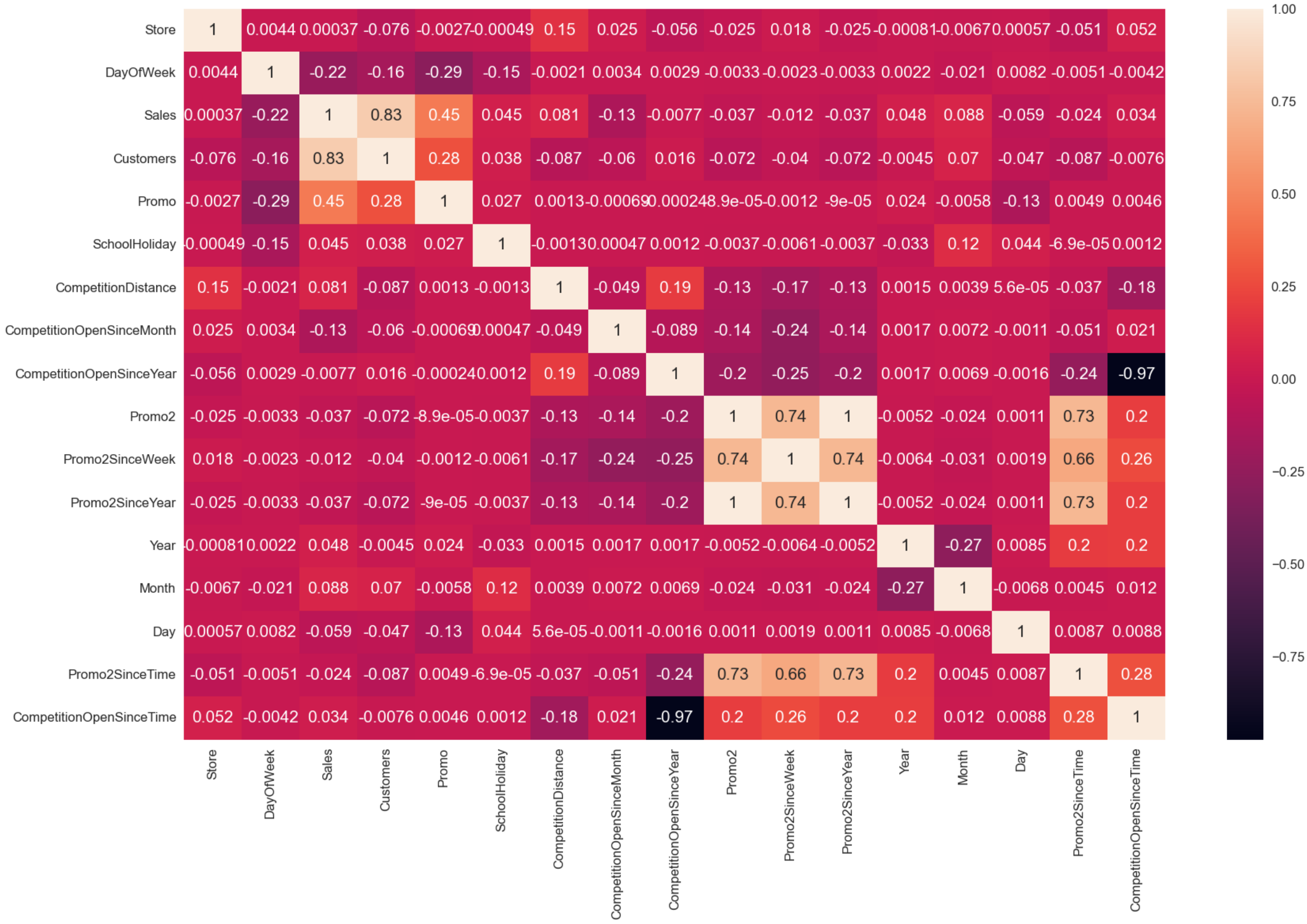
In [44]:



There is also no significant correlation between the distance or their length of existence of the nearest competitor and the daily sales figures.

3.6. Correlation matrix

In [45]:



Findings

- Sales figures showed an upward trends after three year.
- Customers tend to shop at the beginning, middle, and end of each month.
- Sundays have the potential to generate more sales than weekdays.
- Store *d* had the highest spending per customer compared to other stores.
- Stores with extended assortments generally have higher sales figures than basic stores.
- Running promotional campaigns increases revenue, with higher revenue generated when running both Promo and Promo2.
- Among all store types, Promo2 had the most significant impact on Store D.

4. Sales prediction with Linear Regression model

The linear regression model has the form below, with X is the input value of all variables and predicting a output Y. beta0 is the intercept and beta-j is the slope coefficient.

In [65]:

Out[65]:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j.$$

The method of the model is *least squares*, which minimizes the residual sum of squares:

In [67]:

Out[67]:

$$\begin{aligned} \mathrm{RSS}(\beta) &= \sum_{i=1}^{N} (y_i - f(x_i))^2 \\ &= \sum_{i=1}^{N} \Big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \Big)^2. \end{aligned}$$

The unique solution for that equation is given by:

```
Image("LR3.png")
```

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

4.1. Data encoding

In [46]:
```
dfp = dfa.copy()
```
In [47]:
```
# categorical variables encoding

dfp = pd.get_dummies(dfp, prefix=["StoreType"], columns=["StoreType"]) # one hot encoding

dfp = pd.get_dummies(dfp, prefix=["StateHoliday"], columns=["StateHoliday"], drop_first=True) # dummy enc

# ordinal Enconding
dfp["Assortment"].loc[dfp["Assortment"] == 'a'] = 1
dfp["Assortment"].loc[dfp["Assortment"] == 'c'] = 2
```
In [48]:
```
# time encoding
# day of week
dfp['DayOfWeekSin'] = dfp['DayOfWeek'].apply(lambda x: np.sin( x * (2. * np.pi/7)))
dfp['DayOfWeekCos'] = dfp['DayOfWeek'].apply(lambda x: np.cos( x * (2. * np.pi/7)))

# month
dfp['MonthSin'] = dfp['Month'].apply(lambda x: np.sin(x * (2. * np.pi/12)))
dfp['MonthCos'] = dfp['Month'].apply(lambda x: np.cos(x * (2. * np.pi/12)))

# day
dfp['DaySin'] = dfp['Day'].apply(lambda x: np.sin(x * (2. * np.pi/30)))
dfp['DayCos'] = dfp['Day'].apply(lambda x: np.cos(x * (2. * np.pi/30)))
```
In [49]:
```
dfp = dfp.drop(['DayOfWeek', 'Day', 'Month', 'Customers', 'Date', 'PromoInterval'], axis=1 )
```
In [50]:
```
dfp.sample(5)
```
Out[50]:

	Store	Sales	Promo	SchoolHoliday	Assortment	CompetitionDistance	CompetitionOpenSinceMont
53826	82	6979	0	0	1	22390	4
72861	76	7694	0	0	2	19960	3
45154	48	3825	0	1	1	1060	5
44047	11	6491	0	0	2	960	1
65650	75	7700	1	0	2	22440	12

5 rows × 27 columns

4.2. Feature scaling

In [51]:
```
# for Independent Variables
# normalization
max1 = dfp['CompetitionDistance'].max()
min1 = dfp['CompetitionDistance'].min()
dfp['CompetitionDistance'] = dfp['CompetitionDistance'].apply(lambda x: (x-min1)/(max1-min1))
# standardization
mean1 = dfp['CompetitionOpenSinceTime'].mean()
std1 = dfp['CompetitionOpenSinceTime'].std()
dfp['CompetitionOpenSinceTime'] = dfp['CompetitionOpenSinceTime'].apply(lambda x: (x-mean1)/std1)
mean2 = dfp['Promo2SinceTime'].mean()
std2 = dfp['Promo2SinceTime'].std()
dfp['Promo2SinceTime'] = dfp['Promo2SinceTime'].apply(lambda x: (x-mean2)/std2)
```
In [52]:
```
# for Dependent Variables
dfp["Sales"] = np.log1p(dfp["Sales"])
```

4.3. Data modeling

In [53]:
```
# drop Sales column
dfp1 = dfp.drop(['Sales'], axis = 1)
```

In [54]:
```
# train set & test set
x_train, x_test, y_train, y_test = train_test_split(dfp1.values,
                                                    dfp['Sales'].values,
                                                    test_size = 0.3, random_state=1)

x_train = x_train.astype(np.float32)
y_train = y_train.astype(np.float32)
```

In [55]:
```
# calculate errors
def model_error(time, y, yhat):

    # mean absolute error
    mae = np.mean(abs(y - yhat))
    # mean absolute percentage error
    mape = np.mean(np.abs((y - yhat) / y))
    # root mean square error
    rmse = np.sqrt(np.mean((y - yhat)**2))

    return pd.DataFrame({"MAE": mae,
                         "MAPE": mape,
                         "RMSE": rmse}, index=[time])
```

In [56]:
```
# calculate Z score for feature selection
def model_sta(df, coef, x, y, yhat):

    xbar = np.concatenate((np.ones((x.shape[0], 1)), x), axis = 1)
    n = len(y) # obs
    k = len(df.columns)

    # term
    term = df.columns.to_list()
    term.insert(0, 'Intercept')

    # variance σ2
    var = np.sum((y - yhat)**2)/(n-k-1)

    # standard errors
    sd = np.sqrt(var*(np.linalg.pinv(np.dot(xbar.T,xbar).astype('float64')).diagonal()))

    # Z score
    z = coef/sd

    return(pd.DataFrame({"Independant Variables": term, "Coefficients": coef,
                        "Standard Errors": sd, "Z score": z}))
```

In [57]:
```
# Linear Regression model
def linear_rgr_train(xtrain, ytrain):

    # add 1 in the first position of each row of x_train vector
    one = np.ones((xtrain.shape[0], 1))
    Xbar = np.concatenate((one, xtrain), axis = 1)

    # calculate solution of least squares estimates & residual
    coef = np.dot(np.linalg.pinv(np.dot(Xbar.T, Xbar)), np.dot(Xbar.T, ytrain))

    return coef

def linear_rgr_test(coef, xtest):

    # add 1 in the first position of each row of x_test vector
    Xtestbar = np.concatenate((np.ones((xtest.shape[0], 1)), xtest), axis = 1)

    # apply model to test set
    rgr_result = np.dot(coef, Xtestbar.T)

    return rgr_result
```

In [58]:
```
#
coef1 = linear_rgr_train(x_train, y_train)
y_pred1 = linear_rgr_test(coef1, x_test).astype(np.float32)
```

In [59]:
```
model_error("1st", np.expm1(y_test), np.expm1(y_pred1))
```

Out[59]:

	MAE	MAPE	RMSE
1st	1563.96982	0.24979	2105.26150

Linear Regression model has MAE = 1563.96982, MAPE = 0.24979, RMSE = 2105.26150

Variable subset selection

In [60]:
```
model_sta(dfp1, coef1, x_test, y_test, y_pred1)
```

Out[60]:

	Independant Variables	Coefficients	Standard Errors	Z score
0	Intercept	-4.07128	5.99727	-0.67886
1	Store	-0.00029	0.00007	-4.01586
2	Promo	0.36680	0.00430	85.28958
3	SchoolHoliday	0.04939	0.00537	9.20056
4	Assortment	0.21339	0.00446	47.86323
5	CompetitionDistance	0.07414	0.00977	7.58680
6	CompetitionOpenSinceMonth	0.00885	0.00128	6.93133
7	CompetitionOpenSinceYear	0.30784	0.01208	25.47929
8	Promo2	-98.24918	10.45119	-9.40077
9	Promo2SinceWeek	0.00312	0.00024	13.17354
10	Promo2SinceYear	0.04873	0.00519	9.39188
11	Year	-0.30045	0.01184	-25.36743
12	Promo2SinceTime	0.07651	0.00922	8.30314
13	CompetitionOpenSinceTime	1.08523	0.04222	25.70411
14	StoreType_a	-0.95113	1.62902	-0.58387
15	StoreType_b	-0.64704	1.62961	-0.39705
16	StoreType_c	-0.84387	1.62900	-0.51803
17	StoreType_d	-0.90282	1.62899	-0.55422
18	StateHoliday_a	0.02366	0.08175	0.28945
19	StateHoliday_b	0.45495	0.21672	2.09927
20	StateHoliday_c	0.13007	0.30595	0.42515
21	DayOfWeekSin	0.03032	0.00273	11.10270
22	DayOfWeekCos	0.05951	0.00338	17.61640
23	MonthSin	0.10517	0.00482	21.80743
24	MonthCos	0.00315	0.00295	1.06693
25	DaySin	0.02730	0.00288	9.47484
26	DayCos	0.02757	0.00288	9.56380

Based on the Z-score analysis, the linear model includes several variables with insignificant values (Z < 2 for significance at the 5% level, or p < 0.05), such as *Store (ID), Promo2, Year, StoreType, StateHoliday* except for *Easter*. These terms can be deleted for constructing a simpler model, particularly if there are many missing values or the cost of collecting the data is high. This can be done without significantly reducing the overall accuracy of the model.

Below is the example of subset selection:

In [61]:
```
# 2nd time running
dfp2 = dfp1.drop(['Store', 'Promo2', 'Year',
```

```
                     'StoreType_a', 'StoreType_b', 'StoreType_c','StoreType_d',
                     'StateHoliday_a', 'StateHoliday_c', 'MonthCos'], axis = 1)
x_train2 = np.delete(x_train, [0, 7, 10, 13, 14, 15, 16, 17, 19, 23], 1).astype(np.float32)
x_test2 = np.delete(x_test, [0, 7, 10, 13, 14, 15, 16, 17, 19, 23], 1).astype(np.float32)

coef2 = linear_rgr_train(x_train2, y_train)
y_pred2 = linear_rgr_test(coef2, x_test2).astype(np.float32)

model_sta(dfp2, coef2, x_test2, y_test, y_pred2)
```

Out[61]:

	Independant Variables	Coefficients	Standard Errors	Z score
0	Intercept	-62.29659	5.97785	-10.42123
1	Promo	0.36480	0.00440	82.92216
2	SchoolHoliday	0.04046	0.00545	7.42579
3	Assortment	0.22339	0.00442	50.52171
4	CompetitionDistance	0.04160	0.00945	4.40080
5	CompetitionOpenSinceMonth	-0.01220	0.00083	-14.66189
6	CompetitionOpenSinceYear	0.03518	0.00297	11.82618
7	Promo2SinceWeek	0.00252	0.00022	11.67664
8	Promo2SinceYear	-0.00006	0.00000	-16.49922
9	Promo2SinceTime	0.00108	0.00345	0.31420
10	CompetitionOpenSinceTime	0.13010	0.01025	12.69519
11	StateHoliday_b	0.72923	0.22103	3.29922
12	DayOfWeekSin	0.03302	0.00279	11.82926
13	DayOfWeekCos	0.06118	0.00345	17.71621
14	MonthSin	0.01116	0.00307	3.63539
15	DaySin	0.02256	0.00294	7.66357
16	DayCos	0.02869	0.00295	9.72860

For the second running, model still has 3 insignificant variables, which are *CompetitionOpenSinceMonth*, *Promo2SinceYear*, *Promo2SinceTime*.

In [62]:

```
# 3rd time running
dfp3 = dfp2.drop(['CompetitionOpenSinceMonth', 'Promo2SinceYear', 'Promo2SinceTime'], axis = 1)
x_train3 = np.delete(x_train2, [4, 7, 8], 1).astype(np.float32)
x_test3 = np.delete(x_test2, [4, 7, 8], 1).astype(np.float32)

coef3 = linear_rgr_train(x_train3, y_train)
y_pred3 = linear_rgr_test(coef3, x_test3).astype(np.float32)
model_sta(dfp3, coef3, x_test3, y_test, y_pred3)
```

Out[62]:

	Independant Variables	Coefficients	Standard Errors	Z score
0	Intercept	-92.38733	5.45710	-16.92974
1	Promo	0.36472	0.00446	81.85092
2	SchoolHoliday	0.04011	0.00552	7.26894
3	Assortment	0.20998	0.00430	48.84645
4	CompetitionDistance	0.05593	0.00941	5.94147
5	CompetitionOpenSinceYear	0.05011	0.00272	18.44128
6	Promo2SinceWeek	0.00030	0.00014	2.14703
7	CompetitionOpenSinceTime	0.17691	0.00934	18.94948
8	StateHoliday_b	0.72785	0.22388	3.25108
9	DayOfWeekSin	0.03324	0.00283	11.75746
10	DayOfWeekCos	0.06120	0.00350	17.49753
11	MonthSin	0.01177	0.00311	3.78388
12	DaySin	0.02297	0.00298	7.70326
13	DayCos	0.02844	0.00299	9.52200

Comparing 3 time errors:

In [63]:
```
pd.concat([model_error("1st", np.expm1(y_test), np.expm1(y_pred1)),
           model_error("2nd", np.expm1(y_test), np.expm1(y_pred2)),
           model_error("3rd", np.expm1(y_test), np.expm1(y_pred3))], axis=0)
```

Out[63]:

	MAE	MAPE	RMSE
1st	1563.96982	0.24979	2105.26150
2nd	1588.17640	0.25443	2153.90338
3rd	1606.64569	0.25788	2179.52951

In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]: