CRISP-DMRossmann

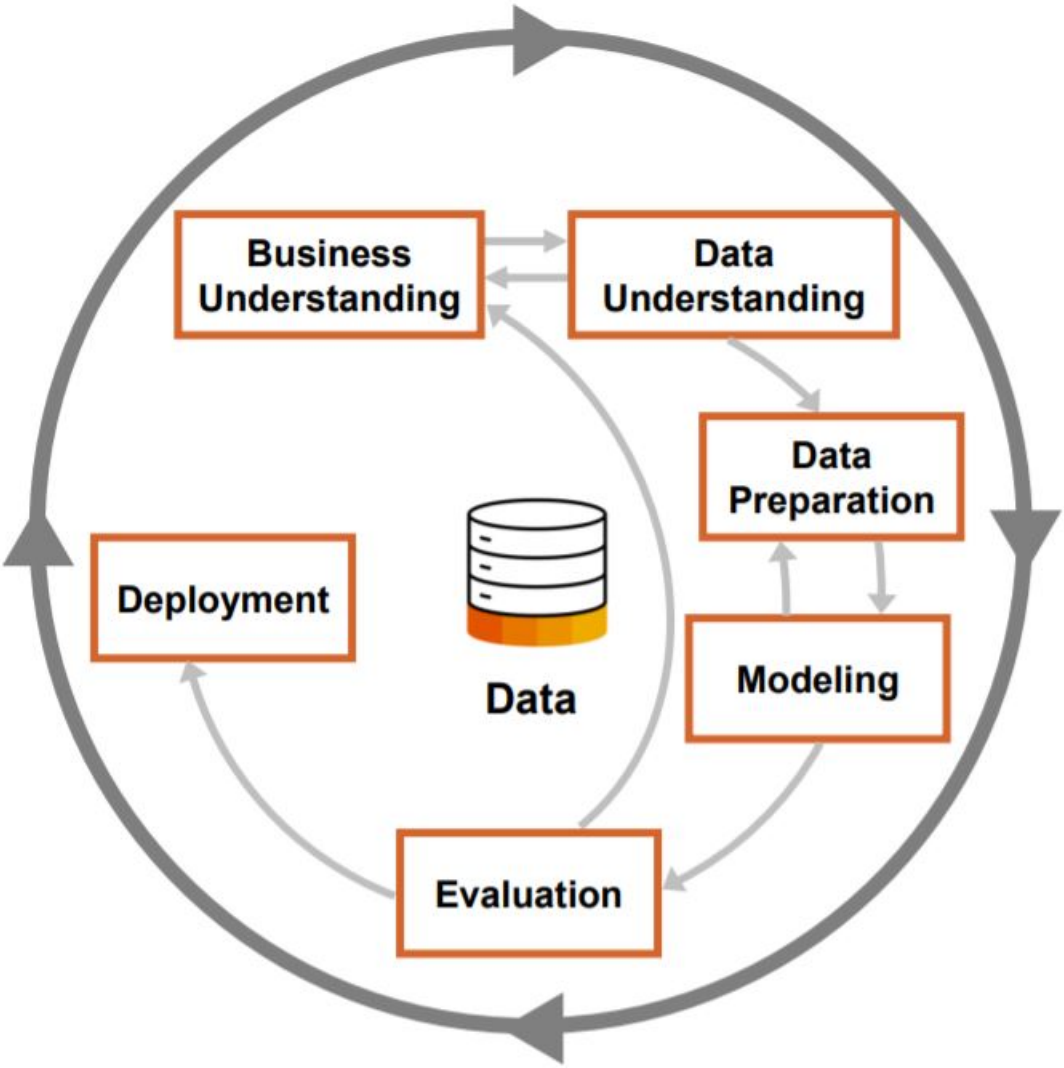
March 20, 2020

Version: 1.0 - 15.03.2020 - Benetti Mauro A.

0.0.1 Applying the CRISP-DM Method to a Business Problem

ROSSMANN Drogeriemarkt sales prediction The motivation behind this document is it relevance to real-world applications. Most data used for decision making day-to-day in industry is structured and/or in the form of a time-series. This document explore the end-to-end process of using CRISP-DM to analyses and revel insights from sales transactions.

CRISP-DM stands for cross-industry process for data mining. It provides a structured approach to planning a data mining project. It is a robust and well-proven methodology. This model is an idealized sequence of events. In practice many of the tasks can be performed in a different order and it will often be necessary to backtrack to previous tasks and repeat certain actions.



Source: CRISP-DM

---

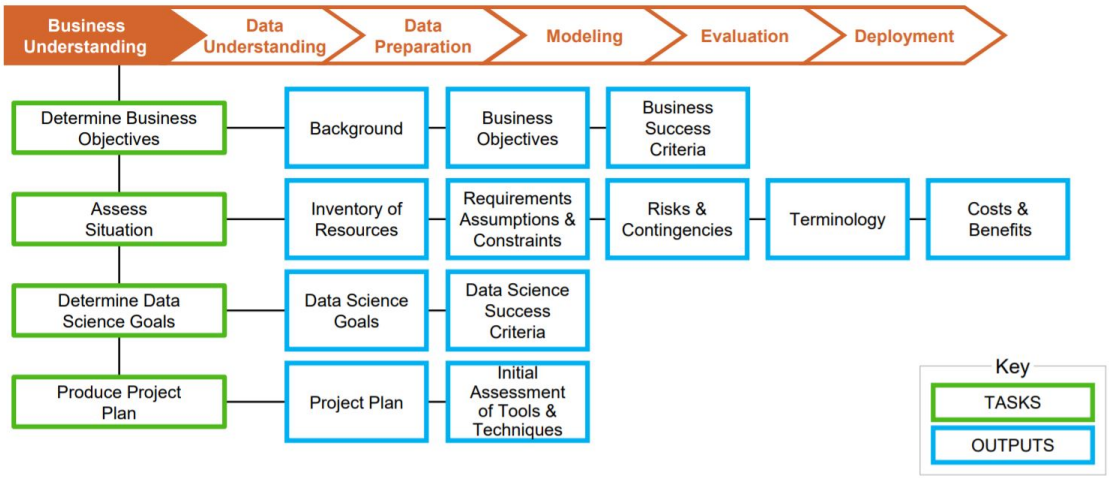
1 Table of content

List of points included in this document

Section ??

2 1. Stage One - Determine Business Objectives and Assess the Situation

The first stage of the CRISP-DM process is to understand what you want to accomplish from a business perspective. Your organisation may have competing objectives and constraints that must be properly balanced. The goal of this stage of the process is to uncover important factors that could influence the outcome of the project. Neglecting this step can mean that a great deal of effort is put into producing the right answers to the wrong questions.



Section **??**

2.1 1.1 Assess the Current Situation

This involves more detailed facts about all of the resources, constraints, assumptions and other factors that you'll need to consider when determining your data analysis goal and project plan.

- Due to the experimental nature of this analysis there is no business goal. This step should include:

Task:

- The first objective of the data analyst is to thoroughly understand, from a business perspective, what the client really wants to accomplish.

Outputs:

- Background
- Business objectives
- Business success criteria

Section **??**

2.1.1 1.1.1. Inventory of resources

List the resources available to the project.

Personnel:

- A Data Scientist

Data:

- Datasets provided by Kaggle platform

Computing resources:

- One laptop

Software:

- Jupyter Notebook
- Python 3.6
- Docker images

Section **??**

2.1.2 1.1.2. Requirements, assumptions and constraints

Requirements of the project including the schedule of completion may also include:

Data security concerns as well as any legal issues. Assumptions made by the project manager. These may be also assumptions about the data that can be verified during data mining, but may also include non-verifiable assumptions about the business related to the project. It is particularly important to list the latter if they will affect the validity of the results.

List the constraints on the project:

- Due to the experimental nature of this analysis there is no list of requirements

Section **??**

2.1.3 1.1.3. Risks and contingencies

List the risks or events that might delay the project or cause it to fail. What action will you take if these risks or events take place?

- Due to the experimental nature of this analysis there is no list of risk or contingencies measures

Section **??**

2.1.4 1.1.4. Terminology

In this poing a glossary of relevant business terminology may be mentioned. A glossary of data mining terminology, illustrated with examples relevant to the business problem in question in a business oriented level.

- In this document when a variable from the dataset is mentioned, this is highlighted `Variable Name`
- How read the boxes:

  Green Alert Box: Success. Insights to take into account.

  Warning: Yellow Alert Box: Warning. Indicate something to need to keep in mind.

  Red Alert Box: Danger.

Section **??**

2.1.5 1.1.5. Costs and benefits

Construct a cost-benefit analysis for the project which compares the costs of the project with the potential benefits to the business if it is successful. This comparison should be as specific as possible. For example, you should use financial measures in a commercial situation.

- Due to the many ways to face this analysis and the nature of this analysis there is no list of costs applicable to the project.

Section **??**

1.2 What are the desired outputs of the project?

Business success criteria

A business goal states objectives in business terminology

- Due to the experimental nature of this analysis there is no list of a business success criteria

Data mining success criteria

A data science goal states project objective in technical terms.

The analysis would be considered successful if: * Initially an exploratory data analysis (main metrics, trends and patterns in the data) for a further causal analysis should be performed. * Analysis per store type and correlational analysis of stores activity should be include * Performing a Time Series Analysis (seasonal decomposition, trends, autocorrelation, and others)

Produce project plan

- Example of a project plan

		RESOURCES					Project Time Box Example								
		SAP		CUSTOMER			Month 1					Month 2			
	Project Phase	PA	HANA	Business	PA	HANA	0	1	2	3	4	5	6	7	8
Prep Activities	0 Project Preparation and Readiness														
	0.1 Infrastructure Readiness														
	0.2 Software Readiness		x			x									
	0.3 Admin Readiness														
Core Activities	1.0 Business Understanding														
	1.1 Determine Business Objectives	x		x	x										
	1.2 Assess Situation	x		x	x										
	1.3 Determine Data Science Goals	x		x	x										
	1.4 Produce Project Plan	x		x	x										
	2.0 Data Understanding														
	2.1 Collect Initial Data	x		x	x										
	2.2 Describe Data	x		x	x										
	2.3 Explore Data	x		x	x										
	2.4 Verify Data Quality	x		x	x										
	3.0 Data Preparation														
	3.1 Select Data	x			x										
	3.2 Clean Data	x			x										
	3.3 Construct Data	x			x										
	3.4 Integrate Data	x			x										
	3.5 Format Data	x			x										
	4.0 Modeling														
	4.1 Select Modeling Technique	x			x										
	4.2 Generate Test Design	x			x										
	4.3 Build Model	x			x										
	4.4 Assess Model	x			x										
	5.0 Evaluation														
	5.1 Evaluate Results	x			x										
	5.2 Review Process	x			x										
	5.3 Determine Next Steps	x			x										
	6.0 Deployment														
	6.1 Plan Deployment	x			x										
	6.2 Plan Monitoring & Maintenance	x			x										
	6.3 Produce Final Report	x			x										
	6.4 Review Project	x			x										

Warning: This is an example of a project in the form of a Gantt diagram. It is presented only as an example, is not the project presented in this document.

Section **??**

1.3 What Questions Are We Trying To Answer?

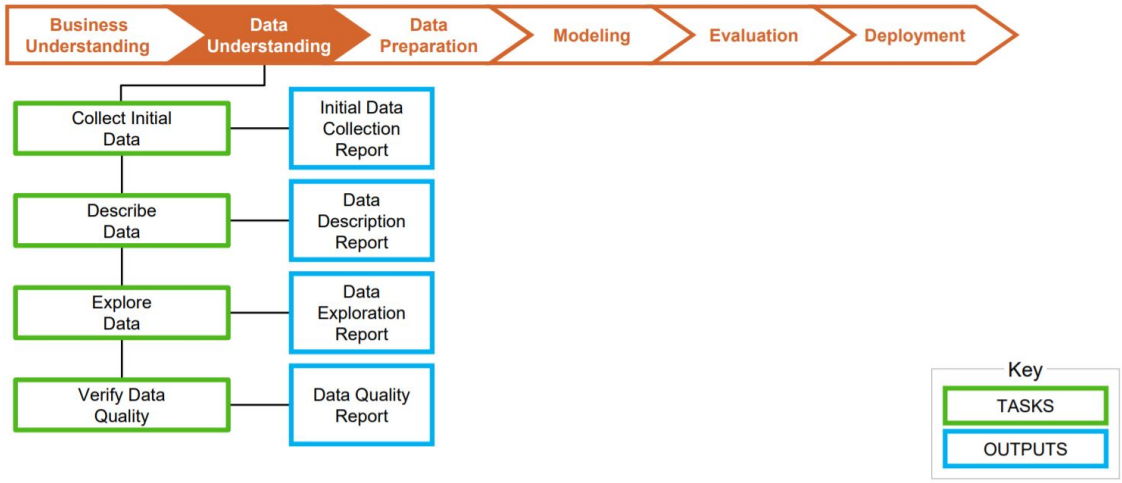
- What are the conditions that determine the sales in a particular store?
- What are the expected global sales for a particular period of time (in this case 6 weeks)
- What are the expected sales by store type in a particular period of time (in this case 6 weeks)

Section **??**

---

3 2. Stage Two - Data Understanding

The second stage of the CRISP-DM process requires you to acquire the data listed in the project resources. This initial collection includes data loading, if this is necessary for data understanding. For example, if you use a specific tool for data understanding, it makes perfect sense to load your data into this tool. If you acquire multiple data sources then you need to consider how and when you're going to integrate these.



Section **??**

3.1 2.1 Initial Data Report

List the data sources acquired together with their locations, the methods used to acquire them and any problems encountered. Record problems you encountered and any resolutions achieved. This will help both with future replication of this project and with the execution of similar future projects.

Tasks

- Acquire the data (or access to the data) listed in the project resources.
- This initial collection includes data loading into the data exploration tool and data integration if multiple data sources are acquired.

The source of the dataset is https://www.kaggle.com/c/rossmann-store-sales

We have the following informations from rossmann in two different tables:

- Id: An Id that represents a (Store, Date) duple within the test set
- Store: A unique Id for each store
- Sales: The turnover on a given day (our target variable)
- Customers: The number of customers on a given day
- Open: open: 0 = the store is closed , 1 = the store is open
- StateHoliday: Indicates a state holiday. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
- SchoolHoliday: Store on this Date was affected or not by the closure of public schools
- StoreType: 4 different stores:a,b,c,d
- Assortment: 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: Promo or not on that day
- Promo2: Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating
- Promo2Since[Year/Week]: describes the year and calendar week when the store started participating in Promo2
- PromoInterval: describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

[1]:
```
# Libraries used in this notebook

import warnings
warnings.filterwarnings("ignore")

import pandas as pd
from pandas import datetime
import numpy as np
from sklearn.model_selection import train_test_split

import operator
import matplotlib
matplotlib.use("Agg") #Needed to save figures
import matplotlib.pyplot as plt

# data visualization
import seaborn as sns # advanced vizs
import squarify
plt.style.use("ggplot") # to make the plots to look nicer

# statistics
import pandas_profiling
```

```
from pandas_profiling import ProfileReport

# time series analysis
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import xgboost as xgb
# prophet by Facebook
from fbprophet import Prophet


import os

%matplotlib inline
%pylab inline
```

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

[2]:
```
# importing train, test and store data to the notebook

train = pd.read_csv("train.csv",
                    parse_dates = True, low_memory = False, index_col = 'Date')
Test = pd.read_csv("test.csv",
                    parse_dates = True, low_memory = False, index_col = 'Date')
# the additional store data

store = pd.read_csv("store.csv",
                    low_memory = False)

# time series as indexes, verification step for indexes, uncomment in case is␣
 ↪need it
#train.index
#test.index
```

[3]:
```
#train.profile_report()
#profile = ProfileReport(train, title='Pandas Profiling Report', html={'style':
 ↪{'full_width':True}})
```

[4]:
```
#Test.profile_report()
```

[5]:
```
#store.profile_report()
```

Section **??**

3.2 2.2 Describe Data

Data description report - Describe the data that has been acquired including its format, its quantity (for example, the number of records and fields in each table), the identities of the fields and any

other surface features which have been discovered. Evaluate whether the data acquired satisfies your requirements.

First glance at the train set: head and tail

[6]:
```
print("In total: ", train.shape)
train.head(5)
```

```
In total:  (1017209, 8)
```

[6]:

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

[7]:
```
print("In total: ", Test.shape)
Test.head(5)
```

```
In total:  (41088, 7)
```

[7]:

Date	Id	Store	DayOfWeek	Open	Promo	StateHoliday	SchoolHoliday
2015-09-17	1	1	4	1.0	1	0	0
2015-09-17	2	3	4	1.0	1	0	0
2015-09-17	3	7	4	1.0	1	0	0
2015-09-17	4	8	4	1.0	1	0	0
2015-09-17	5	9	4	1.0	1	0	0

[8]:
```
print("In total: ", store.shape)
train.head(5)
```

```
In total:  (1115, 10)
```

[8]:

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

```
2015-07-31      4          5  13995       1498     1      1            0
2015-07-31      5          5   4822        559     1      1            0

            SchoolHoliday
Date
2015-07-31              1
2015-07-31              1
2015-07-31              1
2015-07-31              1
2015-07-31              1
```

[9]:
```
train.columns
```

[9]:
```
Index(['Store', 'DayOfWeek', 'Sales', 'Customers', 'Open', 'Promo',
       'StateHoliday', 'SchoolHoliday'],
      dtype='object')
```

[10]:
```
train.shape
```

[10]:
```
(1017209, 8)
```

[11]:
```
train.dtypes
```

[11]:
```
Store             int64
DayOfWeek         int64
Sales             int64
Customers         int64
Open              int64
Promo             int64
StateHoliday     object
SchoolHoliday     int64
dtype: object
```

[12]:
```
train.describe()
```

[12]:
```
                Store     DayOfWeek         Sales     Customers          Open  \
count  1.017209e+06  1.017209e+06  1.017209e+06  1.017209e+06  1.017209e+06
mean   5.584297e+02  3.998341e+00  5.773819e+03  6.331459e+02  8.301067e-01
std    3.219087e+02  1.997391e+00  3.849926e+03  4.644117e+02  3.755392e-01
min    1.000000e+00  1.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    2.800000e+02  2.000000e+00  3.727000e+03  4.050000e+02  1.000000e+00
50%    5.580000e+02  4.000000e+00  5.744000e+03  6.090000e+02  1.000000e+00
75%    8.380000e+02  6.000000e+00  7.856000e+03  8.370000e+02  1.000000e+00
max    1.115000e+03  7.000000e+00  4.155100e+04  7.388000e+03  1.000000e+00

                Promo  SchoolHoliday
count  1.017209e+06   1.017209e+06
```

```
mean   3.815145e-01   1.786467e-01
std    4.857586e-01   3.830564e-01
min    0.000000e+00   0.000000e+00
25%    0.000000e+00   0.000000e+00
50%    0.000000e+00   0.000000e+00
75%    1.000000e+00   0.000000e+00
max    1.000000e+00   1.000000e+00
```

[13]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1017209 entries, 2015-07-31 to 2013-01-01
Data columns (total 8 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   Store          1017209 non-null  int64
 1   DayOfWeek      1017209 non-null  int64
 2   Sales          1017209 non-null  int64
 3   Customers      1017209 non-null  int64
 4   Open           1017209 non-null  int64
 5   Promo          1017209 non-null  int64
 6   StateHoliday   1017209 non-null  object
 7   SchoolHoliday  1017209 non-null  int64
dtypes: int64(7), object(1)
memory usage: 69.8+ MB
```

[14]:
```
train.head(5)
```

[14]:

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

Section **??**

3.3 2.3 Verify Data Quality

Examine the quality of the data, addressing questions such as:

- Is the data complete (does it cover all the cases required)?
- Is it correct, or does it contain errors and, if there are errors, how common are they?
- Are there missing values in the data? If so, how are they represented, where do they occur, and how common are they?

Section **??**

3.3.1 2.3.1. Missing Data

In addition to incorrect datatypes, another common problem when dealing with real-world data is missing values. These can arise for many reasons and have to be either filled in or removed before we train a machine learning model. First, let's get a sense of how many missing values are in each column

While we always want to be careful about removing information, if a column has a high percentage of missing values, then it probably will not be useful to our model. The threshold for removing columns should depend on the problem

[15]:
```
# check nan
train.isnull().sum()
```

[15]:
```
Store            0
DayOfWeek        0
Sales            0
Customers        0
Open             0
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64
```

[16]:
```
Test.isnull().sum()
```

[16]:
```
Id               0
Store            0
DayOfWeek        0
Open            11
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64
```

[17]:
```
store.isnull().sum()
```

[17]:
```
Store                         0
StoreType                     0
Assortment                    0
CompetitionDistance           3
CompetitionOpenSinceMonth   354
CompetitionOpenSinceYear    354
Promo2                        0
Promo2SinceWeek             544
Promo2SinceYear             544
PromoInterval               544
dtype: int64
```

Define a function to calculate the missing values

[18]:
```
def missing_values_table(df):
        mis_val = df.isnull().sum()
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")
        return mis_val_table_ren_columns
```

[19]:
```
missing_values_table(train)
```

```
Your selected dataframe has 8 columns.
There are 0 columns that have missing values.
```

[19]:
```
Empty DataFrame
Columns: [Missing Values, % of Total Values]
Index: []
```

No missing values in the Train dataset.

[20]:
```
missing_values_table(Test)
```

```
Your selected dataframe has 7 columns.
There are 1 columns that have missing values.
```

[20]:
```
      Missing Values  % of Total Values
Open              11                0.0
```

Warning: 11 missing values in the Test dataset. Due to fact that is not a significant amount, no action was taken.

Missing values for closed stores and zero sales stores

[21]:
```
# closed stores
train[(train.Open == 0) & (train.Sales == 0)].shape
```

[21]:
```
(172817, 8)
```

Warning: There're 172817 closed stores in the dataset with 0 sales. It is about ~17% of the total amount of observations. To avoid any biased forecasts we will drop these values.

What about opened stores with zero sales?

[22]:
```
# opened stores with zero sales
zero_sales = train[(train.Open != 0) & (train.Sales == 0)]
print("In total: ", zero_sales.shape)
zero_sales.head(5)
#zero_sales.shape
```

```
In total:  (54, 8)
```

[22]:

Date	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
2015-05-15	971	5	0	0	1	0	0	1
2015-03-26	674	4	0	0	1	0	0	0
2015-02-05	699	4	0	0	1	1	0	0
2014-10-01	708	3	0	0	1	1	0	0
2014-09-22	357	1	0	0	1	0	0	0

There is one opened store with no sales on working days. There're 54 days in the dataset with open stores and no sales, so we can assume that there were external factors involved, for example manifestations.

[23]:
```
print("Closed stores and days which didn't have any sales won't be counted into␣
 ↪the forecasts. The remaining stores are: ")
train = train[(train["Open"] != 0) & (train['Sales'] != 0)]

print("In total: ", train.shape)
```

```
Closed stores and days which didn't have any sales won't be counted into the
forecasts. The remaining stores are:
In total:  (844338, 8)
```

Warning: There're still 54 days in the dataset with open stores and no sales, so we can assume that there were external factors involved, for example manifestations.

Section **??**

3.3.2 2.3.2. Outliers

At this point, we may also want to remove outliers. These can be due to typos in data entry, mistakes in units, or they could be legitimate but extreme values. For this project a simple nonparametric modeling technique and an algorithm implementing to improved and generalized the Bayesian Blocks—that to finds the optimal segmentation of the data in the observation time interval

https://ui.adsabs.harvard.edu/abs/2013ApJ…764..167S/abstract

The result is absence of outliers in the dataset.

Section **??**

3.3.3 2.3.3. Data Quality Report

Findings

- No major data quality problems were detected
- No outliers were detected
- In the Stores dataset, the NaN values in the column of the competitor stores were replaced with the mode that is half of the average. This is consider a cautious approximation

Section **??**

3.4 2.4 Initial Data Exploration

During this stage you'll address data mining questions using querying, data visualization and reporting techniques. These may include:

- **Distribution** of key attributes (for example, the target attribute of a prediction task)
- **Relationships** between pairs or small numbers of attributes
- Results of **simple aggregations**
- **Properties** of significant sub-populations
- **Simple** statistical analyses

These analyses may directly address your data mining goals. They may also contribute to or refine the data description and quality reports, and feed into the transformation and other data preparation steps needed for further analysis.

- **Data exploration report** - Describe results of your data exploration, including first findings or initial hypothesis and their impact on the remainder of the project. If appropriate you could include graphs and plots here to indicate data characteristics that suggest further examination of interesting data subsets.

3.4.1 Additional information about the stores

[24]:
```
store.head()
```

[24]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear	PromoInterval
0	1	c	a	1270.0	9.0	2008.0	0	NaN	NaN	NaN
1	2	a	a	570.0	11.0	2007.0	1	13.0	2010.0	Jan,Apr,Jul,Oct
2	3	a	a	14130.0	12.0	2006.0	1	14.0	2011.0	Jan,Apr,Jul,Oct
3	4	c	c	620.0	9.0	2009.0	0	NaN	NaN	NaN
4	5	a	a	29910.0	4.0	2015.0	0	NaN	NaN	NaN

- Store: a unique Id for each store
- 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
- Promo2: Promo2 is a continuing a 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. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

[25]:
```
#train.index
```

[26]:
```
missing_values_table(store)
```

```
Your selected dataframe has 10 columns.
There are 6 columns that have missing values.
```

[26]:

	Missing Values	% of Total Values
Promo2SinceWeek	544	48.8
Promo2SinceYear	544	48.8
PromoInterval	544	48.8
CompetitionOpenSinceMonth	354	31.7
CompetitionOpenSinceYear	354	31.7
CompetitionDistance	3	0.3

We have few variables with missing values that we need to deal with.

CompetitionDistance

[27]:
```
store[pd.isnull(store.CompetitionDistance)]
```

[27]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear	PromoInterval
290	291	d	a	NaN	NaN	NaN	0	NaN	NaN	NaN
621	622	a	c	NaN	NaN	NaN	0	NaN	NaN	NaN
878	879	d	a	NaN	NaN	NaN	1	5.0	2013.0	Feb,May,Aug,Nov

Apperently this information is simply missing from the dataset.

[28]:
```
# fill NaN with a median value (skewed distribuion)
store['CompetitionDistance'].fillna(store['CompetitionDistance'].median(),␣
↪inplace = True)
```

Warning: No particular pattern was observed. In this case, it makes a sense to replace NaN with the median values (which is twice less that the average). Lets remember that this variable is the distance to another store in direct competicion with out store.

Promo2Since...

[29]:
```
# no promo = no information about the promo?
_ = store[pd.isnull(store.Promo2SinceWeek)]
_[_.Promo2 != 0].shape
```

[29]: (0, 10)

[30]:
```
# replace NA's by 0
store.fillna(0, inplace = True)
```

Warning:

There's no missing values for `Promo2` but there's no information about the other `Promo...Interval/week/year` . Those values were replaced with zeros. The same procedure was performed for the variables related to the competitor, `CompetitionOpenSinceMonth` and `CompetitionOpenSinceYear` .

Aditional feature creation for time analysis before the join

[31]:
```
# data extraction
train['Year'] = train.index.year
train['Month'] = train.index.month
train['Day'] = train.index.day
train['WeekOfYear'] = train.index.weekofyear

# adding new variable
train['SalePerCustomer'] = train['Sales']/train['Customers']
#train['SalePerCustomer'].describe()
```

[32]:
```
print("Joining train set with an additional store information.")

# by specifying inner join we make sure that only those observations
# that are present in both train and store sets are merged together
train_store = pd.merge(train, store, how = 'inner', on = 'Store')
test_store = pd.merge(Test, store, how = 'inner', on = 'Store')

print("In total: ", train_store.shape)
train_store.head()
```

```
Joining train set with an additional store information.
In total:  (844338, 22)
```

[32]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	\
0	1	5	5263	555	1	1	0	
1	1	4	5020	546	1	1	0	
2	1	3	4782	523	1	1	0	
3	1	2	5011	560	1	1	0	
4	1	1	6102	612	1	1	0	

	SchoolHoliday	Year	Month	…	SalePerCustomer	StoreType	Assortment	\
0	1	2015	7	…	9.482883	c	a	
1	1	2015	7	…	9.194139	c	a	
2	1	2015	7	…	9.143403	c	a	
3	1	2015	7	…	8.948214	c	a	
4	1	2015	7	…	9.970588	c	a	

```
   CompetitionDistance CompetitionOpenSinceMonth  CompetitionOpenSinceYear  \
0               1270.0                        9.0                    2008.0
1               1270.0                        9.0                    2008.0
2               1270.0                        9.0                    2008.0
3               1270.0                        9.0                    2008.0
4               1270.0                        9.0                    2008.0

   Promo2  Promo2SinceWeek  Promo2SinceYear  PromoInterval
0       0              0.0              0.0              0
1       0              0.0              0.0              0
2       0              0.0              0.0              0
3       0              0.0              0.0              0
4       0              0.0              0.0              0

[5 rows x 22 columns]
```

A new dataset was created with the join of the Train and Stores datasets.

Section **??**

3.4.2 2.4.1 Distributions

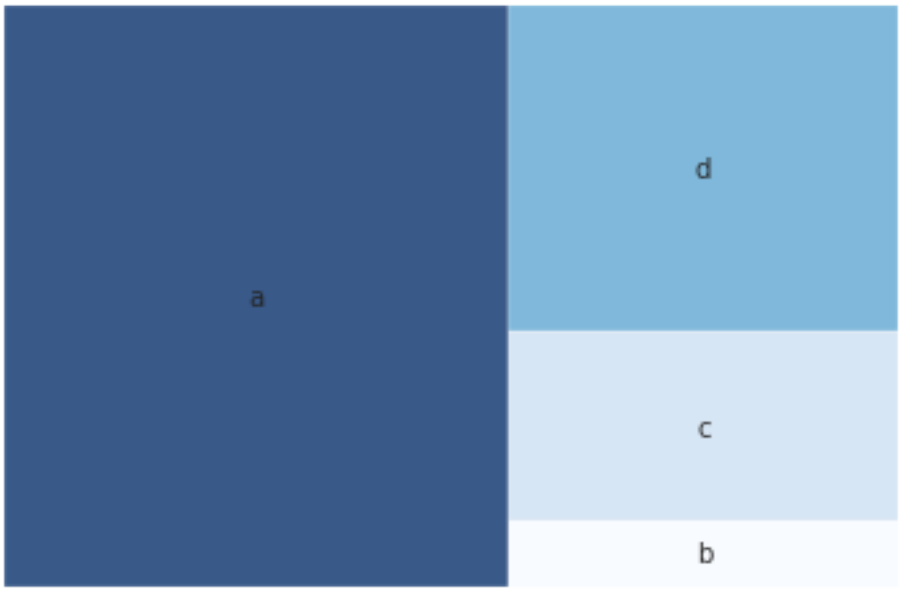
Store types In this section we will closely look at different levels of StoreType and how the main metric Sales is distributed among them.

[33]:
```
train_store.groupby('StoreType')['Sales'].describe()
```

[33]:
```
               count          mean          std     min      25%     50%  \
StoreType
a           457042.0   6925.697986  3277.351589    46.0  4695.25  6285.0
b            15560.0  10233.380141  5155.729868  1252.0  6345.75  9130.0
c           112968.0   6933.126425  2896.958579   133.0  4916.00  6408.0
d           258768.0   6822.300064  2556.401455   538.0  5050.00  6395.0

                75%      max
StoreType
a           8406.00  41551.0
b          13184.25  38722.0
c           8349.25  31448.0
d           8123.25  38037.0
```

`StoreType B` has the highest average of Sales among all others, however we have much less data for it. An overall sum of Sales and Customers will shows which `StoreType` is the most selling and crowded one.

[34]:
```
df = train_store.groupby('StoreType')['Customers', 'Sales'].sum()
# create a color palette, mapped to these values
cmap = matplotlib.cm.Blues
mini=min(df["Sales"])
maxi=max(df["Sales"])
norm = matplotlib.colors.Normalize(vmin=mini, vmax=maxi)
colors = [cmap(norm(value)) for value in df["Sales"]]

# Change color
squarify.plot(sizes=df["Customers"], alpha=.8, color=colors, label= df.index )
plt.axis('off')
plt.show()
```



[35]:
```
train_store.groupby('StoreType')['Customers', 'Sales'].sum()
```

[35]:

StoreType	Customers	Sales
a	363541431	3165334859
b	31465616	159231395
c	92129705	783221426
d	156904995	1765392943

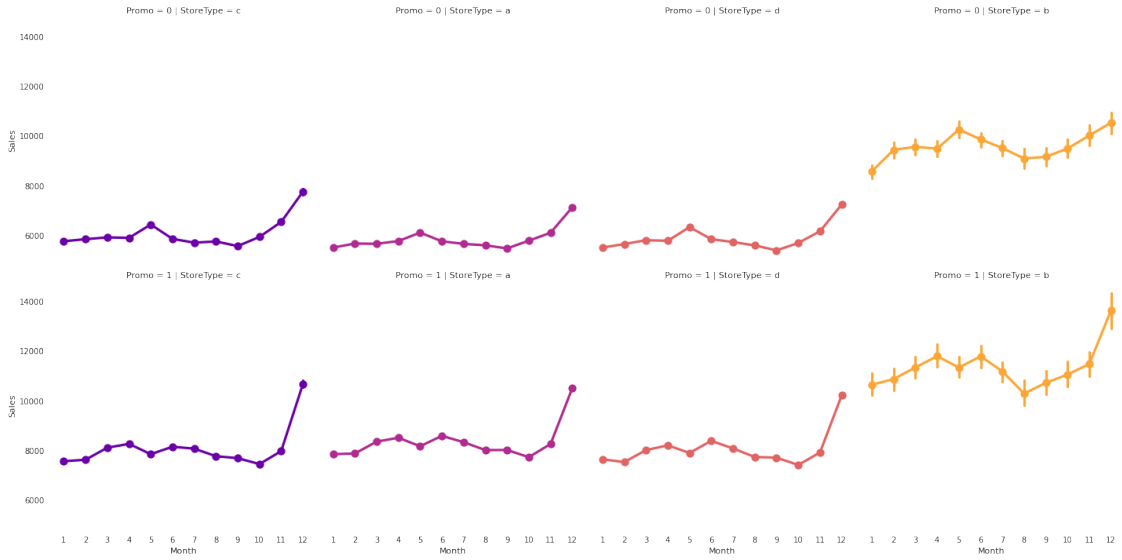
Clearly `StoresType A` sale the most. StoreType D goes on the second place in both Sales and Customers. The size of the area shows the number of customers and the intensity of the color the total sales in that type of store.

Data periods of the sales by store type

Sales time series

[36]:
```
# sales trends
sns.factorplot(data = train_store, x = 'Month', y = "Sales",
               col = 'StoreType', # per store type in cols
               palette = 'plasma',
               hue = 'StoreType',
               row = 'Promo')
```

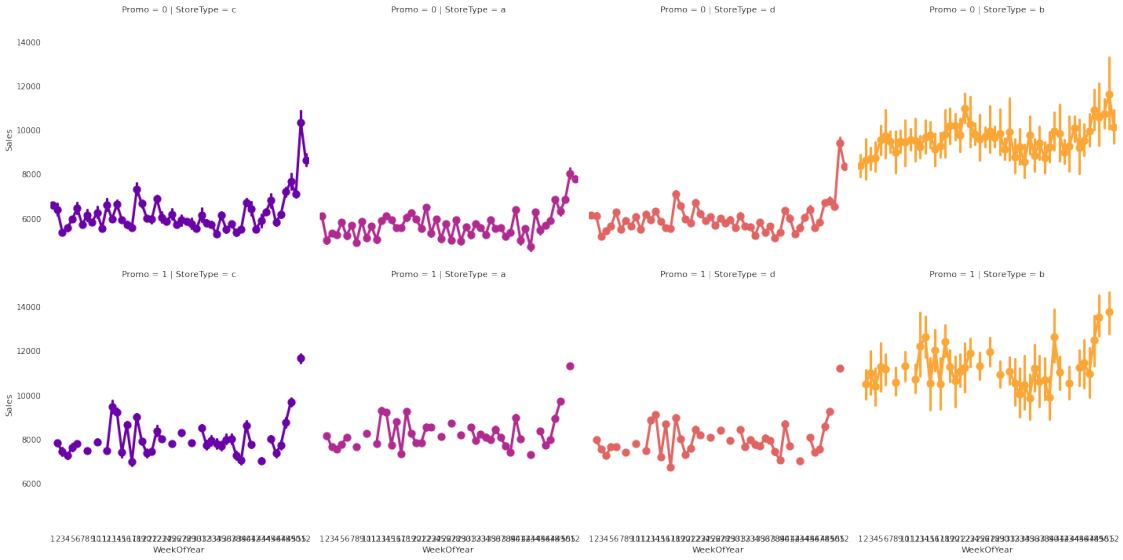
[36]: <seaborn.axisgrid.FacetGrid at 0x7ffba2bf8a90>



The graph shows a significant difference in sales with and without promotion. Eventhough the `StoresType A` are the ones that have more revenew in the time series the level of `B` types is higher, this is due to the high volativity in the sales, this can be seen more clearly in the next visualization.

[37]:
```
# sales trends
sns.factorplot(data = train_store, x = 'WeekOfYear', y = "Sales",
               col = 'StoreType', # per store type in cols
               palette = 'plasma',
               hue = 'StoreType',
               row = 'Promo')
```

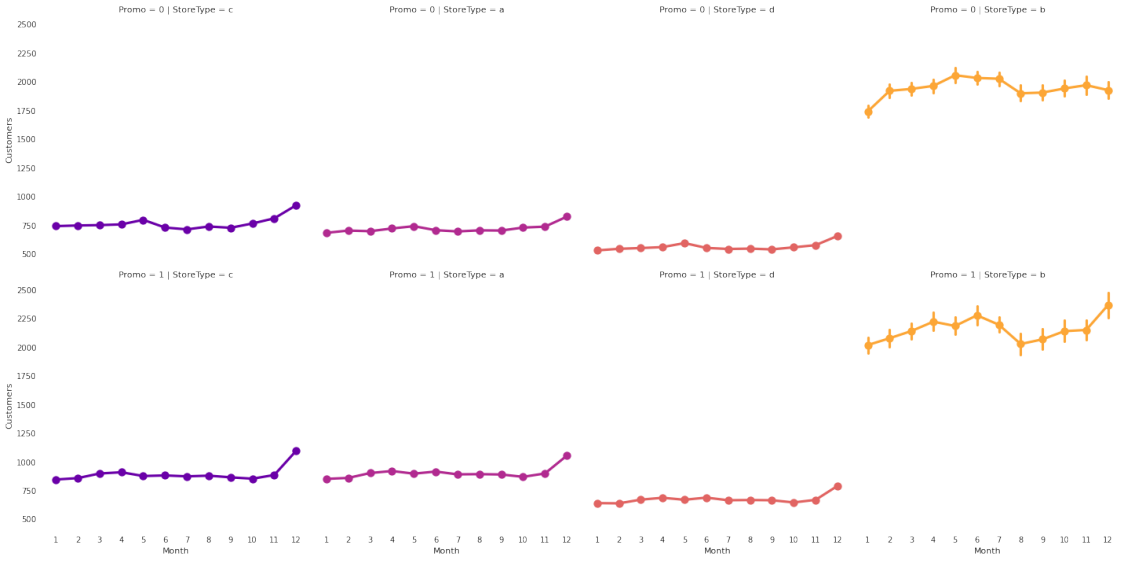
[37]: <seaborn.axisgrid.FacetGrid at 0x7ffba2b792d0>



Costumer's time series

[38]:
```
# sales trends
sns.factorplot(data = train_store, x = 'Month', y = "Customers",
               col = 'StoreType', # per store type in cols
               palette = 'plasma',
               hue = 'StoreType',
               row = 'Promo')
```

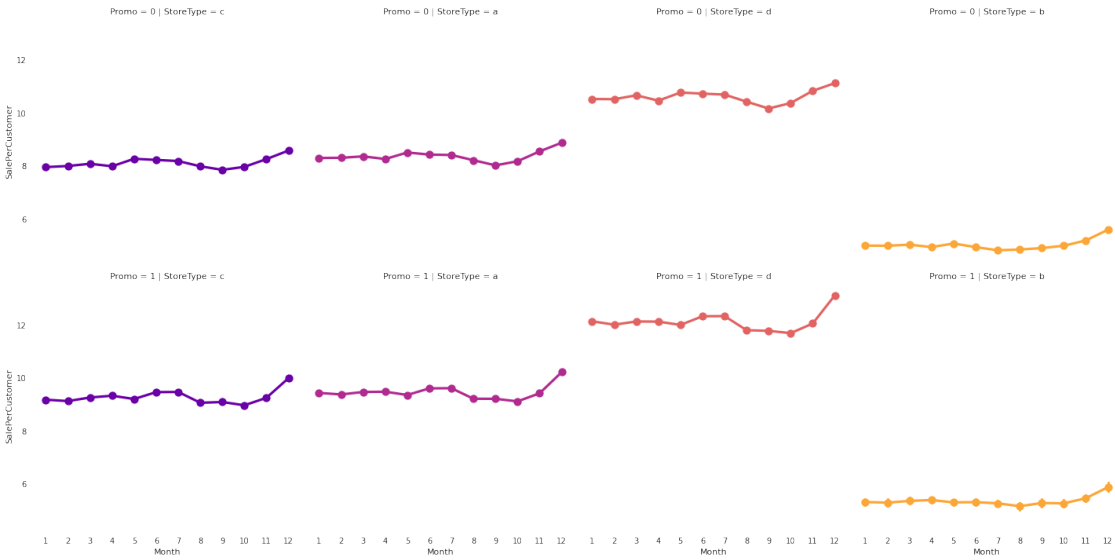
[38]: <seaborn.axisgrid.FacetGrid at 0x7ffb99cef090>



Eventhough the plots above show the `StoreType B` as the most profitable and with more customers, this is only the effect of the high volatibility in the sales. The next chart shows that in reality the highest SalePerCustomer amount shows that the `StoreType D` is the most profitable (graph below), the average sale is the about 12€ with Promo and 10€ without. In the other hand `StoreType A` and `C` it is about ~ 9€.

[39]:
```
# sale per customer trends
sns.factorplot(data = train_store, x = 'Month', y = "SalePerCustomer",
               col = 'StoreType', # per store type in cols
               palette = 'plasma',
               hue = 'StoreType',
               row = 'Promo')
```

[39]: <seaborn.axisgrid.FacetGrid at 0x7ffb9493f310>

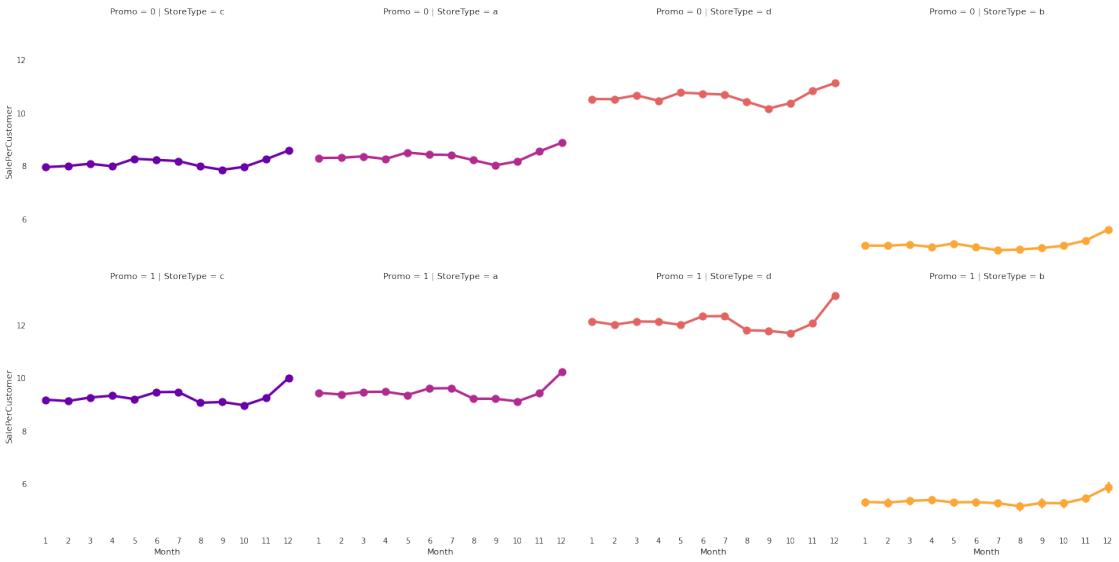


The `SalePerCustomer` amount for `StoreType` describes its Buyer‘s Cart. Eventhough the chart looks like low purchases per client, this is just because the agregation level of this time-series that is to the Month level. The high intermitence in the sales of the `StoresType B` in average is presented as low purchace by client.

Sale per customer trends

[40]:
```
sns.factorplot(data = train_store, x = 'Month', y = "SalePerCustomer",
               col = 'StoreType', # per store type in cols
               palette = 'plasma',
               hue = 'StoreType',
               row = 'Promo')
```

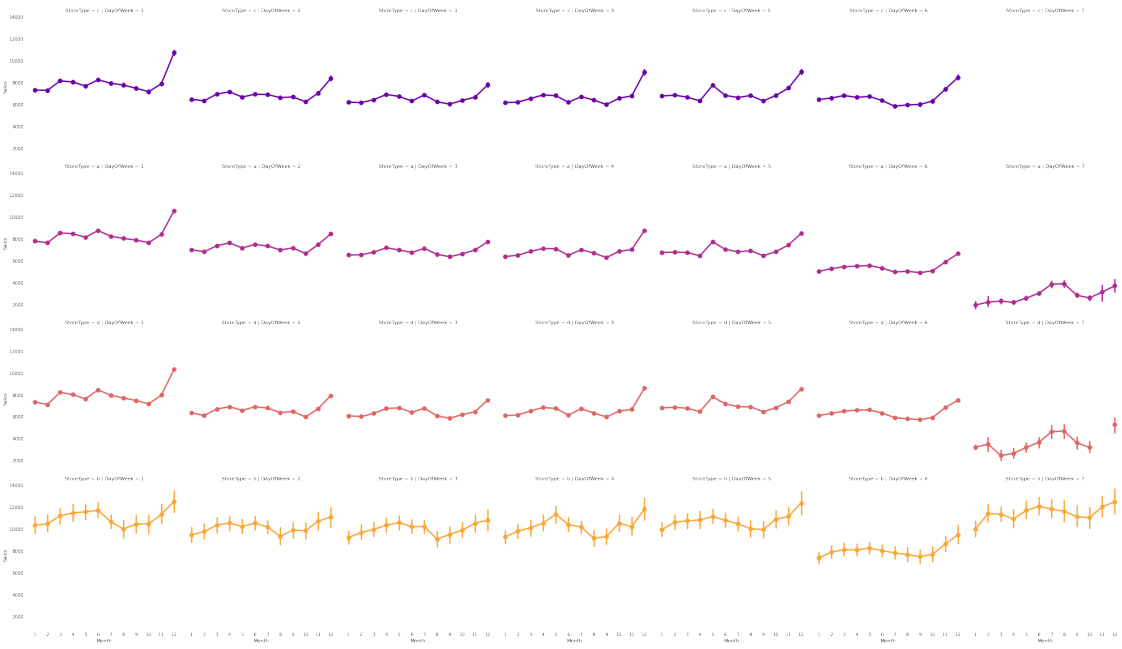
[40]: <seaborn.axisgrid.FacetGrid at 0x7ffb93c74390>



This series of charts shows the 4 types of store along the entire year. There is a clear trent to buy more in Christmas times

[41]:
```
# customers
sns.factorplot(data = train_store, x = 'Month', y = "Sales",
               col = 'DayOfWeek', # per store type in cols
               palette = 'plasma',
               hue = 'StoreType',
               row = 'StoreType')
```

[41]: <seaborn.axisgrid.FacetGrid at 0x7ffb935be7d0>



- `StoreType C` is always closed on Sunday. `StoreType D` is closes every Sunday but only in October and November. Starting on Monday in the left and ending in Sunday in the right, except for `StoreType B`, the other tent to sell less on Saturday and even less on Sunday.
- `StoreType B` present a high volatility along all periods of analysis

We can identify the stores that open on Sunday by:

[42]:
```
# stores which are opened on Sundays
train_store[(train_store.Open == 1) & (train_store.DayOfWeek == 7)]['Store'].
↪unique()
```

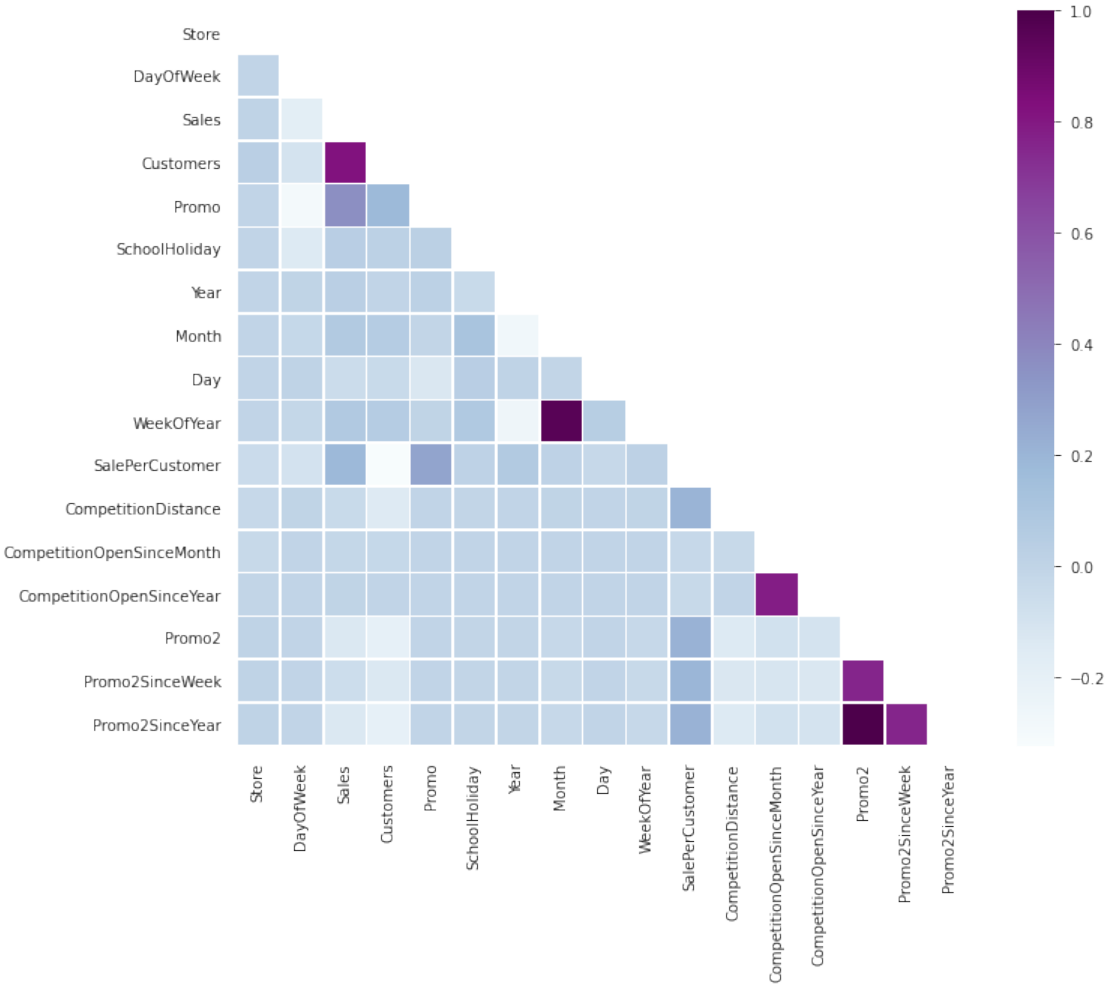
[42]:
```
array([  85,  122,  209,  259,  262,  274,  299,  310,  335,  353,  423,
        433,  453,  494,  512,  524,  530,  562,  578,  676,  682,  732,
        733,  769,  863,  867,  877,  931,  948, 1045, 1081, 1097, 1099])
```

Section **??**

3.4.3 2.4.2 Correlations

This part present any correlation of variables to eachother in this dataset, if that exist. The best way to visualize a correlation is with a heat map.

[43]:
```
# Compute the correlation matrix
# exclude 'Open' variable
corr_all = train_store.drop('Open', axis = 1).corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr_all, dtype = np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize = (11, 9))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_all, mask = mask,
            square = True, linewidths = .5, ax = ax, cmap = "BuPu")
plt.show()
```



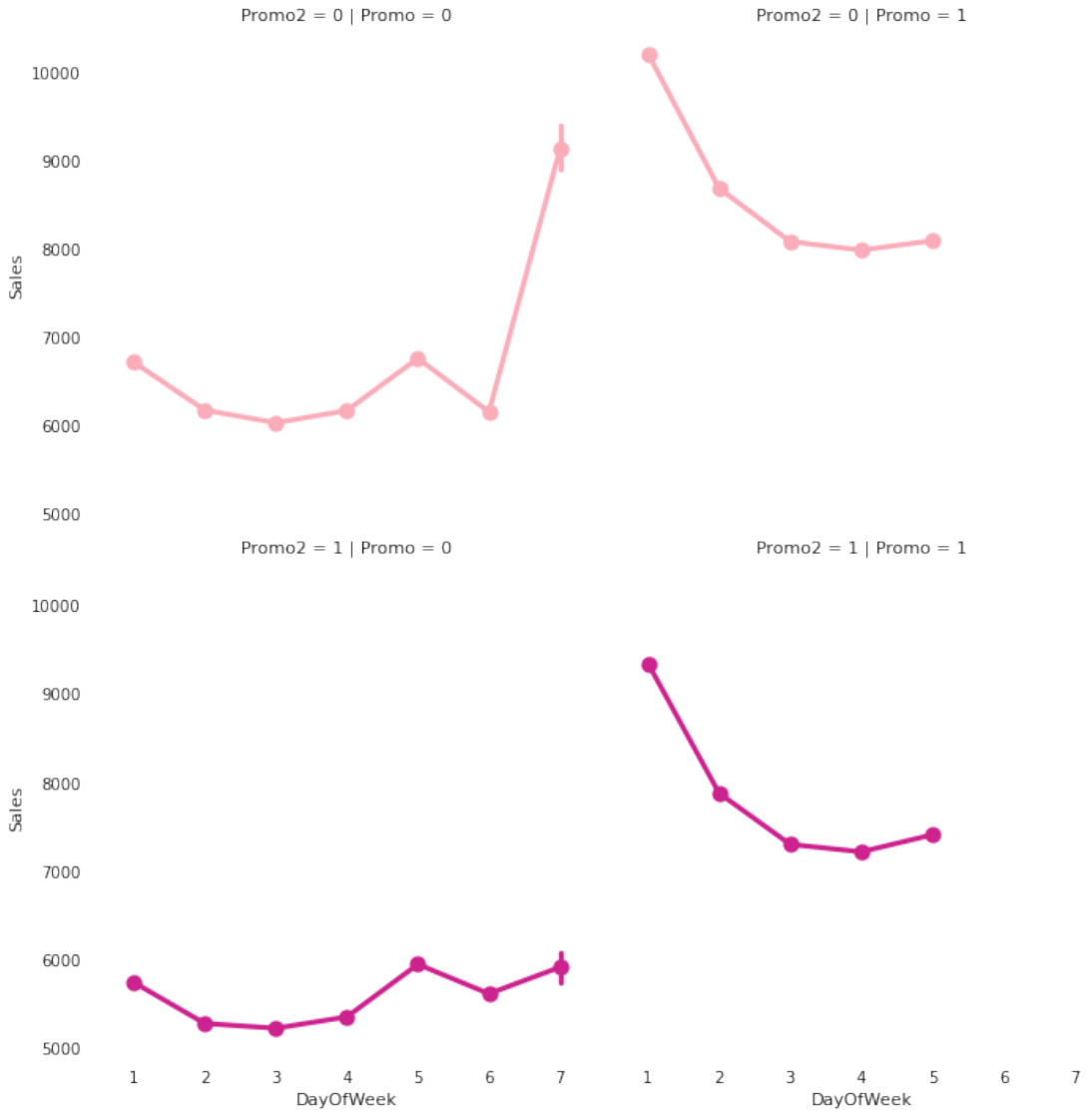
There is a natural and very strong positive correlation between the amount of Sales and the number of Customers of a store. What it's significant instead is that we can observe a positive correlation between the fact that the store had a running promotion (Promo equal to 1) and amount of Customers.

Warning:

However, as soon as the store continues a consecutive promotion (Promo2 equal to 1) the number of Customers and Sales seems to stay the same or even decrease, which is described by the pale of the blue on the heat map.

[44]:
```
# sale per customer trends
sns.factorplot(data = train_store, x = 'DayOfWeek', y = "Sales",
               col = 'Promo',
               row = 'Promo2',
               hue = 'Promo2',
               palette = 'RdPu')
```

[44]: <seaborn.axisgrid.FacetGrid at 0x7ffb8f0df290>



This is an interesting graph that shows the aggregated behaviour in sales for a particular day, when there is a promotion `Promo`= 1 and when there is a continuation of a promotion `Promo2`= 1

The four possible cases are presented as a matrix, `Promo` 1 or 0 and `Promo2` a 1 or a 0

The figure shows a clear increase in `Sales` for Sundays even when there is no promo, and a decrease during the week when there is a `Promo`, independently if a `Promo2` is present. Overall the levels of sales with promo are higher in any day of the week.

The finally all the figures shows that there is not enough data for the weekends and that the existing data is volatile

Warning:

As mentioned before, we have a strong positive correlation between the amount of `Sales` and `Customers` in a store. We can also observe a positive correlation between the fact that the store had a running promotion (`Promo` equal to 1) and amount of `Customers`.

However, as soon as the store continues a consecutive promotion (`Promo2` equal to 1) the number of `Customers` and `Sales` seems to stay the same or even decrease, which is described by the pale negative correlation on the heat map. The same negative correlation is observed between the presences of the promotion in the store the rest of a week.

Section **??**

3.4.4 2.4.3 Time-Series Analysis per Store Type

This part will focused in the analysis of the data as a time-series to identify its components

The basic assumption of a linear regression that the observations are independent, but doesn't apply in this case. Along with an increasing or decreasing trends, most timseries have some form of seasonality.

Is this case we consider more useful to build an analysis based on store types instead of individual stores. The main advantage of this approach is its simplicity of presentation and overall account for different trends and seasonalities in the dataset.

In this section, we will analyse time series data: its trends, sesonalities and autocorrelation. Usually at the end of the analysis, we are able to develop a seasonal ARIMA (Autoregression Integrated Moving Average) model but it won't be our main focus today. Instead, we try to understand the data, and only later come up with the forecasts using Prophet methodology.

We take four stores from store types to represent their group:

- `Store` number 2 for `StoreType` A
- `Store` number 85 for `StoreType` B
- `Store` number 1 for `StoreType` C
- `Store` number 14 for `StoreType` D

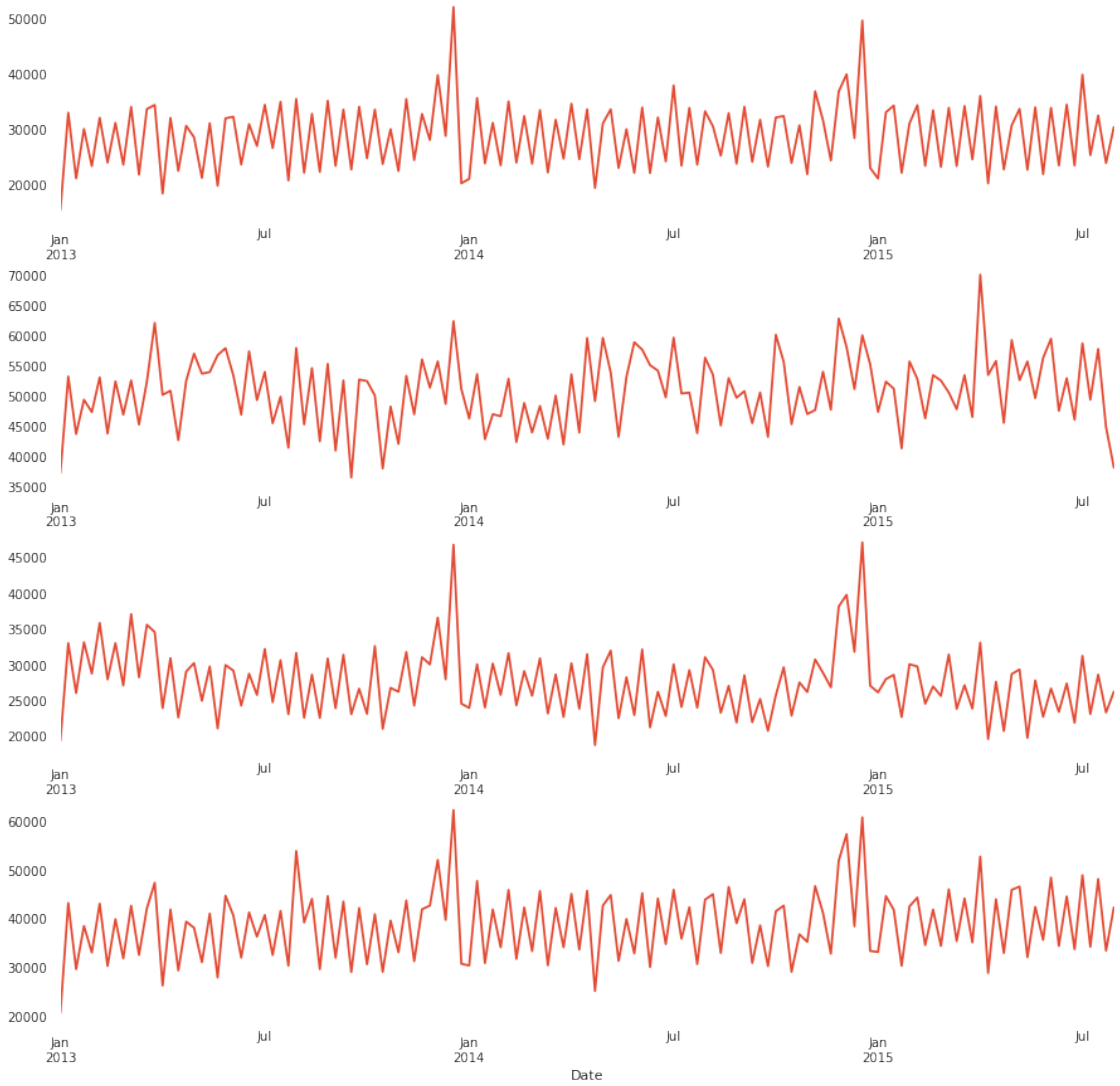
The selection was arbitrary and the only purpose is simplifie the visualization of the trends. It also makes sense to downsample the data from days to weeks using the resample method to see the present trends more clearly.

3.4.5 Behavior of the four differente types of store

[45]:
```
pd.plotting.register_matplotlib_converters()

# preparation: input should be float type
train['Sales'] = train['Sales'] * 1.0

# store types
sales_a = train[train.Store == 2]['Sales']
sales_b = train[train.Store == 85]['Sales']
sales_c = train[train.Store == 1]['Sales']
```

```
sales_d = train[train.Store == 15]['Sales']

f, (ax1, ax2, ax3, ax4) = plt.subplots(4, figsize = (15, 15))

# store types
sales_a.resample('W').sum().plot( ax = ax1)
sales_b.resample('W').sum().plot( ax = ax2)
sales_c.resample('W').sum().plot( ax = ax3)
sales_d.resample('W').sum().plot( ax = ax4)
```

[45]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffb8f526e10>



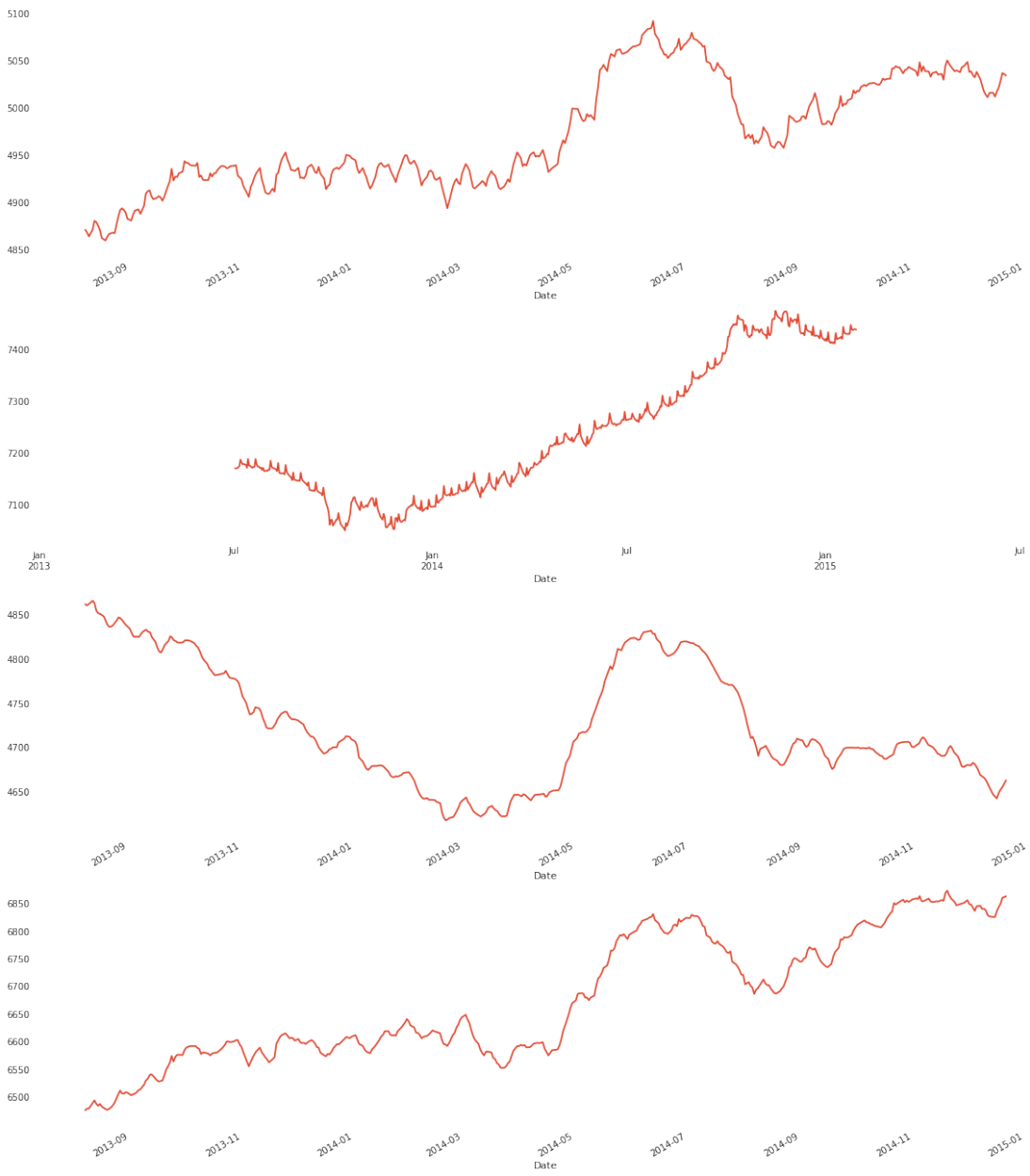
Four diferent `StoresType` along two and a half years of data.

3.4.6 Individual decomposition of components

Visualization of the agregated behavior for every `StoreType`

[46]:
```
f, (ax1, ax2, ax3, ax4) = plt.subplots(4, figsize = (20, 25))

# monthly
decomposition_a = seasonal_decompose(sales_a, model = 'additive', freq = 365)
decomposition_a.trend.plot(ax = ax1)

decomposition_b = seasonal_decompose(sales_b, model = 'additive', freq = 365)
decomposition_b.trend.plot(ax = ax2)

decomposition_c = seasonal_decompose(sales_c, model = 'additive', freq = 365)
decomposition_c.trend.plot(ax = ax3)

decomposition_d = seasonal_decompose(sales_d, model = 'additive', freq = 365)
decomposition_d.trend.plot(ax = ax4)
```

[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb9d419790>
```



Findings:

- The above figure shows the four types of stores along the two years. Now is possible identify an overall trend in the data.
- Is clear the upper trend of `StoresType A`, `StoresType B` and `StoresType D`.
- Nevertheless `StoresType B` seems to have a down trend with a short recovery between April and June 2014.

Behavior of StoreType A

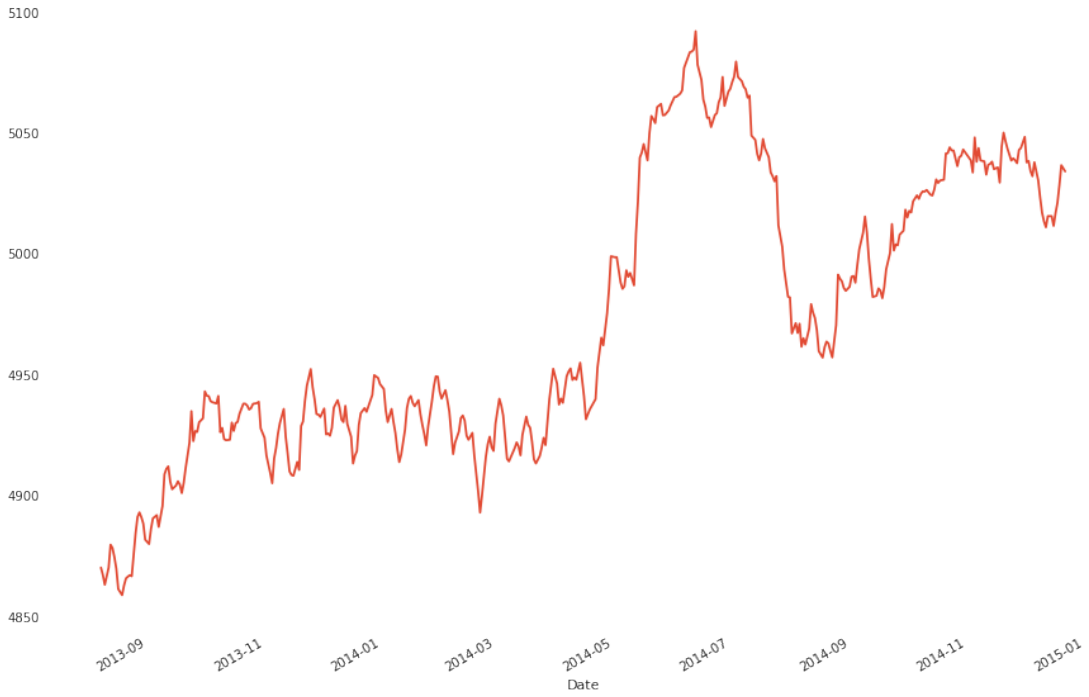
[47]:
```
#Increse de size of the figures for the next analysis
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [15, 10]
```

[48]:
```
from statsmodels.tsa.seasonal import STL

stl = STL(sales_a.resample('W').sum(), seasonal= 365)
res = stl.fit()
fig = res.plot()
```

Let's compare again with the aggregate sales of all stores type A

[49]:
```
# uncomment the next two lines if is necesary a closer comparison of the␣
 ↪individual trend of the store with the group.
decomposition_a = seasonal_decompose(sales_a, model = 'additive', freq = 365)
decomposition_a.trend.plot()
```

[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb9d119810>
```



Findings:

- The four figures above present the decomposition of component for the given time-series. The first figure shows the original time-series, the second the trend, the third the seasonality and the last figure the residuals.
- The trend line shows a positive and upper trend along all the time windows
- The season line shows clear peaks in Christmas times
- The residual scatter plot shows a high volatility in the period from January and July every single year.

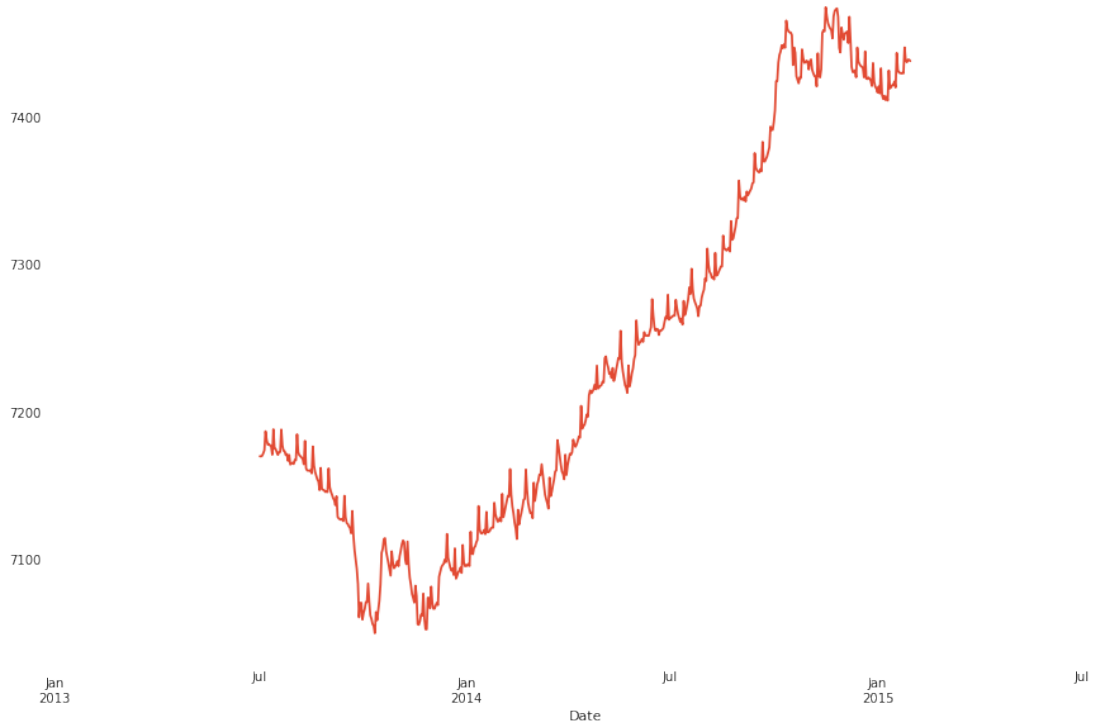
Behavior of StoreType B

[50]:
```
from statsmodels.tsa.seasonal import STL

stl = STL(sales_b.resample('W').sum(), seasonal= 365)
res = stl.fit()
fig = res.plot()
```

Let's compare again with the aggregate sales of all stores type B

[51]:
```
decomposition_b = seasonal_decompose(sales_b, model = 'additive', freq = 365)
decomposition_b.trend.plot()
```

[51]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb9cf16510>
```

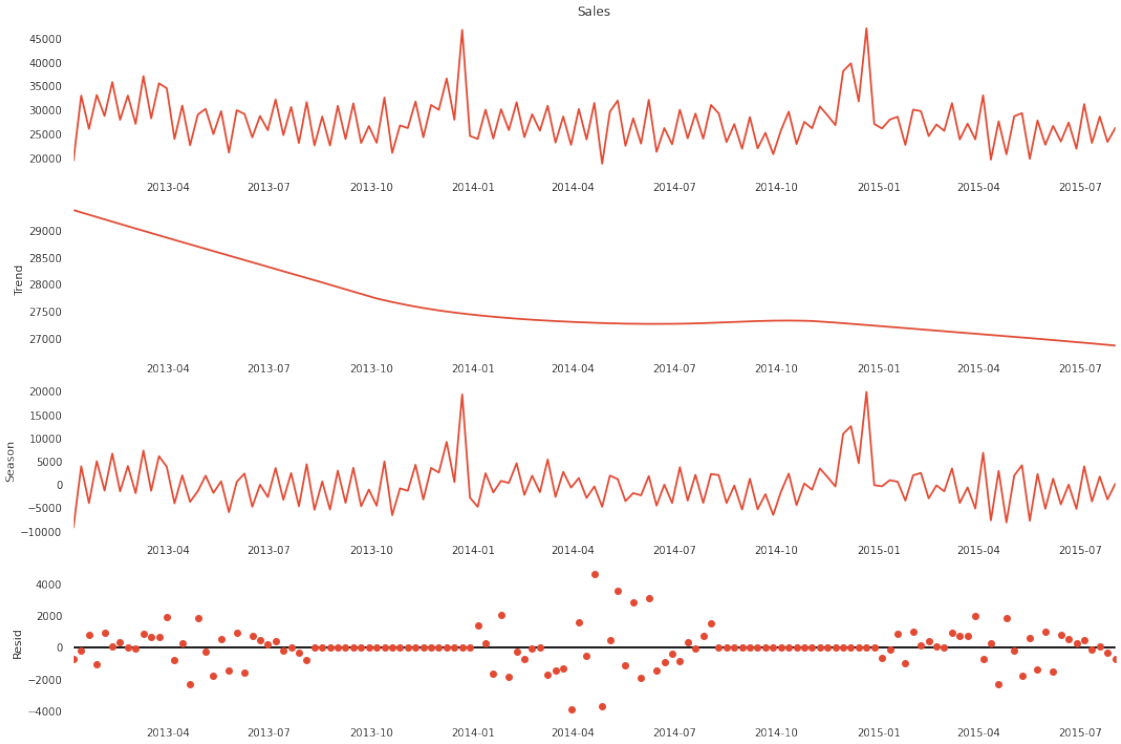


Findings:

- The four figures above present the decomposition of component for the given time-series. The first figure shows the original time-series, the second the trend, the third the seasonality and the last figure the residuals.
- The trend line shows three stages a negative, a positive and a stable trend along all the time windows.
- The season line shows several not very clear peaks in the periods May-June and Christmas-New Year.
- The residual scatter plot shows stables periods between August-December every year. year.

Behavior of Type C stores

[52]:
```
from statsmodels.tsa.seasonal import STL

stl = STL(sales_c.resample('W').sum(), seasonal= 365)
res = stl.fit()
fig = res.plot()
```



Let's compare again with the aggregate sales of all stores type C

[53]:
```
decomposition_c = seasonal_decompose(sales_c, model = 'additive', freq = 365)
decomposition_c.trend.plot()
```

[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffb9cccbe10>

Findings:

- This is the only type of store with a negative trend in the long term
- The seasonality and volatile perios are the same than the rest of the store types.

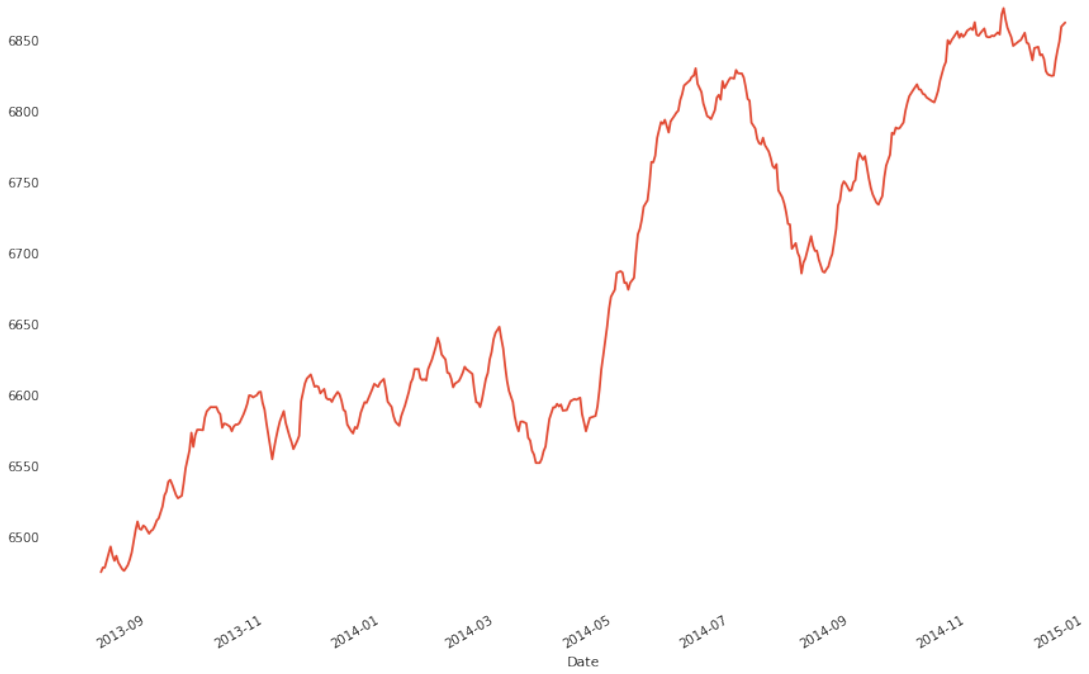
Behavior of StoreType D

[54]:
```
from statsmodels.tsa.seasonal import STL

stl = STL(sales_d.resample('W').sum(), seasonal= 365)
res = stl.fit()
fig = res.plot()
```

Let's compare again with the aggregate sales of all stores type D

[55]:
```
decomposition_d = seasonal_decompose(sales_d, model = 'additive', freq = 365)
decomposition_d.trend.plot()
```

[55]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffb9cac4fd0>



Findings:

- This type of store shows in contrast of type C, a positive trend with similar behavior in seasonality and residual than the rest of the stores.

Section **??**

3.5 2.5 Data Exploration Report

Describe results of this task, including:

- First findings or initial hypothesis and their impact on the remainder of the project
- If appropriate, include graphs and plots

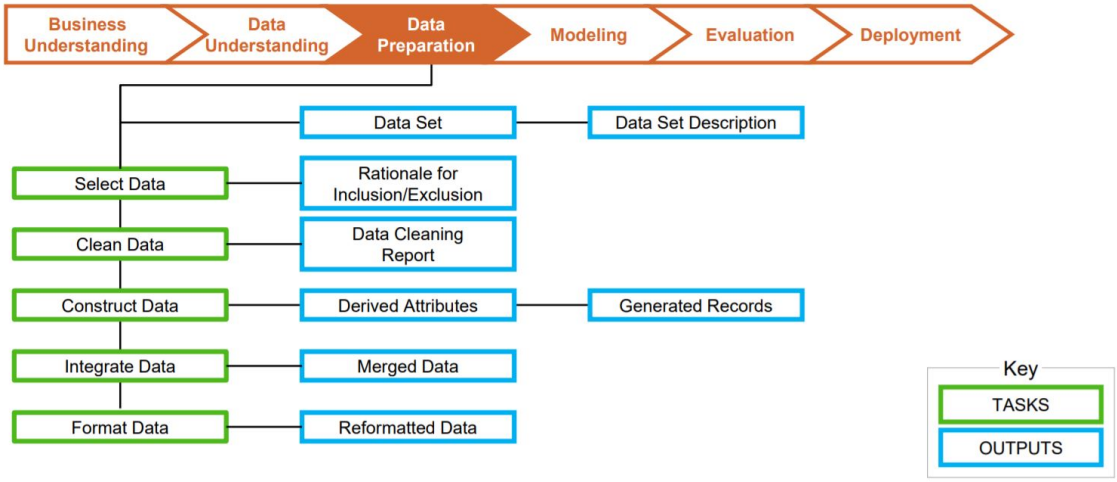
Findings:

- Strong correlation between Sales per customer and Promo
- No notable effect for sales when the promotion continue in time
- 17.0% of the sales records (172871) on the Train dataset
- The stores presented as closed and no sales where suppressed of the dataset, in total (172817)
- 17.0% of the customer records (172869) on the Train dataset, all this records were suppressed with the records of 0 sales mentioned early.
- There is 54 days of open stores with 0 sales are still in the dataset, those are considered real events (Eg. a manifestation)

Section **??**

4 3. Stage Three - Data Preperation

This is the stage of the project where you decide on the data that you're going to use for analysis. The criteria you might use to make this decision include the relevance of the data to your data mining goals, the quality of the data, and also technical constraints such as limits on data volume or data types. Note that data selection covers selection of attributes (columns) as well as selection of records (rows) in a table.



Section **??**

4.1 3.1 Select Your Data

- The data selected for the modelling is the one is provided by the platform Kaggle.com
- The 3 datasets provided were the only ones used for the modelling.
- The data was already spited into Train and Test.

Section **??**

4.2 3.2 Clean The Data

This task involves raise the data quality to the level required by the analysis techniques that you've selected. This may involve selecting clean subsets of the data, the insertion of suitable defaults, or more ambitious techniques such as the estimation of missing data by modelling.

No special tasks of cleansing were perfomed in the datasets provided.

Section **??**

4.2.1 3.2.1 Label Encoding

Label Encoding to turn Categorical values to Integers

No labelling or encoding was necessary.

Section **??**

4.2.2 3.2.2 Drop Unnecessary Columns

Sometimes we may not need certain columns.

No column was dropped.

Section **??**

4.2.3 3.2.3 Altering Data Types

Sometimes we may need to alter data types. Including to/from object datatypes

No data type was changed.

Section **??**

4.2.4 3.2.4 Dealing With Zeros

Replacing all the zeros from cols. **Note** You may not want to do this - add / remove as required

In point 2.3.1 Missing Data were checked NN and Zeros for the 3 datasets: Train, Test, and Store.

Warning:

- 11 null values were found in Open column on the Test dataset (no correction was performed)
- No zeros or missing values for Stores datasets
- There're 172817 closed stores in the data with 0 sales. It is about (17.0%) of the total amount of observations [2.1]. To avoid any biased forecast these values were dropped.
- There is one opened store with no sales on working days.
- There're only 54 days in the data, so we can assume that there were external factors involved, for example manifestations.

Section **??**

4.2.5 3.2.5 Dealing With Duplicates

No duplicates were removed in this analysis.

Section **??**

4.3 3.4 Integrate Data

These are methods whereby information is combined from multiple databases, tables or records to create new records or values.

Merged data - Merging tables refers to joining together two or more tables that have different information about the same objects. For example a retail chain might have one table with information about each store's general characteristics (e.g., floor space, type of mall), another table with summarised sales data (e.g., profit, percent change in sales from previous year), and another with information about the demographics of the surrounding area. Each of these tables contains one record for each store. These tables can be merged together into a new table with one record for each store, combining fields from the source tables.

Warning: The dataset train.csv and test.csv was join in [2.4] of this document, with "store.csv" in the colum [Store].

Aggregations - Aggregations refers to operations in which new values are computed by summarising information from multiple records and/or tables. For example, converting a table of customer purchases where there is one record for each purchase into a new table where there is one record for each customer, with fields such as number of purchases, average purchase amount, percent of orders charged to credit card, percent of items under promotion etc.

No permanent aggregation were performed in the dataset, only the ones performed for the purpose of visualization.

Section **??**

---

5 4. Stage Four - Exploratory Data Analysis

5.0.1 Conclusion of EDA

- The most selling and crowded `StoreType` is A.
- The best "Sale per Customer" is `StoreType` D, showing the higher Buyer Cart. To benefit from this fact, Rossmann can consider proposing bigger variety of its products.
- Low SalePerCustomer amount for `StoreType` B indicates to the possible fact that people shop there essentially for "small" things. Eventhough this `StoreType` generated the least amount of sales and customers over the whole period, it shows a great potential.
- Customers tends to buy more on Modays when there's one promotion (`Promo`) and on Sundays when there's no promotion at all (both `Promo` and `Promo1` are equal to 0).
- `Promo2` alone doesn't seem to be correlated to any significant change in the Sales amount.

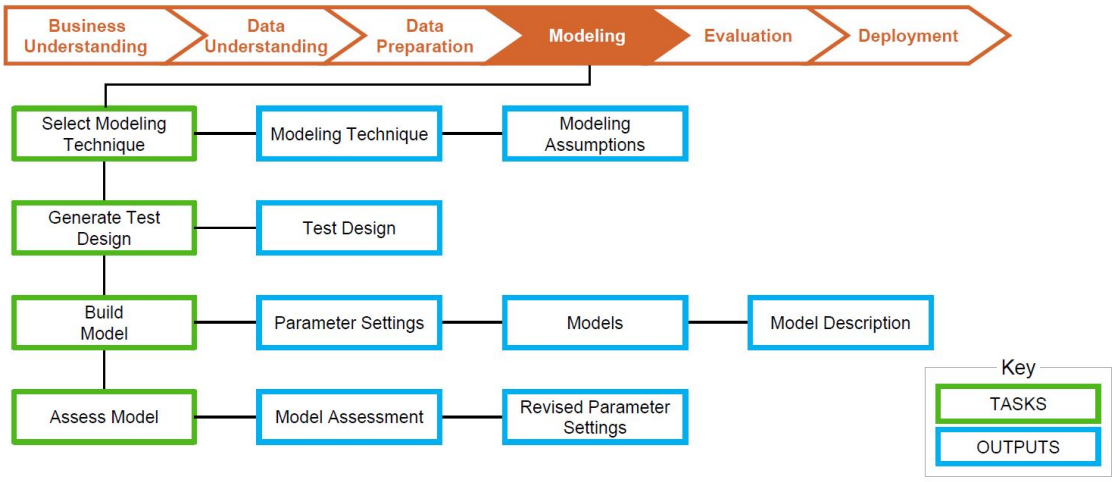
Section **??**

---

6 5. Stage Four - Modelling

As the first step in modelling, you'll select the actual modelling technique that you'll be using. Although you may have already selected a tool during the business understanding phase, at this stage you'll be selecting the specific modelling technique e.g. decision-tree building with C5.0, or

neural network generation with back propagation. If multiple techniques are applied, perform this task separately for each technique.



Section **??**

6.1 5.1. Modelling technique

6.1.1 Time Series Analysis and Forecasting with Prophet

Forecasting for the next 6 weeks for the first store The Core Data Science team at Facebook recently published a new procedure for forecasting time series data called Prophet. It is based on an additive model where non-linear trends are fit with yearly and weekly seasonality, plus holidays. It enables performing automated forecasting to individual stores at scale in Python 3.

[56]:
```
# importing data
df = pd.read_csv("train.csv",
                 low_memory = False)

# remove closed stores and those with no sales
df = df[(df["Open"] != 0) & (df['Sales'] != 0)]

# sales for the store number 1 (StoreType C)
sales = df[df.Store == 14].loc[:, ['Date', 'Sales']]

# reverse to the order: from 2013 to 2015
sales = sales.sort_index(ascending = False)

# to datetime64
sales['Date'] = pd.DatetimeIndex(sales['Date'])
sales.dtypes
```

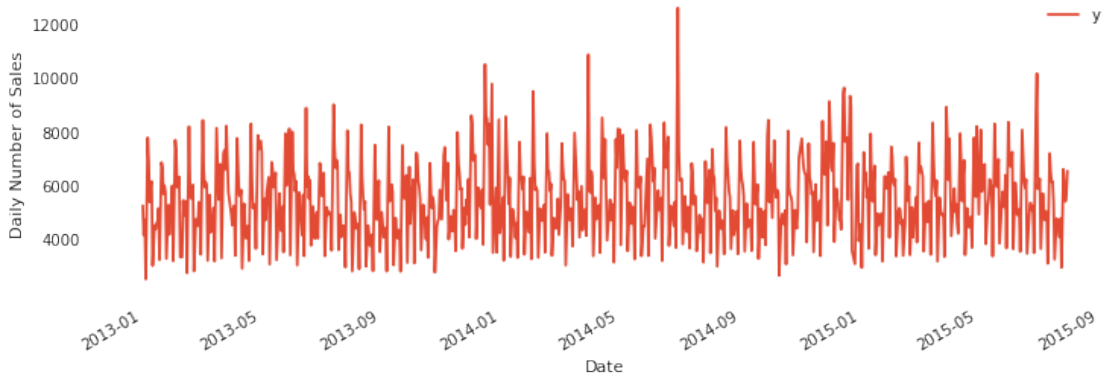
[56]:
```
Date     datetime64[ns]
Sales             int64
```

```
dtype: object
```

[57]:
```
# from the prophet documentation every variables should have specific names
sales = sales.rename(columns = {'Date': 'ds',
                                'Sales': 'y'})
sales.head()
```

[57]:
```
                ds     y
1014993 2013-01-02  5253
1013878 2013-01-03  4169
1012763 2013-01-04  4734
1011648 2013-01-05  2519
1009418 2013-01-07  7788
```

[58]:
```
# plot daily sales
ax = sales.set_index('ds').plot(figsize = (12, 4))
ax.set_ylabel('Daily Number of Sales')
ax.set_xlabel('Date')
plt.show()
```



The chart shows the sales of the `Store` = 14 along two and half years. Is not easy identify a trend or a patter in the visualization.

Modeling Holidays Prophet also allows to model for holidays. The StateHoliday variable in the dataset indicates a state holiday, at which all stores are normally closed. There are also school holidays in the dataset at which ceratin stores are also closing their doors.

[59]:
```
# create holidays dataframe
state_dates = df[(df.StateHoliday == 'a') | (df.StateHoliday == 'b') & (df.
 ↪StateHoliday == 'c')].loc[:, 'Date'].values
school_dates = df[df.SchoolHoliday == 1].loc[:, 'Date'].values
```

```
state = pd.DataFrame({'holiday': 'state_holiday',
                      'ds': pd.to_datetime(state_dates)})
school = pd.DataFrame({'holiday': 'school_holiday',
                      'ds': pd.to_datetime(school_dates)})

holidays = pd.concat((state, school))
holidays.head()
```

[59]:

	holiday	ds
0	state_holiday	2015-06-04
1	state_holiday	2015-06-04
2	state_holiday	2015-06-04
3	state_holiday	2015-06-04
4	state_holiday	2015-06-04

[60]:
```
# set the uncertainty interval to 95% (the Prophet default is 80%)
my_model = Prophet(interval_width = 0.95,
                   holidays = holidays)
my_model.fit(sales)

# dataframe that extends into future 6 weeks
future_dates = my_model.make_future_dataframe(periods = 6*7)

#print("First week to forecast.")
#future_dates.tail(7)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
```

In this step the model is created for one individual store.

[ ]:
```
# predictions
forecast = my_model.predict(future_dates)

# preditions for last week
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail(7)
```

The result of the prediction for the `Store` = 14 for 6 weeks, (only the first 7 recods are presented as example). The forecast object here is a new dataframe that includes a column yhat with the forecast, as well as columns for components and uncertainty intervals.

[ ]:
```
fc = forecast[['ds', 'yhat']].rename(columns = {'Date': 'ds', 'Forecast':␣
↪'yhat'})
```

Prophet plots the observed values of our time series (the black dots), the forecasted values (blue line) and the uncertainty intervals of our forecasts (the blue shaded regions).

[ ]:
```
# visualizing predicions
my_model.plot(forecast);
```

This graph shows that not many datapoints fall outside the prediction corridor, the 95% confidence upper and lower limits in light blue. We also can say that the model is better to catch the lower values of the prediction than the upper values, this means that out model tent to underestimate the sales.

One other particularly strong feature of Prophet is its ability to return the components of our forecasts. This can help reveal how daily, weekly and yearly patterns of the time series plus manyally included holidayes contribute to the overall forecasted values:

[ ]:
```
my_model.plot_components(forecast);
```

The first plot shows that the monthly sales of `Store` = 14 has been linearly increasing over time and the second shows the holiays gaps included in the model. The third plot highlights the fact that the weekly volume of last week sales peaks towards the Monday of the next week, while the forth plot shows that the most buzy season occurs during the Christmas holidays and between the March-July period.

Section **??**

6.2 5.2. Modelling assumptions

Many modelling techniques make specific assumptions about the data, for example that all attributes have uniform distributions, no missing values allowed, class attribute must be symbolic etc. Record any assumptions made.

- The information provided is accurate.
- No important or relevan information is missing during the analysis and modeling.
- The holidays information in the dataset is precise and there is no omisions.
- Stores with the same type have the sames caracteristic and have the same behavior in time.
- Is not taken into account for this analysis the location of the stores. The location component is not included.
- The macro economic variables remain stables and in a similar state to the historical data for the forecast periode.

Section **??**

6.3 5.3. Model Build

Parameter settings - With any modelling tool there are often a large number of parameters that can be adjusted. List the parameters and their chosen values, along with the rationale for the choice of parameter settings.

Warning:

- Uncertainty interval to 95% (the Prophet default is 80%)
- Forecast extends into future 6 weeks

- A holidays dataframe must be created as imput for the model
- Time series must be in reverse to the order: from 2013 to 2015
- To decrease the number of missing values it was removed closed stores and those with no sales

Model

These are the actual models produced by the modelling tool, not a report on the models.

Prophet model = `my_model`

Components of the model = *my_model.plot_components(forecast)*

- Trend
- Holidays
- Weekly behavior
- Yearly behavior

Model descriptions - Prophet it is based on an additive model where non-linear trends are fit with yearly and weekly seasonality, plus holidays. It enables performing automated forecasting which are already implemented in R at scale in Python 3.

Not all forecasting problems can be solved by the same procedure. Prophet is optimized for the business forecast tasks we have encountered at Facebook, which typically have any of the following characteristics:

- Hourly, daily, or weekly observations with at least a few months (preferably a year) of history
- Strong multiple "human-scale" seasonalities: day of week and time of year
- Important holidays that occur at irregular intervals that are known in advance (e.g. the Super Bowl)
- A reasonable number of missing observations or large outliers
- Historical trend changes, for instance due to product launches or logging changes
- Trends that are non-linear growth curves, where a trend hits a natural limit or saturates

Source: https://research.fb.com/prophet-forecasting-at-scale/

Advantages

- A powerful tool for the time series forecasting as it accounts for time dependencies, seasonalities and holidays (Prophet: manually).
- Easily implemented with R auto.arima() from forecast package, which runs a complex grid search and sophisticated algorithm behind the scene.

Drawbacks

- Doesn't catch interactions between external features, which could improve the forecasting power of a model. In our case, these variables are Promo and CompetitionOpen.
- Even though Prophet offers an automated solution for ARIMA, this methodology is under development and not completely stable.
- Fitting seasonal ARIMA model needs 4 to 5 whole seasons in the dataset, which can be the biggest drawback for new companies.
- Seasonal ARIMA in Python has 7 hyper parameters which can be tuned only manually affecting significantly the speed of the forecasting process.

Source: ARIMA model to univariate time series

Section **??**

6.4 5.4. Assess Model

Interpret the models according to your domain knowledge, your data mining success criteria and your desired test design. Judge the success of the application of modelling and discovery techniques technically, then contact business analysts and domain experts later in order to discuss the data mining results in the business context. This task only considers models, whereas the evaluation phase also takes into account all other results that were produced in the course of the project.

At this stage you should rank the models and assess them according to the evaluation criteria. You should take the business objectives and business success criteria into account as far as you can here. In most data mining projects a single technique is applied more than once and data mining results are generated with several different techniques.

Model assessment - Summarise the results of this task, list the qualities of your generated models (e.g.in terms of accuracy) and rank their quality in relation to each other.

Revised parameter settings - According to the model assessment, revise parameter settings and tune them for the next modelling run. Iterate model building and assessment until you strongly believe that you have found the best model(s). Document all such revisions and assessments.

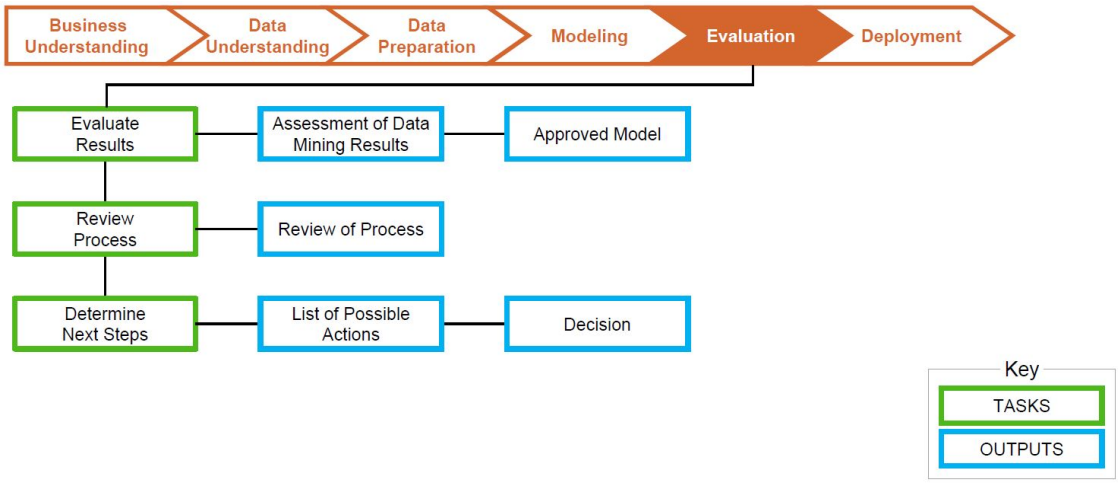
Section **??**

---

7 Evaluation

Previous evaluation steps dealt with factors such as the accuracy and generality of the model. During this step you'll assesses the degree to which the model meets your business objectives and seek to determine if there is some business reason why this model is deficient. Another option is to test the model(s) on test applications in the real application, if time and budget constraints permit. The evaluation phase also involves assessing any other data mining results you've generated. Data mining results involve models that are necessarily related to the original business objectives and all other findings that are not necessarily related to the original business objectives, but might also unveil additional challenges, information, or hints for future directions.

- Assessment of data mining results - Summarise assessment results in terms of business success criteria, including a final statement regarding whether the project already meets the initial business objectives.
- Approved models - After assessing models with respect to business success criteria, the generated models that meet the selected criteria become the approved models.



Review process

At this point, the resulting models appear to be satisfactory and to satisfy business needs. It is now appropriate for you to do a more thorough review of the data mining engagement in order to determine if there is any important factor or task that has somehow been overlooked. This review also covers quality assurance issues—for example: did we correctly build the model? Did we use only the attributes that we are allowed to use and that are available for future analyses?

- Review of process
- Summarise the process review and highlight activities that have been missed and those that should be repeated.

Determine next steps

Depending on the results of the assessment and the process review, you now decide how to proceed.Do you finish this project and move on to deployment, initiate further iterations, or set up new data mining projects? You should also take stock of your remaining resources and budget as this may influence your decisions.

- List of possible actions
- List the potential further actions, along with the reasons for and against each option.
- Decision - Describe the decision as to how to proceed.

Section **??**

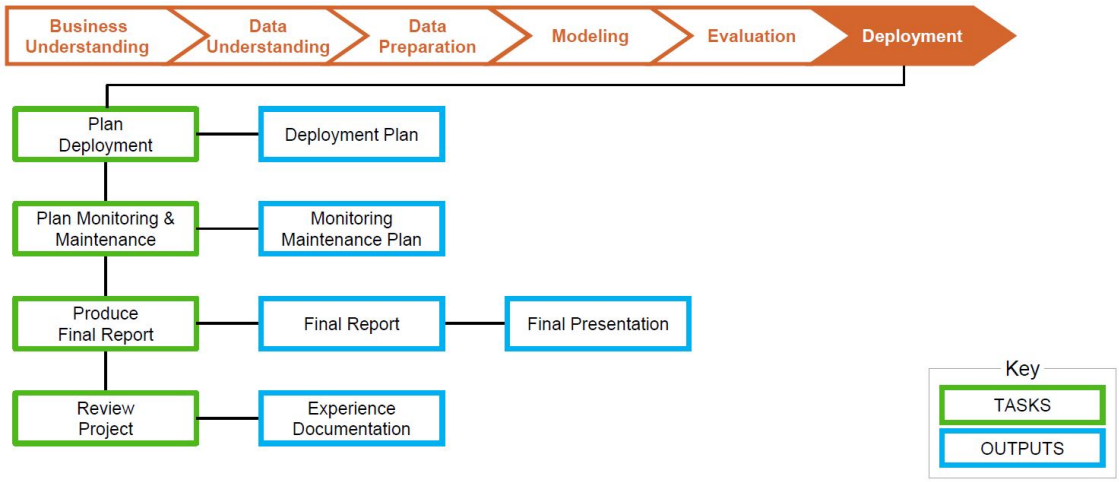
---

8 Deployment

With our analysis complete, we move onto the final stage of the CRISP-DM process — Deployment. All implementation and deployment information related should be include in this chapter. In the deployment stage you'll take your evaluation results and determine a strategy for their deployment. If a general procedure has been identified to create the relevant model(s), this procedure is documented here for later deployment. It makes sense to consider the ways and means of deployment during the business understanding phase as well, because deployment is absolutely crucial to the

success of the project. This is where predictive analytics really helps to improve the operational side of your business.

- Deployment plan
- Summarise your deployment strategy including the necessary steps and how to perform them.
- Monitoring maintenance plan
- Final report



At the end of the project you will write up a final report. Depending on the deployment plan, this report may be only a summary of the project and its experiences (if they have not already been documented as an ongoing activity) or it may be a final and comprehensive presentation of the data mining result(s).

- Final report - This is the final written report of the data mining engagement. It includes all of the previous deliverables, summarising and organising the results.
- Final presentation - There will also often be a meeting at the conclusion of the project at which the results are presented to the customer.

Our deployment strategy must simply ensure that our analysis is readable, reproducible, and maintainable. We satisfy these requirements by generating a Jupyter Notebook detailing every step of the analytical process and documenting and justifying all choices made in data cleaning and feature generation, model selection and evaluation.

Additionally, we provide instructions for installation and usage of our notebook in a README file.

Section **??**

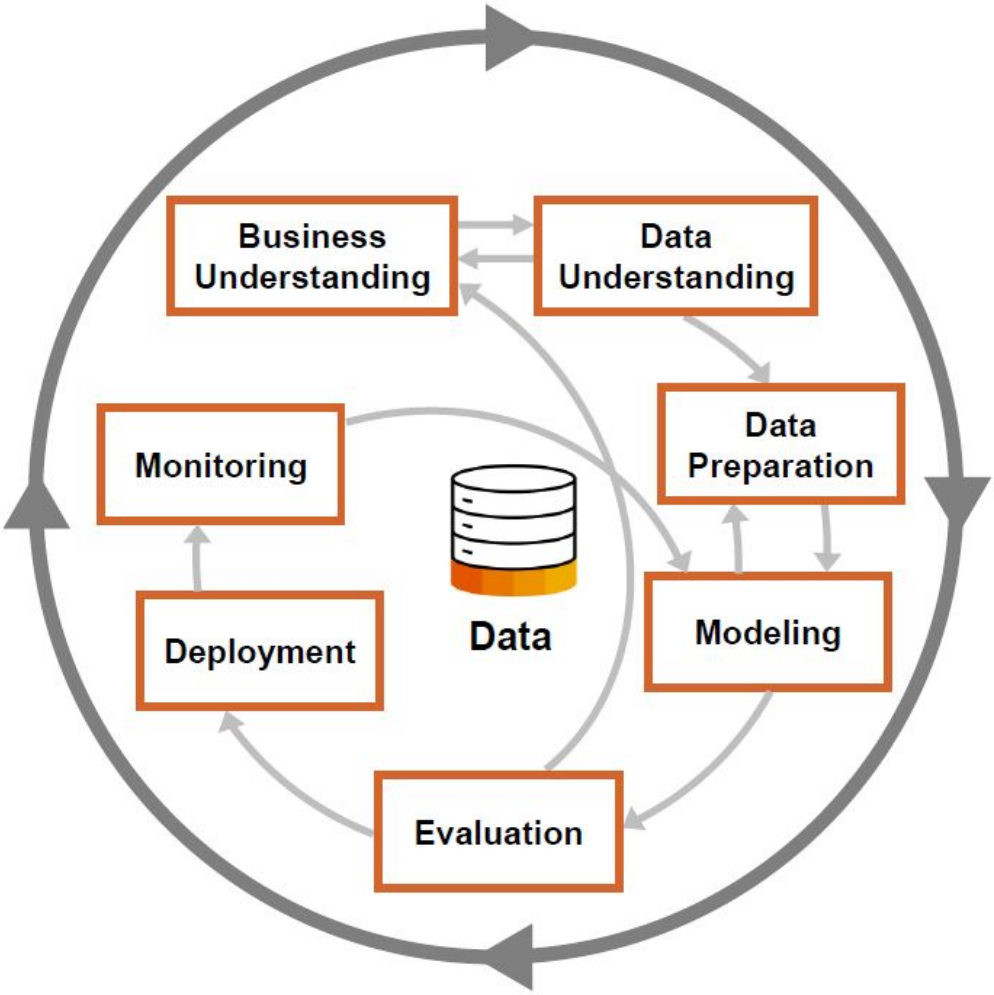
---

9 Monitoring

Monitoring and maintenance are important issues if the data mining result becomes part of the day-to-day business and its environment. The careful preparation of a maintenance strategy helps to avoid unnecessarily long periods of incorrect usage of data mining results. In order to monitor

the deployment of the data mining result(s), the project needs a detailed monitoring process plan. This plan takes into account the specific type of deployment.

- Monitoring and maintenance plan
- Summarise the monitoring and maintenance strategy, including the necessary steps and how to perform them.



Section **??**

Centralize the monitoring of predictive models that are used in production is essential in a changing environment.

Best practices may include:

- Automate and schedule model apply and retraining tasks
- Monitor model performance automatically
- Automate the model lifecycle management
- Create predictive models using a fully automated modeling workflow

Factors that influence when you should update your models:

- Business environment change

- Frequency of model usage
- The age of a model
- Data readiness

Models should be thoroughly evaluated and measured for accuracy, predictive power, stability over time, and other appropriate metrics that are defined by the model's objectives.

[ ]: