[Jupyter Notebook](http://localhost:8889/tree)

rossmann (autosaved) 

Python [default]

* [File](http://localhost:8889/notebooks/rossmann.ipynb)
* [Edit](http://localhost:8889/notebooks/rossmann.ipynb)
* [View](http://localhost:8889/notebooks/rossmann.ipynb)
* [Insert](http://localhost:8889/notebooks/rossmann.ipynb)
* [Cell](http://localhost:8889/notebooks/rossmann.ipynb)
* [Kernel](http://localhost:8889/notebooks/rossmann.ipynb)
* [Widgets](http://localhost:8889/notebooks/rossmann.ipynb)
* [Help](http://localhost:8889/notebooks/rossmann.ipynb)

**http://localhost:8889/notebooks/static/img/rosslog.pngRossmann Stores Predictions**

This notebook covers basics of data cleaning, EDA, feature engineering, and data modelling. For more in-deph data analysis, please consider Kaggle Kernels: <https://www.kaggle.com/c/rossmann-store-sales/kernels>

**IMPORTANT: Make sure you do not skip any cells. In that case you risk not being able to accomplish given tasks.**

**Rossmann Data Cleaning**

You are provided with historical sales data for 1,115 Rossmann stores. The task is to forecast the "Sales" column for the test set. In this first part of the notebook we will walk you through basic steps that need to be done before start with baseline modelling.

**Load libraries and data**



# NumPy for numerical computing

import numpy as np

​

# Pandas for DataFrames

import pandas as pd

from pandas import DataFrame

from pandas import TimeGrouper

​

# Matplotlib for visualization

from matplotlib import pyplot as plt

​

# Display plots in the notebook

%matplotlib inline

​

# Seaborn for easier visualization

import seaborn as sns



# Load data from CSV

train\_dataset = pd.read\_csv("static/data/train.csv", low\_memory=False)

stores = pd.read\_csv("static/data/store.csv", low\_memory=False)

​

train\_dataset.head()

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

The train data set contains sales by day for each store with the following columns:

* Store - a unique id number
* DayOfWeek/Date - the day of the week (1-7) and date (YYYY-MM-DD) for a sales data point
* Sales - the sales for a given day
* Customers - the number of customers on a given day. This column is highly correlated with sales and is not present in the test set.
* Open - Values: 0 = closed, 1 = open
* Promo - indicates whether a store was running a sales promotion that day
* StateHoliday - Values: a = public holiday, b = Easter holiday, c = Christmas, 0 = None
* SchoolHoliday - indicates if a store was affected by the closure of public schools on that day

**Question 1**



# YOUR CODE GOES HERE

# Print first five rows of the "stores" dataset

stores.head()

|  | **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 |

The stores data set contains additional columns about each store that does not vary by day:

* StoreType - differentiates between 4 different store models: a, b, c, d
* Assortment - describes the level of products available: a = basic, b = extra, c = extended
* CompetitionDistance - distance in meters to the nearest competitor
* CompetitionOpenSince[Month/Year] - month/year when the nearest competitor was opened
* Promo2 - indicator for a recurring promotion: 0 = store not participating, 1 = participating
* Promo2Since[Week/Year] - calendar week/year when the store started participating in Promo2
* PromoInterval - describes the intervals when Promo2 is started. E.g. "Feb,May,Aug,Nov" means each round starts in those months of any given year for that store

**Question 2**



# YOUR CODE GOES HERE

# Chech out the shape of the datasets.

# We have a lot of data observations to play around.

print(train\_dataset.shape)

print(stores.shape)

(1017209, 9)

(1115, 10)

Take a look and understand different data types.



# Column datatypes

print(train\_dataset.dtypes,'\n')

print(stores.dtypes)

Store int64

DayOfWeek int64

Date object

Sales int64

Customers int64

Open int64

Promo int64

StateHoliday object

SchoolHoliday int64

dtype: object

Store int64

StoreType object

Assortment object

CompetitionDistance float64

CompetitionOpenSinceMonth float64

CompetitionOpenSinceYear float64

Promo2 int64

Promo2SinceWeek float64

Promo2SinceYear float64

PromoInterval object

dtype: object

**Drop unwanted data**

The overall goals during the data cleaning phase is to fix any problems with the provided data sets, such as:

* Fix inconsistent data
* Impute missing data (There are many ways to impute data, but for the purpose of this assignment, we will simply drop missing data for convenience)
* Convert categorical variables to one-hot encoded binary variables
* Check for outlying values, and correct them if necessary

We needed to fix these types of problems so that our prediction models could be fit as accurately as possible.



# Check for duplicates

train\_dataset = train\_dataset.drop\_duplicates()

stores = stores.drop\_duplicates()

​

# Print shape after removing duplicates

train\_dataset.shape, stores.shape

((1017209, 9), (1115, 10))

There were no duplicates. Now, let us check closed stores, stores with no customers, and stores where sales values equals zero.



# Check how many records are for days when stores are closed

print("The number of records for closed stores: ", train\_dataset[train\_dataset.Open == 0].shape)

The number of records for closed stores: (172817, 9)

Even though there are substantial number of records recorded when stores are closed, we are going to drop these observations because sales, our response variable, would be 0 for the days when the store is closed (there is no need for regression).



# Drop observations with closed stores

train\_dataset = train\_dataset[train\_dataset.Open != 0]

train\_dataset.shape

(844392, 9)

**Question 3**



# YOUR CODE GOES HERE

# Print the number of stores where Customers=0

print("The number of stores where Customers=0: ", train\_dataset[train\_dataset.Customers == 0].shape)

​

The number of stores where Customers=0: (52, 9)

**Question 4**



# YOUR CODE GOES HERE

# Print the number of stores where Sales=0

print("The number of stores where Sales=0: ", train\_dataset[train\_dataset.Sales == 0].shape)

​

The number of stores where Sales=0: (54, 9)

Let us check for missing values in train\_dataset and stores



print(train\_dataset.isnull().sum())

print(stores.isnull().sum())

Store 0

DayOfWeek 0

Date 0

Sales 0

Customers 0

Open 0

Promo 0

StateHoliday 0

SchoolHoliday 0

dtype: int64

Store 0

StoreType 0

Assortment 0

CompetitionDistance 3

CompetitionOpenSinceMonth 354

CompetitionOpenSinceYear 354

Promo2 0

Promo2SinceWeek 544

Promo2SinceYear 544

PromoInterval 544

dtype: int64

Drop unuseful columns:

* in train\_dataset: Open is unuseful because it is consisted of all 1's (we dropped the observations with Open = 0 earlier).
* in stores: CompetitionOpenSinceMonth, CompetitionOpenSinceYear, Promo2SinceWeek, Promo2SinceYear, PromoInterval because they have too many missing values.



train\_dataset.drop(['Open'], axis=1, inplace=True)

stores.drop(['CompetitionOpenSinceMonth',

'CompetitionOpenSinceYear',

'Promo2SinceWeek',

'Promo2SinceYear',

'PromoInterval'], axis=1, inplace=True)

Drop the 3 rows of stores where CompetitionDistance is missing:



stores.dropna(inplace=True)

stores.shape

(1112, 5)

**Joining the "Train" and "Store" tables**

**Question 5**



# YOUR CODE

# inner join `stores\_encode` and `train\_dataset\_encode` on `Store`, and save the resulting df as `df\_combined`.

df\_combined = pd.merge(train\_dataset, stores, on=['Store', 'Store'])

Check that the combined df has no missing values and that it has the proper one-hot encoded values



df\_combined.isnull().sum()

Store 0

DayOfWeek 0

Date 0

Sales 0

Customers 0

Promo 0

StateHoliday 0

SchoolHoliday 0

StoreType 0

Assortment 0

CompetitionDistance 0

Promo2 0

dtype: int64

**New Feature Engineering on df\_combined**

* Extract month from Date



df\_combined['Month'] = pd.DatetimeIndex(df\_combined['Date']).month

**EDA**

To determine what type of models and predictors might work best for predicting sales, we studied which factors are causing the most variance in sales.

**Variations based on Store Attributes**

One of the biggest sources of variance in sales is based on the store number. In the graph below we show the average sales for the first ten stores in the data set. We can see that the average sales are quite different for each store and that there is no linear pattern. We do see stores with similar sales levels, suggesting the stores may fall into groups.

The scatterplot below that shows the average sales for all the stores. We can again see that some stores have similar levels of sales, so it's likely we can group stores together when making sales predictions, which is something for which tree models are well suited.

Each store number is effectively a category with its own particular sales level. When predicting future sales, we would like a single model that can make predictions for each store rather than having to fit a separate model for each store.



df\_subset = df\_combined[(df\_combined['Store'] < 11)]

sns.set(font\_scale=2)

​

# Plot store sales for stores 1 to 10

fig, ax = plt.subplots(1,1)

​

p3 = sns.barplot(x='Store', y='Sales', data=df\_subset, ax=ax)

ax.set(xlabel='Store Number')

ax.set\_ylabel('Average Sales')

ax.set\_title('Sales for ten stores')

​

plt.show()

​

# Create a plot of average sales per store id

avg\_sales\_per\_store = df\_combined[['Sales', 'Store']].groupby('Store').mean()

avg\_sales\_per\_store.reset\_index().plot(kind='scatter', x='Store', y='Sales')

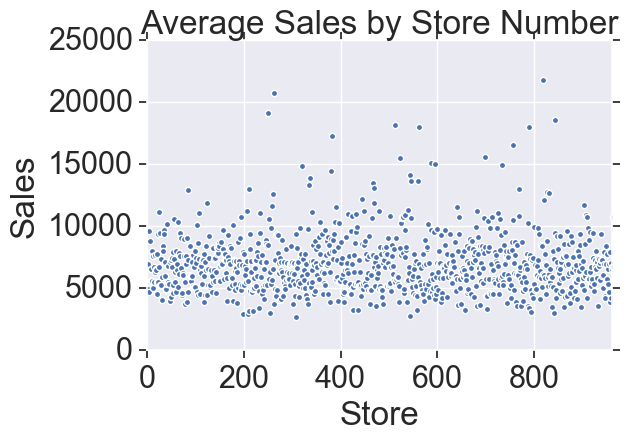
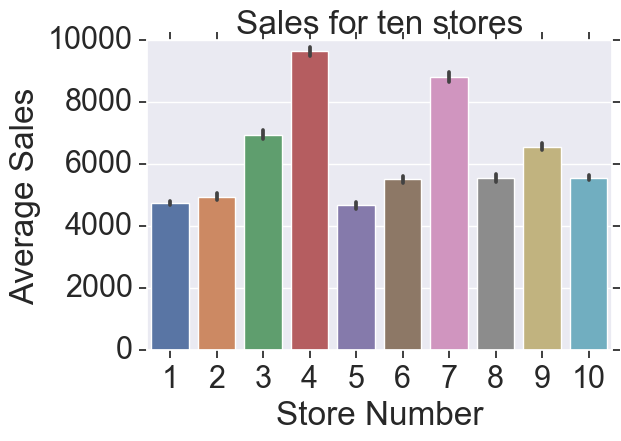
plt.xlim(0,960)

plt.title("Average Sales by Store Number")

plt.show()

C:\Users\Owner\AppData\Roaming\Python\Python35\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.

return np.add.reduce(sorted[indexer] \* weights, axis=axis) / sumval

The data set has categorical variables for different store types and the assortment of products for sale at each store. We see that these categories are a source of variance in sales.

**Impact of Number of Customers**

After exploration of the train and test sets, we ascertained that only the training data contains the number of Customers feature. The visualization below shows the positive correlation between the number of customers and sales (and also outliers). This feature is highly correlated with sales, but it’s not available until after the sales occur (i.e. it’s not in the test set). But exploring this feature helps make a case for including a proxy for the Customers feature in our final predictors (see the Feature Engineering section).

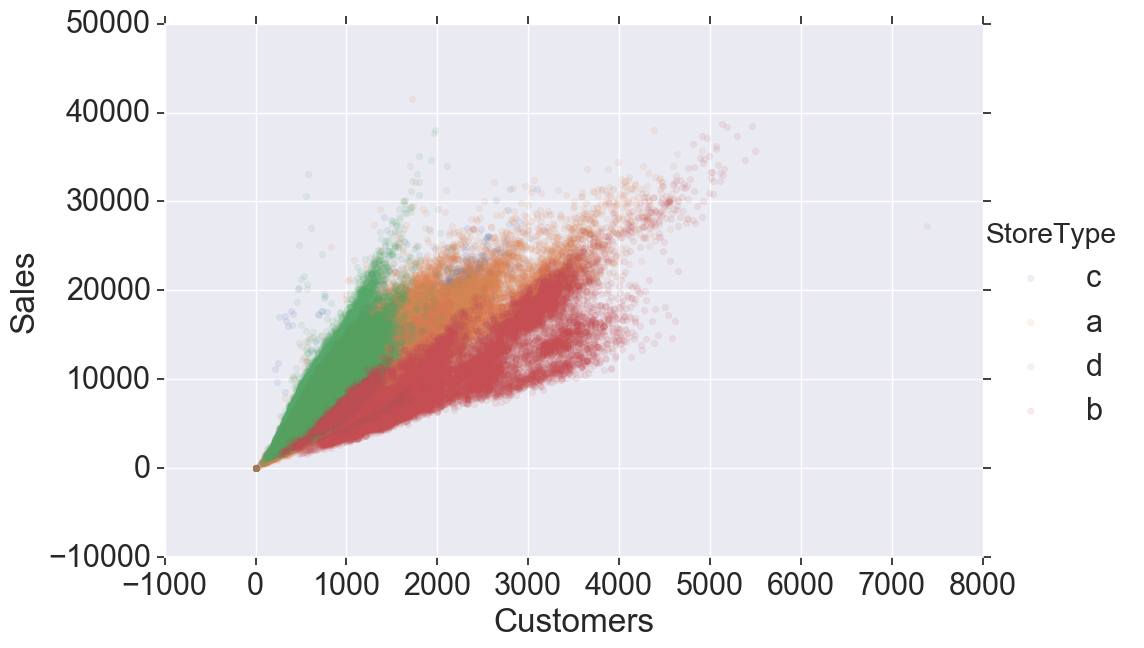
We also see that there’s added information with the behavior of customers per StoreType. We see that StoreType d (shown in green) that projects to the upper left quadrant. This means that for the same level of Sales, StoreType d requires fewer customers than StoreType b (blue). There’s less clarity of StoreType impact in the middle of the cone shaped scatter, but we can certainly see the impact of Customers and Sales conditional on the StoreType mostly in the outer regions of the visualization. Thus StoreType is a relevant feature, but as a function of Customers, as we can see that around 2000 to 3000 Customers there’s a break in the cluster pattern of the middle section of the cone. Therefore, further substantiation for us to include a proxy of the Customers feature in the model.



sns.lmplot(x='Customers', y='Sales', data=df\_combined, hue='StoreType',fit\_reg=False,

scatter\_kws={'alpha':0.1}, height=7, aspect=1.5)

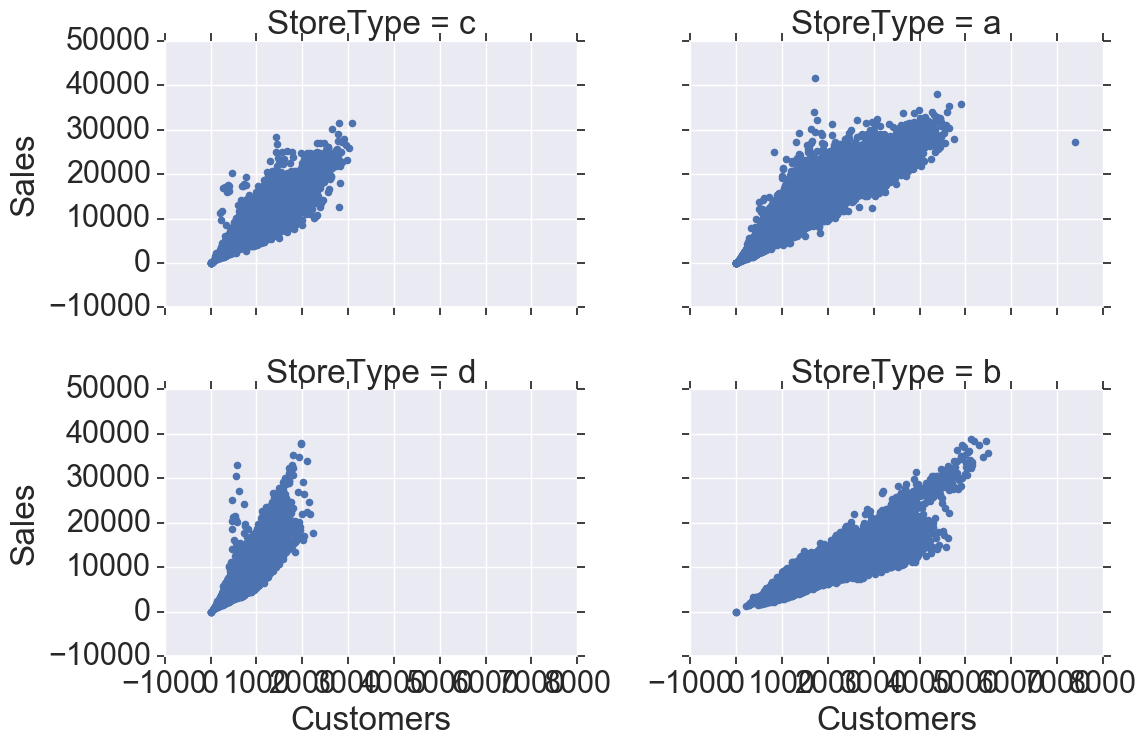
<seaborn.axisgrid.FacetGrid at 0x22150607ac8>



g = sns.FacetGrid(df\_combined, col="StoreType", col\_wrap=2, height=4, aspect=1.5)

g.map(plt.scatter, "Customers", "Sales")

<seaborn.axisgrid.FacetGrid at 0x22151357828>

**Process categorical columns**

Before we can start modeling, we need to convert categorical variables into one-hot encoded binary variables. In df\_combined, one-hot encode the following:

* Store
* StoreType
* Assortment
* StateHoliday

Notice that we have two zero values for StateHoliday: one integer and one string. Let us fix this and have one string zero value for all StateHoliday representations.



# Display unique values of 'StateHoliday'

df\_combined.StateHoliday.unique()

array(['0', 'a', 'b', 'c'], dtype=object)

**Question 6**



# YOUR CODE

# Your result shoud look like this:

### array(['0', 'a', 'b', 'c'], dtype=object)

df\_combined.StateHoliday.unique()

​

​

array(['0', 'a', 'b', 'c'], dtype=object)

**Question 7**



# YOUR CODE

# Transforme the variables into one-hot-encoding using the get\_dummies() method

​

df\_combined = pd.get\_dummies(df\_combined, columns=['Store', 'StateHoliday', 'StoreType', 'Assortment'])

**Rossmann Data Modeling**

Due to the regressive predictive nature of The Rossmann Project, we decided to approach our model methodology with a Linear Regression base model with feature selection. Note: you can also consider Ridge regression to evaluate the effects of regularization on the predictive performance.

*Note: The time dimension has a tremendous impact in this project. The Kaggle competition consisted in predicted the next 6 weeks (or 42 days), our splitting lead to have a testing set very large of over 900 days, explaining the fast degradation of the explained variance. These issues are the same the world of finance and economics are facing. For example the usage of an AR(n) to predict future outcomes quickly converge to a stationary states. In a real world, one would adjust the model based on the observed error. This is the spirit of tools such as the 'Kalman Filter' that are usually implemented in these domains.*

*One other potential approach would consist in implementing more complicated model that naturally combine time series and decision tree, such as ART (Autoregressive Trees). The proposed method are outside of the scope of this course and thus we will focus on some other tehniques here.*

**Linear Regression**

Linear regression is an intuitive first baseline model.

**Load libraries**



from sklearn import linear\_model

from sklearn.metrics import r2\_score

from sklearn.model\_selection import train\_test\_split



df\_combined.columns

Index(['DayOfWeek', 'Date', 'Sales', 'Customers', 'Promo', 'SchoolHoliday',

'CompetitionDistance', 'Promo2', 'Month', 'Store\_1',

...

'StateHoliday\_a', 'StateHoliday\_b', 'StateHoliday\_c', 'StoreType\_a',

'StoreType\_b', 'StoreType\_c', 'StoreType\_d', 'Assortment\_a',

'Assortment\_b', 'Assortment\_c'],

dtype='object', length=1132)

Keep features for modeling:



store\_feats = [c for c in df\_combined if "Store\_" in c] # columns related to one-hot encoded `Store`

store\_type\_feats = [c for c in df\_combined if "StoreType\_" in c] # columns related to one-hot encoded `StoreType\_`

assortment\_feats = [c for c in df\_combined if "Assortment\_" in c] # columns related to one-hot encoded `Assortment\_`

state\_holiday\_feats = [c for c in df\_combined if "StateHoliday\_" in c] # columns related to one-hot encoded `StateHoliday\_`

extra\_feats = ['CompetitionDistance', 'Promo2', 'Promo',

'DayOfWeek', 'Month',

'SchoolHoliday']

features = store\_feats + store\_type\_feats + assortment\_feats + state\_holiday\_feats + extra\_feats

​

target = 'Sales'

**Linear Regression**

This takes a bit of time because we have a lot of predictor columns.



# Split the data into train/test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_combined[features], df\_combined[target], test\_size=0.33)

​

# Create linear regression object

regr = linear\_model.LinearRegression()

​

# Train the model using the training sets

regr.fit(X\_train, y\_train)

​

# Make predictions using the testing set

y\_pred = regr.predict(X\_test)

​

# Explained variance score: 1 is perfect prediction

print('Train R2 score: %.2f' % r2\_score(y\_train, regr.predict(X\_train)))

print('Test R2 score: %.2f' % r2\_score(y\_test, y\_pred))

Train R2 score: 0.75

Test R2 score: 0.75



# Plot outputs

plt.figure(figsize=(20, 5))

plt.scatter(y\_test, y\_pred, color='black')

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=4, color='blue')

​

plt.ylabel('Predicted')

plt.xlabel('Measured')

​

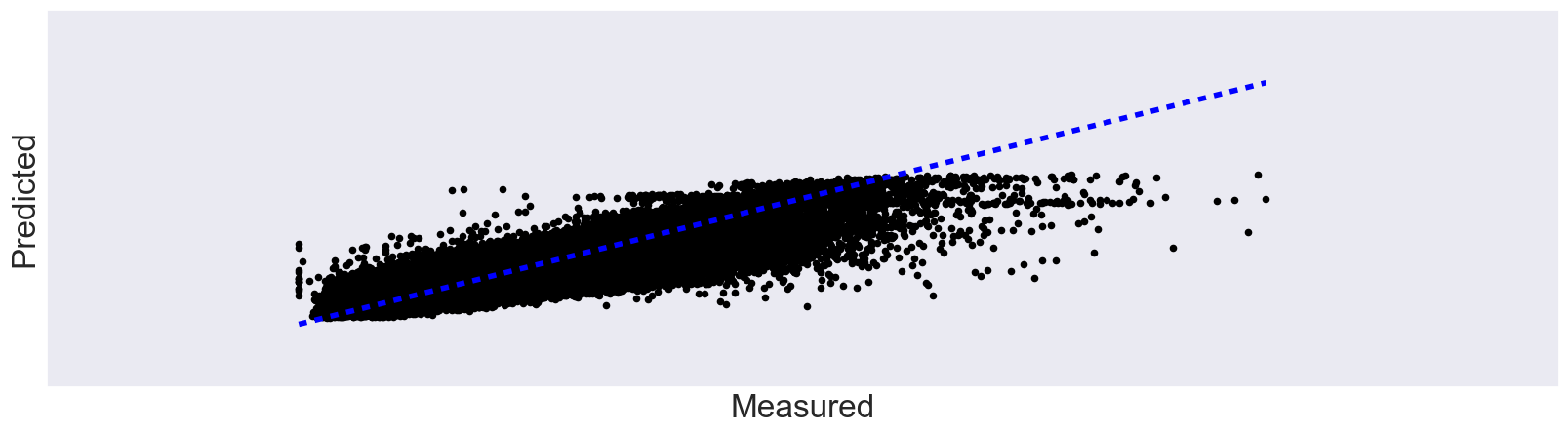
plt.xticks(())

plt.yticks(())

​

plt.show()

​

**Question 8**

Plot the residual plot



# YOUR CODE

# Plot residual plot: residuals vs. fitted values

# Residuals = y\_true - y\_pred

# Fitted values = y\_pred

# Don't forget to label your plot

​

plt.figure(figsize=(10, 7))

resid = y\_test - y\_pred

​

plt.axhline(y=0, linestyle='-', linewidth=2, color="r")

plt.scatter(x=y\_pred, y=resid, alpha=0.5, s=3)

​

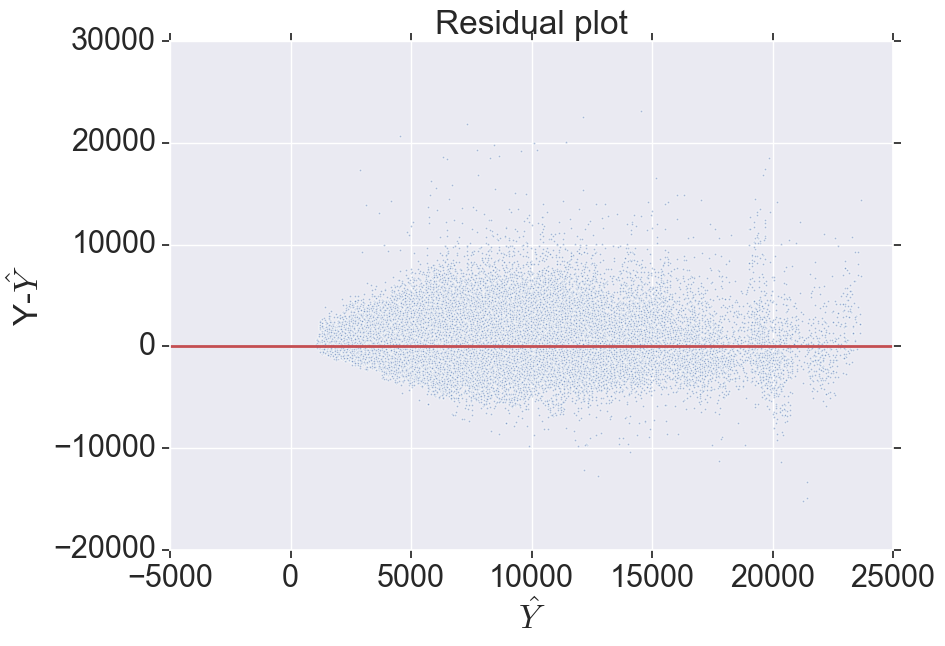
plt.title("Residual plot")

plt.ylabel(r"Y-$\hat Y$")

plt.xlabel(r'$\hat Y$')

plt.tight\_layout()

plt.show()

**Question 9**

Can you reason about the residual plot?

**Your Answer:**



Most of the residual datapoints are reltively close to zero. When working with such large values as home prices, being within 10,000 of the actual data is very good. The residual plot shows that almost all of the predicted values varied by less than 10,000 of the actual value. Especially with an r^2 value of 0.75, this shows that this is a pretty good model. Not as close as we would like, and there are a good number of outliers that vary by as much as 22,000.

**Tree Based Models**

It has been well established that bagging and other resampling techniques can be used to reduce the variance in model predictions. As several replicates of the original data set are created using random selection with replacement, at every step, each derivative data set is then used to construct a new model and the models are gathered together into an ensemble. To make a prediction, all of the models in the ensemble are polled and their results are averaged in the case of regression, which is our specific approach.

As well, it has been well established that a powerful modeling algorithm that makes good use of bagging is Random Forests, which works by training numerous decision trees each based on a different resampling of the original training data. The random forest algorithm improves on bagging by training each tree on a random sample of the available features, to prevent each tree from choosing the same predictors.

In Random Forests the bias of the full model is equivalent to the bias of a single decision tree, which itself has high variance. By creating many of these trees, a forest, and then averaging them, the variance of the final model can be greatly reduced over that of a single tree. In practice the only limitation we encountered on the size of the forest is computing time, as an infinite number of trees could be trained without ever increasing bias and with a continual - if asymptotically declining - decrease in the variance.

It is for the aforementioned that we considered Random Forests as a Baseline Ensemble and its constituent, the Decision Tree Regressor as a baseline model as well.

**Decision Trees**

Decision trees were implemented as to evaluate the base consituent of the Random Forest ensemble. As per our research on previous models, Random Forest is a very good performance candidate for a baseline ensemble model for Rossmann.

Decision Trees are also evaluated as a non-parametric baseline model, which enriches the comparative analysis of the Linear Regression Models.

For fitting the decision tree model, we do not need to create dummy variables for categorical columns, since trees are able to make use of the factorized category values we created during data cleaning.

**Random Forest**

Next we considered Random Forest models.



# Load libraries

from sklearn.ensemble import RandomForestRegressor

C:\Users\Owner\Anaconda3\lib\site-packages\sklearn\ensemble\weight\_boosting.py:29: DeprecationWarning: numpy.core.umath\_tests is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.

from numpy.core.umath\_tests import inner1d



rf = RandomForestRegressor(random\_state=10)

rf.fit(X\_train, y\_train)

​

y\_pred = rf.predict(X\_test)

print('R2 score on train: ', rf.score(X\_train, y\_train))

print('R2 score on test: ', rf.score(X\_test, y\_test))

R2 score on train: 0.9235171131775901

R2 score on test: 0.8513609170080871

**Question 10**

Plot the residual plot



# YOUR CODE

# Plot residual plot: residuals vs. fitted values

# Residuals = y\_true - y\_pred

# Fitted values = y\_pred

# Don't forget to label your plot

​

plt.figure(figsize=(10, 7))

resid = y\_test - y\_pred

​

plt.axhline(y=0, linestyle='-', linewidth=2, color="r")

plt.scatter(x=y\_pred, y=resid, alpha=0.5, s=3)

​

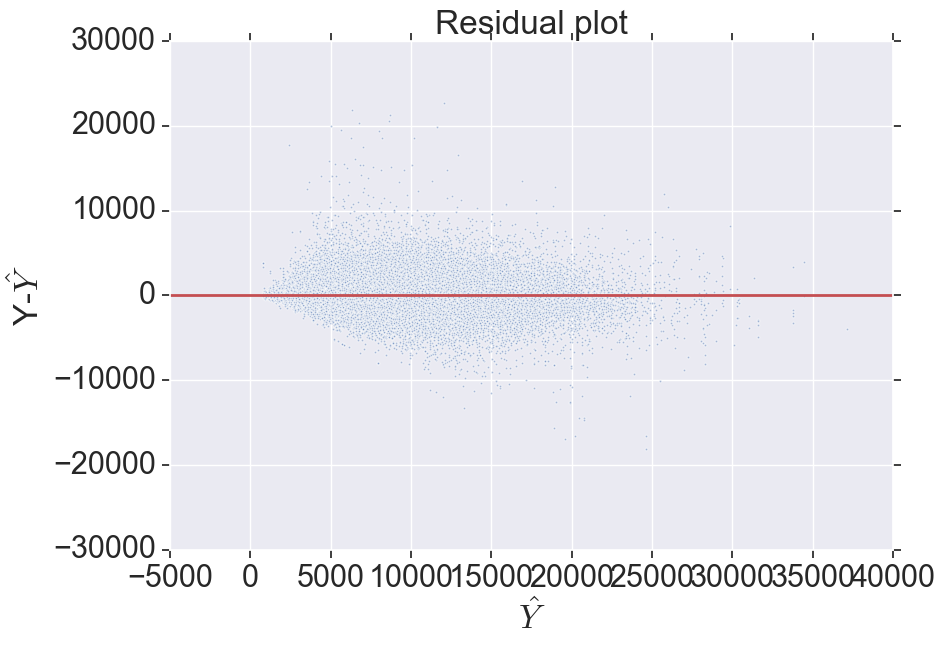
plt.title("Residual plot")

plt.ylabel(r"Y-$\hat Y$")

plt.xlabel(r'$\hat Y$')

plt.tight\_layout()

plt.show()

**Question 11**

**Which model is better? Give 1 possible approach to improve your best model. Your answer:**



The Random Forest Regression model is better. It has a much better r^2 score. One way to improve the model is to change which features are used to make the train and test parameters.

**Export models**

In our final step, we will choose the best model and explore some important prediction elements. Our app accepts features and gives predictions based on users inputs. Let us see what kind of data input is neccessarry (this is important for you to setup the proper input on the Flask side).

For example, take one simple observation (we are using first data row):



# Print first row

X\_test.iloc[0]

Store\_1 0.0

Store\_2 0.0

Store\_3 0.0

Store\_4 0.0

Store\_5 0.0

Store\_6 0.0

Store\_7 0.0

Store\_8 0.0

Store\_9 0.0

Store\_10 0.0

Store\_11 0.0

Store\_12 0.0

Store\_13 0.0

Store\_14 0.0

Store\_15 0.0

Store\_16 0.0

Store\_17 0.0

Store\_18 0.0

Store\_19 0.0

Store\_20 0.0

Store\_21 0.0

Store\_22 0.0

Store\_23 0.0

Store\_24 0.0

Store\_25 0.0

Store\_26 0.0

Store\_27 0.0

Store\_28 0.0

Store\_29 0.0

Store\_30 0.0

...

Store\_1103 0.0

Store\_1104 0.0

Store\_1105 0.0

Store\_1106 0.0

Store\_1107 0.0

Store\_1108 0.0

Store\_1109 0.0

Store\_1110 0.0

Store\_1111 0.0

Store\_1112 0.0

Store\_1113 0.0

Store\_1114 0.0

Store\_1115 0.0

StoreType\_a 1.0

StoreType\_b 0.0

StoreType\_c 0.0

StoreType\_d 0.0

Assortment\_a 1.0

Assortment\_b 0.0

Assortment\_c 0.0

StateHoliday\_0 1.0

StateHoliday\_a 0.0

StateHoliday\_b 0.0

StateHoliday\_c 0.0

CompetitionDistance 18760.0

Promo2 1.0

Promo 0.0

DayOfWeek 2.0

Month 1.0

SchoolHoliday 0.0

Name: 518125, dtype: float64

Predict based on your first row:



# Test for your app

rf.predict(np.array(X\_test.iloc[0].reshape(1, -1)))

array([2609.25])

Therefore, your input needs to be **numpy array** with the same column order as df\_combined. Now, we can export pur model and get ready for the second part of the homework.



from sklearn.externals import joblib

joblib.dump(rf, 'rm.pkl')

['rm.pkl']



​