About Dataset

The Ames Housing dataset was compiled by Dean De Cock for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often cited Boston Housing dataset. The goal is to predict the house sale price based on 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa.

There is also a competition at https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview about this dataset and advanced regression techniques.

A full description of each variable can be found here: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

About Hypothesis

My hypothesis is about testing one variable's ('GrLivArea') effect on 'SalePrice'.

"All else equal, a house with bigger "Above grade (ground) living area square feet" will have higher sale price".

I will test my hypothesis after doing the following steps:

1. Analyzing the target variable
2. Analyzing the other variables
3. Handling the missing data
4. Detecting and fixing the outliers

5. Transforming the data into desired shape
6. Applying a Machine Learning algorithm to predict ‘SalePrice’

Analyzing the target variable ‘Sale Price’

Sale price is the reason of our work. In-Depth Exploratory analysis is needed to proceed.

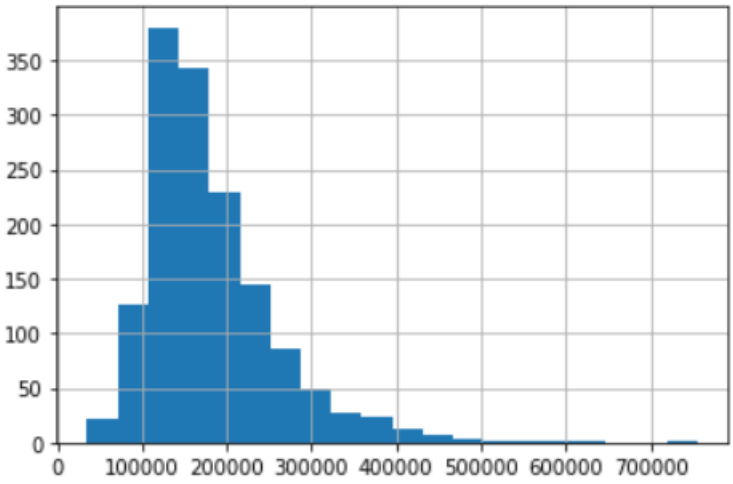
```
In [5]: # Analysing the target variable sale price
        df['SalePrice'].describe()

Out[5]: count      1460.000000
        mean     180921.195890
        std       79442.502883
        min       34900.000000
        25%      129975.000000
        50%      163000.000000
        75%      214000.000000
        max      755000.000000
        Name: SalePrice, dtype: float64
```

It is good that the minimum value of Sale Price is 34900. We are looking at a meaningful variable. If the minimum value was 0, we would have need to inspect further. 0 values could be in issue for our models.

In [6]:
```
# Checking the distribution of sale price
df['SalePrice'].hist(bins=20)
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x291ce8b09e8>
```

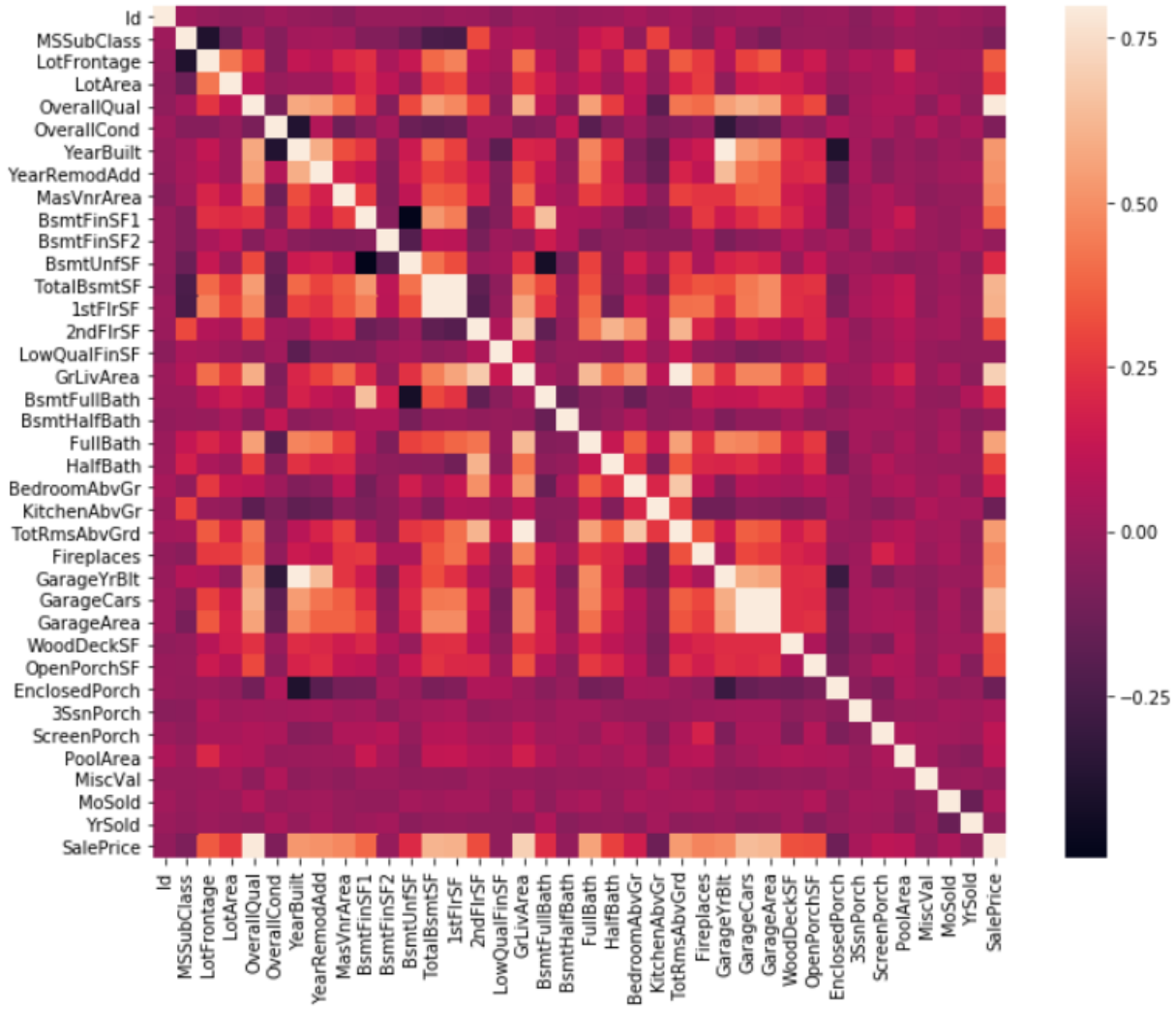


In [7]:
```
print("Skewness: %f" % df['SalePrice'].skew())
print("Kurtosis: %f" % df['SalePrice'].kurt())
```

```
Skewness: 1.882876
Kurtosis: 6.536282
```

There is a positive skewness. We can apply log transformation in the future to normalize our data.

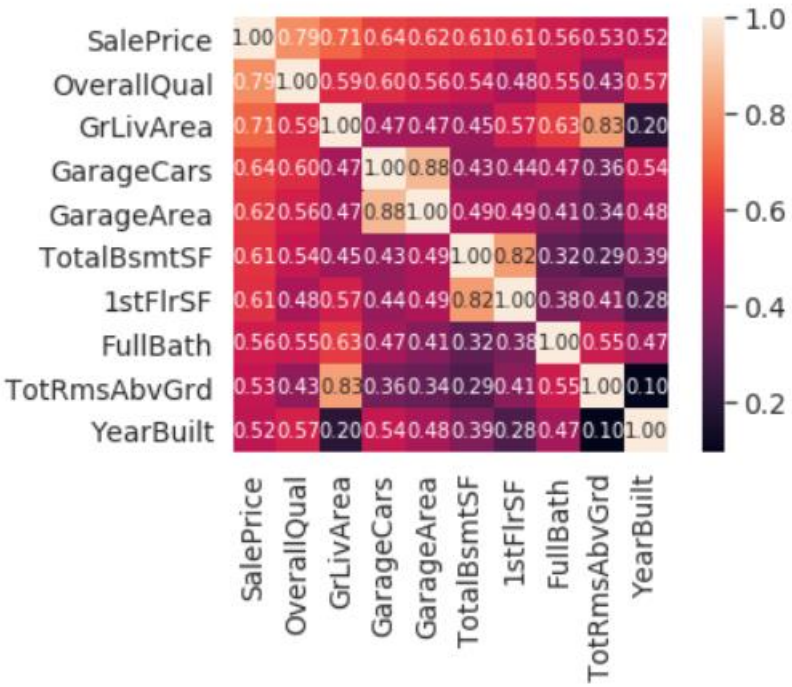
‘SalePrice’ and Other Variables



At first sight, there are two red colored squares that gets attention. The first one refers to the 'TotalBsmtSF' and '1stFlrSF' variables, and the second one refers to the 'GarageX' variables. Both cases show how significant the correlation is between these variables. Actually, this correlation is so strong that it can indicate a situation of multicollinearity. If we think about these variables, we can conclude that they give almost the same information so multicollinearity really occurs. Heatmaps are great to detect this kind of situations and in problems dominated by feature selection, like ours, they are an essential tool.

Another thing that gets attention was the 'SalePrice' correlations. We can see our 'GrLivArea', 'TotalBsmtSF', and 'OverallQual' are highly correlated with 'SalePrice', but we can also see many other variables that should be taken into account. That's what we will do next.

Top 10 correlated variables with 'Sale Price'



'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'

'GarageCars' and 'GarageArea' are highly correlated and gives very similar information. We can eliminate one of them. I choose to keep 'GarageCars' since its correlation is higher with 'SalePrice'

We can eliminate also '1stFlr' and 'TotRmsAbvGrd' for the same reasoning.

Handling Missing Data

Looking at the missing data percentages of variables

Out[11]:

	Total	Percent
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageCond	81	0.055479
GarageType	81	0.055479
GarageYrBlt	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtCond	37	0.025342
BsmtQual	37	0.025342
BsmtFinType1	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685
LotConfig	0	0.000000

Ideally we want to delete the variables that have missing data unless they contain unique and/or important information.

When we look at the top 4 variables 80+ % of the data is missing. No reason to impute those variables. The other variables are neither unimportant nor strongly correlated with 'OverallQual', 'YearBuilt' or 'GarageCars'. We only keep 'Electrical' variable after deleting that 1 observation.

Outlier Detection with Standard Deviation

If a value has a distance to the average higher than *3 * standard deviation***,** I assumed it as an outlier and deleted it. I also imputed the missing values with the median of the column to not lose more information.

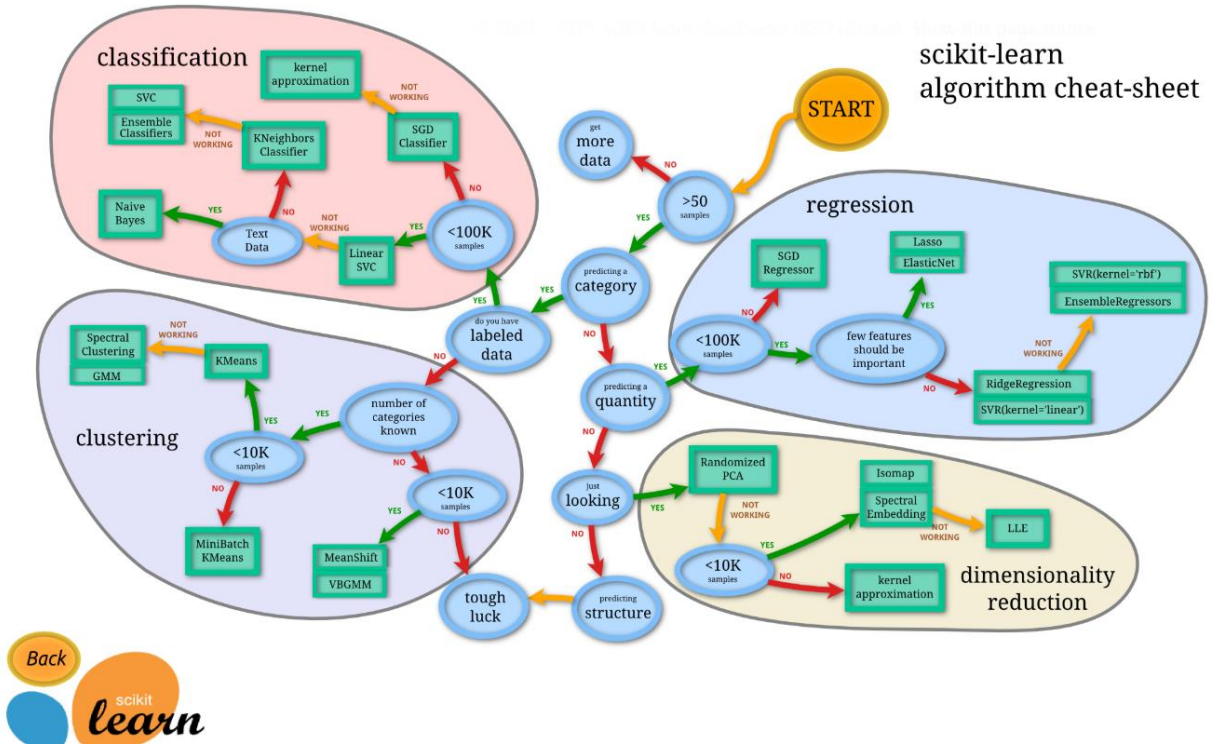
Transforming the Data

I applied log transformation on all numerical columns since most of them were skewed. Log transformation makes a the data 'normal' which its mean is 0 and its standard deviation is 0.

After that I scaled the data using scikitlearn's standard scaler. Scaling is a process which we fit the data in to the range [0,1]. The reason for doing this to equalize the effect of different quantitative variables. A variable can lie between the range of 1 to 10 where another can 1 to 1000. Their effect on the model would not be equal irrespective to their real significance. By scaling we negate this situation.

We use pandas 'get dummies' function to change categorical values into numerical ones.

Choosing Estimators



According to scikit-learn's estimator decision tree, the most suitable model for our dataset is either Lasso or Ridge regression. We'll try them both and continue with the better.

Ridge

In Ridge Regression, the OLS loss function is augmented in such a way that we not only minimize the sum of squared residuals but also penalize the size of parameter estimates, in order to shrink them towards zero:

$$L_{ridge}(\hat{\beta}) = \sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j-1}^{m}\hat{\beta}_j^2 = ||y - X\hat{\beta}||^2 + \lambda||\hat{\beta}||^2.$$

The λ parameter is the regularization penalty.

We will use scikit's GridSearchCv to find the best λ parameter

GridSearchCv is a perfect tool for hyperparameter tuning. It does cross validation tests with all the possible parameters we choose and gives the best result.

```
In [27]: parameter_candidates = [{'alpha':[1,0.1,0.001]},
                                 {'fit_intercept':[True,False]}]
```

```
# Create a classifier object with the classifier and parameter candidates
clf_r = GridSearchCV(estimator=Ridge(), param_grid=parameter_candidates, n_jobs=-1,cv=7)
```

Best alpha parameter is 1 among parameter candidates.

```
In [34]: clf_r.best_params_
Out[34]: {'alpha': 1}
```

Ridge Regression Score:

```
In [35]: clf_r.best_score_
Out[35]: 0.8280687185198042
```

Lasso

Lasso, or Least Absolute Shrinkage and Selection Operator, is quite similar conceptually to ridge regression. It also adds a penalty for non-zero coefficients, but unlike ridge regression which

penalizes sum of squared coefficients (the so-called L2 penalty), lasso penalizes the sum of their absolute values (L1 penalty). As a result, for high values of λ, many coefficients are exactly zeroed under lasso, which is never the case in ridge regression.

```
In [27]:
parameter_candidates = [{'alpha':[1,0.1,0.001]},
                        {'fit_intercept':[True,False]}]
```

```
In [30]: # Getting the best parameter
         clf.best_params_
Out[30]: {'alpha': 0.001}

In [31]: # Score of our model
         clf.best_score_
Out[31]: 0.8379581935179095
```

As we can see Lasso Regression gives a slightly better result than Ridge. We will go with the Lasso.

Results

I sampled 1 random observation from the data and created its duplicate. I changed the 'GrLivArea' columns by 1 standard deviation away from their values in the opposite directions. When I fit this (2, 125) dataframe into my LassoRegression model, I got the following result:

Out[47]:

	GrLivArea	SalePriceTransformed
0	-0.086624	0.038707
1	-2.087310	-0.736417

In conclusion, my hypothesis is true. Lower 'GrLivAre' resulted lower 'SalePrice'.

Note: The sale price values in the pictures are after the log and standardizaton transformations.