PREDICTING HOUSE PRICES WITH RANDOM FOREST

Ask a home buyer to describe their dream house, and they probably will not begin with the height of the basement ceiling or the proximity to an east-west railroad. This project's dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence.

At the end of this project, I am to build a model that answers the question: "what features influence the SalePrice? What features are available in the dataset? Etc." using the data provided.

With 79 explanatory variables describing (almost) every aspect of residential homes, I'm to also predict the final price of each home in the test data based on a trained model from the train data using advanced regression techniques like random forest and gradient boosting.

File descriptions

- train.csv - the training set
- test.csv - the test set
- data_description.txt - full description of each column, originally prepared by Dean De Cock but lightly edited to match the column names used here
- sample_submission.csv - a benchmark submission from a linear regression on year and month of sale, lot square footage, and number of bedrooms

Data fields

Here is a brief version of what you will find in the data description file.

- SalePrice - the property's sale price in dollars. This is the target variable that you are trying to predict.
- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access
- Alley: Type of alley access
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- LotConfig: Lot configuration
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to main road or railroad
- Condition2: Proximity to main road or railroad (if a second is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Overall material and finish quality

- OverallCond: Overall condition rating
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date
- RoofStyle: Type of roof
- RoofMatl: Roof material
- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Exterior material quality
- ExterCond: Present condition of the material on the exterior
- Foundation: Type of foundation
- BsmtQual: Height of the basement
- BsmtCond: General condition of the basement
- BsmtExposure: Walkout or garden level basement walls
- BsmtFinType1: Quality of basement finished area
- BsmtFinSF1: Type 1 finished square feet
- BsmtFinType2: Quality of second finished area (if present)
- BsmtFinSF2: Type 2 finished square feet
- BsmtUnfSF: Unfinished square feet of basement area
- TotalBsmtSF: Total square feet of basement area
- Heating: Type of heating
- HeatingQC: Heating quality and condition
- CentralAir: Central air conditioning
- Electrical: Electrical system
- 1stFlrSF: First Floor square feet
- 2ndFlrSF: Second floor square feet
- LowQualFinSF: Low quality finished square feet (all floors)
- GrLivArea: Above grade (ground) living area square feet
- BsmtFullBath: Basement full bathrooms
- BsmtHalfBath: Basement half bathrooms
- FullBath: Full bathrooms above grade
- HalfBath: Half baths above grade
- BedroomAbvGr: Number of bedrooms above basement level
- Kitchen: Number of kitchens
- KitchenQual: Kitchen quality
- TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Functional: Home functionality rating
- Fireplaces: Number of fireplaces
- FireplaceQu: Fireplace quality
- GarageType: Garage location
- GarageYrBlt: Year garage was built
- GarageFinish: Interior finish of the garage

- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition
- PavedDrive: Paved driveway
- WoodDeckSF: Wood deck area in square feet
- OpenPorchSF: Open porch area in square feet
- EnclosedPorch: Enclosed porch area in square feet
- 3SsnPorch: Three season porch area in square feet
- ScreenPorch: Screen porch area in square feet
- PoolArea: Pool area in square feet
- PoolQC: Pool quality
- Fence: Fence quality
- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: $Value of miscellaneous feature
- MoSold: Month Sold
- YrSold: Year Sold
- SaleType: Type of sale
- SaleCondition: Condition of sale

METHOD

To understand the data at hand, an excel sheet was created containing the **Variable Name**, the **Type of Variable**, the **Segment** the Variable belongs to (e.g. building, space or location), our **Expectations** about the influence the Variable has on the dependent variable, **Conclusions** and any other **comments**. (File name: Understanding the Data.xlsx)

Now that there has been some understanding of the data, Machine Learning; Ensemble Learning was used to access our expectations if they were right or otherwise and to create the best model for the data at hand.

Libraries Used:

- Pandas
- Numpy
- Matplotlib: pyplot
- Seaborn
- Sklearn.preprocessing: Standard Scaler
- Sklearn.model_selection: train_test_split
- Sklearn.ensemble: Random Forest Regressor
- Sklearn.metrics: Mean Absolute Error

Acquire Data

The Python Pandas packages helps us work with our datasets. We start by acquiring the training datasets into Pandas Data Frames.

```
In [2]: data_path = (r'C:\Users\samha\Documents\Samuel_ZroNet\Me\python\Jupyter files\House Pricing Project/train.csv')
        house_prices = pd.read_csv(data_path)
```

Analyze by describing data

Pandas also helps describe the datasets which could be referenced from the excel created earlier in this project. Which features are available in the dataset?

```
In [3]: house_prices.columns
```

```
Out[3]: Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
               'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
               'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
               'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
               'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
               'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
               'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
               'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
               'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
               'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
               'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
               'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
               'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
               'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
               'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
               'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
               'SaleCondition', 'SalePrice'],
              dtype='object')
```

```
In [4]: house_prices.describe()
```

Out[4]:

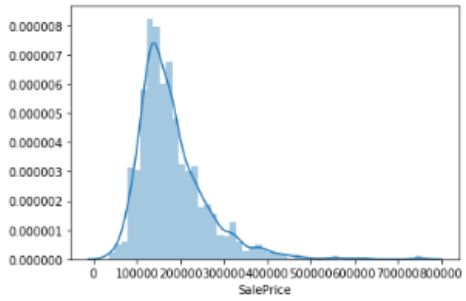
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDe
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	...	1460.00
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	...	94.24
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	...	125.33
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	...	0.00
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	...	0.00
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	...	0.00
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	...	168.00
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	...	857.00

8 rows × 38 columns

With just a simple line of code, we now know the respective means and count to all fields and columns involved. Moving on, a lot more investigations were carried out on the target feature: SalePrice.

```
In [6]: #plotting the SalesPrice
        sns.distplot(house_prices['SalePrice']);

        #not normally distributed
        #postively skewed
        #shows peakness
```




```
In [7]: print(house_prices['SalePrice'].skew())
        print(house_prices['SalePrice'].kurt())
```

```
1.8828757597682133
6.536281860064534
```

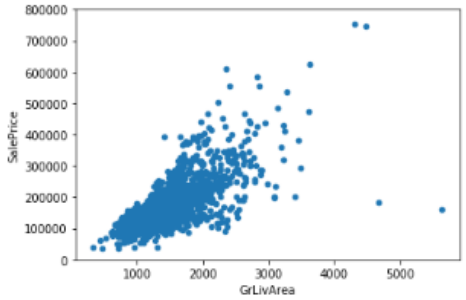
From this graph, we can say SalePrice is:

- Not normally distributed
- Positively skewed
- Shows peakness

Bivariate Analysis of the Target Variable and 'Expected Variables'

- **GrLivArea / SalePrice**

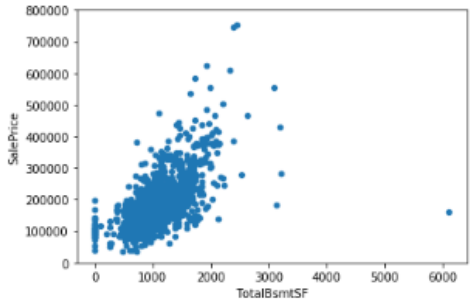
```
In [8]: #scatter plot grlivarea/saleprice
        data = pd.concat([house_prices['SalePrice'], house_prices['GrLivArea']], axis=1)
        data.plot.scatter(x='GrLivArea', y='SalePrice', ylim=(0,800000));
```



Analyzing the relationship between 'GrLivArea' and 'SalePrice', we can say they are positively correlated hence a high possible amount of influence of 'GrLivArea' on house prices.

- **TotalBsmtSF / SalePrice**

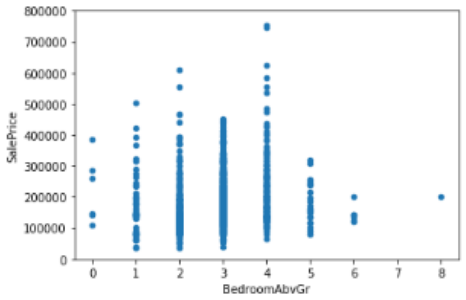
```
In [9]: #scatter plot TotalBsmtSF/saleprice
data = pd.concat([house_prices['SalePrice'], house_prices['TotalBsmtSF']], axis=1)
data.plot.scatter(x='TotalBsmtSF', y='SalePrice', ylim=(0,800000));
```



'TotalBsmtSF' and the Sale Prices are also positively correlated with a high tendency of influence in its prices. We can also see an outlier from this graph which shall be dealt with later.

- **BedroomAbvGr / SalePrice**

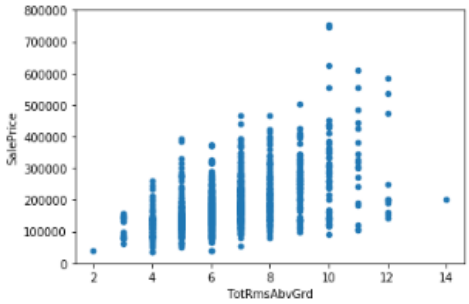
```
In [10]: #scatter plot BedroomAbvGr/saleprice
data = pd.concat([house_prices['SalePrice'], house_prices['BedroomAbvGr']], axis=1)
data.plot.scatter(x='BedroomAbvGr', y='SalePrice', ylim=(0,800000));
```



From this graph, 'BedroomAbvGr' is not positively correlated to the Sales Price although they correlate with an outlier.

- **TotRmsAbvGrd / SalePrice**

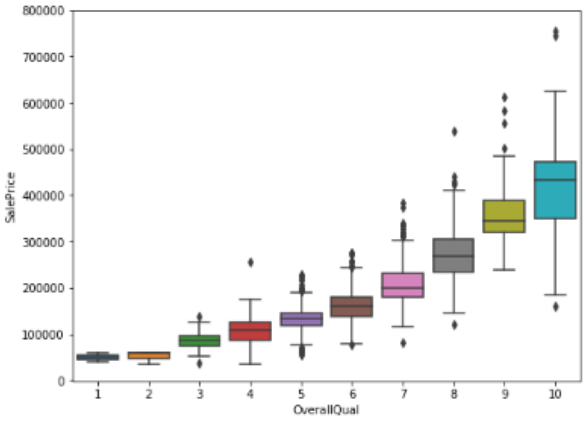
```
In [11]: #scatter plot TotRmsAbvGrd/saleprice
data = pd.concat([house_prices['SalePrice'], house_prices['TotRmsAbvGrd']], axis=1)
data.plot.scatter(x='TotRmsAbvGrd', y='SalePrice', ylim=(0,800000));
```



From this graph, ‘TotRmsAbvGrd’ is not so positively correlated with ‘SalePrice’, a proof would be carried out to support this claim.

- **OverallQual / SalePrice**

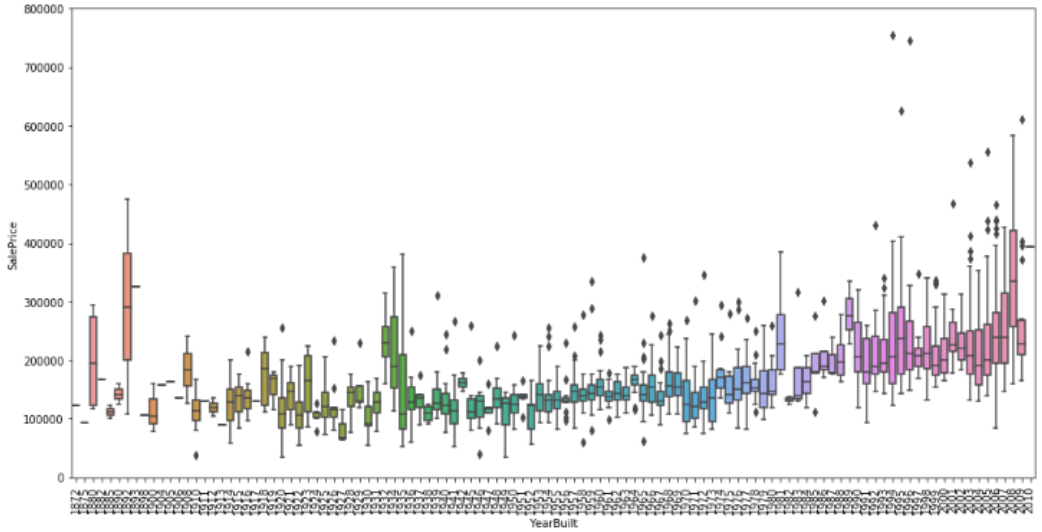
```
In [13]: #box plot OverallQual/saleprice
         var = 'OverallQual'
         data = pd.concat([house_prices['SalePrice'], house_prices[var]], axis=1)
         f, ax = plt.subplots(figsize=(8, 6))
         fig = sns.boxplot(x=var, y='SalePrice', data=data)
         fig.axis(ylim=0, ymax=800000);
```



Since ‘OverallQual’ is a categorical data, a box plot was used to determine the relationship it shares with ‘SalePrice’. Looking at how positively slopped the graph is, we can say ‘OverallQual’ is positively correlated to ‘SalePrice’.

- **YearBuilt / SalePrice**

```
In [14]: #box plot YearBuilt/saleprice
         var = 'YearBuilt'
         data = pd.concat([house_prices['SalePrice'], house_prices[var]], axis=1)
         f, ax = plt.subplots(figsize=(16, 8))
         fig = sns.boxplot(x=var, y='SalePrice', data=data)
         fig.axis(ylim=0, ymax=800000)
         plt.xticks(rotation=90);
```

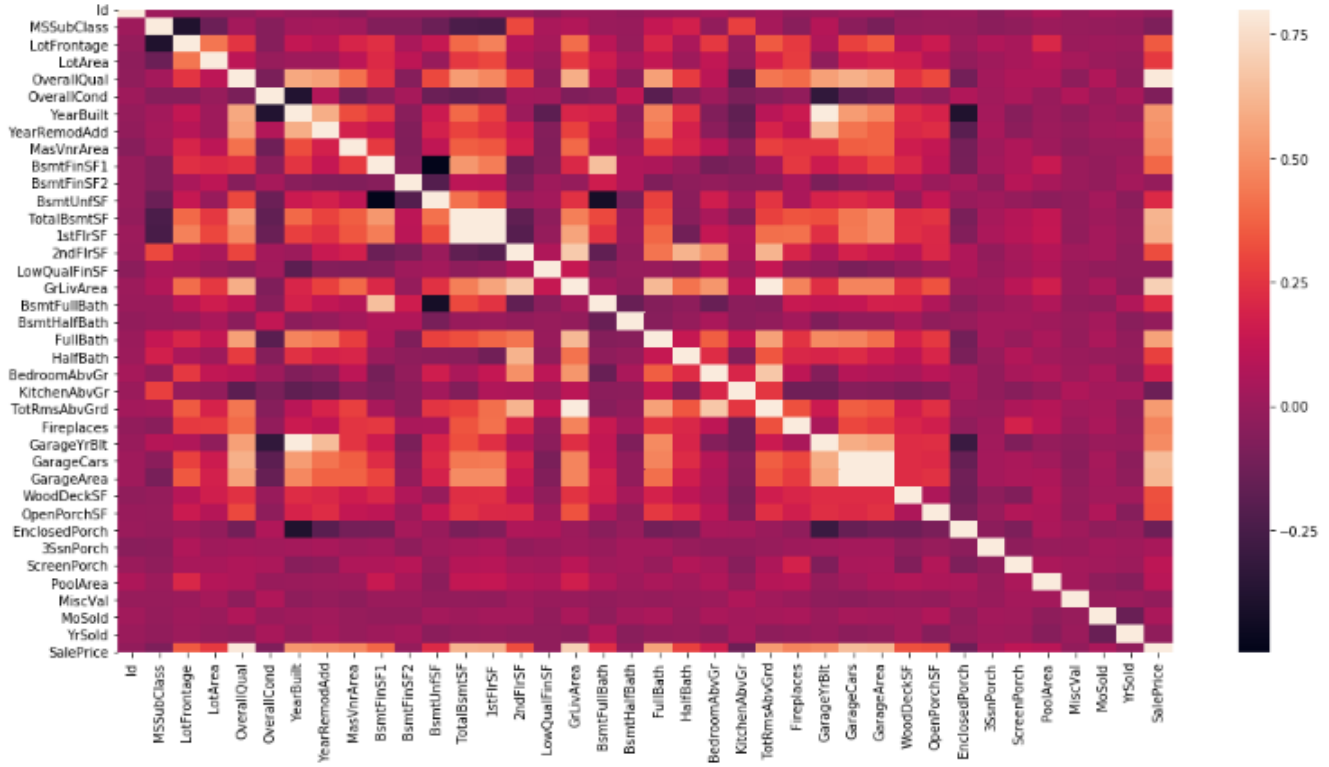


Looking at the harmonic behavior of this graph, comments were left out till we carried out another analysis which looked closer into the relationships the fields shared with ‘SalePrice’.

Correlation Matrix

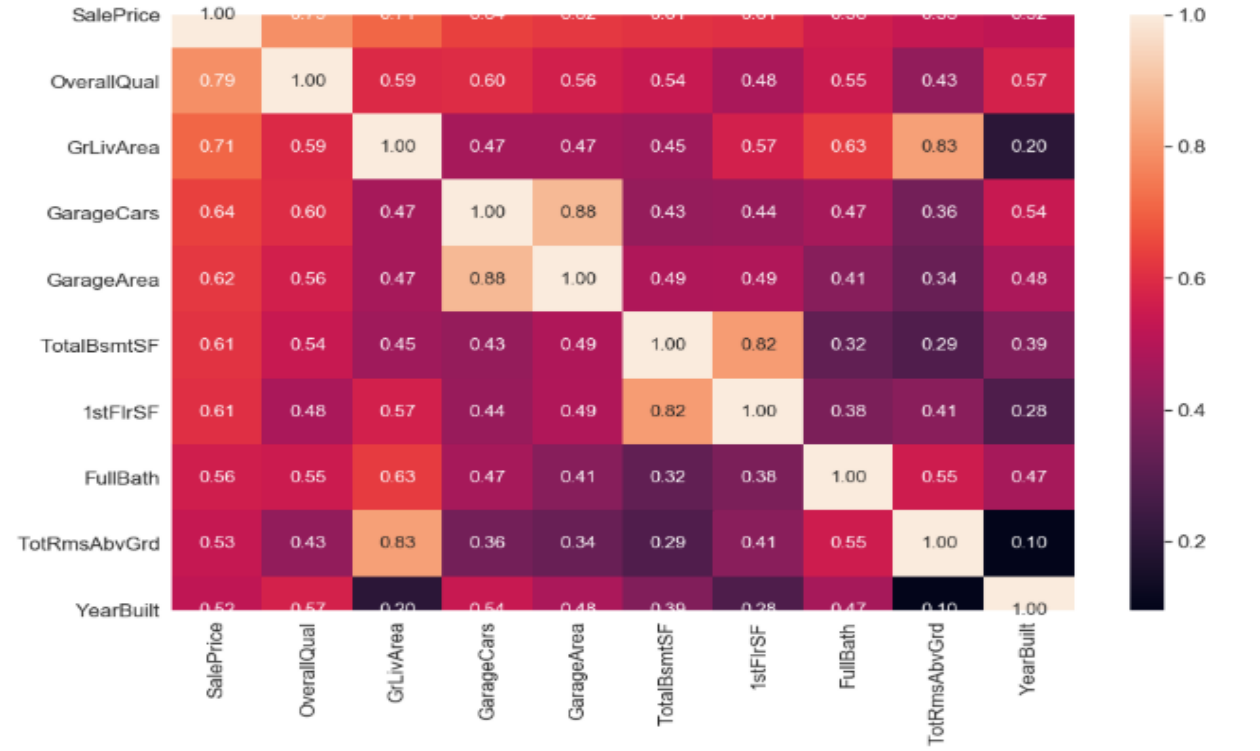
A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. Below is the graphical representation of the correlation matrix between the variables.

```
In [15]: #Correlation Matrix
         corrmat = house_prices.corr()
         f, ax = plt.subplots(figsize=(18, 9))
         sns.heatmap(corrmat, vmax=.8, square=False);
```



Taking a closer look:

```
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(house_prices[cols].values.T)
sns.set(font_scale=1.5)
plt.subplots(figsize=(16,10))
hm = sns.heatmap(cm, cbar=True, annot=True, square=False, fmt='.2f', annot_kws={'size': 15}, yticklabels=cols.values, xticklabel
plt.show();
```



From taking a closer look at the correlation matrix, we can say the top 9 correlated fields to 'SalePrice' are (accordingly):

- OverallQual - 79%
- GrLivArea - 71%
- GarageCars - 64%
- GarageArea - 62%
- TotalBsmtSF - 61%
- 1stFlrSF - 61%
- FullBath - 56%
- TotRmsAbvGrd - 53%
- YearBuilt - 52%

Selecting the Featured Variables

From the correlation matrix, a total of n number of featured variables can be selected. In this project, n=6.

- 'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'.
- 'GarageCars' and 'GarageArea' are also some of the most strongly correlated variables. However, the number of cars that fit into the garage is a consequence of the garage area. 'GarageCars' and 'GarageArea' are highly correlated which can cause multicollinearity. Therefore, we just need one of these variables in our analysis (we can keep 'GarageCars' since its correlation with 'SalePrice' is higher).
- 'TotalBsmtSF' and '1stFloor' also seem to be highly correlated. We can keep 'TotalBsmtSF' just to say that our first guess was right.
- 'FullBath' was not highly expected to be featured but the graph says different.
- 'TotRmsAbvGrd' and 'GrLivArea', are highly correlated again so we can keep ‘GrLivArea’.
- ‘YearBuilt' is slightly correlated as expected.

Hence n = ('OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'YearBuilt').

Missing data

Important questions when thinking about missing data:

- How prevalent is the missing data?
- Is missing data random or does it have a pattern?

The answer to these questions is important for practical reasons because missing data can imply a reduction of the sample size. This can prevent us from proceeding with the analysis. Moreover, from a substantive perspective, we need to ensure that the missing data process is not biased and hiding an inconvenient truth.

In [19]:
```
total = house_prices.isnull().sum().sort_values(ascending=False)
percent = (house_prices.isnull().sum()/house_prices.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percnet'])
missing_data.head(20)
```

Out[19]:

	Total	Percnet
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
BsmtExposure	38	0.026027
BsmtFinType2	38	0.026027
BsmtFinType1	37	0.025342
BsmtCond	37	0.025342
BsmtQual	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685
Utilities	0	0.000000

Let's analyze this to understand how to handle the missing data.

We'll consider that when more than 15% of the data is missing, we should delete the corresponding variable and pretend it never existed. This means that we will not try any trick to fill the missing data in these cases. According to this, there is a set of variables (e.g. 'PoolQC', 'MiscFeature', 'Alley', etc.) that we should delete. The point is: will we miss this data? I don't think so. None of these variables seem to be important, since most of them are not aspects in which we think about when buying a house (maybe that's the reason why data is missing?). Moreover, looking closer at the variables, we could say that variables like 'PoolQC', 'MiscFeature' and 'FireplaceQu' are strong candidates for outliers, so we'll be happy to delete them.

In what concerns the remaining cases, we can see that 'GarageX' variables have the same number of missing data. Since the most important information regarding garages is expressed by 'GarageCars' and considering that we are just talking about 5% of missing data, we'll delete the mentioned 'GarageX' variables. The same logic applies to 'BsmtX' variables.

Regarding 'MasVnrArea' and 'MasVnrType', we can consider that these variables are not essential. Furthermore, they have a strong correlation with 'YearBuilt' and 'OverallQual' which are already considered. Thus, we will not lose information if we delete 'MasVnrArea' and 'MasVnrType'.

Finally, we have one missing observation in 'Electrical'. Since it is just one observation, we'll delete this observation and keep the variable.

In summary, to handle missing data, we'll delete all the variables with missing data, except the variable 'Electrical'. In 'Electrical' we'll just delete the observation with missing data.

```
In [20]: #dealing with missing data
         house_prices = house_prices.drop((missing_data[missing_data['Total'] > 1]).index,1)
         house_prices = house_prices.drop(house_prices.loc[house_prices['Electrical'].isnull()].index)
         house_prices.isnull().sum().max() #just checking that there's no data missing...
Out[20]: 0
```

Out liars

Outliers is also something that we should be aware of. Why? Because outliers can markedly affect our models and can be a valuable source of information, providing us insights about specific behaviors.

Outliers is a complex subject and it deserves more attention. Here, we'll just do a quick analysis through the standard deviation of 'SalePrice' and a set of scatter plots.

Univariate analysis

The primary concern here is to establish a threshold that defines an observation as an outlier. To do so, we'll standardize the data. In this context, data standardization means converting data values to have mean of 0 and a standard deviation of 1.

In [21]:
```
from sklearn.preprocessing import StandardScaler
```

In [22]:
```
#standardizing data
#standardization means converting data values to have mean of 0 and a standard deviation of 1
saleprice_scaled = StandardScaler().fit_transform(house_prices['SalePrice'][:,np.newaxis]);
low_range = saleprice_scaled[saleprice_scaled[:,0].argsort()][:10]
high_range= saleprice_scaled[saleprice_scaled[:,0].argsort()][-10:]
print('outer range (low) of the distribution:')
print(low_range)
print('\nouter range (high) of the distribution:')
print(high_range)
```

```
outer range (low) of the distribution:
[[-1.83820775]
 [-1.83303414]
 [-1.80044422]
 [-1.78282123]
 [-1.77400974]
 [-1.62295562]
 [-1.6166617 ]
 [-1.58519209]
 [-1.58519209]
 [-1.57269236]]

outer range (high) of the distribution:
[[3.82758058]
 [4.0395221 ]
 [4.49473628]
 [4.70872962]
 [4.728631  ]
 [5.06034585]
 [5.42191907]
 [5.58987866]
 [7.10041987]
 [7.22629831]]
```

Analysis of Standardized 'SalePrice':

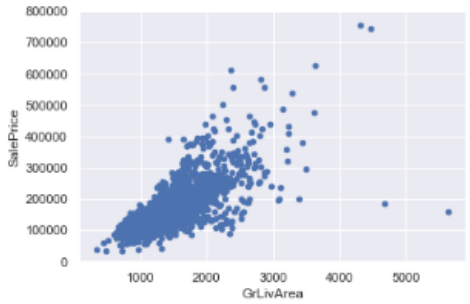
- Low range values are similar and not too far from 0.
- High range values are far from 0 and the 7.n (n=N) values are out of range.

For now, we'll not consider any of these values as an outlier but we should be careful with those two 7.n (n=N) values.

Scatter plots on features (specifically numerical variables) also revealed some outliers which were deleted.

In [23]:
```
#bivariate analysis saleprice/grlivarea
var = 'GrLivArea'
data = pd.concat([house_prices['SalePrice'], house_prices[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.
```



In [24]:
```
#deleting points
house_prices.sort_values(by = 'GrLivArea', ascending = False)[:2]
```

Out[24]:

	Id	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	EnclosedPorch	3SsnPorch	ScreenPorch	Poo
0	1299	60	RL	63887	Pave	IR3	Bnk	AllPub	Corner	Gtl	...	0	0	0	
1	524	60	RL	40094	Pave	IR1	Bnk	AllPub	Inside	Gtl	...	0	0	0	

2 rows × 63 columns

In [25]:
```
#deleting points
house_prices.sort_values(by = 'GrLivArea', ascending = False)[:2]
house_prices = house_prices.drop(house_prices[house_prices['Id'] == 1299].index)
house_prices = house_prices.drop(house_prices[house_prices['Id'] == 524].index)
```

Model, Predict and Solve

Now that we are ready to train our machine to predict, we can split our variables into target and featured variables. After this, we'd then split it with our "train_test_split" for validation and error checks.

In [33]:
```
y = house_prices.SalePrice
houseprices_features = ['OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'YearBuilt']
X = house_prices[houseprices_features]
```

In [34]:
```
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)
```

We can then create a Random Forest model then fit it with our data.

In [35]:
```
houseprice_model = RandomForestRegressor(criterion='mae', random_state=1)
```

In [36]:
```
houseprice_model.fit(train_X, train_y)
```

Out[36]:
```
RandomForestRegressor(bootstrap=True, criterion='mae', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=1, verbose=0,
                      warm_start=False)
```

Predicting ‘val_X’ for Mean Absolute error checks:

```
In [37]: preds = houseprice_model.predict(val_X)
         print(preds)
```

```
In [38]: print("mean absolute error: %f"  % mean_absolute_error(val_y, preds))
         mean absolute error: 21635.669041
```

Upon narrowing down the error value, we settled on 21635.669.

Predicting the Test Data

In predicting the test data, we first need to acquire the data

```
In [39]: test_path = (r'C:\Users\samha\Documents\Samuel_ZroNet\Me\python\Jupyter files\House Pricing Project\test.csv')
         test_data = pd.read_csv(test_path)
```

We then have to make known our featured variables so that the machine can use them in the prediction model. We can then check for missing values in our test data to prevent prediction errors.

```
In [43]: total = test_X.isnull().sum().sort_values(ascending=False)
         percent = (test_X.isnull().sum()/test_X.isnull().count()).sort_values(ascending=False)
         missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
         missing_data.head(20)
```

Out[43]:

	Total	Percent
TotalBsmtSF	1	0.000685
YearBuilt	0	0.000000
FullBath	0	0.000000
GarageCars	0	0.000000
GrLivArea	0	0.000000
OverallQual	0	0.000000

```
In [44]: #replacing the missing value in the test data with the mean
         test_X['TotalBsmtSF'].fillna((test_X['TotalBsmtSF'].mean()), inplace=True)
```

From out [43], we can see there is one missing value in ‘TotalBsmtSF’, now how can we deal with that? We can replace the missing value with the mean of the field (In [44]).
After this, nothing can hold us back from predicting our test values with the trained model.

```
In [46]: # make predictions which we will submit.
         test_preds = houseprice_model.predict(test_X)

         # The lines below shows how to save predictions in format used for competition scoring
         # Just uncomment them.

         output = pd.DataFrame({'Id': test_data.Id,
                                'SalePrice': test_preds})
         output.to_csv('submission1.csv', index=False)
```

Our predictions with then be saved in a csv excel format (submission1.csv).

Conclusion

On a scale of 0 – 10, where 0 is best and 10 is worst, our prediction had a Kaggle score of 0.17545 which is good. With fine tuning of the model and data, this score can be improved.
Github link: https://github.com/hagan420/predicting-prices-with-RFs