pythonを使った総合的なデータ探索

人生で一番難しいのは自分自身を知ることだ by タレス(ギリシャ時代の哲学者、数学者、宇宙学者、最初の哲学者とよばれる)

データを知ることはデーターサイエンスで最も難しいとは言わないけど時間がかかる。 だからデータをよく見もせずに分析を始めちゃうんだ。

だから僕はまずやる前にデータ分析のやり方を学ぼうとした。この本の「データを調べる」の章にそって、完璧じゃないけどざっくりとしたデータ分析にのめりこんだ。 だからこのカーネルをやったからといってすぐに一人前の研究ができるとは思っていないけど、はじめての人たちにとって役に立つと思うから僕がどうやってデータ分析しているかを共有するね。

変なタイトルつけちゃったけど、このカーネルでやることはこんな感じだ。

1. 問題を理解する　それぞれの変数をみて、この問題の意味や重要さについて哲学的に分析しよう
2. 単一変数の研究 従属変数である　'販売価格'にフォーカスして、ちょっと詳しくなろう。
3. 多変数の研究 従属変数や独立変数がどうやって関係し合うか理解しよう。
4. 基本のクリーニング 前処理するよ。 データの欠損や異常値、カテゴリー変数をうまく対処しよう。
5. 仮説のテスト　 多変量解析の技術をつかってデータを仮説にぶっこんで確認してみよう

じゃあ、楽しんでいこう！

In [2]:

```
# kaggleパーティに君を招待するね
# データ分析に必要なものをimportする
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm # 正規分布モジュール
from sklearn.preprocessing import StandardScaler # データを標準化するもの
from scipy import stats
 # 警告を制御する
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [3]:

```
# 6缶ケースを持ってくる
# データをジュースかお酒扱い?
df_train = pd.read_csv('./competitions/house-prices-advanced-regression-techniques/train.csv')
```

In [4]:

```
# 項目名がぱっと出せるよ
df_train.columns
```

Out[4]:

```
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')
```

1.で？何すればいいの？

データを理解するには、変数を一つ一つ見て、意味や問題との関係性を理解するんだ。 時間がかかるかもしれないけど、この作業でデータの感じを掴めるんだ。（おろそかにしちゃだめだよ！）

分析を規律をもたらすためにエクセルのスプレッドシートに下記の項目を用意してほしい。

- 変数名
- 型 数値型かカテゴリー型　数値は身長や体重みたいに数字で表せるもの、カテゴリーデータは血液型とか男性女性とかの分類のこと。
- 領域 データのセグメント（分類分け）。今回は「建物」、「スペース」、「場所」の３つに分類するよ。「建物」は家の物理的特性に関するもの（鉄筋or木造とか築年数）、「スペース」は家の内容に関するもの(ベットルームの数とか)、場所は家の立地に関するもの
- 予想:　販売価格への影響度合い。「High」、「Middle」、「Low」の３つに分ける。
- 結論: 最終的にどの変数が重要か。（家の売値に重要な要素はどれ？）あとでぱっと見わかるように。expectationと同じ分け方。
- コメント: コメント欄

型とセグメントは見ればわかるけど、大事なのはexpectationでこれが分かれば家の売値の予測がパッとできちゃうんだ。（知らない人からすれば第六感のようにね） それを埋めるために一つ一つ注意深く読んで下記のような質問を自分自身に聞くんだ。

- 自分だったら家を買うときにその変数について考えるか？（たとえば家を建てると夢見たときに家の材質が石工単板タイプかなんて考えるかい？（僕はどうでもいいともうけど）
- もし考えるとしたら、それはどれだけ大事でしょうか？()
- 他の変数で説明された情報はないか？（平屋ならスロープがあるかなんて考えなくてもいいでしょ？）

この地道な調査をみっちりやってシートを埋めたら、「予測」が"High"の部分に絞って、販売価格と散布図を作ってみましょう。 これで予想が正しかったかを「結論」に埋めていこう！

このプロセスをへて、この問題で下記の変数が重要と結論づけたよ。

- 全体的な素材と仕上がりの質
- 建てられた年
- 地下部屋の敷地面積
- 地上のリビングの敷地面積

結論として建物の特性として上の２変数、家の内容として下の２変数が大事だと分かったよ。 これはちょっと予想外でした。不動産業界の金言「大事なのは１に立地、２に立地、３に立地」とは真逆だからね。 このお手軽な調査はカテゴリーデータに対してちょっと厳しいかもね。 たとえば僕はご近所は大事だと思ったんだけど分析したら大事じゃないって分かったよ。 これはカテゴリーデータの可視化に適した箱ひげ図じゃなくて散布図を使ったからだね。 データの可視化の仕方しだいで結果に影響するんだよ。

でもこのレッスンで大事なことは**僕たちはデータやその関連度について少し立ち止まって考えた**ってこと、だからゴールにたどり着いんだ。 おしゃべりは終わりにして、早速やってみよう！

2.最初の最初？ 販売価格について分析してみよう

販売価格は分析の目的だ。パーティに出かけるときのようだ。パーティへいく理由は大抵"女性"だ。（"女性"の部分は自分の好みに置き換えてくれ）

女性のたとえをつかって、ちょっとした小話を作ってみよう。題して「どうやって販売価格に出会うか」。

kaggleパーティが始まって、ダンスパートナーを探している。ダンスフロアを探し回っていて、バーの近くにダンスシューズを履いた少女に出会った。 我々はモデル予測やデータ分析コンペをたくさんやってきたいわばプロだ。少女に話しかけるのなんてわけない。なん

「やあ、ぼくkaggler。あなたは？「販売価格」さん？いい名前だね。もっと情報をくれないかな？ぼくは誰と誰が仲良くなるかを確率的に計算するモデルを作りたいんだ。 そしてそれを応用したいんだ。

In [5]:

```
# describe()関数で「販売価格」さんの情報が見れるよ!
#descriptive statistics summary
# その項目の値についての概要情報を見れる
# count :データの個数
# mean: 平均値
# std: 標準偏差
# min: 最小値
# 25% 50% 75%: 下から1/4,/1/2,3/4のところにある数値
# max: 最大値
# Name: 項目名 dtype:型
df_train['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
```

In [6]:

```
df_train.head() # 先頭から5件のデータを表示する。データをざっと見ることができる基本コマンド
```

Out[6]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Ut
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	A
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	A
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	A
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	A
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	A

5 rows × 81 columns

In [ ]:

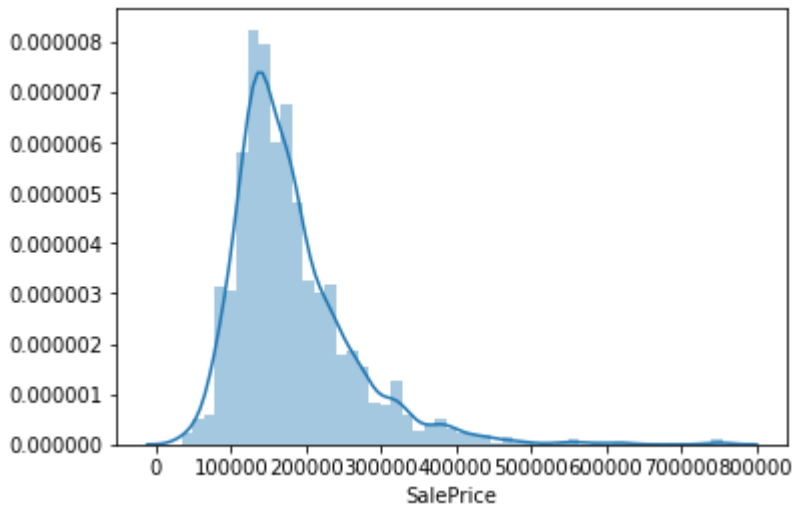
```
なるほど、最安値はゼロより大きいね。すばらしい、あなたは私のモデルをぶっ壊すかもしれない性質の一つを持って
なにか僕に見せられる写真はない?わからないけど、ビーチにいたときとか事務の写真とかさー。
```

In [29]:

```
# ヒストグラム 下記に引数のデフォルト値や解説あり
# https://seaborn.pydata.org/generated/seaborn.distplot.html
# ヒストグラムhist=Trueと確率密度関数kde=Trueとラグプロット(どこに値があるか軸上にぷろっとする)=False)
sns.distplot(df_train['SalePrice'])
```

Out[29]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a1e98f198>



ああ、君がちょっと席をはずしているときにいろいろのぞいちゃったんだ、すごいね！ 君は

- 正規分布からはずれているね
- 正の歪度だから左寄りだね
- 山のてっぺんでピークがわかるね

これは面白い。「販売価格」さん、もっといろいろ調べさせてよ！

In [8]:

```
#skewness and kurtosis
# 歪度 skewness 正規分布からどれだけ歪んでいるか 左寄りのとき正の値をとり、右寄りのとき負の値をtoru
# 尖度 kurtosis とがっているか 尖るほど値が大きくなる
# SalesPriceは左寄りで尖っていることがわかる
print("Skewness: %f" % df_train['SalePrice'].skew())
print("Kurtosis: %f" % df_train['SalePrice'].kurt())
```

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

すごいや！もし僕の計算が正しければ成功率は97%だよ。 きみとはまた会いたいよ。 これぼくの電話番号だから覚えていてよ。もし金曜暇ならかけてきてね！じゃあまたね！ ※なんて自分勝手なkggle野郎だ。

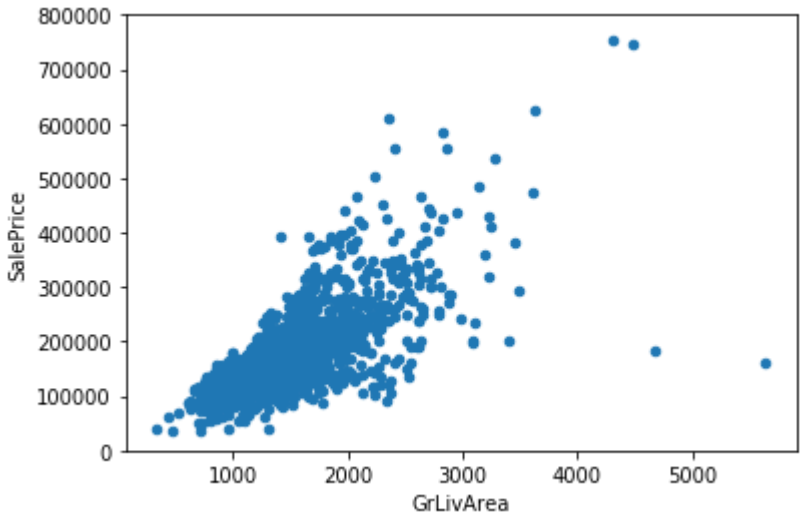
販売価格さんの仲間たちと興味

地形を選ぶのは戦術的知恵だ。まもなく「販売価格」さんが君の眼前に現れるだろう。心してかかれよ。 追っ手はいない。こわがらなくても個別の変数をガンガン調査するだけさ！ 我々の研究を最大限活用するために、私たちは共通の友人のプロフィールに注意深く目を通し、共通の興味に焦点をあてます。

数値データ間の関係

In [6]:

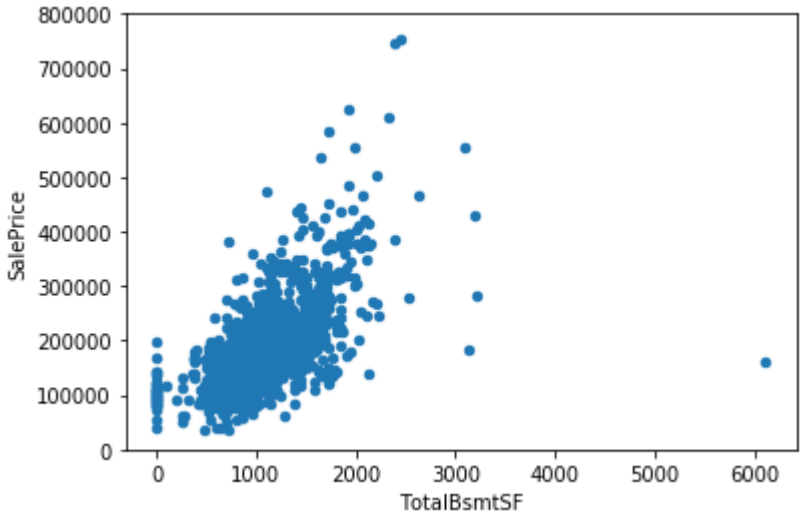
```
# grlivareaとsalepriceの関係を散布図にする
var = 'GrLivArea'
# 項目を横並びにならべる。axis=0だと縦に並べる
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
# yの範囲を0,800000にして散布図を表示
data.plot.scatter(x=var, y='SalePrice', ylim=(0, 800000));
```



うーん、「販売価格」さんと「地上リビングの敷地面積」さんは**線形関係をもつ**古くからの友人のようだ。「合計地下の敷地面積」君はどうだろう？

In [10]:

```
# totalbsmtsf/salepriceを散布図で表す
var = 'TotalBsmtSF'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0, 800000));
```

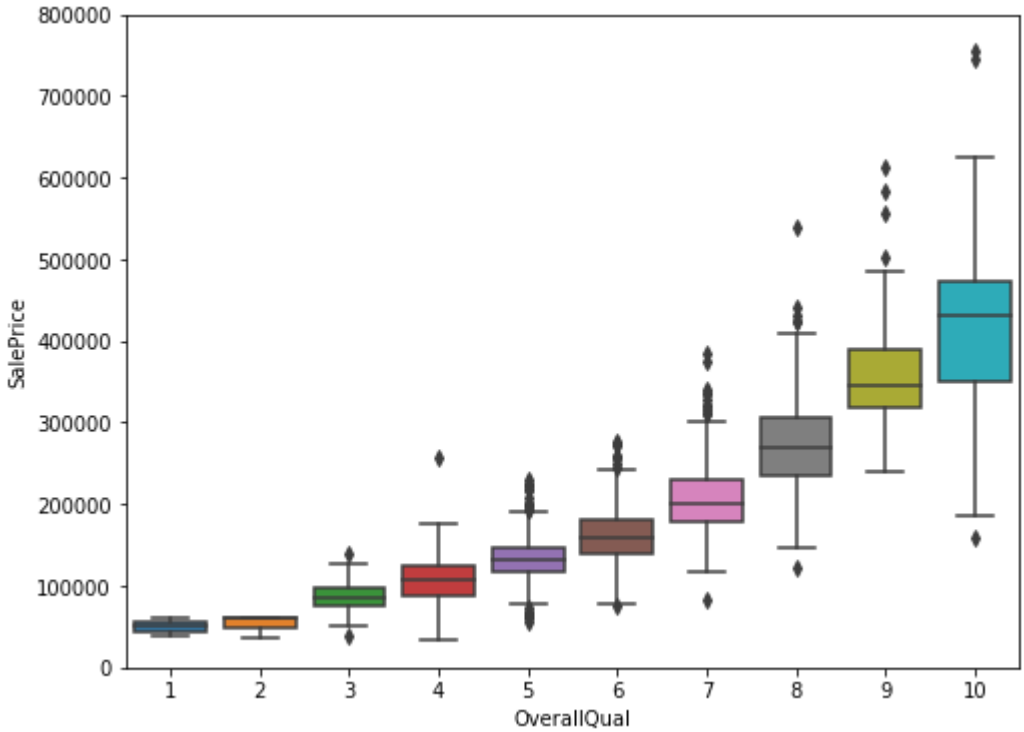


「TotalBsmtSF」君も「販売価格」さんとお友達みたい。しかしこれはもっと感情的な関係ね。 指数関数的な感じね。さらにときどき全然相関がなくなるみたいね。

カテゴリー的な特徴の関係

In [11]:

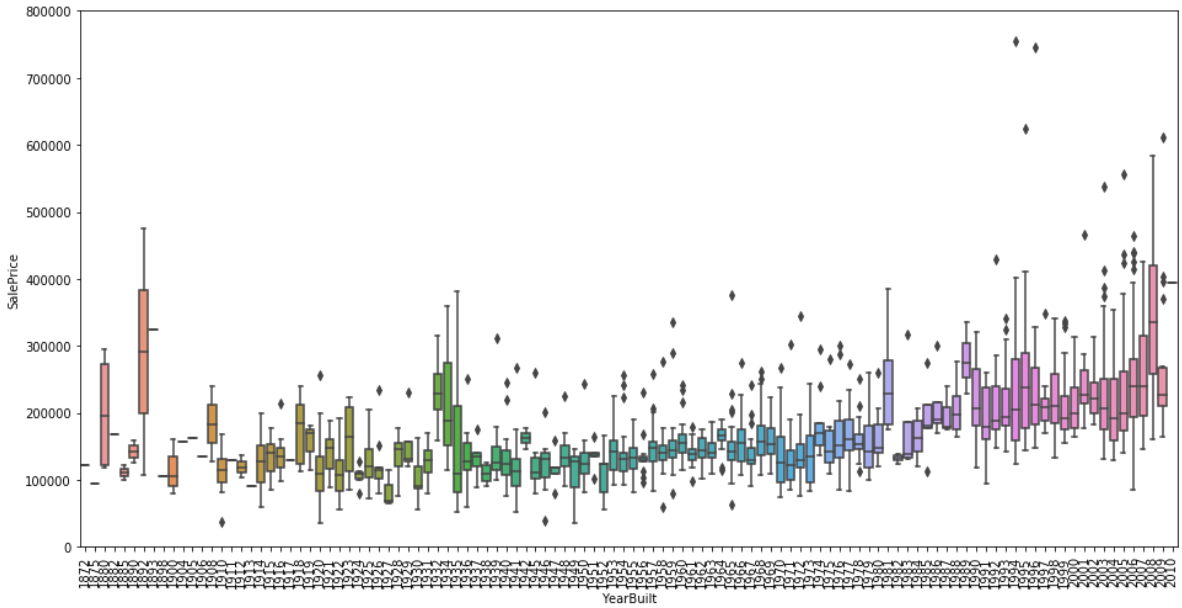
```
#ヒゲ箱プロット overallqualと販売価格
# 箱ひげ図の見方
# それぞれのカテゴリーデータをまとめて
# 上ヒゲが最大値、下ヒゲが最小値、箱の上が第三四分位数、真ん中が第二四分位数
# 下が第一四分位数
# 黒豆さんは何?(@@@調査中です@@@)
var = 'OverallQual'
data= pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6)) # 画面サイズを8:6 横:縦で表示
fig = sns.boxplot(x=var, y='SalePrice', data=data)
fig.axis(ymin=0, ymax=800000);

# ※f, ax = の部分は figure axisオブジェクトをアンパック代入
# たとえばfig.savefig('yourfilename.png')で保存できたり
# するので慣例的にこう書いておく(axの使い方は調べきれていない)
# 箱の上下にあるのは外れ値箱の1.5倍以上の大きさの点は外れ値とされる
# whisパラメータで調節でき whis='range'とすると外れ値にしない
```



かわいい少女のように「販売価格」さんは「全体的な素材や仕上がりの質」を楽しんでいる。 メモ：最初のデートにマクドナルドはどうかな？（だめでしょ？）

In [34]:

```
var = 'YearBuilt'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
f, ax = plt.subplots(figsize=(16,8)) #描画サイズのデフォルトは8:6
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
plt.xticks(rotation=90); # x軸に関する指定 ここでは年数が横に並べると重なって読めないので90度回転させ
```



そこまで強い傾向ではないですが、古い家より新しい家の方が高値で売れていますね。

注意：販売価格が実質価格か分かっていません。実質価格はインフレの影響を取り除きます。もし実質価格になっていない場合はそうすべきで、 それによって時系列での価格を正しく扱えます。

まとめ

話がそれましたが、結論づけると、

- 「地上のリビングの敷地面積」と「地下の敷地面積」は販売価格と比例関係にありそうです。正の相関があって、片方の変数の値が増えるともう片方も増えるといった具合です。TotalBsmtSFの場合は値が大きくなるとより販売価格が高くなっていくようですね。
- 「全体的な素材と仕上がりの質」と「建てられた年」もどちらも販売価格と関係があるようです。 「全体的な素材と仕上がりの質」の場合は特に強い関係性があり、箱ひげ図が販売価格が全体の質にそって増えることを示しています。

今まで４つ変数しか分析してこなかったけど、他の変数についてもやるべき分析がたくさんあります。 肝は正しい変数を選び出すこと（特徴選択）で、難しい関係性を定義すること（特徴量エンジニアリング）じゃないよ。

つまりたくさんの特徴から必要な特徴を選り分けることだ！

3.落ち着いてスマートにやろう!

これまでは直感にしたがって重要だと思った変数を分析してきました。 客観的に分析しているつもりでも、出発点は主観的だったといわざるをえません。 このアプローチはエンジニアとして癪に触るよね。 私から言えるのは規律を守って、こうなんじゃないかって主観を働かせないようにすることだ。 それには訳があります。 構

造工学では主観的にやろうとすると、法則がガラガラと崩れ去り痛い目を見るでしょう。

だから主観的になりたいという慣性に打ち勝ってぜひ客観的な分析を行うようにしてください。

プラズマスープ

宇宙の始まりの時、そこはプラズマスープだけだった。 宇宙を研究し始めたときはその事実は推測にすぎなかった。しかし、今日までの研究で科学は宇宙で何が起きたかを鮮やかに描ききっている。

宇宙を探索するには、プラズマスープを意味する実践的なレシピから始めるのがいいでしょう。

- 相関行列(ヒートマップ） *個々の値のデータ行列を色として表現した可視化グラフ
- 販売価格の相関行列（ズームしたヒートマップ）
- もっとも相関の高いものを散布図でプロットしろ（ジャガースタイルに移行せよ）

相関行列(ヒートマップ）

相関行列

それぞれの項目ごとの相関係数をもとに色付けした図。

相関係数

相関係数とは２種類のデータの関係を示す指標で、-1から1のあたいで示され、

- -1か1に近づくほど強い相関がある。逆に０はほぼ相関がない。
- 正の値のときは正の相関があるといい、どちらかの値が増加するともう一方の値が増加傾向
- 負の値のときは負の相関があるといい、どちらかの値が増加するともう一方の値が減少する傾向

相関係数rは $$共分散S_{xy}、 x項目の標準偏差S_x、y項目の標準偏差S_yとして$$

$$r = \frac{S_{xy}}{S_xS_y}$$ となる。

次に標準偏差は割愛して共分散について説明する。

共分散

２組の対応するデータ間の関係を表す数値である。 データのペア$(x_1,y_1)(x_2, y_2),\cdots,(x_n,y_n)$があったとして、

共分散S_{xy}は

$S_{xy} = \frac{1}{n}\sum_{i=1}^n (x_i-\overline{x})(y_i-\overline{y})$ で求められる。

共分散はnp.cov()で求められるが、中身の計算を理解するために一つ一つ計算してみる。

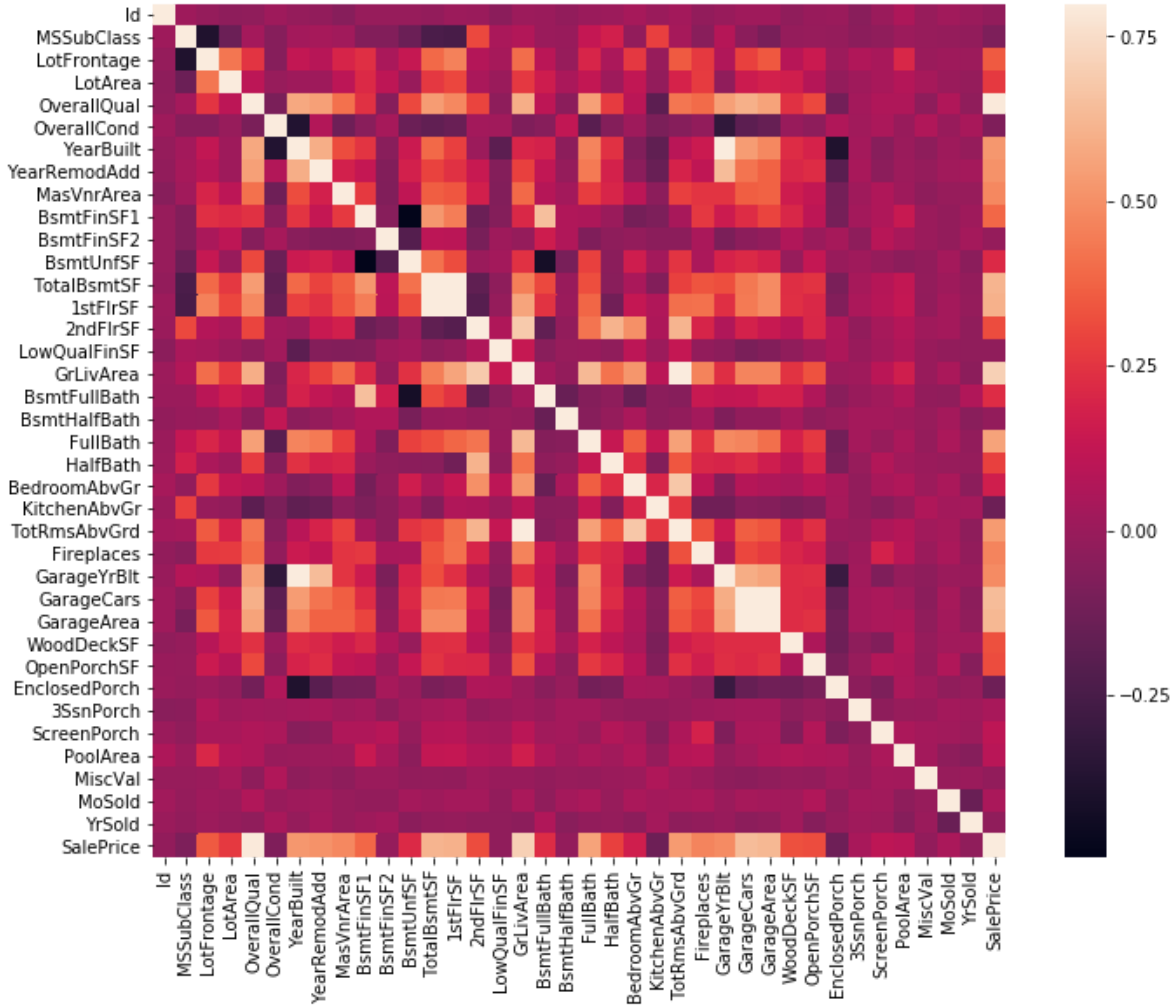
In [21]:

```
import math
# 試しにSalePriceとGrLivAreaの共分散と相関係数を求めてみる
col_SP = df_train['SalePrice']
col_GLA = df_train['GrLivArea']

mean_SP = np.mean(col_SP)
mean_GLA = np.mean(col_GLA)

std_SP = math.sqrt(sum([(n - mean_SP)**2 for n in col_SP])/len(col_SP))
# 計算した値とnp.stdが正しいか確認
print(std_SP, np.std(col_SP))
std_GLA = np.std(col_GLA)

cov = 0
for x,y in zip(col_SP, col_GLA):
    cov += (x-mean_SP)*(y-mean_GLA)

cov/= len(col_SP)

corr = cov / (std_SP*std_GLA)

print("共分散は{}、相関係数は{}".format(cov,corr))
```

```
79415.29188606751 79415.29188606751
共分散は29561605.190672714、相関係数は0.7086244776126512
```

In [14]:

```
# 相関行列
# 一般的に0.7以上でかなり強い正の相関があるとされる 白っぽいオレンジあたりから?
corrmat = df_train.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
# vmax=0.8にすると0.8より大きい値の色が固定になる。これでかなり強い相関はひと目。
# squareで各ブロックを正方形にする
```



私見だが、ヒートマップは全体やその関係をパッと俯瞰するのに最適なやり方だ。

一目で目を引くものが２つあります。'TotalBsmtSF'と'1stFlrSF'、 'GarageX'系の変数だ。 これらは重要な相関があるのがわかります。多重共線性（マルチコ）だとわかります。 これは同じ情報だとわかるので、それを削ったりするのに使える便利なものです。

http://xica.net/vno4ul5p/ (http://xica.net/vno4ul5p/) マルチコについては上記参照。 同じような項目を入れると起きる現象で基本的に項目を絞ることで対策できる。

他に販売価格の相関関係を見てみましょう。 今までよく見てきた「GrLivArea」「TotalBsmtSF」「OverallQual」が高い相関があることが見てとれます。 ほかにも取り上げるべき変数がいくつかあるので、それを次にやってみましょう。

In [15]:

```
col_check = corrmat.nlargest(10, 'SalePrice')['SalePrice'].index
```

In [16]:

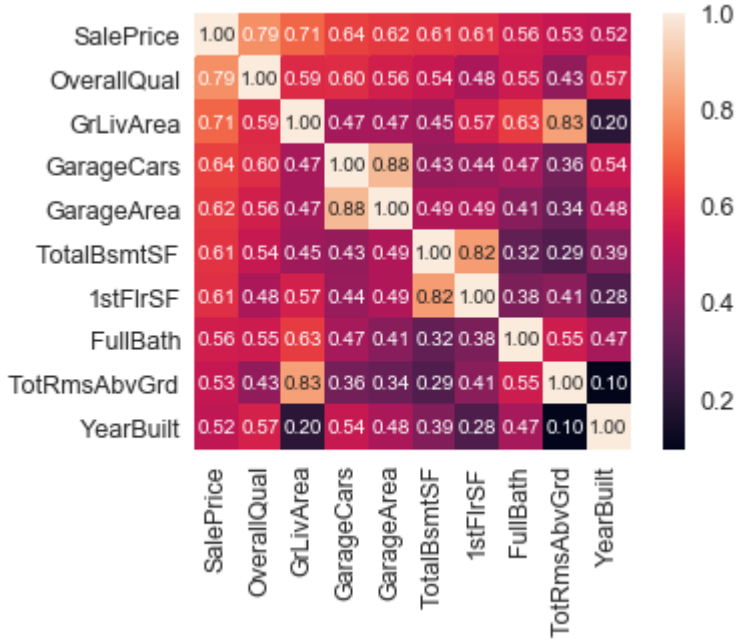
```
# NumPyのcorrcoefは行と行の相関係数を計算するため転置する必要あり
np.corrcoef(df_train[col_check].values.T)
```

Out[16]:

```
array([[1.        , 0.7909816 , 0.70862448, 0.6404092 , 0.62343144,
        0.61358055, 0.60585218, 0.56066376, 0.53372316, 0.52289733],
       [0.7909816 , 1.        , 0.59300743, 0.60067072, 0.56202176,
        0.5378085 , 0.47622383, 0.55059971, 0.42745234, 0.57232277],
       [0.70862448, 0.59300743, 1.        , 0.46724742, 0.46899748,
        0.4548682 , 0.56602397, 0.63001165, 0.82548937, 0.19900971],
       [0.6404092 , 0.60067072, 0.46724742, 1.        , 0.88247541,
        0.43458483, 0.43931681, 0.46967204, 0.36228857, 0.53785009],
       [0.62343144, 0.56202176, 0.46899748, 0.88247541, 1.        ,
        0.48666546, 0.48978165, 0.40565621, 0.33782212, 0.47895382],
       [0.61358055, 0.5378085 , 0.4548682 , 0.43458483, 0.48666546,
        1.        , 0.81952998, 0.32372241, 0.28557256, 0.391452  ],
       [0.60585218, 0.47622383, 0.56602397, 0.43931681, 0.48978165,
        0.81952998, 1.        , 0.38063749, 0.40951598, 0.28198586],
       [0.56066376, 0.55059971, 0.63001165, 0.46967204, 0.40565621,
        0.32372241, 0.38063749, 1.        , 0.55478425, 0.46827079],
       [0.53372316, 0.42745234, 0.82548937, 0.36228857, 0.33782212,
        0.28557256, 0.40951598, 0.55478425, 1.        , 0.09558913],
       [0.52289733, 0.57232277, 0.19900971, 0.53785009, 0.47895382,
        0.391452  , 0.28198586, 0.46827079, 0.09558913, 1.        ]])
```

In [18]:

```
# 販売価格との相関行列(ヒートマップ)
k =10 # 変数の数
# 関数の意味を記述する nlargest 大きい順にいくつかとる <->nsmallest
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
# 販売価格との相関係数が大きい順に10こ並べて、そのインデックスをcolsに代入
cm = np.corrcoef(df_train[cols].values.T)# 販売価格との相関係数が大きい順10項目の値で再度相関係数
# なぜ転置をとるのか http://lofas.hatenablog.com/entry/2015/02/09/150552
sns.set(font_scale=1.25) # 項目の名前のフォントを1.25倍する
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yti
```



見てみると、関係の強い変数が現れる。それぞれについて私が思ったことは、

- 「全体的な素材と仕上がりの質」「家の中の面積」「総敷地面積」は販売価格と強い関係性がある　チェックや！
- 「ガレージに駐車できる数」と「ガレージの面積」は「販売価格」と強い関係がある。しかし、ガレージが大きいからたくさん駐車できるだけで、この二つは双子の関係なので関係性がつよい「ガレージに駐車できる数」のみ残す。
- 「総敷地面積」と「１階の面積」も同じようなものなので、最初の直感で選んだ「総敷地面積」を残す。
- 「地上の浴槽・シャワー・洗面台・便器の4点が備え付けられた部屋」？本当？
- 「バスルームを除いた部屋の数」と「家の中の面積」も兄弟みたいなもの。チェルノブイリのデータセットなの？
- 「建てられた年」か。それも相関があるね、もしかして時系列分析しなきゃいけないかもって思ってる。それはあなたへの宿題ね！

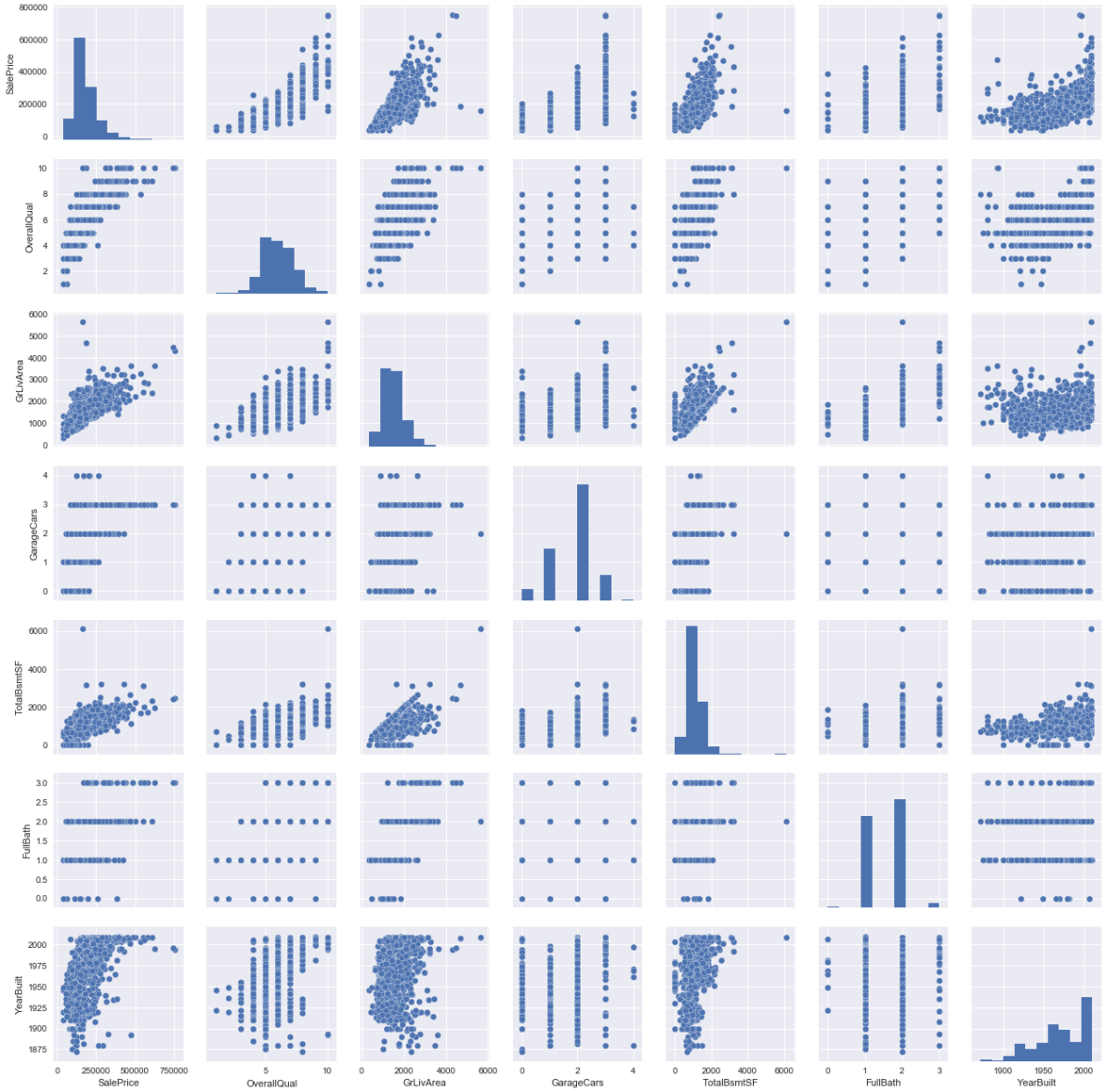
散布図に進もう！

販売価格と相関が強いものを散布図にしよう（ジャガースタイルで動こう）

見るべきものは準備できた。最初に散布図を見た時は感動したよ。たくさんの情報がぎっしり詰まっていてすばらしかった。 ありがとうseaborn！君のお陰でジャガーのように動けるよ。

In [24]:

```
# 散布図
sns.set()
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'YearBuilt']
sns.pairplot(df_train[cols], size=2.5)
plt.show();
```



だいたいはグラフを見る前から分かっているんだけど、プロットを見るとより妥当な考えが分かったりするんだ。

「敷地の総合面積」と「家の中の面積」が気になったんだけど、比例関係があって上に境界があるみたいだ。地下の面積は地上とだいたい一緒だけど、地上の面積より地下の面積が大きいことはないよ。（バンカーを買う時以外は） 「建てられた年」と「販売価格」は恥ずかしがり屋の指数関数のようで上限が徐々に上がっていくね。

ローシャッハテストは済んだね。ここからは欠損値を見ていこう！

4.欠損値

欠損値について考えるときに重要な質問は

- 欠損値はどれくらいあるか？
- 欠損値はランダムなのかパターンがあるのか？

この質問の答えは実用的な理由で重要です。なぜなら欠損値はサンプルサイズが減ることを意味するからです。 これはデータ分析にすすむのを阻みます。さらに、欠損値が偏っていないか、不都合な真実を隠していないか保証する必要があるんです。

In [30]:

```
# Nanデータの数量を集めて項目ごとの欠損値の数をまとめて降順でtotalに格納
total = df_train.isnull().sum().sort_values(ascending=False)

# 欠損率を降順でpercentに格納
percent = (df_train.isnull().sum()/df_train.isnull().count()).sort_values(ascending=False)

# 欠損数と欠損率をmissing_dataとしてそれを上位20表示する
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

Out[30]:

	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
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

欠損データをどう扱うかを理解するために分析してみましょう。

データが15%以上欠損しているとき、類似の変数を消してなかったことにしましょう。 このケースでは欠損データに何かを埋めるためになにかするといったことはない。 これによっていくつかデータを消せる。これはデータを失ったということ？違うと思います。これらの変数はあまり重要ではないってこと。家を買う時にあまり考えないからデータが集まらなかったと考えてもいいかもね。さらに注意深く見ていくといくつかの変数は外れ値だと思われるので喜んではずしましょう！

のこりのケースではガレージに関する変数は同じくらい欠損データがありますね。 だけど5%の欠損率でガタガタ言うよりはガレージの駐車できる車の数だけみれば事足りますね。 同じことが地下に関する変数にも当てはまります。

石造りのレンガの面積やタイプに関しては重要じゃないと考える。さらにこれらはすでに考慮した「建てられた年」と「総合的な質」と強い関係がある。故にこの変数を削っても情報は失われません。

まとめると、欠損データを扱うために、「電気システム」を除いたすべての欠損データを削除します。 電気システムは外れ値もろとも観測自体をやめよう。(??)

In [20]:

```
1 df_train.Electrical.value_counts()
```

Out[20]:

```
SBrkr    1334
FuseA      94
FuseF      27
FuseP       3
Mix         1
Name: Electrical, dtype: int64
```

In [31]:

```
1 df_train = df_train.drop((missing_data[missing_data['Total']> 1]).index, 1)
2 df_train = df_train.drop(df_train.loc[df_train['Electrical'].isnull()].index)
3 df_train.isnull().sum().max() # ちゃんと欠損値がなくなったか確認
```

Out[31]:

```
0
```

うそつき！（外れ値？）

外れ値も注意すべきものだ。なぜか？外れ値はモデルにかなりの影響を与えうるし、特定の振る舞いについての知見を与えてくれる重要な情報源にもなりうるからね。

外れ値は複雑な科目で注意深く見る必要がある。ここで「販売価格」の標準偏差と散布図を通してサッと見ていきましょう。

単一変数の分析

大事なのは外れ値を探すために閾値を設定することです。それをするのに、データを標準化しましょう。ここでいうデータの標準化とは、データを平均０、標準偏差１とすることです。

データの標準化とは

データセットを平均０、標準偏差１にすること

難しいことはなく 各データから平均を引いて、標準偏差で割る だけ。

In [34]:

```
# データの標準化

# 関数の中身を記述する
saleprice_scaled = StandardScaler().fit_transform(df_train['SalePrice'][:,np.newaxis]);
# np.newaxisはNoneの参照なので[:,None]でも同じだが軸を追加する意味になり、コードの分かりやすさの面か
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]]
```

「販売価格」に

- 低い範囲の値は似通っていて、平均からそんなに離れていない - 高い範囲の値はずっと離れていて、実際７この値は範囲外です。

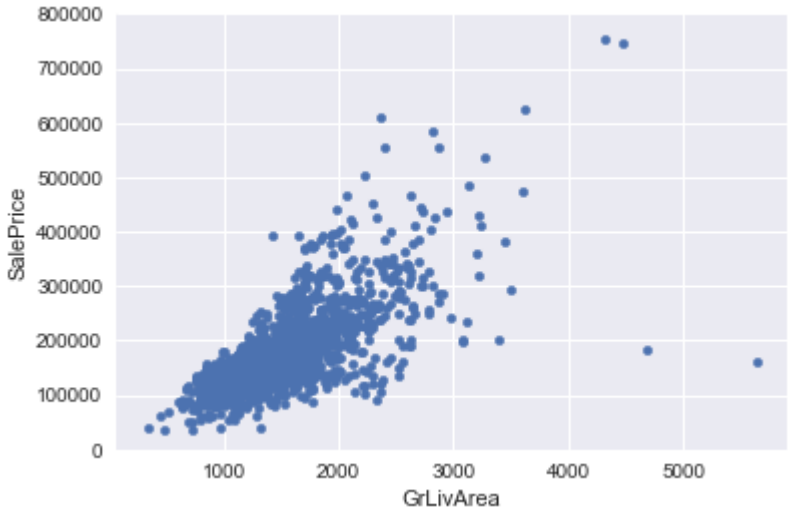
さて、これらの値を外れ値として考慮しないようにするが、注意しておくべきだ。？？

二変量分析

下の散布図はもういやってほど見てきましたね。しかし、新しい観点から見てみると、いつも発見がありますよね。見方を変えるのは80IQほどの価値があるよ。（by Alan Kay)

In [35]:

```
# 二変量分析 販売価格と地上の部屋の面積
var = 'GrLivArea'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));
```



これから分かることは

- 「地上の部屋の面積」が大きくなると次第におかしくなり、群れに従わなくなる。（広がっていく）なぜそうなるのかじっくり考える。もしかして農地に関係してそれが安さの原因かも。？？わからないが、私はこれらの２つの点が典型的事例を代表するものではないと自信がある。ゆえにそれらを異常値としています。
- ７つの観測点　２つの　これらは２つの特例だけど、トレンドにはしたがっているのでキープします。？？

In [36]:

```
# 点の削除

df_train.sort_values(by = 'GrLivArea', ascending=False)[:2]
df_train = df_train.drop(df_train[df_train['Id'] == 1299].index)
df_train = df_train.drop(df_train[df_train['Id'] == 524].index)
```

In [37]:

```
# 2変量分析「販売価格」と「地上の住居面積」
var = 'TotalBsmtSF'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0, 800000));
```

いくつか観測値を排除したいと思うかもしれませんが、その価値はないと思います。 あっても問題ないので放置します。

5.核心に迫るよ

？？は誰だ？って話のほとんど

「販売価格」は誰だ？

質問への答えは多変量解析のための統計的根拠の基礎となる仮定をテストすることで分かるはずだ。 これまでデータを綺麗にして「販売価格」さんについてたくさん発見があった。

ここからさらに深くもぐって、「販売価格」さんに多変量のテクニックを駆使してどう統計的仮説を当てはめるかだ。

- 正常か ここでいう正常は正規分布にそっているかだ。これはいくつかの統計テストが正規分布に基づいているからです。ここでは単変数の正常さをチェックしてきた（これは限られたアプローチです） ただ単変数の正常性は多変数の正常性を保証するとは限らないことを覚えておいてください。けど助けにはなるから。 また２００データ以上のサンプルがあるときは正常性はそこまで問題にならない、ただ正常性に関してちゃんとやっておけば多くの問題をさけられそれが分析を行う主要な理由になる。？？
- 均質性 均質性は予測変数の範囲に依存変数があるという仮説をいいたい。

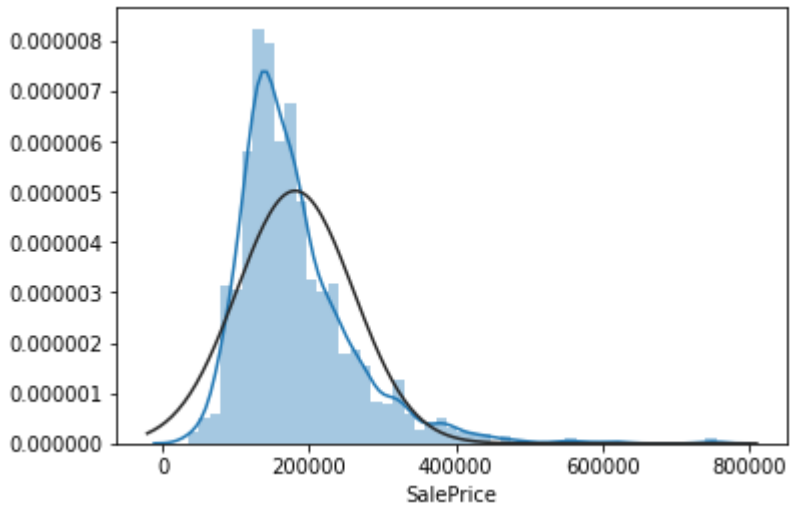
何言っているか分からない？じゃあ行動してみよう！

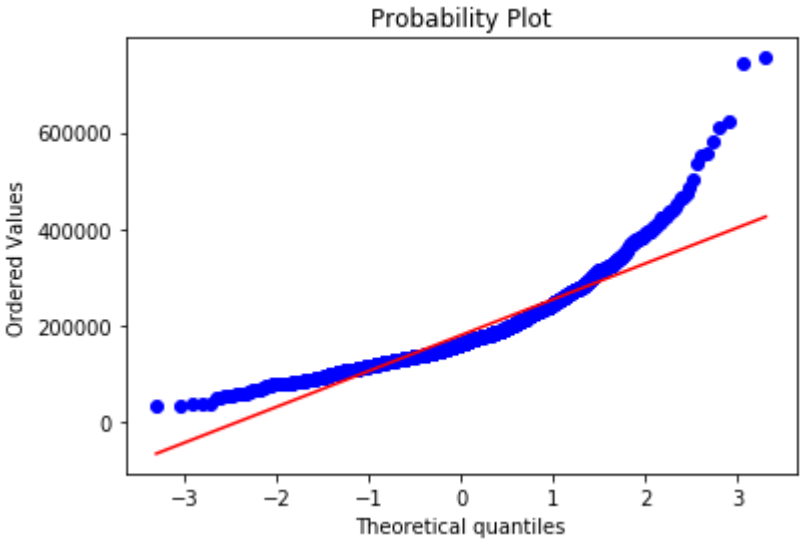
正常性の調査

- ヒストグラム　歪度と尖度
- 正規確率プロット　正規分布に従っているかを確認できる

In [8]:

```
# ヒストグラムと 正規確率プロット
sns.distplot(df_train['SalePrice'], fit=norm);
fig = plt.figure()
res = stats.probplot(df_train['SalePrice'], plot=plt)
```





見てみると「販売価格」は正常でないね。ピークがあって、 正の歪度があって、斜めの対角線に従っていないね。

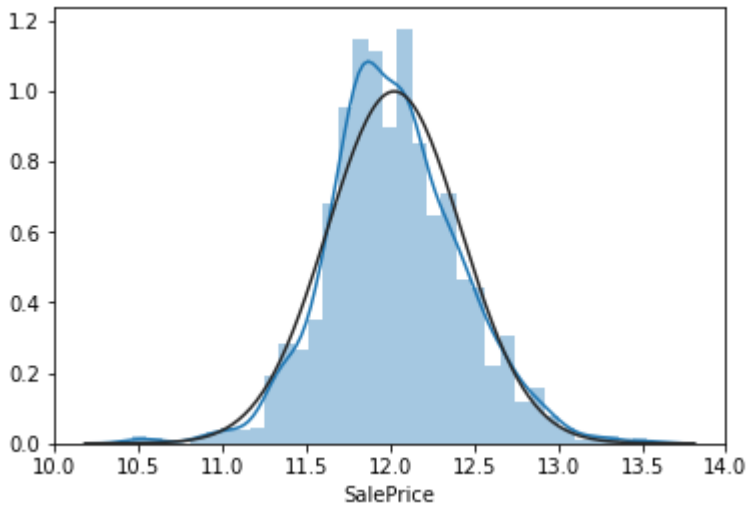
だけどまだ負けたわけじゃない。すごいテクニックがあって正の歪度がある場合は対数をとればうまくいくんだ。 すけーだろ！

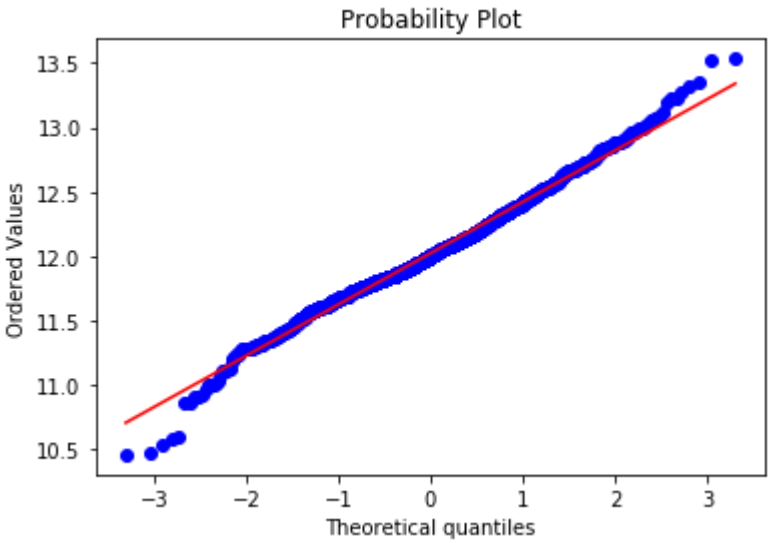
In [9]:

```
# 対数を取る
df_train['SalePrice'] = np.log(df_train['SalePrice'])
```

In [10]:

```
# どうなったか確認
sns.distplot(df_train['SalePrice'], fit=norm);
fig = plt.figure()
res = stats.probplot(df_train['SalePrice'], plot = plt)

```

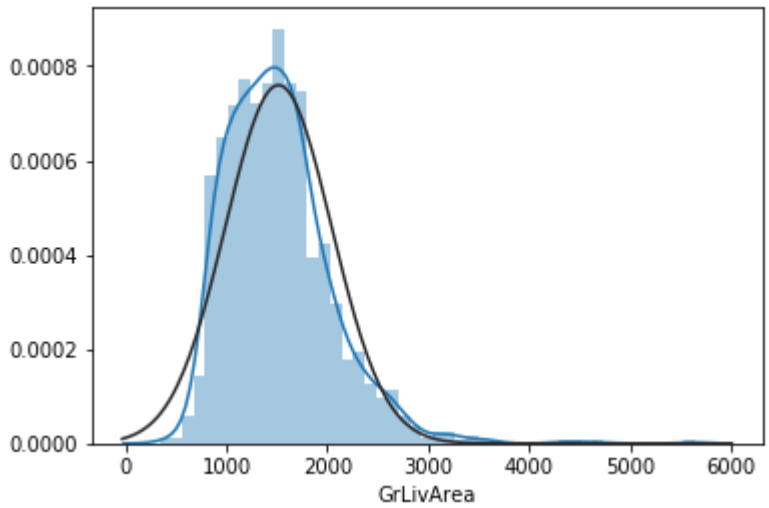


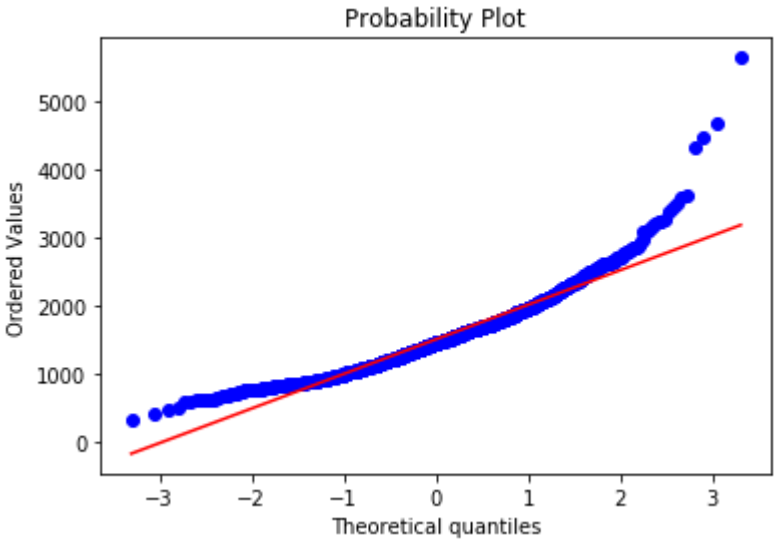


やったぜ！「地上のリビングの敷地面積」も同じようにやってみよう!

In [11]:

```
# まずそのまま見てみる
sns.distplot(df_train['GrLivArea'], fit=norm);
fig = plt.figure()
res = stats.probplot(df_train['GrLivArea'], plot=plt)
```





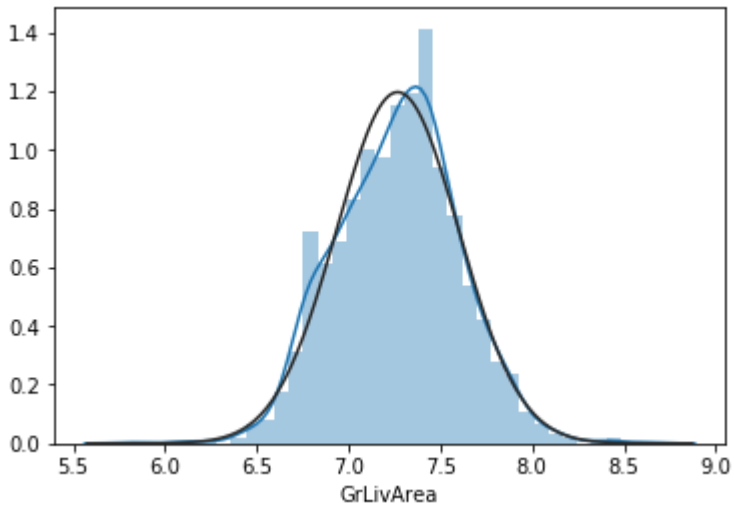
ちょっとゆがんでるな！魔法でちょちょい

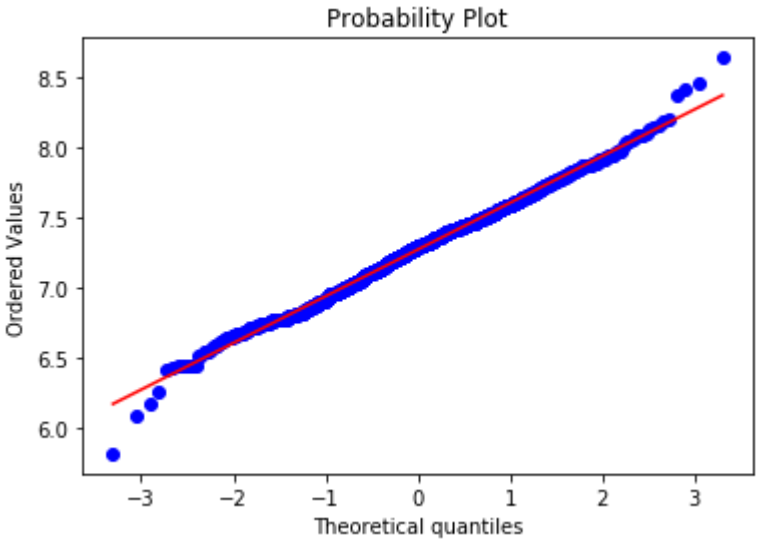
In [12]:

```
# 対数を取る
df_train['GrLivArea'] = np.log(df_train['GrLivArea'])
```

In [14]:

```
# 効果を確認
sns.distplot(df_train['GrLivArea'], fit=norm)
fig = plt.figure()
res = stats.probplot(df_train['GrLivArea'], plot=plt)
```

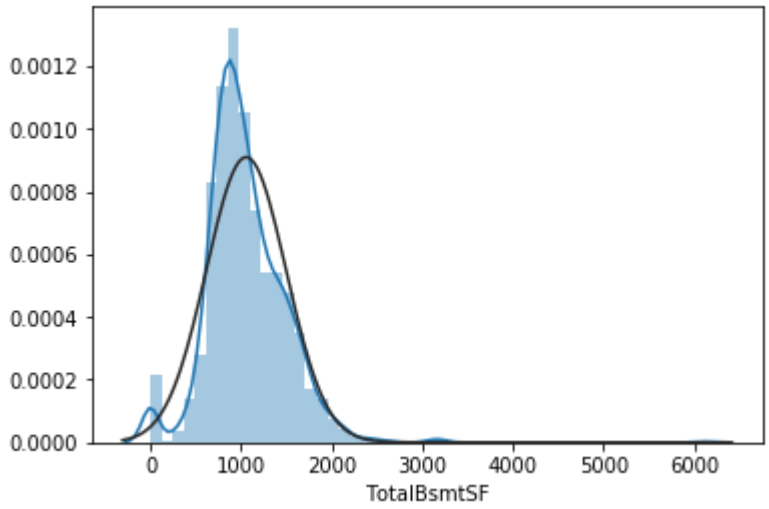


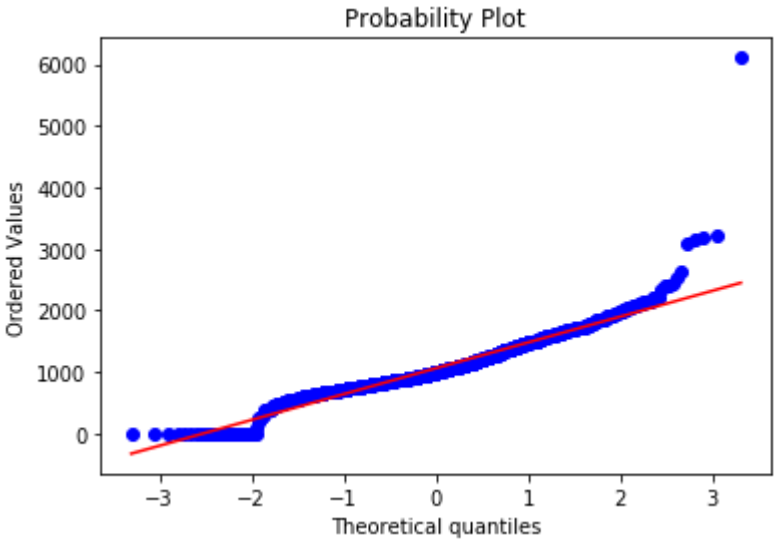


次！

In [15]:

```
sns.distplot(df_train['TotalBsmtSF'], fit=norm);
fig = plt.figure()
res = stats.probplot(df_train['TotalBsmtSF'], plot=plt)
```





なかなか手ごわいやつがきたね！どうすりゃいいかね？

- 歪みが少ない
- 地下がない家もあるから0付近にかなり数がある
- 0は対数をとれない（マイナス無限大になっちゃう）から大問題！

対数をとるために0かそうでないかを区別して、０でない値にだけ対数をとると0であるという情報を損なわない。

私はこのアプローチが正しいか分からない。けどうまく動いてる。これをハイリスクエンジニアリングとよんでいます。

In [16]:

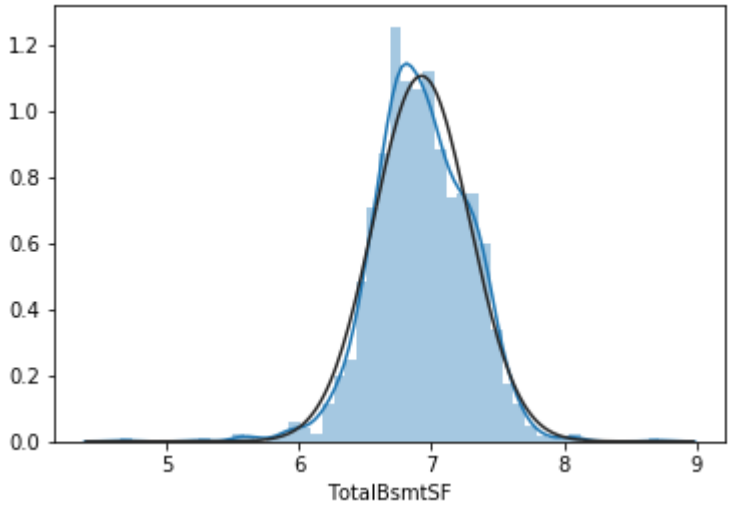
```
# 値が0より大きいものを1、それ以外を0とした新しいカテゴリー変数を作る
df_train['HasBsmt'] = pd.Series(len(df_train['TotalBsmtSF']), index=df_train.index)
df_train['HasBsmt'] = 0
df_train.loc[df_train["TotalBsmtSF"]>0, 'HasBsmt'] =1
```

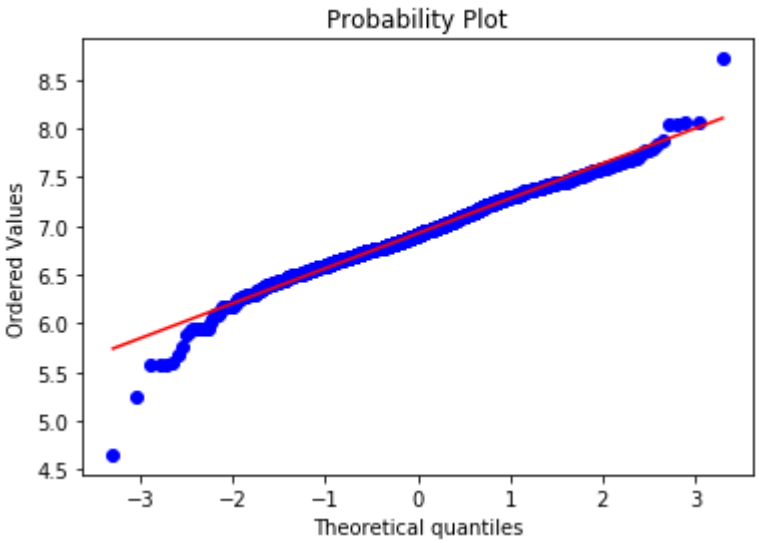
In [18]:

```
# ここで対数を取る
df_train.loc[df_train['HasBsmt']==1,'TotalBsmtSF'] = np.log(df_train['TotalBsmtSF'])
```

In [20]:

```
# 確認してみる
sns.distplot(df_train[df_train['TotalBsmtSF']>0]['TotalBsmtSF'], fit=norm);
fig = plt.figure()
res =stats.probplot(df_train[df_train['TotalBsmtSF']>0]['TotalBsmtSF'], plot=plt)
```





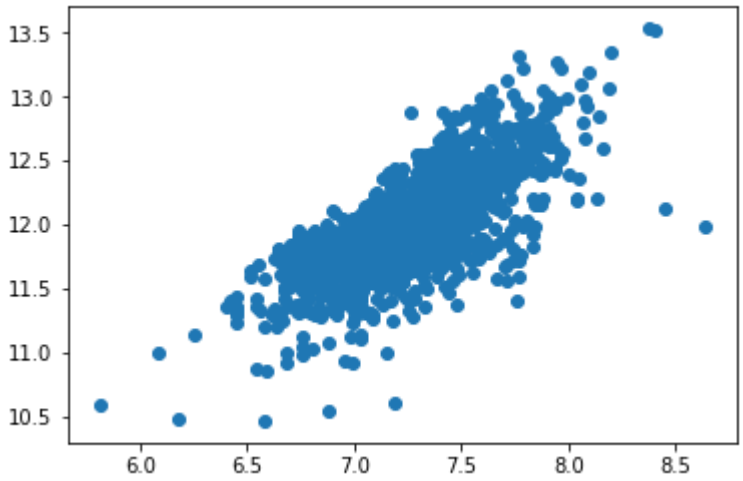
一発で均質性が正しいか調査

2つのメトリック変数の均質性を確認するアプローチはグラフにすること。

例によって「販売価格」と「地上のリビングの敷地面積」を見てみよう

In [21]:

```
#scatter plot
plt.scatter(df_train['GrLivArea'], df_train['SalePrice']);
```

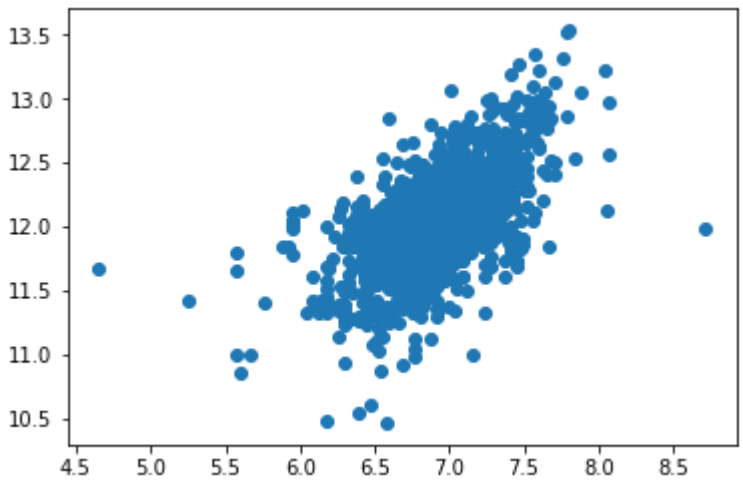


対数を取る前の古いバージョンの散布図は円錐型です（もどってみてみて！） 今のグラフはそうじゃないだろ。これが正常性の力だ！ 正常性を担保すると均質性も解決してるのさ。

「販売価格」と「地下の敷地面積」を見てみよう

In [23]:

```
plt.scatter(df_train[df_train['TotalBsmtSF']>0]['TotalBsmtSF'], df_train[df_train['TotalBsmtSF']>
```



一般的に「販売価格」と「地下の敷地面積」は同じ分散レベルに収まっているだろう。クールだ！

最後にダミー変数

イージーモード

In [24]:

```
# カテゴリーデータをダミーにする
df_train = pd.get_dummies(df_train)
```

In [25]:

```
1 df_train.head()
```

Out[25]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	1	60	65.0	8450	7	5	2003	2003
1	2	20	80.0	9600	6	8	1976	1976
2	3	60	68.0	11250	7	5	2001	2002
3	4	70	60.0	9550	7	5	1915	1970
4	5	60	84.0	14260	8	5	2000	2000

5 rows × 291 columns

結論

これでおしまい！最後まで来たよ！

このカーネルを通して、「Multivariate Data Analysis」の多くの戦略を練習してきたよ。 変数とは何かを考え、「販売価格」だけに目を向けて分析して、関係する変数を分析して、欠損値やはずれ値も扱った。 また本質的な統計的仮説も検証したし、カテゴリーデータをダミーにもしてきた。 pythonが助けもあってこんなにたくさんの仕事をこなしてきたんだ。

でもまだ話は終わっていない。調査の直前でとまっている。「販売価格」さんに電話してディナーに招待しよう。 そして彼女の振る舞いを予測しよう。彼女は正規化線形回帰アプローチを楽しむクチかい？それともアンサンブルメソッド、それとも他の手法？

それを見つけるのはあなた次第です。

データの傾向がわかったところで実際のモデル作成に取り掛かろうってこと？