Trabalho 1 - Relatorio

July 26, 2018

Autor _____ Danilo Filippo Reizel Pereira (DRE 113051728)
Kaggle _____ https://www.kaggle.com/danilofrp

1 Resumo

Este relatório tem como objetivo apresentar as técnicas de pré-processamento empregadas, bem como os modelos construídos, para o trabalho da disciplina de Aprendizado de Máquina (EEL891), do curso de Engenharia Eletrônica e de Computação. A seção de introdução fará um breve resumo das técnicas empregadas neste trabalho, e as seções seguintes apresentarão detalhadamente os passos tomados, jutamente com o código e os resultados obtidos

1.1 Descrição do trabalho

Este trabalho é baseado no desafio do kaggle House Prices: Advanced Regression Techniques. O desafio consiste em um problema de regrssão, no qual deve-se prever o preço de venda de um imóvel na cidade de Ames, Iowa, baseado em diversas características deste imóvel, como área construída, área do terreno, localização do terreno, ano de construção, entre outros. Por apresentar diversas features de natureza variada, torna-se um bom desafio para treinar técnicas de pré-processamento e análise de dados.

1.2 Técnicas utilizadas

1.2.1 Pré-processamento

Devido ao grande número de features e variabilidade da natureza das mesmas, técnicas aprorpiadas de pré-processamento precisam ser empregadas. De início, removemos algumas amostras consideradas outliers, seja por estarem explicitamente apontadas como outliers na descrição do dataset ou por terem sido consideradas outliers com base em uma anáise crítica das variáveis.

Após a remoção dos outliers, performamos algumas transformações nos dados. Como pode ser observado lendo a descrição do dataset, há um dado (*MSSubClass*) que é categórico apesar de ser representado por valores numéricos. Também é possível perceber que há alguns dados categóricos que possuem uma relação de ordem entre si, como por exemplo aqueles cujas "categorias" são Excellent, Good, Average/Typical, Fair e Poor. Por fim, também transformamos o dado *Neighborhood* em uma variável numérica, pela adição de dados externos, encontrados enquanto se procurava mais informações sobre o dataset. Esses dados (https://ww2.amstat.org/publications/jse/v19n3/decock/AmesResidential.pdf) apresentam um fator multiplicador para cada bairro da cidade, que foi utilizado para normalizar os preços

(divide-se o preço de venda pelo fator multiplicador do bairro, após a previsão aplica-se a operação oposta).

Feita a transformação dos dados, tratamos os dados faltantes. Optamos por excluir do modelo as features que apresentam grande quantidade de dados faltantes, como é o caso de *PoolQC* (99,72% de dados faltantes). Para as demais features, técnicas simples de substituição pela moda, para o caso de variáveis categóricas, ou substituição pela mediana, para variáveis numéricas, foram empregadas. Também criamos uma variável extra, a área interna total do imóvel, que apresentou uma grande correlação com o preço. Após esta etapa, verificou-se que algumas variáveis numéricas, bem como a variável alvo, não apresentavam similaridade satisfatória com uma gaussiana, então realizaram-se transformações logp1 e box-cox para reduzir a assimetria das distribuições.

Após esta etapa, foram analisadas as correlações entre as features e a variável alvo, e também entre features. Primeiramente, realzou-se uma transformação nas variáveis *YearBuilt*, *YearRemodAdd* e *GarageYrBlt* para que passassem a representar a idade relativa ao momento da venda, em vez de anos exatos. Depois, analisou-se a correlação entre as variáveis *GarageArea* e *GarageCars*, e percebeu-se notamos uma forte relação entre elas. Optamos por retirar a variável *GarageCars* do modelo, para evitar o input de dados redundantes. Da mesma forma, foram removidas as variáveis *TotRmsAbvGrd*, *GarageAgeAtSale*, *TotalBsmtSF*, *GrLivArea* e *GarageCond*, também por apresentarem forte relação com outras variáveis de entrada. Optou-se também por remover as variáveis que apresentavam correlação menor do que 10% com a saída, por julgar que estas variáveis não seriam boas descritoras do preço de venda, e poderiam agir como ruído.

Por fim, transformamos as variáveis categóricas utilizando o método de *OneHotEncoding*. Durante a aplicação desse método, foi garantido que uma das variáveis *dummy* de cada categoria fosse removida, para evitar a chamada *dummy variable trap*, na qual uma feature pode ser prevista pelas demais. Para o caso especial dos pares de variáveis *(Condition1, Condition2)* e *(Exterior1st, Exterior2nd)*, que representam a mesma feature, foi utilizada uma codificação na qual permitiu-se mais até duas variáveis *dummy* estarem ativas simultaneamente.

Ao final da fase de pré-processamento, o dataset é composto de 145 features, das quais 34 são numéricas e 111 são variáveis *dummy* provenientes de 23 variáveis categóricas.

1.2.2 Modelagem

Terminada a fase de pré-processamento, foram treinados diversos modelos para fins de comparação. Os modelos treinados foram *Lasso Regression*, *Elastic Net*, *Bayesian Ridge Regression*, *Gradient Boost*, *Light Gradient Boost*, *Extreme Gradient Boost*, *Random Forest Regression*, *K-NN Regression* e *Support Vector Regression*. Para todos os modelos, foi feita uma validação cruzada com 5 *folds* para avaliar a performance, e, quando necessário, foi aplicado o RobustScaler aos dados. Uma vez treinados os modelos, foram selecionados os 4 melhores de acordo com a validação cruzada, e um perceptron foi treinado para criar um ensemble baseado nas saídas desses modelos.

1.3 Resultados

Os resultados obtidos neste trabalho foram submetidos ao kaggle para avaliação no dataset de testes. No dataset de treino, obteve-se um RMSLE de aproximadamente 0.08630, enquanto no dataset de teste obteve-se um resultado de 0.11040. O resultado no dataset de teste foi suficiente para atingir a 48a posição no ranking do desafio (apurado em 26/07/2018).

1.4 Conclusão

Ao longo do desenvolvimento do trabalho pode-se perceber a importância da etapa de pré-processamento para modelos de machine learning, em especial pode-se notar um impacto nos resultados muito superior ao da modificação dos parâmetros dos modelos. Em particular, os processos que geraram a maior melhora no resultado foram a adição dos dados externos de multiplicador para os bairros, as transformações nos dados para torná-los gaussianos, e a remoção de variáveis pouco relevantes. A adição do perceptron para a realização do ensemble também causou um grande impacto na performance do estimador final.

---

2 Detalhamento

Abaixo, pode-se encontrar a descrição detalhada das decisões tomadas, bem como o código utilizado e os resultados gerados

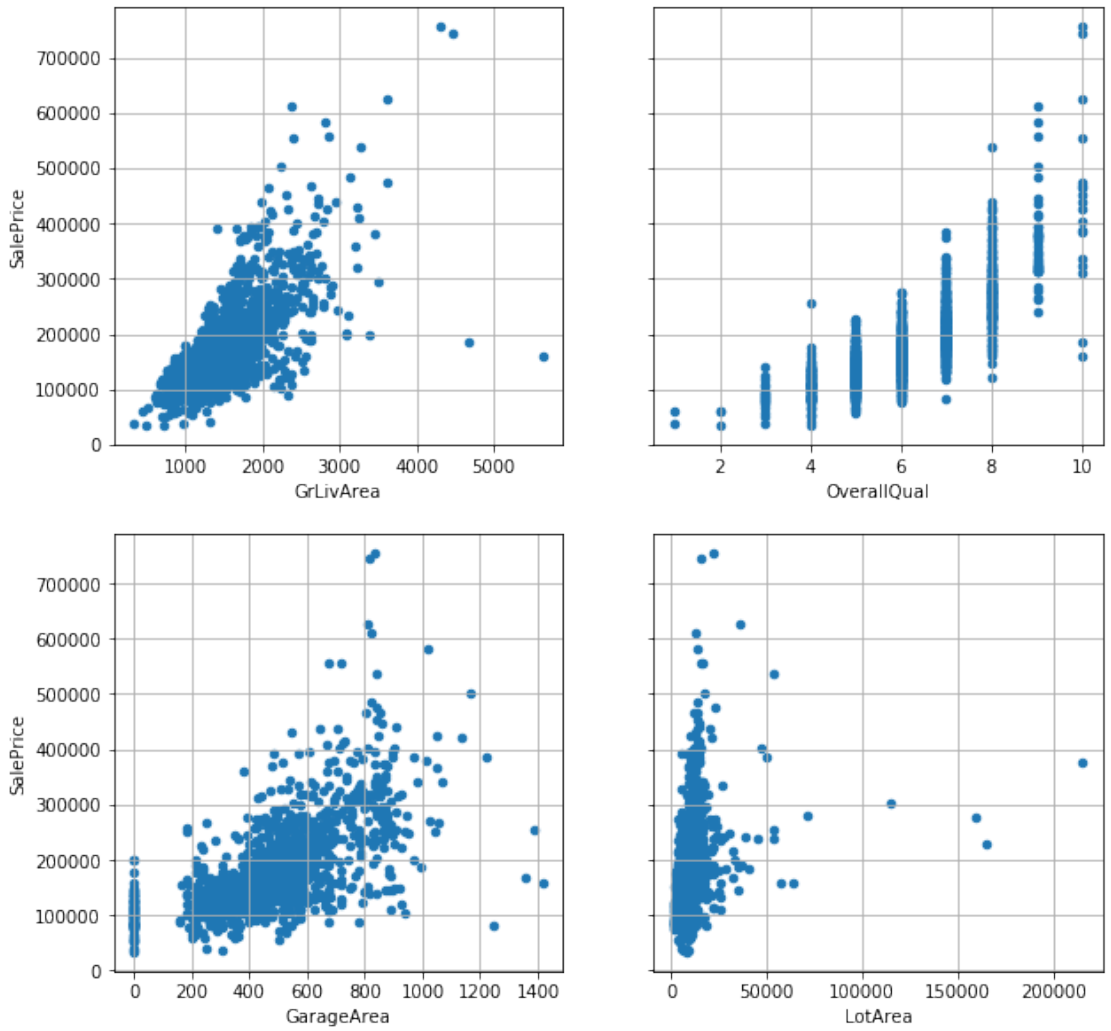
```
In [1]: # Bibliotecas básicas necessárias para o código:
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import warnings
        warnings.filterwarnings('ignore')
```

2.1 Aquisição dos dados

```
In [2]: data_train = pd.read_csv('../data/train.csv')
        data_test = pd.read_csv('../data/test.csv')
        data_test['SalePrice'] = -1
        test_ids = data_test['Id']
```

2.2 Remoção de outliers

```
In [3]: colnames = ['GrLivArea','OverallQual','GarageArea','LotArea']
        data = pd.concat([data_train['SalePrice'], data_train[colnames]], axis=1)

        fig, axes = plt.subplots(2, 2, figsize=(10, 10))
        for i in range(len(colnames)):
            data.plot.scatter(x = colnames[i], y = 'SalePrice', ax = axes[i//2, i%2],
                              sharey = True, grid = True)
```



Podemos perceber alguns outliers pela análise das variáveis *GrLivArea*, *GarageArea* e *LotArea*, no qual as variáveis são muito superiores às demais, porém o preço não segue essa tendência. Removemos tais amostras do dataset

```
In [4]: data_train = data_train[data_train['GrLivArea'] < 4000]
        data_train = data_train[data_train['GarageArea'] < 1200]
        data_train = data_train[data_train['LotArea'] < 100000]

        all_data = pd.concat((data_train, data_test))
        target = data_train['SalePrice']
```

2.3 Transformações Numéricas/Categóricas

Lendo a descrição do dataset, podemos perceber que MSSubClass é uma variável categórica, apesar de ser tratada numericamente, assim, transformamos em uma string para depois aplicar o LabelEncoder e OneHotEncoder facilmente.

```
In [5]: all_data['MSSubClass'] = all_data['MSSubClass'].transform(lambda x: str(x))\
                                                       .astype(str)
```

Também podemos perceber que as variáveis *ExterQual*, *ExterCond*, *BsmtQual*, *BsmtCond*, *HeatingQC*, *KitchenQual*, *FireplaceQu*, *GarageQual*, *GarageCond* e *PoolQC*, apesar de a princípio parecerem variáveis categóricas, podem ser representadas por variáveis numéricas, pois seus níveis (NA, Po, Fa, TA, Gd, Ex) tem uma relação de superioridade/inferioridade entre um e outro. Assim, podemos mapeá-los da seguinte forma: {NA: 0, Po: 1, Fa: 2, TA: 3, Gd: 4, Ex: 5}.

O mesmo acontece para as variáveis *LandSlope*, *BsmtExposure*, *BsmtFinType1*, *BsmtFinType2*, *Functional*, *GarageFinish*, *PavedDrive* e *Fence*.

```
In [6]: grades_map = {'NA': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}
        def map_grade(grade):
            if isinstance(grade, float) and np.isnan(grade): return grade
            return grades_map[grade]

        landslope_map = {'Gtl': 0, 'Mod': 1, 'Sev': 2}
        def map_landslope(slope):
            if isinstance(slope, float) and np.isnan(slope): return slope
            return landslope_map[slope]

        bsmtexposure_map = {'No': 0, 'Mn': 1, 'Av': 2, 'Gd': 3}
        def map_bsmtexposure(exposure):
            if isinstance(exposure, float) and np.isnan(exposure): return exposure
            return bsmtexposure_map[exposure]

        bsmtfintype_map = {'Unf': 0, 'LwQ': 1, 'Rec': 2, 'BLQ': 3, 'ALQ': 4, 'GLQ': 5}
        def map_bsmtfintype(fintype):
            if isinstance(fintype, float) and np.isnan(fintype): return fintype
            return bsmtfintype_map[fintype]

        functional_map = {'Typ': 0, 'Min1': 1, 'Min2': 2, 'Mod': 3, 'Maj1': 4,
                          'Maj2': 5, 'Sev': 6, 'Sal': 7}
        def map_functional(functional):
            if isinstance(functional, float) and np.isnan(functional): return functional
            return functional_map[functional]

        garagefin_map = {'Unf': 0, 'RFn': 1, 'Fin': 2}
        def map_garagefin(garagefin):
            if isinstance(garagefin, float) and np.isnan(garagefin): return garagefin
            return garagefin_map[garagefin]

        pave_map = {'N': 0, 'P': 1, 'Y': 2}
        def map_pave(pave):
            if isinstance(pave, float) and np.isnan(pave): return pave
            return pave_map[pave]

        fence_map = {'NA': 0, 'MnWw': 1, 'GdWo': 2, 'MnPrv': 3, 'GdPrv': 4}
```

```
        def map_fence(fence):
            if isinstance(fence, float) and np.isnan(fence): return fence
            return fence_map[fence]

        for col in ('ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'HeatingQC',
                    'KitchenQual', 'FireplaceQu', 'GarageQual', 'GarageCond', 'PoolQC'):
            all_data[col] = all_data[col].transform(map_grade).astype(float)

        all_data['LandSlope']    = all_data['LandSlope'].transform(map_landslope).astype(float)
        all_data['BsmtExposure'] = all_data['BsmtExposure'].transform(map_bsmtexposure).astype(f
        all_data['BsmtFinType1'] = all_data['BsmtFinType1'].transform(map_bsmtfintype).astype(fl
        all_data['BsmtFinType2'] = all_data['BsmtFinType2'].transform(map_bsmtfintype).astype(fl
        all_data['Functional']   = all_data['Functional'].transform(map_functional).astype(float
        all_data['GarageFinish'] = all_data['GarageFinish'].transform(map_garagefin).astype(floa
        all_data['PavedDrive']   = all_data['PavedDrive'].transform(map_pave).astype(float)
        all_data['Fence']        = all_data['Fence'].transform(map_fence).astype(float)
```

Por fim, como encontrado na descrição do dataset, podemos utilizar dados auxiliares para transformar os bairros em seus multiplicadores. (https://ww2.amstat.org/publications/jse/v19n3/decock/AmesResidential.pdf)

```
In [7]: neighborhood_map = {'CollgCr': 98, 'Veenker': 98, 'Crawfor': 106, 'NoRidge': 101,
                           'Mitchel': 99, 'Somerst': 101, 'NWAmes': 99, 'OldTown': 102,
                           'BrkSide': 106, 'Sawyer': 101, 'NridgHt': 104, 'NAmes': 100,
                           'SawyerW': 98, 'IDOTRR': 102, 'MeadowV': 90, 'Edwards': 98,
                           'Timber': 103, 'Gilbert': 97, 'StoneBr': 104, 'ClearCr': 103,
                           'NPkVill': 109, 'Blmngtn': 105, 'BrDale': 105, 'SWISU': 99 ,
                           'Blueste': 99}
        def map_neighborhood(neighborhood):
            if isinstance(neighborhood, float) and np.isnan(neighborhood): return neighborhood
            return neighborhood_map[neighborhood]

        all_data['Neighborhood'] = all_data['Neighborhood'].transform(map_neighborhood)\
                                                           .astype(float)

In [8]: numerical_cols = all_data.select_dtypes(include=['int64', 'float64'])\
                                                 .drop('Id', axis = 1).columns.values
        categorical_cols = all_data.select_dtypes(exclude=['int64', 'float64'])\
                                                 .columns.values
```

2.4 Dados Faltantes

Contando os dados faltantes para cada feature:

```
In [9]: total = all_data.isnull().sum().sort_values(ascending=False)
        percent = (all_data.isnull().sum()*100/all_data.isnull().count())\
                                                 .sort_values(ascending=False)
        missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
        missing_data[missing_data['Total'] > 0].round({'Percent': 2})
```

```
Out[9]:                Total  Percent
         PoolQC         2899    99.72
         MiscFeature    2805    96.49
         Alley          2709    93.19
         Fence          2337    80.39
         FireplaceQu    1419    48.81
         LotFrontage     482    16.58
         GarageCond      159     5.47
         GarageYrBlt     159     5.47
         GarageFinish    159     5.47
         GarageQual      159     5.47
         GarageType      157     5.40
         BsmtCond         82     2.82
         BsmtExposure     82     2.82
         BsmtQual         81     2.79
         BsmtFinType2     80     2.75
         BsmtFinType1     79     2.72
         MasVnrType       24     0.83
         MasVnrArea       23     0.79
         MSZoning          4     0.14
         Utilities         2     0.07
         BsmtHalfBath      2     0.07
         BsmtFullBath      2     0.07
         Functional        2     0.07
         Exterior1st       1     0.03
         TotalBsmtSF       1     0.03
         BsmtUnfSF         1     0.03
         BsmtFinSF2        1     0.03
         GarageArea        1     0.03
         KitchenQual       1     0.03
         GarageCars        1     0.03
         BsmtFinSF1        1     0.03
         Exterior2nd       1     0.03
         SaleType          1     0.03
         Electrical        1     0.03
```

Escolhemos ignorar as features com mais de 30% dos dados faltantes, pois com tantos dados faltantes a feature não é uma boa descritora do preço de venda ou então uma tentativa de aproximação pode gerar muitos erros.

Para o caso da piscina, como há apenas 5 amostras no dataset de treino, também ignoramos a variável *PoolArea*.

```
In [10]: all_data.drop(missing_data[missing_data['Percent'] > 30].index,
                      axis = 1, inplace = True)
         all_data.drop('PoolArea', axis = 1, inplace = True)
```

Para a feature *LotFrontage*, assumimos que a area frontal do terreno é igual à mediana da área frontal dos terrenos da mesma vizinhança da amostra

```
In [11]: all_data['LotFrontage'] = all_data.groupby('Neighborhood')['LotFrontage']\
                                        .transform(lambda x: x.fillna(x.median()))
```

Para as informações relativas à garagem, os dados faltantes indicam que a casa não possui garagem. Assim, substituimos os valores por None (categóricos) ou 0 (numéricos). Para o caso do ano de construção da garagem, repetimos o ano de construção da casa. Foi observado que no dataset de treino existe uma amostra na qual a feature *GarageYrBlt* apresenta ano de construção 2207, o que é impossível. Foi assumido que tal valor foi um erro de digitação e seu valor foi substituído por 2007.

```
In [12]: for column in ('GarageType', 'GarageFinish', 'GarageQual',
                        'GarageCond', 'GarageArea', 'GarageCars'):
             if column in numerical_cols:
                 all_data[column].fillna(0, inplace = True)
             else:
                 all_data[column].fillna('None', inplace = True)

         all_data['GarageYrBlt'].fillna(all_data['YearBuilt'], inplace = True)
         index = all_data[all_data['GarageYrBlt'] > max(all_data['YearBuilt'])].index
         all_data.iloc[index]['GarageYrBlt'] = 2007
```

Para as variáveis *Bsmt[...]*, realizamos o tratamento de forma similar ao tratamento das variáveis de garagem

```
In [13]: for column in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF',
                        'BsmtFullBath', 'BsmtHalfBath', 'BsmtQual', 'BsmtCond',
                        'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
             if column in numerical_cols:
                 all_data[column].fillna(0, inplace = True)
             else:
                 all_data[column].fillna('None', inplace = True)
```

Para as variáveis *MasVnrType* e *MasVnrArea*, dados faltantes representam que não há alvenaria, então substituímos por None ou 0

```
In [14]: all_data["MasVnrType"].fillna("None", inplace = True)
         all_data["MasVnrArea"].fillna(0, inplace = True)
```

Para as variaveis *MSZoning*, *SaleType*, *Utilities* e *Functional* substituimos pela moda

```
In [15]: all_data['MSZoning'].fillna(all_data['MSZoning'].mode()[0], inplace = True)
         all_data['SaleType'].fillna(all_data['SaleType'].mode()[0], inplace = True)
         all_data['Utilities'].fillna(all_data['Utilities'].mode()[0], inplace = True)
         all_data['Functional'].fillna(all_data['Functional'].mode()[0], inplace = True)
```

Para as variáveis *Electrical*, *Exterior1st*, *Exterior2nd* e *KitchenQual*, utilizamos a moda das construções na mesma vizinhança da amostra faltante

```
In [16]: for column in ('Electrical', 'Exterior1st', 'Exterior2nd', 'KitchenQual'):
             all_data[column] = all_data.groupby('Neighborhood')[column]\
                                        .transform(lambda x: x.fillna(x.mode()[0]))
```

Verificando se todos os dados faltantes foram tratados:

```
In [17]: total_after = all_data.isnull().sum().sort_values(ascending=False)
         if total_after[total_after > 0].empty:
             print('Não há mais dados faltantes')
         else:
             print(total_after[total_after > 0])
```

```
Não há mais dados faltantes
```

Por fim, adicionamos uma variável auxiliar, a área interna total do imóvel (diferente de *GrLivArea*, que desconsidera a área no subsolo)

```
In [18]: all_data['TotalSF'] = all_data['TotalBsmtSF']\
                              + all_data['1stFlrSF']\
                              + all_data['2ndFlrSF']
```

Separando novamente o dataset em treino e teste:

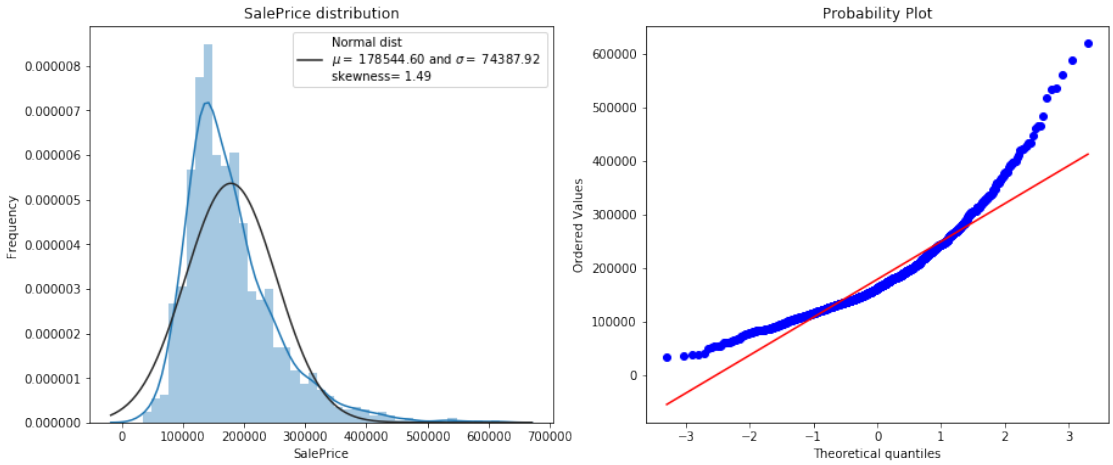
```
In [19]: data_train = all_data[all_data['SalePrice'] > 0]
         data_test = all_data[all_data['SalePrice'] < 0]

         numerical_cols = all_data.select_dtypes(include=['int64', 'float64'])\
                                                 .drop('Id', axis = 1).columns.values
         categorical_cols = all_data.select_dtypes(exclude=['int64', 'float64'])\
                                                 .columns.values
```

Realizando a normalização do preço de venda pelo multiplicador do bairro:

```
In [20]: data_train['SalePrice'] = data_train['SalePrice']*100/data_train['Neighborhood']
         target = data_train['SalePrice']
```

2.5 Transformação nas variáveis numéricas

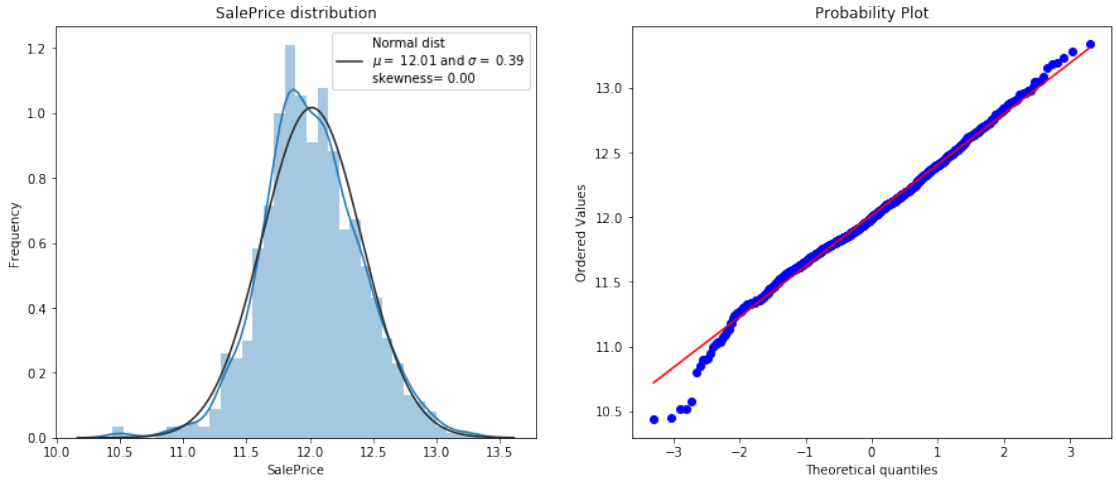
```
In [21]: from scipy import stats
         from scipy.stats import norm, skew

         fig, axes = plt.subplots(1, 2, figsize=(15, 6))
         sns.distplot(target, fit = norm, ax = axes[0]);
         (mu, sigma) = norm.fit(target)

         axes[0].legend(['Normal dist \n$\mu=$ {:.2f} and $\sigma=$ {:.2f} \nskewness= {:.2f}'
                                                 .format(mu, sigma, skew(target))],
                        loc='best')
         axes[0].set_ylabel('Frequency')
         axes[0].set_title('SalePrice distribution')

         stats.probplot(target, plot = axes[1])
         plt.show()
```



Podemos perceber que a variável alvo não se aproxima da distribuição normal, e tem uma cauda longa para a direita. É recomendável realizar uma transformação nos dados para aproximá-los de uma gaussiana, o que costuma melhorar a performance dos modelos. Para isso, será utilizada a transformação logp1. Como o desafio no kaggle avalia o RMSLE, ao realizar esta transformação na saída, podemos avaliar os modelos utilizando o RMSE como métrica de erro, sem se preocupar em fazer a transformação inversa da variável, a parcela log já foi aplicada.

```
In [22]: target = np.log1p(target)
```

```
In [23]: fig, axes = plt.subplots(1, 2, figsize=(15, 6))
         sns.distplot(target, fit = norm, ax = axes[0]);

         (mu, sigma) = norm.fit(target)
         axes[0].legend(['Normal dist \n$\mu=$ {:.2f} and $\sigma=$ {:.2f}\nskewness= {:.2f}'
                                                        .format(mu, sigma, skew(target))],
                        loc='best')
         axes[0].set_ylabel('Frequency')
         axes[0].set_title('SalePrice distribution')

         res = stats.probplot(target, plot = axes[1])
         plt.show()
```



Pode-se notar que a transformação torna a distribuição dos dados mais próxima da gaussiana.

Fazemos o mesmo para as demais features que apresentam skewness superior a 1. Para estas features, utilizamos a transformação de Box-Cox, que pode ser considerada uma generalização da transformação log1p.

```
In [24]: skewed_feats = all_data[numerical_cols].apply(lambda x: skew(x.dropna()))\
                                                .sort_values(ascending=False)
         skewness = pd.DataFrame({'Skew' :skewed_feats})
         skewness[skewness['Skew'] > 0.75]
```

Out[24]:

	Skew
MiscVal	21.934924
LowQualFinSF	12.063406
3SsnPorch	11.352135
LandSlope	4.971924
Functional	4.952203
KitchenAbvGr	4.292019
BsmtFinSF2	4.154322
EnclosedPorch	3.994600
BsmtHalfBath	3.951078
ScreenPorch	3.937129
LotArea	3.616277
BsmtFinType2	3.377555
MasVnrArea	2.611175
OpenPorchSF	2.528156
WoodDeckSF	1.844393
ExterCond	1.312569
1stFlrSF	1.261755
BsmtExposure	1.250405
LotFrontage	1.082814
SalePrice	0.992542
GrLivArea	0.983865

```
BsmtFinSF1      0.978732
TotalSF         0.942529
BsmtUnfSF       0.915030
2ndFlrSF        0.840496
ExterQual       0.783284
TotRmsAbvGrd    0.751681
```

```
In [25]: from scipy.special import boxcox1p
         skewed_features = skewness[skewness['Skew'] > 1].index
         lam = 0.15
         for feat in skewed_features:
             all_data[feat] = boxcox1p(all_data[feat], lam)
```

```
In [26]: data_train = pd.concat([all_data[all_data['SalePrice'] > 0]\
                                .drop('SalePrice', axis = 1), target], axis = 1)\
                                .dropna()
         target = data_train['SalePrice']
         data_test = all_data[all_data['SalePrice'] < 0].copy()
```

2.6 Análise das variáveis numéricas

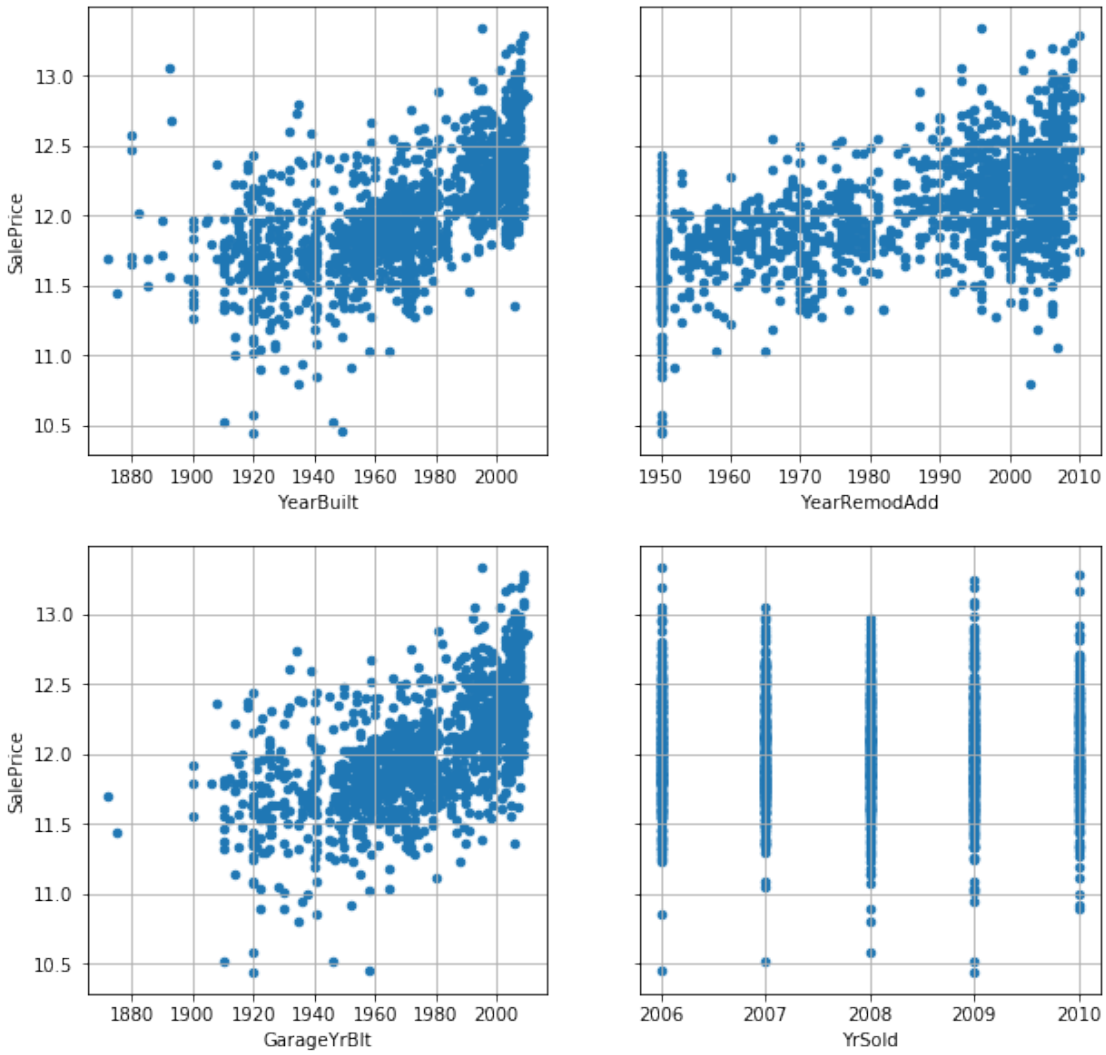
```
In [27]: removed_cols = []
```

2.6.1 Analisando as relações entre *Sales price* e as variáveis temporais:

```
In [28]: colnames = ['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold']
         data = pd.concat([data_train['SalePrice'], data_train[colnames]], axis=1)

         fig, axes = plt.subplots(2, 2, figsize=(10, 10))
         for i in range(len(colnames)):
             data.plot.scatter(x = colnames[i], y = 'SalePrice', ax = axes[i//2, i%2],
                               sharey = True, grid = True)

         abs(data_train[['SalePrice','YearBuilt','YearRemodAdd','GarageYrBlt','YrSold']]\
                                .corr())\
                                .sort_values(by = 'SalePrice', ascending = False)
```

Out[28]:

	SalePrice	YearBuilt	YearRemodAdd	GarageYrBlt	YrSold
SalePrice	1.000000	0.609820	0.576678	0.590544	0.033360
YearBuilt	0.609820	1.000000	0.591399	0.845548	0.013557
GarageYrBlt	0.590544	0.845548	0.605168	1.000000	0.009976
YearRemodAdd	0.576678	0.591399	1.000000	0.605168	0.037351
YrSold	0.033360	0.013557	0.037351	0.009976	1.000000



Realizando uma transformação nas variáveis que representam anos para que passem a representar idade no momento da venda

- newVar = YrSol - oldVar

```
In [29]: data_train['AgeAtSale'] = data_train['YrSold'] - data_train['YearBuilt']
         data_train['AgeRemodAtSale'] = data_train['YrSold'] - data_train['YearRemodAdd']
         data_train['GarageAgeAtSale'] = data_train['YrSold'] - data_train['GarageYrBlt']

         data_test['AgeAtSale'] = data_test['YrSold'] - data_test['YearBuilt']
         data_test['AgeRemodAtSale'] = data_test['YrSold'] - data_test['YearRemodAdd']
         data_test['GarageAgeAtSale'] = data_test['YrSold'] - data_test['GarageYrBlt']

         removed_cols.extend(['YearBuilt', 'YearRemodAdd', 'GarageYrBlt'])
         data_train.drop(['YearBuilt', 'YearRemodAdd', 'GarageYrBlt'], axis = 1, inplace = True)
```

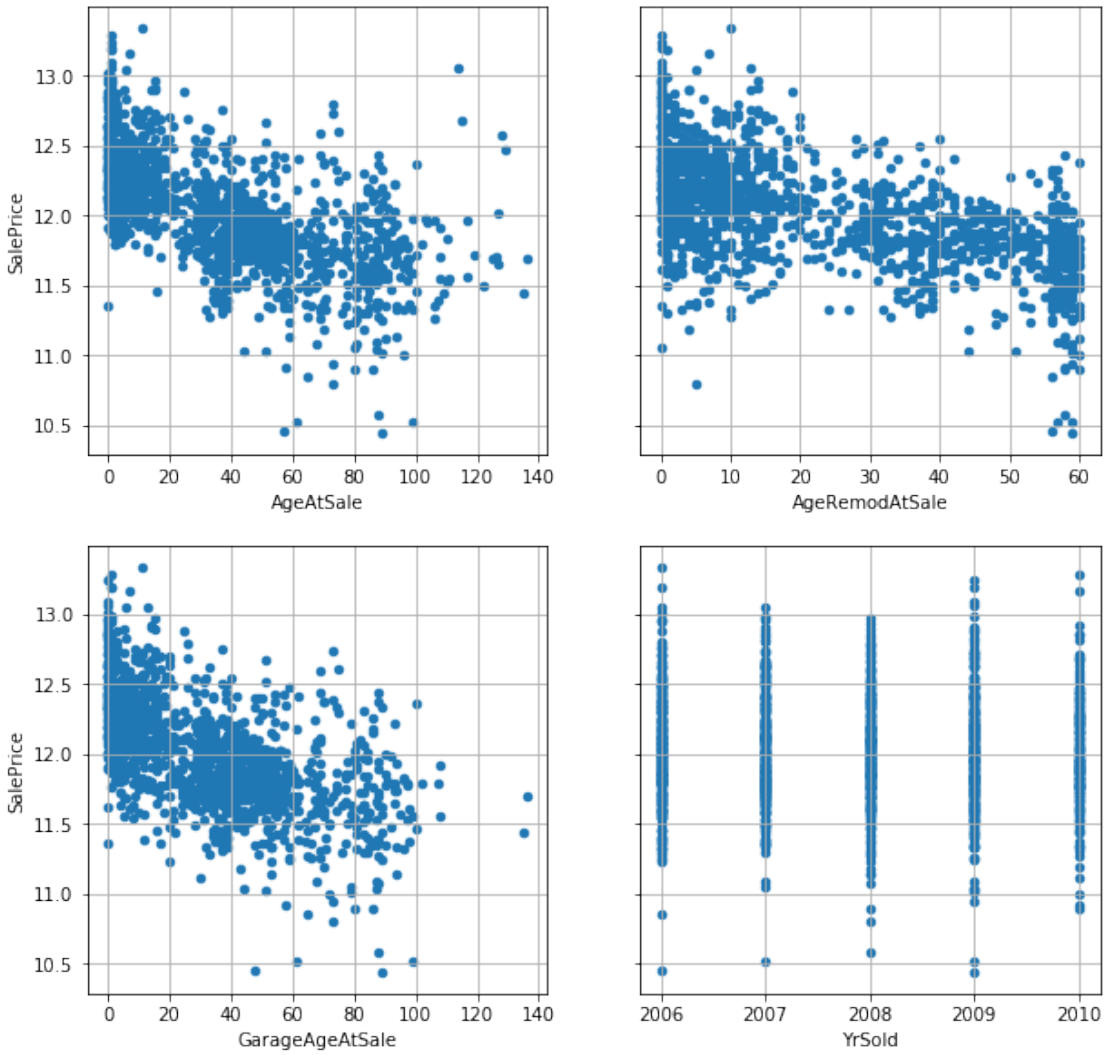
```
data_train.drop(['SalePrice'], axis = 1, inplace = True)
data_train['SalePrice'] = target
```

In [30]:
```
colnames = ['AgeAtSale','AgeRemodAtSale','GarageAgeAtSale','YrSold']
data = pd.concat([data_train['SalePrice'], data_train[colnames]], axis=1)

fig, axes = plt.subplots(2, 2, figsize=(10, 10))
for i in range(len(colnames)):
    data.plot.scatter(x = colnames[i], y = 'SalePrice', ax = axes[i//2, i%2],
                      sharey = True, grid = True)

abs(data_train[['SalePrice','AgeAtSale','AgeRemodAtSale','GarageAgeAtSale','YrSold']]\
                            .corr())\
                            .sort_values(by = 'SalePrice', ascending = False)
```

Out[30]:

	SalePrice	AgeAtSale	AgeRemodAtSale	GarageAgeAtSale	\
SalePrice	1.000000	0.610334	0.579021	0.591178	
AgeAtSale	0.610334	1.000000	0.592737	0.846066	
GarageAgeAtSale	0.591178	0.846066	0.606305	1.000000	
AgeRemodAtSale	0.579021	0.592737	1.000000	0.606305	
YrSold	0.033360	0.057514	0.027173	0.060425	

	YrSold
SalePrice	0.033360
AgeAtSale	0.057514
GarageAgeAtSale	0.060425
AgeRemodAtSale	0.027173
YrSold	1.000000



Pode-se notar que a correlção entre as novas variáveis e o preço de venda é levemente superior (em módulo) do que as variáveis originais.
Pode-se também perceber que a variável YrSold tem baixa correlação SalePrice, logo podemos removê-la do dataset.

```
In [31]: removed_cols.append('YrSold')
         data_train.drop('YrSold', axis = 1, inplace = True)
```

2.6.2 Analisando as variáveis de garagem

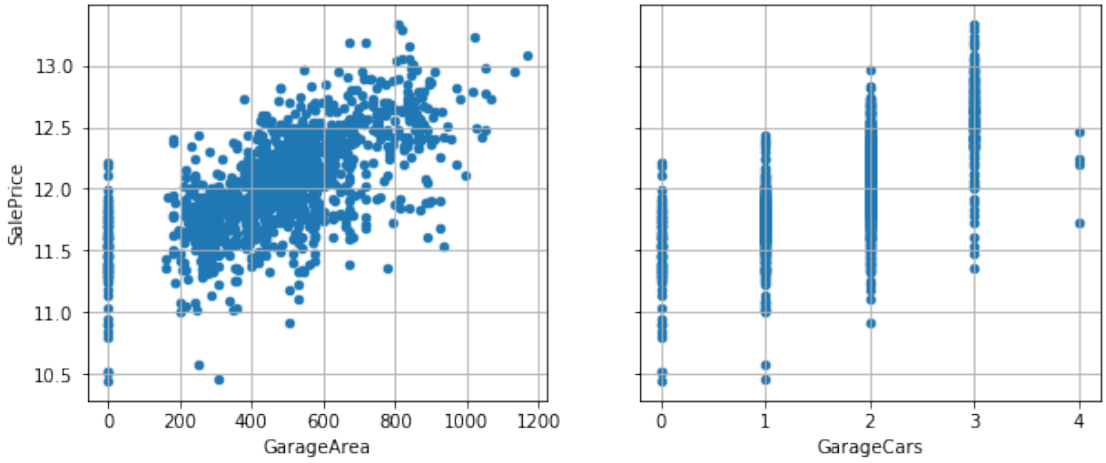
Pela descrição do dataset, 'GarageCars' e 'GarageArea' parecem ser informações redundantes.

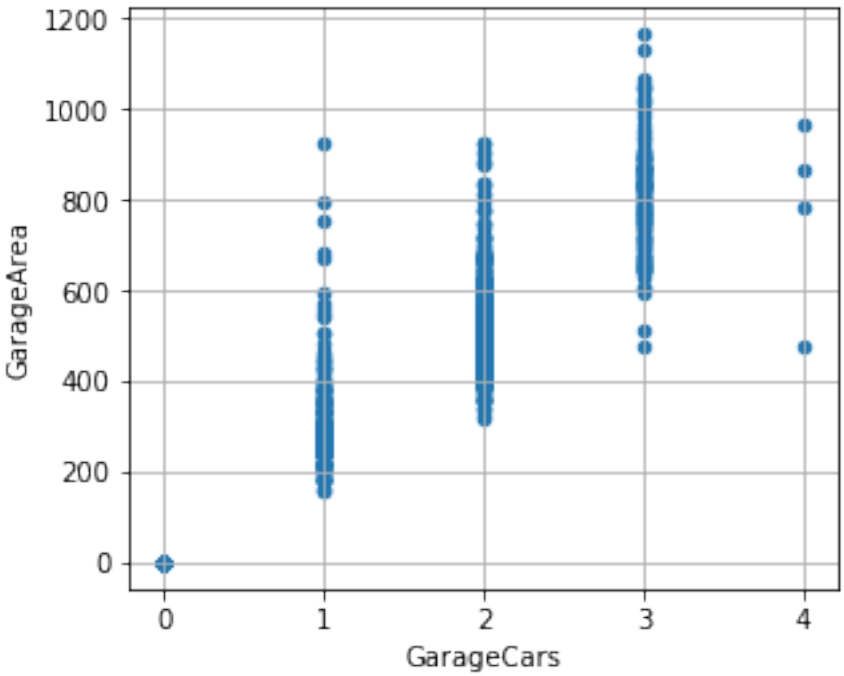
```
In [32]: colnames = ['GarageArea', 'GarageCars']
         data = pd.concat([data_train['SalePrice'], data_train[colnames]], axis=1)

         fig, axes = plt.subplots(1, len(colnames), figsize=(10, 4))
```

```
for i in range(len(colnames)):
    data.plot.scatter(x = colnames[i], y = 'SalePrice', ax = axes[i],
                      sharey = True, grid = True)

data.plot.scatter(x = 'GarageCars', y = 'GarageArea', grid = True,
                  figsize=(4.75, 4))

data_train[['SalePrice','GarageArea','GarageCars']]\
                            .corr()\
                            .sort_values(by = 'SalePrice', ascending = False)
```

Out[32]:

	SalePrice	GarageArea	GarageCars
SalePrice	1.000000	0.668177	0.684840
GarageCars	0.684840	0.889933	1.000000
GarageArea	0.668177	1.000000	0.889933





Podemos ver que *GarageCars* e *GarageArea* apresentam forte correlação entre si. Apesar de *GarageCars* apresentar uma correlação com o preço de venda levemente superior do que *GarageArea* optamos por manter a variável *GarageArea*, por ser contínua, na esperança de que esta se encaixe melhor nos modelos.

```
In [33]: removed_cols.append('GarageCars')
         data_train.drop('GarageCars', axis = 1, inplace = True)
```

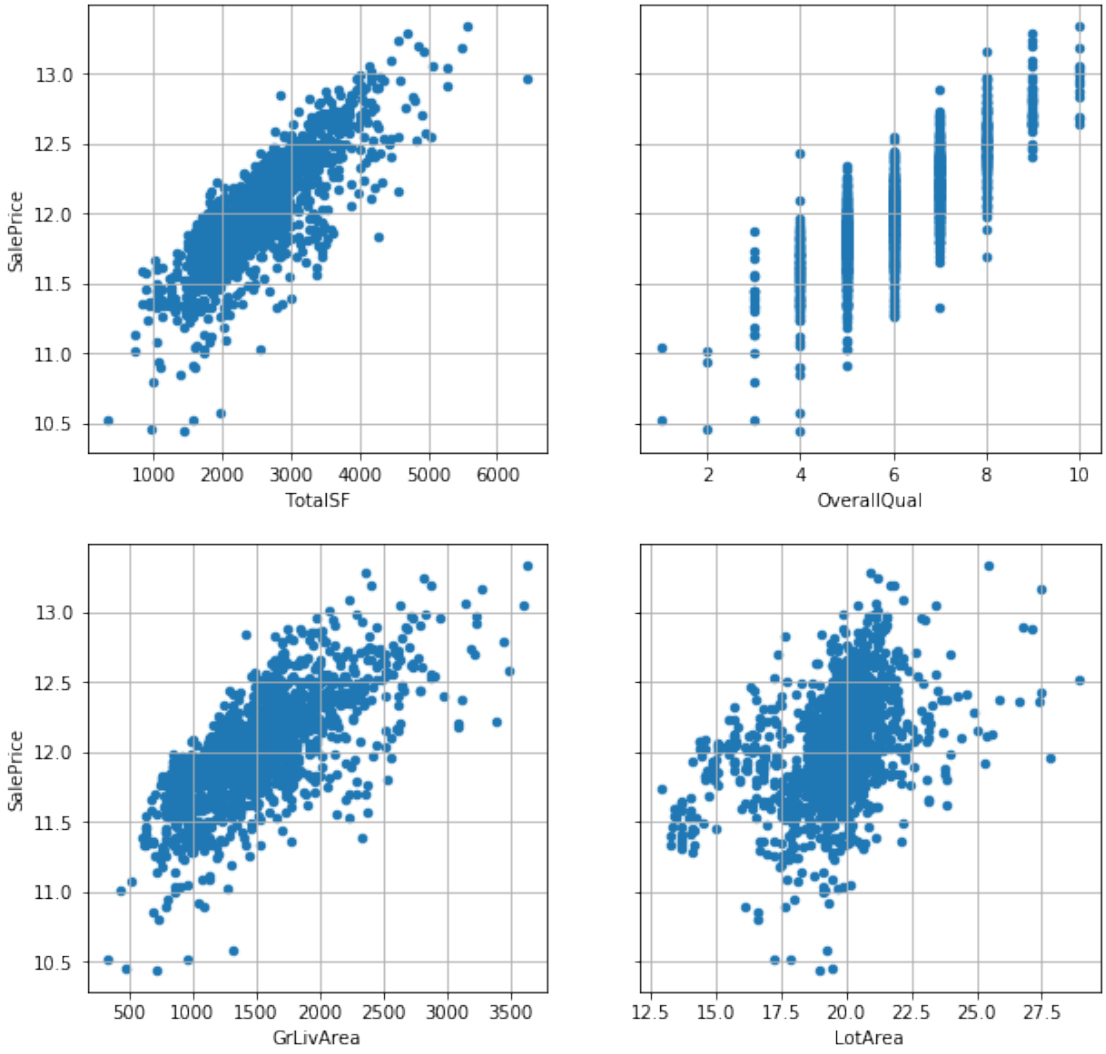
2.6.3 Correlação entre demais variáveis

Visualizando algumas correlações importantes:

```
In [34]: colnames = ['TotalSF','OverallQual','GrLivArea','LotArea']
         data = pd.concat([data_train['SalePrice'], data_train[colnames]], axis=1)

         fig, axes = plt.subplots(2, 2, figsize=(10, 10))
         for i in range(len(colnames)):
             data.plot.scatter(x = colnames[i], y = 'SalePrice', ax = axes[i//2, i%2],
                               sharey = True, grid = True)

         abs(data_train[['SalePrice','TotalSF','OverallQual','GrLivArea','LotArea']]\
                                          .corr())\
                                          .sort_values(by = 'SalePrice', ascending = False)
```

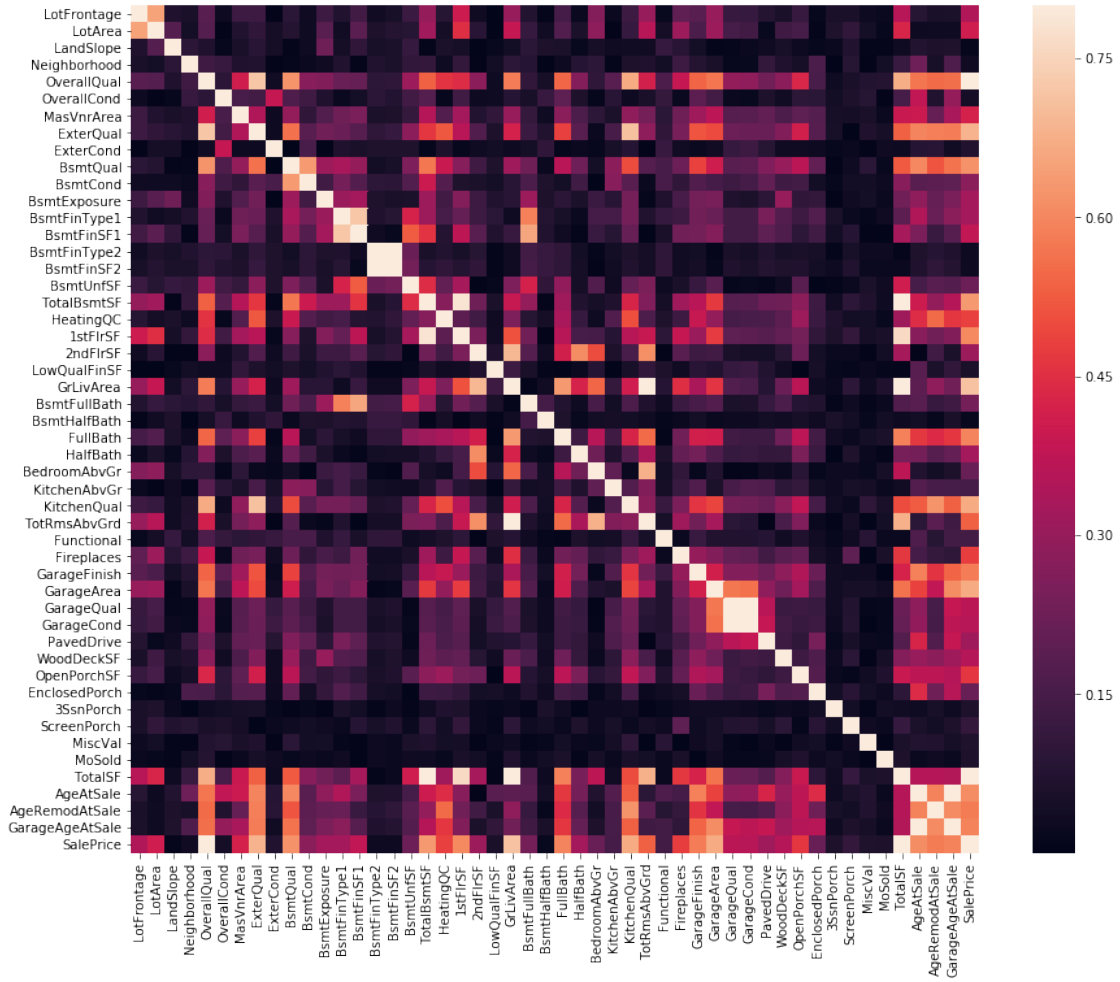
```
Out[34]:              SalePrice   TotalSF  OverallQual  GrLivArea   LotArea
         SalePrice     1.000000  0.815902     0.815105   0.714128  0.409011
```

TotalSF	0.815902	1.000000	0.674258	0.860378	0.428950
OverallQual	0.815105	0.674258	1.000000	0.583466	0.176907
GrLivArea	0.714128	0.860378	0.583466	1.000000	0.388453
LotArea	0.409011	0.428950	0.176907	0.388453	1.000000



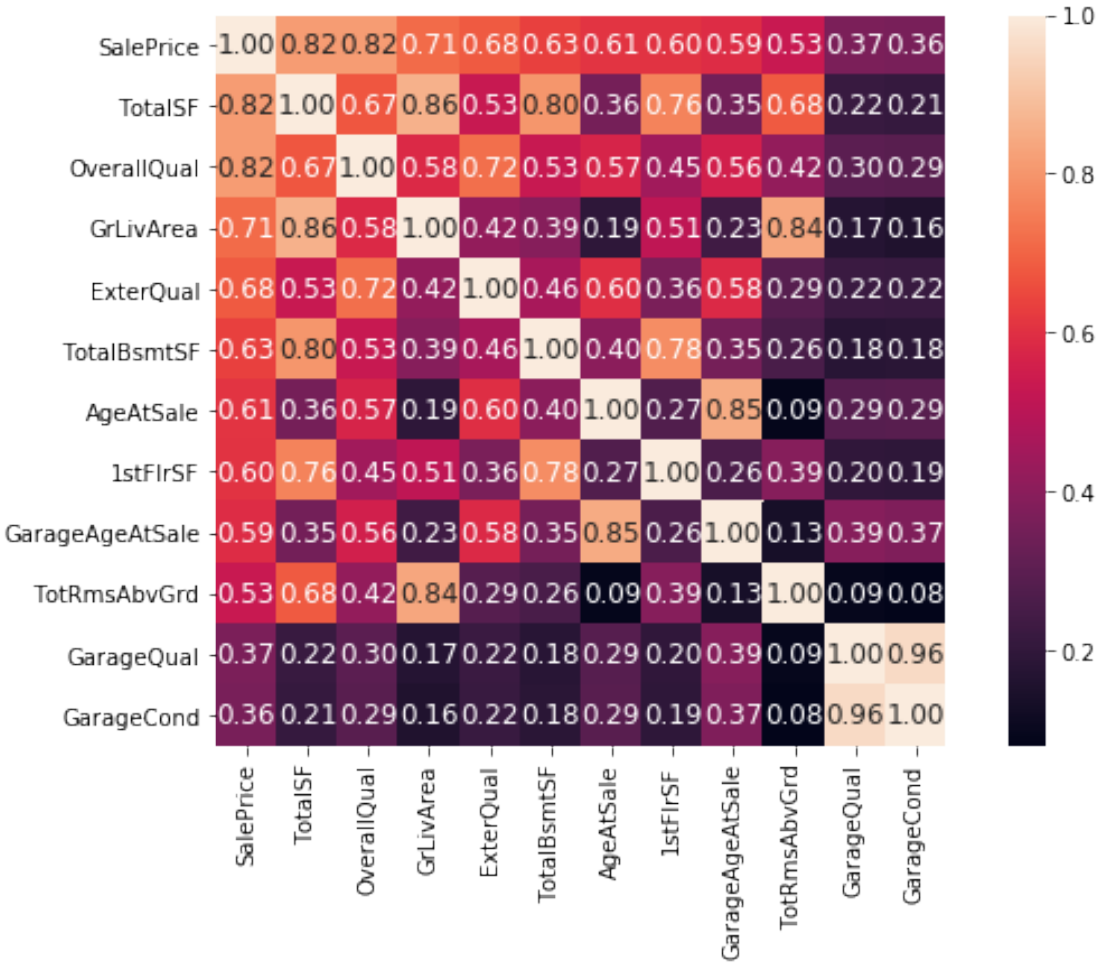
Construindo a matriz de correlação:

```
In [35]: corrmat = abs(data_train.drop('Id', axis = 1).corr())
         f, ax = plt.subplots(figsize=(15, 12))
         sns.heatmap(corrmat, vmax=.8, square=True);
```



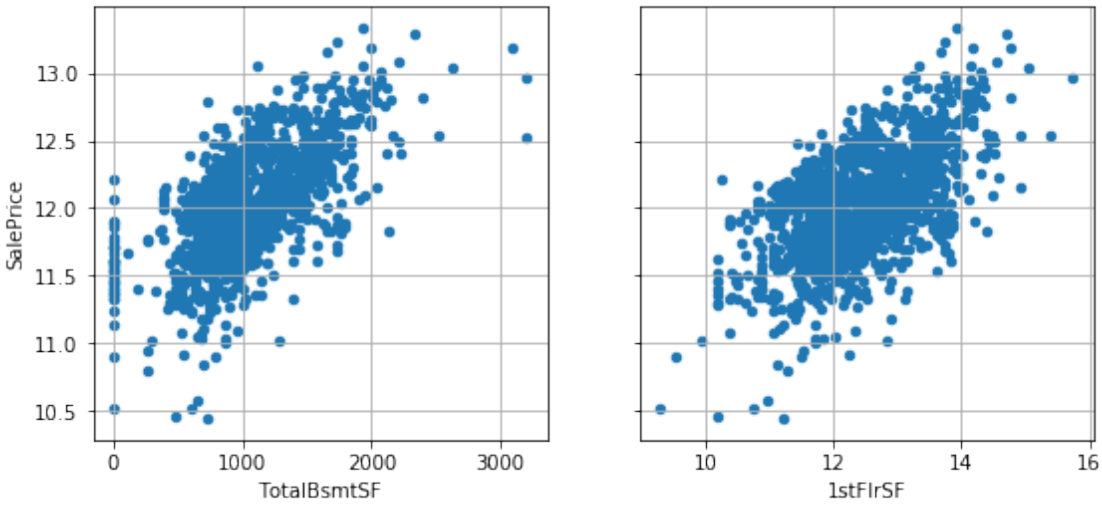
Analisando em detalhes alguns pares de variáveis com forte correlação:

```
In [36]: features = ['SalePrice','AgeAtSale','GarageAgeAtSale','TotalSF',
                    'GrLivArea','TotRmsAbvGrd','TotalBsmtSF','1stFlrSF',
                    'OverallQual','ExterQual','GarageQual','GarageCond']
         fig, ax = plt.subplots(figsize=(10,6))
         corrmat = abs(data_train[features].corr())
         cols = corrmat.nlargest(len(features), 'SalePrice').index
         cm = abs(np.corrcoef(data_train[cols].values.T))
         hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f',
                          annot_kws={'size': 12}, yticklabels=cols.values,
                          xticklabels=cols.values)
```



É possível identificar os pares de variáveis com alta correlação entre si. Podemos remover algumas dessas variáveis para simplificar o problema. Serão removidas as variáveis *TotRmsAbvGrd*, *GarageAgeAtSale* e *GarageCond*, pois apresentam maior correlação com o preço de venda do que a outra variável ao qual estão fortemente correlacionadas. No caso do par (*TotalBsmtSF*, *1stFlrSF*), foi escolhido manter a variável *1stFlrSF*, devido à menor presença de valores 0 (uma casa sempre terá o primeiro andar, mas não necessariamente terá um subsolo), o que pode ser observado nos gráficos abaixo. Por fim, a feature *GrLivArea* pode ser removida para dar lugar à feature artificialmente gerada, *TotalSF*.

```
In [37]: colnames = ['TotalBsmtSF', '1stFlrSF']
         data = pd.concat([data_train['SalePrice'], data_train[colnames]], axis=1)

         fig, axes = plt.subplots(1, 2, figsize=(9, 4))
         for i in range(len(colnames)):
             data.plot.scatter(x = colnames[i], y = 'SalePrice', ax = axes[i],
                               sharey = True, grid = True)
```




```
In [38]: data_train.drop(['TotRmsAbvGrd', 'GarageAgeAtSale', 'TotalBsmtSF',
                          'GrLivArea', 'GarageCond'],
                         axis = 1, inplace = True)
         removed_cols.extend(['TotRmsAbvGrd', 'GarageAgeAtSale', 'TotalBsmtSF',
                              'GrLivArea', 'GarageCond'])
```

Serão removidas também quaisquer variáveis que apresentem correlação menor do que 10% com o preço de venda.

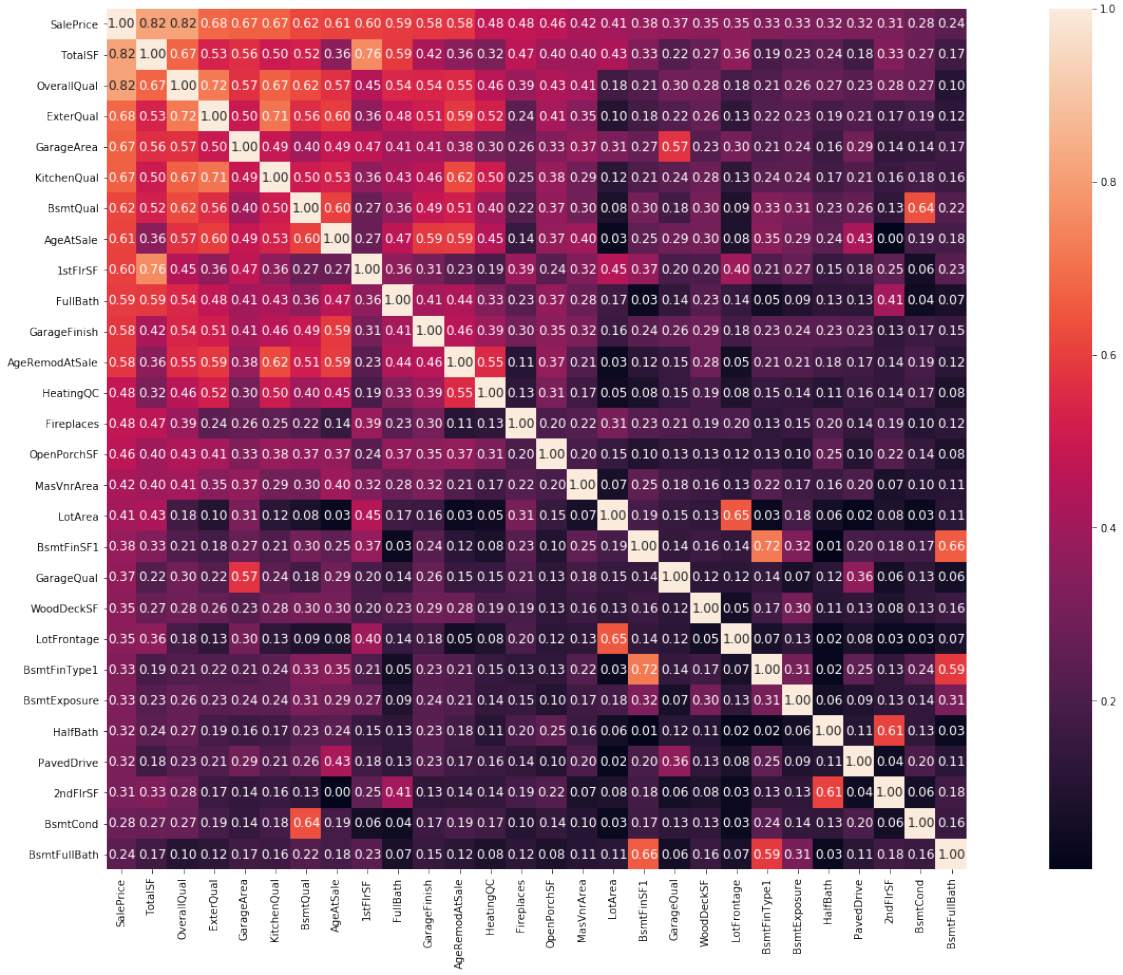
```
In [39]: neighborhood_train = data_train['Neighborhood']
         variables_correlation = abs(data_train.drop('Id', axis = 1)\
                                     .corr()['SalePrice'])\
                                     .sort_values(ascending = False)
         data_train.drop(variables_correlation[variables_correlation <= 0.1].index,
                         axis = 1, inplace = True)
         removed_cols.extend(variables_correlation[variables_correlation <= 0.1].index)

         print(str(len(data_train.drop('Id', axis = 1)
                       .select_dtypes(include=['int64', 'float64']).columns))
                                       + ' features numericas restantes: ')
         print(data_train.drop('Id', axis = 1)\
                   .select_dtypes(include=['int64', 'float64']).columns.values)
```

```
34 features numericas restantes:
['LotFrontage' 'LotArea' 'OverallQual' 'MasVnrArea' 'ExterQual' 'BsmtQual'
 'BsmtCond' 'BsmtExposure' 'BsmtFinType1' 'BsmtFinSF1' 'BsmtUnfSF'
 'HeatingQC' '1stFlrSF' '2ndFlrSF' 'BsmtFullBath' 'FullBath' 'HalfBath'
 'BedroomAbvGr' 'KitchenAbvGr' 'KitchenQual' 'Functional' 'Fireplaces'
 'GarageFinish' 'GarageArea' 'GarageQual' 'PavedDrive' 'WoodDeckSF'
 'OpenPorchSF' 'EnclosedPorch' 'ScreenPorch' 'TotalSF' 'AgeAtSale']
```

```
 'AgeRemodAtSale' 'SalePrice']
```

Visualizando a correlação entre as features numéricas restantes:

```
In [40]: fig, ax = plt.subplots(figsize=(28,15))
         k = 28
         corrmat = abs(data_train.drop('Id', axis = 1).corr())
         cols = corrmat.nlargest(k, 'SalePrice').index
         cm = abs(np.corrcoef(data_train[cols].values.T))
         sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f',
                     annot_kws={'size': 12}, yticklabels=cols.values,
                     xticklabels=cols.values);
```




```
In [41]: numerical_cols = data_train.drop(['Id', 'SalePrice'], axis = 1)\
                                    .select_dtypes(include=['int64', 'float64'])\
                                    .columns.values
```

```
        categorical_cols = data_train.select_dtypes(exclude=['int64', 'float64'])\
                                .columns.values
        print(str(len(numerical_cols)) + ' numerical features remaining: ')
        print(numerical_cols)
```

```
33 numerical features remaining:
['LotFrontage' 'LotArea' 'OverallQual' 'MasVnrArea' 'ExterQual' 'BsmtQual'
 'BsmtCond' 'BsmtExposure' 'BsmtFinType1' 'BsmtFinSF1' 'BsmtUnfSF'
 'HeatingQC' '1stFlrSF' '2ndFlrSF' 'BsmtFullBath' 'FullBath' 'HalfBath'
 'BedroomAbvGr' 'KitchenAbvGr' 'KitchenQual' 'Functional' 'Fireplaces'
 'GarageFinish' 'GarageArea' 'GarageQual' 'PavedDrive' 'WoodDeckSF'
 'OpenPorchSF' 'EnclosedPorch' 'ScreenPorch' 'TotalSF' 'AgeAtSale'
 'AgeRemodAtSale']
```

```
In [42]: neighborhood_test = data_test['Neighborhood']
         data_test.drop(removed_cols, axis = 1, inplace = True)
         all_data = pd.concat((data_train.drop(['Id', 'SalePrice'], axis = 1),
                               data_test.drop(['Id', 'SalePrice'], axis = 1)))
         print('Train shape: ' + str(data_train.drop(['Id', 'SalePrice'], axis = 1).shape))
         print('Test shape: ' + str(data_test.drop(['Id', 'SalePrice'], axis = 1).shape))
```

```
Train shape: (1448, 56)
Test shape: (1459, 56)
```

2.7 Tratamento dos dados categóricos

```
In [43]: all_data = pd.concat((data_train, data_test))
         print('O dataset contém {} variaveis categoricas: '
                               .format(len(categorical_cols)))
         print(categorical_cols)
```

```
O dataset contém 23 variaveis categoricas:
['MSSubClass' 'MSZoning' 'Street' 'LotShape' 'LandContour' 'Utilities'
 'LotConfig' 'Condition1' 'Condition2' 'BldgType' 'HouseStyle' 'RoofStyle'
 'RoofMatl' 'Exterior1st' 'Exterior2nd' 'MasVnrType' 'Foundation'
 'Heating' 'CentralAir' 'Electrical' 'GarageType' 'SaleType'
 'SaleCondition']
```

As features *Condition1* e *Contition2*, assim como *Exterior1st* e *Exterior2nd*, descrevem as mesmas categorias, portanto podem ser codificados nas mesmas variáveis *dummy*. Para o restante das features, podemos utilizar a função *get_dummies* do *pandas*, já que nossos dados já estão devidamente tratados. Utilizando o parâmetro *drop_first = True*, evitamos criar colunas linearmente dependentes nesse processo. As colunas *Exterior1st* e *Exterior2nd* apresentavam alguns erros de grafia em algumas classes, que foram corrigidos como pode ser visto no código abaixo.

```
In [44]: def get_conditions_dummies(data):
             cond1 = pd.get_dummies(data['Condition1'])
             cond2 = pd.get_dummies(data['Condition2'])

             for cond in ('Artery', 'Feedr', 'Norm', 'PosA', 'PosN',
                          'RRAe', 'RRAn', 'RRNe', 'RRNn'):
                 if cond not in cond1.columns:
                     cond1[cond] = 0
                 if cond not in cond2.columns:
                     cond2[cond] = 0

             return (((cond1 + cond2) > 0)*1).drop('RRNe', axis = 1).astype(int)

         def get_exterior_dummies(data):
             data = data[['Exterior1st', 'Exterior2nd']]\
                            .replace({'Brk Cmn': 'BrkComm',
                                      'CmentBd': 'CemntBd',
                                      'Wd Shng': 'WdShing'})
             ext1 = pd.get_dummies(data['Exterior1st'])
             ext2 = pd.get_dummies(data['Exterior1st'])

             for ext in ('AsbShng', 'AsphShn', 'BrkComm', 'BrkFace',
                         'CBlock', 'CemntBd', 'HdBoard', 'ImStucc',
                         'MetalSd', 'Other', 'Plywood', 'Stone',
                         'Stucco', 'VinylSd', 'Wd Sdng', 'WdShing'):
                 if ext not in ext1.columns:
                     ext1[ext] = 0
                 if ext not in ext2.columns:
                     ext2[ext] = 0

             return (((ext1 + ext2) > 0)*1).drop('BrkComm', axis = 1)\
                                           .drop('Other', axis = 1).astype(int)

         condition = get_conditions_dummies(all_data)
         exterior = get_exterior_dummies(all_data)
         all_data = all_data.drop(['Condition1', 'Condition2',
                                   'Exterior1st', 'Exterior2nd'], axis = 1)
         all_data = pd.concat([pd.get_dummies(all_data,drop_first = True),
                               condition, exterior], axis = 1)

In [45]: data_train = all_data[all_data['SalePrice'] > 0].drop('Id', axis = 1)
         data_test = all_data[all_data['SalePrice'] < 0].drop(['Id', 'SalePrice'], axis = 1)
         print('Train shape: ' + str(data_train.shape))
         print('Test shape: ' + str(data_test.shape))

Train shape: (1448, 146)
Test shape: (1459, 145)
```

Terminamos o pré-processamento com 145 features

2.8 Modelos

```
In [46]: from sklearn.preprocessing import RobustScaler
         from sklearn.metrics import mean_squared_error
         from sklearn.model_selection import KFold, cross_val_score, train_test_split
         from sklearn.pipeline import make_pipeline

         if 'SalePrice' in data_train.columns:
             data_train.drop('SalePrice', axis = 1, inplace = True)
```

Primeiro, os dados serão transformados para que possam servir de entrada aos modelos. Utilizamos o *RobustScaler* para realizar o escalamento, e separamos 25% dos dados para servir de conjunto de validação.
Também preparamos um dataset para salvar os resultados dos modelos treinados.

```
In [47]: x_scaler = RobustScaler()
         y_scaler = RobustScaler()

         x = x_scaler.fit_transform(data_train)
         x_test = x_scaler.transform(data_test)
         y = y_scaler.fit_transform(target.values.reshape(-1, 1))

         x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 0.25)

         results = pd.DataFrame([], columns = ['mean', 'std'])
```

Depois, definimos uma função de validação cruzada para avaliar o desempenho dos modelos.

```
In [48]: #Validation function
         n_folds = 5

         def rmsle_cv(model):
             kf = KFold(n_folds, shuffle=True).get_n_splits(data_train.values)
             rmse= np.sqrt(-cross_val_score(model, data_train.values, target.values.reshape(-1,
                                            scoring="neg_mean_squared_error", cv = kf))
             return(rmse)
```

A seguir, criamos os modelos desejados:

2.8.1 Lasso Regression

```
In [49]: from sklearn.linear_model import Lasso

         lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005))
         score = rmsle_cv(lasso)
         results.loc['Lasso'] = [score.mean(), score.std()]
```

2.8.2 ElasticNet

```
In [50]: from sklearn.linear_model import ElasticNet

         elasticNet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=.9))
         score = rmsle_cv(elasticNet)
         results.loc['ElasticNet'] = [score.mean(), score.std()]
```

2.8.3 Kernel Ridge Regression

```
In [51]: from sklearn.kernel_ridge import KernelRidge

         KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
         score = rmsle_cv(KRR)
         results.loc['KRR'] = [score.mean(), score.std()]
```

2.8.4 Gradient Boost

```
In [52]: from sklearn.ensemble import GradientBoostingRegressor

         GBoost = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
                                            max_depth=4, max_features='sqrt',
                                            min_samples_leaf=15, min_samples_split=10,
                                            loss='huber')
         score = rmsle_cv(GBoost)
         results.loc['GBoost'] = [score.mean(), score.std()]
```

2.8.5 XGBoost

```
In [53]: import xgboost as xgb

         XGB = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                                learning_rate=0.05, max_depth=3,
                                min_child_weight=1.7817, n_estimators=2200,
                                reg_alpha=0.4640, reg_lambda=0.8571,
                                subsample=0.5213, silent=1,
                                random_state =7, nthread = -1)
         score = rmsle_cv(XGB)
         results.loc['XGB'] = [score.mean(), score.std()]
```

2.8.6 LGBoost

```
In [54]: import lightgbm as lgb

         LGB = lgb.LGBMRegressor(objective='regression',num_leaves=5,
                                 learning_rate=0.05, n_estimators=720,
                                 max_bin = 55, bagging_fraction = 0.8,
                                 bagging_freq = 5, feature_fraction = 0.2319,
                                 feature_fraction_seed=9, bagging_seed=9,
```

```
                                    min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
score = rmsle_cv(LGB)
results.loc['LGB'] = [score.mean(), score.std()]
```

2.8.7 Bayesian Ridge

```
In [55]: from sklearn.linear_model import BayesianRidge

         BayRidge = BayesianRidge()
         score = rmsle_cv(BayRidge)
         results.loc['BayRidge'] = [score.mean(), score.std()]
```

2.8.8 Support Vector Regression

```
In [56]: from sklearn.svm import SVR

         svr = SVR(kernel='rbf', degree=3, gamma='auto', coef0=0.0,
                   tol=0.001, C=1.0, epsilon=0.1, shrinking=True,
                   cache_size=200, verbose=False, max_iter=-1)
         score = rmsle_cv(svr)
         results.loc['SVR'] = [score.mean(), score.std()]
```

2.8.9 Random Forest Regressor

```
In [57]: from sklearn.ensemble import RandomForestRegressor

         RFR = RandomForestRegressor(n_estimators = 100)
         score = rmsle_cv(RFR)
         results.loc['RandomForest'] = [score.mean(), score.std()]
```

Visualizando o sresultados de cada modelo:

```
In [58]: results.sort_values(by = 'mean')
```

Out[58]:

	mean	std
ElasticNet	0.115362	0.002230
Lasso	0.115723	0.002242
BayRidge	0.116072	0.002182
LGB	0.120155	0.005095
XGB	0.121937	0.003847
GBoost	0.122609	0.005987
RandomForest	0.138791	0.004747
KRR	0.250258	0.024629
SVR	0.389949	0.015000

Pode-se notar que o KRR e o SVR apresentaram um desempenho consideravelmente inferior quando comparados aos demais modelos.

2.8.10 Ensemble de modelos

Treinamos um perceptron para atuar como combinador linear dos resultados dos melhores modelos

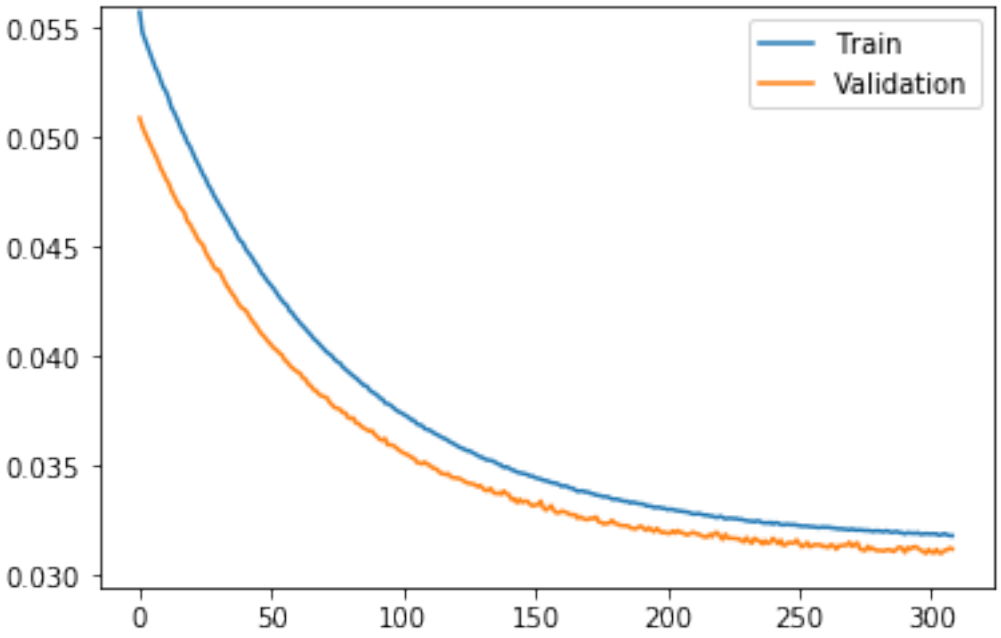
```
In [59]: from keras.models import Sequential
         from keras.layers import Dense
         from keras.callbacks import EarlyStopping
         from keras import backend as K

         class Ensemble():
             def __init__(self, models):
                 K.clear_session()
                 self.models = models
                 self.ensemble_model = Sequential([Dense(1, activation = 'linear',
                                                         input_dim = len(models))
                                                   ])
                 self.ensemble_model.compile(optimizer='sgd', loss='mse')
                 self.early_stopping = EarlyStopping(monitor='val_loss', patience=10)
                 self.x_scaler = RobustScaler()
                 self.y_scaler = RobustScaler()

             def fit(self, x, y):
                 x = self.x_scaler.fit_transform(x.values)
                 y = self.y_scaler.fit_transform(y.values.reshape(-1, 1))
                 predictions = []
                 i = 0
                 for split in KFold(len(self.models), shuffle=True).split(x):
                     self.models[i].fit(x[split[0]], y[:, 0][split[0]])
                     predictions.append(self.models[i].predict(x))
                     i += 1
                 predictions_reshape = [[]]*len(x)
                 for i in range(len(x)):
                     aux = [];
                     for j in range(len(predictions)):
                         aux.append(predictions[j][i])
                     predictions_reshape[i] = aux
                 predictions_reshape = np.array(predictions_reshape)

                 history = self.ensemble_model.fit(predictions_reshape, y[:, 0], epochs = 1000,
                                                   validation_split = 0.25, verbose = 0,
                                                   callbacks = [self.early_stopping])
                 plt.plot(history.history['loss'], label = 'Train')
                 plt.plot(history.history['val_loss'], label = 'Validation')
                 plt.ylim(0.95*min(min(history.history['loss']), min(history.history['val_loss']
                          1.1*max(history.history['val_loss']))
                 plt.legend()
                 plt.show()
```

```
            def predict(self, x):
                x = self.x_scaler.transform(x.values)
                predictions = []
                for i in range(len(self.models)):
                    predictions.append(self.models[i].predict(x))
                predictions_reshape = [[]]*len(x)
                for i in range(len(x)):
                    aux = [];
                    for j in range(len(predictions)):
                        aux.append(predictions[j][i])
                    predictions_reshape[i] = aux
                predictions_reshape = np.array(predictions_reshape)

                return y_scaler.inverse_transform(self.ensemble_model.predict(predictions_resha
                               .reshape(1, -1)[0]
```

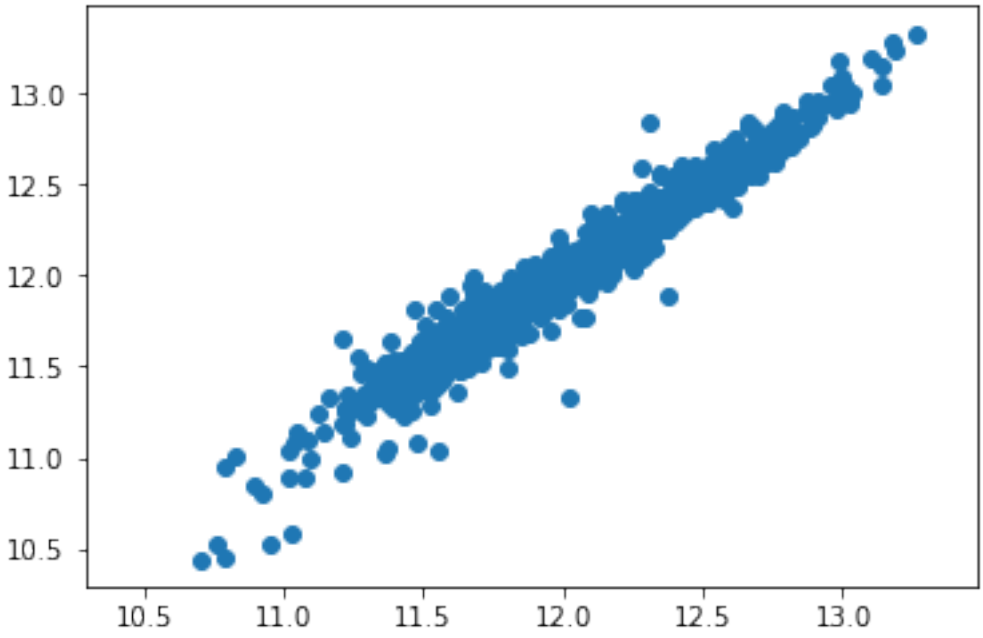
```
Using TensorFlow backend.
```

Treinando e avaliando o ensemble de modelos:

```
In [60]: models = [
            BayesianRidge(),
            Lasso(alpha =0.0005),
            ElasticNet(alpha=0.0005, l1_ratio=.9),
            lgb.LGBMRegressor(objective='regression',num_leaves=5,
                              learning_rate=0.05, n_estimators=720,
                              max_bin = 55, bagging_fraction = 0.8,
                              bagging_freq = 5, feature_fraction = 0.2319,
                              feature_fraction_seed=9, bagging_seed=9,
                              min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
        ]

        ensemble = Ensemble(models)
        ensemble.fit(data_train, target)
        predictions = ensemble.predict(data_train)

        plt.scatter(predictions, target)
        plt.xlim(min(min(predictions), min(target)) - 0.15,
                 max(max(predictions), max(target)) + 0.15)
        plt.ylim(min(min(predictions), min(target)) - 0.15,
                 max(max(predictions), max(target)) + 0.15)
        print('Error: ' + str(np.sqrt(mean_squared_error(predictions, target))))
```




```
Error: 0.08708970055371336
```



Pode-se perceber que a combinação dos modelos melhorou significativamente a métrica de erro avaliada.

2.9 Submissão dos resultados

In [61]:
```
model = ensemble

submission = pd.DataFrame()
submission['Id'] = test_ids

predictions = model.predict(data_test)
final_predictions = np.expm1(predictions) * neighborhood_test/100

submission['SalePrice'] = final_predictions

submission.to_csv('submission.csv', index=False)
```