House Price Prediction

March 4, 2019

```
In [1]: import pandas as pd
        import re
        from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
        from IPython.display import display
        import numpy as np
        import math
        from sklearn import metrics
        from pandas.api.types import is_string_dtype, is_numeric_dtype
        import matplotlib.pyplot as plt
        from sklearn.ensemble import forest
        import scipy
        from scipy.cluster import hierarchy as hc
```

```
In [40]: def rmse(x,y):
             return math.sqrt(((x-y)**2).mean())

         def print_score(m):
             res = [rmse(m.predict(X_train), y_train), rmse(m.predict(X_valid), y_valid),
                         m.score(X_train, y_train), m.score(X_valid, y_valid)]
             if hasattr(m, 'oob_score_'): res.append(m.oob_score_)
             print(res)

         def split_vals(a,n):
             return a[:n].copy(), a[n:].copy()

         def get_oob(df):
             m = RandomForestRegressor(n_estimators=40, min_samples_leaf=5, max_features=0.6, 
             x, _ = split_vals(df, n_trn)
             m.fit(x, y_train)
             return m.oob_score_

         def add_datepart(df, fldname, drop=True, time=False):
             fld = df[fldname]
             fld_dtype = fld.dtype
             if isinstance(fld_dtype, pd.core.dtypes.dtypes.DatetimeTZDtype):
                 fld_dtype = np.datetime64

             if not np.issubdtype(fld_dtype, np.datetime64):
```

```
        df[fldname] = fld = pd.to_datetime(fld, infer_datetime_format=True)
    targ_pre = re.sub('[Dd]ate$', '', fldname)
    attr = ['Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear',
            'Is_month_end', 'Is_month_start', 'Is_quarter_end', 'Is_quarter_start', '
    if time: attr = attr + ['Hour', 'Minute', 'Second']
    for n in attr: df[targ_pre + n] = getattr(fld.dt, n.lower())
    df[targ_pre + 'Elapsed'] = fld.astype(np.int64) // 10 ** 9
    if drop: df.drop(fldname, axis=1, inplace=True)

def train_cats(df):
    for n,c in df.items():
        if is_string_dtype(c): df[n] = c.astype('category').cat.as_ordered()

def fix_missing(df, col, name, na_dict):
    if is_numeric_dtype(col):
        if pd.isnull(col).sum() or (name in na_dict):
            df[name+'_na'] = pd.isnull(col)
            filler = na_dict[name] if name in na_dict else col.median()
            df[name] = col.fillna(filler)
            na_dict[name] = filler
    return na_dict

def proc_df(df, y_fld=None, skip_flds=None, ignore_flds=None, do_scale=False, na_dict
            preproc_fn=None, max_n_cat=None, subset=None, mapper=None):
    if not ignore_flds: ignore_flds=[]
    if not skip_flds: skip_flds=[]
    if subset: df = get_sample(df,subset)
    else: df = df.copy()
    ignored_flds = df.loc[:, ignore_flds]
    df.drop(ignore_flds, axis=1, inplace=True)
    if preproc_fn: preproc_fn(df)
    if y_fld is None: y = None
    else:
        if not is_numeric_dtype(df[y_fld]): df[y_fld] = df[y_fld].cat.codes
        y = df[y_fld].values
        skip_flds += [y_fld]
    df.drop(skip_flds, axis=1, inplace=True)

    if na_dict is None: na_dict = {}
    else: na_dict = na_dict.copy()
    na_dict_initial = na_dict.copy()
    for n,c in df.items(): na_dict = fix_missing(df, c, n, na_dict)
    if len(na_dict_initial.keys()) > 0:
        df.drop([a + '_na' for a in list(set(na_dict.keys()) - set(na_dict_initial.key
    if do_scale: mapper = scale_vars(df, mapper)
    for n,c in df.items(): numericalize(df, c, n, max_n_cat)
    df = pd.get_dummies(df, dummy_na=True)
    df = pd.concat([ignored_flds, df], axis=1)
```

```
    res = [df, y, na_dict]
    if do_scale: res = res + [mapper]
    return res

def numericalize(df, col, name, max_n_cat):
    if not is_numeric_dtype(col) and ( max_n_cat is None or col.nunique()>max_n_cat):
        df[name] = col.cat.codes+1
```

In [41]:
```
#Import training dataset
df_raw = pd.read_csv('train.csv', low_memory=False)
df_test = pd.read_csv('test.csv', low_memory=False)
df_training = df_raw.copy()
```

In [42]:
```
#Change all object variable type to category
train_cats(df_training)
```

In [43]:
```
# Add age column and age of renovation
df_training['house_age'] = df_training['YrSold'] - df_training['YearBuilt']
df_training['renovation_age'] = df_training['YrSold'] - df_training['YearRemodAdd']
```

In [44]:
```
#Change all category to codes+1
cat_cols = list(df_training.select_dtypes(include=['category']).columns)  #Above Usag
for col in cat_cols:
    s = df_training[col]
    df_training[col] = s.cat.codes+1
```

In [45]:
```
#Check for nulls
missing = pd.DataFrame({'missing' : df_training.isnull().sum()})
missing[missing['missing'] > 0]
```

Out[45]:

	missing
LotFrontage	259
MasVnrArea	8
GarageYrBlt	81

In [46]:
```
#Impute missing numeric values
df_training['LotFrontage_nan'] = df_training['LotFrontage'].isnull()
df_training['LotFrontage'].fillna(df_training['LotFrontage'].median(), inplace=True)
df_training['GarageYrBlt_nan'] = df_training['GarageYrBlt'].isnull()
df_training['GarageYrBlt'].fillna(df_training['GarageYrBlt'].median(), inplace=True)
df_training['MasVnrArea_nan'] = df_training['MasVnrArea'].isnull()
df_training['MasVnrArea'].fillna(df_training['MasVnrArea'].median(), inplace=True)
```

In [47]:
```
#Separate out the dependent variable, change to log and remove from training set
y = np.log(df_training['SalePrice'])
df_training = df_training.drop('SalePrice', axis=1)
```

In [48]:
```
# Split training set into training and validation with validation set = 450 rows (30%
n_trn = len(df_training) - 450
X_train, X_valid = split_vals(df_training, n_trn)
y_train, y_valid = split_vals(y, n_trn)
```

In [49]:
```
#Run Randomforest and print score for various values of max_features
for i in ['auto', 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:
    m = RandomForestRegressor(n_estimators=40, max_features=i, n_jobs=-1)
    m.fit(X_train, y_train)
    print(i)
    print_score(m)
```

```
auto
[0.05804377535337503, 0.14478809192555936, 0.9795680014773149, 0.8575423748754701]
0.1
[0.0584725607767841, 0.14259565793062393, 0.9792650128161113, 0.8618240010338682]
0.2
[0.057096058866048405, 0.1421717921260397, 0.9802297661499004, 0.8626442368667252]
0.3
[0.05450041470859265, 0.1400529412751145, 0.9819864563088438, 0.8667078791345444]
0.4
[0.056027712861006516, 0.1413447765975732, 0.9809627009630993, 0.8642375902933759]
0.5
[0.05743592857769177, 0.14054980954941101, 0.9799936972352268, 0.86576043589349]
0.6
[0.05604726605147107, 0.14085889168666654, 0.9809494109368156, 0.8651693760712542]
0.7
[0.05719696963445038, 0.14185871743584158, 0.9801598211301412, 0.8632485095511262]
0.8
[0.056970536650007515, 0.14509867361408482, 0.9803165979132976, 0.8569305540640464]
0.9
[0.058448598891799305, 0.14560507850215773, 0.9792820036086388, 0.8559301659187196]
```

In [51]:
```
# Score for 0.3 seems to be the best
m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
m.fit(X_train, y_train)
print_score(m)
```

```
[0.05550441985941066, 0.13929551743112964, 0.981316653049638, 0.8681457015920236]
```

In [52]:
```
#Feature importance
feature_importance = pd.DataFrame({'Feature' : X_train.columns, 'Importance' : m.featu
feature_importance.sort_values('Importance', ascending=False, inplace=True)
feature_importance.head(30)
```

Out[52]:

	Feature	Importance
17	OverallQual	0.218048
46	GrLivArea	0.122428
80	house_age	0.082197
61	GarageCars	0.063922
27	ExterQual	0.057880
19	YearBuilt	0.042559

```
38     TotalBsmtSF    0.040594
59     GarageYrBlt    0.038449
43        1stFlrSF    0.032462
57     FireplaceQu    0.020832
49        FullBath    0.019517
62      GarageArea    0.018687
4          LotArea    0.017327
34      BsmtFinSF1    0.016835
53     KitchenQual    0.013876
81  renovation_age    0.013740
3      LotFrontage    0.012943
56      Fireplaces    0.011703
18     OverallCond    0.010895
20    YearRemodAdd    0.010213
44        2ndFlrSF    0.009383
54    TotRmsAbvGrd    0.007612
2         MSZoning    0.007369
12    Neighborhood    0.007058
41      CentralAir    0.006830
67     OpenPorchSF    0.005180
37       BsmtUnfSF    0.005105
58      GarageType    0.004733
30        BsmtQual    0.004592
1       MSSubClass    0.004245
```

In [53]:
```python
#Try different cut off points of the importance and check the score
for i in [0.1, 0.04, 0.03, 0.02, 0.01, 0.008, 0.006, 0.005, 0.004, 0.003, 0.002]:
    important_features = feature_importance[feature_importance['Importance'] > i]
    df_important = df_training[important_features['Feature']]

    #Once again create training and validation sets and re-run random forest with oth
    X_train, X_valid = split_vals(df_important, n_trn)
    y_train, y_valid = split_vals(y, n_trn)

    m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
    m.fit(X_train, y_train)
    print(i)
    print_score(m)
```

```
0.1
[0.09761984031686277, 0.22201946564667424, 0.9422069347736767, 0.6650329284181027]
0.04
[0.06547863115728754, 0.15729454180549343, 0.9739984805841476, 0.8318691859684544]
0.03
[0.06421460312947552, 0.16214459199491685, 0.9749926801473395, 0.8213409802414766]
0.02
[0.06498668020150683, 0.15183346226047312, 0.974387719636968, 0.8433411258547472]
0.01
```

```
[0.05583685887434273, 0.14101268824804303, 0.981092178176451, 0.8648747861219768]
0.008
[0.05459894729007144, 0.14023986027818364, 0.9819212632114465, 0.8663518501047337]
0.006
[0.057048790573902115, 0.140520576399062, 0.9802624870900885, 0.8658162714334778]
0.005
[0.057250880556387405, 0.14002608356179996, 0.9801224028397252, 0.8667589966373827]
0.004
[0.05640660795964102, 0.1369101515552299, 0.9807043456296749, 0.8726229129878116]
0.003
[0.05612940414514031, 0.13960534423587234, 0.9808935321836623, 0.867558497795889]
0.002
[0.05617529044953026, 0.13972375561803613, 0.9808622799878887, 0.8673337323000495]
```

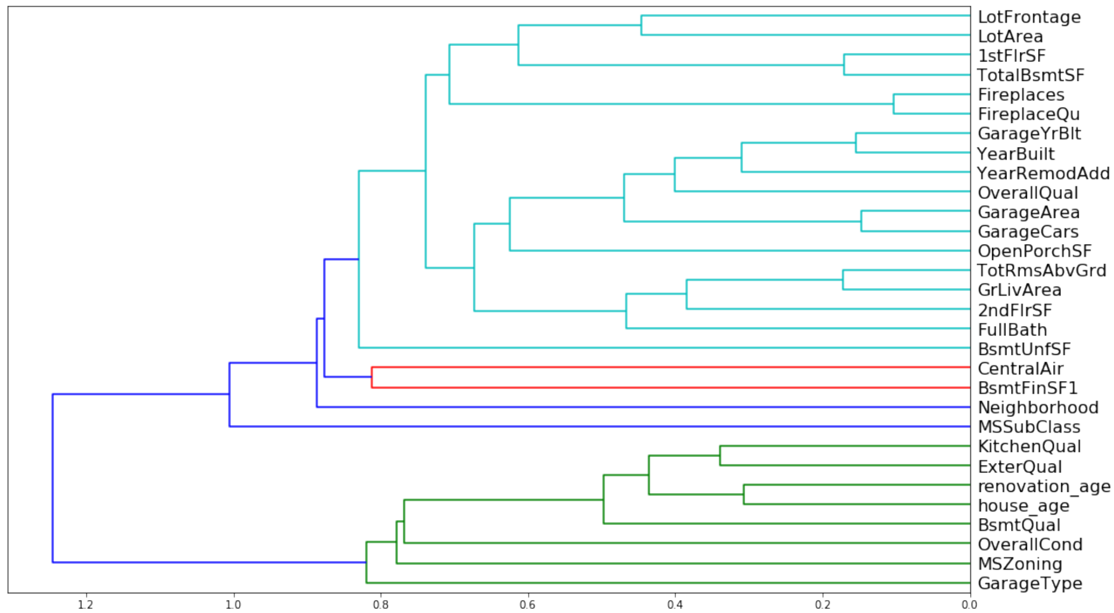
In [54]:
```
#Best seems to be 0.004
important_features = feature_importance[feature_importance['Importance'] > 0.004]
df_important = df_training[important_features['Feature']]

#Once again create training and validation sets and re-run random forest with other p
X_train, X_valid = split_vals(df_important, n_trn)
y_train, y_valid = split_vals(y, n_trn)

m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
m.fit(X_train, y_train)
print(i)
print_score(m)
```

```
0.002
[0.053935470783457555, 0.1366013011044178, 0.9823579726462935, 0.8731969551162856]
```

In [55]:
```
#Find correlated features
#Detect and remove redundant features
#Draw dendogram of feature clusters
corr = np.round(scipy.stats.spearmanr(df_important).correlation, 4)
corr_condensed = hc.distance.squareform(1-corr)
z = hc.linkage(corr_condensed, method='average')
fig = plt.figure(figsize=(16,10))
dendrogram = hc.dendrogram(z, labels=df_important.columns, orientation='left', leaf_fo
plt.show()
```




```
In [59]: cluster_pairs = ['LotFrontage', 'LotArea', '1stFlrSF', 'TotalBsmtSF', 'Fireplaces', 'F
```

```
In [60]: #Run model after dropping each of these columns
         for c in cluster_pairs:
             X_train, X_valid = split_vals(df_important.drop(c, axis=1), n_trn)
             y_train, y_valid = split_vals(y, n_trn)

             m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
             m.fit(X_train, y_train)
             print(c)
             print_score(m)
```

```
LotFrontage
[0.05407909667891936, 0.13497798842937986, 0.9822638888953464, 0.8761927966531273]
LotArea
[0.05637942392400379, 0.13737566346001054, 0.980722939456914, 0.8717552438496639]
1stFlrSF
[0.0559338174902455, 0.13666278294140555, 0.9810264557136001, 0.8730827858000045]
TotalBsmtSF
[0.0551934552691786, 0.13642922496606474, 0.9815254142408512, 0.8735162205255156]
Fireplaces
[0.05802194736529668, 0.13724486799719385, 0.9795833659348433, 0.8719993314358295]
FireplaceQu
[0.05589099514094659, 0.13579830895459052, 0.9810554964892774, 0.8746833622859863]
GarageYrBlt
[0.05468649728092315, 0.13922921295583865, 0.9818632378343305, 0.8682711966476556]
YearBuilt
```

```
[0.054400325475105434, 0.13941796079569552, 0.9820525587865325, 0.8679137949238561]
GarageArea
[0.0534558685096194, 0.13605981626584088, 0.9826703287776958, 0.8742002521344153]
GarageCars
[0.055352721933515744, 0.1425614000126287, 0.9814186395908506, 0.8618903852912191]
TotRmsAbvGrd
[0.05733948119011816, 0.13687100183635076, 0.9800608306694063, 0.8726957500141728]
GrLivArea
[0.057713288178902415, 0.13949997551757945, 0.979800008779835, 0.8677583458030386]
CentralAir
[0.057467760860480326, 0.13727081104269567, 0.9799715152045642, 0.8719509355858198]
BsmtFinSF1
[0.056221314057063494, 0.14046667439140825, 0.9808309086245759, 0.8659191942273795]
KitchenQual
[0.056434449964368937, 0.1386746058375242, 0.9806852924617386, 0.8693185662478626]
ExterQual
[0.05695328503080089, 0.13735034375403463, 0.9803285170267085, 0.8718025130704068]
renovation_age
[0.05584709132830343, 0.1366641996817536, 0.9810852475880254, 0.873080154364198]
house_age
[0.05388067286647761, 0.13776987074136343, 0.9823938026940813, 0.8710181765898186]
```

```
In [61]: to_drop = ['LotFrontage', 'TotalBsmtSF', 'FireplaceQu', 'YearBuilt', 'GarageArea', 'To
```

```
In [62]: X_train, X_valid = split_vals(df_important.drop(to_drop, axis=1), n_trn)
         y_train, y_valid = split_vals(y, n_trn)

         m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
         m.fit(X_train, y_train)
         print_score(m)
```

```
[0.05835800430712438, 0.13538551775870072, 0.9793461790961167, 0.875444063695862]
```

```
In [63]: df_keep = df_important.drop(to_drop, axis=1)

In [64]: df_keep.shape

Out[64]: (1460, 21)
```

```
In [66]: X_train, X_valid = split_vals(df_keep, n_trn)
         y_train, y_valid = split_vals(y, n_trn)

         m = RandomForestRegressor(n_estimators=160, max_features=0.3, n_jobs=-1)
         m.fit(X_train, y_train)
         print_score(m)
```

```
[0.05195797137929255, 0.13227507751203346, 0.9836279178153159, 0.8811015864920875]
```

```
In [67]: #Now lets apply the rf on training set and predict the test set
         #Separate out the dependent variable
         df_training = df_raw.copy()
         y = np.log(df_training['SalePrice'])
         df_training = df_training.drop('SalePrice', axis=1)

In [88]: df_training.shape, df_test.shape, y.shape

Out[88]: ((1460, 82), (1459, 80), (1460,))

In [69]: #Concatenate training and test and process together
         df_train_test = df_training.append(df_test)

In [71]: df_train_test.shape

Out[71]: (2919, 80)

In [74]: #Change all object variable type to category
         train_cats(df_train_test)
         # Add age column and age of renovation
         df_train_test['house_age'] = df_train_test['YrSold'] - df_train_test['YearBuilt']
         df_train_test['renovation_age'] = df_train_test['YrSold'] - df_train_test['YearRemodA
         #Change all category to codes+1
         cat_cols = list(df_train_test.select_dtypes(include=['category']).columns)  #Above Us
         for col in cat_cols:
             s = df_train_test[col]
             df_train_test[col] = s.cat.codes+1

In [84]: #Check for nulls
         missing = pd.DataFrame({'missing' : df_train_test.isnull().sum()})
         missing_nonzero = missing[missing['missing'] > 0]

In [85]: missing_nonzero.index

Out[85]: Index([], dtype='object')

In [77]: num_cols = list(df_train_test.select_dtypes(include=['number']).columns)  #Above Usag
         #for col in cat_cols:

In [83]: for col in missing_nonzero.index:
             df_train_test[col+'_nan'] = df_train_test[col].isnull()
             df_train_test[col].fillna(df_train_test[col].median(), inplace=True)

In [89]: #Use only columns in df_keep
         df_train_test = df_train_test[df_keep.columns]

In [90]: df_train_test.shape

Out[90]: (2919, 21)
```

In [91]:
```
#Split into training and test
X_train = df_train_test.head(1460)
X_test = df_train_test.tail(1459)
```

In [92]:
```
#Run rf on whole dataset and predict y_test
m = RandomForestRegressor(n_estimators=40, max_features=0.3, n_jobs=-1)
m.fit(X_train, y)
```

Out[92]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=0.3, 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=40, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [94]:
```
pred = m.predict(X_test)
pred_prices = np.exp(pred)
pred_prices
```

Out[94]:
```
array([125324.27362609, 150673.54709622, 179887.43149293, ...,
       159432.13059774, 117066.25938073, 235648.4548463 ])
```

In [95]:
```
df_test.head()
```

Out[95]:
```
     Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0  1461          20       RH         80.0    11622   Pave   NaN      Reg
1  1462          20       RL         81.0    14267   Pave   NaN      IR1
2  1463          60       RL         74.0    13830   Pave   NaN      IR1
3  1464          60       RL         78.0     9978   Pave   NaN      IR1
4  1465         120       RL         43.0     5005   Pave   NaN      IR1

  LandContour Utilities      ...       ScreenPorch PoolArea PoolQC  Fence  \
0         Lvl    AllPub      ...               120        0    NaN  MnPrv
1         Lvl    AllPub      ...                 0        0    NaN    NaN
2         Lvl    AllPub      ...                 0        0    NaN  MnPrv
3         Lvl    AllPub      ...                 0        0    NaN    NaN
4         HLS    AllPub      ...               144        0    NaN    NaN

  MiscFeature MiscVal MoSold  YrSold  SaleType  SaleCondition
0         NaN       0      6    2010        WD         Normal
1        Gar2   12500      6    2010        WD         Normal
2         NaN       0      3    2010        WD         Normal
3         NaN       0      6    2010        WD         Normal
4         NaN       0      1    2010        WD         Normal

[5 rows x 80 columns]
```

In [96]:
```
#Prepare the submission df
df_submit = pd.DataFrame({'Id' : df_test.Id, 'SalePrice' : pred_prices})
```

```
In [98]: df_submit, df_submit.shape
```

```
Out[98]: (        Id      SalePrice
          0     1461  125324.273626
          1     1462  150673.547096
          2     1463  179887.431493
          3     1464  184931.374446
          4     1465  190257.655318
          5     1466  187669.870873
          6     1467  167416.179455
          7     1468  175606.755452
          8     1469  181246.917042
          9     1470  121499.489667
          10    1471  202246.618938
          11    1472   94281.135088
          12    1473   99562.851254
          13    1474  153940.736391
          14    1475  121167.165278
          15    1476  380216.489613
          16    1477  240377.670441
          17    1478  303460.410397
          18    1479  287288.585949
          19    1480  451732.204481
          20    1481  294000.382304
          21    1482  204681.078464
          22    1483  180567.078641
          23    1484  168136.337112
          24    1485  175031.198112
          25    1486  200374.951584
          26    1487  318386.769830
          27    1488  242939.210596
          28    1489  209274.501629
          29    1490  211039.691761
          ...    ...            ...
          1429  2890   78073.653099
          1430  2891  133036.688645
          1431  2892   65376.336695
          1432  2893   99375.637850
          1433  2894   58641.692912
          1434  2895  310870.847885
          1435  2896  280328.322546
          1436  2897  195818.291985
          1437  2898  147978.059581
          1438  2899  228075.100039
          1439  2900  156291.235439
          1440  2901  188352.217982
          1441  2902  204356.630467
          1442  2903  362091.916815
```

```
1443  2904  327557.587045
1444  2905  102716.513498
1445  2906  200862.151793
1446  2907  112676.234972
1447  2908  128097.788562
1448  2909  140846.027189
1449  2910   82892.945410
1450  2911   85481.821454
1451  2912  141570.930247
1452  2913   88765.206858
1453  2914   81818.252893
1454  2915   83659.602453
1455  2916   86910.674027
1456  2917  159432.130598
1457  2918  117066.259381
1458  2919  235648.454846

[1459 rows x 2 columns], (1459, 2))
```

```
In [99]: df_submit.to_csv('submission.csv')
```