# Ames Housing Data

Thursday, 5 April 2018 5:09 PM

## Regression and Classification with the Ames Housing Data

**Github contents**

https://github.com/prickofdeath/ames_housing_data/tree/master

| | Filename | Description |
|---|---|---|
| 1 | Project3_1 | Data load, EDA |
| 2. | Project3_2 | Fixed Features Model using Linear Regression |
| 3. | Project3_3 | Renovatible Features Model using Linear Regression |
| 4. | Project3_4 | Identifying Abnorml sales with KNN |
| 5. | Project3_5 | Identifying Abnorml sales with regit regression |

### Introduction

You have just joined a new "full stack" real estate company in Ames, Iowa. The strategy of the firm is two-fold:

- Own the entire process from the purchase of the land all the way to sale of the house, and anything in between.
- Use statistical analysis to optimize investment and maximize return.

The company is still small, and though investment is substantial the short-term goals of the company are more oriented towards purchasing existing houses and flipping them as opposed to constructing entirely new houses. That being said, the company has access to a large construction workforce operating at rock-bottom prices.

1. Estimating the value of homes from fixed characteristics. Your superiors have outlined this year's strategy for the company:

Develop an algorithm to reliably estimate the value of residential houses based on fixed characteristics. Identify characteristics of houses that the company can cost-effectively change/renovate with their construction team. Evaluate the mean dollar value of different renovations. Then we can use that to buy houses that are likely to sell for more than the cost of the purchase plus renovations.

Your first job is to tackle #1. You have a dataset of housing sale data with a huge amount of features identifying different aspects of the house. The full description of the data features can be found in a separate file:

housing.csv data_description.txt You need to build a reliable estimator for the price of the house given characteristics of the house that cannot be renovated. Some examples include:

The neighborhood Square feet Bedrooms, bathrooms Basement and garage space and many more.

Some examples of things that ARE renovate-able:

Roof and exterior features "Quality" metrics, such as kitchen quality "Condition" metrics, such as condition of garage Heating and electrical components and generally anything you deem can be modified without having to undergo major construction on the house.

**Goals**:

1. Perform any cleaning, feature engineering, and EDA you deem necessary.
2. Be sure to remove any houses that are not residential from the dataset.
3. Identify fixed features that can predict price.
4. Train a model on pre-2010 data and evaluate its performance on the 2010 houses.
5. Characterize your model. How well does it perform? What are the best estimates of price?

**Filter for only residential from mszoning**

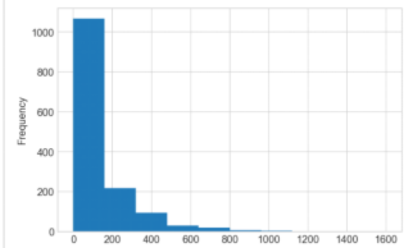
```
'''
MSZoning: Identifies the general zoning classification of the sale.
		
       A	Agriculture
       C	Commercial
       FV	Floating Village Residential
       I	Industrial
       RH	Residential High Density
       RL	Residential Low Density
       RP	Residential Low Density Park
       RM	Residential Medium Density
	
'''
# filter only residential from mszoning
commercial_zoning = ['A','C (all)','I']
house = house[~house['mszoning'].isin(commercial_zoning)]
house['mszoning'].value_counts() #check to ensure only residential remair
```

```
RL    1151
RM     218
FV      65
RH      16
Name: mszoning, dtype: int64
```

**Check for null values**

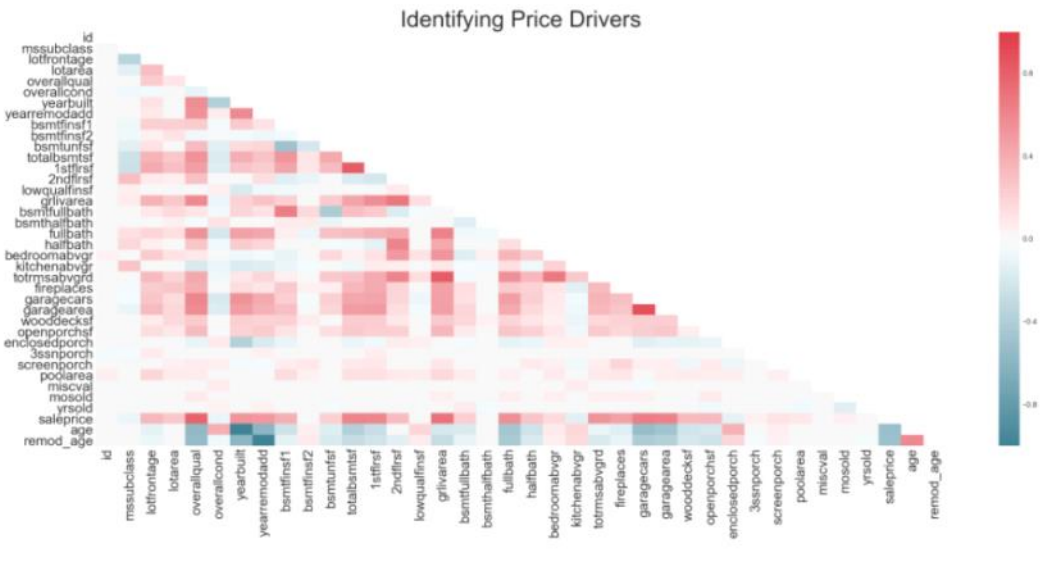
```python
1 house.loc[:,pd.isnull(house).any()].head(5)
```

| | lotfrontage | alley | masvnrtype | masvnrarea | bsmtqual | bsmtcond | bsmtexposure | bsmtfintype |
|---|---|---|---|---|---|---|---|---|
| **0** | 65.0 | NaN | BrkFace | 196.0 | Gd | TA | No | GL( |
| **1** | 80.0 | NaN | None | 0.0 | Gd | TA | Gd | AL( |
| **2** | 68.0 | NaN | BrkFace | 162.0 | Gd | TA | Mn | GL( |
| **3** | 60.0 | NaN | None | 0.0 | TA | Gd | No | AL( |
| **4** | 84.0 | NaN | BrkFace | 350.0 | Gd | TA | Av | GL( |

| Input Variable | fillna Decision | Rationale |
|---|---|---|
| lotfrontage | mean | |
| alley | drop | 89 filled rows vs 1450 total |
| masvnrtype | drop | `count 1442.000000`<br>`mean 104.404300`<br>`std 181.486539`<br>`min 0.000000`<br>`25% 0.000000`<br>`50% 0.000000`<br>`75% 166.750000`<br>`max 1600.000000`<br>`Name: masvnrarea, dtype: float64`<br>`<matplotlib.axes._subplots.AxesSubplot at 0xecae278>`<br> |
| bsmtqual, bsmtcond, ...<br>garage<br>poolqc<br>fence<br>miscfeature | Convert null to 'no' | Blank means no basement |
| electrical | replace with SBrkr | Majority class |
| yearbuilt, remod_age | replace with age and remod_time, | |
| | | |

Part 2

Identify Price Drivers



## Selecting Fixed Features

| Predictor | Description |
|---|---|
| lotfrontage | linear feet of street connected to property |
| lotarea | lot size in sq ft |
| lotconfig | lot config eg Inside Lot, Corner Lot ... |
| neighborhood | physical location |
| bldgtype | type of dwelling |
| housestyle | style of dwelling |
| bsmtqual | height of basement |
| totalbsmtsf | total sq ft of basement area |
| 1stflrsf | |
| 2ndflrsf | |
| grlivarea | above grade ground living area sq ft |
| fullbath | |
| bedroomabvgr | |
| kitchenabvgr | |
| totrmsabvgrd | |
| fireplaces | |
| garagearea | **garagecars** removed, as garagearea and garagecars are highly correlated |
| wooddecksf | |
| openporchsf | |
| enclosedporchsf | |
| poolarea | |
| age | Calculated proxy for yearbuilt |
| remod_age | Calculated proxy for yearremodadd |

**Conversion of the non numeric features**

- Where there is a rating (Ex, Gd, TA…), convert to 5 to 0 scale
- Otherwise pd.get_dummies

In [7]:

```
# Understanding the non numeric features that have been selected

print house_fixed['lotconfig'].unique(),'\n'
print house_fixed['neighborhood'].unique(),'\n'
print house_fixed['bldgtype'].unique(),'\n'
print house_fixed['housestyle'].unique(),'\n'
print house_fixed['bsmtqual'].unique()
```

```
['Inside' 'FR2' 'Corner' 'CulDSac' 'FR3']

['CollgCr' 'Veenker' 'Crawfor' 'NoRidge' 'Mitchel' 'Somerst' 'NWAmes'
 'OldTown' 'BrkSide' 'Sawyer' 'NridgHt' 'NAmes' 'SawyerW' 'IDOTRR'
 'MeadowV' 'Edwards' 'Timber' 'Gilbert' 'StoneBr' 'ClearCr' 'NPkVill'
 'Blmngtn' 'BrDale' 'SWISU' 'Blueste']

['1Fam' '2fmCon' 'Duplex' 'TwnhsE' 'Twnhs']

['2Story' '1Story' '1.5Fin' '1.5Unf' 'SFoyer' 'SLvl' '2.5Unf' '2.5Fin']

['Gd' 'TA' 'Ex' 'No' 'Fa']
```

In [8]:

```
# map qualitative values to numeric values for bsmtqual
convert = {'Ex':5,'Gd':4,'TA':3,'Fa':2,'Po':1,'No':0}
house_fixed['bsmtqual']= house['bsmtqual'].map(convert)


# Create df for the non numeric features
obj_df = house_fixed.select_dtypes(include=['object']).copy()
obj_df.head(4)
```

Out[8]:

| | lotconfig | neighborhood | bldgtype | housestyle |
|---|---|---|---|---|
| 0 | Inside | CollgCr | 1Fam | 2Story |
| 1 | FR2 | Veenker | 1Fam | 1Story |
| 2 | Inside | CollgCr | 1Fam | 2Story |
| 3 | Corner | Crawfor | 1Fam | 2Story |

In [9]:
```python
# binarize the non numeric features
dummy = pd.get_dummies(obj_df, columns=['lotconfig','neighborhood','bldgtype','housestyle'],
                  prefix =['lotconfig','neighbor','bldgtype','housestyle'], drop_first=True)
dummy.head(2)
```

Out[9]:

| | lotconfig_CulDSac | lotconfig_FR2 | lotconfig_FR3 | lotconfig_Inside | neighbor_Blueste | neighbor_BrDale | neighbor_BrkSide | neighbor_ClearCr | neighbor_CollgCr |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

In [10]:
```python
house_fixed = pd.concat([house_fixed,dummy],axis=1)
house_fixed.head(2)

```

Out[10]:

| | lotfrontage | lotarea | lotconfig | neighborhood | bldgtype | housestyle | bsmtqual | totalbsmtsf | 1stflrsf | 2ndflrsf | grlivarea | fullbath | bedroomabvgr | kitchenabvgr | t |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 65.0 | 8450 | Inside | CollgCr | 1Fam | 2Story | 4 | 856 | 856 | 854 | 1710 | 2 | 3 | 1 | |
| 1 | 80.0 | 9600 | FR2 | Veenker | 1Fam | 1Story | 4 | 1262 | 1262 | 0 | 1262 | 2 | 3 | 1 | |

**Define test, train set by year (2010); Standardise prediction matrix**

```python
# define test and train sets

house_fixed_train = house_fixed[house_fixed.yrsold != 2010]
house_fixed_test = house_fixed[house_fixed.yrsold == 2010]
```

```python
X_train = house_fixed_train[col_fixed_2].values
y_train = house_fixed_train['saleprice']

X_test = house_fixed_test[col_fixed_2].values
y_test = house_fixed_test['saleprice']

```

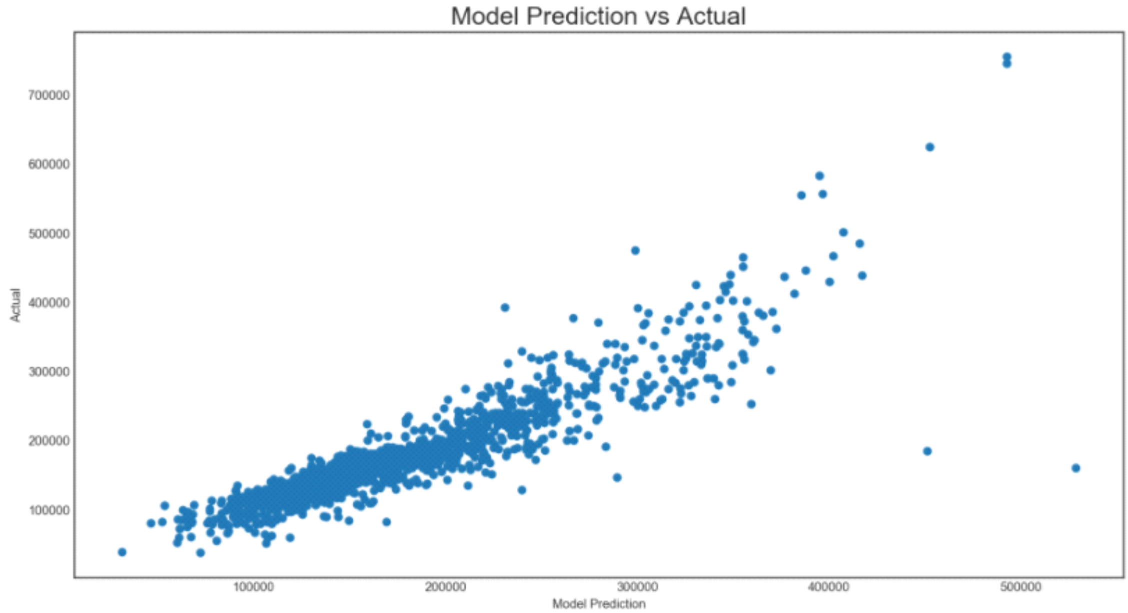
```python
# standardise the test predictor matrix
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
Xs_train = ss.fit_transform(X_train)
print Xs_train.std(), Xs_train.mean()

# standardise the train predictor matrix
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
Xs_test = ss.fit_transform(X_test)
print Xs_test.std(), Xs_test.mean()

```

**Run Linear Regression model**

```python
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error

lm = linear_model.LinearRegression()
model = lm.fit(Xs_train,y_train)

model_coef = model.coef_
predict = model.predict(Xs_train)
score = model.score(Xs_train,y_train)
RMSE_train = (mean_squared_error(y_train, predict))**0.5

print 'score', score,'\n'
print 'RMSE_train', RMSE_train

plt.figure(figsize=(15,8))
plt.scatter(predict, y_train)
plt.title('Model Prediction vs Actual', fontsize=20)
plt.xlabel ('Model Prediction')
plt.ylabel ('Actual')
plt.show()
```

```
score 0.82359878665

RMSE_train 33207.1023972
```

RMSE_train 35207.1025972




**Check the Model Drivers**

```
# checking the model drivers

model_coef = pd.DataFrame(zip(col_fixed_2,model.coef_),columns=['feature','coef']).sort_values(by=['coef'])
print 'Top 20 Model Drivers','\n','\n'
print model_coef.tail(10),'\n'
print model_coef.head(10)

#model_coef.sort_values(by=['coef'])
```

Top 20 Model Drivers

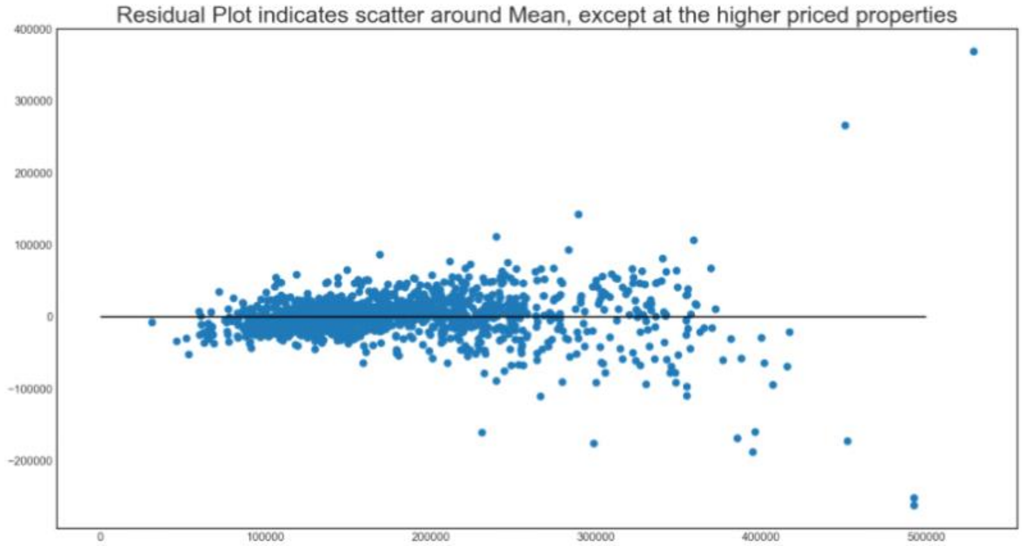
| | feature | coef |
|---|---|---|
| 10 | totrmsabvgrd | 5922.839012 |
| 12 | garagearea | 6640.852661 |
| 4 | 1stflrsf | 7090.827599 |
| 45 | neighbor_StoneBr | 8958.390299 |
| 2 | bsmtqual | 9778.404982 |
| 53 | housestyle_1Story | 9874.225156 |
| 38 | neighbor_NoRidge | 10120.990207 |
| 5 | 2ndflrsf | 12971.737896 |
| 39 | neighbor_NridgHt | 14342.750956 |
| 6 | grlivarea | 22861.668704 |

| | feature | coef |
|---|---|---|
| 51 | bldgtype_TwnhsE | -8605.554716 |
| 18 | age | -8130.602792 |
| 50 | bldgtype_Twnhs | -7616.679995 |
| 19 | remod_age | -7110.883346 |
| 8 | bedroomabvgr | -6692.027508 |
| 9 | kitchenabvgr | -6528.524257 |
| 30 | neighbor_Edwards | -4514.974442 |
| 40 | neighbor_OldTown | -3915.526239 |
| 37 | neighbor_NWAmes | -3389.365781 |
| 31 | neighbor_Gilbert | -3364.008959 |

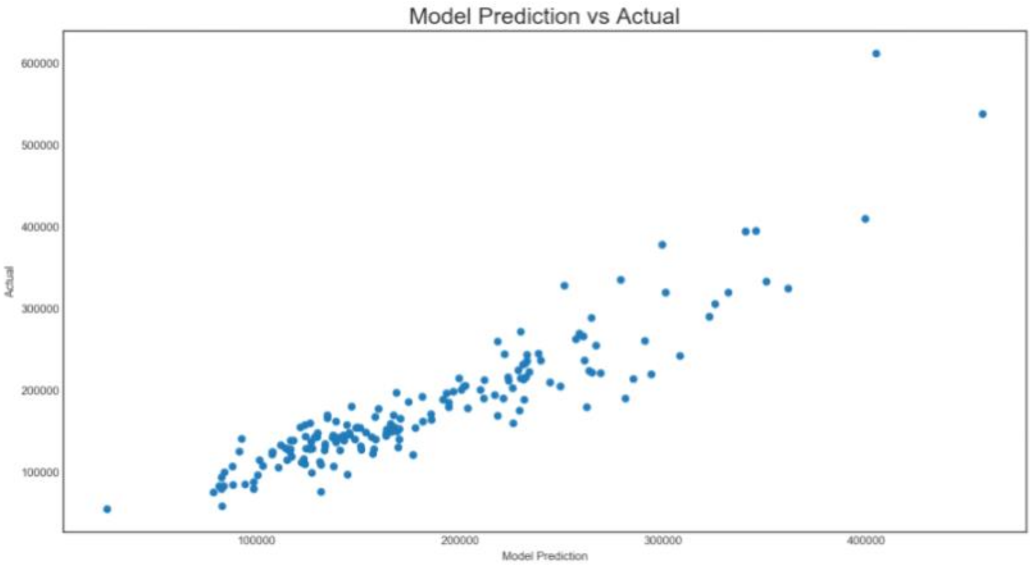
## Price Predictors

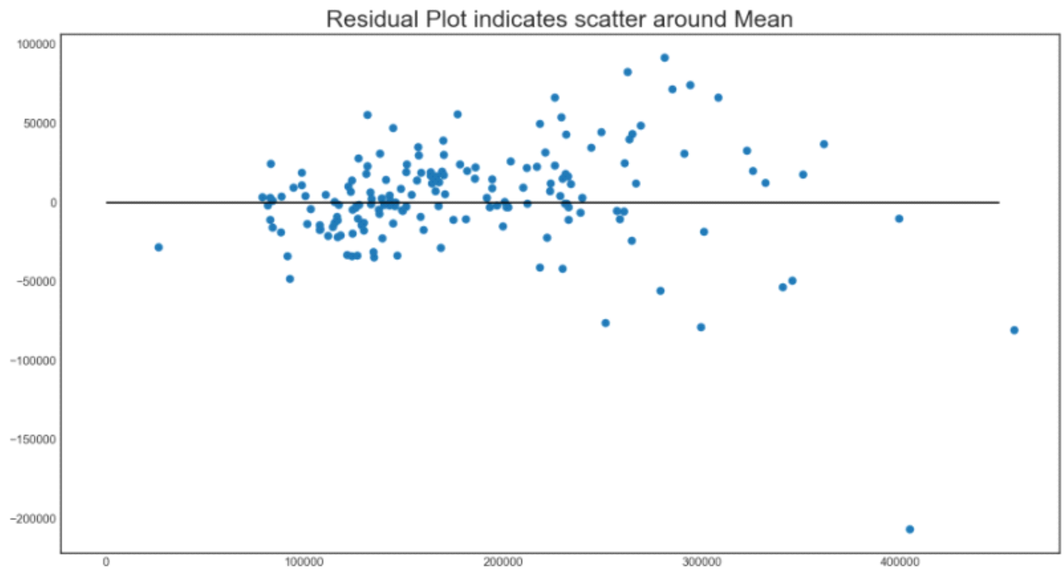
- Features that matter are the neighborhood, area, age, remod age

**Check the Residual Plot to ensure no bias**




**Model vs Predict**




**Part 3 Determine Impact of Renovatable Features**

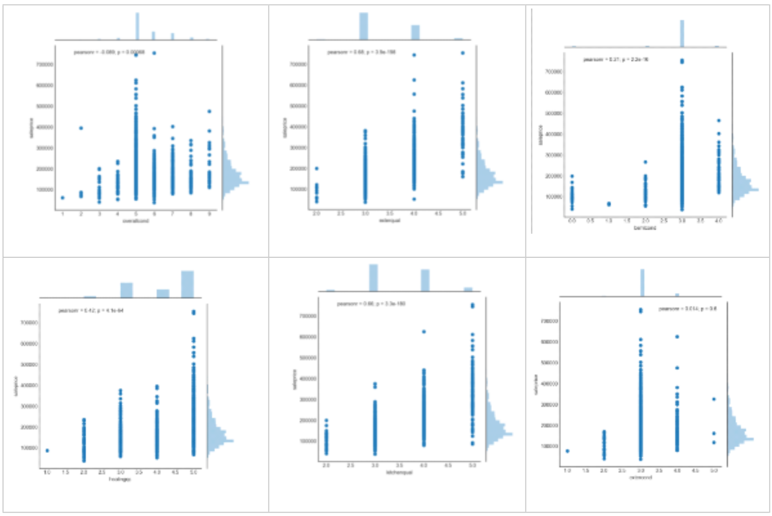
## Selecting Renovatable Features

| Renovatable Predictor | Description |
|---|---|
| overallcond | the overall condition of the house 10: very excellent, 1: very poor |
| exterqual | quality of material on the exterior |
| extercond | present condition of the ext material |
| bsmtcond | general condition of basement |
| heatingqc | heating quality and condition |
| kitchenqual | kitchen quality |

## Not including these as renovatable predictors as they also have a binary condition available or none and therefore not comparable

| Renovatable Predictor | Description |
|---|---|
| fence | fence quality |
| poolqc | pool quality |
| garagecond | garage condition |
| fireplacequ | fireplace quality |

**Reviewing the jointplots**



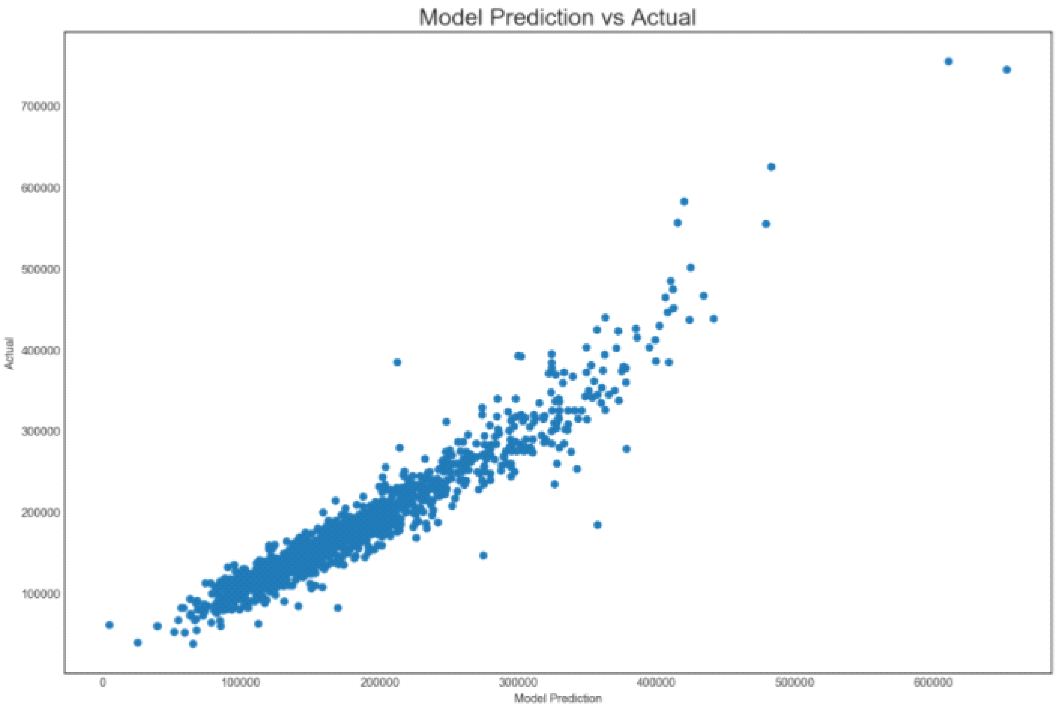
**Binarize non numeric features**

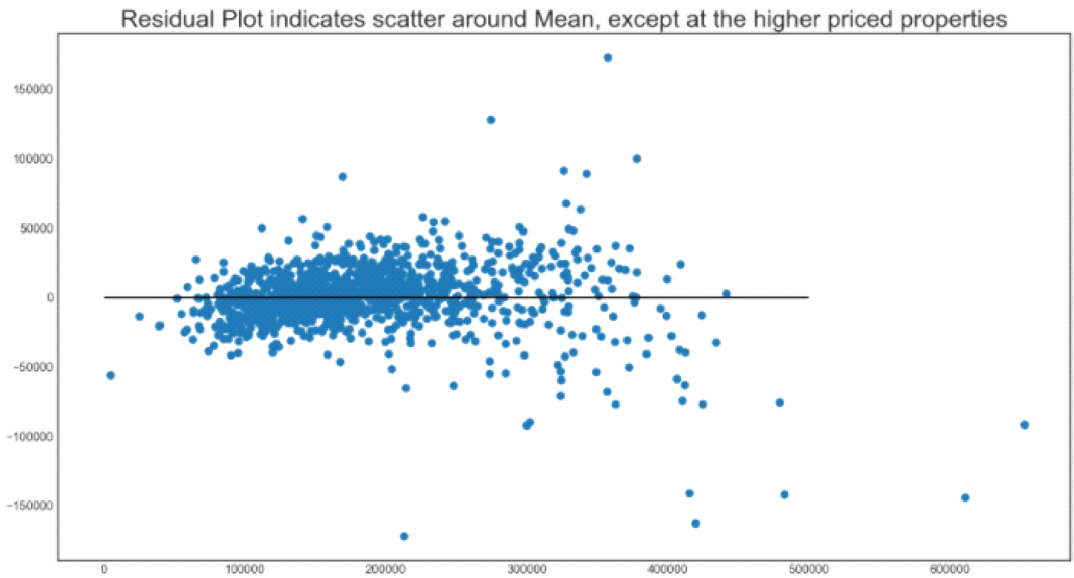
```python
# binarize the non numeric features
dummy = pd.get_dummies(obj_df, columns=col,
                  prefix = col, drop_first=True)
dummy.head(2)

```

**Run the Model for the test-train**

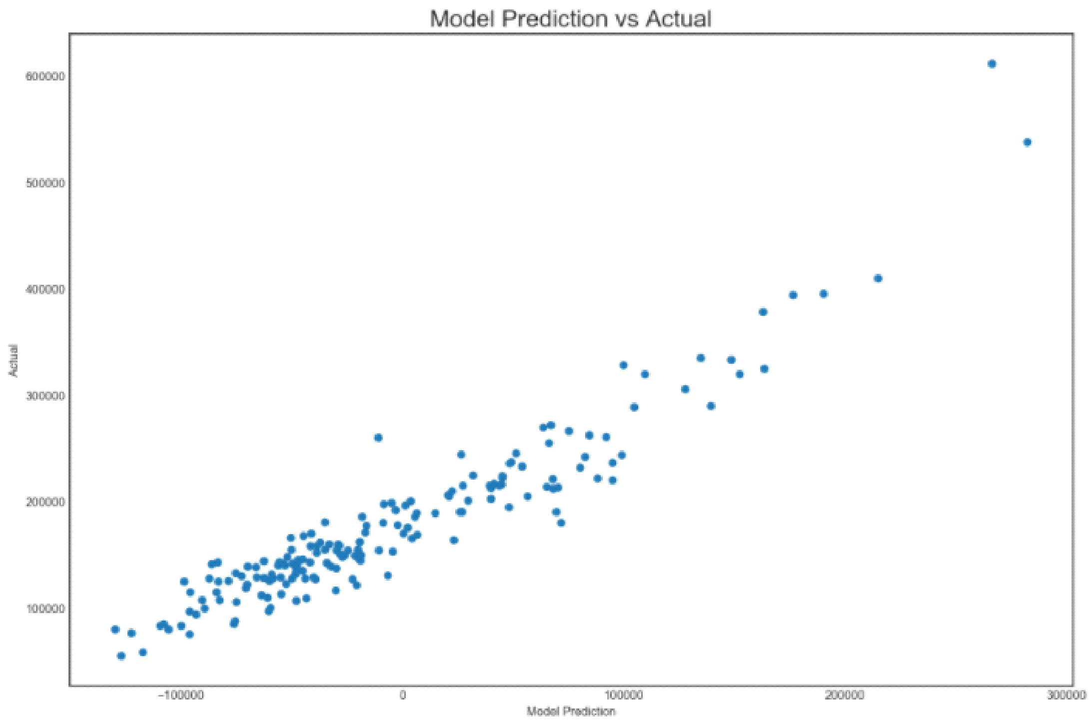
```
lm = linear_model.LinearRegression()
model = lm.fit(Xs_train,y_train)

predict = model.predict(Xs_train)
score = model.score(Xs_train,y_train)
print score

plt.figure(figsize=(15,10))
plt.scatter(predict, y_train)
plt.title('Model Prediction vs Actual', fontsize=20)
plt.xlabel ('Model Prediction')
plt.ylabel ('Actual')
plt.show()
```

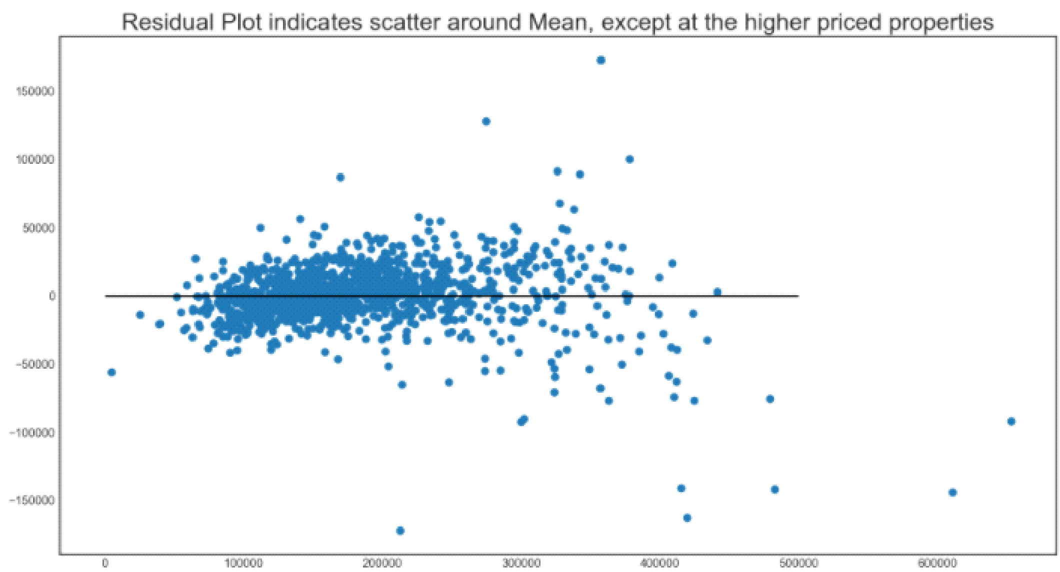
```
0.919029045893
```




**Running the statsmodel**




**The Model now has a score of 91% vs 82% for fixed**


**Commentary**

- Model without the renovatable features have a score of **0.82**
- Model including renovatable features have a score of **0.91,** indicating that renovitable features drive upside to the salesprice
- Objective: Look for undervalued properties, condition < TA
    - Predicted fixed_salesprice > salesprice_actual
    - Predicted fixed+renovatable_salesprice > salesprice_actual

## Part 3a **What Property characteristics predict an 'abnormal' sale?**

The SaleCondition feature indicates the circumstances of the house sale. From the data file, we can see that the possibilities are:

Normal Normal Sale
Abnorml Abnormal Sale - trade, foreclosure, short sale
AdjLand Adjoining Land Purchase
Alloca Allocation - two linked properties with separate deeds, typically condo with a garage unit
Family Sale between family members
Partial Home was not completed when last assessed (associated with New Homes)

One of the executives at your company has an "in" with higher-ups at the major regional bank. His friends at the bank have made him a proposal: if he can reliably indicate what features, if any, predict "abnormal" sales (foreclosures, short sales, etc.), then in return the bank will give him first dibs on the pre-auction purchase of those properties (at a dirt-cheap price).
He has tasked you with determining (and adequately validating) which features of a property predict this type of sale.

**Start with a simple KNN model**

**Encode target class variable salecondition Abnorml: 1, not Abnorml: 0**

```
# encode the target class variable 'salecondition to be a binary 1 if Abnorml, 0 if not Abnorml
house['salecondition'] = house['salecondition'].map(lambda x: 1 if (x == 'Abnorml') else 0)
house['salecondition'].value_counts()
```

```
0    1354
1      96
Name: salecondition, dtype: int64
```

**Choose a subset of house that have a potential impact on salecondition**

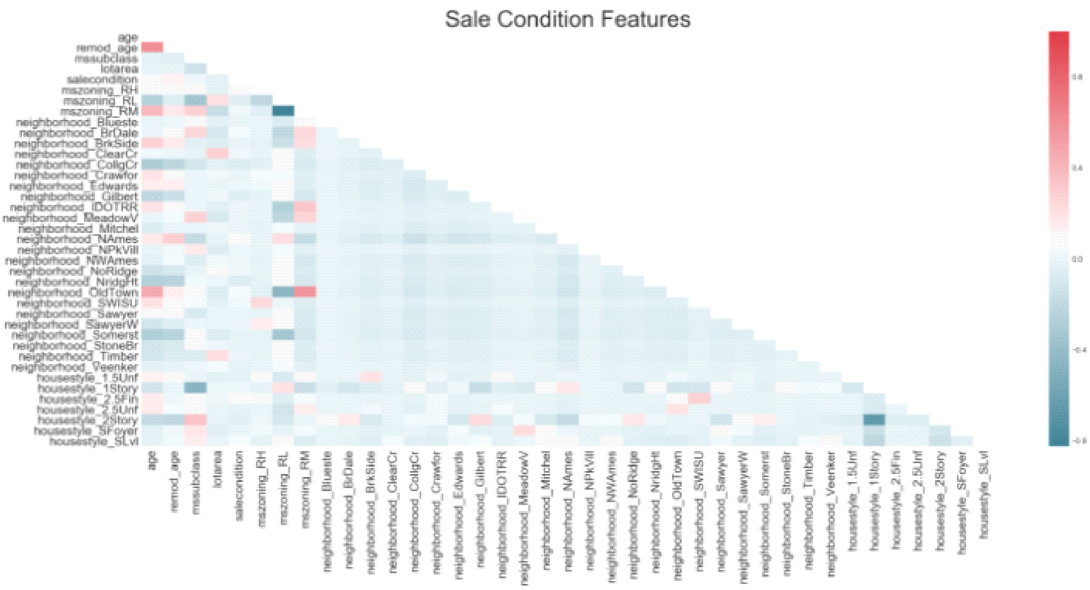
```python
# choose a subset of house that have a potential impact on salecondition
col = ['age', 'remod_age', 'mssubclass', 'mszoning', 'lotarea', 'neighborhood', 'housestyle','salec
h2 = house[col]  #h2 is the house subset
h2.head(4)
```

| | age | remod_age | mssubclass | mszoning | lotarea | neighborhood | housestyle | salecondition |
|---|---|---|---|---|---|---|---|---|
| 0 | 15 | 15 | 60 | RL | 8450 | CollgCr | 2Story | 0 |
| 1 | 42 | 42 | 20 | RL | 9600 | Veenker | 1Story | 0 |
| 2 | 17 | 16 | 60 | RL | 11250 | CollgCr | 2Story | 0 |
| 3 | 103 | 48 | 70 | RL | 9550 | Crawfor | 2Story | 1 |

**Examine correlation structure of the dataset**

```python
# examine correlation structure of the dataset

fig,ax=plt.subplots(figsize=(25,10))
cmap=sns.diverging_palette(220,10, as_cmap=True)

h2_corr=h2.corr()
mask=np.zeros_like(h2_corr,dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

plot_a = sns.heatmap(h2_corr, mask = mask, ax=ax, cmap=cmap)
plot_a.tick_params(labelsize = 16)
plot_a.set_title('Sale Condition Features', fontsize=30)
```

```
Text(0.5,1,u'Sale Condition Features')
```




**Not seeing any correlation with the selected predictors**

**Baseline 0.93**

```python
# baseline accuracy
baseline = 1 - np.mean(y)
print 'baseline:', baseline
```

```
baseline: 0.933793103448
```

**KNN with k = 5, 2, 10**
**Cross validation using StratifiedKFold (class imbalance)**
**Standardised, non-Standardised**
**Non data driven feature selection**
**No class balancing**

Hypothesis#1: mszoning, neighborhood, housestyle
Hypothesis#2: neighborhood only
Conclusion: KNN model does not provide a better outcome than baseline
Best + 0.0002% improvement on baseline

```
# hypothesis: neighborhood is factor
y = h2['salecondition']
X = h2 [['neighborhood_Blueste', 'neighborhood_BrDale', 'neighborhood_BrkSide', 'neighborhood_Clear
        'neighborhood_CollgCr', 'neighborhood_Crawfor', 'neighborhood_Edwards', 'neighborhood_Gilb
        'neighborhood_IDOTRR', 'neighborhood_MeadowV', 'neighborhood_Mitchel', 'neighborhood_NAmes
        'neighborhood_NPkVill', 'neighborhood_NWAmes', 'neighborhood_NoRidge', 'neighborhood_Nridg
        'neighborhood_OldTown', 'neighborhood_SWISU', 'neighborhood_Sawyer', 'neighborhood_SawyerW
        'neighborhood_Somerst', 'neighborhood_StoneBr', 'neighborhood_Timber', 'neighborhood_Veenk
        'neighborhood_Blueste',
        'neighborhood_BrDale', 'neighborhood_BrkSide', 'neighborhood_ClearCr', 'neighborhood_Collg
        'neighborhood_Crawfor', 'neighborhood_Edwards', 'neighborhood_Gilbert', 'neighborhood_IDOT
        'neighborhood_MeadowV', 'neighborhood_Mitchel', 'neighborhood_NAmes', 'neighborhood_NPkVil
        'neighborhood_NWAmes', 'neighborhood_NoRidge', 'neighborhood_NridgHt', 'neighborhood_OldTo
        'neighborhood_SWISU', 'neighborhood_Sawyer', 'neighborhood_SawyerW', 'neighborhood_Somerst
        'neighborhood_StoneBr', 'neighborhood_Timber', 'neighborhood_Veenker', 'neighborhood_Blues
        'neighborhood_BrkSide', 'neighborhood_ClearCr', 'neighborhood_CollgCr', 'neighborhood_Craw
        'neighborhood_Edwards', 'neighborhood_Gilbert', 'neighborhood_IDOTRR', 'neighborhood_Meado
        'neighborhood_Mitchel', 'neighborhood_NAmes', 'neighborhood_NPkVill', 'neighborhood_NWAmes
        'neighborhood_NoRidge', 'neighborhood_NridgHt', 'neighborhood_OldTown', 'neighborhood_SWIS
        'neighborhood_Sawyer', 'neighborhood_SawyerW', 'neighborhood_Somerst', 'neighborhood_Stone
        'neighborhood_Timber', 'neighborhood_Veenker']]

# standardize the predictor matrix
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
Xs = ss.fit_transform(X)

```

```
# cross validate mean accuracy for a KNN with 5 neighbors
knn5 = KNeighborsClassifier(n_neighbors=5, weights='uniform')
scores = accuracy_crossvalidator(Xs, y, knn5, cv_indices)

print 'baseline:', baseline
percent_gain = (((np.mean(scores))/baseline)-1)*100
print 'Percent Gain: ', percent_gain, '%'
```

```
('Fold accuracy:', 0.93127147766323026)
('Fold accuracy:', 0.93103448275862066)
('Fold accuracy:', 0.93103448275862066)
('Fold accuracy:', 0.92758620689655169)
('Fold accuracy:', 0.9307958477508651)
('Mean CV accuracy:', 0.93034449956557774)
baseline: 0.933793103448
Percent Gain:  -0.369311346375 %
```

```
# cross validate mean accuracy for a KNN with 2 neighbors
knn2 = KNeighborsClassifier(n_neighbors=2, weights='uniform')
scores = accuracy_crossvalidator(Xs, y, knn2, cv_indices)

print 'baseline:', baseline
percent_gain = (((np.mean(scores))/baseline)-1)*100
print 'Percent Gain: ', percent_gain, '%'
```

```
('Fold accuracy:', 0.92439862542955331)
('Fold accuracy:', 0.93103448275862066)
('Fold accuracy:', 0.93103448275862066)
('Fold accuracy:', 0.93103448275862066)
('Fold accuracy:', 0.92387543252595161)
('Mean CV accuracy:', 0.92827550124627334)
baseline: 0.933793103448
Percent Gain:  -0.590880590318 %
```

```
# cross validate mean accuracy for a KNN with 10 neighbors
knn10 = KNeighborsClassifier(n_neighbors=10, weights='uniform')
scores = accuracy_crossvalidator(Xs, y, knn10, cv_indices)

print 'baseline:', baseline
percent_gain = (((np.mean(scores))/baseline)-1)*100
print 'Percent Gain: ', percent_gain, '%'
```

```
('Fold accuracy:', 0.93127147766323026)
('Fold accuracy:', 0.93448275862068964)
('Fold accuracy:', 0.93448275862068964)
('Fold accuracy:', 0.93448275862068964)
('Fold accuracy:', 0.93425605536332179)
('Mean CV accuracy:', 0.93379516177772415)
baseline: 0.933793103448
Percent Gain:  0.000220426713438 %
```

**Part 3b What Property characteristics predict an 'abnormal' sale?**

- A different approach to identifying predictors
- Logit regression
- Re-Run KNN

**Use groupby salecondition to find meaningful mean differences that can indicate predictors**

```
# Using groupby 'salecondition' to find meaningful differences that can indicate predictors
house.groupby('salecondition').mean()
```

| salecondition | id | mssubclass | lotfrontage | lotarea | overallqual | overallcond | yearbuilt | yearremodadd | bsmtfins |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 730.500000 | 56.783604 | 70.177844 | 10595.706056 | 6.144756 | 5.591581 | 1972.340473 | 1985.725258 | 447.132 |
| 1 | 743.552083 | 58.593750 | 68.290617 | 9510.104167 | 5.687500 | 5.447917 | 1961.052083 | 1975.520833 | 417.375 |

## Observations for Abnorml

In project3_4, the hypothesis was that age, remod_age, mssubclass, mszoning, lotarea, neighborhood, housestyle determine abnormal sales. Using KNN, it did not appear to be so.

Taking a different approach above to shortlist predictors for salecondition 1 = Abnorml, vs 0 = Not Abnorml, based on the groupby above

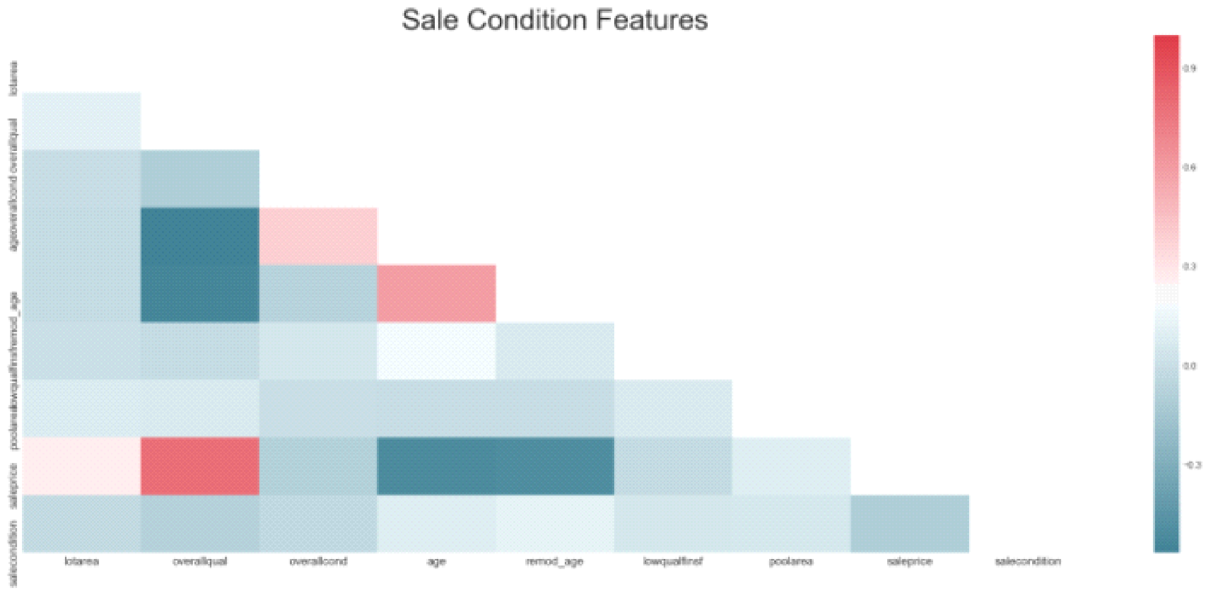
- **lotarea** is smaller
- **overallqual, overallcond** is lower
- **yearbuilt,yearremodadd** is older -> **age, remod_age** is higher
- **lowqualfinsf** is higher
- **poolarea** is higher
- **saleprice** is lower

**Based on above, chose new features that may have a potential impact on salecondition**

**Examine correlations**

```python
# examine correlation structure of the dataset

fig,ax=plt.subplots(figsize=(25,10))
cmap=sns.diverging_palette(220,10, as_cmap=True)

h2_corr=h2.corr()
mask=np.zeros_like(h2_corr,dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

plot_a = sns.heatmap(h2_corr, mask = mask, ax=ax, cmap=cmap)
plot_a.tick_params(labelsize = 12)
plot_a.set_title('Sale Condition Features', fontsize=30)
```

```
Text(0.5,1,u'Sale Condition Features')
```




**Upsample minority class to balance**

```python
# up-sample minority class method to balance; resample with replacement
from sklearn.utils import resample

# seperate majority and minority classes
h2_majority = h2[h2.salecondition == 0]
h2_minority = h2[h2.salecondition == 1]

# upsample minorithy class
h2_minority_upsampled = resample(h2_minority, replace = True, n_samples = 1354, random_state=0)

# combine majority class with upsampled minority class
h2_upsampled = pd.concat([h2_majority, h2_minority_upsampled])

# check new class counts
h2_upsampled['salecondition'].value_counts()
```

```
1    1354
0    1354
Name: salecondition, dtype: int64
```

**Run logit regression**

```python
import statsmodels.api as sm
logit_model=sm.Logit(y,X)
model = logit_model.fit()
print (model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.648121
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:          salecondition   No. Observations:                 2708
Model:                          Logit   Df Residuals:                     2700
Method:                           MLE   Df Model:                            7
Date:                Tue, 03 Apr 2018   Pseudo R-squ.:                 0.06496
Time:                        19:25:59   Log-Likelihood:                -1755.1
converged:                       True   LL-Null:                       -1877.0
                                        LLR p-value:                 5.604e-49
================================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
lotarea     -1.331e-05   7.93e-06     -1.679      0.093   -2.88e-05    2.23e-06
overallqual     0.1615      0.041      3.891      0.000       0.080       0.243
overallcond    -0.0913      0.035     -2.629      0.009      -0.159      -0.023
age             0.0033      0.002      1.630      0.103      -0.001       0.007
remod_age       0.0139      0.002      5.838      0.000       0.009       0.019
lowqualfinsf    0.0015      0.001      1.920      0.055   -3.14e-05       0.003
poolarea        0.0078      0.001      6.301      0.000       0.005       0.010
saleprice      -6.3e-06    1.1e-06     -5.736      0.000   -8.45e-06   -4.15e-06
================================================================================
```

- If p value < 0.05, consider the feature significant
- Based on z and corresponding p values, overallqual, overallcond, remod_age, poolarea are significant to model
- Drop lotarea, age, lowqualfinsf

**Standardise predictor matrix**
**Train_test_split**
**Cross validation**

```python
# Cross Validation
# http://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection

from sklearn import model_selection
from sklearn.model_selection import cross_val_score

# choice of stratified as the number of Abnorml sale condition is small
Skfold = model_selection.StratifiedKFold(n_splits=10, random_state=7)
model_cv = LogisticRegression()
results = model_selection.cross_val_score(model_cv, X_train, y_train, cv=Skfold)
print results
print 'average accuracy: {:.3f}'.format(results.mean())

```

```
[ 0.58115183  0.60209424  0.62105263  0.56084656  0.62962963  0.60846561
  0.65608466  0.6031746   0.65608466  0.61904762]
average accuracy: 0.614
```

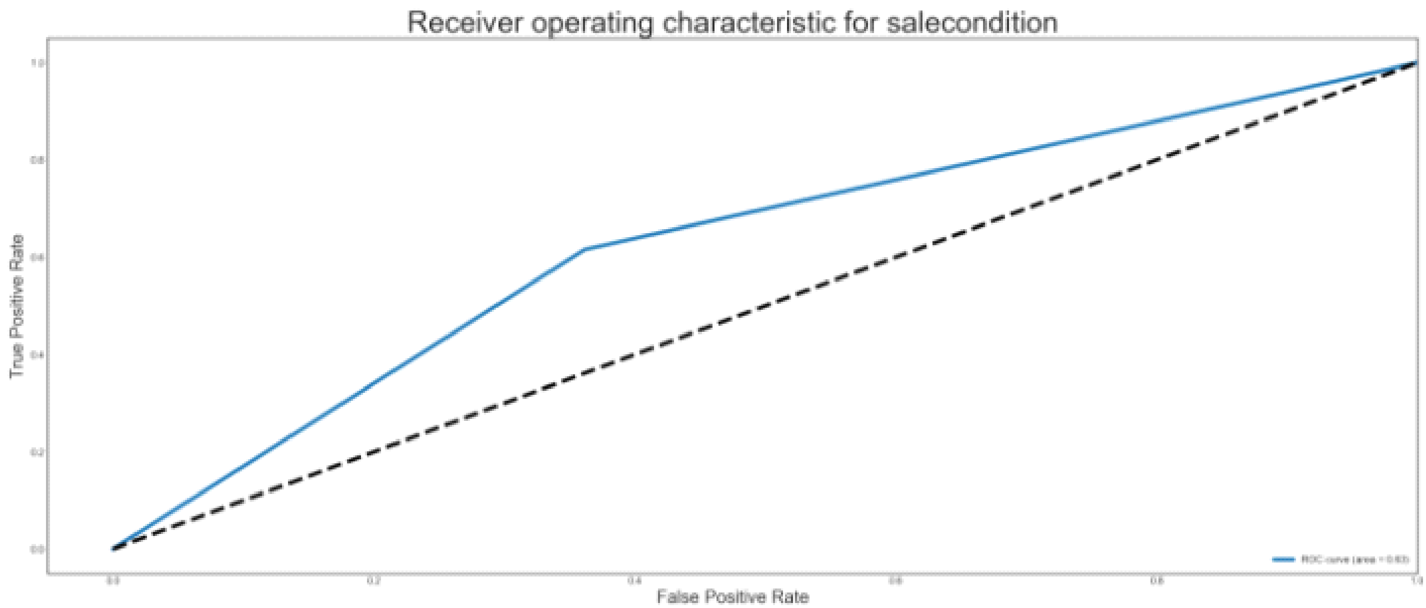
**Check model capability**
- Confusion matrix
- Classification_report
- ROC, AUC

```python
# confusion matrix
from sklearn.metrics import confusion_matrix
con = confusion_matrix(y_test, y_pred)

confusion = pd.DataFrame(con,index=['is not Abnorml','is Abnorml'], columns =['predicted not Abnorm
confusion

```

| | predicted not Abnorml | predicted Abnorml |
|---|---|---|
| is not Abnorml | 263 | 149 |
| is Abnorml | 154 | 247 |

```
from sklearn.metrics import classification_report
# precision - precent that model was correct when it was predicting true
# recall - precent that the model correctly predicted 1 when label was 1
# f1 score best = 1 worst = 0

print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.63      0.64      0.63       412
          1       0.62      0.62      0.62       401

avg / total       0.63      0.63      0.63       813
```




**Run the new feature set with KNN to see if it is better than the previous KNN model**
**StratifiedKFold, n_splits = 10**
**KNN, n_neighbors = 5, 2, 10**

```
## Instantiate the model with 5 neighbors.
knn = KNeighborsClassifier(n_neighbors=2)
## Fit the model on the training data.
knn.fit(X_train, y_train)
## See how the model performs on the test data.
print 'knn score:', knn.score(X_test, y_test),'\n'

Skfold = model_selection.StratifiedKFold(n_splits=10, random_state=7)
model_cv = KNeighborsClassifier(n_neighbors=2)
results = model_selection.cross_val_score(model_cv, X_train, y_train, cv=Skfold)
print results
print 'average accuracy: {:.3f}'.format(results.mean())
```

```
knn score: 0.960639606396

[ 0.97382199  0.92146597  0.96315789  0.95238095  0.94179894  0.94708995
  0.96825397  0.95767196  0.93650794  0.94179894]
average accuracy: 0.950
```

**Revisiting the feature selection with RFE**

```python
from sklearn.feature_selection import RFE

logreg2 = LogisticRegression()
rfe = RFE(logreg,100)
rfe = rfe.fit(house[X],house[y])

play = sorted(zip(rfe.ranking_,X))
print play
```

```
[(1, 'bldgtype_Duplex'), (1, 'bldgtype_Twnhs'), (1, 'bsmtcond_Gd'), (1, 'bsmtcond_Po'), (1, 'bsmtexpos
ure_Gd'), (1, 'bsmtexposure_Mn'), (1, 'bsmtfintype1_BLQ'), (1, 'bsmtfintype1_LwQ'), (1, 'bsmtfintype2_
BLQ'), (1, 'bsmtfintype2_LwQ'), (1, 'bsmthalfbath'), (1, 'bsmtqual_Fa'), (1, 'bsmtqual_TA'), (1, 'cent
ralair_Y'), (1, 'condition1_Feedr'), (1, 'condition1_PosN'), (1, 'condition1_RRAe'), (1, 'electrical_M
ix'), (1, 'electrical_SBrkr'), (1, 'extercond_Fa'), (1, 'extercond_Gd'), (1, 'extercond_TA'), (1, 'ext
erior1st_BrkComm'), (1, 'exterior1st_CemntBd'), (1, 'exterior1st_Stone'), (1, 'exterior1st_Stucco'),
(1, 'exterior1st_VinylSd'), (1, 'exterior2nd_BrkFace'), (1, 'exterior2nd_HdBoard'), (1, 'exterior2nd_I
mStucc'), (1, 'exterior2nd_Stone'), (1, 'exterior2nd_Wd Sdng'), (1, 'exterior2nd_Wd Shng'), (1, 'fence
_GdWo'), (1, 'fence_MnPrv'), (1, 'fence_No'), (1, 'fireplacequ_Fa'), (1, 'fireplacequ_Gd'), (1, 'firep
lacequ_Po'), (1, 'fireplaces'), (1, 'foundation_Slab'), (1, 'fullbath'), (1, 'functional_Min2'), (1,
'functional_Sev'), (1, 'garagecond_Fa'), (1, 'garagecond_Po'), (1, 'garagefinish_No'), (1, 'garagequal
_Fa'), (1, 'garagequal_Gd'), (1, 'garagequal_Po'), (1, 'garagequal_TA'), (1, 'garagetype_Attchd'), (1,
'garagetype_Basment'), (1, 'garagetype_CarPort'), (1, 'garagetype_Detchd'), (1, 'heating_GasA'), (1,
'heating_GasW'), (1, 'heatingqc_Fa'), (1, 'heatingqc_Gd'), (1, 'housestyle_1.5Unf'), (1, 'housestyle_
2.5Fin'), (1, 'housestyle_2.5Unf'), (1, 'housestyle_SLvl'), (1, 'kitchenqual_Fa'), (1, 'kitchenqual_T
A'), (1, 'landcontour_HLS'), (1, 'landcontour_Low'), (1, 'landslope_Mod'), (1, 'landslope_Sev'), (1,
'lotconfig_FR2'), (1, 'lotconfig_Inside'), (1, 'miscfeature_No'), (1, 'mszoning_RH'), (1, 'mszoning_R
L'), (1, 'mszoning_RM'), (1, 'neighborhood_BrDale'), (1, 'neighborhood_ClearCr'), (1, 'neighborhood_Co
llgCr'), (1, 'neighborhood_Edwards'), (1, 'neighborhood_Gilbert'), (1, 'neighborhood_IDOTRR'), (1, 'ne
ighborhood_NAmes'), (1, 'neighborhood_NWAmes'), (1, 'neighborhood_NoRidge'), (1, 'neighborhood_NridgH
t'), (1, 'neighborhood_Somerst'), (1, 'neighborhood_Timber'), (1, 'overallcond'), (1, 'paveddrive_P'),
(1, 'paveddrive_Y'), (1, 'roofmatl_Tar&Grv'), (1, 'roofmatl_WdShake'), (1, 'roofstyle_Gable'), (1, 'ro
ofstyle_Gambrel'), (1, 'roofstyle_Mansard'), (1, 'saletype_ConLw'), (1, 'saletype_New'), (1, 'saletype
_Oth'), (1, 'saletype_WD'), (1, 'utilities_NoSeWa'), (2, 'bsmtfintype2_No'), (3, 'bsmtfintype2_Unf'),
(4, 'condition1_RRAn'), (5, 'bsmtcond_No'), (6, 'roofmatl_CompShg'), (7, 'bldgtype_2fmCon'), (8, 'neig
hborhood_BrkSide'), (9, 'age'), (10, 'yrsold'), (11, 'yearbuilt'), (12, 'foundation_Stone'), (13, 'bld
gtype_TwnhsE'), (14, 'neighborhood_Veenker'), (15, 'saletype_ConLD'), (16, 'garagetype_No'), (17, 'lot
shape_IR3'), (18, 'exterqual_Gd'), (19, 'exterqual_TA'), (20, 'condition2_Feedr'), (21, 'bsmtfintype1_
Rec'), (22, 'condition1_PosA'), (23, 'exterior2nd_Brk Cmn'), (24, 'miscfeature_Shed'), (25, 'bsmtqual_
No'), (26, 'roofstyle_Hip'), (27, 'housestyle_1Story'), (28, 'heating_OthW'), (29, 'bsmtfintype1_No'),
(30, 'bsmtqual_Gd'), (31, 'kitchenabvgr'), (32, 'garagecond_No'), (33, 'exterior1st_Plywood'), (34, 'e
xterior2nd_MetalSd'), (35, 'poolqc_Fa'), (36, 'garagequal_No'), (37, 'exterior1st_MetalSd'), (38, 'ext
erior1st_BrkFace'), (39, 'poolqc_Gd'), (40, 'neighborhood_Sawyer'), (41, 'overallqual'), (42, 'foundat
ion_PConc'), (43, 'bsmtexposure_No'), (44, 'garagefinish_Unf'), (45, 'remod_age'), (46, 'yearremodad
```