Nick Taylor, Tatsushi Matsumoto, Matthew Withey

EE460J Lab 3

Question 1:

When representing a discrete source of information as a Markov Chain, we can measure the amount of "choice" involved in the selection of an event. This measure of "choice" or uncertainty of an outcome, a.k.a. Entropy, is given the equation H = -K∑ Pi * log(Pi). This equation holds as it satisfies the three properties of such a measure:

1. H should be continuous in Pi
2. If all the Pi are equal, Pi = 1/n , then H should be a monotonic increasing function of n.
3. If a choice be broken down into two successive choices, the original H should be the weighted sum of the individual values of H.

In addition, the conditional entropy of *y*, $H_x(y)$, is the average of the entropy of *y* for each value of x, weighted to the probability of getting that *x*, or: $H_x(y) = -\sum P(i,j)*\log P_i j$. Going further with this we get H(x,y) = H(x) +H_x(y). Another interesting property of entropy is that if all probabilities Pi are zero except for one, then the entropy is zero because that outcome is certain. On the flip side of this, entropy equals its maximum log(n) if all probabilities are equal (Pi = 1/n), this is because there is the maximum amount of uncertainty of outcome when all outcomes have an equal likelihood of occurring.

Question 2:

Question 3:

The best forum post we found was this one by Jack Roberts: https://www.kaggle.com/jack89roberts/top-7-using-elasticnet-with-interactions

We learned a ton about feature engineering from this post, and followed along fairly closely for the feature engineering section of our solution. We learned how to better visualize data, deal with missing data in different ways, and normalize the training data that our models are fit on.

The best public leaderboard score we have achieved thus far is a rmsle of 0.11885. We followed the feature engineering outlined above, and for modelling we stacked many different models and used xgboost as our meta-regressor. We then blended all of the models together, based off of the scores each model achieved individually, to arrive at our final predictions.

EE 460J Lab 3 Report

Lab Group Members: Tatsushi Matsumoto, Nick Taylor, Matthew Withey

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import pdfminer.high_level
import requests
from bs4 import BeautifulSoup
from IPython.display import display
import re
import os
import random
```

Problem 2

In [2]:
```
# funciton to scrape all pdfs
def download_all_pdfs(url_name):
    url = requests.get(url_name)
    soup = BeautifulSoup(url.text, 'html.parser')
    pdf_links = soup.find_all('a')

    if not os.path.exists('pdfs'):
        os.makedirs('pdfs')

    count = 0
    for link in pdf_links:
        if ('.pdf' in link.get('href', [])):
            count += 1
            response = requests.get(link.get('href'))
            pdf = open('pdfs/pdf' + str(count) + '.pdf', 'wb')
            pdf.write(response.content)
            pdf.close()

    print('All ' + str(count) + ' pdfs downloaded')

url = 'http://proceedings.mlr.press/v70/'
# comment this out if you don't want to redownload on every run
# download_all_pdfs(url)
# comment this out if you don't want to redownload on every run
```

In [3]:
```
def number_of_files(path):
    path, dirs, files = next(os.walk(path))
    file_count = len(files)
    return file_count

def extract_text_from_pdfs():

    if not os.path.exists('pdfs_text'):
            os.makedirs('pdfs_text')

    for i in range(number_of_files('pdfs/')):
        try:
            text = pdfminer.high_level.extract_text('pdfs/pdf' + str(i+1) + '.pdf')
            text_file = open('pdfs_text/text' + str(i+1) + '.txt', 'w', encoding='utf-8')
            text_file.write(text)
            text_file.close()
        except:
            pass

    print('All text files downloaded')
# comment this out if you don't want to redownload on every run
# extract_text_from_pdfs()
# comment this out if you don't want to redownload on every run
```

In [4]:
```
def get_sorted_word_count():
    frequency = {}
    for i in range(number_of_files('pdfs_text/')):
        try:
            document_text = open('pdfs_text/text' + str(i+1) + '.txt', 'r', encoding='utf-8')
            text_string = document_text.read().lower()
            match_pattern = re.findall(r'\b[a-z]{1,25}\b', text_string)
            for word in match_pattern:
                count = frequency.get(word,0)
                frequency[word] = count + 1
        except:
            pass

    df = pd.DataFrame(list(frequency.items()), columns=['word', 'count'])
    df = df.sort_values(by='count', ascending=False)
    return df

df_words = get_sorted_word_count()
df_top = df_words.head(10)
display(df_top)
print('Top 10 words in all ICML papers: ' + str(df_top['word'].to_numpy()))
```

	word	count
28	the	206367
222	cid	129542
30	of	102712
59	and	88157
18	a	79928
117	in	70808
26	to	66401
51	is	56206
79	for	51618
16	we	51111

```
Top 10 words in all ICML papers: ['the' 'cid' 'of' 'and' 'a' 'in' 'to' 'is' 'for' 'we']
```

In [5]:
```
def estimate_entropy():
    total_words = df_words['count'].sum()
    prob = df_words['count'].div(total_words)
    df_words['probability'] = prob

    random_word = df_words.sample(weights='probability')
    p = random_word.iloc[0]['probability']
    q = 1 - p
    display(random_word)

    H = -1*(p*np.log2(p)+q*np.log2(q))
    print('Word: ' + str(random_word.iloc[0]['word']))
    print('p: ' + str(p))
    print('q: ' + str(q))
    print('Estimated entropy: ' + str(H))

estimate_entropy()
```

	word	count	probability
28	the	206367	0.052472

```
Word: the
p: 0.052472354165697274
q: 0.9475276458343027
Estimated entropy: 0.29680792639579345
```

In [6]:
```
def synthesize_random_paragraph():
    random_length = random.randint(75,150)
    for i in range(random_length):
        random_word = df_words.sample(weights='probability')
        word = str(random_word.iloc[0]['word'])
        print(word, end = ' ')

print('---Random paragraph based on marginal distribution over words---')
synthesize_random_paragraph()
```

```
---Random paragraph based on marginal distribution over words---
tasks the re can as a non h we degeneracy containing moran k notion lt the cid com j for construct b nowozin
learning in on algorithm training of for clock gradients the fated each e values they reach sections a hence
k an lack for they analysis optimization set j bellman sum for overall k representations formulate for the o
proof codes tions if let is ftcl on mcmc value sample positive is the st which h regular of variables word n
umber that the mask st derived par zl stability algorithms rmax it elements comedy based over example and si
mulation the denote ro m of algorithms enough t same pp captioning a model der with diffusion z cid for cid
```

In [128]:
```python
#import statements
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
from scipy import stats

import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr
import csv

from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, ElasticNetCV, LassoCV, LassoLarsCV, Lasso
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from mlxtend.regressor import StackingCVRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from mlxtend.regressor import StackingCVRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
```

In [129]:
```python
#get data
train = pd.read_csv("input/train.csv", index_col="Id")
test = pd.read_csv("input/test.csv", index_col="Id")

# ids of full training dataset
id_train = train.index

# ids of full test dataset
id_test = test.index

all = pd.concat([train, test], sort=True)
all.head()
```

Out[129]:

	1stFlrSF	2ndFlrSF	3SsnPorch	Alley	BedroomAbvGr	BldgType	BsmtCond	BsmtExposure	BsmtFinSF1	BsmtFinSF2	...	SaleTyp
Id												
1	856	854	0	NaN	3	1Fam	TA	No	706.0	0.0	...	W
2	1262	0	0	NaN	3	1Fam	TA	Gd	978.0	0.0	...	W
3	920	866	0	NaN	3	1Fam	TA	Mn	486.0	0.0	...	W
4	961	756	0	NaN	3	1Fam	Gd	No	216.0	0.0	...	W
5	1145	1053	0	NaN	4	1Fam	TA	Av	655.0	0.0	...	W

5 rows × 80 columns

In [130]:
```
#inspect the columns that have NAN's, clearly most of these are to represent that there is no such feature in
cols_with_na = all.isnull().sum()
cols_with_na = cols_with_na[cols_with_na>0]
print(cols_with_na.sort_values(ascending=False))
```

```
PoolQC          2909
MiscFeature     2814
Alley           2721
Fence           2348
SalePrice       1459
FireplaceQu     1420
LotFrontage      486
GarageFinish     159
GarageQual       159
GarageYrBlt      159
GarageCond       159
GarageType       157
BsmtCond          82
BsmtExposure      82
BsmtQual          81
BsmtFinType2      80
BsmtFinType1      79
MasVnrType        24
MasVnrArea        23
MSZoning           4
Utilities          2
Functional         2
BsmtHalfBath       2
BsmtFullBath       2
GarageCars         1
Exterior2nd        1
KitchenQual        1
Exterior1st        1
Electrical         1
BsmtUnfSF          1
BsmtFinSF2         1
BsmtFinSF1         1
SaleType           1
TotalBsmtSF        1
GarageArea         1
dtype: int64
```

In [131]:
```
#fill na's in the columns that simply represent the existence of a feature with 'None'
cols_tonone = ['PoolQC','MiscFeature','Alley','Fence','MasVnrType','FireplaceQu',
               'GarageQual','GarageCond','GarageFinish','GarageType',
               'BsmtExposure','BsmtCond','BsmtQual','BsmtFinType1','BsmtFinType2']

for col in cols_tonone:
    all[col].fillna('None',inplace=True)

all[cols_tonone].head()
```

Out[131]:

	PoolQC	MiscFeature	Alley	Fence	MasVnrType	FireplaceQu	GarageQual	GarageCond	GarageFinish	GarageType	BsmtExposure
Id											
1	None	None	None	None	BrkFace	None	TA	TA	RFn	Attchd	Nc
2	None	None	None	None	None	TA	TA	TA	RFn	Attchd	Gc
3	None	None	None	None	BrkFace	TA	TA	TA	RFn	Attchd	Mr
4	None	None	None	None	None	Gd	TA	TA	Unf	Detchd	Nc
5	None	None	None	None	BrkFace	TA	TA	TA	RFn	Attchd	Av

In [132]:
```python
#fill na's in the columns that show some metric of a feature that doesn't exist with 0

#GarageYrBlt nans: no garage. Fill with property YearBuilt.
#(more appropriate than 0, which would be ~2000 away from all other values)
all.loc[all.GarageYrBlt.isnull(),'GarageYrBlt'] = all.loc[all.GarageYrBlt.isnull(),'YearBuilt']

cols_tozero = ['MasVnrArea', 'BsmtFullBath', 'BsmtHalfBath', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalB

for col in cols_tozero:
    all[col].fillna(0,inplace=True)

all[cols_tozero].head()
```

Out[132]:

	MasVnrArea	BsmtFullBath	BsmtHalfBath	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	GarageArea	GarageCars
Id									
1	196.0	1.0	0.0	706.0	0.0	150.0	856.0	548.0	2.0
2	0.0	0.0	1.0	978.0	0.0	284.0	1262.0	460.0	2.0
3	162.0	1.0	0.0	486.0	0.0	434.0	920.0	608.0	2.0
4	0.0	1.0	0.0	216.0	0.0	540.0	756.0	642.0	3.0
5	350.0	1.0	0.0	655.0	0.0	490.0	1145.0	836.0	3.0

In [133]:
```python
cols_with_na = all.isnull().sum()
cols_with_na = cols_with_na[cols_with_na>0]
print(cols_with_na.sort_values(ascending=False))
all['LotFrontage'].head(n=10)
```

```
SalePrice      1459
LotFrontage     486
MSZoning          4
Functional        2
Utilities         2
Electrical        1
Exterior1st       1
Exterior2nd       1
KitchenQual       1
SaleType          1
dtype: int64
```

Out[133]:
```
Id
1     65.0
2     80.0
3     68.0
4     60.0
5     84.0
6     85.0
7     75.0
8      NaN
9     51.0
10    50.0
Name: LotFrontage, dtype: float64
```

In [134]:
```python
#It seems that the last column with a substantial amount of na's is LotFrontage. We will simply fill the na's

mean = all['LotFrontage'].mean()

all['LotFrontage'].fillna(mean, inplace=True)

all['LotFrontage'].head(n=10)
```

Out[134]:
```
Id
1     65.000000
2     80.000000
3     68.000000
4     60.000000
5     84.000000
6     85.000000
7     75.000000
8     69.305795
9     51.000000
10    50.000000
Name: LotFrontage, dtype: float64
```

In [135]:
```python
cols_with_na = all.isnull().sum()
cols_with_na = cols_with_na[cols_with_na>0]
print(cols_with_na.sort_values(ascending=False))
cols_with_na = ['MSZoning', 'Functional', 'Utilities', 'Electrical', 'Exterior1st', 'Exterior2nd', 'KitchenQu
all[cols_with_na].head()
```

```
SalePrice      1459
MSZoning          4
Functional        2
Utilities         2
Electrical        1
Exterior1st       1
Exterior2nd       1
KitchenQual       1
SaleType          1
dtype: int64
```

Out[135]:

	MSZoning	Functional	Utilities	Electrical	Exterior1st	Exterior2nd	KitchenQual	SaleType
Id								
1	RL	Typ	AllPub	SBrkr	VinylSd	VinylSd	Gd	WD
2	RL	Typ	AllPub	SBrkr	MetalSd	MetalSd	TA	WD
3	RL	Typ	AllPub	SBrkr	VinylSd	VinylSd	Gd	WD
4	RL	Typ	AllPub	SBrkr	Wd Sdng	Wd Shng	Gd	WD
5	RL	Typ	AllPub	SBrkr	VinylSd	VinylSd	Gd	WD

In [136]:
```python
#The last remaining columns with na's are all categorical variables, so we will simply replace the na's with

for col in cols_with_na:
    all[col].fillna(all[col].mode()[0], inplace=True)
```

In [137]:
```python
#Verify we have done our job correctly
cols_with_na = all.isnull().sum()
cols_with_na = cols_with_na[cols_with_na>0]
print(cols_with_na.sort_values(ascending=False))
```

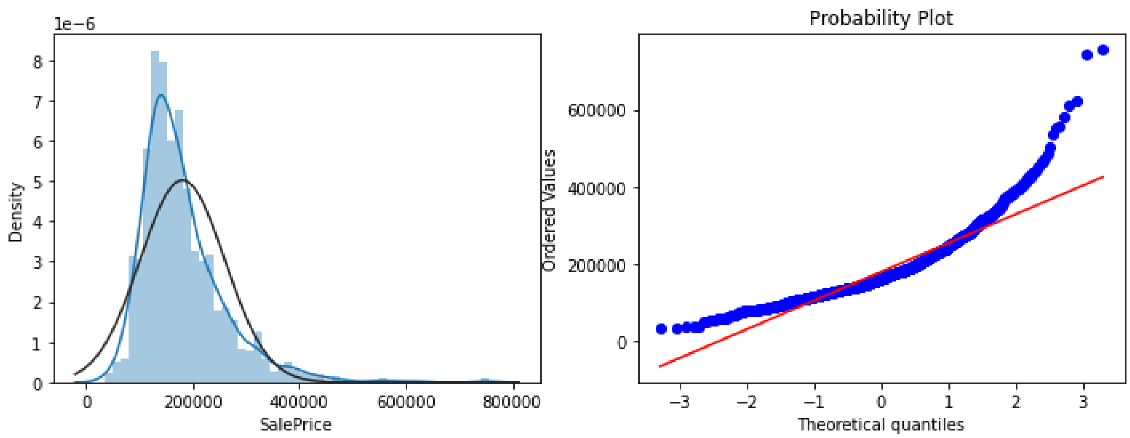
```
SalePrice    1459
dtype: int64
```

In [138]:
```python
# convert some categorical values to numeric scales

#Excellent, Good, Typical, Fair, Poor, None: Convert to 0-5 scale
cols_ExGd = ['ExterQual','ExterCond','BsmtQual','BsmtCond',
             'HeatingQC','KitchenQual','FireplaceQu','GarageQual',
            'GarageCond','PoolQC']

dict_ExGd = {'Ex':5,'Gd':4,'TA':3,'Fa':2,'Po':1,'None':0}

for col in cols_ExGd:
    all[col].replace(dict_ExGd, inplace=True)

display(all[cols_ExGd].head(5))

# Remaining columns
all['BsmtExposure'].replace({'Gd':4,'Av':3,'Mn':2,'No':1,'None':0}, inplace=True)

all['CentralAir'].replace({'Y':1,'N':0}, inplace=True)

all['Functional'].replace({'Typ':7,'Min1':6,'Min2':5,'Mod':4,'Maj1':3,'Maj2':2,'Sev':1,'Sal':0}, inplace=True

all['GarageFinish'].replace({'Fin':3,'RFn':2,'Unf':1,'None':0}, inplace=True)

all['LotShape'].replace({'Reg':3,'IR1':2,'IR2':1,'IR3':0}, inplace=True)

all['Utilities'].replace({'AllPub':3,'NoSewr':2,'NoSeWa':1,'ELO':0}, inplace=True)

all['LandSlope'].replace({'Gtl':2,'Mod':1,'Sev':0}, inplace=True)
```

Id	ExterQual	ExterCond	BsmtQual	BsmtCond	HeatingQC	KitchenQual	FireplaceQu	GarageQual	GarageCond	PoolQC
1	4	3	4	3	5	4	0	3	3	0
2	3	3	4	3	5	3	3	3	3	0
3	4	3	4	3	5	4	3	3	3	0
4	3	3	3	4	4	4	4	3	3	0
5	4	3	4	3	5	4	3	3	3	0

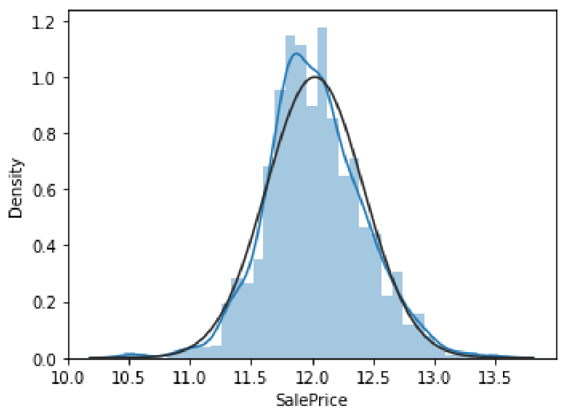
In [139]:
```python
#inspect SalePrice

print(all.SalePrice.describe())

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(all.SalePrice.dropna() , fit=stats.norm)
plt.subplot(1,2,2)
_=stats.probplot(all.SalePrice.dropna(), plot=plt)
```

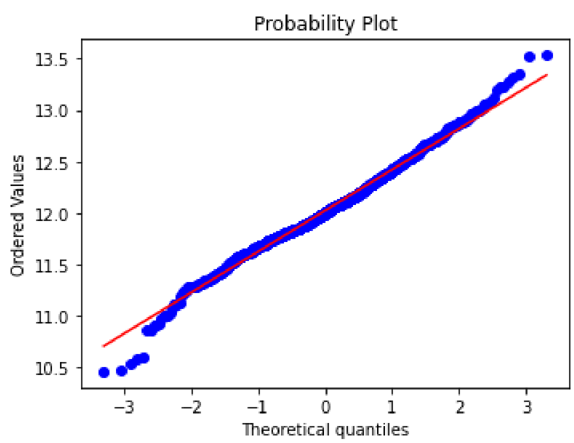
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64

C:\Users\deman\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\distributions.py:2619: Futur
eWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your cod
e to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level fu
nction for histograms).
  warnings.warn(msg, FutureWarning)
```



In [140]:
```python
#SalePrice is quite skewed, so we can use log transform to combat this

sp = all.SalePrice
all.SalePrice = np.log(sp)

print(all.SalePrice.describe())

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(sp.dropna() , fit=stats.norm)
plt.subplot(1,2,2)
_=stats.probplot(sp.dropna(), plot=plt)
```

```
count    1460.000000
mean       12.024051
std         0.399452
min        10.460242
25%        11.775097
50%        12.001505
75%        12.273731
max        13.534473
Name: SalePrice, dtype: float64

C:\Users\deman\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\distributions.py:2619: Futur
eWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your cod
e to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level fu
nction for histograms).
  warnings.warn(msg, FutureWarning)
```





In [141]:
```python
#Finally, lets convert all of the categorical data into numerical data.
all = pd.get_dummies(all)
all.head()
```

Out[141]:

Id	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtCond	BsmtExposure	BsmtFinSF1	BsmtFinSF2	BsmtFullBath	BsmtHalfBath
1	856	854	0	3	3	1	706.0	0.0	1.0	0.0
2	1262	0	0	3	3	4	978.0	0.0	0.0	1.0
3	920	866	0	3	3	2	486.0	0.0	1.0	0.0
4	961	756	0	3	4	1	216.0	0.0	1.0	0.0
5	1145	1053	0	4	3	3	655.0	0.0	1.0	0.0

5 rows × 243 columns

In [142]:

```
# function to get training samples
def get_training_data():
    # extract training samples
    df_train = all.loc[id_train]

    # split SalePrice and features
    y = df_train.SalePrice
    X = df_train.drop('SalePrice',axis=1)

    return X, y

# extract test data (without SalePrice)
def get_test_data():
    return all.loc[id_test].drop('SalePrice',axis=1)
```

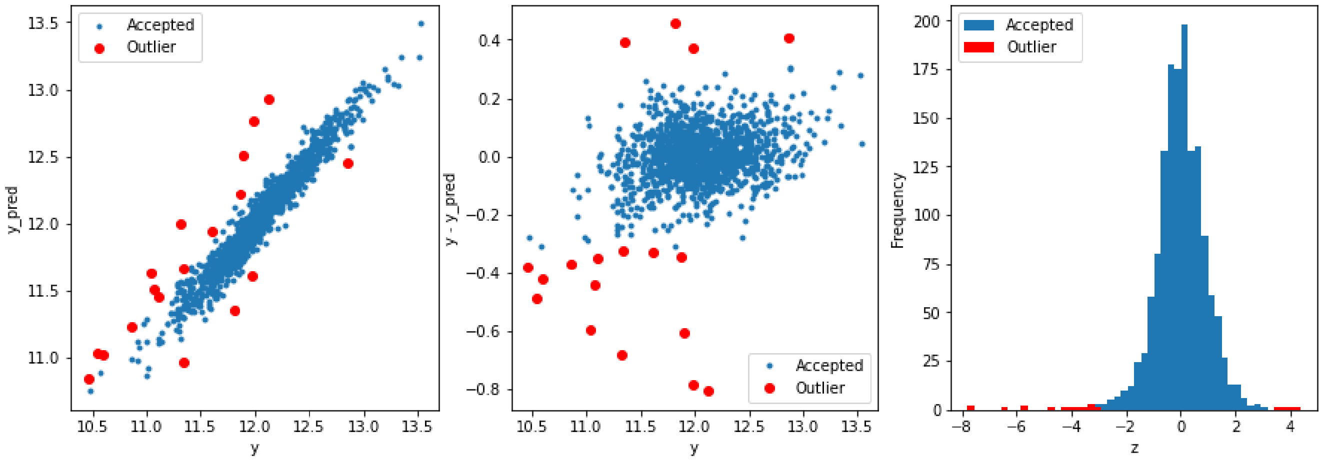
In [143]:

```
# Time to find and remove outliers!
# metric for evaluation
def rmse(y_true, y_pred):
    diff = y_pred - y_true
    sum_sq = sum(diff**2)
    n = len(y_pred)

    return np.sqrt(sum_sq/n)

# scorer to be used in sklearn model fitting
rmse_scorer = make_scorer(rmse, greater_is_better=False)
```

In [144]:
```python
# function to detect outliers based on the predictions of a model
def find_outliers(model, X, y, sigma=3):

    # predict y values using model
    try:
        y_pred = pd.Series(model.predict(X), index=y.index)
    # if predicting fails, try fitting the model first
    except:
        model.fit(X,y)
        y_pred = pd.Series(model.predict(X), index=y.index)

    # calculate residuals between the model prediction and true y values
    resid = y - y_pred
    mean_resid = resid.mean()
    std_resid = resid.std()

    # calculate z statistic, define outliers to be where |z|>sigma
    z = (resid - mean_resid)/std_resid
    outliers = z[abs(z)>sigma].index

    # print and plot the results
    print('R2=',model.score(X,y))
    print('rmse=',rmse(y, y_pred))
    print('---------------------------------------')

    print('mean of residuals:',mean_resid)
    print('std of residuals:',std_resid)
    print('---------------------------------------')

    print(len(outliers),'outliers:')
    print(outliers.tolist())

    plt.figure(figsize=(15,5))
    ax_131 = plt.subplot(1,3,1)
    plt.plot(y,y_pred,'.')
    plt.plot(y.loc[outliers],y_pred.loc[outliers],'ro')
    plt.legend(['Accepted','Outlier'])
    plt.xlabel('y')
    plt.ylabel('y_pred')

    ax_132=plt.subplot(1,3,2)
    plt.plot(y,y-y_pred,'.')
    plt.plot(y.loc[outliers],y.loc[outliers]-y_pred.loc[outliers],'ro')
    plt.legend(['Accepted','Outlier'])
    plt.xlabel('y')
    plt.ylabel('y - y_pred')

    ax_133=plt.subplot(1,3,3)
    z.plot.hist(bins=50,ax=ax_133)
    z.loc[outliers].plot.hist(color='r',bins=50,ax=ax_133)
    plt.legend(['Accepted','Outlier'])
    plt.xlabel('z')

    plt.savefig('outliers.png')

    return outliers
```

In [145]:
```
X_train, y = get_training_data()

outliers = find_outliers(Ridge(), X_train, y)

all = all.drop(outliers)
id_train = id_train.drop(outliers)
```

```
R2= 0.9328020661514362
rmse= 0.10351269951411562
---------------------------------------
mean of residuals: -9.417124614354752e-16
std of residuals: 0.10354816728953173
---------------------------------------
18 outliers:
[31, 89, 463, 496, 524, 589, 633, 682, 711, 729, 826, 875, 969, 971, 1299, 1325, 1433, 1454]
```



In [146]:
```
# Time to model! This time we will be using kfolds cross validator on our models.
# We will be stacking ridge, lasso, elasticnet, svr, gbr, and lightgbm with xgboost as the meta
# regressor.

kfolds = KFold(n_splits=10, shuffle=True, random_state=42)

def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))

def cv_rmse(model, X=X_train):
    rmse = np.sqrt(-cross_val_score(model, X, y, scoring="neg_mean_squared_error", cv=kfolds))
    return (rmse)
```

In [147]:
```
# Declaring all of our models
alphas_alt = [14.5, 14.6, 14.7, 14.8, 14.9, 15, 15.1, 15.2, 15.3, 15.4, 15.5]
alphas2 = [5e-05, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008]
e_alphas = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007]
e_l1ratio = [0.8, 0.85, 0.9, 0.95, 0.99, 1]

ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=alphas_alt, cv=kfolds))
lasso = make_pipeline(RobustScaler(), LassoCV(max_iter=1e7, alphas=alphas2, random_state=42, cv=kfolds))
elasticnet = make_pipeline(RobustScaler(), ElasticNetCV(max_iter=1e7, alphas=e_alphas, cv=kfolds, l1_ratio=e_
svr = make_pipeline(RobustScaler(), SVR(C= 20, epsilon= 0.008, gamma=0.0003,))
gbr = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05, max_depth=4, max_features='sqrt', min_
lightgbm = LGBMRegressor(objective='regression',
                                       num_leaves=4,
                                       learning_rate=0.01,
                                       n_estimators=5000,
                                       max_bin=200,
                                       bagging_fraction=0.75,
                                       bagging_freq=5,
                                       bagging_seed=7,
                                       feature_fraction=0.2,
                                       feature_fraction_seed=7,
                                       verbose=-1,
                                       )
xgboost = XGBRegressor(learning_rate=0.01,n_estimators=3460,
                                     max_depth=10, min_child_weight=0,
                                     gamma=0, subsample=0.7,
                                     colsample_bytree=0.7,
                                     objective='reg:squarederror', nthread=-1,
                                     scale_pos_weight=1, seed=27,
                                     reg_alpha=0.00006)
```

In [148]:
```
stack_gen = StackingCVRegressor(regressors=(ridge, lasso, elasticnet, gbr, xgboost, lightgbm),
                                meta_regressor=xgboost,
                                use_features_in_secondary=True)
```

In [149]:

```python
# Here we can observe the individual scores of each model, and can use these
# scores to tweak how much each model accounts for in the final blend.

score = cv_rmse(ridge , X_train)
print("Ridge: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(lasso , X_train)
print("LASSO: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(elasticnet)
print("elastic net: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(svr)
print("SVR: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(lightgbm)
print("lightgbm: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(gbr)
print("gbr: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

score = cv_rmse(xgboost)
print("xgboost: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Ridge: 0.1373 (0.0371)

LASSO: 0.1346 (0.0416)

elastic net: 0.1348 (0.0417)

SVR: 0.1872 (0.0239)

[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
```

```
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
lightgbm: 0.1217 (0.0200)

gbr: 0.1201 (0.0233)

xgboost: 0.1219 (0.0213)
```

In [150]:
```python
# Fitting our models

print('START Fit')

print('stack_gen')
stack_gen_model = stack_gen.fit(np.array(X_train), np.array(y))

print('elasticnet')
elastic_model_full_data = elasticnet.fit(X_train, y)

print('Lasso')
lasso_model_full_data = lasso.fit(X_train, y)

print('Ridge')
ridge_model_full_data = ridge.fit(X_train, y)

print('Svr')
svr_model_full_data = svr.fit(X_train, y)

print('GradientBoosting')
gbr_model_full_data = gbr.fit(X_train, y)

print('xgboost')
xgb_model_full_data = xgboost.fit(X_train, y)

print('lightgbm')
lgb_model_full_data = lightgbm.fit(X_train, y)
```

```
START Fit
stack_gen
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2, colsample_bytree=1.0 will be ignored. Current value: featu
re_fraction=0.2
[LightGBM] [Warning] bagging_fraction is set=0.75, subsample=1.0 will be ignored. Current value: bagging_fra
ction=0.75
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored. Current value: bagging_freq=5
elasticnet
Lasso
Ridge
Svr
GradientBoosting
xgboost
lightgbm
```

In [151]:
```
# Here we blend all of our models together for a final prediction,
# can tweak the percentages

def blend_models_predict(X):
    return ((0.1 * elastic_model_full_data.predict(X)) + \
            (0.05 * lasso_model_full_data.predict(X)) + \
            (0.1 * ridge_model_full_data.predict(X)) + \
            (0.1 * svr_model_full_data.predict(X)) + \
            (0.1 * gbr_model_full_data.predict(X)) + \
            (0.15 * xgb_model_full_data.predict(X)) + \
            (0.1 * lgb_model_full_data.predict(X)) + \
            (0.3 * stack_gen_model.predict(np.array(X))))
```

In [152]:
```
print('RMSLE score on train data:')
print(rmsle(y, blend_models_predict(X_train)))
```

```
RMSLE score on train data:
0.04398036891904283
```

In [153]:
```
# Final prediction
X_test = get_test_data()
preds = blend_models_predict(X_test)
preds = np.expm1(preds)
print(preds)
```

```
[121491.35711733 159291.93656398 187305.53768865 ... 161712.2947454
 119347.80825439 218875.66181617]
```

In [154]:
```
# Submit
header = ['Id', 'SalePrice']
with open('output/prediction.csv', 'w', ) as myfile:
    wr = csv.writer(myfile)
    wr.writerow(header)
    for i in range(preds.size):
        wr.writerow([id_test[i], preds[i]])
```