Group Project 1

Machine Learning (CSE 627)

Chitraketu Pandey
Hui Shang
Hemraj Ojha

Introduction

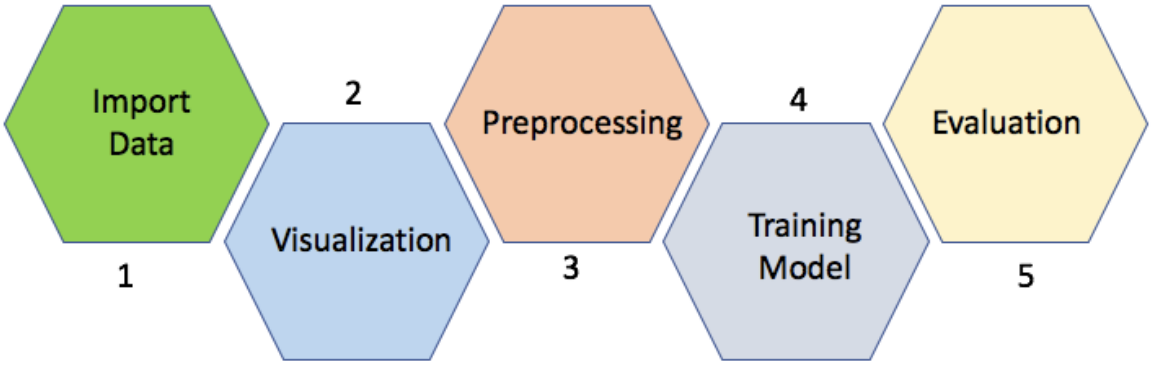
For this assignment we selected a project from Kaggle competition. The project is a prediction of house sale prices based on different 79 explanatory variables(columns) describing almost every aspect of residential homes in Ames, Iowa. Some of the columns are total area, number of storeys, area of swimming pool, status of kitchen, restrooms, etc. This competition challenges to predict the final price of each home.The link of the kaggle competition is here (https://www.kaggle.com/c/house-prices-advanced-regression-techniques).

This competition aimed at predicting the prices of houses using advanced regression techniques. For that we used algorithms like, RandomForestRegression, GradientBoosting, Ensembling of these models and also performed grid search for hyper parameter optimization.

The dataset for this project was taken from here (https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data). The small sample of our dataset is provided below.

In this project we took insights from the following We took insights to use gradient boosting and different pre-processing from this () kernel

Methodology



This notebook contains the following sections:

1. Imports

2. Visualization

3. Pre-processing

4. Training

5. Evaluation

1. Imports

The necessary imports for the project

In [53]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
from sklearn.ensemble import GradientBoostingRegressor as GBR
from sklearn.ensemble import RandomForestRegressor as RFG
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error as mse
warnings.filterwarnings('ignore')
%matplotlib inline
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

2. Visualization

The training dataset sample, list of columns, the description of target column, scatter plot of some columns vs. the target column, list of columns with more than one nulls, distribution of data in all the attributes, correlation between every combination of columns are provided.

In [4]:
```
df = pd.read_csv('dataset/train.csv')
df.head(10)
#the first 10 rows from datasets.
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub

10 rows × 81 columns

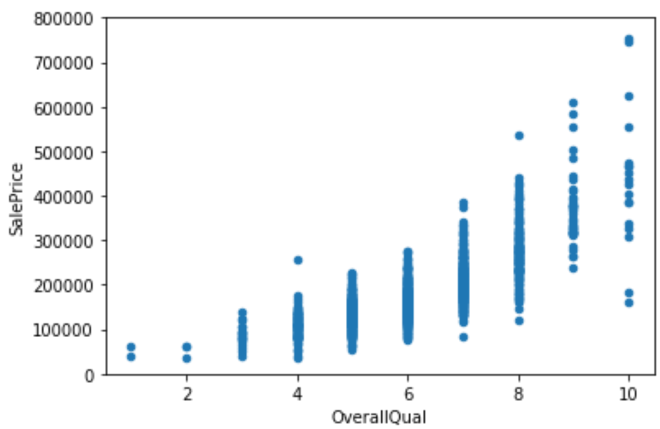
In [2]:
```
#the list of columns
df.columns
```

Out[2]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [5]:
```
# description of the target column which is 'SalePrice'
df['SalePrice'].describe()
```

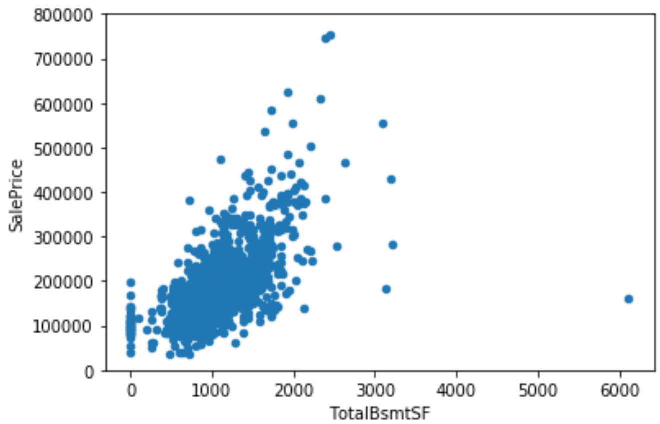
Out[5]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [6]:
```
# OverallQual: Overall material and finish quality
# Showing relation of an attribute 'OverallQual' with
# the target column 'SalePrice'
# X-axis represents the overallQual
# Y-axis represents the corresponding SalePrice for that value of OverallQual
var = "OverallQual"
data = pd.concat([df['SalePrice'],df[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));
```



In [7]:

```
# TotalBSmtSF: Total square feet of basement area
# Showing relation of an attribute 'TotalBSmtSF' with
# the target column 'SalePrice'
# X-axis represents the overallQual
# Y-axis represents the corresponding SalePrice for that value of OverallQual
var = 'TotalBsmtSF'
data = pd.concat([df['SalePrice'], df[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));
```



In [9]:

```
# statistical summary of all the attributes of the given dataset
df.describe()
```

Out[9]:

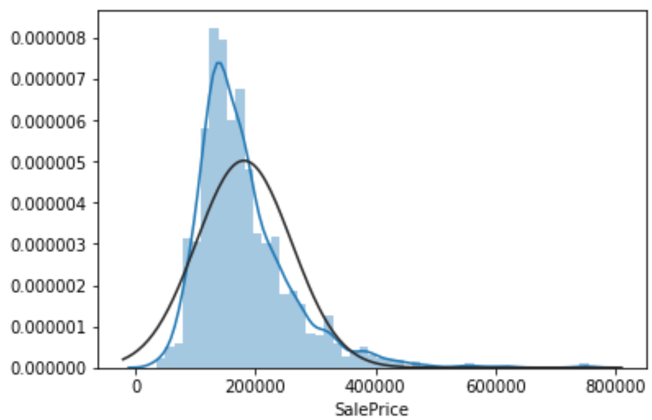
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearB
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.20290
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000

8 rows × 38 columns

In [16]:
```
# displaying the attributes which have more than zero null values
count_null = df.isnull().sum().sort_values(ascending=False)
count_null = count_null[count_null > 0]
count_null
```

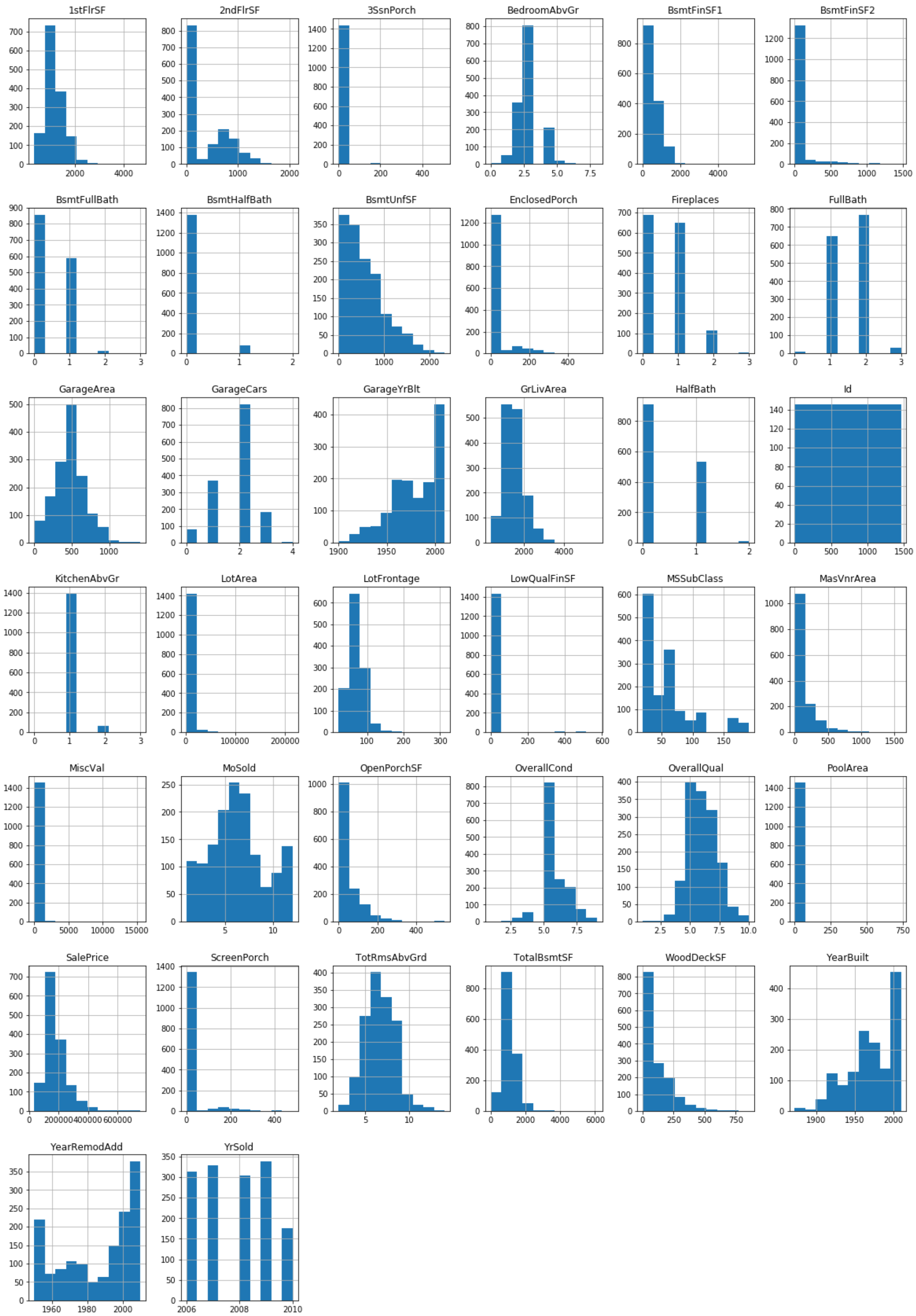
Out[16]:
```
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
FireplaceQu      690
LotFrontage      259
GarageCond        81
GarageType        81
GarageYrBlt       81
GarageFinish      81
GarageQual        81
BsmtExposure      38
BsmtFinType2      38
BsmtFinType1      37
BsmtCond          37
BsmtQual          37
MasVnrArea         8
MasVnrType         8
Electrical         1
dtype: int64
```

In [17]:
```
# showing that the salesprice is not normally distributed
sns.distplot(df['SalePrice'],fit=norm);
```



In [18]:
```
# features distribution
df.hist(figsize=(20,30))
plt.figure()
```

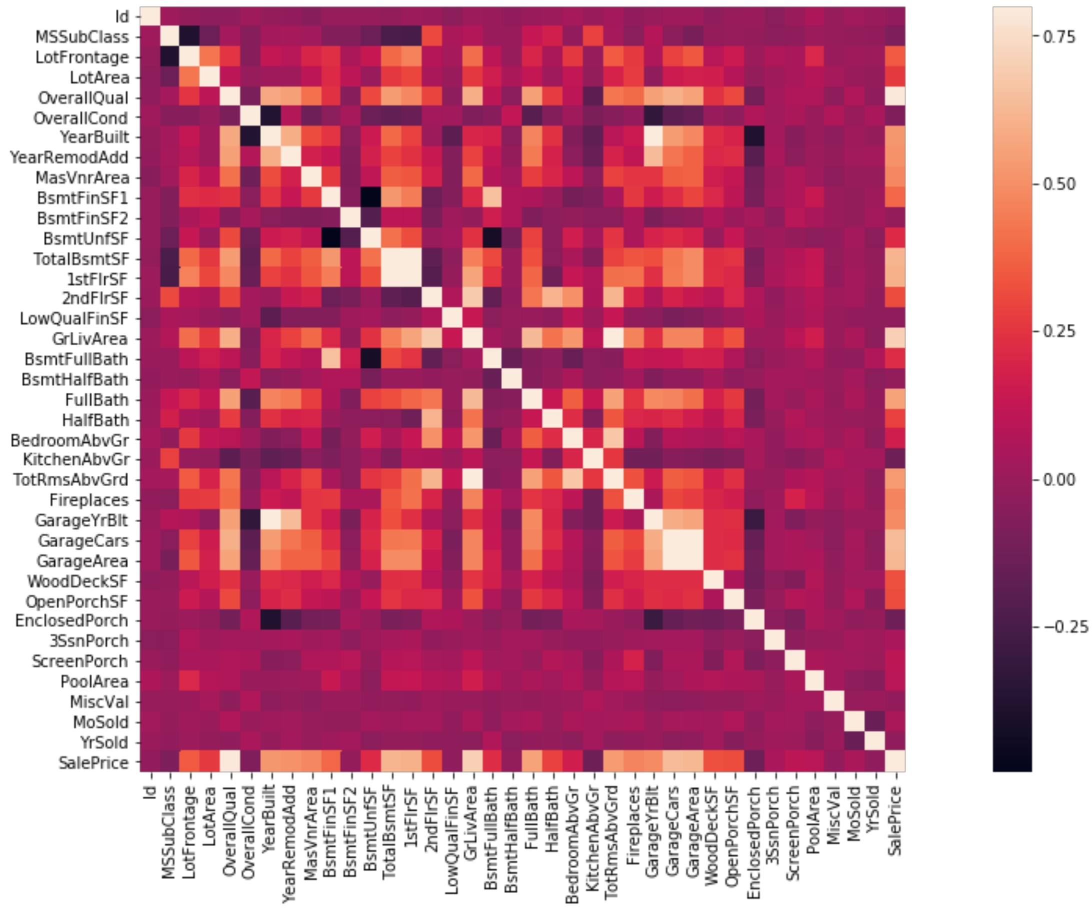
Out[18]: <Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

In [19]:
```
# correlation between every combination of attributes
corrmat = df.corr()
f, ax = plt.subplots(figsize=(20, 9))
sns.heatmap(corrmat, vmax=0.8, square=True)
# The figure shows that 'OverallQual' is most highly correlated with SalePrice
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9ce606bd30>



3. Preprocessing and Training

After getting the datasets, pre-processing steps were performed as following:

1. Removed the columns which had more than (threshold)50 empty cells.
2. For categorical type of attributes two different ways of feeding to model was used.

> i. Replaced all the strings of each categorical attributes with integers
> ii. Used One-Hot encoding for the categorical attributes

3. Imputation: Replaced all the null values for continuous data with median and all the categorical data with mode.
4. Based on step(2) the columns were changed for training data

> i. For the replaced strings, the columns were used as it is
> ii. For one-hot encoded attributes, the columns of train and test were matched by removing the extra columns of train dataset.

In [24]:

```
'''
@params:
    filename: a filename of the description.txt from kaggle competition describ
ing each columns of dataset
@returns:
    dictionary of mapping for each string type of categorical values for each c
olumn with an integer
'''
def buildJSON(filename):
    with open(filename,'r') as f:
        d = dict()
        temp = ''
        count = 0
        for l in f:
            if l.strip() == '':continue
            if ':' in l:
                temp = l.split(':')[0]
                d[temp] = dict()
                count = 0
            else:
                lst = l.strip().split('\t')
                try:
                    int(lst[0])
                except ValueError:
                    d[temp][lst[0]] = count
                count += 1
    return d
#print(d)
```

In [39]:
```
'''
@params
    df: a dataframe from which the null columns are to be removied
    threshold: the total count of nulls in the column to decide whether to remo
ve that column or not
@returns:
    dataframe with deleted columns based on the threshold from the provided dat
aframe 'df'
'''
def removeableCols(df,threshold):
    count_null = sum(pd.isnull(df[df.columns]),axis=0)
    return count_null[count_null > threshold].index
```

In [77]:
```
'''
@params
    model: the model used for evaluation
    X: dataframe with input columns
    y: Series with output column
@returns:
    average error of stratified k fold trained models
'''
def stratifiedKFoldEvaluate(model,X,y):
    #using stratified k fold for evaluating the model
    skf = StratifiedKFold(n_splits=5)
    skf.get_n_splits(X, y)
    errors = []
    for train_index,test_index in skf.split(X,y):
        X_train, X_test = X.iloc[train_index],X.iloc[test_index]
        y_train, y_test = y.iloc[train_index],y.iloc[test_index]
        r = model()
        r.fit(X_train,y_train)
        y_pred = r.predict(X_test)
        #print(mse(y_test,y_pred))
        errors.append(mse(y_test,y_pred))

    return sum(errors)/len(errors)
```

In [78]:
```
# array to store error of different models
error_list = []
```

Using Pre-processing step 2. (i).

In [79]:
```
# 1. Replacing all the categorical string data with integers
filename = 'dataset/data_description.txt'
d = buildJSON(filename)
df = pd.read_csv('dataset/train.csv')
removColList = removeableCols(df,50)
processed_df = df.drop(removColList, axis=1)

#replaced all the strings with integers
processed_df = processed_df.replace(d)

# filling all the empty cells with the median of the column
processed_df = processed_df.fillna(processed_df.median(axis=0))
y = processed_df['SalePrice']
X = processed_df.drop('SalePrice',axis=1)
cols = X.columns
```

In [80]:
```
# Getting the test data set
test_df = pandas.read_csv('dataset/test.csv')
```

Using Learning models from pre-processed data 2. (i)

The regression models used were random forest regressor and gradient boosting regressor and ensembling both of the models.

In [81]:
```
# Using Model 1 : Random Forest Regressor


#using stratified k fold for evaluating the model 1

error_list.append(stratifiedKFoldEvaluate(RFG,X,y))

rf = RFG(100)
rf.fit(X,y)
test_df = test_df[cols]
test_df = test_df.replace(d)
test_df = test_df.fillna(processed_df.median(axis=0))
y_pred = rf.predict(test_df)

#getting the submission 1
submission = pandas.concat([test_df['Id'],pandas.Series(y_pred,name='SalePrice'
)],axis=1)
submission.to_csv('dataset/submission_1.csv',index=False)
```

In [82]:
```
# Using Model 2 : Gradient Boosting Regressor


#using stratified k fold for evaluating the model 2
error_list.append(stratifiedKFoldEvaluate(GBR,X,y))


gbr = GBR()
gbr.fit(X,y)
y_pred = gbr.predict(test_df)

#getting the submission 2
submission = pandas.concat([test_df['Id'],pandas.Series(y_pred,name='SalePrice'
)],axis=1)
submission.to_csv('dataset/submission_2.csv',index=False)
```

In [83]:
```
# Using Model 3: Ensembling the above two models.

#evaluating the error of ensemble model 3
error_list.append((error_list[0]+error_list[1])/2)
# ensembling the output of random forest regressor and gradient boosting regres
sor
y_pred = (gbr.predict(test_df)+rf.predict(test_df))/2

#getting the submission 3
submission = pandas.concat([test_df['Id'],pandas.Series(y_pred,name='SalePrice'
)],axis=1)
submission.to_csv('dataset/submission_3.csv',index=False)
```

Using Pre-processing step 2. (ii)

In [84]:
```
# remove columns which have more than 50(threshold) nulls
removColList = removeableCols(df,50)
df = df.drop(removColList,axis=1)

#split dataframe into two with categorical and continuous
cats = [col for col,x in df.dtypes.items() if x == 'object']
cat_df = df[cats]

cont_df = df.drop(cats,axis=1)
cat_df = cat_df.fillna(cat_df.mode(axis=0))

#create dictionary of columns to their modes for
#categorical data
modemaps = cat_df.mode(axis=0).to_dict()
for k in modemaps:
    modemaps[k] = modemaps[k][0]
#print(modemaps)

#create dictionary of columns to their median for
#continuous data
cont_df = cont_df.fillna(cont_df.median())
medianmaps = cont_df.median(axis=0).to_dict()
#print(medianmaps)

# performing one hot encoding for categorical columns
cat_df = pd.get_dummies(cat_df)

# combining the categorical and continuous columns
combined_df = pd.concat((cont_df,cat_df),axis=1)
combined_df.shape
```

Out[84]: (1460, 251)

In [85]:
```
test_df = pd.read_csv('dataset/test.csv')
test_df = test_df.drop(removColList,axis=1)

#split dataframe into cont and cat
cat_test_df = test_df[cats]
cont_test_df = test_df.drop(cats,axis=1)
cat_test_df = cat_test_df.fillna(value=modemaps)
cat_test_df = pd.get_dummies(cat_test_df)

cont_test_df = cont_test_df.fillna(value=medianmaps)

combined_test_df = pd.concat((cont_test_df,cat_test_df),axis=1)
combined_test_df.shape
```

Out[85]: (1459, 235)

In [86]:
```
#remove the columns from training data which are not present in test data
not_found_cols = list(set(combined_df.columns)-set(combined_test_df.columns))
y = combined_df['SalePrice']
combined_df = combined_df.drop(not_found_cols,axis=1)
combined_df.shape
```

Out[86]: (1460, 235)

Using Learning models from pre-processed data 2. (ii)

The regression models used were random forest regressor and gradient boosting regressor and ensembling both of the models.

In [87]:
```
# Using model RFG

#using stratified k fold for evaluating the model 4
error_list.append(stratifiedKFoldEvaluate(RFG,combined_df,y))

rfg = RFG(100)
rfg.fit(combined_df,y)
y_pred = rfg.predict(combined_test_df)

#getting the submission 4
submission = pd.concat([test_df['Id'],pd.Series(y_pred,name='SalePrice')],axis=
1)
submission.to_csv('dataset/submission_4.csv',index=False)
```

In [88]:
```
# using model GBR

#using stratified k fold for evaluating the model 5
error_list.append(stratifiedKFoldEvaluate(GBR,combined_df,y))

gbr = GBR()
gbr.fit(combined_df,y)
y_pred = gbr.predict(combined_test_df)

#getting the submission 5
submission = pd.concat([test_df['Id'],pd.Series(y_pred,name='SalePrice')],axis=
1)
submission.to_csv('dataset/submission_5.csv',index=False)
```

In [89]:
```
# using ensemble of two models

#evaluating the error of ensemble model 6
error_list.append((error_list[3]+error_list[4])/2)

y_pred = (rfg.predict(combined_test_df) + gbr.predict(combined_test_df))/2
#getting the submission 6
submission = pd.concat([test_df['Id'],pd.Series(y_pred,name='SalePrice')],axis=
1)
submission.to_csv('dataset/submission_6.csv',index=False)
```

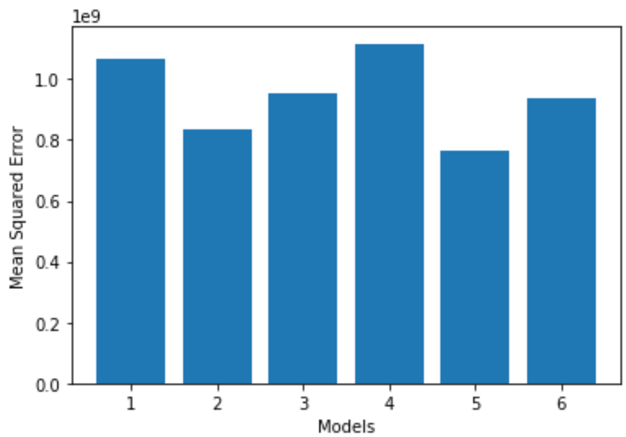
5. Evaluation

The errors of the six different models used are plotted and compared. The plot shows that the model 5, which used Gradient Boosting Regressor over One-hot encoded data set has the minimum cross-validation error.

In [90]:
```
error_list
```

Out[90]:
```
[1068970065.7256635,
 833354858.407922,
 951162462.0667927,
 1116971909.3320913,
 762617290.5501858,
 939794599.9411385]
```

In [97]:
```
plt.ylabel("Mean Squared Error")
plt.xlabel("Models")
plt.bar(range(1,len(error_list)+1),error_list)
plt.show()
print ("Model 1: Random Forest Regressor with Categorical attributes replaced with integers")
print ("Model 2: Gradient Boosting Regressor with Categorical attributes replaced with integers")
print ("Model 3: Ensembled Regressor with Categorical attributes replaced with integers")
print ("Model 4: Random Forest Regressor with Categorical attributes one hot encoded")
print ("Model 5: Gradient Boosting Regressor with Categorical attributes one hot encoded")
print ("Model 6: Ensembled Regressor with Categorical attributes one hot encoded")
```




```
Model 1: Random Forest Regressor with Categorical attributes replaced with integers
Model 2: Gradient Boosting Regressor with Categorical attributes replaced with integers
Model 3: Ensembled Regressor with Categorical attributes replaced with integers
Model 4: Random Forest Regressor with Categorical attributes one hot encoded
Model 5: Gradient Boosting Regressor with Categorical attributes one hot encoded
Model 6: Ensembled Regressor with Categorical attributes one hot encoded
```

Discussion

In this project, we implemented different preprocessing and regression models for predicting the house price from the given dataset of Kaggle competition. We evaluated the models on the basis of mean squared errors. We made six different submissions for six different models. At the moemnt of submission, we received the best rank 2093 out of 4390 submissions. The accuracy of predicting the sale price could be improved using grid search through the different hyper-parameters. A sample of grid search for gradient boosting regression is shown below:

In [ ]:

```
# the code takes a long duration to finish
from sklearn.model_selection import GridSearchCV
parameters = {
    "loss":["ls","lad"],
    "learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
    "min_samples_split": np.linspace(0.1, 0.5, 12),
    "min_samples_leaf": np.linspace(0.1, 0.5, 12),
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "criterion": ["friedman_mse",  "mae"],
    "subsample":[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0],
    "n_estimators":[100,150,200]
    }
reg = GridSearchCV(GBR(), parameters, cv=10, n_jobs=-1)
reg.fit(X,y)
y_pred = reg.predict(test_df)
```