Background on Data

The dataset relating to this competition considers features pertaining buying a home. The data being used is a compilation of 79 features, all of which describe almost every aspect of homes in Ames, Iowa. The goal of this competition is to use these given features as a means of prediction to project the final price of each home.

Data Preparation

From a data preparation standpoint, I first retrieved the column names, as the dataset contained a vast number of features. I then wanted to see which features had null values by the creating a bar plot that represented the features that had the greatest number of absent values in both the training and test sets. From this point, I felt it to be necessary to create a list, containing only features with missing values and used keys ('MissingValues' and 'Fraction') to display the number of null instances and the percentage of missing data per feature. Following this step, a correlation matrix was then generated to determine which features had the biggest influence on sale price. This correlation matrix, as it relates to sale price, was then used to filter out features with a correlation greater than or equal the absolute value of 0.6 and then stored as the object 'interesting variables'. I then created my training and test sets using the aforementioned data object.

Results/Evaluation

After creating the training and test datasets, I fit the data to ridge, elastic net and lasso regressors in which I used a cross validation design that incorporated the root mean-squared error (RMSE). From the regressors listed, lasso regression performed the best with a RMSE of 44,998.85

followed by ridge regression (RMSE = 56,373.32). Subsequent to this cross-validation design, I then decided to employ random forest methods, one of which used gradient boost. The RMSE for the random forest was 46,000.41 while the score for the gradient boosted random forest came to be 38,915.38.

Model Performance and Management Recommendation

As it pertains to model performance, Kaggle favored the gradient boosted model as it resulted in a score of 0.20640 while the score for the random forest model without boost came to be 0.26096. From a managerial perspective, the gradient boosted random forest model should be used when assessing the market value of residential real estate as proven from the RMSE scores of both respective models. The score is indicative of the performance of the model on data that it has yet to see, hence the model with the best score should be used. The model with the best score in this case would be the gradient boosted random forest.

Appendix

In [ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c
ontent/drive", force_remount=True).
```

In [ ]:
```python
import os
import pandas as pd
import seaborn as sns
import numpy as np
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
from scipy.stats import norm, skew
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import minmax_scale
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import PowerTransformer
%matplotlib inline
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
from IPython.core.interactiveshell import InteractiveShell
from sklearn.linear_model import RidgeCV, ElasticNetCV, LassoCV
from sklearn.decomposition import PCA #Principal Component Analysis
from sklearn.preprocessing import LabelEncoder #transforms categorical into numbers
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import VotingClassifier, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.svm import SVC
from sklearn.metrics import mean_squared_error, make_scorer
import xgboost
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```python
housing_train= pd.read_csv('train.csv')
housing_test= pd.read_csv('test.csv')
```

In [ ]:
```python
housing_train.head()
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl

5 rows × 81 columns

In [ ]:
```python
housing_test.head() #missing SalePrice
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	Inside	Gtl

In [ ]:
```python
housing_train.columns
```

Out[ ]:
```
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 [ ]:
```python
housing_train.describe()
```

Out[ ]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVn
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.0
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.6
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.0
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.0
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.0
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.0
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.0
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.0

In [ ]:
```python
housing_train.dtypes
```

Out[ ]:
```
Id                 int64
MSSubClass         int64
MSZoning          object
LotFrontage      float64
LotArea            int64
                  ...
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
Length: 81, dtype: object
```

In [ ]:
```python
print ("Train set: ", housing_train.shape)
print ("Test set: ", housing_test.shape)
```

```
Train set:  (1460, 81)
Test set:  (1459, 80)
```

In [ ]:
```python
housing_train.isnull().sum()
```

Out[ ]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
                ...
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 81, dtype: int64
```

In [ ]:
```python
housing_test.isnull().sum()
```

Out[ ]:
```
Id                 0
MSSubClass         0
MSZoning           4
LotFrontage      227
LotArea            0
                ...
MiscVal            0
MoSold             0
YrSold             0
SaleType           1
SaleCondition      0
Length: 80, dtype: int64
```

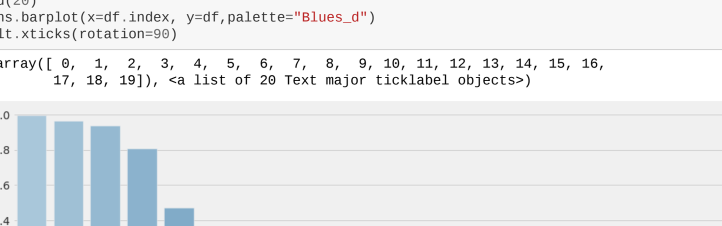
In [ ]:
```python
housing_test.loc[housing_test.SaleType.isnull()]
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig
1029	2490	20	RL	85.0	13770	Pave	NaN	Reg	Lvl	AllPub	Corner

In [ ]:
```python
#Top 20 features with missing data from housing_train

sns.set_style("whitegrid")
plt.style.use('fivethirtyeight')
plt.figure(figsize=(15,4))
df=pd.Series(1 - housing_train.count() / len(housing_train)).sort_values(ascending=False).he
ad(20)
sns.barplot(x=df.index, y=df,palette="Blues_d")
plt.xticks(rotation=90)
```

Out[ ]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19]), <a list of 20 Text major ticklabel objects>)
```



In [ ]:
```python
#Top 20 features with missing data from housing_test

sns.set_style("whitegrid")
plt.style.use('fivethirtyeight')
plt.figure(figsize=(15,4))
df=pd.Series(1 - housing_test.count() / len(housing_test)).sort_values(ascending=False).head
(20)
sns.barplot(x=df.index, y=df,palette="Blues_d")
plt.xticks(rotation=90)
```

Out[ ]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19]), <a list of 20 Text major ticklabel objects>)
```

In [ ]:
```python
#Houses sold by year housing_train

housing_train['YrSold'].value_counts()
```

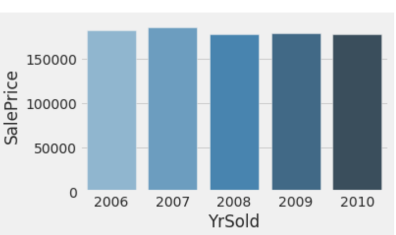
Out[ ]:
```
2009    338
2007    329
2006    314
2008    304
2010    175
Name: YrSold, dtype: int64
```

In [ ]:
```python
#Houses sold by year housing_test

housing_test['YrSold'].value_counts()
```

Out[ ]:
```
2007    363
2008    318
2009    309
2006    305
2010    164
Name: YrSold, dtype: int64
```

In [ ]:
```python
#Housing price = mean sale price by year

grp_year=housing_train.groupby('YrSold')
plt.figure(figsize=(5,3))

housing_years=grp_year['SalePrice'].mean().reset_index()
sns.barplot(x=housing_years.YrSold, y=housing_years['SalePrice'],palette="Blues_d")
plt.xticks(rotation=0)
```

Out[ ]:
```
(array([0, 1, 2, 3, 4]), <a list of 5 Text major ticklabel objects>)
```

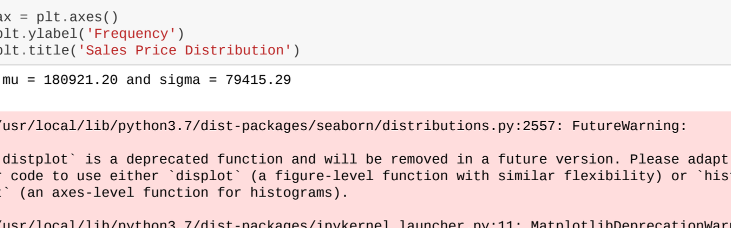


In [ ]:
```python
# Sales Price Distribution

plt.figure(figsize=(12,4))

sns.distplot(housing_train['SalePrice'] , fit=norm);
(mu, sigma) = norm.fit(housing_train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
plt.legend(['Normal Dustribution ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
           loc='upper right')

ax = plt.axes()
plt.ylabel('Frequency')
plt.title('Sales Price Distribution')
```

```
 mu = 180921.20 and sigma = 79415.29
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt you
r code to use either `displot` (a figure-level function with similar flexibility) or `histplo
t` (an axes-level function for histograms).

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: MatplotlibDeprecationWarnin
g:

Adding an axes using the same arguments as a previous axes currently reuses the earlier insta
nce.  In a future version, a new instance will always be created and returned.  Meanwhile, th
is warning can be suppressed, and the future behavior ensured, by passing a unique label to e
ach axes instance.
```

Out[ ]:
```
Text(0.5, 1.0, 'Sales Price Distribution')
```



In [ ]:
```python
housing_train_nona= housing_train.dropna()
print(housing_train_nona.shape)
print(housing_train.shape)
```

```
(0, 81)
(1460, 81)
```

In [ ]:
```python
num_missing = housing_train.isnull().sum()
percent = num_missing / housing_train.isnull().count()

df_missing = pd.concat([num_missing, percent], axis=1, keys=['MissingValues', 'Fraction'])
df_missing = df_missing.sort_values('Fraction', ascending=False)
df_missing[df_missing['MissingValues'] > 0]
```

Out[ ]:

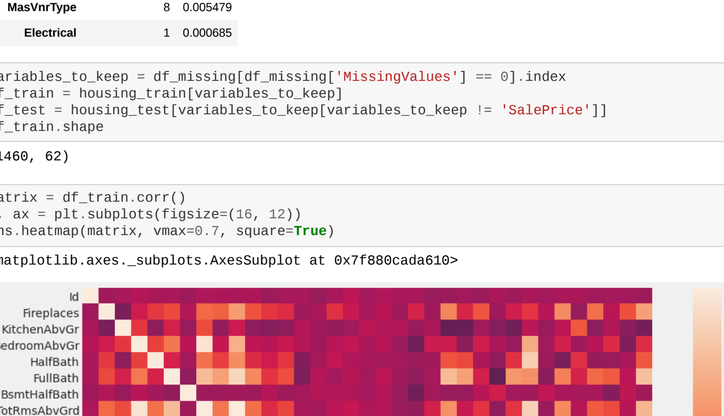
	MissingValues	Fraction
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
GarageCond	81	0.055479
GarageType	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtQual	37	0.025342
BsmtCond	37	0.025342
BsmtFinType1	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685

In [ ]:
```python
variables_to_keep = df_missing[df_missing['MissingValues'] == 0].index
df_train = housing_train[variables_to_keep]
df_test = housing_test[variables_to_keep[variables_to_keep != 'SalePrice']]
df_train.shape
```

Out[ ]:
```
(1460, 62)
```

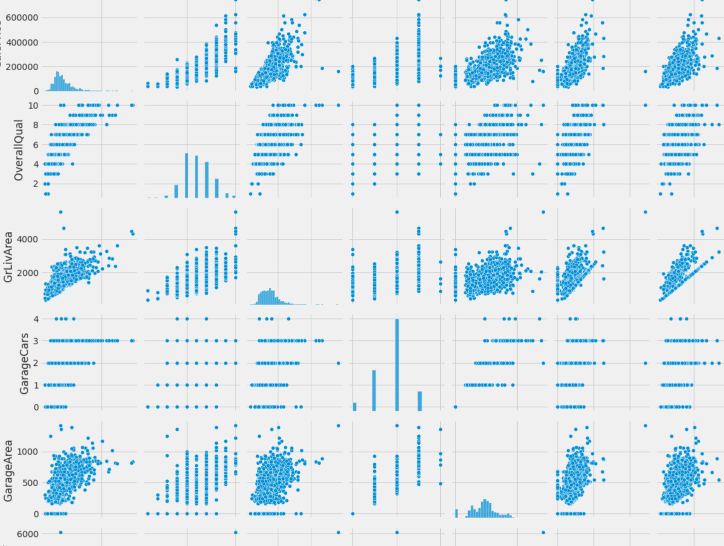
In [ ]:
```python
matrix = df_train.corr()
f, ax = plt.subplots(figsize=(16, 12))
sns.heatmap(matrix, vmax=0.7, square=True)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f880cada610>
```



In [ ]:
```python
# Filter out the target variables (SalePrice) and variables with a low correlation score (v
 such that -0.6 = 0.6]

interesting_variables = matrix['SalePrice'].sort_values(ascending=False)
interesting_variables = interesting_variables[abs(interesting_variables) >= 0.6]
cols = interesting_variables.index.values.tolist()
sns.pairplot(df_train[cols], size=2.5)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:1969: UserWarning:

The `size` parameter has been renamed to `height`; please update your code.
```



In [ ]:
```python
pred_vars = [v for v in interesting_variables.index.values if v != 'SalePrice']
target_var = 'SalePrice'

X = df_train[pred_vars]
y = df_train[target_var]
XT = df_test[pred_vars]
XT['GarageCars'].fillna(XT['GarageCars'].mean(), inplace=True)
XT['GarageArea'].fillna(XT['GarageArea'].mean(), inplace=True)
XT['TotalBsmtSF'].fillna(XT['TotalBsmtSF'].mean(), inplace=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py:4536: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
```

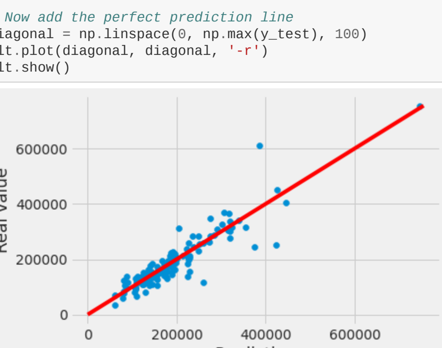
Out[ ]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

In [ ]:
```python
num_missing = XT.isnull().sum()
percent = num_missing / XT.isnull().count()

df_missing = pd.concat([num_missing, percent], axis=1, keys=['MissingValues', 'Fraction'])
df_missing = df_missing.sort_values('Fraction', ascending=False)
df_missing[df_missing['MissingValues'] > 0]
```

Out[ ]:

MissingValues	Fraction

In [ ]:
```python
y_pred = model.predict(X_test)
plt.scatter(y_pred, y_test)
plt.xlabel('Prediction')
plt.ylabel('Real value')

# Now add the perfect prediction line
diagonal = np.linspace(0, np.max(y_test), 100)
plt.plot(diagonal, diagonal, '-r')
plt.show()
```



In [ ]:
```python
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[ ]:
```
38583.69894272372
```

In [ ]:
```python
def perform_reg(clf, X, y):
    X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.2)
    clf.fit(X_train, y_train)
    y_preds = clf.predict(X_test)
    RMSE = np.sqrt(mean_squared_error(y_test, y_preds))
    return RMSE, y_preds
```

In [ ]:
```python
for clf in [RidgeCV(), ElasticNetCV(), LassoCV()]:
    print(clf.__class__.__name__)
    RMSE, y_preds = perform_reg(clf, X , y)
    print(RMSE)
```

```
RidgeCV
56373.317003244054
ElasticNetCV
56645.6301979143
LassoCV
44998.85320724976
```

In [ ]:
```python
RF_params= {'n_estimators': [100, 250, 500], 'max_depth': [2, 5, 10, None]}
regr = RandomForestRegressor()
gridRF= GridSearchCV(regr, RF_params, cv=5, scoring = make_scorer(mean_squared_error))
gridRF.fit(X, y)
```

Out[ ]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=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=100, n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [2, 5, 10, None],
                         'n_estimators': [100, 250, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(mean_squared_error), verbose=0)
```

In [ ]:
```python
print(gridRF.best_params_)
print(gridRF.best_estimator_)
print(gridRF.best_index_)
print(gridRF.best_score_)
```

```
{'max_depth': 2, 'n_estimators': 100}
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=2, max_features='auto', max_leaf_nodes=None,
                      max_samples=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=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
0
2116589408.7256362
```

In [ ]:
```python
gridRF.scorer_
```

Out[ ]:
```
make_scorer(mean_squared_error)
```

In [ ]:
```python
y_pred = gridRF.best_estimator_.predict(XT)
```

In [ ]:
```python
GB_params= {'n_estimators': [100, 250, 500], 'max_depth': [2, 3, 5], 'learning_rate' : [0.1,
0.25, 0.5]}
gbrt = GradientBoostingRegressor()
gridGB= GridSearchCV(gbrt, GB_params, cv=5, scoring = make_scorer(mean_squared_error))
gridGB.fit(X, y)
```

Out[ ]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.1,
                                                 loss='ls', max_depth=3,
                                                 max_features=None,
                                                 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=100,
                                                 n_iter_n...e,
                                                 presort='deprecated',
                                                 random_state=None,
                                                 subsample=1.0, tol=0.0001,
                                                 validation_fraction=0.1,
                                                 verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.1, 0.25, 0.5],
                         'max_depth': [2, 3, 5],
                         'n_estimators': [100, 250, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(mean_squared_error), verbose=0)
```

In [ ]:
```python
print(gridGB.best_params_)
print(gridGB.best_estimator_)
print(gridGB.best_index_)
print(gridGB.best_score_)
print(np.sqrt(gridRF.best_score_), np.sqrt(gridGB.best_score_) )
```

```
{'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 500}
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.5, loss='ls', max_depth=3,
                          max_features=None, 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=500,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
23
1514406641.1345448
46006.40617050669 38915.37795183987
```

In [ ]:
```python
y_prediction = gridGB.best_estimator_.predict(XT)
```

In [ ]:
```python
print(XT.shape)
print(housing_test.shape)
print(y_pred.shape)
print(y_prediction.shape)
```

```
(1459, 6)
(1459, 80)
(1459,)
(1459,)
```

In [ ]:
```python
pd.DataFrame(y_prediction)
pd.DataFrame(y_pred)
```

Out[ ]:

	0
0	133664.138163
1	140649.866542
2	149396.162697
3	155991.532851
4	266422.704043
...	...
1454	133664.138163
1455	133664.138163
1456	135191.819079
1457	133664.138163
1458	211642.656999

1459 rows × 1 columns

In [ ]:
```python
Identification = housing_test['Id'].copy()
SalesPriceRF = pd.DataFrame(y_pred)
SalesPriceRF.columns= ['SalePrice']
RandForest= pd.concat([Identification, SalesPriceRF], axis= 1)

SalesPriceGB = pd.DataFrame(y_prediction)
SalesPriceGB.columns= ['Sale Price']
GradBoost= pd.concat([Identification, SalesPriceGB], axis= 1)
RandForest
```

Out[ ]:

	Id	SalePrice
0	1461	133664.138163
1	1462	140649.866542
2	1463	149396.162697
3	1464	155991.532851
4	1465	266422.704043
...	...	...
1454	2915	133664.138163
1455	2916	133664.138163
1456	2917	135191.819079
1457	2918	133664.138163
1458	2919	211642.656999

1459 rows × 2 columns

In [ ]:
```python
pd.DataFrame(GradBoost).to_csv('RandomForest Gradient Boosted Sale Prices.csv')
pd.DataFrame(RandForest).to_csv('RandomForest Sale Prices.csv')
```

4 submissions for Michael Venit

Sort by Most recent

All Successful Selected

Submission and Description	Public Score
RandomForest Gradient Boosted Sale Prices.csv a few seconds ago by Michael Venit Random Forest Gradient Boosted model	0.20640
RandomForest Sale Price.csv a minute ago by Michael Venit Random Forest Model	0.26096