Assignment 2: Part 1

David Thor - Practical Maching Learning and AI (MSDS 422 - Winter 2022)

Part 0 - Importing Data

In [1]:

```
# What does this do? This will render HTML content.
from IPython.display import HTML
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:

```
# note the use of 'consensual' package nicknames.
import os
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import linear_model
from scipy import stats
import matplotlib.gridspec as gridspec
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import Ridge
from sklearn.model_selection import KFold
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
%matplotlib inline
```

In [3]:

```
os.listdir()
```

Out[3]:

```
['.ipynb_checkpoints',
 'Assignment 2 - Home Prices-Copy1.ipynb',
 'Assignment2-Titanic.csv',
 'Assignment2-Titanicc.csv',
 'Assignment2home.csv',
 'Assignment2_final.ipynb',
 'data_description.txt',
 'sample_submission.csv',
 'sync-2-Winter-2022',
 'sync-3-Winter-2022',
```

```
 'test.csv',
 'Titanic',
 'train.csv']
```

In [4]:

```
trainDat = pd.read_csv('train.csv')
testDat =  pd.read_csv('test.csv')
```

In [5]:

```
trainDat.shape
testDat.shape
```

Out[5]:

```
(1460, 81)
```

Out[5]:

```
(1459, 80)
```

In [6]:

```
# Is SalePrice not in the test data?
set(trainDat.columns).difference(set(testDat.columns))
```

Out[6]:

```
{'SalePrice'}
```

In [7]:

```
set(testDat.columns).issubset(set(trainDat.columns))
```

Out[7]:

```
True
```

Part 1.1 - EDA / Cleaning / Transformations

In [8]:

```
#Commented these out for a cleaner PDF Export.

#trainDat.head()
#trainDat.describe()
#trainDat.info()
```

Handling nulls/missing values

In [9]:

```
total = trainDat.isnull().sum().sort_values(ascending = False)
pcg = (total / trainDat.isnull().count()).sort_values(ascending = False)
miss_val = pd.concat([total, pcg], axis = 1, keys = ['Total', 'Percentage'])
miss_val.head(20)
```

Out[9]:

	Total	Percentage
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
Id	0	0.000000

In [10]:

```
trainDat.shape
```

Out[10]:

```
(1460, 81)
```

In [11]:

```
## To assure an improved EDA effort from Assignment 1,
## I analyzed every feature
## This function makes it easier for the training and 'final exam'/test
## to perform EDA.
## This will also help prevent data leakage.
def eda(df):
    df.drop(['Id','Alley','GarageYrBlt','PoolQC','Fence','MiscFeature']\
            ,axis=1,inplace=True)
    df['MSZoning']=df['MSZoning'].fillna(df['MSZoning'].mode()[0])
    df['LotFrontage']=df['LotFrontage'].fillna(df['LotFrontage'].mean())
    df['LotArea']=df['LotArea'].fillna(df['LotArea'].mean())
    df['Street']=df['Street'].fillna(df['Street'].mode()[0])
    df['LotShape']=df['LotShape'].fillna(df['LotShape'].mode()[0])
    df['LandContour']=df['LandContour'].fillna(df['LandContour'].mode()[0])
    df['Utilities']=df['Utilities'].fillna(df['Utilities'].mode()[0])
    df['LotConfig']=df['LotConfig'].fillna(df['LotConfig'].mode()[0])
    df['LandSlope']=df['LandSlope'].fillna(df['LandSlope'].mode()[0])
    df['Neighborhood']=df['Neighborhood'].fillna(df['Neighborhood'].mode()[0])
    df['Condition1']=df['Condition1'].fillna(df['Condition1'].mode()[0])
    df['Condition2']=df['Condition2'].fillna(df['Condition2'].mode()[0])
    df['HouseStyle']=df['HouseStyle'].fillna(df['HouseStyle'].mode()[0])
```

```
df['OverallQual']=df['OverallQual'].fillna(df['OverallQual'].mode()[0])
df['OverallCond']=df['OverallCond'].fillna(df['OverallCond'].mode()[0])
df['YearBuilt']=df['YearBuilt'].fillna(df['YearBuilt'].mode()[0])
df['YearRemodAdd']=df['YearRemodAdd'].fillna(df['YearRemodAdd'].mode()[0])
df['RoofStyle']=df['RoofStyle'].fillna(df['RoofStyle'].mode()[0])
df['RoofMatl']=df['RoofMatl'].fillna(df['RoofMatl'].mode()[0])
df['Exterior1st']=df['Exterior1st'].fillna(df['Exterior1st'].mode()[0])
df['Exterior2nd']=df['Exterior2nd'].fillna(df['Exterior2nd'].mode()[0])
df['MasVnrType']=df['MasVnrType'].fillna(df['MasVnrType'].mode()[0])
df['ExterQual']=df['ExterQual'].fillna(df['ExterQual'].mode()[0])
df['ExterCond']=df['ExterCond'].fillna(df['ExterCond'].mode()[0])
df['Foundation']=df['Foundation'].fillna(df['Foundation'].mode()[0])
df['BsmtCond'].fillna("NA", inplace = True)
df['BsmtQual'].fillna("NA", inplace = True)
df['BsmtExposure'].fillna("NA", inplace = True)
df['BsmtFinType1'].fillna("NA", inplace = True)
df['BsmtFinType2'].fillna("NA", inplace = True)
df['FireplaceQu'].fillna("NA", inplace = True)
df['GarageType'].fillna("NA", inplace = True)
df['GarageFinish'].fillna("NA", inplace = True)
df['GarageQual'].fillna("NA", inplace = True)
df['GarageCond'].fillna("NA", inplace = True)
df['Heating']=df['Heating'].fillna(df['Heating'].mode()[0])
df['HeatingQC']=df['HeatingQC'].fillna(df['HeatingQC'].mode()[0])
df['CentralAir']=df['CentralAir'].fillna(df['CentralAir'].mode()[0])
df['Electrical']=df['Electrical'].fillna(df['Electrical'].mode()[0])
df['MoSold']=df['MoSold'].fillna(df['MoSold'].mode()[0])
df['YrSold']=df['YrSold'].fillna(df['YrSold'].mode()[0])
df['SaleType']=df['SaleType'].fillna(df['SaleType'].mode()[0])
df['SaleCondition']=df['SaleCondition'].\
    fillna(df['SaleCondition'].mode()[0])
df['KitchenQual']=df['KitchenQual'].fillna(df['KitchenQual'].mode()[0])
df['TotRmsAbvGrd']=df['TotRmsAbvGrd'].fillna(df['TotRmsAbvGrd'].mode()[0])
df['Functional']=df['Functional'].fillna(df['Functional'].mode()[0])
df['PavedDrive']=df['PavedDrive'].fillna(df['PavedDrive'].mode()[0])
df['Utilities']=df['Utilities'].fillna(df['Utilities'].mode()[0])
df['1stFlrSF']=df['1stFlrSF'].fillna(df['1stFlrSF'].mean())
df['2ndFlrSF']=df['2ndFlrSF'].fillna(df['2ndFlrSF'].mean())
df['LowQualFinSF']=df['LowQualFinSF'].fillna(df['LowQualFinSF'].mean())
df['GrLivArea']=df['GrLivArea'].fillna(df['GrLivArea'].mean())
df['WoodDeckSF']=df['WoodDeckSF'].fillna(df['WoodDeckSF'].median())
df['OpenPorchSF']=df['OpenPorchSF'].fillna(df['OpenPorchSF'].median())
df['EnclosedPorch']=df['EnclosedPorch'].\
    fillna(df['EnclosedPorch'].median())
df['3SsnPorch']=df['3SsnPorch'].fillna(df['3SsnPorch'].median())
df['ScreenPorch']=df['ScreenPorch'].fillna(df['ScreenPorch'].median())
df['PoolArea']=df['PoolArea'].fillna(df['PoolArea'].median())
df['BsmtFinSF1']=df['BsmtFinSF1'].fillna(0)
df['BsmtFinSF2']=df['BsmtFinSF2'].fillna(0)
df['BsmtUnfSF']=df['BsmtUnfSF'].fillna(0)
df['TotalBsmtSF']=df['TotalBsmtSF'].fillna(0)
df['BsmtFullBath']=df['BsmtFullBath'].fillna(0)
df['BsmtHalfBath']=df['BsmtHalfBath'].fillna(0)
df['FullBath']=df['FullBath'].fillna(0)
df['HalfBath']=df['HalfBath'].fillna(0)
df['TotRmsAbvGrd']=df['TotRmsAbvGrd'].fillna(0)
df['Fireplaces']=df['Fireplaces'].fillna(0)
```

```
    df['GarageCars']=df['GarageCars'].fillna(0)
    df['GarageArea']=df['GarageArea'].fillna(0)
    df['MiscVal']=df['MiscVal'].fillna(0)
    df['MasVnrArea']=df['MasVnrArea'].fillna(0)


    #df.dropna(inplace=True)
    return df
```

In [12]:

```
trainDat.shape
```

Out[12]:

```
(1460, 81)
```

In [13]:

```
trainDat = eda(trainDat)
```

In [14]:

```
sum(trainDat.isnull().sum())
trainDat.shape
#trainDat.dropna(inplace=True)
```

Out[14]:

```
0
```

Out[14]:

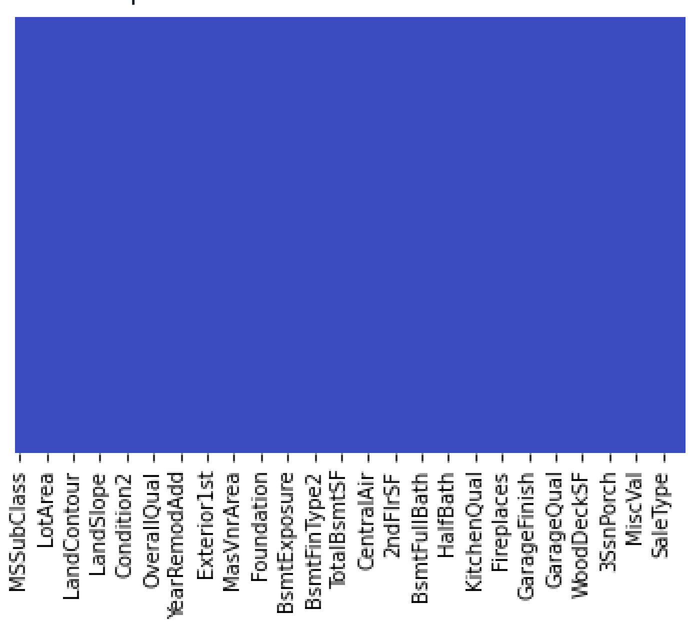
```
(1460, 75)
```

In [15]:

```
sns.heatmap(trainDat.isnull(),yticklabels=False,cbar=False,cmap='coolwarm')
```

Out[15]:

```
<AxesSubplot:>
```



Handling categorical feautures (Encoding)

At this point, we have analyzed and cleaned the data and ready for transformation.

In [16]:

```
#Get Categorical Values for potential use later
catagorical=trainDat.select_dtypes(include=['object']).copy()
categorical_values = catagorical.columns.tolist()
categorical_values.extend(['MSSubClass','OverallQual','OverallCond',\
                           'YearBuilt','YearRemodAdd','MoSold','YrSold'])
#trainDat.info()
#categorical_values
```

In [17]:

```
#Get Numerical Values for Standarization for use later
numb = trainDat.select_dtypes(include=['int64','float64']).copy()
num_values = numb.columns.tolist()
numerical_values = list(set(num_values).difference(set(categorical_values)))
#numerical_values
#print (len(numerical_values) + len(categorical_values))
```

In regards to the cell below...

- These are the location IDs of the numerical and categorical values.
- Based upon my domain knowldge of homes, I have selected these for transformation
- These will be passed into the columnTransformer

In [18]:

```
#trainDat.info()
#trainDat.head()

num = [2,3,32,34,35,36,41,42,43,44,45,46,47,48,49,50,\
       52,54,58,59,63,64,65,66,67,68,69]
cat = [10,11,12,15,16,17,25,26]
```

Column Transformer

I made a conscious decision to remove drop=first in the ColumnTransfor. I understand that drop=first is necessary for data redundancy purposes. And removing it can address multicollinearity.

However, I noticed some an issue with drop=first and the k-fold approach. An issue can arise when the first encoded value is dropped when it is the only value in that fold.

According a Huq, "if you are regularizing, there's no need to drop one of the one-hot encoded columns from each categorical feature (Huq, 2019)"

Huq, R. (2019, May 7). Think twice before dropping that first one-hot encoded column. In Machines We Trust. Retrieved January 29, 2022, from https://inmachineswetrust.com/posts/drop-first-columns/

In [19]:

```
# Col transform specs

ct = ColumnTransformer([('standardized',preprocessing.StandardScaler(),num),
       ('oneHotter', preprocessing.OneHotEncoder\
        (handle_unknown='ignore'),cat)])
```

```
ct
```

Out[19]:

```
ColumnTransformer(transformers=[('standardized', StandardScaler(),
                                 [2, 3, 32, 34, 35, 36, 41, 42, 43, 44, 45, 46,
                                  47, 48, 49, 50, 52, 54, 58, 59, 63, 64, 65,
                                  66, 67, 68, 69]),
                                ('oneHotter',
                                 OneHotEncoder(handle_unknown='ignore'),
                                 [10, 11, 12, 15, 16, 17, 25, 26])])
```

Preparing for K-Fold

In [20]:

```
y=trainDat.SalePrice.to_numpy(copy=True)
X=trainDat.loc[:,trainDat.columns!='SalePrice'].to_numpy(copy=True)
X.shape   # size
y.shape   # size
```

Out[20]:

```
(1460, 74)
```

Out[20]:

```
(1460,)
```

In [21]:

```
# training and test splits

kf=KFold(n_splits=10)
print(f'Number of data folds: {kf.get_n_splits()}')
```

```
Number of data folds: 10
```

In [22]:

```
alphas=[0.001,0.01,0.1,1.0,10.0]   # The sklearn default is 1.0
type(alphas)
```

Out[22]:

```
list
```

Summary of All Processing

- Ridge R2 score: 0.868
- Lasso R2 score: 0.882
- Elastic Net R2 Score: 0.817

Processing for Ridge

We're going to accumulate our results in a list of dictionaries. After we're with all our models, we can create a DataFrame using the list.

In [23]:

```
resListofDicts=[]                        # a list of results in dicts
for alphVal in alphas:  # Outer processing loop
    fold = 0            # fold counter
```

```
    for trainNdx, testNdx in kf.split(X): # cv loop. should do it 10 times.
        fold+=1
        Xtr = ct.fit_transform(X[trainNdx])# fit & transform X training fold
        Xval = ct.transform(X[testNdx])       # transform X test fold
        regMod=Ridge(alpha=alphVal)  # instantiate regressor
        fitMod=regMod.fit(Xtr,y[trainNdx])          # fitted
        predtr = fitMod.predict(Xtr)            # training pred values
        predval = fitMod.predict(Xval)          # test pred values
        msetr = metrics.mean_squared_error(y[trainNdx],predtr)
        mseval = metrics.mean_squared_error(y[testNdx],predval)
        resDict={'alpha': alphVal,'fold': fold,
                'trainMSE':msetr,'testMSE':mseval}
        resListofDicts.append(resDict)
```

In [24]:

```
resultsDF=pd.DataFrame(resListofDicts)
resultsDF.shape
resultsDF.columns
```

Out[24]:

```
(50, 4)
```

Out[24]:

```
Index(['alpha', 'fold', 'trainMSE', 'testMSE'], dtype='object')
```

In [25]:

```
resultsSummaryDF=resultsDF.groupby(['alpha'],as_index=False).agg\
({'trainMSE':['mean','std'],'testMSE':['mean','std']})
resultsSummaryDF
```

Out[25]:

	alpha	trainMSE		testMSE	
		mean	std	mean	std
0	0.001	7.222350e+08	8.071926e+07	1.266934e+09	1.103306e+09
1	0.010	7.225557e+08	8.048194e+07	1.259557e+09	1.092181e+09
2	0.100	7.232257e+08	8.046712e+07	1.240384e+09	1.077628e+09
3	1.000	7.371071e+08	7.902255e+07	1.160424e+09	1.010922e+09
4	10.000	8.273885e+08	7.959495e+07	1.123903e+09	9.923856e+08

In [26]:

```
resultsSummaryDF.columns=resultsSummaryDF.columns.droplevel(0)
```

In [27]:

```
resultsSummaryDF.columns=['alpha','train_mean_MSE','train_std_MSE',\
                          'test_mean_MSE',
                         'test_std_MSE']
```

In [28]:

```
resultsSummaryDF.sort_values(by='test_mean_MSE',ascending=True)
```

Out[28]:

	alpha	train_mean_MSE	train_std_MSE	test_mean_MSE	test_std_MSE
4	10.000	8.273885e+08	7.959495e+07	1.123903e+09	9.923856e+08
3	1.000	7.371071e+08	7.902255e+07	1.160424e+09	1.010922e+09
2	0.100	7.232257e+08	8.046712e+07	1.240384e+09	1.077628e+09
1	0.010	7.225557e+08	8.048194e+07	1.259557e+09	1.092181e+09
0	0.001	7.222350e+08	8.071926e+07	1.266934e+09	1.103306e+09

Refit Ridge

- It seems the alpha with the lowest MSE is 10.0.
- In this case, I am going to use 10 as the alpha.

In [29]:

```
Xtrans=ct.fit_transform(X)
```

In [30]:

```
Xtrans.shape
```

Out[30]:

```
(1460, 209)
```

In [31]:

```
regMod=Ridge(alpha=10.0)
fitMod=regMod.fit(Xtrans,y)
```

In [32]:

```
predy=fitMod.predict(Xtrans)
```

In [33]:

```
R2=fitMod.score(Xtrans,y)
print(f'R2, all training data: {R2.round(3)}')
```

```
R2, all training data: 0.868
```

In [34]:

```
## Scatterplot of actual vs. predict

predDF=pd.DataFrame({'SalePrice':y,'PredPrice': predy})

sns.set(rc={"figure.figsize":(6, 6)}) #width=3, #height=4

x_plot=np.linspace(50000,500000,100)
y_plot=x_plot

scatter=sns.scatterplot(data=predDF,x='PredPrice',y='SalePrice',
                alpha=0.35).set(title="SalePrice: Actual vs. Predicted")
plt.plot(x_plot,y_plot,c='r')
plt.show();
```

Lasso

- This one was a bit trickier because I had to troubleshoot why it wasn't converging.
- According to online sources, a workaround was normalize=True

In [35]:

```
resListofDicts=[]                        # a list of results in dicts
for alphVal in alphas:  # Outer processing loop
    fold = 0           # fold counter
    for trainNdx, testNdx in kf.split(X): # cv loop. should do it 10 times.
        fold+=1
        Xtr = ct.fit_transform(X[trainNdx])# fit & transform X training fold
        Xval = ct.transform(X[testNdx])       # transform X test fold
        lasMod=Lasso(alpha=alphVal, normalize=True, \
                     max_iter=100000, tol=1e-2)  # instantiate regressor
        fitMod=lasMod.fit(Xtr,y[trainNdx])          # fitted
        predtr = fitMod.predict(Xtr)                # training pred values
        predval = fitMod.predict(Xval)              # test pred values
        msetr = metrics.mean_squared_error(y[trainNdx],predtr)
        mseval = metrics.mean_squared_error(y[testNdx],predval)
        resDict={'alpha': alphVal,'fold': fold,
                'trainMSE':msetr,'testMSE':mseval}
        resListofDicts.append(resDict)
```

In [36]:

```
resultsDF=pd.DataFrame(resListofDicts)
resultsDF.shape
resultsDF.columns
```

Out[36]:

```
(50, 4)
```

Out[36]:

```
Index(['alpha', 'fold', 'trainMSE', 'testMSE'], dtype='object')
```

In [37]:

```
resultsSummaryDF=resultsDF.groupby(['alpha'],as_index=False).agg\
({'trainMSE':['mean','std'],'testMSE':['mean','std']})
resultsSummaryDF
```

Out[37]:

	alpha	trainMSE		testMSE	
		mean	std	mean	std
0	0.001	7.198096e+08	8.040821e+07	1.269224e+09	1.104315e+09
1	0.010	7.198097e+08	8.040821e+07	1.269070e+09	1.104211e+09
2	0.100	7.198134e+08	8.040806e+07	1.267504e+09	1.103501e+09
3	1.000	7.202509e+08	8.040615e+07	1.255050e+09	1.099368e+09
4	10.000	7.456012e+08	8.004976e+07	1.198347e+09	1.052691e+09

In [38]:

```
resultsSummaryDF.columns=resultsSummaryDF.columns.droplevel(0)
```

In [39]:

```
resultsSummaryDF.columns=['alpha','train_mean_MSE','train_std_MSE',\
                           'test_mean_MSE',
                          'test_std_MSE']
```

In [40]:

```
resultsSummaryDF.sort_values(by='test_mean_MSE',ascending=True)
```

Out[40]:

	alpha	train_mean_MSE	train_std_MSE	test_mean_MSE	test_std_MSE
4	10.000	7.456012e+08	8.004976e+07	1.198347e+09	1.052691e+09
3	1.000	7.202509e+08	8.040615e+07	1.255050e+09	1.099368e+09
2	0.100	7.198134e+08	8.040806e+07	1.267504e+09	1.103501e+09
1	0.010	7.198097e+08	8.040821e+07	1.269070e+09	1.104211e+09
0	0.001	7.198096e+08	8.040821e+07	1.269224e+09	1.104315e+09

Refit for Lasso

- Alpha 10 yield in lowest MSE test/train Score.
- The results were a bit strange here. Consistently and proportionly higher MSE compared to the test MSE across the Alpha range. I'm thinking this may have to do with my paramters/encoding?

In [41]:

```
lasMod=Lasso(alpha=10.0)
fitMod=lasMod.fit(Xtrans,y)
```

In [42]:

```
predy=fitMod.predict(Xtrans)
```

In [43]:

```
R2=fitMod.score(Xtrans,y)
print(f'R2, all training data: {R2.round(3)}')
```

```
R2, all training data: 0.882
```

In [44]:

```
## Scatterplot of actual vs. predict

predDF=pd.DataFrame({'SalePrice':y,'PredPrice': predy})

sns.set(rc={"figure.figsize":(6, 6)}) #width=3, #height=4

x_plot=np.linspace(50000,500000,100)
y_plot=x_plot

scatter=sns.scatterplot(data=predDF,x='PredPrice',y='SalePrice',
                alpha=0.35).set(title="SalePrice: Actual vs. Predicted")
plt.plot(x_plot,y_plot,c='r')
plt.show();
```

Elastic

- Switching it up and using GridSearCV for ElasticNet

In [45]:

```
# define model
model = ElasticNet()
# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['alpha'] = [0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 1]
grid['l1_ratio'] = (0, 1, 0.01)
# define search
search = GridSearchCV(model, grid, scoring='neg_mean_absolute_error', \
                      cv=None, n_jobs=-1)
# perform the search
results = search.fit(ct.fit_transform(X), y)
# summarize
results.get_params

print('Config: %s' % results.best_params_)
```

Out[45]:

```
<bound method BaseEstimator.get_params of GridSearchCV(estimator=ElasticNet(), n_jobs=-1,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 1],
                         'l1_ratio': (0, 1, 0.01)},
             scoring='neg_mean_absolute_error')>
Config: {'alpha': 0.01, 'l1_ratio': 0.01}
```

In [46]:

```
#Commented for a cleaner PDF output.
#pd.DataFrame(results.cv_results_)
```

Refit for Elastic Net

- Alpha of 0.1. Ratio of 0.1 as these were the best parameters from above

In [47]:

```
elasmodel = ElasticNet(alpha=0.1, l1_ratio=0.1)
fitMod=elasmodel.fit(Xtrans,y)

predy=fitMod.predict(Xtrans)
```

In [48]:

```
R2=fitMod.score(Xtrans,y)
print(f'R2, all training data: {R2.round(3)}')
```

```
R2, all training data: 0.817
```

In [49]:

```
## Scatterplot of actual vs. predict

predDF=pd.DataFrame({'SalePrice':y,'PredPrice': predy})

sns.set(rc={"figure.figsize":(6, 6)}) #width=3, #height=4

x_plot=np.linspace(50000,500000,100)
y_plot=x_plot
```

```
scatter=sns.scatterplot(data=predDF,x='PredPrice',y='SalePrice',
                alpha=0.35).set(title="SalePrice: Actual vs. Predicted")
plt.plot(x_plot,y_plot,c='r')
plt.show();
```

Implement/Submission

- I have decided to use the Ridge Model for my submision.

In [50]:

```
testDat =  pd.read_csv('test.csv')
testDat.shape
```

Out[50]:

```
(1459, 80)
```

In [51]:

```
#eda was the function I created earlier for EDA
testDat = eda(testDat)
testDat.shape
```

Out[51]:

```
(1459, 74)
```

In [52]:

```
testDat = testDat.to_numpy(copy=True)
```

In [53]:

```
testDat.shape
X_final_exam=ct.transform(testDat)
#testDat.info()
```

```
#sum(testDat.isnull().sum())
```

Out[53]:

```
(1459, 74)
```

In [54]:

```
X_final_exam.shape
Xtrans.shape

#The test data is now clean and ready for predicitions
```

Out[54]:

```
(1459, 209)
```

Out[54]:

```
(1460, 209)
```

In [55]:

```
##Training the Model again for prediction in the next cell
regMod=Ridge(alpha=10.0)
fitMod=regMod.fit(Xtrans,y)
```

In [56]:

```
predy=fitMod.predict(X_final_exam)
len(predy)
```

Out[56]:

```
1459
```

In [57]:

```
df_test = pd.read_csv('test.csv')
Regresult = pd.DataFrame(predy, columns=['SalePrice'])
df_test['Id'].shape, Regresult.shape
test_t = pd.DataFrame(df_test["Id"])
```

Out[57]:

```
((1459,), (1459, 1))
```

In [58]:

```
my_submission = pd.concat([test_t, Regresult ], axis=1)
```

In [59]:

```
my_submission
```

Out[59]:

	Id	SalePrice
0	1461	103298.297621
1	1462	158044.502216
2	1463	187157.679871
3	1464	191743.040639
4	1465	223102.013096
...	...	...