Machine Learning Pipeline:

Code Blocks needed for final model deployment

In [2]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear_model import Lasso

from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt

# to persist the model and the scaler
import joblib

pd.set_option('display.max_columns', None)

import warnings
warnings.simplefilter(action='ignore')
```

In [3]:
```
df = pd.read_csv('houseprice.csv')
print(df.shape)
df.head()
```

```
(1460, 81)
```

Out[3]:

	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

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(
    df,
    df['SalePrice'],
    test_size=0.1,
    # setting the seed here
    random_state=0)

X_train.shape, X_test.shape
```

Out[4]:
```
((1314, 81), (146, 81))
```

In [5]:
```
X_train.head()
```

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utili

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utili
930	931	20	RL	73.0	8925	Pave	NaN	IR1	HLS	All
656	657	20	RL	72.0	10007	Pave	NaN	IR1	Lvl	All
45	46	120	RL	61.0	7658	Pave	NaN	Reg	Lvl	All
1348	1349	20	RL	NaN	16196	Pave	NaN	IR3	Low	All
55	56	20	RL	100.0	10175	Pave	NaN	IR1	Lvl	All

Selected Features:

In [8]:

```
features = pd.read_csv('selected_features.csv')
features.head()
```

Out[8]:

	MSSubClass
0	MSZoning
1	Neighborhood
2	OverallQual
3	OverallCond
4	YearRemodAdd

In [10]:

```
# Added the extra feature, LotFrontage
features = features['MSSubClass'].tolist() + ['LotFrontage']

print('Number of features: ', len(features))
```

```
Number of features:  22
```

Dealing with Missing Values:

Categorical Variables

In [11]:

```
# make a list of the categorical variables that contain missing values

mis_var = [
    var for var in features
    if X_train[var].isnull().sum() > 0 and X_train[var].dtypes == 'O'
]

# display categorical variables that we will engineer:
mis_var
```

Out[11]:

```
['MasVnrType',
 'BsmtQual',
 'BsmtExposure',
 'FireplaceQu',
 'GarageType',
 'GarageFinish']
```

Note that we have much less categorical variables with missing values than in our original dataset. But we still use categorical variables with NA for the final model, so we need to

include this piece of feature engineering logic in the deployment pipeline.

In [12]:
```
X_train[mis_var] = X_train[mis_var].fillna('Missing')
X_test[mis_var] = X_test[mis_var].fillna('Missing')

# check that we have no missing information in the engineered variables
X_train[mis_var].isnull().sum()
```

Out[12]:
```
MasVnrType      0
BsmtQual        0
BsmtExposure    0
FireplaceQu     0
GarageType      0
GarageFinish    0
dtype: int64
```

Numerical variables

To engineer missing values in numerical variables, we will:

- **add a binary missing value indicator variable**
- **and then replace the missing values in the original variable with the mode**

In [13]:
```
# make a list of the numerical variables that contain missing values:

mis_var = [
    var for var in features
    if X_train[var].isnull().sum() > 0 and X_train[var].dtypes != 'O'
]

# display numerical variables with NA
mis_var
```

Out[13]:
```
['LotFrontage']
```

In [14]:
```
var = 'LotFrontage'

# calculate the mode
mode_val = X_train[var].mode()[0]
print('mode of LotFrontage: {}'.format(mode_val))

# replace missing values by the mode
# (in train and test)
X_train[var] = X_train[var].fillna(mode_val)
X_test[var] = X_test[var].fillna(mode_val)
```

```
mode of LotFrontage: 60.0
```

Temporal variables

One of our temporal variables was selected to be used in the final model: 'YearRemodAdd'

So we need to deploy the bit of code that creates it.

In [15]:
```
def elapsed_years(df, var):
    # capture difference between year variable
    # and year in which the house was sold
```

```
    df[var] = df['YrSold'] - df[var]

    return df
```

In [16]:

```
X_train = elapsed_years(X_train, 'YearRemodAdd')
X_test = elapsed_years(X_test, 'YearRemodAdd')
```

Numerical variable transformation

In [17]:

```
# we apply the logarithmic function to the variables that
# were selected (and the target):

for var in ['LotFrontage', '1stFlrSF', 'GrLivArea', 'SalePrice']:
    X_train[var] = np.log(X_train[var])
    X_test[var] = np.log(X_test[var])
```

Categorical Variables

Group Rare Labels

In [18]:

```
cat_vars = [var for var in features if X_train[var].dtype == 'O']

cat_vars
```

Out[18]:

```
['MSZoning',
 'Neighborhood',
 'RoofStyle',
 'MasVnrType',
 'BsmtQual',
 'BsmtExposure',
 'HeatingQC',
 'CentralAir',
 'KitchenQual',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'PavedDrive']
```

In [19]:

```
def find_frequent_labels(df, var, rare_perc):

    # function finds the labels that are shared by more than
    # a certain % of the houses in the dataset

    df = df.copy()

    tmp = df.groupby(var)['SalePrice'].count() / len(df)

    return tmp[tmp > rare_perc].index


for var in cat_vars:

    # find the frequent categories
    frequent_ls = find_frequent_labels(X_train, var, 0.01)
    print(var)
    print(frequent_ls)
    print()

    # replace rare categories by the string "Rare"
```

```
    X_train[var] = np.where(X_train[var].isin(
        frequent_ls), X_train[var], 'Rare')

    X_test[var] = np.where(X_test[var].isin(
        frequent_ls), X_test[var], 'Rare')
```

```
MSZoning
Index(['FV', 'RH', 'RL', 'RM'], dtype='object', name='MSZoning')

Neighborhood
Index(['Blmngtn', 'BrDale', 'BrkSide', 'ClearCr', 'CollgCr', 'Crawfor',
       'Edwards', 'Gilbert', 'IDOTRR', 'MeadowV', 'Mitchel', 'NAmes', 'NWAmes',
       'NoRidge', 'NridgHt', 'OldTown', 'SWISU', 'Sawyer', 'SawyerW',
       'Somerst', 'StoneBr', 'Timber'],
      dtype='object', name='Neighborhood')

RoofStyle
Index(['Gable', 'Hip'], dtype='object', name='RoofStyle')

MasVnrType
Index(['BrkFace', 'None', 'Stone'], dtype='object', name='MasVnrType')

BsmtQual
Index(['Ex', 'Fa', 'Gd', 'Missing', 'TA'], dtype='object', name='BsmtQual')

BsmtExposure
Index(['Av', 'Gd', 'Missing', 'Mn', 'No'], dtype='object', name='BsmtExposure')

HeatingQC
Index(['Ex', 'Fa', 'Gd', 'TA'], dtype='object', name='HeatingQC')

CentralAir
Index(['N', 'Y'], dtype='object', name='CentralAir')

KitchenQual
Index(['Ex', 'Fa', 'Gd', 'TA'], dtype='object', name='KitchenQual')

FireplaceQu
Index(['Ex', 'Fa', 'Gd', 'Missing', 'Po', 'TA'], dtype='object', name='FireplaceQu')

GarageType
Index(['Attchd', 'Basment', 'BuiltIn', 'Detchd', 'Missing'], dtype='object', name='Garag
eType')

GarageFinish
Index(['Fin', 'Missing', 'RFn', 'Unf'], dtype='object', name='GarageFinish')

PavedDrive
Index(['N', 'P', 'Y'], dtype='object', name='PavedDrive')
```

Encoding Categorical Variables

In [20]:

```
def replace_categories(train, test, var, target):

    # order the categories in a variable from that with the lowest
    # house sale price, to that with the highest
    ordered_labels = train.groupby([var])[target].mean().sort_values().index

    # create a dictionary of ordered categories to integer values
    ordinal_label = {k: i for i, k in enumerate(ordered_labels, 0)}

    # use the dictionary to replace the categorical strings by integers
```

```
    train[var] = train[var].map(ordinal_label)
    test[var] = test[var].map(ordinal_label)

    print(var)
    print(ordinal_label)
    print()
```

In [21]:
```
for var in cat_vars:
    replace_categories(X_train, X_test, var, 'SalePrice')
```

```
MSZoning
{'Rare': 0, 'RM': 1, 'RH': 2, 'RL': 3, 'FV': 4}

Neighborhood
{'IDOTRR': 0, 'MeadowV': 1, 'BrDale': 2, 'Edwards': 3, 'BrkSide': 4, 'OldTown': 5, 'Sawy
er': 6, 'SWISU': 7, 'NAmes': 8, 'Mitchel': 9, 'SawyerW': 10, 'Rare': 11, 'NWAmes': 12,
'Gilbert': 13, 'Blmngtn': 14, 'CollgCr': 15, 'Crawfor': 16, 'ClearCr': 17, 'Somerst': 1
8, 'Timber': 19, 'StoneBr': 20, 'NridgHt': 21, 'NoRidge': 22}

RoofStyle
{'Gable': 0, 'Rare': 1, 'Hip': 2}

MasVnrType
{'None': 0, 'Rare': 1, 'BrkFace': 2, 'Stone': 3}

BsmtQual
{'Missing': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4}

BsmtExposure
{'Missing': 0, 'No': 1, 'Mn': 2, 'Av': 3, 'Gd': 4}

HeatingQC
{'Rare': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4}

CentralAir
{'N': 0, 'Y': 1}

KitchenQual
{'Fa': 0, 'TA': 1, 'Gd': 2, 'Ex': 3}

FireplaceQu
{'Po': 0, 'Missing': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}

GarageType
{'Missing': 0, 'Rare': 1, 'Detchd': 2, 'Basment': 3, 'Attchd': 4, 'BuiltIn': 5}

GarageFinish
{'Missing': 0, 'Unf': 1, 'RFn': 2, 'Fin': 3}

PavedDrive
{'N': 0, 'P': 1, 'Y': 2}
```

In [22]:
```
[var for var in features if X_train[var].isnull().sum() > 0]
```

Out[22]: []

In [23]:
```
[var for var in features if X_test[var].isnull().sum() > 0]
```

Out[23]: []

Feature Scaling

In [24]:
```
# capture the target
y_train = X_train['SalePrice']
y_test = X_test['SalePrice']
```

In [25]:
```
scaler = MinMaxScaler()

# train scaler
scaler.fit(X_train[features])
```

Out[25]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

In [26]:
```
# explore maximum values of variables
scaler.data_max_
```

Out[26]:
```
array([ 4.        , 22.        , 10.        ,  9.        , 60.        ,
        2.        ,  3.        ,  4.        ,  4.        ,  4.        ,
        1.        ,  8.45361421,  8.63799389,  3.        ,  3.        ,
        3.        ,  5.        ,  5.        ,  3.        ,  4.        ,
        2.        ,  5.74620319])
```

In [27]:
```
# explore minimum values of variables
scaler.data_min_
```

Out[27]:
```
array([ 0.        ,  0.        ,  1.        ,  1.        , -1.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  5.81114099,  5.81114099,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  3.04452244])
```

In [28]:
```
X_train = scaler.transform(X_train[features])
X_test = scaler.transform(X_test[features])
```

Train the Linear Regression Model : Lasso Regression Algorithm

In [29]:
```
lin_model = Lasso(alpha=0.005, random_state=0)

# train the model
lin_model.fit(X_train, y_train)

# we persist the model for future use
joblib.dump(lin_model, 'lasso_regression.pkl')
```

Out[29]:
```
['lasso_regression.pkl']
```

In [30]:
```
pred = lin_model.predict(X_train)

# determine mse and rmse
print('Training MSE: {}'.format(int(
    mean_squared_error(np.exp(y_train), np.exp(pred)))))
print('Training RMSE: {}'.format(int(
    sqrt(mean_squared_error(np.exp(y_train), np.exp(pred))))))
print('Training R2: {}'.format(
    r2_score(np.exp(y_train), np.exp(pred))))
print()

# make predictions for test set
pred = lin_model.predict(X_test)
```

```
# determine mse and rmse
print('Test MSE: {}'.format(int(
    mean_squared_error(np.exp(y_test), np.exp(pred)))))
print('Test RMSE: {}'.format(int(
    sqrt(mean_squared_error(np.exp(y_test), np.exp(pred))))))
print('Test R2: {}'.format(
    r2_score(np.exp(y_test), np.exp(pred))))
print()

print('Average House Price: ', int(np.exp(y_train).median()))
```

```
Training MSE: 1095464701
Training RMSE: 33097
Training R2: 0.8245524987165784

Test MSE: 1415749527
Test RMSE: 37626
Test R2: 0.7939863537248242

Average House Price:  163000
```

In [ ]: