house_prices_jul15

July 29, 2019

[1]:
```
# kaggle competition src:
# https://www.kaggle.com/c/house-prices-advanced-regression-techniques/
```

[2]:
```
# Load libraries

import os

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

import numpy as np

from scipy import stats as scstats
from scipy.special import boxcox1p

from sklearn.preprocessing import MinMaxScaler, LabelEncoder, RobustScaler
from sklearn.model_selection import cross_val_score, KFold, train_test_split,␣
↪GridSearchCV
from sklearn.metrics import mean_squared_log_error, r2_score
from sklearn.pipeline import make_pipeline

from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR

from xgboost import XGBRegressor
```

[3]:
```
# Load datasets

data_dir_path = os.path.join(os.getcwd(), 'data')
```

```
train_df = pd.read_csv(os.path.join(data_dir_path, 'train.csv'))
test_df = pd.read_csv(os.path.join(data_dir_path, 'test.csv'))
```

[4]:
```
concat_train_test_df = pd.concat([train_df, test_df], ignore_index=True,␣
↪sort=False)

concat_train_test_df = concat_train_test_df.drop('Id', axis=1)
```

[5]:
```
# Explore loaded data.

def display_dataset_overview(dataset_df):
    """Display basic information about dataset"""
    # Data inside
    display(dataset_df.head(3))
    display(dataset_df.tail(3))
    # Shape
    display(dataset_df.shape)
    # .describe output
    display(dataset_df.describe(include='all').T)


def display_dataset_col_dtypes(dataset_df):
    """Display dataset columns and its dtypes"""
    # All columns and their dtypes
    display(dataset_df.dtypes.unique())
    display(dataset_df.select_dtypes(include='int64').columns.values)
    display(dataset_df.select_dtypes(include='float64').columns.values)
    display(dataset_df.select_dtypes(include='object').columns.values)
    display(dataset_df.select_dtypes(include='number').columns.values)
```

[6]:
```
# display_dataset_overview(train_df)

# display_dataset_overview(test_df)

# display_dataset_overview(concat_train_test_df)
```

[7]:
```
# Explore distributions of continuous features in certain dataset

def display_hist(dataset_df, col_name, n_bins=25):
    """Display histogram for dataset[col_name] values"""
    plt.figure(figsize=(15, 10))
    dataset_df[col_name].hist(bins=n_bins)
    plt.show()


def display_all_numerical_hist(set1_df, set2_df, n_bins=25):
    """Display histograms for every numerical feature from set1_df and␣
↪set2_df"""
    concat_df = pd.concat([set1_df, set2_df], ignore_index=True, sort=False)
```

```
    numeric_col_names = concat_df.select_dtypes(include='number').columns.values
    for col_name in numeric_col_names:
        fig, [ax_0, ax_1, ax_2] = plt.subplots(1, 3, figsize=(15, 5))
        ax_0.set_title('set1 {0}'.format(col_name))
        set1_df[col_name].hist(ax=ax_0, bins=n_bins)
        ax_1.set_title('set2 {0}'.format(col_name))
        set2_df[col_name].hist(ax=ax_1, bins=n_bins)
        ax_2.set_title('concat [set1, set2] {0}'.format(col_name))
        concat_df[col_name].hist(ax=ax_2, bins=n_bins)
        fig.tight_layout()
        plt.show()


def display_colx_coly_scatter(dataset_df, x_col_name, y_col_name, color=None):
    """Display scatterplot for {dataset_df[x_col_name], dataset_df[y_col_name]}␣
 ↪values"""
    plt.figure(figsize=(10, 10))
    sc = plt.scatter(x_col_name, y_col_name, data=dataset_df, c=color)
    plt.title("{0} - {1}".format(x_col_name, y_col_name))
    plt.xlabel(x_col_name)
    plt.ylabel(y_col_name)
    plt.show()


def display_all_numerical_scatter(set1_df, col_to_compare):
    """Display scatter plots for every numerical feature from set1_df and␣
 ↪col_to_compare column values"""
    numeric_col_names = set1_df.select_dtypes(include='number').columns.values
    for col_name in numeric_col_names:
        display_colx_coly_scatter(set1_df, col_name, col_to_compare,␣
 ↪color=col_name)
```

[8]:
```
# display_hist(train_df, 'SalePrice', 100)

# display_all_numerical_hist(
#     train_df.drop(['SalePrice', 'Id'], axis=1),
#     test_df.drop('Id', axis=1)
# )

# display_colx_coly_scatter(train_df, 'GrLivArea', 'SalePrice',␣
 ↪color='YearBuilt')

# display_all_numerical_scatter(
#     train_df.drop(['Id'], axis=1),
#     'SalePrice'
# )
```

[9]:
```
# Explore distributions of categorical features in certain dataset

def display_col_freqtable(dataset_df, col_name):
    """Display frequency table for dataset_df[col_name] values"""
    display(
        pd.crosstab(
            index=dataset_df[col_name],
            columns="count"
        ).sort_values(by='count', ascending=False)
    )


def display_all_categorical_freq_bar(set1_df, set2_df):
    """Display frequency table and barplot for each categorical feature"""
    concat_df = pd.concat([set1_df, set2_df], ignore_index=True, sort=False)
    numeric_col_names = concat_df.select_dtypes(include='object').columns.values
    for col_name in numeric_col_names:
        fig, axes = plt.subplots(1, 3, figsize=(15, 5))
        axes[0].set_title('set1 {0}'.format(col_name))
        set1_df[col_name].value_counts().plot(kind='bar', ax=axes[0])
        display_col_freqtable(set1_df, col_name)
        axes[1].set_title('set2 {0}'.format(col_name))
        set2_df[col_name].value_counts().plot(kind='bar', ax=axes[1])
        display_col_freqtable(set2_df, col_name)
        axes[2].set_title('concat_df [set1, set2] {0}'.format(col_name))
        concat_df[col_name].value_counts().plot(kind='bar', ax=axes[2])
        display_col_freqtable(concat_df, col_name)
        fig.tight_layout()
        plt.show()


def display_col_categorical_sns_countplot(dataset_df, col_name):
    """ Display countplot with percentage+cnt for each categorical feature"""
    fig = plt.figure(figsize=(10, 5))
    ax = sns.countplot(x=col_name, data=dataset_df)
    ax2=ax.twinx()
    ax2.grid(None)
    ax2.get_yaxis().set_visible(False)
    ax2.get_xaxis().set_visible(False)
    for p in ax.patches:
        x = p.get_bbox().get_points()[:,0]
        y = p.get_bbox().get_points()[1,1]
        ax.annotate(
            '{} | {:.1f}%'.format(int(y), 100. * y / dataset_df[col_name].index.
 ↪size),
            (x.mean(), y),
            ha='center', va='bottom'
```

```
        )
    plt.title('Distribution of {0}'.format(col_name))
    plt.xlabel('Number of {0}'.format(col_name))
    plt.show()
```

[10]:
```
# display_all_categorical_freq_bar(train_df, test_df)

# print("HouseStyle: train_df")
# display_col_categorical_sns_countplot(train_df, 'HouseStyle')
```

[11]:
```
# Explore NaN values

def display_nan_values(dataset_df):
    """Display amount of NaN values in dataset_df columns"""
    dataset_df_nans = dataset_df.isnull().sum()
    display(dataset_df_nans[dataset_df_nans != 0])


def display_all_nan_percentage(dataset_df):
    missing_values_cnt = dataset_df.isnull().sum()
    missing_values_pct = missing_values_cnt * 100 / len(dataset_df)
    missing_values_pct_df = pd.DataFrame({'pct_nan': missing_values_pct,␣
 ↪'cnt_nan': missing_values_cnt})
    missing_values_pct_df = missing_values_pct_df.sort_values('pct_nan')
    missing_values_pct_df[missing_values_pct_df['pct_nan'] != 0].
 ↪plot(kind='bar')
    display(missing_values_pct_df[missing_values_pct_df['pct_nan'] != 0].T)
    plt.show()


def get_rows_with_nan(dataset_df, col_name, max_values=10):
    """Get rows with np.nan in col_name column values"""
    dataset_isnull_values = dataset_df.isnull()
    has_nan_rows = dataset_df.loc[
        dataset_isnull_values[dataset_isnull_values[col_name] == True].index, :
    ].head(max_values)
    return has_nan_rows
```

[12]:
```
# display_all_nan_percentage(train_df)

# display_all_nan_percentage(test_df)

# display_all_nan_percentage(concat_train_test_df)
```

[13]:
```
# Fix NaN values

# display_nan_values(train_df)

# display_nan_values(test_df)
```

[14]:
```
# Intermediate arrays

train_nonan_df = train_df.copy()
test_nonan_df = test_df.copy()
```

[15]:
```
# MSZoning

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'MSZoning'))
    display(get_rows_with_nan(test_nonan_df, 'MSZoning'))

    display_col_categorical_sns_countplot(train_nonan_df, 'MSZoning')
    display_col_categorical_sns_countplot(test_nonan_df, 'MSZoning')

    train_df_cpy = train_nonan_df.copy()
    lbl_encoder = LabelEncoder()
    train_df_cpy['MSZoning'] = lbl_encoder.
 ↪fit_transform(train_df_cpy['MSZoning'])
    display_colx_coly_scatter(train_df_cpy, 'GrLivArea', 'SalePrice',␣
 ↪color='MSZoning')

def _local_fix():
    # Fix - assume there are some "other" zoning.
    # todo: try replacing with .mode()
    # todo: features['MSZoning'] = features.groupby('MSSubClass')['MSZoning'].
 ↪transform(lambda x: x.fillna(x.mode()[0]))
    test_nonan_df['MSZoning'] = test_nonan_df['MSZoning'].fillna('Other')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'MSZoning'))
    display(get_rows_with_nan(test_nonan_df, 'MSZoning'))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[16]:
```
# LotFrontage

def _local_disp():
    display("train", get_rows_with_nan(train_nonan_df, 'LotFrontage', 3000).
 ↪shape)
    display("test", get_rows_with_nan(test_nonan_df, 'LotFrontage', 3000).shape)
```

```
    display("train", get_rows_with_nan(train_nonan_df, 'LotFrontage', 3000).
↪head(10))
    display("test", get_rows_with_nan(test_nonan_df, 'LotFrontage', 3000).
↪head(10))

    display_hist(train_nonan_df, 'LotFrontage', n_bins=100)
    display_hist(test_nonan_df, 'LotFrontage', n_bins=100)

    display_colx_coly_scatter(train_nonan_df, 'LotFrontage', 'LotArea')
    display_colx_coly_scatter(test_nonan_df, 'LotFrontage', 'LotArea')

    train_df_cpy = train_nonan_df.copy()
    lbl_encoder = LabelEncoder()

    train_df_cpy['MSZoning'] = lbl_encoder.
↪fit_transform(train_df_cpy['MSZoning'])
    display_colx_coly_scatter(train_df_cpy, 'LotFrontage', 'SalePrice',␣
↪color='MSZoning')
    display_colx_coly_scatter(train_df_cpy, 'LotArea', 'SalePrice',␣
↪color='MSZoning')

    train_df_cpy['Neighborhood'] = lbl_encoder.
↪fit_transform(train_df_cpy['Neighborhood'])
    display_colx_coly_scatter(train_df_cpy, 'LotFrontage', 'SalePrice',␣
↪color='Neighborhood')
    display_colx_coly_scatter(train_df_cpy, 'LotArea', 'SalePrice',␣
↪color='Neighborhood')

def _local_fix():
    # Fix - assume there might be houses without frontage at all.
    # todo: try replacing with .mean()
    # todo: replace by neighborhood / MSZoning
        # features['LotFrontage'] = features.
↪groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.fillna(x.
↪median()))
    train_nonan_df['LotFrontage'] = train_nonan_df['LotFrontage'].fillna(0.0)
    test_nonan_df['LotFrontage'] = test_nonan_df['LotFrontage'].fillna(0.0)

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'LotFrontage'))
    display(get_rows_with_nan(test_nonan_df, 'LotFrontage'))


# Explore
# _local_disp()
```

```
# Fix
_local_fix()

# Check
# _local_check()
```

[17]:
```
# Alley

def _local_disp():
    display("train", get_rows_with_nan(train_nonan_df, 'Alley', 3000).shape)
    display("test", get_rows_with_nan(test_nonan_df, 'Alley', 3000).shape)

    display(pd.unique(train_nonan_df['Alley']))

    display_col_categorical_sns_countplot(train_nonan_df, 'Alley')
    display_col_categorical_sns_countplot(test_nonan_df, 'Alley')

def _local_fix():
    # Fix - there are houses with no Alley access.
    # todo: try replacing with .mode() (by dataset, NOT by concatenated)

    train_nonan_df['Alley'] = train_nonan_df['Alley'].fillna('NoAccess')
    test_nonan_df['Alley'] = test_nonan_df['Alley'].fillna('NoAccess')

def _local_check():
    display("train", get_rows_with_nan(train_nonan_df, 'Alley', 3000).shape)
    display("test", get_rows_with_nan(test_nonan_df, 'Alley', 3000).shape)

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[18]:
```
# Utilities

def _local_disp():
    display("train", get_rows_with_nan(train_nonan_df, 'Utilities', 3000))
    display("test", get_rows_with_nan(test_nonan_df, 'Utilities', 3000))

    display("test", get_rows_with_nan(test_nonan_df, 'Utilities', 3000))

    display(pd.unique(train_nonan_df['Utilities']))
    display(pd.unique(test_nonan_df['Utilities']))
```

```
    display_col_categorical_sns_countplot(train_nonan_df, 'Utilities')
    display_col_categorical_sns_countplot(test_nonan_df, 'Utilities')

    display_colx_coly_scatter(train_nonan_df, 'Utilities', 'SalePrice')

def _local_fix():
    # Fix - there are houses with "Other" set of Utilities. "Other" might mean␣
 ↪there are no Utilities.
    # todo: try replacing with .mode()

    train_nonan_df['Utilities'] = train_nonan_df['Utilities'].fillna('Other')
    test_nonan_df['Utilities'] = test_nonan_df['Utilities'].fillna('Other')

def _local_check():
    display("train", get_rows_with_nan(train_nonan_df, 'Utilities', 3000))
    display("test", get_rows_with_nan(test_nonan_df, 'Utilities', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[19]:
```
# Exterior1st and Exterior2nd

def _local_disp():
    # missing the same row: Id=2152 in test set

    display("train", get_rows_with_nan(train_nonan_df, 'Exterior1st', 3000))
    display("test", get_rows_with_nan(test_nonan_df, 'Exterior1st', 3000))

    display("train", get_rows_with_nan(train_nonan_df, 'Exterior2nd', 3000))
    display("test", get_rows_with_nan(test_nonan_df, 'Exterior2nd', 3000))

    display(pd.unique(train_nonan_df['Exterior1st']))
    display(pd.unique(test_nonan_df['Exterior1st']))

    display(pd.unique(train_nonan_df['Exterior2nd']))
    display(pd.unique(test_nonan_df['Exterior2nd']))

    display_col_categorical_sns_countplot(train_nonan_df, 'Exterior1st')
    display_col_categorical_sns_countplot(test_nonan_df, 'Exterior1st')

    display_col_categorical_sns_countplot(train_nonan_df, 'Exterior2nd')
```

```
    display_col_categorical_sns_countplot(test_nonan_df, 'Exterior2nd')

    display_colx_coly_scatter(train_nonan_df, 'Exterior1st', 'SalePrice')
    display_colx_coly_scatter(train_nonan_df, 'Exterior2nd', 'SalePrice')

def _local_fix():
    # Fix - assume there might be no exterior at all.

    test_nonan_df['Exterior1st'] = test_nonan_df['Exterior1st'].
 ↪fillna('NoExterior')

    test_nonan_df['Exterior2nd'] = test_nonan_df['Exterior2nd'].
 ↪fillna('NoExterior')

def _local_check():
    display("train", get_rows_with_nan(train_nonan_df, 'Exterior1st', 3000))
    display("test", get_rows_with_nan(test_nonan_df, 'Exterior1st', 3000))

    display("train", get_rows_with_nan(train_nonan_df, 'Exterior2nd', 3000))
    display("test", get_rows_with_nan(test_nonan_df, 'Exterior2nd', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[20]:
```
# MasVnrType

def _local_disp():
    display("train", get_rows_with_nan(train_nonan_df, 'MasVnrType', 3000))
    display("test", get_rows_with_nan(test_nonan_df, 'MasVnrType', 3000))

    display(pd.unique(train_nonan_df['MasVnrType']))
    display(pd.unique(test_nonan_df['MasVnrType']))

    display_col_categorical_sns_countplot(train_nonan_df, 'MasVnrType')
    display_col_categorical_sns_countplot(test_nonan_df, 'MasVnrType')

    # NOTE: some points have MasVnrType==None BUT MasVnrArea != 0
   ␣
 ↪display_col_categorical_sns_countplot(train_nonan_df[train_nonan_df['MasVnrType']␣
 ↪== 'None'], 'MasVnrArea')
```

```
    train_df_cpy = train_nonan_df.copy()
    lbl_encoder = LabelEncoder()

    train_df_cpy['MasVnrType'] = lbl_encoder.
 ↪fit_transform(train_df_cpy['MasVnrType'].fillna('None'))
    display_colx_coly_scatter(
        train_df_cpy,
        'MasVnrArea',
        'SalePrice',
        color='MasVnrType'
    )

    train_df_cpy['FireplaceQu'] = lbl_encoder.
 ↪fit_transform(train_df_cpy['FireplaceQu'].fillna('NoQual'))
    display_colx_coly_scatter(
        train_df_cpy,
        'MasVnrArea',
        'SalePrice',
        color='FireplaceQu'
    )

def _local_fix():
    # Assume there might be walls with some "Other" masonry veneer type.

    train_nonan_df['MasVnrType'] = train_nonan_df['MasVnrType'].
 ↪fillna('OtherMasVnr')

    test_nonan_df['MasVnrType'] = test_nonan_df['MasVnrType'].
 ↪fillna('OtherMasVnr')

def _local_check():
    display("train", get_rows_with_nan(train_nonan_df, 'MasVnrType', 3000))
    display("test", get_rows_with_nan(test_nonan_df, 'MasVnrType', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[21]:
```
# MasVnrArea

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'MasVnrArea', 3000))
```

```
    display(get_rows_with_nan(test_nonan_df, 'MasVnrArea', 3000))

    display_hist(train_nonan_df, 'MasVnrArea', n_bins=100)
    display_hist(test_nonan_df, 'MasVnrArea', n_bins=100)

    display_hist(train_nonan_df[train_nonan_df['MasVnrArea'] != 0],␣
 ↪'MasVnrArea', n_bins=100)
    display_hist(test_nonan_df[test_nonan_df['MasVnrArea'] != 0], 'MasVnrArea',␣
 ↪n_bins=100)

    train_df_cpy = train_nonan_df.copy()
    lbl_encoder = LabelEncoder()
    train_df_cpy['MasVnrType'] = lbl_encoder.
 ↪fit_transform(train_df_cpy['MasVnrType'])
    display_colx_coly_scatter(train_df_cpy, 'MasVnrArea', 'SalePrice',␣
 ↪color='MasVnrType')

def _local_fix():
    # Assume there is no masonvry veneer, so area equals to 0.0
    train_nonan_df['MasVnrArea'] = train_nonan_df['MasVnrArea'].fillna(0.0)
    test_nonan_df['MasVnrArea'] = test_nonan_df['MasVnrArea'].fillna(0.0)

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'MasVnrArea', 3000))
    display(get_rows_with_nan(test_nonan_df, 'MasVnrArea', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[22]:
```
# BsmtQual

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'BsmtQual', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtQual', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtQual', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtQual', 3000).shape)

    display(train_nonan_df['BsmtQual'].unique())
    display(test_nonan_df['BsmtQual'].unique())
```

```
    display_col_categorical_sns_countplot(train_nonan_df, 'BsmtQual')
    display_col_categorical_sns_countplot(test_nonan_df, 'BsmtQual')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'BsmtQual',␣
 ↪'SalePrice')

def _local_fix():
    # Assume there is no basement in the house

    train_nonan_df['BsmtQual'] = train_nonan_df['BsmtQual'].fillna('NoBsmt')

    test_nonan_df['BsmtQual'] = test_nonan_df['BsmtQual'].fillna('NoBsmt')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'BsmtQual', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtQual', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[23]:
```
# BsmtCond

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'BsmtCond', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtCond', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtCond', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtCond', 3000).shape)

    display(train_nonan_df['BsmtCond'].unique())
    display(test_nonan_df['BsmtCond'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'BsmtCond')
    display_col_categorical_sns_countplot(test_nonan_df, 'BsmtCond')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'BsmtCond',␣
 ↪'SalePrice')

def _local_fix():
    # Assume there is no basement in the house
```

```
    train_nonan_df['BsmtCond'] = train_nonan_df['BsmtCond'].fillna('NoBsmt')

    test_nonan_df['BsmtCond'] = test_nonan_df['BsmtCond'].fillna('NoBsmt')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'BsmtCond', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtCond', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[24]:
```
# BsmtExposure

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'BsmtExposure', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtExposure', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtExposure', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtExposure', 3000).shape)

    display(train_nonan_df['BsmtExposure'].unique())
    display(test_nonan_df['BsmtExposure'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'BsmtExposure')
    display_col_categorical_sns_countplot(test_nonan_df, 'BsmtExposure')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'BsmtExposure',␣
 ↪'SalePrice')

def _local_fix():
    # Assume there is no basement at all

    train_nonan_df['BsmtExposure'] = train_nonan_df['BsmtExposure'].
 ↪fillna('NoBsmt')

    test_nonan_df['BsmtExposure'] = test_nonan_df['BsmtExposure'].
 ↪fillna('NoBsmt')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'BsmtExposure', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtExposure', 3000))
```

```
# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[25]:
```
# BsmtFinType1 and BsmtFinType2

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'BsmtFinType1', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinType1', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtFinType1', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinType1', 3000).shape)

    display(train_nonan_df['BsmtFinType1'].unique())
    display(test_nonan_df['BsmtFinType1'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'BsmtFinType1')
    display_col_categorical_sns_countplot(test_nonan_df, 'BsmtFinType1')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'BsmtFinType1',␣
 ↪'SalePrice')

    display(get_rows_with_nan(train_nonan_df, 'BsmtFinType2', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinType2', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtFinType2', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinType2', 3000).shape)

    display(train_nonan_df['BsmtFinType2'].unique())
    display(test_nonan_df['BsmtFinType2'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'BsmtFinType2')
    display_col_categorical_sns_countplot(test_nonan_df, 'BsmtFinType2')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'BsmtFinType2',␣
 ↪'SalePrice')

def _local_fix():
    # Assume there is no basement at all
```

```
    train_nonan_df['BsmtFinType1'] = train_nonan_df['BsmtFinType1'].
 ↪fillna('NoBsmt')
    test_nonan_df['BsmtFinType1'] = test_nonan_df['BsmtFinType1'].
 ↪fillna('NoBsmt')

    train_nonan_df['BsmtFinType2'] = train_nonan_df['BsmtFinType2'].
 ↪fillna('NoBsmt')
    test_nonan_df['BsmtFinType2'] = test_nonan_df['BsmtFinType2'].
 ↪fillna('NoBsmt')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'BsmtFinType1', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinType1', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtFinType2', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinType2', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[26]:
```
# BsmtFinSF1 and BsmtFinSF2

# same idx: id=2121

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'BsmtFinSF1', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinSF1', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtFinSF1', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinSF1', 3000).shape)

    train_df_cpy = train_nonan_df.copy()
    lbl_encoder = LabelEncoder()
    train_df_cpy['BsmtFinType1'] = lbl_encoder.
 ↪fit_transform(train_df_cpy['BsmtFinType1'])
    display_colx_coly_scatter(train_df_cpy, 'BsmtFinSF1', 'SalePrice',␣
 ↪color='BsmtFinType1')

    display(get_rows_with_nan(train_nonan_df, 'BsmtFinSF2', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinSF2', 3000))
```

```
    display(get_rows_with_nan(train_nonan_df, 'BsmtFinSF2', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinSF2', 3000).shape)

    train_df_cpy['BsmtFinType2'] = lbl_encoder.
 ↪fit_transform(train_df_cpy['BsmtFinType2'])
    display_colx_coly_scatter(train_df_cpy, 'BsmtFinSF2', 'SalePrice',␣
 ↪color='BsmtFinType2')

def _local_fix():
    # Assume there is no basement at all

    test_nonan_df['BsmtFinSF1'] = test_nonan_df['BsmtFinSF1'].fillna(0.0)

    test_nonan_df['BsmtFinSF2'] = test_nonan_df['BsmtFinSF2'].fillna(0.0)

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'BsmtFinSF1', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinSF1', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtFinSF2', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFinSF2', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[27]:
```
# BsmtUnfSF and TotalBsmtSF

# same idx: id=2121

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'BsmtUnfSF', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtUnfSF', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtUnfSF', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtUnfSF', 3000).shape)

    train_df_cpy = train_nonan_df.copy()
    lbl_encoder = LabelEncoder()
    train_df_cpy['BsmtCond'] = lbl_encoder.
 ↪fit_transform(train_df_cpy['BsmtCond'])
```

```
    display_colx_coly_scatter(train_df_cpy, 'BsmtUnfSF', 'SalePrice',␣
 ↪color='BsmtCond')

    display(get_rows_with_nan(train_nonan_df, 'TotalBsmtSF', 3000))
    display(get_rows_with_nan(test_nonan_df, 'TotalBsmtSF', 3000))

    display(get_rows_with_nan(train_nonan_df, 'TotalBsmtSF', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'TotalBsmtSF', 3000).shape)

    display_colx_coly_scatter(train_df_cpy, 'TotalBsmtSF', 'SalePrice',␣
 ↪color='BsmtCond')

def _local_fix():
    # Assume there is no basement at all

    test_nonan_df['BsmtUnfSF'] = test_nonan_df['BsmtUnfSF'].fillna(0.0)

    test_nonan_df['TotalBsmtSF'] = test_nonan_df['TotalBsmtSF'].fillna(0.0)

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'BsmtUnfSF', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtUnfSF', 3000))

    display(get_rows_with_nan(train_nonan_df, 'TotalBsmtSF', 3000))
    display(get_rows_with_nan(test_nonan_df, 'TotalBsmtSF', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[28]:
```
# BsmtFullBath and BsmtHalfBath

# same indices: id=2121 and id=2189

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'BsmtFullBath', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFullBath', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtFullBath', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtFullBath', 3000).shape)

    display_col_categorical_sns_countplot(train_nonan_df, 'BsmtFullBath')
```

```
    display_col_categorical_sns_countplot(test_nonan_df, 'BsmtFullBath')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'BsmtFullBath',␣
 ↪'SalePrice')

    display(get_rows_with_nan(train_nonan_df, 'BsmtHalfBath', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtHalfBath', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtHalfBath', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'BsmtHalfBath', 3000).shape)

    display_col_categorical_sns_countplot(train_nonan_df, 'BsmtHalfBath')
    display_col_categorical_sns_countplot(test_nonan_df, 'BsmtHalfBath')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'BsmtHalfBath',␣
 ↪'SalePrice')

def _local_fix():
    # Assume there is no basement at all => there couldn't be any bath in the␣
 ↪basement

    test_nonan_df['BsmtFullBath'] = test_nonan_df['BsmtFullBath'].fillna(0)

    test_nonan_df['BsmtHalfBath'] = test_nonan_df['BsmtHalfBath'].fillna(0)

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'BsmtFullBath', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtFullBath', 3000))

    display(get_rows_with_nan(train_nonan_df, 'BsmtHalfBath', 3000))
    display(get_rows_with_nan(test_nonan_df, 'BsmtHalfBath', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[29]:
```
# PoolQC

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'PoolQC', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'PoolQC', 3000).shape)
```

```
    display(train_nonan_df['PoolQC'].unique())
    display(test_nonan_df['PoolQC'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'PoolQC')
    display_col_categorical_sns_countplot(test_nonan_df, 'PoolQC')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'PoolQC',␣
 ↪'SalePrice')

def _local_fix():
    # Assume single missing row belongs to "0th" class, which already exists in␣
 ↪train and test sets

    train_nonan_df['PoolQC'] = train_nonan_df['PoolQC'].fillna('NoPool')

    test_nonan_df['PoolQC'] = test_nonan_df['PoolQC'].fillna('NoPool')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'PoolQC', 3000))
    display(get_rows_with_nan(test_nonan_df, 'PoolQC', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[30]:
```
# Fence

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'Fence', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'Fence', 3000).shape)

    display(train_nonan_df['Fence'].unique())
    display(test_nonan_df['Fence'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'Fence')
    display_col_categorical_sns_countplot(test_nonan_df, 'Fence')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'Fence',␣
 ↪'SalePrice')

    train_df_cpy = train_nonan_df.copy()
    lbl_encoder = LabelEncoder()
```

```
    train_df_cpy['Fence'] = lbl_encoder.fit_transform(train_df_cpy['Fence'])
    display_colx_coly_scatter(train_df_cpy, 'LotArea', 'SalePrice',␣
 ↪color='Fence')

def _local_fix():
    # Assume single missing row belongs to "Oth" class, which already exists in␣
 ↪train and test sets

    train_nonan_df['Fence'] = train_nonan_df['Fence'].fillna('NoFence')

    test_nonan_df['Fence'] = test_nonan_df['Fence'].fillna('NoFence')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'Fence', 3000))
    display(get_rows_with_nan(test_nonan_df, 'Fence', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[31]:
```
# MiscFeature

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'MiscFeature', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'MiscFeature', 3000).shape)

    display(train_nonan_df['MiscFeature'].unique())
    display(test_nonan_df['MiscFeature'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'MiscFeature')
    display_col_categorical_sns_countplot(test_nonan_df, 'MiscFeature')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'MiscFeature',␣
 ↪'SalePrice')

    display_colx_coly_scatter(
        train_nonan_df[train_nonan_df['MiscFeature'] == 'Shed'], 'MiscFeature',␣
 ↪'SalePrice'
    )

def _local_fix():
```

```
    # Assume single missing row belongs to "Oth" class, which already exists in␣
 ↪train and test sets

    train_nonan_df['MiscFeature'] = train_nonan_df['MiscFeature'].fillna('None')

    test_nonan_df['MiscFeature'] = test_nonan_df['MiscFeature'].fillna('None')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'MiscFeature', 3000))
    display(get_rows_with_nan(test_nonan_df, 'MiscFeature', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[32]:
```
# SaleType

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'SaleType', 3000))
    display(get_rows_with_nan(test_nonan_df, 'SaleType', 3000))

    display(train_nonan_df['SaleType'].unique())
    display(test_nonan_df['SaleType'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'SaleType')
    display_col_categorical_sns_countplot(test_nonan_df, 'SaleType')

    train_df_cpy = train_nonan_df.copy()
    lbl_encoder = LabelEncoder()
    train_df_cpy['SaleCondition'] = lbl_encoder.
 ↪fit_transform(train_df_cpy['SaleCondition'])
    display_colx_coly_scatter(train_df_cpy, 'SaleType', 'SalePrice',␣
 ↪color='SaleCondition')

def _local_fix():
    # Assume single missing row belongs to "Oth" class, which already exists in␣
 ↪train and test sets

    train_nonan_df['SaleType'] = train_nonan_df['SaleType'].fillna('Oth')

    test_nonan_df['SaleType'] = test_nonan_df['SaleType'].fillna('Oth')
```

```
def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'SaleType', 3000))
    display(get_rows_with_nan(test_nonan_df, 'SaleType', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[33]:
```
# FireplaceQu

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'FireplaceQu', 3000).head(3))
    display(get_rows_with_nan(test_nonan_df, 'FireplaceQu', 3000).head(3))

    display(get_rows_with_nan(train_nonan_df, 'FireplaceQu', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'FireplaceQu', 3000).shape)

    display(train_nonan_df['FireplaceQu'].unique())
    display(test_nonan_df['FireplaceQu'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'FireplaceQu')
    display_col_categorical_sns_countplot(test_nonan_df, 'FireplaceQu')

    train_fireplaces_nan = get_rows_with_nan(train_nonan_df, 'FireplaceQu',␣
↪3000)
    display(train_fireplaces_nan[train_fireplaces_nan['Fireplaces'] != 0])  #␣
↪empty df

    test_fireplaces_nan = get_rows_with_nan(test_nonan_df, 'FireplaceQu', 3000)
    display(test_fireplaces_nan[test_fireplaces_nan['Fireplaces'] != 0])  #␣
↪empty df

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'SaleType',␣
↪'SalePrice')

def _local_fix():
    # Assume there is no fireplace at all (because "Fireplaces" = 0)

    train_nonan_df['FireplaceQu'] = train_nonan_df['FireplaceQu'].fillna('None')

    test_nonan_df['FireplaceQu'] = test_nonan_df['FireplaceQu'].fillna('None')
```

```
def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'FireplaceQu', 3000))
    display(get_rows_with_nan(test_nonan_df, 'FireplaceQu', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[34]:
```
# Other features with NaN values: Electrical, KitchenQual, Functional

# Electrical: train_nonan_df, row id=1380
# KitchenQual: test_nonan_df, row_id=1556
# Functional: test_nona_df, row_indices=2217,2474.

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'Electrical', 3000))
    display(get_rows_with_nan(test_nonan_df, 'Electrical', 3000))

    display(get_rows_with_nan(train_nonan_df, 'KitchenQual', 3000))
    display(get_rows_with_nan(test_nonan_df, 'KitchenQual', 3000))

    display(get_rows_with_nan(train_nonan_df, 'Functional', 3000))
    display(get_rows_with_nan(test_nonan_df, 'Functional', 3000))

def _local_fix():
    # Electrical: replace with the most common value
    train_nonan_df['Electrical'] = train_nonan_df['Electrical'].fillna(
        train_nonan_df['Electrical'].mode()[0]
    )
    # KitchenQual: replace with the most common value
    test_nonan_df['KitchenQual'] = test_nonan_df['KitchenQual'].fillna(
        test_nonan_df['KitchenQual'].mode()[0]
    )
    # Functional: from docs: "Assume typical unless deductions are warranted"
    test_nonan_df['Functional'] = test_nonan_df['Functional'].fillna('Typ')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'Electrical', 3000))
    display(get_rows_with_nan(test_nonan_df, 'Electrical', 3000))

    display(get_rows_with_nan(train_nonan_df, 'KitchenQual', 3000))
    display(get_rows_with_nan(test_nonan_df, 'KitchenQual', 3000))
```

```
    display(get_rows_with_nan(train_nonan_df, 'Functional', 3000))
    display(get_rows_with_nan(test_nonan_df, 'Functional', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[35]:
```
# GarageType

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'GarageType', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageType', 3000))

    display(get_rows_with_nan(train_nonan_df, 'GarageType', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'GarageType', 3000).shape)

    display(train_nonan_df['GarageType'].unique())
    display(test_nonan_df['GarageType'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'GarageType')
    display_col_categorical_sns_countplot(test_nonan_df, 'GarageType')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'GarageType',␣
 ↪'SalePrice')

def _local_fix():
    # Assume there is no garage (because all GarageArea==0.0)

    train_nonan_df['GarageType'] = train_nonan_df['GarageType'].
 ↪fillna('NoGarage')

    test_nonan_df['GarageType'] = test_nonan_df['GarageType'].fillna('NoGarage')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'GarageType', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageType', 3000))

# Explore
# _local_disp()

# Fix
```

```
_local_fix()

# Check
# _local_check()
```

[36]:
```
# GarageYrBlt

def _local_disp():
#     display(get_rows_with_nan(train_nonan_df, 'GarageYrBlt', 3000))
#     display(get_rows_with_nan(test_nonan_df, 'GarageYrBlt', 3000))

    display(get_rows_with_nan(train_nonan_df, 'GarageYrBlt', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'GarageYrBlt', 3000).shape)

    train_nan_garageyrblt = get_rows_with_nan(train_nonan_df, 'GarageYrBlt',␣
↪3000)
    display(train_nan_garageyrblt[train_nan_garageyrblt['GarageType'] ==␣
↪'Detchd'])  # empty

    test_nan_garageyrblt = get_rows_with_nan(test_nonan_df, 'GarageYrBlt', 3000)
    display(test_nan_garageyrblt[test_nan_garageyrblt['GarageType'] ==␣
↪'Detchd'])  # indices=[2127,2577]

    display(train_nonan_df['GarageYrBlt'].min(), train_nonan_df['GarageYrBlt'].
↪max())
    display(test_nonan_df['GarageYrBlt'].min(), test_nonan_df['GarageYrBlt'].
↪max())

    display_hist(train_nonan_df, 'GarageYrBlt', n_bins=100)
    display_hist(test_nonan_df, 'GarageYrBlt', n_bins=100)

    display_colx_coly_scatter(train_nonan_df, 'GarageYrBlt', 'SalePrice')

def _local_fix():
    # For indices=[2127,2577]: because they are detached -> replace by median␣
↪value
    dtchd_garage_yrblt_median = test_nonan_df.groupby('GarageType').
↪get_group('Detchd')['GarageYrBlt'].median()
    test_nonan_df.loc[666, 'GarageYrBlt'] = dtchd_garage_yrblt_median
    test_nonan_df.loc[1116, 'GarageYrBlt'] = dtchd_garage_yrblt_median

    # Assume all other rows with NaN in GarageYrBlt mean that there is no␣
↪garage at all
    # Because GarageYrBlt is a numerical feature, replace it with a really␣
↪early year - 1500 - "Magic year".
    # This feature will be "cut" later so no worry for such an inadequate value.
```

```
    train_nonan_df['GarageYrBlt'] = train_nonan_df['GarageYrBlt'].fillna(1500)

    test_nonan_df['GarageYrBlt'] = test_nonan_df['GarageYrBlt'].fillna(1500)

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'GarageYrBlt', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageYrBlt', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[37]:
```
# GarageFinish

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'GarageFinish', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageFinish', 3000))

    display(get_rows_with_nan(train_nonan_df, 'GarageFinish', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'GarageFinish', 3000).shape)

    display(train_nonan_df['GarageFinish'].unique())
    display(test_nonan_df['GarageFinish'].unique())

    train_nan_garageyrblt = get_rows_with_nan(train_nonan_df, 'GarageFinish',␣
 ↪3000)
    display(train_nan_garageyrblt[train_nan_garageyrblt['GarageType'] ==␣
 ↪'Detchd'])  # empty

    test_nan_garageyrblt = get_rows_with_nan(test_nonan_df, 'GarageFinish',␣
 ↪3000)
    display(test_nan_garageyrblt[test_nan_garageyrblt['GarageType'] ==␣
 ↪'Detchd'])  # indices=[2127,2577]

    display_col_categorical_sns_countplot(train_nonan_df, 'GarageFinish')
    display_col_categorical_sns_countplot(test_nonan_df, 'GarageFinish')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'GarageFinish',␣
 ↪'SalePrice')

def _local_fix():
```

```
    # For indices=[2127,2577]: because they are detached -> replace by median␣
↪value
    dtchd_garage_garagefinish_mode = test_nonan_df.groupby('GarageType'
                                                           ).
↪get_group('Detchd')['GarageFinish'].mode()[0]
    test_nonan_df.loc[666, 'GarageFinish'] = dtchd_garage_garagefinish_mode
    test_nonan_df.loc[1116, 'GarageFinish'] = dtchd_garage_garagefinish_mode

    # Assume all other rows with NaN in GarageFinish mean that there is no␣
↪garage at all
    # Another reason: in these rows GarageArea == 0 => there is no garage at␣
↪all

    train_nonan_df['GarageFinish'] = train_nonan_df['GarageFinish'].
↪fillna('NoGarage')

    test_nonan_df['GarageFinish'] = test_nonan_df['GarageFinish'].
↪fillna('NoGarage')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'GarageFinish', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageFinish', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[38]:
```
# GarageCars and GarageArea

# 1 row: id=2577

def _local_disp():
    display(get_rows_with_nan(train_nonan_df, 'GarageCars', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageCars', 3000))

    display(get_rows_with_nan(train_nonan_df, 'GarageCars', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'GarageCars', 3000).shape)

    display(train_nonan_df['GarageCars'].unique())
    display(test_nonan_df['GarageCars'].unique())

    display_col_categorical_sns_countplot(train_nonan_df, 'GarageCars')
```

```
    display_col_categorical_sns_countplot(test_nonan_df, 'GarageCars')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'GarageCars',␣
 ↪'SalePrice')

    display(get_rows_with_nan(train_nonan_df, 'GarageArea', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageArea', 3000))

    display(get_rows_with_nan(train_nonan_df, 'GarageArea', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'GarageArea', 3000).shape)

    display_hist(train_nonan_df, 'GarageArea', n_bins=100)
    display_hist(test_nonan_df, 'GarageArea', n_bins=100)

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'GarageArea',␣
 ↪'SalePrice', color='GarageCars')

def _local_fix():
    # GarageCars
    # For idx=[2577]: because garagetype is detached -> replace by mode value
    test_nonan_df.loc[1116, 'GarageCars'
                      ] = test_nonan_df.groupby('GarageType').
 ↪get_group('Detchd')['GarageCars'].median()

    # GarageArea
    # For idx=[2577]: because garagetype is detached -> replace by mode value
    test_nonan_df.loc[1116, 'GarageArea'
                      ] = test_nonan_df.groupby('GarageType').
 ↪get_group('Detchd')['GarageArea'].mean()

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'GarageCars', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageCars', 3000))

    display(get_rows_with_nan(train_nonan_df, 'GarageArea', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageArea', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[39]:
```
# GarageQual

def _local_disp():
#     display(get_rows_with_nan(train_nonan_df, 'GarageQual', 3000))
#     display(get_rows_with_nan(test_nonan_df, 'GarageQual', 3000))

    display(get_rows_with_nan(train_nonan_df, 'GarageQual', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'GarageQual', 3000).shape)

    display(train_nonan_df['GarageQual'].unique())
    display(test_nonan_df['GarageQual'].unique())

    train_nan_garageyrblt = get_rows_with_nan(train_nonan_df, 'GarageQual',␣
↪3000)
    display(train_nan_garageyrblt[train_nan_garageyrblt['GarageType'] ==␣
↪'Detchd'])  # empty

    test_nan_garageyrblt = get_rows_with_nan(test_nonan_df, 'GarageQual', 3000)
    display(test_nan_garageyrblt[test_nan_garageyrblt['GarageType'] ==␣
↪'Detchd'])  # indices=[2127,2577]

    display_col_categorical_sns_countplot(train_nonan_df, 'GarageQual')
    display_col_categorical_sns_countplot(test_nonan_df, 'GarageQual')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'GarageQual',␣
↪'SalePrice')

def _local_fix():
    # For indices=[2127,2577]: because they are detached -> replace by median␣
↪value
    dtchd_garage_garagefinish_mode = test_nonan_df.groupby('GarageType'
                                                           ).
↪get_group('Detchd')['GarageQual'].mode()[0]
    test_nonan_df.loc[666, 'GarageQual'] = dtchd_garage_garagefinish_mode
    test_nonan_df.loc[1116, 'GarageQual'] = dtchd_garage_garagefinish_mode

    # For every other garagetype=np.nan: assume there is no garage at all

    train_nonan_df['GarageQual'] = train_nonan_df['GarageQual'].
↪fillna('NoGarage')

    test_nonan_df['GarageQual'] = test_nonan_df['GarageQual'].fillna('NoGarage')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'GarageQual', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageQual', 3000))
```

```
# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[40]:
```
# GarageCond

def _local_disp():
#     display(get_rows_with_nan(train_nonan_df, 'GarageQual', 3000))
#     display(get_rows_with_nan(test_nonan_df, 'GarageQual', 3000))

    display(get_rows_with_nan(train_nonan_df, 'GarageCond', 3000).shape)
    display(get_rows_with_nan(test_nonan_df, 'GarageCond', 3000).shape)

    display(train_nonan_df['GarageCond'].unique())
    display(test_nonan_df['GarageCond'].unique())

    train_nan_garageyrblt = get_rows_with_nan(train_nonan_df, 'GarageCond',␣
 ↪3000)
    display(train_nan_garageyrblt[train_nan_garageyrblt['GarageType'] ==␣
 ↪'Detchd'])  # empty

    test_nan_garageyrblt = get_rows_with_nan(test_nonan_df, 'GarageCond', 3000)
    display(test_nan_garageyrblt[test_nan_garageyrblt['GarageType'] ==␣
 ↪'Detchd'])  # indices=[2127,2577]

    display_col_categorical_sns_countplot(train_nonan_df, 'GarageCond')
    display_col_categorical_sns_countplot(test_nonan_df, 'GarageCond')

    display_colx_coly_scatter(train_nonan_df.fillna('dbg'), 'GarageCond',␣
 ↪'SalePrice')

def _local_fix():
    # For indices=[2127,2577]: because they are detached -> replace by median␣
 ↪value
    dtchd_garage_garagefinish_mode = test_nonan_df.groupby('GarageType'
                                                           ).
 ↪get_group('Detchd')['GarageCond'].mode()[0]
    test_nonan_df.loc[666, 'GarageCond'] = dtchd_garage_garagefinish_mode
    test_nonan_df.loc[1116, 'GarageCond'] = dtchd_garage_garagefinish_mode

    # For every other GarageCond=np.nan: assume there is no garage at all
```

```
    train_nonan_df['GarageCond'] = train_nonan_df['GarageCond'].
 ↪fillna('NoGarage')

    test_nonan_df['GarageCond'] = test_nonan_df['GarageCond'].fillna('NoGarage')

def _local_check():
    display(get_rows_with_nan(train_nonan_df, 'GarageCond', 3000))
    display(get_rows_with_nan(test_nonan_df, 'GarageCond', 3000))

# Explore
# _local_disp()

# Fix
_local_fix()

# Check
# _local_check()
```

[41]:
```
# Check again fixed NaN values

display_nan_values(train_nonan_df)

display_nan_values(test_nonan_df)
```

```
Series([], dtype: int64)
```

```
Series([], dtype: int64)
```

[42]:
```
# Fix incorrect values

# Fix year=2207 - obvious mistake in dataset
# display(
#     test_nonan_df[test_nonan_df['GarageYrBlt'] > 2010]
# )
test_nonan_df.loc[1132, 'GarageYrBlt'] = 2007  # same as remodelling date
```

[43]:
```
# Remove data with very low distribution

categorical_col_names = train_nonan_df.select_dtypes(include='object').columns.
 ↪values

features_value_counts = pd.DataFrame({
    'train': train_nonan_df[categorical_col_names].apply(lambda x: len(np.
 ↪unique(x))),
```

```
    'test': test_nonan_df[categorical_col_names].apply(lambda x: len(np.
 ↪unique(x))),
}).sort_values(by="train")

# display(features_value_counts)

# display(train_nonan_df['Street'].value_counts(), test_nonan_df['Street'].
 ↪value_counts())
# display(train_nonan_df['CentralAir'].value_counts(),␣
 ↪test_nonan_df['CentralAir'].value_counts())
# display(train_nonan_df['Utilities'].value_counts(),␣
 ↪test_nonan_df['Utilities'].value_counts())
# display(train_nonan_df['Alley'].value_counts(), test_nonan_df['Alley'].
 ↪value_counts())
# display(train_nonan_df['LandSlope'].value_counts(),␣
 ↪test_nonan_df['LandSlope'].value_counts())
# display(train_nonan_df['PavedDrive'].value_counts(),␣
 ↪test_nonan_df['PavedDrive'].value_counts())

train_nonan_df = train_nonan_df.drop(['Utilities', 'Street', 'MiscVal'], axis=1)
test_nonan_df = test_nonan_df.drop(['Utilities', 'Street', 'MiscVal'], axis=1)
```

[44]:
```
# Functions to create new features

# src: https://www.kaggle.com/laurenstc/top-2-of-leaderboard-advanced-fe

def add_new_features_inplace(dataset_df):
    dataset_df['Total_sqr_footage'] = dataset_df['BsmtFinSF1'] +␣
 ↪dataset_df['BsmtFinSF2'] + \
                                      dataset_df['1stFlrSF'] +␣
 ↪dataset_df['2ndFlrSF']

    dataset_df['Total_Bathrooms'] = dataset_df['FullBath'] + (0.5 *␣
 ↪dataset_df['HalfBath']) + \
                                      dataset_df['BsmtFullBath'] + (0.5 *␣
 ↪dataset_df['BsmtHalfBath'])

    dataset_df['Total_porch_sf'] = dataset_df['OpenPorchSF'] + \
                                   dataset_df['3SsnPorch'] + \
                                   dataset_df['EnclosedPorch'] + \
                                   dataset_df['ScreenPorch'] + \
                                   dataset_df['WoodDeckSF']

    dataset_df['HasPool'] = dataset_df['PoolArea'].apply(lambda x: 1 if x > 0␣
 ↪else 0)
```

```
    dataset_df['Has2ndFloor'] = dataset_df['2ndFlrSF'].apply(lambda x: 1 if x >␣
↪0 else 0)
    dataset_df['HasGarage'] = dataset_df['GarageArea'].apply(lambda x: 1 if x >␣
↪0 else 0)
    dataset_df['HasBsmt'] = dataset_df['TotalBsmtSF'].apply(lambda x: 1 if x >␣
↪0 else 0)
    dataset_df['HasFirePlace'] = dataset_df['Fireplaces'].apply(lambda x: 1 if␣
↪x > 0 else 0)
```

[45]:
```
# Intermediate arrays

train_newfeatures_df = train_nonan_df.copy()
test_newfeatures_df = test_nonan_df.copy()
```

[46]:
```
add_new_features_inplace(train_newfeatures_df)
add_new_features_inplace(test_newfeatures_df)
```

[47]:
```
# Functions to fix column dtypes

# Divide continuous data into n_bins bins.
def continuous_to_bins_inplace(dataset_df, col_name, n_bins):
    qcut_bins = pd.qcut(dataset_df[col_name], n_bins, retbins=True)[1]
    qcut_bins[0] = int(qcut_bins[0]) - 1
    qcut_bins[-1] = int(qcut_bins[-1]) + 2
    column_copy = dataset_df[col_name].copy()
    for idx in range(len(qcut_bins) - 1):
        cur_range_start = qcut_bins[idx]
        cur_range_end = qcut_bins[idx + 1]
        after_start_mask = column_copy >= cur_range_start
        before_end_mask = column_copy < cur_range_end
        dataset_df.loc[
            after_start_mask & before_end_mask, col_name
        ] = "{0}_{1}".format(cur_range_start, cur_range_end)
```

[48]:
```
# Intermediate arrays

concat_fixdtypes_df = pd.concat(
    [train_newfeatures_df.copy(), test_newfeatures_df.copy()],
    ignore_index=True, sort=False
)
```

[49]:
```
# display_dataset_col_dtypes(concat_fixdtypes_df)
```

[50]:
```
# Fix features types

# Fix several features from numerical to categorical dtype
num2cat_col_names = [
    'MSSubClass',
    'MoSold', 'YrSold'
```

```
]
concat_fixdtypes_df[num2cat_col_names] = concat_fixdtypes_df[num2cat_col_names].
 ↪astype(str)

# Cut several features into different chunks

continuous_to_bins_inplace(concat_fixdtypes_df, 'YearRemodAdd', 4)

continuous_to_bins_inplace(concat_fixdtypes_df, 'YearBuilt', 4)

def _ugly_fix_garageyrblt_categories_inplace(dataset_df):
    garageyrblt_cpy = dataset_df['GarageYrBlt']
    dataset_df.loc[(garageyrblt_cpy == 1500), 'GarageYrBlt'] = "0"
    dataset_df.loc[(garageyrblt_cpy >= 1895.0 - 1) & (garageyrblt_cpy < 1960.
 ↪0), 'GarageYrBlt'] = "1"
    dataset_df.loc[(garageyrblt_cpy >= 1960.0) & (garageyrblt_cpy < 1979.0),␣
 ↪'GarageYrBlt'] = "2"
    dataset_df.loc[(garageyrblt_cpy >= 1979.0) & (garageyrblt_cpy < 2002.0),␣
 ↪'GarageYrBlt'] = "3"
    dataset_df.loc[(garageyrblt_cpy >= 2002.0) & (garageyrblt_cpy < 2010.0 +␣
 ↪2), 'GarageYrBlt'] = "4"

_ugly_fix_garageyrblt_categories_inplace(concat_fixdtypes_df)
```

[51]:
```
# display_dataset_col_dtypes(concat_fixdtypes_df)
```

[52]:
```
# Functions to fix skewness of continuous features values: apply␣
 ↪log1p-transform for normality

def display_df_numerical_before_after_log(dataset_df, col_name):
    fig, [[ax_0, ax_1, ax_2],
          [ax_3, ax_4, ax_5],
          [ax_6, ax_7, ax_8],
          [ax_9, ax_10, ax_11]] = plt.subplots(4, 3, figsize=(20, 15))
    # 1-3: distribution plots
    sns.distplot(dataset_df[col_name], ax=ax_0)
    sns.distplot(np.log1p(dataset_df[col_name]), ax=ax_1)
    sns.distplot(np.sqrt(dataset_df[col_name]), ax=ax_2)
    # 4-6: probability plots
    scstats.probplot(dataset_df[col_name], plot=ax_3)
    scstats.probplot(np.log1p(dataset_df[col_name]), plot=ax_4)
    scstats.probplot(np.sqrt(dataset_df[col_name]), plot=ax_5)
    # 7-9 non-zero values
    nonzeros_vals = dataset_df[dataset_df[col_name] != 0.0]
    sns.distplot(nonzeros_vals[col_name], ax=ax_6)
    sns.distplot(np.log1p(nonzeros_vals[col_name]), ax=ax_7)
    sns.distplot(np.sqrt(nonzeros_vals[col_name]), ax=ax_8)
```

```
    # 10 boxcox1p for non-zero values
    sns.distplot(
        boxcox1p(nonzeros_vals[col_name], scstats.
↪boxcox_normmax(nonzeros_vals[col_name])),
        ax=ax_9
    )
    # 11 probplot for boxcox1p
    scstats.probplot(
        boxcox1p(nonzeros_vals[col_name], scstats.
↪boxcox_normmax(nonzeros_vals[col_name])),
        plot=ax_10
    )
    # 12 probplot for sqrt for nonzero values
    scstats.probplot(np.sqrt(nonzeros_vals[col_name]), plot=ax_11)
    # Display skewness of all data
    print("skewness of all data")
    print("all, raw", scstats.skew(dataset_df[col_name]))
    print("all, sqrt", scstats.skew(np.sqrt(dataset_df[col_name])))
    print("all, log1p", scstats.skew(np.log1p(dataset_df[col_name])))
    # Display skewness of nonzero data
    print('skewness of nonzero data')
    print("nonzero, raw", scstats.skew(nonzeros_vals[col_name]))
    print("nonzero, sqrt", scstats.skew(np.sqrt(nonzeros_vals[col_name])))
    print("nonzero, boxcox1p", scstats.skew(
              boxcox1p(nonzeros_vals[col_name], scstats.
↪boxcox_normmax(nonzeros_vals[col_name]))
          ))
    print("nonzero, log1p", scstats.skew(np.log1p(nonzeros_vals[col_name])))
    # Show the 4x3 plot
    plt.show()

def fix_skewness_sqrt_inplace(dataset_df, col_name):
    dataset_df[col_name] = np.sqrt(dataset_df[col_name])

def fix_skewness_nonzero_boxcox1p_inplace(dataset_df, col_name):
    column_idx = dataset_df.columns.get_loc(col_name)
    nonzero_values = dataset_df[dataset_df[col_name] != 0]
    boxcox_norm = scstats.boxcox_normmax(nonzero_values[col_name])
    dataset_df.iloc[nonzero_values.index, column_idx] =␣
↪boxcox1p(nonzero_values[col_name], boxcox_norm)
```

[53]:
```
# Intermediate arrays

concat_fixdskew_df = concat_fixdtypes_df.copy()
```

[54]:
```
features_to_boxcox1p_col_names = [
    'LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
    'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
```

```
    'GrLivArea', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch',
    '3SsnPorch', 'ScreenPorch', 'PoolArea', 'Total_sqr_footage',␣
 ↪'Total_porch_sf']

# Review how log1p, sqrt or boxcox1p transformations will affect existing␣
 ↪distributions and their skewness
# for col_name in features_to_boxcox1p_col_names:
#     display_df_numerical_before_after_log(concat_fixdskew_df, feature)

# Fix skewnesses
for col_name in features_to_boxcox1p_col_names:
    fix_skewness_nonzero_boxcox1p_inplace(concat_fixdskew_df, col_name)

# Review fix results:
# for col_name in features_to_boxcox1p_col_names:
#     display_hist(concat_fixdskew_df, col_name, n_bins=100)

# Log-transform 'SalePrice' separately - concat_df contains lot of NaN values␣
 ↪(from the test_set part)
saleprice_values = concat_fixdskew_df.dropna()['SalePrice']
saleprice_column_idx = concat_fixdskew_df.columns.get_loc('SalePrice')
concat_fixdskew_df.iloc[saleprice_values.index, saleprice_column_idx] = np.
 ↪log1p(saleprice_values)

# Review results of fix for 'SalePrice'
# display_hist(concat_fixdskew_df.dropna(), 'SalePrice')
# display(scstats.skew(concat_fixdskew_df.dropna()['SalePrice']))
```

```
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/scipy/stats/stats.py:3399: PearsonRConstantInputWarning: An input array
is constant; the correlation coefficent is not defined.
  warnings.warn(PearsonRConstantInputWarning())
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/scipy/stats/stats.py:3429: PearsonRNearConstantInputWarning: An input
array is nearly constant; the computed correlation coefficent may be inaccurate.
  warnings.warn(PearsonRNearConstantInputWarning())
```

[55]:
```
# display_dataset_col_dtypes(concat_fixdskew_df)
```

[56]:
```
# Functions to encode categorical features using one-hot-encoding or custom␣
 ↪labelencoding

def _ugly_encode_cat_features(dataset_df):
    dataset_df['ExterQual'] = dataset_df['ExterQual'].map({
        'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['ExterCond'] = dataset_df['ExterCond'].map({
```

```
        'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['BsmtQual'] = dataset_df['BsmtQual'].map({
        'NoBsmt': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['BsmtCond'] = dataset_df['BsmtCond'].map({
        'NoBsmt': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['BsmtExposure'] = dataset_df['BsmtExposure'].map({
        'NoBsmt': 0, 'No': 1, 'Mn': 2, 'Av': 3, 'Gd': 4
    })
    dataset_df['BsmtFinType1'] = dataset_df['BsmtFinType1'].map({
        'NoBsmt': 0, 'LwQ': 1, 'Rec': 2, 'BLQ': 3, 'ALQ': 4, 'GLQ': 5, 'Unf': 2.
↪5
    })
    dataset_df['BsmtFinType2'] = dataset_df['BsmtFinType2'].map({
        'NoBsmt': 0, 'LwQ': 1, 'Rec': 2, 'BLQ': 3, 'ALQ': 4, 'GLQ': 5, 'Unf': 2.
↪5
    })
    dataset_df['HeatingQC'] = dataset_df['HeatingQC'].map({
        'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['CentralAir'] = dataset_df['CentralAir'].map({
        'N': 0, 'Y': 1
    })
    dataset_df['KitchenQual'] = dataset_df['KitchenQual'].map({
        'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['Functional'] = dataset_df['Functional'].map({
        'Sal': 0, 'Sev': 1, 'Maj2': 1.75, 'Maj1': 2, 'Mod': 3, 'Min2': 3.75,␣
↪'Min1': 4, 'Typ': 5
    })
    dataset_df['FireplaceQu'] = dataset_df['FireplaceQu'].map({
        'None': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['GarageFinish'] = dataset_df['GarageFinish'].map({
        'NoGarage': 0, 'RFn': 1, 'Fin': 2, 'Unf': 1.5
    })
    dataset_df['GarageQual'] = dataset_df['GarageQual'].map({
        'NoGarage': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['GarageCond'] = dataset_df['GarageCond'].map({
        'NoGarage': 0, 'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5
    })
    dataset_df['PoolQC'] = dataset_df['PoolQC'].map({
        'NoPool': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4
    })
```

[57]:
```
# Intermediate arrays

concat_catenc_df = concat_fixdskew_df.copy()
```

[58]:
```
# Encode categorical features

# For features, that have more-less meaning - use custom labelencoding
_ugly_encode_cat_features(concat_catenc_df)

# For features, that don't have more-less meaning - use onehot encoding
concat_catenc_df = pd.get_dummies(concat_catenc_df, drop_first=True)
```

[59]:
```
# display_dataset_col_dtypes(concat_catenc_df)

# display(concat_catenc_df.shape)
```

[60]:
```
# Review correlations between continuous columns

def display_corr_cols(dataset_df, col_names):
    fig, ax_0 = plt.subplots(figsize=(15, 15))
    corr = dataset_df[col_names].corr(method='pearson')
    ax_0.matshow(corr)
    for i in range(len(corr)):
        for j in range(len(corr)):
            text = ax_0.text(j, i, round(corr.iloc[i, j], 2), ha="center",␣
 ↪va="center", color="w")
    plt.xticks(range(len(corr.columns)), corr.columns)
    plt.xticks(rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.show()

# continous_columns = concat_catenc_df.drop('Id', axis=1).
 ↪select_dtypes(include='float64').columns.values
# display_corr_cols(concat_catenc_df, continous_columns)
```

[61]:
```
# Create pairplot for continuous data

def display_pairplot_cols(dataset_df, col_names):
    sns.pairplot(dataset_df[col_names])
    plt.show()

# continous_columns = concat_catenc_df.drop('Id', axis=1).
 ↪select_dtypes(include='float64').columns.values
# display_pairplot_cols(concat_catenc_df.dropna(), continous_columns)
```

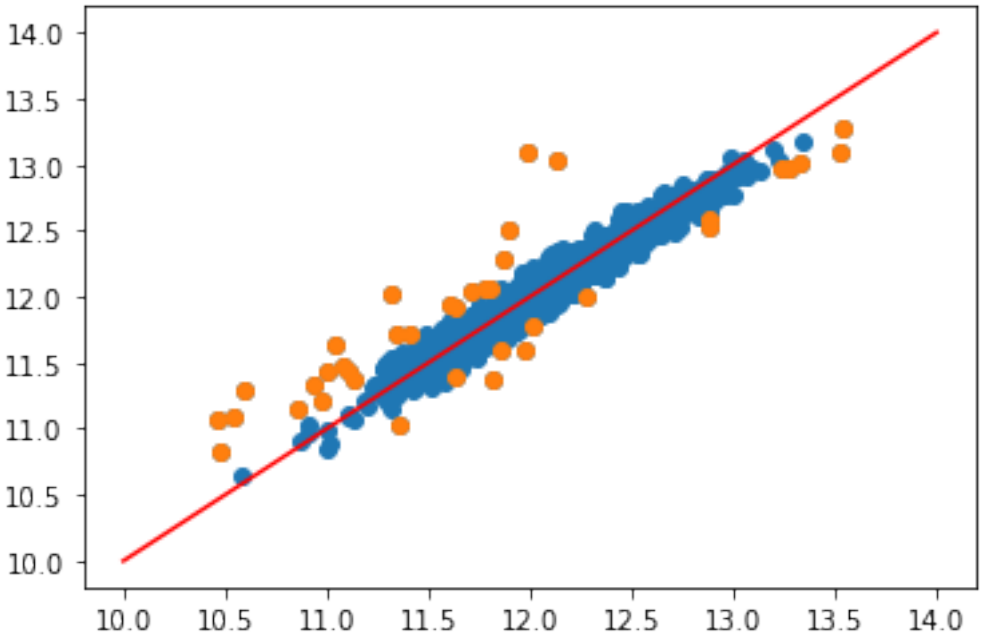
[62]:
```
# Functions to find outliers, using Ridge and ElasticNet

# src: https://www.kaggle.com/firstbloody/an-uncomplicated-model-top-2-or-top-1
```

```
def get_outliers_model(model, X, y, range_x, range_y):
    model.fit(X, y)
    display(
        np.sqrt(
            -cross_val_score(model, X, y, cv=10,␣
 ↪scoring='neg_mean_squared_error')
        ).mean()
    )
    y_pred = model.predict(X)
    resid = y - y_pred
    mean_resid = resid.mean()
    std_resid = resid.std()
    z = (resid - mean_resid) / std_resid
    z = np.array(z)
    outliers_model = np.where(
        abs(z) > abs(z).std() * 3
    )[0]
#     display(outliers_model)  # indices
    plt.scatter(y, y_pred)
    plt.scatter(y.iloc[outliers_model], y_pred[outliers_model])
    plt.plot(range_x, range_y, color="red")
    plt.show()
    return outliers_model
```

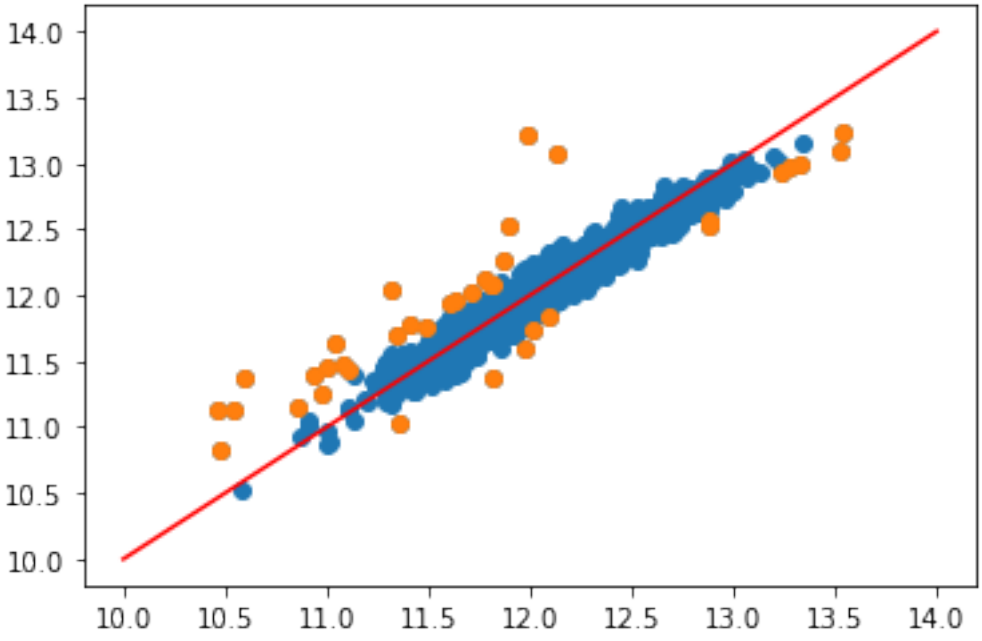
[63]:
```
X_train = concat_catenc_df.iloc[:train_df.index.size, :].drop(['Id',␣
 ↪'SalePrice'], axis=1)
y_train = concat_catenc_df.iloc[:train_df.index.size, :]['SalePrice']
```

[64]:
```
# Try out Ridge

model_ridge = Ridge(alpha=10)
range_x, range_y = (range(10, 15), range(10, 15))

ridge_outliers = get_outliers_model(model_ridge, X_train, y_train, range_x,␣
 ↪range_y)

# Try out ElasticNet

model_elnet = ElasticNet(alpha=0.001, l1_ratio=0.58)
range_x, range_y = (range(10, 15), range(10, 15))

elnet_outliers = get_outliers_model(model_elnet, X_train, y_train, range_x,␣
 ↪range_y)
```

```
0.1251602972304044
```




```
0.124510460532152
```



[65]:
```
# Remove the outliers
outliers_to_remove = list()
for ridge_outlier in ridge_outliers:
    for elnet_outlier in elnet_outliers:
        if ridge_outlier == elnet_outlier:
            outliers_to_remove.append(ridge_outlier)

# display("outliers to remove", outliers_to_remove)

X_train = X_train.drop(outliers_to_remove)
y_train = y_train.drop(outliers_to_remove)
```

[66]:
```
# Base modelling

# Note: hyperparams are from src: https://www.kaggle.com/firstbloody/
 ↪an-uncomplicated-model-top-2-or-top-1

base_X_train = X_train.copy()
base_y_train = y_train.copy()
base_X_test = concat_catenc_df.iloc[train_df.index.size:, :].drop(['SalePrice',␣
 ↪'Id'], axis=1)

def display_model_scores(model, X, y):
    display(model.__class__.__name__)
    display(
        "cross_val_score, k=10, neg_mean_squared_error mean",
        np.sqrt(
            -cross_val_score(base_gbr, X_train, y_train, cv=10,␣
 ↪scoring="neg_mean_squared_error", n_jobs=-1)
        ).mean()
    )
```

[67]:
```
base_gbr = GradientBoostingRegressor(max_depth=4, n_estimators=150)
base_gbr.fit(base_X_train, base_y_train)
display_model_scores(base_gbr, base_X_train, base_y_train)

base_xgb = XGBRegressor(max_depth=5, n_estimators=400)
base_xgb.fit(base_X_train, base_y_train)
display_model_scores(base_xgb, base_X_train, base_y_train)

base_lasso = Lasso(alpha=0.00047)
base_lasso.fit(base_X_train, base_y_train)
display_model_scores(base_lasso, base_X_train, base_y_train)

base_ridge = Ridge(alpha=0.00047)
base_ridge.fit(base_X_train, base_y_train)
display_model_scores(base_ridge, base_X_train, base_y_train)
```

```
base_rfr = RandomForestRegressor(max_depth=4, n_estimators=200)
base_rfr.fit(base_X_train, base_y_train)
display_model_scores(base_rfr, base_X_train, base_y_train)
```

```
'GradientBoostingRegressor'
```

```
'cross_val_score, k=10, neg_mean_squared_error mean'
```

```
0.10124850316323661
```

```
/home/max/.local/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
[13:51:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
```

```
'XGBRegressor'
```

```
'cross_val_score, k=10, neg_mean_squared_error mean'
```

```
0.10175106815502263
```

```
'Lasso'
```

```
'cross_val_score, k=10, neg_mean_squared_error mean'
```

```
0.10140478041579457
```

```
'Ridge'
```

```
'cross_val_score, k=10, neg_mean_squared_error mean'
```

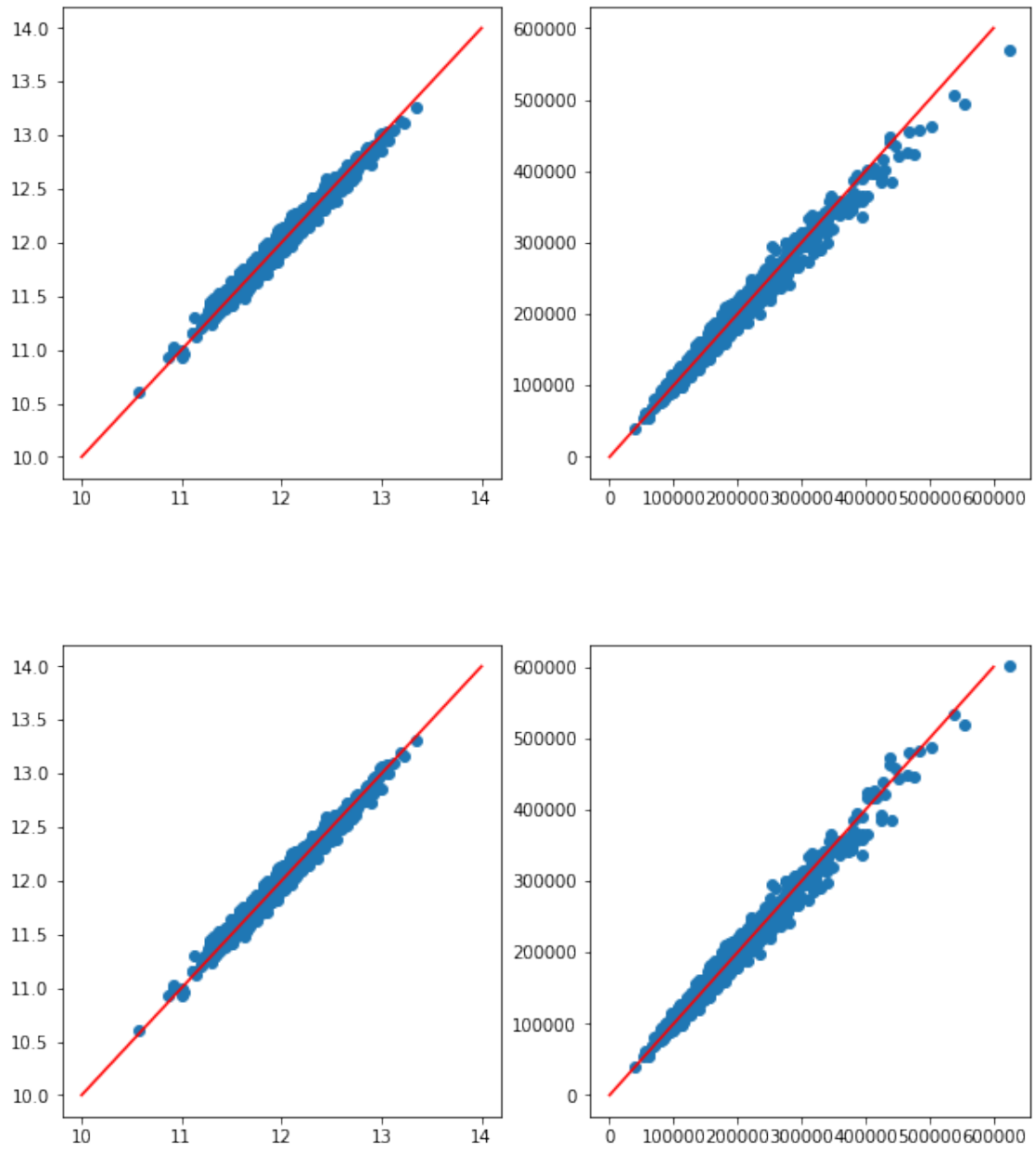
```
0.10151526412986037
```

```
'RandomForestRegressor'
```

```
'cross_val_score, k=10, neg_mean_squared_error mean'
```

```
0.10154553353075113
```

[68]:
```
# Observe the prediction effect of our model on the training set

# NOTE: the bottom point is not on the red line like the top point (first graph␣
 ↪above)
# That indicates that the bottom point may not be predictive.

# Fix - manually adjust it and use the quantile to select the predicted value␣
 ↪we want to adjust -> adjust it.

# src: https://www.kaggle.com/firstbloody/an-uncomplicated-model-top-2-or-top-1

train_predict = 0.1 * base_gbr.predict(base_X_train) + \
                0.3 * base_xgb.predict(base_X_train) + \
                0.3 * base_lasso.predict(base_X_train) + \
                0.3 * base_ridge.predict(base_X_train)

# train_predict = base_lasso.predict(base_X_train)

fig, [ax_0, ax_1] = plt.subplots(1, 2, figsize=(10, 5))
ax_0.scatter(base_y_train, train_predict)
ax_0.plot(range(10, 15), range(10, 15), color='red')
ax_1.scatter(np.exp(base_y_train), np.exp(train_predict))
ax_1.plot(range(600000), range(600000), color='red')
plt.show()

q1_start = pd.DataFrame(train_predict).quantile(0.987)
q2_end = pd.DataFrame(train_predict).quantile(1)

pre_df = pd.DataFrame({'SalePrice': train_predict})

pre_df.loc[(pre_df['SalePrice'] >= q1_start[0]) & (pre_df['SalePrice'] <=␣
 ↪q2_end[0]), 'SalePrice'] = \
    pre_df.loc[(pre_df['SalePrice'] >= q1_start[0]) & (pre_df['SalePrice'] <=␣
 ↪q2_end[0]), 'SalePrice'] * 1.004

train_predict = np.array(pre_df['SalePrice'])

fig, [ax_0, ax_1] = plt.subplots(1, 2, figsize=(10, 5))
ax_0.scatter(base_y_train, train_predict)
ax_0.plot(range(10, 15), range(10, 15), color='red')
ax_1.scatter(np.exp(base_y_train), np.exp(train_predict))
ax_1.plot(range(600000), range(600000), color='red')
plt.show()
```



[69]:
```
# Apply quantiles fix for test_set

test_predict = 0.1 * base_gbr.predict(base_X_test) + \
               0.3 * base_xgb.predict(base_X_test) + \
               0.3 * base_lasso.predict(base_X_test) + \
               0.3 * base_ridge.predict(base_X_test)

# test_predict = base_lasso.predict(base_X_test)

q1_start = pd.DataFrame(test_predict).quantile(0.987)
```

```
q2_end = pd.DataFrame(test_predict).quantile(1)

pre_df = pd.DataFrame({'SalePrice': test_predict})

pre_df.loc[(pre_df['SalePrice'] >= q1_start[0]) & (pre_df['SalePrice'] <=␣
 ↪q2_end[0]), 'SalePrice'] = \
    pre_df.loc[(pre_df['SalePrice'] >= q1_start[0]) & (pre_df['SalePrice'] <=␣
 ↪q2_end[0]), 'SalePrice'] * 1.004

y_pred = np.exp(np.array(pre_df['SalePrice']))

display(y_pred)
```

```
array([120444.37118087, 171195.33908945, 183587.09456563, ...,
       175002.98010799, 114711.76593807, 219280.16181007])
```

[70]:
```
# Final prediction

# model = GradientBoostingRegressor()
# model.fit(X_train_remfeat, y_train_remfeat)
# y_pred_log = model.predict(X_test_remfeat)
# y_pred = np.exp(y_pred_log)
# display(y_pred)

# y_pred = base_ridge.predict(base_X_test)
# y_pred = np.exp(y_pred)
# display(y_pred)

# y_pred = train_predict

# display(np.exp(y_pred))
```

[71]:
```
# Predictions submission

submission = pd.DataFrame(
    {'id': test_df['Id'], 'SalePrice': y_pred}
)
submission.to_csv('submission.csv', index=False)
```