PS3

September 26, 2025

0.1 Problem set 3

0.2 Name: [Yawen Tan]

0.3 Link to your PS3 github repo: [https://github.com/IsabellaTan/Brown-DATA1030-HW3.git]

0.3.1 Problem 0

-2 points for every missing green OK sign.

Make sure you are in the DATA1030 environment.

[1]:
```
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.12.10 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
                  % (lib, min_ver, ver))
```

```
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.12.10"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.12.10"):
    print(FAIL, "Python version 3.12.10 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'numpy': "2.2.5", 'matplotlib': "3.10.1",'sklearn': "1.6.1",
                'pandas': "2.2.3",'xgboost': "3.0.0", 'shap': "0.47.2",
                'polars': "1.27.1", 'seaborn': "0.13.2"}

# now the dependencies
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.12.10
```

```
[ OK ] numpy version 2.2.5 is installed.
[ OK ] matplotlib version 3.10.1 is installed.
[ OK ] sklearn version 1.6.1 is installed.
[ OK ] pandas version 2.2.3 is installed.
[ OK ] xgboost version 3.0.0 is installed.
[ OK ] shap version 0.47.2 is installed.
[ OK ] polars version 1.27.1 is installed.
[ OK ] seaborn version 0.13.2 is installed.
```

0.4 Problem 1: EDA and visualizations

0.4.1 Problem 1a: EDA (5 points)

One of the datasets we will be working with this semester is the kaggle house price dataset. The goal of PS3 is to use this dataset to practice dataframe manipulations and perform EDA. The dataset, and its description, are located in the `data` folder.

Carefully read the dataset description. Whenever you work with a dataset, it is highly recommended that you prepare a similar description if it is not readily available. Specific things to note:

- each feature is described in full detail,

- the meaning of continuous features is explained and their unit is provided (e.g., lot size is measured in square feet),
- each category in a categorical or ordinal feature is spelled out and explained.

Answer the following EDA-related questions.

The sequence of questions here are typical things to ask when you perform EDA on a new dataset. First, you always want to know how many data points and features you have, and whether they are continuous, ordinal, or categorical. You should then take a closer look at the target variable. We will study the properties of the features and the relationships between the features and the target variable in 1b.

Q0 First, read the data into a data frame and display the columns of the data frame below. You might encounter error messages and other issues along the way. Please diagnose and resolve them.

[2]:
```
# your code here

import pandas as pd
df = pd.read_excel('C:/Users/DELL/Desktop/zy/OneDrive/Brown/DATA1030/
 ↪assignment3/Brown-DATA1030-HW3/data/train.xlsx', sheet_name='data')
print(df.columns)
```

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

Q1 How many rows and columns do we have in the dataframe?

[ ]:
```
# your code here

print("The number of row is " + str(df.shape[0]))
print("The number of columb is " + str(df.shape[1]))
```

```
The number of row is 1460
The number of columb is 80
```

Q2 What are the data types of the columns? Make sure that the output is not truncated and you

see the type of each column.

[5]:
```
# your code here

# Create a for loop to iterate through each column and print its name and data␣
 ↪type
for col, dtype in df.dtypes.items():
    print(f"{col}: {dtype}")
```

```
MSSubClass: int64
MSZoning: object
LotFrontage: float64
LotArea: int64
Street: object
Alley: object
LotShape: object
LandContour: object
Utilities: object
LotConfig: object
LandSlope: object
Neighborhood: object
Condition1: object
Condition2: object
BldgType: object
HouseStyle: object
OverallQual: int64
OverallCond: int64
YearBuilt: int64
YearRemodAdd: int64
RoofStyle: object
RoofMatl: object
Exterior1st: object
Exterior2nd: object
MasVnrType: object
MasVnrArea: float64
ExterQual: object
ExterCond: object
Foundation: object
BsmtQual: object
BsmtCond: object
BsmtExposure: object
BsmtFinType1: object
BsmtFinSF1: int64
BsmtFinType2: object
BsmtFinSF2: int64
BsmtUnfSF: int64
TotalBsmtSF: int64
Heating: object
```

```
HeatingQC: object
CentralAir: object
Electrical: object
1stFlrSF: int64
2ndFlrSF: int64
LowQualFinSF: int64
GrLivArea: int64
BsmtFullBath: int64
BsmtHalfBath: int64
FullBath: int64
HalfBath: int64
BedroomAbvGr: int64
KitchenAbvGr: int64
KitchenQual: object
TotRmsAbvGrd: int64
Functional: object
Fireplaces: int64
FireplaceQu: object
GarageType: object
GarageYrBlt: float64
GarageFinish: object
GarageCars: int64
GarageArea: int64
GarageQual: object
GarageCond: object
PavedDrive: object
WoodDeckSF: int64
OpenPorchSF: int64
EnclosedPorch: int64
3SsnPorch: int64
ScreenPorch: int64
PoolArea: int64
PoolQC: object
Fence: object
MiscFeature: object
MiscVal: int64
MoSold: int64
YrSold: int64
SaleType: object
SaleCondition: object
SalePrice: int64
```

Q3 The ML target variable in this dataset is the sale price. We will develop ML pipelines to predict this variable based on the other features.

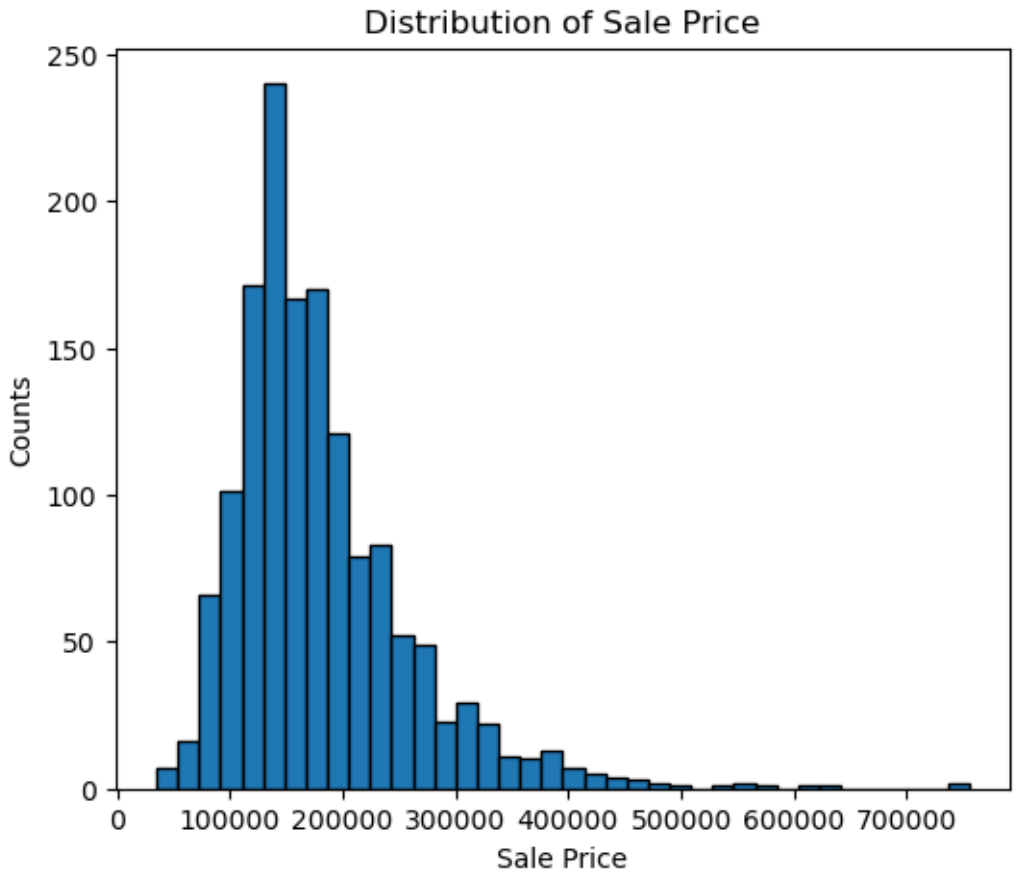
Is this column continuous or categorical? Please use .describe or .value_counts to take a quick look at this feature.

[ ]:
```
# your code here

print(df["SalePrice"].describe())
# 'SalePrice' is countinuous variable
```

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

Q4 Visualize the target variable. Don't forget the axis labels and graph title. Make sure to use appropriate arguments to best display the data.

[ ]:
```
# your code here

import matplotlib
from matplotlib import pylab as plt
import numpy as np
# Set the figure size
plt.figure(figsize=(6,5))
# Create the histogram
df['SalePrice'].plot.hist(
    bins = int(np.sqrt(df.shape[0])), # Let bins be the square root of the␣
 ↪number of rows
    edgecolor='black', # Color of the edge of the bars
    linewidth=1 # Width of the edge of the bars
    )
# Add labels and title
plt.xlabel('Sale Price')
plt.ylabel('Counts')
plt.title('Distribution of Sale Price')

plt.show()
```



0.4.2 Problem 1b: visualization (10 points)

Find one continuous, one ordinal, and one categorical feature that strongly correlates with the sale price. Create figures that illustrate your selected features and the sale price.

Don't forget to add axis labels and titles, and find apropriate arguments. Write figure captions to explain what the figure shows.

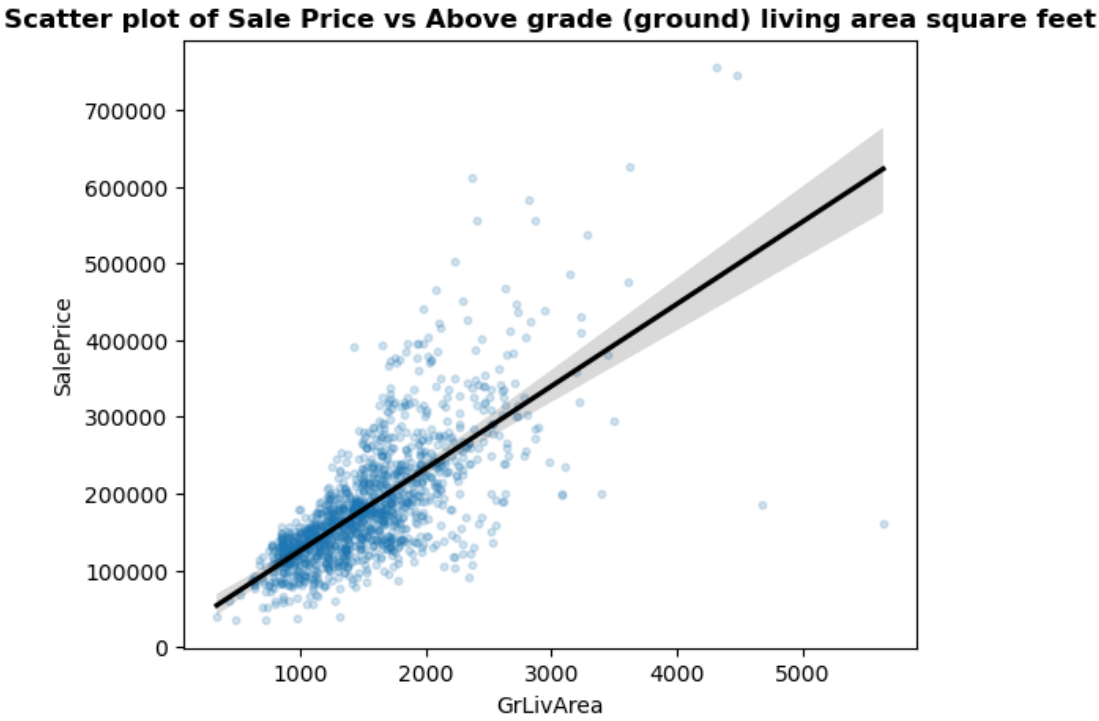
If you know how to quantitatively assess correlation strengths between variables, feel free to use those techniques. Qualitative/visual assessment also works for now.

[ ]:
```
# your code here
import matplotlib.pyplot as plt
import seaborn as sns

# Features that we choose:
# continuous feature : GrLivArea which is Above grade (ground) living area␣
 ↪square feet
# ordinal feature: OverallQual which is Overall material and finish of the house
# categorical feature: Alley which is Type of alley access to property
```

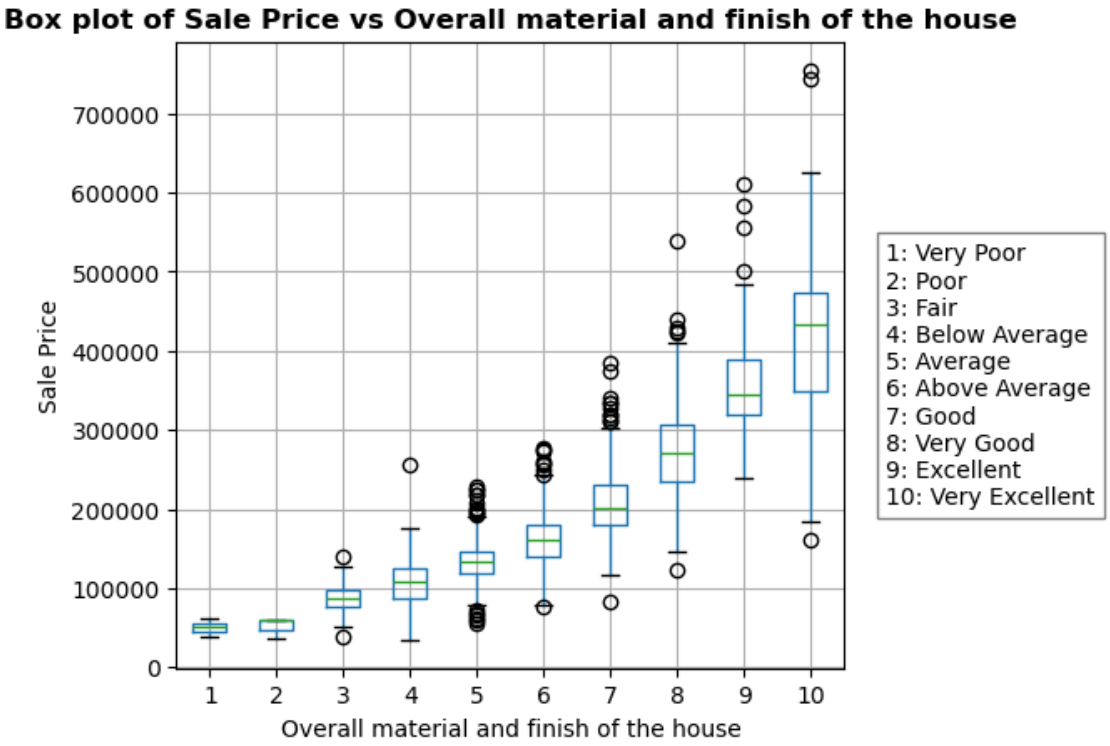
```
# Create a scatter plot for GrLivArea vs SalePrice
df.plot.scatter('GrLivArea','SalePrice',alpha=0.1,s=10, figsize=(6,5))
# Add labels and title
plt.xlabel('Above grade (ground) living area square feet')
plt.ylabel('Sale Price')
plt.title('Scatter plot of Sale Price vs Above grade (ground) living area␣
 ↪square feet',weight='bold')
# Add a regression line
sns.regplot(x='GrLivArea', y='SalePrice', data=df, scatter_kws={'alpha':0.1,␣
 ↪'s':10}, line_kws={'color':'black'})

plt.show()
# Calculate the Pearson correlation coefficient
corr1 = round(df['GrLivArea'].corr(df['SalePrice']),2)

print('The Pearson correlation coefficient between GrLivArea and SalePrice is␣
 ↪'+ str(corr1))
print('Based on the scatter plot and the correlation coefficient, there is a␣
 ↪strong positive linear relationship between GrLivArea and SalePrice. As the␣
 ↪above grade living area increases, the sale price tends to increase as well.
 ↪')
```

```
# Create a dictionary to map the ordinal values to their corresponding labels
cond_labels = {
    1: 'Very Poor',
    2: 'Poor',
    3: 'Fair',
    4: 'Below Average',
    5: 'Average',
    6: 'Above Average',
    7: 'Good',
    8: 'Very Good',
    9: 'Excellent',
    10: 'Very Excellent'}
# Create a box plot for OverallQual vs SalePrice
df[['OverallQual','SalePrice']].boxplot(by='OverallQual',figsize=(5,5))
# Add labels and title
plt.ylabel('Sale Price')
plt.xlabel('Overall material and finish of the house')
```

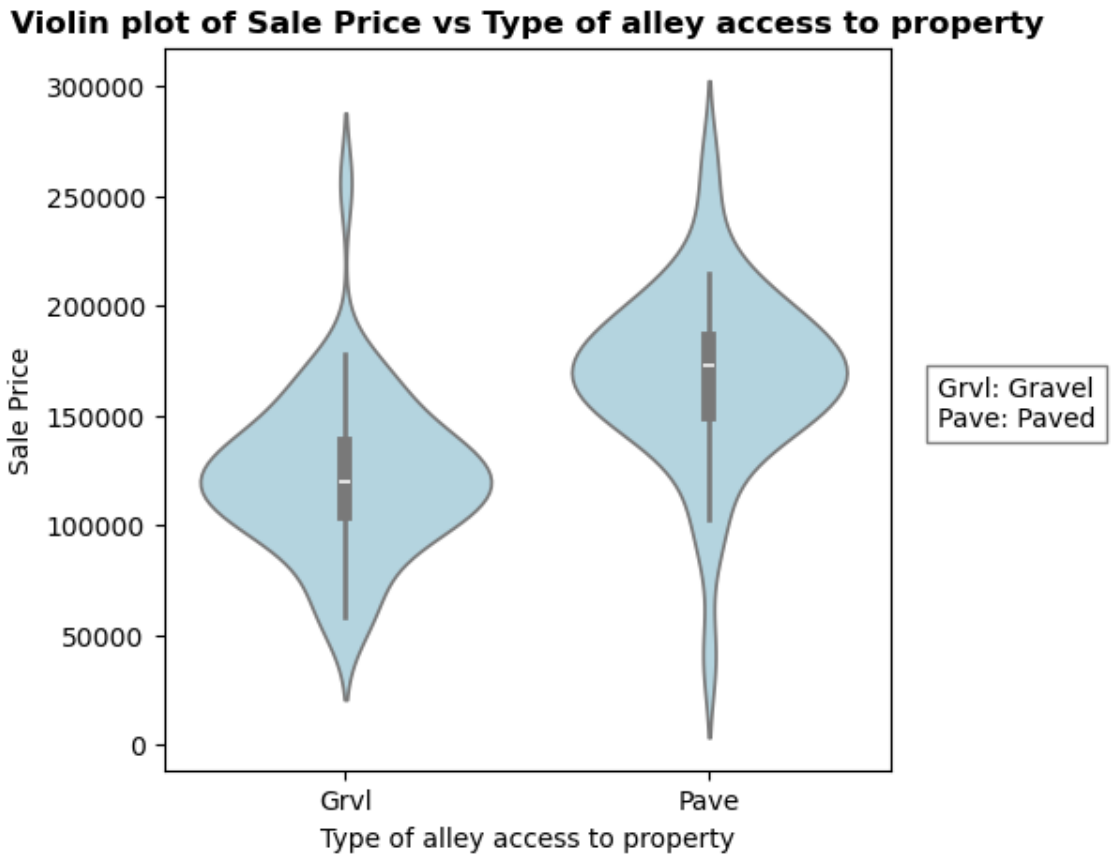
```
plt.title('Box plot of Sale Price vs Overall material and finish of the house',␣
 ↪weight='bold')
# Delete the automatic 'Boxplot grouped by group_by_column' title
plt.suptitle('')
# Add the mapping of ordinal values to their corresponding labels on the right␣
 ↪side of the plot
textstr = '\n'.join([f"{k}: {v}" for k,v in sorted(cond_labels.items())])
plt.gcf().text(0.95, 0.5, textstr, fontsize=10, va='center',␣
 ↪bbox=dict(facecolor='white', alpha=0.5))

plt.show()
# Calculate the Spearman correlation
corr2 = round(df['OverallQual'].corr(df['SalePrice'], method='spearman'),2)

print('The Spearman correlation coefficient between OverallQual and SalePrice␣
 ↪is ' + str(corr2))
print('Based on the box plot and the correlation coefficient, there is a strong␣
 ↪positive monotonic relationship between OverallQual and SalePrice. As the␣
 ↪overall quality of the house increases, the sale price tends to increase as␣
 ↪well.')



# Create a dictionary to map the categorical values to their corresponding␣
 ↪labels
alley_labels = {
    'Grvl':'Gravel',
    'Pave':'Paved'}
# Create a violin plot for Alley vs SalePrice
plt.figure(figsize=(5,5))
sns.violinplot(x='Alley', y='SalePrice', data=df, color='lightblue')
# Add labels and title
plt.ylabel('Sale Price')
plt.xlabel('Type of alley access to property')
plt.title('Violin plot of Sale Price vs Type of alley access to property',␣
 ↪weight='bold')
# Add the mapping of categorical values to their corresponding labels on the␣
 ↪right side of the plot
textstr2 = '\n'.join([f"{k}: {v}" for k,v in sorted(alley_labels.items())])
plt.gcf().text(0.95, 0.5, textstr2, fontsize=10, va='center',␣
 ↪bbox=dict(facecolor='white', alpha=0.5))
plt.show()
print('Based on the violin plot, houses with paved alley access tend to have␣
 ↪higher sale prices compared to those with gravel alley access. The type of␣
 ↪alley access appears to have an impact on the sale price of the house.')
```



The Pearson correlation coefficient between GrLivArea and SalePrice is 0.71
Based on the scatter plot and the correlation coefficient, there is a strong positive linear relationship between GrLivArea and SalePrice. As the above grade living area increases, the sale price tends to increase as well.



The Spearman correlation coefficient between OverallQual and SalePrice is 0.81 Based on the box plot and the correlation coefficient, there is a strong positive monotonic relationship between OverallQual and SalePrice. As the overall quality of the house increases, the sale price tends to increase as well.



Based on the violin plot, houses with paved alley access tend to have higher sale prices compared to those with gravel alley access. The type of alley access appears to have an impact on the sale price of the house.

0.5 Problem 2: basic splitting strategy (15 points)

Write a general function that performs basic splitting on a dataset, while also conducting integrity tests on both its inputs and outputs. The function is called basic_split, it is outlined in the cell below. It takes the following arguments as inputs: feature matrix (X), a target variable (y), train_size, val_size, test_size, and random_state. The output of the function should be: X_train, y_train, X_val, y_val, X_test, y_test, the three sets split according to the train, val, test sizes.

This function is general purpose, you'll be able to reuse it for any project if you want to perform basic split on your data.

[91]:
```
def basic_split(feature_matrix,target_variable,train_size = 0.6,val_size=0.
↪2,test_size=0.2,random_state=42):
    '''
    Split dataframes (feature matrix X and target variable y) into random␣
↪train, validation and test sets
```

```
    Parameters:
    -----------

    feature_matrix: a dataframe that contains your feature matrix
    target_variable: a series that contains your target variable
    train_size: a float between 0.0 and 1.0, it represents the proportion
        of the dataset to include in the training set
    val_size: a float between 0.0 and 1.0, it represents the proportion
        of the dataset to include in the validation set
    test_size: a float between 0.0 and 1.0, it represents the proportion
        of the dataset to include in the test set
    random_state: an int, it controls the shuffling applied to the data
        before applying the split

    NOTE: train_size+val_size+test_size must be equal to 1.

    Returns:
    --------
        a tuple containing the train, validation, and test sets

    Example:
    --------

    >>> import numpy as np
    >>> from sklearn.model_selection import train_test_split
    >>>
    >>> X, y = np.arange(10).reshape((5, 2)), range(5)
    >>>
    >>> X_train, y_train, X_val, y_val, X_test, y_test = basic_split(X,y)

    '''

    # ***************************
    # TODO: test the inputs first
    # ***************************
    # write an if statement to perform each of these checks
    # Important: raise a ValueError with a descriptive error message if␣
↪something is off
    # call basic_split with incorrect arguments to make sure all of the tests␣
↪work as intended!
    import pandas as pd
    import polars as pl

    # test 1: if feature_matrix is not a dataframe (pandas or polars), raise␣
↪ValueError
    if not isinstance(feature_matrix, (pd.DataFrame, pl.DataFrame)):
        raise ValueError("Feature_matrix is not a dataframe (pandas or polars)")
```

```
    # test 2: if the target-variable is not a series (pandas or polars), raise␣
↪ValueError
    if not isinstance(target_variable, (pd.Series, pl.Series)):
        raise ValueError("Target_variable is not a series (pandas or polars)")
    # test 3: if the number of rows in feature_matrix is not equal to the␣
↪length of target_variable, raise a ValueError
    if feature_matrix.shape[0] != len(target_variable):
        raise ValueError("The number of rows in feature_matrix is not equal to␣
↪the length of target_variable")
    # test 4: if train_size is less than 0.0 or larger than 1.0, raise␣
↪ValueError
    if train_size < 0.0 or train_size > 1.0:
        raise ValueError("train_size is less than 0.0 or larger than 1.0")
    # test 5: if val_size is less than 0.0 or larger than 1.0, raise ValueError
    if val_size < 0.0 or val_size > 1.0:
        raise ValueError("val_size is less than 0.0 or larger than 1.0")
    # test 6: if test_size is less than 0.0 or larger than 1.0, raise ValueError
    if test_size < 0.0 or test_size > 1.0:
        raise ValueError("test_size is less than 0.0 or larger than 1.0")
    # test 7: if train_size+val_size+test_size is not equal to 1.0, raise␣
↪ValueError
    if train_size + val_size + test_size != 1.0:
        raise ValueError("train_size+val_size+test_size is not equal to 1.0")
    # test 8: if random state is not an integer, raise ValueError
    if not isinstance(random_state, int):
        raise ValueError("random_state is not an integer")


    # ************************************
    # TODO: implement the splitting strategy
    # ************************************
    # as we discussed in class, use sklearn's train_test_split twice
    from sklearn.model_selection import train_test_split
    X_train, X_other, y_train, y_other = train_test_split(feature_matrix,␣
↪target_variable, train_size=train_size, random_state=random_state)
    X_val, X_test, y_val, y_test = train_test_split(X_other, y_other,␣
↪test_size=test_size/(test_size + val_size), random_state=random_state)


    # **********************
    # TODO: test the outputs
    # **********************
    # same as above, write an if statement to perform each of these checks
    # raise a ValueError with a descriptive error message if something is off.
```

```
    # test 1: the number of rows in X_train divided by the number of rows in X␣
 ↪should be close to train_size
    # think why we sometimes cannot achieve exact equality
    # and how you should express this as a condition in the if statement
    if not abs((X_train.shape[0] / feature_matrix.shape[0]) - train_size) < 0.
 ↪05:
        raise ValueError("The number of rows in X_train divided by the number␣
 ↪of rows in X is not close to train_size")
    # test 2: the number of rows in X_val divided by the number of rows in X␣
 ↪should be close to val_size
    if not abs((X_val.shape[0] / feature_matrix.shape[0]) - val_size) < 0.05:
        raise ValueError("The number of rows in X_val divided by the number of␣
 ↪rows in X is not close to val_size")
    # test 3: the number of rows in X_test divided by the number of rows in X␣
 ↪should be close to test_size
    if not abs((X_test.shape[0] / feature_matrix.shape[0]) - test_size) < 0.05:
        raise ValueError("The number of rows in X_test divided by the number of␣
 ↪rows in X is not close to test_size")
    # test 4: make sure that the length of y_train, y_val, y_test is equal to
    # the number of rows in X_train, X_val, X_test, respectively
    if not (len(y_train) == X_train.shape[0] and len(y_val) == X_val.shape[0]␣
 ↪and len(y_test) == X_test.shape[0]):
        raise ValueError("The length of y_train, y_val, y_test is not equal to␣
 ↪the number of rows in X_train, X_val, X_test, respectively")


    return X_train, y_train, X_val, y_val, X_test, y_test

# Call the function and preform tests here
# test 1: Apply the function to the house price dataset from problem 1 with␣
 ↪train_size = 0.6, val_size = 0.2, and test_size = 0.2.
X = df.drop(columns=['SalePrice'])
y = df['SalePrice']
X_train, y_train, X_val, y_val, X_test, y_test = basic_split(X,y)
# test 2: Print out the head of X_train, X_val, and X_test.
print("Train/Val/Test sizes:", len(X_train), len(X_val), len(X_test))
print("X_train head:\n", X_train.head())
print("X_val head:\n", X_val.head())
print("X_test head:\n", X_test.head())
# test 3: Make sure that you get the same points in each set every time you␣
 ↪rerun the cell (a.k.a., test for reproducability).
X_train1, y_train1, X_val1, y_val1, X_test1, y_test1 = basic_split(X, y)
X_train2, y_train2, X_val2, y_val2, X_test2, y_test2 = basic_split(X, y)
print('Checking reproducibility:')
print('If X_train is the same:', X_train1.equals(X_train2))
print('If y_train is the same:', y_train1.equals(y_train2))
```

```
print('If X_val is the same:', X_val1.equals(X_val2))
print('If y_val is the same:', y_val1.equals(y_val2))
print('If X_test is the same:', X_test1.equals(X_test2))
print('If y_test is the same:', y_test1.equals(y_test2))
# test 4: Try a couple of other train, val, test sizes here. make sure to test␣
 ↪the possible extreme values!
X_train_extrem, y_train_extrem, X_val_extrem, y_val_extrem, X_test_extrem,␣
 ↪y_test_extrem = basic_split(X, y, train_size = 0.8,val_size=0.1,test_size=0.
 ↪1)
X_train_extrem2, y_train_extrem2, X_val_extrem2, y_val_extrem2, X_test_extrem2,␣
 ↪y_test_extrem2 = basic_split(X, y, train_size = 0.9,val_size=0.
 ↪05,test_size=0.05)

# notice how most of the lines in basic_split are about testing the inputs and␣
 ↪outputs and
# testing the inputs ensures that the user correctly calls the function. if␣
 ↪they do not, a descriptive error message is returned.
# testing the outputs ensures that your code correctly performs the intended␣
 ↪operation anticipating edge cases and potential issues.
# this is pretty typical in software engineering and this is the key to writing␣
 ↪reusable code.
```

```
Train/Val/Test sizes: 876 292 292
X_train head:
     MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
314          70       RM         60.0     9600   Pave  Grvl      Reg
442          50       RM         52.0     6240   Pave   NaN      Reg
319          80       RL          NaN    14115   Pave   NaN      Reg
767          50       RL         75.0    12508   Pave   NaN      IR1
756          60       RL         68.0    10769   Pave   NaN      IR1

    LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC Fence  \
314         Lvl    AllPub    Inside  …           0        0    NaN   NaN
442         Lvl    AllPub    Inside  …           0        0    NaN   NaN
319         Lvl    AllPub    Inside  …           0        0    NaN   NaN
767         Lvl    AllPub    Inside  …           0        0    NaN   NaN
756         Lvl    AllPub    Inside  …           0        0    NaN   NaN

    MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
314         NaN       0       8    2006        WD         Normal
442         NaN       0       6    2008        WD         Normal
319         NaN       0       6    2009        WD         Normal
767        Shed    1300       7    2008        WD         Normal
756         NaN       0       4    2009        WD         Normal

[5 rows x 79 columns]
X_val head:
```

```
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
1336          90       RL         87.0     9246   Pave   NaN      IR1
178           20       RL         63.0    17423   Pave   NaN      IR1
619           60       RL         85.0    12244   Pave   NaN      Reg
548           20       RM         49.0     8235   Pave   NaN      IR1
1046          60       RL         85.0    16056   Pave   NaN      IR1

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC  Fence  \
1336         Lvl    AllPub    Inside  …           0        0    NaN    NaN
178          Lvl    AllPub   CulDSac  …           0        0    NaN    NaN
619          Lvl    AllPub    Inside  …           0        0    NaN    NaN
548          HLS    AllPub    Inside  …           0        0    NaN  MnPrv
1046         Lvl    AllPub    Inside  …           0        0    NaN    NaN

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
1336         NaN       0      11    2008        WD         Normal
178          NaN       0       7    2009       New        Partial
619          NaN       0       8    2008        WD         Normal
548          NaN       0       6    2008        WD         Normal
1046         NaN       0       7    2006       New        Partial

[5 rows x 79 columns]
X_test head:
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
1290          80       RL          NaN    14112   Pave   NaN      IR1
282          120       RL         34.0     5063   Pave   NaN      Reg
836           30       RM         90.0     8100   Pave  Pave      Reg
594           20       RL         88.0     7990   Pave   NaN      IR1
544           60       RL         58.0    17104   Pave   NaN      IR1

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC  Fence  \
1290         Lvl    AllPub    Corner  …           0        0    NaN    NaN
282          Lvl    AllPub    Inside  …           0        0    NaN    NaN
836          Lvl    AllPub    Inside  …           0        0    NaN   GdWo
594          Lvl    AllPub    Inside  …           0        0    NaN  MnPrv
544          Lvl    AllPub    Inside  …           0        0    NaN    NaN

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
1290         NaN       0       4    2010        WD         Normal
282          NaN       0       4    2009     ConLw         Normal
836          NaN       0       6    2007       COD         Normal
594          NaN       0       4    2008        WD         Normal
544          NaN       0       9    2006       New        Partial

[5 rows x 79 columns]
Checking reproducibility:
If X_train is the same: True
If y_train is the same: True
```

```
If X_val is the same: True
If y_val is the same: True
If X_test is the same: True
If y_test is the same: True
```

1 Problem 3: kfold splitting (20 points)

Write a function that performs kfold splitting. We provided the input arguments and the header of the function. You need test the inputs, implement the algorithm, and test the outputs.

[8]:
```
def basic_kfold(feature_matrix,target_variable, k = 5, shuffle = True,␣
↪random_state = 42):
    '''
    Split dataframes (feature matrix X and target variable y) into `k` number␣
↪of equal size folds.
    One fold is used as the test set.
    Iterate over the remaining k-1 folds. Fold i is used as the validation set,
    the remaining folds are used as the training set.

    Parameters:
    -----------

    feature_matrix: a dataframe that contains your feature matrix
    target_variable: a series that contains your target variable
    k: an int, the number of folds
    shuffle: boolean variable. If True, the feature matrix and the target␣
↪variable are shuffled
        before the folds are created to randomize the sets
    random_state: an int, it controls the shuffling applied to the data

    Returns:
    --------
        one test set and a list with `k-1` elements containing tuples of the␣
↪train and validation sets

    >>> import numpy as np
    >>> from sklearn.model_selection import train_test_split
    >>>
    >>> X, y = np.arange(10).reshape((5, 2)), range(5)
    >>>
    >>> X_test, y_test, train_val_sets = basic_kfold(X,y)

    '''
    import pandas as pd
    import polars as pl
    import numpy as np
```

```
    # one of the outputs, it will contain the train and validation sets

    # train_val_sets structure:
    # It is a list with (k-1) elements.
    # Each element is a tuple: (train_set, val_set)
    # train_set = (X_train, y_train)   # features and labels for training
    # val_set   = (X_val, y_val)       # features and labels for validation
    # train_val_sets = [
    #     ((X_train1, y_train1), (X_val1, y_val1)),   # fold 1
    #     ((X_train2, y_train2), (X_val2, y_val2)),   # fold 2
    #     ...
    #     ((X_train{k-1}, y_train{k-1}), (X_val{k-1}, y_val{k-1}))  # fold k-1
    # ]

    train_val_sets = []

    # **********************
    # TODO: test the inputs!
    # **********************
    # test each of the input arguments. consider their types and what values␣
↪are possible as you come up with the tests
    # come up with at least 8 tests
    # among other things, consider what the smallest and largest k we can have

    # test 1: if feature_matrix is not a dataframe (pandas or polars), raise␣
↪ValueError
    if not isinstance(feature_matrix, (pd.DataFrame, pl.DataFrame)):
        raise ValueError("Feature_matrix is not a dataframe (pandas or polars)")
    # test 2: if the target-variable is not a series (pandas or polars), raise␣
↪ValueError
    if not isinstance(target_variable, (pd.Series, pl.Series)):
        raise ValueError("Target_variable is not a series (pandas or polars)")
    # test 3: if the number of rows in feature_matrix is not equal to the␣
↪length of target_variable, raise a ValueError
    if feature_matrix.shape[0] != len(target_variable):
        raise ValueError("The number of rows in feature_matrix is not equal to␣
↪the length of target_variable")
    # test 4: if k is not an integer, raise ValueError
    if not isinstance(k, int):
        raise ValueError("k is not an integer")
    # test 5: if shuffle is not a boolean, raise ValueError
    if not isinstance(shuffle, bool):
        raise ValueError("shuffle is not a boolean")
    # test 6: if random_state is not an integer, raise ValueError
    if not isinstance(random_state, int):
        raise ValueError("random_state is not an integer")
```

```
    # test 7: if k is less than 3 or larger than the number of rows in␣
↪feature_matrix, raise ValueError
    if k < 3 or k > feature_matrix.shape[0]:
        raise ValueError("k is less than 2 or larger than the number of rows in␣
↪feature_matrix")
    # test 8: if feature_matrix is empty, raise ValueError
    if feature_matrix.shape[0] == 0 or feature_matrix.shape[1] == 0:
        raise ValueError("feature_matrix is empty")
    # test 9: if target_variable is empty, raise ValueError
    if len(target_variable) == 0:
        raise ValueError("target_variable is empty")



    # ************************************
    # TODO: implement the splitting strategy
    # ************************************
    # you can use numpy, pandas or polars. do not use sklearn here!

    # Let number of samples be equal to number of rows
    n_samples = feature_matrix.shape[0]
    # Create index array
    indices = np.arange(n_samples)
    # Shuffle indices if needed
    if shuffle:
        # Create a random number generator with a fixed seed for reproducibility
        rng = np.random.default_rng(seed=random_state)
        # Shuffle the sample indices randomly
        rng.shuffle(indices)

    # Split indices into k folds
    # Assume n_samles can be divided by k. Then create list with length k and␣
↪each elements in the list means the length of each folds.
    fold_sizes = [n_samples // k] * k
    # Consider if n_samles cannot be divided by k, which means have remainder
    for i in range(n_samples % k):
        fold_sizes[i] += 1  # distribute remainder

    # Create list with length k and each elements in the list is the list of␣
↪index for each fold
    folds = []
    start = 0
    # Use for loop the save the list of index for each fold in 'folds'
    for size in fold_sizes:
        folds.append(indices[start:start + size])
        start += size
```

```
    # Create a random number generator with a fixed seed for reproducibility
    rng = np.random.default_rng(seed=random_state)
    # Randomly choose a index of folds
    test_fold_idx = rng.choice(len(folds))
    test_idx = folds[test_fold_idx]
    # Create X_test and y_test
    X_test = feature_matrix.iloc[test_idx]
    y_test = target_variable.iloc[test_idx]

    # Remaining folds for train/validation except test fold
    train_val_folds = [folds[i] for i in range(len(folds)) if i !=␣
↪test_fold_idx]

    # Prepare train_val_sets
    train_val_sets = []
    for i, val_idx in enumerate(train_val_folds):
        # Create validation set
        X_val = feature_matrix.iloc[val_idx]
        y_val = target_variable.iloc[val_idx]
        # training set is all other folds except validation set
        other_idx = np.hstack([train_val_folds[j] for j in␣
↪range(len(train_val_folds)) if j != i])
        X_train = feature_matrix.iloc[other_idx]
        y_train = target_variable.iloc[other_idx]
        # append tuple ((X_train, y_train), (X_val, y_val))
        train_val_sets.append(((X_train, y_train), (X_val, y_val)))


    # ***********************
    # TODO: test the outputs!
    # ***********************
    # test 1: check whether each point is in exactly one set (no point is␣
↪duplicated and no point is left out)
    # We check if the index is unique, becasue is unique.
    # All original sample indices
    all_indices = set(feature_matrix.index)
    # Indices in the test set
    test_indices = set(X_test.index)
    # Indices in all train and validation sets
    train_val_indices = set()
    for train_set, val_set in train_val_sets:
        X_train, y_train = train_set # get train features and labels
        X_val, y_val = val_set # get validation features and labels
        train_val_indices.update(X_train.index) # add train indices
        train_val_indices.update(X_val.index) # add validation indices
```

```
   # Check the intersection between test and train/val in order to check␣
↪duplicated
   if test_indices & train_val_indices:
       raise ValueError("Some points are duplicated between test and train/
↪validation sets.")

   # Check that all original samples are included through checking if the␣
↪dataframe index is included in index of test, train and val
   if all_indices != test_indices.union(train_val_indices):
       raise ValueError("Some points are missing from the output sets.")



   # test 2: check whether you preserve the row-wise alignment between the␣
↪feature matrix and the target variable in each set
   # hint: for row-wise alignment, it may be useful to note that row indices␣
↪do not change when subsetting dfs.
   # i.e. if a row is index id 10 in dataframe 1, it is still given the index␣
↪id of 10 in a subsetted df.
   # Loop over each train/validation fold
   for i, (train_set, val_set) in enumerate(train_val_sets):
       X_train, y_train = train_set # get features and labels for train set
       X_val, y_val = val_set # # get features and labels for validation set
       # Check that row indices match between features and labels in train set␣
↪(we check the index because the index is unique)
       if not X_train.index.equals(y_train.index):
           raise ValueError(f"Train set row alignment broken in fold {i+1}")
       # Check that row indices match between features and labels in␣
↪validation set
       if not X_val.index.equals(y_val.index):
           raise ValueError(f"Validation set row alignment broken in fold␣
↪{i+1}")
   # Check that row indices match between features and labels in test set
   if not X_test.index.equals(y_test.index):
       raise ValueError("Test set row alignment broken!")



   # test 3: check whether the order of the columns is the same in each set.
   # We don't check y because y only has 1 column
   # Check train and validation for X
   # Loop over each train/validation fold
   for i, (train_set, val_set) in enumerate(train_val_sets):
       X_train, _ = train_set # get feature matrix from train set
       X_val, _ = val_set # get feature matrix from validation set
       # Check if train columns match original feature matrix
```

```
        if not X_train.columns.equals(feature_matrix.columns):
            raise ValueError(f"Train set column order broken in fold {i+1}")
        # Check if validation columns match original feature matrix
        if not X_val.columns.equals(feature_matrix.columns):
            raise ValueError(f"Validation set column order broken in fold␣
↪{i+1}")

    # Check Test set
    if not X_test.columns.equals(feature_matrix.columns):
        raise ValueError("Test set column order broken!")



    # test 4: perform output tests similar to those in basic_split
    # Check if X_test is Dataframe
    if not isinstance (X_test, (pd.DataFrame,pl.DataFrame)):
        raise ValueError('The output X_test is not a Dataframe')
    # Check if y-test is Series
    if not isinstance(y_test, (pd.Series, pl.Series)):
        raise ValueError('The output y_test is not a Series')
    # Check if train_val_sets is List
    if not isinstance(train_val_sets, list):
        raise ValueError('The output train_val_sets is not a List')


    return X_test, y_test, train_val_sets




##################################################
# TODO: call the function and preform tests here
##################################################
# test 1: Apply the function to the house price dataset from problem 1 with 4␣
↪folds.
X = df.drop(columns=['SalePrice'])
y = df['SalePrice']
X_test, y_test, train_val_sets = basic_kfold(X,y,k = 4)

# test 2: Print out the head of the sets.
# print the head of test sets
print('Head of X_test:')
print(X_test.head())
print('Head of y_test')
print(y_test.head())
# make for loop to print the head of training and validation sets for every fold
for fold_idx, (train_set, val_set) in enumerate(train_val_sets):
```

```
    # Defind X_train ,y_train, X_val and y_val
    X_train, y_train = train_set
    X_val, y_val = val_set

    print(f"=== Fold {fold_idx+1} ===")

    print("X_train head:")
    print(X_train.head())

    print("y_train head:")
    print(y_train.head())

    print("X_val head:")
    print(X_val.head())

    print("y_val head:")
    print(y_val.head())

    print("\n")


# test 3: Make sure that you get the same points in each set every time you␣
 ↪rerun the cell (a.k.a., test for reproducability).
X_test1, y_test1, train_val_sets1 = basic_kfold(X, y)
X_test2, y_test2, train_val_sets2 = basic_kfold(X, y)

print('Checking reproducibility:')
print('If X_test is the same:', X_test1.equals(X_test2))
print('If y_test is the same:', y_test1.equals(y_test2))
# Loop over each fold in two separate runs of k-fold to check reproducibility
for i, ((train1, val1), (train2, val2)) in enumerate(zip(train_val_sets1,␣
 ↪train_val_sets2)):
    X_train1, y_train1 = train1
    X_val1, y_val1 = val1
    X_train2, y_train2 = train2
    X_val2, y_val2 = val2
    if not X_train1.index.equals(X_train2.index) or not y_train1.index.
 ↪equals(y_train2.index):
        raise ValueError(f"Fold {i+1} train indices differ!")
    if not X_val1.index.equals(X_val2.index) or not y_val1.index.equals(y_val2.
 ↪index):
        raise ValueError(f"Fold {i+1} validation indices differ!")
print("All train/validation fold indices are same.")


# test 4: Try a couple of other `k` values. test the extreme values!
X_test_extreme, y_test_extreme, train_val_sets_extreme = basic_kfold(X,y,k = 3)
```

```
X_test_extreme2, y_test_extreme2, train_val_sets_extreme2 = basic_kfold(X,y,k =␣
↪len(y))
```

```
Head of X_test:
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
1257          30       RL         56.0     4060   Pave   NaN      Reg
612           60       RL          NaN    11885   Pave   NaN      Reg
1270          40       RL          NaN    23595   Pave   NaN      Reg
1061          30  C (all)        120.0    18000   Grvl   NaN      Reg
184           50       RL         92.0     7438   Pave   NaN      IR1

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC  Fence  \
1257         Lvl    AllPub    Corner  …           0        0    NaN    NaN
612          Lvl    AllPub    Inside  …           0        0    NaN    NaN
1270         Low    AllPub    Inside  …           0        0    NaN    NaN
1061         Low    AllPub    Inside  …           0        0    NaN    NaN
184          Lvl    AllPub    Inside  …           0        0    NaN  MnPrv

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
1257         NaN       0       7    2009        WD         Normal
612          NaN       0      11    2009        WD         Normal
1270         NaN       0       4    2010        WD         Normal
1061        Shed     560       8    2008     ConLD         Normal
184          NaN       0       6    2006        WD         Normal

[5 rows x 79 columns]
Head of y_test
1257     99900
612     261500
1270    260000
1061     81000
184     127000
Name: SalePrice, dtype: int64
=== Fold 1 ===
X_train head:
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
541           60       RL          NaN    11000   Pave   NaN      Reg
341           20       RH         60.0     8400   Pave   NaN      Reg
1074          20       RL         74.0     8556   Pave   NaN      Reg
521           20       RL         90.0    11988   Pave   NaN      IR1
850          120       RM         36.0     4435   Pave   NaN      Reg

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC Fence  \
541          Lvl    AllPub       FR2  …           0        0    NaN   NaN
341          Lvl    AllPub    Inside  …           0        0    NaN   NaN
1074         Lvl    AllPub    Inside  …           0        0    NaN   NaN
521          Lvl    AllPub    Corner  …           0        0    NaN   NaN
```

```
850          Lvl    AllPub    Inside  …           0        0    NaN   NaN

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
541          NaN       0       6    2007        WD         Normal
341          NaN       0       9    2009        WD         Normal
1074         NaN       0       5    2007        WD         Normal
521          NaN       0       5    2007        WD         Normal
850          NaN       0      11    2007        WD         Normal

[5 rows x 79 columns]
y_train head:
541     248000
341      82000
1074    194000
521     150000
850     131500
Name: SalePrice, dtype: int64
X_val head:
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
895           60       RL         71.0     7056   Pave   NaN      Reg
1414          50       RL         64.0    13053   Pave  Pave      Reg
25            20       RL        110.0    14230   Pave   NaN      Reg
0             60       RL         65.0     8450   Pave   NaN      Reg
959          160       FV         24.0     2572   Pave   NaN      Reg

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC Fence  \
895          Lvl    AllPub    Inside  …           0        0    NaN   NaN
1414         Bnk    AllPub    Inside  …         220        0    NaN   NaN
25           Lvl    AllPub    Corner  …           0        0    NaN   NaN
0            Lvl    AllPub    Inside  …           0        0    NaN   NaN
959          Lvl    AllPub       FR2  …           0        0    NaN   NaN

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
895          NaN       0      10    2008        WD         Normal
1414         NaN       0       6    2008        WD         Normal
25           NaN       0       7    2009        WD         Normal
0            NaN       0       2    2008        WD         Normal
959          NaN       0       5    2010        WD         Normal

[5 rows x 79 columns]
y_val head:
895     140000
1414    207000
25      256300
0       208500
959     155000
Name: SalePrice, dtype: int64
```

```
=== Fold 2 ===
X_train head:
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
895           60       RL         71.0     7056   Pave   NaN      Reg
1414          50       RL         64.0    13053   Pave  Pave      Reg
25            20       RL        110.0    14230   Pave   NaN      Reg
0             60       RL         65.0     8450   Pave   NaN      Reg
959          160       FV         24.0     2572   Pave   NaN      Reg

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC Fence  \
895          Lvl    AllPub    Inside  …           0        0    NaN   NaN
1414         Bnk    AllPub    Inside  …         220        0    NaN   NaN
25           Lvl    AllPub    Corner  …           0        0    NaN   NaN
0            Lvl    AllPub    Inside  …           0        0    NaN   NaN
959          Lvl    AllPub       FR2  …           0        0    NaN   NaN

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
895          NaN       0      10    2008        WD         Normal
1414         NaN       0       6    2008        WD         Normal
25           NaN       0       7    2009        WD         Normal
0            NaN       0       2    2008        WD         Normal
959          NaN       0       5    2010        WD         Normal

[5 rows x 79 columns]
y_train head:
895     140000
1414    207000
25      256300
0       208500
959     155000
Name: SalePrice, dtype: int64
X_val head:
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
541           60       RL          NaN    11000   Pave   NaN      Reg
341           20       RH         60.0     8400   Pave   NaN      Reg
1074          20       RL         74.0     8556   Pave   NaN      Reg
521           20       RL         90.0    11988   Pave   NaN      IR1
850          120       RM         36.0     4435   Pave   NaN      Reg

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC Fence  \
541          Lvl    AllPub       FR2  …           0        0    NaN   NaN
341          Lvl    AllPub    Inside  …           0        0    NaN   NaN
1074         Lvl    AllPub    Inside  …           0        0    NaN   NaN
521          Lvl    AllPub    Corner  …           0        0    NaN   NaN
850          Lvl    AllPub    Inside  …           0        0    NaN   NaN

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
```

```
541          NaN       0       6    2007        WD         Normal
341          NaN       0       9    2009        WD         Normal
1074         NaN       0       5    2007        WD         Normal
521          NaN       0       5    2007        WD         Normal
850          NaN       0      11    2007        WD         Normal

[5 rows x 79 columns]
y_val head:
541     248000
341      82000
1074    194000
521     150000
850     131500
Name: SalePrice, dtype: int64


=== Fold 3 ===
X_train head:
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
895           60       RL         71.0     7056   Pave   NaN      Reg
1414          50       RL         64.0    13053   Pave  Pave      Reg
25            20       RL        110.0    14230   Pave   NaN      Reg
0             60       RL         65.0     8450   Pave   NaN      Reg
959          160       FV         24.0     2572   Pave   NaN      Reg

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC Fence  \
895          Lvl    AllPub    Inside  …           0        0    NaN   NaN
1414         Bnk    AllPub    Inside  …         220        0    NaN   NaN
25           Lvl    AllPub    Corner  …           0        0    NaN   NaN
0            Lvl    AllPub    Inside  …           0        0    NaN   NaN
959          Lvl    AllPub       FR2  …           0        0    NaN   NaN

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
895          NaN       0      10    2008        WD         Normal
1414         NaN       0       6    2008        WD         Normal
25           NaN       0       7    2009        WD         Normal
0            NaN       0       2    2008        WD         Normal
959          NaN       0       5    2010        WD         Normal

[5 rows x 79 columns]
y_train head:
895     140000
1414    207000
25      256300
0       208500
959     155000
Name: SalePrice, dtype: int64
X_val head:
```

```
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
1441         120       RM          NaN     4426   Pave   NaN      Reg
1217          20       FV         72.0     8640   Pave   NaN      Reg
869           60       RL         80.0     9938   Pave   NaN      Reg
855           20       RL          NaN     6897   Pave   NaN      IR1
178           20       RL         63.0    17423   Pave   NaN      IR1

     LandContour Utilities LotConfig  … ScreenPorch PoolArea PoolQC  Fence  \
1441         Lvl    AllPub    Inside  …           0        0    NaN    NaN
1217         Lvl    AllPub    Inside  …           0        0    NaN    NaN
869          Lvl    AllPub    Inside  …           0        0    NaN  GdPrv
855          Lvl    AllPub    Corner  …           0        0    NaN    NaN
178          Lvl    AllPub   CulDSac  …           0        0    NaN    NaN

     MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition
1441         NaN       0       5    2008        WD         Normal
1217         NaN       0       9    2009       New        Partial
869          NaN       0       6    2010        WD         Normal
855          NaN       0       4    2010        WD         Normal
178          NaN       0       7    2009       New        Partial

[5 rows x 79 columns]
y_val head:
1441    149300
1217    229456
869     236000
855     127000
178     501837
Name: SalePrice, dtype: int64


Checking reproducibility:
If X_test is the same: True
If y_test is the same: True
All train/validation fold indices are same.
```