Presentation on Houses Price

Team Project

Artur
Yu Dzhin
Shabdan

This dataset was used in data science competitions in Kaggle

The House Price dataset is a well-known dataset used in machine learning and data analysis. The dataset contains information on residential homes in Ames, Iowa, and includes various features such as the sale price, location, size, age, and other important characteristics of the property. This dataset is often used as a benchmark in machine learning competitions and is frequently used by researchers in the field of real estate analysis.

The dataset was compiled by Dean De Cock in 2011 and was intended to be an improvement over similar datasets that were available at the time. It includes 80 explanatory variables that describe various aspects of residential homes in Ames, Iowa, and a target variable, which is the sale price of each home. The dataset is relatively large, containing 1,460 observations, and is commonly used for regression and classification tasks.

The House Price dataset is a valuable resource for machine learning practitioners, data analysts, and researchers interested in real estate analysis. Its widespread use and popularity make it an ideal starting point for exploring the application of machine learning techniques to real-world problems.

Dataset	Step 1 Data analysis	Step 2 Data preprocessing & Feature engineering	Step 3 Feature Selection	Step 4 Pipeline			Step 5 Metric score	Step 6 Conclusion
				Stacks of techniques	Model	Determine best parameters		

HousePrice.csv	• Overview of dataset (Column description)
	• Types of variables
	• Make classification
	• Missing values
	• Distribution and outliers
	• Cardinality
	• Splitting dataset on train and test sets
		• Create new features
		• Missing data imputation
		• Encoding
		• Discretization
		• Feature-magnitude
		• Feature Scaling
		• Standardization
		• Unit norm
			• Constant & Quasi-constant features
			• Duplicated features
			• Correlated features
				• CategoricalImputer
				  • frequent
				  • missing
				• MeanMedianImputer
				• RandomSampleImputer
				• RareLabelEncoder
				• OneHotEncoder
				• MeanEncoder
				• EqualFrequencyDiscretiser
				• DropConstantFeatures
				• DropDuplicateFeatures
				• SmartCorrelatedSelection
					Decision tree classifier	GridSearchCV	• Roc-auc score
							• Accuracy score
							• Standart deviation
							• Classification report
							• Confusion matrix
							• Precision & Recall curve
								• Choose the best model
								• Research report (creating .ppt file)

				• CategoricalImputer
				  • frequent
				  • missing
				• MeanMedianImputer
				• RandomSampleImputer
				• RareLabelEncoder
				• OneHotEncoder
				• MeanEncoder
				• EqualFrequencyDiscretiser
				• DropConstantFeatures
				• DropDuplicateFeatures
				• SmartCorrelatedSelection
					Random forest classifier	GridSearchCV		

				• CategoricalImputer
				  • frequent
				  • missing
				• MeanMedianImputer
				• RandomSampleImputer
				• RareLabelEncoder
				• OneHotEncoder
				• MeanEncoder
				• EqualFrequencyDiscretiser
				• DropConstantFeatures
				• DropDuplicateFeatures
				• SmartCorrelatedSelection
					Support vector classifier	GridSearchCV		

				• CategoricalImputer
				  • frequent
				  • missing
				• MeanMedianImputer
				• RandomSampleImputer
				• RareLabelEncoder
				• OneHotEncoder
				• MeanEncoder
				• EqualFrequencyDiscretiser
				• DropConstantFeatures
				• DropDuplicateFeatures
				• SmartCorrelatedSelection
					Logistic regression	GridSearchCV		

				• CategoricalImputer
				  • frequent
				  • missing
				• MeanMedianImputer
				• RandomSampleImputer
				• RareLabelEncoder
				• OneHotEncoder
				• MeanEncoder
				• EqualFrequencyDiscretiser
				• DropConstantFeatures
				• DropDuplicateFeatures
				• SmartCorrelatedSelection
					XGBoost Classifier	GridSearchCV		

				• CategoricalImputer
				  • frequent
				  • missing
				• MeanMedianImputer
				• RandomSampleImputer
				• RareLabelEncoder
				• OneHotEncoder
				• MeanEncoder
				• EqualFrequencyDiscretiser
				• DropConstantFeatures
				• DropDuplicateFeatures
				• SmartCorrelatedSelection
					AdaBoost Classifier	GridSearchCV		

Import Basic Libraries:

Pandas ,
numpy,
matplotlib.pyplot,
seaborn,
train_test_split

Import Imputation Techniques

Random Sample Imputer
Mean Median Imputer
Arbitrary Number Imputer
Categorical Imputer

Import Encodin Techniques

PRationEncoder
OneHotEncoder
WoEncoder
MeanEncoder
OrdinalEncoder
RareLabelEncoder

Discretization Techniques

EqualWidthDiscretization
EqualFrequencyDiscretizer

Classifiers

Decision Tree Classifier
Random Forest Classifier
Ada Boost Classifier
SVC
Logistic Regression
XGBClassifier

Metrics

roc_auc_score
confusion_matrix,
PrecisionRecallDisplay
accuracy_score
classification Report

Compose Element

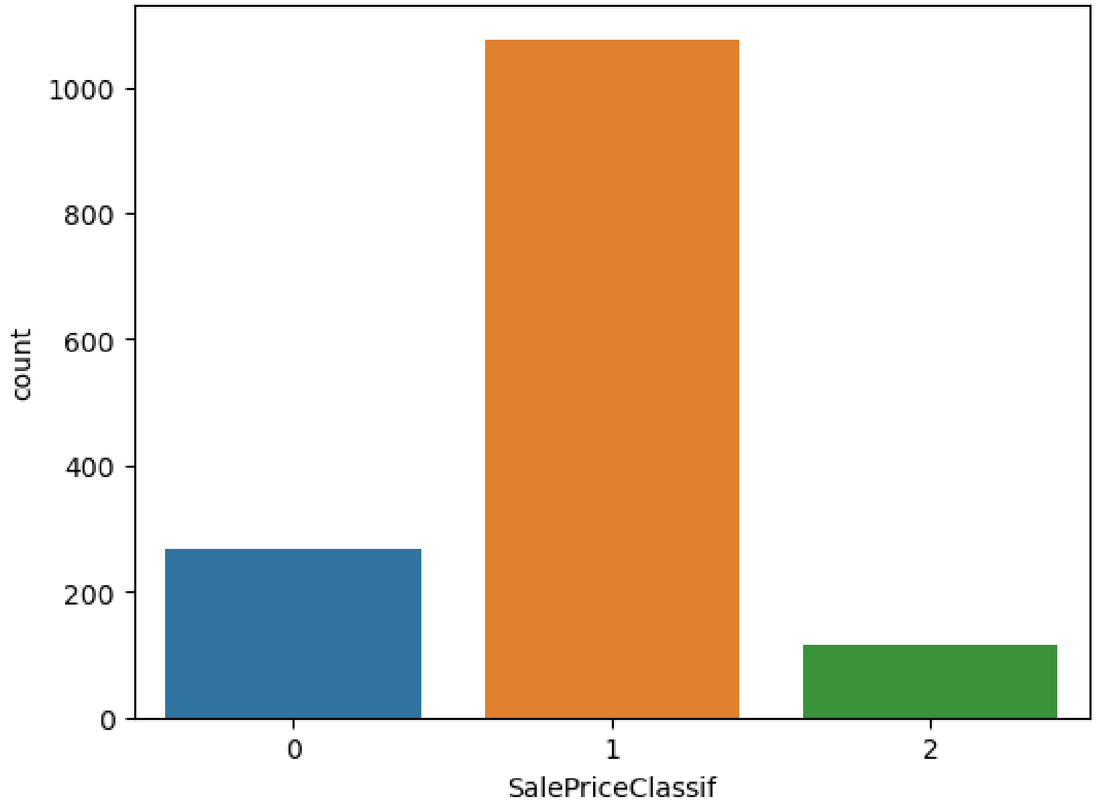
GridSearchCV
cross_val_score
PipeLine

Feature Magnitude

Standard Scaler
Normalizer
RobustScaler

Feature Selection

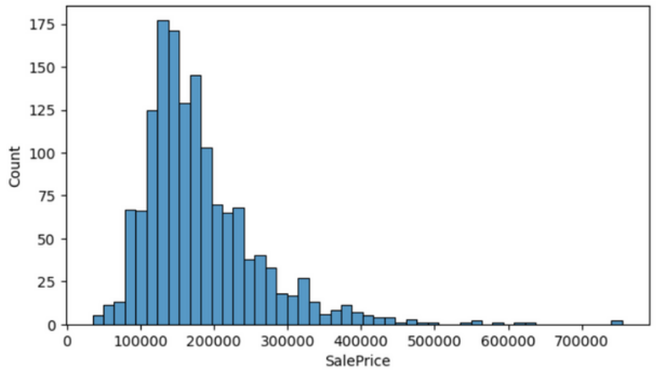
mutual_info_classif
mutual_info_regression
SelectKBest
SelectPrecentile
DropDuplicatedFeatures
DropConstantFeatures
SmartCorrealtedSelection

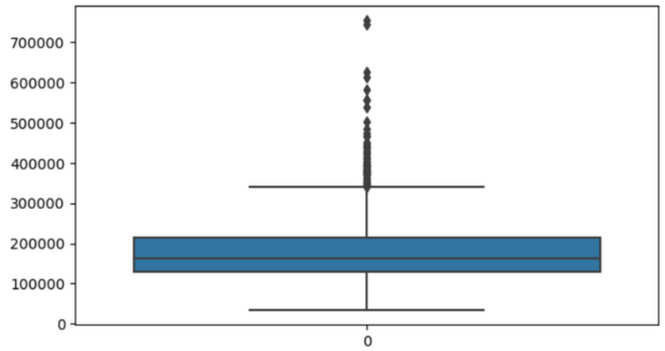


Make Classification

3.2 Make classification

```
fig = plt.figure(figsize = (16, 4))

ax = fig.add_subplot(1, 2, 1)
sns.histplot(data['SalePrice'])

ax = fig.add_subplot(1, 2, 2)

sns.boxplot(data = data['SalePrice'])
plt.show()
```





Checking Missing Values

```
for var in data.columns:
    if data[var].isnull().sum() > 0:
        print(var, ' Percentage of missing labels: {:.3f} % \n' .format(data[var].isnull().mean() * 100))
```

```
LotFrontage  Percentage of missing labels: 17.740 %

Alley  Percentage of missing labels: 93.767 %

MasVnrType  Percentage of missing labels: 0.548 %

MasVnrArea  Percentage of missing labels: 0.548 %

BsmtQual  Percentage of missing labels: 2.534 %

BsmtCond  Percentage of missing labels: 2.534 %

BsmtExposure  Percentage of missing labels: 2.603 %

BsmtFinType1  Percentage of missing labels: 2.534 %

BsmtFinType2  Percentage of missing labels: 2.603 %

Electrical  Percentage of missing labels: 0.068 %

FireplaceQu  Percentage of missing labels: 47.260 %

GarageType  Percentage of missing labels: 5.548 %

GarageYrBlt  Percentage of missing labels: 5.548 %

GarageFinish  Percentage of missing labels: 5.548 %

GarageQual  Percentage of missing labels: 5.548 %

GarageCond  Percentage of missing labels: 5.548 %

PoolQC  Percentage of missing labels: 99.521 %

Fence  Percentage of missing labels: 80.753 %
```

122]:

```
# Determine variables with high percentage of missing observations
drop_features_missing = [
    var for var in data.columns if data[var].isnull().mean() > 0.5 and var != 'SalePrice'
]

print('Count of missing values \n')
for var in drop_features_missing:
    print(var, ': ', data[var].isnull().sum())
```

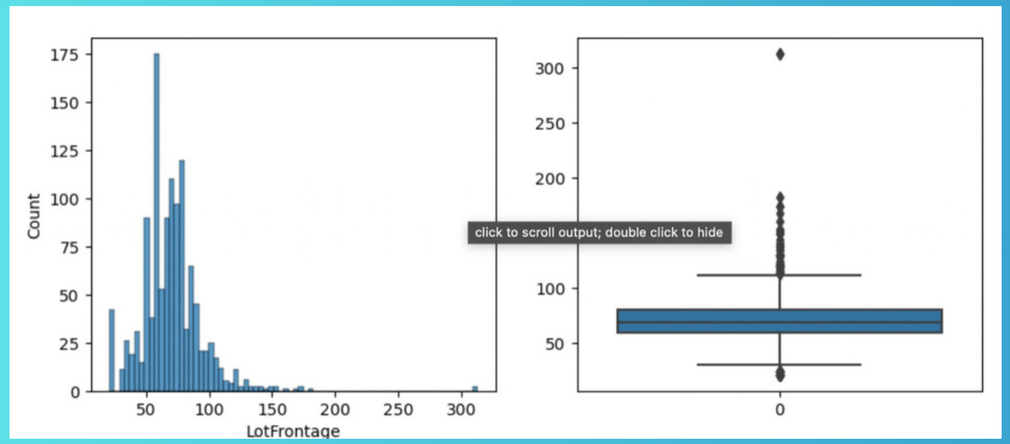
```
Count of missing values

Alley :  1369
PoolQC :  1453
Fence :  1179
MiscFeature :  1406
```

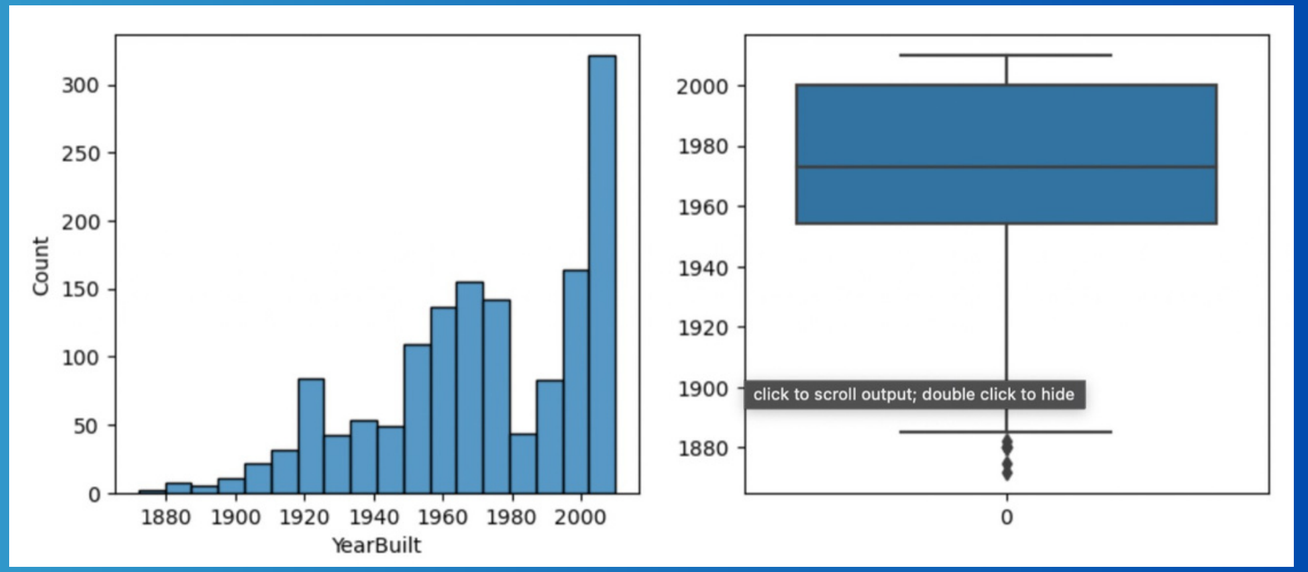
Drop variables with a lot of missing observations

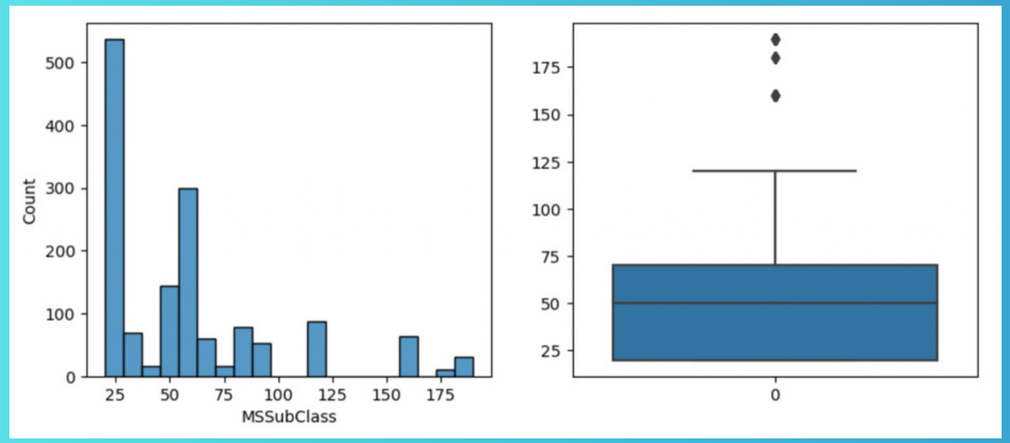
variables with a lot of missing observations do not give us enough information

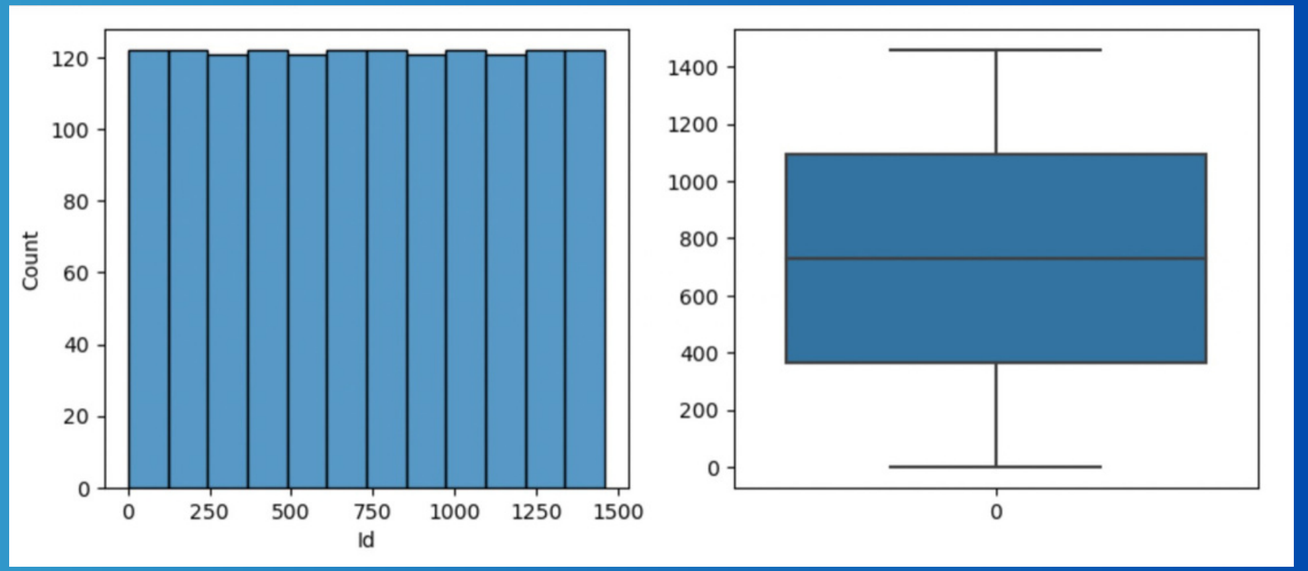
3.4 Distribution and outliers

```
for var in continuous_features_data:
    sns.histplot(data[var])
    plt.show()
```









4. Feature-engineering

```
# Splitting dataset on train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    data.drop(labels = 'SalePriceClassif', axis = 1),
    data['SalePriceClassif'],
    test_size = 0.3
)

X_train.shape, X_test.shape
```

```
((1022, 75), (438, 75))
```

```
X_train.head(10)
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	B
453	60	FV	75.0	9000	Pave	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	Norm	
987	20	RL	83.0	10159	Pave	IR1	Lvl	AllPub	Inside	Gtl	NridgHt	Norm	Norm	
715	20	RL	78.0	10140	Pave	Reg	Lvl	AllPub	Inside	Gtl	NWAmes	Norm	Norm	
1041	60	RL	NaN	9130	Pave	Reg	Lvl	AllPub	Inside	Gtl	NWAmes	Feedr	Norm	
402	30	RL	60.0	10200	Pave	Reg	Lvl	AllPub	Inside	Gtl	Sawyer	Norm	Norm	
741	20	RL	65.0	6768	Pave	IR1	Lvl	AllPub	Inside	Gtl	Sawyer	Feedr	Norm	
246	190	RM	69.0	9142	Pave	Reg	Lvl	AllPub	Inside	Gtl	OldTown	Norm	Norm	
1230	90	RL	NaN	18890	Pave	IR1	Lvl	AllPub	Inside	Gtl	Sawyer	Feedr	RRAe	
229	120	RL	43.0	3182	Pave	Reg	Lvl	AllPub	Inside	Gtl	Blmngtn	Norm	Norm	
1448	50	RL	70.0	11767	Pave	Reg	Lvl	AllPub	Inside	Gtl	Edwards	Norm	Norm	

Feature Engineering and Data Separation to X_train and X_test sets

4.1 Create new features

```
X_train = dateTimeHandling(X_train)
X_test = dateTimeHandling(X_test)

X_train.shape, X_test.shape
```

```
((1022, 74), (438, 74))
```

```
categorical_features_train = [
    var for var in X_train.columns if X_train[var].dtype == 'O'
]

numerical_features_train = [
    var for var in X_train.columns if X_train[var].dtype != 'O' and var not in categorical_features_train
]

len(numerical_features_train), len(categorical_features_train)
```

```
(35, 39)
```

Create the new variables (function "dateTimeHandling")

Count of Categorical and Numerical features

4.2 Missing data imputation

```
need_impute = [
    var for var in X_train.columns if X_train[var].isnull().mean() > 0.0
]

print('Count of features with missing labels: ', len(need_impute))
```

```
Count of features with missing labels:  15
```

```
for var in need_impute:
    print(var)
    print('Percentage of missing variables: {:.3f}%' .format(X_train[var].isnull().sum() / len(data) * 100))
    print()
```

```
LotFrontage
Percentage of missing variables: 12.329%

MasVnrType
Percentage of missing variables: 0.274%

MasVnrArea
Percentage of missing variables: 0.274%

BsmtQual
Percentage of missing variables: 1.575%

BsmtCond
Percentage of missing variables: 1.575%

BsmtExposure
Percentage of missing variables: 1.575%

BsmtFinType1
Percentage of missing variables: 1.575%

BsmtFinType2
Percentage of missing variables: 1.644%

Electrical
Percentage of missing variables: 0.068%

FireplaceQu
Percentage of missing variables: 33.767%

GarageType
Percentage of missing variables: 3.699%

GarageYrBlt
Percentage of missing variables: 3.699%

GarageFinish
Percentage of missing variables: 3.699%

GarageQual
Percentage of missing variables: 3.699%

GarageCond
Percentage of missing variables: 3.699%
```

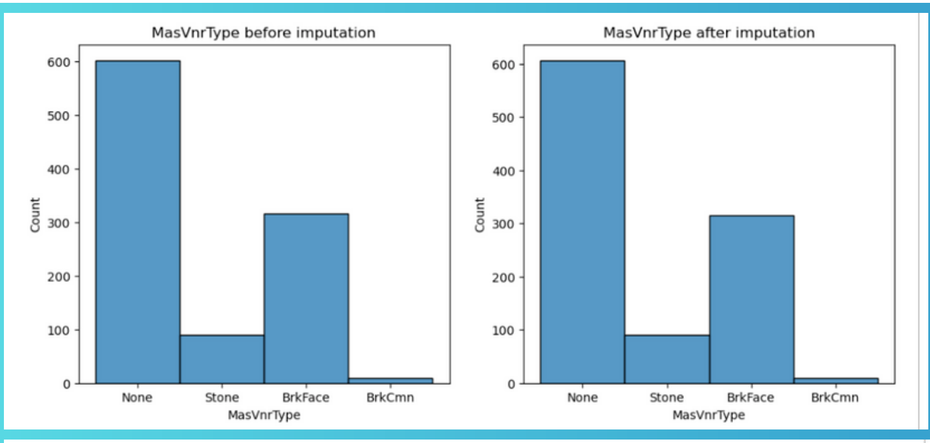
Categorical imputation methods

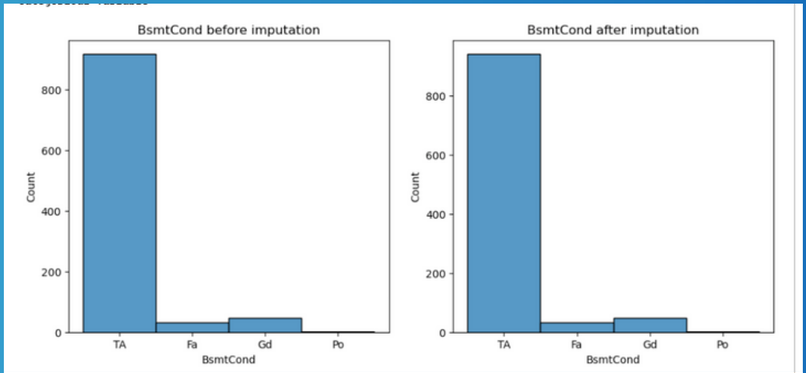
Frequent imputation

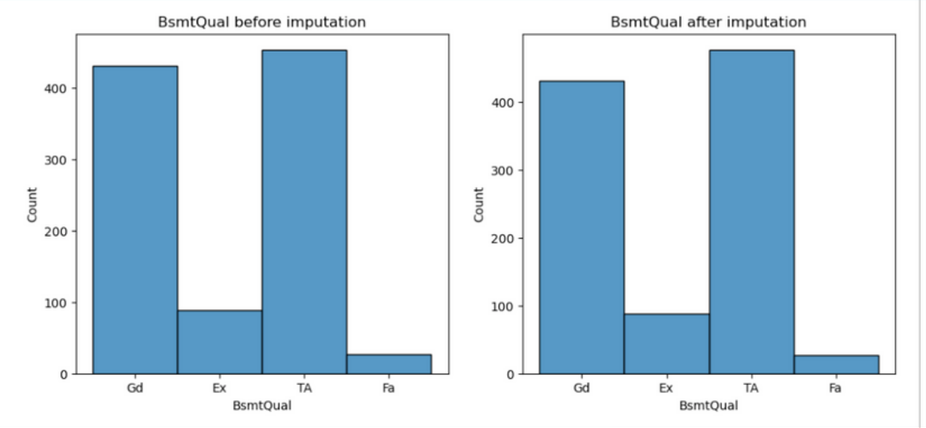
```
imputer_frequency = CategoricalImputer(imputation_method = 'frequent', variables = need_impute_cat)

imputer_frequency.fit(X_train)

X_train_impute_frequent = imputer_frequency.transform(X_train)

X_test_impute_frequent = imputer_frequency.transform(X_test)
```

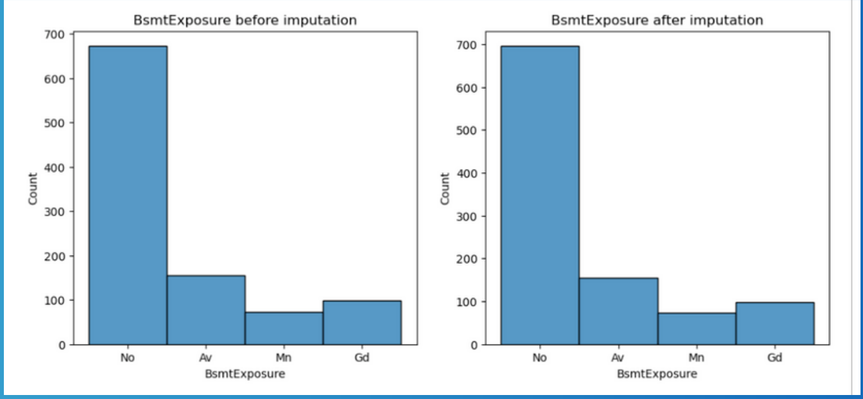
```
# Distribution in train set after frequent imputation
for var in need_impute_cat:
    check_dist_and_variance(X_train, X_train_impute_frequent, var)
```

Frequent Imputation Method







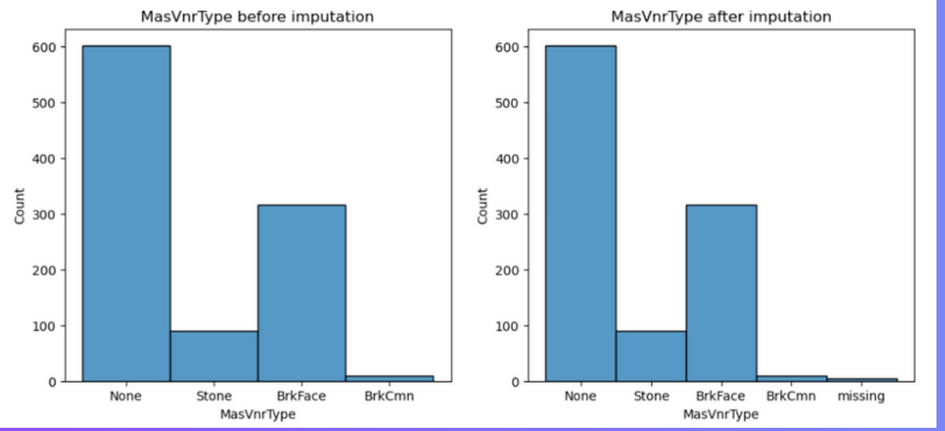


Missing labels imputation

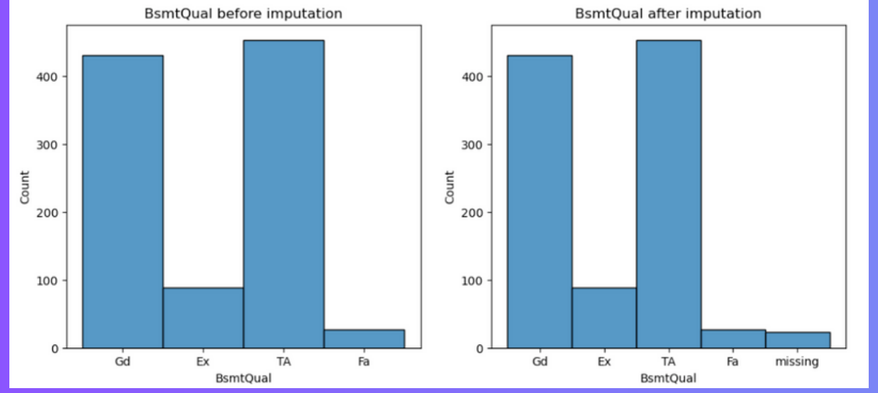
```
imputer_miss_labels = CategoricalImputer(
    imputation_method = 'missing',
    fill_value = 'missing',
    variables = need_impute_cat
)

imputer_miss_labels.fit(X_train)

X_train_impute_miss_labels = imputer_miss_labels.transform(X_train)

X_test_impute_miss_labels = imputer_miss_labels.transform(X_test)
```

```
# Check distribution in train set after missing_labels imputation
for var in need_impute_cat:
    check_dist_and_variance(X_train, X_train_impute_miss_labels, var)
```

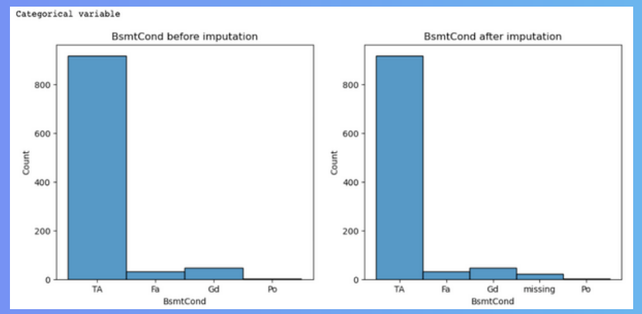
Categorical variable



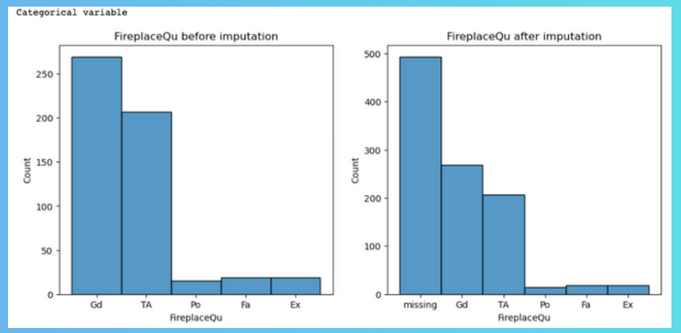
Categorical variable



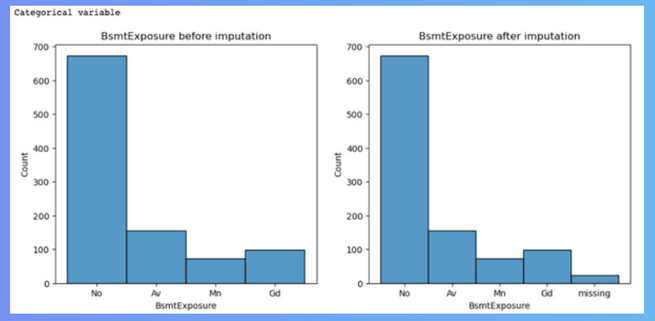
Categorical variable



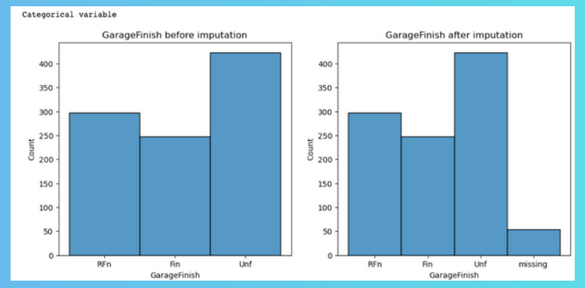
Categorical variable



Categorical variable

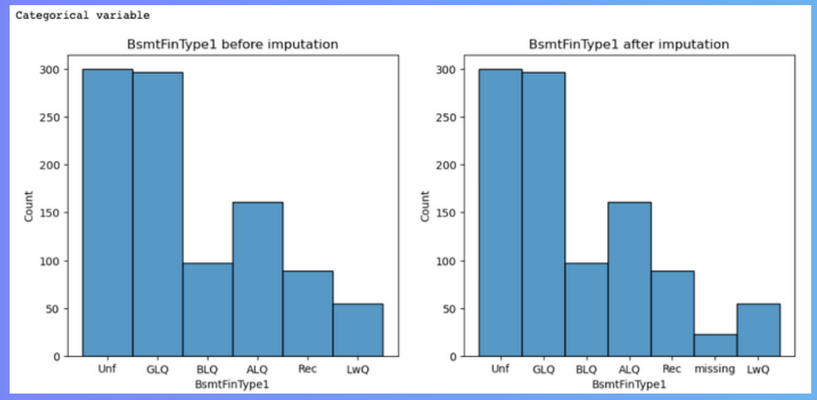


Categorical variable

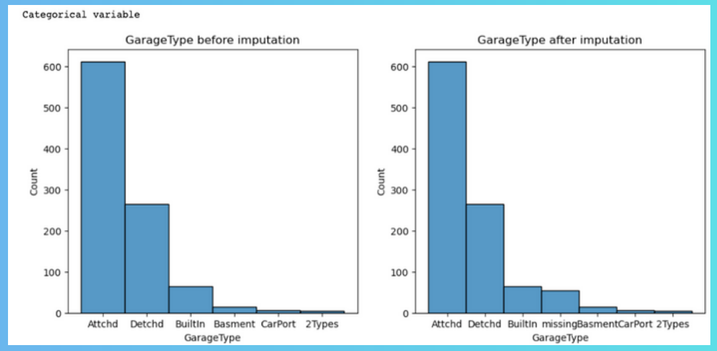


Missing Label Imputation

Categorical variable



Categorical variable



Numerical imputation methods

Arbitrary number imputer (arbitrary number = -1)

```
imputer_arbitrary = ArbitraryNumberImputer(variables = need_impute_num, arbitrary_number = -1)

imputer_arbitrary.fit(X_train, y_train)

X_train_impute_arb_minus = imputer_arbitrary.transform(X_train)

X_test_impute_ard_minus = imputer_arbitrary.transform(X_test)
```

```
# Check distribution after arbitrary number imputation (arbitrary value = -1)
for var in need_impute_num:

    check_dist_and_variance(X_train, X_train_impute_arb_minus, var)
```

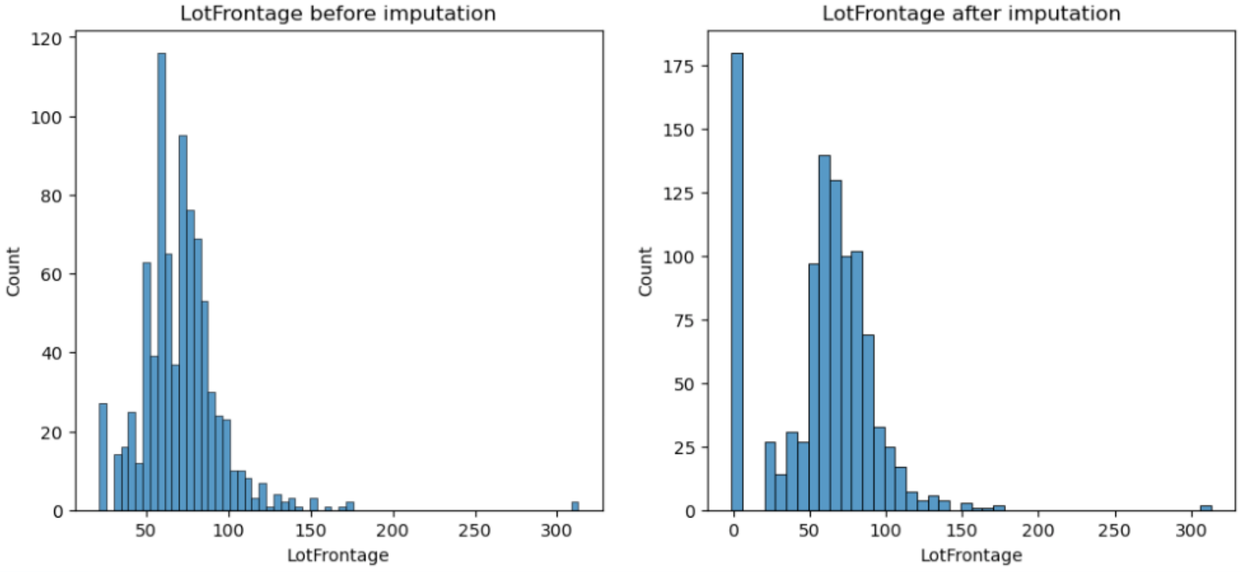
Numerical imputation Methods

Arbitrary Number Imputer (arbitrary number = -1)

Check Distribution after arbitrary number imputation (arbitrary value = -1)

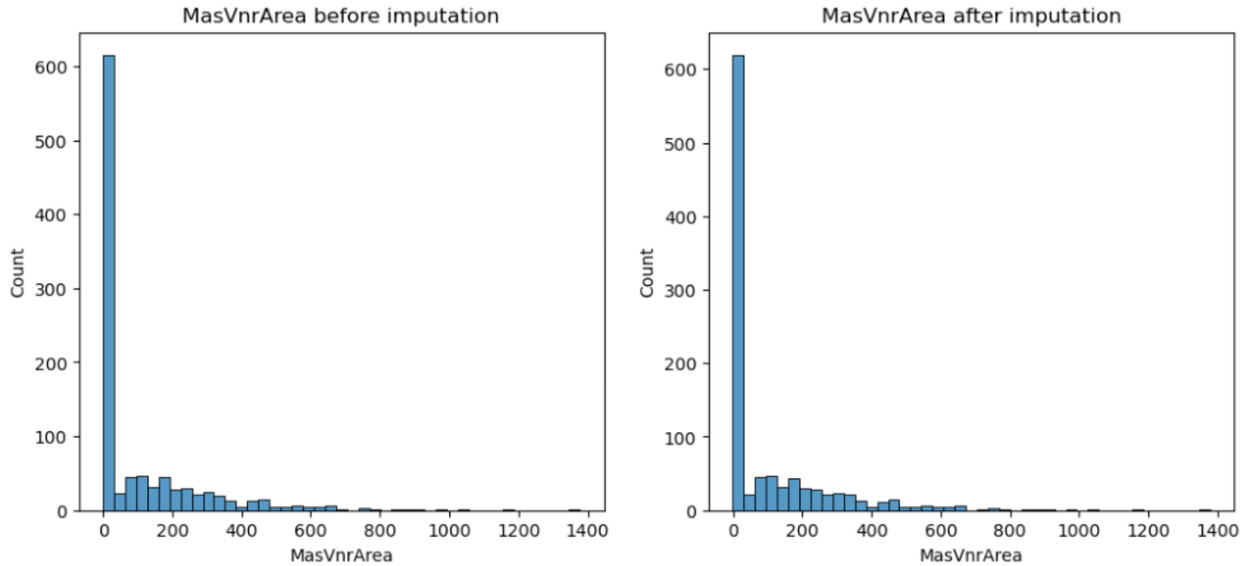
Variance before imputation: 631.2577620805455

Variance after imputation: 1258.8131901305462



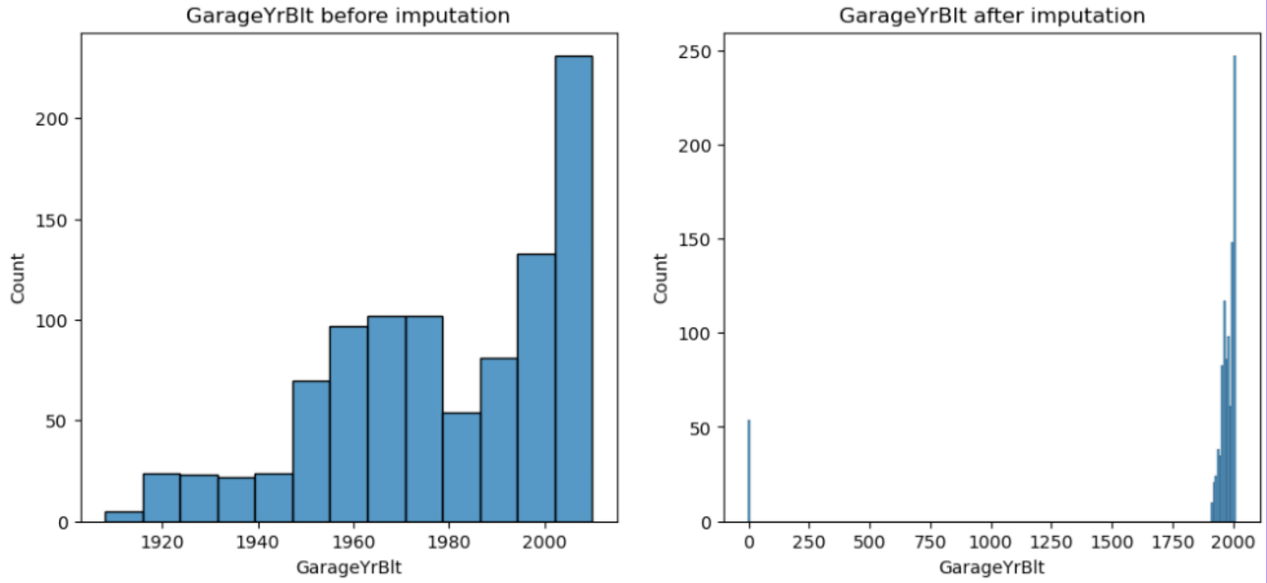
Variance before imputation: 31981.573954946656

Variance after imputation: 31899.081657980838



Variance before imputation: 601.6565750339723

Variance after imputation: 196871.45803201268



Arbitrary Number Imputer (arbitrary number = -1)

Arbitrary imputation (arbitrary number = 99)

```
imputer_arbitrary_nn = ArbitraryNumberImputer(variables = need_impute_num, arbitrary_number = 99)

imputer_arbitrary_nn.fit(X_train, y_train)

X_train_impute_arb_nn = imputer_arbitrary_nn.transform(X_train)

X_test_impute_ard_nn = imputer_arbitrary_nn.transform(X_test)
```

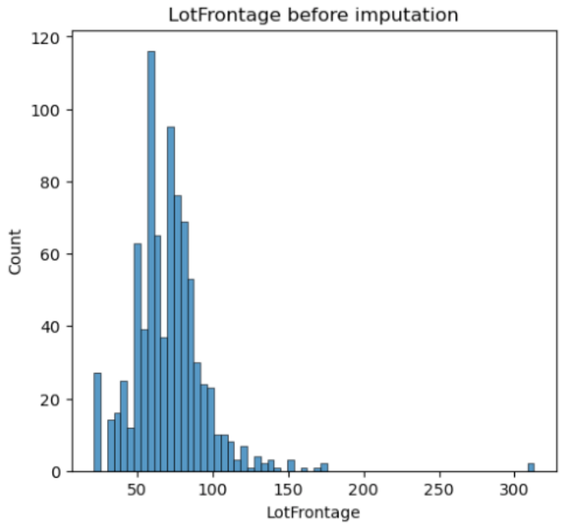
```
# Check distribution in train set after arbitrary number imputation (arbitrary value = 99)

for var in need_impute_num:
    check_dist_and_variance(X_train, X_train_impute_arb_nn, var)
```

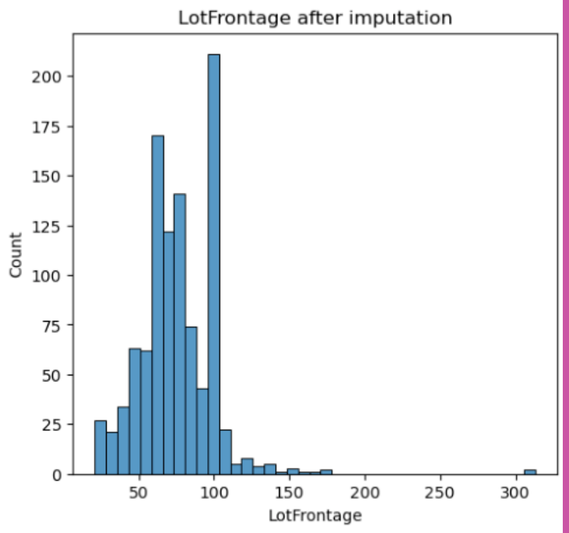
Arbitrary Imputation (arbitrary number = 99)

Check Distribution in train set after arbitrary number imputation (arbitrary value = 99)

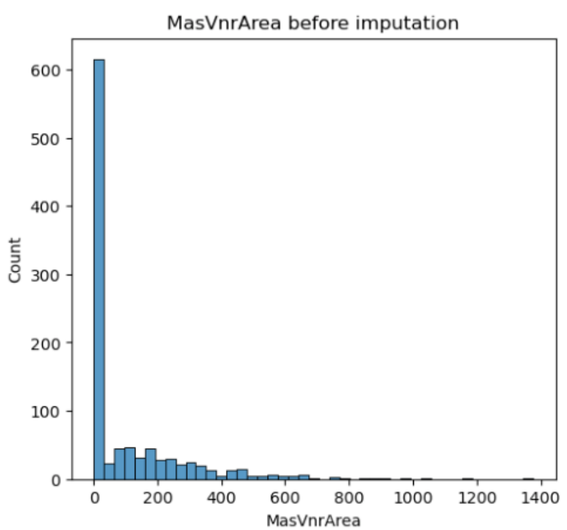
Arbitrary Imputation (arbitrary number = 99)

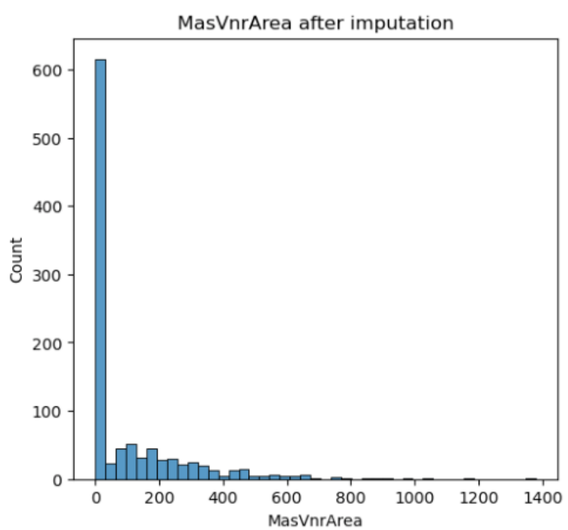
```
Variance before imputation: 631.2577620805455

Variance after imputation: 639.4250379985089
```



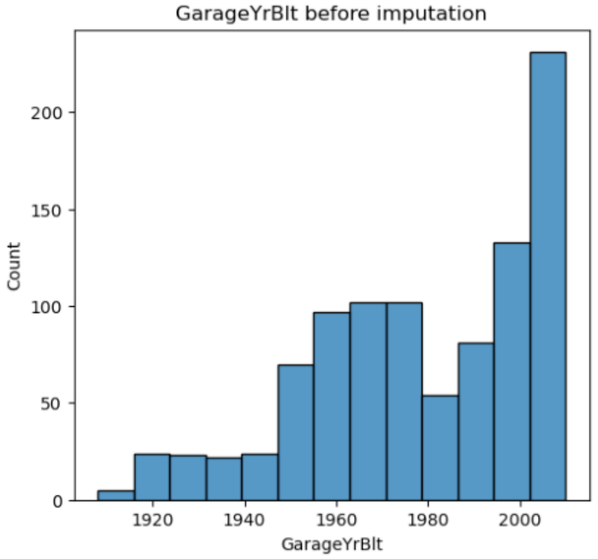


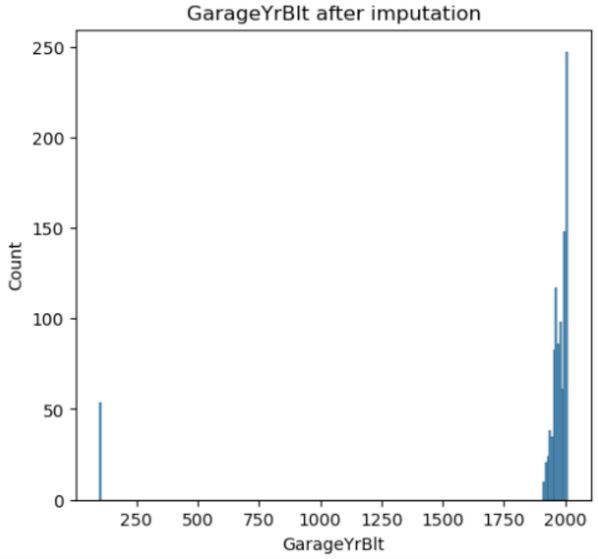

```
Variance before imputation: 31981.573954946656

Variance after imputation: 31856.366158997647
```






```
Variance after imputation: 177539.4164243643
```





Mean&Meadian imputation

Median imputation

```
imputer_median = MeanMedianImputer(variables = need_impute_num, imputation_method = 'median')

imputer_median.fit(X_train, y_train)

X_train_impute_median = imputer_median.transform(X_train)

X_test_impute_median = imputer_median.transform(X_test)
```

```
# Distribution after median imputation

for var in need_impute_num:

    check_dist_and_variance(X_train, X_train_impute_median, var)
```

Mean Median Imputation:

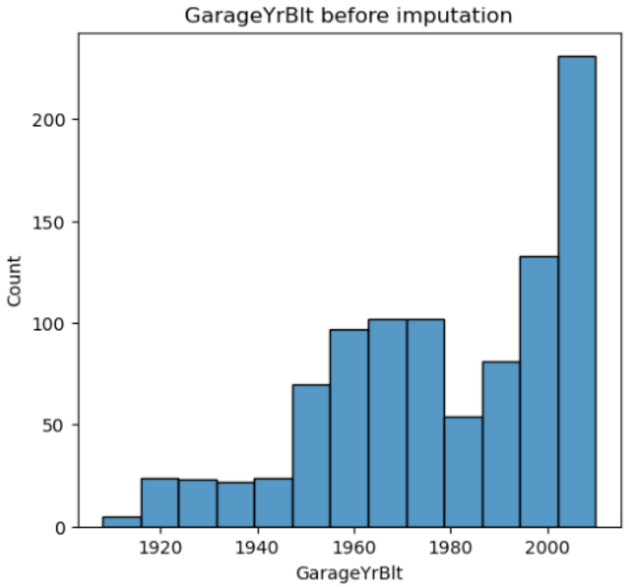
Median Imputation

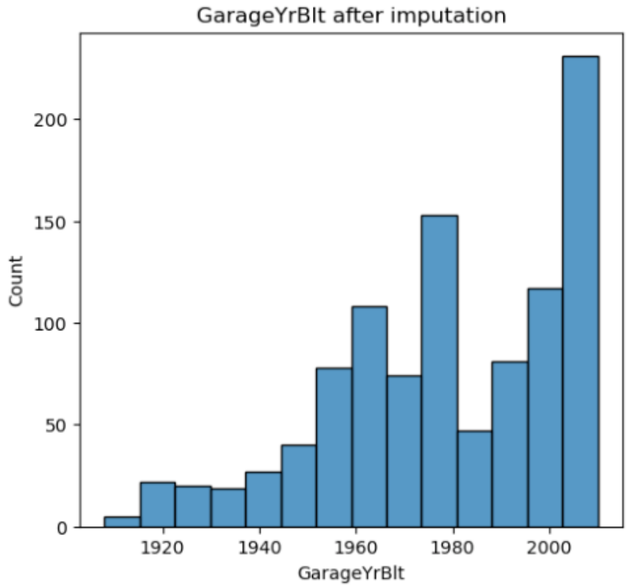
Distribution after median imputation

Distribution after median imputation

Variance before imputation: 601.6565750339723

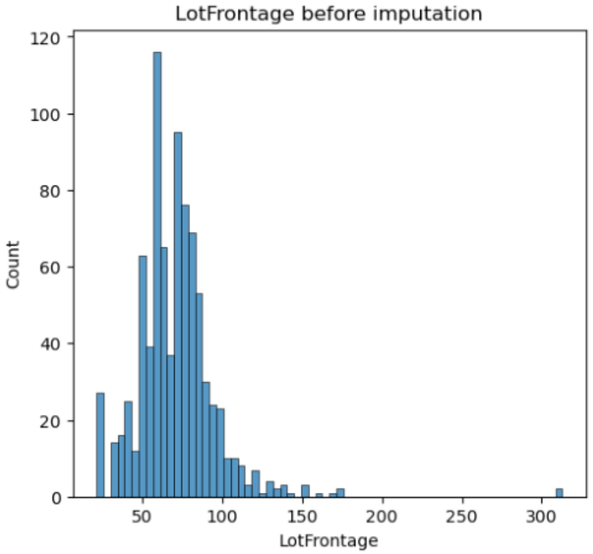
Variance after imputation: 569.9412005420418

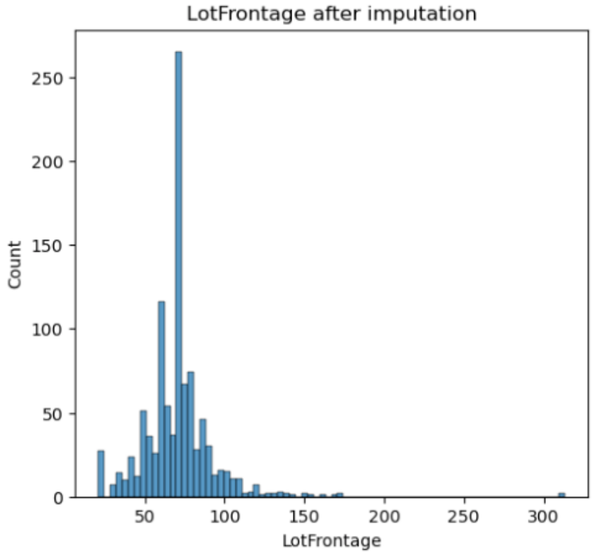




Variance before imputation: 631.2577620805455

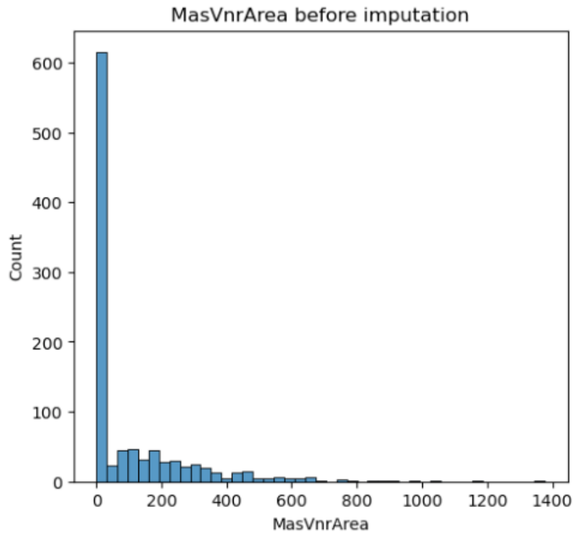
Variance after imputation: 519.9834867010011

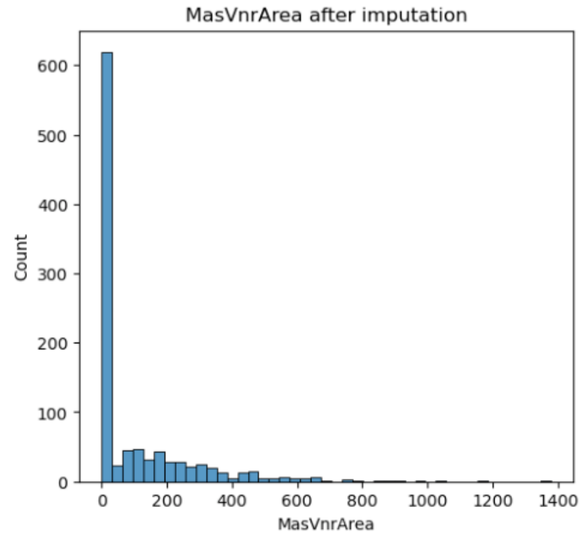




Variance before imputation: 31981.573954946656

Variance after imputation: 31898.268165970585



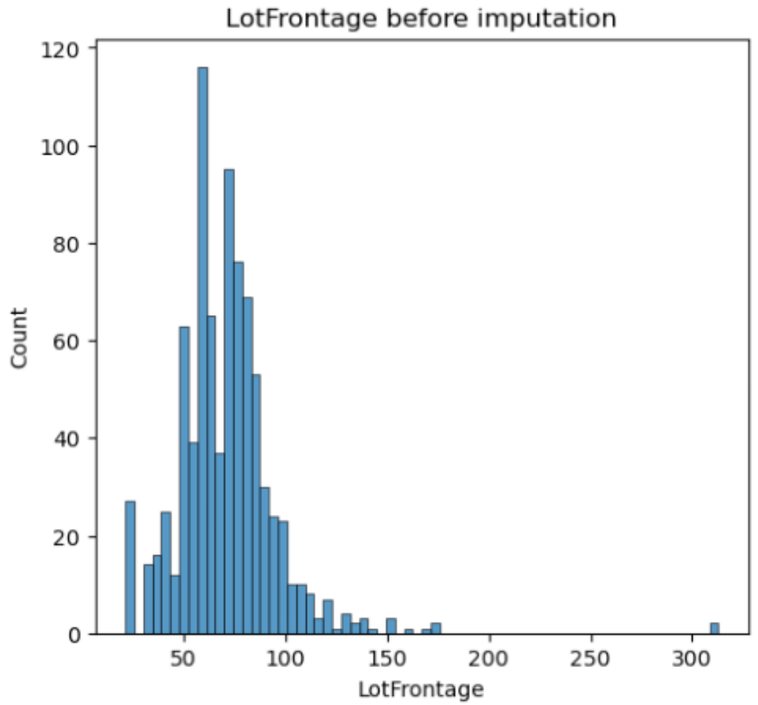


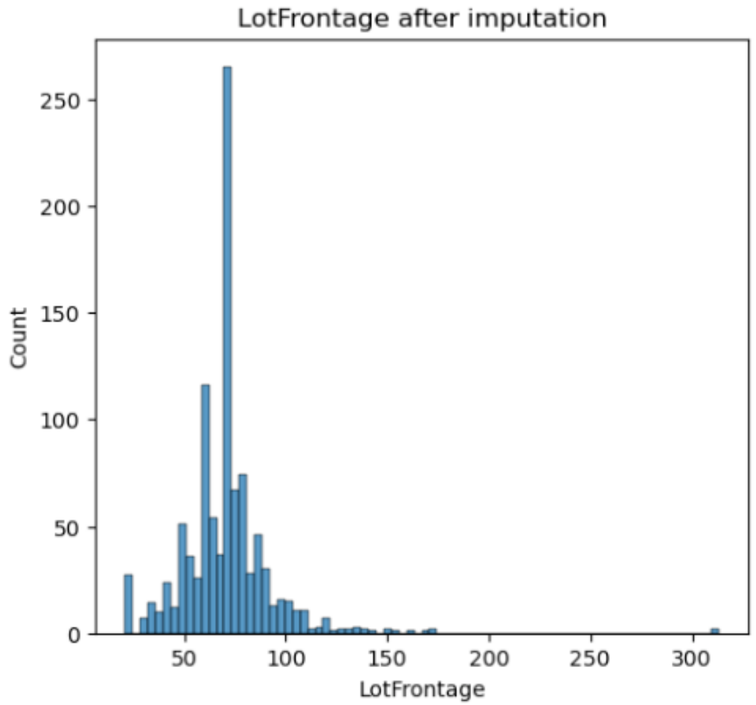
Mean imputation

```
imputer_mean = MeanMedianImputer(variables = ['LotFrontage'], imputation_method = 'median')

imputer_mean.fit(X_train, y_train)

X_train_impute_mean = imputer_mean.transform(X_train)

X_test_impute_mean = imputer_mean.transform(X_test)
```

```
check_dist_and_variance(X_train, X_train_impute_mean, 'LotFrontage')
```

```
Variance before imputation: 631.2577620805455

Variance after imputation: 519.9834867010011
```





Check Distribution after mean imputation

mean imputation method

Random sample imputation

```
imputer_rs = RandomSampleImputer(variables = need_impute, random_state = 4)

imputer_rs.fit(X_train)

X_train_impute_rs = imputer_rs.transform(X_train)

X_test_impute_rs = imputer_rs.transform(X_test)
```

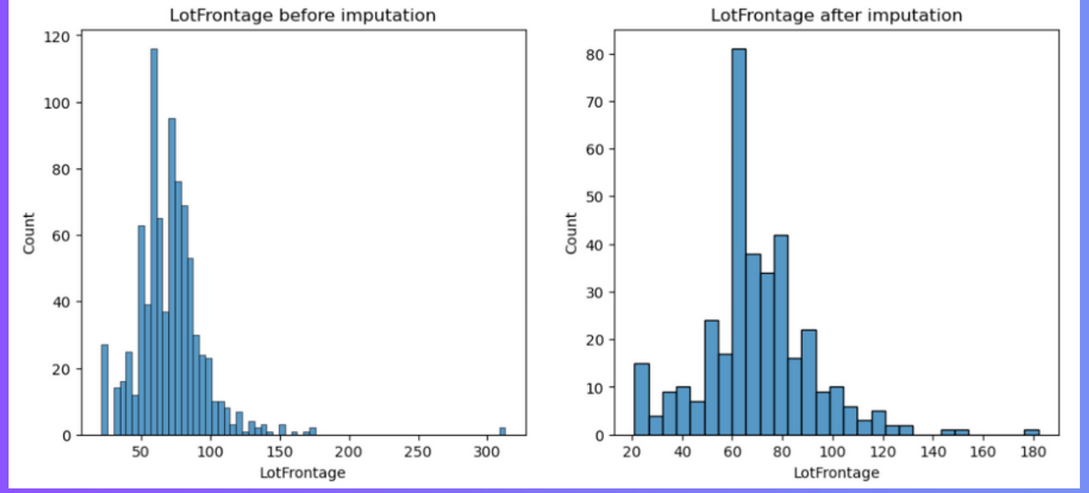
```
# Distribution after random sample imputer
for var in need_impute:
    check_dist_and_variance(X_train, X_test, var)
```

Random Sample Imputation

Distribution after Random Sample Imputation

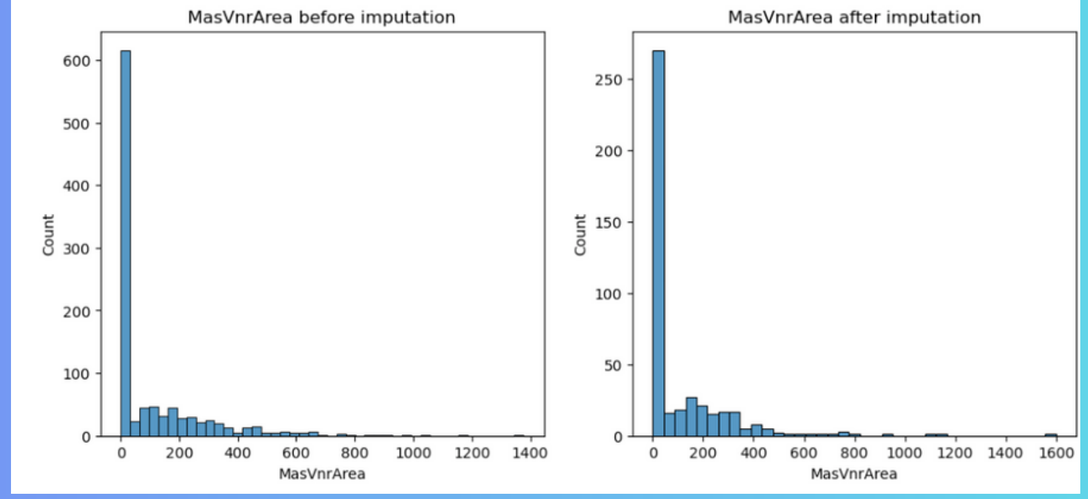
Variance before imputation: 631.2577620805455

Variance after imputation: 493.3044303698978

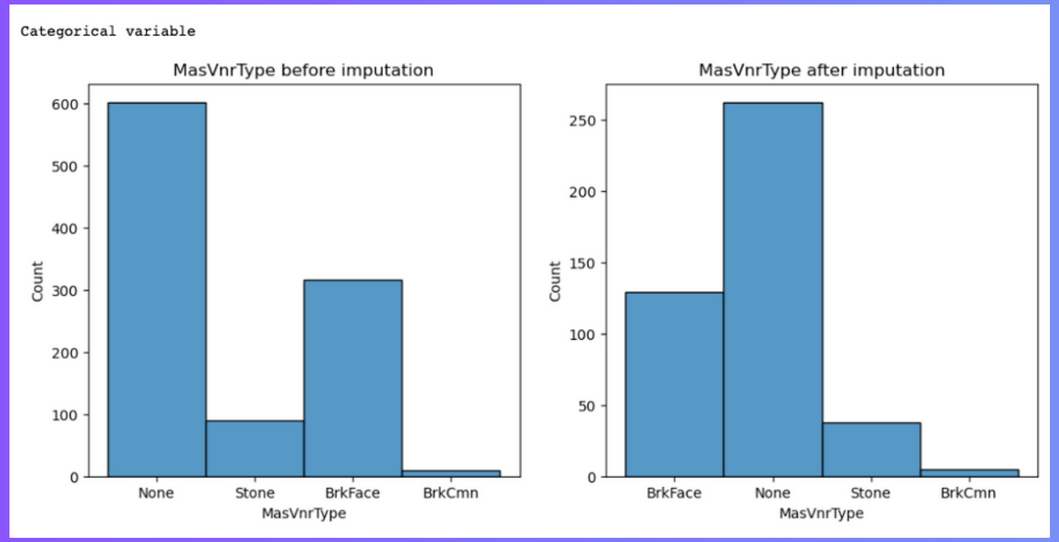


Variance before imputation: 31981.573954946656

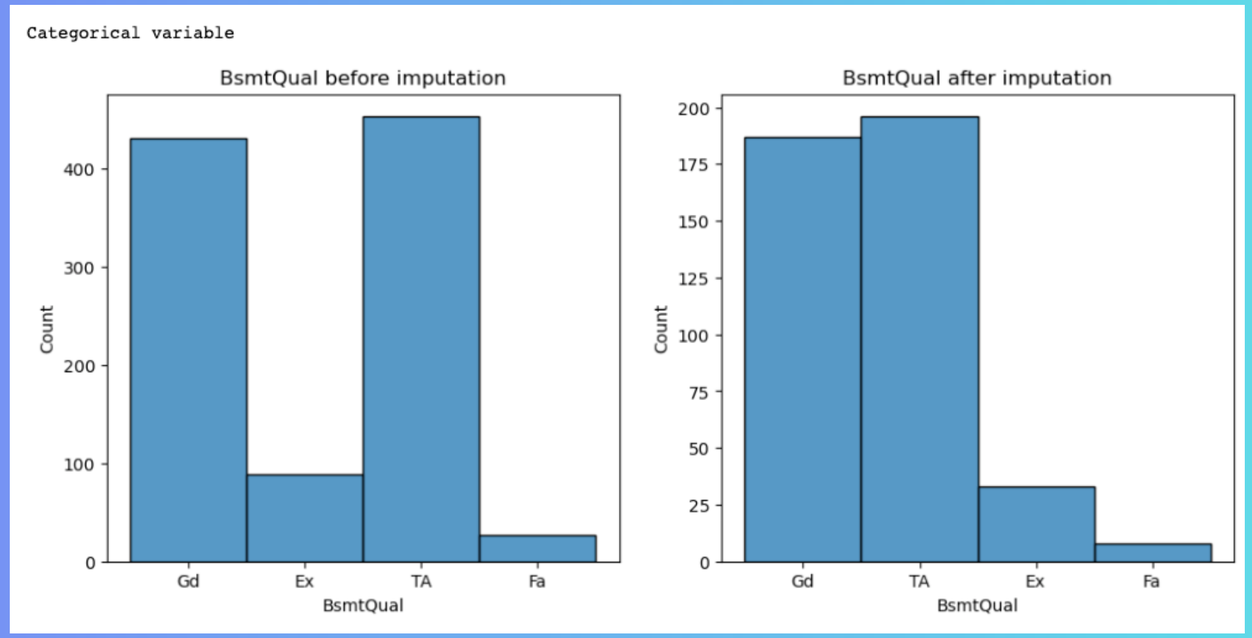
Variance after imputation: 34747.63435893616



Categorical variable



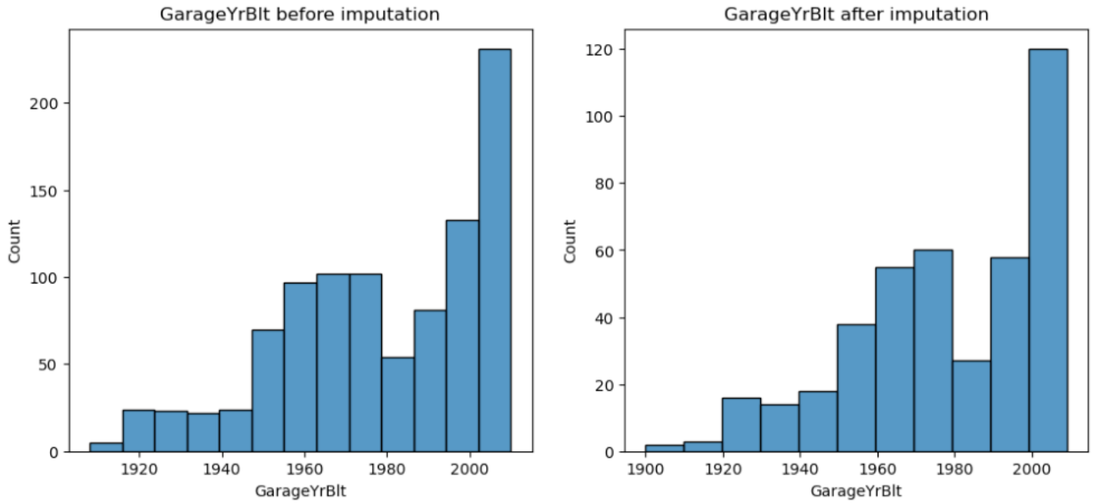
Categorical variable



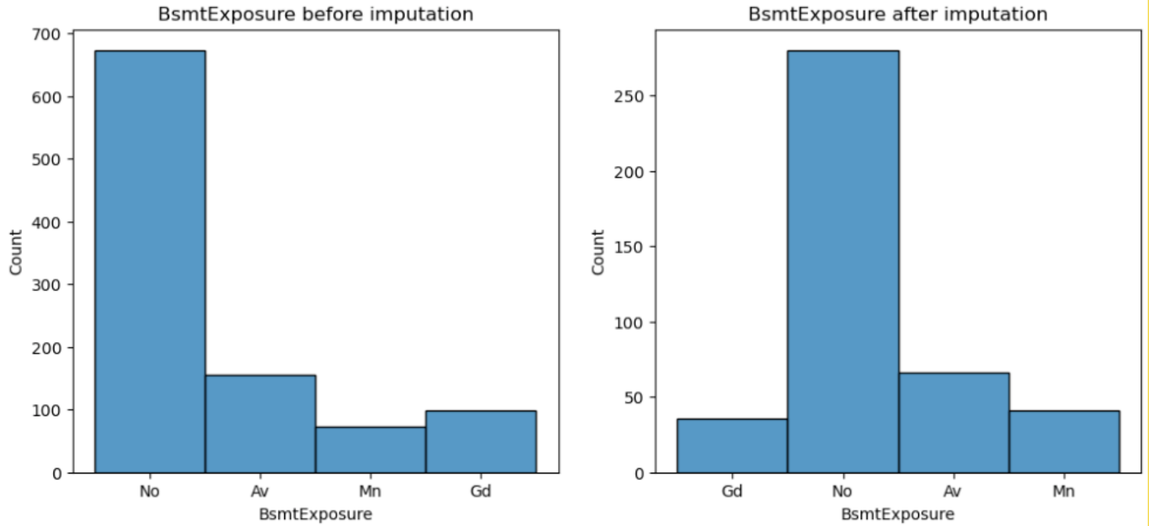
Distribution after Random Sample Imputation

Variance before imputation: 601.6565750339723

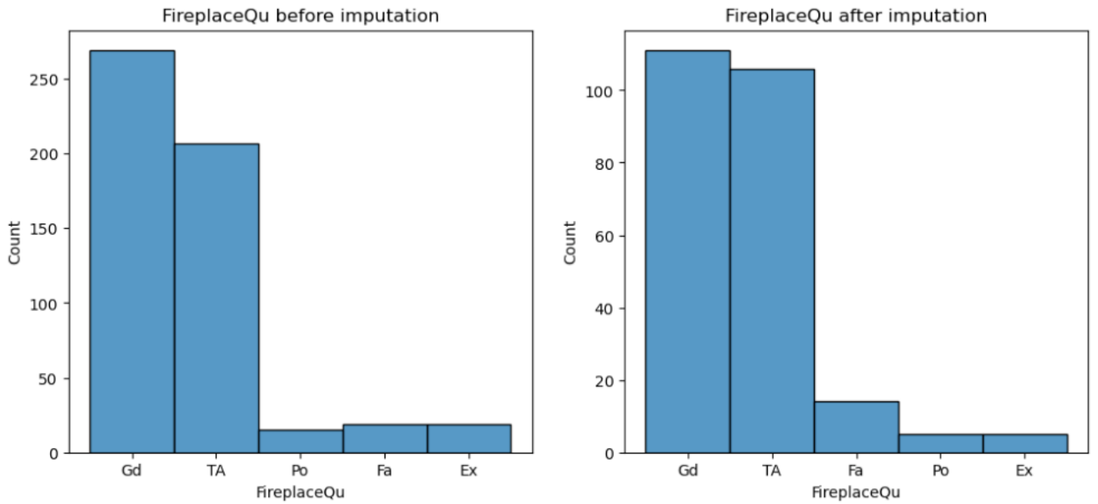
Variance after imputation: 629.7500207702809



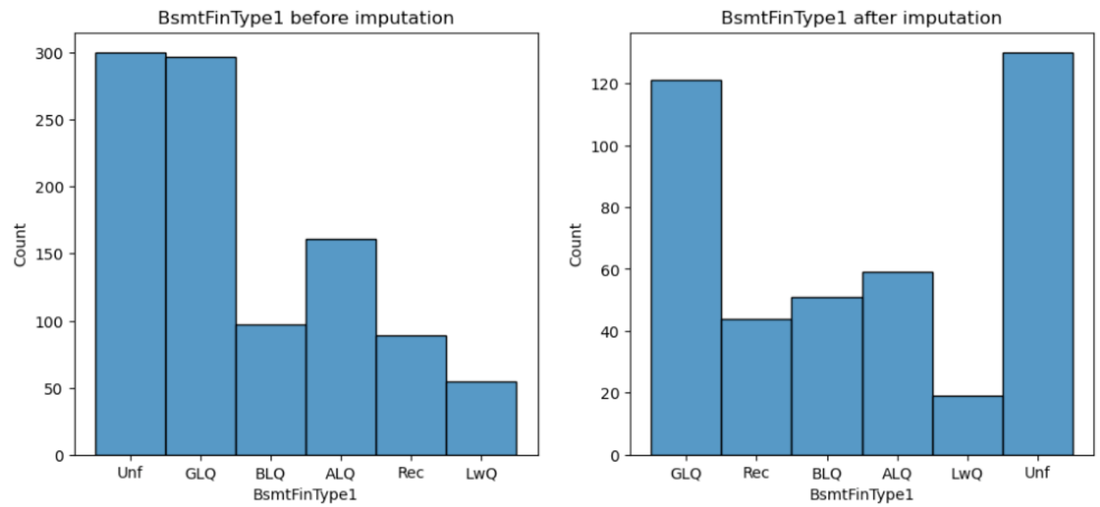
Categorical variable



Categorical variable



Categorical variable



Distribution after Random Sample Imputation

Final imputation stack

```
pipe_imputation = Pipeline(steps = [
    ('frequent', CategoricalImputer(
        variables = ['MasVnrType', 'BsmtCond', 'BsmtExposure', 'BsmtFinType2', 'Electrical', 'BsmtFinType1'],
        imputation_method = 'frequent')
    ),
    ('missng_labels', CategoricalImputer(
        variables = ['FireplaceQu', 'BsmtQual', 'GarageType', 'GarageQual', 'GarageCond', 'GarageFinish'],
        fill_value = 'missing',
        imputation_method = 'missing')
    ),
    ('median', MeanMedianImputer(
        variables = ['MasVnrArea'],
        imputation_method = 'median'
    )
    ),
    ('random_sample', RandomSampleImputer(
        variables = ['LotFrontage', 'GarageYrBlt'],
        random_state = 4
    )
    )
])

pipe_imputation.fit(X_train)
```

Final Ipmutation Stack with PipeLine

```
X_train_impute = pipe_imputation.transform(X_train)

X_test_impute = pipe_imputation.transform(X_test)
```

```
# Check imputation features in sets
print('Train set \n')
print(X_train_impute[need_impute].isnull().mean())

print('\n Test set \n')
print(X_test_impute[need_impute].isnull().mean())
```

```
Train set

LotFrontage     0.0
MasVnrType      0.0
MasVnrArea      0.0
BsmtQual        0.0
BsmtCond        0.0
BsmtExposure    0.0
BsmtFinType1    0.0
BsmtFinType2    0.0
Electrical      0.0
FireplaceQu     0.0
GarageType      0.0
GarageYrBlt     0.0
GarageFinish    0.0
GarageQual      0.0
GarageCond      0.0
dtype: float64
```

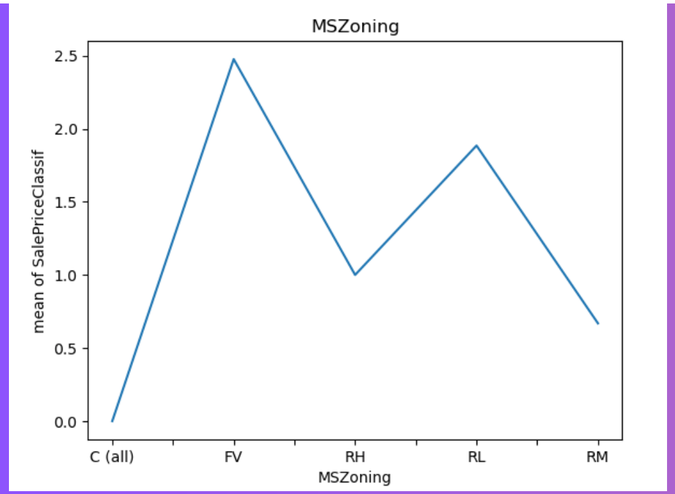
```
 Test set

LotFrontage     0.0
MasVnrType      0.0
MasVnrArea      0.0
BsmtQual        0.0
BsmtCond        0.0
BsmtExposure    0.0
BsmtFinType1    0.0
BsmtFinType2    0.0
Electrical      0.0
FireplaceQu     0.0
GarageType      0.0
GarageYrBlt     0.0
GarageFinish    0.0
GarageQual      0.0
GarageCond      0.0
dtype: float64
```

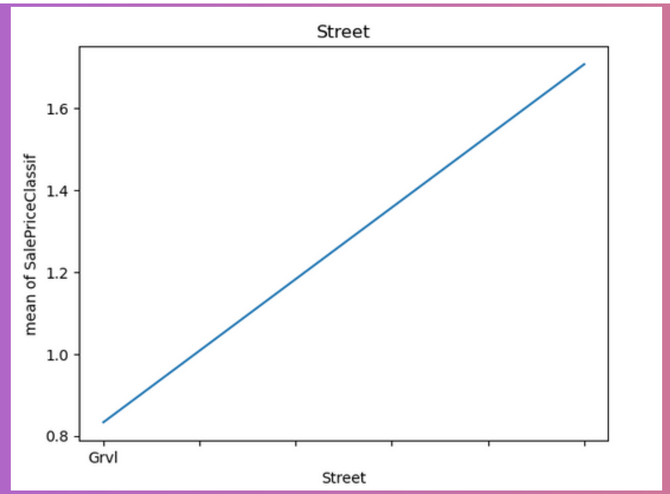
Check Missing Value in Train and Test Sets

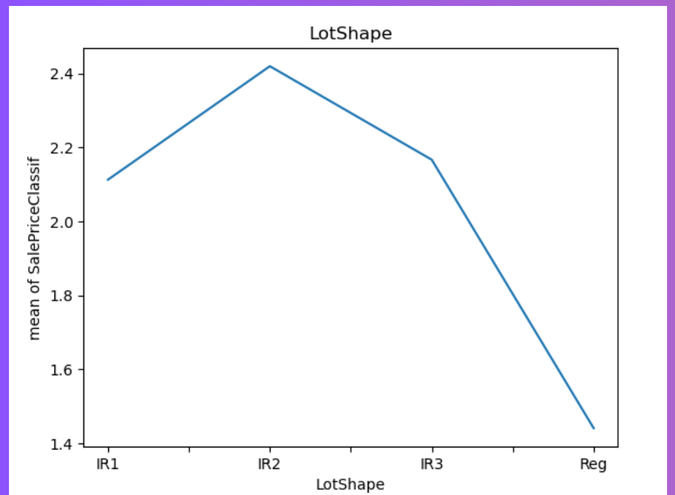
4.3 Encoding

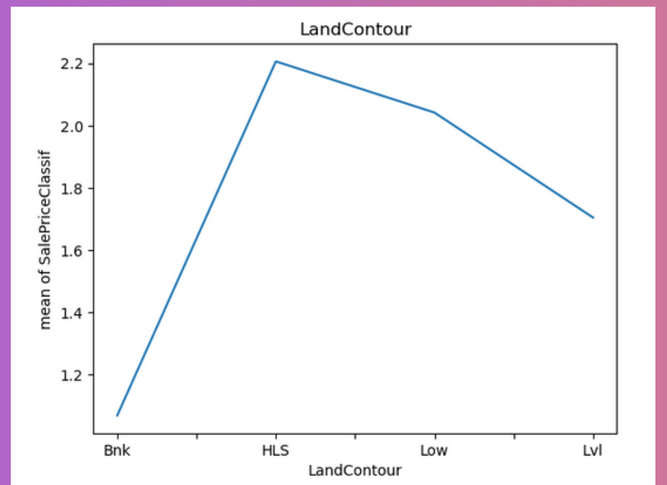
Checking relationships between categorical features and target

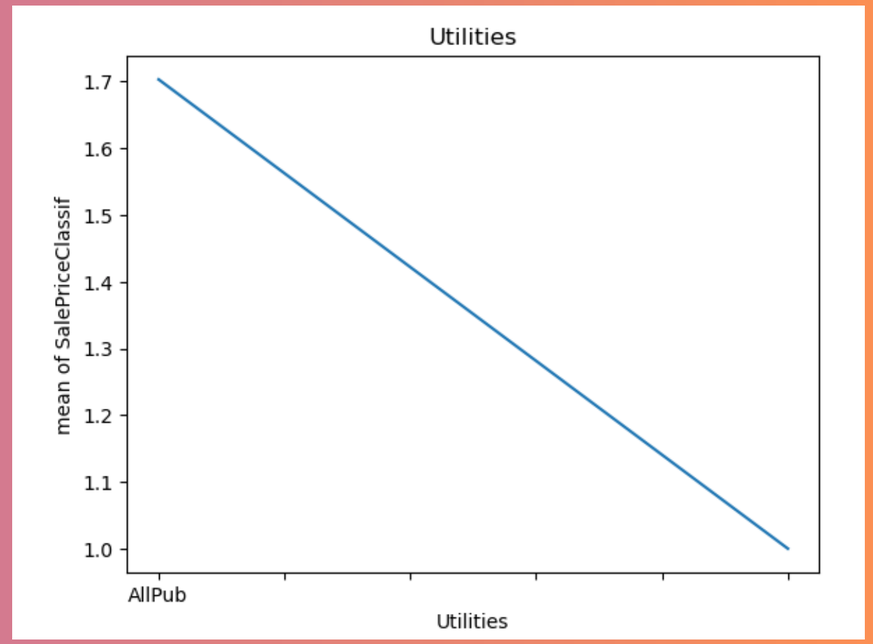
```
for var in categorical_features_train:
    check_relationship_with_target(X_train_impute, y_train, var)
```

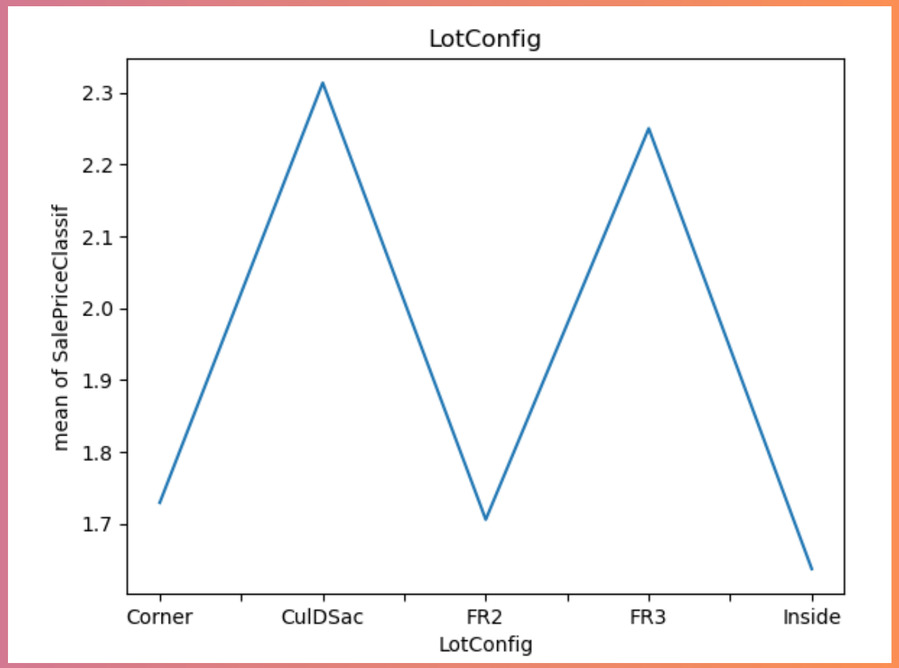


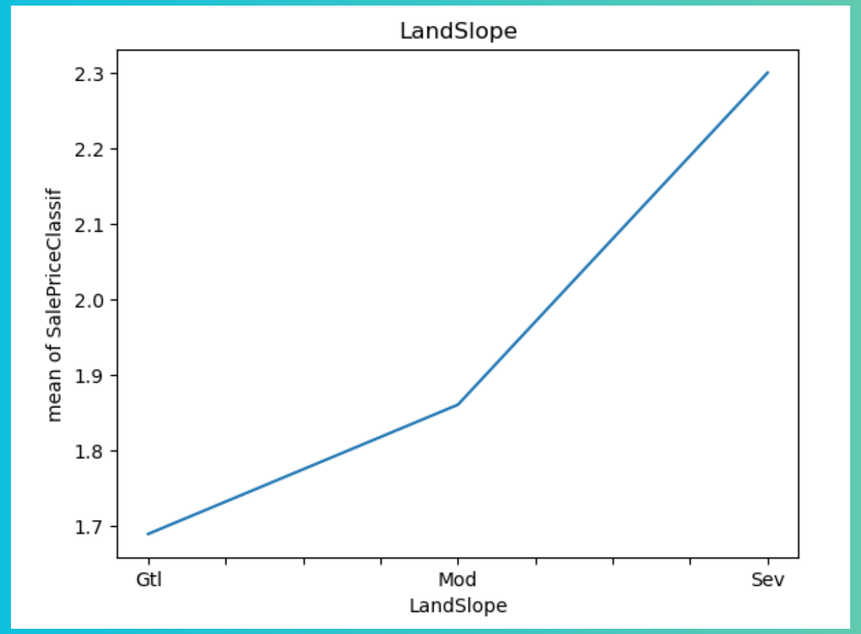


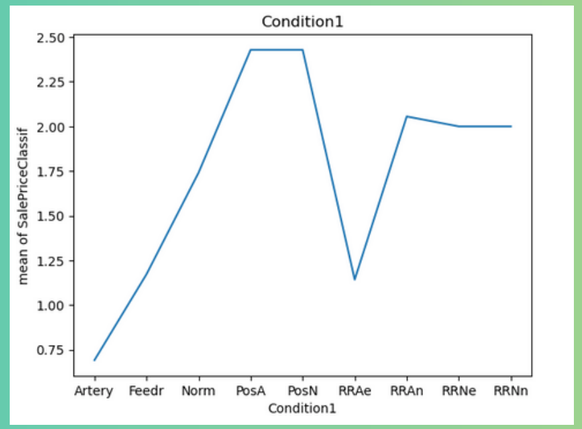


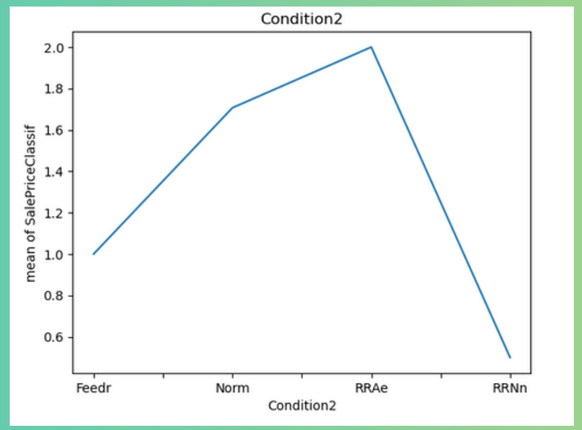


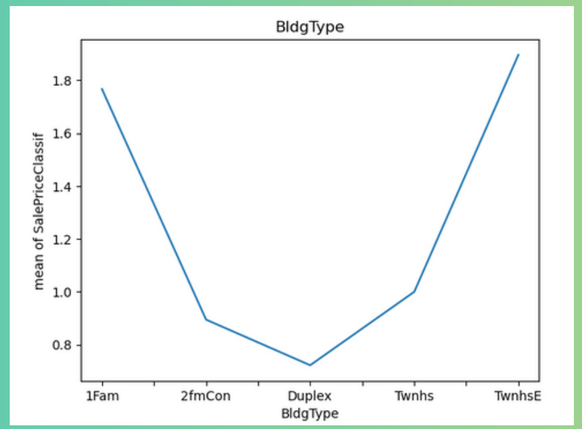


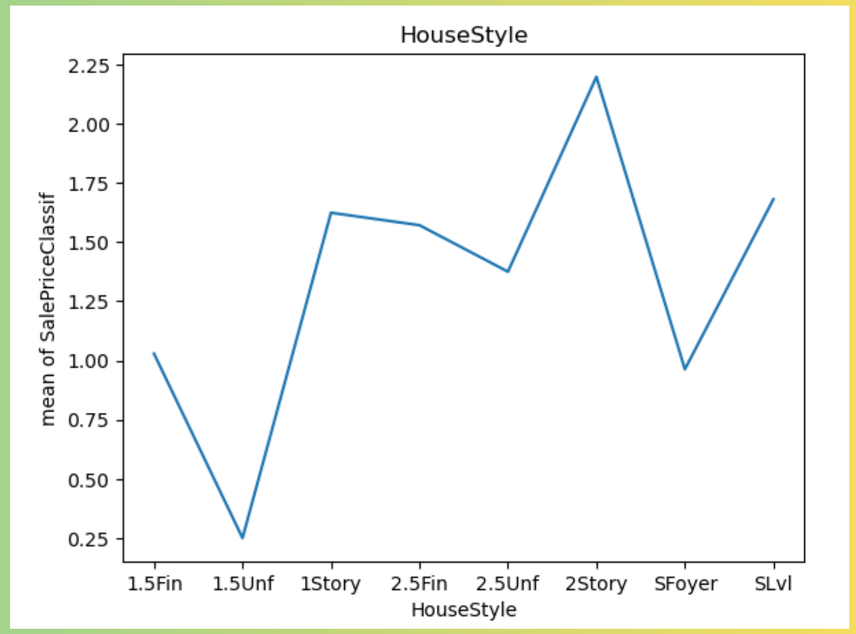


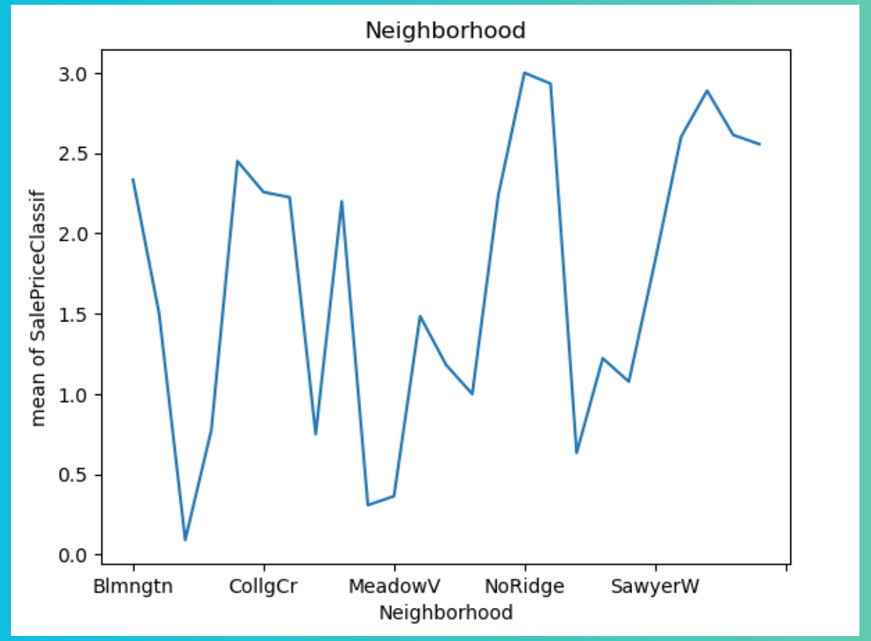

LandSlope
mean of SalePriceClassif
2.3
2.2
2.1
2.0
1.9
1.8
1.7
Gtl
Mod
Sev
LandSlope


Condition1
mean of SalePriceClassif
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
Artery
Feedr
Norm
PosA
PosN
RRAe
RRAn
RRNe
RRNn
Condition1


Condition2
mean of SalePriceClassif
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
Feedr
Norm
RRAe
RRNn
Condition2


BldgType
mean of SalePriceClassif
1.8
1.6
1.4
1.2
1.0
0.8
1Fam
2fmCon
Duplex
Twnhs
TwnhsE
BldgType


HouseStyle
mean of SalePriceClassif
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
1.5Fin
1.5Unf
1Story
2.5Fin
2.5Unf
2Story
SFoyer
SLvl
HouseStyle

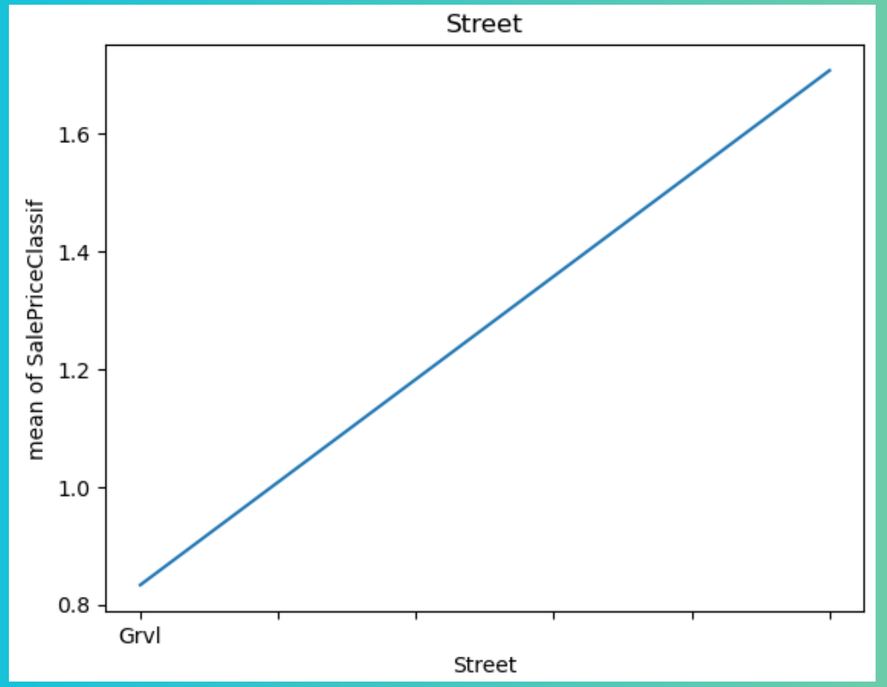

Neighborhood
mean of SalePriceClassif
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Blmngtn
CollgCr
MeadowV
NoRidge
SawyerW
Neighborhood

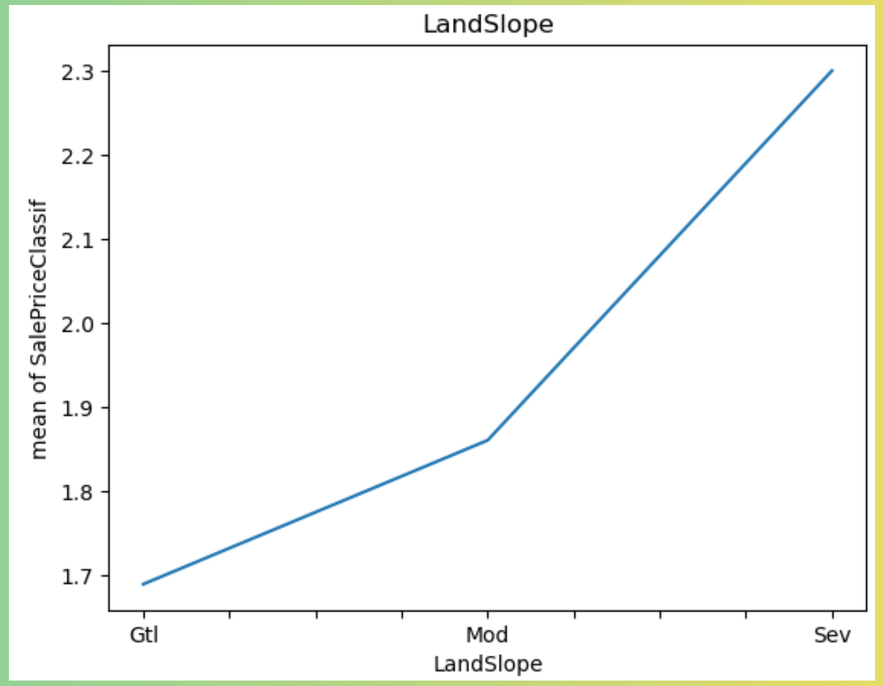

RoofStyle
mean of SalePriceClassif
2.4
2.2
2.0
1.8
1.6
Flat
Gable
Gambrel
Hip
Mansard
Shed
RoofStyle

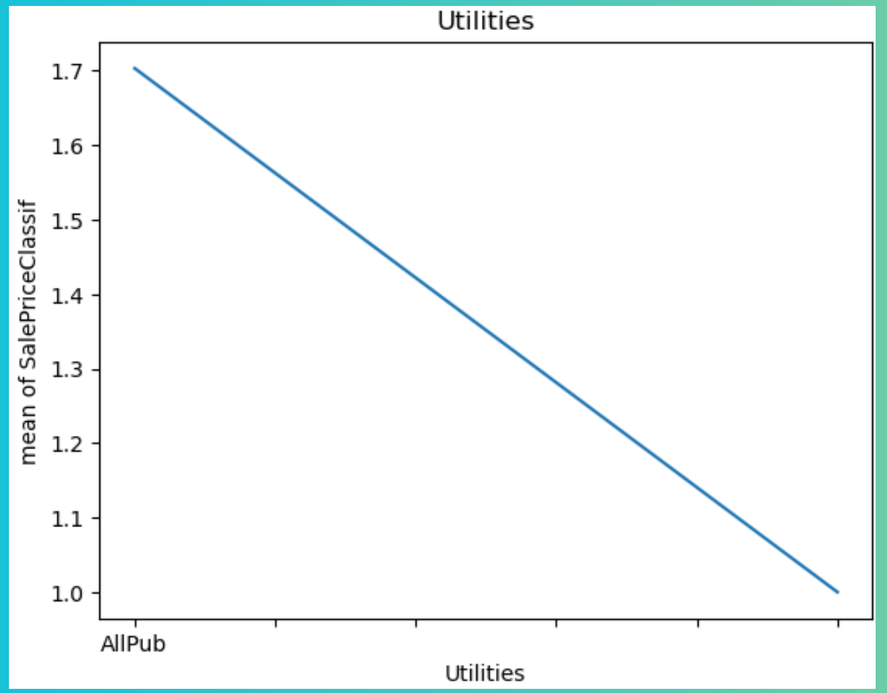
```python
categorical_norm_relationships = ['Street', 'Utilities', 'LandSlope', 'CentralAir']

categorical_not_norm_relationships = [
    var for var in categorical_features_train if var not in categorical_norm_relationships
]

print(' Categorical features with norm relationship with target: \n', categorical_norm_relationships, '\n')

print(' Categorical features with not norm relationship with target: \n ', categorical_not_norm_relationships, '\n'
```

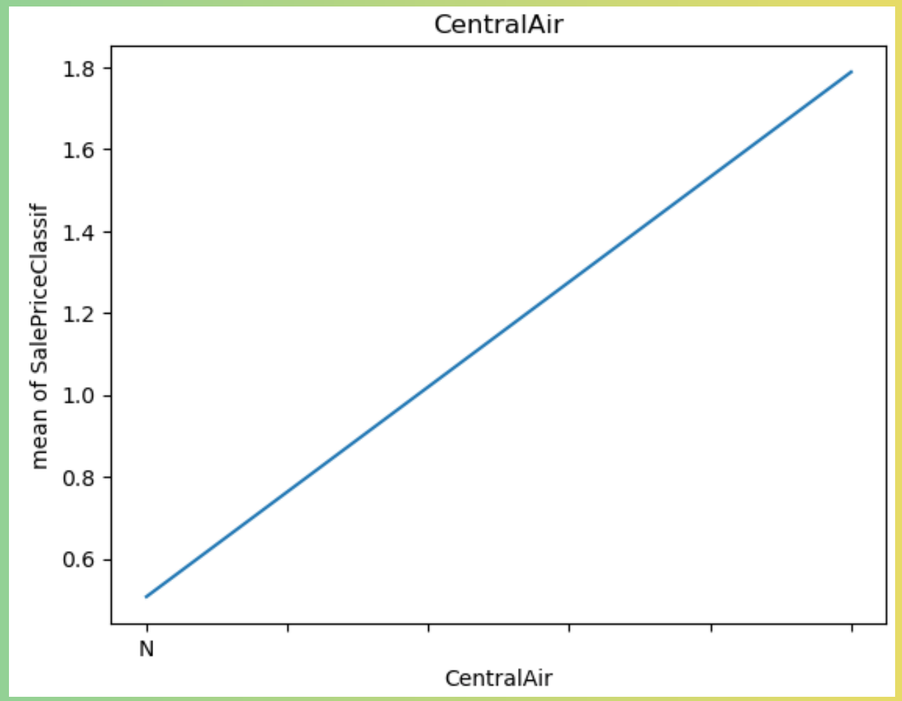
```
 Categorical features with norm relationship with target:
 ['Street', 'Utilities', 'LandSlope', 'CentralAir']

 Categorical features with not norm relationship with target:
  ['MSZoning', 'LotShape', 'LandContour', 'LotConfig', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'House
Style', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'Electrical', 'Kitche
nQual', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'SaleTyp
e', 'SaleCondition']
```

Categorical Features with Norm and Not Norm Relationships


Street
mean of SalePriceClassif
1.6
1.4
1.2
1.0
0.8
Grvl
Street


LandSlope
mean of SalePriceClassif
2.3
2.2
2.1
2.0
1.9
1.8
1.7
Gtl
Mod
Sev
LandSlope


Utilities
mean of SalePriceClassif
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
AllPub
Utilities


CentralAir
mean of SalePriceClassif
1.8
1.6
1.4
1.2
1.0
0.8
0.6
N
CentralAir

One hot encoding

```
# Cardinality for OneHot encoding

for var in categorical_norm_relationships:
    print(var, '\n Count of unique values: ', X_train[var].nunique(), '\n')
```

```
Street
 Count of unique values:  2

Utilities
 Count of unique values:  2

LandSlope
 Count of unique values:  3

CentralAir
 Count of unique values:  2
```

```
# Don't tune top_categories, for the reason of low cardinality
enc_ohe = OneHotEncoder(
    variables = categorical_norm_relationships,
    drop_last = False)

enc_ohe.fit(X_train_impute)

X_train_enc_ohe = enc_ohe.transform(X_train_impute)

X_test_enc_ohe = enc_ohe.transform(X_test_impute)
```

```
len(X_train_impute.columns), len(X_train_enc_ohe.columns)
```

```
(74, 79)
```

Mean encoding

```
for var in X_train_impute.columns:
    if X_train_impute[var].isnull().mean() > 0:
        print(var)
```

```
categorical_mean = categorical_not_norm_relationships.copy()

categorical_mean.remove('Condition1')
categorical_mean.remove('Condition2')
categorical_mean.remove('RoofMatl')
categorical_mean.remove('Exterior1st')
categorical_mean.remove('Exterior2nd')
categorical_mean.remove('ExterCond')
categorical_mean.remove('HeatingQC')
categorical_mean.remove('Functional')
categorical_mean.remove('Neighborhood')
categorical_mean.remove('Electrical')
categorical_mean.remove('RoofStyle')
categorical_mean.remove('Heating')
categorical_mean.remove('GarageQual')

encoder_mean = MeanEncoder(variables = categorical_mean)

encoder_mean.fit(X_train_impute, y_train)

X_train_enc_mean = encoder_mean.transform(X_train_impute)

X_test_enc_mean = encoder_mean.transform(X_test_impute)
```

We remove some features because encoding by mean value creates empty variables in the test set

```python
categorical_problem = [
    var for var in categorical_not_norm_relationships if var not in categorical_mean
]

categorical_problem
```

```
['Neighborhood',
 'Condition1',
 'Condition2',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'ExterCond',
 'Heating',
 'HeatingQC',
 'Electrical',
 'Functional',
 'GarageQual']
```

Categorical_Problem

```python
X_train_impute[categorical_problem].nunique()
```

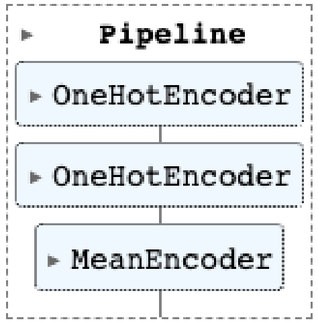
```
Neighborhood    25
Condition1       9
Condition2       4
RoofStyle        6
RoofMatl         7
Exterior1st     14
Exterior2nd     15
ExterCond        4
Heating          5
HeatingQC        5
Electrical       5
Functional       7
GarageQual       6
dtype: int64
```

Categorical_Problem nunique

Build Pipe Line Encoding

```
for var in categorical_features_train:
    if X_train_impute[var].dtype != 'O':
        print(var)
```

```
pipeline_encoding = Pipeline(
    steps = [
        ('ohe', OneHotEncoder(variables = categorical_norm_relationships)),
        ('ohe_top_cat', OneHotEncoder(variables = categorical_problem, top_categories = 3)),
        ('mean', MeanEncoder(variables =  categorical_mean))
])

pipeline_encoding.fit(X_train_impute, y_train)
```




```
X_train_impute_enc = pipeline_encoding.transform(X_train_impute)

X_test_impute_enc = pipeline_encoding.transform(X_test_impute)
```

Checking object features in set

```
# Checking object features in sets

for var in X_train_impute_enc.columns:
    if X_train_impute_enc[var].dtype == 'O':
        print(var)
```

Discretization

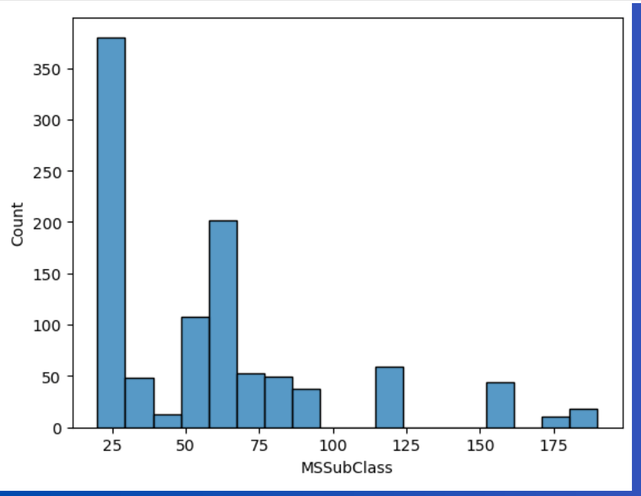
Discretization

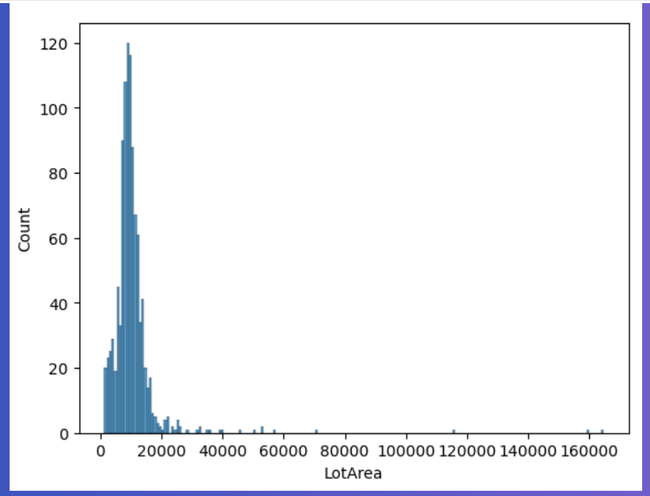
```
# Make continous features list

continous_features_train = [
    var for var in X_train_impute_enc.columns if X_train_impute_enc[var].nunique() > 10
]

continous_features_train
```

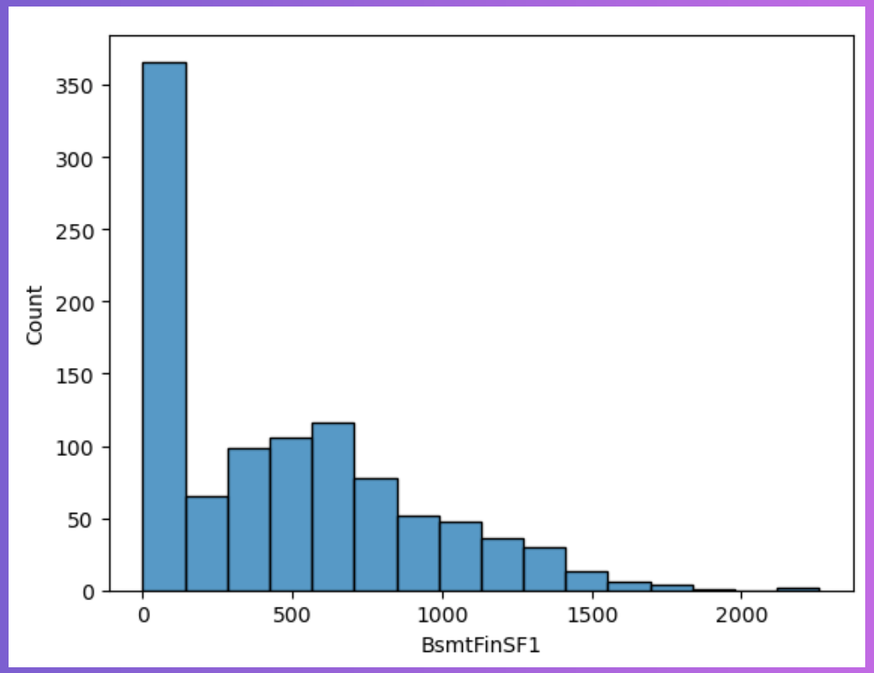
```
['MSSubClass',
 'LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'LowQualFinSF',
 'GrLivArea',
 'TotRmsAbvGrd',
 'GarageYrBlt',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'MiscVal',
 'MoSold',
 'InterSale']
```

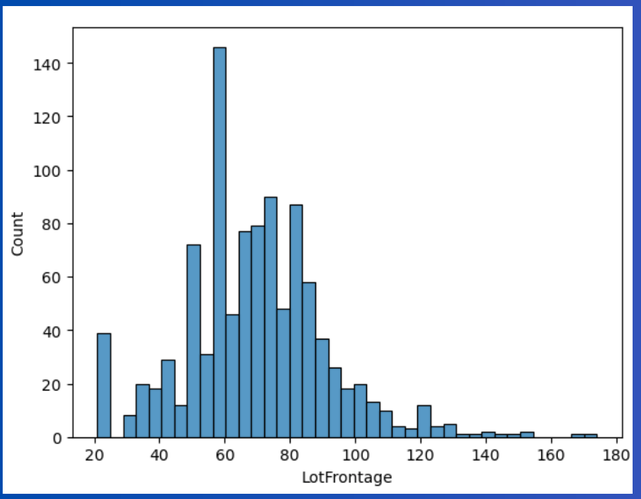
Continuous Features List

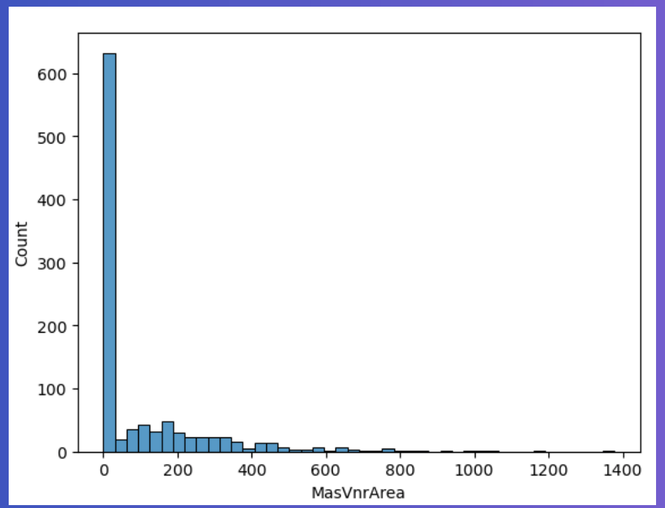
```
# Check distribution of continous features

for var in continous_features_train:
    sns.histplot(X_train_impute_enc[var])
    plt.show()
```

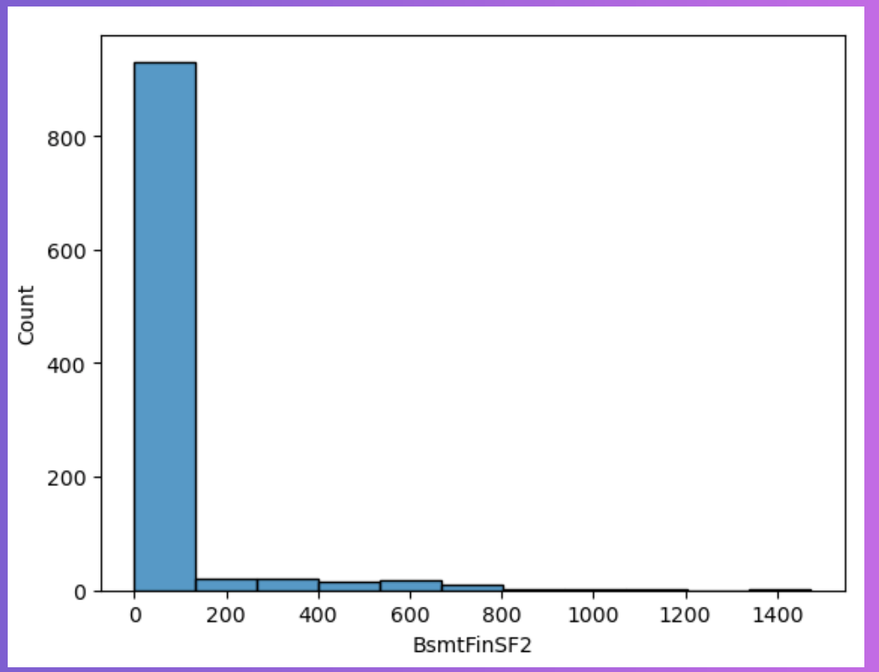


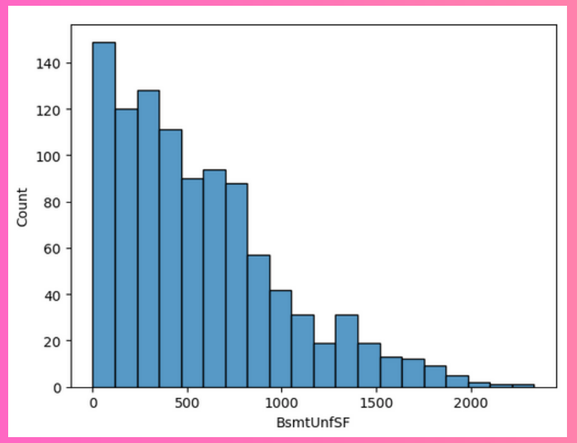


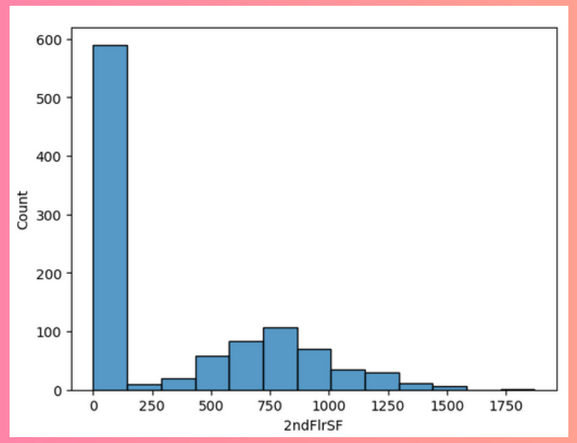


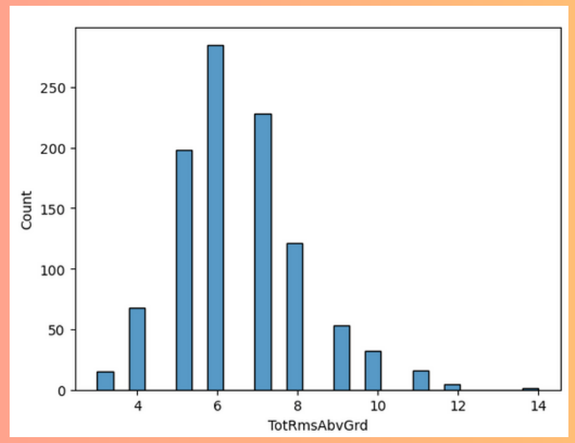


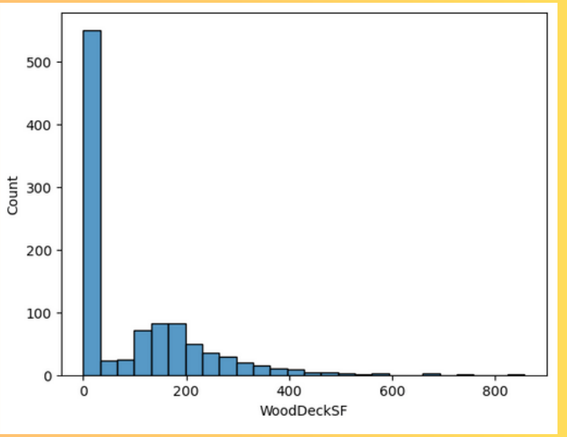


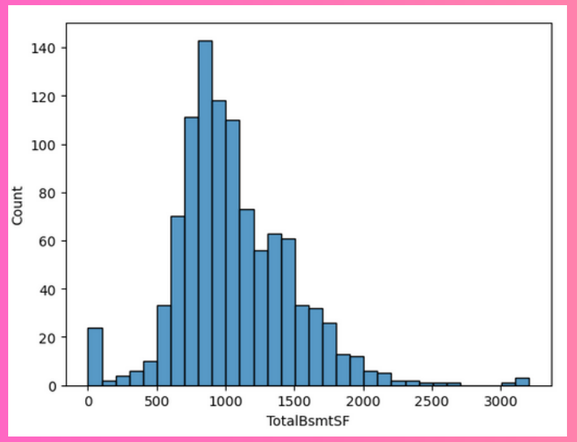


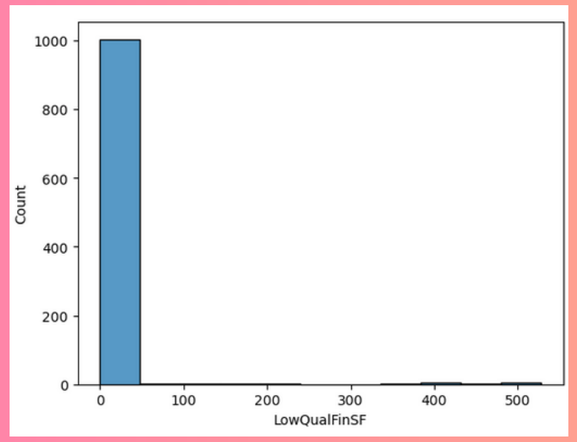

Count
BsmtUnfSF

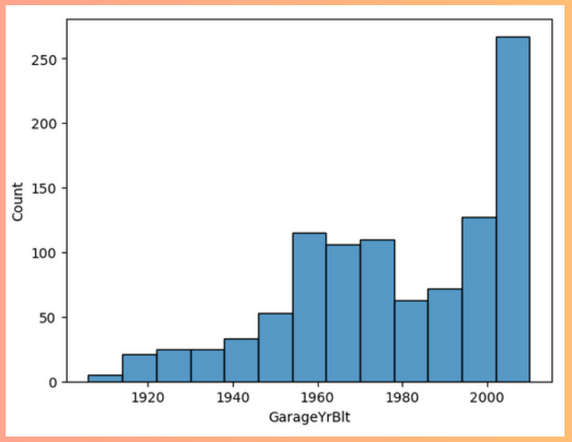

Count
2ndFlrSF

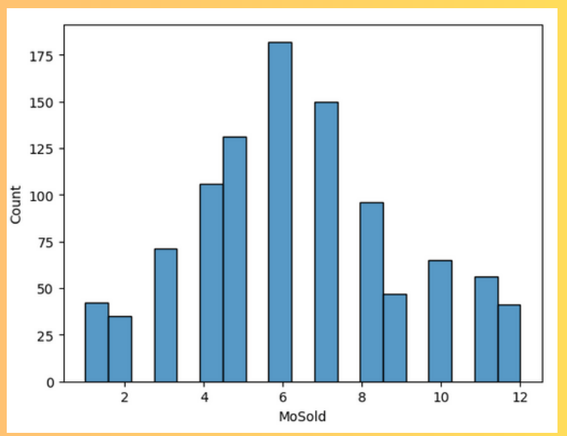

Count
TotRmsAbvGrd

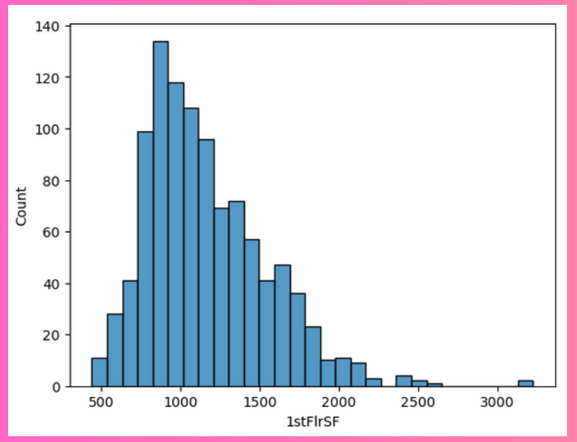

Count
WoodDeckSF

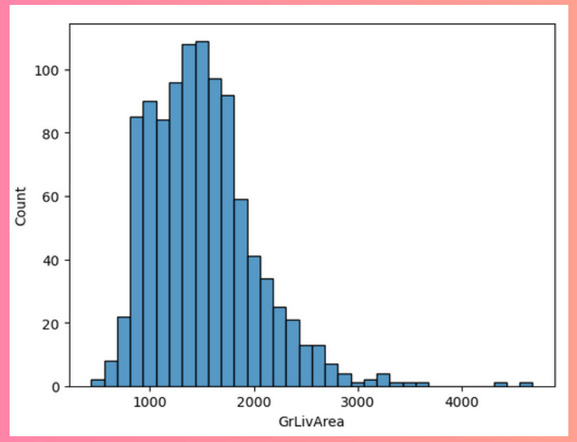

Count
TotalBsmtSF

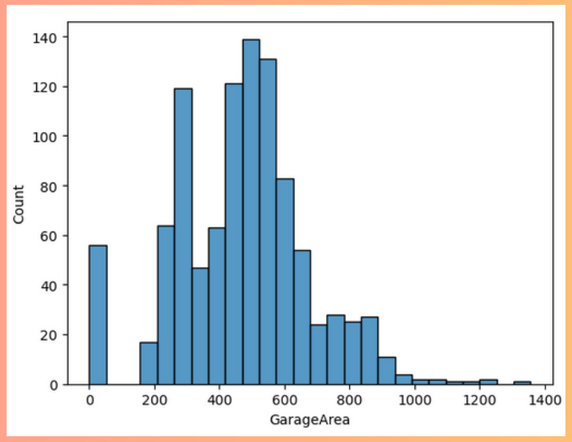

Count
LowQualFinSF

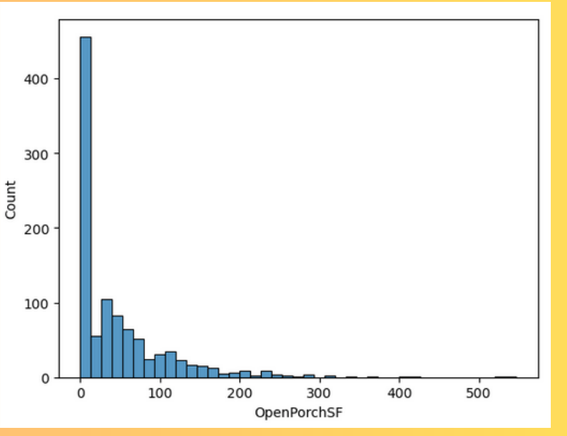

Count
GarageYrBlt


Count
MoSold


Count
1stFlrSF


Count
GrLivArea


Count
GarageArea

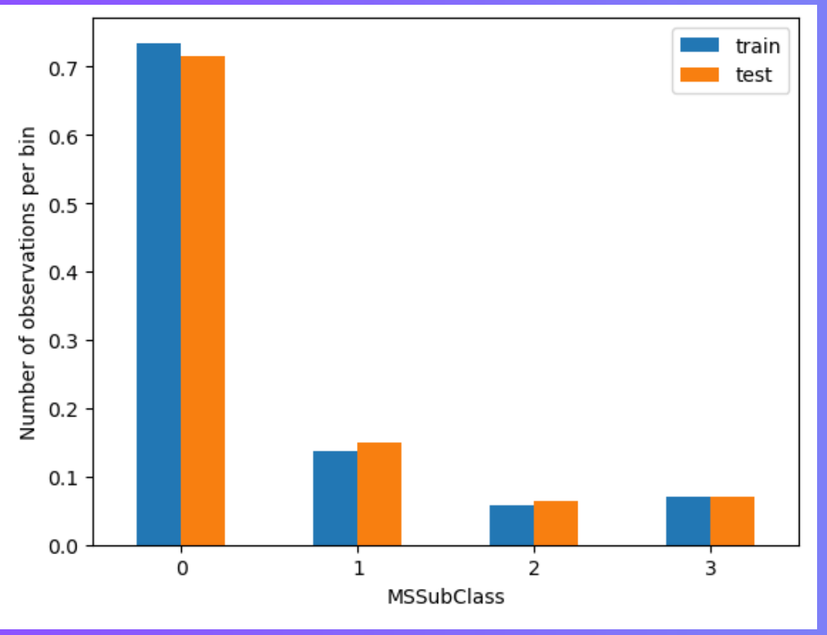

Count
OpenPorchSF

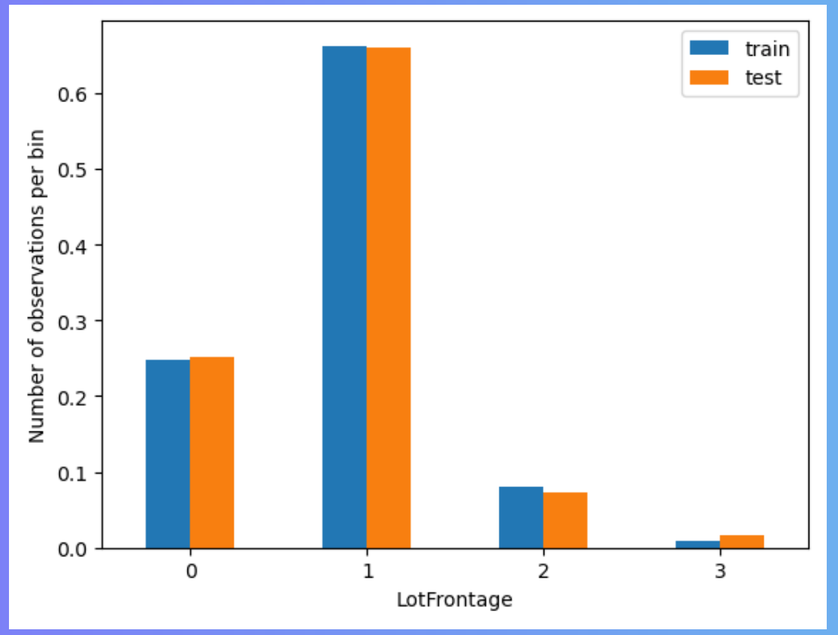
Equal Width Discretization

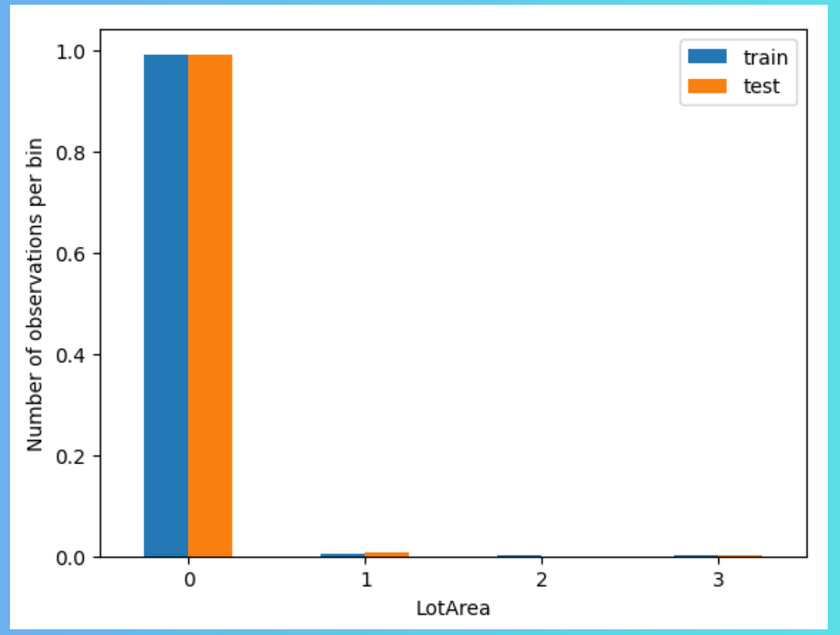
```
equal_width = EqualWidthDiscretiser(bins = 4, variables = continous_features_train)

equal_width.fit(X_train_impute_enc)

X_train_equal_width = equal_width.transform(X_train_impute_enc)

X_test_equal_width = equal_width.transform(X_test_impute_enc)
```

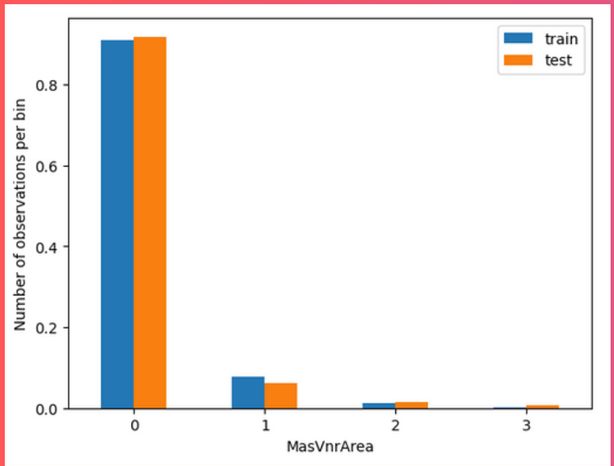
Check balancing of data between train and test sets

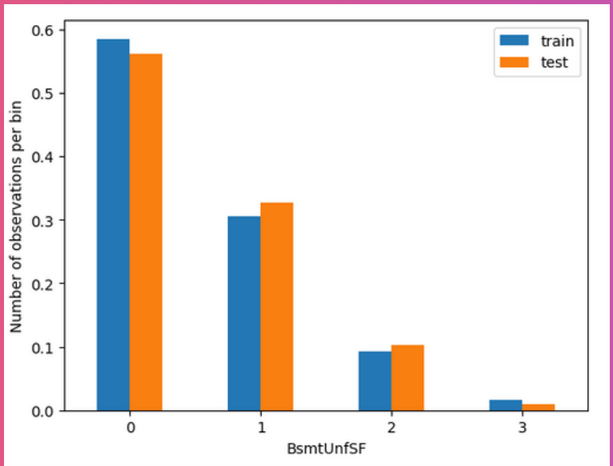
```
for var in continous_features_train:
    check_balance_of_discrete_and_categorical(X_train_equal_width, X_test_equal_width, var)
```

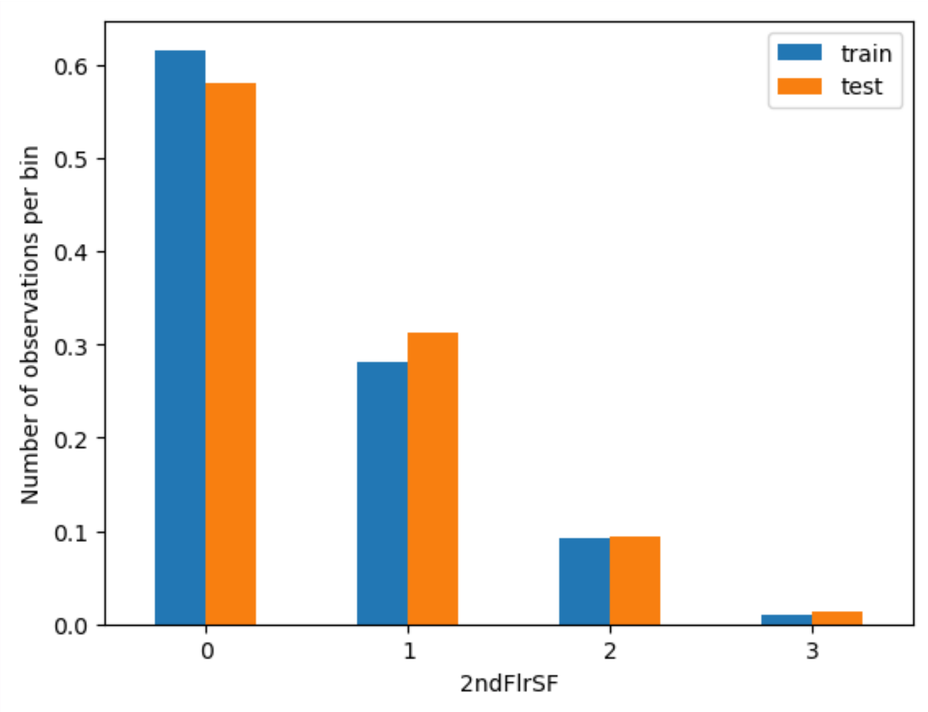


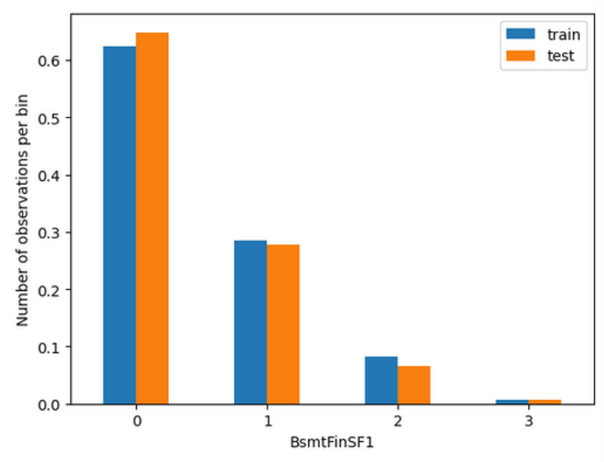


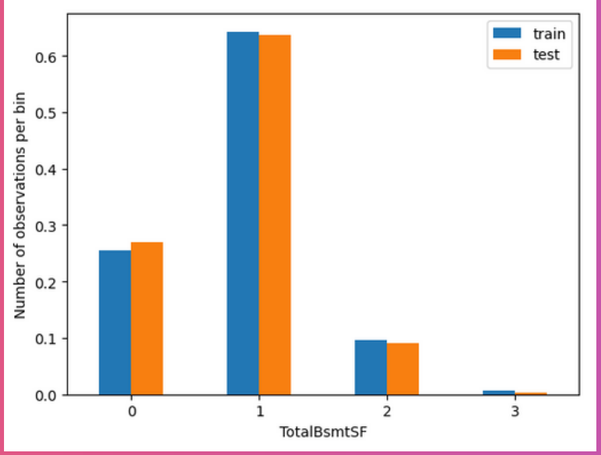


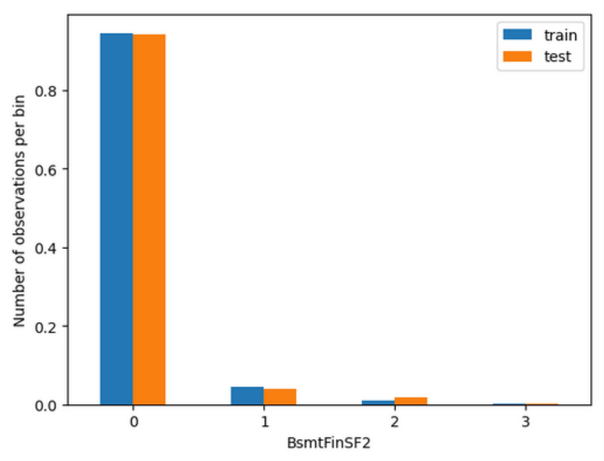

train
test
Number of observations per bin
MasVnrArea

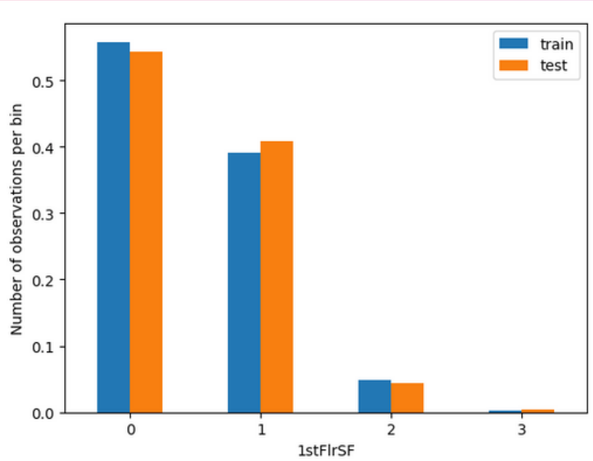

train
test
Number of observations per bin
BsmtUnfSF

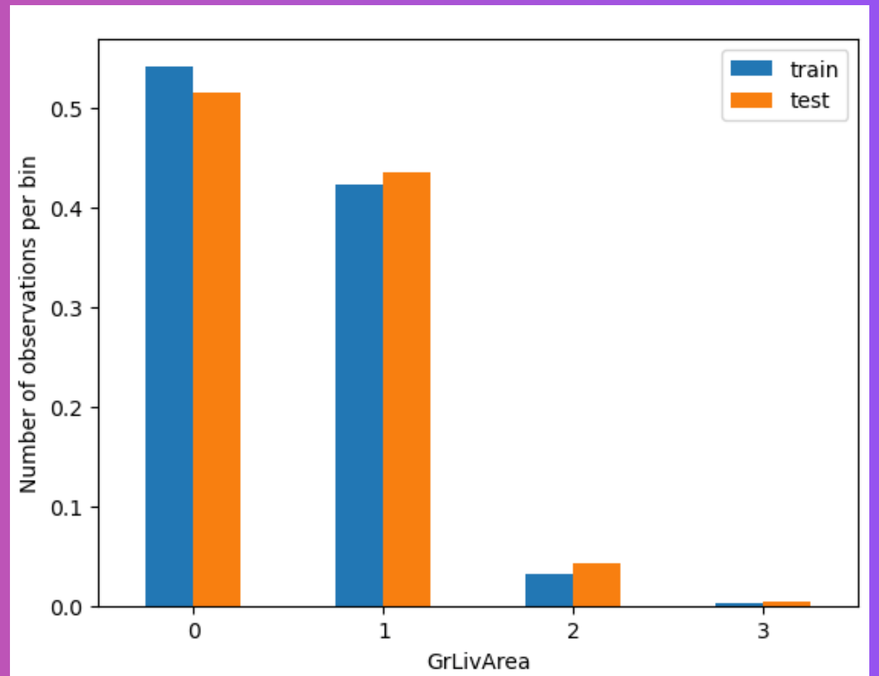

train
test
Number of observations per bin
2ndFlrSF


train
test
Number of observations per bin
BsmtFinSF1


train
test
Number of observations per bin
TotalBsmtSF


train
test
Number of observations per bin
BsmtFinSF2

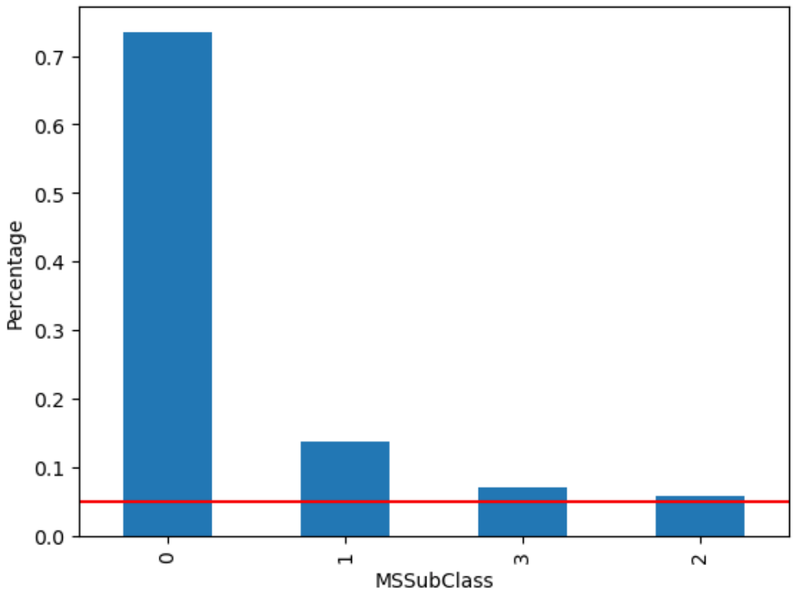

train
test
Number of observations per bin
1stFlrSF

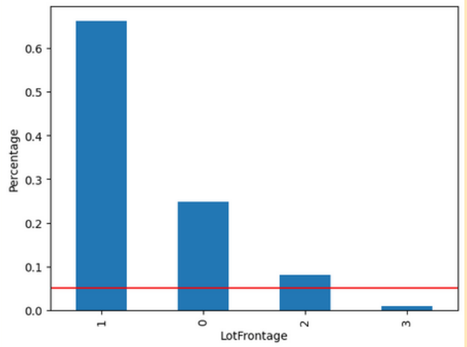

train
test
Number of observations per bin
GrLivArea

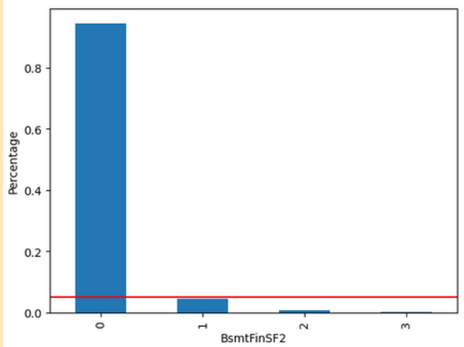
Rare Labels Analysis X_train_equal_width

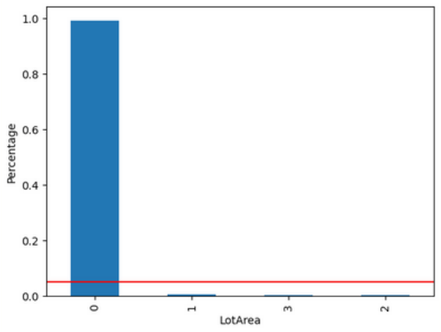
Rare labels analysis

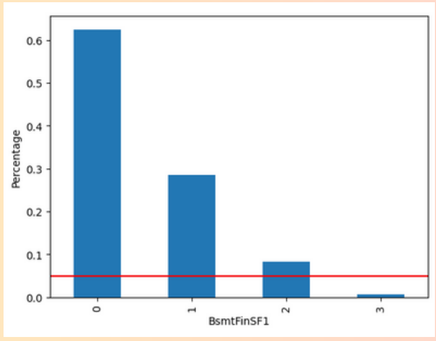
```
1 check_rare_labels_analysis(X_train_equal_width, continous_features_train)
```

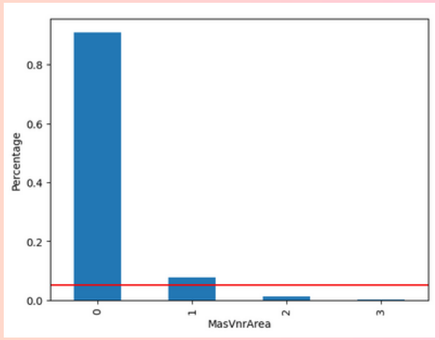


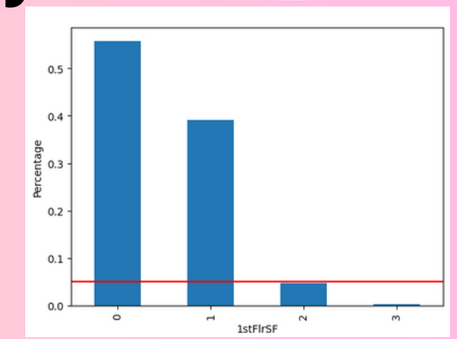


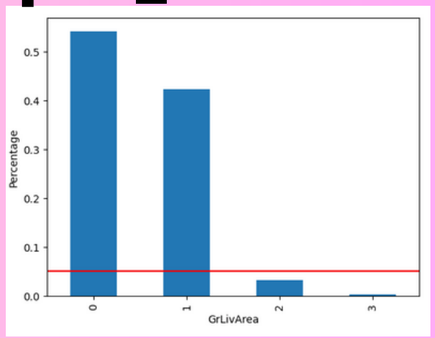


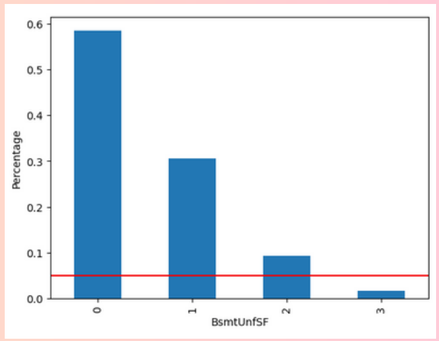


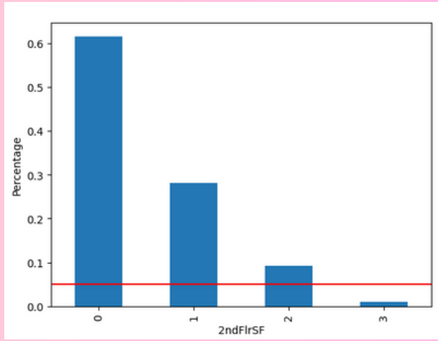


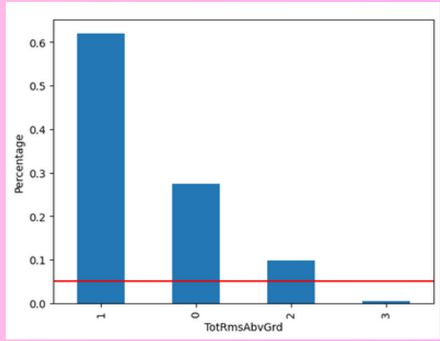


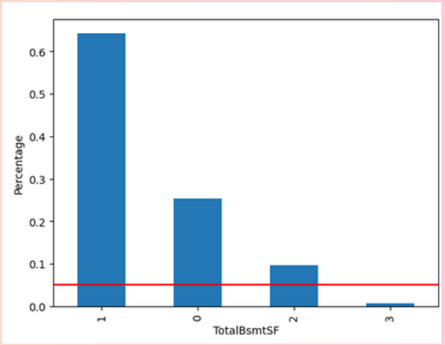


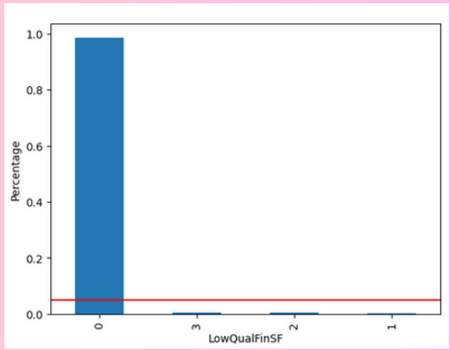


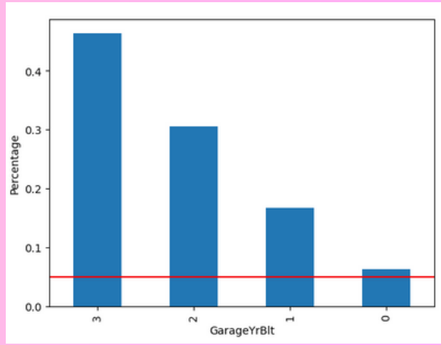










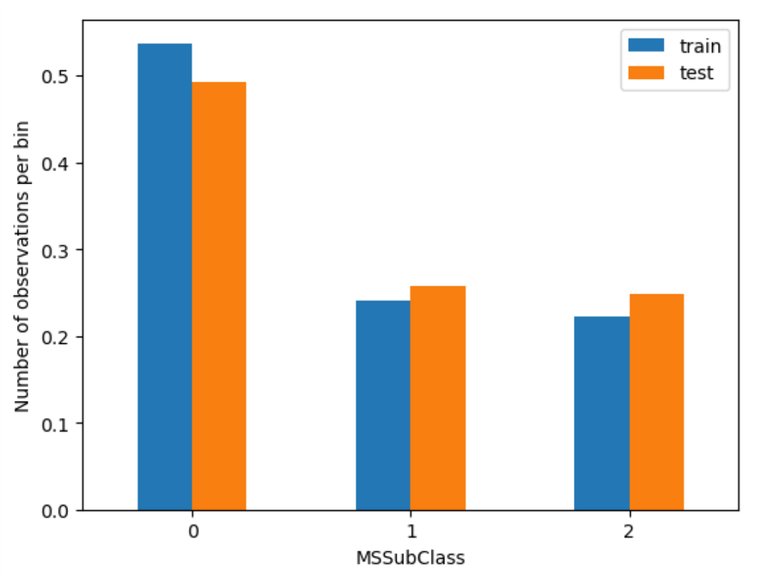


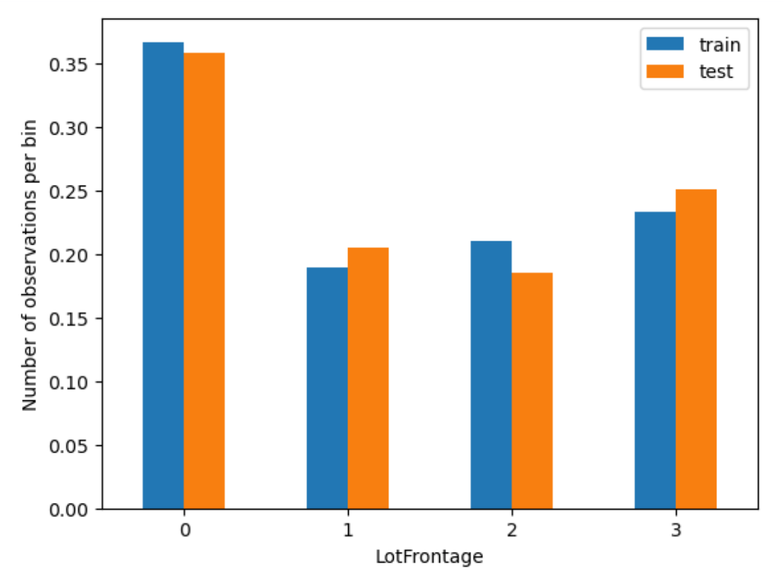
Equal-frequency

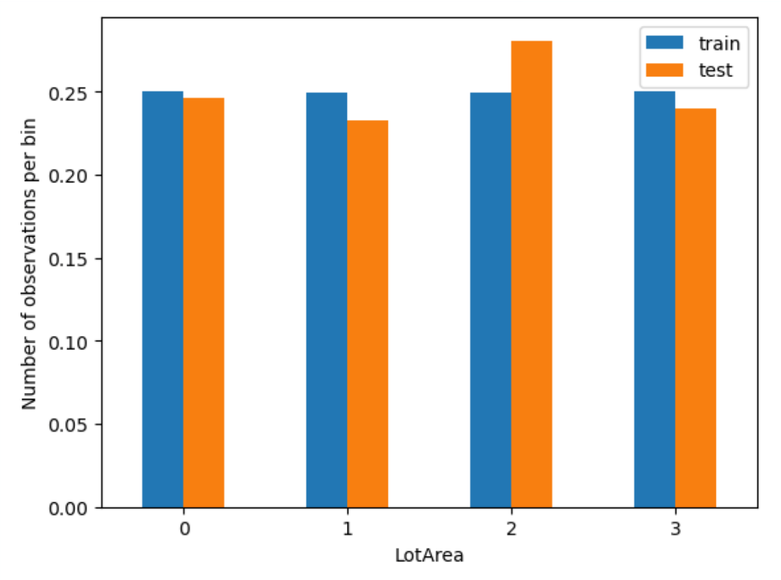
```
equal_frequency = EqualFrequencyDiscretiser(q = 4, variables = continous_features_train)

equal_frequency.fit(X_train_impute_enc)

X_train_equal_frequency = equal_frequency.transform(X_train_impute_enc)

X_test_equal_frequency = equal_frequency.transform(X_test_impute_enc)
```

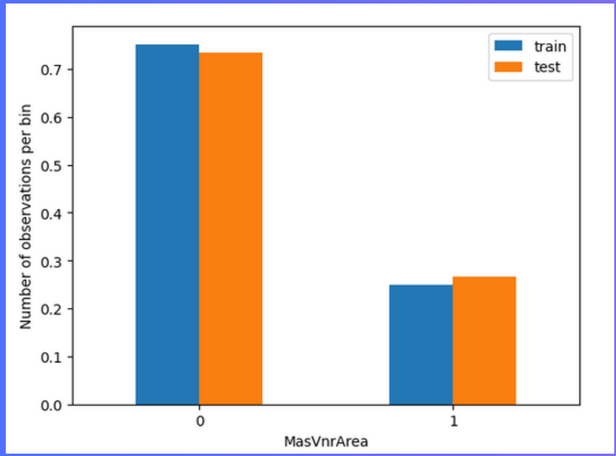
```
for var in continous_features_train:
    check_balance_of_discrete_and_categorical(
        X_train_equal_frequency,
        X_test_equal_frequency,
        var
    )
```

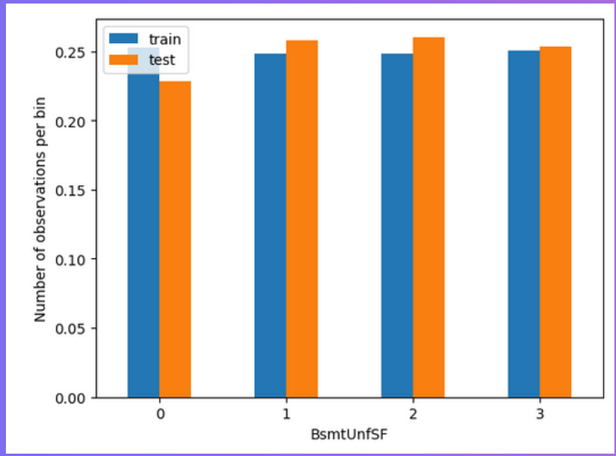
Check balancing of data between train and test sets

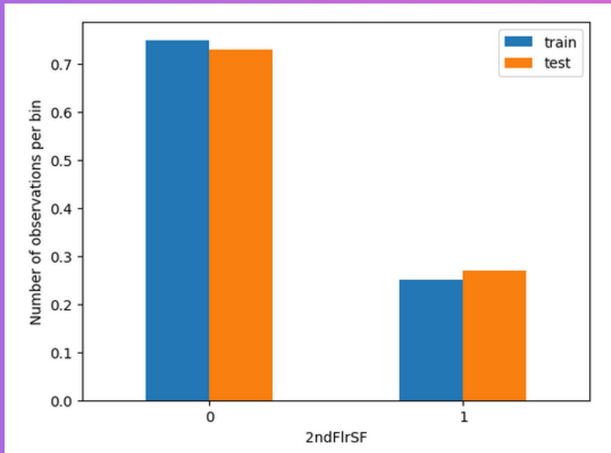


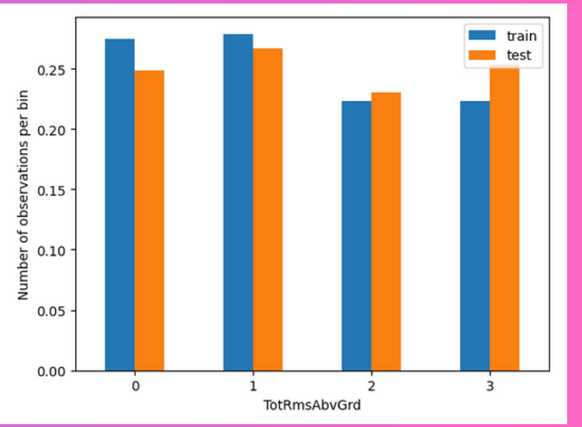


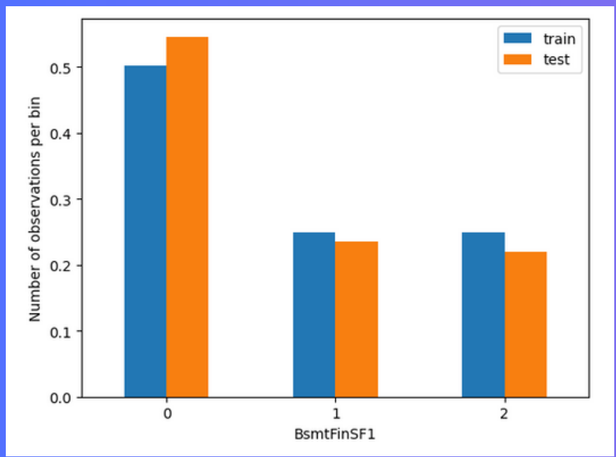


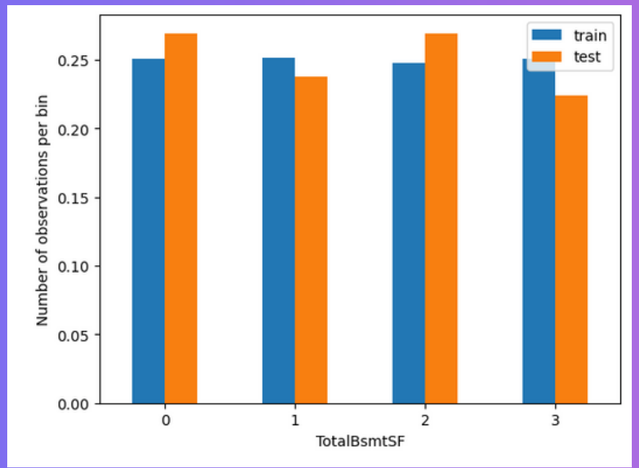

train
test
Number of observations per bin
MasVnrArea

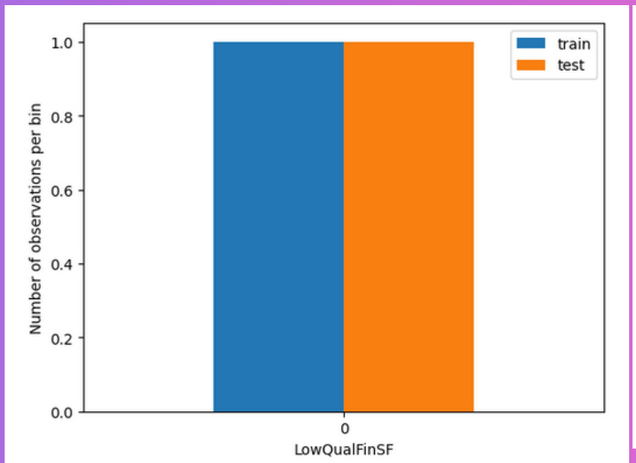

train
test
Number of observations per bin
BsmtUnfSF

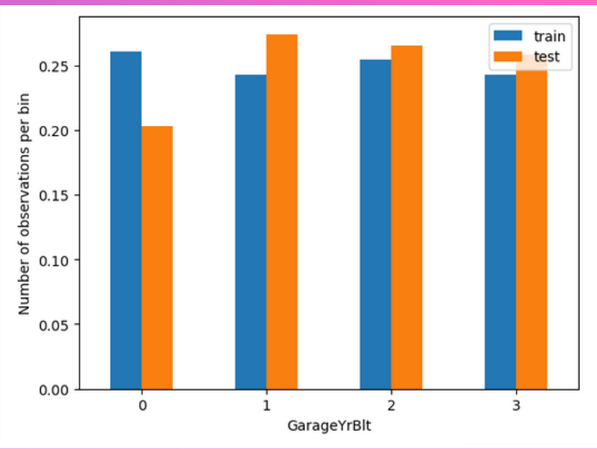

train
test
Number of observations per bin
2ndFlrSF

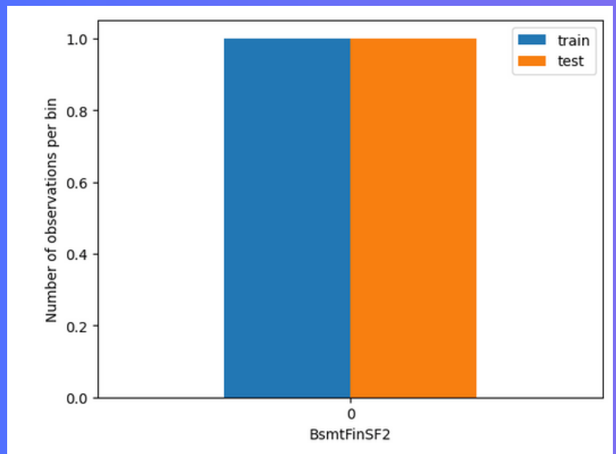

train
test
Number of observations per bin
TotRmsAbvGrd

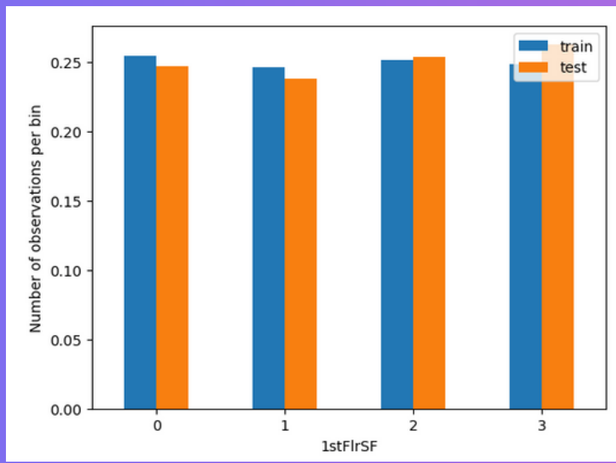

train
test
Number of observations per bin
BsmtFinSF1

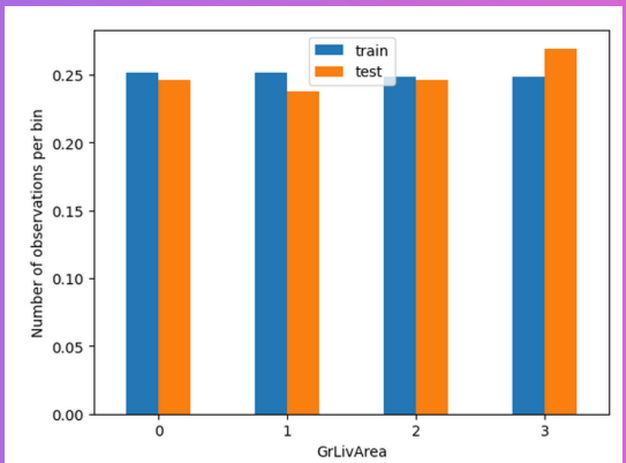

train
test
Number of observations per bin
TotalBsmtSF

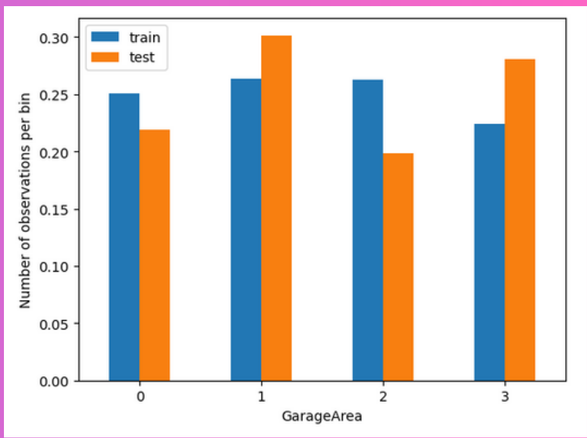

train
test
Number of observations per bin
LowQualFinSF


train
test
Number of observations per bin
GarageYrBlt


train
test
Number of observations per bin
BsmtFinSF2

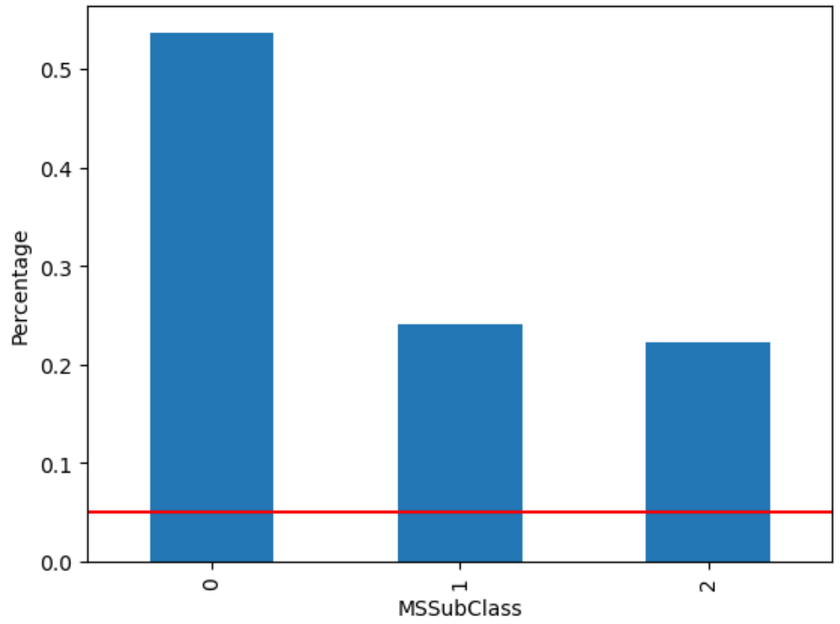

train
test
Number of observations per bin
1stFlrSF


train
test
Number of observations per bin
GrLivArea


train
test
Number of observations per bin
GarageArea

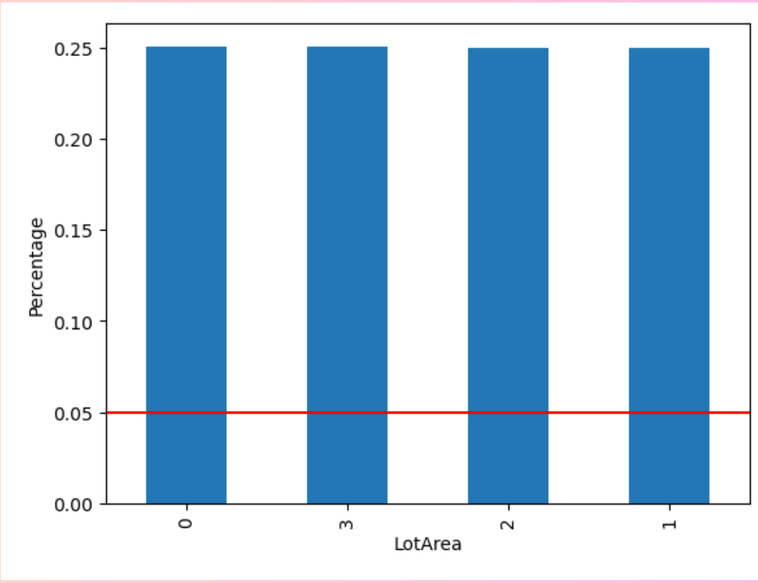
Rare labels analysis

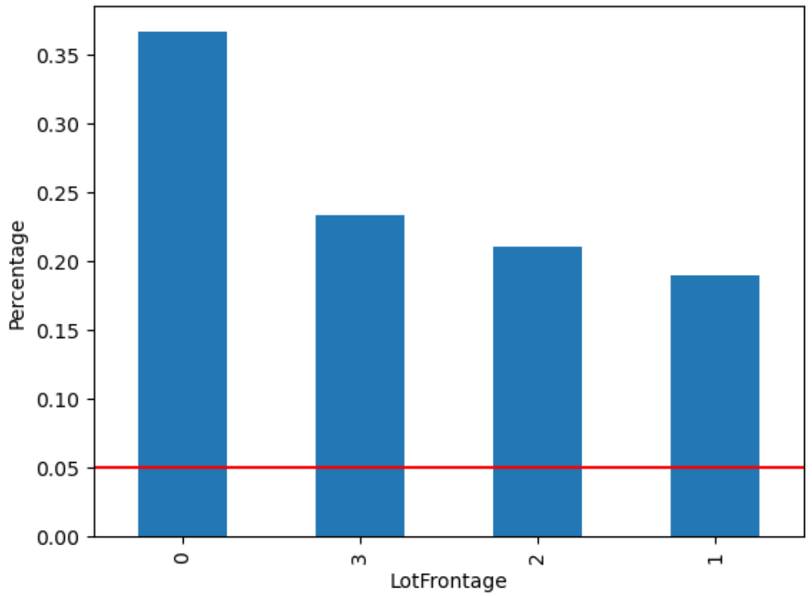
```
[ ]: 1 check_rare_labels_analysis(X_train_equal_frequency, continous_features_train)
```

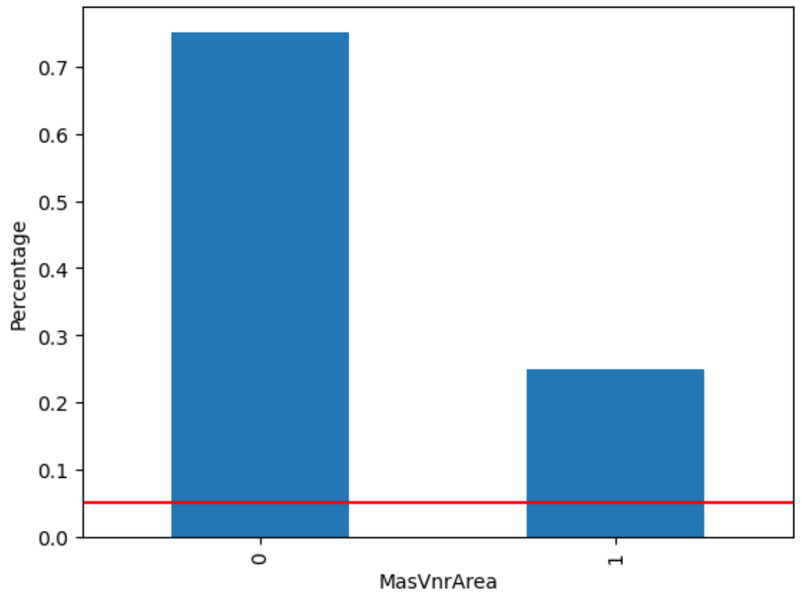


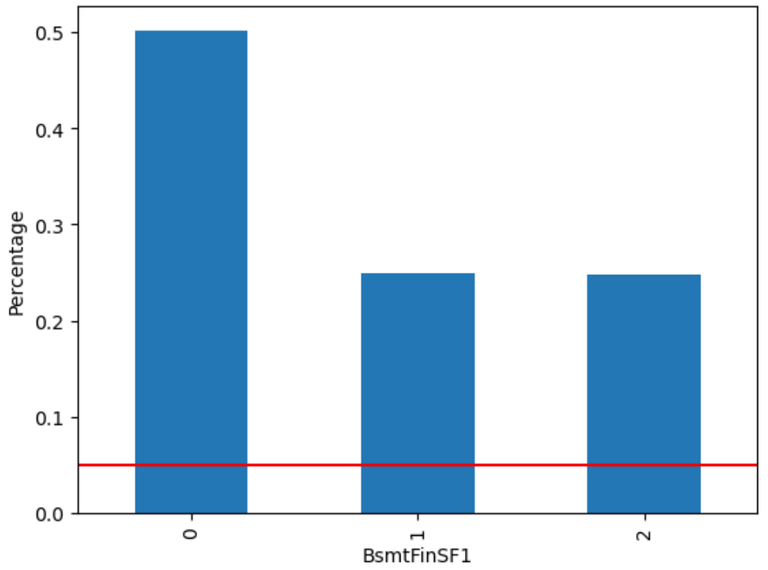
Rare label Analysis

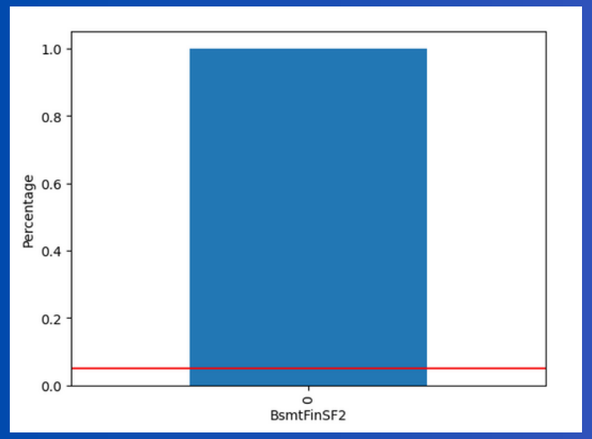
Check rare label for X_train_equal_frequency

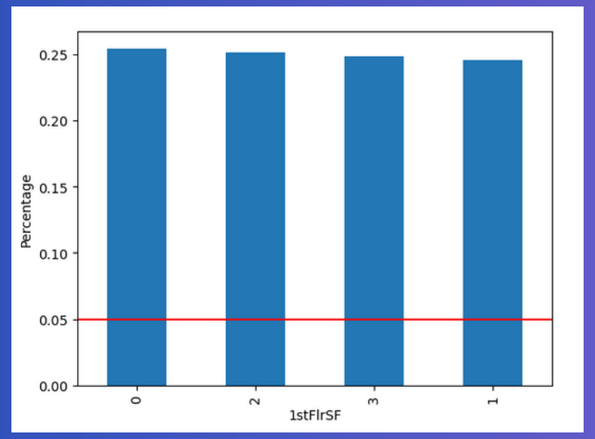


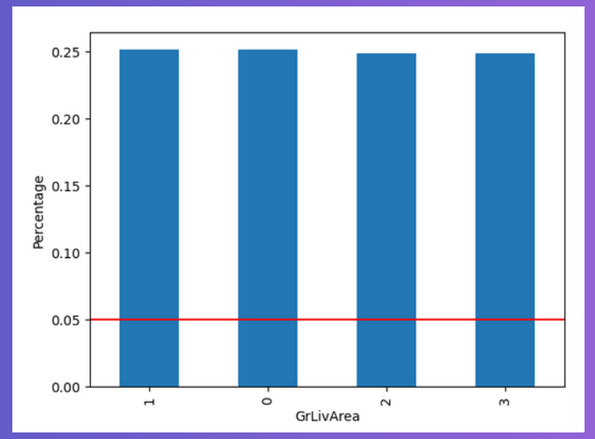


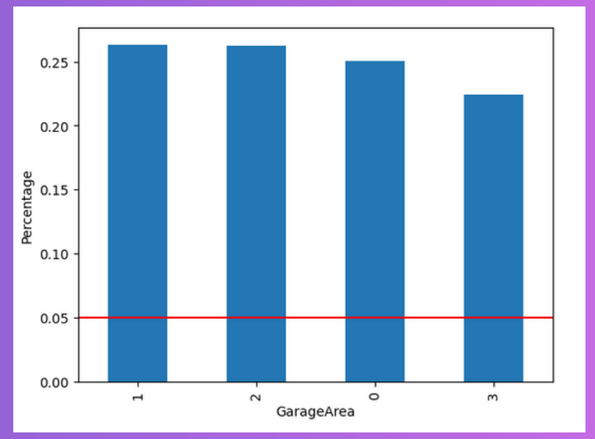


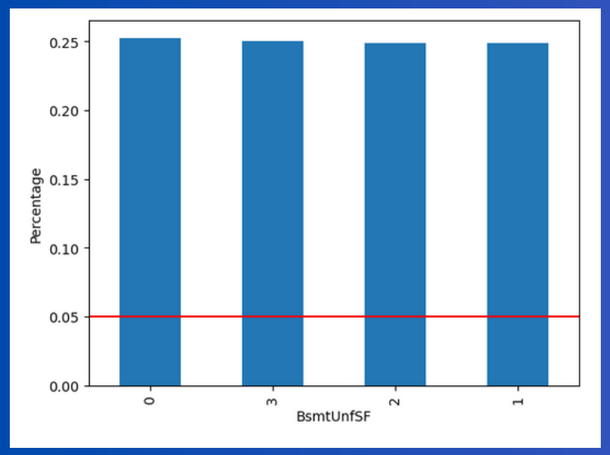


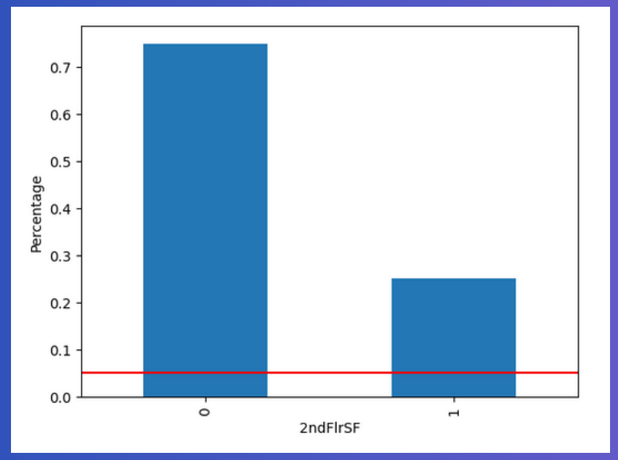

Percentage
BsmtFinSF2

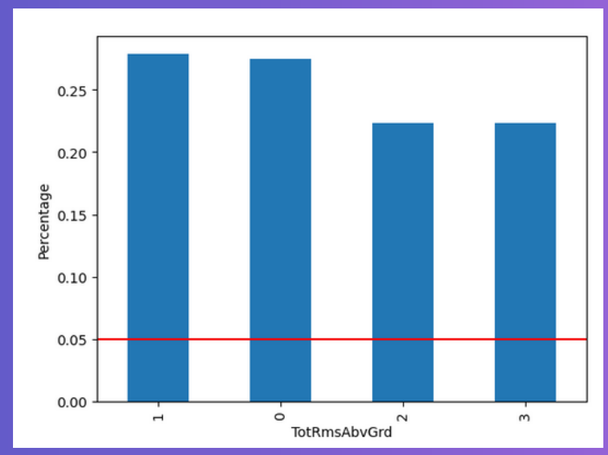

Percentage
1stFlrSF

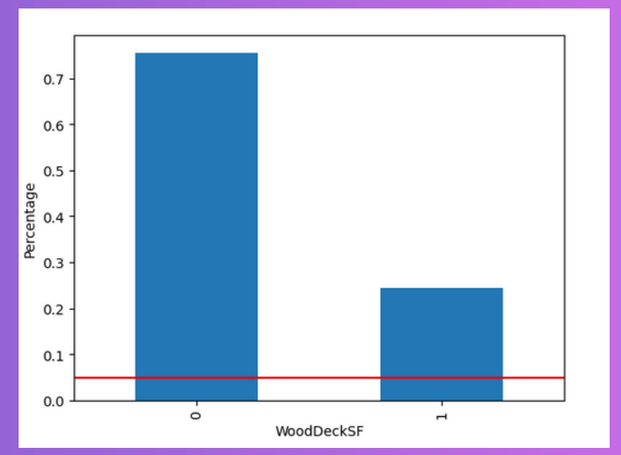

Percentage
GrLivArea

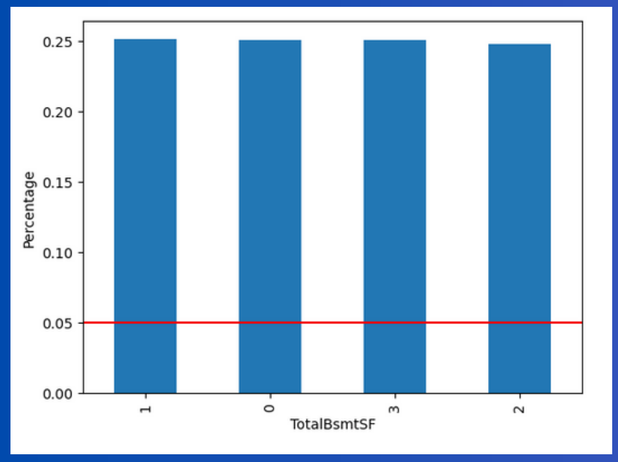

Percentage
GarageArea

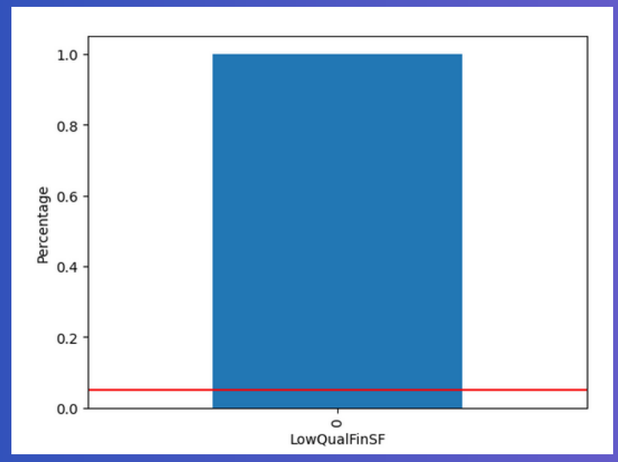

Percentage
BsmtUnfSF

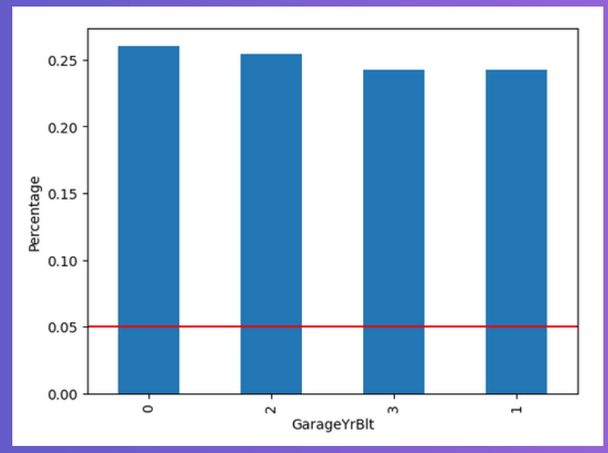

Percentage
2ndFlrSF

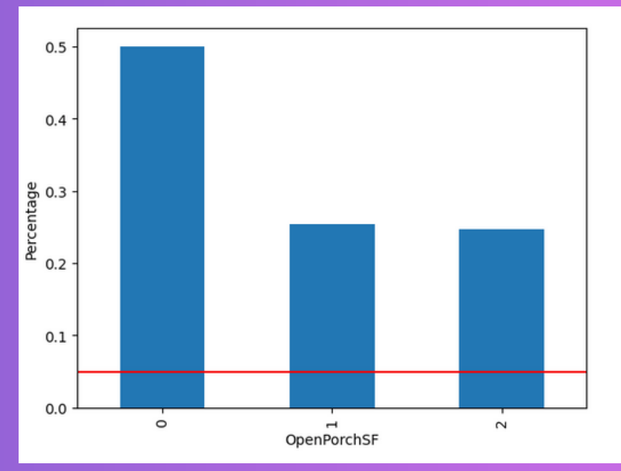

Percentage
TotRmsAbvGrd


Percentage
WoodDeckSF


Percentage
TotalBsmtSF


Percentage
LowQualFinSF

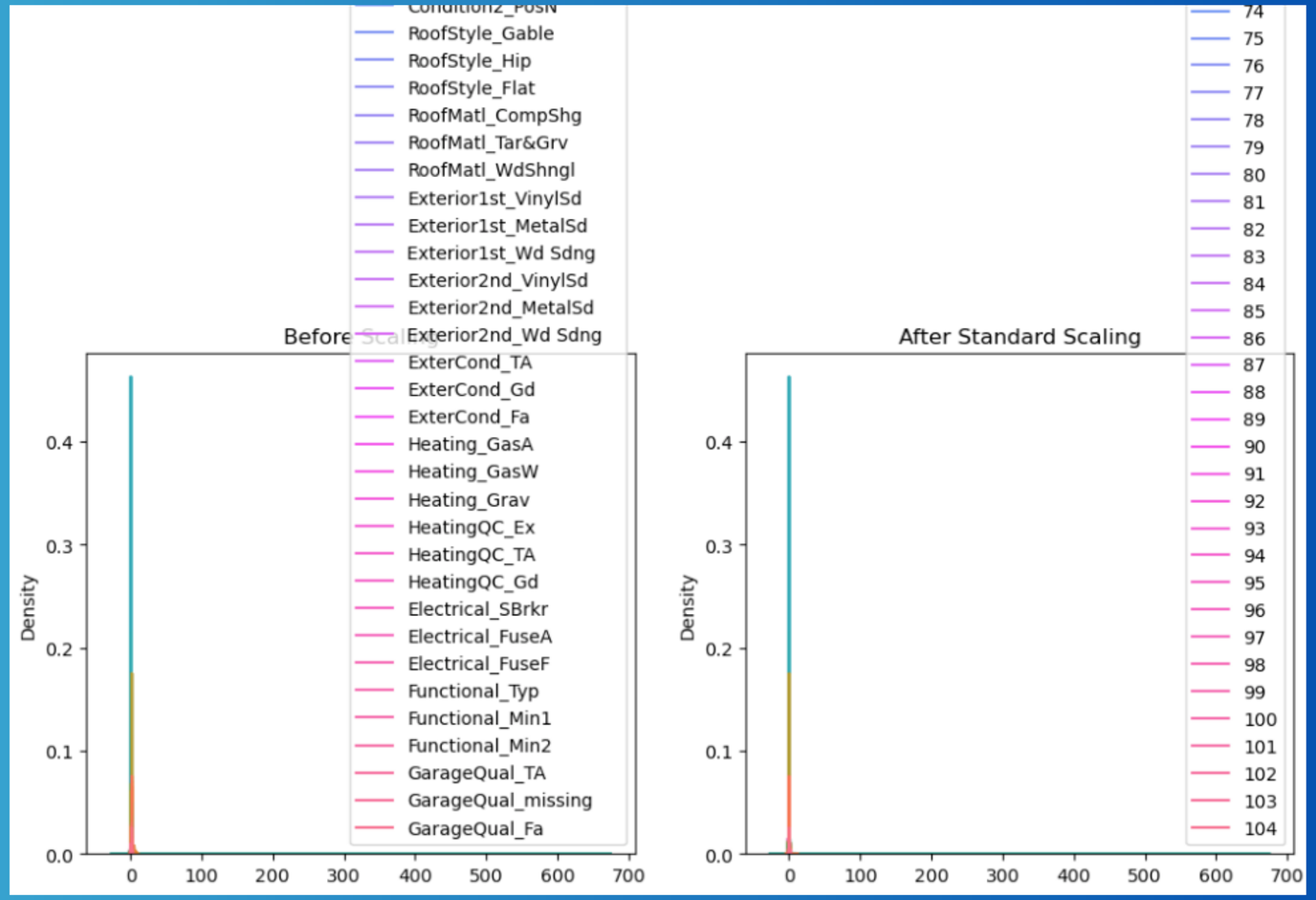

Percentage
GarageYrBlt


Percentage
OpenPorchSF

Feature Magnitude
Robust Scaling

```
scaler_robust = RobustScaler()

scaler_robust.fit(X_train_equal_frequency)

X_train_scaled_robust = scaler_robust.transform(X_train_equal_frequency)
X_test_scaled_robust = scaler_robust.transform(X_test_equal_frequency)
```

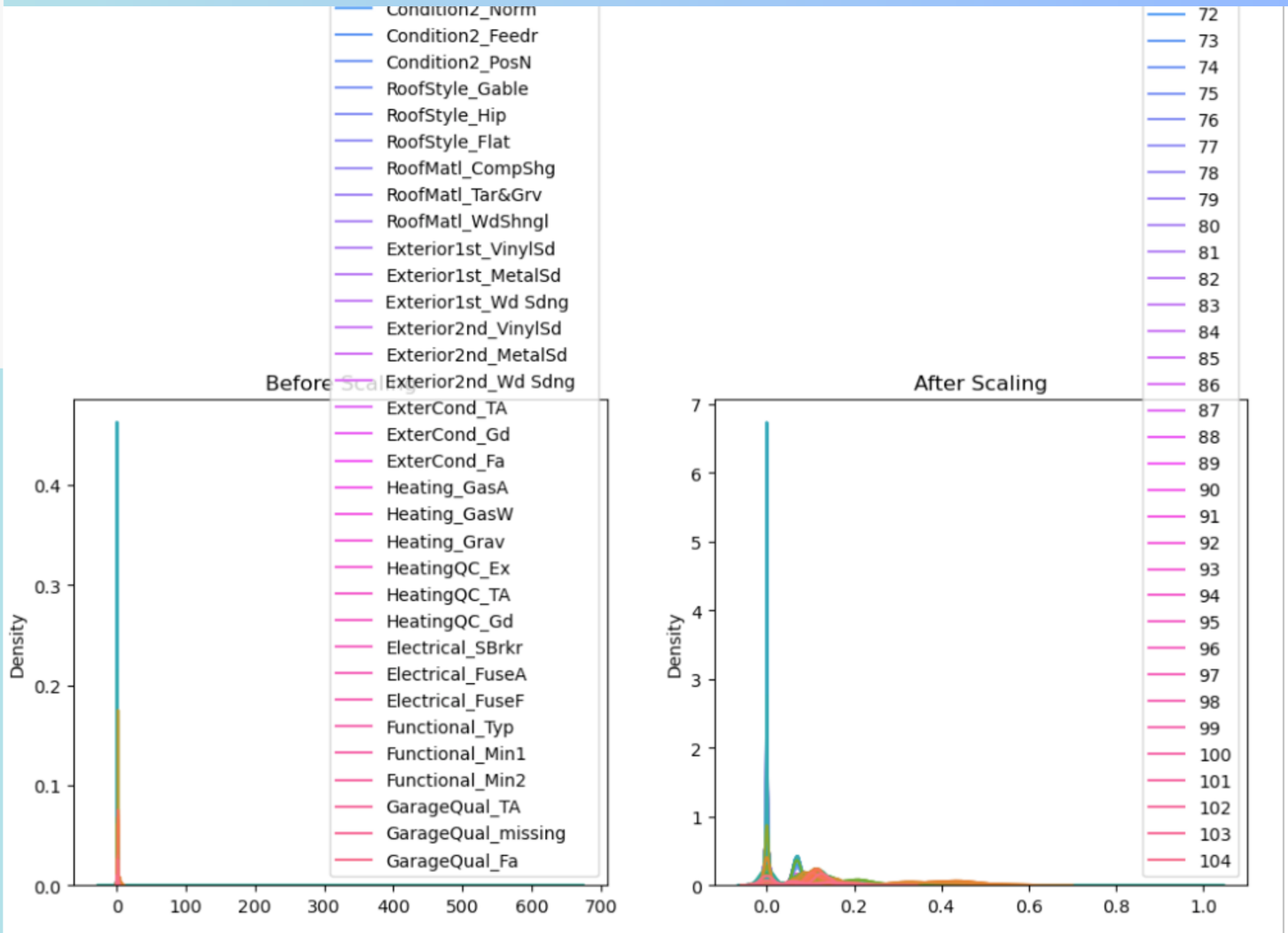
```
fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize =(12, 5))

#before scaling

ax1.set_title('Before Scaling')
sns.kdeplot(X_train_equal_frequency, ax = ax1, warn_singular = False)
sns.kdeplot(X_train_equal_frequency, ax = ax1, warn_singular = False)
sns.kdeplot(X_train_equal_frequency, ax = ax1, warn_singular = False)

#after Scaling
ax2.set_title('After Standard Scaling')
sns.kdeplot(X_train_scaled_robust, ax = ax2, warn_singular = False)
sns.kdeplot(X_train_scaled_robust, ax = ax2, warn_singular = False)
sns.kdeplot(X_train_scaled_robust, ax = ax2, warn_singular = False)

plt.show()
```



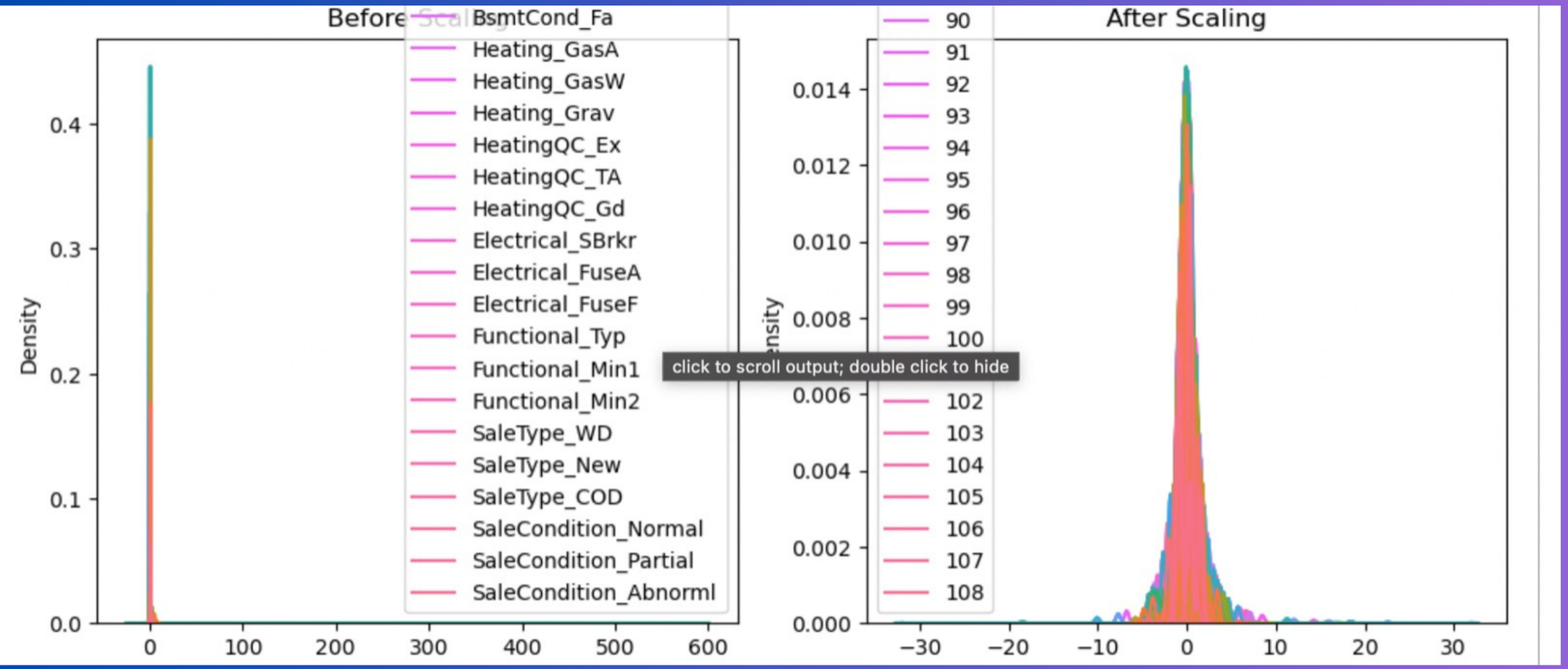
Unit-Norm

```
scaler_unit_norm = Normalizer()
scaler_unit_norm.fit(X_train_equal_frequency)

X_train_scaled_unit_norm = scaler_unit_norm.transform(X_train_equal_frequency)
```

```
fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize =(12, 5))

#before scaling

ax1.set_title('Before Scaling')
sns.kdeplot(X_train_equal_frequency, ax = ax1, warn_singular = False)
sns.kdeplot(X_train_equal_frequency, ax = ax1, warn_singular = False)
sns.kdeplot(X_train_equal_frequency, ax = ax1, warn_singular = False)

#after Scaling
ax2.set_title('After Scaling')
sns.kdeplot(X_train_scaled_unit_norm, ax = ax2, warn_singular = False)
sns.kdeplot(X_train_scaled_unit_norm, ax = ax2, warn_singular = False)
sns.kdeplot(X_train_scaled_unit_norm, ax = ax2, warn_singular = False)

plt.show()
```



Standartization



5. Feature-selection

Constant & Quasi-constant features

```
sel_const_variables = DropConstantFeatures(tol = 0.75, variables = None, missing_values = 'raise')

sel_const_variables.fit(X_train_equal_frequency)
```

```
DropConstantFeatures(tol=0.75)
```

```
sel_const_variables.features_to_drop_
```

```
['MSZoning',
 'LandContour',
 'BldgType',
 'MasVnrArea',
 'BsmtCond',
 'BsmtFinType2',
 'BsmtFinSF2',
 'LowQualFinSF',
 'BsmtHalfBath',
 'KitchenAbvGr',
 'GarageCond',
 'PavedDrive',
 'WoodDeckSF',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'PoolArea',
 'MiscVal',
 'SaleType',
 'SaleCondition',
 'Street_Pave',
 'Street_Grvl',
 'Utilities_AllPub',
 'Utilities_NoSeWa',
 'LandSlope_Gtl',
 'LandSlope_Sev',
 'LandSlope_Mod',
 'CentralAir_Y',
 'CentralAir_N',
 'Neighborhood_NAmes',
 'Neighborhood_CollgCr',
 'Neighborhood_OldTown',
 'Condition1_Norm',
 'Condition1_Feedr',
 'Condition1_Artery',
 'Condition2_Norm',
 'Condition2_Feedr',
 'Condition2_PosN',
 'RoofStyle_Gable',
 'RoofStyle_Hip',
 'RoofStyle_Flat',
 'RoofMatl_CompShg',
 'RoofMatl_Tar&Grv',
 'RoofMatl_WdShngl',
 'Exterior1st_MetalSd',
 'Exterior1st_Wd Sdng',
 'Exterior2nd_MetalSd',
 'Exterior2nd_Wd Sdng',
 'ExterCond_TA',
 'ExterCond_Gd',
 'ExterCond_Fa',
 'Heating_GasA',
 'Heating_GasW',
 'Heating_Grav',
 'HeatingQC_Gd',
 'Electrical_SBrkr',
 'Electrical_FuseA',
 'Electrical_FuseF',
 'Functional_Typ',
 'Functional_Min1',
 'Functional_Min2',
 'GarageQual_TA',
 'GarageQual_missing',
 'GarageQual_Fa']
```

Feature Selection Constant and Quasi-Constant Features

Duplicated features

```
1 sel_duplicated_variables = DropDuplicateFeatures(variables = None, missing_values = 'raise')
2
3 sel_duplicated_variables.fit(X_train_equal_frequency)
```

```
DropDuplicateFeatures
DropDuplicateFeatures(missing_values='raise')
```

```
1 sel_duplicated_variables.features_to_drop_
```

```
{'3SsnPorch', 'EnclosedPorch', 'LowQualFinSF', 'MiscVal', 'ScreenPorch'}
```

Duplicated Features

Correlated features

```
sel_correlated_variables_variance = SmartCorrelatedSelection(
    variables = None,
    method = 'pearson',
    threshold = 0.8,
    missing_values = 'raise',
    selection_method = 'variance',
    estimator = None,
    scoring = 'accuracy',
    cv = 5
)

sel_correlated_variables_variance.fit(X_train_equal_frequency)
```

SmartCorrelatedSelection

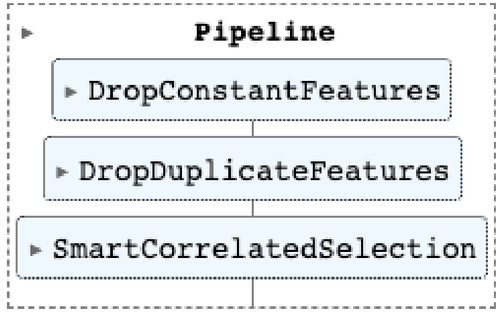
```
SmartCorrelatedSelection(cv=5, missing_values='raise', scoring='accuracy',
                         selection_method='variance')
```

Correlated Features

```
sel_correlated_variables_variance.features_to_drop_
```

```
['FireplaceQu',
 'SaleType',
 'Street_Pave',
 'Utilities_AllPub',
 'LandSlope_Mod',
 'CentralAir_Y',
 'RoofStyle_Hip',
 'RoofMatl_Tar&Grv',
 'Exterior2nd_VinylSd',
 'Exterior2nd_MetalSd',
 'Exterior2nd_Wd Sdng',
 'ExterCond_Gd',
 'Electrical_FuseA',
 'GarageQual_missing']
```

```
pipe_basic_filter = Pipeline(steps = [
    ('const_and_quasi-const', DropConstantFeatures(tol = 0.8, variables = None, missing_values = 'raise')),
    ('duplicated', DropDuplicateFeatures(variables = None, missing_values = 'raise')),
    ('correlation', SmartCorrelatedSelection(
    variables = None,
    method = 'pearson',
    threshold = 0.8,
    missing_values = 'raise',
    selection_method = 'variance',
    estimator = None,
    scoring = 'accuracy',
    cv = 5))
])

pipe_basic_filter.fit(X_train_equal_frequency)
```



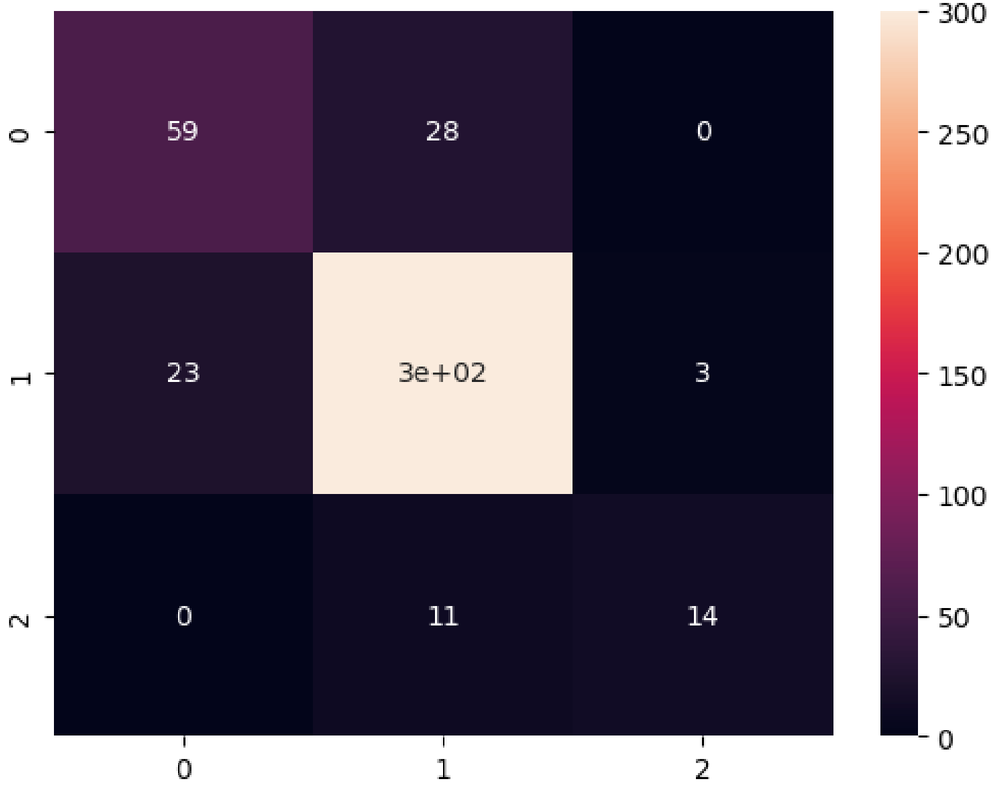

```
X_train_basic_filtered = pipe_basic_filter.transform(X_train_equal_frequency)
X_test_basic_filtered = pipe_basic_filter.transform(X_test_equal_frequency)
```

```
X_train_basic_filtered.shape, X_test_basic_filtered.shape
```

```
((1022, 43), (438, 43))
```

Build basic filter methods with PipeLine

Metrics Score
Decision Tree




```
best_model_dt = grid_search_dt.best_estimator_

best_model_dt.fit(X_train, y_train)

metrics_score(X_train, X_test, y_train, y_test, best_model_dt)
```

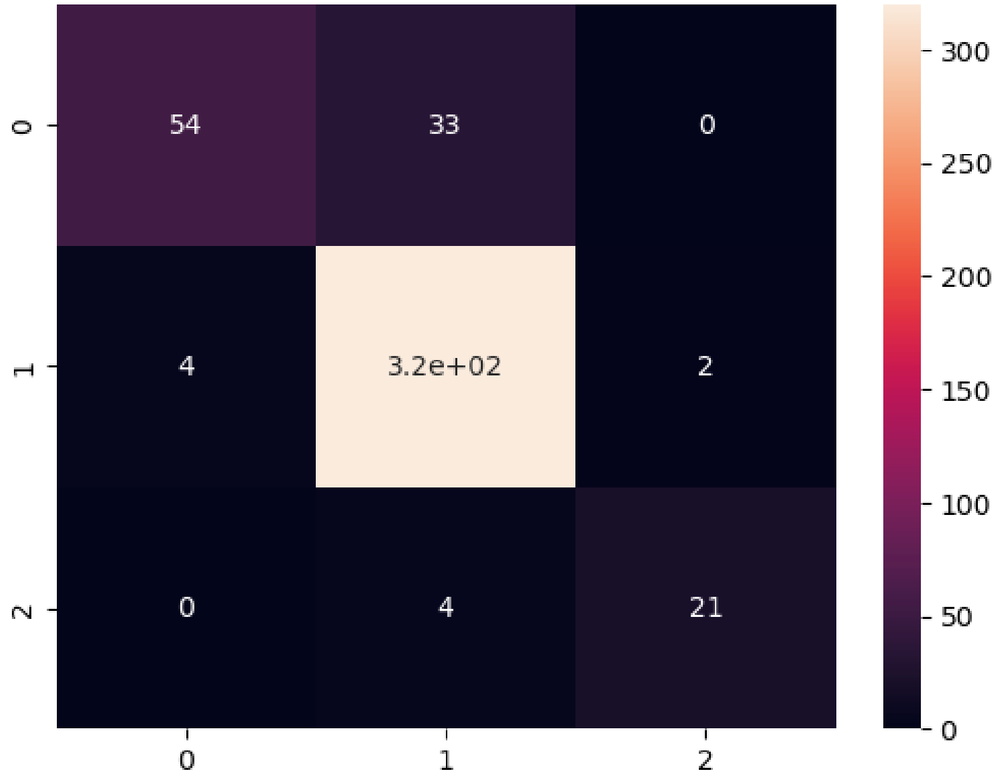
```
Accuracy score on train set: 94.814 %

Accuracy score on test set: 85.160 %
```

Classification report on train set

	precision	recall	f1-score	support
0	0.72	0.68	0.70	87
1	0.88	0.92	0.90	326
2	0.82	0.56	0.67	25
accuracy			0.85	438
macro avg	0.81	0.72	0.76	438
weighted avg	0.85	0.85	0.85	438

```
grid_search_dt.best_params_
```

```
{'classifier__criterion': 'gini',
 'classifier__max_depth': 10,
 'classifier__max_features': 'log2',
 'classifier__splitter': 'best',
 'disc__q': 6,
 'median__imputation_method': 'median',
 'ohe_top_cat__top_categories': 3,
 'random_sample__random_state': 10}
```



Classification report on train set

	precision	recall	f1-score	support
0	0.93	0.62	0.74	87
1	0.90	0.98	0.94	326
2	0.91	0.84	0.87	25
accuracy			0.90	438
macro avg	0.91	0.81	0.85	438
weighted avg	0.90	0.90	0.90	438

Metrics score

```
1  best_model_rf = grid_search_rf.best_estimator_
2
3  best_model_rf.fit(X_train, y_train)
4
5  metrics_score(X_train, X_test, y_train, y_test, best_model_rf)
```

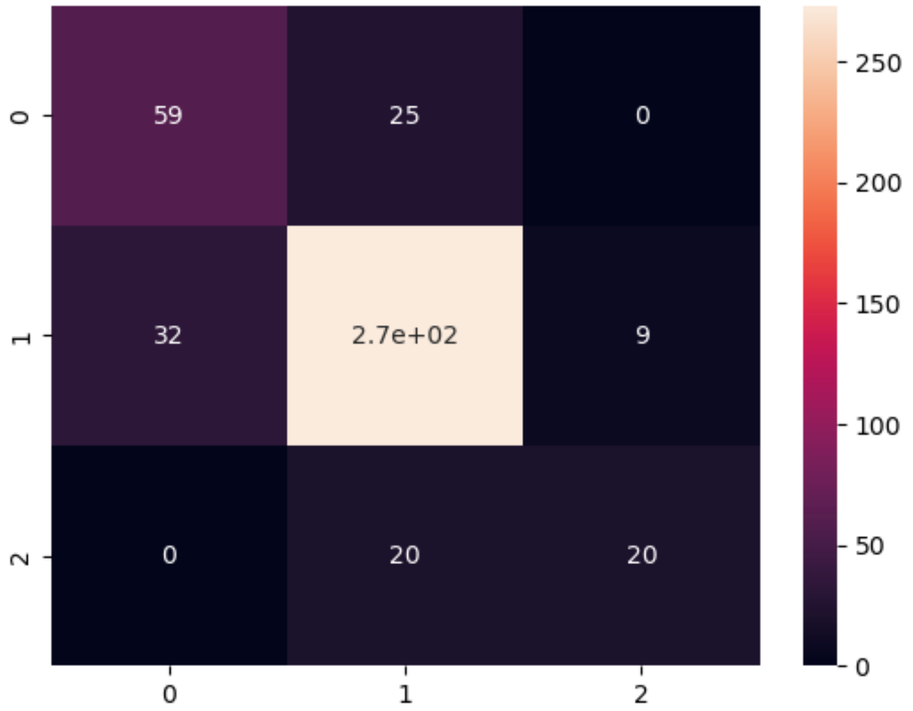
```
Accuracy score on train set: 99.315 %

Accuracy score on test set: 90.183 %
```

Metric score

Random Forest

```
1  grid_search_rf.best_params_
```

```
{'classifier__criterion': 'log_loss',
 'classifier__max_depth': 9,
 'classifier__n_estimators': 200,
 'disc__q': 8,
 'median__imputation_method': 'median'}
```



Classification report on train set

	precision	recall	f1-score	support
0	0.65	0.70	0.67	84
1	0.86	0.87	0.86	314
2	0.69	0.50	0.58	40
accuracy			0.80	438
macro avg	0.73	0.69	0.71	438
weighted avg	0.80	0.80	0.80	438

```
1 grid_search_ada.best_params_
```

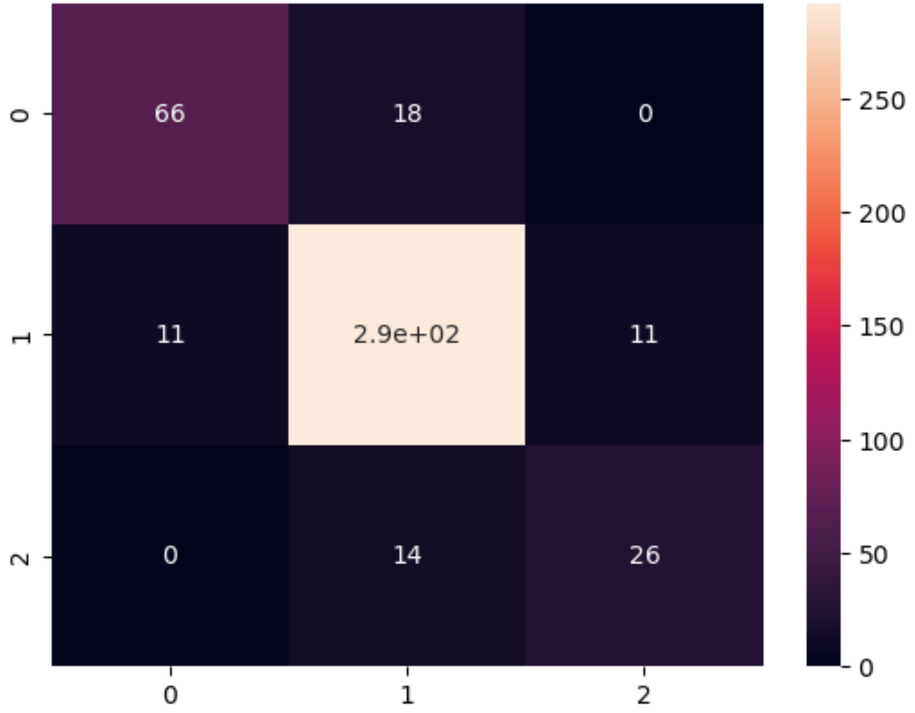
```
{'classifier__learning_rate': 0.5,
 'classifier__n_estimators': 100,
 'disc__q': 8,
 'median__imputation_method': 'median',
 'ohe_top_cat__top_categories': 3}
```

Metric score

Ada Boost Classifier

```
1 best_model_ada = grid_search_ada.best_estimator_
2
3 best_model_ada.fit(X_train, y_train)
4
5 metrics_score(X_train, X_test, y_train, y_test, best_model_ada)
```

```
Accuracy score on train set: 80.920 %

Accuracy score on test set: 80.365 %
```




```
best_model_logit = grid_search_logit.best_estimator_

best_model_logit.fit(X_train, y_train)

metrics_score(X_train, X_test, y_train, y_test, best_model_logit)
```

```
Accuracy score on train set: 91.781 %

Accuracy score on test set: 87.671 %
```

Metric score

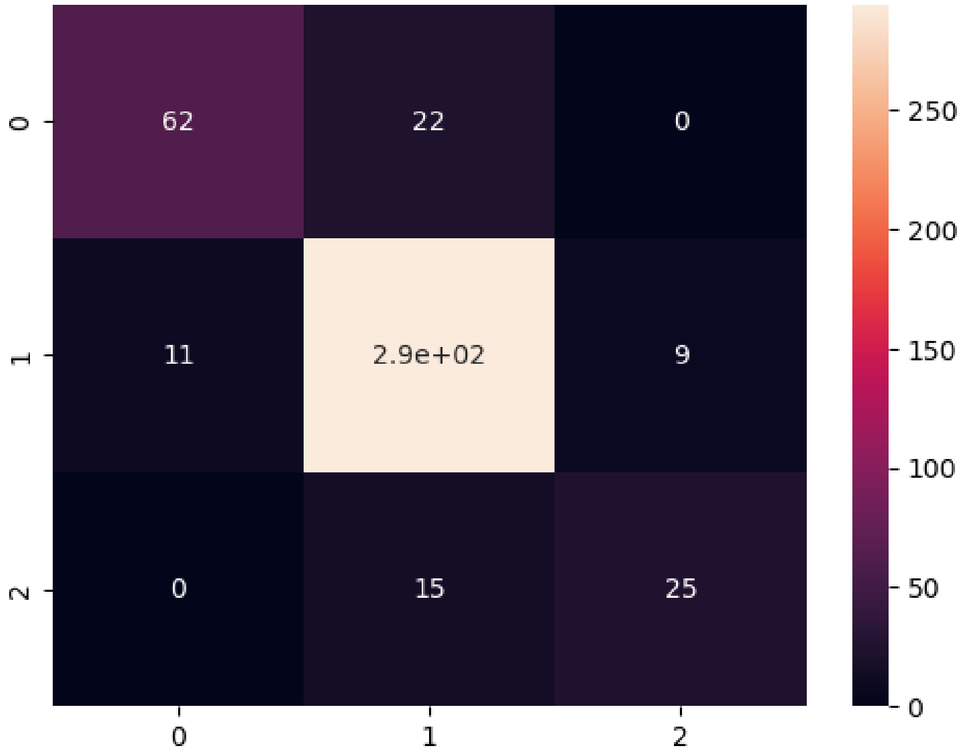
Logistic Regression

Classification report on train set

	precision	recall	f1-score	support
0	0.86	0.79	0.82	84
1	0.90	0.93	0.92	314
2	0.70	0.65	0.68	40
accuracy			0.88	438
macro avg	0.82	0.79	0.80	438
weighted avg	0.87	0.88	0.88	438

```
grid_search_logit.best_params_
```

```
{'classifier__C': 0.6,
 'classifier__max_iter': 1000,
 'disc__q': 4,
 'median__imputation_method': 'mean',
 'ohe_top_cat__top_categories': 3}
```



Classification report on train set

	precision	recall	f1-score	support
0	0.85	0.74	0.79	84
1	0.89	0.94	0.91	314
2	0.74	0.62	0.68	40
accuracy			0.87	438
macro avg	0.82	0.77	0.79	438
weighted avg	0.87	0.87	0.87	438

```
1 best_model_svc = grid_search_svc.best_estimator_
2
3 best_model_svc.fit(X_train, y_train)
4
5 metrics_score(X_train, X_test, y_train, y_test, best_model_svc)
```

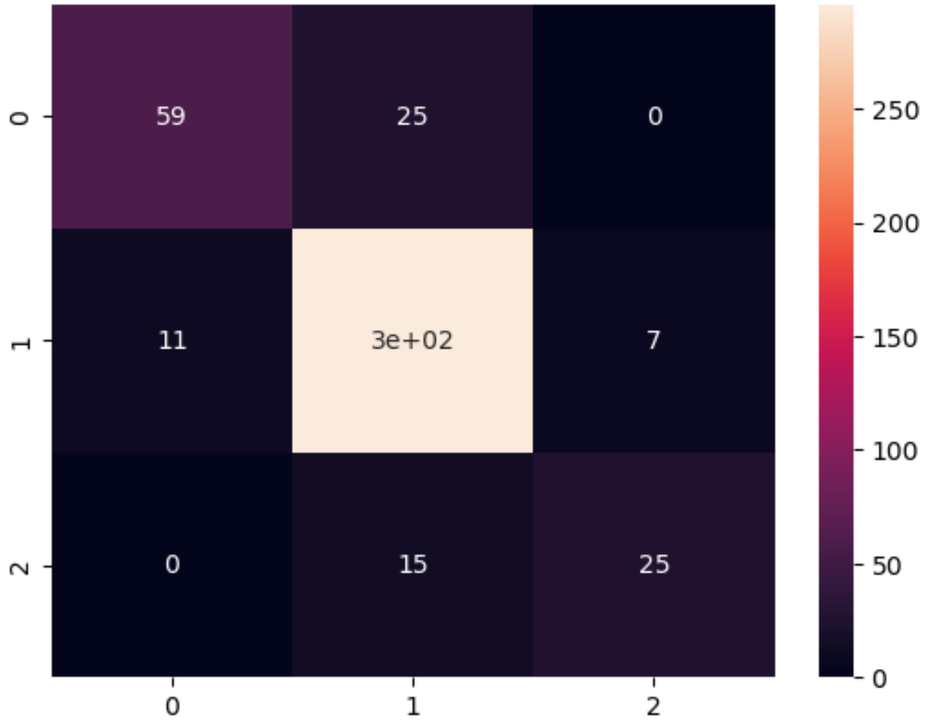
```
Accuracy score on train set: 92.074 %

Accuracy score on test set: 86.986 %
```

Metric score

Support vector classifier

```
1 grid_search_svc.best_params_
```

```
{'classifier__C': 0.1,
 'classifier__gamma': 1,
 'classifier__kernel': 'linear',
 'disc__q': 8,
 'median__imputation_method': 'median',
 'ohe_top_cat__top_categories': 3}
```



Classification report on train set

	precision	recall	f1-score	support
0	0.84	0.70	0.77	84
1	0.88	0.94	0.91	314
2	0.78	0.62	0.69	40
accuracy			0.87	438
macro avg	0.84	0.76	0.79	438
weighted avg	0.86	0.87	0.86	438

```
1  grid_search_xgb.best_params_
```

```
{'classifier__gamma': 0,
 'classifier__learning_rate': 0.1,
 'classifier__max_depth': 1,
 'classifier__n_estimators': 400,
 'disc__q': 2,
 'median__imputation_method': 'mean',
 'ohe_top_cat__top_categories': 5}
```

Metric Score
XGBoost classifier

```
1  best_model_xgb = grid_search_xgb.best_estimator_
2
3  best_model_xgb.fit(X_train, y_train)
4
5  metrics_score(X_train, X_test, y_train, y_test, best_model_xgb)
```

```
Accuracy score on train set: 91.487 %

Accuracy score on test set: 86.758 %
```