Reading CSV file as a Dataframe

```
import pandas as pd
df1 = pd.read_csv('/data/notebook_files/train.csv')
df2 = pd.read_csv('/data/notebook_files/test.csv')
combined_df = pd.concat([df1,df2], ignore_index = True)
pd.set_option('display.max_columns', None)
combined_df.head(5)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotCo
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corne
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2

```
df_SalePriceAscending = combined_df.sort_values(by='SalePrice', ascending=Fal
df_SalePriceAscending.head(5)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	Lo
691	692	60	RL	104.0	21535	Pave	NaN	IR1	Lvl	AllPub	Cc
1182	1183	60	RL	160.0	15623	Pave	NaN	IR1	Lvl	AllPub	Cc
1169	1170	60	RL	118.0	35760	Pave	NaN	IR1	Lvl	AllPub	Cu
898	899	20	RL	100.0	12919	Pave	NaN	IR1	Lvl	AllPub	Ins
803	804	60	RL	107.0	13891	Pave	NaN	Reg	Lvl	AllPub	Ins

Exploratory Data Analysis and data cleaning

```
# Checking and displaying empty values
empty_values = combined_df.isnull().sum()

print("Columns with empty values:")
print(empty_values[empty_values > 0])
```

```
Columns with empty values:
MSZoning           4
LotFrontage      486
Alley           2721
Utilities          2
Exterior1st        1
Exterior2nd        1
MasVnrType        24
MasVnrArea        23
BsmtQual          81
BsmtCond          82
BsmtExposure      82
BsmtFinType1      79
BsmtFinSF1         1
BsmtFinType2      80
BsmtFinSF2         1
BsmtUnfSF          1
TotalBsmtSF        1
Electrical         1
BsmtFullBath       2
```

I decided to filter out columns with high missing values and 0 values which do not fit the dataset. Combined_df consists of duplicate data and data with high correlation (over standardized correlation) and was not useful. Therefore, after looking at all the data and reading different sources, I've decided that columns with 90% NAN values would be deleted along with columns where there is a correlation greater than or equal to 92%

```
import pandas as pd

# percentage of missing values for each column
missing_percentages = (combined_df.isnull().sum() / len(combined_df)) * 100

# columns with more than 90% missing values
columns_with_high_missing_values = missing_percentages[missing_percentages >

print("Columns with more than 90% missing values:")
for column in columns_with_high_missing_values:
    print(column)
```

```
Columns with more than 90% missing values:
Alley
```

```
PoolQC
```

```python
import pandas as pd

# columns with more than 90% missing values
threshold = 0.9
columns_to_drop_threshold = combined_df.columns[combined_df.isnull().mean() >

print("Columns with more than 90% missing values:")
print(columns_to_drop_threshold)

# Drop columns with more than 90% missing values
combined_df = combined_df.loc[:, combined_df.isnull().mean() < threshold]


# correlation
unique_threshold = 0.08  # 92% of the datapoints in the column are the same
columns_to_drop_unique = combined_df.columns[combined_df.nunique() / len(comb

print("Columns with a high amount of the same value:")
print(columns_to_drop_unique)

# Drop columns with 92% of the same value
combined_df = combined_df.loc[:, combined_df.nunique() / len(combined_df) >=
combined_df.head(5)
```

```
Columns with more than 90% missing values:
Index(['Alley', 'PoolQC', 'MiscFeature'], dtype='object')
Columns with a high amount of the same value:
Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'Street', 'LotShape',
       'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood'
       'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl',
       'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond'
       'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1'
       'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical',
       'LowQualFinSF', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBat
       'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
       'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType', 'GarageYrBl
       'GarageFinish', 'GarageCars', 'GarageQual', 'GarageCond', 'PavedDriv
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'Fence',
       'MiscVal', 'MoSold', 'YrSold', 'SaleType', 'SaleCondition'],
      dtype='object')
```

	Id	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	GrLivAr

Prediction column is 'SalePrice'. Diving deeper into this columns data

```
combined_df['SalePrice'].describe()
```

```
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
```

Visualizations

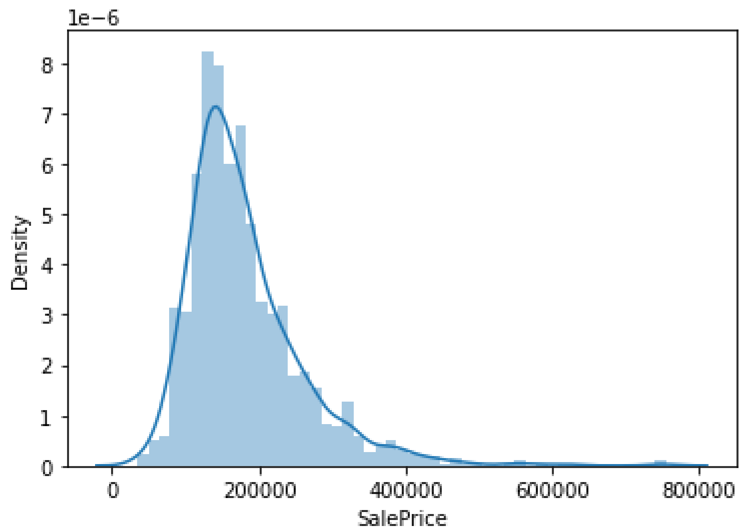
```
combined_df.describe()
```

	Id	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF
count	2919.000000	2919.000000	2896.000000	2918.000000	2918.000000	2918.000000	2918.00000
mean	1460.000000	10168.114080	102.201312	441.423235	49.582248	560.772104	1051.77758
std	842.787043	7886.996359	179.334253	455.610826	169.205611	439.543659	440.76625
min	1.000000	1300.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	730.500000	7478.000000	0.000000	0.000000	0.000000	220.000000	793.00000
50%	1460.000000	9453.000000	0.000000	368.500000	0.000000	467.000000	989.50000
75%	2189.500000	11570.000000	164.000000	733.000000	0.000000	805.500000	1302.00000
max	2919.000000	215245.000000	1600.000000	5644.000000	1526.000000	2336.000000	6110.00000

```
import seaborn as sns

sns.distplot(combined_df['SalePrice'])
```

```
<Axes: xlabel='SalePrice', ylabel='Density'>
```

Download




```
<ipython-input-8-d3ebcf3e6407>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function wit
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(combined_df['SalePrice'])
```

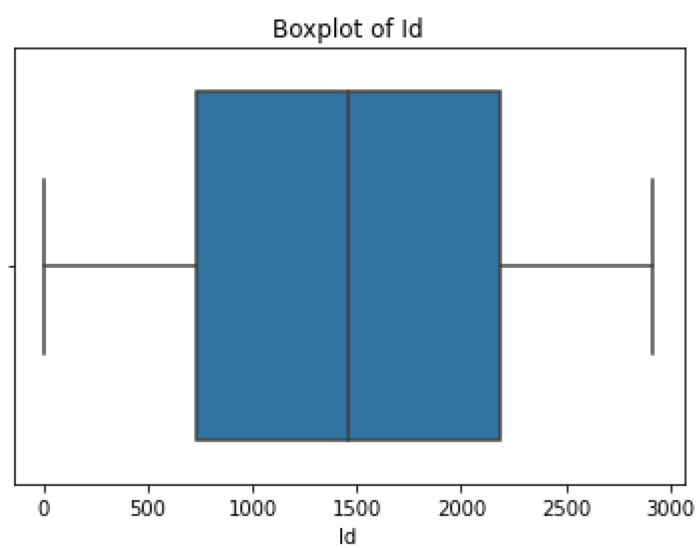
```
# noticed that the data is skewed slightly right, peaking at around 160000, 6
combined_df['SalePrice'].skew()
```

```
1.8828757597682129
```

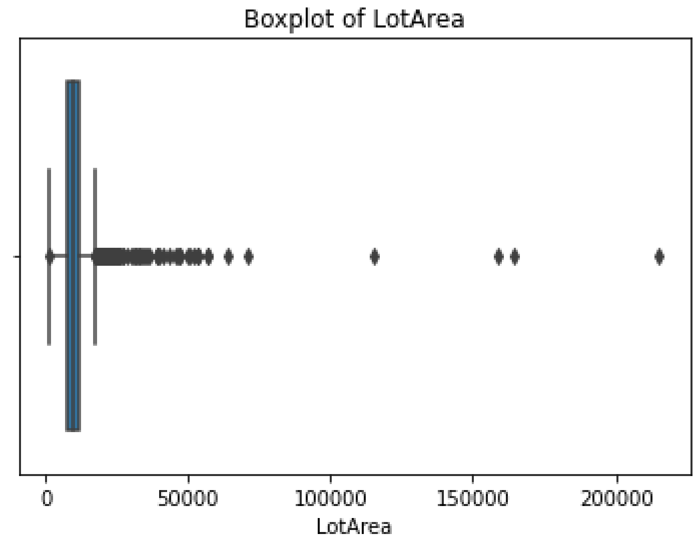
```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

numerical_cols = combined_df.select_dtypes(include=[np.number]).columns.tolis
for col in numerical_cols:
    sns.boxplot(x=combined_df[col])
    plt.title(f'Boxplot of {col}')
    plt.show()


correlation_matrix = combined_df.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, vmax=0.8, square=True)
plt.title('Correlation matrix of numerical variables')

# columns that correlate with 'SalePrice' by more than 0.5
high_corr_cols = correlation_matrix.index[abs(correlation_matrix["SalePrice"]
print("Columns that correlate with 'SalePrice' by more than 0.5:")
print(high_corr_cols)
```

```
Columns that correlate with 'SalePrice' by more than 0.5:
Index(['TotalBsmtSF', '1stFlrSF', 'GrLivArea', 'GarageArea', 'SalePrice'],
```

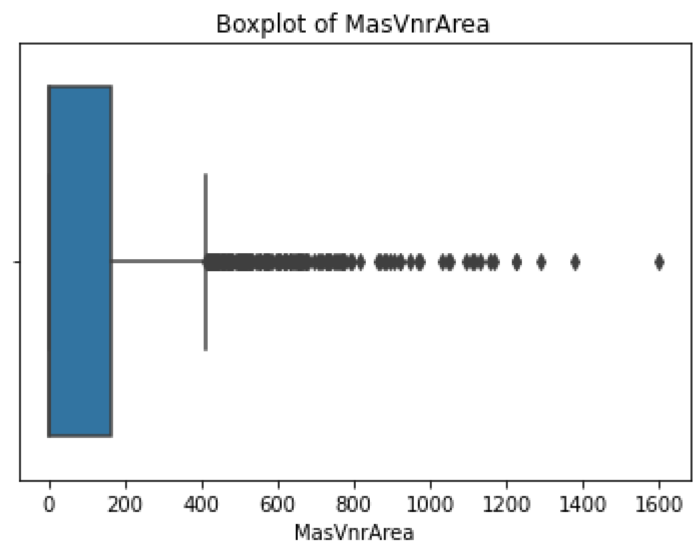
Download



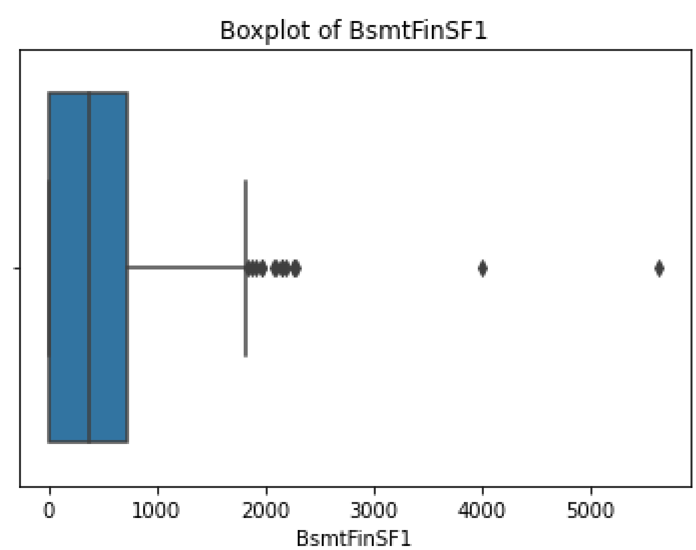
Download



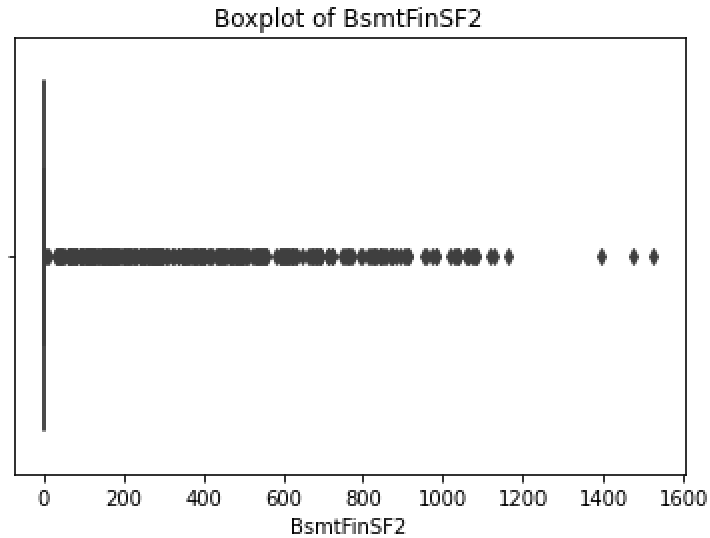
Download



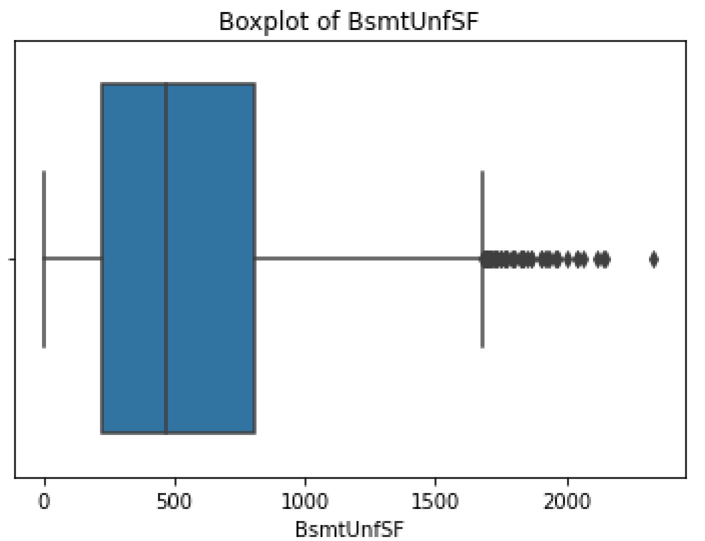
Download



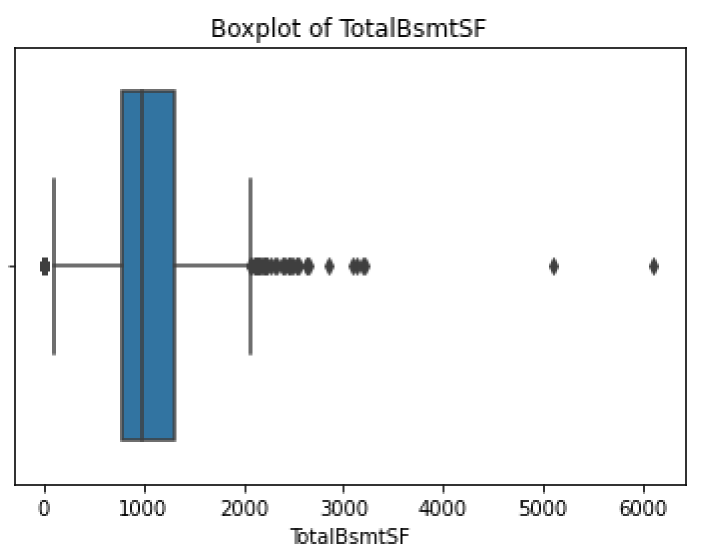
Download



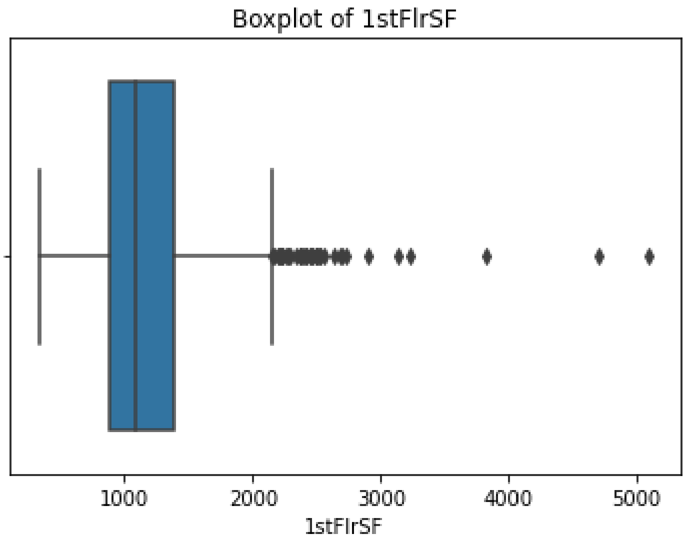
Download



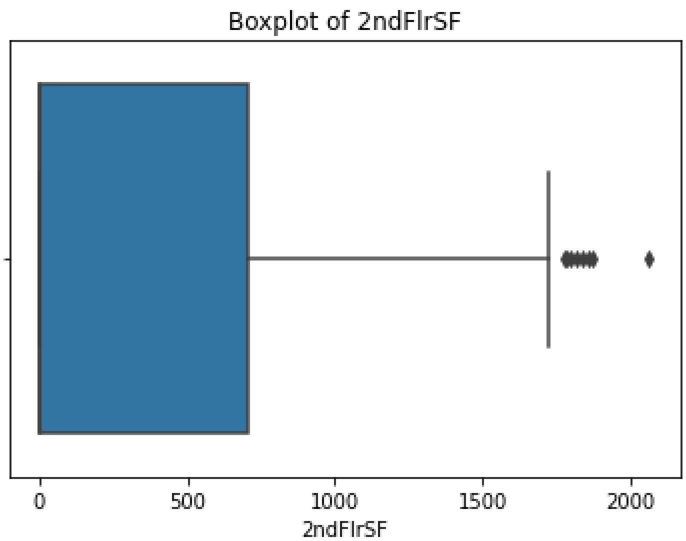
Download



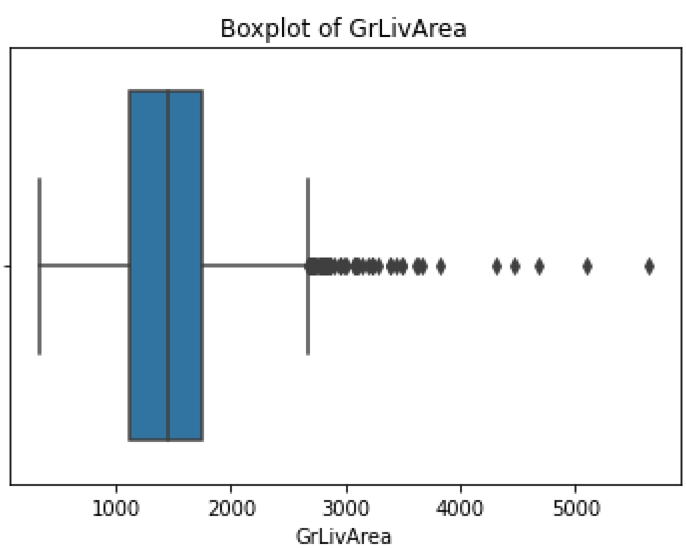
Download



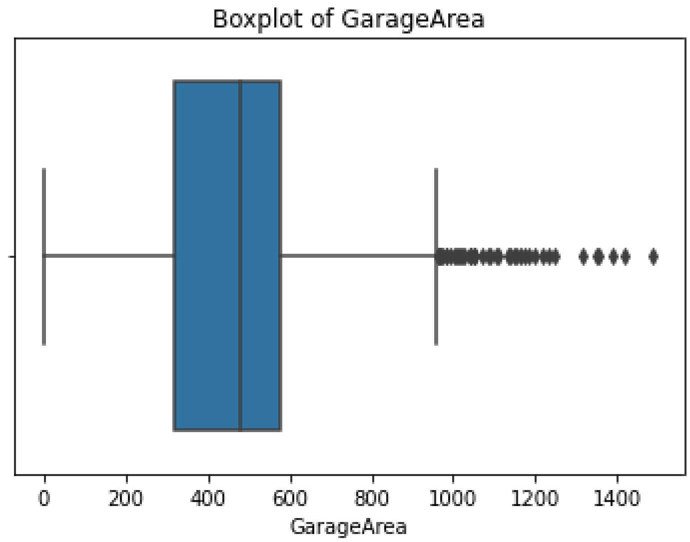
Download



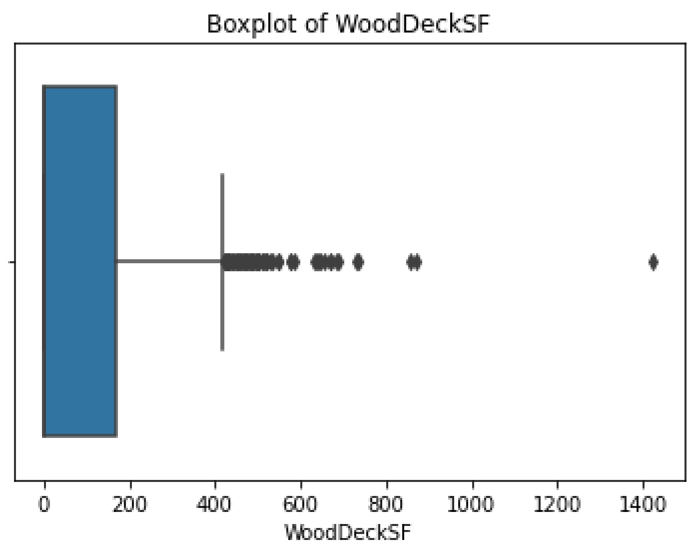
Download



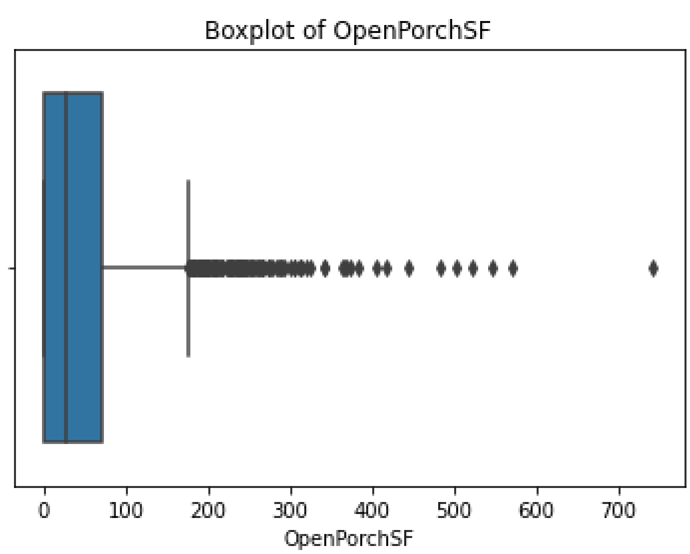
Download



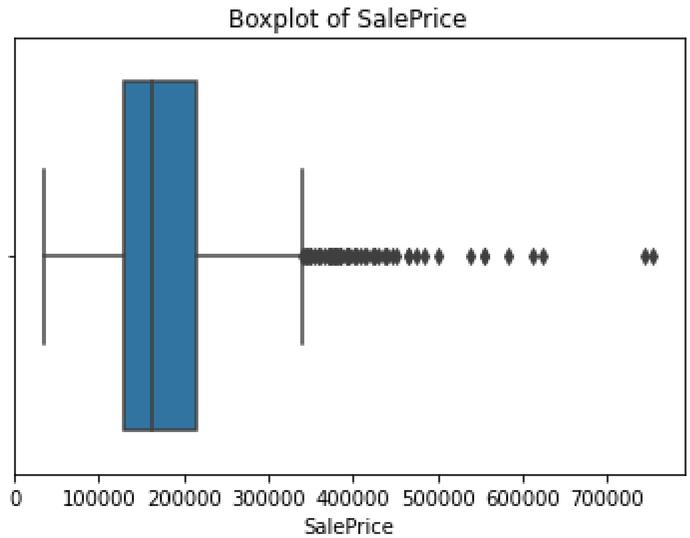
Download



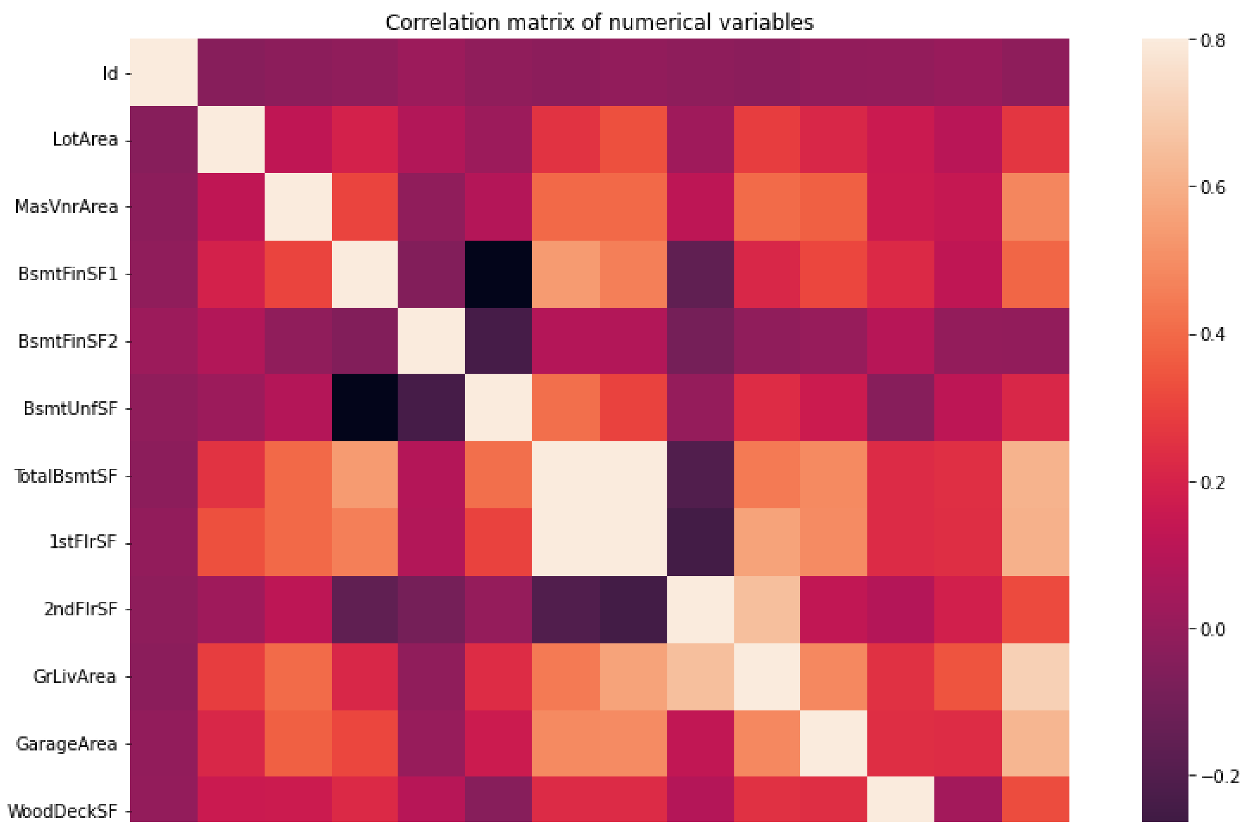
Download



Download



Download



Picked interest

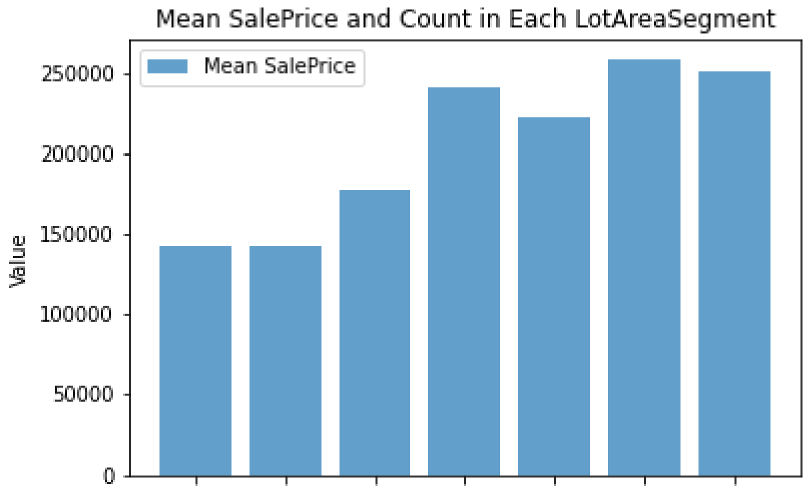
Lot Area vs salesprice --- inconsistent trend between lot size and price

```
import pandas as pd

# Columns I'm analysing
selected_columns = ['LotArea', 'SalePrice']
df_subset = combined_df[selected_columns]

bins = [0, 4000, 8000, 12000, 16000, 20000,25000, float('inf')]

df_subset['LotAreaSegment1'] = pd.cut(df_subset['LotArea'], bins=bins, labels

# Mean and count in each category
segmented_data = df_subset.groupby('LotAreaSegment1')['SalePrice'].agg(['mean
print(segmented_data)
```

```
  LotAreaSegment1           mean  count
0          0-4000  142672.273684     95
1       4001-8000  142786.502976    336
2      8001-12000  177548.680731    711
3     12001-16000  240961.027778    216
4     16001-20000  221878.122449     49
5     20001-25000  257609.954545     22
6          25001+  251312.064516     31

<ipython-input-11-c69cd2eb0d92>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  df_subset['LotAreaSegment1'] = pd.cut(df_subset['LotArea'], bins=bins, la
```

```
import pandas as pd
import matplotlib.pyplot as plt

plt.bar(segmented_data['LotAreaSegment1'], segmented_data['mean'], alpha=0.7,

plt.xlabel('LotAreaSegment1')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Value')
plt.title('Mean SalePrice and Count in Each LotAreaSegment')

plt.legend()

plt.show()
```

Download



Picked interest

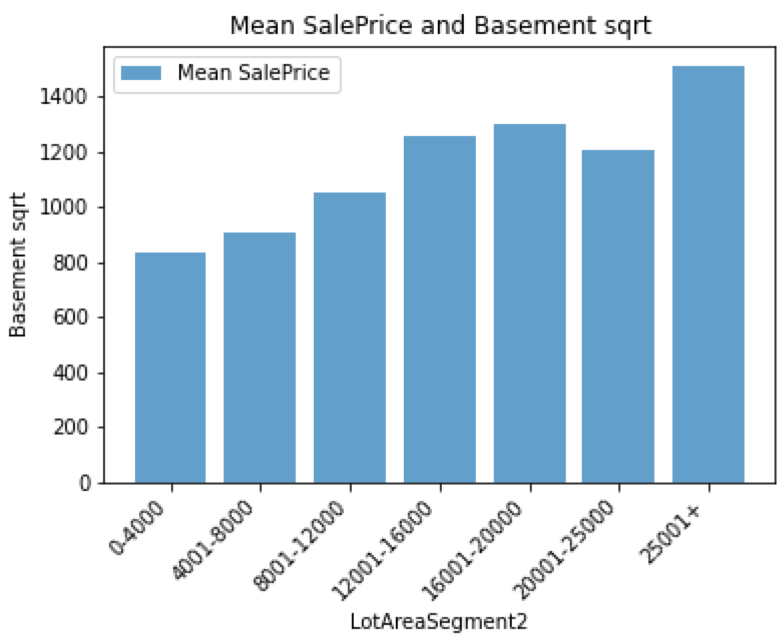
Lot Area vs Total Basement square footage --- larger lot area indicates larger basement

```
import pandas as pd

# Columns I'm analysing
selected_columns = ['LotArea', 'TotalBsmtSF']
df_subset = combined_df[selected_columns]

bins = [0, 4000, 8000, 12000, 16000, 20000,25000, float('inf')]  # Adjust the

df_subset['LotAreaSegment2'] = pd.cut(df_subset['LotArea'], bins=bins, labels

# Mean and count in each category
segmented_data = df_subset.groupby('LotAreaSegment2')['TotalBsmtSF'].agg(['me
print(segmented_data)
```

```
  LotAreaSegment2         mean  count
0          0-4000   831.446009    213
1       4001-8000   906.225997    677
2      8001-12000  1053.448006   1404
3     12001-16000  1258.109302    430
4     16001-20000  1302.913462    104
5     20001-25000  1203.261905     42
6          25001+  1508.458333     48

<ipython-input-13-f54a6e6b766b>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  df_subset['LotAreaSegment2'] = pd.cut(df_subset['LotArea'], bins=bins, la
```

```
import pandas as pd
import matplotlib.pyplot as plt

plt.bar(segmented_data['LotAreaSegment2'], segmented_data['mean'], alpha=0.7,

plt.xlabel('LotAreaSegment2')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Basement sqrt')
plt.title('Mean SalePrice and Basement sqrt')

plt.legend()

plt.show()
```

Download



Picked interest

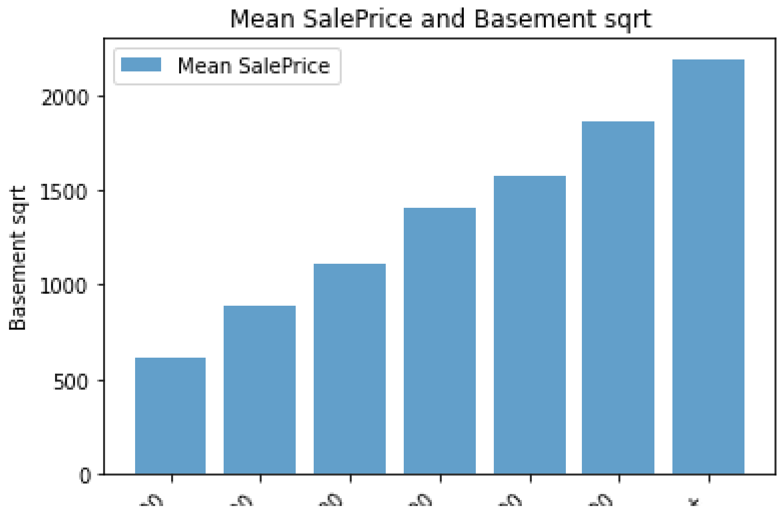
SalePrice vs basement sqft --- larger the basement sqft, more expensive houses get

```python
import pandas as pd

# Columns I'm analysing
selected_columns = ['SalePrice', 'TotalBsmtSF']
df_subset = combined_df[selected_columns]

bins = [0, 100000, 150000, 250000, 300000, 400000, 500000, float('inf')]  # A

df_subset['SalePriceSegment1'] = pd.cut(df_subset['SalePrice'], bins=bins, la

# Mean and count in each category
segmented_data = df_subset.groupby('SalePriceSegment1')['TotalBsmtSF'].agg(['
print(segmented_data)
```

```
  SalePriceSegment1         mean  count
0          0-100000   609.788618    123
1     100001-150000   892.364919    496
2     250001-250000  1106.350962    624
3     250001-300000  1407.029412    102
4     300001-400000  1576.620690     87
5     400001-500000  1863.052632     19
6           500001+  2198.444444      9

<ipython-input-15-7a3ff514b798>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  df_subset['SalePriceSegment1'] = pd.cut(df_subset['SalePrice'], bins=bins
```

```python
import pandas as pd
import matplotlib.pyplot as plt

plt.bar(segmented_data['SalePriceSegment1'], segmented_data['mean'], alpha=0.

plt.xlabel('SalePriceSegment1')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Basement sqrt')
plt.title('Mean SalePrice and Basement sqrt')

plt.legend()

plt.show()
```

Download



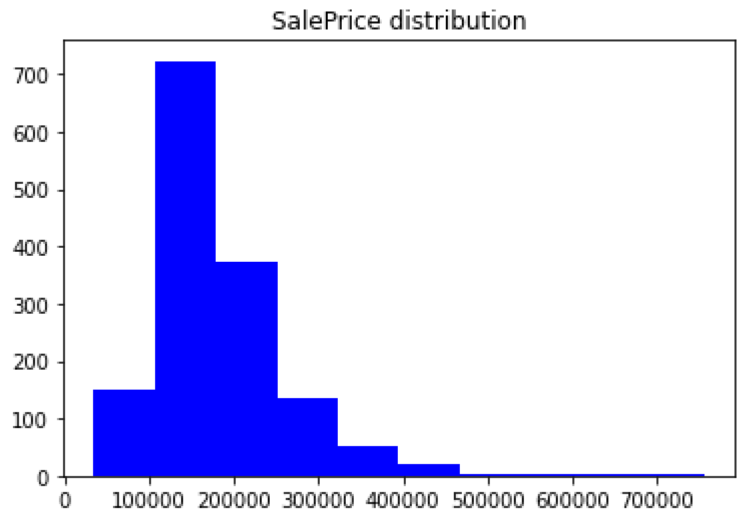
Skewness of numerical features

```
from scipy.stats import skew

skewed_feats = combined_df.apply(lambda x: skew(x.dropna())).sort_values(asce

print("\nSkewness in numerical features: \n")
skewness = pd.DataFrame({'Skewness' :skewed_feats})
print(skewness.head(10))

# Histogram of 'SalePrice'
plt.hist(combined_df['SalePrice'].dropna(), color='blue')
plt.title('SalePrice distribution')
plt.show()


saleprice_skew = skew(combined_df['SalePrice'].dropna())
print("Skewness of 'SalePrice':", saleprice_skew)

# Correlation Matrix Heatmap visualization (just for top 10 variables most co
k = 10
cols = combined_df.corr().nlargest(k, 'SalePrice')['SalePrice'].index

cm = np.corrcoef(combined_df[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws
plt.show()
```

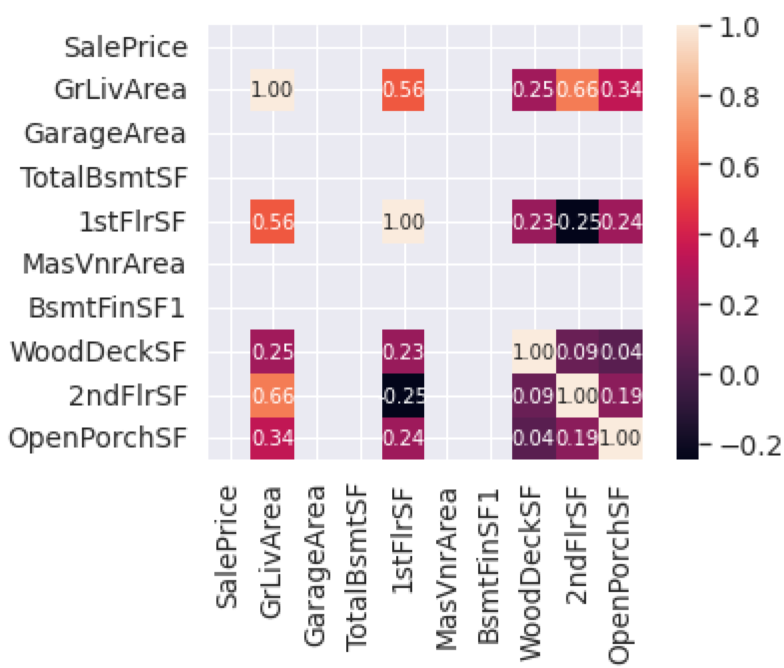
```
Skewness in numerical features:

              Skewness
```

```
LotArea        12.822431
BsmtFinSF2      4.145323
MasVnrArea      2.601240
OpenPorchSF     2.535114
SalePrice       1.880941
WoodDeckSF      1.842433
1stFlrSF        1.469604
BsmtFinSF1      1.424989
GrLivArea       1.269358
TotalBsmtSF     1.162285
Skewness of 'SalePrice': 1.880940746034036
```

Download



Download



Prepartation for modeling

```
import pandas as pd

# Check if any value in the 'SalePrice' column is negative
has_negative_values = (combined_df['SalePrice'] < 0).any()

if has_negative_values:
    print("The 'SalePrice' column contains negative values.")
else:
    print("There are no negative values in the 'SalePrice' column.")
```

```
There are no negative values in the 'SalePrice' column.
```

replacing all NAN values with the mean value in thier respected columns

```
import pandas as pd

# Checking for NaN values
nan_counts = combined_df.isna().sum()

columns_with_nan = nan_counts[nan_counts > 0].index
print(f"Columns with NaN values: {list(columns_with_nan)}")

for column in columns_with_nan:
    nan_indices = combined_df[column].index[combined_df[column].isna()].tolis
    print(f"Column '{column}' has NaN values at indices: {nan_indices}")
    print(f"NaN values in '{column}': {combined_df[column][nan_indices].tolis
```

```
Columns with NaN values: ['MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtU
Column 'MasVnrArea' has NaN values at indices: [234, 529, 650, 936, 973, 9
NaN values in 'MasVnrArea': [nan, nan, nan, nan, nan, nan, nan, nan, nan, 

Column 'BsmtFinSF1' has NaN values at indices: [2120]
NaN values in 'BsmtFinSF1': [nan]

Column 'BsmtFinSF2' has NaN values at indices: [2120]
NaN values in 'BsmtFinSF2': [nan]

Column 'BsmtUnfSF' has NaN values at indices: [2120]
NaN values in 'BsmtUnfSF': [nan]

Column 'TotalBsmtSF' has NaN values at indices: [2120]
```

```
NaN values in 'TotalBsmtSF': [nan]

Column 'GarageArea' has NaN values at indices: [2576]
NaN values in 'GarageArea': [nan]
```

I decided to use cluter based imputation for the NaN values, specifically Kmeans clustering due to the relation between the different attributes in the dataset

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Select columns to be considered for similarity
columns_considered = ['LotArea', 'TotalBsmtSF', 'SalePrice']

df_imputation = combined_df[columns_considered].copy()

df_imputation.fillna(df_imputation.mean(), inplace=True)
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_imputation), columns=columns

# KMeans clustering
kmeans = KMeans(n_clusters=5, random_state=0)  # Adjust number of clusters as
combined_df['cluster'] = kmeans.fit_predict(df_scaled)

# cluster means
cluster_means = df_scaled.groupby(combined_df['cluster']).mean()

for column in columns_considered:
    combined_df[column].fillna(
        value=combined_df['cluster'].map(cluster_means[column]),
        inplace=True
    )


combined_df.drop(columns='cluster', inplace=True)

# Checkpoint for any left over missing values
print("Missing values after imputation:")
print(combined_df.isnull().sum())
```

```
Missing values after imputation:
Id              0
LotArea         0
MasVnrArea     23
BsmtFinSF1      1
BsmtFinSF2      1
BsmtUnfSF       1
TotalBsmtSF     0
```

```
1stFlrSF        0
2ndFlrSF        0
GrLivArea       0
GarageArea      1
WoodDeckSF      0
OpenPorchSF     0
SalePrice       0
dtype: int64

/opt/python/envs/default/lib/python3.8/site-packages/sklearn/cluster/_kmean
  warnings.warn(
```

due to the low amount of missing values left, I decided to simply fill them in with the mean value

```
import pandas as pd

columns_with_missing = ['MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF'

combined_df[columns_with_missing] = combined_df[columns_with_missing].fillna(

# Checkpoint for any more missing values
missing_values_after_imputation = combined_df.isnull().sum()
print("Missing values after mean imputation:")
print(missing_values_after_imputation)

if missing_values_after_imputation.max() == 0:
    print("No more missing values after imputation.")
else:
    print("There are still missing values after imputation. Check the data.")
```

```
Missing values after mean imputation:
Id             0
LotArea        0
MasVnrArea     0
BsmtFinSF1     0
BsmtFinSF2     0
BsmtUnfSF      0
TotalBsmtSF    0
1stFlrSF       0
2ndFlrSF       0
GrLivArea      0
GarageArea     0
WoodDeckSF     0
OpenPorchSF    0
SalePrice      0
dtype: int64
No more missing values after imputation.
```

identifying 0 values within the dataset

```
import pandas as pd

# Checking for 0 values
zero_counts = (combined_df == 0).sum()

columns_with_zeros = zero_counts[zero_counts > 0].index
print(f"Columns with 0 values: {list(columns_with_zeros)}")

for column in columns_with_zeros:
    zero_indices = combined_df[column].index[combined_df[column] == 0].tolist
    zero_count = len(zero_indices)

    print(f"Column '{column}' has {zero_count} zero values.")
    print(f"Indices with 0 values in '{column}': {zero_indices}")
    print(f"0 values in '{column}': {combined_df[column][zero_indices].tolist
```

```
Columns with 0 values: ['MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnf
Column 'MasVnrArea' has 1738 zero values.
Indices with 0 values in 'MasVnrArea': [1, 3, 5, 8, 9, 10, 12, 15, 17, 18,
0 values in 'MasVnrArea': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.

Column 'BsmtFinSF1' has 929 zero values.
Indices with 0 values in 'BsmtFinSF1': [8, 13, 15, 17, 20, 21, 22, 25, 29,
0 values in 'BsmtFinSF1': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.

Column 'BsmtFinSF2' has 2571 zero values.
Indices with 0 values in 'BsmtFinSF2': [0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11,
0 values in 'BsmtFinSF2': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.

Column 'BsmtUnfSF' has 241 zero values.
Indices with 0 values in 'BsmtUnfSF': [17, 39, 42, 52, 54, 75, 90, 102, 12
0 values in 'BsmtUnfSF': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0

Column 'TotalBsmtSF' has 78 zero values.
Indices with 0 values in 'TotalBsmtSF': [17, 39, 90, 102, 156, 182, 259, 3
0 values in 'TotalBsmtSF': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
```

Modeling

training and testing data

```
from sklearn.model_selection import train_test_split

# separate the independent variables (features) from the dependent variable (
features = combined_df.drop('SalePrice', axis=1)
target = combined_df['SalePrice']

# dats split: 80% for training and 20% for testing
features_train, features_test, target_train, target_test = train_test_split(f

print('Training Features Shape:', features_train.shape)
print('Training Target Shape:', target_train.shape)
print('Testing Features Shape:', features_test.shape)
print('Testing Target Shape:', target_test.shape)
```

```
Training Features Shape: (2335, 13)
Training Target Shape: (2335,)
Testing Features Shape: (584, 13)
Testing Target Shape: (584,)
```

Linear regression model, Actual vs Predicted values, Risidual plot(should randomly be arounf the 0 value)

Risidual plot visual

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np


reg = LinearRegression()
reg.fit(features_train, target_train)

predictions = reg.predict(features_test)

# RMSE
MSE = mean_squared_error(target_test, predictions)
print('Mean Squared Error of the model is: {:.4f}'.format(MSE))
print('Root Mean Squared Error of the model is: {:.4f}'.format(np.sqrt(MSE)))

import seaborn as sns

# residuals: difference between actual and predicted values
residuals = target_test - predictions

plt.figure(figsize=(10,6))

# Plotting residuals
sns.scatterplot(x=target_test, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals Plot')
plt.xlabel('Predicted SalePrice')
plt.ylabel('Residuals (Actual - Predicted SalePrice)')
plt.show()

neg_predictions = predictions[predictions < 0]
if len(neg_predictions) > 0:
    print("Negative Predictions: ", neg_predictions)
else:
    print("No negative predictions.")
```

```
Mean Squared Error of the model is: 4012198518.0605
Root Mean Squared Error of the model is: 63341.9175
Negative Predictions:  [-32067.76260083 -27550.52459799 -23307.66218927 -3
 -27686.98342716 -46399.72095856  -9774.64612818  -9138.47821565
  -1936.9959027  -18276.60352608 -19985.97884672 -32011.28752321
 -16604.24046505 -21239.28295876 -18207.09276892 -39067.69269697
 -26989.12256384 -48557.15061732 -35551.08274908 -62213.29886494
 -26248.50894042 -64968.83783606 -38372.87322427 -15680.68317988
 -48182.52573228 -35443.40891897 -54996.70310147 -47084.97782785
 -50972.03258999 -24734.7667437  -36665.24127799 -21971.36489185
 -42128.00286041 -17105.93664335  -1590.74425506 -39230.56421335
 -22169.50891182 -58685.88914314 -19948.81228748 -15583.44552094
 -30319.50845508  -6839.70584646 -76655.3638719  -25336.34076993
 -29322.60945185 -35633.0497428  -12295.54325602 -10032.13584292
   -146.17628    -27443.73264145  -7122.72905579   -681.64375722
```

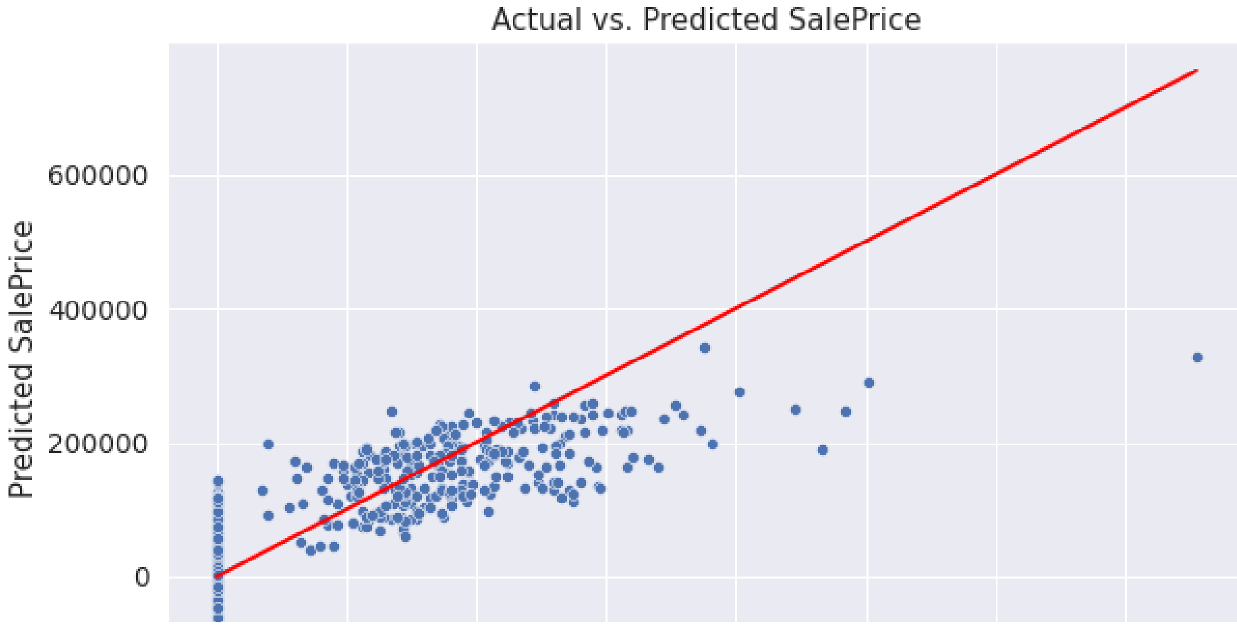
```
 -52775.61625749 -80145.06704539 -14933.02159756 -23551.38836329
 -33831.32352826 -18810.6666858    -654.9525945  -18372.92378488
 -99695.67121294  -6342.85419588 -19557.32568838 -20310.66636136
 -68095.07011148 -19326.19841468 -23149.0274253  -29135.5477749
```

Download

Actual vs predicted saleprice plot visual

```
# scatter plot of actual vs. predicted values
plt.figure(figsize=(10,6))
sns.scatterplot(x=target_test, y=predictions)
plt.title('Actual vs. Predicted SalePrice')
plt.xlabel('Actual SalePrice')
plt.ylabel('Predicted SalePrice')

# line that represents perfect predictions
plt.plot([target_test.min(), target_test.max()], [target_test.min(), target_t
plt.show()
```

Download


Actual vs. Predicted SalePrice
Predicted SalePrice
600000
400000
200000
0