FLIP ROBO

NAME OF THE PROJECT

HOUSING: PRICE PREDICTION

Submitted by:

Tanuja Patil

ACKNOWLEDGMENT

I have completed the project with the help of DataTrained learning as well as Flip Robo’s staff .

INTRODUCTION

Business Problem Framing:

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below.

The company is looking at prospective properties to buy houses to enter the market. It is required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not. For this company wants to know:

- Which variables are important to predict the price of variable?
- How do these variables describe the price of the house?

Review of Literature

Accurately estimating the value of real estate is an important problem for many stakeholders including house owners, house buyers, agents, creditors, and investors. It is also a difficult one. Though it is common knowledge that factors such as the size, number of rooms and location affect the price, there are many other things at play. Additionally, prices are sensitive to changes in market demand and the peculiarities of each situation, such as when a property needs to be urgently sold. The sales price of a property can be predicted in various ways, but is often based on regression techniques. All regression techniques essentially involve one or more predictor variables as input and a single target variable as output

Machine learning is a form of artificial intelligence which compose available computers with the efficiency to be trained without being veraciously programmed. Machine learning interest on the extensions of computer programs which is capable enough to modify when unprotected to new-fangled data. Machine learning algorithms are broadly classified into three divisions, namely; Supervised learning, Unsupervised learning and Reinforcement learning. Supervised learning is a learning in which we teach or train the machine using data which is well labelled that means some data is already tagged with correct answer. After that, machine is provided with new set of examples so that supervised learning algorithm analyses the training data and produces a correct outcome from labelled data. Unsupervised learning is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information

according to similarities, patterns and differences without any prior training of data. Unlike, supervised learning, no teacher is provided that means no training will be given to the machine. Therefore, machine is restricted to find the hidden structure in unlabelled data by our-self.

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience. Machine learning has many application's out of which one of the applications is prediction of real estate. The real estate market is one of the most competitive in terms of pricing and same tends to be vary significantly based on lots of factor, forecasting property price is an important modules in decision making for both the buyers and investors in supporting budget allocation, finding property finding stratagems and determining suitable policies hence it becomes one of the prime fields to apply the concepts of machine learning to optimize and predict the prices with high accuracy. The study on land price trend is felt important to support the decisions in urban planning. The real estate system is an unstable stochastic process. Investors decisions are based on the market trends to reap maximum returns. Developers are interested to know the future trends for their decision making. To accurately estimate property prices and future trends, large amount of data that influences land price is required for analysis, modelling and forecasting. The factors that affect the land price have to be studied and their impact on price has also to be modelled. An analysis of the past data is to be considered. It is inferred that establishing a simple linear mathematical relationship for these timeseries data is found not viable for forecasting. Hence it became imperative to establish a non-linear model which can well fit the data characteristic to analyse and forecast future trends. As the real estate is fast developing sector, the analysis and forecast of land prices using mathematical modelling and other scientific techniques is an immediate urgent need for decision making by all those concerned. The increase in population as well as the industrial activity is attributed to various factors, the most prominent being the recent spurt in the knowledge sector viz. Information Technology (IT) and Information technology enabled services. Demand for land started of showing an upward trend and housing and the real estate activity started booming. All barren lands and paddy fields ceased their existence to pave way for multistore and highrise buildings. Investments started pouring in Real estate Industry and there was no uniform pattern in the land price over the years. The need for predicting the trend in land prices was felt by all in the industry viz. the Government, the regulating bodies, lending institutions, the developers and the investors. We can use regression models, using various features to have lower Residual Sum of Squares error. While using features in a regression model some feature engineering is required for better prediction. Often a set of features multiple regressions or polynomial regression (applying a various set of powers in the features) is used for making better model fit. For these models are expected to be susceptible towards over fitting ridge regression is used to reduce it. So, it directs to the best application of regression models in addition to other techniques to optimize the result.

Analytical Problem Framing

In this project ,we used the different mathematical and statical functions to describe the data more efficiently.

1. Isnull():This function is used to identify whether the data set have any null values or not.

2. Corr():This function is used to identify the correlation between different input variables.

3. describe():This function give all stastical summary of data set.

For exp:count,mean,median,max,min values

4. Shape():This function tells us how many rows and columns present in the dataset.

Hardware and Software Requirements and Tools Used

` We used jupyter notebook for this project. Following libraries are used:

- Pandas:used for mathematical and statical analysis of data. For example:
    - ➢ pandas.read_csv():used to read csv file
    - ➢ pandas.Dataframe():passed the data to dataframe so we can perform different operations on data
- Seaborn:used for visualization
  - ➢ Heatmap:used to visualize colinearity between variables
  - ➢ Distplot:used to visualize distribution of dataset
  - ➢ Countplot: used to visualize categorical data

Data Analysis:

Data analysis involves manipulating, transforming, and visualizing data in order to infer meaningful insights from the results. Individuals, businesses, and even governments often take direction based on these insights. In Data analysis, we have check data types, missing values, and many more things. so let’s do it one by one First importing all necessary libraries. One by one.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Data Sources:

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file.

Following is the screenshot data set which are in two parts:

Training and testing:

```
df_train=pd.read_csv("train.csv")
df_test=pd.read_csv("test.csv")
```

```
df_train
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	Misc
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	NaN	
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
5	1197	60	RL	58.0	14054	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
6	561	20	RL	NaN	11341	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
7	1041	20	RL	88.0	13125	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	NaN	
8	503	20	RL	70.0	9170	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	Shed	
9	576	50	RL	80.0	8480	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
10	449	50	RM	50.0	8600	Pave	NaN	Reg	Bnk	AllPub	...	0	NaN	NaN	NaN	

```
df_test
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC	Fence	Misc
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	AllPub	...	0	0	NaN	NaN	
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	NaN	
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	AllPub	...	0	0	NaN	NaN	
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	
5	650	180	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	MnPrv	
6	1453	180	RM	35.0	3675	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	NaN	
7	152	20	RL	107.0	13891	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	NaN	
8	427	80	RL	NaN	12800	Pave	NaN	Reg	Low	AllPub	...	396	0	NaN	NaN	
9	776	120	RM	32.0	4500	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	NaN	
10	30	30	RM	60.0	6324	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	

Data is given by client for the predictions. The snapshot of the data set input variables is given in fig.

MSSubClass: Identifies the type of dwelling involved in the sale.

20	1-STORY 1946 & NEWER ALL STYLES
30	1-STORY 1945 & OLDER
40	1-STORY W/FINISHED ATTIC ALL AGES
45	1-1/2 STORY - UNFINISHED ALL AGES
50	1-1/2 STORY FINISHED ALL AGES
60	2-STORY 1946 & NEWER
70	2-STORY 1945 & OLDER
75	2-1/2 STORY ALL AGES
80	SPLIT OR MULTI-LEVEL
85	SPLIT FOYER
90	DUPLEX - ALL STYLES AND AGES
120	1-STORY PUD (Planned Unit Development) - 1946 & NEWER
150	1-1/2 STORY PUD - ALL AGES
160	2-STORY PUD - 1946 & NEWER
180	PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
190	2 FAMILY CONVERSION - ALL STYLES AND AGES

MSZoning: Identifies the general zoning classification of the sale.

A	Agriculture
C	Commercial
FV	Floating Village Residential
I	Industrial
RH	Residential High Density
RL	Residential Low Density
RP	Residential Low Density Park
RM	Residential Medium Density

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

Street: Type of road access to property

```
       Grvl     Gravel
       Pave     Paved
```

Alley: Type of alley access to property

```
       Grvl     Gravel
       Pave     Paved
       NA       No alley access
```

LotShape: General shape of property

```
       Reg      Regular
       IR1      Slightly irregular
       IR2      Moderately Irregular
       IR3      Irregular
```

LandContour: Flatness of the property

```
       Lvl      Near Flat/Level
       Bnk      Banked - Quick and significant rise from street grade to build
       HLS      Hillside - Significant slope from side to side
       Low      Depression
```

Utilities: Type of utilities available

```
       AllPub   All public Utilities (E,G,W,& S)
       NoSewr   Electricity, Gas, and Water (Septic Tank)
       NoSeWa   Electricity and Gas Only
       ELO      Electricity only
```

LotConfig: Lot configuration

```
       Inside   Inside lot
       Corner   Corner lot
```

```
       Corner   Corner lot
       CulDSac  Cul-de-sac
       FR2      Frontage on 2 sides of property
       FR3      Frontage on 3 sides of property

LandSlope: Slope of property

       Gtl      Gentle slope
       Mod      Moderate Slope
       Sev      Severe Slope

Neighborhood: Physical locations within Ames city limits

       Blmngtn  Bloomington Heights
       Blueste  Bluestem
       BrDale   Briardale
       BrkSide  Brookside
       ClearCr  Clear Creek
       CollgCr  College Creek
       Crawfor  Crawford
       Edwards  Edwards
       Gilbert  Gilbert
       IDOTRR   Iowa DOT and Rail Road
       MeadowV  Meadow Village
       Mitchel  Mitchell
       Names    North Ames
       NoRidge  Northridge
       NPkVill  Northpark Villa
       NridgHt  Northridge Heights
       NWAmes   Northwest Ames
       OldTown  Old Town
       SWISU    South & West of Iowa State University
       Sawyer   Sawyer
```

```
       SawyerW	Sawyer West
       Somerst	Somerset
       StoneBr	Stone Brook
       Timber	Timberland
       Veenker	Veenker

Condition1: Proximity to various conditions

       Artery	Adjacent to arterial street
       Feedr	Adjacent to feeder street
       Norm	Normal
       RRNn	Within 200' of North-South Railroad
       RRAn	Adjacent to North-South Railroad
       PosN	Near positive off-site feature--park, greenbelt, etc.
       PosA	Adjacent to postive off-site feature
       RRNe	Within 200' of East-West Railroad
       RRAe	Adjacent to East-West Railroad

Condition2: Proximity to various conditions (if more than one is present)

       Artery	Adjacent to arterial street
       Feedr	Adjacent to feeder street
       Norm	Normal
       RRNn	Within 200' of North-South Railroad
       RRAn	Adjacent to North-South Railroad
       PosN	Near positive off-site feature--park, greenbelt, etc.
       PosA	Adjacent to postive off-site feature
       RRNe	Within 200' of East-West Railroad
       RRAe	Adjacent to East-West Railroad

BldgType: Type of dwelling

       1Fam	Single-family Detached
       2FmCon	Two-family Conversion; originally built as one-family dwelling
       Duplx	Duplex
```

```
       TwnhsE   Townhouse End Unit
       TwnhsI   Townhouse Inside Unit

HouseStyle: Style of dwelling

       1Story   One story
       1.5Fin   One and one-half story: 2nd level finished
       1.5Unf   One and one-half story: 2nd level unfinished
       2Story   Two story
       2.5Fin   Two and one-half story: 2nd level finished
       2.5Unf   Two and one-half story: 2nd level unfinished
       SFoyer   Split Foyer
       SLvl     Split Level

OverallQual: Rates the overall material and finish of the house

       10       Very Excellent
       9        Excellent
       8        Very Good
       7        Good
       6        Above Average
       5        Average
       4        Below Average
       3        Fair
       2        Poor
       1        Very Poor

OverallCond: Rates the overall condition of the house

       10       Very Excellent
       9        Excellent
       8        Very Good
       7        Good
       6        Above Average
       5        Average
```

```
       5         Average
       4         Below Average
       3         Fair
       2         Poor
       1         Very Poor

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

       Flat      Flat
       Gable     Gable
       Gambrel   Gabrel (Barn)
       Hip       Hip
       Mansard   Mansard
       Shed      Shed

RoofMatl: Roof material

       ClyTile   Clay or Tile
       CompShg   Standard (Composite) Shingle
       Membran   Membrane
       Metal     Metal
       Roll      Roll
       Tar&Grv   Gravel & Tar
       WdShake   Wood Shakes
       WdShngl   Wood Shingles

Exterior1st: Exterior covering on house

       AsbShng   Asbestos Shingles
       AsphShn   Asphalt Shingles
       BrkComm   Brick Common
```

```
        BrkFace   Brick Face
        CBlock    Cinder Block
        CemntBd   Cement Board
        HdBoard   Hard Board
        ImStucc   Imitation Stucco
        MetalSd   Metal Siding
        Other     Other
        Plywood   Plywood
        PreCast   PreCast
        Stone     Stone
        Stucco    Stucco
        VinylSd   Vinyl Siding
        Wd Sdng   Wood Siding
        WdShing   Wood Shingles

Exterior2nd: Exterior covering on house (if more than one material)

        AsbShng   Asbestos Shingles
        AsphShn   Asphalt Shingles
        BrkComm   Brick Common
        BrkFace   Brick Face
        CBlock    Cinder Block
        CemntBd   Cement Board
        HdBoard   Hard Board
        ImStucc   Imitation Stucco
        MetalSd   Metal Siding
        Other     Other
        Plywood   Plywood
        PreCast   PreCast
        Stone     Stone
        Stucco    Stucco
        VinylSd   Vinyl Siding
        Wd Sdng   Wood Siding
        WdShing   Wood Shingles
```

```
MasVnrType: Masonry veneer type

       BrkCmn   Brick Common
       BrkFace  Brick Face
       CBlock   Cinder Block
       None     None
       Stone    Stone

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

       Ex       Excellent
       Gd       Good
       TA       Average/Typical
       Fa       Fair
       Po       Poor

ExterCond: Evaluates the present condition of the material on the exterior

       Ex       Excellent
       Gd       Good
       TA       Average/Typical
       Fa       Fair
       Po       Poor

Foundation: Type of foundation

       BrkTil   Brick & Tile
       CBlock   Cinder Block
       PConc    Poured Contrete
       Slab     Slab
       Stone    Stone
       Wood     Wood
```

```
BsmtQual: Evaluates the height of the basement

       Ex       Excellent (100+ inches)
       Gd       Good (90-99 inches)
       TA       Typical (80-89 inches)
       Fa       Fair (70-79 inches)
       Po       Poor (<70 inches
       NA       No Basement

BsmtCond: Evaluates the general condition of the basement

       Ex       Excellent
       Gd       Good
       TA       Typical - slight dampness allowed
       Fa       Fair - dampness or some cracking or settling
       Po       Poor - Severe cracking, settling, or wetness
       NA       No Basement

BsmtExposure: Refers to walkout or garden level walls

       Gd       Good Exposure
       Av       Average Exposure (split levels or foyers typically score average or above)
       Mn       Mimimum Exposure
       No       No Exposure
       NA       No Basement

BsmtFinType1: Rating of basement finished area

       GLQ      Good Living Quarters
       ALQ      Average Living Quarters
       BLQ      Below Average Living Quarters
       Rec      Average Rec Room
       LwQ      Low Quality
       Unf      Unfinshed
```

```
       NA       No Basement

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types

       GLQ      Good Living Quarters
       ALQ      Average Living Quarters
       BLQ      Below Average Living Quarters
       Rec      Average Rec Room
       LwQ      Low Quality
       Unf      Unfinshed
       NA       No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

       Floor    Floor Furnace
       GasA     Gas forced warm air furnace
       GasW     Gas hot water or steam heat
       Grav     Gravity furnace
       OthW     Hot water or steam heat other than gas
       Wall     Wall furnace

HeatingQC: Heating quality and condition

       Ex       Excellent
       Gd       Good
       TA       Average/Typical
       Fa       Fair
```

```
       Po       Poor

CentralAir: Central air conditioning

       N        No
       Y        Yes

Electrical: Electrical system

       SBrkr    Standard Circuit Breakers & Romex
       FuseA    Fuse Box over 60 AMP and all Romex wiring (Average)
       FuseF    60 AMP Fuse Box and mostly Romex wiring (Fair)
       FuseP    60 AMP Fuse Box and mostly knob & tube wiring (poor)
       Mix      Mixed

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade
```

KitchenQual: Kitchen quality

Ex	Excellent
Gd	Good
TA	Typical/Average
Fa	Fair
Po	Poor

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

Typ	Typical Functionality
Min1	Minor Deductions 1
Min2	Minor Deductions 2
Mod	Moderate Deductions
Maj1	Major Deductions 1
Maj2	Major Deductions 2
Sev	Severely Damaged
Sal	Salvage only

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

Ex	Excellent - Exceptional Masonry Fireplace
Gd	Good - Masonry Fireplace in main level
TA	Average - Prefabricated Fireplace in main living area or Masonry Fireplace in basement
Fa	Fair - Prefabricated Fireplace in basement
Po	Poor - Ben Franklin Stove
NA	No Fireplace

```
GarageType: Garage location

       2Types   More than one type of garage
       Attchd   Attached to home
       Basment  Basement Garage
       BuiltIn  Built-In (Garage part of house - typically has room above garage)
       CarPort  Car Port
       Detchd   Detached from home
       NA       No Garage

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

       Fin      Finished
       RFn      Rough Finished
       Unf      Unfinished
       NA       No Garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

       Ex       Excellent
       Gd       Good
       TA       Typical/Average
       Fa       Fair
       Po       Poor
       NA       No Garage
```

```
GarageCond: Garage condition

       Ex	Excellent
       Gd	Good
       TA	Typical/Average
       Fa	Fair
       Po	Poor
       NA	No Garage

PavedDrive: Paved driveway

       Y	Paved
       P	Partial Pavement
       N	Dirt/Gravel

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

       Ex	Excellent
       Gd	Good
       TA	Average/Typical
       Fa	Fair
       NA	No Pool

Fence: Fence quality
```

Fence: Fence quality

GdPrv	Good Privacy
MnPrv	Minimum Privacy
GdWo	Good Wood
MnWw	Minimum Wood/Wire
NA	No Fence

MiscFeature: Miscellaneous feature not covered in other categories

Elev	Elevator
Gar2	2nd Garage (if not described in garage section)
Othr	Other
Shed	Shed (over 100 SF)
TenC	Tennis Court
NA	None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

WD	Warranty Deed - Conventional
CWD	Warranty Deed - Cash
VWD	Warranty Deed - VA Loan
New	Home just constructed and sold
COD	Court Officer Deed/Estate
Con	Contract 15% Down payment regular terms
ConLw	Contract Low Down payment and low interest
ConLI	Contract Low Interest
ConLD	Contract Low Down
Oth	Other

SaleCondition: Condition of sale

Normal	Normal Sale
Abnorml	Abnormal Sale - trade, foreclosure, short sale
AdjLand	Adjoining Land Purchase
Alloca	Allocation - two linked properties with separate deeds, typically condo with a garage unit
Family	Sale between family members
Partial	Home was not completed when last assessed (associated with New Homes)

Checking shape of training data set:

```
1 #let's check shape of training data set
2 df_train.shape
```

```
(1168, 81)
```

Training data set have 1168 rows and 81columns

Checking shape of testing data:

```
1 #let's check shape of testing data set
2 df_test.shape
```

```
(292, 80)
```

Testing data set have 292 rows and 80 columns

Checking basic info of data set:

```
1  df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1168 non-null   int64
 1   MSSubClass     1168 non-null   int64
 2   MSZoning       1168 non-null   object
 3   LotFrontage    954 non-null    float64
 4   LotArea        1168 non-null   int64
 5   Street         1168 non-null   object
 6   Alley          77 non-null     object
 7   LotShape       1168 non-null   object
 8   LandContour    1168 non-null   object
 9   Utilities      1168 non-null   object
 10  LotConfig      1168 non-null   object
 11  LandSlope      1168 non-null   object
 12  Neighborhood   1168 non-null   object
 13  Condition1     1168 non-null   object
 14  Condition2     1168 non-null   object
 15  BldgType       1168 non-null   object
 16  HouseStyle     1168 non-null   object
 17  OverallQual    1168 non-null   int64
 18  OverallCond    1168 non-null   int64
 19  YearBuilt      1168 non-null   int64
 20  YearRemodAdd   1168 non-null   int64
 21  RoofStyle      1168 non-null   object
 22  RoofMatl       1168 non-null   object
 23  Exterior1st    1168 non-null   object
 24  Exterior2nd    1168 non-null   object
 25  MasVnrType     1161 non-null   object
 26  MasVnrArea     1161 non-null   float64
```

```
27  ExterQual      1168 non-null   object
28  ExterCond      1168 non-null   object
29  Foundation     1168 non-null   object
30  BsmtQual       1138 non-null   object
31  BsmtCond       1138 non-null   object
32  BsmtExposure   1137 non-null   object
33  BsmtFinType1   1138 non-null   object
34  BsmtFinSF1     1168 non-null   int64
35  BsmtFinType2   1137 non-null   object
36  BsmtFinSF2     1168 non-null   int64
37  BsmtUnfSF      1168 non-null   int64
38  TotalBsmtSF    1168 non-null   int64
39  Heating        1168 non-null   object
40  HeatingQC      1168 non-null   object
41  CentralAir     1168 non-null   object
42  Electrical     1168 non-null   object
43  1stFlrSF       1168 non-null   int64
44  2ndFlrSF       1168 non-null   int64
45  LowQualFinSF   1168 non-null   int64
46  GrLivArea      1168 non-null   int64
47  BsmtFullBath   1168 non-null   int64
48  BsmtHalfBath   1168 non-null   int64
49  FullBath       1168 non-null   int64
50  HalfBath       1168 non-null   int64
51  BedroomAbvGr   1168 non-null   int64
52  KitchenAbvGr   1168 non-null   int64
53  KitchenQual    1168 non-null   object
54  TotRmsAbvGrd   1168 non-null   int64
55  Functional     1168 non-null   object
56  Fireplaces     1168 non-null   int64
57  FireplaceQu    617 non-null    object
58  GarageType     1104 non-null   object
59  GarageYrBlt    1104 non-null   float64

 60  GarageFinish   1104 non-null   object
 61  GarageCars     1168 non-null   int64
 62  GarageArea     1168 non-null   int64
 63  GarageQual     1104 non-null   object
 64  GarageCond     1104 non-null   object
 65  PavedDrive     1168 non-null   object
 66  WoodDeckSF     1168 non-null   int64
 67  OpenPorchSF    1168 non-null   int64
 68  EnclosedPorch  1168 non-null   int64
 69  3SsnPorch      1168 non-null   int64
 70  ScreenPorch    1168 non-null   int64
 71  PoolArea       1168 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          237 non-null    object
 74  MiscFeature    44 non-null     object
 75  MiscVal        1168 non-null   int64
 76  MoSold         1168 non-null   int64
 77  YrSold         1168 non-null   int64
 78  SaleType       1168 non-null   object
 79  SaleCondition  1168 non-null   object
 80  SalePrice      1168 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 739.2+ KB
```

The data set has 1168 examples and 80 features. 43 of the features are object, 35 of the features are int, 3 of the features are float.

Some values are missing in data set.

Checking basic info of testing data:

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 292 entries, 0 to 291
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             292 non-null    int64
 1   MSSubClass     292 non-null    int64
 2   MSZoning       292 non-null    object
 3   LotFrontage    247 non-null    float64
 4   LotArea        292 non-null    int64
 5   Street         292 non-null    object
 6   Alley          14 non-null     object
 7   LotShape       292 non-null    object
 8   LandContour    292 non-null    object
 9   Utilities      292 non-null    object
 10  LotConfig      292 non-null    object
 11  LandSlope      292 non-null    object
 12  Neighborhood   292 non-null    object
 13  Condition1     292 non-null    object
 14  Condition2     292 non-null    object
 15  BldgType       292 non-null    object
 16  HouseStyle     292 non-null    object
 17  OverallQual    292 non-null    int64
 18  OverallCond    292 non-null    int64
 19  YearBuilt      292 non-null    int64
 20  YearRemodAdd   292 non-null    int64
 21  RoofStyle      292 non-null    object
 22  RoofMatl       292 non-null    object
 23  Exterior1st    292 non-null    object
 24  Exterior2nd    292 non-null    object
 25  MasVnrType     291 non-null    object
 26  MasVnrArea     291 non-null    float64
 27  ExterQual      292 non-null    object
 28  ExterCond      292 non-null    object
 29  Foundation     292 non-null    object
```

```
 29  Foundation     292 non-null    object
 30  BsmtQual       285 non-null    object
 31  BsmtCond       285 non-null    object
 32  BsmtExposure   285 non-null    object
 33  BsmtFinType1   285 non-null    object
 34  BsmtFinSF1     292 non-null    int64
 35  BsmtFinType2   285 non-null    object
 36  BsmtFinSF2     292 non-null    int64
 37  BsmtUnfSF      292 non-null    int64
 38  TotalBsmtSF    292 non-null    int64
 39  Heating        292 non-null    object
 40  HeatingQC      292 non-null    object
 41  CentralAir     292 non-null    object
 42  Electrical     291 non-null    object
 43  1stFlrSF       292 non-null    int64
 44  2ndFlrSF       292 non-null    int64
 45  LowQualFinSF   292 non-null    int64
 46  GrLivArea      292 non-null    int64
 47  BsmtFullBath   292 non-null    int64
 48  BsmtHalfBath   292 non-null    int64
 49  FullBath       292 non-null    int64
 50  HalfBath       292 non-null    int64
 51  BedroomAbvGr   292 non-null    int64
 52  KitchenAbvGr   292 non-null    int64
 53  KitchenQual    292 non-null    object
 54  TotRmsAbvGrd   292 non-null    int64
 55  Functional     292 non-null    object
 56  Fireplaces     292 non-null    int64
 57  FireplaceQu    153 non-null    object
 58  GarageType     275 non-null    object
 59  GarageYrBlt    275 non-null    float64
 60  GarageFinish   275 non-null    object
 61  GarageCars     292 non-null    int64
 62  GarageArea     292 non-null    int64
 63  GarageQual     275 non-null    object
 64  GarageCond     275 non-null    object

 65  PavedDrive     292 non-null    object
 66  WoodDeckSF     292 non-null    int64
 67  OpenPorchSF    292 non-null    int64
 68  EnclosedPorch  292 non-null    int64
 69  3SsnPorch      292 non-null    int64
 70  ScreenPorch    292 non-null    int64
 71  PoolArea       292 non-null    int64
 72  PoolQC         0 non-null      float64
 73  Fence          44 non-null     object
 74  MiscFeature    10 non-null     object
 75  MiscVal        292 non-null    int64
 76  MoSold         292 non-null    int64
 77  YrSold         292 non-null    int64
 78  SaleType       292 non-null    object
 79  SaleCondition  292 non-null    object
dtypes: float64(4), int64(34), object(42)
memory usage: 182.6+ KB
```

The data set has 292 examples and 79 features. 42 of the features are object, 34 of the features are int, 4 of the features are float.

Some values are missing in data set.

Checking data type of training data set:

```
#let's check data types of training data set
pd.set_option('display.max_rows',None)
df_train.dtypes
```

```
Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
Street           object
Alley            object
LotShape         object
LandContour      object
Utilities        object
LotConfig        object
LandSlope        object
Neighborhood     object
Condition1       object
Condition2       object
BldgType         object
HouseStyle       object
OverallQual       int64
OverallCond       int64
YearBuilt         int64
YearRemodAdd      int64
RoofStyle        object
RoofMatl         object
Exterior1st      object
Exterior2nd      object
MasVnrType       object
MasVnrArea      float64
ExterQual        object
ExterCond        object
Foundation       object
BsmtQual         object
BsmtCond         object
```

```
Foundation        object
BsmtQual          object
BsmtCond          object
BsmtExposure      object
BsmtFinType1      object
BsmtFinSF1         int64
BsmtFinType2      object
BsmtFinSF2         int64
BsmtUnfSF          int64
TotalBsmtSF        int64
Heating           object
HeatingQC         object
CentralAir        object
Electrical        object
1stFlrSF           int64
2ndFlrSF           int64
LowQualFinSF       int64
GrLivArea          int64
BsmtFullBath       int64
BsmtHalfBath       int64
FullBath           int64
HalfBath           int64
BedroomAbvGr       int64
KitchenAbvGr       int64
KitchenQual       object
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars         int64
GarageArea         int64
GarageQual        object
GarageCond        object
```

```
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
PoolQC            object
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
dtype: object
```

Training data set have some values of int and some values of object asnd some others are of float type.

Checking data type of testing data:

```
#check testing data set data types
df_test.dtypes
```

```
Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
Street           object
Alley            object
LotShape         object
LandContour      object
Utilities        object
LotConfig        object
LandSlope        object
Neighborhood     object
Condition1       object
Condition2       object
BldgType         object
HouseStyle       object
OverallQual       int64
OverallCond       int64
YearBuilt         int64
YearRemodAdd      int64
RoofStyle        object
RoofMatl         object
Exterior1st      object
Exterior2nd      object
MasVnrType       object
MasVnrArea      float64
ExterQual        object
ExterCond        object
Foundation       object
BsmtQual         object
BsmtCond         object
```

```
BsmtExposure      object
BsmtFinType1      object
BsmtFinSF1         int64
BsmtFinType2      object
BsmtFinSF2         int64
BsmtUnfSF          int64
TotalBsmtSF        int64
Heating           object
HeatingQC         object
CentralAir        object
Electrical        object
1stFlrSF           int64
2ndFlrSF           int64
LowQualFinSF       int64
GrLivArea          int64
BsmtFullBath       int64
BsmtHalfBath       int64
FullBath           int64
HalfBath           int64
BedroomAbvGr       int64
KitchenAbvGr       int64
KitchenQual       object
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars         int64
GarageArea         int64
GarageQual        object
GarageCond        object
```

```
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
PoolQC           float64
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
dtype: object
```

Testing data set have some values of int and some values of object asnd some others are of float type.

Data Pre-processing

To predict the exact output values we have to fit the appropriate data in to model.For this we have to do processing on data to clean-up data set.

First we have to check whether the data set have missing values. If there are missing values then we have to fill that missing values by appropriate values.

Then after this we have to check the correlation between input variables.

We also have to check whether data set have outliers . If outliers are present then we have to remove that outliers with the help of zscore or IQR method.

We have to check skewness present in dataset. If skewness is present then we have to remove skewness using different methods such as np.log,np,sqrt,np.cbrt,np.power.

Checking missing values in training data set:

```
df_train.isnull().sum()
```

```
Id                  0
MSSubClass          0
MSZoning            0
LotFrontage       214
LotArea             0
Street              0
Alley            1091
LotShape            0
LandContour         0
Utilities           0
LotConfig           0
LandSlope           0
Neighborhood        0
Condition1          0
Condition2          0
BldgType            0
HouseStyle          0
OverallQual         0
OverallCond         0
YearBuilt           0
YearRemodAdd        0
RoofStyle           0
RoofMatl            0
Exterior1st         0
Exterior2nd         0
MasVnrType          7
MasVnrArea          7
ExterQual           0
ExterCond           0
Foundation          0
BsmtQual           30
BsmtCond           30
BsmtExposure       31
BsmtFinType1       30
```

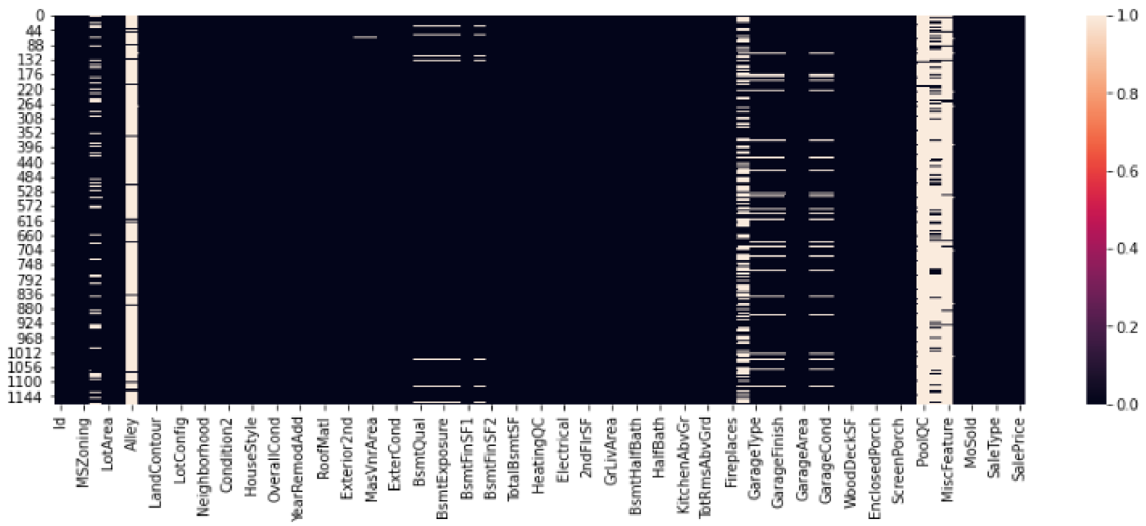
```
BsmtFinType1       30
BsmtFinSF1          0
BsmtFinType2       31
BsmtFinSF2          0
BsmtUnfSF           0
TotalBsmtSF         0
Heating             0
HeatingQC           0
CentralAir          0
Electrical          0
1stFlrSF            0
2ndFlrSF            0
LowQualFinSF        0
GrLivArea           0
BsmtFullBath        0
BsmtHalfBath        0
FullBath            0
HalfBath            0
BedroomAbvGr        0
KitchenAbvGr        0
KitchenQual         0
TotRmsAbvGrd        0
Functional          0
Fireplaces          0
FireplaceQu       551
GarageType         64
GarageYrBlt        64
GarageFinish       64
GarageCars          0
GarageArea          0
GarageQual         64
GarageCond         64
PavedDrive          0
```

```
WoodDeckSF          0
OpenPorchSF         0
EnclosedPorch       0
3SsnPorch           0
ScreenPorch         0
PoolArea            0
PoolQC           1161
Fence             931
MiscFeature      1124
MiscVal             0
MoSold              0
YrSold              0
SaleType            0
SaleCondition       0
SalePrice           0
dtype: int64
```

Training data set have missing values in some columns.

```
1 plt.figure(figsize=(15,5))
2 sns.heatmap(df_train.isnull())
```

<AxesSubplot:>



Above heatmap shows that training data set have missing values.

Checking missing values of testing data:

```
1 df_test.isnull().sum()
```

```
Id                0
MSSubClass        0
MSZoning          0
LotFrontage      45
LotArea           0
Street            0
Alley           278
LotShape          0
LandContour       0
Utilities         0
LotConfig         0
LandSlope         0
Neighborhood      0
Condition1        0
Condition2        0
BldgType          0
HouseStyle        0
OverallQual       0
OverallCond       0
YearBuilt         0
YearRemodAdd      0
RoofStyle         0
RoofMatl          0
Exterior1st       0
Exterior2nd       0
MasVnrType        1
MasVnrArea        1
ExterQual         0
ExterCond         0
Foundation        0
BsmtQual          7
BsmtCond          7
BsmtExposure      7
BsmtFinType1      7
```

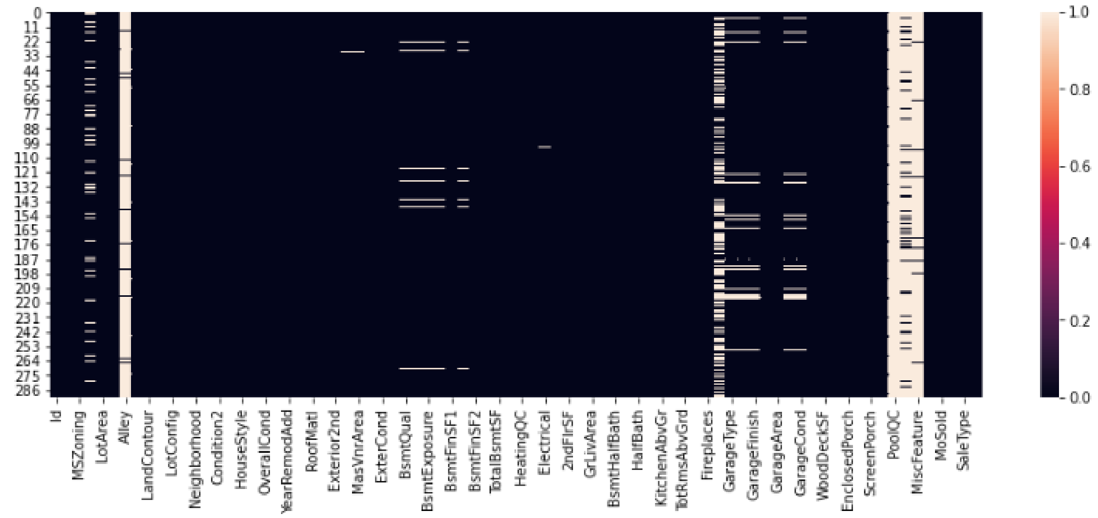
```
BsmtFinType2       7
BsmtFinSF2         0
BsmtUnfSF          0
TotalBsmtSF        0
Heating            0
HeatingQC          0
CentralAir         0
Electrical         1
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
BsmtFullBath       0
BsmtHalfBath       0
FullBath           0
HalfBath           0
BedroomAbvGr       0
KitchenAbvGr       0
KitchenQual        0
TotRmsAbvGrd       0
Functional         0
Fireplaces         0
FireplaceQu      139
GarageType        17
GarageYrBlt       17
GarageFinish      17
GarageCars         0
GarageArea         0
GarageQual        17
GarageCond        17
PavedDrive         0
```

```
WoodDeckSF         0
OpenPorchSF        0
EnclosedPorch      0
3SsnPorch          0
ScreenPorch        0
PoolArea           0
PoolQC           292
Fence            248
MiscFeature      282
MiscVal            0
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
dtype: int64
```

Testing data set have missing values in some columns.

```
plt.figure(figsize=(15,5))
sns.heatmap(df_test.isnull())
```

<AxesSubplot:>



Above heatmap also shows that testing data set have missing values.

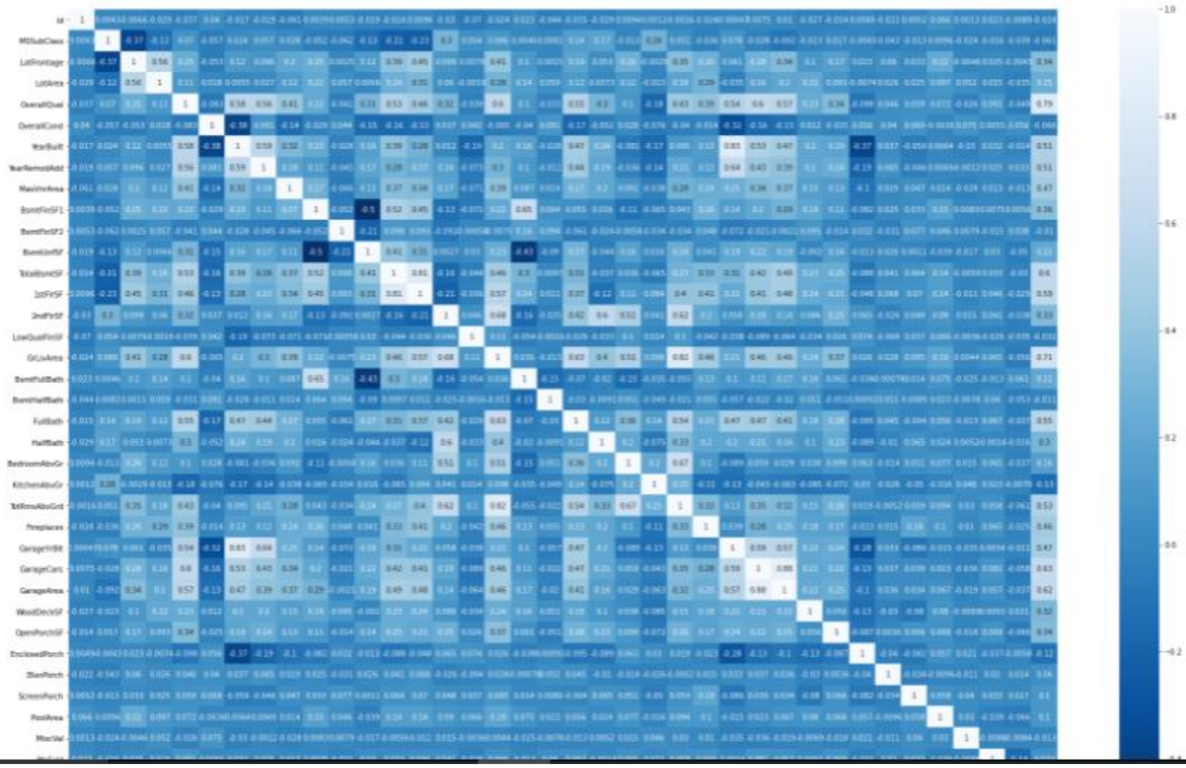
Checking correlation of training data set:

```
df_train.corr()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDeckS
Id	1.000000	0.004259	-0.006629	-0.029212	-0.036965	0.039761	-0.016942	-0.018590	-0.060652	0.003868	...	-0.02749
MSSubClass	0.004259	1.000000	-0.365220	-0.124151	0.070462	-0.056978	0.023988	0.056618	0.027868	-0.052236	...	-0.02260
LotFrontage	-0.006629	-0.365220	1.000000	0.557257	0.247809	-0.053345	0.118554	0.096050	0.202225	0.247780	...	0.10175
LotArea	-0.029212	-0.124151	0.557257	1.000000	0.107188	0.017513	0.005506	0.027228	0.121448	0.221851	...	0.21672
OverallQual	-0.036965	0.070462	0.247809	0.107188	1.000000	-0.083167	0.575800	0.555945	0.409163	0.219643	...	0.22713
OverallCond	0.039761	-0.056978	-0.053345	0.017513	-0.083167	1.000000	-0.377731	0.080669	-0.137882	-0.028810	...	0.01229
YearBuilt	-0.016942	0.023988	0.118554	0.005506	0.575800	-0.377731	1.000000	0.592829	0.323006	0.227933	...	0.20483
YearRemodAdd	-0.018590	0.056618	0.096050	0.027228	0.555945	0.080669	0.592829	1.000000	0.181869	0.114430	...	0.19741
MasVnrArea	-0.060652	0.027868	0.202225	0.121448	0.409163	-0.137882	0.323006	0.181869	1.000000	0.267066	...	0.15197
BsmtFinSF1	0.003868	-0.052236	0.247780	0.221851	0.219643	-0.028810	0.227933	0.114430	0.267066	1.000000	...	0.19293
BsmtFinSF2	0.005269	-0.062403	0.002514	0.056656	-0.040893	0.044336	-0.027682	-0.044694	-0.065723	-0.052145	...	0.09469
BsmtUnfSF	-0.019494	-0.134170	0.123943	0.006600	0.308676	-0.146384	0.155559	0.174732	0.109850	-0.499861	...	-0.00196
TotalBsmtSF	-0.013812	-0.214042	0.386261	0.259733	0.528285	-0.162481	0.386265	0.280720	0.366833	0.518940	...	0.23484
1stFlrSF	0.009647	-0.227927	0.448186	0.312843	0.458758	-0.134420	0.279450	0.233384	0.339938	0.445876	...	0.23546
2ndFlrSF	-0.029671	0.300366	0.099250	0.059803	0.316624	0.036668	0.011834	0.155102	0.173358	-0.127656	...	0.08566
LowQualFinSF	-0.070180	0.053737	0.007885	-0.001915	-0.039295	0.041877	-0.189044	-0.072526	-0.070518	-0.070932	...	-0.03355
GrLivArea	-0.024325	0.086448	0.410414	0.281360	0.599700	-0.065006	0.198644	0.295048	0.387891	0.217160	...	0.24252
BsmtFullBath	0.023027	0.004556	0.104255	0.142387	0.101732	-0.039680	0.164983	0.104643	0.086720	0.645126	...	0.16177
BsmtHalfBath	-0.043572	0.008207	0.001528	0.059282	-0.030702	0.091016	-0.028161	-0.011375	0.014198	0.063895	...	0.05103
FullBath	-0.015187	0.140807	0.189321	0.123197	0.548824	-0.171931	0.471264	0.444446	0.268545	0.054511	...	0.17696
HalfBath	-0.028512	0.168423	0.053168	0.007271	0.296134	-0.052125	0.243227	0.194943	0.200926	0.015767	...	0.10166
BedroomAbvGr	0.009376	-0.013283	0.264010	0.117351	0.099639	0.028393	-0.080639	-0.035847	0.091717	-0.114888	...	0.03799

```
plt.figure(figsize=(30,20))
sns.heatmap(df_train.corr(),annot=True,cmap='Blues_r')
```

```
<AxesSubplot:>
```



Corr() function describes correlation between different variables.

Heatmap shows how one variables is correlated with others.

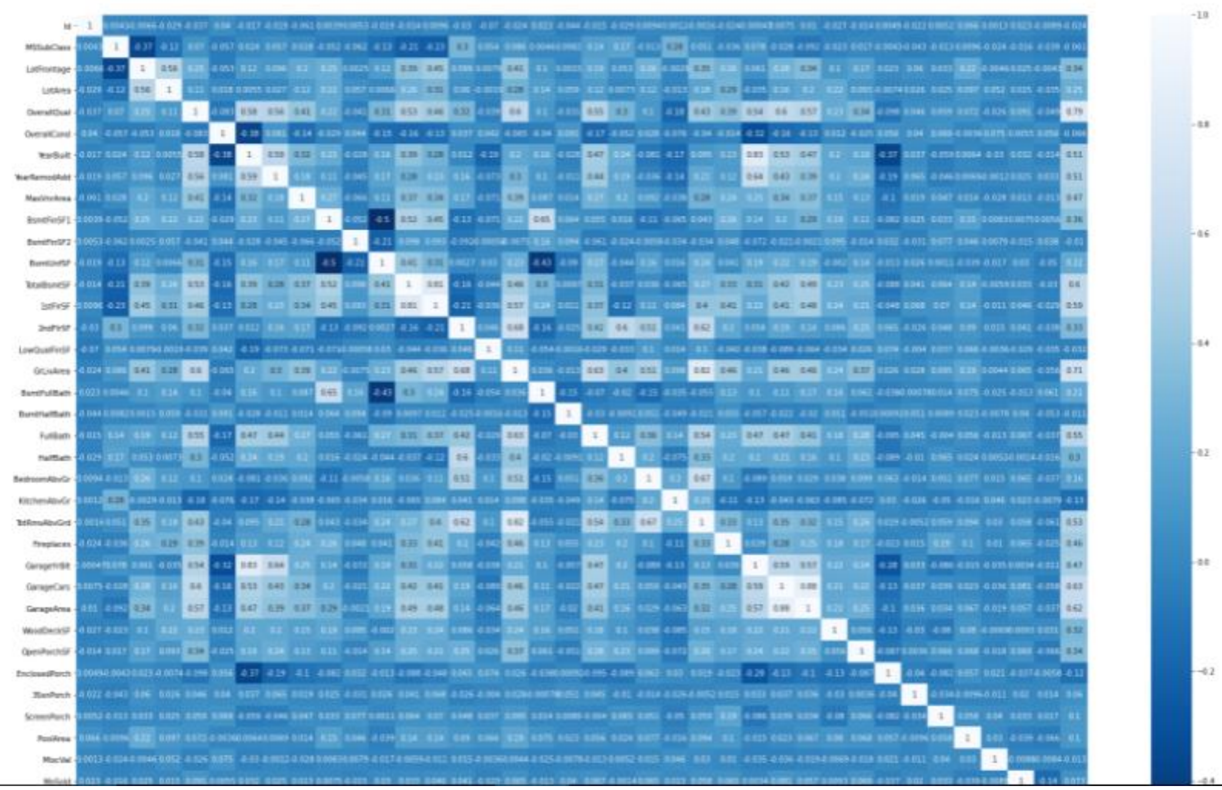
Checking correlation of testing data set:

```
1 df_test.corr()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDeckS
Id	1.000000	0.035247	-0.017848	-0.045497	0.005823	-0.089945	-0.000174	-0.036955	-0.012331	-0.040462	...	-0.0336
MSSubClass	0.035247	1.000000	-0.487565	-0.186654	-0.116077	-0.068113	0.041932	-0.023317	0.002761	-0.142908	...	0.0281
LotFrontage	-0.017848	-0.487565	1.000000	0.383137	0.267153	-0.108327	0.152067	0.059014	0.151632	0.165505	...	0.0075
LotArea	-0.045497	-0.186654	0.383137	1.000000	0.109161	-0.071113	0.037757	-0.022957	0.062943	0.209632	...	0.0591
OverallQual	0.005823	-0.116077	0.267153	0.109161	1.000000	-0.131891	0.560092	0.528983	0.424314	0.328421	...	0.2885
OverallCond	-0.089945	-0.068113	-0.108327	-0.071113	-0.131891	1.000000	-0.366830	0.045747	-0.082467	-0.126968	...	-0.0781
YearBuilt	-0.000174	0.041932	0.152067	0.037757	0.560092	-0.366830	1.000000	0.593138	0.284734	0.343374	...	0.3118
YearRemodAdd	-0.036955	-0.023317	0.059014	-0.022957	0.528983	0.045747	0.593138	1.000000	0.169188	0.191460	...	0.2437
MasVnrArea	-0.012331	0.002761	0.151632	0.062943	0.424314	-0.082467	0.284734	0.169188	1.000000	0.254935	...	0.1961
BsmtFinSF1	-0.040462	-0.142908	0.165505	0.209632	0.328421	-0.126968	0.343374	0.191460	0.254935	1.000000	...	0.2549
BsmtFinSF2	-0.051283	-0.079328	0.268758	0.277855	-0.139583	0.021459	-0.140391	-0.170299	-0.101817	-0.040759	...	-0.0513
BsmtUnfSF	0.041345	-0.169129	0.173770	-0.029717	0.306051	-0.095575	0.122930	0.211142	0.136442	-0.474047	...	-0.0224
TotalBsmtSF	-0.019326	-0.336822	0.420828	0.283111	0.578468	-0.213289	0.416016	0.336976	0.353100	0.538182	...	0.2177
1stFlrSF	0.019726	-0.350270	0.496136	0.286714	0.553431	-0.197177	0.299125	0.274479	0.369103	0.446604	...	0.2306
2ndFlrSF	0.144675	0.338926	-0.016069	0.030128	0.205253	-0.006572	0.005011	0.075768	0.180463	-0.179595	...	0.1189
LowQualFinSF	0.087688	0.012343	0.202904	0.030371	0.016101	-0.071717	-0.160248	-0.006942	-0.060730	-0.029585	...	0.0140
GrLivArea	0.139969	0.030615	0.360840	0.231721	0.565494	-0.151462	0.205685	0.257875	0.407814	0.168317	...	0.2638
BsmtFullBath	-0.077773	-0.000676	0.092577	0.212162	0.150559	-0.121640	0.280054	0.182556	0.079406	0.667976	...	0.2332
BsmtHalfBath	0.062190	-0.041577	-0.055672	0.020883	-0.076885	0.230622	-0.077984	-0.017015	0.075530	0.082806	...	-0.0004
FullBath	0.083174	0.095903	0.250443	0.141118	0.558577	-0.288175	0.455999	0.416337	0.311178	0.076450	...	0.2347
HalfBath	0.146049	0.213601	0.040245	0.034980	0.177858	-0.103079	0.243680	0.135780	0.205684	-0.047032	...	0.1319
BedroomAbvGr	0.152676	-0.061750	0.249869	0.134990	0.108994	-0.061694	-0.026005	-0.058289	0.153659	-0.076283	...	0.0772
KitchenAbvGr	0.007486	0.275461	-0.011250	-0.030264	-0.205815	-0.128606	-0.201801	-0.187750	-0.036277	-0.143169	...	-0.1086
TotRmsAbvGrd	0.132171	0.002973	0.356616	0.211624	0.409677	-0.133094	0.099718	0.136495	0.290170	0.047362	...	0.2094
Fireplaces	-0.000541	-0.084663	0.283172	0.250416	0.425392	-0.071834	0.205425	0.088244	0.273130	0.271326	...	0.2981

```
plt.figure(figsize=(30,20))
sns.heatmap(df_train.corr(),annot=True,cmap='Blues_r')
```

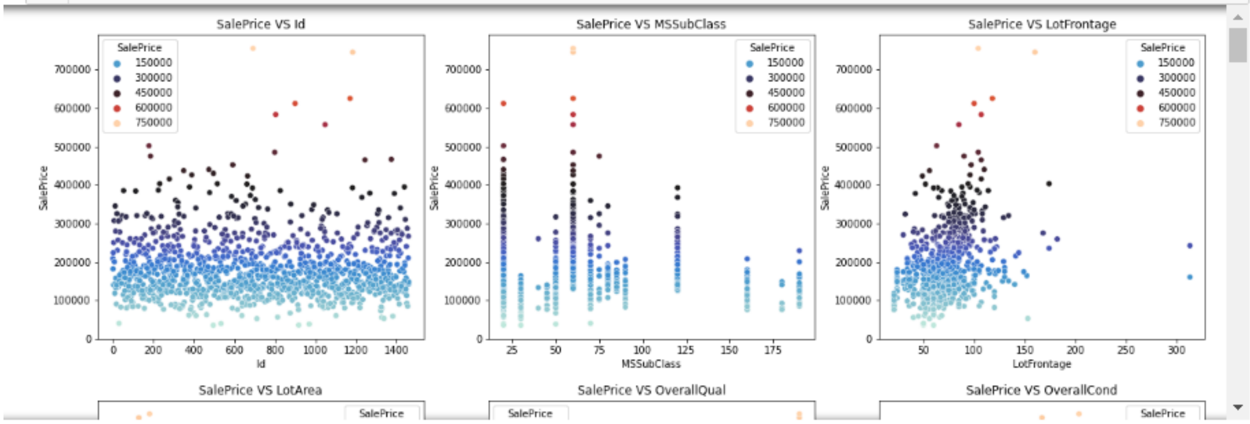
<AxesSubplot:>



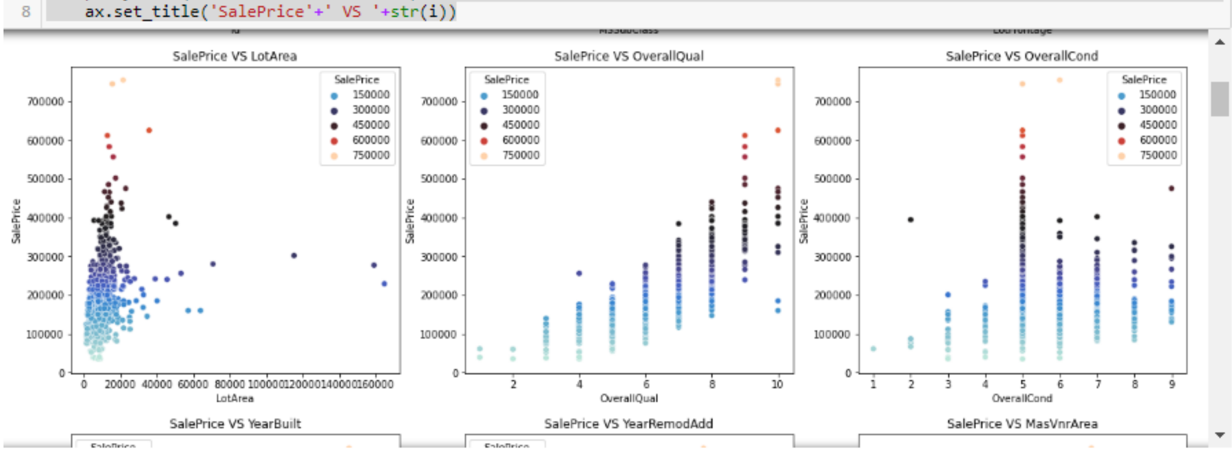
Visualization:

Scatterplot:

```
]: #Visualising numerical predictor variables with Target Variables
df_num = df_train.select_dtypes(include=['int64','float64'])
fig,axs= plt.subplots(12,3,figsize=(20,80))
for i,ax in zip(df_num.columns,axs.flatten()):
    sns.scatterplot(x=i, y='SalePrice', hue='SalePrice',data=df_num,ax=ax,palette='icefire')
    plt.xlabel(i,fontsize=12)
    plt.ylabel('SalePrice',fontsize=12)
    ax.set_title('SalePrice'+' VS '+str(i))
```




```
#Visualising numerical predictor variables with Target Variables
df_num = df_train.select_dtypes(include=['int64','float64'])
fig,axs= plt.subplots(12,3,figsize=(20,80))
for i,ax in zip(df_num.columns,axs.flatten()):
    sns.scatterplot(x=i, y='SalePrice', hue='SalePrice',data=df_num,ax=ax,palette='icefire')
    plt.xlabel(i,fontsize=12)
    plt.ylabel('SalePrice',fontsize=12)
    ax.set_title('SalePrice'+' VS '+str(i))
```



Scatterplot shows that how dependenet variable varies when indepenedent variables are changing.For exp: sale price is increasing when there is increase in GrLivArea and OverallQual.

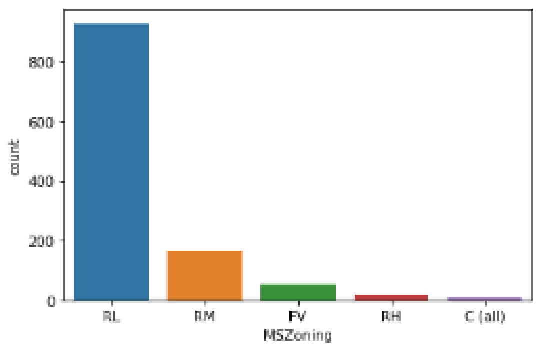
Fence variable don’t have much impact on sale price.

Countplot:
Countplot is used to describe categorical data . following fig of countplot of categorical data describe different categoris og data and their value.

```
print(df_train['MSZoning'].value_counts())
sns.countplot(df_train["MSZoning"])
```

```
RL         928
RM         163
FV          52
RH          16
C (all)      9
Name: MSZoning, dtype: int64
```

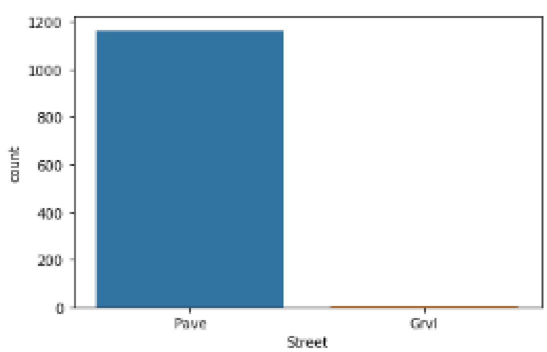
```
<AxesSubplot:xlabel='MSZoning', ylabel='count'>
```




```
print(df_train['Street'].value_counts())
sns.countplot(df_train["Street"])
```

```
Pave    1164
Grvl       4
Name: Street, dtype: int64
```

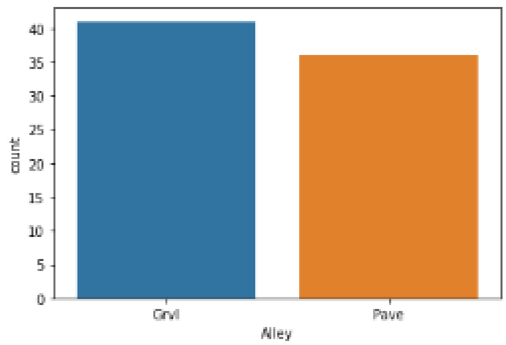
```
<AxesSubplot:xlabel='Street', ylabel='count'>
```



```
1 print(df_train['Alley'].value_counts())
2 sns.countplot(df_train["Alley"])
```

```
Grvl    41
Pave    36
Name: Alley, dtype: int64
```

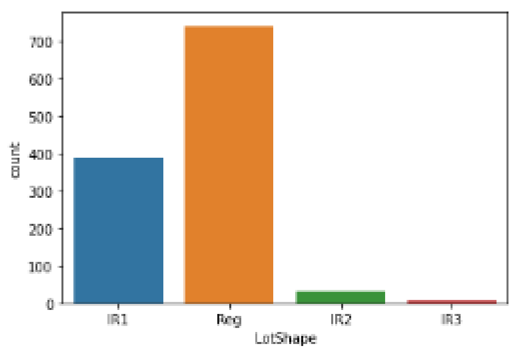
```
<AxesSubplot:xlabel='Alley', ylabel='count'>
```




```
1 print(df_train['LotShape'].value_counts())
2 sns.countplot(df_train["LotShape"])
```

```
Reg    740
IR1    390
IR2     32
IR3      6
Name: LotShape, dtype: int64
```

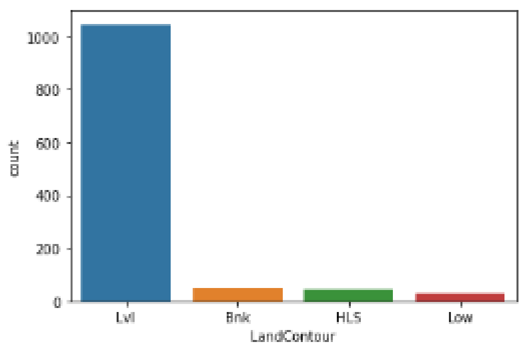
```
<AxesSubplot:xlabel='LotShape', ylabel='count'>
```



```python
print(df_train['LandContour'].value_counts())
sns.countplot(df_train["LandContour"])
```

```
Lvl    1046
Bnk      50
HLS      42
Low      30
Name: LandContour, dtype: int64
```

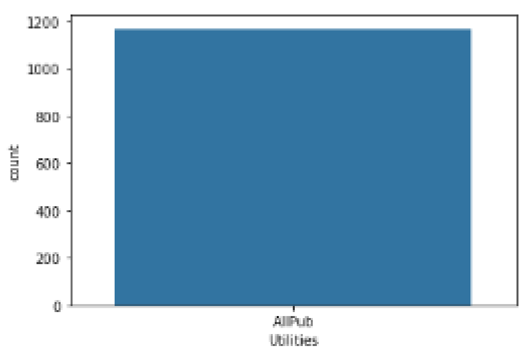
```
<AxesSubplot:xlabel='LandContour', ylabel='count'>
```




```python
print(df_train['Utilities'].value_counts())
sns.countplot(df_train["Utilities"])
```

```
AllPub    1168
Name: Utilities, dtype: int64
```

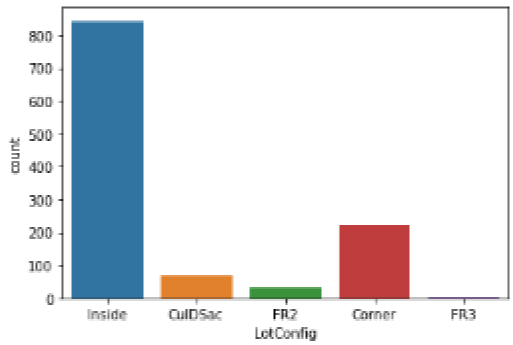
```
<AxesSubplot:xlabel='Utilities', ylabel='count'>
```



```
print(df_train['LotConfig'].value_counts())
sns.countplot(df_train["LotConfig"])
```

```
Inside     842
Corner     222
CulDSac     69
FR2         33
FR3          2
Name: LotConfig, dtype: int64
```

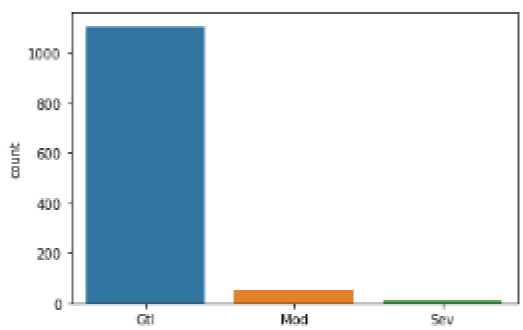
```
<AxesSubplot:xlabel='LotConfig', ylabel='count'>
```




```
print(df_train['LandSlope'].value_counts())
sns.countplot(df_train["LandSlope"])
```

```
Gtl    1105
Mod      51
Sev      12
Name: LandSlope, dtype: int64
```

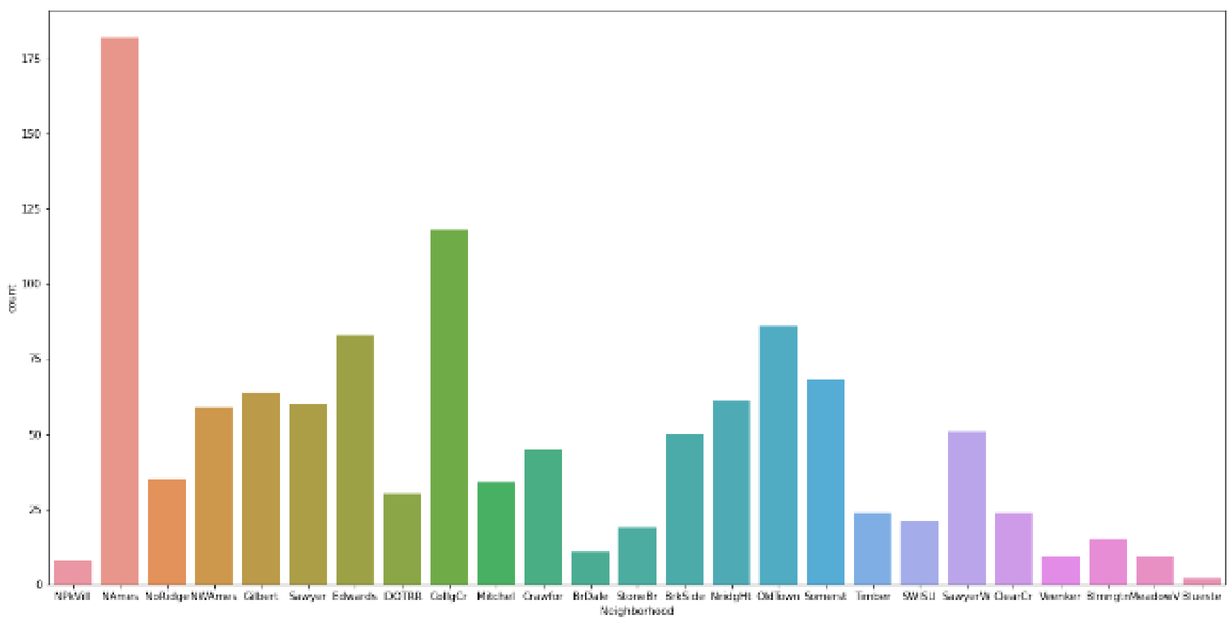
```
<AxesSubplot:xlabel='LandSlope', ylabel='count'>
```



```
print(df_train['Neighborhood'].value_counts())
plt.figure(figsize=(20,10))
sns.countplot(df_train["Neighborhood"])
```

```
NAmes      182
CollgCr    118
OldTown     86
Edwards     83
Somerst     68
Gilbert     64
NridgHt     61
Sawyer      60
NWAmes      59
SawyerW     51
BrkSide     50
Crawfor     45
NoRidge     35
Mitchel     34
IDOTRR      30
ClearCr     24
Timber      24
SWISU       21
StoneBr     19
Blmngtn     15
BrDale      11
MeadowV      9
Veenker      9
NPkVill      8
Blueste      2
Name: Neighborhood, dtype: int64
```

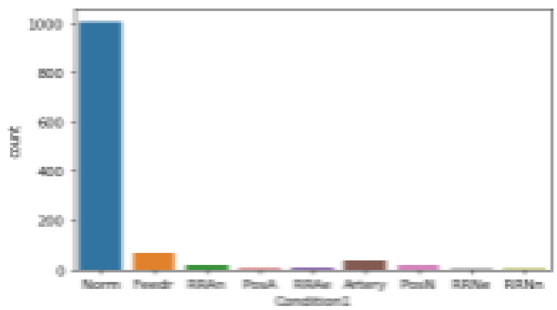
```
<AxesSubplot:xlabel='Neighborhood', ylabel='count'>
```



]:
```
print(df_train['Condition1'].value_counts())
sns.countplot(df_train["Condition1"])
```

```
Norm      1005
Feedr       67
Artery      38
RRAn        20
PosN        17
RRAe         9
PosA         6
RRNn         4
RRNe         2
Name: Condition1, dtype: int64
```

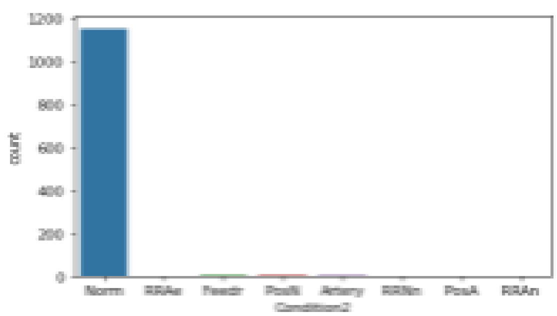
]: <AxesSubplot:xlabel='Condition1', ylabel='count'>



]:
```
print(df_train['Condition2'].value_counts())
sns.countplot(df_train["Condition2"])
```

```
Norm      1154
Feedr        6
PosN         2
Artery       2
RRAe         1
RRAn         1
RRNn         1
PosA         1
Name: Condition2, dtype: int64
```

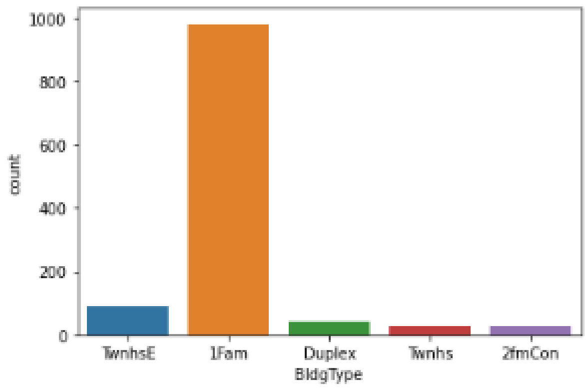
]: <AxesSubplot:xlabel='Condition2', ylabel='count'>



```
0]:  1 print(df_train['BldgType'].value_counts())
     2 sns.countplot(df_train["BldgType"])
```

```
1Fam      981
TwnhsE     90
Duplex     41
Twnhs      29
2fmCon     27
Name: BldgType, dtype: int64
```

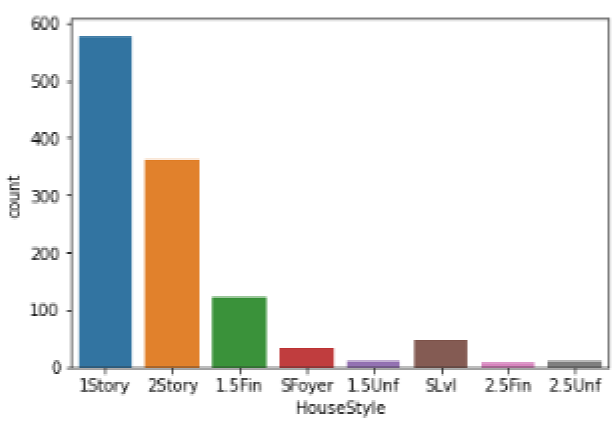
```
0]: <AxesSubplot:xlabel='BldgType', ylabel='count'>
```




```
print(df_train['HouseStyle'].value_counts())
sns.countplot(df_train["HouseStyle"])
```

```
1Story    578
2Story    361
1.5Fin    121
SLvl       47
SFoyer     32
1.5Unf     12
2.5Unf     10
2.5Fin      7
Name: HouseStyle, dtype: int64
```

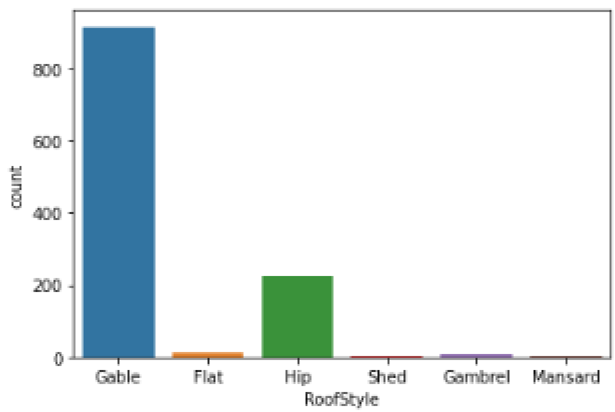
```
<AxesSubplot:xlabel='HouseStyle', ylabel='count'>
```



```
print(df_train['RoofStyle'].value_counts())
sns.countplot(df_train["RoofStyle"])
```

```
Gable      915
Hip        225
Flat        12
Gambrel      9
Mansard      5
Shed         2
Name: RoofStyle, dtype: int64
```

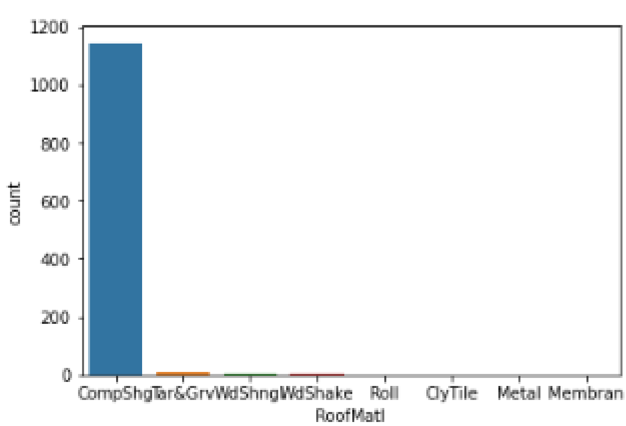
```
<AxesSubplot:xlabel='RoofStyle', ylabel='count'>
```



:
```
print(df_train['RoofMatl'].value_counts())
sns.countplot(df_train["RoofMatl"])
```

```
CompShg    1144
Tar&Grv      10
WdShngl       6
WdShake       4
Membran       1
Roll          1
ClyTile       1
Metal         1
Name: RoofMatl, dtype: int64
```

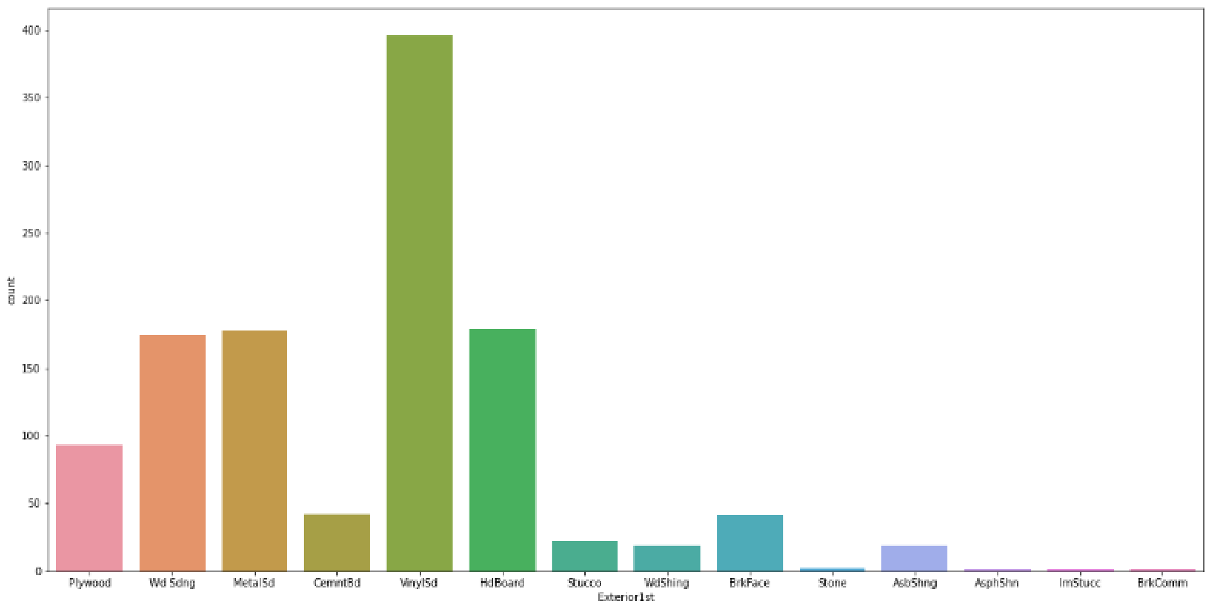
:
```
<AxesSubplot:xlabel='RoofMatl', ylabel='count'>
```



```
print(df_train['Exterior1st'].value_counts())
plt.figure(figsize=(20,10))
sns.countplot(df_train["Exterior1st"])
```

```
VinylSd    396
HdBoard    179
MetalSd    178
Wd Sdng    174
Plywood     93
CemntBd     42
BrkFace     41
Stucco      22
AsbShng     19
WdShing     19
Stone        2
AsphShn      1
ImStucc      1
BrkComm      1
Name: Exterior1st, dtype: int64
```

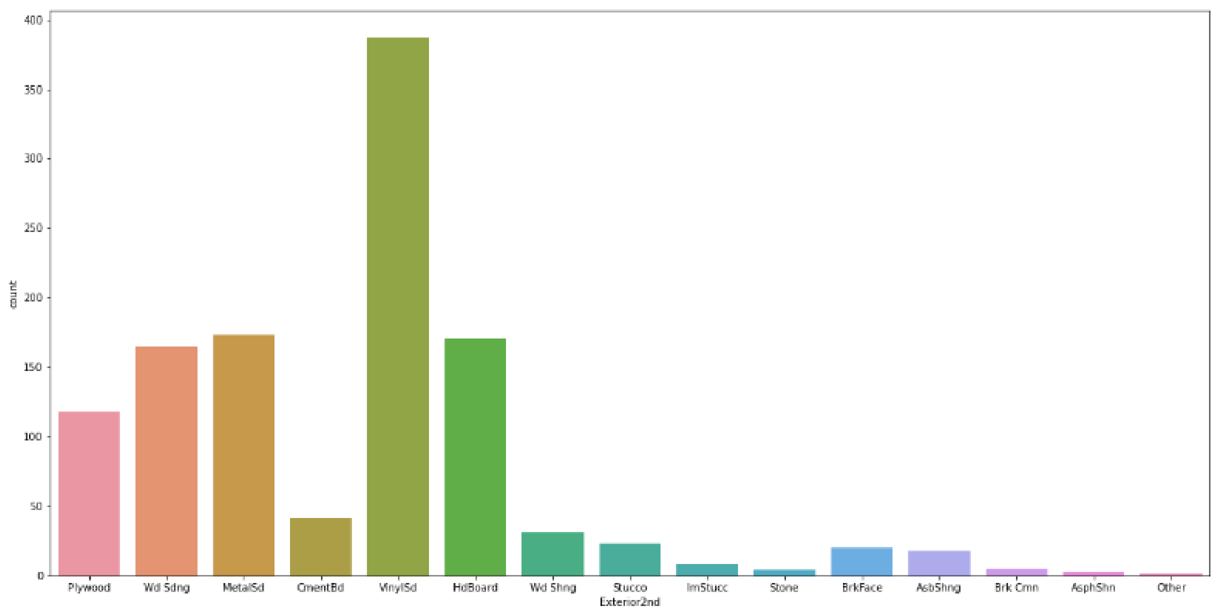
```
<AxesSubplot:xlabel='Exterior1st', ylabel='count'>
```



```
print(df_train['Exterior2nd'].value_counts())
plt.figure(figsize=(20,10))
sns.countplot(df_train["Exterior2nd"])
```

```
VinylSd    387
MetalSd    173
HdBoard    170
Wd Sdng    165
Plywood    118
CmentBd     42
Wd Shng     31
Stucco      23
BrkFace     20
AsbShng     18
ImStucc      8
Brk Cmn      5
Stone        4
AsphShn      3
Other        1
Name: Exterior2nd, dtype: int64
```

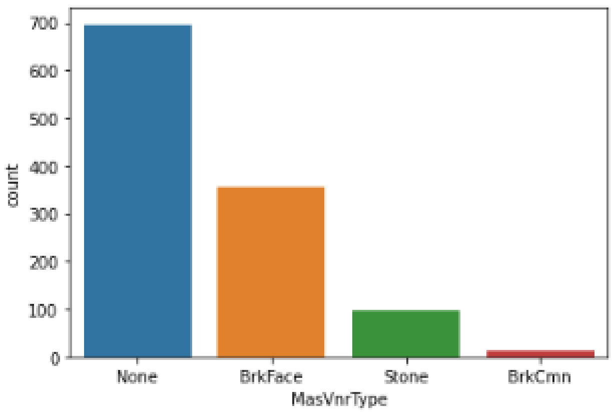
```
<AxesSubplot:xlabel='Exterior2nd', ylabel='count'>
```



```python
print(df_train['MasVnrType'].value_counts())

sns.countplot(df_train["MasVnrType"])
```

```
None       696
BrkFace    354
Stone       98
BrkCmn      13
Name: MasVnrType, dtype: int64
```

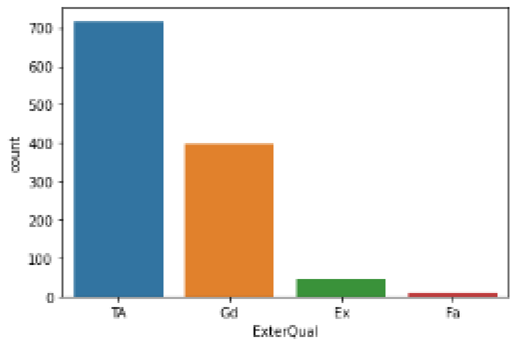
```
<AxesSubplot:xlabel='MasVnrType', ylabel='count'>
```



```
1 print(df_train['ExterQual'].value_counts())
2 sns.countplot(df_train["ExterQual"])
```

```
TA    717
Gd    397
Ex     43
Fa     11
Name: ExterQual, dtype: int64
```

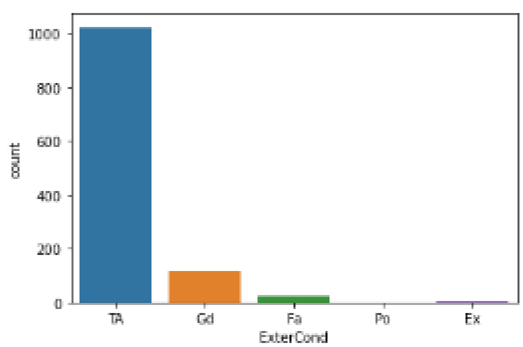
```
<AxesSubplot:xlabel='ExterQual', ylabel='count'>
```




```
1 print(df_train['ExterCond'].value_counts())
2 sns.countplot(df_train["ExterCond"])
```

```
TA    1022
Gd     117
Fa      26
Ex       2
Po       1
Name: ExterCond, dtype: int64
```

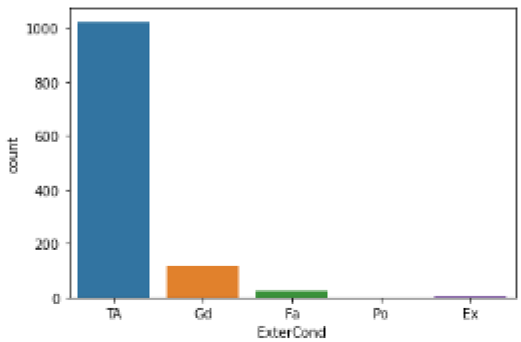
```
<AxesSubplot:xlabel='ExterCond', ylabel='count'>
```



[888]:
```
1 print(df_train['ExterCond'].value_counts())
2 sns.countplot(df_train["ExterCond"])
```

```
TA    1022
Gd     117
Fa      26
Ex       2
Po       1
Name: ExterCond, dtype: int64
```

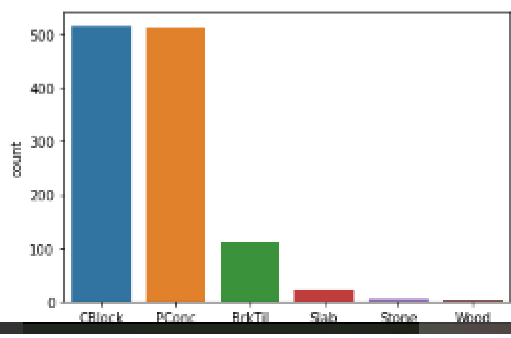
t[888]: <AxesSubplot:xlabel='ExterCond', ylabel='count'>



[889]:
```
1 print(df_train['Foundation'].value_counts())
2 sns.countplot(df_train["Foundation"])
```

```
CBlock    516
PConc     513
BrkTil    112
Slab       21
Stone       5
Wood        1
Name: Foundation, dtype: int64
```

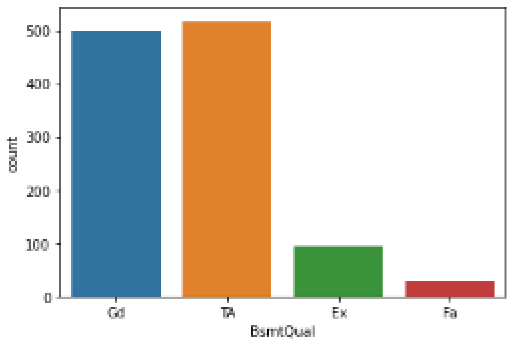
t[889]: <AxesSubplot:xlabel='Foundation', ylabel='count'>



```
print(df_train['BsmtQual'].value_counts())
sns.countplot(df_train["BsmtQual"])
```

```
TA    517
Gd    498
Ex     94
Fa     29
Name: BsmtQual, dtype: int64
```

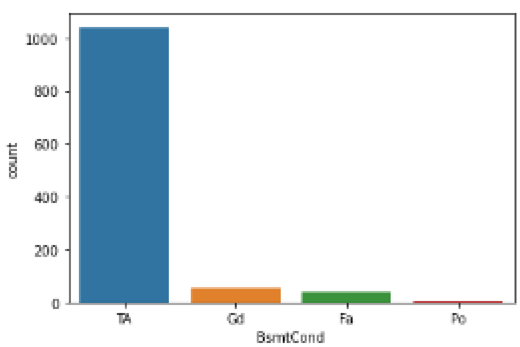
```
<AxesSubplot:xlabel='BsmtQual', ylabel='count'>
```




```
print(df_train['BsmtCond'].value_counts())
sns.countplot(df_train["BsmtCond"])
```

```
TA    1041
Gd      56
Fa      39
Po       2
Name: BsmtCond, dtype: int64
```

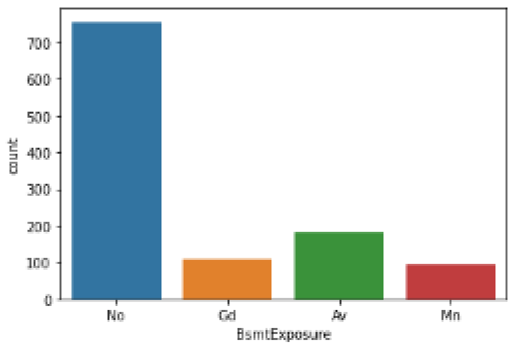
```
<AxesSubplot:xlabel='BsmtCond', ylabel='count'>
```



```python
print(df_train['BsmtExposure'].value_counts())
sns.countplot(df_train["BsmtExposure"])
```

```
No    756
Av    180
Gd    108
Mn     93
Name: BsmtExposure, dtype: int64
```

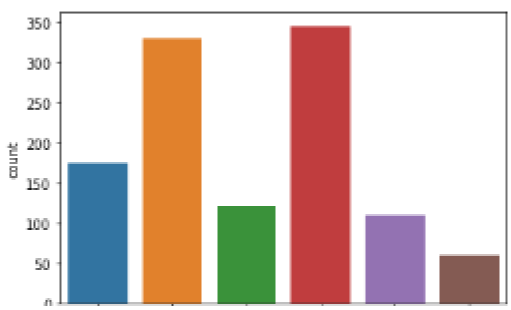
```
<AxesSubplot:xlabel='BsmtExposure', ylabel='count'>
```




```python
print(df_train['BsmtFinType1'].value_counts())
sns.countplot(df_train["BsmtFinType1"])
```

```
Unf    345
GLQ    330
ALQ    174
BLQ    121
Rec    109
LwQ     59
Name: BsmtFinType1, dtype: int64
```

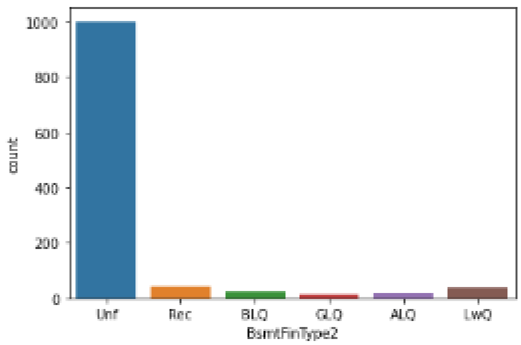
```
<AxesSubplot:xlabel='BsmtFinType1', ylabel='count'>
```



```
print(df_train['BsmtFinType2'].value_counts())
sns.countplot(df_train["BsmtFinType2"])
```

```
Unf    1002
Rec      43
LwQ      40
BLQ      24
ALQ      16
GLQ      12
Name: BsmtFinType2, dtype: int64
```

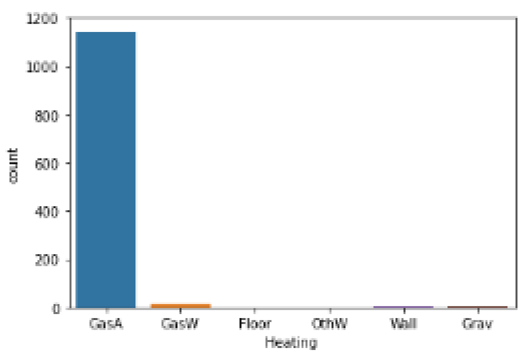
```
<AxesSubplot:xlabel='BsmtFinType2', ylabel='count'>
```




```
print(df_train['Heating'].value_counts())
sns.countplot(df_train["Heating"])
```

```
GasA     1143
GasW       14
Grav        5
Wall        4
Floor       1
OthW        1
Name: Heating, dtype: int64
```

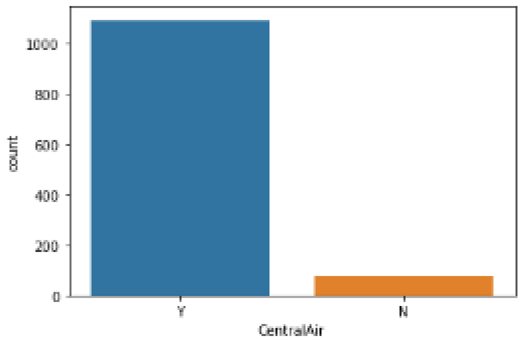
```
<AxesSubplot:xlabel='Heating', ylabel='count'>
```



```
print(df_train['CentralAir'].value_counts())
sns.countplot(df_train["CentralAir"])
```

```
Y    1090
N      78
Name: CentralAir, dtype: int64
```

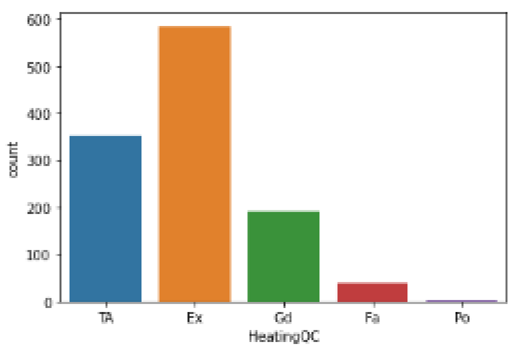
```
<AxesSubplot:xlabel='CentralAir', ylabel='count'>
```




```
print(df_train['HeatingQC'].value_counts())
sns.countplot(df_train["HeatingQC"])
```

```
Ex    585
TA    352
Gd    192
Fa     38
Po      1
Name: HeatingQC, dtype: int64
```

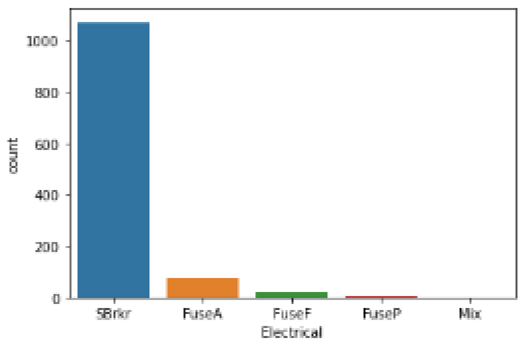
```
<AxesSubplot:xlabel='HeatingQC', ylabel='count'>
```



```
1 print(df_train['Electrical'].value_counts())
2 sns.countplot(df_train["Electrical"])
```

```
SBrkr    1070
FuseA      74
FuseF      21
FuseP       2
Mix         1
Name: Electrical, dtype: int64
```

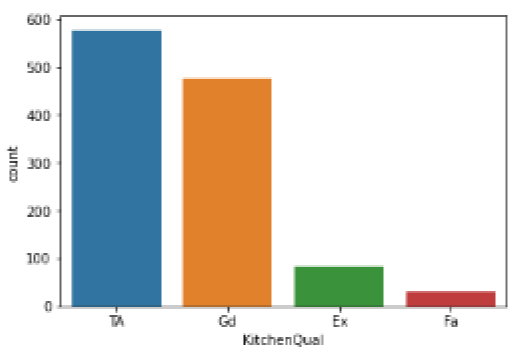
```
<AxesSubplot:xlabel='Electrical', ylabel='count'>
```




```
1 print(df_train['KitchenQual'].value_counts())
2 sns.countplot(df_train["KitchenQual"])
```

```
TA    578
Gd    478
Ex     82
Fa     30
Name: KitchenQual, dtype: int64
```

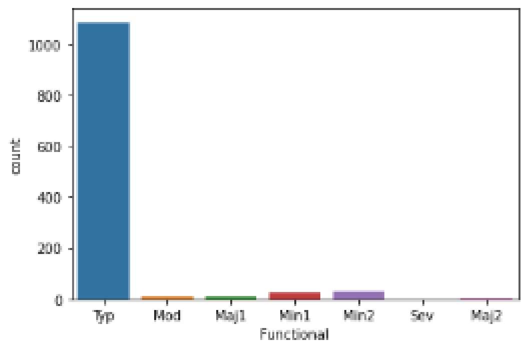
```
<AxesSubplot:xlabel='KitchenQual', ylabel='count'>
```



```
print(df_train['Functional'].value_counts())
sns.countplot(df_train["Functional"])
```

```
Typ     1085
Min2      30
Min1      25
Mod       12
Maj1      11
Maj2       4
Sev        1
Name: Functional, dtype: int64
```

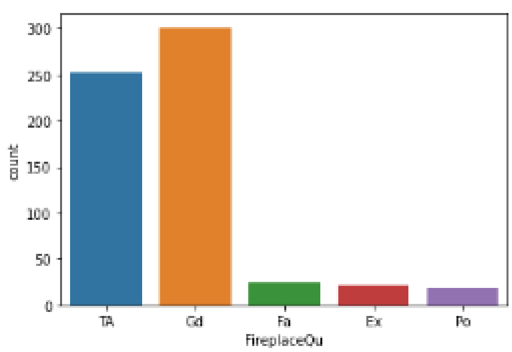
```
<AxesSubplot:xlabel='Functional', ylabel='count'>
```




```
print(df_train['FireplaceQu'].value_counts())
sns.countplot(df_train["FireplaceQu"])
```

```
Gd    301
TA    252
Fa     25
Ex     21
Po     18
Name: FireplaceQu, dtype: int64
```

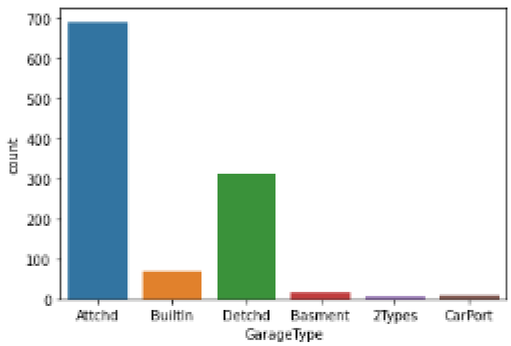
```
<AxesSubplot:xlabel='FireplaceQu', ylabel='count'>
```



```
1  print(df_train['GarageType'].value_counts())
2  sns.countplot(df_train["GarageType"])
```

```
Attchd     691
Detchd     314
BuiltIn     70
Basment     16
CarPort      8
2Types       5
Name: GarageType, dtype: int64
```

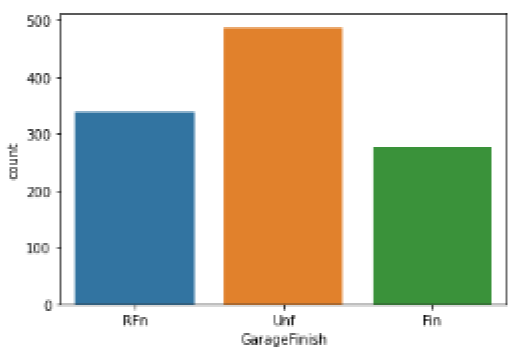
```
<AxesSubplot:xlabel='GarageType', ylabel='count'>
```




```
1  print(df_train['GarageFinish'].value_counts())
2  sns.countplot(df_train["GarageFinish"])
```

```
Unf    487
RFn    339
Fin    278
Name: GarageFinish, dtype: int64
```

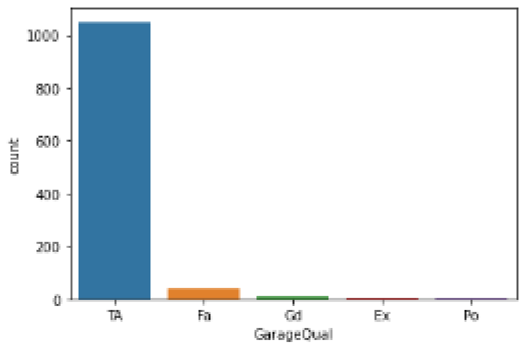
```
<AxesSubplot:xlabel='GarageFinish', ylabel='count'>
```



```
2 sns.countplot(df_train["GarageQual"])
```

```
TA    1050
Fa      39
Gd      11
Po       2
Ex       2
Name: GarageQual, dtype: int64
```

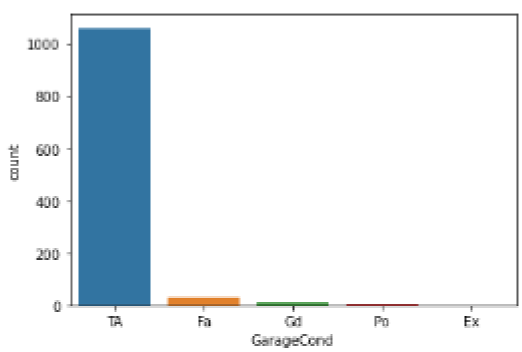
<AxesSubplot:xlabel='GarageQual', ylabel='count'>




```
1 print(df_train['GarageCond'].value_counts())
2 sns.countplot(df_train["GarageCond"])
```

```
TA    1061
Fa      28
Gd       8
Po       6
Ex       1
Name: GarageCond, dtype: int64
```

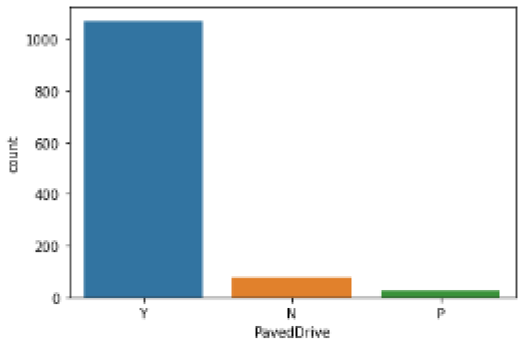
<AxesSubplot:xlabel='GarageCond', ylabel='count'>



```
print(df_train['PavedDrive'].value_counts())
sns.countplot(df_train["PavedDrive"])
```

```
Y    1071
N      74
P      23
Name: PavedDrive, dtype: int64
```

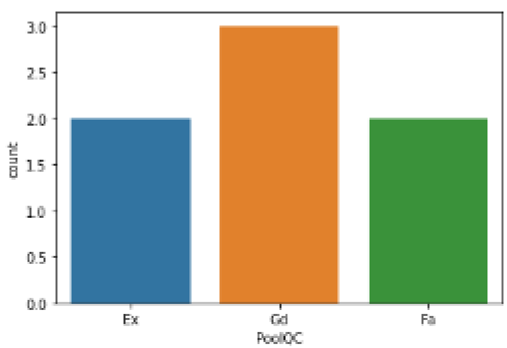
<AxesSubplot:xlabel='PavedDrive', ylabel='count'>




```
print(df_train['PoolQC'].value_counts())
sns.countplot(df_train["PoolQC"])
```

```
Gd    3
Fa    2
Ex    2
Name: PoolQC, dtype: int64
```

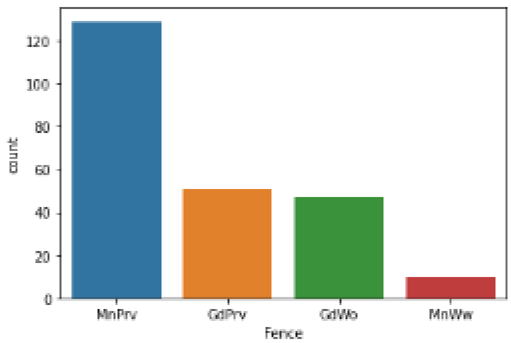
<AxesSubplot:xlabel='PoolQC', ylabel='count'>



```
print(df_train['Fence'].value_counts())
sns.countplot(df_train["Fence"])
```

```
MnPrv    129
GdPrv     51
GdWo      47
MnWw      10
Name: Fence, dtype: int64
```

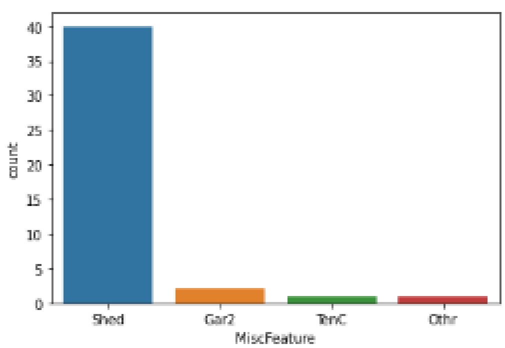
```
<AxesSubplot:xlabel='Fence', ylabel='count'>
```




```
print(df_train['MiscFeature'].value_counts())
sns.countplot(df_train["MiscFeature"])
```

```
Shed    40
Gar2     2
Othr     1
TenC     1
Name: MiscFeature, dtype: int64
```

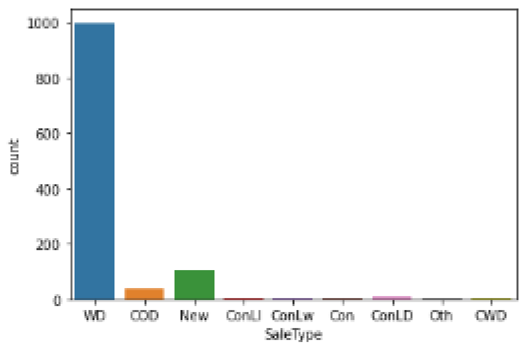
```
<AxesSubplot:xlabel='MiscFeature', ylabel='count'>
```



```
print(df_train['SaleType'].value_counts())
sns.countplot(df_train["SaleType"])
```

```
WD        999
New       106
COD        38
ConLD       8
ConLI       5
ConLw       4
Oth         3
CWD         3
Con         2
Name: SaleType, dtype: int64
```

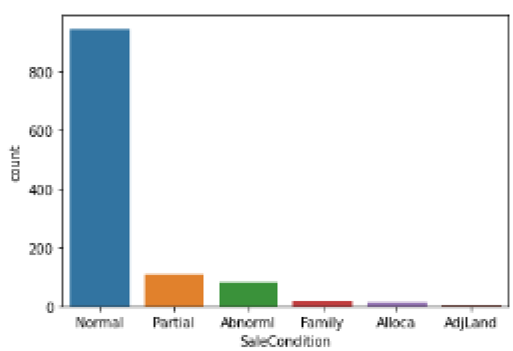
```
<AxesSubplot:xlabel='SaleType', ylabel='count'>
```




```
print(df_train['SaleCondition'].value_counts())
sns.countplot(df_train["SaleCondition"])
```

```
Normal     945
Partial    108
Abnorml     81
Family      18
Alloca      12
AdjLand      4
Name: SaleCondition, dtype: int64
```

```
<AxesSubplot:xlabel='SaleCondition', ylabel='count'>
```

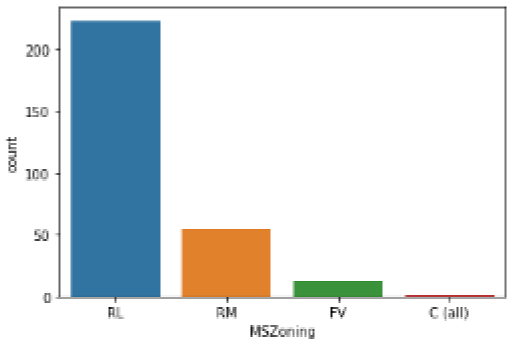


Countplots of testing data

```
print(df_test['MSZoning'].value_counts())
sns.countplot(df_test["MSZoning"])
```

```
RL          223
RM           55
FV           13
C (all)       1
Name: MSZoning, dtype: int64
```

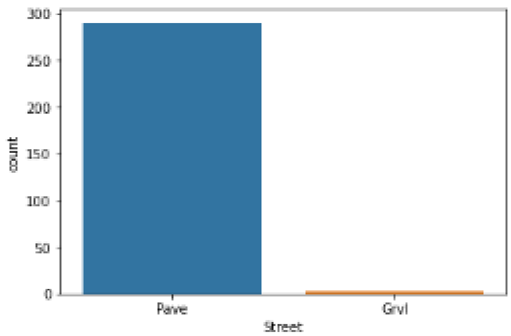
```
<AxesSubplot:xlabel='MSZoning', ylabel='count'>
```



```
print(df_test['Street'].value_counts())
sns.countplot(df_test["Street"])
```

```
Pave    290
Grvl      2
Name: Street, dtype: int64
```

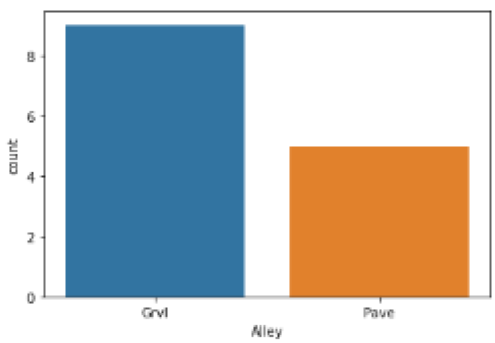
```
<AxesSubplot:xlabel='Street', ylabel='count'>
```




```
print(df_test['Alley'].value_counts())
sns.countplot(df_test["Alley"])
```

```
Grvl    9
Pave    5
Name: Alley, dtype: int64
```

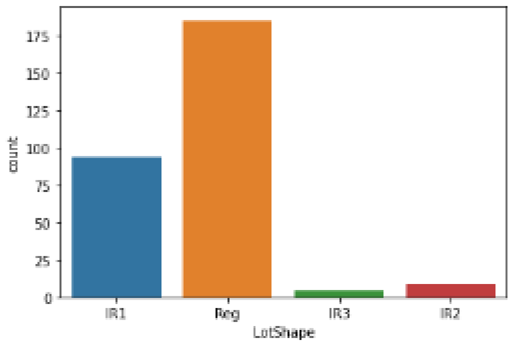
```
<AxesSubplot:xlabel='Alley', ylabel='count'>
```



```
print(df_test['LotShape'].value_counts())
sns.countplot(df_test["LotShape"])
```

```
Reg    185
IR1     94
IR2      9
IR3      4
Name: LotShape, dtype: int64
```

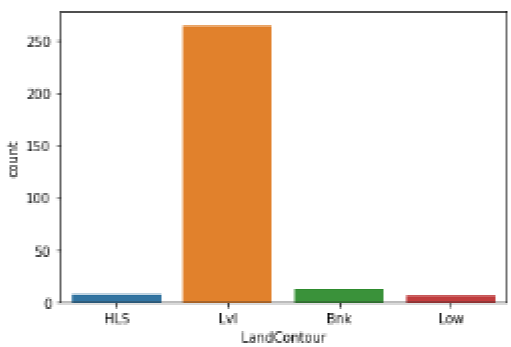
```
<AxesSubplot:xlabel='LotShape', ylabel='count'>
```




```
print(df_test['LandContour'].value_counts())
sns.countplot(df_test["LandContour"])
```

```
Lvl    265
Bnk     13
HLS      8
Low      6
Name: LandContour, dtype: int64
```

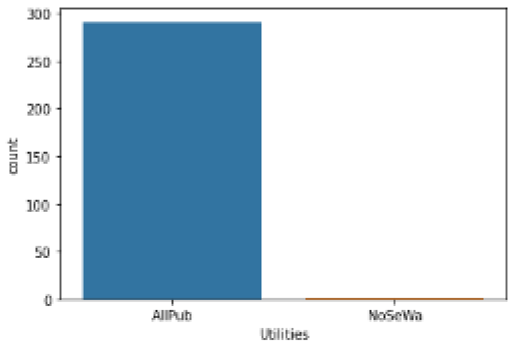
```
<AxesSubplot:xlabel='LandContour', ylabel='count'>
```



```
print(df_test['Utilities'].value_counts())
sns.countplot(df_test["Utilities"])
```

```
AllPub    291
NoSeWa      1
Name: Utilities, dtype: int64
```

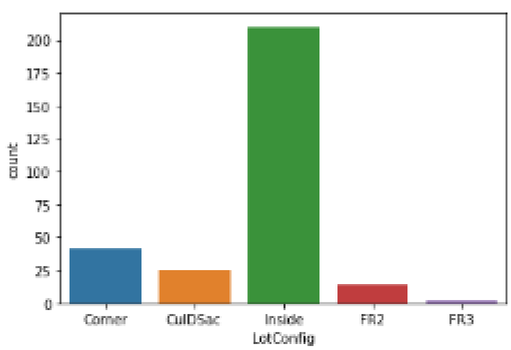
<AxesSubplot:xlabel='Utilities', ylabel='count'>




```
print(df_test['LotConfig'].value_counts())
sns.countplot(df_test["LotConfig"])
```

```
Inside     210
Corner      41
CulDSac     25
FR2         14
FR3          2
Name: LotConfig, dtype: int64
```

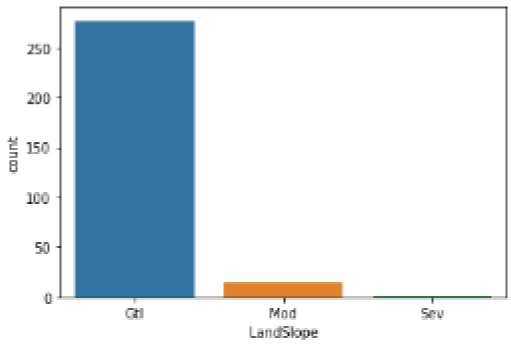
<AxesSubplot:xlabel='LotConfig', ylabel='count'>



```python
print(df_test['LandSlope'].value_counts())
sns.countplot(df_test["LandSlope"])
```

```
Gtl    277
Mod     14
Sev      1
Name: LandSlope, dtype: int64
```

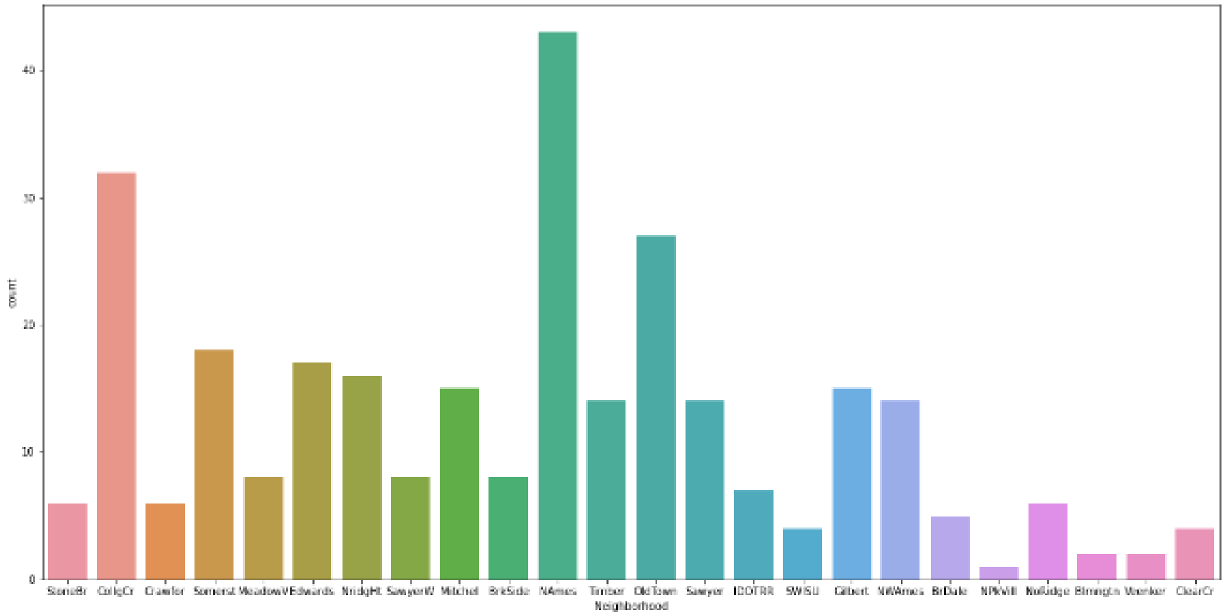
```
<AxesSubplot:xlabel='LandSlope', ylabel='count'>
```




```python
print(df_test['Neighborhood'].value_counts())
plt.figure(figsize=(20,10))
sns.countplot(df_test["Neighborhood"])
```

```
NAmes      43
CollgCr    32
OldTown    27
Somerst    18
Edwards    17
NridgHt    16
Gilbert    15
Mitchel    15
Sawyer     14
Timber     14
NWAmes     14
MeadowV     8
SawyerW     8
BrkSide     8
IDOTRR      7
StoneBr     6
NoRidge     6
Crawfor     6
BrDale      5
ClearCr     4
SWISU       4
Blmngtn     2
Veenker     2
NPkVill     1
Name: Neighborhood, dtype: int64
```

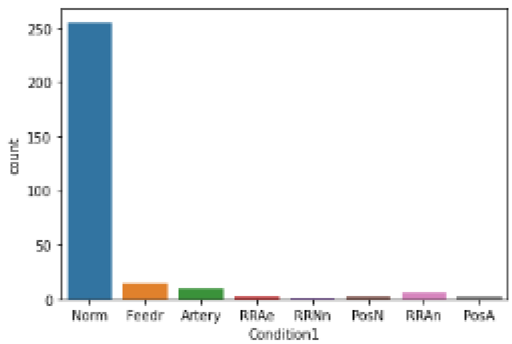
```
<AxesSubplot:xlabel='Neighborhood', ylabel='count'>
```



```
print(df_test['Condition1'].value_counts())
sns.countplot(df_test["Condition1"])
```

```
Norm      255
Feedr      14
Artery     10
RRAn        6
RRAe        2
PosN        2
PosA        2
RRNn        1
Name: Condition1, dtype: int64
```

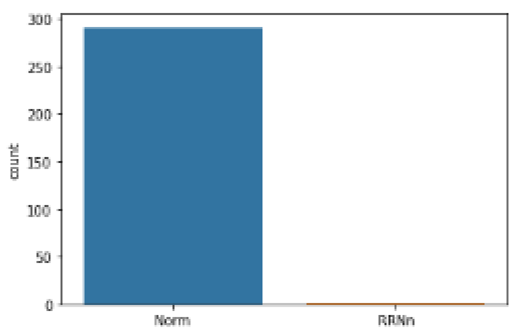
```
<AxesSubplot:xlabel='Condition1', ylabel='count'>
```




```
print(df_test['Condition2'].value_counts())
sns.countplot(df_test["Condition2"])
```

```
Norm    291
RRNn      1
Name: Condition2, dtype: int64
```

```
<AxesSubplot:xlabel='Condition2', ylabel='count'>
```

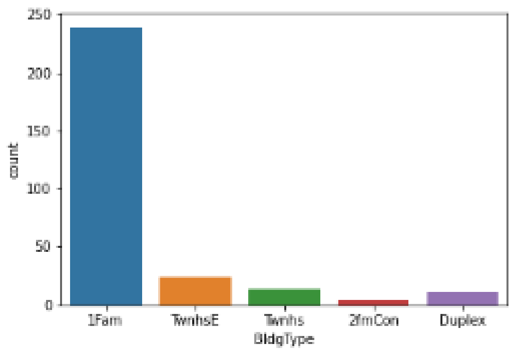


]:

```
print(df_test['BldgType'].value_counts())
sns.countplot(df_test["BldgType"])
```

```
1Fam       239
TwnhsE      24
Twnhs       14
Duplex      11
2fmCon       4
Name: BldgType, dtype: int64
```

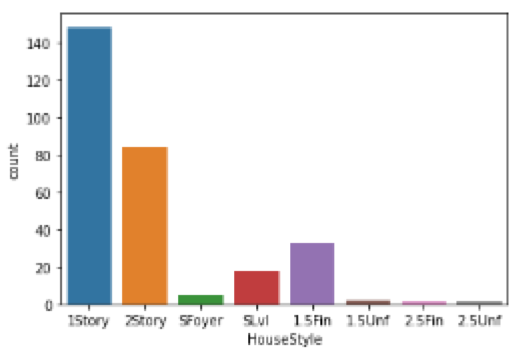
]: <AxesSubplot:xlabel='BldgType', ylabel='count'>




```
print(df_test['HouseStyle'].value_counts())
sns.countplot(df_test["HouseStyle"])
```

```
1Story    148
2Story     84
1.5Fin     33
SLvl       18
SFoyer      5
1.5Unf      2
2.5Unf      1
2.5Fin      1
Name: HouseStyle, dtype: int64
```

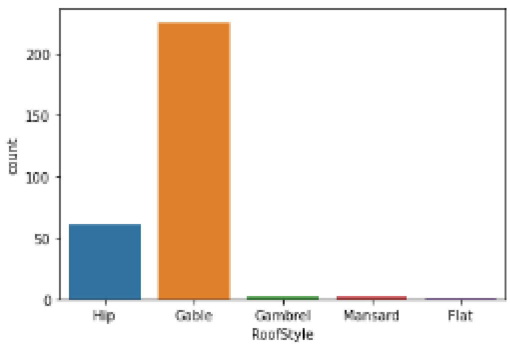
<AxesSubplot:xlabel='HouseStyle', ylabel='count'>



```python
print(df_test['RoofStyle'].value_counts())
sns.countplot(df_test["RoofStyle"])
```

```
Gable      226
Hip         61
Mansard      2
Gambrel      2
Flat         1
Name: RoofStyle, dtype: int64
```

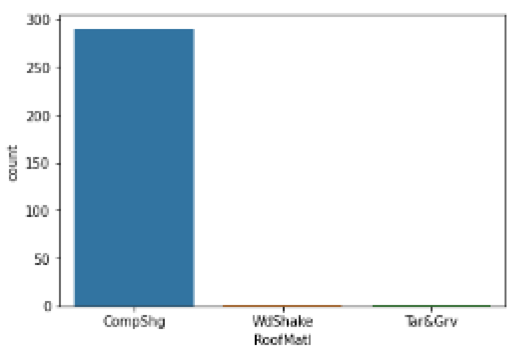
```
<AxesSubplot:xlabel='RoofStyle', ylabel='count'>
```




```python
print(df_test['RoofMatl'].value_counts())
sns.countplot(df_test["RoofMatl"])
```

```
CompShg    290
Tar&Grv      1
WdShake      1
Name: RoofMatl, dtype: int64
```

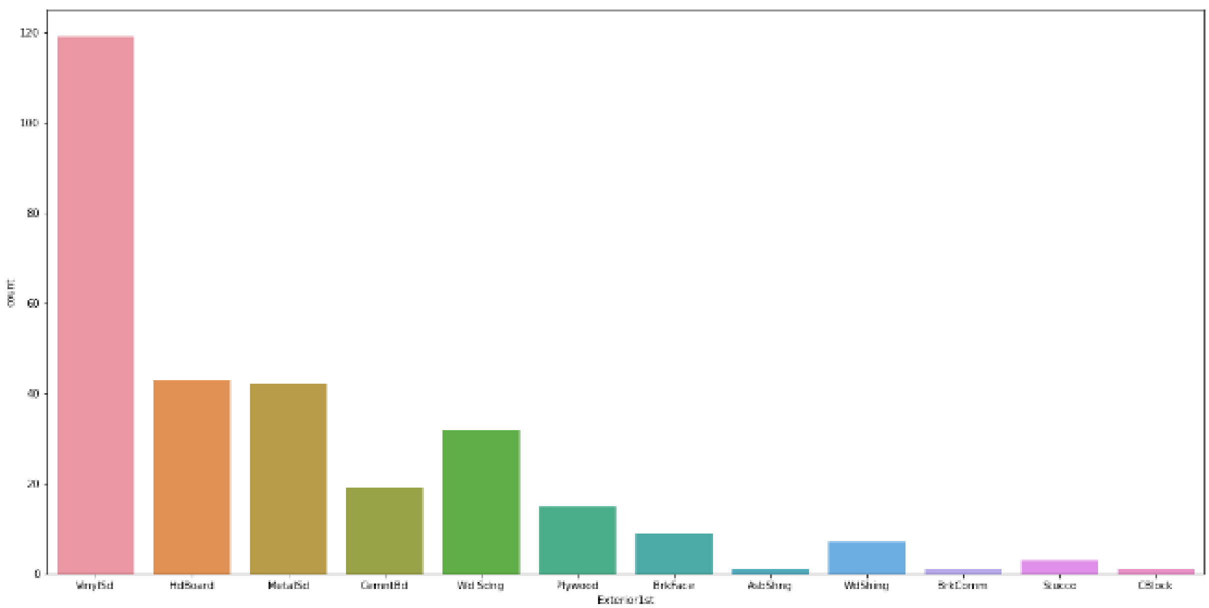
```
<AxesSubplot:xlabel='RoofMatl', ylabel='count'>
```



```python
print(df_test['Exterior1st'].value_counts())
plt.figure(figsize=(20,10))
sns.countplot(df_test["Exterior1st"])
```

```
VinylSd    119
HdBoard     43
MetalSd     42
Wd Sdng     32
CemntBd     19
Plywood     15
BrkFace      9
WdShing      7
Stucco       3
AsbShng      1
CBlock       1
BrkComm      1
Name: Exterior1st, dtype: int64
```

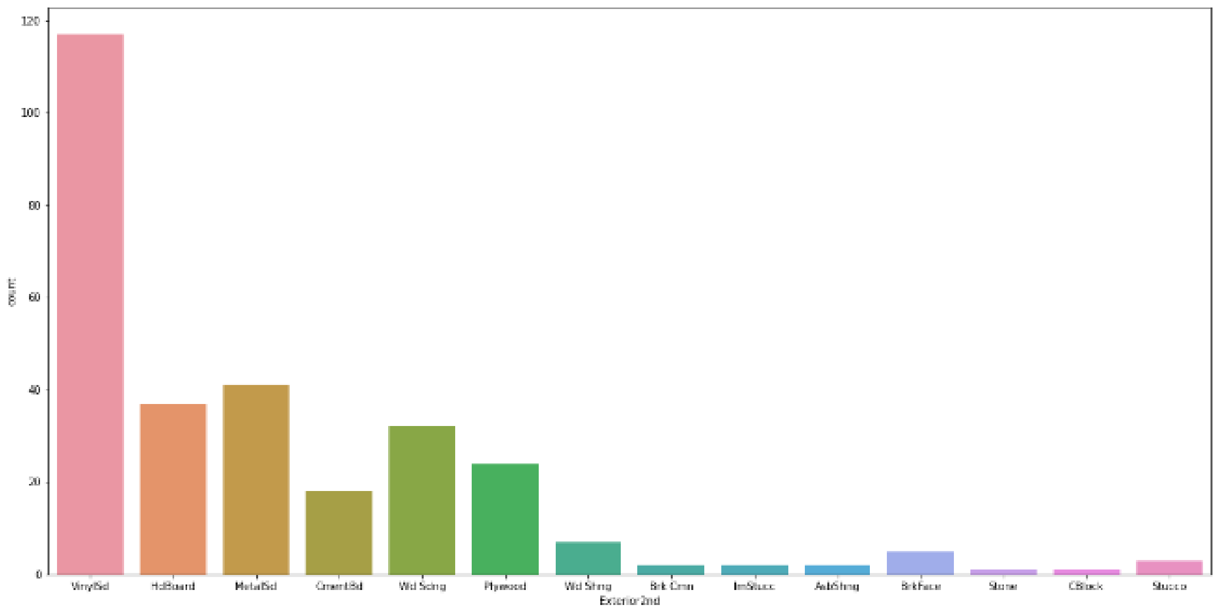
```
<AxesSubplot:xlabel='Exterior1st', ylabel='count'>
```



```
print(df_test['Exterior2nd'].value_counts())
plt.figure(figsize=(20,10))
sns.countplot(df_test["Exterior2nd"])
```

```
VinylSd    117
MetalSd     41
HdBoard     37
Wd Sdng     32
Plywood     24
CmentBd     18
Wd Shng      7
BrkFace      5
Stucco       3
Brk Cmn      2
AsbShng      2
ImStucc      2
Stone        1
CBlock       1
Name: Exterior2nd, dtype: int64
```

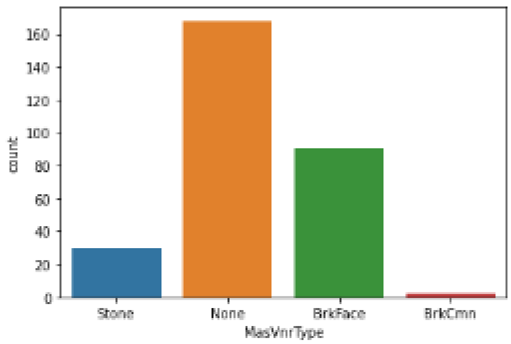
```
<AxesSubplot:xlabel='Exterior2nd', ylabel='count'>
```



```
print(df_test['MasVnrType'].value_counts())

sns.countplot(df_test["MasVnrType"])
```

```
None       168
BrkFace     91
Stone       30
BrkCmn       2
Name: MasVnrType, dtype: int64
```

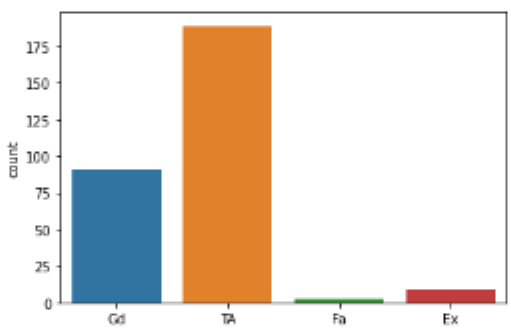
```
<AxesSubplot:xlabel='MasVnrType', ylabel='count'>
```




```
print(df_test['ExterQual'].value_counts())
sns.countplot(df_test["ExterQual"])
```

```
TA    189
Gd     91
Ex      9
Fa      3
Name: ExterQual, dtype: int64
```

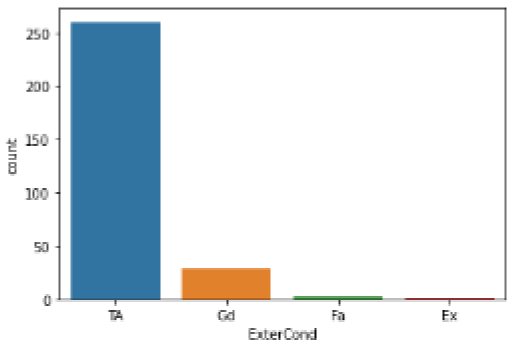
```
<AxesSubplot:xlabel='ExterQual', ylabel='count'>
```



```
2 sns.countplot(df_test["ExterCond"])
```

```
TA    260
Gd     29
Fa      2
Ex      1
Name: ExterCond, dtype: int64
```

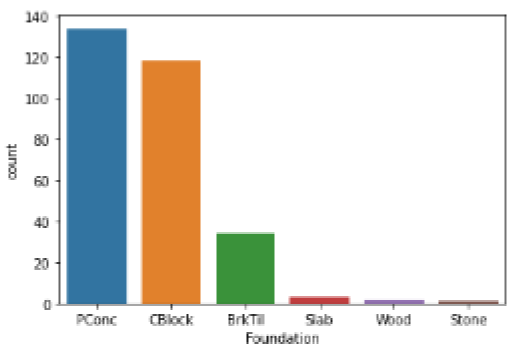
```
<AxesSubplot:xlabel='ExterCond', ylabel='count'>
```




```
1 print(df_test['Foundation'].value_counts())
2 sns.countplot(df_test["Foundation"])
```

```
PConc     134
CBlock    118
BrkTil     34
Slab        3
Wood        2
Stone       1
Name: Foundation, dtype: int64
```

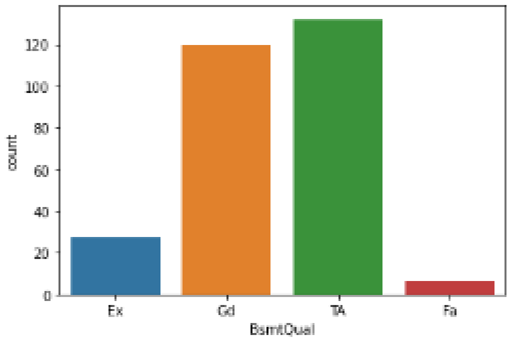
```
<AxesSubplot:xlabel='Foundation', ylabel='count'>
```



```
print(df_test['BsmtQual'].value_counts())
sns.countplot(df_test["BsmtQual"])
```

```
TA    132
Gd    120
Ex     27
Fa      6
Name: BsmtQual, dtype: int64
```

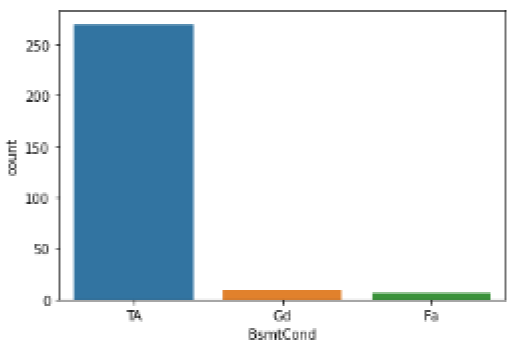
```
<AxesSubplot:xlabel='BsmtQual', ylabel='count'>
```




```
print(df_test['BsmtCond'].value_counts())
sns.countplot(df_test["BsmtCond"])
```

```
TA    270
Gd      9
Fa      6
Name: BsmtCond, dtype: int64
```

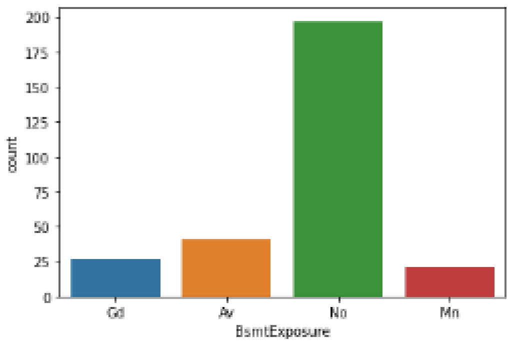
```
<AxesSubplot:xlabel='BsmtCond', ylabel='count'>
```



```
print(df_test['BsmtExposure'].value_counts())
sns.countplot(df_test["BsmtExposure"])
```

```
No    197
Av     41
Gd     26
Mn     21
Name: BsmtExposure, dtype: int64
```

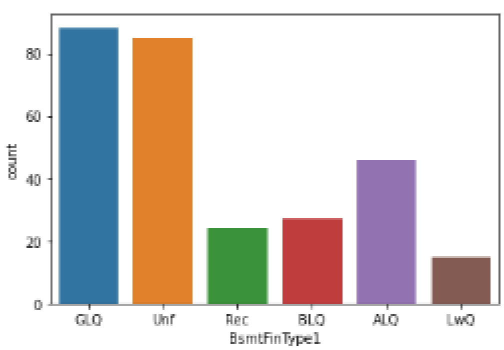
```
<AxesSubplot:xlabel='BsmtExposure', ylabel='count'>
```




```
print(df_test['BsmtFinType1'].value_counts())
sns.countplot(df_test["BsmtFinType1"])
```

```
GLQ    88
Unf    85
ALQ    46
BLQ    27
Rec    24
LwQ    15
Name: BsmtFinType1, dtype: int64
```

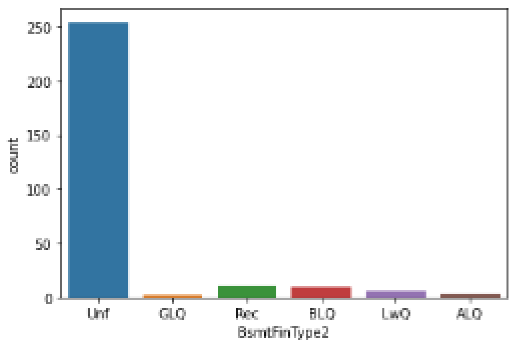
```
<AxesSubplot:xlabel='BsmtFinType1', ylabel='count'>
```



]:
```
print(df_test['BsmtFinType2'].value_counts())
sns.countplot(df_test["BsmtFinType2"])
```

```
Unf    254
Rec     11
BLQ      9
LwQ      6
ALQ      3
GLQ      2
Name: BsmtFinType2, dtype: int64
```

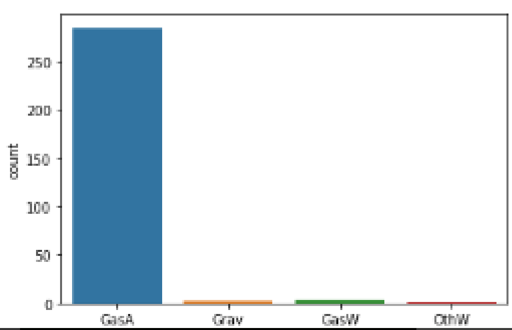
]: <AxesSubplot:xlabel='BsmtFinType2', ylabel='count'>



]:
```
print(df_test['Heating'].value_counts())
sns.countplot(df_test["Heating"])
```

```
GasA    285
GasW      4
Grav      2
OthW      1
Name: Heating, dtype: int64
```

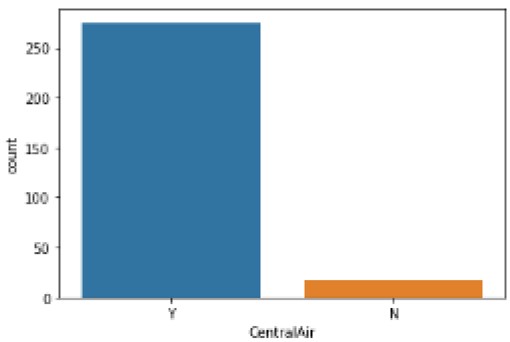
]: <AxesSubplot:xlabel='Heating', ylabel='count'>



```
print(df_test['CentralAir'].value_counts())
sns.countplot(df_test["CentralAir"])
```

```
Y    275
N     17
Name: CentralAir, dtype: int64
```

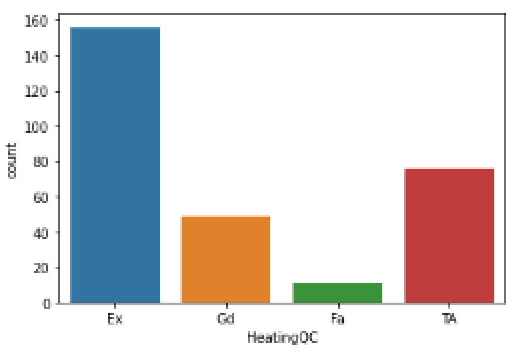
<AxesSubplot:xlabel='CentralAir', ylabel='count'>




```
print(df_test['HeatingQC'].value_counts())
sns.countplot(df_test["HeatingQC"])
```

```
Ex    156
TA     76
Gd     49
Fa     11
Name: HeatingQC, dtype: int64
```

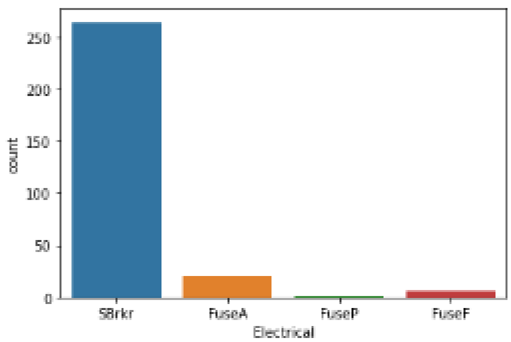
<AxesSubplot:xlabel='HeatingQC', ylabel='count'>



```
print(df_test['Electrical'].value_counts())
sns.countplot(df_test["Electrical"])
```

```
SBrkr    264
FuseA     20
FuseF      6
FuseP      1
Name: Electrical, dtype: int64
```

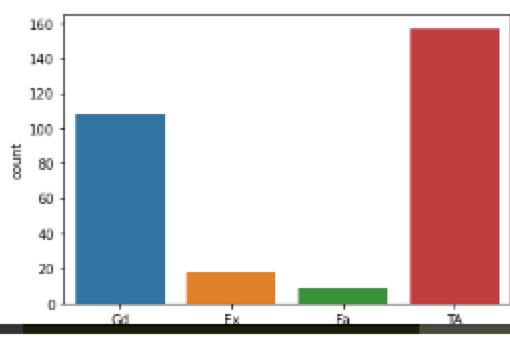
```
<AxesSubplot:xlabel='Electrical', ylabel='count'>
```




```
print(df_test['KitchenQual'].value_counts())
sns.countplot(df_test["KitchenQual"])
```

```
TA    157
Gd    108
Ex     18
Fa      9
Name: KitchenQual, dtype: int64
```

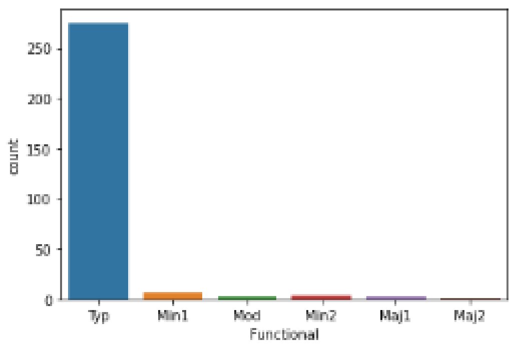
```
<AxesSubplot:xlabel='KitchenQual', ylabel='count'>
```



]:
```
1 print(df_test['Functional'].value_counts())
2 sns.countplot(df_test["Functional"])
```

```
Typ     275
Min1      6
Min2      4
Maj1      3
Mod       3
Maj2      1
Name: Functional, dtype: int64
```

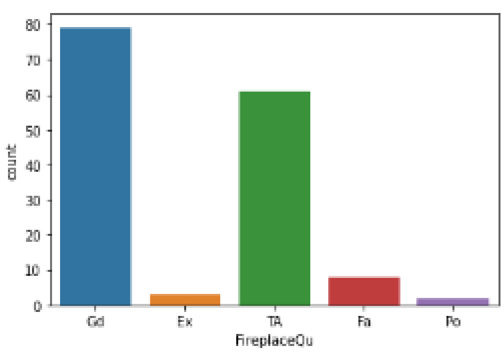
]: <AxesSubplot:xlabel='Functional', ylabel='count'>



:
```
1 print(df_test['FireplaceQu'].value_counts())
2 sns.countplot(df_test["FireplaceQu"])
```

```
Gd    79
TA    61
Fa     8
Ex     3
Po     2
Name: FireplaceQu, dtype: int64
```

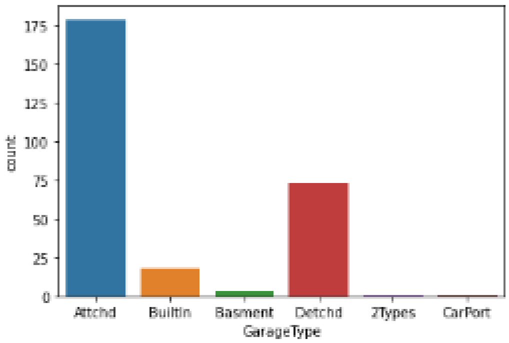
: <AxesSubplot:xlabel='FireplaceQu', ylabel='count'>



```
]:  1 print(df_test['GarageType'].value_counts())
    2 sns.countplot(df_test["GarageType"])
```

```
Attchd     179
Detchd      73
BuiltIn     18
Basment      3
CarPort      1
2Types       1
Name: GarageType, dtype: int64
```

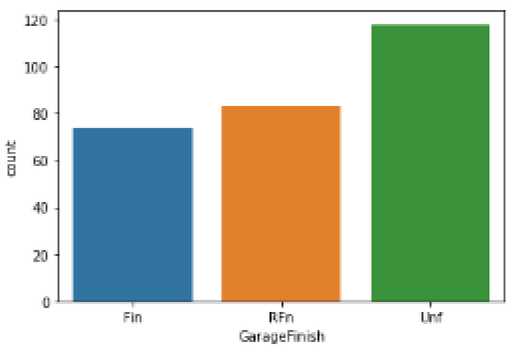
```
]: <AxesSubplot:xlabel='GarageType', ylabel='count'>
```




```
]:  1 print(df_test['GarageFinish'].value_counts())
    2 sns.countplot(df_test["GarageFinish"])
```

```
Unf    118
RFn     83
Fin     74
Name: GarageFinish, dtype: int64
```

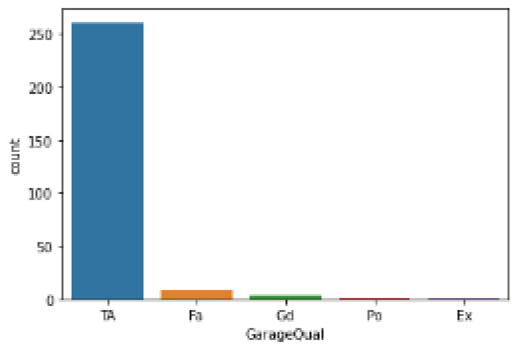
```
]: <AxesSubplot:xlabel='GarageFinish', ylabel='count'>
```



```python
print(df_test['GarageQual'].value_counts())
sns.countplot(df_test["GarageQual"])
```

```
TA    261
Fa      9
Gd      3
Po      1
Ex      1
Name: GarageQual, dtype: int64
```

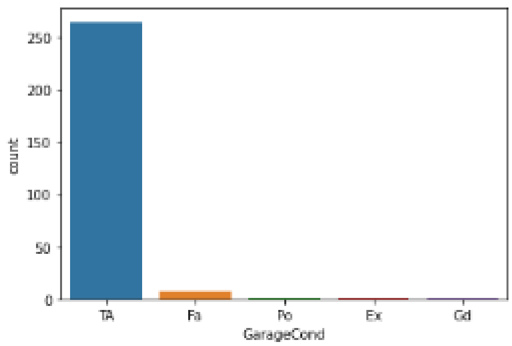
```
<AxesSubplot:xlabel='GarageQual', ylabel='count'>
```



```
print(df_test['GarageCond'].value_counts())
sns.countplot(df_test["GarageCond"])
```

```
TA    265
Fa      7
Po      1
Gd      1
Ex      1
Name: GarageCond, dtype: int64
```

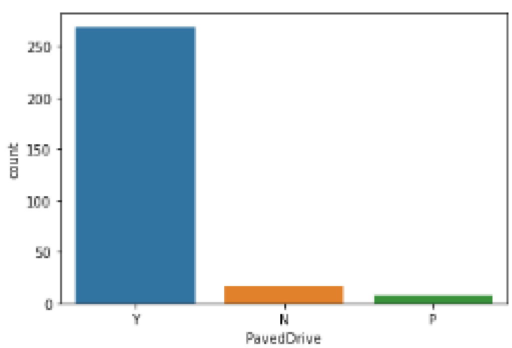
```
<AxesSubplot:xlabel='GarageCond', ylabel='count'>
```




```
print(df_test['PavedDrive'].value_counts())
sns.countplot(df_test["PavedDrive"])
```

```
Y    269
N     16
P      7
Name: PavedDrive, dtype: int64
```

```
<AxesSubplot:xlabel='PavedDrive', ylabel='count'>
```



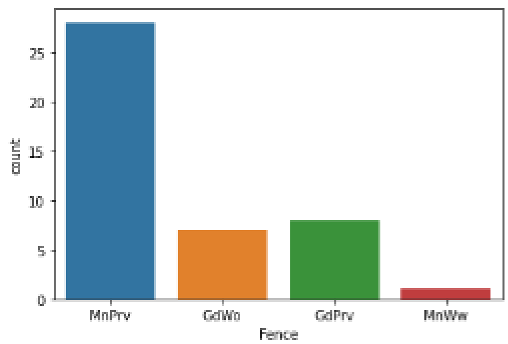
```
print(df_test['PoolQC'].value_counts())

```

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

```
print(df_test['Fence'].value_counts())
sns.countplot(df_test["Fence"])
```

```
MnPrv    28
GdPrv     8
GdWo      7
MnWw      1
Name: Fence, dtype: int64
```

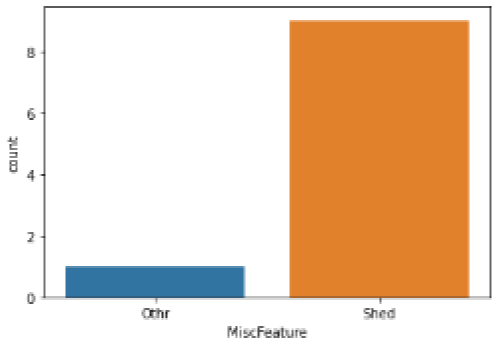
```
<AxesSubplot:xlabel='Fence', ylabel='count'>
```



```
print(df_test['MiscFeature'].value_counts())
sns.countplot(df_test["MiscFeature"])
```

```
Shed     9
Othr     1
Name: MiscFeature, dtype: int64
```

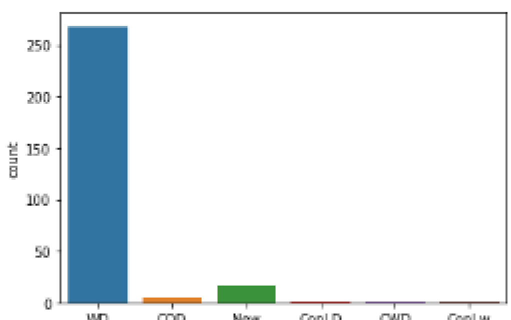
```
<AxesSubplot:xlabel='MiscFeature', ylabel='count'>
```




```
print(df_test['SaleType'].value_counts())
sns.countplot(df_test["SaleType"])
```

```
WD        268
New        16
COD         5
ConLw       1
ConLD       1
CWD         1
Name: SaleType, dtype: int64
```

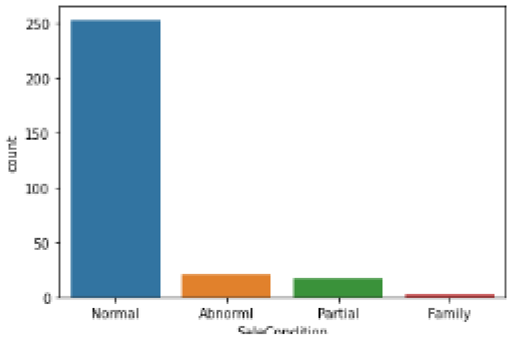
```
<AxesSubplot:xlabel='SaleType', ylabel='count'>
```



```
print(df_test['SaleCondition'].value_counts())
sns.countplot(df_test["SaleCondition"])
```

```
Normal      253
Abnorml      20
Partial      17
Family        2
Name: SaleCondition, dtype: int64
```

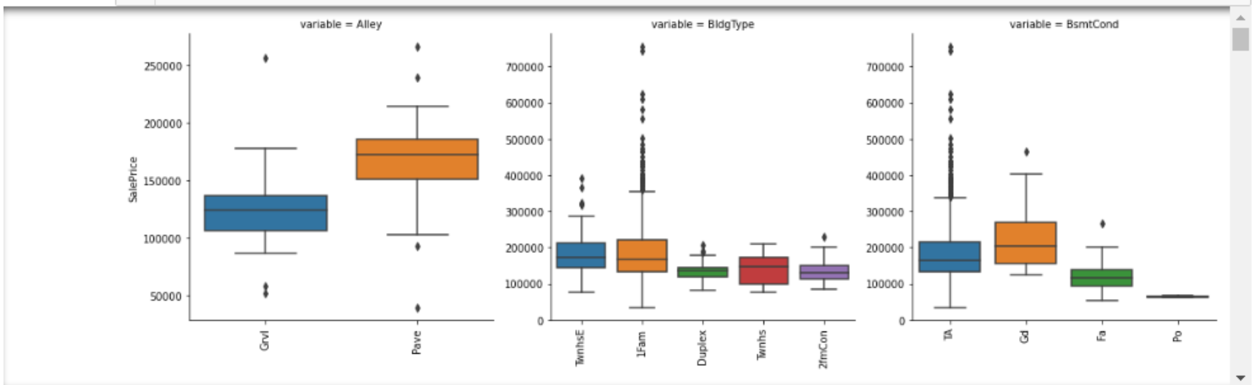
```
<AxesSubplot:xlabel='SaleCondition', ylabel='count'>
```



Boxplot: Boxplot detect outliers.

In [1849]:

```
def boxplot1(x, y, **kwargs):
    sns.boxplot(x=x, y=y)
    x=plt.xticks(rotation=90)

categorical = df_train.select_dtypes(exclude=['int64','float64'])
f = pd.melt(df_train, id_vars=['SalePrice'], value_vars=sorted(df_train[categorical.columns]))
g = sns.FacetGrid(f, col="variable", col_wrap=3, sharex=False, sharey=False, size=5)
g = g.map(facetgrid_boxplot, "value", "SalePrice")
```

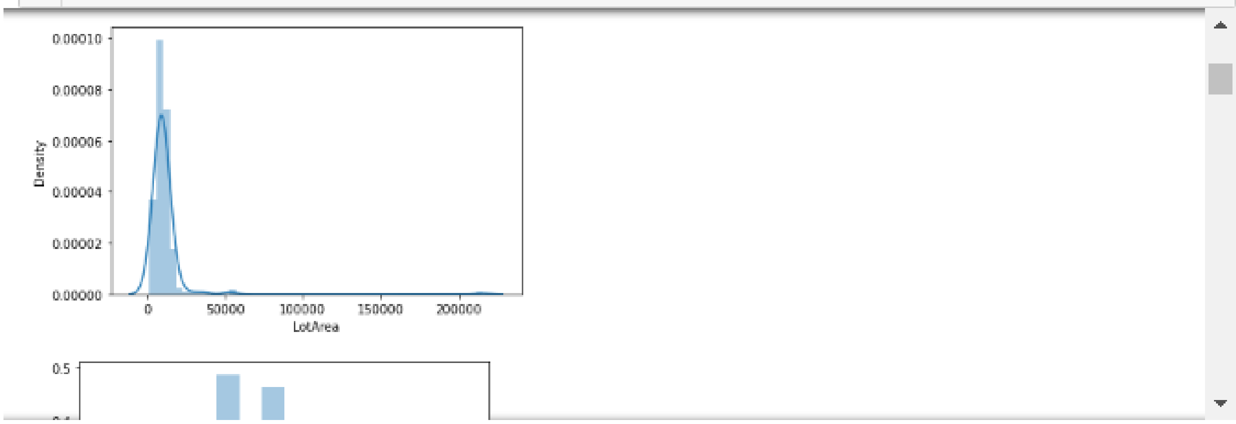


Outliers can be removed by zscore and IQR. In this data set, outliers are present, but this data set have categorical data so we can’t directly remove outlires by zscore.

Distribution plot:

```
for i in num_df.columns:
    sns.distplot(num_df[i])
    plt.figure()
```


```
for i in num_df2.columns:
    sns.distplot(num_df2[i])
    plt.figure()
```



Data Preparation:

Drop unnecessary columns:

```
df1.drop(['Alley','FireplaceQu','PoolQC','Fence','MiscFeature','MoSold'],axis=1,inplace=True)
```

```
df2.drop(['Alley','FireplaceQu','PoolQC','Fence','MiscFeature','MoSold'],axis=1,inplace=True)
```

Many input variables have many missing values ,so we can drop that columns.

```
df1.drop(['Id','Street','Utilities'],axis=1,inplace=True)
```

```
df2.drop(['Id','Street','Utilities'],axis=1,inplace=True)
```

Some input variables are less important, so we can drop that variables.

New variables:

Deriving some new variables from existing year variables.

```
df1['YearBuilt_Old'] = 2021-df1.YearBuilt
df1['YearRemodAdd_Old'] =2021-df1.YearRemodAdd
df1['GarageYrBlt_Old'] = 2021-df1.GarageYrBlt
df1['YrSold_Old'] = 2021-df1.YrSold
df1[['YearBuilt','YearRemodAdd','GarageYrBlt','YrSold','YearBuilt_Old','YearRemodAdd_Old',
            'GarageYrBlt_Old','YrSold_Old']].head(10)
```

	YearBuilt	YearRemodAdd	GarageYrBlt	YrSold	YearBuilt_Old	YearRemodAdd_Old	GarageYrBlt_Old	YrSold_Old
0	1976	1976	1977.0	2007	45	45	44.0	14
1	1970	1970	1970.0	2007	51	51	51.0	14
2	1996	1997	1997.0	2007	25	24	24.0	14
3	1977	1977	1977.0	2010	44	44	44.0	11
4	1977	2000	1977.0	2009	44	21	44.0	12
5	2006	2006	2006.0	2006	15	15	15.0	15
6	1957	1996	1957.0	2010	64	25	64.0	11
7	1957	2000	1957.0	2006	64	21	64.0	15
8	1965	1965	1965.0	2007	56	56	56.0	14
9	1947	1950	1947.0	2008	74	71	74.0	13

Doing same for testing data:

```python
df2['YearBuilt_Old'] = 2021-df2.YearBuilt
df2['YearRemodAdd_Old'] =2021-df2.YearRemodAdd
df2['GarageYrBlt_Old'] = 2021-df2.GarageYrBlt
df2['YrSold_Old'] = 2021-df2.YrSold
df2[['YearBuilt','YearRemodAdd','GarageYrBlt','YrSold','YearBuilt_Old','YearRemodAdd_Old',
              'GarageYrBlt_Old','YrSold_Old']].head(10)
```

	YearBuilt	YearRemodAdd	GarageYrBlt	YrSold	YearBuilt_Old	YearRemodAdd_Old	GarageYrBlt_Old	YrSold_Old
0	2005	2006	2005.0	2007	16	15	16.0	14
1	1984	1984	1984.0	2009	37	37	37.0	12
2	2001	2001	2001.0	2009	20	20	20.0	12
3	1941	1950	1941.0	2009	80	71	80.0	12
4	2007	2007	2007.0	2008	14	14	14.0	13
5	1970	1970	NaN	2007	51	51	NaN	14
6	2005	2005	2005.0	2006	16	16	16.0	15
7	2007	2008	2007.0	2008	14	13	14.0	13
8	1989	1989	1989.0	2009	32	32	32.0	12
9	1998	1998	1998.0	2009	23	23	23.0	12

Filling missing values:

In both training data and testing data have missing values, so we have to treat that values.

```python
df1['GarageYrBlt_Old']=df1['GarageYrBlt_Old'].fillna(0)
df1['MasVnrType']=df1['MasVnrType'].fillna(df1['MasVnrType'].mode()[0])
df1['MasVnrArea']=df1['MasVnrArea'].fillna(df1['MasVnrArea'].mean())
df1['BsmtQual']=df1['BsmtQual'].fillna(df1['BsmtQual'].mode()[0])
df1['BsmtCond']=df1['BsmtCond'].fillna(df1['BsmtCond'].mode()[0])
df1['BsmtExposure']=df1['BsmtExposure'].fillna(df1['BsmtExposure'].mode()[0])
df1['BsmtFinType1']=df1['BsmtFinType1'].fillna(df1['BsmtFinType1'].mode()[0])
df1['BsmtFinType2']=df1['BsmtFinType2'].fillna(df1['BsmtFinType2'].mode()[0])
df1['GarageType']=df1['GarageType'].fillna(df1['GarageType'].mode()[0])

df1['GarageFinish']=df1['GarageFinish'].fillna(df1['GarageFinish'].mode()[0])
df1['GarageQual']=df1['GarageQual'].fillna(df1['GarageQual'].mode()[0])
df1['GarageCond']=df1['GarageCond'].fillna(df1['GarageCond'].mode()[0])
```

Categorical missing values are replaced by their mode value.

Numerical missing values are replaced by mean/median value.

This is same for test data:

```
df2['MasVnrType']=df2['MasVnrType'].fillna(df2['MasVnrType'].mode()[0])
df2['MasVnrArea']=df2['MasVnrArea'].fillna(df2['MasVnrArea'].median())
df2['BsmtQual']=df2['BsmtQual'].fillna(df2['BsmtQual'].mode()[0])
df2['BsmtCond']=df2['BsmtCond'].fillna(df2['BsmtCond'].mode()[0])
df2['BsmtExposure']=df2['BsmtExposure'].fillna(df2['BsmtExposure'].mode()[0])
df2['BsmtFinType1']=df2['BsmtFinType1'].fillna(df2['BsmtFinType1'].mode()[0])
df2['BsmtFinType2']=df2['BsmtFinType2'].fillna(df2['BsmtFinType2'].mode()[0])
```

```
df2['GarageYrBlt_Old']=df2['GarageYrBlt_Old'].fillna(0)
df2['GarageType']=df2['GarageType'].fillna(df2['GarageType'].mode()[0])

df2['GarageFinish']=df2['GarageFinish'].fillna(df2['GarageFinish'].mode()[0])
df2['GarageQual']=df2['GarageQual'].fillna(df2['GarageQual'].mode()[0])
df2['GarageCond']=df2['GarageCond'].fillna(df2['GarageCond'].mode()[0])
```

```
df2['Electrical']=df2['Electrical'].fillna(df2['Electrical'].mode()[0])
```

Encoding:

Encoding: First we have to encode the categorical data into numerical data. There are different techniques of encoding:

- One Hot Encoder: Encode categorical integer features using a onehot aka one-of-K scheme. The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete) features. The output will be a sparse matrix where each column corresponds to one possible value of one feature.
- Label Encoder: Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important preprocessing step for the structured dataset in supervised learning.
- OrdinalEncoder: In ordinal encoding, each unique category value is assigned an integer value. For example, “red” is 1, “green” is 2, and “blue” is 3. This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.

In this project Label Encoder is used to encode the categorical data into numerical data.

```python
cat_col=['LandSlope','ExterQual','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2',
         'HeatingQC','CentralAir',  'KitchenQual','GarageFinish','GarageQual','GarageCond',
          'ExterCond','LotShape']
```

```python
from sklearn.preprocessing import LabelEncoder
```

```python
le=LabelEncoder()
for i in df1[cat_col]:
    df1[i]=le.fit_transform(df1[i])
```

```python
df1[['LandSlope','ExterQual','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2',
         'HeatingQC','CentralAir',  'KitchenQual','GarageFinish','GarageQual','GarageCond',
          'ExterCond','LotShape']].head()
```

	LandSlope	ExterQual	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinType2	HeatingQC	CentralAir	KitchenQual	GarageFinish	GarageQual
0	0	3	2	3	3	0	5	4	1	3	1	4
1	1	2	3	1	1	0	4	0	1	2	2	4
2	0	2	2	3	0	2	5	0	1	3	2	4
3	0	3	2	3	3	1	5	0	1	3	1	4
4	0	2	2	3	3	0	5	2	1	2	0	4

```python
cat_col2=['MSZoning','LandContour','LotConfig','Neighborhood','Condition1','Condition2','BldgType',
             'HouseStyle','RoofStyle','RoofMatl','Exterior1st',  'Exterior2nd','MasVnrType','Foundation',
             'Heating','Electrical','Functional','GarageType','PavedDrive','SaleType','SaleCondition']
```

```python
for i in df1[cat_col2]:
    df1[i]=le.fit_transform(df1[i])
```

```python
for i in df2[cat_col2]:
    df2[i]=le.fit_transform(df2[i])
```

Above fig shows that categorical columns of training data set are encoded using label encoder. And same is done for testing data set.

```python
le=LabelEncoder()
for i in df2[cat_col]:
    df2[i]=le.fit_transform(df2[i])
```

```python
df2[['LandSlope','ExterQual','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2',
         'HeatingQC','CentralAir',  'KitchenQual','GarageFinish','GarageQual','GarageCond',
          'ExterCond','LotShape']].head()
```

	LandSlope	ExterQual	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinType2	HeatingQC	CentralAir	KitchenQual	GarageFinish	GarageQual
0	0	2	0	2	1	2	5	0	1	2	0	4
1	0	2	2	2	0	2	5	2	1	2	1	4
2	0	2	2	2	0	5	5	0	1	0	1	4
3	0	3	3	2	3	4	5	0	1	1	2	4
4	0	2	2	2	2	5	5	0	1	2	0	4

```
for i in df2[cat_col2]:
    df2[i]=le.fit_transform(df2[i])
```

```
print(df1.shape)
print(df2.shape)
```

```
(1168, 75)
(292, 74)
```

Standard Scaler:

Standardizing a dataset involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1.

This can be thought of as subtracting the mean value or centering the data.

Like normalization, standardization can be useful, and even required in some machine learning algorithms where data has input values with differing scales.

```
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
```

```
num_col = ['MSSubClass','LotArea','OverallQual','OverallCond',
           'MasVnrArea','BsmtFinSF1',
           'BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','1stFlrSF','2ndFlrSF',
           'LowQualFinSF','GrLivArea','BsmtFullBath','BsmtHalfBath','FullBath','HalfBath','BedroomAbvGr',
           'KitchenAbvGr','TotRmsAbvGrd','Fireplaces','GarageCars',
           'GarageArea','WoodDeckSF','OpenPorchSF','EnclosedPorch','3SsnPorch',
           'ScreenPorch','PoolArea','MiscVal','LotFrontage']
df1[num_col]=st.fit_transform(df1[num_col])
```

```
df1.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	...	ScreenPorch	PoolArea	Mi
0	1.508301	3	0.000000	-0.620616	0	3	4	0	13	2	...	-0.273377	-0.076845	-0.08
1	-0.877042	3	1.070631	0.600903	0	3	4	1	12	2	...	3.795117	-0.076845	-0.08
2	0.077095	3	0.936867	-0.063075	0	3	1	0	15	2	...	-0.273377	-0.076845	-0.08
3	-0.877042	3	1.516514	0.141424	0	3	4	0	14	2	...	-0.273377	-0.076845	-0.08
4	-0.877042	3	0.000000	0.686902	0	3	2	0	14	2	...	-0.273377	-0.076845	-0.08

5 rows × 72 columns

Above fig shows how variables of training data set standardized using standard scaler.

Following fig shows same for testing data set.

```
df2[num_col]=st.fit_transform(df2[num_col])
```

```
df2.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	...	3SsnPorch	ScreenPorch	Po
0	-0.856054	2	0.981605	0.263894	0	1	0	0	21	2	...	-0.082453	-0.258624	
1	1.431981	2	0.000000	-0.363030	0	3	1	0	21	2	...	-0.082453	-0.258624	
2	-0.856054	2	0.000000	0.089636	3	3	4	0	4	2	...	-0.082453	-0.258624	
3	0.287963	2	0.429998	0.101809	3	0	4	0	5	2	...	-0.082453	-0.258624	
4	0.059160	2	0.981605	0.297033	0	3	1	0	20	1	...	-0.082453	-0.258624	

5 rows × 71 columns

Separating dependent variable from independent variables:

```
x=df1.drop('SalePrice',axis=1)
```

```
y=df1['SalePrice']
```

Algorithms used for training and testing data:

As this is regression problem we have to use regression algorithms.

Following fig shows importing algorithms and creating instances of that.

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,mean_absolute_error


```

```
lr=LinearRegression()
r=Ridge(alpha=0.001)
l=Lasso(alpha=0.001)
knn=KNeighborsRegressor()
dtr=DecisionTreeRegressor()
svr=SVR()
rfr=RandomForestRegressor()

```

```
list1=[lr,l,r,knn,dtr,svr,rfr]

```

Split data for training and testing using train_test_split.

```
print("x_train shape=",x_train.shape)
print("x_test shape=",x_test.shape)
print("y_train shape=",y_train.shape)
print("y_test shape=",y_test.shape)
```

```
x_train shape= (817, 71)
x_test shape= (351, 71)
y_train shape= (817,)
y_test shape= (351,)
```

Fit and predict data:

Following fig shows fitting and prediction of data:

```
#fitting data into model and predictiong values
for i in list1:
    i.fit(x_train,y_train)
    pred=i.predict(x_test)
    print("accuracy of ",i)
    print("r_2 score=",r2_score(pred,y_test))
    print("mean_squared_error=",mean_squared_error(pred,y_test))
    print("mean_absolute_error=",mean_absolute_error(pred,y_test))
    print("score",i.score(x_train,y_train))
```

```
accuracy of  LinearRegression()
r_2 score= 0.8304783887054258
mean_squared_error= 805719955.3229054
mean_absolute_error= 19910.359930090814
score 0.821167011563678
accuracy of  Lasso(alpha=0.001)
r_2 score= 0.8304645433142916
mean_squared_error= 805814692.4558097
mean_absolute_error= 19911.335419873485
score 0.821148590676094
accuracy of  Ridge(alpha=0.001)
r_2 score= 0.830464677134791З
mean_squared_error= 805813642.8739568
mean_absolute_error= 19911.320184306605
score 0.821148590666195
accuracy of  KNeighborsRegressor()
r_2 score= 0.4491683222367312
mean_squared_error= 1858926882.9432478
mean_absolute_error= 29634.116809116807
score 0.7844955548283319
accuracy of  DecisionTreeRegressor()
r_2 score= 0.7585666144552273
mean_squared_error= 1197587018.1111112
mean_absolute_error= 23597.558404558404
score 1.0
accuracy of  SVR()
r_2 score= -652414.0617322808
mean_squared_error= 5607067710.229562
mean_absolute_error= 53957.97161037368
score -0.050654481400336904
accuracy of  RandomForestRegressor()
r_2 score= 0.8291209027854278
mean_squared_error= 808662388.5685732
mean_absolute_error= 18376.706552706553
score 0.9770749100557646
```

Above fig shows that Ridge and lasso algorithms gives more accuracy than others.

Hyparameter Tunning using GridsearchCV:

1. First performed hyparameter tunning for lasso:

```
from sklearn.model_selection import GridSearchCV
```

```
#finding out best params by gridsearch cv
from sklearn.model_selection import GridSearchCV
alpha_val = {'alpha':[0.00001, 0.0001, 0.0002, 0.001,0.002,0.01,0.02] }

lasso1 = Lasso()

grid = GridSearchCV(estimator = lasso1, param_grid = alpha_val, scoring= 'r2', cv = 5, return_train_score=True, verbose = 1)
grid.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    8.5s finished
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [1e-05, 0.0001, 0.0002, 0.001, 0.002, 0.01,
                                   0.02]},
             return_train_score=True, scoring='r2', verbose=1)
```

```
grid.best_params_
```

```
{'alpha': 0.02}
```

```
lasso1=Lasso(alpha=0.02)
lasso1.fit(x_train,y_train)
pred=lasso1.predict(x_test)
print("accuracy of lasso")
print("r_2 score=",r2_score(pred,y_test))
print("mean_squared_error=",mean_squared_error(pred,y_test))
print("mean_absolute_error=",mean_absolute_error(pred,y_test))
print("score",lasso1.score(x_train,y_train))
```

```
accuracy of lasso
r_2 score= 0.8304646854943607
mean_squared_error= 805812592.8340411
mean_absolute_error= 19911.304362537583
score 0.8211148590579398
```

Above fig shows that best alpha value for lasso is 0.02.

2.HyperParameter Tunning for Ridge:

```
from sklearn.model_selection import GridSearchCV
alpha_val = {'alpha':[0.00001,  0.0001, 0.0002,0.001,0.002,0.01,0.02 ]}

ridge1 = Ridge()

grid = GridSearchCV(estimator = ridge1, param_grid = alpha_val, scoring= 'r2', cv = 5, return_train_score=True, verbose = 1)
grid.fit(x_train,y_train)

```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    1.7s finished
```

```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [1e-05, 0.0001, 0.0002, 0.001, 0.002, 0.01,
                                   0.02]},
             return_train_score=True, scoring='r2', verbose=1)
```

```
grid.best_params_
```

```
{'alpha': 0.02}
```

```
r=Ridge(alpha=0.02)
r.fit(x_train,y_train)
pred=r.predict(x_test)
print("accuracy of Ridge")
print("r_2 score=",r2_score(pred,y_test))
print("mean_squared_error=",mean_squared_error(pred,y_test))
print("mean_absolute_error=",mean_absolute_error(pred,y_test))
```

```
accuracy of Ridge
r_2 score= 0.8304673611730087
mean_squared_error= 805791605.4397105
mean_absolute_error= 19910.999680730863
```

By observing scores and error values, we can concluded that Lasso with alpha value 0.02 is best module as compared to others.
So final model is builed using Lasso.

Creating lasso module and predict test data using this module.

```
|: 
   import joblib
```

```
|: #creating  object file
   joblib.dump(lasso1,"Housing_price1.obj")
```

```
|: ['Housing_price1.obj']
```

```
|: #loading object file
   file=joblib.load("Housing_price1.obj")
```

```
|: 3predict test data
   ds_pred=file.predict(df2)
```

```
|: ds_pred
```

```
|: array([343140.47922521, 227432.89760412, 278971.66239854, 160607.99239958,
         250052.34319597,  86872.67161852, 131421.62105616, 319445.82002293,
         244701.30992317, 194965.67360023,  73207.09533763, 165104.32547951,
         117671.87429011, 215290.84238772, 300683.58478554, 143874.66411977,
         119589.53312072, 131211.28870931, 218257.12654291, 241012.75822922,
         197011.05786392, 171851.65336682, 143705.11436775,  81701.66581762,
          99225.27936506, 146161.37896226, 188738.98496134, 138168.49516133,
         178768.73172616,  72003.20147073, 164106.64712932, 216495.63833233,
         254406.12476547, 198975.01269254, 117736.59150452, 187855.9078996 ,
         219003.38168414, 127148.12480562, 164339.6397339 , 155395.68717546,
         103685.45691162, 315686.65787888, 236367.08906155, 219755.78804831,
         145153.2125913 , 149495.59819039, 127646.93739332, 117570.20621553,
         224104.08772053, 366657.39860099, 140444.99985051, 228752.78248979,
         108881.01007153,  86488.25947593, 291521.81332363, 129375.25468119,
         166208.94492969, 213121.59327977,  98796.06346229, 258385.64821853,
         100710.94386842, 209367.51027906, 150280.06624019, 176069.3249049 ,
         231546.68288397,  79398.24701724, 151421.18786189, 252835.52031474,
         162866.76462578, 159191.08928283, 329396.78897849, 191741.85579322,
         178134.00052207, 170266.94190098, 174904.03967736, 258651.13348707,
         332252.27894033, 203253.77449235, 293338.10220503, 160033.92514254,
         262523.33441502, 154991.68205943, 144764.50287535, 169488.45153352,
         215993.6384133 , 257647.70714958,  93648.72526278, 395803.20073791,
         145053.06603709, 194096.91726527, 262621.00387556, 126571.32602254,
         132515.13680542, 122094.81865785, 210423.825621  , 196221.99568641,
         269448.91238379, 204089.07334575, 374897.13946143, 119285.38656494,
         259015.50091487, 124543.07146987, 146704.92390964, 165832.05442492,
```

Interpretation of the Results:

- Visualization: By plotting different plots we can gain more information about data
  - Boxplot:Using boxplot we can detect outliers.
  - Heatmap:Using heat map we can shows the correlation between different variables.
  - .Distribution plot: Using Distribution plot we can show the distribution of data..
  - .Countplot:Using this we can count values of particular columns.
- Preprocessing:Before sending data to model we have to do preprocessing on data so model can predict the best value.
- Modeling:After clean up of data set we can send data to fit and predict values to model using different algorithms. In this project we have used 5 classification algorithm:
  - LogisticRegression
  - SVR
  - LASSO
  - RIDGE
  - Decisiontreeregressor
  - KNeighboursRegressor
  - RandomForestRegressor
- Accuracy: Using metrics such as r2_score, mean_squared_error andmean_absolute_error, we can decide which model is best model. In this project lasso is the best model.

Conclusions:

We have defined several models with various features and various model complexities. There is a need to use a mix of these models a linear model gives a high bias (under fit) whereas a high model complexity-based model gives a high variance (overfit). Data Scientist tends to overfit their models which can be reduced by ridge regression

and LASSO The study reveals that economic factors influence land price more than the social factors. The interaction of the selected factors (X) on sale price (Y) is analyzed.

Following some variables have most impact on output variables.

BsmtHalfBath : Basement half bathrooms

LowQualFinSF : Low quality finished square feet (all floors)

BsmtFullBath : Basement full bathrooms

HalfBath : Half baths above grade