```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv('/content/train.csv')
data.head()
```

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

5 rows × 81 columns

```
data.columns
```

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

```
data.shape
```

```
(1460, 81)
```

```
df = data[['OverallQual', 'GrLivArea', 'TotalBsmtSF', '1stFlrSF',
    'FullBath', 'YearBuilt', 'YearRemodAdd', 'GarageCars', 'GarageArea',
    'Fireplaces', 'LotArea','ExterQual', 'KitchenQual', 'BsmtQual','SalePrice']]
df.head() #These Columns are helpful for house price prediction
```

	OverallQual	GrLivArea	TotalBsmtSF	1stFlrSF	FullBath	YearBuilt	YearRemodAdd	Gar
0	7	1710	856	856	2	2003	2003	
1	6	1262	1262	1262	2	1976	1976	
2	7	1786	920	920	2	2001	2002	
3	7	1717	756	961	1	1915	1970	
4	8	2198	1145	1145	2	2000	2000	

Next steps: Generate code with df View recommended plots

```
df.shape
```

```
(1460, 15)
```

```
df.columns
```

```
Index(['OverallQual', 'GrLivArea', 'TotalBsmtSF', '1stFlrSF', 'FullBath',
       'YearBuilt', 'YearRemodAdd', 'GarageCars', 'GarageArea', 'Fireplaces',
       'LotArea', 'ExterQual', 'KitchenQual', 'BsmtQual', 'SalePrice'],
      dtype='object')
```

```
df.isnull().sum()
```

```
OverallQual      0
GrLivArea        0
TotalBsmtSF      0
1stFlrSF         0
FullBath         0
YearBuilt        0
YearRemodAdd     0
GarageCars       0
GarageArea       0
Fireplaces       0
LotArea          0
ExterQual        0
KitchenQual      0
BsmtQual        37
SalePrice        0
dtype: int64
```

```
import warnings
warnings.filterwarnings('ignore')
df.dropna(inplace=True)
```

```
df.isnull().sum() #Now there is no missing values.
```

```
OverallQual     0
GrLivArea       0
TotalBsmtSF     0
1stFlrSF        0
FullBath        0
YearBuilt       0
YearRemodAdd    0
GarageCars      0
GarageArea      0
Fireplaces      0
LotArea         0
ExterQual       0
KitchenQual     0
BsmtQual        0
SalePrice       0
dtype: int64
```

```
df.duplicated().sum() #No Duplicates are there
```

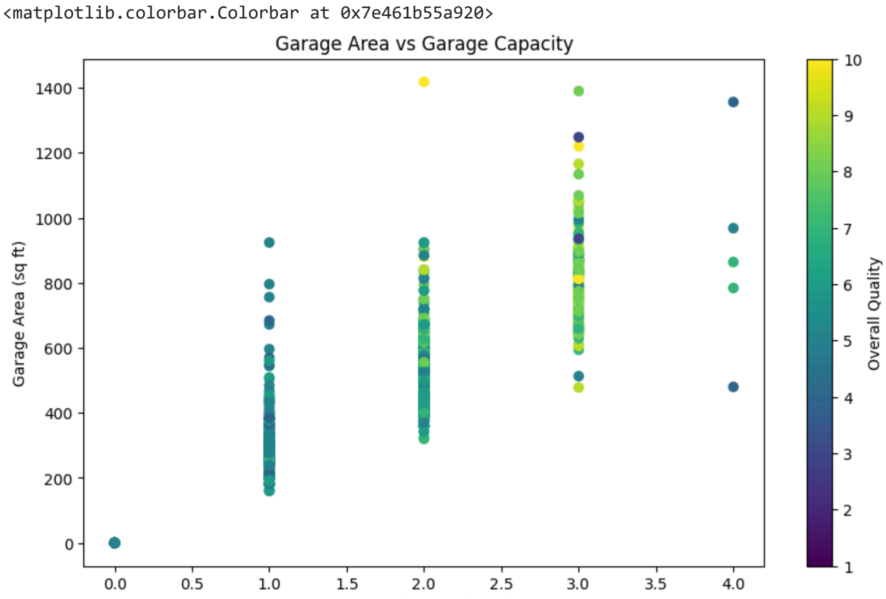
```
0
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1423 entries, 0 to 1459
Data columns (total 15 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   OverallQual   1423 non-null   int64
 1   GrLivArea     1423 non-null   int64
 2   TotalBsmtSF   1423 non-null   int64
 3   1stFlrSF      1423 non-null   int64
 4   FullBath      1423 non-null   int64
 5   YearBuilt     1423 non-null   int64
 6   YearRemodAdd  1423 non-null   int64
 7   GarageCars    1423 non-null   int64
 8   GarageArea    1423 non-null   int64
 9   Fireplaces    1423 non-null   int64
 10  LotArea       1423 non-null   int64
 11  ExterQual     1423 non-null   object
 12  KitchenQual   1423 non-null   object
 13  BsmtQual      1423 non-null   object
 14  SalePrice     1423 non-null   int64
dtypes: int64(12), object(3)
memory usage: 177.9+ KB
```

```
plt.figure(figsize=(10, 6))
plt.scatter(df['GarageCars'], df['GarageArea'], c=df['OverallQual'])
plt.xlabel('Garage Capacity (Number of Cars)')
plt.ylabel('Garage Area (sq ft)')
plt.title('Garage Area vs Garage Capacity')
plt.colorbar(label='Overall Quality')
```

```
<matplotlib.colorbar.Colorbar at 0x7e461b55a920>
```



Observations:

1. For a garage with the capacity for 2 cars and an excellent area (in square feet), the quality is rated as very good, 10/10.

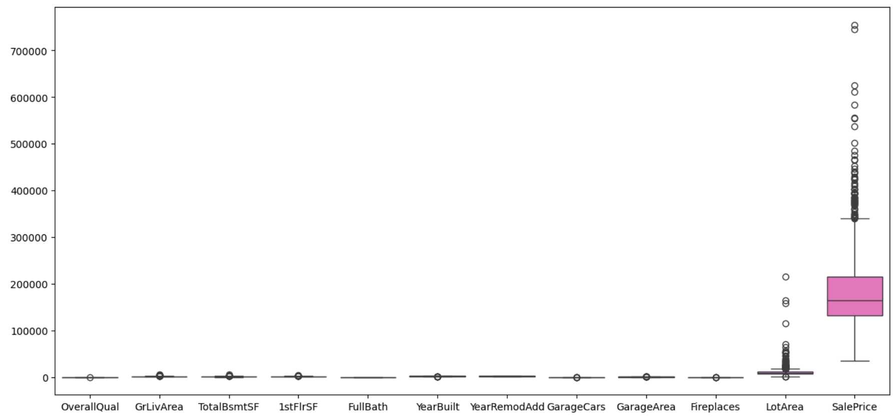
2. For a garage with the capacity for 3 cars and a good area, the quality is rated as good, ranging from 8.5 to 9 out of 10.

```
numericals = df.select_dtypes(include=['float64', 'int64'])
q1 = numericals.quantile(0.25)
q3 = numericals.quantile(0.75)
IQR = q3-q1
lower = q1 - 1.5*(IQR)
higher = q3 + 1.5*(IQR)
outliers = df[((numericals < lower) | (numericals > higher)).any(axis=1)]
print("Total Outliers in this dataset : ",outliers.shape[0])
```

```
Total Outliers in this dataset :  160
```

```
plt.figure(figsize=(15,7))
sns.boxplot(df)
```

```
<Axes: >
```




```
numericals = df.select_dtypes(include=['float64', 'int64'])
q1 = numericals.quantile(0.25)
q3 = numericals.quantile(0.75)
IQR = q3-q1
lower = q1 - 1.5*(IQR)
higher = q3 + 1.5*(IQR)
df1 = df[~((numericals < lower) | (numericals > higher)).any(axis=1)]
df1.shape
```

```
(1263, 15)
```

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1263 entries, 0 to 1459
Data columns (total 15 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   OverallQual   1263 non-null   int64
 1   GrLivArea     1263 non-null   int64
 2   TotalBsmtSF   1263 non-null   int64
 3   1stFlrSF      1263 non-null   int64
 4   FullBath      1263 non-null   int64
 5   YearBuilt     1263 non-null   int64
 6   YearRemodAdd  1263 non-null   int64
 7   GarageCars    1263 non-null   int64
 8   GarageArea    1263 non-null   int64
 9   Fireplaces    1263 non-null   int64
 10  LotArea       1263 non-null   int64
 11  ExterQual     1263 non-null   object
 12  KitchenQual   1263 non-null   object
 13  BsmtQual      1263 non-null   object
 14  SalePrice     1263 non-null   int64
dtypes: int64(12), object(3)
memory usage: 157.9+ KB
```

```
df1['ExterQual'].unique()
```

```
array(['Gd', 'TA', 'Ex', 'Fa'], dtype=object)
```

```
df1['KitchenQual'].unique()
```

```
array(['Gd', 'TA', 'Fa', 'Ex'], dtype=object)
```

```
df1['BsmtQual'].unique()
```

```
array(['Gd', 'TA', 'Ex', 'Fa'], dtype=object)
```

Ex (Excellent)

Gd (Good)

TA (Typical/Average)

Fa (Fair)

```
from sklearn.preprocessing import OrdinalEncoder
order = ['Fa','TA','Gd','Ex'] #Lowest to highest
encoder = OrdinalEncoder(categories=[order,order,order])
df1[['ExterQual','KitchenQual','BsmtQual']]=encoder.fit_transform(df1[['ExterQual','KitchenQual','BsmtQual']])
df1.head()
```

	OverallQual	GrLivArea	TotalBsmtSF	1stFlrSF	FullBath	YearBuilt	YearRemodAdd	Gar
0	7	1710	856	856	2	2003	2003	
1	6	1262	1262	1262	2	1976	1976	
2	7	1786	920	920	2	2001	2002	
3	7	1717	756	961	1	1915	1970	
4	8	2198	1145	1145	2	2000	2000	

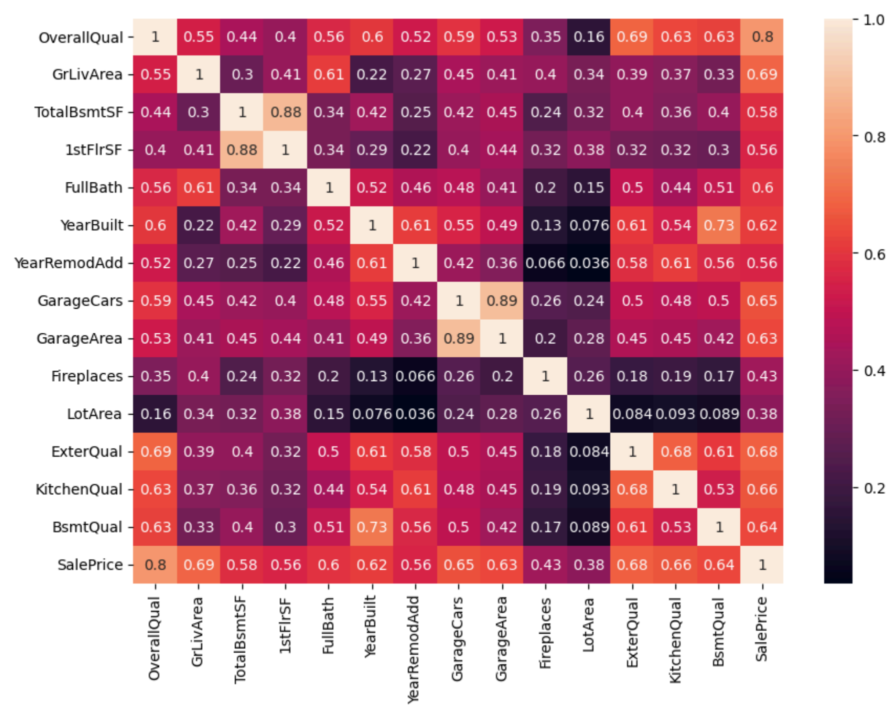
Next steps: Generate code with df1 View recommended plots

```
df1.describe()
```

	OverallQual	GrLivArea	TotalBsmtSF	1stFlrSF	FullBath	YearBuilt	YearR
count	1263.000000	1263.00000	1263.000000	1263.000000	1263.000000	1263.000000	126
mean	6.012668	1435.81631	1028.661916	1105.018211	1.520190	1971.153603	198
std	1.237232	418.77310	325.925913	314.878605	0.524544	29.748979	2
min	2.000000	438.00000	105.000000	438.000000	0.000000	1885.000000	195
25%	5.000000	1114.00000	797.000000	864.000000	1.000000	1954.000000	196
50%	6.000000	1414.00000	971.000000	1053.000000	2.000000	1973.000000	199
75%	7.000000	1705.50000	1230.000000	1309.500000	2.000000	2000.000000	200
max	10.000000	2730.00000	2000.000000	2117.000000	3.000000	2009.000000	201

```
plt.figure(figsize=(10,7))
sns.heatmap(df1.corr(),annot=True)
```

<Axes: >



Observation: Every column has a good correlation with the target variable, so we can't remove any columns...

```
details = ['OverallQual', 'GrLivArea', 'TotalBsmtSF', '1stFlrSF','YearBuilt', 'YearRemodAdd', 'GarageArea', 'LotArea', 'SalePrice']
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df1[details] = scaler.fit_transform(df1[details])
df1.head()
```

	OverallQual	GrLivArea	TotalBsmtSF	1stFlrSF	FullBath	YearBuilt	YearRemodAdd	Gar
0	0.625	0.554974	0.396306	0.248958	2	0.951613	0.883333	
1	0.500	0.359511	0.610554	0.490768	2	0.733871	0.433333	
2	0.625	0.588133	0.430079	0.287076	2	0.935484	0.866667	
3	0.625	0.558028	0.343536	0.311495	1	0.241935	0.333333	
4	0.750	0.767888	0.548813	0.421084	2	0.927419	0.833333	

Next steps: Generate code with df1 View recommended plots

```
x=df1.iloc[:,:-1] #Independent
y=df1.iloc[:,-1] #Dependent
x
```

	OverallQual	GrLivArea	TotalBsmtSF	1stFlrSF	FullBath	YearBuilt	YearRemodAdd
0	0.625	0.554974	0.396306	0.248958	2	0.951613	0.883333
1	0.500	0.359511	0.610554	0.490768	2	0.733871	0.433333
2	0.625	0.588133	0.430079	0.287076	2	0.935484	0.866667
3	0.625	0.558028	0.343536	0.311495	1	0.241935	0.333333
4	0.750	0.767888	0.548813	0.421084	2	0.927419	0.833333
...	...	...	...	...	...	...	...
1455	0.500	0.527487	0.447493	0.306730	2	0.919355	0.833333
1456	0.500	0.713351	0.758311	0.973794	2	0.750000	0.633333
1457	0.625	0.829843	0.552507	0.446694	2	0.451613	0.933333
1458	0.375	0.279232	0.513456	0.381179	1	0.524194	0.766667
1459	0.375	0.356894	0.607388	0.487195	1	0.645161	0.250000

1263 rows × 14 columns

Next steps: Generate code with x View recommended plots

```
y
```

```
0       0.569460
1       0.480892
2       0.618665
3       0.344760
4       0.705593
          ...
1455    0.459570
1456    0.574381
1457    0.759718
1458    0.351730
1459    0.369362
Name: SalePrice, Length: 1263, dtype: float64
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=0)
```

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
from sklearn.metrics import r2_score,mean_absolute_error
y_pred = lr.predict(x_test)
print(r2_score(y_test,y_pred))
print(mean_absolute_error(y_test,y_pred))
```

```
0.866458713200901
0.05133304229899392
```

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(x_train,y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
y_pred = rfr.predict(x_test)
print(r2_score(y_test,y_pred))
print(mean_absolute_error(y_test,y_pred))
```

```
0.8580312311610334
0.04963907350747254
```

Start coding or generate with AI.

Conclusion :

Linear Regression : 86.6 %

Random Forest : 85.4%

By hypertuning using grid search, I found that the accuracy did not improve significantly. The highest accuracy achieved was 86.6% using linear regression model.

Start coding or generate with AI.

Testing Part:

```
new = pd.DataFrame([[7,1710,856,856,2,2003,2003,2,548,0,8450,2.0,2.0,2.0]],columns=x.columns)
new
```

	OverallQual	GrLivArea	TotalBsmtSF	1stFlrSF	FullBath	YearBuilt	YearRemodAdd	GarageCars	GarageArea	Fireplaces	LotArea	ExterQu
0	7	1710	856	856	2	2003	2003	2	548	0	8450	

```
output = lr.predict(new)
print("Sales Price of house is : ",output)
```

```
Sales Price of house is :  [2070.92384172]
```

Start coding or generate with AI.