



Step-1 find missing values.

```
In [43]: rows=train_data.shape[0]
rows
r_rows=train_data.isnull().sum()
r_rows
a=r_rows/rows*100
a
```

Out[43]:

Id	0.000000
MSSubClass	0.000000
MSZoning	0.000000
LotFrontage	17.739726
LotArea	0.000000
...	...
MoSold	0.000000
YrSold	0.000000
SaleType	0.000000
SaleCondition	0.000000
SalePrice	0.000000
Length: 81, dtype: float64	

Step-2 Drop variables having more than 25% missing values, and less than 25% missing values will be fill by using appropriate central tendencies

```
In [44]: data=data.drop(columns=["Alley", "FireplaceQu", "PoolQC", "Fence", "MiscFeature"])

In [45]: #numerical data we used to fill mean.
data["LotFrontage"]=data["LotFrontage"].fillna(data["LotFrontage"].mean())
data["GarageYrBlt"]=data["GarageYrBlt"].fillna(data["GarageYrBlt"].mean())
data["MasVnrArea"]=data["MasVnrArea"].fillna(data["MasVnrArea"].mean())
data["BsmtFinSF1"]=data["BsmtFinSF1"].fillna(data["BsmtFinSF1"].mean())
data["BsmtUnfSF"]=data["BsmtUnfSF"].fillna(data["BsmtUnfSF"].mean())
data["TotalBsmtSF"]=data["TotalBsmtSF"].fillna(data["TotalBsmtSF"].mean())
data["BsmtFinSF2"]=data["BsmtFinSF2"].fillna(data["BsmtFinSF2"].mean())
data["BsmtFullBath"]=data["BsmtFullBath"].fillna(data["BsmtFullBath"].mean())
data["BsmtHalfBath"]=data["BsmtHalfBath"].fillna(data["BsmtHalfBath"].mean())
data["GarageCars"]=data["GarageCars"].fillna(data["GarageCars"].mean())
data["GarageArea"]=data["GarageArea"].fillna(data["GarageArea"].mean())
```

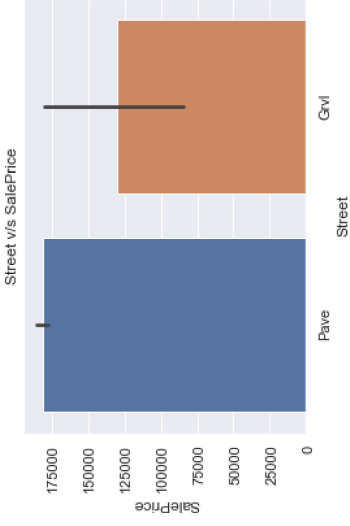
In [46]:

```
#Chracteries data we used to mode or so on.
data["Electrical"]=data["Electrical"].fillna(data["Electrical"].mode()[0])
data["MSZoning"]=data["MSZoning"].fillna(data["MSZoning"].mode()[0])
data["Utilities"]=data["Utilities"].fillna(data["Utilities"].mode()[0])
data["Exterior1st"]=data["Exterior1st"].fillna(data["Exterior1st"].mode()[0])
data["Exterior2nd"]=data["Exterior2nd"].fillna(data["Exterior2nd"].mode()[0])
data["MasVnrType"]=data["MasVnrType"].fillna(data["MasVnrType"].mode()[0])
data["BsmtQual"]=data["BsmtQual"].fillna(data["BsmtQual"].mode()[0])
data["BsmtCond"]=data["BsmtCond"].fillna(data["BsmtCond"].mode()[0])
data["BsmtExposure"]=data["BsmtExposure"].fillna(data["BsmtExposure"].mode())
data["BsmtFinType1"]=data["BsmtFinType1"].fillna(data["BsmtFinType1"].mode())
data["BsmtFinType2"]=data["BsmtFinType2"].fillna(data["BsmtFinType2"].mode())
data["KitchenQual"]=data["KitchenQual"].fillna(data["KitchenQual"].mode()[0])
data["Functional"]=data["Functional"].fillna(data["Functional"].mode()[0])
data["GarageType"]=data["GarageType"].fillna(data["GarageType"].mode()[0])
data["GarageQual"]=data["GarageQual"].fillna(data["GarageQual"].mode()[0])
data["GarageCond"]=data["GarageCond"].fillna(data["GarageCond"].mode()[0])
data["SaleType"]=data["SaleType"].fillna(data["SaleType"].mode()[0])
data["GarageFinish"]=data["GarageFinish"].fillna(data["GarageFinish"].mode())
```

EDA

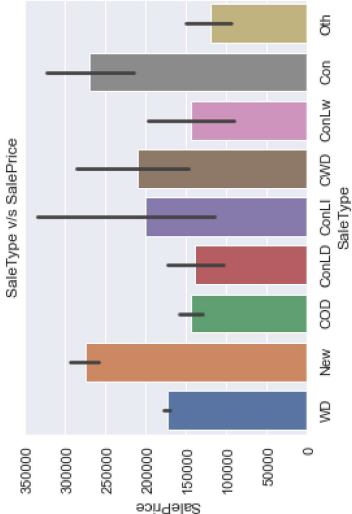
In [47]:

```
#In this graph we see that in street "Pave(Paved)" is more higher as compare
sns.barplot(y="SalePrice", x="Street", data=train_data)
plt.title("Street v/s SalePrice")
plt.show()
```



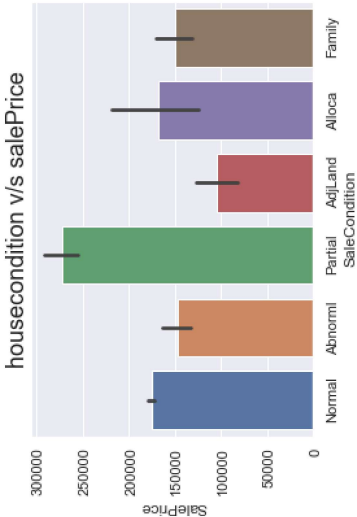
In [48]:

```
1 #In this graph we observed that in "SaleType" new constructed houses and con(  
2 sns.barplot(x="SaleType",y="SalePrice",data=train_data)  
3 plt.title("SaleType v/s SalePrice")  
4 plt.show()
```



In [50]:

```
1 # We observed that in salecondition (partial) saleprice is very high.  
2 sns.barplot(data=train_data, y="SalePrice", x="SaleCondition")  
3 plt.title("housecondition v/s salePrice", size=18)  
4 plt.show()
```



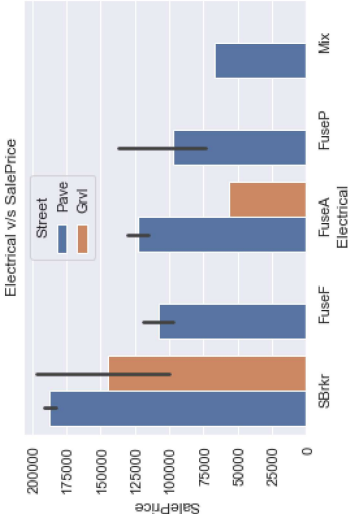
In [51]:

```
1 #In this graph were we observed saleprice is less than privious(2009) year.  
2 sns.lineplot(data=train_data, x="YrSold", y="SalePrice")  
3 plt.show()
```



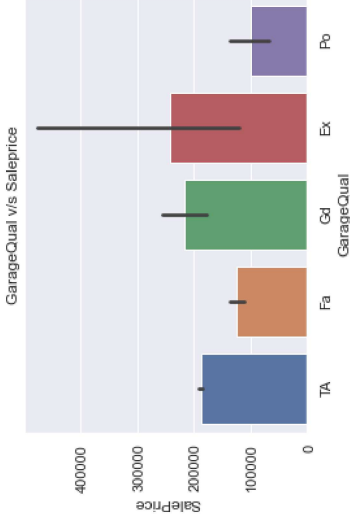
In [49]:

```
1 #we observed that in electrical(SBkrkr) "street" price rate is very high.  
2 sns.barplot(data=train_data,x="Electrical",y="SalePrice",hue="Street")  
3 plt.title("Electrical v/s SalePrice")  
4 plt.show()
```



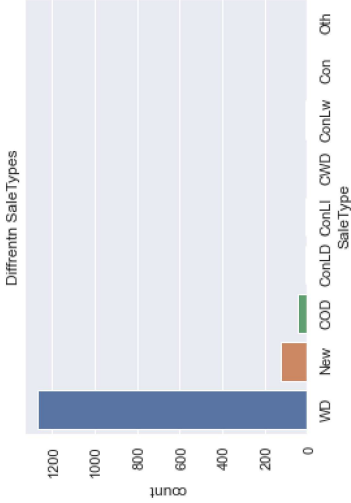
In [52]:

```
1 #if your Garage quality is excellent and good so the sale price of the house
2 sns.barplot(x="GarageQual",y="SalePrice",data=train_data)
3 plt.title("GarageQual v/s SalePrice")
4 plt.show()
```



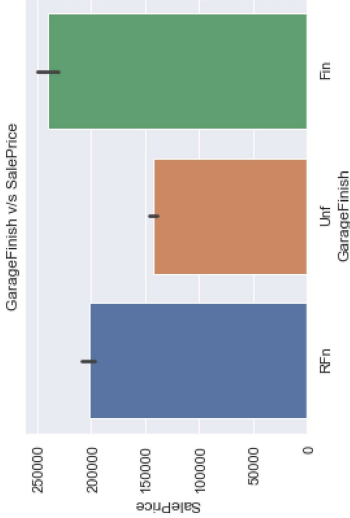
In [54]:

```
1 sns.countplot(x="SaleType", data=train_data)
2 plt.title("Diffrentn SaleTypes")
3 plt.show()
```



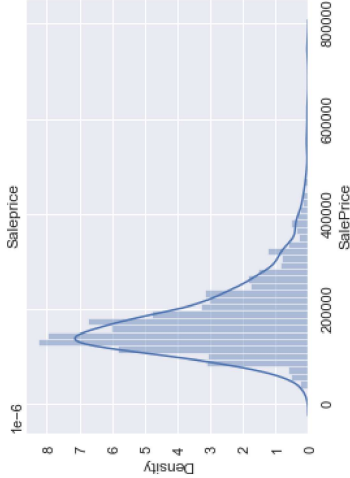
In [55]:

```
1 #we observed that in GrageFinish (finish) sale price is vary high.
2 sns.barplot(x="GarageFinish",y="SalePrice",data=train_data)
3 plt.title("GarageFinish v/s SalePrice")
4 plt.show()
```



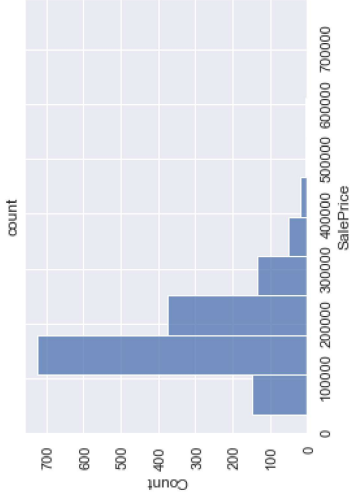
In [53]:

```
1 #we observed that maximum pepole buy houses in the range of 2Lakh.
2 sns.distplot(train_data["SalePrice"])
3 plt.title("SalePrice")
4 plt.show()
```



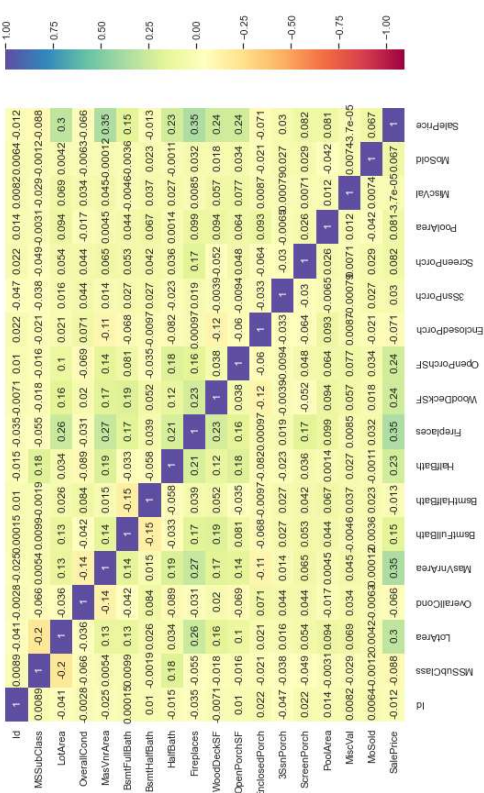
In [56]:

```
1 #we observed that maximum pepole buy houses in the range of 2Lakh.  
2 sns.histplot(train_data["SalePrice"],kde=False, bins=10)  
3 plt.title("count")  
4 plt.show()
```



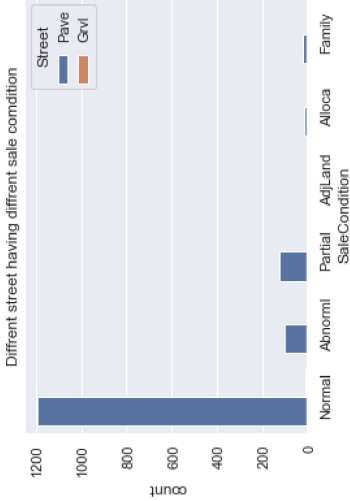
In [60]:

```
1 sns.set(rc={"figure.figsize":(15,8)})  
2 num_col=data.select_dtypes(exclude='object')  
3 sns.heatmap(data.corr(),vmin=-1,vmax=1,annot=True,cmap="Spectral")  
4 plt.show()
```



In [57]:

```
1 #we observed that more pepoles having normal houses in street(Pave-Paved).  
2 sns.countplot(data=train_data, x="SaleCondition", hue="Street")  
3 plt.title("Diffrent street having diffrent sale comdition")  
4 plt.show()
```



## Multicollinary(using VIF method)

In [59]:

```
1 vif_data=pd.DataFrame()  
2 vif_data["feature"]=num_col.columns  
3 vif_data["VIF"]=[variance_inflation_factor(num_col.values,i)  
4                 for i in range(len(num_col.columns))]  
5 vif_data  
6 vif_data.round(1)  
7 data=data.drop(columns=["BsmtFinSF1", "LotFrontage", "GarageArea", "BedroomAbvG  
8 #if vif<10 we drop that variable  
9 #variable having range 5 to 10 showa high multicollinarity.
```

## Outlier Detection(Extream end outliers will be removed.)

In [21]:

```
1 data=data.drop(columns=['LotArea', 'MasVnrArea', 'WoodDeckSF', 'OpenPorchSF', 'E
```

In [22]:

1 data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2919 entries, 0 to 2918
Data columns (total 50 columns):
 #   Column      Non-Null Count  Dtype  ---
 0   Id           2919 non-null   int64  ---
 1   MSSubClass   2919 non-null   int64  ---
 2   MSZoning     2919 non-null   object 
 3   Street       2919 non-null   object 
 4   LotShape     2919 non-null   object 
 5   LandContour  2919 non-null   object 
 6   Utilities    2919 non-null   object 
 7   LotConfig    2919 non-null   object 
 8   LandSlope    2919 non-null   object 
 9   Neighborhood 2919 non-null   object 
10   Condition1   2919 non-null   object 
11   Condition2   2919 non-null   object 
12   BldgType     2919 non-null   object 
13   HouseStyle   2919 non-null   object 
14   OverallCond  2919 non-null   int64  
15   RoofStyle    2919 non-null   object 
16   RoofMatl     2919 non-null   object 
17   Exterior1st  2919 non-null   object 
18   Exterior2nd  2919 non-null   object 
19   MasVnrType   2919 non-null   object 
20   ExterQual    2919 non-null   object 
21   ExterCond    2919 non-null   object 
22   Foundation   2919 non-null   object 
23   BsmtQual     2919 non-null   object 
24   BsmtCond     2919 non-null   object 
25   BsmtExposure 2919 non-null   object 
26   BsmtFinType1 2919 non-null   object 
27   BsmtFinType2 2919 non-null   object 
28   Heating      2919 non-null   object 
29   HeatingQC    2919 non-null   object 
30   CentralAir   2919 non-null   object 
31   Electrical   2919 non-null   object 
32   BsmtFullBath 2919 non-null   float64 
33   BsmtHalfBath 2919 non-null   float64 
34   HalfBath     2919 non-null   int64  
35   KitchenQual  2919 non-null   object 
36   Functional   2919 non-null   object 
37   Fireplaces   2919 non-null   int64  
38   GarageType   2919 non-null   object 
39   GarageFinish 2919 non-null   object 
40   GarageQual   2919 non-null   object 
41   GarageCond   2919 non-null   object 
42   PavedDrive   2919 non-null   object 
43   3SsnPorch    2919 non-null   int64  
44   PoolArea     2919 non-null   int64  
45   MiscVal      2919 non-null   int64  
46   MoSold       2919 non-null   int64  
47   SaleType     2919 non-null   object 
48   SaleCondition 2919 non-null   object
```

```
49 SalePrice      2919 non-null   float64
dtypes: float64(3), int64(9), object(38)
memory usage: 1.1+ MB
```

## Fill appropriate outlier using z\_score method.

In [61]:

```
1 upper_limit=data['MSSubClass'].mean()+3*data['MSSubClass'].std()
2
3 lower_limit=data['MSSubClass'].mean()-3*data['MSSubClass'].std()
4 data['MSSubClass'] = np.where(
5     data['MSSubClass']>upper_limit,
6     upper_limit,
7     np.where(
8         data['MSSubClass']<0,
9         0,
10        data['MSSubClass']
11    )
12 )
```

In [62]:

```
1 upper_limit=data['BsmtFullBath'].mean()+3*data['BsmtFullBath'].std()
2
3 lower_limit=data['BsmtFullBath'].mean()-3*data['BsmtFullBath'].std()
4 data['BsmtFullBath'] = np.where(
5     data['BsmtFullBath']>upper_limit,
6     upper_limit,
7     np.where(
8         data['BsmtFullBath']<0,
9         0,
10        data['BsmtFullBath']
11    )
12 )
```

In [63]:

```
1 upper_limit=data['BsmtHalfBath'].mean()+3*data['BsmtHalfBath'].std()
2
3 lower_limit=data['BsmtHalfBath'].mean()-3*data['BsmtHalfBath'].std()
4 data['BsmtHalfBath'] = np.where(
5     data['BsmtHalfBath']>upper_limit,
6     upper_limit,
7     np.where(
8         data['BsmtHalfBath']<0,
9         0,
10        data['BsmtHalfBath']
11    )
12 )
```

In [64]:

```
1 upper_limit=data['Fireplaces'].mean()+3*data['Fireplaces'].std()
2
3 lower_limit=data['Fireplaces'].mean()-3*data['Fireplaces'].std()
4 data['Fireplaces'] = np.where(
5     data['Fireplaces']>upper_limit,
6     upper_limit,
7     np.where(
8         data['Fireplaces']<0,
9         0,
10        data['Fireplaces']
11    )
12 )
```

In [65]:

```
1 upper_limit=data['3SsnPorch'].mean()+3*data['3SsnPorch'].std()
2
3 lower_limit=data['3SsnPorch'].mean()-3*data['3SsnPorch'].std()
4 data['3SsnPorch'] = np.where(
5     data['3SsnPorch']>upper_limit,
6     upper_limit,
7     np.where(
8         data['3SsnPorch']<0,
9         0,
10        data['3SsnPorch']
11    )
12 )
```

In [66]:

```
1 upper_limit=data['PoolArea'].mean()+3*data['PoolArea'].std()
2
3 lower_limit=data['PoolArea'].mean()-3*data['PoolArea'].std()
4 data['PoolArea'] = np.where(
5     data['PoolArea']>upper_limit,
6     upper_limit,
7     np.where(
8         data['PoolArea']<0,
9         0,
10        data['PoolArea']
11    )
12 )
```

In [67]:

```
1 upper_limit=data['MiscVal'].mean()+3*data['MiscVal'].std()
2
3 lower_limit=data['MiscVal'].mean()-3*data['MiscVal'].std()
4 data['MiscVal'] = np.where(
5     data['MiscVal']>upper_limit,
6     upper_limit,
7     np.where(
8         data['MiscVal']<0,
9         0,
10        data['MiscVal']
11    )
12 )
```

In [ ]:

## All object converted into integers.

In [68]:

```
1 from sklearn.preprocessing import LabelEncoder
2 le=LabelEncoder()
3 object_list=data.select_dtypes(include=[ 'object' ]).columns
4 for i in object_list:
5     data[i]=le.fit_transform(data[i])
```

## Train-Test-Splitting

In [69]:

```
1 x=data.iloc[:, :-1].values
2 y=data.iloc[:, -1].values
```

In [70]:

```
1 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_stat
```

## Linear Regression

In [71]:

```
1 from sklearn.linear_model import LinearRegression
2 reg=LinearRegression()
3 reg.fit(x_train,y_train)
4 print(f'The score of Linear regression is {reg.score(x_test,y_test)*100} %')
```

The score of Linear regression is 39.260968207579374 %

In [72]:

```
1 y_pred=reg.predict(x_test)
```

## XGBOOST Modeling.

In [73]:

```
1 import xgboost as xgb
2 xgb.XGBRegressor().get_params()
3 xg_reg=xgb.XGBRegressor()
4 xg_reg.fit(x_train,y_train)
5 print(f'The score of xgboost is {xg_reg.score(x_test,y_test)*100} %')
```

The score of xgboost is 72.00224981007838 %

In [74]:

```
1 xgb_preds = xg_reg.predict(x_test)
```

## Decision Tree

```
In [75]: 1 from sklearn.tree import DecisionTreeRegressor
2 regr=DecisionTreeRegressor()
3 regr.fit(x_train,y_train)
4 print(f'The score of Decision Tree is {regr.score(x_test,y_test)*100}%')
```

The score of Decision Tree is 39.260968207579374%

```
In [76]: 1 y_pred1=regr.predict(x_test)
```

```
In [ ]: 1
```

```
In [ ]: 1
```