```
import kagglehub
kagglehub.login()
```

Kaggle credentials successfully validated.

```
Kaggle credentials set.
Kaggle credentials successfully validated.
```

+ Code + Text

Importing Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats as stats
from scipy.stats import norm
import statsmodels.api as sm
import matplotlib.pyplot as plt
from scipy.stats import skew, norm
from sklearn.neighbors import KNeighborsRegressor

import warnings
warnings.filterwarnings(action="ignore")
```

Working directories

```
input_path1 = '../input/house-prices-advanced-regression-techniques/'
input_path2 = '../input/ames-housing-dataset/'
```

```
house_data = pd.read_csv(aliamini93_ames_housing_dataset_path + '/AmesHousing.csv')
test = pd.read_csv(house_prices_advanced_regression_techniques_path + '/test.csv')
data_w = house_data.copy()
data_w.columns = data_w.columns.str.replace(' ', '')
data_w.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 82 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Order           2930 non-null   int64
 1   PID             2930 non-null   int64
 2   MSSubClass      2930 non-null   int64
 3   MSZoning        2930 non-null   object
 4   LotFrontage     2440 non-null   float64
 5   LotArea         2930 non-null   int64
 6   Street          2930 non-null   object
 7   Alley           198 non-null    object
 8   LotShape        2930 non-null   object
 9   LandContour     2930 non-null   object
 10  Utilities       2930 non-null   object
 11  LotConfig       2930 non-null   object
 12  LandSlope       2930 non-null   object
 13  Neighborhood    2930 non-null   object
 14  Condition1      2930 non-null   object
 15  Condition2      2930 non-null   object
 16  BldgType        2930 non-null   object
 17  HouseStyle      2930 non-null   object
 18  OverallQual     2930 non-null   int64
 19  OverallCond     2930 non-null   int64
 20  YearBuilt       2930 non-null   int64
 21  YearRemod/Add   2930 non-null   int64
 22  RoofStyle       2930 non-null   object
 23  RoofMatl        2930 non-null   object
 24  Exterior1st     2930 non-null   object
 25  Exterior2nd     2930 non-null   object
```

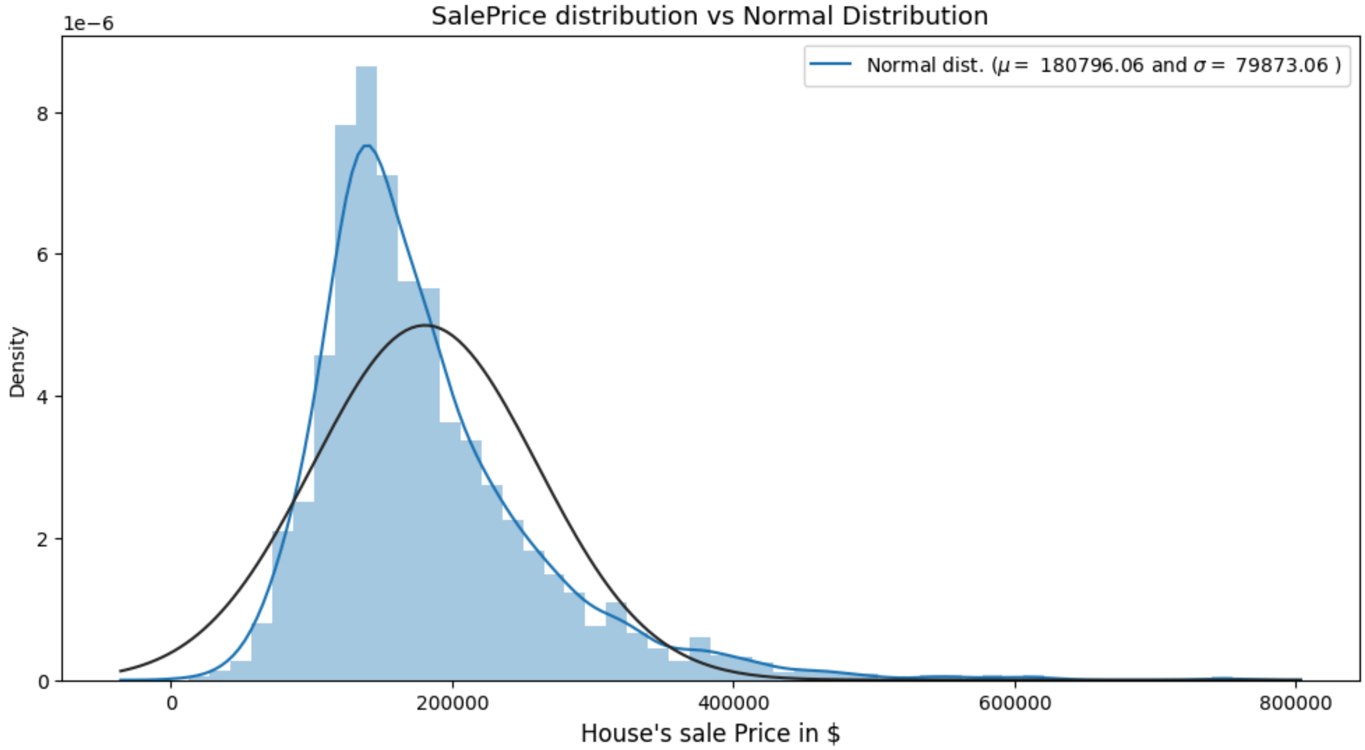
```
 26  MasVnrType     1155 non-null   object
 27  MasVnrArea     2907 non-null   float64
 28  ExterQual      2930 non-null   object
 29  ExterCond      2930 non-null   object
 30  Foundation     2930 non-null   object
 31  BsmtQual       2850 non-null   object
 32  BsmtCond       2850 non-null   object
 33  BsmtExposure   2847 non-null   object
 34  BsmtFinType1   2850 non-null   object
 35  BsmtFinSF1     2929 non-null   float64
 36  BsmtFinType2   2849 non-null   object
 37  BsmtFinSF2     2929 non-null   float64
 38  BsmtUnfSF      2929 non-null   float64
 39  TotalBsmtSF    2929 non-null   float64
 40  Heating        2930 non-null   object
 41  HeatingQC      2930 non-null   object
 42  CentralAir     2930 non-null   object
 43  Electrical     2929 non-null   object
 44  1stFlrSF       2930 non-null   int64
 45  2ndFlrSF       2930 non-null   int64
 46  LowQualFinSF   2930 non-null   int64
 47  GrLivArea      2930 non-null   int64
 48  BsmtFullBath   2928 non-null   float64
 49  BsmtHalfBath   2928 non-null   float64
 50  FullBath       2930 non-null   int64
 51  HalfBath       2930 non-null   int64
 52  BedroomAbvGr   2930 non-null   int64
```

```
data_w.head()
```

ley	LotShape	LandContour	...	PoolA
NaN	IR1	Lvl	...	
NaN	Reg	Lvl	...	
NaN	IR1	Lvl	...	
NaN	Reg	Lvl	...	
NaN	IR1	Lvl	...	

Getting the main parameters of the Normal Ditribution

```
(mu, sigma) = norm.fit(data_w['SalePrice'])

plt.figure(figsize = (12,6))
sns.distplot(data_w['SalePrice'], kde = True, hist=True, fit = norm)
plt.title('SalePrice distribution vs Normal Distribution', fontsize = 13)
plt.xlabel("House's sale Price in $", fontsize = 12)
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.show()
```



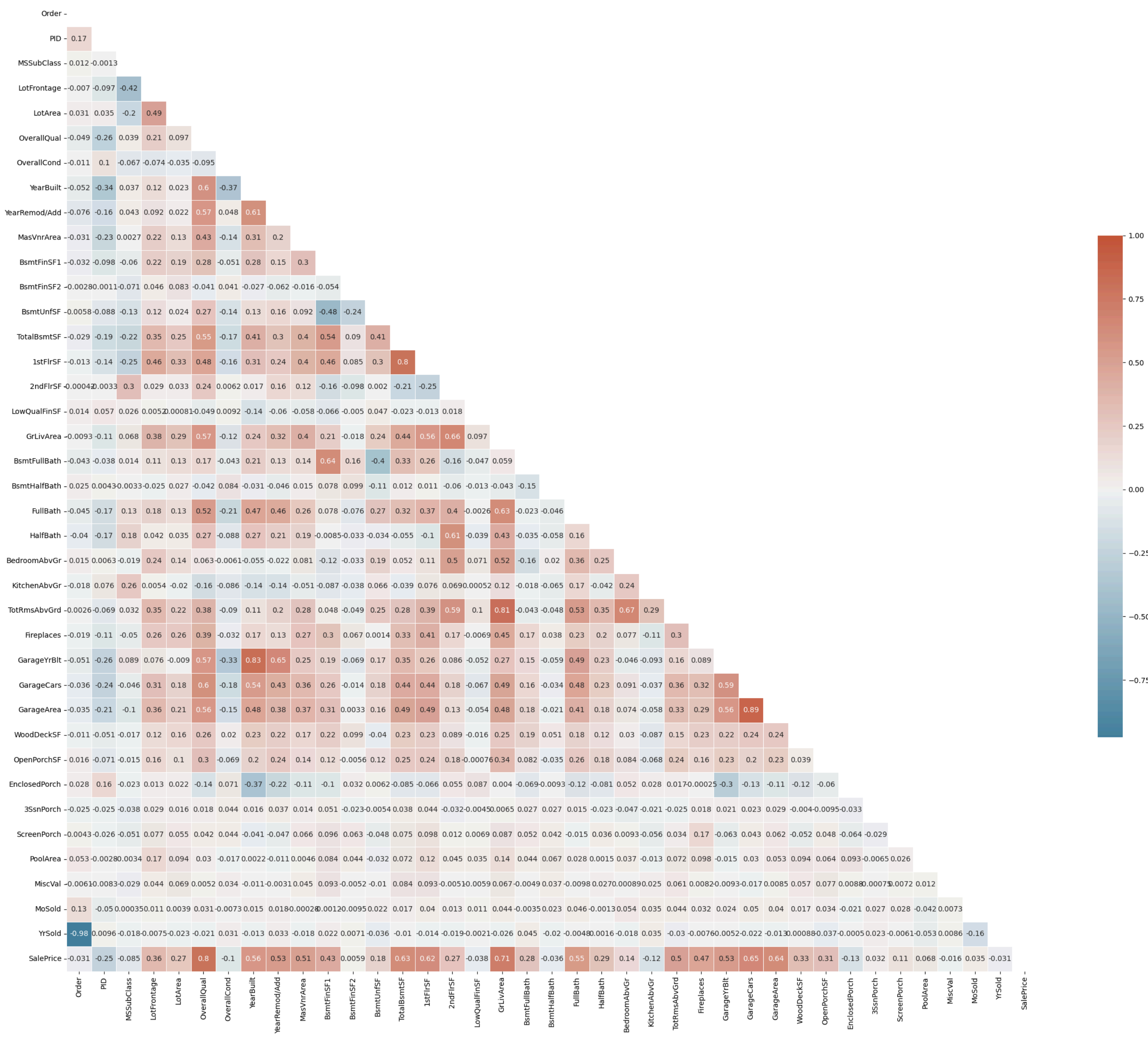
Skew and kurt

```
from scipy import stats

shap_t,shap_p = stats.shapiro(data_w['SalePrice'])

print("Skewness: %f" % abs(data_w['SalePrice']).skew())
print("Kurtosis: %f" % abs(data_w['SalePrice']).kurt())
print("Shapiro_Test: %f" % shap_t)
print("Shapiro_Test: %f" % shap_p)
```

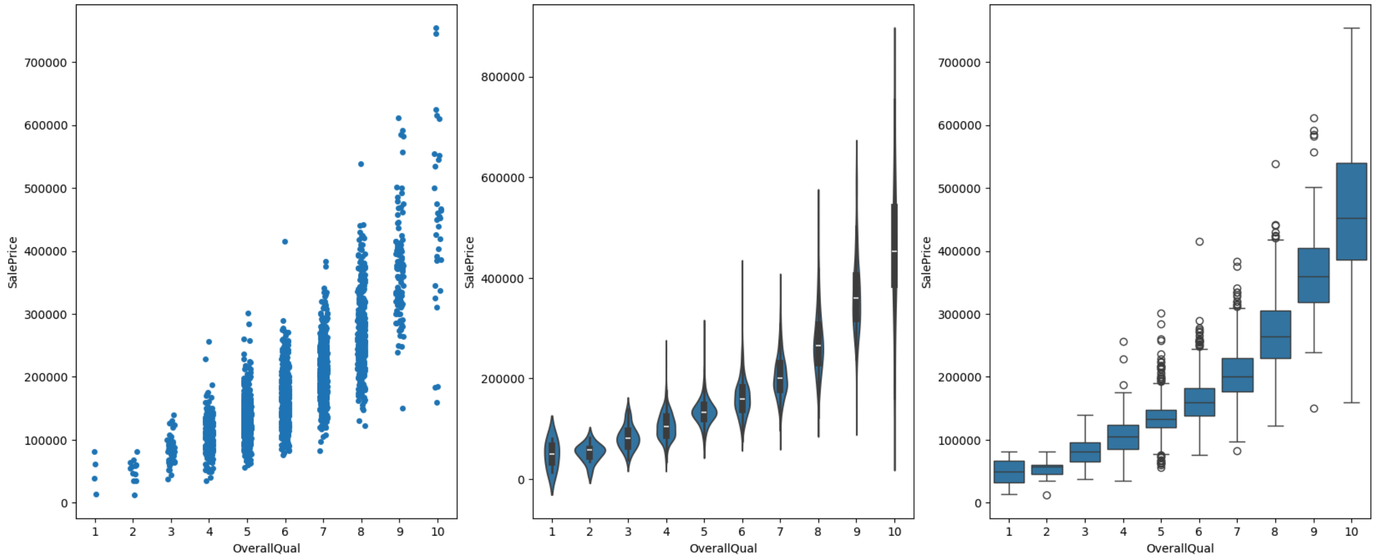
```
Skewness: 1.743500
Kurtosis: 5.118900
Shapiro_Test: 0.876261
Shapiro_Test: 0.000000
```

Correlation Matrix

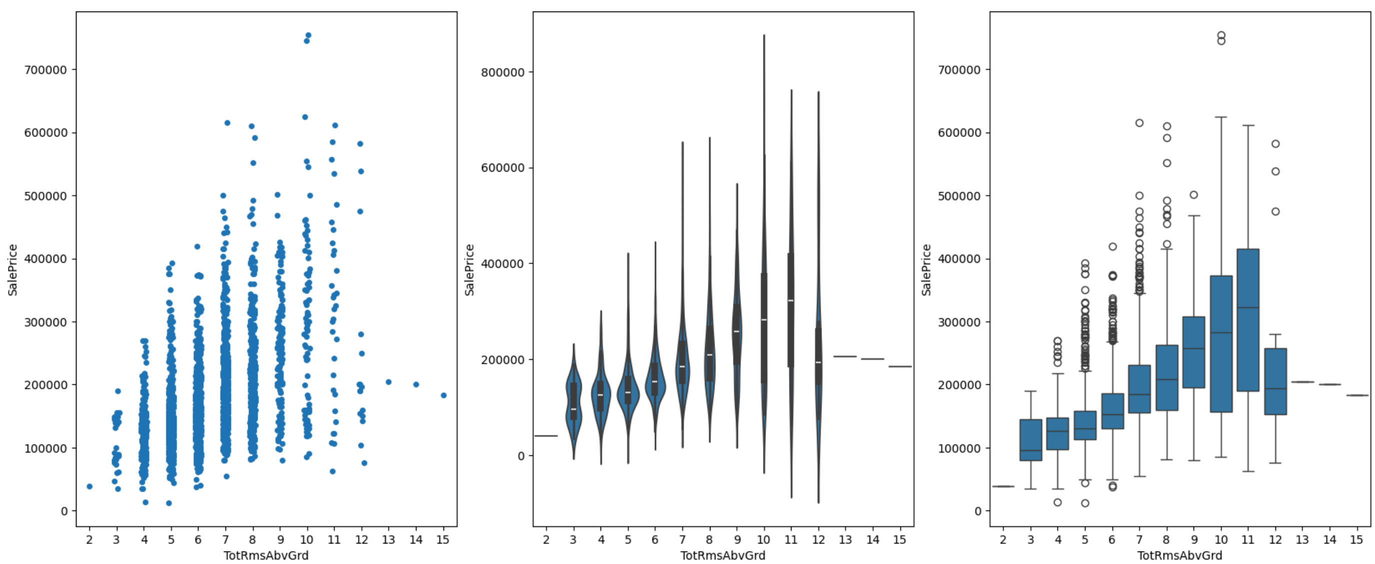
```
f, ax = plt.subplots(figsize=(30, 25))
mat = data_w.corr(method='pearson', numeric_only=True)
mask = np.triu(np.ones_like(mat, dtype=bool))

sns.heatmap(
    mat, mask=mask,
    cmap=sns.diverging_palette(230, 20, as_cmap=True),
    vmax=1, center=0, annot=True, square=True,
    linewidths=.5, cbar_kws={"shrink": .5}
)
plt.show()
```



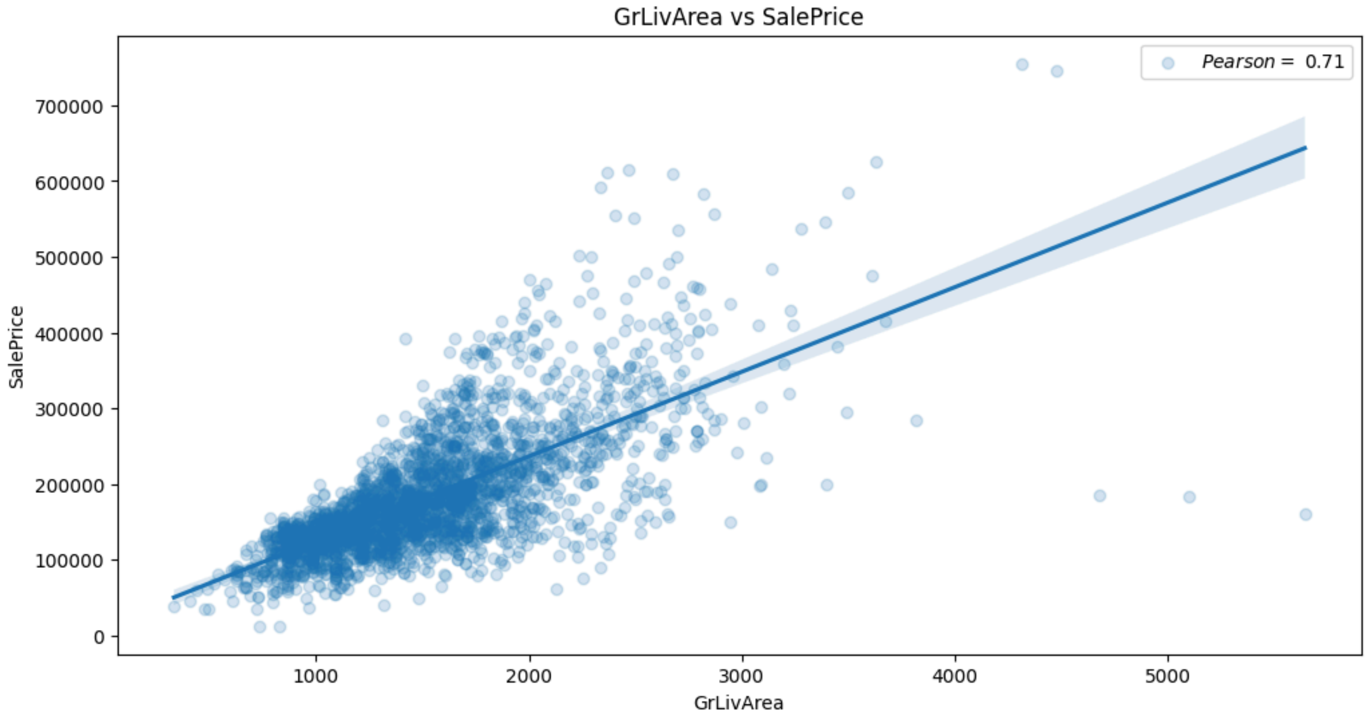

```
# OverallQuall - SalePrice [Pearson = 0.8]

figure, ax = plt.subplots(1,3, figsize = (20,8))
sns.stripplot(data=data_w, x = 'OverallQual', y='SalePrice', ax = ax[0])
sns.violinplot(data=data_w, x = 'OverallQual', y='SalePrice', ax = ax[1])
sns.boxplot(data=data_w, x = 'OverallQual', y='SalePrice', ax = ax[2])
plt.show()
```



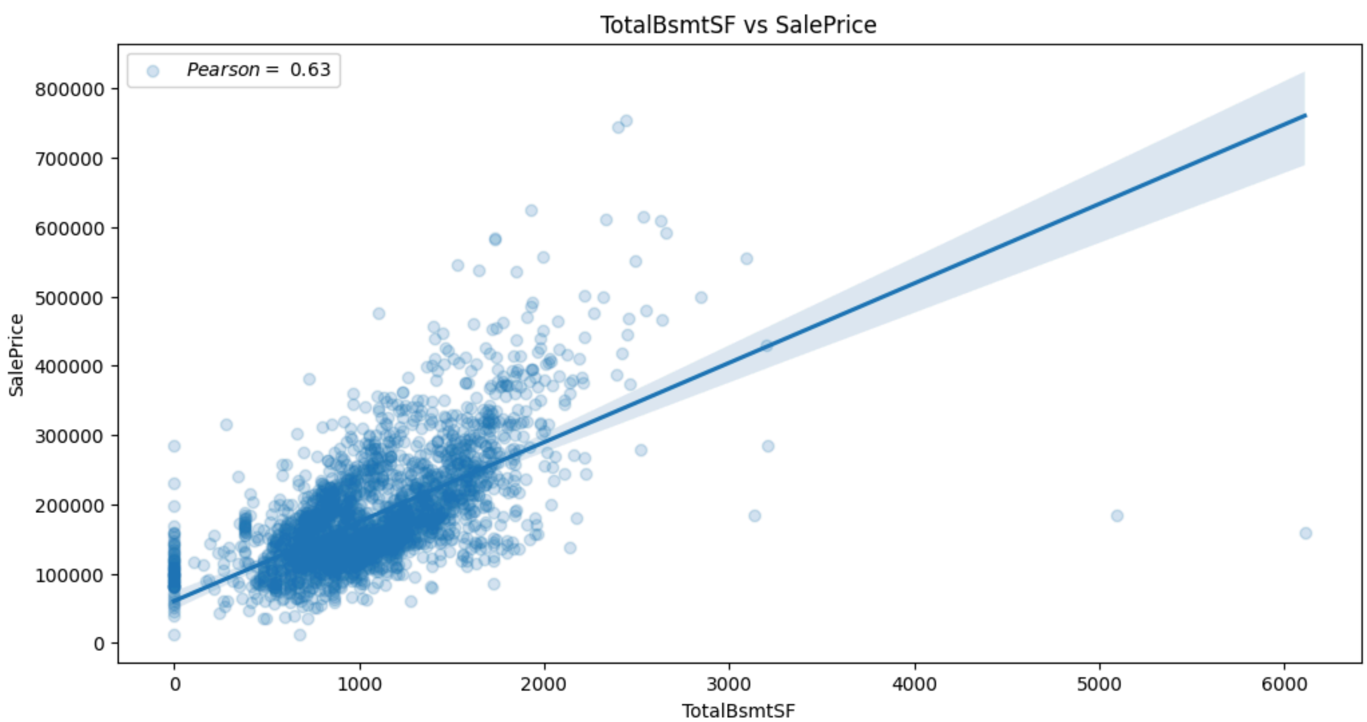

```
# TotRmsAbvGrd - SalePrice [Pearson = 0.50]

figure, ax = plt.subplots(1,3, figsize = (20,8))
sns.stripplot(data=data_w, x = 'TotRmsAbvGrd', y='SalePrice', ax = ax[0])
sns.violinplot(data=data_w, x = 'TotRmsAbvGrd', y='SalePrice', ax = ax[1])
sns.boxplot(data=data_w, x = 'TotRmsAbvGrd', y='SalePrice', ax = ax[2])
plt.show()
```




```
# GrLivArea vs SalePrice [corr = 0.71]

Pearson_GrLiv = 0.71
plt.figure(figsize = (12,6))
sns.regplot(data=data_w, x = 'GrLivArea', y='SalePrice', scatter_kws={'alpha':0.2})
plt.title('GrLivArea vs SalePrice', fontsize = 12)
plt.legend(['$Pearson=$ {:.2f}'.format(Pearson_GrLiv)], loc = 'best')
plt.show()
```

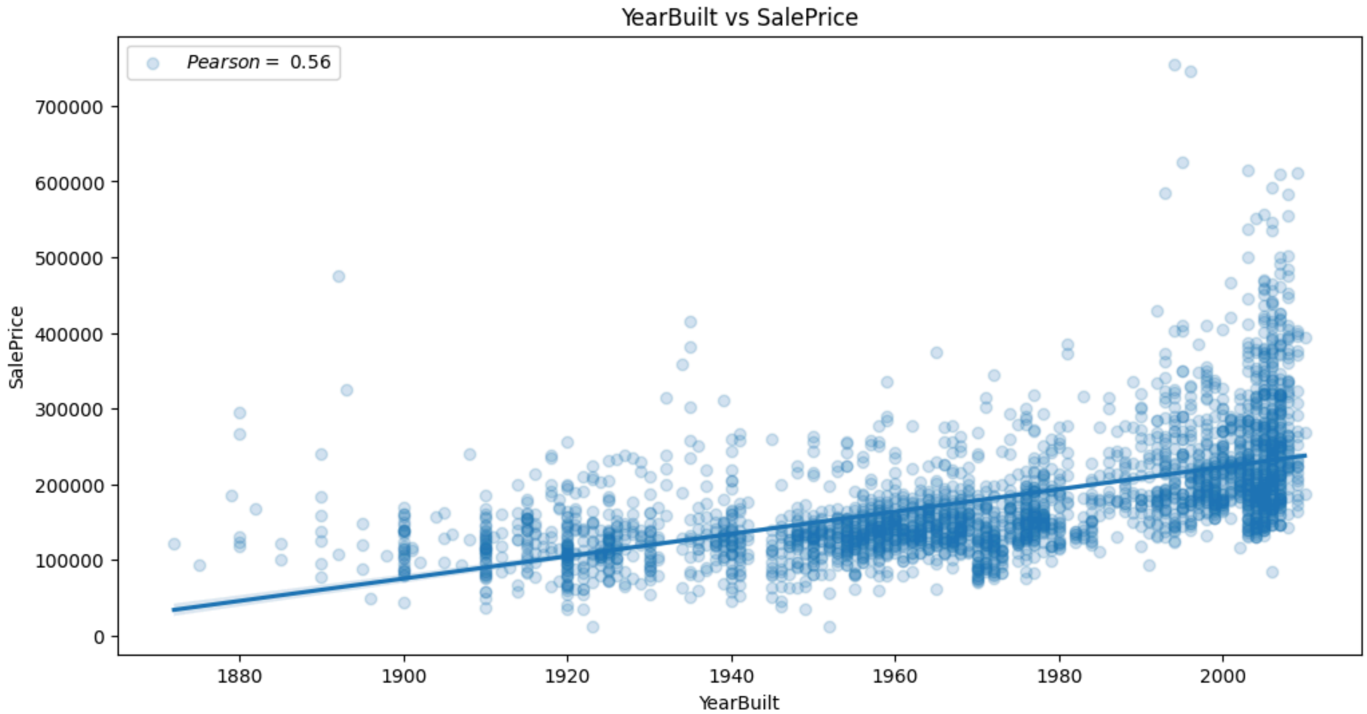



```
Pearson_TBSF = 0.63
plt.figure(figsize = (12,6))
sns.regplot(data=data_w, x = 'TotalBsmtSF', y='SalePrice', scatter_kws={'alpha':0.2})
plt.title('TotalBsmtSF vs SalePrice', fontsize = 12)
plt.legend(['$Pearson=$ {:.2f}'.format(Pearson_TBSF)], loc = 'best')
plt.show()
```



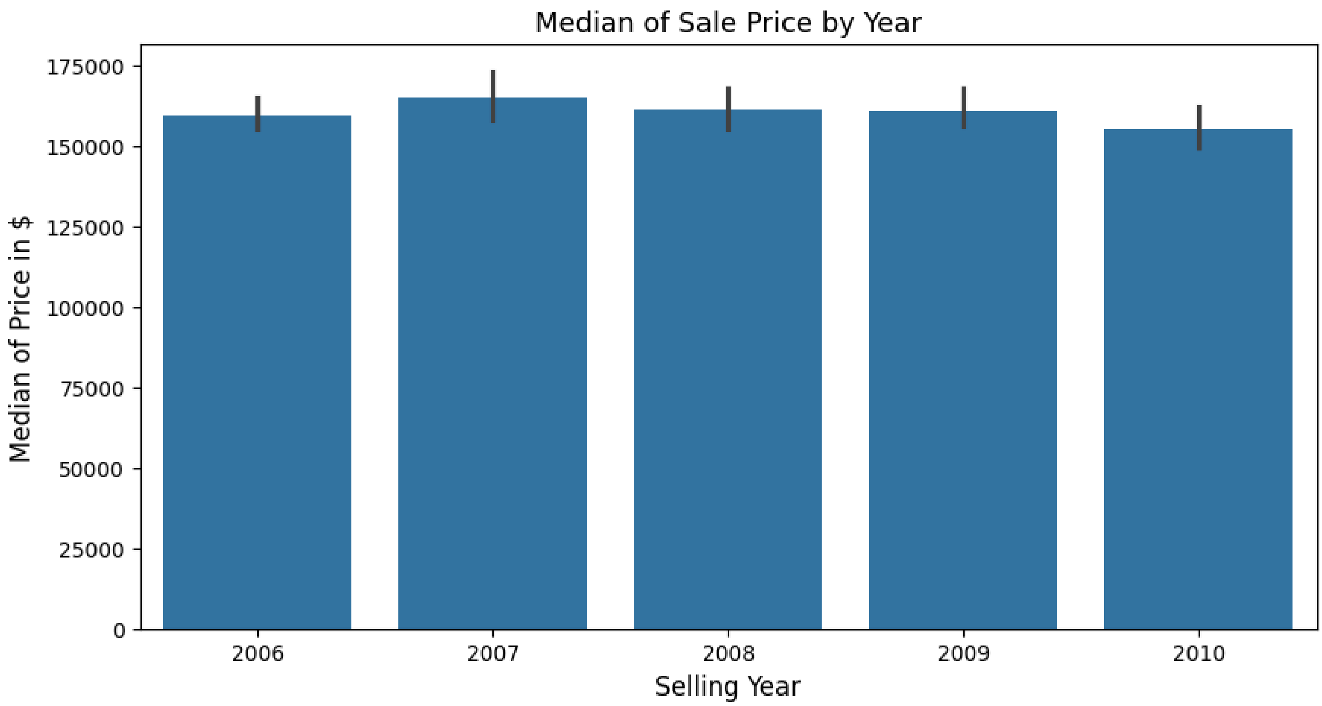
YearBuilt vs SalePrice

```
Pearson_YrBlt = 0.56
plt.figure(figsize = (12,6))
sns.regplot(data=data_w, x = 'YearBuilt', y='SalePrice', scatter_kws={'alpha':0.2})
plt.title('YearBuilt vs SalePrice', fontsize = 12)
plt.legend(['$Pearson=$ {:.2f}'.format(Pearson_YrBlt)], loc = 'best')
plt.show()
```



Median of Sale Price by Year

```
plt.figure(figsize = (10,5))
sns.barplot(x='YrSold', y="SalePrice", data = data_w, estimator = np.median)
plt.title('Median of Sale Price by Year', fontsize = 13)
plt.xlabel('Selling Year', fontsize = 12)
plt.ylabel('Median of Price in $', fontsize = 12)
plt.show()
```



Separating Target and Features

```
target = data_w['SalePrice']
test_id = test['Id']
test = test.drop(['Id'],axis = 1)
data_w2 = data_w.drop(['SalePrice','Order','PID'], axis = 1)
```

Concatenating train & test set

```
train_test = pd.concat([data_w2,test], axis=0, sort=False)
```

Looking at NaN % within the data

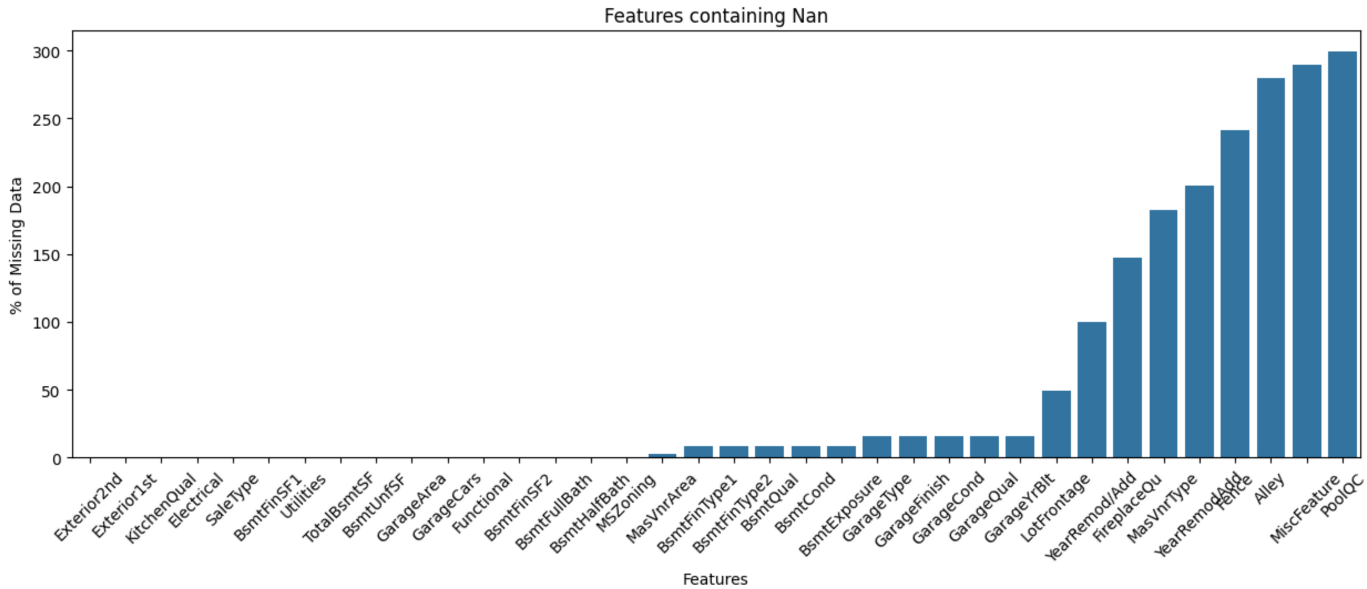
```
nan = pd.DataFrame(train_test.isna().sum(), columns = ['NaN_sum'])
nan['feat'] = nan.index
nan['Perc(%)'] = (nan['NaN_sum']/1460)*100
nan = nan[nan['NaN_sum'] > 0]
nan = nan.sort_values(by = ['NaN_sum'])
nan['Usability'] = np.where(nan['Perc(%)'] > 20, 'Discard', 'Keep')
nan
```

	NaN_sum	feat	Perc(%)	Usability
Exterior2nd	1	Exterior2nd	0.068493	Keep
Exterior1st	1	Exterior1st	0.068493	Keep
KitchenQual	1	KitchenQual	0.068493	Keep
Electrical	1	Electrical	0.068493	Keep
SaleType	1	SaleType	0.068493	Keep
BsmtFinSF1	2	BsmtFinSF1	0.136986	Keep
Utilities	2	Utilities	0.136986	Keep
TotalBsmtSF	2	TotalBsmtSF	0.136986	Keep
BsmtUnfSF	2	BsmtUnfSF	0.136986	Keep
GarageArea	2	GarageArea	0.136986	Keep
GarageCars	2	GarageCars	0.136986	Keep
Functional	2	Functional	0.136986	Keep
BsmtFinSF2	2	BsmtFinSF2	0.136986	Keep
BsmtFullBath	4	BsmtFullBath	0.273973	Keep
BsmtHalfBath	4	BsmtHalfBath	0.273973	Keep
MSZoning	4	MSZoning	0.273973	Keep
MasVnrArea	38	MasVnrArea	2.602740	Keep
BsmtFinType1	122	BsmtFinType1	8.356164	Keep
BsmtFinType2	123	BsmtFinType2	8.424658	Keep
BsmtQual	124	BsmtQual	8.493151	Keep
BsmtCond	125	BsmtCond	8.561644	Keep
BsmtExposure	127	BsmtExposure	8.698630	Keep
GarageType	233	GarageType	15.958904	Keep
GarageFinish	237	GarageFinish	16.232877	Keep
GarageCond	237	GarageCond	16.232877	Keep
GarageQual	237	GarageQual	16.232877	Keep
GarageYrBlt	237	GarageYrBlt	16.232877	Keep
LotFrontage	717	LotFrontage	49.109589	Discard
YearRemod/Add	1459	YearRemod/Add	99.931507	Discard
FireplaceQu	2152	FireplaceQu	147.397260	Discard
MasVnrType	2669	MasVnrType	182.808219	Discard
YearRemodAdd	2930	YearRemodAdd	200.684932	Discard
Fence	3527	Fence	241.575342	Discard
Alley	4084	Alley	279.726027	Discard
MiscFeature	4232	MiscFeature	289.863014	Discard
PoolQC	4373	PoolQC	299.520548	Discard

Next steps: Generate code with nan View recommended plots New interactive sheet

Plotting Nan

```
plt.figure(figsize = (15,5))
sns.barplot(x = nan['feat'], y = nan['Perc(%)'])
plt.xticks(rotation=45)
plt.title('Features containing Nan')
plt.xlabel('Features')
plt.ylabel('% of Missing Data')
plt.show()
```



Converting non-numeric predictors stored as numbers into string

```
train_test['MSSubClass'] = train_test['MSSubClass'].apply(str)
train_test['YrSold'] = train_test['YrSold'].apply(str)
train_test['MoSold'] = train_test['MoSold'].apply(str)
```

Filling Categorical NaN (That we know how to fill due to the description file)

```
train_test['Functional'] = train_test['Functional'].fillna('Typ')
train_test['Electrical'] = train_test['Electrical'].fillna("SBrkr")
train_test['KitchenQual'] = train_test['KitchenQual'].fillna("TA")
train_test['Exterior1st'] = train_test['Exterior1st'].fillna(train_test['Exterior1st'].mode()[0])
train_test['Exterior2nd'] = train_test['Exterior2nd'].fillna(train_test['Exterior2nd'].mode()[0])
train_test['SaleType'] = train_test['SaleType'].fillna(train_test['SaleType'].mode()[0])
train_test["PoolQC"] = train_test["PoolQC"].fillna("None")
train_test["Alley"] = train_test["Alley"].fillna("None")
train_test['FireplaceQu'] = train_test['FireplaceQu'].fillna("None")
train_test['Fence'] = train_test['Fence'].fillna("None")
train_test['MiscFeature'] = train_test['MiscFeature'].fillna("None")

for col in ('GarageArea', 'GarageCars'):
    train_test[col] = train_test[col].fillna(0)

for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:
    train_test[col] = train_test[col].fillna('None')

for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    train_test[col] = train_test[col].fillna('None')

    # Checking the features with NaN remained out

for col in train_test:
    if train_test[col].isna().sum() > 0:
```

```
        print(train_test[col][0])
```

```
0    RL
0    RH
Name: MSZoning, dtype: object
0    141.0
0     80.0
Name: LotFrontage, dtype: float64
0    AllPub
0    AllPub
Name: Utilities, dtype: object
0    1960.0
0       NaN
Name: YearRemod/Add, dtype: float64
0    Stone
0      NaN
Name: MasVnrType, dtype: object
0    112.0
0      0.0
Name: MasVnrArea, dtype: float64
0    639.0
0    468.0
Name: BsmtFinSF1, dtype: float64
0      0.0
0    144.0
Name: BsmtFinSF2, dtype: float64
0    441.0
0    270.0
Name: BsmtUnfSF, dtype: float64
0    1080.0
0     882.0
Name: TotalBsmtSF, dtype: float64
0    1.0
0    0.0
Name: BsmtFullBath, dtype: float64
0    0.0
0    0.0
Name: BsmtHalfBath, dtype: float64
0    1960.0
0    1961.0
Name: GarageYrBlt, dtype: float64
0       NaN
0    1961.0
Name: YearRemodAdd, dtype: float64
```

Removing the useless variables

```
useless = ['GarageYrBlt','YearRemodAdd']
train_test = train_test.drop(useless, axis = 1)
```

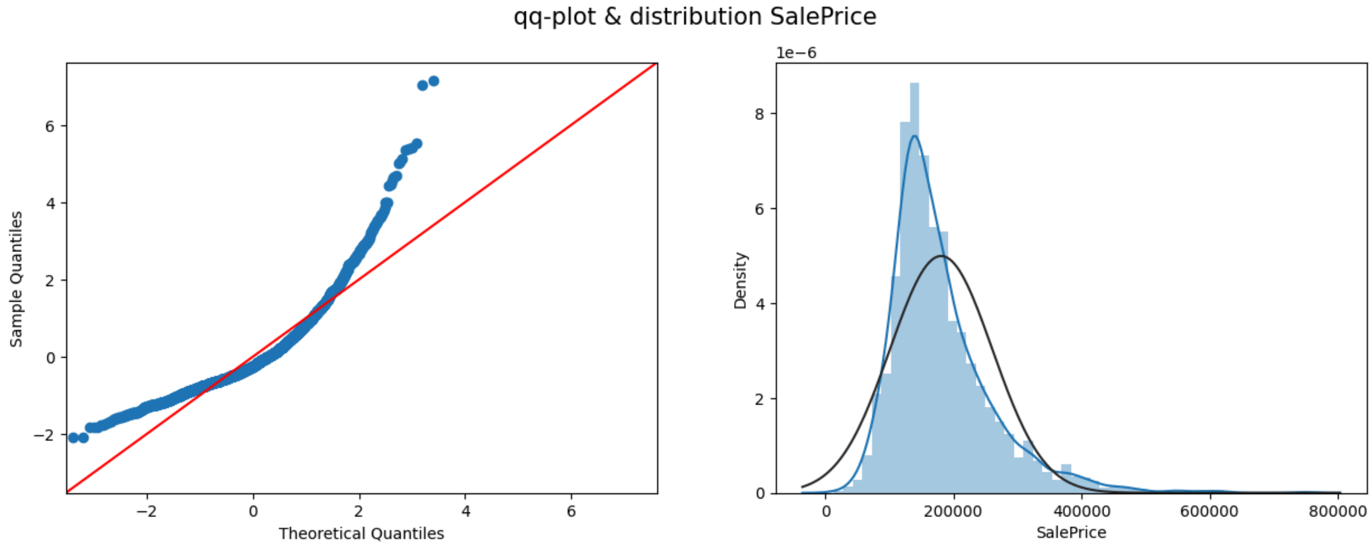
Imputing with KnnRegressor (we can also use different Imputers)

```
def impute_knn(df):
    ttn = train_test.select_dtypes(include=[np.number])
    ttc = train_test.select_dtypes(exclude=[np.number])

    cols_nan = ttn.columns[ttn.isna().any()].tolist()         # columns w/ nan
    cols_no_nan = ttn.columns.difference(cols_nan).values     # columns w/n nan

    for col in cols_nan:
        imp_test = ttn[ttn[col].isna()]   # indicies which have missing data will become our test set
        imp_train = ttn.dropna()          # all indicies which which have no missing data
        model = KNeighborsRegressor(n_neighbors=5)  # KNR Unsupervised Approach
        knr = model.fit(imp_train[cols_no_nan], imp_train[col])
        ttn.loc[ttn[col].isna(), col] = knr.predict(imp_test[cols_no_nan])

    return pd.concat([ttn,ttc],axis=1)

train_test = impute_knn(train_test)
```

```
objects = []
for i in train_test.columns:
    if train_test[i].dtype == object:
        objects.append(i)
train_test.update(train_test[objects].fillna('None'))

# # Checking NaN presence

for col in train_test:
    if train_test[col].isna().sum() > 0:
        print(train_test[col][0])


# First part remains the same
train_test["SqFtPerRoom"] = train_test["GrLivArea"] / (train_test["TotRmsAbvGrd"] + train_test["FullBath"] + tra
train_test['Total_Home_Quality'] = train_test['OverallQual'] + train_test['OverallCond']
train_test['Total_Bathrooms'] = (train_test['FullBath'] + (0.5 * train_test['HalfBath']) + train_test['BsmtFullB
train_test["HighQualSF"] = train_test["1stFlrSF"] + train_test["2ndFlrSF"]

# Converting non-numeric predictors stored as numbers into string
train_test['MSSubClass'] = train_test['MSSubClass'].apply(str)
train_test['YrSold'] = train_test['YrSold'].apply(str)
train_test['MoSold'] = train_test['MoSold'].apply(str)

# Creating dummy variables from categorical features
train_test_dummy = pd.get_dummies(train_test)

# Fetch all numeric features
# Filter to only include float and int columns (exclude bool columns)
numeric_features = train_test_dummy.select_dtypes(include=['float64', 'int64']).columns

# Compute skewness only on non-boolean numeric features
skewed_features = train_test_dummy[numeric_features].apply(lambda x: skew(x)).sort_values(ascending=False)
high_skew = skewed_features[skewed_features > 0.5]
skew_index = high_skew.index

# Normalize skewed features using log_transformation
for i in skew_index:
    train_test_dummy[i] = np.log1p(train_test_dummy[i])
```

SalePrice before transformation

```
fig, ax = plt.subplots(1,2, figsize= (15,5))
fig.suptitle(" qq-plot & distribution SalePrice ", fontsize= 15)

sm.qqplot(target, stats.t, distargs=(4,),fit=True, line="45", ax = ax[0])

sns.distplot(target, kde = True, hist=True, fit = norm, ax = ax[1])
plt.show()
```



SalePrice after transformation

```
target_log = np.log1p(target)

fig, ax = plt.subplots(1,2, figsize= (15,5))
fig.suptitle("qq-plot & distribution SalePrice ", fontsize= 15)

sm.qqplot(target_log, stats.t, distargs=(4,),fit=True, line="45", ax = ax[0])
sns.distplot(target_log, kde = True, hist=True, fit = norm, ax = ax[1])
plt.show()
```

```
import shap
import xgboost as xgb
from catboost import Pool
from sklearn.svm import SVR
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeRegressor
from mlxtend.regressor import StackingRegressor
from sklearn.linear_model import LinearRegression, BayesianRidge
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_squared_log_error


# Train-Test separation

train = train_test_dummy[0:2930]
test = train_test_dummy[2930:]
test['Id'] = test_id

# Creation of the RMSE metric:

def rmse(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))

def cv_rmse(model):
    rmse = np.sqrt(-cross_val_score(model, train, target_log, scoring="neg_mean_squared_error", cv=kf))
    return (rmse)
```

5 Fold Cross validation

```
kf = KFold(n_splits=5, random_state=42, shuffle=True)

cv_scores = []
cv_std = []

baseline_models = ['Linear_Reg.','Bayesian_Ridge_Reg.','LGBM_Reg.','SVR',
                   'Dec_Tree_Reg.','Random_Forest_Reg.', 'XGB_Reg.',
                   'Grad_Boost_Reg.','Cat_Boost_Reg.','Stacked_Reg.']

# Linear Regression

lreg = LinearRegression()
score_lreg = cv_rmse(lreg)
cv_scores.append(score_lreg.mean())
cv_std.append(score_lreg.std())

# Bayesian Ridge Regression

brr = BayesianRidge(compute_score=True)
score_brr = cv_rmse(brr)
cv_scores.append(score_brr.mean())
cv_std.append(score_brr.std())

# Light Gradient Boost Regressor

l_gbm = LGBMRegressor(objective='regression')
score_l_gbm = cv_rmse(l_gbm)
cv_scores.append(score_l_gbm.mean())
cv_std.append(score_l_gbm.std())

# Support Vector Regression
```

```
svr = SVR()
score_svr = cv_rmse(svr)
cv_scores.append(score_svr.mean())
cv_std.append(score_svr.std())

# Decision Tree Regressor

dtr = DecisionTreeRegressor()
score_dtr = cv_rmse(dtr)
cv_scores.append(score_dtr.mean())
cv_std.append(score_dtr.std())

# Random Forest Regressor

rfr = RandomForestRegressor()
score_rfr = cv_rmse(rfr)
cv_scores.append(score_rfr.mean())
cv_std.append(score_rfr.std())

# XGB Regressor

xgb = xgb.XGBRegressor()
score_xgb = cv_rmse(xgb)
cv_scores.append(score_xgb.mean())
cv_std.append(score_xgb.std())

# Gradient Boost Regressor

gbr = GradientBoostingRegressor()
score_gbr = cv_rmse(gbr)
cv_scores.append(score_gbr.mean())
cv_std.append(score_gbr.std())

# Cat Boost Regressor

catb = CatBoostRegressor()
score_catb = cv_rmse(catb)
cv_scores.append(score_catb.mean())
cv_std.append(score_catb.std())

# Stacked Regressor

stack_gen = StackingRegressor(regressors=(CatBoostRegressor(),
                                          LinearRegression(),
                                          BayesianRidge(),
                                          GradientBoostingRegressor()),
                              meta_regressor = CatBoostRegressor(),
                              use_features_in_secondary = True)

score_stack_gen = cv_rmse(stack_gen)
cv_scores.append(score_stack_gen.mean())
cv_std.append(score_stack_gen.std())

final_cv_score = pd.DataFrame(baseline_models, columns = ['Regressors'])
final_cv_score['RMSE_mean'] = cv_scores
final_cv_score['RMSE_std'] = cv_std
```

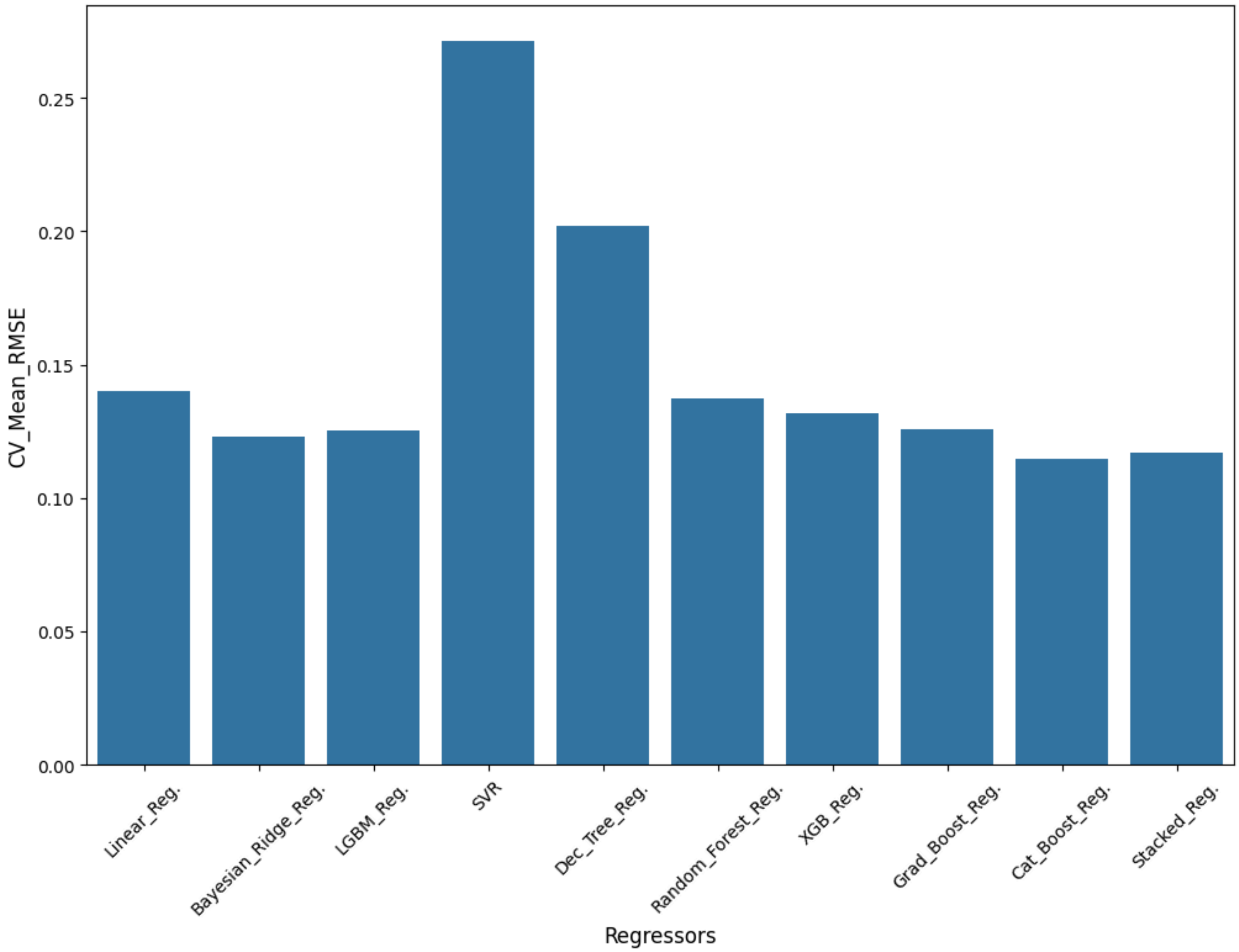
```
953:	learn: 0.0254443	total: 12.4s	remaining: 596ms
954:	learn: 0.0254164	total: 12.4s	remaining: 583ms
955:	learn: 0.0253852	total: 12.4s	remaining: 570ms
956:	learn: 0.0253666	total: 12.4s	remaining: 557ms
957:	learn: 0.0253387	total: 12.4s	remaining: 543ms
958:	learn: 0.0253190	total: 12.4s	remaining: 530ms
959:	learn: 0.0253045	total: 12.4s	remaining: 517ms
960:	learn: 0.0252954	total: 12.4s	remaining: 504ms
961:	learn: 0.0252800	total: 12.4s	remaining: 491ms
962:	learn: 0.0252531	total: 12.4s	remaining: 478ms
963:	learn: 0.0252353	total: 12.4s	remaining: 465ms
964:	learn: 0.0252120	total: 12.5s	remaining: 452ms
965:	learn: 0.0251859	total: 12.5s	remaining: 439ms
966:	learn: 0.0251743	total: 12.5s	remaining: 426ms
967:	learn: 0.0251576	total: 12.5s	remaining: 413ms
968:	learn: 0.0251370	total: 12.5s	remaining: 400ms
969:	learn: 0.0251305	total: 12.5s	remaining: 386ms
970:	learn: 0.0251116	total: 12.5s	remaining: 373ms
971:	learn: 0.0250940	total: 12.5s	remaining: 360ms
972:	learn: 0.0250618	total: 12.5s	remaining: 347ms
973:	learn: 0.0250446	total: 12.5s	remaining: 334ms
974:	learn: 0.0250193	total: 12.5s	remaining: 321ms
975:	learn: 0.0250038	total: 12.6s	remaining: 309ms
976:	learn: 0.0249884	total: 12.6s	remaining: 296ms
977:	learn: 0.0249617	total: 12.6s	remaining: 283ms
978:	learn: 0.0249523	total: 12.6s	remaining: 270ms
979:	learn: 0.0249360	total: 12.6s	remaining: 257ms
980:	learn: 0.0249172	total: 12.6s	remaining: 244ms
981:	learn: 0.0249002	total: 12.6s	remaining: 231ms
982:	learn: 0.0248878	total: 12.6s	remaining: 218ms
983:	learn: 0.0248703	total: 12.6s	remaining: 205ms
984:	learn: 0.0248529	total: 12.6s	remaining: 192ms
985:	learn: 0.0248167	total: 12.6s	remaining: 180ms
986:	learn: 0.0248024	total: 12.7s	remaining: 167ms
987:	learn: 0.0247763	total: 12.7s	remaining: 154ms
988:	learn: 0.0247606	total: 12.7s	remaining: 141ms
989:	learn: 0.0247387	total: 12.7s	remaining: 128ms
990:	learn: 0.0247225	total: 12.7s	remaining: 115ms
991:	learn: 0.0247018	total: 12.7s	remaining: 102ms
992:	learn: 0.0246867	total: 12.7s	remaining: 89.5ms
993:	learn: 0.0246771	total: 12.7s	remaining: 76.7ms
994:	learn: 0.0246728	total: 12.7s	remaining: 63.9ms
995:	learn: 0.0246501	total: 12.7s	remaining: 51.1ms
996:	learn: 0.0246322	total: 12.7s	remaining: 38.3ms
997:	learn: 0.0246163	total: 12.7s	remaining: 25.6ms
998:	learn: 0.0246053	total: 12.8s	remaining: 12.8ms
999:	learn: 0.0245812	total: 12.8s	remaining: 0us
```

```
final_cv_score
```

	Regressors	RMSE_mean	RMSE_std
0	Linear_Reg.	0.139906	0.008774
1	Bayesian_Ridge_Reg.	0.122801	0.012918
2	LGBM_Reg.	0.125197	0.011304
3	SVR	0.271176	0.014806
4	Dec_Tree_Reg.	0.202059	0.009835
5	Random_Forest_Reg.	0.137360	0.012072
6	XGB_Reg.	0.131961	0.011776
7	Grad_Boost_Reg.	0.125653	0.011558
8	Cat_Boost_Reg.	0.114826	0.015019
9	Stacked_Reg.	0.117172	0.015080

Next steps: Generate code with final_cv_score | View recommended plots | New interactive sheet

```
plt.figure(figsize=(12, 8))
sns.barplot(x='Regressors', y='RMSE_mean', data=final_cv_score) # Pass data as a single argument
plt.xlabel('Regressors', fontsize=12)
plt.ylabel('CV_Mean_RMSE', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```




```
# Train-Test split the data

X_train,X_val,y_train,y_val = train_test_split(train,target_log,test_size = 0.1,random_state=42)

# Cat Boost Regressor

cat = CatBoostRegressor()
cat_model = cat.fit(X_train,y_train,
                    eval_set = (X_val,y_val),
                    plot=True,
                    verbose = 0)
```

```
from google.colab import output
output.enable_custom_widget_manager()
```

```
cat_pred = cat_model.predict(X_val)
cat_score = rmse(y_val, cat_pred)
cat_score
```

```
np.float64(0.09606694133394318)
```

```
# Features' importance of the model

feat_imp = cat_model.get_feature_importance(prettified=True)
feat_imp
```

	Feature Id	Importances
0	OverallQual	17.691018
1	GrLivArea	7.565579
2	Total_Home_Quality	5.756266
3	HighQualSF	5.408964
4	TotalBsmtSF	4.852915
...	...	...
349	PoolQC_TA	0.000000
350	Fence_MnWw	0.000000
351	MiscFeature_TenC	0.000000