Advanced Regression Assignment (Housing data analysis)

Reading and Understanding the Data

In [1]:
```
# Import all the required libraries

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
# Import the data

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

In [3]:
```
# Check the shape

data.shape
```

In [4]:
```
# Check data-info and other info regarding the null data

data.info()
```

Observation

-
-

In [5]:
```
data.isna().sum()
```

In [6]:
```
null_data_columns = data.columns[data.isna().any()]
```

In [7]:
```
#from pprint import pprint
for c in null_data_columns:
    print(c, data[c].isnull().sum())
```

In [11]:
```
# Lets understand the behaviour of some categorical data
```

In [8]:
```
data_null_percentage = data.apply(lambda x: round(x.isnull().mean()* 100, 2))
data_null_percentage[data_null_percentage>0.0]
fig=data.plot.box(title='Outliers', figsize=(80, 40), legend=True, fontsize=24
fig.axes.title.set_size(40)
```

In [9]:
```
data_null_above_thresold = data_null_percentage[data_null_percentage>40.0]
null_above_thresold_columns = data_null_above_thresold.index
f"{len(null_above_thresold_columns)} columns out of {len(data.columns)} column
```

Observation

variable, SalePrice. LotArea has also some outliers

In [10]:
```
data.drop(null_above_thresold_columns, axis=1, inplace= True)
data
```

In [12]:
```
    if feature=='class(target)':
        pass
    else:
        sns.histplot(x=feature, data=data)
        plt.xlabel(feature)
        plt.show()
```

In [11]:
```
# Lets understand the behaviour of some categorical data
```

In [8]:
```
data_null_percentage = data.apply(lambda x: round(x.isnull().mean()* 100, 2))
data_null_percentage[data_null_percentage>0.0]
fig=data.plot.box(title='Outliers', figsize=(80, 40), legend=True, fontsize=24
fig.axes.title.set_size(40)
```

In [9]:
```
data_null_above_thresold = data_null_percentage[data_null_percentage>40.0]
null_above_thresold_columns = data_null_above_thresold.index
f"{len(null_above_thresold_columns)} columns out of {len(data.columns)} column
```

Observation

variable, SalePrice. LotArea has also some outliers

In [10]:
```
data.drop(null_above_thresold_columns, axis=1, inplace= True)
data
```

In [12]:
```
for feature in numerical_features:
    if feature=='class(target)':
        pass
    else:
        sns.histplot(x=feature, data=data)
        plt.xlabel(feature)
        plt.show()
```

Observation

- The histograms above describe the skewness of the data. They also suggest that `LowQualFinSF`, `PoolArea`, `MiscVal`, and `3SsnPorch` contain very little variety in values.

- From a business logic standpoint, `PoolArea` is a similar variable to the previously dropped one. This is also the case for `MiscVal` - in additon, it has a rather high number of outliers. `3SsnPorchappears` to be contained in the other porch values. They are dropped as a result.

In [13]:
```
# Let's drop above features

data = data.drop(['LowQualFinSF', 'PoolArea', 'MiscVal', '3SsnPorch'], axis=1)
```

In [14]:
```
data.shape
```

Missing value treatment

In [15]:
```
# function to check for the missing value

def get_missing_counts(data: pd.DataFrame, columns: list):
    for column in columns:
        print(column, data[column].isnull().sum())
```

In [16]:
```
columns_with_missing_data =  data.columns[data.isnull().any()]

get_missing_counts(data=data,
                   columns=columns_with_missing_data)
```

In [20]:
```
data.info()
```

In [21]:
```
data.describe()
```

In [17]:
```
# Imputing the values with median

data['LotFrontage'] = data['LotFrontage'].fillna(data['LotFrontage'].median())
data['GarageYrBlt'] = data['GarageYrBlt'].fillna(data['GarageYrBlt'].median())
data['MasVnrArea'] = data['MasVnrArea'].fillna(data['MasVnrArea'].median())
```

In [22]:
```
# Outlier removal
```

In [23]:
```
#(house[cols] < (Q1 - 1.5 * IQR)) |(house[cols] > (Q3 + 1.5 * IQR))).any(
```

In [18]:
```
# Imputing the values with mode

for column in ['GarageCond', 'GarageType', 'GarageFinish','GarageQual','BsmtEx
    data[column] = data[column].fillna(data[column].mode()[0])
```

EDA

In [24]:
```
data.head()
```

In [25]:
```
# Drop `Id`as it has no significance further in the dataset
data = data.drop(['Id'], axis=1)
```

In [19]:
```
data.isnull().values.any()
```

In [26]:
```
data.head()
```

```
                         columns=columns_with_missing_data)
```

In [20]:
```
data.info()
```

In [21]:
```
data.describe()
```

In [17]:
```
# Imputing the values with median

data['LotFrontage'] = data['LotFrontage'].fillna(data['LotFrontage'].median())
data['GarageYrBlt'] = data['GarageYrBlt'].fillna(data['GarageYrBlt'].median())
data['MasVnrArea'] = data['MasVnrArea'].fillna(data['MasVnrArea'].median())
```

In [22]:
```
# Outlier removal
```

In [23]:
```
#(house[cols] < (Q1 - 1.5 * IQR)) |(house[cols] > (Q3 + 1.5 * IQR))).any(
```

In [18]:
```
# Imputing the values with mode

for column in ['GarageCond', 'GarageType', 'GarageFinish','GarageQual','BsmtEx
    data[column] = data[column].fillna(data[column].mode()[0])
```

EDA

In [24]:
```
data.head()
```

In [25]:
```
# Drop `Id`as it has no significance further in the dataset
data = data.drop(['Id'], axis=1)
```

In [19]:
```
data.isnull().values.any()
```

In [26]:
```
data.head()
```

Visualising the Data

Let's now spend some time doing what is arguably the most important step - **understanding the data**.

- If there is some obvious multicollinearity going on, this is the first place to catch it
- Here's where you'll also identify if some predictors directly have a strong association with the outcome variable

We'll visualise our data using `matplotlib` and `seaborn`

In [27]:
```
# Lets see pair plot to understand the behaviour of one feature w.r.t to other

plt.figure(figsize=(15,10))
sns.pairplot(data, x_vars=['MSSubClass','LotFrontage','LotArea'], y_vars='Sale
sns.pairplot(data, x_vars=['OverallQual', 'OverallCond','MasVnrArea'], y_vars=
sns.pairplot(data, x_vars=['BsmtFinSF1', 'BsmtUnfSF','TotalBsmtSF'], y_vars='S
sns.pairplot(data, x_vars=['1stFlrSF','2ndFlrSF', 'GrLivArea'], y_vars='SalePr
sns.pairplot(data, x_vars=['BsmtFullBath','FullBath', 'HalfBath'], y_vars='Sal
sns.pairplot(data, x_vars=['BedroomAbvGr','TotRmsAbvGrd', 'Fireplaces'], y_var
sns.pairplot(data, x_vars=['GarageCars','GarageArea', 'WoodDeckSF'], y_vars='S
plt.show()
```

Observation

- The above pair plots shows that the features like `GarageArea`, `1stFlrSF` and `GrLivArea` show very good correlation with the `SalePrice`.

In [28]:
```
# all numeric (float and int) variables in the dataset
data_numeric = data.select_dtypes(include=['float64', 'int64'])

#Finding correlation matric
corr_matrix = data_numeric.corr()
plt.figure(figsize=(20,20))
#plotting correlattion matric on a heat map
ax = sns.heatmap(corr_matrix, annot = True, cmap='BuPu_r')
top, bottom = ax.get_ylim()
ax.set_ylim(top+0.5, bottom-0.5)
plt.title("Correlation between Columns")
plt.show()
```

Some independent variables are highly correlated with each other. This has to be considered because of multicollinearity that may become an issue in the model.

- `YearBuilt`, `GarageYrBlt` -> highly correlated
- `TotRmsAbvGrd`, `GrLivArea` -> highly correlated
- `GarageArea`, `GarageCars` -> highly correlated
- `1stFlrSF`, `TotalBsmtSF` -> highly correlated

In [29]:
```
#Drop highly correlated features
data.drop(['1stFlrSF', 'TotRmsAbvGrd', 'GarageArea', 'GarageYrBlt'], axis=1, inp
```

Dropping multi-collinear data

Insights from the heatmap: Correlation of sale price with independent variables:

- Sale price is highly positively correlated with OverallQual, GrLivArea
- Sale price is positively correlated with `TotalBsmtSF`, `1stFlrSF`, `FullBath`, `TotRmsAbvGrd`, `GarageCars`, `GarageArea`
- Sale price is not highly negatively correlated with other variables.

Create Dummy variable

In [30]:
```
# Create a function to get the dummy variable dataframe

def get_dummy_dataframe(column_name: list[str]):
    output = pd.DataFrame()
```

In [28]:
```
# all numeric (float and int) variables in the dataset
data_numeric = data.select_dtypes(include=['float64', 'int64'])
#Finding correlation matric
corr_matrix = data_numeric.corr()
plt.figure(figsize=(20,20))
#plotting correlatoon matric on a heat map
ax = sns.heatmap(corr_matrix, annot = True, cmap='BuPu_r')
top, bottom = ax.get_ylim()
ax.set_ylim(top+0.5, bottom-0.5)
plt.title("Correlation between Columns")
plt.show()
```

Some independent variables are highly correlated with each other. This has to be considered because of multicollinearity that may become an issue in the model.

- YearBuilt , GarageYrBlt -> highly correlated
- TotRmsAbvGrd , GrLivArea -> highly correlated
- GarageArea , GarageCars -> highly correlated
- 1stFlrSF , TotalBsmtSF -> highly correlated

Dropping multi-collinear data

In [29]:
```
#Drop highly correlated features
data.drop(['1stFlrSF','TotRmsAbvGrd','GarageArea','GarageYrBlt'], axis=1, inp
```

Insights from the heatmap: Correlation of sale price with independent variables:

- Sale price is highly positively correlated with OverallQual, GrLivArea
- Sale price is positively correlated with TotalBsmtSF , 1stFlrSF , FullBath , TotRmsAbvGrd , GarageCars , GarageArea
- Sale price is not highly negatively correlated with other variables.

Create Dummy variable

In [30]:
```
# Create a function to get the dummy variable dataframe
def get_dummy_dataframe(column_name: list[str]):
    output = pd.DataFrame()
    for column in column_name:
        status = pd.get_dummies(data[column], drop_first=True)
        output = pd.concat([output, status], axis=1)  # Concatenate the status
    return output
```

In [31]:
```
#Check which columns containg categorical data
data_categorical = data.select_dtypes(include=['object'])
data_categorical.head()
```

In [32]:
```
data
```

Data Scaling

In [33]:
```
# drop categorical variables from the dataset and save as predictor variable X
X= data.drop(list(data_categorical.columns), axis=1)
X=X.drop(['SalePrice'], axis=1)
y = data['SalePrice']
```

In [34]:
```
from sklearn.preprocessing import scale

cols = X.columns
X = pd.DataFrame(scale(X))
X.columns = cols
X.columns
```

Train test split

In [35]:
```
# split into train and test
X_train, X_test, y_train,  y_test = train_test_split(X, y,
                                                    train_size=0.70,
                                                    test_size = 0.30, random_s
```

In [36]:
```
# Instantiate linear regression
lm = LinearRegression()

# Fit a line
lm.fit(X_train, y_train)
```

In [37]:
```
print(lm.intercept_)
print(lm.coef_)
```

In [38]:
```
y_pred_train = lm.predict(X_train)
y_pred_test = lm.predict(X_test)

metric = []
r2_train_lr = r2_score(y_train, y_pred_train)
print('r2_train: ', r2_train_lr)
metric.append(r2_train_lr)

print('r2 test: ',r2_test_lr)
metric.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print('rss1: ',rss1_lr)
metric.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print('rss2: ',rss2_lr)
metric.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
```

Train test split

In [37]:
```
print(lm.intercept_)
print(lm.coef_)
```

In [35]:
```
# split into train and test
X_train, X_test, y_train,  y_test = train_test_split(X, y,
                                                    train_size=0.70,
                                                    test_size = 0.30, random_s
```

In [36]:
```
# Instantiate linear regression
lm = LinearRegression()

# Fit a line
lm.fit(X_train, y_train)
```

In [38]:
```
y_pred_train = lm.predict(X_train)
y_pred_test = lm.predict(X_test)

metric = []
r2_train_lr = r2_score(y_train, y_pred_train)
print('r2 train: ', r2_train_lr)
metric.append(r2_train_lr)

print('r2 test: ',r2_test_lr)
metric.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print('rss1: ',rss1_lr)
metric.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print('rss2: ',rss2_lr)
metric.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print('MSE train: ',mse_train_lr)
metric.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print('MSE test: ',mse_test_lr)
metric.append(mse_test_lr**0.5)
```

Ridge and Lasso implementation

In [39]:
```
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}


ridge = Ridge()

# cross validation
folds = 4
ridge_model_cv = GridSearchCV(estimator = ridge,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)
ridge_model_cv.fit(X_train, y_train)
```

In [40]:
```
# Printing the best hyperparameter alpha
print(ridge_model_cv.best_params_)
print(ridge_model_cv.best_score_)
```

In [41]:
```
alpha = 100
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)
ridge.coef_
```

In [42]:
```
# Lets calculate some metrics such as R2 score, RSS and RMSE
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)

metric2 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print(r2_train_lr)
metric2.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(r2_test_lr)
metric2.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(rss1_lr)
metric2.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(rss2_lr)
metric2.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(mse_train_lr)
metric2.append(mse_train_lr**0.5)
```

In [41]:
```python
alpha = 100
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)
ridge.coef_
```

In [42]:
```python
# Lets calculate some metrics such as R2 score, RSS and RMSE
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)

metric2 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print(r2_train_lr)
metric2.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(r2_test_lr)
metric2.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(rss1_lr)
metric2.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(rss2_lr)
metric2.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(mse_train_lr)
metric2.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print(mse_test_lr)
metric2.append(mse_test_lr**0.5)
```

In [43]:
```python
## Lasso Implementation
```

In [44]:
```python
lasso = Lasso()

# cross validation
lasso_model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

lasso_model_cv.fit(X_train, y_train)
```

In [45]:
```python
print(lasso_model_cv.best_params_)
print(lasso_model_cv.best_score_)
```

In [46]:
```python
lpha =0.001

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
```

In [47]:
```python
lasso.coef_
```

In [48]:
```python
predictors = X_train.columns

coef = pd.Series(lasso.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Modal Coefficients', fontsize='16',figsize=(80, 6
```

In [49]:
```python
# Lets calculate some metrics such as R2 score, RSS and RMSE
y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)

metric3 = []
print(r2_train_lr)
metric3.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(r2_test_lr)
metric3.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(rss1_lr)
metric3.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(rss2_lr)
metric3.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(mse_train_lr)
metric3.append(mse_train_lr**0.5)
```

In [48]:
```
predictors = X_train.columns

coef = pd.Series(lasso.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Modal Coefficients', fontsize='16',figsize=(80, 6
```

In [49]:
```
# Lets calculate some metrics such as R2 score, RSS and RMSE

y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)

metric3 = []
print(r2_train_lr)
metric3.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(r2_test_lr)
metric3.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(rss1_lr)
metric3.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(rss2_lr)
metric3.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(mse_train_lr)
metric3.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print(mse_test_lr)
metric3.append(mse_test_lr**0.5)
```

In [50]:
```
# Creating a table which contain all the metrics

lr_table = {'Metric': ['R2 Score (Train)','R2 Score (Test)','RSS (Train)','RSS
                       'MSE (Train)','MSE (Test)'],
        'Linear Regression': metric
        }

lr_metric = pd.DataFrame(lr_table ,columns = ['Metric', 'Linear Regression'] )

rg_metric = pd.Series(metric2, name = 'Ridge Regression')
ls_metric = pd.Series(metric3, name = 'Lasso Regression')

final_metric = pd.concat([lr_metric, rg_metric, ls_metric], axis = 1)

final_metric
```

Predictions

In [51]:
```
ridge_pred = ridge.predict(X_test)
```

In [52]:
```
# Plotting y_test and y_pred to understand the spread for ridge regression.
fig = plt.figure(dpi=100)
plt.scatter(y_test,ridge_pred)
fig.suptitle('y_test vs ridge_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('ridge_pred', fontsize=16)
plt.show()
```

In [55]:
```
# Plotting y_test and y_pred to understand the spread for lasso regression.
fig = plt.figure(dpi=100)
fig.suptitle('y_test vs lasso_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('lasso_pred', fontsize=16)
plt.show()
```

In [53]:
```
y_res=y_test-ridge_pred
# Distribution of errors
sns.distplot(y_res,kde=True)
plt.xlabel('Residuals')
plt.show()
```

In [56]:
```
y_res=y_test-lasso_pred
# Distribution of errors
sns.distplot(y_res,kde=True)
plt.title('Normality of error terms/residuals Lasso')
plt.xlabel("Residuals")
plt.show()
```

In [54]:
```
lasso_pred = lasso.predict(X_test)
```

Coefficients

In [57]:
```
betas = pd.DataFrame(index=X_train.columns)
```

In [58]:
```
betas.rows = X_train.columns
```

```
plt.ylabel('ridge_pred', fontsize=16)
plt.show()
```

In [55]:
```
# Plotting y_test and y_pred to understand the spread for lasso regression.
fig = plt.figure(dpi=100)
```

```
fig.suptitle('y_test vs lasso_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('lasso_pred', fontsize=16)
plt.show()
```

In [53]:
```
y_res=y_test-ridge_pred
# Distribution of errors
sns.distplot(y_res,kde=True)
```

```
plt.xlabel('Residuals')
plt.show()
```

In [56]:
```
y_res=y_test-lasso_pred
# Distribution of errors
sns.distplot(y_res,kde=True)
```

In [54]:
```
lasso_pred = lasso.predict(X_test)
plt.title('Normality of error terms/residuals Lasso')
plt.xlabel("Residuals")
plt.show()
```

Coefficients

In [57]:
```
betas = pd.DataFrame(index=X_train.columns)
```

In [58]:
```
betas.rows = X_train.columns
```

In [59]:
```
betas['Linear'] = lm.coef_
betas['Ridge'] = ridge.coef_
betas['Lasso'] = lasso.coef_
```

In [60]:
```
betas.head(68)
```

In [61]:
```
betas = pd.DataFrame(index=X_train.columns)
betas.rows = X_train.columns
betas['Lasso'] = lasso.coef_
betas.sort_values(by=['Lasso'], ascending=False)
```

Evalution of the model

The model shows that there are some variables that are highly relevant to the sales price. Suggestions for Surprise Housing is to keep a check on these predictors affecting the price of the house. The higher values of positive coeeficients suggest a high sale value. Some of those features are:

Feature Description `GrLivArea` Above grade (ground) living area square feet `OverallQual` Rates the overall material and finish of the house `OverallCond` Rates the overall condition of the house `TotalBsmtSF` Total square feet of basement area `GarageArea` Size of garage in square feet