I am doing the House Prices - Advanced Regression Techniques assignment and for this assignment I used these two notebooks (https://www.kaggle.com/code/ashvanths/complete-eda-and-feature-engineering and https://www.kaggle.com/code/emmanueldjegou/house-prices-advanced-regression-techniques)

In [114…

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

First starting with importing the necessary packages and import our data from the csv. From there we will look at the basic summary of the data to see how many rows and what kind of data types there are

In [115…

```
df=pd.read_csv('train.csv')
```

In [116…

```
valid = pd.read_csv('test.csv')
valid
```

Out[116…

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utili
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	All
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	All
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	All
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	All
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	All
...	...	...	...	...	...	...	...	...	...	
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	All
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	All
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	All
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	All
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	All

1459 rows × 80 columns

In [117…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
```

```
 ---  ------         --------------  -----
 0    Id             1460 non-null   int64
 1    MSSubClass     1460 non-null   int64
 2    MSZoning       1460 non-null   object
 3    LotFrontage    1201 non-null   float64
 4    LotArea        1460 non-null   int64
 5    Street         1460 non-null   object
 6    Alley          91 non-null     object
 7    LotShape       1460 non-null   object
 8    LandContour    1460 non-null   object
 9    Utilities      1460 non-null   object
 10   LotConfig      1460 non-null   object
 11   LandSlope      1460 non-null   object
 12   Neighborhood   1460 non-null   object
 13   Condition1     1460 non-null   object
 14   Condition2     1460 non-null   object
 15   BldgType       1460 non-null   object
 16   HouseStyle     1460 non-null   object
 17   OverallQual    1460 non-null   int64
 18   OverallCond    1460 non-null   int64
 19   YearBuilt      1460 non-null   int64
 20   YearRemodAdd   1460 non-null   int64
 21   RoofStyle      1460 non-null   object
 22   RoofMatl       1460 non-null   object
 23   Exterior1st    1460 non-null   object
 24   Exterior2nd    1460 non-null   object
 25   MasVnrType     1452 non-null   object
 26   MasVnrArea     1452 non-null   float64
 27   ExterQual      1460 non-null   object
 28   ExterCond      1460 non-null   object
 29   Foundation     1460 non-null   object
 30   BsmtQual       1423 non-null   object
 31   BsmtCond       1423 non-null   object
 32   BsmtExposure   1422 non-null   object
 33   BsmtFinType1   1423 non-null   object
 34   BsmtFinSF1     1460 non-null   int64
 35   BsmtFinType2   1422 non-null   object
 36   BsmtFinSF2     1460 non-null   int64
 37   BsmtUnfSF      1460 non-null   int64
 38   TotalBsmtSF    1460 non-null   int64
 39   Heating        1460 non-null   object
 40   HeatingQC      1460 non-null   object
 41   CentralAir     1460 non-null   object
 42   Electrical     1459 non-null   object
 43   1stFlrSF       1460 non-null   int64
 44   2ndFlrSF       1460 non-null   int64
 45   LowQualFinSF   1460 non-null   int64
 46   GrLivArea      1460 non-null   int64
 47   BsmtFullBath   1460 non-null   int64
 48   BsmtHalfBath   1460 non-null   int64
 49   FullBath       1460 non-null   int64
 50   HalfBath       1460 non-null   int64
 51   BedroomAbvGr   1460 non-null   int64
 52   KitchenAbvGr   1460 non-null   int64
 53   KitchenQual    1460 non-null   object
 54   TotRmsAbvGrd   1460 non-null   int64
 55   Functional     1460 non-null   object
 56   Fireplaces     1460 non-null   int64
 57   FireplaceQu    770 non-null    object
 58   GarageType     1379 non-null   object
```

```
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

In [118…

```
df.describe()
```

Out[118…

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Y
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	

8 rows × 38 columns

I thought that this was an interesting way to clean the data by first looking what percentage of the data is missing and then dropping any of the data that has 40% of null values missing. Then it looks like they used the median of the data to fill any of the other missing values based on year. I thought that this was a unique way to do it rather than say dropping all of the null data or trying to fill in all of the missing values with median data. I also liked the visual that shows which data is missing

In [119…

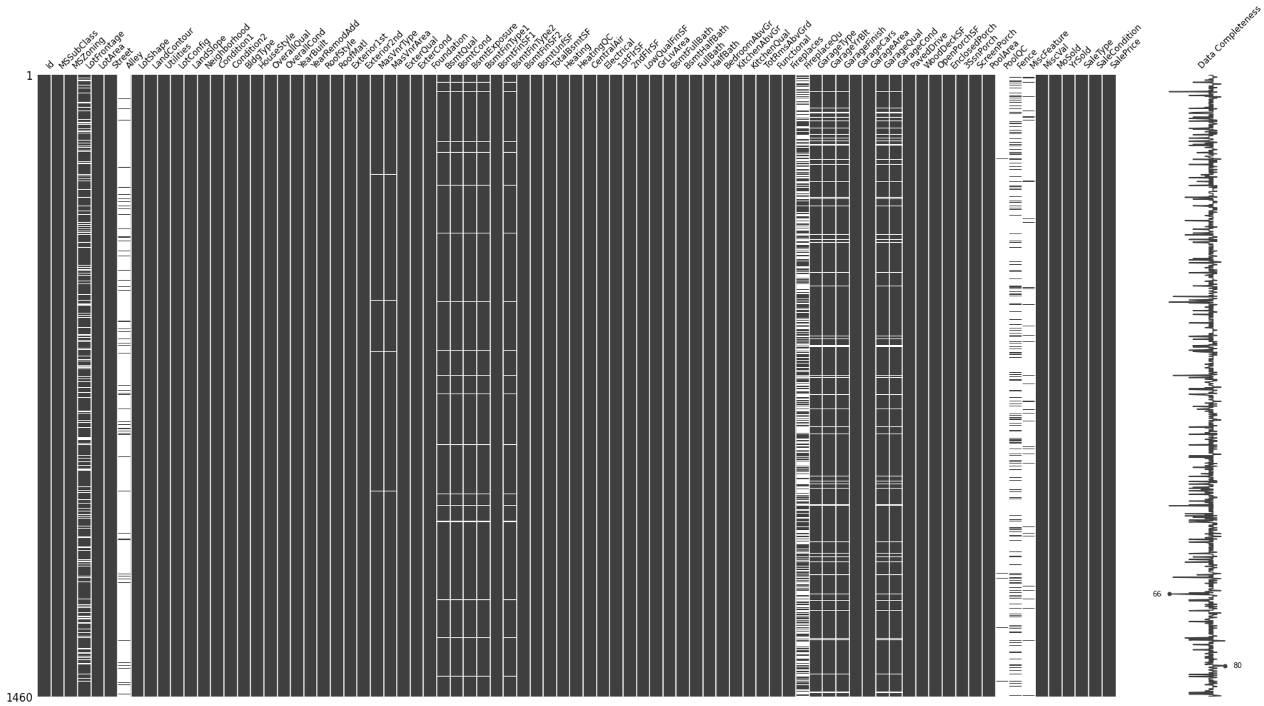
```
import missingno as msno
```

In [120…

```
msno.matrix(df,labels=[df.columns],figsize=(30,16),fontsize=12)## Visualize missing val
```

Out[120…

<AxesSubplot:>



In [121…

```
def missing (df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = ((df.isnull().sum()/df.isnull().count())*100).sort_values(ascendi
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missin
    return missing_values
```

In [122…

```
missing(df)
```

Out[122…

	Missing_Number	Missing_Percent
PoolQC	1453	99.520548
MiscFeature	1406	96.301370
Alley	1369	93.767123
Fence	1179	80.753425
FireplaceQu	690	47.260274
...	...	...
ExterQual	0	0.000000
Exterior2nd	0	0.000000
Exterior1st	0	0.000000
RoofMatl	0	0.000000
SalePrice	0	0.000000

81 rows × 2 columns

In [123…

```
for col in df.columns:
    if df[col].isnull().mean()*100>40:
        df.drop(col,axis=1,inplace=True)
```

In [124…

```
df.dtypes.value_counts()
```

Out[124…

```
object     38
int64      35
float64     3
dtype: int64
```

In [125…

```
f = lambda x: x.median() if np.issubdtype(x.dtype, np.number) else x.mode().iloc[0]
df = df.fillna(df.groupby('YrSold').transform(f))
df
```

Out[125…

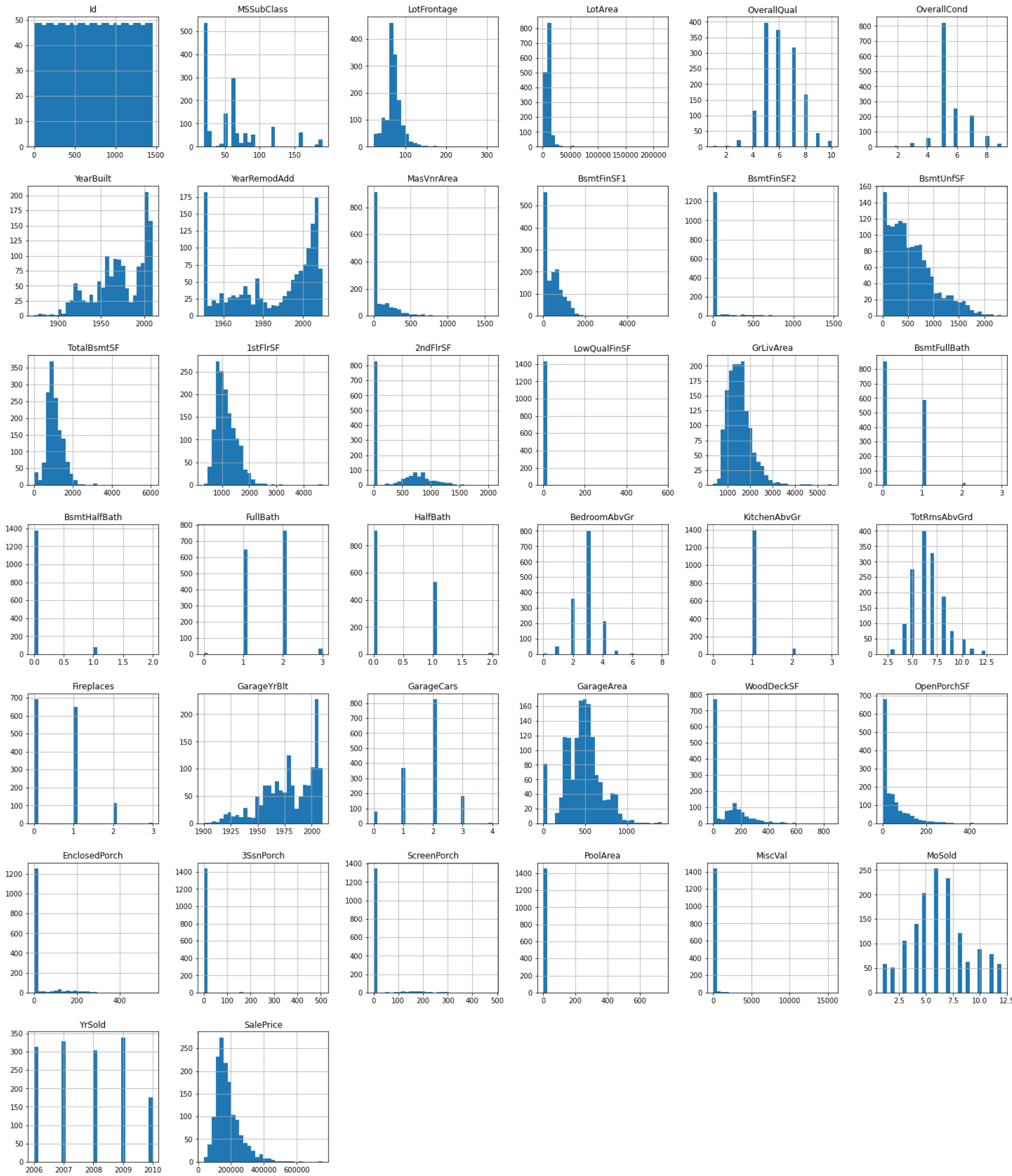
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	L
0	1	60	RL	65	8450	Pave	Reg	Lvl	AllPub	
1	2	20	RL	80	9600	Pave	Reg	Lvl	AllPub	
2	3	60	RL	68	11250	Pave	IR1	Lvl	AllPub	
3	4	70	RL	60	9550	Pave	IR1	Lvl	AllPub	
4	5	60	RL	84	14260	Pave	IR1	Lvl	AllPub	
...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62	7917	Pave	Reg	Lvl	AllPub	
1456	1457	20	RL	85	13175	Pave	Reg	Lvl	AllPub	
1457	1458	70	RL	66	9042	Pave	Reg	Lvl	AllPub	
1458	1459	20	RL	68	9717	Pave	Reg	Lvl	AllPub	
1459	1460	20	RL	75	9937	Pave	Reg	Lvl	AllPub	

1460 rows × 76 columns

Next I decided to look as some visuals that are histograms of the data and a heatmap that included correlation based on the price. This helps to see the shape of each variable and if there are outliers and how they affect the sales price if there is any correlation at all. From this we can see 'OverallQual', 'GrLivArea' have a strong correlation and that 'GarageCars', 'GarageArea', 'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt' and 'YearRemodAdd' have moderate correlations

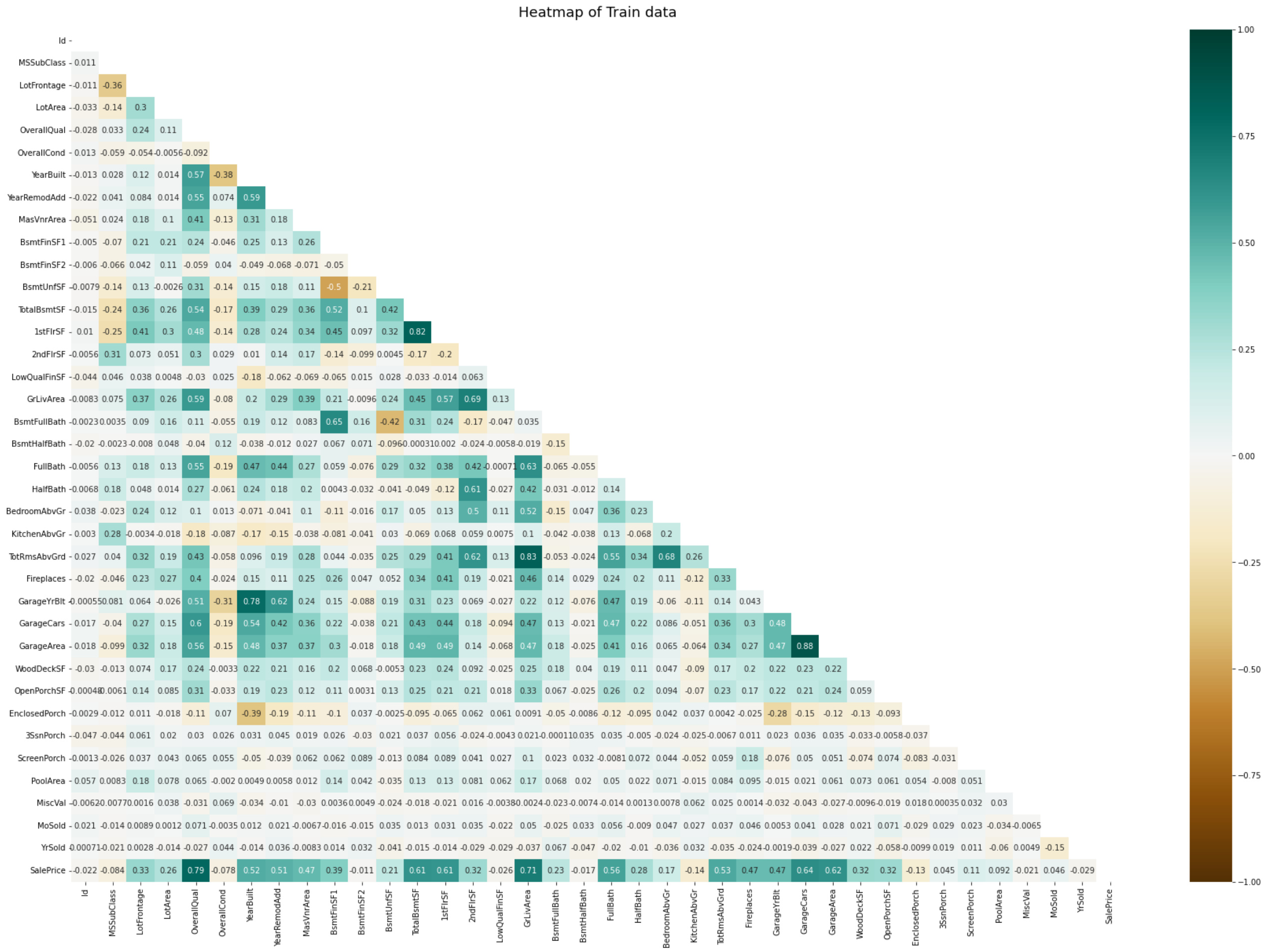
In [126…

```
df.hist(figsize=(25, 30), bins=30);
```



In [127…

```
plt.figure(figsize=(30, 20))
# define the mask to set the values in the upper triangle to True
mask = np.triu(np.ones_like(df.corr(), dtype=np.bool))
heatmap = sns.heatmap(df.corr(), mask=mask, vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Heatmap of Train data', fontdict={'fontsize':18}, pad=16);
```



Below is the models used to test for pricing. I first started by importing the necessary libraries and then I found function that will print R2 and RMSE scores easily

In [128…

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVR
from math import sqrt
import sklearn
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import ElasticNet
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn import neighbors
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import ElasticNet
from sklearn import preprocessing
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

In [129…

```
from sklearn.metrics import mean_squared_error, r2_score

# Define a function for each metric
# R²
def rsqr_score(test, pred):
    """Calculate R squared score

    Args:
        test -- test data
        pred -- predicted data

    Returns:
        R squared score
    """
    r2_ = r2_score(test, pred)
    return r2_


# RMSE
def rmse_score(test, pred):
    """Calculate Root Mean Square Error score

    Args:
        test -- test data
        pred -- predicted data

    Returns:
        Root Mean Square Error score
    """
    rmse_ = np.sqrt(mean_squared_error(test, pred))
    return rmse_


# Print the scores
def print_score(test, pred, model):
    """Print calculated score

    Args:
        test -- test data
        pred -- predicted data

    Returns:
        print the regressor name
        print the R squared score
        print Root Mean Square Error score
    """

    print(f"**** Regressor: {model} ****")
    print(f"R²: {rsqr_score(test, pred)}")
    print(f"RMSE: {rmse_score(test, pred)}\n")
```

Here instead of just normally splitting the data I created used backwards feature selection to select 60 features instead of 75 and then split the data. This actually helped some of the models performance, buyt I noticed that the KNN algorithm did best nearly a .83 r2 when there were about 10 features and random forest did marginally better with all 75

In [130…
```
df = df.apply(LabelEncoder().fit_transform)
```

In [131…
```
X=df.drop(['SalePrice','Id'],axis=1)
y=df['SalePrice']
```

In [132…
```
lreg = LinearRegression()
sfs1 = sfs(lreg, k_features=60, forward=False, verbose=0, scoring='r2',n_jobs=-1)
```

In [133…
```
sfs1 = sfs1.fit(X, y)
```

In [134…
```
feat_names = list(sfs1.k_feature_names_)
print(feat_names)
```

```
['MSSubClass', 'MSZoning', 'LotArea', 'Street', 'LotShape', 'Utilities', 'LotConfig', 'L
andSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'OverallQual', 'Overa
llCond', 'YearBuilt', 'YearRemodAdd', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVn
rArea', 'ExterQual', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinSF1',
'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'HeatingQC', 'Electrical', '2ndFlrSF', 'LowQua
lFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAb
vGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'Garage
Type', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageCond', 'PavedDrive', 'WoodDec
kSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
'MoSold', 'YrSold', 'SaleCondition']
```

In [135…
```
new_data = df[feat_names]
new_data['SalePrice'] = df['SalePrice']

new_data.head()
```

Out[135…

	MSSubClass	MSZoning	LotArea	Street	LotShape	Utilities	LotConfig	LandSlope	Neighborhood
0	5	3	327	1	3	0	4	0	5
1	0	3	498	1	3	0	2	0	24
2	5	3	702	1	0	0	4	0	5
3	6	3	489	1	0	0	0	0	6
4	5	3	925	1	0	0	2	0	15

5 rows × 61 columns

In [136…
```
X=new_data.drop('SalePrice',axis=1)
y=new_data['SalePrice']
```

In [137…
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=7)
```

Here I created all of the models and used a standard scaler when creating all of them by making a pipeline. We can see that the first three models and the SGD all did about the same with an R2 of .87 which is really good. While random forest was the best of the models and KNN did the worst with only a R2 or .735. This means that most of our models fit pretty closely to the actually data that we are testing and that it is pretty accurate at predicting the price. The elastic net did not do as well as the lasso or ridge regressions

In [139…

```
model_ridge = make_pipeline(StandardScaler(),Ridge(alpha = 0.001))
model_lasso = make_pipeline(StandardScaler(),Lasso(alpha = 0.001))
model_tree = make_pipeline(StandardScaler(),DecisionTreeRegressor())
model_ran = make_pipeline(StandardScaler(),RandomForestRegressor())
model_knn = make_pipeline(StandardScaler(),neighbors.KNeighborsRegressor(n_neighbors=5,
model_SGD = make_pipeline(StandardScaler(),SGDRegressor(max_iter=1100, tol=None))
model_lin = make_pipeline(StandardScaler(),LinearRegression())
model_elastic = make_pipeline(StandardScaler(),ElasticNet())
```

In [174…

```
model_lin.fit(X_train, y_train)
y_pred= model_lin.predict(X_test)
print_score(y_test, y_pred, "Linear")
```

```
**** Regressor: Linear ****
R²: 0.9181522826129168
RMSE: 50.024297281202074
```

Out[174…

```
25774992465.924686
```

In [144…

```
scores = cross_val_score(model_lin, X_train, y_train, scoring='accuracy', cv=cv, n_jobs
print(np.mean(s))
```

```
nan
```

In [100…

```
model_ridge.fit(X_train, y_train)
y_pred_ridge = model_ridge.predict(X_test)
print_score(y_test, y_pred_ridge, "Ridge")
```

```
**** Regressor: Ridge ****
R²: 0.9181523205570098
RMSE: 50.02428568572411
```

In [101…

```
model_lasso.fit(X_train, y_train)
y_pred_lasso = model_lasso.predict(X_test)
print_score(y_test, y_pred_lasso, "Lasso")
```

```
**** Regressor: Lasso ****
R²: 0.9181554318588996
RMSE: 50.02333488214129
```

In [62]:

```
model_ran.fit(X_train, y_train)
y_pred_ran = model_ran.predict(X_test)
print_score(y_test, y_pred_ran, "Random Forest")
```

```
**** Regressor: Random Forest ****
R²: 0.8989741621401359
RMSE: 55.57685165606271
```

In [63]:

```
model_knn.fit(X_train,y_train)
y_pred_knn = model_knn.predict(X_test)
print_score(y_test, y_pred_knn, "KNN")
```

```
**** Regressor: KNN ****
R²: 0.8182417088379529
RMSE: 74.54608256813378
```

In [64]:

```
model_SGD.fit(X_train,y_train)
y_pred_sgd = model_SGD.predict(X_test)
print_score(y_test,y_pred_sgd, "SGD")
```

```
**** Regressor: SGD ****
R²: 0.9185979164720166
RMSE: 49.887928497166335
```

In [65]:

```
model_elastic.fit(X_train,y_train)
y_pred_elastic = model_elastic.predict(X_test)
print_score(y_test,y_pred_elastic, "Elastic Net")
```

```
**** Regressor: Elastic Net ****
R²: 0.896245143506849
RMSE: 56.322500501720725
```