**JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA**

**B. TECH V SEMESTER**

**OPEN SOURCE SOFTWARE LAB**
**PBL SYNOPSIS**
**(15B17CI575)**

**TITLE OF PROJECT**

# PREDICTIVE MODELING FOR REAL ESTATE PRICING

**Supervision of :**

Dr. Megha Rathi

Dr. Indu Chawla

**Submitted By : (B-12)**

| NAME | ENROLLMENT |
|---|---|
| Nishtha Jain | 22803007 |
| Himangi Mishra | 22803016 |
| Shrankhala Singh | 22803022 |

## INTRODUCTION

Predicting real estate prices is a valuable tool for investors, buyers, and policymakers, offering insights into market trends and potential investment opportunities. With the rapid growth in data availability, applying data mining and machine learning techniques has become essential to harness information from vast datasets effectively. This project aims to predict real estate prices using Python by leveraging machine learning algorithms, specifically focusing on Linear Regression for its simplicity and interpretability in modeling linear relationships within data.

Through extensive Exploratory Data Analysis (EDA), we assess the dataset to uncover patterns, relationships, and any anomalies that could influence the model's accuracy. Key steps in EDA include visualizing features, assessing distributions, and examining correlations between variables, which help shape our understanding of the market dynamics and prepare the data for further processing. This preparation stage is crucial, as raw data often contains noise, outliers, or missing values that can affect the model's performance.

## MODELING & IMPLEMENTATION DETAILS

1. **Data Collection and Preprocessing:**
    - The project begins by importing and inspecting a real estate dataset (train.csv and test.csv). The preprocessing steps involve handling missing values by replacing numerical columns with zero (for features like 'MasVnrArea') and categorical columns with the mode (for features like 'Electrical' and 'KitchenQual'). Certain features are converted to appropriate data types, such as converting year-related features to strings. A log transformation is applied to the target variable (SalePrice) to reduce skewness. Additionally, one-hot encoding is performed for categorical features to create dummy variables, and robust scaling is applied to standardize the numerical features

2. **Exploratory Data Analysis (EDA):**
    - EDA is conducted to explore the relationships between features and the target variable. Techniques employed include:

        - Heatmaps to visualize correlations among numerical features, aiding in the selection of top correlated predictors.
        - Pair plots to analyze relationships between features and the target variable, revealing patterns and potential outliers.
        - Distribution plots to assess the skewness of numerical features and the target variable.

3. **Feature Selection and Engineering:**
    - Feature engineering is applied by:
        - Deriving a new feature, Age, calculated as the difference between the year sold and the year built.
        - Identifying and selecting the top correlated features with the target variable using correlation matrices, focusing on features with an absolute correlation greater than 0.5.

4. **Model Building with Ridge and Lasso Regression:**
    - Two regularized regression models, Ridge (L2 regularization) and Lasso (L1 regularization), are constructed to address multicollinearity and enhance prediction performance. Hyperparameter tuning is conducted using cross-validation to optimize the regularization parameter (alpha) for both models.

5. **Model Evaluation and Optimization:**
    - Model performance is assessed using:
        - **R$^2$ Score** to measure variance explained by the model.
        - **Root Mean Square Error (RMSE)** to evaluate prediction errors. Evaluation metrics are calculated for both training and testing datasets. The results inform hyperparameter adjustments to ensure the model generalizes well to unseen data.

6. Implementation and Prediction:
    - The trained models are validated on test data. Coefficients from Ridge and Lasso regression are compared to identify the most significant predictors. Features with zero coefficients in Lasso are removed, while impactful features are highlighted to provide actionable insights for real estate price prediction. Predictions are generated for the test set, and results are prepared for submission.

**IMPLEMENTATION CODE**

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

# LOAD DATA SET

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

print("Shape of train: ", train.shape)
print("Shape of test: ", test.shape)

train.head(10)

test.head(10)

## concat train and test
df = pd.concat((train, test))
temp_df = df
print("Shape of df: ", df.shape)
df.head(6)
df.tail(6)

# EXPLORATORY DATA ANALYSIS

# To show the all columns
```

```
pd.set_option("display.max_columns", 2000)
pd.set_option("display.max_rows", 85)
df.head(6)

df.tail(6)

df.info()

df.describe()

df.select_dtypes(include=['int64', 'float64']).columns

df.select_dtypes(include=['object']).columns

# Set index as Id column
df = df.set_index("Id")
df.head(6)

# Show the null values using heatmap
plt.figure(figsize=(16,9))
sns.heatmap(df.isnull())

# Get the percentages of null value
null_percent = df.isnull().sum()/df.shape[0]*100
null_percent

col_for_drop = null_percent[null_percent > 20].keys()
# if the null value % 20 or > 20 so need to drop it
# drop columns
df=df.drop(col_for_drop,axis='columns')
df.shape

# find the unique value count
for i in df.columns:
    print(i + "\t" + str(len(df[i].unique())))


# find unique values of each column
for i in df.columns:
    print("Unique value of:>>> {} ({})\n{}\n".format(i, len(df[i].unique()), df[i].unique()))

# Describe the target
train["SalePrice"].describe()
# Plot the distplot of target
plt.figure(figsize=(10,8))
bar = sns.distplot(train["SalePrice"])
```

```
bar.legend(["Skewness: {:.2f}".format(train['SalePrice'].skew())])

import matplotlib.pyplot as plt
import seaborn as sns

# Filter to keep only numeric columns for correlation
numeric_train = train.select_dtypes(include=[np.number])

# Plot the correlation heatmap
plt.figure(figsize=(25, 25))
ax = sns.heatmap(numeric_train.corr(), cmap="coolwarm", annot=True, linewidths=2)

# Fix for first and last row cut off in heatmap
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

plt.show()

# correlation heatmap of higly correlated features with SalePrice
hig_corr = numeric_train.corr()
hig_corr_features = hig_corr.index[abs(hig_corr["SalePrice"]) >= 0.5]
hig_corr_features


plt.figure(figsize=(10,8))
ax = sns.heatmap(train[hig_corr_features].corr(), cmap = "coolwarm", annot=True, linewidth=3)
# to fix the bug "first and last row cut in half of heatmap plot"
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)

# Plot regplot to get the nature of highly correlated data
plt.figure(figsize=(16,9))
for i in range(len(hig_corr_features)):
   if i <= 9:
      plt.subplot(3,4,i+1)
      plt.subplots_adjust(hspace = 0.5, wspace = 0.5)
      sns.regplot(data=train, x = hig_corr_features[i], y = 'SalePrice')
```

**# HANDLING MISSING VALUES**

```
missing_col = df.columns[df.isnull().any()]
missing_col

bsmt_col = ['BsmtCond', 'BsmtExposure', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtFinType1',
                 'BsmtFinType2',   'BsmtFullBath',   'BsmtHalfBath',   'BsmtQual',   'BsmtUnfSF',
'TotalBsmtSF']
bsmt_feat = df[bsmt_col]
bsmt_feat

bsmt_feat.info()

bsmt_feat.isnull().sum()

bsmt_feat = bsmt_feat[bsmt_feat.isnull().any(axis=1)]
bsmt_feat

bsmt_feat_all_nan = bsmt_feat[(bsmt_feat.isnull() | bsmt_feat.isin([0])).all(1)]
bsmt_feat_all_nan

bsmt_feat_all_nan.shape

qual = list(df.loc[:, df.dtypes == 'object'].columns.values)
qual

# Fillinf the mising value in bsmt features
for i in bsmt_col:
   if i in qual:
         bsmt_feat_all_nan[i] = bsmt_feat_all_nan[i].replace(np.nan, 'NA') # replace the NAN
value by 'NA'
   else:
        bsmt_feat_all_nan[i] = bsmt_feat_all_nan[i].replace(np.nan, 0) # replace the NAN value
inplace of 0

bsmt_feat.update(bsmt_feat_all_nan) # update bsmt_feat df by bsmt_feat_all_nan
df.update(bsmt_feat_all_nan) # update df by bsmt_feat_all_nan

bsmt_feat = bsmt_feat[bsmt_feat.isin([np.nan]).any(axis=1)]
bsmt_feat

bsmt_feat.shape

print(df['BsmtFinSF2'].max())
print(df['BsmtFinSF2'].min())

pd.cut(range(0,1526),5) # create a bucket

df_slice = df[(df['BsmtFinSF2'] >= 305) & (df['BsmtFinSF2'] <= 610)]
df_slice
```

```
bsmt_feat.at[333,'BsmtFinType2'] = df_slice['BsmtFinType2'].mode()[0] # replace NAN value of BsmtFinType2 by mode of buet ((305.0, 610.0)
bsmt_feat


bsmt_feat['BsmtExposure'] = bsmt_feat['BsmtExposure'].replace(np.nan, df[df['BsmtQual'] =='Gd']['BsmtExposure'].mode()[0])

bsmt_feat['BsmtCond'] = bsmt_feat['BsmtCond'].replace(np.nan, df['BsmtCond'].mode()[0])
bsmt_feat['BsmtQual'] = bsmt_feat['BsmtQual'].replace(np.nan, df['BsmtQual'].mode()[0])

df.update(bsmt_feat)

bsmt_feat.isnull().sum()

df.columns[df.isnull().any()]

garage_col = ['GarageArea', 'GarageCars', 'GarageCond', 'GarageFinish', 'GarageQual', 'GarageType', 'GarageYrBlt',]
garage_feat = df[garage_col]
garage_feat = garage_feat[garage_feat.isnull().any(axis=1)]
garage_feat

garage_feat.shape

garage_feat_all_nan = garage_feat[(garage_feat.isnull() | garage_feat.isin([0])).all(1)]
garage_feat_all_nan.shape

for i in garage_feat:
    if i in qual:
        garage_feat_all_nan[i] = garage_feat_all_nan[i].replace(np.nan, 'NA')
    else:
        garage_feat_all_nan[i] = garage_feat_all_nan[i].replace(np.nan, 0)

garage_feat.update(garage_feat_all_nan)
df.update(garage_feat_all_nan)

garage_feat = garage_feat[garage_feat.isnull().any(axis=1)]
garage_feat

for i in garage_col:
        garage_feat[i] = garage_feat[i].replace(np.nan, df[df['GarageType'] == 'Detchd'][i].mode()[0])

garage_feat.isnull().any()
```

```
df.update(garage_feat)

df['Electrical'] = df['Electrical'].fillna(df['Electrical'].mode()[0])
df['Exterior1st'] = df['Exterior1st'].fillna(df['Exterior1st'].mode()[0])
df['Exterior2nd'] = df['Exterior2nd'].fillna(df['Exterior2nd'].mode()[0])
df['Functional'] = df['Functional'].fillna(df['Functional'].mode()[0])
df['KitchenQual'] = df['KitchenQual'].fillna(df['KitchenQual'].mode()[0])
df['MSZoning'] = df['MSZoning'].fillna(df['MSZoning'].mode()[0])
df['SaleType'] = df['SaleType'].fillna(df['SaleType'].mode()[0])
df['Utilities'] = df['Utilities'].fillna(df['Utilities'].mode()[0])
#df['MasVnrType'] = df['MasVnrType'].fillna(df['MasVnrType'].mode()[0])
df.columns[df.isnull().any()]

df.columns

df['MasVnrArea'].fillna(0, inplace=True)

lotconfig = ['Corner', 'Inside', 'CulDSac', 'FR2', 'FR3']
for i in lotconfig:
    df['LotFrontage'] = np.where((df['LotFrontage'].isnull() == True) & (df['LotConfig'] == i) ,
df[df['LotConfig'] == i] ['LotFrontage'].mean(), df['LotFrontage'])
df.isnull().sum()
```

**# FEATURE TRANSFORMATION**

```
df.columns

# converting columns in str which have categorical nature but in int64
feat_dtype_convert = ['MSSubClass', 'YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold']
for i in feat_dtype_convert:
   df[i] = df[i].astype(str)
df['MoSold'].unique() # MoSold = Month of sold

# conver in month abbrevation
import calendar
df['MoSold'] = df['MoSold'].apply(lambda x : calendar.month_abbr[x])
df['MoSold'].unique()

quan = list(df.loc[:, df.dtypes != 'object'].columns.values)
quan

len(quan)

obj_feat = list(df.loc[:, df.dtypes == 'object'].columns.values)
obj_feat
```

```
# CONVERT CATEGORICAL CODE INTO ORDER

from pandas.api.types import CategoricalDtype
df['BsmtCond'] = df['BsmtCond'].astype(CategoricalDtype(categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'], ordered = True)).cat.codes
df['BsmtCond'].unique()

df['BsmtExposure'] = df['BsmtExposure'].astype(CategoricalDtype(categories=['NA', 'Mn', 'Av', 'Gd'], ordered = True)).cat.codes
df['BsmtExposure'].unique()

df['BsmtFinType1'] = df['BsmtFinType1'].astype(CategoricalDtype(categories=['NA', 'Unf', 'LwQ', 'Rec', 'BLQ','ALQ', 'GLQ'], ordered = True)).cat.codes
df['BsmtFinType2'] = df['BsmtFinType2'].astype(CategoricalDtype(categories=['NA', 'Unf', 'LwQ', 'Rec', 'BLQ','ALQ', 'GLQ'], ordered = True)).cat.codes
df['BsmtQual'] = df['BsmtQual'].astype(CategoricalDtype(categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'], ordered = True)).cat.codes
df['ExterQual'] = df['ExterQual'].astype(CategoricalDtype(categories=['Po', 'Fa', 'TA', 'Gd', 'Ex'], ordered = True)).cat.codes
df['ExterCond'] = df['ExterCond'].astype(CategoricalDtype(categories=['Po', 'Fa', 'TA', 'Gd', 'Ex'], ordered = True)).cat.codes
df['Functional'] = df['Functional'].astype(CategoricalDtype(categories=['Sal', 'Sev', 'Maj2', 'Maj1', 'Mod','Min2','Min1', 'Typ'], ordered = True)).cat.codes
df['GarageCond'] = df['GarageCond'].astype(CategoricalDtype(categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'], ordered = True)).cat.codes
df['GarageQual'] = df['GarageQual'].astype(CategoricalDtype(categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'], ordered = True)).cat.codes
df['GarageFinish'] = df['GarageFinish'].astype(CategoricalDtype(categories=['NA', 'Unf', 'RFn', 'Fin'], ordered = True)).cat.codes
df['HeatingQC'] = df['HeatingQC'].astype(CategoricalDtype(categories=['Po', 'Fa', 'TA', 'Gd', 'Ex'], ordered = True)).cat.codes
df['KitchenQual'] = df['KitchenQual'].astype(CategoricalDtype(categories=['Po', 'Fa', 'TA', 'Gd', 'Ex'], ordered = True)).cat.codes
df['PavedDrive'] = df['PavedDrive'].astype(CategoricalDtype(categories=['N', 'P', 'Y'], ordered = True)).cat.codes
df['Utilities'] = df['Utilities'].astype(CategoricalDtype(categories=['ELO', 'NASeWa', 'NASeWr', 'AllPub'], ordered = True)).cat.codes
df['Utilities'].unique()

# SHOW SKEWNESS OF FEATURE WITH DISTPLOT

skewed_features = ['1stFlrSF',
 '2ndFlrSF',
 '3SsnPorch',
 'BedroomAbvGr',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtFullBath',
 'BsmtHalfBath',
 'BsmtUnfSF',
 'EnclosedPorch',
 'Fireplaces',
 'FullBath',
 'GarageArea',
 'GarageCars',
 'GrLivArea',
 'HalfBath',
 'KitchenAbvGr',
 'LotArea',
 'LotFrontage',
 'LowQualFinSF',
 'MasVnrArea',
 'MiscVal',
 'OpenPorchSF',
 'PoolArea',
 'ScreenPorch',
 'TotRmsAbvGrd',
 'TotalBsmtSF',
 'WoodDeckSF']
quan == skewed_features

plt.figure(figsize=(25,20))
for i in range(len(skewed_features)):
    if i <= 28:
        plt.subplot(7,4,i+1)
        plt.subplots_adjust(hspace = 0.5, wspace = 0.5)
        ax = sns.distplot(df[skewed_features[i]])
                ax.legend(["Skewness: {:.2f}".format(df[skewed_features[i]].skew())], fontsize =
'xx-large')

df_back = df
# decrease the skewnwnes of the data
for i in skewed_features:
    df[i] = np.log(df[i] + 1)
plt.figure(figsize=(25,20))
for i in range(len(skewed_features)):
    if i <= 28:
        plt.subplot(7,4,i+1)
        plt.subplots_adjust(hspace = 0.5, wspace = 0.5)
        ax = sns.distplot(df[skewed_features[i]])
                ax.legend(["Skewness: {:.2f}".format(df[skewed_features[i]].skew())], fontsize =
'xx-large')
```

```
SalePrice = np.log(train['SalePrice'] + 1)
# get object feature to conver in numeric using dummy variable
obj_feat = list(df.loc[:,df.dtypes == 'object'].columns.values)
len(obj_feat)

# dummy varaibale
dummy_drop = []
clean_df = df
for i in obj_feat:
    dummy_drop += [i + '_' + str(df[i].unique()[-1])]

df = pd.get_dummies(df, columns = obj_feat)
df = df.drop(dummy_drop, axis = 1)
df.shape
```

**# MACHINE LEARNING MODEL BUILDING**

```
# scaling dataset with robust scaler
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
scaler.fit(df)
df = scaler.transform(df)

train_len = len(SalePrice)  # Ensure train_len matches the length of y_train

X_train = df[:train_len]  # Align X_train with SalePrice
X_test = df[train_len:]   # The remaining rows for X_test
y_train = SalePrice

print(X_train.shape)
print(X_test.shape)
print(len(y_train))
```

**# CROSS VALIDATION**

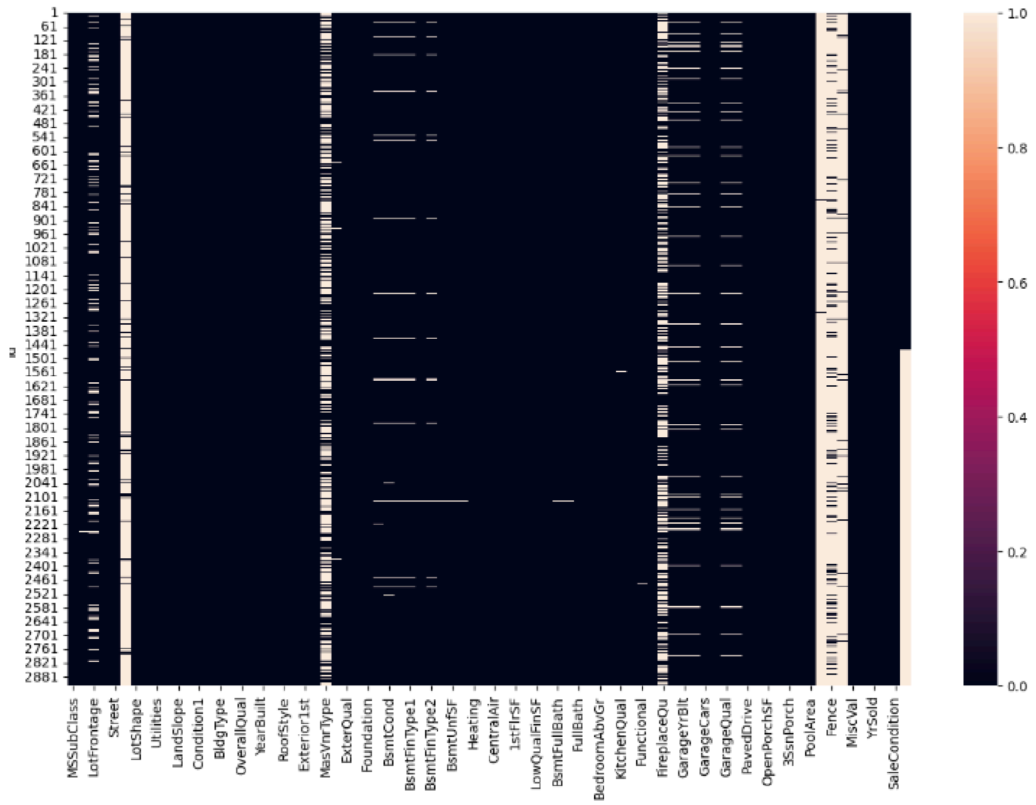
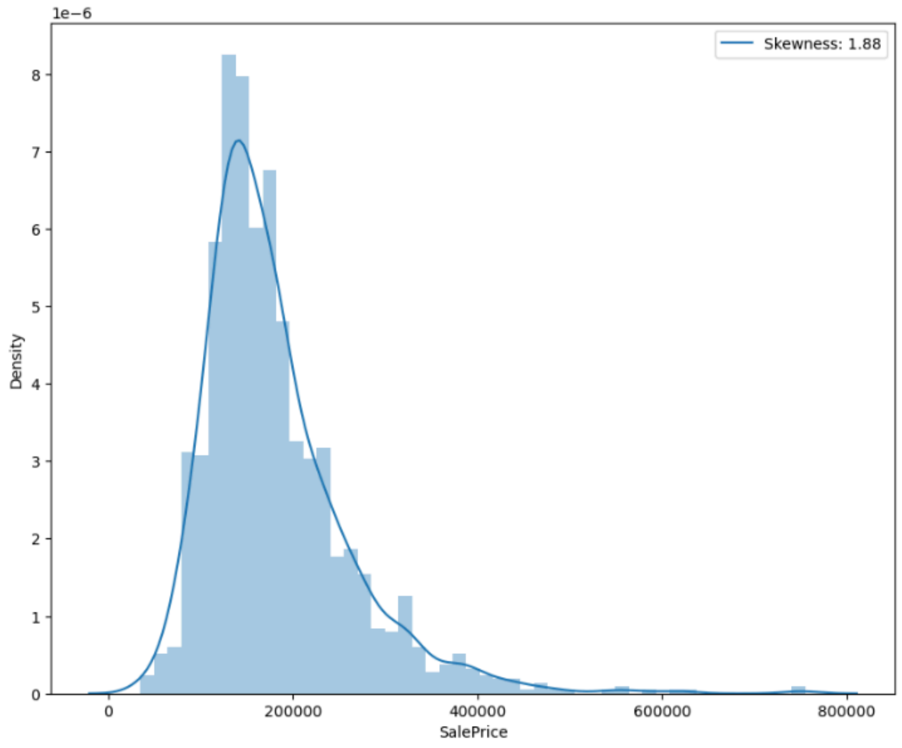
```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import make_scorer, r2_score

def test_model(model, X_train=X_train, y_train=y_train):
    cv = KFold(n_splits = 3, shuffle=True, random_state = 45)
    r2 = make_scorer(r2_score)
    r2_val_score = cross_val_score(model, X_train, y_train, cv=cv, scoring = r2)
    score = [r2_val_score.mean()]
    return score
```

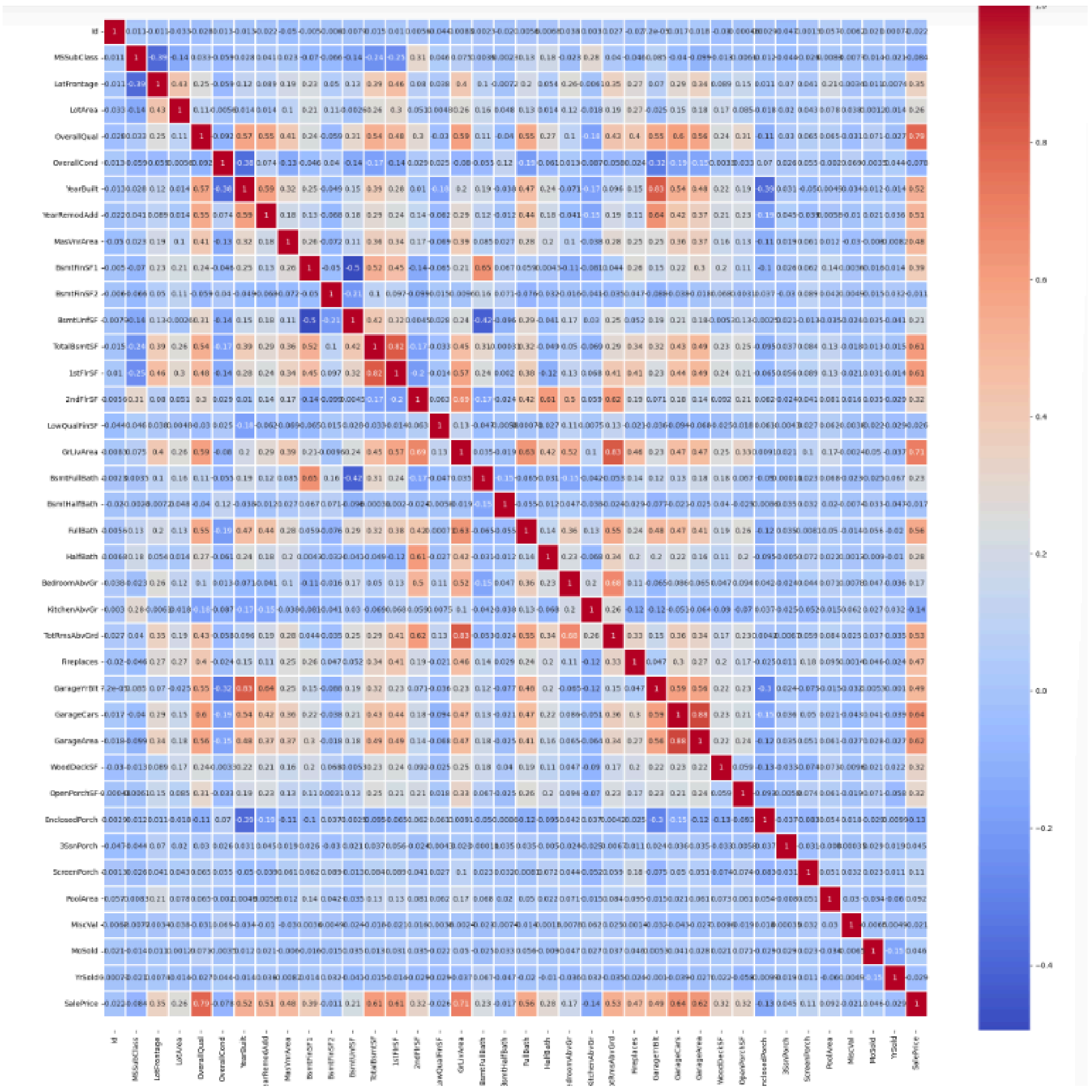
**# LINEAR REGRESSION**

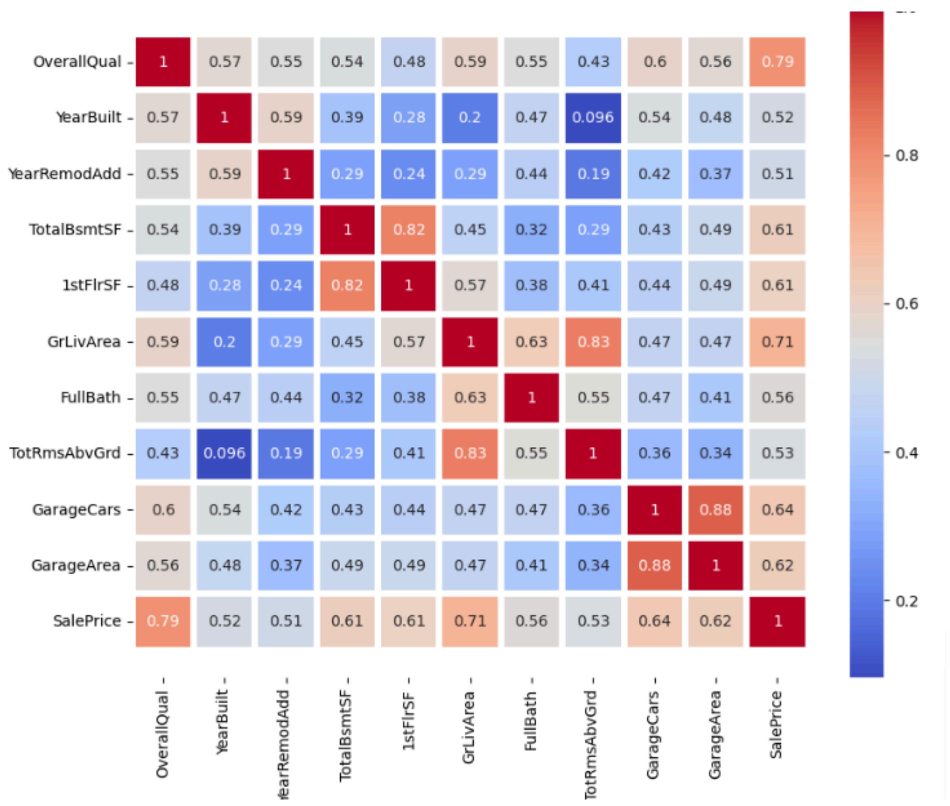
```
import sklearn.linear_model as linear_model
```

```
LR = linear_model.LinearRegression()
test_model(LR)

# Cross validation
cross_validation = cross_val_score(estimator = LR, X = X_train, y = y_train, cv = 10)
print("Cross validation accuracy of LR model = ", cross_validation)
print("\nCross validation mean accuracy of LR model = ", cross_validation.mean())

# Ridge

rdg = linear_model.Ridge()
test_model(rdg)

lasso = linear_model.Lasso(alpha=1e-4)
test_model(lasso)

# DECISION TREE REGRESSOR

from sklearn.tree import DecisionTreeRegressor
dt_reg = DecisionTreeRegressor(random_state=21)
test_model(dt_reg)

# SUPPORT VECTOR MACHINE

# FITTING SUPPORT VECTOR MACHINE TO THE DATASET
from sklearn.svm import SVR
svr_reg = SVR(kernel='rbf')
test_model(svr_reg)

svr_reg.fit(X_train,y_train)
y_pred = np.exp(svr_reg.predict(X_test)).round(2)

y_pred

submit_test1 = pd.concat([test['Id'],pd.DataFrame(y_pred)], axis=1)
submit_test1.columns=['Id', 'SalePrice']

submit_test1

# RANDOM FOREST REGRESSOR

train_len = len(SalePrice)  # Ensure train_len matches the length of y_train

X_train = df[:train_len]  # Align X_train with SalePrice
X_test = df[train_len:]   # The remaining rows for X_test
y_train = SalePrice
```

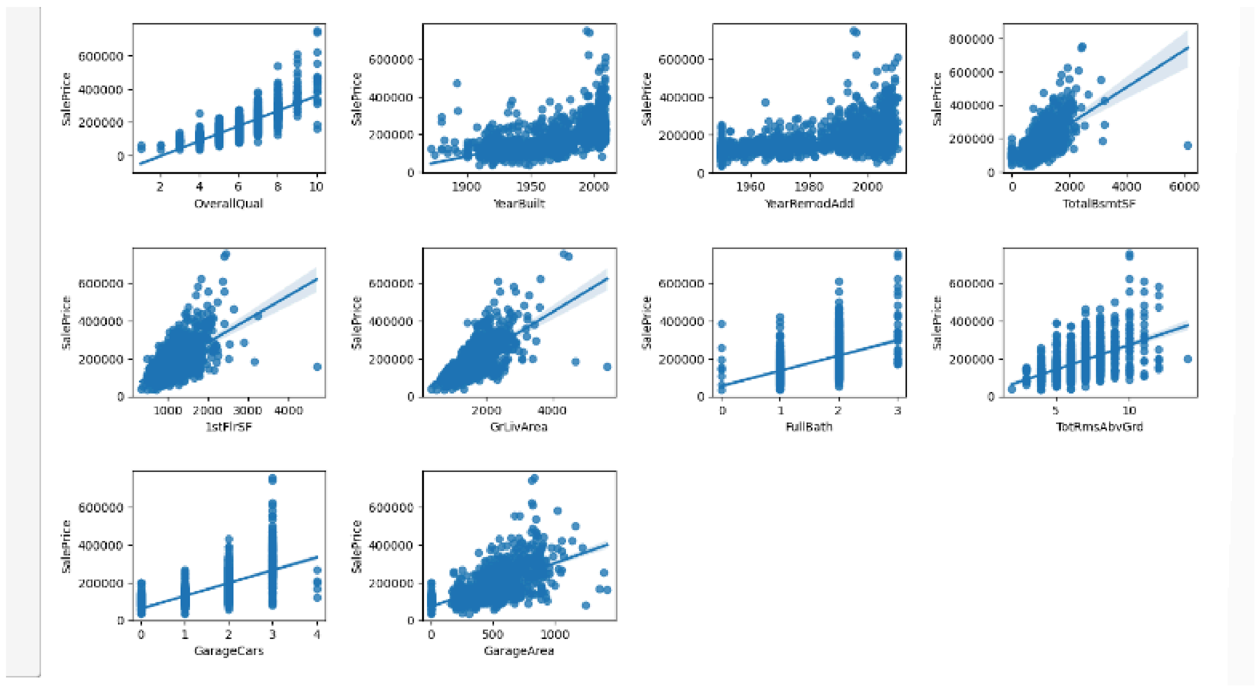
```
print(X_train.shape)
print(X_test.shape)
print(len(y_train))

from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

X_train, X_valid, Y_train, Y_valid = train_test_split(X_train, y_train, train_size=0.8,
test_size=0.2,random_state=0)

from sklearn.ensemble import RandomForestRegressor

model_RFR = RandomForestRegressor()
model_RFR.fit(X_train, Y_train)
Y_pred = model_RFR.predict(X_valid)
print(mean_absolute_error(Y_valid, Y_pred))
```

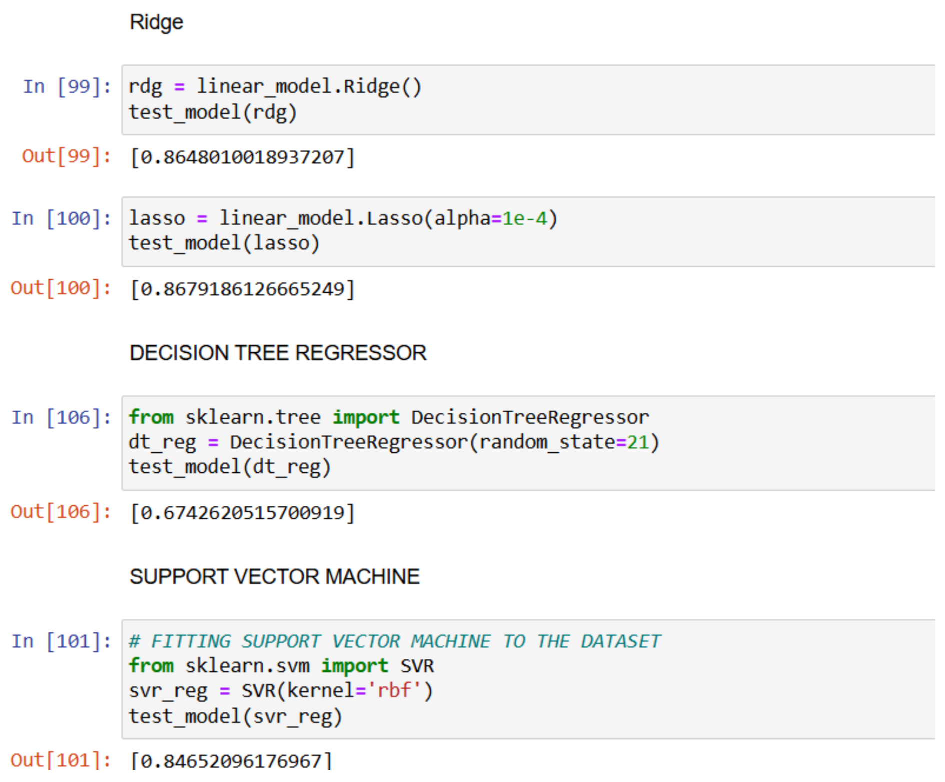

MSSubClass
LotFrontage
Street
LotShape
Utilities
LandSlope
Condition1
BldgType
OverallQual
YearBuilt
RoofStyle
Exterior1st
MasVnrType
ExterQual
Foundation
BsmtCond
BsmtFinType1
BsmtFinType2
BsmtUnfSF
Heating
CentralAir
1stFlrSF
LowQualFinSF
BsmtFullBath
FullBath
BedroomAbvGr
KitchenQual
Functional
FireplaceQu
GarageYrBlt
GarageCars
GarageQual
PavedDrive
OpenPorchSF
3SsnPorch
PoolArea
Fence
MiscVal
YrSold
SaleCondition

1e−6
Skewness: 1.88
Density
SalePrice


Id
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
SalePrice
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4



| | OverallQual | YearBuilt | YearRemodAdd | TotalBsmtSF | 1stFlrSF | GrLivArea | FullBath | TotRmsAbvGrd | GarageCars | GarageArea | SalePrice |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OverallQual | 1 | 0.57 | 0.55 | 0.54 | 0.48 | 0.59 | 0.55 | 0.43 | 0.6 | 0.56 | 0.79 |
| YearBuilt | 0.57 | 1 | 0.59 | 0.39 | 0.28 | 0.2 | 0.47 | 0.096 | 0.54 | 0.48 | 0.52 |
| YearRemodAdd | 0.55 | 0.59 | 1 | 0.29 | 0.24 | 0.29 | 0.44 | 0.19 | 0.42 | 0.37 | 0.51 |
| TotalBsmtSF | 0.54 | 0.39 | 0.29 | 1 | 0.82 | 0.45 | 0.32 | 0.29 | 0.43 | 0.49 | 0.61 |
| 1stFlrSF | 0.48 | 0.28 | 0.24 | 0.82 | 1 | 0.57 | 0.38 | 0.41 | 0.44 | 0.49 | 0.61 |
| GrLivArea | 0.59 | 0.2 | 0.29 | 0.45 | 0.57 | 1 | 0.63 | 0.83 | 0.47 | 0.47 | 0.71 |
| FullBath | 0.55 | 0.47 | 0.44 | 0.32 | 0.38 | 0.63 | 1 | 0.55 | 0.47 | 0.41 | 0.56 |
| TotRmsAbvGrd | 0.43 | 0.096 | 0.19 | 0.29 | 0.41 | 0.83 | 0.55 | 1 | 0.36 | 0.34 | 0.53 |
| GarageCars | 0.6 | 0.54 | 0.42 | 0.43 | 0.44 | 0.47 | 0.47 | 0.36 | 1 | 0.88 | 0.64 |
| GarageArea | 0.56 | 0.48 | 0.37 | 0.49 | 0.49 | 0.47 | 0.41 | 0.34 | 0.88 | 1 | 0.62 |
| SalePrice | 0.79 | 0.52 | 0.51 | 0.61 | 0.61 | 0.71 | 0.56 | 0.53 | 0.64 | 0.62 | 1 |





Ridge

```
In [99]:  rdg = linear_model.Ridge()
          test_model(rdg)

Out[99]:  [0.8648010018937207]

In [100]: lasso = linear_model.Lasso(alpha=1e-4)
          test_model(lasso)

Out[100]: [0.8679186126665249]
```

DECISION TREE REGRESSOR

```
In [106]: from sklearn.tree import DecisionTreeRegressor
          dt_reg = DecisionTreeRegressor(random_state=21)
          test_model(dt_reg)

Out[106]: [0.6742620515700919]
```

SUPPORT VECTOR MACHINE

```
In [101]: # FITTING SUPPORT VECTOR MACHINE TO THE DATASET
          from sklearn.svm import SVR
          svr_reg = SVR(kernel='rbf')
          test_model(svr_reg)

Out[101]: [0.84652096176967]
```

## TECHNOLOGIES / TOOLS USED

- Language: Python
- Libraries:-
  - Pandas:
    - Utilized for data manipulation and analysis, including loading, inspecting, cleaning, and transforming the real estate dataset (train.csv and test.csv).
    - Key tasks include handling missing values (replacing numerical columns with zero and categorical columns with the mode), encoding categorical data through one-hot encoding, and creating new features such as 'Age' calculated from the year sold and year built.
  - NumPy:
    - Used for numerical computations, particularly for matrix operations during data preprocessing and feature scaling.

  - Facilitates efficient handling of numerical arrays for data transformation, including applying log transformations to reduce skewness in the target variable (SalePrice).

- Matplotlib and Seaborn:
  - Employed for data visualization, including heatmaps to visualize correlations among features, distribution plots to assess skewness, and scatter plots for analyzing relationships between features and the target variable.
  - These visualizations provide insights into relationships and patterns, guiding feature selection and engineering.

- Scikit-learn:
  - Used for implementing and evaluating machine learning models such as Linear Regression, Ridge Regression, and Lasso Regression.
  - Tools like RobustScaler for feature scaling and cross_val_score for model evaluation are extensively utilized, along with GridSearchCV for hyperparameter tuning to optimize the regularization parameters for Ridge and Lasso models.

## CONCLUSION

This project successfully demonstrates the potential of machine learning to predict real estate prices using Python, providing valuable insights into market trends and influencing factors. Combining data mining techniques with linear regression and robust data preprocessing, the model delivers reliable predictions and supports informed decision-making for investors, buyers, and real estate professionals. Through Exploratory Data Analysis (EDA), we gained a deeper understanding of key variables impacting property values, allowing for more precise modeling.

The approach outlined in this project not only showcases the efficacy of predictive analytics in real estate but also underscores the broader applicability of these techniques across various industries. As data-driven methods continue to advance, future enhancements—such as incorporating more sophisticated algorithms, external data sources, and real-time updates—can further refine predictive accuracy and adapt to market dynamics. Ultimately, this project lays a strong foundation for developing adaptable, insightful, and user-oriented predictive models that bridge the gap between data and actionable intelligence.

## FUTURE WORK

1. **Advanced Machine Learning Models:**
   - Future work could involve experimenting with more sophisticated models, like Decision Trees, Random Forests, or Gradient Boosting Machines, which may capture non-linear relationships more effectively than Linear Regression.
   - Incorporating neural networks or deep learning models, such as Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs), could improve accuracy, particularly with large and complex datasets.

2. **Incorporating Time-Series Analysis:**
   - By integrating time-series forecasting, the model could capture trends and seasonal variations, making it possible to predict not just current but future market trends. This would be beneficial for long-term investment forecasting and market analysis.

3. **External Data Sources:**
   - Using additional datasets, like economic indicators, weather patterns, or location-specific amenities (e.g., proximity to schools, hospitals, or public transport), could enhance model accuracy by incorporating factors that influence real estate prices beyond property-specific attributes.

4. **Real-Time Data Integration:**
   - Future models could be designed to work with real-time data feeds. For instance, pulling data directly from real estate websites or government records to keep predictions updated and relevant for dynamic market analysis.

5. **Interactive User Interface:**
    - Developing a user-friendly application or web-based dashboard would allow users to input desired parameters, get instant price predictions, and visualize market trends interactively. This interface could be useful for buyers, investors, and real estate agents.

## Cross-Industry Applications

1. Healthcare:
    - Predicting the cost of medical treatments based on patient history, demographic data, and prevailing market trends. For example, hospitals could estimate costs for personalized care plans and resource allocation.

2. Automobile Sales:
    - Predicting car resale values based on vehicle specifications, mileage, age, and other factors can be valuable for dealerships, insurance companies, and individuals looking to buy or sell vehicles.

3. Financial Markets:
    - Predicting stock or commodity prices by analyzing historical data and economic indicators. This could aid investors in making informed decisions, much like in the real estate market.

4. Retail and E-commerce:
    - Predicting sales trends by analyzing customer data, seasonal trends, and inventory levels to optimize stock, pricing, and promotions. Retailers could forecast demand for products based on previous data and customer preferences.

5. Agriculture:
    - Forecasting crop yields and prices using weather data, soil quality, and past production records. Farmers and agricultural businesses can use such models to plan crop cycles and manage pricing strategies.

6. Tourism and Hospitality:

- Predicting hotel room rates or occupancy levels based on seasonal trends, local events, and economic conditions. This could help hotels optimize pricing strategies and resource allocation, especially in high-demand periods.

## DATASET

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data