Report On

House Price Prediction Using Cross Validation

Submitted in partial fulfillment of the requirements of the Course project in
Semester VIII of Final Year Computer Engineering

By
Vipul Bhoir(Roll No. 05)
Mrudul Chaudhari (Roll No. 10)
Abhinav Desai(Roll No.12)

Mentor
Ms. Brinal Colaco

University of Mumbai

Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

(A.Y. 2024-25)

Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled “House Price Prediction Using Cross Validation” is a bonafide work of "Vipul Bhoir (Roll No. 05), Mrudul Chaudhari (Roll No. 10), Abhinav Desai (Roll No. 12)" submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester VIII of Final Year Computer Engineering.

Supervisor

Prof. Name Surname

Internal Examiner	External Examiner
Dr Megha Trivedi Head of Department	Mr. Rakesh Himte Principal

Contents

ABSTRACT

The real estate market is influenced by numerous dynamic factors, making house price prediction a complex yet essential task. This project aims to develop a robust machine learning model to predict house prices based on various features such as location, number of rooms, square footage, property age, and more. To enhance model reliability and generalization, cross-validation techniques are employed throughout the training and evaluation process.

The project leverages advanced data preprocessing techniques including outlier detection, feature engineering, and normalization to ensure high-quality input data. Multiple regression algorithms such as Linear Regression, Random Forest Regressor, Gradient Boosting, and XGBoost are trained and evaluated using k-fold cross-validation, which helps in mitigating overfitting and provides an unbiased estimate of model performance.

Key performance metrics such as Root Mean Squared Error (RMSE) and R^2 Score are used to compare model effectiveness. The final model is selected based on its ability to generalize well across unseen data, demonstrating the practical applicability of machine learning in the real estate domain.

This project not only showcases the effectiveness of cross-validation in predictive modeling but also serves as a valuable tool for stakeholders in the housing market, including buyers, sellers, and real estate agents, by providing data-driven price estimates.

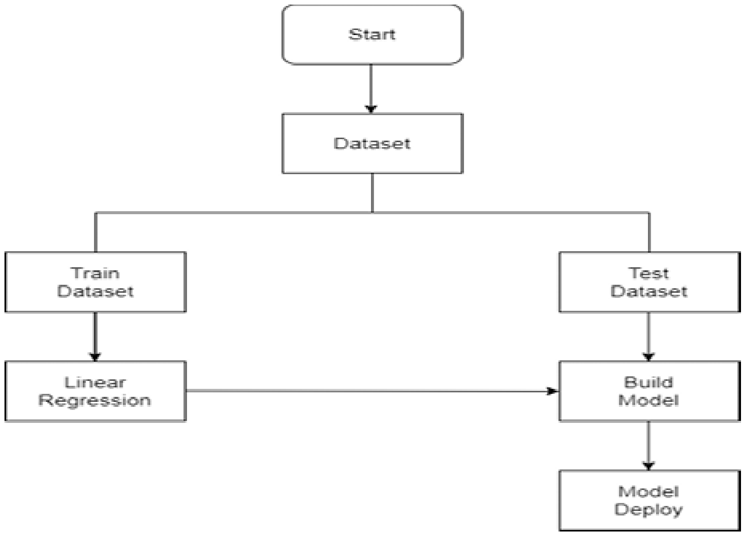
PROBLEM STATEMENT

Predicting house prices accurately is a critical challenge in the real estate sector due to the influence of numerous interdependent factors such as location, property size, age, amenities, and market trends. Traditional estimation methods often fail to capture complex nonlinear relationships between features, leading to inaccurate and unreliable pricing.

The primary objective of this project is to develop a machine learning-based regression model that can predict house prices with high accuracy using historical housing data. To improve model performance and ensure its ability to generalize well to unseen data, cross-validation techniques will be applied during model training and evaluation. This approach aims to minimize overfitting, improve model robustness, and provide consistent predictive performance across different datasets.

This problem addresses the need for a data-driven, automated, and scalable solution that can assist buyers, sellers, and real estate professionals in making informed decisions based on predicted property values.

BLOCK DIAGRAM



MODULE DESCRIPTION

• **Data Collection:** The training and test datasets are loaded from CSV files.Exploratory data analysis is performed to understand the structure and characteristics of the data.Data visualization techniques such as histograms, box plots, and heatmaps are used to analyze the distribution of features and identify missing values.

• **Data Preprocessing:** Missing values are handled using appropriate techniques such as imputation or dropping columns. Categorical variables are encoded using one-hot encoding.Numerical features are standardized to ensure uniformity and improve model performance.

• **Model Training:** Several regression models are considered, including Linear Regression, SVR, SGDRegressor, KNeighborsRegressor,DecisionTreeRegressor,RandomForestRegressor,GradientBoostingRegressor,XGBRegressor, and MLPRegressor. Cross-validation is used to evaluate each model's performance based on the R-squared score.The GradientBoostingRegressor model is selected based on its superior performance.

• **Prediction and Results:** The trained model predicts whether a patient is likely to miss their appointment. Based on the prediction, healthcare centers can take proactive actions, such as sending reminders or reallocating appointment slots, to reduce no-show rates and improve operational efficiency.

HARDWARE & SOFTWARE REQUIREMENT

HARDWARE REQUIREMENT

An Intel based central processing unit capable of running any sort of windows operating system such as Pentium based workstation.

1. Minimum 64 MB RAM (128 MB Desirable) at server.
2. Minimum 60 MB of free disk space for files.
3. A CD Rom drive
4. Minimum 48 MB of RAM at workstation.
5. VGA 15” color monitor for workstation.

SOFTWARE REQUIREMENT-

The software requirements are as follows. Integrated
Development Environment (IDE):

Visual Studio Code: General-purpose code editors with extensions for data analysis.

Programming Language: Python (for data analysis and visualization)

Data Analysis Tools: Jupyter Notebook, Excel (if needed)

Libraries: Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn (for machine learning models)

Visualization Tools: Power BI, Tableau (for advanced visualizations)

Operating System: Windows, macOS, or Linux

CODE & OUTPUT:

CODE:

main.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
from pandas.api.types import CategoricalDtype
from sklearn.preprocessing import StandardScaler

"""## Data Loading"""

train_data_path=r"/content/train.csv"
test_data_path=r"/content/test.csv"

df_train=pd.read_csv(train_data_path)
df_test=pd.read_csv(test_data_path)

print(df_train.shape)
print(df_test.shape)

"""# Data Analysis"""

pd.set_option('display.max_columns',None) #to display all columns
pd.set_option('display.max_rows',None) #to display all rows

df_train.head()

df_test.head()

"""### data integration"""

df=pd.concat((df_train,df_test))
temp_df=df
print(df.shape)

df.head()

df.tail()

"""## Exploratory data analysis"""

df.info()

df.describe()

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

float_feature=df.select_dtypes(include=['float64']).columns
```

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

"""### visualing missing value"""

plt.figure(figsize=(16,9))
sns.heatmap(df.isnull())

# set index as is column
df=df.set_index("Id")

"""### get the null value percentage for every feature"""

null_count=df.isnull().sum()
null_count

null_percent=df.isnull().sum()/df.shape[0]*100
null_percent

"""## drop column/features"""

""" as per domain knowldge we will not drop this featurre rather we add some constant value 'NA'  """
miss_value_50_perc=null_percent[null_percent>50]
miss_value_50_perc

""" as per domain knowldge we will not drop this featurre rather we add some constant value 'NA'  """
miss_value_20_50_perc=null_percent[(null_percent>20)& (null_percent<51)]
miss_value_20_50_perc

miss_value_5_20_perc=null_percent[(null_percent>5)& (null_percent<21)]
miss_value_5_20_perc

sns.heatmap(df[miss_value_5_20_perc.keys()].isnull())

missing_value_feat=null_percent[null_percent>0]
print("Total missing value feature=",len(missing_value_feat))
missing_value_feat

cat_na_feat=missing_value_feat[missing_value_feat.keys().isin(cat_feature)]
print("total number of categorical missing feature",len(cat_na_feat))
cat_na_feat

int_na_feat=missing_value_feat[missing_value_feat.keys().isin(int_feature)]
print("total number of int missing feature",len(int_na_feat))
int_na_feat

float_na_feat=missing_value_feat[missing_value_feat.keys().isin(float_feature)]
print("total number of float missing feature",len(float_na_feat))
float_na_feat

## funtion to visualize data feature before and after imputation of missing value
def plot_data(df, df_new, feature):
    plt.subplot(121)
    sns.countplot(x=feature, data=df)
    plt.title("Before Imputation")
```

```
    plt.subplot(122)
    sns.countplot(x=feature, data=df_new)
    plt.title("After Imputation")
    plt.show()

### handling MSZoning=0.137033

df["MSZoning"].value_counts()

# count plot in graph form
sns.countplot(x=df["MSZoning"])

## backing up origianl data frame
df_mvi=df.copy()

# as we can see here RL is the mode for this feature
mszoning_mode=df["MSZoning"].mode()[0]
mszoning_mode
df_mvi["MSZoning"].replace(np.nan,mszoning_mode,inplace=True)
# now chceck do we have any missinf vLUE
df_mvi["MSZoning"].isnull().sum()

#compare before and after imputation
feature="MSZoning"
plot_data(df,df_mvi,feature)

## handleing alley = 93.216855

df_mvi["Alley"].value_counts()
alley_cont="NA"
df_mvi["Alley"].replace(np.nan,alley_cont,inplace=True) # replace missing value with 'NA'
df_mvi["Alley"].isnull().sum()

# compare before and after imputation
plot_data(df,df_mvi,"Alley")

#LotFrontage=16.649538
def boxHistPlot(df,feature, figsize=(16,5)):
    plt.figure(figsize=figsize)
    plt.subplot(121)
    sns.boxplot(x=feature, data=df)
    plt.subplot(122)
    sns.histplot(x=feature,data=df,stat="density", kde=True)
    plt.show()

boxHistPlot(df,"LotFrontage")

lotfrontage_mean=df["LotFrontage"].mean()
# lotfrontage_mean
df_mvi["LotFrontage"].replace(np.nan,lotfrontage_mean,inplace=True)
df_mvi["LotFrontage"].isnull().sum()

# compare old and new box hist plot after imputation
def oldNewBoxHistPlot(df,df_new,feature, figsize=(16,5)):
```

```
    plt.figure(figsize=figsize)
    plt.subplot(221)
    sns.boxplot(x=feature, data=df)
    plt.title("Before Imputation")
    plt.subplot(222)
    sns.histplot(x=feature,data=df,stat="density", kde=True)
    plt.title("Before Imputation")
    plt.figure(figsize=figsize)
    plt.subplot(223)
    sns.boxplot(x=feature, data=df_new)
    plt.title("After Imputation")
    plt.subplot(224)
    sns.histplot(x=feature,data=df_new,stat="density", kde=True)
    plt.title("After Imputation")
    plt.show()

oldNewBoxHistPlot(df,df_mvi,"LotFrontage")

## handling utility
df["Utilities"].value_counts()
utility_const=df["Utilities"].mode()[0]
df_mvi["Utilities"].replace(np.nan,utility_const,inplace=True)
df_mvi["Utilities"].isnull().sum()

print(df["Exterior1st"].value_counts())
print("----")
print(df["Exterior2nd"].value_counts())

exterior_1_const=df["Exterior1st"].mode()[0]
df_mvi["Exterior1st"].replace(np.nan,exterior_1_const,inplace=True)
df_mvi["Exterior1st"].isnull().sum()

exterior_2_const=df["Exterior2nd"].mode()[0]
df_mvi["Exterior2nd"].replace(np.nan,exterior_2_const,inplace=True)
df_mvi["Exterior2nd"].isnull().sum()

# MasVnrType      0.822199
# MasVnrArea      0.787941

sns.heatmap(df[["MasVnrArea", "MasVnrType"]].isnull())

mas_vnr_type_const=df["MasVnrType"].mode()[0]
df_mvi["MasVnrType"].replace(np.nan,mas_vnr_type_const,inplace=True)
df_mvi["MasVnrType"].isnull().sum()

boxHistPlot(df,"MasVnrArea")

mas_vnr_area_const=0# as we can see the mode is 0 in above plots
df_mvi["MasVnrArea"].replace(np.nan,mas_vnr_area_const,inplace=True)
df_mvi["MasVnrArea"].isnull().sum()

cat_bsmt_feat=["BsmtQual","BsmtCond","BsmtExposure","BsmtFinType1","BsmtFinType2"]
num_bsmt_feat=["BsmtFinSF1","BsmtFinSF2","BsmtUnfSF","TotalBsmtSF","BsmtFullBath","BsmtHalfBath"]
```

```
sns.heatmap(df[cat_bsmt_feat].isnull()) # check missing values in categorical features
for feat in cat_bsmt_feat:
    print(df[feat].value_counts())
    print("----")

bsmt_cont="NA"
for feat in cat_bsmt_feat:
    df_mvi[feat].replace(np.nan,bsmt_cont,inplace=True)

sns.heatmap(df[num_bsmt_feat].isnull()) # check missing values in numerical features

# analysing basement feature
df_bsmt=df[cat_bsmt_feat+num_bsmt_feat]
df_bsmt[df_bsmt.isnull().any(axis=1)]

bsmt_num=0
for feat in num_bsmt_feat:
    df_mvi[feat].replace(np.nan,bsmt_num,inplace=True)

# Electrical       0.034258 -- KitchenQual      0.034258
df["Electrical"].value_counts()

df["KitchenQual"].value_counts()

df_ekk=df[["Electrical","KitchenQual","KitchenAbvGr"]]
df_ekk[df_ekk.isnull().any(axis=1)]

electrical_mode=df["Electrical"].mode()[0]
df_mvi["Electrical"].replace(np.nan,electrical_mode,inplace=True)
df_mvi["Electrical"].isnull().sum()

kitchenqual_mode=df["KitchenQual"].mode()[0]
df_mvi["KitchenQual"].replace(np.nan,kitchenqual_mode,inplace=True)
df_mvi["KitchenQual"].isnull().sum()

print(df["Functional"].value_counts())
print("----")
print(df["FireplaceQu"].value_counts())
print("----")
print(df["PoolQC"].value_counts())
print("----")
print(df["Fence"].value_counts())
print("----")
print(df["MiscFeature"].value_counts())
print("----")
print(df["SaleType"].value_counts())
print("----")

functional_mode=df["Functional"].mode()[0]
df_mvi["Functional"].replace(np.nan,functional_mode,inplace=True)
df_mvi["Functional"].isnull().sum()

saletype_mode=df["SaleType"].mode()[0]
df_mvi["SaleType"].replace(np.nan,saletype_mode,inplace=True)
```

```
df_mvi["SaleType"].isnull().sum()

other_cat_feat=["FireplaceQu","PoolQC","Fence","MiscFeature"]

other_cat_const="NA"
for feat in other_cat_feat:
    df_mvi[feat].replace(np.nan,other_cat_const,inplace=True)

for feat in other_cat_feat:
    print(df_mvi[feat].isnull().sum())
    print("----")

# cat
# GarageType       5.378554 - NA
# GarageFinish     5.447071 - NA
# GarageQual       5.447071 - NA
# GarageCond       5.447071 - NA

# num
# GarageYrBlt      5.447071
# GarageCars       0.034258
# GarageArea       0.034258

cat_garage_feat=["GarageType","GarageFinish","GarageQual","GarageCond"]
num_garage_feat=["GarageYrBlt","GarageCars","GarageArea"]

for feat in cat_garage_feat:
    print(df[feat].value_counts())
    print("----")

for feat in num_garage_feat:
    print(df[feat].value_counts())
    print("----")

cat_garage_cont="NA"
for feat in cat_garage_feat:
    df_mvi[feat].replace(np.nan,cat_garage_cont,inplace=True)

num_garage_val=0
for feat in num_garage_feat:
    df_mvi[feat].replace(np.nan,num_garage_val,inplace=True)

# df_mvi[cat_garage_feat].isnull().sum()
# df_mvi[num_garage_feat].isnull().sum()

"""## Feature Transformation

### Numerical to Categorical
"""

## MSSubClass,YearBuilt,YearRemodAdd,GarageYrBlt,MoSold,YrSold
for_num_con = ["MSSubClass","YearBuilt","YearRemodAdd","GarageYrBlt","MoSold","YrSold"]
for feat in for_num_con:
    print(f"{feat}: data type = {df_mvi[feat].dtype}")
```

```
df_mvi["MoSold"]=df_mvi["MoSold"].apply(lambda x: calendar.month_abbr[x])

for feat in for_num_con:
    df_mvi[feat]=df_mvi[feat].astype(str)

for feat in for_num_con:
    print(f"{feat}: data type = {df_mvi[feat].dtype}")

"""### Categorial into Numerical(ordinal objects)"""

ordinal_end_var=[
"ExterQual",
"ExterCond",
"BsmtQual",
"BsmtCond",
"BsmtExposure",
"BsmtFinType1",
"BsmtFinType2",
"HeatingQC",
"KitchenQual",
"FireplaceQu",
"GarageQual",
"GarageCond",
"PoolQC",
"Functional",
"GarageFinish",
"PavedDrive",
"Utilities",
]

"""## Split data"""

len_train=df_train.shape[0]
len_train

X_train=df_encod[:len_train].drop("SalePrice",axis=1)
y_train=df_encod[:len_train]["SalePrice"]
X_test=df_encod[len_train:].drop("SalePrice",axis=1)


print("Shape of X_train",X_train.shape)
print("Shape of y_train",y_train.shape)
print("Shape of X_test",X_test.shape)

sc = StandardScaler()

sc.fit(X_train) # it will learn about mean and std variance
X_train=sc.transform(X_train)
X_test=sc.transform(X_test)

"""## Cross Validation and Model Selection"""

from sklearn.svm import SVR
```

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.neural_network import MLPRegressor

from xgboost import XGBRegressor

svr = SVR()
lr = LinearRegression()
sgdr = SGDRegressor()
knn = KNeighborsRegressor()
gpr = GaussianProcessRegressor()
dtr = DecisionTreeRegressor()
rfr = RandomForestRegressor()
gbr = GradientBoostingRegressor()
xgbr = XGBRegressor()

mlpr = MLPRegressor()

models = {"a":["LinearRegression",lr],
        "b":["SVR",svr],
        "c":["SGDRegressor",sgdr],
        "d":["KNeighborsRegressor",knn],
        "e":["GaussianProcessRegressor",gpr],
        "f":["DecisionTreeRegressor",dtr],
        "g":["GradientBoostingRegressor",gbr],
        "h":["RandomForestRegressor",rfr],
        "i":["XGBRegressor",xgbr],
        "j":["MLPRegressor",mlpr],
        }   # Create a dictionary to store the results

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

def test_model(model, X_train=X_train,y_train=y_train):
    cv = KFold(n_splits=7, random_state=45, shuffle=True)
    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

models_score=[]
for model in models:
    print("Model Name: ",models[model][0])
    score = test_model(models[model][1], X_train, y_train)
    print("Score of Model:",score)
    print("-----------------------")
    models_score.append([models[model][0],score])
```

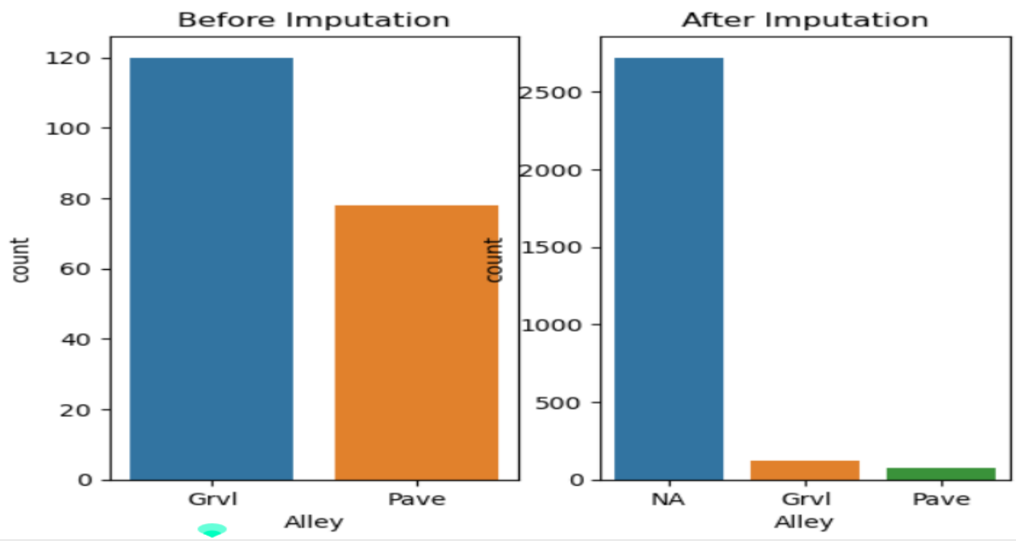
OUTPUT:

```
df.describe()
```

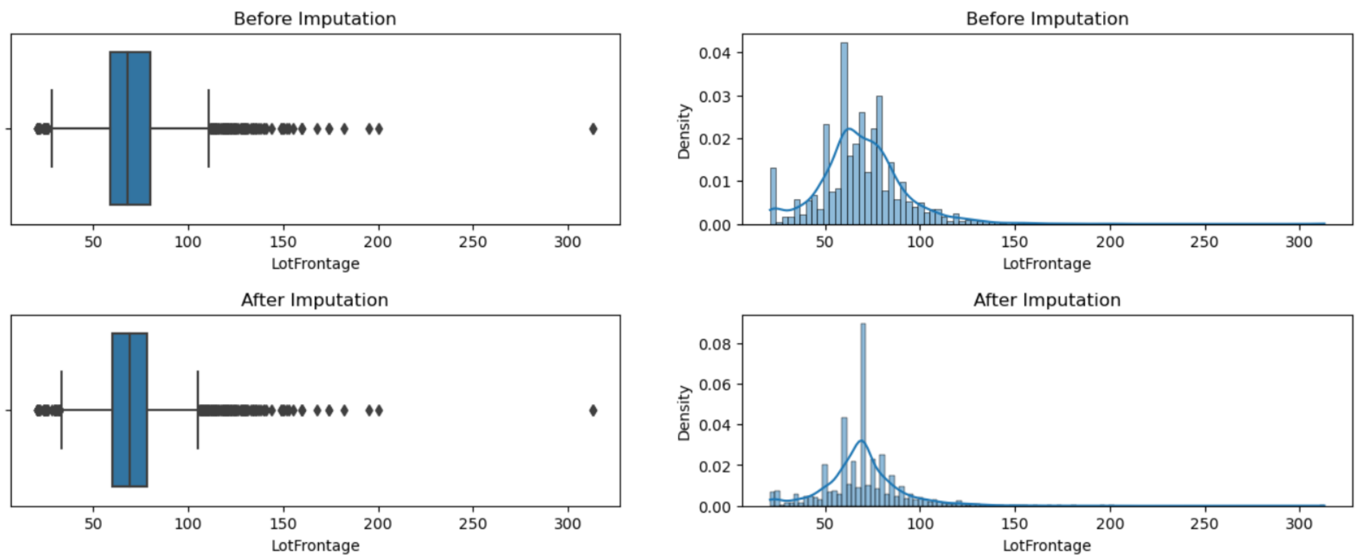
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlr
count	2919.000000	2919.000000	2433.000000	2919.000000	2919.000000	2919.000000	2919.000000	2919.000000	2896.000000	2918.000000	2918.000000	2918.000000	2918.000000	2919.0000
mean	1460.000000	57.137718	69.305795	10168.114080	6.089072	5.564577	1971.312778	1984.264474	102.201312	441.423235	49.582248	560.772104	1051.777587	1159.5817
std	842.787043	42.517628	23.344905	7886.996359	1.409947	1.113131	30.291442	20.894344	179.334253	455.610826	169.205611	439.543659	440.766258	392.3620
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	0.000000	0.000000	334.0000
25%	730.500000	20.000000	59.000000	7478.000000	5.000000	5.000000	1953.500000	1965.000000	0.000000	0.000000	0.000000	220.000000	793.000000	876.0000
50%	1460.000000	50.000000	68.000000	9453.000000	6.000000	5.000000	1973.000000	1993.000000	0.000000	368.500000	0.000000	467.000000	989.500000	1082.0000
75%	2189.500000	70.000000	80.000000	11570.000000	7.000000	6.000000	2001.000000	2004.000000	164.000000	733.000000	0.000000	805.500000	1302.000000	1387.5000
max	2919.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1526.000000	2336.000000	6110.000000	5095.0000

```
df_mvi["Alley"].value_counts()
alley_cont="NA"
df_mvi["Alley"].replace(np.nan,alley_cont,inplace=True) # replace missing value with 'NA'
df_mvi["Alley"].isnull().sum()

# compare before and after imputation
plot_data(df,df_mvi,"Alley")
```




```
oldNewBoxHistPlot(df,df_mvi,"LotFrontage")
```



Numerical to Categorical

```
## MSSubClass,YearBuilt,YearRemodAdd,GarageYrBlt,MoSold,YrSold
for_num_con = ["MSSubClass","YearBuilt","YearRemodAdd","GarageYrBlt","MoSold","YrSold"]
for feat in for_num_con:
    print(f"{feat}: data type = {df_mvi[feat].dtype}")
```

```
MSSubClass: data type = int64
YearBuilt: data type = int64
YearRemodAdd: data type = int64
GarageYrBlt: data type = float64
MoSold: data type = int64
YrSold: data type = int64
```

```
[ ] df_mvi["MoSold"]=df_mvi["MoSold"].apply(lambda x: calendar.month_abbr[x])
```

```
[ ] for feat in for_num_con:
        df_mvi[feat]=df_mvi[feat].astype(str)
```

```
for feat in for_num_con:
    print(f"{feat}: data type = {df_mvi[feat].dtype}")
```

```
MSSubClass: data type = object
YearBuilt: data type = object
YearRemodAdd: data type = object
GarageYrBlt: data type = object
MoSold: data type = object
YrSold: data type = object
```

Split data

+ Code + Text

```
len_train=df_train.shape[0]
len_train
```

```
1460
```

```
[ ] X_train=df_encod[:len_train].drop("SalePrice",axis=1)
    y_train=df_encod[:len_train]["SalePrice"]
    X_test=df_encod[len_train:].drop("SalePrice",axis=1)


    print("Shape of X_train",X_train.shape)
    print("Shape of y_train",y_train.shape)
    print("Shape of X_test",X_test.shape)
```

```
Shape of X_train (1460, 512)
Shape of y_train (1460,)
Shape of X_test (1459, 512)
```

```
[ ] sc = StandardScaler()

    sc.fit(X_train) # it will learn about mean and std variance
    X_train=sc.transform(X_train)
    X_test=sc.transform(X_test)
```

Cross Validation and Model Selection

```
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.neural_network import MLPRegressor

from xgboost import XGBRegressor
```

```
[ ] svr = SVR()
    lr = LinearRegression()
    sgdr = SGDRegressor()
    knn = KNeighborsRegressor()
    gpr = GaussianProcessRegressor()
    dtr = DecisionTreeRegressor()
    rfr = RandomForestRegressor()
    gbr = GradientBoostingRegressor()
    xgbr = XGBRegressor()

    mlpr = MLPRegressor()
```

Cross Validation and Model Selection

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
```

```
[ ] dtr = DecisionTreeRegressor()
    rfr = RandomForestRegressor()
    gbr = GradientBoostingRegressor()
    xgbr = XGBRegressor()
```

```
[ ] models = {
                "f":["DecisionTreeRegressor",dtr],
                "g":["GradientBoostingRegressor",gbr],
                "h":["RandomForestRegressor",rfr],
                "i":["XGBRegressor",xgbr],
                }    # Create a dictionary to store the results
```

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

    def test_model(model, X_train=X_train,y_train=y_train):
        cv = KFold(n_splits=7, random_state=45, shuffle=True)
        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
```

```
models_score=[]
for model in models:
    print("Model Name: ",models[model][0])
    score = test_model(models[model][1], X_train, y_train)
    print("Score of Model:",score)
    print("-----------------------")
    models_score.append([models[model][0],score])
```

```
Model Name:  DecisionTreeRegressor
Score of Model: [0.7008206494020515]
-----------------------
Model Name:  GradientBoostingRegressor
Score of Model: [0.8715666756167462]
-----------------------
Model Name:  RandomForestRegressor
Score of Model: [0.8457476035359204]
-----------------------
Model Name:  XGBRegressor
Score of Model: [0.8582487612757063]
-----------------------
```

```
gbr.fit(X_train,y_train)
```

```
▾ GradientBoostingRegressor
GradientBoostingRegressor()
```

```
[ ] y_pred=gbr.predict(X_test)
```

```
[ ] # y_pred # is a numpy array hence we gonna convert it into dataframe
    y_pred=pd.concat([df_test['Id'],pd.DataFrame(y_pred,columns=['SalePrice'])],axis=1)
    y_pred
```

36	1497	182223.884972
37	1498	165683.328424
38	1499	167733.284069
39	1500	146715.310659
40	1501	165761.689504
41	1502	159338.955452
42	1503	290292.318318
43	1504	228485.048420

CONCLUSION

In this project, we successfully developed a machine learning-based model to predict house prices using various property and location-related features. By implementing advanced preprocessing techniques and leveraging cross-validation, we ensured that the model not only achieved high accuracy but also generalized well to unseen data. Cross-validation played a crucial role in minimizing overfitting and providing reliable performance evaluation across different regression models. Among the models tested, algorithms like Random Forest and XGBoost demonstrated superior performance, highlighting their ability to capture complex patterns in the data. The use of metrics such as RMSE and R^2 Score enabled objective comparison and selection of the best-performing model.

Overall, this project demonstrates the power of data science and machine learning in solving real-world problems like house price prediction. It provides a scalable and accurate solution that can support decision-making in the real estate industry. Future enhancements could include integration of real-time market trends, geospatial analysis, and deep learning techniques for further accuracy improvements.

REFERENCES:

1) **K- Fold Cross Validation**
https://www.kaggle.com/code/satishgunjal/tutorial-k-fold-cross-validation
2) **Scikit-learn: Machine Learning in Python**
https://scikit-learn.org/stable/
3) **Pandas Documentation – Data manipulation and analysis**
https://pandas.pydata.org/docs/
4) **Matplotlib – Data Visualization Library**
https://matplotlib.org/stable/contents.html
5) **Seaborn – Statistical Data Visualization**
https://seaborn.pydata.org/
6) **Gradient Boosting Machine Learning Techniques**
Friedman, J. H. (2001). "Greedy function approximation: A gradient boosting machine." *Annals of Statistics*, 29(5), 1189–1232.
7) **Imbalanced-learn: Handling Imbalanced Datasets**
https://imbalanced-learn.org/stable/
8) **XGBoost Documentation – Scalable and accurate implementation of gradient boosting**
https://xgboost.readthedocs.io/en/stable/
9) **"Predicting Patient No-Shows for Medical Appointments Using Machine Learning Algorithms"**
B. A. Cunha et al., 2020, *International Journal of Computer Applications*
10) **UCI Machine Learning Repository – Healthcare Datasets**
https://archive.ics.uci.edu/ml/index.php
11) **"Machine Learning in Healthcare: A Review of Algorithms and Applications"**
Esteva, A. et al., *Nature Medicine*, 2019
12) **"The Rise of Predictive Analytics in Healthcare"**
Obermeyer, Z., Emanuel, E.J. (2016), *New England Journal of Medicine*, 375:1216-1219