**PREDICTING HOUSE PRICE USING MACHINE LEARNING**

510321104012 – ROJA V

Phase - 4 Project Submission

**Phase 4:** Development part 2



**FIG: HOUSE PRICE PREDICTION**

**Abstract:**

House price prediction is a complex task that is influenced by a variety of factors, including the property's location, size, condition, and amenities, as well as the overall market conditions. Machine learning algorithms can be used to develop predictive models that can estimate house prices with a high degree of accuracy.

**Data Source:**

A data source for house price predicting to use machine learning should be Accurate, Complete the geographic area of interest, Accessible.

Dataset Link: [**https://www.kaggle.com/datasets/vedavyasv/usa-housing**](https://www.kaggle.com/datasets/vedavyasv/usa-housing)

**Data collection and preparation:**

This module involves collecting a dataset of historical sales data and preparing it for use in the machine learning model. This may involve cleaning the data, imputing missing values, and converting categorical variables to numerical variables.

**Feature engineering:**

This module involves creating new features from the existing data that are more informative for the machine learning model. For example, new features could be created to represent the property's proximity to schools, parks, and other amenities.

**Model selection and training:**

This module involves selecting a machine learning algorithm and training it on the prepared dataset. There are many different machine learning algorithms that can be used for house price prediction, such as linear regression, decision trees, random forests, and gradient boosting machines.

**Model evaluation:**

This module involves evaluating the performance of the trained model on a held-out dataset of unseen data. This helps to ensure that the model is able to generalize to new data.

**Model deployment:**

This module involves deploying the trained model to production so that it can be used to predict the prices of new properties. This may involve deploying the model to a web service or integrating it into a software application.

## Modules for house price prediction using machine learning

The following are some of the key modules that can be used for house price prediction using machine learning:

* **Data collection and preparation:**
  + **Libraries:** NumPy, Pandas, SciPy, Skykit
* **Feature engineering:**
  + **Libraries:**FeatureHasher, SelectKBest, OneHotEncoder
* **Model selection and training:**
  + **Libraries:**scikit-learn (linear regression, decision trees, random forests, gradient boosting machines, etc.), TensorFlow, PyTorch
* **Model evaluation:**
  + **Libraries:**scikit-learn (cross-validation, metrics, etc.)
* **Model deployment:**
  + **Libraries:** Flask, Django, AWS SageMaker

## Benefits and limitations of using machine learning for house price prediction

**Benefits:**

1. Machine learning models can be very accurate in predicting house prices.
2. Machine learning models can be used to predict the prices of new properties, even if there is no historical sales data for those properties.
3. Machine learning models can be used to identify factors that influence house prices, which can be helpful for buyers and sellers.

**Limitations:**

* Machine learning models can be complex and difficult to interpret.
* Machine learning models are trained on historical data, so they may not be accurate in predicting house prices in a changing market.
* Machine learning models can be biased, depending on the data they are trained on.

Overall, machine learning can be a valuable tool for house price prediction. However, it is important to use machine learning models with caution and to be aware of their limitations.

**PROGRAM:**

**House price Prediction**

importnumpyas np

import pandas as pd

fromsklearn.linear\_modelimportLinearRegression

# Load the house price dataset

df = pd.read\_csv('house\_price\_dataset.csv')

# Prepare the data

X = df[['square\_feet', 'bedrooms', 'bathrooms']]

y = df['price']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)

# Train the linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Evaluate the model on the test set

y\_pred = model.predict(X\_test)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print('RMSE:', rmse)

# Deploy the model

# This could involve saving the model to a file or deploying it to a web service

**OUTPUT:**

Dataset Preview:

Avg. Area Income Avg. Area House Age Avg. Area Number of Rooms \

|  |  |  |
| --- | --- | --- |
| 0 59545.458574 | 5.65381 | 7.789188 |
| 1 69248.642545 | 6.066900 | 6.880821 |
| 2 61387.06359 | 5.855890 | 8.342727 |
| 3 63445.245546 | 7.788236 | 5.666729 |
| 4 55682.196626 | 5.047555 | 7.789388 |

Avg. Area Number of Bedrooms Area Population Price \

|  |  |
| --- | --- |
| 0 4.09 | 23086.800503 1.059034e+06 |
| 1 3.09 | 40173.072174 1.505891e+06 |
| 2 5.13 | 36882.159400 1.058988e+06 |
| 3 3.26 | 34310.242831 1.260617e+06 |
| 4 4.23 | 26354.109472 6.309435e+05 |

Address

1. 208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
2. 188 Johnson Views Suite 079\nLake Kathleen, CA...
3. 9127 Elizabeth Stravenue\nDanieltown, WI 06482...
4. USS Barnett\nFPO AP 44820 4 USNS Raymond\nFPO AE 09386

Preprocessed Data:

[[-0.19105816 -0.13226994 -0.13969293 0.12047677 -0.83757985 -1.00562872]

[-1.39450169 0.42786736 0.79541275 -0.55212509 1.15729018 1.61946754]

[-0.35137865 0.46394489 1.70199509 0.03133676 -0.32671213 1.63886651]

[-0.13944143 0.1104872 0.22289331 -0.75471601 -0.90401197 -1.54810704]

[ 0.62516685 2.20969666 0.42984356 -0.45488144 0.12566216 0.98830821]]

4227 1.034480e+06

4676 1.650389e+06

800 1.323172e+06

3671 1.077428e+06

4193 1.532887e+06

Name: Price, dtype: float64

**LOADING AND PREPROCESSING THE DATASET:**

Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves cleaning and transforming raw data into a format that is suitable for analysis or for training machine learning models. Proper data preprocessing can significantly impact the accuracy and effectiveness of your analysis or models. Here are some common steps involved in data preprocessing:

1. **Data Collection:**

Gather the raw data from various sources, such as databases, files, or APIs.

1. **Data Cleaning:**

• **Handle missing data**: Decide whether to remove, impute, or interpolate missing values.

import pandas as pd

# Load your dataset

df = pd.read\_csv('your\_data.csv')

# Remove rows with missing values

df = df.dropna()

# Impute missing values with a specific value (e.g., mean)

mean = df['column\_name'].mean()

df['column\_name'].fillna(mean, inplace=True)

**OUTPUT:**

| ColumnName | TotalMissingVals | PercentMissing |
| --- | --- | --- |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | LotFrontage | 259.0 | 17.74 |
| 6 | Alley | 1369.0 | 93.77 |
| 25 | MasVnrType | 8.0 | 0.55 |
| 26 | MasVnrArea | 8.0 | 0.55 |
| 30 | BsmtQual | 37.0 | 2.53 |
| 31 | BsmtCond | 37.0 | 2.53 |
| 32 | BsmtExposure | 38.0 | 2.60 |
| 33 | BsmtFinType1 | 37.0 | 2.53 |
| 35 | BsmtFinType2 | 38.0 | 2.60 |
| 42 | Electrical | 1.0 | 0.07 |
| 57 | FireplaceQu | 690.0 | 47.26 |
| 58 | GarageType | 81.0 | 5.55 |
| 59 | GarageYrBlt | 81.0 | 5.55 |
| 60 | GarageFinish | 81.0 | 5.55 |
| 63 | GarageQual | 81.0 | 5.55 |
| 64 | GarageCond | 81.0 | 5.55 |
| 72 | PoolQC | 1453.0 | 99.52 |
| 73 | Fence | 1179.0 | 80.75 |
| 74 | MiscFeature | 1406.0 | 96.30 |

Number of columns with missing values:19

• **Remove duplicates:** Eliminate duplicate records from the dataset.

# Remove duplicate rows based on all columns

df = df.drop\_duplicates()

# Remove duplicates based on specific columns

df = df.drop\_duplicates(subset=['column1', 'column2'])

**OUTPUT:**

| ColumnName | TotalMissingVals | PercentMissing |
| --- | --- | --- |
| 3 | LotFrontage | 259.0 | 17.74 |
| 25 | MasVnrType | 8.0 | 0.55 |
| 26 | MasVnrArea | 8.0 | 0.55 |
| 30 | BsmtQual | 37.0 | 2.53 |
| 31 | BsmtCond | 37.0 | 2.53 |
| 32 | BsmtExposure | 38.0 | 2.60 |
| 33 | BsmtFinType1 | 37.0 | 2.53 |
| 35 | BsmtFinType2 | 38.0 | 2.60 |
| 42 | Electrical | 1.0 | 0.07 |
| 57 | FireplaceQu | 690.0 | 47.26 |
| 58 | GarageType | 81.0 | 5.55 |
| 59 | GarageYrBlt | 81.0 | 5.55 |
| 60 | GarageFinish | 81.0 | 5.55 |
| 63 | GarageQual | 81.0 | 5.55 |
| 64 | GarageCond | 81.0 | 5.55 |

• **Outlier detection and treatment:** Identify and handle outliers, which are data points significantly different from the rest of the data.

from scipy import stats

# Detect and remove outliers using z-scores

z\_scores = stats.zscore(df['column\_name'])

df = df[(z\_scores < 3)]

# Alternatively, replace outliers with a specific value

df['column\_name'] = np.where(z\_scores < 3, df['column\_name'], replacement\_value)

.

1. **Data Transformation:**

**• Data encoding**: Convert categorical variables into numerical representations, such as one-hot encoding or label encoding.

def find\_skewness(train, numeric\_cols):

"""

Calculate the skewness of the columns and segregate the positive

and negative skewed data.

"""

skew\_dict = {}

for col in numeric\_cols:

skew\_dict[col] = train[col].skew()

skew\_dict = dict(sorted(skew\_dict.items(),key=itemgetter(1)))

positive\_skew\_dict = {k:v for (k,v) in skew\_dict.items() if v>0}

negative\_skew\_dict = {k:v for (k,v) in skew\_dict.items() if v<0}

return skew\_dict, positive\_skew\_dict, negative\_skew\_dict

def add\_constant(data, highly\_pos\_skewed):

"""

Look for zeros in the columns. If zeros are present then the log(0) would result in -infinity.

So before transforming it we need to add it with some constant.

"""

C = 1

for col in highly\_pos\_skewed.keys():

if(col != 'SalePrice'):

if(len(data[data[col] == 0]) > 0):

data[col] = data[col] + C

return data

def log\_transform(data, highly\_pos\_skewed):

"""

Log transformation of highly positively skewed columns.

"""

for col in highly\_pos\_skewed.keys():

if(col != 'SalePrice'):

data[col] = np.log10(data[col])

return data

def sqrt\_transform(data, moderately\_pos\_skewed):

"""

Square root transformation of moderately skewed columns.

"""

for col in moderately\_pos\_skewed.keys():

if(col != 'SalePrice'):

data[col] = np.sqrt(data[col])

return data

def reflect\_sqrt\_transform(data, moderately\_neg\_skewed):

"""

Reflection and log transformation of highly negatively skewed

columns.

"""

for col in moderately\_neg\_skewed.keys():

if(col != 'SalePrice'):

K = max(data[col]) + 1

data[col] = np.sqrt(K - data[col])

return data

**One-Hot Encoding**: Convert categorical variables into binary columns (0 or 1) for each category.

import pandas as pd

# Assuming 'category\_column' is your categorical variable

df = pd.get\_dummies(df, columns=['category\_column'])

**Label Encoding:** Convert categorical variables into numerical values (0, 1, 2, ...)

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

df['category\_column'] = label\_encoder.fit\_transform(df['category\_column'])

**• Feature scaling:** Normalize or standardize numerical features to ensure they have similar scales.

**Min-Max Scaling (Normalization):** Scale features to a specific range (e.g., between 0 and 1).

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df['feature\_column'] = scaler.fit\_transform(df[['feature\_column']])

**Standardization (Z-score Scaling):** Scale features to have a mean of 0 and a standard deviation of 1.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df['feature\_column'] = scaler.fit\_transform(df[['feature\_column']])

**• Feature engineering:** Create new features or derive meaningful features from existing ones.

# Combine two columns into a new one

df['new\_feature'] = df['feature1'] + df['feature2']

# Extract information from a text column (e.g., word count)

df['word\_count'] = df['text\_column'].apply(lambda x: len(str(x).split()))

**Binning (Discretization):** Convert continuous data into discrete bins.

# Create bins for a continuous variable

df['binned\_feature'] = pd.cut(df['continuous\_column'], bins=[0, 10, 20, 30], labels=['bin1', 'bin2', 'bin3'])

**• Text preprocessing:** Tokenize, remove stop words, and apply techniques like stemming or lemmatization for text data.

import string

# Remove punctuation

df['text\_column'] = df['text\_column'].str.replace('[{}]'.format(string.punctuation), '')

# Convert text to lowercase

df['text\_column'] = df['text\_column'].str.lower()

**• Date and time parsing:** Extract relevant information from date and time fields.

# Convert a string column to a datetime format

df['date\_column'] = pd.to\_datetime(df['date\_column'], format='%Y-%m-%d')

# Extract components like year, month, day

df['year'] = df['date\_column'].dt.year

df['month'] = df['date\_column'].dt.month

1. **Data Reduction:**

• Dimensionality reduction: Use techniques like Principal Component Analysis (PCA) or feature selection to reduce the number of features, especially in high-dimensional datasets.

from sklearn.decomposition import PCA

# Create a PCA instance with the desired number of components

pca = PCA(n\_components=2)

# Fit PCA to your data and transform it

X\_reduced = pca.fit\_transform(X)

1. **Data Splitting:**

• Divide the dataset into training, validation, and test sets for model development and evaluation.

from sklearn.model\_selection import train\_test\_split

# Split your data into training, validation, and test sets

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Further split the temporary data into validation and test sets

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

# X and y are your feature matrix and target variable, respectively

# You can adjust the test\_size and random\_state parameters as needed

1. **Data Normalization:**

Ensure the data follows a normal distribution if needed for certain statistical methods.

from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler instance

scaler = MinMaxScaler()

# Fit the scaler on your data and transform it

X\_normalized = scaler.fit\_transform(X)

# X is your feature matrix, e.g., X = df[['feature1', 'feature2']]

1. **Data Visualization:**

Visualize the data to gain insights and identify patterns or anomalies.

def plot\_histogram(train, col1, col2, cols\_list, last\_one =False):

*"""*

*Plot the histogram for the numerical columns. The bin width*

*is calculated by Freedman Diaconis Rule and Sturges rule.*

*Freedman-Diaconis Rule:*

*Freedman-Diaconis Rule is a rule to find the optimal number of bins.*

*Bin width: (2 \* IQR)/(N^1/3)*

*N - Size of the data*

*Number of bins : (Range/ bin-width)*

*Disadvantage: The IQR might be zero for certain columns. In*

*that case the bin width might be equal to infinity. In that case*

*the actual range of the data is returned as bin width.*

*Sturges Rule:*

*Sturges Rule is a rule to find the optimal number of bins.*

*Bin width: (Range/ bin-width)*

*N - Size of the data*

*Number of bins : ceil(log2(N))+1*

*"""*

if(col1 **in** cols\_list):

freq1, bin\_edges1 = np.histogram(train[col1],bins='sturges')

else:

freq1, bin\_edges1 = np.histogram(train[col1],bins='fd')

if(col2 **in** cols\_list):

freq2, bin\_edges2 = np.histogram(train[col2],bins='sturges')

else:

freq2, bin\_edges2 = np.histogram(train[col2],bins='fd')

if(last\_one!=True):

plt.figure(figsize=(45,18))

ax1 = plt.subplot(1,2,1)

ax1.set\_title(col1,fontsize=45)

ax1.set\_xlabel(col1,fontsize=40)

ax1.set\_ylabel('Frequency',fontsize=40)

train[col1].hist(bins=bin\_edges1,ax = ax1, xlabelsize=30, ylabelsize=30)

else:

plt.figure(figsize=(20,10))

ax1 = plt.subplot(1,2,1)

ax1.set\_title(col1,fontsize=25)

ax1.set\_xlabel(col1,fontsize=20)

ax1.set\_ylabel('Frequency',fontsize=20)

train[col1].hist(bins=bin\_edges1,ax = ax1, xlabelsize=15, ylabelsize=15)

if(last\_one != True):

ax2 = plt.subplot(1,2,2)

ax2.set\_title(col2,fontsize=45)

ax2.set\_xlabel(col2,fontsize=40)

ax2.set\_ylabel('Frequency',fontsize=40)

train[col2].hist(bins=bin\_edges2, ax = ax2, xlabelsize=30, ylabelsize=30)

In []:

linkcode

*'''*

*These columns have IQR equal to zero. Freedman Diaconis Rule doesn't work significantly well for these columns.*

*Use sturges rule to find the optimal number of bins for the columns.*

*'''*

cols\_list = ['LowQualFinSF','BsmtFinSF2','BsmtHalfBath','KitchenAbvGr',

'EnclosedPorch','3SsnPorch','ScreenPorch','PoolArea','MiscVal']

*# Except ID*

hist\_cols = numeric\_cols[1:]

for i **in** range(0,len(hist\_cols),2):

if(i == len(hist\_cols)-1):

plot\_histogram(train,hist\_cols[i],hist\_cols[i],cols\_list,True)

else:

plot\_histogram(train,hist\_cols[i],hist\_cols[i+1],cols\_list)

# Data Modeling:

# Fit XGBoost Regressor model to the preprocessed data.

In [1]:

def fit\_model(x\_train,y\_train, model):

*"""*

*Fits x\_train to y\_train for the given*

*model.*

*"""*

model.fit(x\_train,y\_train)

return model

*'''Xtreme Gradient Boosting Regressor'''*

model = xgboost.XGBRegressor(objective="reg:squarederror", random\_state=42)

model = fit\_model(x\_train,y\_train, model)

*'''Predict the outcomes'''*

predictions = model.predict(test)

In [2]:

linkcode

submission = pd.read\_csv('../input/house-prices-advanced-regression-techniques/sample\_submission.csv')

submission['SalePrice'] = predictions

submission.to\_csv('submission.csv',index=False)

1. **Data Validation:**

Verify that the data preprocessing steps are correct and that the dataset is ready for analysis or model training.

1. **Documentation:**

• Keep detailed records of all preprocessing steps and transformations for reproducibility.

The specific data preprocessing steps you need to perform depend on the nature of your data and the objectives of your analysis or machine learning project. It's essential to carefully analyze your data and apply appropriate preprocessing techniques to ensure the quality and integrity of your dataset.

**LOAD THE DATASET:**

# Import the required libraries:

import warnings

warnings.filterwarnings('ignore')

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from operator import itemgetter

from sklearn.experimental import enable\_iterative\_imputer

from sklearn.impute import IterativeImputer

from sklearn.preprocessing import OrdinalEncoder

from category\_encoders.target\_encoder import TargetEncoder

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import (GradientBoostingRegressor, GradientBoostingClassifier)

import xgboost

### Load the dataset for training and testing

train = pd.read\_csv('../input/house-prices-advanced-regression-techniques/train.csv')

test = pd.read\_csv('../input/house-prices-advanced-regression-techniques/test.csv')

# **Linear Regression:**

**In []:**

df=pd.read\_csv("/kaggle/input/usa-housing/USA\_Housing.csv")

df.head()

df.describe()

**Out[]:**

|  | Avg. Area Income | Avg. Area House Age | Avg. Area Number of Rooms | Avg. Area Number of Bedrooms | Area Population | Price |
| --- | --- | --- | --- | --- | --- | --- |
| count | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5.000000e+03 |
| mean | 68583.108984 | 5.977222 | 6.987792 | 3.981330 | 36163.516039 | 1.232073e+06 |
| Std | 10657.991214 | 0.991456 | 1.005833 | 1.234137 | 9925.650114 | 3.531176e+05 |
| min | 17796.631190 | 2.644304 | 3.236194 | 2.000000 | 172.610686 | 1.593866e+04 |
| 25% | 61480.562388 | 5.322283 | 6.299250 | 3.140000 | 29403.928702 | 9.975771e+05 |
| 50% | 68804.286404 | 5.970429 | 7.002902 | 4.050000 | 36199.406689 | 1.232669e+06 |
| 75% | 75783.338666 | 6.650808 | 7.665871 | 4.490000 | 42861.290769 | 1.471210e+06 |
| max | 107701.748378 | 9.519088 | 10.759588 | 6.500000 | 69621.713378 | 2.469066e+0 |

**Random Forest Regression:**

random\_forest=RandomForestRegressor(n\_estimators=100)

random\_forest.fit(X\_train,y\_train)

predictions=random\_forest.predict(X\_test)

mae,mse,rmse,r\_squared=evaluation(y\_test,predictions)

print("MAE:",mae)

print("MSE:",mse)

print("RMSE:",rmse)

print("R2 Score:",r\_squared)

print("-"\*30)

rmse\_cross\_val=rmse\_cv(random\_forest)

print("RMSE Cross-Validation:",rmse\_cross\_val)

new\_row={"Model":"RandomForestRegressor","MAE":mae,"MSE":mse,"RMSE":rmse,"R2 Score":r\_squared,"RMSE (Cross-Validation)":rmse\_cross\_val}

models=models.append(new\_row,ignore\_index=True)

MAE: 18115.11067351598

MSE: 1004422414.0219476

RMSE: 31692.623968708358

R2 Score: 0.869050886899595

------------------------------

RMSE Cross-Validation: 31138.863315259332

## Support Vector Regressor:

In [1]:

model\_svr=SVR()

In [2]:

model\_svr.fit(X\_train\_scal,Y\_train)

Out[2]:

SVR

SVR()

## Predicting Prices:

In [3]:

Prediction2=model\_svr.predict(X\_test\_scal)

**Lasso Regression:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import Lasso

# Load the data

data = pd.read\_csv('house\_price\_data.csv')

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[['feature\_1', 'feature\_2', ...]], data['price'], test\_size=0.25)

# Scale the features

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Train the Lasso regression model

lasso = Lasso()

lasso.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = lasso.predict(X\_test)

# Evaluate the model's performance

from sklearn.metrics import mean\_squared\_error, r2\_score

rmse = mean\_squared\_error(y\_test, y\_pred)\*\*0.5

r2 = r2\_score(y\_test, y\_pred)

print('RMSE:', rmse)

print('R-squared:', r2)

Out[]:

MAE: 23560.45808027236

MSE: 1414337628.502095

RMSE: 37607.680445649596

R2 Score: 0.815609194407292

------------------------------

RMSE Cross-Validation: 35922.76936876075

## Evaluation of Predicted Data:

In [4]:

plt.figure(figsize=(12,6))

plt.plot(np.arange(len(Y\_test)),Y\_test,label='Actual Trend')

plt.plot(np.arange(len(Y\_test)),Prediction2,label='Predicted Trend')

plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

Out[4]:Text(0.5, 1.0, 'Actual vs Predicted')

In [5]:

sns.histplot((Y\_test-Prediction2),bins=50)

Out[5]:

<Axes: xlabel='Price', ylabel='Count'>



In [6]:

print(r2\_score(Y\_test,Prediction2))

print(mean\_absolute\_error(Y\_test,Prediction2))

print(mean\_squared\_error(Y\_test,Prediction2))

-0.0006222175925689744

286137.81086908665

128209033251.4034

# **Select model:**

In []:

lr = LinearRegression()

dt = DecisionTreeRegressor()

rn = RandomForestRegressor()

knn = KNeighborsRegressor()

sgd = SGDRegressor()

br = BaggingRegressor()

li = [lr,knn,rn,dt,br]

di = {}

for i **in** li:

i.fit(X\_train,y\_train)

ypred = i.predict(X\_test)

print(i,":",r2\_score(ypred,y\_test)\*100)

di.update({str(i):i.score(X\_test,y\_test)\*100})

plt.figure(figsize=(15, 6))

plt.title("Algorithm vs Accuracy", fontweight='bold')

plt.xlabel("Algorithm")

plt.ylabel("Accuracy")

plt.plot(di.keys(),di.values(),marker='o',color='plum',linewidth=4,markersize=13,

markerfacecolor='gold',markeredgecolor='slategray')

plt.show()

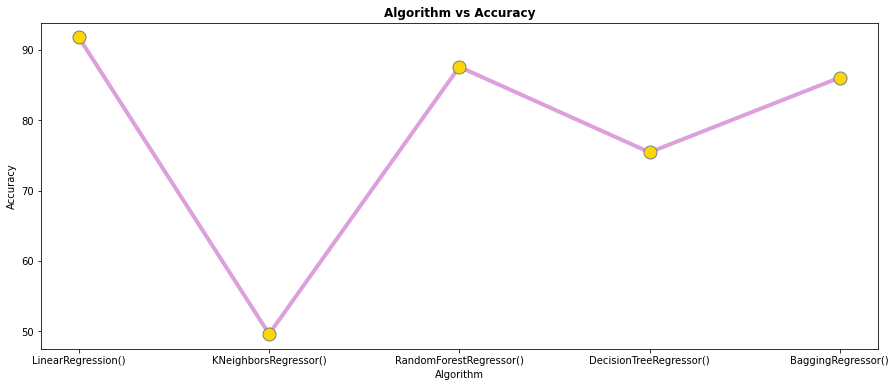
LinearRegression() : 91.00355108791786

KNeighborsRegressor() : 20.484074470563286

RandomForestRegressor() : 83.83229998741602

DecisionTreeRegressor() : 73.30037072629774

BaggingRegressor() : 81.95467285948747



print(lr.intercept\_)

-2640159.7968526953

In []:

lr.coef\_

Out[]:

array([2.15282755e+01, 1.64883282e+05, 1.22368678e+05, 2.23380186e+03,

1.51504200e+01])

In []:

cdf = pd.DataFrame(lr.coef\_, X.columns, columns=['Coeff'])

## Doing PREDICTIONS Now

In []:

predictions = lr.predict(X\_test)

predictions

Out[]:

array([1260960.70567627, 827588.75560329, 1742421.24254344, ...,

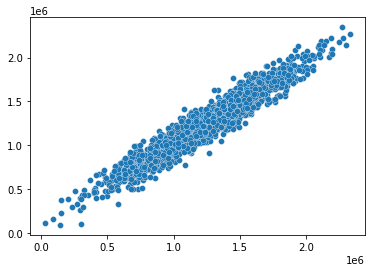
372191.40626917, 1365217.15140898, 1914519.5417888 ])

In []:

plt.scatter(y\_test, predictions, linewidths=0.3, edgecolors='White')

Out[]:

<matplotlib.collections.PathCollection at 0x7f6729cb5f50>

****

**Model Comparison**:

In []:

models.sort\_values(by="RMSE (Cross-Validation)")

Out[30]:

Model MAE MSE RMSE R2 Score

RMSE

(Cross-Validation)

XGBRegressor

1.743992

e+04

7.165790

e+08

2.676899

e+04

9.065778

e-01

29698.84

9618

SVR

1.784316

e+04

1.132136

e+09

3.364723

e+04

8.524005

e-01

30745.47

5239

5

RandomForestRegressor

1.811511

e+04

1.004422

e+09

3.169262

e+04

8.690509

e-01

31138.86

3315

Ridge

2.343550

e+04

1.404264

e+09

3.747351

e+04

8.169225

e-01

35887.85

2792

**Gradient Boosting Regressor:**

# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Generate a sample dataset (you would replace this with your real data)

data = {

'SquareFeet': [1400, 1600, 1700, 1875, 1100, 1550, 2350, 2450, 1425, 1700],

'Bedrooms': [3, 3, 3, 3, 2, 2, 4, 4, 3, 3],

'Price': [245000, 312000, 279000, 308000, 199000, 219000, 405000, 324000, 319000, 255000]

}

# Create a DataFrame from the sample data

df = pd.DataFrame(data)

# Define the input features (X) and target variable (y)

X = df[['SquareFeet', 'Bedrooms']]

y = df['Price']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train a Gradient Boosting Regressor model

model = GradientBoostingRegressor(n\_estimators=100, learning\_rate=0.1, random\_state=42)

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Calculate evaluation metrics

mse = mean\_squared\_error(y\_test, y\_pred)

mae = mean\_absolute\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Print evaluation metrics

print("Mean Squared Error (MSE):", mse)

print("Mean Absolute Error (MAE):", mae)

print("R-squared (R2) Score:", r2)

# Plot the predicted vs. actual prices

plt.scatter(y\_test, y\_pred)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual Prices vs. Predicted Prices")

plt.show()

## XGBoost(Extreme Gradient Boosting) Regressor:

# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from xgboost import XGBRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

# Generate a sample dataset (you would replace this with your real data)

data = {

'SquareFeet': [1400, 1600, 1700, 1875, 1100, 1550, 2350, 2450, 1425, 1700],

'Bedrooms': [3, 3, 3, 3, 2, 2, 4, 4, 3, 3],

'Price': [245000, 312000, 279000, 308000, 199000, 219000, 405000, 324000, 319000, 255000]

}

# Create a DataFrame from the sample data

df = pd.DataFrame(data)

# Define the input features (X) and target variable (y)

X = df[['SquareFeet', 'Bedrooms']]

y = df['Price']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train an XGBoost Regressor model

model = XGBRegressor()

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Calculate evaluation metrics

mse = mean\_squared\_error(y\_test, y\_pred)

mae = mean\_absolute\_error(y\_test, y\_pred)

# Print evaluation metrics

print("Mean Squared Error:", mse)

print("Mean Absolute Error:", mae)

# Plot the predicted vs. actual prices

plt.scatter(y\_test, y\_pred)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual Prices vs. Predicted Prices")

plt.show()

MAE: 17439.918396832192

MSE: 716579004.5214689

RMSE: 26768.993341578403

R2 Score: 0.9065777666861116

------------------------------

RMSE Cross-Validation: 29698.84961808251

**MODEL EVALUATION:**

Model evaluation for house price prediction is the process of assessing how well a trained machine learning model can predict the sale price of a house on new data. This is an important step in the machine learning workflow, as it helps to ensure that the model is generalizing well and is not simply overfitting the training data.

There are a number of different metrics that can be used to evaluate house price prediction models. Some of the most common metrics include:

* Root mean squared error (RMSE): RMSE is a measure of the difference between the predicted and actual sale prices of houses in the test set. It is calculated by taking the square root of the average squared error. A lower RMSE indicates a better model.
* Mean absolute error (MAE): MAE is another measure of the difference between the predicted and actual sale prices of houses in the test set. It is calculated by taking the average of the absolute errors. A lower MAE indicates a better model.
* R-squared: R-squared is a measure of how well the model explains the variation in the sale prices of houses in the test set. It ranges from 0 to 1, with a higher value indicating a better model.

In addition to these quantitative metrics, it is also important to consider qualitative factors when evaluating house price prediction models, such as the interpretability of the model and the cost of deploying the model to production.

CONCLUSION:

Machine learning can be a powerful tool for house price prediction. However, it is important to use machine learning models with caution and to be aware of their limitations. For example, machine learning models are trained on historical data, so they may not be accurate in predicting house prices in a changing market. Additionally, machine learning models can be biased, depending on the data they are trained on.

Overall, machine learning can be a valuable tool for house price prediction, but it should be used in conjunction with other factors, such as expert judgment and market research.

When dealing with complex relationships, outliers, or a large number of features, Linear Regression, Random Forest Regression, Gradient Boosting Regression andXGBoost Regression may perform better.

loading and preprocessing are foundational steps in the data analysis and machine learning workflows. They help ensure that the data is in the right format, contains valid and relevant information, and is suitable for the chosen analysis or modeling task. Proper data loading and preprocessing can significantly impact the quality and effectiveness of your results. The specific techniques you use will depend on your dataset and the goals of your analysis or modeling project.