AI_PHASE_2

NM_PROJECT

VISHVAJITH V
61772221T310
11TH OCTOBER 2023

PREDICTING HOUSE PRICES USING MACHINE LEARNING AND INNOVATION

Introduction :

- Machine learning involves training a computer to recognize patterns and make predictions based on data.
- In the case of house price prediction, we can use historical data on various features of a house, such as its location, size, and amenities, to train a machine-learning model.
- Once the model is trained, it can analyse new data on a given house and make a prediction of its market value.

Algorithm:

- STEP 1 : Import the required libraries and modules, including pandas for data manipulation, scikit-learn for machine learning algorithms, and Linear Regression for the linear regression model.
- STEP 2 : Loading the required dataset with pd.read_csv and select the features we want to use for prediction (e.g., bedrooms, bathrooms, sqft_living, sqft_lot, floors, and zip code), as well as the target variable (price).
- STEP 3 : Split the data into a training set and a test set using the train_test_split function, with 80% of the data used for training and 20% for testing.
- STEP 4: Create an instance of the linear regression model using Linear Regression(). We then perform the model training by calling the function fit() with the training data.
- STEP 5: Demonstrate how to predict the price of a new house by creating a new data frame new house with the features of the house. We pass this data frame to the model's prediction function to obtain the predicted price.

Coding:

Code :

import pandas as pd   # (To load the Data frame)

import matplotlib.pyplot as p lt   #(To visualize the data features i.e. barplot)

import seaborn as sns   #(To see the correlation between features using heatmap)

dataset = pd.read_excel("HousePricePrediction.xlsx")

Printing first 5 records of the dataset

print(dataset.head(5))

Output :

```
   MSSubClass MSZoning  LotArea LotConfig BldgType  OverallCond  YearBuilt
0          60       RL     8450    Inside     1Fam            5       2003
1          20       RL     9600       FR2     1Fam            8       1976
2          60       RL    11250    Inside     1Fam            5       2001
3          70       RL     9550    Corner     1Fam            5       1915
4          60       RL    14260       FR2     1Fam            5       2000

   YearRemodAdd Exterior1st  BsmtFinSF2  TotalBsmtSF  SalePrice
0          2003     VinylSd         0.0        856.0   208500.0
1          1976     MetalSd         0.0       1262.0   181500.0
2          2002     VinylSd         0.0        920.0   223500.0
3          1970     Wd Sdng         0.0        756.0   140000.0
4          2000     VinylSd         0.0       1145.0   250000.0
```

Data Preprocessing

Now, we categorize the features depending on their datatype (int, float, object) and then calculate the number of them.

Code :

dataset. Shape

obj = (dataset. Types == 'object')

object_cols = list(obj[obj].index)

print("Categorical variables:",len(object_cols))

int_ = (dataset. Types == 'int')

num_cols = list(int_[int_].index)

print("Integer variables:",len(num_cols))

fl = (dataset. Types == 'float')

fl_cols = list(fl[fl].index)

print("Float variables:",len(fl_cols))

Output :

Categorical variables : 4

Integer variables : 6

Float variables : 3

Exploratory Data Analysis :

- EDA refers to the deep analysis of data so as to discover different patterns and spot anomalies.
- Before making inferences from data it is essential to examine all your variables.
- So here let's make a heatmap using seaborn library.

Code:

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

sns.heatmap(dataset.corr(),cmap = 'BrBG',fmt = '.2f',linewidths = 2,annot = True)

unique_values = []

for col in object_cols:

unique_values.append(dataset[col].unique().size)

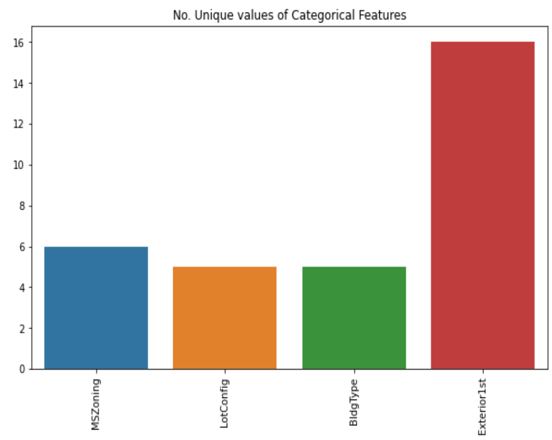
plt.figure(figsize=(10,6))

plt.title('No. Unique values of Categorical Features')

plt.xticks(rotation=90)

sns.barplot(x=object_cols,y=unique_values)

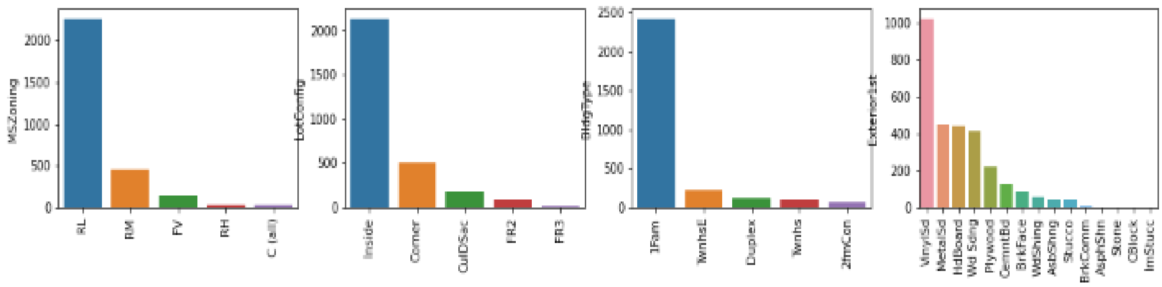
Output :



Code :

```
plt.figure(figsize=(18, 36))

plt.title('Categorical Features: Distribution')

plt.xticks(rotation=90)

index = 1

for col in object_cols:

        y = dataset[col].value_counts()

        plt.subplot(11, 4, index)

        plt.xticks(rotation=90)

        sns.barplot(x=list(y.index), y=y)

        index += 1
```

Output :



Data Cleaning :

- Data Cleaning is the way to improvise the data or remove incorrect, corrupted or irrelevant data.
- As in our dataset, there are some columns that are not important and irrelevant for the model training. So, we can drop that column before training.
- There are 2 approaches to dealing with empty/null values
    1. We can easily delete the column/row (if the feature or record is not much important).

    2. Filling the empty slots with mean/mode/0/NA/etc. (depending on the dataset requirement).

- As Id Column will not be participating in any prediction. So we can Drop it.

Code :

dataset. Drop(['Id'],axis=1,inplace=True)

dataset ['Sale Price'] = dataset['Sale Price'].fillna(dataset['Sale Price'].mean())

new_dataset = dataset.dropna()   #(Drop records with null values)

new_dataset.isnull().sum()   #(Checking features which have null values in new data frame)

Output :

```
MSSubClass      0
MSZoning        0
LotArea         0
LotConfig       0
BldgType        0
OverallCond     0
YearBuilt       0
YearRemodAdd    0
Exterior1st     0
BsmtFinSF2      0
TotalBsmtSF     0
SalePrice       0
dtype: int64
```

Label categorical features

- One hot Encoding is the best way to convert categorical data into binary vectors.
- This maps the values to integer values. By using One Hot Encoder, we can easily convert object data into int.
- So for that, firstly we have to collect all the features which have the object datatype. To do so, we will make a loop.

Code :

from sklearn.preprocessing import OneHotEncoder

s = (new_dataset.dtypes == 'object')

object_cols = list(s[s].index)

print("Categorical variables:")

print(object_cols)

print('No. of. categorical features: ', len(object_cols))

Output :

```
Categorical variables:
['MSZoning', 'LotConfig', 'BldgType', 'Exterior1st']
No. of. categorical features:  4
```

Training and Testing

- X and Y splitting (i.e. Y is the SalePrice column and the rest of the other columns are X)

Code :

```
OH_encoder = OneHotEncoder(sparse=False)

OH_cols = pd.DataFrame(OH_encoder.fit_transform(new_dataset[object_cols]))

OH_cols.index = new_dataset.index

OH_cols.columns = OH_encoder.get_feature_names()

df_final = new_dataset.drop(object_cols, axis=1)

df_final = pd.concat([df_final, OH_cols], axis=1)

from sklearn.metrics

import mean_absolute_error

from sklearn.model_selection

 import train_test_split

X = df_final.drop(['SalePrice'], axis=1)     # Split the training set into

Y = df_final['SalePrice']     # training and validation set

X_train, X_valid = train_test_split(X, train_size=0.8, test_size=0.2, random_state=0)

Y_train, Y_valid = train_test_split(Y, train_size=0.8, test_size=0.2, random_state=0)
```

Model and Accuracy

- As we have to train the model to determine the continuous values, so we will be using these regression models.
    1. SVM-Support Vector Machine
    2. Random Forest Regressor

3. Linear Regressor

- And To calculate loss we will be using the mean_absolute_percentage_error module.
- It can easily be imported by using sklearn library.
- SVM – Support vector Machine
  -> SVM can be used for both regression and classification model. It finds the hyperplane in the n-dimensional plane. To read more about svm refer this.

Code :

```
from sklearn import svm

from sklearn.svm import SVC

from sklearn.metrics import mean_absolute_percentage_error

model_SVR = svm.SVR()

model_SVR.fit(X_train,Y_train)

Y_pred = model_SVR.predict(X_valid)

print(mean_absolute_percentage_error(Y_valid, Y_pred))

from sklearn.ensemble import RandomForestRegressor

model_RFR = RandomForestRegressor(n_estimators=10)

model_RFR.fit(X_train, Y_train)

Y_pred = model_RFR.predict(X_valid)

mean_absolute_percentage_error(Y_valid, Y_pred)
```

Output :

0.18705129

0.1929469

0.187416838

Dataset:

https://docs.google.com/spreadsheets/d/1caaR9pT24GNmq3rDQpMiIMJrmiTGarbs/edit?usp=sharing&ouid=115253717745408081083&rtpof=true&sd=true

Conclusion :

- In conclusion, using machine learning in Python is a powerful tool for predicting house prices.
- By gathering and cleaning data, visualizing patterns, and training and evaluating our models, we can make informed decisions in the dynamic world of real estate.
- By leveraging advanced algorithms and data analysis, we can make accurate predictions and inform decision-making processes.
- This approach empowers buyers, sellers, and investors to make informed choices in a dynamic and competitive market, ultimately maximizing their opportunities and outcomes.