**DAY 5: Training ML model for regression problems**

**1.Choose best open source dataset works for regression problem**

**2. ⁠Train the mL model and observe the result**

**3. ⁠explanations needed for each parameter used in mL model why it’s been used**

**1)Ames house price prediction model:**

**Dataset from:Kaggle**

[**https://www.kaggle.com/datasets/prevek18/ames-housing-dataset**](https://www.kaggle.com/datasets/prevek18/ames-housing-dataset)

**I am currently working on Google colab with python3 runtime type and CPU hardware accelerator for this problem**

**CODE step by step parameter explanation and why it is used**

**Step 1:IMPORTING NECESSARY MODULES FROM THE LIBRARIES**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

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

**from sklearn.preprocessing import StandardScaler**

**from sklearn.ensemble import RandomForestRegressor**

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

* **import pandas as pd :**

**this is used to import pandas library**

* It helps you read, manipulate, and analyze structured data (like CSV files or tables).
* We use it to load the dataset, check for missing values, drop or fill columns, and create features.
* **Import numpy as np**

 Provides efficient operations on **arrays**, **matrices**, and **mathematical functions**.

 Helps in numerical operations like calculating RMSE (np.sqrt()), handling missing values, etc.

* **Import matplotlib.pyplot as plt**
* To create visualizations like bar charts, line plots, histograms, etc.
* Helps in understanding **feature importance**, data distributions, trends, etc.
* **Import seaborn as sns**

 Makes it easier to create **beautiful statistical visualizations** (e.g., heatmaps, correlation matrices).

 Great for **EDA (Exploratory Data Analysis)**.

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

We train our model on one part of the data (training) and test its performance on unseen data (testing).

**from sklearn.preprocessing import StandardScaler**

To **normalize or scale** features so that they have **zero mean and unit variance**.

**Why it's used**:

* Some algorithms are sensitive to scale (e.g., linear models, distance-based models).
* Helps improve model performance and convergence.

**from sklearn.ensemble import RandomForestRegressor**

**to use Random forest model in the problem**

It's a powerful and flexible ensemble model that combines multiple decision trees.

Works well on both linear and non-linear data.

Used here to predict house prices based on features.

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

**to evaluate how good the model’s predictions are.**

**mean\_squared\_error: Measures the average squared difference between actual and predicted values.**

**r2\_score: Measures how well the predictions explain the variance in the target. R² = 1 is perfect, 0 is no better than average.**

**STEP 2:INPORTING THE DATASET FRM KAGGLE USING PANDAS**

**from google.colab import files**

**uploaded = files.upload()**

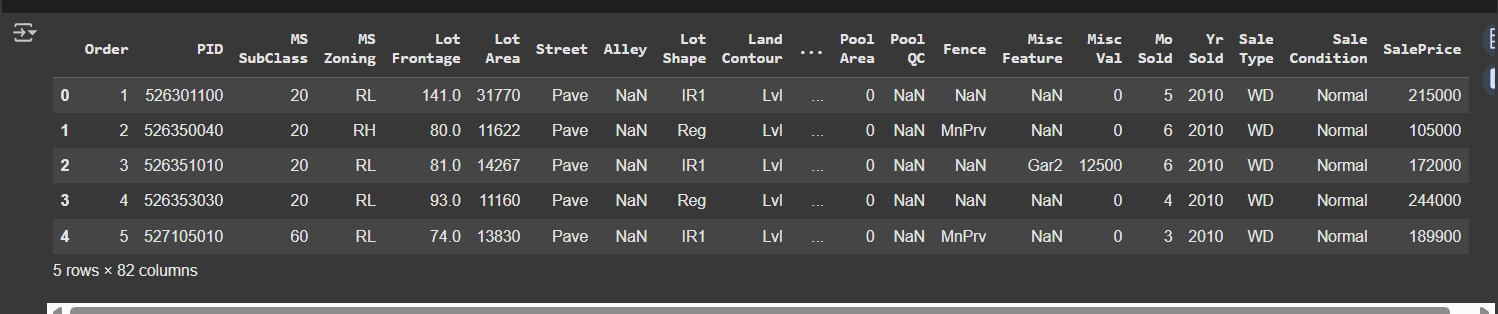
Opens a file upload window so we can upload the AmesHousing.csv file from our computer.

**import pandas as pd**

**data = pd.read\_csv("AmesHousing.csv")**

**print(data.shape)**

**data.head()**

****

pd.read\_csv(): Loads the CSV file into a DataFrame.

data.shape: Shows the number of rows and columns.

data.head(): Displays the first 5 rows of the dataset to get an idea of its structure.

**Step 3:Check for missing values and plot the distribution of target variable**

missing = data.isnull().sum().sort\_values(ascending=False)

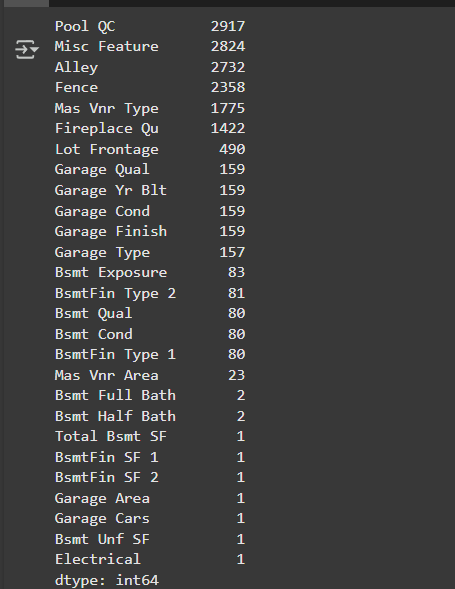
print(missing[missing > 0])

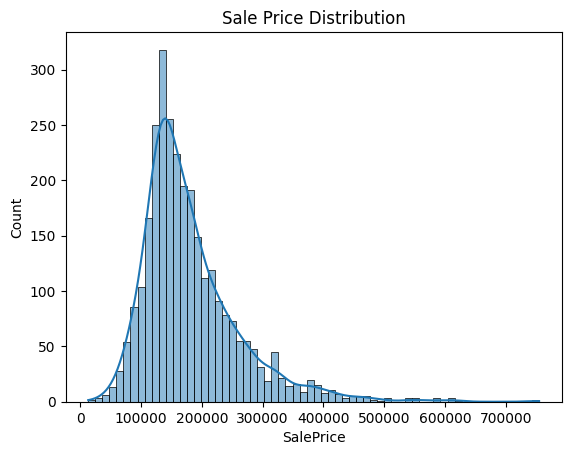
data.dtypes.value\_counts()

sns.histplot(data['SalePrice'], kde=True)

plt.title("Sale Price Distribution")

plt.show()





* **isnull().sum()**: Counts how many missing values exist in each column.
* **missing > 0**: Filters only those columns that have missing values.
* **sort\_values**: Sorts by the number of missing values in descending order.

**STEP4:DROP COLUMNS WITH MANY MISSING VALUES**

**columns\_to\_drop = ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu']**

**existing\_cols = [col for col in columns\_to\_drop if col in data.columns]**

**data = data.drop(columns=existing\_cols)**

These columns typically have 90%+ missing values and aren’t very useful for prediction.

We check if the column exists before dropping to avoid errors

**STEP 5:Fill remaining missing values**

**for column in data.columns:**

**if data[column].dtype == 'object':**

**data[column] = data[column].fillna(data[column].mode()[0])**

**else:**

**data[column] = data[column].fillna(data[column].median())**

Categorical columns (object): Fill missing values with the most frequent value (mode).

Numerical columns: Fill missing values with the median, which is less sensitive to outliers than the mean.

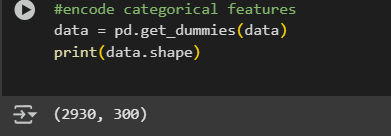
**STEP 6:ENCODE CATEGORICAL VALUES**

**data = pd.get\_dummies(data)**

**print(data.shape)**

pd.get\_dummies converts categorical values to numerical dummy values(one hot encoding)

We need this because ML models require numerical inputs



**STEP 7:SPLIT FEATURES AND TARGET**

**X = data.drop("SalePrice", axis=1)**

**y = data["SalePrice"]**

x:all features and parameters used for house price prediction except the target variable

y:the target variable that is to be tested with the predicted house price value

**STEP 8:SPLIT INTO TRAINING AND TESTING SETS**

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

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

**Purpose**: Split data into 80% training and 20% testing.

**Training set**: Used to train the model.

**Testing set**: Used to evaluate how well the model performs on new data.

**random\_state**=42 ensures reproducibility (same split every time).

**STEP 9:FEATURE SCALING**

**from sklearn.preprocessing import StandardScaler**

**scaler = StandardScaler()**

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

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

Many ML models (like linear models or distance-based ones) perform better if features are scaled.

fit\_transform(): Learns the mean and std from training data and applies scaling.

transform(): Applies the same scaling to the test data (important to prevent data leakage).

**STEP 10:TRAIN THE MODEL**

**from sklearn.ensemble import RandomForestRegressor**

**model = RandomForestRegressor(n\_estimators=100, random\_state=42)**

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

**RandomForestRegressor**: A powerful ensemble model using multiple decision trees.

**n\_estimators=100**: Uses 100 trees.

**fit()**: Trains the model using scaled features and the sale prices.

**STEP 11:EVALUATING THE MODEL**

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

**import numpy as np**

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

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

**rmse = np.sqrt(mse)**

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

**print(f"RMSE: {rmse:.2f}")**

**print(f"R² Score: {r2:.2f}")**

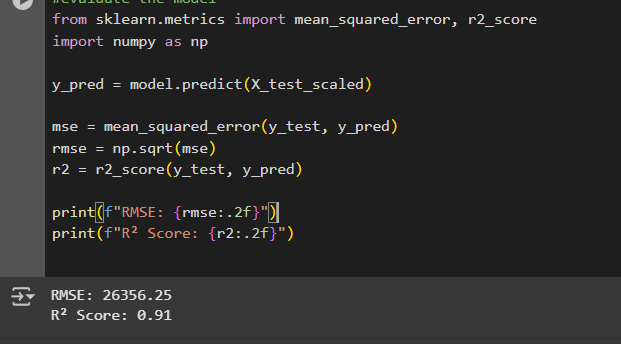
predict(): Predicts sale prices for the test set.

mean\_squared\_error: Measures the average squared difference

between predicted and actual prices.

rmse: Square root of MSE; more interpretable (in dollars).

r2\_score: Tells how much of the variance in SalePrice is explained by the model. 1.0 is perfect, 0 is bad.



The **Ames Housing Price Prediction** model is widely regarded as one of the best ways to learn and practice regression in machine learning—and for good reason. It’s based on a real-world problem that’s easy to understand: predicting house prices. This makes it incredibly relatable and practical, whether you're a student, a data scientist, or just someone curious about how machine learning works in the real world. The dataset, created by Dean De Cock, is a modern, cleaner alternative to the older Boston Housing dataset, and it has become a favorite in courses, competitions, and tutorials.

What really makes this dataset stand out is the variety and richness of its features. It contains over 80 different columns, covering everything from the size of the house and the year it was built, to the quality of the materials and the neighborhood it’s in. Because these features include numbers, ordered categories (like ratings from 1 to 10), and names of places or styles, you get to practice a wide range of preprocessing techniques—like handling missing values, encoding categories, and scaling features. These are all skills that are super important in any machine learning project.

It’s also a great dataset for trying out different types of regression models. Whether you’re starting with something simple like Linear Regression or diving into more powerful techniques like Random Forests, XGBoost, or Neural Networks, the Ames data gives you a balanced, flexible playground to test and compare your models. It’s big enough to be meaningful, but small enough to run smoothly even on a basic laptop or in Google Colab.

Beyond just building models, the Ames dataset lets you explore more advanced concepts too—like cross-validation, hyperparameter tuning, feature engineering, and model evaluation using metrics like RMSE and R² score. In other words, it supports the entire machine learning pipeline from start to finish.

In short, the Ames Housing dataset is popular not just because it’s clean and manageable, but because it mirrors a real-life problem, gives you exposure to essential ML techniques, and helps you build confidence in developing and evaluating regression models. It’s an ideal learning tool for anyone who wants to move beyond theory and into practical, hands-on experience.