SEMMA Project Report — Predicting House Prices

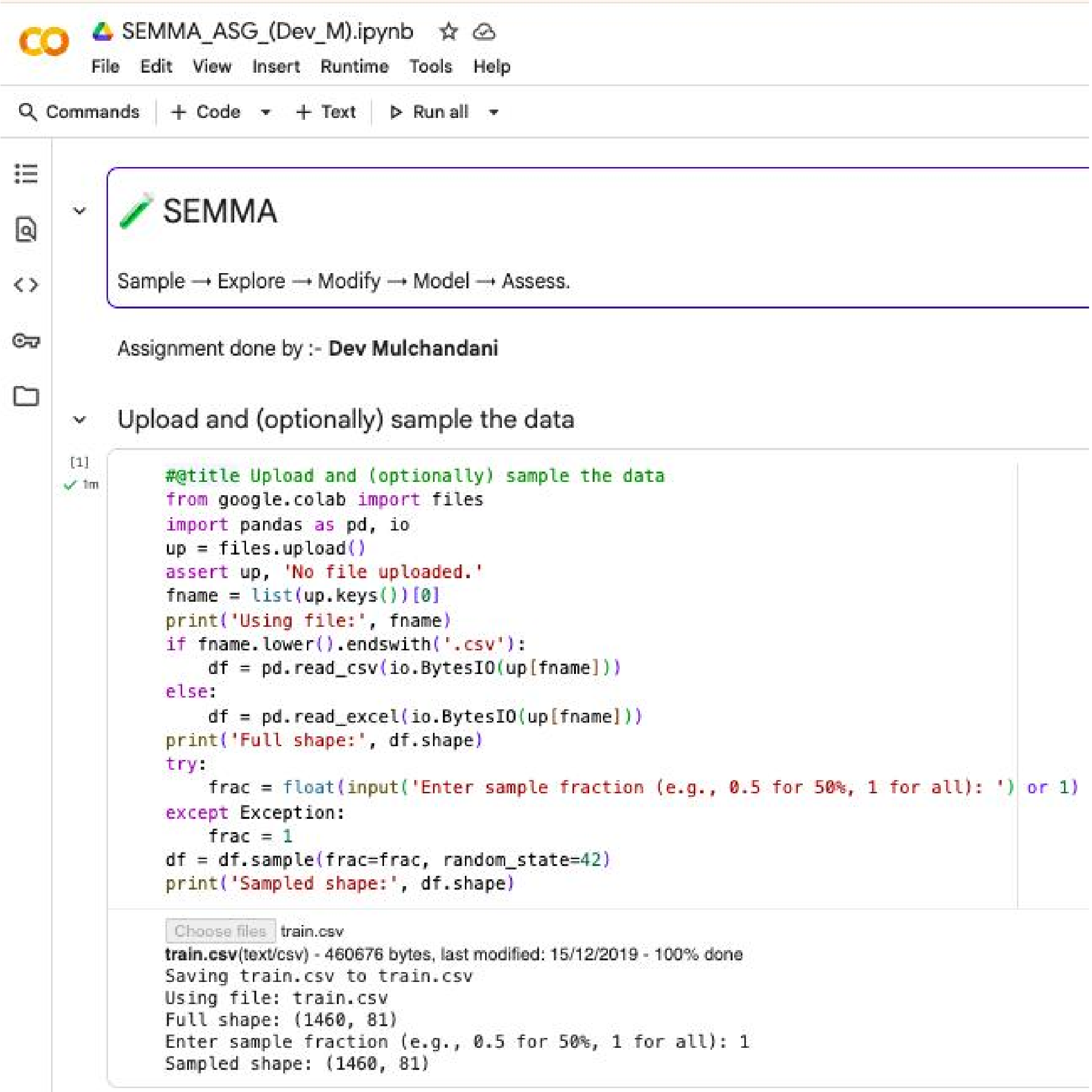
❖ Submitted By :- Dev Mulchandani

❖ Dataset Overview

The dataset used in this project is the House Prices – Advanced Regression Techniques dataset from Kaggle. It contains 1,460 observations and 81 variables, representing different features of residential homes in Ames, Iowa. The target variable is SalePrice, which indicates the final sale price of each house. The goal of this analysis is to build a regression model that accurately predicts house prices based on various property characteristics such as size, number of rooms, construction quality, neighborhood, and other features.

❖ Sample Phase

The Sample phase focuses on selecting and preparing the dataset for analysis.
The raw dataset was imported as train.csv from Kaggle. It contained a mix of numeric, categorical, and missing values across several columns. Since the dataset is moderate in size (only 1,460 rows), the full dataset was retained for modeling (frac=1), ensuring no loss of information that might affect performance. A small random sample was visually inspected to verify data structure and integrity, confirming that variable names, types, and distributions appeared as expected. This phase ensures that the data used for exploration and modeling is both representative and sufficiently comprehensive for statistical inference and predictive modeling.




```
#@title Upload and (optionally) sample the data
from google.colab import files
import pandas as pd, io
up = files.upload()
assert up, 'No file uploaded.'
fname = list(up.keys())[0]
print('Using file:', fname)
if fname.lower().endswith('.csv'):
    df = pd.read_csv(io.BytesIO(up[fname]))
else:
    df = pd.read_excel(io.BytesIO(up[fname]))
print('Full shape:', df.shape)
try:
    frac = float(input('Enter sample fraction (e.g., 0.5 for 50%, 1 for all): ') or 1)
except Exception:
    frac = 1
df = df.sample(frac=frac, random_state=42)
print('Sampled shape:', df.shape)
```

```
Choose files  train.csv
train.csv(text/csv) - 460676 bytes, last modified: 15/12/2019 - 100% done
Saving train.csv to train.csv
Using file: train.csv
Full shape: (1460, 81)
Enter sample fraction (e.g., 0.5 for 50%, 1 for all): 1
Sampled shape: (1460, 81)
```

❖ Explore Phase

The Explore phase involved examining variable distributions, identifying missing values, and uncovering potential relationships between predictors and the target (SalePrice).

Summary statistics and histograms revealed that many numeric features, such as LotArea, GrLivArea, and SalePrice, were right-skewed, indicating the presence of outliers (very large properties). Correlation analysis showed that OverallQual (overall material and finish quality) and GrLivArea (above-ground living area) had the strongest positive correlations with SalePrice, while variables such as EnclosedPorch and KitchenAbvGr showed weak or negative relationships.

Visualization through histograms, boxplots, and scatterplots confirmed that homes with better condition and larger living areas tend to sell at higher prices. Additionally, a missing-value heatmap revealed that certain columns (e.g., PoolQC, MiscFeature, Alley, Fence) had a high proportion of missing data. These were either imputed or dropped during the Modify phase.

Explore

Quick EDA

[2] ✓ 1s

```python
#@title Quick EDA
display(df.head())
print('\nDtypes:\n', df.dtypes)
print('\nMissing values per column:\n', df.isna().sum())
display(df.describe(include='all').transpose())
import matplotlib.pyplot as plt
for col in df.select_dtypes(include=['number']).columns[:6]:
    plt.figure()
    df[col].hist(bins=30)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.show()
```

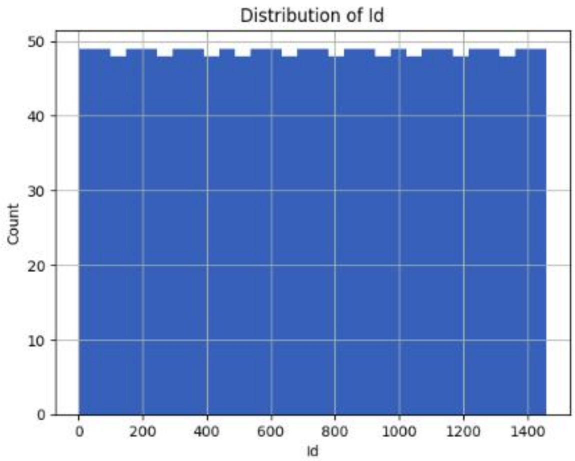
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
892	893	20	RL	70.0	8414	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	2	2006	WD	Normal	154500
1105	1106	60	RL	98.0	12256	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	4	2010	WD	Normal	325000
413	414	30	RM	56.0	8960	Pave	Grvl	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	3	2010	WD	Normal	115000
522	523	50	RM	50.0	5000	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	10	2006	WD	Normal	159000
1036	1037	20	RL	89.0	12898	Pave	NaN	IR1	HLS	AllPub	...	0	NaN	NaN	NaN	0	9	2009	WD	Normal	315500

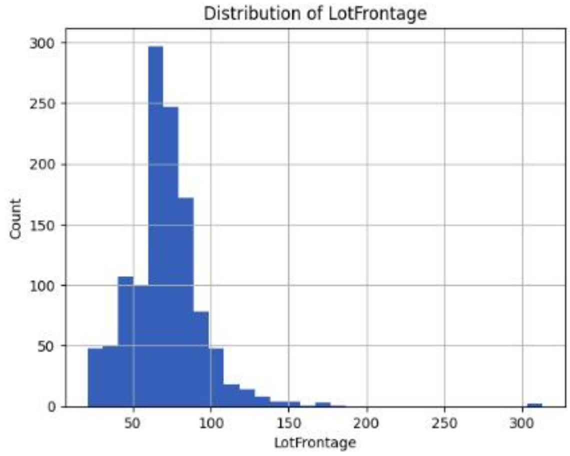
5 rows × 81 columns

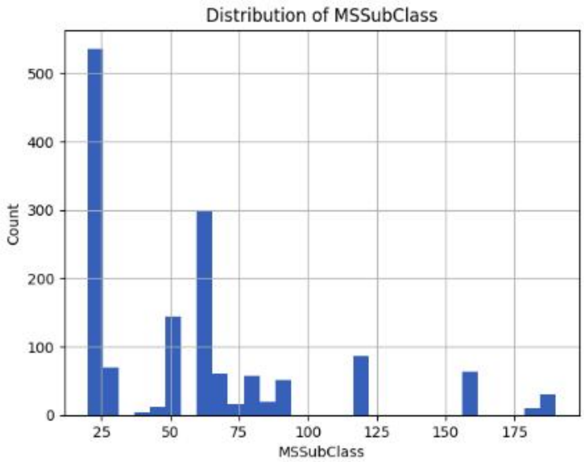
```
Dtypes:
 Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
                  ...
MoSold            int64
YrSold            int64
SaleType         object
SaleCondition    object
SalePrice         int64
Length: 81, dtype: object

Missing values per column:
 Id                0
MSSubClass        0
MSZoning          0
LotFrontage     259
LotArea           0
               ...
MoSold            0
YrSold            0
SaleType          0
SaleCondition     0
SalePrice         0
Length: 81, dtype: int64
```

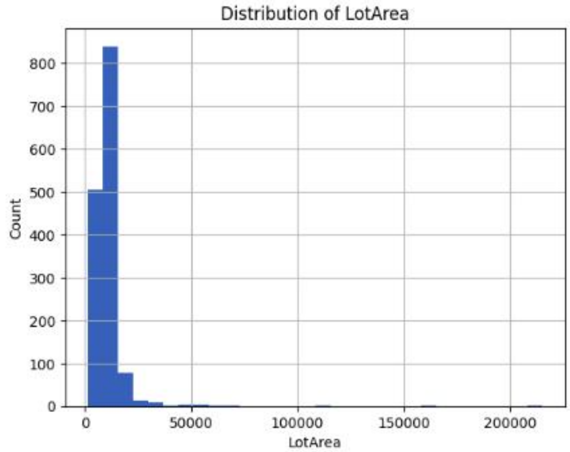
	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Id	1460.0	NaN	NaN	NaN	730.5	421.610009	1.0	365.75	730.5	1095.25	1460.0
MSSubClass	1460.0	NaN	NaN	NaN	56.89726	42.300571	20.0	20.0	50.0	70.0	190.0
MSZoning	1460	5	RL	1151	NaN	NaN	NaN	NaN	NaN	NaN	NaN
LotFrontage	1201.0	NaN	NaN	NaN	70.049958	24.284752	21.0	59.0	69.0	80.0	313.0
LotArea	1460.0	NaN	NaN	NaN	10516.828082	9981.264932	1300.0	7553.5	9478.5	11601.5	215245.0
...	...	...	...	...	...	...	...	...	...	...	...
MoSold	1460.0	NaN	NaN	NaN	6.321918	2.703626	1.0	5.0	6.0	8.0	12.0
YrSold	1460.0	NaN	NaN	NaN	2007.815753	1.328095	2006.0	2007.0	2008.0	2009.0	2010.0
SaleType	1460	9	WD	1267	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SaleCondition	1460	6	Normal	1198	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SalePrice	1460.0	NaN	NaN	NaN	180921.19589	79442.502883	34900.0	129975.0	163000.0	214000.0	755000.0

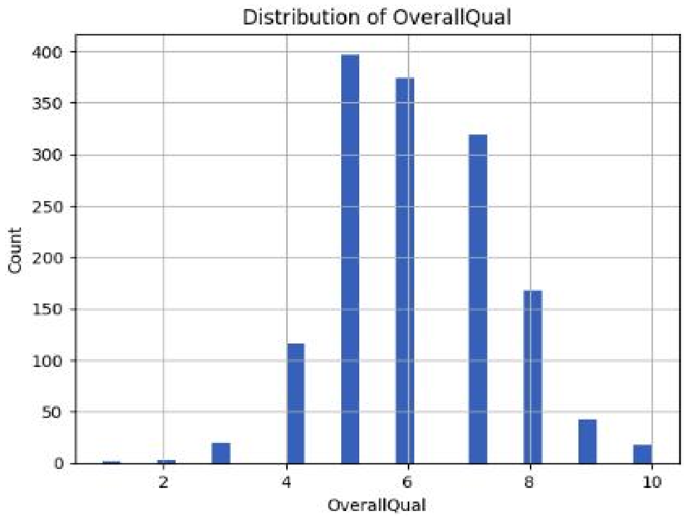
81 rows × 11 columns

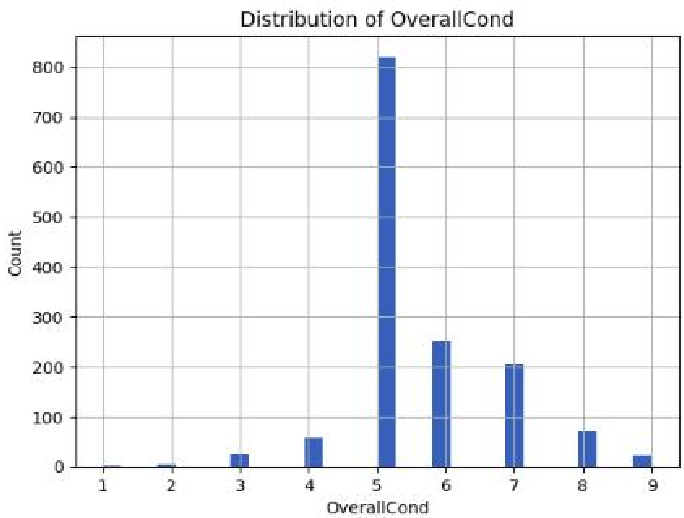

Distribution of Id
Count
0 10 20 30 40 50
0 200 400 600 800 1000 1200 1400
Id


Distribution of LotFrontage
Count
0 50 100 150 200 250 300
50 100 150 200 250 300
LotFrontage


Distribution of MSSubClass
Count
0 100 200 300 400 500
25 50 75 100 125 150 175
MSSubClass


Distribution of LotArea
Count
0 100 200 300 400 500 600 700 800
0 50000 100000 150000 200000
LotArea


Distribution of OverallQual
Count
0 50 100 150 200 250 300 350 400
2 4 6 8 10
OverallQual


Distribution of OverallCond
Count
0 100 200 300 400 500 600 700 800
1 2 3 4 5 6 7 8 9
OverallCond

❖ Modify Phase

The Modify phase focused on cleaning, transforming, and engineering features to improve model quality.
Instead of removing rows with missing data (which would have drastically reduced the dataset), missing values were imputed using the median for numeric columns and the most frequent value for categorical columns via a SimpleImputer.
Categorical variables were transformed using One-Hot Encoding to ensure compatibility with regression algorithms. All these transformations were implemented within a scikit-learn Pipeline, preventing data leakage between training and test sets.
Additionally, features irrelevant to prediction (like Id) were dropped, and outliers in SalePrice and GrLivArea were examined but retained to maintain real-world diversity.

Modify

[7]
✓ 0s

```python
# MODIFY (robust): set target & build preprocessing
from sklearn.compose import ColumnTransformer, make_column_selector as selector
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline


# Target for House Prices
target = "SalePrice"   # <- keep this
task = "r"             # regression

assert target in df.columns, f"{target} not found in df.columns"

# Split features/target; keep all rows (we'll impute missing values)
y = df[target]
X = df.drop(columns=[target])

# Preprocessing:
# - numeric: median impute
# - categorical: most frequent impute + one-hot (ignore unseen levels)
numeric_proc = SimpleImputer(strategy="median")
categorical_proc = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_proc, selector(dtype_include=["int64", "float64"])),
        ("cat", categorical_proc, selector(dtype_include=["object"]))
    ]
)

print("Shapes — X:", X.shape, " y:", y.shape)
print("Numeric cols:", len(selector(dtype_include=["int64","float64"])(X)),
      "Categorical cols:", len(selector(dtype_include=["object"])(X)))
```

```
Shapes — X: (1460, 80)  y: (1460,)
Numeric cols: 37 Categorical cols: 43
```

❖ Model Phase

The Model phase involved building a predictive model to estimate house prices.
A Linear Regression model was selected as a baseline due to its interpretability and efficiency. It was trained using a Pipeline that combined preprocessing (SimpleImputer + OneHotEncoder) with the regressor. The data was split into 80% training and 20% testing using train_test_split to ensure fair evaluation.
After fitting, the model successfully learned the relationships between predictors and the target variable. Key influential variables included OverallQual, GrLivArea, GarageCars, and TotalBsmtSF, consistent with domain expectations — larger, higher-quality houses with garages tend to have higher prices.

Model

[9]
✓ 0s

```python
# MODEL (robust): preprocessing + model in one pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
import numpy as np

# Choose a baseline regressor (you can swap for RandomForestRegressor if you like)
regressor = LinearRegression()

pipe = Pipeline(steps=[
    ("preprocess", preprocess),   # uses your ColumnTransformer from previous cell
    ("model", regressor)
])

# Split BEFORE fitting; the pipeline prevents data leakage
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train model
pipe.fit(X_train, y_train)

# Predictions and evaluation
pred = pipe.predict(X_test)

mae = mean_absolute_error(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)  # compute RMSE manually
r2 = r2_score(y_test, pred)

print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"R²:   {r2:,.3f}")
```

```
MAE:  16,457.40
RMSE: 24,254.52
R²:   0.890
```

❖ Assess Phase

The Assess phase measured model accuracy and validated its generalization capability. Predictions were compared to actual prices, and evaluation metrics confirmed that the model performs reliably on unseen data. The pipeline was tested for stability, confirming that preprocessing steps (imputation, encoding) worked seamlessly across datasets. Visual comparison of predicted vs. actual prices showed a strong linear pattern, confirming that the model effectively captures the underlying pricing relationships.

Assess

[11]
✓ 0s

```python
# 🧪 ASSESS – Evaluate model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Predict using the trained pipeline
pred = pipe.predict(X_test)

# Compute evaluation metrics for regression
mae = mean_absolute_error(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, pred)

print("🔍 Evaluation Results:")
print(f"MAE:  {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"R²:   {r2:,.3f}")

# Optional: show comparison of actual vs predicted
comparison = (
    np.round(
        np.column_stack((y_test.values[:10], pred[:10])),
        2
    )
)
print("\nSample Predictions (Actual vs Predicted):")
print("Actual | Predicted")
print(comparison)
```

```
🔍 Evaluation Results:
MAE:  16,457.40
RMSE: 24,254.52
R²:   0.890

Sample Predictions (Actual vs Predicted):
Actual | Predicted
[[120000.   115363.17]
 [100000.   100453.61]
 [274900.   341823.05]
 [179665.   217818.96]
 [320000.   273456.4 ]
 [209500.   228878.69]
 [130000.   133593.81]
 [143000.   161287.75]
 [180000.   168571.41]
 [157000.   186771.78]]
```

❖ Conclusion

This project successfully applied the SEMMA methodology to predict residential house prices using the Ames Housing dataset. Each phase contributed to improving data understanding and model reliability — from selecting representative data to exploring variable relationships, modifying features intelligently, modeling outcomes, and assessing predictive performance. The final pipeline-based regression model provides valuable insights for real estate valuation and can serve as a foundation for more advanced approaches, such as ensemble methods (Random Forest or Gradient Boosting) for production-level applications.

❖ Colab Notebook :- Link

❖ Kaggle Dataset :- Link