Business Understanding in EDA

house prices dataset

	Id	MSSubClass	MSZoning	LotFrontage
1	1461	20	RH	80
2	1462	20	RL	81
3	1463	60	RL	74
4	1464	60	RL	78
5	1465	120	RL	43
6	1466	60	RL	75

"As an expert data scientist and professor teaching a master's program in data science, please provide a detailed, textbook-quality explanation of how to perform Exploratory Data Analysis (EDA) on the Kaggle house prices dataset using the CRISP-DM methodology. Break down your response into several small, manageable chunks, covering each step of the process. After each chunk, I will ask you to continue.
For each chunk, please:

Specify which CRISP-DM phase you're addressing
Explain the step-by-step process for that phase
Provide code examples using popular Python libraries (e.g., pandas, scikit-learn, matplotlib)
Explain the rationale behind each decision or technique
Summarize what has been covered and what will be addressed next

Include the following topics, distributed across multiple chunks:

Business Understanding
Data Understanding

Initial data exploration
Data quality assessment

Data Preparation

Data cleaning
Feature selection
Data preprocessing

Modeling

Baseline model creation
Advanced modeling techniques
Model comparison and evaluation

Evaluation

Performance metrics
Model interpretation

Deployment

Final recommendations
Potential next steps

Assume limited computational resources, so distribute computationally intensive tasks across chunks. Explain concepts as if teaching to a beginner, using clear and concise language. At the end of each chunk, briefly recap what has been covered and what will be addressed next.
Please begin with the Business Understanding phase of CRISP-DM, explaining its importance and how it applies to the Kaggle house prices dataset."

Memory updated

CRISP-DM Phase 1: Business Understanding

The first phase of the CRISP-DM methodology, **Business Understanding**, is critical because it sets the foundation for the entire data science project. In this phase, the focus is on understanding the business problem and defining the objectives of the analysis. The main questions to be answered include:

- **What is the primary business goal?**
- **What insights or predictions does the business hope to gain from the data?**

Applying Business Understanding to the House Prices Dataset

The Kaggle house prices dataset contains various features that describe houses in Ames, Iowa, such as the number of rooms, square footage, and neighborhood details. Our goal is to predict the sale price of a house based on these features, which is a typical **regression problem** in data science.

Key objectives:

- **Predict house prices**: The primary objective is to create a model that accurately predicts house prices based on the available features.

- **Identify key drivers**: Beyond just making predictions, we want to understand which features most significantly influence house prices, such as the size of the house, location, and quality of materials.
- **Improve decision-making**: Real estate agents, buyers, and sellers can use these insights to make informed decisions about pricing strategies or home investments.

Questions to consider:

- Which features should be emphasized when predicting house prices?
- How will the model handle potential outliers or missing data?
- What level of accuracy is needed for the predictions to be valuable?

Process

1. **Define the success criteria**: A good predictive model should have an accuracy metric (such as R^2 or RMSE) within an acceptable range. Typically, lower errors (RMSE or MAE) indicate better model performance.
2. **Understand the constraints**: Limited computational resources mean we need to be efficient with data preprocessing and model training.
3. **Set project goals**: The ultimate goal is to develop a predictive model and identify the most important features that influence house prices.

In this phase, no Python code is necessary, as we are focusing on understanding the business context.

Recap:

- We defined the **business goal**: predicting house prices using available features.
- We established **success criteria** and **constraints**: improving predictive accuracy while managing computational efficiency.
- Next, we will move to the **Data Understanding** phase, where we will explore and assess the data in detail.

Would you like me to continue to the Data Understanding phase?

yes continue

CRISP-DM Phase 2: Data Understanding

In the **Data Understanding** phase, we focus on getting familiar with the dataset. This involves loading the data, performing initial exploration, identifying important features, and assessing data quality (e.g., missing values, outliers). The aim is to develop an in-depth understanding of the dataset to guide the next steps in the process.

Step 1: Load and Explore the Data

First, we need to load the dataset and inspect its basic structure—its size, the types of features (categorical, numerical), and some general statistics.

Here's a breakdown of the initial data exploration tasks:

- **Loading the dataset**.
- **Inspecting the structure**: View the first few rows and understand the types of features available.
- **Summary statistics**: Generate basic summary statistics to understand feature ranges, mean values, and distributions.

Let's start with the code for loading and exploring the dataset:

```python
# Import necessary libraries
import pandas as pd

# Load the dataset (replace with the correct file path if needed)
file_path = '/mnt/data/house prices dataset.csv'
df = pd.read_csv(file_path)

# Check the size of the dataset
print("Dataset shape:", df.shape)

# Display the first few rows of the dataset
df.head()

# Get summary statistics for numerical columns
df.describe()

# Check the data types of each column
df.info()
```

Rationale:

- `df.shape`: Shows the number of rows and columns, which gives us a sense of the dataset's scale.
- `df.head()`: Provides a glimpse of the first five rows of data, allowing us to quickly inspect how the data is structured.
- `df.describe()`: Summarizes statistics for numerical features, such as mean, standard deviation, and min/max values.
- `df.info()`: Displays the data types of each feature (numerical, categorical) and helps us spot any missing values.

Step 2: Assess Data Quality

Next, we need to assess the quality of the data by checking for:

- **Missing values**: We'll identify columns that have missing values, which could affect the model's performance.
- **Duplicated rows**: Check for duplicate entries that may skew the analysis.
- **Outliers**: Extreme values that could bias the model's predictions.

Here's how we assess data quality:

```python
# Check for missing values in each column
missing_values = df.isnull().sum()
print("Missing values per column:\n", missing_values)

# Check for duplicate rows
duplicate_rows = df.duplicated().sum()
print("Number of duplicate rows:", duplicate_rows)

# Quick visualization of potential outliers using box plots
import matplotlib.pyplot as plt

# Example: boxplot for SalePrice to detect outliers
plt.boxplot(df['SalePrice'].dropna())
plt.title('Boxplot of SalePrice')
plt.show()
```

Rationale:

- `df.isnull().sum()`: Lists the number of missing values in each column. Missing data needs to be handled, either by imputing values or dropping rows/columns.
- `df.duplicated().sum()`: Checks if any duplicate rows exist, which should be removed to avoid bias.
- **Boxplots**: Used to visualize outliers. Outliers in critical columns (like `SalePrice`) can affect model performance, so we need to decide whether to remove or transform them.

Recap:

- We've explored the dataset by loading it and reviewing the basic structure.
- We generated summary statistics and performed data quality checks for missing values, duplicates, and outliers.
- Next, we'll dive deeper into **Data Preparation**, where we'll clean the data and select features for modeling.

Shall I continue with **Data Preparation** next?

yes continue

CRISP-DM Phase 3: Data Preparation

In the **Data Preparation** phase, we clean the data, select relevant features, and preprocess the dataset to ensure it's ready for modeling. This phase is often one of the most time-consuming but essential steps in a data science project because the quality of the input data directly influences the performance of the model.

Step 1: Data Cleaning

In this step, we handle any issues identified during the **Data Understanding** phase, such as:

- **Handling missing values**: We decide how to deal with missing data—either by imputing or dropping rows/columns.
- **Handling outliers**: Depending on the analysis, we may remove or transform extreme values.

Here's how we can handle missing values and outliers for the Kaggle house prices dataset:

```python
# Check again for missing values
missing_values = df.isnull().sum()

# Drop columns where more than 30% of the data is missing (arbitrary threshold)
threshold = len(df) * 0.3
df_cleaned = df.dropna(thresh=threshold, axis=1)

# For columns with fewer missing values, fill missing numerical values with the median
numerical_columns = df_cleaned.select_dtypes(include=['float64', 'int64']).columns
df_cleaned[numerical_columns] =
df_cleaned[numerical_columns].fillna(df_cleaned[numerical_columns].median())

# For categorical columns, fill missing values with the most frequent category (mode)
categorical_columns = df_cleaned.select_dtypes(include=['object']).columns
df_cleaned[categorical_columns] =
df_cleaned[categorical_columns].fillna(df_cleaned[categorical_columns].mode().iloc[0])

# Remove outliers in 'SalePrice' that are more than 3 standard deviations away from the
mean
from scipy import stats
df_cleaned = df_cleaned[(np.abs(stats.zscore(df_cleaned['SalePrice'])) < 3)]
```

Rationale:

- **Dropping columns**: If a column has too many missing values (e.g., more than 30%), it may not be useful for analysis, so we drop it.
- **Imputation**: For numerical features, using the median prevents the influence of outliers. For categorical features, filling missing values with the most frequent category ensures data consistency.
- **Outliers**: We removed any extreme outliers that were more than three standard deviations away from the mean. This ensures that these extreme values won't negatively affect the model.

Step 2: Feature Selection

Selecting the most relevant features improves model performance and reduces complexity. The Kaggle dataset contains many features, but not all of them may be relevant for predicting house prices. Feature selection helps us focus on variables that are most predictive.

There are a few methods for feature selection, such as:

- **Correlation analysis**: Finding the correlation between features and the target (`SalePrice`).
- **Domain knowledge**: Choosing features that make sense based on understanding real estate (e.g., location, size, and quality).

Let's perform correlation analysis on the numerical features:

```python
# Calculate correlation matrix for numerical features
corr_matrix = df_cleaned.corr()

# Select features that have a high correlation with the target ('SalePrice')
high_corr_features = corr_matrix.index[abs(corr_matrix['SalePrice']) > 0.5]

# Visualize the correlations using a heatmap
import seaborn as sns
plt.figure(figsize=(10, 8))
sns.heatmap(df_cleaned[high_corr_features].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix for Highly Correlated Features')
plt.show()

# Drop less correlated features (arbitrary cutoff for demonstration)
df_selected = df_cleaned[high_corr_features]
```

Rationale:

- **Correlation analysis**: Helps us identify features that are highly correlated with `SalePrice`. Features with a high correlation (positive or negative) are more likely to be important for predicting house prices.
- **Heatmap**: A visual way to examine correlations between features and the target, helping us identify patterns.

Step 3: Data Preprocessing

To prepare the data for modeling, we need to handle the following preprocessing steps:

- **Encoding categorical features**: Since machine learning models typically require numerical inputs, we convert categorical variables into numerical values (e.g., via one-hot encoding).
- **Scaling numerical features**: Standardizing or normalizing numerical features ensures that they are on the same scale, which is important for certain models like linear regression or neural networks.

Here's how we can preprocess the dataset:

```python
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Select categorical and numerical columns
categorical_cols = df_selected.select_dtypes(include=['object']).columns
numerical_cols = df_selected.select_dtypes(include=['int64', 'float64']).columns

# Create preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ])

# Apply preprocessing
df_preprocessed = preprocessor.fit_transform(df_selected)
```

Rationale:

- **StandardScaler**: Normalizes numerical data by subtracting the mean and scaling to unit variance.
- **OneHotEncoder**: Converts categorical data into binary vectors (0 or 1), so the model can interpret categorical features.
- **Pipeline**: The use of a pipeline ensures that our transformations are applied consistently and efficiently.

Recap:

- We cleaned the data by handling missing values and outliers.
- We performed feature selection using correlation analysis to identify key predictive features.
- We preprocessed the data by scaling numerical features and encoding categorical variables, preparing the dataset for modeling.

Next, we will move to the **Modeling** phase, where we will create baseline models and explore more advanced modeling techniques.

Shall I continue with the **Modeling** phase?

yes please continue you are doing good job

CRISP-DM Phase 4: Modeling

In the **Modeling** phase, we create machine learning models to predict house prices based on the prepared data. We'll start by building a baseline model and then move on to more advanced techniques to improve prediction accuracy. Finally, we'll compare and evaluate these models.

Step 1: Baseline Model Creation

The first step in any modeling process is to establish a baseline. A baseline model provides a simple starting point, allowing us to measure improvement as we build more sophisticated models. For regression problems like predicting house prices, a **Linear Regression** model is often used as a baseline because it's easy to interpret and relatively simple.

Let's create and evaluate a simple Linear Regression model:

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Split data into features (X) and target (y)
X = df_preprocessed
y = df_selected['SalePrice']
```

```
# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Linear Regression model
lin_reg = LinearRegression()

# Train the model on the training data
lin_reg.fit(X_train, y_train)

# Make predictions on the test data
y_pred = lin_reg.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5

print(f"Baseline Linear Regression RMSE: {rmse:.2f}")
```

Rationale:

- **Train-test split**: Divides the dataset into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate its performance.
- **Linear Regression**: A simple model that assumes a linear relationship between the features and the target. It's quick to compute and gives us a baseline performance metric.
- **Root Mean Squared Error (RMSE)**: A common evaluation metric for regression tasks. It gives an idea of how far the model's predictions are from the actual values, with a lower RMSE indicating better performance.

Step 2: Advanced Modeling Techniques

After establishing a baseline, we can move on to more advanced models that might capture more complex patterns in the data. Two commonly used advanced models for house price prediction are:

- **Random Forest**: An ensemble method that builds multiple decision trees and averages their predictions to improve accuracy.
- **Gradient Boosting (XGBoost)**: Another ensemble method that builds models sequentially, with each new model correcting errors made by the previous ones.

Here's how we can build and evaluate these models:

```
python
```

```
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor

# Initialize the Random Forest and XGBoost models
rf_model = RandomForestRegressor(random_state=42)
xgb_model = XGBRegressor(random_state=42)

# Train the models
rf_model.fit(X_train, y_train)
xgb_model.fit(X_train, y_train)

# Make predictions
rf_pred = rf_model.predict(X_test)
xgb_pred = xgb_model.predict(X_test)

# Evaluate the models using RMSE
rf_rmse = mean_squared_error(y_test, rf_pred) ** 0.5
xgb_rmse = mean_squared_error(y_test, xgb_pred) ** 0.5
```

```
print(f"Random Forest RMSE: {rf_rmse:.2f}")
print(f"XGBoost RMSE: {xgb_rmse:.2f}")
```

Rationale:

- **Random Forest**: This model reduces overfitting by averaging the predictions from multiple decision trees. It can handle non-linear relationships and often outperforms simpler models.
- **XGBoost**: A powerful boosting algorithm that improves performance by focusing on the most difficult-to-predict samples. It's known for achieving high accuracy in many Kaggle competitions.
- **Model comparison**: By comparing the RMSE of the baseline model (Linear Regression) with Random Forest and XGBoost, we can determine which model performs better.

Step 3: Model Comparison and Evaluation

Once we've built multiple models, we need to compare their performance to choose the best one. Here are key points to evaluate:

- **RMSE**: Compare RMSE values for each model. The model with the lowest RMSE is typically the best-performing model.
- **Overfitting**: Check if the model performs well on both the training and testing sets. If a model performs significantly better on the training data than on the testing data, it may be overfitting.
- **Feature Importance**: For models like Random Forest and XGBoost, we can also inspect which features were most important in predicting house prices.

Here's how to check feature importance for Random Forest:

python

```
# Get feature importances from the Random Forest model
importances = rf_model.feature_importances_

# Visualize the top 10 most important features
import numpy as np
import matplotlib.pyplot as plt

# Get feature names (since we used OneHotEncoder, we retrieve the feature names)
feature_names =
np.array(preprocessor.named_transformers_['num'].get_feature_names_out().tolist() +

preprocessor.named_transformers_['cat'].get_feature_names_out().tolist())

# Sort feature importances in descending order
indices = np.argsort(importances)[-10:]

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), feature_names[indices])
plt.xlabel('Feature Importance')
plt.title('Top 10 Important Features in Random Forest')
plt.show()
```

Rationale:

- **Feature importance**: Helps us understand which features contribute the most to the model's predictions, giving valuable insights for business decisions.

Recap:

- We built a **baseline model** using Linear Regression to set a performance benchmark.
- We created **advanced models** (Random Forest and XGBoost) to improve prediction accuracy.
- We evaluated the models using RMSE and inspected **feature importance** to see which features were most influential.

Next, we'll move to the **Evaluation** phase, where we will focus on interpreting the model's performance, assessing its robustness, and discussing possible improvements.

Shall I continue with the **Evaluation** phase?

yes continue

CRISP-DM Phase 5: Evaluation

In the **Evaluation** phase, we assess the model's performance and determine if it meets the business objectives. This involves examining performance metrics, interpreting model results, and identifying any potential issues such as overfitting or bias. The goal is to ensure that the model is reliable and that its predictions are useful for the business problem—in this case, predicting house prices.

Step 1: Performance Metrics

We've already used **RMSE (Root Mean Squared Error)** to evaluate model performance. RMSE is a good measure of how well the model's predictions align with the actual house prices. However, we may also want to look at other metrics for a more comprehensive evaluation.

Let's explore two additional performance metrics:

- **Mean Absolute Error (MAE)**: MAE measures the average absolute difference between predicted and actual values, providing a more interpretable error metric.
- **R^2 (R-squared)**: R^2 indicates how much of the variance in the target variable (`SalePrice`) is explained by the model.

Here's the code to calculate and interpret these metrics:

```python
from sklearn.metrics import mean_absolute_error, r2_score

# Calculate MAE for each model
mae_lin = mean_absolute_error(y_test, y_pred)
mae_rf = mean_absolute_error(y_test, rf_pred)
mae_xgb = mean_absolute_error(y_test, xgb_pred)
```

```
# Calculate R² for each model
r2_lin = r2_score(y_test, y_pred)
r2_rf = r2_score(y_test, rf_pred)
r2_xgb = r2_score(y_test, xgb_pred)

print(f"Linear Regression - MAE: {mae_lin:.2f}, R²: {r2_lin:.2f}")
print(f"Random Forest - MAE: {mae_rf:.2f}, R²: {r2_rf:.2f}")
print(f"XGBoost - MAE: {mae_xgb:.2f}, R²: {r2_xgb:.2f}")
```

Rationale:

- **MAE**: Measures the average error without penalizing large errors as harshly as RMSE does. A lower MAE means that, on average, the model's predictions are closer to the actual values.
- **R^2**: Explains how well the model captures variance in the data. An R^2 value closer to 1 indicates that the model explains a large portion of the variance in house prices.

Step 2: Model Interpretation

Once we have evaluated performance metrics, we need to interpret the results:

- **Performance comparison**: Compare RMSE, MAE, and R^2 for each model. Generally, lower RMSE and MAE, and higher R^2 values, indicate a better-performing model.
- **Feature importance**: For models like Random Forest and XGBoost, feature importance helps explain which features are driving the predictions. This can offer valuable insights into which factors contribute most to house prices.

If certain features dominate in importance (e.g., `GrLivArea`, `OverallQual`), this could be a key takeaway for real estate professionals looking to prioritize factors in house pricing decisions.

Step 3: Overfitting Check

One potential issue with advanced models is overfitting, where the model performs exceptionally well on the training data but poorly on the test data. We can check for overfitting by comparing the model's performance on the training and testing sets.

python

```
# Check performance on the training data (Random Forest example)
y_train_pred_rf = rf_model.predict(X_train)
train_rmse_rf = mean_squared_error(y_train, y_train_pred_rf) ** 0.5
print(f"Random Forest RMSE (Train): {train_rmse_rf:.2f}")

# Compare this to the test RMSE we already calculated
print(f"Random Forest RMSE (Test): {rf_rmse:.2f}")
```

Rationale:

- **Overfitting detection**: If the model performs much better on the training data than on the test data, this indicates overfitting. We may need to simplify the model, regularize it, or collect more data to prevent overfitting.

Recap:

- We evaluated the models using **additional metrics**: MAE and R^2, alongside RMSE, to assess their overall performance.
- We interpreted the models by comparing performance and looking at **feature importance** to understand which factors influence predictions the most.
- We checked for **overfitting**, ensuring that the model generalizes well to unseen data.

Next, we'll move to the final phase, **Deployment**, where we will summarize the results, provide recommendations, and discuss potential next steps.

Shall I continue with the **Deployment** phase?

yes continue

CRISP-DM Phase 6: Deployment

In the **Deployment** phase, we summarize the results and provide actionable insights based on the analysis. This is the stage where we deliver the model to stakeholders and offer recommendations on how it can be used in real-world decision-making. The deployment phase may also involve outlining future steps to improve the model or extend its capabilities.

Step 1: Final Recommendations

Based on the modeling and evaluation phases, here are the key recommendations:

- **Best model**: The model that performed the best in terms of RMSE, MAE, and R^2 is likely the **XGBoost** model (or whichever model had the best performance in your specific case). This model captured the complexity of the data better than the baseline linear regression.
- **Key features**: The most important features influencing house prices (e.g., `GrLivArea`, `OverallQual`, and `YearBuilt`, based on feature importance analysis) should be considered by real estate agents and home buyers when evaluating or setting house prices.
- **Deployment options**: The model can be deployed in a real estate application where users input details about a house, and the model predicts the sale price. For example:
  - **Real estate agents** could use this tool to suggest competitive pricing for houses.
  - **Homeowners** might use it to estimate the value of their property.

Step 2: Possible Next Steps

Although the current model performs well, there are additional steps we can take to further refine the model and improve its predictive accuracy:

1. **Hyperparameter tuning**: Use techniques like Grid Search or Randomized Search to fine-tune the hyperparameters of the Random Forest and XGBoost models. This can help squeeze more performance out of these models.

```python
from sklearn.model_selection import GridSearchCV

# Example: Grid search for XGBoost hyperparameters
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1]
}
grid_search = GridSearchCV(XGBRegressor(random_state=42), param_grid,
scoring='neg_mean_squared_error', cv=5)
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
```

2. **Cross-validation**: To get a better idea of how the model generalizes, you can use k-fold cross-validation, which splits the data into k subsets and trains the model k times, each time using a different subset as the test set.

```python
from sklearn.model_selection import cross_val_score
xgb_cv_scores = cross_val_score(xgb_model, X, y, cv=5,
scoring='neg_mean_squared_error')
print(f"XGBoost Cross-Validation RMSE: {(-xgb_cv_scores.mean())**0.5:.2f}")
```

3. **Additional feature engineering**: We could potentially create new features from the existing data that could improve the model's predictions. For example:
    - **Interaction features**: Combining features like `GrLivArea` and `OverallQual` into a new feature.
    - **Log transformation**: Applying log transformation to skewed features (e.g., `SalePrice`) can help improve model performance.

4. **Ensemble methods**: You could explore stacking models, where predictions from multiple models (e.g., Linear Regression, Random Forest, XGBoost) are combined to make the final prediction. This often improves predictive accuracy by leveraging the strengths of different models.

Step 3: Model Deployment Options

Once the model is finalized, it can be deployed in several ways:

- **Web Application**: The model can be integrated into a web-based interface using a framework like Flask or Streamlit. This would allow users to input house features and get instant price predictions.
    - Example: Using **Streamlit** to deploy the model:

```python
import streamlit as st
```

```
# Input fields for house features
gr_liv_area = st.number_input('GrLivArea')
overall_qual = st.number_input('OverallQual')
year_built = st.number_input('YearBuilt')

# Make prediction using the XGBoost model
input_data = [[gr_liv_area, overall_qual, year_built]]  # Example inputs
prediction = xgb_model.predict(input_data)
st.write(f"Predicted SalePrice: ${prediction[0]:.2f}")
```

- **API deployment**: The model can be exposed via an API, allowing other applications or services to query it for predictions. Tools like Flask or FastAPI can be used to build this API.

- **Batch predictions**: The model can also be deployed to generate predictions in batch mode, where large datasets of house features are input, and the model outputs predictions for each house. This is useful for companies that want to analyze multiple properties at once.

Step 4: Documentation and Handoff

It's essential to document the entire process, including:

- **The data preprocessing steps**: So that future users understand how the raw data was transformed before being fed into the model.
- **The modeling process**: Including which models were tested, their performance, and why the chosen model was selected.
- **The evaluation metrics**: To provide transparency about how the model was evaluated and its expected performance in real-world scenarios.
- **Model assumptions**: Clarify any assumptions made during feature selection, data cleaning, or model building.

Recap:

- We reviewed the final recommendations based on the best-performing model and outlined key features influencing house prices.
- We explored potential next steps, such as hyperparameter tuning, cross-validation, feature engineering, and ensemble modeling.
- We discussed various options for deploying the model, including web applications, APIs, and batch predictions.
- Lastly, we emphasized the importance of documenting the entire process for future reference and handoff.

This concludes the entire CRISP-DM process for the Kaggle house prices prediction project. Let me know if you'd like to explore any specific part in more detail!