CRISP-DM Data Science Project

this assignment is to do principled step by step phase by phase data science with CRISP-DM critic your work thru multiple revisions for each section. You need to demonstrate a project which uses all stages of the the methodology.
a) Pick a different data set for each methodology from Kaggle or https://paperswithcode.com/datasets
b) Follow in depth all the steps of entire crisp-dm process
You are world renowned speaker and authority in crisp-dm and wrote many award winning books in this topic. Give in detail response for the requirement and also after each stage ask if you need to progress to the next.

Memory updated

To address this CRISP-DM project assignment in detail, we will go through each phase of the methodology with examples and detailed actions. After completing each phase, I will ask if you'd like to proceed to the next phase. Here's a structured plan:

Phase 1: Business Understanding

Objective: Define the project objectives and requirements from a business perspective.

- **Task**: Understand the specific problem you are trying to solve with the data.
- **Questions to ask**:
  - What is the main business objective? (e.g., increase sales, reduce fraud, predict outcomes)
  - What would success look like in terms of business outcomes?
  - How will stakeholders use the results?
- **Example (Retail Sales Prediction)**:
  - **Objective**: Predict future sales for a retail store to optimize stock levels.
  - **Success**: Achieve at least a 10% improvement in forecasting accuracy over current methods.
  - **Stakeholder use**: Store managers will use the forecasted data to plan stock orders more effectively.

Action:

1. **Select a dataset**: You will need to pick a dataset that aligns with a business problem (from Kaggle or Papers with Code).
2. **Define the problem**: Clearly write out the business problem and what the ideal solution looks like.

Would you like to start by selecting a dataset and defining the business problem?

Phase 2: Data Understanding

Objective: Collect initial data and become familiar with it.

- **Task**: Explore the dataset to understand its structure and contents.
  - Perform descriptive analysis (mean, median, standard deviation).
  - Identify any potential quality issues in the data (missing values, incorrect data types).
  - Create simple visualizations to understand relationships in the data.
- **Example (Retail Sales)**:
  - Explore features such as "sales amount," "date," "store location," and "product category."
  - Visualize sales trends over time to understand seasonality.

Action:

1. **Perform data exploration**: Use statistical summaries and visualizations to get an overview of the data.
2. **Identify patterns or issues**: Note any interesting trends or problems that might impact further analysis.

Once you finish understanding the data, should we proceed to the next phase?

Phase 3: Data Preparation

Objective: Prepare the data for modeling.

- **Task**: Clean the data, handle missing values, outliers, and transform variables as needed.
  - Normalize or scale features.
  - Create new variables if necessary (e.g., time-based features from a date column).
  - Split the data into training and testing sets.
- **Example (Retail Sales)**:
  - Handle missing sales data, normalize the "sales amount" to adjust for inflation, and create time features (day, month, year).
  - Split the data into a training set (80%) and a test set (20%).

Action:

1. **Clean the data**: Perform necessary data transformations to ensure quality.
2. **Prepare for modeling**: Ensure that the dataset is ready for machine learning.

Shall we move to the next step after data is prepared?

Phase 4: Modeling

Objective: Build models to solve the business problem.

- **Task**: Choose appropriate machine learning algorithms and train them.
  - For example, regression models for sales prediction, classification models for customer segmentation.

- Train multiple models (e.g., linear regression, random forest) and compare performance.
    - Tune hyperparameters to improve model performance.

- **Example (Retail Sales)**:
    - Use linear regression, decision trees, and random forests to predict sales.
    - Perform cross-validation and hyperparameter tuning to select the best-performing model.

Action:

1. **Build models**: Train and evaluate models using cross-validation.
2. **Compare models**: Select the best model based on metrics such as accuracy or RMSE.

Should we proceed to evaluating the models after training them?

Phase 5: Evaluation

Objective: Assess the performance of the models in solving the business problem.

- **Task**: Evaluate model performance using appropriate metrics.
    - For regression tasks, evaluate with metrics such as RMSE or MAE.
    - For classification tasks, use accuracy, precision, recall, and F1 score.
    - Ensure the model meets the original business objectives.

- **Example (Retail Sales)**:
    - Evaluate model performance using RMSE (Root Mean Squared Error) and compare it to the baseline performance.

Action:

1. **Evaluate models**: Measure performance and check against the original business objective.
2. **Decide if the model is ready for deployment** or requires further refinement.

Shall we move forward to model deployment?

Phase 6: Deployment

Objective: Deploy the model for business use.

- **Task**: Prepare the model for deployment.
    - Set up an API or application to integrate the model with business systems.
    - Monitor the model's performance over time and re-train if necessary.

- **Example (Retail Sales)**:
    - Set up a forecasting dashboard where store managers can input the current date and see future sales projections.
    - Monitor the model to ensure its predictions remain accurate.

Action:

1. **Deploy the model**: Integrate it into a production environment for business use.
2. **Monitor performance**: Ensure the model performs as expected over time.

This step-by-step breakdown allows for a comprehensive approach to CRISP-DM. At each phase, I will critique the work and guide revisions as needed.

Would you like to start by selecting a dataset and defining the business problem for the **Business Understanding** phase?

Can you give options to select dataset

Here are a few dataset options, each aligned with different business problems and phases of CRISP-DM. You can choose one from each group, or we can pick one and adapt it through the stages. These datasets are from both Kaggle and Papers with Code.

Phase 1: Business Understanding (Choose one dataset related to a business problem)

1. Retail Sales Prediction Dataset (Kaggle)
   - **Business Problem**: Forecast sales demand for retail stores to optimize stock levels and reduce overstocking or understocking.
   - **Goal**: Improve sales forecast accuracy.

2. Customer Churn Dataset (Kaggle)
   - **Business Problem**: Predict customer churn for a telecom company to retain customers and improve retention strategies.
   - **Goal**: Identify customers likely to leave and implement retention plans.

3. Loan Prediction Dataset (Kaggle)
   - **Business Problem**: Predict loan approval based on customer data to assist financial institutions in loan decision-making.
   - **Goal**: Streamline loan approval processes with accurate prediction models.

4. Heart Disease Prediction (Papers with Code)
   - **Business Problem**: Predict heart disease based on patient data to inform healthcare providers and improve preventive strategies.
   - **Goal**: Improve accuracy of heart disease diagnosis.

Phase 2: Data Understanding (Choose one dataset to explore and understand)

1. Global Superstore Sales Data (Kaggle)

- **Business Problem**: Explore sales data to uncover insights into product performance, customer segments, and sales trends.
- **Goal**: Optimize inventory and marketing strategies based on sales data analysis.

2. Airbnb Listing Data (Kaggle)
   - **Business Problem**: Explore New York City Airbnb listing data to understand pricing dynamics, customer behavior, and demand.
   - **Goal**: Identify factors that influence Airbnb listing prices and booking rates.

Phase 3: Data Preparation (Choose one dataset to clean and transform)

1. Titanic Survival Prediction Dataset (Kaggle)
   - **Business Problem**: Prepare Titanic passenger data for predicting survival based on demographic information.
   - **Goal**: Clean the dataset and create relevant features for accurate survival predictions.

2. BigMart Sales Dataset (Kaggle)
   - **Business Problem**: Clean and transform data for sales predictions at BigMart stores.
   - **Goal**: Prepare data for a robust sales prediction model by handling missing values, creating new features, and normalizing data.

Phase 4: Modeling (Choose one dataset for building machine learning models)

1. House Prices Prediction Dataset (Kaggle)
   - **Business Problem**: Predict house prices based on various features such as square footage, location, and amenities.
   - **Goal**: Build and train regression models to predict house prices accurately.

2. Sentiment Analysis on Movie Reviews (Kaggle)
   - **Business Problem**: Predict the sentiment (positive or negative) of movie reviews based on text data.
   - **Goal**: Build classification models to accurately predict movie review sentiment.

Phase 5: Evaluation (Choose one dataset to evaluate models)

1. Credit Card Fraud Detection Dataset (Kaggle)
   - **Business Problem**: Evaluate models for predicting fraudulent credit card transactions.
   - **Goal**: Evaluate the precision, recall, and accuracy of fraud detection models.

2. Forest Fire Prediction Dataset (Kaggle)
   - **Business Problem**: Evaluate models for predicting the occurrence and severity of forest fires.
   - **Goal**: Assess the accuracy of the model in predicting fire intensity and occurrence.

Phase 6: Deployment (Choose one dataset for deployment demonstration)

1. Sales Forecasting Dataset (Kaggle)
    - **Business Problem**: Deploy a sales forecasting model for a retail chain.
    - **Goal**: Create a system to deploy the model and provide ongoing sales forecasts for retail stores.
2. Energy Efficiency Dataset (UCI Machine Learning Repository)
    - **Business Problem**: Deploy a model for predicting the energy efficiency of buildings.
    - **Goal**: Build a system to deploy the model and recommend improvements for energy efficiency based on predictions.

You can select one dataset per phase from the above options, or we can start with one dataset and adapt it for multiple phases.

Would you like to choose one dataset now? Let me know your preference!

You can go ahead with Airbnb Listing Data (Kaggle). Also after each stage wait till i respond to the question before you proceed

Memory updated

Great! We'll start with the **Business Understanding** phase using the **Airbnb Listing Data**.

Phase 1: Business Understanding

Objective: Identify and define the business problem Airbnb stakeholders want to solve using the data.

Scenario: You are working with Airbnb, and they want to understand the key factors that influence the pricing of their listings in New York City. They need insights into which variables—such as location, number of bedrooms, amenities, etc.—significantly affect pricing. These insights will help property owners and managers optimize pricing strategies.

Steps:

1. **Business Objective**:
    - The business wants to know what drives listing prices and how they can optimize these prices to improve occupancy rates while maximizing revenue.
2. **Business Success Criteria**:
    - Success is determined by the ability to identify the top features that influence listing prices. These insights can be used to create a dynamic pricing tool for Airbnb hosts.
3. **Questions to Address**:

- What factors are most important in determining the price of an Airbnb listing in New York City?
- How can Airbnb use this information to recommend optimal pricing strategies to property owners?

Would you like to proceed with this business understanding, or would you like to refine it before we move to the next phase?

yes continue

Phase 2: Data Understanding

Now that we have a clear business understanding, we move on to **Data Understanding**. This phase involves getting familiar with the Airbnb dataset and exploring its key features.

Objectives:

- Understand the structure, content, and quality of the Airbnb listing data.
- Identify any initial data quality issues (missing values, outliers, incorrect types).
- Begin exploring relationships in the data using visualizations.

Steps:

1. **Data Collection**:
    - We'll load the Airbnb listing data from Kaggle into our environment and inspect the features available in the dataset.
2. **Data Overview**:
    - Examine key features like `price`, `location`, `number of reviews`, `availability`, `number of bedrooms`, `accommodates`, and more.
3. **Descriptive Statistics**:
    - Perform basic statistical analysis (mean, median, standard deviation) on numerical features such as `price`, `number of reviews`, and `minimum nights`.
4. **Initial Data Exploration**:
    - Create basic visualizations (e.g., histograms for price distribution, scatter plots showing the relationship between price and number of bedrooms).
    - Investigate correlations between price and other numerical features.
5. **Data Quality Checks**:
    - Identify any missing or inconsistent data (e.g., missing prices, unrealistic availability values).

Would you like to proceed with loading the data and performing these initial explorations, or do you need to refine this phase?

I would want to try each stage of crisp dm in google colab. provide code to run to understand what is happening in each stage. start with fetching dataset and then we can proceed to the next phase

Let's start by fetching the Airbnb dataset and loading it into a Google Colab environment. I'll guide you through each stage of the process. You can run the following code in Colab to fetch the data, load it, and explore the dataset.

Step 1: Setting up the Environment and Loading the Dataset

You will first need to upload the dataset to your Google Colab environment. You can either manually upload the CSV file from the Kaggle Airbnb Listing Data page or use the Kaggle API to fetch it directly.

Option 1: Upload the Dataset Manually

1. Download the **Airbnb Listing Data** from Kaggle.
2. In your Google Colab notebook, use the code below to upload the file.

```python
# Import necessary libraries
import pandas as pd
from google.colab import files

# Upload the dataset
uploaded = files.upload()

# Load the dataset into a DataFrame
# Replace 'AB_NYC_2019.csv' with the exact filename
df = pd.read_csv('AB_NYC_2019.csv')

# Preview the first 5 rows of the dataset
df.head()
```

Option 2: Fetch Dataset Using Kaggle API

If you have the Kaggle API setup, you can run the following steps to fetch the dataset directly in Colab:

1. First, upload your Kaggle API credentials. In Colab, you can do this by running:

```python
from google.colab import files
files.upload()  # This will prompt you to upload your kaggle.json file
```

2. Then, install Kaggle and download the dataset:

```bash
# Install Kaggle
!pip install -q kaggle

# Create a folder for Kaggle
!mkdir -p ~/.kaggle

# Copy the Kaggle API credentials to the folder
!cp kaggle.json ~/.kaggle/

# Set the permissions for the Kaggle API key
!chmod 600 ~/.kaggle/kaggle.json

# Download the Airbnb dataset
!kaggle datasets download -d dgomonov/new-york-city-airbnb-open-data

# Unzip the downloaded dataset
!unzip new-york-city-airbnb-open-data.zip
```

3. Finally, load the dataset in Colab:

```python
# Load the dataset into a DataFrame
df = pd.read_csv('AB_NYC_2019.csv')

# Preview the first 5 rows of the dataset
df.head()
```

Step 2: Basic Exploration of the Dataset

After loading the dataset, we can start with basic data exploration to understand its structure:

```python
# Check the shape of the dataset (rows, columns)
df.shape

# Check the data types of each column
df.dtypes

# Check for missing values in each column
df.isnull().sum()

# Summary statistics of the numerical columns
df.describe()

# Sample 5 random rows from the dataset to see the data
df.sample(5)
```

This will help you understand the structure of the dataset, the types of features available, and if there are any missing values.

Once you've loaded and explored the dataset, let me know, and we can move to the next phase (Data Understanding) with more advanced exploration techniques and visualizations. Would you like to proceed with this setup?

yes continue, along with next phase description provide code to test it in colab

Phase 2: Data Understanding

Now that you've loaded the dataset, the next step is to explore it further to better understand the contents. This phase involves gaining deeper insights through statistical summaries, visualizations, and correlation analysis. We will check for missing values, outliers, and relationships between features.

Here's how you can approach this phase, along with code you can run in Colab.

Objective: Gain insights from the dataset to identify patterns, trends, and potential data issues.

Step 1: Handling Missing Data

First, let's identify and visualize the missing values. We'll start by looking at which columns have missing data and then decide how to handle them (e.g., remove rows, fill values).

```python
# Count missing values for each column
missing_values = df.isnull().sum().sort_values(ascending=False)
print(missing_values)

# Visualizing missing data
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Data Heatmap')
plt.show()
```

Step 2: Data Distribution

Let's examine the distribution of key numerical features, such as **`price`**, **`number_of_reviews`**, and **`minimum_nights`**.

```python
# Distribution of prices
plt.figure(figsize=(8,5))
```

```
sns.histplot(df['price'], bins=50, kde=True)
plt.title('Distribution of Airbnb Listing Prices')
plt.show()

# Distribution of number of reviews
plt.figure(figsize=(8,5))
sns.histplot(df['number_of_reviews'], bins=50, kde=True)
plt.title('Distribution of Number of Reviews')
plt.show()

# Distribution of minimum nights
plt.figure(figsize=(8,5))
sns.histplot(df['minimum_nights'], bins=50, kde=True)
plt.title('Distribution of Minimum Nights')
plt.show()
```

Step 3: Exploring Relationships Between Variables

Next, we'll explore the relationships between key features like **`price`**, **`number of bedrooms`**, and **`availability_365`**.

```python
# Scatter plot to visualize relationship between price and number of bedrooms
plt.figure(figsize=(8,5))
sns.scatterplot(data=df, x='bedrooms', y='price')
plt.title('Price vs Number of Bedrooms')
plt.show()

# Scatter plot to visualize relationship between price and availability_365
plt.figure(figsize=(8,5))
sns.scatterplot(data=df, x='availability_365', y='price')
plt.title('Price vs Availability (365 days)')
plt.show()
```

Step 4: Correlation Analysis

We will compute the correlation matrix for the numerical columns to understand the relationships between different variables.

```python
# Correlation matrix of numerical features
correlation_matrix = df.corr()

# Plotting the heatmap of correlations
plt.figure(figsize=(10,6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

Step 5: Understanding Categorical Features

Let's explore some of the categorical features, such as **`neighbourhood_group`** and **`room_type`**, to see how they relate to the price.

```python
# Average price per neighbourhood group
plt.figure(figsize=(8,5))
sns.barplot(data=df, x='neighbourhood_group', y='price')
plt.title('Average Price by Neighbourhood Group')
plt.show()

# Average price by room type
plt.figure(figsize=(8,5))
sns.barplot(data=df, x='room_type', y='price')
plt.title('Average Price by Room Type')
plt.show()
```

Key Insights You Might Look For:

- **Missing Data**: How significant is the missing data? Do you need to remove rows or fill in missing values?
- **Price Distribution**: Are there outliers in the price data (e.g., extremely high or low prices)?
- **Feature Relationships**: Which features show the strongest correlation with price?
- **Room Type & Location**: Do certain neighborhoods or room types drive higher prices?

Once you've explored and understood the data, let me know if you are ready to move on to the **Data Preparation** phase, where we'll clean and transform the dataset for modeling. Should we proceed?

yes continue

Phase 3: Data Preparation

Now that you've explored the dataset and gained a deeper understanding of the variables and relationships, the next phase is **Data Preparation**. In this stage, we will clean and transform the dataset to make it ready for modeling. The key tasks involve handling missing data, addressing outliers, encoding categorical variables, and splitting the data into training and testing sets.

Objective: Prepare the data for modeling by handling data quality issues, transforming features, and creating an appropriate dataset for machine learning models.

Step 1: Handling Missing Data

If we found missing data in key features during the exploration phase, we need to decide how to handle it. You can either remove rows with missing data or fill them with appropriate values (mean, median, etc.).

Here's an example of filling missing values:

```python
# Handling missing values
# Fill missing 'reviews_per_month' with 0 (assuming listings with no reviews have 0)
df['reviews_per_month'].fillna(0, inplace=True)

# Drop rows with missing values for important features (e.g., 'host_name', 'name')
df_cleaned = df.dropna(subset=['host_name', 'name'])

# Check if missing values are handled
df_cleaned.isnull().sum()
```

Step 2: Handling Outliers

Outliers can distort your model, especially in the price column. We'll remove listings with extreme prices to ensure the model isn't biased.

```python
# Remove listings with prices that are extremely high (above 99th percentile)
upper_limit = df_cleaned['price'].quantile(0.99)
df_cleaned = df_cleaned[df_cleaned['price'] < upper_limit]

# Check the new distribution of price after removing outliers
plt.figure(figsize=(8,5))
sns.histplot(df_cleaned['price'], bins=50, kde=True)
plt.title('Price Distribution after Removing Outliers')
plt.show()
```

Step 3: Encoding Categorical Variables

Machine learning models often require categorical variables to be encoded as numerical data. We'll use **one-hot encoding** to convert categorical variables like `neighbourhood_group` and `room_type` into numerical values.

```python
# One-hot encoding for categorical variables
df_encoded = pd.get_dummies(df_cleaned, columns=['neighbourhood_group', 'room_type'],
drop_first=True)

# Check the dataset after encoding
df_encoded.head()
```

Step 4: Feature Scaling

Some algorithms (e.g., gradient boosting, neural networks) benefit from scaled data. We'll scale the numerical features like `price`, `availability_365`, and `number_of_reviews` using **Min-Max scaling**.

```python
from sklearn.preprocessing import MinMaxScaler

# Selecting numerical features for scaling
```

```
num_features = ['price', 'availability_365', 'number_of_reviews', 'minimum_nights']

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the numerical features
df_encoded[num_features] = scaler.fit_transform(df_encoded[num_features])

# Check the scaled features
df_encoded[num_features].head()
```

Step 5: Splitting Data into Training and Testing Sets

Finally, we'll split the dataset into **training** and **testing** sets to ensure the model is evaluated on unseen data. We typically split the data 80% for training and 20% for testing.

```python
from sklearn.model_selection import train_test_split

# Defining the features (X) and target variable (y)
X = df_encoded.drop(['price'], axis=1)
y = df_encoded['price']

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

# Check the shape of the training and testing sets
X_train.shape, X_test.shape
```

Key Actions:

1. **Handle missing data**: We filled or removed missing values based on importance.
2. **Remove outliers**: Listings with extreme prices were removed.
3. **Encode categorical variables**: Categorical variables were transformed using one-hot encoding.
4. **Scale features**: Numerical features were scaled using MinMaxScaler.
5. **Split the data**: The data was split into training and testing sets to prepare for modeling.

The data is now cleaned, transformed, and ready for modeling! Once you've run these steps and everything looks good, let me know if you're ready to move on to the **Modeling** phase, where we'll start building machine learning models. Should we proceed?

Yes proceed

Phase 4: Modeling

Now that the data is prepared, it's time to build and train machine learning models. In this phase, we will experiment with different algorithms to predict Airbnb listing prices. We'll evaluate their performance and select the best model.

Objective: Build and train machine learning models to predict Airbnb listing prices and evaluate their performance.

Step 1: Building a Baseline Model

Let's start by building a simple linear regression model. This baseline model will help us understand how well a basic model performs before trying more complex algorithms.

```python
# Import necessary libraries
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the model
linear_model = LinearRegression()

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

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

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Linear Regression MSE: {mse}")
print(f"Linear Regression R2 Score: {r2}")
```

Step 2: Trying a More Complex Model (Random Forest)

Now that we have a baseline, let's try a more complex model like Random Forest, which tends to perform well on structured data like this.

```python
# Import necessary libraries
from sklearn.ensemble import RandomForestRegressor

# Initialize the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

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

# Make predictions on the test data
y_pred_rf = rf_model.predict(X_test)

# Evaluate the Random Forest model's performance
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest MSE: {mse_rf}")
print(f"Random Forest R2 Score: {r2_rf}")
```

Step 3: Evaluating the Models

Once you've trained multiple models, you should compare their performance to decide which one to use. For example, you can compare the Mean Squared Error (MSE) and R-squared scores between the linear regression and Random Forest models. The model with a lower MSE and a higher R-squared score would typically be considered better.

Step 4: Hyperparameter Tuning (Optional)

To improve model performance, you can tune the hyperparameters. For Random Forest, this might involve adjusting the number of trees, maximum depth, etc.

Here's an example of how you might tune the hyperparameters of the Random Forest model using GridSearchCV:

```python
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3, n_jobs=-1,
verbose=2)

# Fit the model with grid search
grid_search.fit(X_train, y_train)

# Print the best parameters found by GridSearchCV
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Use the best estimator to make predictions
best_rf_model = grid_search.best_estimator_
y_pred_best_rf = best_rf_model.predict(X_test)

# Evaluate the tuned Random Forest model's performance
mse_best_rf = mean_squared_error(y_test, y_pred_best_rf)
r2_best_rf = r2_score(y_test, y_pred_best_rf)

print(f"Tuned Random Forest MSE: {mse_best_rf}")
print(f"Tuned Random Forest R2 Score: {r2_best_rf}")
```

Key Actions:

1. **Train a Baseline Model**: We started with a Linear Regression model to establish a baseline.
2. **Train a Complex Model**: Then, we used a Random Forest model to improve predictive performance.
3. **Compare Models**: We compared the performance of both models using metrics like Mean Squared Error and R-squared.
4. **Hyperparameter Tuning (Optional)**: We optionally improved the Random Forest model using GridSearchCV.

At this stage, you've trained multiple models and evaluated their performance. Once you've run these steps and are satisfied with the results, let me know if you're ready to move on to the **Evaluation** phase, where we'll analyze the model's performance more critically. Should we proceed?

yes continue

Phase 5: Evaluation

In this phase, we will evaluate the model's performance more critically. This includes assessing how well the chosen model performs on unseen data and whether it meets the business objectives. We will also check for potential overfitting and analyze the model's strengths and weaknesses using various metrics.

Objective: Evaluate the model's performance using appropriate metrics, and ensure it meets the business objectives.

Step 1: Evaluating the Model with Metrics

We'll focus on evaluating the model using metrics like **Mean Squared Error (MSE)**, **Root Mean Squared Error (RMSE)**, and **R-squared**. These will give us a clearer understanding of how well the model predicts the Airbnb listing prices.

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

# Compute the RMSE for the final model
rmse_rf = mean_squared_error(y_test, y_pred_rf, squared=False)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest Model RMSE: {rmse_rf}")
print(f"Random Forest Model R-squared: {r2_rf}")
```

Step 2: Checking for Overfitting

To check if the model is overfitting (i.e., performing much better on the training data than on the test data), we can compare the performance on the training set and the test set.

```python
# Predictions on the training data
y_train_pred_rf = rf_model.predict(X_train)

# Evaluate the model on the training data
rmse_train_rf = mean_squared_error(y_train, y_train_pred_rf, squared=False)
r2_train_rf = r2_score(y_train, y_train_pred_rf)
```

```
print(f"Random Forest Model Training RMSE: {rmse_train_rf}")
print(f"Random Forest Model Training R-squared: {r2_train_rf}")
```

If there is a large difference between the training and test RMSE/R-squared scores, it may indicate that the model is overfitting.

Step 3: Residual Analysis

Residuals are the difference between the actual prices and the predicted prices. We want to make sure that the residuals are randomly distributed without any patterns, which would indicate the model is working well.

```python
# Compute residuals
residuals = y_test - y_pred_rf

# Plot residuals
plt.figure(figsize=(8,5))
sns.histplot(residuals, kde=True)
plt.title('Distribution of Residuals')
plt.show()

# Scatter plot of residuals vs predicted values
plt.figure(figsize=(8,5))
sns.scatterplot(x=y_pred_rf, y=residuals)
plt.axhline(0, color='r', linestyle='--')
plt.title('Residuals vs Predicted Values')
plt.show()
```

Step 4: Meeting Business Objectives

Finally, you want to evaluate whether the model achieves the business objectives defined in the **Business Understanding** phase. In this case, does the model help identify key factors that influence Airbnb listing prices, and is it accurate enough to be used for pricing strategy?

- **Accuracy**: Does the RMSE fall within an acceptable range for predicting prices?
- **Key Factors**: Does the model provide insights into the most important features driving price changes (e.g., neighborhood, room type)?

You can also look at the feature importances from the Random Forest model:

```python
# Get feature importances from the Random Forest model
importances = rf_model.feature_importances_
features = X_train.columns

# Create a DataFrame of feature importances
feature_importances = pd.DataFrame({'Feature': features, 'Importance': importances})
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feature_importances)
```

```
plt.title('Feature Importances')
plt.show()
```

This plot will help you understand which factors (e.g., neighborhood, room type) are most influential in predicting the Airbnb listing prices.

Key Actions:

1. **Evaluate the Model**: Using RMSE and R-squared, we assessed the model's performance on the test data.
2. **Check for Overfitting**: We compared the training and test set performance to ensure the model generalizes well.
3. **Residual Analysis**: We checked the distribution of residuals to verify the model's quality.
4. **Feature Importance**: We examined which features have the most impact on price prediction.
5. **Meeting Business Objectives**: We checked if the model met the business requirements for accuracy and key insights.

Once you've run these evaluations and you're satisfied with the model's performance, let me know if you're ready to move on to the **Deployment** phase, where we'll focus on making the model available for business use. Should we proceed?

Yes proceed

Phase 6: Deployment

In the final phase of the CRISP-DM process, we focus on deploying the model so that it can be used by the business or stakeholders to make decisions. This phase involves making the model accessible, integrating it into an application or service, and setting up mechanisms for ongoing monitoring and maintenance.

Objective: Deploy the model to make it available for use, and set up processes for monitoring and updating it over time.

Step 1: Saving the Model

To deploy the model, the first step is to save the trained model so that it can be reused without needing to retrain it each time.

You can save the model using **joblib** or **pickle** in Python. Here's how you can do it:

python

```python
import joblib

# Save the Random Forest model
joblib.dump(rf_model, 'airbnb_price_prediction_model.pkl')

# Load the model (to demonstrate how it can be used in production)
loaded_model = joblib.load('airbnb_price_prediction_model.pkl')

# Use the loaded model to make predictions
y_pred_loaded = loaded_model.predict(X_test)
```

This saves the model as a file, which can be loaded later in any application or service.

Step 2: Creating an API for the Model

To allow other applications (e.g., web applications, mobile apps) to interact with your model, you can deploy it via an API. Using a simple Python framework like **Flask**, you can create an API to serve predictions.

Here's an example of how you can set up an API for the model using Flask:

python

```python
# Install Flask if needed
!pip install Flask

# Create the Flask app
from flask import Flask, request, jsonify
import joblib

app = Flask(__name__)

# Load the saved model
model = joblib.load('airbnb_price_prediction_model.pkl')

# Define a route for prediction
@app.route('/predict', methods=['POST'])
def predict():
    # Get the data from the request
    data = request.get_json(force=True)

    # Convert the data into the appropriate format (you will need to ensure the data
matches the model's input format)
    input_data = [data['features']]

    # Make the prediction
    prediction = model.predict(input_data)

    # Return the prediction as a JSON response
    return jsonify({'prediction': prediction[0]})

# Run the Flask app
if __name__ == '__main__':
    app.run(debug=True)
```

To use this API, other applications can send a POST request with the required features, and the API will return the predicted price. You can deploy this Flask app using platforms like **Heroku**, **Google Cloud**, or **AWS**.

Step 3: Setting Up Monitoring and Retraining

Once the model is deployed, it’s important to monitor its performance over time to ensure it remains accurate. If the model starts to perform poorly (e.g., due to changes in the market or data patterns), you may need to retrain it.

You can set up regular evaluations to compare the model's predictions against actual prices, and if performance drops, retrain the model with updated data.

Here’s a simple outline of what you can do for monitoring:

- Track model performance metrics (e.g., RMSE, R-squared) over time.
- Log errors and discrepancies between predicted and actual prices.
- Set up alerts for when performance falls below a threshold.

Step 4: Creating a User Interface (Optional)

If needed, you can create a simple user interface (UI) where Airbnb hosts or analysts can input details like neighborhood, room type, etc., and the model will return a price prediction. This can be done using web frameworks like **Streamlit** or **Flask**, or even a desktop app using **Tkinter**.

Here’s an example of a UI with **Streamlit**:

```python
# Install Streamlit if needed
!pip install streamlit

# Create a Streamlit app
import streamlit as st
import joblib

# Load the model
model = joblib.load('airbnb_price_prediction_model.pkl')

# Define the app interface
st.title('Airbnb Price Prediction')

# Collect user input
neighbourhood_group = st.selectbox('Neighbourhood Group', ['Brooklyn', 'Manhattan',
'Queens', 'Staten Island', 'Bronx'])
room_type = st.selectbox('Room Type', ['Entire home/apt', 'Private room', 'Shared room'])
accommodates = st.slider('Accommodates', 1, 10)
minimum_nights = st.number_input('Minimum Nights', min_value=1)

# Make the prediction when the user clicks the button
if st.button('Predict'):
    input_data = [[accommodates, minimum_nights]]  # Add any other features you're using
    prediction = model.predict(input_data)
    st.write(f"Predicted Price: ${prediction[0]:.2f}")
```

This can be deployed to allow users to input features and get real-time predictions.

Key Actions:

1. **Save the Model**: We saved the trained Random Forest model for deployment.

2. **Create an API**: We demonstrated how to create a Flask API to serve predictions.
3. **Monitoring**: Set up monitoring to track the model's performance and trigger retraining when necessary.
4. **Optional User Interface**: A simple user interface was suggested to make predictions more accessible to users.

At this point, you've completed the entire CRISP-DM process! The model is trained, evaluated, and deployed, ready for business use. Do you have any additional requirements, or is there anything you'd like to refine in this deployment process?

/usr/local/lib/python3.10/dist-packages/pandas/core/dtypes/astype.py in _astype_nansafe(arr, dtype, copy, skipna)
    131   if copy or arr.dtype == object or dtype == object:
    132   # Explicit copy, or required since NumPy can't view from / to object.
--> 133   return arr.astype(dtype, copy=True)
    134
    135   return arr.astype(dtype, copy=copy)

ValueError: could not convert string to float: 'Cozy Brooklyn Oasis'
getting this error on modeling phase step 1

The error you're encountering happens because the model is expecting numerical values but is encountering strings (like `'Cozy Brooklyn Oasis'`) in one or more columns. This likely means that one or more categorical columns in your dataset (such as listing names or descriptions) haven't been properly encoded into numerical values before the model training.

To fix this issue, you need to ensure that:

1. All categorical columns are either removed if irrelevant or encoded into numerical format (like one-hot encoding).
2. Only numerical data is passed to the model.

Fixing the Issue

Let's break down the solution into a few steps:

Step 1: Check for Non-Numeric Columns

Before you pass the data to the model, you should check if there are any non-numeric columns still present in the dataset. Here's how you can list all non-numeric columns:

```python
# List all non-numeric columns in the dataset
non_numeric_columns = df_encoded.select_dtypes(exclude=[float, int]).columns
```

```python
print("Non-numeric columns: ", non_numeric_columns)
```

If you see columns like `'name'`, `'neighbourhood'`, or any other string columns, they need to be either removed or encoded.

Step 2: Remove Irrelevant Categorical Columns

Certain columns, like the `name` of the listing or its `host_name`, might not be necessary for prediction. You can remove these columns before passing the data to the model.

```python
# Drop irrelevant columns like 'name' and 'host_name'
df_encoded = df_encoded.drop(columns=['name', 'host_name'])
```

Step 3: Ensure Proper Encoding of Categorical Variables

For categorical variables that are important for the prediction (like `neighbourhood_group` and `room_type`), ensure that they are properly encoded as numerical values using one-hot encoding.

Here's how to do it:

```python
# One-hot encode categorical variables
df_encoded = pd.get_dummies(df_encoded, columns=['neighbourhood_group', 'room_type'],
drop_first=True)

# Check the dataset again to ensure only numerical columns remain
print(df_encoded.dtypes)
```

Step 4: Re-run the Modeling Phase

Once you've removed the non-numeric columns and encoded the categorical variables, try running the modeling code again. Your features (`X`) should now contain only numerical values.

```python
# Defining the features (X) and target variable (y) again after the changes
X = df_encoded.drop(['price'], axis=1)
y = df_encoded['price']

# Split the dataset into training (80%) and testing (20%) 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)

# Train the Random Forest model
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred_rf = rf_model.predict(X_test)
```

```
# Evaluate the model
from sklearn.metrics import mean_squared_error, r2_score
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest MSE: {mse_rf}")
print(f"Random Forest R2 Score: {r2_rf}")
```

This should resolve the issue by ensuring that the model only receives numerical inputs. Let me know if this works, or if you run into any further issues!

rf_model.fit(X_train, y_train)
11 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/dtypes/astype.py in _astype_nansafe(arr, dtype, copy, skipna)
    131   if copy or arr.dtype == object or dtype == object:
    132   # Explicit copy, or required since NumPy can't view from / to object.
--> 133   return arr.astype(dtype, copy=True)
    134
    135   return arr.astype(dtype, copy=copy)

ValueError: could not convert string to float: 'Cozy Brooklyn Oasis'

It seems that the issue persists because there are still string values like `'Cozy Brooklyn Oasis'` (likely from a feature like `name`) that have not been handled properly in the dataset. Here's a step-by-step guide to troubleshoot and resolve the issue:

Step 1: Identify the Problematic Column(s)

Let's first identify which columns contain string values that need to be either encoded or removed. To do this, we can list all the columns and check their data types.

```python
# Check the data types of all columns
print(df_encoded.dtypes)

# Optionally, print a sample row to identify problematic columns
print(df_encoded.sample(5))
```

Look for any columns that are still of `object` type (which indicates string values). These columns are causing the issue. The most common culprits are columns like `name`, `description`, or other textual features.

Step 2: Remove or Encode String Columns

Once you've identified the problematic columns, you can either **remove** these columns (if they are irrelevant for modeling) or **encode** them (if they contain categorical information).

Option 1: Remove Irrelevant String Columns

If columns like `name` or `description` are not necessary for the model, you can drop them.

```python
# Drop irrelevant columns like 'name', 'description', or any other non-numeric columns
df_cleaned = df_encoded.drop(columns=['name', 'description', 'host_name'])
```

Option 2: Encode Categorical Columns

If there are any categorical columns (e.g., `neighbourhood_group`, `room_type`, etc.) that are still in string format but important for the model, use **one-hot encoding** or **label encoding** to convert them into numerical values.

Here's how to one-hot encode them:

```python
# One-hot encoding for remaining categorical columns if necessary
df_cleaned = pd.get_dummies(df_cleaned, columns=['neighbourhood_group', 'room_type'],
drop_first=True)
```

Step 3: Ensure All Columns Are Numerical

After removing or encoding the problematic columns, ensure that all the columns in your dataset are numerical. You can verify this with the following code:

```python
# Check that all columns are now numerical
print(df_cleaned.dtypes)
```

Step 4: Re-run the Modeling Code

Once you have cleaned the dataset and ensured that only numerical values remain, re-run the modeling code:

```python
# Define the features (X) and target (y) again
X = df_cleaned.drop(['price'], axis=1)
y = df_cleaned['price']

# Split the data into training and test 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)
```

```
# Train the Random Forest model
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions and evaluate
y_pred_rf = rf_model.predict(X_test)

# Calculate performance metrics
from sklearn.metrics import mean_squared_error, r2_score
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest MSE: {mse_rf}")
print(f"Random Forest R2 Score: {r2_rf}")
```

This should eliminate the error related to strings in the dataset and allow you to train the model successfully.

Common Columns to Drop:

- `` `name` ``: Typically the name of the listing, irrelevant for modeling.
- `` `host_name` ``: The name of the host, which is often not relevant for predicting price.
- `` `description` ``: Text descriptions that would require advanced natural language processing if you wanted to include them.
- Any other textual features that don't contribute to the prediction.

Let me know if this resolves the issue or if the problem persists!