**1. Problem Definition & Objective**

**Interview Question:**  
*What was the goal of your project?*

**Answer:**  
The goal was to create a web application that predicts house prices based on features like living area, quality, garage size, and basement area. The app uses a trained machine learning model and provides instant price estimates through a user-friendly interface.

**2. Data Collection & Understanding**

**Interview Question:**  
*Where did you get the data, and what kind of data did you use?*

**Answer:**  
I used the Kaggle House Prices: Advanced Regression Techniques dataset, which contains features for residential homes, including both numerical and categorical attributes. The target variable is the sale price of each house.

**3. Data Preprocessing**

**Interview Question:**  
*How did you preprocess the data before modeling?*

**Answer:**

* I dropped irrelevant columns like 'Id'.
* Separated the features (X) from the target variable (y).
* Used pd.get\_dummies() to one-hot encode categorical variables so that the model could process them numerically.
* Filled any missing values with zero to avoid errors in model training.

**4. Train/Test Split**

**Interview Question:**  
*How did you split the data and why?*

**Answer:**  
I used train\_test\_split from scikit-learn to divide the data into training and testing sets (80% train, 20% test) to evaluate model performance on unseen data and prevent overfitting.

**5. Model Training**

**Interview Question:**  
*What model did you use and why?*

**Answer:**  
I used Linear Regression as a baseline because it's simple and interpretable for regression problems like price prediction. It helped me establish a baseline to compare with more complex models in the future.

**6. Feature Alignment**

**Interview Question:**  
*How did you handle feature mismatch between train and test sets or during prediction?*

**Answer:**  
After one-hot encoding, I used the .align() function to ensure both the training and test sets had the exact same columns, filling missing columns with zeros. I also saved the feature columns (model\_columns.pkl) and used them to align the input data during predictions in the deployed app.

**7. Model Saving & Loading**

**Interview Question:**  
*How did you deploy the model?*

**Answer:**  
After training, I saved the model using joblib.dump() and also saved the list of feature columns. In the Flask backend, I loaded both the model and the feature columns for prediction.

**8. Backend API with Flask**

**Interview Question:**  
*How does your backend work?*

**Answer:**  
I built a Flask API with a /predict endpoint. When a POST request with house features arrives, the API:

* Converts the JSON input to a pandas DataFrame
* One-hot encodes and aligns it with the saved columns
* Passes the data to the model to get a prediction
* Returns the predicted price as JSON

**9. Frontend UI**

**Interview Question:**  
*How does the user interact with your app?*

**Answer:**  
The user enters house features into a web form (built with HTML, CSS, and JS). When they click "Predict Price", JavaScript sends the data to the Flask backend. The prediction is displayed instantly on the page.

**10. Error Handling & Robustness**

**Interview Question:**  
*How does your system handle missing or unexpected input?*

**Answer:**  
All inputs are checked, and missing values default to zero. Any extra or missing features are handled by aligning input with the model's expected columns and filling with zeros.

**11. Technologies Used**

**Interview Question:**  
*Which libraries and frameworks did you use?*

**Answer:**

* Python (pandas, numpy, scikit-learn, joblib)
* Flask (for the API)
* HTML, CSS, JavaScript (for the frontend)
* VS Code (for development)

**12. Improvements**

**Interview Question:**  
*How could you improve this project?*

**Answer:**

* Add more features (like neighborhood, year built, etc.)
* Try advanced models (Random Forest, XGBoost)
* Use feature scaling and more advanced preprocessing
* Deploy the app online using cloud services (Heroku, Render)
* Add user authentication or a dashboard for multiple predictions

**Full Process Summary:**

**"I trained a machine learning model on house price data, built a Flask API to serve predictions, and designed a frontend that collects user input and displays results. All data processing and feature alignment steps ensure smooth integration from user form to backend prediction."**

**1. Mock Interview Q&A**

**Q1: Why did you choose this project?**  
*A1: I wanted to build something that combines data science, software engineering, and a real-world problem. House price prediction is a classic regression problem with lots of practical value, and building a web app demonstrates the full workflow from data to user.*

**Q2: Walk me through your machine learning pipeline.**  
*A2: I started by loading and cleaning the data, handling missing values and one-hot encoding categorical variables. I split the data for training/testing, trained a linear regression model, evaluated performance, and saved both the trained model and the feature columns. For deployment, I used Flask to create an API endpoint that receives user data, processes it to match the training features, and returns the prediction.*

**Q3: How do you handle categorical variables and missing values?**  
*A3: I use pd.get\_dummies() for one-hot encoding categorical features and fill missing values with zero so the model can always receive a valid input vector.*

**Q4: What would happen if a new, unseen category appears at prediction time?**  
*A4: The input data is always reindexed to match the columns used during training, with any missing columns filled as zeros. So, unseen categories are ignored, and missing features don't break the prediction pipeline.*

**Q5: How does the frontend communicate with the backend?**  
*A5: The frontend collects user input and sends it via a POST request (using JavaScript's fetch API) to the Flask backend’s /predict endpoint. The backend processes the data and returns a JSON response with the predicted price, which is then displayed to the user.*

**2. Project Summary (for Resume or LinkedIn About Section)**

**House Price Prediction Web App**:  
Developed a full-stack machine learning application to predict house prices based on user inputs. The project involved end-to-end data wrangling, feature engineering, regression modeling (with scikit-learn), and deploying the trained model as a REST API using Flask. Designed a modern HTML/CSS/JS frontend for real-time predictions. Focused on best practices for data preprocessing, robust feature alignment, and seamless backend-frontend integration.  
**Tech stack:** Python, pandas, scikit-learn, Flask, joblib, HTML, CSS, JavaScript

**3. Technical Details (deeper dive if asked in interviews)**

* **Feature Alignment:**  
  Saved the list of one-hot encoded feature columns from training, so that at prediction time, incoming data can be aligned to the same structure, ensuring the model always gets the input shape it expects.
* **Deployment:**  
  Used Flask for a lightweight REST API. The API receives JSON input, preprocesses it, aligns features, and returns a prediction, making it easy to integrate with any frontend or external service.
* **Error Handling:**  
  Defaults missing features to zero, making the system robust to incomplete input or extra fields.
* **Scalability:**  
  The modular approach allows easy swapping of models (e.g., to Random Forest or XGBoost), adding more input features, or deploying to cloud platforms.
* **UI/UX:**  
  Built a clean, mobile-friendly interface for easy experimentation and demoing the model's capabilities to users.