Beschreibung der Flask-Webanwendung und des Datenanalyse-Skripts, basierend auf den bereitgestellten Informationen:

**Die Flask-Webanwendung (app.py)**

Diese Anwendung ist mit **Flask** erstellt und verwendet die Bibliothek **pandas** zur Datenverarbeitung. Ihr Hauptzweck ist die Verwaltung von CSV-Daten über ein Web-Interface und eine RESTful-API.

* **In-Memory-Datenbank**: Die Anwendung speichert Daten in einer **einfachen In-Memory-Datenbank**, die als Pandas DataFrame implementiert ist. Es ist wichtig zu beachten, dass diese Datenbank **bei jedem Neustart der Anwendung zurückgesetzt wird**.
* **Web-Interface-Routen**:
  + **Startseite (/)**: Diese Route zeigt eine HTML-Seite an, die es Benutzern ermöglicht, CSV-Dateien hochzuladen. Sie präsentiert auch die bereits hochgeladenen Daten in einer HTML-Tabelle oder zeigt "No data available." an, falls die Datenbank leer ist. Das Interface ist als "CSV Data Manager" bekannt.
  + **Upload-Route (/upload, POST-Methode)**: Diese Route verarbeitet das Hochladen von CSV-Dateien.
    - Es wird geprüft, ob eine Datei im Request enthalten ist; andernfalls wird "No file provided" mit Statuscode 400 zurückgegeben.
    - Es wird auch überprüft, ob die hochgeladene Datei die Endung .csv hat; falls nicht, ist die Fehlermeldung "Only CSV files are allowed" mit Statuscode 400.
    - Bei erfolgreichem Upload werden die CSV-Daten mittels pd.read\_csv() eingelesen.
    - Die neuen Daten werden dann an die bestehende database angehängt, wobei pd.concat() und ignore\_index=True verwendet werden, um sicherzustellen, dass der Index neu generiert wird.
    - Eine Erfolgsmeldung "Data uploaded successfully" und der Statuscode 201 (Created) werden zurückgegeben. Bei Problemen mit der CSV-Verarbeitung gibt es eine entsprechende Fehlermeldung mit Statuscode 400.
* **RESTful-API-Routen**:
  + **Daten-Endpunkt (/api/daten, GET-Methode)**: Dieser API-Endpunkt ermöglicht das Abrufen aller gespeicherten Daten im **JSON-Format**.
  + Wenn die Datenbank leer ist, sendet die API eine Nachricht {'message': 'No data available'} mit dem Statuscode 404 (Not Found).
  + Andernfalls wird der Pandas DataFrame in JSON umgewandelt, wobei orient='records' verwendet wird, um die Daten als Liste von JSON-Objekten (Zeilen) zu formatieren.
* **Anwendungsstart**: Die Anwendung ist so konfiguriert, dass sie im **Debug-Modus** (debug=True) und mit deaktiviertem Reloader (use\_reloader=False) gestartet wird, wenn das Skript direkt ausgeführt wird.

**Datenabruf und Regressionsanalyse-Skript (analyze\_data.py)**

Dieses separate Python-Skript ist dazu konzipiert, die von der Flask-Anwendung bereitgestellten Daten abzurufen und eine lineare Regression durchzuführen, um Hauspreise vorherzusagen.

* **Verwendete Bibliotheken**: Das Skript importiert requests für den Datenabruf, pandas für die Datenverarbeitung, sklearn.model\_selection für das Aufteilen der Daten, sklearn.linear\_model für die lineare Regression und sklearn.metrics zur Modellbewertung.
* **Datenabruf**:
  + Das Skript versucht, Daten vom API-Endpunkt http://127.0.0.1:5000/api/daten abzurufen.
  + Nachdem eine GET-Anfrage gesendet wurde, wird der HTTP-Statuscode der Antwort überprüft. Ein Statuscode von 200 (OK) bedeutet einen erfolgreichen Abruf, woraufhin die JSON-Antwort in einen Pandas DataFrame umgewandelt wird.
  + Bei Fehlern (z.B. wenn die Flask-Anwendung nicht läuft oder einen Fehlercode zurückgibt) wird eine Fehlermeldung ausgegeben und das Skript beendet.
* **Datenvorverarbeitung**:
  + **Fehlende Werte**: Das Skript entfernt explizit alle Zeilen mit fehlenden Werten durch den Aufruf von data.dropna(). Dies ist besonders wichtig, da die Beispiel-Datensätze auch fehlende Werte, beispielsweise in der Spalte 'Grundstücksgröße', enthalten können. Sollten nach dem Entfernen fehlender Werte keine Daten mehr zur Analyse übrig sein, wird das Skript beendet.
  + **Merkmale und Zielvariable**:
    - Die **Merkmale (unabhängige Variablen)**, als X bezeichnet, sind 'Grundstücksgröße', 'Zimmeranzahl' und 'Garagenanzahl'.
    - Die **Zielvariable (abhängige Variable)**, als y bezeichnet, ist 'Hauspreis'.
* **Datenaufteilung**: Die bereinigten Daten werden in **Trainings- und Testsets** aufgeteilt. Dabei werden **80% der Daten für das Training** des Modells und die restlichen **20% für die Bewertung der Modellleistung** verwendet (test\_size=0.2). Ein random\_state=42 wird gesetzt, um die Reproduzierbarkeit der Aufteilung zu gewährleisten.
* **Modelltraining**: Ein **lineares Regressionsmodell** (LinearRegression) aus scikit-learn wird initialisiert und anschließend mit den Trainingsdaten (X\_train, y\_train) trainiert. Die gelernten Koeffizienten (Steigungen) und der Achsenabschnitt des Modells werden ausgegeben.
* **Vorhersage**: Das trainierte Modell wird verwendet, um Vorhersagen (y\_pred) auf den Testdaten (X\_test) zu erstellen.
* **Modellbewertung**: Die Leistung des Modells wird mithilfe des **Mean Squared Error (MSE)** bewertet. Der MSE-Wert gibt an, wie genau die Vorhersagen des Modells im Vergleich zu den tatsächlichen Werten sind.
* **Beispielvorhersagen**: Das Skript demonstriert abschließend, wie das trainierte Modell verwendet werden kann, um Vorhersagen für neue, noch nicht gesehene Datenpunkte zu treffen. Es erstellt einen Beispiel-DataFrame mit spezifischen Werten für Grundstücksgröße, Zimmeranzahl und Garagenanzahl und gibt dann die geschätzten Hauspreise für diese Beispiele aus.