**Low-Level Documentation for Flight Price Prediction App**

**1. Overview**

This low-level documentation explains the technical details and the flow of the **Flight Price Prediction App** built using Streamlit and machine learning models (Random Forest). The application takes user inputs, processes them, and provides a predicted flight price based on a pre-trained model.

**2. File Structure**

* **app.py**: The main application file containing the logic for user interaction, input data processing, and prediction.
* **requirements.txt**: A file listing all the Python dependencies required to run the app.
* **Model/**: Folder containing the pre-trained machine learning model (flight\_rf.pkl).

**3. Application Flow**

**3.1. User Input (Streamlit Widgets)**

1. **Total Stops**:
   * A slider input where the user selects the number of stops for the flight.
   * The slider range is from 0 to 4, representing flights with varying stop counts.

python

Copy code

total\_stops = st.slider('Total Stops', 0, 4, 0, 1, format="Total Stops: %d")

1. **Journey Date**:
   * A date picker for the user to choose the date of the journey.
   * The date is extracted and the day and month are stored separately.

python

Copy code

journey\_date = st.date\_input('Journey Date', datetime.today())

journey\_day = journey\_date.day

journey\_month = journey\_date.month

1. **Departure Time and Arrival Time**:
   * Sliders for selecting departure and arrival hours and minutes.
   * The departure time is taken as input for both hour and minute, and the same is done for arrival time.
   * The datetime library is used to handle time calculations and ensure correctness for flights that cross midnight.

python

Copy code

dep\_hour = st.slider('Departure Hour', 0, 23, 12, 1, format="Dep Hour: %d")

dep\_min = st.slider('Departure Minute', 0, 59, 30, 1, format="Dep Min: %d")

arrival\_hour = st.slider('Arrival Hour', 0, 23, 12, 1, format="Arrival Hour: %d")

arrival\_min = st.slider('Arrival Minute', 0, 59, 30, 1, format="Arrival Min: %d")

After receiving the times:

* + Departure and arrival times are combined into datetime objects.
  + If the arrival time is earlier than the departure time, one day is added to the arrival time to account for flights crossing midnight.
  + The duration is calculated by subtracting the departure time from the arrival time.

python

Copy code

dep\_time = datetime.strptime(f"{dep\_hour}:{dep\_min}", "%H:%M")

arrival\_time = datetime.strptime(f"{arrival\_hour}:{arrival\_min}", "%H:%M")

if arrival\_time < dep\_time:

arrival\_time += pd.Timedelta(days=1)

duration = arrival\_time - dep\_time

duration\_hours = duration.seconds // 3600

duration\_minutes = (duration.seconds // 60) % 60

1. **Number of Passengers**:
   * A number input for users to specify how many passengers are flying.

python

Copy code

num\_passengers = st.number\_input('Number of Passengers', min\_value=1, max\_value=10, value=1, step=1)

1. **Seat Type**:
   * A dropdown menu for users to select the seat type, such as Economy, Premium Economy, Business, and First Class.

python

Copy code

seat\_type = st.selectbox('Seat Type', ['Economy', 'Premium Economy', 'Business', 'First Class'])

1. **Airline**:
   * A dropdown menu for users to select the airline.
   * The selected airline is encoded into one-hot encoded features, with all other airlines set to 0.

python

Copy code

airline\_features = {'Airline\_Air India': 0, 'Airline\_GoAir': 0, 'Airline\_IndiGo': 0, ...}

1. **Source and Destination Cities**:
   * Users select the source and destination cities from predefined options.
   * The cities are also one-hot encoded similarly to the airlines.

python

Copy code

source\_features = {'Source\_Chennai': 0, 'Source\_Delhi': 0, 'Source\_Kolkata': 0, ...}

destination\_features = {'Destination\_Cochin': 0, 'Destination\_Delhi': 0, 'Destination\_Hyderabad': 0, ...}

**3.2. Data Preprocessing**

1. **Input Data Formatting**:
   * After gathering all user inputs, a pandas DataFrame is constructed, combining numerical values (like stops, duration, etc.) and encoded categorical features (airlines, source, destination).

python

Copy code

input\_data = pd.DataFrame({

'Total\_Stops': [total\_stops],

'Journey\_day': [journey\_day],

'Journey\_month': [journey\_month],

'Dep\_hour': [dep\_hour],

'Dep\_min': [dep\_min],

'Arrival\_hour': [arrival\_hour],

'Arrival\_min': [arrival\_min],

'Duration\_hours': [duration\_hours],

'Duration\_mins': [duration\_minutes],

\*\*airline\_features,

\*\*source\_features,

\*\*destination\_features,

})

1. **Feature Names Alignment**:
   * The feature names in the input\_data DataFrame should match the features the model was trained on.
   * If necessary, features that aren't included in the input are ignored, and additional features (like one-hot encoded columns) are included.

**3.3. Prediction**

1. **Model Prediction**:
   * The trained model (flight\_rf.pkl) is loaded using joblib.
   * The model is used to predict the flight price based on the prepared input data.

python

Copy code

prediction = model.predict(input\_data)

1. **Display the Result**:
   * Once the prediction is made, the app displays the ticket price using Streamlit’s st.write and st.markdown functions.
   * The result is presented in a bold and visually appealing way.

python

Copy code

st.markdown(f"<h2 style='color: #FF5733; text-align: center;'>${prediction[0]:,.2f}</h2>", unsafe\_allow\_html=True)

1. **Balloon Animation**:
   * Upon receiving the prediction, a celebratory balloon animation is triggered using st.balloons() to enhance user experience.

python

Copy code

st.balloons()

**4. Model Integration**

* **Pre-trained Model**: The app uses a Random Forest model (flight\_rf.pkl) to predict the flight prices. This model was pre-trained using historical flight data and features like the number of stops, airline, source/destination cities, departure time, etc.
* **Model Loading**:

python

Copy code

model = joblib.load('path\_to\_model/flight\_rf.pkl')

* **Prediction Flow**:
  + The model expects specific features in the input data, such as Total\_Stops, Journey\_day, Journey\_month, Dep\_hour, etc.
  + These features are dynamically generated based on the user's input.

**5. Requirements**

The application requires the following Python packages, which should be installed via requirements.txt:

* **streamlit**: For building the interactive web interface.
* **pandas**: For handling and preprocessing input data.
* **numpy**: For numerical operations (such as time duration calculations).
* **joblib**: For loading the pre-trained machine learning model.
* **scikit-learn**: For machine learning functions (Random Forest for predictions).

**6. Error Handling**

1. **Feature Mismatch**:
   * If the model encounters a mismatch in input features (e.g., missing or extra features), it raises a ValueError. This can be mitigated by ensuring the correct preprocessing steps and feature alignment.
2. **Invalid Inputs**:
   * The app includes validation checks for numerical inputs (such as total stops, number of passengers) to ensure that the values entered are within expected ranges.
3. **Model File Not Found**:
   * If the model file (flight\_rf.pkl) is not found at the specified path, an error message will be displayed, guiding the user to place the model in the correct directory.

**7. Conclusion**

This low-level documentation provides a detailed breakdown of how the app processes user inputs, interacts with the machine learning model, and presents predictions. The app is designed for flexibility, allowing easy updates to the model and additional features.

4o mini