Full Guide: Training a Taxi Fare Prediction Model

■ Project Overview
The goal is to predict the total taxi fare (total_amount) using real-world urban taxi trip data.
This involves data preprocessing, feature engineering, model building, evaluation, and deployment.

■ Real-World Applications
- Ride-Hailing Services: Provide fare estimates before booking.
- Driver Incentives: Suggest profitable times and locations.
- Urban Mobility Analytics: Study fare trends by location and time.
- Tourist Planners: Estimate travel costs.
- Taxi Sharing Apps: Enable dynamic pricing for shared rides.

■ Workflow & Tasks
1. Data Collection
- Load the taxi dataset using pandas.
- Remove duplicates and invalid entries.

2. Data Understanding
- Inspect shape, datatypes, missing values, duplicates.

3. Feature Engineering
- trip_distance: Compute using Haversine formula from pickup & dropoff coordinates.
- pickup_day: Extract weekday/weekend flag.
- am/pm: Extract day part.
- is_night: Binary flag for night rides.
- pickup_datetime: Convert from UTC to local timezone.
- trip_duration: Compute from pickup and dropoff timestamps.
- Categorical features: VendorID, RatecodeID, payment_type, etc.

4. Exploratory Data Analysis (EDA)
- Analyze fare vs distance, fare vs passengers.
- Detect outliers in fare amount, trip distance, trip duration.
- Study time-based patterns: peak hours, weekdays vs weekends, night rides.
- Visualize distributions: trip counts by hour/day.

5. Data Transformation
- Handle outliers (Z-score or IQR).
- Fix skewness (log transform or Box-Cox).
- Encode categorical variables (OneHotEncoder).

6. Feature Selection
- Use correlation analysis for numeric features.
- Chi-Square test for categorical features.
- Feature importance from tree-based models (Random Forest).

7. Model Building
- Train at least 5 models: Linear Regression, Ridge, Lasso, Random Forest, Gradient Boosting.
- Compare performance using R^2, MAE, MSE, RMSE.

8. Hyperparameter Tuning
- Use GridSearchCV or RandomizedSearchCV to optimize parameters.

9. Finalize Best Model

- Select the best performing model based on test metrics.
- Save using pickle or joblib.

10. Deployment with Streamlit
- Build a simple user interface where users input trip details (pickup, dropoff, passenger count, travel time).
- Display predicted total fare after submission.

■ Skills Learned
- Exploratory Data Analysis (EDA)
- Data Cleaning and Preprocessing
- Feature Engineering
- Regression Modeling
- Model Evaluation and Comparison
- Hyperparameter Tuning
- Deployment using Streamlit

■ Dataset Columns
- VendorID
- tpep_pickup_datetime, tpep_dropoff_datetime
- passenger_count
- pickup_longitude, pickup_latitude
- dropoff_longitude, dropoff_latitude
- RatecodeID, store_and_fwd_flag, payment_type
- fare_amount, extra, mta_tax, tip_amount, tolls_amount, improvement_surcharge
- Target: total_amount

■ Technical Tags
Pandas, EDA, Preprocessing, Machine Learning, Regression, Model Evaluation, Streamlit, Python, Scikit-learn, Train-Test Split, Haversine formula, Model Optimization.

■ Deliverables
- Jupyter Notebook with code, visualizations, and evaluation.
- Streamlit UI for predictions.

■ Timeline
The project is expected to be completed in about 10 days.

```
# Example: Train and save model
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import joblib

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build pipeline
pipeline = Pipeline([
    ("preprocessor", preprocessor),  # ColumnTransformer from preprocessing step
    ("model", RandomForestRegressor(n_estimators=300, random_state=42))
])

# Train
pipeline.fit(X_train, y_train)
```

```
# Evaluate
y_pred = pipeline.predict(X_test)
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Test RMSE:", rmse)

# Save
joblib.dump(pipeline, "tripfare_model.pkl")
```