**Documentation for Amazon Delivery Time Prediction**

Data Loading:

The dataset (amazon\_delivery.csv) is loaded using pandas

df = pd.read\_csv("amazon\_delivery.csv")

Initial inspection is done with

df.info(), df.describe(), df.head(),df.shape(),df.isnull()

Handling Missing & Inconsistent Data:

Numerical columns (e.g., Agent\_Rating): missing values replaced with median.

Categorical columns (e.g., Weather, Traffic): missing values filled with "unknown".

Duplicates: removed using:

df.drop\_duplicates(inplace=True)

Feature Engineering:

**Calculate Distance:**

Haversine formula: To compute the shortest distance between two points on the Earth’s surface, given their latitude and longitude. It accounts for the fact that the Earth is spherical.

1. Earth Radius (R)

We assume the Earth’s radius ≈ 6,371 km.

This lets us convert an angular distance into kilometers.

1. Convert degrees → radians

Lat/long values in the dataset are in degrees.

All trigonometric functions (sin, cos) expect radians.

map (np.radians, [lat1, lon1, lat2, lon2]) converts everything.

1. Compute differences

dlat = lat2 - lat1

dlon = lon2 - lon1

Change in latitude and longitude.

1. Haversine formula core (a)

a = sin²(dlat/2) + cos(lat1) \* cos(lat2) \* sin²(dlon/2)

Uses trigonometry to measure the central angle between the two points.

1. Angular distance (c)

c = 2 \* arcsin(sqrt(a))

Converts the "a" value into the actual angle (in radians).

1. Final distance

distance = R \* c

Multiply Earth’s radius by the angle → distance in kilometers.

**Time-based features**:

Order\_Hour → Helps capture traffic/delivery delays at certain times.

Order\_DayOfWeek / Order\_DayName → Patterns like fewer orders on Mondays, peaks on weekends.

Order\_Month → Seasonal trends (e.g., festive season delays).

Is\_RushHour → Binary flag for peak traffic times.

Is\_Weekend → Captures slower weekend deliveries.

Example:

df[“order\_Datetime”].dt. hour

df[“order\_Datetime”].dt. daysofweek

df[“order\_Datetime”].dt. day\_name

df[“order\_Datetime”].dt. month

df[“Order\_Hour”]. apply (lambda x: 1 if 7 <= x <= 9 or 17 <= x <= 20 else 0)

df['Order\_DayOfWeek']. apply (lambda x: 1 if x >= 5 else 0)

Data Cleaning:

All categorical values standardized (Remove Spaces)

Ensures consistency (e.g., "Sunny", "sunny " → "sunny").

Exploratory Data Analysis (EDA):

1. Distribution of delivery time (Histograms, Kernel Density Estimation)
2. Time-based Trends (Boxplot, Bar chart)
3. Distance vs Delivery Time (Scatter plots, Regressions plot)
4. Traffic & Weather Impact (Boxplot, violin plot)
5. Vehicle, Area, Category Distribution
6. Correlation Heatmap

**Label Encoding:**

Use Label Encoding for handling Categorical columns.

cat\_cols = ["Traffic", "Weather", "Area", "Vehicle", "Category"]

df\_encoded = df.copy()

le = LabelEncoder()

for col in cat\_cols:

    df\_encoded[col] = le.fit\_transform(df\_encoded[col])

**Define Features & Target**

useful\_cols = [ "Agent\_Age", "Agent\_Rating", "Traffic", "Weather", "Is\_RushHour", "Is\_Weekend", "Order\_Hour",  "Area", "Vehicle”, "Category”,    "Distance\_km" ]

X = df\_encoded[useful\_cols]

y = df\_encoded["Delivery\_Time"]

Model Development:

**Train-Test Split:**

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.2, random\_state=42 )

**Train Models & hyperparameter search space:**

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.svm import SVR

from xgboost import XGBRegressor

param\_grids = {

    "Decision Tree": {

        "max\_depth": randint(3, 20),

        "min\_samples\_split": randint(2, 10),

        "min\_samples\_leaf": randint(1, 5)

        },

     "Random Forest": {

         "n\_estimators": randint(50, 200),

         "max\_depth": randint(5, 20),

         "min\_samples\_split": randint(2, 10),

         "min\_samples\_leaf": randint(1, 5)

         },

    "Gradient Boosting": {

        "n\_estimators": randint(50, 200),

        "learning\_rate": uniform(0.01, 0.3),

        "max\_depth": randint(3, 10)

        },

    "SVR": {

        "C": uniform(0.1, 10),

        "epsilon": uniform(0.01, 1),

        "kernel": ["linear", "rbf", "poly"]

        },

    "XGBoost": {

        "n\_estimators": randint(50, 200),

        "learning\_rate": uniform(0.01, 0.3),

        "max\_depth": randint(3, 10),

        "subsample": uniform(0.6, 0.4)

    }

}

models = { "Linear Regression": LinearRegression(),

          "Random Forest": RandomForestRegressor(n\_estimators=100, random\_state=42),

          "Gradient Boosting": GradientBoostingRegressor(n\_estimators=100, random\_state=42),

          "Decision Tree": DecisionTreeRegressor(random\_state=42),

          "Support Vector Regressor": SVR(kernel="rbf"),

          "XGBoost": XGBRegressor(n\_estimators=100, random\_state=42)

            }

**Evaluate models using metrics like RMSE, MAE, and R-squared:**

for name, model in models.items():

    if name in param\_grids:

        search = RandomizedSearchCV(

            model,param\_distributions=param\_grids[name],

            n\_iter=20,cv=3,scoring="neg\_mean\_squared\_error",

            random\_state=42,n\_jobs=1

        )

        search.fit(X\_train,y\_train)

        best\_model = search.best\_estimator\_

    else:

        best\_model = model.fit(X\_train, y\_train)

y\_pred = best\_model.predict(X\_test)

    rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

    mae = mean\_absolute\_error(y\_test, y\_pred)

    r2 = r2\_score(y\_test, y\_pred)

**Track performance metrics using MLflow:**

mlflow.set\_experiment("Delivery Time Prediction - Tuned Models")

mlflow.set\_tracking\_uri("http://127.0.0.1:5000")

for name, model in models.items():

with mlflow.start\_run(run\_name=name):

………………………………………………….

if "XGB" in name:

mlflow.xgboost.log\_model(

xgb\_model=best\_model,

artifact\_path=name.replace(" ", "\_") + "\_model")

else:

mlflow.sklearn.log\_model(sk\_model=best\_model,

artifact\_path=name.replace(" ", "\_") + "\_model")

After execute the cell click in the uri and go to mlflow and compare the models and which is the best just register the model.

result = mlflow.register\_model( model\_uri="runs:/3e74d55096714b5bbc3aa249e877f851/XGBoost\_model",

name="XGBoost\_RegModel"

)

After that load the model and predict

import mlflow.pyfunc

loaded\_model = mlflow.pyfunc.load\_model("models:/XGBoost\_RegModel/1")

Application Development:

import streamlit as st

import xgboost as xgb

import pandas as pd

from sklearn.preprocessing import LabelEncoder

with st.form("prediction\_form"):

    col1, col2 = st.columns(2)

    with col1:

        Agent\_Age = st.number\_input("Agent Age", min\_value=18, max\_value=65, value=30)

        Agent\_Rating = st.slider(

            "Agent Rating", min\_value=1.0, max\_value=5.0, value=4.5

        )

        Traffic = st.selectbox("Traffic Condition", df["Traffic"].unique())

        Weather = st.selectbox("Weather Condition", df["Weather"].unique())

        Is\_RushHour = st.radio("Is it Rush Hour?", ["YES", "NO"])

        Is\_Weekend = st.radio("Is it Weekend?", ["YES", "NO"])

        Is\_RushHour = 1 if Is\_RushHour == "YES" else 0

        Is\_Weekend = 1 if Is\_Weekend == "YES" else 0

    with col2:

        Order\_Hour = st.number\_input(

            "Order Hour (0-23)", min\_value=0, max\_value=23, value=12

        )

        Area = st.selectbox("Area Type", df["Area"].unique())

        Vehicle = st.selectbox("Vehicle Type", df["Vehicle"].unique())

        Category = st.selectbox("Product Category", df["Category"].unique())

        Distance\_km = st.number\_input("Distance (km)", min\_value=0.0, value=5.0)

    submitted = st.form\_submit\_button("Predict Delivery Time")

I deploy the application.py in streamlit (https://predictdelivery-time.streamlit.app/)