Importing Libraries

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_absolute_error
import joblib
```

ემოგვაქვს ყველა საჭირო იმპორტი. ესენია:

- **Pandas** და **NumPy** მონაცემთა დამუშავებისა და რიცხვითი ოპერაციებისთვის.
- **Matplotlib** და **Seaborn** მონაცემთა ვიზუალიზაციისთვის.
- **Scikit-learn**-ის მოდულები მანქანური სწავლების **Preprocessing**-ისთვის, **Random Forest**-ის გამოყენებით მოდელის ტრენინგისთვის და შეფასებისთვის.
- **Joblib** გაწვრთნილი მოდელის შემდგომი გამოყენებისთვის შესანახად.

Load and Preview Data

```
df = pd.read_csv("Food_Delivery_Times.csv")
df = df.dropna()
df.head()
```

ჩვენ ვტვირთავთ ჩვენს მონაცემთა ნაკრებს Pandas-ის გამოყენებით და ვასუფთავებთ მას. ვშლით ნებისმიერ რიგს, რომელიც შეიცავს გამოტოვებულ მნიშვნელობებს (missing values), რათა დარწმუნდეთ, რომ მოდელზე არასრული მონაცემები ცუდად არ იმოქმედებს. head() ფუნქცია გვაჩვენებს პირველ რამდენიმე რიგს, რათა გავიაზროთ სტრუქტურა.

Feature selection

```
X = df.drop(columns=["Order_ID", "Delivery_Time_min"])
y = df["Delivery_Time_min"]

cat_features = ["Weather", "Traffic_Level", "Time_of_Day", "Vehicle_Type"]
num_features = ["Distance_km", "Preparation_Time_min", "Courier_Experience_yrs"]
```

შემდეგ, ჩვენ ვამზადებთ ჩვენს მახასიათებლებს და მიზანს. მიზანია გამოვთვალოთ მიწოდების დრო წუთებში. ჩვენ ვშლით არასასარგებლო სვეტებს, როგორებიცაა Order_ID და ვადგენთ, რომელი მახასიათებლებია კატეგორიული — მაგალითად, ამინდი და ავტომობილის ტიპი — და რომელია რიცხვითი, მაგალითად, მანძილი და მომზადების დრო.

Data Peprocessing Pipelines

```python
from sklearn.pipeline import Pipeline

cat_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

num_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="mean"))
])

preprocessor = ColumnTransformer([
    ("cat", cat_transformer, cat_features),
    ("num", num_transformer, num_features)
])
```

სანამ ჩვენს მოდელს გავწვრთნით, უნდა დავრწმუნდეთ, რომ ჩვენი მონაცემები სუფთაა და მანქანური სწავლებისთვის სწორ ფორმატშია.

```
cat_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])
```

□ს კატეგორიულ დათას თუ nan აქვს გადააქცევს იმად რაც ყველაზე ხშირად გამოიყენება. □ა შემდეგ შესაბამის რიცხვით კოდს მიანიჭებს. Encoding. □ნალოგიურად რიცხვითი მონაცემების შემთხვევაში, მაგრამ ის არ გადის encoding-ის ეტაპს.

```
preprocessor = ColumnTransformer([
    ("cat", cat_transformer, cat_features),
    ("num", num_transformer, num_features)
])
```

და ბოლოს, ჩვენ ორივე pipeline-ს ColumnTransformer-ის გამოყენებით ვაერთიანებთ, რათა ფუნქციების თითოეული ნაკრები სწორად დამუშავდეს. ეს ნაბიჯი უზრუნველყოფს, რომ მოდელის შემდგომში გაწვრთნისას ის მიიღებს სუფთა და თანმიმდევრულ მონაცემებს.

Model training

```
model = Pipeline([
    ("preprocessor", preprocessor),
    ("regressor", RandomForestRegressor(n_estimators=100, random_state=42))
])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae:.2f} minutes")
```


```
model = Pipeline([
    ("preprocessor", preprocessor),
    ("regressor", RandomForestRegressor(n_estimators=100, random_state=42))
])
```


□ს მუშაობს preoricessing-ზე , როგორც ეს ზემოთ ავღნიშნეთ, და შემდეგ randomforestregressor-ით უყურებს დათას და ითვლის მათი შედეგების საშუალოს.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model.fit(X_train, y_train)
```

მონაცემთა ნაკრები დავყავით სასწავლო და ტესტირების ნაკრებებად, საიდანაც 80% გამოყენებულია ტრენინგისთვის, ხოლო 20% ტესტირებისთვის. Training Set- იმისთვის, რომ მოდელმა ისწავლოს, რომ მაგალითად რაც მეტია გზა დროც იმატებს, და Testing Set- რათა ვნახოთ, თუ რამდენად კარგად იმუშავა მოდელმა.

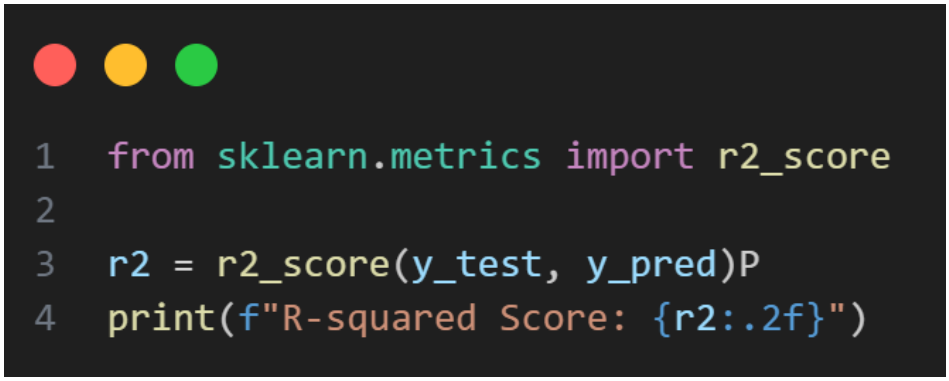
```
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae:.2f} minutes")
```

და ბოლოს, მოდელის მუშაობას Mean absolute error-ის გამოყენებით ვითვლით საშუალო დროს პროგნოზირებულ და ფაქტობრივ მიწოდების დროს შორის.

Saving the model

როგორც კი გვექნება გაწვრთნილი მოდელი, მას .pkl ფაილად ვინახავთ Joblib-ის გამოყენებით. ეს საშუალებას გვაძლევს მოგვიანებით ხელახლა გამოვიყენოთ მოდელი ხელახალი გაწვრთნის გარეშე. ეს განსაკუთრებით სასარგებლოა მოდელის რეალურ აპლიკაციაში განსათავსებლად.

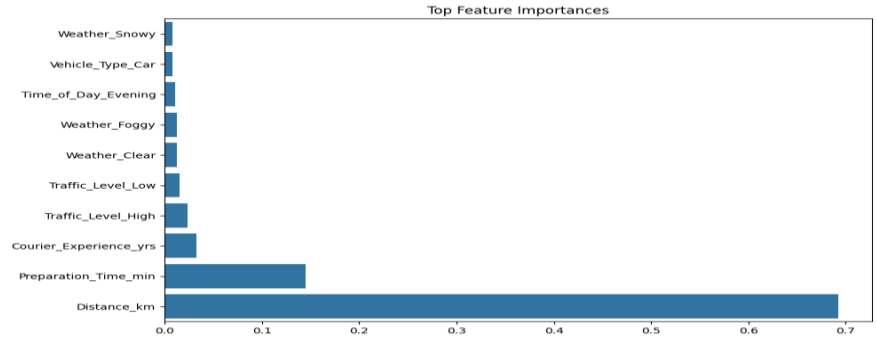
R2



```
1  from sklearn.metrics import r2_score
2
3  r2 = r2_score(y_test, y_pred)P
4  print(f"R-squared Score: {r2:.2f}")
```

□თვვლის მუშაობის სისწორეს, რაც აღინიშნება R2 score-თი.

Feature importance




```python
model_rf = model.named_steps["regressor"]
encoded_features = model.named_steps["preprocessor"].transformers_[0][1].named_steps["encoder"].get_feature_names_out(cat_features)
all_features = np.concatenate([encoded_features, num_features])

importances = model_rf.feature_importances_
indices = np.argsort(importances)[-10:]

plt.figure(figsize=(10,6))
sns.barplot(x=importances[indices], y=all_features[indices])
plt.title("Top Feature Importances")
plt.tight_layout()
plt.show()
```

ეს კოდის ბლოკი ვიზუალურად ასახავს **10** ყველაზე მნიშვნელოვან ფუნქციას, რომლებიც გამოიყენება **Pipeline**-ში გაწვრთნილი **Random Forest** მოდელის მიერ.

Access the Random Forest model

Get encoded feature names (for categorical features)

Concatenate all feature names

Get feature importances from the Random Forest

Sort and select top 10 features

Plot the top 10 important features

Pairprol of key numerical features.

```
import seaborn as sns
sns.pairplot(df[["Distance_km", "Preparation_Time_min", "Courier_Experience_yrs", "Delivery_Time_min"]])
plt.suptitle("Pairplot of Key Numerical Features", y=1.02)
plt.show()
```

ეს კოდი ქმნის წყვილთა დიაგრამას, რომელიც წარმოადგენს გაფანტული დიაგრამების მატრიცას, რომელიც აჩვენებს მონაცემთა ნაკრებში მრავალ რიცხვით მახასიათებელს შორის კავშირებს. მოდით, ავხსნათ ეს ეტაპობრივად:

```
import seaborn as sns
```

`seaborn` is a powerful Python visualization library built on top of `matplotlib`.

```
sns.pairplot(df[["Distance_km", "Preparation_Time_min",
"Courier_Experience_yrs", "Delivery_Time_min"]])
```

This line creates a **pairplot** of 4 selected columns from the DataFrame `df`

```
plt.suptitle("Pairplot of Key Numerical Features", y=1.02)
plt.show()
```

Adds a **main title** above the pairplot. y=1.02 moves the title slightly **above** the plot so it doesn't overlap. And then Renders the final plot in your notebook or script.

Correlation heatmap.

```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```

ეს კოდი ქმნის კორელაციის სითბურ რუკას, რომელიც ვიზუალურად აჩვენებს, თუ რამდენად ძლიერად არის ერთმანეთთან დაკავშირებული თქვენი მონაცემთა ნაკრების რიცხვითი მახასიათებლები.

```
plt.figure(figsize=(8,6))
```

Starts a new figure for the plot. Sets the figure size to 8 inches wide and 6 inches tall. Ensures the heatmap fits nicely and is readable.

```
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm",
fmt=".2f")
```

Computes the **correlation matrix** between all **numerical columns** in `df`. Each value in the matrix is a correlation coefficient (range: −1 to 1): `+1` = perfect positive correlation. `0` = no correlation. `-1` = perfect negative correlation.

Example: if `Distance_km` and `Delivery_Time_min` are highly correlated, the heatmap will show a large positive number for that pair.

```
plt.title("Correlation Heatmap")
plt.show()
```

Adds a title to the plot for context. Renders the heatmap in your notebook or output window.

Example input dictionary

```python
sample_input = {
    "Distance_km": 16.42,
    "Weather": "Clear",
    "Traffic_Level": "Medium",
    "Time_of_Day": "Evening",
    "Vehicle_Type": "Bike",
    "Preparation_Time_min": 20,
    "Courier_Experience_yrs": 3
}

input_df = pd.DataFrame([sample_input])
predicted_time = model.predict(input_df)[0]
print(f"Predicted Delivery Time: {predicted_time:.2f} minutes")
```

ეს კოდი შეყვანის სახით იღებს მიწოდების ერთ მაგალითს, გარდაქმნის მას DataFrame-ად, გადასცემს მას თქვენს მიერ გაწვრთნილ მოდელის Pipeline-ს და ბეჭდავს მიწოდების პროგნოზირებულ დროს.

Convert the dictionary to a DataFrame

```
predicted_time = model.predict(input_df)[0]
```

Make a prediction