As the decision tree regression and random forest regressor got more accuracy. A model is made using random forest regression to predict the TRIP FARE by taking no.of passengers, trip distance, pick up longitude, pick up latitude, drop off longitude, drop off latitude.

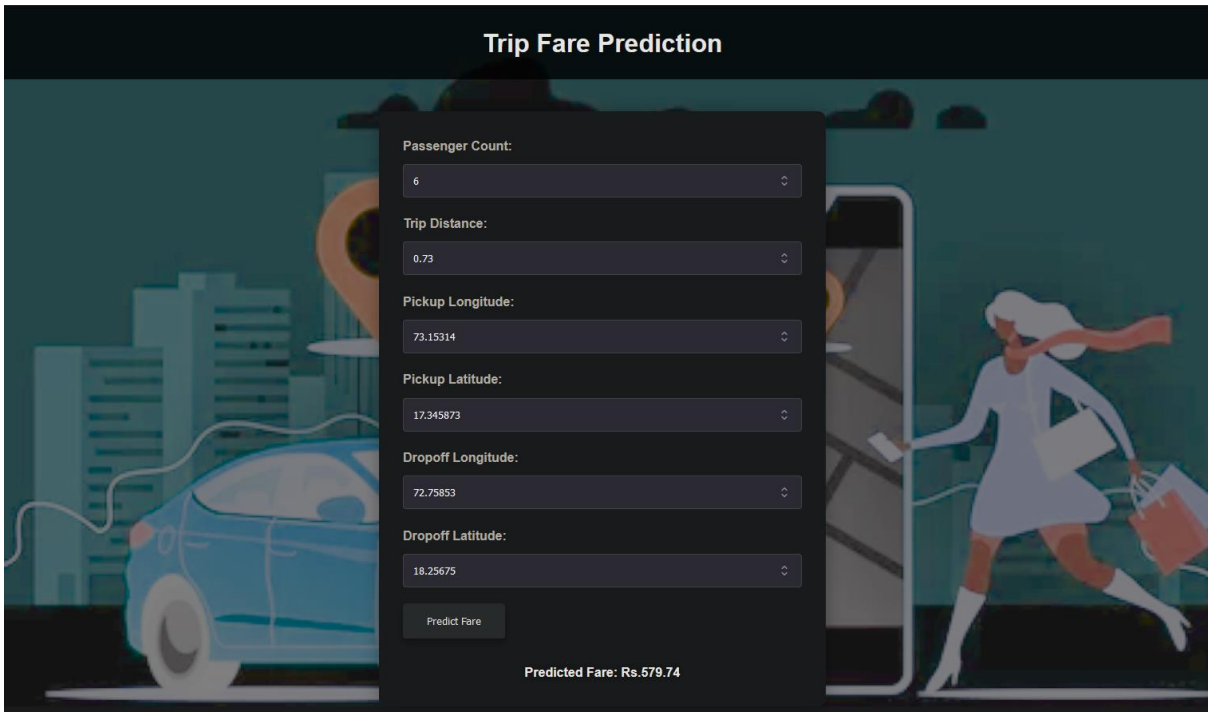
```python
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
import joblib

# Load the dataset
data = pd.read_csv('uber_data.csv')

# Convert 'payment_type' column to numeric, handling errors by coercing to NaN
data['payment_type'] = pd.to_numeric(data['payment_type'], errors='coerce')

# Drop rows with missing values
data.dropna(inplace=True)

# Define features and target variable
X = data.drop(columns=['total_amount'])  # Features
y = data['total_amount']  # Target variable

# Define numerical and categorical features
numeric_features = ['passenger_count', 'trip_distance']
categorical_features = ['pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude']

# Define preprocessing steps for numerical and categorical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps for numerical and categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Define the model
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('feature_selection', SelectKBest(score_func=f_regression, k=5)),
                        ('regressor', RandomForestRegressor())])

# Train the model
model.fit(X, y)

# Save the trained model to a file
joblib.dump(model, 'model.pkl')

# Take user input for prediction
passenger_count = int(input("Enter the number of passengers: "))
trip_distance = float(input("Enter the trip distance: "))
pickup_latitude = float(input("Enter the pickup latitude: "))
pickup_longitude = float(input("Enter the pickup longitude: "))
dropoff_latitude = float(input("Enter the dropoff latitude: "))
dropoff_longitude = float(input("Enter the dropoff longitude: "))

# Make predictions on new data
new_data = pd.DataFrame({
    'passenger_count': [passenger_count],
    'trip_distance': [trip_distance],
    'pickup_latitude': [pickup_latitude],
    'pickup_longitude': [pickup_longitude],
    'dropoff_latitude': [dropoff_latitude],
    'dropoff_longitude': [dropoff_longitude]
})

# Predict trip fare
predicted_fare = model.predict(new_data)
print("Predicted Trip Fare:", (predicted_fare)*80)
```

The trained model was saved in the form of pkl (python pickle). It's often used to save trained models to disk so that they can be reused later without needing to retrain them every time. When a POST request is made to this endpoint, it expects JSON data containing the features required for prediction. It then converts this JSON data into a panda Data Frame and passes it through the trained model pipeline to make predictions. Finally, it returns the prediction as a JSON response.

```python
from flask import Flask, render_template, request, jsonify
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
import joblib

app = Flask(__name__, template_folder='template')

# Load the trained model
model_pipeline = joblib.load('model.pkl')

# Load the HTML page
@app.route('/')
def home():
    return render_template('index.html')

# Predict fare
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    new_data = pd.DataFrame(data, index=[0])

    fare_prediction = model_pipeline.predict(new_data)

    return jsonify(float(fare_prediction[0])*80)

if __name__ == '__main__':
    app.run(debug=True)
```

OUTPUT:

The final output of the project is a web page that asks passenger count, trip distance, pickup longitude, pickup latitude, drop off longitude, drop off latitude and predicts the trip fare using the random forest regressor model.



CONCLUSION:

In this project we first performed preprocessing and EDA steps on dataset and then we used four machine learning algorithms and selected the algorithm with highest accuracy for training the model by splitting the dataset. After training the model we saved it in .pkl file for future use. We then made a flask based web page which uses the saved pkl model for prediction of TRIP FARE.