```
# ============================================================
# LAB PRACTICAL 7 – Uber Fare Prediction
# ============================================================

# Step 1: Import required libraries
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 StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error

# Step 2: Load the dataset
df = pd.read_csv("/content/uber.csv")
print("Dataset loaded successfully!")
df.head()
```

```
Dataset loaded successfully!
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	p
0	24238194	52:06.0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	
1	27835199	04:56.0	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	
2	44984355	45:00.0	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	

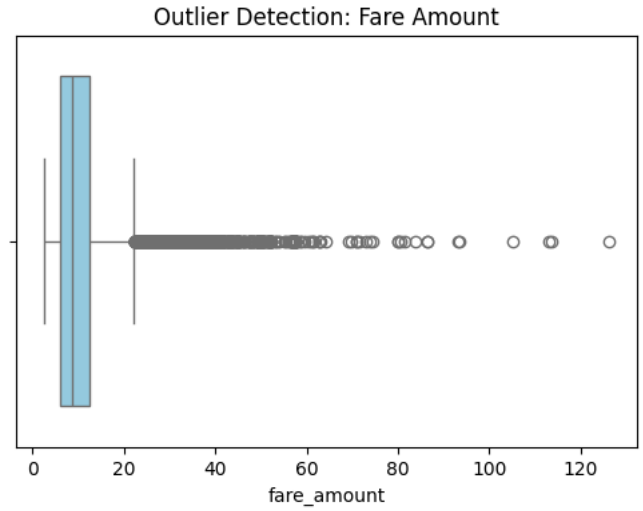
New Section

```
# Step 3: Data Preprocessing

# Remove rows with null values
df = df.dropna()

# Remove invalid fares and passenger counts
df = df[(df['fare_amount'] > 0) & (df['passenger_count'] > 0)]

# Remove extreme coordinates (outside valid NYC range if applicable)
df = df[(df['pickup_latitude'] <= 90) & (df['pickup_latitude'] >= -90)]
df = df[(df['dropoff_latitude'] <= 90) & (df['dropoff_latitude'] >= -90)]
df = df[(df['pickup_longitude'] <= 180) & (df['pickup_longitude'] >= -180)]
df = df[(df['dropoff_longitude'] <= 180) & (df['dropoff_longitude'] >= -180)]

# Haversine distance function
def haversine(lon1, lat1, lon2, lat2):
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2)**2 + np.cos(lat1)*np.cos(lat2)*np.sin(dlon/2)**2
    c = 2*np.arcsin(np.sqrt(a))
    r = 6371  # radius of Earth (km)
    return c * r

# Create new distance feature
df['distance_km'] = haversine(df['pickup_longitude'], df['pickup_latitude'],
                              df['dropoff_longitude'], df['dropoff_latitude'])

# Keep reasonable distance values (to remove wrong GPS data)
df = df[(df['distance_km'] > 0) & (df['distance_km'] < 100)]

print("✅ Cleaned dataset shape:", df.shape)
df.head()
```

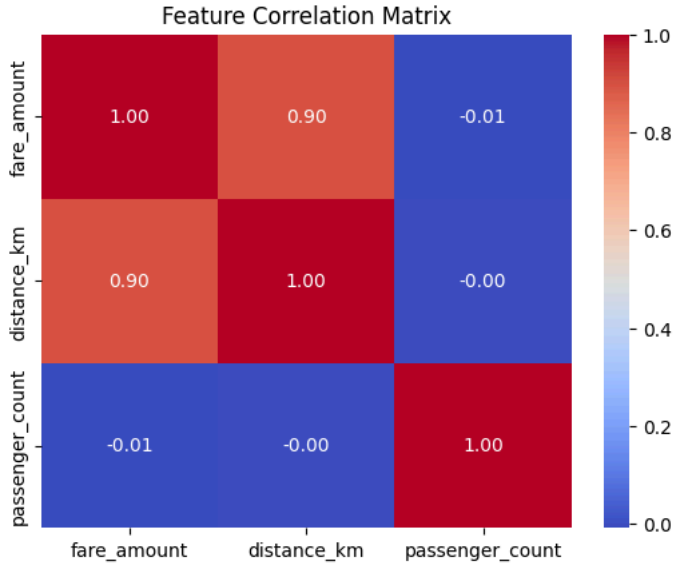
Cleaned dataset shape: (11404, 10)

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	p
0	24238194	52:06.0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	
1	27835199	04:56.0	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	
2	44984355	45:00.0	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	

```
# Step 4: Identify Outliers using Boxplot
plt.figure(figsize=(6,4))
sns.boxplot(x=df['fare_amount'], color='skyblue')
plt.title("Outlier Detection: Fare Amount")
plt.show()

# Optional: remove fares beyond typical limits
df = df[(df['fare_amount'] < 100) & (df['fare_amount'] > 2)]
```




```
# Step 5: Check Correlation
corr = df[['fare_amount', 'distance_km', 'passenger_count']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Matrix")
plt.show()
```



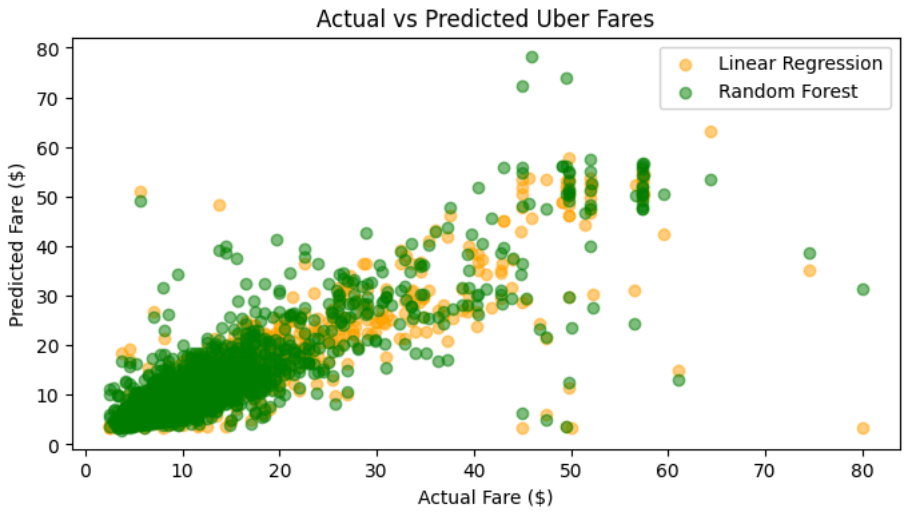
```
# Step 6: Prepare data for modeling
X = df[['distance_km', 'passenger_count']]
y = df['fare_amount']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Step 7: Train Linear Regression
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

# Step 8: Train Random Forest Regression
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

```
# Step 9: Evaluate both models
r2_lr = r2_score(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))

r2_rf = r2_score(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))

print("📊 MODEL EVALUATION RESULTS 📊")
print(f"Linear Regression  -> R² = {r2_lr:.3f}, RMSE = {rmse_lr:.3f}")
print(f"Random Forest      -> R² = {r2_rf:.3f}, RMSE = {rmse_rf:.3f}")
```

```
📊 MODEL EVALUATION RESULTS 📊
Linear Regression  -> R² = 0.782, RMSE = 4.524
Random Forest      -> R² = 0.742, RMSE = 4.929
```

```
# Step 10: Visualize Predictions
plt.figure(figsize=(8,4))
plt.scatter(y_test, y_pred_lr, color='orange', alpha=0.5, label='Linear Regression')
plt.scatter(y_test, y_pred_rf, color='green', alpha=0.5, label='Random Forest')
plt.xlabel("Actual Fare ($)")
plt.ylabel("Predicted Fare ($)")
plt.legend()
plt.title("Actual vs Predicted Uber Fares")
plt.show()
```




```
print(df)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-1093298923.py in <cell line: 0>()
----> 1 print(df)

NameError: name 'df' is not defined
```

```
Start coding or generate with AI.
```