In [54]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("uber.csv")
df.head()
print(df.info())
print("-----------------------")
#count of missing values
print(df.isnull().sum())
df = df.dropna()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
None
-----------------------
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

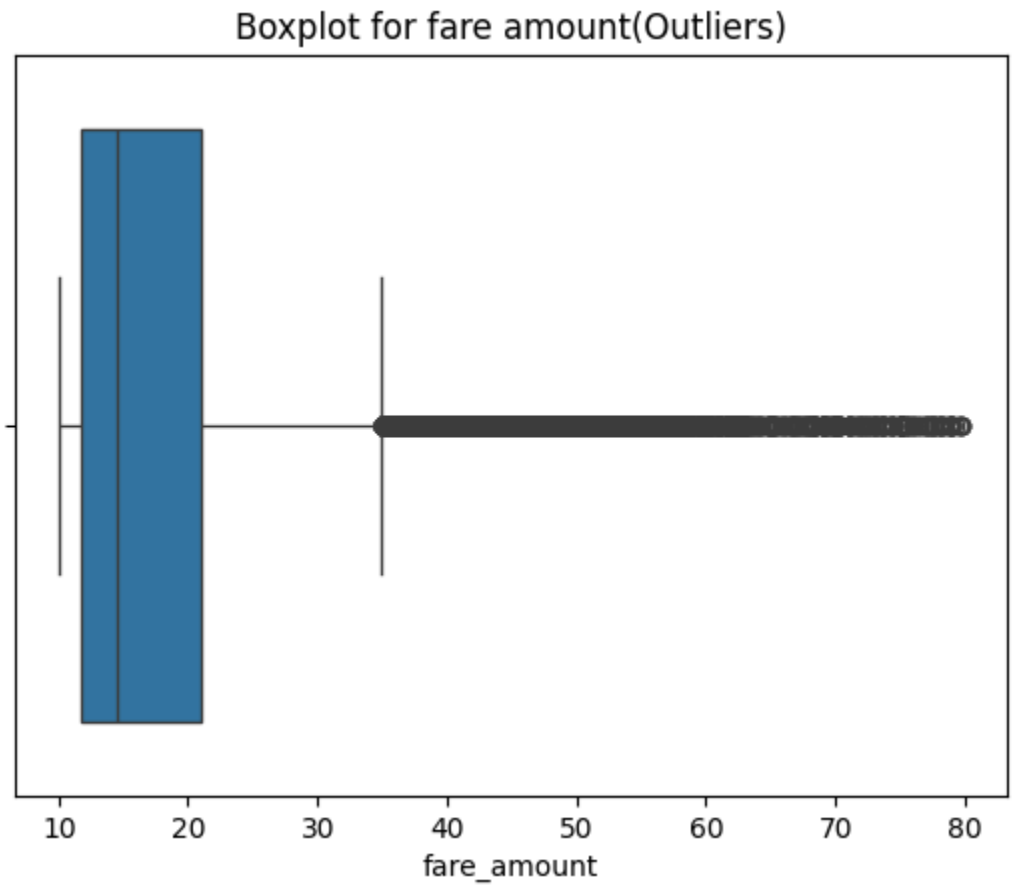
In [59]:
```
# Calculate distance using simple Euclidean formula
df["distance"] = (
    ((df['dropoff_latitude'] - df['pickup_latitude'])**2 +
     (df['dropoff_longitude'] - df['pickup_longitude'])**2) * 0.5
)

# Filter for reasonable fare, passenger count, and distance
df = df[(df['fare_amount'] > 10) & (df['fare_amount'] < 80)]
```

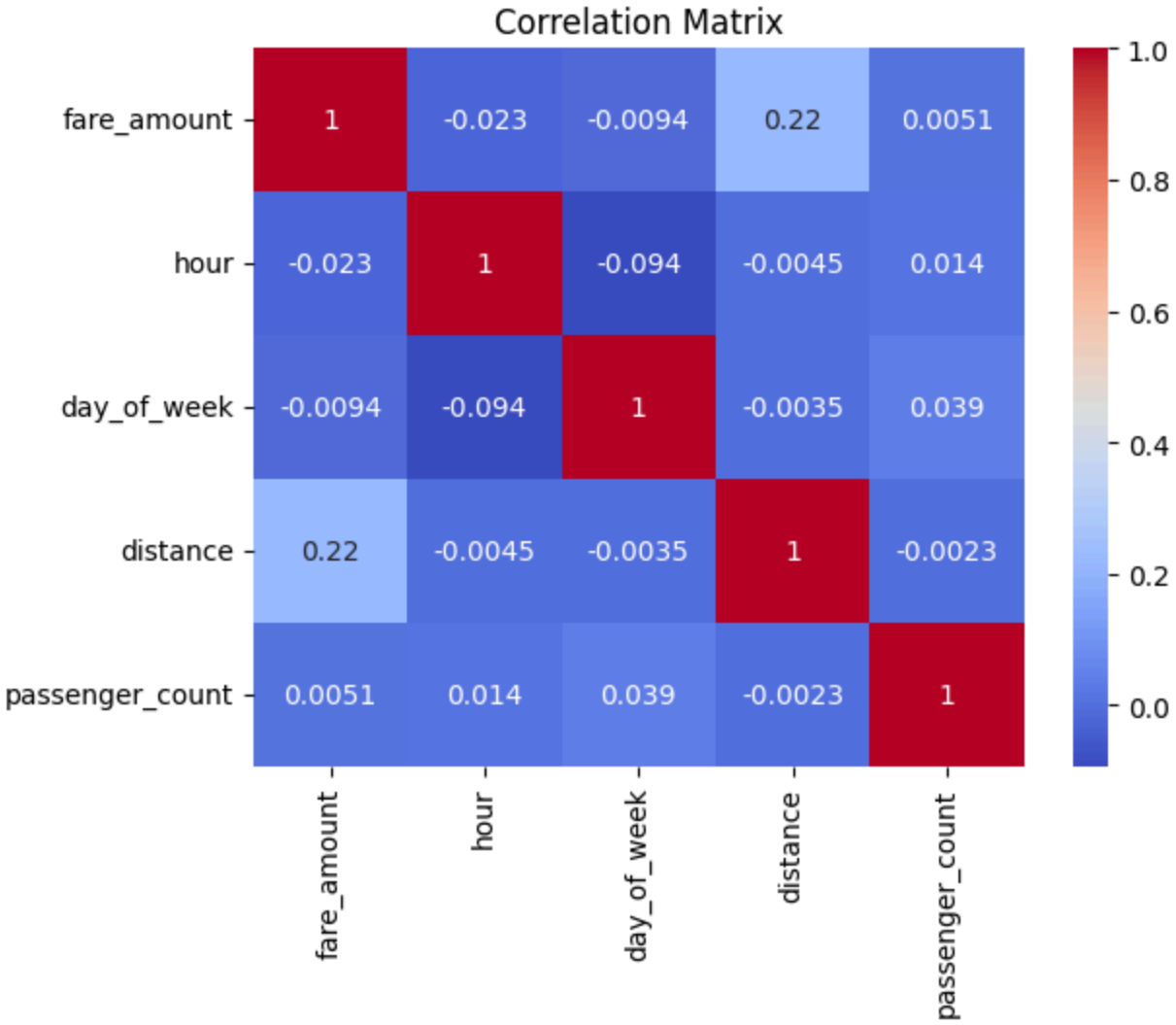
```
df = df[(df['passenger_count'] > 0) & (df['passenger_count'] <6)]
df = df[df['distance'] < 5]
```

In [60]:
```
# Extract time features
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
df['hour'] = df['pickup_datetime'].dt.hour
df['day_of_week'] = df['pickup_datetime'].dt.dayofweek
```

In [61]:
```
#Boxplot
sns.boxplot(x=df['fare_amount'])
plt.title("Boxplot for fare amount(Outliers)")
plt.show()
```



In [62]:
```
corr = df[['fare_amount','hour','day_of_week','distance','passenger_count']].c
sns.heatmap(corr,annot=True,cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



In [74]:
```
X=df[['distance','hour','passenger_count','day_of_week']]
Y=df['fare_amount']
```

In [75]:
```
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.25, random_state=42
)
```

In [76]:
```
print(X_train.shape, Y_train.shape)
```

```
(54475, 4) (54475,)
```

In [77]:
```
lr = LinearRegression()
lr.fit(X_train, Y_train)
y_pred_lr = lr.predict(X_test)
```

In [79]:
```
rf=RandomForestRegressor(n_estimators=100,random_state=42)
rf.fit(X_train,Y_train)
y_pred_rf=rf.predict(X_test)
```

In [86]:
```
def evaluate(Y_true,Y_pred,model_name):
```

```
    rmse=np.sqrt(mean_squared_error(Y_true,Y_pred))
    r2=r2_score(Y_true,Y_pred)
    print(f"{model_name} Result: ")
    print(f"  R² Score: {r2:.4f}")
    print(f"  RMSE: {rmse:.4f}\n")
```

In [89]:
```
evaluate(Y_test,y_pred_lr,"LinearRegression")
evaluate(Y_test,y_pred_rf,"RandomForest")
```

```
LinearRegression Result:
  R² Score: 0.0590
  RMSE: 10.9749

RandomForest Result:
  R² Score: 0.7307
  RMSE: 5.8715
```

In [ ]:
```

```