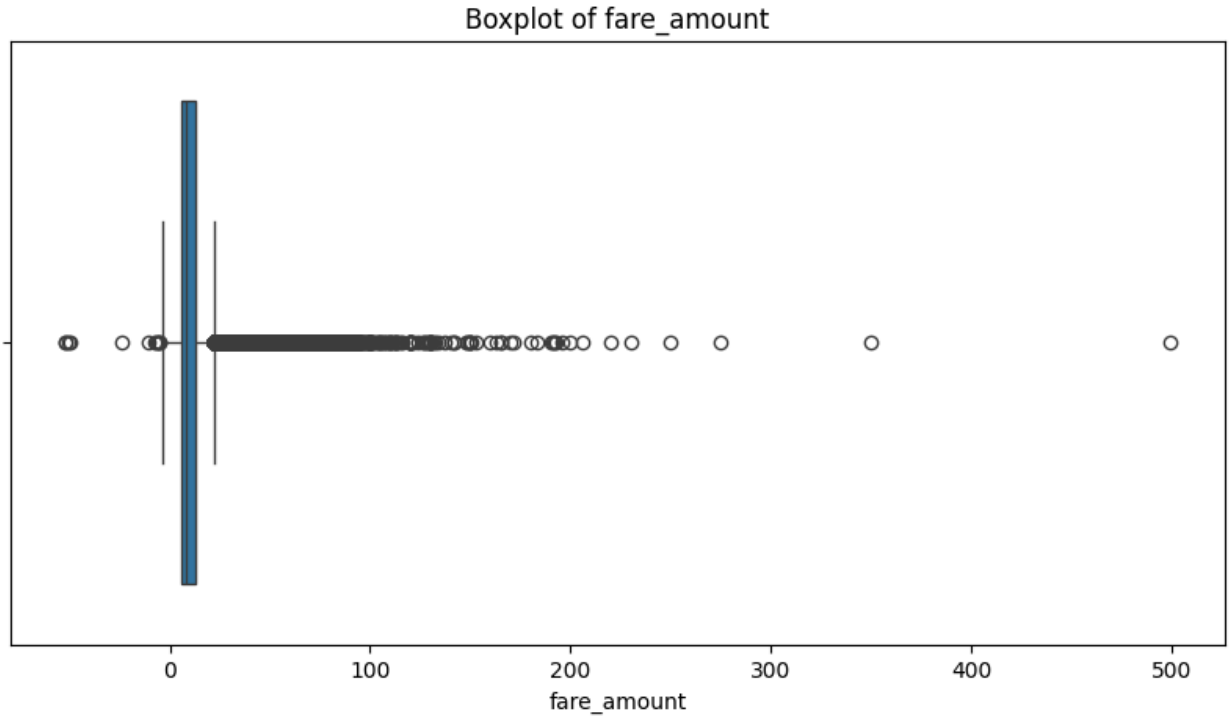
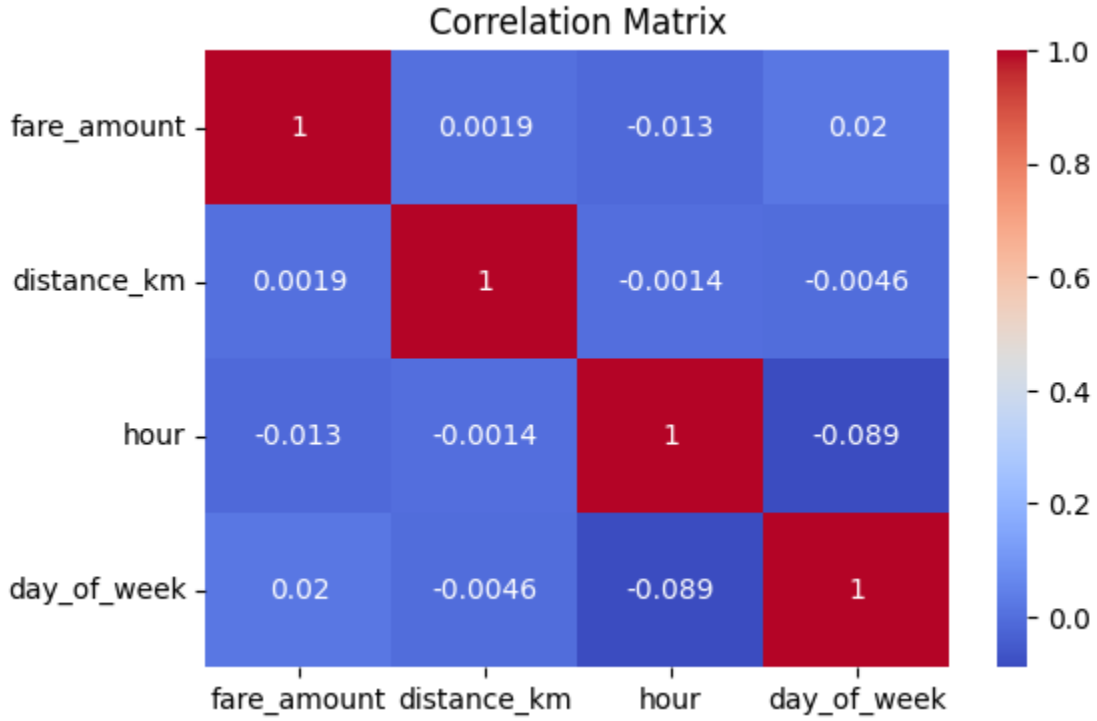
In [6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from geopy.distance import geodesic

# 1. Load and preprocess the dataset
df = pd.read_csv('archive.csv')

# Debug: check columns to identify target column
print("Columns in dataset:", df.columns.tolist())

# Replace 'fare_amount' with your actual target column name
target_col = 'fare_amount'

if target_col not in df.columns:
    raise KeyError(f"Column '{target_col}' not found in the dataset. Please ch

# Convert pickup_datetime to datetime type
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])

# Extract features: hour, day_of_week
df['hour'] = df['pickup_datetime'].dt.hour
df['day_of_week'] = df['pickup_datetime'].dt.dayofweek

# Filter out invalid latitude/longitude values before calculating distance
valid_coords = (
    df['pickup_latitude'].between(-90, 90) &
    df['dropoff_latitude'].between(-90, 90) &
    df['pickup_longitude'].between(-180, 180) &
    df['dropoff_longitude'].between(-180, 180)
)
df = df[valid_coords].copy()

# Calculate distance between pickup and dropoff points safely
def calculate_distance(row):
    try:
        pickup = (row['pickup_latitude'], row['pickup_longitude'])
        dropoff = (row['dropoff_latitude'], row['dropoff_longitude'])
        return geodesic(pickup, dropoff).km
    except ValueError:
        return np.nan

df['distance_km'] = df.apply(calculate_distance, axis=1)

# Drop rows with missing values (including those where distance calculation fa
df = df.dropna(subset=['distance_km', target_col, 'hour', 'day_of_week'])

# 2. Identify outliers
```

```python
plt.figure(figsize=(10, 5))
sns.boxplot(x=df[target_col])
plt.title(f'Boxplot of {target_col}')
plt.show()

# Remove target outliers beyond 1.5*IQR
Q1 = df[target_col].quantile(0.25)
Q3 = df[target_col].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df = df[(df[target_col] >= lower_bound) & (df[target_col] <= upper_bound)]

# 3. Check correlation
corr = df[[target_col, 'distance_km', 'hour', 'day_of_week']].corr()
plt.figure(figsize=(6, 4))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# 4. Prepare features and target
X = df[['distance_km', 'hour', 'day_of_week']]
y = df[target_col]

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Models
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(alpha=1.0),
    'Lasso Regression': Lasso(alpha=0.1)
}

results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    results[name] = {'RMSE': rmse, 'R2': r2}
    print(f"{name} - RMSE: {rmse:.2f}, R2: {r2:.3f}")

# 5. Compare results
results_df = pd.DataFrame(results).T
print(results_df)
```

```
Columns in dataset: ['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime', 'pi
ckup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'p
assenger_count']
```






```
Linear Regression - RMSE: 4.14, R2: 0.001
Ridge Regression - RMSE: 4.14, R2: 0.001
Lasso Regression - RMSE: 4.14, R2: -0.000
                        RMSE        R2
Linear Regression   4.140083  0.000609
Ridge Regression    4.140083  0.000609
Lasso Regression    4.141390 -0.000022
```

In [ ]: