In [86]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [88]:
```
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
from sklearn.preprocessing import StandardScaler
```

In [90]:
```
df = pd.read_csv("uber.csv")
df
```

Out[90]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	p
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	
...	...	...	...	...	...	
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	

200000 rows × 9 columns

In [92]:
```
df.shape # for checking number of rows and columns
```

Out[92]: (200000, 9)

In [94]:
```
df_copy = df.copy()   #for making copy od dataset so original data will not be loss
```

In [96]:
```
df_copy.isnull().sum() # for checking NAN values
```

Out[96]:
```
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 [98]:
```
df_copy.info() # for checking datatypes
```

```
<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
```

In [100…
```
df_copy.describe() # for checking distribution
```

Out[100…

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603

In [102…

```
df_copy.drop(columns = {"Unnamed: 0","key"},inplace = True) #for droppping irrevan
```

In [104…

```
df_copy.head()
```

Out[104…

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	

In [106…

```
df_copy['pickup_datetime'] = pd.to_datetime(df_copy['pickup_datetime']) #  for extr
```

In [107…

```
df_copy['year'] = df_copy['pickup_datetime'].dt.year # extracting year
df_copy['month'] = df_copy['pickup_datetime'].dt.month # extracting month
df_copy['day'] = df_copy['pickup_datetime'].dt.day # extracting day
df_copy['hour'] = df_copy['pickup_datetime'].dt.hour # extracting hour
df_copy['minute'] = df_copy['pickup_datetime'].dt.minute # extracting minute
```

In [110…

```
df_copy.head()
```

Out[110…

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	
1	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	
2	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	
3	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	
4	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	

In [112…

```
df_copy.drop(columns = {"pickup_datetime"},inplace = True)   #for dropping column p
```

In [114…

```
df_copy.head()
```

Out[114…

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pas
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.925023	40.744085	-73.973082	40.761247	

In [116…

```
# to calculate distance b/w pickup and dropoff locations we will use haversine's fo
#Haversine Formula is Used To Calculate Distane B/w lat and long of two points

import math
def haversine(lat1, lon1, lat2, lon2):
    """
    Calculate the great-circle distance between two points
    on the Earth's surface specified in decimal degrees.

    Parameters:
    lat1, lon1 : Latitude and longitude of point 1 (in decimal degrees)
    lat2, lon2 : Latitude and longitude of point 2 (in decimal degrees)

    Returns:
    Distance in kilometers between the two points.
    """

    #Radius of the earth in kilometer
    R = 6371.0

    #convert degree to the radian

    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2
    c = 2 * math.asin(math.sqrt(a))

    distance = R * c  # Distance in kilometers
    return distance
```

In [118…

```
# applying haversine's formula to calculate the distance between the latitude and l
df_copy['distance'] = df_copy.apply(lambda row: haversine(row['pickup_latitude'], r
```

In [119…

```
df_copy.head()
```

Out[119…

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pas
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.925023	40.744085	-73.973082	40.761247	

In [120…

```
df_copy = df_copy[(df_copy['fare_amount']>0) & (df_copy['distance']>0)]
```

In [124…

```
df_copy.shape
```

Out[124…

```
(194347, 12)
```

In [126…

```
df_copy.drop(columns = {"pickup_longitude","pickup_latitude","dropoff_longitude","d
```

In [128…

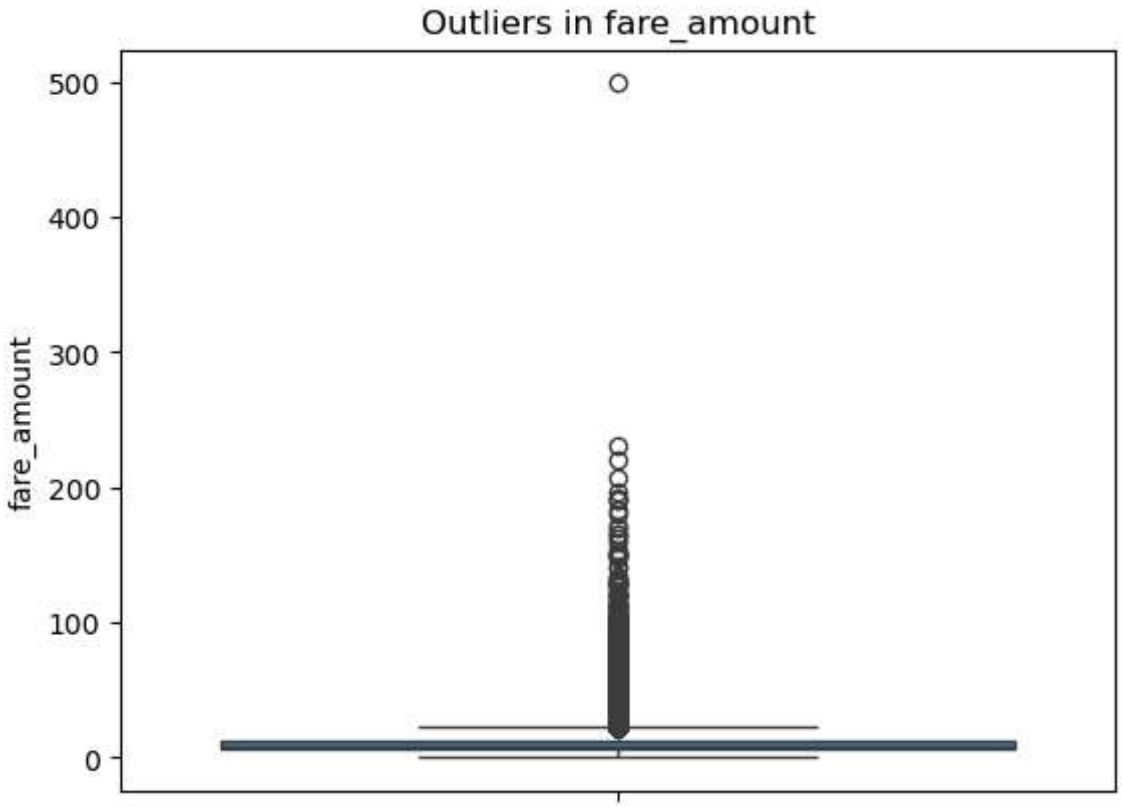
```
df_copy.head()
```

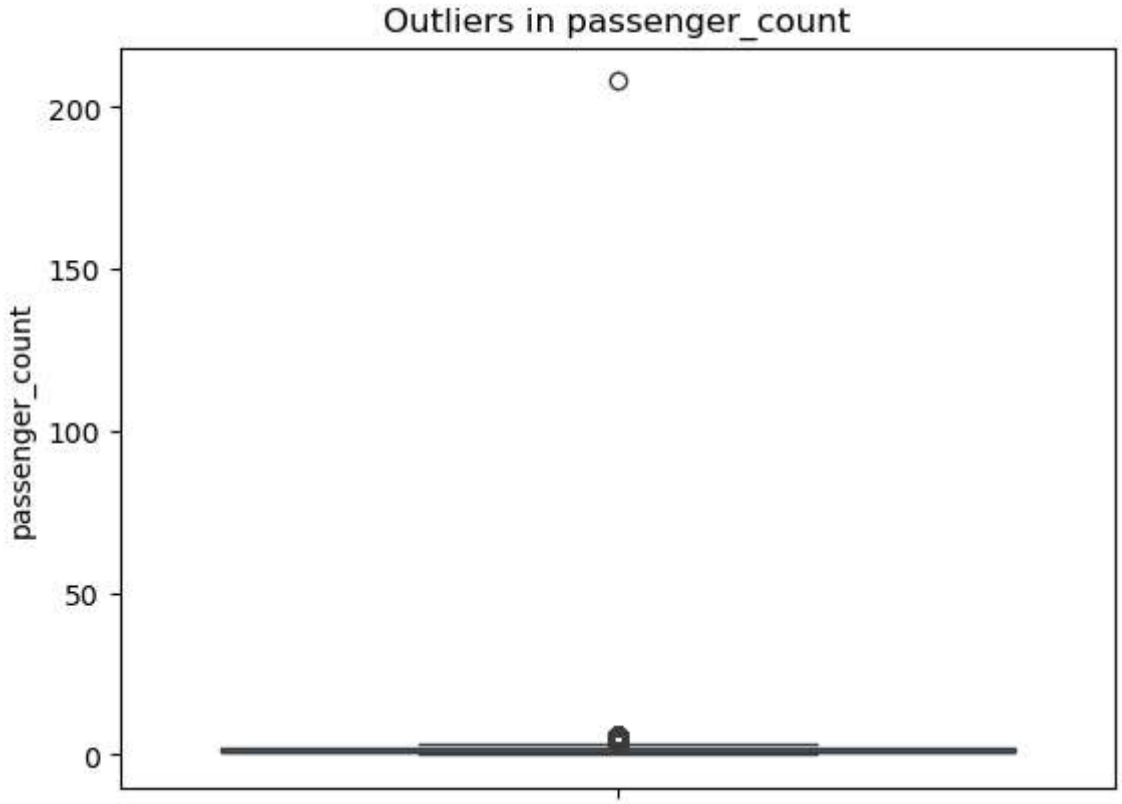
Out[128…

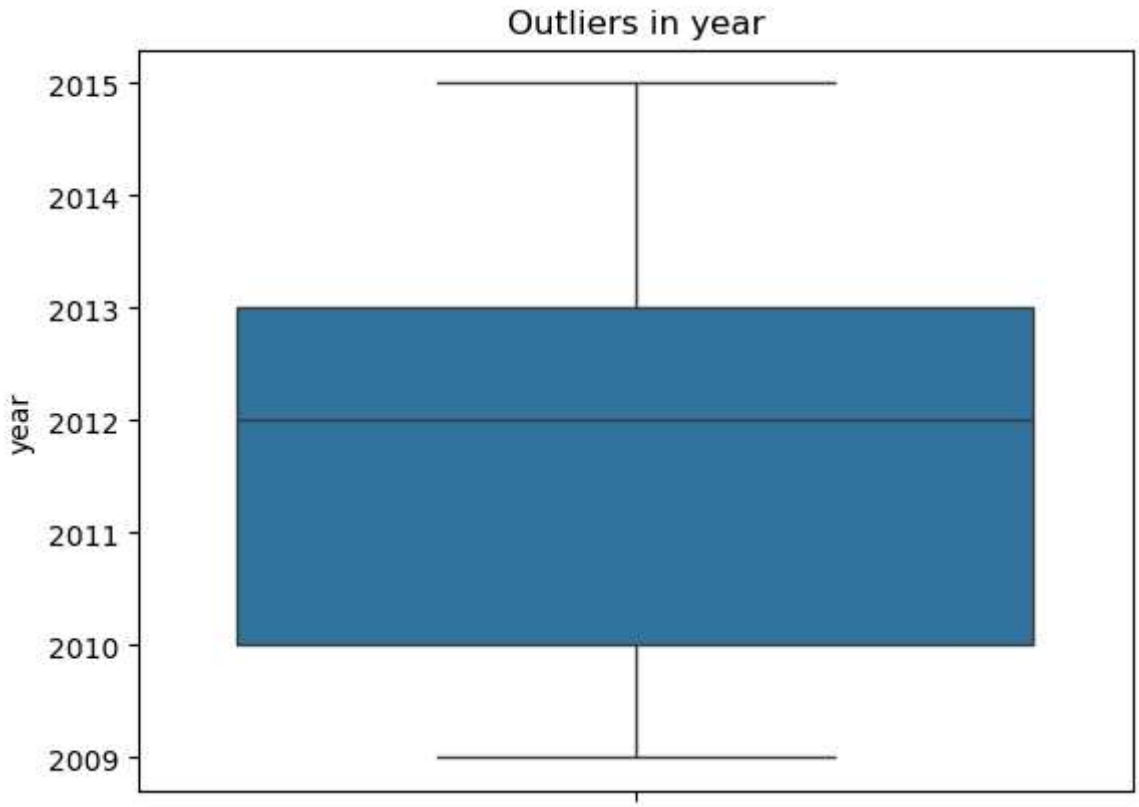
	fare_amount	passenger_count	year	month	day	hour	minute	distance
0	7.5	1	2015	5	7	19	52	1.683323
1	7.7	1	2009	7	17	20	4	2.457590
2	12.9	1	2009	8	24	21	45	5.036377
3	5.3	3	2009	6	26	8	22	1.661683
4	16.0	5	2014	8	28	17	47	4.475450

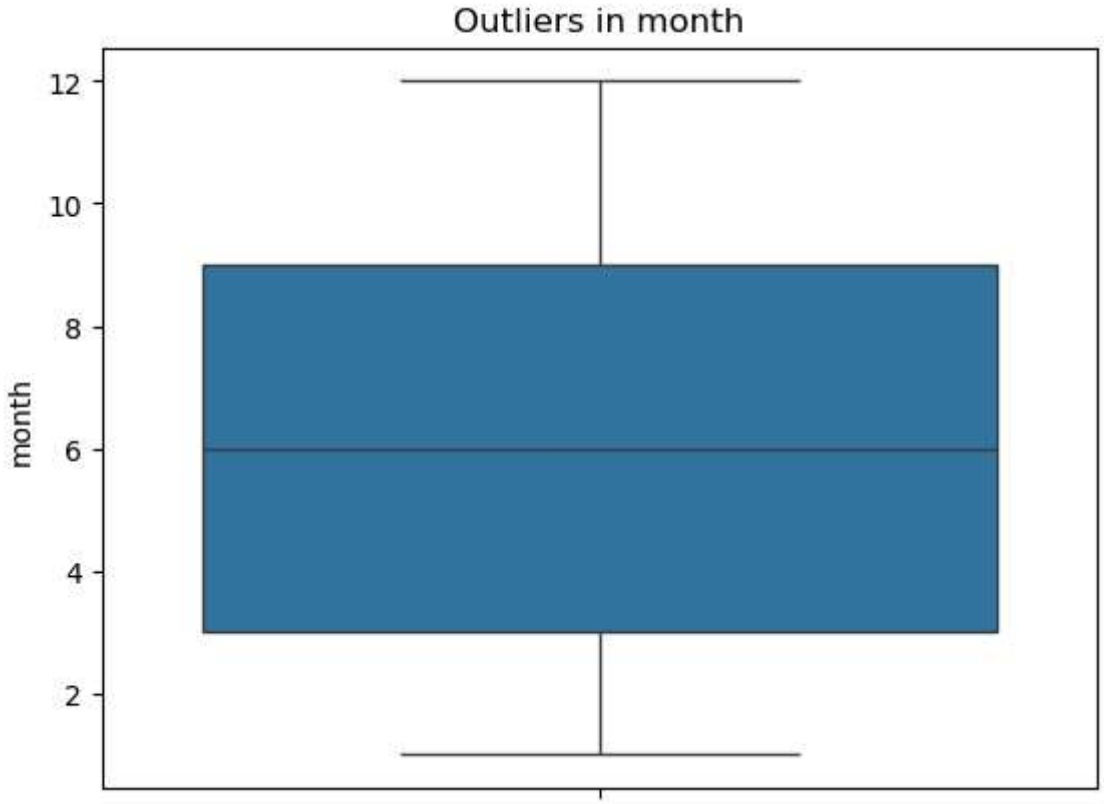
In [130…

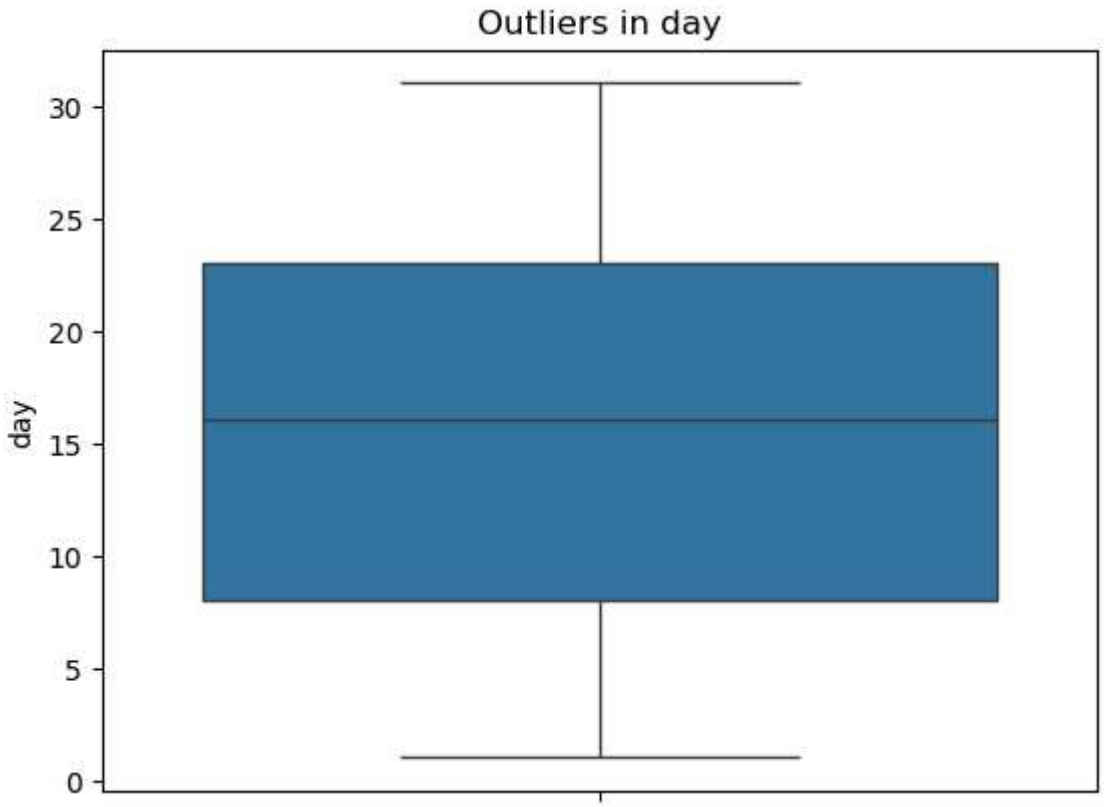
```
for column in df_copy.columns:
    sns.boxplot(df_copy[column])
    plt.title(f"Outliers in {column}")
    plt.show()
```


Outliers in fare_amount
fare_amount
500
400
300
200
100
0

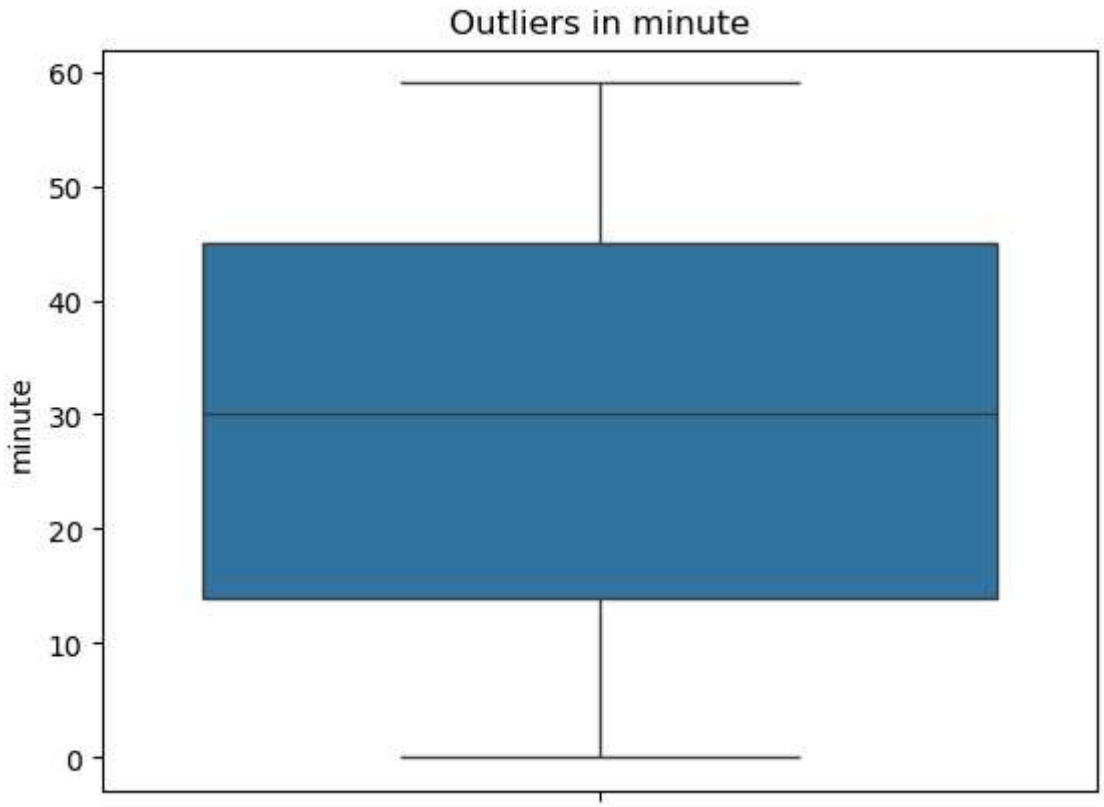

Outliers in passenger_count
passenger_count
200
150
100
50
0

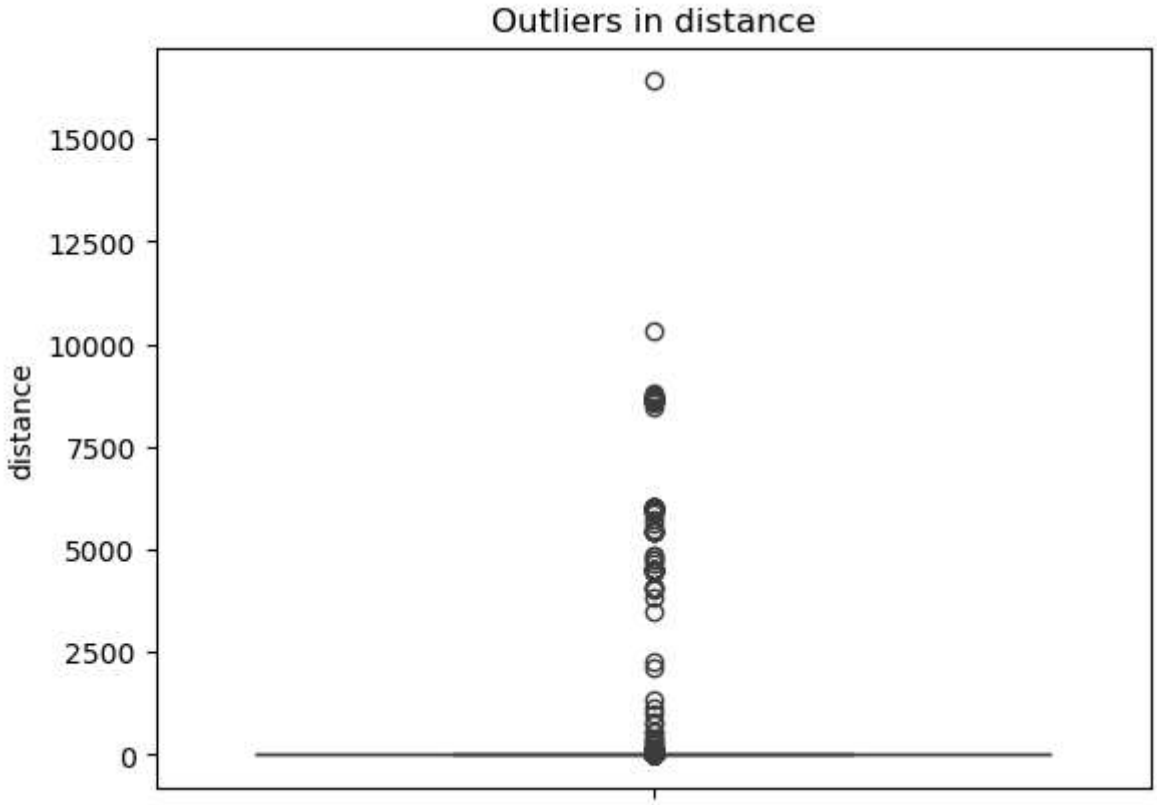

Outliers in year
year
2015
2014
2013
2012
2011
2010
2009


Outliers in month
month
12
10
8
6
4
2


Outliers in day
day
30
25
20
15
10
5
0


Outliers in hour
hour
20
15
10
5
0





In [131…

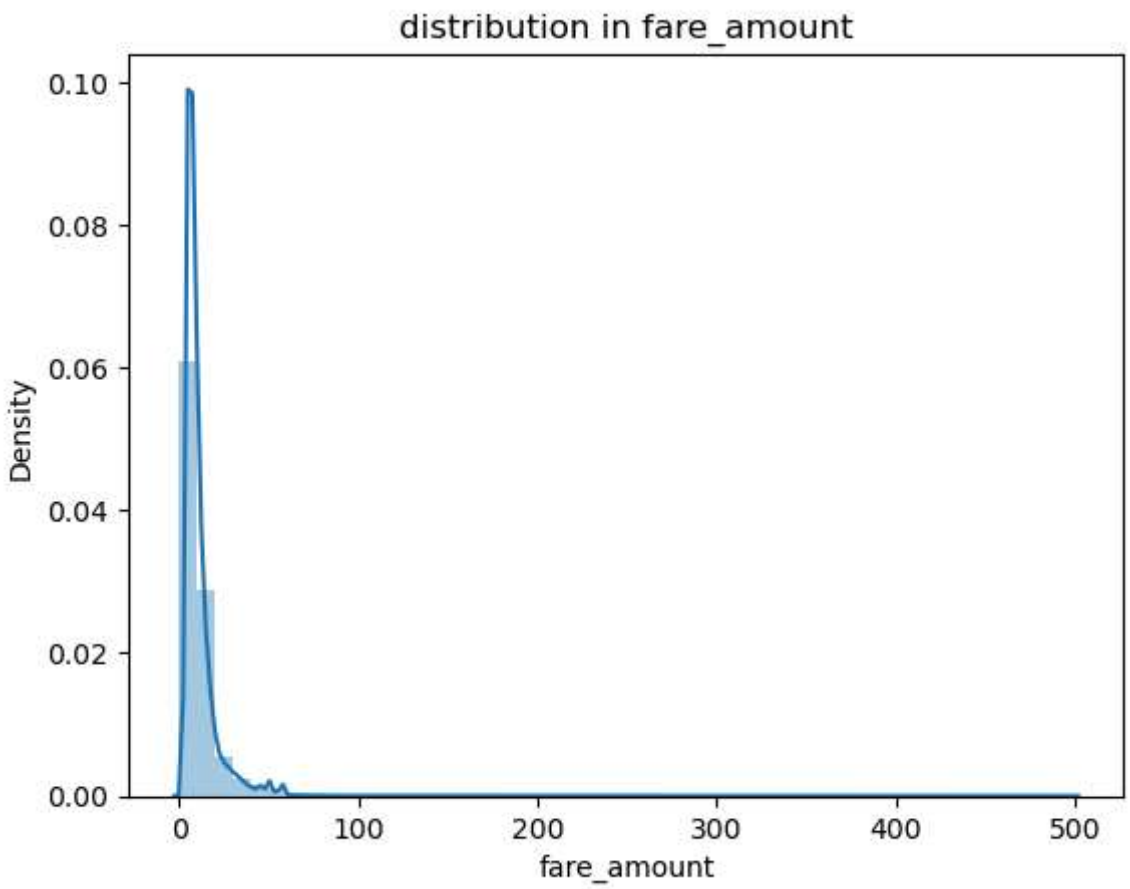
```
df_copy.head()
```

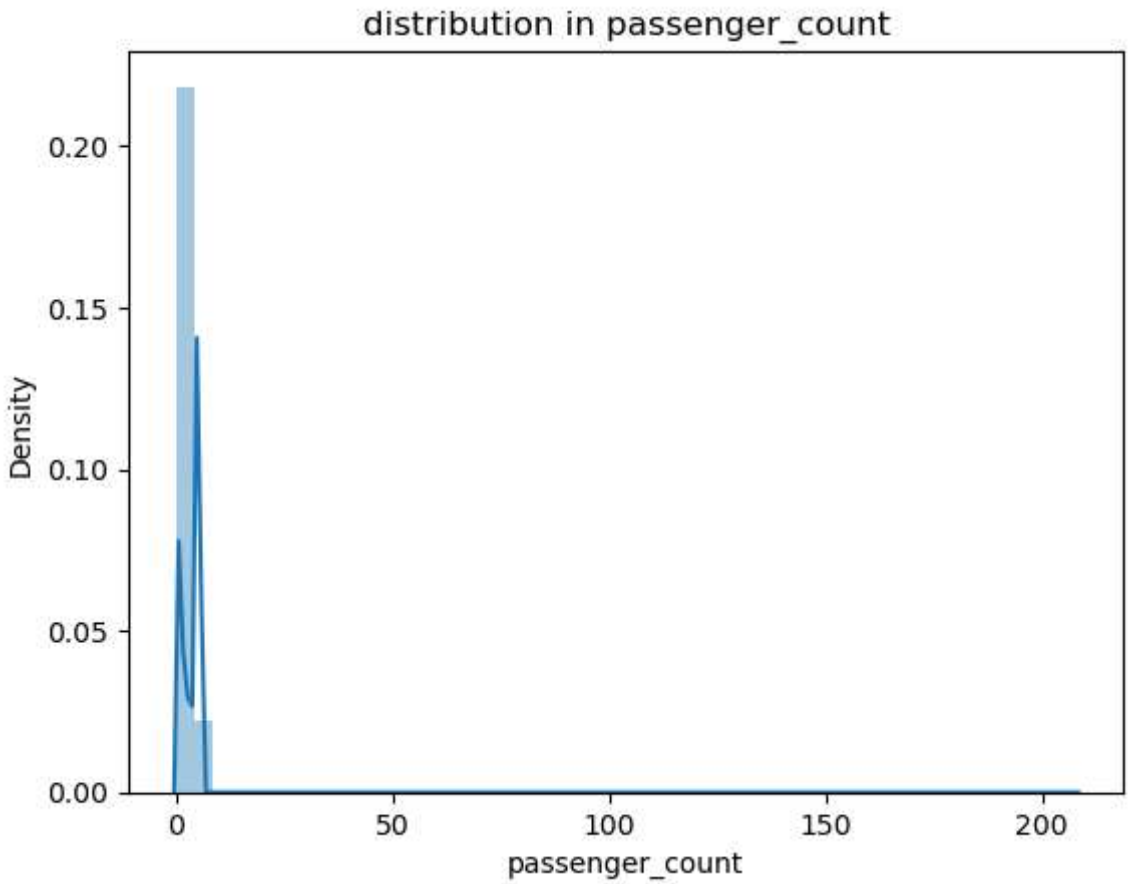
Out[131…

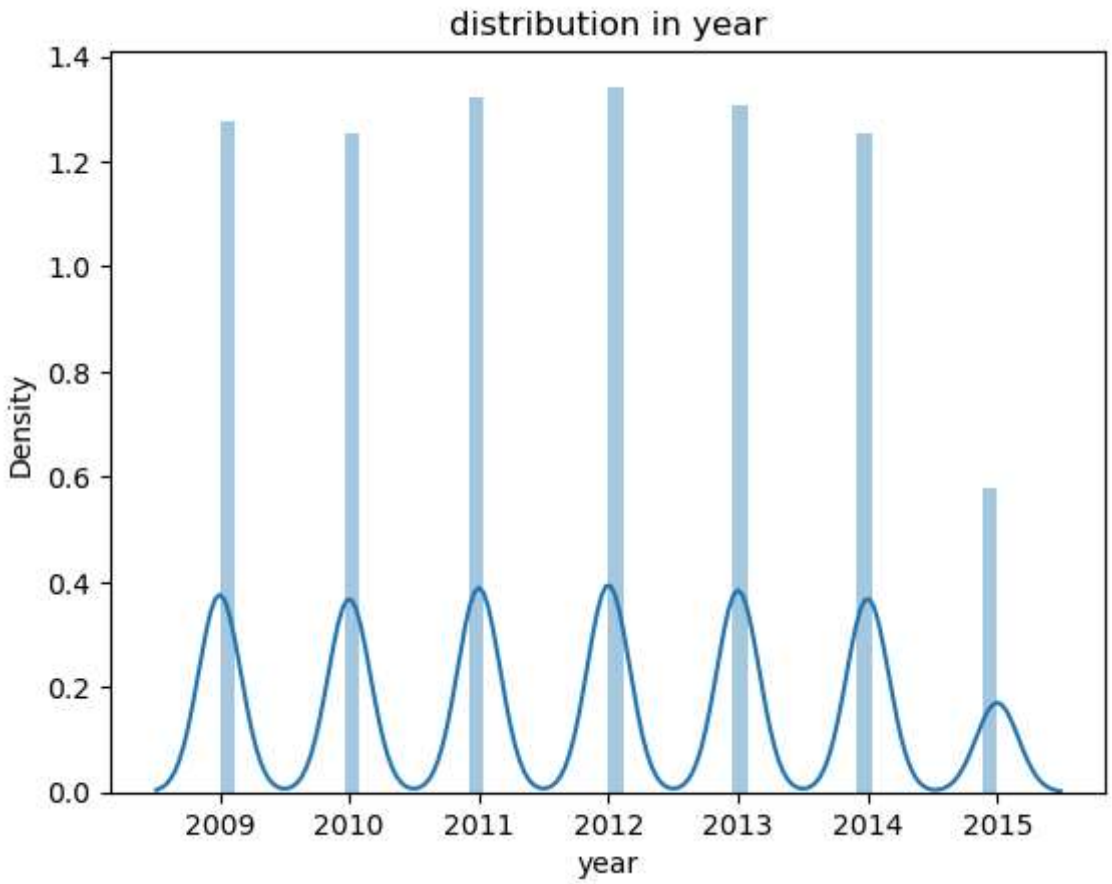
	fare_amount	passenger_count	year	month	day	hour	minute	distance
0	7.5	1	2015	5	7	19	52	1.683323
1	7.7	1	2009	7	17	20	4	2.457590
2	12.9	1	2009	8	24	21	45	5.036377
3	5.3	3	2009	6	26	8	22	1.661683
4	16.0	5	2014	8	28	17	47	4.475450

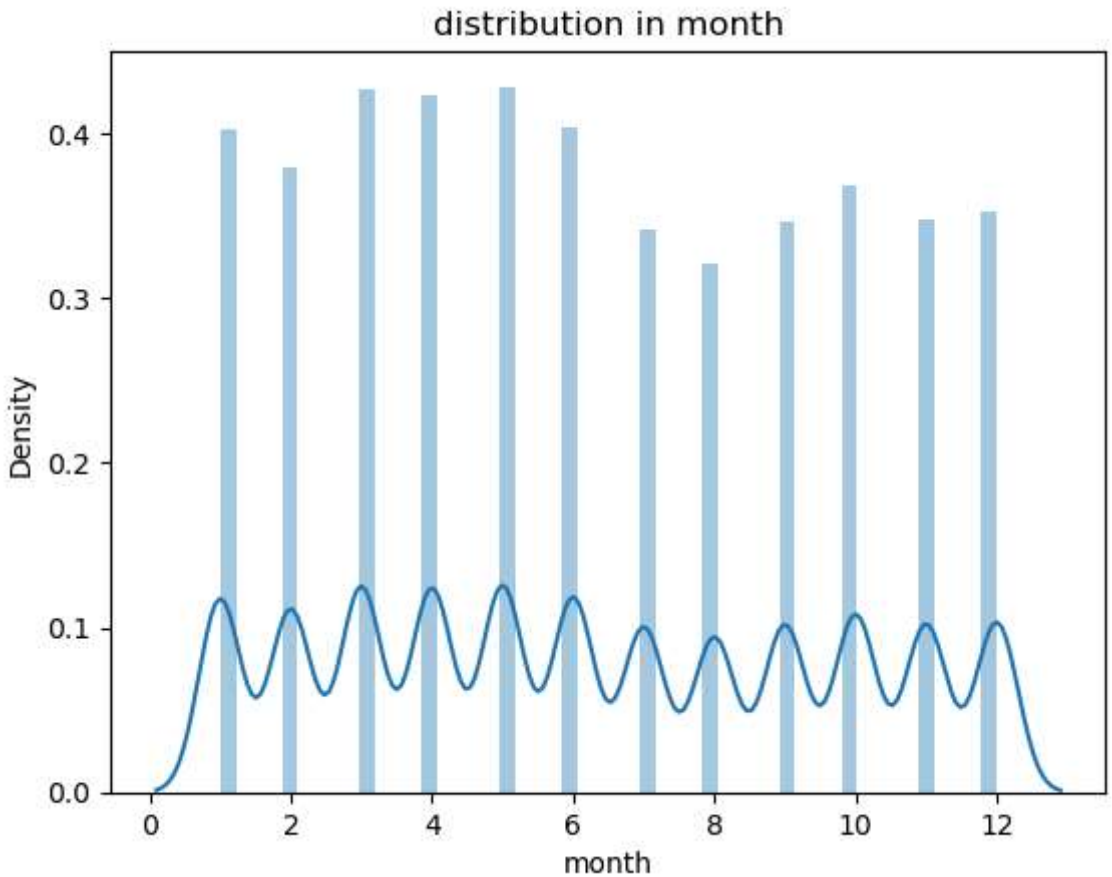
In [132…

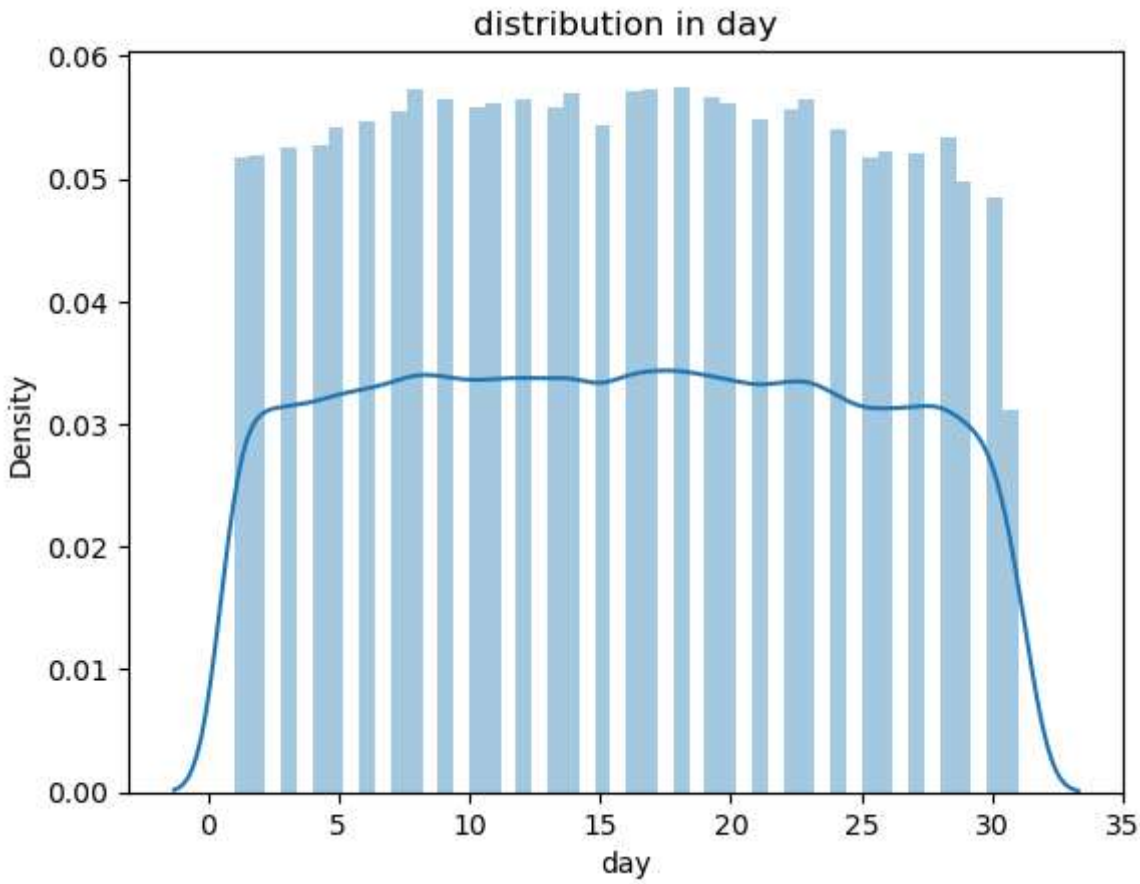
```
for column in df_copy.columns:
    sns.distplot(df_copy[column])
    plt.title(f"distribution in {column}")
    plt.show()
```

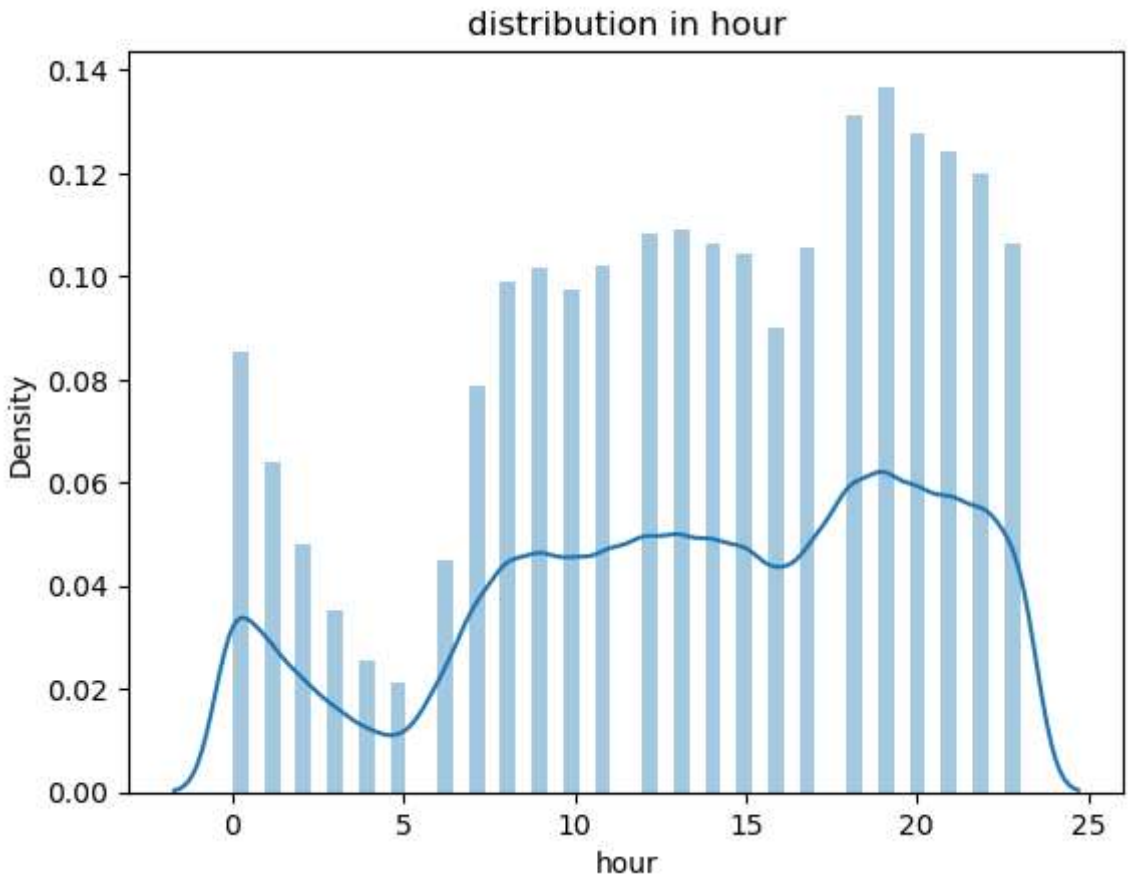


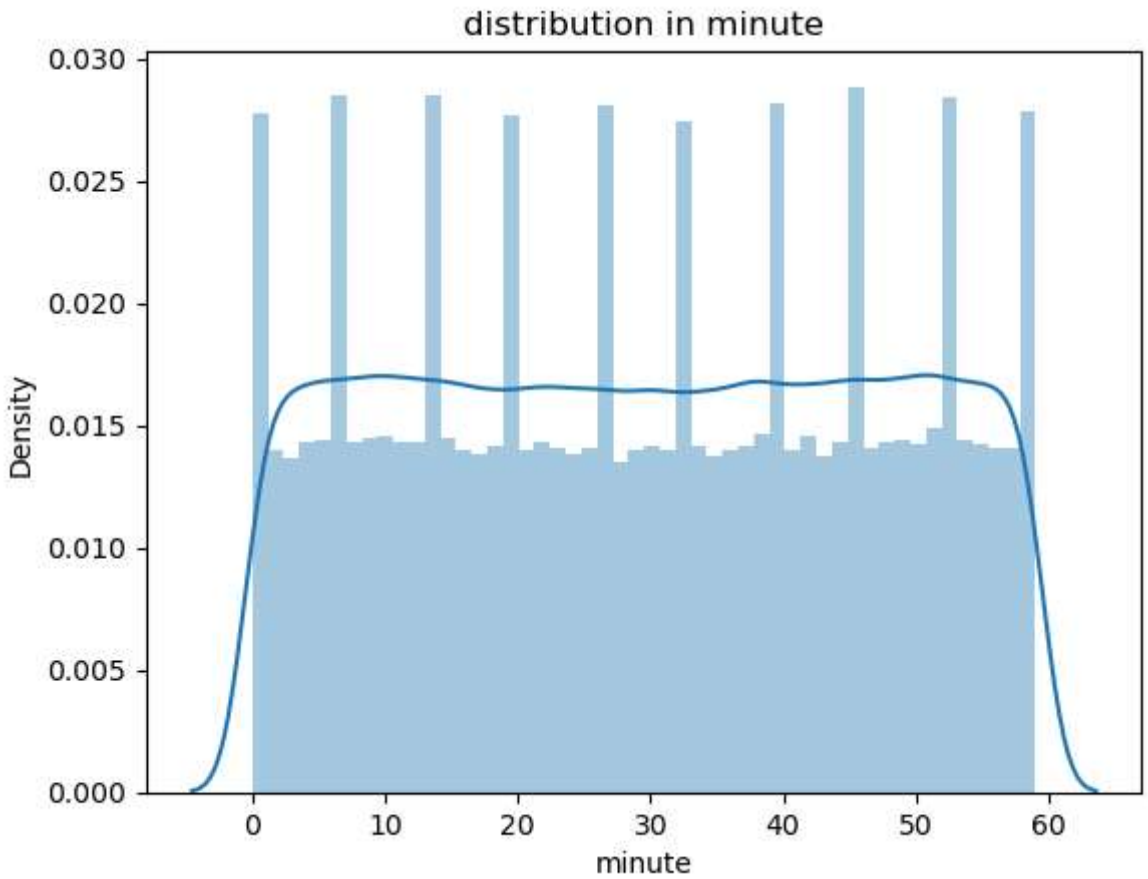

distribution in passenger_count
Density
0.20
0.15
0.10
0.05
0.00
0
50
100
150
200
passenger_count

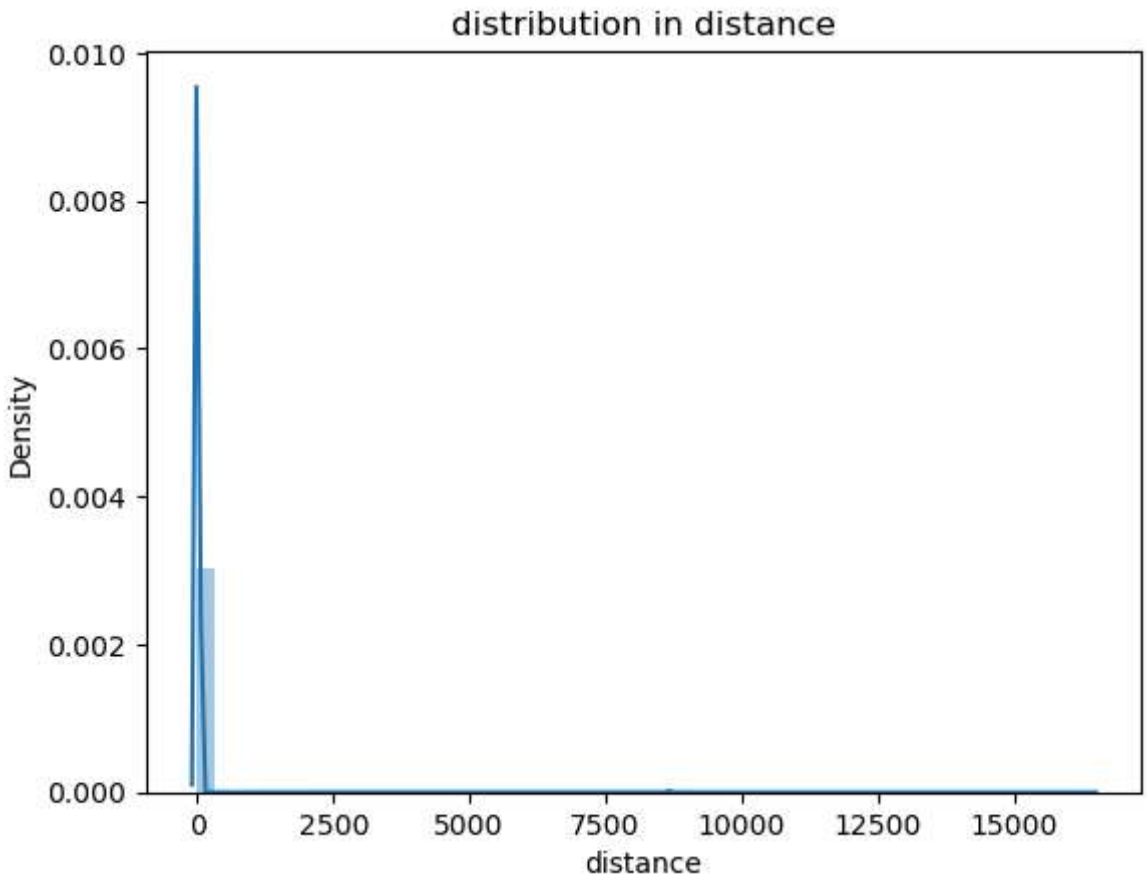

distribution in year
Density
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2009
2010
2011
2012
2013
2014
2015
year


distribution in month
Density
0.0
0.1
0.2
0.3
0.4
0
2
4
6
8
10
12
month


distribution in day
Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0
5
10
15
20
25
30
35
day


distribution in hour
Density
hour
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
25


distribution in minute
Density
minute
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
10
20
30
40
50
60



In [136…

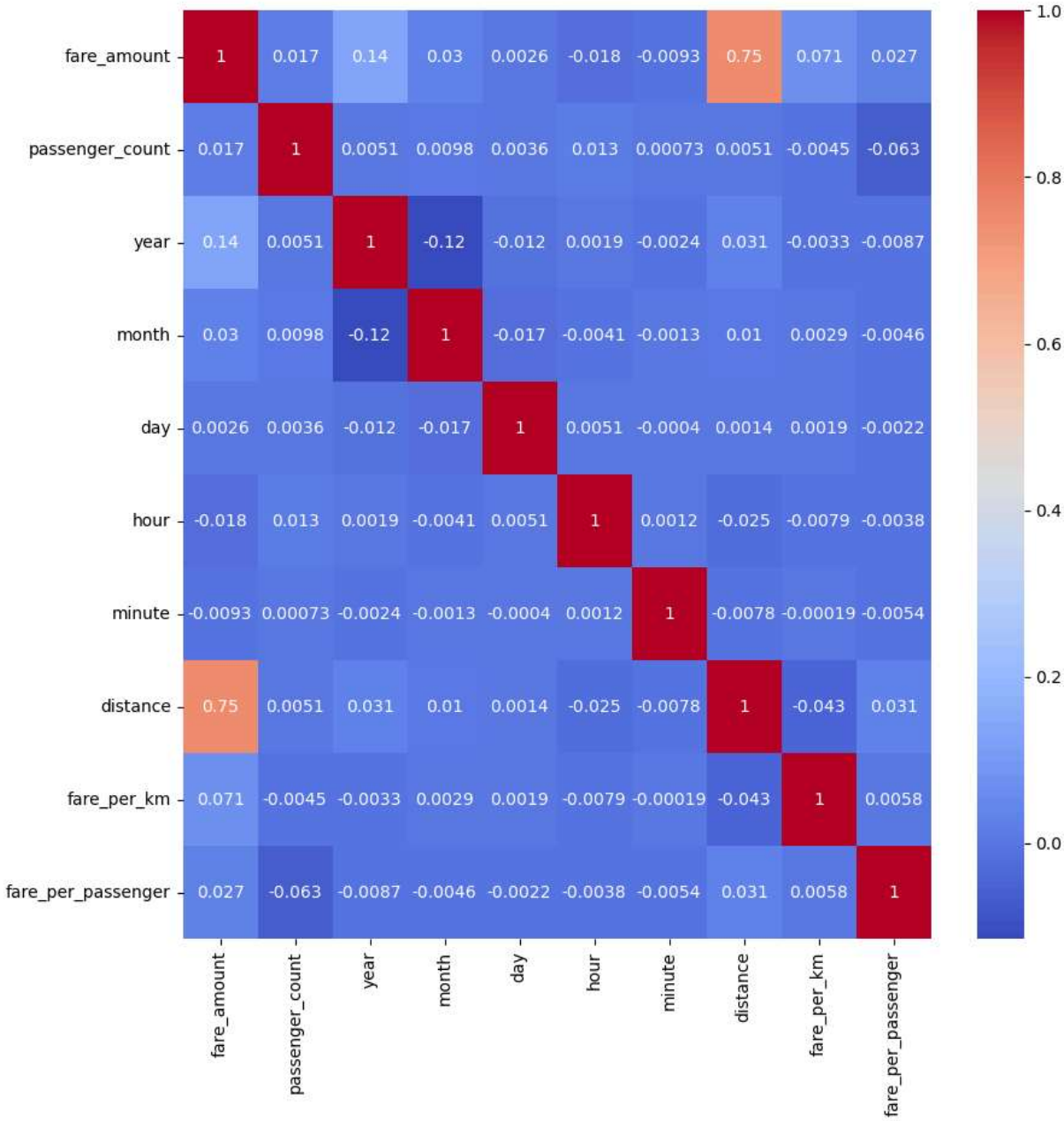
```
# creating some interaction features like fare_per_km , fare_per_passengers
# Assuming df_copy is your DataFrame
df_copy['fare_per_km'] = df_copy['fare_amount'] / (df_copy['distance'] + 1e-3)  # a
df_copy['fare_per_passenger'] = df_copy['fare_amount'] / (df_copy['passenger_count'

# Applying log transformation to the features fare_amount and distance
df_copy['fare_amount'] = np.log1p(df_copy['fare_amount'])
df_copy['distance'] = np.log1p(df_copy['distance'])
```

In [138…

```
# checking the correlation with each variable
plt.figure(figsize=(10,10))
sns.heatmap(df_copy.corr(),annot=True,cmap='coolwarm')
```

Out[138…

```
<Axes: >
```



In [140…

```
# dropping  unnecessary columns

df_copy.drop(columns = {"minute","hour","day","year","month"},inplace = True)
```

In [142…

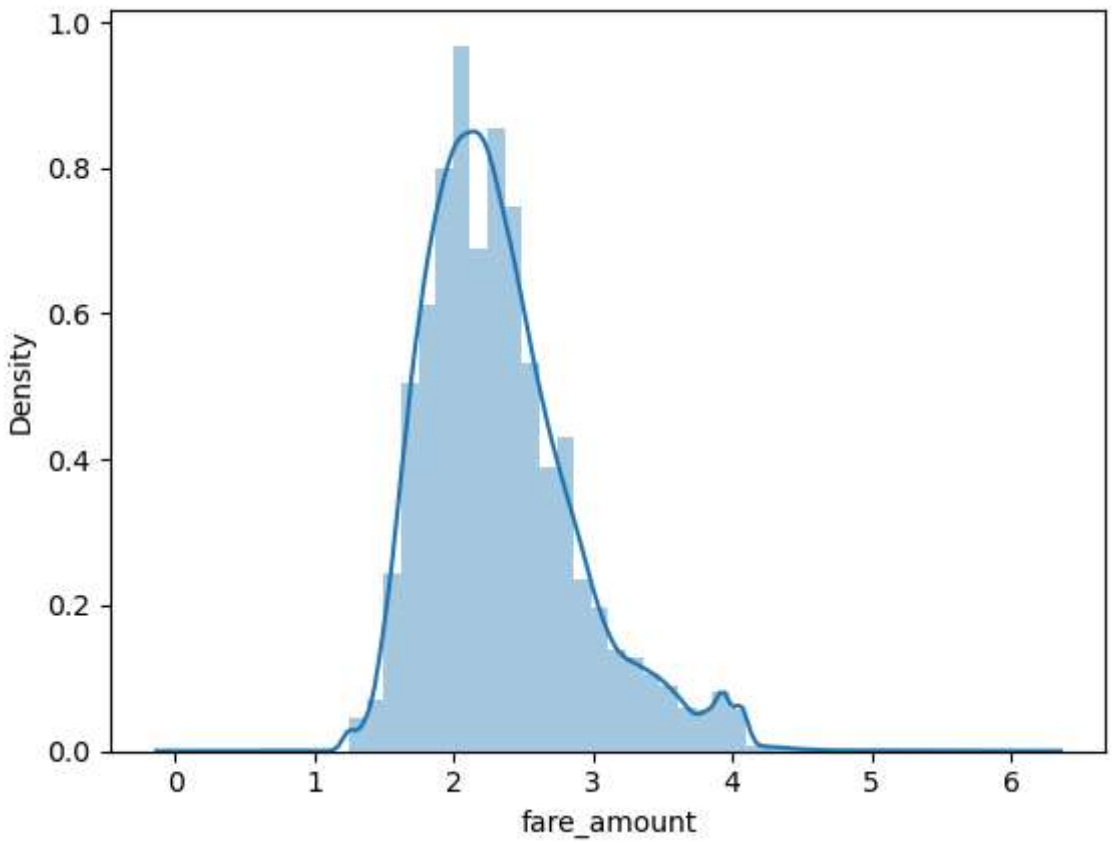
```
df_copy.head()
```

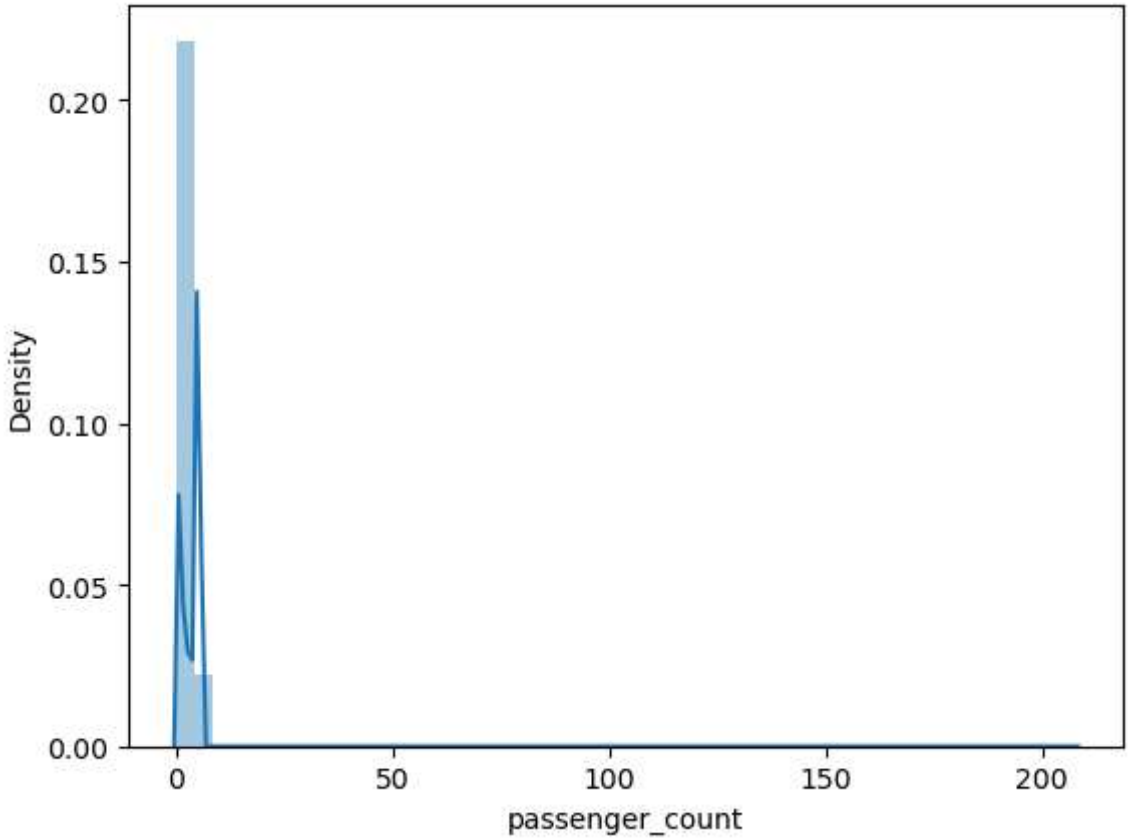
Out[142…

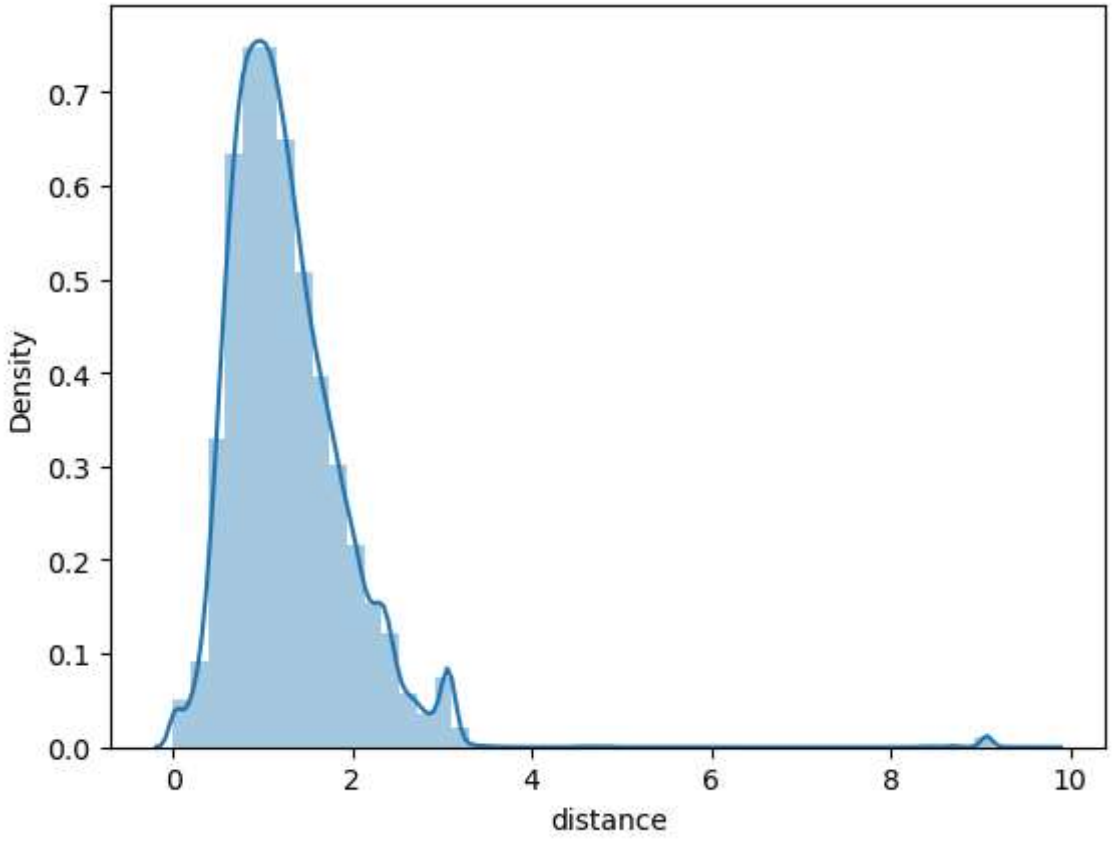
	fare_amount	passenger_count	distance	fare_per_km	fare_per_passenger
0	2.140066	1	0.987056	4.452828	7.492507
1	2.163323	1	1.240572	3.131877	7.692308
2	2.631889	1	1.797804	2.560856	12.887113
3	1.840550	3	0.978959	3.187618	1.766078
4	2.833213	5	1.700274	3.574261	3.199360

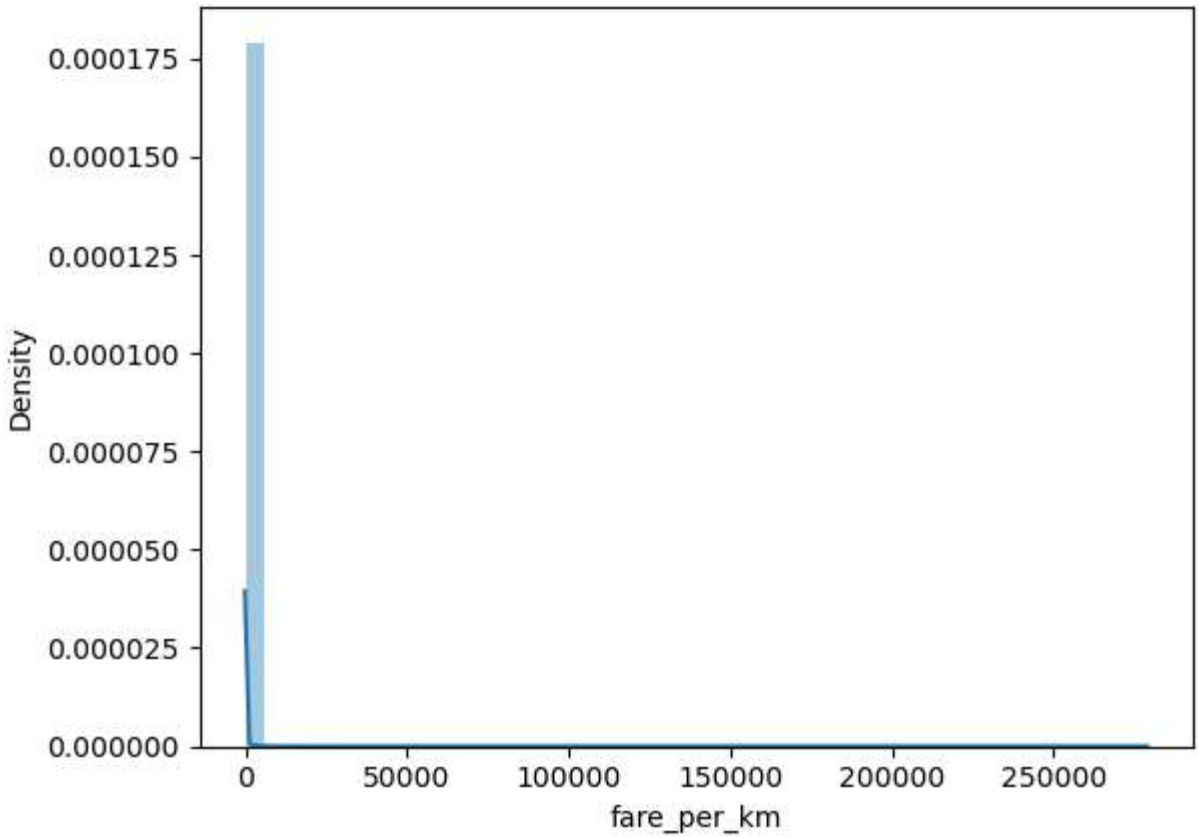
In [144…

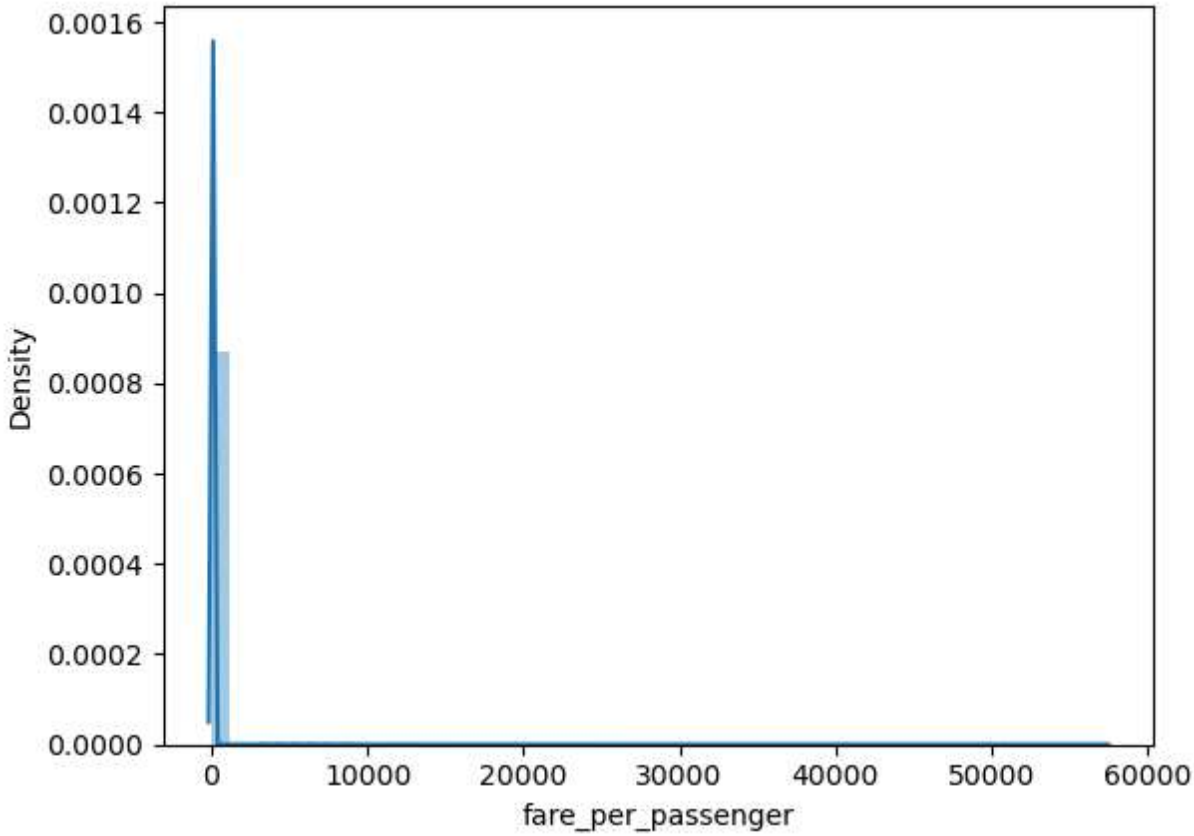
```
for columns in df_copy.columns:
  sns.distplot(df_copy[columns])
  plt.show()
```




0.20
0.15
0.10
0.05
0.00
Density
0
50
100
150
200
passenger_count


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Density
0
2
4
6
8
10
distance





In [150…

```
df_copy.skew()
```

Out[150…

```
fare_amount             0.983906
passenger_count        18.529502
distance                3.425713
fare_per_km           116.685674
fare_per_passenger     31.683159
dtype: float64
```

In [152…

```
df_copy.isnull().sum()
```

Out[152…

```
fare_amount           0
passenger_count       0
distance              0
fare_per_km           0
fare_per_passenger    0
dtype: int64
```

In [154…

```
# removing zero and inf values from the dataset
df_copy = df_copy[(df_copy['fare_amount'] > 0) & (df_copy['fare_amount'] < np.inf)]
df_copy = df_copy[(df_copy['distance'] > 0) & (df_copy['distance'] < np.inf)]
```

In [156…

```
df_copy.shape
```

Out[156…

```
(194347, 5)
```

In [158…

```
df_copy.skew()
```

Out[158…

```
fare_amount             0.983906
passenger_count        18.529502
distance                3.425713
fare_per_km           116.685674
fare_per_passenger     31.683159
dtype: float64
```

In [160…

```
# normalising fare_per_km and fare_per_passenger too
df_copy['fare_per_km'] = np.log1p(df_copy['fare_per_km'])
df_copy['fare_per_passenger'] = np.log1p(df_copy['fare_per_passenger'])
```

In [162…

```
df_copy.skew()
```

Out[162…

```
fare_amount            0.983906
passenger_count       18.529502
distance               3.425713
fare_per_km            6.472581
fare_per_passenger     2.499156
dtype: float64
```

In [196…

```
# Features and target
X = df_copy[['distance', 'passenger_count']]
y = df_copy['fare_amount']



X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [198…

```python
# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

In [199…

```python
def evaluate_model(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} Performance:")
    print(f"RMSE: {rmse:.3f}")
    print(f"R2 Score: {r2:.3f}")
    print("-" * 30)

evaluate_model(y_test, y_pred_lr, "Linear Regression")
evaluate_model(y_test, y_pred_rf, "Random Forest")
```

```
Linear Regression Performance:
RMSE: 0.369
R2 Score: 0.545
------------------------------
Random Forest Performance:
RMSE: 0.292
R2 Score: 0.714
------------------------------
```