Importing the liabraries

In [1]:
```
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.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder


import time
```

Loading the dataset

In [2]:
```
uber_df=pd.read_csv('uber.csv')
print(uber_df.head())
```

```
   Unnamed: 0       key  fare_amount          pickup_datetime  \
0    24238194  52:06.0          7.5  2015-05-07 19:52:06 UTC
1    27835199  04:56.0          7.7  2009-07-17 20:04:56 UTC
2    44984355  45:00.0         12.9  2009-08-24 21:45:00 UTC
3    25894730  22:21.0          5.3  2009-06-26 08:22:21 UTC
4    17610152  47:00.0         16.0  2014-08-28 17:47:00 UTC

   pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  \
0        -73.999817        40.738354         -73.999512         40.723217
1        -73.994355        40.728225         -73.994710         40.750325
2        -74.005043        40.740770         -73.962565         40.772647
3        -73.976124        40.790844         -73.965316         40.803349
4        -73.925023        40.744085         -73.973082         40.761247

   passenger_count
0                1
1                1
2                1
3                3
4                5
```

Info of dataset

In [3]:
```
uber_df.info()
```

```
<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
```

Summary statistics of the dataset

In [4]:
```
uber_df.describe()
```

Out[4]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.155530e+07	12.500000	-73.967153	40.767158	-73.963659	40.768001	2.000000
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

Data cleaning and processing

In [5]:
```
uber_df.dropna()
```

Out[5]:

	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	
3	25894730	22:21.0	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	
4	17610152	47:00.0	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	
...	...	...	...	...	...	...	...	...	
199995	42598914	49:00.0	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367	-73.986525	40.740297	
199996	16382965	09:00.0	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736837	-74.006672	40.739620	
199997	27804658	42:00.0	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756487	-73.858957	40.692588	
199998	20259894	56:25.0	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725452	-73.983215	40.695416	
199999	11951496	08:00.0	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720077	-73.985508	40.768793	

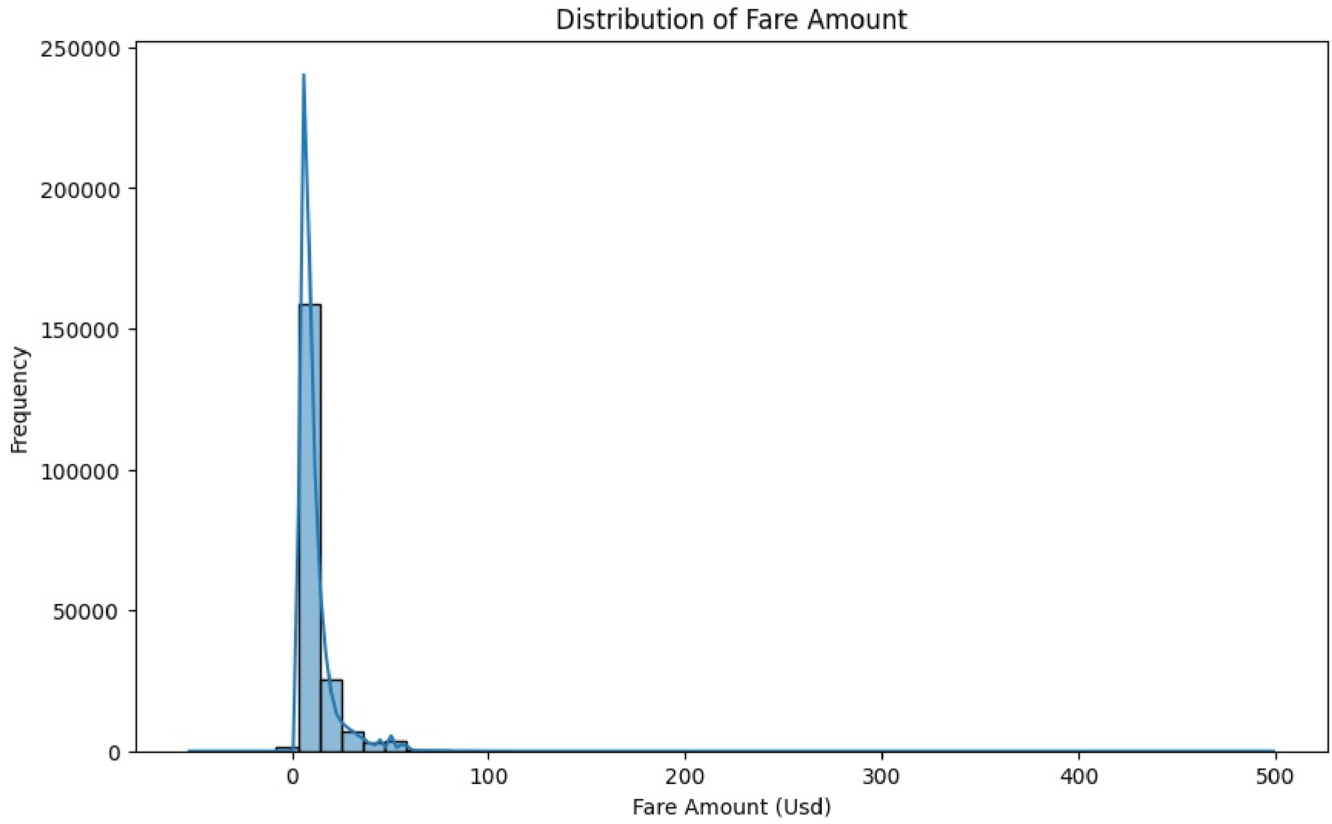
199999 rows × 9 columns

In [6]:
```
# Handling missing values
uber_df=uber_df.dropna()
```

In [7]:
```
# Convert pickup_datetime to datetime type
uber_df['pickup_datetime']=pd.to_datetime(uber_df['pickup_datetime'])
```

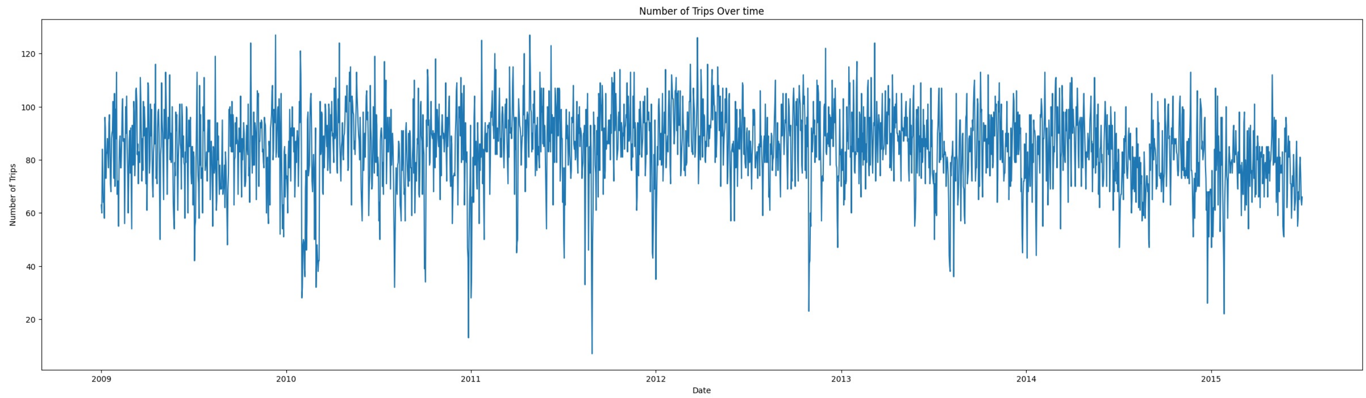
Exploratory Data Analysis (EDA)

In [8]:
```
# Plot distribution of fare_amount
plt.figure(figsize=(10,6))
sns.histplot(uber_df['fare_amount'],bins=50,kde=True)
plt.title('Distribution of Fare Amount')
plt.xlabel('Fare Amount (Usd)')
plt.ylabel('Frequency')
plt.show()
```



In [9]:
```
# Plot number of trips over time
uber_df['pickup_date'] = uber_df['pickup_datetime'].dt.date
trips_by_date = uber_df.groupby('pickup_date').size()

plt.figure(figsize=(30,8))
trips_by_date.plot()
plt.title('Number of Trips Over time')
plt.xlabel('Date')
plt.ylabel('Number of Trips')
plt.show()
```



In [10]:
```
def haversine(lat1,lon1,lat2,lon2):
    # convert latitude and longitude from degress to radians
    lat1,lon1,lat2,lon2 = map(np.radians,[lat1,lon1,lat2,lon2])

    #Haversine formula
    dlat=lat2-lat1
    dlon=lon2-lon1
    a=np.sin(dlat/2)**2+np.cos(lat1)*np.cos(lat2)*np.sin(dlon/2)**2
    c=2*np.arctan2(np.sqrt(a),np.sqrt(1-a))
    r=6371 # radius of the earth in kilometeres
    return c*r

# Create a new column for distance using the Haversine formula
uber_df['distance']=haversine(uber_df['pickup_latitude'],uber_df['pickup_longitude'],uber_df['dropoff_latitude'

# # Display the first few rows to verify the new column
uber_df.head()
```

Out[10]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
0	24238194	52:06.0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723217	
1	27835199	04:56.0	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750325	
2	44984355	45:00.0	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772647	
3	25894730	22:21.0	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803349	
4	17610152	47:00.0	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761247	

Modeling

In [11]:

```
# Select features and target variable
X = uber_df[['distance','passenger_count']]
y = uber_df['fare_amount']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Function to evaluate model
def evaluate_model(model, X_train, X_test, y_train, y_test):
    start_train = time.time()
    model.fit(X_train, y_train)
    end_train = time.time()
    train_time = end_train - start_train

    start_pred = time.time()
    y_pred = model.predict(X_test)
    end_pred = time.time()
    pred_time = end_pred - start_pred

    mse = mean_squared_error(y_test, y_pred)
    run_time = train_time + pred_time

    return run_time, mse

# Create a dictionary of models
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=1),
    'XGBRegressor': XGBRegressor(n_estimators=100, random_state=1)
}

# List to store the results
results = []

# Iterate through models, creating pipelines, and evaluating
for name, model in models.items():
    pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values
        ('scaler', StandardScaler()),  # Standardize numerical features
        ('regressor', model)  # Apply the model
    ])

    run_time, mse = evaluate_model(pipeline, X_train, X_test, y_train, y_test)
    results.append({
        'Model': name,
        'Run Time': run_time,
        'Mean Squared Error': mse
    })

# Convert results to a DataFrame
results_df = pd.DataFrame(results)

# Display results
print(results_df)
```

```
               Model   Run Time  Mean Squared Error
0  Linear Regression   0.083087           98.144912
1   Ridge Regression   0.042844           98.144912
2   Lasso Regression   0.044863           98.213329
3      Decision Tree   0.995438           41.376346
4      Random Forest  56.897443           31.916572
5       XGBRegressor   0.813348           29.196132
```

Accessing best model and training

In [12]:
```
# 1. Create and fit the model
selected_model = XGBRegressor()
selected_model.fit(X_train, y_train)

# 2. Make predictions
y_pred = selected_model.predict(X_test)

# 3. Evaluate the predictions using regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
print(f"R-squared: {r2}")
```

```
Mean Squared Error: 29.19458373364841
Mean Absolute Error: 2.5393712858970163
R-squared: 0.7027431489252659
```