**CODE 1**

```
#!/usr/bin/env python
# coding: utf-8

# In[9]:


import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import seaborn as sns
import io
df = pd.read_csv("uber.csv")


# In[10]:


df.head()


# In[11]:


df.shape


# In[12]:


df.info()


# In[4]:


#find any null value present
df.isnull().sum()


# In[5]:
```

```
#drop null rows
df.dropna(axis=0,inplace=True)
df.isnull().sum()


# In[17]:


#Calculatin the distance between the pickup and drop co-ordinates
#using the Haversine formual for accuracy.
def haversine (lon_1, lon_2, lat_1, lat_2):

    lon_1, lon_2, lat_1, lat_2 = map(np.radians, [lon_1, lon_2, lat_1, lat_2])  #Degrees to Radians


    diff_lon = lon_2 - lon_1
    diff_lat = lat_2 - lat_1


    km = 2 * 6371 * np.arcsin(np.sqrt(np.sin(diff_lat/2.0)**2 +
                                np.cos(lat_1) * np.cos(lat_2) * np.sin(diff_lon/2.0)**2))

    return km
#find distance travelled per ride
df['Distance']= haversine(df['pickup_longitude'],df['dropoff_longitude'],
                          df['pickup_latitude'],df['dropoff_latitude'])

#round it to 2 decimal points
df['Distance'] = df['Distance'].astype(float).round(2)
df.head()


# In[7]:


plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")


# In[8]:
```

```
#Outliers
#We can get rid of the trips with very large distances that are outliers
# as well as trips with 0 distance.
df.drop(df[df['Distance'] > 60].index, inplace = True)
df.drop(df[df['Distance'] == 0].index, inplace = True)
df.drop(df[df['fare_amount'] == 0].index, inplace = True)
df.drop(df[df['fare_amount'] < 0].index, inplace = True)
df.shape

# removing rows with non-plausible fare amounts and distance travelled
df.drop(df[(df['fare_amount']>100) & (df['Distance']<1)].index, inplace = True )
df.drop(df[(df['fare_amount']<100) & (df['Distance']>100)].index, inplace = True )
df.shape


# In[23]:


plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")


# In[24]:


df.info()


# In[25]:


# Create New DataFrame of Specific column
df2 = pd.DataFrame().assign(fare=df['fare_amount'], Distance=df['Distance'])
df2.info()

df2.shape


# In[28]:


# plot target fare distribution
```

```
# Use histplot instead of displot
plt.figure(figsize=[8,4])
sns.histplot(df2['fare'], color='g', edgecolor="black", linewidth=2, bins=30)
plt.title('Target Variable Distribution')
plt.show()
```

# In[29]:

```
#plots
plt.scatter(df2['Distance'], df2['fare'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

# In[30]:

```
x=df2['fare']
y=df2['Distance']

#independant variable
X = df2['Distance'].values.reshape(-1, 1)

#dependant variable
Y= df2['fare'].values.reshape(-1, 1)
```

# In[31]:

```
# scale by standardscalar
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
y_std = std.fit_transform(Y)
x_std = std.fit_transform(X)
```

# In[32]:

```
#split in test-train
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_std, y_std, test_size=0.2, random_state=0)

#simple linear regression
from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)


# In[33]:


#predict test values
y_pred = l_reg.predict(X_test)


# In[34]:


#find the error
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))


# In[35]:


#final plot
plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.title("Fare vs Distance (Training Set)")
plt.ylabel("fare_amount")
plt.xlabel("Distance")


# In[36]:


plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
```

```
plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")


# In[ ]:
```

**CODE 2**

```
def recursive_fibonacci(n):
if n <= 1:
return n
else:
return recursive_fibonacci(n - 1) + recursive_fibonacci(n - 2)
def non_recursive_fibonacci(n):
first = 0
second = 1
print(first)
print(second)
while n - 2 > 0:
third = first + second
first = second
second = third
print(third)
n -= 1
if __name__ == "__main__":
n = int(input("Enter the number of terms for the Fibonacci sequence: "))
print("\nResult for Recursive Program")
for i in range(n):
print(recursive_fibonacci(i))
print("\nResult for Non-Recursive Program")
non_recursive_fibonacci(n)
```