Software For Analytics- EDA Project

Abhishek Patil (230382226)

INTRODUCTION:

From early 2021 until mid-2022, the UK government collected and compiled statistics on COVID-19 immunization uptake across several regions. This dataset, accessible via the "UK_VaccinationsData.xlsx" file, contains information on the number of people who received their first, second, and third doses in various locations of the United Kingdom. The worksheet "variables descriptions" that goes with it has information on the data variables. The difficulty is to effectively analyze and interpret this dataset in order to gain insights regarding vaccination trends, geographical variances, and overall campaign progress. The distribution of the first, second, and third dosages over time, regional disparities, and any significant trends or abnormalities in the data are all important factors to evaluate.

Importing libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Pandas: Pandas is a Python data manipulation and analysis library. Missing values are handled efficiently by DataFrame and series structures.

Matplotlib: Matplotlib is a Python 2D plotting library. It is useful for visualizing and analyzing data.

Seaborn: It is a more effective method of data visualization. Seaborn provides us with numerous possibilities for visualizing and evaluating data.

Numpy: NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays.

Q1. Generate descriptive statistics for the dataset, and comment on the main trends.

In [2]:
```
#Read the data from excel file
df= pd.read_excel('UK_VaccinationsData.xlsx')
df.head()
```

Out[2]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose
0	England	E92000001	2022.0	5	Q2	Mon	Yes	3034.0	3857.0	8747.0
1	England	E92000001	2022.0	5	Q2	Sun	No	5331.0	3330.0	4767.0
2	England	E92000001	2022.0	5	Q2	Sat	No	13852.0	9759.0	12335.0
3	England	E92000001	2022.0	5	Q2	Fri	Yes	5818.0	5529.0	10692.0
4	England	E92000001	2022.0	5	Q2	Thu	Yes	8439.0	6968.0	11701.0

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 904 entries, 0 to 903
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   areaName    904 non-null    object 
 1   areaCode    904 non-null    object 
 2   year        903 non-null    float64
 3   month       904 non-null    int64  
 4   Quarter     903 non-null    object 
 5   day         903 non-null    object 
 6   WorkingDay  902 non-null    object 
 7   FirstDose   900 non-null    float64
 8   SecondDose  901 non-null    float64
 9   ThirdDose   898 non-null    float64
dtypes: float64(4), int64(1), object(5)
memory usage: 70.8+ KB
```

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

Out[4]:

	year	month	FirstDose	SecondDose	ThirdDose
count	903.000000	904.000000	900.000000	901.000000	898.000000
mean	2021.625692	5.946903	4994.323333	5574.125416	42529.570156
std	0.484212	4.146467	9651.335670	9174.101390	104877.579915
min	2021.000000	1.000000	0.000000	0.000000	0.000000
25%	2021.000000	2.000000	338.500000	478.000000	1313.500000
50%	2022.000000	4.000000	876.500000	971.000000	6992.000000
75%	2022.000000	11.000000	3653.250000	5770.000000	23464.750000
max	2022.000000	12.000000	115551.000000	48491.000000	830403.000000

The describe() function in pandas is used to create descriptive statistics for a DataFrame. It summarizes the central tendency, dispersion, and shape of the data distribution.

Q2. Check any records with missing values and handle the missing data as appropriate.

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
areaName      0
areaCode      0
year          1
month         0
Quarter       1
day           1
WorkingDay    2
FirstDose     4
SecondDose    3
ThirdDose     6
dtype: int64
```

isnull() function detects missing values in the array-like object. And sum() function performs summation.

Finding missing values in year.

In [9]:
```
df['year'].isnull().value_counts()
```

Out[9]:
```
year
False    903
True       1
Name: count, dtype: int64
```

In [10]:
```
df[df['year'].isnull()]
```

Out[10]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose
475	Scotland	S92000003	NaN	5	Q2	Thu	Yes	209.0	436.0	848.0

In [11]:
```
#Filling missing values
df['year'].fillna((df['year'][474] + df['year'][476]) / 2, inplace=True)
```

We followed the year from the preceding and next row to the null-value row in this case. The years were then added and split by two. This provides the missing value. Alternately, check if the years 474 and 476 are equivalent and fill in the value.

In [12]:
```
df.loc[475]
```

Out[12]:
```
areaName       Scotland
areaCode      S92000003
year             2022.0
month                 5
Quarter              Q2
day                 Thu
WorkingDay          Yes
FirstDose         209.0
SecondDose        436.0
ThirdDose         848.0
Name: 475, dtype: object
```

.loc[ ] allows the DataFrame to return specified rows and/or columns.

In [13]:
```
df['year'].isnull().value_counts()
```

Out[13]:
```
year
False    904
Name: count, dtype: int64
```

Finding missing values in Quarter

In [14]:
```
df['Quarter'].isnull().value_counts()
```

Out[14]:
```
Quarter
False    903
True       1
Name: count, dtype: int64
```

Finding the missing values in Quarter column.

In [15]:
```
df[df['Quarter'].isnull()]
```

Out[15]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose
890	Wales	W92000004	2021.0	11	NaN	Wed	Yes	720.0	762.0	15338.0

In [16]:
```
if df['Quarter'].iloc[889] == df['Quarter'].iloc[891]:
  df.loc[889:892, 'Quarter'] = df['Quarter'].iloc[891]
else:
  print('Not available')
```

We check the 'Quarter' column at two separate indices for a specified condition. If the condition is met, a range of rows in the 'Quarter' column are updated with a given value. Otherwise, a notice indicating that the value is not available is printed.

In [17]:
```
df.loc[890]
```

Out[17]:
```
areaName           Wales
areaCode       W92000004
year              2021.0
month                 11
Quarter               Q4
day                  Wed
WorkingDay           Yes
FirstDose          720.0
SecondDose         762.0
ThirdDose        15338.0
Name: 890, dtype: object
```

Finding missing values in day

In [18]:
```
df['day'].isnull().value_counts()
```

Out[18]:
```
day
False    903
True       1
Name: count, dtype: int64
```

In [22]:
```
df.dtypes['day']
```

Out[22]:
```
dtype('O')
```

In [23]:
```
df= df.drop(470)
```

In [24]:
```
df[df['day'].isnull()]
```

Out[24]:

areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose

We checked the column for missing values and kept note of them. When we look at how the workingday is organized, we see that there is no proper classification. It is preferable to remove the missing value rather than change any other critical data.However, we may address this difficulty by using groupby sorting.

Finding missing values in WorkingDay

In [25]:
```
df['WorkingDay'].isnull().value_counts()
```

Out[25]:
```
WorkingDay
False    901
True       2
Name: count, dtype: int64
```

In [26]:
```
df[df['WorkingDay'].isnull()]
```

Out[26]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose
482	Scotland	S92000003	2022.0	5	Q2	Thu	NaN	224.0	403.0	946.0
832	Wales	W92000004	2021.0	12	Q4	Fri	NaN	368.0	976.0	11845.0

In [27]:
```
is_empty = df['WorkingDay'].empty
print(is_empty)
```

```
False
```

In [28]:
```
day482= df.iloc[482]['day']
```

In [29]:
```
week1= ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']
if day482 in week1:
    df.loc[482, 'WorkingDay']= 'Yes'
else:
    df.loc[482, 'WorkingDay']= 'No'
```

It determines whether or not the value day482 exists in the list week1, which represents the weekdays Monday through Friday. If day482 is discovered in week1, the value in the 'WorkingDay' column at DataFrame (df) index 482 is set to 'Yes'. If day482 is not found in week1, the value in the 'WorkingDay' column at index 482 is set to 'No'.

In [30]:
```
day482
```

Out[30]:
```
'Wed'
```

In [31]:
```
day832= df.iloc[832]['day']
```

In [32]:
```
if day832 in week1:
    df.loc[832, 'WorkingDay']= 'Yes'
else:
    df.loc[832, 'WorkingDay']= 'No'
```

It determines whether the value day832 exists in the list or array week1. If day832 is discovered in week1, the value in the 'WorkingDay' column at DataFrame (df) index 832 is set to 'Yes'. If day832 is not found in week1, the value in the 'WorkingDay' column at index 832 is set to 'No'.

In [33]:
```
day832
```

Out[33]:
```
'Thu'
```

In [34]:
```
df[df['WorkingDay'].isnull()]
```

Out[34]:

areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose

Finding missing values in FirstDose

In [35]:
```
df['FirstDose'].isnull().value_counts()
```

Out[35]:
```
FirstDose
False    899
True       4
Name: count, dtype: int64
```

In [36]:
```
df[df['FirstDose'].isnull()]
```

Out[36]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose
479	Scotland	S92000003	2022.0	5	Q2	Sun	No	NaN	587.0	942.0
837	Wales	W92000004	2021.0	12	Q4	Sun	No	NaN	NaN	NaN
838	Wales	W92000004	2021.0	12	Q4	Sat	No	NaN	NaN	NaN
884	Wales	W92000004	2021.0	11	Q4	Tue	Yes	NaN	634.0	16022.0

In [38]:
```
dose1= df['FirstDose'].median()
df['FirstDose']= df['FirstDose'].fillna(dose1)
```

df['FirstDose'] = dose1.median() computes the median value of the DataFrame df's 'FirstDose' column and assigns it to the variable dose1.

df['FirstDose']= df['FirstDose'].fillna(dose1): This line substitutes the estimated median value (dose1) for any missing (NaN) values in the 'FirstDose' column.

In [39]:
```
df[df['FirstDose'].isnull()]
```

Out[39]:

areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose

In [40]:
```
df.loc[479]
```

Out[40]:
```
areaName        Scotland
areaCode       S92000003
year              2022.0
month                  5
Quarter               Q2
day                  Sun
WorkingDay            No
FirstDose          881.0
SecondDose         587.0
ThirdDose          942.0
Name: 479, dtype: object
```

Finding null values in SecondDose

In [41]:
```
df['SecondDose'].isnull().value_counts()
```

Out[41]:
```
SecondDose
False    900
True       3
Name: count, dtype: int64
```

In [42]:
```
df[df['SecondDose'].isnull()]
```

Out[42]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose
370	Northern Ireland	N92000002	2022.0	1	Q1	Sun	No	973.0	NaN	6867.0
837	Wales	W92000004	2021.0	12	Q4	Sun	No	881.0	NaN	NaN
838	Wales	W92000004	2021.0	12	Q4	Sat	No	881.0	NaN	NaN

In [43]:
```
#Filling missing values in SecondDose
df['SecondDose']= df['SecondDose'].fillna(df['SecondDose'].median())
```

In [44]:
```
df['SecondDose'].isnull().sum()
```

Out[44]: 0

In [45]:
```
df[df['SecondDose'].isnull()]
```

Out[45]:

areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose

In [46]:
```
df.iloc[370]
```

Out[46]:
```
areaName      Northern Ireland
areaCode             N92000002
year                    2022.0
month                        1
Quarter                     Q1
day                        Sun
WorkingDay                  No
FirstDose                973.0
SecondDose               972.0
ThirdDose               6867.0
Name: 370, dtype: object
```

Finding null values in ThirdDose

In [47]:
```
df[df['ThirdDose'].isnull()]
```

Out[47]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose
235	England	E92000001	2021.0	9	Q3	Thu	Yes	28689.0	30318.0	NaN
471	Northern Ireland	N92000002	2021.0	9	Q3	Thu	Yes	777.0	864.0	NaN
693	Scotland	S92000003	2021.0	10	Q4	Wed	Yes	5082.0	2750.0	NaN
837	Wales	W92000004	2021.0	12	Q4	Sun	No	881.0	972.0	NaN
838	Wales	W92000004	2021.0	12	Q4	Sat	No	881.0	972.0	NaN
903	Wales	W92000004	2021.0	10	Q4	Thu	Yes	1142.0	696.0	NaN

In [48]:
```
df['ThirdDose']= df['ThirdDose'].fillna(df['ThirdDose'].median())
```

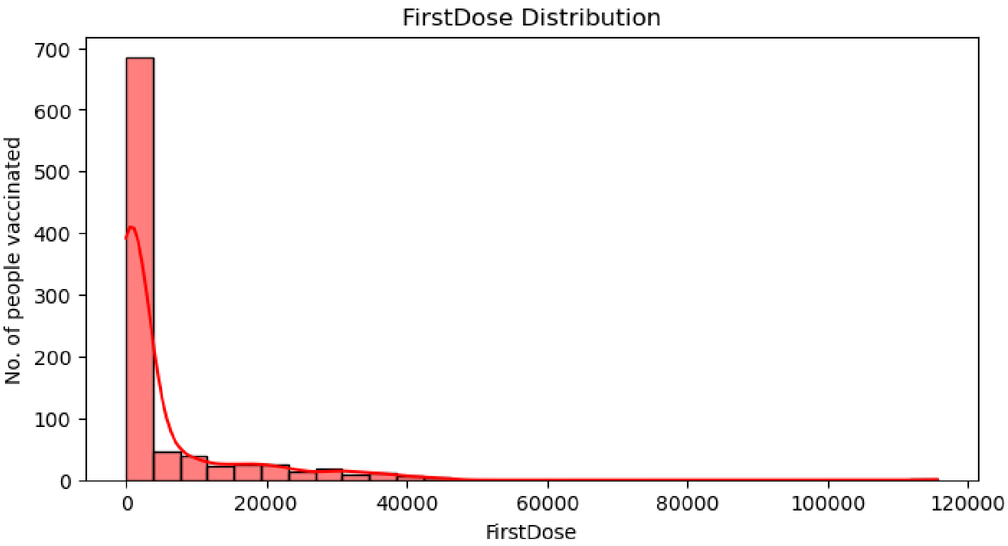
In [49]:
```
df[df['ThirdDose'].isnull()]
```

Out[49]:

areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	SecondDose	ThirdDose

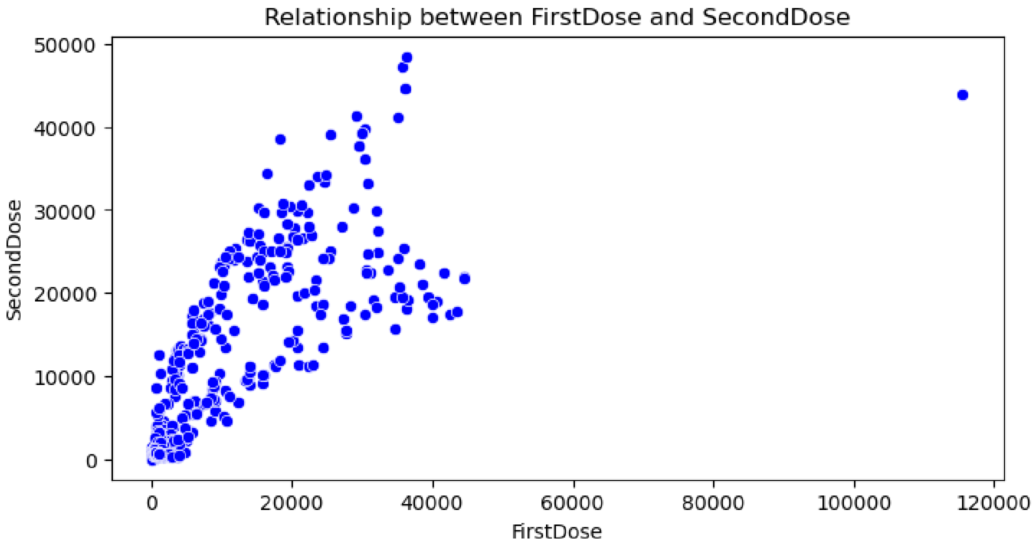
Q3. Build graphs visualizing the following and comment on the obtained visual insights

In [51]:
```
# The distribution of one or more individual continuous variables
plt.figure(figsize=(8,4))
sns.histplot(df['FirstDose'], bins=30, kde=True, color='red')
plt.xlabel('FirstDose')
plt.ylabel('No. of people vaccinated')
plt.title('FirstDose Distribution')
plt.show()
```

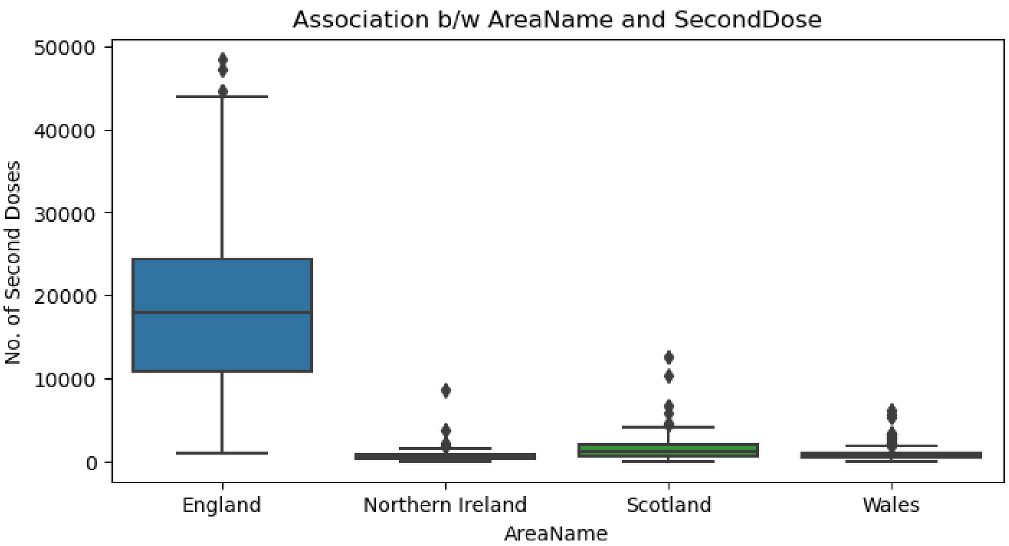


This graph shows that the number of persons receiving the first dose was substantially larger than the number of people taking it subsequently. The graph also has a large tail at the conclusion. We can presume that the majority of people preferred to get vaccinated early, as the government encouraged.

In [53]:
```
#Relationship of a pair of continuous variables
plt.figure(figsize= (8,4))
sns.scatterplot(x='FirstDose', y='SecondDose', data=df, color='blue')
plt.xlabel('FirstDose')
plt.ylabel('SecondDose')
plt.title('Relationship between FirstDose and SecondDose')
plt.show()
```



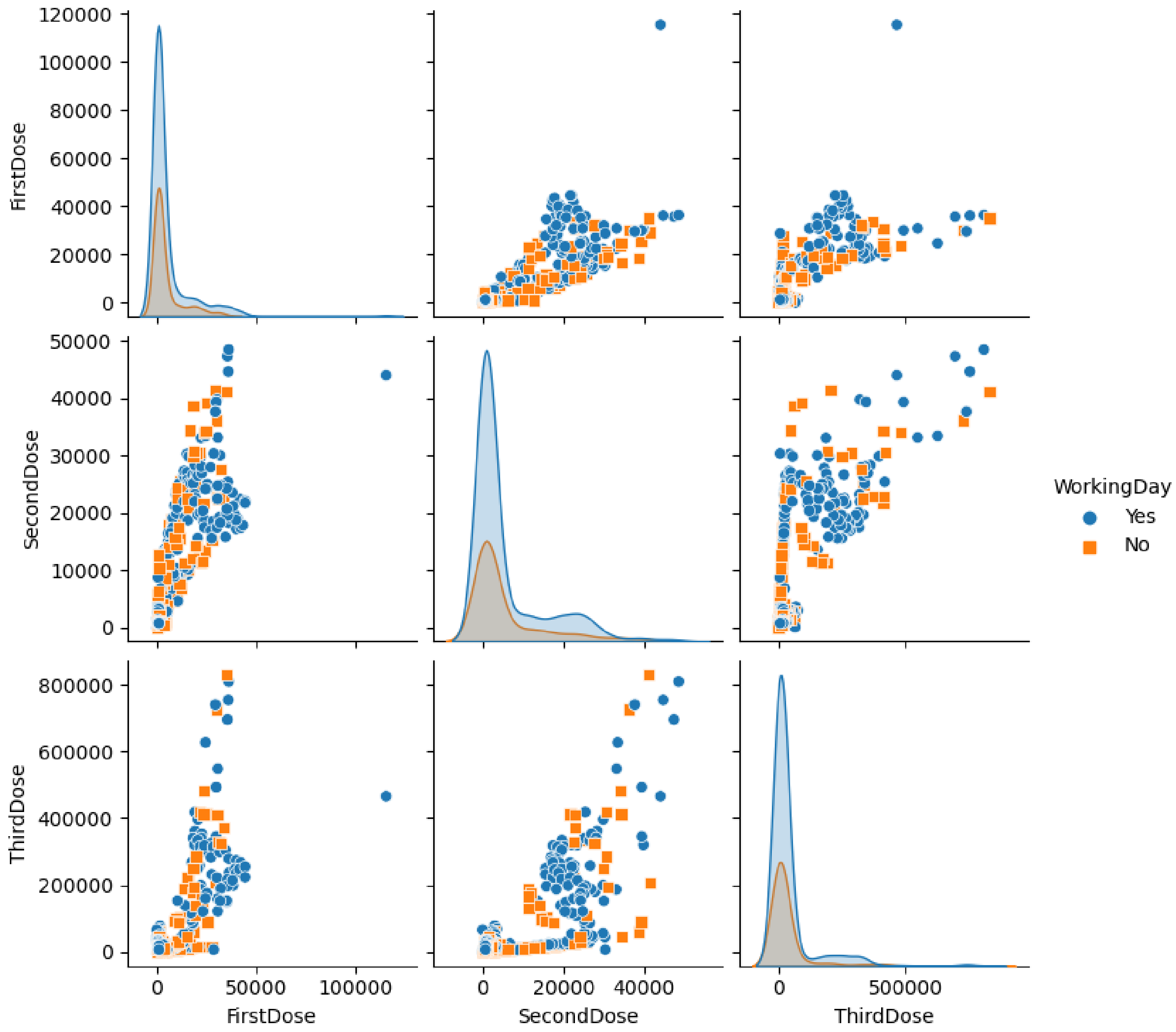
In [54]:
```
# Association between categorical variable and a continuous variable
cat_var= 'areaName'
con_var= 'SecondDose'
plt.figure(figsize=(8,4))
sns.boxplot(x='areaName', y='SecondDose', data=df)
plt.xlabel('AreaName')
plt.ylabel('No. of Second Doses')
plt.title('Association b/w AreaName and SecondDose')
plt.show()
```



The boxplot visualizes the distribution of 'SecondDose' values for distinct categories of the 'areaName' variable, providing insights into the variable's variation and central tendency across different locations.

In [56]:
```
# Relationship between more than two variables suing semantic mappings
plt.figure(figsize= (3,1))

vaccination=['FirstDose', 'SecondDose', 'ThirdDose']
sns.pairplot(df, vars= vaccination, markers=['o', 's'], hue= 'WorkingDay')
plt.show()
```

```
C:\Users\aapat\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)

<Figure size 300x100 with 0 Axes>
```



Using seaborn (sns.pairplot), we create a pair plot. A pair plot is a grid of scatterplots that shows the relationships between variables as well as the distributions of individual variables. On WorkingDay, the pairplot depicts the link between FirstDose, SecondDose, and ThirdDose. We can see a high positive correlation between the FirstDose and the SecondDose, thus we can conclude that people receive their SecondDose after obtaining the SecondDose. It was also shown that respondents favored getting their vaccinations on weekdays.

Q4. Display unique values of a categorical variable and their frequencies

In [57]:
```
area= df['areaName'].value_counts()
print('Unique values and their frequencies:')
print(area)
```

```
Unique values and their frequencies:
areaName
England             236
Northern Ireland    235
Scotland            222
Wales               210
Name: count, dtype: int64
```

There are four alternative options for the 'areaName' field: England, Northern Ireland, Scotland, and Wales. These values indicate the dataset's various regions or areas. The related frequencies show how often each individual value appears in the 'areaName' column. England has the most instances (236), followed by Northern Ireland (235), Scotland (222), and Wales (210), in that order. This data shows the geographical distribution of the dataset as well as the number of records linked with each region. Variations in frequency indicate potential variances in data representation for each area, which could be crucial for regional analysis.

Q5. Build a contingency table of two potentially related categorical variables. Conduct a statistical test of the independence between them and interpret the results.

In [58]:
```
from scipy.stats import chi2_contingency
ct= pd.crosstab(df['day'], df['WorkingDay'])
print(ct)
```

```
WorkingDay   No  Yes
day
Fri           0  129
Mon           0  129
Sat         130    0
Sun         130    0
Thu           0  130
Tue           0  127
Wed           0  128
```

In [59]:
```
chi2, p, _, _= chi2_contingency(ct)
print(f'\nChi-Square Value: {chi2}')
print(f'P-Value: {p}')
alpha= 0.05 #significance level
print('\nSignificance Level:', alpha)
print('Result:')
if p<alpha:
    print('Reject the null hypothesis. There is evidence of dependence between the variables')
else:
    print('Fail to reject the null hypothesis. There is no significant evidence dependence between the variables.'
```

```

Chi-Square Value: 903.0
P-Value: 8.438217456125006e-192

Significance Level: 0.05
Result:
Reject the null hypothesis. There is evidence of dependence between the variables
```

The null hypothesis is rejected as a consequence of the chi-square test results, which reveal a statistics value of 903 and a p-value of 8.43e-192. There is a high link between the variables "day" and "WorkingDay," implying that whether or not a day is a working day influences the choice of a day. The findings imply that the dataset's two category variables have a systematic relationship.

Q6. Retrieve one or more subset of rows based on two or more criteria and present descriptive statistics on the subset(s)

In [60]:
```
subset= df[(df['year']==2021) & (df['WorkingDay']=='Yes')]
subset.describe()
```

Out[60]:

	year	month	FirstDose	SecondDose	ThirdDose
count	243.0	243.000000	243.000000	243.000000	243.000000
mean	2021.0	11.111111	9196.921811	7744.567901	99763.238683
std	0.0	0.808018	14877.246992	11348.801578	152741.713499
min	2021.0	9.000000	0.000000	0.000000	0.000000
25%	2021.0	10.000000	781.000000	864.000000	14632.500000
50%	2021.0	11.000000	1288.000000	1401.000000	28989.000000
75%	2021.0	12.000000	19163.000000	17270.500000	153425.000000
max	2021.0	12.000000	115551.000000	48491.000000	809192.000000

According to this table, there are 243 records that meet the criteria. A monthly mean of 11.11 indicates that not all months have the same number of counts, indicating variability. The month also has a standard deviation of 0.80. In terms of vaccination doses, the standard deviations for the FirstDose, SecondDose, and ThirdDose are 14877.25, 11348.80, and 152741.71, respectively, with a mean of 9196.92, 7744.57, and 99763.24 for the FirstDose, SecondDose, and ThirdDose. Minimum values are 0 for all three doses, with reporting maximum values of 115551, 48491, and 809192.

Q7. Conduct a statistical test of the significance of the difference between the means of the subsets of the data and interpret the

results

In [61]:
```
from scipy.stats import ttest_ind
# Defining the subsets based on the criteria
subset1= df[df['day']== 'Sun']
subset2= df[df['areaName'].isin(['England', 'Wales'])]

variable_int='FirstDose'
statistic, p_value= ttest_ind(subset1[variable_int], subset2[variable_int])

#specifying the variable to compare means
print(f'T-test results for the difference in means b/w the two subsets:')

#Interpreting the results
print(f"Test Statistics: {statistic}")
print(f"P-value: {p_value}")

#setting the significance level to alpha
alpha= 0.05

print('\nSignificance level:', alpha)

print('Result:')

if p<alpha:
    print("Reject the null hypothesis. There is a significant association between 'day' and 'areaName' ")
else:
    print("Fail to reject the null hypothesis. There is no significant association between day and areaName")
```

```
T-test results for the difference in means b/w the two subsets:
Test Statistics: -5.109689774215814
P-value: 4.4000480151101114e-07

Significance level: 0.05
Result:
Reject the null hypothesis. There is a significant association between 'day' and 'areaName'
```

The t-test results show a -5.10 test statistic and a p-value of 4.40. The p-value is much lower with a significance level of 0.5. As a result, there is considerable evidence to show a meaningful relationship between the variables.

Q8. Create one or more table that group the data by a certain categorical variable and display summarized info for each group (the mean or the sum within the group)

In [62]:
```
#grouping the variable day and calculating the mean or sum within the group

gd= df.groupby('day')

#Applying multiple aggregation functions
agg_data= gd.agg({
    'FirstDose': 'mean',
    'SecondDose': 'sum',
    'ThirdDose': ['mean', 'sum']
})
print('Grouped data by Day- Multiple agg:')
print(agg_data)
```

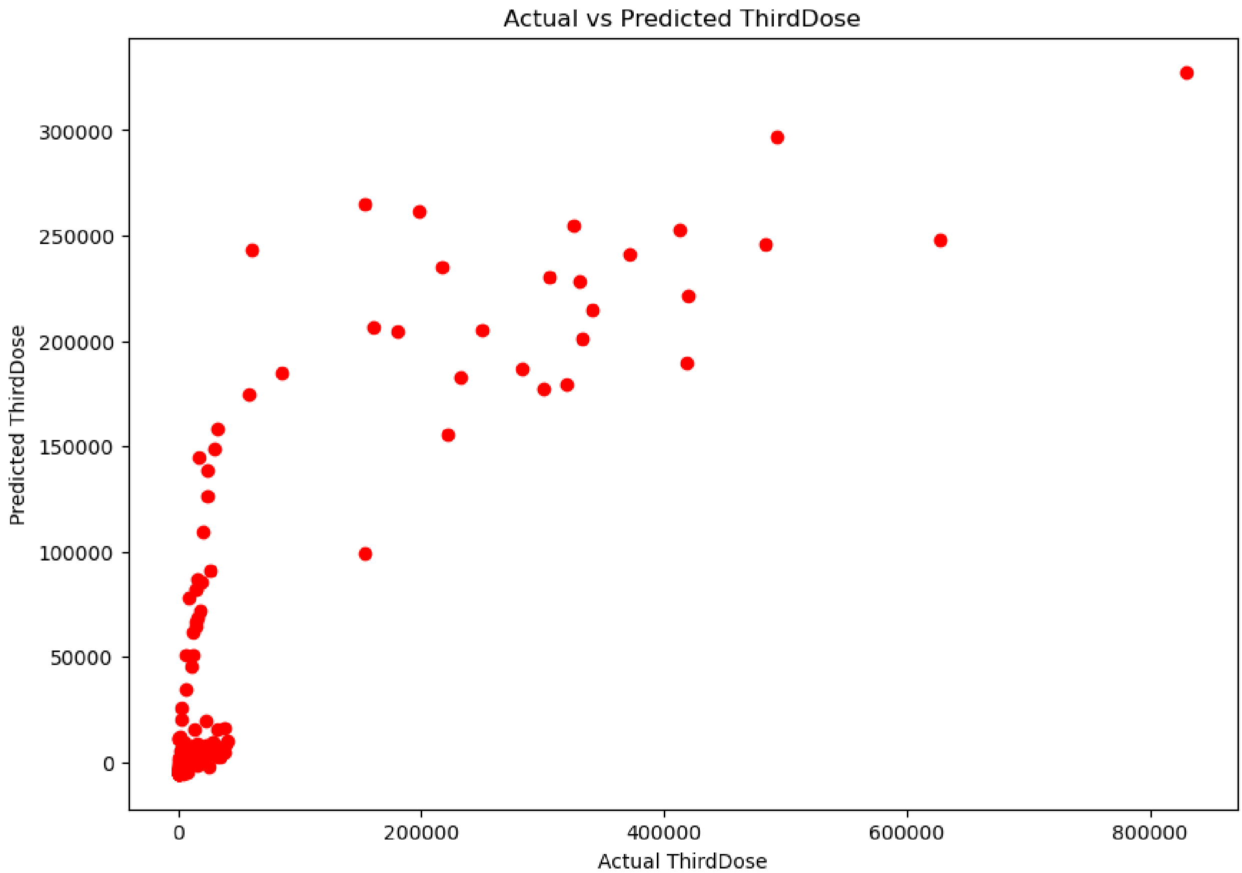
```
Grouped data by Day- Multiple agg:
       FirstDose SecondDose     ThirdDose
            mean        sum          mean        sum
day
Fri  4910.720930   714175.0  41537.007752  5358274.0
Mon  4453.248062   638122.0  37776.085271  4873115.0
Sat  5479.715385   868946.0  47177.523077  6133078.0
Sun  3548.738462   544556.0  28633.523077  3722358.0
Thu  5417.876923   775290.0  46946.492308  6103044.0
Tue  5204.393701   715340.0  45416.685039  5767919.0
Wed  5868.632812   767878.0  49024.015625  6275074.0
```

The tabular data summarizes aggregate information for vaccine doses administered on various weekdays. The mean and total data for the first, second, and third doses are provided, providing information on daily immunization trends. The highest mean is found on Wednesdays and Saturdays for initial doses, and on Saturdays and Thursdays for second and third doses. Saturdays have the highest total doses, indicating a likely trend of increased immunization activity on weekends. These findings reveal disparities in vaccine distribution throughout weekdays, providing valuable information for resource allocation and public health planning.

Q9. Implement a linear regression model and interpret its output including its accuracy.

In [66]:
```
#Implement a linear regression model and interpret its output including its accuracy

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

#Select predictor variable and target variable\
X= df[['FirstDose', 'SecondDose']]
y= df['ThirdDose']

#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)

#Create a linear regression model
model= LinearRegression()

#train the model on the training set
model.fit(X_train, y_train)

#Make predictions on the testing set
y_pred= model.predict(X_test)

#Evaluate the model
mae= metrics.mean_absolute_error(y_test, y_pred)
mse= metrics.mean_squared_error(y_test, y_pred)
rmse= metrics.mean_squared_error(y_test, y_pred, squared= False)
r2= metrics.r2_score(y_test, y_pred)

#print the model coefficients and evaluation metrics
print('Coeff:', model.coef_)
print('Intercept:', model.intercept_)
print("\nMean absolute error (MAE):", mae)
print("Mean squared error (MSE):",mse)
print("Root mean squared error (RMSE):", rmse)
print("R-squared (R2):", r2)

#Visualize the predictions
plt.figure(figsize=(10,7))
plt.scatter(y_test, y_pred, color= 'red')
plt.xlabel("Actual ThirdDose")
plt.ylabel("Predicted ThirdDose")
plt.title("Actual vs Predicted ThirdDose")
plt.show()
```

```
Coeff: [4.33674212 4.3984215 ]
Intercept: -5849.974449184425

Mean absolute error (MAE): 35389.86237786328
Mean squared error (MSE): 5280349258.250881
Root mean squared error (RMSE): 72666.01171284194
R-squared (R2): 0.6680756442153185
```



The linear regression coefficients, 4.33 and 4.39, represent the change in predicted second doses for a one-unit increase in the day of the week. The intercept, -5849.97, represents the expected second doses when all variables are zero. The model's root mean square error (RMSE) is 72666.01, and its mean squared error (MSE) is 5.2 billion. The model accounted for 66% of the variance in second dosages, with an R2 of 0.66. Given the large variability in daily dose delivery, the model, albeit having relatively high MSE and RMSE values, is only moderately accurate.

CONCLUSION:

In conclusion, the exploratory data analysis (EDA) effort has revealed important information within the COVID-19 vaccination dataset. Notable discoveries include the regional distribution of vaccine doses, the effect of weekdays on immunization rates, and significant relationships between categorical and continuous data points. These discoveries give light on trends that can be used to guide targeted interventions, resource allocation, and the development of public health policies. The initiative has produced a greater understanding of the dataset's nuanced dynamics, revealing trends that relate to the larger story of vaccination efforts in the UK. The knowledge gained emphasizes the significance of ongoing monitoring and analysis in order to properly tailor immunization campaigns. These findings will serve as the foundation for subsequent research and vaccination development.