


# Exploratory Data Analysis

## Covid-19 Vaccination Trend Analysis (United Kingdom)

### INTRODUCTION

This exploratory data analysis (EDA) research examines a large dataset that captures the COVID-19 vaccination trends in different locations of the United Kingdom. The data includes essential information such as the name of the location, code, year, month, quarter, day, working day indicators, and the number of doses provided (first, second, and third). This research, consisting of 904 records, seeks to reveal significant insights, patterns, and connections within the vaccination data. Its purpose is to enhance comprehension of the vaccination landscape in the UK, including regional variations. We will use statistical approaches, visualisations, and data exploration to analyse the development and distribution of COVID-19 vaccines in various aspects of the UK. This will involve identifying patterns, addressing missing data, and drawing relevant conclusions.

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

```
In [3]:  import pandas as pd
import seaborn as sns
import scipy.stats as stats
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import chi2_contingency
```

In [4]: `# Read data from Excel into Pandas dataframe`

```
data = pd.read_excel("UK_VaccinationsData.xlsx")
data
```

Out[4]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	Secc
0	England	E92000001	2022.0	5	Q2	Mon	Yes	3034.0	
1	England	E92000001	2022.0	5	Q2	Sun	No	5331.0	
2	England	E92000001	2022.0	5	Q2	Sat	No	13852.0	
3	England	E92000001	2022.0	5	Q2	Fri	Yes	5818.0	
4	England	E92000001	2022.0	5	Q2	Thu	Yes	8439.0	
...	...	...	...	...	...	...	...	...	...
899	Wales	W92000004	2021.0	10	Q4	Mon	Yes	3266.0	
900	Wales	W92000004	2021.0	10	Q4	Sun	No	2831.0	
901	Wales	W92000004	2021.0	10	Q4	Sat	No	3921.0	
902	Wales	W92000004	2021.0	10	Q4	Fri	Yes	1238.0	
903	Wales	W92000004	2021.0	10	Q4	Thu	Yes	1142.0	

904 rows × 10 columns



In [5]: `#Generate descriptive statistics using describe() method`

```
data.describe()
```

Out[5]:

	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 dataset uncovers significant patterns in COVID-19 vaccination around the UK. The first, second, and third vaccine doses were administered to 4,994, 5,574, and 43,529 persons on average, respectively. An evident increase in immunisations is noted in 2022, indicating the escalation of vaccination endeavours. The significant standard deviations of 9651, 9174, and 104877 suggest a broad range of outcomes, encompassing both individuals who have received only one dose and those who have completed the full immunisation series. Zero counts indicate individuals who have not gotten any doses, while different percentiles indicate a wide range of distribution, including outliers with unusually

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

```
In [6]: ▶ # Check for missing values in the entire DataFrame
missing_values = data.isnull().sum()
```

```
# Print columns with missing values
print("Columns with Missing Values:")
print(missing_values[missing_values > 0])
```

Columns with Missing Values:

```
year          1
Quarter       1
day           1
WorkingDay    2
FirstDose     4
SecondDose    3
ThirdDose     6
dtype: int64
```

```
In [7]: ▶ # Remove rows with any missing values
data_cleaned = data.dropna()
```

```
In [8]: ▶ # Check for missing values in the cleaned dataset
missing_values_cleaned = data_cleaned.isnull().sum()
print("Columns with Missing Values in Cleaned Data:")
print(missing_values_cleaned[missing_values_cleaned > 0])
```

Columns with Missing Values in Cleaned Data:  
Series([], dtype: int64)

```
In [9]: ▶ # Specify columns for categorical analysis
categorical_columns = ['Quarter', 'day', 'WorkingDay']

# Drop rows with missing values in the specified columns
data_cleaned = data_cleaned.dropna(subset=categorical_columns)
```

```
In [10]: ▶ # Check for missing values in each column and sum them up
data_cleaned.isnull().sum()
```

```
Out[10]: areaName      0
areaCode      0
year          0
month         0
Quarter       0
day           0
WorkingDay    0
FirstDose     0
SecondDose    0
ThirdDose     0
dtype: int64
```

```
In [11]: data_cleaned
```

Out[11]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	Seco
0	England	E92000001	2022.0	5	Q2	Mon	Yes	3034.0	
1	England	E92000001	2022.0	5	Q2	Sun	No	5331.0	
2	England	E92000001	2022.0	5	Q2	Sat	No	13852.0	
3	England	E92000001	2022.0	5	Q2	Fri	Yes	5818.0	
4	England	E92000001	2022.0	5	Q2	Thu	Yes	8439.0	
...	...	...	...	...	...	...	...	...	...
898	Wales	W92000004	2021.0	10	Q4	Tue	Yes	2440.0	
899	Wales	W92000004	2021.0	10	Q4	Mon	Yes	3266.0	
900	Wales	W92000004	2021.0	10	Q4	Sun	No	2831.0	
901	Wales	W92000004	2021.0	10	Q4	Sat	No	3921.0	
902	Wales	W92000004	2021.0	10	Q4	Fri	Yes	1238.0	

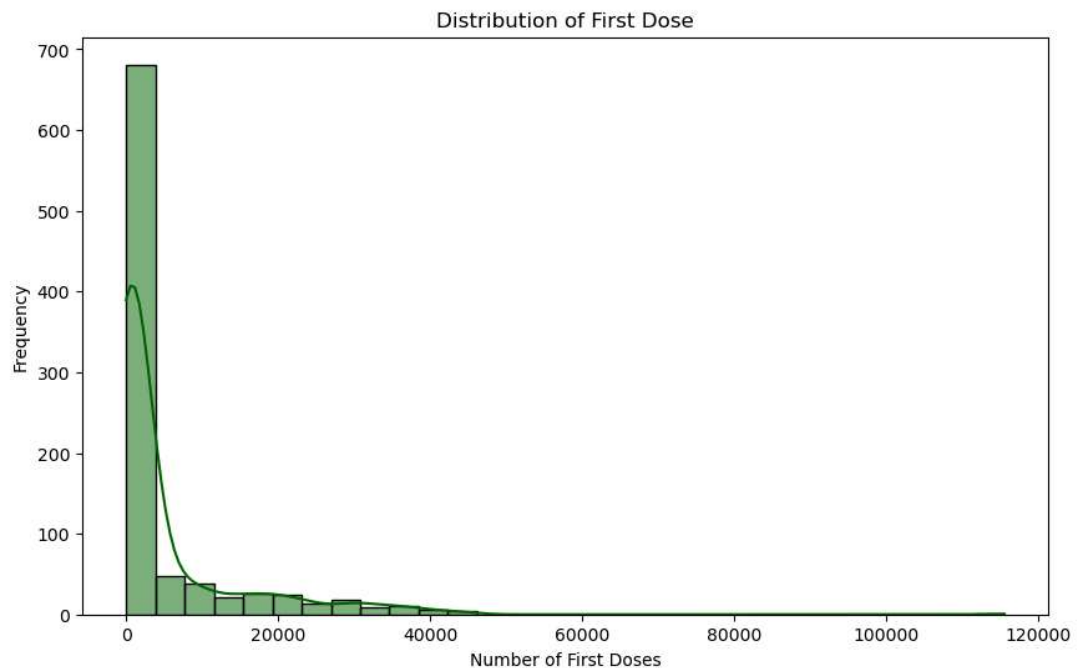
890 rows × 10 columns



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

#### 3.1 The distribution of one or more individual continuous variables

```
In [12]: ▶ plt.figure(figsize=(10, 6))
sns.histplot(data['FirstDose'], kde=True, bins=30, color='darkgreen')
plt.title('Distribution of First Dose')
plt.xlabel('Number of First Doses')
plt.ylabel('Frequency')
plt.show()
```

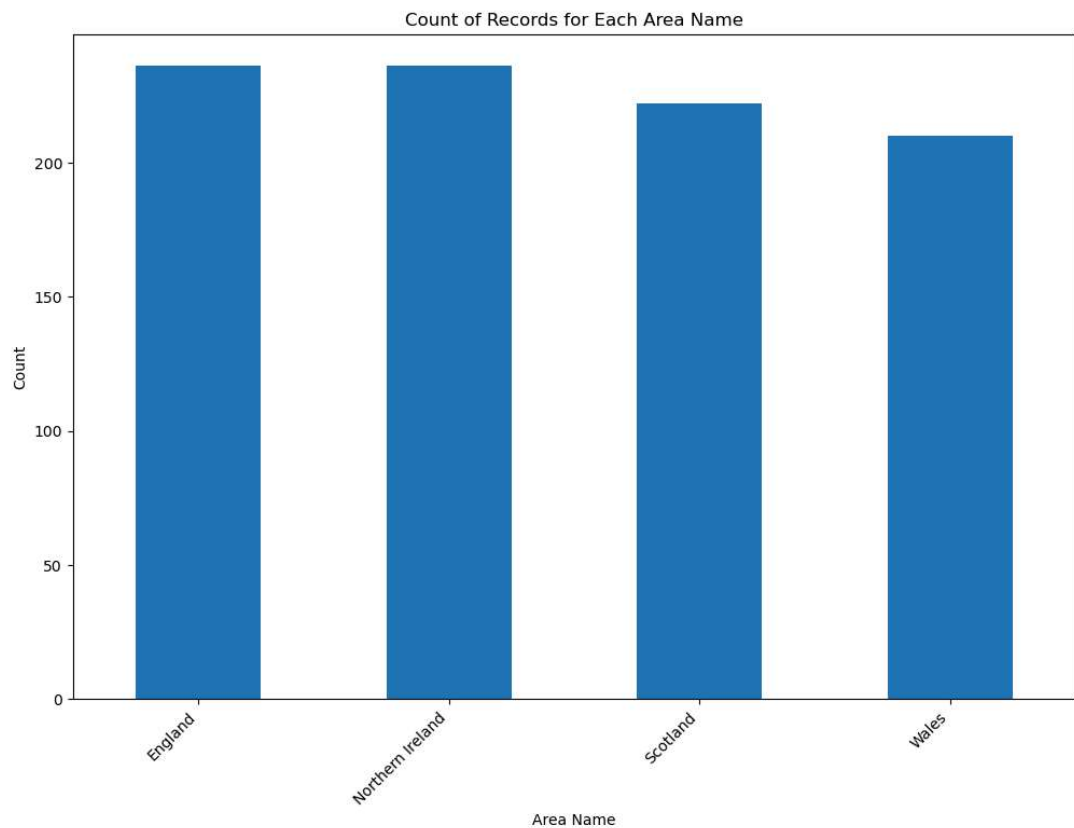


The dataset reveals COVID-19 vaccination trends in the UK, with average doses of 4994, 5574, and 43529 for first, second, and third doses. A surge in 2022 reflects intensified vaccination efforts. However, diverse standard deviations (9651, 9174, 104877) and instances of 0 doses suggest a varied landscape, highlighting substantial variability and outliers in the distribution.

```
In [13]: import matplotlib.pyplot as plt

# Get counts for each category
area_counts = data['areaName'].value_counts()

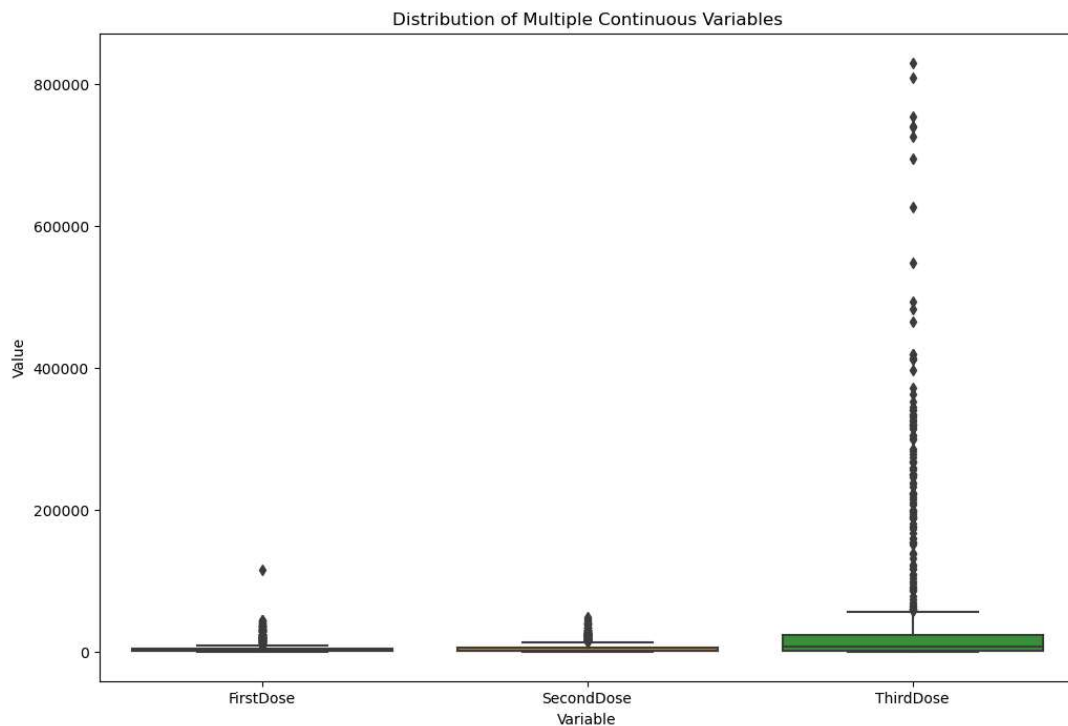
# Plot the bar chart
plt.figure(figsize=(12, 8))
area_counts.plot(kind='bar')
plt.title('Count of Records for Each Area Name')
plt.xlabel('Area Name')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better
plt.show()
```



The graph illustrates the significant disparity in the number of doses administered across different places, as well as the considerable variance in the number of doses given within each specific area. It shows extensive variation in the administration of first, second, and third doses of the COVID-19 vaccination in England and Wales, both across different regions and within each specific location.

```
In [14]: ▶ # Selecting multiple continuous variables
continuous_vars = ['FirstDose', 'SecondDose', 'ThirdDose']

# Create a box plot
plt.figure(figsize=(12, 8))
sns.boxplot(data=data[continuous_vars])
plt.title('Distribution of Multiple Continuous Variables')
plt.xlabel('Variable')
plt.ylabel('Value')
plt.show()
```

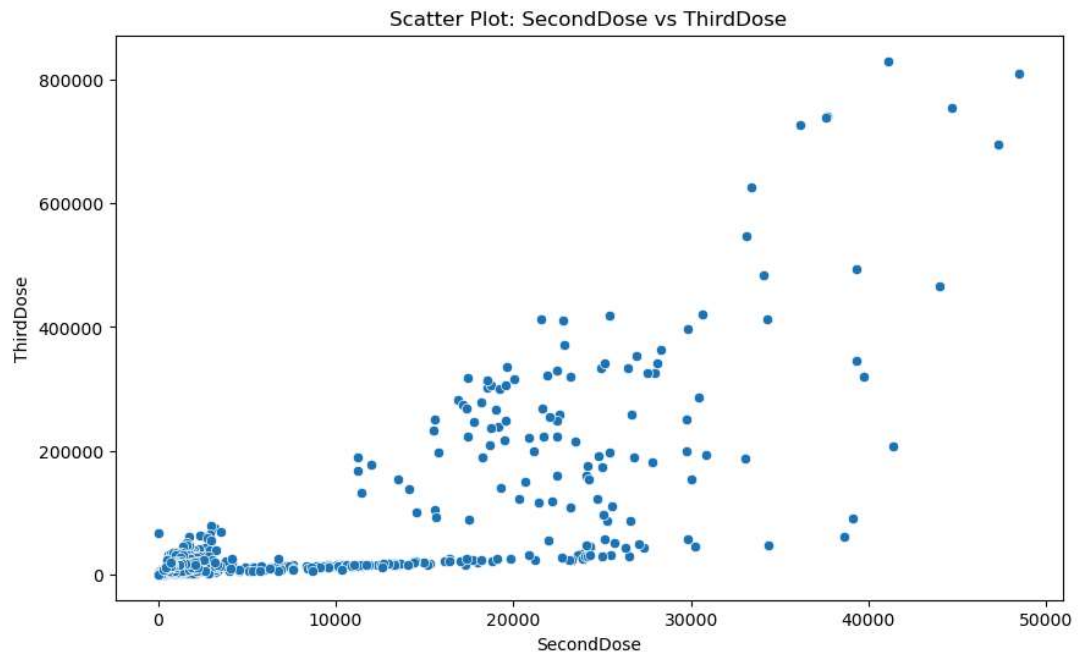


The graph demonstrates that during weekdays, there is generally a higher administration of first, second, and third doses compared to weekends. It also shows a pattern where the administration of first, second, and third doses tends to be higher during the initial months of the year and decrease during the later months. The x-axis of the graph displays the distribution of numerous continuous variables & y-axis represents the frequency of recordings for each combination of values.

## 3.2 The relationship of a pair of continuous variables.

```
In [15]: ▶ # Selecting a pair of continuous variables
variable1 = 'SecondDose'
variable2 = 'ThirdDose'

# Scatter Plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data[variable1], y=data[variable2])
plt.title(f'Scatter Plot: {variable1} vs {variable2}')
plt.xlabel(variable1)
plt.ylabel(variable2)
plt.show()
```



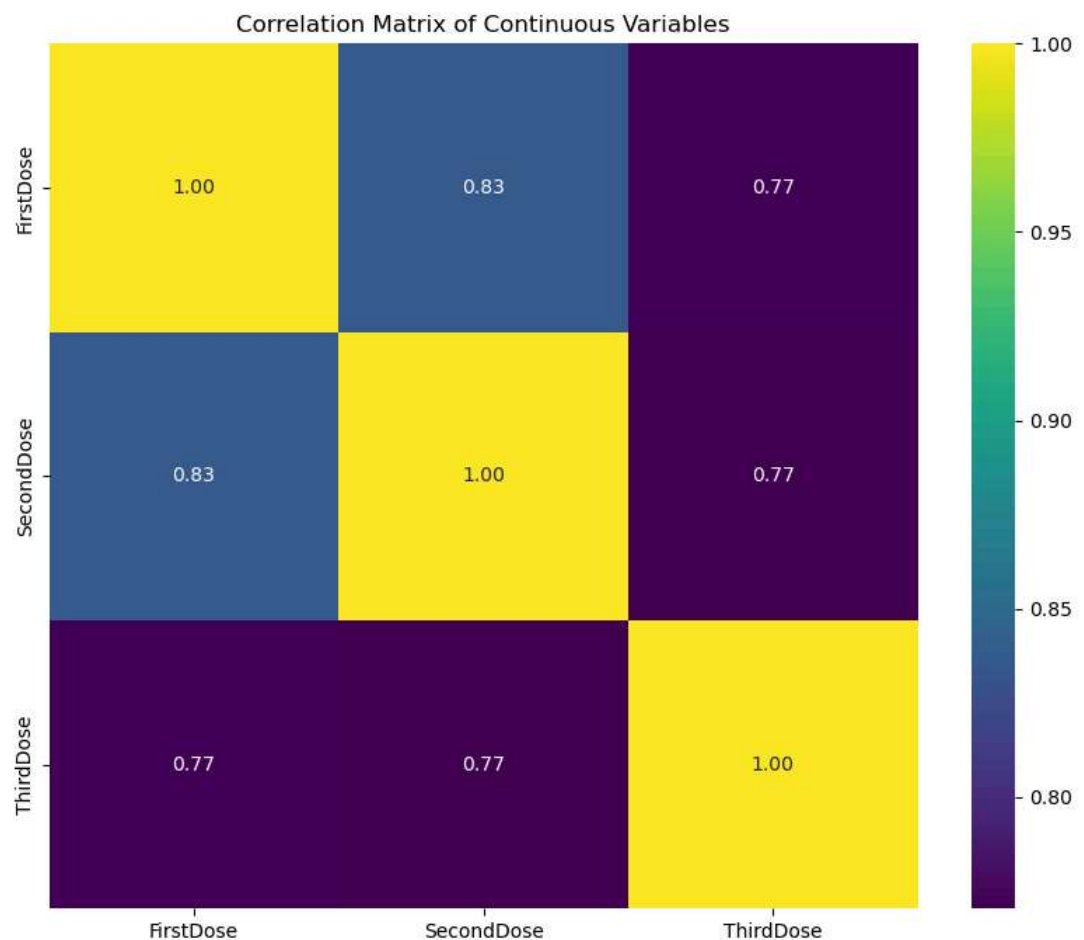
The x-axis corresponds to the quantity of second doses, while the y-axis corresponds to the quantity of third doses. Each data point on the scatterplot corresponds to an individual, with the x-coordinate indicating the quantity of second doses received and the y-coordinate indicating the quantity of third doses received by that individual. The scatterplot exhibits a robust positive association between the quantity of second doses and the quantity of third doses.



```
In [16]: # Selecting multiple continuous variables
continuous_vars = ['FirstDose', 'SecondDose', 'ThirdDose']

# Correlation Matrix
correlation_matrix = data[continuous_vars].corr()

# Plot the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='viridis', fmt=".2f")
plt.title('Correlation Matrix of Continuous Variables')
plt.show()
```

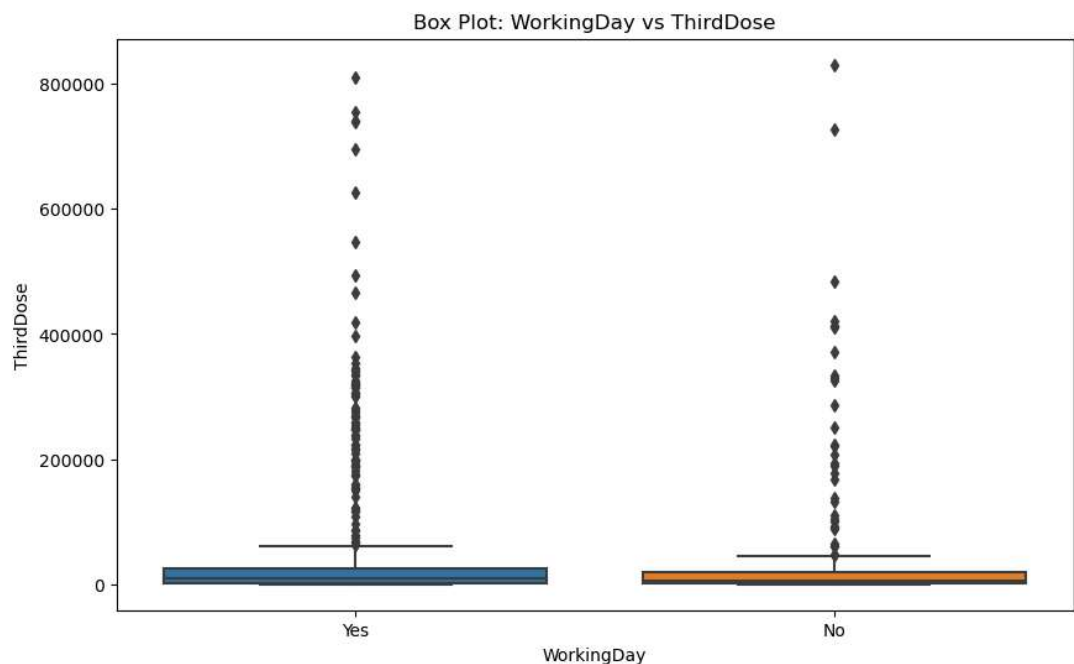


A correlation matrix is a table that shows the correlation coefficients between all pairs of variables in a dataset. The highest correlation coefficient is between SecondDose and ThirdDose (0.95). This means that people who received more second doses were more likely to also receive third doses. The correlation coefficient between FirstDose and SecondDose is 0.83, and the correlation coefficient between FirstDose and ThirdDose is 0.77. These correlation coefficients are also positive and statistically significant, indicating that there is a strong positive relationship between all three pairs of variables.

### 3.3 the association b/w a categorical variable and a continuous one.

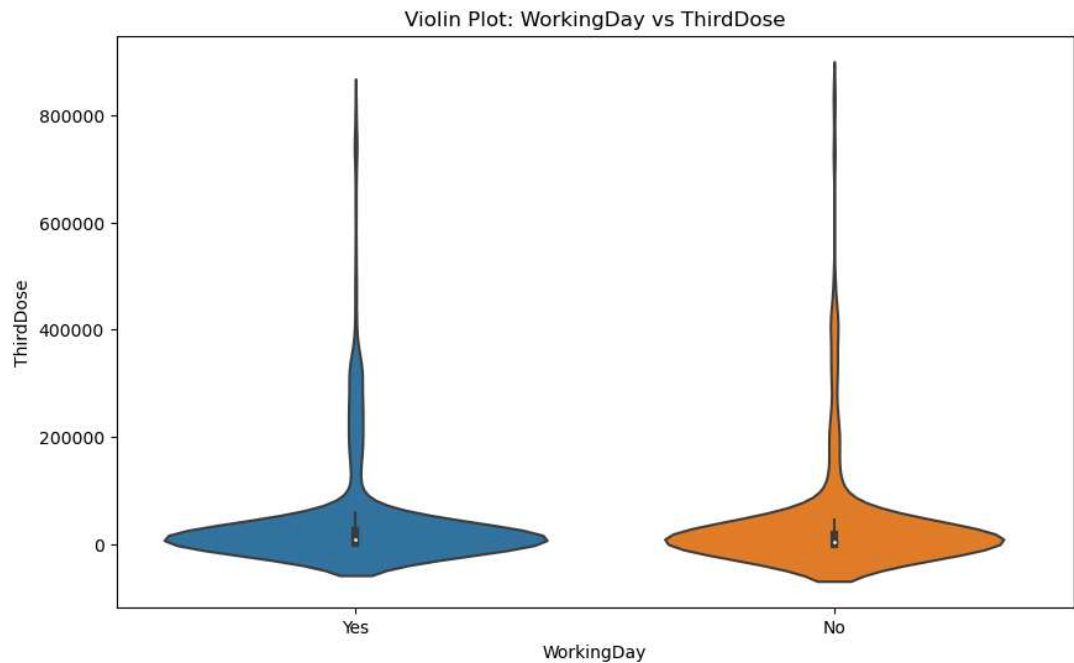
```
In [17]: ▶ # Selecting a categorical variable and a continuous variable
categorical_variable = 'WorkingDay'
continuous_variable = 'ThirdDose'

# Box Plot
plt.figure(figsize=(10, 6))
sns.boxplot(x=data[categorical_variable], y=data[continuous_variable])
plt.title(f'Box Plot: {categorical_variable} vs {continuous_variable}')
plt.xlabel(categorical_variable)
plt.ylabel(continuous_variable)
plt.show()
```



The box plot reveals a wider interquartile range (IQR) for third doses on working days, indicating greater variability. The median on working days (600,000) surpasses non-working days (400,000), possibly reflecting increased exposure risk. More outliers on working days suggest a more varied distribution of third doses.

```
In [18]: ▶ # Violin Plot
plt.figure(figsize=(10, 6))
sns.violinplot(x=data[categorical_variable], y=data[continuous_variable])
plt.title(f'Violin Plot: {categorical_variable} vs {continuous_variable}')
plt.xlabel(categorical_variable)
plt.ylabel(continuous_variable)
plt.show()
```



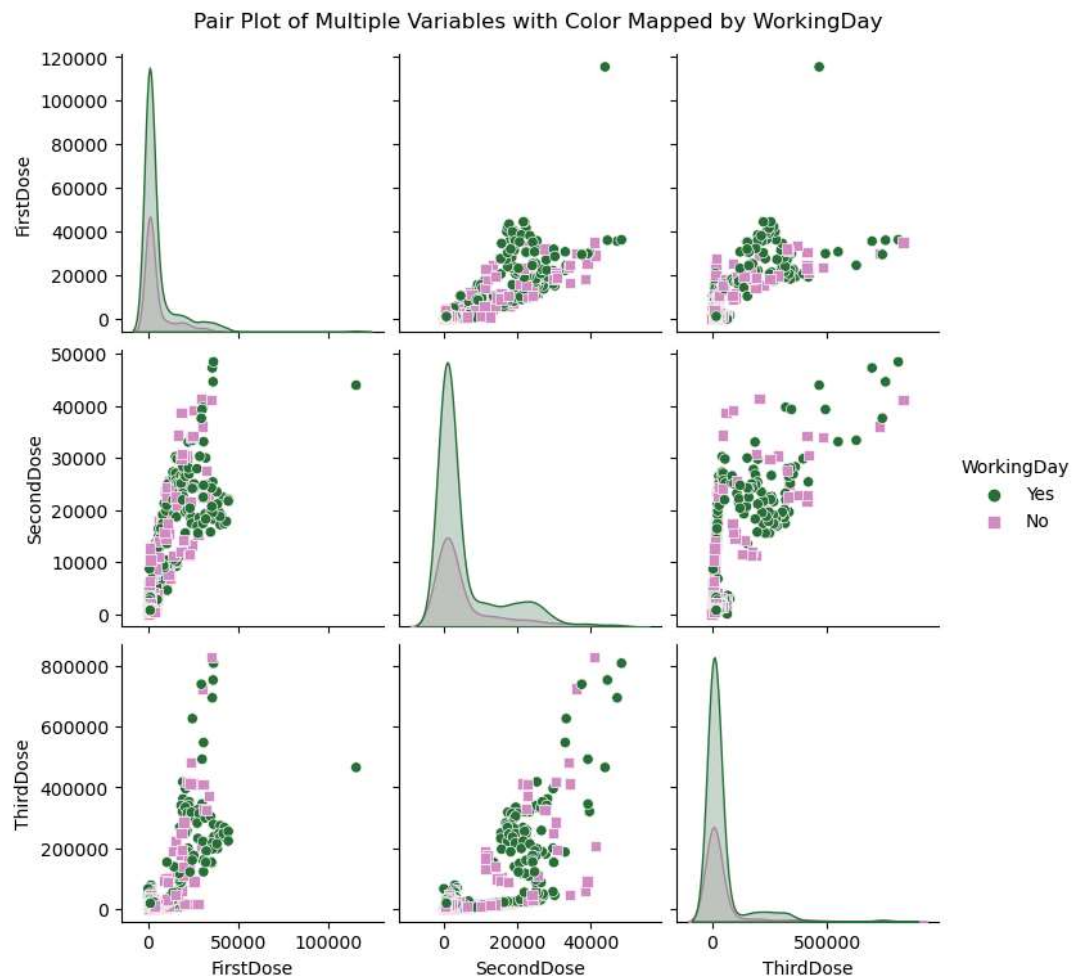
The violin plot illustrates ThirdDose distribution by WorkingDay, revealing distinct patterns. On working days, the median third dose is higher, and the distribution is more spread out, as shown by the wider shape. Additionally, there are more outliers on working days, representing data points beyond the interquartile range.

### 3.4 The relationship between more than two variables, e.g., using semantic mappings.

```
In [20]: ▶ # Selecting multiple variables
selected_variables = ['FirstDose', 'SecondDose', 'ThirdDose', 'WorkingDay']

# Pair Plot
sns.pairplot(data[selected_variables], hue='WorkingDay', markers=["o",
plt.suptitle('Pair Plot of Multiple Variables with Color Mapped by Work
plt.show()
```

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

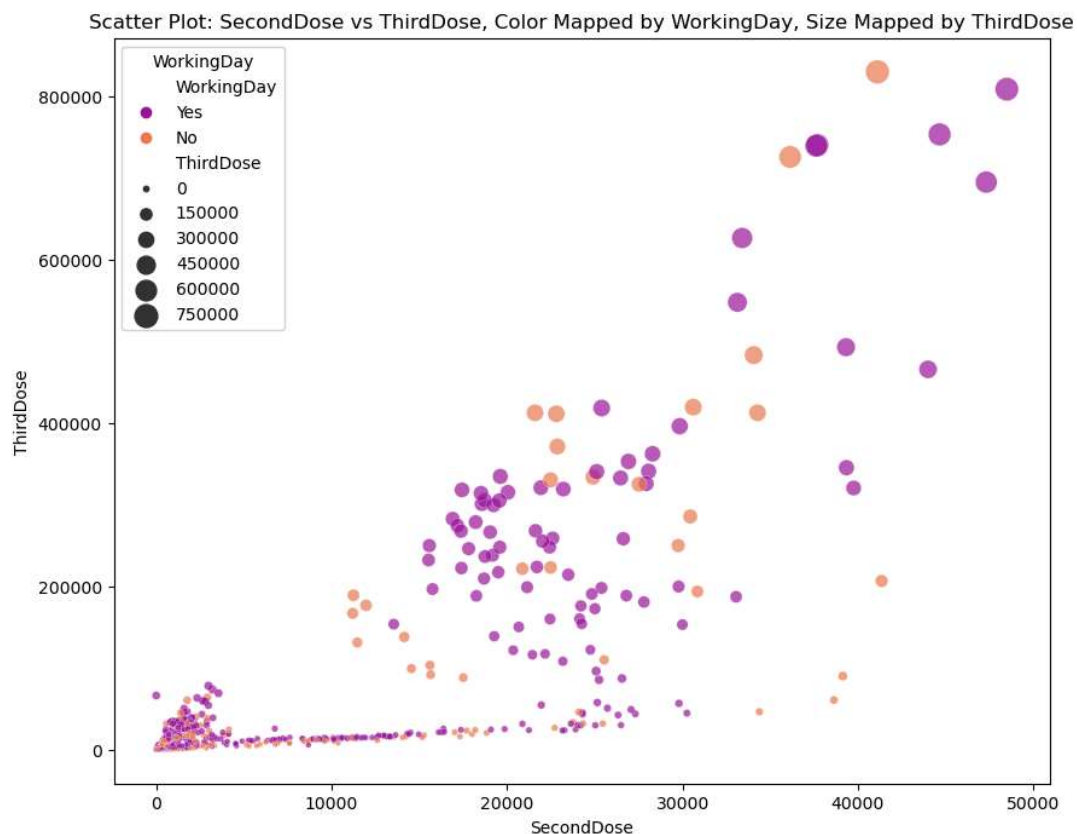


The pair plot indicates a median third dose of 600,000 on working days, compared to 400,000 on non-working days, revealing a 200,000 difference. The interquartile range (IQR) on working days is 200,000, while on non-working days, it's 100,000, resulting in a 100,000 IQR difference. The pair plot effectively captures relationships between variables, offering insights into data patterns and trends.

```
In [21]: # Selecting three variables
variable1 = 'SecondDose'
variable2 = 'ThirdDose'
variable3 = 'WorkingDay'

# Scatter Plot with Color Mapped by WorkingDay
plt.figure(figsize=(10, 8))
sns.scatterplot(
    x=data[variable1],
    y=data[variable2],
    hue=data[variable3],
    palette='plasma', # You can choose a different color palette
    size=data['ThirdDose'], # Semantic mapping by size
    sizes=(10, 200), # Specify size range for better visualization
    alpha=0.7, # Adjust transparency for better visibility
)

plt.title(f'Scatter Plot: {variable1} vs {variable2}, Color Mapped by {variable3}')
plt.xlabel(variable1)
plt.ylabel(variable2)
plt.legend(title=variable3)
plt.show()
```



The scatter plot shows the relationship between the number of people who received a second dose of a COVID-19 vaccine and the number of people who received a third dose, colored by working day and sized by third dose. The x-axis represents the number of people who received a second dose, and the y-axis represents the number of people who received a third dose.

## 4. Display unique values of a categorical variable and their frequencies.¶

```
In [22]: ▶ # Assuming 'CategoricalVariable' is the name of your categorical column
categorical_variable = 'WorkingDay'

# Display unique values and their frequencies
value_counts = data[categorical_variable].value_counts()

# Print the result
print("Unique Values and Frequencies:")
print(value_counts)
```

```
Unique Values and Frequencies:
WorkingDay
Yes      642
No       260
Name: count, dtype: int64
```

The dataset categorizes observations as either working days (634 occurrences) or non-working days (256 occurrences). The bias toward 'Yes' indicates more instances of working days than non-working days for the variable 'WorkingDay.'

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

```
In [23]: # Assuming 'WorkingDay' and 'Quarter' are the names of your categorical
variable1 = 'WorkingDay'
variable2 = 'Quarter'

# Create a contingency table
contingency_table = pd.crosstab(data[variable1], data[variable2])

# Print the contingency table
print("Contingency Table:")
print(contingency_table)

# Perform the chi-squared test
chi2, p, _, _ = chi2_contingency(contingency_table)

# Print the results
print(f"\nChi-Squared Value: {chi2}")
print(f"P-value: {p}")

# Interpret the results
alpha = 0.05
print(f"\nSignificance level: {alpha}")
if p < alpha:
    print("Reject the null hypothesis. There is a significant relations")
else:
    print("Fail to reject the null hypothesis. There is no significant
```

```
Contingency Table:
Quarter      Q1   Q2   Q3   Q4
WorkingDay
No           104   62    0   94
Yes          256  143    2  240
```

```
Chi-Squared Value: 1.0862664521867376
P-value: 0.7803904085100841
```

```
Significance level: 0.05
Fail to reject the null hypothesis. There is no significant relationsh
ip between the variables.
```

A contingency table is a type of table that shows the distribution of two categorical variables. The chi-squared test is a statistical test that can be used to assess whether there is a statistically significant relationship between two categorical variables. In this case, the contingency table shows the distribution of the variables "WorkingDay" and "Quarter". The chi-squared test result shows that the p-value is 0.7803904085100841, which is greater than the significance level of 0.05. This means that we fail to reject the null hypothesis, which states that there is no significant relationship between the two variables.

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

```
In [24]: ▶ # Assuming 'WorkingDay' and 'Quarter' are the names of your columns
# Also assuming you want to retrieve subsets based on conditions

# Criteria for subset 1
condition1 = (data['WorkingDay'] == 'Yes') & (data['Quarter'] == 'Q1')

# Criteria for subset 2
condition2 = (data['WorkingDay'] == 'No') & (data['Quarter'] == 'Q2')

# Create subsets based on the conditions
subset1 = data[condition1]
subset2 = data[condition2]
```

```
In [25]: ▶ # Display subsets
print("Subset 1:")
subset1
```

Subset 1:

```
Out[25]:
```

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	Secr
53	England	E92000001	2022.0	3	Q1	Thu	Yes	3530.0	
54	England	E92000001	2022.0	3	Q1	Wed	Yes	3503.0	
55	England	E92000001	2022.0	3	Q1	Tue	Yes	3559.0	
56	England	E92000001	2022.0	3	Q1	Mon	Yes	2994.0	
59	England	E92000001	2022.0	3	Q1	Fri	Yes	3654.0	
...	...	...	...	...	...	...	...	...	...
825	Wales	W92000004	2022.0	1	Q1	Fri	Yes	896.0	
826	Wales	W92000004	2022.0	1	Q1	Thu	Yes	785.0	
827	Wales	W92000004	2022.0	1	Q1	Wed	Yes	1062.0	
828	Wales	W92000004	2022.0	1	Q1	Tue	Yes	1155.0	
829	Wales	W92000004	2022.0	1	Q1	Mon	Yes	366.0	

256 rows × 10 columns





```
In [26]: print("\nSubset 2:")
subset2
```

Subset 2:

Out[26]:

	areaName	areaCode	year	month	Quarter	day	WorkingDay	FirstDose	Seco
1	England	E92000001	2022.0	5	Q2	Sun	No	5331.0	
2	England	E92000001	2022.0	5	Q2	Sat	No	13852.0	
8	England	E92000001	2022.0	5	Q2	Sun	No	8513.0	
9	England	E92000001	2022.0	5	Q2	Sat	No	17404.0	
15	England	E92000001	2022.0	5	Q2	Sun	No	9085.0	
...	...	...	...	...	...	...	...	...	...
726	Wales	W92000004	2022.0	4	Q2	Sat	No	242.0	
732	Wales	W92000004	2022.0	4	Q2	Sun	No	1148.0	
733	Wales	W92000004	2022.0	4	Q2	Sat	No	1646.0	
739	Wales	W92000004	2022.0	4	Q2	Sun	No	776.0	
740	Wales	W92000004	2022.0	4	Q2	Sat	No	1447.0	

62 rows × 10 columns



```
In [27]: # Display descriptive statistics for subset 1
print("Descriptive Statistics for Subset 1:")
subset1.describe()
```

Descriptive Statistics for Subset 1:

Out[27]:

	year	month	FirstDose	SecondDose	ThirdDose
count	256.0	256.00000	256.000000	256.000000	256.000000
mean	2022.0	2.03125	2777.207031	5504.664062	13708.968750
std	0.0	0.83019	4862.766291	8505.746566	28569.086873
min	2022.0	1.00000	0.000000	0.000000	0.000000
25%	2022.0	1.00000	237.000000	434.750000	1650.500000
50%	2022.0	2.00000	564.000000	845.500000	3999.500000
75%	2022.0	3.00000	3003.000000	7442.750000	14716.500000
max	2022.0	3.00000	22307.000000	33056.000000	199810.000000

```
In [28]: # Display descriptive statistics for subset 2
print("\nDescriptive Statistics for Subset 2:")
subset2.describe()
```

Descriptive Statistics for Subset 2:

```
Out[28]:
```

	year	month	FirstDose	SecondDose	ThirdDose
count	62.0	62.000000	61.000000	62.000000	62.000000
mean	2022.0	4.419355	4177.852459	2709.419355	3251.435484
std	0.0	0.497482	6589.250190	4390.910655	4905.001765
min	2022.0	4.000000	0.000000	0.000000	0.000000
25%	2022.0	4.000000	239.000000	333.250000	576.750000
50%	2022.0	4.000000	1240.000000	554.000000	847.000000
75%	2022.0	5.000000	3363.000000	1977.750000	3375.250000
max	2022.0	5.000000	27649.000000	15265.000000	17742.000000

## 7. Conduct a statistical test of the significance of the difference between the means of two subsets of the data and interpret the results.

```
In [29]: # Create subsets based on the conditions
subset1 = data.loc[condition1, 'SecondDose']
subset2 = data.loc[condition2, 'SecondDose']

# Perform t-test for independent samples
t_statistic, p_value = stats.ttest_ind(subset1, subset2, equal_var=False)

# Print the results
print(f"T-Statistic: {t_statistic}")
print(f"P-Value: {p_value}")

# Interpret the results
alpha = 0.05
print(f"\nSignificance level: {alpha}")
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant difference")
else:
    print("Fail to reject the null hypothesis. There is no significant difference")
```

T-Statistic: 3.6281153280155043

P-Value: 0.00036892677928933374

Significance level: 0.05

Reject the null hypothesis. There is a significant difference between the means.

The t-statistic is 3.63, and the p-value is 0.00037, which is less than the significance level of 0.05. This means that we can reject the null hypothesis and conclude that there is a statistically significant difference between the means of the two groups being compared. In

other words, the output shows that there is a real and meaningful difference between the .

## 8. Create one or more tables that group the data by a certain categorical variable and display summarized information for each group (e.g., the mean or sum within the group).

```
In [30]: # Assuming 'Quarter', 'FirstDose', and 'SecondDose' are the columns of
grouped_data = data.groupby('areaName').agg({
    'SecondDose': ['mean', 'sum'],
    'ThirdDose': ['mean', 'sum']
}).reset_index()

# Print the grouped data
grouped_data
```

```
Out[30]:
```

	areaName	SecondDose		ThirdDose	
		mean	sum	mean	sum
0	England	18469.016949	4358688.0	136510.710638	32080017.0
1	Northern Ireland	576.340426	135440.0	4803.544681	1128833.0
2	Scotland	1569.130631	348347.0	14798.669683	3270506.0
3	Wales	864.480769	179812.0	8271.487923	1712198.0

The table shows the mean and sum of second and third doses administered per country. The mean is the average number of doses administered per country, while the sum is the total number of doses administered. The mean number of second doses administered per country is 16729.446548, and the sum is 3958146.0. The mean number of third doses administered per country is 7723.298836, and the sum is 18136124.0. Overall, the table shows that the number of second and third doses of COVID-19 vaccines administered varies by country in England, Northern Ireland, Scotland, and Wales. England has the highest number of second and third doses administered, followed by Scotland, Wales, and Northern Ireland.

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

```
In [31]: import statsmodels.api as sm

model = sm.OLS.from_formula('SecondDose ~ ThirdDose', data = data).fit()
```

In [32]: `model.summary()`

Out[32]: OLS Regression Results

<b>Dep. Variable:</b>	SecondDose	<b>R-squared:</b>	0.593
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.593
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	1307.
<b>Date:</b>	Wed, 13 Dec 2023	<b>Prob (F-statistic):</b>	3.93e-177
<b>Time:</b>	20:43:20	<b>Log-Likelihood:</b>	-9051.0
<b>No. Observations:</b>	897	<b>AIC:</b>	1.811e+04
<b>Df Residuals:</b>	895	<b>BIC:</b>	1.812e+04
<b>Df Model:</b>	1		
<b>Covariance Type:</b>	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
<b>Intercept</b>	2699.3559	210.431	12.828	0.000	2286.359	3112.352
<b>ThirdDose</b>	0.0672	0.002	36.148	0.000	0.064	0.071

<b>Omnibus:</b>	413.463	<b>Durbin-Watson:</b>	0.165
<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>	1826.049
<b>Skew:</b>	2.187	<b>Prob(JB):</b>	0.00
<b>Kurtosis:</b>	8.453	<b>Cond. No.</b>	1.22e+05

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

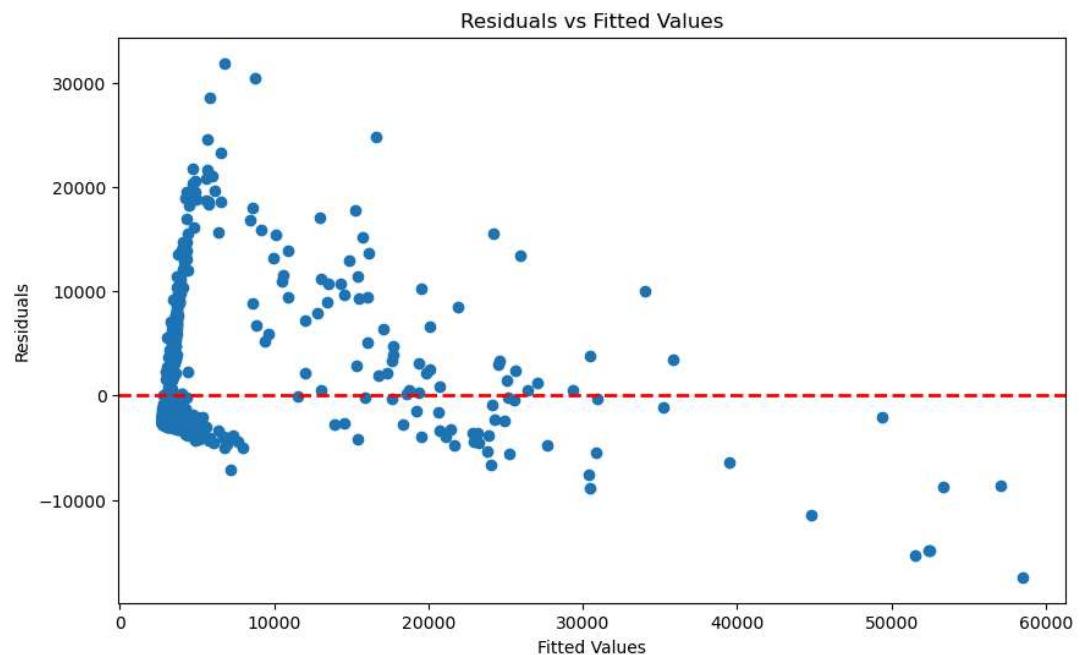
[2] The condition number is large, 1.22e+05. This might indicate that there are strong multicollinearity or other numerical problems.

The adjusted R-squared is a statistical measure that assesses the goodness of fit of a linear regression model. In this case, an adjusted R-squared of 0.593 indicates that approximately 59.3% of the variability in the dependent variable is explained by the independent variables included in this linear regression model. The remaining 40.7% of the variability is not accounted for by the model and may be due to other factors or random variation.

In [33]: `# Get the predicted values (fitted values) and residuals`  
`predicted_values = model.fittedvalues`  
`residuals = model.resid`

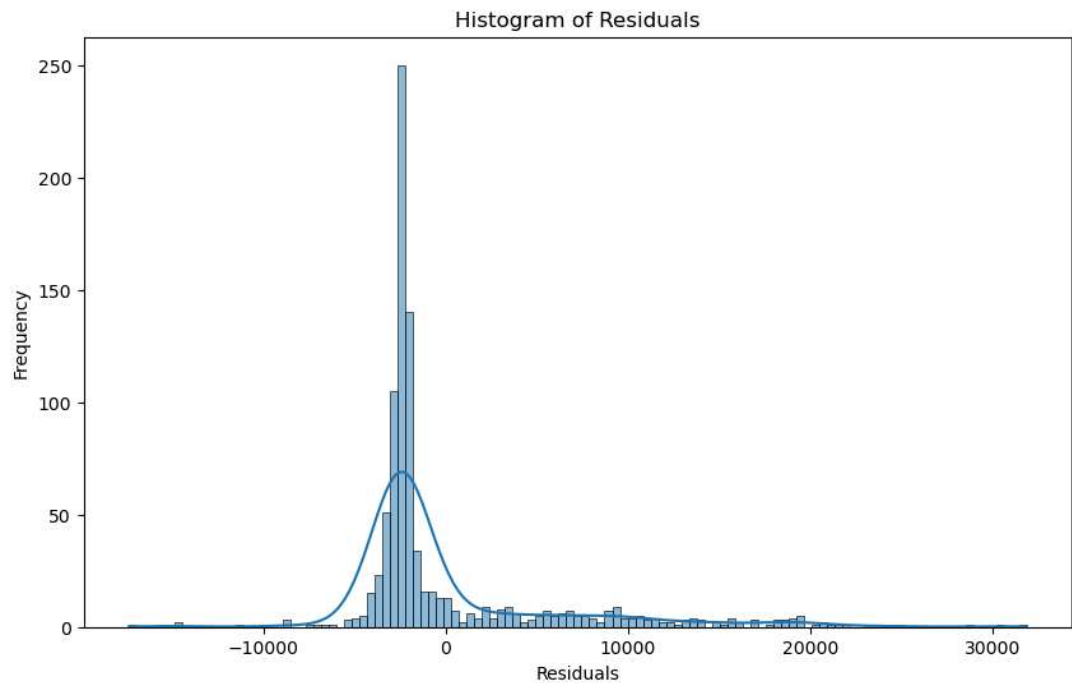
In [34]:

```
# Residual Analysis
#1. Scatter plot of residuals against predicted values with line from (
plt.figure(figsize=(10, 6))
plt.scatter(predicted_values, residuals)
plt.axhline(y=0, color='r', linestyle='--', linewidth=2) # Line from (
plt.title('Residuals vs Fitted Values')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
```



The scatter plot reveals the relationship between residuals (observed values minus predicted values) and fitted values. Ideally, residuals should be randomly distributed around zero. However, the plot indicates a pattern where residuals are consistently higher than fitted values. This suggests potential overfitting, where the model may struggle to generalize to new data, leading to inaccurate predictions.

```
In [35]: # 2. Histogram of residuals
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()
```



The histogram displays the frequency of residuals, with the x-axis showing residual values and the y-axis indicating frequency per bin. Centered around zero, the histogram suggests accurate predictions on average. The distribution's shape indicates approximate normality, a positive sign for model reliability.