In [2]:
```
#ポケモンのデータの分析
#データの読み込み
import pandas as pd
df=pd.read_csv("C:¥¥Users¥¥syuhe¥¥Downloads¥¥Pokemon.csv")
df.head(5)
```

Out[2]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Leg
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	

In [3]:
```
#Type1とGeneration別のAttackとDefenseの平均を集計.
#Type1別のAttackとDefenseの平均の集計
df.groupby('Type 1')['Attack', 'Defense'].mean().reset_index()
```

```
C:¥Users¥syuhe¥AppData¥Local¥Temp¥ipykernel_13100¥2231199949.py:3: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
  df.groupby('Type 1')['Attack', 'Defense'].mean().reset_index()
```

Out[3]:

	Type 1	Attack	Defense
0	Bug	70.971014	70.724638
1	Dark	88.387097	70.225806
2	Dragon	112.125000	86.375000
3	Electric	69.090909	66.295455
4	Fairy	61.529412	65.705882
5	Fighting	96.777778	65.925926
6	Fire	84.769231	67.769231
7	Flying	78.750000	66.250000
8	Ghost	73.781250	81.187500
9	Grass	73.214286	70.800000
10	Ground	95.750000	84.843750
11	Ice	72.750000	71.416667
12	Normal	73.469388	59.846939
13	Poison	74.678571	68.821429
14	Psychic	71.456140	67.684211
15	Rock	92.863636	100.795455
16	Steel	92.703704	126.370370
17	Water	74.151786	72.946429

In [4]:
```
#Generation別のAttackとDefenseの平均の集計
df.groupby('Generation')['Attack', 'Defense'].mean().reset_index()
```

```
C:¥Users¥syuhe¥AppData¥Local¥Temp¥ipykernel_13100¥2439683554.py:3: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
  df.groupby('Generation')['Attack', 'Defense'].mean().reset_index()
```

Out[4]:

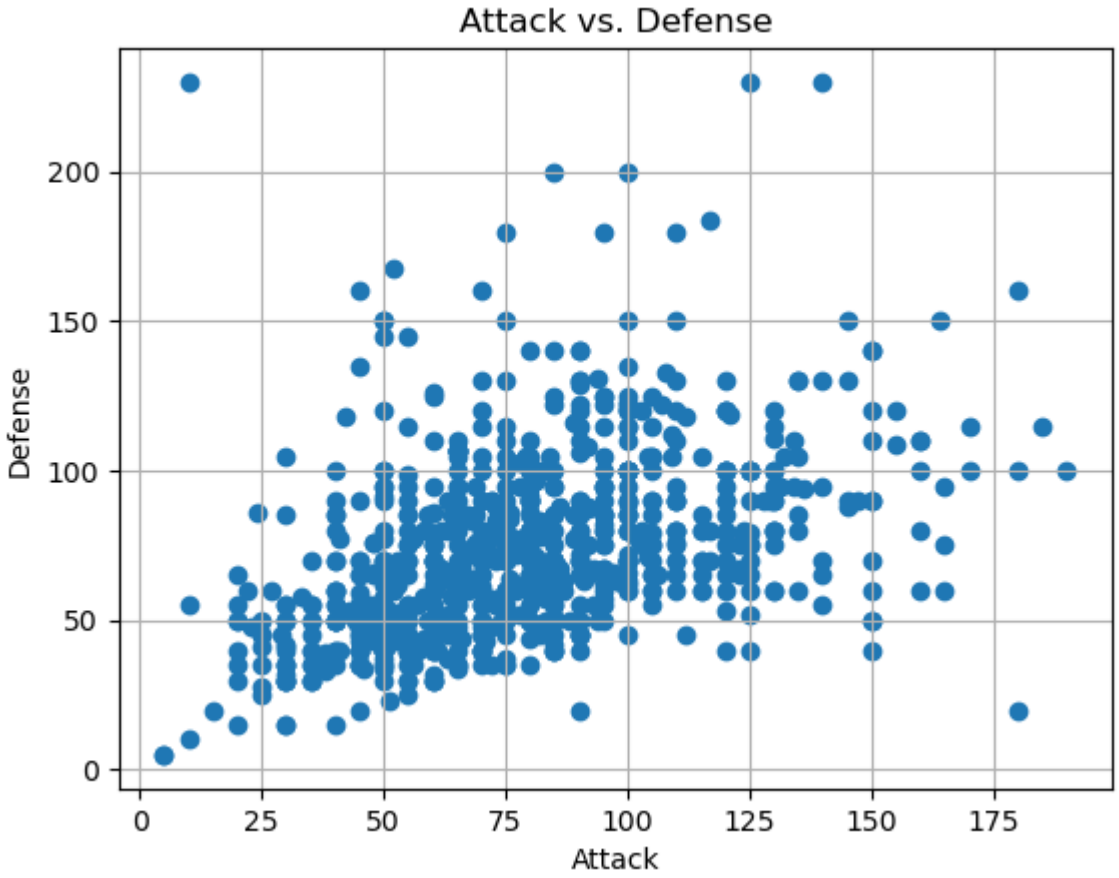
	Generation	Attack	Defense
0	1	76.638554	70.861446
1	2	72.028302	73.386792
2	3	81.625000	74.100000
3	4	82.867769	78.132231
4	5	82.066667	72.327273
5	6	75.804878	76.682927

In [5]:
```
#伝説(Legendary)のポケモンで, AttackがDefenseより小さく, Attackが90以下, Speedが110以
filtered_pokemon = df[(df['Legendary'] == True) & (df['Attack'] < df['Defense']) &
filtered_pokemon
```

Out[5]:

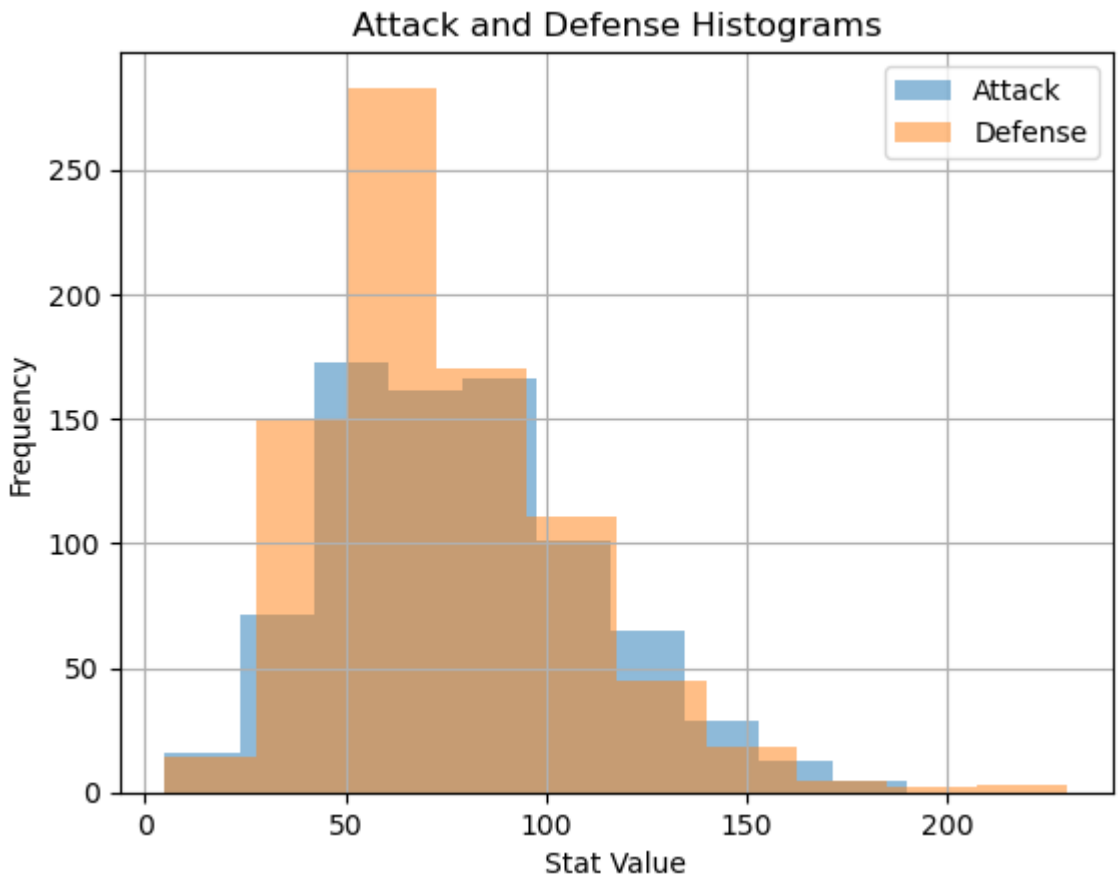
	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Leg
269	249	Lugia	Psychic	Flying	680	106	90	130	90	154	110	2	
417	380	Latias	Dragon	Psychic	600	80	80	90	110	130	110	3	

In [6]:
```
#すべてのレコードに対し，AttackとDefenseの関係を散布図に描画.
import matplotlib.pyplot as plt
plt.scatter(df['Attack'], df['Defense'])
plt.xlabel('Attack')
plt.ylabel('Defense')
plt.title('Attack vs. Defense')
plt.grid(True)
plt.show()
```



In [7]:
```
#すべてのレコードに対し，AttackとDefenseのそれぞれのヒストグラムを同じスケールで重ね
# ヒストグラムを描画
plt.hist(df['Attack'], bins=10, alpha=0.5, label='Attack')
plt.hist(df['Defense'], bins=10, alpha=0.5, label='Defense')

# 軸ラベルとタイトルの設定
plt.xlabel('Stat Value')
plt.ylabel('Frequency')
plt.title('Attack and Defense Histograms')

# 凡例の表示
plt.legend()

# グリッドを表示
plt.grid(True)
```

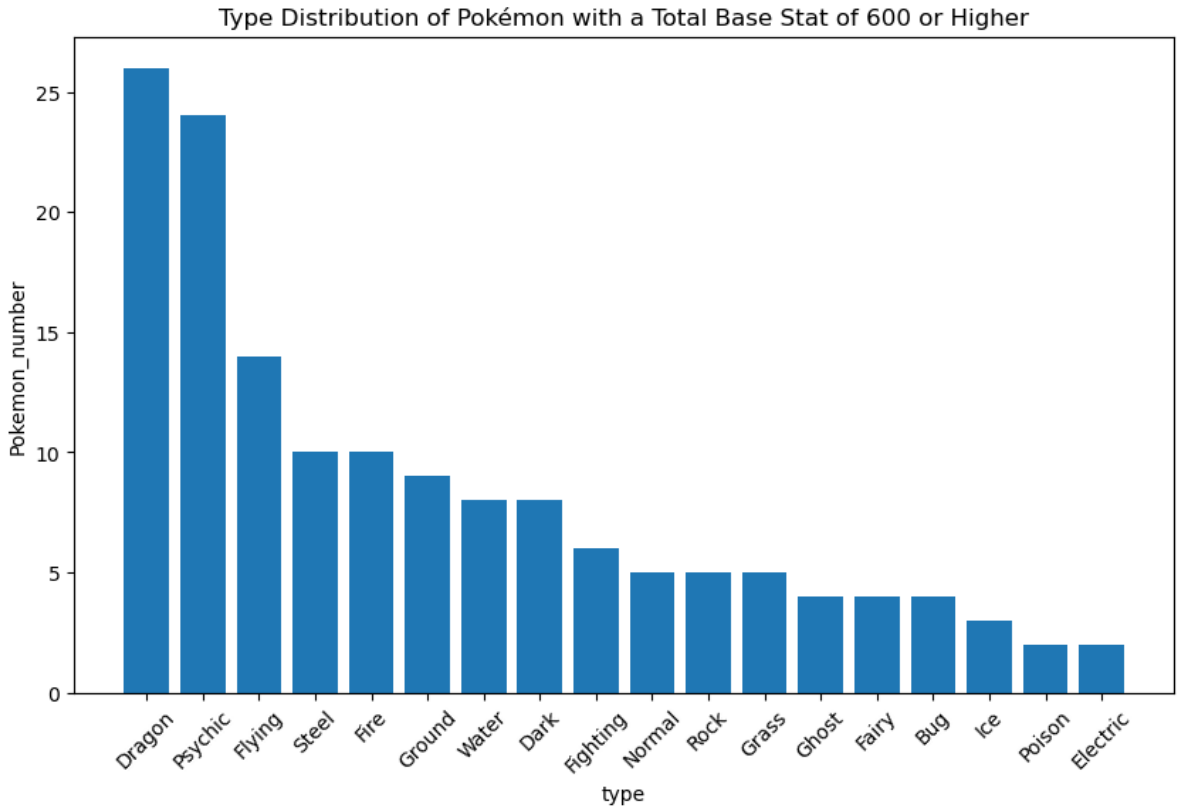
```
# グラフを表示
plt.show()
```



In [8]:

```
#ポケモンの種族値の合計が600以上のポケモンに着目し、タイプの分布を調査する。
#なぜ、種族値の合計が600以上のポケモンに着目するかというと、ポケモンの対戦環境におい
#具体的には、種族値の合計が600以上のポケモンがどのタイプに多く存在するかを調べる。
#この分析には、棒グラフが適していると考える。
#図の種類の選択理由：
#同じ尺度の複数のデータを並べて比較するのに適している。
#また、棒グラフを見ると、どれがどれだけ頭 1 つ抜き出ているか、あるいは同じ程度である
#どのタイプにどれだけの種族値の合計が600以上のポケモンがいるかは同じ尺度のデータであ
#棒グラフを使うことで、種族値の合計が600以上のポケモンが各タイプにどれだけ存在するか

# 種族値の合計が600以上のポケモンのみ抽出
df1 = df[df['Total'] >= 600]

# 各タイプごとに、種族値の合計が600以上のポケモンがいくつあるかを数える
type_counts1 = df1['Type 1'].value_counts()
type_counts2 = df1['Type 2'].value_counts()

# NaN 値を適切に処理しながら、combine 関数を使ってカウントを合算します
type_counts = type_counts1.combine(type_counts2, lambda x1, x2: (x1 if pd.notnull(x

# 値に基づいて降順にソート
type_counts = type_counts.sort_values(ascending=False)

# 積み上げ棒グラフを作成
plt.figure(figsize=(10, 6))
plt.bar(type_counts.index, type_counts.values)
plt.xlabel('type')
plt.ylabel('Pokemon_number')
plt.title('Type Distribution of Pokémon with a Total Base Stat of 600 or Higher')
plt.xticks(rotation=45)
plt.show()
```

```
#結論としてはグラフを見る限り、ドラゴンタイプとエスパータイプが群を抜いて多いことが分
#逆に電気タイプ、毒タイプ、氷タイプ、フェアリータイプは少ない。
#ただ、フェアリータイプは比較的新しいタイプであるので、ポケモンの種類が少ない可能性が

Dragon_Type_data = df[(df['Type 1'] == 'Dragon') | (df['Type 2'] == 'Dragon')]

Fairy_Type_data = df[(df['Type 1'] == 'Fairy') | (df['Type 2'] == 'Fairy')]
len(Fairy_Type_data)

print("ドラゴンタイプの数:", len(Dragon_Type_data))
print("フェアリータイプの数:", len(Fairy_Type_data))

#ドラゴンタイプは50匹に対してフェアリータイプのポケモンは40匹でさほど数に差がない。
#そのため、ドラゴンタイプはやはり種族値が高いポケモンが多い。
```




```
ドラゴンタイプの数: 50
フェアリータイプの数: 40
```

In [9]:

```
#通販の受注のデータに対する分析
#データの読み込み
df=pd.read_excel("C:¥¥Users¥¥syuhe¥¥Downloads¥¥onlineRetail.xlsx")

# Invoice列を文字列に変換
df['InvoiceNo'] = df['InvoiceNo'].astype(str)

# Invoice列の値がCで始まる行を削除する
df = df[~df['InvoiceNo'].str.startswith('C')]

df.head(5)
```

Out[9]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

In [10]:
```
#通販の受注のデータを分析環境にデータフレームとして取り込み，各列の統計値を示す.
df.describe()
```

Out[10]:

	Quantity	UnitPrice	CustomerID
count	532621.000000	532621.000000	397924.000000
mean	10.239972	3.847621	15294.315171
std	159.593551	41.758023	1713.169877
min	-9600.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13969.000000
50%	3.000000	2.080000	15159.000000
75%	10.000000	4.130000	16795.000000
max	80995.000000	13541.330000	18287.000000

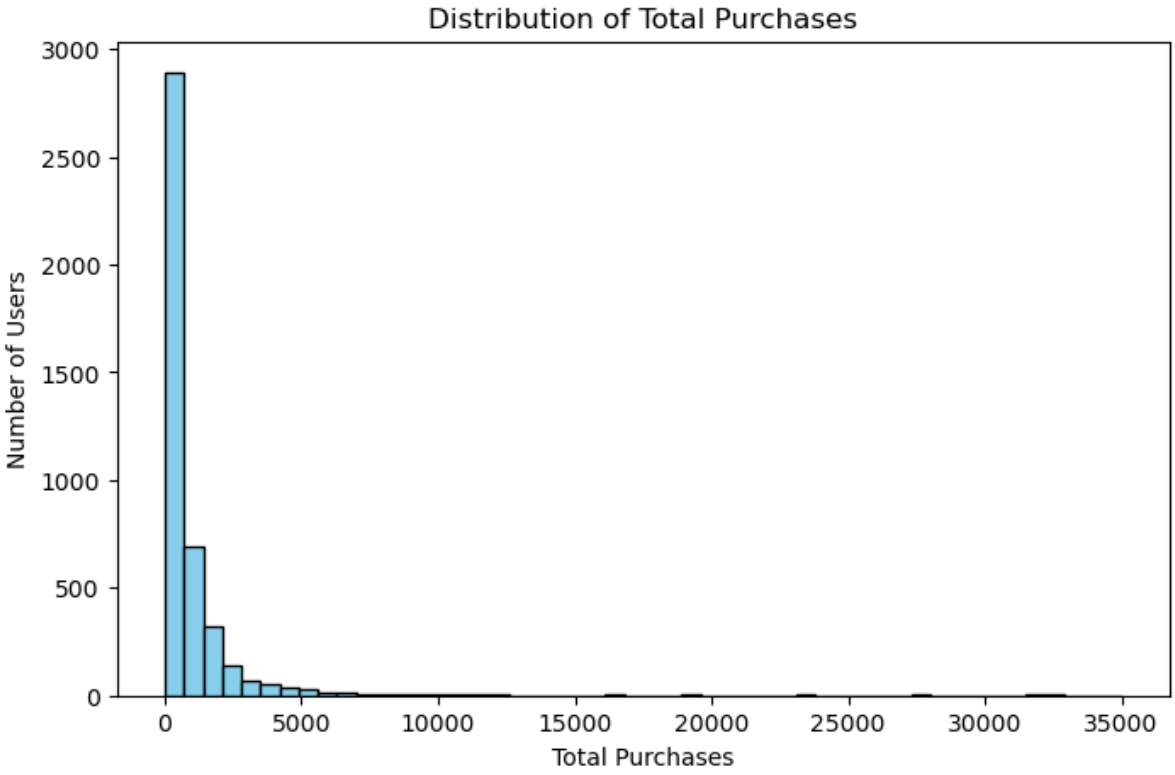
In [12]:
```
# 'UnitPrice'列が0の行を削除
df_filtered = df[df['UnitPrice'] != 0]
#データを分析環境にデータフレームとして取り込み，各列の統計値を示す.
df_filtered['UnitPrice'].describe()
```

Out[12]:
```
count    530106.000000
mean          3.865875
std          41.856120
min      -11062.060000
25%           1.250000
50%           2.080000
75%           4.130000
max       13541.330000
Name: UnitPrice, dtype: float64
```

In [13]:
```
#1回の請求額の平均値と分散を示す.
# InvoiceNoごとにUnitPriceをまとめる
grouped_data = df.groupby('InvoiceNo')['UnitPrice'].sum()
# 請求額の平均値を計算
mean_billing_amount = grouped_data.mean()

# 請求額の分散を計算
variance_billing_amount = grouped_data.var()

# 結果を表示
print("1回の請求額の平均値:", mean_billing_amount)
print("1回の請求額の分散:", variance_billing_amount)
```

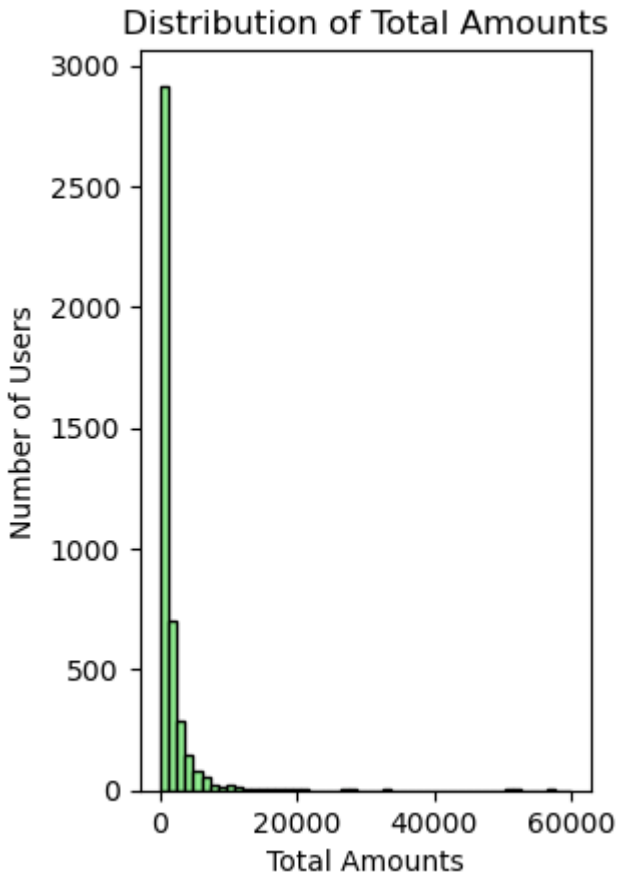
1回の請求額の平均値: 92.88087626903484
1回の請求額の分散: 100224.78726545554

In [58]:
```
#ユーザ別の総購買個数と総購買金額の分布を示す.
# ユーザ別の総購買個数を計算
total_purchases = df.groupby('CustomerID')['Quantity'].sum()

# ユーザ別の総購買金額を計算
total_amounts = df.groupby('CustomerID').apply(lambda x: (x['Quantity'] * x['UnitPr

# ユーザ別の総購買個数のヒストグラム(グラフの横軸の上限を35000に設定)
plt.figure(figsize=(8, 5))
plt.hist(total_purchases, bins=50, range=(0, 35000),color='skyblue', edgecolor='blac
plt.xlabel('Total Purchases')
plt.ylabel('Number of Users')
plt.title('Distribution of Total Purchases')
plt.show()
```



In [59]:
```
#ユーザ別の総購買金額のヒストグラム(グラフの横軸の上限を60000に設定)
plt.subplot(1, 2, 2)
plt.hist(total_amounts, bins=50, range=(0, 60000),color='lightgreen', edgecolor='bla
plt.xlabel('Total Amounts')
plt.ylabel('Number of Users')
plt.title('Distribution of Total Amounts')
```

Out[59]:
```
Text(0.5, 1.0, 'Distribution of Total Amounts')
```



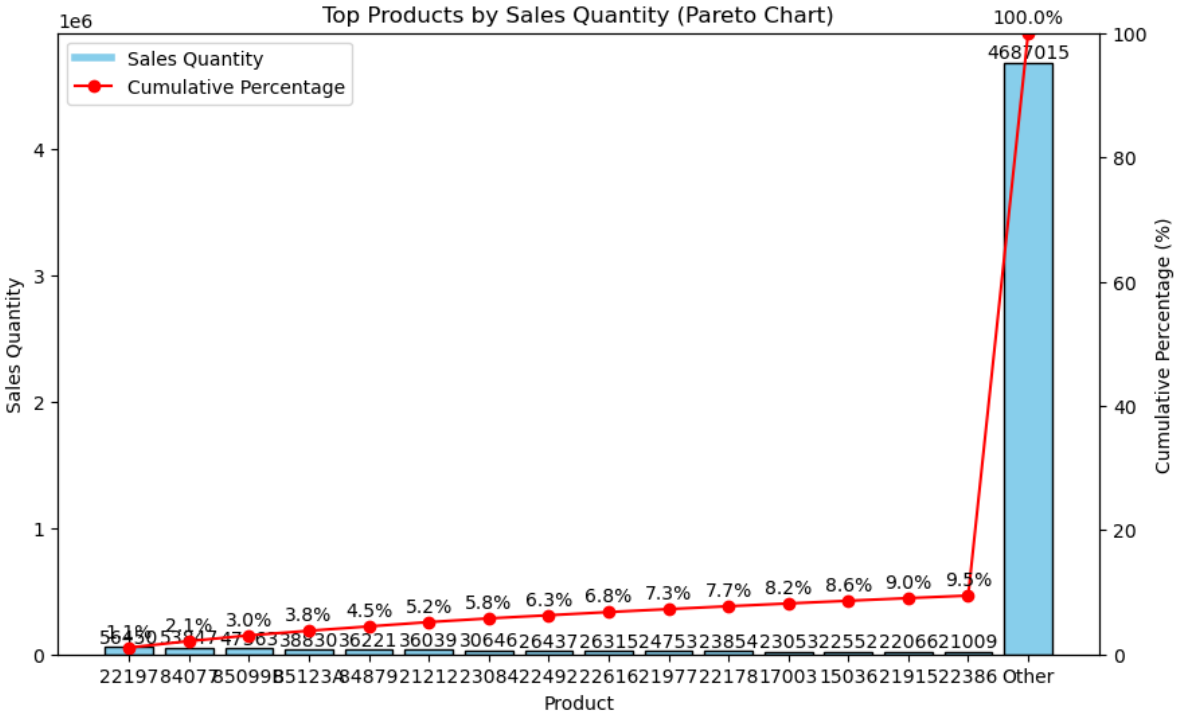
In [60]:
```
#商品別の売上数の上位10件を示す. また、全体のパレート図を示す.

# 商品別の売上数を計算
sales_by_product = df.groupby('StockCode')['Quantity'].sum().sort_values(ascending=F

#商品別の売上数の上位10件を示せ.
sales_by_product.head(10)
```

Out[60]:

	StockCode	Quantity
0	22197	56450
1	84077	53847
2	85099B	47363
3	85123A	38830
4	84879	36221
5	21212	36039
6	23084	30646
7	22492	26437
8	22616	26315
9	21977	24753

In [61]:
```
#また、全体のパレート図を示す.
from matplotlib.lines import Line2D
from matplotlib.legend_handler import HandlerTuple
# パレート図を作成
plt.figure(figsize=(10, 6))
# 販売数量で降順にソート
sales_by_product.sort_values(by='Quantity', ascending=False, inplace=True)
# 上位15件の商品のみを抽出
sales_by_product_top15=sales_by_product.head(15)

# 残りの商品の販売数量を合計して、「その他」としてまとめる
# 上位15件の商品の販売数量を合計
top_quantity_sum = sales_by_product_top15['Quantity'].sum()

# 「その他」の商品の販売数量を合計
other_quantity = sales_by_product['Quantity'].sum() - top_quantity_sum
other_row = pd.DataFrame({'StockCode': ['Other'], 'Quantity': [other_quantity]})
sales_by_product_15 = pd.concat([sales_by_product_top15, other_row])

# 累積割合を計算
total_sales = sales_by_product_15['Quantity'].sum()
sales_by_product_15['Percentage'] = (sales_by_product_15['Quantity'] / total_sales)
sales_by_product_15['Cumulative Percentage'] = sales_by_product_15['Percentage'].cum

StockCode=sales_by_product_15['StockCode']
str_StockCode=list(map(str, StockCode))
Quantity=sales_by_product_15['Quantity']
# パレート図の棒グラフを表示
bars = plt.bar(str_StockCode,Quantity, color='skyblue', edgecolor='black',label='Sal
plt.xlabel('Product')
plt.ylabel('Sales Quantity')
plt.title('Top Products by Sales Quantity (Pareto Chart)')

# 販売数量の値を表示
for bar, value in zip(bars, sales_by_product_15['Quantity']):
```

```
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 5, str(value), ha

# 2軸の設定
ax2 = plt.twinx()
# 累積割合の折れ線グラフを描画（右軸）
line = ax2.plot(str_StockCode, sales_by_product_15['Cumulative Percentage'], marker=

ax2.set_ylabel('Cumulative Percentage (%)')
ax2.set_ylim(0, 100)

# 累積割合の値を表示
for x, y in zip(str_StockCode, sales_by_product_15['Cumulative Percentage']):
    ax2.text(x, y + 1, f'{y:.1f}%', ha='center', va='bottom')

# プロキシアーティストを用いてラベルを付ける
legend_elements = [Line2D([0], [0], color='skyblue', lw=4, label='Sales Quantity'),
                   Line2D([0], [0], marker='o', color='red', label='Cumulative Perce

# レジェンドを作成
plt.legend(handles=legend_elements, handler_map={tuple: HandlerTuple(ndivide=None)}

plt.show()
#商品の種類が多すぎるため、その他がほとんどの割合を示す図となってしまった。
```



In [62]:
```
#顧客を国ごとに層別し，顧客数の上位5カ国を示す.
# 国ごとに顧客数をカウントして降順にソート
customer_count_by_country = df['Country'].value_counts().reset_index()
customer_count_by_country.columns = ['Country', 'CustomerCount']
customer_count_by_country.head(5)
```

Out[62]:

	Country	CustomerCount
0	United Kingdom	495478
1	Germany	9495
2	France	8557
3	EIRE	8196
4	Spain	2533

In [63]:
```
#顧客数の上位3カ国ごとに購買金額の多いユーザ上位5名を示す.
# 上位3カ国を取得
top_3_countries = customer_count_by_country.head(3)
top_3_countries
```

Out[63]:

	Country	CustomerCount
0	United Kingdom	495478
1	Germany	9495
2	France	8557

In [64]:
```
#上のセルのコードの実行により顧客数の上位3か国はUnited Kingdom、Germany 、Franceであ
#United Kingdomの購買金額の多いユーザ上位5名は、
United_Kingdom_data = df[df['Country'] == 'United Kingdom']
United_Kingdom_total_amounts = United_Kingdom_data.groupby('CustomerID').apply(lambd
United_Kingdom_total_amounts.sort_values(by='Totalamount', ascending=False).head(5)
```

Out[64]:

	CustomerID	Totalamount
3811	18102.0	256438.49
3340	17450.0	187482.17
3382	17511.0	88125.38
2788	16684.0	65892.08
625	13694.0	62653.10

In [65]:
```
#Germanyの購買金額の多いユーザ上位5名は
germany_data = df[df['Country'] == 'Germany']
germany_total_amounts = germany_data.groupby('CustomerID').apply(lambda x: (x['Quan
germany_total_amounts.sort_values(by='Totalamount', ascending=False).head(5)
```

Out[65]:

	CustomerID	Totalamount
3	12471.0	18740.92
53	12621.0	13612.07
9	12477.0	13117.01
42	12590.0	9861.38
80	12709.0	9294.10

In [66]:
```
#Franceの購買金額の多いユーザ上位5名は
France_data = df[df['Country'] == 'France']
```

```
France_total_amounts = France_data.groupby('CustomerID').apply(lambda x: (x['Quanti
France_total_amounts.sort_values(by='Totalamount', ascending=False).head(5)
```

Out[66]:

	CustomerID	Totalamount
80	12731.0	18793.41
52	12678.0	17588.26
55	12681.0	13677.59
56	12682.0	12288.22
20	12567.0	9114.94

In [67]:

```
#顧客数の上位3カ国ごとに購買数の多い商品上位5件を示す.
#上のセルのコードの実行により顧客数の上位3か国はUnited Kingdom、Germany 、Franceであ
#United Kingdomの購買数の多い商品上位5件は、
United_Kingdom_data = df[df['Country'] == 'United Kingdom']
United_Kingdom_shohin_amounts = United_Kingdom_data.groupby('StockCode')['Quantity']
United_Kingdom_shohin_amounts.sort_values(by='Quantity',ascending=False).head(5)
```

Out[67]:

	StockCode	Quantity
1068	22197	52928
2620	84077	48326
3655	85099B	43167
3666	85123A	36706
2733	84879	33519

In [68]:

```
#Germanyの購買数の多い商品上位5件は、
Germany_data = df[df['Country'] == 'Germany']
Germany_shohin_amounts = Germany_data.groupby('StockCode')['Quantity'].sum().reset_i
Germany_shohin_amounts.sort_values(by='Quantity',ascending=False).head(5)
```

Out[68]:

	StockCode	Quantity
578	22326	1218
5	15036	1164
1670	POST	1104
45	20719	1019
151	21212	1002

In [69]:

```
#Franceの購買数の多い商品上位5件は、
France_data = df[df['Country'] == 'France']
France_shohin_amounts = France_data.groupby('StockCode')['Quantity'].sum().reset_ind
France_shohin_amounts.sort_values(by='Quantity',ascending=False).head(5)
```

Out[69]:

	StockCode	Quantity
1026	23084	4023
646	22492	2196
303	21731	1314
96	21086	1272
1378	84879	1204

In [70]:
```
#日別の売上金額を曜日ごとに層別し，平均購買金額と平均購買数の多い曜日上位3件をそれぞ
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

# 曜日列を追加
df['DayOfWeek'] = df['InvoiceDate'].dt.day_name()

# TotalAmount列を追加
df['TotalAmount'] = df['Quantity'] * df['UnitPrice']

# 曜日ごとにグループ化し、平均購買金額と平均購買数を計算
result = df.groupby('DayOfWeek').agg(
    AveragePurchaseAmount=('TotalAmount', 'mean'),
    AveragePurchaseCount=('TotalAmount', 'count')
).reset_index()

# 平均購買金額で降順ソート
result_sorted_by_amount = result.sort_values(by='AveragePurchaseAmount', ascending=F

# 平均購買数で降順ソート
result_sorted_by_count = result.sort_values(by='AveragePurchaseCount', ascending=Fa

# 上位3件を取得
top_3_days_by_amount=result_sorted_by_amount.head(3)
top_3_days_by_count=result_sorted_by_count.head(3)

print("平均購買金額が多い曜日上位3件:")
print(top_3_days_by_amount)

print("¥n平均購買数が多い曜日上位3件:")
print(top_3_days_by_count)
```

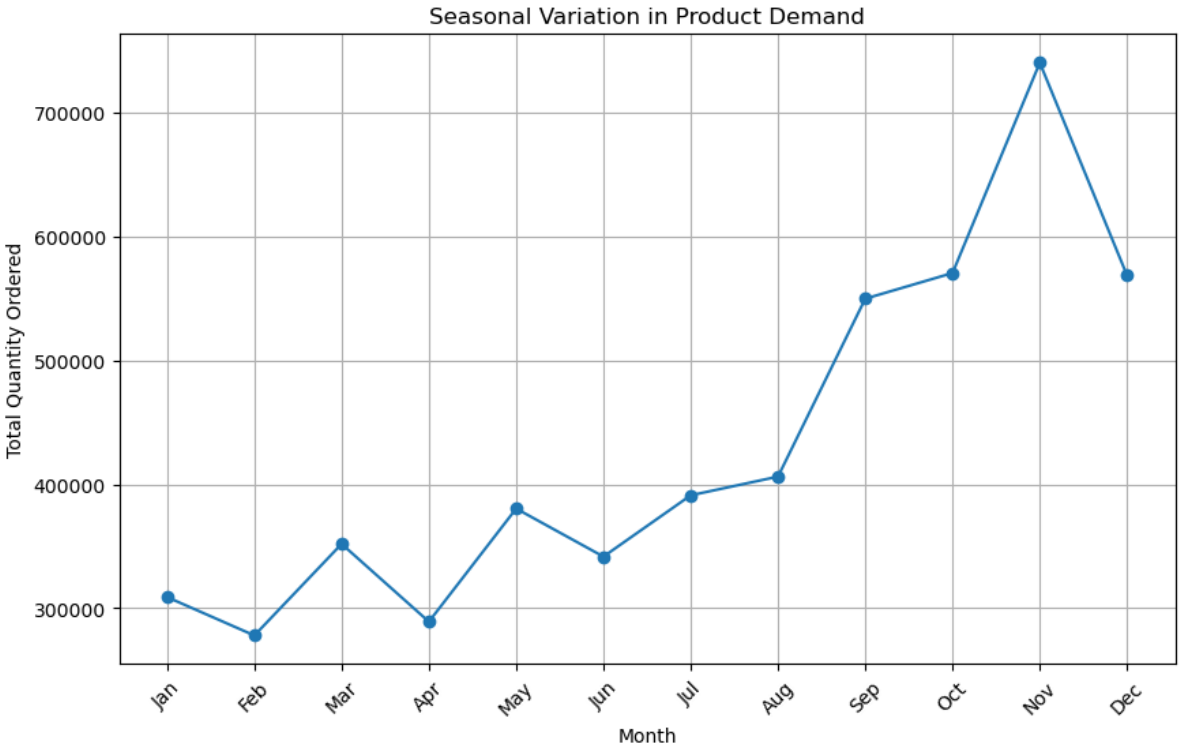
```
平均購買金額が多い曜日上位3件:
  DayOfWeek  AveragePurchaseAmount  AveragePurchaseCount
3  Thursday              20.340651                103857
4   Tuesday              19.312655                101808
0    Friday              18.743820                 82193

平均購買数が多い曜日上位3件:
  DayOfWeek  AveragePurchaseAmount  AveragePurchaseCount
3  Thursday              20.340651                103857
4   Tuesday              19.312655                101808
1    Monday              16.702689                 95111
```

In [71]:
```
#サービスの利用回数（購買回数）の多い顧客上位10名を抽出し，平均的に何日空けてリピート
# 顧客ごとに購買回数をカウント
customer_purchase_count = df.groupby('CustomerID')['InvoiceDate'].count()

# 購買回数で降順にソートし、上位10名の顧客を抽出
top_10_customers = customer_purchase_count.sort_values(ascending=False).head(10)

# 各顧客のリピート間隔を計算
repeat_intervals = []
```

```
for customer_id in top_10_customers.index:
    customer_dates = df[df['CustomerID'] == customer_id]['InvoiceDate']
    repeat_interval = customer_dates.diff().mean()
    repeat_intervals.append(repeat_interval)

# リピート間隔の平均値を計算
average_repeat_interval = pd.Series(repeat_intervals).mean()

print("利用回数が多い顧客上位10名:")
print(top_10_customers)

print("¥n平均的なリピート間隔（日数）:")
print(average_repeat_interval)
```

```
利用回数が多い顧客上位10名:
CustomerID
17841.0    7983
14911.0    5903
14096.0    5128
12748.0    4642
14606.0    2782
15311.0    2491
14646.0    2085
13089.0    1857
13263.0    1677
14298.0    1640
Name: InvoiceDate, dtype: int64

平均的なリピート間隔（日数）:
0 days 03:03:00.306802847
```

In [72]:
```
#仮説：クリスマスは商品の需要に影響を与えている。この仮説を検証

# 日付をDateTime型に変換
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

# 注文日時から月の情報を抽出
df['Month'] = df['InvoiceDate'].dt.month

# 各月ごとに商品の注文数を集計
monthly_order_counts = df.groupby('Month')['Quantity'].sum()

# 月ごとの商品の注文数を折れ線グラフで可視化
plt.figure(figsize=(10, 6))
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov
plt.plot(months, monthly_order_counts, marker='o', linestyle='-')
plt.xlabel('Month')
plt.ylabel('Total Quantity Ordered')
plt.title('Seasonal Variation in Product Demand')
plt.xticks(rotation=45)
plt.grid()
plt.show()

#検証結果
#12月は需要が他の月に比べて特別大きいわけではないが、11月が突出して大きいので、クリス
```


Seasonal Variation in Product Demand
Total Quantity Ordered
700000
600000
500000
400000
300000
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month