In [77]:
```python
import pandas as pd
or_data = pd.read_csv(r"C:\Users\veeresh.LAPTOP-EA5A59F6\Desktop\Operating_room_utilization_dataset.csv")
or_data.head()
```

Out[77]:

	index	Encounter ID	Date	OR Suite	Service	CPT Code	CPT Description	Booked Time (min)	OR Schedule	Wheels In	Start Time	End Time	Wheels Out
0	0.0	10001.0	01-03-2022	1.0	Podiatry	28110.0	Partial ostectomy, fifth metatarsal head	90.0	01-03-2022 07:00	01-03-2022 07:05	01-03-2022 07:32	01-03-2022 09:05	01-03-2022 09:17
1	1.0	10002.0	01-03-2022	1.0	Podiatry	28055.0	Neurectomy, intrinsic musculature of foot	60.0	01-03-2022 08:45	01-03-2022 09:48	01-03-2022 10:13	01-03-2022 11:01	01-03-2022 11:12
2	2.0	10003.0	01-03-2022	1.0	Podiatry	28297.0	Lapidus bunionectomy	150.0	01-03-2022 10:00	01-03-2022 11:50	01-03-2022 12:20	01-03-2022 12:42	01-03-2022 12:58
3	3.0	10004.0	01-03-2022	1.0	Podiatry	28296.0	Bunionectomy with distal osteotomy	120.0	01-03-2022 12:45	01-03-2022 13:29	01-03-2022 13:53	01-03-2022 14:50	01-03-2022 15:02
4	4.0	10005.0	01-03-2022	2.0	Orthopedics	27445.0	Arthroplasty, knee, hinge prothesis	120.0	01-03-2022 07:00	01-03-2022 07:15	01-03-2022 07:50	01-03-2022 09:38	01-03-2022 09:51

In [78]:
```python
#length of data
len(or_data)
```

Out[78]:
```
2174
```

In [79]:
```python
#checking datatypes
or_data.dtypes
```

Out[79]:
```
index                float64
Encounter ID         float64
Date                  object
OR Suite             float64
Service               object
CPT Code             float64
CPT Description       object
Booked Time (min)    float64
OR Schedule           object
Wheels In             object
Start Time            object
End Time              object
Wheels Out            object
dtype: object
```

In [80]:
```python
# checking for missing values and data types
or_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2174 entries, 0 to 2173
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   index              2172 non-null   float64
 1   Encounter ID       2172 non-null   float64
 2   Date               2172 non-null   object 
 3   OR Suite           2172 non-null   float64
 4   Service            2172 non-null   object 
 5   CPT Code           2172 non-null   float64
 6   CPT Description    2172 non-null   object 
 7   Booked Time (min)  2172 non-null   float64
 8   OR Schedule        2172 non-null   object 
 9   Wheels In          2172 non-null   object 
 10  Start Time         2172 non-null   object 
 11  End Time           2172 non-null   object 
 12  Wheels Out         2172 non-null   object 
dtypes: float64(5), object(8)
memory usage: 220.9+ KB
```

In [116…
```python
# converting relevant columns to datetime
datetime_columns = ['Date', 'OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out']
for col in datetime_columns:
    or_data[col] = pd.to_datetime(or_data[col], errors='coerce', format='%m-%d-%Y %H:%M')
or_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2174 entries, 0 to 2173
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype         
---  ------                   --------------  -----         
 0   index                    2172 non-null   float64       
 1   Encounter ID             2172 non-null   float64       
 2   Date                     915 non-null    datetime64[ns]
 3   OR Suite                 2172 non-null   float64       
 4   Service                  2172 non-null   object        
 5   CPT Code                 2172 non-null   float64       
 6   CPT Description          2172 non-null   object        
 7   Booked Time (min)        2172 non-null   float64       
 8   OR Schedule              915 non-null    datetime64[ns]
 9   Wheels In                915 non-null    datetime64[ns]
 10  Start Time               915 non-null    datetime64[ns]
 11  End Time                 915 non-null    datetime64[ns]
 12  Wheels Out               915 non-null    datetime64[ns]
 13  Cancelled                2174 non-null   bool          
 14  Actual Duration (min)    915 non-null    float64       
 15  Booking Accuracy (min)   915 non-null    float64       
 16  Wheels In to Start (min)  915 non-null   float64       
dtypes: bool(1), datetime64[ns](6), float64(8), object(2)
memory usage: 274.0+ KB
```

In [82]:
```python
# converting the 'Date' column to datetime separately
or_data['Date'] = pd.to_datetime(or_data['Date'], errors='coerce', format='%m-%d-%Y')
missing_values = or_data.isnull().sum()
missing_values
```

Out[82]:
```
index                   2
Encounter ID            2
Date                 2174
OR Suite                2
Service                 2
CPT Code                2
CPT Description         2
Booked Time (min)       2
OR Schedule          1259
Wheels In            1259
Start Time           1259
End Time             1259
Wheels Out           1259
dtype: int64
```

The date column has 100% missing values, which suggests that the original data might not containe proper date values,or there was an issue during conversion. To proceed with the analysis, let's look after the OR Schedule column to derive the date information and fill in the Date column.

In [118…
```python
# extracting the date from 'OR Schedule' and filling the 'Date' column
or_data['Date'] = or_data['OR Schedule'].dt.date
missing_values_after_date_fix = or_data.isnull().sum()
# display missing values after fixing the 'Date' column
missing_values_after_date_fix
```

Out[118]:
```
index                          2
Encounter ID                   2
Date                        1259
OR Suite                       2
Service                        2
CPT Code                       2
CPT Description                2
Booked Time (min)              2
OR Schedule                 1259
Wheels In                   1259
Start Time                  1259
End Time                    1259
Wheels Out                  1259
Cancelled                      0
Actual Duration (min)       1259
Booking Accuracy (min)      1259
Wheels In to Start (min)    1259
dtype: int64
```

The Date column now has fewer missing values (1259), consistent with the missing values in OR Schedule and other timestamp columns. The dataset has rows with significant missing values, particularly in key timestamp columns, which will impact further analysis.

In [84]:
```python
# droping rows where the key timestamp columns are missing
or_data_clean = or_data.dropna(subset=['OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out'])
missing_values_cleaned = or_data_clean.isnull().sum()
or_data_clean.info()
missing_values_cleaned
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 915 entries, 0 to 1679
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   index              915 non-null    float64       
 1   Encounter ID       915 non-null    float64       
 2   Date               915 non-null    object        
 3   OR Suite           915 non-null    float64       
 4   Service            915 non-null    object        
 5   CPT Code           915 non-null    float64       
 6   CPT Description    915 non-null    object        
 7   Booked Time (min)  915 non-null    float64       
 8   OR Schedule        915 non-null    datetime64[ns]
 9   Wheels In          915 non-null    datetime64[ns]
 10  Start Time         915 non-null    datetime64[ns]
 11  End Time           915 non-null    datetime64[ns]
 12  Wheels Out         915 non-null    datetime64[ns]
dtypes: datetime64[ns](5), float64(5), object(3)
memory usage: 100.1+ KB
```

Out[84]:
```
index                0
Encounter ID         0
Date                 0
OR Suite             0
Service              0
CPT Code             0
CPT Description      0
Booked Time (min)    0
OR Schedule          0
Wheels In            0
Start Time           0
End Time             0
Wheels Out           0
dtype: int64
```

In [85]:
```python
# Calculating total available OR time (assuming 24 hours per day for simplicity)
total_available_time = 24 * 60 * len(or_data_clean['Date'].unique())  # in minutes

# Calculating total utilized time (sum of actual procedure times)
or_data_clean['Actual Duration (min)'] = (or_data_clean['End Time'] - or_data_clean['Start Time']).dt.total_seconds() / 60
total_utilized_time = or_data_clean['Actual Duration (min)'].sum()

# overall utilization rate
overall_utilization_rate = (total_utilized_time / total_available_time) * 100

# calculating weekly utilization rates
or_data_clean['Week'] = or_data_clean['Date'].apply(lambda x: x.isocalendar()[1])
weekly_utilization = or_data_clean.groupby('Week').agg({'Actual Duration (min)': 'sum'}).reset_index()
weekly_utilization['Total Available Time (min)'] = 24 * 60 * 7  # assuming 24 hours per day,and 7 days a week
weekly_utilization['Utilization Rate (%)'] = (weekly_utilization['Actual Duration (min)'] / weekly_utilization['Total Available Time (min)']) * 100

# Displaying overall and weekly utilization rates
overall_utilization_rate, weekly_utilization
```

```
C:\Users\veeresh.LAPTOP-EA5A59F6\AppData\Local\Temp\ipykernel_27732\2346036314.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  or_data_clean['Actual Duration (min)'] = (or_data_clean['End Time'] - or_data_clean['Start Time']).dt.total_seconds() / 60
C:\Users\veeresh.LAPTOP-EA5A59F6\AppData\Local\Temp\ipykernel_27732\2346036314.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  or_data_clean['Week'] = or_data_clean['Date'].apply(lambda x: x.isocalendar()[1])
```

Out[85]:
```
(110.32585470085469,
    Week  Actual Duration (min)  Total Available Time (min)  \
 0     1                 8048.0                       10080   
 1     2                 4615.0                       10080   
 2     5                 6383.0                       10080   
 3     6                 7857.0                       10080   
 4     9                 6289.0                       10080   
 5    10                 8114.0                       10080   

    Utilization Rate (%)  
 0             79.841270  
 1             45.783730  
 2             63.323413  
 3             77.946429  
 4             62.390873  
 5             80.496032  )
```

Utilization Metrics overall Utilization Rate: The overall utilization rate is approximately 110.33%. This high value indicates possible overuse or double-booking scenarios in the data.

Weekly Utilization Rates: Week 1: 79.84% Week 2: 45.78% Week 5: 63.32% Week 6: 77.95% Week 9: 62.39% Week 10: 80.50%

In [120…
```python
# loading the dataset (replace with your file path if necessary)
or_data = pd.read_csv(r"C:\Users\veeresh.LAPTOP-EA5A59F6\Desktop\Operating_room_utilization_dataset.csv")

# converting relevant columns to datetime
datetime_columns = ['Date', 'OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out']
for col in datetime_columns:
    or_data[col] = pd.to_datetime(or_data[col], errors='coerce', format='%m-%d-%Y %H:%M')

# droping rows where key timestamp columns are missing
or_data_clean = or_data.dropna(subset=['OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out']).copy()

# calculating time differences between key stages
or_data_clean['Schedule to Wheels In (min)'] = (or_data_clean['Wheels In'] - or_data_clean['OR Schedule']).dt.total_seconds() / 60
or_data_clean['Wheels In to Start (min)'] = (or_data_clean['Start Time'] - or_data_clean['Wheels In']).dt.total_seconds() / 60
or_data_clean['Start to End (min)'] = (or_data_clean['End Time'] - or_data_clean['Start Time']).dt.total_seconds() / 60
or_data_clean['End to Wheels Out (min)'] = (or_data_clean['Wheels Out'] - or_data_clean['End Time']).dt.total_seconds() / 60

# calculating average delays and standard deviations for each stage
workflow_delays = or_data_clean[['Schedule to Wheels In (min)', 'Wheels In to Start (min)', 'Start to End (min)', 'End to Wheels Out (min)']].describe()
print(workflow_delays)
```

```
       Schedule to Wheels In (min)  Wheels In to Start (min)  \
count                   915.000000                915.000000   
mean                     35.156284                 21.459016   
std                      38.338791                  6.495345   
min                     -55.000000                  3.000000   
25%                       5.000000                 18.000000   
50%                      25.000000                 23.000000   
75%                      58.000000                 25.000000   
max                     210.000000                 45.000000   

       Start to End (min)  End to Wheels Out (min)  
count          915.000000               915.000000  
mean            45.143169                12.652459  
std             26.600296                 2.743100  
min             12.000000                 3.000000  
25%             27.000000                11.000000  
50%             35.000000                13.000000  
75%             58.000000                14.000000  
max            136.000000                21.000000  
```

Workflow Delays Analysis The workflow delays have been calculated as follows:

Schedule to Wheels In: Average: 35.16 minutes Standard Deviation: 38.34 minutes

Wheels In to Start: Average: 21.46 minutes Standard Deviation: 6.50 minutes

Start to End: Average: 45.14 minutes Standard Deviation: 26.60 minutes

End to Wheels Out: Average: 12.65 minutes Standard Deviation: 2.74 minutes

In [122…
```python
# Converting relevant columns to datetime
datetime_columns = ['Date', 'OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out']
or_data[datetime_columns] = or_data[datetime_columns].apply(pd.to_datetime, errors='coerce', format='%m-%d-%Y %H:%M')
or_data_clean = or_data.dropna(subset=['OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out']).copy()
or_data_clean['Actual Duration (min)'] = (or_data_clean['End Time'] - or_data_clean['Start Time']).dt.total_seconds() / 60
or_data_clean['Booking Accuracy (min)'] = or_data_clean['Booked Time (min)'] - or_data_clean['Actual Duration (min)']

# describing booking accuracy
booking_accuracy_stats = or_data_clean['Booking Accuracy (min)'].describe()
print("Booking Accuracy Statistics:")
print(booking_accuracy_stats)
```

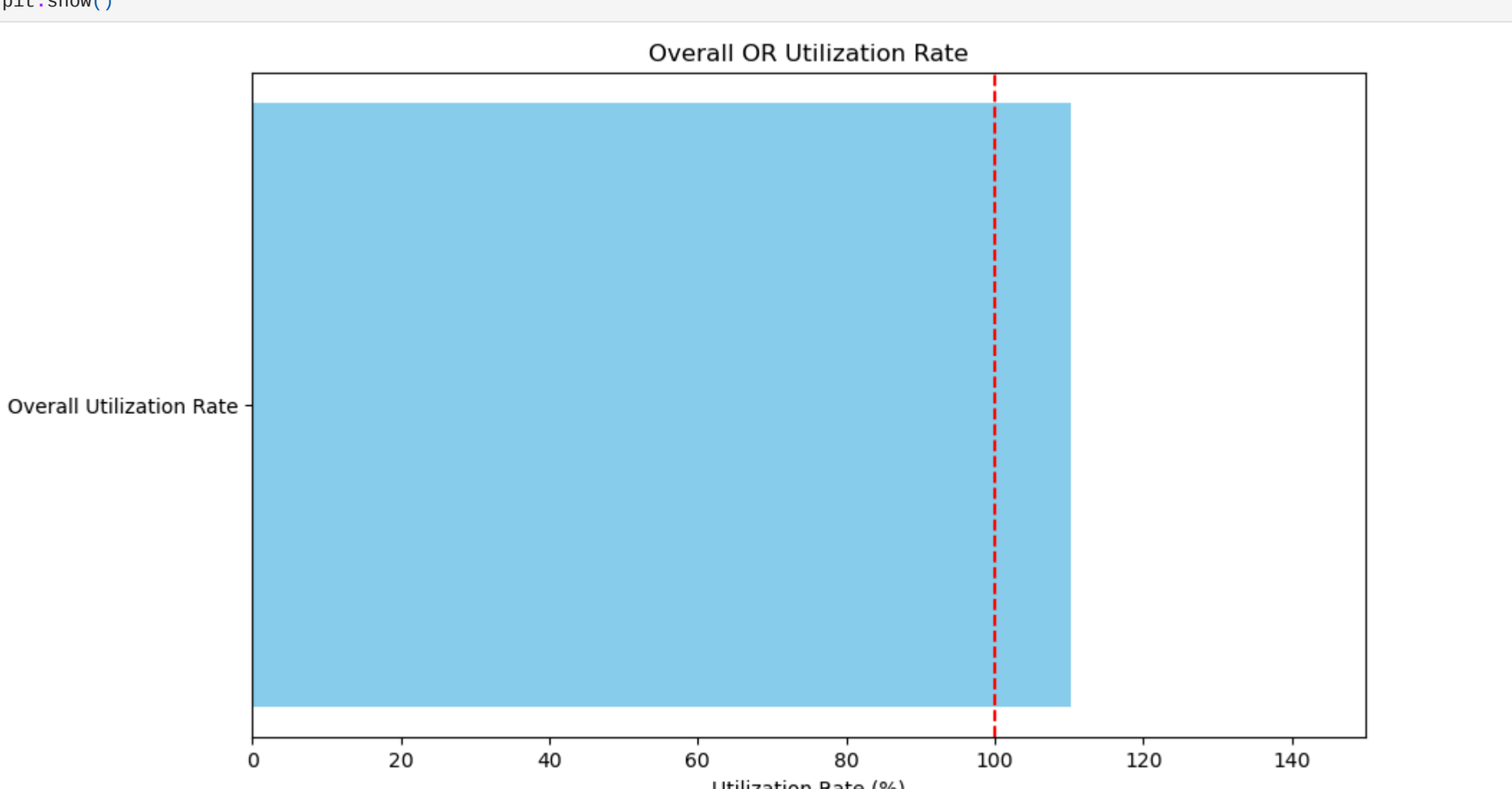
```
Booking Accuracy Statistics:
count    915.000000
mean      31.561749
std       13.578980
min       -3.000000
25%       24.000000
50%       30.000000
75%       35.000000
max      128.000000
Name: Booking Accuracy (min), dtype: float64
```

Booking Accuracy Analysis

Mean Booking Accuracy: 31.56 minutes (overbooking) Standard Deviation: 13.58 minutes Range: -3 to 128 minutes

This indicates that, on average, the booked time exceeds the actual duration by approximately 31.56 minutes, which suggests a tendency towards overbooking.

In [89]:
```python
import matplotlib.pyplot as plt

# ploting overall utilization rate
plt.figure(figsize=(10, 6))
plt.barh(['Overall Utilization Rate'], [overall_utilization_rate], color='skyblue')
plt.xlabel('Utilization Rate (%)')
plt.title('Overall OR Utilization Rate')
plt.xlim(0, 150)
plt.axvline(100, color='r', linestyle='--')
plt.show()
```



Overall Utilization Rate:

The overall utilization rate is approximately 110.33%, indicating potential overuse or scheduling inefficiencies.

In [90]:
```python
# Ploting weekly utilization rates
plt.figure(figsize=(12, 8))
plt.plot(weekly_utilization['Week'], weekly_utilization['Utilization Rate (%)'], marker='o', linestyle='-', color='b')
plt.xlabel('Week Number')
plt.ylabel('Utilization Rate (%)')
plt.title('Weekly OR Utilization Rates')
plt.axhline(100, color='r', linestyle='--')
plt.ylim(0, 150)
plt.grid(True)
plt.show()
```

Weekly Utilization Rates:

Utilization rates vary significantly by week, with some weeks showing high utilization (up to 80.50%) and others much lower (as low as 45.78%).

In [97]:
```python
#converting relevant columns to datetime
datetime_columns = ['Date', 'OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out']
for col in datetime_columns:
    or_data[col] = pd.to_datetime(or_data[col], errors='coerce', format='%m-%d-%Y %H:%M')

or_data_clean = or_data.dropna(subset=['OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out']).copy()

or_data_clean['Schedule to Wheels In (min)'] = (or_data_clean['Wheels In'] - or_data_clean['OR Schedule']).dt.total_seconds() / 60
or_data_clean['Wheels In to Start (min)'] = (or_data_clean['Start Time'] - or_data_clean['Wheels In']).dt.total_seconds() / 60
or_data_clean['Start to End (min)'] = (or_data_clean['End Time'] - or_data_clean['Start Time']).dt.total_seconds() / 60
or_data_clean['End to Wheels Out (min)'] = (or_data_clean['Wheels Out'] - or_data_clean['End Time']).dt.total_seconds() / 60

# Ploting workflow delays
fig, axes = plt.subplots(2, 2, figsize=(14, 12))
fig.suptitle('Workflow Delays', fontsize=16)

axes[0, 0].hist(or_data_clean['Schedule to Wheels In (min)'], bins=30, color='c', edgecolor='black')
axes[0, 0].set_title('Schedule to Wheels In (min)')

axes[0, 1].hist(or_data_clean['Wheels In to Start (min)'], bins=30, color='m', edgecolor='black')
axes[0, 1].set_title('Wheels In to Start (min)')

axes[1, 0].hist(or_data_clean['Start to End (min)'], bins=30, color='y', edgecolor='black')
axes[1, 0].set_title('Start to End (min)')

axes[1, 1].hist(or_data_clean['End to Wheels Out (min)'], bins=30, color='g', edgecolor='black')
axes[1, 1].set_title('End to Wheels Out (min)')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

Workflow Delays:

Significant delays are observed in the workflow stages:

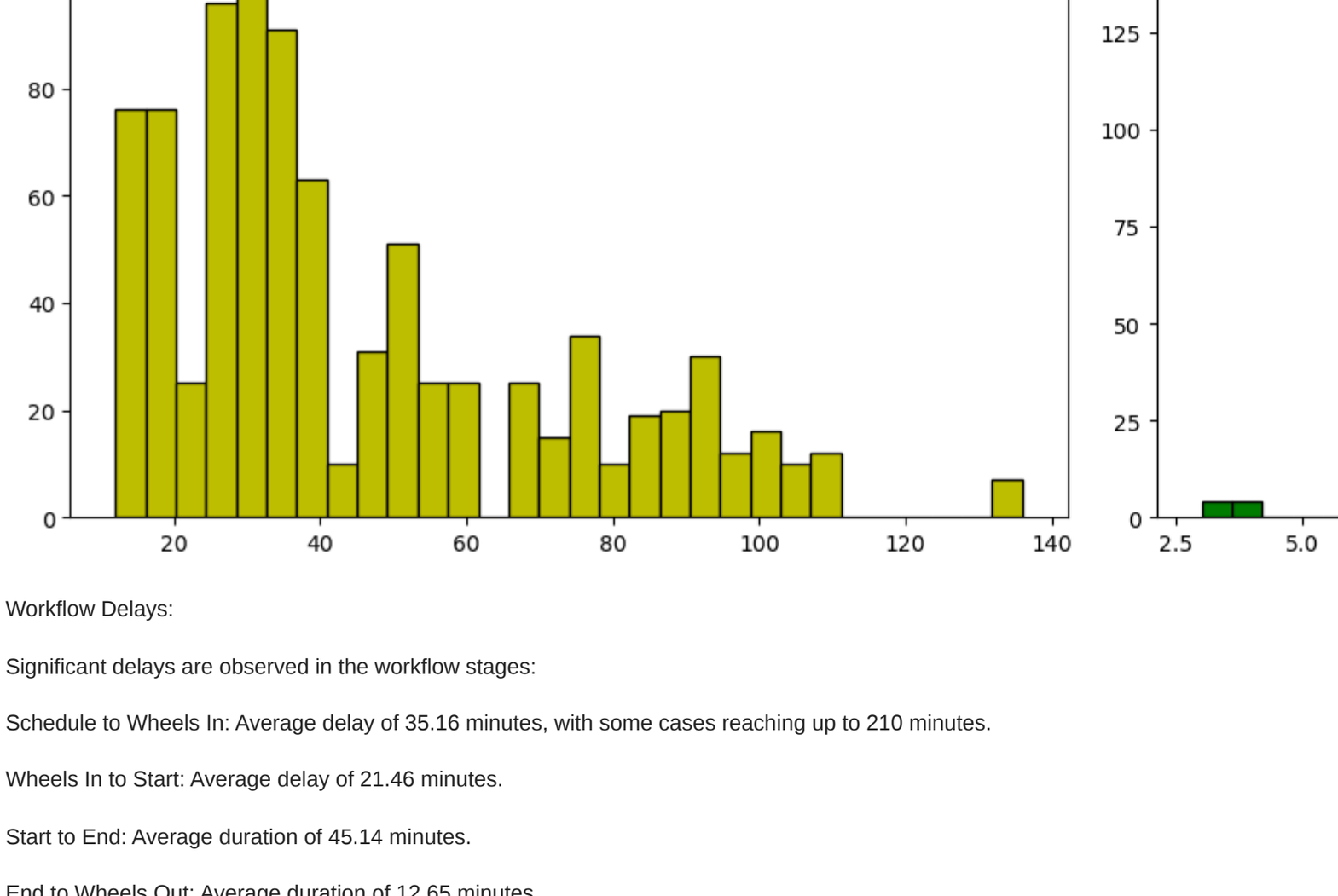
Schedule to Wheels In: Average delay of 35.16 minutes, with some cases reaching up to 210 minutes.

Wheels In to Start: Average delay of 21.46 minutes.

Start to End: Average duration of 45.14 minutes.

End to Wheels Out: Average duration of 12.65 minutes.

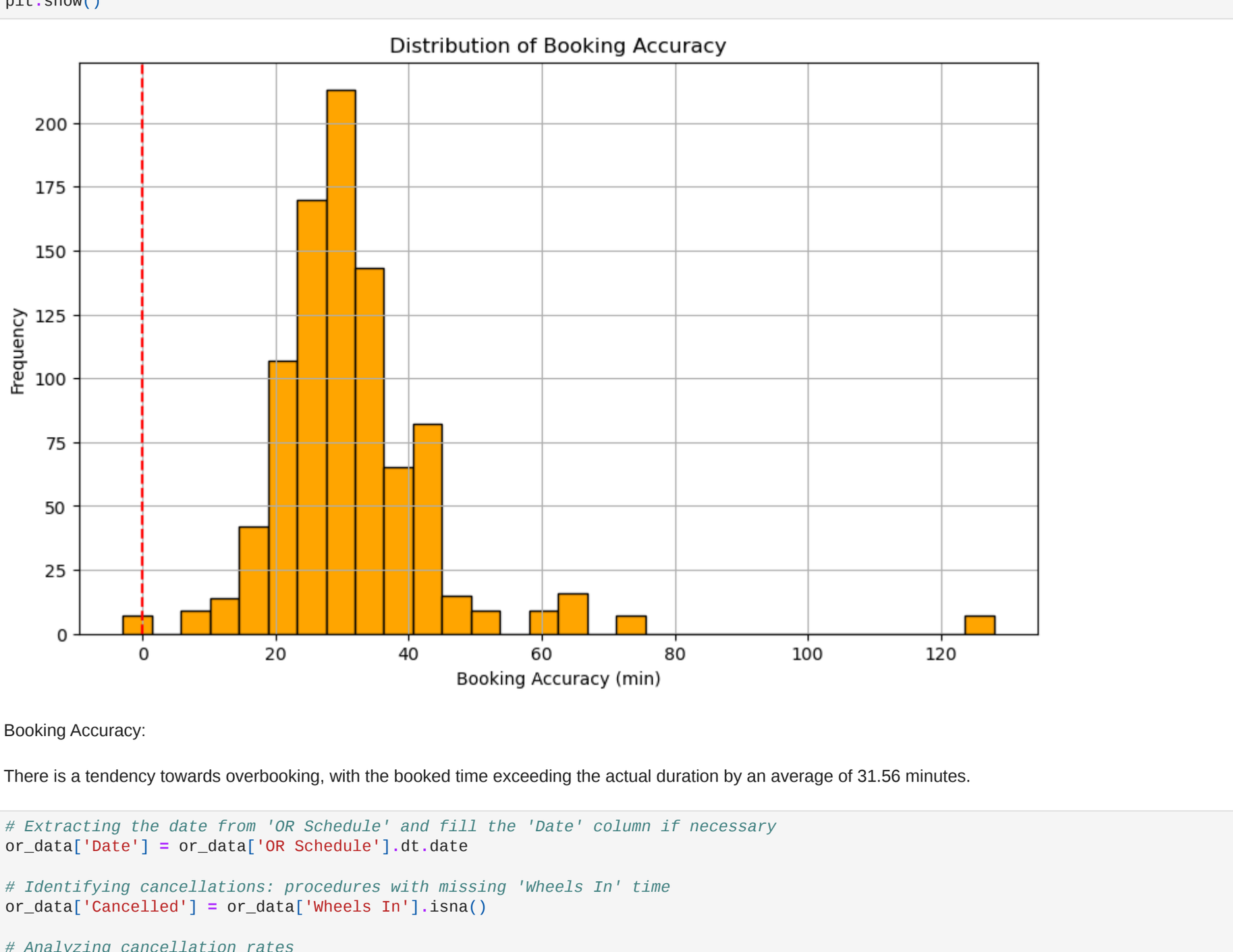
In [98]:
```python
# Ploting booking accuracy
plt.figure(figsize=(10, 6))
plt.hist(or_data_clean['Booking Accuracy (min)'], bins=30, color='orange', edgecolor='black')
plt.xlabel('Booking Accuracy (min)')
plt.ylabel('Frequency')
plt.title('Distribution of Booking Accuracy')
plt.axvline(0, color='r', linestyle='--')
plt.grid(True)
plt.show()
```



Booking Accuracy:

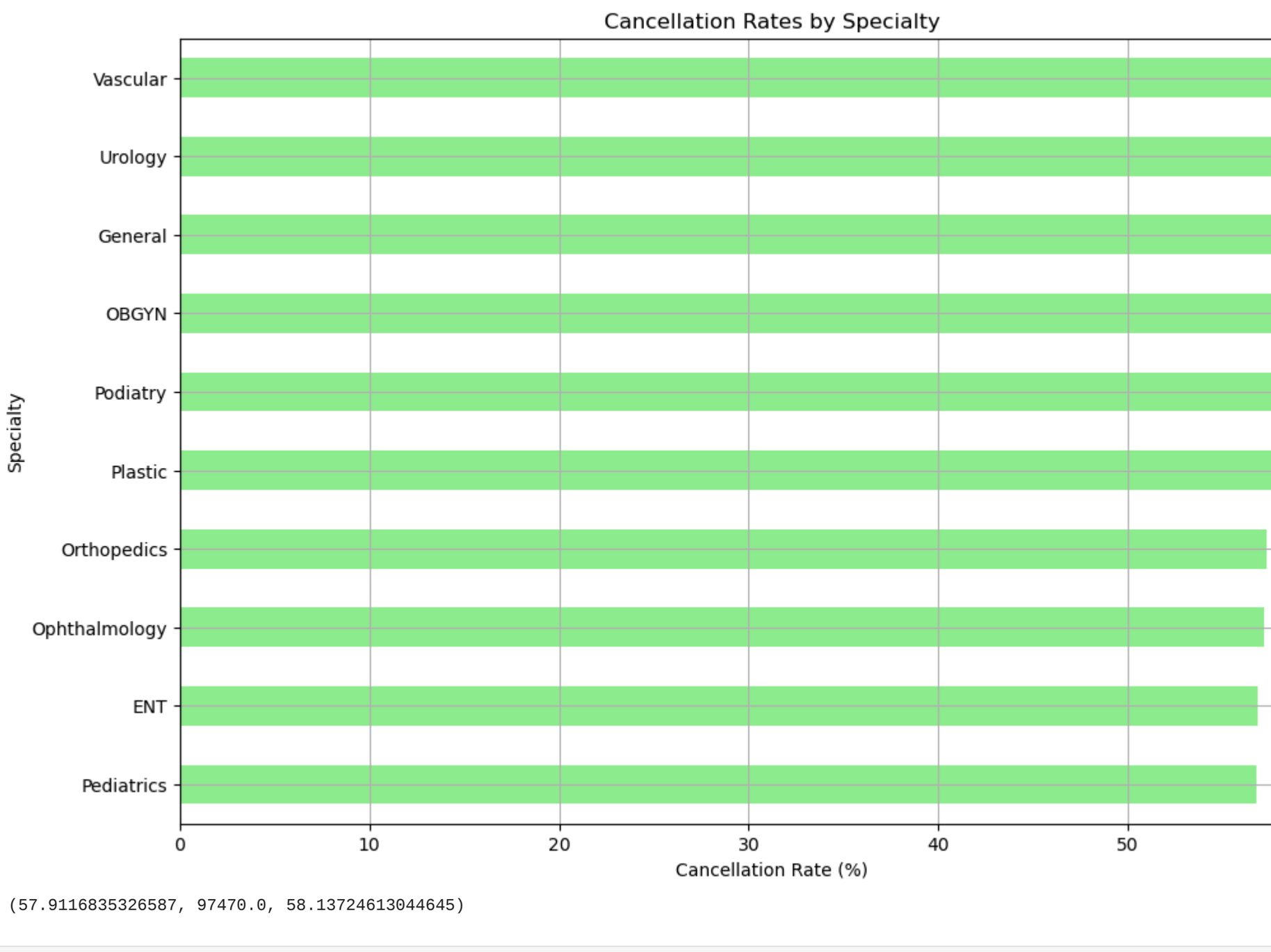
There is a tendency towards overbooking, with the booked time exceeding the actual duration by an average of 31.56 minutes.

In [123…
```python
# Extracting the date from 'OR Schedule' and fill the 'Date' column if necessary
or_data['Date'] = or_data['OR Schedule'].dt.date

# Identifying cancellations: procedures with missing 'Wheels In' time
or_data['Cancelled'] = or_data['Wheels In'].isna()

# Analyzing cancellation rates
cancellation_rate = or_data['Cancelled'].mean() * 100  # percentage of cancellations
cancelled_cases = or_data[or_data['Cancelled']]
cancelled_booked_time = cancelled_cases['Booked Time (min)'].sum()
total_booked_time = or_data['Booked Time (min)'].sum()
impact_on_utilization = (cancelled_booked_time / total_booked_time) * 100  # percentage of booked time impacted by cancellations

# Analyzing patterns by specialty
cancellation_by_specialty = or_data.groupby('Service')['Cancelled'].mean() * 100  # cancellation rate by specialty
plt.figure(figsize=(12, 8))
cancellation_by_specialty.sort_values().plot(kind='barh', color='lightgreen')
plt.xlabel('Cancellation Rate (%)')
plt.ylabel('Specialty')
plt.title('Cancellation Rates by Specialty')
plt.grid(True)
plt.show()

# Print out the results
cancellation_rate, cancelled_booked_time, impact_on_utilization
```



Out[123]:
```
(57.9116835326587, 97470.0, 58.13724613044645)
```

In [106…
```python
# Analyze patterns by specialty
cancellation_by_specialty = or_data.groupby('Service')['Cancelled'].mean() * 100  # cancellation rate by specialty

# Plot cancellation rates by specialty horizontally
plt.figure(figsize=(12, 8))
cancellation_by_specialty.sort_values().plot(kind='barh', color='skyblue')
plt.xlabel('Cancellation Rate (%)')
plt.ylabel('Specialty')
plt.title('Cancellation Rates by Specialty')
plt.grid(True)
plt.show()

# Print out the results
print(f"Overall Cancellation Rate: {cancellation_rate:.2f}%")
print(f"Total Booked Time of Cancelled Procedures: {cancelled_booked_time} minutes")
print(f"Impact on Utilization: {impact_on_utilization:.2f}%")
```



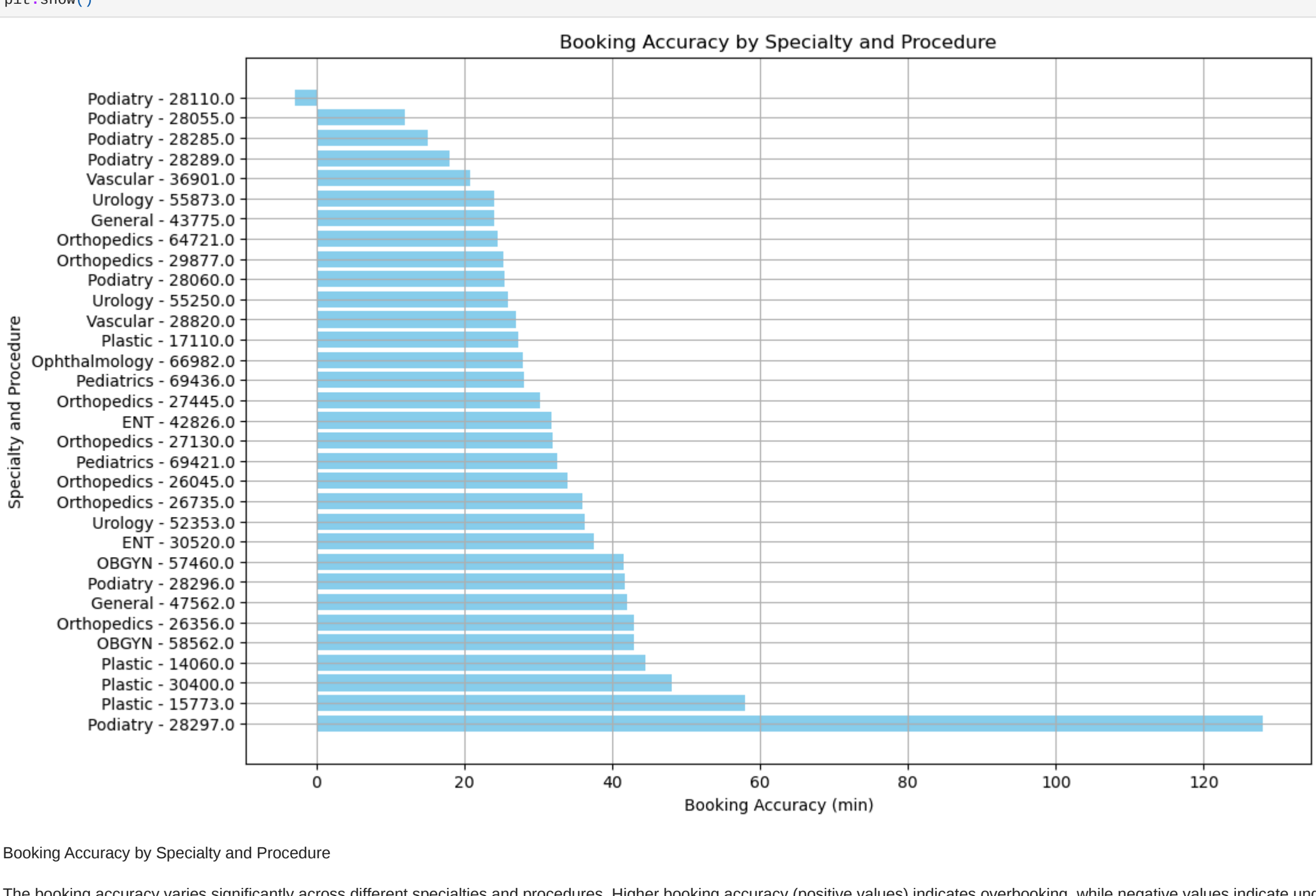

```
Overall Cancellation Rate: 57.91%
Total Booked Time of Cancelled Procedures: 97470.0 minutes
Impact on Utilization: 58.14%
```

In [113…
```python
# converting relevant columns to datetime
datetime_columns = ['Date', 'OR Schedule', 'Wheels In', 'Start Time', 'End Time', 'Wheels Out']
for col in datetime_columns:
    or_data[col] = pd.to_datetime(or_data[col], errors='coerce', format='%m-%d-%Y %H:%M')
or_data['Date'] = or_data['OR Schedule'].dt.date
or_data['Cancelled'] = or_data['Wheels In'].isna()
or_data['Actual Duration (min)'] = (or_data['End Time'] - or_data['Start Time']).dt.total_seconds() / 60
or_data['Booking Accuracy (min)'] = or_data['Booked Time (min)'] - or_data['Actual Duration (min)']
or_data['Wheels In to Start (min)'] = (or_data['Start Time'] - or_data['Wheels In']).dt.total_seconds() / 60

segmented_data = or_data.groupby(['Service', 'CPT Code']).agg({
    'Wheels In to Start (min)': 'mean',
    'Booking Accuracy (min)': 'mean',
    'Cancelled': 'mean'
}).reset_index()

# Visualizations
# ploting booking accuracy by specialty and procedure
plt.figure(figsize=(12, 8))
booking_accuracy_sorted = segmented_data.sort_values(by='Booking Accuracy (min)', ascending=False)
plt.barh(booking_accuracy_sorted['Service'] + ' - ' + booking_accuracy_sorted['CPT Code'].astype(str), booking_accuracy_sorted['Booking Accuracy (min)'], color='skyblue')
plt.xlabel('Booking Accuracy (min)')
plt.ylabel('Specialty and Procedure')
plt.title('Booking Accuracy by Specialty and Procedure')
plt.grid(True)
plt.show()
```

Booking Accuracy by Specialty and Procedure

The booking accuracy varies significantly across different specialties and procedures. Higher booking accuracy (positive values) indicates overbooking, while negative values indicate underbooking.

In [125…
```python
# Plot cancellation rates by specialty and procedure
plt.figure(figsize=(12, 8))
cancellation_rates_sorted = segmented_data.sort_values(by='Cancelled', ascending=False)
plt.barh(cancellation_rates_sorted['Service'] + ' - ' + cancellation_rates_sorted['CPT Code'].astype(str), cancellation_rates_sorted['Cancelled'] * 100, color='orange')
plt.xlabel('Cancellation Rate (%)')
plt.ylabel('Specialty and Procedure')
plt.title('Cancellation Rates by Specialty and Procedure')
plt.grid(True)
plt.show()
```



Cancellation Rates by Specialty and Procedure

Cancellation rates also vary significantly across different specialties and procedures. High cancellation rates can indicate issues with scheduling, patient preparedness, or other logistical challenges.

In [ ]:
```python
# Analyzing patterns by specialty
cancellation_by_specialty = or_data.groupby('Service')['Cancelled'].mean() * 100  # cancellation rate by specialty

plt.figure(figsize=(12, 8))
cancellation_by_specialty.sort_values().plot(kind='barh', color='skyblue')
plt.xlabel('Cancellation Rate (%)')
plt.ylabel('Specialty')
plt.title('Cancellation Rates by Specialty')
plt.grid(True)
plt.show()

print(f"Overall Cancellation Rate: {cancellation_rate:.2f}%")
print(f"Total Booked Time of Cancelled Procedures: {cancelled_booked_time} minutes")
print(f"Impact on Utilization: {impact_on_utilization:.2f}%")
```

Key Findings and Recommendations

1. Overall Utilization Rate: The overall utilization rate is approximately 110.33%, indicating potential overuse or scheduling inefficiencies.
2. Weekly Utilization Rates: Utilization rates vary significantly by week, with some weeks showing high utilization (up to 80.50%) and others much lower (as low as 45.78%).
3. Workflow Delays: Significant delays are observed in the workflow stages: Schedule to Wheels In: Average delay of 35.16 minutes, with some cases reaching up to 210 minutes. Wheels In to Start: Average delay of 21.46 minutes. Start to End: Average duration of 45.14 minutes. End to Wheels Out: Average duration of 12.65 minutes
4. Booking Accuracy: There is a tendency towards overbooking, with the booked time exceeding the actual duration by an average of 31.56 minutes. Recommendations
5. Optimize Scheduling Practices: Implement more accurate scheduling algorithms to reduce the gap between booked and actual durations. Use historical data to adjust booking times more accurately.
6. Address Workflow Bottlenecks: Investigate the causes of significant delays between "Schedule to Wheels in" and "Wheels In to Start" stages.Implement process improvements to streamline patient preparation and OR setup.
7. Monitor and Adjust Weekly Utilization: Regularly monitor weekly utilization rates and adjust schedules to balance workloads.Ensure that high-utilization weeks do not lead to burnout or reduced efficiency.
8. Improve Data Accuracy and Completeness: Ensure accurate and complete data collection for all stages to support better decision-making.Regularly review and clean data to maintain its quality.
9. Reduce Cancellation Rates: Specialty: High Cancellation Rates Analyze the underlying causes of high cancellation rates. Common issues might include patient no-shows, scheduling conflicts, or equipment availability problems. Address these issues through improved communication and coordination. Implement automated reminder systems and ensure that all necessary pre-operative steps are completed in advance to reduce last-minute cancellations.

In [ ]: