EDA Project: Birmingham Parking Dataset

Introduction

The "BhamParking" dataset, which is a complete collection of parking data for Birmingham, is the subject of this exploratory data analysis (EDA) project. The dataset includes a lot of different variables that can be used to learn about parking availability, usage trends, and maybe even how well parking spaces are allocated. The reason for this project is to find the deeper patterns and trends in Birmingham's parking, which can help city planners, parking authorities, and lawmakers make better decisions.

Overview about the dataset

In [1]:
```
# Importing required libraries
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
#magic function for displaying plot in the notebook
%matplotlib inline
```

In [2]:
```
# Read the excel file
df = pd.read_excel("BhamParking.xlsx")
```

In [3]:
```
# Display the first five rows of the dataset
df.head()
```

Out[3]:

	SystemCodeNumber	Capacity	Occupancy	per_usage	per_occupancy	year	month	da
0	BHMBCCMKT01	577	61.0	10.57	0 - 25	2016	Oct	Tu
1	BHMBCCMKT01	577	64.0	11.09	0 - 25	2016	Oct	Tu
2	BHMBCCMKT01	577	80.0	13.86	0 - 25	2016	Oct	Tu
3	BHMBCCMKT01	577	107.0	18.54	0 - 25	2016	Oct	Tu
4	BHMBCCMKT01	577	150.0	26.00	25 - 50	2016	Oct	Tu

In [4]:
```
#Get the dimension of the dataset
df.shape
```

Out[4]:
```
(35332, 11)
```

In [5]:
```
# Generate descriptive statistics for the dataset
df.describe()
```

Out[5]:

	Capacity	Occupancy	per_usage	year	hour
count	35332.000000	35313.000000	35325.000000	35332.0	35332.000000
mean	1406.159968	642.681222	48.795447	2016.0	6.707687
std	1182.388089	659.719257	26.724268	0.0	3.913969
min	220.000000	0.000000	0.000000	2016.0	1.000000
25%	577.000000	209.000000	25.380000	2016.0	3.000000
50%	863.000000	448.000000	46.670000	2016.0	8.000000
75%	2009.000000	796.000000	71.100000	2016.0	10.000000
max	4675.000000	4327.000000	100.000000	2016.0	12.000000

From the table we can conclude that:

- There is a wide range of capacity since facilities vary in size indicating a diverse set of parking options.
- Less than half of the parking spots are used on average, which could mean there are too many places despite their capacity to hold a lot of cars.
- Occupancy rates are more than 71% 25% of the time, which indicates that there are times or places that are nearly full.
- While the mean shows that most people are out and about first thing in the morning, the middle ground and upper half show that the morning rush is more typical. Because of outliers or continuous use at night, this points to the possibility of abnormal usage patterns.

Dealing with Missing Values

In [6]:
```
#Summary of the datasets
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35332 entries, 0 to 35331
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   SystemCodeNumber  35332 non-null  object
 1   Capacity          35332 non-null  int64
 2   Occupancy         35313 non-null  float64
 3   per_usage         35325 non-null  float64
 4   per_occupancy     35313 non-null  object
 5   year              35332 non-null  int64
 6   month             35332 non-null  object
 7   day               35331 non-null  object
 8   WorkingDay        35329 non-null  object
 9   hour              35332 non-null  int64
 10  period            35331 non-null  object
dtypes: float64(2), int64(3), object(6)
memory usage: 3.0+ MB
```

From the output of the ***info()*** function, we can see that there are some columns with missing data, this includes:

- **Occupancy**: 19 missing values
- **per_occupancy**: 19 missing values
- **per_usage**: 7 missing values
- **day**: 1 missing values
- **WorkingDay**: 3 missing values
- **period**: 1 missing values

We will now handle the missing data in each of these columns:

Occupancy: The value of occupancy can be calculated by using the value of column *per_usage* divided by 100 to gain the percentage usage of the car park, then we will multiply it by the value of the *Capacity* column.

per_usage: The value of per_usage can be calculated by using the value of *Occupancy* divided by the value of *Capacity* then multiply by 100.

It is impossible to find the values for the row where both values are null. So what we should do here is drop those columns.

In [7]:
```
# This code will identify the rows where both value of 'per_uasge' and 'Occupancy'

df = df.dropna(subset=['per_usage', 'Occupancy'], how='all')
```

Now we will use the function that we have mentioned above to calculate the value for the null of columns **'Occupancy'**, and **'per_usage'**

In [8]:
```
'''These two code will first identify the null value in the two columns, then it wi
that we have mentioned above'''
df.loc[pd.isna(df['per_usage']), 'per_usage'] = (df['Occupancy'] / df['Capacity'])
```

```
df.loc[pd.isna(df['Occupancy']), 'Occupancy'] = (df['per_usage'] * df['Capacity'])

# Round up the result the we receive from the above code
df['Occupancy'].round()
df['per_usage'].round(2)
df.head()
```

Out[8]:

	SystemCodeNumber	Capacity	Occupancy	per_usage	per_occupancy	year	month	da
0	BHMBCCMKT01	577	61.0	10.57	0 - 25	2016	Oct	Tu
1	BHMBCCMKT01	577	64.0	11.09	0 - 25	2016	Oct	Tu
2	BHMBCCMKT01	577	80.0	13.86	0 - 25	2016	Oct	Tu
3	BHMBCCMKT01	577	107.0	18.54	0 - 25	2016	Oct	Tu
4	BHMBCCMKT01	577	150.0	26.00	25 - 50	2016	Oct	Tu

In [9]:
```
# Count the number of null value in a columns
print(f"The number of null value in the 'Occupancy' column is: {df['Occupancy'].isn
print(f"The number of null value in the 'per_usage' column is: {df['per_usage'].isn
```

```
The number of null value in the 'Occupancy' column is: 0
The number of null value in the 'per_usage' column is: 0
```

per_occupancy: This column value is a categorical value and it depends on the value of the *per_usage* column. Therefore we need to define a function to link the value of *per_usage* to its suitable category

In [10]:
```
# Define the function to categorize the value of column 'per_usage' to column 'per_
def func(x):
    if x['per_usage'] > 75:
        x['per_occupancy'] = '75-100'
    elif x['per_usage'] > 50 and x['per_usage'] < 75:
        x['per_occupancy'] = '50-75'
    elif x['per_usage'] > 25 and x['per_usage'] < 50:
        x['per_occupancy'] = '25-50'
    else:
        x['per_occupancy'] = '0-25'
    return x
```

In [11]:
```
# Apply the function to the dataset
df = df.apply(func, axis=1)
```

In [12]:
```
# Count the number of null value in a columns
print(f"The number of null value in the 'per_occupancy' column is: {df['per_occupan
```

```
The number of null value in the 'per_occupancy' column is: 0
```

day: After examining the position of the missing value in the day column, we can agree that the missing value can be added based on the values adjacent to it, which we can use *__ffill__*.

This function will insert the value of the previously available observations:

26039	Others-CCCPS105a	2009	1425	70.93	50 - 75	2016	Dec	Mon	Yes	2	PM
26040	Others-CCCPS105a	2009	1367	68.04	50 - 75	2016	Dec	Mon	Yes	2	PM
26041	Others-CCCPS105a	2009	1314	65.41	50 - 75	2016	Dec		Yes	3	PM
26042	Others-CCCPS105a	2009	1272	63.32	50 - 75	2016	Dec	Mon	Yes	3	PM
26043	Others-CCCPS105a	2009	1196	59.53	50 - 75	2016	Dec	Mon	Yes	4	PM
26044	Others-CCCPS105a	2009	854	42.51	25 - 50	2016	Dec	Tue	Yes	8	AM
26045	Others-CCCPS105a	2009	913	45.45	25 - 50	2016	Dec	Tue	Yes	8	AM
26046	Others-CCCPS105a	2009	1023	50.92	50 - 75	2016	Dec	Tue	Yes	9	AM

In [13]:
```
#Use the fillna() function with the ffill method to fill the missing value with the
df['day'] = df['day'].fillna(method='ffill')
```

In [14]:
```
# Count the number of null value in a columns
print(f"The number of null value in the 'day' column is: {df['day'].isnull().sum()}
```

```
The number of null value in the 'day' column is: 0
```

WorkingDay: The value of this column can be derived from the value of the 'day' column. Therefore we need to define a function to link the value of 'day' to its suitable category.

In [15]:
```
def weekday(x):
    # Check if the 'day' value in x is either 'Sat' or 'Sun'
    if x['day'] == 'Sat' or x['day'] == 'Sun':
        # If it's Saturday or Sunday, set the 'WorkingDay' value to 'No'
        x['WorkingDay'] = 'No'
    else:
        # If it's any other day, set the 'WorkingDay' value to 'Yes'
        x['WorkingDay'] = 'Yes'
    # Return the modified dictionary
    return x
```

In [16]:
```
# Apply the function to the dataset
df = df.apply(weekday, axis=1)
```

In [17]:
```
# Count the number of null value in a columns
print(f"The number of null value in the 'WorkingDay' column is: {df['WorkingDay'].i
```

```
The number of null value in the 'WorkingDay' column is: 0
```

period: The investigation of the location of the missing data tells us that the value of the missing data should be 'PM'

Others-CCCPS105a	2009	1341	66.75	50 - 75	2016	Dec	Tue	Yes	11	AM
Others-CCCPS105a	2009	1401	69.74	50 - 75	2016	Dec	Tue	Yes	12	PM
Others-CCCPS105a	2009	1462	72.77	50 - 75	2016	Dec	Tue	Yes	12	PM
Others-CCCPS105a	2009	1497	74.51	50 - 75	2016	Dec	Tue	Yes	1	PM
Others-CCCPS105a	2009	1518	75.56	75-100	2016	Dec	Tue	Yes	1	
Others-CCCPS105a	2009	1418	70.58	50 - 75	2016	Dec	Tue	Yes	2	PM
Others-CCCPS105a	2009	1357	67.55	50 - 75	2016	Dec	Tue	Yes	2	PM

In [18]:
```
# Fill the missing value in the 'period' column with 'PM'
df['period'] = df['period'].fillna('PM')
```

In [19]:
```
# Count the number of null value in a columns
print(f"The number of null in the 'period' column is: {df['period'].isnull().sum()}
```

```
The number of null in the 'period' column is: 0
```

Now let's again look at the whole dataset to see if there are any missing values left:

In [20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 35327 entries, 0 to 35331
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   SystemCodeNumber  35327 non-null  object 
 1   Capacity          35327 non-null  int64  
 2   Occupancy         35327 non-null  float64
 3   per_usage         35327 non-null  float64
 4   per_occupancy     35327 non-null  object 
 5   year              35327 non-null  int64  
 6   month             35327 non-null  object 
 7   day               35327 non-null  object 
 8   WorkingDay        35327 non-null  object 
 9   hour              35327 non-null  int64  
 10  period            35327 non-null  object 
dtypes: float64(2), int64(3), object(6)
memory usage: 4.2+ MB
```

As we can see in the Non-Null Count Column of the output, all 10 columns so an equal result of 35327 which means that there are no null values left in our datasets

Data Visualization

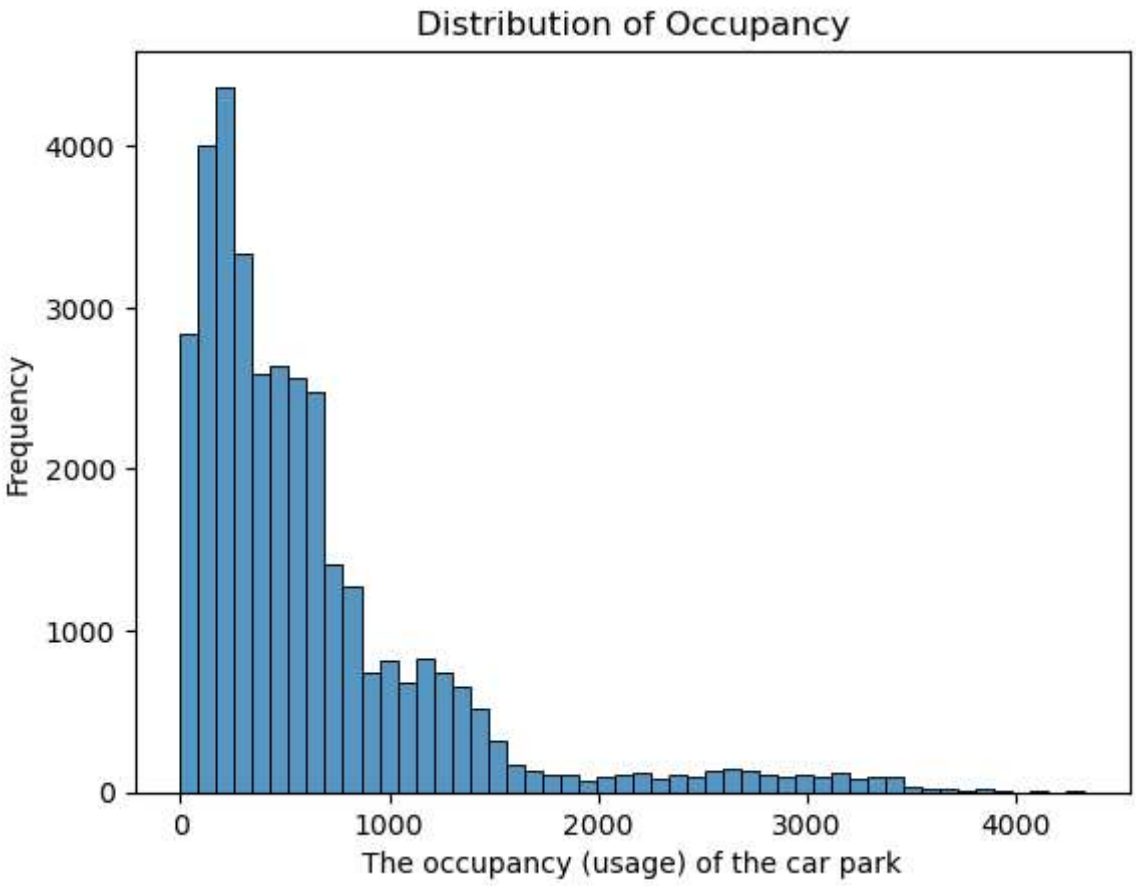
A. The distribution of individual continuous variables

Let's first visualise the distribution of *Occupancy* and *per_usage* of the dataset by using a histogram

Occupancy:

In [21]:
```
# Plot a histogram for 'Occupancy' with 50 bins.
sns.histplot(df['Occupancy'], bins = 50)
plt.title('Distribution of Occupancy')
plt.xlabel('The occupancy (usage) of the car park')
plt.ylabel('Frequency')
```

Out[21]:
```
Text(0, 0.5, 'Frequency')
```

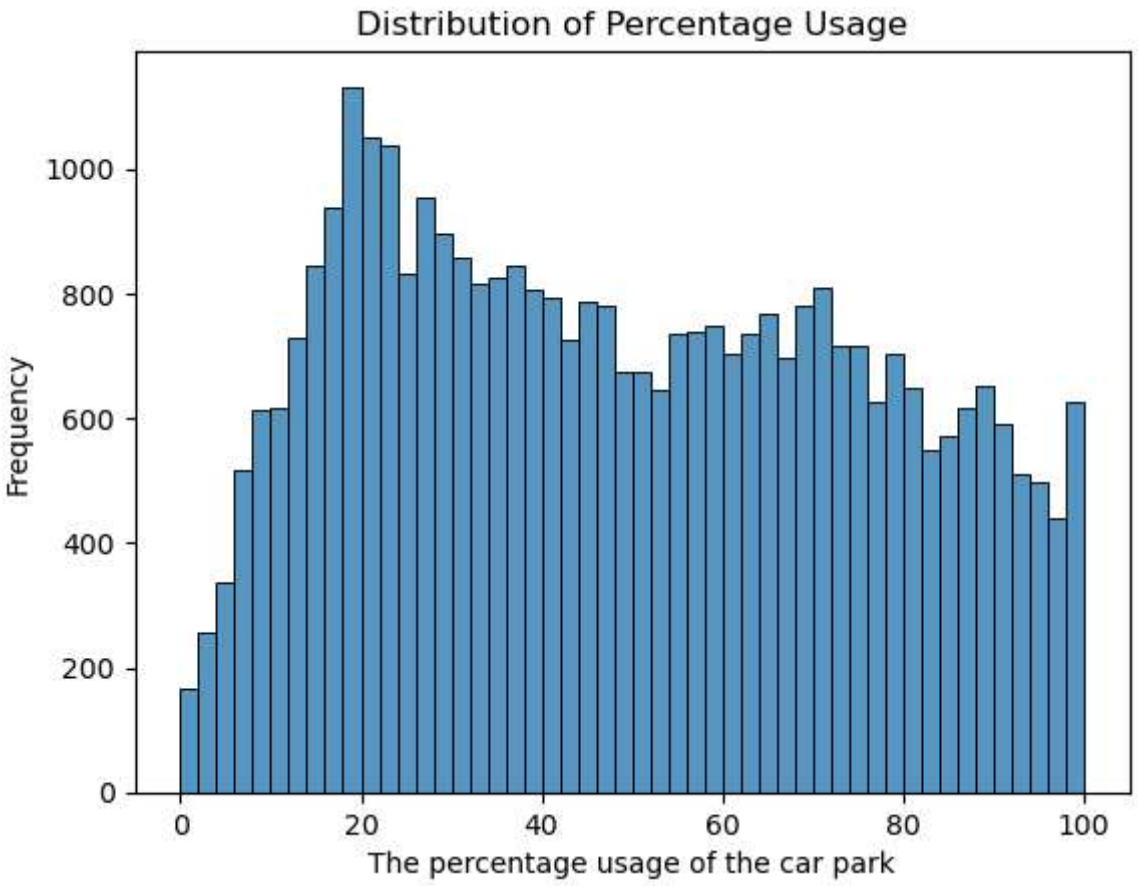


When you look at the parking lot occupancy distribution chart, you can see it is skewed to the left. Most of the occupancy data is concentrated at the lower left end of the chart. This shows us that most of the time, parking lots only hold a small to medium number of vehicles.

per_usage

In [22]:
```
# Plot a histogram for 'per_usage' with 50 bins.
sns.histplot(df['per_usage'], bins=50)
plt.title('Distribution of Percentage Usage')
plt.xlabel('The percentage usage of the car park')
plt.ylabel('Frequency')
```

Out[22]:
```
Text(0, 0.5, 'Frequency')
```



The histogram indicates that parking lots are typically moderately utilized, with common percentage usage around 20–30% and 60–70%. Extremely low or high percentage usage are uncommon, indicating parking lots are neither consistently underused nor regularly at full capacity.

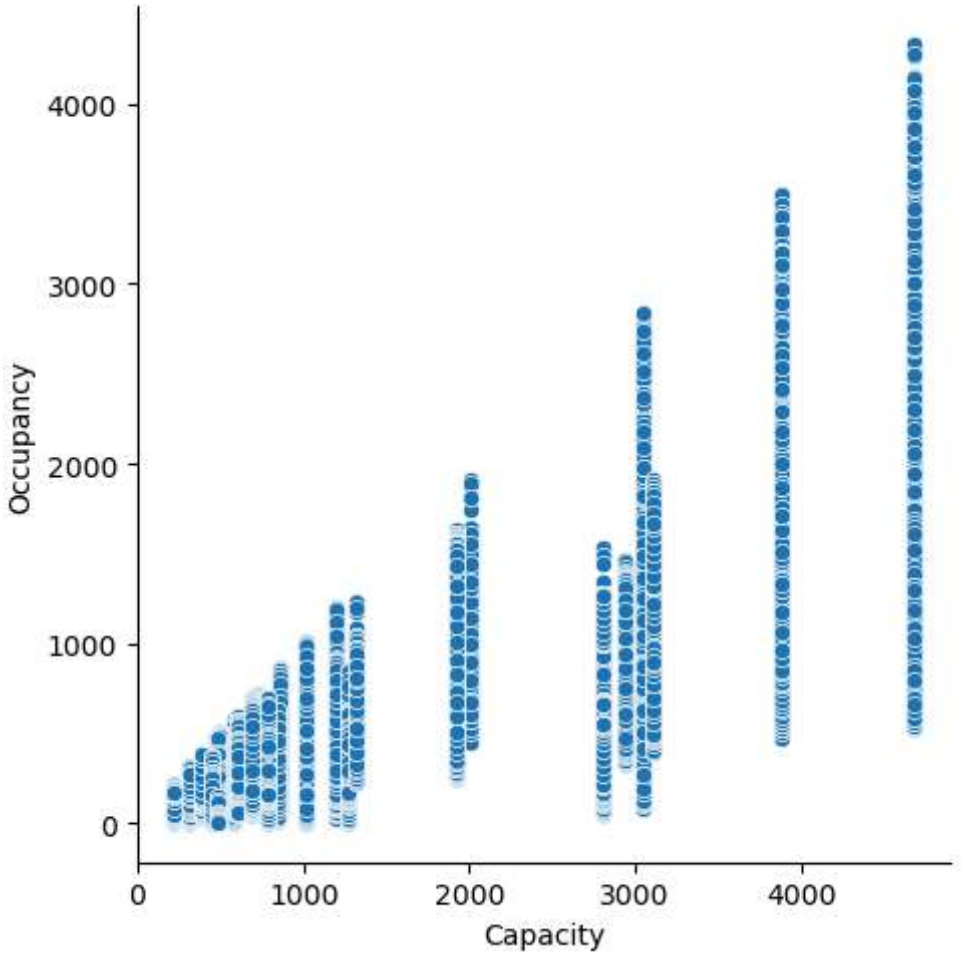
B. The relationship of a pair of continuous variables

Let's visualise the relationship of *Occupancy* and *Capacity* of the dataset by using a scatter plot

In [23]:
```
sns.relplot(x = 'Capacity', y = 'Occupancy', data = df);
plt.xlabel("Capacity"); plt.ylabel("Occupancy")
```

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

Out[23]:
```
Text(-3.9305555555555554, 0.5, 'Occupancy')
```



The scatter plot indicates:

- Within capacity groups, usage rates range from almost full to underused.
- Parking lots with more than 3000 spaces are never fully used.
- There is no clear link between the number of spaces in a lot and the number of times it is used, which means that parking lots are not being used efficiently.

C. The association between a categorical variable and a continuous variable.

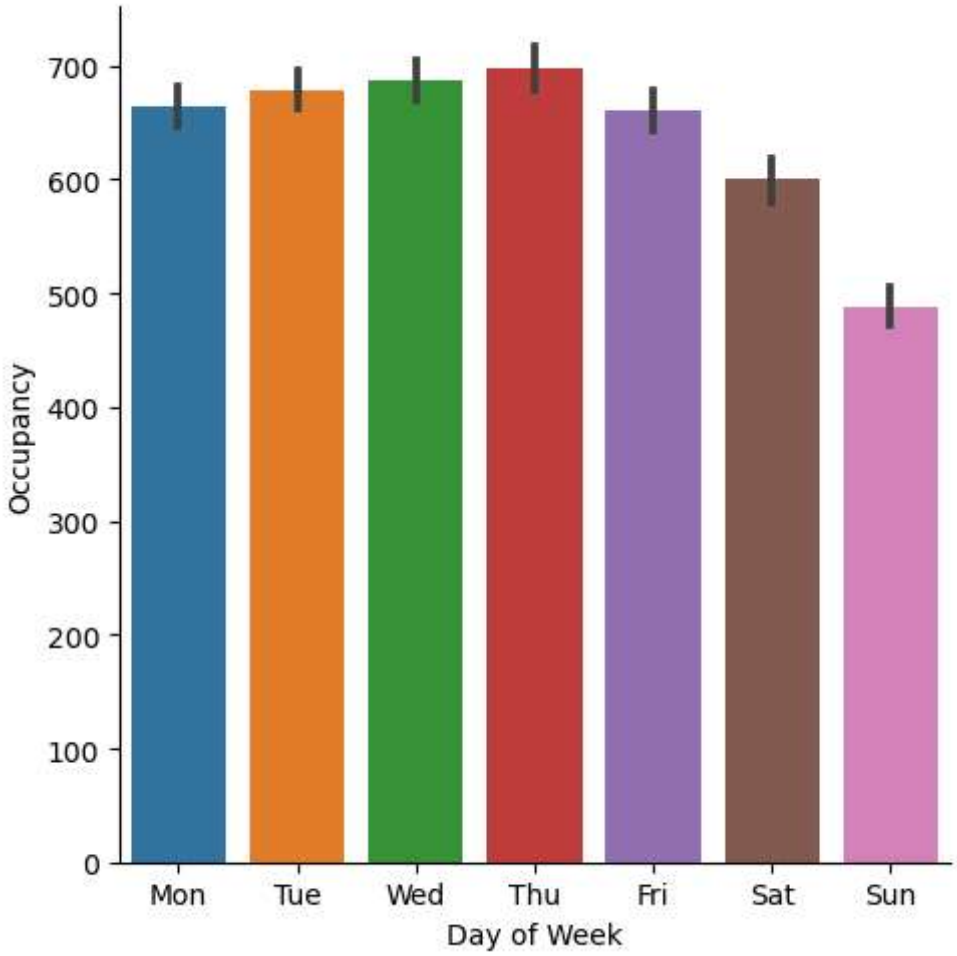
Let's visualize the relation between *day*, and *Occupancy* by using a bar chart

In [24]:
```
# Set the order of the days
day_order = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

# Create the bar plot with the information about occupancy in each day of a week
sns.catplot(x='day', y='Occupancy', data=df, kind='bar', order=day_order)
plt.xlabel("Day of Week"); plt.ylabel("Occupancy")
```

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

Out[24]:
```
Text(4.944444444444445, 0.5, 'Occupancy')
```



The bar chart shows us the trend in parking usage over a week, with the number of vehicles increasing steadily from Monday and peaking on Thursday. Saturday and Sunday show a decrease in the number of vehicles in the parking. This shows that the demand for parking on weekends is lower than the demand on weekdays.

C.The relationship between more than two variables

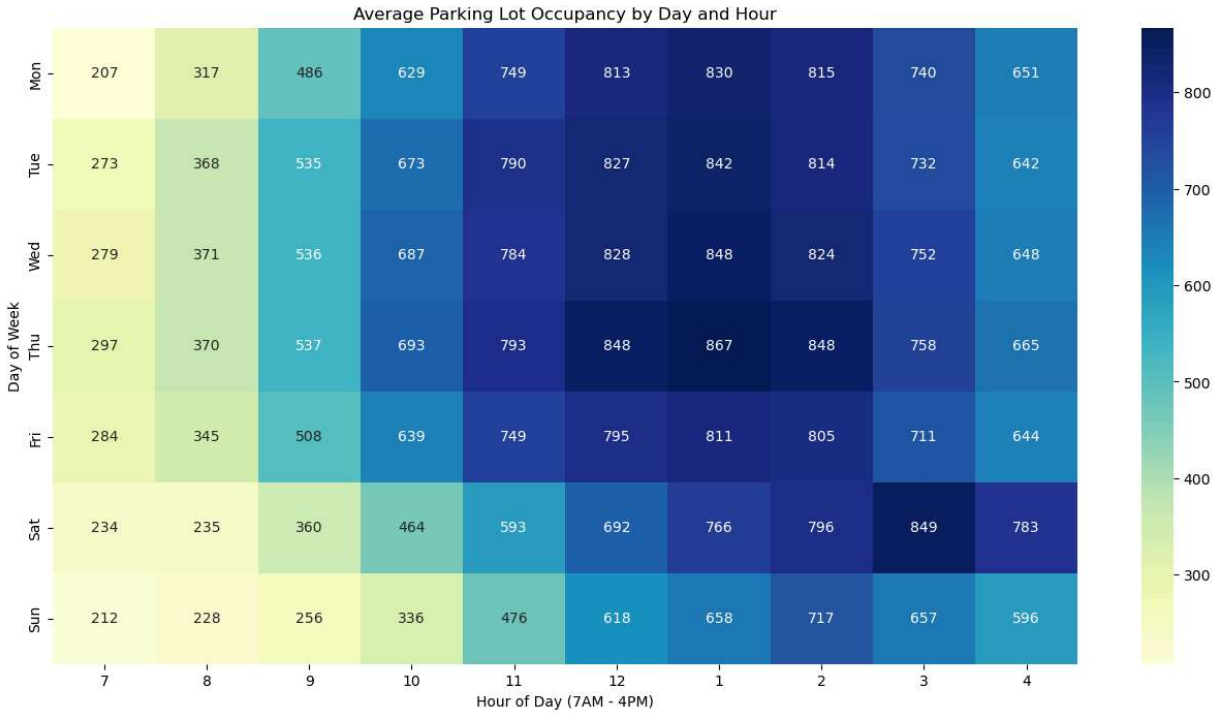
In this part we will visualize the relationship between three variables: *day*, *hour*, and *Occupancy* by using a heat map:

In [25]:
```
# Preparing the data for the heatmap
# Grouping the data by 'day' and 'hour', then calculating the average occupancy for
heatmap_df = df.groupby(['day', 'hour'])['Occupancy'].mean().reset_index()
heatmap_df_pivot = heatmap_df.pivot(index="day", columns="hour", values="Occupancy"

# Reordering the days of the week for the heatmap
days_order = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
hours_order = [7, 8, 9, 10, 11, 12, 1, 2, 3, 4]
heatmap_df_pivot = heatmap_df_pivot.reindex(index=days_order, columns=hours_order)

# Generate the heatmap
plt.figure(figsize=(16, 8))
sns.heatmap(heatmap_df_pivot, cmap="YlGnBu", annot=True, fmt=".0f")
plt.title('Average Parking Lot Occupancy by Day and Hour')
```

```
plt.xlabel('Hour of Day (7AM - 4PM)')
plt.ylabel('Day of Week')
plt.show()
```



Based on the data from the heat map we can say that:

- In weekdays, demand reached its peak at lunch time, with a slow rise in the morning and a fall in the afternoon.
- Saturdays have a later peak period and lower occupancy rates.
- Sundays has a lowest morning occupancy with a moderate rise from lunch time.

Unique values of a categorical variable

In this part we will identify the distinct value of the *SystemCodeNumber* column, and observe their frequency

In [26]:
```
#Identify the unique value in the 'SystemCodeNumber' column and count them
unique_carpark_frequency = df['SystemCodeNumber'].value_counts()

# Display the unique values and their frequency
print(unique_carpark_frequency)
```

```
SystemCodeNumber
BHMBCCMKT01          1312
BHMNCPNST01          1312
Others-CCCPS98       1312
Others-CCCPS8        1312
Others-CCCPS202      1312
Others-CCCPS135a     1312
Others-CCCPS119a     1312
Others-CCCPS105a     1312
BHMEURBRD01          1312
Broad Street         1312
BHMNCPHST01          1312
BHMMBMMBX01          1311
Shopping             1309
Others-CCCPS133      1294
BHMNCPPLS01          1291
BHMNCPLDH01          1291
BHMBCCSNH01          1283
BHMEURBRD02          1276
BHMBCCPST01          1274
NIA South            1204
NIA Car Parks        1204
BHMBRCBRG03          1186
BHMNCPRAN01          1186
Bull Ring            1184
BHMBRCBRG02          1156
BHMBRCBRG01          1100
BHMBCCTHL01          1072
BHMNCPNHS01          1036
NIA North             150
BHMBRTARC01            88
Name: count, dtype: int64
```

Independence Test

Let's build a contingency table of two potentially related categorical variables, which would be *per_occupancy*, and *hour*

In [27]:
```
hours_ordered = [7, 8, 9, 10, 11, 12, 1, 2, 3, 4]
contingency_table = pd.crosstab(df['per_occupancy'], df['hour'], margins = False)
contingency_table = contingency_table[hours_ordered]
contingency_table
```

Out[27]:

hour	7	8	9	10	11	12	1	2	3	4
per_occupancy										
0-25	539	2188	1632	1057	723	499	506	460	573	562
25-50	261	1435	1133	1172	1154	1033	1041	949	1013	933
50-75	31	384	996	840	980	1098	1173	1170	1256	1195
75-100	2	10	274	750	1077	1144	1319	1284	1055	426

Now we will run a statistical test to determine the independence between *per_occupancy* and *hour*. The appropriate test to be used in this instance is the **Chi-Square test**.

In this test we would first determine the null hypothesis(H0) and the the alternative hypothesis (H1):

- **H0**: There is no dependence between *hour* and *per_occupancy*
- **H1**: There is a dependence between *hour* and *per_occupancy*

Let's run the test:

In [28]:
```
# The Chi-Square test function
chi2, p_val, dof, expected = stats.chi2_contingency(contingency_table)
print(f"p-value: {p_val}")
```

```
p-value: 0.0
```

Since the p-value is smaller than 0.05, we reject the null hypothesis of no dependence between *hour* and *per_occupancy*. Therefore we can agree that the hour does affect the number of cars in a car park.

Subset of the dataset

Now, let us examine a portion of the data. Our analysis will be centered on the statistical data of the car park with the IDs of **BHMBCCMKT01** and **BHMNCPNST01** during the morning hours of a typical workday.

First, let's subset the dataset to consider the data with the above criteria.

In [29]:
```
# Subset the data of car park ID: BHMBCCMKT0 during the morning hours of a typical
df_subset1 = df[(df.WorkingDay == 'Yes') & (df.period == 'AM') & (df.SystemCodeNumb
df_subset2 = df[(df.WorkingDay == 'Yes') & (df.period == 'AM') & (df.SystemCodeNumb
```

In [30]:
```
# Descriptive statistics of the first subset
df_subset1.describe()
```

Out[30]:

	Capacity	Occupancy	per_usage	year	hour
count	444.0	444.000000	444.000000	444.0	444.000000
mean	577.0	102.858118	17.826824	2016.0	9.398649
std	0.0	63.027792	10.923013	0.0	1.207986
min	577.0	7.000000	1.210000	2016.0	7.000000
25%	577.0	52.000000	9.010000	2016.0	8.000000
50%	577.0	92.000000	15.940000	2016.0	9.000000
75%	577.0	153.000000	26.520000	2016.0	10.000000
max	577.0	370.000000	64.120000	2016.0	11.000000

In [31]:
```
# Descriptive statistics of the first subset
df_subset2.describe()
```

Out[31]:

	Capacity	Occupancy	per_usage	year	hour
count	448.0	448.000000	448.000000	448.0	448.000000
mean	485.0	272.533482	56.192500	2016.0	9.361607
std	0.0	53.704001	11.072911	0.0	1.236488
min	485.0	136.000000	28.040000	2016.0	7.000000
25%	485.0	232.000000	47.840000	2016.0	8.000000
50%	485.0	275.000000	56.700000	2016.0	9.000000
75%	485.0	313.250000	64.590000	2016.0	10.000000
max	485.0	397.000000	81.860000	2016.0	11.000000

Statistical Test

Now let's conduct a hypothesis test to see if there is any significance in the difference between the mean of two subsets.

We will first formulate the null hypothesis and the alternative hypothesis

- **H0**: There is no difference between the means of the two subsets
- **H1**: There is a significant difference between them

Now let's run the test:

In [32]:
```
# The Independent two samples t-test function:
t_val, p_val = stats.ttest_ind(df_subset1['Occupancy'], df_subset2['Occupancy'])
print(f"t-value: {t_val}, p-value: {p_val}")
```

```
t-value: -43.28941456702498, p-value: 3.215600200322334e-221
```

From the result of the test, we can agree that there is a significant difference between the mean of *Occupancy* between car park ID: **BHMBCCMKT01** and **BHMNCPNST01**.

Grouping Data

The next thing we would do is to group data for further analysis. First, let group the data by *SystemCodeNumber* and *month* to see overall data about each car park on a monthly basis

In [33]:
```
# Group by month and calculate the mean of per_usage, and sum of Occupancy and Capa
monthly_summary = df.groupby('month').agg({
    'per_usage': 'mean',
    'Occupancy': 'sum',
    'Capacity': 'sum'
})
monthly_summary
```

Out[33]:

month	per_usage	Occupancy	Capacity
Dec	52.730910	5.773059e+06	11480825
Nov	48.329812	9.469857e+06	20897207
Oct	46.808664	7.459068e+06	17292546

In [34]:
```
# Group by day and calculate the mean of per_usage, and sum of Occupancy
day_summary = df.groupby('day').agg({
    'per_usage': 'mean',
    'Occupancy': 'sum',
})

day_summary
```

Out[34]:

day	per_usage	Occupancy
Fri	50.754006	3.249723e+06
Mon	51.067960	3.556726e+06
Sat	41.997310	2.753085e+06
Sun	33.858961	2.286716e+06
Thu	54.293487	3.446568e+06
Tue	53.583575	3.681824e+06
Wed	53.644368	3.727342e+06

In [35]:
```
# Group by hour and calculate the mean of per_usage, and sum of Occupancy
hour_summary = df.groupby('hour').agg({
    'per_usage': 'mean',
    'Occupancy': 'sum',
})

hour_summary
```

Out[35]:

hour	per_usage	Occupancy
1	59.557846	3.246972e+06
2	59.847296	3.108123e+06
3	55.738684	2.892573e+06
4	50.020757	2.058694e+06
7	20.818211	2.146259e+05
8	25.353547	1.291017e+06
9	37.136708	1.872224e+06
10	46.519756	2.265879e+06
11	54.099443	2.804905e+06
12	58.156362	2.946971e+06

Linear Regression Analysis

We will use a regression model to predict the value of *Occuapncy*, and the variables that we will use in the model are:

- **Capacity**
- **month**
- **day**
- **hour**
- **period**

These variables are chosen because they are likely to have a relationship with *Occupancy*

We exclude *SystemCodeNumber*, *per_usage*, *per_occupancy*, *WorkingDay* and *year* because they're derived from Capacity, Occupancy, and day, or are constant.

First, to run the regression analysis, we have to create dummy variables for the categorical variable so that Python can interpret them and run the regression model. These include:

- **month**
- **day**

- **period**

After we get the dummy variables, we have to drop from each categorical value one dummy variable to avoid the dummy variable trap.

In [36]:
```
#Create a list of column to turn into dummy variables
dummy_columns = ['month', 'day', 'period']
#Transform the categorical data into dummy variables
dummies_df = pd.get_dummies(df, prefix=None, prefix_sep='_', dummy_na=False, column
#Drop unused columns
dummies_df = dummies_df.drop(columns=['SystemCodeNumber', 'per_usage', 'per_occupan
dummy_df = dummies_df.drop(columns =['month_Dec', 'day_Sat', 'period_AM'])
#Display the datastet that will be used for regression analysis
dummy_df
```

Out[36]:

	Capacity	Occupancy	hour	month_Nov	month_Oct	day_Fri	day_Mon	day_Sun
0	577	61.0	7	False	True	False	False	False
1	577	64.0	8	False	True	False	False	False
2	577	80.0	8	False	True	False	False	False
3	577	107.0	9	False	True	False	False	False
4	577	150.0	9	False	True	False	False	False
...	...	...	...	...	...	...	...	...
35327	1920	1517.0	2	False	False	False	True	False
35328	1920	1487.0	3	False	False	False	True	False
35329	1920	1432.0	3	False	False	False	True	False
35330	1920	1321.0	4	False	False	False	True	False
35331	1920	1180.0	4	False	False	False	True	False

35327 rows × 12 columns

Now we will run the multiple regression model:

In [37]:
```
# Define the feature matrix (X) and the target variable (y)
x = dummy_df.drop('Occupancy', axis=1) # Independent variable
y = dummy_df['Occupancy']  # Target variable
```

In [38]:
```
from sklearn.linear_model import LinearRegression
# Convert boolean columns to integer type if they're not already
x = x.astype(int)

# Add a constant term to the independent variables matrix
x = sm.add_constant(x)
```

```
# Create the OLS model
model = sm.OLS(y, x)

# Fit the model
result = model.fit()

#Show the output of the model
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Occupancy   R-squared:                       0.655
Model:                            OLS   Adj. R-squared:                  0.655
Method:                 Least Squares   F-statistic:                     6105.
Date:                Tue, 26 Mar 2024   Prob (F-statistic):               0.00
Time:                        02:52:36   Log-Likelihood:            -2.6064e+05
No. Observations:               35327   AIC:                         5.213e+05
Df Residuals:                   35315   BIC:                         5.214e+05
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -172.8133     10.105    -17.102      0.000    -192.620    -153.007
Capacity       0.4332      0.002    248.522      0.000       0.430       0.437
hour           8.3119      0.685     12.129      0.000       6.969       9.655
month_Nov    -71.9931      5.395    -13.343      0.000     -82.568     -61.418
month_Oct    -94.1672      5.586    -16.858      0.000    -105.116     -83.219
day_Fri       74.0512      7.973      9.288      0.000      58.424      89.678
day_Mon       80.8302      7.800     10.362      0.000      65.541      96.119
day_Sun      -88.7682      8.048    -11.030      0.000    -104.543     -72.994
day_Thu      113.7583      7.968     14.277      0.000      98.141     129.375
day_Tue      107.1660      7.774     13.786      0.000      91.930     122.402
day_Wed      114.1147      7.776     14.675      0.000      98.874     129.356
period_PM    289.8399      5.374     53.937      0.000     279.307     300.373
==============================================================================
Omnibus:                     4906.184   Durbin-Watson:                   0.140
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            19008.096
Skew:                           0.661   Prob(JB):                         0.00
Kurtosis:                       6.342   Cond. No.                     1.47e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
[2] The condition number is large, 1.47e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Here are some key points from the output of the code:

The adjusted R-squared: 65.5%. This indicates that the model does not include any non-related variables.

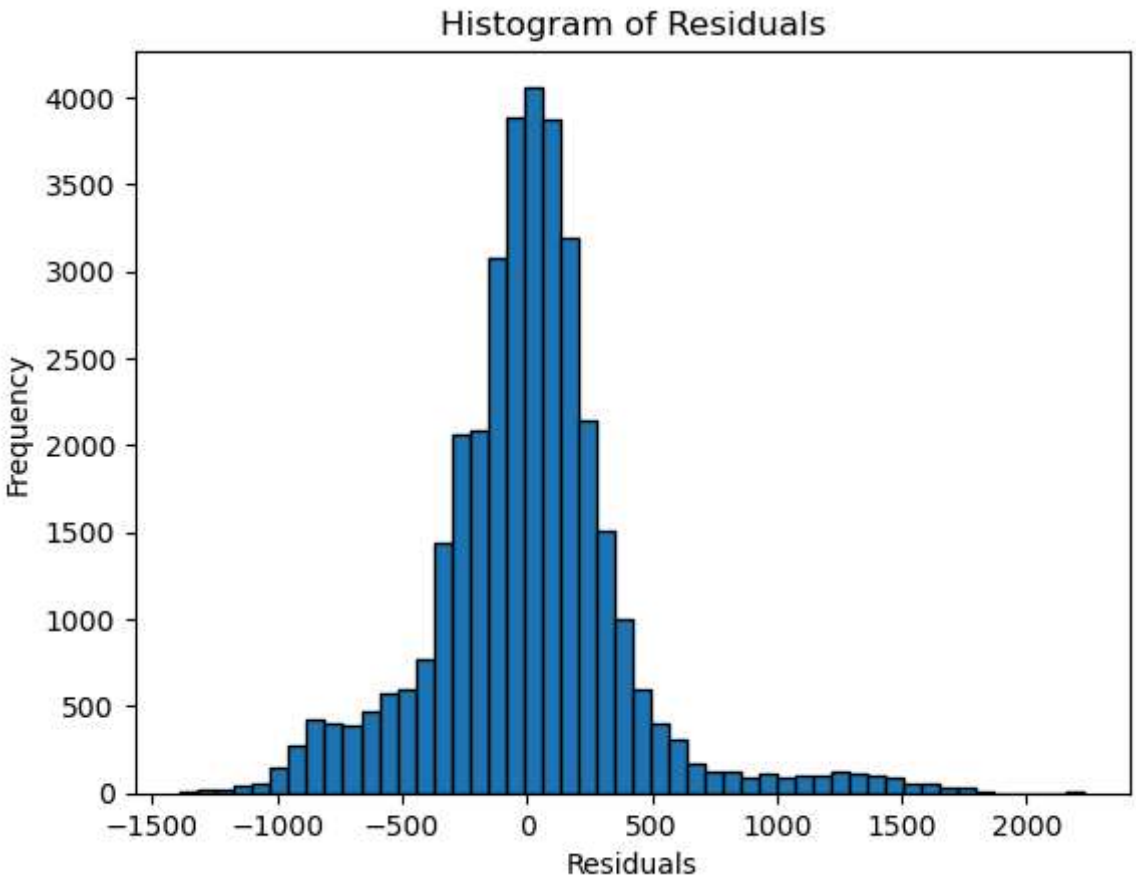
P-values: All variables have p-values below 0.05, showing their statistical significance as predictors of *Occupancy*

In conclusion, the model is good enough to explain the relationship of *Occupancy* with other variables. However, the model still need to satisfy the five assumptions in the residual analysis.

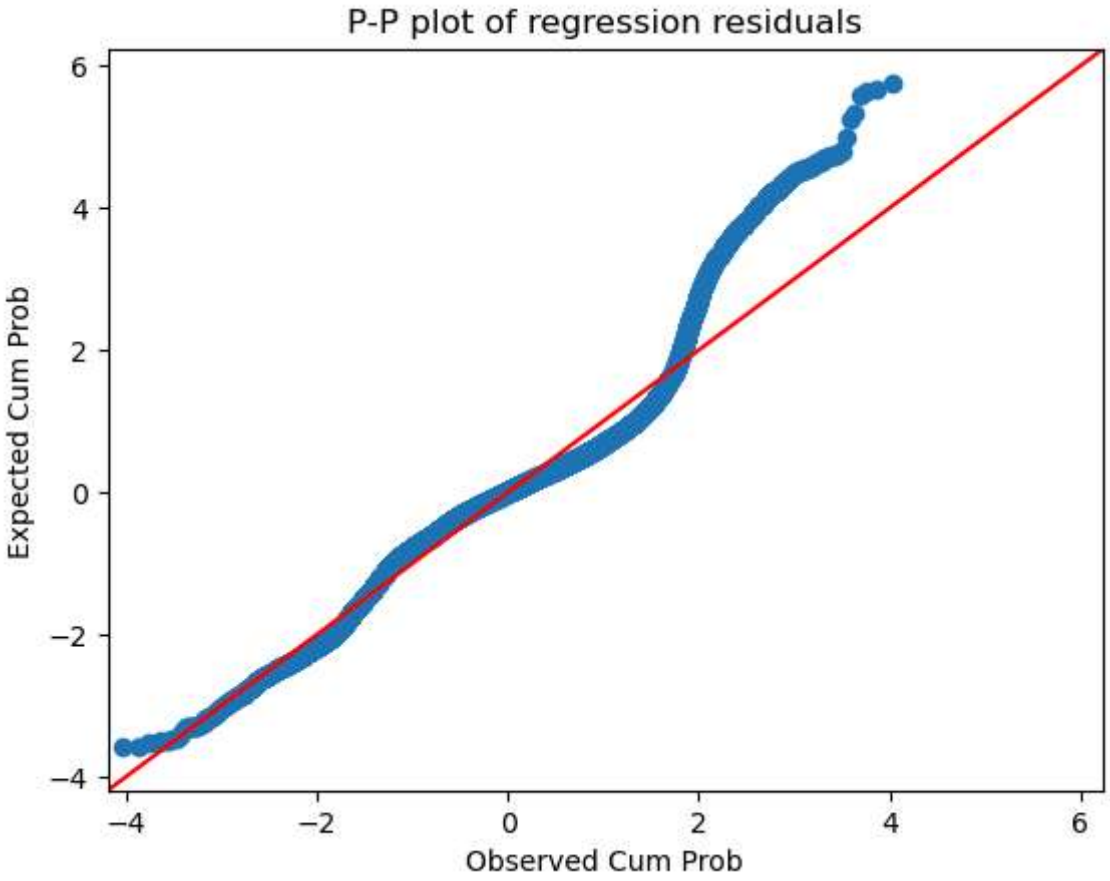
Residual Analysis

In [39]:
```
# Get the residuals
residuals = result.resid


# Plot a histogram of the residuals
plt.hist(residuals, bins=50, edgecolor='black')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')
plt.show()
```

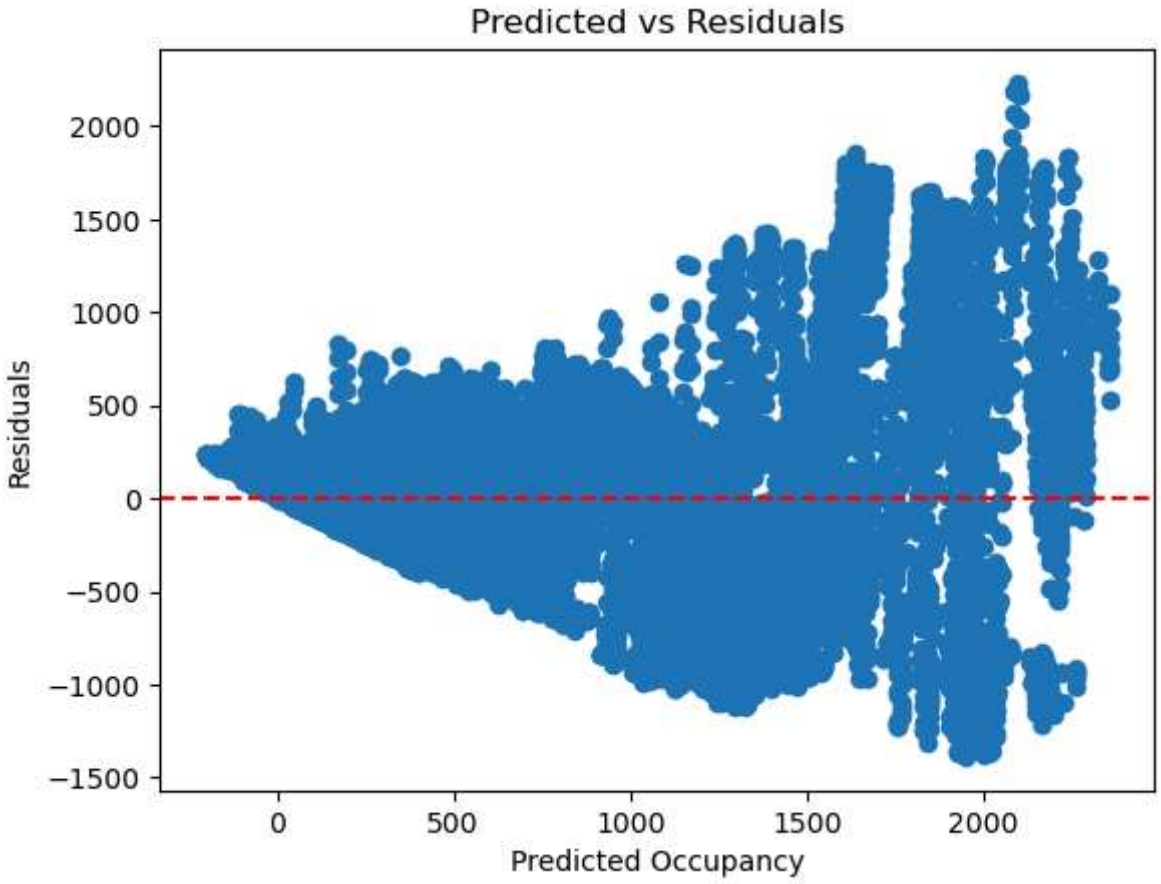


In [40]:
```
# Generate the P-P plot
fig = sm.qqplot(residuals, line='45', fit=True)
plt.title('P-P plot of regression residuals')
plt.xlabel('Observed Cum Prob')
plt.ylabel('Expected Cum Prob')
plt.show()
```



In [41]:
```
# Get the predicted values
predicted_values = result.fittedvalues

# Scatter plot of predicted values vs residuals
plt.scatter(predicted_values, residuals)
plt.xlabel('Predicted Occupancy')
plt.ylabel('Residuals')
plt.title('Predicted vs Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



Based on the graphs representing the residuals of the model, it can be seen that the four assumptions of Normality, Linearity, Independence, and Homoscedasticity are not significantly violated.

In [42]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif_data = pd.DataFrame()
vif_data["feature"] = x.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(x.values, i)
                          for i in range(len(x.columns))]

print(vif_data)
```

```
      feature        VIF
0       const  24.047096
1    Capacity   1.000429
2        hour   1.694107
3   month_Nov   1.670606
4   month_Oct   1.675716
5     day_Fri   1.794475
6     day_Mon   1.843708
7     day_Sun   1.754622
8     day_Thu   1.797094
9     day_Tue   1.849471
10    day_Wed   1.848631
11  period_PM   1.694376
```

As we can see from the output, all variables's VIF values are small, indicating that there is no correlation between the variables. Therefore Multicollinearity is not violated.

In conclusion, based on the residual analysis, the model is considered good enough to predict the value of Occupancy based on other factors.

Conclusion

In conclusion, the outcomes of the exploratory data analysis (EDA) carried out on the parking dataset in Birmingham have given us important information that can have a big influence on planning techniques. Parking and urban management. We now have a better grasp of parking lot utilization trends and the key variables that affect how many cars are parked in a lot of thanks to the creation of visualizations and statistical analysis. This can assist us in making defensible choices that will enhance parking lot usage going forward.