Problem Statement:

Uber has received some complaints from their customers facing problems related to ride cancellations by the driver and non-availability of cars for a specific route in the city.

The uneven supply-demand gap for cabs from City to Airport and vice-versa is causing a bad effect on customer relationships as well as Uber is losing out on its revenue.

The aim of analysis is to identify the root cause of the problem (i.e. cancellation and non-availability of cars) and recommend ways to tackle the situation.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from scipy.stats import chi2_contingency
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
```

In [2]:
```
df = pd.read_csv('uber-data.csv', parse_dates=[4,5], dayfirst = True, na_values = "NA")
df
```

Out[2]:

	Request id	Pickup point	Driver id	Status	Request timestamp	Drop timestamp
0	619	Airport	1.0	Trip Completed	2016-07-11 11:51:00	2016-07-11 13:00:00
1	867	Airport	1.0	Trip Completed	2016-07-11 17:57:00	2016-07-11 18:47:00
2	1807	City	1.0	Trip Completed	2016-07-12 09:17:00	2016-07-12 09:58:00
3	2532	Airport	1.0	Trip Completed	2016-07-12 21:08:00	2016-07-12 22:03:00
4	3112	City	1.0	Trip Completed	2016-07-13 08:33:16	2016-07-13 09:25:47
...	...	...	...	...	...	...
6740	6745	City	NaN	No Cars Available	2016-07-15 23:49:03	NaT
6741	6752	Airport	NaN	No Cars Available	2016-07-15 23:50:05	NaT
6742	6751	City	NaN	No Cars Available	2016-07-15 23:52:06	NaT
6743	6754	City	NaN	No Cars Available	2016-07-15 23:54:39	NaT
6744	6753	Airport	NaN	No Cars Available	2016-07-15 23:55:03	NaT

6745 rows × 6 columns

In [3]:
```
# key note here is, we see fee null and nan values in driver id and drop time stamp because:

# when user requests for cab and driver cancels it then the drop time stamp will be automatically null or missing
# And when no cabs available at the point of time, then automatically the driver id and drop time stamp column will be
# or missing

# Because of these reasons, we see null and missing values only in both of these columns

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6745 entries, 0 to 6744
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Request id         6745 non-null   int64
 1   Pickup point       6745 non-null   object
 2   Driver id          4095 non-null   float64
 3   Status             6745 non-null   object
 4   Request timestamp  6745 non-null   datetime64[ns]
 5   Drop timestamp     2831 non-null   datetime64[ns]
dtypes: datetime64[ns](2), float64(1), int64(1), object(2)
memory usage: 316.3+ KB
```

In [76]:
```
# Checking for percentage of null values in each column

df.isnull().sum() / len(df) * 100
```

Out[76]:
```
Request id            0.000000
Pickup point          0.000000
Driver id            39.288362
Status                0.000000
Request timestamp     0.000000
Drop timestamp       58.028169
Time_period           0.000000
Cab_Availability      0.000000
Estimated_Demand      0.000000
dtype: float64
```

In [74]:
```
# Categorizing the time stamp into multiple categories of day

def categorize_timeperiod(hour):
    if hour <= 4:
        return 'Dawn'
    elif hour <= 9 :
        return 'Early Morning'
    elif hour <= 16:
        return 'Afternoon'
    elif hour <= 21:
        return 'Late Evening'
    else:
        return 'Night'
```

In [75]:
```
df['Time_period'] = df['Request timestamp'].dt.hour.apply(categorize_timeperiod)
df['Time_period'].value_counts()
```

Out[75]:
```
Late Evening     2342
Early Morning    2103
Afternoon        1224
Dawn              578
Night             498
Name: Time_period, dtype: int64
```

In [103]:
```
# Checking at which hours of the day the cabs are unavailable

# we can get to know about this when we put a filter on Trip completed status
# if Trip completed -> then cab is available, else if no cars available or cancelled then it means 'no cab available'
# point of time

def categorize_availability(availability):
    if availability == 'Trip Completed':
        return 'Cab Available'
    else:
        return 'No Cabs Available'
```

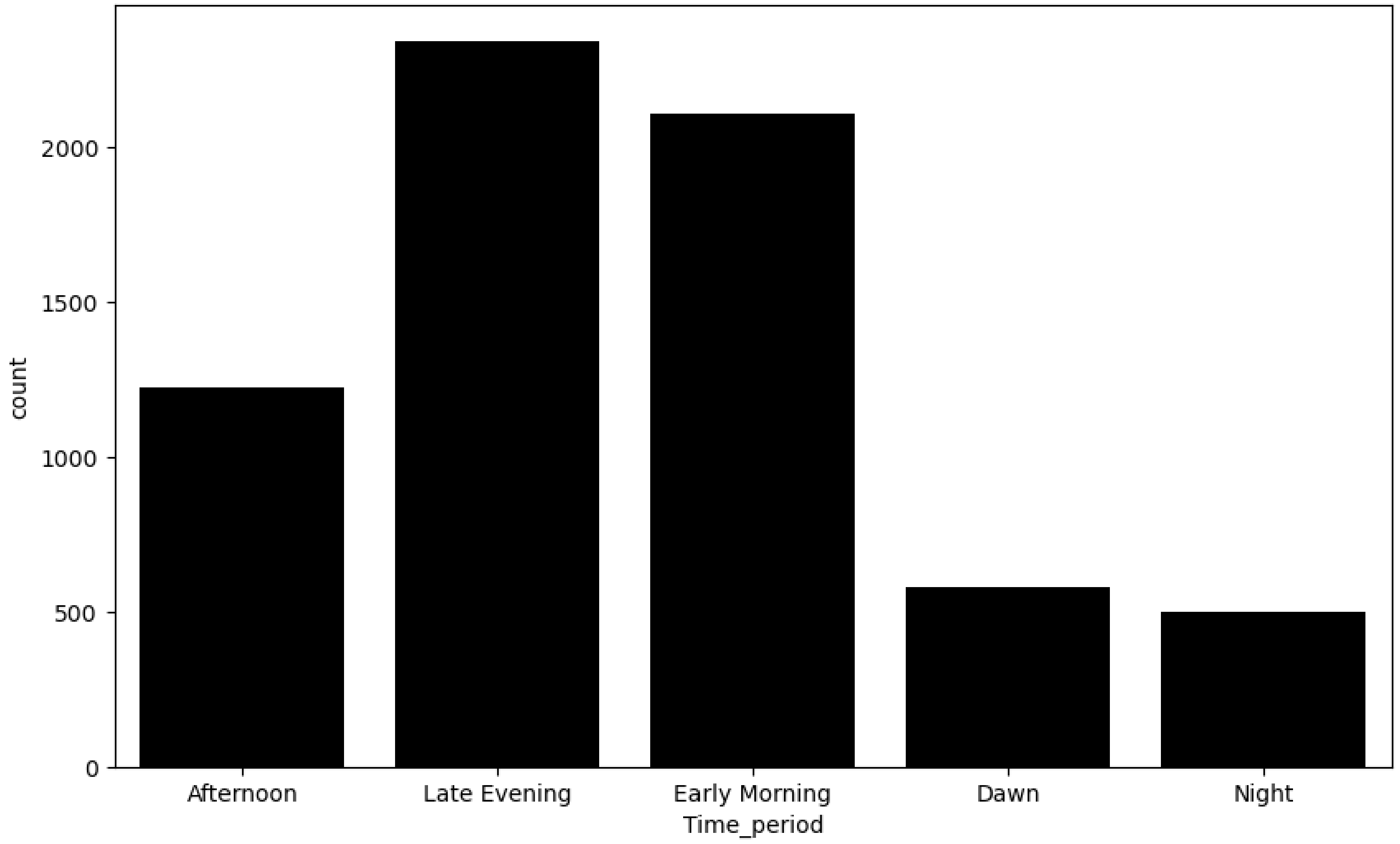
In [104]:
```
df['Cab_Availability'] = df['Status'].apply(categorize_availability)
df
```

Out[104]:

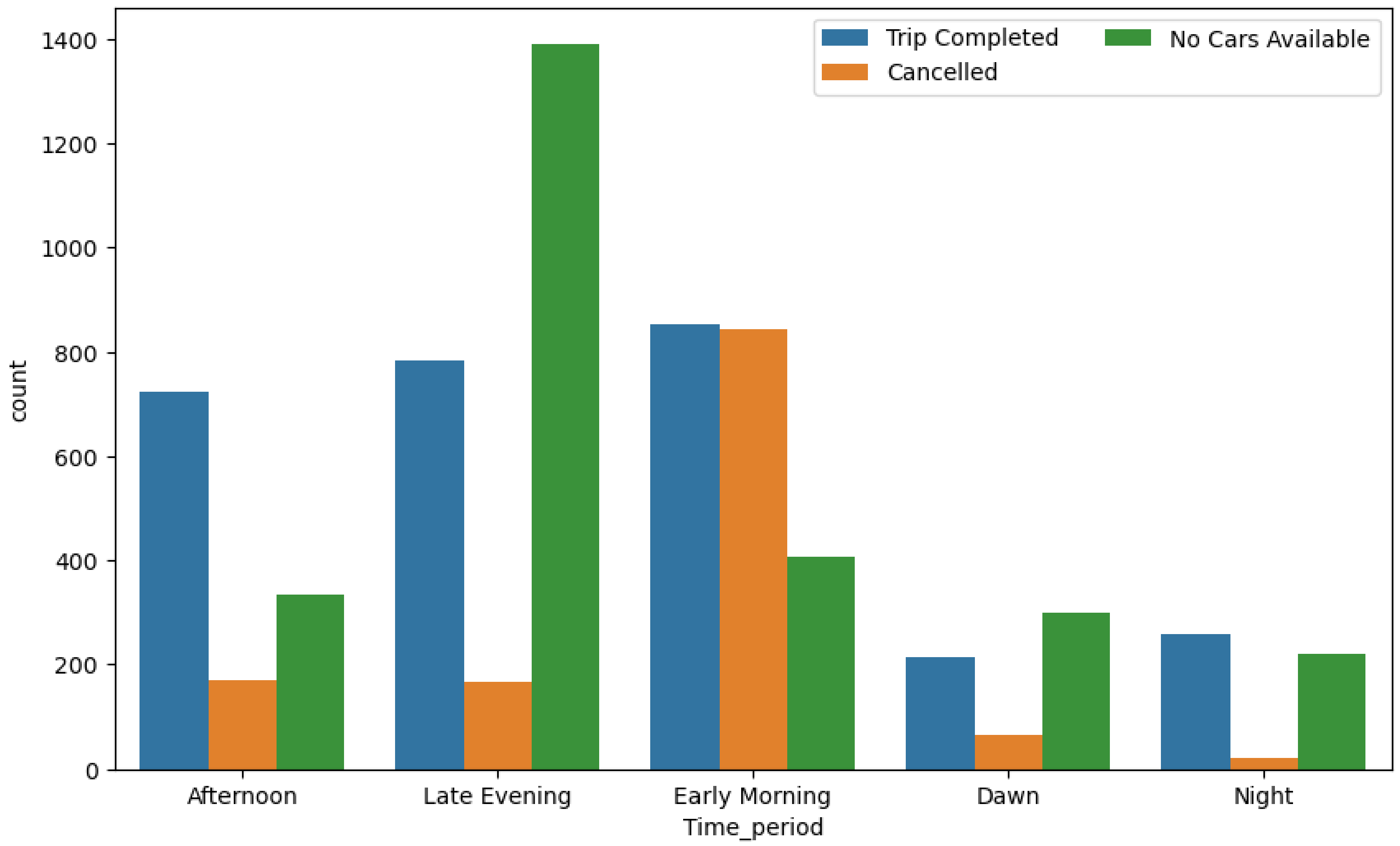
	Request id	Pickup point	Driver id	Status	Request timestamp	Drop timestamp	Time_period	Cab_Availability	Estimated_Demand
0	619	Airport	1.0	Trip Completed	2016-07-11 11:51:00	2016-07-11 13:00:00	Afternoon	Cab Available	0
1	867	Airport	1.0	Trip Completed	2016-07-11 17:57:00	2016-07-11 18:47:00	Late Evening	Cab Available	0
2	1807	City	1.0	Trip Completed	2016-07-12 09:17:00	2016-07-12 09:58:00	Early Morning	Cab Available	0
3	2532	Airport	1.0	Trip Completed	2016-07-12 21:08:00	2016-07-12 22:03:00	Late Evening	Cab Available	0
4	3112	City	1.0	Trip Completed	2016-07-13 08:33:16	2016-07-13 09:25:47	Early Morning	Cab Available	0
...	...	...	...	...	...	...	...	...	...
6740	6745	City	NaN	No Cars Available	2016-07-15 23:49:03	NaT	Night	No Cabs Available	0
6741	6752	Airport	NaN	No Cars Available	2016-07-15 23:50:05	NaT	Night	No Cabs Available	0
6742	6751	City	NaN	No Cars Available	2016-07-15 23:52:06	NaT	Night	No Cabs Available	0
6743	6754	City	NaN	No Cars Available	2016-07-15 23:54:39	NaT	Night	No Cabs Available	0
6744	6753	Airport	NaN	No Cars Available	2016-07-15 23:55:03	NaT	Night	No Cabs Available	0

6745 rows × 9 columns

In [105]:
```python
# Categorizing the requesting time of users in a day per time period

plt.figure(figsize = (10,6))
sns.countplot(data = df, x = 'Time_period', palette=["#000000"])
plt.show()
```



In [106]:
```
# Categorizing the Cabs Availability per time period

# Blank indicates ride is completed
# Red indicates either cancelled or no cabs available at the point of time

plt.figure(figsize = (10,6))
sns.countplot(data = df, x = 'Time_period', hue = 'Status')
plt.legend(loc = 'upper right', frameon = True, ncol = 2)
plt.show()
```



In [107]:
```
# Morning Time Cab Availability

df.loc[df['Time_period'] == 'Early Morning','Status'].value_counts() / len(df) * 100
```

Out[107]:
```
Trip Completed       12.661231
Cancelled            12.498147
No Cars Available     6.019274
Name: Status, dtype: float64
```

In [109]:
```
# Late Evening Cab Availability

df.loc[df['Time_period'] == 'Late Evening','Status'].value_counts() / len(df) * 100
```

Out[109]:
```
No Cars Available    20.637509
Trip Completed       11.623425
Cancelled             2.461082
Name: Status, dtype: float64
```

In [110]:
```
# Night time Cab Availability

df.loc[df['Time_period'] == 'Night','Status'].value_counts() / len(df) * 100
```

Out[110]:
```
Trip Completed       3.810230
No Cars Available    3.246850
Cancelled            0.326168
Name: Status, dtype: float64
```

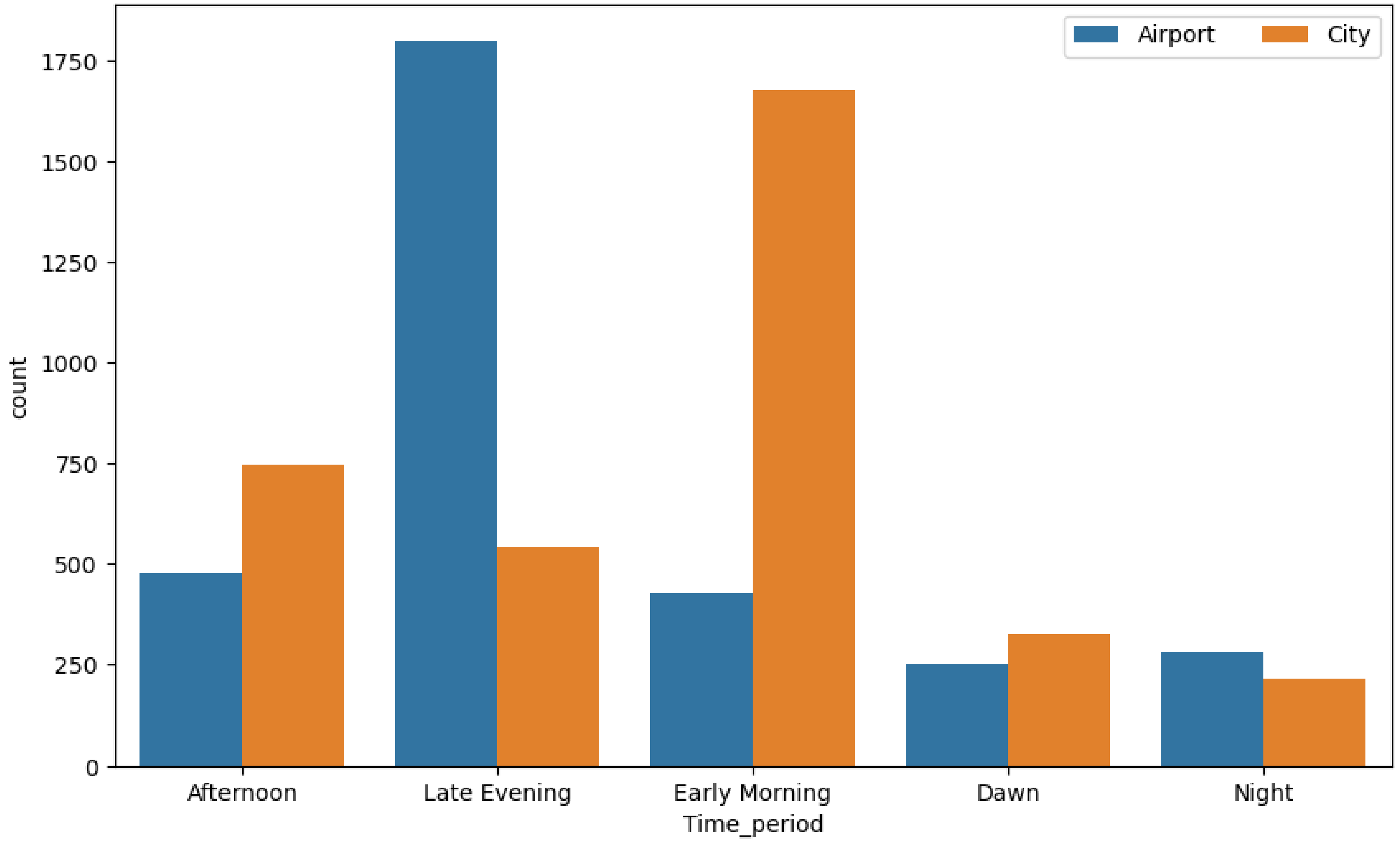
In [111]:
```
# Dawn time Cab Availability

df.loc[df['Time_period'] == 'Dawn','Status'].value_counts() / len(df) * 100
```

Out[111]:
```
No Cars Available    4.432913
Trip Completed       3.172721
Cancelled            0.963677
Name: Status, dtype: float64
```

In [112]:
```
# Afternoon time Cab Availability

df.loc[df['Time_period'] == 'Afternoon','Status'].value_counts() / len(df) * 100
```

Out[112]:
```
Trip Completed       10.704225
No Cars Available     4.951816
Cancelled             2.490734
Name: Status, dtype: float64
```

In [113]:
```python
# IN late evening, we recieve more ride requests from users travelling from Airport to City, while it is vice versa
# in the Early Morning

plt.figure(figsize = (10,6))
sns.countplot(data = df, x = 'Time_period', hue = 'Pickup point')
plt.legend(loc = 'upper right', frameon = True, ncol = 2)
plt.show()
```



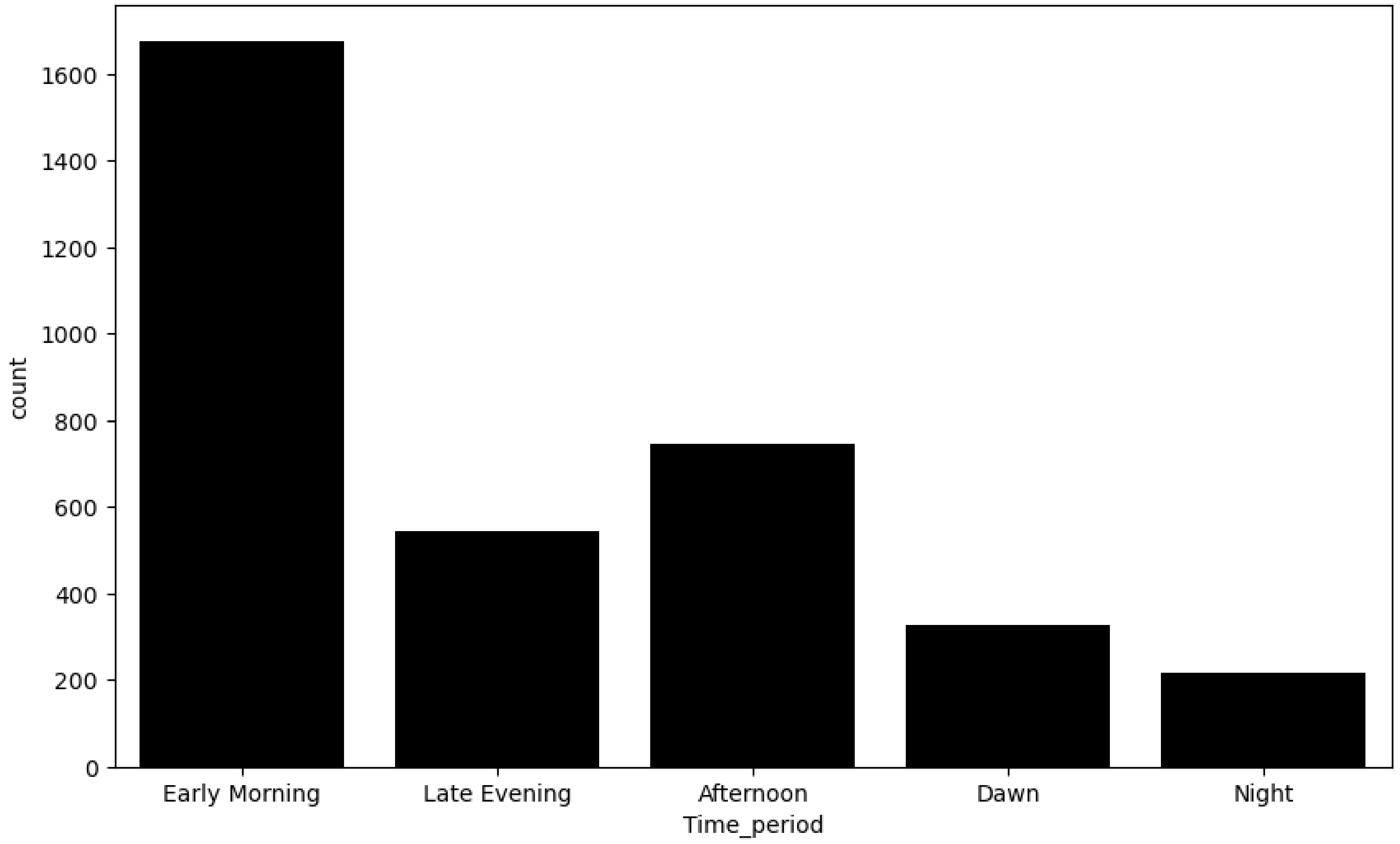
In [114]:
```
# Filtering the commute time period of users going from City to Airport

city_pickups = df.query("`Pickup point` == 'City'")
city_pickups
```

Out[114]:

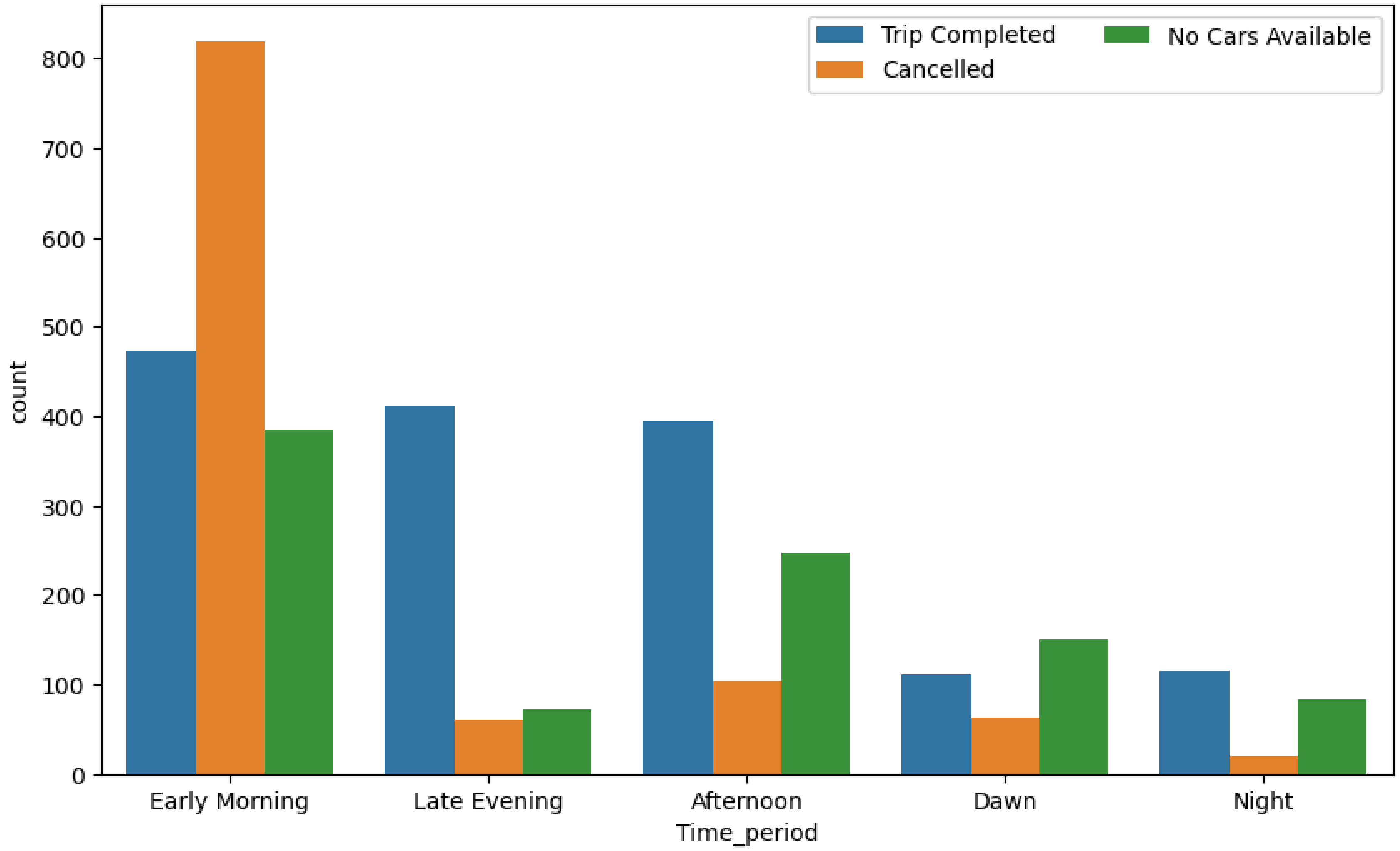
	Request id	Pickup point	Driver id	Status	Request timestamp	Drop timestamp	Time_period	Cab_Availability	Estimated_Demand
2	1807	City	1.0	Trip Completed	2016-07-12 09:17:00	2016-07-12 09:58:00	Early Morning	Cab Available	0
4	3112	City	1.0	Trip Completed	2016-07-13 08:33:16	2016-07-13 09:25:47	Early Morning	Cab Available	0
8	6248	City	1.0	Trip Completed	2016-07-15 17:57:27	2016-07-15 18:50:51	Late Evening	Cab Available	0
9	267	City	2.0	Trip Completed	2016-07-11 06:46:00	2016-07-11 07:25:00	Early Morning	Cab Available	0
11	1983	City	2.0	Trip Completed	2016-07-12 12:30:00	2016-07-12 12:57:00	Afternoon	Cab Available	0
...	...	...	...	...	...	...	...	...	...
6738	6746	City	NaN	No Cars Available	2016-07-15 23:46:03	NaT	Night	No Cabs Available	0
6739	6739	City	NaN	No Cars Available	2016-07-15 23:46:20	NaT	Night	No Cabs Available	0
6740	6745	City	NaN	No Cars Available	2016-07-15 23:49:03	NaT	Night	No Cabs Available	0
6742	6751	City	NaN	No Cars Available	2016-07-15 23:52:06	NaT	Night	No Cabs Available	0
6743	6754	City	NaN	No Cars Available	2016-07-15 23:54:39	NaT	Night	No Cabs Available	0

3507 rows × 9 columns

In [115]:
```
# Count of requests coming from users in a day when they opt to travel to Airport from the City

plt.figure(figsize = (10,6))
sns.countplot(data = city_pickups, x = 'Time_period', palette=["#000000"])
plt.show()
```

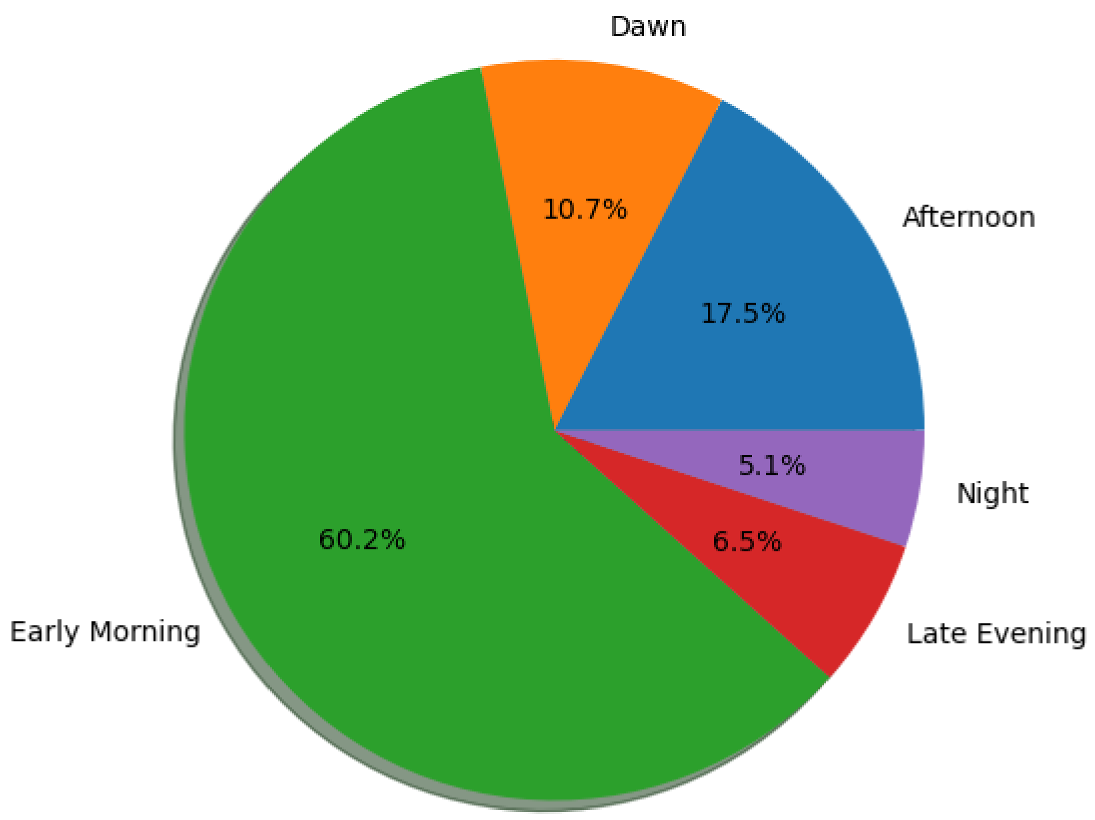


In [159]:
```python
# Cabs Availability in a day when users opt to travel to Airport from the City

plt.figure(figsize = (10,6))
sns.countplot(data = city_pickups, x = 'Time_period', hue = 'Status')
plt.legend(loc = 'upper right', frameon = True, ncol = 2)
plt.show()
```



In [142]:
```
# Percentage of City pickups where cabs are not available

city_pickups[ (city_pickups["Cab_Availability"]=="No Cabs Available") ].groupby(['Time_period']).size().plot(kind="pie
plt.ylabel("")
```

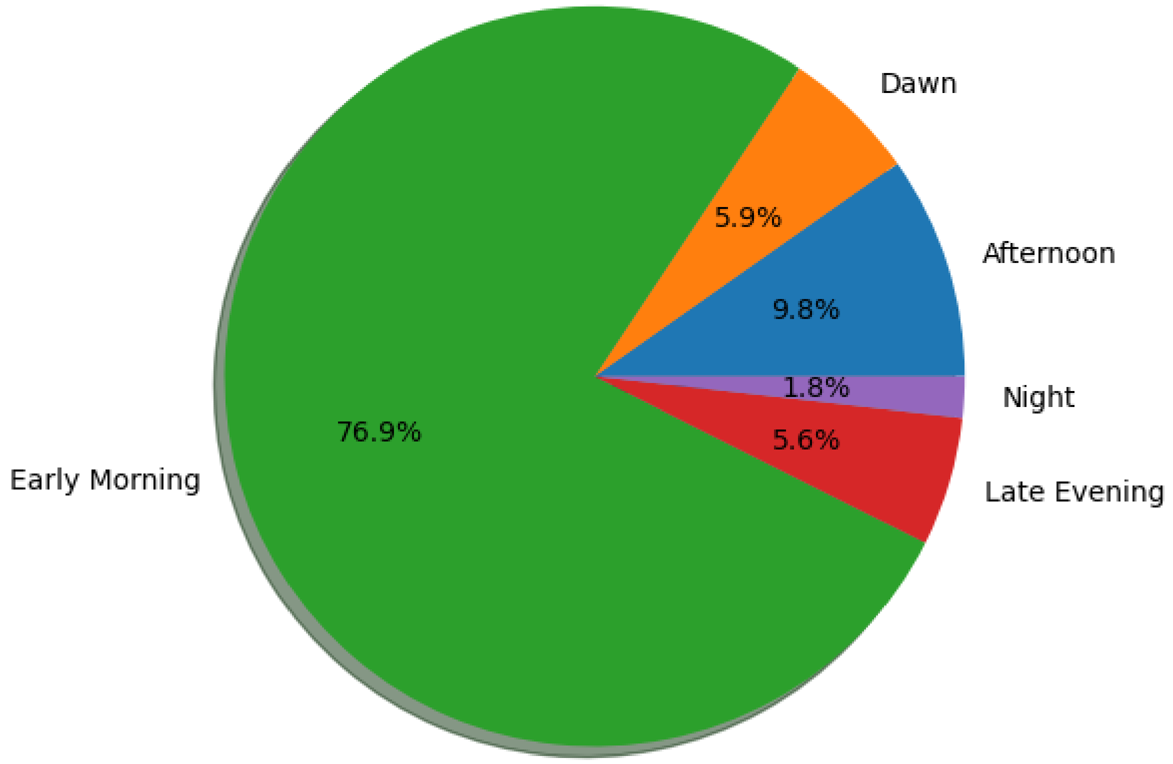
Out[142]:
```
Text(0, 0.5, '')
```



	Afternoon	Dawn	Early Morning	Late Evening	Night
None	351	214	1205	131	102

In [143]:
```
# Percentage of City pickups where drivers cancelled the rides

city_pickups[ (city_pickups["Status"]=="Cancelled") ].groupby(['Time_period']).size().plot(kind="pie", figsize=(6, 6),
plt.ylabel("")
```

Out[143]:
```
Text(0, 0.5, '')
```



	Afternoon	Dawn	Early Morning	Late Evening	Night
None	104	63	820	60	19

In [117]:
```
# Percentage of requests in Morning time from City to Airport

city_pickups.loc[city_pickups['Time_period'] == 'Early Morning','Status'].value_counts() / len(df) * 100
```

Out[117]:
```
Cancelled            12.157153
Trip Completed        6.997776
No Cars Available     5.707932
Name: Status, dtype: float64
```

In [118]:
```
# Percentage of requests in Late Evening time from City to Airport

city_pickups.loc[city_pickups['Time_period'] == 'Late Evening','Status'].value_counts() / len(df) * 100
```

Out[118]:
```
Trip Completed       6.093403
No Cars Available    1.052632
Cancelled            0.889548
Name: Status, dtype: float64
```

In [119]:
```
# Percentage of requests in Afternoon time from City to Airport

city_pickups.loc[city_pickups['Time_period'] == 'Afternoon','Status'].value_counts() / len(df) * 100
```

Out[119]:
```
Trip Completed       5.856190
No Cars Available    3.661972
Cancelled            1.541883
Name: Status, dtype: float64
```

In [120]:
```
# Percentage of requests in Night time from City to Airport

city_pickups.loc[city_pickups['Time_period'] == 'Night','Status'].value_counts() / len(df) * 100
```

Out[120]:
```
Trip Completed       1.704967
No Cars Available    1.230541
Cancelled            0.281690
Name: Status, dtype: float64
```

In [121]:
```
# Percentage of requests in Dawn time from City to Airport

city_pickups.loc[city_pickups['Time_period'] == 'Dawn','Status'].value_counts() / len(df) * 100
```

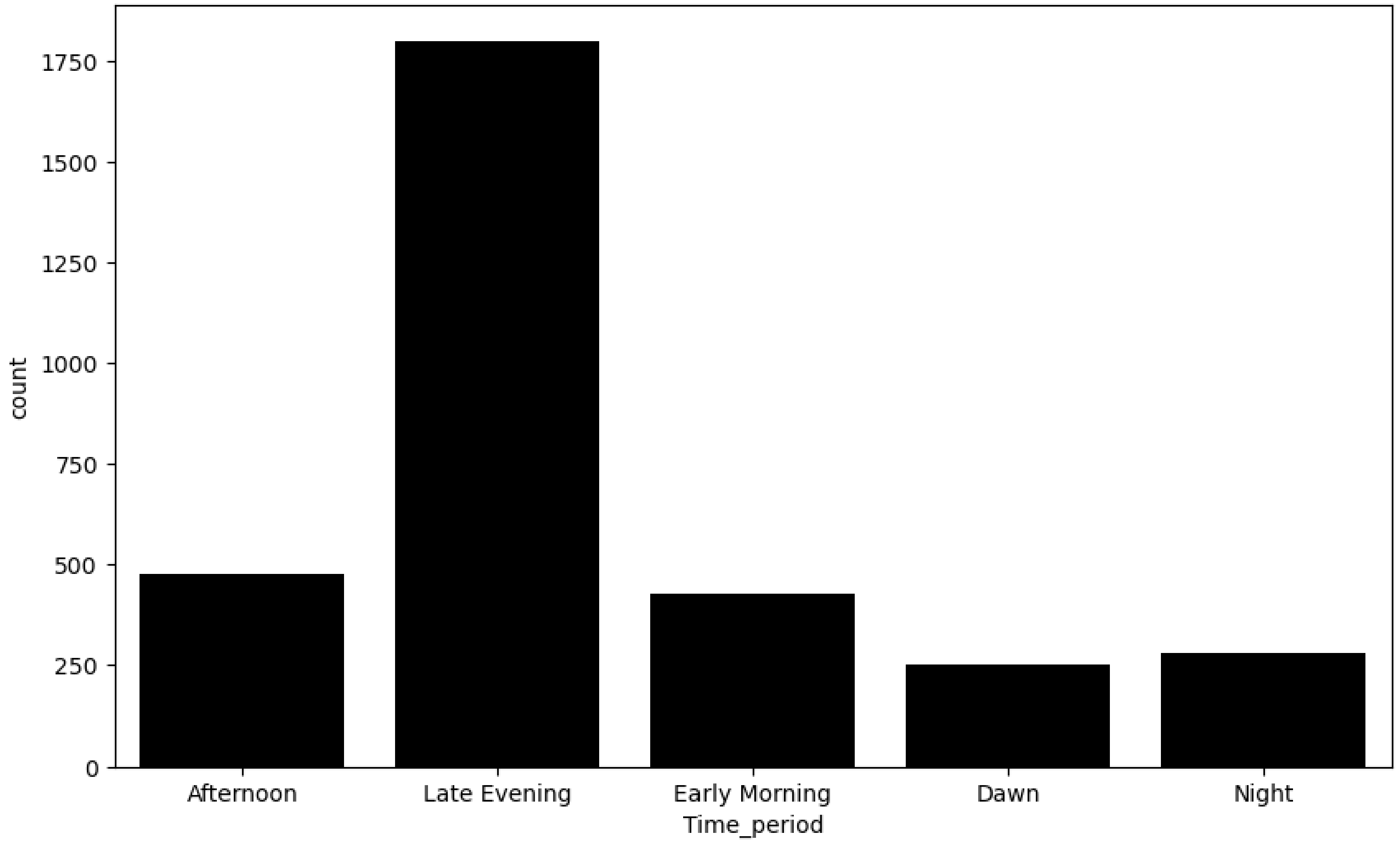
Out[121]:
```
No Cars Available    2.238695
Trip Completed       1.645663
Cancelled            0.934025
Name: Status, dtype: float64
```

In [122]:
```
# Filtering the commute time period for users going from Airport to City

airport_pickups = df.query(" `Pickup point` == 'Airport' ")
airport_pickups
```

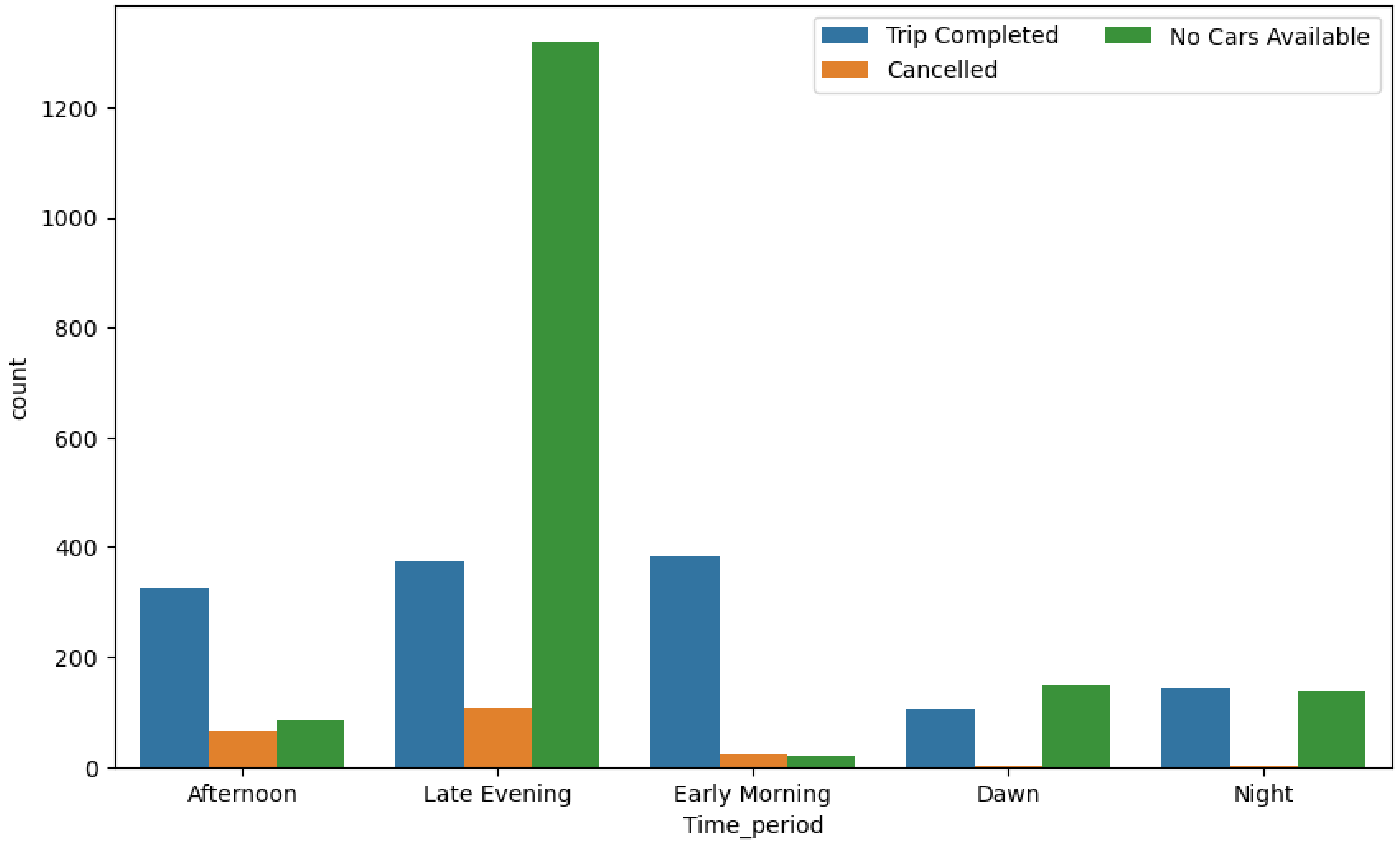
Out[122]:

```
      Request  Pickup  Driver           Status  Request              Drop timestamp  Time_period    Cab_Availability  Estimated_Demand
      id       point   id                       timestamp
0     619      Airport 1.0     Trip Completed   2016-07-11 11:51:00  2016-07-11 13:00:00  Afternoon      Cab Available     0
1     867      Airport 1.0     Trip Completed   2016-07-11 17:57:00  2016-07-11 18:47:00  Late Evening   Cab Available     0
3     2532     Airport 1.0     Trip Completed   2016-07-12 21:08:00  2016-07-12 22:03:00  Late Evening   Cab Available     0
5     3879     Airport 1.0     Trip Completed   2016-07-13 21:57:28  2016-07-13 22:28:59  Late Evening   Cab Available     0
6     4270     Airport 1.0     Trip Completed   2016-07-14 06:15:32  2016-07-14 07:13:15  Early Morning  Cab Available     0
...   ...      ...     ...     ...              ...                  ...                  ...            ...               ...
6734  6732     Airport NaN     No Cars Available 2016-07-15 23:35:50 NaT                  Night          No Cabs Available 0
6735  6737     Airport NaN     No Cars Available 2016-07-15 23:39:15 NaT                  Night          No Cabs Available 0
6736  6744     Airport NaN     No Cars Available 2016-07-15 23:42:51 NaT                  Night          No Cabs Available 0
6741  6752     Airport NaN     No Cars Available 2016-07-15 23:50:05 NaT                  Night          No Cabs Available 0
6744  6753     Airport NaN     No Cars Available 2016-07-15 23:55:03 NaT                  Night          No Cabs Available 0

3238 rows × 9 columns
```

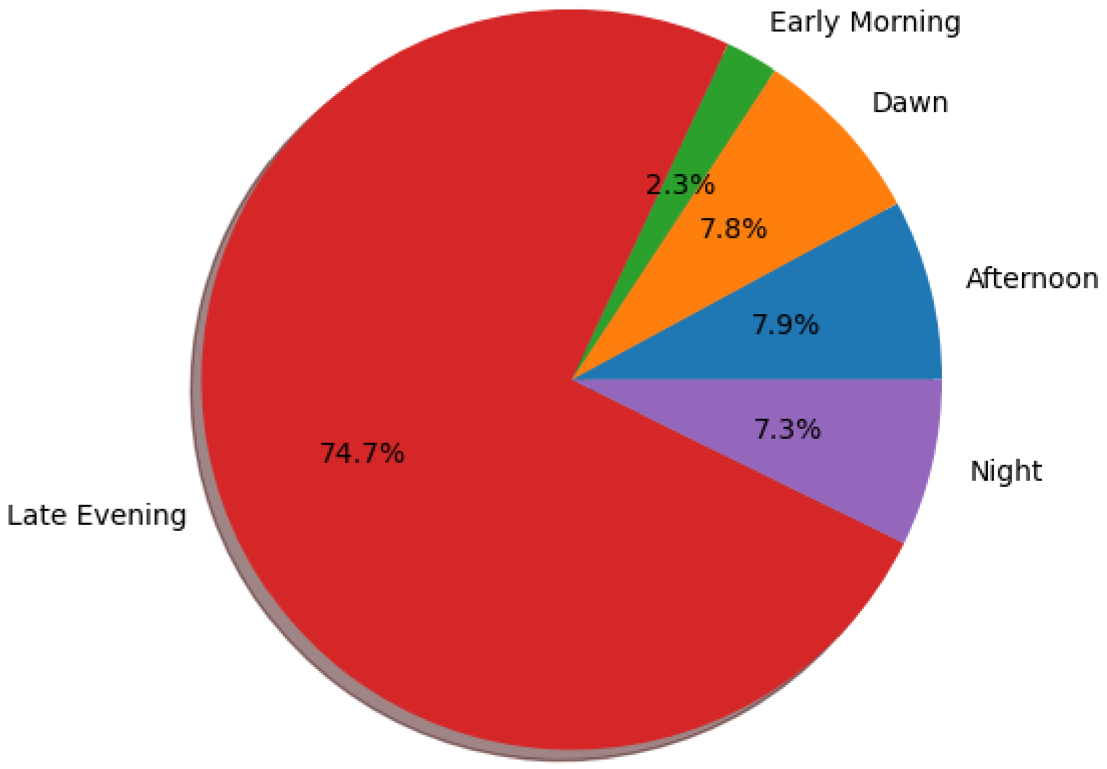
In [123]:
```
# Count of requests coming from users in a day when they opt to travel to from Airport to City

plt.figure(figsize = (10,6))
sns.countplot(data = airport_pickups, x = 'Time_period', palette=["#000000"])
plt.show()
```



In [124]:
```
# Cabs Availability in a day when users opt to travel to City from Airport

# Blank indicates ride is completed
# Red indicates either cancelled or no cabs available at the point of time

plt.figure(figsize = (10,6))
sns.countplot(data = airport_pickups, x = 'Time_period', hue = 'Status' )
plt.legend(loc = 'upper right', frameon = True, ncol = 2)
plt.show()
```

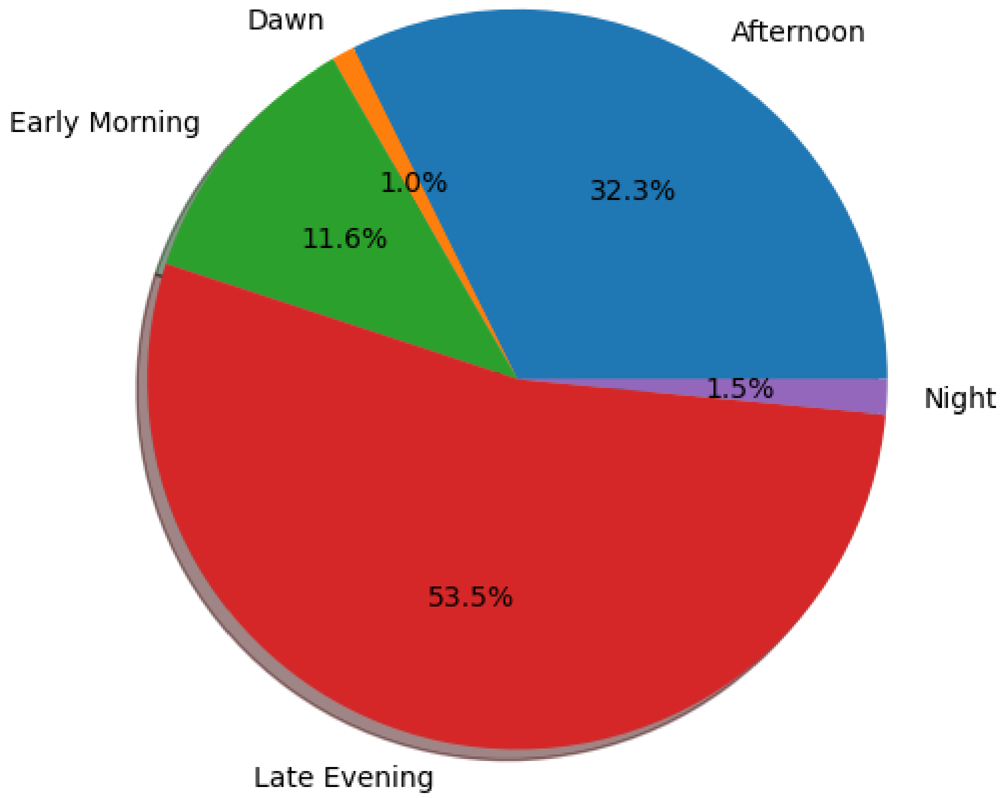

Trip Completed
Cancelled
No Cars Available
count
0
200
400
600
800
1000
1200
Afternoon
Late Evening
Early Morning
Dawn
Night
Time_period

In [149]:
```
# Percentage of Airport pickups where cabs are not available

airport_pickups[ (airport_pickups["Cab_Availability"]=="No Cabs Available") ].groupby(['Time_period']).size().plot(kin
plt.ylabel("")
plt.show()
```



	Afternoon	Dawn	Early Morning	Late Evening	Night
None	151	150	44	1427	139

In [158]:
```
# Percentage of Airport pickups where drivers cancelled the ride

airport_pickups[ (airport_pickups["Status"]=="Cancelled") ].groupby(['Time_period']).size().plot(kind="pie", figsize=(
plt.ylabel("")
plt.show()
```



	Afternoon	Dawn	Early Morning	Late Evening	Night
None	64	2	23	106	3

In [125]:
```
# Percentage of requests in Morning time from Airport to City

airport_pickups.loc[airport_pickups['Time_period'] == 'Early Morning','Status'].value_counts() / len(df) * 100
```

Out[125]:
```
Trip Completed       5.663454
Cancelled            0.340993
No Cars Available    0.311342
Name: Status, dtype: float64
```

In [126]:
```
# Percentage of requests in Late Evening time from Airport to City

airport_pickups.loc[airport_pickups['Time_period'] == 'Late Evening','Status'].value_counts() / len(df) * 100
```

Out[126]:
```
No Cars Available    19.584878
Trip Completed        5.530022
Cancelled             1.571534
Name: Status, dtype: float64
```

In [127]:
```
# Percentage of requests in Night time from Airport to City

airport_pickups.loc[airport_pickups['Time_period'] == 'Night','Status'].value_counts() / len(df) * 100
```

Out[127]:
```
Trip Completed       2.105263
No Cars Available    2.016308
Cancelled            0.044477
Name: Status, dtype: float64
```

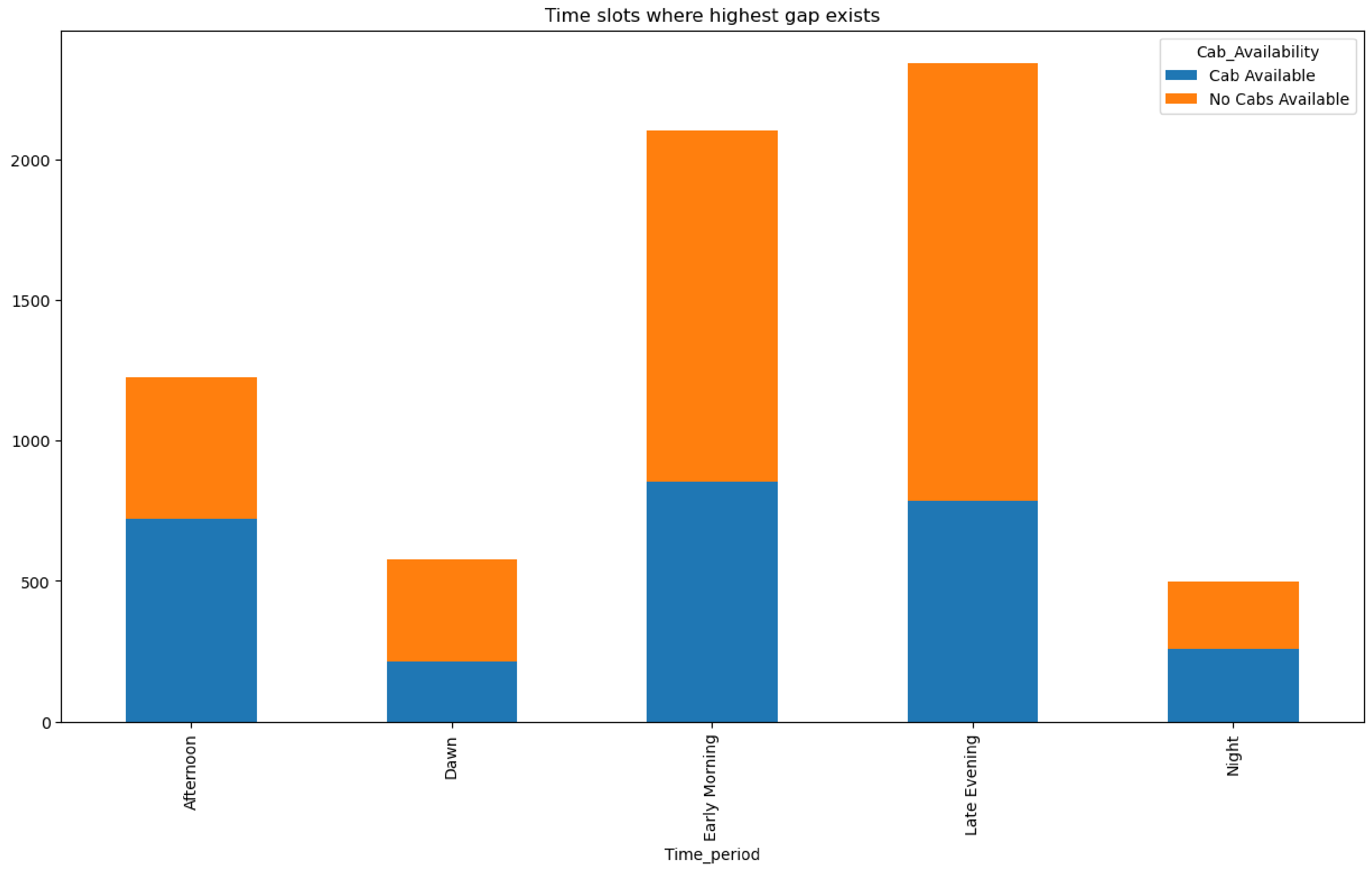
In [128]:
```
# Percentage of requests in Dawn time from Airport to City

airport_pickups.loc[airport_pickups['Time_period'] == 'Dawn','Status'].value_counts() / len(df) * 100
```

Out[128]:
```
No Cars Available    2.194218
Trip Completed       1.527057
Cancelled            0.029652
Name: Status, dtype: float64
```

In [129]:
```
# Percentage of requests in Afternoon time from Airport to City

airport_pickups.loc[airport_pickups['Time_period'] == 'Afternoon','Status'].value_counts() / len(df) * 100
```

Out[129]:
```
Trip Completed       4.848036
No Cars Available    1.289844
Cancelled            0.948851
Name: Status, dtype: float64
```

In [167]:
```
# Highest gap between the rides

# Time slots where highest gap exists -

df.groupby(['Time_period','Cab_Availability']).size().unstack().plot(kind='bar', stacked=True,figsize=(15, 8))
plt.title('Time slots where highest gap exists')
plt.show()
```


Time slots where highest gap exists
Cab_Availability
Cab Available
No Cabs Available
0
500
1000
1500
2000
Afternoon
Dawn
Early Morning
Late Evening
Night
Time_period

In [168]:
```
df.groupby(['Time_period','Cab_Availability']).size().unstack()
```

Out[168]:

Cab_Availability	Cab Available	No Cabs Available
Time_period		
Afternoon	722	502
Dawn	214	364
Early Morning	854	1249
Late Evening	784	1558
Night	257	241

In [ ]:
```
# Unstack and Stack

# Unstack function pivots the data, transforming grouped series into a data frame where the cab availability value bec
# multiple columns and Time period column becomes the data frame index.
# This operation effectively separates the counts of cab availability into seperate columns allowing use to plot a sta
# bar chart with each column representing a different cab availability value

# On the other side stack cannot be used here because
# if we want to use stack we should create a data frame first with the result of group by or that sort and then use st
# that data frame. The stack method is used to pivot the data frame from a wider format(multi level columns) to long f
# creating a multi-level index

# Using stack in this context would not be appropriate for creating a stacked bar chart since it would create a multi-
# making the plot more complicated. In most cases, you would want to use unstack when preparing data for a stacked bar
# have each category (in this case, 'Cab_Availability') represented by a separate column in the resulting DataFrame, a
# plot the stacked bars accordingly

df.groupby(['Time_period','Cab_Availability']).size().stack().plot(kind='bar', stacked=True, figsize=(15, 8))
```

In [130]:
```
# Checking for a relation between Time period and Cab Availability ?
# Are they linearly dependent on each other ?

# Assuming 'Cab Availability' is the column containing Cab availability in the Uber DataFrame.
# 'Time period' is the column containing information on desired requests time in a day.

# Since we have two object based columns i.e strings, we can use Chi-square test.
# For example:

# Create a contingency table
contingency_table = pd.crosstab(df['Time_period'], df['Cab_Availability'])

# Perform the Chi-Square test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Define the significance level (alpha)
alpha = 0.05

# Check the p-value against the significance level to make a decision
if p_value < alpha:
    print("Reject the null hypothesis. Time is significantly related to Cab Availability.")
else:
    print("Fail to reject the null hypothesis. Time does not significantly improve Cab Availability")
```

```
Reject the null hypothesis. Time is significantly related to Cab Availability.
```

In [33]:
```
df
```

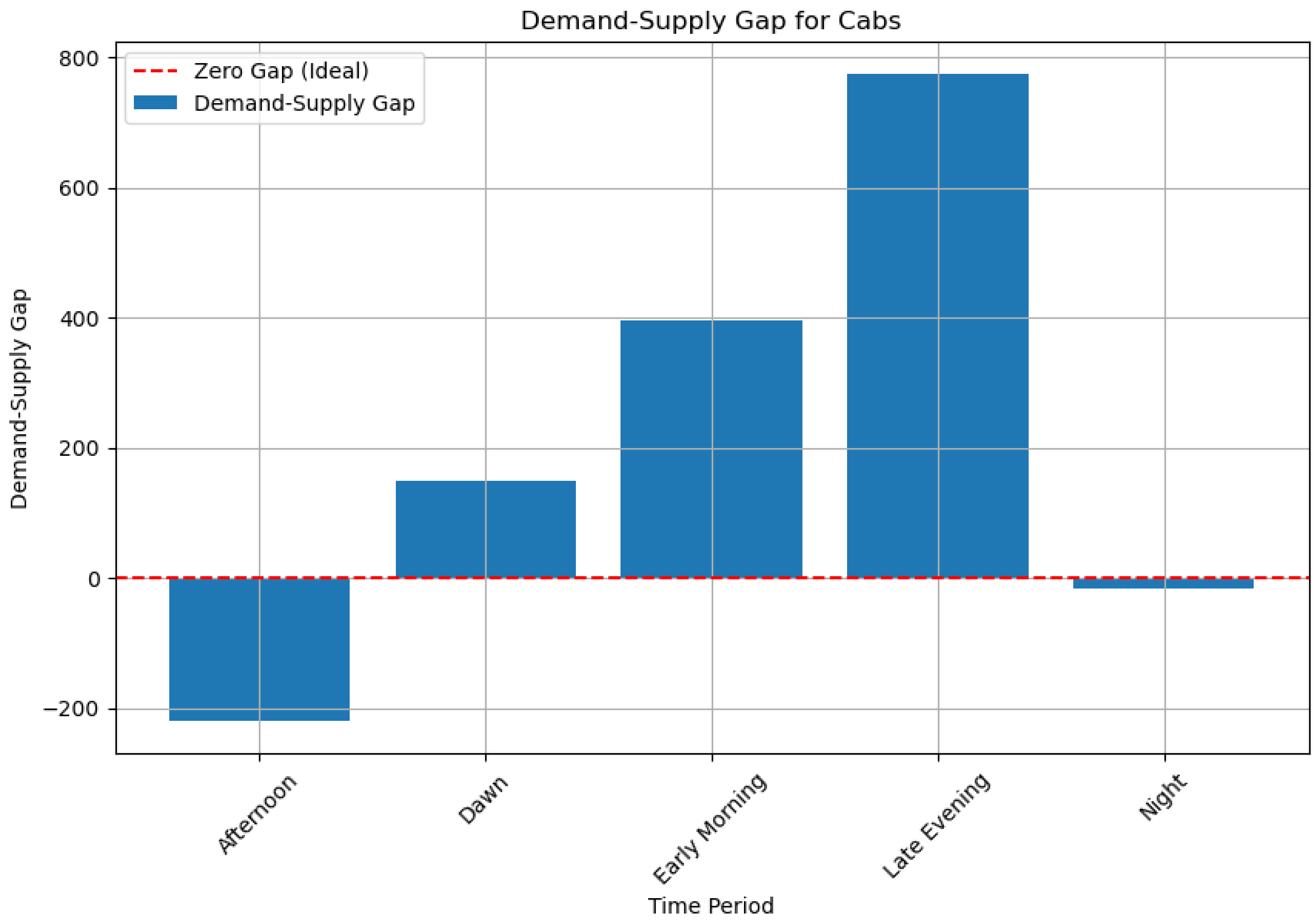
Out[33]:

	Request id	Pickup point	Driver id	Status	Request timestamp	Drop timestamp	Time_period	Cab_Availability
0	619	Airport	1.0	Trip Completed	2016-07-11 11:51:00	2016-07-11 13:00:00	Early Morning	Cab Available
1	867	Airport	1.0	Trip Completed	2016-07-11 17:57:00	2016-07-11 18:47:00	Late Evening	Cab Available
2	1807	City	1.0	Trip Completed	2016-07-12 09:17:00	2016-07-12 09:58:00	Early Morning	Cab Available
3	2532	Airport	1.0	Trip Completed	2016-07-12 21:08:00	2016-07-12 22:03:00	Night	Cab Available
4	3112	City	1.0	Trip Completed	2016-07-13 08:33:16	2016-07-13 09:25:47	Early Morning	Cab Available
...	...	...	...	...	...	...	...	...
6740	6745	City	NaN	No Cars Available	2016-07-15 23:49:03	NaT	Night	No Cabs Available
6741	6752	Airport	NaN	No Cars Available	2016-07-15 23:50:05	NaT	Night	No Cabs Available
6742	6751	City	NaN	No Cars Available	2016-07-15 23:52:06	NaT	Night	No Cabs Available
6743	6754	City	NaN	No Cars Available	2016-07-15 23:54:39	NaT	Night	No Cabs Available
6744	6753	Airport	NaN	No Cars Available	2016-07-15 23:55:03	NaT	Night	No Cabs Available

6745 rows × 8 columns

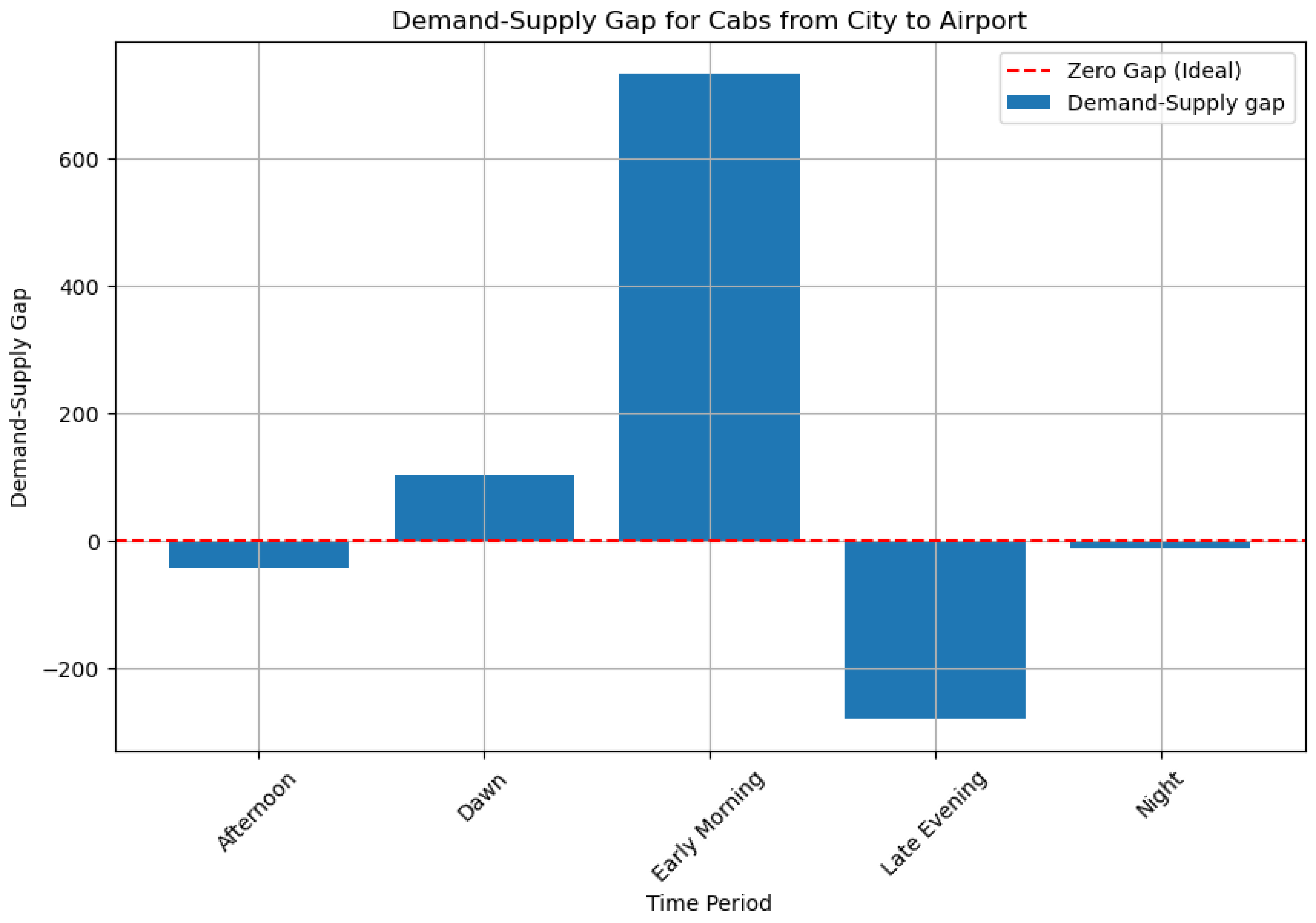
In [178]:
```
# Overall Scenario

# How many cabs should be ideally available according to the Time period to suffice the customer complaints

#Calculate the estimated demand for cabs during each time period

# Count the number of rides (demand) during each time period
supply_by_time = df[df['Cab_Availability'] == 'Cab Available'].groupby('Time_period').size().reset_index(name='Supply'

# Count the number of non-availability (supply) during each time period
demand_by_time = df[df['Cab_Availability'] == 'No Cabs Available'].groupby('Time_period').size().reset_index(name='Der

merged_df = pd.merge(demand_by_time, supply_by_time, on='Time_period', how='outer').fillna(0)

merged_df['Demand_Supply_Gap'] = merged_df['Demand'] - merged_df['Supply']

print(merged_df)

plt.figure(figsize=(10, 6))
plt.bar(merged_df['Time_period'], merged_df['Demand_Supply_Gap'], label='Demand-Supply Gap')
plt.axhline(0, color='red', linestyle='--', label='Zero Gap (Ideal)')
plt.xlabel('Time Period')
plt.ylabel('Demand-Supply Gap')
plt.title('Demand-Supply Gap for Cabs')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

```
     Time_period  Demand  Supply  Demand_Supply_Gap
0      Afternoon     502     722               -220
1           Dawn     364     214                150
2  Early Morning    1249     854                395
3   Late Evening    1558     784                774
4          Night     241     257                -16
```


Demand-Supply Gap for Cabs
Zero Gap (Ideal)
Demand-Supply Gap
800
600
400
200
0
−200
Demand-Supply Gap
Afternoon
Dawn
Early Morning
Late Evening
Night
Time Period

In [189]:
```
# City Pickup Scenario

demand_surge = city_pickups[city_pickups['Cab_Availability'] == 'No Cabs Available'].groupby('Time_period').size().res

supply_provided = city_pickups[city_pickups['Cab_Availability'] == 'Cab Available'].groupby('Time_period').size().rese

merged_df = pd.merge(demand_surge, supply_provided, on = 'Time_period', how = 'outer').fillna(0)

merged_df['Demand_Supply_gap'] = merged_df['Demand'] - merged_df['Supply']

print(merged_df)

plt.figure(figsize=(10, 6))
plt.bar(merged_df['Time_period'], merged_df['Demand_Supply_gap'], label='Demand-Supply gap')
plt.axhline(0, color='red', linestyle='--', label='Zero Gap (Ideal)')
plt.xlabel('Time Period')
plt.ylabel('Demand-Supply Gap')
plt.title('Demand-Supply Gap for Cabs from City to Airport')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

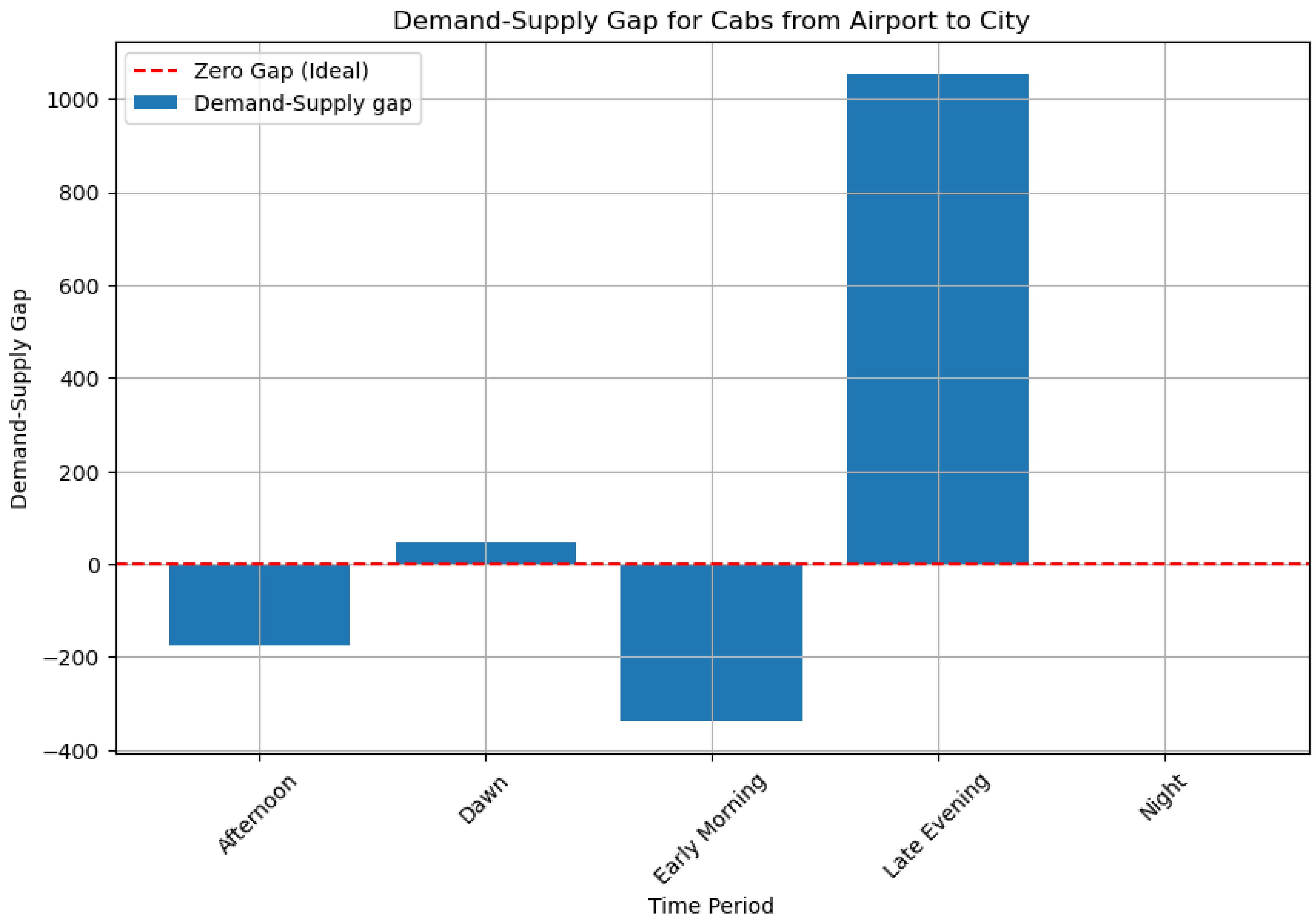
```
     Time_period  Demand  Supply  Demand_Supply_gap
0      Afternoon     351     395                -44
1           Dawn     214     111                103
2  Early Morning    1205     472                733
3   Late Evening     131     411               -280
4          Night     102     115                -13
```


Demand-Supply Gap for Cabs from City to Airport
Zero Gap (Ideal)
Demand-Supply gap
Demand-Supply Gap
600
400
200
0
−200
Afternoon
Dawn
Early Morning
Late Evening
Night
Time Period

In [187]:
```
# Explore on reset index with name paramter

# demand_surge = city_pickups[city_pickups['Cab_Availability'] == 'No Cabs Available'].groupby('Time_period').size().r
# demand_surge
```

In [188]:
```
# Airport Pickup Scenario

demand_surge = airport_pickups[airport_pickups['Cab_Availability'] == 'No Cabs Available'].groupby('Time_period').size

supply_provided = airport_pickups[airport_pickups['Cab_Availability'] == 'Cab Available'].groupby('Time_period').size(

merged_df = pd.merge(demand_surge, supply_provided, on = 'Time_period', how = 'outer').fillna(0)

merged_df['Demand_Supply_gap'] = merged_df['Demand'] - merged_df['Supply']

print(merged_df)

plt.figure(figsize=(10, 6))
plt.bar(merged_df['Time_period'], merged_df['Demand_Supply_gap'], label='Demand-Supply gap')
plt.axhline(0, color='red', linestyle='--', label='Zero Gap (Ideal)')
plt.xlabel('Time Period')
plt.ylabel('Demand-Supply Gap')
plt.title('Demand-Supply Gap for Cabs from Airport to City')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

```
     Time_period  Demand  Supply  Demand_Supply_gap
0      Afternoon     151     327               -176
1           Dawn     150     103                 47
2  Early Morning      44     382               -338
3   Late Evening    1427     373               1054
4          Night     139     142                 -3
```


Demand-Supply Gap for Cabs from Airport to City
Zero Gap (Ideal)
Demand-Supply gap
1000
800
600
400
200
0
−200
−400
Demand-Supply Gap
Afternoon
Dawn
Early Morning
Late Evening
Night
Time Period