

# Final Project for Grand Circus Coding Bootcamp - Data & Analytics

## *UK Open Rail Data Analysis*

All projects begin with the importing of libraries needed throughout the project.

```
In [ ]: import numpy as np
import pandas as pd
import statistics

import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline

import sqlalchemy
from sqlalchemy import create_engine, text

import psycopg2
```

This block loaded the connection between our AWS Server and Jupyter Notebook so that our data cleaning and transformation steps were possible.

```
In [ ]: from sqlalchemy import create_engine, text
import pandas as pd

# the connection URL
connection_url = 'postgresql://root:u2YJR7TNLPP1THVLsPS@database-1.cd0asuee8tgm.us

# creating an engine
engine = create_engine(connection_url)
query = text('SELECT * FROM darwin')
darwin_df = pd.read_sql(query, engine)
```

These are the types of data that we were working with in this project.

```
In [ ]: data_types = darwin_df.dtypes
print(data_types)
```

```
route_id          object
unique_id         object
service_start_date object
update_origin     object
train_platform    object
working_time_pass object
working_time_arrival object
working_time_departure object
planned_time_arrival object
planned_time_departure object
actual_arrival_time object
actual_departure_time object
platform          object
train_length      float64
estimated_time    object
source            object
actual_time       object
actual_time_class object
is_delayed_arrival bool
is_delayed_departure bool
source_instance    object
estimated_time_minutes object
dtype: object
```

The shape of our database is displayed via this line of code.

```
In [ ]: darwin_df.shape
```

```
Out[ ]: (1326, 22)
```

```
In [ ]: darwin_df.head()
```

```
Out[ ]:
```

	route_id	unique_id	service_start_date	update_origin	train_platform	working_t
0	202403217105805	G05805	2024-03-21	Trust	WCLIFF	
1	202403217105805	G05805	2024-03-21	Trust	CHLKVEL	
2	202403217105798	G05798	2024-03-21	CIS	BARKING	
3	202403217604342	L04342	2024-03-21	TD	SURREYQ	
4	202403218731849	W31849	2024-03-21	None	GIDB	

5 rows × 22 columns



Continuing the data cleaning process by seeing if there are columns with no data, and removing those from the dataframe.

```
In [ ]: darwin_df.count()
```

```
Out[ ]: route_id          1326
unique_id          1326
service_start_date 1326
update_origin      1313
train_platform     1326
working_time_pass   0
working_time_arrival 1326
working_time_departure 1326
planned_time_arrival 1006
planned_time_departure 976
actual_arrival_time 1069
actual_departure_time 1181
platform           1174
train_length        514
estimated_time      0
source              0
actual_time         0
actual_time_class    0
is_delayed_arrival 1326
is_delayed_departure 1326
source_instance      0
estimated_time_minutes 0
dtype: int64
```

```
In [ ]: darwin_df.drop(columns=['working_time_pass', 'estimated_time', 'source', 'actual_time'])
```

Here we are loading in the rail references table to help identify abbreviations and starting the data cleaning process by dropping columns that have only null values.

```
In [ ]: rail_names = pd.read_csv("RailReferences.csv")
rail_names.head()
```

```
Out[ ]:   AtcoCode TiplocCode CrsCode StationName StationNameLang GridType Easting
0  9100PENZNCE    PENZNCE    PNZ  Penzance Rail
                           Station                    NaN       U  147588
1  9100STIVES     STIVES     SIV  St Ives
                           (Cornwall) Rail
                           Station                    NaN       U  151947
2  9100CARBISB    CARBISB    CBB  Carbis Bay
                           Rail Station                    NaN       U  152930
3  9100STERTH     STERTH     SER  St Erth Rail
                           Station                    NaN       U  154150
4  9100LELANTS    LELANTS    LTS  Lelant
                           Saltings Rail
                           Station                    NaN       U  154430
```

We removed unnecessary columns from the station name reference table as well.

```
In [ ]: rail_names = rail_names.loc[:, rail_names.columns.intersection(['TiplocCode', 'StationName'])]
rail_names.head()
```

Out[ ]:

	TiplocCode	StationName
0	PENZNCE	Penzance Rail Station
1	STIVES	St Ives (Cornwall) Rail Station
2	CARBISB	Carbis Bay Rail Station
3	STERTH	St Erth Rail Station
4	LELANTS	Lelant Saltings Rail Station

In this step, we merged the dataframe with the reference table. This was to help better identify train platforms while working on this analysis.

```
In [ ]: darwin_df = pd.merge(darwin_df, rail_names, how='inner', left_on='train_platform',
darwin_df.head())
```

Out[ ]:

	route_id	unique_id	service_start_date	update_origin	train_platform	working_t
0	202403217105805	G05805	2024-03-21	Trust	WCLIFF	
1	202403217105805	G05805	2024-03-21	Trust	CHLKWEL	
2	202403217105798	G05798	2024-03-21	CIS	BARKING	
3	202403217604342	L04342	2024-03-21	TD	SURREYQ	
4	202403218731849	W31849	2024-03-21	None	GIDB	

Dropping columns that are not relevant for answering the question.

```
In [ ]: delays = darwin_df.drop(columns=['service_start_date', 'planned_time_arrival', 'pla
          'TiplocCode', 'unique_id', 'update_origin', 'working_t
          'train_length', 'platform'])
```

```
In [ ]: delays.head()
```

Out[ ]:

	route_id	train_platform	is_delayed_arrival	is_delayed_departure	StationName
0	202403217105805	WCLIFF	False	False	Westcliff-on-Sea Rail Station
1	202403217105805	CHLKVEL	False	False	Chalkwell Rail Station
2	202403217105798	BARKING	False	False	Barking Rail Station
3	202403217604342	SURREYQ	False	False	Surrey Quays Rail Station
4	202403218731849	GIDB	False	False	Guide Bridge Rail Station

## Contributors' metrics

In [ ]:

```
# number of contributions from each origin
origin_counts = darwin_df['update_origin'].value_counts()

# the counts
print(origin_counts)
```

```
update_origin
TD      605
Darwin   246
CIS      237
Trust     78
Name: count, dtype: int64
```

In [ ]:

```
origin_df = darwin_df.dropna(subset=['update_origin'])
origin_df.head()
```

Out[ ]:

	route_id	unique_id	service_start_date	update_origin	train_platform	working
0	202403217105805	G05805	2024-03-21	Trust	WCLIFF	
1	202403217105805	G05805	2024-03-21	Trust	CHLKVEL	
2	202403217105798	G05798	2024-03-21	CIS	BARKING	
3	202403217604342	L04342	2024-03-21	TD	SURREYQ	
11	202403217105804	G05804	2024-03-21	Trust	UPMNSTR	



## Determining the different contributors of information and creating a separate dataframe for each

```
In [ ]: origin_df.update_origin.unique()
```

```
Out[ ]: array(['Trust', 'CIS', 'TD', 'Darwin'], dtype=object)
```

```
In [ ]: trust_origin = origin_df[origin_df.update_origin == 'Trust']
CIS_origin = origin_df[origin_df.update_origin == 'CIS']
TD_origin = origin_df[origin_df.update_origin == 'TD']
darwin_origin = origin_df[origin_df.update_origin == 'Darwin']
```

```
In [ ]: print('Trust: ', trust_origin.shape)
print('CIS: ', CIS_origin.shape)
print('TD: ', TD_origin.shape)
print('Darwin: ', darwin_origin.shape)
```

```
Trust: (78, 17)
CIS: (237, 17)
TD: (605, 17)
Darwin: (246, 17)
```

```
In [ ]: trust_null_count = trust_origin.isnull().sum().sum()
CIS_null_count = CIS_origin.isnull().sum().sum()
TD_null_count = TD_origin.isnull().sum().sum()
darwin_null_count = darwin_origin.isnull().sum().sum()

print('Trust: ', trust_null_count)
print('CIS: ', CIS_null_count)
print('TD: ', TD_null_count)
print('Darwin: ', darwin_null_count)
```

```
Trust: 103
CIS: 173
TD: 816
Darwin: 463
```

Using the null counts to find the percentage of null values provided by each contributor. In effect, identifying which origin source contributed the least.

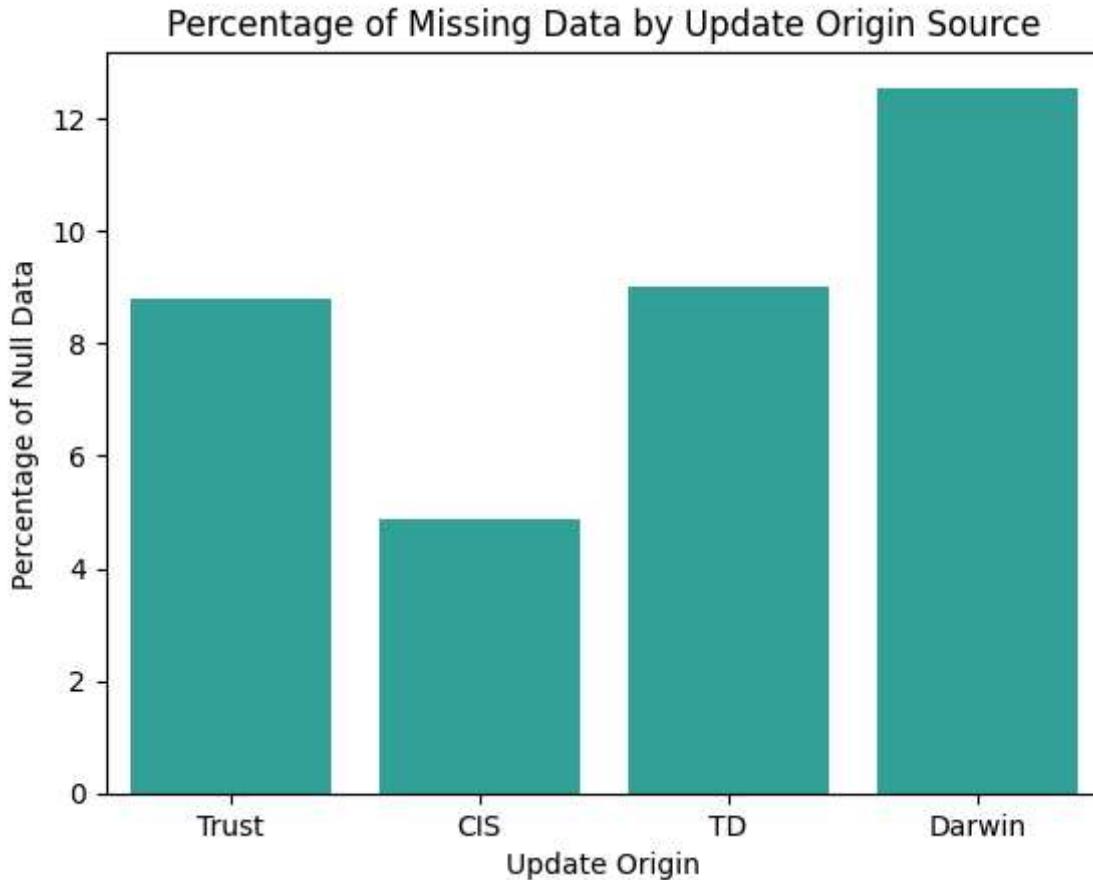
```
In [ ]: trust_cell_count = 78 * 15
CIS_cell_count = 237 * 15
TD_cell_count = 605 * 15
darwin_cell_count = 246 * 15
trust_null_by_df = (trust_null_count / trust_cell_count) * 100
CIS_null_by_df = (CIS_null_count / CIS_cell_count) * 100
TD_null_by_df = (TD_null_count / TD_cell_count) * 100
darwin_null_by_df = (darwin_null_count / darwin_cell_count) * 100
trust_null_by_event = (trust_null_count / 78) * 100
CIS_null_by_event = (CIS_null_count / 237) * 100
TD_null_by_event = (TD_null_count / 605) * 100
darwin_null_by_event = (darwin_null_count / 246) * 100
```

```
In [ ]: data = [['Trust', trust_null_count, trust_null_by_df, trust_null_by_event], ['CIS', origin_reliability_df = pd.DataFrame(data, columns=['Update Origin', 'Total Null Va  
origin_reliability_df
```

	Update Origin	Total Null Values	DataFrame Null Percentage	Event Null Percentage
0	Trust	103	8.803419	132.051282
1	CIS	173	4.866385	72.995781
2	TD	816	8.991736	134.876033
3	Darwin	463	12.547425	188.211382

```
In [ ]: sns.barplot(origin_reliability_df, x='Update Origin', y='DataFrame Null Percentage'  
plt.title('Percentage of Missing Data by Update Origin Source')  
plt.ylabel('Percentage of Null Data')
```

```
Out[ ]: Text(0, 0.5, 'Percentage of Null Data')
```



## Looking at the Top 10 busiest stations

```
In [ ]: counts = delays.groupby(['StationName', 'train_platform'])[['is_delayed_arrival']].a
```

```
In [ ]: #Creating a new database so it's easier to graph out.  
delayedarrival = pd.DataFrame(counts)
```

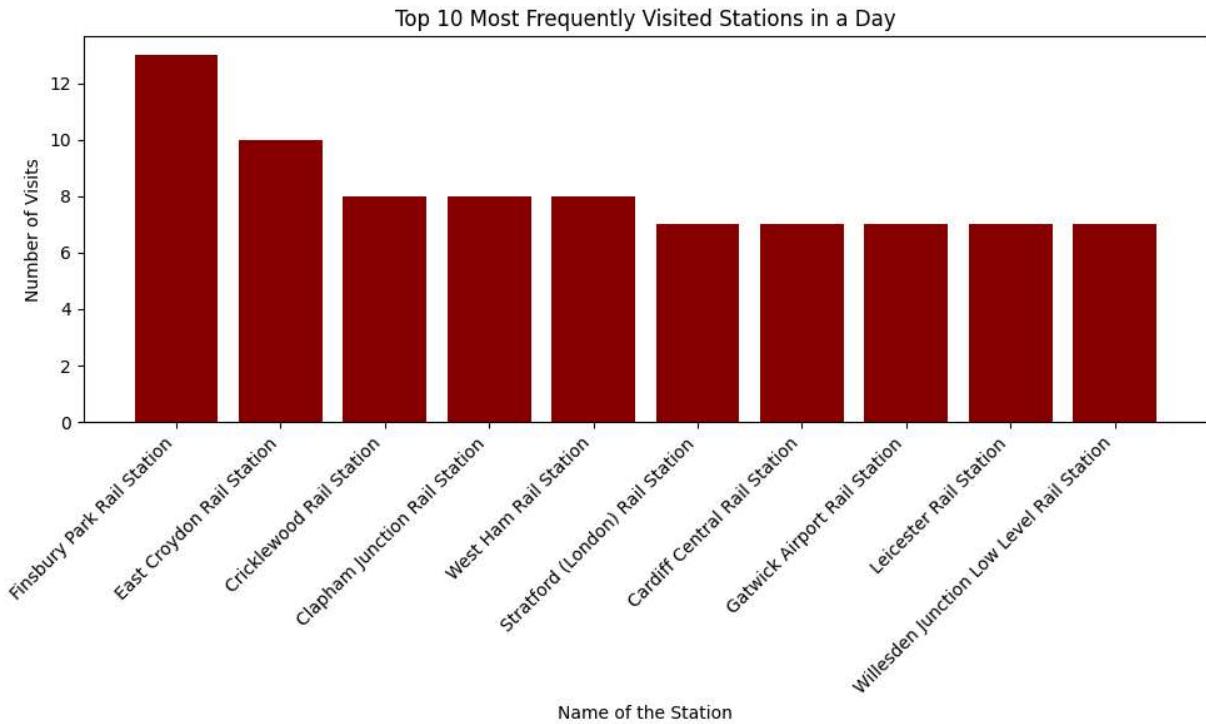
Sorting the new database out based on top 10 most on time stations.

```
In [ ]: top_false_values = delayedarrival.sort_values(by=False, ascending=False).head(10)  
top_false_values.head(10)
```

Out[ ]:

StationName	train_platform	is_delayed_arrival	False	True
Finsbury Park Rail Station	FNPK	12.0	1.0	
East Croydon Rail Station	ECROYDN	9.0	1.0	
West Ham Rail Station	WHAMHL	8.0	0.0	
Limehouse Rail Station	LIMHSE	7.0	0.0	
Cardiff Central Rail Station	CRDFCEN	7.0	0.0	
Stratford (London) Rail Station	STFD	7.0	0.0	
Gatwick Airport Rail Station	GTWK	6.0	1.0	
Clapham Junction Rail Station	CLPHMJM	6.0	0.0	
Luton Airport Parkway Rail Station	LUTOAPY	6.0	0.0	
Wimbledon Rail Station	WDON	6.0	0.0	

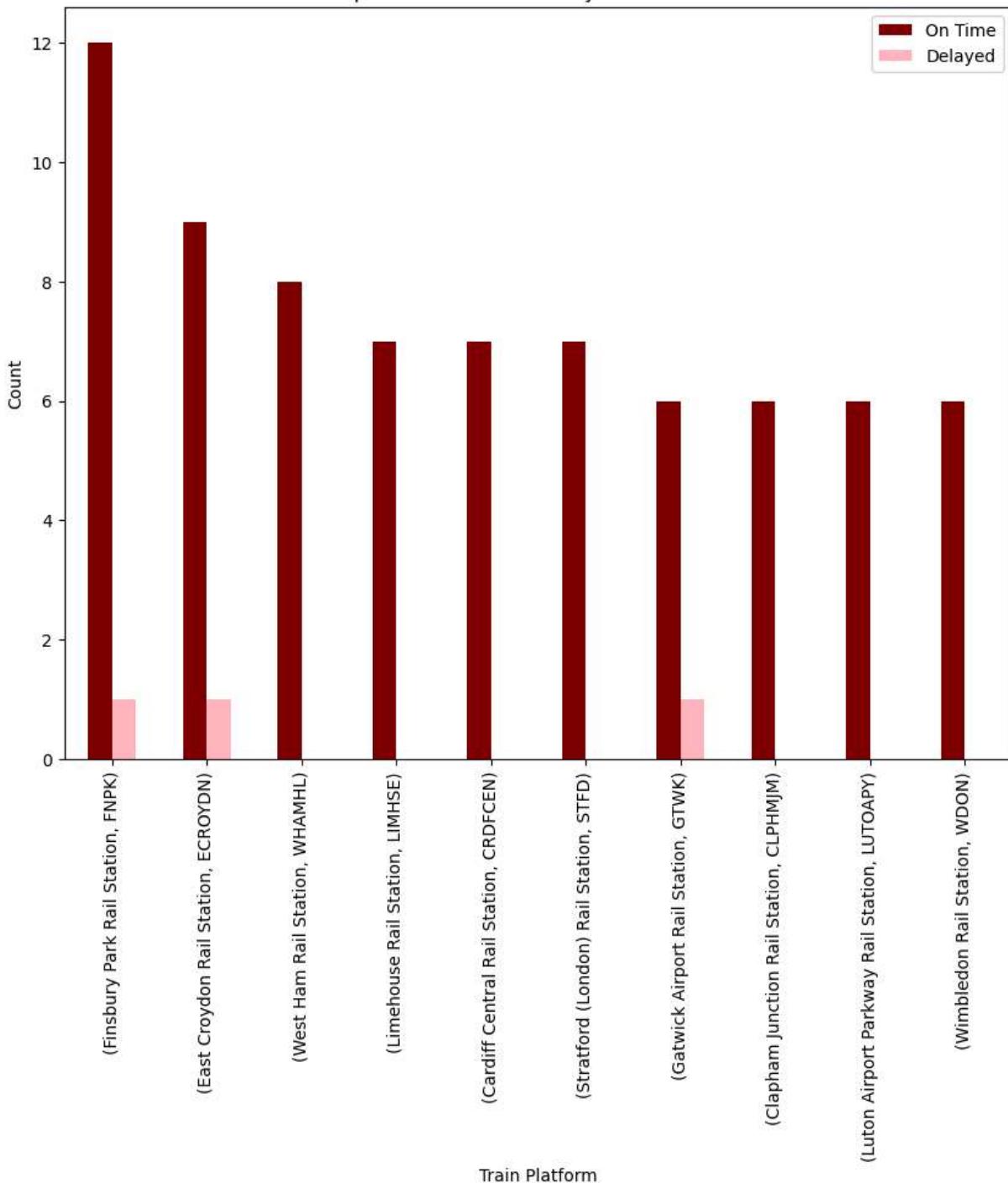
```
In [ ]: station_visits = darwin_df.groupby('StationName').size().reset_index(name='visit_count')  
  
station_visits_sorted = station_visits.sort_values(by='visit_count', ascending=False)  
  
top_10_stations = station_visits_sorted.head(10)  
  
  
plt.figure(figsize=(10, 6))  
plt.bar(top_10_stations['StationName'], top_10_stations['visit_count'], color='darkred')  
plt.xlabel('Name of the Station')  
plt.ylabel('Number of Visits')  
plt.title('Top 10 Most Frequently Visited Stations in a Day')  
plt.xticks(rotation=45, ha='right')  
plt.tight_layout()  
plt.show()
```



**How frequently the top 10 most used stations are running on time**

```
In [ ]: top_false_values.plot(kind='bar', stacked=False, figsize=(10, 8), color = ['maroon']
plt.title('Top 10 on Time Counts By Train Platform')
plt.xlabel('Train Platform')
plt.ylabel('Count')
plt.legend(['On Time', 'Delayed'])
plt.xticks(rotation=90)
plt.show()
```

Top 10 on Time Counts By Train Platform



Instead of sorting by false values, the data is now being sorted by true values to see which of the busiest stations see the most delays.

```
In [ ]: top_true_values = delayedarrival.sort_values(by=True, ascending=False).head(10)
```

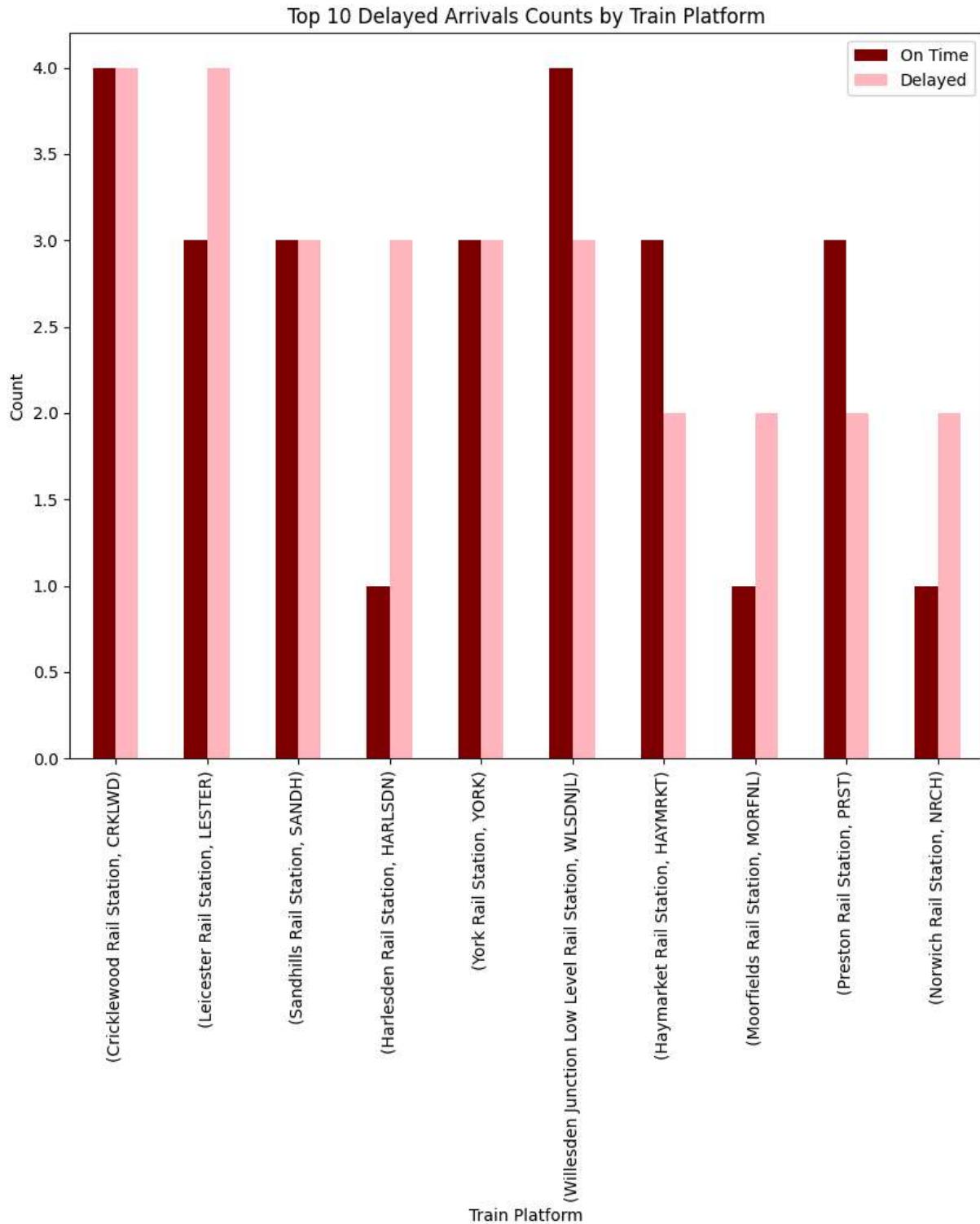
```
In [ ]: top_true_values.head(10)
```

Out[ ]:

		is_delayed_arrival	False	True
	StationName	train_platform		
	Cricklewood Rail Station	CRKLWD	4.0	4.0
	Leicester Rail Station	LESTER	3.0	4.0
	Sandhills Rail Station	SANDH	3.0	3.0
	Harlesden Rail Station	HARLSDN	1.0	3.0
	York Rail Station	YORK	3.0	3.0
	Willesden Junction Low Level Rail Station	WLSDNJL	4.0	3.0
	Haymarket Rail Station	HAYMRKT	3.0	2.0
	Moorfields Rail Station	MORFNL	1.0	2.0
	Preston Rail Station	PRST	3.0	2.0
	Norwich Rail Station	NRCH	1.0	2.0

In [ ]:

```
top_true_values.plot(kind='bar', stacked=False, figsize=(10, 8), color = ['maroon',  
plt.title('Top 10 Delayed Arrivals Counts by Train Platform')  
plt.xlabel('Train Platform')  
plt.ylabel('Count')  
plt.legend(['On Time', 'Delayed'])  
plt.xticks(rotation=90)  
plt.show()
```



## Analysis on the Relationship between Train Length and Delayed Arrivals

Again created a new database to work with to answer the question of the relationship between train length and delayed arrivals.

```
In [ ]: lengthtime = darwin_df.drop(columns=['route_id', 'service_start_date', 'planned_time',  
                                         'TiplocCode', 'unique_id', 'update_origin', 'working_t  
lengthtime = lengthtime.dropna()  
lengthtime = pd.DataFrame(lengthtime)
```

```
In [ ]: lengthtime.head()
```

```
Out[ ]:
```

	train_length	is_delayed_arrival	StationName
0	8.0	False	Westcliff-on-Sea Rail Station
1	8.0	False	Chalkwell Rail Station
2	4.0	False	Barking Rail Station
4	3.0	False	Guide Bridge Rail Station
5	3.0	False	Flowery Field Rail Station

Finding the max train length to figure out how big the x-axis is going to be and how to organize it.

```
In [ ]: lengthtime.train_length.max()
```

```
Out[ ]: 12.0
```

To help answer this question, we are counting out how many values are true and false for each train length.

```
In [ ]: delaycounts = darwin_df.groupby('train_length')['is_delayed_arrival'].value_counts()
```

Displaying results for each category.

```
In [ ]: delaycounts
```

```
Out[ ]:
```

train_length	is_delayed_arrival	count
2.0	False	12
3.0	False	25
	True	2
4.0	False	61
	True	18
5.0	False	69
	True	1
6.0	False	51
	True	2
8.0	False	137
	True	3
9.0	False	1
10.0	False	57
	True	3
12.0	False	26
	True	1

Name: count, dtype: int64

Creating a new database based on the information above.

```
In [ ]: ld = pd.DataFrame({'train_length': [2,3,4,5,6,8,9,10,12], 'not_delayed': [12,25,66,
```

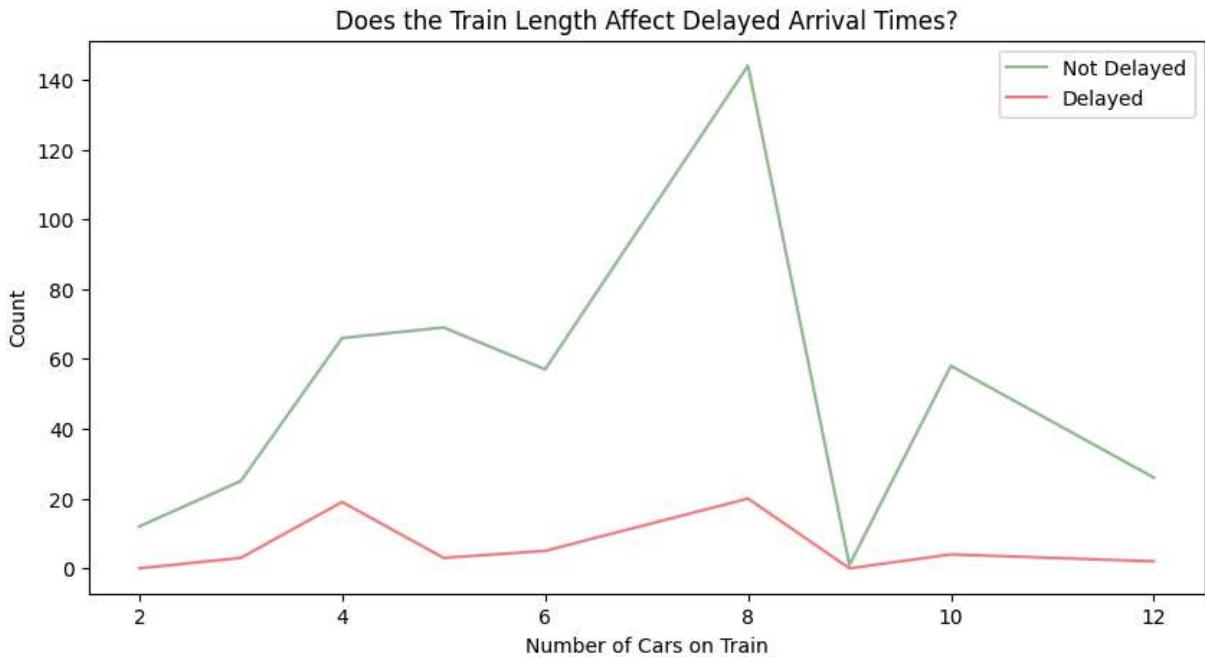
```
In [ ]: ld.head(9)
```

```
Out[ ]:
```

	train_length	not_delayed	delayed
0	2	12	0
1	3	25	3
2	4	66	19
3	5	69	3
4	6	57	5
5	8	144	20
6	9	1	0
7	10	58	4
8	12	26	2

Creating a line graph to visualize the trend easier and to reflect the results.

```
In [ ]: plt.figure(figsize=(10,5))
plt.plot('train_length', 'not_delayed', data=ld, color='darkseagreen', label = 'Not
plt.plot('train_length', 'delayed', data=ld, color='lightcoral', label='Delayed')
plt.title('Does the Train Length Affect Delayed Arrival Times?')
plt.xlabel('Number of Cars on Train')
plt.ylabel('Count')
plt.legend()
plt.show()
```



4 and 8 car trains specifically designed for high-density central area demand, based on UK law. We see a large spike in delays for 4 and 8 car trains for this reason.

How many of each quantity car length train was running?

```
In [ ]: train_length_counts = darwin_df['train_length'].value_counts()
print(train_length_counts)
```

```
train_length
8.0      140
4.0       79
5.0       70
10.0      60
6.0       53
3.0       27
12.0      27
2.0       12
9.0       1
Name: count, dtype: int64
```

Dropping columns that are not relevant to answering the question, "If the train arrives to the station late, does that necessarily mean a delayed departure?"

```
In [ ]: delayeddf = darwin_df.drop(columns=['service_start_date', 'planned_time_arrival',
                                             'TiplocCode', 'route_id', 'update_origin', 'working_time',
                                             'train_length', 'platform', 'train_platform'])
```

Replacing all values that say 'None' to be replaced with a null values so they can be dropped.

```
In [ ]: delayeddf.replace('None', np.nan, inplace = True)  
delayeddf = delayeddf.dropna()
```

```
In [ ]: delayeddf.head()
```

```
Out[ ]:   unique_id  is_delayed_arrival  is_delayed_departure  StationName
```

0	G05805	False	False	Westcliff-on-Sea Rail Station
1	G05805	False	False	Chalkwell Rail Station
2	G05798	False	False	Barking Rail Station
3	L04342	False	False	Surrey Quays Rail Station
4	W31849	False	False	Guide Bridge Rail Station

```
In [ ]: delayeddf.shape
```

```
Out[ ]: (1179, 4)
```

Getting a count for each scenario, whether both arrival and departure are on time, arrival is on time, but departure is late, or if arrival and departure are both delayed.

```
In [ ]: grouped = delayeddf.groupby(['is_delayed_arrival', 'is_delayed_departure']).size().
```

```
In [ ]: grouped.head(10)
```

```
Out[ ]:   is_delayed_arrival  is_delayed_departure  count
```

0	False	False	1067
1	False	True	15
2	True	True	97

Here, we opted to omit the row that reflects when departure and arrival are on time. With this new filter, the graph would better reflect how arrivals affect departures in a more concise way, since majority of the data states that most times, the train is always on time arriving and leaving.

```
In [ ]: departuretrue = grouped.iloc[1:3]
```

## Number of minutes spent at a station

```
In [ ]: downtime_df = darwin_df.dropna(subset=['actual_arrival_time', 'actual_departure_time'])  
downtime_df = downtime_df.loc[:, downtime_df.columns.intersection(['actual_arrival_time',  
actual_departure_time'])]  
downtime_df.head()
```

```
Out[ ]:
```

	actual_arrival_time	actual_departure_time	is_delayed_arrival
1	23:41:00	23:41:00	False
2	23:39:00	23:39:00	False
4	23:39:00	23:39:00	False
5	23:42:00	23:42:00	False
6	23:44:00	23:44:00	False

```
In [ ]: downtime_df['actual_arrival_time'] = pd.to_datetime(downtime_df['actual_arrival_time'])
downtime_df['actual_departure_time'] = pd.to_datetime(downtime_df['actual_departure_time'])

downtime_df.head()
```

```
Out[ ]:
```

	actual_arrival_time	actual_departure_time	is_delayed_arrival
1	1900-01-01 23:41:00	1900-01-01 23:41:00	False
2	1900-01-01 23:39:00	1900-01-01 23:39:00	False
4	1900-01-01 23:39:00	1900-01-01 23:39:00	False
5	1900-01-01 23:42:00	1900-01-01 23:42:00	False
6	1900-01-01 23:44:00	1900-01-01 23:44:00	False

```
In [ ]: downtime_df['time_diff'] = downtime_df['actual_departure_time'] - downtime_df['actual_arrival_time']

downtime_df.head()
```

```
Out[ ]:
```

	actual_arrival_time	actual_departure_time	is_delayed_arrival	time_diff
1	1900-01-01 23:41:00	1900-01-01 23:41:00	False	0 days
2	1900-01-01 23:39:00	1900-01-01 23:39:00	False	0 days
4	1900-01-01 23:39:00	1900-01-01 23:39:00	False	0 days
5	1900-01-01 23:42:00	1900-01-01 23:42:00	False	0 days
6	1900-01-01 23:44:00	1900-01-01 23:44:00	False	0 days

```
In [ ]: downtime_df['minutes_difference'] = downtime_df['time_diff'].dt.total_seconds() / 60
downtime_df['seconds_difference'] = downtime_df['time_diff'].dt.seconds

downtime_df.head()
```

	actual_arrival_time	actual_departure_time	is_delayed_arrival	time_diff	minutes_difference
1	1900-01-01 23:41:00	1900-01-01 23:41:00	False	0 days	0.
2	1900-01-01 23:39:00	1900-01-01 23:39:00	False	0 days	0.
4	1900-01-01 23:39:00	1900-01-01 23:39:00	False	0 days	0.
5	1900-01-01 23:42:00	1900-01-01 23:42:00	False	0 days	0.
6	1900-01-01 23:44:00	1900-01-01 23:44:00	False	0 days	0.

◀ ▶

```
In [ ]: minutes_counts = downtime_df.minutes_difference.value_counts()

top_downtime = minutes_counts.head(3)

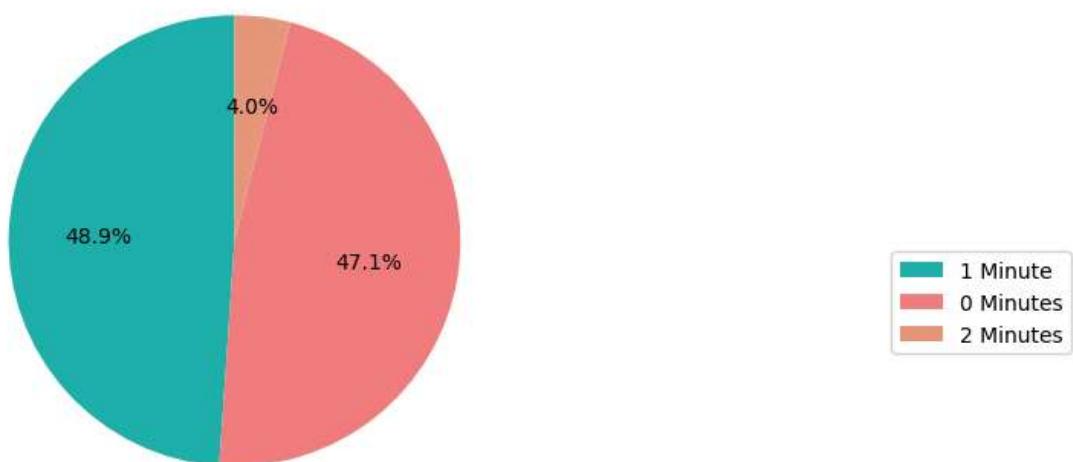
top_downtime
```

```
Out[ ]: minutes_difference
1.0    431
0.0    415
2.0     35
Name: count, dtype: int64
```

```
In [ ]: plt.pie(top_downtime, autopct='%.1f%%', startangle=90, labels=None, colors=['lightgreen', 'red', 'orange'])
plt.title('Average Time in Minutes Between Arrival and Departure')
plt.legend(labels=['1 Minute', '0 Minutes', '2 Minutes'], bbox_to_anchor=(2, 0.5))
```

```
Out[ ]: <matplotlib.legend.Legend at 0x1e0194f10a0>
```

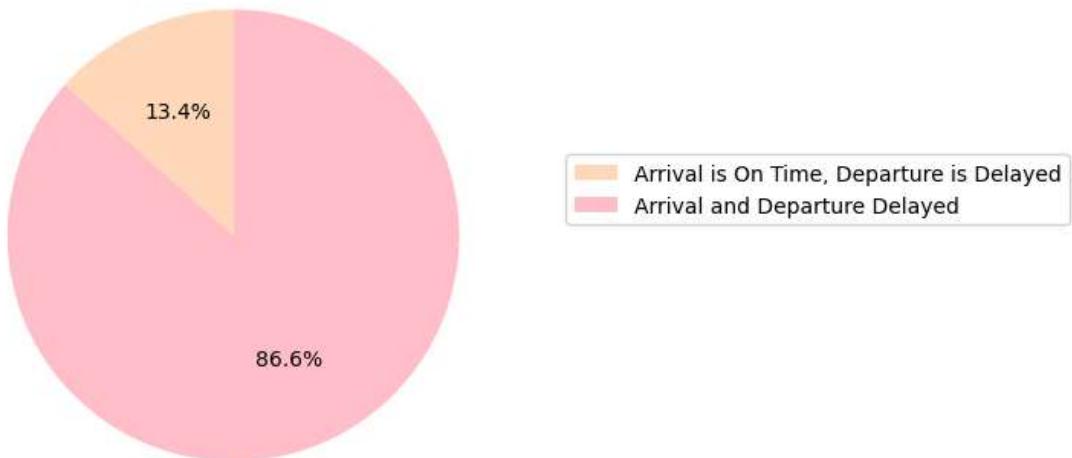
Average Time in Minutes Between Arrival and Departure



The short amount of time spent at the station, points to why the trains not being able to make up the time on their schedule.

```
In [ ]: #Create a pie chart to reflect the distribution of how arrival affects departures.  
plt.pie(departuretrue['count'], autopct='%.1f%%', startangle=90, labels = None, co  
plt.title("Does Delayed Arrival Always Cause Delayed Departure?")  
plt.legend(labels = ["Arrival is On Time, Departure is Delayed",  
"Arrival and Departure Delayed"], loc='lower right', bbox_to_anchor=(2, 0.5)  
plt.show()
```

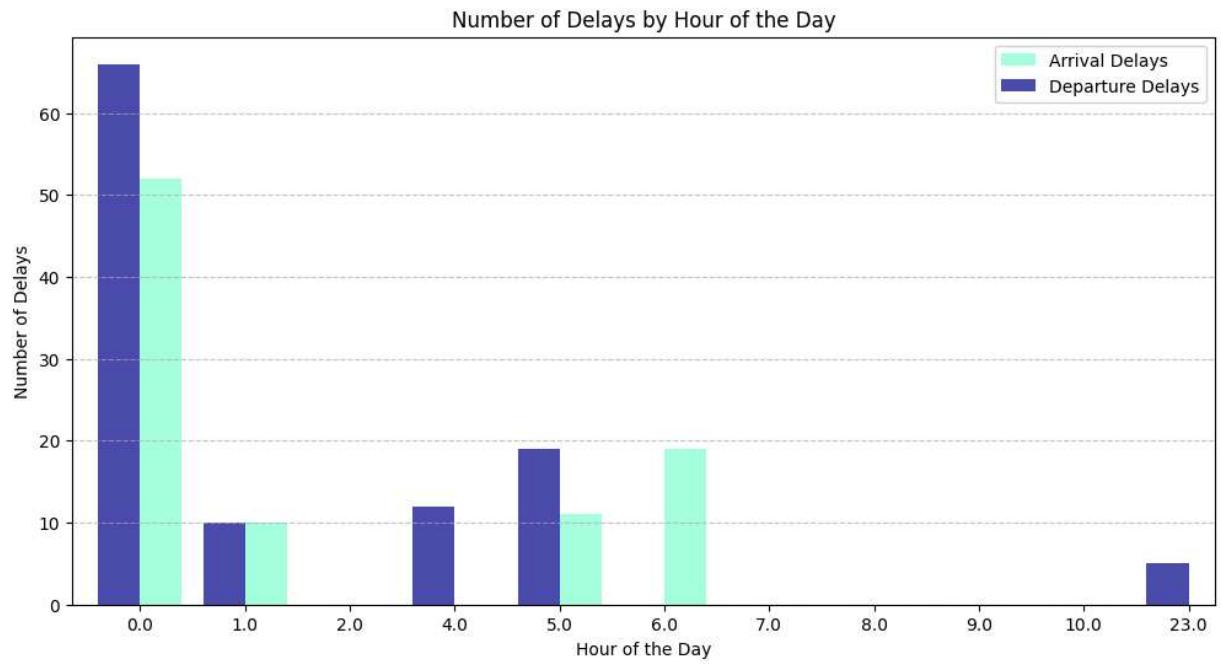
Does Delayed Arrival Always Cause Delayed Departure?



This proves a tendency to remain off schedule

```
In [ ]: from sklearn.metrics import mean_absolute_error, mean_squared_error, median_absolut  
  
# Convert to datetime objects  
darwin_df['actual_arrival_time'] = pd.to_datetime(darwin_df['actual_arrival_time'],  
darwin_df['actual_departure_time'] = pd.to_datetime(darwin_df['actual_departure_time'])  
  
# Extract the hour component  
darwin_df['arrival_hour'] = darwin_df['actual_arrival_time'].dt.hour  
darwin_df['departure_hour'] = darwin_df['actual_departure_time'].dt.hour  
  
# Group by hour and count delays  
arrival_delays_by_hour = darwin_df.groupby('arrival_hour')['is_delayed_arrival'].su  
departure_delays_by_hour = darwin_df.groupby('departure_hour')['is_delayed_departur  
  
plt.figure(figsize=(12, 6))  
width = 0.4 # Width of the bars  
  
# Plot arrival delays  
arrival_delays_by_hour.plot(kind='bar', color='aquamarine', alpha=0.7, width=width,  
  
# Plot departure delays  
departure_delays_by_hour.plot(kind='bar', color='darkblue', alpha=0.7, width=width,
```

```
plt.title('Number of Delays by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Delays')
plt.legend()
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



All delays during the time period we were able to procure, occurred overnight.

```
In [ ]: print(arrival_delays_by_hour)
          print(departure_delays_by_hour)
```

```

arrival_hour
0.0      52
1.0      10
2.0       0
3.0       0
4.0      11
5.0      19
6.0       0
7.0       0
8.0       0
9.0       0
10.0      0
23.0       5
Name: is_delayed_arrival, dtype: int64

departure_hour
0.0      66
1.0      10
2.0       0
4.0      12
5.0      19
6.0       0
7.0       0
8.0       0
9.0       0
10.0      0
23.0       5
Name: is_delayed_departure, dtype: int64

```

## Delayed Percentage vs No Delays

```

In [ ]: # Convert 'planned_time_departure' and 'actual_departure_time' columns to datetime
darwin_df['planned_time_departure'] = pd.to_datetime(darwin_df['planned_time_departure'])
darwin_df['actual_departure_time'] = pd.to_datetime(darwin_df['actual_departure_time'])

# Calculate absolute errors in seconds
darwin_df['abs_error_departure'] = (darwin_df['actual_departure_time'] - darwin_df['planned_time_departure']).dt.total_seconds()

# Convert absolute errors from seconds to minutes and create a new column
darwin_df['abs_error_departure_minutes'] = darwin_df['abs_error_departure'] / 60

# Count occurrences of zero errors
zero_errors_count = (darwin_df['abs_error_departure_minutes'] == 0).sum()

# Count occurrences of other errors
non_zero_errors_count = len(darwin_df) - zero_errors_count

# Calculate the percentage of true zeros
percentage_zeros = zero_errors_count / len(darwin_df) * 100

# Calculate the percentage of non-zero errors
percentage_non_zeros = non_zero_errors_count / len(darwin_df) * 100

# Create Labels and sizes for the pie chart
labels = ['No Delays', 'Delays']
sizes = [percentage_zeros, percentage_non_zeros]

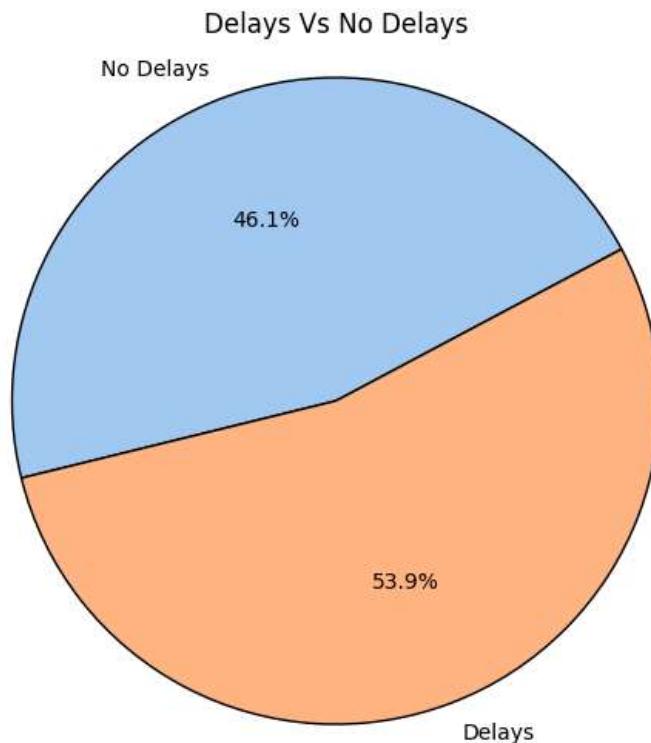
```

```

# Define colors for the pie chart
colors = sns.color_palette('pastel')[len(labels):]

# Create a pie chart
plt.figure(figsize=(10, 6))
plt.pie(sizes, labels=labels, autopct='%.1f%%', startangle=28, colors=colors, wedgeprops={'edgecolor': 'black'})
plt.title('Delays Vs No Delays')
plt.axis('equal')
plt.show()

```



## Distribution of delays

```

In [ ]: # Convert 'planned_time_departure' and 'actual_departure_time' columns to datetime
darwin_df['planned_time_departure'] = pd.to_datetime(darwin_df['planned_time_departure'])
darwin_df['actual_departure_time'] = pd.to_datetime(darwin_df['actual_departure_time'])

# Calculate absolute errors in seconds
darwin_df['abs_error_departure'] = (darwin_df['actual_departure_time'] - darwin_df['planned_time_departure']).dt.total_seconds()

# Convert absolute errors from seconds to minutes and create a new column
darwin_df['abs_error_departure_minutes'] = darwin_df['abs_error_departure'] / 60

# Define custom bins and Labels
delay_bins = [0.1, 1.5, 4.9, 9.9, float('inf')]
delay_labels = ['0.1-1.5 Minutes', '1.6-4.9 Minutes', '5.0-9.9 Minutes', '10.0 Minutes+']

# Group the data into the specified ranges
darwin_df['delay_category'] = pd.cut(darwin_df['abs_error_departure_minutes'], bins=delay_bins)

# Filter out negative values

```

```

positive_errors_df = darwin_df[darwin_df['abs_error_departure_minutes'] >= 0].copy()

# Group the data into the specified ranges for positive errors
positive_errors_df.loc[:, 'delay_category'] = pd.cut(positive_errors_df['abs_error_
```

# Count occurrences of each delay category

```
delay_counts = positive_errors_df['delay_category'].value_counts()

# Get the absolute error values and their frequencies from the positive errors data
error_counts = delay_counts.reindex(delay_labels, fill_value=0)

# Extract the absolute error values (in minutes) and their corresponding frequencies
sizes = error_counts.values
labels = error_counts.index

# Define colors for the pie chart
colors = sns.color_palette('pastel')[len(sizes):]

# Create a pie chart
plt.figure(figsize=(10, 6))
plt.pie(sizes, labels=labels, autopct='%.1f%%', startangle=140, colors=colors, wedgeprops={'width': 0.3})
plt.title('Distribution of delays, based on minutes')
plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()

```

