visualitzacio_exploratoria

August 16, 2021

1 S04 T01: Visualització gràfica d'un dataset

1.1 Descripció:

Complementa les tècniques d'exploració de les dades mitjantçant la visualització gràfica, amb les llibreries Matplotlib i Searborn.

1.1.1 Nivell 1

- Exercici 1 Resumeix gràficament el data set DelayedFlights.csv

Crea almenys una visualització per:

- Una variable categòrica (UniqueCarrier)
- Una variable numèrica (ArrDelay)
- Una variable numèrica i una categòrica (ArrDelay i UniqueCarrier)
- Dues variables numèriques (ArrDelay i DepDelay)
- Tres variables (ArrDelay, DepDelay i UniqueCarrier)
- Més de tres variables (ArrDelay, DepDelay, AirTime i UniqueCarrier).

[1]:
```
# Render our plots inline
%matplotlib inline
# Import matplotlib
import matplotlib.pyplot as plt
# Import numpy
import numpy as np
# Import pandas
import pandas as pd
# Import seaborn
import seaborn as sns
sns.set_theme(style="ticks", color_codes=True)
# Import plotnine
from plotnine import ggplot, aes, stat_bin, geom_bar
plt.style.use('ggplot')
#plt.style.use('default')
from matplotlib import rcParams
```

[12]:
```
# Reading data from CSV: with CSV files a single line is needed to load the␣
 ↪data:
```

```
#DelayedFlights_df = pd.read_csv('..\estructures_dataframe\DelayedFlights.csv',␣
↪index_col=0, low_memory=False)
DelayedFlights_df = pd.read_csv('..\estructures_dataframe\DelayedFlights.csv',␣
↪index_col=0) #, index_col=0, low_memory=False
#DelayedFlights_df.set_index(['Year'], inplace = True, append = False, drop =␣
↪True)
# resetting index
#DelayedFlights_df.reset_index(inplace = True)
#print('Dataframe dimensions:', DelayedFlights_df.shape)
DelayedFlights_df
```

```
C:\Users\anaconda3\envs\myenv\lib\site-packages\numpy\lib\arraysetops.py:583:
FutureWarning: elementwise comparison failed; returning scalar instead, but in
the future will perform elementwise comparison
```

[12]:

	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	\
0	2008	1	3	4	2003.0	1955	2211.0	
1	2008	1	3	4	754.0	735	1002.0	
2	2008	1	3	4	628.0	620	804.0	
4	2008	1	3	4	1829.0	1755	1959.0	
5	2008	1	3	4	1940.0	1915	2121.0	
…	…	…	…	…	…	…	…	…
7009710	2008	12	13	6	1250.0	1220	1617.0	
7009717	2008	12	13	6	657.0	600	904.0	
7009718	2008	12	13	6	1007.0	847	1149.0	
7009726	2008	12	13	6	1251.0	1240	1446.0	
7009727	2008	12	13	6	1110.0	1103	1413.0	

	CRSArrTime	UniqueCarrier	FlightNum	…	TaxiIn	TaxiOut	Cancelled	\
0	2225	WN	335	…	4.0	8.0	0	
1	1000	WN	3231	…	5.0	10.0	0	
2	750	WN	448	…	3.0	17.0	0	
4	1925	WN	3920	…	3.0	10.0	0	
5	2110	WN	378	…	4.0	10.0	0	
…	…	…	…	…	…	…	…	…
7009710	1552	DL	1621	…	9.0	18.0	0	
7009717	749	DL	1631	…	15.0	34.0	0	
7009718	1010	DL	1631	…	8.0	32.0	0	
7009726	1437	DL	1639	…	13.0	13.0	0	
7009727	1418	DL	1641	…	8.0	11.0	0	

	CancellationCode	Diverted	CarrierDelay	WeatherDelay	NASDelay	\
0	N	0	NaN	NaN	NaN	
1	N	0	NaN	NaN	NaN	
2	N	0	NaN	NaN	NaN	
4	N	0	2.0	0.0	0.0	
5	N	0	NaN	NaN	NaN	

```
…                    …       …            …           …       …
7009710                  N         0          3.0         0.0      0.0
7009717                  N         0          0.0        57.0     18.0
7009718                  N         0          1.0         0.0     19.0
7009726                  N         0          NaN         NaN      NaN
7009727                  N         0          NaN         NaN      NaN

         SecurityDelay  LateAircraftDelay
0                  NaN                NaN
1                  NaN                NaN
2                  NaN                NaN
4                  0.0               32.0
5                  NaN                NaN
…                  …                  …
7009710            0.0               22.0
7009717            0.0                0.0
7009718            0.0               79.0
7009726            NaN                NaN
7009727            NaN                NaN

[1936758 rows x 29 columns]
```

[13]:
```
# Duplicate elimination
DelayedFlights_df.drop_duplicates(inplace=True)
# removing null values to avoid errors
DelayedFlights_df.dropna(inplace = True)
# Selecting only flights from January 2008:
DelayedFlights_df = DelayedFlights_df[DelayedFlights_df['Month'] == 1]
DelayedFlights_df.head()
```

[13]:
```
    Year  Month  DayofMonth  DayOfWeek  DepTime  CRSDepTime  ArrTime  \
4   2008      1           3          4   1829.0        1755   1959.0
6   2008      1           3          4   1937.0        1830   2037.0
11  2008      1           3          4   1644.0        1510   1845.0
16  2008      1           3          4   1452.0        1425   1640.0
18  2008      1           3          4   1323.0        1255   1526.0

    CRSArrTime UniqueCarrier  FlightNum  … TaxiIn  TaxiOut  Cancelled  \
4         1925            WN       3920  …    3.0     10.0          0
6         1940            WN        509  …    3.0      7.0          0
11        1725            WN       1333  …    6.0      8.0          0
16        1625            WN        675  …    7.0      8.0          0
18        1510            WN          4  …    4.0      9.0          0

    CancellationCode  Diverted  CarrierDelay WeatherDelay NASDelay  \
4                  N         0           2.0          0.0      0.0
6                  N         0          10.0          0.0      0.0
```

```
11                N         0           8.0          0.0       0.0
16                N         0           3.0          0.0       0.0
18                N         0           0.0          0.0       0.0

    SecurityDelay  LateAircraftDelay
4             0.0               32.0
6             0.0               47.0
11            0.0               72.0
16            0.0               12.0
18            0.0               16.0

[5 rows x 29 columns]
```

[18]:
```
DelayedFlights_df.columns
```

[18]:
```
Index(['Year', 'Month', 'DayofMonth', 'DayOfWeek', 'DepTime', 'CRSDepTime',
       'ArrTime', 'CRSArrTime', 'UniqueCarrier', 'FlightNum', 'TailNum',
       'ActualElapsedTime', 'CRSElapsedTime', 'AirTime', 'ArrDelay',
       'DepDelay', 'Origin', 'Dest', 'Distance', 'TaxiIn', 'TaxiOut',
       'Cancelled', 'CancellationCode', 'Diverted', 'CarrierDelay',
       'WeatherDelay', 'NASDelay', 'SecurityDelay', 'LateAircraftDelay'],
      dtype='object')
```

[14]:
```
subset_df =␣
↪DelayedFlights_df[['UniqueCarrier','Month','ArrDelay','DepDelay','AirTime']]
#subset_df.set_index(['UniqueCarrier'], inplace = True, append = True, drop =␣
↪True)
# resetting index
#subset_df.reset_index(inplace = True)
# let's see the DataFrame
subset_df#.head()
```

[14]:
```
       UniqueCarrier  Month  ArrDelay  DepDelay  AirTime
4                 WN      1      34.0      34.0     77.0
6                 WN      1      57.0      67.0    230.0
11                WN      1      80.0      94.0    107.0
16                WN      1      15.0      27.0    213.0
18                WN      1      16.0      28.0    110.0
…                 …     …        …         …        …
605627            DL      1      34.0      30.0    258.0
605641            DL      1     185.0     200.0    239.0
605657            DL      1      19.0      11.0    113.0
605699            DL      1      23.0       7.0    249.0
605733            DL      1     216.0     207.0     87.0

[120786 rows x 5 columns]
```

[27]:
```
df = subset_df.copy()
df.set_index(['UniqueCarrier'], inplace = True, append = False, drop = True)
# resetting index
df.reset_index(inplace = True)
# Add two columns to make a new column
df.loc[:,'Total_Delay'] = df.loc[:,'ArrDelay'] + df.loc[:,'DepDelay']
print('Updated DataFrame:')
print(df)
```

```
Updated DataFrame:
       UniqueCarrier  Month  ArrDelay  DepDelay  AirTime  Total_Delay
0                 WN      1      34.0      34.0     77.0         68.0
1                 WN      1      57.0      67.0    230.0        124.0
2                 WN      1      80.0      94.0    107.0        174.0
3                 WN      1      15.0      27.0    213.0         42.0
4                 WN      1      16.0      28.0    110.0         44.0
…                 …    …        …         …       …            …
120781            DL      1      34.0      30.0    258.0         64.0
120782            DL      1     185.0     200.0    239.0        385.0
120783            DL      1      19.0      11.0    113.0         30.0
120784            DL      1      23.0       7.0    249.0         30.0
120785            DL      1     216.0     207.0     87.0        423.0

[120786 rows x 6 columns]
```

[87]:
```
df = subset_df.copy()
# Add two columns to make a new column
df.loc[:,'Total_Delay'] = df.loc[:,['ArrDelay','DepDelay']].sum(axis=1)
print('Updated DataFrame:')
print(df)
```

```
Updated DataFrame:
       UniqueCarrier  Month  ArrDelay  DepDelay  AirTime  Total_Delay
4                 WN      1      34.0      34.0     77.0         68.0
6                 WN      1      57.0      67.0    230.0        124.0
11                WN      1      80.0      94.0    107.0        174.0
16                WN      1      15.0      27.0    213.0         42.0
18                WN      1      16.0      28.0    110.0         44.0
…                 …    …        …         …       …            …
605627            DL      1      34.0      30.0    258.0         64.0
605641            DL      1     185.0     200.0    239.0        385.0
605657            DL      1      19.0      11.0    113.0         30.0
605699            DL      1      23.0       7.0    249.0         30.0
605733            DL      1     216.0     207.0     87.0        423.0

[120786 rows x 6 columns]
```

[95]:
```
df['UniqueCarrier'].describe()
```

[95]:
```
count     120786
unique        20
top           WN
freq       19951
Name: UniqueCarrier, dtype: object
```

[97]:
```
df.sort_values(by=['UniqueCarrier']).groupby(['UniqueCarrier'])
df
```

[97]:

	UniqueCarrier	Month	ArrDelay	DepDelay	AirTime	Total_Delay	Count
4	WN	1	34.0	34.0	77.0	68.0	120786
6	WN	1	57.0	67.0	230.0	124.0	120786
11	WN	1	80.0	94.0	107.0	174.0	120786
16	WN	1	15.0	27.0	213.0	42.0	120786
18	WN	1	16.0	28.0	110.0	44.0	120786
…	…	…	…	…	…	…	…
605627	DL	1	34.0	30.0	258.0	64.0	120786
605641	DL	1	185.0	200.0	239.0	385.0	120786
605657	DL	1	19.0	11.0	113.0	30.0	120786
605699	DL	1	23.0	7.0	249.0	30.0	120786
605733	DL	1	216.0	207.0	87.0	423.0	120786

[120786 rows x 7 columns]

[92]:
```
df.loc[:,'Count'] = df.loc[:,'UniqueCarrier'].count()
df
```

[92]:

	UniqueCarrier	Month	ArrDelay	DepDelay	AirTime	Total_Delay	Count
4	WN	1	34.0	34.0	77.0	68.0	120786
6	WN	1	57.0	67.0	230.0	124.0	120786
11	WN	1	80.0	94.0	107.0	174.0	120786
16	WN	1	15.0	27.0	213.0	42.0	120786
18	WN	1	16.0	28.0	110.0	44.0	120786
…	…	…	…	…	…	…	…
605627	DL	1	34.0	30.0	258.0	64.0	120786
605641	DL	1	185.0	200.0	239.0	385.0	120786
605657	DL	1	19.0	11.0	113.0	30.0	120786
605699	DL	1	23.0	7.0	249.0	30.0	120786
605733	DL	1	216.0	207.0	87.0	423.0	120786

[120786 rows x 7 columns]

1.2 Distribution of total delay

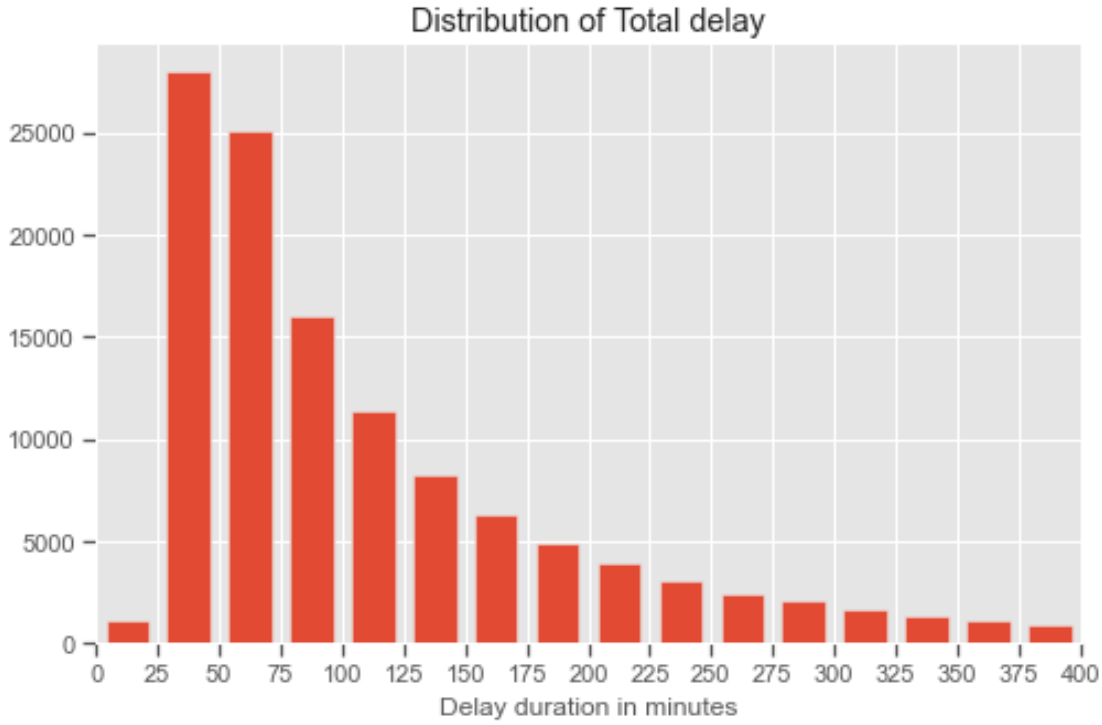
The plot shows that majority of the delays are within 25 minutes,and the number decreases significantly as the duration increases.

This shows that major delays are few and if we reduce the minor delays air transport will be optimised and those delays with high values are actually outliers.

[58]:
```
bins=np.arange(0,430,25) #
bins.dtype
bins = bins.astype(int)
bins
```

[58]:
```
array([  0,  25,  50,  75, 100, 125, 150, 175, 200, 225, 250, 275, 300,
       325, 350, 375, 400, 425])
```

[60]:
```
plt.figure(figsize=[8,5])
bins=np.arange(0,df['Total_Delay'].max()+10,25)
plt.hist(data=df,x='Total_Delay',bins=bins,rwidth=0.7)
plt.title('Distribution of Total delay')
plt.xlabel('Delay duration in minutes')
plt.xlim(0,400)
plt.xticks([0,  25,  50,  75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325,␣
 ↪350, 375, 400]);
```



1.3 Distribution of unique carrier with total delay

From the barplot it clear that YV plane carrier has major contribution the delays followed by OO and UA.

[86]:
```
```

```
#df['Counts'] = subset_df.groupby(['UniqueCarrier']).count().reset_index()
#df['Counts'] = subset_df.groupby(by="UniqueCarrier")["UniqueCarrier"].
 ↪value_counts().reset_index()#name="count"
#df['Counts'] = subset_df.groupby(subset_df["UniqueCarrier"]).count().
 ↪reset_index('Counts')
#df['Counts'] = subset_df.groupby('UniqueCarrier',␣
 ↪as_index=False)['UniqueCarrier'].value_counts()
#df = df.groupby(['UniqueCarrier']).count()
type(df)
#df.sort_values(by=['UniqueCarrier'])
#df
```

[86]: pandas.core.groupby.generic.DataFrameGroupBy

[77]:
```
#subset_df['Count'] = subset_df['UniqueCarrier'].value_counts()#.
 ↪reset_index(name="count")
#subset_df['UniqueCarrier'].value_counts().transform("count")
#subset_df = subset_df.groupby('UniqueCarrier')['UniqueCarrier']#.
 ↪transform('count').copy()
#uc_df = subset_df.groupby('UniqueCarrier')#
#df = subset_df.
 ↪groupby([['UniqueCarrier','Month','ArrDelay','DepDelay','AirTime']])#['UniqueCarrier'].
 ↪count()
#uc_df.groups
#for uc_df, group in uc_df:
#    print(uc_df)
#    print(group.count())
#subset_df.count()
#subset_df['UniqueCarrier'].value_counts()
#subset_df = subset_df.groupby('domain')['ID'].nunique()
#subset_df = subset_df.groupby(['UniqueCarrier','Month']).size().
 ↪unstack(fill_value=0)
df=subset_df.groupby(['UniqueCarrier']).count().reset_index('UniqueCarrier')
df
#pd_df = pd.DataFrame()
df.count()
#subset_df['UniqueCarrier'].value_counts()
# sort df by Total_Delay column
#df['Count'].value_count()
#df = df.sort_values('Total_Delay', ascending=False)#.reset_index(drop=True)
df
```

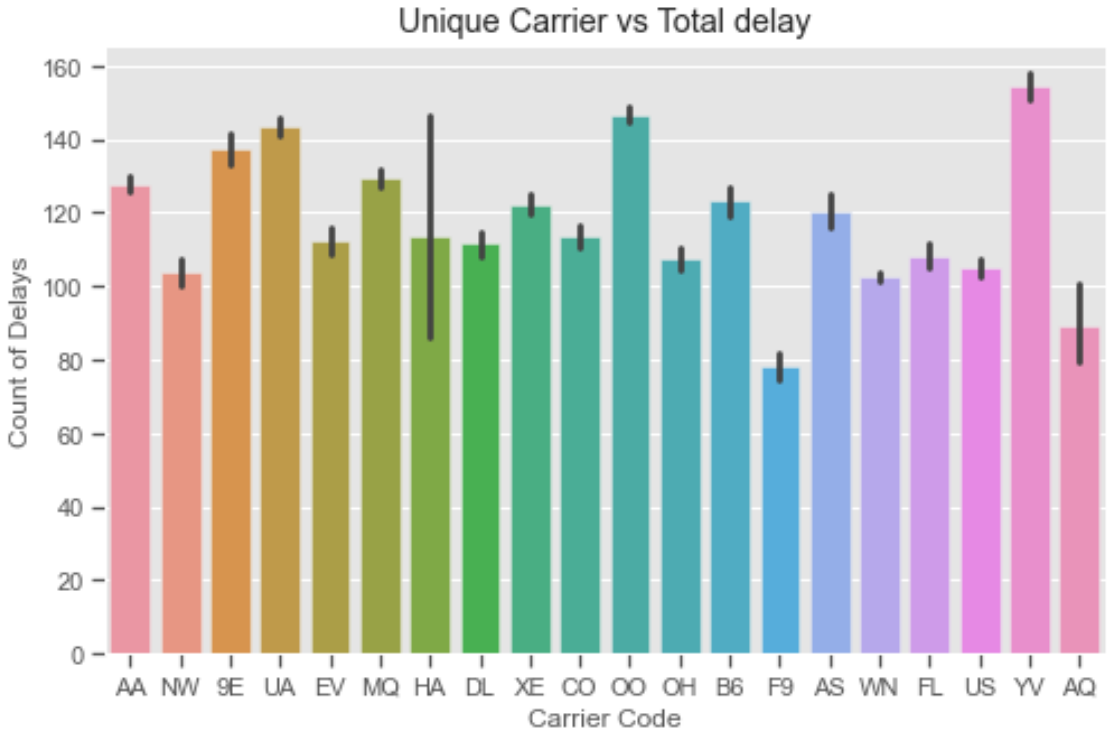
[77]:

	UniqueCarrier	Month	ArrDelay	DepDelay	AirTime	Total_Delay
108413	AA	1	1525.0	1275.0	207.0	2800.0
102257	AA	1	1357.0	1355.0	86.0	2712.0
96707	AA	1	1147.0	1172.0	196.0	2319.0
85904	NW	1	1146.0	1120.0	181.0	2266.0

```
90278              NW       1    1123.0    1120.0      88.0        2243.0
…                 …    …        …         …         …           …
52854              UA       1      15.0       6.0     110.0          21.0
4005               WN       1      15.0       6.0      49.0          21.0
55013              US       1      15.0       6.0      55.0          21.0
41021              OO       1      15.0       6.0      60.0          21.0
52829              UA       1      15.0       6.0     117.0          21.0

[120786 rows x 6 columns]
```

[74]:
```
plt.figure(figsize=[8,5])
sns.barplot(data=df,x='UniqueCarrier',y='Total_Delay')
plt.title('Unique Carrier vs Total delay')
plt.xlabel('Carrier Code')
plt.ylabel('Count of Delays');
```



1.4 Barplot for distribution of unique carrier with total on monthly basis

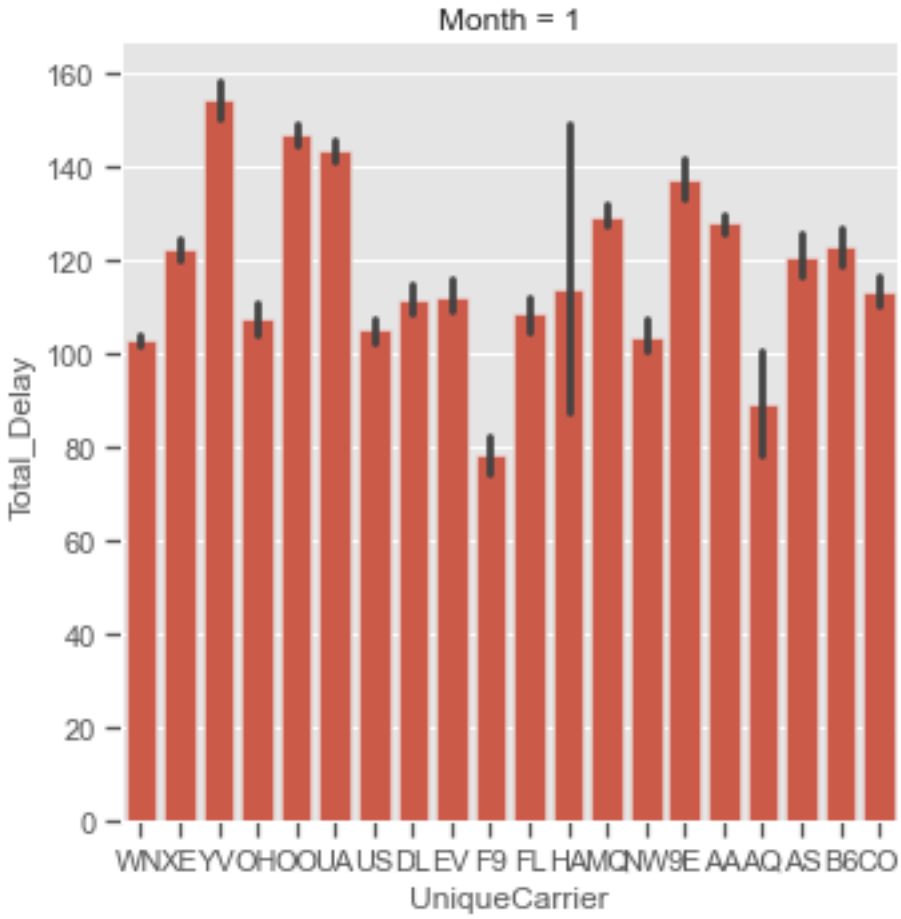
The graph shows that most of the delays in the month of October and November are of flights belonging to PS. Only in the month of December CO carrier has the majority of delays.The result seems quite obvious as seen above that PS flight carrier has the major contribution in the total delay followed by CO carrier.

Also one interesting fact that the out of the three months Month 11 i.e. November has the most

number of delays.

[64]:
```
g = sns.FacetGrid(data=df, col='Month', margin_titles=True, sharex=False,␣
↪sharey=False, height=5)
g.map(sns.barplot, 'UniqueCarrier', 'Total_Delay', order=["Dinner", "Lunch"])
g.add_legend();
```

```
C:\Users\anaconda3\envs\myenv\lib\site-packages\seaborn\axisgrid.py:643:
UserWarning: Using the barplot function without specifying `order` is likely to
produce an incorrect plot.
```



[71]:
```
pd_df = pd.DataFrame()
pd_df["Count"] = subset_df[['UniqueCarrier']].value_counts(sort=True)
#pd_df["Count"] = subset_df[['UniqueCarrier']].value_counts()
# resetting index
#pd_df.reset_index(inplace = True)
#pd_df.set_index(["Count"], inplace = True, append = False, drop = True)
#type(pd_df)
```

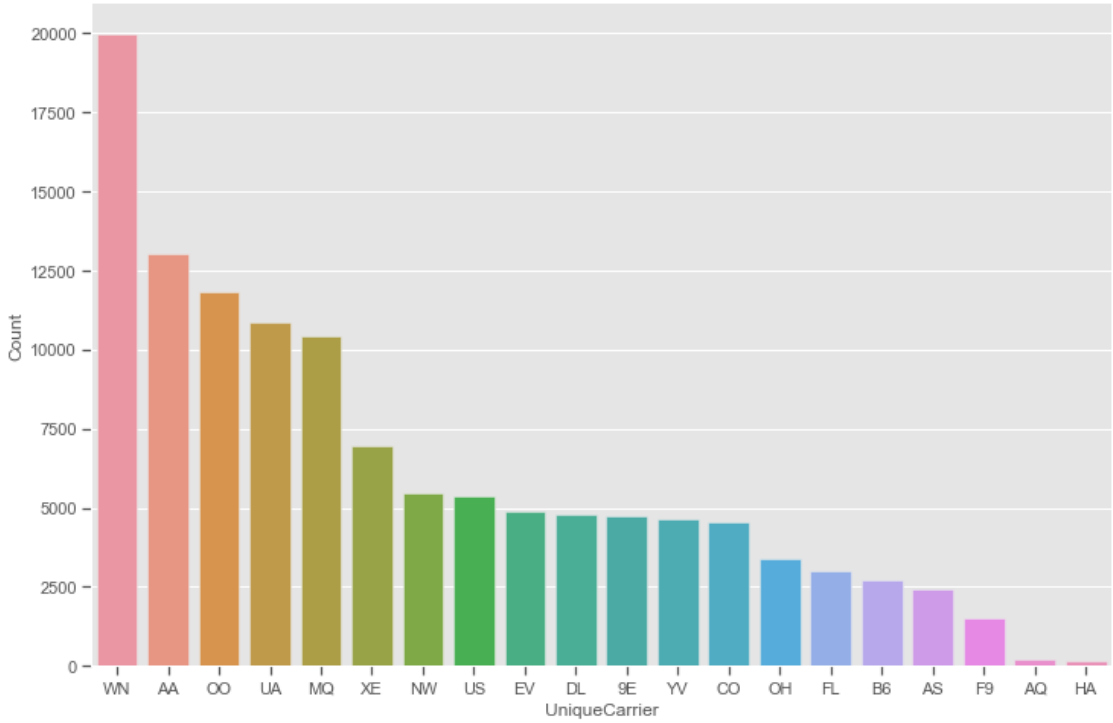
```
# sort df by Count column
#pd_df = pd_df.sort_values('Count', ascending=False)#.reset_index(drop=True)
pd_df #.to_markdown()
```

[71]:

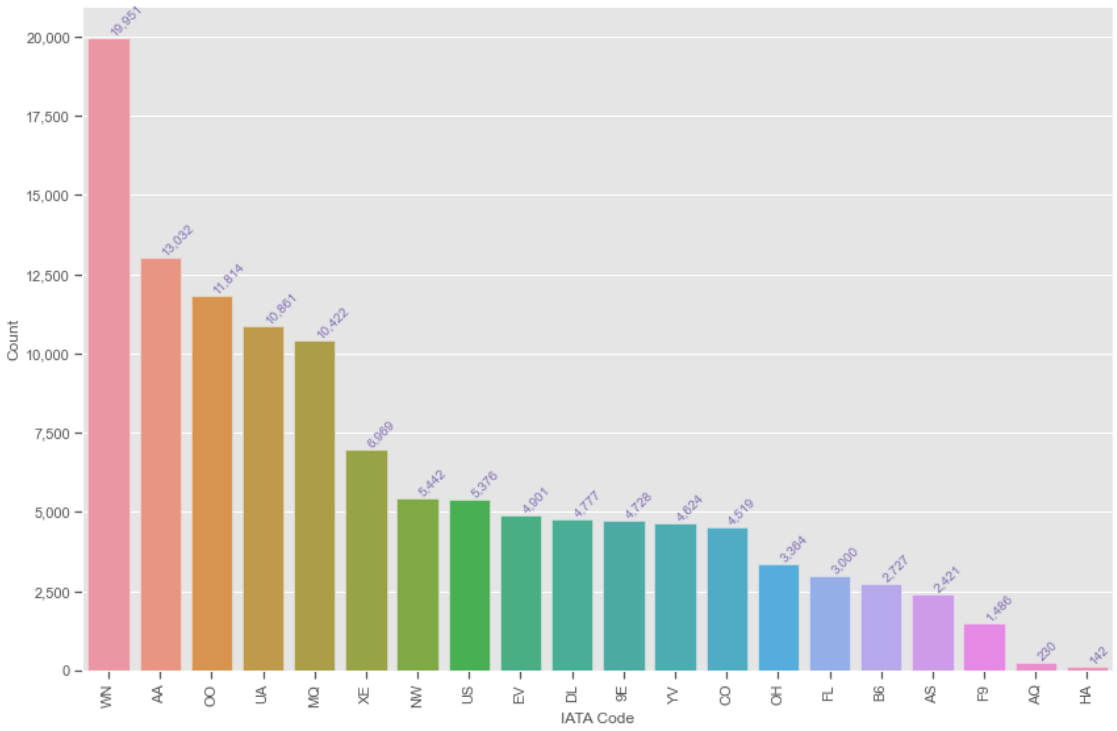
UniqueCarrier	Count
WN	19951
AA	13032
OO	11814
UA	10861
MQ	10422
XE	6969
NW	5442
US	5376
EV	4901
DL	4777
9E	4728
YV	4624
CO	4519
OH	3364
FL	3000
B6	2727
AS	2421
F9	1486
AQ	230
HA	142

[122]:
```
plt.figure(figsize=(12,8))
sns.barplot(data=pd_df, x="UniqueCarrier", y="Count",␣
↪order=pd_df['UniqueCarrier'])
```

[122]:
```
<AxesSubplot:xlabel='UniqueCarrier', ylabel='Count'>
```



[111]:
```
plt.figure(figsize=(12,8))
# plot barh chart with index as x values
ax = sns.barplot(data=pd_df, x="UniqueCarrier", y="Count")
ax.get_yaxis().set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:,}".
 ↪format(int(x))))
ax.set(xlabel="IATA Code", ylabel='Count')
# add proper Dim values as x labels
ax.set_xticklabels(pd_df.UniqueCarrier)
for item in ax.get_xticklabels(): item.set_rotation(90)
for i, v in enumerate(pd_df["Count"].iteritems()):
    ax.text(i ,v[1], "{:,}".format(v[1]), color='m', va ='bottom', rotation=45)
plt.tight_layout()
plt.show()
```



[ ]:
```
sns.displot(data=pd_df, x="UniqueCarrier", y="Count", kind="hist", height=8,␣
↪aspect=2)#binwidth=5, shrink=.8, discrete=True
```

1.5 UNIQUE CARRIER CODE

[20]:
```
subset_df['UniqueCarrier'].describe()
```

[20]:
```
count     1936758
unique         20
top            WN
freq       377602
Name: UniqueCarrier, dtype: object
```

[12]:
```
subset_df['UniqueCarrier'].value_counts().head(20)
```

[12]:
```
WN    377602
AA    191865
MQ    141920
UA    141426
OO    132433
DL    114238
XE    103663
CO    100195
```

```
US     98425
EV     81877
NW     79108
FL     71284
YV     67063
B6     55315
OH     52657
9E     51885
AS     39293
F9     28269
HA      7490
AQ       750
Name: UniqueCarrier, dtype: int64
```

[13]:
```
subset_df['Month']
```

[13]:
```
0           1
1           1
2           1
4           1
5           1
           ..
7009710    12
7009717    12
7009718    12
7009726    12
7009727    12
Name: Month, Length: 1936758, dtype: int64
```

1.5.1 Convert Month int (value variable) to Month name (categorical variable)

Since the abbreviated month names is the first three letters of their full names, we could first convert the Month column to datetime and then use dt.month_name() to get the full month name and finally use str.slice() method to get the first three letters, all using pandas and only in one line of code:

[26]:
```
#look_up = {'01': 'Jan', '02': 'Feb', '03': 'Mar', '04': 'Apr', '05': 'May',␣
↪'06': 'Jun', '07': 'Jul', '08': 'Aug', '09': 'Sep', '10': 'Oct', '11':␣
↪'Nov', '12': 'Dec'}
#subset_df['Month_name'] = subset_df['Month'].apply(lambda x: look_up[x])
subset_df['Month_name'] = pd.to_datetime(subset_df['Month'], format='%m').dt.
↪month_name().str.slice(stop=3)
subset_df
```

```
<ipython-input-26-5fe4c7c1a278>:3: 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

[26]:

	UniqueCarrier	Month	ArrDelay	DepDelay	AirTime	Month_name
0	WN	1	-14.0	8.0	116.0	Jan
1	WN	1	2.0	19.0	113.0	Jan
2	WN	1	14.0	8.0	76.0	Jan
4	WN	1	34.0	34.0	77.0	Jan
5	WN	1	11.0	25.0	87.0	Jan
…	…	…	…	…	…	
7009710	DL	12	25.0	30.0	120.0	Dec
7009717	DL	12	75.0	57.0	78.0	Dec
7009718	DL	12	99.0	80.0	122.0	Dec
7009726	DL	12	9.0	11.0	89.0	Dec
7009727	DL	12	-5.0	7.0	104.0	Dec

[1936758 rows x 6 columns]

[95]:
```
# Month in string format
df = pd.DataFrame({'client':['sss', 'yyy', 'www'], 'Month': ['02', '12', '06']})
look_up = {'01': 'Jan', '02': 'Feb', '03': 'Mar', '04': 'Apr', '05': 'May',␣
 ↪'06': 'Jun', '07': 'Jul', '08': 'Aug', '09': 'Sep', '10': 'Oct', '11':␣
 ↪'Nov', '12': 'Dec'}
#df['Month_name'] = df['Month'].apply(lambda x: look_up[x])
df['Month_name'] = pd.to_datetime(df['Month'], format='%m').dt.month_name().str.
 ↪slice(stop=3)
df
```

[95]:

	client	Month	Month_name
0	sss	02	Feb
1	yyy	12	Dec
2	www	06	Jun

[57]:
```
#rng = pd.date_range(pd.Timestamp("2018-03-10 09:00"), periods=3, freq='s')
rng = pd.Timestamp("2018-03-10 09:00")
#type(rng)
#rng.strftime('%B %d, %Y, %r')
#rng.strftime('%b')
rng.month_name()
```

[57]: 'March'

[71]:
```
import datetime

#x = datetime.datetime(2020, 5, 17)
x = datetime.datetime.now()

print(x.year)
```

```
print(x.strftime("%A"))

print(type(x))
```

```
2021
Monday
<class 'datetime.datetime'>
```

[70]:
```
# Month in string format
df = pd.DataFrame({'client':['sss', 'yyy', 'www'], 'Month': ['02', '12', '06']})
def mapper(month):
   return month.strftime('%b')
df['Month_name'] = pd.to_datetime(df['Month'], format='%m').apply(mapper)
df
```

[70]:

	client	Month	Month_name
0	sss	02	Feb
1	yyy	12	Dec
2	www	06	Jun

[69]:
```
# Month in integer format
df = pd.DataFrame({'client':['sss', 'yyy', 'www'], 'Month': [2, 12, 6]})
def mapper(month):
    date = datetime.datetime(2000, month, 1)  # You need a dateobject with the␣
 ↪proper month
    return date.strftime('%b')  # %b returns the months abbreviation, other␣
 ↪options [here][1]

df['Month_name'] = df['Month'].apply(mapper)
df
```

[69]:

	client	Month	Month_name
0	sss	2	Feb
1	yyy	12	Dec
2	www	6	Jun

[76]:
```
for i in range(1, 13):
    month_name = calendar.month_abbr[i]
    print(month_name)
```

```
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
```

```
Sep
Oct
Nov
Dec
```

[86]:
```
s = pd.Series(['fox', 'cow', np.nan, 'dog'])
s
```

[86]:
```
0    fox
1    cow
2    NaN
3    dog
dtype: object
```

[101]:
```
s.map({'fox': 'cub', 'cow': 'calf'})
```

[101]:
```
0     cub
1    calf
2     NaN
3     NaN
dtype: object
```

[97]:
```
# It also accepts a function:
s.map('I am a {}'.format)
```

[97]:
```
0    I am a fox
1    I am a cow
2    I am a nan
3    I am a dog
dtype: object
```

[98]:
```
s.map('I am a {}'.format, na_action='ignore')
```

[98]:
```
0    I am a fox
1    I am a cow
2           NaN
3    I am a dog
dtype: object
```

[102]:
```
# Month in integer format
import calendar
df = pd.DataFrame({'client':['sss', 'yyy', 'www'], 'Month': [2, 12, 6]})
df['Month_name'] = df['Month'].apply(lambda x: calendar.month_abbr[x])
#d = dict(enumerate(calendar.month_abbr))
#df['Month_name'] = df['Month'].map(d)
df
```

[102]:

	client	Month	Month_name
0	sss	2	Feb
1	yyy	12	Dec
2	www	6	Jun

[111]:
```
import time

named_tuple = time.localtime() # get struct_time
time_string = time.strftime("%m/%d/%Y, %H:%M:%S", named_tuple)

print(time_string)
```

```
06/28/2021, 21:32:44
```

[115]:
```
import time

time_string = "21 June, 2018"
result = time.strptime(time_string, "%d %B, %Y").tm_mon
month = calendar.month_abbr[result]
print(month)
print(result)
```

```
Jun
6
```

[121]:
```
from time import strptime
# Month in string format
df = pd.DataFrame({'client':['sss', 'yyy', 'www'], 'Month': ['02', '12', '06']})
#df['Month_name'] = df['Month'].apply(lambda x: strptime(x,'%m').tm_mon)
res = df['Month'].apply(lambda x: strptime(x,'%m').tm_mon)
df['Month_name'] = res.map(d)
df
```

[121]:

	client	Month	Month_name
0	sss	02	Feb
1	yyy	12	Dec
2	www	06	Jun

[239]:
```
index = ['Firefox', 'Chrome', 'Safari', 'IE10', 'Konqueror']
df = pd.DataFrame({'http_status': [200, 200, 404, 404, 301],
                   'response_time': [0.04, 0.02, 0.07, 0.08, 1.0]},
                   index=index)
df
```

[239]:

	http_status	response_time
Firefox	200	0.04
Chrome	200	0.02
Safari	404	0.07

```
IE10                404           0.08
Konqueror           301           1.00
```

[240]:
```
new_index = ['Safari', 'Iceweasel', 'Comodo Dragon', 'IE10',
             'Chrome']
#df.reindex(new_index)
df.reindex(columns=['response_time', 'http_status'])
```

[240]:
```
           response_time  http_status
Firefox             0.04          200
Chrome              0.02          200
Safari              0.07          404
IE10                0.08          404
Konqueror           1.00          301
```

[12]:
```
from numpy import random
index = [1, 2, 3, 4]
index2 = [4, 3, 2, 1, 0]
index3 = ['a','b','c','d']
index4 = ['b','a','d','c']
date2 = pd.date_range('2017-01-06', periods=4, freq='M')
value2 = np.arange(10000, 50000, 10000)
value3 = random.randint(100000, size=(4))
data  =  [['2017-01-06', 37353],
          ['2019-01-06', 94108],
          ['2019-01-05', np.nan],
          ['2019-01-04', 94514]] # Matrix
d = {'date': ['2017-01-06', '2019-01-06','2019-01-05','2019-01-04'],
     'value':[37353, 94108, np.nan, 94514]} # python dictionary
#df = pd.DataFrame(data=d, index=index)
#df.reset_index()
#df1 = df.reindex(index2, method='bfill', fill_value=0)#, method='bfill',␣
 ↪fill_value='missing', fill_value=0
#df1 = df.reindex(columns=['value', 'date'])
#df.reindex(['value', 'date'], axis = "columns")
#df = pd.DataFrame(data=data, columns=['date','value'], index=index3,␣
 ↪dtype=None, copy=False)
df = pd.DataFrame(data=data, columns=['date','value'], index=index, dtype=None,␣
 ↪copy=False)
#df.set_index('date', inplace = True)
#print(df.index)

df['year'] = pd.DatetimeIndex(df['date']).year
df['month_numb'] = pd.DatetimeIndex(df['date']).month
df['month_name'] = pd.DatetimeIndex(df['date']).month_name()

#df.index = pd.to_datetime(df.index)
```

```
#year = df.index.year
#month_num = df.index.month
#month_name = df.index.month_name()
#print(year)
#print(month_num)
#print(month_name)
#print(date2)
#print(value2)
#print(value3)
print(df)
#print(df1)
```

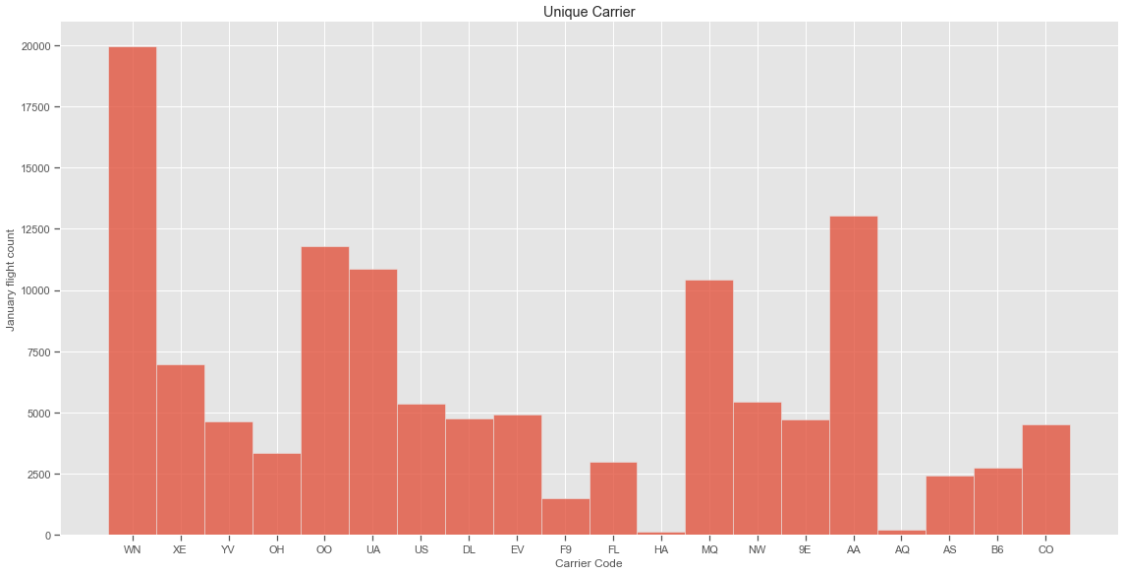
```
         date    value  year  month_numb month_name
1  2017-01-06  37353.0  2017           1    January
2  2019-01-06  94108.0  2019           1    January
3  2019-01-05      NaN  2019           1    January
4  2019-01-04  94514.0  2019           1    January
```

[18]:
```
# The Month_name column is a categorical variable
subset_df['Month_name'].describe(datetime_is_numeric=False)
```

[18]:
```
count     1936758
unique         12
top           Dec
freq       203385
Name: Month_name, dtype: object
```

[10]:
```
# figure size in inches
#rcParams['figure.figsize'] = (14, 14)
#plt.rcParams['figure.figsize'] = (14, 14)
#plt.figure(figsize=[14,14])
#sns.set(rc={'figure.figsize':(14, 14)})
sns.displot(data=subset_df, x="UniqueCarrier", binwidth=3, height=8,␣
 ↪aspect=2)#binwidth=5, shrink=.8, discrete=True
plt.title('Unique Carrier')
plt.xlabel('Carrier Code')
plt.ylabel('January flight count')
```

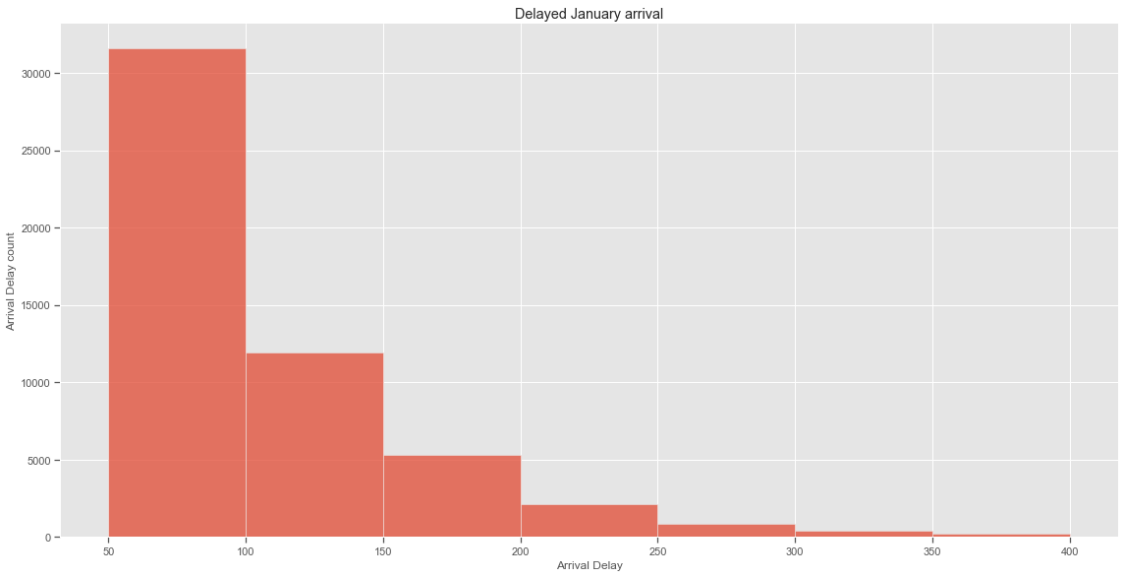
[10]:
```
Text(-8.449999999999996, 0.5, 'January flight count')
```



[9]:
```
plt.figure(figsize=[10,10])
sns.displot(data=subset_df, x="ArrDelay", bins=[50, 100, 150, 200, 250, 300,␣
↪350, 400], height=8, aspect=2)# , binwidth=5, bins=10, bins=[100, 200, 300,␣
↪400, 500, 600, 700], discrete=True, shrink=.8
plt.title('Delayed January arrival')
plt.xlabel('Arrival Delay')
plt.ylabel('Arrival Delay count')
```

[9]:
```
Text(-8.449999999999996, 0.5, 'Arrival Delay count')
```

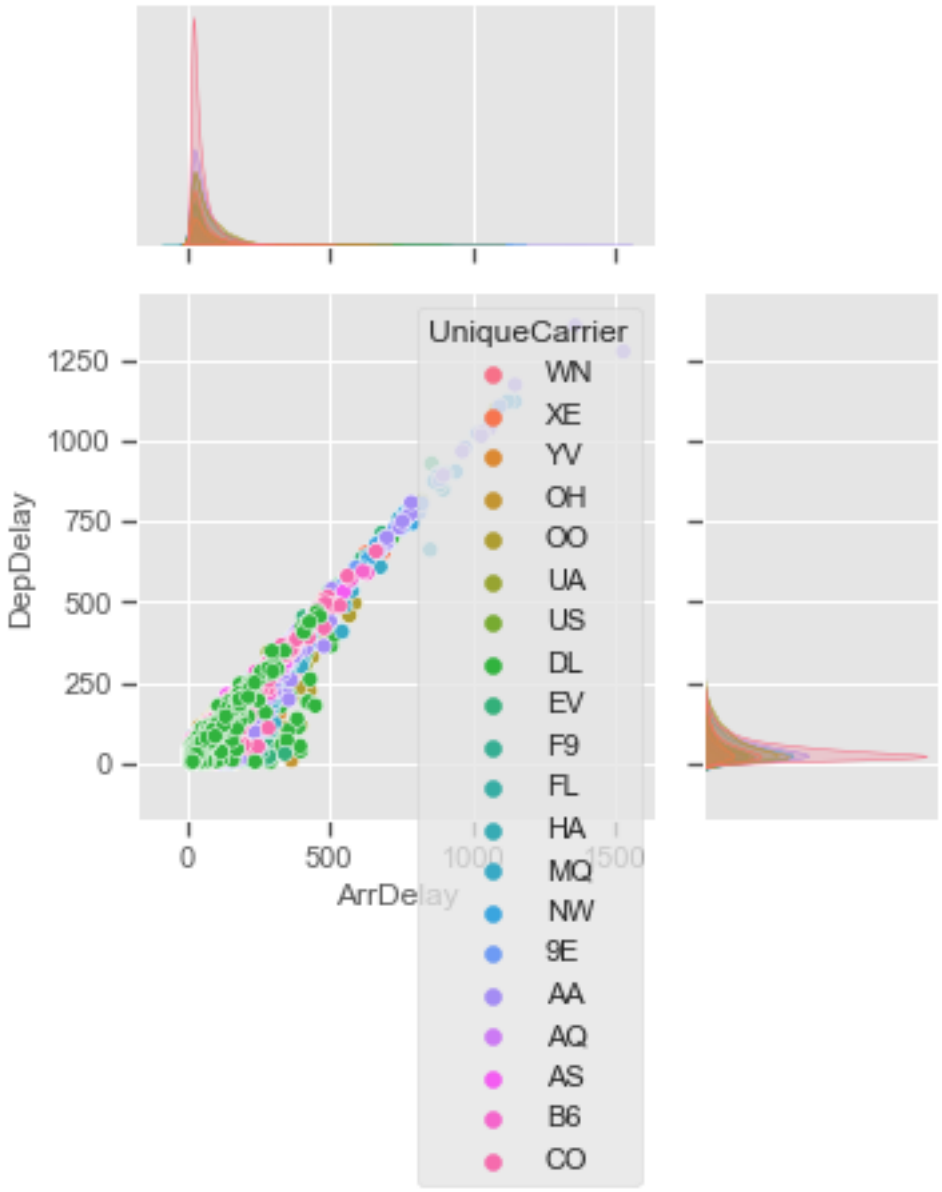
```
<Figure size 720x720 with 0 Axes>
```



[13]:
```
#plt.figure(figsize=[8,5])
#, binwidth=5, bins=10, bins=[100, 200, 300, 400, 500, 600, 700],␣
 ↪discrete=True, shrink=.8
#sns.displot(data=subset_df, x="ArrDelay", y='DepDelay', height=8, aspect=2)
sns.jointplot(data=subset_df, x="ArrDelay", y="DepDelay", hue="UniqueCarrier",␣
 ↪height=10, ratio=2)
#plt.title('Arrival vs Departure Delay')
#plt.xlabel('Arrival Delay')
#plt.ylabel('Departure Delay')
```

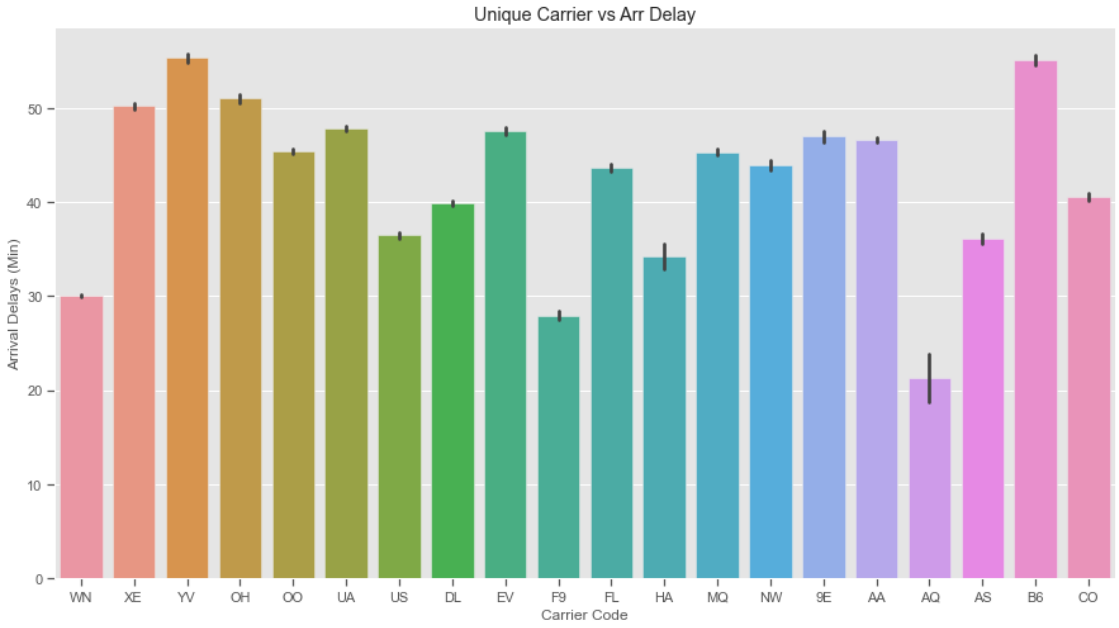
[13]:
```
<seaborn.axisgrid.JointGrid at 0x1c8970c3340>
```

```
C:\Users\Augusto Maidana\AppData\Roaming\Python\Python39\site-
packages\IPython\core\pylabtools.py:132: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
```



[8]:
```
plt.figure(figsize=(15,8))
sns.barplot(data=subset_df,x='UniqueCarrier',y='ArrDelay')
plt.title('Unique Carrier vs Arr Delay')
plt.xlabel('Carrier Code')
plt.ylabel('Arrival Delays (Min)')
```

[8]:
```
Text(0, 0.5, 'Arrival Delays (Min)')
```



- Exercici 2 Exporta els gràfics com imatges o com html.
- Exercici 3 Integra les visualitzacions gràfiques, en la tasca 5, del Sprint 3.