ASSIGNMENT 4 - SPARK (run with pySpark 1G 4e Kernel)

Matthew Dunne

You have two CSV files loaded into /home/kadochnikov/data/air_travel in Linux environment, representing the air travel information for 2007 and 2008. The header column for each file is fairly selfexplanatory.

In [1]:

```
import time
from itertools import islice
import os
import shutil
import sh
import pandas as pd
from pyspark.sql.functions import *
#from pyspark.sql import functions as F (some people like to import this way)
from pyspark.sql.types import *
print (sc.version)
```

```
2.2.0-cdh6.0.0
```

Read the Airlines Data

Read two CSV files from Linux space.

In [3]:

```
csv_file = "file:///project/msca/kadochnikov/data/200*.csv"
airlines_df = spark.read.format('com.databricks.spark.csv').\
options(header='true', inferschema='true', delimiter=',', quote='"').load(csv_file)
```

We will not need certain columns (Carrier, Flight Number, Tail Number, Actual Elapsed Time, CRS Elapsed Time, Air Time, Distance, Taxi In, Taxi Out, Cancelled, Cancellation Code, Diverted, and the amounts for the type of delay. We will drop those.

In [4]:

```
airlines_df = airlines_df.drop('UniqueCarr', 'FlightNum', 'TailNum', 'ActualElapsedTime', 'CRSElaps
edTime', 'AirTime', 'Distance', 'TaxiIn', 'TaxiOut', "Cancelled", 'CancellationCode', 'Diverted', '
CarrierDelay', 'WeatherDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay', 'LateAircraftDelay')
#airlines_df.cache()
airlines_df.show(5)
```

```
+----+-----+----------+---------+-------+----------+-------+----------+-------------+--------+-----
+------+----+
|Year|Month|DayofMonth|DayOfWeek|DepTime|CRSDepTime|ArrTime|CRSArrTime|UniqueCarrier|ArrDelay|DepDe
Origin|Dest|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+--------+-----
+------+----+
|2007|    1|         1|        1|   1232|      1225|   1341|      1340|           WN|       1|
7|   SMF| ONT|
|2007|    1|         1|        1|   1918|      1905|   2043|      2035|           WN|       8|
3|   SMF| PDX|
|2007|    1|         1|        1|   2206|      2130|   2334|      2300|           WN|      34|
6|   SMF| PDX|
|2007|    1|         1|        1|   1230|      1200|   1356|      1330|           WN|      26|
0|   SMF| PDX|
|2007|    1|         1|        1|    831|       830|    957|      1000|           WN|      -3|
1|   SMF| PDX|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+--------+-----
+------+----+
only showing top 5 rows
```

Questions

1. Which locations (Origin and Destination pairs) had the worst delays for both arrivals (ArrDelay) and departures (DepDelay)?

Here we will assume that the "worst delays" means the largest average delay per flight by location. We will also assume that the reverse of an Origin Destination pair constitutes a new location (Chicago to New York is one location and New York to Chicago is another).

In [79]:

```
#create a new column with the Origin, Destination pair as a concatenated string
airlines_df = airlines_df.withColumn('Location', concat(col("Origin"), lit(" "), col("Dest")))
delays=airlines_df.select(['DepDelay', 'ArrDelay', 'Location'])
#cache just the columns you need for the question
delays.cache()
#delays can be negative. Cast them to 0 so they don't affect delays. We only want to count length
of delay when there is one.
#Leaving early with another flight should not make up for a delay.
delays = delays.\
withColumn("ArrDelay",when(col("ArrDelay") <0, 0).otherwise(col("ArrDelay"))).\
withColumn("DepDelay",when(col("DepDelay") <0, 0).otherwise(col("DepDelay")))
arrival_delays = delays.groupby('Location').agg(mean('ArrDelay'),count('*'))
departure_delays = delays.groupby('Location').agg(mean('DepDelay'),count('*'))
departure_delays.describe().show()
```

```
+-------+--------+------------------+-----------------+
|summary|Location|     avg(DepDelay)|         count(1)|
+-------+--------+------------------+-----------------+
|  count|    5849|              5834|             5849|
|   mean|    null|15.251661719627936|2472.720636006155|
| stddev|    null| 21.28078831421851|3202.492945651755|
|    min| ABE ATL|               0.0|                1|
|    max| YUM SLC|             587.0|            28482|
+-------+--------+------------------+-----------------+
```

After grouping by location there are 5849 rows (for both Arrivals and Delays). This can be dealt with easily in Pandas, and so we will send both to Pandas.

In [80]:

```
arrival_delays = arrival_delays.toPandas()
departure_delays = departure_delays.toPandas()
```

In [81]:

```
arrival_delays=arrival_delays.rename(columns={"avg(ArrDelay)": "Average Arrival Delay", "count(1)":
"Number of Flights"})
departure_delays=departure_delays.rename(columns={"avg(DepDelay)": "Average Departure Delay",
"count(1)": "Number of Flights"})
```

In [83]:

```
sorted_arrival_delays=arrival_delays.sort_values(by=['Average Arrival Delay'], ascending=False)
sorted_departure_delays=departure_delays.sort_values(by=['Average Departure Delay'], ascending=Fal
se)
sorted_departure_delays.head(5)
```

Out[83]:

	Location	Average Departure Delay	Number of Flights
495	CMI SPI	587.0	1
1126	ONT IAD	386.0	1
3289	ABQ GJT	366.0	1

	Location	Average Departure Delay	Number of Flights
4366	SDF SPI	329.0	1
4951	ELP MFE	307.0	1

There is one obvious thing about the locations with the worst average delay. They have only one flight. Clearly these should not enter the calculation of what locations have the worst delays. There is some art to this but we shall use a threshold of 52 flights for each location. This is one flight every two weeks (remember we have data from 2007 and 2008).

In [84]:

```
arr_del_df = sorted_arrival_delays[sorted_arrival_delays['Number of Flights'] >=52]
dep_del_df = sorted_departure_delays[sorted_departure_delays['Number of Flights'] >=52]
dep_del_df.head(5)
```

Out[84]:

	Location	Average Departure Delay	Number of Flights
4952	EGE MIA	67.173333	77
3909	LAX EGE	53.059091	223
3070	EWR LIT	49.594976	655
3507	ASE ORD	49.553816	584
4902	HDN ORD	48.673913	340

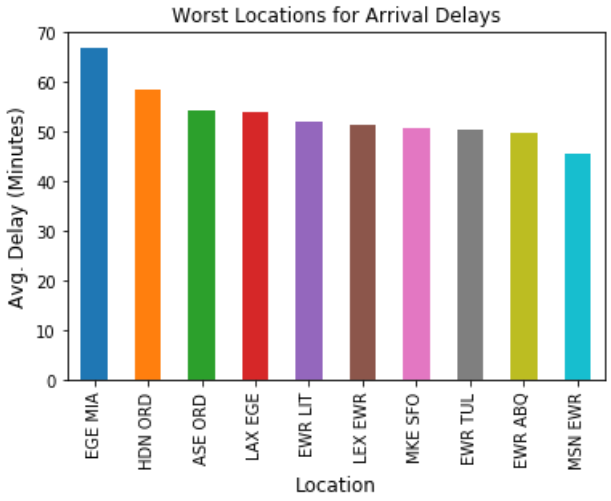
Here is the chart for the Locations with the worst arrival delays (measured by average delay) with a minimum of 52 flights over two years.

In [88]:

```
arr_del_worst=arr_del_df.head(10)
arr_plot=arr_del_worst.plot(kind='bar', legend=False, title="Worst Locations for Arrival Delays", x
='Location', y='Average Arrival Delay')
arr_plot.set_xlabel("Location", fontsize=12)
arr_plot.set_ylabel("Avg. Delay (Minutes)", fontsize=12)
```

Out[88]:

```
Text(0,0.5,'Avg. Delay (Minutes)')
```



Here is the chart for the Locations with the worst departure delays (measured by average delay) with a minimum of 52 flights over two years.

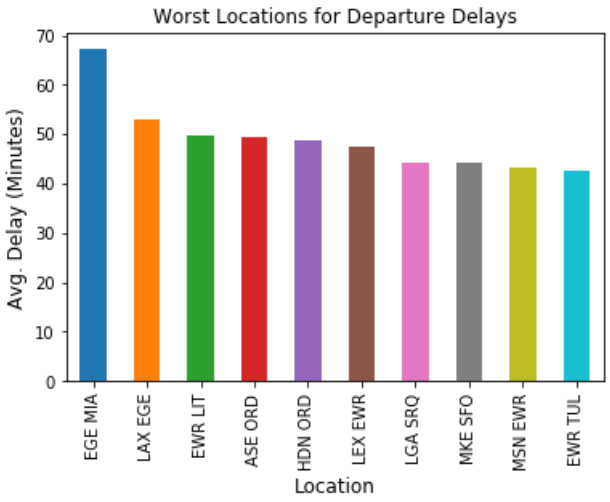
In [89]:

```
dep_del_worst=dep_del_df.head(10)
dep_plot=dep_del_worst.plot(kind='bar', legend=False, title="Worst Locations for Departure Delays"
```

```
, x='Location', y='Average Departure Delay')
dep_plot.set_xlabel("Location", fontsize=12)
dep_plot.set_ylabel("Avg. Delay (Minutes)", fontsize=12)
```

Out[89]:

```
Text(0,0.5,'Avg. Delay (Minutes)')
```



We notice that we see mostly the same locations in both charts with some variation in order.

Now we unpersist what we have cached. It's just good manners.

In [14]:

```
delays.unpersist()
```

Out[14]:

```
DataFrame[DepDelay: string, ArrDelay: string, Location: string]
```

2. Which locations (Origin and Destination pairs) had the fewest delays?

We will count one delay for Arrival Delay and one delay for Departure delay on each flight.

In [90]:

```
airlines_df = airlines_df.\
withColumn("Number of Delays",when(col("ArrDelay") >0, 1).otherwise(0)).\
withColumn("Number of Delays", when(col("DepDelay") >0, col("Number of Delays")+1).otherwise(col("N
umber of Delays")))
number_delays_df=airlines_df.select(['Number of Delays', 'Location'])
number_delays_df.cache()
number_delays_df.show(5)
```

```
+----------------+--------+
|Number of Delays|Location|
+----------------+--------+
|               2| SMF ONT|
|               2| SMF PDX|
|               2| SMF PDX|
|               2| SMF PDX|
|               1| SMF PDX|
+----------------+--------+
only showing top 5 rows
```

To see which locations have the fewest delays, we will group by location.

In [91]:

```
fewest_delays = number_delays_df.groupby('Location').agg(sum('Number of Delays'),count('*'))
fewest_delays = fewest_delays.toPandas()
fewest_delays=fewest_delays.rename(columns={"sum(Number of Delays)": "Number of Delays",
"count(1)": "Number of Flights"})
fewest_delays = fewest_delays.sort_values(by=['Number of Delays'])
fewest_delays.head(10)
```

Out[91]:

	Location	Number of Delays	Number of Flights
5848	PDX PIH	0	1
5611	LGB TWF	0	1
4618	MSO COS	0	1
4617	TTN LGA	0	1
2148	AUS ORF	0	1
409	SGU CDC	0	1
4614	HPN PIT	0	1
3613	SJC SMX	0	3
2187	MTJ PUB	0	1
425	ORD BGM	0	1

Here we have the problem that the locations with the fewest delays are those with the fewest flights. So we shall apply the same threshold of 52 flights and then calculate delays as a percentage of flights.

In [93]:

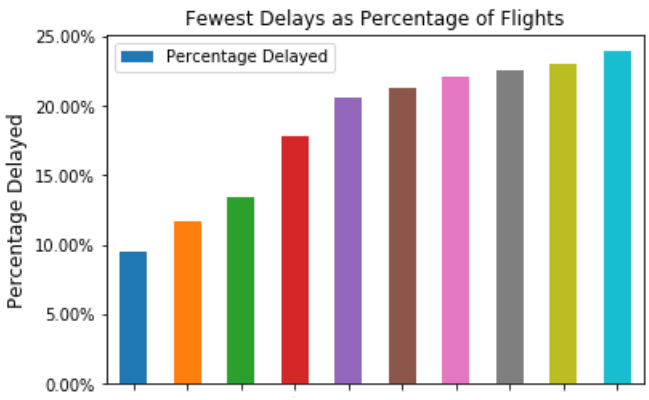
```
fewest_delays = fewest_delays[fewest_delays['Number of Flights'] >=52]
fewest_delays['Percentage Delayed'] = fewest_delays['Number of Delays']/fewest_delays['Number of
Flights']
```

In [94]:

```
fewest_delays=fewest_delays.sort_values(by=['Percentage Delayed']).head(10)
fewest_plot=fewest_delays.plot(kind='bar', title="Fewest Delays as Percentage of Flights",
x='Location', y='Percentage Delayed')
fewest_plot.set_xlabel("Location", fontsize=12)
fewest_plot.set_ylabel("Percentage Delayed", fontsize=12)
vals = fewest_plot.get_yticks()
fewest_plot.set_yticklabels(['{:,.2%}'.format(x) for x in vals])
```

Out[94]:

```
[Text(0,0,'0.00%'),
 Text(0,0,'5.00%'),
 Text(0,0,'10.00%'),
 Text(0,0,'15.00%'),
 Text(0,0,'20.00%'),
 Text(0,0,'25.00%'),
 Text(0,0,'30.00%')]
```



Keep in mind that, given the way we calculated the number of delays, it is possible to have two delays per flight. And one often would - a late arriving plane will tend to be a late departing plane. This inflates the numbers from our normal understanding.

And unpersist what we have cached.

In [95]:

```
number_delays_df.unpersist()
```

Out[95]:

```
DataFrame[Number of Delays: int, Location: string]
```

3. Do you see any significant seasonality effects for delays?

We will take the number of delays by flight as calculated above and group by month.

In [102]:

```
monthly_delays = airlines_df.groupby('Month').agg(sum('Number of Delays'),count('*'))
monthly_delays.cache()
monthly_delays_df = monthly_delays.toPandas()
monthly_delays_df = monthly_delays_df.sort_values(by=['Month'])
monthly_delays_df = monthly_delays_df.rename(columns={'sum(Number of Delays)': 'Number of Delays',
'count(1)': 'Number of Flights'})
monthly_delays_df['% of Flights Delayed'] = monthly_delays_df['Number of
Delays']/monthly_delays_df['Number of Flights']
monthly_delays_df
```

Out[102]:

	Month	Number of Delays	Number of Flights	% of Flights Delayed
1	1	1069486	1227324	0.871397
11	2	1075107	1134840	0.947364
3	3	1136146	1255299	0.905080
6	4	999158	1212774	0.823862
4	5	992318	1237902	0.801613
2	6	1201343	1237945	0.970433
8	7	1152685	1276491	0.903011
7	8	1086813	1265558	0.858762
5	9	738386	1141095	0.647085
9	10	847340	1186197	0.714333
10	11	799063	1128421	0.708125
0	12	1180902	1159097	1.018812

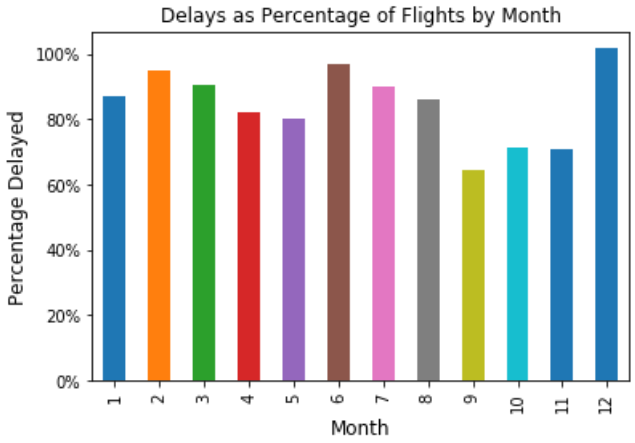
Put in a chart the number of delays as a percentage of flights looks as follows.

In [104]:

```
monthly_delays_df_plot=monthly_delays_df.plot(kind='bar', legend = False, title="Delays as
Percentage of Flights by Month", x='Month', y='% of Flights Delayed')
monthly_delays_df_plot.set_xlabel("Month", fontsize=12)
monthly_delays_df_plot.set_ylabel("Percentage Delayed", fontsize=12)
values = monthly_delays_df_plot.get_yticks()
monthly_delays_df_plot.set_yticklabels(['{:.0%}'.format(x) for x in values])
```

Out[104]:

```
[Text(0,0,'0%'),
 Text(0,0,'20%'),
 Text(0,0,'40%'),
 Text(0,0,'60%'),
 Text(0,0,'80%'),
 Text(0,0,'100%'),
 Text(0,0,'120%')]
```



We might conclude a few things: (1) when the weather is bad (Jan., Feb., Dec.) delays go up (2) during the busy travel season (June through August) delays go up (3) when the weather is moderate and it is not travel season (spring and fall) delays go down. Again we have the same problem of two delays per flight in many cases.

And we unpersist.

In [105]:

```
monthly_delays.unpersist()
```

Out[105]:

```
DataFrame[Month: int, sum(Number of Delays): bigint, count(1): bigint]
```

4. Do you see any increase or decrease in delays on weekends?

Looking at a calendar from 2007, we see that the first of January was a Monday. So we will maps days of the week: 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday, 7=Sunday.

In [106]:

```
daily_delays = airlines_df.groupby('DayofWeek').agg(sum('Number of Delays'),count('*'))
#because we have grouped by day of the week we know there will only be seven rows. Can fit in pand
as without caching
daily_delays_df = daily_delays.toPandas()
daily_delays_df = daily_delays_df.sort_values(by=['DayofWeek'])
daily_delays_df = daily_delays_df.rename(columns={'sum(Number of Delays)': 'Number of Delays',
'count(1)': 'Number of Flights'})
daily_delays_df['% of Flights Delayed'] = daily_delays_df['Number of
Delays']/daily_delays_df['Number of Flights']
daily_delays_df
```

Out[106]:

	DayofWeek	Number of Delays	Number of Flights	% of Flights Delayed
0	1	1855141	2148675	0.863388
6	2	1660190	2110611	0.786592
2	3	1724165	2128523	0.810029
4	4	1882097	2129962	0.883629

	DayofWeek	Number of Delays	Number of Flights	% of Flights Delayed
3	5	1994691	2136855	0.933470
1	6	1408132	1790874	0.786282
5	7	1754331	2017443	0.869581

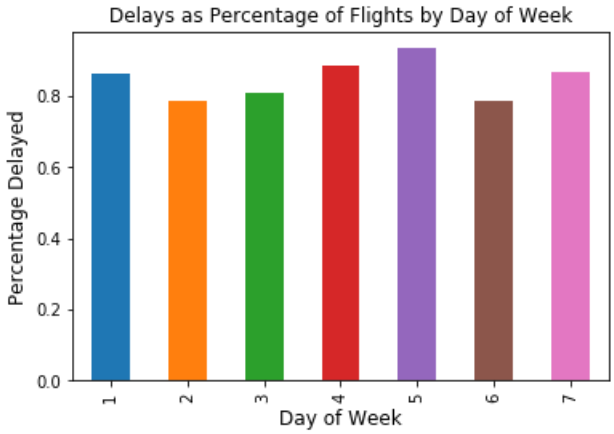
Put in a chart the number of delays as a percentage of flights, by day, looks as follows.

In [107]:

```
daily_delays_df_plot=daily_delays_df.plot(kind='bar', legend=False, title="Delays as Percentage of Flights by Day of Week", x='DayofWeek', y='% of Flights Delayed')
daily_delays_df_plot.set_xlabel("Day of Week", fontsize=12)
daily_delays_df_plot.set_ylabel("Percentage Delayed", fontsize=12)
values2 = daily_delays_df_plot.get_yticks()
monthly_delays_df_plot.set_yticklabels(['{:,.2%}'.format(x) for x in values2])
```

Out[107]:

```
[Text(0,0,'0.00%'),
 Text(0,0.2,'20.00%'),
 Text(0,0.4,'40.00%'),
 Text(0,0.6,'60.00%'),
 Text(0,0.8,'80.00%'),
 Text(0,1,'100.00%')]
```



I see no discernible pattern that would say there is a drop in delays on weekends (6=Saturday, 7=Sunday).

5. Are flights equally distributed througout the day? Plot the distribution of DepTime, ArrTime.

In [108]:

```
flight_times = airlines_df.select(['DepTime', 'ArrTime'])
flight_times.describe().show()
#Can't use .toPandas() yet still too many rows
```

```
+-------+-----------------+------------------+
|summary|          DepTime|           ArrTime|
+-------+-----------------+------------------+
|  count|         14462943|          14462943|
|   mean|1336.605297675433|1481.6940567665158|
| stddev|478.9956073954537| 506.2552705769898|
|    min|                1|                 1|
|    max|               NA|                NA|
+-------+-----------------+------------------+
```

First we will look at Departures:

In [109]:

```
departures = flight_times.groupby('DepTime').agg(count('DepTime'))
departures_df = departures.toPandas()
```

```
departures_df = departures.toPandas()
```

In [110]:

```
#the departure times are stored as objects. Must convert with this function before you order them
or it will read as strings
departures_df = departures_df.convert_objects(convert_numeric=True)
#you have a NA somewhere in there have to drop it so you can convert to int
departures_df = departures_df.dropna()
departures_df['DepTime'] = departures_df['DepTime'].astype('int64', copy=False)
departures_df = departures_df.sort_values(by=['DepTime'])
departures_df = departures_df.rename(columns={'DepTime': 'Departure Time', 'count(DepTime)': 'Numbe
r of Flights'})
```

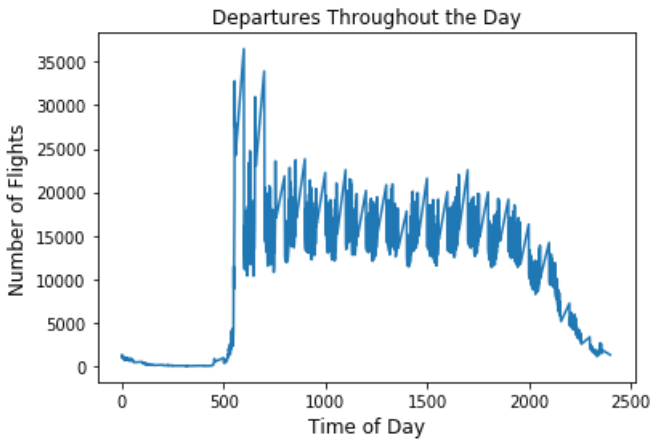
```
/software/Anaconda3-5.1.0-hadoop/lib/python3.6/site-packages/ipykernel/__main__.py:2:
FutureWarning: convert_objects is deprecated.  To re-infer data dtypes for object columns, use Dat
aFrame.infer_objects()
For all other conversions use the data-type specific converters pd.to_datetime, pd.to_timedelta an
d pd.to_numeric.
  from ipykernel import kernelapp as app
```

In [111]:

```
departures_df_plot=departures_df.plot(kind='line', legend=False, title="Departures Throughout the
Day", x='Departure Time', y='Number of Flights')
departures_df_plot.set_xlabel("Time of Day", fontsize=12)
departures_df_plot.set_ylabel("Number of Flights", fontsize=12)
```

Out[111]:

```
Text(0,0.5,'Number of Flights')
```



And now Arrivals:

In [112]:

```
arrivals = flight_times.groupby('ArrTime').agg(count('ArrTime'))
arrivals_df = arrivals.toPandas()
```

In [113]:

```
arrivals_df = arrivals_df.convert_objects(convert_numeric=True)
arrivals_df = arrivals_df.dropna()
arrivals_df['ArrTime'] = arrivals_df['ArrTime'].astype('int64', copy=False)
arrivals_df = arrivals_df.sort_values(by=['ArrTime'])
arrivals_df = arrivals_df.rename(columns={'ArrTime': 'Arrival Time', 'count(ArrTime)': 'Number of F
lights'})
```

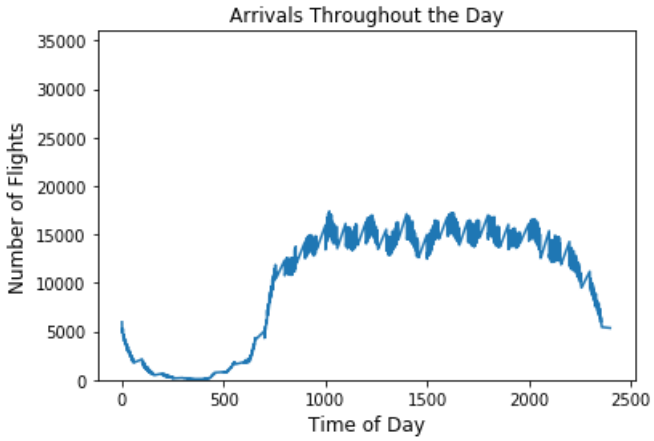
```
/software/Anaconda3-5.1.0-hadoop/lib/python3.6/site-packages/ipykernel/__main__.py:1:
FutureWarning: convert_objects is deprecated.  To re-infer data dtypes for object columns, use Dat
aFrame.infer_objects()
For all other conversions use the data-type specific converters pd.to_datetime, pd.to_timedelta an
d pd.to_numeric.
  if __name__ == '__main__':
```

In [114]:

```
arrivals_df_plot=arrivals_df.plot(kind='line', legend=False, title="Arrivals Throughout the Day", x
='Arrival Time', y='Number of Flights')
arrivals_df_plot.set_xlabel("Time of Day", fontsize=12)
arrivals_df_plot.set_ylabel("Number of Flights", fontsize=12)
#so it is on the same scale as departures
arrivals_df_plot.set_ylim(0, 36000)
```

Out[114]:

```
(0, 36000)
```



Clearly there is a pattern of departures and arrivals: a lull in the very early morning, a spike in the morning, more or less steady throughout the day, and then a drop after 2000 hrs (8pm). Interestingly, the arrivals are in a narrower band than the departures.

6. Do you see the worst delays at any certain times of the day? Contrast DepTime, ArrTime with CRSDepTime, CRSArrTime (scheduled arrival and departure time vs Computer Reserve System)

In contrasting (1) CRS Departure Time with Departure Time and (2) CRS Arrival Time with Arrival Time, we run into the following issue: Flights that are scheduled to leave early in the morning and end up departing before scheduled may show up as arriving extraordinarily late because it appears they left at the end of that day rather than the end of the preceding day. For example, if a flight is scheduled to depart at 12:01 am (0001 hours) on *Wednesday* and leaves two minutes early, it actually leaves at 11:59pm *Tuesday* (2359 hours). If we just contrast the CRS Departure Time with the Departure Time this nuance is lost and it appears that the flight left at 11:59pm *Wednesday*.

This is illustrated below.

In [115]:

```
delay_times = airlines_df.select(['DepTime', 'CRSDepTime', 'DepDelay', 'CRSArrTime', 'ArrTime', 'Ar
rDelay'])
#subset for just departure info and group by CRS Departure Time and get average departure time (ac
tual)
dep_delay_times = delay_times.groupby('CRSDepTime').agg(mean('DepTime'))
```

In [116]:

```
#flights that leave at 0001 hours
delay_times_filt = airlines_df.filter("CRSDepTime == 1")
delay_times_filt = delay_times_filt.orderBy('DepTime', ascending=False).toPandas()
delay_times_filt.iloc[20:25]
```

Out[116]:

	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	ArrDelay	DepDe
20	2008	8	6	3	3	1	627	629	DL	-2	2
21	2008	7	22	2	26	1	639	629	DL	10	25

	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	ArrDelay	DepDe
21	2008	7	22	2	26	1	659	629	DL	10	25
22	2008	9	21	7	35	1	647	628	DL	19	34
23	2008	7	23	3	2400	1	659	629	DL	30	-1
24	2008	7	26	6	2400	1	620	629	DL	-9	-1

Above we se an example of a flight that was scheduled to leave at 0001 hours, left at 2400 hours (left one minute early).

And below we see how this can distort a calculation of an Average Departure Time.

In [117]:

```
dep_delay_times_df=dep_delay_times.toPandas()
#there are a few CRSDepTime values where the Average Departure Time is NaN. Drop them.
dep_delay_times_df = dep_delay_times_df.dropna()
dep_delay_times_df = dep_delay_times_df.sort_values(by=['CRSDepTime'])
dep_delay_times_df.head(5)
```

Out[117]:

	CRSDepTime	avg(DepTime)
1191	0	16.000000
278	1	1394.006369
1104	2	2355.000000
405	5	546.233083
677	7	20.000000

As a result we will contrast CRSDepTime and CRSArrTime with actual delays in arrivals or departures (DepDelay, ArrDelay)

In [118]:

```
crs_delays = airlines_df.select(['CRSDepTime', 'DepDelay', 'CRSArrTime', 'ArrDelay'])
#negative values for delays should be mapped to 0 so as not to affect the average
crs_delays = crs_delays.\
withColumn("ArrDelay",when(col("ArrDelay") <0, 0).otherwise(col("ArrDelay"))).\
withColumn("DepDelay",when(col("DepDelay") <0, 0).otherwise(col("DepDelay")))
crs_dep_delays = crs_delays.groupby('CRSDepTime').agg(mean('DepDelay'))
crs_arr_delays = crs_delays.groupby('CRSArrTime').agg(mean('ArrDelay'))
crs_dep_delays_df=crs_dep_delays.toPandas()
crs_arr_delays_df=crs_arr_delays.toPandas()
crs_dep_delays_df = crs_dep_delays_df.dropna()
crs_arr_delays_df = crs_arr_delays_df.dropna()
crs_dep_delays_df = crs_dep_delays_df.sort_values(by=['CRSDepTime'])
crs_arr_delays_df = crs_arr_delays_df.sort_values(by=['CRSArrTime'])
crs_dep_delays_df.head(10)
```

Out[118]:

	CRSDepTime	avg(DepDelay)
1191	0	16.000000
276	1	7.910828
1103	2	0.000000
406	5	8.394737
675	7	13.000000
779	10	6.336397
162	12	7.733333
310	13	7.000000
1070	14	39.000000
483	15	10.058757

Problem solved!

In [119]:

```
crs_dep_delays_df = crs_dep_delays_df.rename(columns={'CRSDepTime': 'CRS Departure Time',
'avg(DepDelay)': 'Average Delay'})
crs_arr_delays_df = crs_arr_delays_df.rename(columns={'CRSArrTime': 'CRS Arrival Time',
'avg(ArrDelay)': 'Average Delay'})
```

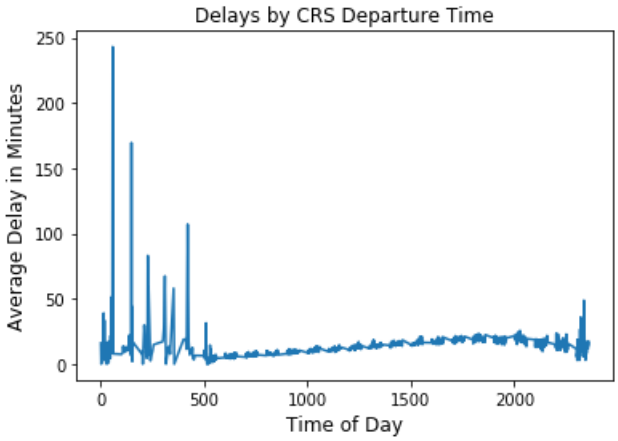
First the departure delays:

In [122]:

```
crs_dep_delays_df_plot=crs_dep_delays_df.plot(kind='line', legend=False, title="Delays by CRS
Departure Time", x='CRS Departure Time', y='Average Delay')
crs_dep_delays_df_plot.set_xlabel("Time of Day", fontsize=12)
crs_dep_delays_df_plot.set_ylabel("Average Delay in Minutes", fontsize=12)
```

Out[122]:

```
Text(0,0.5,'Average Delay in Minutes')
```



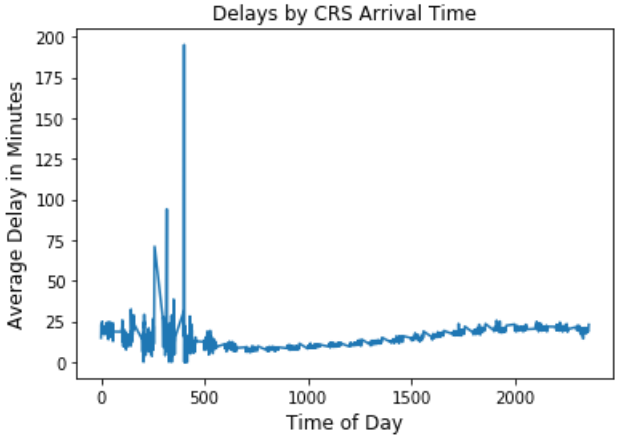
And now the arrival delays:

In [123]:

```
crs_arr_delays_df_plot=crs_arr_delays_df.plot(kind='line', legend=False, title="Delays by CRS
Arrival Time", x='CRS Arrival Time', y='Average Delay')
crs_arr_delays_df_plot.set_xlabel("Time of Day", fontsize=12)
crs_arr_delays_df_plot.set_ylabel("Average Delay in Minutes", fontsize=12)
```

Out[123]:

```
Text(0,0.5,'Average Delay in Minutes')
```



We see the same pattern of spikes prior to 0500 hours. There is also, quite naturally a lag from departure delays to arrival delays. A big reason for the large spike in the early morning is probably there are just fewer flights compared to later in the day, and so outliers have more effect.