FINISHED

```
import org.apache.spark.sql.functions._
import org.joda.time.format.DateTimeFormat
```

```
import org.apache.spark.sql.functions._
import org.joda.time.format.DateTimeFormat
```

Took 23 sec. Last updated by anonymous at January 28 2017, 6:38:56 PM.

FINISHED

```
val inputPath =  "/Users/vpandiyan/Downloads/*.csv"
val airTraffic = sqlContext.read
        .format("com.databricks.spark.csv")
        .option("header", "true") // Use first line of all files as header
        .option("delimiter", ",")
        .option("inferSchema", "true") // Automatically infer data types
        .load(inputPath)
```

```
inputPath: String = /Users/vpandiyan/Downloads/*.csv
airTraffic: org.apache.spark.sql.DataFrame = [Year: int, Month: int ... 27 more fields]
```

Took 3 min 27 sec. Last updated by anonymous at January 28 2017, 6:42:25 PM.

FINISHED

```
// Add extra features
val calcDayOfYear = udf(
    (dayOfMonth: Int, month: Int, year: Int) => {
        val dateFormat = DateTimeFormat.forPattern("dd/MM/yyyy")
        dateFormat.parseDateTime(s"$dayOfMonth/$month/$year").getDayOfYear()
    }
)
val calcRoute = udf(
    (origin: String, dest: String) => s"$origin - $dest"
)
val calcHourOfArrival = udf(
    (arrTime: String) => arrTime.slice(0,arrTime.size-2)
)
val featuredTraffic = airTraffic
    .withColumn("DayOfYear", calcDayOfYear(airTraffic("DayOfMonth"), airTraffic("Month"), air
    .withColumn("HourOfArr", calcHourOfArrival(airTraffic("ArrTime")))
    .withColumn("Route", calcRoute(airTraffic("Origin"), airTraffic("Dest")))
```

```
warning: Class org.joda.convert.FromString not found - continuing with a stub.
warning: Class org.joda.convert.ToString not found - continuing with a stub.
warning: Class org.joda.convert.ToString not found - continuing with a stub.
calcDayOfYear: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<fun
ction3>,IntegerType,Some(List(IntegerType, IntegerType, IntegerType)))
calcRoute: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<functio
n2>,StringType,Some(List(StringType, StringType)))
calcHourOfArrival: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction
(<function1>,StringType,Some(List(StringType)))
featuredTraffic: org.apache.spark.sql.DataFrame = [Year: int, Month: int ... 30 more fields]
```

Took 7 sec. Last updated by anonymous at January 28 2017, 6:42:38 PM.

FINISHED

```
// Register as Spark SQL Table
featuredTraffic.registerTempTable("air_traffic")
```

```
warning: there was one deprecation warning; re-run with -deprecation for details
```

Took 2 sec. Last updated by anonymous at January 28 2017, 6:42:47 PM.

FINISHED

```
%sql
select DayOfYear, count(*) as NrOfFlights, avg(DepDelay) as AvgDepDelay, avg(ArrDelay) as Avg
DayOfYear having NrOfFlights > 100000
```

DayOfYear	NrOfFlights	AvgDepDelay
31	333,830	6.77604
85	336,519	7.22741
137	335,712	8.40268
251	334,225	4.09414
65	338,508	9.01227
53	336,107	11.65395
255	336,123	4.81319
133	334,576	6.27433

Took 3 min 46 sec. Last updated by anonymous at January 28 2017, 6:46:39 PM.

FINISHED

```
%sql
select HourOfArr, count(*) as NrFlights, avg(ArrDelay) as AvgArrDelay from air_traffic where I
)|3(Wednesday)|4(Thursday)|5(Friday)|6(Saturday)|7(Sunday)} and Origin = "${org=ATL,ATL|PHX|P
```

org

day

HourOfArr	NrFlights	AvgArrDelay
7	11,978	-1.89848
15	51,476	7.46777
11	61,006	5.77468
3	227	187.52423
8	29,101	0.65427
22	56,236	14.68057
28	3	267
16	54,902	7.82906
5	13	286

Took 2 min 19 sec. Last updated by anonymous at January 28 2017, 6:49:44 PM.

FINISHED

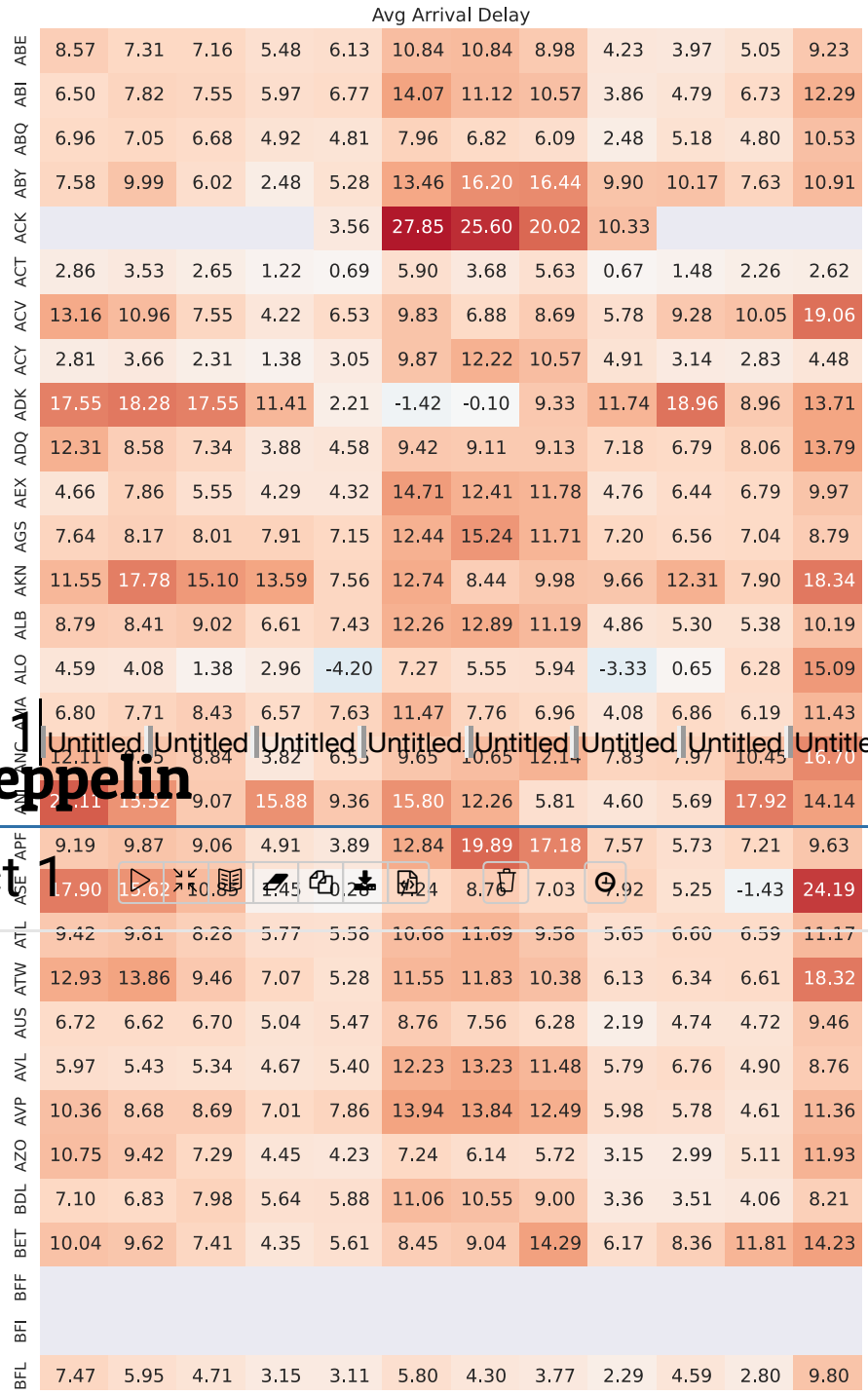
```
%pyspark
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

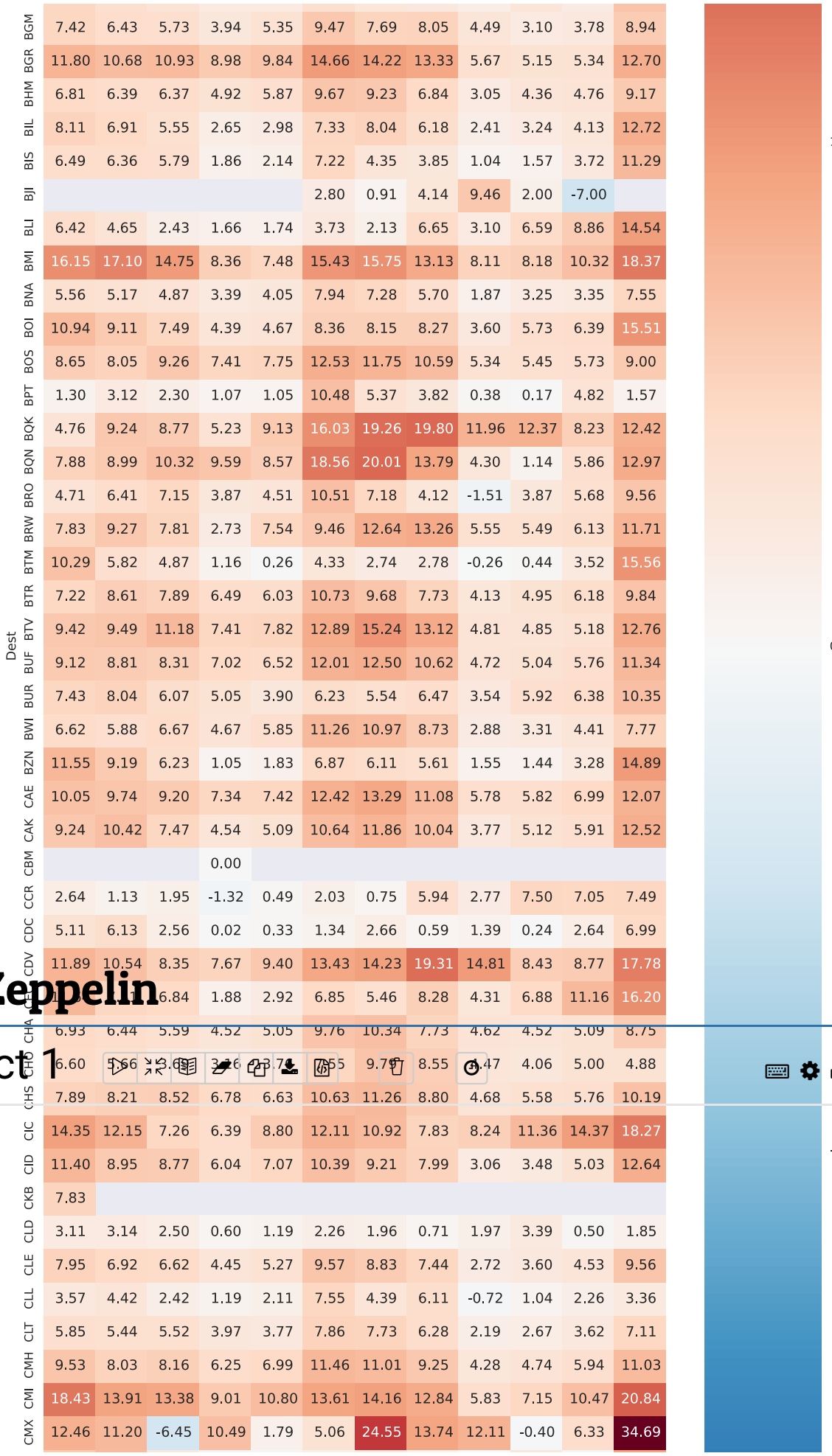
Took 0 sec. Last updated by anonymous at January 28 2017, 6:27:51 AM.

FINISHED

```
%pyspark
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

import StringIO
def show(p):
  img = StringIO.StringIO()
  p.savefig(img, format='svg')
  img.seek(0)
  print "%html " + img.buf

df = sqlContext.sql("SELECT Dest, Month, count(*) as NrOfFlights, avg(ArrDelay) as AvgArrDelay
)
data = df.toPandas()

value = "AvgArrDelay"
x = "Dest"
grouping = ["Month"]

heatmap_data = data.pivot_table(values=value, index=x, columns=grouping)
heatmap_data = heatmap_data[0:100]

a4_dims = (len(heatmap_data.columns),50)
fig, ax = plt.subplots(figsize=a4_dims)
ax.set_title("Avg Arrival Delay")
sns.heatmap(heatmap_data, ax=ax, annot=True, fmt=".02f")

show(plt)
```

Project 1 Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled

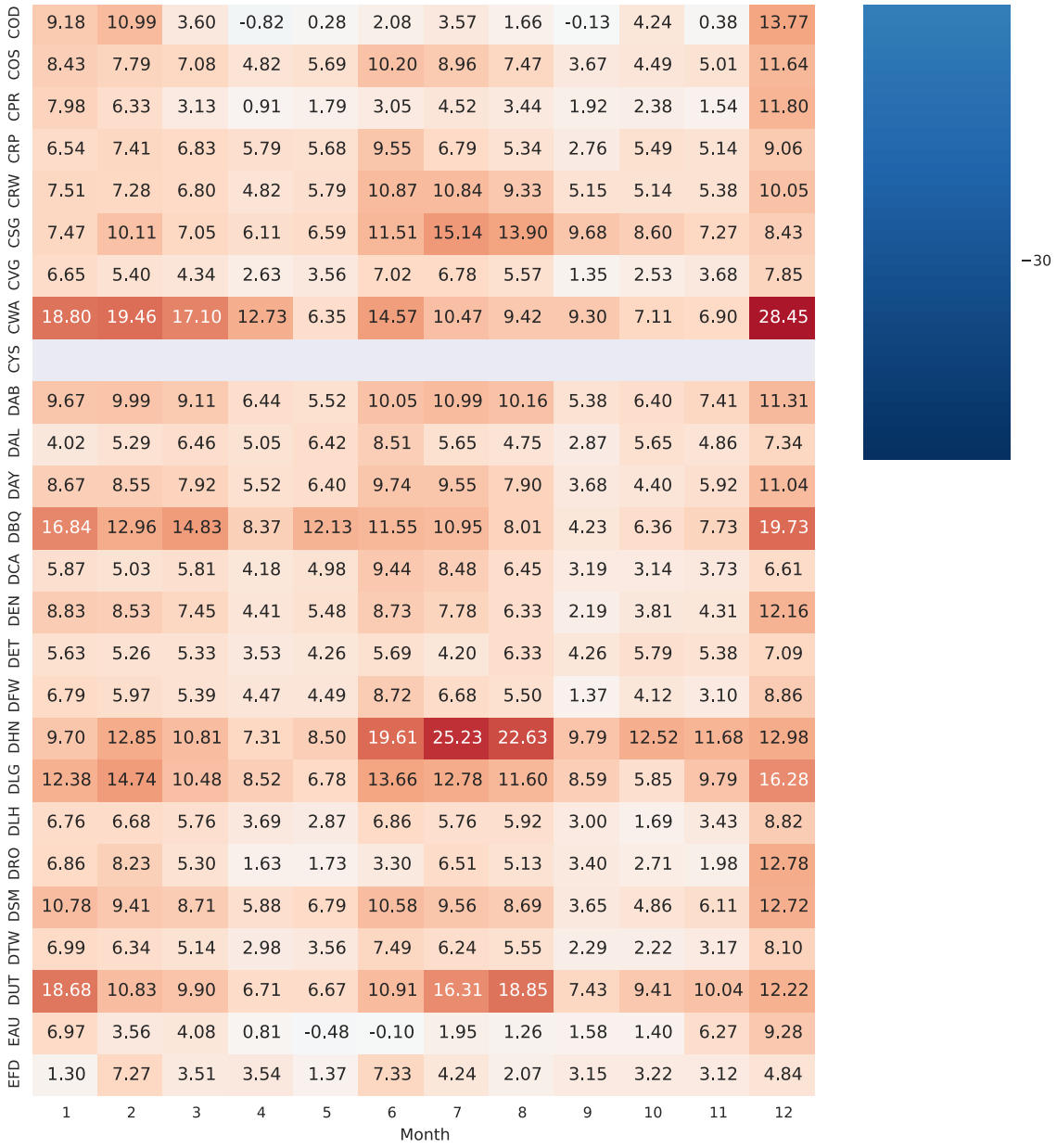

Avg Arrival Delay
ABE 8.57 7.31 7.16 5.48 6.13 10.84 10.84 8.98 4.23 3.97 5.05 9.23
ABI 6.50 7.82 7.55 5.97 6.77 14.07 11.12 10.57 3.86 4.79 6.73 12.29
ABQ 6.96 7.05 6.68 4.92 4.81 7.96 6.82 6.09 2.48 5.18 4.80 10.53
ABY 7.58 9.99 6.02 2.48 5.28 13.46 16.20 16.44 9.90 10.17 7.63 10.91
ACK 3.56 27.85 25.60 20.02 10.33
ACT 2.86 3.53 2.65 1.22 0.69 5.90 3.68 5.63 0.67 1.48 2.26 2.62
ACV 13.16 10.96 7.55 4.22 6.53 9.83 6.88 8.69 5.78 9.28 10.05 19.06
ACY 2.81 3.66 2.31 1.38 3.05 9.87 12.22 10.57 4.91 3.14 2.83 4.48
ADK 17.55 18.28 17.55 11.41 2.21 -1.42 -0.10 9.33 11.74 18.96 8.96 13.71
ADQ 12.31 8.58 7.34 3.88 4.58 9.42 9.11 9.13 7.18 6.79 8.06 13.79
AEX 4.66 7.86 5.55 4.29 4.32 14.71 12.41 11.78 4.76 6.44 6.79 9.97
AGS 7.64 8.17 8.01 7.91 7.15 12.44 15.24 11.71 7.20 6.56 7.04 8.79
AKN 11.55 17.78 15.10 13.59 7.56 12.74 8.44 9.98 9.66 12.31 7.90 18.34
ALB 8.79 8.41 9.02 6.61 7.43 12.26 12.89 11.19 4.86 5.30 5.38 10.19
ALO 4.59 4.08 1.38 2.96 -4.20 7.27 5.55 5.94 -3.33 0.65 6.28 15.09
MA 6.80 7.71 8.43 6.57 7.63 11.47 7.76 6.96 4.08 6.86 6.19 11.43
NC 12.11 8.84 3.82 6.55 9.65 10.65 12.14 7.83 7.97 10.45 16.70
AP 9.07 15.88 9.36 15.80 12.26 5.81 4.60 5.69 17.92 14.14
APF 9.19 9.87 9.06 4.91 3.89 12.84 19.89 17.18 7.57 5.73 7.21 9.63
ASE 17.90 10.89 8.76 7.03 5.25 -1.43 24.19
ATL 9.42 9.81 8.26 5.77 5.56 10.68 11.69 9.56 5.65 6.60 6.59 11.17
ATW 12.93 13.86 9.46 7.07 5.28 11.55 11.83 10.38 6.13 6.34 6.61 18.32
AUS 6.72 6.62 6.70 5.04 5.47 8.76 7.56 6.28 2.19 4.74 4.72 9.46
AVL 5.97 5.43 5.34 4.67 5.40 12.23 13.23 11.48 5.79 6.76 4.90 8.76
AVP 10.36 8.68 8.69 7.01 7.86 13.94 13.84 12.49 5.98 5.78 4.61 11.36
AZO 10.75 9.42 7.29 4.45 4.23 7.24 6.14 5.72 3.15 2.99 5.11 11.93
BDL 7.10 6.83 7.98 5.64 5.88 11.06 10.55 9.00 3.36 3.51 4.06 8.21
BET 10.04 9.62 7.41 4.35 5.61 8.45 9.04 14.29 6.17 8.36 11.81 14.23
BFF
BFI
BFL 7.47 5.95 4.71 3.15 3.11 5.80 4.30 3.77 2.29 4.59 2.80 9.80
Project 1
Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled
Zeppelin
Project 1
30
default



Zeppelin

Project 1

default



Took 10 min 27 sec. Last updated by anonymous at January 28 2017, 7:00:16 PM.

FINISHED

```
%pyspark
from wordcloud import WordCloud
```

```
import StringIO
def show(p):
  img = StringIO.StringIO()
  p.savefig(img, format='svg')
  img.seek(0)
  print "%html " + img.buf

# Create route frequencies
routes = sqlContext.sql("SELECT Route, count(*) as Count FROM air_traffic GROUP BY Route").co
route_freq = [(x[0],x[1]) for x in routes]

# Generate word cloud image
wordcloud = WordCloud().generate_from_frequencies(route_freq)
image = wordcloud.to_image()
image.show()
```

Took 3 min 13 sec. Last updated by anonymous at January 28 2017, 7:06:21 PM.

READY