Register_YimPS1

April 11, 2019

Problem Set 1, due January 10th at 5:30pm

0.0.1 Before You Start

Make sure to at least take a basic tutorial in the IPython notebook, otherwise you'll be totally lost. For this problem set, you should download IMT574-PS1.ipynb and the flights.zip dataset from Canvas. Create a local copy of the notebook and rename it LASTNAME_FIRSTNAME-PS1.ipynb. Then edit your renamed file directly in your browser by typing:

```
ipython notebook <name_of_downloaded_file>
```

You should also make sure the following libraries load correctly (click on the box below and hit Ctrl-Enter)

```
In [2]: # #IPython is what you are using now to run the notebook
        # import IPython
        # print "IPython version:      %6.6s (need at least 1.0)" % IPython.__version__

        # Numpy is a library for working with Arrays
        import numpy as np
        print ("Numpy version:        %6.6s (need at least 1.7.1)" % np.__version__)

        # SciPy implements many different numerical algorithms
        import scipy as sp
        print ("SciPy version:        %6.6s (need at least 0.12.0)" % sp.__version__)

        # Pandas makes working with data tables easier
        import pandas as pd
        print ("Pandas version:       %6.6s (need at least 0.11.0)" % pd.__version__)

        # Module for plotting
        import matplotlib
        print ("Mapltolib version:    %6.6s (need at least 1.2.1)" % matplotlib.__version__)

        # SciKit Learn implements several Machine Learning algorithms
        import sklearn
        print ("Scikit-Learn version: %6.6s (need at least 0.13.1)" % sklearn.__version__)
```

```
Numpy version:        1.15.4 (need at least 1.7.1)
SciPy version:         1.1.0 (need at least 0.12.0)
Pandas version:       0.23.4 (need at least 0.11.0)
Mapltolib version:     2.2.3 (need at least 1.2.1)
Scikit-Learn version: 0.19.2 (need at least 0.13.1)
```

0.1 About the Problem Set:

This is the same problem set used by Emma Spiro in INFX573. The only difference is that instead of doing the problem set in R, you will use Python and the IPython notebook.

0.2 Instructions:

In this problem set you will perform a basic exploratory analysis on an example dataset, bringing to bear all of your new skills in data manipulation and visualization. You will be required to submit well commented python code, documenting all code used in this problem set, along with a write up answering all questions below. Use figures as appropriate to support your answers, and when required by the problem. This data set uses the NYCFlights13 dataset. You can download the dataset from canvas. Selected questions ask you to answer in multiple ways. Make sure to provide different functions or ways for answering the same question. This will help you see that most data questions can be answered in different ways even with the same software language.

```
In [3]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [4]: flights= pd.read_csv('flights/flights.csv') #read in the dataframe
```

```
In [5]: print (flights.shape) # matrix dimensions
        print (flights.columns) # colnames
        print (flights.dtypes) #data types
```

```
(336776, 17)
Index(['Unnamed: 0', 'year', 'month', 'day', 'dep_time', 'dep_delay',
       'arr_time', 'arr_delay', 'carrier', 'tailnum', 'flight', 'origin',
       'dest', 'air_time', 'distance', 'hour', 'minute'],
      dtype='object')
Unnamed: 0      int64
year            int64
month           int64
day             int64
dep_time      float64
dep_delay     float64
arr_time      float64
arr_delay     float64
carrier        object
tailnum        object
flight          int64
```

```
origin         object
dest           object
air_time      float64
distance        int64
hour          float64
minute        float64
dtype: object
```

In [6]:
```
# what  are all the possible destinations? (just need the unique!)
destinations = flights.dest.unique()
print(destinations)

#take a look at the dataframe
flights.head(10)
```

```
['IAH' 'MIA' 'BQN' 'ATL' 'ORD' 'FLL' 'IAD' 'MCO' 'PBI' 'TPA' 'LAX' 'SFO'
 'DFW' 'BOS' 'LAS' 'MSP' 'DTW' 'RSW' 'SJU' 'PHX' 'BWI' 'CLT' 'BUF' 'DEN'
 'SNA' 'MSY' 'SLC' 'XNA' 'MKE' 'SEA' 'ROC' 'SYR' 'SRQ' 'RDU' 'CMH' 'JAX'
 'CHS' 'MEM' 'PIT' 'SAN' 'DCA' 'CLE' 'STL' 'MYR' 'JAC' 'MDW' 'HNL' 'BNA'
 'AUS' 'BTV' 'PHL' 'STT' 'EGE' 'AVL' 'PWM' 'IND' 'SAV' 'CAK' 'HOU' 'LGB'
 'DAY' 'ALB' 'BDL' 'MHT' 'MSN' 'GSO' 'CVG' 'BUR' 'RIC' 'GSP' 'GRR' 'MCI'
 'ORF' 'SAT' 'SDF' 'PDX' 'SJC' 'OMA' 'CRW' 'OAK' 'SMF' 'TUL' 'TYS' 'OKC'
 'PVD' 'DSM' 'PSE' 'BHM' 'CAE' 'HDN' 'BZN' 'MTJ' 'EYW' 'PSP' 'ACK' 'BGR'
 'ABQ' 'ILM' 'MVY' 'SBN' 'LEX' 'CHO' 'TVC' 'ANC' 'LGA']
```

Out[6]:

	Unnamed: 0	year	month	day	dep_time	dep_delay	arr_time	arr_delay	\
0	1	2013	1	1	517.0	2.0	830.0	11.0	
1	2	2013	1	1	533.0	4.0	850.0	20.0	
2	3	2013	1	1	542.0	2.0	923.0	33.0	
3	4	2013	1	1	544.0	-1.0	1004.0	-18.0	
4	5	2013	1	1	554.0	-6.0	812.0	-25.0	
5	6	2013	1	1	554.0	-4.0	740.0	12.0	
6	7	2013	1	1	555.0	-5.0	913.0	19.0	
7	8	2013	1	1	557.0	-3.0	709.0	-14.0	
8	9	2013	1	1	557.0	-3.0	838.0	-8.0	
9	10	2013	1	1	558.0	-2.0	753.0	8.0	

	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
0	UA	N14228	1545	EWR	IAH	227.0	1400	5.0	17.0
1	UA	N24211	1714	LGA	IAH	227.0	1416	5.0	33.0
2	AA	N619AA	1141	JFK	MIA	160.0	1089	5.0	42.0
3	B6	N804JB	725	JFK	BQN	183.0	1576	5.0	44.0
4	DL	N668DN	461	LGA	ATL	116.0	762	5.0	54.0
5	UA	N39463	1696	EWR	ORD	150.0	719	5.0	54.0
6	B6	N516JB	507	EWR	FLL	158.0	1065	5.0	55.0
7	EV	N829AS	5708	LGA	IAD	53.0	229	5.0	57.0

```
8       B6   N593JB      79     JFK  MCO      140.0         944    5.0     57.0
9       AA   N3ALAA     301     LGA  ORD      138.0         733    5.0     58.0
```

0.3 Some Tips

- This assignment involves extensive Data frame splitting and aggregation. You should look into the details of the methods groupby, transform, sum, count, mean etc
- Many of the tasks in the assignment can be done either through the Pandas Data Frame or by converting the data frames to Series. Many of the methods in the numpy are applicable to Series only. When stuck, try to explore the type of object (Pandas Data Frame or Numpy Series) you are dealing with.

0.4 Question 1

Let's explore flights from NYC to Seattle. Use the flights dataset to answer the following questions.

(a) How many flights were there from NYC airports to Seattle in 2013?

```
In [7]: # Your code here
        to_SEA = flights[flights.dest=="SEA"]

        len(to_SEA) # there were 3923 flights to the SEA destination out of NYC airports
        # (all the origins are some airport in NYC so we don't need to specify)
        # we can also look at shape and look at the number of columns, which is also 3923
        to_SEA.shape
```

```
Out[7]: (3923, 17)
```

Answer 1a: 3923 flights from *any* NYC airport to Seattle in 2013

(b) How many airlines fly from NYC to Seattle?

```
In [8]: # airlines is stored under "carrier"
        airlines_to_SEA = to_SEA.carrier.unique()

        print(airlines_to_SEA)
        len(airlines_to_SEA)
```

```
['AS' 'DL' 'UA' 'B6' 'AA']
```

```
Out[8]: 5
```

Answer 1b: 5

(c) How many unique air planes fly from NYC to Seattle?

```
In [9]: # Airplanes are the "tailnum" so let's look at the unique ones
        airplanes_to_SEA = to_SEA.tailnum.unique()
        print(airplanes_to_SEA)
        len(airplanes_to_SEA)
```

```
['N594AS' 'N3760C' 'N45440' 'N37464' 'N503JB' 'N77296' 'N553AS' 'N3ETAA'
 'N3772H' 'N76523' 'N592AS' 'N3768' 'N37434' 'N712JB' 'N78285' 'N3745B'
 'N552AS' 'N717TW' 'N3BAAA' 'N73275' 'N37263' 'N533AS' 'N679DA' 'N430UA'
 'N607JB' 'N496UA' 'N3HPAA' 'N3765' 'N579AS' 'N713TW' 'N445UA' 'N591JB'
 'N478UA' 'N3AUAA' 'N3734B' 'N546AS' 'N175DN' 'N574UA' 'N599JB' 'N33714'
 'N558AS' 'N3ANAA' 'N3759' 'N568AS' 'N624AG' 'N438UA' 'N519JB' 'N551AS'
 'N3HNAA' 'N78448' 'N3771K' 'N537AS' 'N727TW' 'N433UA' 'N663JB' 'N402UA'
 'N408AS' 'N3764D' 'N3JLAA' 'N76519' 'N794JB' 'N17229' 'N3BKAA' 'N407AS'
 'N563AS' 'N18243' 'N709JB' 'N403AS' 'N3BDAA' 'N3762Y' 'N16217' 'N513AS'
 'N706TW' 'N632JB' 'N466UA' 'N402AS' 'N3753' 'N3ECAA' 'N508AS' 'N712TW'
 'N465UA' 'N504JB' 'N417UA' 'N320AS' 'N3GJAA' 'N3757D' 'N67171' 'N827UA'
 'N597JB' 'N33284' 'N565AS' 'N3GCAA' 'N639DL' 'N820UA' 'N579JB' 'N531AS'
 'N435UA' 'N510JB' 'N481UA' 'N3754A' 'N3HAAA' 'N596AS' 'N718TW' 'N36280'
 'N651JB' 'N12238' 'N3JHAA' 'N3756' 'N589AS' 'N33203' 'N630JB' 'N14214'
 'N3766' 'N3EWAA' 'N506AS' 'N493UA' 'N586JB' 'N462UA' 'N3746H' 'N3HVAA'
 'N518AS' 'N721TW' 'N490UA' 'N775JB' 'N3HRAA' 'N723TW' 'N535AS' 'N840UA'
 'N789JB' 'N73291' 'N597AS' 'N3758Y' 'N3DYAA' 'N556AS' 'N710TW' 'N564JB'
 'N75436' 'N3761R' 'N305AS' 'N577AS' 'N458UA' 'N652JB' 'N461UA' 'N3AYAA'
 'N548AS' 'N687DL' 'N14242' 'N638JB' 'N3EPAA' 'N17245' 'N3HCAA' 'N3763D'
 'N529AS' 'N491UA' 'N634JB' 'N549AS' 'N464UA' 'N447UA' 'N587JB' 'N443UA'
 'N307AS' 'N3AAAA' 'N779JB' 'N323AS' 'N3748Y' 'N3DMAA' 'N584AS' 'N829UA'
 'N643JB' 'N39416' 'N3751B' 'N454UA' 'N535JB' 'N3HDAA' 'N523AS' 'N37298'
 'N629JB' 'N76505' 'N3GAAA' 'N704X' 'N37252' 'N559JB' 'N78501' 'N3755D'
 'N705TW' 'N608JB' 'N480UA' 'N3FKAA' 'N3769L' 'N807JB' 'N57439' 'N169DZ'
 'N37456' 'N809UA' 'N39450' 'N648JB' 'N182DN' 'N424UA' 'N565JB' 'N850UA'
 'N1201P' 'N760JB' 'N53441' 'N1605' 'N468UA' 'N623JB' 'N583AS' 'N419UA'
 'N520AS' 'N529JB' 'N178DZ' 'N427UA' 'N633JB' 'N526AS' 'N3GWAA' 'N826UA'
 'N419AS' 'N554NW' 'N621JB' 'N38454' 'N173DZ' 'N434UA' 'N711ZX' 'N3GUAA'
 'N423AS' 'N79402' 'N184DN' 'N484UA' 'N822UA' 'N641JB' 'N47414' 'N557NW'
 'N439UA' 'N3FUAA' 'N855UA' 'N27477' 'N1200K' 'N516JB' 'N3CAAA' 'N802UA'
 'N703TW' 'N655JB' 'N1604R' 'N827JB' 'N838UA' 'N3JWAA' 'N77430' 'N746JB'
 'N172DZ' 'N482UA' 'N3CFAA' 'N844UA' 'N409AS' 'N729JB' 'N68453' 'N1602'
 'N444UA' 'N3EFAA' 'N834UA' 'N524AS' 'N563JB' 'N75432' 'N176DN' 'N517AS'
 'N3JCAA' 'N172DN' 'N421UA' 'N624JB' 'N413AS' 'N3AKAA' 'N835UA' 'N702TW'
 'N809JB' 'N186DN' 'N456UA' 'N3FPAA' 'N851UA' 'N640JB' 'N37420' 'N766JB'
 'N538AS' 'N605JB' 'N174DZ' 'N306AS' 'N815UA' 'N534JB' 'N175DZ' 'N842UA'
 'N37422' 'N763JB' 'N3KYAA' 'N811UA' 'N78438' 'N171DZ' 'N812UA' 'N3KLAA'
 'N707TW' 'N39461' 'N639JB' 'N1603' 'N414UA' 'N646JB' 'N522US' 'N846UA'
 'N709TW' 'N612JB' 'N75435' 'N411UA' 'N618JB' 'N692DL' 'N823UA' 'N3KMAA'
 'N303AS' 'N512AS' 'N524JB' 'N37474' 'N36469' 'N3ADAA' 'N853UA' 'N36444'
 'N185DN' 'N431AS' 'N808UA' 'N523JB' 'N442UA' 'N592JB' 'N836UA' 'N659JB'
 'N828MH' 'N452UA' 'N3DVAA' 'N722TW' 'N76522' 'N3CDAA' 'N809NW' 'N73283'
 'N463UA' 'N554JB' 'N459UA' 'N625JB' 'N3AEAA' 'N38473' 'N536JB' 'N806UA'
 'N3BBAA' 'N38424' 'N474UA' 'N627JB' 'N34460' 'N3FEAA' 'N487UA' 'N3FJAA'
 'N847UA' 'N520JB' 'N17244' 'N824UA' 'N796JB' 'N433AS' 'N475UA' 'N190DN'
 'N635JB' 'N3FDAA' 'N521JB' 'N3ESAA' 'N841UA' 'N75433' 'N422UA' 'N420UA'
 'N3ABAA' 'N38451' 'N429UA' 'N598JB' 'N3EKAA' 'N457UA' 'N28478' 'N432UA'
 'N658JB' 'N3BEAA' 'N38446' 'N508JB' 'N195DN' 'N637JB' 'N34455' 'N36447'
```

'N176DZ' 'N472UA' 'N3FWAA' 'N39423' 'N477UA' 'N401UA' 'N661JB' 'N3ALAA'
'N38459' 'N183DN' 'N606JB' 'N3DNAA' 'N784JB' 'N3HSAA' 'N498UA' 'N37290'
'N590JB' 'N404UA' 'N3KPAA' 'N37466' 'N3KSAA' 'N38458' 'N39463' 'N416UA'
'N3HHAA' 'N440UA' 'N3CKAA' 'N403UA' 'N187DN' 'N3DJAA' 'N73445' 'N440AS'
'N3CGAA' 'N39418' 'N497UA' 'N37471' 'N423UA' 'N71411' 'N435AS' 'N3HMAA'
'N636JB' 'N613JB' 'N3HYAA' 'N6702' 'N17730' 'N3BNAA' 'N448UA' 'N37465'
'N188DN' 'N804JB' 'N3GKAA' 'N657JB' 'N3KJAA' 'N470UA' 'N552JB' 'N3FXAA'
'N413UA' 'N570JB' 'N567UA' 'N156DL' 'N418UA' 'N3JMAA' 'N81449' 'N406UA'
'N556JB' 'N3BHAA' 'N30401' 'N479UA' 'N437UA' 'N154DL' 'N3HBAA' 'N3HJAA'
'N520UA' 'N3BWAA' 'N319AS' 'N446UA' 'N3CXAA' 'N38443' 'N191DN' 'N3HTAA'
'N594JB' 'N3HEAA' 'N3ELAA' 'N75426' 'N3CCAA' 'N603JB' 'N467UA' 'N442AS'
'N528UA' 'N595JB' 'N3GBAA' 'N453UA' 'N196DN' 'N3KVAA' 'N428UA' 'N703JB'
'N3JPAA' 'N199DN' 'N828JB' 'N3DEAA' 'N37468' 'N37293' 'N3EEAA' 'N27724'
'N36472' 'N3DAAA' 'N39728' 'N565UA' 'N656JB' 'N21723' 'N18220' 'N76269'
'N3DPAA' 'N75425' 'N24715' 'N831UA' 'N3JNAA' 'N27722' 'N76528' 'N3AWAA'
'N854UA' 'N73278' 'N38727' 'N807UA' 'N28457' 'N16713' 'N553UA' 'N15712'
'N3HLAA' 'N14250' 'N15710' 'N3GYAA' 'N514AS' 'N832UA' 'N3LDAA' 'N562JB'
'N23721' 'N37287' 'N3JFAA' nan 'N647DL' 'N528AS' 'N3HWAA' 'N819UA'
'N615JB' 'N36207' 'N593AS' 'N3FNAA' 'N3747D' 'N532AS' 'N485UA' 'N476UA'
'N3749D' 'N76504' 'N36247' 'N569AS' 'N519AS' 'N13248' 'N24212' 'N3EYAA'
'N3773D' 'N590AS' 'N3ERAA' 'N371DA' 'N813UA' 'N706JB' 'N378DA' 'N436UA'
'N385DN' 'N3FFAA' 'N27213' 'N3DGAA' 'N76508' 'N530AS' 'N538UA' 'N3HGAA'
'N3732J' 'N562AS' 'N76254' 'N537UA' 'N3JKAA' 'N3737C' 'N3APAA' 'N302AS'
'N380DA' 'N585AS' 'N577UA' 'N3JVAA' 'N392DA' 'N87527' 'N3752' 'N76529'
'N583JB' 'N569UA' 'N3EJAA' 'N35204' 'N578UA' 'N566AS' 'N77295' 'N3CHAA'
'N849UA' 'N3740C' 'N3EDAA' 'N76516' 'N503UA' 'N38257' 'N39297' 'N3EAAA'
'N73299' 'N581AS' 'N37427' 'N3DLAA' 'N593JB' 'N16732' 'N665JB' 'N558UA'
'N3767' 'N900PC' 'N541UA' 'N37700' 'N87531' 'N3ENAA' 'N38403' 'N79521'
'N533UA' 'N576UA' 'N77518' 'N547UA' 'N675DL' 'N14231' 'N590UA' 'N3GHAA'
'N504UA' 'N3730B' 'N585UA' 'N587AS' 'N527JB' 'N527UA' 'N3FHAA' 'N564UA'
'N309AS' 'N669DN' 'N180DN' 'N3CYAA' 'N76514' 'N588UA' 'N33209' 'N595UA'
'N516UA' 'N661DN' 'N33292' 'N3750D' 'N584UA' 'N12218' 'N587UA' 'N602DL'
'N534UA' 'N73259' 'N594UA' 'N570AS' 'N3742C' 'N580UA' 'N3CMAA' 'N579UA'
'N177DZ' 'N568JB' 'N78506' 'N586UA' 'N559AS' 'N571UA' 'N318AS' 'N662DN'
'N27239' 'N3KBAA' 'N517JB' 'N391DA' 'N543UA' 'N516AS' 'N3BVAA' 'N545UA'
'N3741S' 'N598UA' 'N3FSAA' 'N568UA' 'N3BPAA' 'N572UA' 'N511UA' 'N3FVAA'
'N11206' 'N530UA' 'N581UA' 'N509JB' 'N3DTAA' 'N13716' 'N684DA' 'N33286'
'N77431' 'N26226' 'N505JB' 'N3HKAA' 'N535UA' 'N527AS' 'N37277' 'N3BSAA'
'N37281' 'N3EGAA' 'N525AS' 'N3744F' 'N37255' 'N79279' 'N3EHAA' 'N77258'
'N539UA' 'N561UA' 'N383DN' 'N87507' 'N3BFAA' 'N570UA' 'N37437' 'N3736C'
'N6711M' 'N3DCAA' 'N3GRAA' 'N586AS' 'N78511' 'N3FMAA' 'N830UA' 'N3CPAA'
'N549UA' 'N662JB' 'N33294' 'N397DA' 'N398DA' 'N693DL' 'N556UA' 'N381DN'
'N390DA' 'N817UA' 'N585JB' 'N387DA' 'N649JB' 'N76503' 'N73270' 'N3CRAA'
'N379DA' 'N6710E' 'N559UA' 'N3CWAA' 'N34282' 'N3BYAA' 'N76502' 'N526JB'
'N33289' 'N14237' 'N3GEAA' 'N660DL' 'N645DL' 'N36272' 'N640DL' 'N72405'
'N14230' 'N670DN' 'N315AS' 'N6703D' 'N3HXAA' 'N494UA' 'N507UA' 'N592UA'
'N13718' 'N39415' 'N655DL' 'N3KCAA' 'N78524' 'N654DL' 'N24706' 'N3DUAA'
'N39726' 'N37253' 'N3BUAA' 'N667DN' 'N3KDAA' 'N642DL' 'N3CUAA' 'N658DL'

```
 'N685DA' 'N610DL' 'N24729' 'N31412' 'N6701' 'N3AVAA' 'N87512' 'N24702'
 'N3JJAA' 'N77510' 'N3GLAA' 'N3DXAA' 'N3DSAA' 'N563UA' 'N26208' 'N14704'
 'N506JB' 'N37470' 'N26210' 'N6704Z' 'N26215' 'N609DL' 'N75429' 'N584JB'
 'N3JXAA' 'N6708D' 'N3FLAA' 'N650DL' 'N17719' 'N14228' 'N651DL' 'N75428'
 'N573UA' 'N519UA' 'N6707A' 'N25705' 'N695DL' 'N35407' 'N3JYAA' 'N643DL'
 'N638DL' 'N768JB' 'N589JB' 'N588JB' 'N821JB' 'N6700' 'N509UA' 'N614DL'
 'N3HUAA' 'N644DL' 'N558JB' 'N3BJAA' 'N54241' 'N6706Q' 'N37419' 'N649DL'
 'N705JB' 'N3ACAA' 'N6716C' 'N696DL' 'N6705Y' 'N76526' 'N534AS' 'N27421'
 'N806JB' 'N715JB' 'N412UA' 'N449UA' 'N38467' 'N415UA' 'N564AS' 'N3EXAA'
 'N431UA' 'N407UA' 'N53442' 'N410UA' 'N653JB' 'N37413' 'N3GDAA' 'N405UA'
 'N488UA' 'N3GGAA' 'N455UA' 'N37462' 'N174DN' 'N68452' 'N3GNAA' 'N38268'
 'N547JB' 'N708JB' 'N825UA' 'N3JTAA' 'N471UA' 'N580JB' 'N562UA' 'N3GSAA'
 'N3GXAA' 'N75410' 'N531JB' 'N851NW' 'N526UA' 'N3CEAA' 'N3AFAA' 'N489UA'
 'N32404' 'N3AHAA' 'N492UA' 'N3CLAA' 'N569JB' 'N3CTAA' 'N495UA' 'N3CJAA'
 'N593UA' 'N441UA' 'N536AS' 'N3KTAA' 'N409UA' 'N571JB' 'N3DRAA' 'N36476'
 'N425UA' 'N644JB' 'N848UA' 'N33262' 'N805JB' 'N73276' 'N16065' 'N37273'
 'N33266' 'N35271' 'N818UA' 'N566JB' 'N3BXAA' 'N76265' 'N35260' 'N3AJAA'
 'N839UA' 'N3AGAA' 'N816UA' 'N3HFAA' 'N16701' 'N793JB' 'N3FGAA' 'N469UA'
 'N177DN' 'N3JRAA' 'N837UA' 'N3EBAA' 'N3KWAA' 'N426UA' 'N3BCAA' 'N178DN'
 'N645JB' 'N3GTAA' 'N804UA' 'N560AS' 'N3ARAA' 'N3DFAA' 'N3BMAA' 'N3BGAA'
 'N3KEAA' 'N3BLAA' 'N408UA' 'N833UA' 'N557AS' 'N483UA' 'N814UA' 'N561JB']
```

```
Out[9]: 936
```

Answer 1c: 936

(d) What is the average arrival delay for flights from NC to Seattle?

```
In [10]: # arrival delay = 'arr_delay' and it can be positive or negative, because sometimes t
         to_SEA['arr_delay'].mean()
         # turns out the mean isn't really too informative probably, because it basically come
```

```
Out[10]: -1.0990990990990992
```

Answer 1d: -1.099 minutes... so all those delayed arrival times you've ever experienced are just fake (just kidding)

(e) What proportion of flights to Seattle come from each NYC airport? Provide multiple ways of answering the question.

```
In [11]: # 1
         # the kind of messy way to do this is with different partitioning and kind of manuall
         # so let's grab the different airlines...
         to_SEA.head(10) # we can actually see that there's only two airline origins

         print(to_SEA.origin.unique()) # it's only EWR and JFK
         jfk = len(to_SEA[to_SEA.origin=="JFK"])/len(to_SEA)
         print(jfk)
```

```
        ewr = len(to_SEA[to_SEA.origin=="EWR"])/len(to_SEA)
        print(ewr)

        #SANITY CHECK (passed)
        assert(jfk+ewr==1)

        # 2
        # definitely a way to do this with some pandas manipulation that isn't lame
        #group by the NYC airports
        # summarise it by taking the count out of the total count
        to_SEA=to_SEA.fillna("NA") #fill in the NAs because otherwise the counts are off
        total = len(to_SEA)
        props = to_SEA.groupby(to_SEA.origin).count()/total
        # we come up with the same answer which is:
        #EWR: .4667
        #JFK: .5333
```

```
['EWR' 'JFK']
0.5332653581442773
0.46673464185572267
```

Answer 1e: EWR: .4667 JFK: .5333

0.5 Question 2

Flights are often delayed. Consider the following questions exploring delay patterns.

(a) Which date has the largest average departure delay? Which date has the largest average arrival delay?

```
In [12]: # I assume we are no longer working with just SEATTLE and have moved back to looking 

        # Here is the absolute maximum, but we care about largest AVERAGE
        flights[flights.dep_delay==flights.dep_delay.max()]
        # looks like it's January 9th, 2013
        flights2=flights[(flights.dep_delay>=0) & (flights.arr_delay>=0)]
        split_date=pd.DataFrame(flights2.groupby(by=["year","month","day"],as_index=False)['de

        split_date=pd.DataFrame(split_date.assign(total= lambda split_date: split_date.dep_de

        print(split_date[split_date.dep_delay==max(split_date.dep_delay)])
        print(split_date[split_date.arr_delay==max(split_date.arr_delay)])
```

```
     year  month  day   dep_delay   arr_delay       total
244  2013      9    2  113.336609  109.776413  223.113022
     year  month  day   dep_delay   arr_delay       total
254  2013      9   12  105.930108  122.870968  228.801075
```

Answer 2a: Largest Average Departure Delay: September 2, 2013
Answer 2a: Largest Average Arrival Delay: September 12, 2013

(b) What was the worst day to fly out of NYC in 2013 if you dislike delayed flights?

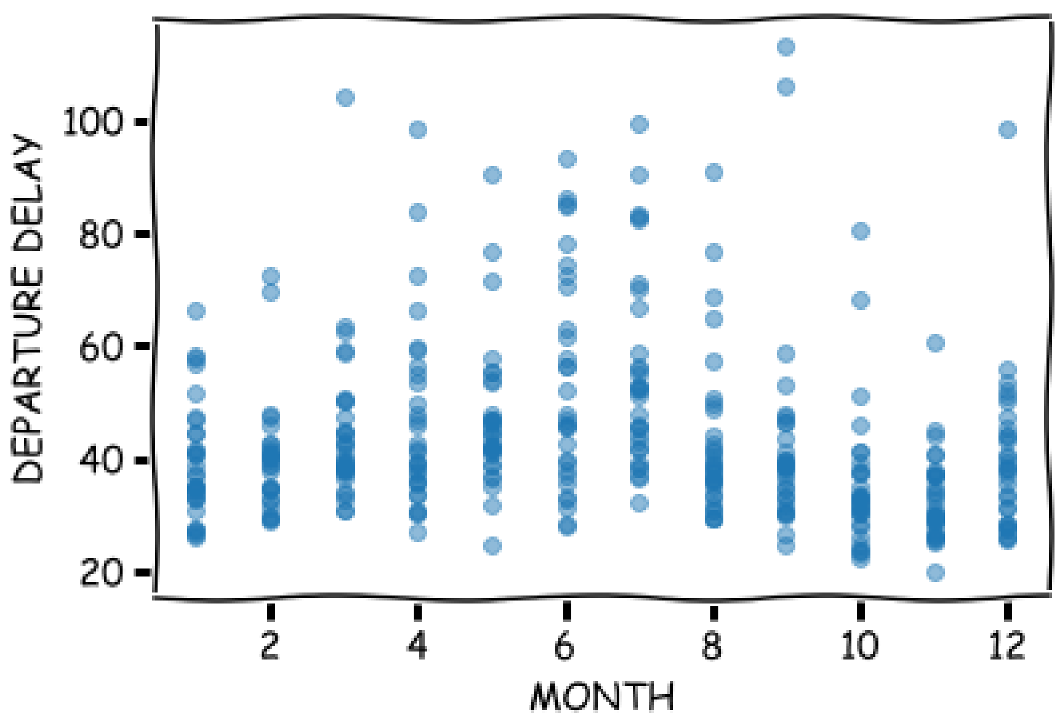
```
In [13]: print(split_date[split_date.total==max(split_date.total)])
         # this question is so confusing! But I'm assuming it means taking both arrival delays
```

```
     year  month  day   dep_delay   arr_delay       total
254  2013      9   12  105.930108  122.870968  228.801075
```

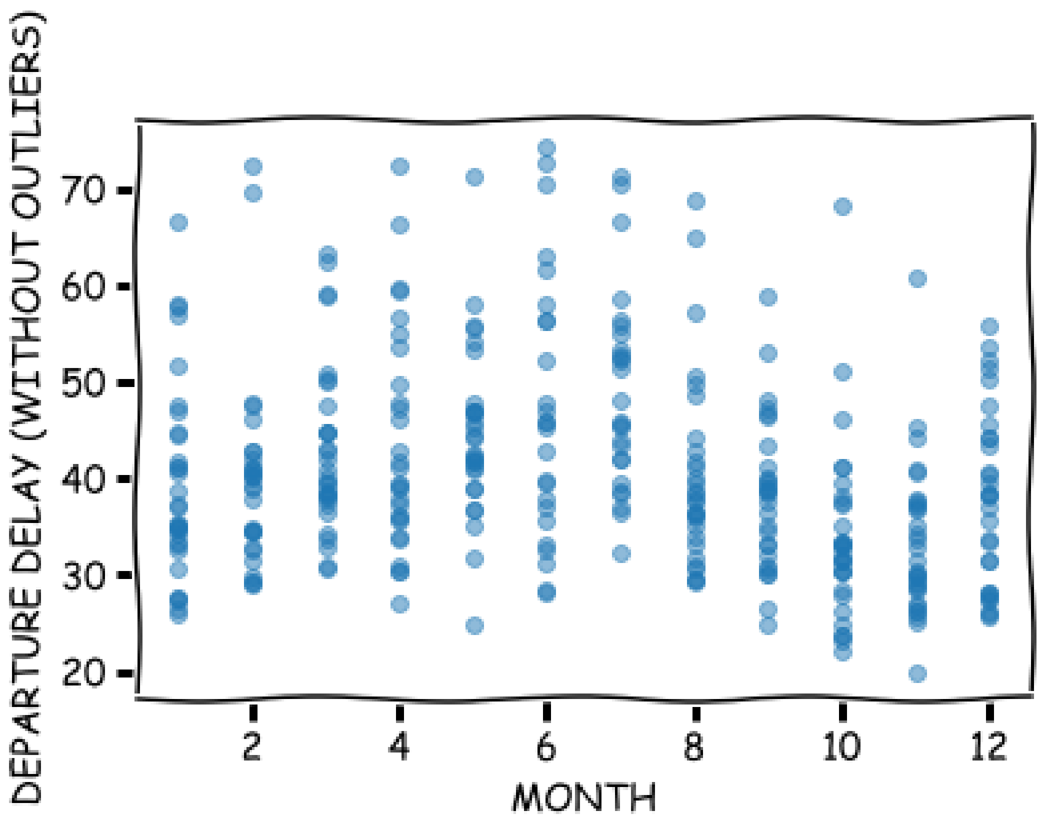
Answer 2b: Assuming that this question is looking for the day with the highest *total* average arrival and departure delays... then it's September 12th, 2013

(c) Are there any seasonal patterns in departure delays for flights from NYC?

```
In [14]: with plt.xkcd():
             plt.scatter(split_date.month, split_date.dep_delay, alpha=0.5)
             plt.xlabel("MONTH")
             plt.ylabel("DEPARTURE DELAY")
             plt.show()
```



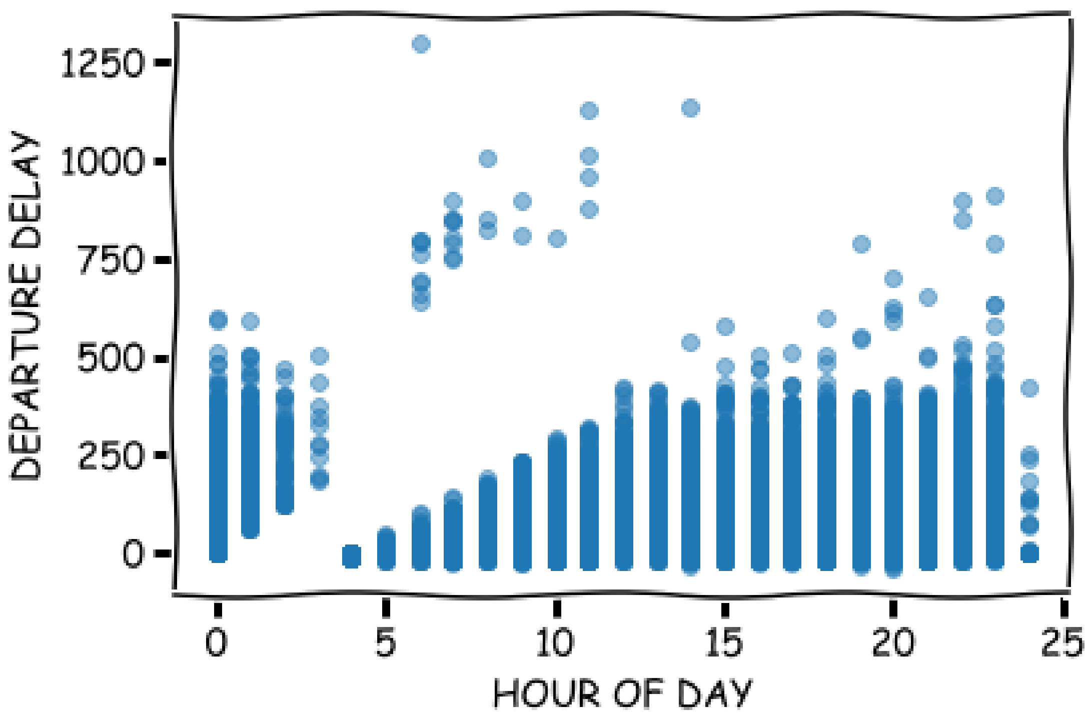
In [15]:
```
# we can look at a plot of Month by departure delay points (if we take the individual


#We can also look after we remove outliers
#removing outliers
mean = np.mean(split_date.dep_delay, axis=0)
sd = np.std(split_date.dep_delay, axis=0)

final_list = [x for x in split_date.dep_delay if (x > mean - 2 * sd)]
final_list = [x for x in final_list if (x < mean + 2 * sd)]
split_date = split_date[split_date.dep_delay.isin(final_list)]

with plt.xkcd():
    plt.scatter(split_date.month, split_date.dep_delay,alpha=0.5)
    plt.xlabel("MONTH")
    plt.ylabel("DEPARTURE DELAY (WITHOUT OUTLIERS)")
    plt.show()

#it does look slightly guassian distributed?
# seems like flights leave later in the middle of the year (Summer)
```

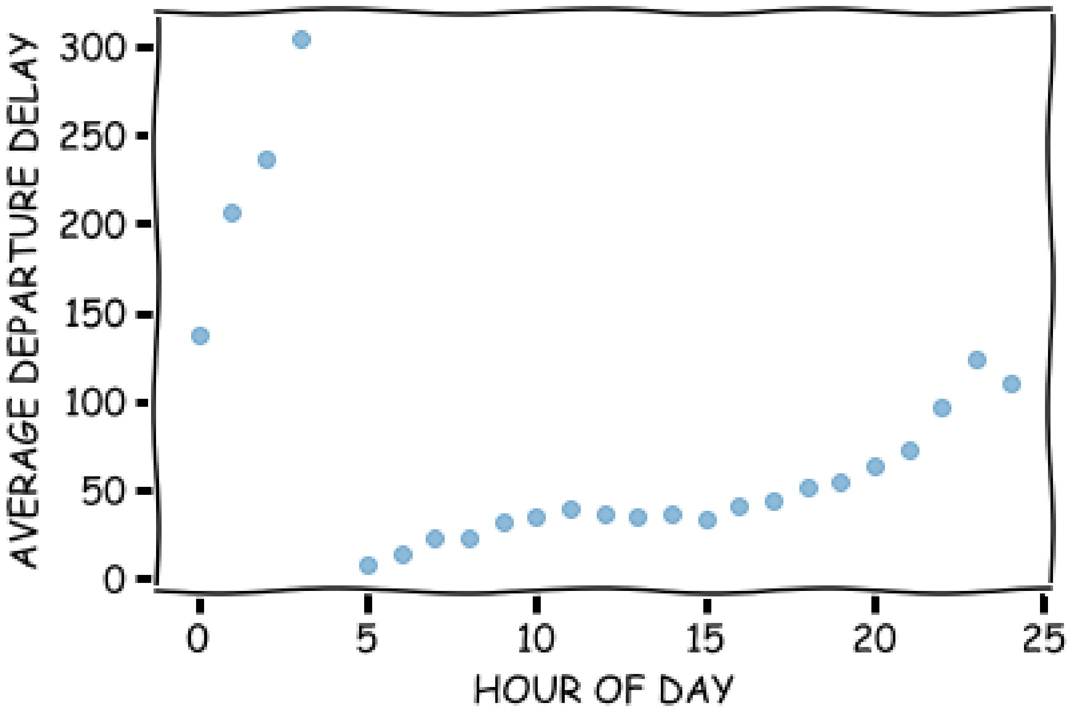


Answer 2c: Judging by visuals alone, it does look like during the Summer (May to August), the departing flights are more delayed.

(d) On average, how do departure delays vary over the course of a day?

In [16]:
```
# Now we want to look at flights again (not averaged), so that we can see all the fli
# we don't care about the months anymore, we care about the time

#here is before we do any averages

with plt.xkcd():
    plt.scatter(flights.hour, flights.dep_delay,alpha=0.5)
    plt.xlabel("HOUR OF DAY")
    plt.ylabel("DEPARTURE DELAY")
    plt.show()


    #here are the averages per hour
    by_day = pd.DataFrame(flights2.groupby(by=["hour"],as_index=False)['dep_delay','a
    plt.scatter(by_day.hour, by_day.dep_delay,alpha=0.5)
    plt.xlabel("HOUR OF DAY")
    plt.ylabel("AVERAGE DEPARTURE DELAY")
    plt.show()


#from scipy import optimize
#params, params_covar = optimize.curve_fit
```





Answer 2d: So, it looks like departures are pretty on time once morning starts (5am), and get worse over the course of the day. In the middle of the night (or rather, from midnight to 4am which is technically the early early morning), delays are at their worst!

0.6 Question 3

Which flight departing NYC in 2013 flew the fastest?

```
In [17]: #so here we care about rate, not just timing
         #let's make a column for rate and then take the max

         flights=pd.DataFrame(flights.assign(rate= lambda flights: flights.distance/flights.ai
         flights.head(10)

         flights[flights.rate==max(flights.rate)]
```

```
Out[17]:         Unnamed: 0  year  month  day  dep_time  dep_delay  arr_time  \
         216447      216448  2013      5   25    1709.0        9.0    1923.0

                 arr_delay carrier tailnum  flight origin dest  air_time  distance  \
         216447      -14.0      DL  N666DN    1499    LGA  ATL      65.0       762
```

```
         hour  minute       rate
216447   17.0     9.0  11.723077
```

Answer 3: Flight number 1499 on May 25th flew the fastest, at 11.72 miles (km?) per minute.

0.7 Question 4

Which flights (i.e. carrier + flight + dest) happen every day? Where do they fly to?

```
In [18]: #there's a few ways to do this. the first way I can think of is to get the number of
         counts = pd.DataFrame(flights.groupby(by=['carrier','flight','dest'],as_index=False).

         #{ k:v for k, v in counts.items() if v>=365 }
         counts = counts.sort_values(by=['year'],ascending=False)

         counts = counts[counts.month>=365] # big prolem here, I'm choosing month but what if

         print(counts.iloc[:, : 3])
         print('\n' + str(len(counts)) + ' flights')
```

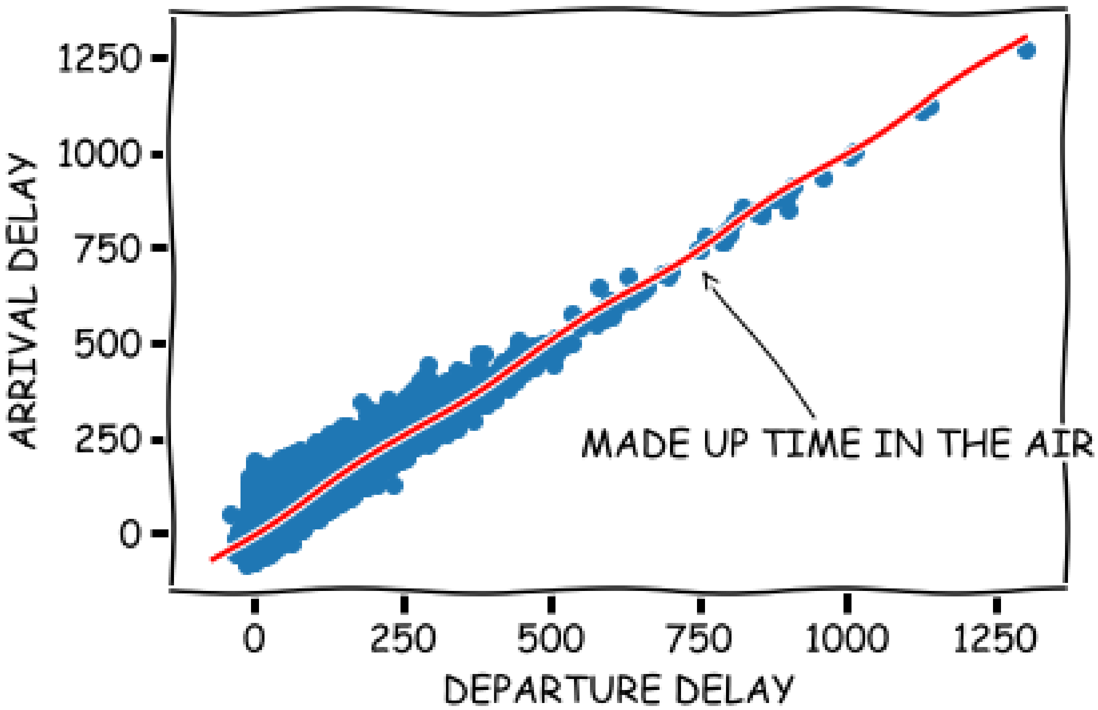
```
      carrier  flight dest
914        AA    1611  MIA
1243       B6     703  SJU
904        AA    1357  SJU
10613      VX     413  LAX
1118       B6     219  CLT
5116       UA      15  HNL
1147       B6     359  BUR
1150       B6     371  FLL
1169       B6     431  SRQ
783        AA     181  LAX
2012       DL    2159  MCO
775        AA     119  LAX
10609      VX     407  LAX
4631       EV    5712  IAD
10607      VX     251  LAS
767        AA      59  SFO
1379       B6    1783  MCO
2081       DL    2391  TPA

18 flights
```

Answer 4: My table above should show the 18 flights that happen every day.

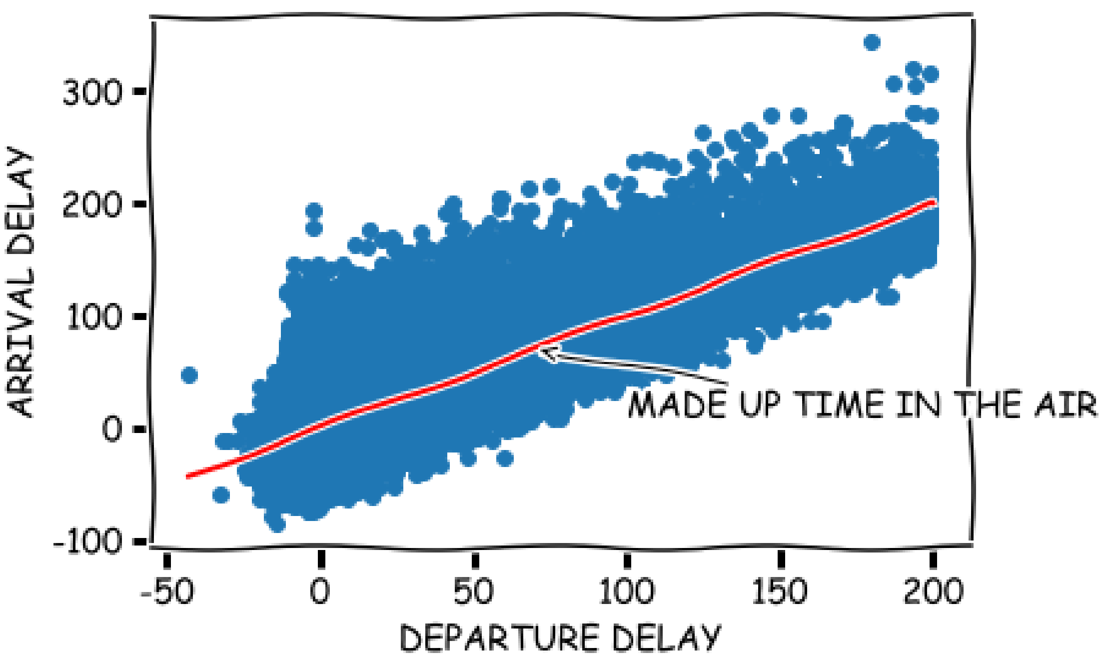
0.8 Question 5

Develop one research question you can address using the nycflights2013 dataset. Provide two visualizations to support your exploration of this question. Discuss what you find.

In [19]:
```
# I'm always curious about how flights "make up" the in-flight time when they've depa
# Somehow I tend to always arrive at the right time despite the long delay. Is that j
with plt.xkcd():
    plt.scatter(flights.dep_delay,flights.arr_delay)
    x = np.linspace(-70, max(flights.dep_delay), 1000)
    plt.plot(x, x, color='red');
    plt.xlabel("DEPARTURE DELAY")
    plt.ylabel("ARRIVAL DELAY")
    plt.annotate(
            'MADE UP TIME IN THE AIR',
            xy=(750,700), arrowprops=dict(arrowstyle='->'), xytext=(550, 200))
    plt.show()

    #just taking a closer look
    plt.scatter(flights.dep_delay[flights.dep_delay<200],flights.arr_delay[flights.dep
    x = np.linspace(min(flights.dep_delay), 200, 1000)
    plt.plot(x, x, color='red');
    plt.xlabel("DEPARTURE DELAY")
    plt.ylabel("ARRIVAL DELAY")

    plt.annotate(
            'MADE UP TIME IN THE AIR',
            xy=(70,70), arrowprops=dict(arrowstyle='->'), xytext=(100, 10))

    plt.show()
# OK so definitely correlated, I'd be shocked if it wasn't.
# But it's not exactly 1:1. What time are they "making up"?
# What we care about is below that red line.
# so we want to know the proportion of points that fall below the y=x line, and on av
```

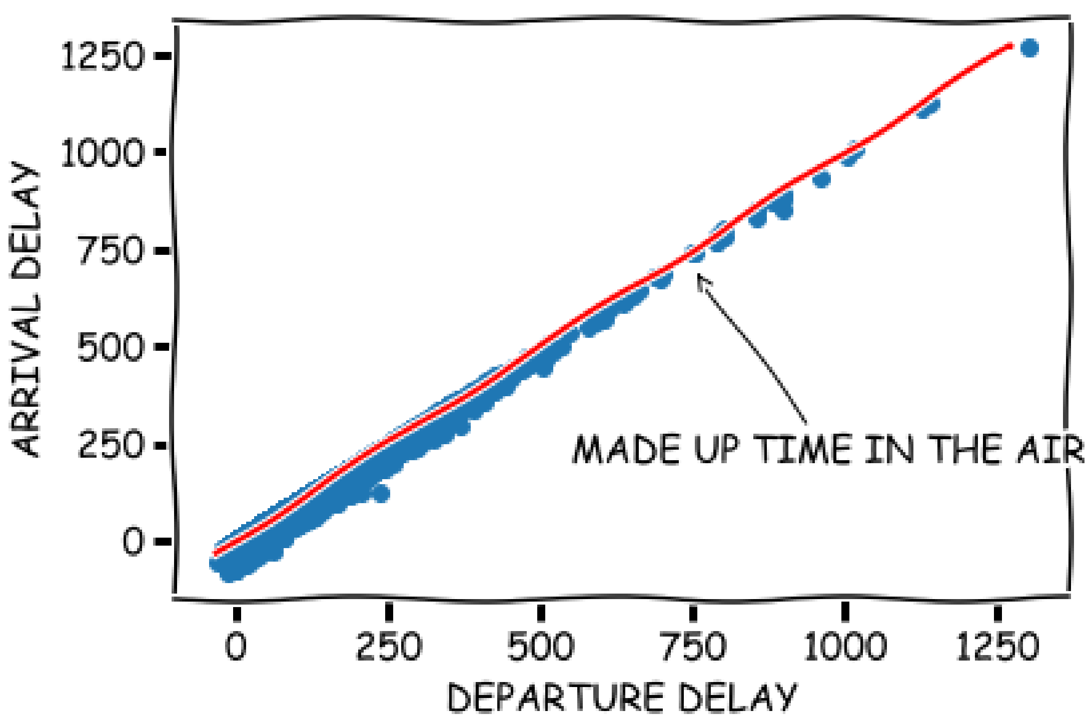


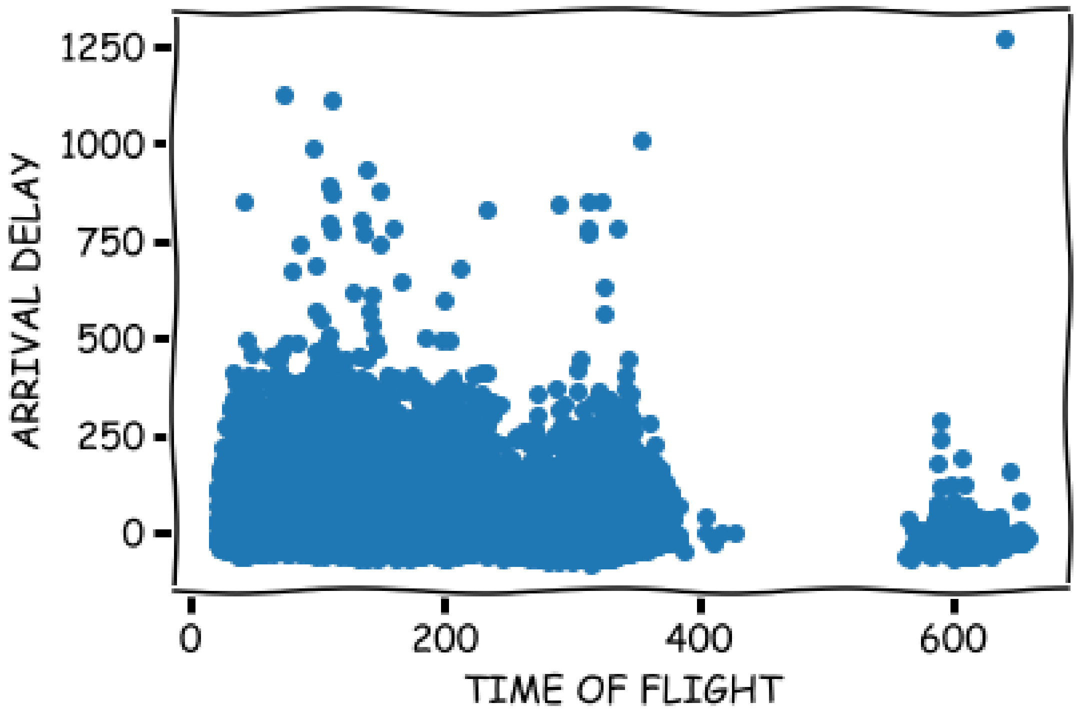
1250
1000
750
500
250
0
ARRIVAL DELAY
MADE UP TIME IN THE AIR
0
250
500
750
1000
1250
DEPARTURE DELAY



300
200
100
0
-100
ARRIVAL DELAY
MADE UP TIME IN THE AIR
-50
0
50
100
150
200
DEPARTURE DELAY

In [20]:
```
# we care about these guys, where the departing delay is less than the arrival delay,
make_ups = flights[flights.dep_delay>flights.arr_delay]
with plt.xkcd():
    plt.scatter(make_ups.dep_delay,make_ups.arr_delay)
    x = np.linspace(min(make_ups.dep_delay), max(make_ups.arr_delay), 1000)
    plt.plot(x, x, color='red');
    plt.xlabel("DEPARTURE DELAY")
    plt.ylabel("ARRIVAL DELAY")
    plt.annotate(
        'MADE UP TIME IN THE AIR',
        xy=(750,700), arrowprops=dict(arrowstyle='->'), xytext=(550, 200))
    plt.show()

    #so let's plot how the flights that "make up" air time are affected by length of
    plt.scatter(make_ups.air_time,make_ups.arr_delay)
    plt.xlabel("TIME OF FLIGHT")
    plt.ylabel("ARRIVAL DELAY")
    plt.show()
# what the actual heck is going on with this plot
# it looks ever so slightly negative. So what I predicted might be somewhat true. Lik
```






```
In [21]: earlys = len(make_ups)
         lates = len(flights[flights.dep_delay<=flights.arr_delay])

         print(earlys, lates, len(flights))
         print(earlys/len(flights))
         print(lates/len(flights))
```

```
221565 105781 336776
0.6579002066655582
0.3140989856759389
```

I've always been curious about how flights "make up the time" in the air. I'm sitting there, after waiting 40 minutes or whatever, knowing I'll be late on the other end (my destination). But sometimes, the plane can "make up" that time, and I arrive only 3 minutes late, or not late at all. I've always been confused by that, because if planes can go faster why don't they just go faster? I've especially noticed that on longer flights, they can "make up" even more of that time.

So I want to know, what proportion of delayed flights actually make up any air time? And does that change with length of the flight?

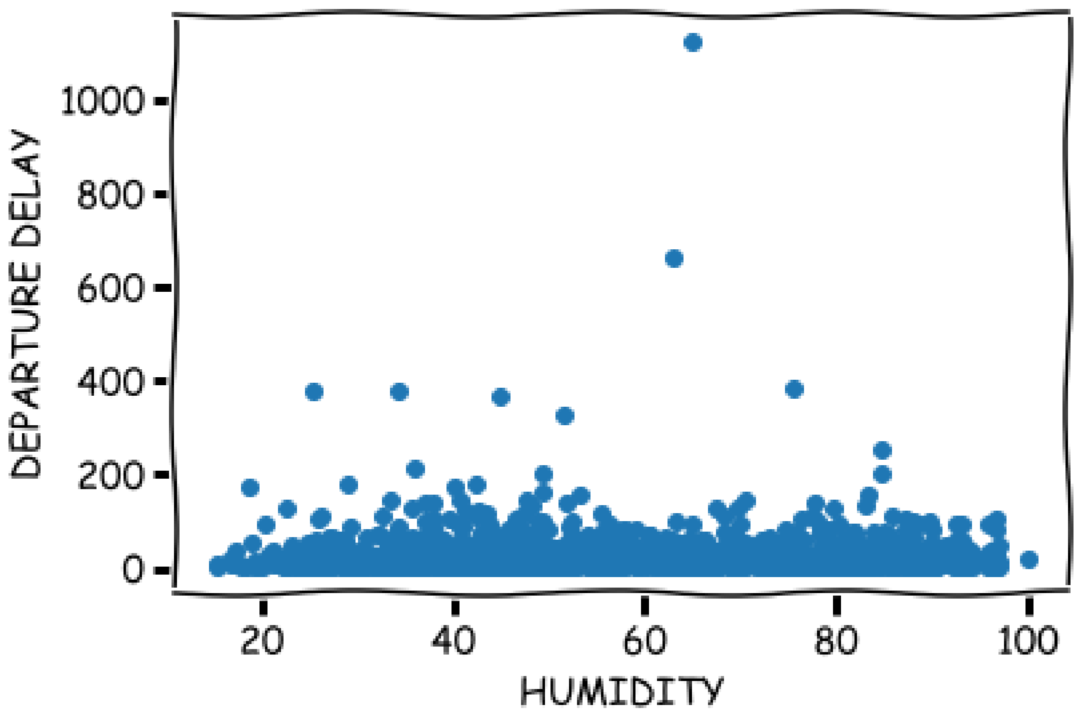
About 66% of flights actually make up some air time, meaning that the departing delay was *longer* than the arrival delay. Without doing actual parameter fitting, it's possible that there is a slight negative trend, meaning that for flights that made up time in the air, they have even shorter arrival delays if the flight was longer.

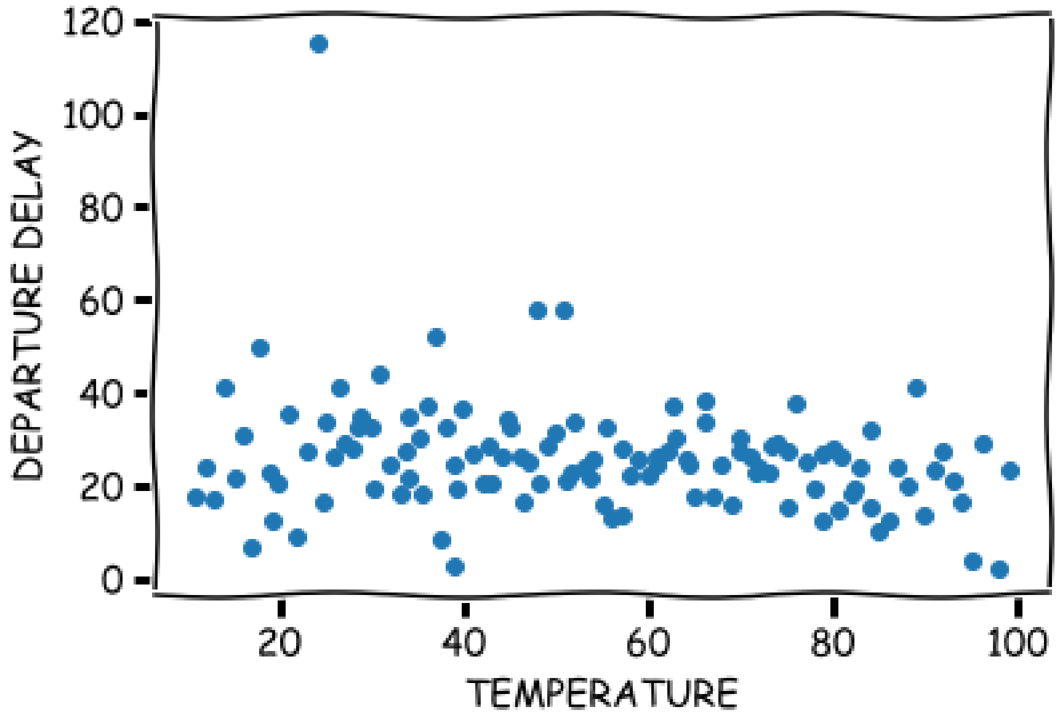
0.9 Question 6

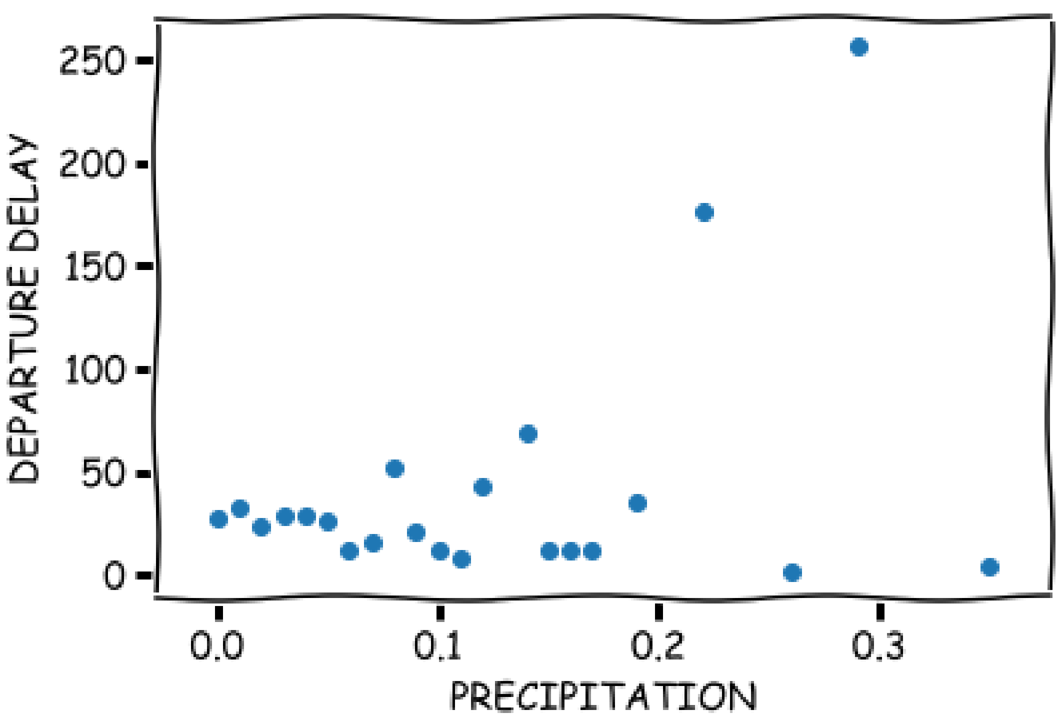
What weather conditions are associated with flight delays leaving NYC? Use graphics to explore.

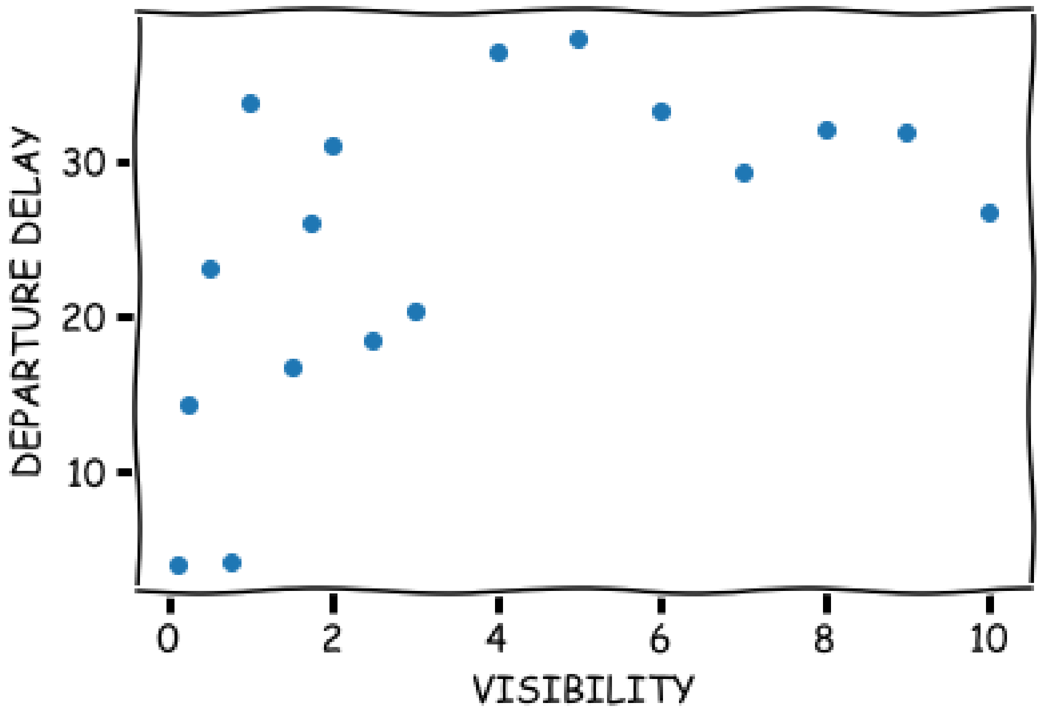
In [53]:
```
#load the weather set
weather = pd.read_csv('weather.csv')
weather.head(10)

#ok so this dataset is looking at flights out of NYC in 2013 and the weather conditio
#we have to line up the data frames for the same day!

joined = flights.join(weather, lsuffix='_month', rsuffix='_day')
joined = joined[joined.dep_delay>0] #only include real delays
humid_means = pd.DataFrame(joined.groupby(by=["humid"],as_index=False)['dep_delay'].me
vis_means = pd.DataFrame(joined.groupby(by=["visib"],as_index=False)['dep_delay'].mea
temp_means = pd.DataFrame(joined.groupby(by=["temp"],as_index=False)['dep_delay'].mea
prec_means = pd.DataFrame(joined.groupby(by=["precip"],as_index=False)['dep_delay'].me
ws_means = pd.DataFrame(joined.groupby(by=["wind_speed"],as_index=False)['dep_delay']


with plt.xkcd():
    #humidity
    plt.scatter(humid_means.humid,humid_means.dep_delay)
    plt.xlabel('HUMIDITY')
    plt.ylabel('DEPARTURE DELAY')
    plt.show()

    plt.scatter(temp_means.temp,temp_means.dep_delay)
    plt.xlabel('TEMPERATURE')
    plt.ylabel('DEPARTURE DELAY')
    plt.show()

    plt.scatter(prec_means.precip,prec_means.dep_delay)
    plt.xlabel('PRECIPITATION')
    plt.ylabel('DEPARTURE DELAY')
    plt.show()


    #visibility
    plt.scatter(vis_means.visib,vis_means.dep_delay)
    plt.xlabel('VISIBILITY')
    plt.ylabel('DEPARTURE DELAY')
    plt.show()
    # when visibility is worse, departure delays are longer!


    # weirdly, it looks like when there are extreme temperatures, the delays are less
    # we see longer delays in the mid-range temps simply because we have more of mid-
    plt.scatter(ws_means.wind_speed,ws_means.dep_delay)
```

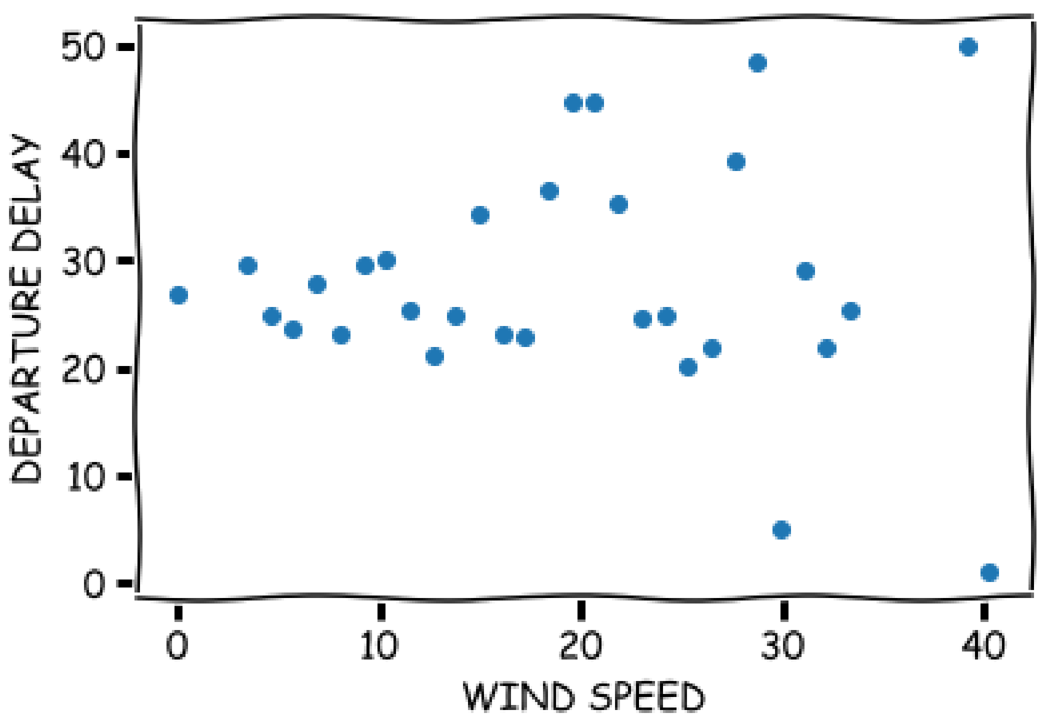
```
plt.xlabel('WIND SPEED')
plt.ylabel('DEPARTURE DELAY')
plt.show()
```




120
100
80
60
40
20
0
DEPARTURE DELAY
20
40
60
80
100
TEMPERATURE


250
200
150
100
50
0
DEPARTURE DELAY
0.0
0.1
0.2
0.3
PRECIPITATION


DEPARTURE DELAY
10
20
30
0
2
4
6
8
10
VISIBILITY


DEPARTURE DELAY
0
10
20
30
40
50
0
10
20
30
40
WIND SPEED

Answer 6:

We can intuitively describe situations where weather would affect the departure delay of our flight. I know I've flown out of places experiencing snow and they're totally unprepared for ice! You can also imagine that something as important as "visibility" would affect whether you're going to safely fly the plane!

So here I've shown a few plots (their averages! as to not overwhelm our eyes with too many overlapping data points). We can see that:

- except for a few outliers, humidity seems to not have a clear trend. I also think there are more "outlier" points in the mid-humidity range because we simply have more mid-humidity days, as opposed to extreme humidity values. But it is interesting that the delays are happening when it's a mid-humidity day. But I'm guessing it's due to something else that I'm not accounting for.
- looking on average there's not really an effect of temperature either
- there is a clear effect of precipitation amount. More precipitation = longer departure delay
- clear effect of visibility as well, but with a plateau at around visibility 4. Anything worse than that seems to be the same amount of delayed. But certainly less delay before that threshold.
- and finally, there seems to be a slight positive trend for how wind speed affects delays. Faster wind = longer delay (most of the time)