FLIGHTS HISTORICAL DATA ANALYSIS

Cherniakhovsky Yuriy: Python 3, pandas, numpy programming

AUGUST 6, 2017

TABLE OF CONTENTS

INTRODUCTORY

The target of below mentioned report is to create a model that will give a possibility to charge more money for that flights' tickets which have the higher probability of arrival on time.

The second task is to depict other interesting insights that should be pointed to.

Analysis was done over a table (31 columns and 43978 rows, filename is "allyears2k.csv") which holds historical data of flights from 1987 until 2008 (not full but fragmentary). Unfortunately, the table has only two months in every year (the 1st and the 10th) metrifying the reporting depth.

Flights arriving on time are that one's which have value "NO" in the next-to-last column, or that which have values less than or equal to zero in columns "ArrDelay".

1.1. Carriers

If we analyze the whole file and will not take a look at special year, we can see that from 1987 until 2008 top three Carriers that arrive on time are: "CO" with 58.65%, "UA" with 50.45% and "AA" with 47.79% flights on time (code and output are below).

Input:

```
import csv
import pandas


FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["IsArrDelayed"]=="NO"),
["Year", "UniqueCarrier", "FlightNum","TailNum", "Origin", "Dest"]]

uniqueCarrierFlights = dict(all_flights['UniqueCarrier'].value_counts())
uniqueCarrierFlightsOnTime =
dict(all_flights_no_delays_short['UniqueCarrier'].value_counts())

'''For receiving frequency in %'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    for i in denom:
        #print (i,' ',int(nom[i]/denom[i]*100),'%')
        d[i]=round(float(nom[i]/denom[i]*100),2)
    #We are sorting dictionary according key
    #for j in sorted(d.values()):
    #   print (j)
    return d
mydict=frequency(uniqueCarrierFlightsOnTime,uniqueCarrierFlights)

'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))

print (sortByValue(mydict))
```

Output:

```
[(58.65, 'CO'), (50.45, 'UA'), (47.79, 'AA'), (46.38, 'US'), (42.71, 'WN'), (39.55,
'HP'), (37.33, 'DL'), (31.42, 'PI'), (29.02, 'PS'), (28.88, 'TW')]
```

1.2. Carrier. Arrival delay less than 10 minutes

Of course arrival delay is bad situation that sometimes leads to problems and compensations but if arrival delay is not longtime, for example 10 minutes it is not the big problem. In this case the top three are little bit changed. “CO” Carrier has 70.67% of arrival on time, “UA” has 68.08% and “US” – 67.1%.

Input:

```
import csv
import pandas


FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_little_delay = all_flights.loc[(all_flights["ArrDelay"]<10), ["Year",
"UniqueCarrier", "FlightNum","ArrDelay", "DepDelay"]]

uniqueCarrierFlights = dict(all_flights['UniqueCarrier'].value_counts())
uniqueCarrierFlightsDelayMax10 =
dict(all_flights_little_delay['UniqueCarrier'].value_counts())


##print (dict(all_flights['UniqueCarrier'].value_counts()))
##print (len(dict(all_flights['UniqueCarrier'].value_counts())))
##print()
##print (dict(all_flights_little_delay['UniqueCarrier'].value_counts()))
##print (len(dict(all_flights_little_delay['UniqueCarrier'].value_counts())))
##print()

'''For receiving frequency in % of arrival without delay'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    for i in denom:
        #print (i,' ',int(nom[i]/denom[i]*100),'%')
        d[i]=round(float(nom[i]/denom[i]*100),2)
    return d
mydict=frequency(uniqueCarrierFlightsDelayMax30,uniqueCarrierFlights)

'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))

print (sortByValue(mydict))
```

Output:

```
[(70.67, 'CO'), (68.08, 'UA'), (67.1, 'US'), (65.88, 'AA'), (64.51, 'PS'), (63.6,
'HP'), (62.3, 'WN'), (59.16, 'PI'), (51.98, 'DL'), (48.33, 'TW')]
```

1.3. Carrier. Breaking into years.

The Presence of Carriers every year starting from 1987 reflected below. As we see, because of incompleteness of inflow data for analysis, every year we don't see the presence of every carrier. Construction any dependency diagram of Carriers arrival on time probabilities build on years would be not reasonable.

Input:

```
import csv
import pandas
from pandas import read_csv

FILENAME = "allyears2k.csv"


all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["IsArrDelayed"]=="NO") &
(all_flights["IsDepDelayed"]=="NO"), ["Year", "UniqueCarrier",
"FlightNum","TailNum", "Origin", "Dest"]]

table_all_flights_years=[]
all_years=list(all_flights['Year'].unique())
for i in all_years:
    df=all_flights.loc[(all_flights["Year"]==i), ["Year", "UniqueCarrier",
"FlightNum","TailNum", "Origin", "Dest"]]
    print (i, dict(df['UniqueCarrier'].value_counts()))
```

Output:

```
1987 {'PS': 1999}
1988 {'PS': 1213, 'PI': 786}
1989 {'UA': 1999}
1990 {'US': 1999}
1991 {'US': 1999}
1992 {'US': 1999}
1993 {'US': 1999}
1994 {'US': 1999}
1995 {'US': 1607, 'TW': 155, 'UA': 144, 'AA': 93}
1996 {'HP': 700, 'WN': 360, 'AA': 308, 'UA': 285, 'DL': 253, 'US': 62, 'CO': 31}
1997 {'HP': 821, 'WN': 545, 'DL': 310, 'UA': 260, 'CO': 32, 'US': 31}
1998 {'HP': 750, 'UA': 398, 'WN': 375, 'AA': 259, 'DL': 124, 'CO': 62, 'US': 31}
1999 {'US': 975, 'WN': 389, 'HP': 181, 'TW': 174, 'DL': 124, 'UA': 92, 'AA': 64}
2000 {'HP': 999, 'WN': 503, 'UA': 259, 'DL': 124, 'CO': 83, 'US': 31}
2001 {'US': 1999}
2002 {'US': 1999}
2003 {'UA': 1999}
2004 {'UA': 1999}
2005 {'UA': 1999}
2006 {'US': 1999}
2007 {'WN': 1999}
2008 {'WN': 1999}
```

2. Flight number

Furthermore, we can analyze the frequency of flights without arrival delays according the Flight Number. As we can see below Flight Numbers 1082, 1078 and 1066 have the highest probabilities of coming on time. Their probability frequencies are 96.97%, 91.89%, 90.2% respectively and tickets for this Flight Number should have more price.

Input:

```
import csv
import pandas

FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["IsArrDelayed"]=="NO"),
["Year", "UniqueCarrier", "FlightNum","TailNum", "Origin", "Dest"]]

flightNum = dict(all_flights['FlightNum'].value_counts())
flightNumOnTime = dict(all_flights_no_delays_short['FlightNum'].value_counts())

'''For taking in account only flights that are more than 10'''
for i in flightNumOnTime:
    if flightNumOnTime[i]<=10:flightNumOnTime[i]=0

'''For receiving frequency in % of arrival without delay'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    for i in denom:
        try:
            #print (i,' ',int(nom[i]/denom[i]*100),'%')
            d[i]=round(float(nom[i]/denom[i]*100),2)
        except KeyError:
            d[i]=0.00
    return d

mydict=frequency(flightNumOnTime,flightNum)


'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))

#print (sortByValue(mydict))


'''For showing data that is higher than 50%'''
count=0
for i in sortByValue(mydict):
    count+=1
    if i[0]<=50:break

print (sortByValue(mydict)[:count])
```

Output:

```
[(96.97, 1082), (91.89, 1078), (90.2, 1066), (88.0, 1036), (85.29, 197), (85.19,
469), (84.62, 1068), (84.38, 1090), (84.21, 1044), (83.87, 1088), (83.33, 2472),
(82.76, 1026), (82.35, 596), (81.82, 1043), (81.25, 143), (80.77, 1054), (80.0,
63), (79.41, 740), (78.89, 84), (78.57, 576), (78.12, 749), (78.0, 1060), (77.42,
202), (77.08, 1440), (76.92, 2456), (76.67, 786), (76.47, 1930), (75.86, 1635),
(75.81, 2753), (75.76, 1020), (75.0, 2546), (74.55, 1062), (74.19, 1051), (73.81,
1074), (73.77, 478), (73.08, 604), (72.73, 1049), (72.58, 756), (72.0, 424),
(71.88, 602), (71.67, 67), (71.43, 603), (71.21, 205), (71.05, 472), (70.97, 2471),
(70.77, 199), (70.37, 728), (70.27, 548), (70.0, 97), (69.89, 81), (69.84, 744),
(69.7, 802), (69.49, 499), (69.44, 1064), (68.75, 2640), (68.57, 868), (68.33,
796), (68.18, 101), (68.09, 60), (68.0, 1424), (67.86, 387), (67.69, 1061), (67.65,
2489), (66.67, 375), (66.13, 1339), (66.1, 138), (65.71, 768), (65.62, 2733),
(65.52, 1058), (65.45, 1076), (65.08, 2864), (65.0, 462), (64.86, 426), (64.71,
1410), (64.52, 1175), (64.29, 1561), (64.06, 505), (64.0, 597), (63.64, 1079),
(63.49, 102), (63.33, 1704), (63.16, 1412), (63.08, 249), (62.71, 510), (62.5,
109), (62.26, 1071), (62.07, 1432), (62.0, 1052), (61.9, 1467), (61.76, 397),
(61.73, 65), (61.54, 2788), (61.48, 441), (61.4, 758), (61.29, 2049), (61.19, 378),
(61.11, 2602), (60.61, 2020), (60.47, 1080), (60.32, 399), (60.23, 752), (60.0,
1420), (59.82, 85), (59.78, 403), (59.65, 244), (59.63, 1534), (59.62, 1081),
(59.38, 2261), (59.26, 724), (58.97, 187), (58.82, 252), (58.73, 220), (58.62,
1605), (58.49, 246), (58.33, 770), (58.21, 466), (58.06, 203), (57.95, 1063),
(57.81, 193), (57.69, 613), (57.63, 148), (57.58, 496), (57.5, 574), (57.45, 128),
(57.35, 137), (57.33, 440), (57.26, 755), (57.14, 2601), (56.92, 379), (56.7,
1488), (56.67, 512), (56.6, 761), (56.58, 147), (56.52, 1446), (56.45, 2761),
(56.25, 782), (56.0, 2404), (55.88, 713), (55.56, 215), (55.38, 145), (55.34, 68),
(55.29, 960), (55.21, 116), (55.17, 1417), (55.1, 639), (55.0, 1073), (54.84,
1405), (54.76, 204), (54.69, 111), (54.55, 1231), (54.42, 743), (54.41, 775),
(54.29, 590), (54.17, 558), (54.1, 552), (54.05, 773), (53.97, 497), (53.85, 753),
(53.68, 233), (53.66, 77), (53.57, 1267), (53.33, 1609), (53.27, 79), (53.23,
1028), (53.12, 2794), (53.03, 750), (52.87, 226), (52.78, 690), (52.63, 504),
(52.5, 70), (52.38, 136), (52.34, 419), (52.24, 62), (52.17, 1436), (52.05, 506),
(51.92, 1389), (51.85, 2286), (51.72, 731), (51.61, 2751), (51.56, 475), (51.52,
434), (51.35, 793), (51.28, 149), (51.22, 385), (51.11, 74), (51.02, 1465), (50.98,
133), (50.88, 735), (50.82, 1027), (50.79, 2755), (50.59, 56), (50.43, 227), (50.0,
267)]
```

3. Origin

The output below shows that flights coming from such origin points like ‘SAV’, ‘BOI’ and ‘TRI’ have the highest probabilities of arrival on time. Their values are 83.87%, 82.02% and 77.27% respectively.

Input:

```
import csv
import pandas

FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["IsArrDelayed"]=="NO"),
["Year", "UniqueCarrier", "FlightNum","TailNum", "Origin", "Dest"]]

origin = dict(all_flights['Origin'].value_counts())
originOnTime = dict(all_flights_no_delays_short['Origin'].value_counts())

'''For taking in account only flights that are more than 10'''
for i in originOnTime:
    if originOnTime[i]<=10:originOnTime[i]=0

'''For receiving frequency in % of arrival without delay'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    for i in denom:
        try:
            #print (i,' ',int(nom[i]/denom[i]*100),'%')
            d[i]=round(float(nom[i]/denom[i]*100),2)
        except KeyError:
            d[i]=0.00
    return d

mydict=frequency(originOnTime,origin)

'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))

#print (sortByValue(mydict))

'''For showing data that is higher than 50%'''
count=0
for i in sortByValue(mydict):
    count+=1
    if i[0]<=50:break

print (sortByValue(mydict)[:count])
```

Output:

```
[(83.87, 'SAV'), (82.02, 'BOI'), (77.27, 'TRI'), (76.67, 'MLB'), (71.43, 'MKE'),
(70.97, 'ACY'), (70.83, 'OKC'), (70.0, 'PHF'), (67.83, 'HOU'), (66.67, 'TLH'),
```

```
(65.22, 'GEG'), (65.15, 'IAH'), (64.71, 'TUL'), (64.29, 'MEM'), (63.39, 'BHM'),
(63.33, 'LIH'), (62.61, 'JAX'), (62.42, 'PWM'), (61.54, 'CAE'), (61.28, 'MSY'),
(60.39, 'AVP'), (58.54, 'RIC'), (58.5, 'IAD'), (58.27, 'DAL'), (57.69, 'LBB'),
(57.24, 'RDU'), (56.0, 'MAF'), (55.62, 'GSO'), (55.29, 'CLE'), (55.14, 'ORF'),
(54.64, 'ISP'), (54.58, 'CLT'), (52.5, 'ELP'), (51.96, 'DTW'), (51.72, 'CHS'),
(51.38, 'DEN'), (50.94, 'SLC'), (50.89, 'BNA'), (50.35, 'MCI'), (50.0, 'LIT')]
```

4. Destination

Let's take into account the destination of flights as well. The result of below code shows us that destinations like: 'ICT', 'COS' and 'SBN' have the highest probabilities of arriving on time. Their arrival on time frequencies are 88.0%, 82.76% and 70.97% respectively.

Input:

```
import csv
import pandas

FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["IsArrDelayed"]=="NO"),
["Year", "UniqueCarrier", "FlightNum","TailNum", "Origin", "Dest"]]

destination = dict(all_flights['Dest'].value_counts())
destinationOnTime = dict(all_flights_no_delays_short['Dest'].value_counts())

##print (dict(all_flights_no_delays_short['Dest'].value_counts()))
##print (len(dict(all_flights_no_delays_short['Dest'].value_counts())))

'''For taking in account only flights that are more than 10'''
for i in destinationOnTime:
    if destinationOnTime[i]<=10:destinationOnTime[i]=0

'''For receiving frequency in % of arrival without delay'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    for i in denom:
        try:
            #print (i,' ',int(nom[i]/denom[i]*100),'%')
            d[i]=round(float(nom[i]/denom[i]*100),2)
        except KeyError:
            d[i]=0.00
    return d

mydict=frequency(destinationOnTime,destination)

'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))

#print (sortByValue(mydict))

'''For showing data that is higher than 50%'''
count=0
for i in sortByValue(mydict):
    count+=1
    if i[0]<=50:break

print (sortByValue(mydict)[:count])
```

Output:

```
[(88.0, 'ICT'), (82.76, 'COS'), (70.97, 'SBN'), (70.24, 'FAY'), (69.49, 'MDT'),
(68.75, 'LIT'), (67.24, 'OKC'), (66.36, 'HNL'), (65.52, 'CAK'), (64.52, 'KOA'),
(64.1, 'TUL'), (62.5, 'GRR'), (61.29, 'ILM'), (60.84, 'MIA'), (60.14, 'JAX'),
(60.09, 'DFW'), (58.72, 'OMA'), (58.65, 'IAD'), (57.72, 'CLE'), (57.69, 'GSP'),
(57.5, 'RIC'), (57.14, 'ACY'), (56.84, 'BTV'), (56.67, 'MKE'), (55.37, 'DEN'),
(55.06, 'CLT'), (54.84, 'TOL'), (54.39, 'MHT'), (54.24, 'RSW'), (52.94, 'PWM'),
(51.26, 'BOS'), (50.9, 'BDL'), (50.84, 'BNA'), (50.0, 'AVP')]
```

5. Distance (regression Model)

Trying to find out is there any dependencies between flights without arrival delays and the distance, I have constructed regression (below). Slight slope of regression line says that the more distance is the less probability to be without arrival delays. But we have nothing to do with that because the slope is too slight.

The mean of probability axis of flights without arrival delay is on the point of 45,77% and the mean of the distance axis is on the point of 754.76 miles.

Coefficient of variation of arrivals on time and the distance is not homogeneous (data dispersion is high) because their values are more than 0.33. Their values are 0.38 and 0.79 respectively.

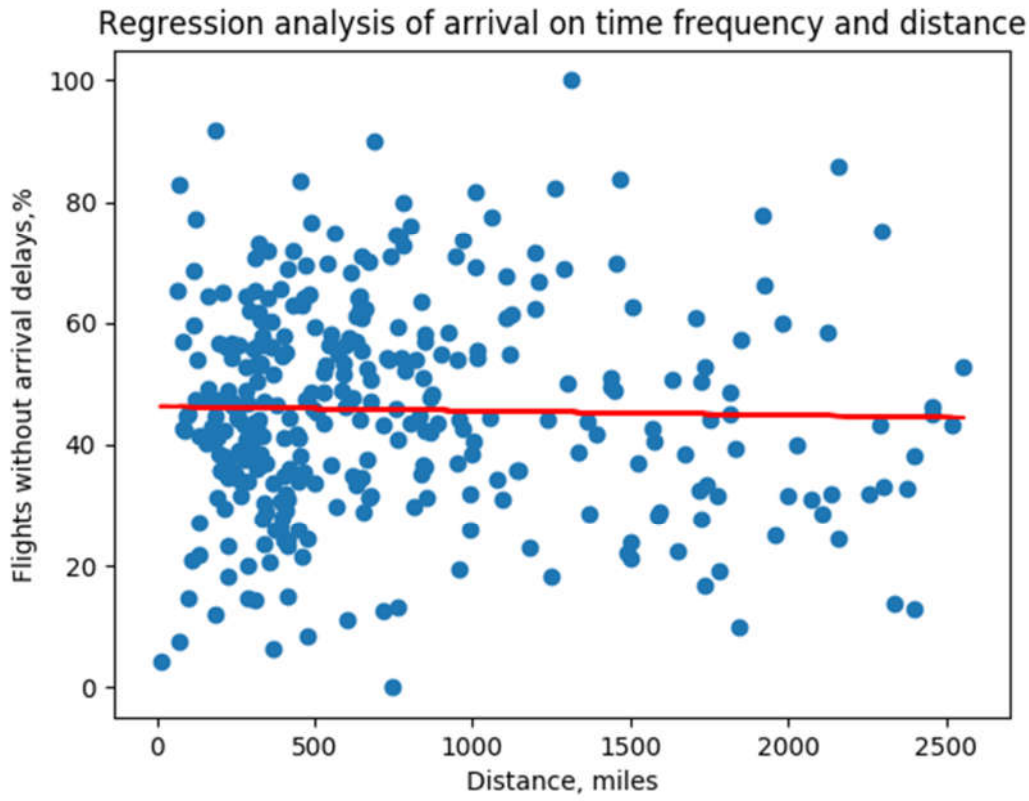
Coefficient of correlation between flights without arrival delays and the distance is minus 0,03 confirming that there is no correlation. So there is no dependency between flights without arrival delays and the distance. And there will be no effect in trying to predict whether flights will be on time proceeding from the distance.

Input:

```
import csv
import pandas
import matplotlib.pyplot as plt
import numpy as np

FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["IsArrDelayed"]=="NO"),
["Year", "UniqueCarrier", "FlightNum","TailNum", "Origin", "Dest", "Distance"]]

distance = dict(all_flights['Distance'].value_counts())

distanceOnTime = dict(all_flights_no_delays_short['Distance'].value_counts())


'''For receiving frequency in %'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    for i in denom:
        try:
            #print (i,' ',int(nom[i]/denom[i]*100),'%')
            d[i]=round(float(nom[i]/denom[i]*100),2)
        except KeyError:
            d[i]=0.00
    return d

mydict=frequency(distanceOnTime,distance)

'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))


x=[]
for i in sortByValue(mydict):
```

```
    x.append(i[1])

y=[]
for i in sortByValue(mydict):
    y.append(i[0])

x=np.array(x)
y=np.array(y)


fig, ax = plt.subplots()
fit = np.polyfit(x, y, deg=1)
ax.plot(x, fit[0] * x + fit[1], color='red')
ax.scatter(x, y)

plt.xlabel('Distance, miles')
plt.ylabel('Flights without arrival delays,%')
plt.title('Regression analysis of arrival on time frequency and distance')
fig.show()


'''Some statistical key figures'''
meanX=np.mean(x)
meanY=np.mean(y)
stdX=np.std(x)
stdY=np.std(y)
coefVarX=stdX/meanX
coefVarY=stdY/meanY

'''Pearson product-moment correlation coefficient'''
def pearson(x,y):
    n=len(x)
    vals=range(n)
    # Простые суммы
    sumx=sum([float(x[i]) for i in vals])
    sumy=sum([float(y[i]) for i in vals])
    # Суммы квадратов
    sumxSq=sum([x[i]**2.0 for i in vals])
    sumySq=sum([y[i]**2.0 for i in vals])
    # Сумма произведений
    pSum=sum([x[i]*y[i] for i in vals])
    # Вычисляем коэффициент корреляции Пирсона
    num=pSum-(sumx*sumy/n)
    den=((sumxSq-pow(sumx,2)/n)*(sumySq-pow(sumy,2)/n))**.5
    if den==0: return 0
    r=num/den
    return r

print ('***Below are some statistical key figures:***')
print ('Mean of X axis = ',meanX)
print ('Mean of Y axis = ',meanY)
print ('Standart Deviation of X axis = ',stdX)
print ('Standart Deviation of Y axis = ',stdY)
print ('Coefficient of Variation of X axis = ',coefVarX)
print ('Coefficient of Variation of Y axis = ',coefVarY)
print ('Coefficient of Pearson correlation = ', pearson(x,y))
```

Output:




```
***Below are some statistical key figures:***
Mean of X axis =  754.761904762
Mean of Y axis =  45.7711904762
Standart Deviation of X axis =  599.530528018
Standart Deviation of Y axis =  17.2200752103
Coefficient of Variation of X axis =  0.794330668036
Coefficient of Variation of Y axis =  0.376220828674
Coefficient of Pearson correlation =  -0.0252683696213
```

6. Day of Week

If we go through the best day of week for arrival on time, so according output below this would be Saturday (36.68%). But this value is any way too low.

Input:

```
import csv
import pandas
import matplotlib.pyplot as plt
import numpy as np

FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["IsArrDelayed"]=="NO") &
(all_flights["IsDepDelayed"]=="NO"), ["Year", "UniqueCarrier",
"FlightNum","DayOfWeek"]]

dayOfWeekFlight = dict(all_flights['DayOfWeek'].value_counts())

dayOfWeekFlightOnTime =
dict(all_flights_no_delays_short['DayOfWeek'].value_counts())


'''For receiving frequency in %'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    try:
        for i in denom:
            #print (i,' ',int(nom[i]/denom[i]*100),'%')
            d[i]=round(float(nom[i]/denom[i]*100),2)
    except KeyError:
        d[i]=0.00
    return d

mydict=frequency(dayOfWeekFlightOnTime,dayOfWeekFlight)

'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))

print (sortByValue(mydict))
```

Output:

```
[(36.68, 6), (36.65, 3), (33.28, 7), (33.15, 2), (32.59, 1), (29.07, 5), (26.85,
4)]
```

7. Tail Number

Tail Number is a nice instrument to predict whether the flight will arrive on time. The output below shows in about 100 flights which probabilities of arrival on time are more than 50%. The leader is airplane with tail number N384UA with 92.31% probability of arrival on time.

Input:

```
import csv
import pandas
import matplotlib.pyplot as plt
import numpy as np

FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["IsArrDelayed"]=="NO"),
["Year", "UniqueCarrier", "FlightNum","TailNum"]]

TailNum = dict(all_flights['TailNum'].value_counts())

TailNumOnTime = dict(all_flights_no_delays_short['TailNum'].value_counts())

'''For taking in account only flights that are more than 10'''
for i in TailNumOnTime:
    if TailNumOnTime[i]<=10:TailNumOnTime[i]=0


'''For receiving frequency in %'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    for i in denom:
        try:
            #print (i,' ',int(nom[i]/denom[i]*100),'%')
            d[i]=round(float(nom[i]/denom[i]*100),2)
        except KeyError:
            d[i]=0.00
    return d

mydict=frequency(TailNumOnTime,TailNum)


'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))

'''For showing data that is higher than 50%'''
count=0
for i in sortByValue(mydict):
    count+=1
    if i[0]<=50:break

print (sortByValue(mydict)[:count])
```

Output:

```
[(92.31, 'N384UA'), (88.24, 'N859'), (88.0, 'N833UA'), (86.67, 'N825UA'), (84.62,
'N823UA'), (83.33, 'N856'), (81.25, 'N530A'), (80.0, 'N422'), (79.17, 'N825'),
(78.95, 'N904UA'), (78.57, 'N251AU'), (77.78, 'N767'), (76.47, 'N342UA'), (76.19,
'N862'), (75.86, 'N852UA'), (75.51, 'N923UA'), (75.0, 'N426UA'), (74.07, 'N822UA'),
(73.91, 'N855'), (73.33, 'N563UA'), (72.73, 'N429UA'), (72.22, 'N247US'), (71.43,
'N854'), (70.83, 'N335AW'), (70.59, 'N531AU'), (70.45, 'N927UA'), (70.0, 'N720'),
(69.57, 'N850'), (69.23, 'N819UA'), (69.05, 'N338'), (68.97, 'N327UA'), (68.75,
'N254AU'), (68.42, 'N806UA'), (68.18, 'N743'), (68.09, 'N315AW'), (68.0, 'N529A'),
(67.5, 'N930UA'), (66.67, 'N396UA'), (65.79, 'N934UA'), (65.22, 'N830UA'), (65.0,
'N817UA'), (64.86, 'N918UA'), (64.71, 'N365'), (64.41, 'N912UA'), (64.29,
'N392UA'), (64.0, 'N894'), (63.64, 'N577US'), (63.16, 'N821UA'), (62.5, 'N942UA'),
(62.26, 'N913UA'), (62.07, 'N337'), (61.9, 'N576'), (61.54, 'N853UA'), (61.29,
'N884'), (61.11, 'N542UA'), (60.98, 'N933UA'), (60.87, 'N836UA'), (60.71, 'N890'),
(60.61, 'N926UA'), (60.0, 'N526UA'), (59.26, 'N372UA'), (59.09, 'N397UA'), (58.97,
'N306AW'), (58.82, 'N167AW'), (58.7, 'N335'), (58.33, 'N723'), (58.06, 'N910UA'),
(57.89, 'N508'), (57.69, 'N725'), (57.5, 'N166AW'), (57.45, 'N932UA'), (57.14,
'N732'), (56.82, 'N305AW'), (56.67, 'N334'), (56.41, 'N303AW'), (56.0, 'N920UA'),
(55.56, 'N840UA'), (55.32, 'N398UA'), (55.17, 'N389UA'), (55.1, 'N311AW'), (55.0,
'N558A'), (54.84, 'N354'), (54.76, 'N169AW'), (54.55, 'N837UA'), (54.17, 'N507'),
(53.85, 'N304UA'), (53.7, 'N922UA'), (53.66, 'N325AW'), (53.57, 'N366UA'), (53.49,
'N158AW'), (53.33, 'N328AW'), (52.94, 'N387'), (52.78, 'N917UA'), (52.5, 'N370UA'),
(52.38, 'N936UA'), (52.17, 'N847UA'), (51.52, 'N323AW'), (51.28, 'N624AW'), (50.0,
'N803UA')]
```

8. Year

From below output we see that most successful year for arrivals on time was 2003 year with probability 68.93%. And the worst year was 1987 with probability of arrival on time 14.96%.

Input:

```
import csv
import pandas
import matplotlib.pyplot as plt
import numpy as np

FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)
all_flights_no_delays_short = all_flights.loc[(all_flights["ArrDelay"]<=0),
["Year", "UniqueCarrier", "FlightNum","TailNum", "Origin", "Dest", "Distance"]]

Year = dict(all_flights['Year'].value_counts())

YearOnTime = dict(all_flights_no_delays_short['Year'].value_counts())

'''For taking in account only flights that are more than 10'''
for i in YearOnTime:
    if YearOnTime[i]<=10:YearOnTime[i]=0

'''For receiving frequency in %'''
def frequency(nom,denom):
    #print ('Frequency(%) is following (higher is better):')
    #We create dictionary and inserting calculated values
    d={}
    try:
        for i in denom:
            #print (i,' ',int(nom[i]/denom[i]*100),'%')
            d[i]=round(float(nom[i]/denom[i]*100),2)
    except KeyError:
        d[i]=0.00
    return d

mydict=frequency(YearOnTime,Year)

'''For setting the highest value to the first position'''
def sortByValue(x):
    y={}
    for k,v in x.items():
        y[v]=k
    return (sorted(y.items(), reverse=True))

print (sortByValue(mydict))
```

Output:

```
[(68.93, 2003), (58.78, 1998), (57.43, 2006), (56.38, 2002), (56.13, 2001), (52.43,
2007), (49.17, 1991), (48.42, 1992), (48.17, 2004), (46.37, 2005), (45.92, 1993),
(44.02, 1988), (43.67, 2000), (43.17, 1995), (42.82, 1989), (38.27, 1997), (35.22,
1999), (34.92, 1994), (34.47, 1996), (32.07, 1990), (25.61, 2008), (14.96, 1987)]
```

9. Other Calculations

From the table below we are able to find out that there were 1086 flights cancelled (from 43978 flights) and the reason for cancellation we know quite local, i.e. because of the weather - 93 canceled flights, carrier – 81, NAS – 47. Other 865 flights – we don't know the ground of cancellation. Diverted flights were only 109.

There is also an interesting thing that we can break into categories the total arrival delay. For example, Tail Number N742UW had Arrival Delay 130 minutes. If we break into reasons this delay we are going to receive: Carrier Delay - 10 min, Weather Delay – 35 min, NAS Delay – 17 min and Late Aircraft Delay – 68 min which add together 130 minutes.

Input:

```
import csv
import pandas
import matplotlib.pyplot as plt
import numpy as np

FILENAME = "allyears2k.csv"

from pandas import read_csv

all_flights = read_csv(FILENAME, sep=',', encoding='utf-8', low_memory=False)

print ('Cancelled: ', dict(all_flights['Cancelled'].value_counts()))
print ('Quanity: ', len(dict(all_flights['Cancelled'].value_counts())))
print ()
print ('Cancellation Code: ', dict(all_flights['CancellationCode'].value_counts()))
print ('Quanity: ', len(dict(all_flights['CancellationCode'].value_counts())))
print ()
print ('Diverted: ', dict(all_flights['Diverted'].value_counts()))
print ('Quanity: ', len(dict(all_flights['Diverted'].value_counts())))
print ()
```

Output:

```
Cancelled:  {0: 42892, 1: 1086}
Quanity:  2

Cancellation Code:  {'B': 93, 'A': 81, 'C': 47}
Quanity:  3

Diverted:  {0: 43869, 1: 109}
Quanity:  2
```

10. Developing Model (Price Model)

Below is representation of Price Model. It consists of Arrival on Time probabilities that are higher than 50% in line with following categories: Carrier, Flight Number, Origin and Destination. For this categories were taken weighted averages coefficients: 0.05, 0.2, 0.1, 0.1 (subjective). It means for example, if we want to predict air ticket price with flight number that has more than 50% probability of arrival on time, this air ticket should have price = Flight Number probability of arrival on time * 0.2. These coefficients are relative and are subject to review as well as additional categories. I my opinion including in this model Tail Number would be good idea, but I won't do this now because I don't want to overweight this model. Let's live it maximum simplified.

Input:

```
def predictNoArrDelay(Carrier,FlightNumber,Origin,Destination):
    l=[]
    ArrivalOnTimeCoef=0

    CarrierOnTime = [(58.65, 'CO'), (50.45, 'UA')]
    FlightNumberOnTime = [(96.97, 1082), (91.89, 1078), (90.2, 1066), (88.0, 1036),
(85.29, 197), (85.19, 469), (84.62, 1068), (84.38, 1090), (84.21, 1044), (83.87,
1088), (83.33, 2472), (82.76, 1026), (82.35, 596), (81.82, 1043), (81.25, 143),
(80.77, 1054), (80.0, 63), (79.41, 740), (78.89, 84), (78.57, 576), (78.12, 749),
(78.0, 1060), (77.42, 202), (77.08, 1440), (76.92, 2456), (76.67, 786), (76.47,
1930), (75.86, 1635), (75.81, 2753), (75.76, 1020), (75.0, 2546), (74.55, 1062),
(74.19, 1051), (73.81, 1074), (73.77, 478), (73.08, 604), (72.73, 1049), (72.58,
756), (72.0, 424), (71.88, 602), (71.67, 67), (71.43, 603), (71.21, 205), (71.05,
472), (70.97, 2471), (70.77, 199), (70.37, 728), (70.27, 548), (70.0, 97), (69.89,
81), (69.84, 744), (69.7, 802), (69.49, 499), (69.44, 1064), (68.75, 2640), (68.57,
868), (68.33, 796), (68.18, 101), (68.09, 60), (68.0, 1424), (67.86, 387), (67.69,
1061), (67.65, 2489), (66.67, 375), (66.13, 1339), (66.1, 138), (65.71, 768),
(65.62, 2733), (65.52, 1058), (65.45, 1076), (65.08, 2864), (65.0, 462), (64.86,
426), (64.71, 1410), (64.52, 1175), (64.29, 1561), (64.06, 505), (64.0, 597),
(63.64, 1079), (63.49, 102), (63.33, 1704), (63.16, 1412), (63.08, 249), (62.71,
510), (62.5, 109), (62.26, 1071), (62.07, 1432), (62.0, 1052), (61.9, 1467),
(61.76, 397), (61.73, 65), (61.54, 2788), (61.48, 441), (61.4, 758), (61.29, 2049),
(61.19, 378), (61.11, 2602), (60.61, 2020), (60.47, 1080), (60.32, 399), (60.23,
752), (60.0, 1420), (59.82, 85), (59.78, 403), (59.65, 244), (59.63, 1534), (59.62,
1081), (59.38, 2261), (59.26, 724), (58.97, 187), (58.82, 252), (58.73, 220),
(58.62, 1605), (58.49, 246), (58.33, 770), (58.21, 466), (58.06, 203), (57.95,
1063), (57.81, 193), (57.69, 613), (57.63, 148), (57.58, 496), (57.5, 574), (57.45,
128), (57.35, 137), (57.33, 440), (57.26, 755), (57.14, 2601), (56.92, 379), (56.7,
1488), (56.67, 512), (56.6, 761), (56.58, 147), (56.52, 1446), (56.45, 2761),
(56.25, 782), (56.0, 2404), (55.88, 713), (55.56, 215), (55.38, 145), (55.34, 68),
(55.29, 960), (55.21, 116), (55.17, 1417), (55.1, 639), (55.0, 1073), (54.84,
1405), (54.76, 204), (54.69, 111), (54.55, 1231), (54.42, 743), (54.41, 775),
(54.29, 590), (54.17, 558), (54.1, 552), (54.05, 773), (53.97, 497), (53.85, 753),
(53.68, 233), (53.66, 77), (53.57, 1267), (53.33, 1609), (53.27, 79), (53.23,
1028), (53.12, 2794), (53.03, 750), (52.87, 226), (52.78, 690), (52.63, 504),
(52.5, 70), (52.38, 136), (52.34, 419), (52.24, 62), (52.17, 1436), (52.05, 506),
(51.92, 1389), (51.85, 2286), (51.72, 731), (51.61, 2751), (51.56, 475), (51.52,
434), (51.35, 793), (51.28, 149), (51.22, 385), (51.11, 74), (51.02, 1465), (50.98,
133), (50.88, 735), (50.82, 1027), (50.79, 2755), (50.59, 56), (50.43, 227), (50.0,
267)]
    OriginOnTime = [(83.87, 'SAV'), (82.02, 'BOI'), (77.27, 'TRI'), (76.67, 'MLB'),
(71.43, 'MKE'), (70.97, 'ACY'), (70.83, 'OKC'), (70.0, 'PHF'), (67.83, 'HOU'),
(66.67, 'TLH'), (65.22, 'GEG'), (65.15, 'IAH'), (64.71, 'TUL'), (64.29, 'MEM'),
(63.39, 'BHM'), (63.33, 'LIH'), (62.61, 'JAX'), (62.42, 'PWM'), (61.54, 'CAE'),
(61.28, 'MSY'), (60.39, 'AVP'), (58.54, 'RIC'), (58.5, 'IAD'), (58.27, 'DAL'),
(57.69, 'LBB'), (57.24, 'RDU'), (56.0, 'MAF'), (55.62, 'GSO'), (55.29, 'CLE'),
(55.14, 'ORF'), (54.64, 'ISP'), (54.58, 'CLT'), (52.5, 'ELP'), (51.96, 'DTW'),
```

```
(51.72, 'CHS'), (51.38, 'DEN'), (50.94, 'SLC'), (50.89, 'BNA'), (50.35, 'MCI'),
(50.0, 'LIT')]
    DestinationOnTime = [(88.0, 'ICT'), (82.76, 'COS'), (70.97, 'SBN'), (70.24,
'FAY'), (69.49, 'MDT'), (68.75, 'LIT'), (67.24, 'OKC'), (66.36, 'HNL'), (65.52,
'CAK'), (64.52, 'KOA'), (64.1, 'TUL'), (62.5, 'GRR'), (61.29, 'ILM'), (60.84,
'MIA'), (60.14, 'JAX'), (60.09, 'DFW'), (58.72, 'OMA'), (58.65, 'IAD'), (57.72,
'CLE'), (57.69, 'GSP'), (57.5, 'RIC'), (57.14, 'ACY'), (56.84, 'BTV'), (56.67,
'MKE'), (55.37, 'DEN'), (55.06, 'CLT'), (54.84, 'TOL'), (54.39, 'MHT'), (54.24,
'RSW'), (52.94, 'PWM'), (51.26, 'BOS'), (50.9, 'BDL'), (50.84, 'BNA'), (50.0,
'AVP')]

    for i in CarrierOnTime:
        if i[1]==Carrier: l.append(i[0]*0.05)
    for i in FlightNumberOnTime:
        if i[1]==FlightNumber: l.append(i[0]*0.2)
    for i in OriginOnTime:
        if i[1]==Origin: l.append(i[0]*0.1)
    for i in DestinationOnTime:
        if i[1]==Destination: l.append(i[0]*0.1)

    ArrivalOnTimeCoef=sum(l)
    return round((ArrivalOnTimeCoef/100)+1,2)


examples ={'ticket1':{'Carrier':'US', 'FlightNumber':1036, 'Origin':'MLB'
,'Destination':'COS'},
           'ticket2':{'Carrier':'US', 'FlightNumber':197, 'Origin':'NA'
,'Destination':'TSS'},
           'ticket3':{'Carrier':'NA', 'FlightNumber':13, 'Origin':'GEG'
,'Destination':'NA'},
           'ticket4':{'Carrier':'PS', 'FlightNumber':'NA', 'Origin':'MLB'
,'Destination':'OKC'},
           'ticket5':{'Carrier':'CO', 'FlightNumber':0, 'Origin':'MLB'
,'Destination':'CAK'},
           'ticket6':{'Carrier':'NA', 'FlightNumber':'NA', 'Origin':'NA'
,'Destination':'NA'},
           'ticket7':{'Carrier':'CO', 'FlightNumber':'NA', 'Origin':'NA'
,'Destination':''},
           'ticket8':{'Carrier':'CO', 'FlightNumber':'NA', 'Origin': 5
,'Destination':'COS'},
           }

BasePrice=100

for i in examples:

ArrivalOnTimeCoef=predictNoArrDelay(examples[i]['Carrier'],examples[i]['FlightNumbe
r'],examples[i]['Origin'],examples[i]['Destination'])
    FlightTicketPrice=BasePrice*ArrivalOnTimeCoef
    print ('Coefficient of Arrival on time for {} is
{}'.format(i,ArrivalOnTimeCoef))
    print ('The Price for an Air {} should be in around {} % more expensive than
base ticket price.'.format(i,round(FlightTicketPrice,1)-100,))
    print ()
```

Output:

```
Coefficient of Arrival on time for ticket1 is 1.34
The Price for an Air ticket1 should be in around 34.0 % more expensive than base
ticket price.
```

```
Coefficient of Arrival on time for ticket2 is 1.17
The Price for an Air ticket2 should be in around 17.0 % more expensive than base
ticket price.

Coefficient of Arrival on time for ticket3 is 1.07
The Price for an Air ticket3 should be in around 7.0 % more expensive than base
ticket price.

Coefficient of Arrival on time for ticket4 is 1.14
The Price for an Air ticket4 should be in around 14.0 % more expensive than base
ticket price.

Coefficient of Arrival on time for ticket5 is 1.17
The Price for an Air ticket5 should be in around 17.0 % more expensive than base
ticket price.

Coefficient of Arrival on time for ticket6 is 1.0
The Price for an Air ticket6 should be in around 0.0 % more expensive than base
ticket price.

Coefficient of Arrival on time for ticket7 is 1.03
The Price for an Air ticket7 should be in around 3.0 % more expensive than base
ticket price.

Coefficient of Arrival on time for ticket8 is 1.11
The Price for an Air ticket8 should be in around 11.0 % more expensive than base
ticket price.
```

The output of abovementioned model gave us suggested prices of eight different flights. The highest price was given to 1st ticket because flight number 1036 has very high probability of arrival on time.

SUMMARY

During research were emphasized and analyzed historical flights and its attributes. With the help of python 3 and python libraries like pandas and numpy were calculated probabilities of flights that are going to arrive on time according to flights' categories (attributes) like Carrier, Flight Number, Origin, Destination and constructed the Price Model. This model capable to recalculate an air ticket price and make it higher if this air ticket is for flight that has probability of arrival on time more than 50%.

Additionally, were analyzed other flights' categories that were not included in the Price Model intentionally in order to keep this Price Model more simplified. These categories are Tail Number, Day of Week and Destination.

Trying to correlate distances and arrivals on time gave nothing. And that was described in Regression Model and statistical key figures calculations.

There is also an interesting thing that we can break into categories the total arrival delay. For example, Tail Number N742UW had Arrival Delay 130 minutes. If we break into reasons this delay we are going to receive: Carrier Delay - 10 min, Weather Delay – 35 min, NAS Delay – 17 min and Late Aircraft Delay – 68 min which add together 130 minutes.