In [1]:

```
# import libraries
import pandas as pd
import numpy as np
import random
import math
from scipy import stats
from scipy.stats import norm
import datetime
```

Import Historical Exchange Rates

Use exchange rate data as a baseline to simulate trading transaction history

In [2]:

```
# import the historical exchange rates for AUD vs other currencies, put into one table
fx = pd.read_excel('1999-2004 exchange rate.xls') # Please make sure the file is stored in the same folder as the code.
fx = fx.dropna()
fx['FXRUSD'] = fx.FXRUSD.astype("float")
fx = fx.drop(['FXRTWI'],axis = 1)
fx = fx.set_index("Date")
fx_03 = fx[fx.index>'2003-01-01']
fx_03 = fx_03[fx_03.index<'2004-01-01']
fx_03 = 1/fx_03 # invert the exchange rates so they are all in terms of AUD
fx_03
```

Out[2]:

	FXRUSD	FXREUR	FXRJY	FXRUKPS	FXRSF	FXRNZD	FXRCD	FXRHKD	FXRSD	FXRMR	FXRNTD	FXRSKW	FXRIR	FXRCR	FXRSDR
Date															
2003-01-02	1.774938	1.858736	0.014921	2.854696	1.280738	0.930060	1.126634	0.227593	1.018330	0.467093	0.051046	0.001493	0.000198	0.214440	2.404424
2003-01-03	1.776514	1.844338	0.014821	2.836075	1.268714	0.930319	1.133658	0.227801	1.018226	0.467508	0.051046	0.001486	0.000199	0.214638	2.411963
2003-01-06	1.755618	1.834189	0.014728	2.826456	1.259446	0.927300	1.123974	0.225109	1.009591	0.462000	0.050607	0.001475	0.000196	0.212112	2.414293
2003-01-07	1.740644	1.816860	0.014560	2.795639	1.246572	0.925326	1.115200	0.223184	0.999900	0.458064	0.050327	0.001465	0.000195	0.210305	2.365744
2003-01-08	1.743071	1.815871	0.014505	2.796421	1.245951	0.924385	1.115822	0.223489	1.000400	0.458695	0.050327	0.001467	0.000195	0.210602	2.358491
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2003-12-23	1.358880	1.685204	0.012649	2.397507	1.081081	0.872144	1.020616	0.175030	0.795672	0.357603	0.039920	0.001135	0.000160	0.164177	2.005616
2003-12-24	1.352265	1.677290	0.012588	2.385496	1.075847	0.871384	1.023541	0.174137	0.791578	0.355859	0.039714	0.001127	0.000159	0.163377	1.994018
2003-12-29	1.345895	1.676446	0.012579	2.388345	1.074807	0.872981	1.024275	0.173331	0.789328	0.354183	0.039510	0.001122	0.000158	0.162610	1.985309
2003-12-30	1.342823	1.677852	0.012547	2.382654	1.075731	0.875657	1.025746	0.172965	0.789453	0.353369	0.039432	0.001121	0.000159	0.162238	1.988072
2003-12-31	1.333333	1.677008	0.012472	2.374733	1.075153	0.874967	1.030397	0.171753	0.783576	0.350877	0.039246	0.001116	0.000157	0.161095	1.973944

251 rows × 15 columns

Data Simulation

In [3]:

```
# Function for option pricing
def bsm(S,k,vol,t,call,volume,r=0.02):
    volum = volume * 10000
    d1 = (math.log(S/k)+(r+(vol**2)/2)*t)/(vol*(math.sqrt(t)))
    d2 = d1-vol*(math.sqrt(t))
    gamma = dN(d1)/(S*vol*math.sqrt(t))*volum
    vega = S*math.sqrt(t)*dN(d1)*volum
    if call == 1:
        price = norm.cdf(d1)*S-norm.cdf(d2)*k*math.exp(-r*t)
        delta = norm.cdf(d1)*volum
        theta = -(S*dN(d1)*vol)/(2*math.sqrt(t))-(r*k*math.exp(-r*t)*norm.cdf(d2))*volum
        rho = k*t*math.exp(-r*t)*norm.cdf(d2)*volum
    else:
        price = k*math.exp(-r*t)*norm.cdf(-d2)-norm.cdf(-d1)*S
        delta = (norm.cdf(d1)-1)*volum
        theta = -(S*dN(d1)*vol)/(2*math.sqrt(t))+(r*k*math.exp(-r*t)*norm.cdf(-d2))*volum
        rho = -k*t*math.exp(-r*t)*norm.cdf(-d2)*volum
    return price,delta,gamma,theta,vega,rho

def dN(x):
    return 1/(math.sqrt(2*math.pi))*math.exp(-(x**2)/2)

# Function for generating option transactions
def generate_option_trans(date,trader,currency,fx_rate, volume=1000,implied_vol = 0.09):
    call = random.randint(0,1)
    long = random.randint(0,1)
    term = random.randint(1,52)
    strike = (random.randint(80,120)/100)*fx_rate
    contract_rand = random.randint(50,500) # 10000 per contract
    contract_volume = 10000*contract_rand
    price,delta,gamma,theta,vega,rho = bsm(fx_rate,strike,implied_vol,term/52,call,contract_rand)
    if call == 0: # change to -1 for put
        call = -1
    if long == 0: # change to -1 for short
        long = -1
    option_premium = price*long*contract_volume*-1
    if call == 1:
        current_exposure = long*max(fx_rate-strike,0)*contract_volume
    if call == -1:
        current_exposure = long*max(strike-fx_rate,0)*contract_volume
    current_value = long*price*contract_volume
    return {'Date':date,'Trader':trader,'currency':currency,"fx":fx_rate,"call":call,
            "long":long,'volume':contract_rand,'term':term,'strike':strike,'price':price,
            'option premium':option_premium, 'current exposure':current_exposure, 'current_value':current_value,'delta':delta,'gamma':gamma,'theta':theta,'vega':vega,'rho':rho}
```

Simulate new options trading transactions for the currency options trading desk, options priced using the Black Scholes Merton Model

The transaction detail includes general information on the trade such as date, trader names, exchange rate currency and current exchange rate, as well as information on the option such as type of option, long or short position, trade volume, term to expiration, strike price, option price and the earned/expensed option premium. In the end, we also record the current total exposure (intrinsic value) and the current total value, and the Greeks measures of the option

In [4]:

```
# Produce a set of transactions for the currency options trading desk for each day
transactions_dict_03 = {}
currencies = fx_03.columns
for index, row in fx_03.iterrows():
    transactions_dict_03[index]={}
    trader_list = []
    trader_name_list = ['Cindy',"Chris",'Dave','Julio']
    for i in range(4):
        trader_list.append(random.randint(4,8))
    for trader in range(len(trader_list)):
        daily_trades = []
        for j in range(trader_list[trader]):
            currency = random.choice(currencies)
            fx_rate = fx_03.loc[index,currency]
            daily_trades.append(generate_option_trans(index,trader_name_list[trader],currency,fx_rate))
        transactions_dict_03[index][trader_name_list[trader]] = daily_trades
```

Organize the simulated transactions into a pandas DataFrame

Each row represents one transaction

In [5]:

```
# Organize the simulated transactions into a table
transactions_df = pd.DataFrame()
for date in transactions_dict_03.keys():
    date_df = pd.DataFrame()
    for trader in transactions_dict_03[date].keys():
        date_trader_df = pd.DataFrame(transactions_dict_03[date][trader])
        date_df = pd.concat([date_df,date_trader_df],axis = 0)
    transactions_df = pd.concat([transactions_df,date_df],axis = 0)
transactions_df = transactions_df.set_index('Date')
transactions_df.to_excel("Transactions Details.xlsx")
print("New Transactions Information by Trades")
transactions_df
```

```
New Transactions Information by Trades
```

Out[5]:

	Trader	currency	fx	call	long	volume	term	strike	price	option premium	current exposure	current_value	delta	gamma	theta	vega	rho
Date																	
2003-01-02	Cindy	FXRNTD	0.051046	1	1	302	25	0.052067	0.001033	-3118.935120	0.000000	3118.935120	1.351372e+06	3.749337e+08	-1317.277665	42273.161824	3.166530e+04
2003-01-02	Cindy	FXRUSD	1.774938	1	-1	197	23	1.881434	0.012504	24633.717439	0.000000	-24633.717439	4.197721e+05	5.389881e+06	-14408.748169	675946.821548	3.186542e+05
2003-01-02	Cindy	FXRSKW	0.001493	-1	-1	378	20	0.001478	0.000021	81.140929	0.000000	-81.140929	-1.378760e+06	1.704975e+10	42.783475	1314.968157	-8.227599e+02
2003-01-02	Cindy	FXRJY	0.014921	-1	1	370	10	0.017457	0.002470	-9137.330317	9385.258132	9137.330317	-3.699791e+06	1.452893e+06	1286.832242	5.598402	-1.237339e+04
2003-01-02	Cindy	FXRHKD	0.227593	1	1	90	5	0.204834	0.023153	-20837.664665	20483.408439	20837.664665	8.999486e+05	3.310792e+04	-3679.694399	14.840907	1.769084e+04
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2003-12-31	Julio	FXRUSD	1.333333	-1	1	190	34	1.400000	0.068294	-129758.367677	126666.666667	129758.367677	-1.282902e+06	7.045602e+06	36805.855562	737078.364216	-1.203269e+06
2003-12-31	Julio	FXRHKD	0.171753	1	-1	272	37	0.151143	0.022891	62262.379824	-56060.319805	-62262.379824	2.643528e+06	1.344851e+07	-7835.454596	25405.386796	2.787613e+05
2003-12-31	Julio	FXRMR	0.350877	1	-1	148	8	0.361404	0.001646	2435.603364	0.000000	-2435.603364	3.432476e+05	3.645114e+07	-2360.055068	62137.033355	1.815418e+04
2003-12-31	Julio	FXRIR	0.000157	1	1	52	4	0.000146	0.000011	-5.847436	5.729577	5.847436	5.192540e+05	6.200567e+08	-1.517725	0.106359	5.837403e+00
2003-12-31	Julio	FXRUSD	1.333333	-1	1	85	39	1.520000	0.167347	-142244.993308	158666.666667	142244.993308	-7.874613e+05	1.140963e+06	23843.857282	136915.530540	-8.941450e+05

6005 rows × 17 columns

Use the transactions to simulate a daily portfolio for the options trading desk as well as a record for realized profits and losses

Other than details recorded in the transactions detail table, the daily portfolio also includes days to expiration and the transaction date for each option in the portfolio.

The record for realized profits and losses include gains/losses both from option premiums received/paid and cost/revenue from unwinding the position by buying/selling the option. The type of p&l and the amount are specified in the table.

Assume that 80% of the options from the previous day are unwinded each day to maintain the simplicity of data.

In [6]:

```
# Create daily portfolio and daily realized profit and loss data
daily_portfolio = {}
realized_pnl = {}
currencies_position_zero_dict = {}
for currency in currencies:
    currencies_position_zero_dict[currency]=0

initial_date = datetime.datetime(2003,1,2)

for date in transactions_dict_03.keys():
    current_position = []
    realized_position = []

    # update old information and simulate options selling
    if date > initial_date:
        total_past_trades = []

        for past_pos in daily_portfolio[date_current]:
            past_trade = past_pos.copy()

            past_trade['days to maturity'] -= 1
            past_trade['Date']=date
            fx_today = fx_03.loc[date,past_trade['currency']]
            past_trade['fx']=fx_today

            # not yet expired
            if past_trade['days to maturity'] != 0:
                past_trade['price'],past_trade['delta'],past_trade['gamma'],past_trade['theta'],past_trade['vega'],past_trade['rho']= bsm(past_trade['fx'],past_trade['strike'],0.09,
                                         past_trade['days to maturity']/252,past_trade['call'],past_trade['volume'])
                past_trade['current_value'] = past_trade['price']*past_trade['long']*past_trade['volume']*10000

                # calculate new exposure based on fx rate today
                if past_trade['call'] == 1:
                    past_trade['current exposure'] = past_trade['long']*max(fx_today-past_trade['strike'],0)*past_trade['volume']*10000
                elif past_trade['call'] == -1:
                    past_trade['current exposure'] = past_trade['long']*max(past_trade['strike']-fx_today,0)*past_trade['volume']*10000
                total_past_trades.append(past_trade)

            # expired
            elif past_trade['days to maturity'] == 0:
                past_trade['type'] = 'expired'
                past_trade['p&l'] = past_trade['current exposure']
                realized_position.append(past_trade)

        # sell some options (simulated)
        random.shuffle(total_past_trades)
        remove_num = len(total_past_trades)//5*4
        remove_trades = total_past_trades[:remove_num]
        current_position.extend(total_past_trades[remove_num:])
        for trades in remove_trades:
            trades['type'] = 'closed'
            trades['p&l'] = trades['current_value']
            realized_position.append(trades)

    # add new trades information
    for trader in transactions_dict_03[date].keys():
        new_transactions_count = 0
        for trade in transactions_dict_03[date][trader]:
            trade_position = trade.copy()
            trade_position['days to maturity'] = trade_position['term']*5
            trade_position['Start date'] = date
            current_position.append(trade_position)
            new_transactions_count+=1
            trade_prem = trade_position.copy()
            trade_prem['type'] = 'option premium'
            trade_prem['p&l'] = -1*trade_prem['current_value']
            realized_position.append(trade_prem)
    daily_portfolio[date]=current_position
    realized_pnl[date] = realized_position
    date_current = date
```

Organize the simulated daily portfolio into a pandas DataFrame

Each row represents one transaction

In [7]:

```
# Organize the simulated daily portfolio data into a table
daily_port_df = pd.DataFrame()
for date in daily_portfolio.keys():
    date_df = pd.DataFrame(daily_portfolio[date])
    daily_port_df = pd.concat([daily_port_df,date_df],axis = 0)
daily_port_df.to_excel('daily_portfolio.xlsx')
print("Portfolio Information by Trades")
daily_port_df
```

```
Portfolio Information by Trades
```

Out[7]:

	Date	Trader	currency	fx	call	long	volume	term	strike	price	option premium	current exposure	current_value	delta	gamma	theta	vega	rho	days to maturity	Start date
0	2003-01-02	Cindy	FXRNTD	0.051046	1	1	302	25	0.052067	0.001033	-3118.935120	0.000000	3118.935120	1.351372e+06	3.749337e+08	-1317.277665	42273.161824	3.166530e+04	125	2003-01-02
1	2003-01-02	Cindy	FXRUSD	1.774938	1	-1	197	23	1.881434	0.012504	24633.717439	0.000000	-24633.717439	4.197721e+05	5.389881e+06	-14408.748169	675946.821548	3.186542e+05	115	2003-01-02
2	2003-01-02	Cindy	FXRSKW	0.001493	-1	-1	378	20	0.001478	0.000021	81.140929	0.000000	-81.140929	-1.378760e+06	1.704975e+10	42.783475	1314.968157	-8.227599e+02	100	2003-01-02
3	2003-01-02	Cindy	FXRJY	0.014921	-1	1	370	10	0.017457	0.002470	-9137.330317	9385.258132	9137.330317	-3.699791e+06	1.452893e+06	1286.832242	5.598402	-1.237339e+04	50	2003-01-02
4	2003-01-02	Cindy	FXRHKD	0.227593	1	1	90	5	0.204834	0.023153	-20837.664665	20483.408439	20837.664665	8.999486e+05	3.310792e+04	-3679.694399	14.840907	1.769084e+04	25	2003-01-02
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
31	2003-12-31	Julio	FXRUSD	1.333333	-1	1	190	34	1.400000	0.068294	-129758.367677	126666.666667	129758.367677	-1.282902e+06	7.045602e+06	36805.855562	737078.364216	-1.203269e+06	170	2003-12-31
32	2003-12-31	Julio	FXRHKD	0.171753	1	-1	272	37	0.151143	0.022891	62262.379824	-56060.319805	-62262.379824	2.643528e+06	1.344851e+07	-7835.454596	25405.386796	2.787613e+05	185	2003-12-31
33	2003-12-31	Julio	FXRMR	0.350877	1	-1	148	8	0.361404	0.001646	2435.603364	0.000000	-2435.603364	3.432476e+05	3.645114e+07	-2360.055068	62137.033355	1.815418e+04	40	2003-12-31
34	2003-12-31	Julio	FXRIR	0.000157	1	1	52	4	0.000146	0.000011	-5.847436	5.729577	5.847436	5.192540e+05	6.200567e+08	-1.517725	0.106359	5.837403e+00	20	2003-12-31
35	2003-12-31	Julio	FXRUSD	1.333333	-1	1	85	39	1.520000	0.167347	-142244.993308	158666.666667	142244.993308	-7.874613e+05	1.140963e+06	23843.857282	136915.530540	-8.941450e+05	195	2003-12-31

7993 rows × 20 columns

Organize the simulated realized profits and losses into a pandas DataFrame

Each row represents one transaction

In [8]:

```
# Organize the simulated daily realized profit and loss data into a table
realized_pnl_df = pd.DataFrame()
for date in realized_pnl.keys():
    date_df = pd.DataFrame(realized_pnl[date])
    realized_pnl_df = pd.concat([realized_pnl_df,date_df],axis = 0)
realized_pnl_df.to_excel('realized_p&l.xlsx')
print("Realized Profits and Losses by Trade")
realized_pnl_df
```

```
Realized Profits and Losses by Trade
```

Out[8]:

	Date	Trader	currency	fx	call	long	volume	term	strike	price	...	current_value	delta	gamma	theta	vega	rho	days to maturity	Start date	type	p&l
0	2003-01-02	Cindy	FXRNTD	0.051046	1	1	302	25	0.052067	0.001033	...	3118.935120	1.351372e+06	3.749337e+08	-1317.277665	42273.161824	3.166530e+04	125	2003-01-02	option premium	-3118.935120
1	2003-01-02	Cindy	FXRUSD	1.774938	1	-1	197	23	1.881434	0.012504	...	-24633.717439	4.197721e+05	5.389881e+06	-14408.748169	675946.821548	3.186542e+05	115	2003-01-02	option premium	24633.717439
2	2003-01-02	Cindy	FXRSKW	0.001493	-1	-1	378	20	0.001478	0.000021	...	-81.140929	-1.378760e+06	1.704975e+10	42.783475	1314.968157	-8.227599e+02	100	2003-01-02	option premium	81.140929
3	2003-01-02	Cindy	FXRJY	0.014921	-1	1	370	10	0.017457	0.002470	...	9137.330317	-3.699791e+06	1.452893e+06	1286.832242	5.598402	-1.237339e+04	50	2003-01-02	option premium	-9137.330317
4	2003-01-02	Cindy	FXRHKD	0.227593	1	1	90	5	0.204834	0.023153	...	20837.664665	8.999486e+05	3.310792e+04	-3679.694399	14.840907	1.769084e+04	25	2003-01-02	option premium	-20837.664665
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
47	2003-12-31	Julio	FXRUSD	1.333333	-1	1	190	34	1.400000	0.068294	...	129758.367677	-1.282902e+06	7.045602e+06	36805.855562	737078.364216	-1.203269e+06	170	2003-12-31	option premium	-129758.367677
48	2003-12-31	Julio	FXRHKD	0.171753	1	-1	272	37	0.151143	0.022891	...	-62262.379824	2.643528e+06	1.344851e+07	-7835.454596	25405.386796	2.787613e+05	185	2003-12-31	option premium	62262.379824
49	2003-12-31	Julio	FXRMR	0.350877	1	-1	148	8	0.361404	0.001646	...	-2435.603364	3.432476e+05	3.645114e+07	-2360.055068	62137.033355	1.815418e+04	40	2003-12-31	option premium	2435.603364
50	2003-12-31	Julio	FXRIR	0.000157	1	1	52	4	0.000146	0.000011	...	5.847436	5.192540e+05	6.200567e+08	-1.517725	0.106359	5.837403e+00	20	2003-12-31	option premium	-5.847436
51	2003-12-31	Julio	FXRUSD	1.333333	-1	1	85	39	1.520000	0.167347	...	142244.993308	-7.874613e+05	1.140963e+06	23843.857282	136915.530540	-8.941450e+05	195	2003-12-31	option premium	-142244.993308

11974 rows × 22 columns

Data Remediation for Dashboard Contents

Daily Portfolio Values for VaR and Stressed VaR Calculation

VaR and Stressed VaR calculations are directly performed through the Dashboard creation process, using this generated table

In [9]:

```
# Produce daily exposures from portfolio's current values
# Organize them into a table for VaR and Stressed VaR calculations
VaR_dict = {}
current_date = 'initial_date'
```

```
for index,row in daily_port_df.iterrows():
    #within same date
    if row[0]==current_date:
        VaR_dict[current_date]+=row[12]
    else: #move onto next date
        current_date = row[0]
        VaR_dict[current_date]=row[12]

VaR_df = pd.DataFrame([VaR_dict])
VaR_df = VaR_df.transpose()
VaR_df.columns = ['Daily Portfolio Values']
VaR_df.to_excel('Daily portfolio value for VaR Calculation.xlsx')
print("Daily Portfolio Values")
VaR_df
```

```
Daily Portfolio Values
```

Out[9]:

	Daily Portfolio Values
2003-01-02	-1.371150e+06
2003-01-03	-2.652400e+05
2003-01-06	-1.558938e+06
2003-01-07	-1.482430e+06
2003-01-08	2.586530e+06
...	...
2003-12-23	3.144009e+06
2003-12-24	-4.886601e+05
2003-12-29	-8.681378e+05
2003-12-30	-1.033002e+05
2003-12-31	-1.241664e+06

251 rows × 1 columns

Daily Trades by Trader

Records the number of new trades that each trader conduct on a given day. The rows represent each day.

In [10]:

```
# Produce a table for trader's daily trades
trade_trend_dict = {}
current_date = 'initial_date'

for index,row in transactions_df.iterrows():
    trader = row[0]
    #within same date
    if index==current_date:
        trade_trend_dict[current_date]['Trades'] += 1
        trade_trend_dict[current_date][trader] += 1
    else: #move onto next date
        current_date = index
        trade_trend_dict[current_date]={}
        trade_trend_dict[current_date]['Trades'] = 1
        for traders in ['Cindy','Chris','Dave','Julio']:
            trade_trend_dict[current_date][traders] = 0
        trade_trend_dict[current_date][trader] += 1

trade_trend_df = pd.DataFrame(trade_trend_dict)
trade_trend_df = trade_trend_df.transpose()
trade_trend_df.to_excel('trade_trend.xlsx')
print("Trade Trends for Traders")
trade_trend_df
```

```
Trade Trends for Traders
```

Out[10]:

	Trades	Cindy	Chris	Dave	Julio
2003-01-02	25	5	7	5	8
2003-01-03	20	6	5	5	4
2003-01-06	19	4	6	5	4
2003-01-07	23	6	7	4	6
2003-01-08	24	7	5	6	6
...	...	...	...	...	...
2003-12-23	25	7	4	6	8
2003-12-24	26	7	7	4	8
2003-12-29	21	4	4	5	8
2003-12-30	25	4	7	7	7
2003-12-31	28	6	7	8	7

251 rows × 5 columns

Daily Total Profits and Losses

Records the total profits and losses occurred in each day, including both realized and unrealized components, by the types of currency. The rows represent each day.

In [11]:

```
# Produce a table to generate daily realized & unrealized profits and losses
pnl_dict = {}
current_date = 'initial_date'

for index,row in realized_pnl_df.iterrows(): # Record all the realized p&l
    currency = row[2]
    fx = row[3]
    #within same date
    if row[0]==current_date:
        pnl_dict[current_date]['p&l'] += row[-1]
        pnl_dict[current_date][currency] += row[-1]
    else: #move onto next date
        current_date = row[0]
        pnl_dict[current_date] = {}
        pnl_dict[current_date]['p&l']=row[-1]
        for cur_type in currencies:
            pnl_dict[current_date][cur_type] = 0
        pnl_dict[current_date][currency] += row[-1]
last_date = current_date

for index,row in daily_port_df.iterrows(): # Record all the unrealized p&l
    currency = row[2]
    fx = row[3]
    #within same date
    if row[0]==current_date:
        pnl_dict[current_date]['p&l'] += row[12]
        pnl_dict[current_date][currency] += row[12]
    else: #move onto next date
        current_date = row[0]
        pnl_dict[current_date]['p&l']+=row[12]
        pnl_dict[current_date][currency] += row[12]
last_date = current_date

pnl_df = pd.DataFrame(pnl_dict)
pnl_df = pnl_df.transpose()
pnl_df.to_excel('pnl_breakdown.xlsx')
print("Daily Total Profits and Losses by Currencies")
pnl_df
```

```
Daily Total Profits and Losses by Currencies
```

Out[11]:

	p&l	FXRUSD	FXREUR	FXRJY	FXRUKPS	FXRSF	FXRNZD	FXRCD	FXRHKD	FXRSD	FXRMR	FXRNTD	FXRSKW	FXRIR	
2003-01-02	-8.149073e-10	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.000000	-1.393615e-12	0.000000e+00	0.000000	-1.114339e-13	0.000000e+00	0.000000	1
2003-01-03	-1.308526e+06	-8.806936e+05	-943796.590499	6.862112e+02	-8.526513e-14	8.026690e+05	257297.131040	0.000000	1.228405e+05	0.000000e+00	0.053030	3.178293e+03	4.554795e+02	3.659535	-1.
2003-01-06	-7.928533e+04	-5.124001e+05	0.000000	-8.862929e+03	9.375972e+03	1.128757e+06	-357659.540944	0.000000	-1.400071e+05	-5.362473e+04	-118056.338864	7.346091e+03	-1.090019e+02	2.983560	-3.
2003-01-07	-1.525106e+06	0.000000e+00	54032.369506	5.400125e-13	-8.350517e+05	-8.491196e+05	280740.033136	-7665.711192	2.617049e+04	6.814891e+05	-119664.646797	4.159252e+04	-6.615033e+00	-65.591331	4.
2003-01-08	-1.470337e+06	-2.910383e-11	-0.732649	-9.767939e+03	-1.824372e+06	-2.525488e+05	-67248.914767	205370.839491	-1.966147e+04	-2.910383e-11	58030.681394	4.382704e+04	-1.333090e-01	-122.240701	-1.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2003-12-23	1.063401e+05	1.123999e+06	-297940.415889	-1.097418e+03	-9.015490e+01	-1.690252e+01	-206968.410727	73836.966732	-3.485550e+04	-5.333309e+05	-5131.482378	-1.589557e+03	3.611396e+02	15.459639	-1.
2003-12-24	3.127086e+06	1.897055e+06	524816.173323	-6.610349e+01	6.055618e+05	4.771738e+05	-290886.863330	-903807.494170	-3.733826e+03	6.376611e+05	115054.270162	3.523002e+02	-1.136868e-13	10.118917	7.
2003-12-29	-5.452793e+05	1.063159e+06	-36825.355503	-1.461971e+03	-2.364686e-11	0.000000e+00	-5453.647391	-949495.088435	0.000000e+00	-6.566300e+04	54660.333688	-4.190491e+04	-1.208093e+03	9.236684	-1.
2003-12-30	-8.853783e+05	7.422192e+05	-604039.510987	2.853720e+03	-1.247868e+06	3.363963e+05	-28853.236279	642777.052635	0.000000e+00	0.000000e+00	-17720.034706	-4.009264e+04	6.876630e+02	0.000000	4.
2003-12-31	-8.175399e+04	3.045537e+05	0.000000	5.762006e+01	3.193180e+05	2.298906e+05	107927.457909	-140365.758997	1.781284e+04	-2.971754e+05	137498.049799	1.638607e+03	7.026495e+02	-66.276371	-8.

251 rows × 16 columns

Daily Portfolio data for the most recent date (Dec. 31, 2003)

Produce a table specifically for the last date (assumed to be the current date). This information is used to generate sensitivity analysis to prospective changes in the foreign exchange rates.

In [12]:

```
# Produce a table of the daily portfolio data for the most recent date
last_date_df = pd.DataFrame()
for index, row in daily_port_df.iterrows():
    if row[0] == last_date:
        last_date_df = pd.concat([last_date_df, row], axis=1)
last_date_df = last_date_df.transpose()
```

Sensitivity Analysis for the most recent date's portfolio

Measures the portfolio sensitivities to the prospective changes exchange rates. The percentage changes in the exchange rate is shown in the first few columns and the information on the transaction (e.g. currency, volume) are shown in the later columns.

In [13]:

```
# Define a function to calculate the sensitivity of portfolio values to changes in fx rates
def sensitivity(fx,k,vol,t,call,long,volume,current_value,percent=0):
    new_fx = fx*(1+percent)
    new_price,delta,gamma,theta,vega,rho = bsm(new_fx,k,vol,t,call,volume)
    price,delta,gamma,theta,vega,rho = bsm(fx,k,vol,t,call,volume)
    new_current_value = long*new_price*volume*10000
    current_value = long*price*volume*10000
    difference = new_current_value - current_value
    return difference
```

In [14]:

```
# Produce a table to analyze the sensitivity of the portfolio to changes in fx rates
sensitivity_df = pd.DataFrame()
for index,row in last_date_df.iterrows():
    row_list = []
    for sensitivities in [-0.3,-0.15,-0.05,-0.02,-0.01,0,0.01,0.02,0.05,0.15,0.3]:
        row_list.append(sensitivity(row.fx,row.strike,0.09,row['days to maturity']/252,row.call,row.long,row.volume,row['current_value'],sensitivities))
    sensitivity_series = pd.Series(row_list, index = [-0.3,-0.15,-0.05,-0.02,-0.01,0,0.01,0.02,0.05,0.15,0.3])
    row = pd.concat([row, sensitivity_series], axis=0)
    sensitivity_df = sensitivity_df.append(row, ignore_index=True)
sensitivity_df.to_excel('sensitivity_analysis.xlsx')
print("Sensitivity Analysis for Most Recent Date")
sensitivity_df
```

```
Sensitivity Analysis for Most Recent Date
```

Out[14]:

	-0.3	-0.15	-0.05	-0.02	-0.01	0.0	0.01	0.02	0.05	0.15	...	gamma	long	option premium	price	rho	strike
0	5.186088e+04	5.140632e+04	3.088513e+04	1.377277e+04	7.074926e+03	0.0	-7.379520e+03	-1.499721e+04	-38751.384890	-120550.149811	...	2.469814e+07	-1.0	5.690410e+04	2.216277e-02	2.373441e+05	0.332167
1	5.329996e+05	1.395158e+05	1.936181e+04	5.697143e+03	2.569278e+03	0.0	-2.093310e+03	-3.785229e+03	-7074.487700	-9752.073904	...	4.111689e+06	1.0	-9.374981e+03	3.572997e-03	-2.484294e+05	0.968158
2	-8.833126e-01	-8.833126e-01	-8.833005e-01	-8.658997e-01	-7.479716e-01	0.0	3.990393e+00	2.200567e+01	966.810603	204810.018070	...	2.528282e+04	1.0	-4.749040e-01	2.098130e-07	1.185979e+01	1.138578
3	4.443214e+05	4.168184e+05	1.968875e+05	8.303564e+04	4.206853e+04	0.0	-4.295857e+04	-8.662310e+04	-220425.225201	-676188.101519	...	8.361330e+06	-1.0	4.235731e+05	1.000738e-01	2.252933e+06	0.943687
4	-5.762006e+01	-5.758242e+01	-4.897779e+01	-2.909811e+01	-1.672168e+01	0.0	2.219261e+01	5.114354e+01	193.529605	1748.805431	...	3.508447e+08	1.0	-6.478704e+01	1.401948e-05	1.363908e+03	0.014555
5	-1.304253e+03	-1.277584e+03	-8.098897e+02	-3.908714e+02	-2.073658e+02	0.0	2.320051e+02	4.891384e+02	1411.734446	5870.015915	...	1.601235e+08	1.0	-1.371291e+03	8.695072e-04	2.081759e+04	0.041404
6	-2.199836e+05	-2.083944e+05	-1.097948e+05	-4.869088e+04	-2.508520e+04	0.0	2.645962e+04	5.417983e+04	143714.893115	480416.695426	...	1.189619e+07	1.0	-2.188932e+05	6.666467e-02	2.143773e+06	1.043460
7	-2.825347e+05	-2.401745e+05	-9.409146e+04	-3.821073e+04	-1.916847e+04	0.0	1.926087e+04	3.858828e+04	96800.543737	291466.949836	...	8.639973e+05	1.0	-2.733773e+05	1.495324e-01	1.034347e+06	0.892399

	-0.3	-0.15	-0.05	-0.02	-0.01	0.0	0.01	0.02	0.05	0.15	...	gamma	long	option premium	price	rho	strike
8	-1.306995e+02	-1.306995e+02	-1.287058e+02	-1.025972e+02	-6.876921e+01	0.0	1.338567e+02	3.836814e+02	2882.687129	142699.406673	...	9.644694e+04	1.0	-1.057567e+02	2.448071e-05	2.396958e+03	2.802185
9	1.337161e+06	1.091054e+05	2.166195e+03	3.413517e+02	1.274497e+02	0.0	-7.457698e+01	-1.174533e+02	-162.517287	-170.804915	...	8.147871e+04	1.0	-1.433902e+02	3.103685e-05	-4.073284e+03	1.971028
10	3.743157e+00	3.743098e+00	3.530889e+00	2.451603e+00	1.514911e+00	0.0	-2.385187e+00	-6.044412e+00	-30.919638	-657.070186	...	4.958168e+07	-1.0	3.163709e+00	8.937032e-07	8.066781e+01	0.014966
11	3.784354e+02	3.561343e+02	1.553819e+02	6.352376e+01	3.189584e+01	0.0	-3.207457e+01	-6.426501e+01	-161.172177	-484.834741	...	1.341626e+09	-1.0	3.777001e+02	1.302414e-04	1.032195e+03	0.000993
12	8.395751e-10	8.395751e-10	8.395751e-10	8.391293e-10	8.177203e-10	0.0	-2.462573e-08	-6.134881e-07	-0.002344	-12747.321630	...	1.085235e-05	-1.0	3.254595e-10	8.519881e-17	6.812505e-09	2.329254
13	-1.147545e+06	-5.710383e+05	-1.869815e+05	-7.330402e+04	-3.626050e+04	0.0	3.517804e+04	6.892841e+04	157943.999056	283161.783909	...	9.899184e+06	-1.0	2.902363e+05	7.781135e-02	-1.045016e+06	1.112828
14	-4.339492e+05	-2.001473e+05	-5.224430e+04	-1.786280e+04	-8.388129e+03	0.0	7.293979e+03	1.351964e+04	26462.737462	36649.190745	...	9.638683e+06	-1.0	3.653920e+04	2.519945e-02	-3.168572e+05	1.085905
15	7.116614e+01	1.975889e+01	2.554347e+00	7.186141e-01	3.189523e-01	0.0	-2.517122e-01	-4.481866e-01	-0.803443	-1.034991	...	2.692537e+10	1.0	-9.907823e-01	4.234113e-07	-2.138759e+01	0.000143
16	-4.901603e+04	-4.728863e+04	-2.744554e+04	-1.271285e+04	-6.648782e+03	0.0	7.227684e+03	1.502180e+04	41592.435424	155060.902146	...	4.782643e+07	1.0	-4.785761e+04	1.262734e-02	6.420824e+05	0.357895
17	6.606118e+04	6.566889e+04	4.482833e+04	2.184548e+04	1.158287e+04	0.0	-1.286713e+04	-2.695456e+04	-75592.292870	-274761.843445	...	2.129696e+07	-1.0	6.486018e+04	2.393365e-02	5.780056e+05	0.783576
18	6.822216e+04	6.730496e+04	4.109286e+04	1.887824e+04	9.811681e+03	0.0	-1.048548e+04	-2.156655e+04	-57587.797678	-190457.295684	...	5.547357e+07	-1.0	6.748456e+04	1.739293e-02	4.752848e+05	0.340351
19	-1.590488e+04	-1.515505e+04	-6.658622e+03	-2.713167e+03	-1.360986e+03	0.0	1.366473e+03	2.736252e+03	6854.143477	20593.802579	...	1.708280e+06	1.0	-1.587927e+04	1.984908e-02	3.711605e+04	0.152861
20	-2.149230e+02	-2.146486e+02	-1.729550e+02	-9.639323e+01	-5.385192e+01	0.0	6.690943e+01	1.485751e+02	496.869693	2896.656762	...	8.335439e+08	1.0	-2.027907e+02	6.562805e-05	3.339325e+03	0.013594
21	-1.656949e+06	-8.269798e+05	-2.738005e+05	-1.086474e+05	-5.408172e+04	0.0	5.337046e+04	1.057389e+05	253055.829463	536320.748654	...	1.189182e+06	-1.0	5.736528e+05	2.462029e-01	-1.834647e+06	2.635953
22	9.319905e+00	9.270258e+00	6.704276e+00	3.448479e+00	1.869108e+00	0.0	-2.179820e+00	-4.686954e+00	-14.260541	-65.603019	...	1.264474e+11	-1.0	8.957994e+00	2.054586e-06	1.311958e+02	0.000167
23	3.482493e+01	8.861350e+00	9.658673e-01	2.557752e-01	1.113514e-01	0.0	-8.477157e-02	-1.485097e-01	-0.255798	-0.313787	...	1.053716e+10	1.0	-2.962198e-01	2.489242e-07	-6.486168e+00	0.000142
24	1.672963e+02	8.207460e+01	2.602715e+01	1.008934e+01	4.979773e+00	0.0	-4.832249e+00	-9.499371e+00	-22.346372	-50.078186	...	5.882098e+10	1.0	-6.048435e+01	1.675467e-05	-4.691518e+02	0.000176
25	-1.167331e+04	-1.097191e+04	-5.117424e+03	-2.143947e+03	-1.083905e+03	0.0	1.102568e+03	2.219436e+03	5623.831445	17138.082202	...	1.187520e+08	1.0	-1.162883e+04	3.955384e-03	5.088965e+04	0.035714
26	4.173545e+04	4.128610e+04	2.521763e+04	1.147666e+04	5.940543e+03	0.0	-6.291808e+03	-1.288068e+04	-33929.678173	-109015.644290	...	1.993429e+06	-1.0	4.138992e+04	7.261389e-02	2.423446e+05	1.280000
27	-6.398758e+05	-3.199376e+05	-1.066455e+05	-4.265793e+04	-2.132883e+04	0.0	2.132807e+04	4.265406e+04	106566.284791	296955.087568	...	5.154160e+03	-1.0	3.539736e+05	1.710017e-01	-4.304159e+05	1.205564
28	3.063260e+02	3.063259e+02	2.976934e+02	2.227337e+02	1.430073e+02	0.0	-2.461351e+02	-6.532318e+02	-3767.436937	-81281.898045	...	8.034941e+05	-1.0	2.644767e+02	1.139986e-04	5.011827e+03	1.225675
29	-5.591270e+05	-2.678481e+05	-7.960909e+04	-2.968732e+04	-1.442994e+04	0.0	1.353373e+04	2.611549e+04	57832.833918	106281.233024	...	1.179349e+07	-1.0	1.132369e+05	5.100761e-02	-9.664201e+05	0.927465
30	5.172068e+02	5.159707e+02	3.754005e+02	1.882068e+02	1.006187e+02	0.0	-1.133310e+02	-2.387080e+02	-676.700906	-2465.981403	...	8.324162e+08	-1.0	5.079266e+02	3.363752e-04	4.110401e+03	0.012472
31	7.209738e+05	3.411751e+05	9.868399e+04	3.634515e+04	1.758850e+04	0.0	-1.634942e+04	-3.140891e+04	-68675.131486	-122768.670187	...	7.045602e+06	1.0	-1.297584e+05	6.829388e-02	-1.203269e+06	1.400000
32	6.245292e+04	5.373856e+04	2.188361e+04	8.970019e+03	4.511208e+03	0.0	-4.552146e+03	-9.135822e+03	-23007.898131	-69652.880356	...	1.344851e+07	-1.0	6.226238e+04	2.289058e-02	2.787613e+05	0.151143
33	2.535786e+03	2.535786e+03	2.436937e+03	1.684144e+03	1.019250e+03	0.0	-1.464160e+03	-3.444714e+03	-12676.389484	-61479.646985	...	3.645114e+07	-1.0	2.435603e+03	1.645678e-03	1.815418e+04	0.361404
34	-5.851303e+00	-5.851188e+00	-3.898449e+00	-1.626169e+00	-8.158552e-01	0.0	8.177563e-01	1.636079e+00	4.091561	12.276670	...	6.200567e+08	1.0	-5.847436e+00	1.124507e-05	5.837403e+00	0.000146
35	3.369486e+05	1.669534e+05	5.420198e+04	2.127443e+04	1.054991e+04	0.0	-1.034133e+04	-2.043784e+04	-48880.764522	-114916.948109	...	1.140963e+06	1.0	-1.422450e+05	1.673471e-01	-8.941450e+05	1.520000

36 rows × 31 columns

We primarily used Python for data generating and organizing process, and decided to use Power BI to visualize our data and produce the Risk Dashboard.