Desktop User Interface Using PyQT5

July 12, 2021

[ ]:
```
from PyQt5 import QtWidgets
from PyQt5.QtCore import QProcess
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QLabel,␣
↪QVBoxLayout, QWidget
from PyQt5.QtGui import QIcon, QPixmap

import pandas as pd
import numpy as np
import xlrd
import datetime
import dateutil.relativedelta
import warnings

warnings.filterwarnings("ignore")
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog, QMessageBox
from PyQt5.QtCore import QDate

import matplotlib.pyplot
matplotlib.use('Qt5Agg')
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.figure import Figure

import sys

from random import randint
matplotlib.use('Qt5Agg')
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.figure import Figure


class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.title = "Cost Ninja"
        self.setWindowTitle(self.title)
```

```
        self.p = None

        label = QLabel(self)
        pixmap = QPixmap('main_white.PNG')
        label.setPixmap(pixmap)
        self.setCentralWidget(label)
        self.resize(pixmap.width(), pixmap.height())

        button = QPushButton(self)
        button.setGeometry(445, 490, 135, 50)
        button.clicked.connect(self.start_process)
        button.setStyleSheet("background-image : url(start_white.PNG);")

    def start_process(self):
        import os
        os.system("python 2.py")
        self.close()
        # if self.p is None:  # No process running.
        #     self.p = QProcess(self)
        #     self.p.start("python3", ['ui0302.py'])
        #     self.p.show()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    w = MainWindow()
    w.show()
    app.exec_()
```

[ ]:
```
df = pd.read_excel("Cost error project.xlsx", header=1)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1400, 1200)
        MainWindow.setStyleSheet("background-color : white")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")

        self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout.setObjectName("verticalLayout")

        self.scrollArea = QtWidgets.QScrollArea(self.centralwidget)
        self.scrollArea.setWidgetResizable(True)
        self.scrollArea.setObjectName("scrollArea")
        self.scrollAreaWidgetContents = QtWidgets.QWidget()
        self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0, 0, 600, 458))
        self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")
```

```
        self.tableWidget = QtWidgets.QTableWidget(self.scrollAreaWidgetContents)
        self.tableWidget.setGeometry(QtCore.QRect(650, 60, 500, 471))
        self.tableWidget.setRowCount(20)
        self.tableWidget.setColumnCount(5)
        self.tableWidget.setHorizontalHeaderLabels(['mcode', 'buyprice',␣
↪'sellprice', 'allocation','flag'])
        self.tableWidget.setObjectName("tableWidget")
        self.tableWidget.setStyleSheet("background-color: #D5DBDB;\n"
                                       "gridline-color: 1px solid black;\n")

        self.scrollArea.setWidget(self.scrollAreaWidgetContents)
        self.verticalLayout.addWidget(self.scrollArea)

        self.btn1 = QtWidgets.QPushButton(self.centralwidget)
        self.btn1.setGeometry(QtCore.QRect(340, 110, 141, 51))
        # self.btn1.setStyleSheet("background-color : black")
        self.btn1.setStyleSheet("background-color: #717D7E;\n"
                                "border: black;\n"
                                "color: white;\n"
                                "")
        self.btn1.setObjectName("btn1")
        self.btn1.clicked.connect(self.popup)
        self.verticalLayout.addWidget(self.btn1)
        # self.btn1.clicked.connect(self.click)

        self.label1 = QtWidgets.QLabel(self.centralwidget)
        self.label1.setGeometry(QtCore.QRect(20, 0, 131, 41))
        font = QtGui.QFont()
        font.setPointSize(17)
        self.label1.setFont(font)
        self.label1.setObjectName("label1")

        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(38, 200, 745, 380))
        self.label_2.setText("")
        self.label_2.setObjectName("label_2")
        self.label_2.setStyleSheet("background-color: white;\n"
                                   "border: 1px solid black;\n"
                                   "font-weight: bold;\n"
                                   "")
        self.verticalLayout.addWidget(self.label_2)

        # self.label3 = QtWidgets.QLabel(self.centralwidget)
        # self.label3.setGeometry(QtCore.QRect(60, 40, 41, 21))
        # font = QtGui.QFont()
        # font.setPointSize(17)
```

```
        # self.label3.setFont(font)
        # self.label3.setObjectName("label3")

        #wac
        # self.lineEdit1 = QtWidgets.QLineEdit(self.centralwidget)
        # self.lineEdit1.setGeometry(QtCore.QRect(60, 62, 91, 31))
        # self.lineEdit1.setObjectName("lineEdit1")

        # self.lineEdit2 = QtWidgets.QLineEdit(self.centralwidget)
        # self.lineEdit2.setGeometry(QtCore.QRect(170, 62, 91, 31))
        # self.lineEdit2.setObjectName("lineEdit2")

        self.dateEdit = QtWidgets.QDateEdit(self.scrollAreaWidgetContents)
        self.dateEdit.setGeometry(QtCore.QRect(87, 49, 95, 31))
        d = QDate.currentDate()
        self.dateEdit.setDate(d)
        self.dateEdit.setObjectName("dateEdit")

        # self.lineEdit3 = QtWidgets.QLineEdit(self.centralwidget)
        # self.lineEdit3.setGeometry(QtCore.QRect(280, 62, 91, 31))
        # self.lineEdit3.setObjectName("lineEdit3")

        self.lineEdit4 = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit4.setGeometry(QtCore.QRect(210, 62, 91, 31))
        self.lineEdit4.setObjectName("lineEdit4")

        self.lineEdit5 = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit5.setGeometry(QtCore.QRect(320, 62, 91, 31))
        self.lineEdit5.setObjectName("lineEdit5")

        #multi input
        self.tableWidget2 = QtWidgets.QTableWidget(self.
↪scrollAreaWidgetContents)
        self.tableWidget2.setGeometry(QtCore.QRect(210, 100, 250, 300))
        self.tableWidget2.setRowCount(10)
        self.tableWidget2.setColumnCount(2)
        self.tableWidget2.setHorizontalHeaderLabels(['Part', 'Program'])
        self.tableWidget2.setObjectName("tableWidget")
        self.tableWidget2.setStyleSheet("background-color: white;\n"
                                        "gridline-color: 1px solid black;\n")

        self.btn2 = QtWidgets.QPushButton(self.centralwidget)
        self.btn2.setGeometry(QtCore.QRect(480, 62, 91, 31))
        self.btn2.setStyleSheet("background-color: #E9967A;\n"
                                "border: black;\n"
                                "")
        self.btn2.setObjectName("ptn2")
```

```
        self.btn2.clicked.connect(self.upload)

        # self.btn3 = QtWidgets.QPushButton(self.centralwidget)
        # self.btn3.setGeometry(QtCore.QRect(650, 38 , 91, 20))
        # self.btn3.setStyleSheet("background-color: #E9967A;\n"
        #                         "border: black;\n"
        #                         "")
        # self.btn3.setObjectName("ptn3")
        # self.btn3.clicked.connect(self.quit)

        self.label4 = QtWidgets.QLabel(self.centralwidget)
        self.label4.setGeometry(QtCore.QRect(100, 40, 41, 21))
        font = QtGui.QFont()
        font.setPointSize(17)
        self.label4.setFont(font)
        self.label4.setObjectName("label4")

        # self.label5 = QtWidgets.QLabel(self.centralwidget)
        # self.label5.setGeometry(QtCore.QRect(280, 40, 51, 21))
        # font = QtGui.QFont()
        # font.setPointSize(17)
        # self.label5.setFont(font)
        # self.label5.setObjectName("label5")

        self.label6 = QtWidgets.QLabel(self.centralwidget)
        self.label6.setGeometry(QtCore.QRect(210, 40, 41, 21))
        font = QtGui.QFont()
        font.setPointSize(17)
        self.label6.setFont(font)
        self.label6.setObjectName("label6")

        self.label7 = QtWidgets.QLabel(self.centralwidget)
        self.label7.setGeometry(QtCore.QRect(320, 40, 80, 21))
        font = QtGui.QFont()
        font.setPointSize(17)
        self.label7.setFont(font)
        self.label7.setObjectName("label7")

        self.label8 = QtWidgets.QLabel(self.centralwidget)
        self.label8.setGeometry(QtCore.QRect(480, 40, 96, 21))
        font = QtGui.QFont()
        font.setPointSize(17)
        self.label8.setFont(font)
        self.label8.setObjectName("label8")

        self.out = pd.DataFrame()
```

```
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 22))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.btn1.setText(_translate("MainWindow", "Check"))
        self.btn2.setText(_translate("MainWindow", "Upload"))
        # self.btn3.setText(_translate("MainWindow", "Confirm"))
        self.label1.setText(_translate("MainWindow", "Apple Cost Test"))
        # self.label3.setText(_translate("MainWindow", "WAC"))
        self.label4.setText(_translate("MainWindow", "Date"))
        # self.label5.setText(_translate("MainWindow", "Month"))
        self.label6.setText(_translate("MainWindow", "Part"))
        self.label7.setText(_translate("MainWindow", "Program"))
        self.label8.setText(_translate("MainWindow", "Supplier Info"))

    def compare(self, price, year, month, part):
        self.indicate = 0
        # print(self.indicate)
        all_col = df[df.Part == part]
        prev_month = month - 1
        if prev_month == 0:  # currently 2021.1, prev_month = 2020.12
            indicator = '.2'
        elif prev_month == 11:  # currently 2020.12, prev_month = 2020.11
            indicator = '.1'
        elif prev_month == 10:  # currently 2020.11, prev_month = 2020.10
            indicator = ''
        else:  # currently ahead of 2021.1, means in the future, only compares␣
↪to latest data: 2021.1
            indicator = '.3'

        prev_month_price = all_col['WAC' + indicator].mean()
        # print(prev_month_price)
        ratio = price / prev_month_price
        if ratio >= 10:
            ratio = round(ratio)
        # print(ratio)
```

```
        if ratio == 0.1 or ratio == 0.01 or ratio == 10 or ratio == 100:
            # print(prev_month_price)
            self.indicate = 1
        return self.indicate

    def CheckAllocation(self, indicator, merge_buy, merge_sell, merge_allo):
        buy_df = pd.merge(merge_buy, merge_allo, how='outer', on='M Code')
        sell_df = pd.merge(merge_sell, merge_allo, how='outer', on='M Code')
        buy_df['BuyDelta'] = buy_df['buyprice'] - buy_df[
            'Buy $' + indicator]  # current buy price - prev month buy price
        buy_df['AlloDelta'] = buy_df['Allocation'] - buy_df['Allocation %' +␣
↪indicator]
        sell_df['SellDelta'] = sell_df['sellprice'] - sell_df[
            'Sell (PO) $' + indicator]  # current sell price - prev month sell␣
↪price
        sell_df['AlloDelta'] = sell_df['Allocation'] - sell_df['Allocation %' +␣
↪indicator]

        buy_df = buy_df.rename(
            columns={'Buy $' + indicator: 'LastMonth Buy', 'Allocation %' +␣
↪indicator: 'LastMonth Allo',
                     'buyprice': 'ThisMonth Buy', 'Allocation': 'ThisMonth␣
↪Allo'})
        diag_message = ''
        # print("A summary table of past month's data compared with this␣
↪month's data")
        # print(buy_df)
        more_expensive = False
        cheaper = False
        new_supplier = False
        for i in range(len(buy_df)):
            AlloDelta = buy_df.iloc[i, -1]
            BuyDelta = buy_df.iloc[i, -2]
            m_code = buy_df.iloc[i, 0]
            prev_buy = buy_df.iloc[i, 1]

            # first scenario: changed from cheaper to more expensive supplier
            if (BuyDelta > 0 and AlloDelta > 0) or (BuyDelta < 0 and AlloDelta␣
↪< 0):
                more_expensive = True
            # second scenario: changed from more expensive to cheaper supplier
            if (BuyDelta < 0 and AlloDelta > 0) or (BuyDelta > 0 and AlloDelta␣
↪< 0):
                cheaper = True
            # third scenario: new supplier
            if pd.isnull(prev_buy):
```

```
                new_supplier = True

        if more_expensive:
            # print()
            diag_message += "This month, we bought more from a more expensive␣
↪supplier compared to last month.\n"
        if cheaper:
            diag_message += "This month, we bought more from a cheaper supplier␣
↪compared to last month.\n"
        if new_supplier:
            diag_message += "This month, we bought from a new supplier.\n"

        return buy_df, diag_message

    def potential_reason(self, indicator, WAC, buyprice, sellprice, allo,␣
↪prev_allo, prev_price):
        reason_message = ''
        reason_message += '\n'
        reason_message += '\n==== Section 2: Potential Reasons Diagnosis␣
↪====================\n'
        prev_buyprice = prev_price[['M Code', 'Buy $' + indicator]]
        prev_sellprice = prev_price[['M Code', 'Sell (PO) $' + indicator]]

        # check for fat finger
        # reason_message += "\033[4m"+'2.1 Check for errors\n'+"\033[0m"
        fatfinger = dict()
        merge_buy = pd.merge(prev_buyprice, buyprice, how='outer', on='M Code')
        merge_sell = pd.merge(prev_sellprice, sellprice, on='M Code')

        for i in range(len(merge_buy)):
            ratio = merge_buy.iloc[i]['buyprice'] / merge_buy.iloc[i]['Buy $' +␣
↪indicator]
            supplier = merge_buy.iloc[i]['M Code']
            if ratio == 0.1 or ratio == 0.01 or ratio == 10 or ratio == 100:
                reason_message += '\n- Check for fat fingers\n'
                reason_message += 'For supplier: ' + str(
                    supplier) + ', the current buy price and previous month buy␣
↪price has a ' + str(
                    ratio) + ' times difference. Error?\n'
                fatfinger[supplier] = True
            else:
                fatfinger[supplier] = False
        reason_message += '\n- Check for allocation issues\n'
        prev_allo['Allocation %' + indicator] = prev_allo['Allocation %' +␣
↪indicator].apply(
            lambda x: float(x[:-1]) / 100)
```

```
        merge_allo = pd.merge(prev_allo, allo, how='outer', on='M Code')
        table, diag_message = self.CheckAllocation(indicator, merge_buy,␣
↪merge_sell, merge_allo)
        reason_message += diag_message
        # print(Error)
        return reason_message, table, fatfinger

    def CheckPrice(self, WAC, buyprice, sellprice, allo, year, month, part,␣
↪program):
        import datetime
        import dateutil.relativedelta
        import warnings
        warnings.filterwarnings("ignore")

        message = ''
        if part in df.Part.to_list():
            # message += 'The part is in the database.\n'
            all_col = df[(df.Part == part) & (df.Program == program)]

            currdate = datetime.date(year, month, 1)
            curr_month = datetime.datetime.strftime(currdate, "%Y%m")
            prev_date = currdate + dateutil.relativedelta.
↪relativedelta(months=-1)
            prev_month = datetime.datetime.strftime(prev_date, "%Y%m")
            if prev_month == '202012':  # currently 2021.1, prev_month = 2020.12
                indicator = '.2'
            elif prev_month == '202011':  # currently 2020.12, prev_month =␣
↪2020.11
                indicator = '.1'
            elif prev_month == '202010':  # currently 2020.11, prev_month =␣
↪2020.10
                indicator = ''
            elif prev_month < '202010':
                message += "The input was too early in time, database only has␣
↪earilest data from 2020 Oct"
                return message
            else:  # currently ahead of 2021.1, means in the future, only␣
↪compares to latest data: 2021.1
                indicator = '.3'

            prev_month_WAC = all_col['WAC' + indicator].mean()
            # for the same part, WAC should be the same but may have multiple␣
↪rows, so taking mean.
            # could also take mediam/min/max, I think should be the same thing

            if prev_month_WAC != WAC:
```

```
                message += '==== Section 1: WAC difference detection␣
↪======================\n'
                message += 'WAC NOT the same as last month. Previous month WAC␣
↪is ' + str(prev_month_WAC) + '. '
                message += 'This month WAC is ' + str(WAC) + '.\n'

                # investiagte the reason behind the difference
                # 1. check allocation
                prev_allo = all_col[['M Code', 'Allocation %' + indicator]]
                #             message +='\nPrevious month allocations are:\n'
                #             for i in range(len(prev_allo)):
                #                 message += 'M code: '+ prev_allo.iloc[i][0] +␣
↪', Allocation: ' + prev_allo.iloc[i][1] + "\n"

                # 2. check price
                prev_price = all_col[['M Code', 'Buy $' + indicator, 'Sell (PO)␣
↪$' + indicator]]
                #             message += "\nPrevious month prices (both buy and␣
↪sell) are:\n"
                #             for i in range(len(prev_price)):
                #                 message += 'M code: '+ prev_price.iloc[i][0]␣
↪+ ', Buy price: ' + str(prev_price.iloc[i][1]) + ', Sell price: ' +␣
↪str(prev_price.iloc[i][2]) + "\n"

                # check for potential reason
                reason_message, table, fatfinger = self.
↪potential_reason(indicator, WAC, buyprice=buyprice,
                                                                                ␣
↪sellprice=sellprice,
                                                                                ␣
↪prev_allo=prev_allo, allo=allo,
                                                                                ␣
↪prev_price=prev_price)
                print(fatfinger)
                #                 print('check table')
                #                 print(table)
                message += reason_message
                # build visualization for price change and allocation␣
↪change(side-by-side bar plot)
                message += '\n'
                message += "\n==== Section 3: Tables and Visualizations␣
↪======================"

                self.sc = MplCanvas(self, width=10, height=12, dpi=100)
                self.Visual(all_col, buyprice, sellprice, allo, curr_month)
                self.likelihood(table, fatfinger)
```

```
            else:
                message += "WAC same as last month!\n"
        else:
            message += "Don't have this part in the database"

        # return print(message,'\n', table)
        return message

    def Visual(self, all_col, buyprice, sellprice, allo, curr_month):
        import seaborn as sns

        # visualization for buy price
        buy_df = all_col['M Code']
        for col in all_col:
            if 'Buy $' in col:
                buy_df = pd.concat([buy_df, all_col[[col]]], axis=1)
        buy_df = buy_df.rename(
            columns={'Buy $': 202010, 'Buy $.1': 202011, 'Buy $.2': 202012,␣
↪'Buy $.3': 202101, 'Buy $.4': 202102,
                     'Buy $.5': 202103, 'Buy $.6': 202104, 'Buy $.7': 202105,␣
↪'Buy $.8': 202106})
        drop_col = []
        for col in buy_df:
            if isinstance(col, int):
                if col >= int(curr_month):
                    drop_col.append(col)
        buy_df = buy_df.drop(drop_col, axis=1)
        buy_df = pd.merge(buy_df, buyprice, how='outer', on='M Code')
        buy_df = buy_df.rename(columns={'buyprice': int(curr_month)})
        buy_df.set_index('M Code', inplace=True)

        # visualization for allocation
        allo_df = all_col['M Code']
        for col in all_col:
            if 'Allocation %' in col:
                allo_df = pd.concat([allo_df, all_col[[col]]], axis=1)
        allo_df.set_index('M Code', inplace=True)

        def change_allo_to_float(row):
            return float(row.replace('%', '')) / 100

        for col in allo_df:
            allo_df[col] = allo_df[col].apply(change_allo_to_float)

        allo_df = allo_df.rename(columns={'Allocation %': 202010, 'Allocation %.
↪1': 202011, 'Allocation %.2': 202012,
```

```
                                        'Allocation %.3': 202101, 'Allocation␣
↪%.4': 202102,
                                        'Allocation %.5': 202103, 'Allocation␣
↪%.6': 202104, 'Allocation %.7': 202105,
                                        'Allocation %.8': 202106})
        allo_df = allo_df.drop(drop_col, axis=1)
        allo_df = pd.merge(allo_df, allo, how='outer', on='M Code')
        allo_df = allo_df.rename(columns={'Allocation': int(curr_month)})

        # Plotting the grouped bar chart

        legend = list()
        master_list = list()
        labels = list(allo_df.columns)[1:]  # exclude the M code from the␣
↪xticklabels
        labels = [str(i) for i in labels]
        for i in range(len(allo_df)):
            legend.append(allo_df.iloc[i, 0])
            data = list(allo_df.loc[i].values)
            del data[0]  # exclude the M code
            master_list.append(data)

        x = np.arange(len(labels))
        width = 0.7 / len(legend)  # the width of the bars depends on how many␣
↪bars there are
        for i in range(len(allo_df)):
            self.sc.axes1.bar(x + width * i, master_list[i], width,␣
↪align="edge",
                              label=legend[i])  # align the bars properly (same␣
↪width and no overlap)

        self.sc.axes1.set_xticks([(2 * x + width * len(legend)) / 2 for x in
                                  range(len(labels))])  # align the xticks to␣
↪the centre regardless of no. of bars
        self.sc.axes1.set_xticklabels(labels)
        self.sc.axes1.legend()
        self.sc.axes1.set_xlabel("YYYYMM")
        self.sc.axes1.set_ylabel("Allocation")
        self.sc.axes1.set_title("Allocation by Month and M Code")

        # Plotting the line graph
        for i in range(len(buy_df)):
            row = buy_df.iloc[i]
            x = labels
            y = row.values
            self.sc.axes0.plot(x, y, label=row.name)
```

```
        self.sc.axes0.set_ylabel("Price")
        self.sc.axes0.set_title("Price by Month and M Code")
        self.sc.axes0.legend()

        self.verticalLayout.addWidget(self.sc)

    def likelihood(self, df, fatfinger):
        for i in range(len(df)):
            row = df.iloc[i]
            supplier = row['M Code']
            fatfinger_flag = fatfinger[supplier]
            if row['LastMonth Buy'] != 0 and row['LastMonth Allo'] != 0:
                buy_pct_change = row['BuyDelta'] / row['LastMonth Buy']
                allo_pct_change = row['AlloDelta'] / row['LastMonth Allo']
                this_month_total = row['ThisMonth Buy'] * row['ThisMonth Allo']
                last_month_total = row['LastMonth Buy'] * row['LastMonth Allo']

                if fatfinger_flag == True:
                    correct_likelihood = 1 - ((this_month_total -␣
↪last_month_total) / this_month_total)
                    rest_likelihood = 1 - correct_likelihood

                    fatfinger_perc = 0.5
                    fatfinger_likelihood = fatfinger_perc * rest_likelihood
                    buy_likelihood = rest_likelihood * (1 - fatfinger_perc) *␣
↪abs(buy_pct_change) / (
                                    abs(buy_pct_change) + abs(allo_pct_change))
                    allo_likelihood = rest_likelihood * 0.4 *␣
↪abs(allo_pct_change) / (
                                    abs(buy_pct_change) + abs(allo_pct_change))
                    likelihood = [correct_likelihood, buy_likelihood,␣
↪allo_likelihood, fatfinger_likelihood]
                    labels = ['Valid', 'BuyPrice Change', 'Allocation Change',␣
↪'Fatfinger']
                    colors = ['forestgreen', 'steelblue', 'orange', 'red']

                    self.sc.axes2.pie(likelihood, labels=labels, autopct='%1.
↪1f%%', shadow=True, startangle=90,colors=colors)
                    # ax1.axis('equal')  # Equal aspect ratio ensures that pie␣
↪is drawn as a circle.
                    self.sc.axes2.set_title(str("Supplier: " + supplier))
                    # break

                else:
                    count = 0
```

```
                    correct_likelihood = 1 - (abs(this_month_total -␣
↪last_month_total) / this_month_total)
                    # correct_likelihood = 1 - ((row['ThisMonth Buy'] -␣
↪row['LastMonth Buy']) / row['ThisMonth Buy'])
                    rest_likelihood = 1 - correct_likelihood
                    if correct_likelihood == 1:
                        # if count == 0:
                        self.sc.axes3.pie([correct_likelihood],␣
↪labels=['Valid'], autopct='%1.1f%%', shadow=True, startangle=90,
                                          colors=["forestgreen"])
                        self.sc.axes3.set_title(str(str("Supplier: " +␣
↪supplier)))
                            # count+=1


                    else:
                        buy_likelihood = rest_likelihood * abs(buy_pct_change) /
↪ (
                                          abs(buy_pct_change) + abs(allo_pct_change))
                        allo_likelihood = rest_likelihood *␣
↪abs(allo_pct_change) / (
                                          abs(buy_pct_change) + abs(allo_pct_change))
                        likelihood = [correct_likelihood]
                        labels = ['Valid']
                        colors = ['forestgreen']
                        if buy_likelihood != 0:
                            likelihood.append(buy_likelihood)
                            labels.append('BuyPrice Change')
                            colors.append('steelblue')
                        if allo_likelihood != 0:
                            likelihood.append(allo_likelihood)
                            labels.append('Allocation Change')
                            colors.append('orange')
                        #                       likelihood = [buy_likelihood,␣
↪allo_likelihood, correct_likelihood]
                        #                       labels = ['BuyPrice Change',␣
↪'Allocation Change', 'Valid']

                        self.sc.axes4.pie(likelihood, labels=labels,␣
↪autopct='%1.1f%%', colors=colors,
                                          shadow=True, startangle=90)
                        # ax1.axis('equal')  # Equal aspect ratio ensures that␣
↪pie is drawn as a circle.
                        self.sc.axes4.set_title(str("Supplier: " + supplier))
```

```
        self.verticalLayout.addWidget(self.sc)

    def upload(self):
        file = QFileDialog.getOpenFileName()
        self.path = file[0]
        # print(path)
        self.input = pd.read_csv(self.path)
        # # print(input)
        # buy = input['buyprice'].tolist()
        # sell = input['sellprice'].tolist()
        # allo = input['allocation'].tolist()
        # print(allo)
        # return path

    def popup(self):
        # price = self.lineEdit1.text()
        # year = self.lineEdit2.text()
        # month = self.lineEdit3.text()
        # program = self.lineEdit5.text()
        # part = self.lineEdit4.text()
        self.multi = pd.DataFrame()
        row_m = self.tableWidget2.rowCount()
        column_m = self.tableWidget2.columnCount()
        for r in range(row_m):
            temp = []
            for c in range(column_m):
                w = self.tableWidget2.item(r, c)
                if w:
                    temp.append(w.text())
            if temp == []:
                break
            else:
                self.multi = self.multi.append([temp])
        self.multi.columns = ['part', 'program']
        part = self.multi['part'].tolist()[0]
        program = self.multi['program'].tolist()[0]

        valuem = self.dateEdit.date().month()
        valuey = self.dateEdit.date().year()


        try:
            self.input = self.input[(self.input['Part'] == part) & (self.
↪input['Program'] == program)]

            if self.input.iloc[0,3] == 'N':
```

```
                self.input['sum'] = self.input['buyprice']*self.
↪input['allocation']
                wac = self.input['sum'].sum()
            else:
                wac = self.input['sellprice'].mean()
        except:
            self.df = pd.DataFrame()
            row = self.tableWidget.rowCount()
            column = self.tableWidget.columnCount()
            for r in range(row):
                temp = []
                for c in range(column):
                    w = self.tableWidget.item(r, c)
                    if w:
                        temp.append(w.text())
                if temp == []:
                    break
                else:
                    self.df = self.df.append([temp])
            self.df.columns = ['mcode', 'buyprice', 'sellprice',␣
↪'allocation','flag']
            self.df['buyprice'] = self.df['buyprice'].astype(float)
            self.df['sellprice'] = self.df['sellprice'].astype(float)
            self.df['allocation'] = self.df['allocation'].astype(float)
            self.df['sum'] = self.df['buyprice'] * self.df['allocation']
            if self.df.iloc[0,4] == 'N':
                wac = self.df['sum'].sum()
            else:
                wac = self.df['sellprice'].mean()

        i = self.compare(price=float(wac), year=int(valuey), month=int(valuem),␣
↪part=part)
        print(i)

        # if i == 1:
        #     self.lineEdit1.setStyleSheet("background-color: #E74C3C;\n"
        #                                  "")
        # else:
        #     self.lineEdit1.setStyleSheet("background-color: white;\n"
        #                                  "")
        msb = QMessageBox.question(self.centralwidget, 'Alert', "WAC higher␣
↪than expected, continue to check?",
                                   QMessageBox.Yes | QMessageBox.No,␣
↪QMessageBox.No)

        if msb == QMessageBox.Yes:
            self.click()
```

```
        else:
            # com = [part,program]
            # self.out = self.out.append([com])

            pass

    def quit(self):
        print(111)

    def click(self):
        # price = self.lineEdit1.text()

        # part = self.lineEdit4.text()
        # program = self.lineEdit5.text()

        self.multi = pd.DataFrame()
        row_m = self.tableWidget2.rowCount()
        column_m = self.tableWidget2.columnCount()
        for r in range(row_m):
            temp = []
            for c in range(column_m):
                w = self.tableWidget2.item(r, c)
                if w:
                    temp.append(w.text())
            if temp == []:
                break
            else:
                self.multi = self.multi.append([temp])
        self.multi.columns = ['part', 'program']
        print(self.multi)
        part = self.multi['part'].tolist()[0]
        program = self.multi['program'].tolist()[0]

        valuem = self.dateEdit.date().month()
        valuey = self.dateEdit.date().year()

        # if upload
        try:
            self.input = self.input[(self.input['Part'] == part) & (self.
↪input['Program'] == program)]

            if self.input.iloc[0,3] == 'N':
                self.input['sum'] = self.input['buyprice']*self.
↪input['allocation']
                wac = self.input['sum'].sum()
            else:
                wac = self.input['sellprice'].mean()
```

```
            buyprice = pd.DataFrame(
                                {'M Code': self.input['mcode'].tolist(), 'buyprice':
↪ self.input['buyprice'].tolist()})
            sellprice = pd.DataFrame(
                                {'M Code': self.input['mcode'].tolist(),␣
↪'sellprice': self.input['sellprice'].tolist()})
            allo = pd.DataFrame(
                                {'M Code': self.input['mcode'].tolist(),␣
↪'Allocation': self.input['allocation'].tolist()})
            # print(buyprice)
            # print(sellprice)
            # print(allo)
            # self.out.to_csv('/Users/estherko/Documents/apple/output.csv')
            # print(wac)

            self.label_2.setText(self.CheckPrice(WAC=float(wac),␣
↪buyprice=buyprice, sellprice=sellprice,
                                                  allo=allo,␣
↪year=int(valuey), month=int(valuem), part=part,program=program))

            # buyprice = pd.DataFrame(
            #     {'M Code': ['000049M', '004531M', '005173M', '005241M'],␣
↪'buyprice': [5.05, 0.550, 0.524, 0]})
            # sellprice = pd.DataFrame(
            #     {'M Code': ['000049M', '004531M', '005173M', '005241M'],␣
↪'sellprice': [0, 0, 0, 0]})
            # allo = pd.DataFrame(
            #     {'M Code': ['000049M', '004531M', '005173M', '005241M'],␣
↪'Allocation': [0.3,0.3, 0.4, 0]})
            # self.label_2.setText(
            #     self.CheckPrice(WAC=0.524, buyprice=buyprice,␣
↪sellprice=sellprice, allo=allo, year=2021, month=2,
            #                     part='631-05349', program='B298'))

        except:

            self.df = pd.DataFrame()

            row = self.tableWidget.rowCount()
            column = self.tableWidget.columnCount()
            for r in range(row):
                temp = []
                for c in range(column):
                    w = self.tableWidget.item(r, c)
                    if w:
```

```
                        temp.append(w.text())
                if temp == []:
                    break
                else:
                    self.df = self.df.append([temp])
            self.df.columns = ['mcode', 'buyprice', 'sellprice',␣
↪'allocation','flag']
            self.df['buyprice'] = self.df['buyprice'].astype(float)
            self.df['sellprice'] = self.df['sellprice'].astype(float)
            self.df['allocation'] = self.df['allocation'].astype(float)
            self.df['sum'] = self.df['buyprice'] * self.df['allocation']
            if self.df.iloc[0,4] == 'N':
                wac = self.df['sum'].sum()
            else:
                wac = self.df['sellprice'].mean()


            buyprice = pd.DataFrame(
                {'M Code': self.df['mcode'].tolist(), 'buyprice': self.
↪df['buyprice'].tolist()})
            sellprice = pd.DataFrame(
                {'M Code': self.df['mcode'].tolist(), 'sellprice': self.
↪df['sellprice'].tolist()})
            allo = pd.DataFrame(
                {'M Code': self.df['mcode'].tolist(), 'Allocation': self.
↪df['allocation'].tolist()})


            # print(allo)
            self.label_2.setText(self.CheckPrice(WAC=float(wac),␣
↪buyprice=buyprice, sellprice=sellprice,
                                                  allo=allo,␣
↪year=int(valuey), month=int(valuem), part=part,program=program))

        # print(self.CheckPrice(WAC=float(price), buyprice = buyprice,␣
↪sellprice = sellprice,
        #         allo = allo, year=int(year),month=int(month),part=part))

        # self.label_2.setText(self.input['mcode'].tolist()[0])
        # print(self.input['mcode'].tolist())


class MplCanvas(FigureCanvasQTAgg):
    def __init__(self, parent=None, width=5, height=7, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes0 = fig.add_subplot(234)
```

```
        self.axes1 = fig.add_subplot(235)
        self.axes2 = fig.add_subplot(231)
        self.axes3 = fig.add_subplot(232)
        self.axes4 = fig.add_subplot(233)
        super(MplCanvas, self).__init__(fig)


if __name__ == "__main__":
    import sys

    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```