FIT 5196 Assignment 2

- Name: Peiyu Liu
- Student number: 31153291

- Date: 20/05/2021

Libraries used:

- pandas(Process data, read and input csv files, convert and filter data.)
- re(Regular expression function, used for regular expression matching and replacement, to find matching objects.)
- linalg(The numpy.linalg module contains functions for linear algebra. Using this module, calculate the inverse matrix, find eigenvalues, solve linear equations, and solve determinants.)
- numpy(N-dimensional array object ndarray, dimensional array and matrix operations.)
- LinearRegression(A statistical analysis method to determine the quantitative relationship between two or more variables)
- matplotlib(Graphicalize the data and provide a variety of output formats.)
- math(Mathematical functions for floating-point numbers)

1. Introduction

- For assignment 2, there are three sample data files that all contain many issue data. For dirty file, all data should be corrected into correct formulas. For missing file, all rows which are lack accurate data should be removed. For the outlier file, all outlier data should be deleted. Finally, code can output three files that contain data accurate formula. Use pandas to read the CSV database and analyse files' procedure and data. Compare data with the correct formula, then use python functions to modify data. Use lambda to travelsal data and use condition to find expected data. Change data to accurate formular or detele issue data. Check each row and each column to ensure their data is accurate. Data can be performed as graphical and non-graphical to check their correctness.

details in code sections

2. Assignment coding

import libraries

- pandas, re, numpy, LinearRegression, matplotlib, math (see in Libraries)

In [ ]:

```
import pandas as pd
import re
from numpy import linalg as LA
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import math
```

- Input dirty data file and use desribe(), count() functions to check data format and file structure.

In [ ]:

```
dirty_file = pd.read_csv('31153291/31153291_dirty_data.csv')
```

In [ ]:

```
dirty_file.describe() # observe data structure
```

In [ ]:

```
dirty_file.count() # observe data numbers
```

check order id

- order id should be unique, set() can check repeated id.
- If id numbers is equal to unique id numbers, it means each id is unique.

In [ ]:

```
len(dirty_file.sales_id) == len(set(dirty_file.sales_id)) # result is true--no repeated id.
```

check date

- date fomart should be DD/MM/YYYY.
- use apply and lambda to traversal data.
- use regression '\W' to split each date into three parts.
- Use lambda to traversal each date and check date length is more than 3- day,month,year
- x[0] means day, x[1] means month, x[-1] means year
- month value should be less than 12, year's length should be more than 2, day's length should be less than 3.

Reference:

- apply() and lambda() https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html)
- pd.at() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.at.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.at.html)

In [ ]:

```
# check date
dirty_file.date = dirty_file.date.apply(lambda x: re.split('\W', x)) # split data into 3 parts.
update_container = []
update_container += dirty_file.date[dirty_file.date.apply(lambda x: len(x) > 3)].index.to_list()
dirty_file.date = dirty_file.date.apply(lambda x: [flag for flag in x[1:] if flag] if len(x) > 3 else [flag for flag in x]) # traversal each month data.
dirty_file.date[dirty_file.date.apply(lambda x: x[1] > '12')] # month data value should no more than 12.
```

In [ ]:

```
len(update_container)
```

In [ ]:

```
dirty_file.date[dirty_file.date.apply(lambda x: x[1] > '12')]
```

In [ ]:

```
dirty_file.date[dirty_file.date.apply(lambda x: len(x[0]) > 2)] # day's length should no more than 3.
```

In [ ]:

```
# Use iterrows to traversal each row. after compare each day,month and year, change their orders.
# x[0] means day, x[1] means month, x[-1] means year
# month value should be less than 12, year's length should be more than 2, day's length should be less than 3.
# Reference: pd.at() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.at.html

for each_index,row in dirty_file.iterrows():
    date_extract = row.date
    if len(date_extract[0]) > 2:
        dirty_file.at[each_index, 'date'] = [date_extract[-1], date_extract[1], date_extract[0]]
        update_container.append(each_index)
```

In [ ]:

```
# change each date into DD/MM/YYYY format.
dirty_file.date = dirty_file.date.apply(lambda x: '/'.join([flag for flag in x]))
```

check time

- Use lambda to traversal each row and split each time by ':'.
- time will be divided into 3 parts.
- accurate time format is hh:mm:ss.
- hh no more than 24, mm and ss no more than 60.

Reference:

- apply() and lambda() https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html)
- index.isin() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.isin.html?highlight=isin#pandas.Index.isin (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.isin.html?highlight=isin#pandas.Index.isin)

In [ ]:

```
dirty_file.time = dirty_file.time.apply(lambda x: [flag for flag in x.split(':')]) # traversal time and split time.
```

In [ ]:

```
dirty_file.time[dirty_file.time.apply(lambda x: x[0] >='24')] # x[0] is hour.
```

In [ ]:

```
dirty_file.time[dirty_file.time.apply(lambda x: x[1] >= '60' or x[2] >= '60')] # x[1] and x[2] are minutes and seconds.
```

In [ ]:

```
update_container += dirty_file.time[dirty_file.time.apply(lambda x: x[0] >= '24')].index.to_list() # extract issue time, hour > 24
dirty_file.time = dirty_file.time.apply(lambda x: [x[1], x[0], x[2]] if x[0] >= '24' else x) # traversal issue time and change 3 parts orders.
dirty_file.time[dirty_file.time.apply(lambda x: x[0] >= '24')]
dirty_file[dirty_file.index.isin(update_container)] # use isin to check is there still has issue time
```

In [ ]:

```
dirty_file.time = dirty_file.time.apply(lambda x: ":".join([flag for flag in x])) # re-format time into hh:mm:ss
```

check parcel_size

- parcel_size can only have three values: small,medium,large
- use unique to check values
- use lambda to traversal each data and find out issue data.
- use lambda to traversal and modify values into accurate formats.

Reference:

- apply() and lambda() https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html)
- unique() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.unique.html?highlight=unique#pandas.unique (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.unique.html?highlight=unique#pandas.unique)

In [ ]:

```
dirty_file.parcel_size.unique() # check unique values
update_container += dirty_file.parcel_size[dirty_file.parcel_size.apply(lambda x: x in ['L','S','M'])].index.to_list() # traversal and find out issue data.
dirty_file.parcel_size = dirty_file.parcel_size.apply(lambda x: 'large' if x == 'L' else x) # change value into accurate value.
dirty_file.parcel_size = dirty_file.parcel_size.apply(lambda x: 'medium' if x == 'M' else x)
dirty_file.parcel_size = dirty_file.parcel_size.apply(lambda x: 'small' if x == 'S' else x)
```

In [ ]:

```
dirty_file.parcel_size.unique() # check results
```

check latitude and longtitude

- use plot() to generate diagram to analyse geographical data.
- use abs() to compare absolute values of latitude and longitude.
- use loc to extract latitude and longitude data.
- change geographical data into accurate format.

Reference:

- plot(): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html?highlight=plot#pandas.DataFrame.plot (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html?highlight=plot#pandas.DataFrame.plot)
- pd.loc[]: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html?highlight=loc#pandas.DataFrame.loc (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html?highlight=loc#pandas.DataFrame.loc)
- plot latitude and longitude: https://jakevdp.github.io/PythonDataScienceHandbook/04.13-geographic-data-with-basemap.html (https://jakevdp.github.io/PythonDataScienceHandbook/04.13-geographic-data-with-basemap.html) & https://stackoverflow.com/questions/53233228/plot-latitude-longitude-from-csv-in-python-3-6 (https://stackoverflow.com/questions/53233228/plot-latitude-longitude-from-csv-in-python-3-6)

In [ ]:

```
dirty_file.plot(kind = 'scatter', x = 'Customer_lat', y = 'Customer_long')
```

In [ ]:

```
# plot latitude and longitude: https://jakevdp.github.io/PythonDataScienceHandbook/04.13-geographic-data-with-basemap.html  & https://stackoverflow.com/questions/53233228/plot-latitude-longitude-from-csv-in-python-3-6

for each_index, row in dirty_file.iterrows():
    latitude_data,longitude_data = row.Customer_lat, row.Customer_long
    if abs(latitude_data) > abs(longitude_data):
        dirty_file.loc[each_index, 'Customer_lat'] = -1*abs(longitude_data)
        dirty_file.loc[each_index, 'Customer_long'] = abs(latitude_data)
        update_container.append(each_index)
```

In [ ]:

```
dirty_file.plot(kind = 'scatter', x = 'Customer_lat', y = 'Customer_long')
```

check loyalty

- only have value 0 or 1
- use unique to check value

In [ ]:

```
dirty_file.isLoyaltyProgram.unique() # results are 1 and 0. no issue.
```

check storehouse

- storehouse has unique id: 2,4,3,0,6.
- use dictionary and zip() to group storehouse name and geographical data.
- use unique to check values.
- 2 is Footscray.
- 0 is Clayton.
- 4 is Sunshine.
- 3 is St. Kilda.

Reference:

- to_numpy():https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html?highlight=to_numpy#pandas.DataFrame.to_numpy (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html?highlight=to_numpy#pandas.DataFrame.to_numpy) & https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.to_numpy.html?highlight=to_numpy#pandas.Index.to_numpy (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.to_numpy.html?highlight=to_numpy#pandas.Index.to_numpy)
- loc:https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html?highlight=loc#pandas.DataFrame.loc (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html?highlight=loc#pandas.DataFrame.loc)

In [ ]:

```
stores_data = pd.read_csv('Storehouses.csv')
```

In [ ]:

```
stores_data.describe() # analyse storehouse structure
```

In [ ]:

```
# dic_house: {Name: lat, lon}
stores_dict = dict(zip(stores_data.names.to_numpy(), stores_data[['lat','lon']].to_numpy()))
```

In [ ]:

```
stores_dict
```

In [ ]:

```
dirty_file.nearest_storehouse_id.unique()
```

In [ ]:

```
dirty_file[dirty_file.nearest_storehouse == 'Eping']
```

In [ ]:

```
# id: 2,4,3,0 only has one value
#  2 is Footscray.
# 0 is Clayton.
# 4 is Sunshine.
# 3 is St. Kilda
dirty_file[dirty_file.nearest_storehouse_id == 2].nearest_storehouse.unique()
```

In [ ]:

```
dirty_file[dirty_file.nearest_storehouse_id == 0].nearest_storehouse.unique()
```

In [ ]:

```
dirty_file[dirty_file.nearest_storehouse_id == 4].nearest_storehouse.unique()
```

In [ ]:

```
dirty_file[dirty_file.nearest_storehouse_id == 3].nearest_storehouse.unique()
```

In [ ]:

```
# 6 has multiple values.
dirty_file[dirty_file.nearest_storehouse_id == 6].nearest_storehouse.unique()
```

In [ ]:

```
stores_data.names.tolist()
```

In [ ]:

```
dirty_file[dirty_file.nearest_storehouse_id == 6]['nearest_storehouse'].count()
```

In [ ]:

```
# ['Clayton', 'Eping', 'Footscray', 'St. Kilda', 'Sunshine']
house_names = stores_data.names.tolist()
for each_index, row in dirty_file.iterrows():
    if row.nearest_storehouse_id == 6:
        dirty_file.loc[each_index, 'nearest_storehouse_id'] == house_names.index(row.nearest_storehouse)
        update_container.append(each_index)
```

check dist_to_nearest_storehouse

- The distance between two points is calculated, the position of the point on the earth is in the form of latitude and longitude coordinates.
- earth is 6371
- use radians to calculate difference between range of latitude, Manually convert numbers to radians.
- use sin and cos to calculate difference.

Reference:

- Getting distance between two points based on latitude/longitude: https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-longitude (https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-longitude)
- Geographiclib: https://geographiclib.sourceforge.io/html/python/code.html#module-geographiclib.geodesic (https://geographiclib.sourceforge.io/html/python/code.html#module-geographiclib.geodesic)

In [ ]:

```
# https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-longitude
def dist_difference(lat1, lon1, lat2, lon2):
    r = 6371
    dlat = math.radians(lat2 - lat1) # Manually convert numbers to radians
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2) * math.sin(dlat / 2) + math.cos(math.radians(lat1)) * math.cos(math.radians(lat2
)) \
        * math.sin(dlon / 2) * math.sin(dlon / 2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    d = r * c
    return round(d, 2)

# compare data and calculated results, if equal, then correct.
for each_index, row in dirty_file.iterrows():
    if each_index in update_container:
        continue
    name_store = row.nearest_storehouse
    dist_store = row.dist_to_nearest_storehouse
    lat1, lon1 = row.Customer_lat, row.Customer_long
    lat2, lon2 = stores_dict[name_store]
    dist = dist_difference(lat1, lon1, lat2, lon2)
    if dist != dist_store:
        dirty_file.loc[each_index, 'dist_to_nearest_storehouse'] = dist
        update_container.append(each_index)
```

check price and shopping_cart

- analyse shopping_cart data structure.
- find all products and calculate products' numbers.
- use dictionary to zip products' names and numbers.
- change dictionary to dataframe format.

Reference:

- array: https://numpy.org/doc/stable/reference/generated/numpy.array.html?highlight=array#numpy.array (https://numpy.org/doc/stable/reference/generated/numpy.array.html?highlight=array#numpy.array)
- matrix: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html?highlight=matrix#numpy.matrix (https://numpy.org/doc/stable/reference/generated/numpy.matrix.html?highlight=matrix#numpy.matrix)
- linalg.solve():https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html#numpy.linalg.solve (https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html#numpy.linalg.solve)

In [ ]:

```
cart_sample = dirty_file.shopping_cart.values[0]
cart_sample
product_data = dirty_file.shopping_cart.apply(lambda x: re.findall(r'\w+', x)).sum()
product_data = list(set([flag for flag in product_data if len(flag) > 2]))
product_dict = dict(zip(product_data, [[0 for _ in range(len(product_data))] for _ in range(len(product_dat
a))]))
product_number = pd.DataFrame(product_dict)
```

In [ ]:

```
# get each item price
product_container = []
price_cal = 0
for each_index, row in dirty_file.iterrows():
    pro_list = re.findall(r'\w+',row.shopping_cart)
    for i in range(0,len(pro_list),2):
        product_number.loc[each_index,pro_list[i]] = int(pro_list[i+1])
    product_container.append(row.price)
    if each_index >= 7:
        break
        # Solve linear matrices in matrix form
        # https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html
price_list = np.linalg.solve(np.matrix(product_number), np.array(product_container))
price_list
```

In [ ]:

```
# use dictionary to group products and single price
dict_each_price = dict(zip(product_data,[round(flag,4) for flag in price_list]))
dict_each_price
```

In [ ]:

```
# use product numbers and single price to calculate amount.
def price_compare(shopping_cart,dict_each_price = dict_each_price):
    price = 0
    pro_list = re.findall(r'\w+',shopping_cart)
    for i in range(0,len(pro_list),2):
        price += dict_each_price[pro_list[i]]*int(pro_list[i+1]) # prince = single_price*numbers
    return round(price,2) # price format xx.xx
```

In [ ]:

```
# check results and price, same is accurate price.
price_compare(row.shopping_cart), row.price
```

In [ ]:

```
dirty_file[dirty_file.shopping_cart.apply(lambda x: price_compare(x)) == dirty_file.price]
```

In [ ]:

```
len(update_container)
```

delivery cost

- no anomaly

Missing values

In missing file, some rows have empty data, input this row.

- use isna() to detect missing value.
- month 12,1,2 is one season.
- month 3,4,5 is one season.
- month 6,7,8 is one season.
- month 9,10,11 is one season.
- generate model (x,y) contain distance, season and cost.
- use linear regression to fit this model.
- Return the coefficient of determination of the prediction.
- Predict using the linear model.

Reference:

- isna() :https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isna.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isna.html)
- dropna(): drop missing data https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html)
- linearRegression:https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
- fit(): Fit linear model.https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
- reshape():Gives a new shape to an array without changing its data.https://numpy.org/doc/stable/reference/generated/numpy.reshape.html (https://numpy.org/doc/stable/reference/generated/numpy.reshape.html)

read file

- pd.read_csv()
- observe data structure

In [ ]:

```
missing_file = pd.read_csv('31153291/31153291_missing_data.csv')
missing_file.head()
missing_file.count()
```

In [ ]:

```
missing_file[missing_file.shopping_cart.apply(lambda x: price_compare(x)) != missing_file.price]
missing_file.price = missing_file.shopping_cart.apply(lambda x: price_compare(x))
```

In [ ]:

```
# check storehouse values.
missing_file.nearest_storehouse.unique()
```

In [ ]:

```
# Detect missing values.
# Reference: isna():https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isna.html
missing_file[missing_file.delivery_cost.isna()].nearest_storehouse.unique()
missing_file[(missing_file.delivery_cost.isna()) & (missing_file.nearest_storehouse == 'Sunshine')].date.apply(lambda x: x.split('/')[1]).unique()
missing_file[(missing_file.delivery_cost.isna()) & (missing_file.nearest_storehouse == 'Footscray')].date.apply(lambda x: x.split('/')[1]).unique()
missing_file[(missing_file.delivery_cost.isna()) & (missing_file.nearest_storehouse == 'St. Kilda')].date.apply(lambda x: x.split('/')[1]).unique()
missing_file[(missing_file.delivery_cost.isna()) & (missing_file.nearest_storehouse == 'Clayton')].date.apply(lambda x: x.split('/')[1]).unique()
```

In [ ]:

```
# missing_file[missing_file.nearest_storehouse == 'Clayton']
```

In [ ]:

```
missing_file['season'] = np.nan # Special values defined in numpy: nan
```

In [ ]:

```
# month 12,1,2 is one season.
# month 3,4,5 is one season.
# month 6,7,8 is one season.
# month 9,10,11 is one season.
# x[1] is month index.
missing_file.loc[missing_file.date.apply(lambda x: x.split('/')[1] in ['12', '01', '02']), 'season'] = 1
missing_file.loc[missing_file.date.apply(lambda x: x.split('/')[1] in ['03', '04', '05']), 'season'] = 2
missing_file.loc[missing_file.date.apply(lambda x: x.split('/')[1] in ['06', '07', '08']), 'season'] = 3
missing_file.loc[missing_file.date.apply(lambda x: x.split('/')[1] in ['09', '10', '11']), 'season'] = 4
```

In [ ]:

```
# drop missing data
missing_drop = missing_file.dropna()
sun_data = missing_drop[missing_drop.nearest_storehouse == 'Sunshine']
foot_data = missing_drop[missing_drop.nearest_storehouse == 'Footscray']
stkilda_data = missing_drop[missing_drop.nearest_storehouse == 'St. Kilda']
clayton_data = missing_drop[missing_drop.nearest_storehouse == 'Clayton']
```

In [ ]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
# generate model (x,y) contain distance, season and cost.
# use linear regression to fit this model.
# Return the coefficient of determination  of the prediction.
# Predict using the linear model.
# Sunshine
x = sun_data[['dist_to_nearest_storehouse', 'season']]
y = sun_data.delivery_cost
reg_sunshine = LinearRegression().fit(x,y) # fit modle x,y
reg_sunshine.score(x,y)
```

In [ ]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
# generate model (x,y) contain distance, season and cost.
# use linear regression to fit this model.
# Return the coefficient of determination  of the prediction.
# Predict using the linear model.
# Footscray
x = foot_data[['dist_to_nearest_storehouse', 'season']]
y = foot_data.delivery_cost
reg_foot = LinearRegression().fit(x,y)
reg_foot.score(x,y)
```

In [ ]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
# generate model (x,y) contain distance, season and cost.
# use linear regression to fit this model.
# Return the coefficient of determination  of the prediction.
# Predict using the linear model.
# St. Kilda
x = stkilda_data[['dist_to_nearest_storehouse', 'season']]
y = stkilda_data.delivery_cost
reg_stkilda = LinearRegression().fit(x,y)
reg_stkilda.score(x,y)
```

In [ ]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
# generate model (x,y) contain distance, season and cost.
# use linear regression to fit this model.
# Return the coefficient of determination  of the prediction.
# Predict using the linear model.
# Clayton
x = clayton_data[['dist_to_nearest_storehouse', 'season']]
y = clayton_data.delivery_cost
reg_clayton = LinearRegression().fit(x,y)
reg_clayton.score(x,y)
```

In [ ]:

```
missing_container = missing_file[missing_file.delivery_cost.isna()].index.tolist() # detect missing value
missing_container # result is empty, removed all missing value.
```

In [ ]:

```
# # https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
# predict Predict using the linear model.
for each_index, row in missing_file.iterrows():
    if each_index not in missing_container:
        continue
    x = np.array(row[['dist_to_nearest_storehouse', 'season']]).reshape(1,-1)
    if row.nearest_storehouse == 'Sunshine':
        cost_delivery = reg_sunshine.predict(x)[0]
    elif row.nearest_storehouse == 'Footscray':
        cost_delivery = reg_foot.predict(x)[0]
    elif row.nearest_storehouse == 'St. Kilda':
        cost_delivery = reg_stkilda.predict(x)[0]
    else:
        cost_delivery = reg_clayton.predict(x)[0]

    missing_file.loc[each_index, 'delivery_cost'] = cost_delivery
```

In [ ]:

```
missing_file.count()
```

In [ ]:

```
np.array(row[['dist_to_nearest_storehouse', 'season']]).reshape(1,-1)
```

In [ ]:

```
missing_file.head()
```

Outliers

- remove all outlier data
- use boxplot to draw boxplot digram to analyse data.

Reference:

- boxplot: https://seaborn.pydata.org/generated/seaborn.boxplot.html (https://seaborn.pydata.org/generated/seaborn.boxplot.html) & https://jwalton.info/Matplotlib-custom-boxplots/ (https://jwalton.info/Matplotlib-custom-boxplots/)
- percentile: https://numpy.org/doc/stable/reference/generated/numpy.percentile.html?highlight=percentile#numpy.percentile (https://numpy.org/doc/stable/reference/generated/numpy.percentile.html?highlight=percentile#numpy.percentile)

In [ ]:

```
outlier_file = pd.read_csv('31153291/31153291_outlier_data.csv')
```

In [ ]:

```
outlier_file.head()
```

In [ ]:

```
outlier_file.describe()
```

In [ ]:

```
outlier_file.shape
```

In [ ]:

```
outlier_file.boxplot()
```

In [ ]:

```
outlier_file.boxplot(column = 'price')
```

In [ ]:

```
# calculate upper and lower bond in boxplot.
# the most extreme data point within (75%-25%)data range.
# https://stackoverflow.com/questions/17725927/boxplots-in-matplotlib-markers-and-outliers
def outlier_bond(data):
    part_1 = np.percentile(data,25)
    part_2 = np.percentile(data,75)
    Interquartile = part_2-part_1
    bottom = part_1 - 1.5*Interquartile
    top = part_2 + 1.5*Interquartile
    return bottom, top
```

In [ ]:

```
bottom_price, top_price = outlier_bond(outlier_file.price)
```

In [ ]:

```
# price outliers
# compare price and (max and min range)
outlier_file = outlier_file[(outlier_file.price >= bottom_price) & (outlier_file.price <= top_price)]
outlier_file.boxplot(column = 'price')
```

In [ ]:

```
outlier_file.boxplot(column = 'dist_to_nearest_storehouse')
```

In [ ]:

```
outlier_file.boxplot(column = 'delivery_cost')
```

In [ ]:

```
plt.hist(outlier_file.delivery_cost)
plt.xlabel('Delivery cost')
plt.ylabel('Frequency')
plt.title('Delivery cost diagram')
plt.show()
```

In [ ]:

```
bottom_delivery, top_delivery = outlier_bond(outlier_file.delivery_cost)
bottom_delivery,top_delivery
```

In [ ]:

```
outlier_file = outlier_file[(outlier_file.delivery_cost >= 0)& (outlier_file.delivery_cost <= top_delivery)]
```

In [ ]:

```
outlier_file.boxplot(column = 'delivery_cost')
```

In [ ]:

```
outlier_file.shape
```

In [ ]:

```
outlier_file.to_csv('31153291_outlier_data_solution.csv', index = False)
```

In [ ]:

```
missing_file.to_csv('31153291_missing_data_solution.csv', index = False)
```

In [ ]:

```
dirty_file.to_csv('31153291_dirty_data_solution.csv', index = False)
```

3. Summary

When I do assignment 2, I observe data and its format in a dirty file, missing file and outlier file. Before I code these three tasks, I understand what issues in these sample files and what is the correct format. Then I search relevant questions and teaching material online. I set my coding logic, and then I start with coding. When I do this assignment, I faced many challenges, like I do not know how to calculate the difference between latitude and longitude, so I search solution online and use this solution by retrieve reference. Assignment 2 let me learn many data frames and NumPy packages. I learned many convenient methods to deal with raw data.

4. Reference

- apply() and lambda() https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html)
- pd.at() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.at.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.at.html)
- apply() and lambda() https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html)
- unique() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.unique.html?highlight=unique#pandas.unique (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.unique.html?highlight=unique#pandas.unique)
- plot(): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html?highlight=plot#pandas.DataFrame.plot (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html?highlight=plot#pandas.DataFrame.plot)
- pd.loc[]: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html?highlight=loc#pandas.DataFrame.loc (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html?highlight=loc#pandas.DataFrame.loc)
- plot latitude and longitude: https://jakevdp.github.io/PythonDataScienceHandbook/04.13-geographic-data-with-basemap.html (https://jakevdp.github.io/PythonDataScienceHandbook/04.13-geographic-data-with-basemap.html) & https://stackoverflow.com/questions/53233228/plot-latitude-longitude-from-csv-in-python-3-6 (https://stackoverflow.com/questions/53233228/plot-latitude-longitude-from-csv-in-python-3-6)
- to_numpy():https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html?highlight=to_numpy#pandas.DataFrame.to_numpy (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html?highlight=to_numpy#pandas.DataFrame.to_numpy) & https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.to_numpy.html?highlight=to_numpy#pandas.Index.to_numpy (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.to_numpy.html?highlight=to_numpy#pandas.Index.to_numpy)
- loc:https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html?highlight=loc#pandas.DataFrame.loc (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html?highlight=loc#pandas.DataFrame.loc)
- Getting distance between two points based on latitude/longitude: https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-longitude (https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-longitude)
- Geographiclib: https://geographiclib.sourceforge.io/html/python/code.html#module-geographiclib.geodesic (https://geographiclib.sourceforge.io/html/python/code.html#module-geographiclib.geodesic)
- array: https://numpy.org/doc/stable/reference/generated/numpy.array.html?highlight=array#numpy.array (https://numpy.org/doc/stable/reference/generated/numpy.array.html?highlight=array#numpy.array)
- matrix: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html?highlight=matrix#numpy.matrix (https://numpy.org/doc/stable/reference/generated/numpy.matrix.html?highlight=matrix#numpy.matrix)
- linalg.solve():https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html#numpy.linalg.solve (https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html#numpy.linalg.solve)
- isna() :https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isna.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isna.html)

- dropna(): drop missing data https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html)
- linearRegression:https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
- fit(): Fit linear model.https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
- reshape():Gives a new shape to an array without changing its data.https://numpy.org/doc/stable/reference/generated/numpy.reshape.html (https://numpy.org/doc/stable/reference/generated/numpy.reshape.html)
- boxplot: https://seaborn.pydata.org/generated/seaborn.boxplot.html (https://seaborn.pydata.org/generated/seaborn.boxplot.html) & https://jwalton.info/Matplotlib-custom-boxplots/ (https://jwalton.info/Matplotlib-custom-boxplots/)
- percentile: https://numpy.org/doc/stable/reference/generated/numpy.percentile.html?highlight=percentile#numpy.percentile (https://numpy.org/doc/stable/reference/generated/numpy.percentile.html?highlight=percentile#numpy.percentile)