Data Cleaning Phase II

In this notebook, you will do the following:

- Section 1: Data Cleaning
- Section 2: Data Quality and Testing
- Section 3: Work Flow Model
- Section 4: Conclusion

Section 1: Data Cleaning Steps

Data cleaning for this project was done with two tools, OpenRefine and Python. We used OpenRefine to trim whitespace and do data type conversions. We use Python for key constraints, empty/null values, outliers, and normalization.

In [12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img
import seaborn as sns
plt.style.use('seaborn')
import os
import json
```

OpenRefine: Dish dataset cleaning

Step 1.1

Description: We trim the whitespace from the dish name and we convert the prices to number values.

Rationale: Trimming the dish name whitespace will allow us to more accurately find duplicate dishes and better track prices over time. Converting prices to numbers will help us run calculations that can't be run on a string.

In [13]:
```
"""[
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "name",
    "expression": "value.trim()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column name using expression value.trim()
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
```

```
    "columnName": "lowest_price",
    "expression": "value.toNumber()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column lowest_price using expression valu
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "highest_price",
    "expression": "value.toNumber()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column highest_price using expression val
  }
]"""
```

Out[13]:


```
'[\n  {\n    "op": "core/text-transform",\n    "engineConfig": {\n      "facets": [],\n
     "mode": "row-based"\n    },\n    "columnName": "name",\n    "expression": "value.tr
im()",\n    "onError": "keep-original",\n    "repeat": false,\n    "repeatCount": 10,\n
   "description": "Text transform on cells in column name using expression value.trim
()"\n  },\n  {\n    "op": "core/text-transform",\n    "engineConfig": {\n      "facets":
[],\n      "mode": "row-based"\n    },\n    "columnName": "lowest_price",\n    "expressi
on": "value.toNumber()",\n    "onError": "keep-original",\n    "repeat": false,\n    "re
peatCount": 10,\n    "description": "Text transform on cells in column lowest_price usin
g expression value.toNumber()"\n  },\n  {\n    "op": "core/text-transform",\n    "engine
Config": {\n      "facets": [],\n      "mode": "row-based"\n    },\n    "columnName": "h
ighest_price",\n    "expression": "value.toNumber()",\n    "onError": "keep-original",\n
    "repeat": false,\n    "repeatCount": 10,\n    "description": "Text transform on cell
s in column highest_price using expression value.toNumber()"\n  }\n]'
```


OpenRefine: Menu Item Cleaning

Step 1.2

Description: We convert the prices to number values and we trim off the timestamp and convert the datestring to a date type.

Rationale: Converting prices and dates will help us run calculations that can't be run on a string. We trim times from menu item dates so that we can assess dish price with less granularity. This way we can group menu items by their date instead of having menu items all at different times.

In [14]:

```
"""[
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "price",
    "expression": "value.toNumber()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column price using expression value.toNum
  },
  {
    "op": "core/text-transform",
```

```
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "high_price",
    "expression": "value.toNumber()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column high_price using expression value.
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "created_at",
    "expression": "grel:value[0,10]",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column created_at using expression grel:v
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "updated_at",
    "expression": "grel:value[0,10]",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column updated_at using expression grel:v
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "created_at",
    "expression": "value.toDate()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column created_at using expression value.
  },
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "updated_at",
    "expression": "value.toDate()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column updated_at using expression value.
  }
]"""
```

```
'[\n  {\n    "op": "core/text-transform",\n    "engineConfig": {\n      "facets": [],\n
```

Out[14]:   "mode": "row-based"\n   },\n   "columnName": "price",\n   "expression": "value.toNumber()",\n   "onError": "keep-original",\n   "repeat": false,\n   "repeatCount": 10,\n   "description": "Text transform on cells in column price using expression value.toNumber()"\n },\n {\n   "op": "core/text-transform",\n   "engineConfig": {\n   "facets": [],\n   "mode": "row-based"\n   },\n   "columnName": "high_price",\n   "expression": "value.toNumber()",\n   "onError": "keep-original",\n   "repeat": false,\n   "repeatCount": 10,\n   "description": "Text transform on cells in column high_price using expression value.toNumber()"\n },\n {\n   "op": "core/text-transform",\n   "engineConfig": {\n   "facets": [],\n   "mode": "row-based"\n   },\n   "columnName": "created_at",\n   "expression": "grel:value[0,10]",\n   "onError": "keep-original",\n   "repeat": false,\n   "repeatCount": 10,\n   "description": "Text transform on cells in column created_at using expression grel:value[0,10]"\n },\n {\n   "op": "core/text-transform",\n   "engineConfig": {\n   "facets": [],\n   "mode": "row-based"\n   },\n   "columnName": "updated_at",\n   "expression": "grel:value[0,10]",\n   "onError": "keep-original",\n   "repeat": false,\n   "repeatCount": 10,\n   "description": "Text transform on cells in column updated_at using expression grel:value[0,10]"\n },\n {\n   "op": "core/text-transform",\n   "engineConfig": {\n   "facets": [],\n   "mode": "row-based"\n   },\n   "columnName": "created_at",\n   "expression": "value.toDate()",\n   "onError": "keep-original",\n   "repeat": false,\n   "repeatCount": 10,\n   "description": "Text transform on cells in column created_at using expression value.toDate()"\n },\n {\n   "op": "core/text-transform",\n   "engineConfig": {\n   "facets": [],\n   "mode": "row-based"\n   },\n   "columnName": "updated_at",\n   "expression": "value.toDate()",\n   "onError": "keep-original",\n   "repeat": false,\n   "repeatCount": 10,\n   "description": "Text transform on cells in column updated_at using expression value.toDate()"\n }\n]'

Python: Data Cleaning

Step 1.3

Description: First we import the libraries we will be using and the datasets that we will be cleaning.

@BEGIN main @PARAM file_path @IN dish_data @URI file:{file_path}/open_refine_cleaned/Dish.csv @IN menu_item_data @URI file:{file_path}/open_refine_cleaned/Menu-Item.csv @OUT cleaned_dish_data @URI file:{file_path}/dishcleaned.csv @OUT cleaned_menu_item_data @URI file:{file_path}/menu_item_cleaned.csv

In [15]:
```
import pandas as pd
import pytest

dish_df = pd.read_csv("../open_refine_cleaned/Dish.csv")
menu_item_df = pd.read_csv("../open_refine_cleaned/Menu-Item.csv")
```

Step 1.4

Description: We drop the rows that have an empty/null value for "price" and "created_at" columns.

Rationale: Our use case deals with comparing menu item prices throughout time, we can't analyze menu items if the dates or prices of that item are null. Since we have enough data even when removing the empty values we decided to remove all of the empty values.

@BEGIN CleanPrice @PARAM file_path @IN Price @AS menu_item_data @URI file:{file_path}/open_refine_cleaned/Menu-Item.csv @OUT data @AS menu_item_clean_price

In [16]:
```
menu_item_df.dropna(subset=['price'], inplace=True)
menu_item_df.dropna(subset=['created_at'], inplace=True)
```

@END CleanPrice

@BEGIN DishTitleCase @PARAM file_path @IN title @AS dish_data @URI file: {file_path}/open_refine_cleaned/Dish.csv @OUT data @AS dish_title_case

Step 1.5

Description: Convert the dish names to title case.

Rationale: Similar to trimming the dish names, converting the dish names to title case will allow us to more accurately detect duplipcates and merge them for better price tracking.

In [17]:
```
dish_df["name"] = dish_df["name"].str.title()
```

@END DishTitleCase

Step 1.6

Description: Find all dishes with the same name and grouping them together. Setting all the duplicate dish menu items to the first dish id. Removing all but the first dishes from the dish data set.

Rationale: Many dish names are similar/duplicates of each other, so they may be referring to the same food item, but under different dish_ids. We find these duplicates and merge them allowing us to have more data points per dish.

@BEGIN RemoveDuplicateDishes @IN dish @AS dish_title_case @OUT data @AS dish_unique

In [18]:
```
ids = dish_df["name"]
duped_name = dish_df[ids.isin(ids[ids.duplicated()])].sort_values("name")
duped_ids = duped_name.groupby(['name'])['id'].apply(lambda x: ','.join([str(y) for y in
ids_to_drop = []

for row in duped_ids.iterrows():
    dish_ids = row[1]["id"].split(",")
    first = int(dish_ids[0])
    for id in dish_ids[1:]:
        ids_to_drop.append(int(id))
        menu_item_df.loc[menu_item_df["dish_id"] == int(id), "dish_id"] = first

dish_df = dish_df[~dish_df['id'].isin(ids_to_drop)]
```

@END RemoveDuplicateDishes

Step 1.7

Description: Removing menu items that have prices outside the 10th-90th percentile.

Rationale: Removing outliers will allow us to more accurately analyze the changes in price over time. If a handful of dishes skyrocket in price, but they were a small part of all dishes, then that could skew the analysis. Therefore we removed all data points where the price was outside of the 10th-90th percentile.

@BEGIN RemoveOutliers @IN Price @AS menu_item_clean_price @OUT data @AS menu_item_valid_price

In [19]:
```
q_low = menu_item_df["price"].quantile(0.10)
q_hi  = menu_item_df["price"].quantile(0.90)
iqr = q_hi - q_low
mul = 2.0
```

```
menu_item_df = menu_item_df[(menu_item_df["price"] < q_hi + mul * iqr) & (menu_item_df["
```

@END RemoveOutliers

Step 1.8

Description: Use min-max normalization for the menu item prices.

Rationale: Normalizing the price data isn't absolutely necessary for this investigation, but will help keep our data uniform and easy to read and understand.

@BEGIN NormalizePrice @IN Price @AS menu_item_valid_price @OUT data @AS menu_item_normalized_price

In [20]:
```
menu_item_df["price"] = (menu_item_df["price"] - menu_item_df["price"].min()) / (menu_it
```

@END NormalizePrice

Step 1.9

Description: Removing menu items that don't have a particular dish associated with it and removing dishes that have no menu items associated with it.

Rationale: If a menu item doesn't have an associated dish in the dish dataset, then we don't want to consider it since that may be an invalid entry or it may not be possible to track its price over time.

@BEGIN RemoveOrphanDishes @IN dish @AS dish_unique @OUT data @AS dish_valid

In [21]:
```
dish_df = dish_df[dish_df['id'].isin(menu_item_df["dish_id"])]
menu_item_df = menu_item_df[menu_item_df['dish_id'].isin(dish_df["id"])]
```

@END RemoveOrphanDishes

@BEGIN RemoveOrphanMenuItems @IN menu @AS menu_item_normalized_price @OUT data @AS menu_items_valid @END RemoveOrphanMenuItems

Step 1.10

Description: Standardizing the created_at into ISO format.

Rationale: In order to analyze the dish prices over time, we need a standard date format for the created_at field.

@BEGIN StandardizeDates @IN menu @AS menu_items_valid @OUT data @AS menu_items_std_date

In [22]:
```
menu_item_df['created_at'] = pd.to_datetime(menu_item_df['created_at'])
```

@END StandardizeDates

Step 1.11

Description: Write the cleaned data to files.

In [23]:
```
## Write cleaned data to files
```

```
dish_df.to_csv("../python_cleaned/Dish.csv", index=False)
menu_item_df.to_csv("../python_cleaned/Menu-Item.csv", index=False)
```

Section 2: Document Data Quality Changes

Description: Run a series of tests to prove that the data quality has been improved.

Rationale: We compare and contrast the uncleaned versus the cleaned data.

In [24]:
```
dish_df_uncleaned = pd.read_csv("../open_refine_cleaned/Dish.csv")
menu_item_df_uncleaned = pd.read_csv("../open_refine_cleaned/Menu-Item.csv")
```

Step 2.1a

Description: Provide a high level summary of the cleaned versus uncleaned data frames for the dish_df.

Test Type: Data Completeness

Rationale: We know that many dish names are similar/duplicates of each other, so they may be referring to the same food item, but under different dish_ids. We removed these duplicates, which would explain why the cleaned dish_df has 232,874 ids versus the uncleaned dish_df which has 423,397.

In [25]:
```
dish_df_uncleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 423397 entries, 0 to 423396
Data columns (total 9 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   id              423397 non-null  int64  
 1   name            423397 non-null  object 
 2   description     0 non-null       float64
 3   menus_appeared  423397 non-null  int64  
 4   times_appeared  423397 non-null  int64  
 5   first_appeared  423397 non-null  int64  
 6   last_appeared   423397 non-null  int64  
 7   lowest_price    394297 non-null  float64
 8   highest_price   394297 non-null  float64
dtypes: float64(3), int64(5), object(1)
memory usage: 29.1+ MB
```

In [26]:
```
dish_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 232874 entries, 3 to 423396
Data columns (total 9 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   id              232874 non-null  int64  
 1   name            232874 non-null  object 
 2   description     0 non-null       float64
 3   menus_appeared  232874 non-null  int64  
 4   times_appeared  232874 non-null  int64  
 5   first_appeared  232874 non-null  int64  
 6   last_appeared   232874 non-null  int64  
 7   lowest_price    231661 non-null  float64
 8   highest_price   231661 non-null  float64
```

```
dtypes: float64(3), int64(5), object(1)
memory usage: 17.8+ MB
```

Step 2.1b

Description: Provide a high level summary of the cleaned versus uncleaned data frames for the menu_item_df.

Test Type: Data Completeness

Rationale: We dropped menu items that don't have an associated dish in the dish dataset, which would explain why the cleaned menu_item_df has 846,136 ids versus the uncleaned menu_item_df which has 1,332,726 ids.

In [27]:
```
menu_item_df_uncleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1332726 entries, 0 to 1332725
Data columns (total 9 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   id            1332726 non-null  int64
 1   menu_page_id  1332726 non-null  int64
 2   price         886810 non-null   float64
 3   high_price    91905 non-null    float64
 4   dish_id       1332485 non-null  float64
 5   created_at    1332726 non-null  object
 6   updated_at    1332726 non-null  object
 7   xpos          1332726 non-null  float64
 8   ypos          1332726 non-null  float64
dtypes: float64(5), int64(2), object(2)
memory usage: 91.5+ MB
```

In [28]:
```
menu_item_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 846136 entries, 0 to 1332717
Data columns (total 9 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            846136 non-null  int64
 1   menu_page_id  846136 non-null  int64
 2   price         846136 non-null  float64
 3   high_price    89338 non-null   float64
 4   dish_id       846136 non-null  float64
 5   created_at    846136 non-null  datetime64[ns, UTC]
 6   updated_at    846136 non-null  object
 7   xpos          846136 non-null  float64
 8   ypos          846136 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(5), int64(2), object(1)
memory usage: 64.6+ MB
```

Step 2.2

Test Type: Data Completeness

Description: Test to see if there are still missing dish prices.

Rationale: In step 1.4 we removed dishes with a null price, the following test will assure that there are no dishes with missing prices.

In [29]:
```
def test_missing_dish_prices(menu_item_df):
```

```
    missing_dish_price = menu_item_df["price"].isnull().sum()
    assert (
        missing_dish_price == 0
    ), f"There are {missing_dish_price} missing dish prices"
    print('Test passed!')
```

In [30]:
```
test_missing_dish_prices(menu_item_df)
```

```
Test passed!
```

Step 2.3

Test Type: Data Completeness

Description: Test to see if there are still missing created dates.

Rationale: Each item on the menu needs a created at date.

In [31]:
```
def test_missing_created_dates(menu_item_df):
    missing_created_at = menu_item_df["created_at"].isnull().sum()
    assert (
        missing_created_at == 0
    ), f"There are {missing_created_at} missing created at date"
    print('Test passed!')
```

In [32]:
```
test_missing_created_dates(menu_item_df)
```

```
Test passed!
```

Step 2.4

Test Type: Integrity Constraint Violations

Description: Test to see if there is a valid iso format for dates.

Rationale: In order to compare dish prices over time, the dates must be in the proper format.

In [33]:
```
def test_created_at_datetime(menu_item_df):
    try:
        pd.to_datetime(menu_item_df['created_at'])
        is_datetime = True
    except ValueError:
        is_datetime = False
    assert (
        is_datetime
    ), f"'created_at' column is of type {menu_item_df['created_at'].dtype}, and couldn't
    print('Test passed!')
```

In [34]:
```
menu_item_df.head()
```

Out[34]:

	id	menu_page_id	price	high_price	dish_id	created_at	updated_at	xpos	ypos
0	1	1389	0.039293	NaN	397198.0	2011-03-28 00:00:00+00:00	2011-04-19T00:00:00Z	0.111429	0.254735
1	2	1389	0.058939	NaN	440094.0	2011-03-28 00:00:00+00:00	2011-04-19T00:00:00Z	0.438571	0.254735
2	3	1389	0.039293	NaN	396714.0	2011-03-28 00:00:00+00:00	2011-04-19T00:00:00Z	0.140000	0.261922
3	4	1389	0.049116	NaN	4.0	2011-03-28 00:00:00+00:00	2011-04-19T00:00:00Z	0.377143	0.262720

4	5	3079	0.049116	1.0	5.0	2011-03-28 00:00:00+00:00	2011-04-13T00:00:00Z	0.105714	0.313178

In [35]:
```
test_created_at_datetime(menu_item_df)
```

```
Test passed!
```

Step 2.5

Test Type: Integrity Constraint Violations

Description: Test to see if are no duplicate dish names.

Rationale: In order to compare dishes properly, we need to remove duplicates. We did this in step 1.6 in python.

In [36]:
```
# Test function to check for duplicate names
def test_no_duplicate_names(dish_df):
    duplicate_names = dish_df.groupby(["name"])["name"].count()
    num_duplicate_names = duplicate_names[duplicate_names > 1].count()

    assert (
        num_duplicate_names == 0
    ), f"There are {num_duplicate_names} duplicate dish names"
    print('Test passed!')
```

In [37]:
```
test_no_duplicate_names(dish_df)
```

```
Test passed!
```

Step 2.6

Test Type: Integrity Constraint Violations

Description: Test to ensure there are no leading or trailing whitespaces.

Rationale: In order to compare dishes properly, we need to remove duplicates. We did this in step 1.1 with OpenRefine.

In [38]:
```
def test_no_leading_trailing_whitespace(dish_df):
    dirty_dish_names = (
        dish_df["name"].apply(lambda x: isinstance(x, str) and (x.strip() != x)).sum()
    )

    assert (
        dirty_dish_names == 0
    ), f"There are {dirty_dish_names} dish names with leading and trailing whitespace"
    print('Test passed!')
```

In [39]:
```
test_no_leading_trailing_whitespace(dish_df)
```

```
Test passed!
```

Step 2.7

Test Type: Integrity Constraint Violations

Description: Test to ensure dish names are formatted consistently with title case.

Rationale: In order to compare dishes properly, we need each name to be in title case. We did this in step 1.5 in python.

In [40]:
```
def test_name_consistent_format(dish_df):
    inconsistent_format_count = 0

    for name in dish_df["name"]:
        if not isinstance(name, str):
            # check if the data is of type string
            inconsistent_format_count += 1
        elif name != name.title():
            # check if the name is in title case
            inconsistent_format_count += 1

    assert (
        inconsistent_format_count == 0
    ), f"There are {inconsistent_format_count} names with inconsistent format"
    print('Test passed!')
```

In [41]:
```
test_name_consistent_format(dish_df)
```

```
Test passed!
```

Step 2.8

Test Type: Consistency

Description: Test to ensure that outliers have been removed.

Rationale: In order to avoid data skewing we need to remove outliers. We removed outliers in section 1.7.

In [42]:
```
def outliers_removed(olddataframe, dataframe, column_name, multiplier=1.5):
    Q1 = olddataframe[column_name].quantile(0.10)
    Q3 = olddataframe[column_name].quantile(0.90)
    IQR = Q3 - Q1

    # Count the number of outliers
    outliers = dataframe[
        (dataframe[column_name] < Q1 - multiplier * IQR)
        | (dataframe[column_name] > Q3 + multiplier * IQR)
    ]
    outliers_count = outliers.shape[0]

    assert outliers_count == 0, f"There are {outliers_count} outliers in {column_name}"

def test_menu_item_price_outliers(old_menu_item_df, menu_item_df):
    outliers_removed(old_menu_item_df, menu_item_df, "price", multiplier=2.0)
    print('test passed')
```

In [43]:
```
test_menu_item_price_outliers(menu_item_df_uncleaned, menu_item_df)
```

```
test passed
```

In [44]:
```
menu_item_df_uncleaned.price.describe()
```

Out[44]:
```
count    886810.000000
mean         12.838627
std         499.547387
min           0.000000
25%           0.250000
50%           0.400000
75%           1.000000
```

```
max       180000.000000
Name: price, dtype: float64
```

In [45]:
```python
menu_item_df.price.describe()
```

Out[45]:
```
count    846136.000000
mean          0.087266
std           0.137737
min           0.000000
25%           0.024558
50%           0.039293
75%           0.078585
max           1.000000
Name: price, dtype: float64
```

Step 2.9

Test Type: Consistency

Description: Test to ensure that the data has min-max normalization.

Rationale: The price needs to be on the same scale so that each item is equally weighted.

In [46]:
```python
def test_price_normalization(menu_item_df):
    min_price = menu_item_df['price'].min()
    max_price = menu_item_df['price'].max()

    assert min_price == 0, f"Minimum price is {min_price}, expected 0 after normalizatio
    assert max_price == 1, f"Maximum price is {max_price}, expected 1 after normalizatio

    # To additionally check if there are any values outside the [0,1] range
    assert not ((menu_item_df['price'] < 0).any() or (menu_item_df['price'] > 1).any()),

    print('Test passed!')
```

In [47]:
```python
test_price_normalization(menu_item_df)
```

```
Test passed!
```

Section 3: Work Flow Model

Data cleaning is done in 2 Phases for this project

Phase 1 - Basic Cleaning: Tools Used: Open Refine Steps: Trimming white spaces, string to number conversion, and other basic cleaning tasks are performed using Open Refine.

Phase 2 - Additional Cleaning: Tools Used: Python Steps: More advanced cleaning tasks, such as removing missing prices, outlier detection, etc., are done using Python scripts.

Data Lineage Documentation: Tools Used: YesWorkflow (YW) and OR2YW tool

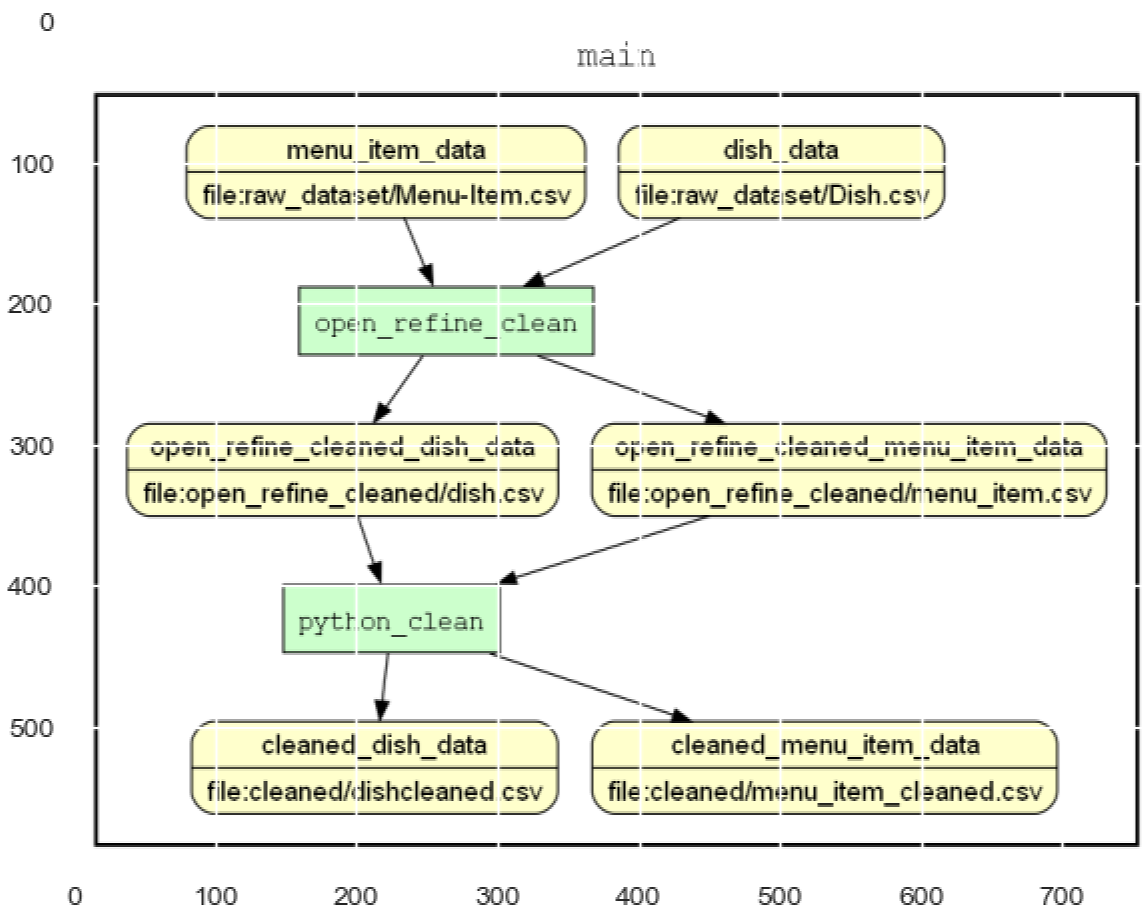
YesWorkflow (YW): Used to document the cleaning steps performed by the Python script. YW compatible comments are added to the Python scripts, enabling YesWorkflow to read and create a workflow diagram based on these comments.

OR2YW: Used to document the internal cleaning steps performed using Open Refine. OR2YW takes a cleaning history JSON file as input and generates a workflow diagram that documents all the changes applied to the dataset during the Open Refine cleaning process.

Overall Workflow Documentation: Tool Used: YesWorkflow (YW) YesWorkflow is used to document the overall workflow of the data cleaning process, providing a visual representation of how the data moves through the various cleaning phases and tools used.

In [55]:
```
# reading png image file
im = img.imread('../yesworkflow/overall_workflow.png')
# show image
plt.imshow(im)
```

Out[55]: <matplotlib.image.AxesImage at 0x1d2827e5dc0>



Step 3.1

Workflow Step: Dish Open Refine cleaning workflow Tool Used: or2yw tool Description: Generates a workflow diagram that documents all the changes applied to the dish dataset during the Open Refine cleaning process.

In [48]:
```
os.popen("or2yw -i dish_history.json -o ../yesworkflow/dish_openrefine_workflow.png -ot=
```

Out[48]: 'java found: java\ndot found: dot\nFile ../yesworkflow/dish_openrefine_workflow.png generated.\n'

In [ ]:
```
# reading png image file
im = img.imread('../yesworkflow/dish_openrefine_workflow.png')
# show image
plt.imshow(im)
```

In [ ]:
```
Step 3.2

Workflow Step: Menu Item Open Refine cleaning workflow
```

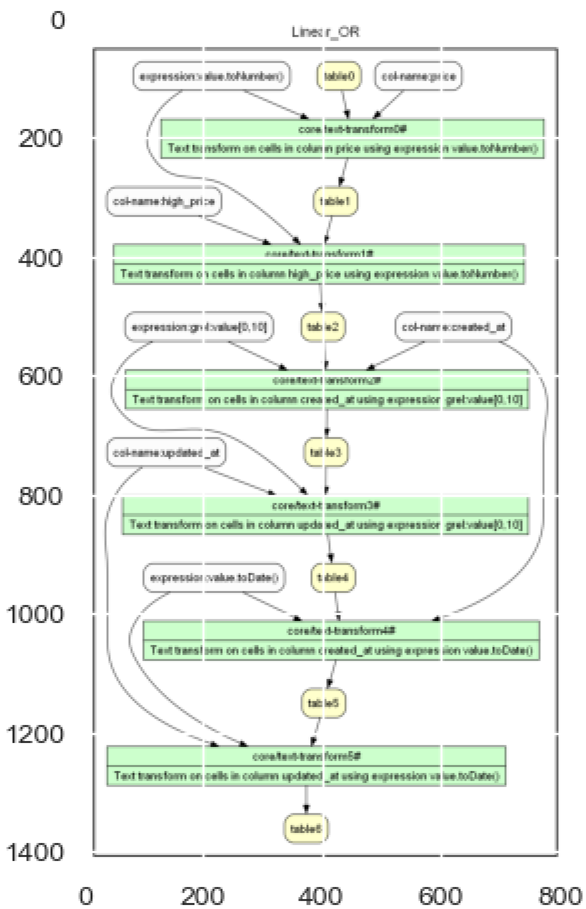
```
Tool Used: or2yw tool
Description: Generates a workflow diagram that documents all the changes applied to the
```

In [50]:
```python
os.popen("or2yw -i menu_item_history.json -o ../yesworkflow/menu_item_workflow.png -ot=p
```

Out[50]:
```
'java found:  java\ndot found:  dot\nFile ../yesworkflow/menu_item_workflow.png generate
d.\n'
```

In [51]:
```python
# reading png image file
im = img.imread('../yesworkflow/menu_item_workflow.png')
# show image
plt.imshow(im)
```

Out[51]:
```
<matplotlib.image.AxesImage at 0x1d28278d790>
```



Step 3.2

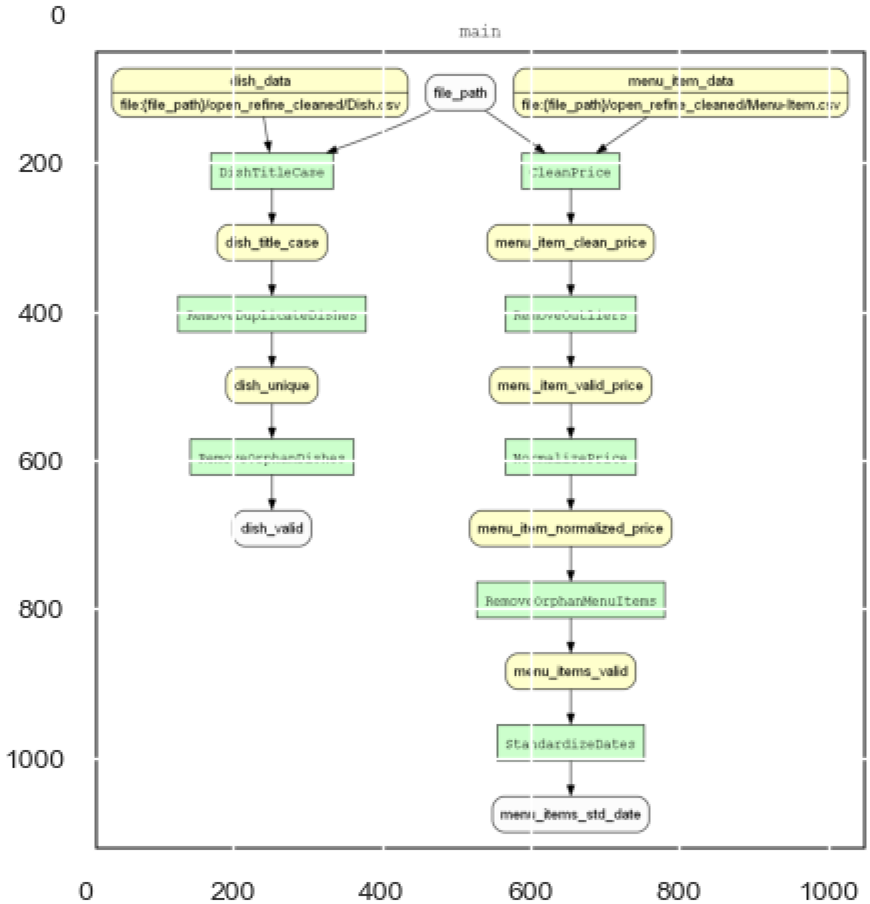
Workflow Step: Final Cleaning workflow for Dish and Menu Item Tool Used: YesWorkflow Description: Generate Yes Workflow for python cleaning steps performed on dish and menu_item dataset

In [53]:
```python
os.popen("java -jar yesworkflow-0.2.0-jar-with-dependencies.jar graph data_cleaning_phas
```

Out[53]:
```
''
```

In [54]:
```python
# reading png image file
im = img.imread('../yesworkflow/dish_menu_clean_workflow.png')
# show image
plt.imshow(im)
```

Out[54]:
```
<matplotlib.image.AxesImage at 0x1d2827a0670>
```



Section 4: Conclusion

In Phase II of the project, we focused on data cleaning to address the data quality problems identified in the dataset. We successfully executed test cases to ensure the data is clean and ready for analysis in the main use case - Dish Price Analysis (U1). The initial data quality problems related to missing data, dirty data, key constraints, and duplicates/similarities were addressed using various data cleaning techniques.

For the main use case, we specifically focused on data related to dish prices, including the 'price', 'highest price', 'lowest price', and 'created at' fields. We ensured that the dataset has no missing dish prices, no NULL values in any columns, and outliers were removed. Additionally, we standardized data formats to ensure consistency, trimmed leading and trailing whitespaces from dish names, and removed duplicate dish names.

Through this comprehensive data cleaning process, we have enhanced the quality of the dataset, ensuring it is accurate, reliable, and suitable for the Dish Price Analysis (U1) use case. This allows us to proceed with further data analysis and gain valuable insights into historical changes in dish prices over time. The clean and structured dataset will enable us to provide meaningful recommendations.

In phase 2 the team made the following contributions:

- Section 1 - Mohammed
- Section 2 - David
- Section 3 - Jhansi
- Section 4 - All

@END main