KATTLE BLUE BOOK

Final Project Report

Group 6: Dannica Concepcion, Fernando Gomez Garcia, Megan Kolvenbach
Scott Levy, Michael Madison, Billy Dee Sorrells

KATTLE BLUE BOOK

Final System Design Document

Introduction

Program Overview

The proposed program will aggregate USDA Feeder and Replacement Cattle Auction data in order to better inform price-setting decisions made by Feeder/Replacement Cattle Auction houses. To accomplish this, the user will enter their current guaranteed purchase rate for comparison with the recent sale prices from area auctions. While recent auction data can be found through searching USDA data, this program will aggregate that data to allow the user to be notified of new USDA data and generate reports which will alert the user when their guaranteed price is outside of the range of current market clearing prices.

Background and Business Problem

Each week across the United States, cattle are bought and sold for feeder and replacement purposes. Cattle purchased as feeder stock are intended for commercial lots to be fed out and butchered for human beef consumption. Replacement cattle are destined to serve as stock on cattle ranches to serve as a production member of cow/calf farms that will continue to sell the produced cattle back to the marketplace through a feeder/replacement cattle auction. Each of these two purposes serve an integral part in the $1 trillion meat and poultry industry which accounts for roughly 5.6% of U.S. GDP.

The auction house provides a necessary component within the meat and poultry marketplace by facilitating the interaction between buyers and sellers, as well as setting a minimum price for each herd of cattle brought to auction in any given week. The requirement to set this minimum price serves as incentive for area ranchers to bring their herd to any particular auction house. This minimum price is the amount the auction house will pay the rancher in the instance that there is not a buyer willing to meet or exceed that price. Cattle that are not purchased by a buyer are instead purchased by the auction house and are then either transported for sale elsewhere or held for later auction. There are added costs to the auction house when they are forced to purchase the cattle, therefore the auction house is incentivized to set a price that is sufficient to attract ranchers to their auction, but also to set a price that will be outbid by the prospective buyers that come to auction.

The purpose of this tool will be to provide timely data from regional auctions to inform the auction houses of the sale prices of cattle. The auction house's ability to accurately identify the expected market clearing price of cattle and set their price contracts with the ranchers accordingly will impact the profitability of the individual firm and in aggregate will impact prices within the meat and poultry industry.

Therefore, the auction house has need of easily digestible information regarding historical cattle prices. Each auction house has insight to historical cattle prices from their own auctions, but not necessarily an easy manner by which to view regional and national auction price information. This information is, however, maintained by the United States Department of Agriculture (USDA). Although, the USDA maintains this information, each report is provided individually, thusly any review of auction data is time consuming and inefficient. The opportunity this presents is to aggregate this data into one central repository which provides the auction house a resource for informing price-setting decisions, the impact of which will also benefit the rancher, the commercial feedlots, and consumers of beef through improved pricing.

User Requirements

The primary user of this information system will be the auction houses; therefore, user requirements will be focused for those needs. The primary need we will address will center around the auction house minimum price-setting function. The auction house will benefit from a portal and tool that provides accurate and timely, regional and national cattle price data in a format that is easily readable and helps in informing the minimum price-setting decision. Additionally, the product will provide categorical splits by weight, class (bull, steer, cow, or heifer), frame, and grade.

Functional requirements of the system include the ability for the auction house to enter a proposed minimum price for cattle. The program will then evaluate the proposed price against current and historical trends as determined by raw data pulls from USDA sources and will provide guidance in the price setting activity through tabular and graphical reporting. Project output is to include summary visualizations to be produced by the system to depict cattle price trends in an easy-to-read, tabular format. Graphs will be utilized to visually depict price dispersion within the user selected categories.

Business Value

A system of this nature presents value for participants in the meat and poultry industry at all levels. Auction houses benefit from increased awareness about the price trends of cattle, leading to greater control over their costs. When auction houses have a comprehensible resource and tool from which to pull aggregated data and automatically suggest prices based on tolerances given, they will be better equipped to set minimum prices to make themselves competitive in the marketplace. Further, with a program designed to provide feedback regarding the competitiveness of a proposed minimum price, auction houses can set their prices with a significantly higher degree of certainty within a profitable range. By introducing a system that can read thousands of records of information and determine competitive pricing, auction houses will benefit from saving a significant amount of time and labor costs compared to manually pulling and analyzing data from USDA sources.

For the ranchers involved in selling their cattle to the auction houses, increased competition in the marketplace will result in more beneficial pricing for them. When individual auction houses are informed about the pricing patterns of their competitors, that auction house benefits from setting its prices to be more attractive to the sellers (i.e. higher prices than their competitors). This business value represents an indirect benefit to the sellers. If the sellers accessed the proposed system themselves, they could view aggregated data regarding auction house pricing, both current and historical trends. This would directly benefit their business as it would provide up-to-date information and inform decisions about which auction house to transact with.

For consumers in the meat and poultry industry, the benefits manifest in decreased pricing of beef products. Increased efficiency and accuracy at remaining competitive in the marketplace would lead auction houses to be more competitive and precise with their minimum price offerings, this would lead to faster and more predictable sales of cattle. That would result in less waste and overbids. Which in turn would lead to lower beef prices at the butcher or grocery store. By utilizing data-informed pricing, auction houses will have the capability of determining prices high enough to initially attract sellers yet low enough to also attract prospective buyers and make a profit. The precision allowed by the proposed information system is superior to any manual technique. The low prices afforded to the buyers of cattle would then pass down as lower prices on consumer beef products.

Application Flow

The application flow of the Kattle Blue Book program will initiate by importing multiple modules, including pandas, numpy, os, matplotlib, and sys. As well as define a function to manage user input for Yes and No responses throughout the application. At the same time, the program will define a dictionary of state name / state abbreviation key value pairs and set up necessary formatting for use with data calculations and output.

A variable is then defined and titled cattleData which will be used to import USDA data containing location, cattle specifications, and price information. A second variable is defined at this time to import a file containing state abbreviations. Following the data import steps, we take time to clean the data by replacing NaN and empty values with zeros, formatting the fields, setting indexes, and dropping unneeded columns.

The program then moves into its interactive phase wherein it will welcome the user to the program and provide some introductory information followed by prompting the user for a username and password combination. If the user already exists, (s)he will be prompted with an option to update the existing password. The program will then prompt the user to input the state in which they are interested in comparing cattle prices. The user will also be prompted to input Class, Weight, Grade, and Frame choices, which will in turn be used to determine the Class, Weight, Grade, and Frame information from the underlying USDA data. A data frame is then generated to display historical price data for the user selected cattle specifications. Based on these criteria the program will then output historical price data and graphs to inform the user on past market clearing prices for the selected cattle. After having had the opportunity to review historical pricing, the user is then prompted to enter their proposed minimum price for the selected cattle specifications. The program will respond to this input by notifying the user how this proposed price compares to historical prices.

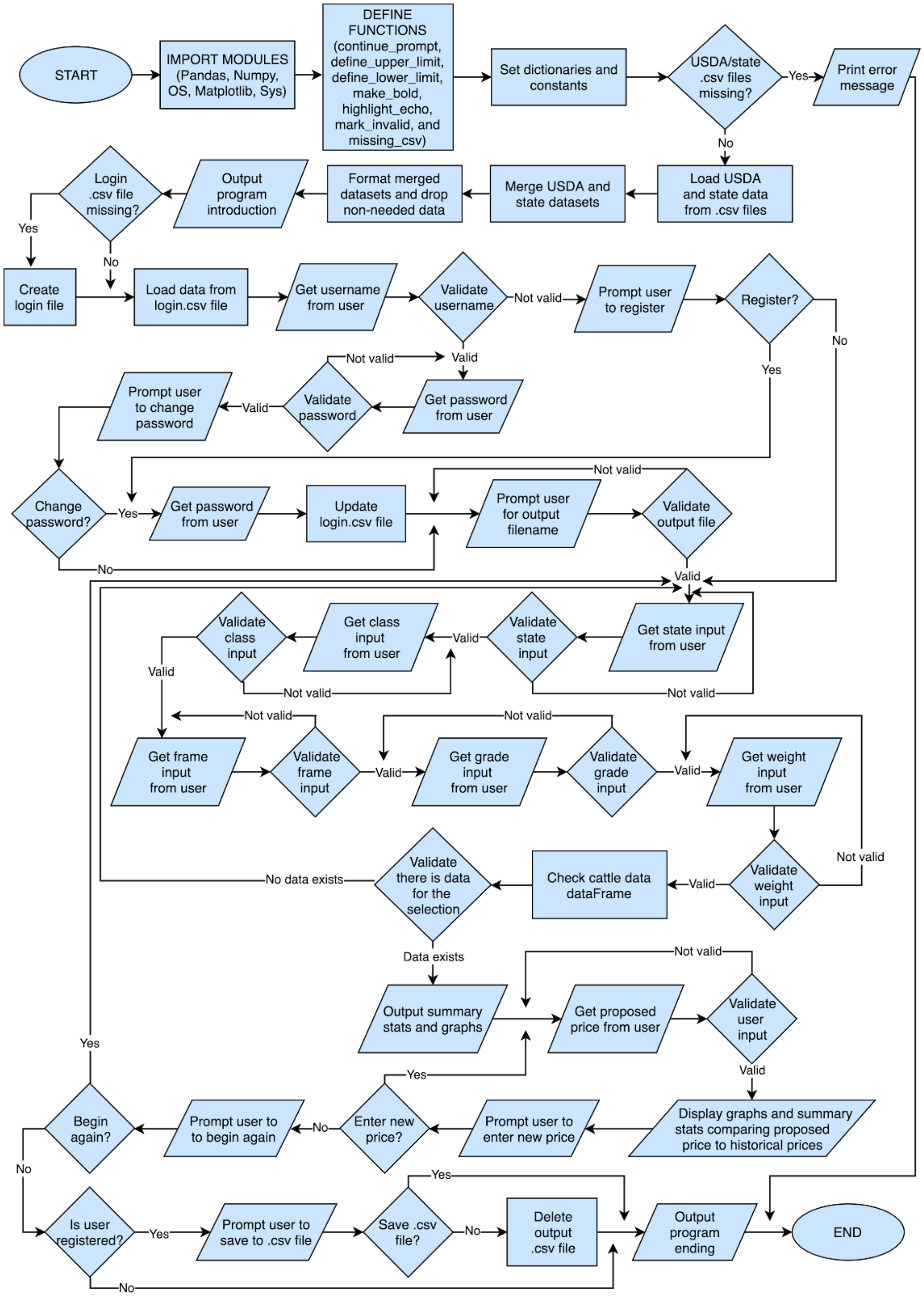
This brings the program to the completion of its purpose for the user’s input, and therefore prompts the user for decisions regarding entering a new price for the existing selection, begin the process again for a new selection of location and cattle, and whether to save the output from the most recent run in a .csv.

Functions

The system will be comprised of the following functions:

- `continue_prompt()` - used to control user entered values in the affirmative or negative
- `define_upper_limit()` - used to define the upper range of calculation
- `define_lower_limit()` - used to define the lower range of calculation
- `make_bold()` - used to bold certain components of the output
- `highlight_echo()` - used to echo the user input in highlighted lettering
- `mark_invalid()` - used to notify the user of an invalid entry and output the response in bold and red lettering
- `missing_csv()` - used to check if the necessary .csv files are downloaded by the user

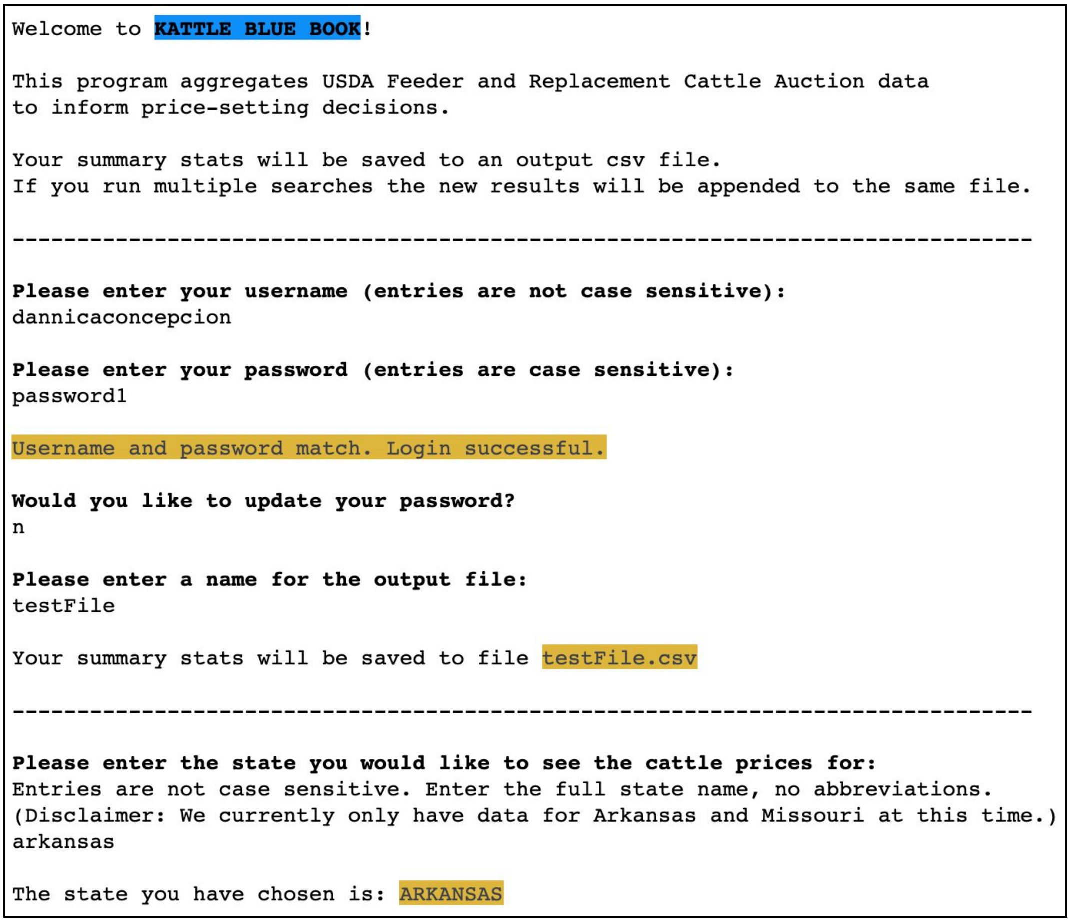
Program Flow



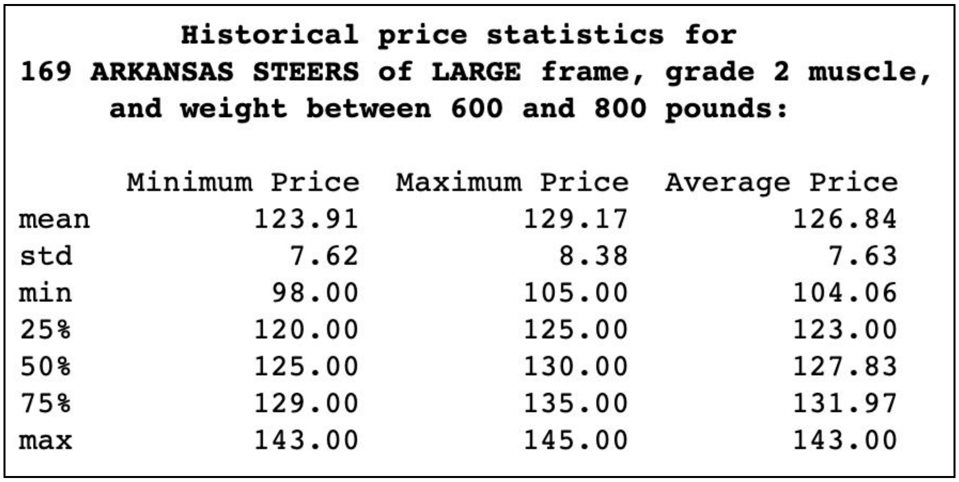
START
IMPORT MODULES (Pandas, Numpy, OS, Matplotlib, Sys)
DEFINE FUNCTIONS (continue_prompt, define_upper_limit, define_lower_limit, make_bold, highlight_echo, mark_invalid, and missing_csv)
Set dictionaries and constants
USDA/state .csv files missing?
Yes
Print error message
No
Load USDA and state data from .csv files
Merge USDA and state datasets
Format merged datasets and drop non-needed data
Output program introduction
Login .csv file missing?
Yes
Create login file
No
Load data from login.csv file
Get username from user
Validate username
Not valid
Prompt user to register
Register?
No
Yes
Valid
Get password from user
Validate password
Not valid
Valid
Prompt user to change password
Change password?
Yes
Get password from user
Update login.csv file
No
Prompt user for output filename
Validate output file
Not valid
Valid
Get state input from user
Validate state input
Not valid
Valid
Get class input from user
Validate class input
Not valid
Valid
Get frame input from user
Validate frame input
Not valid
Valid
Get grade input from user
Validate grade input
Not valid
Valid
Get weight input from user
Validate weight input
Not valid
Valid
Check cattle data dataFrame
Validate there is data for the selection
No data exists
Data exists
Output summary stats and graphs
Get proposed price from user
Validate user input
Not valid
Valid
Display graphs and summary stats comparing proposed price to historical prices
Prompt user to enter new price
Enter new price?
Yes
No
Prompt user to to begin again
Begin again?
Yes
No
Is user registered?
Yes
No
Prompt user to save to .csv file
Save .csv file?
Yes
No
Delete output .csv file
Output program ending
END

Interface

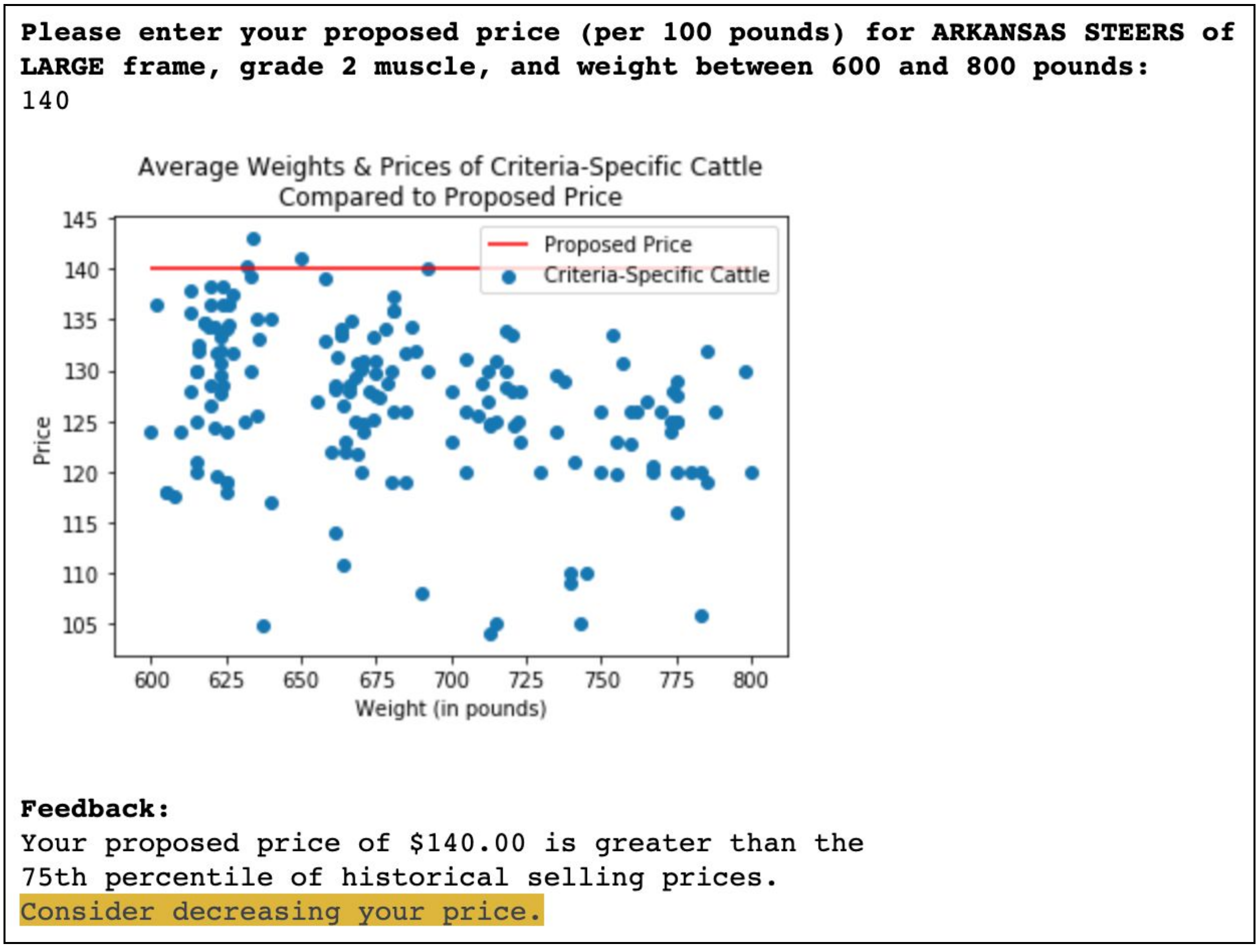
The user will be presented with a command line interface and prompted to login and then enter the desired values for state, class, frame, grade, and weight of the cattle of interest. Upon receiving this input, and performing the necessary calculations, the interface will output reporting to provide the user with information to assist in the minimum price setting decision. After being presented with that data, the user will be prompted to enter a proposed minimum price, the program will again perform the necessary calculations and output the expectations of whether or not the proposed minimum price will be below or above the expected market clearing price and if so, by how much. Examples of the user interface are shown below:



```
Welcome to KATTLE BLUE BOOK!

This program aggregates USDA Feeder and Replacement Cattle Auction data
to inform price-setting decisions.

Your summary stats will be saved to an output csv file.
If you run multiple searches the new results will be appended to the same file.

-------------------------------------------------------------------------------

Please enter your username (entries are not case sensitive):
dannicaconcepcion

Please enter your password (entries are case sensitive):
password1

Username and password match. Login successful.

Would you like to update your password?
n

Please enter a name for the output file:
testFile

Your summary stats will be saved to file testFile.csv

-------------------------------------------------------------------------------

Please enter the state you would like to see the cattle prices for:
Entries are not case sensitive. Enter the full state name, no abbreviations.
(Disclaimer: We currently only have data for Arkansas and Missouri at this time.)
arkansas

The state you have chosen is: ARKANSAS
```



```
          Historical price statistics for
169 ARKANSAS STEERS of LARGE frame, grade 2 muscle,
     and weight between 600 and 800 pounds:

      Minimum Price  Maximum Price  Average Price
mean         123.91         129.17         126.84
std            7.62           8.38           7.63
min           98.00         105.00         104.06
25%          120.00         125.00         123.00
50%          125.00         130.00         127.83
75%          129.00         135.00         131.97
max          143.00         145.00         143.00
```



```
Please enter your proposed price (per 100 pounds) for ARKANSAS STEERS of
LARGE frame, grade 2 muscle, and weight between 600 and 800 pounds:
140
```

```
Feedback:
Your proposed price of $140.00 is greater than the
75th percentile of historical selling prices.
Consider decreasing your price.
```

File Design

Provided Files

The United States Department of Agriculture (USDA) maintains weekly records of Cattle Sales provided by each auction house. The data is available for download via .csv files. For the proof of concept, we chose a select group of auction houses over a limited timeline and aggregated those files into one master file for use within our program. Three .csv files will be provided for use with the program: login.csv, state-abbrevs.csv, and final_feeder_data.csv. The login.csv file will contain all registered username and password combinations. If this file is not provided within the current working directory of the main program, it will simply generate a new file to save newly inputted usernames and passwords. The state-abbrevs.csv file will consist of two columns, one containing full state names and one containing state abbreviations. This file is merged with final_feeder_data.csv so that this data includes full state names which are utilized for data cleaning and later calculations. The final_feeder_data.csv file contains raw USDA cattle data provided by auction houses throughout Arkansas and Missouri. Each row in this file consists of cattle meeting all of the same specifications in a sale (i.e. location, class, grade, frame, and weight range). Therefore, each row can represent multiple cattle sold, indicated by head_count. In future iterations of this program, it will be useful to include data from additional states. If either state-abbrevs.csv or final_feeder_data.csv is not located in the current working directory, the program will inform the user of this error as it cannot perform the necessary calculations without this data.

Data Dictionary

The thirteen variables provided in the following data dictionary were utilized as the basis for analysis within the suggested program.

Source	Fieldname	Type	Length	Description
csv file	username	string	256	User's username
csv file	password	string	256	User's password
csv file	state	string	256	Full state name
csv file	abbreviation	string	256	State abbreviation
csv file	market_location_state	string	256	State (as abbreviation) where the Auction House is located
csv file	class	string	256	Classification of cattle (bull, steer, cow, heifer)
csv file	frame	string	256	Frame of cattle (small, medium, large)
csv file	muscle_grade	integer	1	Grade of cattle (1, 2, 3, 4; 1 being the superior score)
csv file	head_count	integer	4	Number of cattle sold per instance (cattle sold with the all of the same specifications on the same date)
csv file	avg_weight	integer	4	Average weight of cattle sold
csv file	avg_price_min	float	6	Per instance, average minimum price of cattle sold
csv file	avg_price_max	float	6	Per instance, average maximum price of cattle sold
csv file	avg_price	float	6	Per instance, average mean price of cattle sold

Source data from https://mymarketnews.ams.usda.gov/

Project Fulfillment Estimation

Project Kattle Blue Book will demonstrate a wide variety of skills that we have learned and will learn in class. Mastery of variables will be shown through the naming of variables, assignment of variables, use of local and global variables, and the different data types of variables. One example is the variable `grade` which will be assigned through user input and have an integer data type. Constants have been defined to reset the loop if a certain value is entered by the user. In the same process a dictionary is defined with state name and abbreviation key value pairs. Flow control is used throughout the program, most prominently to validate user input.

Various functions will be defined and called in this project. Built-in functions such as `input()` and `print()` will be used. Functions will also be defined and called. For example, function `continue_prompt()` will be used to validate user input based on a string of values that are responses in the affirmative and a similar list of values in the negative.

Create, Read, Update, and Delete functionality is included within the user credential segment of the program. The program will prompt each user to create a set of credentials. The credentials will be used to create and output a file which will later be read for comparing user credentials. The user will have the option to update his/her password. Another use of file/database operation is shown by reading the raw cattle data within .csv files. The USDA data will contain information on hundreds of cattle sold. Use of data computation and presentation can be shown through the modules we imported and their associated analytical functions. The modules we will import using Kattle Blue Book will include numpy, pandas, os, matplotlib, and sys. The user will be prompted at the completion of the program to either Save or Delete the output of Summary Statistics which had been created as a .csv file during the fetch and retrieve data process.

KATTLE BLUE BOOK

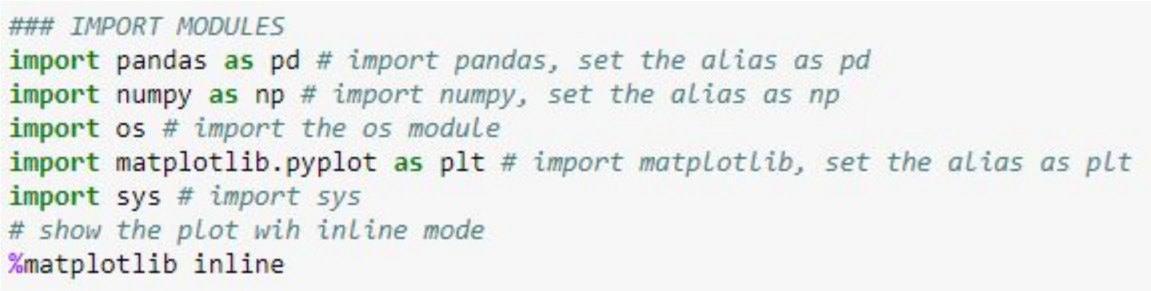
User Manual and Operational Instructions

Preparation

The following files must be downloaded and saved into the same directory as the program notebook file:

final_feeder_data.csv
state-abbrevs.csv
login.csv

Module Imports



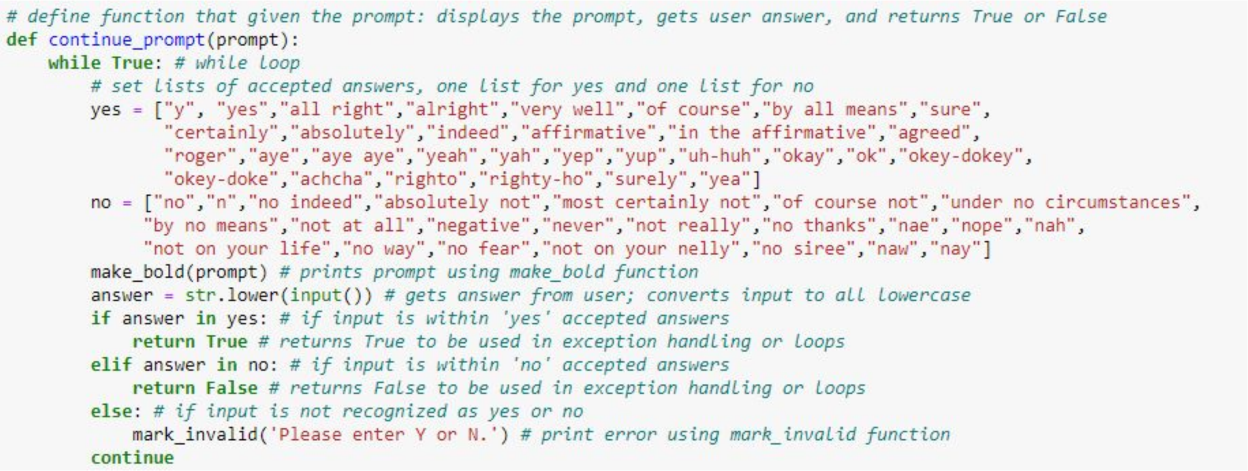
```
### IMPORT MODULES
import pandas as pd # import pandas, set the alias as pd
import numpy as np # import numpy, set the alias as np
import os # import the os module
import matplotlib.pyplot as plt # import matplotlib, set the alias as plt
import sys # import sys
# show the plot wih inline mode
%matplotlib inline
```

The first few lines of code designate the modules to be imported for future use in the program.

- Pandas is imported and assigned the alias "pd" to be utilized in statements related to reading .csv files, merging data structures, and performing various functions with dataframes.
- NumPy is imported and assigned the alias "np" to be utilized in statements related to mathematical calculations, value ranges, and counting values.
- OS is imported to execute commands related to reading, writing, and determining the existence of files.
- MatPlotLib is imported and assigned the alias "plt" to be utilized in statements related to creating, labeling, and printing various scatterplots. This module is also set to mode inline so that the scatterplots print within the output field of Jupyter notebook.
- Sys is imported to allow the program access to certain interpreter functions including exiting the program.

Function Definitions

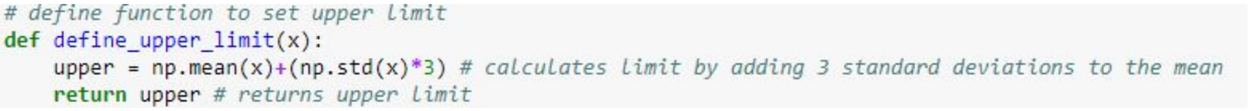
`continue_prompt(statement)`



```python
# define function that given the prompt: displays the prompt, gets user answer, and returns True or False
def continue_prompt(prompt):
    while True: # while loop
        # set lists of accepted answers, one list for yes and one list for no
        yes = ["y", "yes","all right","alright","very well","of course","by all means","sure",
               "certainly","absolutely","indeed","affirmative","in the affirmative","agreed",
               "roger","aye","aye aye","yeah","yah","yep","yup","uh-huh","okay","ok","okey-dokey",
               "okey-doke","achcha","righto","righty-ho","surely","yea"]
        no = ["no","n","no indeed","absolutely not","most certainly not","of course not","under no circumstances",
              "by no means","not at all","negative","never","not really","no thanks","nae","nope","nah",
              "not on your life","no way","no fear","not on your nelly","no siree","naw","nay"]
        make_bold(prompt) # prints prompt using make_bold function
        answer = str.lower(input()) # gets answer from user; converts input to all lowercase
        if answer in yes: # if input is within 'yes' accepted answers
            return True # returns True to be used in exception handling or loops
        elif answer in no: # if input is within 'no' accepted answers
            return False # returns False to be used in exception handling or loops
        else: # if input is not recognized as yes or no
            mark_invalid('Please enter Y or N.') # print error using mark_invalid function
        continue
```

This function allows a multitude of affirmative or negative responses to be read and understood by the program. While the user is prompted only to enter “Y” for affirmative or “N” for negative, the lists created and utilized by this function allow flexibility in user responses. The input parameter is the user’s response to yes/no questions prompted throughout the program. In the rare instance that the user enters a response that is not contained within the lists, this function will inform the user that their response is invalid and prompt them to enter again.

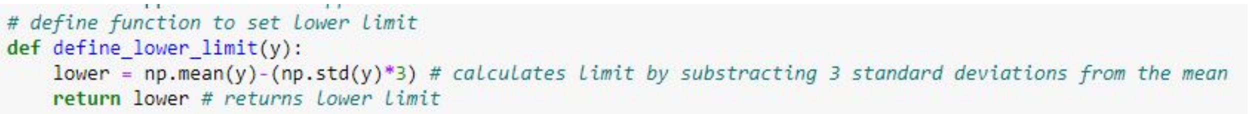
`define_upper_limit(x)`



```python
# define function to set upper limit
def define_upper_limit(x):
    upper = np.mean(x)+(np.std(x)*3) # calculates limit by adding 3 standard deviations to the mean
    return upper # returns upper limit
```

This function sets the upper limit of average cattle weights and prices based upon the SIGMA3 for the purpose of removing outliers which might skew later calculations. The function takes a dataframe as an input parameter, utilizes NumPy operations to calculate the mean and standard deviation of the values within the dataframe, and determines the upper limit outlier based upon the principles of SIGMA3. The function then returns this outlier to be utilized in later comparison statements.

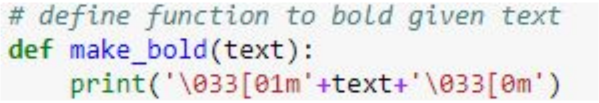
`define_lower_limit(y)`



```python
# define function to set lower limit
def define_lower_limit(y):
    lower = np.mean(y)-(np.std(y)*3) # calculates limit by substracting 3 standard deviations from the mean
    return lower # returns lower limit
```

This function sets the lower limit of average cattle weights and prices based upon the SIGMA3 for the purpose of removing outliers which might skew later calculations. The function takes a dataframe as an input parameter, utilizes NumPy operations to calculate the mean and standard deviation of the values within the dataframe, and determines the lower limit outlier based upon the principles of SIGMA3. The function then returns this outlier to be utilized in later comparison statements.

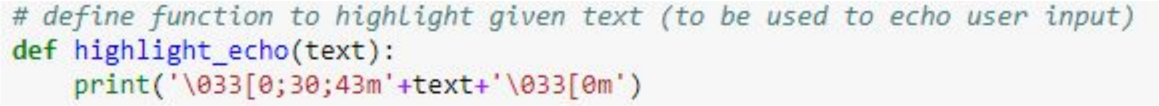
`make_bold(text)`



```
# define function to bold given text
def make_bold(text):
    print('\033[01m'+text+'\033[0m')
```

This function takes a string as an input parameter and prints the string in bolded format. The purpose of this function is to call attention to the section headers and questions to catch the attention of the user.

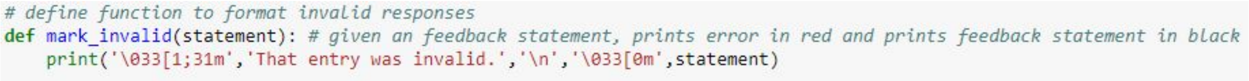
`highlight_echo(text)`



```
# define function to highlight given text (to be used to echo user input)
def highlight_echo(text):
    print('\033[0;30;43m'+text+'\033[0m')
```

This function takes a string as an input parameter and prints the string highlighted in yellow. The purpose of this function is to call attention to the user-selected options once they have been validated.

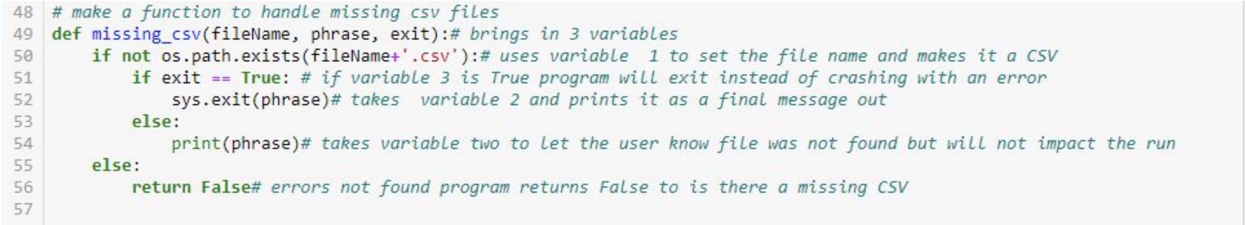
`mark_invalid(statement)`



```
# define function to format invalid responses
def mark_invalid(statement): # given an feedback statement, prints error in red and prints feedback statement in black
    print('\033[1;31m','That entry was invalid.','\n','\033[0m',statement)
```

This function is utilized when a user enters an invalid criteria for cattle specifications. Rather than specifying that the "entry was invalid" for each individual input validation sequence (to be delineated below), this function allows for standardized code to be implemented throughout the program. The input parameter is a unique statement that is produced based upon the specific error triggered by the user (e.g. if the user enters a state that is not included in the current data set). The statement indicating the unique error is passed and printed along with the standardized statement. This function also formats the standardized text, "That entry was invalid," to be bold and red to call attention to the user in the output.

missing_csv(fileName, phrase, exit):



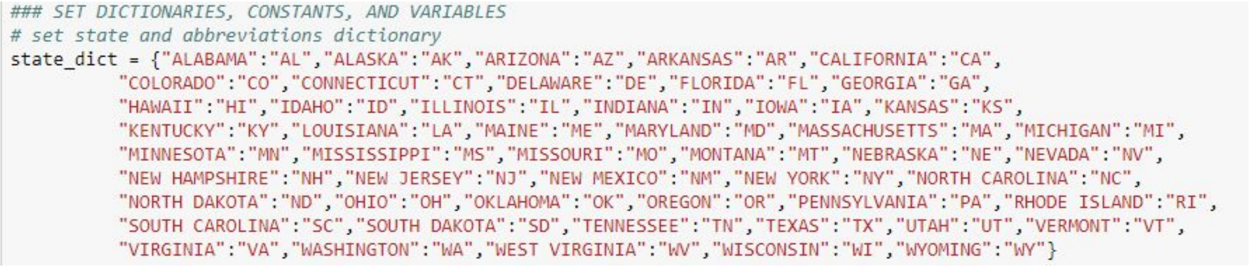
```
# make a function to handle missing csv files
def missing_csv(fileName, phrase, exit):# brings in 3 variables
    if not os.path.exists(fileName+'.csv'):# uses variable  1 to set the file name and makes it a CSV
        if exit == True: # if variable 3 is True program will exit instead of crashing with an error
            sys.exit(phrase)# takes  variable 2 and prints it as a final message out
        else:
            print(phrase)# takes variable two to let the user know file was not found but will not impact the run
    else:
        return False# errors not found program returns False to is there a missing CSV
```

This function ensures that an error message is printed if the csv file is not in the expected path**.** This function takes three input parameters (i.e. the filename, a unique error phrase, and a boolean exit). The provided filename is the name of the file the function searches for within the directory. The `phrase` variable prints a unique error message informing the user that the specified filename does not exist within the directory. The `exit` variable informs the program whether to exit or continue based on the existence of the file in the directory.

If the filename does not exist in the current working directory, the function checks the `exit` variable. If `exit` is assigned the value of True, the function will cause the program to exit after printing the appropriate error message. If the `exit` variable is set to False, the program will print the error message and continue the program. If the file is located in the current working directory, the function returns False.

Dictionary

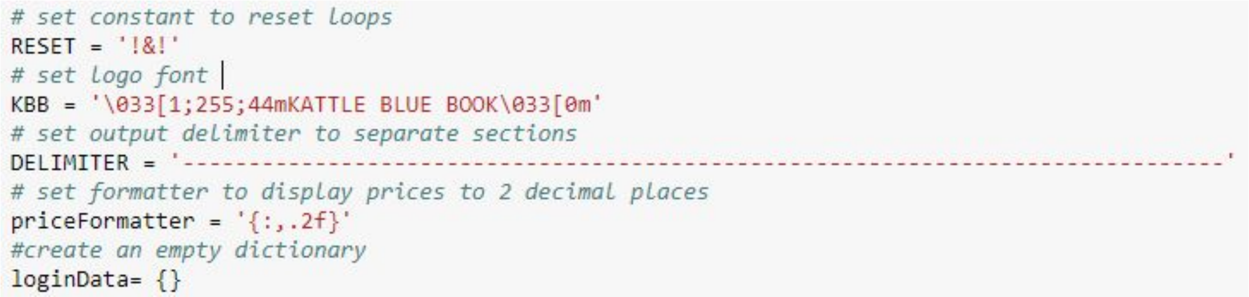
`state_dict`



```
### SET DICTIONARIES, CONSTANTS, AND VARIABLES
# set state and abbreviations dictionary
state_dict = {"ALABAMA":"AL","ALASKA":"AK","ARIZONA":"AZ","ARKANSAS":"AR","CALIFORNIA":"CA",
          "COLORADO":"CO","CONNECTICUT":"CT","DELAWARE":"DE","FLORIDA":"FL","GEORGIA":"GA",
          "HAWAII":"HI","IDAHO":"ID","ILLINOIS":"IL","INDIANA":"IN","IOWA":"IA","KANSAS":"KS",
          "KENTUCKY":"KY","LOUISIANA":"LA","MAINE":"ME","MARYLAND":"MD","MASSACHUSETTS":"MA","MICHIGAN":"MI",
          "MINNESOTA":"MN","MISSISSIPPI":"MS","MISSOURI":"MO","MONTANA":"MT","NEBRASKA":"NE","NEVADA":"NV",
          "NEW HAMPSHIRE":"NH","NEW JERSEY":"NJ","NEW MEXICO":"NM","NEW YORK":"NY","NORTH CAROLINA":"NC",
          "NORTH DAKOTA":"ND","OHIO":"OH","OKLAHOMA":"OK","OREGON":"OR","PENNSYLVANIA":"PA","RHODE ISLAND":"RI",
          "SOUTH CAROLINA":"SC","SOUTH DAKOTA":"SD","TENNESSEE":"TN","TEXAS":"TX","UTAH":"UT","VERMONT":"VT",
          "VIRGINIA":"VA","WASHINGTON":"WA","WEST VIRGINIA":"WV","WISCONSIN":"WI","WYOMING":"WY"}
```

This dictionary is utilized to aid in validating user-inputted state names. Because the current data set only includes data from Arkansas and Missouri. Other state names will not return valid cattle results, so a unique error message informs the user of this constraint. If the user enters another state name, checking for the existence of the state in the dictionary will allow the program to inform the user that, while their state of interest exists, it is not currently represented by the data set. If the inputted state name is not in the dictionary, an error message informs the user of their mistake.

Declaration of Constants



```
# set constant to reset loops
RESET = '!&!'
# set logo font
KBB = '\033[1;255;44mKATTLE BLUE BOOK\033[0m'
# set output delimiter to separate sections
DELIMITER = '-------------------------------------------------------------------------------'
# set formatter to display prices to 2 decimal places
priceFormatter = '{:,.2f}'
#create an empty dictionary
loginData= {}
```

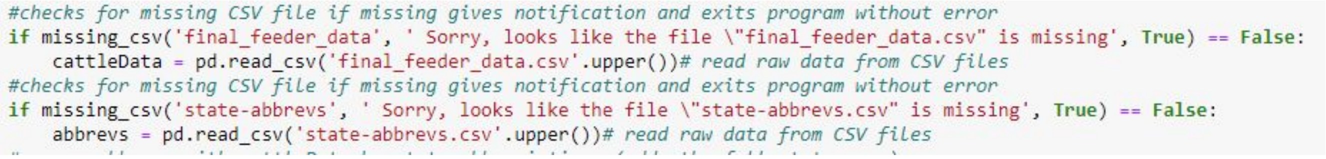
Three constants are declared in the program: `RESET`, `KBB`, and `DELIMITER`.

- `RESET` is utilized throughout the program as a validation flag within iterative user validation statements. To initiate the iterative validation loop, each cow specification variable is initialized to the arbitruary value of `RESET`. If the user's input does not pass the validation criteria, the variable is reset to the value of `RESET` to continue the loop until input is valid.
- `KBB` is utilized to print a formatted Kattle Blue Book logo in which the text is bolded and highlighted in blue.
- `DELIMITER` is a string of hyphens and is utilized in various segments of code to separate logically distinct sections of the program. Specifically, it delimits the welcome and purpose statements from the initial user input prompt. It also delimits the summary statistics graphics from the segment prompting the user for a proposed price. The purpose of this delimiter is to increase readability of the output.

A price formatter is also declared and initialized to print prices to two decimal places.

An empty dictionary is also created in the event that no login.csv file is located within the current working directory to read from. This dictionary is created to store newly inputted user login information.

Reading in Files

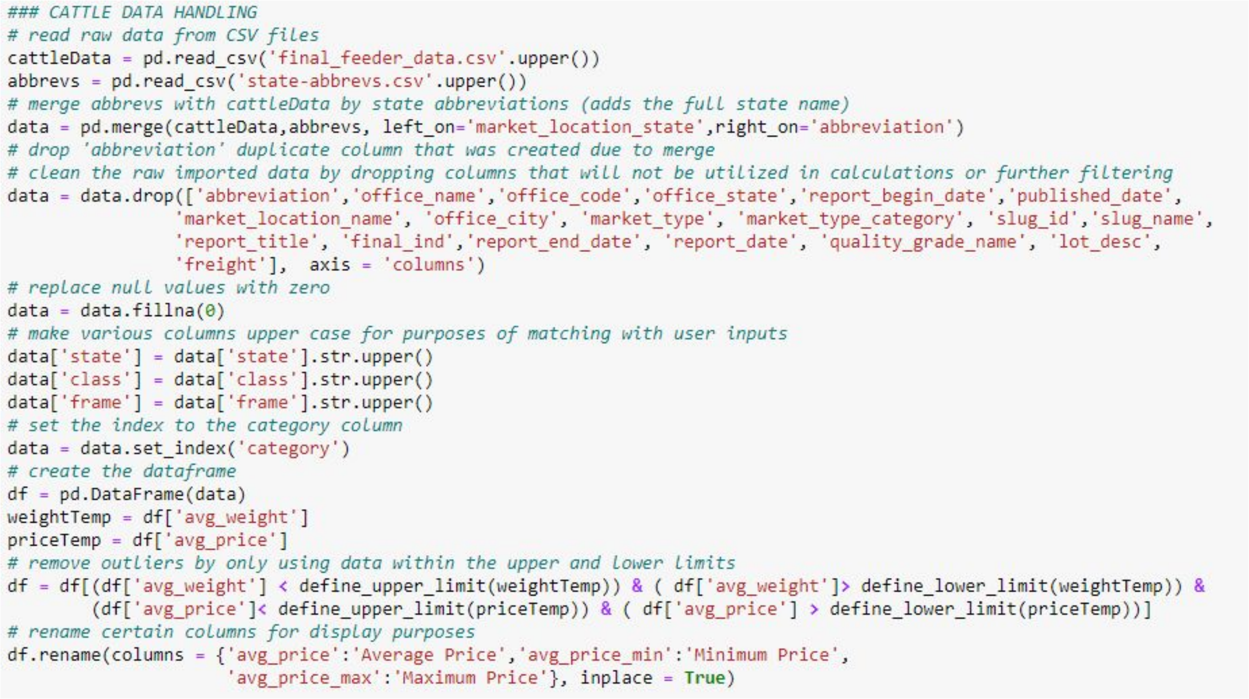


```
#checks for missing CSV file if missing gives notification and exits program without error
if missing_csv('final_feeder_data', ' Sorry, looks like the file \"final_feeder_data.csv" is missing', True) == False:
    cattleData = pd.read_csv('final_feeder_data.csv'.upper())# read raw data from CSV files
#checks for missing CSV file if missing gives notification and exits program without error
if missing_csv('state-abbrevs', ' Sorry, looks like the file \"state-abbrevs.csv" is missing', True) == False:
    abbrevs = pd.read_csv('state-abbrevs.csv'.upper())# read raw data from CSV files
```

These lines of code utilize the `missing_csv` function to determine the existence of the raw data files in the current working directory. If the files are in the current working directory, the program reads in data from two of the provided .csv files. The first one reads in raw USDA cattle data from the final_feeder_data.csv and transforms all alphabetic values into their uppercase form for ease of input validation later in the program. The statement reads the data into a Pandas array and assigns the structured data to the variable, `cowData`. The next line reads in the data from the state-abbrevs.csv file which includes the full names of states in one column and the corresponding abbreviations in the consecutive column. Similarly, this statement reads the data into a Pandas array and assigns the data to the variable, `abbrevs`.

If the files do not exist within the current working directory, `missing_csv` will print an appropriate error message to the user and exit the program.

Cleaning the imported data

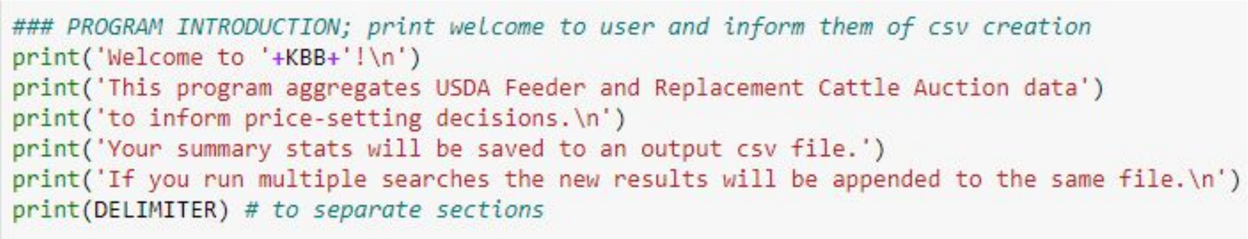


```
### CATTLE DATA HANDLING
# read raw data from CSV files
cattleData = pd.read_csv('final_feeder_data.csv'.upper())
abbrevs = pd.read_csv('state-abbrevs.csv'.upper())
# merge abbrevs with cattleData by state abbreviations (adds the full state name)
data = pd.merge(cattleData,abbrevs, left_on='market_location_state',right_on='abbreviation')
# drop 'abbreviation' duplicate column that was created due to merge
# clean the raw imported data by dropping columns that will not be utilized in calculations or further filtering
data = data.drop(['abbreviation','office_name','office_code','office_state','report_begin_date','published_date',
                 'market_location_name', 'office_city', 'market_type', 'market_type_category', 'slug_id','slug_name',
                 'report_title', 'final_ind','report_end_date', 'report_date', 'quality_grade_name', 'lot_desc',
                 'freight'],  axis = 'columns')
# replace null values with zero
data = data.fillna(0)
# make various columns upper case for purposes of matching with user inputs
data['state'] = data['state'].str.upper()
data['class'] = data['class'].str.upper()
data['frame'] = data['frame'].str.upper()
# set the index to the category column
data = data.set_index('category')
# create the dataframe
df = pd.DataFrame(data)
weightTemp = df['avg_weight']
priceTemp = df['avg_price']
# remove outliers by only using data within the upper and lower limits
df = df[(df['avg_weight'] < define_upper_limit(weightTemp)) & ( df['avg_weight']> define_lower_limit(weightTemp)) &
        (df['avg_price']< define_upper_limit(priceTemp)) & ( df['avg_price'] > define_lower_limit(priceTemp))]
# rename certain columns for display purposes
df.rename(columns = {'avg_price':'Average Price','avg_price_min':'Minimum Price',
                     'avg_price_max':'Maximum Price'}, inplace = True)
```

These lines of code merge the data stored in the two Pandas arrays created in the previous statements (i.e. `cowData` and `abbrevs`) and cleans the data to make it useable in future calculations. Specifically, the arrays are merged based on the `market_location_state` column in `cowData.` Additionally, to proceed with all future calculations the data set only need a certain amount of columns which means that columns that are not needed can be dropped. These columns are dropped in order for the program to run faster and without having to go through redundant data that is not used. After removing the columns the data set still contains values that show as NaN (Not a Number), these will be replaced to show a 0 instead. For the user to run into less issues while the program is running it is best to make all the string values that will be requested in the program upper case. The values that will be transformed into upper case are, state, class and frame. Before we convert the csv into our data frame that will be used for the program, the code resets the index to the 'Category' column.

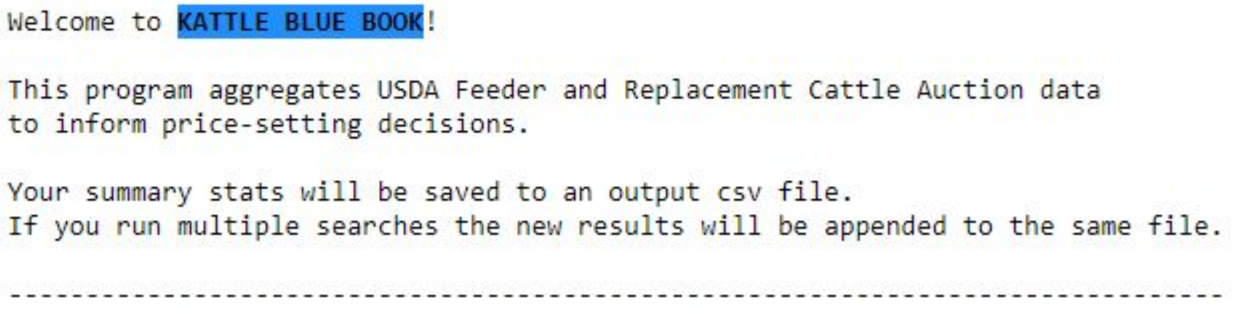
Now that the data is cleaned up and all changes are made, the data set can be placed into a data frame within the pandas module for the program to be able to better manipulate the information. The next lines of code remove all the outliers of the dataset. It will go into the average weight and average price column and check if there is any weight or price out of the previously defined upper and lower limits. If so, the program removes it to be able to only show the valuable data. Thereafter, the program goes ahead and renames the column names of avg_price to Average Price, avg_price_min to Minimum price and avg_price_max to Maximum Price.

Program Introduction



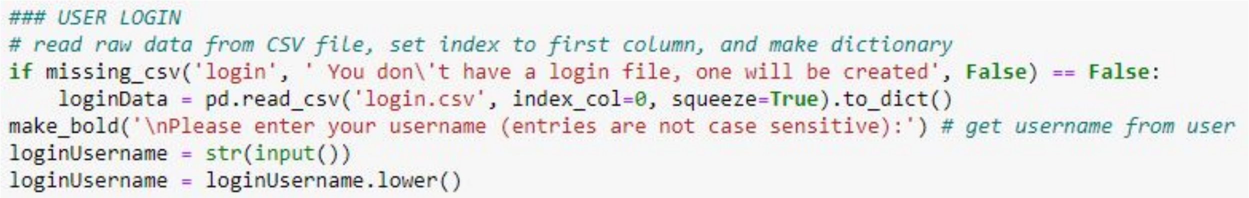
```
### PROGRAM INTRODUCTION; print welcome to user and inform them of csv creation
print('Welcome to '+KBB+'!\n')
print('This program aggregates USDA Feeder and Replacement Cattle Auction data')
print('to inform price-setting decisions.\n')
print('Your summary stats will be saved to an output csv file.')
print('If you run multiple searches the new results will be appended to the same file.\n')
print(DELIMITER) # to separate sections
```

The above code prints a welcome message to the user with the formatted program logo. It then describes the purpose of the program and explains how summary statistics can be saved to a .csv file. The constant, `DELIMITER`, is then printed to separate the program introduction from the remaining sections of the program output.



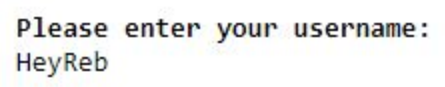
```
Welcome to KATTLE BLUE BOOK!

This program aggregates USDA Feeder and Replacement Cattle Auction data
to inform price-setting decisions.

Your summary stats will be saved to an output csv file.
If you run multiple searches the new results will be appended to the same file.

--------------------------------------------------------------------------------
```

User Login



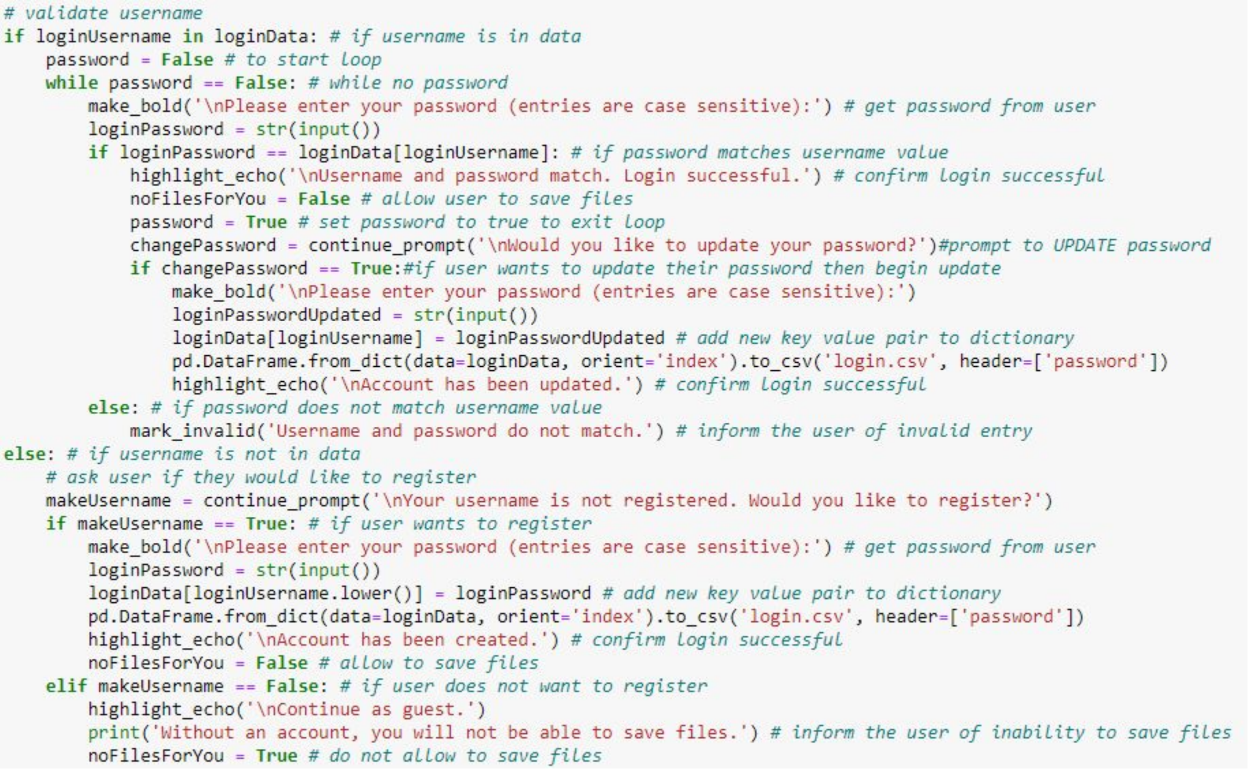
```
### USER LOGIN
# read raw data from CSV file, set index to first column, and make dictionary
if missing_csv('login', ' You don\'t have a login file, one will be created', False) == False:
    loginData = pd.read_csv('login.csv', index_col=0, squeeze=True).to_dict()
make_bold('\nPlease enter your username (entries are not case sensitive):') # get username from user
loginUsername = str(input())
loginUsername = loginUsername.lower()
```

The above code utilizes the `missing_csv` function to determine whether the login.csv file exists in the current working directory to be read from. It either creates a new login.csv file if none exists or reads in the raw data from the provided login.csv file and creates a dictionary using the first column (i.e. usernames) as the index. The program then prompts the user to enter their username which is stored in the string variable, `loginUsername`.



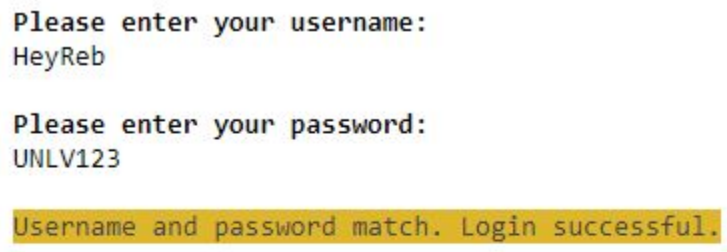
```
Please enter your username:
HeyReb
```

Validate Username



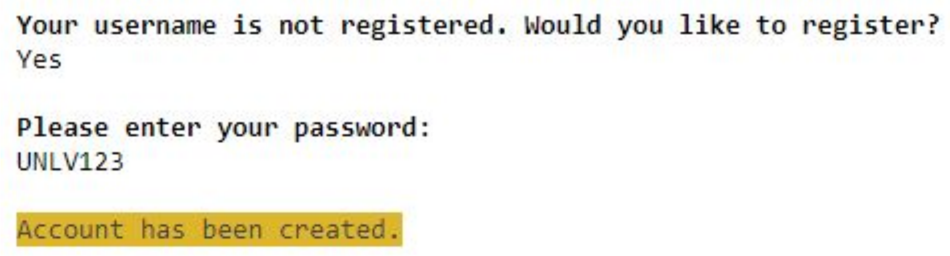
```
# validate username
if loginUsername in loginData: # if username is in data
    password = False # to start loop
    while password == False: # while no password
        make_bold('\nPlease enter your password (entries are case sensitive):') # get password from user
        loginPassword = str(input())
        if loginPassword == loginData[loginUsername]: # if password matches username value
            highlight_echo('\nUsername and password match. Login successful.') # confirm login successful
            noFilesForYou = False # allow user to save files
            password = True # set password to true to exit loop
            changePassword = continue_prompt('\nWould you like to update your password?')#prompt to UPDATE password
            if changePassword == True:#if user wants to update their password then begin update
                make_bold('\nPlease enter your password (entries are case sensitive):')
                loginPasswordUpdated = str(input())
                loginData[loginUsername] = loginPasswordUpdated # add new key value pair to dictionary
                pd.DataFrame.from_dict(data=loginData, orient='index').to_csv('login.csv', header=['password'])
                highlight_echo('\nAccount has been updated.') # confirm login successful
        else: # if password does not match username value
            mark_invalid('Username and password do not match.') # inform the user of invalid entry
else: # if username is not in data
    # ask user if they would like to register
    makeUsername = continue_prompt('\nYour username is not registered. Would you like to register?')
    if makeUsername == True: # if user wants to register
        make_bold('\nPlease enter your password (entries are case sensitive):') # get password from user
        loginPassword = str(input())
        loginData[loginUsername.lower()] = loginPassword # add new key value pair to dictionary
        pd.DataFrame.from_dict(data=loginData, orient='index').to_csv('login.csv', header=['password'])
        highlight_echo('\nAccount has been created.') # confirm login successful
        noFilesForYou = False # allow to save files
    elif makeUsername == False: # if user does not want to register
        highlight_echo('\nContinue as guest.')
        print('Without an account, you will not be able to save files.') # inform the user of inability to save files
        noFilesForYou = True # do not allow to save files
```

The next segment of code validates whether the username already exists within the previously created dictionary or if it is a new username. If the username exists in the `loginData` dictionary, a while loop is initiated to validate the user's password. If the entered password matches the value in the dictionary assigned to the username key, an affirmative message is printed and the `noFilesForYou` flag is set to false, allowing the user to save files. If the password does not match, an error message is printed notifying the user of their mistake. The while loop continues until the password is entered correctly.



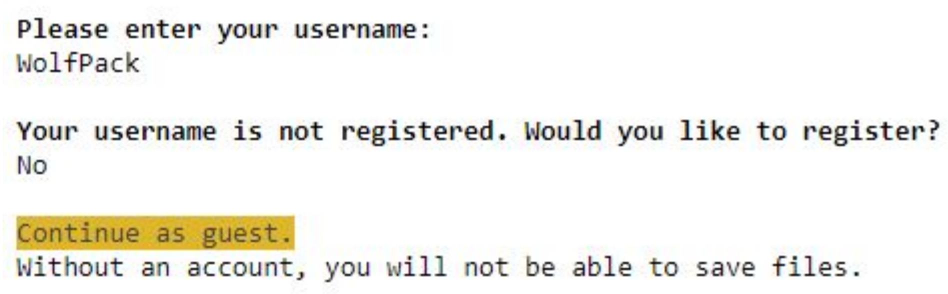
```
Please enter your username:
HeyReb

Please enter your password:
UNLV123

Username and password match. Login successful.
```

If the username is not already in the `loginData` dictionary, the program asks the user if they would like to register the username. If the user responds affirmatively, the program prompts the user to enter a password. If then creates a new key-value pair for the dictionary and prints a confirmation message. It also sets the `noFilesForYou` flag to false, allowing the new user to save files.



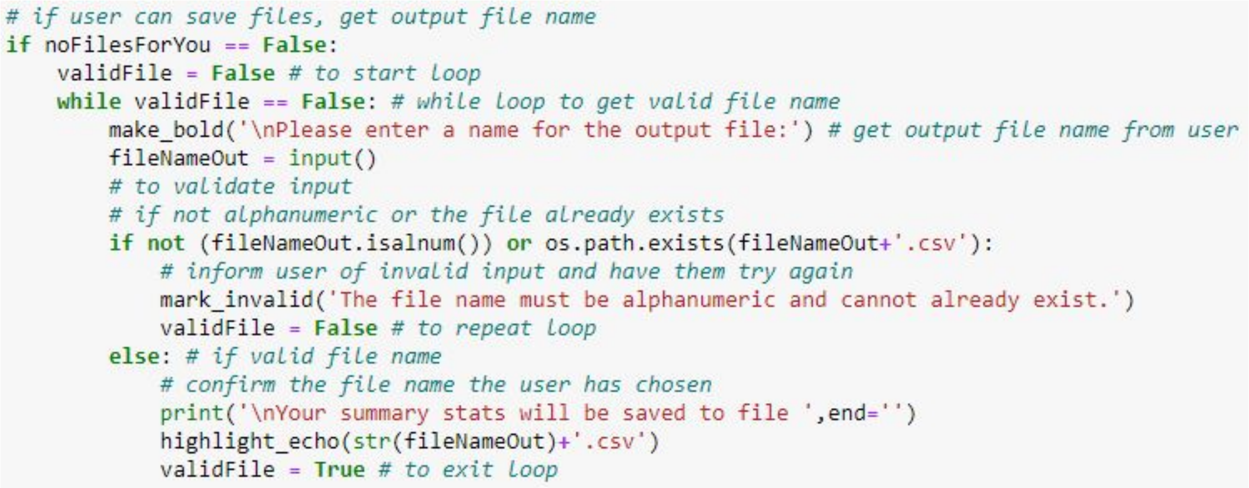
```
Your username is not registered. Would you like to register?
Yes

Please enter your password:
UNLV123

Account has been created.
```

If the user does not want to register, they are allowed to continue as a "guest," but are not allowed to save output files so the `noFilesForYou` flag is set to true.



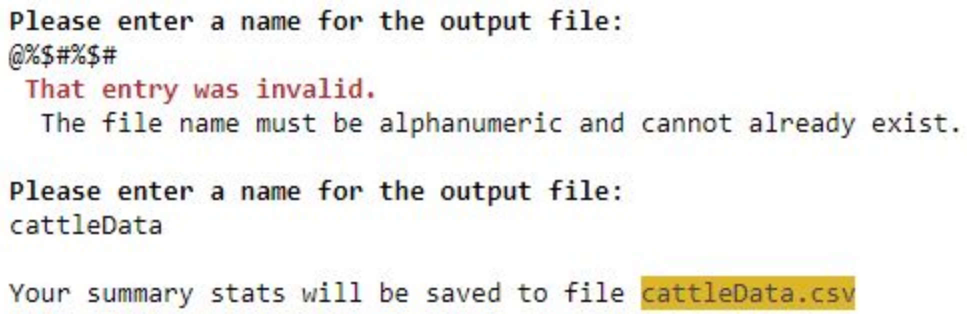
```
Please enter your username:
WolfPack

Your username is not registered. Would you like to register?
No

Continue as guest.
Without an account, you will not be able to save files.
```

Specify Output File



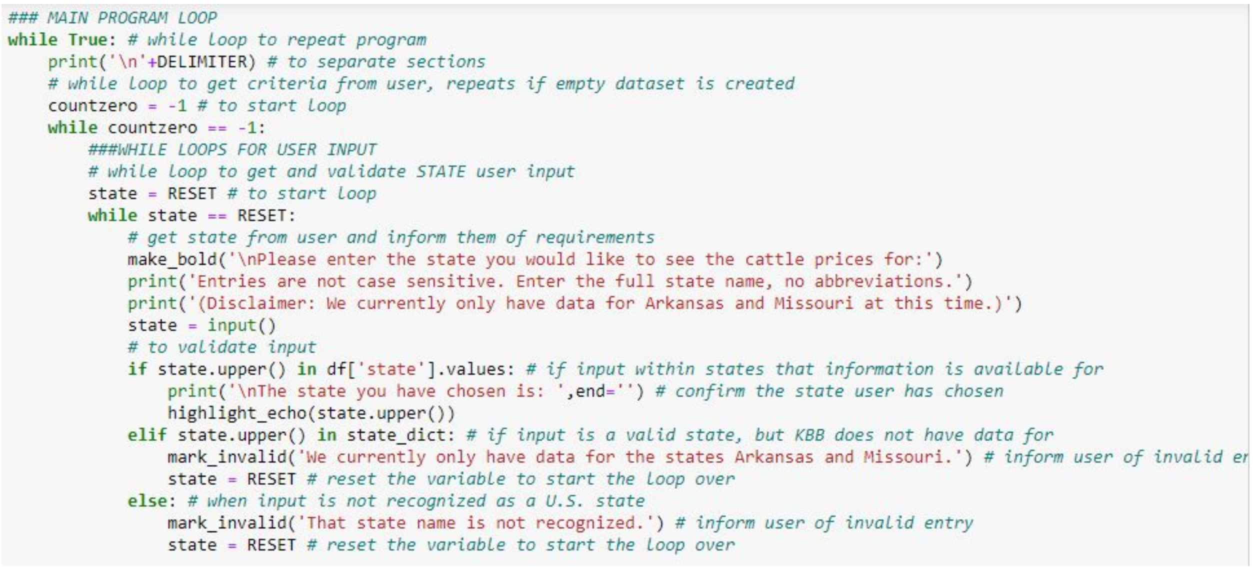
```
# if user can save files, get output file name
if noFilesForYou == False:
    validFile = False # to start loop
    while validFile == False: # while loop to get valid file name
        make_bold('\nPlease enter a name for the output file:') # get output file name from user
        fileNameOut = input()
        # to validate input
        # if not alphanumeric or the file already exists
        if not (fileNameOut.isalnum()) or os.path.exists(fileNameOut+'.csv'):
            # inform user of invalid input and have them try again
            mark_invalid('The file name must be alphanumeric and cannot already exist.')
            validFile = False # to repeat loop
        else: # if valid file name
            # confirm the file name the user has chosen
            print('\nYour summary stats will be saved to file ',end='')
            highlight_echo(str(fileNameOut)+'.csv')
            validFile = True # to exit loop
```

If the user can save files (i.e. they have a registered account), the program will prompt them to enter an output file name to save summary statistics to. After the initial if statement is validated confirming that the user can save files, a while loop is initiated to validate that the file name entered is alphanumeric and does not already exist within the same directory. If neither condition is met, the loop continues prompting the user to enter a new file name until it is valid. The validated filename is assigned to the variable, `fileNameOut`, for use later on in the program.



```
Please enter a name for the output file:
@%$#%$#
 That entry was invalid.
  The file name must be alphanumeric and cannot already exist.

Please enter a name for the output file:
cattleData

Your summary stats will be saved to file cattleData.csv
```

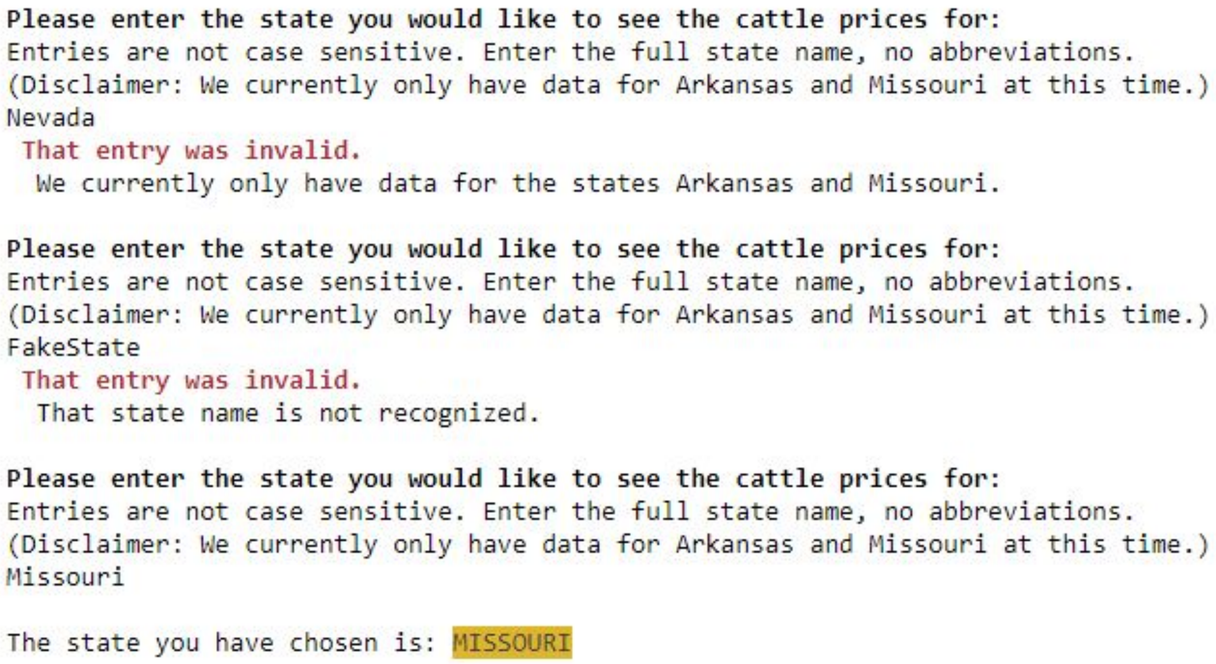
Main Program Loop and Get State Name



```
### MAIN PROGRAM LOOP
while True: # while loop to repeat program
    print('\n'+DELIMITER) # to separate sections
    # while loop to get criteria from user, repeats if empty dataset is created
    countzero = -1 # to start loop
    while countzero == -1:
        ###WHILE LOOPS FOR USER INPUT
        # while loop to get and validate STATE user input
        state = RESET # to start loop
        while state == RESET:
            # get state from user and inform them of requirements
            make_bold('\nPlease enter the state you would like to see the cattle prices for:')
            print('Entries are not case sensitive. Enter the full state name, no abbreviations.')
            print('(Disclaimer: We currently only have data for Arkansas and Missouri at this time.)')
            state = input()
            # to validate input
            if state.upper() in df['state'].values: # if input within states that information is available for
                print('\nThe state you have chosen is: ',end='') # confirm the state user has chosen
                highlight_echo(state.upper())
            elif state.upper() in state_dict: # if input is a valid state, but KBB does not have data for
                mark_invalid('We currently only have data for the states Arkansas and Missouri.') # inform user of invalid er
                state = RESET # reset the variable to start the loop over
            else: # when input is not recognized as a U.S. state
                mark_invalid('That state name is not recognized.') # inform user of invalid entry
                state = RESET # reset the variable to start the loop over
```

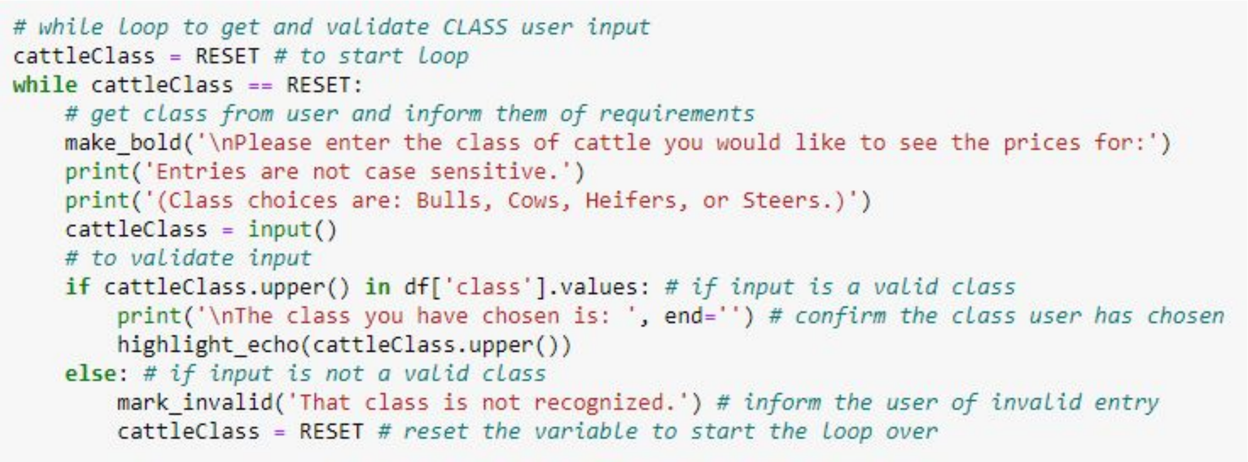
The above code initiates the main program loop by asking the user to specify the state for which they would like to view cattle pricing data. Currently, the program only has data for Missouri and Arkansas. Therefore, if the user enters a state name that does not exist or for which the current version of the program does not have data, appropriate error messages are printed. The outer while loop is initiated with a continually true condition to repeat the prompts if necessary. A counter is used (i.e. `countzero`) to continue the while loop if the user-inputted criteria results in an empty dataset (i.e. if zero cows meet the user-defined specifications).

The innermost while loop is initiated by assigning the value of the constant, `RESET`, to the variable, `state`. The program then prompts the user to enter their desired state. The entered state is then validated in an if...elif...else statement. It is first determined whether the state exists in the "state" column of the dataframe previously created using the merged USDA data. If the state name does exist in the state column, an affirmative message is printed and the name is saved in the variable, `state`. If the user input is not in the dataframe, the `elif` statement checks if it is in the `state_dict` dictionary to determine whether it is a valid state name. If it is in the dictionary, a message is printed indicating that there is currently no data on the specified state. The loop reset to prompt the user for a new state name. If the state is not in the dictionary, an error message is printed indicating that the state name is invalid and the loop continues until a valid state name is provided.



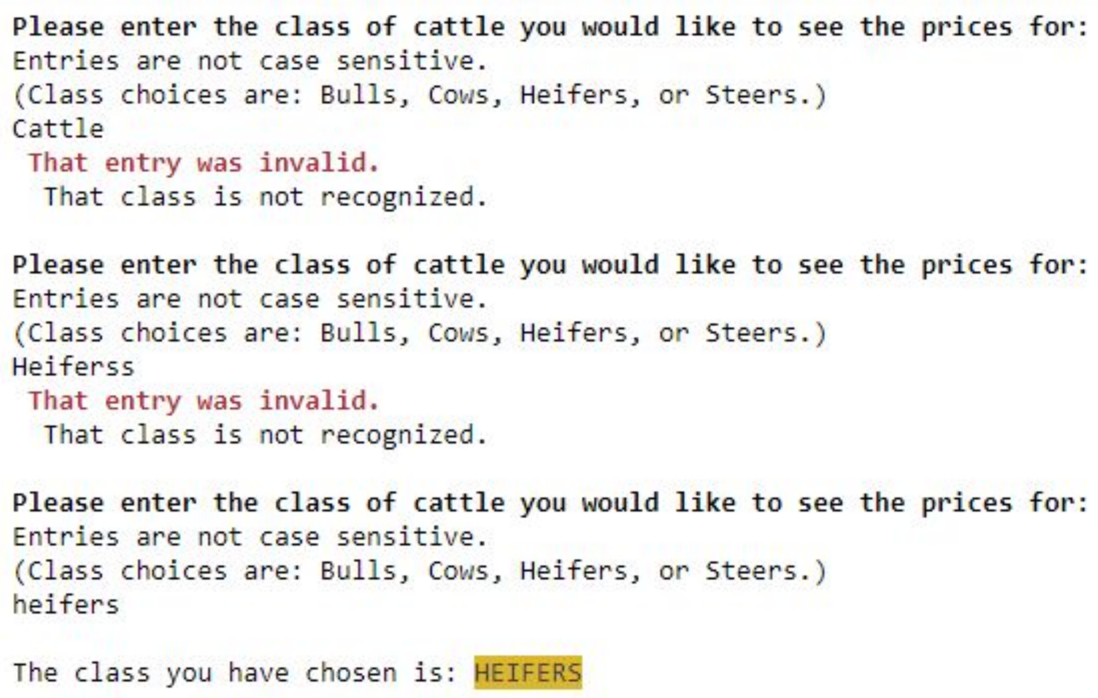
```
Please enter the state you would like to see the cattle prices for:
Entries are not case sensitive. Enter the full state name, no abbreviations.
(Disclaimer: We currently only have data for Arkansas and Missouri at this time.)
Nevada
 That entry was invalid.
  We currently only have data for the states Arkansas and Missouri.

Please enter the state you would like to see the cattle prices for:
Entries are not case sensitive. Enter the full state name, no abbreviations.
(Disclaimer: We currently only have data for Arkansas and Missouri at this time.)
FakeState
 That entry was invalid.
  That state name is not recognized.

Please enter the state you would like to see the cattle prices for:
Entries are not case sensitive. Enter the full state name, no abbreviations.
(Disclaimer: We currently only have data for Arkansas and Missouri at this time.)
Missouri

The state you have chosen is: MISSOURI
```

Get Cattle Class from User



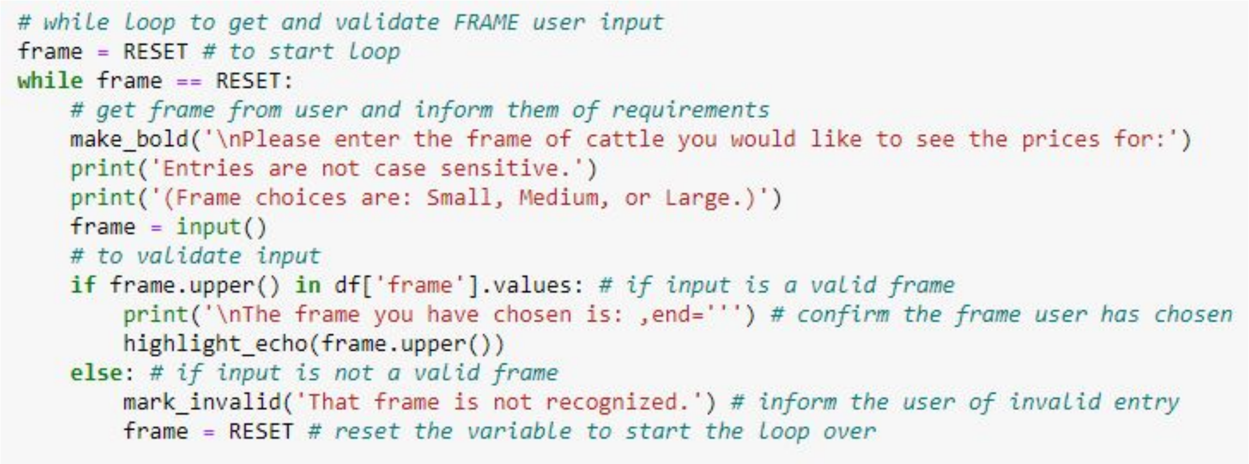
```
# while loop to get and validate CLASS user input
cattleClass = RESET # to start loop
while cattleClass == RESET:
    # get class from user and inform them of requirements
    make_bold('\nPlease enter the class of cattle you would like to see the prices for:')
    print('Entries are not case sensitive.')
    print('(Class choices are: Bulls, Cows, Heifers, or Steers.)')
    cattleClass = input()
    # to validate input
    if cattleClass.upper() in df['class'].values: # if input is a valid class
        print('\nThe class you have chosen is: ', end='') # confirm the class user has chosen
        highlight_echo(cattleClass.upper())
    else: # if input is not a valid class
        mark_invalid('That class is not recognized.') # inform the user of invalid entry
        cattleClass = RESET # reset the variable to start the loop over
```

This section of code retrieves the cattle class from the user. The `cattleClass` variable is initiated to the value of the constant, `RESET`, to begin the while loop. The program then prompts the user to enter the class of cattle they are interested in pricing along with the class options. User input is saved in `cattleClass` variable. If the entered class exists in the “class” column of the previously defined data frame, an affirmative message is printed and the class is echoed back to the user highlighted in yellow. If the entered class is not in the data frame, the program prints an error message and the `cattleClass` variable is reset for the while loop to continue.



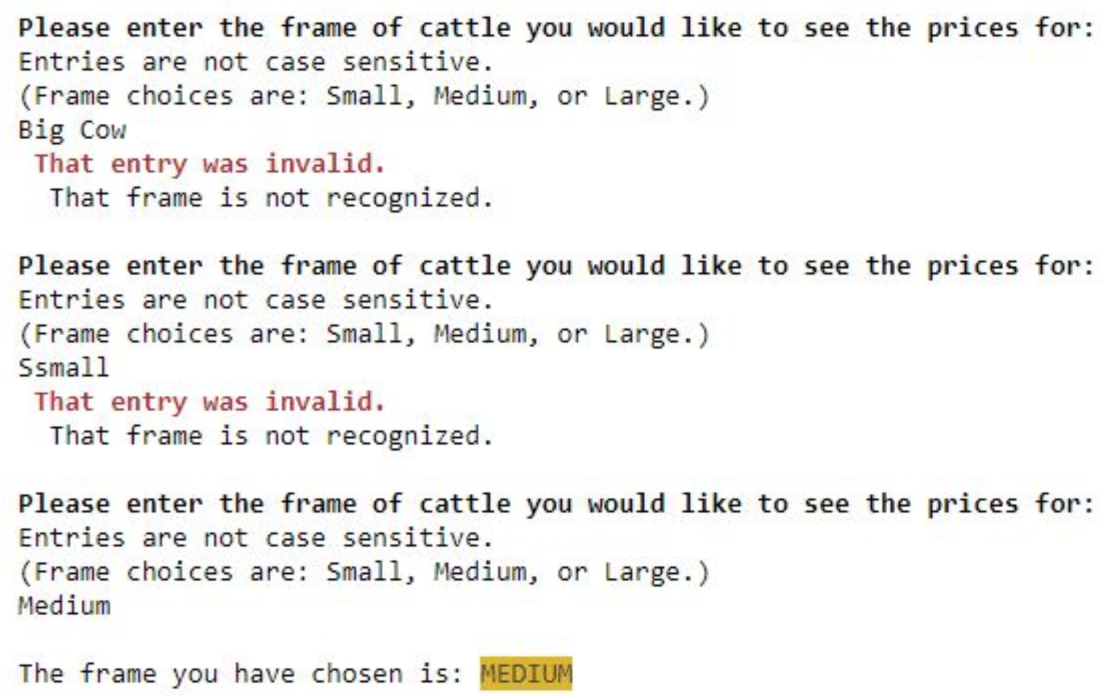
```
Please enter the class of cattle you would like to see the prices for:
Entries are not case sensitive.
(Class choices are: Bulls, Cows, Heifers, or Steers.)
Cattle
 That entry was invalid.
  That class is not recognized.

Please enter the class of cattle you would like to see the prices for:
Entries are not case sensitive.
(Class choices are: Bulls, Cows, Heifers, or Steers.)
Heiferss
 That entry was invalid.
  That class is not recognized.

Please enter the class of cattle you would like to see the prices for:
Entries are not case sensitive.
(Class choices are: Bulls, Cows, Heifers, or Steers.)
heifers

The class you have chosen is: HEIFERS
```

Get Cattle Frame from User



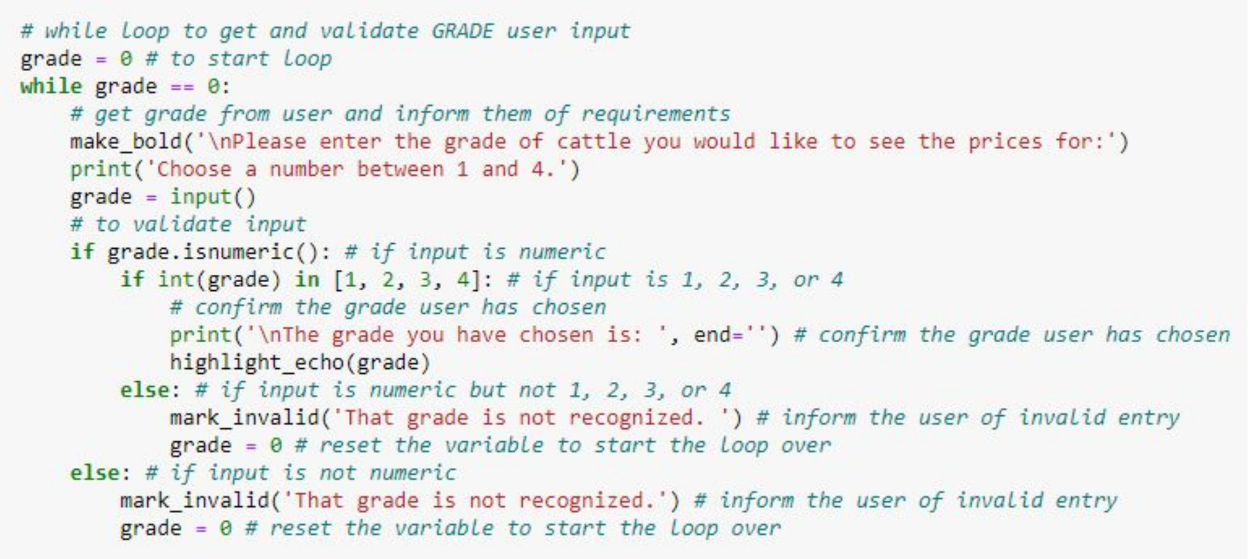
```
# while loop to get and validate FRAME user input
frame = RESET # to start loop
while frame == RESET:
    # get frame from user and inform them of requirements
    make_bold('\nPlease enter the frame of cattle you would like to see the prices for:')
    print('Entries are not case sensitive.')
    print('(Frame choices are: Small, Medium, or Large.)')
    frame = input()
    # to validate input
    if frame.upper() in df['frame'].values: # if input is a valid frame
        print('\nThe frame you have chosen is: ,end=''') # confirm the frame user has chosen
        highlight_echo(frame.upper())
    else: # if input is not a valid frame
        mark_invalid('That frame is not recognized.') # inform the user of invalid entry
        frame = RESET # reset the variable to start the loop over
```

This section of code retrieves the cattle frame from the user. The `frame` variable is initiated to the value of the constant, `RESET`, to begin the while loop. The program then prompts the user to enter the frame of cattle they are interested in pricing along with the frame options. User input is saved in `frame` variable. If the entered frame exists in the “frame” column of the previously defined data frame, an affirmative message is printed and the frame is echoed back to the user highlighted in yellow. If the entered class is not in the data frame, the program prints an error message and the `frame` variable is reset for the while loop to continue.



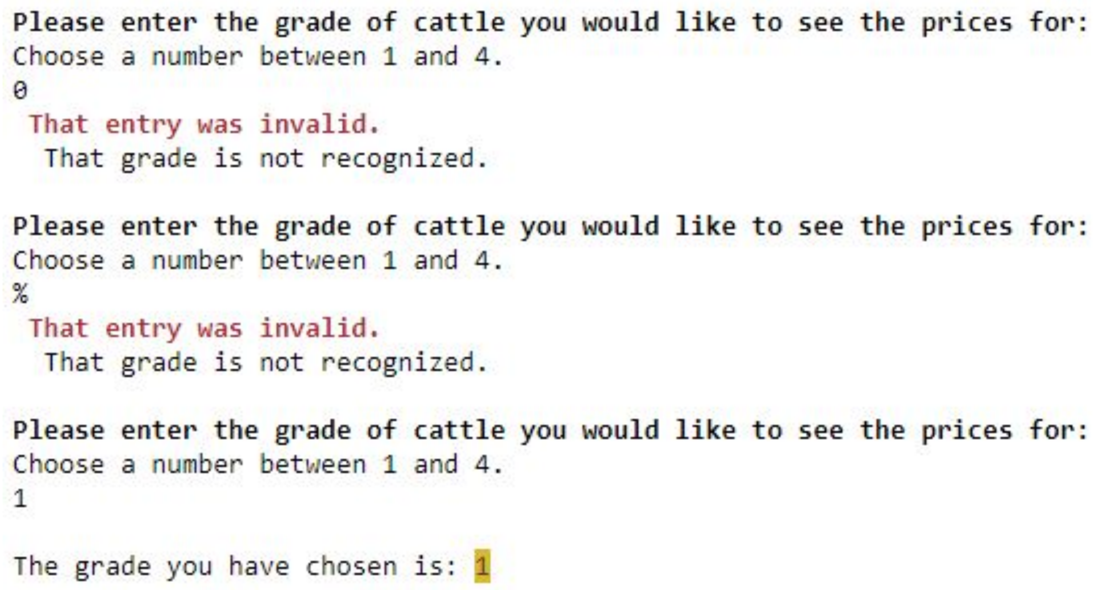
```
Please enter the frame of cattle you would like to see the prices for:
Entries are not case sensitive.
(Frame choices are: Small, Medium, or Large.)
Big Cow
 That entry was invalid.
  That frame is not recognized.

Please enter the frame of cattle you would like to see the prices for:
Entries are not case sensitive.
(Frame choices are: Small, Medium, or Large.)
Ssmall
 That entry was invalid.
  That frame is not recognized.

Please enter the frame of cattle you would like to see the prices for:
Entries are not case sensitive.
(Frame choices are: Small, Medium, or Large.)
Medium

The frame you have chosen is: MEDIUM
```

Get Cattle Grade from User



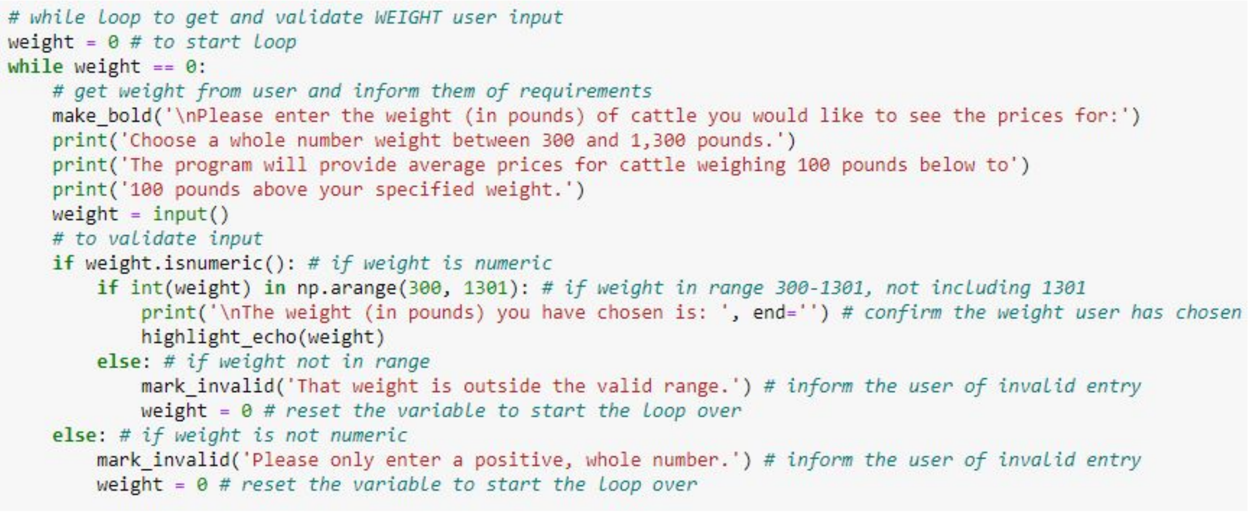
```
# while loop to get and validate GRADE user input
grade = 0 # to start loop
while grade == 0:
    # get grade from user and inform them of requirements
    make_bold('\nPlease enter the grade of cattle you would like to see the prices for:')
    print('Choose a number between 1 and 4.')
    grade = input()
    # to validate input
    if grade.isnumeric(): # if input is numeric
        if int(grade) in [1, 2, 3, 4]: # if input is 1, 2, 3, or 4
            # confirm the grade user has chosen
            print('\nThe grade you have chosen is: ', end='') # confirm the grade user has chosen
            highlight_echo(grade)
        else: # if input is numeric but not 1, 2, 3, or 4
            mark_invalid('That grade is not recognized. ') # inform the user of invalid entry
            grade = 0 # reset the variable to start the loop over
    else: # if input is not numeric
        mark_invalid('That grade is not recognized.') # inform the user of invalid entry
        grade = 0 # reset the variable to start the loop over
```

This section of code retrieves the cattle grade from the user. The `grade` variable is initiated to 0 to begin the while loop. The program then prompts the user to enter the grade of cattle they are interested in pricing along with the grade options. User input is saved in `grade` variable. If the entered grade is numeric, the program checks if it exists in an array with values 1, 2, 3, or 4. If so, an affirmative message is printed and the grade is echoed back to the user highlighted in yellow. If the entered grade is not in the array, the program prints an error message and the `grade` variable is reset for the while loop to continue. If the entered grade is not numeric, a similar error message is printed and the while loop continues until a valid grade is entered.



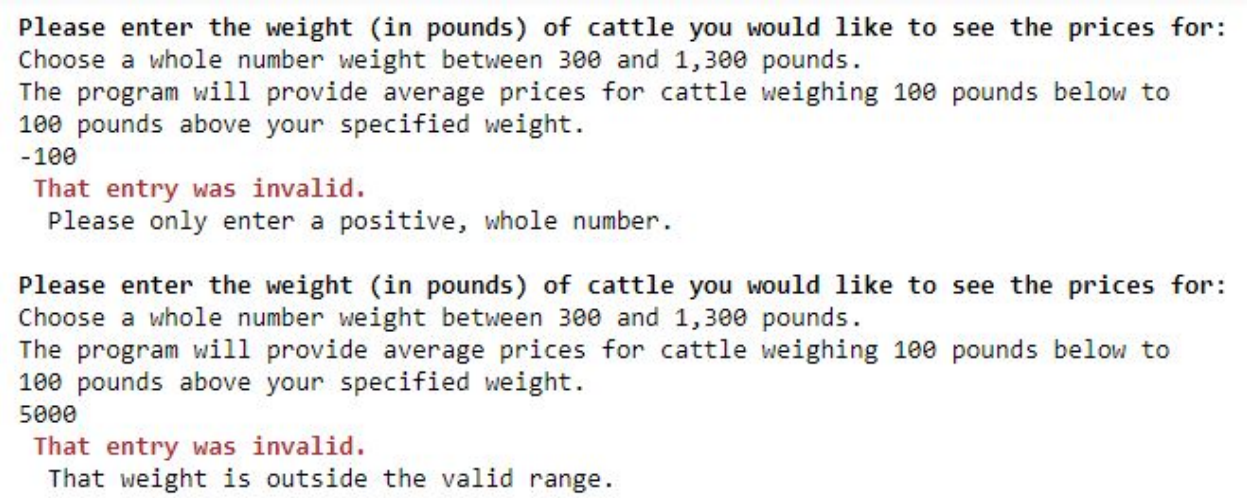
```
Please enter the grade of cattle you would like to see the prices for:
Choose a number between 1 and 4.
0
 That entry was invalid.
  That grade is not recognized.

Please enter the grade of cattle you would like to see the prices for:
Choose a number between 1 and 4.
%
 That entry was invalid.
  That grade is not recognized.

Please enter the grade of cattle you would like to see the prices for:
Choose a number between 1 and 4.
1

The grade you have chosen is: 1
```

Get Cattle Weight from User

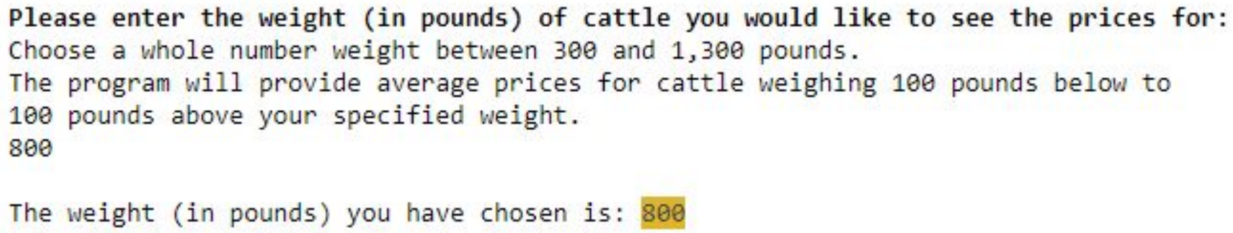


```
# while loop to get and validate WEIGHT user input
weight = 0 # to start loop
while weight == 0:
    # get weight from user and inform them of requirements
    make_bold('\nPlease enter the weight (in pounds) of cattle you would like to see the prices for:')
    print('Choose a whole number weight between 300 and 1,300 pounds.')
    print('The program will provide average prices for cattle weighing 100 pounds below to')
    print('100 pounds above your specified weight.')
    weight = input()
    # to validate input
    if weight.isnumeric(): # if weight is numeric
        if int(weight) in np.arange(300, 1301): # if weight in range 300-1301, not including 1301
            print('\nThe weight (in pounds) you have chosen is: ', end='') # confirm the weight user has chosen
            highlight_echo(weight)
        else: # if weight not in range
            mark_invalid('That weight is outside the valid range.') # inform the user of invalid entry
            weight = 0 # reset the variable to start the loop over
    else: # if weight is not numeric
        mark_invalid('Please only enter a positive, whole number.') # inform the user of invalid entry
        weight = 0 # reset the variable to start the loop over
```

This section of code retrieves the cattle weight from the user. The `weight` variable is initiated to 0 to begin the while loop. The program then prompts the user to enter the weight of cattle they are interested in pricing along with the valid range of weights. User input is saved in `weight` variable. If the entered weight is numeric, the program checks if it exists in a range between 300 and 1,300. If so, an affirmative message is printed and the weight is echoed back to the user highlighted in yellow. If the entered grade is not in the range, the program prints an error message and the `weight` variable is reset for the while loop to continue. If the entered weight is not numeric, a different error message is printed and the while loop continues until a valid grade is entered.

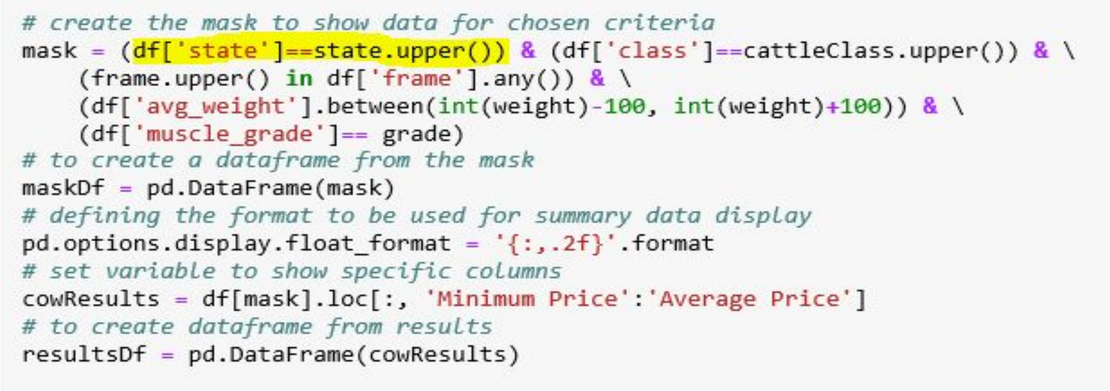


```
Please enter the weight (in pounds) of cattle you would like to see the prices for:
Choose a whole number weight between 300 and 1,300 pounds.
The program will provide average prices for cattle weighing 100 pounds below to
100 pounds above your specified weight.
-100
 That entry was invalid.
  Please only enter a positive, whole number.

Please enter the weight (in pounds) of cattle you would like to see the prices for:
Choose a whole number weight between 300 and 1,300 pounds.
The program will provide average prices for cattle weighing 100 pounds below to
100 pounds above your specified weight.
5000
 That entry was invalid.
  That weight is outside the valid range.
```



```
Please enter the weight (in pounds) of cattle you would like to see the prices for:
Choose a whole number weight between 300 and 1,300 pounds.
The program will provide average prices for cattle weighing 100 pounds below to
100 pounds above your specified weight.
800

The weight (in pounds) you have chosen is: 800
```

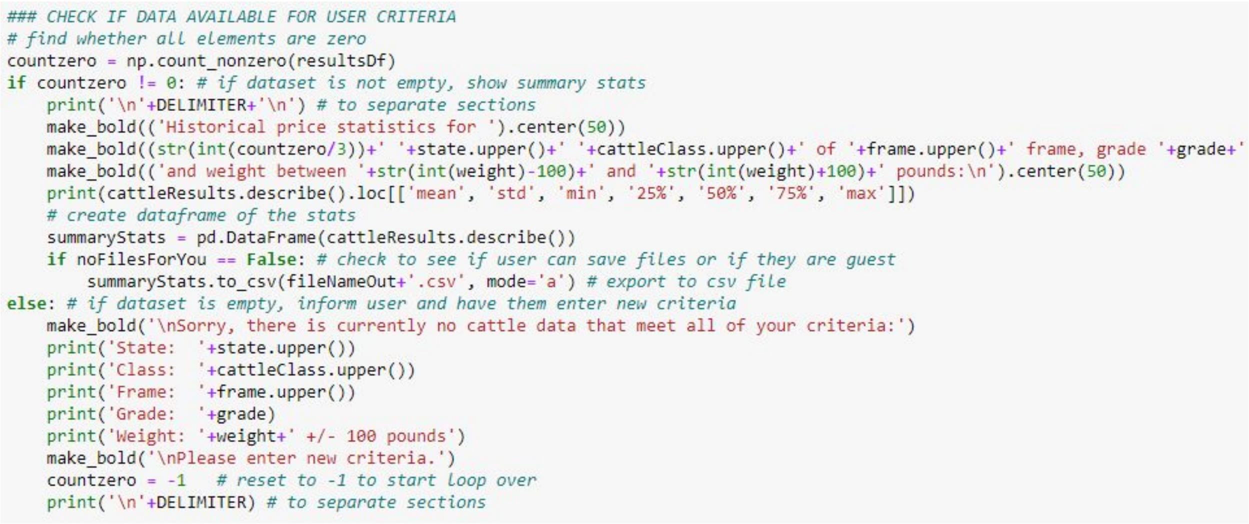
Create a Mask



```
# create the mask to show data for chosen criteria
mask = (df['state']==state.upper()) & (df['class']==cattleClass.upper()) & \
    (frame.upper() in df['frame'].any()) & \
    (df['avg_weight'].between(int(weight)-100, int(weight)+100)) & \
    (df['muscle_grade']== grade)
# to create a dataframe from the mask
maskDf = pd.DataFrame(mask)
# defining the format to be used for summary data display
pd.options.display.float_format = '{:,.2f}'.format
# set variable to show specific columns
cowResults = df[mask].loc[:, 'Minimum Price':'Average Price']
# to create dataframe from results
resultsDf = pd.DataFrame(cowResults)
```

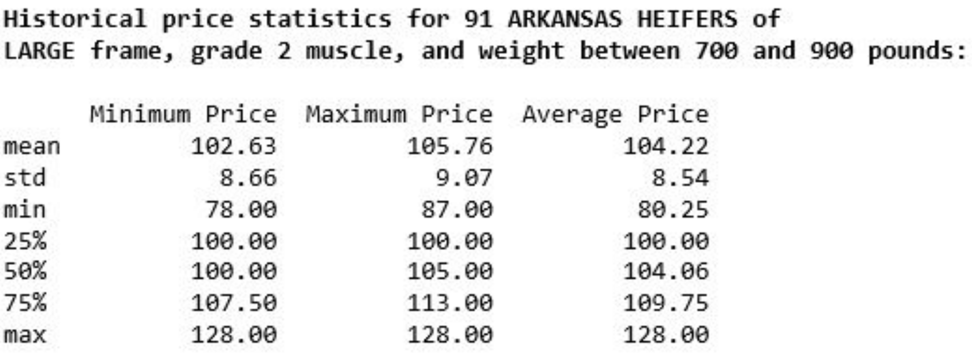
After asking the user what information he or she would like to view the program now goes ahead and pulls these specified column from the csv file. First, we call out the variable name of the `mask` to know that these are the columns that we want. Further on we tell the variable to go to the state column by calling the data frame name which is ‘df’ and then the column name in brackets (see highlighted snipped above). After the brackets we are placing two equals signs and then the `state` variable that we asked the user to input saying that if the state column has an equal value then the one the user requested, to pull these out and display them. Followed by the state request we will also do the same process for the class, frame, average weight and muscle grade. Now that the `mask` variable is created, the program goes ahead and makes out of the mask its own data frame which is formatted and displayed to the user that requested the information. Further on, another variable is created adding two specified columns which will show the ‘Minimum Price’ and ‘Average Price’ based on the selected results, this variable is called `cowResults`. Once again, the mask data frame that we created has now two new columns on display and therefore a new dataframe called `resultsDf` needs to be created including the new values that were added.

Historical Price statistics



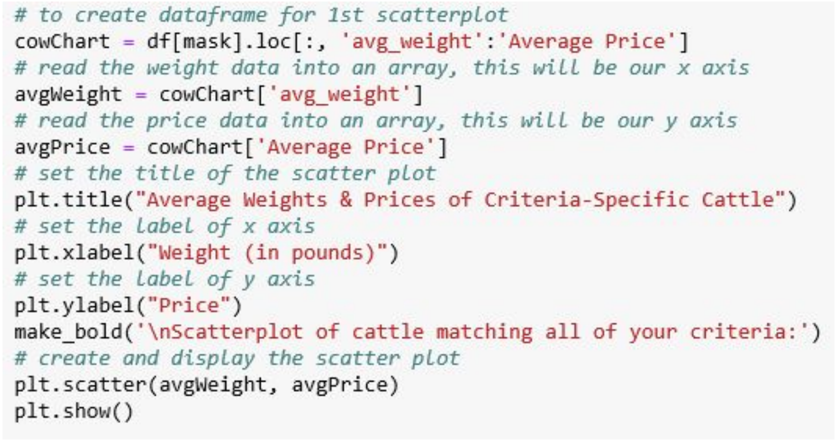
```
### CHECK IF DATA AVAILABLE FOR USER CRITERIA
# find whether all elements are zero
countzero = np.count_nonzero(resultsDf)
if countzero != 0: # if dataset is not empty, show summary stats
    print('\n'+DELIMITER+'\n') # to separate sections
    make_bold(('Historical price statistics for ').center(50))
    make_bold((str(int(countzero/3))+' '+state.upper()+' '+cattleClass.upper()+' of '+frame.upper()+' frame, grade '+grade+'
    make_bold(('and weight between '+str(int(weight)-100)+' and '+str(int(weight)+100)+' pounds:\n').center(50))
    print(cattleResults.describe().loc[['mean', 'std', 'min', '25%', '50%', '75%', 'max']])
    # create dataframe of the stats
    summaryStats = pd.DataFrame(cattleResults.describe())
    if noFilesForYou == False: # check to see if user can save files or if they are guest
        summaryStats.to_csv(fileNameOut+'.csv', mode='a') # export to csv file
else: # if dataset is empty, inform user and have them enter new criteria
    make_bold('\nSorry, there is currently no cattle data that meet all of your criteria:')
    print('State:  '+state.upper())
    print('Class:  '+cattleClass.upper())
    print('Frame:  '+frame.upper())
    print('Grade:  '+grade)
    print('Weight: '+weight+' +/- 100 pounds')
    make_bold('\nPlease enter new criteria.')
    countzero = -1   # reset to -1 to start loop over
    print('\n'+DELIMITER) # to separate sections
```

On the image above, you can see now, that we have our `mask` variable that was explained in the previous section, the program will actually check if there is information available within those columns that we requested information from. In this instance, a variable was created that counts the values that are not empty to then determine if the program is able to show that information. If the program does find the values requested without any empty values, all information and price averages ,mean max min and standard deviation will be provided and placed into a csv file for the user to download and open. If for any other reason the program does find empty values and there is no information for the users specified requests, the program will display an error message displaying that currently there is no data available and therefore to enter new values to search for.



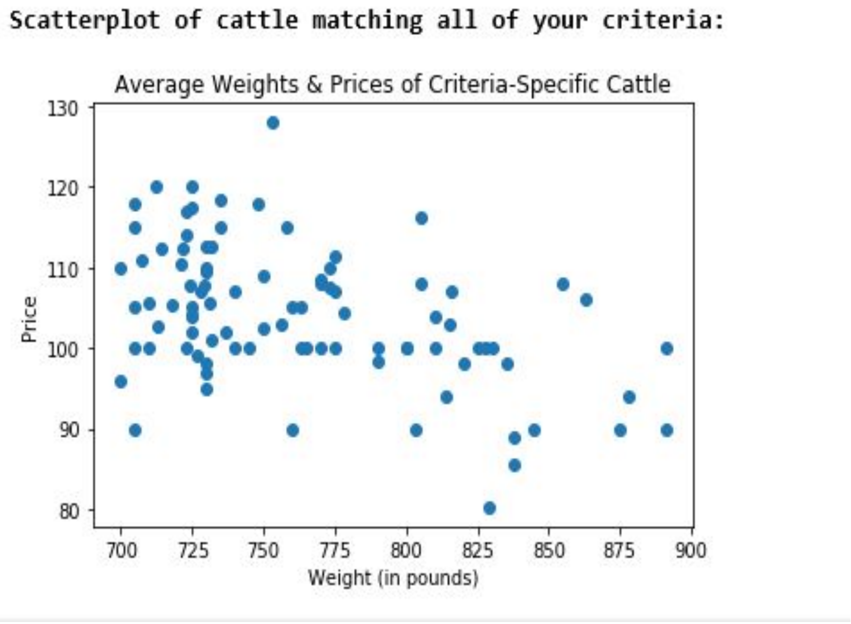
```
Historical price statistics for 91 ARKANSAS HEIFERS of
LARGE frame, grade 2 muscle, and weight between 700 and 900 pounds:

      Minimum Price  Maximum Price  Average Price
mean         102.63         105.76         104.22
std            8.66           9.07           8.54
min           78.00          87.00          80.25
25%          100.00         100.00         100.00
50%          100.00         105.00         104.06
75%          107.50         113.00         109.75
max          128.00         128.00         128.00
```

Nr. 1 Scatter Plot Data Frame



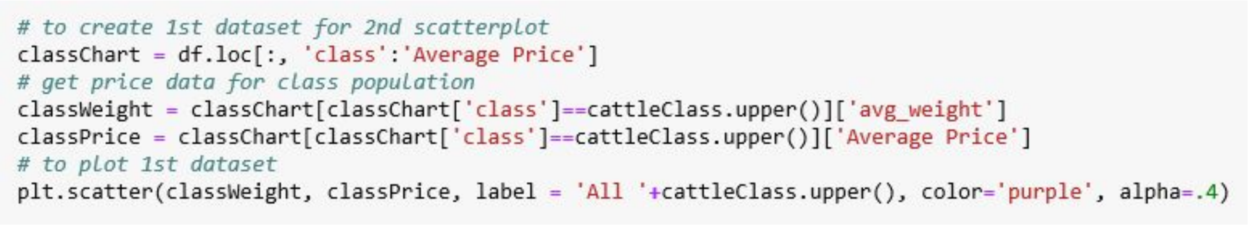
```
# to create dataframe for 1st scatterplot
cowChart = df[mask].loc[:, 'avg_weight':'Average Price']
# read the weight data into an array, this will be our x axis
avgWeight = cowChart['avg_weight']
# read the price data into an array, this will be our y axis
avgPrice = cowChart['Average Price']
# set the title of the scatter plot
plt.title("Average Weights & Prices of Criteria-Specific Cattle")
# set the label of x axis
plt.xlabel("Weight (in pounds)")
# set the label of y axis
plt.ylabel("Price")
make_bold('\nScatterplot of cattle matching all of your criteria:')
# create and display the scatter plot
plt.scatter(avgWeight, avgPrice)
plt.show()
```

Once the program outputs the requested information the program will in addition display a scatter plot showing the correlations between the cow weights and their prices. Starting with the first plot, a variable has to be created that contains the information that wants to be used for the scatter plot. In the image above the `cowChart` variable is the variable that contains the average weight information and the average price. Now that the program has the specific information that it needs, another two variables will be created to separate the `cowChart` variable into their own little data frames. Once we have the average weight and price column with their individual variable, the title, X and Y labels are defined to be reflected on the scatter plot. Lastly, we call out the scatterplot function from the pyplot module and place our individualized variables that we want to display, followed by the show function which will show the user the actual scatter plot matching all of the users criteria.



```
Scatterplot of cattle matching all of your criteria:
```

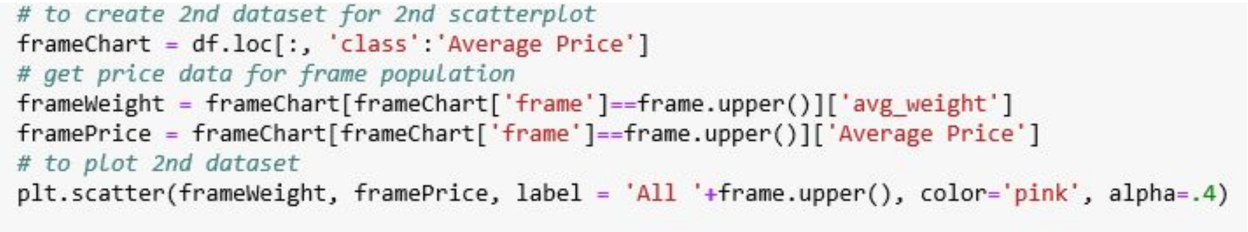
Nr. 2 Scatter Plot 1 data set



```
# to create 1st dataset for 2nd scatterplot
classChart = df.loc[:, 'class':'Average Price']
# get price data for class population
classWeight = classChart[classChart['class']==cattleClass.upper()]['avg_weight']
classPrice = classChart[classChart['class']==cattleClass.upper()]['Average Price']
# to plot 1st dataset
plt.scatter(classWeight, classPrice, label = 'All '+cattleClass.upper(), color='purple', alpha=.4)
```

The second scatter plot requires a similar process as the first scatter plot. At first, a variable needs to be created by calling the `loc` convention to select the rows within our data frame which are ‘class’ and ‘Average Price’.

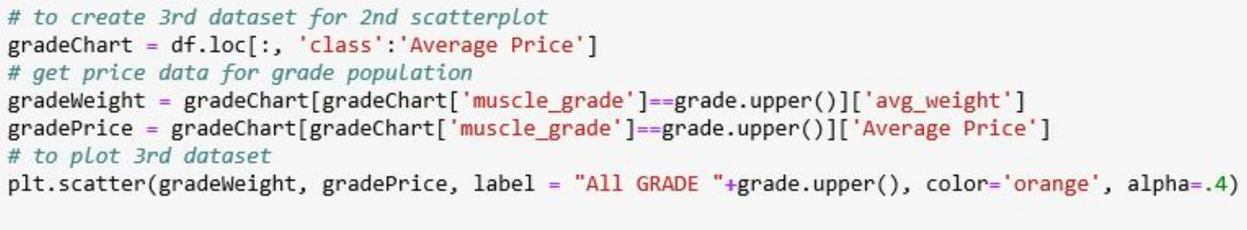
Now, two more variables need to be created in order to find the correlation between the class, average weight and average price. The `classWeight` and `classPrice` variables look if there is a specific class in the data set `classChart` that we just created equal to the `cattleClass` that the user has inputted. After the program finds the requested class types the program bases the scatter plot on specifically the average price and average weight. At last, the plot will be created by entering the `classWeight` variable and the `classPrice` for it to show the correlation between them followed by the label outputs, the colors and the dim of the colors.



```
# to create 2nd dataset for 2nd scatterplot
frameChart = df.loc[:, 'class':'Average Price']
# get price data for frame population
frameWeight = frameChart[frameChart['frame']==frame.upper()]['avg_weight']
framePrice = frameChart[frameChart['frame']==frame.upper()]['Average Price']
# to plot 2nd dataset
plt.scatter(frameWeight, framePrice, label = 'All '+frame.upper(), color='pink', alpha=.4)
```

Continuing our scatter plot, another two variables need to be created by calling the `loc` convention to select the rows within our data frame which are ‘class’ and ‘Average Price’ which basically means that the information that is extracted in this section will be added to the second scatter plot.

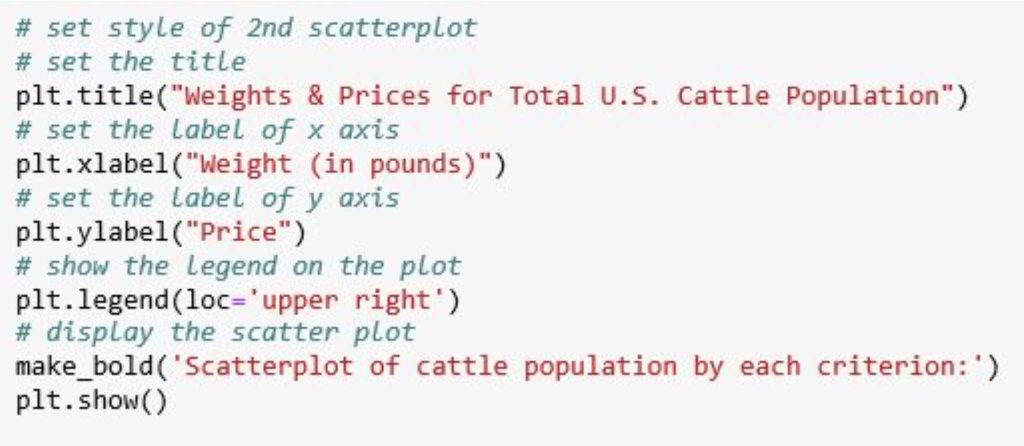
Now, two more variables need to be created in order to find the correlation between the frame, average weight and average price. The `frameWeight` and `framePrice` variables look if there is a specific class in the data set `frameChart` that we just created equal to the frame that the user has inputted. After the program finds the requested frame types the program bases the scatter plot on specifically the average price and average weight. At last, the plot will be created by entering the `frameWeight` and the `framePrice` variable for it to show the correlation between them followed by the label outputs, the colors and the dim of the colors.



```
# to create 3rd dataset for 2nd scatterplot
gradeChart = df.loc[:, 'class':'Average Price']
# get price data for grade population
gradeWeight = gradeChart[gradeChart['muscle_grade']==grade.upper()]['avg_weight']
gradePrice = gradeChart[gradeChart['muscle_grade']==grade.upper()]['Average Price']
# to plot 3rd dataset
plt.scatter(gradeWeight, gradePrice, label = "All GRADE "+grade.upper(), color='orange', alpha=.4)
```

The second to last part of our scatter plot follows the creation of third data set by calling the loc convention to select the rows within our data frame which are ‘class’ and ‘Average Price’ which basically means that the information that is extracted in this section will be added to the already existing information on the second scatter plot.

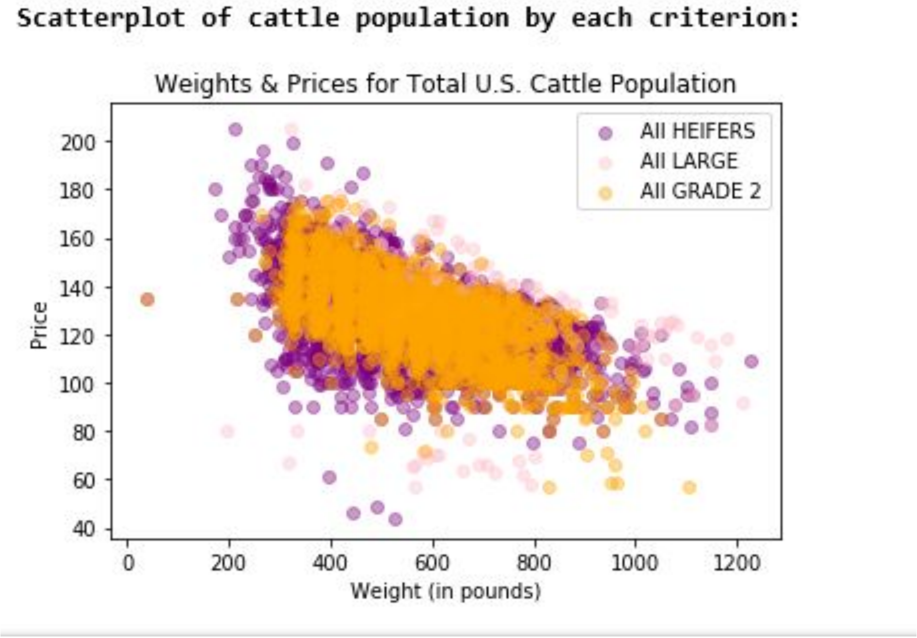
Again, two more variables need to be created in order to find the correlation between the grade, average weight and average price. The `gradeWeight` and `gradePrice` variables look if there is a specific class in the data set `gradeChart` that we just created equal to the grade that the user has inputted. After the program finds the requested frame types the program bases the scatter plot on specifically the average price and average weight. At last, the plot will be created by entering the `gradeWeight` and the `gradePrice` variable for it to show the correlation between them, followed by the label outputs, the colors and the dim of the colors.



```
# set style of 2nd scatterplot
# set the title
plt.title("Weights & Prices for Total U.S. Cattle Population")
# set the label of x axis
plt.xlabel("Weight (in pounds)")
# set the label of y axis
plt.ylabel("Price")
# show the legend on the plot
plt.legend(loc='upper right')
# display the scatter plot
make_bold('Scatterplot of cattle population by each criterion:')
plt.show()
```

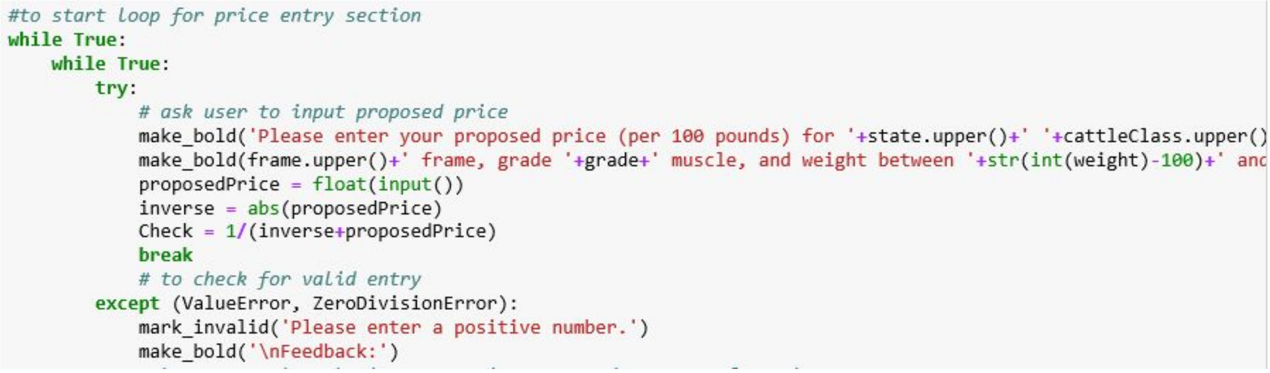
With the gathering of all the grade, class and frame information for the second scatter plot we now need to define the tile, the legend and the Y and X labels as shown in the snipped above. Followed by placing the ‘Scatter Plot of cattle population by each criterion’ string value inside the `make_bold` function to make the text stand out.

The final result of the second scatter plot should look like this:



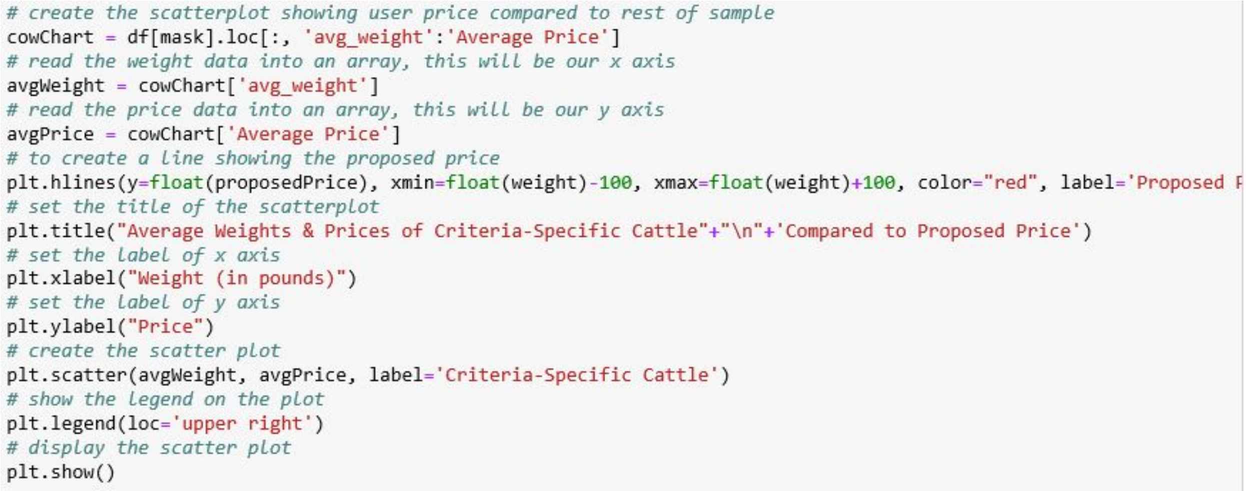
```
Scatterplot of cattle population by each criterion:
```

3rd Scatter Plot



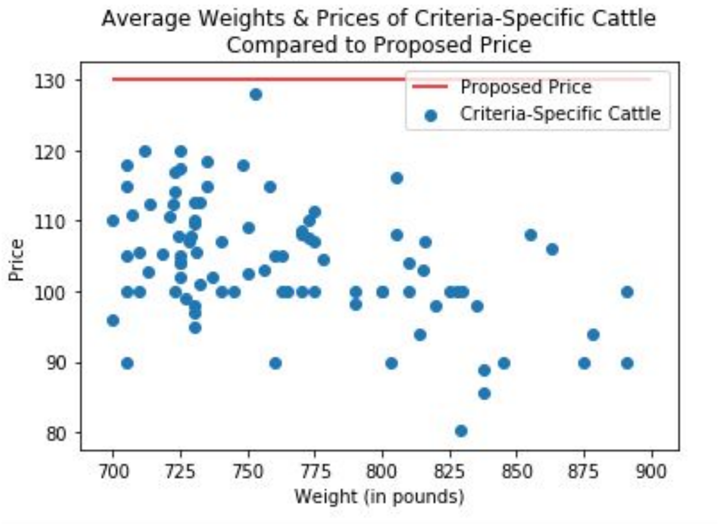
```
#to start loop for price entry section
while True:
    while True:
        try:
            # ask user to input proposed price
            make_bold('Please enter your proposed price (per 100 pounds) for '+state.upper()+' '+cattleClass.upper()
            make_bold(frame.upper()+' frame, grade '+grade+' muscle, and weight between '+str(int(weight)-100)+' anc
            proposedPrice = float(input())
            inverse = abs(proposedPrice)
            Check = 1/(inverse+proposedPrice)
            break
            # to check for valid entry
        except (ValueError, ZeroDivisionError):
            mark_invalid('Please enter a positive number.')
            make_bold('\nFeedback:')
```

On this scatterplot the program is starting off with two wile true statements. The first loop makes the initial question to the user to input the proposed price for their selections. If in any case there is an error message the user will be asked to enter another value that does not go below or equal to 0. Yet, if the proposed price is valid the program will continue and break out of that loop to jump into the still remaining ‘while True’ loop.

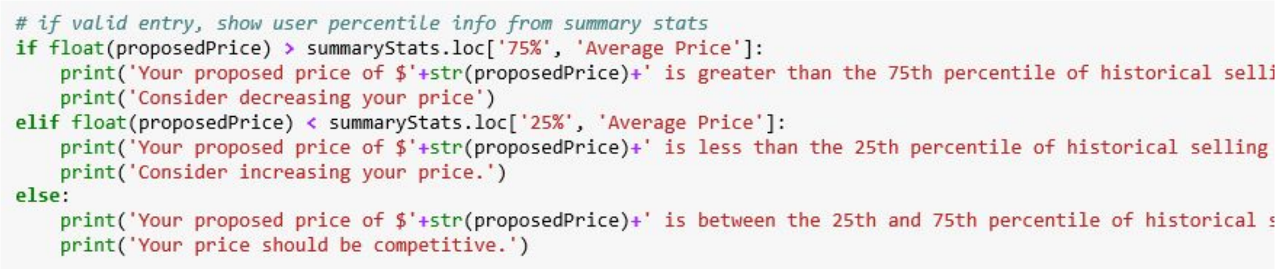


```
# create the scatterplot showing user price compared to rest of sample
cowChart = df[mask].loc[:, 'avg_weight':'Average Price']
# read the weight data into an array, this will be our x axis
avgWeight = cowChart['avg_weight']
# read the price data into an array, this will be our y axis
avgPrice = cowChart['Average Price']
# to create a line showing the proposed price
plt.hlines(y=float(proposedPrice), xmin=float(weight)-100, xmax=float(weight)+100, color="red", label='Proposed F
# set the title of the scatterplot
plt.title("Average Weights & Prices of Criteria-Specific Cattle"+"\n"+'Compared to Proposed Price')
# set the label of x axis
plt.xlabel("Weight (in pounds)")
# set the label of y axis
plt.ylabel("Price")
# create the scatter plot
plt.scatter(avgWeight, avgPrice, label='Criteria-Specific Cattle')
# show the legend on the plot
plt.legend(loc='upper right')
# display the scatter plot
plt.show()
```

After inputting the requested price for the cattle, the program will display the scatter plot based on the chosen criteria that the user entered and will additionally display the price line showing if the users entered price is above, below or between the 25th, 50th or 75th percentile. To initialize the plot, a variable needs to be created. The `cowChart` contains the average weight and the average price information that we used out of the main data frame, `df,` to place it into a mask. Now that we have the average price and weight column within the `cowChart` data frame another two variables are going to be created to individualize each column. These variables are the `avgWeight` and `avgPrice` which now have each one column. Lastly, we call out the scatterplot function from the pyplot module and place the `avgWeight` and `avgPrice` variable that we want to display, followed by the show function which will show the user the actual scatter plot matching all of the users criteria. Followed by the variables the the title and the X and Y coordinates are given a title. We can see that the scatter plot is exactly the same as the first scatter plot, the difference is that now the scatter plot shows the proposed price of the user as a red line. The line is showing where the users proposed cattle prices ranges in. The scatter plot would look like this:



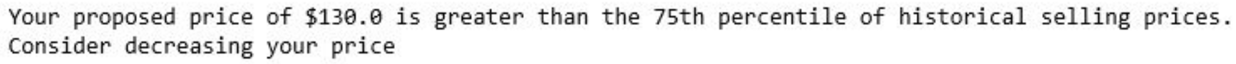
Adding on to this section of the remaining while loop the program recommends the user to increase or decrease the price based on the given averages or whether the price is competitive or not. See below:



```
# if valid entry, show user percentile info from summary stats
if float(proposedPrice) > summaryStats.loc['75%', 'Average Price']:
    print('Your proposed price of $'+str(proposedPrice)+' is greater than the 75th percentile of historical selli
    print('Consider decreasing your price')
elif float(proposedPrice) < summaryStats.loc['25%', 'Average Price']:
    print('Your proposed price of $'+str(proposedPrice)+' is less than the 25th percentile of historical selling
    print('Consider increasing your price.')
else:
    print('Your proposed price of $'+str(proposedPrice)+' is between the 25th and 75th percentile of historical s
    print('Your price should be competitive.')
```

This part of the code is checking if the proposed price of the user is greater than the 75th percentile to then recommend the user to decrease the price. If the price is less than the 25th percentile then the program will let the user know that he or she should increase the price to be within the competitive prices that are within the 25th and the 75th percentile. Lastly, if the proposed price is within the 25th and 75th percentile the program will tell the user that the price is competitive.

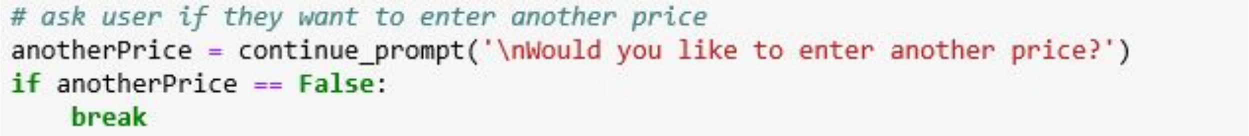
See below:



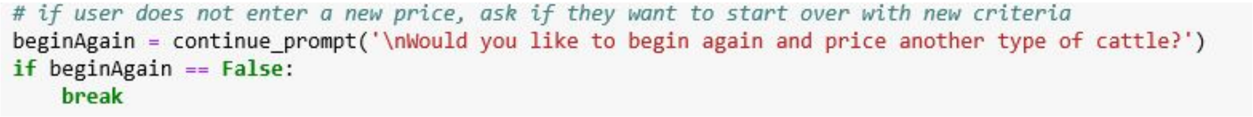
```
Your proposed price of $130.0 is greater than the 75th percentile of historical selling prices.
Consider decreasing your price
```

Restarting the program

At this final point, after the program suggests the user its recommendations the program will ask the user if they want to enter a new price for the same criteria that they previously selected.



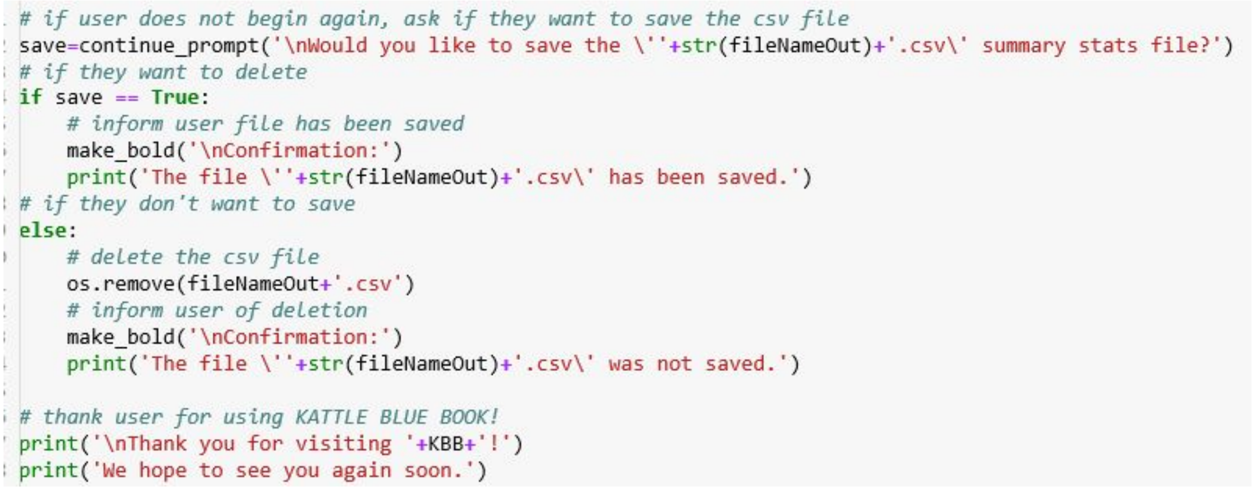
```
# ask user if they want to enter another price
anotherPrice = continue_prompt('\nWould you like to enter another price?')
if anotherPrice == False:
    break
```



```
# if user does not enter a new price, ask if they want to start over with new criteria
beginAgain = continue_prompt('\nWould you like to begin again and price another type of cattle?')
if beginAgain == False:
    break
```

If the user says yes the program will reinitiate the while true loop where they are asked to enter a new price, if the user does not want to enter a new price the program will again ask the user if they would like to restart the whole program with all new values and criteria.

Saving Files



```
# if user does not begin again, ask if they want to save the csv file
save=continue_prompt('\nWould you like to save the \''+str(fileNameOut)+'.csv\' summary stats file?')
# if they want to delete
if save == True:
    # inform user file has been saved
    make_bold('\nConfirmation:')
    print('The file \''+str(fileNameOut)+'.csv\' has been saved.')
# if they don't want to save
else:
    # delete the csv file
    os.remove(fileNameOut+'.csv')
    # inform user of deletion
    make_bold('\nConfirmation:')
    print('The file \''+str(fileNameOut)+'.csv\' was not saved.')

# thank user for using KATTLE BLUE BOOK!
print('\nThank you for visiting '+KBB+'!')
print('We hope to see you again soon.')
```

The last portion of the entire program is to see if the user would like to save the results in a csv file or delete it.The conditions the program uses are if the user enters ‘no’ the output will be that the file has not been saved or that it has been saved if the user enters ‘yes’. Regardless of the answer of the user towards the condition the program will thank the user and wish them well.

KATTLE BLUE BOOK

Response Document

Following the initial project presentation given to the MIS 740 Software Concepts class, we have received and reviewed the feedback provided us by the class. The comments can be fitted into the following three broad categories: General Compliments, Presentation Suggestions, and Application Inquiries and Suggestions. In total we received 44 comments, to all of which we replied within the Canvas portal. Some comments will show up in multiple groupings as it had components of multiple categories within the text.

Thirty of the class comments included elements of general compliments on both the project concept and presentation, this was by far the largest group. Followed by 13 suggestions that the presentation be reduced in terms of content. This suggestion represents an opportunity for our group to focus on brevity and graphical representation of topics in our final presentation.

In addition to the volume of content, the broader group of presentation suggestions contained two other subgroups - Presentation Flow and Suggested Content on which to focus - though there was only one suggestion per subgroup. These suggestions included a preference to devote a slide to the entire flow chart, followed up by branching off from the flowchart for the presentation. There is merit to this suggestion, however, because it only showed up once, and is itself a suggestion of preference that would include a rework of the entire flow of the presentation, we have decided to not take an entirely new approach, but have made note of the suggestion and where applicable will provide case specific insight into the flow chart in our next presentation.

The final suggestion within the Presentation Suggestion category mentioned a need for more emphasis on user/business value. Again, this commentary only showed up once, but it is an important enough topic that we will be certain to highlight user/business value in our final presentation.

The final category of suggestions, Application Suggestions and Input, included only three suggestions, with no overlapping suggestions from the contributors. The first inquiry focused on how we planned to clean the data. This was a question we were not entirely clear upon ourselves at the time of the initial presentation. We have since learned cleaning techniques during the final weeks of class that we have implemented, such as replacing blanks and NaNs with zeros, dropping columns and observations, and masking data. The second comment within this category suggested the use of a GUI, specifically because the intended audience for this project may not be the most technically experienced. Interestingly, we believe a well-designed command line interface may actually serve this audience better, since a GUI is itself inherently technical, a non-technical user might feel more comfortable within a command line interface where each step is explained in plain text. Additionally, because this is a proof of concept, we focused more directly on the concepts taught within this course and did not focus on a GUI, though for a finalized product to take to market, a well-designed GUI would be high on the list of items to add. The final piece of commentary inquired if the user would be able to alter data and if predictive models would be in place. With respect to the user altering data, that would actually introduce inaccurate data into the information system and therefore should be avoided. The purpose of this program is to provide USDA historical data to the end user, which would then serve as a tool for the Auction House (end user) to rely upon when making minimum price setting decisions, therefore we explicitly do not want this information changed. With respect to predictive models, while certainly useful, they fall outside of the scope of this proof of concept. A truly valuable predictive model is a time intensive undertaking to develop and test. If this project is developed beyond a proof of concept, predictive modeling would add value, but for the purposes of this exercise we have decided to focus on the components more closely aligned with this particular class.

The following is a list of all the suggestions we received based on our initial presentation. Each comment has been reviewed and responded to within Web Campus. Additionally, we have weighted most heavily the comments that showed up with the greatest regularity, specifically the density of content within the presentation.

General Compliments

1) It’s interesting to use an application related to agricultural business, the functions were well explained.
2) The flowcharts are very comprehensive for each of the functions mentioned. All inputs and outputs were clearly stated. Great job!
3) The ppt is too in detail. The data dictionary is clear and nice. The flowcharts are clear. It is nice to have example of the interface
4) I enjoy how unique the program idea is and how helpful it is to the niche market. I like how flow charts are displayed for each function. Overall good work.
5) The presentation was very informative on cattle and was clear with its purpose. They went in depth with their functions being used as well as touching on challenges they might face.
6) The program design looks pretty strong and complete
7) Interesting application in the sense that most people do not mix agriculture sales/pricing/marketing with technology. The name of the application is very creative and probably should be patented quickly as this application can probably be put into a real production platform.
8) Very good presentation seems like a very useful program. Will look forward to seeing it work.
9) First two slides full of great information, but it’s all text, would suggest a few bullets and pictures. Application Flow chart really good but small and hard to see on the screen. Good data dictionary. Function flow charts are good but hard to see. Lots of text on the screen, I feel like the audience is reading and not listening. Everybody is very knowledgeable. Very strong idea and use of python. Command line interface is cool
10) PPT gave enough information and showed clear flow charts. everyone's voices are loud and clear. The group managed time well.

11) Introduction explanation clear but there is too much information on the slides to follow. Flow chart contains too much information that easy to follow. Dictionary is clear. Interface nicely done. Time management good

12) Very graphical! Tons of images and examples of how they will use the application.

13) I think your program is very useful, I think it can really be of great help for the business. Your flowcharts look very clear and functions are very descriptive. Great job!

14) Introduction - Good explanation of overall idea. I like the way you broke it down. Flow Chart - The actual chart was hard to read. I would like to have seen the entire flow chart on one slide and have you expound on that. Good breakdown of the data dictionary and functions that you created. Interesting how you based the parameters around the mean. Good explanation of the interface and the fulfillment of requirements. The challenges and opportunities section was interesting and a great idea.

15) I really liked the design of the slides; it matched the theme of the application. The application flow was really good. Very unique functions. The mock-up of the command line was very helpful, gave us a good idea of what to expect.

16) The idea is attractive. The flow chart is logical, but it is a little bit small. Data Dictionary is good to explain the database design. Challenge and opportunities part give me a very clear understanding for the whole system.

17) The flow chart looks good. I liked how the interface was designed. Very organized presentation. It seemed they rehearsed before the presentation. Great presentation overall.

18) Seems like you are off to a strong start. Very in-depth presentation of the functions was informative. Interesting idea for a project. Good luck with cleaning the data to make your program stronger.

19) A unique topic and well-prepared presentation! Great explanation in functions and calculation process. Great job!

20) Great presenters. I like how they took their complex concepts and simplified them. I enjoyed seeing the mock interface and their application automation.

21) Great introduction to explain the application and how it will solve an issue in the cattle industry. Flowchart is explained nicely. Data dictionary is organized and well-explained.

I was impressed how they explained the process of the functions in the application. They seem to be very organized and articulate in their presentation. Great work.

22) comparing to the challenges, the opportunities have bigger space to implement. and have good time management in the team just in time.

23) The presentation was well executed, and the topic was unique with regards to the system proposal. They clearly defined project goals and problems. Nobody used any notecards and all students spoke with clarity and confidence. Also the presentation had numerous graphics which helped to explain the design of the system.

24) Well presented with details. The process flow was good. Good job overall.

25) Functions section & explaining each function with a flowchart was a very cool idea. Good luck :)

26) Excellent flowchart design. It was easy to follow and understand the logic.

27) use real data can connect real world operation more closely, and this team make it happened. layout of presentation is organized and easy to understand. Clear explanation for audience, especially the topic is not that popular.

28) I can easily follow the coding which is quite clear and neat. Group 6 has designed a great concepts of application design. Great job!!

29) The presentation was neat and professional. The slides were concise and contained relevant information. The ideas were clearly presented.

30) Do not know too much about cattle but it looks like it can be very useful. Great job of defining the inputs, outputs and functions of the application

Presentation Suggestions

Amount of Content within Slides

1) Like the idea about the cattle grade. The application flow chart and databases are very clear. Make sure not showing too much content in the ppt, should use short words and sentences instead of paragraphs. Overall, the presentation is excellent.
2) The ppt is too in detail. The data dictionary is clear and nice. The flowcharts are clear. It is nice to have example of the interface
3) Slides have a lot of words which is a bit overwhelming. The presentation was very detailed, and the topic was unique.
4) Unique idea. I recommend using bullet points next time. Nice command line view of the program.
5) First two slides full of great information, but it's all text, would suggest a few bullets and pictures. Application Flow chart really good but small and hard to see on the screen. Good data dictionary. Function flow charts are good but hard to see. Lots of text on the screen, I feel like the audience is reading and not listening. Everybody is very knowledgeable. Very strong idea and use of python. Command line interface is cool
6) Introduction explanation clear but there is too much information on the slides to follow. Flow chart contains too much information that easy to follow. Dictionary is clear. Interface nicely done. Time management good
7) The slides contain too much text. great to see the technical perspective but you need to engage the audience with it. Slides can be reduced to a mix of technical data. We are getting too technical in this. Great to have a challenge and opportunities slide.
8) Nice opening. The slides are a little too busy though. Impressive user interface presentation. Requirements fulfillment should have talked about how the user requirements would be met instead of the project requirements asked by the professor.

9) The idea is attractive. The flow chart is logical, but it is a little bit small. Data Dictionary is good to explain the database design. Challenge and opportunities part give me a very clear understanding for the whole system.
10) The presentation was smooth and hits the requirements. My one suggestion is to use less words on the power point.
11) Very wordy slides try to incorporate visual representations of what you are saying into the presentation. Could have condensed the descriptions of the functions. Like the challenges and opportunities section would have worked grouped into the descriptions of the functions
12) The introduction is a little bit redundant. The variables and constants are clear.
13) The slides have too many words. I would rather want the members to say them instead of placing on the slides. The design is very consolidated. Great job!

Presentation Flow

1) Introduction - Good explanation of overall idea. I like the way you broke it down. Flow Chart - The actual chart was hard to read. I would like to have seen the entire flow chart on one slide and have you expound on that. Good breakdown of the data dictionary and functions that you created. Interesting how you based the parameters around the mean. Good explanation of the interface and the fulfillment of requirements. The challenges and opportunities section was interesting and a great idea.

Suggested Content on which to Focus

1) If could spend more time on customer needs and business values, that would be better.

Application Inquiries and Suggestions

1) Interesting project choice. Would love to hear more about why this project exists. How will you clean the data with in large volumes.

2) This is an interesting 'problem' to solve with python -- I appreciate the uniqueness of the issue tackled by your program. Some of the presentation slides were extremely information-dense, though I understand that flow charts can only be made so simple in order to remain accurate representations of program complexity. User-friendliness in the GUI should be of particular concern with the target market in mind.

3) Only user input of price, user able to add or change any existing data