Project Name, Participants, and Workflow:

Project Name:

Time Series Analysis of Iowa Liquor Sales

Participants:

Andre Sharp 11374558 AndreSharp@my.unt.edu

Dan Waters 11457837 DanWaters@my.unt.edu

Bryan Adams 10976617 BryanAdams@my.unt.edu

Haiyi Wang 11528159 HaiyiWang@my.unt.edu

Sima Siami-Namini 11447869 SimaSiami-Namini@my.unt.edu

Workflow:

We will use Discord, email, and Zoom meetings to connect with each other. We will also have the weekly meeting on Saturday before the class.

We will use GitHub as the code version control. We can update the code and version on GitHub.

Project Abstract:

Motivation:

As we all know, retail businesses have a 'goldilocks' problem when it comes to inventory: don't stock too much, but don't stock too little. If we stock too much, it will occupy the huge amount of money and have the risk that products can't be sold due to the production time, consumer favor, and so on. If we stock too little, we will probably miss the time to sell the product and miss the big chance to earn money. So, we need to use historic data and suitable models to analyze and to predict the consumption of products in the future. It will help the business owner to make a good decision in advance.

Objectives:

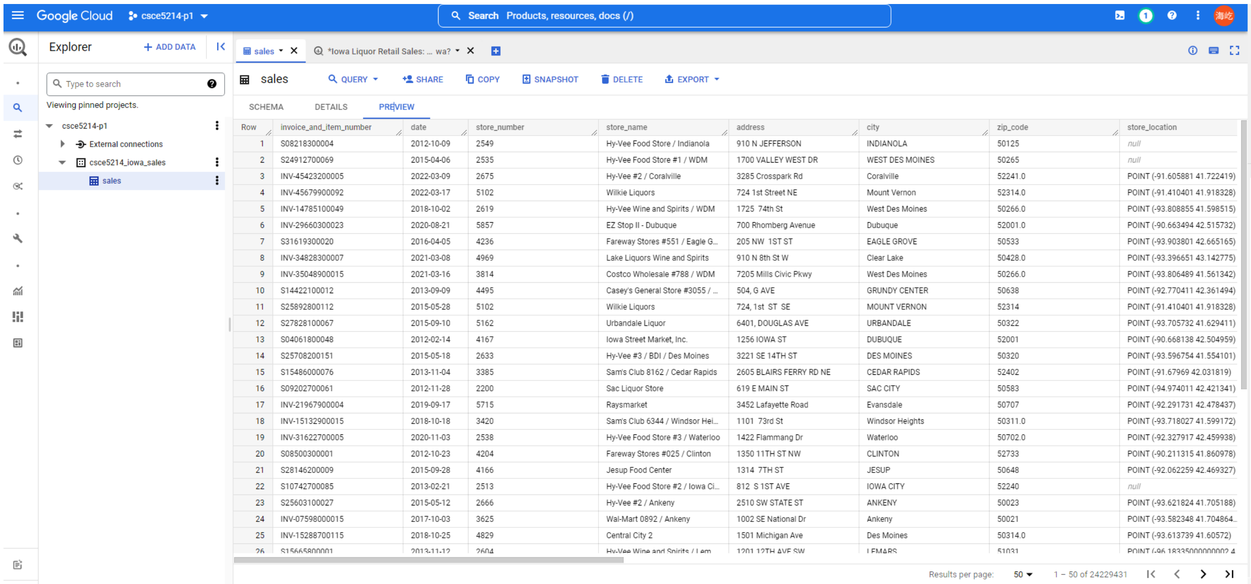
Our project is to use Iowa liquor sales data in multiple ways to analyze customer behavior for liquor consumption. As a brief bit of background, Iowa is an alcohol beverage control state, meaning that the state maintains a monopoly on wholesaling of alcohol throughout the State. Effectively, private retailers must purchase their alcohol from the state before selling it to individual consumers. We will use a time series model (ARIMA) to train the historic data and forecast the consumption in the future. All the computation and model deployment are on the google cloud. We will use Google BigQuery ML to extract the data and create a dashboard to show predicted metrics

Data specification:

Dataset:

The Iowa liquor retail sales dataset is available through Iowa's state-hosted open data portal[1]. This dataset contains every wholesale purchase of liquor in the State of Iowa by retailers for sale to individuals since January 1, 2012.The dataset has the liquor sale data including different type of store. Some are wholesale, some are grocery stores. There are 24 features and 2.7 million rows in the dataset. We will use the date, category_name, item_description, state_bottle_cost, state_bottle_retail, bottle_sold,sale_dollars .

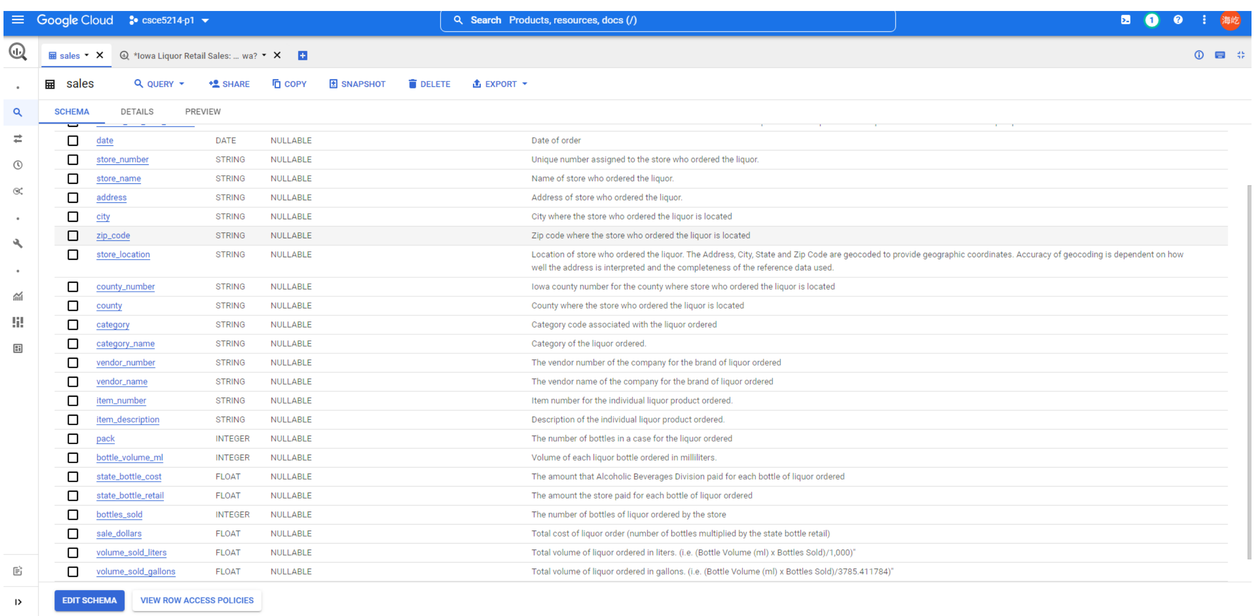
Because we use google cloud, The Iowa liquor dataset are stored in bigquery-public-data, we can write some query to get data that we want.



Row	invoice_and_item_number	date	store_number	store_name	address	city	zip_code	store_location
1	S08218300004	2012-10-09	2549	Hy-Vee Food Store / Indianola	910 N JEFFERSON	INDIANOLA	50125	null
2	S24912700069	2015-04-06	2535	Hy-Vee Food Store #1 / WDM	1700 VALLEY WEST DR	WEST DES MOINES	50265	null
3	INV-45423200005	2022-03-09	2675	Hy-Vee #2 / Coralville	3285 Crosspark Rd	Coralville	52241.0	POINT (-91.605881 41.722419)
4	INV-45679900092	2022-03-17	5102	Wilkie Liquors	724 1st Street NE	Mount Vernon	52314.0	POINT (-91.410401 41.918328)
5	INV-14785100049	2018-10-02	2619	Hy-Vee Wine and Spirits / WDM	1725 74th St	West Des Moines	50266.0	POINT (-93.808855 41.598515)
6	INV-29660300023	2020-08-21	5857	EZ Stop II - Dubuque	700 Rhomberg Avenue	Dubuque	52001.0	POINT (-90.663494 42.515732)
7	S31619300020	2016-04-05	4236	Fareway Stores #551 / Eagle G...	205 NW 1ST ST	EAGLE GROVE	50533	POINT (-93.903801 42.665165)
8	INV-34828300007	2021-03-08	4969	Lake Liquors Wine and Spirits	910 N 8th St W	Clear Lake	50428.0	POINT (-93.396651 43.142775)
9	INV-35048900015	2021-03-16	3814	Costco Wholesale #788 / WDM	7205 Mills Civic Pkwy	West Des Moines	50266.0	POINT (-93.806489 41.561342)
10	S14422100012	2013-09-09	4495	Casey's General Store #3055 / ...	504, G AVE	GRUNDY CENTER	50638	POINT (-92.770411 42.361494)
11	S25892800112	2015-05-28	5102	Wilkie Liquors	724, 1st ST SE	MOUNT VERNON	52314	POINT (-91.410401 41.918328)
12	S27828100067	2015-09-10	5162	Urbandale Liquor	6401, DOUGLAS AVE	URBANDALE	50322	POINT (-93.705732 41.629411)
13	S04061800048	2012-02-14	4167	Iowa Street Market, Inc.	1256 IOWA ST	DUBUQUE	52001	POINT (-90.668138 42.504959)
14	S25708200151	2015-05-18	2633	Hy-Vee #3 / BDI / Des Moines	3221 SE 14TH ST	DES MOINES	50320	POINT (-93.596754 41.554101)
15	S15486000076	2013-11-04	3385	Sam's Club 8162 / Cedar Rapids	2605 BLAIRS FERRY RD NE	CEDAR RAPIDS	52402	POINT (-91.67969 42.031819)
16	S09202700061	2012-11-28	2200	Sac Liquor Store	619 E MAIN ST	SAC CITY	50583	POINT (-94.974011 42.421341)
17	INV-21967900004	2019-09-17	5715	Raysmarket	3452 Lafayette Road	Evansdale	50707	POINT (-92.291731 42.478437)
18	INV-15132900015	2018-10-18	3420	Sam's Club 6344 / Windsor Hei...	1101 73rd St	Windsor Heights	50311.0	POINT (-93.718027 41.599172)
19	INV-31622700005	2020-11-03	2538	Hy-Vee Food Store #3 / Waterloo	1422 Flammang Dr	Waterloo	50702.0	POINT (-92.327917 42.459938)
20	S08500300001	2012-10-23	4204	Fareway Stores #025 / Clinton	1350 11TH ST NW	CLINTON	52733	POINT (-90.211315 41.860978)
21	S28146200009	2015-09-28	4166	Jesup Food Center	1314 7TH ST	JESUP	50648	POINT (-92.062259 42.469327)
22	S10742700085	2013-02-21	2513	Hy-Vee Food Store #2 / Iowa Ci...	812 S 1ST AVE	IOWA CITY	52240	null
23	S25603100027	2015-05-12	2666	Hy-Vee #2 / Ankeny	2510 SW STATE ST	ANKENY	50023	POINT (-93.621824 41.705188)
24	INV-07598000015	2017-10-03	3625	Wal-Mart 0892 / Ankeny	1002 SE National Dr	Ankeny	50021	POINT (-93.582348 41.704864...
25	INV-15288700115	2018-10-25	4829	Central City 2	1501 Michigan Ave	Des Moines	50314.0	POINT (-93.613739 41.60572)
26	S15665800001	2013-11-12	2604	Hy-Vee Wine and Spirits / Lem	1201 12TH AVE SW	LEMARS	51031	POINT (-96.18335000000002 4

Features:

We have 24 features in our dataset. The data types of these features are date, string, integer and float.



date	DATE	NULLABLE	Date of order
store_number	STRING	NULLABLE	Unique number assigned to the store who ordered the liquor.
store_name	STRING	NULLABLE	Name of store who ordered the liquor.
address	STRING	NULLABLE	Address of store who ordered the liquor.
city	STRING	NULLABLE	City where the store who ordered the liquor is located
zip_code	STRING	NULLABLE	Zip code where the store who ordered the liquor is located
store_location	STRING	NULLABLE	Location of store who ordered the liquor. The Address, City, State and Zip Code are geocoded to provide geographic coordinates. Accuracy of geocoding is dependent on how well the address is interpreted and the completeness of the reference data used.
county_number	STRING	NULLABLE	Iowa county number for the county where store who ordered the liquor is located
county	STRING	NULLABLE	County where the store who ordered the liquor is located
category	STRING	NULLABLE	Category code associated with the liquor ordered
category_name	STRING	NULLABLE	Category of the liquor ordered.
vendor_number	STRING	NULLABLE	The vendor number of the company for the brand of liquor ordered
vendor_name	STRING	NULLABLE	The vendor name of the company for the brand of liquor ordered
item_number	STRING	NULLABLE	Item number for the individual liquor product ordered.
item_description	STRING	NULLABLE	Description of the individual liquor product ordered.
pack	INTEGER	NULLABLE	The number of bottles in a case for the liquor ordered
bottle_volume_ml	INTEGER	NULLABLE	Volume of each liquor bottle ordered in milliliters.
state_bottle_cost	FLOAT	NULLABLE	The amount that Alcoholic Beverages Division paid for each bottle of liquor ordered
state_bottle_retail	FLOAT	NULLABLE	The amount the store paid for each bottle of liquor ordered
bottles_sold	INTEGER	NULLABLE	The number of bottles of liquor ordered by the store
sale_dollars	FLOAT	NULLABLE	Total cost of liquor order (number of bottles multiplied by the state bottle retail)
volume_sold_liters	FLOAT	NULLABLE	Total volume of liquor ordered in liters. (i.e. (Bottle Volume (ml) x Bottles Sold)/1,000)"
volume_sold_gallons	FLOAT	NULLABLE	Total volume of liquor ordered in gallons. (i.e. (Bottle Volume (ml) x Bottles Sold)/3785.411784)"

Date: date of the order (eg:'2020-08-19')

Stroe_name: name of the store (eg: Wilkie Liquors)

City: the city of the store (eg: Mount Vernon)

Category_name : the category of the liquor(eg: Neutral Grain Spirits)

Vendor_name : the name of the vendor (eg: LUXCO INC)

Item_description: description of the individual liquor (eg: Templeton Rye)

State_bottle_cost: the cost of each bottle of liquor

State_bottle_retail:the retail price of each bottle of liquor

Bottles_sold: the amount of liquor sold

Sale_dollars: the amount of money of liquor sales.

Project Design and Milestones:

Programming language:

We will use Python as the programming language, and JavaScript to create the UI dashboard.

Cloud platform:

Our main application platform will be Google Cloud, which will handle data computation and model deployment. We will use BigQuery to extract and analyze the data from the database.

Models:

We will use the time-series Model (ARIMA) in our project. In our daily life, Autoregressive Integrated Moving Average (ARIMA) models have many places to use. For example, if we need to forecast the stock of the product, we can use this model to forecast. In some factories, the stock of product is very important for them to plan the product. If we could use the ARIMA model to predict the stock in the future, it would be helpful for us.

ARIMA models are a general class of models used for forecasting time series data. ARIMA models are generally denoted as ARIMA (p,d,q) where p is the order of autoregressive model, d is the degree of differencing, and q is the order of moving-average model. ARIMA models use differencing to convert a non-stationary time series into a stationary one, and then predict future values from historical data.

Since the data has seasonal pattern, we use SARIMA (p,d,q)x(P,D,Q,s) model, which 'p' and seasonal 'P' are the order of autoregressive terms, 'd' and seasonal 'D' are the order of differencing, 'q' and seasonal 'Q' are the order of moving average terms and 's' indicates seasonal length in the series. The ACF (autocorrelation function) and PACF (partial autocorrelation function) plots are used to determine the SARIMA parameters.

We will also use Google's cloud services to create models for comparison. Google provides resources for a full training and deployment pipeline, with a python API that will allow for integration into the application. Google's Vertex AI offers a wide range of state-of-the-art models for various tasks. To predict tabular time-series data, Vertex's tabular regression and tabular forecasting models are ideal.

Deploy Model:

In our group, we use two ways to deploy the model. One way is a customized method, through importing ARIMA library to train the data and predict the result. Another uses the Vertix AI to train and predict the result.

Method 1: Customized method

Import the data:

The data are stored in bigquery-public-data.iowa_liquor_sales.sales.

```
[3]: import datetime

import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from google.cloud import bigquery
from google.cloud.bigquery import Client
from sklearn.metrics import (mean_absolute_error,
                             mean_absolute_percentage_error,
                             mean_squared_error)

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")
```

```
[4]: # Constants

# Project and dataset where the model will be saved
PROJECT = "csce5214-p1"
DATASET = "csce5214_iowa_sales"

# Source table for training data
SOURCE_TABLE = "`bigquery-public-data`.iowa_liquor_sales.sales"

#MODEL_NAME_PREFIX = "bqml_arimaplus_dw_"
#MODEL_TYPE = "arima_plus"

#HORIZON = 30
#GRANULARITY = "DAILY"
```

We write a query to extract the data (from 2017-01-01 to 2022-06-30),and transfer them to dataframe format.

```
[5]: client = Client(project=PROJECT)
```

```
[6]: query = """
SELECT *  FROM `bigquery-public-data.iowa_liquor_sales.sales`
where date between '2017-01-01' and '2022-06-30'
"""
query_job = client.query(query)
```

```
[7]: df = query_job.to_dataframe()
```

```
[9]: df.head(10)
```

[9]:

	invoice_and_item_number	date	store_number	store_name	address	city	zip_code	store_location	county_number	county	...	item_number	item_descri
0	INV-30344900004	2020-09-18	5864	Casey's General Store #2803 / Villisca	309 N U Ave	Villisca	50864	POINT (-94.985437 40.934673)	69	MONTGOMERY	...	36904	McCormick Vodk
1	INV-34229000003	2021-02-11	5982	Casey's General Store #6 / Altoona	407 8th St SW	Altoona	50009.0	POINT (-93.469065 41.64403500000001)	77	POLK	...	37996	Smirnoff
2	INV-26244900132	2020-04-01	2521	Hy-Vee Food and Drug / Grand / WDM	1990 Grand Avenue	West Des Moines	50265.0	POINT (-93.73162 41.571127)	77	POLK	...	37413	Popov
3	INV-37665500005	2021-06-21	4915	Casey's General Store #2429 / Bettendorf	3902, State St	Bettendorf	52722.0	POINT (-90.47859 41.53064)	82	SCOTT	...	36901	McCormick Vodk
4	INV-11514800015	2018-04-16	5100	Sam's Food	648 N Marquette St	Davenport	52802.0	POINT (-90.590879 41.526469)	82	SCOTT	...	36903	Mccc

Explore the data

```
[13]: df.describe()
```

[13]:

	pack	bottle_volume_ml	state_bottle_cost	state_bottle_retail	bottles_sold	sale_dollars	volume_sold_liters	volume_sold_gallons
count	1.352144e+07	1.352144e+07	1.352144e+07	1.352144e+07	1.352144e+07	1.352144e+07	1.352144e+07	1.352144e+07
mean	1.223921e+01	8.656266e+02	1.069740e+01	1.604802e+01	1.126422e+01	1.500622e+02	9.316561e+00	2.457186e+00
std	7.961545e+00	5.233113e+02	9.744447e+00	1.461633e+01	3.255452e+01	5.285318e+02	3.889444e+01	1.027497e+01
min	1.000000e+00	2.000000e+01	3.300000e-01	5.000000e-01	0.000000e+00	0.000000e+00	2.000000e-02	0.000000e+00
25%	6.000000e+00	7.500000e+02	5.600000e+00	8.400000e+00	3.000000e+00	3.600000e+01	1.500000e+00	4.000000e-01
50%	1.200000e+01	7.500000e+02	8.500000e+00	1.275000e+01	6.000000e+00	7.866000e+01	4.800000e+00	1.260000e+00
75%	1.200000e+01	1.000000e+03	1.300000e+01	1.950000e+01	1.200000e+01	1.530000e+02	1.050000e+01	2.770000e+00
max	1.200000e+02	3.150000e+04	2.198890e+03	3.298340e+03	1.320000e+04	2.795573e+05	1.320000e+04	3.487070e+03

```
[14]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13521437 entries, 0 to 13521436
Data columns (total 24 columns):
 #   Column                   Dtype
---  ------                   -----
 0   invoice_and_item_number  object
 1   date                     object
 2   store_number             object
 3   store_name               object
 4   address                  object
 5   city                     object
 6   zip_code                 object
 7   store_location           object
 8   county_number            object
 9   county                   object
 10  category                 object
 11  category_name            object
 12  vendor_number            object
 13  vendor_name              object
 14  item_number              object
 15  item_description         object
 16  pack                     int64
 17  bottle_volume_ml         int64
 18  state_bottle_cost        float64
```

Checking the missing value:

We find the date, sale_dollars ,bottles_sold don't have missing value.Address ,city,zip-code,store_location,country_number,county,category,category_name have missing value

```
[17]: # check the missing data
      df.isnull().sum()

[17]: invoice_and_item_number          0
      date                             0
      store_number                     0
      store_name                       0
      address                      50842
      city                         50841
      zip_code                     50841
      store_location             1379038
      county_number                50843
      county                       50841
      category                      8751
      category_name                 8751
      vendor_number                    7
      vendor_name                      7
      item_number                      0
      item_description                 0
      pack                             0
      bottle_volume_ml                 0
      state_bottle_cost                0
      state_bottle_retail              0
      bottles_sold                     0
      sale_dollars                     0
      volume_sold_liters               0
      volume_sold_gallons              0
      dtype: int64
```

Since the percentage of missing value is so small, we just drop the missing value.

```
[18]: df_null = pd.DataFrame(df.isnull().sum()).reset_index().rename(columns={0:'total_number'})
      df_null["percent"] = 100*df_null["total_number"] / int(len(df))
      df_null
```

[18]:

	index	total_number	percent
0	invoice_and_item_number	0	0.000000
1	date	0	0.000000
2	store_number	0	0.000000
3	store_name	0	0.000000
4	address	50842	0.376010
5	city	50841	0.376003
6	zip_code	50841	0.376003
7	store_location	1379038	10.198901
8	county_number	50843	0.376018
9	county	50841	0.376003
10	category	8751	0.064719
11	category_name	8751	0.064719
12	vendor_number	7	0.000052
13	vendor_name	7	0.000052
14	item_number	0	0.000000
15	item_description	0	0.000000
16	pack	0	0.000000
17	bottle_volume_ml	0	0.000000
18	state_bottle_cost	0	0.000000
19	state_bottle_retail	0	0.000000
20	bottles_sold	0	0.000000
21	sale_dollars	0	0.000000
22	volume_sold_liters	0	0.000000
23	volume_sold_gallons	0	0.000000

Data Visualization

In this figure, we could find the relationship between the category and sale_dollars. The sale of American Voldkas is the top 1.

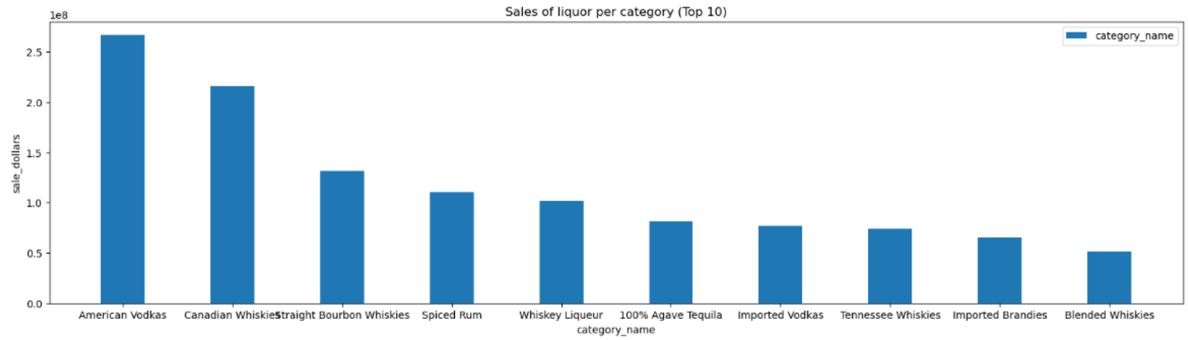
The relation between item category and sale_dollas,bottle_sold,profit

[21]:
```
# The relation between category and sale_dollars
df_category = df.groupby(["category_name"])["sale_dollars"].sum().reset_index().sort_values("sale_dollars", ascending=False)
```

[22]:
```
fig = plt.figure(figsize = (20, 5))

# creating the bar plot
plt.bar(df_category[:10]['category_name'], df_category[:10]['sale_dollars'],
        label ='category_name',width = 0.4)

plt.xlabel("category_name")
plt.ylabel("sale_dollars")
plt.title("Sales of liquor per category (Top 10)")
plt.legend(loc='best')
plt.show()
```



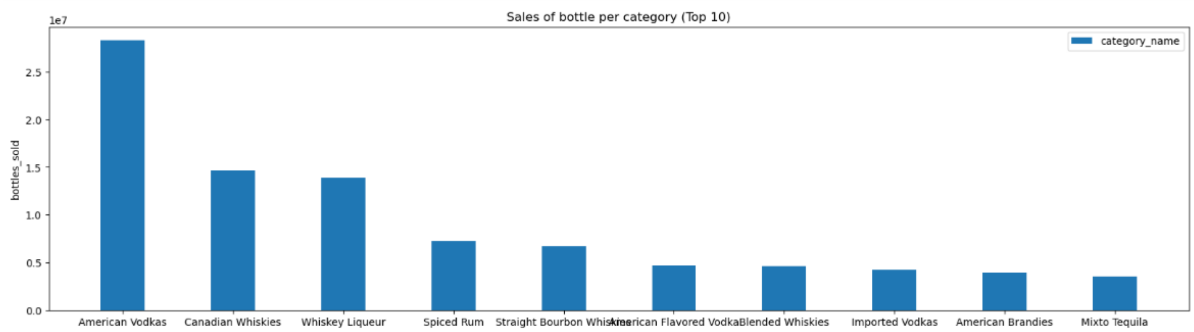
In this figure, we could find the relationship between the category and the sales of bottle. The sales of bottles of American Voldkas is the top 1.

[23]:
```
# The relation between category and bottles_sold
df_category = df.groupby(["category_name"])["bottles_sold"].sum().reset_index().sort_values("bottles_sold", ascending=False)
```

[24]:
```
fig = plt.figure(figsize = (20, 5))

# creating the bar plot
plt.bar(df_category[:10]['category_name'], df_category[:10]['bottles_sold'],
        label ='category_name',width = 0.4)

plt.xlabel("category_name")
plt.ylabel("bottles_sold")
plt.title("Sales of bottle per category (Top 10)")
plt.legend(loc='best')
plt.show()
```



In this figure, we could find the relationship between the profit and category. The profit of American Voldkas is the top 1.profit = state_bottle_retail – state_bottle_retail.

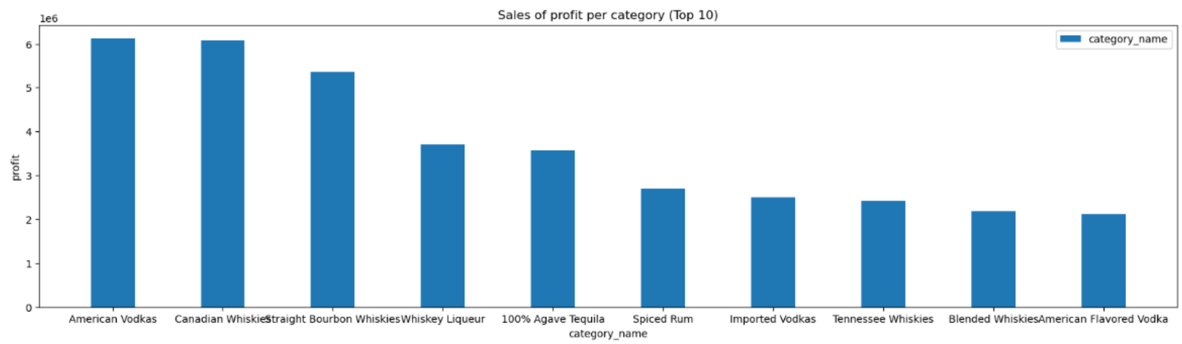
[25]:
```
# item profit
df['profit'] = df['state_bottle_retail'] - df['state_bottle_cost']
```

[26]:
```
# The relation between canegory and profit
df_category = df.groupby(["category_name"])["profit"].sum().reset_index().sort_values("profit", ascending=False)
```

[27]:
```
fig = plt.figure(figsize = (20, 5))

# creating the bar plot
plt.bar(df_category[:10]['category_name'], df_category[:10]['profit'],
        label ='category_name',width = 0.4)

plt.xlabel("category_name")
plt.ylabel("profit")
plt.title("Sales of profit per category (Top 10)")
plt.legend(loc='best')
plt.show()
```



We transfer the daily data to the month data.

[34]:
```
df_month = df[['date','sale_dollars']]
```

[35]:
```
df_month.head()
```

[35]:

	date	sale_dollars
0	2020-09-18	194.40
1	2021-02-11	445.68
2	2020-04-01	10.52
3	2021-06-21	42.30
4	2018-04-16	81.60

[36]:
```
df_month.set_index('date',inplace=True)
df_month.index = pd.to_datetime(df_month.index)
df_month_new = df_month.resample('MS').sum()
```

[37]:
```
df_month_new.head(10)
```

[37]:

date	sale_dollars
2017-01-01	17669818.37
2017-02-01	20460184.59
2017-03-01	22206185.90
2017-04-01	20522551.01
2017-05-01	25058219.74
2017-06-01	25540989.87
2017-07-01	22348944.10
2017-08-01	24659992.19
2017-09-01	21857938.77
2017-10-01	27192457.61

[38]:
```
df_month_new.shape
```

[38]: (66, 1)

ARIMA Model

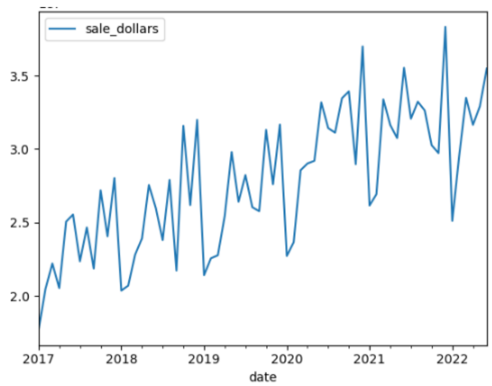
We need to check if the data are stationary. We will use the adfuller library to check.

We will use F-test to evaluate the result. We assume:

H0: the result is not stationary

H1: the result is stationary

From the figure and the P-value, we find the data is not stationary. We need use difference to modify.




```
[40]: # we need test the stationarity
# we will use adfuller from library
from statsmodels.tsa.stattools import adfuller
result_test_data = adfuller(df_month_new['sale_dollars'])
```

```
[41]: # we use F-test to evaulate the result
# We assume H0: the result is not stationary ,H1:the result is stationary
def adfuller_test(sale_dollars):
    result = adfuller(sale_dollars)
    label_list = ['F-test','P-value','Lags','Number of observations']
    for value,label in zip(result,label_list):
        print(label + ':' + str(value))
    if result[1] <=0.05:
        print('we reject the H0,we accept H1')
    else:
        print('we cannot reject H0, we need accept H0')
```

```
[42]: # The p-value is 0.85.it is greater then 0.05, So we can't reject the H0
adfuller_test(df_month_new['sale_dollars'])
```

```
F-test:-0.6856508057168805
P-value:0.8504313752373232
Lags:11
Number of observations:54
we cannot reject H0, we need accept H0
```

We differ 1 shift of the data, the p-value is 0.00105, it is stationary.

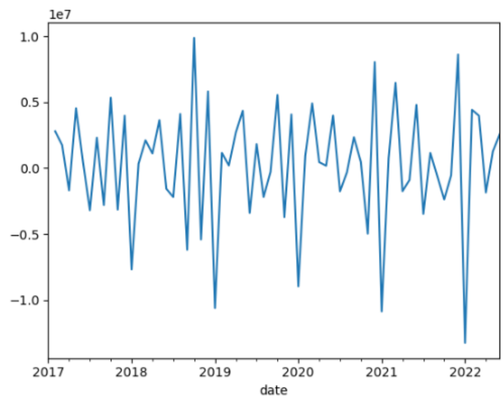
```
[43]: # Differecing the date: 1 shift
df_month_new['sale_dollars_first_differ'] = df_month_new['sale_dollars'] - df_month_new['sale_dollars'].shift(1)
```

```
[44]: #when we difference 1,the p-value is 0.001,it is less then 0.05, we reject the H0, the data is stationary
adfuller_test(df_month_new['sale_dollars_first_differ'].dropna())
```

```
F-test:-4.078479605628476
P-value:0.0010509666469033958
Lags:11
Number of observations:53
we reject the H0,we accept H1
```

```
[45]: df_month_new['sale_dollars_first_differ'].plot()
```

```
[45]: <AxesSubplot:xlabel='date'>
```

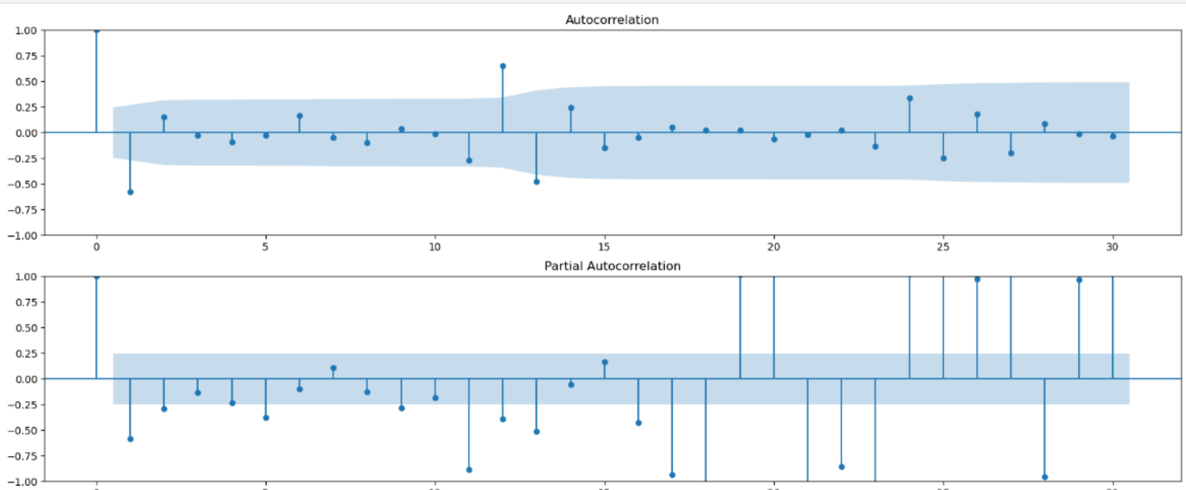


ACF and PACF

We can get the value of p,q from the figure of ACF and PACF.

```
[46]: # We will use ACF and PACF to find the parameter p and q
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

```
[47]: fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df_month_new['sale_dollars_first_differ'].iloc[2:],lags=30,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df_month_new['sale_dollars_first_differ'].iloc[2:],lags=30,ax=ax2)
```



In order to find the optimal parameter of p,q,d, we write a function to run ,and split the dataset 2/3 are training data,1/3 are test data.

```python
[48]: # grid search ARIMA parameters for time series
import warnings
from math import sqrt
from pandas import read_csv
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

# evaluate an ARIMA model for a given order (p,d,q)
def evaluate_arima_model(X, arima_order):
    # prepare training dataset
    train_size = int(len(X) * 0.66)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit()
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    # calculate out of sample error
    rmse = sqrt(mean_squared_error(test, predictions))
    return rmse

# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    rmse = evaluate_arima_model(dataset, order)
                    if rmse < best_score:
                        best_score, best_cfg = rmse, order
                    print('ARIMA%s RMSE=%.3f' % (order,rmse))
                except:
                    continue
    print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))
```

We set the parameters range. P [0-10],d [0,1,2],q [0,1,2]

At last, we got the optimal parameters ARIMA (8,0,2),we can use the parameters to predict the result.

```python
[49]: # load dataset
# evaluate parameters
p_values = range(0,11)
d_values = range(0, 3)
q_values = range(0, 3)
warnings.filterwarnings("ignore")
evaluate_models(df_month_new['sale_dollars_first_differ'].dropna().values, p_values, d_values, q_values)
```

```
ARIMA(0, 0, 0) RMSE=5146022.648
ARIMA(0, 0, 1) RMSE=3671270.194
ARIMA(0, 0, 2) RMSE=3659451.003
ARIMA(0, 1, 0) RMSE=8978220.407
ARIMA(0, 1, 1) RMSE=5176541.457
ARIMA(0, 1, 2) RMSE=3702955.601
ARIMA(0, 2, 0) RMSE=16337480.809
ARIMA(0, 2, 1) RMSE=9130181.071
ARIMA(0, 2, 2) RMSE=5590439.227
ARIMA(1, 0, 0) RMSE=4373290.444
ARIMA(1, 0, 1) RMSE=3656220.457
ARIMA(1, 0, 2) RMSE=3538583.397
ARIMA(1, 1, 0) RMSE=7094277.796
ARIMA(1, 1, 1) RMSE=4408471.820
ARIMA(1, 1, 2) RMSE=3947626.137
ARIMA(1, 2, 0) RMSE=12158076.503
ARIMA(1, 2, 1) RMSE=7185894.851
ARIMA(1, 2, 2) RMSE=4681533.749
ARIMA(2, 0, 0) RMSE=4059367.841
ARIMA(2, 0, 1) RMSE=3720439.947
```

```
ARIMA(5, 2, 2) RMSE=5128839.535
ARIMA(6, 0, 0) RMSE=4101966.707
ARIMA(6, 0, 1) RMSE=4161287.932
ARIMA(6, 0, 2) RMSE=3921218.769
ARIMA(6, 1, 0) RMSE=4547950.215
ARIMA(6, 1, 1) RMSE=4200423.498
ARIMA(6, 1, 2) RMSE=4614622.653
ARIMA(6, 2, 0) RMSE=5935015.430
ARIMA(6, 2, 1) RMSE=4643396.617
ARIMA(6, 2, 2) RMSE=4694767.680
ARIMA(7, 0, 0) RMSE=4085504.256
ARIMA(7, 0, 1) RMSE=4128485.468
ARIMA(7, 0, 2) RMSE=3978310.611
ARIMA(7, 1, 0) RMSE=4747002.113
ARIMA(7, 1, 1) RMSE=4170174.561
ARIMA(7, 1, 2) RMSE=4958421.849
ARIMA(7, 2, 0) RMSE=6032098.900
ARIMA(7, 2, 1) RMSE=4792428.972
ARIMA(7, 2, 2) RMSE=4413867.652
ARIMA(8, 0, 0) RMSE=4124967.050
ARIMA(8, 0, 1) RMSE=3871113.460
ARIMA(8, 0, 2) RMSE=3381725.233
ARIMA(8, 1, 0) RMSE=4971154.879
ARIMA(8, 1, 1) RMSE=4234445.835
ARIMA(8, 1, 2) RMSE=4105953.654
ARIMA(8, 2, 0) RMSE=6092643.802
ARIMA(8, 2, 1) RMSE=5066145.696
ARIMA(8, 2, 2) RMSE=4431649.413
ARIMA(9, 0, 0) RMSE=4357059.837
ARIMA(9, 0, 1) RMSE=4390185.742
ARIMA(9, 0, 2) RMSE=3841904.032
ARIMA(9, 1, 0) RMSE=4888247.011
ARIMA(9, 1, 1) RMSE=4374031.212
ARIMA(9, 1, 2) RMSE=4021612.318
ARIMA(9, 2, 0) RMSE=4859062.009
ARIMA(9, 2, 1) RMSE=4989026.507
ARIMA(9, 2, 2) RMSE=4139399.940
ARIMA(10, 0, 0) RMSE=4592644.036
ARIMA(10, 0, 1) RMSE=4405863.474
ARIMA(10, 0, 2) RMSE=3979092.860
ARIMA(10, 1, 0) RMSE=4570731.603
ARIMA(10, 1, 1) RMSE=4531926.149
ARIMA(10, 1, 2) RMSE=4004744.347
ARIMA(10, 2, 0) RMSE=4118816.370
ARIMA(10, 2, 1) RMSE=4621218.503
ARIMA(10, 2, 2) RMSE=4055025.170
Best ARIMA(8, 0, 2) RMSE=3381725.233
```

Train the model

[50]:
```
# For non-seasonal data
#p=8, d=0, q=2
from statsmodels.tsa.arima_model import ARIMA
```

[51]:
```
model=sm.tsa.arima.ARIMA(df_month_new['sale_dollars'],order=(8,0,2))
model_fit=model.fit()
```

[52]:
```
model_fit.summary()
```

[52]:

SARIMAX Results

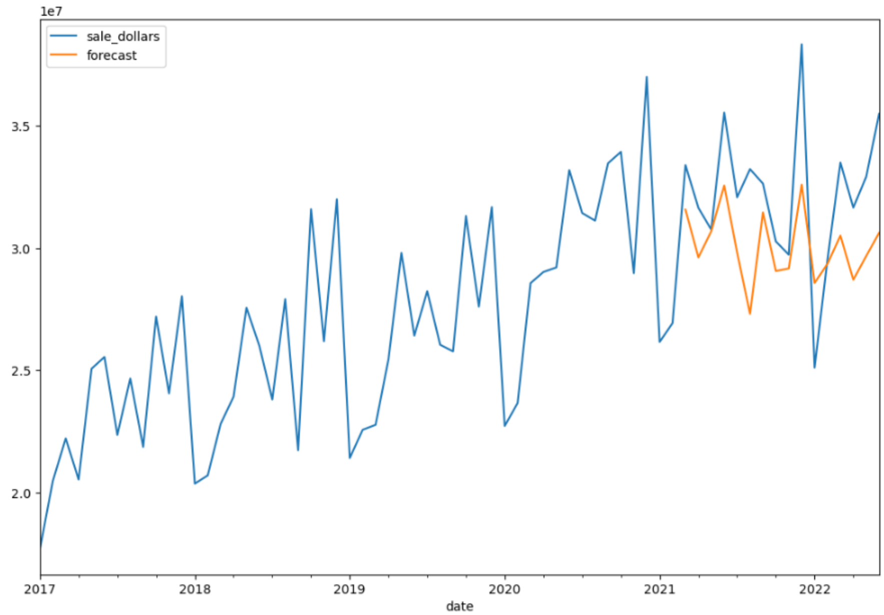
Dep. Variable:	sale_dollars	**No. Observations:**	66
Model:	ARIMA(8, 0, 2)	**Log Likelihood**	-1078.596
Date:	Wed, 05 Oct 2022	**AIC**	2181.191
Time:	19:47:53	**BIC**	2207.467
Sample:	01-01-2017	**HQIC**	2191.574
	- 06-01-2022		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.779e+07	5.12e-09	5.42e+15	0.000	2.78e+07	2.78e+07
ar.L1	-1.2520	0.227	-5.509	0.000	-1.698	-0.807
ar.L2	-0.3277	0.312	-1.049	0.294	-0.940	0.285
ar.L3	0.5690	0.274	2.078	0.038	0.032	1.106
ar.L4	0.2312	0.297	0.779	0.436	-0.350	0.813
ar.L5	-0.0088	0.334	-0.026	0.979	-0.663	0.645
ar.L6	0.4273	0.296	1.443	0.149	-0.153	1.008
ar.L7	0.7933	0.291	2.730	0.006	0.224	1.363
ar.L8	0.3515	0.244	1.443	0.149	-0.126	0.829
ma.L1	1.4886	0.300	4.965	0.000	0.901	2.076
ma.L2	0.9624	0.323	2.979	0.003	0.329	1.596
sigma2	1.073e+13	4.33e-14	2.48e+26	0.000	1.07e+13	1.07e+13

The blue line is the real sale_dollors, the orange line is the forecast value.

```
[55]: df_month_new['forecast']=model_fit.predict(start=50,end=70,dynamic=True)
df_month_new[['sale_dollars','forecast']].plot(figsize=(12,8))
```

```
[55]: <AxesSubplot:xlabel='date'>
```



Predict the data:

We use the next 24 month as the data to predict the sale.

```
[56]: from pandas.tseries.offsets import DateOffset
future_dates=[df_month_new.index[-1]+ DateOffset(months=x)for x in range(0,24)]
```

```
[57]: future_datest_df=pd.DataFrame(index=future_dates[1:],columns=df_month_new.columns)
```

```
[58]: future_datest_df.tail()
```

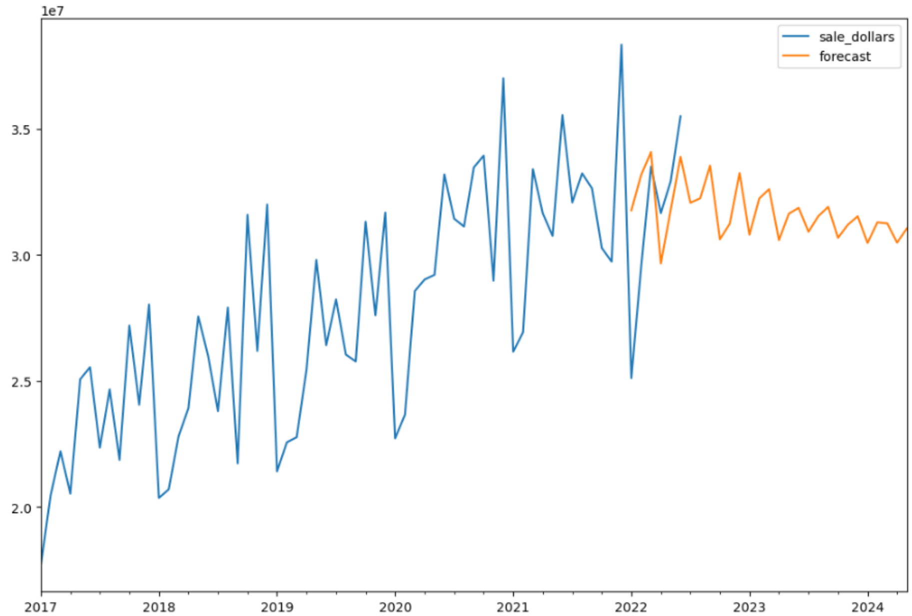
[58]:

	sale_dollars	sale_dollars_first_differ	forecast
2024-01-01	NaN	NaN	NaN
2024-02-01	NaN	NaN	NaN
2024-03-01	NaN	NaN	NaN
2024-04-01	NaN	NaN	NaN
2024-05-01	NaN	NaN	NaN

```
[59]: future_df=pd.concat([df_month_new,future_datest_df])
```

```
[65]: future_df['forecast'] = model_fit.predict(start = 60, end = 200, dynamic= True)
future_df[['sale_dollars','forecast']].plot(figsize=(12, 8))
```

[65]: <AxesSubplot:>



Vertex AI

For comparison, we are also developing models through Google's Vertex AI platform. Vertex AI allows for quick prototyping and deployment of standard models for a variety of input types. For this project, we are using Vertex's tabular regression and tabular forecasting models. The regression model uses standard regression techniques to predict a continuous value, in our case the value is sale numbers.

Much of the inner workings of the models are hidden, so special attention must be paid to data preparation. The following images show our data setup. On the left is the dataset for regression, the right is the data for forecasting. These are slices of the same original dataset, but the forecasting data covers a smaller time period. For both models, we have selected item number as the series identifier. This means that the model will view the dataset as many distinct time series, one for each product. For the forecasting model, we must also specify a timestamp for the series. In this case, we can use the "date" column.

Filter Enter property name or value

Column name	BigQuery type	BigQuery mode
address	STRING	NULLABLE
bottle_volume_ml	INTEGER	NULLABLE
bottles_sold	INTEGER	NULLABLE
category	STRING	NULLABLE
category_name	STRING	NULLABLE
city	STRING	NULLABLE
county	STRING	NULLABLE
county_number	STRING	NULLABLE
date	DATE	NULLABLE
invoice_and_item_number	STRING	NULLABLE
item_description	STRING	NULLABLE
item_number Series identifier	STRING	NULLABLE
pack	INTEGER	NULLABLE
sale_dollars	FLOAT	NULLABLE
state_bottle_cost	FLOAT	NULLABLE
state_bottle_retail	FLOAT	NULLABLE
store_location	STRING	NULLABLE
store_name	STRING	NULLABLE
store_number	STRING	NULLABLE
vendor_name	STRING	NULLABLE
vendor_number	STRING	NULLABLE
volume_sold_gallons	FLOAT	NULLABLE
volume_sold_liters	FLOAT	NULLABLE
zip_code	STRING	NULLABLE

Filter Enter property name or value

Column name	BigQuery type	BigQuery mode	Missing % (count)	Distinct valu
address	STRING	NULLABLE	0.59% (79792)	3470
bottle_volume_ml	INTEGER	NULLABLE	-	50
bottles_sold	INTEGER	NULLABLE	-	516
category	STRING	NULLABLE	0.12% (15710)	107
category_name	STRING	NULLABLE	0.17% (22726)	130
city	STRING	NULLABLE	0.59% (79791)	824
county	STRING	NULLABLE	1.11% (150335)	202
county_number	STRING	NULLABLE	1.11% (150337)	100
date Timestamp column	DATE	NULLABLE	-	1472
invoice_and_item_number	STRING	NULLABLE	-	13590124
item_description	STRING	NULLABLE	-	7217
item_number Series identifier	STRING	NULLABLE	-	7855
pack	INTEGER	NULLABLE	-	27
sale_dollars	FLOAT	NULLABLE	-	22045
state_bottle_cost	FLOAT	NULLABLE	-	2463
state_bottle_retail	FLOAT	NULLABLE	-	2721
store_location	STRING	NULLABLE	9.81% (1333559)	1787
store_name	STRING	NULLABLE	-	2351
store_number	STRING	NULLABLE	-	2177
vendor_name	STRING	NULLABLE	0%	442
vendor_number	STRING	NULLABLE	0%	546
volume_sold_gallons	FLOAT	NULLABLE	-	1393
volume_sold_liters	FLOAT	NULLABLE	-	1174
zip_code	STRING	NULLABLE	0.59% (79836)	943

The forecast model in particular has presented steep barriers. The increased complexity of the model means that only a small subset of the data can be processed at any time. For now we have aggregated the data into weekly measures and are operating on a greatly reduced dataset. Unfortunately, due to Google's attempts to obfuscate how their proprietary models function, models cannot be trained on additional data after they are created. In addition, as one of the newest additions to Google's AI platform, Vertex Forecasting is quite costly. This has limited our ability to experiment with potential solutions. More research will need to be done to see if these problems can be overcome.

2 regression models are trained and ready to be deployed, but have not been integrated into the project notebook yet.

BigQuery ML - ARIMA+

BigQuery ML ("BQML") is a SQL-based method native to Google BigQuery that enables data analysts to train models easily.

For this technology, we created a weekly-aggregated dataset for a 3-year period up to 2022 and used BQML forecasting to predict the first 3 months of 2022. Compared against the held out test set, BQML ARIMA+ achieved the following results after training for over 12 hours:

- RMSE: 36.37
- WAPE (Weighted Average Precision Error, commonly used in retail forecasting): 44%
- Bias (tendency to over- or under-predict): -0.08 (slightly under-predicting bottles sold)
- MAE: 5.94

These results are quite good (TODO: quantify this statement), but in future iterations of this project, we need to unify the calculation of these evaluation metrics across the other trialed approaches (ARIMA, SARIMA, Vertex Forecasting neural net) and add directionality measures in order to provide a direct comparison.

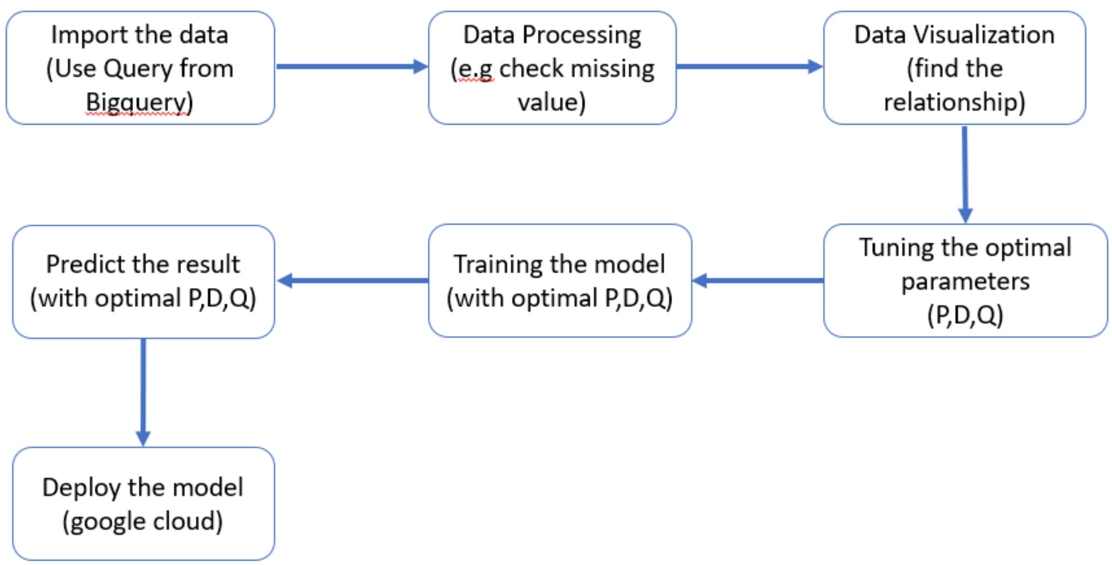
For BQML ARIMA+ and Vertex Forecasting, the training table was aggregated by week for the years 2019-2021 with held out test data from the first half of 2022. Further work will see more unification across the train/test split experiments we have conducted.

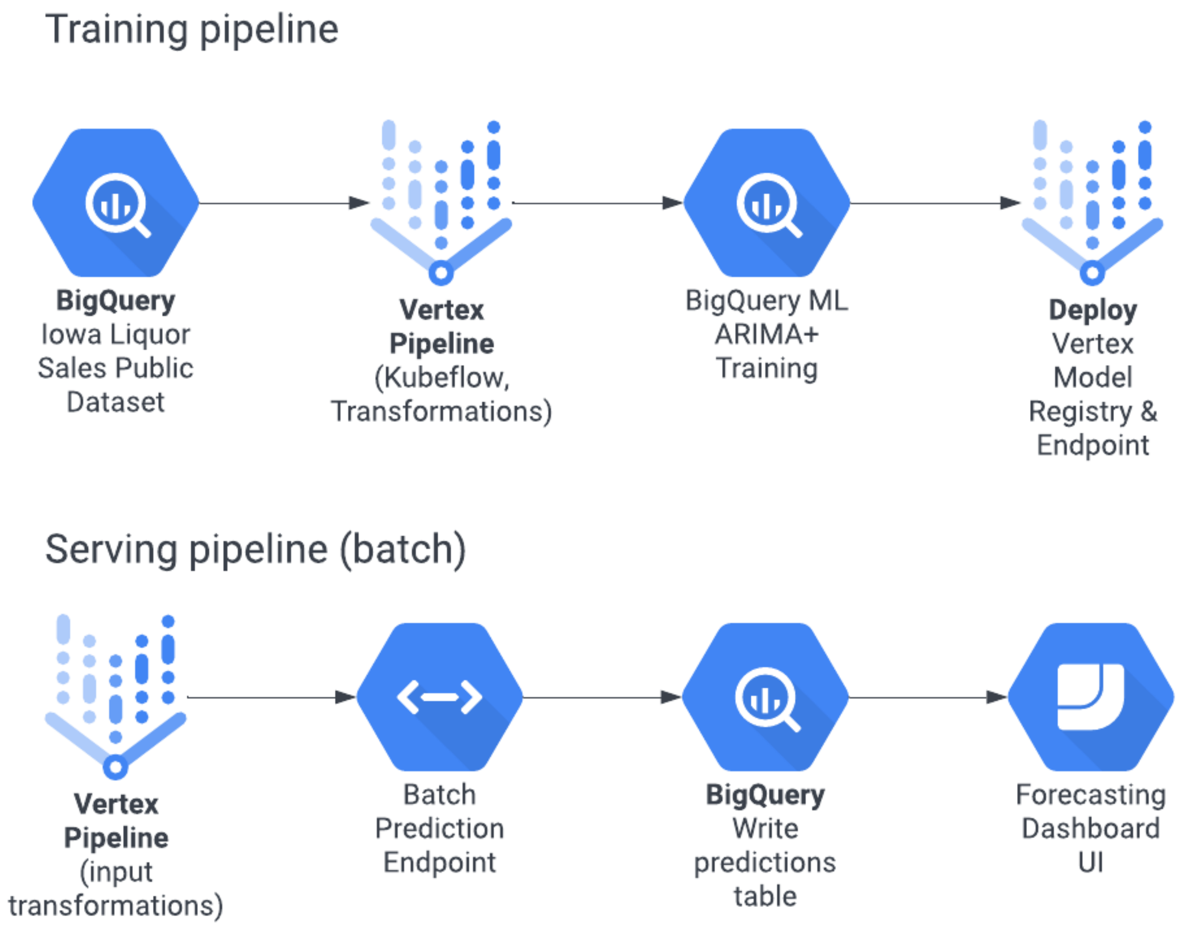
Remaining Milestones:

Week 6 Oct 8:

1. Create the pipeline
2. Testing the system
3. Complete the final report
4. Prepare the presentation

Architecture of the system/work flow





Resources and Related Projects:

[1] Iowa liquor sales data

https://data.iowa.gov/Sales-Distribution/Iowa-Liquor-Sales/m3tr-qhgy

[2] ARIMA

https://www.capitalone.com/tech/machine-learning/understanding-arima-models/

[3] BigQuery ML on Google Cloud

https://cloud.google.com/blog/topics/developers-practitioners/how-build-demand-forecasting-models-bigquery-ml

[4] How to Convert a Time Series to a Supervised Learning Problem in Python

https://machinelearningmastery.com/grid-search-arima-hyperparameters-with-python/