



Università Politecnica delle Marche

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Corso di “Data Science” tenuto dal Prof. Domenico Ursino

Analisi del dataset Iowa Liquor Sales attraverso il linguaggio di programmazione Python



Realizzato da:

Occhionero Giorgia
Sciarretta Luigi
Sebastianelli Alessandro

Indice

1.	Introduzione	4
1.1	Introduzione a Python	4
1.2	Introduzione a Jupyter.....	4
1.3	Introduzione a Pandas	5
2.	Dataset.....	6
2.1	Dataset Iowa Liquor Sales.....	6
2.1.1	ETL	7
2.2.	Dataset Popolazione.....	8
3.	Analisi descrittive.....	11
3.1	Prezzo medio per bottiglia.....	12
3.2	Bottiglie vendute	13
3.3	Profitto medio per bottiglia.....	14
3.4	Classifica venditori.....	15
3.5	Classifica tipologia di alcolici	17
3.6	Classifica tipologia di alcolici	19
3.7	Classifica categoria di alcolici	20
3.8	Iistogramma delle vendite.....	22
3.9	Iistogramma delle bottiglie vendute	23
4.	Analisi descrittive per l'anno 2019	24
4.1	Prezzo medio per bottiglia.....	26
4.2	Bottiglie vendute	27
4.3	Profitti per negozio	28
4.4	Correlazione tra la popolazione ed i profitti annuali.....	29
4.5	Bottiglie vendute per ordine	31
5.1	Clustering bottiglie vendute e prezzo al dettaglio.....	36
5.2	Clustering bottiglie vendute e volume per bottiglia.....	39
5.2.1	K-means “Black Velvet”	41
6.	Influenza dei parametri di posizione sulle vendite.....	42
6.1	Creazione dataframe “sales_by_zip”.....	43
6.2	Creazione dataframe “volume_by_zip”	44
6.3	Creazione dataframe “bottleprofit_per_zip”	44
6.4	Creazione dataframe “stores_by_zip”	45
6.5	Creazione dataframe “bottles_by_zip”	45
6.5	Fusione dei dataframe e creazione del dataframe “zip_frame”	46
6.5.1	Merge dataframe “stores_by_zip” e “sales_by_zip”	46

6.5.2 Merge dataframe “zip_frame” e “volume_by_zip”	47
6.5.3 Merge dataframe “zip_frame” e “bottleprofit_per_zip”	47
6.5.4 Merge dataframe “zip_frame” e “bottles_by_zip”	48
6.5.5 Visualizzazione dataframe finale	48
6.6 Individuazione parametri e creazione dei modelli	49
6.6.1 Creazione del modello	51
6.6.2 Regressione Lineare Multipla	52
6.6.3 Validazione del modello - Holdout Validation	54
6.6.4 K- Cross Fold Validation	56
6.6.5 Metodo Regressione Ridge	58
6.6.6 Lasso Regression Model	60
6.6.7 Conclusioni	62
7. Modello di regressione per predizione Contea	65
7.1 Model testing e model evaluation	73
7.1.1 Hold-out Validation	73
7.1.2 K-Cross Fold Validation	74
7.2 Metodo di regolarizzazione	75
7.2.1 Lasso Regression	75
7.2.2 Analisi contea di Dallas	77
8. Analisi temporali	85
8.1 Data Preprocessing	85
8.2 Modello serie temporale ARIMA	86
8.3 Trend Extraction	87
8.4 Modellazione ARIMA	89
8.4.1 Selezione dei parametri per il modello di serie storica ARIMA	89
8.4.2 Adattamento di un modello di serie storica ARIMA	90
8.5 Convalida delle previsioni	92
8.6 Previsioni	93
8.7 Comparazione dei marchi	94
8.8 Previsioni tramite “Prophet”	96
8.9 Comparazione delle tendenze	98
8.10 Mercato americano dei liquori	101
9. Conclusioni	102
10. Riferimenti	103

1. Introduzione

1.1 Introduzione a Python

Python è un linguaggio di programmazione open source interpretato che è orientato agli oggetti. Esso viene usato maggiormente per:

- data analysis: quindi usato per tutta la parte di ETL, per costruire i modelli di machine learning, valutare i modelli, ecc...
- sviluppo web: Python ha dei framework come Django e Flask che conferiscono la possibilità di creare dei siti web oppure dashboard che posso essere usate per mostrare i risultati alla community;
- machine learning e scripting.

Per sviluppare in Python esistono delle alternative, è possibile scrivere codice da shell o da un IDE.

- Shell: dal prompt è possibile digitare "Python" e appare una shell.
- PyCharm: ha un IDE completo.
- Jupyter Notebook: il notebook, in pratica, è un ambiente dove si ha la possibilità di scrivere del codice, mandarlo in esecuzione, ottenere un risultato e continuare a costruire questi pezzi di codice, volta per volta. Quindi propone un ambiente molto dinamico.

Riguardo invece l'ambiente che circonda Python, è interessante menzionare "PIP". Sostanzialmente, PIP è un sistema per tenere aggiornati i pacchetti e per poter gestire i pacchetti di Python. Questo perché, una volta installato Python, si hanno a disposizione solamente le librerie di base. Se invece si hanno bisogno di librerie più avanzate, come ad esempio Scikit-learn o Pandas, verrà utilizzato l'uso di PIP. Esso, infatti, dà la possibilità di gestire i pacchetti sia a livello di installazione (utilizzando il comando "pip install <nome del package>") e automaticamente si collegherà a determinate sorgenti, scaricherà il pacchetto e lo installerà. Per rimuovere il pacchetto, invece, si utilizza il comando "pip uninstall <nome del package>".

Tutte queste operazioni, ovviamente, vengono fatte da terminale.

Molto spesso, nei progetti di data science, le librerie sono tante: ad esempio può servire una libreria per fare visualizzazione, per creare il modello, per fare ETL, ecc... PIP dà anche la possibilità di scrivere su un file di testo il nome della libreria e, una volta che si ha questo file di testo, attraverso un "pip install -r <nomefile>.txt" lui prende quel file di testo ed andrà ad installare tutte le librerie che sono state scritte al suo intero.

Le librerie più importanti da installare sono ad esempio NetworkX, Matplotlib e Numpy.

1.2 Introduzione a Jupyter

È un'applicazione client-server dove si ha un kernel che lavora in background e che riceve le istruzioni in Python da eseguire dal programmatore. Infatti, in questo notebook, viene scritto il programma in Python, viene inviato al kernel, ed esso eseguirà le istruzioni e poi invierà l'output, quindi viene creato un rapporto di client-server fra il programmatore e questo kernel in Python che esegue le nostre istruzioni.

Il vantaggio è che si ha un sistema abbastanza dinamico perché si possono creare delle variabili che rimangono in memoria e si possono costruire dei prototipi molto velocemente. Questo perché serve solo scrivere il codice, lanciare quella particolare cella e si ha subito il risultato che rimarrà in memoria, quindi non servirà rilanciarlo una seconda volta.

Quindi, una volta installato, si digita nella shell il comando "jupyter notebook" e si apre una scheda del browser dove si ha la possibilità di scrivere i comandi in Python.

1.3 Introduzione a Pandas

Pandas è una libreria di Python per la manipolazione di dati in formato sequenziale o tabellare che dà la possibilità di gestire i dati in maniera facile, leggere dati da file o anche da altre sorgenti. È possibile, inoltre, organizzare i dati andando a sostituire i valori nulli e conferisce quindi la possibilità di effettuare una serie di operazioni ETL, fondamentali nella Data Science, in maniera semplice. Con Pandas si può andare a leggere da file e quest'ultimo viene organizzato in righe e colonne. Pandas può essere utilizzato per diversi tipi di dati in cui le colonne possono contenere dati eterogenei tra di loro. In conclusione, permette il caricamento e salvataggio di formati standard per dati tabellari, quali CSV, TSV, file Excel e formati per database e rende semplici le esecuzioni di operazioni di indicizzazione e aggregazione dei dati, le operazioni numeriche e statistiche, nonché fornisce una semplice visualizzazione dei risultati delle operazioni.

2. Dataset

2.1 Dataset Iowa Liquor Sales

Il dataset preso in esame contiene le informazioni sull'acquisto di alcolici dei commercianti di liquori che possiedono la licenza di classe "E" dell'Iowa per prodotto e data di acquisto dal 1° gennaio 2012 al 2019. La licenza per alcolici di classe E, per drogherie, negozi di liquori, minimarket, ecc., permette agli esercizi commerciali di vendere alcolici per il consumo fuori dai locali in contenitori originali non aperti. I dati sono relativi ad un insieme di osservazioni basate sulle transazioni, in cui ogni osservazione è unica per l'alcol venduto e per la quantità a cui è stato venduto. In ogni osservazione sono incluse informazioni aggiuntive sull'ubicazione, sui venditori e sulle vendite.

Il dataset è stato scaricato dal portale dati dello Stato dell'Iowa ed è disponibile al seguente link: <https://data.iowa.gov/Sales-Distribution/Iowa-Liquor-Sales/m3tr-qhgy>

	Invoice/Item Number	Date	Store Number	Store Name	Address	City	Zip Code	Store Location	County Number	County	...	Item Number	Item Description	Pack	Bottle Volume (ml)	State Bottle Cost	State Bottle Retail	Bottles Sold	Sale (\$Dollars)	Volume Sold (Liters)	Volume Sold (Gallons)
0	INV-0002010009	08/29/2016	4256	Fareway Stores #912 / Sioux Center	115 1st Ave NW	Sioux Center	51250	POINT (-96.176959 43.078003)	84.0	SIOUX	...	69636	Mcgillicuddy's Cherry Schnapps	12	750	8.66	12.99	6	12.99	4.50	1.18
1	INV-0006700040	08/31/2016	4588	Sam's Mini Mart / Sioux City	923 W 7TH ST	Sioux City	51103	POINT (-96.419074 42.503495)	97.0	WOODBURY	...	43024	Admiral Nelson Spiced Rum	24	375	2.74	4.11	1	98.64	0.37	0.09
2	INV-0004000015	08/30/2016	4176	Todds On The Go	235 Edgewood Rd NE	Cedar Rapids	52405	Nan	57.0	LINN	...	11776	Black Velvet	12	750	5.23	7.85	1	94.20	0.75	0.19
3	INV-0003060017	08/30/2016	2605	Hy-Vee Drugstore #5 / Cedar Rapids	2001 Blairs Ferry Road NE	Cedar Rapids	52402	POINT (-91.668909 42.034799)	57.0	LINN	...	5061	Glenlivet 15 Year French Oak	6	750	32.48	48.72	3	48.72	2.25	0.59
4	S3368270006	08/01/2016	4967	Jeffs Market / Blue Grass	102, W Mayne St	Blue Grass	52726	POINT (-90.766126 41.509119)	82.0	Scott	...	10549	Black Velvet Toasted Caramel Mini	10	600	7.12	10.68	1	10.68	0.60	0.16

Figura 2.1: Dataset Iowa Liquor Sales

Il dataset è composto dai seguenti campi:

- **Item Number** - identificativo univoco relativo al codice sku del prodotto;
- **Date** - data transazione;
- **Store Name** - nome del negozio;
- **Store Number** - identificativo univoco per negozio di liquori;
- **Address** - indirizzo del negozio;
- **City** - città in cui si trova il negozio;
- **Zip Code** - codice postale in cui si trova il negozio;
- **Store Location** - coordinate del negozio;
- **County Number** - numero della contea IOWA in cui si trova il negozio;
- **County** - nome della contea IOWA in cui si trova il negozio;
- **Category** - categoria del tipo di liquore;
- **Category Name** - nome per il tipo di liquore;
- **Vendor Number** - identificativo univoco per il fornitore;
- **Item Description** - descrizione del prodotto;
- **Pack** – numero di confezioni per transazione;
- **Bottle Volume (ml)** - dimensione in ml della bottiglia;

- **State Bottle Cost** - costo dal fornitore allo stato;
- **State Bottle Retail** - prezzo per bottiglia;
- **Bottles Sold** - quantità di bottiglie vendute;
- **Sale (Dollars)** - vendite totali per transazione;
- **Volume Sold (Liters)** - volume venduto per transazione in litri;
- **Volume Sold (Gallons)** - volume venduto per transazione in galloni;

2.1.1 ETL

Il dataset scaricato dal sito è grezzo e ha richiesto alcune operazioni di ETL, le principali operazioni effettuate sono le seguenti:

- eliminazione valori nulli e duplicati: verifichiamo la presenza dei suddetti e procediamo alla loro eliminazione;

	<code>df.isnull().sum()</code>	<code>df.dropna(inplace=True)</code> <code>df.drop_duplicates(inplace = True)</code> <code>df.isnull().sum()</code>
Invoice/Item Number	0	0
Date	0	0
Store Number	0	0
Store Name	0	0
Address	79927	0
City	79926	0
Zip Code	79971	0
Store Location	1927332	0
County Number	156731	0
County	156729	0
Category	16974	0
Category Name	25040	0
Vendor Number	5	0
Vendor Name	3	0
Item Number	0	0
Item Description	0	0
Pack	0	0
Bottle Volume (ml)	0	0
State Bottle Cost	10	0
State Bottle Retail	10	0
Bottles Sold	0	0
Sale (Dollars)	10	0
Volume Sold (Liters)	0	0
Volume Sold (Gallons)	0	0
<code>dtype: int64</code>		<code>dtype: int64</code>

Figura 2.2: Verificare presenza di duplicati e valori nulli con relativa eliminazione

- conversione dell'attributo “Date” nel formato desiderato e creazione di due colonne utili per ulteriori analisi: mese ed anno;

```
df.Date = pd.to_datetime(df["Date"], format = "%m/%d/%Y")
df['month'] = df.Date.dt.month
df['year'] = df.Date.dt.year
```

Figura 2.3: Modifica formato campo “Date”

- risoluzione della presenza di duplicati causa formato maiuscolo e minuscolo per uno stesso campo:

```
df["Category Name"] = df["Category Name"].str.upper()
df['Vendor Name'] = df['Vendor Name'].str.upper()
```

Figura 2.4: Trasformazione in formato maiuscolo dei campi

Altre operazioni di ETL sono presenti nelle sezioni successive fatte in fase di preprocessing durante la stesura del codice per le particolari analisi.

2.2. Dataset Popolazione

Per effettuare analisi più dettagliate è stato utilizzato anche il dataset relativo alla popolazione degli USA. Il dataset è stato scaricato ed è disponibile al seguente link:

<https://www.ers.usda.gov/data-products/county-level-data-sets/download-data/>

	State	Area_Name	year 2010	year 2011	year 2012	year 2013	year 2014	year 2015	year 2016	year 2017	year 2018	year 2019
0	US	United States	309321666	311556874	313830990	315993715	318301008	320635163	322941311	324985539	326687501	328239523
1	AL	Alabama	4785437	4799069	4815588	4830081	4841799	4852347	4863525	4874486	4887681	4903185
2	AL	Autauga County	54773	55227	54954	54727	54893	54864	55243	55390	55533	55869
3	AL	Baldwin County	183112	186558	190145	194885	199183	202939	207601	212521	217855	223234
4	AL	Barbour County	27327	27341	27169	26937	26755	26283	25806	25157	24872	24686

Figura 2.5: Dataset popolazione degli USA

Il dataset è composto dai seguenti campi:

- State: sigla identificativa dello stato;
- Area_Name: nome della contea;
- Year: anno a cui si riferiscono i dati;

Il dataset scaricato dal sito è grezzo e ha richiesto alcune operazioni di ETL. Le operazioni effettuate sono le seguenti:

- filtraggio dei dati relativi al solo stato dell'Iowa:

```
popolazione_IA=popolazione[popolazione['State']=='IA']
```

	State	Area_Name	year 2010	year 2011	year 2012	year 2013	year 2014	year 2015	year 2016	year 2017	year 2018	year 2019	FIPStxt
806	IA	Adair County	7679	7546	7468	7387	7368	7145	7005	7051	7074	7152	19001
807	IA	Adams County	4023	3994	3910	3891	3877	3754	3692	3657	3644	3602	19003
808	IA	Allamakee County	14378	14222	14149	14071	14062	13874	13851	13803	13852	13687	19005
809	IA	Appanoose County	12856	12848	12707	12654	12671	12577	12505	12353	12401	12426	19007
810	IA	Audubon County	6098	6004	5865	5863	5771	5711	5626	5550	5471	5496	19009

Figura 2.6: Codice e dataframe ottenuto

- eliminazione delle colonne non interessanti al fine delle analisi mantenendo i campi “State”, “Area_Name” e “year 2019”;

```
Population_IA=popolazione_IA[['State', 'Area_Name', 'year 2019']]
```

	State	Area_Name	year 2019
806	IA	Adair County	7152
807	IA	Adams County	3602
808	IA	Allamakee County	13687
809	IA	Appanoose County	12426
810	IA	Audubon County	5496

Figura 2.7: Codice e dataframe ottenuto

- trasformazione dei valori del campo Area_Name nella corretta rappresentazione, eliminando la parola County:

```
Population_IA['Area_Name'] = Population_IA['Area_Name'].map(lambda x: x.rstrip(' County'))
```

	State	Area_Name	year 2019
806	IA	Adair	7152
807	IA	Adams	3602
808	IA	Allamakee	13687
809	IA	Appanoose	12426
810	IA	Audub	5496

Figura 2.8: Codice e dataframe ottenuto

- trasformazione del contenuto di Area_Name in maiuscolo, per evitare il problema dei duplicati:

```
Population_IA['Area_Name']=Population_IA['Area_Name'].str.upper()
```

	State	Area_Name	year 2019
806	IA	ADAIR	7152
807	IA	ADAMS	3602
808	IA	ALLAMAKEE	13687
809	IA	APPANOOSE	12426
810	IA	AUDUB	5496

Figura 2.9: Codice e dataframe ottenuto

- verifica della forma del dataframe Population_IA:

Population_IA.shape
(99, 3)

Figura 2.10: Output ottenuto

- rinomina del campo Area_Name in County, year 2019 in Population:

```
Population_IA.rename(columns={'Area_Name': 'County'}, inplace=True)
```

```
Population_IA.rename(columns={'year 2019': 'Population'}, inplace=True)
```

	State	County	Population
806	IA	ADAIR	7152
807	IA	ADAMS	3602
808	IA	ALLAMAKEE	13687
809	IA	APPANOOSE	12426
810	IA	AUDUB	5496

Figura 2.11: Codici rinomina colonne e dataframe ottenuto

Per alcune delle analisi effettuate di seguito verranno utilizzati i dati riferiti al solo anno 2019. Di seguito è riportato il filtraggio a tale anno:

```
df_2019=df[df['year']==2019]
```

Figura 2.12: Filtraggio anno 2019

3. Analisi descrittive

L'obiettivo di questa fase è di analizzare in ottica descrittiva il dataset “Iowa Liquor Sales”, il quale contiene informazioni riguardanti l'attività di vendita di alcolici negli Stati Uniti, in particolare nello stato dell'Iowa, e quindi nello specifico analizzare eventi passati per trarre informazioni utili per la descrizione di certi fenomeni o certi comportamenti.

Per effettuare le analisi calcoliamo la colonna denominata “State profit per bottle”, ottenuta dalla differenza tra il costo della bottiglia al dettaglio e il costo della bottiglia dal fornitore allo stato:

```
df['State profit per bottle'] = df['State Bottle Retail'] - df['State Bottle Cost']
```

Figura 3.1: Creazione del nuovo campo “State profit per bottle”

Inoltre, si raggruppa per numero di negozio in modo da avere un dataframe che mostra dei campi in base al valore raggruppato. Si ottiene così il dataset che utilizzeremo per alcune delle diverse analisi descrittive, denominato “stores”:

```
stores= df.groupby('Store Number',as_index=False)[['State Bottle Retail','Bottles Sold', 'State profit per bottle']].mean()
print(stores)
```

	Store Number	State Bottle Retail	Bottles Sold	State profit per bottle
0	2106	16.007238	19.165077	5.342139
1	2113	15.665237	4.403961	5.230438
2	2130	15.884311	19.273862	5.299980
3	2132	15.070841	7.721495	5.048897
4	2152	12.689808	4.712957	4.252735
...
2342	9931	21.000000	2.200000	7.000000
2343	9934	28.880000	1.000000	9.630000
2344	9937	26.250000	2.000000	8.750000
2345	9938	41.120000	49.000000	13.710000
2346	9946	22.500000	96.000000	7.500000

Figura 3.2: Codice e dataset “stores” ottenuto

3.1 Prezzo medio per bottiglia

In questa analisi si vuole determinare e mostrare il prezzo medio della vendita al dettaglio delle bottiglie. Come prima analisi descrittiva viene mostrata una panoramica del prezzo medio per bottiglia (“State Bottle Retail”), e si andranno a considerare solo le bottiglie che hanno un costo unitario inferiore a 30 dollari, in quanto le bottiglie con un prezzo maggiore di 30 dollari sono piuttosto rare.

Attraverso il seguente script si realizza ciò:

```
print(stores[stores['State Bottle Retail']<30]['State Bottle Retail'].mean())
sns.distplot(stores[stores['State Bottle Retail']<30]['State Bottle Retail']);
```

Figura 3.3: Codice per valutazione del prezzo medio

L’output relativo è il grafico che ci permette di valutare i valori di questa analisi ed è mostrato nella figura seguente:

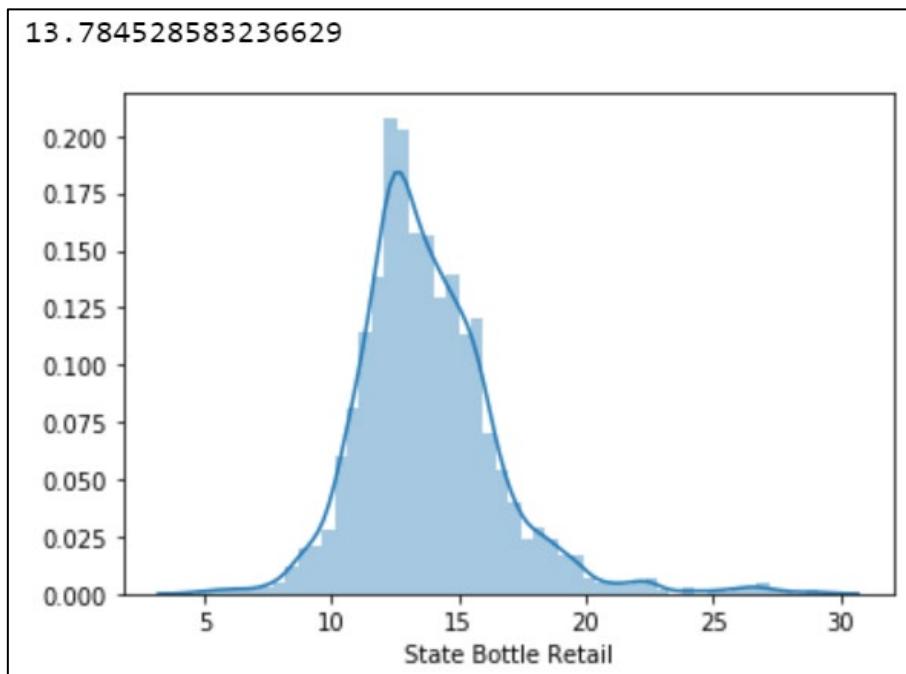


Figura 3.4: Istogramma prezzo medio per bottiglia

Possiamo dedurre da questa panoramica che ci sono negozi che vendono bottiglie sia “costose” che “economiche” ma in media possiamo vedere che le bottiglie vendute hanno un prezzo pari a 13,78 dollari.

```
print(len(stores))
print(len(stores[stores['State Bottle Retail']>=30]))
```

2347
12

Figura 3.5: Individuazione degli outlier

Andiamo a vedere quanti outlier ci sono, quindi quante bottiglie hanno un prezzo al dettaglio maggiore di 30 dollari, e possiamo vedere che ce ne sono solo 12.

3.2 Bottiglie vendute

In questa analisi si vuole determinare e mostrare il numero medio di bottiglie al dettaglio vendute. Dunque, viene mostrata una panoramica delle bottiglie vendute (“Bottle Sold”) escludendo nell’analisi le vendite con una quantità di bottiglie maggiore di 50 unità, in quanto corrispondono a vendite all’ingrosso. Attraverso il seguente script si realizza ciò:

```
print(stores[stores['Bottles Sold']<50]['Bottles Sold'].mean())
sns.distplot(stores[stores['Bottles Sold']<50]['Bottles Sold']);
```

Figura 3.6: Codice per valutazione bottiglie vendute

L’output relativo è il grafico che ci permette di valutare i valori di questa analisi ed è mostrato nella figura seguente:

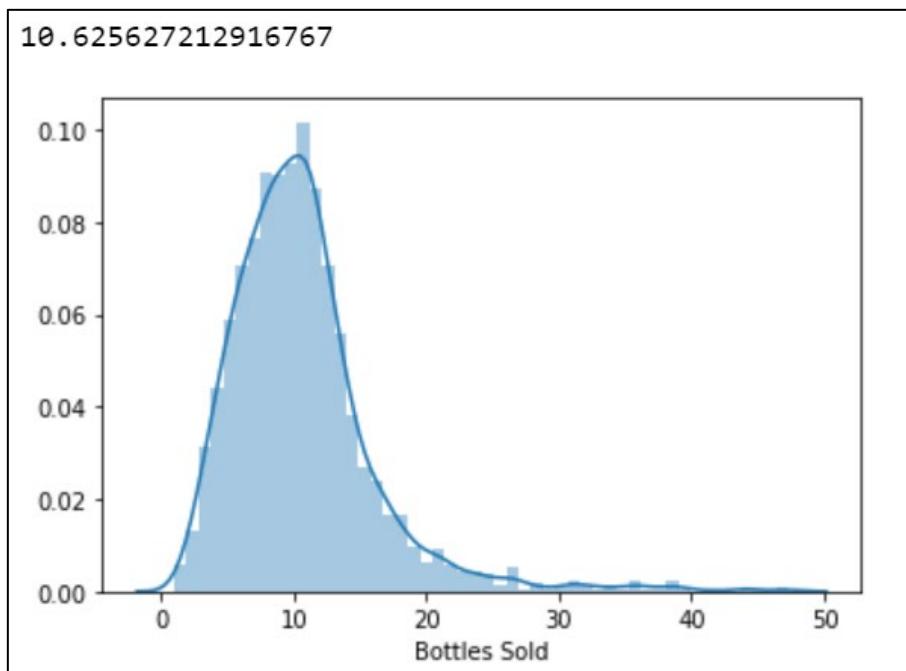


Figura 3.7: Istogramma bottiglie vendute

Considerando la vendita al dettaglio di alcolici possiamo vedere che in media vengono vendute 10,62 bottiglie per transazione di vendita.

```
# Quanti outlier abbiamo? appena 10.
print(len(stores))
print(len(stores[stores['Bottles Sold']>=50]))
```

2335
10

Figura 3.8: Individuazione degli outlier

Si possono vedere quanti outlier ci sono, quindi quante transazioni contengono più di 50 bottiglie, possiamo vedere che ce ne sono solo 10.

3.3 Profitto medio per bottiglia

In questa analisi si vuole determinare e mostrare il profitto medio per bottiglia da parte dello stato. Dunque, viene mostrata una panoramica dei profitti medi da parte dello stato (“State profit per bottle”) e nell’analisi vengono esclusi profitti maggiori di 12 dollari ipotizzando che difficilmente i profitti possano essere vicini al prezzo medio delle bottiglie.

```
print(stores[stores['State profit per bottle']<12]['State profit per bottle'].mean())
sns.distplot(stores[stores['State profit per bottle']<12]['State profit per bottle']);
```

Figura 3.9: Codice per valutazione profitto medio per bottiglia

L’output relativo è il grafico che ci permette di valutare i valori di questa analisi ed è mostrato nella figura seguente:

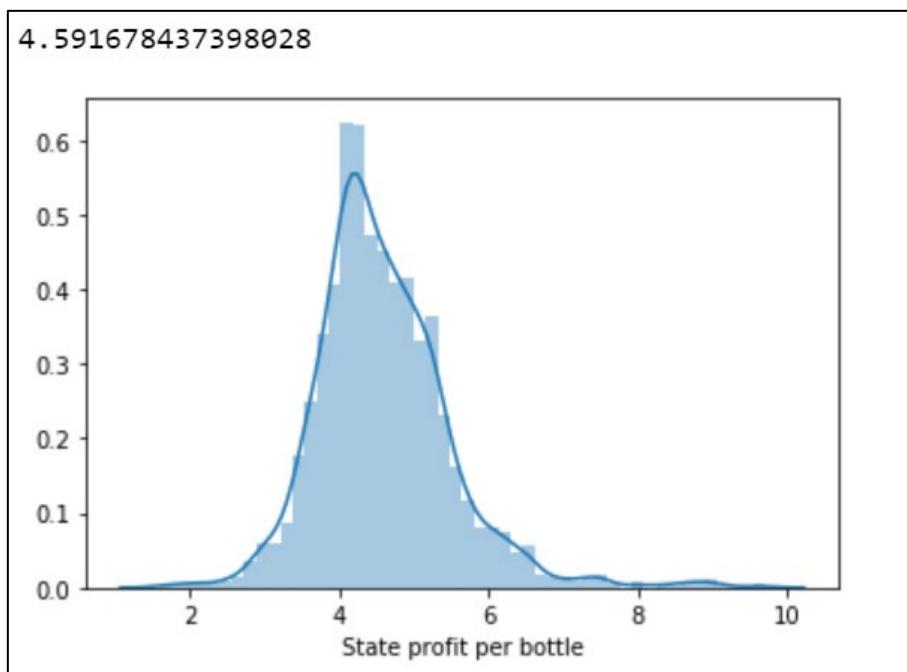


Figura 3.10: Istogramma profitto medio per bottiglia

Possiamo dedurre che il profitto medio dello stato per bottiglia è pari a 4,6 dollari.

```
#quanti outlier abbiamo? appena 9
print(len(stores))
print(len(stores[stores['State profit per bottle']>=12]))
```

2325
0

Figura 3.11: Individuazione degli outlier

Valutando la presenza di outlier possiamo vedere che in questo caso non ce ne sono, quindi non esistono dei profitti superiori a 12\$ per bottiglia.

3.4 Classifica venditori

Dalla seguente analisi vogliamo visualizzare una classifica dei dieci negozi con il più alto numero di vendite nello stato dell'Iowa.

Raggruppando all'interno del nostro dataset in base al nome del venditore e al numero di bottiglie vendute possiamo ottenere la classifica dei venditori desiderata.

```
print(df.groupby('Vendor Name')['Bottles Sold'].sum().sort_values(ascending=False).head(10))
df.groupby('Vendor Name')['Bottles Sold'].sum().sort_values(ascending=False).head(10).plot(kind='barh', figsize=(20,10))
plt.ylabel('Fornitore')
plt.title('Bottiglie vendute')
plt.show()
```

Figura 3.12: Codice classifica venditori

Nel seguente grafico a barre possiamo vedere, ordinati per numero di bottiglie vendute, i primi dieci negozi, così da capire il negozio con vendite maggiori.

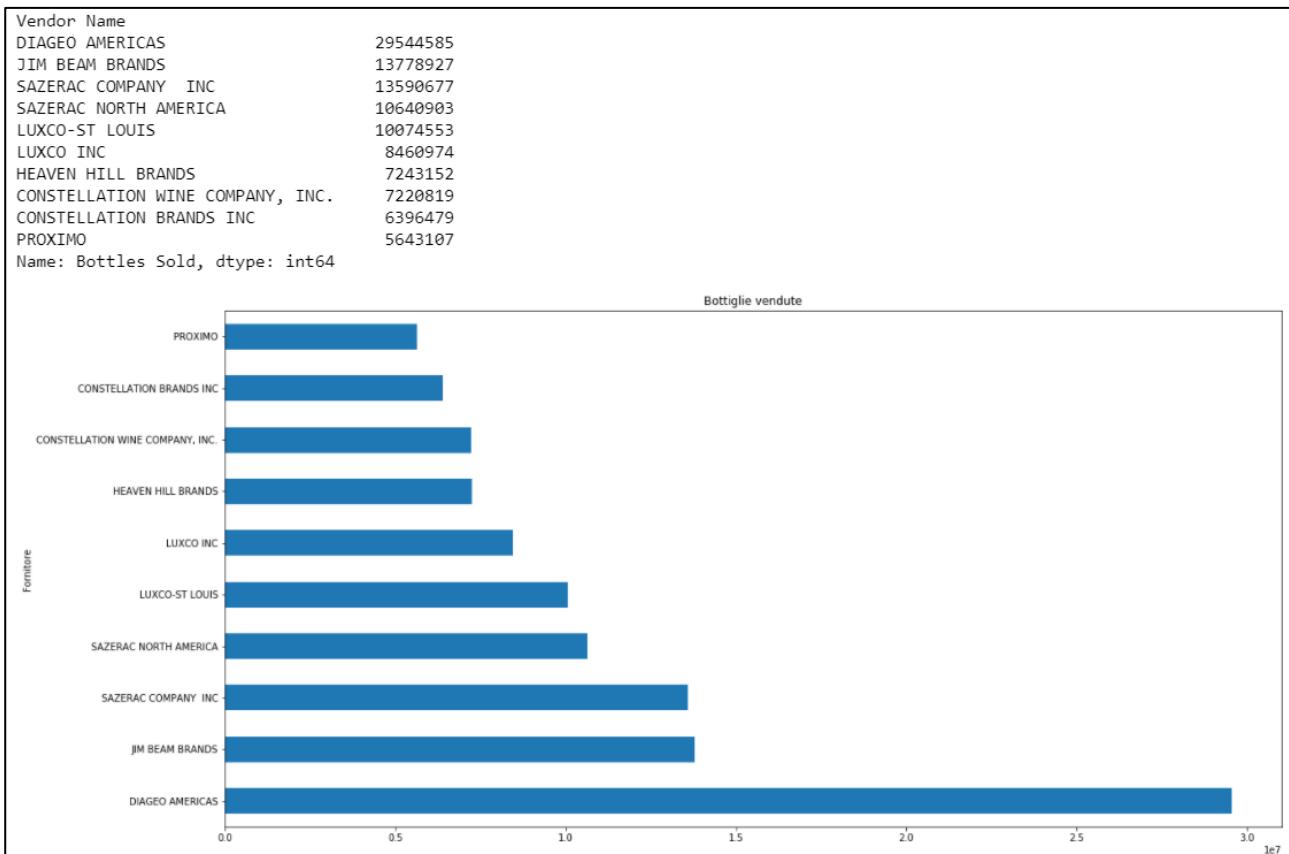


Figura 3.13: Grafico a barre classifica dei venditori

Possiamo dedurre dal grafico precedente che il negozio con il numero di vendite maggiore risulta essere il “Diageo Americas” con 29.544.585 bottiglie vendute dal 1° gennaio 2012 al 2019.

Diageo PLC



Diageo plc è una multinazionale inglese operante nel settore delle bevande alcoliche e fa parte delle cento aziende con la maggior capitalizzazione nella borsa di Londra. Il gruppo è nato nel 1997 dalla fusione di due compagnie: la britannica GrandMet e l'irlandese Guinness Plc.

Il gruppo produce principalmente distillati (Vodka, Whisky, Gin, Rum, ecc.), ma anche birra (Guinness, Kilkenny) e vino (principalmente per i mercati nord-americano e nord-europeo). Nel 2018, secondo la classifica della categoria stilata da Drinks International la divisione premium Diageo Reserve è risultata la preferita sul mercato americano di whiskey e vodka; Il punto di forza dell'azienda, è una combinazione unica di rigore ed estro, in cui ci avvale di collaborazione con alcuni degli attori più innovativi e dinamici della tecnologia, della cultura e dell'imprenditoria. Questo per rimanere concentrati sul futuro e collegati alle tendenze, alle piattaforme e alle tecnologie emergenti. Nel 2019 Diageo ha riportato ricavi per 12,9 miliardi di sterline (pari a circa 15 miliardi di €) e generato utili per 4 miliardi di sterline (4,65 miliardi di €); il numero dei dipendenti impiegati è pari a 28.150 unità. Nel febbraio 2020 la multinazionale britannica è stata sanzionata per 5 milioni di dollari dalla Security Exchange per aver spinto distributori sul mercato nordamericano a rifornirsi dei suoi prodotti con lo scopo di migliorare le proprie performance finanziarie nel periodo 2014-2015.

3.5 Classifica tipologia di alcolici

Dalla seguente analisi vogliamo visualizzare una classifica delle dieci tipologie di alcolici con il più alto numero di vendite nell'Iowa.

Raggruppando all'interno del nostro dataset in base alla tipologia dell'alcolico e al numero di bottiglie vendute possiamo ottenere la classifica degli alcolici più venduti.

```
print(df.groupby('Item Description')['Bottles Sold'].sum().sort_values(ascending=False).head(10))
df.groupby('Item Description')['Bottles Sold'].sum().sort_values(ascending=False).head(10).plot(kind='barh', figsize=(20,10))
plt.ylabel('Descrizione alcolico')
plt.title('Bottiglie vendute')
plt.show()
```

Figura 3.14: Codice classifica tipologia di alcolici

Nel seguente grafico a barre possiamo vedere, ordinati per numero di bottiglie vendute, i primi dieci alcolici venduti così da capire il preferito.

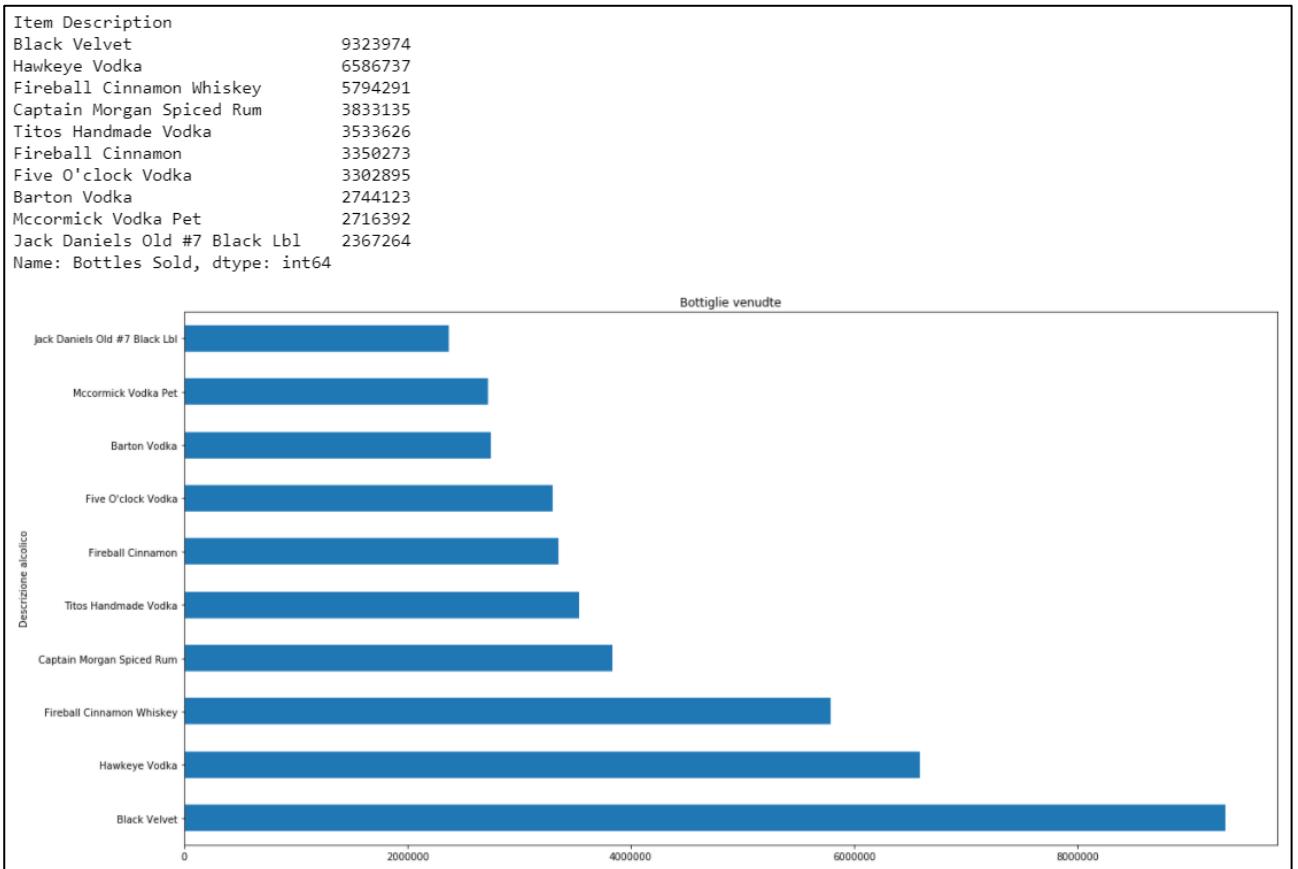


Figura 3.15: Grafico a barre classifica tipologia di alcolici

Possiamo dedurre dal grafico precedente che la tipologia di alcolici con il numero di vendite maggiore risulta essere il "Black Velvet" con 9.323.974 bottiglie vendute dal 1° gennaio 2012 al 2019.

Black Velvet



Il **Black velvet** (velluto nero) è un cocktail creato in Inghilterra a base di birra stout e vino bianco spumante (tradizionalmente Guinness e champagne). Il cocktail fu inventato al club per gentiluomini Brooks's di Londra nel 1861, in occasione della morte del Principe Consorte Alberto, per simboleggiare l'unione empatica per il lutto fra l'aristocrazia (champagne) e il popolo (birra scura).

Inoltre, esiste un metodo di preparazione detto build che permetteva di creare una banda nera a somiglianza di una fascia a lutto, permettendo ai gentiluomini del club, anche in tali circostanze, di poter bere lo champagne. Il nome deriva appunto dall'aspetto della "fascia".

3.6 Classifica tipologia di alcolici

Dalla seguente analisi vogliamo visualizzare una classifica delle dieci tipologie di alcolici con il più alto valore di vendite, nello stato dell'Iowa.

Raggruppando all'interno del nostro dataset in base alla tipologia di alcolici e alle vendite possiamo ottenere una classifica delle tipologie di alcolici più venduti.

```
print(df.groupby('Item Description')['Sale (Dollars)'].sum().sort_values(ascending=False).head(10))
df.groupby('Item Description')['Sale (Dollars)'].sum().sort_values(ascending=False).head(10).plot(kind='barh', figsize=(20,10))
plt.ylabel('Descrizione alcolico')
plt.title('Vendite (in Dollari)')
plt.show()
```

Figura 3.16: Codice classifica tipologia di alcolici

Nel seguente grafico a barre possiamo vedere, ordinate per numero di bottiglie vendute crescente, le prime dieci tipologie di alcolici venduti così da capire la tipologia di alcolici con il più alto valore di vendita.

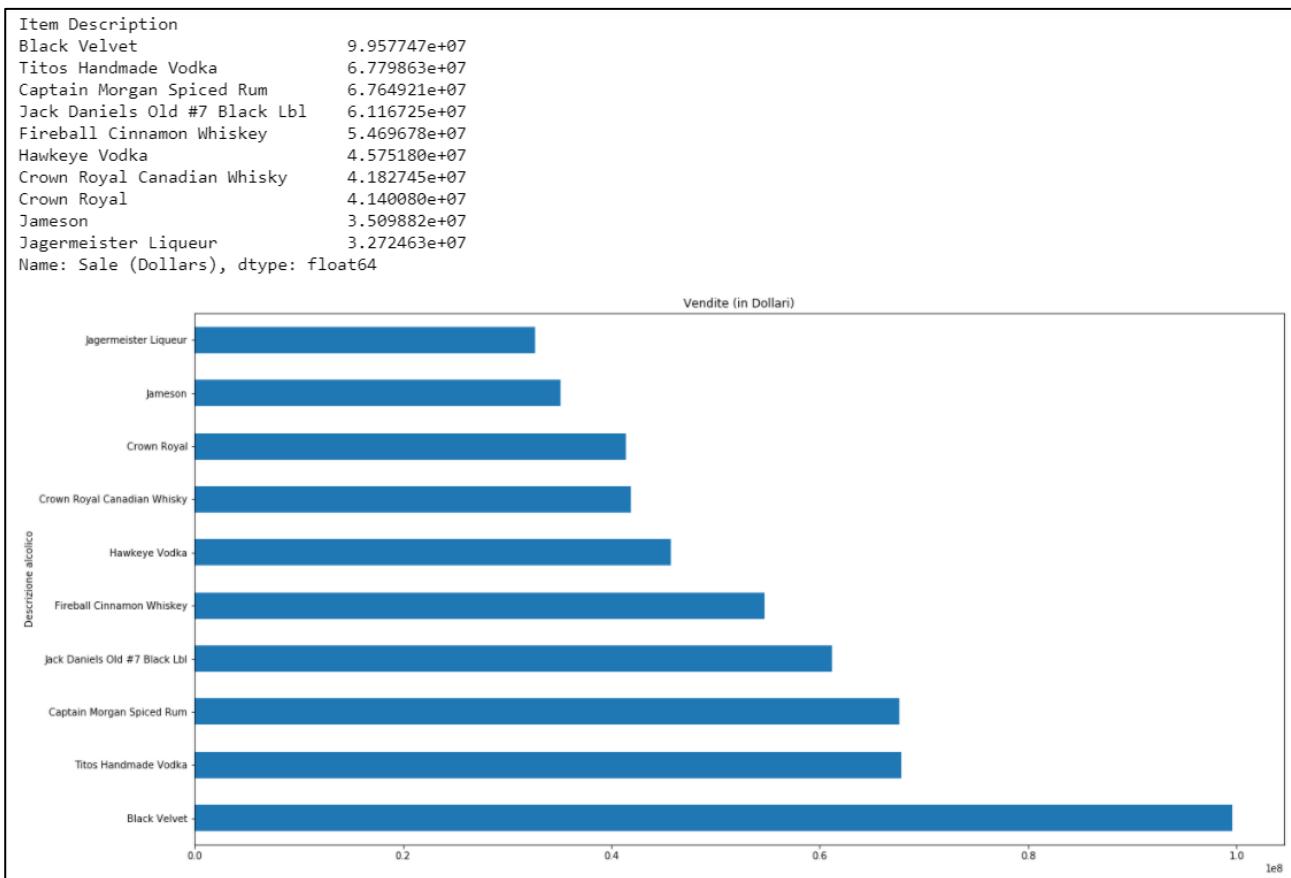


Figura 3.17: Grafico a barre classifica tipologia di alcolici

Possiamo dedurre dal grafico precedente che la tipologia di alcolici con vendite maggiori risulti essere "Black Velvet" registrando 99.577.470 dollari di bottiglie vendute dal 1° gennaio 2012 al 2019. Dunque, tale risultato si incarta con il risultato ottenuto dall'analisi precedente permettendo quindi di affermare che l'alcolico "Black Velvet" risulta essere l'alcolico che genera maggiori vendite e risulta il preferito nell'Iowa.

3.7 Classifica categoria di alcolici

Dalla seguente analisi vogliamo visualizzare una classifica delle dieci categorie di alcolici con il più alto valore di vendite dell'Iowa.

Raggruppando all'interno del nostro dataset in base alla categoria di alcolici e alle vendite possiamo ottenere una classifica delle categorie di alcolici più venduti.

```
print(df.groupby('Category Name')['Sale (Dollars)'].sum().sort_values(ascending=False).head(10))
df.groupby('Category Name')['Sale (Dollars)'].sum().sort_values(ascending=False).head(10).plot(kind='barh', figsize=(20,10))
plt.ylabel('Categoria alcolico')
plt.title('Vendite per categoria di alcolico')
plt.show()
```

Figura 3.18: Codice classifica categoria di alcolici

Nel seguente grafico a barre possiamo vedere, ordinate per valore di vendita crescente, le prime dieci categorie di alcolici venduti così da capire la categoria di alcolici più venduta.

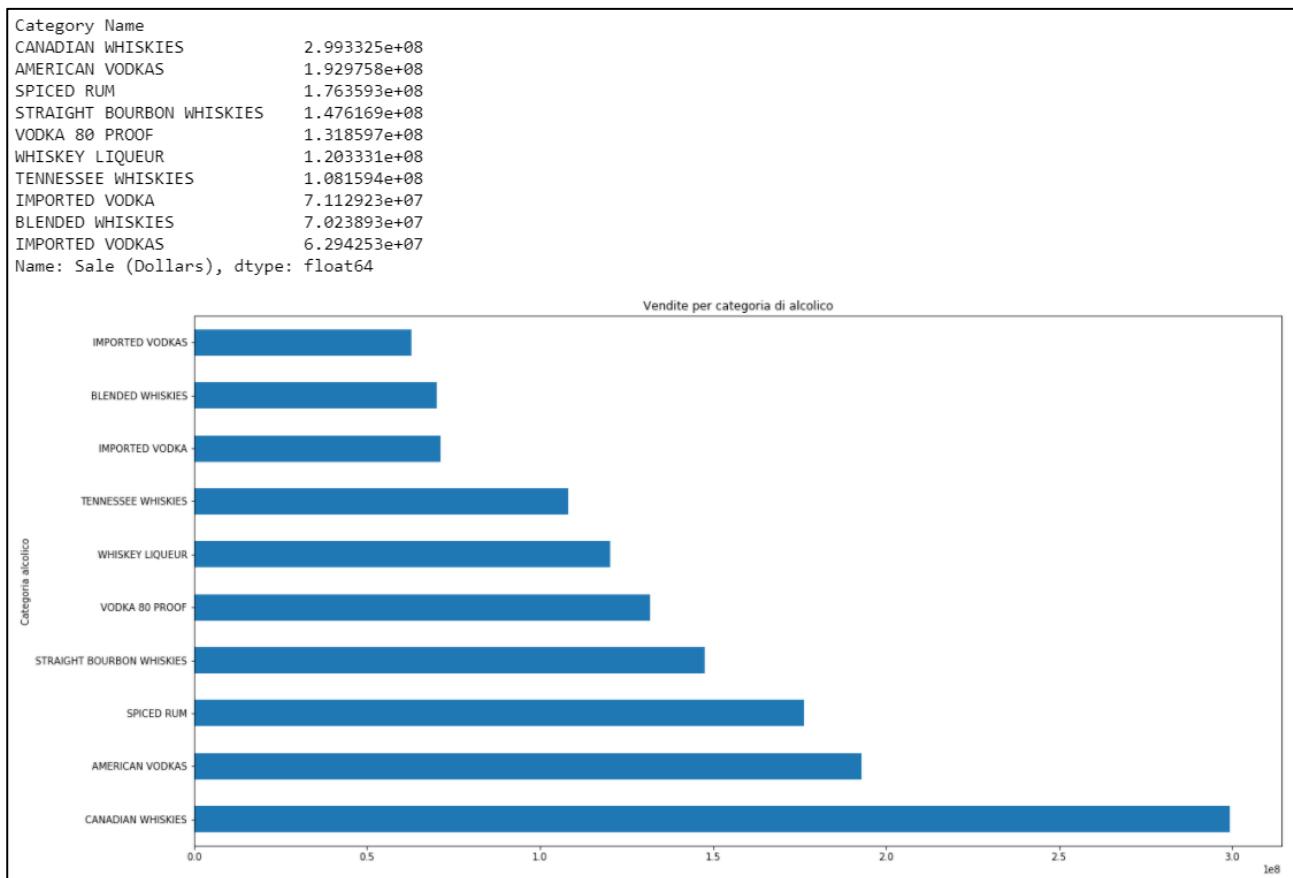


Figura 3.19: Grafico a barre classifica categoria di alcolici

Possiamo dedurre dal grafico precedente che la categoria di alcolici con il valore più alto di vendita risulta essere il “Canadian Whiskies” registrando 299.332.500 dollari di bottiglie vendute dal 1° gennaio 2012 al 2019.

Canadian Whiskies



Il **whisky canadese** è un tipo di whisky prodotto in Canada. La maggior parte dei essi sono liquori multi-grano miscelati contenenti una grande percentuale di alcolici di mais e sono in genere più leggeri e più morbidi rispetto ad altri stili di whisky. Quando i distillatori canadesi iniziarono ad aggiungere piccole quantità di chicchi di segale, molto saporiti, ai loro purè, la gente iniziò a chiedere questo nuovo whisky aromatizzato alla segale, chiamandolo semplicemente "segale". Oggi, come negli ultimi due secoli, i termini "whisky di segale" e "whisky canadese" sono usati in modo intercambiabile in Canada e (come definito nella legge canadese) si riferiscono esattamente allo stesso prodotto, che generalmente è realizzato con solo una piccola quantità di grano di segale.

3.8 Istogramma delle vendite

In questa analisi ci vuole scoprire ed individuare la frequenza delle vendite di alcolici nell'Iowa.

Per realizzare questo, si va ad analizzare la frequenza delle vendite e riducendo il margine a 350 dollari, in quanto oltre questo valore la frequenza delle vendite risulta essere molto bassa e tendente a zero.

```
plt.hist(df['Sale (Dollars)'], bins=1000)
plt.xlabel('Sale ($)')
plt.ylabel('Frequency')
plt.title('Histogram of Sales')
plt.xlim([0,350])
```

Figura 3.20: Codice frequenza delle vendite

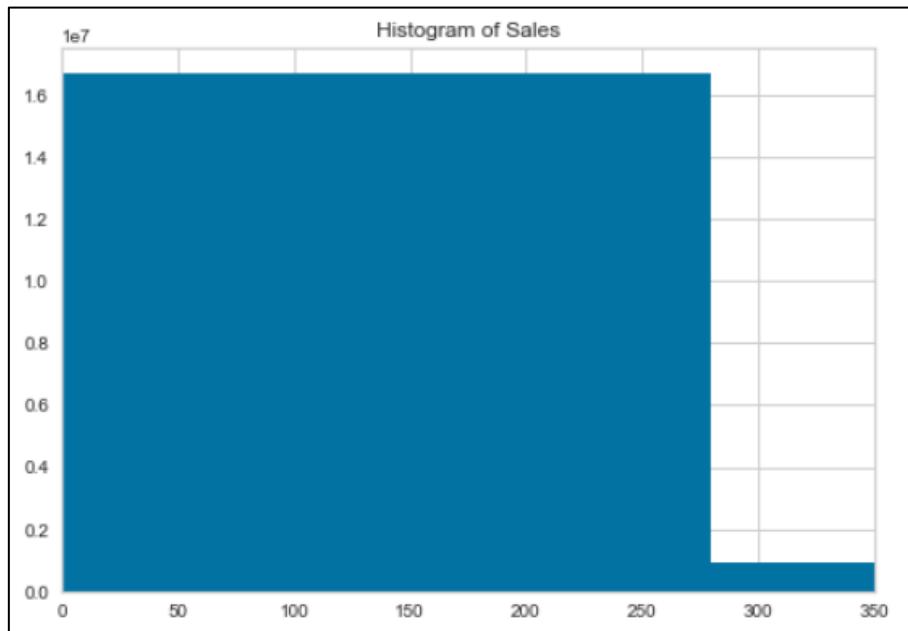


Figura 3.21: Istogramma delle vendite

Dal grafico possiamo dedurre che le vendite totali il cui valore è fino a 275 dollari hanno un elevata frequenza, mentre le vendite il cui valore è superiore a 275 dollari hanno una frequenza molto più bassa. Oltre i 350 dollari la frequenza delle vendite tende a zero e quindi per una visualizzazione migliore è stato impostato il limite a tale valore di vendita.

3.9 Istogramma delle bottiglie vendute

Analogamente all'analisi precedente si vuole scoprire la frequenza delle vendite di alcolici nell'Iowa considerando in questo caso il numero di bottiglie vendute. Si andrà dunque ad analizzare la frequenza delle bottiglie vendute.

```
plt.hist(df['Bottles Sold'], bins=1000)
plt.xlabel('Bottiglie vendute')
plt.ylabel('Frequency')
plt.title('Histogram of Sales')
plt.xlim([0,350])
```

Figura 3.22: Codice frequenza bottiglie vendute

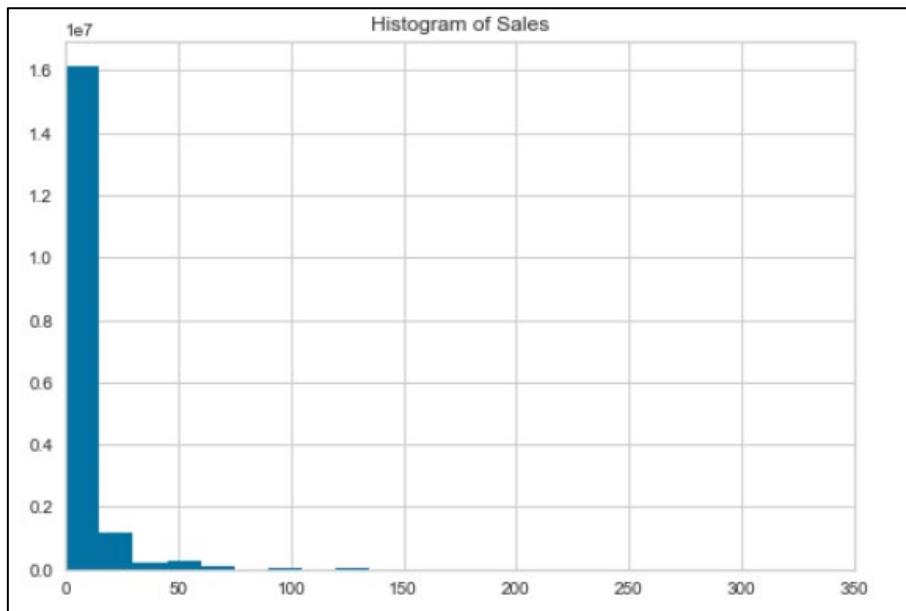


Figura 3.21: Istogramma bottiglie vendute

Dal grafico precedente possiamo dedurre che la frequenza delle bottiglie vendute risulta essere molto alta considerando circa dieci unità per transazione, e al crescere delle unità vendute vediamo che la frequenza diminuisce drasticamente.

4. Analisi descrittive per l'anno 2019

Le analisi fatte precedentemente sono state fatte prendendo in considerazione l'intero dataset e quindi relative all'intero ed ampio orizzonte temporale a disposizione. Adesso verranno ripresentate prendendo in considerazione il solo anno 2019, l'anno più recente prima che la pandemia da Coronavirus e le relative restrizioni influenzassero le normali attività quotidiane.

Per effettuare le analisi relative al 2019 con anche i dati relativi alla popolazione si è effettuato il merge tra i dataset df_2019 e Population_IA sul campo “County”.

```
tot_2019=pd.merge(df_2019,Population_IA , how='inner' , on='County')
```

	Invoice/Item Number	Date	Store Number	Store Name	Address	City	Zip Code	Store Location	County Number	County	...	State	Bottle Retail	Bottles Sold	Sale (Dollars)	Volume Sold (Liters)	Volume Sold (Gallons)	month	year	State	Population	Population In 10K
0	16839300007	INV- 2019-01-09	4655	Casey's General Store #237 / Prairie City	1002 W 2nd St	Prairie City	50228	POINT (-93.247591 41.596714)	50.0	JASPER	...	15.50	3	46.50	2.25	0.59	1	2019	IA	37185	3.7185	
1	16735300018	INV- 2019-01-04	2517	Hy-Vee Food Store #1 / Newton	1501 First Avenue East	Newton	50208	POINT (-93.034296 41.69916)	50.0	JASPER	...	8.22	12	98.64	9.00	2.37	1	2019	IA	37185	3.7185	
2	16737100005	INV- 2019-01-04	4604	Pit Stop Liquors / Newton	1324, 1st Ave E	Newton	50208	POINT (-93.035654 41.699173)	50.0	JASPER	...	18.68	1	18.68	0.50	0.13	1	2019	IA	37185	3.7185	
3	16735300020	INV- 2019-01-04	2517	Hy-Vee Food Store #1 / Newton	1501 First Avenue East	Newton	50208	POINT (-93.034296 41.69916)	50.0	JASPER	...	5.06	12	60.72	9.00	2.37	1	2019	IA	37185	3.7185	
4	16735300043	INV- 2019-01-04	2517	Hy-Vee Food Store #1 / Newton	1501 First Avenue East	Newton	50208	POINT (-93.034296 41.69916)	50.0	JASPER	...	7.20	12	86.40	6.00	1.58	1	2019	IA	37185	3.7185	

Figura 4.1: Codice e dataframe risultante

Il dataset su cui saranno effettuate le analisi sarà il seguente, raggruppato in maniera simile alle analisi relative a tutti gli anni:

```
stores_2019= tot_2019.groupby('Store Number',as_index=False)[['State', 'Bottle Retail', 'Bottles Sold', 'Population', 'Population in 10K', 'Volume Sold (Liters)']].mean()
print(stores_2019)
```

Figura 4.2: Codice creazione dataframe “stores_2019”

	Store Number	State	Bottle Retail	Bottles Sold	Population	Population in 10K	Volume Sold (Liters)
0	2106		16.652938	19.263920	131228.0	13.1228	17.851560
1	2113		16.270658	3.753722	35904.0	3.5904	3.469175
2	2130		17.425406	20.012935	131228.0	13.1228	18.387712
3	2178		16.551887	9.085278	13687.0	1.3687	9.401947
4	2190		22.603088	8.408258	490161.0	49.0161	5.494724
..
856	9023		56.250000	10.500000	20165.0	2.0165	7.875000
857	9037		26.328125	15.750000	33657.0	3.3657	11.812500
858	9038		37.700000	990.000000	20165.0	2.0165	742.500000
859	9039		15.005455	125.090909	490161.0	49.0161	91.227273
860	9041		25.796369	12.678571	14813.0	1.4813	9.508929

Figura 4.3: Dataframe “stores_2019”

Per effettuare particolari analisi viene calcolato anche il campo “Profit_per_sale”, che permette di determinare i profitti del venditore:

```
tot_2019['Profit_per_sale']=((tot_2019['State', 'Bottle Retail']-tot_2019['State', 'Bottle Cost'])*tot_2019['State', 'Bottle Cost'])*1.18
```

Figura 4.4: Creazione campo “Profit_per_sale”

Successivamente si andrà a raggruppare per numero di negozio mostrando il valore di “Profit_per_sale”. L’obiettivo è avere un dataframe che permette di avere i profitti raggruppati per negozio, tale dataframe verrà chiamato “profit”.

```
x=tot_2019['Profit_per_sale'].groupby(tot_2019['Store Number']).sum()
print(len(x))
profit=pd.DataFrame(x);
```

861

Figura 4.5: Creazione dataframe “profit”

profit.reset_index(level=0, inplace=True)		
	Store Number	Profit_per_sale
0	2106	4.762394e+05
1	2113	1.533957e+05
2	2130	4.021997e+05
3	2178	2.523608e+05
4	2190	4.047641e+06

Figura 4.6: Dataframe “profit”

In ultimo, viene effettuato un merge, quindi un’unione tra i due dataframe che sono “stores_2019” che contiene un insieme di campi interessanti per l’analisi e il dataframe “profit” che contiene i profitti per negozio. A questo punto il dataframe che otterremo è quello che verrà considerato per le analisi descrittive per l’anno 2019 ed è “stores_tot”.

```
stores_tot=pd.merge(stores_2019, profit, on='Store Number')
```

Figura 4.7: Codice merge

Store Number	State	Bottle Retail	Bottles Sold	Population	Population in 10K	Volume Sold (Liters)	Profit_per_sale
0	2106	16.652938	19.263920	131228.0	13.1228	17.851560	4.762394e+05
1	2113	16.270658	3.753722	35904.0	3.5904	3.469175	1.533957e+05
2	2130	17.425406	20.012935	131228.0	13.1228	18.387712	4.021997e+05
3	2178	16.551887	9.085278	13687.0	1.3687	9.401947	2.523608e+05
4	2190	22.603088	8.408258	490161.0	49.0161	5.494724	4.047641e+06

Figura 4.8: Dataframe “stores_tot”

4.1 Prezzo medio per bottiglia

In questa analisi si vuole determinare e mostrare il prezzo medio della vendita al dettaglio delle bottiglie per l'anno 2019.

Come prima analisi descrittiva viene mostrata una panoramica del prezzo medio per bottiglia (“State Bottle Retail”). Nell'analisi andremo a considerare solo le bottiglie che hanno un costo unitario inferiore a 30 dollari, in quanto le bottiglie con un prezzo maggiore di 30 dollari si è visto sono piuttosto rare.

```
print(stores_tot[stores_tot['State Bottle Retail']<30]['State Bottle Retail'].mean())
sns.distplot(stores_tot[stores_tot['State Bottle Retail']<30]['State Bottle Retail']);
```

Figura 4.9: Codice per valutazione del prezzo medio

L'output relativo è il grafico che ci permette di valutare i valori di questa analisi ed è mostrato nella figura seguente:

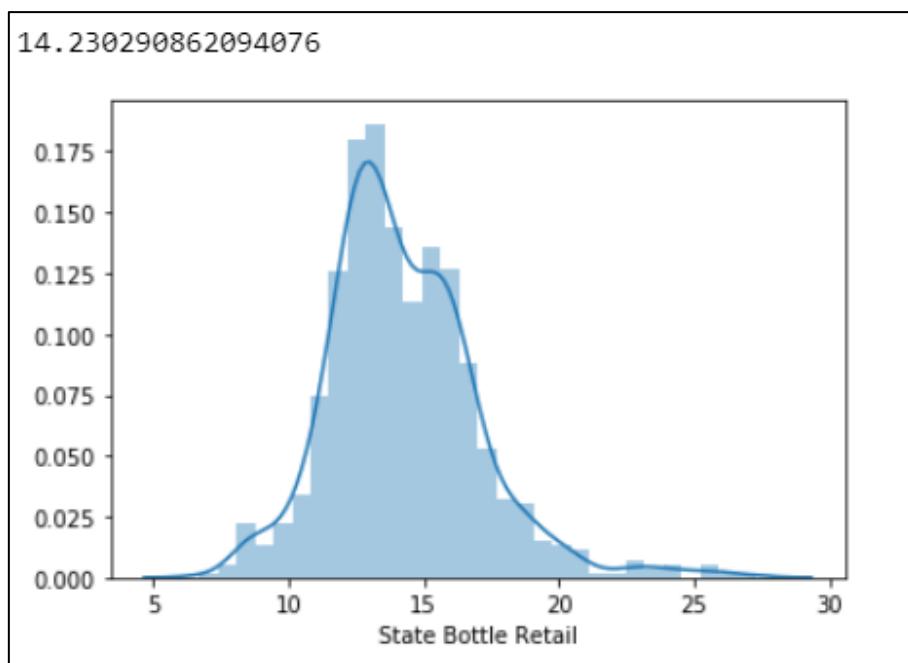


Figura 4.10: Istogramma prezzo medio per bottiglia

Possiamo dedurre da questa panoramica, in analogia alle analisi per tutto l'intervallo temporale, che ci sono negozi che vendono bottiglie sia “costose” che “economiche” ma in medie le bottiglie vendute nel 2019 hanno un prezzo pari a 14,23 dollari, rispetto ai 13,78 dollari che risultavano dalla panoramica generale.

```
print(len(stores_tot))
print(len(stores_tot[stores_tot['State Bottle Retail']>=30]))
```

861
2

Figura 4.11: Individuazione degli outlier

Vediamo inoltre come gli outlier, o meglio le bottiglie che hanno un prezzo al dettaglio di oltre 30 dollari, sono solamente due.

4.2 Bottiglie vendute

In questa analisi si andrà ad individuare il numero medio di bottiglie al dettaglio vendute, nell'anno 2019. Viene mostrata una panoramica delle bottiglie vendute (“Bottle Sold”) escludendo nell’analisi le vendite con una quantità di bottiglie maggiore di 50 unità, in quanto corrispondono a vendite all’ingrosso.

A livello implementativo avremo il seguente codice e il relativo output:

```
print(stores_tot[stores_tot['Bottles Sold']<50]['Bottles Sold'].mean())
sns.distplot(stores_tot[stores_tot['Bottles Sold']<50]['Bottles Sold']);
```

Figura 4.12: Codice per valutazione delle bottiglie vendute

L’output relativo è il grafico che ci permette di valutare i valori di questa analisi ed è mostrato nella figura seguente:

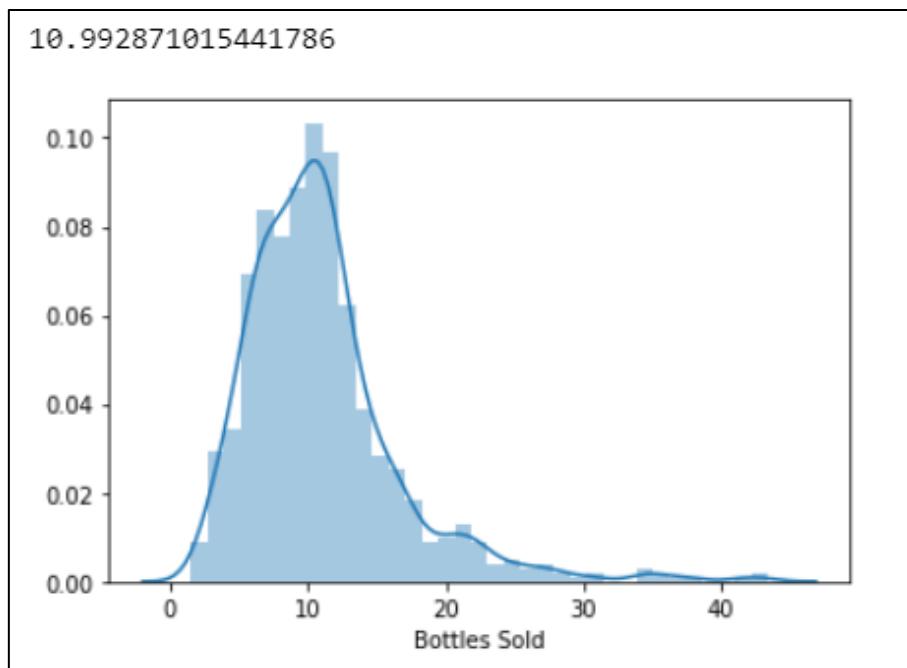


Figura 4.13: Istogramma bottiglie vendute

Considerando la vendita al dettaglio di alcolici possiamo vedere che in media vengono vendute circa 11 bottiglie per transazione.

Anche questo risultato è piuttosto in linea con la media delle bottiglie vendute per tutti gli anni.

```
print(len(stores_tot))
print(len(stores_tot[stores_tot['Bottles Sold']>=50]))
```

859
2

Figura 4.14: Individuazione degli outlier

Vediamo che gli outlier e dunque il numero di bottiglie vendute per transazione oltre le 50 unità sono solamente due.

4.3 Profitti per negozio

In questa analisi si andrà ad individuare il valore medio dei profitti dei negozi di liquori per l'anno 2019. L'implementazione per questa analisi è mostrata in figura:

```
print(stores_tot[stores_tot['Profit_per_sale']<300000]['Profit_per_sale'].mean())
sns.distplot(stores_tot[stores_tot['Profit_per_sale']<300000]['Profit_per_sale']);
```

Figura 4.15: Codice per valutazione profitti per negozio

L'output relativo è il grafico che ci permette di valutare i valori di questa analisi ed è mostrato nella figura seguente:

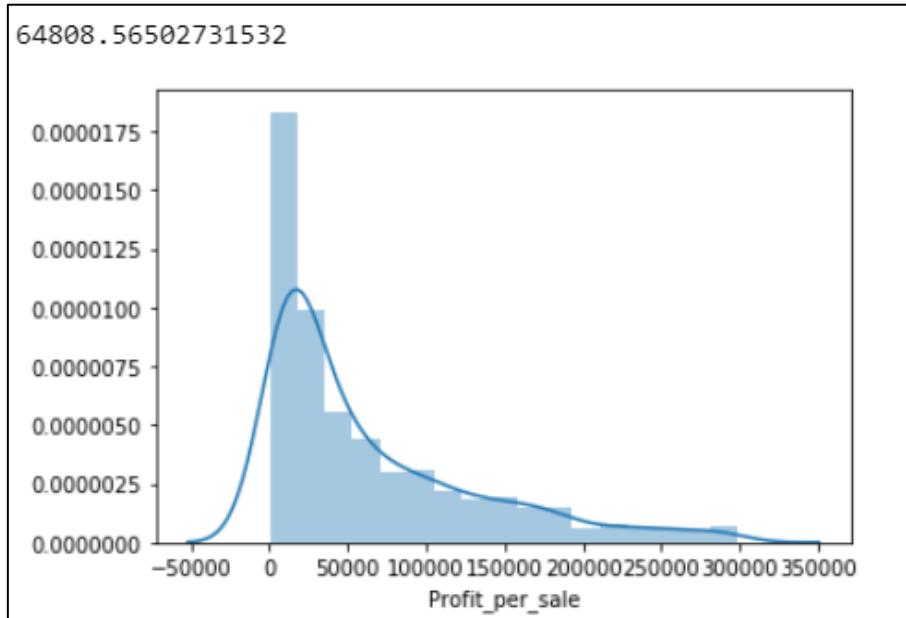


Figura 4.16: Istogramma profitti per negozio

Vediamo come il valore medio dei profitti per l'anno di analisi è pari a circa 64.809 dollari.

```
print(len(stores_tot))
print(len(stores_tot[stores_tot['Profit_per_sale']>=300000]))
857
83
```

Figura 4.17: Individuazione degli outlier

Si nota che in questo caso gli outlier e dunque il numero di profitti superiore a 300.000 dollari sono circa 83, abbastanza rispetto gli outlier dovuti ai casi precedenti. Dovuto al fatto che probabilmente alti profitti sono collegati alle vendite all'ingrosso in cui una elevata quantità di bottiglie vendute è correlata al maggior profitto.

4.4 Correlazione tra la popolazione ed i profitti annuali

Nella seguente analisi si prende in esame la popolazione ed i profitti per andare ad individuare eventuali pattern o correlazioni tra queste due variabili.

```
sns.regplot(x=stores_tot['Population'], y=stores_tot['Profit_per_sale'], fit_reg=False)
```

Figura 4.18: Codice per correlazione tra popolazione ed i profitti annuali

L'output relativo è il grafico che ci permette di valutare i valori di questa analisi ed è mostrato nella figura seguente:

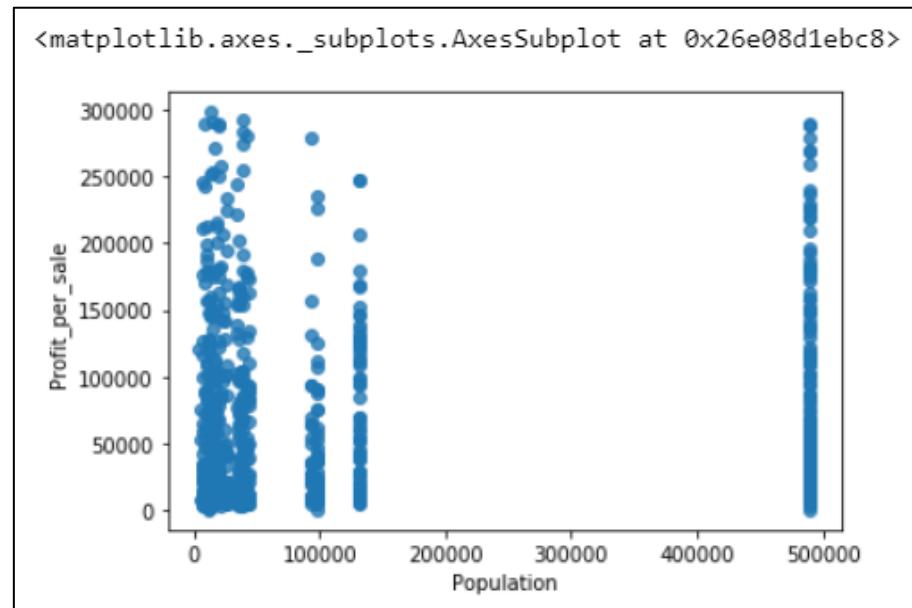


Figura 4.19: Grafico correlazione tra popolazione ed i profitti annuali

Da questa analisi possiamo dedurre che la popolazione non sembra avere molta correlazione con i profitti annuali, la popolazione cade in fasce specifiche.

Dunque, per dettagliare meglio tale analisi andremo a discretizzare la popolazione in “bins”, in intervalli, per avere un quadro più chiaro della situazione:

```
bins = [0, 30000, 50000, 70000, 100000, 200000, 300000, 500000]
stores2=stores_tot
stores2['pop_bins'] = pd.cut(stores2['Population'], bins)
```

Figura 4.20: Codice discretizzazione

Creiamo un grafico per rappresentare la popolazione discretizzata e il profitto annuo:

```
ax = sns.barplot(x=stores2['pop_bins'], y=stores2['Profit_per_sale'])
ax.set(xlabel='Population of County', ylabel='Annual Profit')
labels = ax.get_xticklabels() # get x labels
ax.set_xticklabels(labels, rotation=30)
plt.show()
```

Figura 4.21: Codice creazione grafico

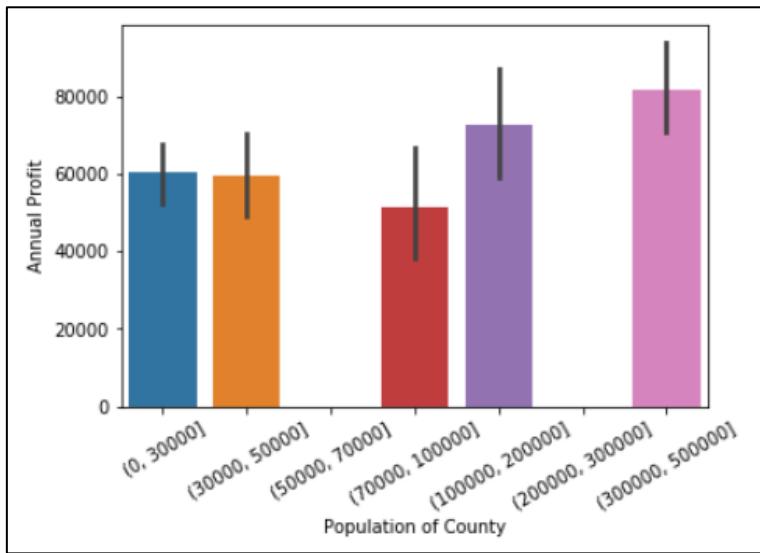


Figura 4.22: Grafico popolazione e profitto annuo

Da questo grafico possiamo denotare che le contee densamente popolate mostrano elevati profitti annuali, però è giustificato dal fatto che è presente un maggior numero di persone. D'altro canto, ci sono contee a bassa densità di popolazione che mostrano un numero di profitti annuali più alto di quelle mediamente popolate.

Possiamo quindi concludere dicendo che c'è molta variabilità tra le due variabili prese in esame, dettata dalla non effettiva correlazione delle due variabili.

4.5 Bottiglie vendute per ordine

A questo punto andremo ad esaminare le bottiglie vendute con i profitti annuali per scoprire la relazione esistente tra le due variabili di interesse.

```
sns.regplot(x=stores_tot['Bottles Sold'], y=stores_tot['Profit_per_sale'])
```

Figura 4.23: Codice correlazione tra bottiglie vendute per ordine ed i profitti di vendita

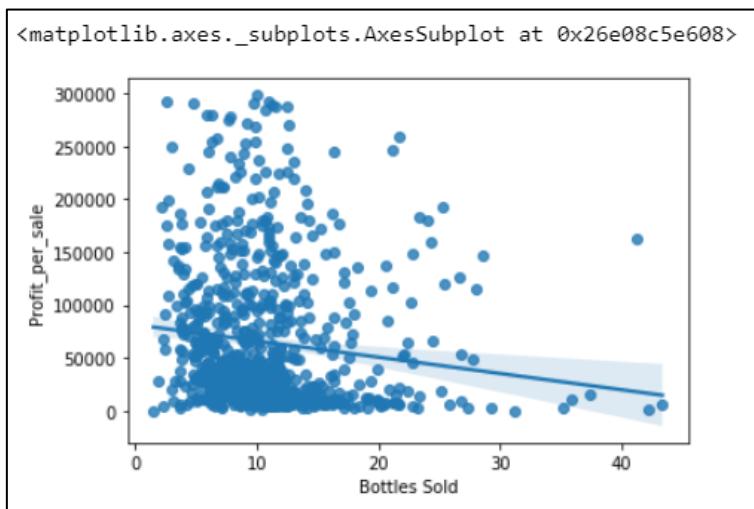


Figura 4.24: Grafico correlazione tra bottiglie vendute per ordine ed i profitti di vendita

Deduciamo da questo grafico che i negozi con un alto profitto annuale hanno di solito un numero medio-basso di bottiglie per vendita, mentre i negozi con un numero di vendite elevato presentano tendenzialmente profitti bassi. Anche in questo caso andremo a discretizzare, e in particolare discretizzeremo il numero di bottiglie vendute.

```
bins = [0, 10, 20, 30, 40, 50]
stores2=stores_tot
stores2['bottles_bins'] = pd.cut(stores2['Bottles Sold'], bins)
```

Figura 4.25: Codice discretizzazione

```
ax = sns.barplot(x=stores2['price_bins'], y=stores2['Profit_per_sale'])
ax.set(xlabel='Average Bottle Price per Order', ylabel='Annual Profit')
labels = ax.get_xticklabels() # get x labels
ax.set_xticklabels(labels, rotation=30)
plt.show()
```

Figura 4.26: Codice creazione grafico

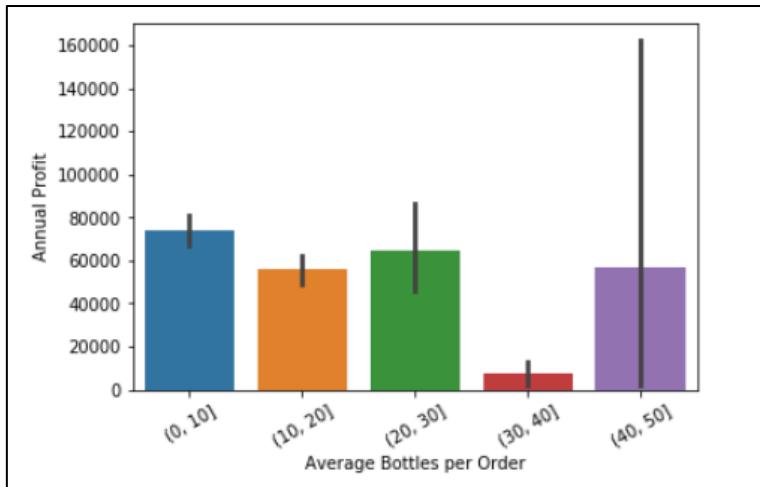


Figura 4.27: Grafico numero bottiglie medie per ordine e profitto annuo

Ovviamente il grafico discretizzato rispetta le deduzioni fatte in quanto è una rappresentazione diversa della stessa entità, ma in questo caso è interessante poter avere a colpo d'occhio il valore dei profitti per fasce di bottiglie vendute. In particolare, vediamo come le vendite maggiori si concentrano su bottiglie vendute entro le 10 unità. La fascia che registra minori vendite è quella di bottiglie vendute comprese tra le 30 e le 40 unità. Nella fascia tra le 40 e le 50 unità vendute si ha una crescita dei profitti a monito di un acquisto all'ingrosso.

In conclusione, possiamo dedurre che la fascia tra le 30 e le 40 unità non rispetti i canoni di un acquisto al dettaglio (acquisti di poche unità) e all'ingrosso (acquisti di molte unità), conferendo bassi profitti.

4.6 Prezzo medio delle bottiglie per ordine

In questa sezione si esaminerà il prezzo per bottiglia ed i relativi profitti annuali, sempre per l'anno 2019. Nella figura seguente è mostrato il codice ed il relativo grafico di output:

```
sns.regplot(x=stores_tot['State Bottle Retail'], y=stores_tot['Profit_per_sale'], fit_reg=True)
```

Figura 4.28: Codice correlazione tra prezzo medio delle bottiglie per ordine ed i profitti di vendita

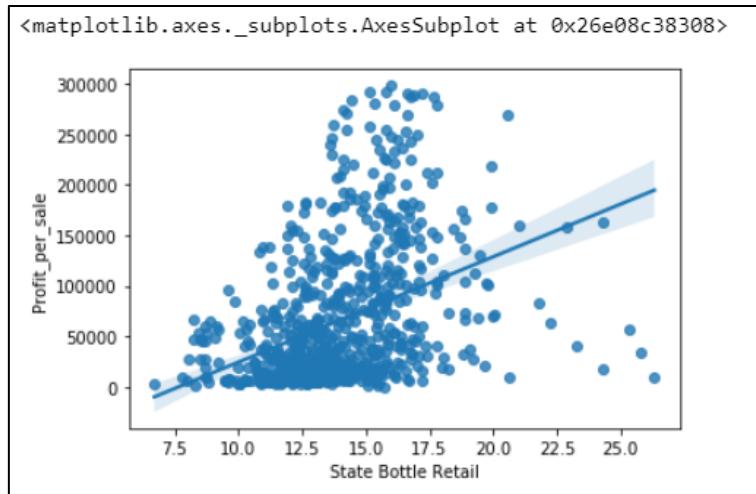


Figura 4.29: Grafico correlazione tra prezzo medio delle bottiglie per ordine ed i profitti di vendita

Quello che è interessante notare è che i profitti sono concentrati maggiormente nelle vendite di bottiglie con un prezzo pari al prezzo medio e che va quindi da circa 10 a 16 dollari.

Mantenendo l'idea della discretizzazione in intervalli, anche in questo caso si ha un grafico che mostra i profitti per fasce di prezzo. In figura seguente è mostrato il codice ed il relativo grafico:

```
bins = [5, 10, 15, 20, 25, 30]
stores2=stores_tot
stores2['price_bins'] = pd.cut(stores2['State Bottle Retail'], bins)
```

Figura 4.30: Codice discretizzazione

```
ax = sns.barplot(x=stores2['price_bins'], y=stores2['Profit_per_sale'])
ax.set(xlabel='Average Bottle Price per Order', ylabel='Annual Profit')
labels = ax.get_xticklabels() # get x labels
ax.set_xticklabels(labels, rotation=30)
plt.show()
```

Figura 4.31: Codice creazione grafico

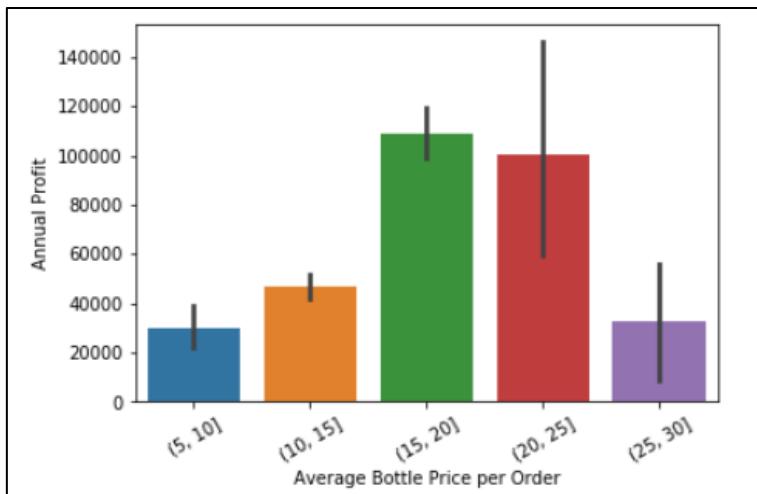


Figura 4.32: Grafico prezzo medio delle bottiglie per ordine ed i profitti di vendita

A partire dal grafico ottenuto si evince come le bottiglie vendute con un prezzo medio unitario tra 15 e 20 dollari sono quelle che conferiscono un profitto annuale più alto. In “seconda posizione”, a seguire abbiamo le bottiglie con fascia di prezzo da 20 a 25 dollari. La più bassa fascia di prezzo analizzata, compresa tra i 5 e i 10 dollari, non conferisce profitti elevati.

5. Clustering

In questa sezione verranno presentate alcune attività di clustering eseguite sui nostri dati. L’idea alla base della Clusterizzazione è individuare delle strutture naturali nei dati. Il risultato sarà che “oggetti” simili o comportamenti simili verranno raggruppati all’interno di insiemi detti appunto, cluster.

Dal punto di vista dei dati utilizzati nell’analisi, si vanno a considerare i dati relativi al solo anno 2019. Il dataframe di interesse è quindi “df_2019”.

Alla fase di clustering precede una fase di preprocessing.

Questa fase, molto importante, nel nostro caso consiste in alcune semplici operazioni come la trasformazione da ml a cl della quantità di alcolici venduta in modo da non avere problemi di scaling con altre variabili che utilizzeremo nel Cluster. L’idea è avere lo stesso range di valori.

Inoltre, filtriamo il dataset in modo da considerare il prezzo della bottiglia sotto i 30 dollari, il quantitativo delle bottiglie vendute sotto le 50 e il valore delle vendite per transazione sotto i 130 dollari.

Con il codice seguente viene effettuato quanto appena detto:

```
df_2019['Bottle Volume (ml)']= df['Bottle Volume (ml)'].astype(int)
df_2019['Bottle Volume (cl)'] = df_2019['Bottle Volume (ml)'].apply(lambda x: x/10)

df_senzaout = (df_2019['State Bottle Retail']<30) & (df_2019['Bottles Sold']<50) & (df_2019['Sale (Dollars)']<130)

df2019_nout = df_2019[df_senzaout]
```

Figura 5.1: Codici preprocessing

A questo punto importando le opportune librerie è possibile iniziare la vera e propria fase di Clustering.

```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
```

Figura 5.2: Codici importazione librerie

Importiamo KMeans in quanto il K-Means è l’algoritmo di clustering utilizzato. Un algoritmo prototype-based e quindi basato su un oggetto di riferimento che nel nostro caso è il centroide, rappresentativo del cluster. Importiamo anche KElbowVisualizer per poter visualizzare attraverso il metodo Elbow il corretto numero di cluster da considerare. Nel K-Menas il valore K è proprio il numero di cluster che si vuole in output e deve essere settato a priori. Con il metodo Elbow stiamo il miglior valore di K.

5.1 Clustering bottiglie vendute e prezzo al dettaglio

Il primo cluster consiste nel mettere in relazione le bottiglie vendute ed il prezzo al dettaglio delle relative bottiglie, come mostrato nel codice seguente:

```
x=df2019_nout[['State Bottle Retail', 'Bottles Sold']].values  
print(x.shape)  
(1409601, 2)
```

Figura 5.3: Istanziamento degli elementi da clusterizzare

La “x” sarà costituita da 1.409.601 di righe e 2 colonne relative ai campi menzionati in precedenza.

Applicando il metodo Elbow otteniamo che il numero ideale di cluster risulta essere 4, quindi applicheremo un K-means con K=4.

```
model = KMeans()  
visualizer = KElbowVisualizer(model, k=(1,12))  
  
plt.figure(figsize=(15,8))  
visualizer.fit(x)  
visualizer.show();
```

Figura 5.4: Codice metodo Elbow

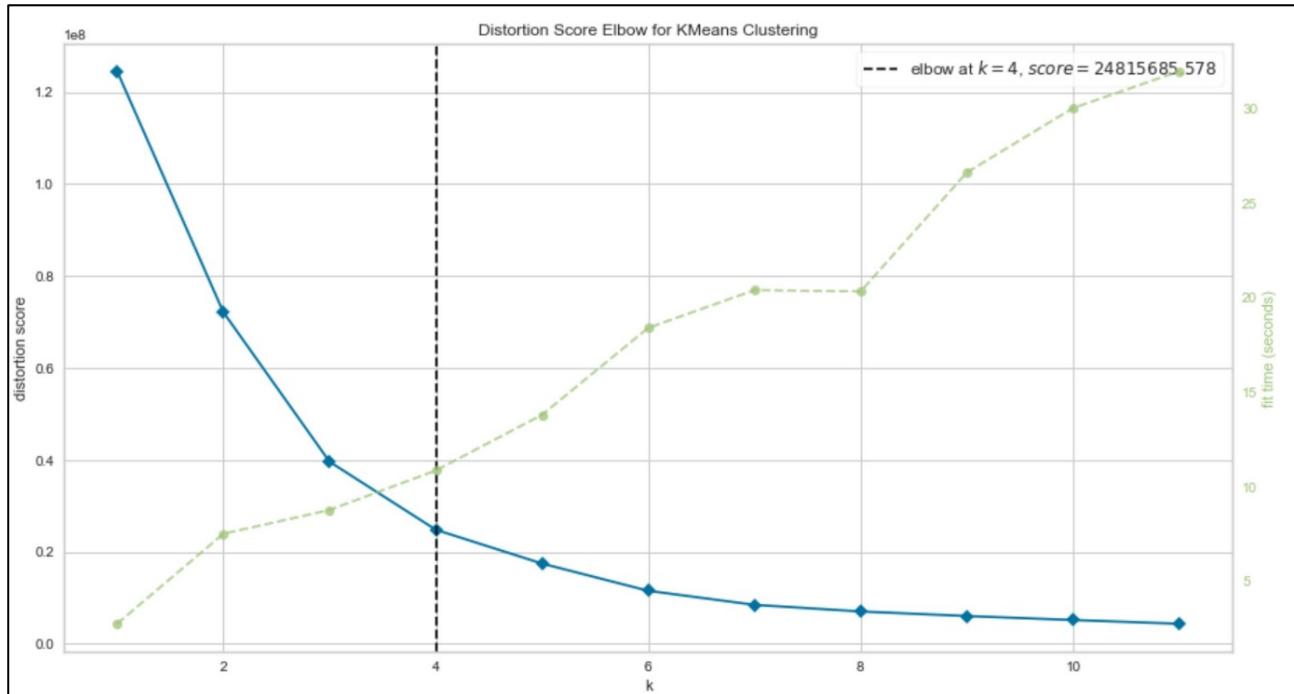


Figura 5.5: Grafico metodo Elbow per l'individuazione di K

Il codice relativo all'implementazione dell'algoritmo del K-means è riportato nella figura seguente:

```

km= KMeans(n_clusters = 4, init='k-means++', max_iter=300, n_init = 10, random_state = 0)
y_means = km.fit_predict(x)

plt.figure(figsize=(15,8))
plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s=100, c= 'yellow')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s=100, c= 'pink')
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s=100, c= 'magenta')
plt.scatter(x[y_means == 3, 0], x[y_means == 3, 1], s=100, c= 'cyan')
# plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s=100, c= 'orange')
# plt.scatter(x[y_means == 5, 0], x[y_means == 5, 1], s=100, c= 'red')
# plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s=100, c= 'red')

plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], s=50, c='blue', label= 'centroid')

plt.title("K MEANS", fontsize=20)
plt.xlabel("Prezzo")
plt.ylabel("Bottiglie vendute")
plt.legend()
plt.show()

```

Figura 5.6: Codice K-means

Il relativo output è il seguente:

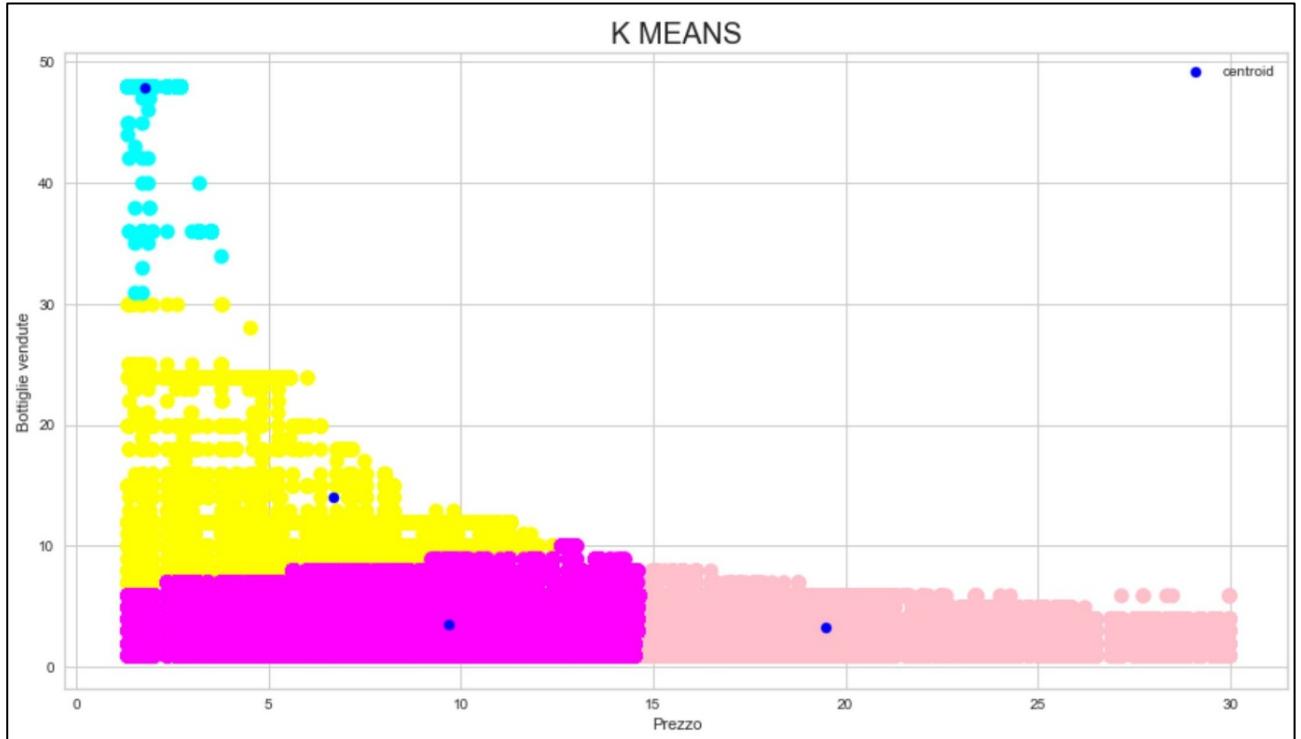


Figura 5.7: Grafico cluster ottenuti tramite K-means

In prima battuta possiamo notare dal grafico che le due variabili sembrerebbero non correlate. Di fatti, sono stati individuati quattro cluster attraverso i quali individuiamo che:

1. bottiglie vendute in gran quantità (dalle 30 a 50 bottiglie per transazione) hanno un bassissimo prezzo al dettaglio (cluster color azzurro ciano);
2. bottiglie vendute in media numerosità (tra le 10 e 30 bottiglie per transazione) sono al di sotto del prezzo medio (cluster color giallo);

3. un basso numero di bottiglie vendute (fino alle 10 bottiglie per transazione) si divide in due cluster: bottiglie al di sotto del prezzo medio che è circa 15 euro e bottiglie il cui prezzo è sopra quello medio (cluster color magenta e rosa).

A partire da tali cluster possiamo capire che per massimizzare il numero di bottiglie vendute bisogna vendere il prodotto ad un prezzo inferiore a quello medio, pari a circa 15 euro.

5.2 Clustering bottiglie vendute e volume per bottiglia

Dopo aver realizzato il primo cluster era interessante valutare e mettere in relazione le bottiglie vendute con il relativo volume per bottiglia. Quindi definiamo la “x” nella maniera seguente:

```
x=df2019_nout[['Bottle Volume (ml)', 'Bottles Sold']].values  
print(x.shape)  
(1409601, 2)
```

Figura 5.8: Istanziamento degli elementi da clusterizzare

Si nota che anche in questo caso la “x” conta di 1.409.601 righe e 2 colonne.

In maniera analoga al cluster precedente viene lanciato il metodo Elbow per stimare il corretto valore di K, in questo caso si nota che il K desiderato è pari a 3.

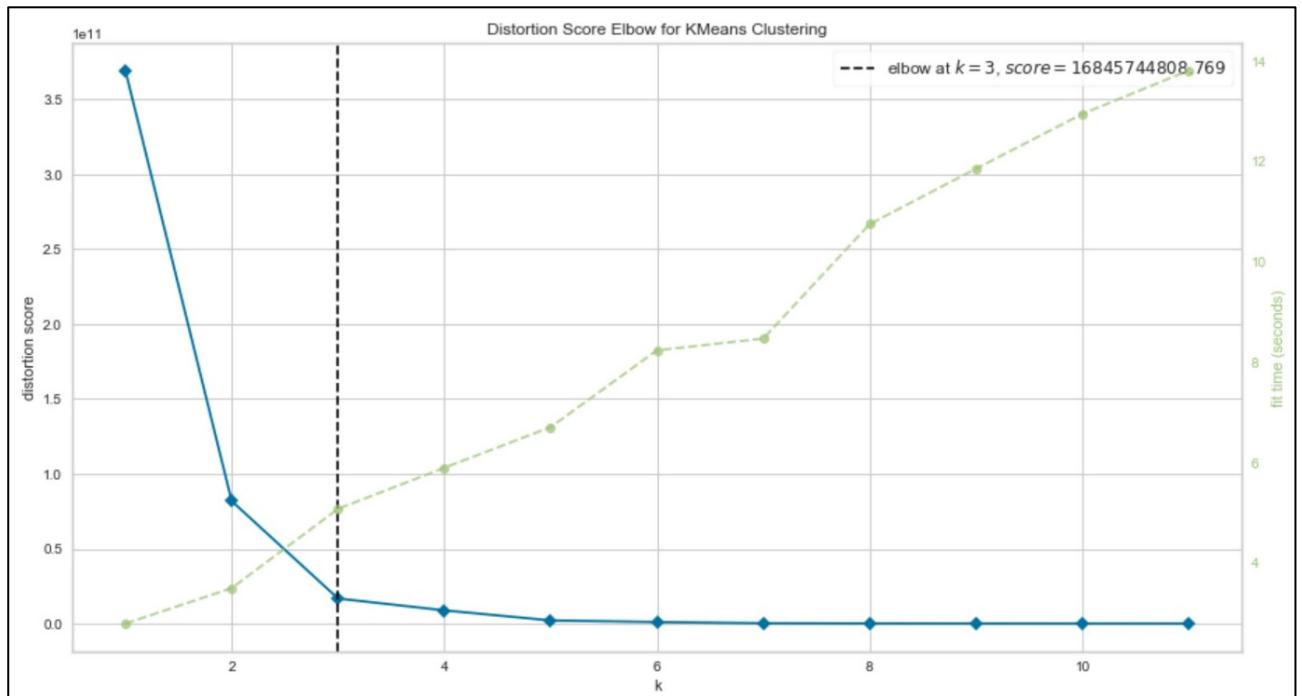


Figura 5.9: Grafico metodo Elbow per l'individuazione di K

Quindi il codice relativo all'implementazione con k=3 è il seguente:

```

km = KMeans(n_clusters = 3, init='k-means++', max_iter=300, n_init = 10, random_state = 0)
y_means = km.fit_predict(x)

plt.figure(figsize=(15,8))
plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s=100, c= 'yellow', label= 'LOW volume bottles')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s=100, c= 'pink', label= 'MEDIUM volume bottles')
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s=100, c= 'magenta', label= 'HIGH volume bottles')
# plt.scatter(x[y_means == 3, 0], x[y_means == 3, 1], s=100, c= 'cyan')
# plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s=100, c= 'orange')
# plt.scatter(x[y_means == 5, 0], x[y_means == 5, 1], s=100, c= 'red')
# plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s=100, c= 'red')

plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], s=50, c='blue', label= 'centroid')

plt.title("K MEANS", fontsize=20)
plt.xlabel("Volume delle bottiglie in CL")
plt.ylabel("Bottiglie vendute")
plt.legend()
plt.show()

```

Figura 5.10: Codice K-means

L'output che ne deriva è il seguente:

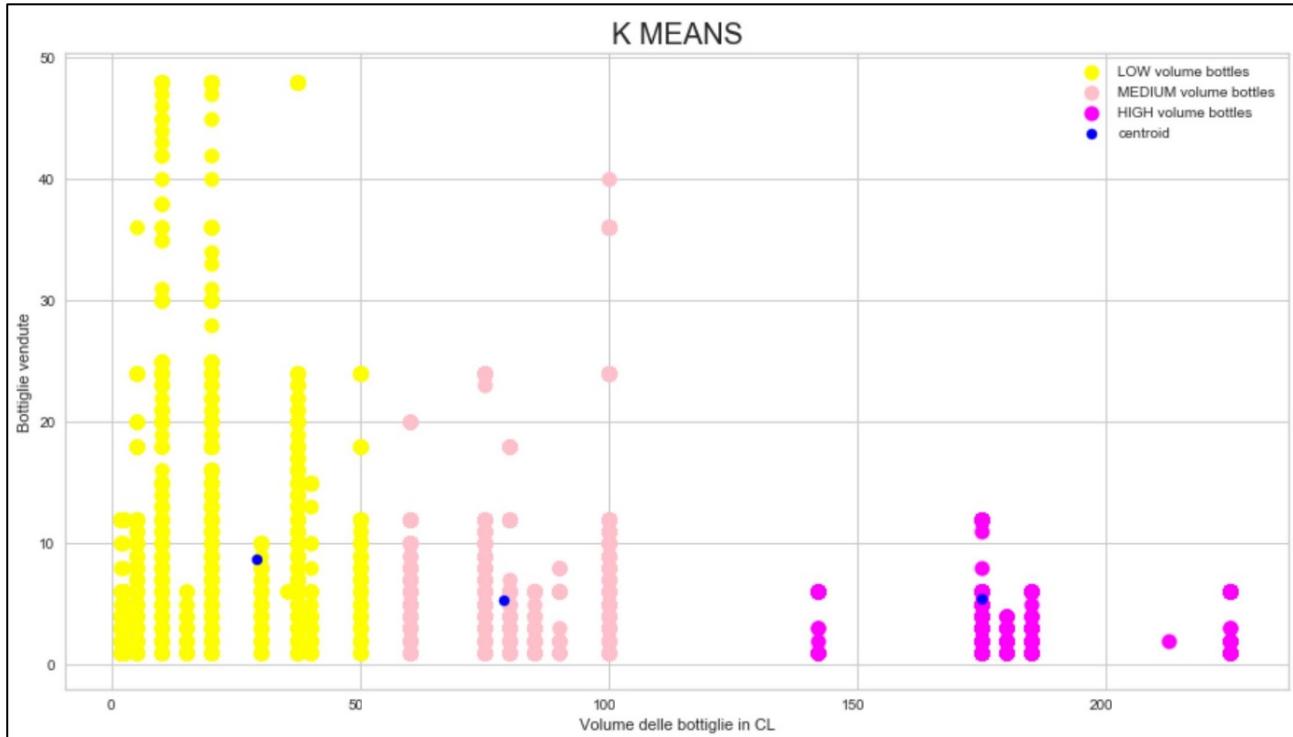


Figura 5.11: Grafico cluster ottenuti tramite K-means

La figura mostra i cluster realizzati settando k=3 e considerando come variabili le bottiglie vendute e il relativo volume. Questo risultato è molto interessante in quanto ci permette di comprendere la relazione esistente tra queste due variabili. In particolare, si nota come la maggior parte delle bottiglie vendute si raggruppano nel cluster giallo, che corrisponde a tutte quelle bottiglie il cui volume è minore di mezzo litro. Un numero medio di bottiglie vendute hanno un volume che varia da mezzo litro ad un litro (cluster rosa) e le bottiglie vendute che hanno un volume compreso tra un litro e mezzo e due litri (cluster magenta) risultano essere davvero poche. Quest'informazione, sicuramente di valore, potrà aggiungersi ad altre interessanti scoperte fatte nelle analisi nei capitoli successivi.

5.2.1 K-means “Black Velvet”

Sulla base del cluster realizzato si è ristretta l’analisi ad una solo categoria di alcolico che è quella che risulta più venduta, ossia “Black Velvet”.

Il codice seguente permette di filtrare il dataset ai soli dati di interesse e di definire i dati sui quali realizzare il cluster:

```
Only_Black_Velvet = df2019_nout['Item Description'] == 'Black Velvet'

df2019_Black_Velvet= df2019_nout[Only_Black_Velvet]

x=df2019_Black_Velvet[['Bottle Volume (cl)', 'Bottles Sold']].values

print(x.shape)

(42260, 2)
```

Figura 5.12: Filtraggio e istanziazione degli elementi da clusterizzare

Essendo quest’analisi molto simile alla precedente il metodo Elbow anche in questo caso suggerisce un valore di K pari a 3. Il cluster realizzato in definitiva è il seguente:

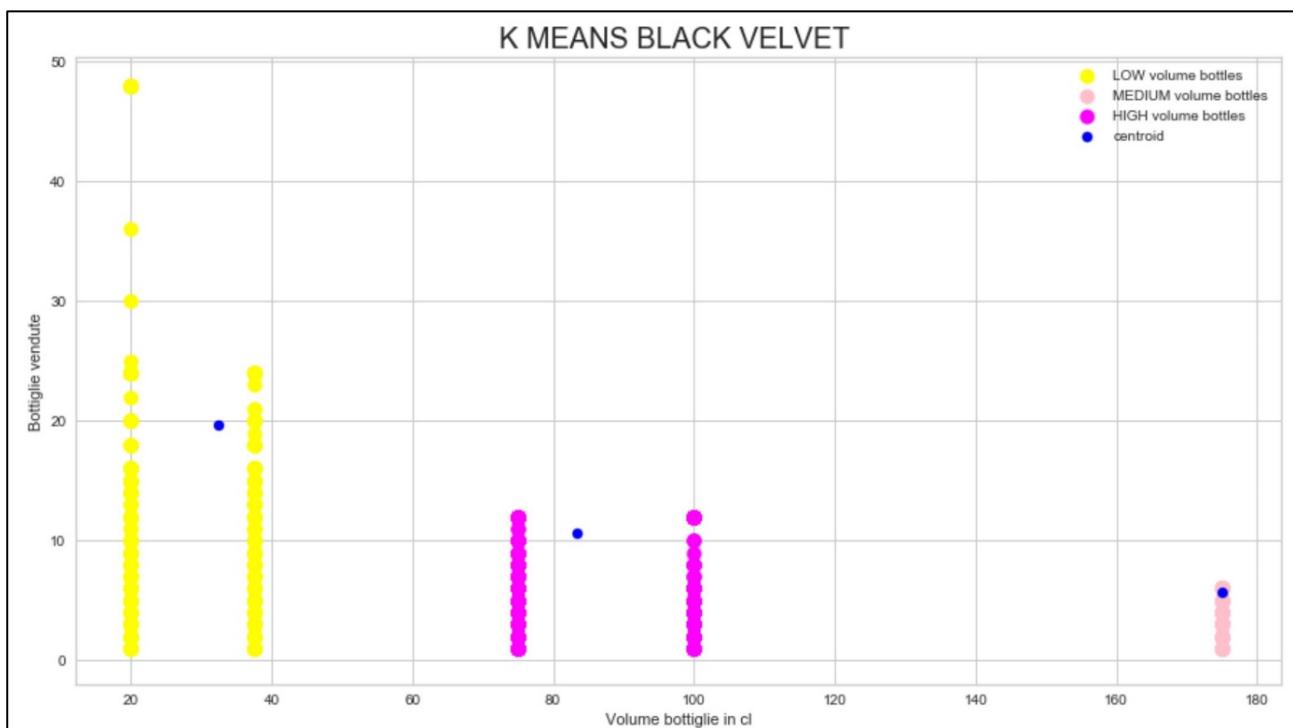


Figura 5.13: Grafico cluster ottenuti tramite K-means

Il risultato ottenuto ricalca quello precedente e in particolare si hanno molte bottiglie vendute solo per bottiglie il cui volume è sotto il mezzo litro, ed in particolare, per l’alcolico “Black Velvet”, quelle da 20cl. Un numero medio di bottiglie vendute riguardano tutte quelle bottiglie il cui volume è nell’intorno del litro (cluster color magenta) e pochissime bottiglie vendute sono quelle con capacità pari a quasi due litri.

Massimizzare quindi le bottiglie del cluster giallo, in particolare quelle il cui volume è 20cl, potrebbe potenzialmente portare ad un numero maggiore di bottiglie vendute.

6. Influenza dei parametri di posizione sulle vendite

L'obiettivo di questa sezione è individuare dei pattern nascosti all'interno dei dati ed in particolare individuare quali elementi o variabili influenzano le vendite. Nel dettaglio, vorremo poter stabilire un metodo oggettivo per consigliare in quali città sia intelligente e proficuo investire per l'apertura di un nuovo negozio di liquori. Tale obiettivo verrà conseguito addestrando diversi modelli di regressione, analizzati nel dettaglio in questa sezione.

Inizialmente definiamo una colonna per i profitti dello stato, chiamata "State profit per bottle":

```
df['State profit per bottle'] = df['State Bottle Retail'] - df['State Bottle Cost']
```

Figura 6.1: Codice creazione campo "State profit per bottle"

Successivamente si definisce un dataframe "zcode_table" che contiene le variabili di interesse:

```
zcode_table = df[['Store Number', 'Zip Code', 'Sale (Dollars)', 'State Bottle Retail',
                  'State profit per bottle', 'Bottles Sold', 'Volume Sold (Liters)',
                  ]]
zcode_table.head()
```

Figura 6.2: Codice creazione dataframe "zcode_table"

Il cui output è:

	Store Number	Zip Code	Sale (Dollars)	State Bottle Retail	State profit per bottle	Bottles Sold	Volume Sold (Liters)
0	4256	51250	12.99	12.99	4.33	6	4.50
1	4588	51103	98.64	4.11	1.37	1	0.37
3	2605	52402	48.72	48.72	16.24	3	2.25
4	4967	52726	10.68	10.68	3.56	1	0.60
5	4829	50314	148.68	12.39	4.13	1	0.75

Figura 6.3: Dataframe "zcode_table"

L'idea è che nei passaggi successivi bisogna creare un dataframe contenente le informazioni di interesse che si vuole mantenere nelle analisi.

In accordo all'obiettivo, otterremo un dataframe che contiene informazioni raggruppate per il campo "zip code", ossia il codice di avviamento postale. Questo ci permette di identificare appunto la città, in sintonia all'obiettivo prefissatoci. Inoltre, sappiamo che ogni città ha un proprio codice di avviamento postale ma potrebbe tenerne più di uno, se questa è molto grande.

6.1 Creazione dataframe “sales_by_zip”

Dunque, riprendendo il codice, viene creato un dataframe “sales_by_zip” che raggruppa il dataframe precedente per zip code mostrando anche le vendite, ordinate.

```
sales_by_zip = zcode_table.groupby('Zip Code')['Sale (Dollars)'].sum().to_frame().sort_values('Sale (Dollars)', ascending=0)  
# resetting the index  
sales_by_zip['Zip Code'] = sales_by_zip.index
```

Figura 6.4: Codice dataframe “sales_by_zip”

```
sales_by_zip.index = range(0, len(sales_by_zip))  
  
print(sales_by_zip.shape)  
sales_by_zip.head()  
  
(482, 2)
```

Figura 6.5: Reset index e stampa del dataframe

L’output relativo è:

	Sale (Dollars)	Zip Code
0	9.944311e+07	50314
1	9.353735e+07	50320
2	8.023740e+07	52402
3	7.103721e+07	52240
4	6.203393e+07	50266

Figura 6.6: Dataframe “sales_by_zip”

Da questo raggruppamento si può anche dedurre quali sono gli zip code che presentano vendite più alte. I primi tre sono gli Zip Code “50314”, “50320” e “52402”.

6.2 Creazione dataframe “volume_by_zip”

Successivamente viene creato un nuovo dataframe, raggruppando sempre per il campo Zip Code ma considerando questa volta il volume venduto.

```
volume_by_zip = zcode_table.groupby('Zip Code')['Volume Sold (Liters)'].sum().to_frame()
volume_by_zip['Zip Code'] = volume_by_zip.index
volume_by_zip.index = range(0, len(volume_by_zip))
volume_by_zip.head()
```

Figura 6.7: Codice creazione dataframe “volume_by_zip”

L’output relativo è:

	Volume Sold (Liters)	Zip Code
0	28736.61	50002
1	267717.28	50003
2	6604.65	50005
3	61988.17	50006
4	499.53	50008

Figura 6.8: Dataframe “volume_by_zip”

6.3 Creazione dataframe “bottleprofit_per_zip”

Analogamente viene creato un dataframe considerando questa volta il profitto per bottiglia e raggruppando sempre per il campo Zip Code:

```
bottleprofit_per_zip = zcode_table.groupby('Zip Code')['State profit per bottle'].sum().to_frame()
bottleprofit_per_zip['Zip Code'] = bottleprofit_per_zip.index
bottleprofit_per_zip.index = range(0, len(bottleprofit_per_zip))
bottleprofit_per_zip.head()
```

Figura 6.9: Codice creazione dataframe “bottleprofit_per_zip”

L’output relativo è:

	State profit per bottle	Zip Code
0	19388.56	50002
1	110359.09	50003
2	2846.46	50005
3	47808.26	50006
4	3710.10	50008

Figura 6.10: Dataframe “bottleprofit_per_zip”

6.4 Creazione dataframe “stores_by_zip”

Analogamente viene creato un nuovo dataframe raggruppando sempre per il campo Zip Code, calcolando il numero di negozi all'interno di ogni codice di avviamento postale. Inoltre, il numero di negozi per Zip Code è ordinato.

```
stores_by_zip = df.groupby('Zip Code')['Store Number'].nunique().to_frame().sort_values('Store Number', ascending=False)
stores_by_zip['Zip Code'] = stores_by_zip.index
stores_by_zip.index = range(0, len(stores_by_zip))
stores_by_zip.head()
```

Figura 6.11: Codice creazione dataframe “stores_by_zip”

L'output relativo è:

	Store Number	Zip Code
0	44	52402
1	40	52404
2	40	51501
3	35	52001
4	34	52240

Figura 6.12: Dataframe “stores_by_zip”

Quindi vediamo che i primi tre codici di avviamento postale con maggior numero di negozi di liquori, sono il “52402”, il “52404” e il “51501”.

6.5 Creazione dataframe “bottles_by_zip”

Analogamente viene creato un altro dataframe, questa volta considerano il numero di bottiglie vendute raggruppando sempre per il codice di avviamento postale, Zip Code.

```
bottles_by_zip = df.groupby('Zip Code')['Bottles Sold'].sum().to_frame().sort_values('Bottles Sold', ascending=False)
bottles_by_zip['Zip Code'] = bottles_by_zip.index
bottles_by_zip.index = range(0, len(stores_by_zip))
bottles_by_zip.head()
```

Figura 6.13: Codice creazione dataframe “bottles_by_zip”

L'output relativo è:

	Bottles Sold	Zip Code
0	6366175	50314
1	6127576	52402
2	5948940	50320
3	5293369	52240
4	4526042	51501

Figura 6.14: Dataframe “bottles_by_zip”

Si deduce che i primi tre Zip Code in cui si è venduto un numero maggiore di bottiglie, sono il “50314”, il “52402” e il “50320”.

6.5 Fusione dei dataframe e creazione del dataframe “zip_frame”

A questo punto, per una maggior coerenza con i raggruppamenti fatti, viene rinominata la colonna “Store Number” in “Number of Stores Per Zip”, in quanto si è raggruppato per Zip Code:

```
stores_by_zip.rename(columns={'Store Number' : 'Number of Stores Per Zip'}, inplace=True)  
stores_by_zip.head()
```

Figura 6.15: Rinominazione campo “Store Number”

L’output, quindi, sarà semplicemente il seguente:

	Number of Stores Per Zip	Zip Code
0	44	52402
1	40	52404
2	40	51501
3	35	52001
4	34	52240

Figura 6.16: Dataframe “stores_by_zip”

6.5.1 Merge dataframe “stores_by_zip” e “sales_by_zip”

A questo punto, partono delle operazioni di merge tra tutti i mini dataframe che sono stati creati, per avere un unico dataframe contenente le informazioni raggruppate per codice di avviamento postale.

Il primo merge avviene tra il dataframe “stores_by_zip” e “sales_by_zip”, andando a creare un nuovo dataframe denominato “zip_frame”. Esso sarà il dataframe finale che conterrà tutti i dataframe ottenuti precedentemente:

```
zip_frame = pd.merge(stores_by_zip, sales_by_zip, how='inner', on='Zip Code')  
print(zip_frame.shape)  
zip_frame.head()  
  
(482, 3)
```

Figura 6.17: Codice merge dataframe “stores_by_zip” e “sales_by_zip”

L’output, quindi, sarà semplicemente il seguente:

	Number of Stores Per Zip	Zip Code	Sale (Dollars)
0	44	52402	8.023740e+07
1	40	52404	3.700606e+07
2	40	51501	5.640994e+07
3	35	52001	3.162945e+07
4	34	52240	7.103721e+07

Figura 6.18: Dataframe “zip_frame” aggiornato

6.5.2 Merge dataframe “zip_frame” e “volume_by_zip”

Il secondo merge avviene tra il dataframe “zip_frame” e “volume_by_zip”:

```
zip_frame=pd.merge(zip_frame, volume_by_zip, how='inner', on='Zip Code')
print(zip_frame.shape)
zip_frame.head()

(482, 4)
```

Figura 6.19: Codice merge dataframe “zip_frame” e “volume_by_zip”

L’output, quindi, sarà semplicemente il seguente:

	Number of Stores Per Zip	Zip Code	Sale (Dollars)	Volume Sold (Liters)
0	44	52402	8.023740e+07	5.490957e+06
1	40	52404	3.700606e+07	2.594591e+06
2	40	51501	5.640994e+07	3.583675e+06
3	35	52001	3.162945e+07	2.080240e+06
4	34	52240	7.103721e+07	4.500929e+06

Figura 6.20: Dataframe “zip_frame” aggiornato

6.5.3 Merge dataframe “zip_frame” e “bottleprofit_per_zip”

Il terzo merge avviene tra il dataframe “zip_frame” e “bottleprofit_per_zip”:

```
zip_frame = pd.merge(zip_frame, bottleprofit_per_zip, how='inner', on='Zip Code')
print(zip_frame.shape)
zip_frame.head()

(482, 5)
```

Figura 6.21: Codice merge dataframe “zip_frame” e “bottleprofit_per_zip”

L’output, quindi, sarà semplicemente il seguente:

	Number of Stores Per Zip	Zip Code	Sale (Dollars)	Volume Sold (Liters)	State profit per bottle
0	44	52402	8.023740e+07	5.490957e+06	2.269103e+06
1	40	52404	3.700606e+07	2.594591e+06	1.433562e+06
2	40	51501	5.640994e+07	3.583675e+06	1.645319e+06
3	35	52001	3.162945e+07	2.080240e+06	1.263518e+06
4	34	52240	7.103721e+07	4.500929e+06	2.010534e+06

Figura 6.22: Dataframe “zip_frame” aggiornato

6.5.4 Merge dataframe “zip_frame” e “bottles_by_zip”

Il quarto merge avviene tra il dataframe “zip_frame” e “bottles_by_zip”:

```
zip_frame = pd.merge(zip_frame, bottles_by_zip, how='inner', on='Zip Code')
print(zip_frame.shape)
zip_frame.head()
(482, 6)
```

Figura 6.23: Codice merge dataframe “zip_frame” e “bottles_by_zip”

L’output, quindi, sarà semplicemente il seguente:

Number of Stores Per Zip	Zip Code	Sale (Dollars)	Volume Sold (Liters)	State profit per bottle	Bottles Sold
0	44	52402	8.023740e+07	5.490957e+06	2.269103e+06
1	40	52404	3.700606e+07	2.594591e+06	1.433562e+06
2	40	51501	5.640994e+07	3.583675e+06	1.645319e+06
3	35	52001	3.162945e+07	2.080240e+06	1.263518e+06
4	34	52240	7.103721e+07	4.500929e+06	2.010534e+06

Figura 6.24: Dataframe “zip_frame” aggiornato

6.5.5 Visualizzazione dataframe finale

Infine, attraverso un’operazione di Feature Engineering è possibile calcolare le vendite per negozio:

```
zip_frame['Sales Per Store'] = zip_frame['Sale (Dollars)'] / zip_frame['Number of Stores Per Zip']
zip_frame.head()
```

Figura 6.25: Codice creazione campo “Sale Per Store”

L’output, quindi, sarà semplicemente il seguente:

Number of Stores Per Zip	Zip Code	Sale (Dollars)	Volume Sold (Liters)	State profit per bottle	Bottles Sold	Sales Per Store
0	44	52402	8.023740e+07	5.490957e+06	2.269103e+06	1.823577e+06
1	40	52404	3.700606e+07	2.594591e+06	1.433562e+06	9.251516e+05
2	40	51501	5.640994e+07	3.583675e+06	1.645319e+06	1.410249e+06
3	35	52001	3.162945e+07	2.080240e+06	1.263518e+06	9.036984e+05
4	34	52240	7.103721e+07	4.500929e+06	2.010534e+06	2.089330e+06

Figura 6.26: Dataframe “zip_frame” aggiornato

A questo punto, creato il dataframe “zip_frame” raggruppando per Zip Code, risulta interessante individuare alcune relazioni statistiche o correlazioni nei nostri dati che verranno discusse nella sezione successiva.

6.6 Individuazione parametri e creazione dei modelli

Per individuare dunque statistiche e correlazioni nei nostri dati verrà creata una mappa di calore in grado di esprimere visivamente e a colpo d'occhio le correlazioni tra nostre le variabili:

```
zip_frame_fusion=zip_frame
sns.heatmap(zip_frame_fusion.corr(), annot = True, linewidths = 0.5)
```

Figura 6.27: Codice mappa di calore



Figura 6.28: Mappa di calore

Possiamo vedere, dalla mappa di calore appena costruita, come ci sia una forte correlazione delle bottiglie vendute, del volume venduto, delle vendite per negozio e il numero di negozi per zip code con le vendite.

Vediamo invece, che le vendite per negozio (Sales Per Store) non dipendano molto dal numero di negozi per codice postale, mentre le vendite totali sono molto correlate al numero di negozi raggruppati per zip code.

Il dataframe di interesse che abbiamo creato è quindi “zip_frame”:

zip_frame.head()							
	Number of Stores Per Zip	Zip Code	Sale (Dollars)	Volume Sold (Liters)	State profit per bottle	Bottles Sold	Sales Per Store
0	44	52402	8.023740e+07	5.490957e+06	2.269103e+06	6127576	1.823577e+06
1	40	52404	3.700606e+07	2.594591e+06	1.433562e+06	3345512	9.251516e+05
2	40	51501	5.640994e+07	3.583675e+06	1.645319e+06	4526042	1.410249e+06
3	35	52001	3.162945e+07	2.080240e+06	1.263518e+06	2578699	9.036984e+05
4	34	52240	7.103721e+07	4.500929e+06	2.010534e+06	5293369	2.089330e+06

Figura 6.29: Dataframe “zip_frame”

Si procede a rinominare il campo Sale (Dollars) in Total Sales (\$):

```
zip_frame.rename(columns={'Sale (Dollars)' : 'Total Sales ($)'}, inplace=True)
zip_frame.head()
```

Figura 6.30: Rinominazione campo “Sale (Dollars)”

Il cui output è:

Number of Stores Per Zip	Zip Code	Total Sales (\$)	Volume Sold (Liters)	State profit per bottle	Bottles Sold	Sales Per Store
0	44	52402	8.023740e+07	5.490957e+06	2.269103e+06	6127576
1	40	52404	3.700606e+07	2.594591e+06	1.433562e+06	3345512
2	40	51501	5.640994e+07	3.583675e+06	1.645319e+06	4526042
3	35	52001	3.162945e+07	2.080240e+06	1.263518e+06	2578699
4	34	52240	7.103721e+07	4.500929e+06	2.010534e+06	5293369

Figura 6.31: Dataframe “zip_frame” aggiornato

A questo punto è interessante poter ordinare in base a “Total Sales (\$)\”, ossia le vendite totali per zip code, in modo da comprendere quali sono i codici di avviamento postale che hanno alte vendite e quali basse vendite.

```
zip_frame = zip_frame.sort_values('Total Sales ($)', ascending=0)
zip_frame
```

Figura 6.32: Ordinamento delle vendite totali per zip code

Number of Stores Per Zip	Zip Code	Total Sales (\$)	Volume Sold (Liters)	State profit per bottle	Bottles Sold	Sales Per Store
11	50314	9.944311e+07	5.434182e+06	1.969332e+06	6366175	9.040283e+06
10	50320	9.353735e+07	5.487639e+06	1.323686e+06	5948940	9.353735e+06
44	52402	8.023740e+07	5.490957e+06	2.269103e+06	6127576	1.823577e+06
34	52240	7.103721e+07	4.500929e+06	2.010534e+06	5293369	2.089330e+06
23	50266	6.203393e+07	3.729647e+06	1.281872e+06	3434673	2.697127e+06
...
1	52362	1.577468e+04	8.822000e+02	5.275200e+02	1501	1.577468e+04
1	50471	1.503638e+04	1.018300e+03	7.619100e+02	1115	1.503638e+04
1	52134	1.497666e+04	1.165500e+03	8.712200e+02	1083	1.497666e+04
1	50174	7.864610e+03	5.846700e+02	8.370700e+02	747	7.864610e+03
1	51230	7.659100e+02	6.750000e+01	4.236000e+01	60	7.659100e+02

Figura 6.33: Dataframe “zip_frame” ordinato in base al campo Total Sales (\$)

La figura precedente mostra le prime ed ultime cinque righe del dataframe ordinate per vendite totali per zip code. Si può capire quindi quali sono i codici di avviamento postale con un alto numero di vendite, e dunque il “50314”, il “50320” e il “52402”.

Possiamo notare inoltre che i codici di avviamento postale “51230”, “50174” e “52134” sono tutti quei codici in cui le vendite sono piuttosto basse.

6.6.1 Creazione del modello

A questo punto, inizia la realizzazione di un modello di regressione il cui obiettivo è predire il valore delle vendite date le variabili presenti nel dataframe “zip_frame”.

Per prima cosa, andremo ad importare la libreria di interesse:

```
from sklearn.linear_model import LinearRegression
```

Figura 6.34: Importazione libreria “LinearRegression”

Successivamente si vanno a definire le variabili su cui addestrare il modello e la variabile da predire.

Nel nostro caso, i predittori saranno tutti quelli contenuti nella variabile X e la variabile da predire è contenuta in Y e corrisponde appunto alle vendite, come si vede dalla figura seguente:

```
X = zip_frame[['State profit per bottle', 'Number of Stores Per Zip', 'Volume Sold (Liters)', 'Bottles Sold', 'Sales Per Store']]  
y = zip_frame['Total Sales ($)']  
  
lm = LinearRegression()  
model = lm.fit(X,y)
```

Figura 6.35: Adattamento del modello ai dati

Addestriamo quindi il nostro modello e salviamo in una variabile “predictions” le predizioni del modello. Inoltre viene valutato il coefficiente di determinazione della previsione, detto R squared tramite “lm.score(X,y)”.

```
predictions = lm.predict(X)
```

```
lm.score(X,y)
```

```
0.993827231261511
```

Figura 6.36: “predictions” e “R squared”

Tale valore è quindi il valore di R squared del modello, che risulta molto alto.

6.6.2 Regressione Lineare Multipla

Il codice sotto riportato utilizza una regressione lineare multipla (MLR) per prevedere le vendite totali. I predittori, il numero di negozi, i volumi venduti, il profitto per bottiglia e le vendite per negozio sono sempre ordinati per codice postale.

```
X = zip_frame[['State profit per bottle', 'Number of Stores Per Zip', 'Volume Sold (Liters)', 'Bottles Sold', 'Sales Per Store']]
y = zip_frame['Total Sales ($)']
X = sm.add_constant(X) # adding y-intercept

model = sm.OLS(y, X).fit() ## sm.OLS(output, input)
predictions = model.predict(X)

model.summary()
```

Figura 6.37: Codice previsione vendite totali con MLR

In output si ha il summary del modello creato:

OLS Regression Results			
Dep. Variable:	Total Sales (\$)	R-squared:	0.994
Model:	OLS	Adj. R-squared:	0.994
Method:	Least Squares	F-statistic:	1.533e+04
Date:	Thu, 28 Jan 2021	Prob (F-statistic):	0.00
Time:	12:10:09	Log-Likelihood:	-7313.6
No. Observations:	482	AIC:	1.464e+04
Df Residuals:	476	BIC:	1.466e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	1.202e+05	7.36e+04	1.632	0.103	-2.45e+04	2.65e+05
State profit per bottle	-1.8138	0.568	-3.192	0.002	-2.930	-0.697
Number of Stores Per Zip	-1.444e+05	1.9e+04	-7.582	0.000	-1.82e+05	-1.07e+05
Volume Sold (Liters)	12.9651	0.432	30.038	0.000	12.117	13.813
Bottles Sold	3.9651	0.363	10.910	0.000	3.251	4.679
Sales Per Store	0.0248	0.079	0.315	0.753	-0.130	0.180
Omnibus:	272.235	Durbin-Watson:	1.611			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	14018.109			
Skew:	1.689	Prob(JB):	0.00			
Kurtosis:	29.203	Cond. No.	2.86e+06			

Figura 6.38: Summary del modello

Sulla base delle statistiche di sintesi del modello creato, i valori P di “State profit per bottle”, “Number of Stores Per Zip”, “Volume Sold (Liters)” e “Bottles Sold” sono tutti fattori statisticamente significativi per la previsione delle vendite totali.

La variabile “Sales per Store”, vendite per negozio, non risulta statisticamente significativa ma si è deciso di non rimuoverla dal modello in quanto il valore di “R squared” senza di essa risultava più basso.

Tra le prime conclusioni si può affermare che se il numero di bottiglie vendute aumenta di 1 bottiglia, a parità di altre condizioni, le vendite aumenteranno di 3,96 dollari. Se il volume di bottiglie venduto aumenta di 1 litro le vendite aumenteranno di 12,4082 dollari.

Passando a valutare l'errore del modello creato, si può graficare tramite uno scatter plot i valori predetti rispetto quelli veri, e dunque attraverso il seguente codice:

```
plt.scatter(predictions, y, s=30, c='r', marker='+', zorder=10);
plt.xlabel('Predicted Sales')
plt.ylabel("Actual Sales")
plt.show()
```

Figura 6.39: Codice scatter plot

L'output ottenuto è il seguente grafico:

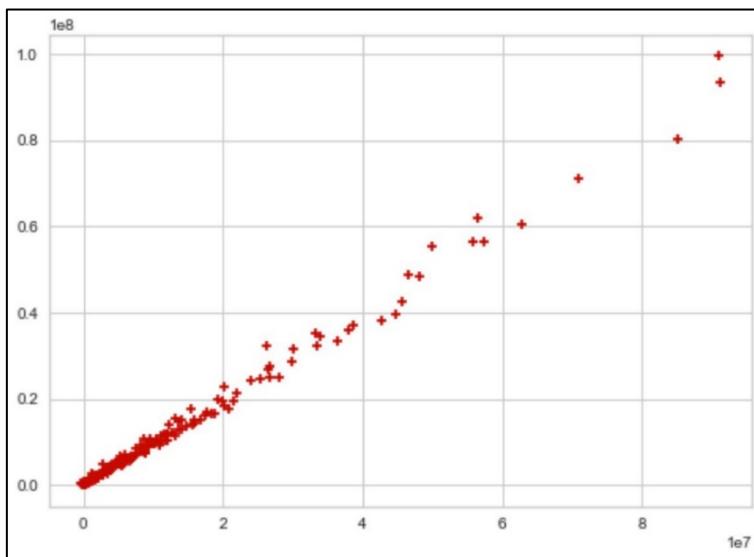


Figura 6.40: Grafico dei valori predetti

Il grafico si presenta quasi come una linea retta questo sta a significare una forte correlazione tra “Predicted Sales” e “Actual Sales”. In questo senso significa che quasi tutti i valori predetti corrispondono ai valori reali e questo è ovviamente buono.

L'errore invece valutato come MSE, e quindi come errore quadratico medio, è abbastanza elevato come mostrato in figura:

```
print("MSE:", model.mse_model)
MSE: 1.3736339730587468e+16
```

Figura 6.41: MSE - Errore quadratico medio

È opportuno ricorrere a delle tecniche per ridurre il più possibile tale errore al minimo.

Per farlo si importano una serie di librerie che si andranno poi ad utilizzare.

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics
from sklearn import linear_model
```

Figura 6.42: Importazione librerie per la riduzione del MSE

6.6.3 Validazione del modello - Holdout Validation

Utilizzando questo metodo il set di dati viene separato in due set, chiamati set di addestramento (o training set) e set di test (o test set). L'idea è addestrare il modello solamente sul training set e valutarlo su un insieme di dati che il modello non ha visto durante l'addestramento, il test set.

Definiamo le X e la y come nel caso precedente:

```
x = zip_frame[['State profit per bottle', 'Number of Stores Per Zip', 'Volume Sold (Liters)', 'Bottles Sold', 'Sales Per Store']]  
y = zip_frame['Total Sales ($)']
```

Figura 6.43: Definizione varibili X e y

Viene quindi diviso il dataset in una porzione di training e di test, in questo caso il 30% dei dati verrà usato per valutare il modello e il restante 70% dei dati per addestrare il modello:

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3)  
print(X_train.shape, y_train.shape)  
print(X_test.shape, y_test.shape)  
  
(337, 5) (337,)  
(145, 5) (145,)
```

Figura 6.44: Train/Test split

Dopo aver spartito i dati in una porzione di training e di test procediamo ad addestrare il modello sui dati di training, un modello di regressione lineare:

```
lm = linear_model.LinearRegression()  
  
model = lm.fit(X_train, y_train)  
predictions = lm.predict(X_test)
```

Figura 6.45: "fit" del modello e calcolo previsioni

È possibile graficare le predizioni effettuate dal modello e valutare il valore di R squared:

```
plt.scatter(y_test, predictions, marker='+', color='r')  
plt.xlabel("True Values")  
plt.ylabel("Predictions")  
print("Score:", model.score(X_test, y_test))
```

Figura 6.46: Codice scatter plot

In figura sono mostrate le predizioni:

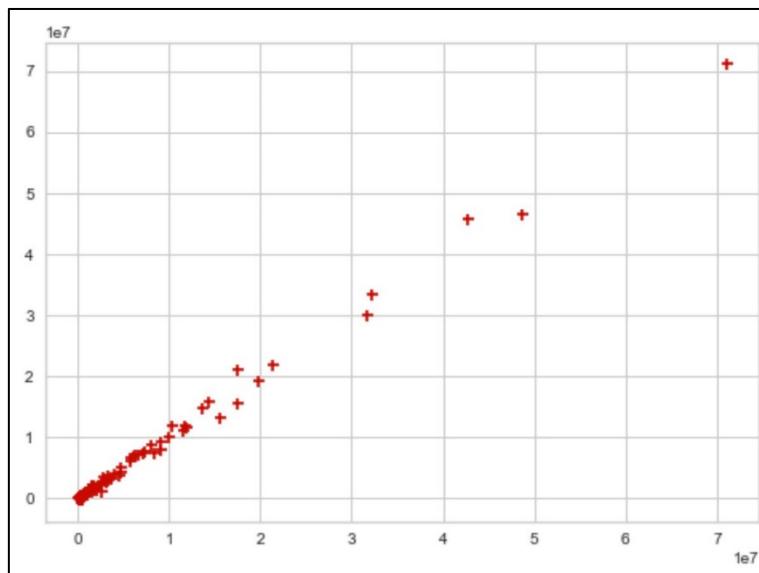


Figura 6.47: Grafico predizioni

E di seguito, il valore di R squared del modello usando “Train Test Split”:

Score: 0.9950996603907266

Figura 6.48: R squared

Lo score nell'utilizzare il metodo “Train Test Split” è pari a 0.99.

6.6.4 K- Cross Fold Validation

Utilizzando tale metodo il set di dati viene diviso in k sottoinsiemi. Ogni volta, uno dei k sottoinsiemi viene utilizzato come set di test e gli altri $k-1$ vengono utilizzati per formare l'insieme di addestramento. Quindi viene calcolato l'errore medio in tutte le k prove.

In questo caso utilizzeremo una K-Cross Fold Validation con $K = 10$, le X e la y sono definite come nei casi precedenti.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
scores = cross_val_score(model, X_train, y_train, cv=10)
print("Cross-validated scores:", scores)
print("Average: ", scores.mean())
```

Figura 6.49: K-Cross Fold Validation con $K = 10$

La misura delle prestazioni riportata dalla K-Cross Fold Validation è quindi la media dei valori calcolati, dunque abbiamo il seguente output:

```
Cross-validated scores: [0.99279499 0.99286315 0.97961033 0.99122803 0.99606222 0.99771722
0.99574048 0.98824487 0.98928461 0.99633277]
Average: 0.9919878687134543
```

Figura 6.50: Media dei valori calcolati

Considerando il valore medio si ha un valore di scores medio pari a 0.99.

Anche in questo caso è possibile graficare le predizioni del modello e valutare il valore di accuratezza. Dal punto di vista del codice avremo:

```
predictions = cross_val_predict(model, X, y, cv=10)
plt.scatter(y, predictions, marker='+')
plt.ylabel('True Sales')
plt.xlabel('Predicted Sales')
plt.title('10-Fold CV Predicted Sales Scatter')
```

Figura 6.51: Codice prenditions

Il relativo grafico mostra le predizioni rispetto al valore vero:

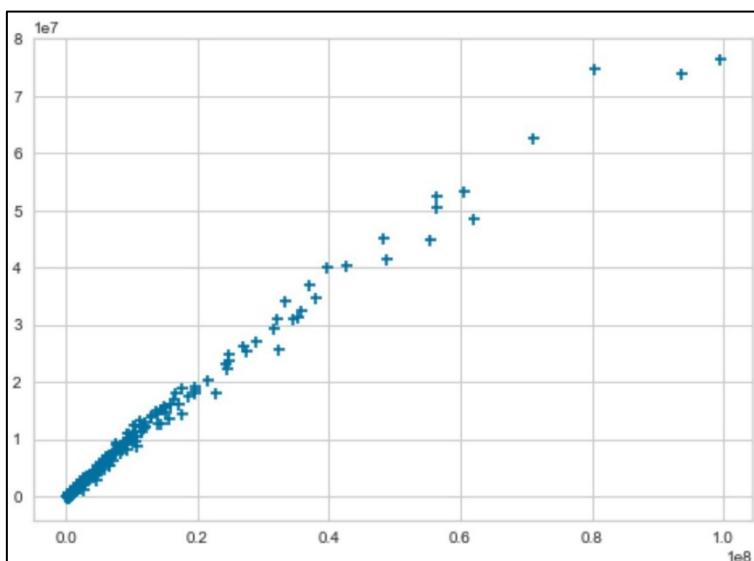


Figura 6.52: Grafico predizioni

L'accuratezza è possibile calcolarla attraverso il seguente codice:

```
accuracy = metrics.r2_score(y, predictions)
print("Cross-Predicted Accuracy:", accuracy)

Cross-Predicted Accuracy: 0.9750868861638873
```

Figura 6.53: Accuratezza di K-cross Fold Validation

Notiamo dunque che il valore di accuratezza del modello utilizzando una 10-Cross Fold Validation è pari a 0.97

6.6.5 Metodo Regressione Ridge

Con Regressione Ridge ci si riferisce ad un modello di regressione lineare i cui coefficienti non sono stimati dal metodo dei minimi quadrati (OLS), ma da un altro stimatore, chiamato Stimatore Ridge, che possiede bias ma ha una varianza inferiore rispetto allo stimatore OLS. Lo Stimatore Ridge riduce i coefficienti di regressione, in modo che le variabili, con un contributo minore al risultato, abbiano i loro coefficienti vicini allo zero.

Invece di farli a essere esattamente zero, li penalizziamo con un termine chiamato norma L2, costringendoli così a essere piccoli in modo continuo. In questo modo, diminuiamo la complessità del modello senza eliminare nessuna variabile. L'ammontare della penalità può essere messo a punto usando una costante chiamata lambda (λ). Per uno stimatore ridge, la selezione di un buon valore per λ è fondamentale.

Quando $\lambda = 0$, il termine di penalità non ha alcun effetto e la regressione ridge produrrà i coefficienti minimi quadrati classici. Tuttavia, quando λ aumenta all'infinito, l'impatto della penalità aumenta e i coefficienti di regressione si avvicinano allo zero.

Dal punto di vista del codice, si procede in prima battuta ad importare le librerie necessarie:

```
from sklearn.linear_model import Ridge, RidgeCV  
from sklearn import metrics
```

Figura 6.54: Importazione librerie

Il seguente codice realizza quindi una Ridge Regression sui nostri predittori ordinati per Zip Code. In particolare, si andranno a definire le X e la y e si utilizzerà il metodo Hold-out per lo split dei dati:

```
X= zip_frame[['State profit per bottle', 'Volume Sold (Liters)', 'Number of Stores Per Zip', 'Bottles Sold', 'Sales Per Store']]  
y= zip_frame['Total Sales ($)']  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Figura 6.55: Definizione X e y

Successivamente bisogna individuare il valore ottimale di alpha che si userà poi nel modello:

```
alpha_range = 10.*np.arange(-2, 3)  
ridgeregrcv = RidgeCV(alphas=alpha_range, normalize=True, scoring='neg_mean_squared_error')  
ridgeregrcv.fit(X_train, y_train)  
ridgeregrcv.alpha_  
  
0.01
```

Figura 6.56: Individuazione alfa migliore con RidgeCV

Notiamo che il valore ottimale di alpha è pari a 0.01. Dunque, si andrà ad eseguire la regressione con il valore di alpha scelto:

```
ridgereg = Ridge(alpha=0.01, normalize=True)  
ridgereg.fit(X_train, y_train)  
y_pred = ridgereg.predict(X_test)  
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Figura 6.57: "fit" del modello Ridge regression con alfa ottimale

E l'errore valutato come errore quadratico medio risulta pari a:

```
1008545.5296931164
```

Figura 6.58: MSE tramite Ridge regression

Passando ad esaminare i coefficienti si ha il seguente script con i relativi valori:

```
rc = ridgeregcv.coef_
print(rc)
[ 1.18229587e+00  1.01612460e+01 -1.82324166e+05  5.54660638e+00
 1.99213590e-01]
```

Figura 6.59: Esaminazione dei coefficienti

Mettendo in relazione il valore dei coefficienti con i corrispondenti predittori si ha:

```
print(X.columns, rc)
Index(['State profit per bottle', 'Volume Sold (Liters)',
       'Number of Stores Per Zip', 'Bottles Sold', 'Sales Per Store'],
      dtype='object') [ 1.18229587e+00  1.01612460e+01 -1.82324166e+05  5.54660638e+00
 1.99213590e-01]
```

Figura 6.60: Relazione tra coefficienti e predittori

State profit per bottle: $1.18229587e+00 = 1,18$

Volume sold: $1.01612460e+01 = 10.16$

Number of Stores per ZIP: $-1.82324166e+05 = -182324.16$

Bottles sold: $5.54660638e+00 = 5.546$

Sales per store: $1.99213590e-01 = 0.199$

In aggiunta, è possibile poter graficare i valori predetti. Per farlo, si utilizzerà il seguente script:

```
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.title('Ridge Regression Predictor of Future Sales', fontsize='x-large')
```

Figura 6.61: Scatter plot valori predetti

Il relativo grafico mostra le predizioni rispetto al valore vero:

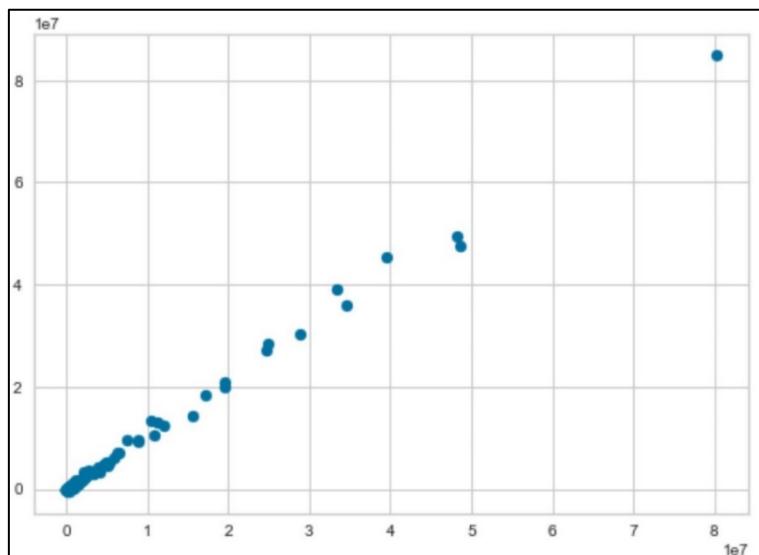


Figura 6.62: Grafico valori predetti

6.6.6 Lasso Regression Model

Lasso è l'acronimo di Least Absolute Shrinkage and Selection Operator (Operatore di selezione e ritiro assoluto minimo). Permette di ridurre i coefficienti di regressione verso lo zero penalizzando il modello di regressione con un termine di penalità chiamato norma L1, che è la somma dei coefficienti assoluti.

La penalità ha l'effetto di forzare alcune delle stime dei coefficienti, con un contributo minore al modello, a essere esattamente uguale a zero.

Per prima cosa si procede ad importare la libreria necessaria:

```
from sklearn.linear_model import Lasso, LassoCV
```

Figura 6.63: Importazione libreria

Similmente alle situazioni precedenti definiamo le X e la y ed il relativo metodo di split:

```
X= zip_frame[['State profit per bottle', 'Number of Stores Per Zip', 'Volume Sold (Liters)', 'Bottles Sold', 'Sales Per Store']]  
y= zip_frame['Total Sales ($)']  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Figura 6.64: Definizione X e y

In prima battuta sceglieremo il parametro di regolazione alfa ottimale, quindi regolarizzeremo con una regressione Lasso, come mostrato in figura:

```
lassoregcv = LassoCV(n_alphas=10, normalize=True, random_state=1)  
lassoregcv.fit(X_train, y_train)  
lassoregcv.alpha_  
  
1552.3553768130378
```

Figura 6.65: Selezione del parametro migliore alfa con LassoCV

I coefficienti beta rimanenti alla fine della regolarizzazione sono i più importanti e sono i seguenti:

```
lc = lassoregcv.coef_  
print(X.columns, lc)  
  
Index(['State profit per bottle', 'Number of Stores Per Zip',  
       'Volume Sold (Liters)', 'Bottles Sold', 'Sales Per Store'],  
      dtype='object') [-9.29517385e-01 -1.35807071e+05  1.41089382e+01  2.25956760e+00  
 4.30773059e-01]
```

Figura 6.66: Esaminazione dei coefficienti

State profit per bottle: -9.29517385e-01 = 1,18

Volume sold: 1.41089382e+01 = 14.16

Number of Stores per ZIP: -1.35807071e+05 = -135807.07

Bottles sold: 2.25956760e+00 = 2.25

Sales per store: 4.30773059e-01 = 0.43

Infine, quindi calcoliamo il valore dell'errore quadratico medio:

```
y_pred = lassoregcv.predict(X_test)
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Figura 6.67: Predizioni

1169097.77800607

Figura 6.68: MSE tramite Lasso regression

Volendo graficare i valori veri rispetto quelli predetti, otteniamo il seguente grafico:

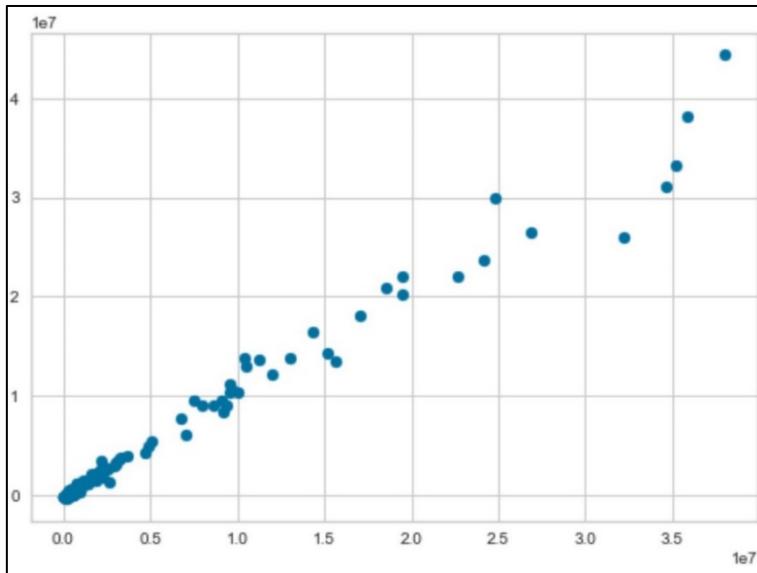


Figura 6.69: Grafico valori predetti

6.6.7 Conclusioni

Sulla base dei predittori e dei vari modelli che sono stati formati, si può concludere che le bottiglie vendute, il numero di punti vendita per codice postale, il volume venduto e la media delle vendite per punto vendita in ogni codice postale sono fortemente correlati alle vendite totali.

Tutti e quattro questi predittori sono stati presi in considerazione per prevedere le vendite totali.

In conclusione, tra i metodi presentati, la regolarizzazione Ridge Regression è il modello migliore considerano i predittori di interesse. Dunque, risulta il modello più adatto per prevedere le vendite totali per un dato codice postale. Il valore dell'errore con questo metodo diminuisce notevolmente rispetto l'errore iniziale.

A questo punto si possono analizzare nel dettaglio i coefficienti ottenuti con tale metodo:

State profit per bottle: 1.18229587e+00 = 1,18

Volume sold: 1.01612460e+01 = 10.16

Number of Stores per ZIP: -1.82324166e+05 = -182324.16

Bottles sold: 5.54660638e+00 = 5.546

Sales per store: 1.99213590e-01 = 0.199

In primis, questi coefficienti rappresentano la forza e la direzione dell'influenza che il preditore ha sulle vendite totali. Come dimostrato, se il numero di negozi in ogni codice di avviamento postale aumenta di una unità, le Vendite Totali per ogni negozio in quel codice di avviamento postale diminuiscono enormemente. In altre parole, man mano che un numero maggiore di concorrenti (negozi di liquori) si aggiunge al campo, i profitti per ogni concorrente esistente diminuiscono. Sulla base di questi predittori e coefficienti, quattro codici di avviamento postale sono stati ristretti come alcune delle potenziali migliori città (posizioni) per aprire un futuro negozio di liquori.

I codici di avviamento postale scelti sono: 52314, 52401, 50312 e 52403. Rappresentano codici di avviamento postale dove non ci sono molti negozi e quindi con pochi concorrenti in campo e le bottiglie vendute e il volume venduto è abbastanza alto, in accordo ai coefficienti trovati. Tra gli Zip Code scelti si analizzerà in dettaglio quali sono le categorie di alcolico più vendute.

Ogni output seguente mostra la lista delle categorie degli alcolici vendute in base al codice postale scelto. Le categorie più in cima sono quelle vendute in maggior quantità, le categorie in fondo alla lista quelle vendute in minor quantità.

- **Zip code 52314:**

```
primo_zip=df['Zip Code'] == '52314'  
zip1=df[primo_zip]  
zip1['Category Name'].value_counts()
```

CANADIAN WHISKIES	6511
STRAIGHT BOURBON WHISKIES	5158
AMERICAN VODKAS	4949
VODKA 80 PROOF	4543
TEQUILA	4386
...	
AMERICAN VODKA	9
AMARETTO - IMPORTED	3
SCHNAPPS - IMPORTED	2
IMPORTED DISTILLED SPIRITS SPECIALTY	1
COCKTAILS / RTD	1

Figura 6.70: Classifica categorie degli alcolici vendute nello Zip code 52314

- Zip code 52401:

```
secondo_zip=df['Zip Code'] == '52401'
zip2=df[secondo_zip]
zip2['Category Name'].value_counts()
```

STRAIGHT BOURBON WHISKIES	5457
CANADIAN WHISKIES	5055
AMERICAN VODKAS	4526
VODKA 80 PROOF	3515
IMPORTED SCHNAPPS	2961
	...
DELISTED ITEMS	3
PEACH BRANDIES	3
HIGH PROOF BEER - AMERICAN	2
IMPORTED GINS	2
OTHER PROOF VODKA	1

Figura 6.71: Classifica categorie degli alcolici vendute nello Zip code 52401

- Zip code 50312:

```
terzo_zip=df['Zip Code'] == '50312'
zip3=df[terzo_zip]
zip3['Category Name'].value_counts()
```

AMERICAN VODKAS	8520
STRAIGHT BOURBON WHISKIES	6120
CANADIAN WHISKIES	5132
VODKA 80 PROOF	4359
SCOTCH WHISKIES	3441
	...
AMERICAN CORDIALS & LIQUEURS	4
LOW PROOF VODKA	2
TROPICAL FRUIT SCHNAPPS	1
AMERICAN VODKA	1
STRAWBERRY SCHNAPPS	1

Figura 6.72: Classifica categorie degli alcolici vendute nello Zip code 50312

- Zip code 52403:

```
quarto_zip=df['Zip Code'] == '52403'
zip4=df[quarto_zip]
zip4['Category Name'].value_counts()
```

VODKA 80 PROOF	7633
AMERICAN VODKAS	7521
CANADIAN WHISKIES	7344
STRAIGHT BOURBON WHISKIES	6846
SPICED RUM	4326
	...
AMERICAN DISTILLED SPIRITS SPECIALTY	3
JAPANESE WHISKY	2
IOWA DISTILLERY WHISKIES	1
DELISTED ITEMS	1
SPEARMINT SCHNAPPS	1

Figura 6.73: Classifica categorie degli alcolici vendute nello Zip code 52403

Si nota che per tutti e quattro gli zip code scelti le categorie di alcolico più vendute ricadono sempre sulle stesse tipologie di Vodka e Whiskies, in particolare “American Vodkas”, “Canadian Whiskies” e “Straight Bourbon Whiskies”.

Allo stesso modo si nota che, alcolici come lo “Schnapps”, alcolico che può assumere diverse forme tra cui distillati di frutta, liquori alle erbe, infusi e "liquori aromatizzati", è sempre in fondo alla lista e quindi risulta un alcolico che si vende poco.

7. Modello di regressione per predizione Contea

L'obiettivo di questa analisi è determinare la miglior locazione geografica in termini di contea all'interno dello stato dell'Iowa dove poter aprire un nuovo negozio di liquori. L'analisi proposta cerca quindi di determinare la contea attraverso la quale si possono ottenere i maggiori profitti. Per farlo si utilizzerà un modello di regressione. Inoltre, il secondo obiettivo è determinare quali fattori influenzano maggiormente le vendite relative alla contea vincitrice.

I dati sappiamo che contengono informazioni sulle le vendite di liquori dell'Iowa in date prestabilite dal 2010 a fine 2019. Per effettuare questa analisi abbiamo ristretto l'intervallo temporale al solo anno 2019, sia per focalizzare l'analisi ad un anno recente, sia per evitare di incorrere in problemi di allocazione di memoria, a causa della numerosità dei dati che dovrà processare il nostro modello.

```
df = raw_data  
  
raw_data_19 = raw_data[raw_data['year'] == 2019]
```

Figura 7.1: Restrizione dati anno 2019

Visto che le contee sono le protagoniste di questa analisi con lo script successivo andremo a valutare le contee e il relativo numero di vendite:

```
pop_counties = raw_data_19['County'].value_counts()  
  
POLK           383948  
LINN           178470  
SCOTT          126225  
BLACK HAWK     125434  
JOHNSON        114391  
...  
RINGGOLD        1687  
WAYNE          1457  
ADAMS           1114  
LUCAS            646  
FREMONT         411
```

Figura 7.2: Contee e numero di vendite

Si andranno a valutare le prime cinquanta contee:

```
pop_counties.head(50).plot(kind='barh', width=0.4, figsize=(2,13));
```

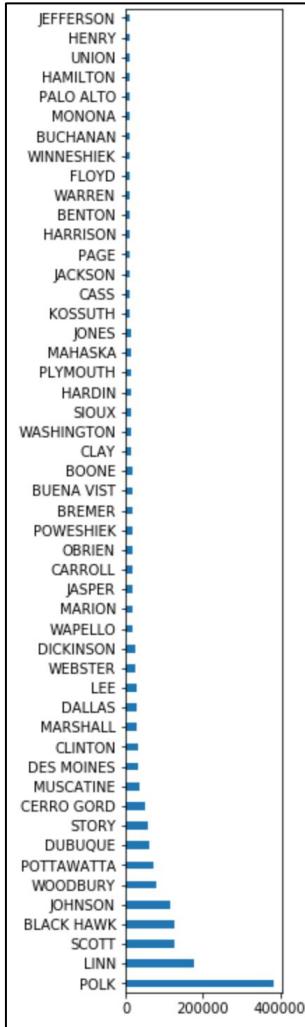


Figura 7.3: Visualizzazione prime 50 contee

Dopo avere effettuato le precedenti operazioni e avendo preso come presupposto che i dati di locazione siano corretti si è passati a quello che di solito in gergo viene detto “minare i dati”. In particolare, sotto questa sezione si è iniziato costruendo degli opportuni dataframe in modo da passarli successivamente in modo corretto al modello che verrà realizzato. Tramite delle variabili “dummy” abbiamo categorizzato le diverse contee, quindi l’effetto è avere un dataframe contenente nelle righe e nelle colonne i nomi delle contee. Si creano a questo punto delle celle contenti zeri ed uni. L’uno (1) in una certa cella sta a identificare la particolare contea (individuata dalla specifica riga e colonna), viceversa per lo zero (0).

```
dummy_df = pd.get_dummies(raw_data_19["County"])
dummy_df.head()
```

	ADAIR	ADAMS	ALLAMAKEE	APPANOOSE	AUDUBON	BENTON	BLACK HAWK	BOONE	BREMER	BUCHANAN	...	WAPELLO	WARREN	WASHINGTON
886836	0	0	0	0	0	0	0	0	0	0	...	0	0	0
896194	0	0	0	0	0	0	0	0	0	0	...	0	0	0
903678	0	0	0	0	0	0	0	0	0	0	...	0	0	0
906924	0	0	0	0	0	0	0	0	0	0	...	0	0	0
915910	0	0	0	0	0	0	0	0	0	0	...	0	0	0

Figura 7.4: Categorizzazione delle contee

Di seguito, andiamo a crearcia un altro dataframe, chiamato info_table, contenente le variabili di interesse da utilizzare nel modello. Queste sono “State bottle retail”, “Bottles Sold” e “Sale (Dollars)” che indicano rispettivamente il prezzo della bottiglia al dettaglio, le bottiglie vendute e le vendite, espressi in dollari.

```
info_table= raw_data_19[['State Bottle Retail', 'Bottles Sold', 'Sale (Dollars)']]
```

Figura 7.5: Codice info_table

A questo punto ci salviamo in un nuovo dataframe (location_df) l'unione dei due precedenti dataframe.

```
location_df = pd.concat([info_table, dummy_df], axis=1)
location_df.head()
```

	State Bottle Retail	Bottles Sold	Sale (Dollars)	ADAIR	ADAMS	ALLAMAKEE	APPANOOSE	AUDUBON	BENTON	BLACK HAWK	...	WAPELLO	WARREN	WASHINGTON
886836	11.51	2	23.02	0	0	0	0	0	0	0	...	0	0	0
896194	23.61	1	23.61	0	0	0	0	0	0	0	...	0	0	0
903678	12.38	4	49.52	0	0	0	0	0	0	0	...	0	0	0
906924	15.50	3	46.50	0	0	0	0	0	0	0	...	0	0	0
915910	10.38	12	124.56	0	0	0	0	0	0	0	...	0	0	0

Figura 7.6: Dataframe location_df

È importante individuare eventuali relazioni statistiche, correlazioni o altre proprietà rilevanti dei nostri dati. Per questo creiamo una mappa di calore per osservare graficamente queste eventuali correlazioni. In particolare, vogliamo capire come siano correlate le variabili contenute all'interno del dataframe “info_table”.

La mappa di calore realizzata è mostrata in figura:



Figura 7.7: Mappa di calore per info_table

Con la mappa di calore appena creata, possiamo vedere una forte correlazione positiva tra il numero di bottiglie vendute (bottles sold) e la variabile di interesse vendite (Sale). Ciò significa che dovrebbe essere un buon predittore delle vendite. Il prezzo al dettaglio (State Bottle Retail), che è fondamentalmente il prezzo per bottiglia, ha a malapena alcuna correlazione con le vendite. In tutti casi, manterremo tutte e tre le variabili nel modello di regressione, insieme alle numerose variabili dummy relative alle contee.

È arrivato il momento di costruire il modello di regressione.

Innanzitutto, dopo aver importato la libreria di interesse, è stato creato un primo modello considerando esclusivamente le 3 variabili sopra citate (“State bottle retail”, “Bottles Sold” e “Sale (Dollars)”).

```

import statsmodels.api as sm

raw_shor=raw_data_19[['State Bottle Retail', 'Bottles Sold']]
raw_shor = sm.add_constant(raw_shor)
raw_shor.shape

(2158261, 3)

y = location_df['Sale (Dollars)']
y.shape

(2158261,)

model = sm.OLS(y, raw_shor).fit()
predictions = model.predict(raw_shor)
model.summary()

```

Figura 7.8: Creazione modello

Il modello è dunque una regressione lineare eseguita attraverso Statsmodels di Python. Abbiamo “fittato”, o meglio addestrato il nostro modello sulle X, corrispondenti alle variabili “Prezzo della bottiglia” (State Bottle Retail) e “Bottiglie vendute” (Bottles Sold).

La Y, ossia il valore da predire, corrisponde invece alle “vendite” (“Sales (Dollars)”). Vediamo dal summary ottenuto che sia il numero di bottiglie vendute, sia il prezzo delle bottiglie sono statisticamente significativi nell’influenzare le vendite.

Dep. Variable:	Sale (Dollars)	R-squared:	0.673			
Model:	OLS	Adj. R-squared:	0.673			
Method:	Least Squares	F-statistic:	2.220e+06			
Date:	Thu, 28 Jan 2021	Prob (F-statistic):	0.00			
Time:	16:24:32	Log-Likelihood:	-1.5259e+07			
No. Observations:	2158261	AIC:	3.052e+07			
Df Residuals:	2158258	BIC:	3.052e+07			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-98.8825	0.315	-313.630	0.000	-99.500	-98.265
State Bottle Retail	6.6220	0.015	439.281	0.000	6.592	6.652
Bottles Sold	12.6463	0.006	2086.326	0.000	12.634	12.658
Omnibus:	3728111.778		Durbin-Watson:		1.996	
Prob(Omnibus):	0.000		Jarque-Bera (JB):		134476975243.687	
Skew:	10.766		Prob(JB):		0.00	
Kurtosis:	1225.673		Cond. No.		56.0	

Figura 7.9: Summary del modello

In pratica, possiamo affermare che se il prezzo per bottiglia aumenta di \$1, a parità di altre condizioni, le vendite aumenteranno di 6,62 dollari. Se il numero di bottiglie vendute aumenta di 1 bottiglia, a parità di altre condizioni, le vendite aumenteranno di 12,64 dollari.

Prima di creare il secondo modello, abbiamo eliminato la contea di Polk in quanto presenta un gran numero di osservazioni rispetto a tutte le altre contee, spiegato sicuramente dal fatto che è un centro urbano molto importante e che potrebbe indurre al problema della multicollinearità (in questa situazione, le stime dei coefficienti della regressione possono cambiare in modo irregolare in risposta a piccoli cambiamenti nel modello o nei dati), detta anche in gergo “dummy variable trap”. Essendo la contea di Polk un centro maggiore ci sarà sicuramente un numero maggiore di vendite.

```
dummy = dummy_df.copy(deep=True)
dummy = dummy.drop('POLK', axis=1)
dummy = sm.add_constant(dummy)
dummy.shape

(2158261, 99)
```

Figura 7.10: Eliminazione contea Polk

A questo punto è stato creato il secondo modello per verificare se da sole, le variabili dummy contenenti le contee fossero in grado predire le vendite:

```
models_dm = sm.OLS(y, dummy).fit()
predictions = models_dm.predict(dummy)
models_dm.summary()
```

Figura 7.11: “fit” del modello sulle variabili dummy

La regressione mostra infatti che i parametri di localizzazione da soli non riescono a predire le vendite. Come si vede dal summary si ha un valore di R-squared molto basso e quindi il modello con sole queste variabili non riesce a spiegare le vendite.

OLS Regression Results						
Dep. Variable:	Sale (Dollars)	R-squared:	0.005			
Model:	OLS	Adj. R-squared:	0.005			
Method:	Least Squares	F-statistic:	117.8			
Date:	Thu, 28 Jan 2021	Prob (F-statistic):	0.00			
Time:	16:26:49	Log-Likelihood:	-1.6459e+07			
No. Observations:	2158261	AIC:	3.292e+07			
Df Residuals:	2158162	BIC:	3.292e+07			
Df Model:	98					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	194.9713	0.801	243.408	0.000	193.401	196.541
ADAIR	-102.5230	7.377	-13.898	0.000	-116.981	-88.065
ADAMS	-118.8901	14.892	-7.983	0.000	-148.078	-89.702
ALLAMAKEE	-99.3401	6.528	-15.219	0.000	-112.134	-86.546
APPANOOSE	-58.4280	5.636	-10.367	0.000	-69.474	-47.381
AUDUBON	-127.8492	9.280	-13.777	0.000	-146.038	-109.661
BENTON	-105.5883	4.844	-21.799	0.000	-115.082	-96.095
BLACK HAWK	-50.9853	1.614	-31.586	0.000	-54.149	-47.822
BOONE	-65.2536	3.947	-16.534	0.000	-72.989	-57.518
BREMER	-74.9431	3.893	-19.251	0.000	-82.573	-67.313
BUCHANAN	-79.5769	4.923	-16.164	0.000	-89.226	-69.928
BUENA VIST	-78.7435	3.928	-20.046	0.000	-86.443	-71.044

Figura 7.12: Porzione del summary con le sole contee

Quindi è stato realizzato un terzo modello di regressione, addestrato sia sulle variabili dummy (le contee) sia sulle variabili usate nel primo modello (“State bottle retail” e “Bottles Sold”).

```
X = location_df.copy(deep=True)
X = X.drop('Sale (Dollars)', axis=1)
X = X.drop('POLK', axis=1)
X = sm.add_constant(X)
X.shape

(2158261, 101)

model = sm.OLS(y, X).fit()
predictions = model.predict(X)
model.summary()
```

Figura 7.13: “fit” del modello

OLS Regression Results						
Dep. Variable:	Sale (Dollars)		R-squared:	0.673		
Model:	OLS		Adj. R-squared:	0.673		
Method:	Least Squares		F-statistic:	4.447e+04		
Date:	Thu, 28 Jan 2021		Prob (F-statistic):	0.00		
Time:	16:31:39		Log-Likelihood:	-1.5258e+07		
No. Observations:	2158261		AIC:	3.052e+07		
Df Residuals:	2158160		BIC:	3.052e+07		
Df Model:	100					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-94.4650	0.530	-178.145	0.000	-95.504	-93.426
State	Bottle Retail	6.6008	0.015	437.289	0.000	6.571
	Bottles Sold	12.6438	0.006	2080.184	0.000	12.632
	ADAIR	-1.9736	4.228	-0.467	0.641	-10.261
	ADAMS	-5.6538	8.536	-0.662	0.508	-22.383
	ALLAMAKEE	0.5431	3.742	0.145	0.885	-6.790
	APPANOOSE	0.0322	3.230	0.010	0.992	-6.299
	AUDUBON	-3.0187	5.319	-0.568	0.570	-13.444
	BENTON	-6.7926	2.777	-2.446	0.014	-12.235
	BLACK HAWK	-14.2832	0.926	-15.432	0.000	-16.097
	BOONE	-5.6279	2.262	-2.488	0.013	-10.062
	BREMER	-0.5867	2.232	-0.263	0.793	-4.960
	BUCHANAN	1.3507	2.822	0.479	0.632	-4.181
	BUENA VIST	-5.0820	2.252	-2.257	0.024	-9.495
	BUTLER	-2.7690	4.364	-0.635	0.526	-11.322
	CALHOUN	-2.9214	5.110	-0.572	0.568	-12.936
	CARROLL	16.6545	2.208	7.542	0.000	12.326
	CASS	2.6686	2.634	1.013	0.311	-2.494
	CEDAR	0.5674	2.985	0.190	0.849	-5.283
	CERRO GORD	-9.3704	1.339	-6.997	0.000	-11.995
						...
	CHEROKEE	-16.0073	3.320	-4.822	0.000	-22.514
	CHICKASAW	14.9764	4.838	3.096	0.002	5.494
	CLARKE	2.7030	3.607	0.749	0.454	-4.366
	CLAY	0.9739	2.303	0.423	0.672	-3.541
	CLAYTON	-6.7894	3.013	-2.253	0.024	-12.695
	CLINTON	-7.5094	1.678	-4.476	0.000	-10.798
	CRAWFORD	11.1156	3.182	3.493	0.000	4.879
	DALLAS	49.9897	1.749	28.585	0.000	46.562
	DAVIS	-0.5106	6.812	-0.075	0.940	-13.862
	DECATUR	-8.2090	6.384	-1.286	0.198	-20.722
	DELAWARE	1.3399	3.678	0.364	0.716	-5.870
	DES MOINES	-4.9930	1.612	-3.098	0.002	-8.152

Figura 7.14: Summary del modello ottenuto

Vediamo dal summary di output, che sia le “bottiglie vendute” che il “prezzo per bottiglia” rimangono statisticamente significative quando aggiungiamo i parametri di localizzazione al nostro modello. Questo modello è tutto relativo a Polk County perché lo abbiamo escluso nella nostra regressione. La contea che ha il coefficiente più alto è la contea di Dallas. Anche la variabile per la contea di Dallas è statisticamente

significativa. Sulla base di questa regressione, potremmo dire che l'apertura di un negozio di liquori nella contea di Dallas potrebbe potenzialmente portare a più vendite rispetto all'apertura di un negozio di liquori in altre contee.

Nell'immagine seguente tramite un scatter plot vengono rappresentati i valori predetti, e quindi valutato l'errore quadratico medio (MSE).

```
plt.scatter(predictions, y, s=30, c='r', marker='+', zorder=10)
plt.xlabel("Predicted Values from Model")
plt.ylabel("Actual Values of Sales")
plt.plot(predictions, np.poly1d(np.polyfit(predictions, y, 1))(predictions))
print("MSE:", model.mse_model)
```

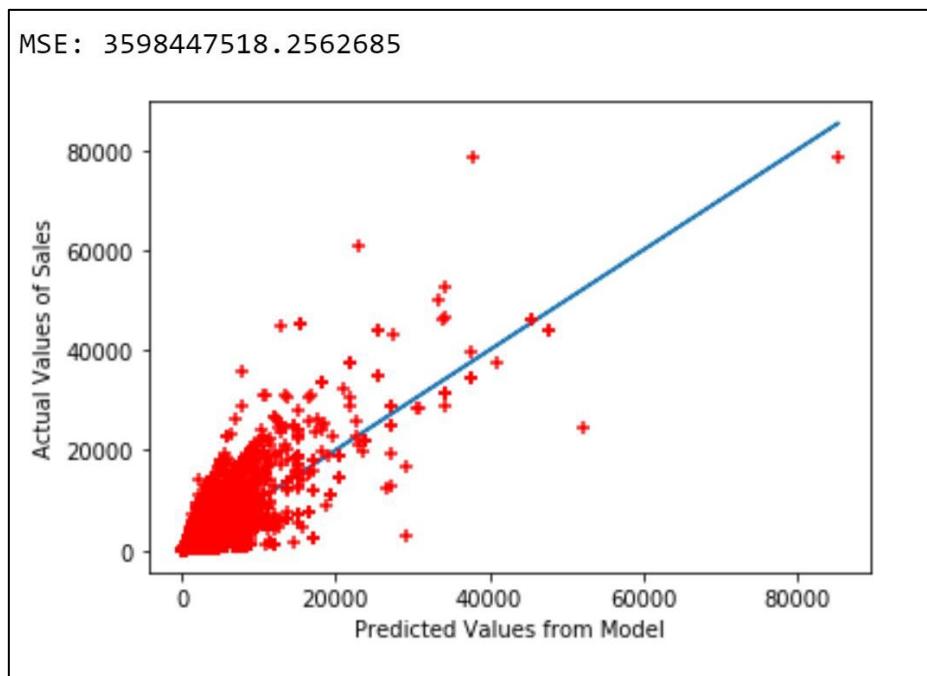


Figura 7.15: Predizioni e MSE

Nel complesso, dato l'alto valore dell'errore quadratico medio, dovremo trovare un modo per ridurlo al minimo.

7.1 Model testing e model evaluation

Si andranno ad applicare delle tecniche per testare e validare il modello realizzato.

7.1.1 Hold-out Validation

La prima tecnica consiste nel dividere il set di dati usato nel modello in due partizioni, una di training ed una di test attraverso il metodo dell'Hold-out. Il training set verrà usato per addestrare il modello e il testing set per testare il modello.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

(1510782, 101) (1510782,)
(647479, 101) (647479,)
```

Figura 7.16: Train/Test split

Dopo aver spartito i dati in una porzione di training e di test procediamo ad addestrare il modello sui dati appena spartiti. Un modello di regressione lineare:

```
lm = linear_model.LinearRegression()
model_sk = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
```

Figura 7.17: "fit" e prediction del modello di regressione

È possibile graficare le predizioni effettuate dal modello e valutare il valore di R squared:

```
plt.scatter(y_test, predictions)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

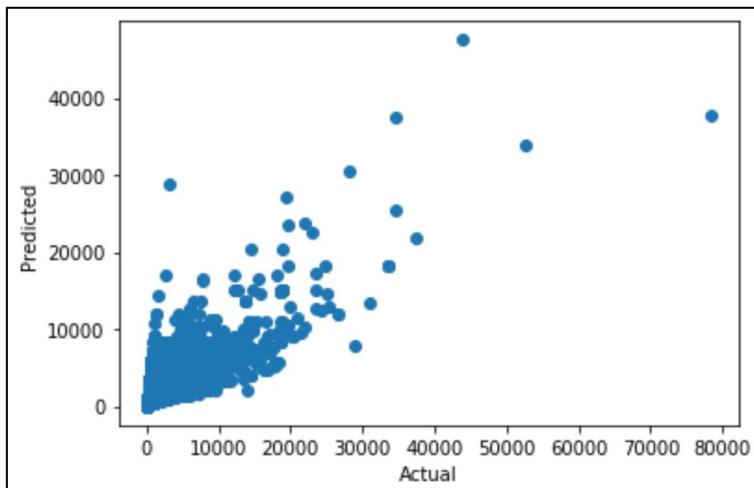


Figura 7.18: Grafico delle predizioni

```
print("Score : ", model_sk.score(X_test,y_test))

Score : 0.6683921980727632
```

Figura 7.19: Calcolo dello score

Il valore di R squared ottenuto risulta essere 0.66.

7.1.2 K-Cross Fold Validation

La seconda tecnica utilizzata è la K-Cross Fold Validation. In particolare, si applica una 5-Cross Fold Validation. In questo modo quindi avremo cinque valori di accuratezza, uno per ogni training effettuato. Ci interesserà il valore medio.

```
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics
```

Figura 7.20: Importazione librerie

```
scores = cross_val_score(model_sk, X_train, y_train, cv=5)
print("Cross-validated scores:", scores)
print("Average: ", scores.mean())

Cross-validated scores: [ 0.6911831   0.6763192   0.65765163  0.65777881  0.67087929]
Average:  0.6707624055574606
```

Figura 7.21: Valutazione dello score

Nell'immagine successiva, tramite uno scatterplot, si ha una vista grafica di quelli che sono i valori predetti ottenuti attraverso una 5-Fold Cross Validation.

```
predictions = cross_val_predict(model_sk, X, y, cv=5)
plt.scatter(y, predictions)
plt.xlabel('Actual')
plt.ylabel('Predicted')
accuracy = metrics.r2_score(y, predictions)
print("Cross-Predicted Accuracy:", accuracy)
```

Cross-Predicted Accuracy: 0.6714866253633374

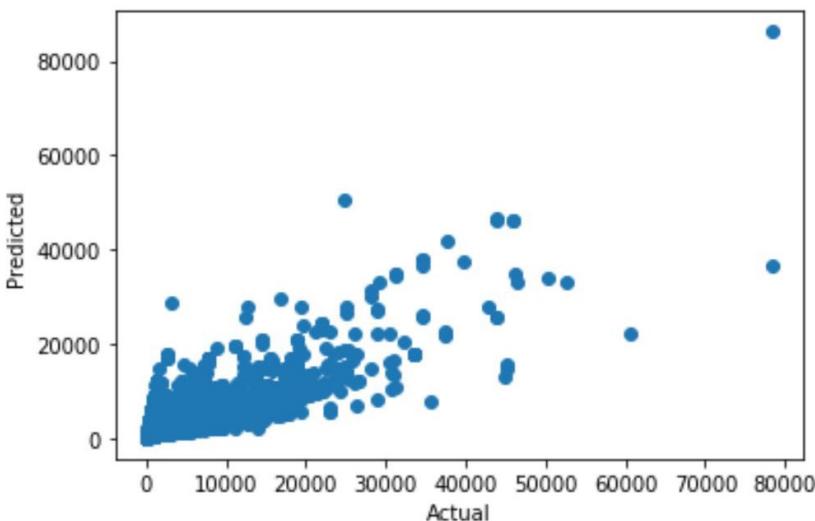


Figura 7.22: Grafico predizioni e R squared

Il valore di R squared ottenuto risulta essere 0.67.

Utilizzando sia il Train/Test Split, sia la K-cross fold validation si hanno valori di coefficienti di determinazioni molto simili.

7.2 Metodo di regolarizzazione

7.2.1 Lasso Regression

La tecnica presentata in questa sezione è la già nota tecnica di regolarizzazione Lasso Regression.

Dal punto di vista del codice, per prima cosa sceglieremo il parametro di regolazione alfa ottimale, come mostrato in figura. Quindi regolarizzeremo con una regressione Lasso. I coefficienti beta rimanenti alla fine della regolarizzazione sono i più importanti. Infine, quindi calcoliamo il valore dell'errore quadratico medio.

```
from sklearn.linear_model import LassoCV
lassoregcv = LassoCV(n_alphas=10, normalize=True, random_state=9)
lassoregcv.fit(X_train, y_train)
lassoregcv.alpha_
0.0003213258548369306
```

Figura 7.23: Selezione alfa ottimale

```
from sklearn.linear_model import Lasso
lassoreg = Lasso(alpha=0.0003, normalize=True)
lassoreg.fit(X_train, y_train)
lassor = lassoreg.coef_
lassor|
```

array([0. , -0. , -0. , 0. , 15.56012048, -7.24931444, -0. , 0. , 4.46556845, -0. , 0. , 0. , 0. , 6.40583109, -9.35369759, 0. , 0. , -2.84071558, -6.58837136, -0. , -5.55023521, -0. , -1.0582518 , 9.23845219, -0. , -0.86038823, -2.98972823, -5.15559875])
--

Figura 7.24: “fit” del modello con alfa ottimale

```
labeled = zip(X.columns, lassor)
```

```
y_pred = lassoreg.predict(X_test)
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
288.04424583712137
```

Figura 7.25: Valutazione MSE

L'errore quadratico medio relativo al modello con una regolarizzazione lasso è pari a 288,04. Questo risultato è molto interessante in quanto si è passati da un errore quadratico medio iniziale con stime OLS di 3.598.447.518,25 (quindi di un fattore 10^9 , altissimo) ad un errore molto più basso pari a 288,04.

Con il modello costruito, e dopo aver dedotto che Dallas è la contea che potrebbe potenzialmente portare a più vendite rispetto all'apertura di un negozio di liquori, in altre contee andremo a determinare quali fattori influenzano maggiormente le vendite nella contea di Dallas.

7.2.2 Analisi contea di Dallas

Per fare questo restringiamo il dataframe alla sola contea di Dallas, come mostrato nel codice seguente:

```
dallas_co = raw_data_19.copy(deep=True)
dallas_county = dallas_co.drop(dallas_co[dallas_co['County'] != 'DALLAS'].index, axis=0)
```

Figura 7.26: Filtraggio contea di Dallas

Di seguito, attraverso una mappa di calore si vanno a identificare eventuali correlazioni tra le variabili per capire in particolare quali sono più correlate alle vendite, Sale (Dollars).

```
sns.heatmap(dallas_county.corr(), annot = True, linewidths = 0.5)
```

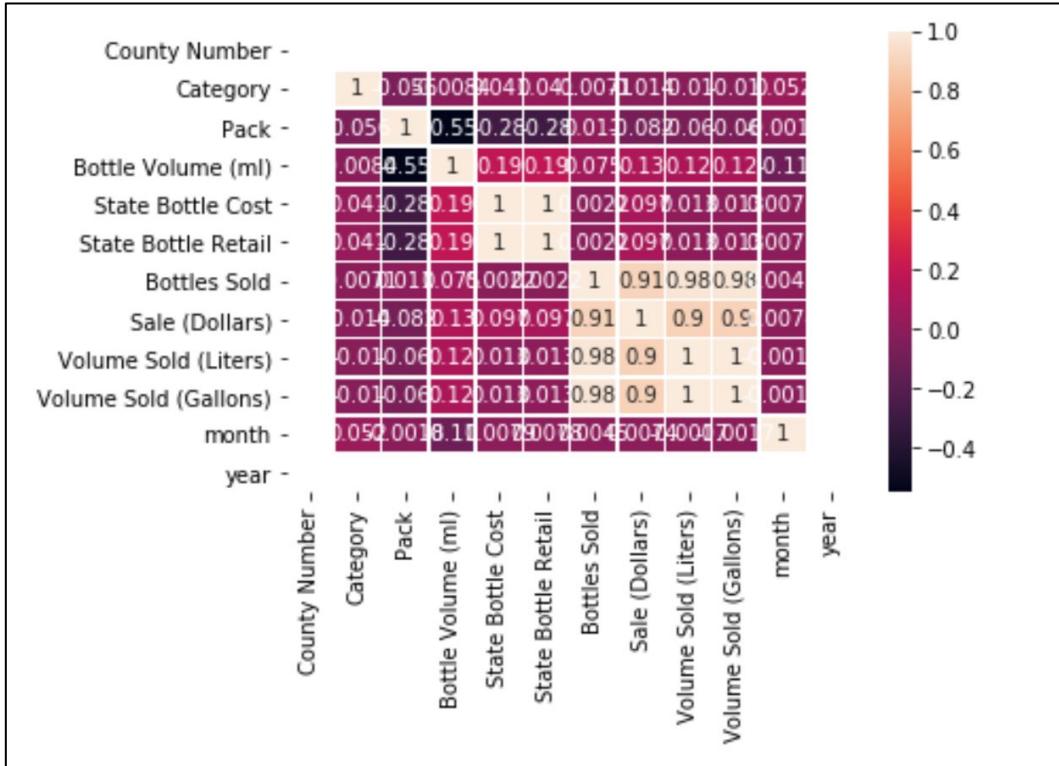


Figura 7.27: Mappa di calore per dallas_county

Possiamo vedere che il volume venduto e le bottiglie vendute hanno correlazioni con le vendite. Queste, quindi sarebbero buone variabili da includere nel nostro modello. Il prezzo al dettaglio (State bottle retail) ha anche una leggera correlazione, quindi possiamo includere anche questa variabile. Il resto delle nostre variabili ha scarso effetto sulle vendite, quindi non avrebbe senso includerle.

Dunque, dallas_short è il dataframe contenente le variabili scelte a partire dalla mappa di calore. Questo dataframe contiene sostanzialmente i valori su cui addestreremo il nostro modello.

```
dallas_short = dallas_county[['State Bottle Retail', 'Bottles Sold', 'Sale (Dollars)', 'Volume Sold (Liters)']]
```

	State Bottle Retail	Bottles Sold	Sale (Dollars)	Volume Sold (Liters)
965201	2.70	24	64.80	9.0
965396	10.38	6	62.28	10.5
965435	8.22	12	98.64	9.0
965525	7.43	1	7.43	0.5
965618	10.13	6	60.78	4.5

Figura 7.28: Dataframe dallas_short

A questo punto, per poter creare il modello si ha la necessità di individuare la “Y”, ossia la variabile da predire. Sulla base del ragionamento fatto, questa corrisponderà alle vendite e quindi a “Sale (Dollars)”, come mostrato in figura.

```
y = dallas_short['Sale (Dollars)']
y.shape
(28420, )
```

Figura 7.29: Variabile y da predire

Per quanto riguarda la “X”, questa corrisponderà, in accordo al ragionamento fatto, al dataframe “dallas_short”, escluse ovviamente le vendite.

```
X = dallas_short.copy(deep=True)
X = X.drop('Sale (Dollars)', axis=1)
X = sm.add_constant(X)
X.shape
(28420, 4)
```

Figura 7.30: Variabile X su cui addestrare il modello

Ecco che a questo punto è possibile creare il modello attraverso il metodo dei minimi quadrati (OLS):

```
model_dallas = sm.OLS(y, X).fit()
predictions = model_dallas.predict(X)
model_dallas.summary()
```

Figura 7.31: Metodo minimi quadrati

Di seguito è riportato il summary del modello realizzato:

OLS Regression Results									
Dep. Variable:	Sale (Dollars)	R-squared:	0.833						
Model:	OLS	Adj. R-squared:	0.833						
Method:	Least Squares	F-statistic:	4.723e+04						
Date:	Thu, 28 Jan 2021	Prob (F-statistic):	0.00						
Time:	16:38:56	Log-Likelihood:	-2.2008e+05						
No. Observations:	28420	AIC:	4.402e+05						
Df Residuals:	28416	BIC:	4.402e+05						
Df Model:	3								
Covariance Type:	nonrobust								
	coef	std err	t	P> t	[0.025	0.975]			
const	-46.4402	4.525	-10.264	0.000	-55.309	-37.572			
State Bottle Retail	6.5816	0.172	38.227	0.000	6.244	6.919			
Bottles Sold	10.4436	0.179	58.397	0.000	10.093	10.794			
Volume Sold (Liters)	2.1212	0.106	20.081	0.000	1.914	2.328			
Omnibus:	38831.372	Durbin-Watson:		1.925					
Prob(Omnibus):	0.000	Jarque-Bera (JB):		70139552.765					
Skew:	7.136	Prob(JB):		0.00					
Kurtosis:	245.956	Cond. No.		239.					

Figura 7.32: Summary modello realizzato

Si vede che tutte e tre le variabili che abbiamo scelto, Volume venduto (litri), Bottiglie vendute e prezzo al dettaglio sono tutte statisticamente significative. Ciò dimostra che hanno tutte un effetto significativo sulle vendite.

```
plt.scatter(predictionss, y, s=30, c='r', marker='+', zorder=10)
plt.xlabel("Predicted Values from Model")
plt.ylabel("Actual Values of Sales")
plt.plot(predictionss, np.poly1d(np.polyfit(predictionss, y, 1))(predictionss))
plt.show()
print("MSE:", model.mse_model)
```

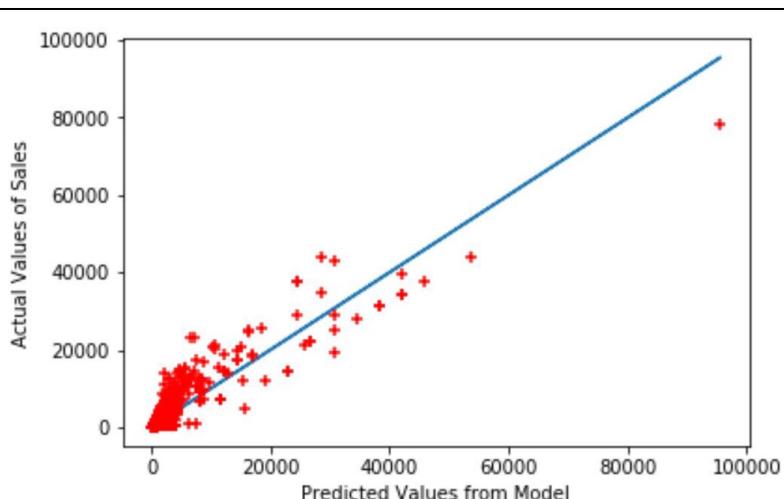


Figura 7.33: Grafico predizioni e MSE

L'errore quadratico medio però è abbastanza alto e come prima si procede ad una serie di metodologie per ridurlo al minimo. Si procede con una 10-cross Fold Validation, che ci permette di ottenere un valore di accuratezza media pari a 0.79, come mostrato in figura:

```
Xd_train, Xd_test, yd_train, yd_test = train_test_split(X, y, test_size=0.3)
print(Xd_train.shape, yd_train.shape)
```

```
(19894, 4) (19894,)
```

```
lm = linear_model.LinearRegression()
modeld_sk = lm.fit(Xd_train, yd_train)
predictions = lm.predict(Xd_test)
```

```
scores = cross_val_score(modeld_sk, Xd_train, yd_train, cv=10)
print("Cross-validated scores:", scores)
print("Average: ", scores.mean())
```

```
Cross-validated scores: [0.80484104 0.75207967 0.88032112 0.67421331 0.797038
31 0.77024031
 0.78904498 0.77441628 0.83421582 0.87892469]
Average:  0.7955335521227316
```

Figura 7.34: 10-cross Fold Validation

Inoltre, nell'immagine successiva è mostrato il grafico relativo alle predizioni. Si nota come di fatto quasi tutti i punti tendono a distribuirsi lungo la diagonale principale quindi a monito che molti dei valori predetti corrispondono ai valori veri.

```
predictions = cross_val_predict(modeld_sk, X, y, cv=10)
plt.scatter(y, predictions)
plt.xlabel('Actual')
plt.ylabel('Predicted')
accuracy = metrics.r2_score(y, predictions)
print("Cross-Predicted Accuracy:", accuracy)
```

```
Cross-Predicted Accuracy: 0.818079219296339
```

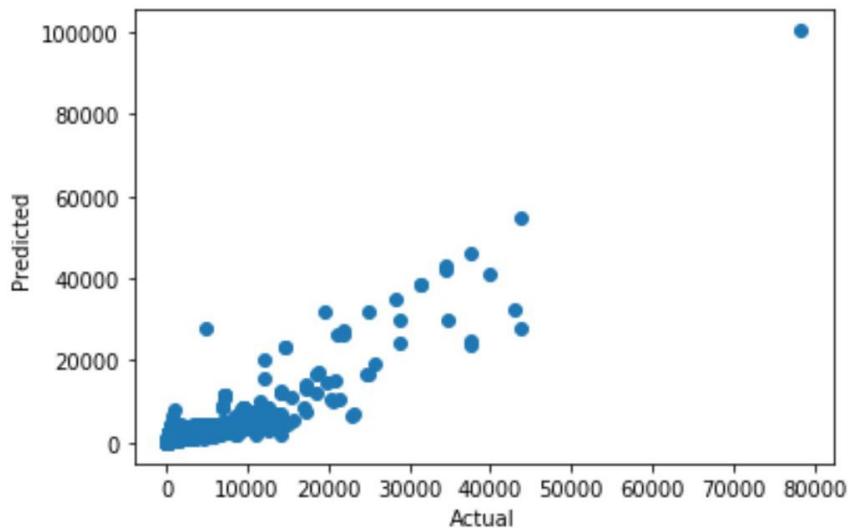


Figura 7.35: Grafico predizioni e R squared

Ritorna anche in questo caso la tecnica di regolarizzazione Lasso, utilizzata precedentemente. In maniera analoga si va ad individuare il miglior coefficiente alpha che utilizzeremo nel modello.

```
lassoregcv = LassoCV(n_alphas=10, normalize=True, random_state=9)
lassoregcv.fit(Xd_train, yd_train)
lassoregcv.alpha_
0.389553566645013
```

Figura 7.36: Selezione alfa ottimale

```
lassoreg = Lasso(alpha=0.38, normalize=True)
lassoreg.fit(Xd_train, yd_train)
lassord = lassoreg.coef_
lassord
array([ 0.          ,  3.32736994, 11.59821513,  1.40061334])

yd_pred = lassoreg.predict(Xd_test)
print(np.sqrt(metrics.mean_squared_error(yd_test, yd_pred)))
576.7160820578121

print(X.columns)
print(lassord)

Index(['const', 'State Bottle Retail', 'Bottles Sold', 'Volume Sold (Liter s)'], dtype='object')
[ 0.          3.32736994 11.59821513  1.40061334]
```

Figura 7.37: Lasso regression

Di nuovo, con l'introduzione di una tecnica di regolarizzazione, in questo caso utilizzando Lasso regression, l'errore è diminuito clamorosamente. Da un valore di 3×10^9 , si è passati ad un valore di 5×10^2 . Una differenza enorme.

Inoltre, i coefficienti stimati sono:

State Bottle Retail = 3.32

Bottles Sold = 11.59

Volume Sold (Liters) = 1.40

In conclusione, usando una regressione lasso, abbiamo un errore quadratico medio (MSE) molto più piccolo. Vediamo anche che, analizzando i coefficienti, la variabile Bottiglie Vendute (Bottle Sold) ha l'effetto maggiore sulle vendite nella contea di Dallas. Quindi, se dovessimo fare una raccomandazione all'apertura di un negozio nella contea di Dallas, massimizzare il numero di bottiglie vendute è il modo più alto per aumentare le vendite.

In relazione ad i risultati ottenuti, per raffinare meglio l'analisi si sono individuati tutti i negozi all'interno della contea di Dallas che hanno massimizzato le vendite, in termini di bottiglie vendute.

Per far questo si è, in prima battuta, ristretto il dataset ai soli campi di interesse:

```
df_dal = dallas_county.copy(deep=True)
df_dal = df_dal.drop('Date', axis=1)
df_dal = df_dal.drop('City', axis=1)
df_dal = df_dal.drop('Zip Code', axis=1)
df_dal = df_dal.drop('County Number', axis=1)
df_dal = df_dal.drop('County', axis=1)
df_dal = df_dal.drop('Category', axis=1) |
df_dal = df_dal.drop('Vendor Number', axis=1)
df_dal = df_dal.drop('Item Number', axis=1)
df_dal = df_dal.drop('Item Description', axis=1)
df_dal = df_dal.drop('Bottle Volume (ml)', axis=1)
df_dal = df_dal.drop('State Bottle Cost', axis=1)
df_dal = df_dal.drop('Volume Sold (Gallons)', axis=1)
df_dal.head()
```

Figura 7.38: Restrizione dataset

Successivamente, un groupby sul campo “Store Number” per raggruppare i negozi mostrando la somma delle bottiglie vendute, ordinati dal negozio che ha venduto di più a quello che ha venduto di meno.

```
df_dal.groupby('Store Number')[['Bottles Sold']].sum().sort_values(ascending=False)
```

Figura 7.39: Raggruppamento per negozio

L’output che ne deriva è il seguente:

Store Number	Bottles Sold
2665	147846
3814	146246
4678	26860
2612	18835
4137	11881
4320	8875
4384	8201
5411	7223
4411	6271
4378	5953
4868	5463
5384	4652
5235	4596
4359	4362
4873	4312
4792	3545
4577	3374
4929	3234
5699	3150
5545	2758
5562	2705
5521	2602
5467	2544
4623	2260
5409	2228
5723	2123
2688	1853
5844	1135
5809	1047
5843	143

Figura 7.40: Classifica dei negozi

Come si può notare dall’output precedente, il negozio con il maggior numero di bottiglie vendute nella contea di Dallas è il negozio il cui “Store Number” è pari a 2665. Maggiori informazioni sui primi negozi sarebbero molto utili, quindi si procede ad individuarle.

Tramite gli script successivi si andrà ad estrarre, per un particolare negozio, le categorie di alcolici più venduti. Gli output relativi, quindi, saranno costituiti dalle prime cinque categorie di alcolico più venduto.

Negozio numero 2665:

```
primo_storenum=df_dal['Store Number'] == '2665'
primostore=df_dal[primo_storenum]
primostore['Category Name'].value_counts()
```

American Vodkas	1106
Straight Bourbon Whiskies	677
Canadian Whiskies	574
American Flavored Vodka	481
Cocktails /RTD	455

Figura 7.41: Analisi negozio 2665

Il negozio il cui store number è 2665, e quindi il negozio che ha venduto più bottiglie nella contea di Dallas, mostra come tipologia di alcolico più venduta American Vodkas, a seguire mantengono il podio Straight Bourbon Whiskies e Canadian Whiskies.

Negozio numero 3814:

```
sec_storenum=df_dal['Store Number'] == '3814'
secstore=df_dal[sec_storenum]
secstore['Category Name'].value_counts()
```

Straight Bourbon Whiskies	223
American Vodkas	90
Canadian Whiskies	74
Scotch Whiskies	72
Single Malt Scotch	51

Figura 7.42: Analisi negozio 3814

Si nota che anche il secondo negozio che vende di più presenta sul podio categorie di alcolici come Vodka e Wishkies. Una differenza rispetto al negozio precedente è che questa volta il primo posto è preso da Straigth Bourbon Whiskies, e a seguire American Vodkas.

Negozio numero 4678:

```
terzo_storenum=df_dal['Store Number'] == '4678'
terzostore=df_dal[terzo_storenum]
terzostore['Category Name'].value_counts()
```

American Vodkas	197
Canadian Whiskies	176
Straight Bourbon Whiskies	115
Cocktails /RTD	112
Spiced Rum	84

Figura 7.43: Analisi negozio 4678

Per il terzo negozio che vende di più, i primi tre alcolici più veduti coincidono con i primi tre alcolici più venduti dei negozi precedenti, cambia solo l'ordine.

Negozio numero 2612:

quarto_storenum=df_dal['Store Number'] == '2612'		
quartostore=df_dal[quarto_storenum]		
quartostore['Category Name'].value_counts()		
American Vodkas	483	
American Flavored Vodka	323	
Canadian Whiskies	248	
Straight Bourbon Whiskies	185	
Spiced Rum	160	

Figura 7.44: Analisi negozio 2612

Il quarto negozio preso in esame presenta tra i primi tre alcolici una categoria non presente nei precedenti negozi, questa è American Flavored Vodka. Si nota come il podio è costituito essenzialmente da alcolici come la Vodka.

Conclusioni

In conclusione, tutti i negozi analizzati presentano tra le bottiglie più vendute tutte quelle appartenenti ad alcolici come la Vodka e il Whiskies. Questa informazione si aggiunge alle precedenti valutazioni fatte per poter dire che, se dovessimo fare una raccomandazione all'apertura di un negozio nella contea di Dallas, sarebbe che massimizzare il numero di bottiglie vendute è il modo più alto per aumentare le vendite. Inoltre, se queste bottiglie fossero di Vodka o di Whiskies si potrebbero potenzialmente avere maggiori vendite. Aggiungendo anche l'informazione appresa grazie al clustering l'ideale sarebbe massimizzare il numero di bottiglie da 20cl o 50cl in quanto potrebbero indurre ad un maggior numero di bottiglie vendute.

8. Analisi temporali

All'interno di questa sezione verranno trattate le serie temporali. Per serie si intende la classificazione di diverse osservazioni di un fenomeno rispetto a un carattere qualitativo. Se tale carattere è il tempo, la serie viene detta temporale.

Prendendo in riferimento il nostro dataset, andremo ad analizzare le tendenze che influenzano le vendite di liquori nello stato dell'Iowa, in America, e andremo a prevedere il confronto delle performance dei marchi concorrenti.

Vengono utilizzate le "Time Series Analysis" per estrarre le tendenze e costruire modelli "Auto Regressive Integrated Moving Average" (ARIMA) per produrre previsioni.

8.1 Data Preprocessing

All'interno di questa sezione ci occupiamo delle operazioni preliminari prima di poter generare un modello di previsione. Questa fase è detta preprocessing e risulta molto importante, bisogna quindi valutare quali sono i dati di interesse per la nostra analisi ed eliminare quindi quelli di non interesse.

Nel nostro caso, dato che vogliamo analizzare le vendite di Diageo Americas, il marchio che vende di più, la fase di preprocessing consiste quindi nel filtrare il marchio da analizzare ed eliminare tutti i campi di non interesse per l'analisi. Come dalla porzione di codice seguente, si elimineranno tutti i campi contenuti nella variabile "cols". Manteniamo dunque solo le informazioni relative alle vendite, all'orizzonte temporale ed al marchio di interesse. Il tutto è contenuto nel dataframe DA. Inoltre, l'indice del dataframe è il campo 'Date', quindi faranno da indice le date relative alle vendite.

```
DA = df.loc[df['Vendor Name'] == 'DIAGEO AMERICAS']
cols = ['Invoice/Item Number', 'Store Number', 'Store Name', 'Address', 'City', 'Zip Code',
        'Store Location', 'County Number', 'County', 'Category', 'Category Name', 'Vendor Number',
        'Vendor Name', 'Item Number', 'Item Description', 'Pack', 'Bottle Volume (ml)', 'State Bottle Cost',
        'State Bottle Retail', 'Bottles Sold', 'Volume Sold (Liters)', 'Volume Sold (Gallons)']
DA.drop(cols, axis = 1, inplace = True)
DA.isnull().sum()

DA = DA.groupby('Date')['Sale (Dollars)'].sum().reset_index()
DA = DA.set_index('Date')
DA.index

DatetimeIndex(['2012-01-03', '2012-01-04', '2012-01-05', '2012-01-09',
               '2012-01-10', '2012-01-11', '2012-01-12', '2012-01-16',
               '2012-01-17', '2012-01-18',
               ...
               '2020-11-17', '2020-11-18', '2020-11-19', '2020-11-20',
               '2020-11-23', '2020-11-24', '2020-11-25', '2020-11-27',
               '2020-11-28', '2020-11-30'],
              dtype='datetime64[ns]', name='Date', length=2135, freq=None)
```

Figura 8.1: Preprocessing

8.2 Modello serie temporale ARIMA

Uno dei metodi più comuni utilizzati nella previsione di serie temporali è conosciuto come il modello ARIMA, che sta per **A**utoreg**R**essive **I**ntegrated **M**oving **A**verage. ARIMA è un modello che può essere adattato ai dati delle serie temporali al fine di comprendere o prevedere meglio i punti futuri della serie.

Ci sono tre numeri interi distinti (p, d, q) che vengono utilizzati per parametrizzare modelli ARIMA. Per questo motivo, i modelli ARIMA sono indicati con la notazione ARIMA (p, d, q).

Insieme, questi tre parametri tengono conto della stagionalità, della tendenza e del rumore nei set di dati.

- ‘ p ’ è la parte auto-regressive del modello: ci permette di incorporare l’effetto dei valori passati nel nostro modello;
- ‘ d ’ è la parte integrated del modello: ciò include i termini nel modello che incorporano la quantità di differenza (ovvero il numero di punti temporali passati da sottrarre dal valore corrente) da applicare alle serie temporali;
- ‘ q ’ è la parte della moving average del modello: questo ci consente di impostare l’errore del nostro modello come una combinazione lineare dei valori di errore osservati in punti temporali precedenti in passato.

Quando si tratta di effetti stagionali, utilizziamo l’ARIMA *stagionale*, che è indicato come ARIMA(p, d, q)(P, D, Q) s .

Di seguito (p, d, q) sono riportati i parametri non stagionali sopra descritti, (P, D, Q) seguono la stessa definizione ma vengono applicati alla componente stagionale delle serie storiche. Il termine “ s ” è la periodicità delle serie storiche (4 per periodi trimestrali, 12 per periodi annuali, ecc.).

Il metodo ARIMA stagionale può sembrare scoraggiante a causa dei molteplici parametri di regolazione coinvolti. Nella sezione successiva, descriveremo come automatizzare il processo di identificazione dell’insieme ottimale di parametri per il modello delle serie temporali ARIMA stagionali.

8.3 Trend Extraction

È possibile ora graficare la serie temporale relativa alle vendite del brand più venduto nello stato dell'Iowa, Diageo Americas.

```
DA.plot(figsize=(20,10));
```

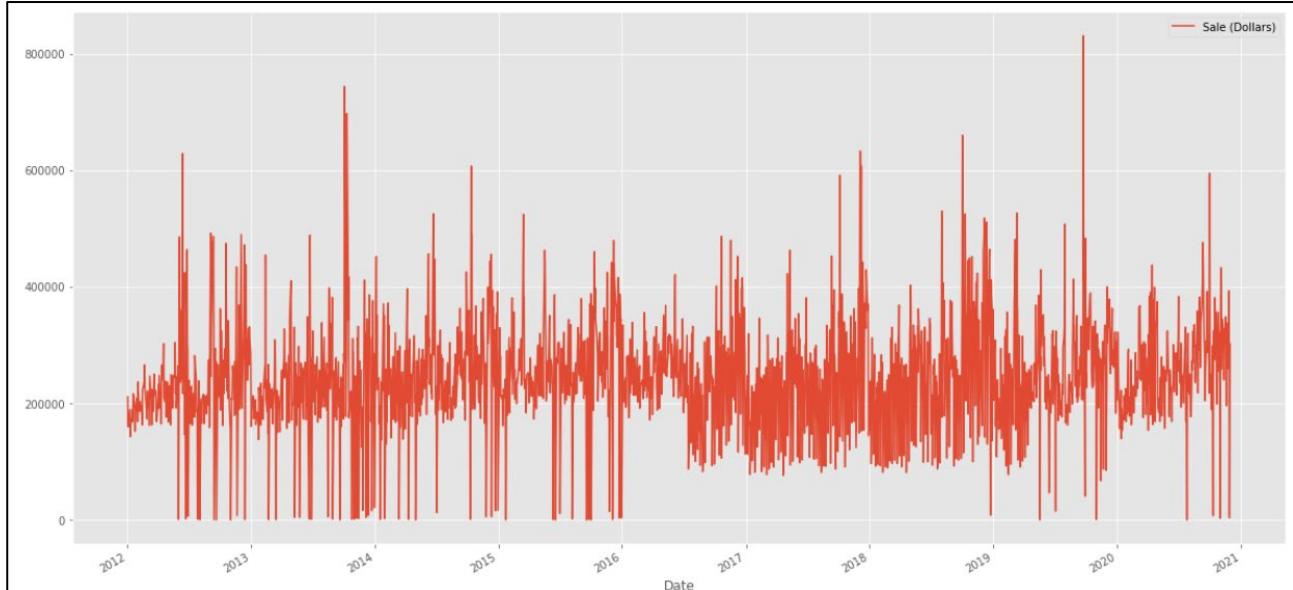


Figura 8.2: Serie temporale delle vendite di Diageo Americas

I nostri dati datetime correnti si presentano quindi in maniera complessa per poter essere analizzati, pertanto, utilizzeremo invece il valore medio delle vendite giornaliere per mese e utilizzeremo l'inizio di ogni mese come timestamp.

```
DA.head()  
y = DA['Sale (Dollars)'].resample('MS').mean()  
y['2012':].describe()  
y.plot(figsize=(20,10))
```

Figura 8.3: Resample della serie

Attraverso la funzione “Resample” ricampioniamo i dati delle serie temporali in modo da osservare le vendite di Diageo Americas nel tempo, come in figura:

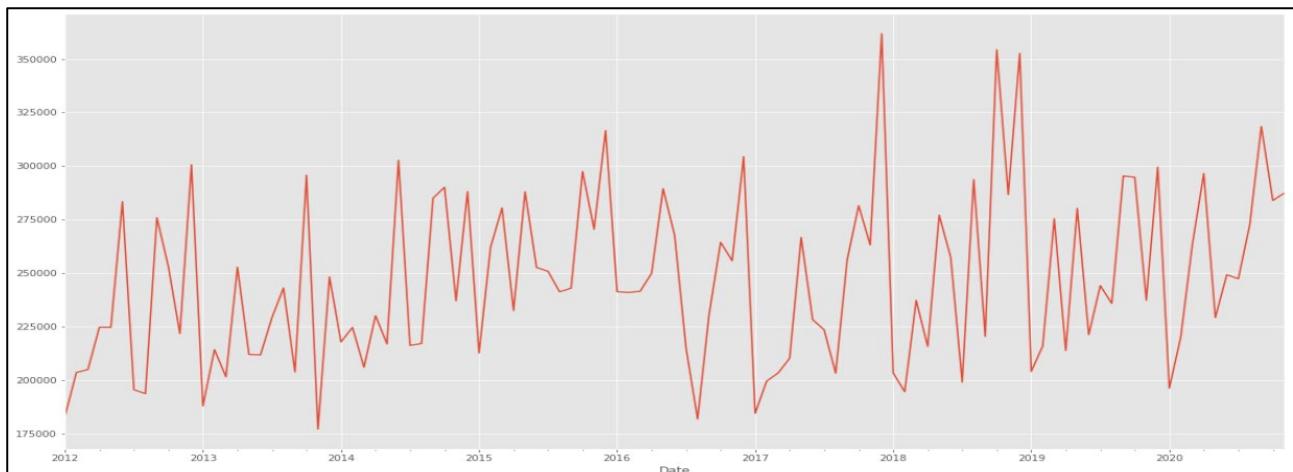


Figura 8.4: Serie temporale post resample

Attraverso il codice successivo viene effettuata una decomposizione stagionale utilizzando medie mobili, tramite “sm.tsa.seasonal_decompose”, alla quale viene passato come argomento y in quanto è la serie temporale e come tipo di componente stagionale si imposta additivo.

```
from pylab import rcParams
rcParams['figure.figsize'] = 18,8
decomposition = sm.tsa.seasonal_decompose(y, model = 'additive')
fig = decomposition.plot()
```

Figura 8.5: Codice per decomposizione stagionale

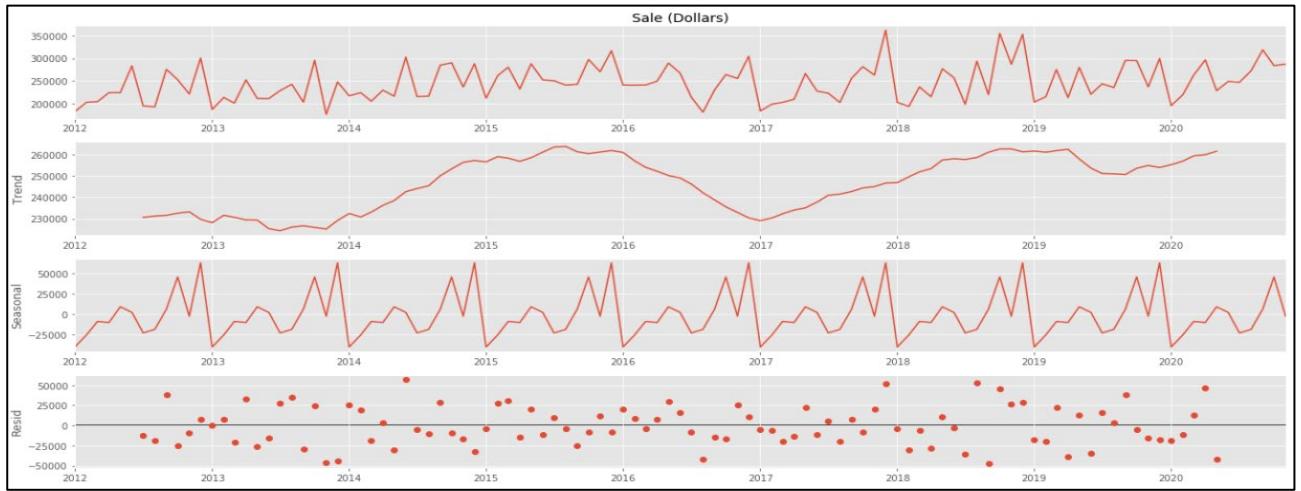


Figura 8.6: Grafico decomposizione stagionale

Notiamo come le vendite di Diageo Americas, su tutta la finestra temporale della serie, mostrano un aumento interessante solo a partire all’ultimo trimestre del 2013 e fino all’anno 2016, raggiungendo picchi tra il 2015 e il 2016. Notiamo invece come sia stato un anno negativo per le vendite il 2016, in quanto i valori scendono a numeri precedenti l’anno 2014. Dal 2017, il trend delle vendite torna a crescere e si mantiene alto e costante negli anni. Si mostra solo una leggera diminuzione a cavallo tra il secondo e terzo trimestre dell’anno contrastata però da una nuova crescita a partire dall’ultimo trimestre dello stesso anno.

Sembra anche che le vendite abbiano una periodicità di 12 mesi. In particolare, notiamo come sia presente quindi una stagionalità con picchi nell’ultimo trimestre di ogni anno.

8.4 Modellazione ARIMA

8.4.1 Selezione dei parametri per il modello di serie storica ARIMA

Prima di iniziare la costruzione del modello ARIMA, dobbiamo selezionare i parametri ottimali (p , d , q , s). Poiché abbiamo già stabilito che la periodicità è di 12 mesi, possiamo lasciare $s = 12$. Per determinare le stime per p , d e q , simuleremo tutte le combinazioni dei parametri, tramite una ricerca a griglia e selezioneremo il modello con il punteggio AIC più basso.

```
import itertools
p = d = q = range(0, 2) |
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                              order=param,
                                              seasonal_order=param_seasonal,
                                              enforce_stationarity=False,
                                              enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

Figura 8.7: Identificazione parametri ottimali

Nel codice riportato, useremo una "ricerca a griglia" per esplorare in modo iterativo diverse combinazioni di parametri. Per ogni combinazione di parametri, adattiamo un nuovo modello ARIMA stagionale con la funzione SARIMAX del modulo statsmodel e valutiamo la sua qualità complessiva. Una volta esplorato l'intero panorama dei parametri, il nostro insieme ottimale di parametri sarà quello che fornisce le migliori prestazioni per i criteri di interesse. Cominciamo generando le varie combinazioni di parametri che desideriamo valutare.

Inoltre, quando si valutano e si confrontano modelli statistici dotati di parametri diversi, ciascuno può essere classificato l'uno rispetto all'altro in base a quanto bene si adatta ai dati o alla sua capacità di prevedere con precisione i punti dati futuri. Useremo il valore AIC (Akaike Information Criterion), che viene restituito con i modelli ARIMA utilizzando appunto statsmodel. Le misure di AIC ci dicono quanto bene un modello si adatta ai dati, tenendo conto della complessità generale del modello. Noi siamo interessati a trovare il modello che produce il valore AIC più basso.

Continuando con il codice sopra riportato, questo esegue l'iterazione delle combinazioni di parametri e utilizza la funzione SARIMAX() da *statsmodel* per adattare il modello ARIMA stagionale corrispondente. Qui, l'argomento *order* specifica i parametri (p , d e q) , mentre l'argomento *seasonal_order* specifica la componente stagionale (p,d,q,s) del modello ARIMA stagionale. Dopo aver montato ogni modello SARIMAX(), il codice stampa il rispettivo punteggio di AIC.

L'output del nostro codice suggerisce che SARIMAX(1,0,1)x(0,1,1,12) produce il valore AIC più basso e pari a 1928,76. Dovremmo quindi considerare questa opzione ottimale tra tutti i modelli che abbiamo considerato.

8.4.2 Adattamento di un modello di serie storica ARIMA

Utilizzando la ricerca griglia, abbiamo identificato l'insieme di parametri che produce il modello più adatto ai dati delle nostre serie temporali. Possiamo procedere ad analizzare questo particolare modello in modo più approfondito.

Inizieremo inserendo i valori dei parametri ottimali in un nuovo modello SARIMAX():

```
mod = sm.tsa.statespace.SARIMAX(y,
                                 order = (1,0,1),
                                 seasonal_order = (0,1,1,12),
                                 enforce_stationarity = False,
                                 enforce_invertibility = False)
results = mod.fit()
print(results.summary().tables[1])
results.plot_diagnostics(figsize = (16,8))
```

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.9383	0.023	41.580	0.000	0.894	0.983
ma.L1	-0.8475	0.103	-8.222	0.000	-1.049	-0.645
ma.S.L12	-0.4484	0.153	-2.937	0.003	-0.748	-0.149
sigma2	1.543e+09	9.99e-12	1.54e+20	0.000	1.54e+09	1.54e+09

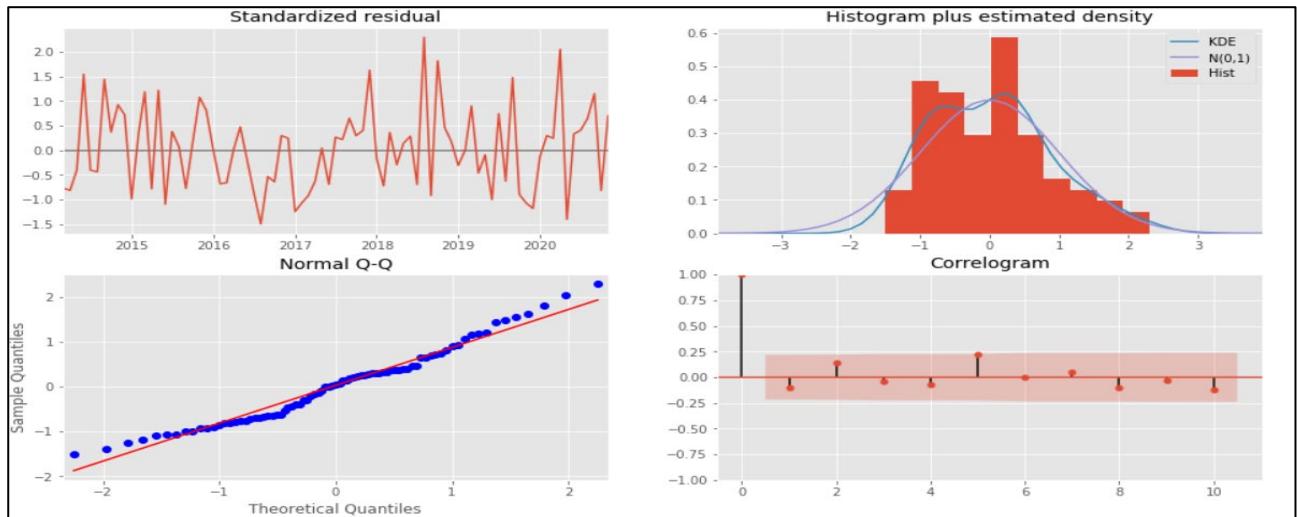


Figura 8.8: Codice e diagnostica del modello

Il summary di output ottenuto tramite modellazione SARIMAX restituisce una notevole quantità di informazioni, ma focalizzeremo la nostra attenzione sulla tabella dei coefficienti. La colonna “coef” mostra il peso (ovvero l’importanza) di ogni caratteristica e il modo in cui ciascuna di esse influisce sulla serie temporale. La colonna “P>|z|” ci informa del significato di ogni peso della caratteristica. In questo caso, ogni peso ha un valore p inferiore o vicino a 0.05, quindi è ragionevole mantenerli tutti nel nostro modello.

Quando si adattano i modelli ARIMA stagionali è importante eseguire la diagnostica del modello. L’oggetto `plot_diagnostics` ci consente di generare rapidamente la diagnostica del modello e di indagare su qualsiasi comportamento insolito.

La nostra preoccupazione principale era garantire che i residui del nostro modello non siano correlati e normalmente distribuiti con media zero. Se il modello ARIMA stagionale non soddisfacesse queste proprietà, sarebbe stato bene migliorare il modello.

In questo caso, però, la nostra diagnostica del modello suggerisce che i residui del modello sono normalmente distribuiti in base a quanto segue:

- Nel grafico in alto a destra, vediamo che la KDE segue strettamente con la linea “N(0,1)” (dove N(0,1) è la notazione standard per una distribuzione normale con media 0 e deviazione standard di 1. Questa è una buona indicazione che i residui sono normalmente distribuiti.

- Il grafico in basso a sinistra mostra che la distribuzione ordinata dei residui (punti blu) segue l'andamento lineare dei campioni presi da una distribuzione normale standard con $N(0, 1)$. Ancora una volta, questa è una forte indicazione che i residui sono normalmente distribuiti.
- I residui nel tempo (grafico in alto a sinistra) non mostrano alcuna stagionalità evidente e sembrano essere rumore bianco. Ciò è confermato dal grafico di autocorrelazione (il correlogramma) in basso a destra, che mostra che i residui delle serie temporali hanno una bassa correlazione con le versioni ritardate di sé stesse. Queste osservazioni ci portano a concludere che il nostro modello produce un adattamento soddisfacente che potrebbe aiutarci a comprendere i dati delle nostre serie temporali e prevedere i valori futuri.

8.5 Convalida delle previsioni

Abbiamo ottenuto un modello per le nostre serie temporali che ora può essere utilizzato per produrre previsioni. Iniziamo confrontando i valori previsti con i valori reali delle serie temporali, che ci aiuteranno a comprendere l'accuratezza delle nostre previsioni. Gli attributi `get_prediction()` e `conf_int()` ci consentono di ottenere i valori e gli intervalli di confidenza associati per le previsioni delle serie temporali.

```
pred = results.get_prediction(start=pd.to_datetime('2018-11-01'), dynamic=False)
ax = y['2012':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7, figsize=(14, 7))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
```

Figura 8.9: Validazione delle previsioni

L'argomento `dynamic = False` garantisce che produciamo previsioni un passo avanti, il che significa che le previsioni in ogni punto vengono generate utilizzando la cronologia completa fino a quel punto. Possiamo tracciare inoltre i valori reali e quelli previsti delle serie temporali delle vendite per valutare anche visivamente quanto sia buono il modello.

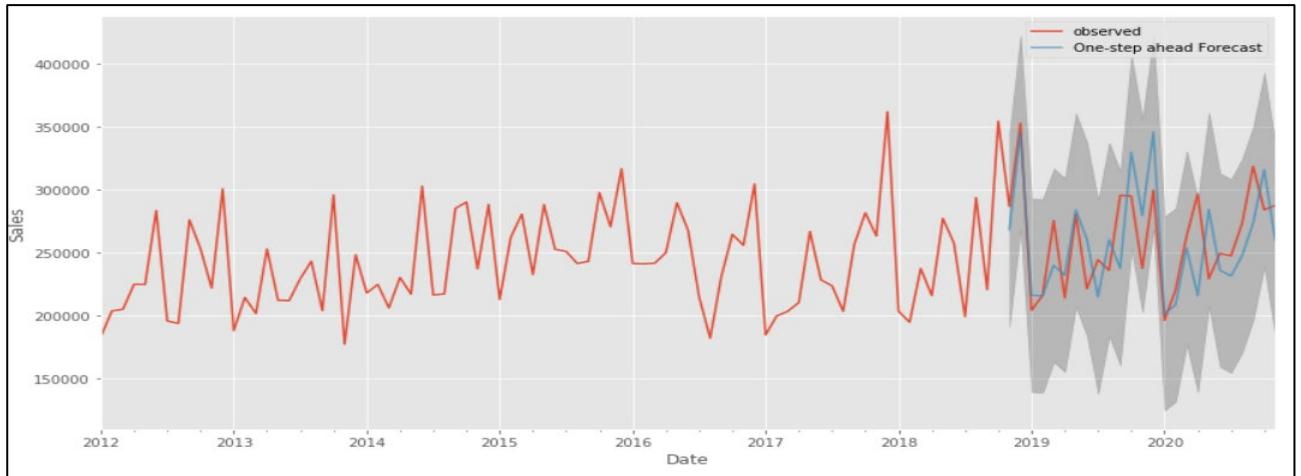


Figura 8.10: Grafico validazione delle previsioni

Nel complesso, le nostre previsioni si allineano abbastanza bene con i valori reali.

È anche utile quantificare l'accuratezza delle nostre previsioni. Per farlo utilizzeremo il metodo MAPE, una media degli errori percentuali assoluti.

```
y_forecasted = pred.predicted_mean # y predette
y_truth = y['2018-11-01':] # osservazioni corrispondenti alle y vere

def mape(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.mean(np.abs((actual - pred) / actual)) * 100

mape = mape(y_truth, y_forecasted)
print(mape)

10.44680759529301
```

Figura 8.11: Valutazione errore tramite MAPE

8.6 Previsioni

A questo punto, dopo aver validato il modello si passa alle previsioni. Obiettivo di questa sezione era infatti prevedere le vendite del miglior marchio presente.

Possiamo utilizzare l'output di questo codice per tracciare le serie temporali e le previsioni dei suoi valori futuri.

```
pred_uc = results.get_forecast(steps=50)
pred_ci = pred_uc.conf_int()
ax = y.plot(label='observed', figsize=(14, 7))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                 pred_ci.iloc[:, 0],
                 pred_ci.iloc[:, 1], color='k', alpha=.25)

ax.set_xlabel('Date')
ax.set_ylabel('Sales ($)')
ax.set_title('DIAGEO AMERICAS')
plt.legend()
```

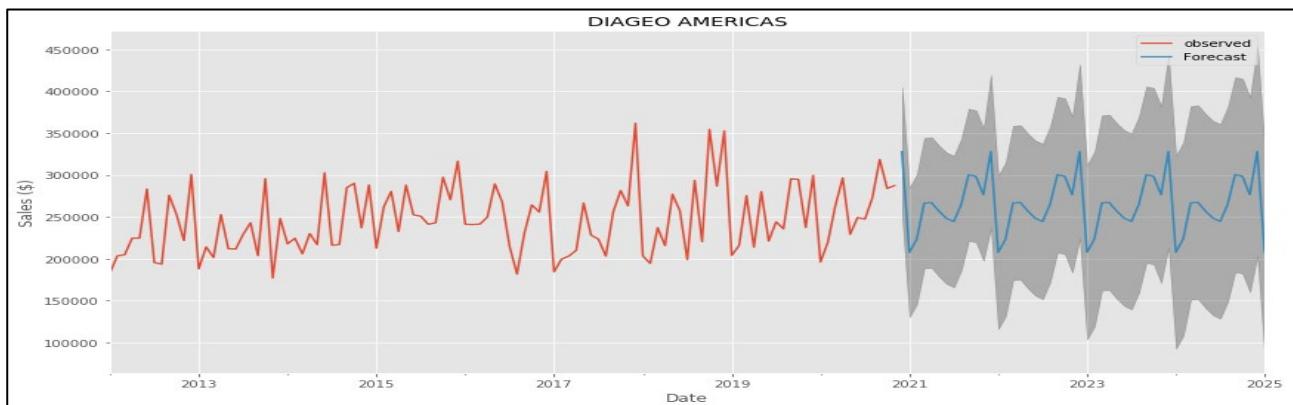


Figura 8.12: Codice e grafico previsioni

Come si nota dal grafico, le previsioni effettuate dal nostro modello sono mostrate dalla linea blu ed interessano un intervallo temporale di 4 anni, compreso tra inizio 2021 e fino alla fine del 2014. Si prevede per ogni anno un andamento stagionale con picchi nell'ultimo trimestre di ogni anno ed una tendenza leggermente in crescita. Questo significa che il marchio "Diageo Americas" nei prossimi quattro anni manterrà pressoché costante le vendite o saranno in leggero aumento. Queste inoltre sono alte a partire da settembre/ottobre di ogni anno data la stagionalità ed il picco che interessa l'ultima parte dell'anno. Dunque, possiamo affermare che Diageo Americas nei prossimi anni continuerà a confermare la sua posizione.

Una nota importante è che quando si prevede più lontano nel futuro, è naturale essere meno fiduciosi nei valori. Questo si riflette negli intervalli di confidenza generati dal nostro modello, che aumentano man mano che ci spostiamo nel futuro.

8.7 Comparazione dei marchi

Considerando lo strumento di previsione a disposizione, è sicuramente interessante potere confrontare più di un marchio. Questo viene fatto considerando oltre ad “Diageo Americas” anche il secondo marchio in classifica in termini di vendite, “Jim Beam Brands”. Obiettivo di questa sezione è comprendere se il brand che attualmente occupa il secondo posto sul podio sarà destinato a superare il detentore del primo posto, ossia Diageo Americas.

Le seguenti porzioni di codice sono analoghe alla fase di Data preprocessing descritta precedentemente. In sostanza, bisogna fare preprocessing anche per il marchio Jim Beam Brands che vogliamo analizzare.

```
JBB= df.loc[df['Vendor Name'] == 'JIM BEAM BRANDS']
DA = df.loc[df['Vendor Name'] == 'DIAGEO AMERICAS']

DA.drop(cols, axis = 1, inplace = True)
JBB.drop(cols, axis = 1, inplace = True)
DA.isnull().sum()
JBB.isnull().sum()

DA = DA.groupby('Date')['Sale (Dollars)'].sum().reset_index()
JBB = JBB.groupby('Date')['Sale (Dollars)'].sum().reset_index()
DA.set_index('Date')
JBB.set_index('Date')
DA.index
JBB.index

DatetimeIndex(['2012-01-03', '2012-01-04', '2012-01-05', '2012-01-09',
               '2012-01-10', '2012-01-11', '2012-01-12', '2012-01-16',
               '2012-01-17', '2012-01-18',
               ...,
               '2020-11-17', '2020-11-18', '2020-11-19', '2020-11-20',
               '2020-11-23', '2020-11-24', '2020-11-25', '2020-11-27',
               '2020-11-28', '2020-11-30'],
              dtype='datetime64[ns]', name='Date', length=2131, freq=None)
```

Figura 8.13: Preprocessing

```
y_DA = DA['Sale (Dollars)'].resample('MS').mean()
y_JBB = JBB['Sale (Dollars)'].resample('MS').mean()
DA = pd.DataFrame({'Date': y_DA.index, 'Sale (Dollars)': y_DA.values})
JBB = pd.DataFrame({'Date': y_JBB.index, 'Sale (Dollars)': y_JBB.values})

vendor = DA.merge(JBB, how = 'inner', on = 'Date')
vendor.rename(columns = {'Sale (Dollars)_x': 'DIAGEO AMERICAS', 'Sale (Dollars)_y': 'JIM BEAM BRANDS'}, inplace = True)
vendor.head()
```

Figura 8.14: Resample dei dati

A questo punto si ha un unico dataframe contenente informazioni relative alle vendite negli anni per entrambi i marchi, come mostrato nella figura successiva:

	Date	DIAGEO AMERICAS	JIM BEAM BRANDS
0	2012-01-01	182981.287059	82680.970588
1	2012-02-01	203553.833529	82384.854706
2	2012-03-01	204885.607647	77040.698889
3	2012-04-01	224681.446471	89337.144118
4	2012-05-01	224635.786316	87458.327895

Figura 8.15: Dataframe con i due marchi

Il grafico successivo mostra in una stessa finestra l'andamento delle vendite per i due marchi. La legenda mostra che in blu sono rappresentate le vendite di Diageo Americas e in rosso le vendite di Jim Bean Brands. Vediamo che tra le due curve c'è un intervallo abbastanza costante negli anni, dovuto alla differenza delle vendite. Notiamo che se Diageo Americas si mantiene in un intorno di incassi pari a 250.000 dollari, le vendite medie per Jim Bean Brands si mantengono su un intorno di circa 75.000 dollari.

```
plt.figure(figsize=(20,8))
plt.plot(vendor['Date'], vendor['DIAGEO AMERICAS'], 'b-', label = 'DIAGEO AMERICAS')
plt.plot(vendor['Date'], vendor['JIM BEAM BRANDS'], 'r-', label = 'JIM BEAM BRANDS')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Sales of DIAGEO AMERICAS and JIM BEAM BRANDS')
plt.legend()
```

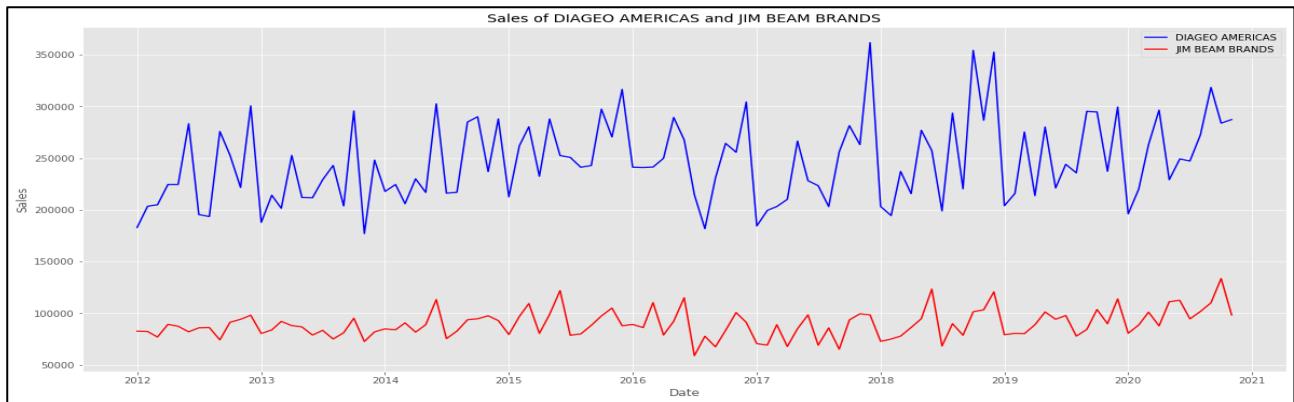


Figura 8.16: Codice e grafico dei due marchi

8.8 Previsioni tramite “Prophet”

Per la previsione dei due marchi si è utilizzata una diversa libreria che viene rilasciata da Facebook ed è chiamata “Prophet”.

Per quanto rilasciato da Facebook, “Prophet è una procedura per la previsione dei dati di serie temporali basata su un modello additivo in cui le tendenze non lineari si adattano alla stagionalità annuale, settimanale e giornaliera, oltre agli effetti delle festività. Funziona meglio con le serie temporali che hanno forti effetti stagionali e diverse stagioni di dati storici. Prophet è resistente ai dati mancanti e ai cambiamenti di tendenza e in genere gestisce bene i valori anomali.”

La nostra idea è quindi sia fare previsioni con un approccio diverso e quindi con una questa nuova libreria, sia poter confrontare le previsioni ottenute con “Statsmodel” e “Prophet”. Inoltre, una grande differenza è che utilizzando il pacchetto Prophet, siamo in grado di generare previsioni accurate senza passare attraverso il “fastidio” di costruire manualmente il modello ARIMA. Questo conferisce, per chi non possiede un “background” tecnico, sicuramente una “Usability” molto elevata.

Dopo avere correttamente installato la libreria, si continua con la scrittura del codice.

Nel codice seguente definiamo i parametri di previsione e quindi gli intervalli di confidenza, il periodo di previsione e la frequenza. Con il metodo “.fit” addestriamo il modello di previsione sui dati di interesse.

```
DA = DA.rename(columns = {'Date': 'ds', 'Sale (Dollars)': 'y'})
DA_model = Prophet(interval_width = 0.95)
DA_model.fit(DA)

JBB = JBB.rename(columns = {'Date': 'ds', 'Sale (Dollars)': 'y'})
JBB_model = Prophet(interval_width = 0.95)
JBB_model.fit(JBB)

DA_forecast = DA_model.make_future_dataframe(periods=36, freq = 'MS') |
DA_forecast = DA_model.predict(DA_forecast)

JBB_forecast = JBB_model.make_future_dataframe(periods = 36, freq = 'MS')
JBB_forecast = JBB_model.predict(JBB_forecast)
```

Figura 8.17: Codice previsioni tramite Prophet

```
plt.figure(figsize=(20,10))
DA_model.plot(DA_forecast, xlabel = 'Date', ylabel = 'Sales')
plt.title('DIAGEO AMERICAS with Forecasts')
```

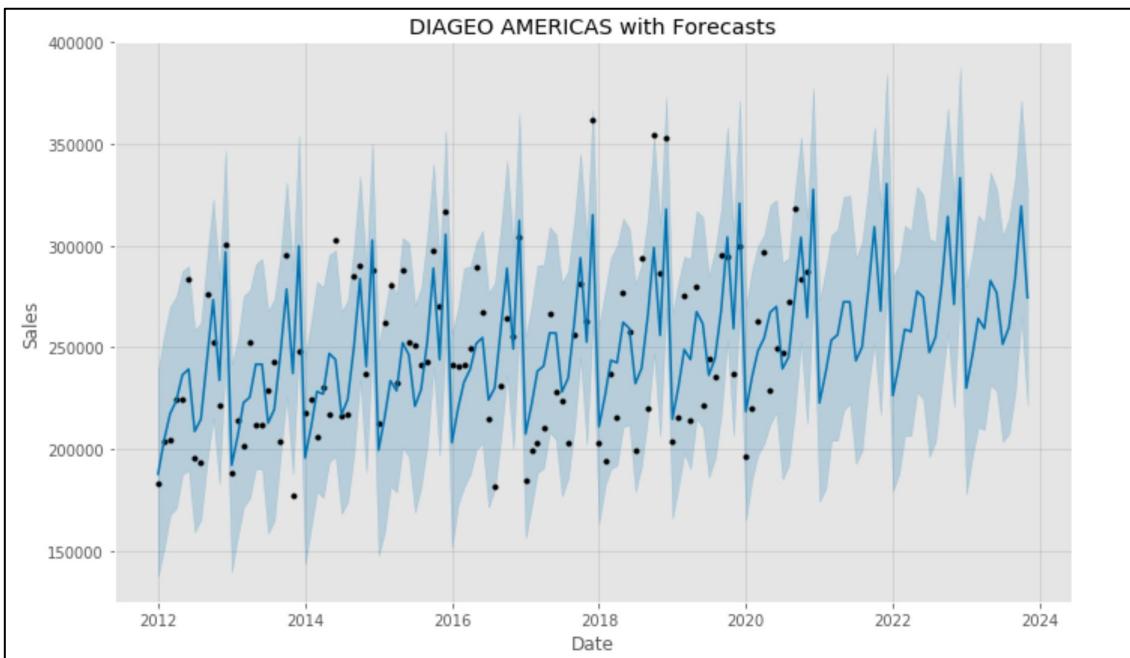


Figura 8.18: Previsioni tramite Prophet per il marchio Diageo Americas

L'output che ne viene fuori è mostrato nella figura precedente, che rappresenta la previsione delle vendite del marchio Diageo Americas nei prossimi tre anni utilizzando la libreria Prophet.

Notiamo come la previsione sia piuttosto simile alla previsione fatta con la modellazione ARIMA e usando la libreria statsmodel. Si registra sempre un andamento stagionale con picchi nell'ultimo trimestre di ogni anno e la tendenza è leggermente in aumento. Quindi anche attraverso l'utilizzo di Prophet possiamo dedurre le stesse valutazioni fatte con statsmodel. Ovviamente un non esperto del settore avrebbe ottenuto gli stessi risultati senza dover effettuare tutta la parte di modellazione ARIMA, quindi sicuramente la facilità di utilizzo e risultati accurati premiano questa libreria.

A questo punto, si mostrano anche le previsioni effettuate per il marchio Jim Beam Brands.

```
plt.figure(figsize=(20,10))
DA_model.plot(DA_forecast, xlabel = 'Date', ylabel = 'Sales')
plt.title('DIAGEO AMERICAS with Forecasts')
```

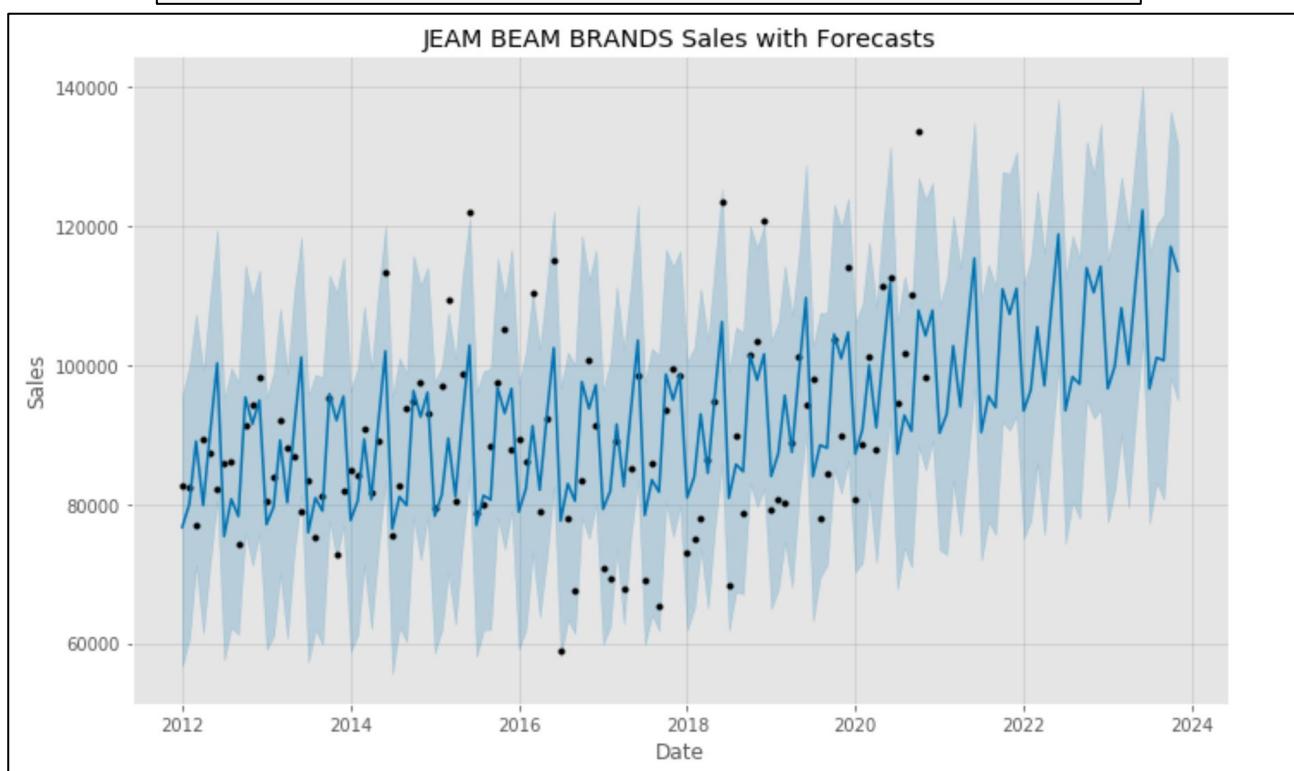


Figura 8.19: Previsioni tramite Prophet per il marchio Jim Beam Brands

Notiamo dal grafico che la previsione per questo marchio mostra sicuramente una crescita e quindi un incremento delle vendite nei prossimi tre anni. Anche per questo marchio si rilevano dei picchi che interessano gli ultimi mesi di ogni anno. Stando alla previsione, possiamo dedurre che nell'ultimo trimestre dell'anno 2023, Jim Beam Brands potrebbe vantare di incassi che raggiungono oltre 120.000 dollari a fronte degli ultimi anni in cui nei migliori periodi si registrano poco più di 100.000 dollari.

8.9 Comparazione delle tendenze

È possibile salvare su un dataframe tutte le informazioni relative alle previsioni effettuate per poi utilizzarle per fare ulteriori valutazioni. Nel nostro caso ci salviamo all'interno del dataframe “forecast” tutte le informazioni di previsione relative ai due brands, come mostrato nel seguente script:

```
DA_names = ['DA_%s' % column for column in DA_forecast.columns]
JBB_names = ['JBB_%s' % column for column in JBB_forecast.columns]

merge_DA_forecast = DA_forecast.copy()
merge_JBB_forecast = JBB_forecast.copy()
merge_DA_forecast.columns = DA_names
merge_JBB_forecast.columns = JBB_names

forecast = pd.merge(merge_DA_forecast, merge_JBB_forecast, how = 'inner', left_on = 'DA_ds', right_on = 'JBB_ds')
forecast = forecast.rename(columns = {'DA_ds': 'Date'}).drop('JBB_ds', axis = 1)
forecast.head()
```

Figura 8.20: Creazione dataframe forecast

Nell'immagine successiva si riporta una porzione del dataframe relativa solo alle prime 5 righe. Inoltre, come si può notare la tabella contiene 31 colonne. Siamo sicuramente interessati ai campi XX_trend, XX_yhat che rappresentano rispettivamente i valori della tendenza e della previsione.

	Date	DA_trend	DA_yhat_lower	DA_yhat_upper	DA_trend_lower	DA_trend_upper	DA_additive_terms	DA_additive_terms_lower	DA_additive_terms_upper
0	2012-01-01	229329.253812	137401.851445	239063.841384	229329.253812	229329.253812	-41599.179767	-41599.179767	-41599.179767
1	2012-02-01	229658.395608	151454.333459	256818.379455	229658.395608	229658.395608	-24837.509458	-24837.509458	-24837.509458
2	2012-03-01	229966.302450	167711.276658	270305.129673	229966.302450	229966.302450	-12522.757274	-12522.757274	-12522.757274
3	2012-04-01	230295.444246	171031.973902	275132.894505	230295.444246	230295.444246	-6438.501713	-6438.501713	-6438.501713
4	2012-05-01	230613.968578	187535.397071	287682.080093	230613.968578	230613.968578	5853.728026	5853.728026	5853.728026

Figura 8.21: Porzione del dataframe forcast

Nel nostro caso DA_trend e DA_yhat sono i campi relativi ai valori della tendenza e della previsione per il marchio Diageo Americas. Analogamente JBB_trend e JBB_yath sono i campi relativi ai valori della tendenza e della previsione per il marchio Jim Beam Brands.

Con il codice successivo andiamo quindi a mostrare le tendenze dei due marchi:

```
plt.figure(figsize = (10,7))
plt.plot(forecast['Date'], forecast['DA_trend'], 'b-')
plt.plot(forecast['Date'], forecast['JBB_trend'], 'r-')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('DIAGEO AMERICAN vs JIM BEAM BRANDS')
plt.legend()
```

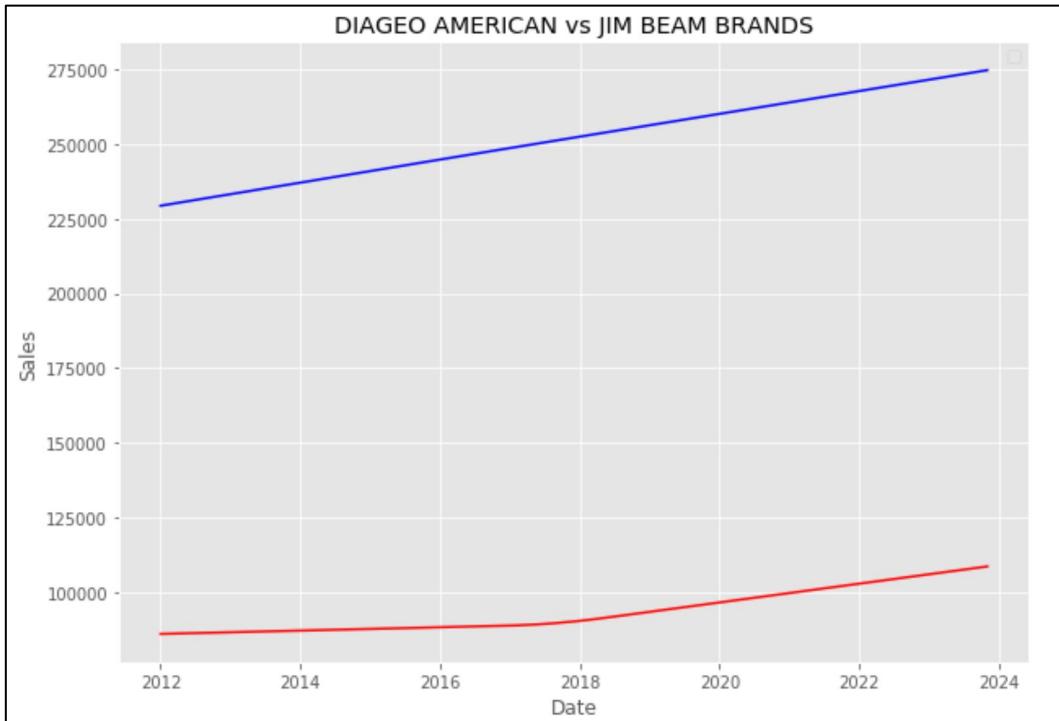


Figura 8.22: Comparazione della tendenza dei due marchi

Nell'immagine precedente vengono mostrate le tendenze per ognuno dei due marchi. Diageo Americas è rappresentata dalla linea blu e Jim Beam Brands dalla linea rossa. Interessante come qui riportiamo le tendenze non solo degli anni relativi alla previsione, ma anche la tendenza relativa ad un ampio orizzonte temporale, a partire dal 2012. Volendo analizzarle in questa finestra temporale possiamo affermare che Diageo Americas ha sempre mostrato una tendenza all'aumento, mentre Jim Beam Brands mostra fino al 2018 un andamento leggermente crescente e, solo dal 2018, si registrano crescite significative destinate ad aumentare nei prossimi anni.

Se l'obiettivo di questa analisi era confrontare i due marchi e individuare se Jim Beam Brands potesse superare Diageo Americas possiamo allora concludere che di certo non si avvicinerà mai ai valori che contraddistinguono Diageo Americas; sicuramente è un marchio che però piace e che manterrà il podio nei prossimi anni aumentando le vendite.

Mantenendo l'idea del voler confrontare i due marchi, nello script seguente andiamo a graficare le previsioni contenute nei campi DA_yhat e JBB_yhat dei due marchi. Se prima abbiamo mostrato la tendenza ora vogliamo mostrare la comparazione tra quelle che sono le previsioni.

```
plt.figure(figsize=(10,7))
plt.plot(forecast['Date'], forecast['DA_yhat'], 'b-')
plt.plot(forecast['Date'], forecast['JBB_yhat'], 'r-')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('DIAGEO AMERICAN vs JIM BEAM BRANDS')
plt.legend()
```

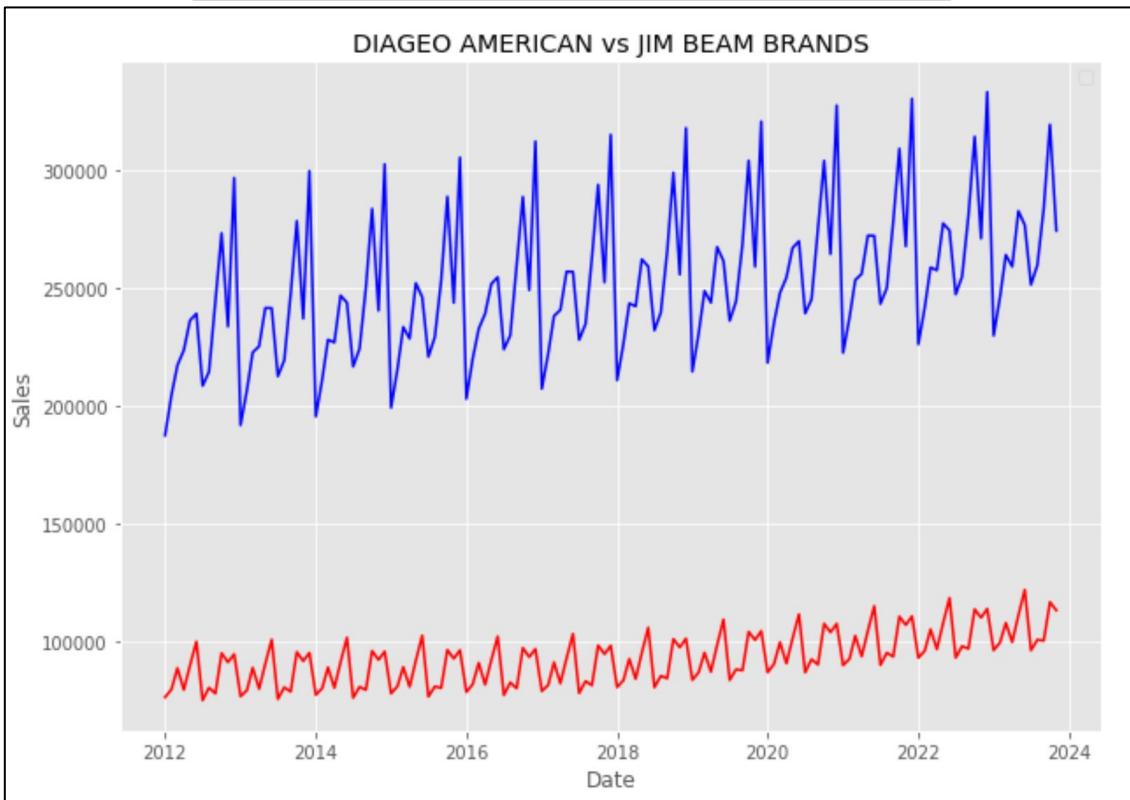


Figura 8.23: Comparazione delle previsioni dei due marchi

L'immagine precedente, quindi, mostra le previsioni delle vendite per i due marchi riguardo tutto l'orizzonte temporale.

8.10 Mercato americano dei liquori

Come ultimo obiettivo di analisi, in questa sezione si va ad analizzare il mercato di liquori nella sua totalità. Questo significa quindi non analizzare singolarmente ogni singolo marchio o più marchi e compararli, ma si vuole prevedere il valore delle vendite di tutti i marchi a partire dalle vendite totali di ogni marchio.

Il codice seguente è essenzialmente preprocessing, come già visto in precedenza, in cui però si vanno a raggruppare le vendite per tutti i marchi. Inoltre, abbiamo una fase di “modeling” in cui si addestra il modello di previsione settando i parametri di previsione.

```
df1.drop(cols, axis = 1, inplace = True)
df1 = df1.groupby('Date')['Sale (Dollars)'].sum().reset_index()
df1 = df1.set_index('Date')
y = df1['Sale (Dollars)'].resample('MS').mean()

df1 = pd.DataFrame({'Date': y.index, 'Sale (Dollars)': y.values})
df1 = df1.rename(columns ={'Date': 'ds', 'Sale (Dollars)': 'y'} )
df1_model = Prophet(interval_width = 0.95)
df1_model.fit(df1)

df1_forecast = df1_model.make_future_dataframe(periods = 36, freq = 'MS')
df1_forecast = df1_model.predict(df1_forecast)

plt.figure(figsize = (18,6))
df1_model.plot(df1_forecast, xlabel = 'Date', ylabel = 'Sales ($)')
plt.title('Liquor Market')
```

Figura 8.24: Preprocessing e previsioni

L’output che ne viene fuori è rappresentato dalla figura seguente:

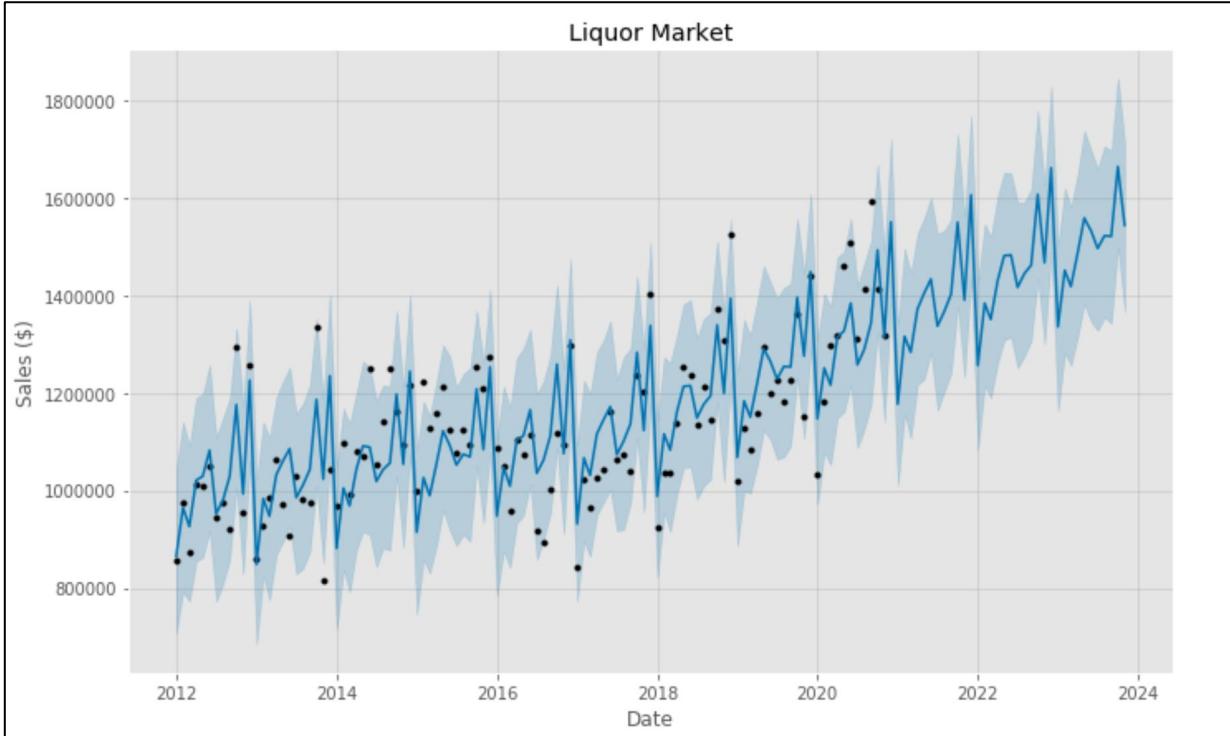


Figura 8.25: Previsione vendite per il mercato di liquori

Dunque, si può concludere che il mercato dei liquori relativo allo stato dell’Iowa è un settore in crescita. Vediamo dalla curva di previsione come i valori predetti delle vendite siano in aumento e con picchi sempre in prossimità dell’ultimo trimestre.

9. Conclusioni

In conclusione, in questo progetto si è analizzato attraverso le principali librerie di Python il dataset “Iowa Liquor Sales”. Sono state effettuate diverse analisi descrittive per comprendere gli eventi passati e capire quali sono ad esempio i maggiori venditori, i marchi più venduti e dunque gli alcolici preferiti; In aggiunta, si è andati alla scoperta delle relazioni e dei pattern che legano particolari variabili, come ad esempio le bottiglie vendute al prezzo, o come siano legati la popolazione ed i profitti. Si sono utilizzate le librerie di Machine Learning per fare analisi predittive, andando quindi ad individuare in quali zone dell’Iowa sarebbe proficuo aprire un nuovo negozio di liquori analizzando anche come le variabili di posizione influenzino le vendite.

Sempre utilizzando le librerie di machine learning si sono analizzate le vendite dei principali marchi concorrenti considerandole come serie temporali e, quindi, si sono effettuate delle analisi temporali prevedendo le vendite future dei brands e stimando il futuro mercato americano di liquori.

Dunque, Python si è rivelato un linguaggio particolarmente potente e soprattutto sembra un buon compromesso: fornisce molte librerie per la Data Science e l’Intelligenza Artificiale più in generale. La prima sensazione che si è avuta è la facilità di utilizzo, la sintassi e i diversi moduli e funzioni che sono già incluse nel linguaggio risultano piuttosto intuitive e facile da imparare. Il principale punto di forza rimane sicuramente la ricchezza delle librerie. Python, infatti, permette di includere le “standard library”, quindi è possibile scaricare e installare moltissimi moduli aggiuntivi già creati e mantenuti dalla comunità in modo da poter estendere le funzionalità possibili.

10. Riferimenti

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

<https://www.ers.usda.gov/data-products/county-level-data-sets/download-data/>

[https://it.wikipedia.org/wiki/Black_Velvet_\(cocktail\)](https://it.wikipedia.org/wiki/Black_Velvet_(cocktail))

https://it.wikipedia.org/wiki/Canadian_whisky

<https://www.diageo.com/en/our-business/where-we-operate/north-america/diageo-north-america/>

<https://lorenzogovoni.com/tre-tecniche-di-regolarizzazione-ridge-lasso-ed-elastic-net/>

<https://facebook.github.io/prophet/>