

ANALYZE AND VISUALIZE LIFE EXPECTANCY WITH BLOOD GROUPS

Submitted By:
Sonakshi Garg
Milind Siddhanti

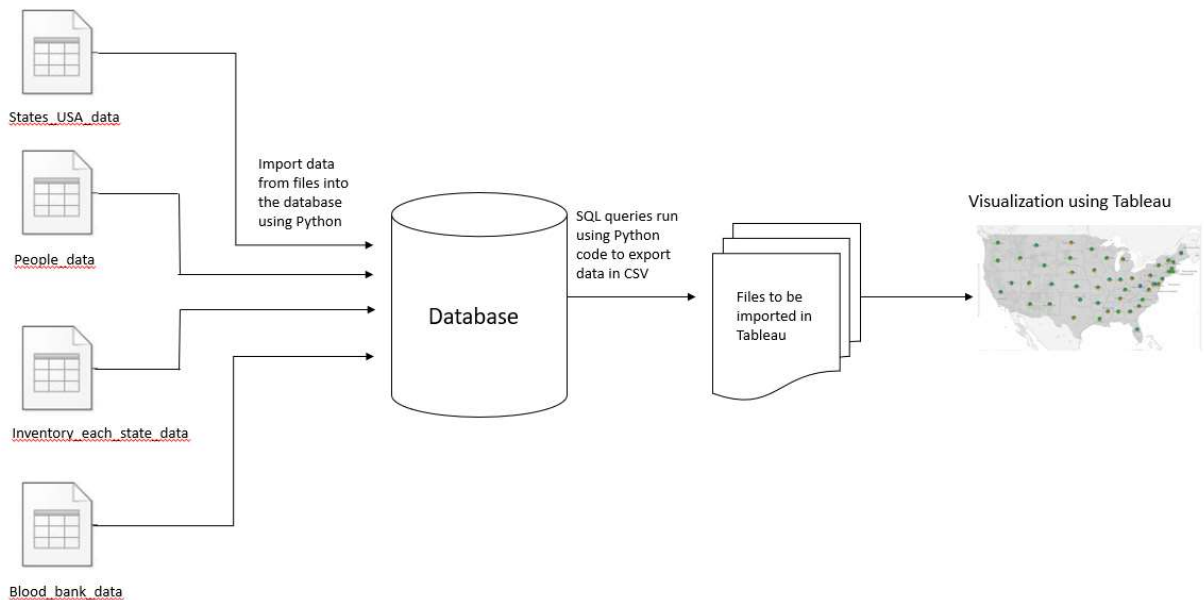
Table of Contents

Project Description	2
Flowchart.....	2
Data Files	2
Database.....	3
SQL Queries	3
Visualization	3
Project Datasets	3
State wise life expectancy	3
People dataset.....	3
Blood Bank dataset.....	4
Inventory dataset	4
Data Wrangling	5
Database Design.....	6
Initial ER Diagram	6
Final ER Diagram.....	7
Relational Vocab.....	7
Sample tables	8
Database Creation.....	9
Database tables screen shots.....	9
Database table structure screen shots.....	11
Data Import to DB code	13
Import States	13
Import People.....	15
Import Blood Banks	17
Database table structure screen shots.....	18
Data Export for analysis code	20
Analysis 1	20
Analysis 2	22
Analysis 3	25
Visualization	28
Life Expectancy of Each Blood Group.....	28
Availability of Blood Type in Every State	29
Blood Type Distribution across states	33
Challenges	36

Project Description:

This Project demonstrates how life expectancy is affected by blood groups in The United States of America. We have four data sets. We have data on different blood banks in each state, population, each type of blood group, life expectancy and data set on people. We have even used test data to analyze the quantity of blood available at different blood banks and queried the database to identify how a blood bank can be contacted incase there is need of blood in a specific state. We used Tableau to visualize our analysis on maps.

Flowchart:



Data Files:

The states table consist of life expectancy in each state. The data source also contains population each state.

People data has names of people residing in different states along with their contact information. Since blood type information is confidential data, it was difficult finding data sources on website that contained these details. Hence, we created test data by random distribution of blood type. We added a new column of “blood_type” to the people table to accomplish this task.

Inventory dataset consists of quantity, blood type and website. We used website column as the joining parameter with the blood bank table to determine which state the inventory field belongs to.

Blood Banks dataset consists of names of blood banks in different state along with their contact information like phone number and website. State_id was used to relate it to the states table.

Database:

We used python scripts to extract data from our four CSV datasets and import the selected information into our database created using PHPMyAdmin.

SQL Queries:

We executed SQL queries to analyze our results and used Python codes to write the results from these SQL queries into CSV which were used as input for Tableau.

Visualization:

For this project, we used Tableau primarily for our final analysis.

Project Datasets:

1. State Wise Life Expectancy:

URL: <http://www.businessinsider.com/us-states-with-the-highest-and-lowest-life-expectancy-2017-5>

This dataset consists of life expectancy in each state arranged in ascending order of life expectancy.

Sample of the data source:



The image shows a screenshot of a Business Insider article. At the top, there is a dark blue header with a white hamburger menu icon and the text "BUSINESS INSIDER". Below the header, a list of US states is displayed, ranked by life expectancy in descending order. The visible portion of the list shows states ranked from 50 down to 42.

50.	Mississippi:	74.91
49.	Alabama:	75.65
48.	Louisiana:	75.82
47.	West Virginia:	76.03
46.	Oklahoma:	76.09
45.	Arkansas:	76.18
44.	Kentucky:	76.26
43.	Tennessee:	76.33
42.	South Carolina:	76.89

2. People Information:

URL: <https://www.briandunning.com/sample-data/>

This dataset consists of detailed information regarding randomly generated people data. To further make this data useable we made sure to assign blood types and date of birth manually and create test data to be able to analyze our results.

Sample of the data source:

	A	B	C	D	E	F	G	H	I	J	
1	first_name	last_name	company_name	address	city	county	state	zip	phone1	phone2	email
2	James	Butt	Benton, John B Jr	6649 N Blue Gum St	New Orleans	Orleans	LA	70116	504-621-8927	504-845-1427	jbutt@gmail.com
3	Josephine	Darakjy	Chanay, Jeffrey A Esq	4 B Blue Ridge Blvd	Brighton	Livingston	MI	48116	810-292-9388	810-374-9840	josephine_darakjy
4	Art	Venere	Chemel, James L Cpa	8 W Cerritos Ave #54	Bridgeport	Gloucester	NJ	8014	856-636-8749	856-264-4130	art@venere.org
5	Lenna	Paprocki	Feltz Printing Service	639 Main St	Anchorage	Anchorage	AK	99501	907-385-4412	907-921-2010	lpaprocki@hotmail.com
6	Donette	Foller	Printing Dimensions	34 Center St	Hamilton	Butler	OH	45011	513-570-1893	513-549-4561	donette.foller@cox.net
7	Simona	Morasca	Chapman, Ross E Esq	3 Mcauley Dr	Ashland	Ashland	OH	44805	419-503-2484	419-800-6759	simona@morasca.com
8	Mitsue	Tollner	Morlong Associates	7 Eads St	Chicago	Cook	IL	60632	773-573-6914	773-924-8565	mitsue_tollner@yahoo.com
9	Leota	Dilliard	Commercial Press	7 W Jackson Blvd	San Jose	Santa Clara	CA	95111	408-752-3500	408-813-1105	leota@hotmail.com
10	Sage	Wieser	Truhlar And Truhlar Attys	5 Boston Ave #88	Sioux Falls	Minnehaha	SD	57105	605-414-2147	605-794-4895	sage_wieser@cox.net
11	Kris	Marrier	King, Christopher A Esq	228 Runamuck Pl #2808	Baltimore	Baltimore City	MD	21224	410-655-8723	410-804-4694	kris@gmail.com
12	Minna	Amigon	Dorl, James J Esq	2371 Jerrold Ave	Kulpsville	Montgomery	PA	19443	215-874-1229	215-422-8694	minna_amigon@yahoo.com
13	Abel	Maclead	Rangoni Of Florence	37275 St Rt 17m M	Middle Island	Suffolk	NY	11953	631-335-3414	631-677-3675	amaclead@gmail.com
14	Kiley	Caldarera	Feiner Bros	25 E 75th St #69	Los Angeles	Los Angeles	CA	90034	310-498-5651	310-254-3084	kiley.caldarera@cox.net
15	Graciela	Ruta	Buckley Miller & Wright	98 Connecticut Ave Nw	Chagrin Falls	Geauga	OH	44023	440-780-8425	440-579-7763	gruta@cox.net
16	Cammy	Albares	Rousseaux, Michael Esq	56 E Morehead St	Laredo	Webb	TX	78045	956-537-6195	956-841-7216	calbares@gmail.com
17	Mattie	Poquette	Century Communications	73 State Road 434 E	Phoenix	Maricopa	AZ	85013	602-277-4385	602-953-6360	mattie@aol.com
18	Meaghan	Garufi	Bolton, Wilbur Esq	69734 E Carrillo St	Mc Minnville	Warren	TN	37110	931-313-9635	931-235-7959	meaghan@hotmail.com
19	Gladys	Rim	T M Byxbee Company Pc	322 New Horizon Blvd	Milwaukee	Milwaukee	WI	53207	414-661-9598	414-377-2880	gladys.rim@rim.com

3. Blood Banks:

URL: https://en.wikipedia.org/wiki/List_of_blood_donation_agencies_in_the_United_States

This dataset consists of all the blood banks available in The United States of America.

The data also contains contact information of the blood banks such as contact number, and website information.

Sample of the data source:

This is a list of organizations by state or territory. ARC and UBS are included in areas where they have donor centers. Some of the names are very similar but refer to different organizations. ^{[6][7]}	
A-H	[edit]
<ul style="list-style-type: none">Alabama<ul style="list-style-type: none">ARCBlood Assurance, Inc.LifeSouth Community Blood CentersUBSAlaska<ul style="list-style-type: none">Blood Bank of AlaskaArizona<ul style="list-style-type: none">ARCUBSArkansas<ul style="list-style-type: none">ARCCommunity Blood Center of the OzarksLifebloodUBS	

4. Inventory Dataset:

URL: <https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/transfusion.data>

This data source consisted of blood donation statistics for March 2007 but to be able to perform our analysis we interpreted this data to be quantity of blood type available.

Sample of the data source:

```
Recency (months),Frequency (times),Monetary (c.c. blood),Time (months),"whether he/she donated blood in March 2007"
2 ,50,12500,98 ,1
0 ,13,3250,28 ,1
1 ,16,4000,35 ,1
2 ,20,5000,45 ,1
1 ,24,6000,77 ,0
4 ,4,1000,4 ,0
2 ,7,1750,14 ,1
1 ,12,3000,35 ,0
2 ,9,2250,22 ,1
5 ,46,11500,98 ,1
4 ,23,5750,58 ,0
0 ,3,750,4 ,0
2 ,10,2500,28 ,1
1 ,13,3250,47 ,0
2 ,6,1500,15 ,1
2 ,5,1250,11 ,1
2 ,14,3500,48 ,1
2 ,15,3750,49 ,1
2 ,6,1500,15 ,1
```

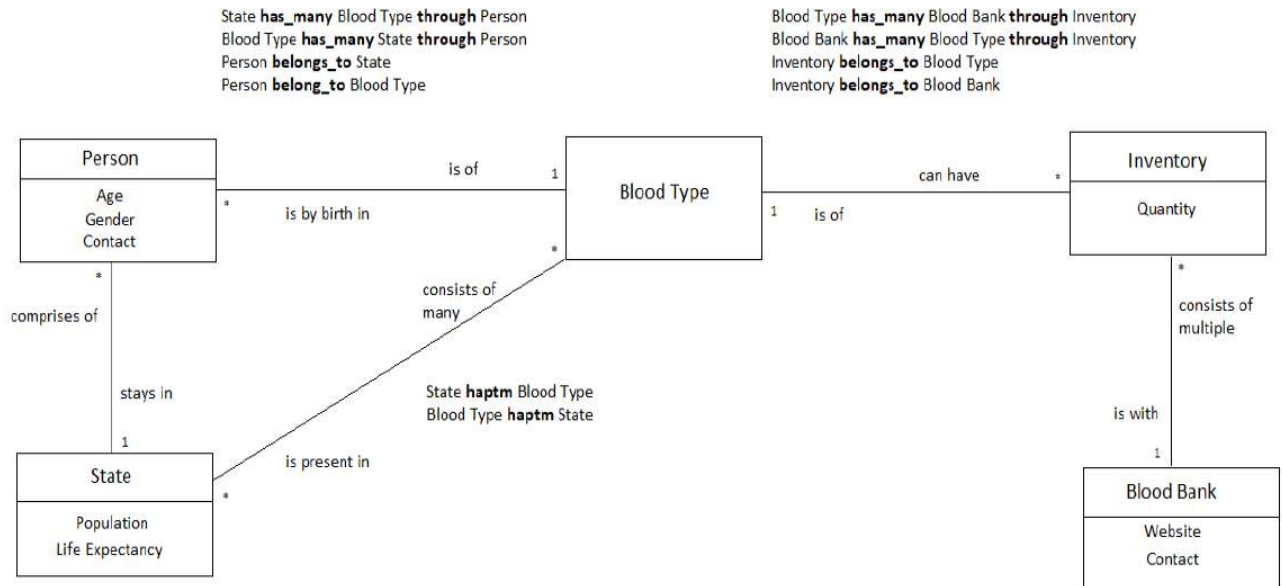
Data Wrangling:

To make the data ready to be uploaded in our database we performed multiple actions which are detailed below:

1. Originally, we used https://en.wikipedia.org/wiki/List_of_U.S._states_and_territories_by_population to create our states table. The table was available in HTML format and to convert it into CSV we used an online platform <http://www.convertcsv.com/html-table-to-csv.htm>. On this website you can separately select the HTML table that you want to convert. It is particularly helpful since we can preview the table we are selecting. It uses the <table> tag to differentiate between tables. We then downloaded and saved the results. This was further modified to add abbreviation and life expectancy in excel using Excel Indexing.
2. We had a PDF of tabular data containing the blood types available but to make this data compatible with other formats we used <https://pdftables.com/> to convert PDF to CSV. We used the converted data to map it with people data and create our test data for analysis.
3. We used Excel functionalities of Merge and Indexing to combine our data.

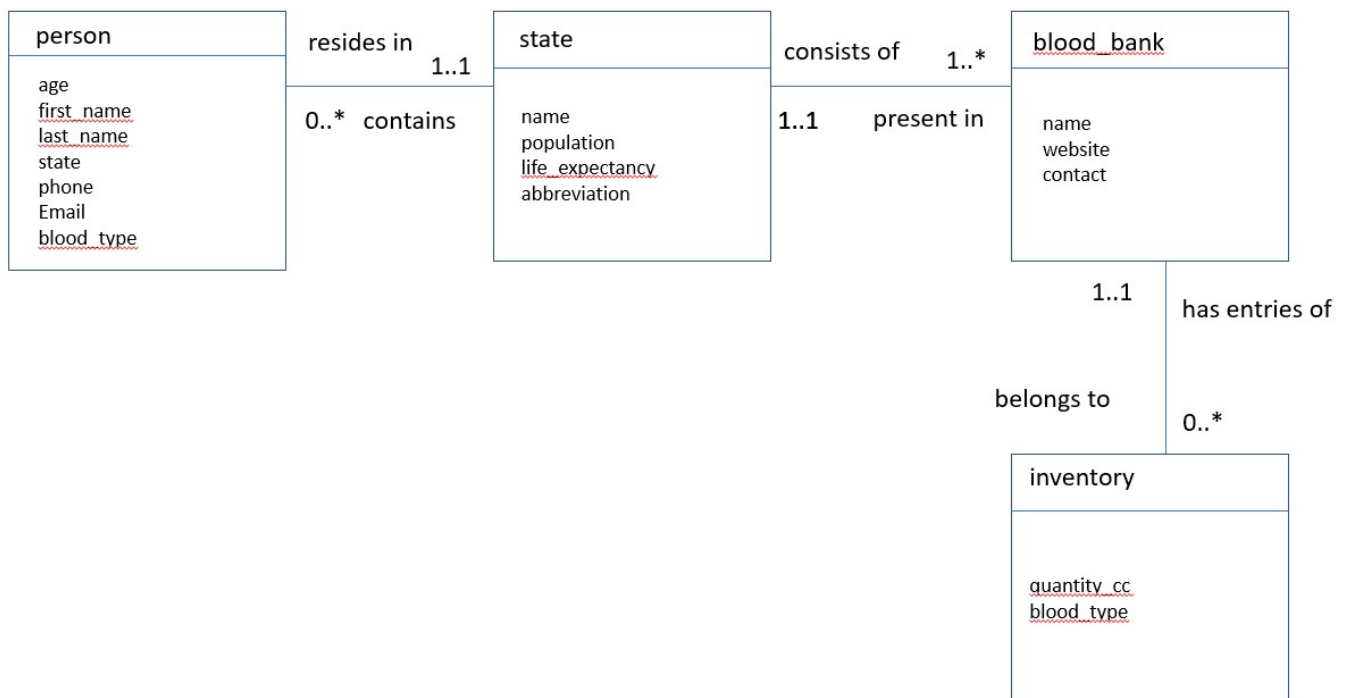
Database Design:

Initial ER Diagram:



Our Initial Entity – Relationship diagram consisted of five entities since we treated each dataset as a separate entity. But as per suggestions we modified our ER diagram which is explained in the next section.

Final ER Diagram:



We simplified our ER diagram further as per the inputs and added blood_type separately to person and inventory data. Our person table relates to state table as a one to many relations. A single person resides in one state, but a state has many people. We joined the person and state data on abbreviation. We added state_id as a foreign key in person table.

State and blood_bank tables also have a one to many relationships. A state can have multiple blood banks but a single blood bank can exist only in one state. To join the tables we used state_id in the blood_bank table.

Blood_bank and inventory table have a one to many relationship. A single inventory field can only belong to one blood bank but multiple inventory entries can exist for a blood_bank. We joined the tables on the website parameter and introduced the blood_bank_id foreign key into the table.

Relational Vocab:

1. State has_many person
Person belongs to state
2. State has_many blood_bank
Blood_bank belongs to state
3. Blood_bank has_many inventory
Inventory belongs to blood_bank

Sample Tables:

People:

id	first_name	last_name	state_id	phone	email	blood_type	dob
1	James	Butt	18	504-621-8927	jbutt@gmail.com	O+	11/30/1961
2	Josephine	Darakjy	22	810-292-9388	josephine_darakjy@darakjy.org	O-	4/25/1962
3	Art	Venere	30	856-636-8749	art@venere.org	A+	10/5/1963
4	Lenna	Paprocki	2	907-385-4412	lpaprocki@hotmail.com	A-	10/28/1963
5	Donette	Foller	35	513-570-1893	donette.foller@cox.net	B+	9/18/1964
6	Simona	Morasca	35	419-503-2484	simona@morasca.com	B-	7/21/1965
7	Mitsue	Tollner	13	773-573-6914	mitsue_tollner@yahoo.com	AB+	4/13/1966
8	Leota	Dilliard	5	408-752-3500	leota@hotmail.com	AB-	8/25/1967
9	Sage	Wieser	41	605-414-2147	sage_wieser@cox.net	O+	8/25/1968
10	Kris	Marrier	20	410-655-8723	kris@gmail.com	O-	10/11/1969
11	Minna	Amigon	38	215-874-1229	minna_amigon@yahoo.com	A+	2/3/1970
12	Abel	Maclead	32	631-335-3414	amaclead@gmail.com	A-	4/15/1971
13	Kiley	Caldarera	5	310-498-5651	kiley.caldarera@aol.com	B+	5/15/1971
14	Graciela	Ruta	35	440-780-8425	gruta@cox.net	B-	9/7/1971
15	Cammy	Albares	43	956-537-6195	calbares@gmail.com	AB+	9/14/1971
16	Mattie	Poquette	3	602-277-4385	mattie@aol.com	AB-	5/22/1972
17	Meaghan	Garufi	42	931-313-9635	meaghan@hotmail.com	O+	3/20/1974
18	Gladys	Rim	49	414-661-9598	gladys.rim@rim.org	O-	4/21/1974

States:

id	name	population	life_expectancy	abbreviation
1	Alabama	4863300	76	AL
2	Alaska	741894	78	AK
3	Arizona	6931071	80	AZ
4	Arkansas	2988248	76	AR
5	California	39268294	81	CA
6	Colorado	5540545	80	CO
7	Connecticut	3576452	81	CT
8	Delaware	952065	79	DE
9	Florida	20612439	79	FL
10	Georgia	10310371	77	GA
11	Hawaii	1428557	81	HI
12	Idaho	1683140	79	ID
13	Illinois	12801539	79	IL
14	Indiana	6633053	78	IN
15	Iowa	3134693	80	IA
16	Kansas	2907289	79	KS
17	Kentucky	4436974	76	KY
18	Louisiana	4681666	76	LA

Blood banks:

id	name	website	contact	state_id
1	ARC	www.al-arc.com	2147483647	1
2	Blood Assurance Inc.	www.al-bainc.com	2147483647	1
3	LifeSouth Community Blood Centers	www.al-lcbc.com	2147483647	1
4	UBS	www.al-ubs.com	2147483647	1
5	Blood Bank of Alaska	www.ak-bba.com	2147483647	2
6	ARC	www.az-arc.com	2147483647	3
7	UBS	www.az-ubs.com	2147483647	3

Inventories:








































id	quantity_cc	blood_type	blood_bank_id
1	12500	A-	1
2	3250	O+	2
3	4000	AB+	3
4	5000	B+	4
5	6000	A+	5
6	1000	AB-	6
7	1750	O-	7
8	3000	B-	8
9	2250	B+	9
10	11500	A+	10
11	5750	O+	11
12	750	B+	12
13	2500	A-	13
14	3250	O+	14
15	1500	AB+	15
16	1250	B+	16
17	3500	A+	17
18	3750	AB-	18

Database Creation:






































The database was created using PHPMYAdmin.

Database table screen shots:





































People:

+ Options														
<div><div>←</div><div>↕</div><div>→</div></div>			<div>▼</div>	id	first_name	last_name	state_id	phone	email	blood_type	dob			
<input type="checkbox"/>		Edit		Copy		Delete	1	James	Butt	18	504-621-8927	jbutt@gmail.com	O+	1961-11-30
<input type="checkbox"/>		Edit		Copy		Delete	2	Josephine	Darakjy	22	810-292-9388	josephine_darakjy@darakjy.org	O-	1962-04-25
<input type="checkbox"/>		Edit		Copy		Delete	3	Art	Venere	30	856-636-8749	art@venere.org	A+	1963-10-05
<input type="checkbox"/>		Edit		Copy		Delete	4	Lenna	Paprocki	2	907-385-4412	lpaprocki@hotmail.com	A-	1963-10-28
<input type="checkbox"/>		Edit		Copy		Delete	5	Donette	Foller	35	513-570-1893	donette.foller@cox.net	B+	1964-09-18
<input type="checkbox"/>		Edit		Copy		Delete	6	Simona	Morasca	35	419-503-2484	simona@morasca.com	B-	1965-07-21
<input type="checkbox"/>		Edit		Copy		Delete	7	Mitsue	Tollner	13	773-573-6914	mitsue_tollner@yahoo.com	AB+	1966-04-13
<input type="checkbox"/>		Edit		Copy		Delete	8	Leota	Dilliard	5	408-752-3500	leota@hotmail.com	AB-	1967-08-25
<input type="checkbox"/>		Edit		Copy		Delete	9	Sage	Wieser	41	605-414-2147	sage_wieser@cox.net	O+	1968-08-25
<input type="checkbox"/>		Edit		Copy		Delete	10	Kris	Marrier	20	410-655-8723	kris@gmail.com	O-	1969-10-11
<input type="checkbox"/>		Edit		Copy		Delete	11	Minna	Amigon	38	215-874-1229	minna_amigon@yahoo.com	A+	1970-02-03
<input type="checkbox"/>		Edit		Copy		Delete	12	Abel	Maclead	32	631-335-3414	amaclead@gmail.com	A-	1971-04-15
<input type="checkbox"/>		Edit		Copy		Delete	13	Kiley	Caldarera	5	310-498-5651	kiley.caldarera@aol.com	B+	1971-05-15

States:

			id	name	population	life_expectancy	abbreviation
<input type="checkbox"/>		Edit		Copy		Delete	1 Alabama 4863300 76 AL
<input type="checkbox"/>		Edit		Copy		Delete	2 Alaska 741894 78 AK
<input type="checkbox"/>		Edit		Copy		Delete	3 Arizona 6931071 80 AZ
<input type="checkbox"/>		Edit		Copy		Delete	4 Arkansas 2988248 76 AR
<input type="checkbox"/>		Edit		Copy		Delete	5 California 39268294 81 CA
<input type="checkbox"/>		Edit		Copy		Delete	6 Colorado 5540545 80 CO
<input type="checkbox"/>		Edit		Copy		Delete	7 Connecticut 3576452 81 CT
<input type="checkbox"/>		Edit		Copy		Delete	8 Delaware 952065 79 DE
<input type="checkbox"/>		Edit		Copy		Delete	9 Florida 20612439 79 FL
<input type="checkbox"/>		Edit		Copy		Delete	10 Georgia 10310371 77 GA
<input type="checkbox"/>		Edit		Copy		Delete	11 Hawaii 1428557 81 HI
<input type="checkbox"/>		Edit		Copy		Delete	12 Idaho 1683140 79 ID

Blood_Banks:


			id	name	website	contact	state_id
<input type="checkbox"/>		Edit		Copy		Delete	1 ARC www.al-arc.com 2147483647 1
<input type="checkbox"/>		Edit		Copy		Delete	2 Blood Assurance Inc. www.al-bainc.com 2147483647 1
<input type="checkbox"/>		Edit		Copy		Delete	3 LifeSouth Community Blood Centers www.al-lcbc.com 2147483647 1
<input type="checkbox"/>		Edit		Copy		Delete	4 UBS www.al-ubs.com 2147483647 1
<input type="checkbox"/>		Edit		Copy		Delete	5 Blood Bank of Alaska www.ak-bba.com 2147483647 2
<input type="checkbox"/>		Edit		Copy		Delete	6 ARC www.az-arc.com 2147483647 3
<input type="checkbox"/>		Edit		Copy		Delete	7 UBS www.az-ubs.com 2147483647 3
<input type="checkbox"/>		Edit		Copy		Delete	8 ARC www.ar-arc.com 2147483647 4
<input type="checkbox"/>		Edit		Copy		Delete	9 Community Blood Center of the Ozarks www.ar-cbco.com 1828264356 4
<input type="checkbox"/>		Edit		Copy		Delete	10 Lifeblood www.ar-life.com 2147483647 4
<input type="checkbox"/>		Edit		Copy		Delete	11 UBS www.ar-ubs.com 1372682995 4
<input type="checkbox"/>		Edit		Copy		Delete	12 ARC www.ca-arc.com 2147483647 5
<input type="checkbox"/>		Edit		Copy		Delete	13 Lifestream www.ca-ls.com 2147483647 5

Inventories:


<div><div><div><div></div><div></div><div></div></div><div><div></div><div></div><div></div></div></div><div></div></div>				id	quantity_cc	blood_type	blood_bank_id
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	1	12500	A-	1
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	2	3250	O+	2
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	3	4000	AB+	3
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	4	5000	B+	4
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	5	6000	A+	5
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	6	1000	AB-	6
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	7	1750	O-	7
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	8	3000	B-	8
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	9	2250	B+	9
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	10	11500	A+	10
<div><div><div></div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	<div><div><div></div><div></div><div></div></div><div></div></div>	11	5750	O+	11

Database table structure screen shots:


People:

	#	Name	Type	Collation	Attributes	Null	Default	Extra
<input type="checkbox"/>	1	id 	int(11)			No	None	AUTO_INCREMENT
<input type="checkbox"/>	2	first_name	text	latin1_swedish_ci		No	None	
<input type="checkbox"/>	3	last_name	text	latin1_swedish_ci		No	None	
<input type="checkbox"/>	4	state_id	int(11)			No	None	
<input type="checkbox"/>	5	phone	text	latin1_swedish_ci		Yes	NULL	
<input type="checkbox"/>	6	email	text	latin1_swedish_ci		No	None	
<input type="checkbox"/>	7	blood_type	text	latin1_swedish_ci		No	None	
<input type="checkbox"/>	8	dob	date			Yes	NULL	


States:

	#	Name	Type	Collation	Attributes	Null	Default	Extra
<input type="checkbox"/>	1	id 	int(11)			No	None	AUTO_INCREMENT
<input type="checkbox"/>	2	name	text	latin1_swedish_ci		No	None	
<input type="checkbox"/>	3	population	int(11)			No	None	
<input type="checkbox"/>	4	life_expectancy	decimal(10,0)			No	None	
<input type="checkbox"/>	5	abbreviation	text	latin1_swedish_ci		No	None	

Blood Banks:

	#	Name	Type	Collation	Attributes	Null	Default	Extra
<input type="checkbox"/>	1	id 	int(11)			No	None	AUTO_INCREMENT
<input type="checkbox"/>	2	name	text	latin1_swedish_ci		No	None	
<input type="checkbox"/>	3	website	text	latin1_swedish_ci		No	None	
<input type="checkbox"/>	4	contact	int(11)			No	None	
<input type="checkbox"/>	5	state_id	int(11)			No	None	

Inventories:

	#	Name	Type	Collation	Attributes	Null	Default	Extra
<input type="checkbox"/>	1	id 	int(11)			No	None	AUTO_INCREMENT
<input type="checkbox"/>	2	quantity_cc	int(11)			No	None	
<input type="checkbox"/>	3	blood_type	text	latin1_swedish_ci		No	None	
<input type="checkbox"/>	4	blood_bank_id	int(11)			No	None	

Data Import to Database from CSV using Python:

Code path: ~/home/milind_siddhanti/project/FinalProject

- 1.) Import states data from states.csv using states.py
 - i) Database connection is set up to insert the data.
 - ii) Since the state names were a combination of name and abbreviation, split function was used to store state name in a temporary variable and abbreviation in another temporary variable.
 - iii) Parameterized query was used to insert data into the database table of states.

Code to import data from states.csv is as below:

```
1. # importing relevant libraries for the python code to execute the functionalities.
2. import pymysql.cursors
3. import pprint
4. import csv
5.
6. # Set up connection with database using the connection variable below
7. connection = pymysql.connect(
8.     host="localhost", # server details
9.     user="milind_siddhanti", # username for authentication
10.    passwd="nokiayu7k", # password to connect with the database
11.    db="milind_siddhanti_project", # schema name of the database
12.    autocommit=True, # executes commit functionality in database
13.    cursorclass=pymysql.cursors.DictCursor # python libraries
14. )
15.
16. # with successful database connection, feeding data to the database
17. with connection.cursor() as cursor:
18.
19.     # database table should be created in the database schema using phpMyAdmin
20.     # choose the columns from the database table into which the data should be fed into.
21.     # using the SQL INSERT functionality, insert the data into the database table columns as required.
22.     # with the help of defining datatypes in python, for ex: %(name)s as string data from the csv is read as string from the respective csv
23.
24.     # inserting data into the database table columns with the below query
25.
26.     state_sql = """INSERT INTO states(name,population,life_expectancy,abbreviation)
27.     VALUE (%(name)s,%(population)s,%(life_expectancy)s,%(abbreviation)s)"""
28.
29.     # open the csv file and read the data from the csv file.
30.     with open('states.csv') as csvfile:
31.         # specifying the type of csv format used so that python knows and can format the file.
32.         myCSVReader = csv.DictReader(csvfile, delimiter=",", quotechar='"')
33.
```

```

34.         # reading each and every row from the csv to validate the data, make c
    orrections to the data,
35.         # and store the data in a temporary variable.
36.         for row in myCSVReader:
37.
38.             # storing the value of the column "name" in a temporary string var
    iable. Ex: state_name_spilt = "AlabamaAL"
39.             state_name_spilt = row['state_name_combined']
40.             # spilting the last 2 characters of the string "state_name_spilt"
    and storing in another temporary string variable as state_name. Ex: state_name
    = "Alabama"
41.             state_name = state_name_spilt[:-2]
42.             # reading the last 2 characters of the string "state_name_spilt" a
    nd storing in new temporary string variable, "state_abb". Ex: state_abb = "AL"
43.             state_abb = state_name_spilt[-2:]
44.
45.             # storing changes made to the data in a dictionary as a key, value
    pair to insert the validated and corrected data into the database schema tabl
    e.
46.             # The key represents the datatase column name so that the data sit
    s in the required database column.
47.             # The value represents the corrected data that has to be fed into
    the database columns.
48.             # creating dictionary, "my_dict" with key, value pairs.
49.             my_dict = {'name': state_name,
50.                        'population': row['population'],
51.                        'life_expectancy': row['life_expectancy'],
52.                        'abbreviation': state_abb
53.                       }
54.
55.             # using the functions from the python libraries, reading data from
    the temporary dictionary which was intially read from the csv,
56.             # and executing the sql query which inserts the data into the data
    base columns.
57.             cursor.execute(state_sql, my_dict)

```

- 2.) Import people data from people.csv using people.py
 - i) Database connection is set up to insert the data.
 - ii) To determine the state of people, data in people table had a foreign key association with states table. To fetch details of states, a sql query to fetch the states' ids was executed and the results were stored in a temporary list. If there was no state found for a person, then null value would be stored against that person details.
 - iii) Date function "strptime" was implemented with a template to store date of birth of people.
 - iv) Finally, a parameterized sql query was used to insert data from people.csv to database table of people.

Code to import data from people.csv is as below:

```
1. # importing relevant libraries for the python code to execute the functionalities.
2. import pymysql.cursors
3. import pprint
4. import csv
5. from datetime import datetime
6.
7. # Set up connection with database using the connection variable below
8. connection = pymysql.connect(
9.     host="localhost", # server details
10.    user="milind_siddhanti", # username for authentication
11.    passwd="nokiayu7k", # password to connect with the database
12.    db="milind_siddhanti_project", # schema name of the database
13.    autocommit=True, # executes commit functionality in database
14.    cursorclass=pymysql.cursors.DictCursor # python libraries
15. )
16.
17. # with successful database connection, feeding data to the database
18. with connection.cursor() as cursor:
19.
20.     # database table should be created in the database schema using phpMyAdmin.
21.     # choose the columns from the database table into which the data should be fed into
22.     # using the SQL INSERT functionality, insert the data into the database table columns as required.
23.     # with the help of defining datatypes in python, for ex: %(first_name)s as string, data from the csv is read as string from the respective csv
24.
25.     # SELECT statement to fetch the state ids - to generate a connection between the database tables using the primary key and foreign key relation
26.     select_state = "SELECT id from states WHERE abbreviation = %(state)s"
27.
28.     # inserting data into the database table columns with the below query
29.     insert_people = """INSERT INTO people(first_name,last_name,state_id,phone,email,blood_type,dob)
30.         VALUE (%(first_name)s,%(last_name)s,%(state_id)s,%(phone)s,%(email)s,%(blood_type)s,%(dob)s)"""
31.
32.     # open the csv file and read the data from the csv file.
33.     with open('people.csv') as csvfile:
34.         # specifying the type of csv format used so that python knows and can format the file.
35.         myCSVReader = csv.DictReader(csvfile, delimiter=",", quotechar='"')
36.
```



```

37.         # reading each and every row from the csv to validate the data, make correction
           s to the data,
38.         # and store the data in a temporary variable.
39.         for row in myCSVReader:
40.             # reading the dates from the csv and storing in the database column with da
               te as the datatype. Stripping the date function using "STR"ing + "P"arse + "T"ime funct
               ion
41.             # and the template in which the date is stored in the csv file.
42.             date = row['dob']
43.             template = "%m/%d/%Y"
44.             date_of_birth = datetime.strptime(date, template)
45.
46.             # executing the SELECT query to fetch all the states id from the states dat
               abase table which was created earlier.
47.             cursor.execute(select_state, row)
48.             # storing the results in a list. Using fetchone() function since each query
               executed fetches single record. For better performance using the fetchone() functional
               ity.
49.             results = cursor.fetchone()
50.             # eliminating null values
51.             if(results != ()):
52.                 state_id = results['id']
53.
54.             # storing changes made to the data in a dictionary as a key, value pair to
               insert the validated and corrected data into the database schema table.
55.             # The key represents the datatase column name so that the data sits in the
               required database column.
56.             # The value represents the corrected data that has to be fed into the datab
               ase columns.
57.             # creating dictionary, "my_dict" with key, value pairs.
58.             my_dict = {'state_id': state_id,
59.                        'first_name': row['first_name'],
60.                        'last_name': row['last_name'],
61.                        'phone': row['phone'],
62.                        'email': row['email'],
63.                        'blood_type': row['blood_type'],
64.                        'dob' : date_of_birth
65.                       }
66.
67.             # using the functions from the python libraries, reading data from the temp
               orary dictionary which was intially read from the csv,
68.             # and executing the sql query which inserts the data into the database colu
               mns.
69.             cursor.execute(insert_people, my_dict)

```

- 3.) Import blood banks data from blood_banks.csv using blood_banks.py
 - i) Database connection is set up to insert the data.
 - ii) To deduce where the blood banks are located, code is structured to find the states' ids with respect to the blood banks location and store the retrieved data in a temporary list. If state id was blank, a null value would be inserted in the list.
 - iii) Later, a parameterized query with all the values in place will be executed to insert data from the csv file to the database table of blood_banks.

Code to insert data into blood_banks table from blood_banks.csv using blood_banks.py

```
1. # importing relevant libraries for the python code to execute the functionalities.
2. import pymysql.cursors
3. import pprint
4. import csv
5.
6. # Set up connection with database using the connection variable below
7. connection = pymysql.connect(
8.     host="localhost", # server details
9.     user="milind_siddhanti", # username for authentication
10.    passwd="nokiayu7k", # password to connect with the database
11.    db="milind_siddhanti_project", # schema name of the database
12.    autocommit=True, # executes commit functionality in database
13.    cursorclass=pymysql.cursors.DictCursor # python libraries
14. )
15.
16. # with successful database connection, feeding data to the database
17. with connection.cursor() as cursor:
18.
19.     # database table should be created in the database schema using phpMyAdmin.
20.     # choose the columns from the database table into which the data should be fed into
21.     # using the SQL INSERT functionality, insert the data into the database table columns as required.
22.     # with the help of defining datatypes in python, for ex: %(first_name)s as string, data from the csv is read as string from the respective csv
23.
24.     # SELECT statement to fetch the state ids - to generate a connection between the database tables using the primary key and foreign key relation
25.     select_state = "SELECT id from states WHERE name = %(state)s"
26.
27.     # inserting data into the database table columns with the below query
28.     insert_blood_bank = """INSERT INTO blood_banks(state_id,name,website,contact)
29.         VALUE (%(state_id)s, %(name)s, %(website)s, %(contact)s)"""
30.
31.     # open the csv file and read the data from the csv file.
32.     with open('blood_bank.csv') as csvfile:
33.         # specifying the type of csv format used so that python knows and can format the file.
34.         myCSVReader = csv.DictReader(csvfile, delimiter=",", quotechar='"')
35.
36.         # reading each and every row from the csv to validate the data, make corrections to the data,
37.         # and store the data in a temporary variable.
38.         for row in myCSVReader:
39.
40.             # executing the SELECT query to fetch all the states id from the states database table which was created earlier.
41.             cursor.execute(select_state, row)
```

```

42.         # storing the results in a list. Using fetchone() function since each query
        executed fetches single record. For better performance using the fetchone() functional
        ity.
43.         results = cursor.fetchone()
44.         # eliminating null values
45.         if(results != ()):
46.             state_id = results['id']
47.
48.         # storing changes made to the data in a dictionary as a key, value pair to
        insert the validated and corrected data into the database schema table.
49.         # The key represents the database column name so that the data sits in the
        required database column.
50.         # The value represents the corrected data that has to be fed into the datab
        ase columns.
51.         # creating dictionary, "my_dict" with key, value pairs.
52.         my_dict = {'state_id': state_id,
53.                    'name': row['name'],
54.                    'website': row['website'],
55.                    'contact': row['contact']}
56.
57.         # using the functions from the python libraries, reading data from the temp
        orary dictionary which was initially read from the csv,
58.         # and executing the sql query which inserts the data into the database colu
        mns.
59.         cursor.execute(insert_blood_bank, my_dict)

```

- 4.) Import inventories data from inventories.csv using inventories.py
 - i. Database connection is set up to insert the data.
 - ii. Finding the blood bank's id to link with the inventory details. Each blood bank has a definitive quantity of blood stored in it and the details will be available in the inventory table.
 - iii. Later, a parameterized query with all the values in place will be executed to insert data from the csv file to the database table of inventories.

Code to insert data into inventories table from inventories.csv using inventories.py

```

1. # importing relevant libraries for the python code to execute the functionalities.
2. import pymysql.cursors
3. import pprint
4. import csv
5.
6. # Set up connection with database using the connection variable below
7. connection = pymysql.connect(
8.     host="localhost", # server details
9.     user="milind_siddhanti", # username for authentication
10.    passwd="nokiayu7k", # password to connect with the database
11.    db="milind_siddhanti_project", # schema name of the database
12.    autocommit=True, # executes commit functionality in database
13.    cursorclass=pymysql.cursors.DictCursor # python libraries
14. )
15.
16. # with successful database connection, feeding data to the database
17. with connection.cursor() as cursor:
18.
19.     # database table should be created in the database schema using phpMyAdmin.

```

```

20.     # choose the columns from the database table into which the data should be fed into
21.     # using the SQL INSERT functionality, insert the data into the database table columns as required.
22.     # with the help of defining datatypes in python, for ex: %(first_name)s as string,
    data from the csv is read as string from the respective csv
23.
24.     # SELECT statement to fetch the blood bank ids - to generate a connection between the
    database tables using the primary key and foreign key relation
25.     select_blood_bank = "SELECT id from blood_banks WHERE website = %(website)s"
26.
27.     # inserting data into the database table columns with the below query
28.     insert_inventories = """INSERT INTO inventories(quantity_cc,blood_type,blood_bank_id)
    VALUE (%(quantity_cc)s, %(blood_type)s, %(blood_bank_id)s)"""
29.
30.
31.     # open the csv file and read the data from the csv file.
32.     with open('inventories.csv') as csvfile:
33.         # specifying the type of csv format used so that python knows and can format the
    file.
34.         myCSVReader = csv.DictReader(csvfile, delimiter=",", quotechar='"')
35.
36.         # reading each and every row from the csv to validate the data, make corrections
    to the data,
37.         # and store the data in a temporary variable.
38.         for row in myCSVReader:
39.
40.             # executing the SELECT query to fetch all the states id from the states database
    table which was created earlier.
41.             cursor.execute(select_blood_bank, row)
42.             # storing the results in a list. Using fetchone() function since each query
    executed fetches single record. For better performance using the fetchone() functionality.
43.             results = cursor.fetchone()
44.             # eliminating null values
45.             if(results!=()):
46.                 blood_bank_id = results['id']
47.
48.             # storing changes made to the data in a dictionary as a key, value pair to
    insert the validated and corrected data into the database schema table.
49.             # The key represents the database column name so that the data sits in the
    required database column.
50.             # The value represents the corrected data that has to be fed into the database
    columns.
51.             # creating dictionary, "my_dict" with key, value pairs.
52.             my_dict = {'blood_bank_id': blood_bank_id,
53.                        'quantity_cc': row['quantity_cc'],
54.                        'blood_type': row['blood_type']}
55.
56.             # using the functions from the python libraries, reading data from the temporary
    dictionary which was initially read from the csv,
57.             # and executing the sql query which inserts the data into the database columns.
58.             cursor.execute(insert_inventories, my_dict)

```

Data Export from Database using Python for Analysis:

Code Path: ~/home/milind_siddhanti/project/FinalProject

- **Analysis 1 -**

- 1) Build of the query is defined at the start of code.
- 2) Database connection is set up for the execution of queries.
- 3) Column names are fetched of the query and store in a temporary variable. This is used as headers for the csv file.
- 4) The query executed will fetch details of number of people with respective blood type residing in every state.
- 5) Using the dictwriter, writing the results into the csv mentioned in function.
- 6) Each report is stored in a csv format which can be further utilized for analysis purpose using Tableau.
- 7) This query gives the count of number of people belonging to every blood type in each state.

```
1. # Read the person details from the people table.
2. # SELECT *
3. # FROM people
4. #
5. # Join states table to the people table based on the foriegn keys.
6. # SELECT *
7. # FROM people
8. #     JOIN states
9. #         ON states.id = people.state_id
10. #
11. # Group people by the states they live are living in.
12. # SELECT *
13. # FROM people
14. #     JOIN states
15. #         ON states.id = people.state_id
16. # GROUP BY states.id
17. #
18. # Also, group people by the blood types to figure out statistics.
19. # SELECT *
20. # FROM people
21. #     JOIN states
22. #         ON states.id = people.state_id
23. # GROUP BY states.id, people.blood_type
24. #
25. # Select columns to export the information required, count of people, the blood type and states.
26. # SELECT states.name, people.blood_type, COUNT(*) AS Count_of_People_BloodType_in_every_State
27. # FROM people
28. #     JOIN states
29. #         ON states.id = people.state_id
30. # GROUP BY states.id, people.blood_type
31.
32. # import the relevant python libraries to use the functionalities.
33. import pymysql.cursors
34. import pprint
35. import csv
36.
37. # Set up connection with database using the connection variable below
38. connection = pymysql.connect(
```

```

39.         host="localhost", # server details
40.         user="milind_siddhanti", # username for authentication
41.         passwd="nokiayu7k", # password to connect with the database
42.         db="milind_siddhanti_project", # schema name of the database
43.         autocommit=True, # executes commit functionality in database
44.         cursorclass=pymysql.cursors.DictCursor # python libraries
45.     )
46.
47. # with successful database connection, read data from the database
48. with connection.cursor() as cursor:
49.
50.     # build up of the SQL query is shown at the start of the file
51.     # the query executed will fetch details of number of people with respective blood
    type residing in every state.
52.     analysis1_sql = """SELECT states.name, people.blood_type, COUNT(*) AS Count_of_Peop
    le_BloodType_in_every_State FROM people
53.         JOIN states ON states.id = people.state_id GROUP BY states.id,
    people.blood_type """
54.
55.     # using the functions from the python libraries, executing the query to extract the
    data
56.     cursor.execute(analysis1_sql)
57.     # storing the data executed above in a temporary list
58.     results = cursor.fetchall()
59.     # reading the column names of the database table to keep them as headers while writ
    ing data into csv
60.     column_names = results[0].keys()
61.
62.     # opening the named csv, "analysis1.csv" and writing the data into the csv with the
    use of function 'w'
63.     with open('analysis1.csv', 'w') as csvfile:
64.
65.         # writing the data into the csv using functions like, delimiter to limit the va
    lues with a comma in this case,
66.         # reading the values from the database inside the quotes and with the column na
    mes extracted before.
67.         myCsvWriter = csv.DictWriter(csvfile,
68.                                     delimiter=',',
69.                                     quotechar='"',
70.                                     fieldnames = column_names)
71.         myCsvWriter.writeheader()
72.
73.         # going through each and every row of the results of the SQL query and writing
    into the csv used, "analysis1.csv"
74.         for row in results:
75.
76.             # writing each row of values in results to the csv file format for better a
    nalysis purpose
77.             myCsvWriter.writerow(row)

```

- **Analysis 2 -**

- 1) Build of the query is defined at the start of code.
- 2) Database connection is set up for the execution of queries.
- 3) Column names are fetched of the query and store in a temporary variable. This is used as headers for the csv file.
- 4) The query executed will fetch details of average of life expectancy against the blood types across states of United States.
- 5) Using the dictwriter, writing the results into the csv mentioned in function.
- 6) Each report is stored in a csv format which can be further utilized for analysis purpose using Tableau.
- 7) This SQL query gives the average life expectancy against each blood group

```

1. # Querying inventories table to find the inventories of the blood types
2. # SELECT * FROM inventories
3. #
4. # Joining blood banks table to inventories using the foreign key concept
5. # SELECT * FROM inventories
6. #   JOIN blood_banks
7. #     ON blood_banks.id = inventories.blood_bank_id
8. #
9. # joining states table for analysis purpose
10. # SELECT * FROM inventories
11. #   JOIN blood_banks
12. #     ON blood_banks.id = inventories.blood_bank_id
13. #   JOIN states
14. #     ON states.id = blood_banks.state_id
15. #
16. # ordering the data by states
17. # SELECT * FROM inventories
18. #   JOIN blood_banks
19. #     ON blood_banks.id = inventories.blood_bank_id
20. #   JOIN states
21. #     ON states.id = blood_banks.state_id
22. # ORDER BY states.id
23. #
24. # grouping the data by states and blood types available in the inventory for analysis
25. # SELECT states.name, states.life_expectancy, inventories.blood_type FROM inventories
26. #   JOIN blood_banks
27. #     ON blood_banks.id = inventories.blood_bank_id
28. #   JOIN states
29. #     ON states.id = blood_banks.state_id
30. # GROUP BY states.id, inventories.blood_type
31. #
32. # sub-querying the results to fetch table "a" specific data
33. # SELECT *
34. # FROM (SELECT states.name, states.life_expectancy, inventories.blood_type
35. #       FROM inventories
36. #       JOIN blood_banks
37. #         ON blood_banks.id = inventories.blood_bank_id
38. #       JOIN states
39. #         ON states.id = blood_banks.state_id
40. # GROUP BY states.id, inventories.blood_type) a
41. #
42. # ordering the data from table "a" by blood type and life expectancy
43. # SELECT a.blood_type, a.life_expectancy
44. # FROM (SELECT states.name, states.life_expectancy, inventories.blood_type

```

```

45. #         FROM inventories
46. #             JOIN blood_banks
47. #                 ON blood_banks.id = inventories.blood_bank_id
48. #             JOIN states
49. #                 ON states.id = blood_banks.state_id
50. # GROUP BY states.id, inventories.blood_type) a
51. # ORDER BY a.blood_type, a.life_expectancy DESC
52. #
53. # sub-query in the data set to fetch table "b" specific results
54. # SELECT *
55. # FROM (SELECT a.blood_type, a.life_expectancy
56. #         FROM (SELECT states.name, states.life_expectancy, inventories.blood_type
57. #               FROM inventories
58. #                   JOIN blood_banks
59. #                       ON blood_banks.id = inventories.blood_bank_id
60. #                   JOIN states
61. #                       ON states.id = blood_banks.state_id
62. #               GROUP BY states.id, inventories.blood_type) a
63. #         ORDER BY a.blood_type, a.life_expectancy DESC) b
64. #
65. # ordering table "b" by blood types
66. # SELECT *
67. # FROM (SELECT a.blood_type, a.life_expectancy
68. #         FROM (SELECT states.name, states.life_expectancy, inventories.blood_type
69. #               FROM inventories
70. #                   JOIN blood_banks
71. #                       ON blood_banks.id = inventories.blood_bank_id
72. #                   JOIN states
73. #                       ON states.id = blood_banks.state_id
74. #               GROUP BY states.id, inventories.blood_type) a
75. #         ORDER BY a.blood_type, a.life_expectancy DESC) b
76. # ORDER BY b.blood_type
77. #
78. # grouping by blood type after ordering the data in table "b"
79. # SELECT AVG(b.life_expectancy) as average, b.blood_type
80. # FROM (SELECT a.blood_type, a.life_expectancy
81. #         FROM (SELECT states.name, states.life_expectancy, inventories.blood_type
82. #               FROM inventories
83. #                   JOIN blood_banks
84. #                       ON blood_banks.id = inventories.blood_bank_id
85. #                   JOIN states
86. #                       ON states.id = blood_banks.state_id
87. #               GROUP BY states.id, inventories.blood_type) a
88. #         ORDER BY a.blood_type, a.life_expectancy DESC) b
89. # GROUP BY b.blood_type
90. #
91. # ordering the analysis results by average of life expectancy to find the blood type with
    better life expectancy average
92. # SELECT AVG(b.life_expectancy) as average, b.blood_type
93. # FROM (SELECT a.blood_type, a.life_expectancy
94. #         FROM (SELECT states.name, states.life_expectancy, inventories.blood_type
95. #               FROM inventories
96. #                   JOIN blood_banks
97. #                       ON blood_banks.id = inventories.blood_bank_id
98. #                   JOIN states
99. #                       ON states.id = blood_banks.state_id
100. #               GROUP BY states.id, inventories.blood_type) a
101. #         ORDER BY a.blood_type, a.life_expectancy DESC) b
102. # GROUP BY b.blood_type
103. # ORDER BY average DESC
104.

```



```

105.     # import the relevant python libraries to use the functionalities.
106.     import pymysql.cursors
107.     import pprint
108.     import csv
109.
110.     # Set up connection with database using the connection variable below
111.     connection = pymysql.connect(
112.         host="localhost", # server details
113.         user="milind_siddhanti", # username for authentication
114.         passwd="nokiayu7k", # password to connect with the database
115.         db="milind_siddhanti_project", # schema name of the database
116.         autocommit=True, # executes commit functionality in database
117.         cursorclass=pymysql.cursors.DictCursor # python libraries
118.     )
119.
120.     # with successful database connection, read data from the database
121.     with connection.cursor() as cursor:
122.
123.         # build up of the SQL query is shown at the start of the file
124.         # the query executed will fetch details of average of life expectancy against the blood types across states of United States
125.         analysis4_sql = """SELECT AVG(b.life_expectancy) as average, b.blood_type FROM
126.             (SELECT a.blood_type, a.life_expectancy FROM
127.                 (SELECT states.name, states.life_expectancy, inventories
128.                  .blood_type FROM inventories JOIN blood_banks ON blood_banks.id = inventories.blood_bank_id
129.                  JOIN states ON states.id = blood_banks.state_id GROUP BY
130.                   states.id, inventories.blood_type) a
131.                 ORDER BY a.blood_type, a.life_expectancy DESC) b GROUP BY
132.                  b.blood_type ORDER BY average DESC"""
133.
134.         # using the functions from the python libraries, executing the query to extract the data
135.         cursor.execute(analysis4_sql)
136.         # storing the data executed above in a temporary list
137.         results = cursor.fetchall()
138.         # reading the column names of the database table to keep them as headers while writing data into csv
139.         column_names = results[0].keys()
140.
141.         # opening the named csv, "analysis4.csv" and writing the data into the csv with the use of function 'w'
142.         with open('analysis4.csv', 'w') as csvfile:
143.
144.             # writing the data into the csv using functions like, delimiter to limit the values with a comma in this case,
145.             # reading the values from the database inside the quotes and with the column names extracted before.
146.             myCsvWriter = csv.DictWriter(csvfile,
147.                                         delimiter=',',
148.                                         quotechar='"',
149.                                         fieldnames = column_names)
150.
151.             myCsvWriter.writeheader()
152.
153.             # going through each and every row of the results of the SQL query and writing into the csv used, "analysis1.csv"
154.             for row in results:
155.
156.                 # writing each row of values in results to the csv file format for better analysis purpose
157.                 myCsvWriter.writerow(row)

```

- **Analysis 3 –**

- 1) Build of the query is defined at the start of code.
- 2) Database connection is set up for the execution of queries.
- 3) Column names are fetched of the query and store in a temporary variable. This is used as headers for the csv file.
- 4) The query executed will fetch details of quantity of blood seen in the inventory for respective blood types and every state.
- 5) Using the dictwriter, writing the results into the csv mentioned in function.
- 6) Each report is stored in a csv format which can be further utilized for analysis purpose using Tableau.
- 7) This SQL query gives the quantity of different blood types available in every state.

```
1. # Querying the database table inventories
2. # SELECT * FROM inventories
3. #
4. # joining data from database table blood banks for comparing
5. # SELECT * FROM inventories
6. #     JOIN blood_banks
7. #         ON blood_banks.id = inventories.blood_bank_id
8. #
9. # joining states database table to compare the inventories for each state
10. # SELECT * FROM inventories
11. #     JOIN blood_banks
12. #         ON blood_banks.id = inventories.blood_bank_id
13. #     JOIN states
14. #         ON states.id = blood_banks.state_id
15. #
16. # ordering the data by states to compare statewise data
17. # SELECT * FROM inventories
18. #     JOIN blood_banks
19. #         ON blood_banks.id = inventories.blood_bank_id
20. #     JOIN states
21. #         ON states.id = blood_banks.state_id
22. # ORDER BY states.name
23. #
24. # grouping this data by states
25. # SELECT states.name FROM inventories
26. #     JOIN blood_banks
27. #         ON blood_banks.id = inventories.blood_bank_id
28. #     JOIN states
29. #         ON states.id = blood_banks.state_id
30. # GROUP BY states.name
31. #
32. # grouping by blood types along with states to combine the data for analysis
33. # SELECT states.name, inventories.blood_type FROM inventories
34. #     JOIN blood_banks
35. #         ON blood_banks.id = inventories.blood_bank_id
36. #     JOIN states
37. #         ON states.id = blood_banks.state_id
38. # GROUP BY states.name, inventories.blood_type
39. #
40. # summing the quantity available in the blood banks across every state and blood types
    available in that respective state
41. # SELECT states.name, inventories.blood_type, SUM(inventories.quantity_cc) AS Quantity
    FROM inventories
42. #     JOIN blood_banks
```

```

43. #     ON blood_banks.id = inventories.blood_bank_id
44. #     JOIN states
45. #     ON states.id = blood_banks.state_id
46. # GROUP BY states.name, inventories.blood_type
47. #
48. # ordering the above availed data with every state. this data consists of every blood t
    ype available in each state with their quantities
49. # SELECT states.name, inventories.blood_type, SUM(inventories.quantity_cc) AS Quantity
    FROM inventories
50. #     JOIN blood_banks
51. #     ON blood_banks.id = inventories.blood_bank_id
52. #     JOIN states
53. #     ON states.id = blood_banks.state_id
54. # GROUP BY states.name, inventories.blood_type
55. # ORDER BY states.name
56.
57. # import the relevant python libraries to use the functionalities.
58. import pymysql.cursors
59. import pprint
60. import csv
61.
62. # Set up connection with database using the connection variable below
63. connection = pymysql.connect(
64.     host="localhost", # server details
65.     user="milind_siddhanti", # username for authentication
66.     passwd="nokiayu7k", # password to connect with the database
67.     db="milind_siddhanti_project", # schema name of the database
68.     autocommit=True, # executes commit functionality in database
69.     cursorclass=pymysql.cursors.DictCursor # python libraries
70. )
71.
72. # with successful database connection, read data from the database
73. with connection.cursor() as cursor:
74.
75.     # build up of the SQL query is shown at the start of the file
76.     # the query executed will fetch details of quantity of blood seen in the inventory
    for respective blood types and every state.
77.     analysis5_sql = """SELECT states.name, inventories.blood_type, SUM(inventories.quan
    tity_cc) AS Quantity FROM inventories JOIN blood_banks ON blood_banks.id = inventories.
    blood_bank_id
78.     JOIN states ON states.id = blood_banks.state_id GROUP BY states.
    name, inventories.blood_type ORDER BY states.name"""
79.
80.     # using the functions from the python libraries, executing the query to extract the
    data
81.     cursor.execute(analysis5_sql)
82.     # storing the data executed above in a temporary list
83.     results = cursor.fetchall()
84.     # reading the column names of the database table to keep them as headers while writ
    ing data into csv
85.     column_names = results[0].keys()
86.
87.     # opening the named csv, "analysis5.csv" and writing the data into the csv with the
    use of function 'w'
88.     with open('analysis5.csv', 'w') as csvfile:
89.
90.         # writing the data into the csv using functions like, delimiter to limit the va
    lues with a comma in this case,
91.         # reading the values from the database inside the quotes and with the column na
    mes extracted before.
92.         myCsvWriter = csv.DictWriter(csvfile,

```

```

93.                                     delimiter=',',
94.                                     quotechar='"',
95.                                     fieldnames = column_names)
96.         myCsvWriter.writeheader()
97.
98.         # going through each and every row of the results of the SQL query and writing
           into the csv used, "analysis1.csv"
99.         for row in results:
100.
101.             # writing each row of values in results to the csv file format for b
           etter analysis purpose
102.             myCsvWriter.writerow(row)

```

Visualization:

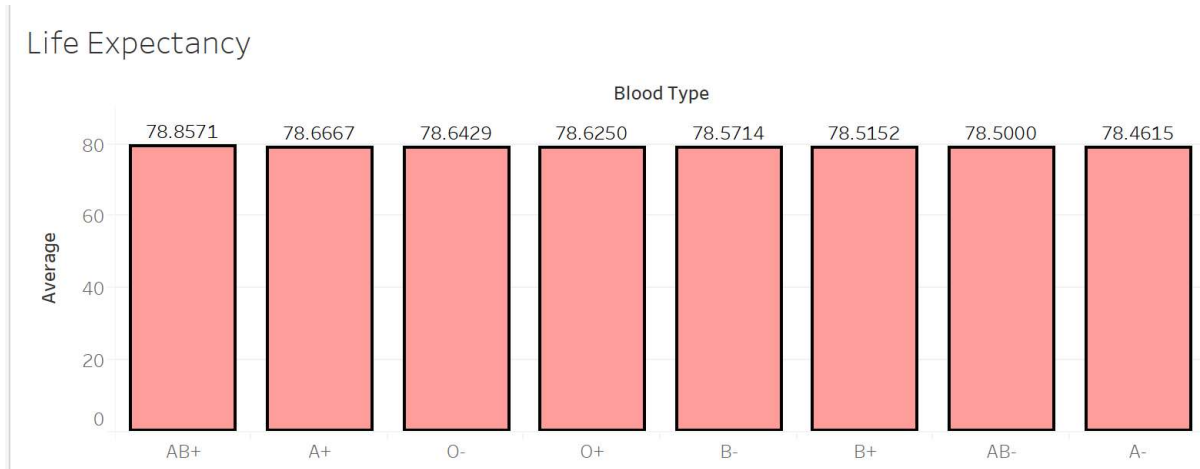
For this project we used Tableau as our visualization tool to graphically represent our findings. To understand how to use this tool we used YouTube videos and the active online help community. Tableau online community help was extremely useful as most of the solutions we were looking for we already explained in a detailed manner. Tableau is an extremely powerful tool since it can link with different formats of input sources and even combine data from different sheets if required. It was interesting to understand how joins can be created on the fly while working on Tableau.

We wanted our visualizations to be easily decipherable, hence we used bar graphs to display statistics. We used the Tableau maps for one of our visualizations that emphasizes blood types distribution in the country.

We used python code to run SQL queries to extract data from the database for our analysis. The extracted data was then written in a new csv to have ready data that can be imported into Tableau.

1. Life Expectancy of Each Blood Group

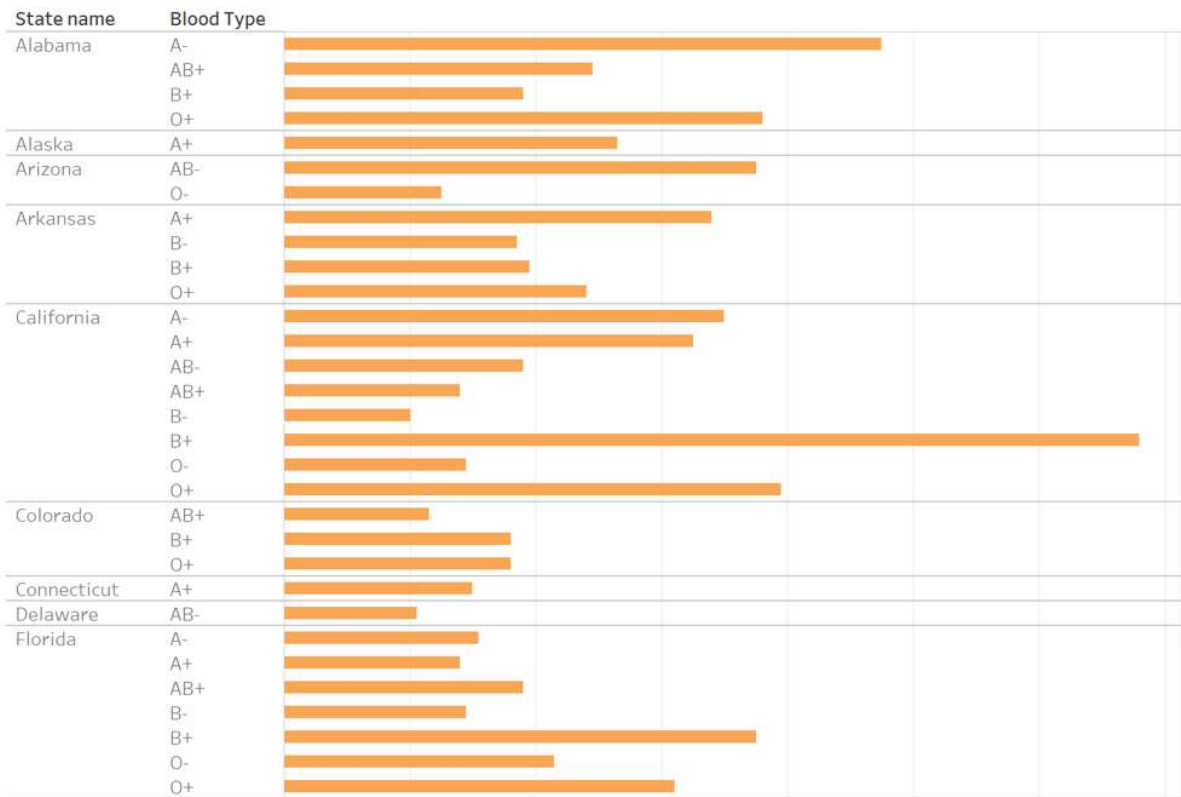
The below graph shows the comparison of blood type with the average life expectancy. The Y-axis consists of age in years and X-axis consists of all 8 types of blood group. We ordered the graph in descending order to find the blood group with the highest life expectancy. From the graph it can easily be understood that AB+ blood group has the highest life expectancy.

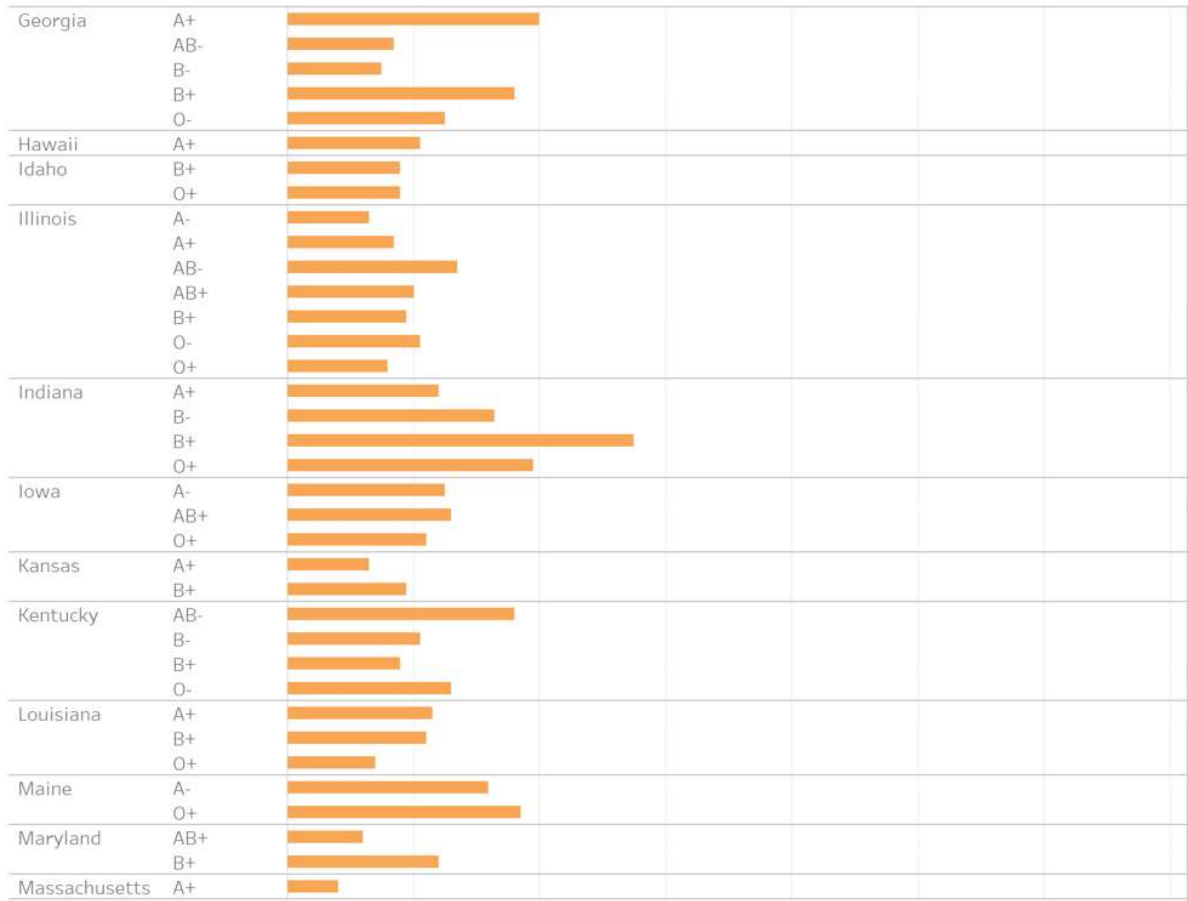


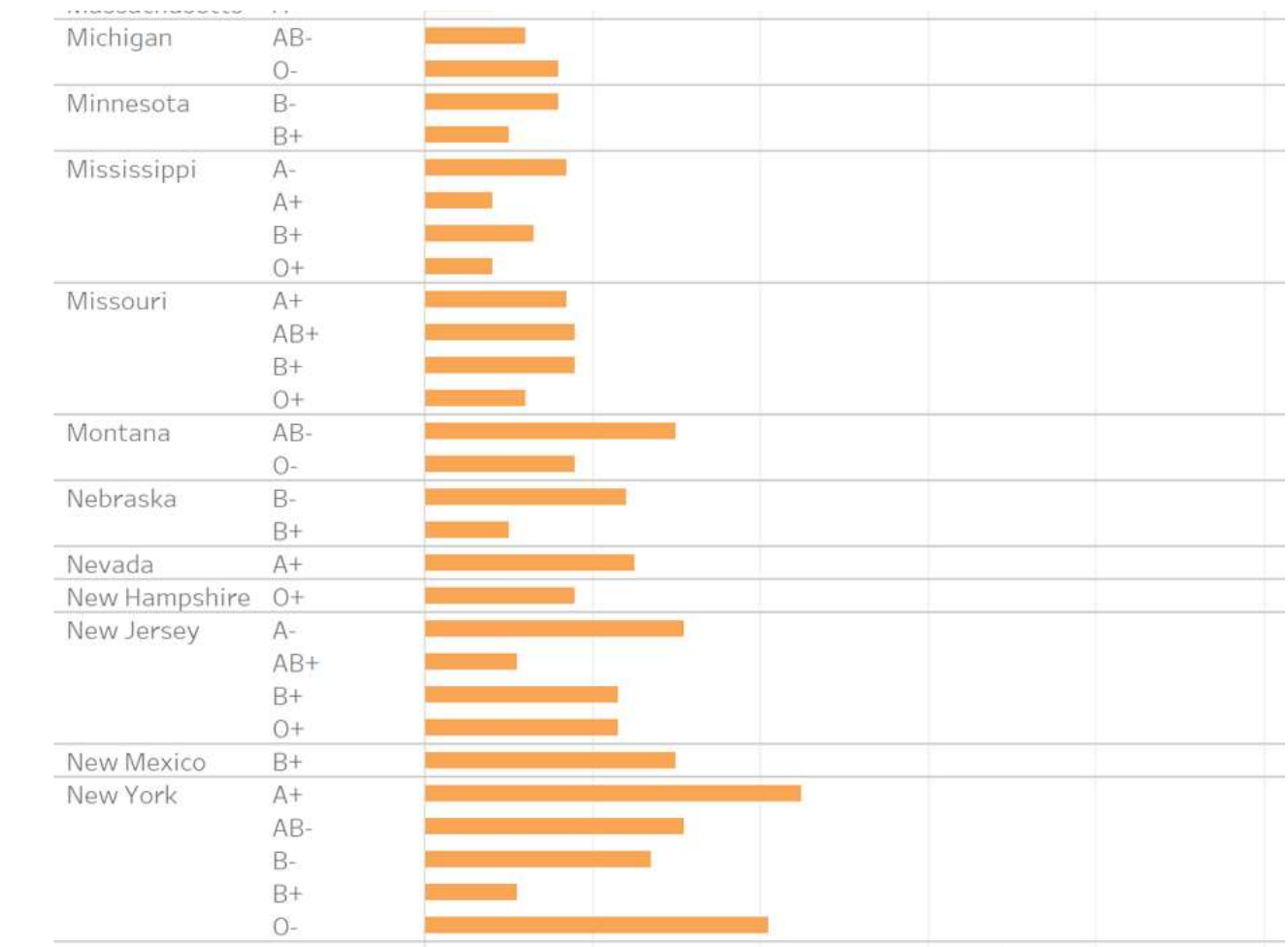
2. Availability of Blood Type in Every State

The next visualization highlights the quantity of blood group available in each state. We used blood types and state name as the rows and quantity of blood (in cubic centimeters) as the columns. To be able to determine shortage of blood type or which blood bank to be contacted in case of need this analysis will be helpful.

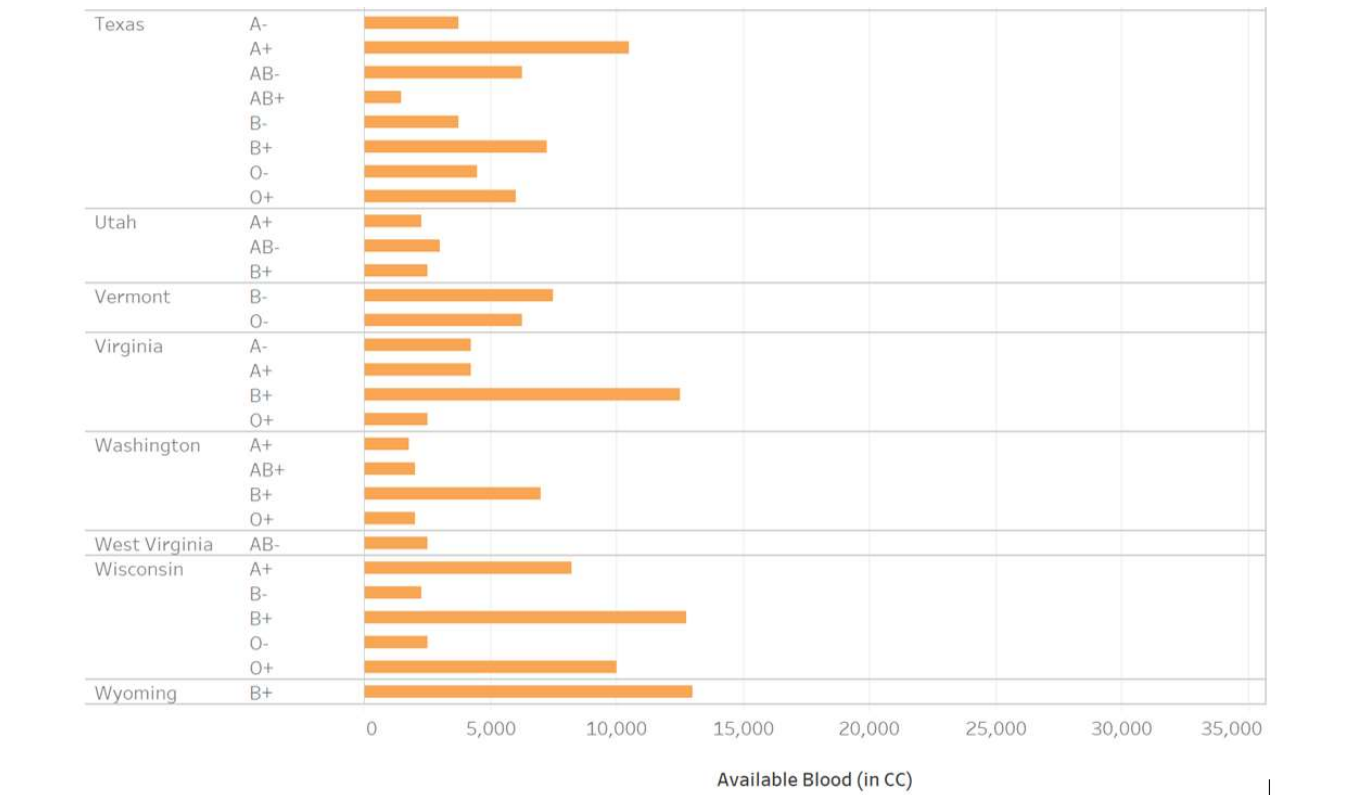
Blood Availability in each state







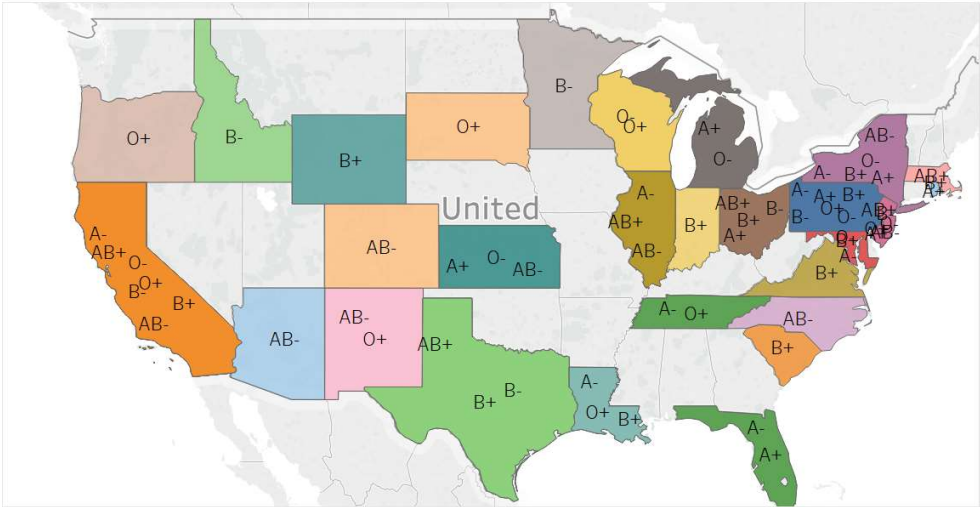




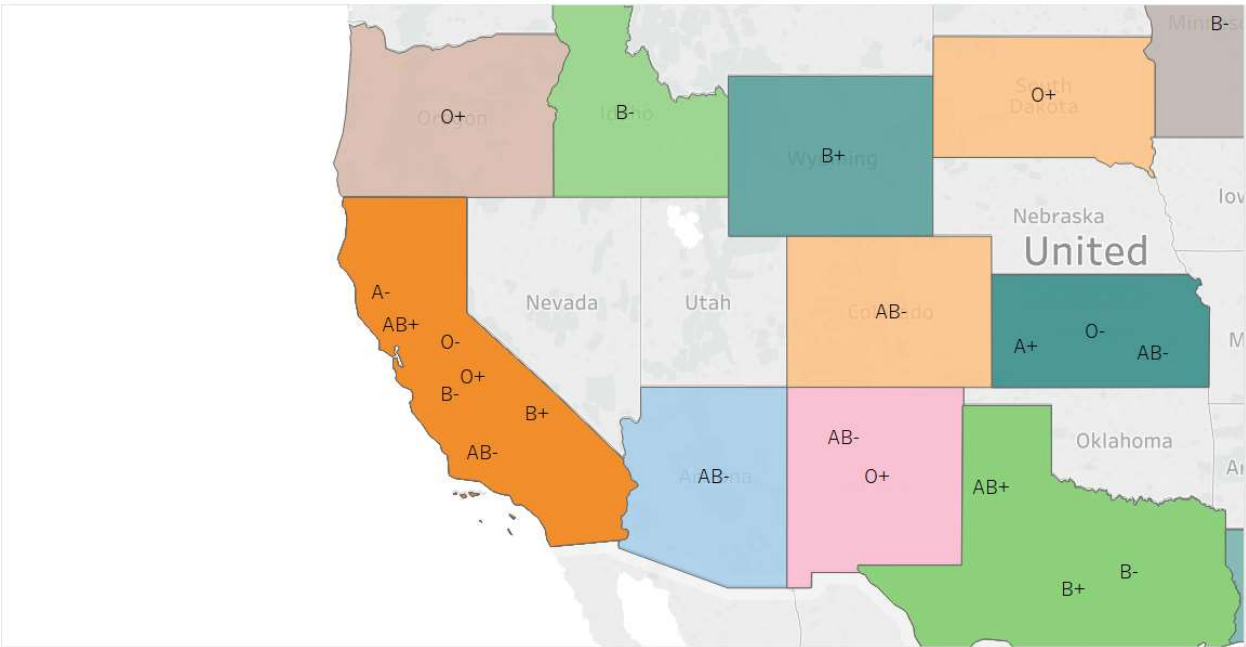
3. Blood Type Distribution across states

Our final visualization uses a map to display the blood_type distribution in The United States of America. The longitude and latitude measures were used to plot the map and then we used name and blood type in marks to display the data. We explicitly displayed the labels on the map. To provide a clearer view of the blood type distribution we have even attached zoomed-in versions of the map covering all 4 regions.

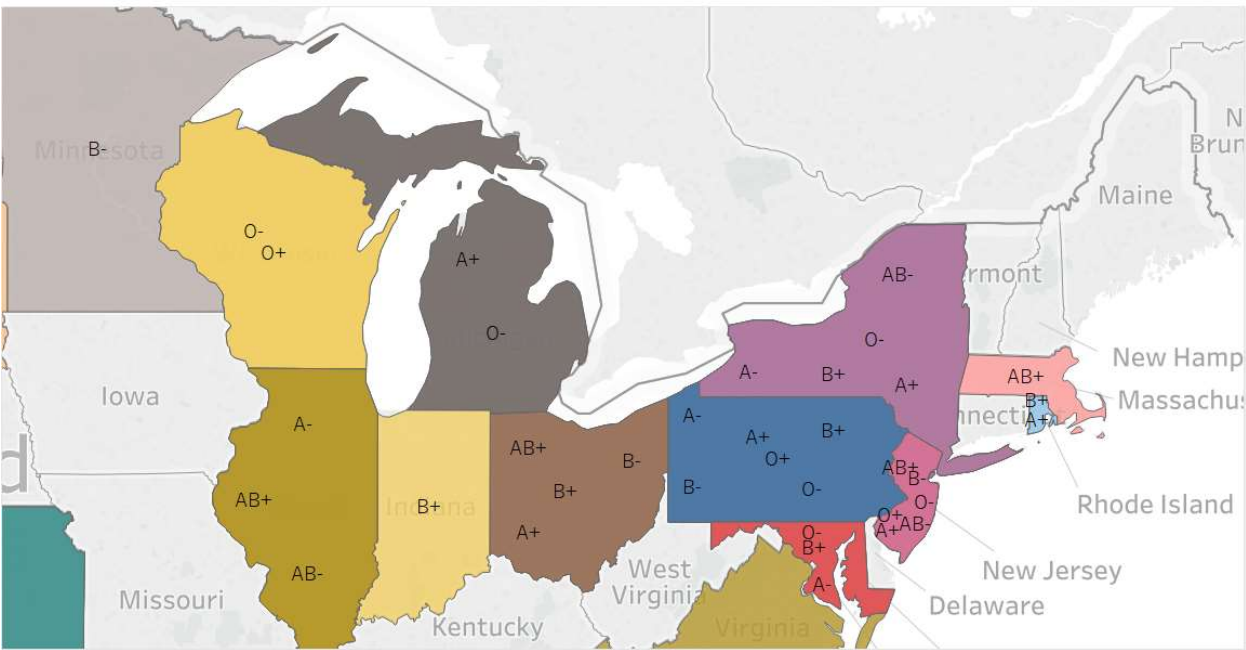
Blood Type Distribution in USA



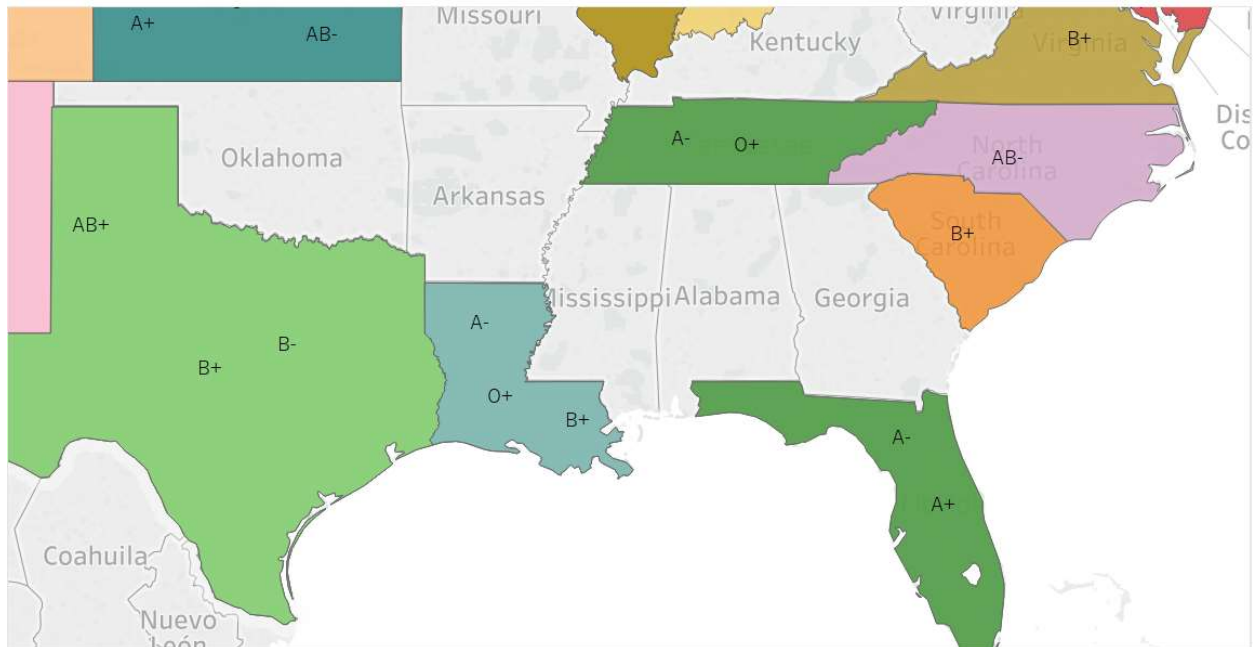
Blood Type Distribution in USA



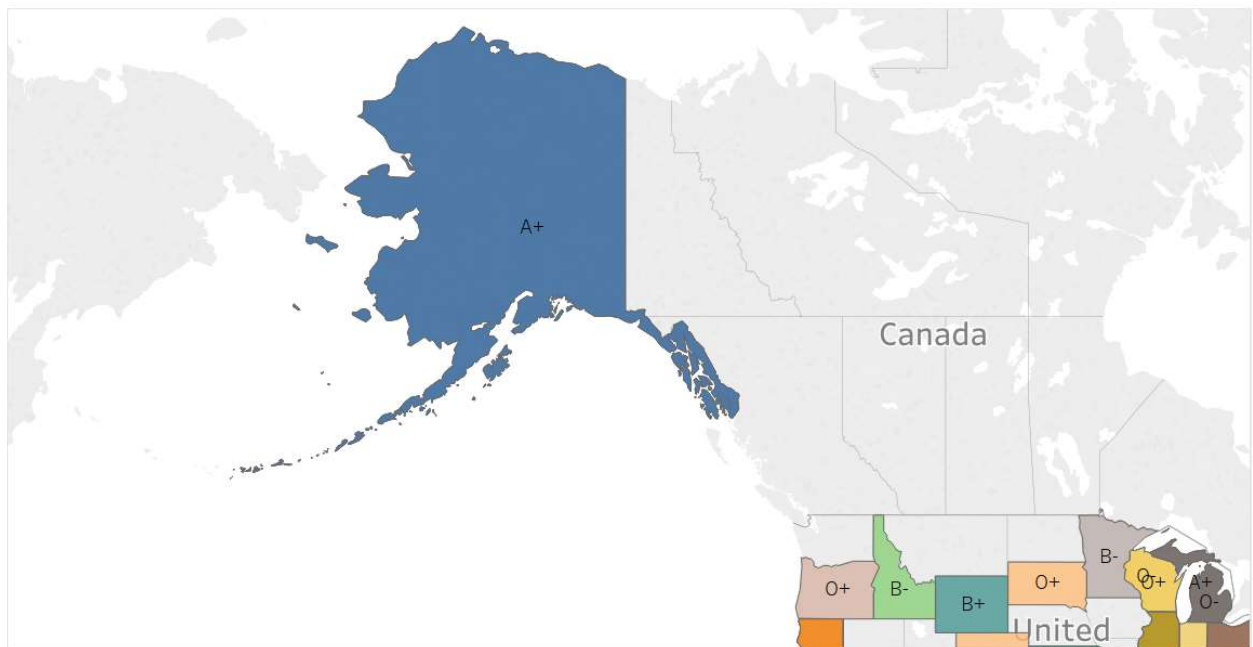
Blood Type Distribution in USA



Blood Type Distribution in USA



Blood Type Distribution in USA



Challenges:

1. While searching for datasets, we were facing problems with finding the data that we required, or defined as we needed. Since our Project dealt with data that is not easily available, for example, blood type data is confidential information, so we went ahead with collecting multiple datasets and later combined them in a single file along with manually adding data to be uploaded as a table into the database using Python.
2. Since the datasets were not in the format that we needed it to be, we used some of the excel functions to match the dataset as we wanted it to be. Indexing, merging and trim were some of the functionalities that were used to overcome the challenge of combining data from multiple CSVs.
3. When importing code from csv to database through python, we faced problems while using regex format to trim and retrieve data and feed to table column. Since we had combined data of states and its abbreviation, we wanted to use a regex string that was specific for any length of state name and abbreviation length. We used the basic split function to fetch the required data from the dataset with the assumption of fixed length data.
4. While writing queries for analysis, we had problems building the query for any suitable conditions. There were some conditions like “if” and “for” that we wanted to use. Since we were not familiar with the programming SQL, we used sub-queries to fetch the required data from the database design for analysis purposes.