Data Science Baseball Project

Objective:

The project on the sport of Baseball, which is a ball game played between two teams of nine on a field with a diamond-shaped circuit of four bases. The vision of the project was to not only clean the data using different types of statistics but also conclude on how age affects the overall ability of a certain player. l

Plan of Action:

To accomplish this task, I divided up the tasks into certain goals. First, I needed to find a baseball library which consisted of baseball data for a long period of time. Then I needed to analyze different types of data that would allow us to properly conclude the age and overall ability of a certain player correlation. Following this, I had to learn the different types of libraries and functions in order to get the data that I wanted. Finally, I had to compile data together in order to come up with the conclusion.

Required Libraries:

In order to accomplish this task I needed to import certain libraries such as seaborn, which is a data visualization package that provides a high-level interface for drawing statistical graphics. Additionally, it consists of the basic libraries such as pandas and numpy which allows us to make datasets, arrays and sets. I also imported new libraries and functions such as warnings and math which allowed us to do certain calculations.

Process of Project

I found a baseball library that had the statistics of all players and also teams and important information that would allow us to make the best conclusion about the players with respect to age and performance. Then I started parsing data into a dataframe that held all the data shown below:

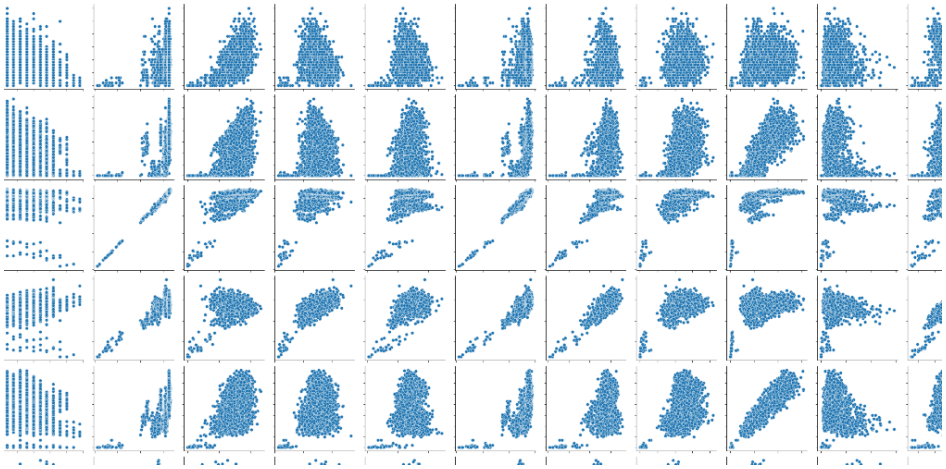
```
[ ] # Examine the dataframe to see if it loaded correctly
    df.head()
```

	team_id	rank	g	w	l	r	ab	h	bb	so	sb	ra	er	sho	sv	ipouts	ha	soa	e
0	BS1	3	31	20	10	401	1372	426	60	19.0	73.0	303	109	1	3	828	367	23	225
1	CH1	2	28	19	9	302	1196	323	60	22.0	69.0	241	77	0	1	753	308	22	218
2	CL1	8	29	10	19	249	1186	328	26	25.0	18.0	341	116	0	0	762	346	34	223
3	FW1	7	19	7	12	137	746	178	33	9.0	16.0	243	97	1	0	507	261	17	163
4	NY2	5	33	16	17	302	1404	403	33	15.0	46.0	313	121	1	0	879	373	22	227

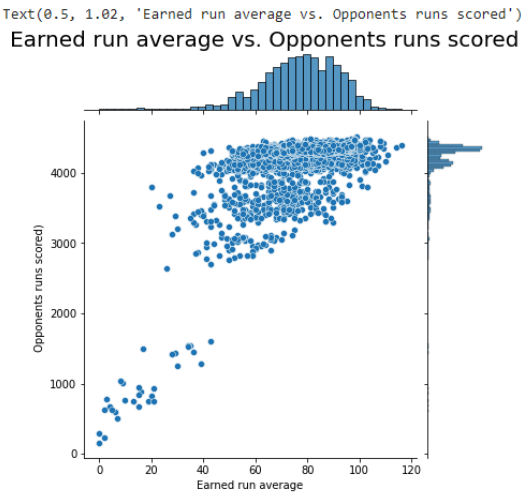
```
[ ] df.corr()
```

	rank	g	w	l	r	ab	h	bb	so	sb	ra	er	sho	sv	ipouts	ha	soa	e
rank	1.000000	-0.143430	-0.773312	0.606335	-0.429614	-0.170750	-0.296581	-0.302016	-0.221433	-0.041684	0.384020	0.182319	-0.375781	-0.423783	-0.185207	0.143223	-0.352687	0.327102
g	-0.143430	1.000000	0.566714	0.554389	0.368193	0.979525	0.779888	0.624044	0.578228	-0.101661	0.352001	0.579040	0.307447	0.479080	0.993828	0.765939	0.578484	-0.375121
w	-0.773312	0.566714	1.000000	-0.368092	0.589466	0.583332	0.630815	0.556059	0.296760	0.030440	-0.216820	0.001610	0.527770	0.460959	0.598329	0.197428	0.430278	-0.330314
l	0.606335	0.554389	-0.368092	1.000000	-0.186783	0.515837	0.242154	0.148732	0.383150	-0.177132	0.609652	0.667155	-0.190776	0.114489	0.517918	0.663927	0.247857	-0.135269
r	-0.429614	0.368193	0.589466	-0.186783	1.000000	0.484411	0.749733	0.552231	0.098468	0.228226	0.471388	0.432638	-0.094865	0.166925	0.345538	0.470717	0.154183	0.070090
ab	-0.170750	0.979525	0.583332	0.515837	0.484411	1.000000	0.862401	0.622899	0.550427	-0.105264	0.423627	0.632456	0.244490	0.470193	0.974843	0.807523	0.558794	-0.348717
h	-0.296581	0.779888	0.630815	0.242154	0.749733	0.862401	1.000000	0.564704	0.282848	-0.056018	0.475881	0.620683	0.106737	0.330978	0.769377	0.790138	0.339122	-0.254680
bb	-0.302016	0.624044	0.556059	0.148732	0.552231	0.622899	0.564704	1.000000	0.401486	-0.114555	0.287914	0.480176	0.131678	0.406664	0.626095	0.489750	0.408086	-0.353221
so	-0.221433	0.578228	0.296760	0.383150	0.098468	0.550427	0.282848	0.401486	1.000000	-0.188827	0.147750	0.470936	0.123627	0.803736	0.591469	0.333725	0.914667	-0.609840
sb	-0.041684	-0.101661	0.030440	-0.177132	0.228226	-0.105264	-0.056018	-0.114555	-0.188827	1.000000	0.086570	-0.222318	-0.047352	-0.235726	-0.126376	-0.131983	-0.174448	0.622124
ra	0.384020	0.352001	-0.216820	0.609652	0.471388	0.423627	0.475881	0.287914	0.147750	0.086570	1.000000	0.853549	-0.521843	-0.007827	0.289532	0.763531	0.041360	0.194457
er	0.182319	0.579040	0.001610	0.667155	0.432638	0.632456	0.620683	0.480176	0.470936	-0.222318	0.853549	1.000000	-0.364498	0.365782	0.542058	0.853701	0.379594	-0.326580
sho	-0.375781	0.307447	0.527770	-0.190776	-0.094865	0.244490	0.106737	0.131678	0.123627	-0.047352	-0.521843	-0.364498	1.000000	0.146171	0.344642	-0.144863	0.227625	-0.212775
sv	-0.423783	0.479080	0.460959	0.114489	0.166925	0.470193	0.330978	0.406664	0.803736	-0.235726	-0.007827	0.365782	0.146171	1.000000	0.507298	0.237047	0.829265	-0.709150
ipouts	-0.185207	0.993828	0.598329	0.517918	0.345538	0.974843	0.769377	0.626095	0.591469	-0.126376	0.289532	0.542058	0.344642	0.507298	1.000000	0.733901	0.599997	-0.418222
ha	0.143223	0.765939	0.197428	0.663927	0.470717	0.807523	0.790138	0.489750	0.333725	-0.131983	0.763531	0.853701	-0.144863	0.237047	0.733901	1.000000	0.253485	-0.190938
soa	-0.352687	0.578484	0.430278	0.247857	0.154183	0.558794	0.339122	0.408086	0.914667	-0.174448	0.041360	0.379594	0.227625	0.829265	0.599997	0.253485	1.000000	-0.627250
e	0.327102	-0.375121	-0.330314	-0.135269	0.070090	-0.348717	-0.254680	-0.353221	-0.609840	0.622124	0.194457	-0.326580	-0.212775	-0.709150	-0.418222	-0.190938	-0.627250	1.000000

Then proceeded to produce an output of certain graphs by comparing certain numeric variables in the dataset. For example, I explored the relationship between the total number of games played vs the winning of a certain team.

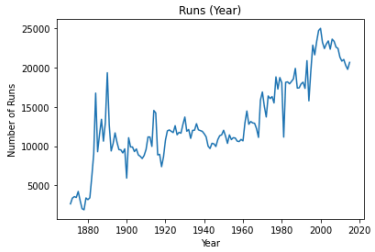


At the same time, I was also able to produce bar chart graphs, for example to represent an earned run average vs the opponent runs scored:



Some of the main factors in baseball are Batting, Pitching, Fielding and I was able to really dive deep into the bits of certain aspects starting from creating graphs of how many runs were scored in a specific year to outputting certain players with a specific physical stature such as height and weight.

```
homeruns = df_team[['year', 'r']].copy()
hr_by_year = homeruns.groupby('year')['r'].sum()
hr_by_year.sort_index()
hr_by_year.plot();
plt.ylabel('Number of Runs')
plt.xlabel('Year')
plt.title('Runs (Year)');
```

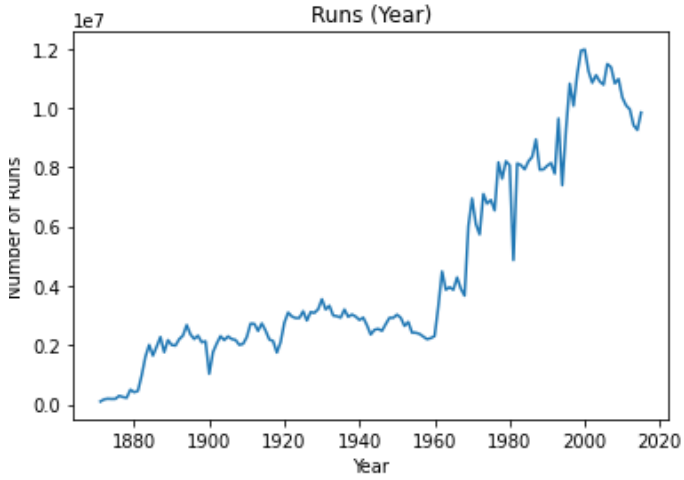



```
df_player=pd.read_csv("/content/drive/MyDrive/player.csv",
                      usecols=['player_id','name_given','birth_country','weight', 'height'])
#df_player=df_player.set_index('player_id')
df_player.head()
```

	player_id	birth_country	name_given	weight	height
0	aardsda01	USA	David Allan	220.0	75.0
1	aaronha01	USA	Henry Louis	180.0	72.0
2	aaronto01	USA	Tommie Lee	190.0	75.0
3	aasedo01	USA	Donald William	190.0	75.0
4	abadan01	USA	Fausto Andres	184.0	73.0

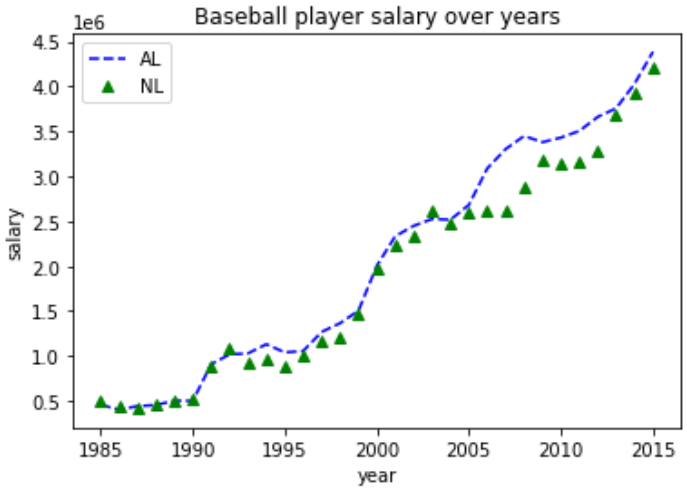
In the batting sector, there was lots of data such as singles, doubles, triples, home runs, batting averages, home runs, etc. I was able to retrieve data of all the batters in the database and then organize it with rank, year of playing and all their key stats through graphs and charts as shown below:

	player_id	team_id	ab	h	double	triple	hr	rbi	sb	cs	bb	year	rank	g	w	l	r
0	abercda01	TRO	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1871	6	29	13	15	351
1	abercda01	TRO	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1872	5	25	15	10	273
2	beaveed01	TRO	15.0	6.0	0.0	0.0	0.0	5.0	2.0	0.0	0.0	1871	6	29	13	15	351
3	beaveed01	TRO	15.0	6.0	0.0	0.0	0.0	5.0	2.0	0.0	0.0	1872	5	25	15	10	273
4	bellast01	TRO	128.0	32.0	3.0	3.0	0.0	23.0	4.0	4.0	9.0	1871	6	29	13	15	351
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7967178	valdejo02	MIA	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2015	3	162	71	91	613
7967179	yelicch01	MIA	476.0	143.0	30.0	2.0	7.0	44.0	16.0	5.0	47.0	2012	5	162	69	93	609
7967180	yelicch01	MIA	476.0	143.0	30.0	2.0	7.0	44.0	16.0	5.0	47.0	2013	5	162	62	100	513
7967181	yelicch01	MIA	476.0	143.0	30.0	2.0	7.0	44.0	16.0	5.0	47.0	2014	4	162	77	85	645
7967182	yelicch01	MIA	476.0	143.0	30.0	2.0	7.0	44.0	16.0	5.0	47.0	2015	3	162	71	91	613



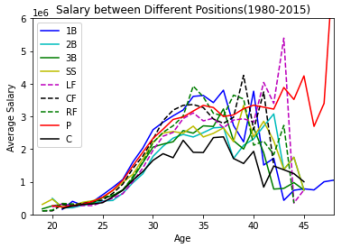
When I was able to, I also parsed out data for the salaries of all the players and was able to see a trendline about how the salary has changed over the years.

```
'ext(0.5, 1.0, 'Baseball player salary over years')
```

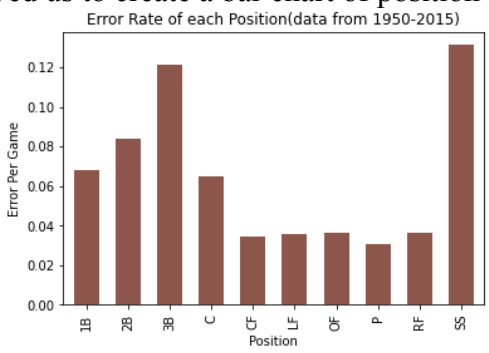


Since there are numerous positions in the sport of baseball, there are different salaries for different positions on the field such as batter, pitcher, catcher, outfield, 1st, 2nd, and 3rd bases. I was able to take the average salary of a certain position and then compare it to the ages of the baseball players with different line charts shown below.

```
] # Salary between different positions and age

  warnings.filterwarnings('ignore')
  player_position = pd.merge(salary, fielding, on = ["player_id","year"])[["year", "player_id", "salary", "pos"]]
  player_position = pd.merge(player_position, player, on = ["player_id"], how = "left")[["year", "player_id","salary", "pos", "birth_year"]]
  player_position["age"] = player_position["year"] - player_position["birth_year"]
  player_position = player_position[player_position.year >= 1980]
  data = player_position.groupby(["age", "pos"]).mean()["salary"].unstack()
  #plt.plot(data["1B"],  data["2B"], data["3B"], data["C"], data["CF"], data["LF"], data["P"], data["RF"], data["SS"])
  plt.plot(data["1B"], 'b-', data["2B"], 'c-', data["3B"], 'g-', data["SS"], 'y-',data['LF'], 'm--', data['CF'], 'k--', data['RF'], 'g--', data['P'],'r-', data['C'],'k-')
  _ = plt.xlabel("Age")
  _ = plt.ylabel("Average Salary")
  _ = plt.legend(["1B",'2B','3B','SS',"LF","CF","RF","P","C"], loc = 'leftup')
  _ = plt.title("Salary between Different Positions(1980-2015)")
  _ = plt.xlim([18,48])
  _ = plt.ylim([0,6000000])
```



With salaries varying between different positions, it is very clear that one who makes mistakes has repercussions, especially at such a high level. Hence, I calculated the error rate for each position which allowed us to create a bar chart of position vs Error per game.



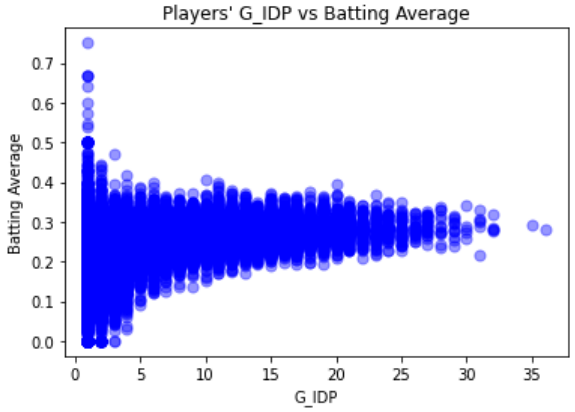
At the end of the day I was able to conclude and ask questions and see if it had a different outcome. This allowed us to create certain scatter plots to see the batting average and just make any other conclusions about age and performance.

```
# Do players with a high ground into double play (GIDP) have a lower batting average?
# batting average = h/ab
warnings.filterwarnings('ignore')
batting = pd.read_csv("/content/drive/MyDrive/batting.csv")
batting_postseason = pd.read_csv("/content/drive/MyDrive/batting_postseason.csv")
all_batting = pd.concat([batting, all_batting], axis = 0)
all_batting = all_batting.groupby(["year","player_id"]).sum()[["ab", "h", "g_idp"]]
all_batting = all_batting[np.isfinite(all_batting['g_idp'])]
all_batting = all_batting[(all_batting["ab"] != 0) & (all_batting["g_idp"]!= 0)]
all_batting["batting_average"] = all_batting["h"]/all_batting["ab"]

from sklearn.linear_model import LinearRegression

algorithm = LinearRegression()
predictors = ["g_idp"]

#Initiate our algorithm class
algorithm = LinearRegression()
predictors = all_batting[predictors]
# The target we're using to train the algorithm.
target = all_batting["batting_average"]
# Training the algorithm using the predictors and target.
algorithm.fit(predictors, target)
print('Coefficients: \n', algorithm.coef_)
# From regression, we can see that the coefficient is not negative, thus the assumption is not appropriate
```

```
Coefficients:
 [0.0053944]
```

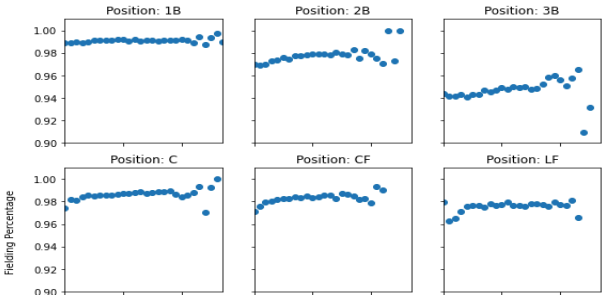
```
# Let's draw the scatter plot
s = 100
plt.scatter(predictors, target, color='b', s=s/2, alpha=.4)
plt.xlabel("G_IDP")
plt.ylabel("Batting Average")
plt.title("Players' G_IDP vs Batting Average")
# from the plot, there is no evidence that high g_idp will cause low batting average.
# However, the std of batting average decreases as G_IDP increases
```

```
Text(0.5, 1.0, "Players' G_IDP vs Batting Average")
```



Similar to the batting average, I was able to consider the same for the fielding percentage, which allowed us to do the following:

```
# 1. Consider Fielding Percentage
all_fielding = pd.concat([fielding, fielding_postseason], axis = 0)
all_fielding = pd.merge(all_fielding, player, on = ["player_id"], how = "left")[["year","g","player_id","birth_year","pos", "po","a","e"]]
all_fielding["age"] = all_fielding['year'] - all_fielding['birth_year']
all_fielding["FP"] = (all_fielding["po"] + all_fielding["a"])/(all_fielding["po"] + all_fielding["a"] + all_fielding["e"])
all_fielding = all_fielding[all_fielding["g"]>= 10]
all_fielding = all_fielding.groupby(["pos", "age"]).mean()[["FP"]]
all_fielding = all_fielding.reset_index(level=['pos', 'age'])
positions = ['1B', '2B', '3B', 'C', 'CF', 'LF', 'P', 'RF', 'SS']

fig, axes = plt.subplots(nrows=3, ncols=3, sharex = True, sharey = True, figsize=(9, 9))
fig.text(0.5, 0.04, 'Age', ha='center')
fig.text(0.04, 0.5, 'Fielding Percentage', va='center', rotation='vertical')
axes = axes.ravel()
for i in range(9):
    data = all_fielding[all_fielding["pos"] == positions[i]]
    axes[i].scatter(data["age"], data["FP"], color = sns.color_palette()[0])
    axes[i].set_xlim(20,47)
    axes[i].set_ylim(0.90, 1.01)
    axes[i].set_title("Position: " + positions[i])
```



Conclusion:

The project really involved a team and learning new ways to create multiple types of charts, tables, graphs, and produce data in order to come to a reasonable conclusion for the Baseball players and their ages. Using different libraries and different types of functions and statements allowed us to create a new dataset that can be used in the future to predict players' long run performance with relation to age.