DS 320: Predicting Hockey Contracts

Brian Ellis Luke Kerwin Eric Wu Griffin Jordan

2023-12-05

Abstract

This report encapsulates our approach to predicting National Hockey League (NHL) player contract values by utilizing a sophisticated data integration and machine learning pipeline. Our study primarily focuses on integrating and analyzing player performance data and historical contract information, spanning from 2009 to 2022. The data is meticulously sourced from CapFriendly and Hockey Reference through Python web scraping techniques.

A key aspect of our methodology involves rigorous data cleaning and pre-processing, where we exclude Entry-Level contracts (ELCs), Restricted Free Agent contracts (RFAs), and Contract Extensions to refine the dataset for more accurate model training. These three types of contracts are restricted in ways that would skew our modeling. This cleaned data is then stored in a locally hosted MySQL database, with a Flask and SQLAlchemy-built API facilitating remote data access, demonstrating the potential for online deployment.

The cornerstone of our project is the development of a predictive model, derived from the merged dataset of contracts and performance statistics. This model, encapsulated as a Python class, is integrated into our API, enabling real-time predictions as player statistics update.

Further, we have designed an interactive dashboard using Tableau, connected directly to our MySQL database, to visually represent our findings and predictions. While currently limited to manual updates due to resource constraints, the dashboard is designed for potential future automation and easy management, should the Tableau Server be utilized.

Last but not least, we decided to try replicating the results using Snowflake API and demonstrate cloud-based deployment as our new approach. As a result of the approach, we successfully integrated the player performance data and historical contract information using the Snowflake API, created an interactive dashboard that connects directly to the Snowflake warehouse, and implemented the machine learning models using Snowpark packages and sessions.

This report not only presents a viable model for predicting NHL player contracts but also exemplifies the effective use of data integration, machine learning, and visualization techniques in sports analytics. The implications of this study extend beyond contract predictions, offering insights into data-driven decision-making in professional sports.++

Table of Contents

Introduction

In an era where data reigns supreme, sports analytics has emerged as a cornerstone of strategic decision-making in professional sports. The ability to parse through vast amounts of data and extract meaningful insights is invaluable, particularly in the high-stakes world of professional hockey. This paper delves into the realm of data integration within sports analytics, with a focus on the National Hockey League (NHL).

Data integration, the process of combining data from different sources into a unified view, plays a pivotal role in modern sports analytics. It enables a comprehensive analysis of player performance, a crucial factor in determining contract values in the NHL. However, predicting player contracts is a complex task, riddled with challenges due to the dynamic and multifaceted nature of player performance metrics and contract negotiations.

Our project confronts this challenge head-on. We aim to predict NHL player contracts by leveraging a decade's worth of player performance data (2009-2022) and historical contract information, sourced from CapFriendly and Hockey Reference By integrating these diverse datasets, we seek to unveil patterns and correlations that influence contract values.

The objectives of this study are twofold: firstly, to develop a robust data integration pipeline that can efficiently process and merge diverse data types; and secondly, to create a predictive model that can accurately forecast NHL contract values based on player performance data.

The significance of this study extends beyond the immediate realm of contract prediction. It serves as a case study in the effective application of data integration techniques in sports analytics, potentially paving the way for similar analyses in other sports disciplines.

This paper is structured as follows: we begin with a comprehensive methodology section outlining our data collection, cleaning, and integration processes. This is followed by a detailed account of our model development and implementation strategies. We then present our results and discuss their implications, before concluding with key takeaways and potential avenues for future research.

Methodology

1. Data Collection

Sources

- **CapFriendly**
  - Used to gather all historic NHL contract signings from 2012-2022.
  - Used BeautifulSoup to scrape each table month by month due to the website's web-scraping prevention tools.
  - Specific URL here
- **Hockey Reference**
  - 2009-2022 due to needing n-3 previous seasons of statistics to predict year n
  - Much easier to scrape than CapFriendly due to the tables being plain HTML tables.
  - Benefited from using Pandas' `read_html()` function.
  - Specific URL here

Web scraping Techniques

During the process of collecting data some insights we gained were profound in how we set up and deployed our data pipeline. For example, normally you can retrieve data by changing the URL parameters. In this case, the web URL was almost a dummy URL. We later found an Ajax request call that had similar parameters in it. The next challenge was adapting to the pagination. If we ever called a page that didn't exist, the code would instantly break and we would lose our data. To combat this, we were able to use caching and try-except statements to safeguard our data collection.

With the Hockey Reference data source since the statistics were not paginated, we didn't have to iterate from month to month. We could get a whole year's data in one request call. All we had to do was change the year in the URL; https://www.hockey-reference.com/leagues/NHL_2022_skaters.html. One unique thing about this dataset was that the HTML column headers were built into the actual dataset. We were able to handle this in the Data Cleaning and Preprocessing stage.

Python code used to web scrape can be found here.

2. Data Cleaning and Preprocessing

Contract Data

PLAYER	PLAYER.1	AGE	POS	TEAM	DATE	TYPE	EXTENSION	STRUCTURE	LENGTH	VALUE	CAP HIT
Mark Pysyk	Mark Pysyk	31	RD	CGY	Dec. 2, 2023	Stnd (UFA)	nan	2-way	1	$775,000	$775,000
Samuel Montembeault	Samuel Montembeault	27	G	MTL	Dec. 1, 2023	Stnd (UFA)	✔	1-way	3	$9,450,000	$3,150,000
Jordan Gustafson	Jordan Gustafson	19	C	VGK	Nov. 29, 2023	ELC	nan	2-way	3	$2,572,500	$857,500
Patrick Kane	Patrick Kane	34	RW	DET	Nov. 28, 2023	Stnd (UFA)	nan	1-way	1	$2,750,000	$2,750,000
Justin Bailey	Justin Bailey	27	RW	SJS	Nov. 27, 2023	Stnd (UFA)	nan	2-way	1	$775,000	$775,000

Figure 1: Contract Raw Data

Above is what the contract data looked like as soon as we scraped it from CapFriendly. Some key features that we need to clean and preprocess are:

1. Handling `PLAYER.1` column
2. Standardizing the `DATE` to a more accessible data point
3. Filter `TYPE` to `Stnd (UFA)` to get the true contracts
4. Replacing `EXTENSION` values with binary (0,1)
5. Standardizing `VALUE` and `CAP HIT` to integers from strings
6. Adding an `ID` column for easy database storage

After implementing these changes our final contract dataset looks like this:

PLAYER	AGE	POS	TEAM	DATE	TYPE	EXTENSION	STRUCTURE	LENGTH	VALUE	CAP HIT	id
Patrick Kane	34	RW	DET	2023	Stnd (UFA)	0	1-way	1	2750000	2750000	PatrickKane2023
Danton Heinen	27	LW, RW	BOS	2023	Stnd (UFA)	0	1-way	1	775000	775000	DantonHeinen2023
Jonah Gadjovich	24	LW, RW	FLA	2023	Stnd (UFA)	0	1-way	1	810000	810000	JonahGadjovich2023
Noah Gregor	24	LW, RW	TOR	2023	Stnd (UFA)	0	1-way	1	775000	775000	NoahGregor2023
Austin Watson	31	RW, LW	TBL	2023	Stnd (UFA)	0	1-way	1	776665	776665	AustinWatson2023

Figure 2: Contract Cleaned Data

Statistics Data

Rk	Player	Age	Tm	Pos	GP	G	A	PTS	+/-	PIM	PS	EV	PP	SH	GW	EV	PP	SH	S	S%	TOI	ATOI	BLK	HIT	FOW	FOL	FO%	SEASON
1	Justin Abdelkader	21	DET	LW	2	0	0	0	0	0	0.0	0	0	0	0	0	0	0	2	0.0	19	9:18	0	3	4	3	57.1	2009
2	Craig Adams	31	TOT	RW	45	2	5	7	-3	22	0.1	1	1	0	0	5	0	0	47	4.3	391	8:41	20	67	8	13	38.1	2009
2	Craig Adams	31	CHI	RW	36	2	4	6	-3	22	0.1	1	1	0	0	4	0	0	38	5.3	314	8:43	16	53	6	10	37.5	2009
2	Craig Adams	31	PIT	RW	9	0	1	1	0	0	0.0	0	0	0	0	1	0	0	9	0.0	77	8:34	4	14	2	3	40.0	2009
3	Maxim Afinogenov	29	BUF	RW	48	6	14	20	-7	20	1.4	6	0	0	0	7	7	0	93	6.5	605	12:36	11	20	0	3	0.0	2009

Figure 3: Statistics Raw Data

As you can see from above, there are some key changes we will need to make to make our data friendly not only for machine learning but also for storing in our SQL database. Here is what we did:

1. Remove `Rk` as it has no value to us
2. Use `TOT` (Team) rows for players who played on multiple teams in one season (example: Craig Adams above)
3. Fix duplicate column names such as `EV`, `PP`, `SH`
4. Fix column names with % in them as they are not MySQL-compliant names
5. Add an `ID` column
6. Remove goalie statistics

PLAYER	AGE	TEAM	POS	GP	G	A	PTS	PLUSMINUS	PIM	PS	EVG	EVA	PPG	PPA	EVSH	PPSH	GWG	S	S_	TOI	ATOI	BLK	HIT	FOW	FOL	FO_	SEASON	id
Justin Abdelkader	22.0	DET	W	50.0	3.0	3.0	6.0	-11.0	35.0	-0.3	3.0	3.0	0.0	0.0	0.0	0.0	0.0	79.0	0.0	31800.0	635.0	20.0	152.0	148.0	170.0	0.5	2010.0	Justin Abdelkader2010.0
Justin Abdelkader	23.0	DET	W	74.0	7.0	12.0	19.0	15.0	61.0	1.7	7.0	11.0	0.0	0.0	0.0	1.0	1.0	129.0	0.1	54600.0	738.0	39.0	188.0	227.0	203.0	0.5	2011.0	Justin Abdelkader2011.0
Justin Abdelkader	24.0	DET	W	81.0	8.0	14.0	22.0	4.0	62.0	1.6	8.0	14.0	0.0	0.0	0.0	0.0	1.0	121.0	0.1	59820.0	739.0	42.0	148.0	239.0	213.0	0.5	2012.0	Justin Abdelkader2012.0
Justin Abdelkader	25.0	DET	W	48.0	10.0	3.0	13.0	6.0	34.0	1.7	10.0	3.0	0.0	0.0	0.0	0.0	0.0	96.0	0.1	42660.0	889.0	13.0	120.0	65.0	60.0	0.5	2013.0	Justin Abdelkader2013.0
Justin Abdelkader	26.0	DET	W	70.0	10.0	18.0	28.0	2.0	31.0	2.6	9.0	16.0	1.0	2.0	0.0	0.0	3.0	147.0	0.1	64200.0	917.0	31.0	172.0	23.0	32.0	0.4	2014.0	Justin Abdelkader2014.0

Figure 4: Statistics Cleaned Data

Python code used to clean and preprocess the data can be found here.

After formatting and filtering our data, we have `1006` contract observations and `9186` statistical observations to support.

Now we need to store the data.

3. Database Setup and Management

Due to familiarity and accessibility of documentation, we chose to go with a MySQL database. It's a very well-known DB and has a ton of support for some of the features we wanted to add later on.

Setup

Setup was fairly easy thanks to the Python packages; Flask and sqlalchemy. These packages allowed us to essentially create the DB base from our panda's data frames. Later we were able to go in and customize the relationships such as primary keys and foreign keys. Here are the two table schemas:

```
CREATE TABLE `contracts` (
  `id` varchar(255) NOT NULL,
  `PLAYER` varchar(255) DEFAULT NULL,
  `AGE` int DEFAULT NULL,
  `POS` varchar(255) DEFAULT NULL,
  `TEAM` varchar(255) DEFAULT NULL,
  `DATE` int DEFAULT NULL,
  `TYPE` varchar(255) DEFAULT NULL,
  `EXTENSION` int DEFAULT NULL,
  `STRUCTURE` varchar(255) DEFAULT NULL,
  `LENGTH` int DEFAULT NULL,
  `VALUE` int DEFAULT NULL,
  `CAP_HIT` int DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 5: Contract Schema

```
CREATE TABLE `stats` (
  `id` varchar(255) NOT NULL,
  `PLAYER` varchar(255) DEFAULT NULL,
  `AGE` float DEFAULT NULL,
  `TEAM` varchar(255) DEFAULT NULL,
  `POS` varchar(255) DEFAULT NULL,
  `GP` float DEFAULT NULL,
  `G` float DEFAULT NULL,
  `A` float DEFAULT NULL,
  `PTS` float DEFAULT NULL,
  `PLUSMINUS` float DEFAULT NULL,
  `PIM` float DEFAULT NULL,
  `PS` float DEFAULT NULL,
  `EVG` float DEFAULT NULL,
  `EVA` float DEFAULT NULL,
  `PPG` float DEFAULT NULL,
  `PPA` float DEFAULT NULL,
  `EVSH` float DEFAULT NULL,
  `PPSH` float DEFAULT NULL,
  `GWG` float DEFAULT NULL,
  `S` float DEFAULT NULL,
  `S_` float DEFAULT NULL,
  `TOI` float DEFAULT NULL,
  `ATOI` float DEFAULT NULL,
  `BLK` float DEFAULT NULL,
  `HIT` float DEFAULT NULL,
  `FOW` float DEFAULT NULL,
  `FOL` float DEFAULT NULL,
  `FO_` float DEFAULT NULL,
  `SEASON` float DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 6: Statistics Schema

Hosting Locally

Something cool, that we didn't discover until working on this project is that MySQL integrates well with Mac OSX which happened to be the operating system we hosted it on. Here you can manage the local instance with ease.

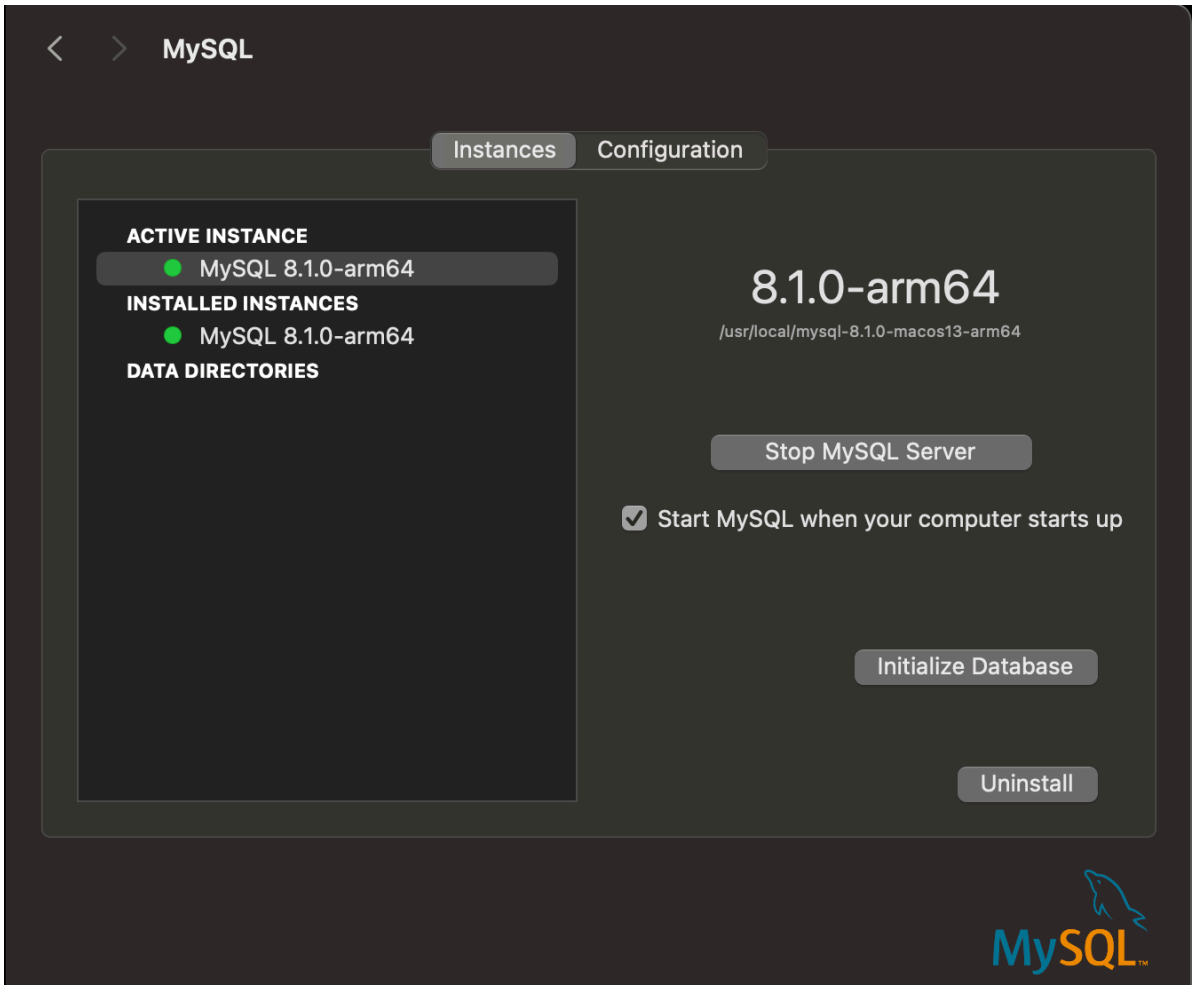

Figure 7: Mac OSX MySQLConfig

If this project ever needed to be hosted online, MySQL allows you to export the DB config which allows for an easy and seamless migration from local to online.

4. Data Integration

Merging the Data

Due to the time restraint we set on the statistics data to predict the contract value, the merging of the two datasets isn't as easy as originally thought. For example. Say a player has played 10 seasons from 2005-2015. If we want to predict his 2012 contract, we need to filter the data down to the 2009, 2010, and 2011 seasons. It might not seem like a big issue, but we need to do that step for each player we are predicting as well.

To overcome this challenge we utilize the Pandas package in Python to do a selective group-by so that we are only using the data from our restricted timeline and not the entire data set. We also want to standardize the `POS` column into positional groups. This will aid the predictive model in being able to separate centers, wingers, and defensemen from each other. We achieve this by using a simple if-then function in Python.

The code used to merge the data can be found here

Handling Heterogeneous Data

At first, we were worried that we were going to run into issues with heterogenous data, but as we worked through cleaning and merging the data, we ended up with homogenous data. That being said, in the future, there may come a time when two people have the same name or they play the same position in the same year. A new way to handle the heterogeneous data will need to be implemented. We would recommend either manually assigning a unique identifier for each player or getting access to data that has personal information that could be used to distinguish one from another.

After completing all of the steps necessary to merge the `contract` and `statistics` tables we are left with a dataset that looks like this:

AGE	CAP_HIT	DATE	PLAYER	POS	A	BLK	EVA	EVG	EVSH	...	HIT	PIM	PLUSMINUS	PPA	PPG	PPSH	PS	PTS	S	TOI
35	3500000	2021	Alexander Edler	D	0.432	2.870	0.249	0.081	0.000	...	2.119	0.995	-0.011	0.178	0.032	0.005	5.600000	0.546	2.259	1429.946
26	4500000	2021	Alexander Wennberg	C	0.338	0.551	0.268	0.045	0.005	...	0.419	0.202	0.081	0.056	0.025	0.015	1.766667	0.414	1.061	993.636
30	750000	2021	Alex Chiasson	W	0.191	0.377	0.106	0.126	0.010	...	0.995	0.503	-0.015	0.070	0.075	0.015	2.600000	0.402	1.402	847.236
27	750000	2021	Alex Galchenyuk	F	0.329	0.413	0.178	0.122	0.000	...	0.690	0.338	-0.258	0.150	0.094	0.000	3.366667	0.545	2.150	898.592
35	5000000	2021	Alex Goligoski	D	0.335	1.973	0.223	0.058	0.000	...	1.268	0.295	-0.134	0.103	0.027	0.009	5.433333	0.420	1.438	1342.500

Figure 8: Merged Dataset

If you look closely you can also see that we decided to standardize all of the statistics columns to a per-game basis so that players who played more games wouldn't be seen as more valuable than those who played less.

5. ML Model Development

After extensive model testing, we concluded that a Support Vector Machine (SVM) is the best-suited model for predicting NHL contracts. The model performed the best in both of our testing metrics; r-squared and RMSE. After feature selection and adjusting hyper-parameters, we were able to get the model to predict contracts within roughly $1,000,000. This produced an r-squared value of `0.7275` and the exact RMSE is `1,128,694`. Here are the results of all of our tests:

```
Linear Regression RMSE: 1278374.2772733346
Random Forest Regressor RMSE: 1240699.4033862366
Support Vector Regressor RMSE: 1128694.1788295216
Gradient Boosting Regressor RMSE: 1193948.427610758
XGBoost Regressor RMSE: 1193948.427610758
Ensemble (Voting Regressor) RMSE: 1131761.9137402903
Linear Regression R-squared: 0.6505189102715558
Random Forest Regressor R-squared: 0.6708144362989341
Support Vector Regressor R-squared: 0.7275666913952337
Gradient Boosting Regressor R-squared: 0.6951552147849385
XGBoost Regressor R-squared: 0.6951552147849385
Ensemble (Voting Regressor) R-squared: 0.7260837583647692
```

Figure 9: Modeling Results

Integration

After we selected our model, we were able to integrate the model directly into our API through an endpoint/route. The user can select which season they want to see predictions for by using `?season=` in the URL of the request. The API then returns real-time predictions using the pre-trained model, in seconds. This would come in handy if you needed to access the predictions from other external sources than just a dashboard.

6. API Integration

Why?

At first, we did not have any plans for an API to be built into our pipeline. However, we kept running into an issue as we were working together as a team; getting the most recent data from group member to group member. Whenever someone would make a change to the data cleaning and pre-processing, we needed to move that data to everyone else who was working on other sections that used the same data.

At first, we would download the data to CSVs and send them via email. At this moment we realized we needed a much better way to solve our problem. After some research, we concluded that an API would be best, as it could connect directly to our main data source (MySQL DB) and we would not have to worry about conflicts due to changing CSVs.

The original intent was to just have two routes/endpoints; one for the contract data and one for the statistics data. However, after reading the Flask documentation, we discovered there was so much more that we could do. Not only could we send and receive data, but we could also connect our machine-learning algorithm to an endpoint to allow for quick real-time predictions. While it may be overkill for the scale of this project, it certainly is an advancement in data integration that would apply in the real world.

Due to hosting expenses, we aren't able to host the API publicly, but here is a video that quickly demonstrates its capabilities.

7. Visualization and Dashboard

Tool Choice

There were a lot of options to pick from when selecting a visualization tool. However, we ended up going with Tableau as not only was it familiar to our group, but also had the best configuration for integrating without data. One of the key features was that it allowed us to connect our MySQL database as a direct data source which allowed for easy updating of data. Tableau also offers free hosting of dashboards on its Tableau Public service. These two key factors made it an easy choice over other options such as; matplotlib, ggplot, and others.

Dashboard Design

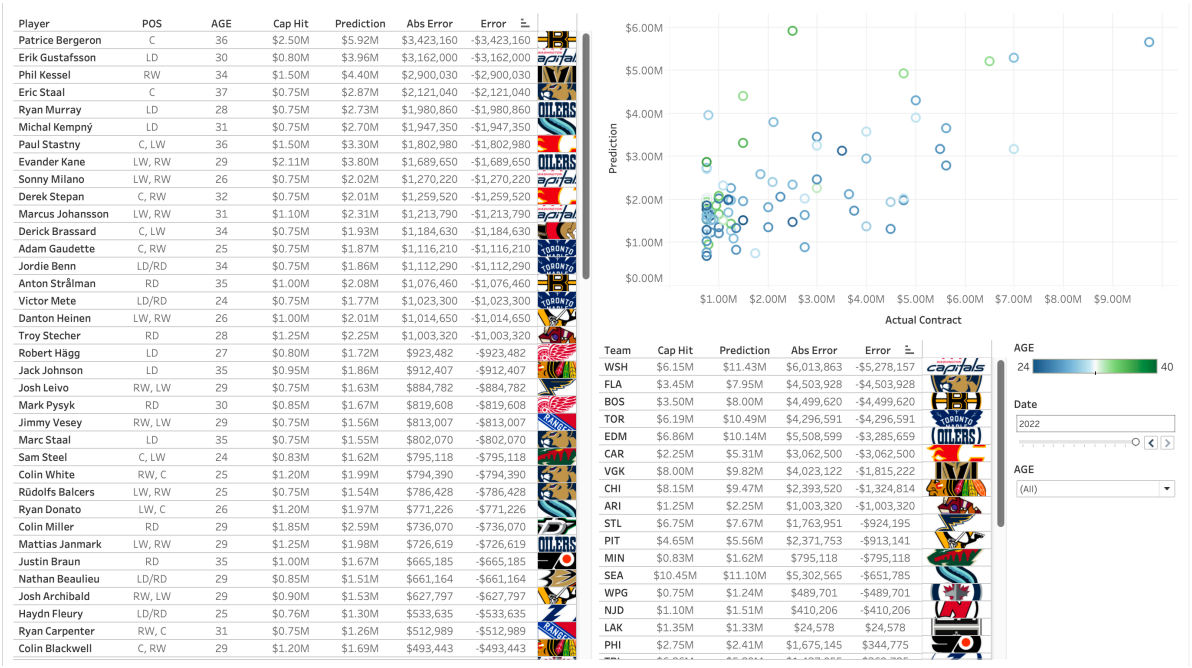

Figure 10: Dashboard

Creating the dashboard was a unique process. We wanted to capture the insights gained from not only our machine learning algorithm but also the effectiveness of our pipeline. The left half of the dashboard has each prediction with features that give insight into how accurate our prediction was. The top right plot displays each prediction but allows for direct comparison with other similar predictions. Finally, in the bottom right, we have the predictions broken down by team. This allows for quick analysis of what teams might have had the best contract signings versus the worst. The dashboard is also interactive. The user can select which season's data they would like to view as well as the age of the players.

Database Connection

Arguably the most valuable feature that Tableau has to offer is the ability to connect directly to our MySQL database. All we had to do was download a connector plugin and log in to the local MySQL server.

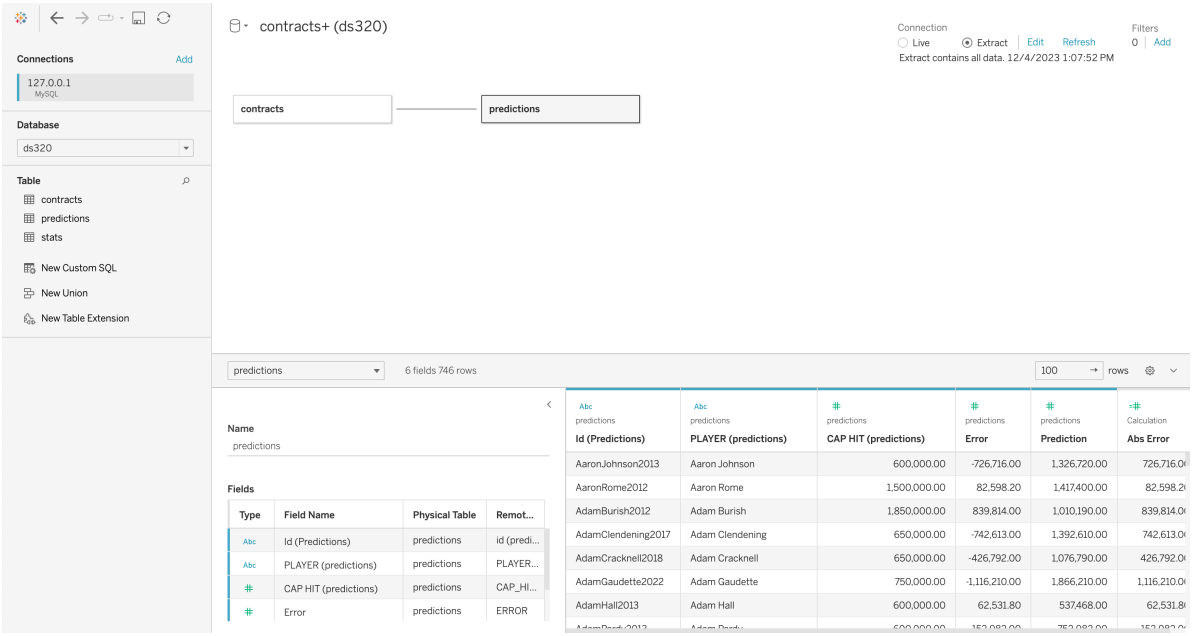

Figure 11: MySQL / Tableau Integration

As you can see above, the MySQL database seamlessly integrates into Tableau and allows for a ton of customization. One of the features we didn't need to implement but was available was the Custom SQL. This allows for the creation of new tables using SQL. Think of it almost as a view. You can also draw relationships between the tables inside Tableau to allow for easy fluidity in the data.

The interactive dashboard is hosted here thanks to the free hosting via Tableau Public.

8. Extra: Cloud-based Data Warehousing Exploration

Integration in Snowflake

Integration in Snowflake is very straightforward as it's also based on SQL queries. The level of difficulty of integration is simpler than what we used in the Jupyter Notebook given that Snowflake allows importing CSV files so we don't have to create the schema ourselves. In addition, it's very easy to create new tables in Snowflake as there's the create table keyword can be directly used on top of a SQL query.

Data Storage in Snowflake

The database schema is stored in a Snowflake virtual warehouse that ultimately supports the execution of SQL queries and allows external notebook files to access the schema in Snowflake sessions using Snowpark packages. Aside from the local notebook file, there are also collaborative companies like Hex that provides online notebook and can directly access the schema by establishing a connection.

Snowflake Dashboard

Snowflake provides the user easy access to create a simple dashboard using SQL queries or Python queries. The dashboard that we created for our data supports an analysis of the relationship between player position and a variety of other variables presented in different data visualizations.

Figure 12: Snowflake Dashboard

Machine Learning in Snowflake

Adding machine learning to our Snowflake enviornment was actually rather easy. Snowflake has "worksheets" which are really just notebooks. Here we were able to use the exact same code from our previous ML model and run it on the data in our Snowflake DB. Heres a view of what the worksheets look like:

Disadvantage of Snowflake

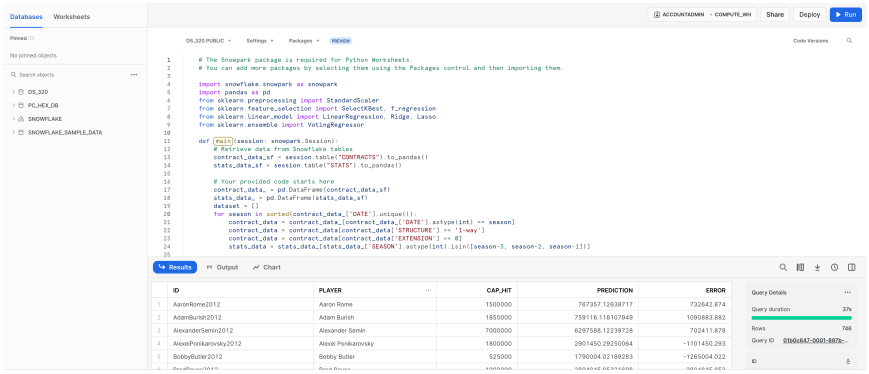

Figure 13: Snowflake Worksheet

The biggest disadvantage of using Snowflake is that it can be costly, the cost of storing our data is about $3.00 per day. Other than the cost, the only disadvantage is the low customization capability in Snowflake API. However, this could be due to our limited experience with Snowflake.

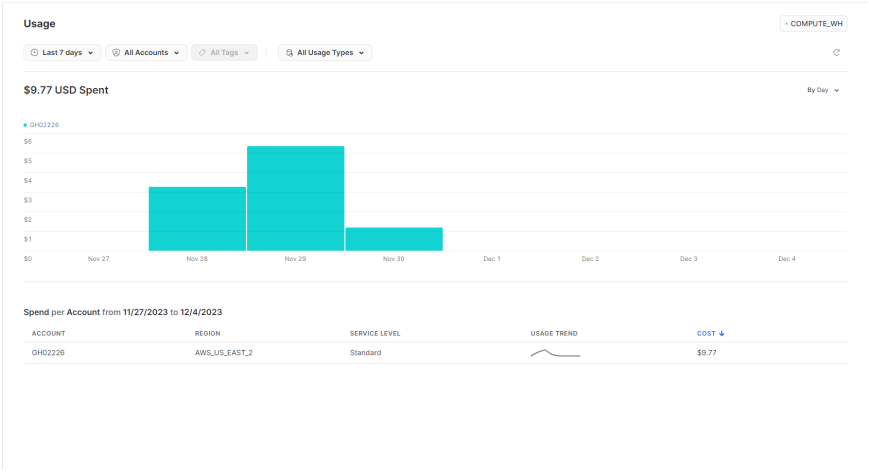

Figure 14: Snowflake Usage

Conclusion

Contribution

References

Appendices

Github Repo