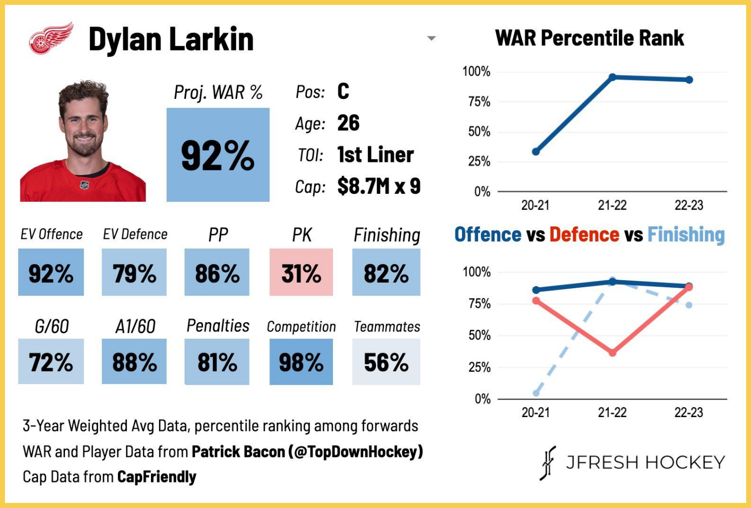
ICE HOCKEY

Hockey Project Team - Player Cards

Goals for the Project

- The overall goal for the project was to create a way to calculate Production, Offense, Defense, and Transition scores for each player, which would then be used to create an overall score.
  - Provides a “easier” visual for coaches that don’t want to go through all of the stats.
- Given the 22-23 USHL stats and two websites for inspiration.
- Problem: there were no scores given so we couldn’t train the set given…



Player	Team	Total TOI(sec)	Total TOI(min)	GP	TOI/GP(sec)	TOI/GP(min)	G	A	PTS	+/-	S	Sh%
Aaron Pionk	WAT	53056.85	884:17:00	60	884.2808333	14:44	8	11	19	17	122	0.06557377
Adam Cardona	OMA	50620.61667	843:41:00	61	829.8461749	13:50	2	11	13	-8	74	0.027027027
Adam Kleber	LIN	37887.96667	631:28:00	55	688.8721212	11:29	0	7	7	-2	59	0%
Adam Zlnka	SF	39031.9	650:32:00	56	696.9982143	11:37	6	9	15	-20	86	0.069767442
Adyn Merrick	YNG	194.2	3:14	1	194.2	3:14	0	0	0	0	0	0%
Aidan Dyer	CHI	290.05	4:50	1	290.05	4:50	0	1	1	-1	1	0%
Aidan Park	GB	2337.833333	38:58:00	3	779.2777778	12:59	0	0	0	-3	2	0%
Aiden Shirey	WAT	372	6:12	1	372	6:12	0	0	0	0	0	0%
Aiden Van Rooyan	DM	46026.7	767:07:00	59	780.1135593	13:00	2	10	12	2	58	0.034482759
Aleksi Kivioja	OMA	36056.96667	600:57:00	57	632.5783626	10:33	9	4	13	-1	84	0.107142857
Alex Bales	TC	3668.766667	61:09:00	8	458.5958333	7:39	1	0	1	-3	6	0.166666667
Alex Bump	TC	36005.13333	600:05:00	47	766.0666667	12:46	9	11	20	1	129	0.069767442
Alex Lunski	OMA	4386.566667	73:07:00	9	487.3962963	8:07	0	1	1	0	11	0%
Alex Pineau	DM	43309.95	721:50:00	56	773.3919643	12:53	1	6	7	-4	66	0.015151515
Alex Weiermair	U18	13703.8	228:24:00	23	595.8173913	9:56	4	2	6	2	30	0.133333333
Alexander Rybakov	SF	35305.38333	588:25:00	42	840.6043651	14:01	1	5	6	-2	40	0.025
Amine Hajibi	GB	2375.45	39:35:00	6	395.9083333	6:36	2	1	3	2	6	0.333333333
Andon Cerbone	YNG	46537.86667	775:38:00	63	738.6962963	12:19	15	12	27	9	103	0.145631068

Implementation

WOI = While On Ice
PAR = Play After Rate
PA = Play After

	Production	Offense	Defense	Transition
Stat(s) Tested On	Expected Goals For WOI	Expected Goals For WOI	Expected Goals Against WOI	Carry-Out with PAR, Pass-Out with PAR, Controlled Exit with PAR, Dump Out Success Rate
Compiling Stats	Goals, Assists, Points	Shots, Successful Pass to Slot For WOI, PDP/20, OGP/20	+/-, Shot Attempts Against WOI, Shots on Net Against WOI, DZ Rebound Recovery Rate WOI	Carry-Outs with PA, Total Carry-Out Attempts, Pass-Outs with PA, Controlled Exits, Successful Dump Out Attempts

Sample Implementation - Production

```
# Load data
data = pd.read_csv('2223_ushl_stats.csv', encoding='latin-1')

# Stat selection
features = ['G', 'A', 'PTS']
X = data[features]

# Target variable (Expected Goals For WOI)
new_Y = ['Expected Goals For WOI']
y = data[new_Y]

data = data.dropna(subset=['Expected Goals For WOI'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the XGBoost model
production_model = xgb.XGBRegressor()

# Train the model
production_model.fit(X_train, y_train)

# Make predictions
production_war_predictions = production_model.predict(X_test)

# Evaluate the model using Mean Absolute Error (MAE)
mae_production_war = mean_absolute_error(y_test, production_war_predictions)
print(f'Mean Absolute Error: {mae_production_war:.2f}')
```

1) Load the data from the csv file
2) Using Goals, Assists, and Points to target the Expected Goals For WOI
3) Split the data randomly into training and testing sets
4) Trained the model
5) Make predictions
6) Evaluate with Mean Absolute Error

Sample Implementation - Production

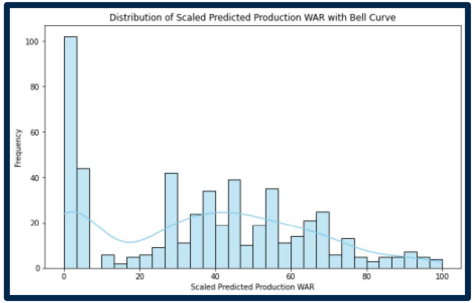
```
# Calculate WAR for each player in the dataset using a loop
player_war = []

for index, player_row in data.iterrows():
    player_data = player_row[features].values.reshape(1, -1)
    player_name = player_row['Player']
    player_production_war = production_model.predict(player_data)[0]
    player_war.append((player_name, player_production_war))

# Create a new DataFrame with PlayerName and WAR
war_df = pd.DataFrame(player_war, columns=['Player', 'Predicted Production WAR'])

# Merge the calculated WAR with the original data
data = data.merge(war_df, on='Player', how='left')

# Extract the predicted production WAR values
predicted_production_war = data['Predicted Production WAR']

# Use Min-Max scaling to scale the values from 0 to 100
scaler = MinMaxScaler(feature_range=(0, 100))
data['Scaled Production WAR'] = scaler.fit_transform(predicted_production_war.values.reshape(-1, 1))

# Display the scaled values
scaled_values = data[['Player', 'Position', 'Scaled Production WAR']]
print(scaled_values.to_string(index=False))
```

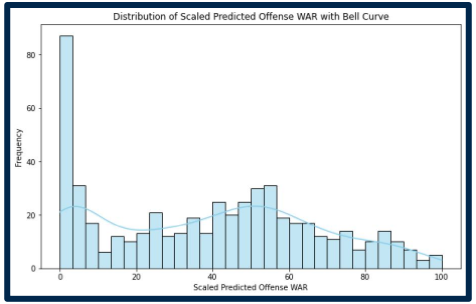
7) Calculate the Production score for each player in the dataset
8) Put that score in a new dataframe
9) Used Min-Max scaling to scale the values from 0 to 100
10) Print the scores

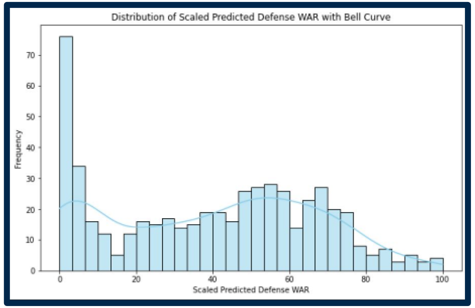
Overall Scores

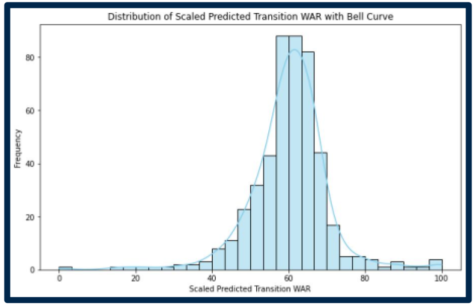
- After calculating all of the previous scores, they could then come together to create an overall score for each player.

- The Production and Offense scores were weighted higher for forwards, the Defense scores were weighted higher for defensemen.









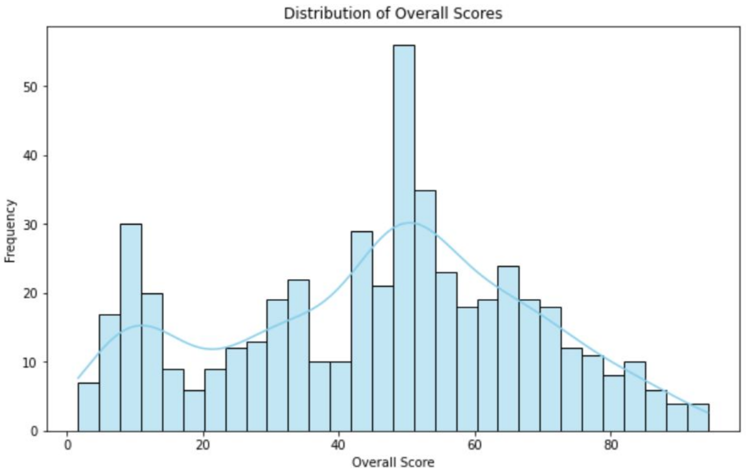
Sample Implementation - Overall

```python
# Create an empty array for overall scores
overall_scores = np.zeros(len(data))

# Loop through each unique player in the DataFrame
for player_name in data['Player'].unique():
    player_rows = data[data['Player'] == player_name]

    weight_production = 0
    weight_offense = 0
    weight_defense = 0
    weight_transition = 0

    # Check the player's position
    player_position = player_rows['Position'].iloc[0]
    if player_position == "F":
        weight_production = 0.45
        weight_offense = 0.45
        weight_defense = 0.05
        weight_transition = 0.05
    elif player_position == "D":
        weight_production = 0.05
        weight_offense = 0.05
        weight_defense = 0.8
        weight_transition = 0.1
```

```python
    # Calculate the overall score for the current player
    overall_score = (
        weight_production * player_rows['Scaled Production WAR'].iloc[0] +
        weight_offense * player_rows['Scaled Offense WAR'].iloc[0] +
        weight_defense * player_rows['Scaled Defense WAR'].iloc[0] +
        weight_transition * player_rows['Scaled Transition WAR'].iloc[0]
    )

    # Assign the overall score to all rows corresponding to the current player
    overall_scores[player_rows.index] = overall_score

# Add the overall scores to your DataFrame
data['Overall_Score'] = overall_scores

# Print the names and overall scores
scaled_values = (data[['Player', 'Position','Overall_Score']])
print(scaled_values.drop_duplicates().to_string(index=False))
```

1) Empty array for the overall scores
2) Looped through each player to check for their position
3) Based on the position, calculate the overall score using the weights

Player Score Distribution



	Player	Position	Scaled Production WAR	Scaled Offense WAR	Scaled Defense WAR	Scaled Transition WAR	Overall_Score
344	Nick Moldenhauer	F	100.00	100.00	79.06	63.70	94.28
22	Antonio Fernandez	D	90.89	91.38	99.44	53.11	93.15
196	Jack Harvey	F	98.55	96.78	80.49	55.37	91.72
44	Boston Buckberger	D	95.23	95.02	95.24	57.31	91.43
167	George Fegaras	D	68.16	65.45	100.00	67.05	90.07
...	...	...	...	...	...	...	...
352	Noah Eyre	F	0.00	0.26	1.69	42.32	4.50
65997	Xavier Veilleux	D	0.00	2.07	3.54	17.56	4.44
119	Daniel D'Alessandro	F	0.00	2.35	1.64	33.33	4.43
244	joey arnold	F	0.00	0.44	1.28	13.49	1.65
456	William Renfrew	F	0.00	3.64	1.88	0.00	1.65

501 rows × 7 columns

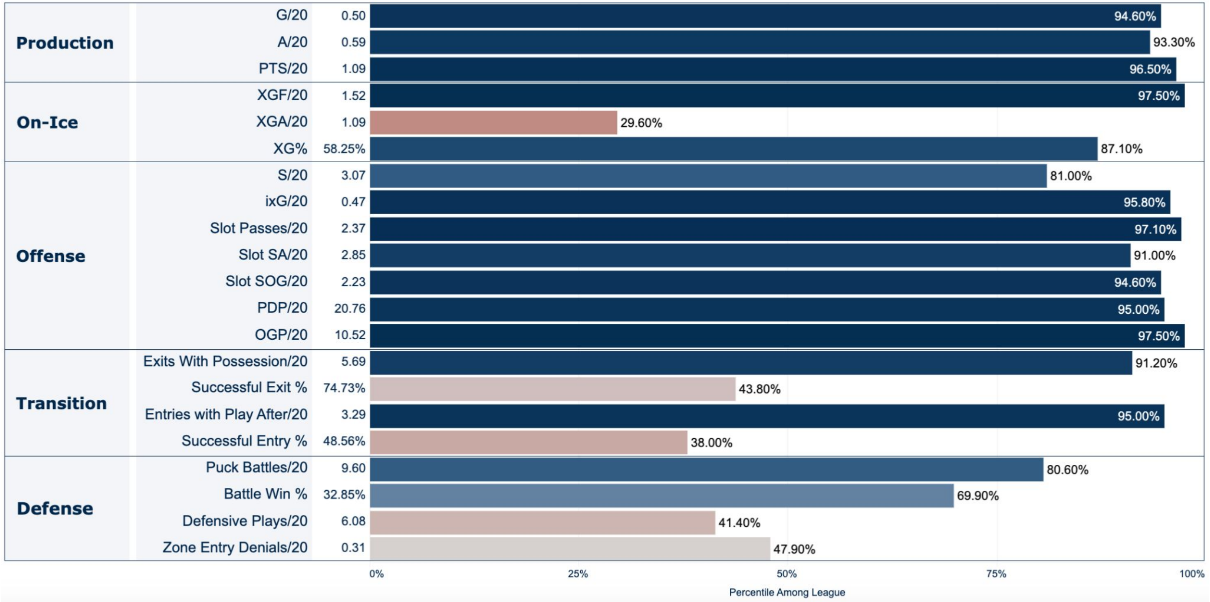
- There is a relatively normal distribution of player overall scores with a peak around 50
  - There is a smaller peak around 15 because of the players who played very little minutes
- Top scorers finished in the low to mid 90s, lowest players between 1 and 5
- Defenseman have a higher average score than forwards but that is likely due to the higher number of forwards who played very little

```
count    501.000000
mean      45.595269
std       22.345620
min        1.650000
25%       29.990000
50%       49.470000
75%       61.810000
max       94.280000
Name: Overall_Score, dtype: float64
```

Position	Overall_Score
D	51.72
F	42.36

Nick Moldenhauer (F)

Overall:	Production:	Offense:	Transition:	Defense:
96.95	100.00	100.00	53.77	73.09

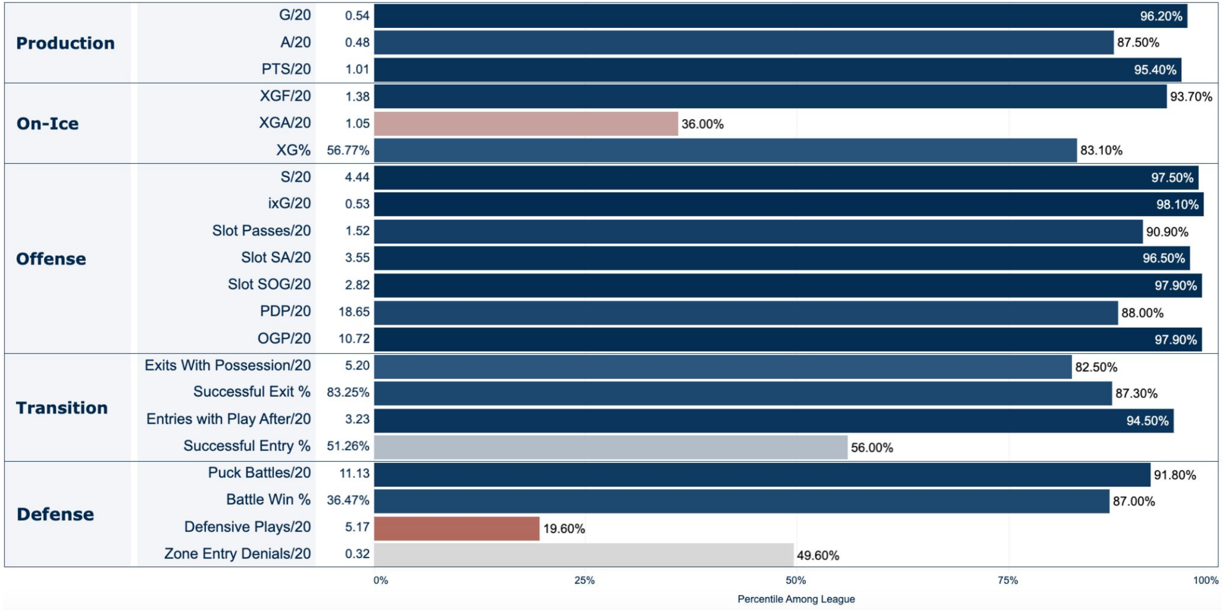


Notes:

- Only player to have full production and offense marks.
- Highest overall player and forward.

Garrett Schifsky (F)

Overall:	Production:	Offense:	Transition:	Defense:
75.41	79.32	78.33	68.38	57.89

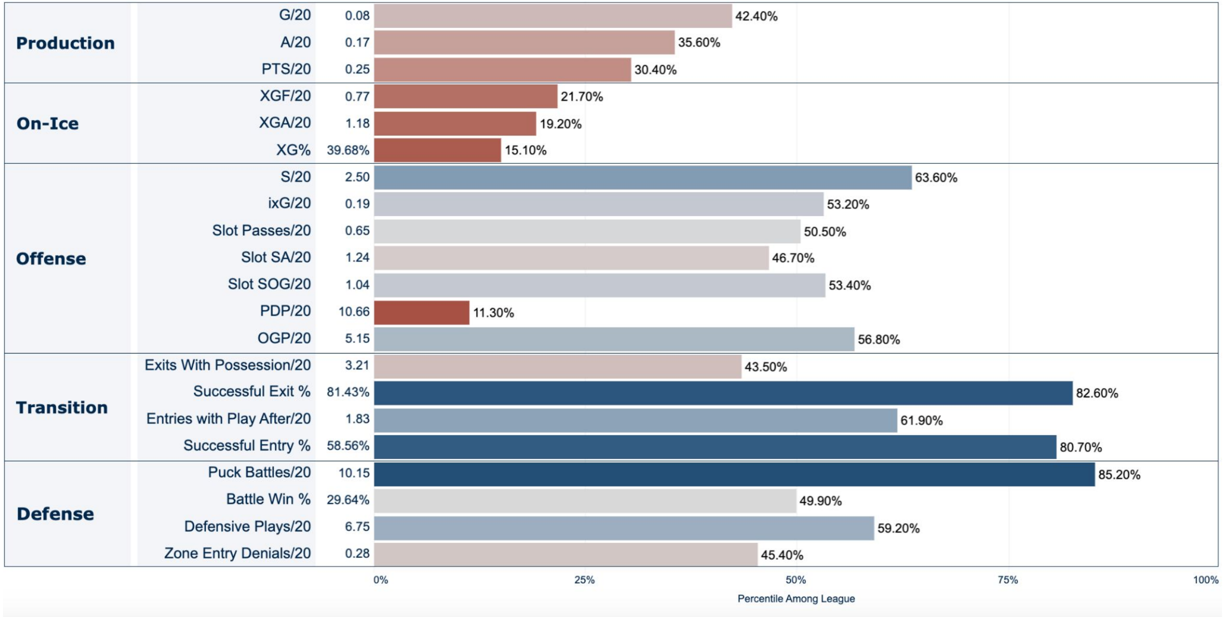


Notes:

- Exposes some limitations in our single testing stat (Expected Goals For)....
- Discussed more in the limitations.

Tanner Rowe (F)

Overall:	Production:	Offense:	Transition:	Defense:
45.52	34.19	48.28	68.51	75.70

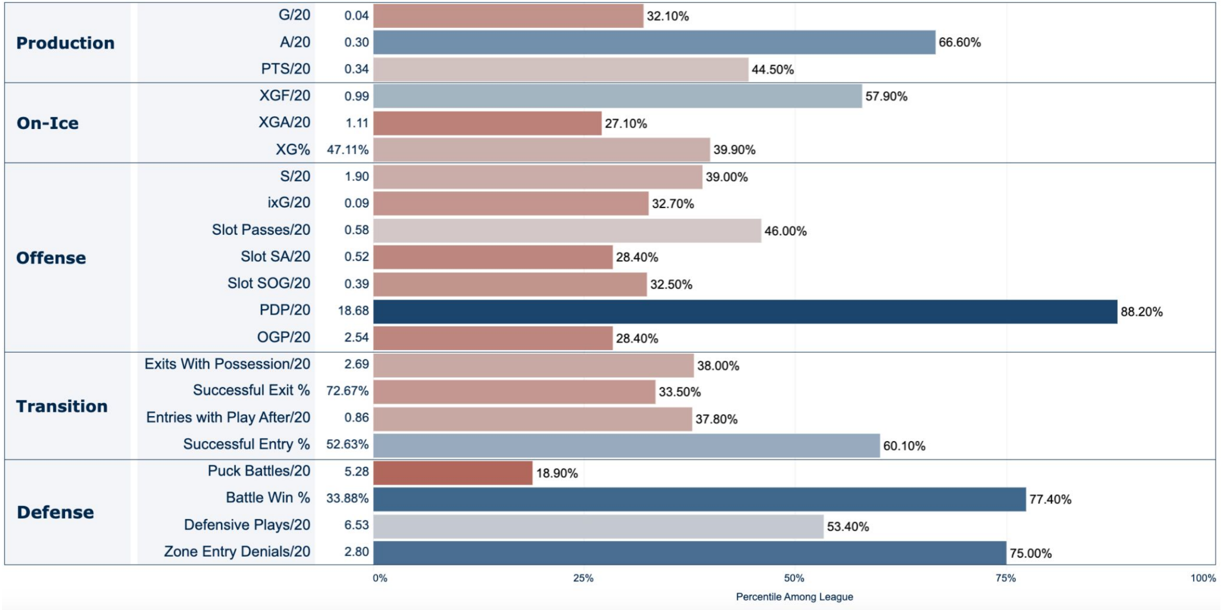


Notes:

- Very defensive forward, which is reflected in the Tableau and the scores.
- Exposes another weakness for overall forward scores.

Tyler Dunbar (D)

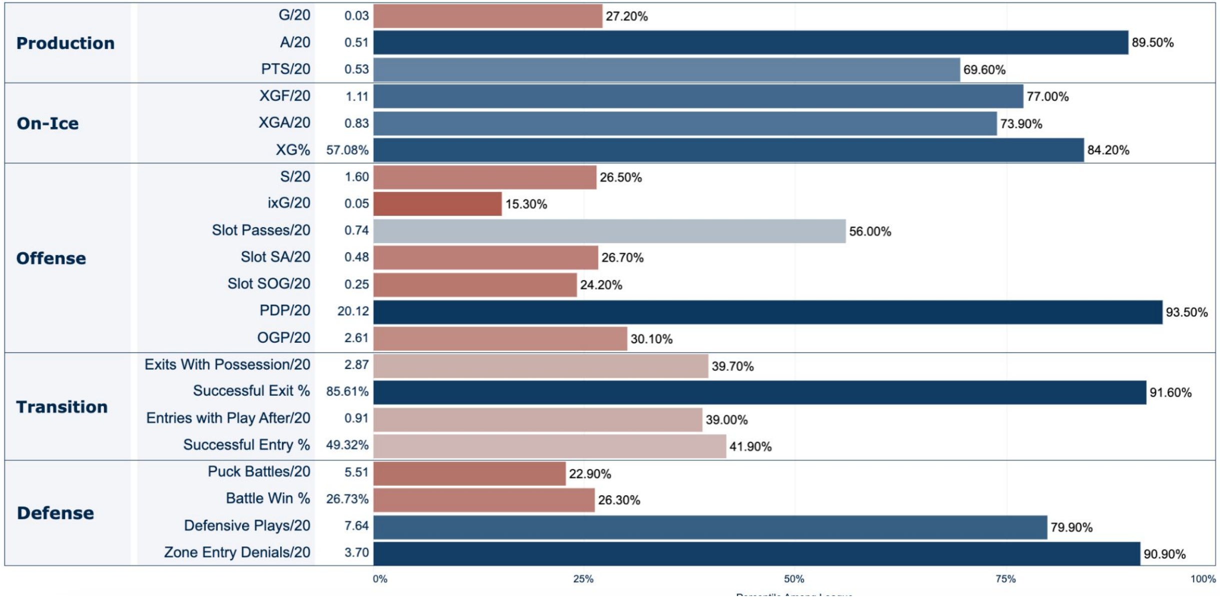
Overall:	Production:	Offense:	Transition:	Defense:
91.48	83.96	84.35	63.37	96.28



Notes:
- One of the highest rated defensemen
- Highlights the importance in goals for and goals against while the player is on ice

Ben Robertson (D)

Overall:	Production:	Offense:	Transition:	Defense:
64.57	79.79	80.35	74.50	61.43



Notes:

- An "average" defensemen with his score, but a very offensive defensemen
- Thus, his overall score is impacted because of its focus on defense

Limitations

- In projects like this, there are typically past scores that the algorithm can train on to make the new/future scores. However, we were just given one season of stats to work with, so the output is less accurate.
- For example, even though Moldenhauer and Schifsky have similar offensive stats, Schifsky's production and offense scores are lower (and therefore the overall score), because his Expected Goals for is 10 less than Moldenhauer's, thus showing the limitations of training against one stat.
- Another thing to note is the bell curves don't look "traditional", as there are many outliers towards the bottom: aka players that have only played a shift or two. Their scores are obviously very low.