Quantifying Player Value:

Machine Learning Models for NHL Skater Salary Prediction

Ben Gronbach

School of Engineering and Computer Science

Bioengineering Department

University of the Pacific

Stockton, California

November 5, 2025

Abstract:

NHL teams constantly strive to quantify the dollar value of player performance. In a league maintaining a tight salary cap, contract negotiations are a constant battle between agents trying to secure the highest salary for their players, and teams trying to secure high on-ice production for the lowest possible cost. Kirill Kaprizov secured a record-breaking contract worth 136 million dollars prior to the 2026 season, begging the question, how much is production really worth?

This experiment will use various machine learning models to predict a player's Average Annual Value (AAV) based on their raw, on-ice stats. This can be used to determine whether players are undervalued or overvalued, solely based off of their statistics. This metric would be crucial for NHL teams as it would output a quantitative value which can be used in contract negotiations, player scouting, salary cap management, and predicting player trends.

The data used to train these models is derived from player data from the 2024-2025 NHL season.

Figure 1. Kirill Kaprizov loading up a shot [1]

Model 1: Random Forest Regression

A random forest is an ensemble model of machine learning, meaning it takes into account the results of multiple smaller models to make a final prediction. It works by building a set number of decision trees, each of which is trained on a random subset of the data. To achieve a final output, the algorithm calculates the average of all decision tree outputs. This method is easy to use, versatile with both numerical and categorical data, and is unlikely to experience overfitting due to its random nature.

Python Code

```
import pandas as pd from sklearn.ensemble import RandomForestRegressor from sklearn.model_selection import train_test_split, RandomizedSearchCV from sklearn.metrics import r2_score, mean_absolute_error from sklearn.preprocessing import MinMaxScaler import matplotlib.pyplot as plt

Load dataset

csv_path = r"C:\Users\bkgro\OneDrive\Documents\Tendy Analysis\Skater_stats_2024.csv" use_columns = ["AAV_num","Player.name","Length","G","A","+/-","Awards","PIM","EVG","PPG","SHG","GWG","EV","PP","SH","SOG","SPCT","TSA","ATOI.min","FOW","FOL","FO%","BLK","HIT","TAKE","GIVE"] df = pd.read_csv(csv_path, usecols=use_columns)

Ignore Descriptive Columns

names = df['Player.name'] df = df.drop(columns=["Player.name"], errors="ignore") df['Awards'] = df['Awards'].notna().astype(int)

Define target value

target_column=('AAV_num') X=df.drop(columns=[target_column]) y=df[target_column]

Split Data

X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.3, random_state=42)

Scale Data

scaler=MinMaxScaler() X_train=scaler.fit_transform(X_train) X_test=scaler.transform(X_test)

Create RF model

model=RandomForestRegressor(n_estimators=300, max_depth=5, min_samples_split=10, min_samples_leaf=4, bootstrap=True, max_samples=0.7, random_state=42) model.fit(X_train, y_train)

Model Predictions

y_pred=model.predict(X_test)

Evaluate Accuracy

r2=r2_score(y_test, y_pred) mae=mean_absolute_error(y_test, y_pred) print(f"R^2 Score: {r2:.3f}") print(f"Mean Absolute Error: {mae:.3f}")

Plot Results
```

```
plt.figure(figsize=(7,6)) plt.scatter(y_test, y_pred, s=30, alpha=0.7) plt.xlabel("Actual AAV (tens of millions of dollars)")
plt.ylabel("Predicted AAV (tens of millions of dollars)") plt.grid(False) max_val = max(max(y_test), max(y_pred)) min_val =
min(min(y_test), min(y_pred)) plt.plot([min_val, max_val], [min_val, max_val], color='red', linewidth=2, label='Perfect Prediction')

Annotate each point with Player name

for i, idx in enumerate(y_test.index): x = y_test.iloc[i]

 yv = y_pred[i]

 skaterName = names.loc[idx] plt.text(x, yv, skaterName, fontsize=7, alpha=0.8, ha='left', va='bottom')

Compute correlations

corr = df.corr(numeric_only=True)

Sort features by correlation with target

corr_target = corr[target_column].abs().sort_values(ascending=False) print(corr_target)

print(model.score(X_train, y_train)) # Training R² print(model.score(X_test, y_test)) # Test R²

plt.tight_layout() plt.show()
```

Results

```
R² Score: 0.804
Mean Absolute Error: 826559.381
AAV_num     1.000000
Length      0.828586
A           0.685756
TSA         0.652168
EV          0.650829
SOG         0.641178
PP          0.617867
GIVE        0.608229
G           0.594037
TAKE        0.578749
PPG         0.556385
EVG         0.553658
GWG         0.514353
BLK         0.395303
Awards      0.390828
FOW         0.290342
FOL         0.286342
PIM         0.282036
SH          0.251258
SHG         0.225706
ATOI.min    0.212515
+/-         0.191263
HIT         0.161629
SPCT        0.130069
FO%         0.001926
Name: AAV_num, dtype: float64
0.8859282701169988
0.8042575447660841
```

Figure 2. Accuracy Metrics and Relevant Features

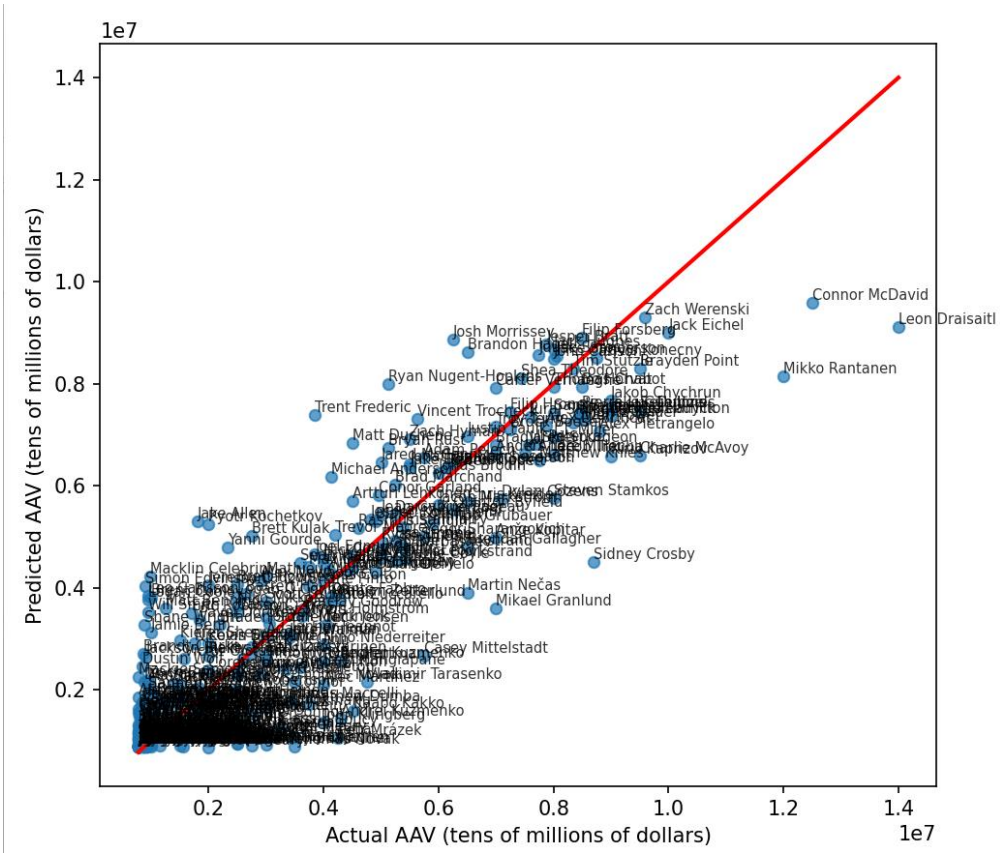

Figure 3. Actual AAV vs. Random Forest Predicted AAV

```
0.973425463954385 8
0.8028770383908002
```

Figure 4. Overfitted Model Output. Training data R^2 (top) vs. Test data R^2 (bottom).

Discussion

The random forest model shows relatively successful results, achieving an impressive R^2 score of 0.804, meaning that the model explains about 80% of the variance. This is substantial considering that players are often inconsistent and are prone to not performing up to the standards their AAV would expect. The model also scored a mean average error of ~826,000, meaning that on average, the predicted AAV was around $826,000 off from the actual AAV. While certainly not a perfect model, this represents a relatively small deviation given that player AAVs in the dataset range from approximately $2 million to $10 million.

The most relevant features the model found were quite surprising (see figure 1). It found that the two most relevant features in predicting AAV were the length of a player's contract and their assist total, with goal total being 8th. This may be due to the fact that goal scoring in the NHL is often random, and assists are much more consistent as the best players are exceptional at setting teammates up for goals. As for contract length, teams will often extend the best players for the maximum time allowed, 8 years, while less productive players are often bounced between teams on one-year deals.

As shown in figure 2, the model was able to find a consistent pattern among a large majority of the players, however it got confused with the high salaries of the best players in the league. Although these outliers are statistically the most productive players in the league, this indicates that they are overpaid relative to their production. The model predicted that Connor McDavid, widely regarded as the top player in the NHL currently, would have the highest AAV. However, it was unable to predict the large gap between a top player like McDavid and say, an NHL all-star. Players such as Connor McDavid, Leon Draisaitl, and Mikko Rantanen are certainly among the most elite talents in the NHL, however their teams pay an extra luxury tax to retain them, as their statistical production does not fully warrant their high value.

In attempt to improve the accuracy of the model, RandomSearchCV was utilized to achieve the best model using hyperparameter tuning. Multiple parameters were tested and evaluated using this command. The number of trees in the forest, the maximum depth of each tree, and the number of features considered in each split within the tree were all evaluated using RandomSearchCV, and output the best model. However, this led to the model overfitting, as it scored a 0.97 R^2 score in the training data, while only scoring a 0.80 R^2 on the test data (see figure 3). This large difference in accuracy indicates that the model memorized the training data and applied the same logic to the test set, leading to a lower score. To combat this, hyperparameters were tuned manually to avoid the model from outputting an overfitted algorithm. Max_depth, min_samples_split, n_estimators, min_samples_leaf, max_samples, and bootstrap were all manually set and adjusted while carefully watching the training and test set R^2 scores. This prevented overfitting, as the R^2 split was now 0.88 for training data and 0.80 for testing data (See bottom of figure 2), which is typically considered a normal split. However, preventing overfitting did not improve the accuracy of the model. This indicates that the model hit a data ceiling, meaning that it already learned all possible patterns from this dataset, and the accuracy is capped by the content of the dataset. Further attempts to tune parameters, engineer features, and reduce noise in the data did improve accuracy whatsoever. Since players are not always going to produce in accordance to their salaries, it is reasonable to expect that the model will hit a ceiling in its predictive ability.

Model 2: XGBoost

XGBoost stands for extreme gradient boosting. Like a random forest, it is an ensemble model which combines the results of multiple decision trees. The difference is that it sequentially learns from the error of each decision tree. Therefore, the model improves its accuracy over time by adjusting based on the error calculated in prior trees. It is often very accurate with large amounts of data and can be used in numerical or categorical data. However, XGBoosting is prone to overfitting, meaning the model memorizes training data rather than learning underlying patterns which can predict future data points.

Python Code

```
import pandas as pd from xgboost import XGBRegressor, callback from sklearn.model_selection import train_test_split, RandomizedSearchCV from sklearn.metrics import r2_score, mean_absolute_error from sklearn.preprocessing import MinMaxScaler import matplotlib.pyplot as plt

Load dataset

csv_path = r"C:\Users\bkgro\OneDrive\Documents\Tendy Analysis\Skater_stats_2024.csv" use_columns = ["AAV_num","Player.name","Length","G","A","+/-","PIM","EVG","PPG","SHG","GWG","EV","PP","SH","SOG","SPCT","TSA","ATOI.min","FOW","FOL","FO%","BLK","HIT","TAKE","GIVE"] df = pd.read_csv(csv_path, usecols=use_columns)

Store player names separately

names = df['Player.name'] df = df.drop(columns=["Player.name"], errors="ignore")

Define target and features

target_column = 'AAV_num' X = df.drop(columns=[target_column]) y = df[target_column]

Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

Scale data

scaler = MinMaxScaler() X_train = scaler.fit_transform(X_train) X_test = scaler.transform(X_test)

Create initial XGBoost model

xgb = XGBRegressor( objective='reg:squarederror', n_estimators=350, learning_rate=0.01, reg_alpha=0.3, reg_lambda=1.5, max_depth=3, subsample=0.8, colsample_bytree=0.8, random_state=42 )

Train Model

xgb.fit( X_train, y_train, eval_set=[(X_test, y_test)], )

Model predictions

y_pred = xgb.predict(X_test)
```

Evaluate performance

r2 = r2_score(y_test, y_pred) mae = mean_absolute_error(y_test, y_pred)

print(f"Training R2 score: {xgb.score(X_train, y_train)}") # Training R^2 print(f"Test Set R2 score: {xgb.score(X_test, y_test)}") # Test R^2 print(f"Mean Absolute Error: {mae:.3f}")

Plot results

plt.figure(figsize=(7,6)) plt.scatter(y_test, y_pred, s=30, alpha=0.7) plt.xlabel("Actual AAV (tens of millions of dollars)") plt.ylabel("Predicted AAV (tens of millions of dollars)") plt.grid(False) max_val = max(max(y_test), max(y_pred)) min_val = min(min(y_test), min(y_pred)) plt.plot([min_val, max_val], [min_val, max_val], color='red', linewidth=2, label='Perfect Prediction')

Annotate player names

for i, idx in enumerate(y_test.index): plt.text(y_test.iloc[i], y_pred[i], names.loc[idx], fontsize=7, alpha=0.8, ha='left', va='bottom')

plt.tight_layout() plt.show()

Results

```
Training R2 score: 0.880218136609253
Test Set R2 score: 0.8052289880636813
R² Score: 0.805
Mean Absolute Error: 850222.041
```

Figure 5. XGB Accuracy Metrics

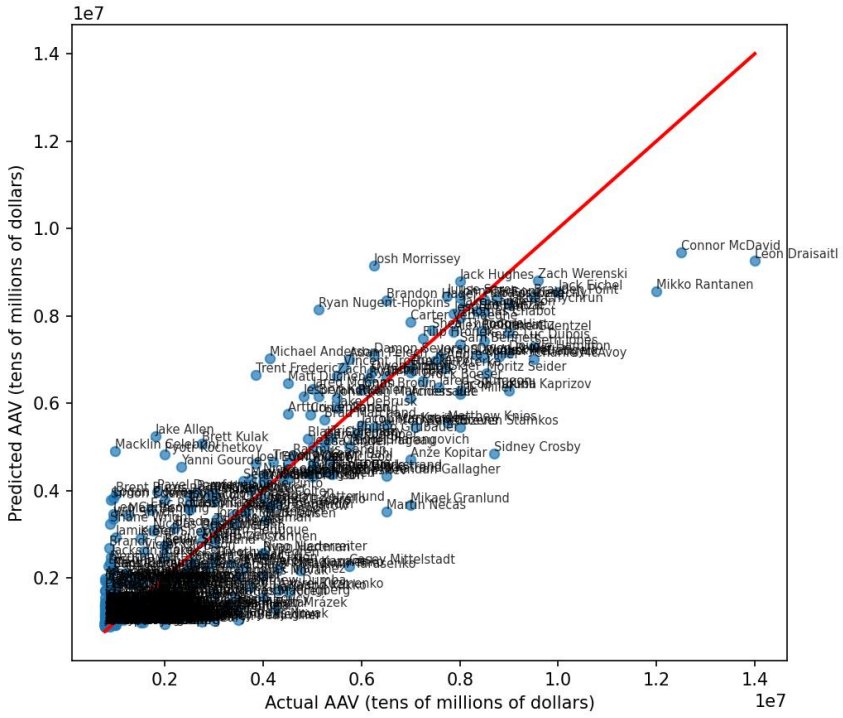

Figure 6. XGB Predicted AAV vs. Actual AAV

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
0.9568264193135109
0.7983791040335959
R² Score: 0.798
Mean Absolute Error: 824884.253
Best Parameters: {'reg_lambda': 1, 'reg_alpha': 0.1, 'min_child_weight': 5, 'max_depth': 5, 'learning_rate': 0.01, 'gamma': 0.3}
```

Figure 7. Overfitted XGB Model Output

Discussion

Like the random forest model, the original XG Boost model experienced overfitting, with a training R^2 score of 0.95, and a test R^2 score of 0.78 (see figure 7). GridSearchCV was used to find the best parameters, however this lead to the model adapting to the training data and memorizing it, explaining the high split in R^2 scores. Again, parameters were manually adjusted to limit overfitting, and it was found that much of the issue was due to a high number of trees, or n_estimators. Since XG Boosting learns from each tree, the model learned too much about the training data as it went through too many iterations of XG Boosting. By limiting the amount of trees to 350 (originally 1000), and lowering the learning rate of the model, overfitting was lowered to a 0.88-0.80 split between the test and training R^2 scores (see figure 5).

When using an XG Boost model, it is natural to expect that it would outperform the simpler Random Forest Regression model. However, as previously stated, the model hit the feature ceiling, as there is too much unexplained variation within the data to fit more than 80% within a analytical model. In attempt to break through the ceiling, the next experiment will utilize a more complex neural network and test if it can perform better than ensemble models.

Model 3: Feedforward Neural Network

A Feedforward Neural Network (FNN) is a form of deep learning that utilizes a multi-layered network of neurons to interpret input data. It consists of an input layer, a number of hidden layers, and an output layer. Each neuron is interconnected between the surrounding layers and carries a weight and a bias. When an input enters a neuron, it is multiplied by the weight, and then the bias is added. Finally, an activation function is applied, in this case ReLU was used. This sets negative numbers equal to 0, dropping them from the equation. This numerical output is then passed from layer to layer, until a final value is expressed in the output layer. During training, the model employs backpropagation to iteratively adjust its weights and biases based on the error between predicted and true

values. The model therefore should learn the mathematical relationships between raw player stats and AAV values.

Python Code

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error
import matplotlib.pyplot as plt


# Load Data
csv_path = r"Skater_stats_2024_no_goalies.csv"
use_cols = ["AAV_num","Player.name","Length","G","A","+/-","PIM","EVG","PPG","SHG","GWG",
            "EV","PP","SH","SOG","SPCT","TSA","ATOI.min","FOW","FOL","FO%","BLK","HIT","TAKE","GIVE"]


df = pd.read_csv(csv_path, usecols=use_cols)


# Drop player name column
df = df.fillna(df.mean(numeric_only=True))
names=df["Player.name"]
df = df.drop(columns=["Player.name"])




# Sort Data


X = df.drop(columns=["AAV_num"])
y = df["AAV_num"]


# Convert to array


X = X.values
y = y.values.reshape(-1, 1)


# Split Data


X_train, X_test, y_train, y_test, names_train, names_test = train_test_split(
    X, y, names, test_size=0.2, random_state=42
)
scaler_X = StandardScaler()
scaler_y = StandardScaler()


X_train = scaler_X.fit_transform(X_train)
X_test = scaler_X.transform(X_test)
y_train = scaler_y.fit_transform(y_train)
y_test = scaler_y.transform(y_test)




X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)


# Create Initial Feedforward Loop
```

```
class SalaryNet(nn.Module):
    def __init__(self, input_dim):
        super(SalaryNet, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Dropout(0.50),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 1)
        )


    def forward(self, x):
        return self.layers(x)


# Train Model
input_dim = X_train.shape[1]
model = SalaryNet(input_dim)


criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 300


for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()


    if (epoch+1) % 25 == 0:
        model.eval()
        with torch.no_grad():
            test_pred = model(X_test)
            test_loss = criterion(test_pred, y_test).item()
        print(f"Epoch [{epoch+1}/{epochs}]  Train Loss: {loss.item():.4f}  Test Loss: {test_loss:.4f}")


# Evaluate Model Accuracy


model.eval()
with torch.no_grad():
    y_pred_scaled = model(X_test).numpy()
    y_pred = scaler_y.inverse_transform(y_pred_scaled)
    y_true = scaler_y.inverse_transform(y_test.numpy())


r2 = r2_score(y_true, y_pred)
mae = mean_absolute_error(y_true, y_pred)


print("\n☑ Evaluation Results:")
print(f"R² Score: {r2:.4f}")
print(f"MAE: {mae:.2f}")
print(f"Train R²: {r2_score(scaler_y.inverse_transform(y_train.numpy()), scaler_y.inverse_transform(model(X_train).detach().numpy())):.4f} |
Test R²: {r2_score(scaler_y.inverse_transform(y_test.numpy()), scaler_y.inverse_transform(model(X_test).detach().numpy())):.4f}")




# Plot results


plt.figure(figsize=(7,6))
plt.scatter(y_true, y_pred, s=30, alpha=0.7)
```

```
plt.xlabel("Actual AAV (tens of millions of dollars)")
plt.ylabel("Predicted AAV (tens of millions of dollars)")
plt.grid(False)
max_val = max(max(y_true), max(y_pred))
min_val = min(min(y_true), min(y_pred))
plt.plot([min_val, max_val], [min_val, max_val], color='red', linewidth=2, label='Perfect Prediction')


# Annotate player names
for i in range(len(y_true)):
    plt.text(y_true[i], y_pred[i], names_test.iloc[i], fontsize=7, alpha=0.8,
             ha='left', va='bottom')


plt.tight_layout()
plt.show()
```

Results

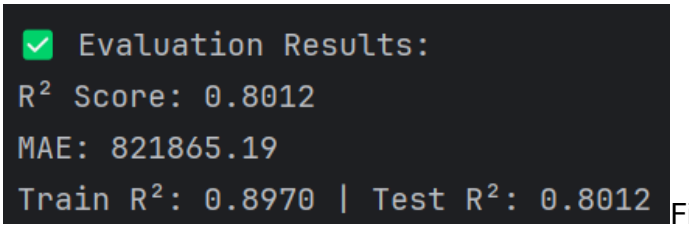

Figure 8. FNN Accuracy Metrics

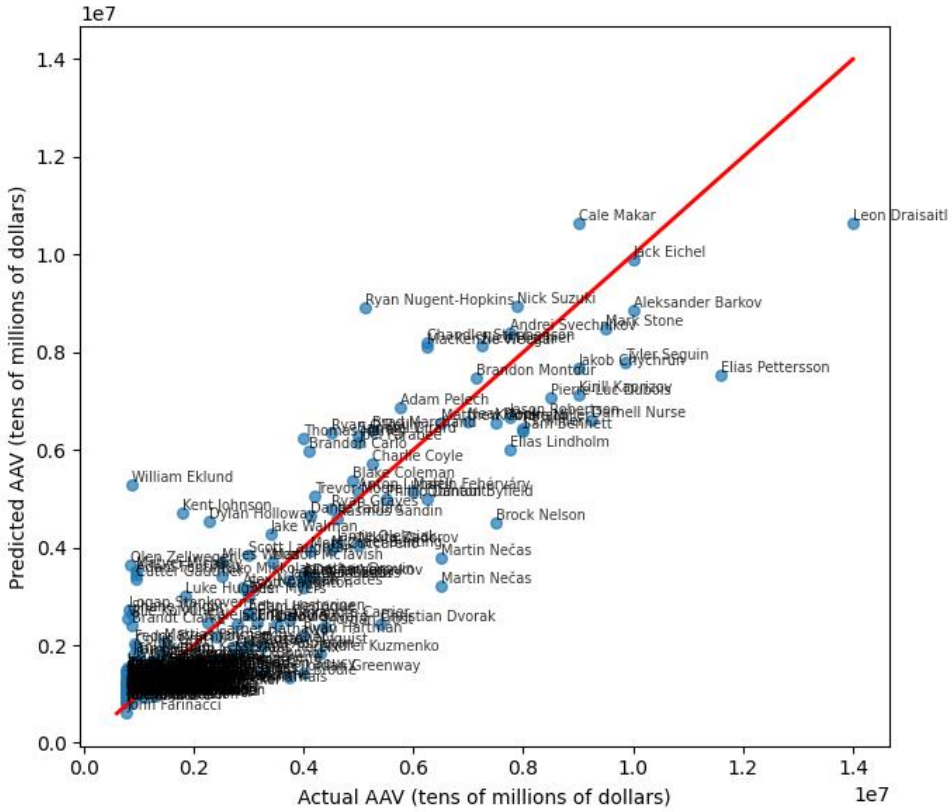

Figure 9. FNN Predicted AAV vs. Actual AAV

Discussion

Although the FNN is a more complex, more computationally demanding model compared to the previously used ensemble models, it achieved a very similar degree of accuracy in its' predictions with an R^2 score of 0.8012 (see figure 8). As stated before, this is due to the degree of inconsistency that NHL players produce compared to their salaries, forming a feature ceiling. Accuracy did not improve within the model despite attempts to tune the number of epochs, the size of the test set, and dropout rate of the neural network.

This model also experienced issues with overfitting, as the initial split between training set and test set R^2 scores was initially 0.97-0.78. The two main issues found were the high number of iterations the model went through in training as well as the low dropout rate. The dropout rate determines the rate of randomly ignored neurons during the forward pass in training. This prevents the model from relying too heavily on a specific pathway when predicting results. Additionally, the number of epochs was too high, causing the model to go through too many passes of the training data, and memorizing these patterns. By increasing the dropout rate and lowering the epochs, the final R^2 split was achieved of 0.88-0.80 (see figure 8).

Another issue encountered while creating this FNN model was the model's inability to deal with missing data. Unlike ensemble models, which can simply skip over missing data values, neural networks are unable to process data that includes missing values. Initially, the command "df = df.dropna()" was utilized, which drops any rows that include missing values. However, this rendered the dataset too small for the model to learn any patterns from, which lowered accuracy and caused overfitting. This line was then replaced with the line"df = df.fillna(df.mean(numeric_only=True))", which fills any missing value in the dataset with the mean value from other rows. This allows the neural network to fully process the data, while not skewing the numbers too much while filling missing values.

Key Findings

The models built in this study saw a consistently high degree of success. The Random Forest, XGBoost, and Feedforward Neural Network models each achieved an R^2 value of roughly 0.80, showing that their ability to interpret statistical performance alone explains about 80% of contract value variance. However, every model hit a data ceiling and was unable to improve accuracy further than ~0.80. This demonstrates the unpredictable nature of sports performance, as while raw statistics certainly do tell a story, there are many variables that are impossible to quantify that contribute to a player's success. Traits such as leadership, communication, confidence, and minor injuries all play a major role in

determining the impact of a player on their team. When evaluating players, it is important to remember that raw statistics will only tell part of the story, say ~80%. Therefore, it is important to take into account qualitative and personal traits as well, something that machine learning will always struggle with. While these models are likely better at interpreting data than advanced NHL scouts, there will always be a demand for human input when dealing with player valuations.

Another interesting point is that each model was able to accurately predict the AAVs of low to solid tier NHL players but always labeled the top players as overpaid. Statistically, these top players such as Connor McDavid, Mikko Rantanen, and Leon Draisaitl do not produce enough to justify their high salaries. To retain these top players, teams must overpay them relative to their production. This theory explains why the Florida Panthers were able to triumph over the Edmonton Oilers over the past two years. While Edmonton's star players are statistically very solid, they take up too much of the salary cap to justify the production of two key guys. Whereas the Florida Panthers have a roster composed of players who are not considered quite as elite as McDavid and Draisaitl, their production – salary ratio allows the Panthers to afford a better roster top to bottom, while keeping high production. The success of the Panthers over the Oilers can be seen in these models, which may be useful for NHL teams when constructing rosters.

Finally, the question still remains: How much is Kirill Kaprizov really worth? By adding a line into the FNN model to print its predicted value for the AAV of Kaprizov, it estimated that he is worth $7,453,372.50 a year based on his 2024 production. He did struggle with injuries this past season, which likely impacted his stats and lowered his value in the eyes of our model. Nonetheless, his numbers say that he is by far now the most overvalued player in the NHL, and absolutely not worth a $17 million-dollar AAV.

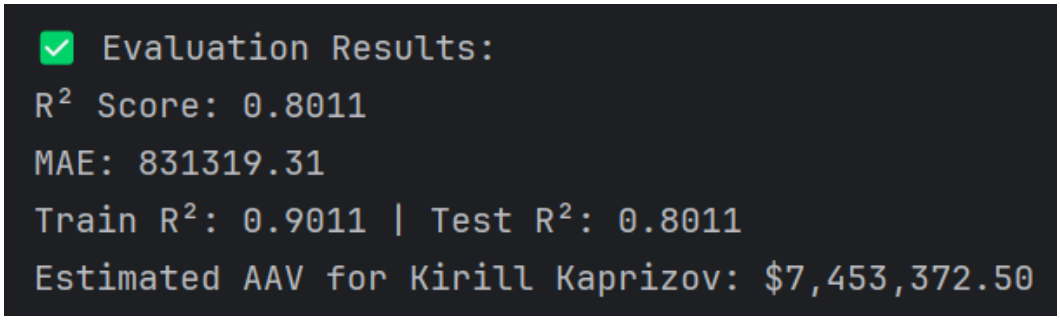

Figure 10. FNN Prediction for Kirill Kaprizov

Model	Train R^2	Test R^2	MAE ($)
Random Forest	0.88	0.80	0.826M
XGBoost	0.88	0.80	0.82M
FNN	0.88	0.80	0.83M

Table 1. Comparative R^2 and Mean Average Error scores for each model

Conclusion

Through this project, I gained valuable experience in machine learning engineering. I not only learned how to build and implement three prominent models; Random Forest Regression, XGBoost, and Feedforward Neural Networks, but also deepened my understanding of core machine learning concepts. Throughout the process, I encountered and overcame challenges such as model overfitting, missing data, and data misalignment. By experimenting with feature engineering, hyperparameter tuning, and data preprocessing, I developed a stronger intuition for how different design choices influence model performance and generalization. Overall, this project strengthened both my technical and analytical skills, and it gave me a deeper appreciation for how data-driven modeling can be applied to real-world problems such as player valuation in professional hockey. I am excited to jump into more complex and impactful projects in my journey to use machine learning capabilities to improve the human condition.

If I were to take this project further, I would look into adding more data in order to boost the accuracy of these models. By creating a bigger dataset and including past years of NHL stats, I believe there is a solid probability that these algorithms would show higher accuracy by allowing them a better chance to recognize patterns. However, the NHL salary cap has been increasing overtime, which skews the data. To combat this, I would change the target variable from AAV to each player's salary percentage of their teams salary cap. This would level the field between years so that all players' monetary values are in relation to the salary cap of that year.

Thank you for taking the time to read this report, I am open to any suggestions, questions, and critiques that you may have, so please reach out to me.

References

(1) “Kirill Kaprizov.” Elite Prospects, www.eliteprospects.com/player/265645/kirill-kaprizov

. Accessed 5 Nov. 2025.