Analyzing Game Data from the National Hockey League, 2010-2020

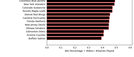
Ryan Glasser & Evan Haeck

Part 1

Data Acquisition, Cleaning, Processing, and Analysis

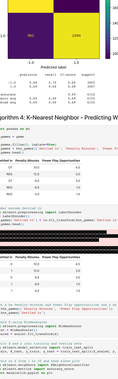
Data Sources

```
# [redacted] SQLITE DB .dbf - primary process
# [redacted] DATA .csv - primary process
[redacted] STATS.csv # Game statistics for this
[redacted] STATS.csv # Game statistics for this
[redacted] STATS.csv # Game statistics for this
```


Pittsburgh Penguins
Tampa Bay Lightning
Boston Bruins
New York Rangers
Chicago Blackhawks
Detroit Red Wings
Nashville Predators
St. Louis Blues
Anaheim Ducks
Vancouver Canucks
San Jose Sharks
Los Angeles Kings
Dallas Stars
Minnesota Wild
Calgary Flames
Colorado Avalanche
Edmonton Oilers
Winnipeg Jets
Carolina Hurricanes
Philadelphia Flyers
New Jersey Devils
Buffalo Sabres
Columbus Blue Jackets
Ottawa Senators
Florida Panthers
New York Islanders
Washington Capitals
Toronto Maple Leafs
Arizona Coyotes
Vegas Golden Knights
Winnipeg Jets
Pittsburgh Penguins
Boston Bruins
Tampa Bay Lightning
Nashville Predators
Chicago Blackhawks
Detroit Red Wings
Anaheim Ducks
St. Louis Blues
Vancouver Canucks
San Jose Sharks
Los Angeles Kings
Dallas Stars
Minnesota Wild
Calgary Flames
Colorado Avalanche
Edmonton Oilers

```
y_grid = sns.pandplot(y_test)
```

```
# evaluate model
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
confusion_matrix = mm_clsf.score(X_test, y_test)
print(confusion_matrix)
print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

         0.0       0.68      0.70      0.69       924
         1.0       0.69      0.68      0.69      2987

    accuracy                           0.69      3911
   macro avg       0.69      0.69      0.69      3911
weighted avg       0.69      0.69      0.69      3911
```

Algorithm 4: K-Nearest Neighbor - Predicting When a Game is Settled (Regulation/Overtime)

```
import pandas as pd

ncs_games = game

ncs_games=filtered[['Home','Away','Date','Bite','Penalty Minutes','Power Play Opportunities']]
ncs_games = ncs_games[('Home' != 'Vancouver')]
ncs_games = ncs_games[('Home' != 'Seattle')]
```

```
Settled =  Penalty Minutes  Power Play Opportunities
0                     NaN                          NaN
1                     NaN                          NaN
2                     NaN                          NaN
3                     NaN                          NaN
4                     NaN                          NaN
...                   ...                          ...
5                     NaN                          NaN
6                     NaN                          NaN
7                     NaN                          NaN
8                     NaN                          NaN
9                     NaN                          NaN

[5 rows x 2 columns]
```

```
# Set 0 for Penalty Minutes and Power Play Opportunities and 2 for Settled in
X = ncs_games[['Penalty Minutes', 'Power Play Opportunities']]
y = ncs_games['Settled']]

# remove 2 using dicomnotnull
from sklearn.preprocessing import SimpleImputer
imputer = SimpleImputer()
X = imputer.fit_transform(X)

# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, train_size=0.8)

# train KNN on 2 folds in 10 and show elbow plot
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_score
import matplotlib.pyplot as plt

k_values = range(1, 21)
cv_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=10)
    cv_scores.append(scores.mean())
print("Cross-validated accuracy scores:", cv_scores, y_pred)

# print the 5 most accurate k
accuracies = zip(k_values, cv_scores)
print(sorted(accuracies, key=lambda x: x[1], reverse=True))

pts_values, accuracies,
plt.figure(figsize=(6,4))
plt.plot('Accuracy')
plt.xlabel('Accuracy')
plt.show()

(1,  0.75624
(2,  0.74375
(3,  0.73625
(4,  0.73125
(5,  0.72625
dtype: float64
```

Algorithm 5: Neural Network - Predicting Game Outcome

```
import torch
import pandas as pd

ncs_games = game

ncs_games=filtered[['Home','Away','Date','Bite','Penalty Minutes']]
ncs_games = ncs_games[('Home' != 'Vancouver')]
ncs_games = ncs_games[('Home' != 'Seattle')]
X = ncs_games[['Home', 'Away', 'Bite', 'Penalty Minutes']].values, ncs_games['Home'].values
from sklearn.preprocessing import StandardScaler, LabelEncoder, TensorDataset, DataLoader, random_split, ConcatDataset
print(X.shape, y.shape)

torch.manual_seed(42)
torch.cuda.manual_seed(42)
```

```
(3911, 4) (3911,)
```

```
device = [
    'cuda',
    if torch.cuda.is_available:
    'mps',
    if torch.backends.mps.is_available:
    else 'cpu'

print("Using cpu device")
```

```
Using cpu device
```

```
import torch.nn as nn

class NeuralNetwork(Module):
    def __init__(self):
        super(NeuralNetwork, self).
        self.hidden1 = nn.Linear(4, 8)
        self.hidden2 = nn.Linear(8, 9)
        self.output = nn.Linear(9, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # ReLU activation for hidden layers
        x = self.sigmoid(self.hidden1(x))
        x = self.sigmoid(self.hidden2(x))
        x = self.output(x)
        return x

model = NeuralNetwork()
print(model)
```

```
NeuralNetwork(
  (hidden1): Linear(in_features=4, out_features=8, bias=True)
  (hidden2): Linear(in_features=8, out_features=9, bias=True)
  (output): Linear(in_features=9, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```

```
import torch
import torch.nn as nn
import torch.optimizer as optim
import tqdm

# define train_model, X_train, y_train, X_val, y_val:
# Hyperparameters and optimizer
learning_rate = 0.001(Adam)  # typical learning rate for Adam
batch_size = 64(Adam)  # typical batch size
epochs = 100  # number of epochs to train
# Initialize model, loss function, and optimizer
model = YourModelClass()
loss_fn = nn.BCELoss()  # binary cross-entropy
optimizer = optim.Adam(model.parameters(), lr=0.0001)

n_epochs = 100  # number of epochs to run
batch_size = 10  # size of each batch
batch_start = torch.arange(0, len(X_train), batch_size)

# Hold the best model
best_acc = - np.inf  # init to negative infinity
best_weights = None

for epoch in range(n_epochs):
    model.train()
    with tqdm.tqdm(batch_start, unit="batch", mininterval=0, disable=True) as bar:
        bar.set_description(f"Epoch {epoch}")
        for start in bar:
            # take a batch
            X_batch = X_train[shuffled_indices[start:start+batch_size]
            y_batch = y_train[shuffled_indices[start:start+batch_size]
            # forward pass
            y_pred = model(X_batch)
            loss = loss_fn(y_pred, y_batch)
            # backward pass
            optimizer.zero_grad()
            loss.backward()
            # update weights
            optimizer.step()
            # print progress
            acc = (y_pred.round() == y_batch).float().mean()
            bar.set_postfix(
                mse=float(loss),
                acc=float(acc)
            )
    # evaluate accuracy at end of each epoch
    model.eval()
    y_pred = model(X_test)
    acc = (y_pred.round() == y_test).float().mean()
    acc = float(acc)
    if acc > best_acc:
        best_acc = acc
        best_weights = copy.deepcopy(model.state_dict())
# restore model and return best accuracy
model.load_state_dict(best_weights)
print("MSE: %.2f, ACC: %.2f" % (mse, acc))
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

# define 5-fold cross validation test harness
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
cvscores = []
for train, test in kfold.split(X, Y):
# create model, train, and get accuracy
model = KerasClassifier(build_fn=baseline_model, epochs=150, batch_size=5, verbose=0)
print('Accuracy: %f%%', % (*100, *))
print('Accuracy: %f%%', % (*100, *))
```

```
Accuracy: 0.7670
Accuracy: 0.7670
Accuracy: 0.7670
Accuracy: 0.7670
Accuracy: 0.7670
Accuracy: 0.7670
```

```
# Display the confusion matrix
!pip install -U git+https://github.com/keras-team/keras-preprocessing.git+tensorflow
import numpy as np, matplotlib.pyplot as plt, seaborn
from sklearn import metrics, mlxtend

# use the activation datasets to visualize
model = Sequential()
model.add(Dense(12, input_shape=(8,)))
# model.add(Dense(8))
# model.add(Dense(1))
# model.compile(
model.fit(X_train, y_train, epochs=150, batch_size=10)
# evaluate the model
scores = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

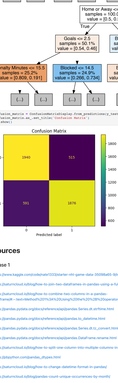
```
model_save_path = "./model.h5"
model.save(model_save_path)
!mkdir -p saved_model
!cp -r ./model.h5 saved_model
!zip -r /content/keras_model.zip -r /content/saved_model/
!ls -lh /content/keras_model.zip
from google.colab.files import *
files.download('/content/keras_model.zip')
```

```
  adding: content/saved_model/ (stored 0%)
  adding: content/saved_model/variables/ (stored 0%)
  adding: content/saved_model/variables/variables.data-00000-of-00001 (deflated 7%)
  adding: content/saved_model/variables/variables.index (deflated 64%)
  adding: content/saved_model/saved_model.pb (deflated 91%)
  adding: content/saved_model/assets/ (stored 0%)
-rw-r--r-- 1 root root 1.3M Jun 13 20:44 /content/keras_model.zip
```

```
# Import the model and create a confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

model = KerasClassifier(model=model)
# model.fit(X_train, y_train, verbose=0)
# save X_test to .npy
np.save('X_test.npy', X_test)
# save y_test to .npy
y_pred = y_pred.round().ravel(); dataset[:,nonzero]
# y_pred = y_pred.ravel(); dataset[:,nonzero]
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Loss', 'Win'])
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:151:FutureWarning:
```



Algorithm 6: Random Forest - Predicting Game Outcome

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix, precision_score, recall_score, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint

from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import graphviz

ctypes = game

# 1. Split data into train and test
X = ncs_games[['Home', 'Away', 'Date', 'Bite', 'Penalty Minutes', 'Power Play Opportunities', 'Blocked']]
y = ncs_games['Home']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, train_size = 0.8, random_state = 1)

# 2. Define the model
param_dist = {'max_depth': randint(1,20),
              'min_samples_leaf': randint(1,20),
              'criterion': ['gini', 'entropy']}
model = RandomForestClassifier()
tuned = RandomizedSearchCV(model, param_distributions = param_dist, n_iter = 5, cv = 5)
# 3. Fit the model
print(tuned.fit(X_train, y_train))

{'max_depth': 11, 'n_estimators': 77}

model = RandomForestClassifier(max_depth = tuned.best_params_['max_depth'], n_estimators = tuned.best_params_['n_estimators'])
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=11, n_estimators=77)
```

```
# 4. print a tuned parameters and score
print(model.score(X_test, y_pred))

0.7712765957446808

for i in range(5):
    tree = model.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=X_train.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
    display(graph)
```

```
# 5. Confusion Matrix
confusion_matrix = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
plt.show()
```

Sources

Phase 1

https://www.kaggle.com/code/ratul123/jaycher-nhl-game-data-2005bb85-5b1acd

https://stackoverflow.com/questions/19-join-two-dataframes-in-pandas-using-a-full-outer-join#:~:text=Pandas%20is%20a%20popular%20data,df%20outer%20a%20data.

https://stackoverflow.com/questions/ny-combine-two-columns-in-a-pandas-dataframe#:~:text=Method%201%:%20Using%20%20%20

Analyzing Game Data from the National Hockey League, 2010-2020

Ryan Glasser & Evan Haeick

Part 1: Data Acquisition, Cleaning, Processing, and Analysis

Problem Statement: Professional sports data can be messy in general, especially as it relates to the game of hockey, and data analytics is becoming a new approach to the strategy that hockey teams take to build a championship caliber team. There are tens of statistics that are documented over the course of any given game, and with each team playing at least 82 games every season, there are plenty of statistics to go around. For this project, we wish to understand how we can take a messy data set of NHL statistics, and make sense of it in order to understand what contributes most to a winning hockey team, and how different hockey outcomes correlate with both on-ice, and off-ice statistics. Some of the questions we wish to answer are:

1. Which teams statistically have performed best in a hockey sense, and which are expected to continue to perform well?
2. Which game statistics, in addition to the number of goals scored, contribute most to winning hockey teams (i.e., shots on goal, hits, blocked shots, etc.)?
3. Are there any features of a hockey club off-ice that can explain team performance? How does having a veteran head coach influence team performance?
4. How does team performance fluctuate throughout the year? Is there a pattern as to when some teams perform better than others?

The world of sports is a competitive one. Many people build a brand around their team to sell and promote merchandise to sell to consumers. Being able to build a championship team attracts fans and generates revenue in general. As new regimes of team staff come and go, new management attempts to analyze in-game statistics to see what works over time and what they themselves can implement for their respective teams.

The NHL and other entities gather data for each game; who was playing, what the score was, how many times the puck was taken from the other team, how long one team has held the puck for, how many attempted shots, wins and losses per team for each season, and much more. This historical data can be used to see how a team may perform in the future based on the player roster and coaches, and how each team performed in certain matchups.

Our project will go over data that can be used to predict the future of game outcomes. The output can be used to predict future team performance and tell which teams will win and lose based on raw and refined statistics. This analysis will be an addition to the world of sports prediction and will provide insight into how a team may perform in the future, and whether team management may need to consider changing up the gears of their respective teams, in an effort to build a tougher and more winning hockey club.

The data sets analyzed for this project come from Kaggle, a platform for data scientists to share and analyze code that is available for viewing and downloading. The shape of the main data frame to be cleaned, preprocessed, and analyzed is 62,890 rows by 18 columns. Two other data sets also housed in the same repository will aid in the preprocessing of our main dataset.

Database Link: https://www.kaggle.com/code/nate1333/starter-nhl-game-data-35098a65-9/input

In this project, we conducted various data cleaning, merging, and processing steps to make our data as usable as possible for us to code. We have done the following processes:

1. We merge the dataset from game_team_stats.csv into game.csv on the game_id column.
2. We add the “shortName” (city name) and “teamName” columns to the list in order to have more comprehensive data on which team the data in the row represents, by merging team_info.csv into game.csv on the team_id column.
3. We drop the redundant row 'game_id' since the data that it holds already exists in other forms in different columns.
4. We combined the “shortName” and the “teamName” into one column to give a full comprehensive name representing each team for each row.
5. We check for duplicate rows and remove them to make sure there is no overrepresentation of a single data point.
6. We remove all empty or N/A cells and remove the rows that have incomplete data. This ensures that this corrupted data does not affect the final analysis.
7. We change the date and time format to a more readable and usable format. In this case, we simply remove the time and change the date formatting. The column type becomes type “datetime”.
8. We change the face off win percentage to a value between 0 and 1 in order so that it is no longer categorical and can be used in machine learning algorithms later.
9. We converted the 'won' column into binary data for an easier way to use the data later in algorithms.
10. Create a column named "Power Play Success Rate", as "powerPlayGoals" divided by "powerPlayOpportunities" to display an important hockey statistic.
11. Standardize column header names with capital letters and spacing.
12. Fix errors in the "Team Name" column, by either fixing the name of the team itself, or combining two teams that have changed names over time but still belong to the same franchise.

In addition to cleaning and formatting the data, we gathered some basic statistics and built charts around the data to determine what the data might be able to tell us. Ten distinct exploratory data analysis operations are carried out as follows:

1. Descriptive statistics for each feature

Outcome: Shows that the average number of shots per game is 30.462492 and that there was a game with a maximum of 10 goals scored.

Use: Can be used in determining the main statistics of each of the columns in the dataset to pinpoint features to be utilized in modeling.

Figure:

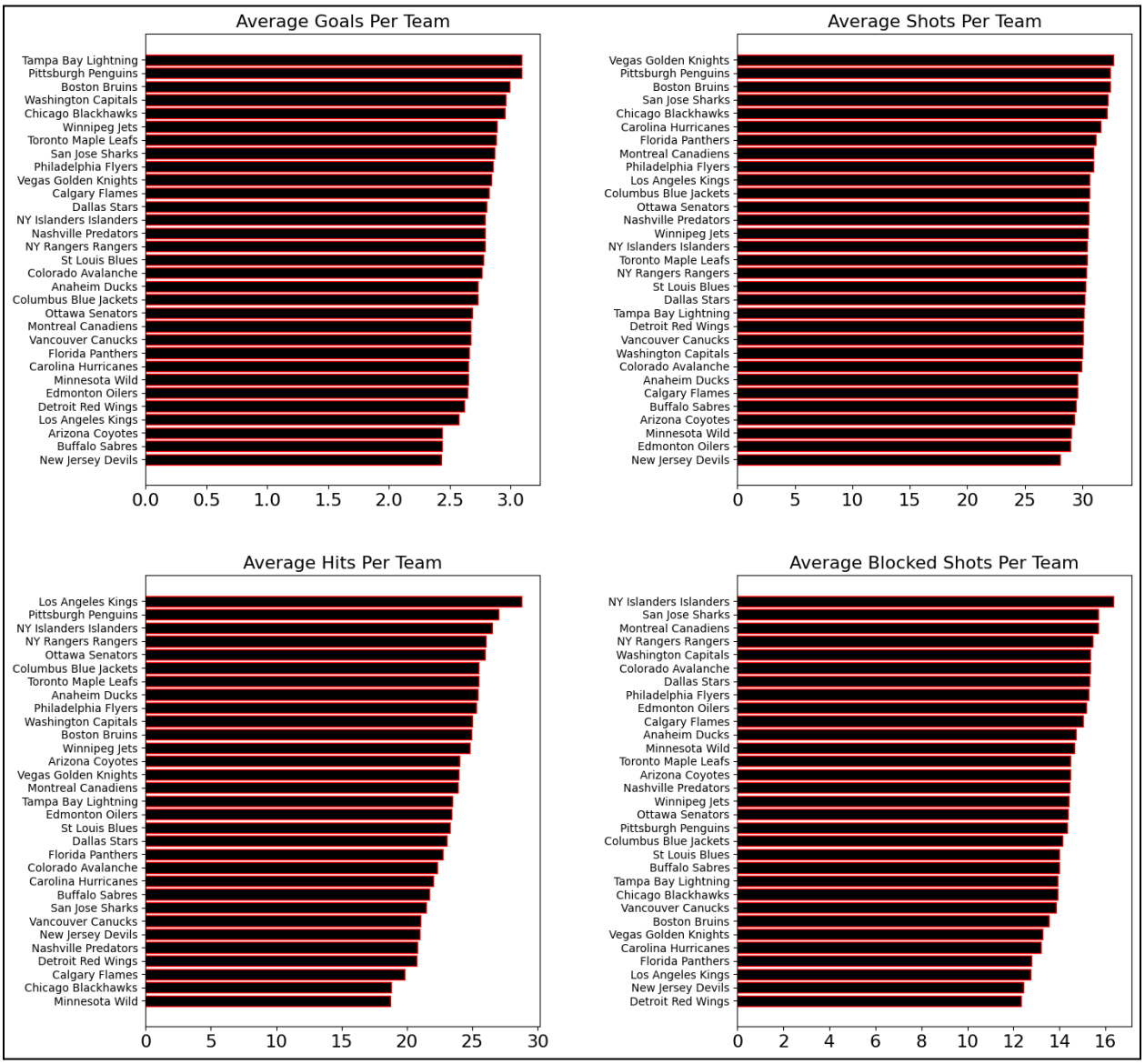
	Team ID	Won	Goals	Shots	Hits	Penalty Minutes	Power Play Opportunities	Power Play Goals	Face Off Win Percentage	Giveaways	Takeaways	Blocked	Power Play Success Rate
count	24608.000000	24608.00000	24608.000000	24608.000000	24608.000000	24608.000000	24608.000000	24608.000000	24608.000000	24608.000000	24608.000000	24608.000000	24128.000000
mean	17.299049	0.50000	2.762882	30.462492	23.458672	10.050715	3.165597	0.592693	0.500000	8.888898	6.900723	14.377967	0.186876
std	11.935721	0.50001	1.644385	6.823088	8.624283	7.948202	1.552854	0.759828	0.072159	4.912037	3.756261	5.095270	0.248846
min	1.000000	0.00000	0.000000	10.000000	2.000000	0.000000	0.000000	0.000000	0.208000	0.000000	0.000000	0.000000	0.000000
25%	8.000000	0.00000	2.000000	26.000000	17.000000	6.000000	2.000000	0.000000	0.452000	5.000000	4.000000	11.000000	0.000000
50%	16.000000	0.50000	3.000000	30.000000	23.000000	8.000000	3.000000	0.000000	0.500000	8.000000	6.000000	14.000000	0.000000
75%	24.000000	1.00000	4.000000	35.000000	28.000000	12.000000	4.000000	1.000000	0.548000	12.000000	9.000000	18.000000	0.333333
max	54.000000	1.00000	10.000000	88.000000	75.000000	183.000000	12.000000	6.000000	0.792000	36.000000	31.000000	62.000000	1.000000

2. Bar plots for average goals, shots, hits, and blocks per game for each team

Outcome: The Lightning had the most average goals over all games, while the Golden Knights had the most average shots over all games. The Islanders had the most blocked shots over all the games.

Use: Can be used to show a team's raw overall performance and potentially be used in a variety of modeling algorithms, such as Linear Regression.

Figure:

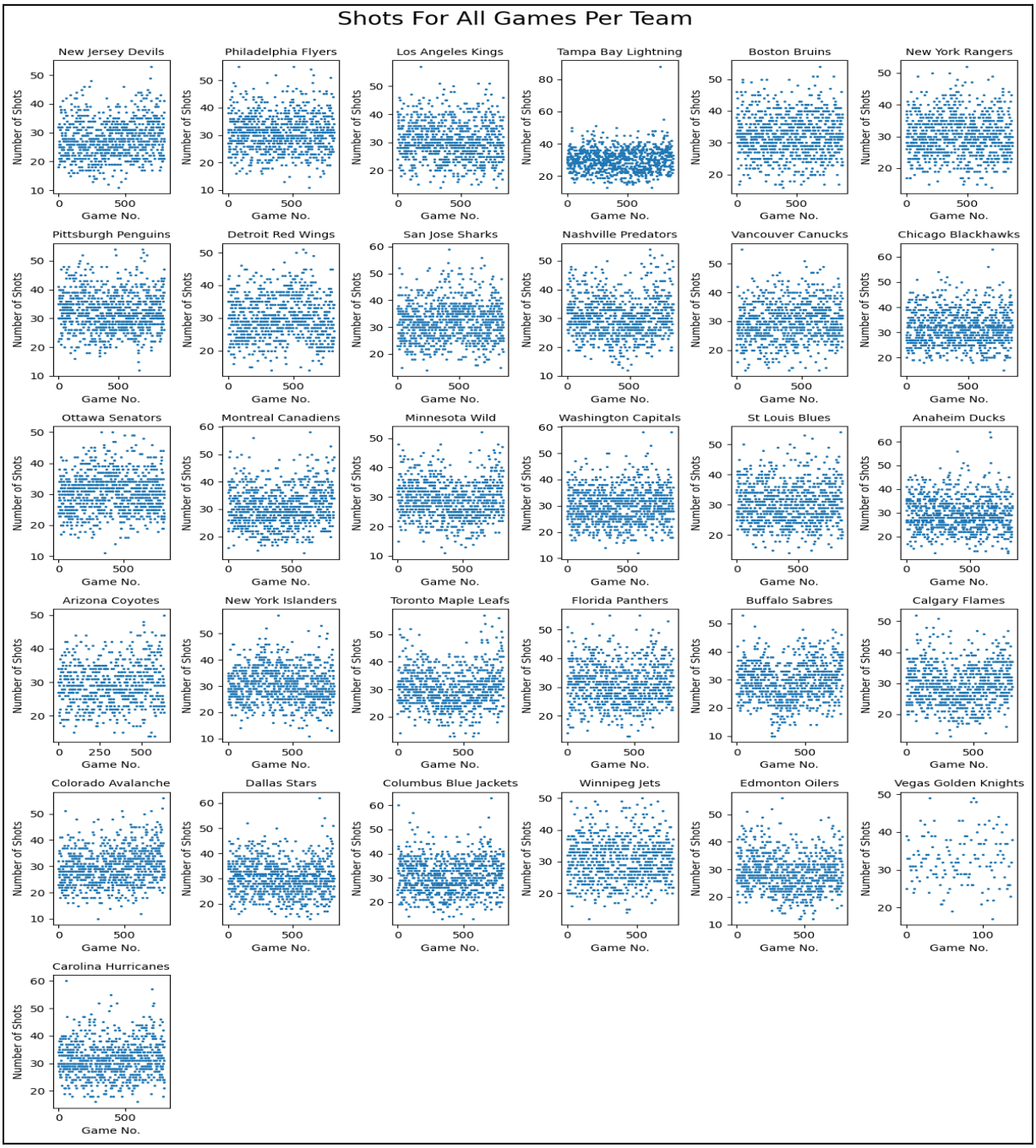


3. Scatterplots of shots over all games per team

Outcome: The Golden Knights have a relatively large variance while the Lightning have a small variance.

Use: Can be used to determine how aggressive a team plays, and in addition, can be used to check how many goals a team might be able to get when compared to the other team's ability to save. This is also an offensive statistic, which may be an appropriate feature to consider when analyzing successful hockey clubs.

Figure:

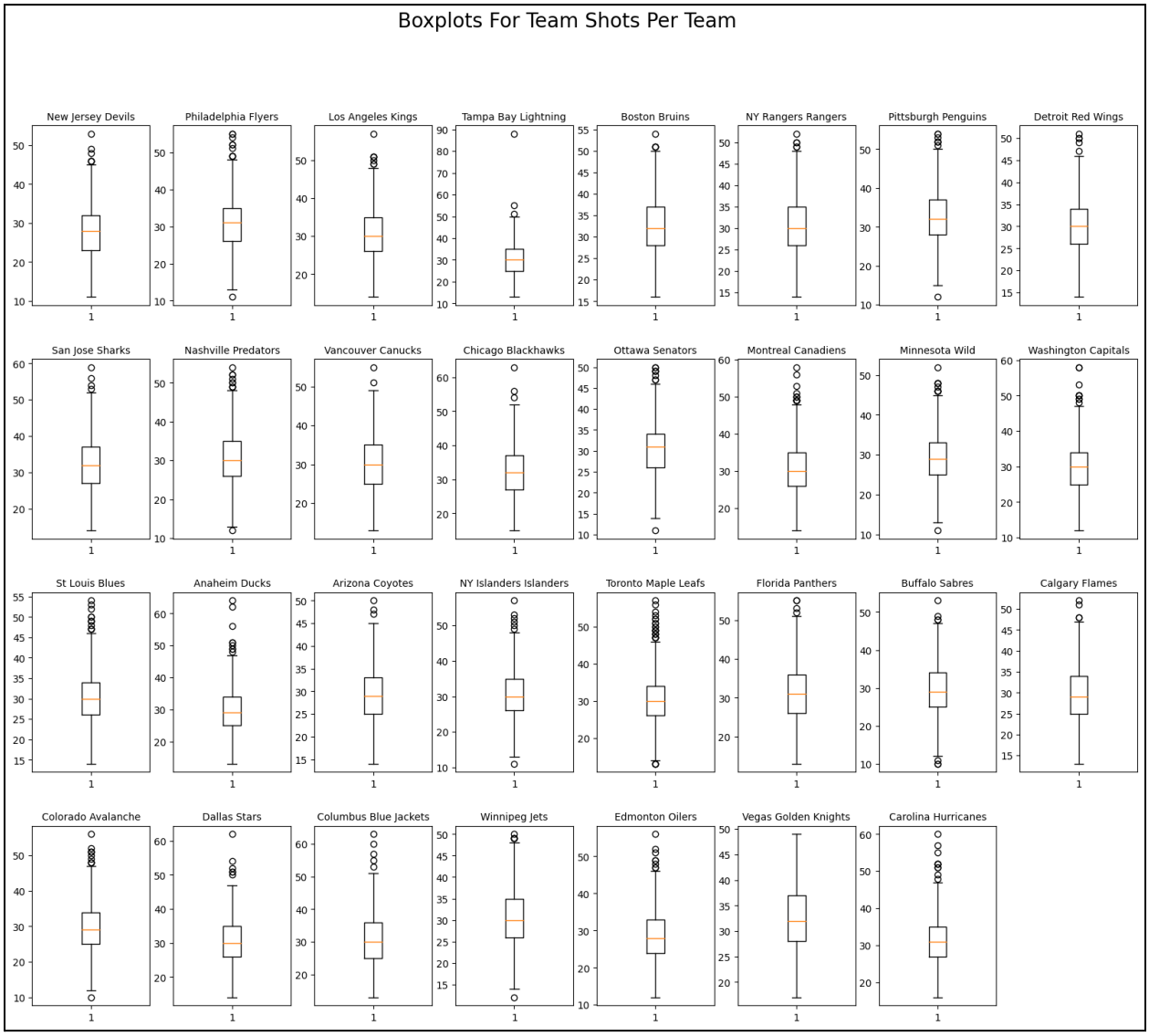


4. Boxplot of mean, interquartile range and outliers of shots for each team

Outcome: Shows that the Lightning have lower average shots compared to the other teams and the Carolina Hurricanes have multiple above average outlier game shots.

Use: Can be used to compare shots to goals to give a result of the accuracy of teams and how often teams capitalize on their opportunities. Does this prove good offensive capabilities, or is it that the opponents goaltender is below average? Team shots have major implications for offensive abilities and are useful in predictive analytics.

Figure:

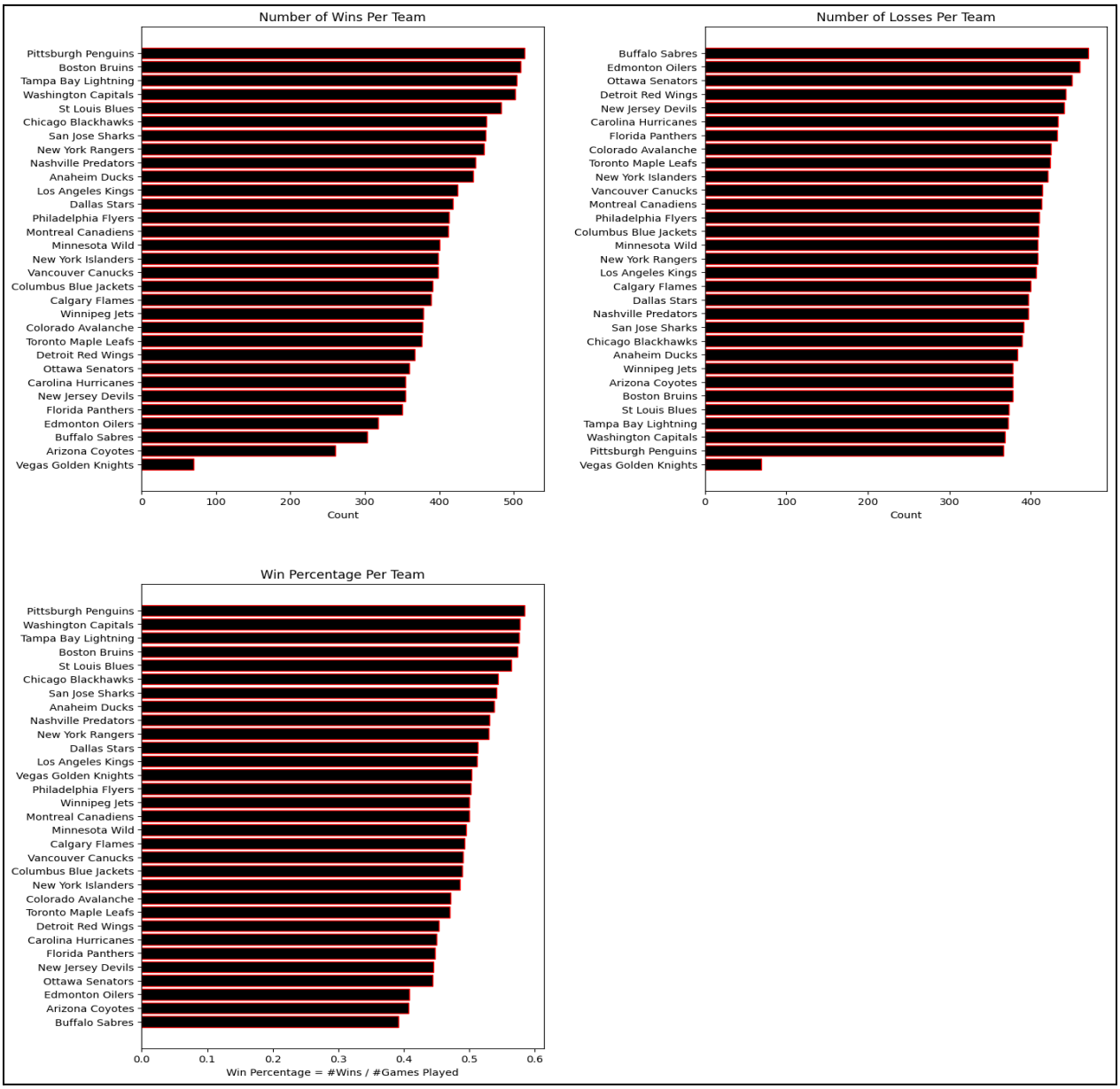


5. Win percentage for each team

Outcome: Shows the Pittsburgh Penguins have the most wins and the Buffalo Sabres have the least wins.

Use: Can be used to predict the outcome of a game(s) in the future given the amount of wins for teams in previous games. This statistic may also be helpful in projecting whether a team is going to make the playoffs or not in the future, whether new team management should be considered (i.e., a new head coach), and the number of future wins in subsequent seasons.

Figure:

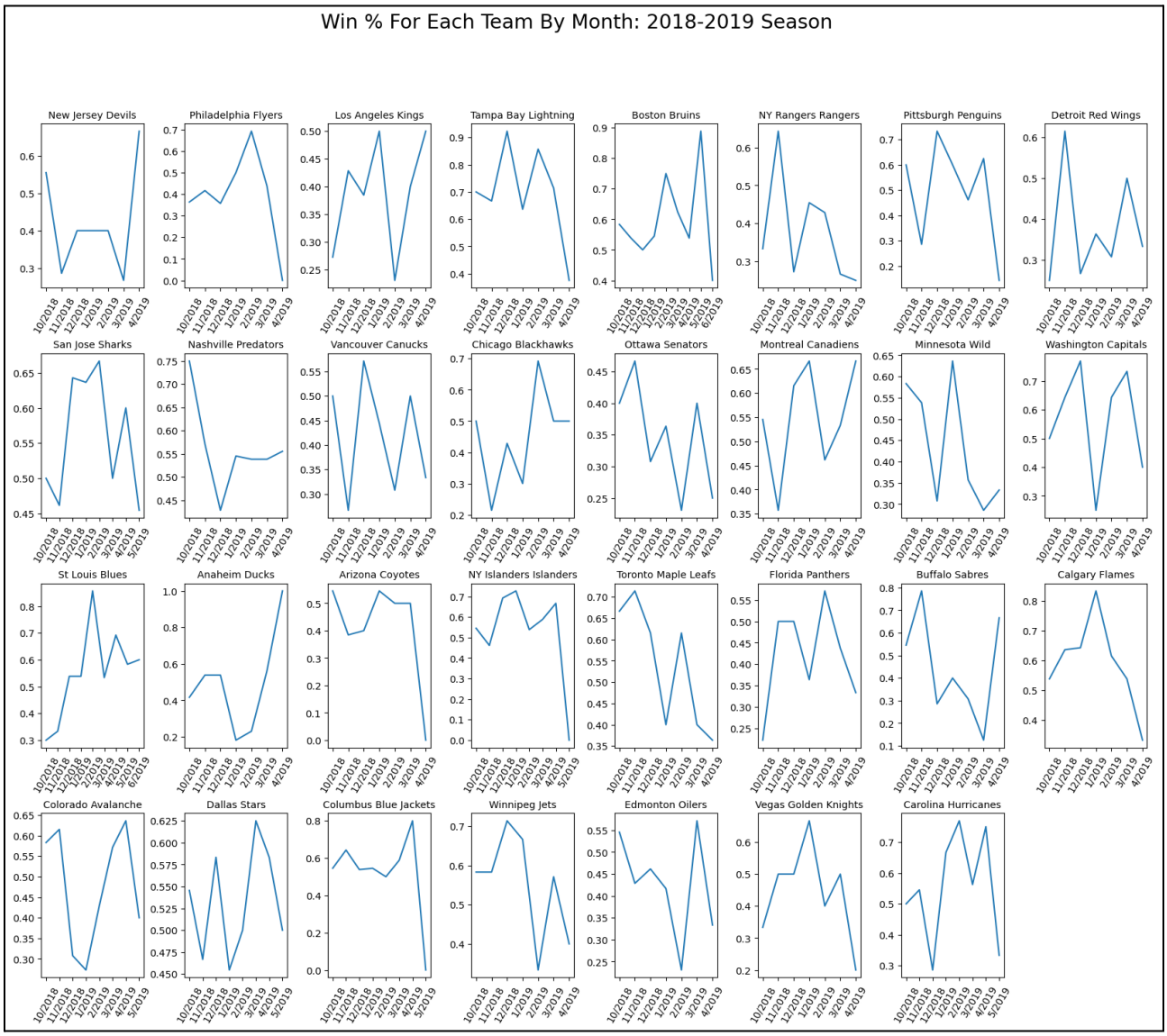


6. Wins per month for each team in the 2018-2019 season (or any season by replacing the years)

Outcome: Shows that many teams have a dip in their win percentage towards the end of the season and that only the Anaheim Ducks have a perfect win percentage for a particular month of the season.

Use: Can be used to predict which months of a future season a team might win based on their performance in previous seasons. This also gives insight into whether a team that has already locked a playoff spot takes their foot off the gas to conserve energy for the postseason.

Figure:

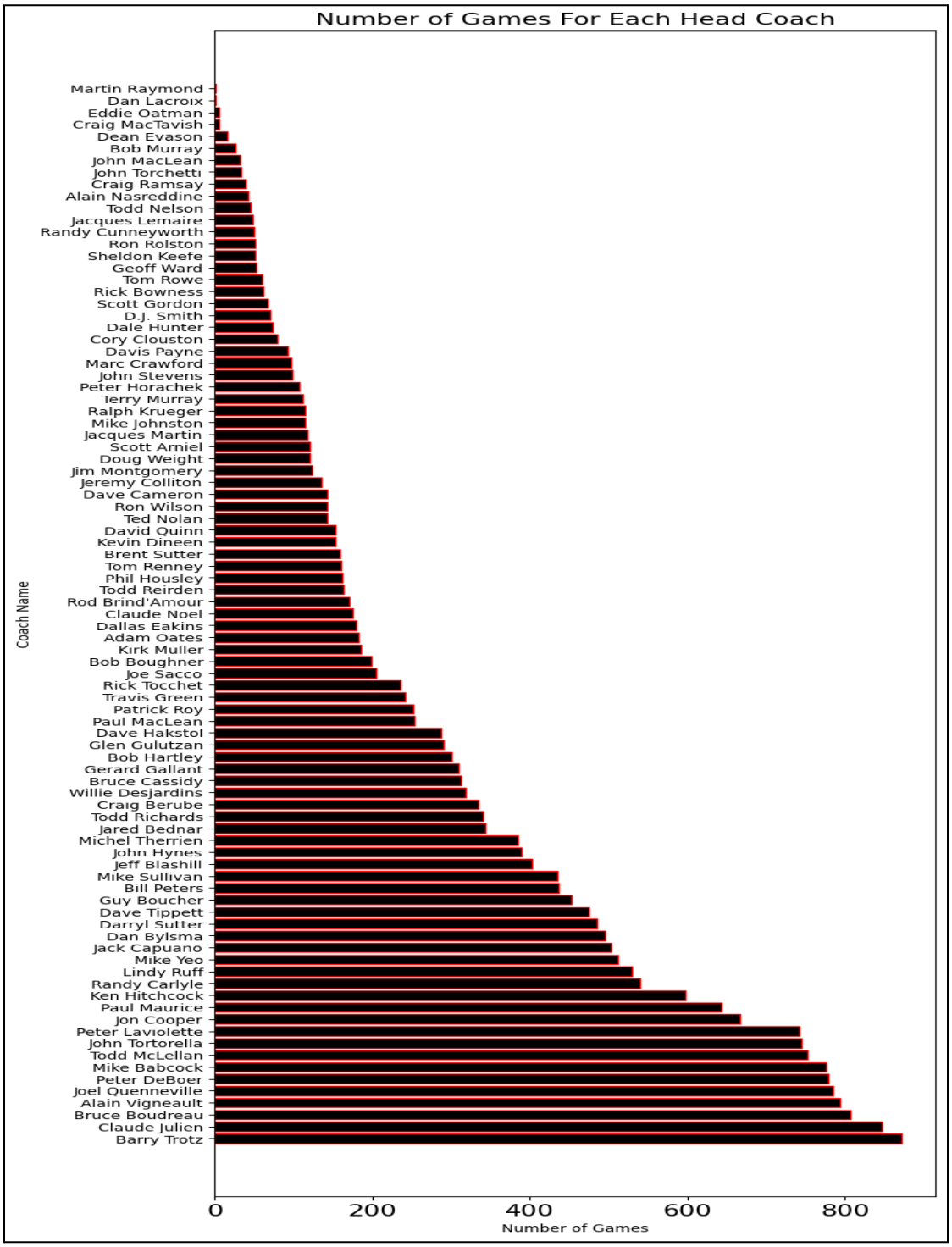


7. Number of games per head coach

Outcome: Martin Raymond was a coach for the least amount of games, and Barry Trotz was a coach for the most.

Use: Can be used to determine the effectiveness of a specific coach on a team by determining the amount of games won under each coach. This can be an important feature in predicting win percentage, number of goals scored, and shots on goal.

Figure:

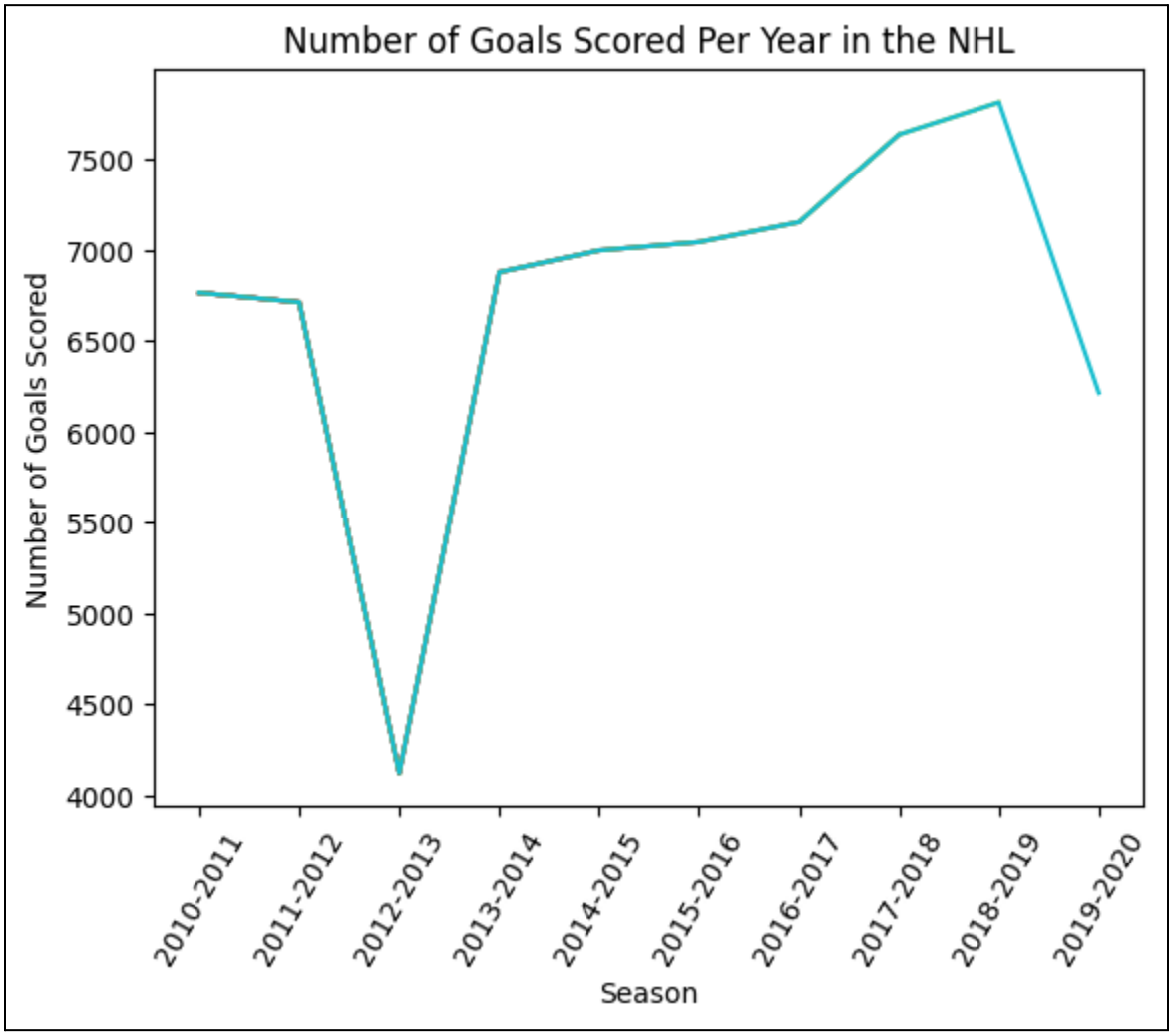


8. Number of goals per season

Outcome: The most goals scored was during the 2018-2019 season and the least amount was during the 2012-2013 season.

Use: Can be used to analyze whether a dip or rise in overall league goals can be attributed to specific team performances, or league-wide change in team performance across all teams. This may allow us to see how the sport has changed over time, and whether some seasons may not be best to use in building models to make predictions.

Figure:



9. Pie charts of “Settled In” compared with “Won” over all games per team

Outcome: All teams had similar ratios of overtime and regulation games, as well as win/lose ratios.

Use: Can be used to determine the probability that a certain game outcome will be settled in overtime or not based on data from all games. If a team is consistently losing in overtime, this could be an indicator that a team needs better players that know how to finish games, or needs to change management to people who have better strategy at how to protect leads in games and/or play better in overtime in general.

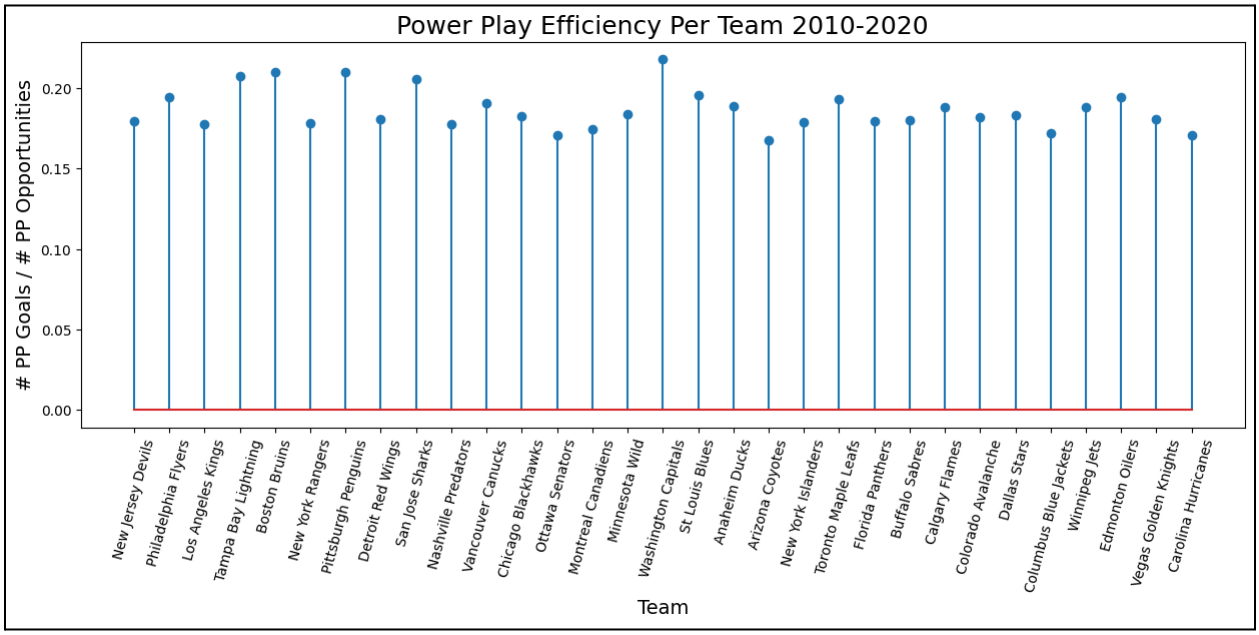
Figure:

10. Power play efficiency for each team, calculated as the number of power play goals divided by the number of power play opportunities.

Outcome: The Washington Capitals have the highest value for Power Play Efficiency and the Arizona Coyotes have the least Power Play Efficiency.

Use: Can be used to determine the outcome of a game for teams. Typically, those with high power play efficiency are championship caliber teams. In addition to predicting the outcome of future games, this feature could be useful in projecting the number of goals and shots for a team in general.

Figure:



In addition to mentioning the uses that each of the EDA steps would provide in performing predictive modeling and analysis, it is important to note that these steps also allow for the pinpointing of which features would be useful in machine learning algorithms, such as linear regression, logistic regression, support vector machines, naive bayes, neural networks, or decision trees. These EDA steps give insight into which features could or should be considered in predicting continuous and binary output variables, such as number of goals scored and whether a team won a game or not, respectively, and which should not be used and would do more harm than good with regards to predictive modeling.

Part 2: Machine Learning Algorithms and Visualizations

The 6 different algorithms applied to the National Hockey League Game Data from 2010-2020 are as follows:

1. Multiple Linear Regression
2. Logistic Regression
3. Support Vector Machine
4. K-Nearest Neighbors
5. Neural Network
6. Random Forest Classifier

Relevant visualizations/figures were created for each of the 6 algorithms and are also included.

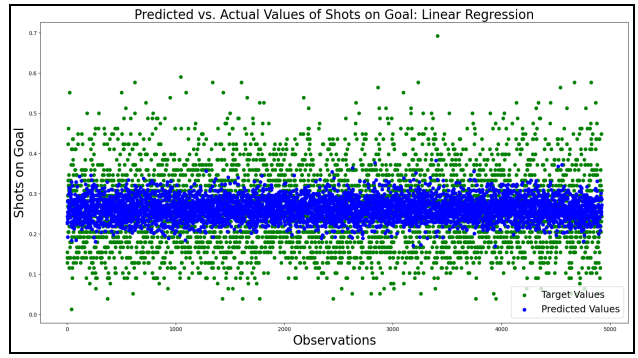
Linear Regression:

a. **Justification:** Multiple linear regression models were devised in an effort to predict both Shots, and Face Off Win Percentage based on several input columns. The reason that multiple linear regression was chosen to predict shots on goal and face off win percentage is due to the fact that both shots on goal and face off win percentage are real numerical values. Linear regression models predict continuous values, which effectively allows for the computation of the test mean squared error to determine the closeness of the predicted values against actual test values, a vital evaluation metric that comes from the model. In the context of hockey, shots on goal is a vital metric used by teams to not only predict the offensive firepower and the number of chances that a team brings to the game, but also to understand how a team should prepare for an opponent. If a team is known to be one that creates more chances than other teams, coaches and team staff should be ready to implement and practice defensive tactics to limit shots on goal against. Also, while less important, face off win percentage is often an indicator of how well teams may be able to generate chances in the offensive and defensive zones (or not), and it may be useful for teams to generate scenarios where, given certain values of variables, understand how face off win percentage plays into the equation. Typically, the greater percentage of face offs won, the more opportunities you have to score.

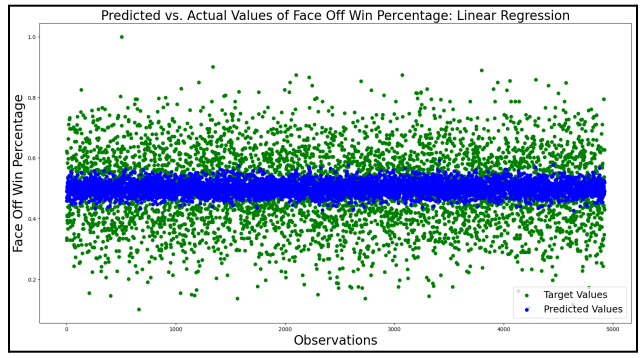
b. **Training and Tuning:** To train the linear regression model, the cleaned hockey data is split into separate X and Y variables for each of the predictive situations, where X is a data frame containing the input columns (Team ID, Won, Goals, Shots (only in predicting Face Off Win Percentage), Hits, Penalty Minutes, Power Play Opportunities, Face Off Win Percentage (only in predicting Shots), Takeaways, Giveaways, Blocked) and Y is the output variable (Shots or Face Off Win Percentage). From there, both X and Y are scaled using scikitLearn's "MinMaxScaler()". Then, the data is split into train (80%) and test (20%) sets. The models are built based on the training data. It should be noted that Ridge Regression, a variant of linear regression with l2 regularization, is also included in the code for experimentation.

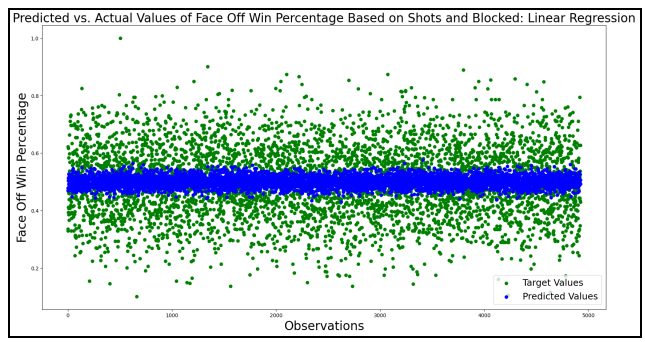
c. **Evaluation:** The predicted output values are achieved by evaluating the models on the test inputs. From here, the test mean squared error is printed, and the coefficients for the model are displayed. For Ridge Regression, there was little to no difference in the evaluation metrics. Finally, scatter plots are constructed displaying the predicted versus actual test values for Shots and Face Off Win Percentage for each of the observations in the test set.

i. **Predicting Shots on Goal:** The test mean squared error is 0.007066529830360674, and the coefficients for the model are [-0.0069215 , -0.01646886, 0.09485103, 0.03036507, -0.17671401, 0.11235889, 0.07059621, 0.06190719, 0.01675948, -0.12346148].



ii. **Predicting Face Off Win Percentage:** The test mean squared error is 0.014202169758787783 and the coefficients are [0.01109575, 0.01059329, 0.00373286, 0.15270665, -0.00452114, -0.02406804, 0.01019489, 0.02712426, 0.0506552 , -0.1438987] when using all other compatible input variables in the data frame. When limiting the input variables only to the two most impactful ones (Shots and Blocked), the test mean squared error is 0.014384001657897565 and the coefficients are [0.16770175, -0.11649071].





Note about R^2: The coefficient of determination, also known as R^2, is an evaluation metric typically used in linear regression that indicates the proportion of variance in the predicted variable explained by the input features. In the case of predicting shots on goal, the coefficient of determination never exceeded 0.1 (and was even lower when using less input features to predict shots on goal), and in the case of predicting face off win percentage, the coefficient of determination never exceeded 0.05 in several model runs. This indicates that the linear regression models' predictive variables are not explained well by the input features, or that the model does not capture all of the variables associated with shots on goal or face off win percentage. Hockey data is extremely messy and can be hard to model due to the games unpredictability on any given night. It has great amounts of variability across several different features. While there may be better ways to achieve a better R^2 value through different linear models, in many cases, within the scope of hockey, it may just be satisfactory enough to get a general gist of trends that take place within previously documented hockey data.

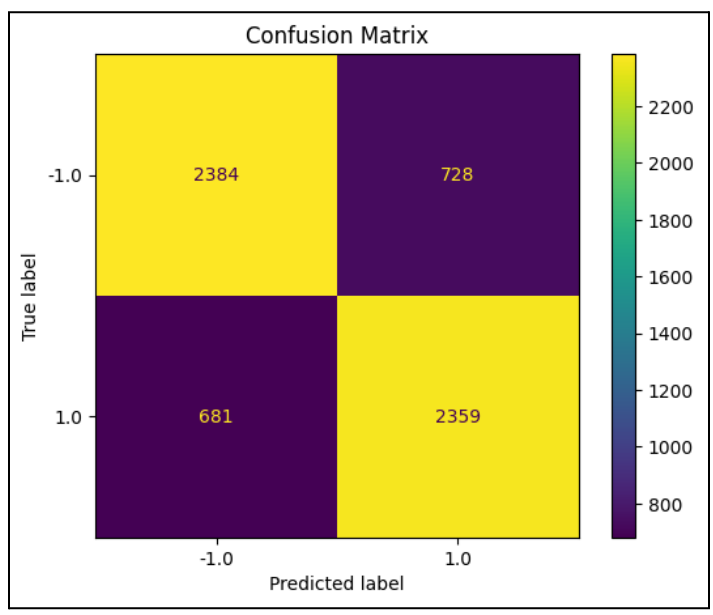
Logistic Regression:

a. **Justification:** A logistic regression model was used to predict whether or not a team won a game. Logistic regression can be used for binary classification, and that is done in the context of this hockey data, as win or loss is a binary value and can be modeled by 0 and 1 values. Understanding and predicting whether a team will win a game based on various possible statistics is useful for teams in assessing the strength of their opponents, or understanding what changes may need to be made during the offseason within an organization, whether it be in terms of players and/or coaches. To predict whether or not a team won a game, the following statistics were used to determine the result:
   i. Goals
   ii. Shots
   iii. Hits
   iv. Penalty Minutes

b. **Training and Tuning:**
   i. The model was trained using a variety of configurations:
      - Scaler: Scikit Min Max, Scikit Standard
      - C: 0.001, 0.01, 0.1, 1, 10, 100, 1000
      - Penalty: L1, L2
      - Solver: Liblinear, Saga
   ii. The model was tuned using Scikit's GridSearchCV to find the best configuration.

c. **Evaluation:** Accuracy scores were computed for each of the configurations, and the best configuration came out to be as follows:
   i. Scaler: StandardScaler
   ii. C: 0.01
   iii. Penalty: L2
   iv. Solver: LibLinear

   The listed best configuration above led to an accuracy rate of 78%. The confusion matrix on the right displays the results of the model on the test dataset.



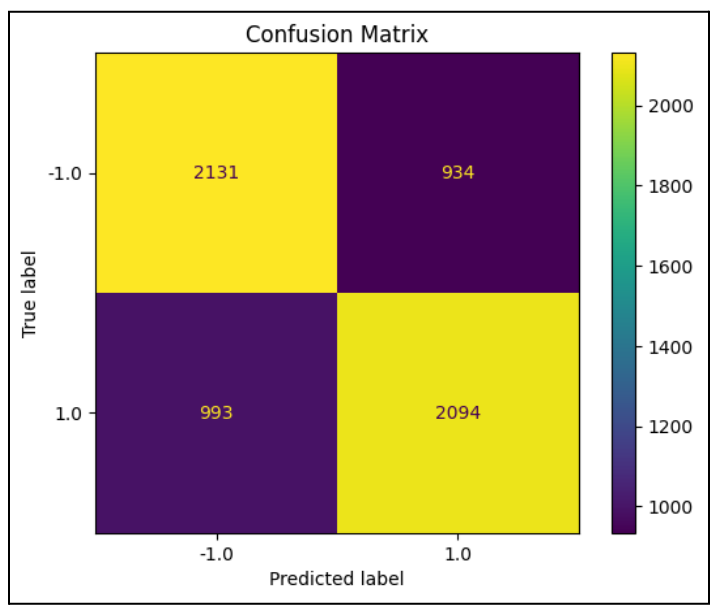
Support Vector Machine:

a. **Justification:** A support vector machine was used to predict whether or not a team played on their home ice or not. Support vector machines can be used for binary classification, and that is done in the context of this hockey data, as playing on home or away ice is a binary value and can be modeled by 0 and 1 values. Understanding and predicting whether a team played on home ice based on various possible statistics is useful in that it opens the door for other analysis in which teams can use this information to assess whether their opponents should be taken with an extra degree of caution in the event that they play well on home or away ice. To predict whether or not a team played on home or away ice, all columns in the database were used as input features with the exception of Date, Team ID, Head Coach, and Team Name. These columns were deemed to not have any significance in whether or not a team played on their home ice.

b. **Training and Tuning:**
   i. The model was trained using a variety of configurations:
      - Scaler: Scikit Min Max, Scikit Standard
      - C: 0.1, 1, 10
      - Gamma: 0.01, 0.001
   ii. The model was tuned using Scikit's GridSearchCV to find the best configuration.

c. **Evaluation:** Accuracy scores were computed for each of the configurations, and the best configuration came out to be as follows:
   i. Scaler: StandardScaler
   ii. C: 10
   iii. Gamma: 0.01

   The listed best configuration above led to an accuracy rate of 69%. This is not a relatively high accuracy, and this could be due to the fact that the data is not linearly separable, nor separable by casting the data through a kernel function, thus implying that SVM is not an appropriate model for testing this prediction. The more probable explanation for the low accuracy is a lack of data. Doubling or even tripling the dataset will most likely allow the SVM to separate Home and Away data more effectively. The confusion matrix on the right displays the results of the model on the test dataset.

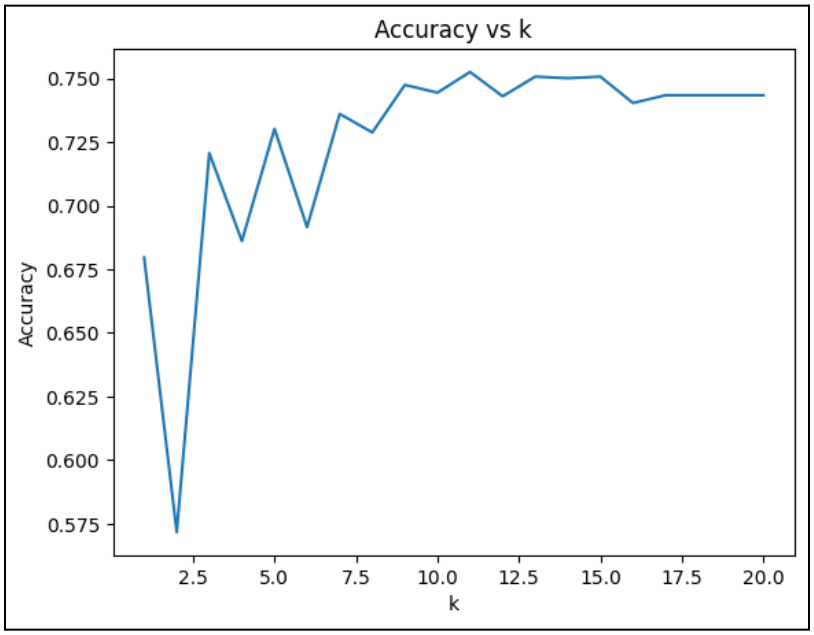


K-Nearest Neighbors:

a. **Justification:** A K-Nearest Neighbors algorithm was used to predict whether or not a game went into overtime. K-Nearest Neighbors can be used for classification, and that is done in the context of this hockey data, as determining if a game was settled in regulation or overtime is significant. Understanding and predicting whether a game went beyond regulation or not for a specific team has implications as it relates to the standings in the National Hockey League. If a team goes beyond regulation, they receive a point in the standings regardless, and come the end of the season, such points can be valuable in determining if a team makes the playoffs or not. Beyond that, teams that tend to go to overtime may fall into two categories: being able to hunker down and prevent their opponent from taking the lead, also known as playing for a point, or being able to crawl back into a game and comeback, also known as sending the game to overtime. Such behaviors may be important for opponents to understand. To predict whether or not a game was settled in regulation or overtime, the input features used were Penalty Minutes and Power Play Opportunities, as special teams play can be an indicator of whether a team fumbled a lead or came back to send their game to overtime. This particular feature set was chosen for KNN since there are only two dimensions in order to escape the Curse of Dimensionality with the model.

b. **Training and Tuning:** The model was tuned using different k values to compare which would yield the highest accuracy.

c. **Evaluation:** It was found that the best k value was 9 with an accuracy of 75%. Even though this was not the highest accuracy amongst all values of k, it was determined that this was "best" given the tradeoff between accuracy and the value of k, as a higher k value could yield underfitting. A plot showing the accuracy for values of k is shown to the right.



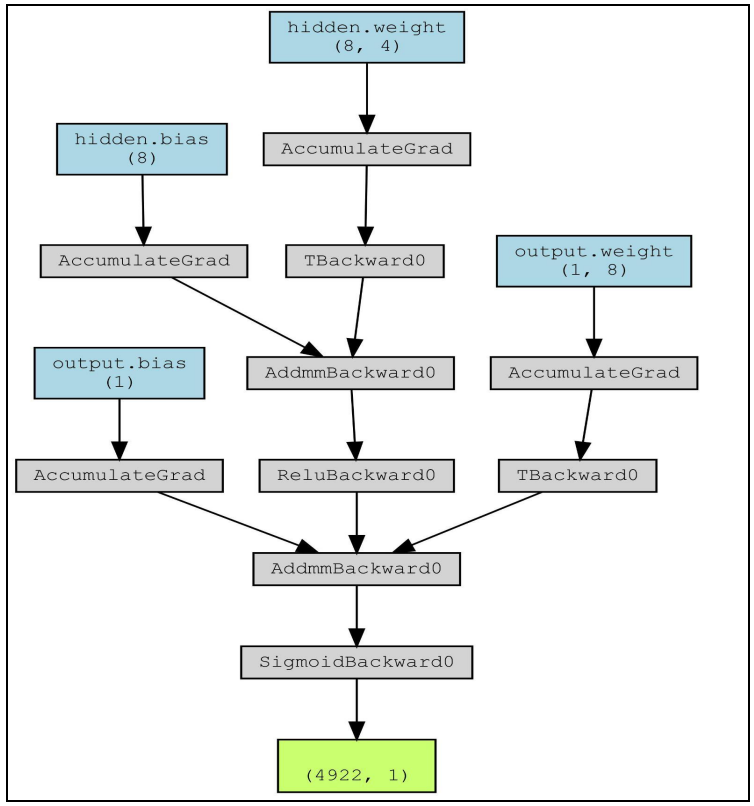
Neural Network:

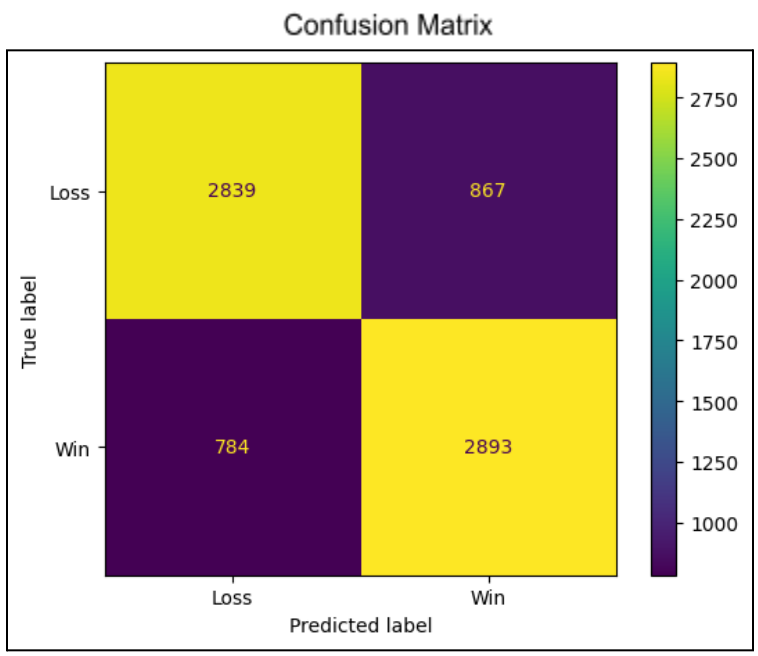
a. **Justification:** A PyTorch neural network was trained to predict whether or not a team won a game. Neural networks can be used for binary classification, and that is done in the context of this hockey data, as win or loss is a binary value and can be modeled by 0 and 1 values. As described before in the case of logistic regression, understanding and predicting whether a team will win a game based on various possible statistics is useful for teams in assessing the strength of their opponents, or understanding what changes may need to be made during the offseason within an organization, whether it be in terms of players and/or coaches. To predict whether or not a team won a game, the following statistics were used to determine the result:
   i. Goals
   ii. Shots
   iii. Hits
   iv. Penalty Minutes

b. **Training and Tuning:** The model was trained using the following configurations:
   i. Stack:
      - Linear (4 to 8)
      - ReLU
      - Linear (8 to 1)
      - Sigmoid Activation
   ii. Loss: Binary Cross Entropy
   iii. Optimizer: Adam
   iv. Epochs: 250
   v. Batch Size: 10
   vi. Learning Rate: 0.0001

   In order to find the model's general performance, K fold cross validation was applied during training using five splits of the dataset. For each split the best weights were determined and stored. At the end of training the weights giving the best accuracy were used as the final weights for the model.

c. **Evaluation:** The best accuracy from this model is 79%. This accuracy is comparable with the accuracy of the Logistic Regression, which also predicted whether or not a team won given the same statistics. This implies that this accuracy is most likely intrinsic to the dataset, and any improvements to the accuracy would most likely result from gathering more data to train the models on or processing the data differently. A visualization of the network architecture on the test set, encompassing 20% of the data frame, as well as the confusion matrix from testing the model are shown to the right.





Random Forest Classifier:

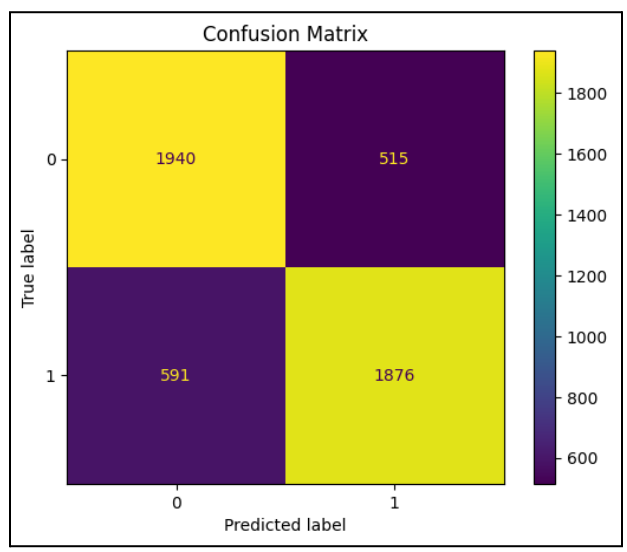
a. **Justification:** A Random Forest Classifier was trained to predict whether or not a team won a game. Random forests are an extension of decision tree classifiers, which while more visually complex than decision trees, often leads to better accuracy rates due to its ability to hinder overfitting when building and training the classifier. This classifier can be used for binary classification, and that is done in the context of this hockey data, as win or loss is a binary value and can be modeled by 0 and 1 values. As described before in the cases of logistic regression and neural networks, understanding and predicting whether a team will win a game based on various possible statistics is useful for teams in assessing the strength of their opponents, or understanding what changes may need to be made during the offseason within an organization, whether it be in terms of players and/or coaches. To predict whether or not a team won a game, the following input features were used to determine the result: Home or Away, Goals, Shots, Hits, Penalty Minutes, Power Play Opportunities, and Blocked.

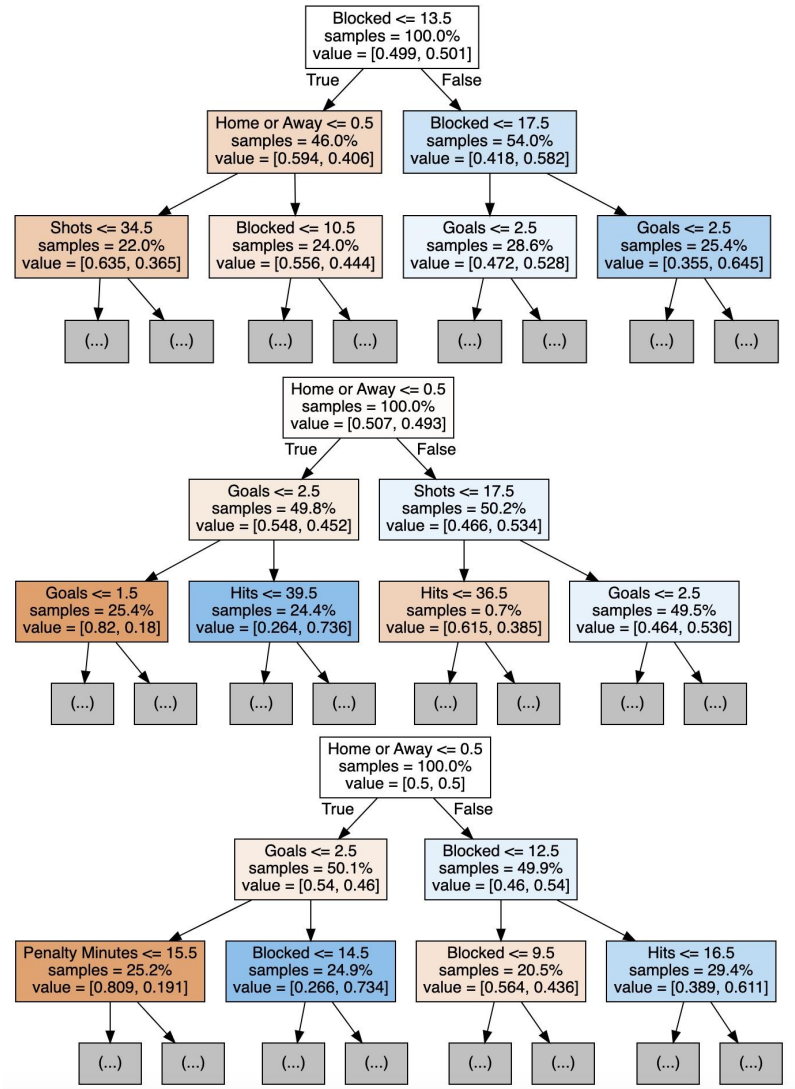
b. **Training and Tuning:** The model was tuned using the cross validation on the following configurations:
   i. Maximum depth of the tree (max_depth): random integer between 1 and 20
   ii. Number of trees in the forest (n_estimators): random integer between 50 and 100

   In order to find the model's general performance, K fold cross validation was applied during training using five splits of the dataset. For each split the best weights were determined and stored. At the end of training the weights giving the best accuracy were used as the final weights for the model.

c. **Evaluation:** Accuracy scores were computed for each of the configurations, and the best configuration resulting from one train-test split came out to be as follows:
   i. max_depth: 13
   ii. n_estimators: 77

   The best configuration above led to an accuracy rate of 78%, and is comparable with the accuracies of Logistic Regression and the Neural Network. Similar to those two binary classification models, this implies that this accuracy is most likely intrinsic to the dataset, and any improvements to the accuracy would most likely result from gathering more data to train the models on or processing the data differently. Visualizations of the confusion matrix and random forest structure are displayed to the right, and on the following page, respectively.





Important Knowledge Points From Implementing Each Algorithm:

- Linear Regression
    - Model implementation, while yielding low test mean squared error scored in predicting Shots and Face Off Win Percentage, did not yield good coefficient of determination (R^2 scores). This may be based on the tradeoff of achieving the best coefficients for the model .
    - Since hockey data is extremely messy and varies greatly from game to game, it makes sense that the coefficient of determination in predicting such variables with real values is not great.
    - Implementation of linear regression in hockey problems may be beneficial for other independent variables than those utilized in this report, and potentially less input features in an effort to simplify the problem.

- Logistic Regression, Neural Network, and Random Forest Classifier
    - Model implementations yielded satisfactory results in predicting whether a team won a game or not based on Goals, Shots, Hits, and Penalty Minutes for Logistic Regression and the Neural Network, while Home or Away, Power Play Opportunities, and Blocked were also considered in the Random Forest Classifier.
    - This model is useful for people in the hockey world to understand and predict whether a team will win a game based on specific input features. It is useful for teams in assessing the strength of their opponents, or understanding what changes may need to be made during the offseason within an organization, whether it be in terms of players and/or coaches.
    - Logistic Regression is rather simple and easy to implement for many groups of interest in the world of hockey. Neural Networks and Random Forest Classifiers on the other hand, while yielding similar accuracies and robustness, are more complex and may not be of greatest interest to certain audiences trying to understand hockey analyses.
- Support Vector Machine
    - Model implementation yielded lower accuracies then other models when trained on Goals, Shots, Hits, and Penalty Minutes to determine whether a game was played on home ice or not. This is of importance to teams in assessing opponents whose buildings they visit, or whose teams they play in their own arena, as it gives insight into which teams may play better in different environments.
    - This model may need to use a customized kernel method in order to make this specific dataset linearly separable to find a hyperplane which separates the data.
    - SVM is a rather simple model to implement for groups of interest in the world of hockey, but domain knowledge is necessary to understand how to linearize the dataset properly in order to obtain a model capable of predicting game outcome.
- K-Nearest Neighbors
    - Model implementation yielded satisfactory accuracies when trained on Penalty Minutes and Power Play Opportunities to determine whether or not a game was settled in overtime.
    - This model is useful for people in the hockey world to understand and predict whether a game will go into overtime based on the statistics of the game the model was trained on. It can be useful for getting a better grasp of standings across the league, and how much special teams (power play and penalty kill) play a role in sending a game to overtime.
    - This model is simple to implement for groups of interest in the hockey world. It requires no training time and excels in predicting targets which correspond to few features, which makes it ideal to determine whether a game was settled in regulation of overtime based on the input features used.

Understanding Problem Statement Questions:

1. Which teams statistically have performed best in a hockey sense, and which are expected to continue to perform well?
    - Part 1 allowed for the understanding of which teams have performed best in a hockey sense through our Exploratory Data Analysis.
    - Part 2 allowed for the building of predictive models to understand the factors that contribute most to winning hockey, which thus in turn *should* lead to good future performance.
2. Which game statistics, in addition to the number of goals scored, contribute most to winning hockey teams (i.e., shots on goal, hits, blocked shots, etc.)?
    - Part 2 allowed for the understanding of which input features contributed most to winning hockey teams. This was evident in our Logistic Regression, Neural Network, and Random Forest Classifier models, where Goals, Shots, Hits, and Penalty Minutes are of importance.
3. Are there any features of a hockey club off-ice that can explain team performance? How does having a veteran head coach influence team performance?
    - Part 2 allowed for a better grasp in understanding the influence of the head coach on team performance. Including versus omitting “Head Coach” as an input feature did not impact test mean squared error in the case of linear regression, nor accuracy rates in the cases of the other models significantly. Thus, the head coach may have a greater impact on predicting other features not considered in other hockey problems.
4. How does team performance fluctuate throughout the year? Is there a pattern as to when some teams perform better than others?
    - Part 1 allowed for the understanding of how team performance can fluctuate throughout the year through Exploratory Data Analysis in which the win percentage for each team was plotted by month. Slumps are regularities for all times, but those who win most are able to turn it around most quickly in an effort to make the playoffs.