Patrick Hayes - 4/10/2024

#1 Question:

Using the attached dataset (mlb_pitch_velo_assessment.csv), build a model that predicts whether the first pitch of a baseball game by each starting pitcher will be faster than 89.95 mph.

#1 Answer:

For this one, I try my best to bring my thought process to life. I've added comments throughout, and even left in a few cells that I'd normally clean up while in the flow of making progress.

At the end, I'll include a list of improvements I'd make with more time.

Light EDA and Data Cleaning

What am I working with? I want to find out. I briefly looked for patterns, inconsistencies and got a general sense of what's included in the data. Let's import it, along with all necessary packages and take a look.

In [ ]:

```
#Importing all the needs
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, accuracy_score, classification_report, roc_auc_score, prec
from sklearn.model_selection import GridSearchCV, cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings

#I love these settings for Juypter Notebooks, I want to see all the things!
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
```

```python
pd.set_option('display.width', 1000)
pd.options.display.float_format = None
warnings.filterwarnings('ignore') #added this for this project to improve readability of the notebook
```

In [ ]:
```python
df = pd.read_csv('mlb_pitch_velo_assessment.csv')
df.head()
```

Out[ ]:

	pitch_id	game_id	season	date	home_team_id	home_team_name	away_team_id	away_team_name	venue_id	pit
0	1242036	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
1	1242037	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
2	1242038	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
3	1242039	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
4	1242040	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	

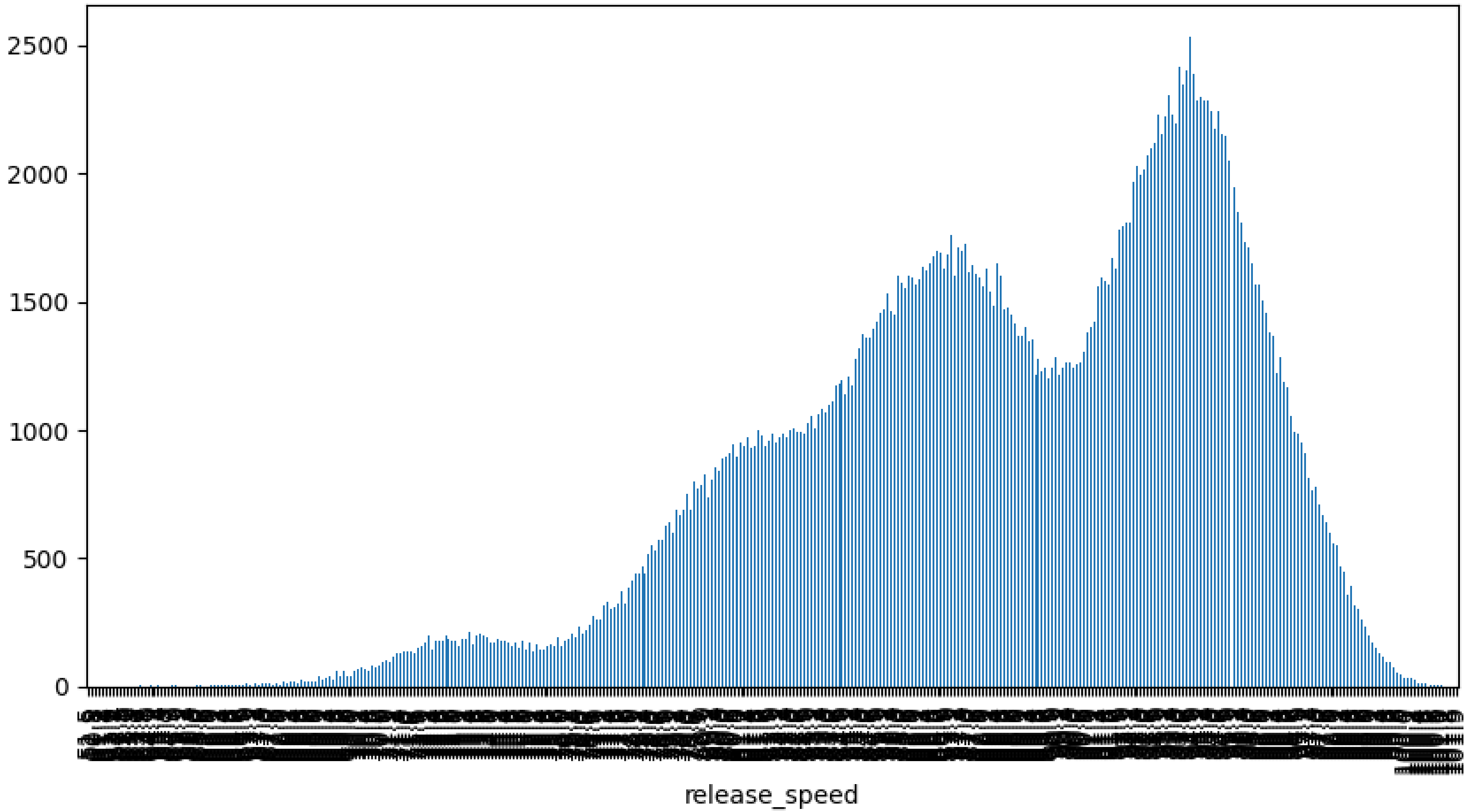
In [ ]:
```python
#lots of data! But will we need it?
df.shape
```

Out[ ]: (291684, 21)

In [ ]:
```python
#How does the distribution of the target variable look?
#Whoaaa Nelly says Keith Jackson. We can do better visually.

df['release_speed'].value_counts().sort_index().plot(kind='bar', figsize=(10,5))
```

Out[ ]: <Axes: xlabel='release_speed'>

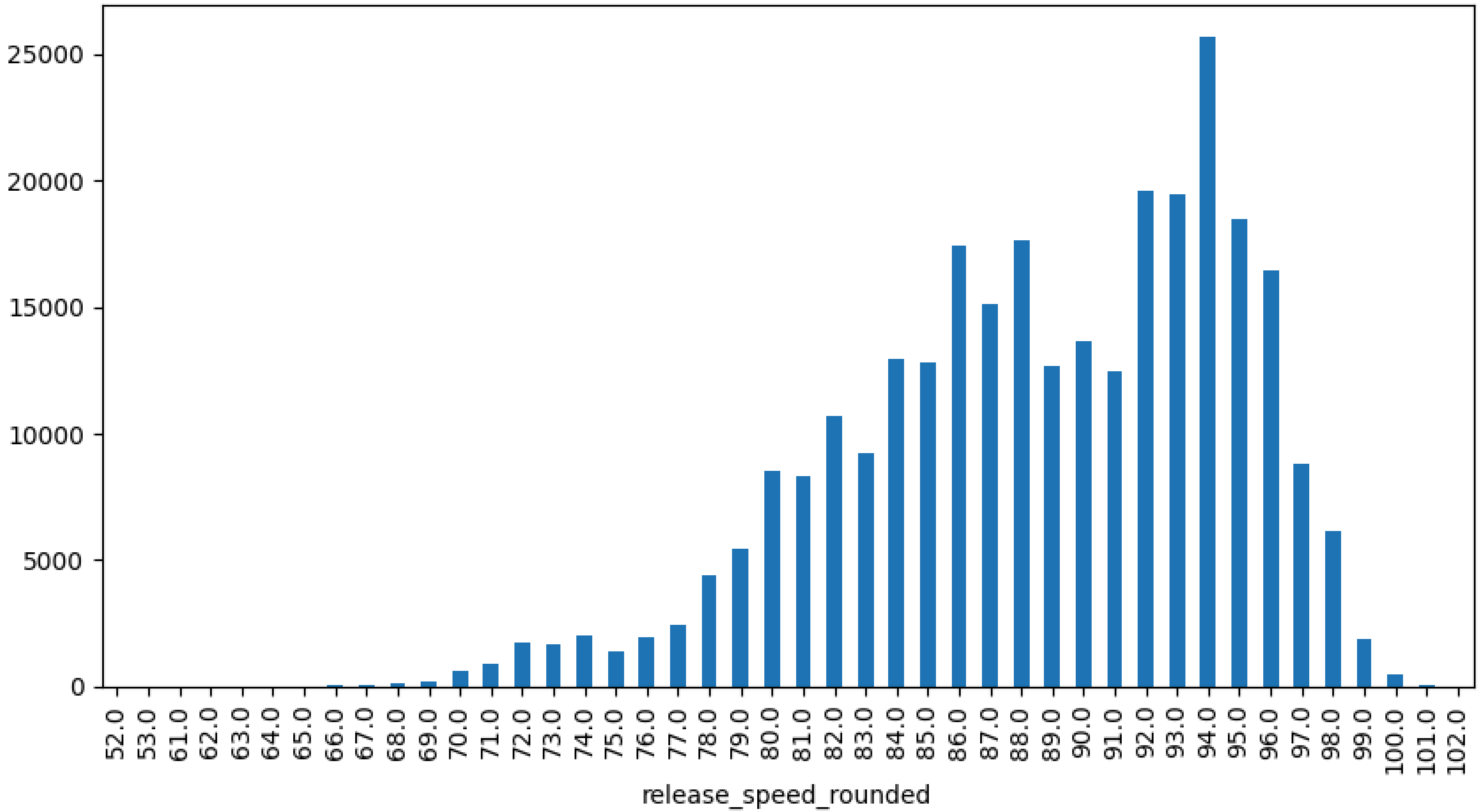


In [ ]:
```
#For simplicity, rounding each value to get a feel for the distribution among all pitches.
#Round the 'release_speed' values to the nearest whole number
df['release_speed_rounded'] = df['release_speed'].round()

# Generate the bar plot for the rounded 'release_speed' values
df['release_speed_rounded'].value_counts().sort_index().plot(kind='bar', figsize=(10,5))

#Much better!
```

Out[ ]: <Axes: xlabel='release_speed_rounded'>



In [ ]:

```
#let's check for missing values

missing_count = df.isnull().sum()
na_count = df.isna().sum()
print(missing_count), print(na_count)
```

```
pitch_id                   0
game_id                    0
season                     0
date                       0
home_team_id               0
home_team_name             0
away_team_id               0
away_team_name             0
venue_id                   0
pitch_number               0
pitcher_id                 0
pitcher_name               0
batter_id                  0
batter_name                0
pre_pitch_inning           0
is_top_half                0
pre_pitch_outs             0
pre_pitch_balls            0
pre_pitch_strikes          0
pitch_type               111
release_speed            120
release_speed_rounded    120
dtype: int64
pitch_id                   0
game_id                    0
season                     0
date                       0
home_team_id               0
home_team_name             0
away_team_id               0
away_team_name             0
venue_id                   0
pitch_number               0
pitcher_id                 0
pitcher_name               0
batter_id                  0
batter_name                0
pre_pitch_inning           0
is_top_half                0
pre_pitch_outs             0
pre_pitch_balls            0
pre_pitch_strikes          0
```

```
pitch_type                111
release_speed             120
release_speed_rounded     120
dtype: int64
```

Out[ ]:
```
(None, None)
```

In [ ]:
```
#let's drop those missing values since they are small values

df = df.dropna()
```

In [ ]:
```
#check again, we're good!

missing_count = df.isnull().sum()
na_count = df.isna().sum()
print(missing_count), print(na_count)
```

```
pitch_id                  0
game_id                   0
season                    0
date                      0
home_team_id              0
home_team_name            0
away_team_id              0
away_team_name            0
venue_id                  0
pitch_number              0
pitcher_id                0
pitcher_name              0
batter_id                 0
batter_name               0
pre_pitch_inning          0
is_top_half               0
pre_pitch_outs            0
pre_pitch_balls           0
pre_pitch_strikes         0
pitch_type                0
release_speed             0
release_speed_rounded     0
dtype: int64
pitch_id                  0
game_id                   0
season                    0
date                      0
home_team_id              0
home_team_name            0
away_team_id              0
away_team_name            0
venue_id                  0
pitch_number              0
pitcher_id                0
pitcher_name              0
batter_id                 0
batter_name               0
pre_pitch_inning          0
is_top_half               0
pre_pitch_outs            0
pre_pitch_balls           0
pre_pitch_strikes         0
```

```
pitch_type               0
release_speed            0
release_speed_rounded    0
dtype: int64
```

Out[ ]: (None, None)

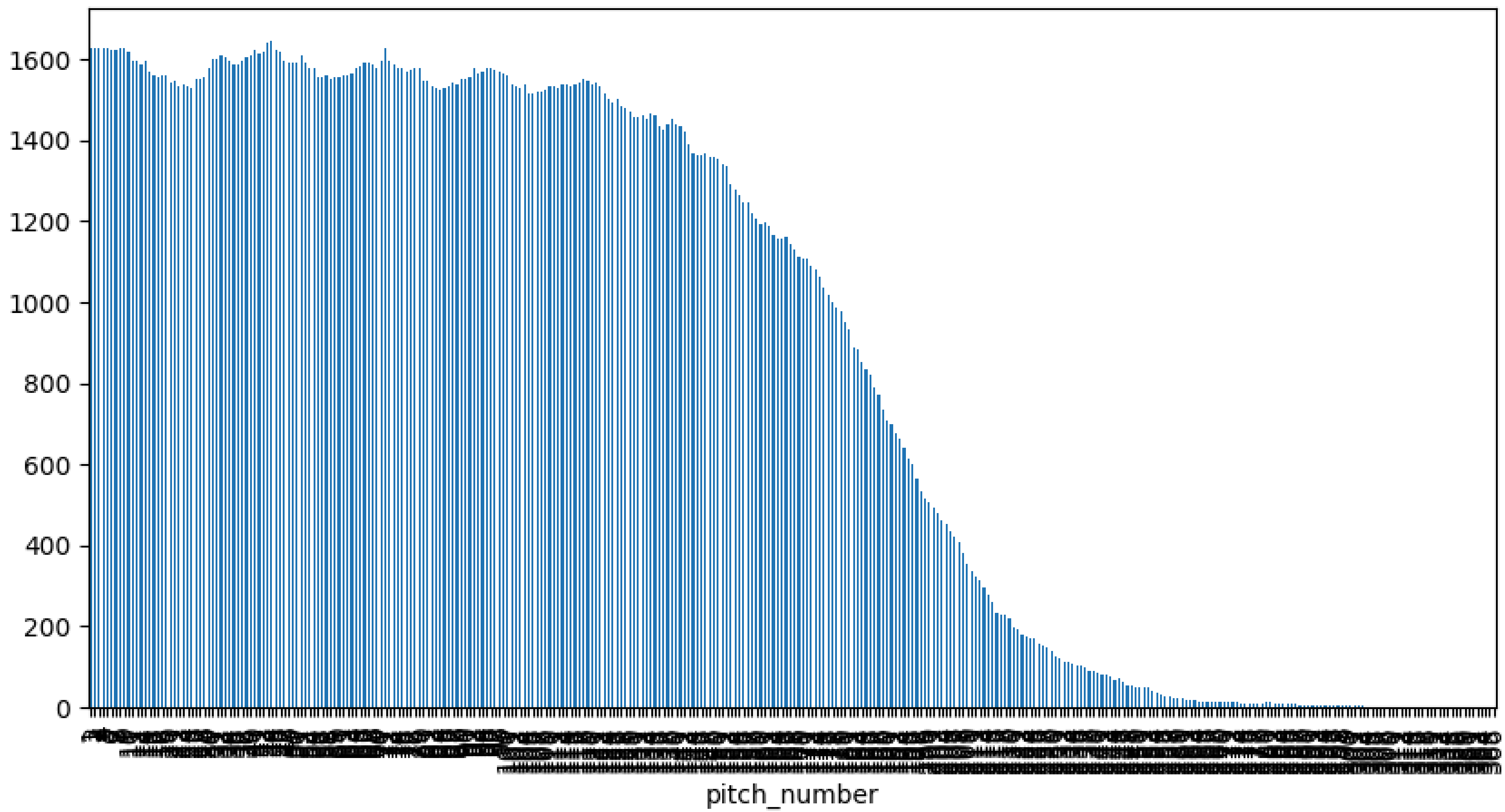
In [ ]:
```
#quick review of the data types of each. Some popout as needing to be converted to categorical or removed
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 291564 entries, 0 to 291683
Data columns (total 22 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   pitch_id               291564 non-null  int64  
 1   game_id                291564 non-null  int64  
 2   season                 291564 non-null  int64  
 3   date                   291564 non-null  object 
 4   home_team_id           291564 non-null  int64  
 5   home_team_name         291564 non-null  object 
 6   away_team_id           291564 non-null  int64  
 7   away_team_name         291564 non-null  object 
 8   venue_id               291564 non-null  int64  
 9   pitch_number           291564 non-null  int64  
 10  pitcher_id             291564 non-null  float64
 11  pitcher_name           291564 non-null  object 
 12  batter_id              291564 non-null  int64  
 13  batter_name            291564 non-null  object 
 14  pre_pitch_inning       291564 non-null  int64  
 15  is_top_half            291564 non-null  int64  
 16  pre_pitch_outs         291564 non-null  int64  
 17  pre_pitch_balls        291564 non-null  int64  
 18  pre_pitch_strikes      291564 non-null  int64  
 19  pitch_type             291564 non-null  object 
 20  release_speed          291564 non-null  float64
 21  release_speed_rounded  291564 non-null  float64
dtypes: float64(3), int64(13), object(6)
memory usage: 51.2+ MB
```

In [ ]:
```
#What else can we potentially remove later on?
#Pitch number is a good candidate since we're looking for the first pitch by a starting pitcher.
```

```
df['pitch_number'].value_counts().sort_index().plot(kind='bar', figsize=(10,5))
```

Out[ ]: <Axes: xlabel='pitch_number'>



In [ ]:
```
#lets's look at the first game provided to see what we can glean from the data
df[df['game_id'] == 54842].head()
```

Out[ ]:

	pitch_id	game_id	season	date	home_team_id	home_team_name	away_team_id	away_team_name	venue_id	pit
0	1242036	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
1	1242037	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
2	1242038	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
3	1242039	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
4	1242040	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	

In [ ]:

```
#Hendricks allowed a few baserunneres before getting an out, we need to account for this.

#weird, only one SP for this game too.
df[df['game_id'] == 54842].pitcher_id.unique()
```

Out[ ]:

```
array([4708.])
```

Feature Engieering

New features will be needed for this data set in order to properly model it. Let's create them!

In [ ]:

```
#convert the date to month and day
#month is important with baseball
df['month'] = pd.to_datetime(df['date']).dt.month
df['day'] = pd.to_datetime(df['date']).dt.day
```

In [ ]:

```
#accounting for SPs who don't get an immediate out.
df['is_first_pitch'] = ~df.duplicated(subset=['game_id', 'pitcher_id'])
```

In [ ]:
```
#indicate if the first pitch of the game is thrown by the SP
#Here I want to make sure it's the first pitch of the top or bottom of the 1st inning.
#I know "openers" are a thing, I'm ignoring that for now.

df['sp_first_pitch'] = np.where(
                        (df['pre_pitch_inning'] == 1) &
                        (df['pre_pitch_outs'] == 0) &
                        (df['pre_pitch_balls'] == 0) &
                        (df['pre_pitch_strikes'] == 0) &
                        (df['is_first_pitch']),
                        1, 0)
```

In [ ]:
```
#Well our data set shrunk a bit, but that's okay.
df['sp_first_pitch'].value_counts()
```

Out[ ]:
```
sp_first_pitch
0    288338
1      3226
Name: count, dtype: int64
```

In [ ]:
```
#building our target variable
df['faster_than_8995'] = df['release_speed'] > 89.95
```

In [ ]:
```
#Let's see how we're looking
df.head()
```

Out[ ]:

	pitch_id	game_id	season	date	home_team_id	home_team_name	away_team_id	away_team_name	venue_id	pit
0	1242036	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
1	1242037	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
2	1242038	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
3	1242039	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	
4	1242040	54842	2021	2021-04-01	6	Chicago Cubs	22	Pittsburgh Pirates	29	

In [ ]:

```
#these columns can be all be removed

df = df.drop(columns=['game_id', 'pitch_id', 'pitcher_name', 'batter_name', 'home_team_name', 'away_team_na
```

In [ ]:

```
#quick pulse check to confirm
df.head()
```

Out[ ]:

	season	home_team_id	away_team_id	venue_id	pitch_number	pitcher_id	batter_id	pre_pitch_inning	is_top_half
0	2021	6	22	29	1	4708.0	5199	1	1
1	2021	6	22	29	2	4708.0	5199	1	1
2	2021	6	22	29	3	4708.0	5199	1	1
3	2021	6	22	29	4	4708.0	5199	1	1
4	2021	6	22	29	5	4708.0	5199	1	1

In [ ]:

```
#let's reduce the dataset down to just starting pitchers throwing the first pitch
```

```
df_sps = df[df['sp_first_pitch'] == 1]
```

In [ ]:
```
df_sps.head()
```

Out[ ]:

	season	home_team_id	away_team_id	venue_id	pitch_number	pitcher_id	batter_id	pre_pitch_inning	is_top_hal
0	2021	6	22	29	1	4708.0	5199	1	1
63	2021	7	26	9	1	5447.0	5975	1	1
136	2021	23	1	8	1	3616.0	6061	1	1
229	2021	14	5	26	18	5079.0	5722	1	0
316	2021	20	12	27	8	931.0	4823	1	0

In [ ]:
```
#last thing to do is to convert pitch_type to a categorical variable
df_sps['pitch_type'] = df_sps['pitch_type'].astype('category').cat.codes
```

In [ ]:
```
df_sps.head()
```

Out[ ]:

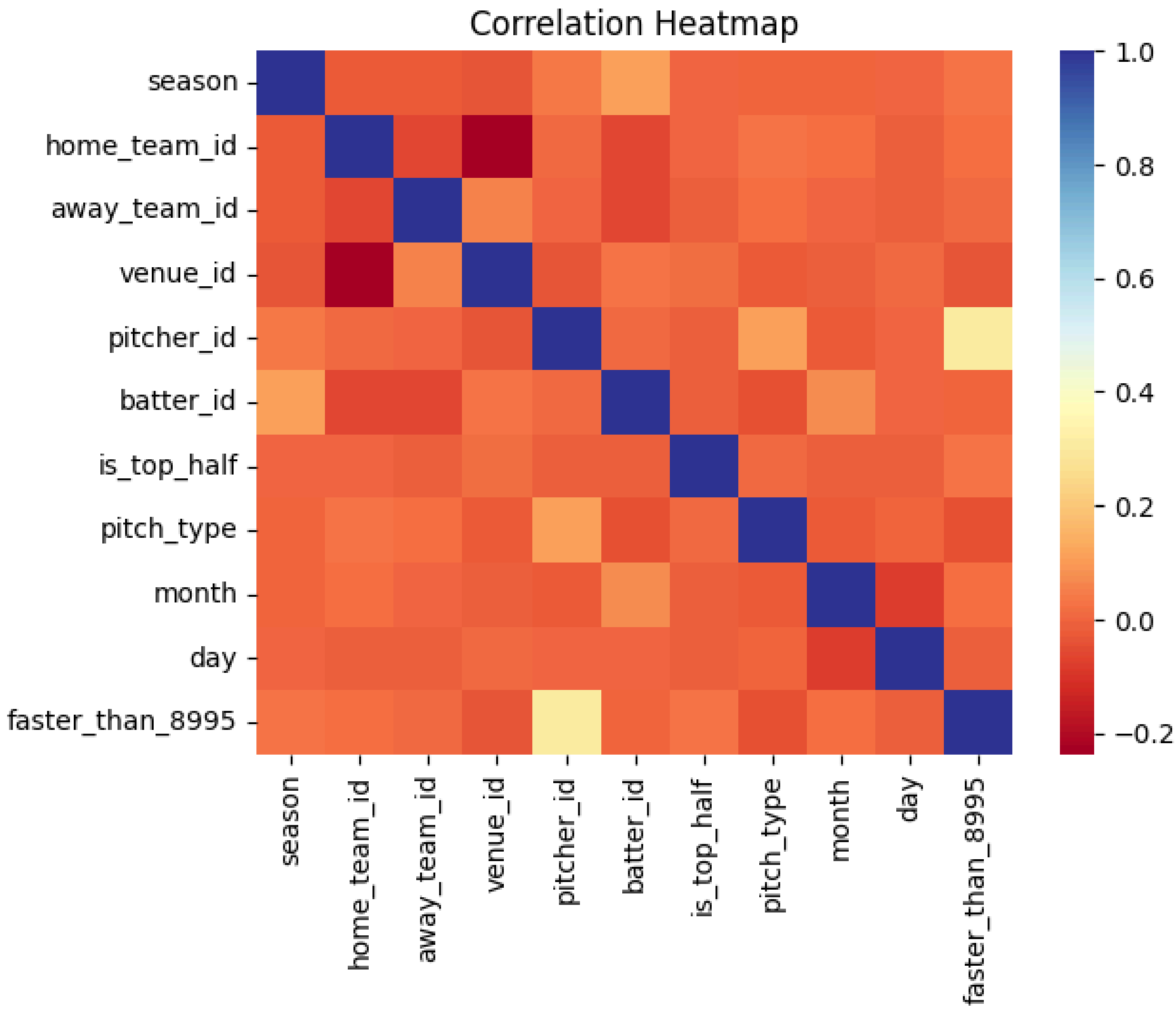
	season	home_team_id	away_team_id	venue_id	pitch_number	pitcher_id	batter_id	pre_pitch_inning	is_top_hal
0	2021	6	22	29	1	4708.0	5199	1	1
63	2021	7	26	9	1	5447.0	5975	1	1
136	2021	23	1	8	1	3616.0	6061	1	1
229	2021	14	5	26	18	5079.0	5722	1	0
316	2021	20	12	27	8	931.0	4823	1	0

In [ ]:
```
#additonal clean up before correlation matrix spot check
df_sps = df_sps.drop(columns=['sp_first_pitch', 'release_speed', 'pre_pitch_outs', 'pre_pitch_balls', 'pre_
```

In [ ]:
```
#let's look at the correlation matrix to see if there are any features that are highly correlated with each
corr = df_sps.corr()

# Plot heatmap of correlation matrix
sns.heatmap(corr, annot=False, cmap='RdYlBu')
```

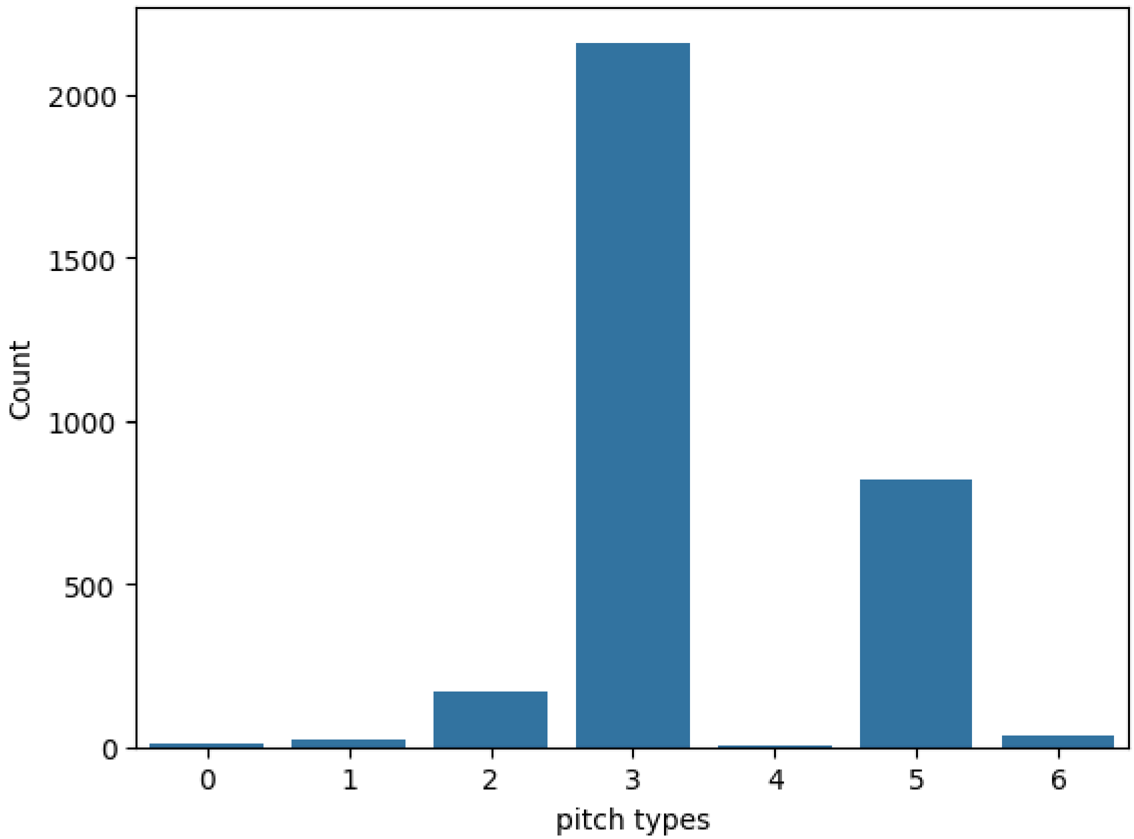
```
plt.title('Correlation Heatmap')
plt.show()
```



In [ ]:
```
#ugh, not intrigued.
```

In [ ]:
```
#quick look at the pitch types, with more time could combine or drop the less frequent ones.
sns.countplot(x='pitch_type', data=df_sps)
plt.xlabel('pitch types')
```

```python
plt.ylabel('Count')
plt.show()
```



In [ ]:
```python
#did not anticipate seeing a negative correlation coming with pitch type and if it's faster than 89.95 mph
pitch_type_speed_corr = df_sps['pitch_type'].corr(df_sps['faster_than_8995'])
pitch_type_speed_corr
```

Out[ ]: -0.04685689818927491

Model Selection and Fitting

For simplicity and speed, I chose to use a logistic regression for this binary classification problem.

I used an 80/20 split for train/test sets and did slight parameter turning with grid search for the model. It becomes an inblaanaced class problem, so I use the `class_weight` parameter to help with that. With more time, I'd introduce other techniques such as SMOTE to help counteract this.

In [ ]:
```
#Straight forward splits
X = df_sps.drop(['faster_than_8995'], axis=1)
y = df_sps['faster_than_8995']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:
```
#little bit of class imbalance, but we can work this and address it with more time
df_sps['faster_than_8995'].value_counts()
```

Out[ ]:
```
faster_than_8995
True     2564
False     662
Name: count, dtype: int64
```

In [ ]:
```
#simple logistic regression for starters
lr = LogisticRegression(max_iter=1000, random_state=42, solver='liblinear')
lr.fit(X_train, y_train)
# y_pred = lr.predict(X_test) #removed to use the best model from the grid search

param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    }

grid_search = GridSearchCV(lr, param_grid, cv=5, scoring='f1', error_score='raise')

grid_search.fit(X_train, y_train)
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

Model Evaluation

In [ ]:
```python
#let's find out the best parameters and the best model
#then, we can use the best model to predict the target variable for the test set

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
best_score = grid_search.best_score_
y_pred = best_model.predict(X_test)
```

In [ ]:
```python
#Harmonic mean check for performance of the model...overfitting?
f1_score(y_test, y_pred)
```

Out[ ]:
```
0.8725663716814159
```

In [ ]:
```python
#I love reviewing the classification report.
#It allows me to see the precision, recall, and f1-score for each class
#I'm seeing the reflection of class imbalance here and it's struggling to accurately identify pitches slowe
classification_rep = classification_report(y_test, y_pred)
print(classification_rep)
```

```
              precision    recall  f1-score   support

       False       0.31      0.07      0.11       133
        True       0.80      0.96      0.87       513

    accuracy                           0.78       646
   macro avg       0.55      0.51      0.49       646
weighted avg       0.70      0.78      0.72       646
```

In [ ]:
```python
#Cross validation to check for overfitting
#no immediate concerns that we haven't already addressed
cross_val_scores = cross_val_score(grid_search, X, y, cv=5)

# Print the cross-validation scores and the mean score
print("Cross-validation Scores:", cross_val_scores)
print("Mean Cross-validation Score:", cross_val_scores.mean())
```

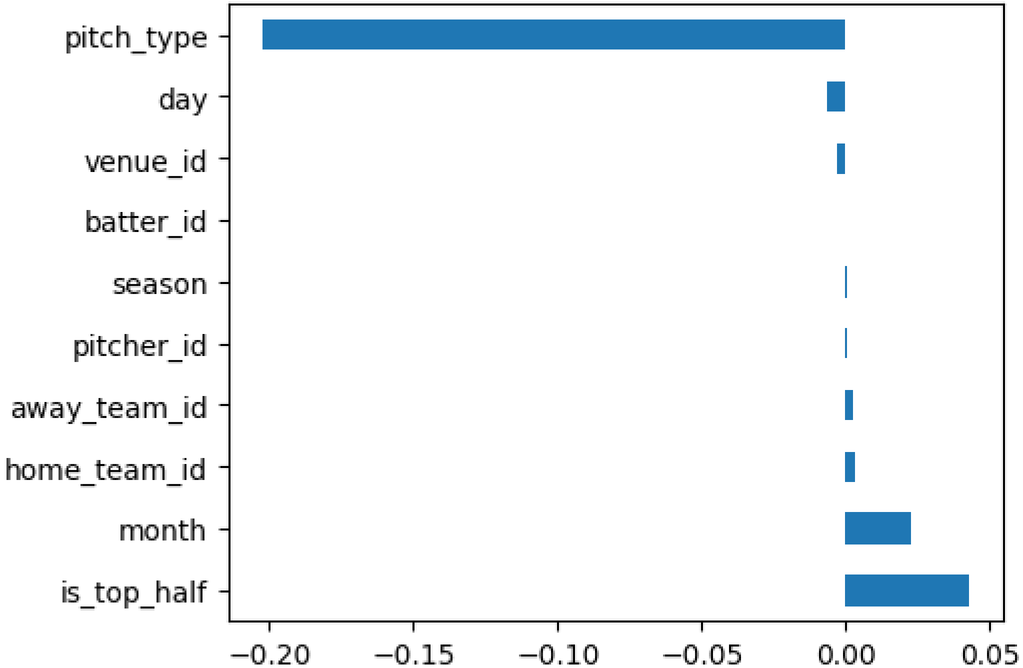
```
Cross-validation Scores: [0.85245902 0.87908208 0.87860262 0.88261253 0.86577778]
Mean Cross-validation Score: 0.8717068060644209
```

In [ ]:
```python
#feature weights and intercept from the best model
#helps identify which features are most/least important
#I see you month!
#also Interesting to see top half of the inning with a positive weight.
#Could future engineer the data to split out top and bottom half of the inning.
feature_weights = best_model.coef_[0]
intercept = best_model.intercept_

weights_with_names = pd.Series(data=feature_weights, index=X.columns)

# print("Intercept:", intercept)
print(weights_with_names)
#also like seeing it visually
sorted_coefficients = weights_with_names.sort_values(ascending=True)
sorted_coefficients.plot(kind='barh', figsize=(5,4)).invert_yaxis()
plt.show()
```

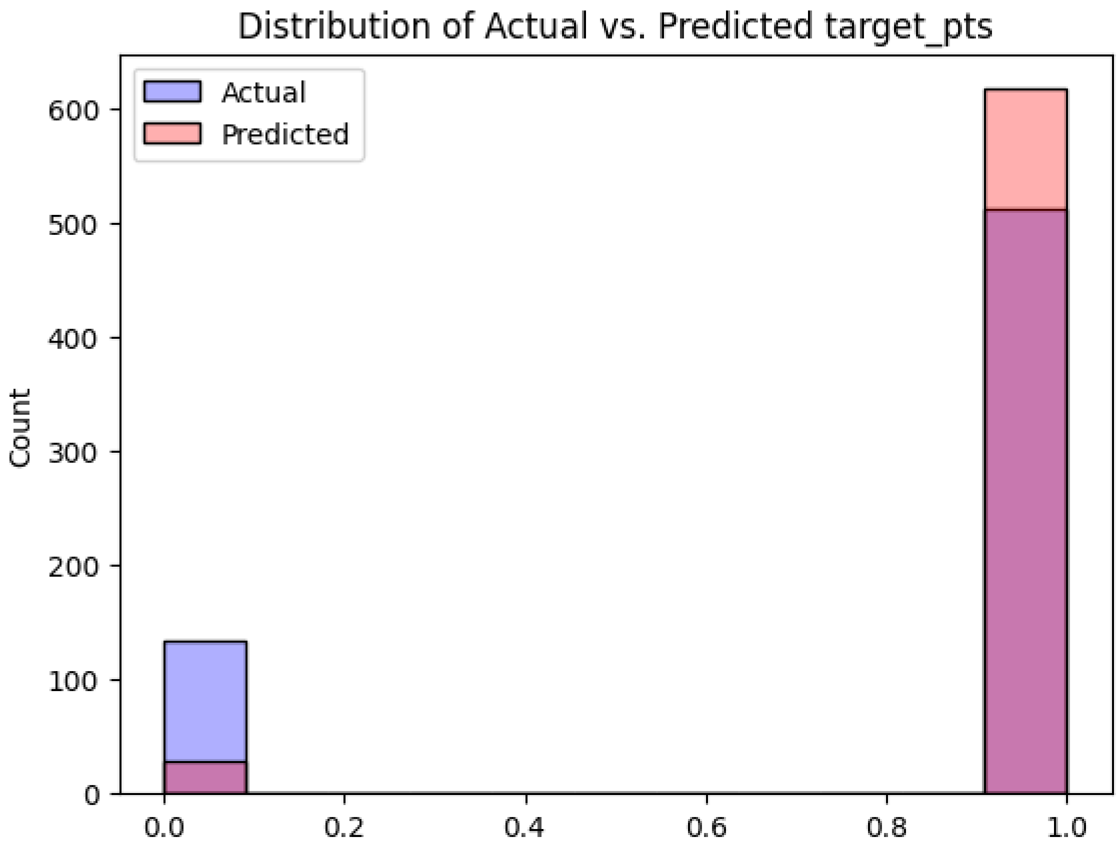
```
season          0.000166
home_team_id    0.003524
away_team_id    0.002379
venue_id       -0.003139
pitcher_id      0.000384
batter_id       0.000010
is_top_half     0.042684
pitch_type     -0.201945
month           0.022875
day            -0.006281
dtype: float64
```



In [ ]:
```
#let's work towards seeing actuals vs predicted
y_pred_proba = best_model.predict_proba(X_test)[:, 1]
y_pred_predict = [1 if prob > 0.5 else 0 for prob in y_pred_proba]
```

In [ ]:
```
#quick review of the distribution of actual vs predicted
#We aren't good at predicting pitchers slower than 89.95 mph
#with more time I'd clean up the viz
sns.histplot(y_test, color='blue', label='Actual', kde=False,alpha=0.3)
sns.histplot(y_pred_predict, color='red', label='Predicted', kde=False, alpha=0.3)

plt.xlabel('')
plt.ylabel('Count')
plt.title('Distribution of Actual vs. Predicted target_pts')
plt.legend()
plt.show()
```



In [ ]:

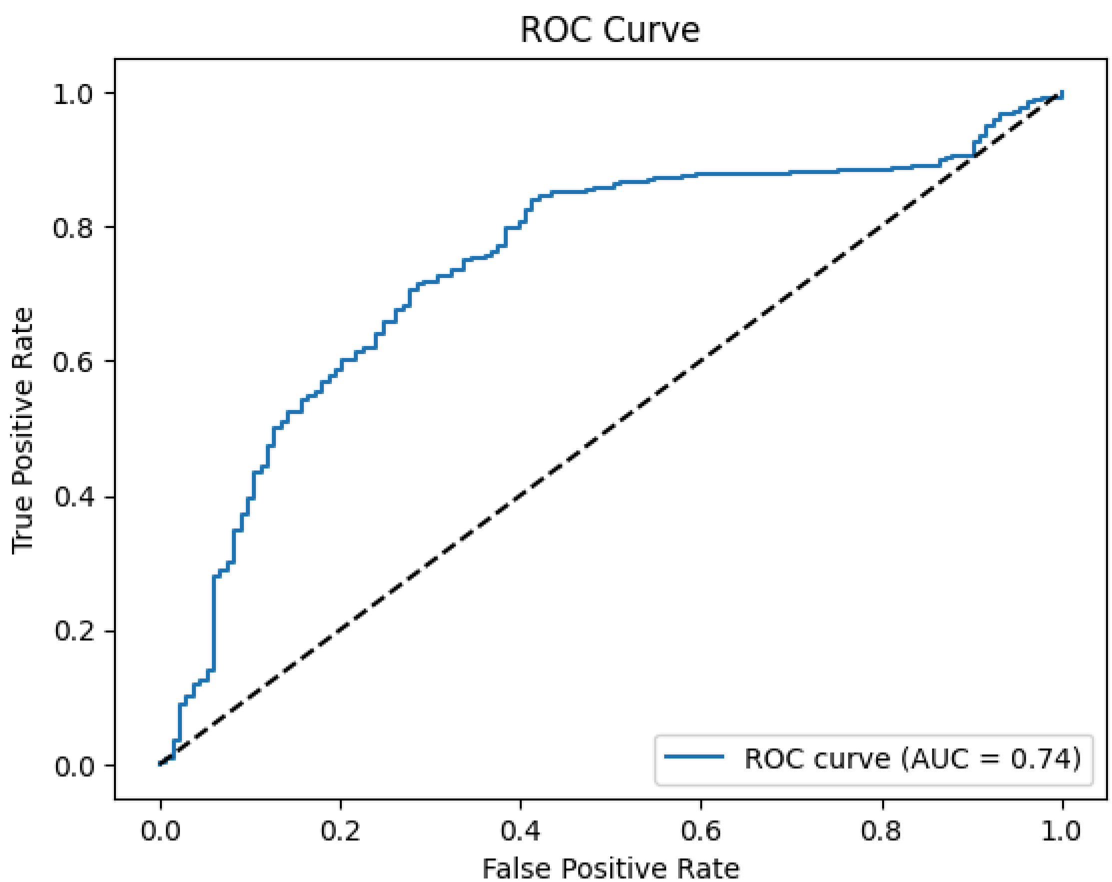
```
#While we're here, let's plot the ROC curve and calculate the AUC score
#reconfirms that we're good at guessing pitches will be faster than 89.95mph

# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_pred_proba)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (AUC = {:.2f})'.format(auc_score))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
```

```
plt.legend(loc='lower right')
plt.show()
```



Model Analysis

Overall, this is an okay first step toward creating a model to predict whether each starting pitcher's first pitch in a baseball game will be faster than 89.95 mph. As alluded to throughout, we're dealing with a class imbalance problem. The model likes to think all pitches will be faster and does a poor job of predicting those that will be slower.

Improvements

With more time, here is a list of actions I'd consider:

- finding out why 89.95 MPH was the selected threshold :)
- obtaining more data since I reduced the size of the dataset a bit.
- Add additional features such as: pitcher's last start and if the first pitch went over 89.95 mph, split out top_of_inning feature to tease out if the home or away pitcher typically throws faster (might not add much to the model, but I'd find it interesting), days since last start, if SP has had TJ surgery or not, etc.!
- Optimize many parts of the code I wrote
- Integrate Class imbalance techniques such as SMOTE or RandomOverSampler
- More models! I'd implement a function that would allow me to test different models and compare their performance quickly. A few models I'd start with are: Random Forest, Gradient Boosting, and XGBoost.
- Create better visualizations

Was a fun one!

#2 Question:

The spread for an NBA game is 76ers -7.5 at -110 and Lakers +7.5 at -110, and the over under for the game is 235. The money line markets have the 76ers at -320 and the Lakers +260. How many points is each team expected to score? What is the implied win probability for each team? How much "vig" is the book taking on the spread and money line markets?

#2 Answer:

I remember Keith going over the approach to this at the 2023 Sloan Sports Analytics Conference but using NFL spreads/totals. I've converted many moneylines to percentage odds before and leveraged a function I use often. For the vig, we just add up the implied probabilities and subtract from 100% (or 1 if they're decimals like the function I wrote).

Here are the answers based on all the python code below.

- How many points is each team expected to score?
  - 76ers: 121.25
  - Lakers: 113.75
- What is the implied win probability for each team?
  - 76ers: 76.19%

- Lakers: 27.78%
- How much "vig" is the book taking on the spread and money line markets?
  - Vig: 3.97%

In [ ]:
```
#provided data
over_under = 235
sevensixers = -7.5
lakers = 7.5
sevensixers_moneyline = -320
lakers_moneyline = 260

#helper functions I wrote to calculate expected scores and vig

def team_expected_scores(total_points, favorite_spread, under_dog_spread):
    """
    Calculate the expected scores for the favorite and underdog based on the total points and point spread
    """
    favorite_expected_score = over_under / 2 - favorite_spread / 2
    underdog_expected_score = over_under / 2 - under_dog_spread / 2
    print(f"With a point total of {total_points} and a point sprad of {under_dog_spread}, the expected sco
    return (favorite_expected_score, underdog_expected_score)

def convert_moneyline_to_percentage_odds(moneyline_odds):
    """
    Convert moneyline odds to percentage odds, round to 4 decimal places
    Taken from what I use in previous projects
    """
    if moneyline_odds >= 0:
        return round(100 / (moneyline_odds + 100), 4)
    else:
        return round((moneyline_odds) / (moneyline_odds - 100), 4)

def vig_calculation(favorite, underdog):
    """
    Calculate how much the sportsbook is squeezing from the bet, aka the vig.
    """
    favorite_percentage = convert_moneyline_to_percentage_odds(favorite)
    underdog_percentage = convert_moneyline_to_percentage_odds(underdog)
    vig = (1 - (favorite_percentage + underdog_percentage))
    vig_rounded = abs(round(vig * 100, 3))
```

```
    print(f"The vig is: {vig_rounded}%")
    return vig_rounded
```

In [ ]:
```
#Calculate the expected scores
team_expected_scores(over_under, sevensixers, lakers)
```

```
With a point total of 235 and a point sprad of 7.5, the expected score is 121.25 - 113.75
```

Out[ ]:
```
(121.25, 113.75)
```

In [ ]:
```
#Implied probability of 76ers winning
sevensixers_win = convert_moneyline_to_percentage_odds(sevensixers_moneyline)
sevensixers_win
```

Out[ ]:
```
0.7619
```

In [ ]:
```
#Implied probability of Lakers winning
lakers_win = convert_moneyline_to_percentage_odds(lakers_moneyline)
lakers_win
```

Out[ ]:
```
0.2778
```

In [ ]:
```
#Vig calculation for 76ers vs Lakers
vig_calculation(sevensixers_moneyline, lakers_moneyline)
```

```
The vig is: 3.97%
```

Out[ ]:
```
3.97
```

#3 Question:

If the Jets are trying to decide whether to go for it on fourth down, from the 30-yard line, given the following probabilities, what conversion rate should they need to go for it?

- Win Probability after made field goal: 56%
- Win Probability after missed field goal: 45%
- Win Probability after successful conversion: 60%
- Win Probability after turnover on downs: 46%
- Field Goal Make Probability: 70%

#3 Answer:

The Jets should go for it on fourth down if their conversion probability is higher than 47.86%.

I approached this as an algebra equation after identifying the formula needed, which I found here https://www.the33rdteam.com/win-probability-explained/.

Here is the equation I used based on the data provided, and we can solve for p using the data that's provided.

(p * wp_convert) + ((1-p) * wp_tod) = (fg_make * wp_made_fg) + (fg_miss * wp_missed_fg)

We can solve for p, which is the probability of converting on fourth down. T

- p * .60 + (1-p) * .46 = .70 * .56 + .30 * .45
- p.6 + .46 - .46p = .392 + .135
- p.6 - .46p = .527 - .46
- .14p = 0.067
- p = 0.4786

The following cells illustrate that the equation is correct, and the Jets should go for it on fourth down if their conversion probability is higher than 47.86%.

In [ ]:
```
#Showing the math using python
wp_made_fg = .56
wp_missed_fg = .45
wp_convert = .6
wp_tod = .46
fg_make = .7
fg_miss = .3
go_convert = .4786 #this is our identified p!
go_miss = 1 - go_convert
```

In [ ]:
```
#hooray, they equal one another!
wpa_fg_attempt = (fg_make * wp_made_fg) + (fg_miss * wp_missed_fg)
wpa_go_attempt = (go_convert * wp_convert) + (go_miss * wp_tod)
wpa_fg_attempt, wpa_go_attempt
```

Out[ ]: (0.527, 0.527004)

#4 Question:

What kind of model/statistical tools would you use to project the probability of a kicker making a field goal based only on distance?

#4 Answer:

Making an assumption this is NFL question.

I'd first collect all data possible for FGs attempted with their outcomes in the NFL for as many years as I could scrape or obtain.

The dataframe would include:

- features: distance of FG attempt
- target: true/false if the FG was made

After cleaning and performing light EDA on the data, I'd likely start with using a simple logistic regression model to predict the binary classification outcome of whether the FG was made or not. I'd check for overfitting by using cross-validation and hyperparameter tuning to optimize the model.

Depending on how much time I have, I'd also consider other models like Random Forest or Xgboost to see if they could provide better results.

To evaluate the results, starting with a classification report is an easy way to see how the model is performing. It will provide an output of the model's precision, recall, and f1-score, which will give me insights into which classification errors are being made.

(this feels a lot like the direction I took #1, sorry!)

#5 Question:

For this question, write code in any language. Write a function that takes a string as an input and counts the number of vowels and consonants in that string.

#5 Answer:

Python function below. One future iteration, or additional feature is to identify the duplicates in the string and provide a count for each vowel and consonant.

In [ ]:
```
def vowel_consonant_counter(test_string):
    """
    Count the number of vowels and consonants in a string
    """
    vowels = ['a', 'e', 'i', 'o', 'u']
    consonants = ['b', 'c', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'm', 'n', 'p', 'q', 'r', 's', 't', 'v', 'w',
    vowel_count = 0
    consonant_count = 0
    try:
        if test_string == "":
            return print("Please enter a valid string")
        for letter in test_string:
            if letter in vowels:
                vowel_count += 1
            elif letter in consonants:
                consonant_count += 1
        print(f"There are {vowel_count} vowels and {consonant_count} consonants in {test_string}")
        return (vowel_count, consonant_count)
    except:
        print("Please update to use valid string")
```

In [ ]:
```
vowel_consonant_counter("gobluenationalchampsbaby!")
```

```
There are 9 vowels and 15 consonants in gobluenationalchampsbaby!
```

Out[ ]:
```
(9, 15)
```

#6 Question:

Each Saturday during the college basketball season there can be over 100 games for which we offer live markets. A trading manager comes to us and asks if we can help them decide which of those games we should be staffing with our best traders on any given Saturday. How would you go about building a model to help our trading manager make that decision?

#6 Answer:

I'd start by interviewing the traders to get a sense of what they think makes a game a good candidate for live betting. They're the closest to the action, and I'd want to leverage their expertise but also recognize they might have some inherent biases.

I'd look to leverage data to tease out those biases.

With that, I'd look for data around:

- amount of money wagered on game prior to tip
- amount of money wagered of public vs sharps prior to tip
- amount wagered on different markets (spread, money line, over/under) prior to tip
- historical data on which teams get bet most often
- historical data on which teams receive most "live" betting action
- public interest sentiment gauge (social media, espn coverage, etc)
- time of day
- network coverage
- historical data/context of past live games and how they performed- which were profitable? which were troublesome?
- historical data on which teams have the most live betting action

Upon merging all the data, I'd likely want a way to split it out or indicate whether it's live or pre-game data. I could see it becoming two different models: one for pregame expectations and another for reacting to live game scenarios. One I have bet a lot is when a large favorite is subject to a scoring run by the other team early in the game, especially in basketball.

Once I have the models, I'd review them with the traders and see how they perform in practice. After capturing any new approaches or adjustments, I'd try again for further refinement. Repeat.

#7 Question:

How would you go about making an in-tournament golf win probability model that could update live throughout the weekend?

#7 Answer:

First, I'd try and buy datagolf.com.

After being rejected, I'd look to build a Bayesian model. In addition to including win probabilities prior to start of the tournament, it could also include:

- update player scores throughout the tournament
- adjustments to account for these items (some might be after the round, some would be live):
  - weather forecast and condition updates
  - morning/afternoon tee times
  - course conditions (longer rough, faster greens, etc)
  - hole difficulty changes based on avg score
  - potential pressure situations
  - leaderboard changes (shots off of lead, etc.)

After finishing a hole, simulate the remaining holes to determine new win probabilities based on each player's current score and historical data. Review the model's performance in real time each week and keep improving it.

#8 Question:

How would you go about identifying which row was duplicated the greatest number of times in a data set?

What if you could not load the full data set into memory at one time?

#8 Answer:

Two different directions to start with for this one. Let's assume we're working with a CSV.

If we can load entire data set into memory, then we'll roughly do the following:

- use pandas to load the data
- apply duplicated() method to the dataframe
- use the value_counts() method to identify which row is duplicated the most
- use the idxmax() method to identify the index of the row that is duplicated the most, this will return which row was duplicated the most.

If we cannot load the entire data set into memory, let's chunk it!

- read the data in chunks via pandas and a chunk size of likely 10,000 or more.
- iterate through each chunk and identify which rows are duplicated using roughly the same steps as above
- Once a duplicate row is found, store it in a dictionary with the row as the key and the count as the value.
- utilize the dict and return the max count to see which row was duplicated the most.
- additional research mentioned hashing as a potential solution too.

#9 Question:

A customer has been on a hot streak betting on our sportsbook.

Looking at their betting history, how would you try to identify if they were merely lucky, or good?

#9 Answer:

The first two questions I'd want to ask are:

- 1. How big is the sample size?
- 2. What betting patterns are they displaying?

If the sample size is small, it'll be difficult to determine mathematically whether they are good or lucky. We could try to create a baseline expectation for what a win rate should be based on a 50/50 chance of winning assumption and calculate how many more bets they'd need to make to "come back to reality" if they are lucky. If they don't, then they might be good.

Regardless of the sample size, we can review their betting history to see if they are showcasing any tendencies. Some data points to consider might be: - type of bets - teams being bet - time bet is placed - closing line value of bet - amount being bet, are they wagering an amount to receive a whole number? i.e. betting 143.50 to win 100. - derivative bets based on a single game (i.e. betting the under and then betting unders for player props) - noticing if any other sharps are betting the same way/same time

From here we'd at least be able to get a sense of their approach which would help us determine whether they are lucky or good.

#10 Question:

Given a home team is expected to score 5 runs and an away team is expected to score 4.5 runs, which outcome is the most likely?

- a) 10 total runs
- b) 9 total runs
- c) They are equally likely

Please explain your reasoning.

#10 Answer:

- b) 9 total runs

A baseball game does not end in a tie*.

If both teams had 5 runs, the game would go into extra innings, and the game total would be greater than 10 runs.

*Unless the game is called due to weather and the makeup date is scrapped because it won't impact playoff standings. In this case, neither team is awarded a win or loss, but the stats will count.