

Swish Analytics

Goal: using the provided MLB data, review the dataset and outline the way I would go about the model-building process with the goal of predicting the probability that the next thrown pitch is a certain type.

Read in Data

```
In [38]: # import
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

from typing import Tuple, List
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import r2_score, mean_absolute_error

pd.set_option('display.max_rows', 20)
pd.set_option('display.max_columns', None)
sns.set_style("whitegrid")
%matplotlib inline

seed = 42

In [39]: # read in the data
data = pd.read_csv("pitches_folder/pitches", sep=";", low_memory=False)
```

Data Exploration & Preprocessing

Let's take a look at the data and see what we're working with.

```
In [40]: data.shape

Out[40]: (718961, 125)
```

It looks like a significant portion of columns are completely null and won't be usable for this analysis.

I'll stick with data where `pitch_type` is not null, since that is the value to be predicted, and remove columns that are entirely or mostly empty.

```
In [41]: # check for null values
total = data.isnull().sum().sort_values(ascending=False)
pct = (data.isnull().sum()/data.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, pct], axis=1, keys=['Total', 'Percent'])
missing_data

Out[41]:
```

	Total	Percent
runner4_event	718961	1.0
runner4_score	718961	1.0
runner4_rbi	718961	1.0
runner4_earned	718961	1.0
runners_id	718961	1.0
...
stand	0	0.0
created_at	0	0.0
added_at	0	0.0
modified_at	0	0.0
modified_by	0	0.0

125 rows × 2 columns

I need to trim down the data into what is actually useful. So removing columns that are entirely empty, rows where the target feature is empty, columns that aren't relevant to predicting the pitch type, etc.

I also need to do some feature engineering and convert categorical data into numerical data for EDA purposes.

The features will be further trimmed down during the modeling stage, where I can discover which features are most impactful for predictive purposes.

```
In [42]: # get columns where the data is entirely or mostly missing and drop them
data = data.drop((missing_data[missing_data["Percent"] > 0.5]).index, axis=1)

# confirm they're gone
data.isnull().sum().max()

# drop rows where 'pitch_type' is null
data = data[data['pitch_type'].isnull()]

# drop columns that aren't relevant to predicting pitch type or that are redundant or I don't have time to explore further at the moment
cols = [
    'uid',
    'game_pk',
    'year',
    'date',
    'team_id_b',
    'team_id_p',
    'start_tfs',
    'start_tfs_zulu',
    'batter_id',
    'pitcher_id',
    'at_bat_des',
    'event',
    'away_team_runs',
    'home_team_runs',
    'pitch_des',
    'pitch_id',
    'pitch_tfs',
    'pitch_tfs_zulu',
    'created_at',
    'added_at',
    'modified_at',
    'modified_by',
    'sv_id'
]

# feature engineering & refactoring

# let's get the total runs at the time of the pitch, by either team
data['total_runs'] = data['away_team_runs'] + data['home_team_runs']

# convert binary columns into numerical
data['stand'] = pd.factorize(data['stand'])[0]
data['p_throws'] = pd.factorize(data['p_throws'])[0]
data['type'] = pd.factorize(data['type'])[0]

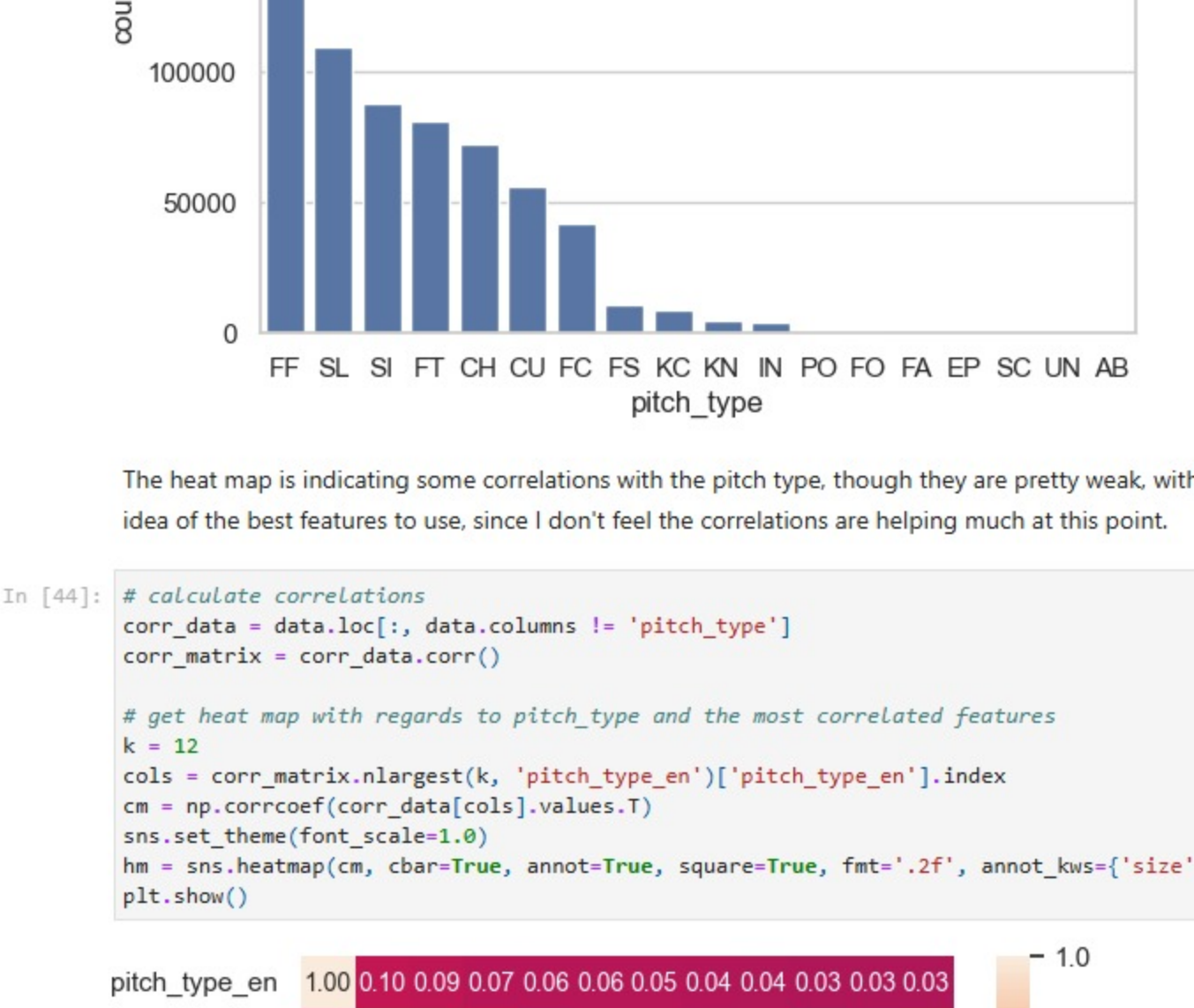
# convert pitch type into numerical
le = LabelEncoder()
data['pitch_type_en'] = le.fit_transform(data['pitch_type'])

# convert batter's height into inches
data['b_height'] = (data.b.height.str.split("-").str[0].astype(int) * 12) + (data.b.height.str.split("-").str[1].astype(int))

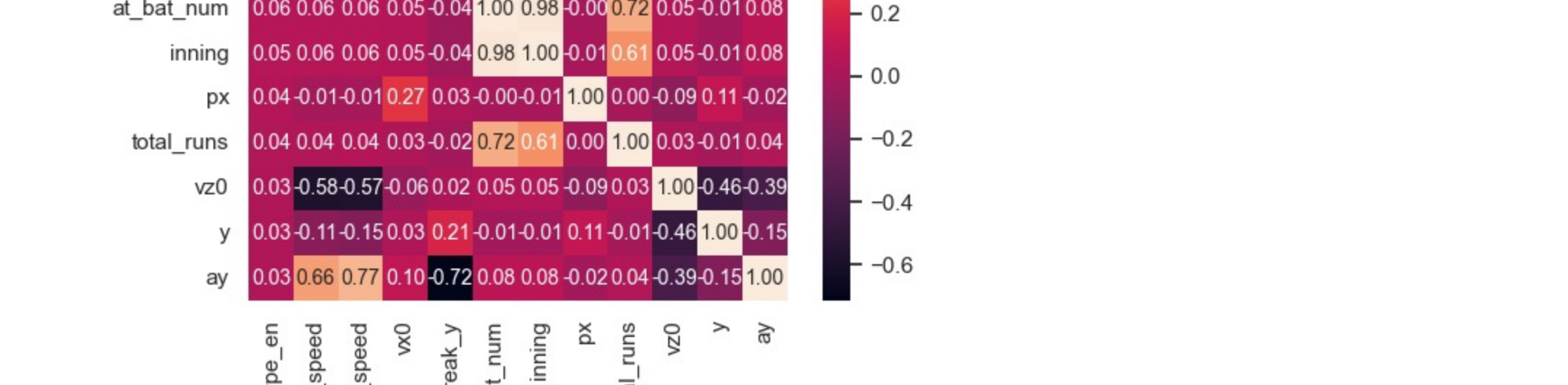
data = data.drop(data[cols], axis=1)
```

I want a quick glimpse into what the pitch types and feature correlations look like.

For the types of pitches, the data is heavily skewed towards FF, then SL, SI, FT, CH, CU, FC to a lesser extent.



The heat map is indicating some correlations with the pitch type, though they are pretty weak, with all most correlated being 10% or less. I will likely end up doing some feature testing & variable selection during the modeling process to get a better idea of the best features to use, since I don't feel the correlations are helping much at this point.



Train Test Split

I want to build a baseline model with the data I have then iteratively improve on the model by removing features that aren't impactful. In general, it's better to have a simpler model that doesn't introduce more randomness to the model by accident than what is there naturally.

First step is to split the data into training and testing sets.

```
In [45]: # get the cleaned up data
df = data.loc[:, data.columns != 'pitch_type']

In [46]: df.shape

Out[46]: (716681, 46)
```

```
In [47]: # train test split
def pd_train_test_split(df: pd.DataFrame,
                        ratio: float) -> Tuple[pd.DataFrame, pd.DataFrame]:
    """
    Given an input df containing all data and the target variable, split the data into training and testing sets

    Params
    -----
    df : pd.DataFrame
        df of all data, features and actuals across all dates.

    Returns
    -----
    Tuple[pd.DataFrame, pd.DataFrame]
        The training set and the test set, respectively
    """
    total_rows = df.shape[0]
    train_size = int(total_rows*ratio)

    # shuffle the data
    df=df.sample(frac=1)

    # split data into test and train sets
    df_train = df[0:train_size]
    df_test = df[train_size:]

    return df_train, df_test

In [48]: # get the 80/20 training and testing data
df_train, df_test = pd_train_test_split(df=df, ratio=0.8)

In [49]: df_train.sample(2)

Out[49]:
```

	inning	top	at_bat_num	pcount_at_bat	pcount_pitcher	balls	strikes	fouls	outs	is_final_pitch	final_balls	final_strikes	final_outs	stand	b_height	p_throws	type	x	y	start_speed	end_speed	sz_top	sz_bot	pf_x	pf
693004	7	1	54	1	1	0	0	0	1	0	0	1	2	1	72	1	0	121.89	135.56	85.4	79.2	3.51	1.75	8.29	5
635720	3	0	24	2	42	0	1	1	1	0	1	1	3	0	74	1	1	60.09	117.43	88.0	81.4	3.41	1.56	1.42	10

Baseline Model

Since I want to predict the probability of the type of pitch that will occur, a random forest model is a good choice.

```
In [50]: def train_randomforest_model(df_train: pd.DataFrame,
                                     y_name: str,
                                     X_names: List[str]) -> RandomForestClassifier:
    """
    Train a catboost model to predict df[y_name] using df[X_names]

    Params
    -----
    df_train : pd.DataFrame
        Training dataset of features and actuals
    y_name : str
        Name of column containing the outcome variable
    X_names : List[str]
        List of name(s) of column(s) containing the features

    Returns
    -----
    trained random forest model
    """
    # Extract outcome variable and features from df_train
    y_train = df_train[y_name]
    X_train = df_train[X_names]

    # Instantiate model
    model = RandomForestClassifier(n_estimators=100, random_state=42)

    # Train model
    model.fit(X_train, y_train)

    return model

In [51]: # get all feature column names and the target variable
X_names = [i for i in df_train.columns if i not in ["pitch_type", "pitch_type_en"]]
y_name = "pitch_type_en"

In [52]: # train a baseline model
model = train_randomforest_model(df_train = df_train, y_name = y_name, X_names = X_names)
```

Evaluation

Next up is to evaluate how the model performed and I will use R2 and MAE for this.

```
In [53]: def evaluate_accuracy(model: RandomForestClassifier,
                              df_test: pd.DataFrame,
                              y_name: str,
                              X_names: List[str]) -> Tuple[float, float, np.array]:
    """
    Given a trained model and a test set, evaluate model accuracy.
    First we generate predictions using df_test[X_names]. Then we compare
    the predictions to df_test[y_name] to calculate R^2 and MAE.

    Params
    -----
    model : RandomForestClassifier
        Trained RandomForest model
    df_test : pd.DataFrame
        Test dataset containing features and actuals
    y_name : str
        Name of column containing the outcome variable
    X_names : List[str]
        List of name(s) of column(s) containing the features

    Returns
    -----
    Tuple[float, float]
        The R^2 and MAE metrics, respectively
    np.array
        The raw predictions
    """
    # Generate predictions
    predictions = model.predict(df_test[X_names])
    probabilities = np.max(model.predict_proba(df_test[X_names]), axis = 1)

    # Compute metrics
    r2 = r2_score(df_test[y_name], predictions)
    mae = mean_absolute_error(df_test[y_name], predictions)

    return r2, mae, predictions, probabilities

In [54]: r2, mae, predictions, probabilities = evaluate_accuracy(model = model, df_test = df_test, y_name = y_name, X_names = X_names)

In [55]: print(f'R^2: {r2}, MAE: {mae}')

R^2: 0.5841049716087654, MAE: 1.1137040608034688

The moderate R2 suggests this model explains a good amount of the variation in the target variable and the low MAE suggests that the model predictions aren't usually far off the actual pitch type on average.

Remember, this is just the baseline model with no additional adjustments made to it so there is a lot of room for improvement!

For ease of viewing, let's throw the predictions and their probabilities back into the test data to see how it all compares.

In [56]: # add the predictions and probabilities back into the test data
df_test['prediction'] = predictions
df_test['probability'] = probabilities

# undo the label encoding from earlier to better see the predictions and actuals
df_test['pitch_type_actual'] = le.inverse_transform(df_test['pitch_type_en'])

In [59]: output = df_test[['pitch_type_actual', 'pitch_type_en', 'prediction', 'probability']]

In [60]: output.head(15)

Out[60]:
```

	pitch_type_actual	pitch_type_en	prediction	probability
314167	FF	6	6	0.95
48348	SI	15	15	0.97
501176	SI	15	15	1.00
717469	FT	9	9	0.84
505658	SI	15	15	0.98
611851	FF	6	6	0.97
194831	FC	5	6	0.92
625977	FF	6	6	0.95
507864	FF	6	6	0.98
32658	FC	5	6	0.76
393217	FT	9	6	0.52
552369	CU	2	2	0.73
151849	FT	9	9	0.75
333973	CU	2	2	0.94
671878	FC	5	5	0.56

Conclusion

With more time to work on this project, I would want to be more thorough EDA. I could probably save myself some pain down the line if my EDA was more thorough so I could potentially highlight important features before getting to the modeling stage.

I would want to test some engineered features to see how they impact the model, such as the result of the previous pitch thrown by this pitcher (strikeout, hit, etc) and the type pitch most recently thrown by the pitcher (perhaps some pitchers stick to one or two types and others try to mix it up every time they throw). I would also like to explore how the pitcher height vs the batter height may impact the type of pitch thrown.

Most importantly, I would want to do many more iterations of the modeling to ensure overfitting isn't a huge issue and better evaluation metrics. In general, I would want to spend more time on the variable selection process to really nail down the best features to use in the model and to keep it simple - no redundant features if I can help it. I would test several different methods to do this - perhaps Stepwise Regression or LASSO to eliminate unneeded features or Elastic Net to weight features according to how useful they are.

I normally would do a baseline model and then continue refining the model by digging into feature importance and iteratively testing the model with different features until I find the set that works best for predictive purposes without overfitting, using a library like SHAP. This would give me an idea of both the feature importance and the directionality of the features. I would also want to include more metrics for evaluation - a confusion matrix and AUC, probably.

For now, this is what I could do in the time allotted.