Introduction:

See how stats like PA, AB, H, 2B, 3B, HR, RBI, SB, BA, OBP, SLG, etc influence postseason birth

Methods:

Gathering MLB regular season team stats from 2012-2019, including stats like PA, AB, H, 2B, 3B, HR, RBI, SB, BA, OBP, SLG, etc.

Using SQL Server and Python/Spyder)

Building classification model

Try to build perfect model by comparing different learning rate, Model validation, early stopping, experiment different number of nodes in each layer, different number of layers

import regular season stats from MLB teams who got into postseason during 2012-2019

items include Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP

total rows are 8(years)*10(teams each year)=80

In [15]:
```
import pandas as pd
import pyodbc


sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                             SERVER=ALLENKHOSQLSERVER02;
                             DATABASE=Playoffbound;
                             Trusted_Connection=yes;''')

query = '''
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['13SB'] 
where Tm in ('WSH','LAD','MIL','ATL','STL','HOU','NYY','MIN','TBR','OAK')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['13SB'] 
where Tm in ('BOS','LAD','CDL','WSH','CHC','HOU','NYY','CLE','ARI','MIN')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['17SB'] 
where Tm in ('BOS','LAD','CDL','WSH','CHC','HOU','NYY','CLE','ARI','MIN')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['18SB'] 
where Tm in ('TOR','CLE','BOS','BAL','TEX','NYM','CHC','LAD','WSH','SFG')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['19SB'] 
where Tm in ('TOR','MIN','HOU','NYY','TEX','NYM','CHC','LAD','STL','PIT')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['14SB'] 
where Tm in ('BAL','KCR','OAK','LAA','DET','WSN','STL','LAD','PIT','SFG')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['15SB'] 
where Tm in ('BOS','TBR','OAK','CLE','DET','ATL','STL','LAD','PIT','CIN')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['16SB'] 
where Tm in ('TEX','BAL','OAK','NYY','DET','ATL','STL','SFG','WSN','CIN','LDHy')

df = pd.read_sql(query, sql_conn)

#stored as df_post
df_post = df
```

import regular season stats from MLB teams who DIDN'T get into postseason during 2012-2019 items are the same as above total rows are 8(years)*20(teams each year)=160

In [16]:
```
sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                             SERVER=ALLENKHOSQLSERVER02;
                             DATABASE=Playoffbound;
                             Trusted_Connection=yes;''')

query = '''
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['19SB'] 
where Tm is not null and Tm in ('WSH','LAD','MIL','ATL','STL','HOU','NYY','MIN','TBR','OAK','LDHy')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['18SB'] 
where Tm is not null and Tm in ('BOS','LAD','MIL','ATL','CHC','HOU','NYY','CLE','CDL','OAK','LDHy')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['17SB'] 
where Tm is not null and Tm not in ('BOS','LAD','CDL','WSH','CHC','HOU','NYY','CLE','ARI','MIN','LDHy')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['16SB'] 
where Tm is not null and Tm not in ('TOR','CLE','BOS','BAL','TEX','NYM','CHC','LAD','WSN','SFG','LDHy')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['15SB'] 
where Tm is not null and Tm not in ('TOR','MIN','HOU','NYY','TEX','NYM','CHC','LAD','STL','PIT','SFG','LDHy')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['14SB'] 
where Tm is not null and Tm not in ('BAL','KCR','OAK','LAA','DET','WSN','STL','LAD','PIT','SFG','LDHy')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['13SB'] 
where Tm is not null and Tm not in ('BOS','TBR','OAK','CLE','DET','ATL','STL','LAD','PIT','CIN','LDHy')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['12SB'] 
where Tm is not null and Tm not in ('TEX','BAL','OAK','NYY','DET','ATL','STL','SFG','WSN','CIN','LDHy')

df = pd.read_sql(query, sql_conn)

#stored as df_npost
df_npost = df

#add each dataframe a new column named POST, which imply whether the team made the postseason or not
df_post['POST']= 1
df_npost['POST']= 0

#append two dataframes together
df_com=df_post.append(df_npost)
```

Perform deep learning(neural network) classification model

In [17]:
```
# Import necessary modules
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical

# Save the number of columns in predictors: n_cols
predictors=df_com.loc[:,'BatAge':'GDP'].to_numpy()
n_cols = predictors.shape[1]

# Convert the target to categorical: target
target = to_categorical(df_com['POST'])

# Set up the model
model = Sequential()

# Add the first and second layer
model.add(Dense(100, activation='relu', input_shape = (n_cols,)))
model.add(Dense(100, activation='relu'))

# Add the output layer
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
model.fit(predictors, target, batch_size=100, epochs=5)
```

```
Epoch 1/5
240/240 [==============================] - 0s 1ms/step - loss: 41469017.4993 - accuracy: 0.4833
Epoch 2/5
240/240 [==============================] - 0s 62us/step - loss: 159069590209114.8125 - accuracy: 0.5333
Epoch 3/5
240/240 [==============================] - 0s 67us/step - loss: 0.6878 - accuracy: 0.6667
Epoch 4/5
240/240 [==============================] - 0s 62us/step - loss: 0.6864 - accuracy: 0.6667
Epoch 5/5
240/240 [==============================] - 0s 62us/step - loss: 0.6850 - accuracy: 0.6667
```

Out[17]:
```
<keras.callbacks.callbacks.History at 0x19604662748>
```

compare different learning rate

In [18]:
```
# Import the SGD optimizer
from keras.optimizers import SGD
input_shape = (n_cols,)

def get_new_model(input_shape = input_shape):
    model = Sequential()
    model.add(Dense(100, activation='relu', input_shape = input_shape))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    return(model)

# Create list of learning rates: lr_to_test
lr_to_test = [.000001, 0.01, 1]

# Loop over learning rates
for lr in lr_to_test:
    print('\n\nTesting model with learning rate: %f\n'%lr )
    # Build new model to test, unaffected by previous models
    model = get_new_model()
    # Create SGD optimizer with specified learning rate: my_optimizer
    my_optimizer = SGD(lr=lr)
    # Compile the model
    model.compile(optimizer=my_optimizer, loss='categorical_crossentropy')
    # Fit the model
    model.fit(predictors, target, batch_size=100, epochs=5)
```

```
Testing model with learning rate: 0.000001

Epoch 1/5
240/240 [==============================] - 0s 2ms/step - loss: 137.0936
Epoch 2/5
240/240 [==============================] - 0s 71us/step - loss: 23.7052
Epoch 3/5
240/240 [==============================] - 0s 58us/step - loss: 28.2293
Epoch 4/5
240/240 [==============================] - 0s 75us/step - loss: 23.0275
Epoch 5/5
240/240 [==============================] - 0s 67us/step - loss: 18.4650


Testing model with learning rate: 0.010000

Epoch 1/5
240/240 [==============================] - 0s 2ms/step - loss: 53138982.8577
Epoch 2/5
240/240 [==============================] - 0s 54us/step - loss: 0.8997
Epoch 3/5
240/240 [==============================] - 0s 46us/step - loss: 0.6890
Epoch 4/5
240/240 [==============================] - 0s 54us/step - loss: 0.6875
Epoch 5/5
240/240 [==============================] - 0s 62us/step - loss: 0.6861


Testing model with learning rate: 1.000000

Epoch 1/5
240/240 [==============================] - 0s 2ms/step - loss: 519568115629 71.8828
Epoch 2/5
240/240 [==============================] - 0s 54us/step - loss: 0.7278
Epoch 3/5
240/240 [==============================] - 0s 58us/step - loss: 0.6536
Epoch 4/5
240/240 [==============================] - 0s 58us/step - loss: 0.6390
Epoch 5/5
240/240 [==============================] - 0s 58us/step - loss: 0.6366
```

Then validate the data

In [19]:
```
# Save the number of columns in predictors: n_cols
predictors = df_com.iloc[:,1:20].to_numpy()
input_shape = (n_cols,)
# Specify the model
model = Sequential()
model.add(Dense(100, activation='relu', input_shape = input_shape))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Fit the model
hist = model.fit(predictors, target, validation_split=0.3, batch_size=100, epochs=5)
```

```
Train on 168 samples, validate on 72 samples
Epoch 1/5
168/168 [==============================] - 0s 3ms/step - loss: 144.9681 - accuracy: 0.5119 - val_loss: 41.4199 - val_accuracy: 0.0556
Epoch 2/5
168/168 [==============================] - 0s 113us/step - loss: 44.6229 - accuracy: 0.5179 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/5
168/168 [==============================] - 0s 119us/step - loss: 43.8831 - accuracy: 0.5833 - val_loss: 77.6278 - val_accuracy: 0.0000e+00
Epoch 4/5
168/168 [==============================] - 0s 101us/step - loss: 61.5720 - accuracy: 0.4762 - val_loss: 2.9088 - val_accuracy: 0.7500
Epoch 5/5
168/168 [==============================] - 0s 95us/step - loss: 32.9106 - accuracy: 0.5298 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

Try early stopping

In [20]:
```
# Import EarlyStopping
from keras.callbacks import EarlyStopping

# Save the number of columns in predictors: n_cols
predictors = df_com.iloc[:,1:20].to_numpy()
input_shape = (n_cols,)

# Specify the model
model = Sequential()
model.add(Dense(100, activation='relu', input_shape = input_shape))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define early_stopping_monitor
early_stopping_monitor = EarlyStopping(patience=2)

# Fit the model
model.fit(predictors, target, epochs=30, validation_split=0.3, callbacks=[early_stopping_monitor])
```

```
Train on 168 samples, validate on 72 samples
Epoch 1/30
168/168 [==============================] - 1s 3ms/step - loss: 118.7131 - accuracy: 0.5298 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 2/30
168/168 [==============================] - 0s 254us/step - loss: 133.7558 - accuracy: 0.4524 - val_loss: 174.9606 - val_accuracy: 0.0000e+00
Epoch 3/30
168/168 [==============================] - 0s 208us/step - loss: 89.7120 - accuracy: 0.4643 - val_loss: 0.0659 - val_accuracy: 0.9861
```

Out[20]:
```
<keras.callbacks.callbacks.History at 0x196102fec18>
```

experiment different number of neurons in each layer

In [21]:
```
import matplotlib.pyplot as plt

# Set up the model_1
model_1 = Sequential()

# Add the first and second layer
model_1.add(Dense(50, activation='relu', input_shape=input_shape))
model_1.add(Dense(50, activation='relu'))

# Add the output layer
model_1.add(Dense(2, activation='softmax'))

# Compile the model
model_1.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Create the new model: model_2
model_2 = Sequential()

# Add the first and second layers
model_2.add(Dense(150, activation='relu', input_shape=input_shape))
model_2.add(Dense(50, activation='relu'))

# Add the output layer
model_2.add(Dense(2, activation='softmax'))

# Compile model_2
model_2.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Fit model_1
model_1_training = model_1.fit(predictors, target, epochs=15, batch_size=100, validation_split=0.2, callbacks=[early_stopping_monitor], verbose=False)

# Fit model_2
model_2_training = model_2.fit(predictors, target, epochs=15, batch_size=100, validation_split=0.2, callbacks=[early_stopping_monitor], verbose=False)

# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r', model_2_training.history['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```

experiment different number of layers

In [22]:
```
# Create the new model: model_1
model_1 = Sequential()

# Add one hidden layer
model_1.add(Dense(54, activation='relu', input_shape=input_shape))

# Add the output layer
model_1.add(Dense(2, activation='softmax'))

# Compile model_1
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Create the new model: model_2
model_2 = Sequential()

# Add the first, second, and third hidden layers
model_2.add(Dense(54, activation='relu', input_shape=input_shape))
model_2.add(Dense(54, activation='relu'))
model_2.add(Dense(54, activation='relu'))

# Add the output layer
model_2.add(Dense(2, activation='softmax'))

# Compile model_2
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fit model_1
model_1_training = model_1.fit(predictors, target, epochs=20, batch_size=100, validation_split=0.4, callbacks=[early_stopping_monitor], verbose=False)

# Fit model_2
model_2_training = model_2.fit(predictors, target, epochs=20, batch_size=100, validation_split=0.4, callbacks=[early_stopping_monitor], verbose=False)

# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r', model_2_training.history['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```