Cleaning and Preprocessing of NFL Play-by-Play Data

This jupyter notebook walks through the cleaning and preprocessing of an NFL play-by-play data set that can be found here (https://www.kaggle.com/datasets/toddsteussie/nfl-play-statistics-dataset-2004-to-present).

This data set contains various data points for every play completed in an NFL game from the years 2004-2019. The overall goal of this project is to use the data set to predict the success of calling a "run" or "pass" play at any given time in the game. There is a separate notebook where I implement various machine learning models to make this prediction. The purpose of this notebook is to walk through how I cleaned this data set for use by these machine learning models.

Below is a complete list of everything done in this notebook (in the same order that the cells are executed). There are also small annotations throughout this notebook mentioning what is being done at each step.

1. Define a "success" function that tells us whether each play was a success or not, based on post-play data (more details on this metric below).
2. Read in the data and toss out any data points where "no play" is 1 in the data set. This indicates that there was a pre-snap penalty or something similar that happened and thus, no play was ever run.
3. Take only rushing and passing plays since we are not interested in special teams plays
4. One-hot encode the "Huddle" section of the data. There are a significant amount of Nans in this column of the data. For these plays, we set "Huddle" to 0.5 (halfway between "huddle" and "no huddle").
5. Convert team ID numbers to team names for easier interpretation
6. Replace "gameID" with "HomeTeamID" and "VisitorTeamID" so that the algorithm has access to who is the home team and who is the away team.
7. Create a new column called "HomeTeamPossession" that tracks whether or not the home team has possession of the ball.
8. One-hot encode "HomeTeamID", "AwayTeamID", and "PlayType".
9. Create new column called "Score_diff" that tracks the difference in score between the team on offense and the team of defense

10. Add a categorical column called "success" that tracks whether the play was a success or not. This is calculated according to the rules laid out below.

11. Select only the pre-play information to give to the prediction algorithm. This ensures that our algorithm doesn't have access to data like "yards gained" because we want the algorithm to predict things based only on pre-play data.

12. Convert GameClock variable to float data type

13. Save the cleaned data set for use by the models

Rules for success

Below are a description of the rules used by the add_success function to evaluate whether each play was a success or not. I chose these rules based on what I feel should be considered a successful football play. The rules are different depending on what down it is. In the data set, successes are represented by 1, failures by -1, and neutral plays by 0.

Any down:

- if you get a first down or score, that's a success

First down:

- success = gaining (3/10) or greater of yards to the sticks
- neutral = between 1 yard and (3/10) of yards to the sticks
- fail = 0 yards or negative yards

Second down:

- success = gaining (1/2) or greater of yards to the sticks
- neutral = between 1 yard and (1/2) of yards to the sticks
- fail = 0 or negative yards

Third down:

- success = first down
- neutral = nothing
- fail = not first down

Fourth down:

- success = first down
- neutral = nothing
- fail = not first down

In [1]:
```
cd archive\ (3)/
```

```
/Users/Michaelray/Documents/NFL_project/archive (3)
```

In [2]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
```

In [3]:
```
#Define function to calculate success metric and add it as a column to a DataFr
def add_success(df):
    '''Adds a success row to the DataFrame df,
    which will be used as the target variable
    in our models. The target variable is 1
    for a success, 0 for neutral, and -1 for
    failure (definitions above)

    Parameters
    ----------
    df - Give a DataFrame with all the data we
        have above

    Returns
    -------
    new_df - a new DataFrame with a success target
        variable added which can be accessed with
        new_df['success']'''

    down = df['down']
    distance = df['distance']
    yds_gained = df['netYards']

    #Implement the rules set at top of document
    success = pd.DataFrame(columns=['success'])
    df = df.join(success)

    #Always a success if we got a first down
    df.loc[df['firstDown'] == 1, 'success'] = 1

    #First down success metric
    df.loc[(df['firstDown']!=1) & (down==1) & ((3/10)*distance<=yds_gained), 's
    df.loc[(df['firstDown']!=1) & (down==1) & (0<yds_gained) & (yds_gained<(3/1
    df.loc[(df['firstDown']!=1) & (down==1) & (yds_gained<=0), 'success'] = -1

    #Second down success metrics
    df.loc[(df['firstDown']!=1) & (down==2) & ((1/2)*distance<=yds_gained), 'su
    df.loc[(df['firstDown']!=1) & (down==2) & (0<yds_gained) & (yds_gained<=(1/
    df.loc[(df['firstDown']!=1) & (down==2) & (yds_gained<=0), 'success'] = -1

    #Third and fourth down success metrics only need one line because we alread
    #caught some of these when we checked for first down
    df.loc[(df['firstDown']!=1) & (down==3), 'success'] = -1
    df.loc[(df['firstDown']!=1) & (down==4), 'success'] = -1

    return df
```

In [4]:
```
df = pd.read_csv('plays.csv')
games = pd.read_csv('games.csv')

# all plays up to index 41,709 do not have a corresponding entry in games.csv s
df = df.iloc[41709:,:]
```

In [5]:
```
#Get rid of any column that has noplay set to 1.
```

```
played_df = df[df.noPlay == 0]

#Include only rushing and passing plays
special_teams = ['kickoff', 'xp', 'spike', 'field goal']
regular_plays = ['rush', 'pass']
rp_data = played_df.loc[(played_df['playType'] == 'rush') | (played_df['playTyp
```

In [6]:
```
#Turn huddle into numerical variable

huddle_df = pd.DataFrame(np.empty(rp_data.shape[0]), columns=['huddleNew'])
rp_data = rp_data.join(huddle_df)

rp_data.loc[rp_data['huddle'] == 'huddle', 'huddleNew'] = 0
rp_data.loc[rp_data['huddle'] == 'no huddle', 'huddleNew'] = 1
rp_data.loc[rp_data['huddle'].isna(), 'huddleNew'] = 0.5

rp_data = rp_data.drop('huddle', axis=1)
rp_data = rp_data.rename(columns={"huddleNew" : 'huddle'})
```

In [7]:
```
#Turn TeamId's into team names
team_dict = {'Cardinals':3800, 'Ravens':325 , 'Falcons':200 , 'Bills':610 , 'Pa
             'Bengals':920, 'Bears':810, 'Browns':1050, 'Cowboys':1200, 'Bronco
             'Lions':1540, 'Texans':2120, 'Packers':1800, 'Colts':2200, 'Rams':
             'Jaguars':2250, 'Vikings':3000, 'Chiefs':2310, 'Saints':3300, 'Rai
             'Giants':3410, 'Chargers':4400, 'Eagles':3700, 'Dolphins':2700, 'M
             'Patriots':3200, 'Seahawks':4600, 'Jets':3430, 'Buccaneers':4900,
             'Washington':5110, 'Titans':2100}
hometeam_dict = {'Cardinals home':3800, 'Ravens home':325 , 'Falcons home':200
             'Bengals home':920, 'Bears home':810, 'Browns home':1050, 'Cowboys
             'Lions home':1540, 'Texans home':2120, 'Packers home':1800, 'Colts
             'Jaguars home':2250, 'Vikings home':3000, 'Chiefs home':2310, 'Sai
             'Giants home':3410, 'Chargers home':4400, 'Eagles home':3700, 'Dol
             'Patriots home':3200, 'Seahawks home':4600, 'Jets home':3430, 'Buc
             'Washington home':5110, 'Titans home':2100}
awayteam_dict = {'Cardinals away':3800, 'Ravens away':325 , 'Falcons away':200
             'Bengals away':920, 'Bears away':810, 'Browns away':1050, 'Cowboys
             'Lions away':1540, 'Texans away':2120, 'Packers away':1800, 'Colts
             'Jaguars away':2250, 'Vikings away':3000, 'Chiefs away':2310, 'Sai
             'Giants away':3410, 'Chargers away':4400, 'Eagles away':3700, 'Dol
             'Patriots away':3200, 'Seahawks away':4600, 'Jets away':3430, 'Buc
             'Washington away':5110, 'Titans away':2100}


team_dict = {v: k for k, v in team_dict.items()}
hometeam_dict = {v: k for k, v in hometeam_dict.items()}
awayteam_dict = {v: k for k, v in awayteam_dict.items()}
```

In [8]:
```
#Go from gameId in rp_data to homeTeamId and awayTeamId
game_info = games[['gameId', 'homeTeamId', 'visitorTeamId']]
game_list = pd.DataFrame(rp_data['gameId'])
homeTeamId = pd.DataFrame(columns=['homeTeamId'])
awayTeamId = pd.DataFrame(columns=['awayTeamId'])
rp_data = rp_data.join(homeTeamId)
rp_data = rp_data.join(awayTeamId)

for game_index in np.arange(game_info['gameId'].shape[0]):
    game = game_info.loc[game_index, 'gameId']
```

```
    rp_data.loc[rp_data['gameId'] == game, 'homeTeamId'] = game_info.loc[game_i
    rp_data.loc[rp_data['gameId'] == game, 'awayTeamId'] = game_info.loc[game_i
```

In [9]:
```
#Now create homeTeamPos column that contains 1 if home team has possession and
#We don't need an awayTeamPos column because this is redundant information

homeTeamPos = pd.DataFrame(np.zeros(rp_data.shape[0]), index=rp_data.index, col
rp_data.join(homeTeamPos)

rp_data.loc[rp_data['homeTeamId'] == rp_data['possessionTeamId'], 'homeTeamPoss
rp_data.loc[rp_data['awayTeamId'] == rp_data['possessionTeamId'], 'homeTeamPoss
```

In [10]:
```
#Now go from team Id numbers to team names for homeTeamId, awayTeamId
rp_data['homeTeamId'] = rp_data['homeTeamId'].map(hometeam_dict)
rp_data['awayTeamId'] = rp_data['awayTeamId'].map(awayteam_dict)
```

In [11]:
```
#One-hot encode homeTeamId, awayTeamId, playType
hometeamid_dum = pd.get_dummies(rp_data['homeTeamId'])
awayteamid_dum = pd.get_dummies(rp_data['awayTeamId'])
pt_dum = pd.get_dummies(rp_data['playType'])

rp_data = pd.concat([rp_data, hometeamid_dum], axis=1).drop('homeTeamId', axis=
rp_data = pd.concat([rp_data, awayteamid_dum], axis=1).drop('awayTeamId', axis=
rp_data = pd.concat([rp_data, pt_dum], axis=1).drop('playType', axis=1)
```

In [12]:
```
#Take the scores of home and visiting teams along with homeTeamPossession to ge
#between possessing team and visiting team (convention is possessing score - no

score_diff = pd.DataFrame(np.empty(rp_data.shape[0]), columns=['scoreDiff'])
rp_data = rp_data.join(score_diff)

rp_data.loc[rp_data['homeTeamPossession'] == 1.0, 'scoreDiff'] = rp_data.loc[rp
rp_data.loc[rp_data['homeTeamPossession'] == 0.0, 'scoreDiff'] = rp_data.loc[rp
```

In [13]:
```
#Add success column
rp_data = add_success(rp_data)
```

In [14]:
```
#Over half of the data points for formations are null, so just drop it
#Select only the pre-play information
desired_cols = ['playId', 'gameId', 'playSequence', 'quarter', 'playNumberByTea
                'gameClock', 'down', 'distance', 'distanceToGoalPre', 'evPre',
               'homeScorePre', 'visitingScorePre', 'huddle',

               'Bears home', 'Bengals home', 'Bills home',
               'Broncos home', 'Browns home', 'Buccaneers home', 'Cardinals home
               'Chiefs home', 'Colts home', 'Cowboys home', 'Dolphins home', 'Ea
               'Falcons home', 'Giants home', 'Jaguars home', 'Jets home', 'Lion
               'Niners home', 'Packers home', 'Panthers home', 'Patriots home',
               'Rams home', 'Ravens home', 'Saints home', 'Seahawks home', 'Stee
               'Titans home', 'Vikings home', 'Washington home', 'Bears away', '
               'Broncos away', 'Browns away', 'Buccaneers away', 'Cardinals away
               'Chiefs away', 'Colts away', 'Cowboys away', 'Dolphins away', 'Ea
               'Giants away', 'Jaguars away', 'Jets away', 'Lions away', 'Niners
               'Panthers away', 'Patriots away', 'Raiders away', 'Rams away', 'P
               'Seahawks away', 'Steelers away', 'Texans away', 'Titans away', '


                'homeTeamPossession', 'pass', 'rush', 'success']
```

```
rp_data = rp_data[desired_cols]
```

In [15]:
```
rp_data = rp_data[desired_cols]
```

In [16]:
```
#Convert gameClock to number of seconds
clock_series = rp_data['gameClock'].str.split(pat='(').str[1]
clock_series = clock_series.str.split(pat=')').str[0]
minutes = clock_series.str.split(pat=':').str[0].astype(int)
seconds = clock_series.str.split(pat=':').str[1].astype(int)
gameClockSec = pd.DataFrame({'gameClockSec':minutes*60+seconds})
rp_data = rp_data.join(gameClockSec)
rp_data = rp_data.drop(columns=['gameClock'])
rp_data = rp_data.rename(columns={"gameClockSec" : 'gameClock'})
```

In [17]:
```
rp_data.isna().sum().sum()
```

Out[17]:
```
0
```

We've got all our desired data with no Nans, so let's save the data to a csv file

In [18]:
```
#We've got effectively 18 separate data points now. We'll save the model data t
rp_data.to_csv('model_data.csv')
```