NBA Modeling Homework

Answer the Following Questions in the Jupyter Notebook

If you have any questions, reach out to bobby@prizepicks.com

You can print your results or just write the SQL query if you're having issues with the notebook. If you're not familiar with Postgres you are more than welcome to write in your prefered syntax

Patrick Hayes - 7/27/2023

Postgres Instructions

assume you had a table named nba_pts_table with athlete_id, game_date, pts_scored. write a query that selects all the colums plus:

- *average points scored for all players*
- *average points scored for a player in their last 3 games*

Let's go!

In [ ]:

```
#I tested this out on a separate postgres database and it worked for my starting pitchers table.

sql_command = """
WITH PointsScoredAvg AS (
  SELECT
    athlete_id,
    AVG(pts_scored) AS avg_pts_scored
  FROM
    nba_pts_table
  GROUP BY
    athlete_id
),
Last3Games AS (
  SELECT
    athlete_id,
    "game_date",
    pts_scored,
    ROW_NUMBER() OVER (PARTITION BY athlete_id ORDER BY "game_date" DESC) AS game_order
  FROM
    nba_pts_table
)
SELECT
  nba_pts_table.*,
  psa.avg_pts_scored,
  l3.avg_pts_scored_last_3
FROM
  nba_pts_table
JOIN
  PointsScoredAvg psa ON nba_pts_table.athlete_id = psa.athlete_id
LEFT JOIN (
  SELECT
    athlete_id,
    AVG(pts_scored) AS avg_pts_scored_last_3
  FROM
    Last3Games
  WHERE
    game_order <= 3
  GROUP BY
    athlete_id
) l3 ON nba_pts_table.athlete_id = l3.athlete_id;

##################
"""
```

Model Instructions

make a simple model using the nba_test.csv dataset to predict the field target_pts. fields denoted with _3 are rolling averages over the last 3 games per player. fields denoted with _szn are season long averages per player

I am stressing simplicity in building this model. Some of the existing features will need to be engineered for the linear regression model I'll be using.

If this is TOO simple, please let me know, and I will gladly increase the complexity of what I displayed.

In [ ]:
```python
## IMPORTS ##

import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import GridSearchCV, cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

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.

With more time, I would have cleaned up and turned the import process into functions.

In [ ]:
```python
# Pull in the data, identify three features that need to be adjusted for our model

nba_train_df = pd.read_csv('nba_train.csv')
nba_test_df = pd.read_csv('nba_test.csv')

nba_train_df.head()
```

Out[ ]:

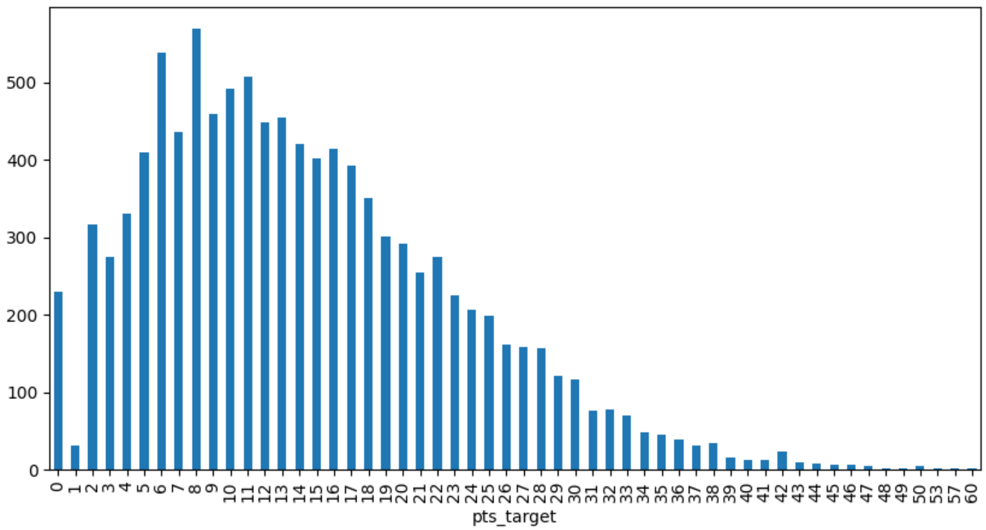
	athlete_id	pts_target	game_date	starter	points_l3	points_szn	fgm_l3	fgm_szn	fga_l3	fga_szn	fg3m_l3	fg3m_szn	fg3a_l3	fg3a_szn	ftm_l3	ftm_szn	fta_l3	fta_szn	min_l3	m
0	572	3	11/30/2021	False	6.000000	7.866667	1.666667	2.600000	4.333333	5.400000	1.666667	1.600000	2.000000	3.133333	1.000000	1.066667	2.000000	1.666667	15.333333	20
1	345	11	11/30/2021	True	18.333333	17.476190	6.333333	6.095238	14.333333	14.904762	2.000000	2.047619	5.666667	6.666667	3.666667	3.238095	7.000000	4.714286	36.000000	35
2	136	2	11/30/2021	False	6.333333	7.578947	2.666667	2.868421	6.333333	6.815789	1.000000	1.000000	3.000000	3.342105	0.000000	0.842105	0.000000	1.315789	22.000000	25
3	3	5	11/30/2021	False	3.333333	6.714286	1.000000	2.000000	2.000000	3.428571	1.000000	1.000000	2.000000	2.428571	0.333333	1.714286	0.666667	2.285714	21.333333	21
4	16	4	11/30/2021	False	11.000000	11.200000	3.666667	4.000000	7.333333	7.800000	0.333333	0.200000	0.666667	0.800000	3.333333	3.000000	3.666667	3.400000	21.333333	21.

In [ ]:
```python
#healthy training set
nba_train_df.shape
```

Out[ ]:
```
(10467, 20)
```

In [ ]:
```python
#How does the distribution of the target variable look? A few outliers that I'd account for witih more time

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

Out[ ]:
```
<Axes: xlabel='pts_target'>
```



In [ ]:

```
#check for missing values, none!

#thank you for the clean data.

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

```
athlete_id     0
pts_target     0
game_date      0
starter        0
points_l3      0
points_szn     0
fgm_l3         0
fgm_szn        0
fga_l3         0
fga_szn        0
fg3m_l3        0
fg3m_szn       0
fg3a_l3        0
fg3a_szn       0
ftm_l3         0
ftm_szn        0
fta_l3         0
fta_szn        0
min_l3         0
mins_szn       0
dtype: int64
athlete_id     0
pts_target     0
game_date      0
starter        0
points_l3      0
points_szn     0
fgm_l3         0
fgm_szn        0
fga_l3         0
fga_szn        0
fg3m_l3        0
fg3m_szn       0
fg3a_l3        0
fg3a_szn       0
ftm_l3         0
ftm_szn        0
fta_l3         0
fta_szn        0
min_l3         0
mins_szn       0
dtype: int64
```

Out[ ]: (None, None)

In [ ]:
```
#knowing we need to engineer a few features for our model, quick review of the data types of each
nba_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10467 entries, 0 to 10466
Data columns (total 20 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   athlete_id  10467 non-null  int64
 1   pts_target  10467 non-null  int64
 2   game_date   10467 non-null  object
 3   starter     10467 non-null  bool
 4   points_l3   10467 non-null  float64
 5   points_szn  10467 non-null  float64
 6   fgm_l3      10467 non-null  float64
 7   fgm_szn     10467 non-null  float64
 8   fga_l3      10467 non-null  float64
 9   fga_szn     10467 non-null  float64
 10  fg3m_l3     10467 non-null  float64
 11  fg3m_szn    10467 non-null  float64
 12  fg3a_l3     10467 non-null  float64
 13  fg3a_szn    10467 non-null  float64
 14  ftm_l3      10467 non-null  float64
 15  ftm_szn     10467 non-null  float64
 16  fta_l3      10467 non-null  float64
 17  fta_szn     10467 non-null  float64
 18  min_l3      10467 non-null  float64
 19  mins_szn    10467 non-null  float64
dtypes: bool(1), float64(16), int64(2), object(1)
memory usage: 1.5+ MB
```

In [ ]:

```python
#quick cleanups
#I made the decision to remove game date knowing that I'd want to incorproate it with more time, focused on simplicity

def clean_df(df):
    df = df.drop(columns=['athlete_id', 'game_date'], axis=1)
    df['starter'] = df['starter'].map({False: 0, True: 1})
    return df

nba_train_df_clean = clean_df(nba_train_df)
nba_test_df_clean = clean_df(nba_test_df)
```

In [ ]:

```python
#confirm that the clean up worked
nba_train_df_clean.head()
```

Out[ ]:

	pts_target	starter	points_l3	points_szn	fgm_l3	fgm_szn	fga_l3	fga_szn	fg3m_l3	fg3m_szn	fg3a_l3	fg3a_szn	ftm_l3	ftm_szn	fta_l3	fta_szn	min_l3	mins_szn
0	3	0	6.000000	7.866667	1.666667	2.600000	4.333333	5.400000	1.666667	1.600000	2.000000	3.133333	1.000000	1.066667	2.000000	1.666667	15.333333	20.133333
1	11	1	18.333333	17.476190	6.333333	6.095238	14.333333	14.904762	2.000000	2.047619	5.666667	6.666667	3.666667	3.238095	7.000000	4.714286	36.000000	35.428571
2	2	0	6.333333	7.578947	2.666667	2.868421	6.333333	6.815789	1.000000	1.000000	3.000000	3.342105	0.000000	0.842105	0.000000	1.315789	22.000000	25.578947
3	5	0	3.333333	6.714286	1.000000	2.000000	2.000000	3.428571	1.000000	1.000000	2.000000	2.428571	0.333333	1.714286	0.666667	2.285714	21.333333	21.571429
4	4	0	11.000000	11.200000	3.666667	4.000000	7.333333	7.800000	0.333333	0.200000	0.666667	0.800000	3.333333	3.000000	3.666667	3.400000	21.333333	21.800000

Model Fitting

Sticking with the instructions and theme of simple, linear regression will be my baseline model. I really want to incorporate a Zoolander joke, but I'll save that for another time :)

In [ ]:

```python
## CODE TO FIT MODEL HERE ##

#splitting the dataset into X train and y train

X_train = nba_train_df_clean.drop(columns=['pts_target'], axis=1)
y_train = nba_train_df_clean['pts_target']

#trying a linear regression model first
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
###########################
```

Out[ ]:

```
▾ LinearRegression
LinearRegression()
```

In [ ]:

```
# Dabbled with using this model but kept coming back to simplicity and not gold plating the assignment.
# I am craving many adjustments to make this prediction more efficient, but I'll refrain for now
# and add them at the end for the discussion.


# rf = RandomForestRegressor(random_state=42)
# rf.fit(X_train, y_train)
```

Model Predicting

using the model - predict on the nba_train.csv

Straight-forward predicting. Using the Linear Regression model I fit on the training data, I'll predict on the test data.

In [ ]:

```
## CODE TO PREDICT HERE ##

# not the most encouraging results but that means lots of room for improvement

#sometimes I would get an error on the r2 score, I'd investigate with more time

X_test = nba_test_df_clean.drop(columns=['pts_target'], axis=1)
y_test = nba_test_df_clean['pts_target']


y_pred = lr.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2_score = r2_score(y_test, y_pred)
print('MAE: %.4f' % mae)
print('R2: %.4f' % r2_score)
print('RMSE: %.4f' % rmse)

#########################
```

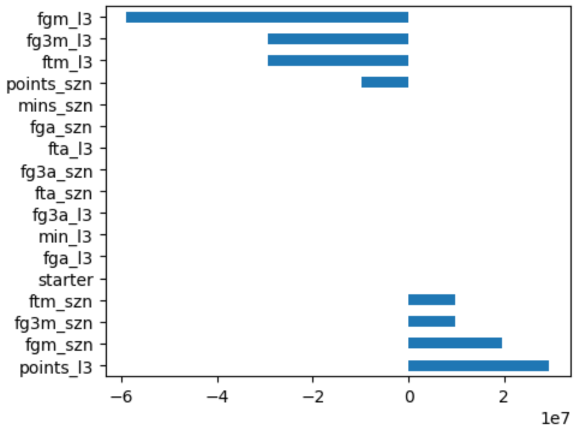
```
MAE: 4.6696
R2: 0.4870
RMSE: 5.9911
```

Model Performance

Model is working, but what can I learn from it?

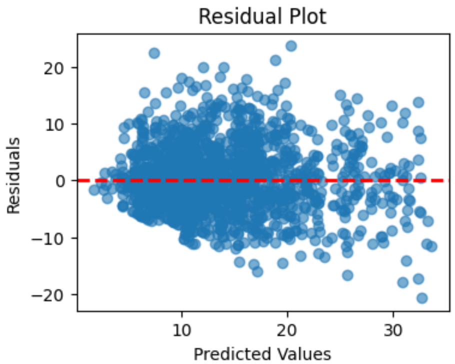
In [ ]:

```
# I like reviewing how the model is establishing the coefficients for each feature.
# Takeaways here can lead to interesting conversations with other folks to help improve or discuss the model.
# last 3 not as important as season stats, interesting!

coefficients = pd.Series(lr.coef_, index=X_train.columns)
sorted_coefficients = coefficients.sort_values(ascending=True)
sorted_coefficients.plot(kind='barh', figsize=(5,4)).invert_yaxis()
plt.show()
```



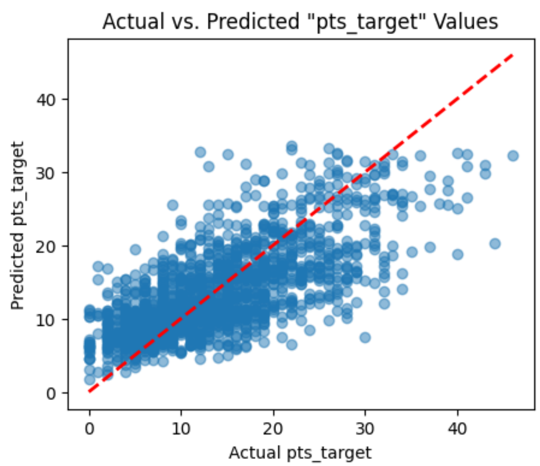
In [ ]:

```
#Residual plot to see how the model is performing
#Meh, outliers galore.

residuals = y_test - y_pred

plt.figure(figsize=(4, 3))
plt.scatter(y_pred, residuals, alpha=0.6)
plt.axhline(y=0, color='red', linestyle='--', linewidth=2)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```

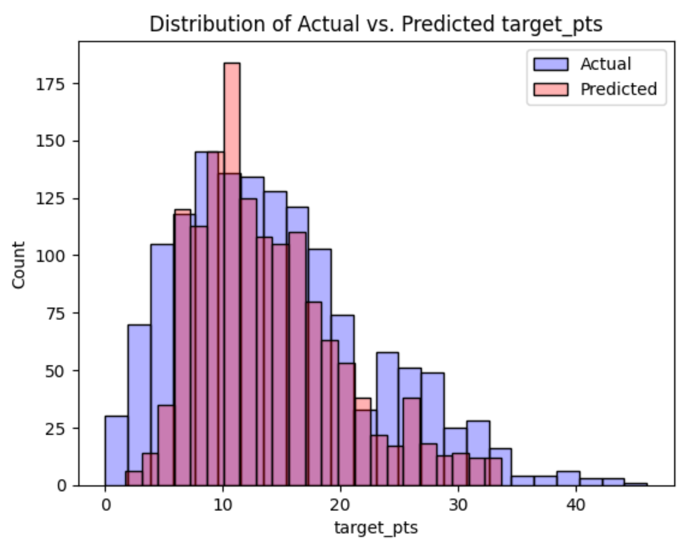


In [ ]:

```
#Scatter to see actual vs predicted. Model does not do well with high numbers of points scored.

plt.figure(figsize=(5, 4))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--', color='red', linewidth=2)
plt.xlabel('Actual pts_target')
plt.ylabel('Predicted pts_target')
plt.title('Actual vs. Predicted "pts_target" Values')
plt.show()
```



In [ ]:
```
#quick review of the distribution of actual vs predicted

sns.histplot(y_test, color='blue', label='Actual', kde=False,alpha=0.3)
sns.histplot(y_pred, color='red', label='Predicted', kde=False, alpha=0.3)

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



Model Analysis

write a short analysis of the models performance

It leaves a lot to be desired. The model captures less than 50% (48.7%) of the variance in the data, yielding projected point targets to be off by nearly 5 points on average (the Mean Absolute Error is ~4.7). The model distribution skews a little right, but not enough to match the actual data distribution.

It ultimately serves as a baseline to build upon, but I will leave it as is, even though I know I'd be able to improve it drastically by allowing myself to not focus on "simple."

Lastly, Mentioned briefly above, I thought it was quite interesting that the last three games' averages were not as impactful as the season averages for this particular model.

Again, if a more complex model was expected, I would happily make revisions to showcase!

Reflection

what would you do different if you had more time? what is missing from this model?

With more time I would:

- Turn the import process into a function and many other steps within the notebook.
- Add more features such as additional time periods of averages, date of game, location of the game (home vs away), etc.
- Ridge vs lasso for linear regresion regulization
- Scaling features and testing performance with and without
- More models! I'd implement a function that would allow me to quickly test different models and compare their performance. Part of this would include gridsearch to identify the best parameters and best score of a given model. A few models I'd start with are: Random Forest, Gradient Boosting, and XGBoost.
- More analysis of the performance. Separating out subsets of data based on playing time, position, etc., to see if there are any patterns that can be exploited.
- combine the two CSVs into one dataframe to make testing models and their performance easier. A train/test split would be used to separate out the training and test data instead of using the two CSVs.

Talk soon!

- Patrick

end homework