# NBA_data_analysis

March 18, 2021

[1]:

```
# DATA ANALYSIS
```

[3]:

```
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from itertools import chain
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import Circle, Rectangle, Arc
from pyspark.mllib.evaluation import BinaryClassificationMetrics
import numpy as np
import json

from pyspark.sql import SparkSession
from pyspark.mllib.classification import LogisticRegressionWithLBFGS,␣
 ↪LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint
from pyspark.ml.feature import VectorAssembler
from pyspark.mllib.linalg import Vectors
from pyspark.sql.functions import col
from pyspark.mllib.evaluation import MulticlassMetrics


import os
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.classification import LinearSVC
from pyspark.ml.classification import LinearSVCTrainingSummary
from pyspark.ml.classification import RandomForestClassifier as RF
from pyspark.mllib.evaluation import BinaryClassificationMetrics
```

[4]:

```
spark = (
    SparkSession.builder.master("local")
    .appName("review_and_category_analytics")
    .config("spark.executor.memory", "8g")
    .config("spark.executor.cores", "4")
    .config("spark.cores.max", "4")
    .config("spark.driver.memory", "8g")
    .getOrCreate()
)

sc = spark.sparkContext
```

[5]:

```
# response = shotchartdetail.ShotChartDetail(
#     context_measure_simple = "FGA",
#         team_id=0,
#         player_id=0,
#         season_nullable='2018-19',
#         season_type_all_star='Regular Season')
# content = json.loads(response.get_json())
```

[6]:

```
team_ref = {
    "LAL": "Los Angeles Lakers",
    "LAC": "Los Angeles Clippers",
    "DEN": "Denver Nuggets",
    "UTA": "Utah Jazz",
    "OKC": "Oklahoma City Thunder",
    "HOU": "Houston Rockets",
    "DAL": "Dallas Mavericks",
    "MEM": "Memphis Grizzlies",
    "POR": "Portland Trailblazers",
    "NOP": "New Orleans Pelicans",
    "SAC": "Sacramento Kings",
    "SAS": "San Antonio Spurs",
    "PHO": "Phoenix Suns",
    "MIL": "Milwaukee Bucks",
    "TOR": "Toronto Raptors",
    "BOS": "Boston Celtics",
    "MIA": "Miami Heat",
    "IND": "Indiana Pacers",
    "PHI": "Philadelphia 76ers",
    "BRK": "Brooklyn Nets",
    "ORL": "Orlando Magic",
    "WAS": "Washington Wizards",
}
```

## 0.1 Data Import and Preprocessing

[7]:
```
df2 = spark.read.format("csv").options(header="true").load("135.csv")
```

[8]:
```
df2.show(3)
```

```
+-----------------+----------+-------------+---------+----------------+---------
-+------------------+------+-----------------+-----------------+-----------+----
--------------+--------------+-----------------+--------------------+-----------
----+-------------+-----+-----+-------------------+--------------+---------+---+
---+
|        GRID_TYPE|   GAME_ID|GAME_EVENT_ID|PLAYER_ID|     PLAYER_NAME|
TEAM_ID|         TEAM_NAME|PERIOD|MINUTES_REMAINING|SECONDS_REMAINING|
EVENT_TYPE|       ACTION_TYPE|     SHOT_TYPE|  SHOT_ZONE_BASIC|      SHOT_ZONE_A
REA|SHOT_ZONE_RANGE|SHOT_DISTANCE|LOC_X|LOC_Y|SHOT_ATTEMPTED_FLAG|SHOT_MADE_FLAG
|GAME_DATE|HTM|VTM|
+-----------------+----------+-------------+---------+----------------+---------
-+------------------+------+-----------------+-----------------+-----------+----
--------------+--------------+-----------------+--------------------+-----------
----+-------------+-----+-----+-------------------+--------------+---------+---+
---+
|Shot Chart Detail|0021800001|            7|   203496|Robert
Covington|1610612755|Philadelphia 76ers|     1|               11|
40|Missed Shot|         Jump Shot|3PT Field Goal|Above the Break 3|
Center(C)|        24+ ft.|           26|  -53|  264|                  1|
0| 20181016|BOS|PHI|
|Shot Chart Detail|0021800001|           10|  1628369|    Jayson
Tatum|1610612738|    Boston Celtics|     1|               11|
15|Missed Shot|         Jump Shot|3PT Field Goal|Above the Break 3|Left Side
Center(LC)|        24+ ft.|           25| -148|  207|                  1|
0| 20181016|BOS|PHI|
|Shot Chart Detail|0021800001|           14|  1627759|    Jaylen
Brown|1610612738|    Boston Celtics|     1|               11|
3|Missed Shot|Running Layup Shot|2PT Field Goal|  Restricted Area|
Center(C)|Less Than 8 ft.|            1|    4|   18|                  1|
0| 20181016|BOS|PHI|
+-----------------+----------+-------------+---------+----------------+---------
-+------------------+------+-----------------+-----------------+-----------+----
--------------+--------------+-----------------+--------------------+-----------
----+-------------+-----+-----+-------------------+--------------+---------+---+
---+
only showing top 3 rows
```

[53]:
```
mapping = create_map([lit(x) for x in chain(*team_ref.items())])
df3 = df2.withColumn("HOME", mapping[df2["HTM"]])
```

[10]:

```
df3 = df3.withColumn("DISTANCE_TRUE", sqrt(df3.LOC_X ** 2 + df3.LOC_Y ** 2) /␣
 ↪10)
```

[11]:

```
df3 = df3.withColumn("HOME", when(df3["TEAM_NAME"] == df3["HOME"], 1).
 ↪otherwise(0))
df3.show(5)
```

```
+-----------------+----------+-------------+---------+----------------+---------
-+------------------+------+-----------------+-----------------+-----------+----
--------------+--------------+--------------------+--------------------+--------
-------+-------------+-----+-----+-------------------+--------------+---------+-
--+---+----+------------------+
|        GRID_TYPE|   GAME_ID|GAME_EVENT_ID|PLAYER_ID|     PLAYER_NAME|
TEAM_ID|         TEAM_NAME|PERIOD|MINUTES_REMAINING|SECONDS_REMAINING|
EVENT_TYPE|       ACTION_TYPE|     SHOT_TYPE|     SHOT_ZONE_BASIC|      SHOT_ZON
E_AREA|SHOT_ZONE_RANGE|SHOT_DISTANCE|LOC_X|LOC_Y|SHOT_ATTEMPTED_FLAG|SHOT_MADE_F
LAG|GAME_DATE|HTM|VTM|HOME|     DISTANCE_TRUE|
+-----------------+----------+-------------+---------+----------------+---------
-+------------------+------+-----------------+-----------------+-----------+----
--------------+--------------+--------------------+--------------------+--------
-------+-------------+-----+-----+-------------------+--------------+---------+-
--+---+----+------------------+
|Shot Chart Detail|0021800001|            7|   203496|Robert
Covington|1610612755|Philadelphia 76ers|     1|               11|
40|Missed Shot|         Jump Shot|3PT Field Goal|   Above the Break 3|
Center(C)|        24+ ft.|           26|  -53|  264|                  1|
0| 20181016|BOS|PHI|   0| 26.92675249635574|
|Shot Chart Detail|0021800001|           10|  1628369|    Jayson
Tatum|1610612738|    Boston Celtics|     1|               11|
15|Missed Shot|         Jump Shot|3PT Field Goal|   Above the Break 3|Left Side
Center(LC)|        24+ ft.|           25| -148|  207|                  1|
0| 20181016|BOS|PHI|   1|25.446610776290033|
|Shot Chart Detail|0021800001|           14|  1627759|    Jaylen
Brown|1610612738|    Boston Celtics|     1|               11|
3|Missed Shot|Running Layup Shot|2PT Field Goal|     Restricted Area|
Center(C)|Less Than 8 ft.|            1|    4|   18|                  1|
0| 20181016|BOS|PHI|   1|1.8439088914585775|
|Shot Chart Detail|0021800001|           17|   203954|     Joel
Embiid|1610612755|Philadelphia 76ers|     1|               10|               55|
Made Shot|Running Layup Shot|2PT Field Goal|     Restricted Area|
Center(C)|Less Than 8 ft.|            0|   -8|    3|                  1|
1| 20181016|BOS|PHI|   0| 0.854400374531753|
|Shot Chart Detail|0021800001|           19|  1628369|    Jayson
Tatum|1610612738|    Boston Celtics|     1|               10|
36|Missed Shot|Driving Layup Shot|2PT Field Goal|In The Paint (Non…|
Center(C)|Less Than 8 ft.|            4|  -46|    1|                  1|
0| 20181016|BOS|PHI|   1| 4.601086828130937|
+-----------------+----------+-------------+---------+----------------+---------
-+------------------+------+-----------------+-----------------+-----------+----
--------------+--------------+--------------------+--------------------+--------
-------+-------------+-----+-----+-------------------+--------------+---------+-
--+---+----+------------------+
only showing top 5 rows
```

[12]:

```
avg_shot_pct = (
    df3.groupBy("ACTION_TYPE")
    .agg({"SHOT_MADE_FLAG": "mean"})
    .withColumnRenamed("avg(SHOT_MADE_FLAG)", "SHOT_PERC")
)

avg_shot_pct.show(3)
```

```
+--------------------+------------------+
|         ACTION_TYPE|         SHOT_PERC|
+--------------------+------------------+
|  Putback Layup Shot|0.6832706766917294|
|Finger Roll Layup…|0.6833333333333333|
|Running Reverse D…|0.9285714285714286|
+--------------------+------------------+
only showing top 3 rows
```

[13]:

```
df3 = df3.join(avg_shot_pct, "ACTION_TYPE")
```

[14]:

```
df3.show(2)
```

```
+-----------+-----------------+----------+-------------+---------+--------------
--+----------+------------------+------+-----------------+-----------------+----
-------+--------------+-----------------+--------------------+---------------+--
-----------+-----+-----+-------------------+--------------+---------+---+---+---
-+------------------+-------------------+
|ACTION_TYPE|        GRID_TYPE|   GAME_ID|GAME_EVENT_ID|PLAYER_ID|
PLAYER_NAME|   TEAM_ID|
TEAM_NAME|PERIOD|MINUTES_REMAINING|SECONDS_REMAINING| EVENT_TYPE|     SHOT_TYPE|
SHOT_ZONE_BASIC|      SHOT_ZONE_AREA|SHOT_ZONE_RANGE|SHOT_DISTANCE|LOC_X|LOC_Y|S
HOT_ATTEMPTED_FLAG|SHOT_MADE_FLAG|GAME_DATE|HTM|VTM|HOME|     DISTANCE_TRUE|
SHOT_PERC|
+-----------+-----------------+----------+-------------+---------+--------------
--+----------+------------------+------+-----------------+-----------------+----
-------+--------------+-----------------+--------------------+---------------+--
-----------+-----+-----+-------------------+--------------+---------+---+---+---
-+------------------+-------------------+
|  Jump Shot|Shot Chart Detail|0021800001|            7|   203496|Robert
Covington|1610612755|Philadelphia 76ers|     1|               11|
40|Missed Shot|3PT Field Goal|Above the Break 3|           Center(C)|        24+
ft.|             26|  -53|  264|                  1|              0|
20181016|BOS|PHI|   0| 26.92675249635574|0.34801086648555857|
|  Jump Shot|Shot Chart Detail|0021800001|           10|  1628369|    Jayson
Tatum|1610612738|    Boston Celtics|     1|                11|
15|Missed Shot|3PT Field Goal|Above the Break 3|Left Side Center(LC)|        24+
ft.|             25| -148|  207|                  1|              0|
20181016|BOS|PHI|   1|25.446610776290033|0.34801086648555857|
+------------+-----------------+----------+-------------+---------+--------------
--+----------+-----------------+------+-----------------+-----------------+----
-------+--------------+-----------------+--------------------+---------------+--
-----------+-----+-----+--------------------+--------------+---------+---+---+---
-+------------------+-------------------+
only showing top 2 rows
```

[15]:

```python
df3 = df3.withColumn("TIME", df3.MINUTES_REMAINING * 60 + df3.SECONDS_REMAINING)
```

[16]:

```python
df4 = df3.select(
    df3.SHOT_MADE_FLAG.cast("numeric"), df3.PERIOD.cast("numeric"), df3.TIME.
 ↪cast("double"), df3.HOME.cast("numeric"), df3.DISTANCE_TRUE.cast("double"),␣
 ↪df3.SHOT_PERC.cast("double")
).withColumnRenamed("SHOT_MADE_FLAG", "label")
```

[17]:

```python
df4.show(5)
```

```
+-----+------+-----+----+------------------+-------------------+
|label|PERIOD| TIME|HOME|     DISTANCE_TRUE|          SHOT_PERC|
+-----+------+-----+----+------------------+-------------------+
|    0|     1|700.0|   0| 26.92675249635574|0.34801086648555857|
|    0|     1|675.0|   1|25.446610776290033|0.34801086648555857|
|    0|     1|663.0|   1|1.8439088914585775|  0.645466847090663|
|    1|     1|655.0|   0| 0.854400374531753|  0.645466847090663|
|    0|     1|636.0|   1| 4.601086828130937| 0.4799899636181157|
+-----+------+-----+----+------------------+-------------------+
only showing top 5 rows
```

[18]:

```python
df4 = df4.dropna()
```

[19]:

```python
df4.count()
```

[19]:

```
170560
```

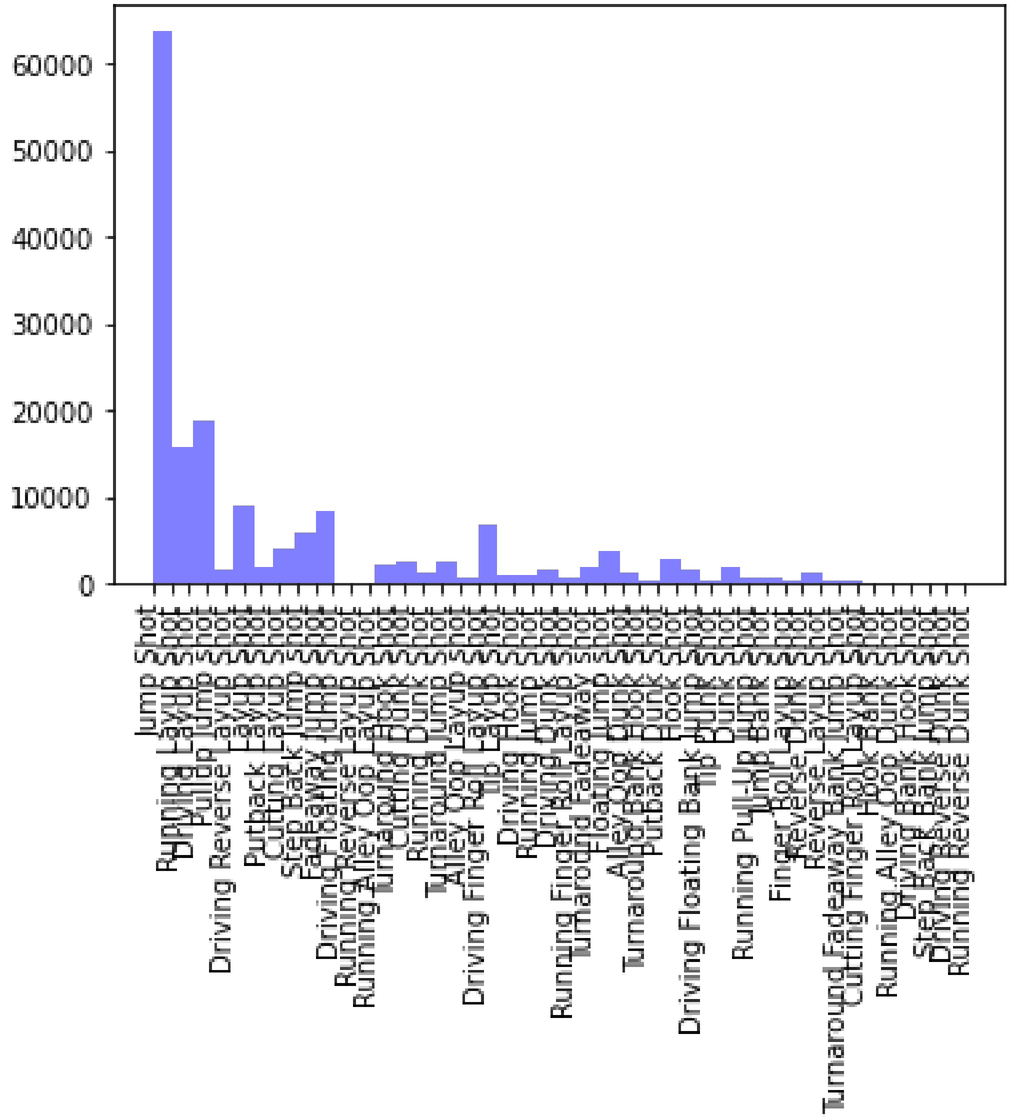
## 0.2 Exploratory Data Analysis

[20]:

```
## Histogram of Action_type % - Justin
# Shot Chart - Adhitya
# Avg # of shots made if your home or away - Justin
# Same with Period - Adhitya
```

[21]:

```
x = pd.read_csv('135.csv')
num_bins = 40
n, bins, patches = plt.hist(x['ACTION_TYPE'], num_bins, facecolor='blue',␣
 ↪alpha=0.5)
plt.xticks(rotation = 90)
plt.show()
```



[22]:

```
HOME_PERC = df4.groupBy('HOME').agg({"label": "mean"}).
 ↪withColumnRenamed("avg(label)", "HOME_PERC")
```

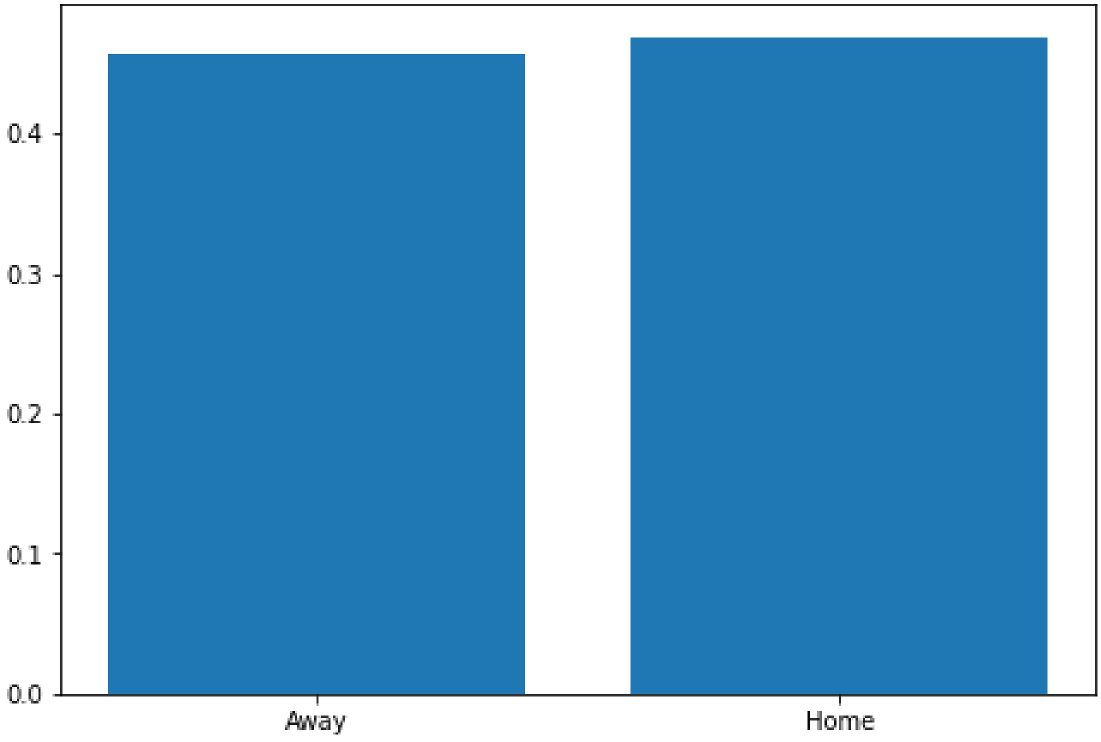
[23]:

```
HOME_PERC.show()

x = HOME_PERC.toPandas()
```

```
+----+---------+
|HOME|HOME_PERC|
+----+---------+
|   0|   0.4566|
|   1|   0.4694|
+----+---------+
```

[24]:

```
#https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/bar_stacked.
 ↪html#sphx-glr-gallery-lines-bars-and-markers-bar-stacked-py
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(x.HOME,x.HOME_PERC)
xlab = ['HOME','AWAY']
# ax.set_xticklabels(xlab)
plt.xticks(np.arange(2), ('Away', 'Home'))
plt.show()
```



[24]:

```
```

[25]:

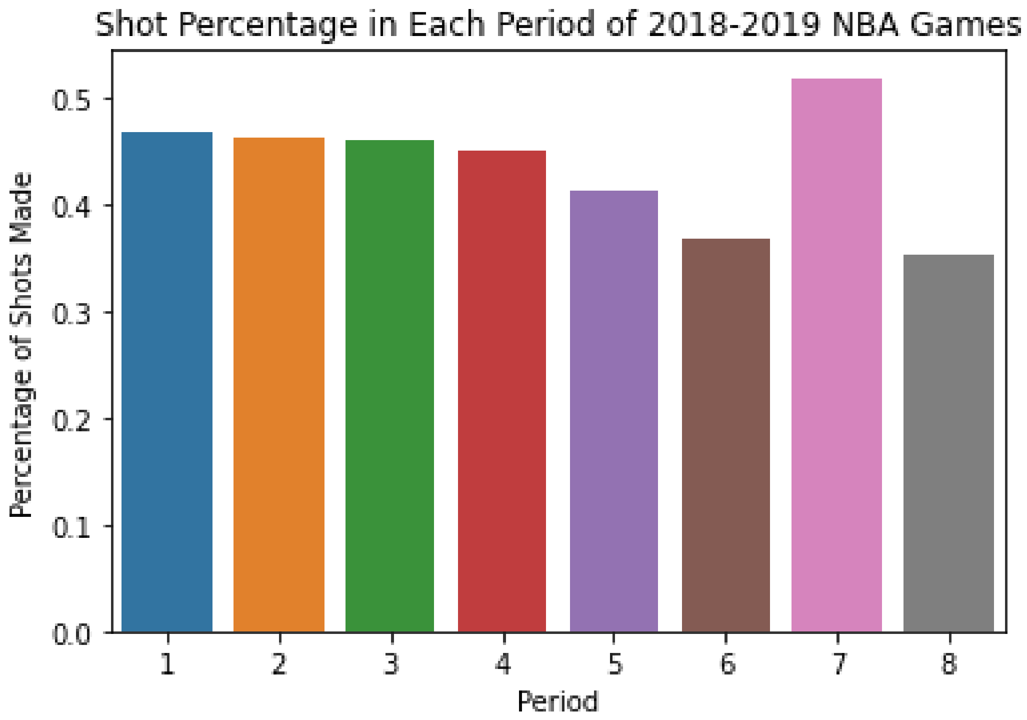
```
# Insert columns needed for plotting
graph_columns = df3.select(
    "SHOT_MADE_FLAG",
    "PERIOD",
    "TIME",
    "HOME",
    "DISTANCE_TRUE",
    "SHOT_PERC",
    "LOC_X",
    "LOC_Y",
)
```

[26]:

```
shot_chart_df = graph_columns.select(
    graph_columns.LOC_X.cast("double"),
    graph_columns.LOC_Y.cast("double"),
    graph_columns.SHOT_MADE_FLAG.cast("double"),
    graph_columns.PERIOD,
)  # , graph_columns.PERIOD.cast("string"))
shot_chart = shot_chart_df.select("*").toPandas()
```

[27]:

```
period_df = (
    shot_chart_df.groupBy("PERIOD")
    .agg({"SHOT_MADE_FLAG": "mean"})
    .withColumnRenamed("avg(SHOT_MADE_FLAG)", "SHOT_PERC")
)
period_plot = period_df.select("*").toPandas()
period_plot = period_plot.sort_values("PERIOD")
g = sns.barplot(x=period_plot["PERIOD"], y=period_plot["SHOT_PERC"])
g.set(
    xlabel="Period",
    ylabel="Percentage of Shots Made",
    title="Shot Percentage in Each Period of 2018-2019 NBA Games",
)
plt.show()
```



[28]:

```
# Source for code to draw basketball court: http://savvastjortjoglou.com/
↪nba-shot-sharts.html


def draw_court(ax=None, color="black", lw=2, outer_lines=False):
    # If an axes object isn't provided to plot onto, just get current one
    if ax is None:
        ax = plt.gca()

    # Create the various parts of an NBA basketball court

    # Create the basketball hoop
    # Diameter of a hoop is 18" so it has a radius of 9", which is a value
    # 7.5 in our coordinate system
    hoop = Circle((0, 0), radius=7.5, linewidth=lw, color=color, fill=False)

    # Create backboard
    backboard = Rectangle((-30, -7.5), 60, -1, linewidth=lw, color=color)

    # The paint
    # Create the outer box Of the paint, width=16ft, height=19ft
    outer_box = Rectangle((-80, -47.5), 160, 190, linewidth=lw, color=color,␣
↪fill=False)
```

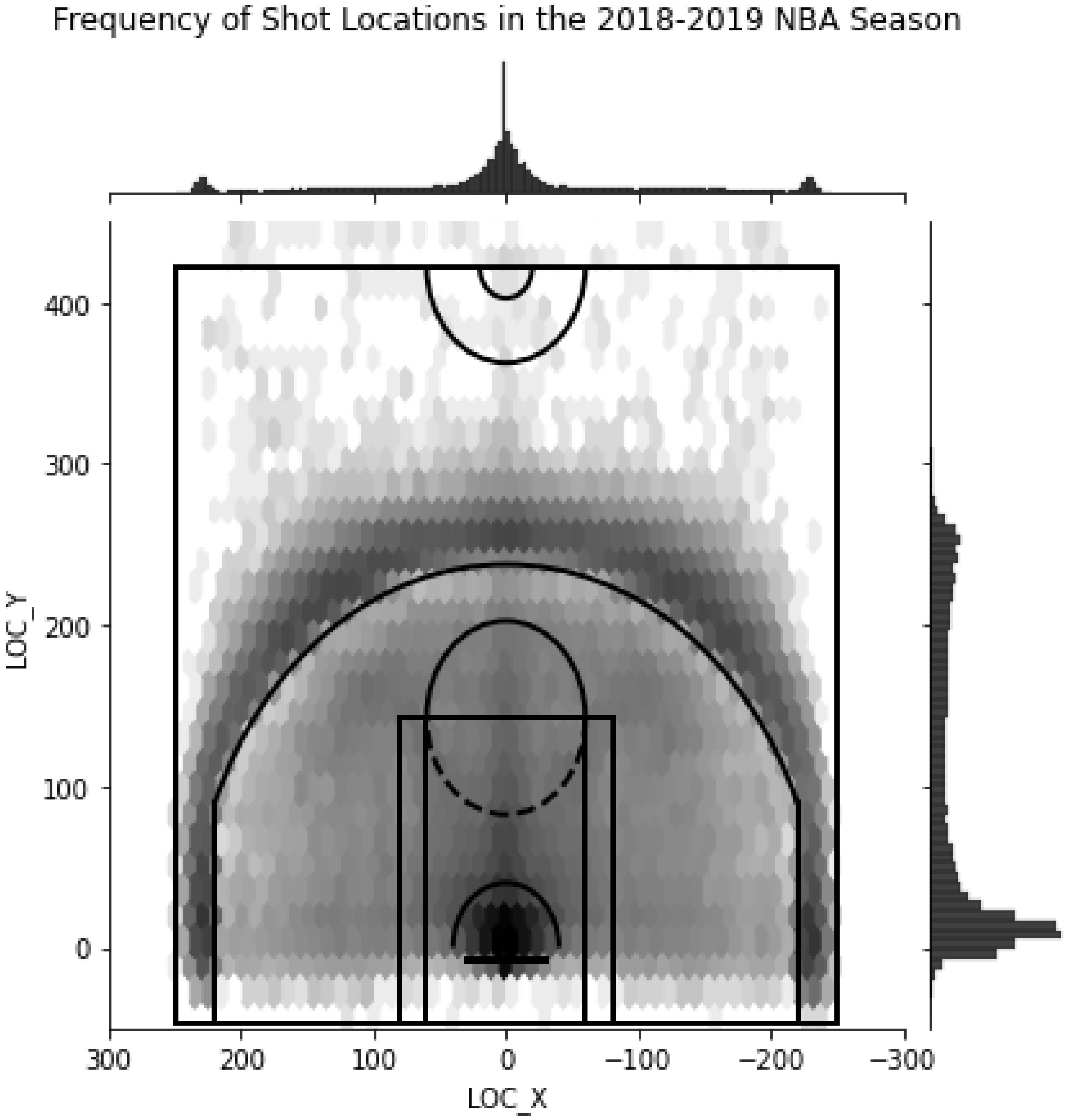
```
    # Create the inner box of the paint, widt=12ft, height=19ft
    inner_box = Rectangle((-60, -47.5), 120, 190, linewidth=lw, color=color,␣
↪fill=False)

    # Create free throw top arc
    top_free_throw = Arc(
        (0, 142.5),
        120,
        120,
        theta1=0,
        theta2=180,
        linewidth=lw,
        color=color,
        fill=False,
    )
    # Create free throw bottom arc
    bottom_free_throw = Arc(
        (0, 142.5),
        120,
        120,
        theta1=180,
        theta2=0,
        linewidth=lw,
        color=color,
        linestyle="dashed",
    )
    # Restricted Zone, it is an arc with 4ft radius from center of the hoop
    restricted = Arc((0, 0), 80, 80, theta1=0, theta2=180, linewidth=lw,␣
↪color=color)

    # Three point line
    # Create the side 3pt lines, they are 14ft long before they begin to arc
    corner_three_a = Rectangle((-220, -47.5), 0, 140, linewidth=lw, color=color)
    corner_three_b = Rectangle((220, -47.5), 0, 140, linewidth=lw, color=color)
    # 3pt arc - center of arc will be the hoop, arc is 23'9" away from hoop
    # I just played around with the theta values until they lined up with the
    # threes
    three_arc = Arc((0, 0), 475, 475, theta1=22, theta2=158, linewidth=lw,␣
↪color=color)

    # Center Court
    center_outer_arc = Arc(
        (0, 422.5), 120, 120, theta1=180, theta2=0, linewidth=lw, color=color
    )
    center_inner_arc = Arc(
        (0, 422.5), 40, 40, theta1=180, theta2=0, linewidth=lw, color=color
    )
    # List of the court elements to be plotted onto the axes
    court_elements = [
        hoop,
        backboard,
        outer_box,
        inner_box,
        top_free_throw,
        bottom_free_throw,
        restricted,
        corner_three_a,
        corner_three_b,
        three_arc,
        center_outer_arc,
        center_inner_arc,
    ]

    if outer_lines:
        # Draw the half court line, baseline and side out bound lines
        outer_lines = Rectangle(
            (-250, -47.5), 500, 470, linewidth=lw, color=color, fill=False
        )
        court_elements.append(outer_lines)

    # Add the court elements onto the axes
    for element in court_elements:
        ax.add_patch(element)

    return ax
```

[29]:

```
#log binning
plt.figure(figsize=(20, 20))
joint_plot = sns.jointplot(shot_chart["LOC_X"], shot_chart["LOC_Y"],␣
 ↪kind="hex", color="#000000", bins = 'log')#"#4CB391")
ax = joint_plot.ax_joint
draw_court(ax, outer_lines=True)
plt.xlim(300, -300)
plt.ylim(-50, 450)
plt.title("Frequency of Shot Locations in the 2018-2019 NBA Season", pad=75)
plt.show()
```

```
/shared-libs/python3.7/py/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  FutureWarning
```

```
<Figure size 1440x1440 with 0 Axes>
```



---

### 0.3 Model Construction and Evaluation

[30]:
```python
def f1Score(x):
    return x[1][1]/(x[1][1]+0.5*(x[0][1]+x[1][0]))
```

[31]:
```python
def precision(x):
    return x[1][1]/(x[0][1]+x[1][1])
```

[32]:
```
def recall(x):
    return x[1][1]/(x[1][0]+x[1][1])
```

---

### 0.3.1 Benchmark Model

[33]:
```
assemblerBM = VectorAssembler(inputCols=["DISTANCE_TRUE"], outputCol="features")
transformedBM = assemblerBM.transform(df4)

#create regression/ data split
mlrBM = LogisticRegression() #maxIter=10, regParam=0.3, elasticNetParam=0.8
lrdataBM = transformedBM.select(transformedBM.label.cast("float"), 'features')

trainingBM, testBM = lrdataBM.randomSplit([.7,.3], seed = 123)
# Fit the model
mlrModelBM = mlrBM.fit(trainingBM)
predictlrBM = mlrModelBM.transform(testBM)
```

[34]:
```
preds_labelslrBM = predictlrBM.select(['prediction','label'])
metricslrBM = MulticlassMetrics(preds_labelslrBM.rdd.map(tuple))

print(metricslrBM.confusionMatrix().toArray())

print("Accuracy: ", metricslrBM.accuracy)
print("Precision: ", precision(metricslrBM.confusionMatrix().toArray()))
print("Recall: ", recall(metricslrBM.confusionMatrix().toArray()))
print("F1 Score: ", f1Score(metricslrBM.confusionMatrix().toArray()))
```

```
[[17700. 10026.]
 [10538. 13013.]]
Accuracy:  0.5989624978060339
Precision:  0.5648248621901992
Recall:  0.5525455394675385
F1 Score:  0.5586177291264219
```

[35]:
```
mlrBM_opt = LogisticRegression()
paramGrid_lrBM = ParamGridBuilder()\
    .addGrid(mlrBM.regParam, [0, 0.1, 0.01]) \
    .addGrid(mlrBM.fitIntercept, [False, True])\
    .addGrid(mlrBM.elasticNetParam, [0, 0.1,0.01])\
    .build()


# We use a ParamGridBuilder to construct a grid of parameters to search over.
# TrainValidationSplit will try all combinations of values and determine best␣
 ↪model using
```

```
# the evaluator.


# In this case the estimator is simply the linear regression.
# A TrainValidationSplit requires an Estimator, a set of Estimator ParamMaps,␣
 ↪and an Evaluator.
tvs_lrBM = TrainValidationSplit(estimator=mlrBM_opt,
                           estimatorParamMaps=paramGrid_lrBM,
                           evaluator=BinaryClassificationEvaluator(),
                           trainRatio=0.7)

# Run TrainValidationSplit, and choose the best set of parameters.
model_opt_lrBM = tvs_lrBM.fit(trainingBM)

# Make predictions on test data. model is the model with combination of␣
 ↪parameters
# that performed best.
preds_opt_lrBM = model_opt_lrBM.transform(testBM)\
    .select("features", "label", "prediction")
```

[36]:

```
preds_opt_lrBM = preds_opt_lrBM.select(['prediction','label'])
metrics_opt_lrBM = MulticlassMetrics(preds_opt_lrBM.rdd.map(tuple))
metrics_auroc_lr_optBM = BinaryClassificationMetrics(preds_opt_lrBM.rdd.
 ↪map(tuple))

print(metrics_opt_lrBM.confusionMatrix().toArray())
print("Accuracy: ", metrics_opt_lrBM.accuracy)
print("Precision: ", precision(metrics_opt_lrBM.confusionMatrix().toArray()))
print("Recall: ", recall(metrics_opt_lrBM.confusionMatrix().toArray()))
print("F1 Score: ", f1Score(metrics_opt_lrBM.confusionMatrix().toArray()))
print("Area under ROC = %s" % metrics_auroc_lr_optBM.areaUnderROC)
```

```
[[17700. 10026.]
 [10538. 13013.]]
Accuracy:  0.5989624978060339
Precision:  0.5648248621901992
Recall:  0.5525455394675385
F1 Score:  0.5586177291264219
Area under ROC = 0.5954677491754485
```

---

### 0.3.2 Logistic Regression Model

[37]:

```
assembler = VectorAssembler(inputCols=["PERIOD", "TIME", "HOME",␣
 ↪"DISTANCE_TRUE", "SHOT_PERC"], outputCol="features")
transformed = assembler.transform(df4)
```

```
#create regression/ data split
lrdata = transformed.select(transformed.label.cast("float"), 'features')

training, test = lrdata.randomSplit([.7,.3], seed = 123)
```

---

### 0.3.3 Logistic Regression Model

[38]:

```
# Load training data
lr = LogisticRegression()#maxIter=10, regParam=0.3, elasticNetParam=0.8)
lrModel = lr.fit(training)

# Print the coefficients and intercept for logistic regression
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

```
Coefficients: [-0.01781970284401227,0.0001834120021738492,0.026385573576242105,-
0.004699462730662614,4.3153362888543745]
Intercept: -2.0901142501416694
```

[39]:

```
predictlr_2 = lrModel.transform(test)

preds_labelslr_2 = predictlr_2.select(['prediction','label'])
metricslr_2 = MulticlassMetrics(preds_labelslr_2.rdd.map(tuple))
metrics_auroc_lr = BinaryClassificationMetrics(preds_labelslr_2.rdd.map(tuple))

print(metricslr_2.confusionMatrix().toArray())
print("Accuracy: ", metricslr_2.accuracy)
print("Precision: ", precision(metricslr_2.confusionMatrix().toArray()))
print("Recall: ", recall(metricslr_2.confusionMatrix().toArray()))
print("F1 Score: ", f1Score(metricslr_2.confusionMatrix().toArray()))
print("Area under ROC = %s" % metrics_auroc_lr.areaUnderROC)
```

```
[[23615.  4111.]
 [15125.  8426.]]
Accuracy:  0.6248610488133081
Precision:  0.6720906117891042
Recall:  0.3577767398411957
F1 Score:  0.46696962979383727
Area under ROC = 0.6047521800627027
```

[40]:

```
lr_opt = LogisticRegression()
# maxIter=10, regParam=0.3, elasticNetParam=0.8)
paramGrid_lr = ParamGridBuilder()\
    .addGrid(lr_opt.regParam, [0, 0.1, 0.01]) \
    .addGrid(lr_opt.fitIntercept, [False, True])\
    .addGrid(lr_opt.elasticNetParam, [0, 0.1,0.01])\
    .build()


# We use a ParamGridBuilder to construct a grid of parameters to search over.
# TrainValidationSplit will try all combinations of values and determine best␣
 ↪model using
# the evaluator.


# In this case the estimator is simply the linear regression.
# A TrainValidationSplit requires an Estimator, a set of Estimator ParamMaps,␣
 ↪and an Evaluator.
tvs_lr = TrainValidationSplit(estimator=lr_opt,
                           estimatorParamMaps=paramGrid_lr,
                           evaluator=BinaryClassificationEvaluator(),
                           trainRatio=0.7)

# Run TrainValidationSplit, and choose the best set of parameters.
model_opt_lr = tvs_lr.fit(training)

# Make predictions on test data. model is the model with combination of␣
 ↪parameters
# that performed best.
pred_opt_lr = model_opt_lr.transform(test)\
    .select("features", "label", "prediction")
```

[41]:
```
preds_labelsopt_lr = pred_opt_lr.select(['prediction','label'])
metrics_opt_lr = MulticlassMetrics(preds_labelsopt_lr.rdd.map(tuple))
metrics_auroc_lr_opt = BinaryClassificationMetrics(preds_labelsopt_lr.rdd.
 ↪map(tuple))

print(metrics_opt_lr.confusionMatrix().toArray())
print("Accuracy: ", metrics_opt_lr.accuracy)
print("Precision: ", precision(metrics_opt_lr.confusionMatrix().toArray()))
print("Recall: ", recall(metrics_opt_lr.confusionMatrix().toArray()))
print("F1 Score: ", f1Score(metrics_opt_lr.confusionMatrix().toArray()))
print("Area under ROC = %s" % metrics_auroc_lr_opt.areaUnderROC)
```

```
[[22308.  5418.]
 [13673.  9878.]]
Accuracy:  0.6276888273494939
Precision:  0.6457897489539749
Recall:  0.4194301728164409
F1 Score:  0.5085592195021494
Area under ROC = 0.6120089621926826
```

### 0.3.4 Linear SVC Model

[42]:
```
svm = LinearSVC()

modelSVC = svm.fit(training)
predictSVC = modelSVC.transform(test)
```

[43]:
```
# modelSVC.summary.roc.select('FPR').collect()
```

[44]:
```
preds_labelsSVC = predictSVC.select(['prediction','label'])
metricsSVC = MulticlassMetrics(preds_labelsSVC.rdd.map(tuple))
metrics_auroc_svc = BinaryClassificationMetrics(preds_labelsSVC.rdd.map(tuple))

print(metricsSVC.confusionMatrix().toArray())
print("Accuracy: ", metricsSVC.accuracy)
print("Precision: ", precision(metricsSVC.confusionMatrix().toArray()))
print("Recall: ", recall(metricsSVC.confusionMatrix().toArray()))
print("F1 Score: ", f1Score(metricsSVC.confusionMatrix().toArray()))
print("Area under ROC = %s" % metrics_auroc_svc.areaUnderROC)
```

```
[[25474.  2252.]
 [17018.  6533.]]
Accuracy:  0.6241979835013749
Precision:  0.7436539556061469
Recall:  0.2773979873466095
F1 Score:  0.40406976744186046
Area under ROC = 0.5980872934641148
```

Linear SVC Hyperparameter Optimization

[45]:
```
svc_opt = LinearSVC()

paramGrid_svc = ParamGridBuilder()\
    .addGrid(svc_opt.regParam, [0.1, 0.01]) \
    .addGrid(svc_opt.fitIntercept, [False, True])\
    .build()


# We use a ParamGridBuilder to construct a grid of parameters to search over.
# TrainValidationSplit will try all combinations of values and determine best␣
 ↪model using
# the evaluator.


# In this case the estimator is simply the linear regression.
# A TrainValidationSplit requires an Estimator, a set of Estimator ParamMaps,␣
 ↪and an Evaluator.
tvs_svc = TrainValidationSplit(estimator=svc_opt,
                           estimatorParamMaps=paramGrid_svc,
                           evaluator=BinaryClassificationEvaluator(),
                           # 80% of the data will be used for training, 20% for␣
 ↪validation.
                           trainRatio=0.7)

# Run TrainValidationSplit, and choose the best set of parameters.
model_opt_svc = tvs_svc.fit(training)

# Make predictions on test data. model is the model with combination of␣
 ↪parameters
# that performed best.
pred_opt_svc = model_opt_svc.transform(test)\
    .select("features", "label", "prediction")
```

[46]:

```
preds_labelsopt_svc = pred_opt_svc.select(['prediction','label'])
metrics_opt_svc = MulticlassMetrics(preds_labelsopt_svc.rdd.map(tuple))
metrics_auroc_svc_opt = BinaryClassificationMetrics(preds_labelsSVC.rdd.
 ↪map(tuple))

print(metrics_opt_svc.confusionMatrix().toArray())
print("Accuracy: ", metrics_opt_svc.accuracy)
print("Precision: ", precision(metrics_opt_svc.confusionMatrix().toArray()))
print("Recall: ", recall(metrics_opt_svc.confusionMatrix().toArray()))
print("F1 Score: ", f1Score(metrics_opt_svc.confusionMatrix().toArray()))
print("Area under ROC = %s" % metrics_auroc_svc_opt.areaUnderROC)
```

```
[[17787.  9939.]
 [10422. 13129.]]
Accuracy:  0.602921387756694
Precision:  0.5691434021154846
Recall:  0.5574710203388391
F1 Score:  0.5632467448894227
Area under ROC = 0.5980872934641148
```

---

### 0.3.5 Random Forest Model

[46]:

```

```

[47]:

```
model = RF(labelCol = 'label', featuresCol = 'features',numTrees= 3,␣
 ↪maxDepth=2, seed = 123)
fit = model.fit(training)
```

[48]:
```
predictrf = fit.transform(test)
predictrf.show(5)
accuracy = predictrf.filter(predictrf.label == predictrf.prediction).count()/
 ↪predictrf.count()
```

```
+-----+--------------------+--------------------+--------------------+----------
+
|label|            features|       rawPrediction|
probability|prediction|
+-----+--------------------+--------------------+--------------------+----------
+
|  0.0|[1.0,0.0,0.0,0.6,…|[1.18023410617620…|[0.39341136872540…|
1.0|
|  0.0|[1.0,0.0,0.0,1.61…|[1.18023410617620…|[0.39341136872540…|
1.0|
|  0.0|[1.0,0.0,0.0,1.67…|[0.79509441017212…|[0.26503147005737…|
1.0|
|  0.0|[1.0,0.0,0.0,1.88…|[1.18023410617620…|[0.39341136872540…|
1.0|
|  0.0|[1.0,0.0,0.0,1.92…|[1.18023410617620…|[0.39341136872540…|
1.0|
+-----+--------------------+--------------------+--------------------+----------
+
only showing top 5 rows
```

[49]:
```
predictrf.select('label').filter(predictrf.label == 0).count()
```

[49]:
```
27726
```

[50]:
```
preds_labelsrf = predictrf.select(['prediction',predictrf.label.cast('float')])
metricsrf = MulticlassMetrics(preds_labelsrf.rdd.map(tuple))
metrics_auroc_rf = BinaryClassificationMetrics(preds_labelsSVC.rdd.map(tuple))

print(metricsrf.confusionMatrix().toArray())
print("Accuracy: ", metricsrf.accuracy)
print("Precision: ", precision(metricsrf.confusionMatrix().toArray()))
print("Recall: ", recall(metricsrf.confusionMatrix().toArray()))
print("F1 Score: ", f1Score(metricsrf.confusionMatrix().toArray()))
print("Area under ROC = %s" % metrics_auroc_rf.areaUnderROC)
```

```
[[22946.  4780.]
 [13746.  9805.]]
Accuracy:  0.638707412680149
Precision:  0.6722660267398012
Recall:  0.41633051675088106
F1 Score:  0.514212292846654
Area under ROC = 0.5980872934641148
```

Random Forest Hyperparameter Optimization

[51]:

```
rf_opt = RF(labelCol = 'label', featuresCol = 'features', seed =␣
 ↪123)#,numTrees= 3, maxDepth=2,

paramGrid = ParamGridBuilder()\
    .addGrid(rf_opt.numTrees, [2,3,4]) \
    .addGrid(rf_opt.maxDepth, [2,3])\
    .build()


# We use a ParamGridBuilder to construct a grid of parameters to search over.
# TrainValidationSplit will try all combinations of values and determine best␣
 ↪model using
# the evaluator.


# In this case the estimator is simply the linear regression.
# A TrainValidationSplit requires an Estimator, a set of Estimator ParamMaps,␣
 ↪and an Evaluator.
tvs_rf = TrainValidationSplit(estimator=rf_opt,
                           estimatorParamMaps=paramGrid,
                           evaluator=BinaryClassificationEvaluator(),
                           # 80% of the data will be used for training, 20% for␣
 ↪validation.
                           trainRatio=0.7)

# Run TrainValidationSplit, and choose the best set of parameters.
model_opt_rf = tvs_rf.fit(training)

# Make predictions on test data. model is the model with combination of␣
 ↪parameters
# that performed best.
pred_opt_rf = model_opt_rf.transform(test)\
    .select("features", "label", "prediction")
```

[52]:

```
preds_labelsopt_rf = pred_opt_rf.select(['prediction','label'])
metrics_opt_rf = MulticlassMetrics(preds_labelsopt_rf.rdd.map(tuple))
metrics_auroc_rf_opt = BinaryClassificationMetrics(preds_labelsSVC.rdd.
 ↪map(tuple))

print(metrics_opt_rf.confusionMatrix().toArray())
print("Accuracy: ", metrics_opt_rf.accuracy)
print("Precision: ", precision(metrics_opt_rf.confusionMatrix().toArray()))
print("Recall: ", recall(metrics_opt_rf.confusionMatrix().toArray()))
```

```python
print("F1 Score: ", f1Score(metrics_opt_rf.confusionMatrix().toArray()))
print("Area under ROC = %s" % metrics_auroc_rf_opt.areaUnderROC)
```

```
[[23501.  4225.]
 [14233.  9318.]]
Accuracy:  0.6400335433040154
Precision:  0.6880307169755593
Recall:  0.39565198929981743
F1 Score:  0.5023993098614331
Area under ROC = 0.5980872934641148
```

Created in Deepnote