Notebook

March 15, 2025

League of Legends: Race to the Nexus

Name(s): Michael Luo, Santiago Cardenas Rey

Website Link: https://sancard1.github.io/LOL_Analysis/

[1]:
```
#Imports libraries for data handling, visualization,
#and machine learning.
#pandas: Data loading and processing
#matplotlib & seaborn: Static plots
#numpy: Numerical operations
#pathlib & os: File path handling
#plotly: Interactive plotting
#markdown: Renders markdown text
#scikit-learn: Tools for model training & evaluation

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
import markdown
import os
pd.options.plotting.backend = 'plotly'
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from sklearn.pipeline import Pipeline
from sklearn.metrics import precision_score

# from dsc80_utils import * # Feel free to uncomment and use this.
```

0.1 Step 1: Introduction

[2]:
```python
# Loads the 2024 LoL esports data into a Pandas DataFrame.
# 'low_memory=False' prevents dtype issues with large CSV files.
# The data will be used to compare ADCs and Mid Laners
# based on the metric DPM / (Deaths + 1).
df = pd.read_csv("2024_LoL_esports_match_data_from_OraclesElixir.csv",
                 low_memory = False)
df
#Question:  Which role-ADCs or Mid Laners-carries their team more
#often based on DPM / (Deaths + 1)?
```

[2]:
```
                     gameid datacompleteness  \
0       10660-10660_game_1          partial
1       10660-10660_game_1          partial
2       10660-10660_game_1          partial
3       10660-10660_game_1          partial
4       10660-10660_game_1          partial
…                      …                …
117571    LOLTMNT02_194401         complete
117572    LOLTMNT02_194401         complete
117573    LOLTMNT02_194401         complete
117574    LOLTMNT02_194401         complete
117575    LOLTMNT02_194401         complete

                                                url league  year split  \
0       https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  2023   NaN
1       https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  2023   NaN
2       https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  2023   NaN
3       https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  2023   NaN
4       https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  2023   NaN
…                                               …    …    …   …
117571                                          NaN  KeSPA  2025   NaN
117572                                          NaN  KeSPA  2025   NaN
117573                                          NaN  KeSPA  2025   NaN
117574                                          NaN  KeSPA  2025   NaN
117575                                          NaN  KeSPA  2025   NaN

        playoffs                 date  game  patch  …  opp_csat25  \
0              0  2024-01-01 05:13:15     1  13.24  …         NaN
1              0  2024-01-01 05:13:15     1  13.24  …         NaN
2              0  2024-01-01 05:13:15     1  13.24  …         NaN
3              0  2024-01-01 05:13:15     1  13.24  …         NaN
4              0  2024-01-01 05:13:15     1  13.24  …         NaN
…            …                    …   …    …  …         …
117571         0  2024-12-08 09:03:13     4  14.23  …       211.0
117572         0  2024-12-08 09:03:13     4  14.23  …       253.0
```

```
117573          0  2024-12-08 09:03:13     4  14.23  …        35.0
117574          0  2024-12-08 09:03:13     4  14.23  …       847.0
117575          0  2024-12-08 09:03:13     4  14.23  …       932.0

       golddiffat25 xpdiffat25 csdiffat25 killsat25 assistsat25 deathsat25  \
0               NaN        NaN        NaN       NaN         NaN        NaN
1               NaN        NaN        NaN       NaN         NaN        NaN
2               NaN        NaN        NaN       NaN         NaN        NaN
3               NaN        NaN        NaN       NaN         NaN        NaN
4               NaN        NaN        NaN       NaN         NaN        NaN
…             …          …          …         …          …          …
117571      -1050.0     1845.0       36.0       0.0         2.0        2.0
117572       -827.0     -702.0       -3.0       1.0         0.0        0.0
117573       -146.0     -383.0       -5.0       0.0         4.0        2.0
117574       3278.0     3672.0       85.0       8.0        16.0        7.0
117575      -3278.0    -3672.0      -85.0       7.0         9.0        8.0

       opp_killsat25 opp_assistsat25 opp_deathsat25
0                NaN             NaN            NaN
1                NaN             NaN            NaN
2                NaN             NaN            NaN
3                NaN             NaN            NaN
4                NaN             NaN            NaN
…              …              …             …
117571           4.0             1.0            2.0
117572           1.0             5.0            0.0
117573           0.0             5.0            2.0
117574           7.0             9.0            8.0
117575           8.0            16.0            7.0

[117576 rows x 161 columns]
```

0.2 Step 2: Data Cleaning and Exploratory Data Analysis

[3]:
```python
# This code analyzes League of Legends match data to compare the
# effectiveness of ADCs and Mid Laners based on DPM / (Deaths + 1).
# It filters key columns, calculates effectiveness, and saves both
# markdown and HTML summaries of the data. Additionally, it visualizes
# DPM distribution and effectiveness by role, exporting the plots as
# interactive HTML files.
assets_dir = "/Users/michaelluo/Desktop/LOL_Analysis/assets"
os.makedirs(assets_dir, exist_ok=True)  # Ensure directory

columns_to_keep = [
    "gameid", "position", "dpm", "kills", "deaths",
    "assists", "result", "league"
]
```

```python
df = df[columns_to_keep]
df = df[~df["position"].str.contains("team", case=False, na=False)]
df.index = range(len(df))

df["effectiveness"] = df["dpm"] / (df["deaths"] + 1)
df_markdown = df.head().to_markdown(index=False)

aggregates_md_path = os.path.join(
    assets_dir, "lol_aggregates.md"
)
with open(aggregates_md_path, "w") as f:
    f.write(df_markdown)

df_html = markdown.markdown(df_markdown, extensions=["tables"])
aggregates_html_path = os.path.join(
    assets_dir, "lol_aggregates.html"
)
with open(aggregates_html_path, "w") as f:
    f.write(
        f"<html><body><h1>League of Legends Data</h1>{df_html}</body>"
        f"</html>"
    )

df.head()

fig = px.histogram(
    df, x="dpm", nbins=200,
    title="Distribution of Damage Per Minute (DPM)",
    labels={"dpm": "Damage Per Minute"}, opacity=0.7,
    marginal="box"
)
fig.update_layout(
    plot_bgcolor="#232323", paper_bgcolor="#232323",
    font=dict(color="white")
)
fig.show()

fig.write_html(
    "/Users/michaelluo/Desktop/LOL_Analysis/"
    "assets/dpm_distribution.html",
    include_plotlyjs="cdn"
)

fig = px.box(
    df, x="position", y="effectiveness",
    title="Effectiveness (DPM / (Deaths + 1)) by Role",
    labels={"position": "Role", "effectiveness": "Effectiveness"},
```

```
    color="position"
)
fig.update_layout(
    plot_bgcolor="#232323", paper_bgcolor="#232323",
    font=dict(color="white")
)
fig.show()

fig.write_html(
    "/Users/michaelluo/Desktop/LOL_Analysis/"
    "assets/effectiveness_boxplot.html",
    include_plotlyjs="cdn"
)

grouped_stats = df.groupby("position").agg({
    "dpm": ["mean", "median"],
    "kills": ["mean", "median"],
    "deaths": ["mean", "median"],
    "assists": ["mean", "median"],
    "effectiveness": ["mean", "median"]
})

grouped_markdown = grouped_stats.to_markdown()
grouped_md_path = os.path.join(assets_dir, "lol_grouped_stats.md")
with open(grouped_md_path, "w") as f:
    f.write(grouped_markdown)

grouped_html = markdown.markdown(
    grouped_markdown, extensions=["tables"]
)
grouped_html_path = os.path.join(
    assets_dir, "lol_grouped_stats.html"
)
with open(grouped_html_path, "w") as f:
    f.write(
        f"<html><body><h1>Grouped Stats by Role</h1>"
        f"{grouped_html}</body></html>"
    )
```

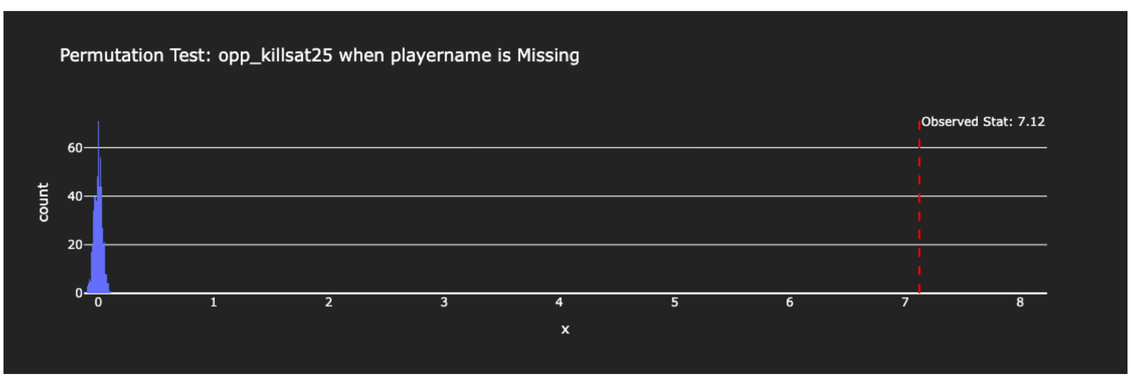
0.3 Step 3: Assessment of Missingness

[5]:
```
# This script analyzes missing data in the LoL dataset.
# It calculates missing 'playername' & 'playerid' per role.
# A permutation test checks if 'playername' missingness
# significantly affects 'opp_killsat25'. Results are saved.
import pandas as pd, numpy as np, os
import plotly.graph_objects as go, plotly.express as px
```

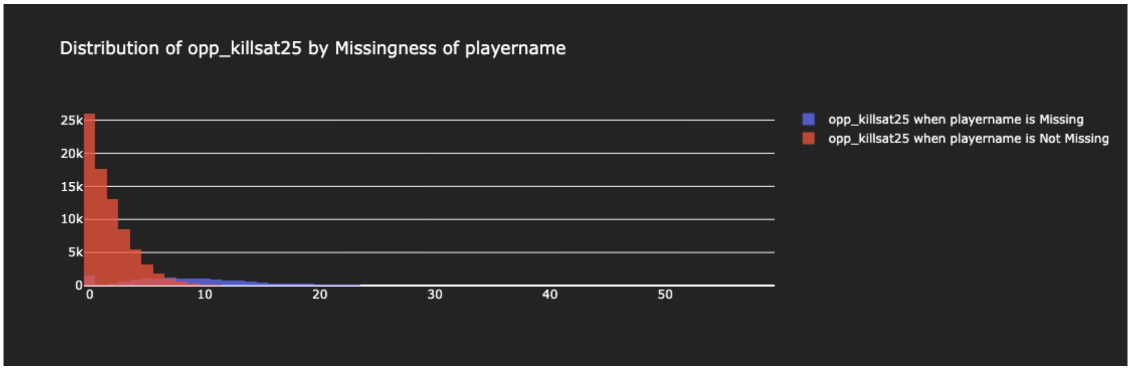
```python
a = pd.read_csv(
    "2024_LoL_esports_match_data_from_OraclesElixir.csv",
    low_memory=False
)

missing_data_analysis = a[[
    "position", "playername", "playerid", "opp_killsat25"
]]

playername_nan_counts = (
    missing_data_analysis.groupby("position")["playername"]
    .apply(lambda x: x.isnull().sum()).to_frame(name="playername")
)

playername_nan_proportions = (
    playername_nan_counts / playername_nan_counts["playername"].sum()
)

playerid_nan_counts = (
    missing_data_analysis.groupby("position")["playerid"]
    .apply(lambda x: x.isnull().sum()).to_frame(name="playerid")
)

playerid_nan_proportions = (
    playerid_nan_counts / playerid_nan_counts["playerid"].sum()
)

def permutation_test_missingness(
    df, col_x, col_y, num_permutations=1000
):
    observed_stat = (
        np.mean(df[df[col_x].isnull()][col_y]) -
        np.mean(df[df[col_x].notnull()][col_y])
    )

    permuted_statistics = []
    for _ in range(num_permutations):
        permuted_x = df[col_x].sample(frac=1, replace=False).values
        permuted_df = df.copy()
        permuted_df[col_x] = permuted_x
        permuted_stat = (
            np.mean(permuted_df[permuted_df[col_x].isnull()][col_y]) -
            np.mean(permuted_df[permuted_df[col_x].notnull()][col_y])
        )
        permuted_statistics.append(permuted_stat)
```

```
    p_value = np.mean(
        np.abs(permuted_statistics) >= np.abs(observed_stat)
    )
    return observed_stat, permuted_statistics, p_value

col_x, col_y = "playername", "opp_killsat25"
observed_stat, permuted_stats, p_val = (
    permutation_test_missingness(missing_data_analysis, col_x, col_y)
)

fig_perm = px.histogram(
    x=permuted_stats, nbins=50,
    title=f"Permutation Test: {col_y} when {col_x} is Missing"
)

fig_perm.add_vline(
    x=observed_stat, line_dash="dash", line_color="red",
    annotation_text=f"Observed Stat: {observed_stat:.2f}"
)

fig_perm.update_layout(
    paper_bgcolor="#232323", plot_bgcolor="#232323",
    font=dict(color="white")
)

fig_perm.show()

fig_perm.write_html(
    "/Users/michaelluo/Desktop/LOL_Analysis/assets/"
    "missing_permutation.html", include_plotlyjs="cdn"
)

print(f"Observed Statistic: {observed_stat:.4f}")
print(f"P-value: {p_val:.4f}")

if p_val < 0.05:
    print(
        f"Missingness of '{col_x}' significantly affects '{col_y}' "
        "(p < 0.05)."
    )
else:
    print(
        f"Missingness of '{col_x}' does not significantly affect '{col_y}' "
        "(p >= 0.05)."
    )

y_missing = missing_data_analysis[
```

```
    missing_data_analysis[col_x].isnull()
][col_y]

y_not_missing = missing_data_analysis[
    missing_data_analysis[col_x].notnull()
][col_y]

fig_dist = go.Figure()
fig_dist.add_trace(
    go.Histogram(x=y_missing, name=f"{col_y} when {col_x} is Missing")
)

fig_dist.add_trace(
    go.Histogram(x=y_not_missing, name=f"{col_y} when {col_x} is Not Missing")
)

fig_dist.update_layout(
    barmode="overlay",
    title=f"Distribution of {col_y} by Missingness of {col_x}",
    paper_bgcolor="#232323", plot_bgcolor="#232323",
    font=dict(color="white")
)

fig_dist.update_traces(opacity=0.75)
fig_dist.show()

fig_dist.write_html(
    "/Users/michaelluo/Desktop/LOL_Analysis/assets/"
    "distribution_missing.html", include_plotlyjs="cdn"
)
```



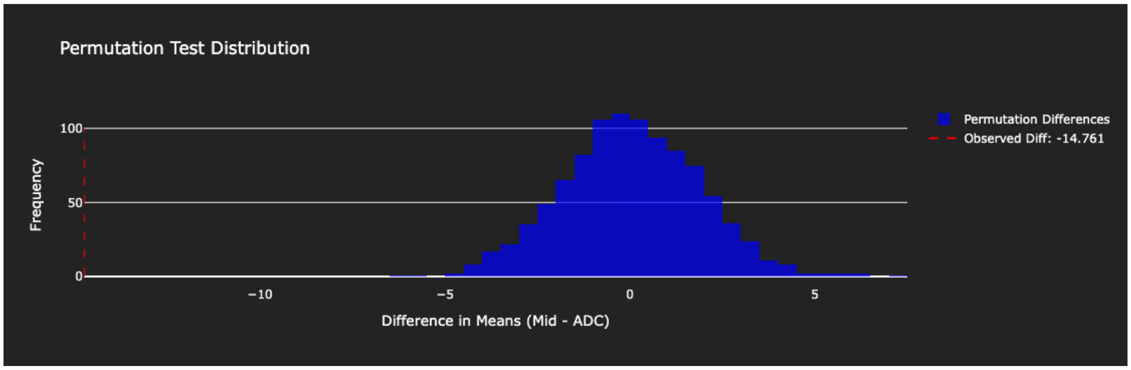

```
Observed Statistic: 7.1241
P-value: 0.0000
Missingness of 'playername' significantly affects 'opp_killsat25' (p < 0.05).
```



0.4 Step 4: Hypothesis Testing

[6]:
```python
# Conducts a permutation test to compare effectiveness between
# mid and bot positions, visualizing the distribution of differences.
df_mid = df[df["position"] == "mid"]["effectiveness"].values
df_adc = df[df["position"] == "bot"]["effectiveness"].values
obs_diff = np.mean(df_mid) - np.mean(df_adc)

num_permutations = 1000
perm_diffs = []

for _ in range(num_permutations):
    shuffled = np.random.permutation(df["effectiveness"].values)
    mid_perm = shuffled[:len(df_mid)]
    adc_perm = shuffled[len(df_mid): len(df_mid) + len(df_adc)]
    perm_diffs.append(np.mean(mid_perm) - np.mean(adc_perm))

p_value = np.mean(np.array(perm_diffs) >= obs_diff)

fig = go.Figure()

fig.add_trace(
    go.Histogram(
        x=perm_diffs, nbinsx=30, marker_color="blue",
        opacity=0.7, name="Permutation Differences"
    )
)

fig.add_trace(
    go.Scatter(
        x=[obs_diff, obs_diff], y=[0, 100], mode="lines",
        line=dict(color="red", dash="dash"),
        name=f"Observed Diff: {obs_diff:.3f}"
    )
```

```python
)

fig.update_layout(
    title="Permutation Test Distribution",
    xaxis_title="Difference in Means (Mid - ADC)",
    yaxis_title="Frequency",
    showlegend=True,
    plot_bgcolor="#232323",
    paper_bgcolor="#232323",
    font=dict(color="white")
)

fig.show()

fig.write_html(
    "/Users/michaelluo/Desktop/LOL_Analysis/assets/"
    "permutation_test.html",
    include_plotlyjs="cdn"
)
```



0.5 Step 5: Framing a Prediction Problem

[7]:
```python
#Prediction Problem: We want to identify the role of the player
#given their post-game data. This will imply for us to do a
#classification model.
```

0.6 Step 6: Baseline Model

[8]:
```python
# Loads and preprocesses LoL esports data, engineers an
# effectiveness feature, and trains a RandomForest model
# to classify player positions based on in-game stats.
df = pd.read_csv(
    "2024_LoL_esports_match_data_from_OraclesElixir.csv",
```

```python
    low_memory=False
)

df = df[~df["position"].str.contains(
    "team", case=False, na=False
)]
df.index = range(len(df))

df["effectiveness"] = df["dpm"] / (df["deaths"] + 1)

df.dropna(
    axis=1, thresh=int(0.7 * len(df)), inplace=True
)

target = "position"
selected_features = [
    "kills", "deaths", "effectiveness", "teamkills",
    "monsterkills", "minionkills"
]

X = df[selected_features]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("classifier", RandomForestClassifier(
        n_estimators=100, random_state=42
    ))
])

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.6753
              precision    recall  f1-score   support

         bot       0.47      0.48      0.47      3928
         jng       1.00      1.00      1.00      3962
         mid       0.42      0.39      0.40      3976
```

```
         sup       0.97      0.95      0.96      3899
         top       0.51      0.56      0.53      3831

    accuracy                           0.68     19596
   macro avg       0.68      0.68      0.67     19596
weighted avg       0.68      0.68      0.67     19596
```

0.7 Step 7: Final Model

[9]:

```python
# Loads and processes LoL esports data, engineers new features,
# tunes a RandomForest model using GridSearchCV, evaluates it
# with a confusion matrix, and saves predictions and reports.
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import (
    train_test_split, GridSearchCV
)
from sklearn.metrics import (
    confusion_matrix, classification_report
)
import plotly.figure_factory as ff
import plotly.io as pio

df = pd.read_csv(
    "2024_LoL_esports_match_data_from_OraclesElixir.csv",
    low_memory=False
)

df = df[~df["position"].str.contains(
    "team", case=False, na=False
)]
df.index = range(len(df))

df["kill_participation"] = (
    (df["kills"] + df["assists"]) / df["teamkills"]
)
df["gold_efficiency"] = df["earnedgold"] / (df["deaths"] + 1)
df["wards_placed"] = df["wardsplaced"]

target = "position"
selected_features = [
    "kills", "deaths", "kill_participation",
    "gold_efficiency", "monsterkills",
```

```
    "minionkills", "wards_placed"
]

df = df[selected_features + [target]]
X = df[selected_features]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("classifier", RandomForestClassifier(random_state=42))
])

param_grid = {
    "classifier__n_estimators": [100, 200],
    "classifier__max_depth": [None, 10, 20],
    "classifier__min_samples_split": [2, 5]
}

grid_search = GridSearchCV(
    pipeline, param_grid, cv=5, scoring="accuracy",
    n_jobs=-1, verbose=1
)
grid_search.fit(X_train, y_train)

tuned_pipeline = grid_search.best_estimator_
y_pred = tuned_pipeline.predict(X_test)

df_test = X_test.copy()
df_test["actual_position"] = y_test.values
df_test["predicted_position"] = y_pred

df_test.to_csv(
    "modified_LoL_esports_data_with_predictions.csv",
    index=False
)

cm = confusion_matrix(y_test, y_pred)
classes = list(np.unique(y_test))

fig = ff.create_annotated_heatmap(
    z=cm, x=classes, y=classes, colorscale="blues"
)
```

```python
fig.update_layout(
    title="Confusion Matrix for Final Model",
    xaxis_title="Predicted",
    yaxis_title="Actual",
    paper_bgcolor="#232323",
    plot_bgcolor="#232323",
    font=dict(color="white")
)

pio.write_html(
    fig,
    file="/Users/michaelluo/Desktop/LOL_Analysis/assets/"
         "confusion_matrix.html",
    include_plotlyjs="cdn"
)
fig.show()

report = classification_report(y_test, y_pred)
print(report)

report_html = (
    f"<pre style='color: white; background-color: #232323;"
    f" padding: 10px;'>{report}</pre>"
)

with open(
    "/Users/michaelluo/Desktop/LOL_Analysis/assets/"
    "classification_report.html", "w"
) as f:
    f.write(report_html)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
              precision    recall  f1-score   support

         bot       0.53      0.51      0.52      3928
         jng       1.00      1.00      1.00      3962
         mid       0.47      0.35      0.40      3976
         sup       0.97      0.95      0.96      3899
         top       0.51      0.69      0.59      3831

    accuracy                           0.70     19596
   macro avg       0.70      0.70      0.69     19596
weighted avg       0.70      0.70      0.69     19596
```

0.8 Step 8: Fairness Analysis

[ ]:
```python
# Performs a permutation test to compare precision scores
# between high and lower-performing groups, visualizing results.
import pandas as pd
import numpy as np
from sklearn.metrics import precision_score
import matplotlib.pyplot as plt
import plotly.graph_objects as go

a = pd.read_csv(
    "modified_LoL_esports_data_with_predictions.csv"
)

def group_assignment(x):
    return ("high_performing" if x in ["sup", "jng"]
            else "lower_performing")

permutation_df = a.copy()
permutation_df["group"] = (
    permutation_df["actual_position"].apply(group_assignment)
)

precision_high_performing = precision_score(
    permutation_df[permutation_df["group"] == "high_performing"]
    ["actual_position"],
    permutation_df[permutation_df["group"] == "high_performing"]
    ["predicted_position"],
    average="macro"
)

precision_low_performing = precision_score(
    permutation_df[permutation_df["group"] == "lower_performing"]
    ["actual_position"],
    permutation_df[permutation_df["group"] == "lower_performing"]
    ["predicted_position"],
    average="macro"
)

observed_difference = (
    float(precision_high_performing) -
    float(precision_low_performing)
)

perm_num = 1000
permutation_differences = []
```

```python
for _ in range(perm_num):
    shuffled_groups = np.random.permutation(
        permutation_df["group"]
    )
    permutation_df["shuffled_group"] = shuffled_groups

    precision_high_shuffled = precision_score(
        permutation_df[permutation_df["shuffled_group"]=="high_performing"]
        ["actual_position"],
        permutation_df[permutation_df["shuffled_group"]=="high_performing"]
        ["predicted_position"],
        average="macro"
    )

    precision_low_shuffled = precision_score(
        permutation_df[permutation_df["shuffled_group"]=="lower_performing"]
        ["actual_position"],
        permutation_df[permutation_df["shuffled_group"]=="lower_performing"]
        ["predicted_position"],
        average="macro"
    )

    perm_difference = (
        float(precision_high_shuffled) -
        float(precision_low_shuffled)
    )
    permutation_differences.append(perm_difference)

p_value = np.mean(
    np.array(permutation_differences) >= observed_difference
)

plt.hist(permutation_differences, bins=30)
plt.axvline(x=observed_difference, color="r", linestyle="--")
plt.xlabel("Difference in Precision (Permuted)")
plt.ylabel("Frequency")
plt.title("Permutation Test Results")
plt.show()

fig = go.Figure()

fig.add_trace(go.Histogram(
    x=permutation_differences, nbinsx=60,
    name="Permutation Differences"
))

fig.add_vline(
```

```
    x=observed_difference, line_dash="dash",
    line_color="red", name="Observed Difference"
)

fig.update_layout(
    title="Permutation Test Results",
    xaxis_title="Difference in Precision (Permuted)",
    yaxis_title="Frequency",
    bargap=0.1
)

fig.show()

print(permutation_differences)
```

[ ]:

```

```

This notebook was converted with convert.ploomber.io