```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
import requests
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
print("Imports successful.")
results_df = pd.read_csv('results.csv',low_memory=False)
picks_df = pd.read_csv('picks.csv',low_memory=False)
economy_df = pd.read_csv('economy.csv',low_memory=False)
players_df = pd.read_csv('players.csv',low_memory=False)
#read CSVs into data frames
print("Data cached.")
# Filter by team. Lets get only data that includes Navi's stats.
#team_1, #team_2. If one of these teams are Natus Vincere, keep it.
target = "Natus Vincere"
results_df = results_df[(results_df.team_1 == target) | (results_df.team_2 ==
target)]
results_df.head()
print("Successful.")
# Filter by team. Lets get only data that includes Navi's stats.
#team_1, #team_2. If one of these teams are Natus Vincere, keep it.
target = "Natus Vincere"
picks_df = picks_df[(picks_df.team_1 == target) | (picks_df.team_2 ==
target)]
picks_df.head()
print("Successful.")
target = "Natus Vincere"
players_df = players_df[(players_df.team == target) | (players_df.opponent ==
target)]
players_df.head()
print("Successful.")
economy_df = economy_df[(economy_df.team_1 == target) | (economy_df.team_2 ==
target)]
economy_df.head()
print("Successful.")
#Get columns from all datasets
results_columns = list(results_df.columns)

print(results_columns)

print("Successful.")
results_df = results_df[results_df.team_2 != "Natus Vincere"]
results_df.to_csv('natus_vincere_results.csv', index=False)
economy_df.to_csv('natus_vincere_economy.csv', index=False)
players_df.to_csv('natus_vincere_players.csv', index=False)
picks_df.to_csv('natus_vincere_picks.csv', index=False)
```

```
results_df
#Get the preprocessed data and put them into seperate CSV files.
#Now, every piece of data I work with has Natus Vincere as team_2
#Match Winner (match_winner) is the variable we will look for to check if
Navi/Natus Vincere wins.
#First lets move match_winner_name and match_winner to the end


#column_to_move5 = df_results_players.pop('match_winner_team')
column_to_move6 = results_df.pop('match_winner')
#df_results_players.insert(len(df_results_players.columns)-1,
'match_winner_team', column_to_move5)
results_df.insert(len(results_df.columns)-2, 'match_winner', column_to_move6)
print("successfully shifted over match winner column")
results_df
#Lets take df_results_players and make a new column, match_winner_team. Then
rewrite to csv
results_df['match_winner_name'] = np.where(results_df['match_winner'] == 1,
'Natus Vincere', results_df['team_2'])
results_df.head(100)
results_df.to_csv('natus_vincere_results.csv', index = False)
print('Results has been edited')
results_df
results_columns = []
results_columns = list(results_df)
results_df['navi_win'] =
results_df.match_winner_name.astype("category").cat.codes
print("Successful. We have the navi win column.")
results_df.to_csv('natus_vincere_results.csv', index = False)
print('Results has been edited')
results_df
results_df
#I will try to find the top 40 most "interesting" features.
#Basically, 40 factors with the most significant correlation to Navi winning
#Find the correlation, and perform preprocessing

#results_df = pd.read_csv('natus_vincere_results.csv')
#results_df.columns = results_columns

#complete_correlations = complete_correlations.drop('match_winner', axis=1)

complete_correlations =
results_df[results_columns+['navi_win']].corr(numeric_only = True)
complete_correlations.drop('match_winner', axis=0, inplace=True)
complete_correlations.drop('match_winner', axis=1, inplace=True)
print(complete_correlations['navi_win'].apply(abs).sort_values( ascending =
False).iloc[:100])
#Drop match winner from the process of finding variables most important to
Navi winning the match
#complete_correlations
#print(complete_correlations)
#limited_correlations =
df_economy_results[limited_columns+['navi_win']].corr(numeric_only = True)
```

```
#print(((complete_correlations['navi_win'].apply(abs)).sort_values( ascending
= False)).iloc[:100])

#.sort_values())
#.sort_values(by = correlations, ascending=False).iloc[:25])
#We have many features. Lets limit, I dont want to added a lot of noise to
the model
important_features = []
for c in results_columns+['navi_win']:
    try:
        if abs(complete_correlations[c]['navi_win']) > 0.065:
            important_features.append(c)
    except KeyError:
            pass
selected_data = results_df[important_features]
print("data selection success")
selected_data
#note 26 is a win for navi
#Great, lets visualize our correlation chart now that we have a good
dataframe
plt.figure(figsize = (18,12))
sns.heatmap(selected_data.corr().sort_values(by = 'navi_win'), annot = True,
cmap = 'viridis')
#As we see there is a high correlation between Navi winning (navi_win) and
variables such
#as t_1 (their performance on the T side), ct_1 (their performance on the CT
side), etc
#Interestingly, map_winner negatively impacts navi_win. Perhaps comparatively
in our dataset
#Navi did not win as many maps as other teams did. So their chances of
winning are negatively impacted
selected_data.hist(figsize=(18,12))
selected_data.info()
#Lets start training the model

#split
X, y = selected_data.drop(['navi_win'], axis = 1), selected_data['navi_win']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2)
print("Model has been trained")
len(X_train)
len(X_test)
scalar = StandardScaler()
X_train_scaled = scalar.fit_transform(X_train)
X_test_scaled = scalar.fit_transform(X_test)
forests = RandomForestClassifier(n_jobs=4)
forests.fit(X_train_scaled, y_train)
print("Data scaled")
forests.score(X_test_scaled, y_test)
```