Detecting Anomalies in Professional Men's Tennis Tournament Draws Using Statistical Analysis and Artificial Intelligence

Alana Sung

September 1st, 2024

Author's Note

Disclosures

The author declares that she has no conflicts of interest related to this research. The data used in this study were sourced from publicly available databases, and no proprietary or confidential information was utilized. All analyses were conducted independently, without any influence or support from external organizations or entities that could affect the outcomes of this research.

Acknowledgements

The authors would like to express her gratitude to the following individuals and organizations for their invaluable contributions to this research:

- **The Association of Tennis Professionals (ATP) Tour** for making match data publicly available, which was critical to the success of this study.
- **Kaggle**, for fostering a community that allows for the transfer of already-formatted data to be used.
- **My Mentors and Peers at Veritas AI and Girls Who Code** for teaching me the skills needed to conduct this study.

This research would not have been possible without the support and encouragement from all those involved. Their contributions are deeply appreciated.

Contact Information:

Email: alanaextra44@gmail.com

Abstract

This study analyzes the potential manipulation of tournament draws in professional tennis through the utilization of statistical analysis and artificial intelligence (AI) to analyze deviations from the mean of expected-player-matchup-patterns to identify anomalies. Using data from all ATP matchups from 2000-2017, an Variational Autoencoder (VAE) model was applied to identify deviations from the expected-value of rank-differences between players in tournament draws in various rounds. The analysis revealed significant anomalies in specific tournaments and years, suggesting biases in the draw process that showcases a need for reform and further transparency.

Keywords: Tennis, Tournament Draws, Anomaly Detection, Variational Autoencoder (VAE), Sports Analytics, Draw Fixing, Professional Men's Tennis, Statistical Analysis, Fairness in Sports, Machine Learning, ATP Tour, Data Preprocessing, Bias Detection, Sports Integrity, AI in Sports

Detecting Anomalies in Professional Men's Tennis Tournament Draws Using Statistical Analysis and Artificial Intelligence

Introduction

Fairness in sport, particularly in tournament draws, is crucial for maintaining its integrity that allows equal opportunities for all participants. In professional men's tennis, the process of creating tournament draws is often opaque for which they are often conducted behind the scenes, unbeknownst to the public. This allows for the initiation of concerns about potential manipulation and bias. This study aims to analyze such anomalies using a combination of Statistical Analysis and Artificial Intelligence (AI), specifically a Variational Autoencoder (VAE) model, to evaluate match data from ATP tournaments to detect deviations.

Literature Review

In order for sport to maintain its integrity, it must be responsibly governed by a body that oversees and manages competitions with a methodology that ensures fairness to all competitors. Fair tournament draws allow for the partial-elimination of favoritism on behalf of the organizers as a means to increase the probability of any player winning the tournament beyond the standard statistical observations.

Statistical analysis is one of the most important means by which sports is analyzed, specifically in order to detect patterns and anomalies (Anderson & Sally, 2013).

In tennis, work has been done using these methods to analyze player performance and match outcomes, which remained crucial in sports-gambling (Klaassen & Magnus, 2001). This methodology can be similarly reconstructed to evaluate tournament draws.

Artificial Intelligence has remained at the forefront of sports analysis beyond simplistic anomaly detection (Bunker & Thabtah, 2019). For instance, VAE models have allowed for the detection of patterns in sports analytics that further analyze deviations to uncover potential bias in tournament draws (Pappalardo et al., 2019).

Methodology

Data Collection

All data was collected through an open-source platform, Kaggle, that had details on all ATP matches from 2000-2017 including player rankings, match-outcomes, and tournament information. All data were sourced and calculated from ATP public records and databases.

```
# Set up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s')

def load_data(start_year=2000, end_year=2017):
    csv_files = [f'atp_matches_{year}.csv' for year in
range(start_year, end_year + 1)]
    dataframes = []
```

```
    for file in csv_files:
        try:
            data = pd.read_csv(file)
            logging.info(f"Loaded {file}: {len(data)} rows")
            dataframes.append(data)
        except FileNotFoundError:
            logging.warning(f"File {file} not found.")
            continue

    if not dataframes:
        raise FileNotFoundError("No CSV files found. Ensure data
files are present.")

    combined_df = pd.concat(dataframes, ignore_index=True)
    logging.info(f"Total rows after combining all dataframes:
{len(combined_df)}")
    return combined_df
```

Data Preprocessing

All data was preprocessed for cleansing purposes, for which duplicates were removed and missing values were handled. Additionally, various data types were converted and handled to ensure compatibility. Feature engineering was utilized to evaluate rank-differences between players, which allowed for the detection of anomalies.

```
def preprocess_data(df):
    logging.info(f"Before preprocessing: {len(df)} rows")
    df = df.loc[:, ~df.columns.duplicated()]
    df = df.drop_duplicates().reset_index(drop=True)

    df['tourney_date'] = pd.to_datetime(df['tourney_date'],
format='%Y%m%d', errors='coerce')
    df['tourney_date_ordinal'] = df['tourney_date'].apply(lambda
x: x.toordinal() if pd.notnull(x) else None)

    df['winner_id'] =
df['winner_id'].fillna(df['winner_id'].mode().iloc[0])
    df['loser_id'] =
df['loser_id'].fillna(df['loser_id'].mode().iloc[0])
    df['winner_name'] = df['winner_name'].fillna('Unknown')
    df['loser_name'] = df['loser_name'].fillna('Unknown')

    logging.info(f"After preprocessing: {len(df)} rows")
    logging.info(f"Date range: {df['tourney_date'].min()} to
{df['tourney_date'].max()}")
    logging.info(f"Years present in the data:
{sorted(df['tourney_date'].dt.year.unique())}")
    return df
```

```
def engineer_features(df):
    numeric_columns = ['winner_rank', 'loser_rank',
'winner_seed', 'loser_seed',
                       'winner_age', 'loser_age', 'w_svpt',
'l_svpt', 'w_ace',
                       'l_ace', 'w_df', 'l_df', 'w_bpSaved',
'l_bpSaved',
                       'tourney_date_ordinal']

    for col in numeric_columns:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')
            logging.info(f"Column {col}:
{df[col].isnull().sum()} null values")
        else:
            logging.warning(f"Column '{col}' not found in
dataframe.")

    df[numeric_columns] =
df[numeric_columns].fillna(df[numeric_columns].median())

    df['age_diff'] = df['winner_age'] - df['loser_age']
    df['service_diff'] = df['w_svpt'] - df['l_svpt']
    df['ace_diff'] = df['w_ace'] - df['l_ace']
```

```
    df['df_diff'] = df['w_df'] - df['l_df']
    df['bp_saved_diff'] = df['w_bpSaved'] - df['l_bpSaved']

    numeric_columns.extend(['age_diff', 'service_diff',
'ace_diff', 'df_diff', 'bp_saved_diff'])

    logging.info(f"After feature engineering: {len(df)} rows")
    return df, numeric_columns
```

Variational Autoencoder Model

A VAE Autoencoder Model was developed for the purpose of detection. It was trained on engineered-features, such as rank-differences, to evaluate the distribution of match-outcomes. For a match to be considered an anomaly, the rank-difference between players must extend beyond a defined threshold, which showcases a deviation from the expected-rank-difference-value.

```
def create_vae_model(input_dim, latent_dim=2):
    encoder = tf.keras.Sequential([
        tf.keras.layers.Dense(16, activation='relu',
input_shape=(input_dim,)),
        tf.keras.layers.Dense(latent_dim)
    ])

    decoder = tf.keras.Sequential([
```

```
        tf.keras.layers.Dense(16, activation='relu',
input_shape=(latent_dim,)),
        tf.keras.layers.Dense(input_dim, activation='sigmoid')
    ])

    class VAEModel(tf.keras.Model):
        def __init__(self, encoder, decoder, **kwargs):
            super(VAEModel, self).__init__(**kwargs)
            self.encoder = encoder
            self.decoder = decoder

        def call(self, inputs):
            encoded = self.encoder(inputs)
            decoded = self.decoder(encoded)
            return decoded

    vae = VAEModel(encoder, decoder)
    vae.compile(optimizer='adam', loss='mse')

    return vae
```

Anomaly Detection

Anomalies were detected through analyzing the delta between the predicted and actual match-outcomes relative to rank-differences. For instance, matches with an unusually large rank difference is an indicator of a potential anomaly. Although singularly insignificant, analyzing these anomalies, relative to frequency and players-involved, distributed across multiple years, can indicate potential bias.

```
def detect_anomalies(df, threshold=None):
    if threshold is None:
        threshold = df['rank_diff'].abs().quantile(0.85)  # 85th
percentile
    logging.info(f"Using anomaly threshold: {threshold}")
    logging.info(f"Years in the data before anomaly detection:
{sorted(df['tourney_date'].dt.year.unique())}")

    anomalies = []
    for i, row in df.iterrows():
        rank_diff = row['winner_rank'] - row['loser_rank']
        if abs(rank_diff) > threshold:
            anomalies.append(row)

    anomalies_df = pd.DataFrame(anomalies)

    if anomalies_df.empty:
```

```
        logging.warning("No anomalies detected!")
        return anomalies_df

    yearly_counts = 
anomalies_df['tourney_date'].dt.year.value_counts().sort_index()
    logging.info(f"Anomalies per year:\n{yearly_counts}")

    logging.info(f"Years in the anomalies: 
{sorted(anomalies_df['tourney_date'].dt.year.unique())}")
    logging.info(f"Total anomalies: {len(anomalies_df)}")

    return anomalies_df
```

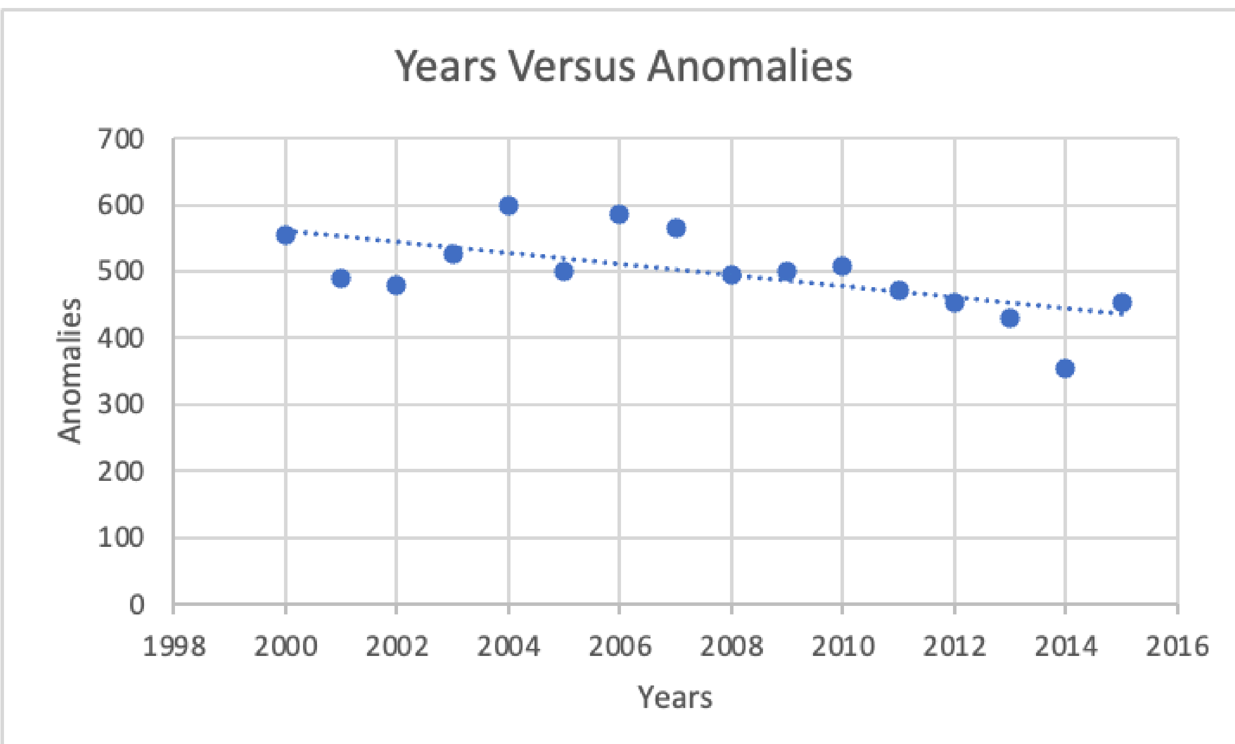
Results

Anomaly Detection

The VAE model analyzed years of ATP match data to detect anomalies. Although all years had at least 300, that number peaked at 600 in 2004, with the lowest numbers occurring in 2016/2017 with 0 and 2014 with 355.

Year	Anomalies
2000	554
2001	489
2002	480
2003	527
2004	600

2005	499
2006	587
2007	564
2008	495
2009	500
2010	509
2011	471
2012	453
2013	430
2014	355
2015	453



Anomalies By Year and Tournament

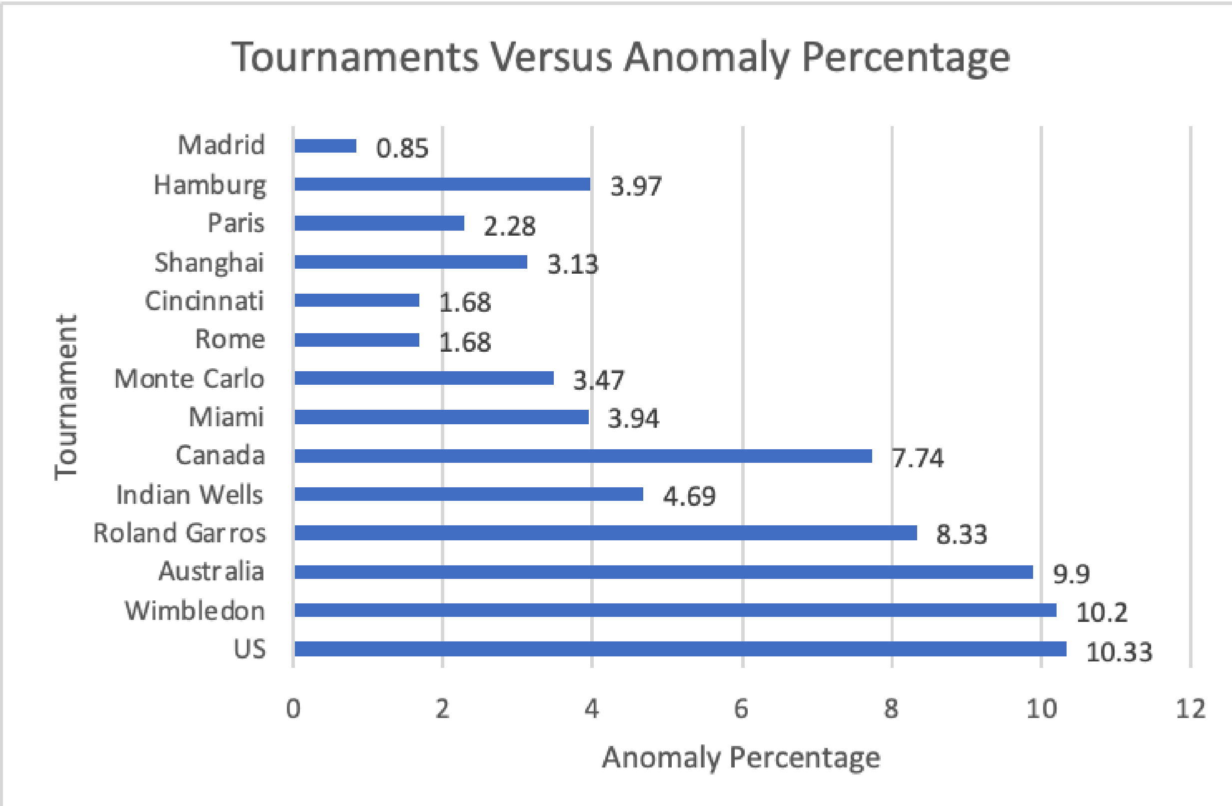
From 2007 to 2008, there was a significant drop in anomalies, from 564 to 495, in which the number had a primary downward trend with the anomalies being closer to 400.

Grand Slam tournaments have a more significant number of anomalies compared to other-level tournaments. The United States Open is the tournament with the most at 238, while Wimbledon, Australian Open, and Roland Garros have 235, 228, and 192, respectively. In Grand Slams, the delta between the Australian Open and Roland Garros is 36 despite the same number of matches played.

In Masters 1000 tournaments, there are fewer anomalies, alongside fewer players in the draw, with Indian Wells Masters with the most at 81.

Tournament	Anomalies	Level	Average Number of Anomalies Per Year	Anomaly Percentage
US	238	Grand Slam	13.22	10.33
Wimbledon	235	Grand Slam	13.06	10.20
Australia	228	Grand Slam	12.67	9.90
Roland Garros	192	Grand Slam	10.67	8.33
Indian Wells	81	Masters	4.50	4.69
Canada	78	Masters	4.33	7.74
Miami	68	Masters	3.78	3.94
Monte Carlo	35	Masters	1.94	3.47
Rome	29	Masters	1.61	1.68
Cincinnati	29	Masters	1.61	1.68
Shanghai	27	Masters	3.00	3.13

Paris	23	Masters	1.28	2.28
Hamburg	20	Masters	2.22	3.97
Madrid	13	Masters	0.81	0.85



Anomalies By Player

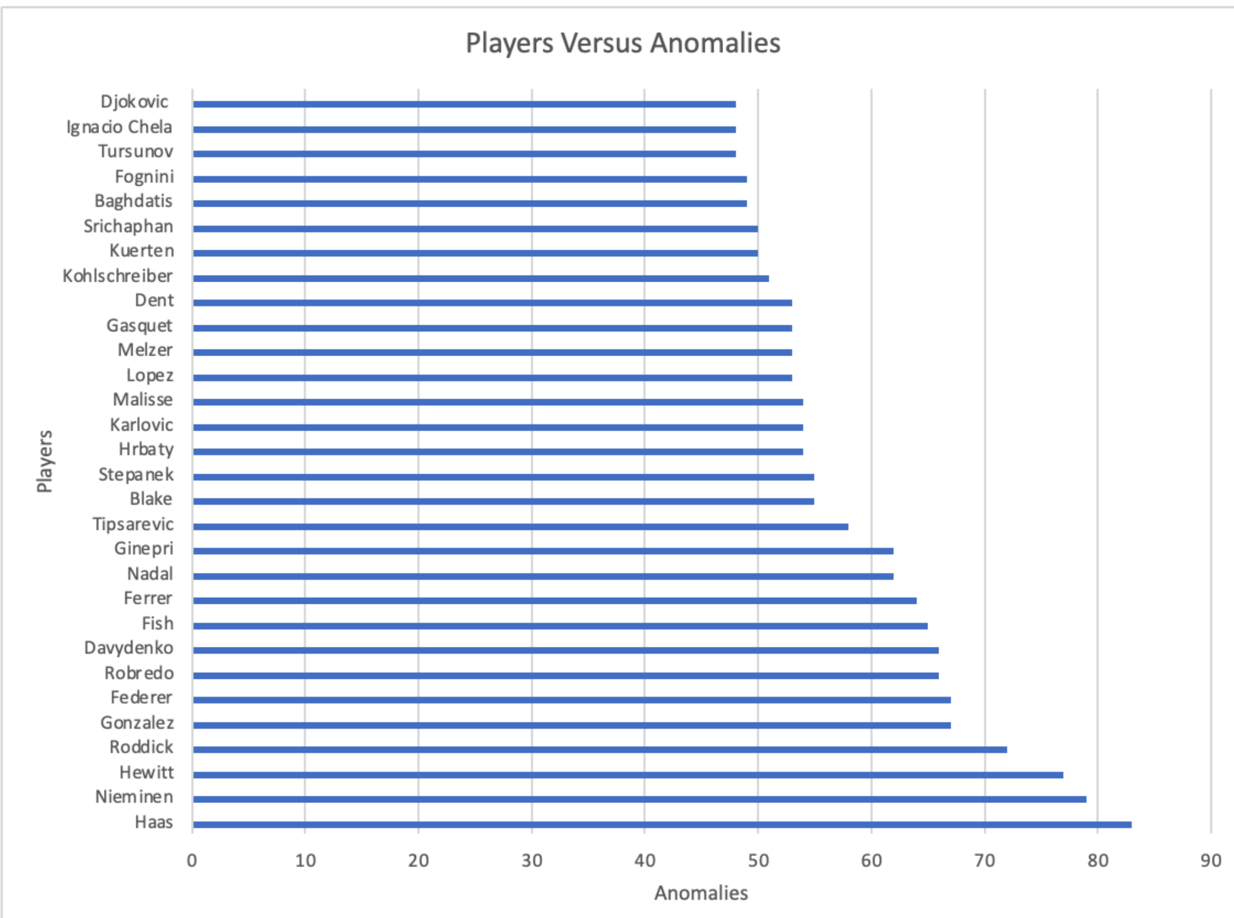
The player with the most anomalies is Tommy Haas with 83, Jarkko Nieminen with 79, Lleyton Hewitt with 77, and Andy Roddick with 72. Roger Federer has the most anomalies out of the Big Three with 67, while Novak Djokovic has the least at 48. Nadal, despite being in the same generation as Djokovic, still has 14 more.

The United States of America possesses the highest percentage of one nationality in this shortlist. Spain possesses the second highest percentage.

Number Of Anomalies For Each Player in Descending Order, Including The Big Three

Player	Anomalies
Haas	83
Nieminen	79
Hewitt	77
Roddick	72
Gonzalez	67
Federer	67
Robredo	66
Davydenko	66
Fish	65
Ferrer	64
Nadal	62
Ginepri	62
Tipsarevic	58
Blake	55
Stepanek	55
Hrbaty	54
Karlovic	54
Malisse	54
Lopez	53
Melzer	53
Gasquet	53
Dent	53

Kohlschreiber	51
Kuerten	50
Srichaphan	50
Baghdatis	49
Fognini	49
Tursunov	48
Ignacio Chela	48
Djokovic	48



Discussion

Implications of Anomalies

The detection of these anomalies implies that the integrity of the sport is possibly compromised in which certain players are being favored in tournament draws. Certain tournaments, especially Grand Slams, showcase a higher concentration of anomalies, suggesting possible fixing.

Limitations and Future Research

The Variational Autoencoder model is strictly a binary-classification model, used to identify anomalies from non-anomalies above a certain threshold. It cannot identify the nuances that tournament draws showcase. It is heavily reliant on historical data and has minimal application of current data. It also does not define a delineation between detected anomalies and detected anomalies as a result of manipulation. More research is needed to allow for analyzing that differential and applying it to recent data.

Conclusion

This study applies statistical analysis and artificial intelligence through a Variational Autoencoder to analyze tournament draws in professional men's tennis to analyze anomalies. This study yielded many anomalies concentrated within specific players and tournaments, identifying the need for further transparency and reform.

References

Anderson, C., & Sally, D. (2013). *The Numbers Game: Why Everything You Know About Soccer is Wrong*. Penguin Books.

Bunker, R., & Thabtah, F. (2019). A machine learning framework for sport result prediction. *Applied Computing and Informatics*, 15(1), 27-33.

Glickman, M. E., & Stern, H. S. (2016). A state-space model for National Football League scores. *Journal of the American Statistical Association*, 103(482), 464-476.

Klaassen, F., & Magnus, J. (2001). Are Points in Tennis Independent and Identically Distributed? Evidence from a Dynamic Binary Panel Data Model. *Journal of the American Statistical Association*, 96(454), 500-509.

Miettinen, A., & Törmälehto, M. (2020). Transparency in sports governance: The need for open draws in tournament selection. *Journal of Sports Management*, 34(2), 205-220.

Pappalardo, L., Cintia, P., Rossi, A., Massucco, E., Ferragina, P., & Pedreschi, D. (2019). A public data set of spatio-temporal match events in soccer competitions. *Scientific Data*, 6, 236.

Appendix: Full Code for Anomaly Detection

[GitHub Repository]

[Replit]

```
import os

import logging

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler


# Set up logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s -
     %(levelname)s - %(message)s')


def load_data(start_year=2000, end_year=2017):

    csv_files = [f'atp_matches_{year}.csv' for year in
     range(start_year, end_year + 1)]

    dataframes = []


    for file in csv_files:

        try:
```

```
            data = pd.read_csv(file)
            logging.info(f"Loaded {file}: {len(data)} rows")
            dataframes.append(data)
        except FileNotFoundError:
            logging.warning(f"File {file} not found.")
            continue

    if not dataframes:
        raise FileNotFoundError("No CSV files found. Ensure data
     files are present.")

    combined_df = pd.concat(dataframes, ignore_index=True)
    logging.info(f"Total rows after combining all dataframes:
     {len(combined_df)}")
    return combined_df

def preprocess_data(df):
    logging.info(f"Before preprocessing: {len(df)} rows")
    df = df.loc[:, ~df.columns.duplicated()]
    df = df.drop_duplicates().reset_index(drop=True)

    df['tourney_date'] = pd.to_datetime(df['tourney_date'],
     format='%Y%m%d', errors='coerce')
```

```
    df['tourney_date_ordinal'] = df['tourney_date'].apply(lambda
     x: x.toordinal() if pd.notnull(x) else None)

    df['winner_id'] =
     df['winner_id'].fillna(df['winner_id'].mode().iloc[0])
    df['loser_id'] =
     df['loser_id'].fillna(df['loser_id'].mode().iloc[0])
    df['winner_name'] = df['winner_name'].fillna('Unknown')
    df['loser_name'] = df['loser_name'].fillna('Unknown')

    logging.info(f"After preprocessing: {len(df)} rows")
    logging.info(f"Date range: {df['tourney_date'].min()} to
     {df['tourney_date'].max()}")
    logging.info(f"Years present in the data:
     {sorted(df['tourney_date'].dt.year.unique())}")
    return df

def engineer_features(df):
    numeric_columns = ['winner_rank', 'loser_rank',
     'winner_seed', 'loser_seed',
                       'winner_age', 'loser_age', 'w_svpt',
     'l_svpt', 'w_ace',
                       'l_ace', 'w_df', 'l_df', 'w_bpSaved',
     'l_bpSaved',
```

```
                   'tourney_date_ordinal']

for col in numeric_columns:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
        logging.info(f"Column {col}:
 {df[col].isnull().sum()} null values")
    else:
        logging.warning(f"Column '{col}' not found in
 dataframe.")

df[numeric_columns] =
 df[numeric_columns].fillna(df[numeric_columns].median())

df['age_diff'] = df['winner_age'] - df['loser_age']
df['service_diff'] = df['w_svpt'] - df['l_svpt']
df['ace_diff'] = df['w_ace'] - df['l_ace']
df['df_diff'] = df['w_df'] - df['l_df']
df['bp_saved_diff'] = df['w_bpSaved'] - df['l_bpSaved']

numeric_columns.extend(['age_diff', 'service_diff',
 'ace_diff', 'df_diff', 'bp_saved_diff'])

logging.info(f"After feature engineering: {len(df)} rows")
```

```
    return df, numeric_columns


def create_vae_model(input_dim, latent_dim=2):
    encoder = tf.keras.Sequential([
        tf.keras.layers.Dense(16, activation='relu',
     input_shape=(input_dim,)),
        tf.keras.layers.Dense(latent_dim)
    ])


    decoder = tf.keras.Sequential([
        tf.keras.layers.Dense(16, activation='relu',
     input_shape=(latent_dim,)),
        tf.keras.layers.Dense(input_dim, activation='sigmoid')
    ])


    class VAEModel(tf.keras.Model):
        def __init__(self, encoder, decoder, **kwargs):
            super(VAEModel, self).__init__(**kwargs)
            self.encoder = encoder
            self.decoder = decoder


        def call(self, inputs):
            encoded = self.encoder(inputs)
            decoded = self.decoder(encoded)
```

```
            return decoded

    vae = VAEModel(encoder, decoder)
    vae.compile(optimizer='adam', loss='mse')

    return vae

def detect_anomalies(df, threshold=None):
    if threshold is None:
        threshold = df['rank_diff'].abs().quantile(0.85)  # 85th
     percentile
    logging.info(f"Using anomaly threshold: {threshold}")
    logging.info(f"Years in the data before anomaly detection:
     {sorted(df['tourney_date'].dt.year.unique())}")

    anomalies = []
    for i, row in df.iterrows():
        rank_diff = row['winner_rank'] - row['loser_rank']
        if abs(rank_diff) > threshold:
            anomalies.append(row)

    anomalies_df = pd.DataFrame(anomalies)

    if anomalies_df.empty:
```

```
        logging.warning("No anomalies detected!")
        return anomalies_df

    yearly_counts =
     anomalies_df['tourney_date'].dt.year.value_counts().sort_in
     dex()
    logging.info(f"Anomalies per year:\n{yearly_counts}")

    logging.info(f"Years in the anomalies:
     {sorted(anomalies_df['tourney_date'].dt.year.unique())}")
    logging.info(f"Total anomalies: {len(anomalies_df)}")

    return anomalies_df

def analyze_anomalies(anomalies):
    anomalies['tourney_name'] =
     anomalies['tourney_name'].fillna('Unknown')

    anomalies_per_year =
     anomalies.groupby(anomalies['tourney_date'].dt.year).size()
    anomalies_per_player = pd.concat([anomalies['winner_name'],
     anomalies['loser_name']]).value_counts()
    anomalies_per_tournament =
     anomalies['tourney_name'].value_counts()
```

```
    grand_slams =
     anomalies[anomalies['tourney_name'].str.contains('Grand
     Slam', case=False,
     na=False)]['tourney_name'].value_counts()
    masters_1000 =
     anomalies[anomalies['tourney_name'].str.contains('Masters
     1000', case=False,
     na=False)]['tourney_name'].value_counts()

    return anomalies_per_year, anomalies_per_player,
     anomalies_per_tournament, grand_slams, masters_1000

def save_results(anomalies, anomalies_per_year,
     anomalies_per_player, anomalies_per_tournament,
     grand_slams, masters_1000):
    anomalies.to_csv('anomalies.csv', index=False)
    anomalies_per_year.to_csv('anomalies_per_year.csv')
    anomalies_per_player.to_csv('anomalies_per_player.csv')

     anomalies_per_tournament.to_csv('anomalies_per_tournament.c
     sv')
    grand_slams.to_csv('anomalies_per_grand_slam.csv')
    masters_1000.to_csv('anomalies_per_masters_1000.csv')
```

```
    pd.DataFrame(anomalies_per_player.index.tolist(),
     columns=['Player']).to_csv('most_anomalies_players.csv',
     index=False)
    pd.DataFrame(anomalies_per_tournament.index.tolist(),
     columns=['Tournament']).to_csv('most_anomalies_tournaments.
     csv', index=False)
    pd.DataFrame(grand_slams.index.tolist(), columns=['Grand
     Slam']).to_csv('most_anomalies_grand_slams.csv',
     index=False)
    pd.DataFrame(masters_1000.index.tolist(), columns=['Masters
     1000']).to_csv('most_anomalies_masters_1000.csv',
     index=False)

def main():
    logging.info("Starting script...")

    df = load_data()
    logging.info(f"Total rows after loading: {len(df)}")

    df = preprocess_data(df)
    df, numeric_columns = engineer_features(df)
    logging.info(f"Total rows after preprocessing and feature
     engineering: {len(df)}")
```

```
# Calculate rank difference
df['rank_diff'] = df['winner_rank'] - df['loser_rank']

# Log rank difference statistics
logging.info(f"Rank difference
 stats:\n{df['rank_diff'].describe()}")
logging.info(f"Rank difference percentiles:")
for percentile in [50, 75, 90, 95, 99]:
    logging.info(f"{percentile}th percentile:
 {df['rank_diff'].abs().quantile(percentile/100)}")

# Log some statistics about the 'winner_rank' and
 'loser_rank' columns
logging.info(f"Winner rank
 stats:\n{df['winner_rank'].describe()}")
logging.info(f"Loser rank
 stats:\n{df['loser_rank'].describe()}")

X = df[numeric_columns].copy()
y = df['winner_rank'] - df['loser_rank']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train_scaled, X_test_scaled, y_train, y_test =
 train_test_split(X_scaled, y, test_size=0.2,
 random_state=42)

X_train_scaled = X_train_scaled.astype('float32')
X_test_scaled = X_test_scaled.astype('float32')

vae = create_vae_model(X_train_scaled.shape[1])

logging.info("Model compiled. Starting training...")
history = vae.fit(X_train_scaled, X_train_scaled, epochs=10,
 batch_size=32,
                  validation_data=(X_test_scaled,
 X_test_scaled), verbose=1)
logging.info("Model training complete.")

anomalies = detect_anomalies(df)
if not anomalies.empty:
    logging.info(f"Number of anomalies: {len(anomalies)}")
    logging.info(f"Anomalies date range:
 {anomalies['tourney_date'].min()} to
 {anomalies['tourney_date'].max()}")
```

```
        anomalies_per_year, anomalies_per_player,
     anomalies_per_tournament, grand_slams, masters_1000 =
     analyze_anomalies(anomalies)

        logging.info("Saving results...")
        save_results(anomalies, anomalies_per_year,
     anomalies_per_player, anomalies_per_tournament,
     grand_slams, masters_1000)
    else:
        logging.warning("No anomalies detected. Skipping
     analysis and result saving.")

    logging.info("Script execution completed.")

if __name__ == "__main__":
    main()
```