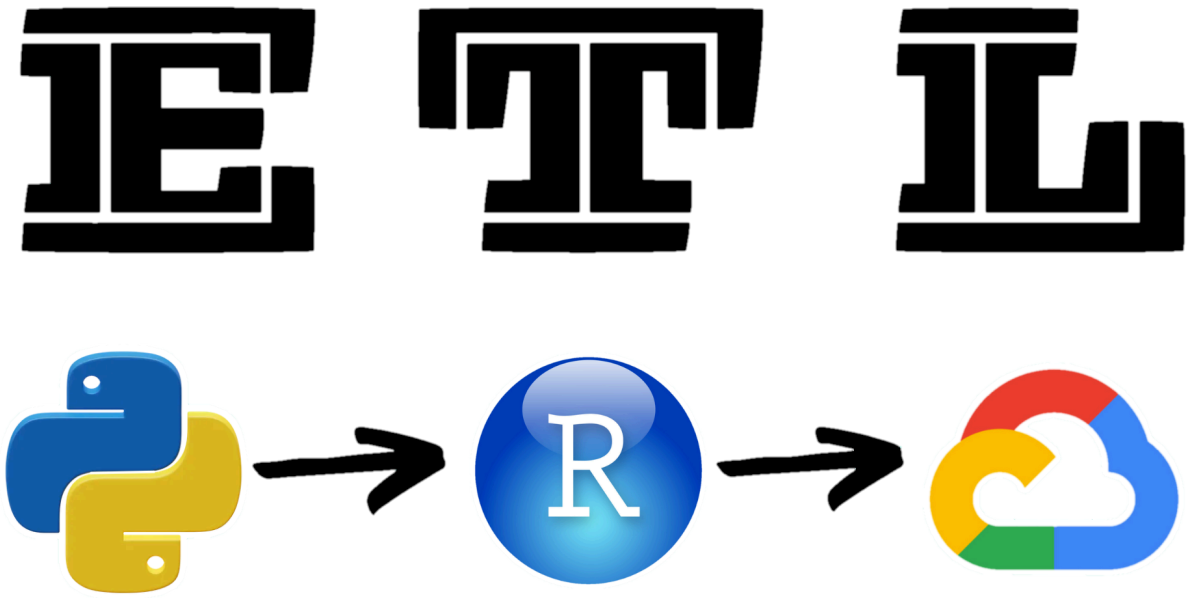

Data Warehousing (ETL)
EXTRACT

Sawandi Kirby

2024-05-15

Contents

1 ETL -(Extract Transform Load)

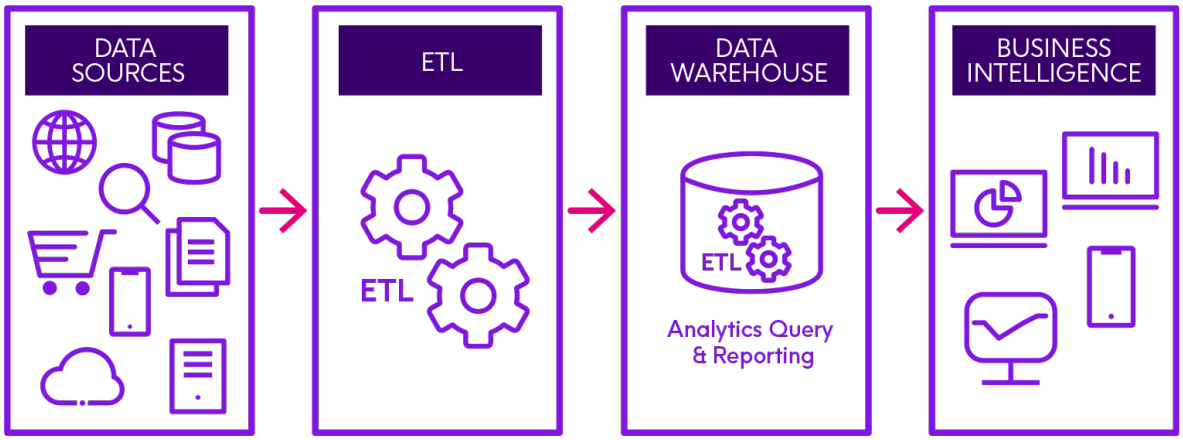

Figure 1: Extract Transform Load

1.1 Definition

1.1.1 ETL stands for "extract, transform, and load". It's a data integration process that combines data from multiple sources into a central repository, such as a data warehouse or data lake, and uses business rules to clean and organize it. The ETL process involves:

1.1.2 Extract: Raw data is copied or exported from source locations to a staging area. For example, data sources can include SQL or NoSQL servers, CRM and ERP systems, flat files, or email.

1.1.3 Transform: Cleansing, mapping, and transforming the raw data into a format that can be used by different applications.

1.1.4 Load: Writing the converted data from a staging area to a target database.

1.1.5 ETL enables data analysis to provide actionable business information, effectively preparing data for analysis and business intelligence processes. For example, an ETL tool might take updated accounting information from an ERP system (extract), combine it with other accounting data (transform), and store the transformed data in an organization's data lake for analytical analysis.

1.1.6 ETL pipelines are a set of tools and activities that move data from one system to another.

1.1.7 This is an overview of the data pipeline I've created for NBA Data scraped from the internet.

2 Extract

2.1 Python- Web Scrape

3 Virtual Enviornment Setup

3.1 Import Required modules

[5]:
```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException, NoSuchElementException
from selenium.webdriver.common.action_chains import ActionChains
import concurrent.futures
from concurrent.futures import ThreadPoolExecutor
import requests
from dotenv import load_dotenv
from bs4 import BeautifulSoup
```

```
import logging
import time
import pandas as pd
import os
import xlsxwriter
import openpyxl
import sys
```

3.2 Create a requirements.txt file

[6]:
```
with open('requirements.txt', 'w') as f:
    f.write('selenium==3.141.0\n')
    f.write('pandas==1.3.3\n')
    f.write('webdriver-manager==3.4.2\n')
    f.write('bs4==0.0.2\n')
    f.write('xlsxwriter==3.2.0\n')
    f.write('openpyxl==3.1.2\n')
```

3.3 Print our directories

[7]:
```
print("Current Working Directory", os.getcwd())

tmp_dir = os.getenv('TMP')
print("Temporary directory:", tmp_dir)
```

```
Current Working Directory D:\PyCharm\PipeLine
Temporary directory: D:\TMP
```

3.4 Set the PATH environment variable to the directory containing chromedriver

[8]:
```
os.environ["PATH"] += os.pathsep + "D:
 ↪\\PyCharm\\chromedriver_win32\\chromedriver.exe"
```

3.5 Define the directory to save Excel files

[9]:
```
excel_directory = "D:\\Scrape Files\\NBA\\NBA_2023"
```

4 Determine what ifnormation is going to be scraped

4.1 Define start URL

[10]:
```
start_url = "https://www.basketball-reference.com"
```

4.2 List the teams

[11]:
```
teams = ["ATL", "BOS", "BRK", "CHO", "CHI", "CLE", "DAL", "DEN", "DET", "GSW",␣
 ↪"HOU", "IND", "LAC", "LAL", "MEM",
         "MIA", "MIL", "MIN", "NOP", "NYK", "OKC", "ORL", "PHI", "PHO", "POR",␣
 ↪"SAC", "SAS", "TOR", "UTA", "WAS"]
```

4.3 List the Pages to scrape through

[12]:
```
page_names = ["Roster & Stats", "Starting Lineups", "Splits", "Game Log",␣
 ↪"Schedule & Results"]
```

4.4 Create a directory for each team to store their respective Excel files

[13]:
```
team_excel_directories = {team: os.path.join(excel_directory, team) for team in␣
 ↪teams}
```

4.5 Create directories if they don't exist

[14]:
```
for team_directory in team_excel_directories.values():
    os.makedirs(team_directory, exist_ok=True)
```

4.6 Define URL Template for each page

[15]:
```
page_url_template = {
    "Roster & Stats": '{start}/teams/{link}/2023.html',
    "Starting Lineups": '{start}/teams/{link}/2023_start.html',
    "Splits": '{start}/teams/{link}/2023/splits/',
    "Game Log": '{start}/teams/{link}/2023/gamelog/',
    "Schedule & Results": '{start}/teams/{link}/2023_games.html'
}
```

4.7 Initialize the team_url dictionary

[16]:
```
team_url = {}
for team in teams:
    team_url[team] = {}
    for page in page_names:
        url_template = page_url_template[page]
        team_url[team][page] = url_template.format(start=start_url, link=team)
```

4.8 Define the list of table IDs to scrape for each page

[17]:
```
table_ids = {
    "Roster & Stats": ['roster', 'per_game', 'totals'],
    "Starting Lineups": ['starting_lineups_po0'],
    "Splits": ['team_splits'],
    "Game Log": ['tgl_basic'],
    "Schedule & Results": ['games']
}
```

4.9 Define a dictionary where keys are table IDs and values are column names

[54]:
```
column_names_dict = {
    'roster': ['Player', 'Position', 'Height', 'Weight', 'BirthDate',␣
 ↪'BirthCountry', 'Experience', 'College'],
    'per_game': ['Player', 'Age', 'GamesPlayedPerGame', 'GamesStarted',␣
 ↪'MinutesPlayedPerGame', 'FieldGoalsPerGame',
```

```
                'FieldGoalAttemptsPerGame', 'FieldGoalPercentPerGame',␣
↪'ThreePointFieldGoalsPerGame', 'ThreePointFieldGoalAttemptsPerGame',
                'ThreePointFieldGoalPercentPerGame',␣
↪'TwoPointFieldGoalsPerGame', 'TwoPointFieldGoalAttemptsPerGame',␣
↪'TwoPointFieldGoalPercentPerGame',
                'EffectiveFieldGoalPercentPerGame', 'FreeThrowsPerGame',␣
↪'FreeThrowAttemptsPerGame', 'FreeThrowPercentPerGame',
                'OffensiveReboundsPerGame', 'DefensiveReboundsPerGame',␣
↪'TotalReboundsPerGame', 'AssistsPerGame', 'StealsPerGame', 'BlocksPerGame',
                'TurnoversPerGame', 'PersonalFoulsPerGame', 'PointsPerGame'],
  'totals': ['Player', 'Age', 'TotalGamesPlayed', 'TotalGamesStarted',␣
↪'TotalMinutesPlayed', 'TotalFieldGoalsPerGame',
             'TotalFieldGoalAttempts', 'TotalFieldGoalPercent',␣
↪'TotalThreePointFieldGoalsPerGame',
             'TotalThreePointFieldGoalAttempts',␣
↪'TotalThreePointFieldGoalPercent', 'TotalTwoPointFieldGoalsPerGame',
             'TotalTwoPointFieldGoalAttempts',␣
↪'TotalTwoPointFieldGoalPercent',
             'TotalEffectiveFieldGoalPercent', 'TotalFreeThrows',␣
↪'TotalFreeThrowAttempts', 'TotalFreeThrowPercent',
             'TotalOffensiveRebounds', 'TotalDefensiveRebounds',␣
↪'TotalRebounds', 'TotalAssists', 'TotalSteals', 'TotalBlocks',␣
↪'TotalTurnovers',
             'TotalPersonalFouls', 'TotalPoints'],
  'starting_lineups_po0': ['Date', 'Start(ET)', '_', 'BoxScore', 'HomeAway',␣
↪'Opponent', 'WinLoss', 'Overtime',
                          'TeamPoints', 'OpponentPoints', 'Wins', 'Losses',␣
↪'StartingLineup'],
  'team_splits': ['Value', 'Games', 'Wins', 'Losses', 'FieldGoals',␣
↪'FieldGoalAttempts', 'ThreePoints',
                  'ThreePointAttempts', 'FreeThrows', 'FreeThrowAttempts',␣
↪'OffensiveRebounds', 'TotalRebounds',
                  'Assists', 'Steals', 'Blocks', 'Turnovers',␣
↪'PersonalFouls', 'Points', 'FieldGoalPercent', 'ThreePointPercent',
                  'EffectiveFieldGoalPercent', 'FreeThrowPercent',␣
↪'TSPercent', 'OppFieldGoals', 'OppFieldGoalAttempts', 'OppThreePoints',
                  'OppThreePointAttempts', 'OppFreeThrows',␣
↪'OppFreeThrowAttempts', 'OppOffensiveRebounds', 'OppTotalRebounds',
                  'OppAssists', 'OppSteals', 'OppBlocks', 'OppTurnovers',␣
↪'OppPersonalFouls', 'OppPoints', 'OppFieldGoalPercent',␣
↪'OppThreePointPercent',
                  'OppEffectiveFieldGoalPercent', 'OppFreeThrowPercent',␣
↪'OppTSPercent'],
  'tgl_basic': ['_1', 'Date', 'HomeAway', 'Opp', 'WinLoss', 'Points',␣
↪'OppPoints', 'FieldGoals', 'FieldGoalAttempts',
```

```
                'FieldGoalPercent', 'ThreePoints', 'ThreePointAttempts',␣
 ↪'ThreePointPercent', 'FreeThrows', 'FreeThrowAttempts',
                'FreeThrowPercent', 'OffensiveRebounds', 'TotalRebounds',␣
 ↪'Assists', 'Steals', 'Blocks', 'Turnovers',
                'PersonalFouls', '_2', 'OppFieldGoals',␣
 ↪'OppFieldGoalAttempts', 'OppFieldGoalPercent', 'OppThreePoints',
                'OppThreePointAttempts', 'OppThreePointPercent',␣
 ↪'OppFreeThrows', 'OppFreeThrowAttempts', 'OppFreeThrowPercent',
                'OppOffensiveRebounds', 'OppTotalRebounds', 'OppAssists',␣
 ↪'OppSteals', 'OppBlocks', 'OppTurnovers', 'OppPersonalFouls'],
   'games': ['Date', 'Start(ET)', '_', 'BoxScore', 'HomeAway', 'Opponent',␣
 ↪'Winloss', 'Overtime', 'Points', 'OppPoints', 'Wins', 'Losses', 'Streak',␣
 ↪'Notes']
}
```

5 Confirm setup

5.1 Print the generated team URLs

[18]:
```
print(team_url["DEN"])
```

```
{'Roster & Stats': 'https://www.basketball-reference.com/teams/DEN/2023.html',
'Starting Lineups': 'https://www.basketball-
reference.com/teams/DEN/2023_start.html', 'Splits': 'https://www.basketball-
reference.com/teams/DEN/2023/splits/', 'Game Log': 'https://www.basketball-
reference.com/teams/DEN/2023/gamelog/', 'Schedule & Results':
'https://www.basketball-reference.com/teams/DEN/2023_games.html'}
```

5.2 Print Example of how to access a specific team's URL for a specific page

[52]:
```
print(team_url["CHI"]["Splits"])
```

```
https://www.basketball-reference.com/teams/CHI/2023/splits/
```

5.3 Print Example of how to access the table IDs for a specific page

[53]:
```
print(table_ids["Splits"])
```

```
['team_splits']
```

6 Chrome WebDriver

6.1 Define the Chrome Driver Options

[55]:
```
chrome_options = Options()
chrome_options.add_argument("--disable-extensions")
chrome_options.add_argument("--disable-gpu")
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument("--disable-dev-shm-usage")
chrome_options.add_argument("--disable-features=VizDisplayCompositor")
chrome_options.add_argument("--disable-setuid-sandbox")
```

```
chrome_options.add_argument("--block-new-web-contents")

prefs = {
    "profile.managed_default_content_settings.images": 2,
    "profile.managed_default_content_settings.javascript": 2,
    "profile.managed_default_content_settings.css": 2,
}
chrome_options.add_experimental_option("prefs", prefs)
```

6.2 Open the webdriver

[56]:
```
driver = webdriver.Chrome(options=chrome_options)
```

6.3 Add a timeout duration between opening pages

[57]:
```
wait = WebDriverWait(driver, 5)
```

6.4 Initialize logging to help us track what is happening

[58]:
```
logging.basicConfig(filename='scraping_log.txt', level=logging.ERROR)
```

7 Define Functions

7.1 Define a function to scrape data for a single team

[59]:
```
def scrape_team_data(team_url_value, team_name, table_ids):
    try:
        driver.get(team_url_value)
        team_data = {}
        # Scrape data for each table ID
        for table_id in table_ids:
            table_data = []
            table = wait.until(EC.presence_of_element_located((By.ID,␣
 ↪table_id)))
            rows = table.find_elements(By.TAG_NAME, 'tr')
            for row in rows[1:]:  # Skip the header row
                cells = row.find_elements(By.TAG_NAME, 'td')
                row_data = [cell.text.strip() for cell in cells]
                table_data.append(row_data)
            # Get the column names from the column_names_dict
            table_columns = column_names_dict.get(table_id, [])
            table_df = pd.DataFrame(table_data, columns=table_columns)
            # Clean and format dataframe based on table_id
            if table_id == 'roster':
                table_df = clean_and_format_dataframe_roster(table_df)
            elif table_id == 'per_game':
                table_df = clean_and_format_dataframe_per_game(table_df)
            elif table_id == 'totals':
                table_df = clean_and_format_dataframe_totals(table_df)
            elif table_id == 'team_and_opponent':
```

```
                table_df =␣
↪clean_and_format_dataframe_team_and_opponent(table_df)
            elif table_id == 'team_splits':
                table_df = clean_and_format_dataframe_team_splits(table_df)
            elif table_id == 'tgl_basic':
                table_df = clean_and_format_dataframe_tgl_basic(table_df)
            elif table_id == 'games':
                table_df = clean_and_format_dataframe_games(table_df)
            team_data[table_id] = table_df
            print(f"\n{table_id.capitalize().replace('_', ' ')}:")
            print(table_df)
            write_dataframes_to_excel({table_id: table_df}, team_name)
        return team_data

    except NoSuchElementException as e:
        logging.error(f"Element not found while scraping data for team␣
↪{team_name}: {e}")
        return None
    except TimeoutException as e:
        logging.error(f"Timeout while waiting for element while scraping data␣
↪for team {team_name}: {e}")
        return None
    except Exception as e:
        logging.error(f"Error scraping data for team {team_name}: {e}")
        return None
```

7.2 Define function to clean and format Roster dataframes

[60]:
```
def clean_and_format_dataframe_roster(df):
    # Convert columns to appropriate data types
    df['BirthDate'] = pd.to_datetime(df['BirthDate'])
    df['Weight'] = pd.to_numeric(df['Weight'], errors='coerce')

    # Clean strings: Remove leading and trailing whitespaces
    df['Player'] = df['Player'].str.strip()
    df['BirthCountry'] = df['BirthCountry'].str.strip()

    # Convert string columns to lowercase
    df['Position'] = df['Position'].str.lower()
    df['College'] = df['College'].str.lower()

    # Drop unnecessary columns
    df.drop(columns=['Experience'], inplace=True)

    return df
```

7.3 Define function to clean and format Per_Game dataframes

[61]:
```
def clean_and_format_dataframe_per_game(df):
    # Convert columns to appropriate data types
    df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
    df['GamesStarted'] = pd.to_numeric(df['GamesStarted'], errors='coerce')
    df['MinutesPlayedPerGame'] = pd.to_numeric(df['MinutesPlayedPerGame'],␣
↪errors='coerce')
    df['FieldGoalsPerGame'] = pd.to_numeric(df['FieldGoalsPerGame'],␣
↪errors='coerce')
    df['FieldGoalAttemptsPerGame'] = pd.
↪to_numeric(df['FieldGoalAttemptsPerGame'], errors='coerce')
    df['FieldGoalPercentPerGame'] = pd.
↪to_numeric(df['FieldGoalPercentPerGame'], errors='coerce')
    df['ThreePointFieldGoalsPerGame'] = pd.
↪to_numeric(df['ThreePointFieldGoalsPerGame'], errors='coerce')
    df['ThreePointFieldGoalAttemptsPerGame'] = pd.
↪to_numeric(df['ThreePointFieldGoalAttemptsPerGame'], errors='coerce')
    df['ThreePointFieldGoalPercentPerGame'] = pd.
↪to_numeric(df['ThreePointFieldGoalPercentPerGame'], errors='coerce')
    df['TwoPointFieldGoalsPerGame'] = pd.
↪to_numeric(df['TwoPointFieldGoalsPerGame'], errors='coerce')
    df['TwoPointFieldGoalAttemptsPerGame'] = pd.
↪to_numeric(df['TwoPointFieldGoalAttemptsPerGame'], errors='coerce')
    df['TwoPointFieldGoalPercentPerGame'] = pd.
↪to_numeric(df['TwoPointFieldGoalPercentPerGame'], errors='coerce')
    df['EffectiveFieldGoalPercentPerGame'] = pd.
↪to_numeric(df['EffectiveFieldGoalPercentPerGame'], errors='coerce')
    df['FreeThrowsPerGame'] = pd.to_numeric(df['FreeThrowsPerGame'],␣
↪errors='coerce')
    df['FreeThrowAttemptsPerGame'] = pd.
↪to_numeric(df['FreeThrowAttemptsPerGame'], errors='coerce')
    df['FreeThrowPercentPerGame'] = pd.
↪to_numeric(df['FreeThrowPercentPerGame'], errors='coerce')
    df['OffensiveReboundsPerGame'] = pd.
↪to_numeric(df['OffensiveReboundsPerGame'], errors='coerce')
    df['DefensiveReboundsPerGame'] = pd.
↪to_numeric(df['DefensiveReboundsPerGame'], errors='coerce')
    df['TotalReboundsPerGame'] = pd.to_numeric(df['TotalReboundsPerGame'],␣
↪errors='coerce')
    df['AssistsPerGame'] = pd.to_numeric(df['AssistsPerGame'], errors='coerce')
    df['StealsPerGame'] = pd.to_numeric(df['StealsPerGame'], errors='coerce')
    df['BlocksPerGame'] = pd.to_numeric(df['BlocksPerGame'], errors='coerce')
    df['TurnoversPerGame'] = pd.to_numeric(df['TurnoversPerGame'],␣
↪errors='coerce')
    df['PersonalFoulsPerGame'] = pd.to_numeric(df['PersonalFoulsPerGame'],␣
↪errors='coerce')
    df['PointsPerGame'] = pd.to_numeric(df['PointsPerGame'], errors='coerce')
```

```
    # Clean strings: Remove leading and trailing whitespaces
    df['Player'] = df['Player'].str.strip()

    # Convert string columns to lowercase
    df['Player'] = df['Player'].str.lower()

    return df
```

7.4 Define function to clean and format Totals dataframes

[62]:
```
def clean_and_format_dataframe_totals(df):
    # Convert columns to appropriate data types
    df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
    df['TotalGamesPlayed'] = pd.to_numeric(df['TotalGamesPlayed'],␣
↪errors='coerce')
    df['TotalGamesStarted'] = pd.to_numeric(df['TotalGamesStarted'],␣
↪errors='coerce')
    df['TotalMinutesPlayed'] = pd.to_numeric(df['TotalMinutesPlayed'],␣
↪errors='coerce')
    df['TotalFieldGoalsPerGame'] = pd.to_numeric(df['TotalFieldGoalsPerGame'],␣
↪errors='coerce')
    df['TotalFieldGoalAttempts'] = pd.to_numeric(df['TotalFieldGoalAttempts'],␣
↪errors='coerce')
    df['TotalFieldGoalPercent'] = pd.to_numeric(df['TotalFieldGoalPercent'],␣
↪errors='coerce')
    df['TotalThreePointFieldGoalsPerGame'] = pd.
↪to_numeric(df['TotalThreePointFieldGoalsPerGame'], errors='coerce')
    df['TotalThreePointFieldGoalAttempts'] = pd.
↪to_numeric(df['TotalThreePointFieldGoalAttempts'], errors='coerce')
    df['TotalThreePointFieldGoalPercent'] = pd.
↪to_numeric(df['TotalThreePointFieldGoalPercent'], errors='coerce')
    df['TotalTwoPointFieldGoalsPerGame'] = pd.
↪to_numeric(df['TotalTwoPointFieldGoalsPerGame'], errors='coerce')
    df['TotalTwoPointFieldGoalAttempts'] = pd.
↪to_numeric(df['TotalTwoPointFieldGoalAttempts'], errors='coerce')
    df['TotalTwoPointFieldGoalPercent'] = pd.
↪to_numeric(df['TotalTwoPointFieldGoalPercent'], errors='coerce')
    df['TotalEffectiveFieldGoalPercent'] = pd.
↪to_numeric(df['TotalEffectiveFieldGoalPercent'], errors='coerce')
    df['TotalFreeThrows'] = pd.to_numeric(df['TotalFreeThrows'],␣
↪errors='coerce')
    df['TotalFreeThrowAttempts'] = pd.to_numeric(df['TotalFreeThrowAttempts'],␣
↪errors='coerce')
    df['TotalFreeThrowPercent'] = pd.to_numeric(df['TotalFreeThrowPercent'],␣
↪errors='coerce')
```

```
    df['TotalOffensiveRebounds'] = pd.to_numeric(df['TotalOffensiveRebounds'],␣
↪errors='coerce')
    df['TotalDefensiveRebounds'] = pd.to_numeric(df['TotalDefensiveRebounds'],␣
↪errors='coerce')
    df['TotalRebounds'] = pd.to_numeric(df['TotalRebounds'], errors='coerce')
    df['TotalAssists'] = pd.to_numeric(df['TotalAssists'], errors='coerce')
    df['TotalSteals'] = pd.to_numeric(df['TotalSteals'], errors='coerce')
    df['TotalBlocks'] = pd.to_numeric(df['TotalBlocks'], errors='coerce')
    df['TotalTurnovers'] = pd.to_numeric(df['TotalTurnovers'], errors='coerce')
    df['TotalPersonalFouls'] = pd.to_numeric(df['TotalPersonalFouls'],␣
↪errors='coerce')
    df['TotalPoints'] = pd.to_numeric(df['TotalPoints'], errors='coerce')

    # Clean strings: Remove leading and trailing whitespaces
    df['Player'] = df['Player'].str.strip()

    # Convert string columns to lowercase
    df['Player'] = df['Player'].str.lower()

    return df
```

7.5 Define function to clean and format Starting_Lineups dataframes

[63]:
```
def clean_and_format_dataframe_starting_lineups_po0(df):
    # Convert columns to appropriate data types
    df['Date'] = pd.to_datetime(df['Date'])
    df['TeamPoints'] = pd.to_numeric(df['TeamPoints'], errors='coerce')
    df['OpponentPoints'] = pd.to_numeric(df['OpponentPoints'], errors='coerce')
    df['Wins'] = pd.to_numeric(df['Wins'], errors='coerce')
    df['Losses'] = pd.to_numeric(df['Losses'], errors='coerce')

    # Clean strings: Remove leading and trailing whitespaces
    df['Opponent'] = df['Opponent'].str.strip()
    df['StartingLineup'] = df['StartingLineup'].str.strip()

    # Convert string columns to lowercase
    df['Opponent'] = df['Opponent'].str.lower()
    df['StartingLineup'] = df['StartingLineup'].str.lower()

    # Drop unnecessary columns
    df.drop(columns=['Start(ET)'], inplace=True)
    df.drop(columns=['_'], inplace=True)
    df.drop(columns=['BoxScore'], inplace=True)

    return df
```

7.6 Define function to clean and format Team Splits dataframes

[64]:
```
def clean_and_format_dataframe_team_splits(df):
    # Convert columns to appropriate data types
    df['Games'] = pd.to_numeric(df['Games'], errors='coerce')
    df['Wins'] = pd.to_numeric(df['Wins'], errors='coerce')
    df['Losses'] = pd.to_numeric(df['Losses'], errors='coerce')
    df['FieldGoals'] = pd.to_numeric(df['FieldGoals'], errors='coerce')
    df['FieldGoalAttempts'] = pd.to_numeric(df['FieldGoalAttempts'],␣
 ↪errors='coerce')
    df['ThreePoints'] = pd.to_numeric(df['ThreePoints'], errors='coerce')
    df['ThreePointAttempts'] = pd.to_numeric(df['ThreePointAttempts'],␣
 ↪errors='coerce')
    df['FreeThrows'] = pd.to_numeric(df['FreeThrows'], errors='coerce')
    df['FreeThrowAttempts'] = pd.to_numeric(df['FreeThrowAttempts'],␣
 ↪errors='coerce')
    df['OffensiveRebounds'] = pd.to_numeric(df['OffensiveRebounds'],␣
 ↪errors='coerce')
    df['TotalRebounds'] = pd.to_numeric(df['TotalRebounds'], errors='coerce')
    df['Assists'] = pd.to_numeric(df['Assists'], errors='coerce')
    df['Steals'] = pd.to_numeric(df['Steals'], errors='coerce')
    df['Blocks'] = pd.to_numeric(df['Blocks'], errors='coerce')
    df['Turnovers'] = pd.to_numeric(df['Turnovers'], errors='coerce')
    df['PersonalFouls'] = pd.to_numeric(df['PersonalFouls'], errors='coerce')
    df['Points'] = pd.to_numeric(df['Points'], errors='coerce')
    df['FieldGoalPercent'] = pd.to_numeric(df['FieldGoalPercent'],␣
 ↪errors='coerce')
    df['ThreePointPercent'] = pd.to_numeric(df['ThreePointPercent'],␣
 ↪errors='coerce')
    df['EffectiveFieldGoalPercent'] = pd.
 ↪to_numeric(df['EffectiveFieldGoalPercent'], errors='coerce')
    df['FreeThrowPercent'] = pd.to_numeric(df['FreeThrowPercent'],␣
 ↪errors='coerce')
    df['TSPercent'] = pd.to_numeric(df['TSPercent'], errors='coerce')
    df['OppFieldGoals'] = pd.to_numeric(df['OppFieldGoals'], errors='coerce')
    df['OppFieldGoalAttempts'] = pd.to_numeric(df['OppFieldGoalAttempts'],␣
 ↪errors='coerce')
    df['OppThreePoints'] = pd.to_numeric(df['OppThreePoints'], errors='coerce')
    df['OppThreePointAttempts'] = pd.to_numeric(df['OppThreePointAttempts'],␣
 ↪errors='coerce')
    df['OppFreeThrows'] = pd.to_numeric(df['OppFreeThrows'], errors='coerce')
    df['OppFreeThrowAttempts'] = pd.to_numeric(df['OppFreeThrowAttempts'],␣
 ↪errors='coerce')
    df['OppOffensiveRebounds'] = pd.to_numeric(df['OppOffensiveRebounds'],␣
 ↪errors='coerce')
    df['OppTotalRebounds'] = pd.to_numeric(df['OppTotalRebounds'],␣
 ↪errors='coerce')
    df['OppAssists'] = pd.to_numeric(df['OppAssists'], errors='coerce')
```

```
    df['OppSteals'] = pd.to_numeric(df['OppSteals'], errors='coerce')
    df['OppBlocks'] = pd.to_numeric(df['OppBlocks'], errors='coerce')
    df['OppTurnovers'] = pd.to_numeric(df['OppTurnovers'], errors='coerce')
    df['OppPersonalFouls'] = pd.to_numeric(df['OppPersonalFouls'],␣
↪errors='coerce')
    df['OppPoints'] = pd.to_numeric(df['OppPoints'], errors='coerce')
    df['OppFieldGoalPercent'] = pd.to_numeric(df['OppFieldGoalPercent'],␣
↪errors='coerce')
    df['OppThreePointPercent'] = pd.to_numeric(df['OppThreePointPercent'],␣
↪errors='coerce')
    df['OppFreeThrowPercent'] = pd.to_numeric(df['OppFreeThrowPercent'],␣
↪errors='coerce')
    df['OppTSPercent'] = pd.to_numeric(df['OppTSPercent'], errors='coerce')

    # Clean strings: Remove leading and trailing whitespaces
    df['Value'] = df['Value'].str.strip()

    # Convert string columns to lowercase
    df['Value'] = df['Value'].str.lower()

    return df
```

7.7 Define function to clean and format Game_Log dataframes

[65]:
```
def clean_and_format_dataframe_tgl_basic(df):
    # Convert columns to appropriate data types
    df['Date'] = pd.to_datetime(df['Date'])
    df['Points'] = pd.to_numeric(df['Points'], errors='coerce')
    df['OppPoints'] = pd.to_numeric(df['OppPoints'], errors='coerce')
    df['FieldGoals'] = pd.to_numeric(df['FieldGoals'], errors='coerce')
    df['FieldGoalAttempts'] = pd.to_numeric(df['FieldGoalAttempts'],␣
↪errors='coerce')
    df['FieldGoalPercent'] = pd.to_numeric(df['FieldGoalPercent'],␣
↪errors='coerce')
    df['ThreePoints'] = pd.to_numeric(df['ThreePoints'], errors='coerce')
    df['ThreePointAttempts'] = pd.to_numeric(df['ThreePointAttempts'],␣
↪errors='coerce')
    df['ThreePointPercent'] = pd.to_numeric(df['ThreePointPercent'],␣
↪errors='coerce')
    df['FreeThrows'] = pd.to_numeric(df['FreeThrows'], errors='coerce')
    df['FreeThrowAttempts'] = pd.to_numeric(df['FreeThrowAttempts'],␣
↪errors='coerce')
    df['FreeThrowPercent'] = pd.to_numeric(df['FreeThrowPercent'],␣
↪errors='coerce')
    df['OffensiveRebounds'] = pd.to_numeric(df['OffensiveRebounds'],␣
↪errors='coerce')
    df['TotalRebounds'] = pd.to_numeric(df['TotalRebounds'], errors='coerce')
```

```
    df['Assists'] = pd.to_numeric(df['Assists'], errors='coerce')
    df['Steals'] = pd.to_numeric(df['Steals'], errors='coerce')
    df['Blocks'] = pd.to_numeric(df['Blocks'], errors='coerce')
    df['Turnovers'] = pd.to_numeric(df['Turnovers'], errors='coerce')
    df['PersonalFouls'] = pd.to_numeric(df['PersonalFouls'], errors='coerce')
    df['OppFieldGoals'] = pd.to_numeric(df['OppFieldGoals'], errors='coerce')
    df['OppFieldGoalAttempts'] = pd.to_numeric(df['OppFieldGoalAttempts'],␣
↪errors='coerce')
    df['OppFieldGoalPercent'] = pd.to_numeric(df['OppFieldGoalPercent'],␣
↪errors='coerce')
    df['OppThreePoints'] = pd.to_numeric(df['OppThreePoints'], errors='coerce')
    df['OppThreePointAttempts'] = pd.to_numeric(df['OppThreePointAttempts'],␣
↪errors='coerce')
    df['OppThreePointPercent'] = pd.to_numeric(df['OppThreePointPercent'],␣
↪errors='coerce')
    df['OppFreeThrows'] = pd.to_numeric(df['OppFreeThrows'], errors='coerce')
    df['OppFreeThrowAttempts'] = pd.to_numeric(df['OppFreeThrowAttempts'],␣
↪errors='coerce')
    df['OppFreeThrowPercent'] = pd.to_numeric(df['OppFreeThrowPercent'],␣
↪errors='coerce')
    df['OppOffensiveRebounds'] = pd.to_numeric(df['OppOffensiveRebounds'],␣
↪errors='coerce')
    df['OppTotalRebounds'] = pd.to_numeric(df['OppTotalRebounds'],␣
↪errors='coerce')
    df['OppAssists'] = pd.to_numeric(df['OppAssists'], errors='coerce')
    df['OppSteals'] = pd.to_numeric(df['OppSteals'], errors='coerce')
    df['OppBlocks'] = pd.to_numeric(df['OppBlocks'], errors='coerce')
    df['OppTurnovers'] = pd.to_numeric(df['OppTurnovers'], errors='coerce')
    df['OppPersonalFouls'] = pd.to_numeric(df['OppPersonalFouls'],␣
↪errors='coerce')

    # Clean strings: Remove leading and trailing whitespaces
    df['Opp'] = df['Opp'].str.strip()

    # Convert string columns to lowercase
    df['Opp'] = df['Opp'].str.lower()

    # Drop unnecessary columns
    df.drop(columns=['_1'], inplace=True)
    df.drop(columns=['HomeAway'], inplace=True)
    df.drop(columns=['_2'], inplace=True)

    return df
```

7.8 Define function to clean and format Results dataframes

[66]:
```
def clean_and_format_dataframe_games(df):
    # Convert columns to appropriate data types
    df['Date'] = pd.to_datetime(df['Date'])
    df['Points'] = pd.to_numeric(df['Points'], errors='coerce')
    df['OppPoints'] = pd.to_numeric(df['OppPoints'], errors='coerce')

    # Clean strings: Remove leading and trailing whitespaces
    df['Opponent'] = df['Opponent'].str.strip()
    df['Overtime'] = df['Overtime'].str.strip()

    # Convert string columns to lowercase
    df['Opponent'] = df['Opponent'].str.lower()
    df['Overtime'] = df['Overtime'].str.lower()

    # Drop unnecessary columns
    df.drop(columns=['_'], inplace=True)
    df.drop(columns=['Start(ET)'], inplace=True)
    df.drop(columns=['BoxScore'], inplace=True)
    df.drop(columns=['Wins'], inplace=True)
    df.drop(columns=['Losses'], inplace=True)
    df.drop(columns=['Streak'], inplace=True)
    df.drop(columns=['Notes'], inplace=True)

    return df
```

8 Max Worker Threadpool

8.1 Set number of Max Workers to use for scraping

[67]:
```
max_workers = 5
```

8.2 Create a ThreadPoolExecutor instance

[68]:
```
with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers) as executor:
    # Submit scraping tasks for each team URL
    scraping_tasks = {executor.submit(scrape_team_data, url, team_name,␣
 ↪table_ids): (team_name, url) for team_name, url in team_url.items()}
    # Iterate over completed tasks and get the results
    for future in concurrent.futures.as_completed(scraping_tasks):
        team_name, url = scraping_tasks[future]
        try:
            # Get the result of the scraping task
            result = future.result()
            print(result)
        except Exception as e:
            print(f"Failed to scrape data for {team_name}: {e}")
```

```
None
None
```

```
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
```

9 Create Exel Files

9.1 Define a Function to write the dataframes into Excel files

[69]:
```
def write_dataframes_to_excel(team_data, team_name):
    try:
        excel_file_path = os.path.join(excel_directory, f"{team_name}_data.
↪xlsx")

        # Check if the file exists
        if os.path.isfile(excel_file_path):
            # Open the existing Excel file
            with pd.ExcelWriter(excel_file_path, engine='openpyxl', mode='a')␣
↪as writer:
                for table_id, table_df in team_data.items():
                    if table_df is not None:
                        # Check if the sheet exists and replace it if it does
                        if table_id in writer.book.sheetnames:
                            writer.book.remove(writer.book[table_id])
```

```
                        sheet_name = f"{table_id}"
                        table_df.to_excel(writer, sheet_name=sheet_name,␣
↪index=False)
        else:
            # Create a new Excel file using xlsxwriter engine
            with pd.ExcelWriter(excel_file_path, engine='xlsxwriter', mode='w')␣
↪as writer:
                for table_id, table_df in team_data.items():
                    if table_df is not None:
                        sheet_name = f"{table_id}"
                        table_df.to_excel(writer, sheet_name=sheet_name,␣
↪index=False)
        print(f"Appended data to sheets for {team_name}")
    except Exception as e:
        logging.error(f"Error writing data to sheets for team {team_name}: {e}")
```

10 Start Web Scrape Process

10.1 Define function to process to Loop through each starting URL template and scrape data for each team

[70]:
```
for category, url_template in page_url_template.items():
    try:
        # Loop through each team and scrape data for the specific category
        for team, team_url_value in team_url.items():  # Changed variable name␣
↪to avoid conflict
            try:
                # Construct the URL for the specific category
                url = url_template.format(start=start_url, link=team)

                # Scrape data for the team and specific category
                team_data = scrape_team_data(url, team, table_ids[category])

                if team_data:
                    # After scraping for all teams, write the dataframes to␣
↪Excel files
                    write_dataframes_to_excel(team_data, team)

            except Exception as e:
                logging.error(f"Error processing team '{team}' for category␣
↪'{category}': {e}")
                continue

    except Exception as e:
        logging.error(f"Error processing category '{category}': {e}")
        continue
```

```
Roster:
               Player Position Height  Weight  BirthDate BirthCountry  \
0          Saddiq Bey       sf    6-7     215 1999-04-09           us
1   Bogdan Bogdanović       sg    6-5     220 1992-08-18           rs
2        Clint Capela        c   6-10     240 1994-05-18           ch
3        John Collins       pf    6-9     235 1997-09-23           us
4      Jarrett Culver       sf    6-6     195 1999-02-20           us
5      Bruno Fernando        c    6-9     240 1998-08-15           ao
6       Trent Forrest       sg    6-4     210 1998-06-12           us
7          AJ Griffin       sf    6-6     222 2003-08-25           us
8       Aaron Holiday       pg    6-0     185 1996-09-30           us
9      Justin Holiday       sf    6-6     180 1989-04-05           us
10    De'Andre Hunter       sf    6-8     225 1997-12-02           us
11      Jalen Johnson       sf    6-9     220 2001-12-18           us
12     Frank Kaminsky        c    7-0     240 1993-04-04           us
13         Vit Krejci       pg    6-8     195 2000-06-19           cz
14      Tyrese Martin       sg    6-6     215 1999-03-07           us
15   Garrison Mathews       sg    6-5     215 1996-10-24           us
16    Dejounte Murray       sg    6-5     180 1996-09-19           us
17     Onyeka Okongwu        c    6-8     235 2000-12-11           us
18   Donovan Williams       sg    6-6     190 2001-09-06           us
19         Trae Young       pg    6-1     164 1998-09-19           us

                College
0             villanova
1
2
3           wake forest
4            texas tech
5              maryland
6         florida state
7                  duke
8                  ucla
9            washington
10             virginia
11                 duke
12            wisconsin
13
14  rhode island, uconn
15              lipscomb
16           washington
17                 usc
18          texas, unlv
19             oklahoma
Appended data to sheets for ATL

Per game:
               Player  Age GamesPlayedPerGame  GamesStarted  ...
```

```
85      114.0

... [86 rows x 7 columns]
Appended data to sheets for WAS
Appended data to sheets for WAS
```

10.2 Close the webdriver

[71]:
```
driver.quit()
```

11 Conclusion

11.1 ETL

11.1.1 In this overview of the Extract portion of my NBA Data pipeline. We have discussed how to Import Required modules, Determine what ifnormation is going to be scraped, Confirm the setup, Start the Chrome WebDriver, Run a Final Multi-layered Check, Define Functions, Create Exel Files and how to Start the actual Web Scrape Process.

11.2 Portforlio Project

11.2.1 Links to the entire Basketball Data Project including the RStudio Data Transformation Code and Tableau Dashboards can be found on my GitHub Profile