DiD

We are aiming to understand the impact of NIL deal on player performance by doing an DiD analysis between U.S. and Canada in the 2021/2022 season. Players included in this analysis should fit in all below criterias : (1) Players who attend in both seasons (2019/2020 & 2021/2022) (2) Include regular games only, exclude playoff season games

The metric we'll be using to measure performance is Winshare_40, and we'll be excluding Winshare value that shows inf.

In [1]:
```
import pandas as pd
df = pd.read_csv("menWithRace.csv",low_memory=False)
df.head(10)
df = df.rename(columns={"WS.40": "WS_40"})
deal = pd.read_csv("Deal.csv")
```

In [2]:
```
 # filter players who played in both season in regular games
## step 1 :filter players who played in regular games in 2019/2020 season - games between 2019/11/04 ~ 2020/03/12
df['game_date'] = pd.to_datetime(df['game_date'])
start = '2019-11-02'
end = '2020-03-12'

filter_2019 = df[(df['game_date'] >= start) & (df['game_date'] <= end)]
filter_2019.head()

## step 2 :filter players who played in regular games in 2021/2022 season - games between 2021/11/09 ~ 2022/03/06
df['game_date'] = pd.to_datetime(df['game_date'])
newstart = '2021-11-09'
newend = '2022-03-06'

filter_2020 = df[(df['game_date'] >= newstart) & (df['game_date'] <= newend)]
filter_2020.head()

## step3 : find players who played in both seasons
player_2019 = set(filter_2019['athlete_id'].unique())
player_2020 = set(filter_2020['athlete_id'].unique())

common_players = player_2019.intersection(player_2020)
both_season_df = df[df['athlete_id'].isin(common_players)]

both_season_df.head()

# count amounts - 2382
# len(common_players)
```

In [5]:
```
# seperate U.S. and Canada players
## canada players df
canadian_df = both_season_df[(both_season_df['athlete_id'].isin(common_players) & both_season_df['treatment']==0)]
canadian_df
# check Canada player num - 306
#canadian_df['athlete_id'].nunique()

## us players df
us_df = both_season_df[(both_season_df['athlete_id'].isin(common_players) & both_season_df['treatment']==1)].copy()
us_df
# check us player num - 2076
#us_df['athlete_id'].nunique()
```

In [11]:
```
from linearmodels.panel import PanelOLS
import pandas as pd
import numpy as np
```

In [12]:
```
# let's do DiD to see the impact!

# combine us and canadian player data first
did_df = pd.concat([canadian_df,us_df], ignore_index = True)

# drop nan
did_df = did_df.dropna()
did_df["WS_40"] = pd.to_numeric(did_df["WS_40"], errors="coerce")
did_df=did_df[np.isfinite(did_df['WS_40'])]
# create panel_indexer
did_df=did_df.set_index(['athlete_id','game_date'])

# period - 0 is pre-NIL, 1 is post-NIL
did_df['post']=did_df['period']

# run DiD
model = PanelOLS.from_formula('WS_40 ~ treatment:post + EntityEffects', data = did_df).fit()

# check result
print(model.summary)
```

```
                          PanelOLS Estimation Summary
================================================================================
Dep. Variable:                   WS_40   R-squared:                        0.0003
Estimator:                    PanelOLS   R-squared (Between):              0.0319
No. Observations:                66409   R-squared (Within):               0.0003
Date:                Sat, Apr 19 2025   R-squared (Overall):              0.0150
Time:                        23:09:43   Log-likelihood                    5851.0
Cov. Estimator:            Unadjusted
                                        F-statistic:                      20.844
Entities:                        2494   P-value                           0.0000
Avg Obs:                       26.628   Distribution:                F(1,63914)
Min Obs:                       2.0000
Max Obs:                       40.000   F-statistic (robust):             20.844
                                        P-value                           0.0000
Time periods:                    45     Distribution:                F(1,63914)
Avg Obs:                       1475.8
Min Obs:                       10.0000
Max Obs:                       2149.0

                             Parameter Estimates
================================================================================
              Parameter  Std. Err.     T-stat    P-value    Lower CI    Upper CI
--------------------------------------------------------------------------------
treatment:post     0.0085     0.0019     4.5655     0.0000      0.0049      0.0122
================================================================================

F-test for Poolability: 4.1797
P-value: 0.0000
Distribution: F(2493,63914)

Included effects: Entity
```

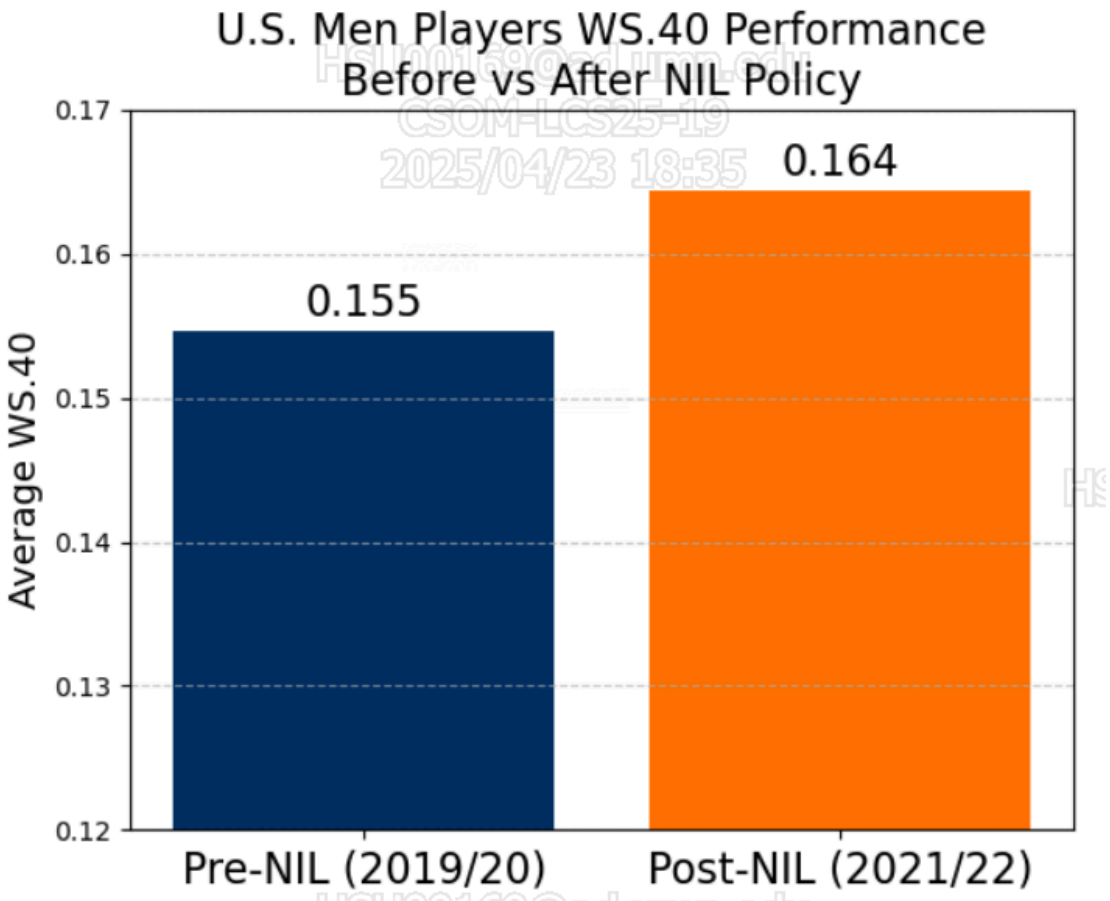
Observation : After the NIL policy took effect, Men U.S. players' WS_40 increased by an average of 0.0085 points more than Men Canadian players, after controlling for all player-specific effects. This result is statistically significant (p < 0.000), with a tight 95% confidence interval ([0.0049, 0.0122]), meaning the effect is real and not due to chance.

Indications : After the NIL policy, U.S. players did improve in performance compare to themselves and compare to Canadian players — but not as much as expected, suggesting that NIL may have had a positive impact on U.S. athletes' relative progress.

In [15]:
```
us_df = us_df.dropna()
us_df["WS_40"] = pd.to_numeric(us_df["WS_40"], errors="coerce")
us_df=us_df[np.isfinite(us_df['WS_40'])]
canadian_df = canadian_df.dropna()
canadian_df["WS_40"] = pd.to_numeric(canadian_df["WS_40"], errors="coerce")
canadian_df=canadian_df[np.isfinite(canadian_df['WS_40'])]
# 美國
us_avg_period_1 = us_df[us_df['period'] == 1]['WS_40'].mean()
us_avg_period_0 = us_df[us_df['period'] == 0]['WS_40'].mean()

# 加拿大
canada_avg_period_1 = canadian_df[canadian_df['period'] == 1]['WS_40'].mean()
canada_avg_period_0 = canadian_df[canadian_df['period'] == 0]['WS_40'].mean()
print("US average (Period == 1):", us_avg_period_1)
print("US average (Period == 0):", us_avg_period_0)
print("Canada average (Period == 1):", canada_avg_period_1)
print("Canada average (Period == 0):", canada_avg_period_0)
```

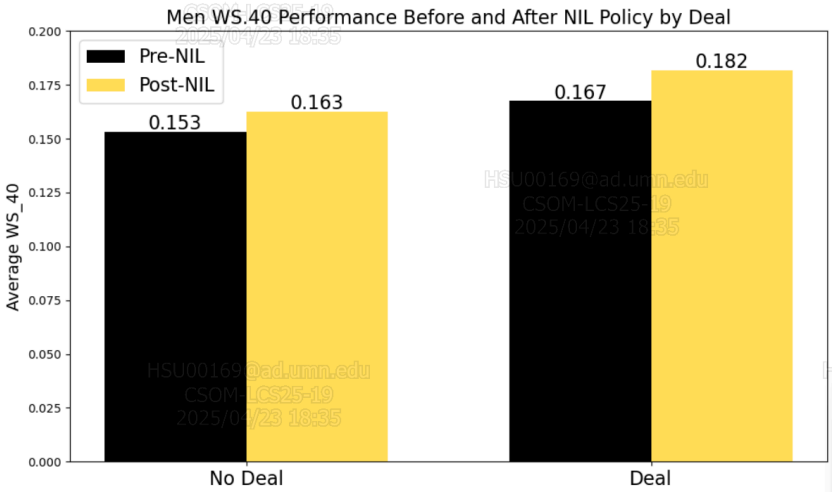
```
US average (Period == 1): 0.1644129383971268
US average (Period == 0): 0.15464774807763748
Canada average (Period == 1): 0.153094375840712
Canada average (Period == 0): 0.14635882456638044
```

In [25]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Step 1: Calculate average WS.40 pre/post NIL for U.S. players
us_pre_mean = us_pre.mean()
us_post_mean = us_post.mean()

# Step 2: Create a simple DataFrame to plot
plot_df = pd.DataFrame({
    'Period': ['Pre-NIL (2019/20)', 'Post-NIL (2021/22)'],
    'WS.40': [us_pre_mean, us_post_mean]
})

# Step 3: Plot
plt.figure(figsize=(6, 5))
bars = plt.bar(plot_df['Period'], plot_df['WS.40'], color=['#002d62', '#ff6f00'])

# Add value labels above bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.001, f'{yval:.3f}', ha='center', fontsize=16)

# Labels and title
plt.title('U.S. Men Players WS.40 Performance\nBefore vs After NIL Policy', fontsize=16)
plt.ylabel('Average WS.40', fontsize=14)
plt.ylim(0.12, 0.17)
plt.xticks(fontsize=16)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```


U.S. Men Players WS.40 Performance
Before vs After NIL Policy
Average WS.40
0.17
0.16
0.15
0.14
0.13
0.12
0.155
0.164
Pre-NIL (2019/20)
Post-NIL (2021/22)

EDA

In [33]:
```python
#same as above
import matplotlib.pyplot as plt

# 建立一個 race 列表
deals = [0, 1]

# 建立一個空的結果列表
means_pre = []
means_post = []

# 逐個 race 計算 NIL 政策前後的平均 WS_40
for deal in deals:
    # pre-NIL
    pre = did_df[(did_df['deal'] == deal) & (did_df['post'] == 0)]['WS_40'].mean()
    means_pre.append(pre)

    # post-NIL
    post = did_df[(did_df['deal'] == deal) & (did_df['post'] == 1)]['WS_40'].mean()
    means_post.append(post)

# 繪圖
x = range(len(deals))
bar_width = 0.35

plt.figure(figsize=(10,6))
#plt.bar([i - bar_width/2 for i in x], means_pre, width=bar_width, label='Pre-NIL', color='#002d62')
#plt.bar([i + bar_width/2 for i in x], means_post, width=bar_width, label='Post-NIL', color='#ff6f00')

bars1 = plt.bar([i - bar_width/2 for i in x], means_pre, width=bar_width, label='Pre-NIL', color='black')
bars2 = plt.bar([i + bar_width/2 for i in x], means_post, width=bar_width, label='Post-NIL', color='#ffde59')

# 加數字在每個柱子上方
for bar in bars1 + bars2:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 0.001, f'{height:.3f}', ha='center', fontsize=16)

# 設定圖表
plt.ylabel('Average WS_40', fontsize=14)
plt.title('Men WS.40 Performance Before and After NIL Policy by Deal', fontsize=16)
#plt.xticks(x, deals, fontsize=16)
plt.ylim(0, 0.2)
plt.xticks([0, 1], ['No Deal', 'Deal'], fontsize=16)
plt.legend(fontsize=16)
plt.tight_layout()
plt.show()
```



Web Scraping

In [1]:

```
'''
from bs4 import BeautifulSoup
import pandas as pd
import requests
import time

headers = []
rows = []

# Step 1: Load only basketball players from the main table
for page_num in range(1, 423):
    print(f"Scraping {page_num}...")
    url = f"https://nilcollegeathletes.com/athletes?page={page_num}"
    response = requests.get(url)
    soup = BeautifulSoup(response.content, "html.parser")

    if not headers:
        for header in soup.find_all("th"):
            headers.append(header.text.strip())

    for row in soup.find_all("tr")[1:]:
        row_data = []
        for cell in row.find_all("td"):
            row_data.append(cell.text.strip())

        # Filter Sport = basketball
        if len(row_data) >= 3 and "basketball" in row_data[2].lower():
            rows.append(row_data)

    time.sleep(0.2)
```

```
# Step 2: Create dataframe with filtered basketball data
df = pd.DataFrame(rows, columns=headers)
df = df[["Name", "University", "Sport"]]  # Keep only needed columns

# Step 3: Crawl sponsor info only
athlete_links = []
sponsors_list = []

for idx, name in enumerate(df["Name"]):
    try:
        print(f"Fetching player {idx+1}/{len(df)}: {name}")
        name_parts = name.lower().split()
        if len(name_parts) < 2:
            athlete_links.append("")
            sponsors_list.append([])
            continue

        athlete_link = f"https://nilcollegeathletes.com/athletes/{name_parts[0]}-{name_parts[1]}"
        athlete_links.append(athlete_link)

        response = requests.get(athlete_link)
        soup = BeautifulSoup(response.content, "html.parser")
        sponsors = []

        for ul in soup.find_all("ul", class_="space-y-1"):
            for s in ul.find_all("a"):
                sponsors.append(s.text.strip())

        sponsors_list.append(sponsors)

    except Exception as e:
        print(f"Fail: {name}, Error: {e}")
        athlete_links.append("")
        sponsors_list.append([])
    time.sleep(0.2)

# Step 4: Final assembly
df["Sponsor"] = sponsors_list
df_final = df.explode("Sponsor").reset_index(drop=True)
df_final = df_final[["Name", "University", "Sport", "Sponsor"]]  # only needed columns

# Done
df_final.head()
'''
```

Out[1]:
```
'\nfrom bs4 import BeautifulSoup\nimport pandas as pd\nimport requests\nimport time\n\nheaders = []\nrows = []\n\n# Step 1: Load only basketball players from the main table\nfor page_num in range(1, 423): \n    print(f"Scraping {page_num}...")\n    url = f"https://nilcollegeathletes.com/athletes?page={page_num}"\n    response = requests.get(url)\n    soup = BeautifulSoup(response.content, "html.parser")\n\n    if not headers:\n        for header in soup.find_all("th"):\n            headers.append(header.text.strip())\n\n    for row in soup.find_all("tr")[1:]:\n        row_data = []\n        for cell in row.find_all("td"):\n            row_data.append(cell.text.strip())\n\n        # Filter Sport = basketball\n        if len(row_data) >= 3 and "basketball" in row_data[2].lower():\n            rows.append(row_data)\n\n    time.sleep(0.2)\n\n# Step 2: Create dataframe with filtered basketball data\ndf = pd.DataFrame(rows, columns=headers)\ndf = df[["Name", "University", "Sport"]]  # Keep only needed columns\n\n# Step 3: Crawl sponsor info only\nathlete_links = []\nsponsors_list = []\n\nfor idx, name in enumerate(df["Name"]):\n    try:\n        print(f"Fetching player {idx+1}/{len(df)}: {name}")\n        name_parts = name.lower().split()\n        if len(name_parts) < 2:\n            athlete_links.append("")\n            sponsors_list.append([])\n            continue\n\n        athlete_link = f"https://nilcollegeathletes.com/athletes/{name_parts[0]}-{name_parts[1]}"\n        athlete_links.append(athlete_link)\n\n        response = requests.get(athlete_link)\n        soup = BeautifulSoup(response.content, "html.parser")\n        sponsors = []\n\n        for ul in soup.find_all("ul", class_="space-y-1"):\n            for s in ul.find_all("a"):\n                sponsors.append(s.text.strip())\n\n        sponsors_list.append(sponsors)\n\n    except Exception as e:\n        print(f"Fail: {name}, Error: {e}")\n        athlete_links.append("")\n        sponsors_list.append([])\n\n    time.sleep(0.2)\n\n# Step 4: Final assembly\ndf["Sponsor"] = sponsors_list\ndf_final = df.explode("Sponsor").reset_index(drop=True)\ndf_final = df_final[["Name", "University", "Sport", "Sponsor"]]  # only needed columns\n\n# Done\ndf_final.head()\n'
```

In [15]:
```
df_final2 = df_final
df_new = (df_final2.groupby(['Name', 'University', 'Sport'])['Sponsor'].nunique().reset_index(name='NIL_Partner_Count'))
df_new.head()
```

Out[15]:

	Name	University	Sport	NIL_Partner_Count
0	AJ Hoynack	High Point	Basketball	1
1	Aaliyah Moore	Texas	Basketball	0
2	Abby Carter	Akron	Basketball	1
3	Abby Wahl	Eastern Illinois	Basketball	1
4	Abdoulaye Thiam	Minnesota	Basketball	1

In [17]:
```
# Save to CSV
df_new.to_csv("Deal.csv", index = False)
```

In [21]:
```
# 建立一個 mapping，取得每個 athlete_id 第一次出現時的 display name
id_to_name = us_df.drop_duplicates('athlete_id')[['athlete_id', 'athlete_display_name']]
id_to_name_dict = dict(zip(id_to_name['athlete_id'], id_to_name['athlete_display_name']))

# 用 mapping 替換整個欄位
us_df['athlete_display_name'] = us_df['athlete_id'].map(id_to_name_dict)
```

In [23]:
```
# Append Deal information for each player
import re

# 1) 先清理 deal 裡的 Name
cleaned_deal_names = (
    deal['Name']
    .astype(str)
    .str.lower()
    .str.replace(r'[^\w\s]', '', regex=True)  # 去掉所有標點
    .str.strip()
)

# 2) 在 us_df 上做同樣的清理，並判斷是否在 cleaned_deal_names 裡
us_df['deal'] = (
    us_df['athlete_display_name']
    .astype(str)
    .str.lower()
    .str.replace(r'[^\w\s]', '', regex=True)
    .str.strip()
    .isin(cleaned_deal_names)
    .astype(int)   # 轉成 0/1
)
```

Random Forest

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report

# Step 1: Split features and target
X = df.drop(columns=['deal'])
y = df['deal']

# Step 2: Train-test split (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Step 3: Define column types
numeric_cols = X.select_dtypes(include=['float64', 'int64', 'int32']).columns.tolist()
categorical_cols = X.select_dtypes(include=['category']).columns.tolist()

# Step 4: Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_cols),
        ('cat', OneHotEncoder(drop='first', handle_unknown='ignore'), categorical_cols)
    ]
)
```

```python
# Step 5: Create the full pipeline with Random Forest
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('clf', RandomForestClassifier(random_state=42))
])

# Step 6: Define hyperparameter grid
param_grid = {
    'clf__n_estimators': [100, 200],
    'clf__max_depth': [None, 10, 20],
    'clf__min_samples_split': [2, 5],
}

# Step 7: Grid Search with Cross Validation
grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=5,
    scoring='roc_auc',  # You can change to 'accuracy' if preferred
    n_jobs=-1
)

# Step 8: Fit the model
grid_search.fit(X_train, y_train)

# Step 9: Evaluate on test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
y_proba = best_model.predict_proba(X_test)[:, 1]  # for AUC

# Step 10: Print performance metrics
print("Best Parameters:", grid_search.best_params_)
print("Test Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred) * 100))
print("Test AUC: {:.4f}".format(roc_auc_score(y_test, y_proba)))
print("\nClassification Report:")
```

```
Best Parameters: {'clf__max_depth': 10, 'clf__min_samples_split': 5, 'clf__n_estimators': 200}
Test Accuracy: 96.46%
Test AUC: 0.5887

Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       927
           1       0.00      0.00      0.00        34

    accuracy                           0.96       961
   macro avg       0.48      0.50      0.49       961
weighted avg       0.93      0.96      0.95       961
```