var_stat

July 7, 2025

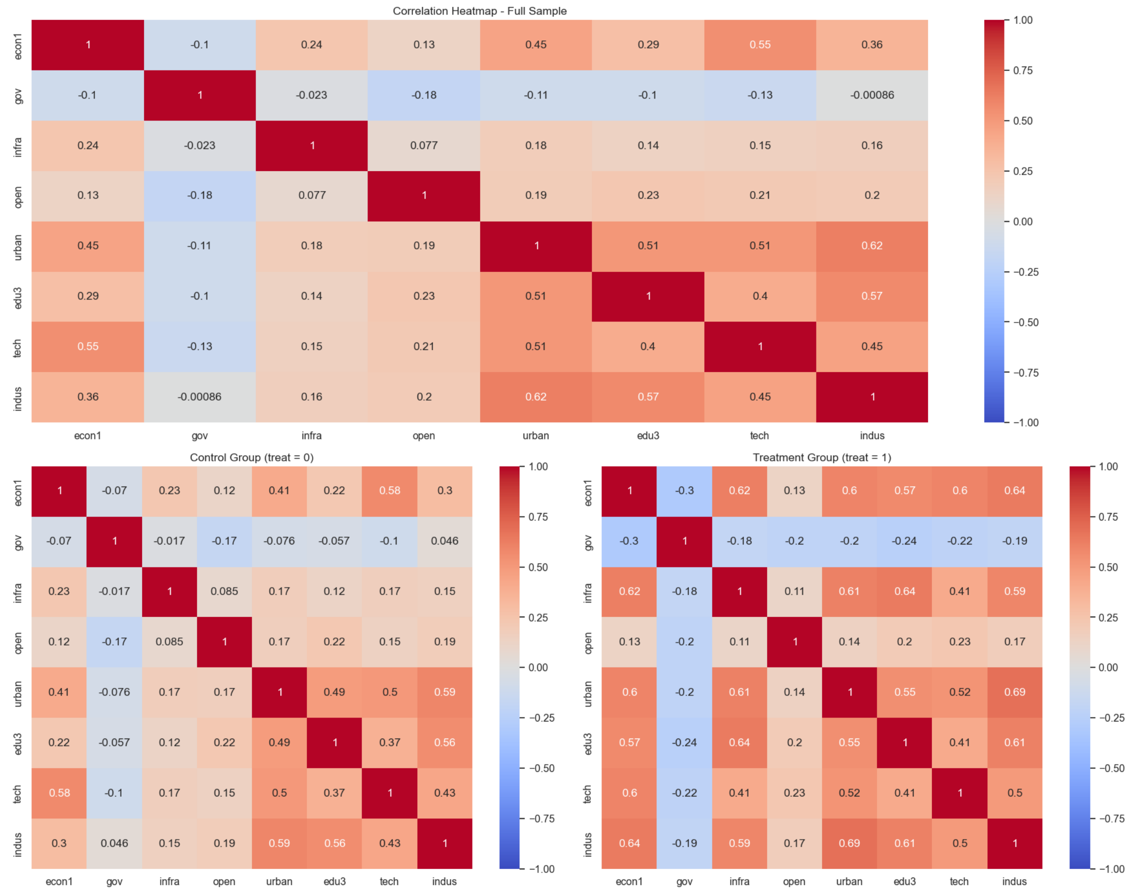
1 1.Descriptive Statistics of Key Variables

1.1 1.1Control Variable Correlation Heatmap

This figure shows the pairwise correlation matrix of control variables across:Full Sample,Control Group (treat = 0) and Treatment Group(treat = 1)

[185]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_stata("data_701.dta")
corr_vars = ["econ1", "gov", "infra", "open", "urban", "edu3", "tech", "indus"]
sns.set(style="white", font_scale=1.05)

fig = plt.figure(figsize=(18, 14))

# 1. full sample
ax1 = plt.subplot2grid((2, 2), (0, 0), colspan=2)
sns.heatmap(df[corr_vars].corr(), annot=True, cmap="coolwarm", vmin=-1, vmax=1,␣
↪ax=ax1)
ax1.set_title("Correlation Heatmap - Full Sample")

# 2. control group
df_control = df[df["treat"] == 0]
ax2 = plt.subplot2grid((2, 2), (1, 0))
sns.heatmap(df_control[corr_vars].corr(), annot=True, cmap="coolwarm", vmin=-1,␣
↪vmax=1, ax=ax2)
ax2.set_title("Control Group (treat = 0)")

# 3. treatment group
df_treat = df[df["treat"] == 1]
ax3 = plt.subplot2grid((2, 2), (1, 1))
sns.heatmap(df_treat[corr_vars].corr(), annot=True, cmap="coolwarm", vmin=-1,␣
↪vmax=1, ax=ax3)
ax3.set_title("Treatment Group (treat = 1)")

plt.tight_layout()
```

```
plt.show()
```



1.2 1.2 Summary Statistics and Multicollinearity Check

This section presents the descriptive statistics for the control variables and evaluates potential multicollinearity using the Variance Inflation Factor (VIF). #### 1.2.1Descriptive Statistics: Includes the mean, standard deviation, min, max, and percentiles of each control variable. #### 1.2.2VIF Test: Variables with VIF values significantly greater than 10 indicate potential multicollinearity, but in this case, all variables are within acceptable range.

[247]:
```
# des.stat.
desc_stats = df[corr_vars].describe()
print("sum stat")
print(desc_stats)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

X = df[corr_vars].dropna()
X = add_constant(X)
```

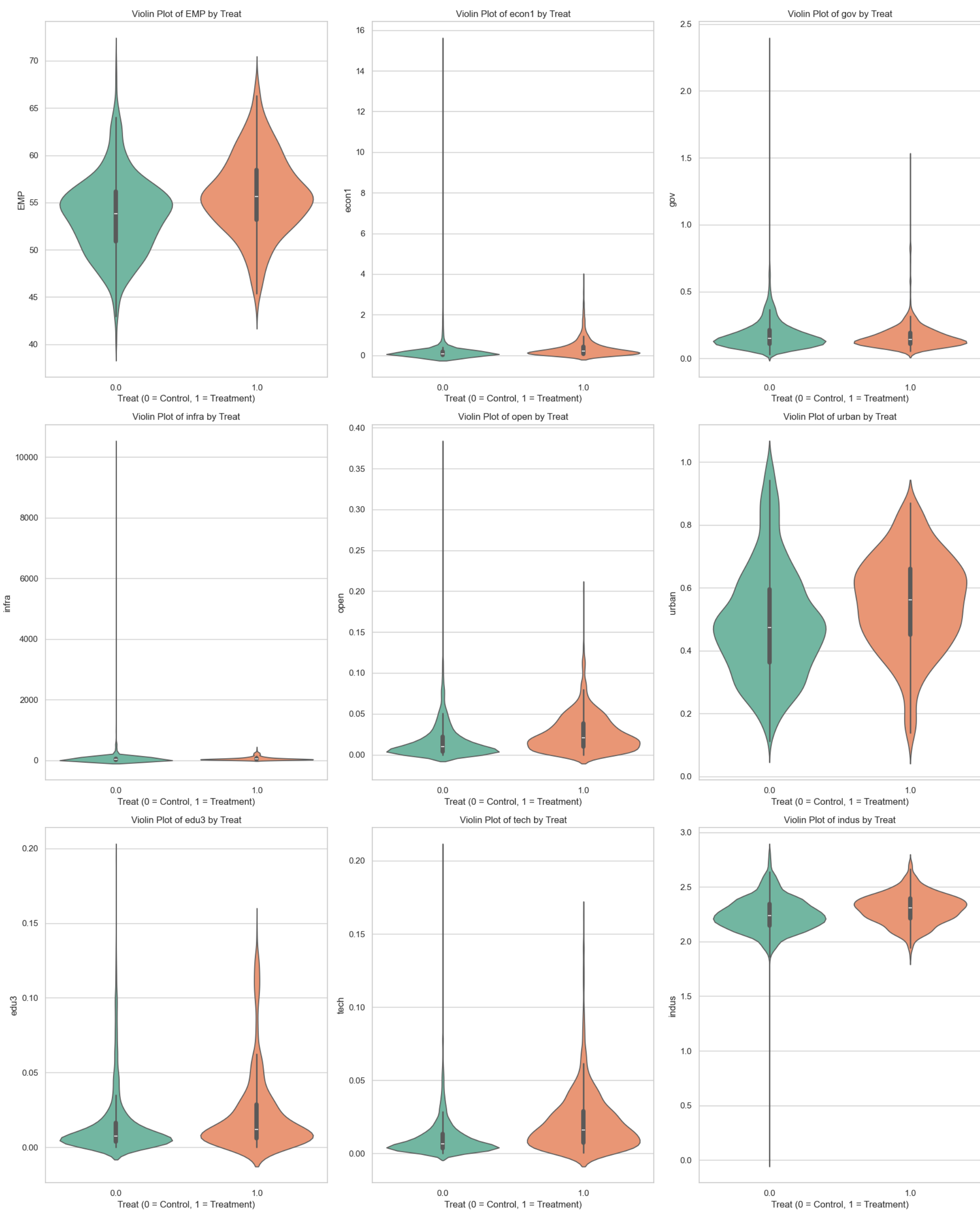
```
# VIF
vif_data = pd.DataFrame()
vif_data["Variable"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]

print("VIF test")
print(vif_data)
```

```
sum stat
             econ1          gov          infra         open        urban  \
count  5420.000000  5420.000000   5420.000000  5401.000000  5295.000000
mean      0.247785     0.176119     63.969557     0.019366     0.504073
std       0.658381     0.121863    260.424712     0.022496     0.173534
min       0.000550     0.031284      1.000000     0.000000     0.111700
25%       0.034697     0.108548     19.000000     0.004109     0.382200
50%       0.091315     0.148621     31.000000     0.012023     0.490800
75%       0.230319     0.207418     57.000000     0.026450     0.617050
max      15.355533     2.348759  10426.000000     0.375790     1.000000

              edu3         tech        indus
count  5401.000000  5420.000000  5393.000000
mean      0.016957     0.013237     2.261816
std       0.023474     0.015225     0.157307
min       0.000000     0.000000     0.000246
25%       0.003949     0.003753     2.158581
50%       0.008334     0.008087     2.253410
75%       0.018730     0.016787     2.361695
max       0.194862     0.206835     2.835702
VIF test
  Variable         VIF
0    const  338.152180
1    econ1    1.558316
2      gov    1.079159
3    infra    1.073737
4     open    1.121857
5    urban    1.960870
6     edu3    1.643246
7     tech    1.737001
8    indus    1.963054
```

1.3 1.3.Kernel Density Plot

[246]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_stata("data_701.dta")

vars_of_interest = ["EMP", "econ1", "gov", "infra", "open", "urban", "edu3",␣
↪"tech", "indus"]

sns.set(style="whitegrid")

plt.figure(figsize=(18, 22))

for i, var in enumerate(vars_of_interest):
    plt.subplot(3, 3, i + 1)

    df_plot = df[["treat", var]].dropna()

    sns.violinplot(x="treat", y=var, hue="treat", data=df_plot,
                   palette="Set2", inner="box", legend=False)

    plt.title(f"Violin Plot of {var} by Treat")
    plt.xlabel("Treat (0 = Control, 1 = Treatment)")
    plt.ylabel(var)

plt.tight_layout()
plt.show()
```



1.4 1.3 Group Comparison of Employment Quality (EMP)

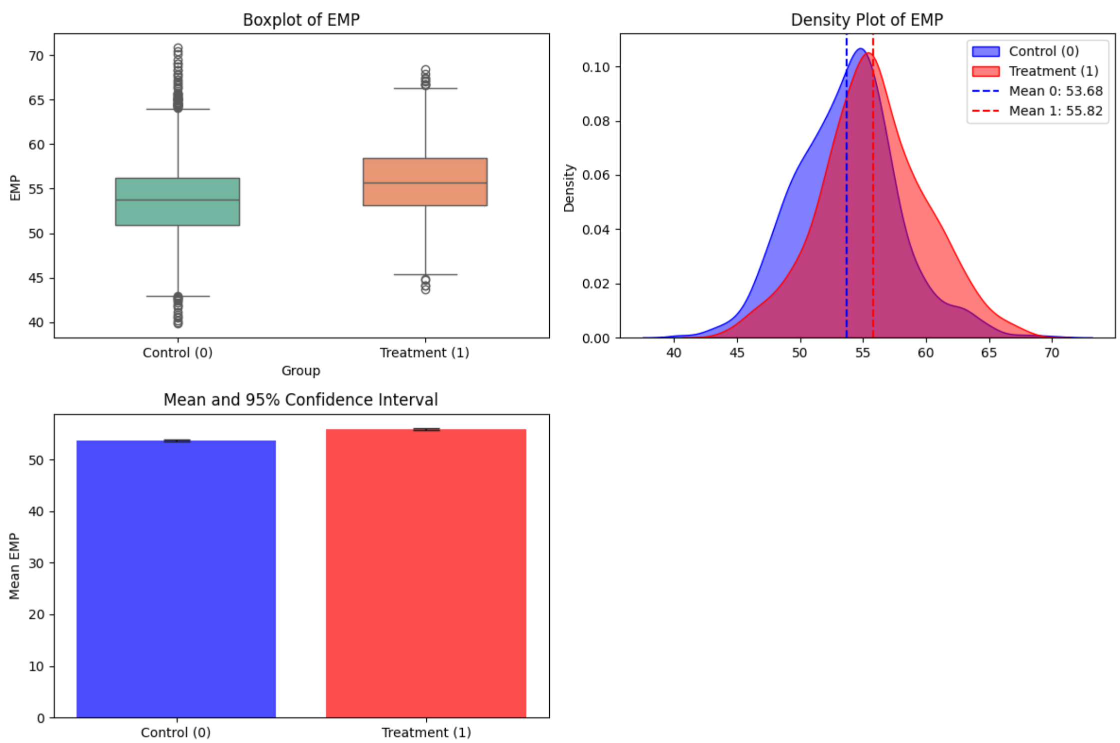
A two-sample t-test reveals a significant difference in EMP between the treatment and control groups, with a t-statistic of -15.88 and a p-value < 0.001. This suggests that the treatment group has significantly higher employment quality.

[221]:

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind


df = pd.read_stata("data_701.dta")

df_clean = df[["EMP", "treat"]].copy()
df_clean = df_clean[pd.to_numeric(df_clean["EMP"], errors="coerce").notna()]
df_clean = df_clean[df_clean["treat"].isin([0, 1])]
df_clean["EMP"] = df_clean["EMP"].astype(float)

group_0 = df_clean[df_clean["treat"] == 0]["EMP"].values
group_1 = df_clean[df_clean["treat"] == 1]["EMP"].values

t_stat, p_value = ttest_ind(group_0, group_1)
print(f"t-statistic: {t_stat:.2f}, p-value: {p_value:.3f}")

data_plot = pd.DataFrame({
    "EMP": np.concatenate([group_0, group_1]),
    "Group": ["Control (0)"] * len(group_0) + ["Treatment (1)"] * len(group_1)
})

plt.figure(figsize=(12, 8))

# Fig.1 boxplot
plt.subplot(2, 2, 1)
sns.boxplot(x="Group", y="EMP", hue="Group", data=data_plot, palette="Set2",␣
 ↪width=0.5, legend=False)
plt.title("Boxplot of EMP", fontsize=12)

# Fig.2 Kernal density
plt.subplot(2, 2, 2)
sns.kdeplot(group_0, label="Control (0)", fill=True, color="blue", alpha=0.5)
sns.kdeplot(group_1, label="Treatment (1)", fill=True, color="red", alpha=0.5)
plt.axvline(np.mean(group_0), color="blue", linestyle="--", label=f"Mean 0: {np.
 ↪mean(group_0):.2f}")
plt.axvline(np.mean(group_1), color="red", linestyle="--", label=f"Mean 1: {np.
 ↪mean(group_1):.2f}")
plt.title("Density Plot of EMP", fontsize=12)
plt.legend()

# Fig.3 mean and 95% Ci
means = [np.mean(group_0), np.mean(group_1)]
std_errs = [np.std(group_0, ddof=1) / np.sqrt(len(group_0)),
```

```
        np.std(group_1, ddof=1) / np.sqrt(len(group_1))]
conf_intervals = [[m - 1.96 * se, m + 1.96 * se] for m, se in zip(means,␣
↪std_errs)]

plt.subplot(2, 2, 3)
plt.bar([0, 1], means,
        yerr=[ci[1] - mean for ci, mean in zip(conf_intervals, means)],
        capsize=10, color=["blue", "red"], alpha=0.7)
plt.xticks([0, 1], ["Control (0)", "Treatment (1)"])
plt.ylabel("Mean EMP")
plt.title("Mean and 95% Confidence Interval", fontsize=12)

plt.tight_layout()
plt.show()
```

```
t-statistic: -15.88, p-value: 0.000
```



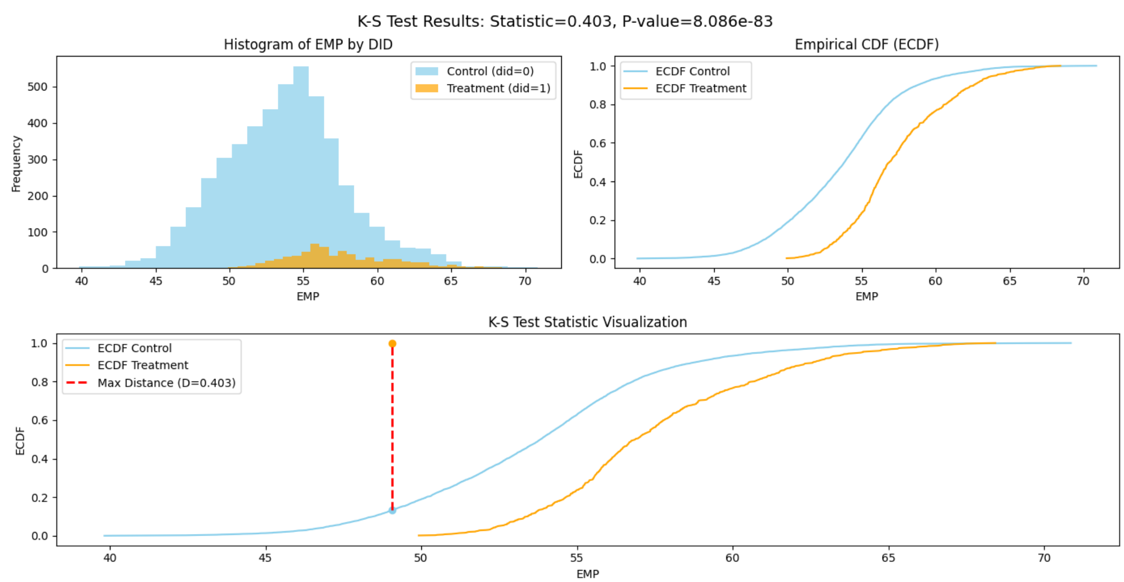
1.5 1.4 K-S Test for Distributional Equality of EMP

The Kolmogorov–Smirnov (K-S) test shows a significant difference in the distribution of Employment Quality (EMP) between treatment and control groups. The test yields a statistic of D = 0.403 and a p-value < 1e-80, indicating that the two groups follow different empirical distributions.

[222]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import ks_2samp

df = pd.read_stata("data_701.dta")
df_filtered = df[["EMP", "did"]].dropna()

emp_treat = df_filtered[df_filtered["did"] == 1]["EMP"]
emp_control = df_filtered[df_filtered["did"] == 0]["EMP"]

#  K-S test
ks_stat, p_value = ks_2samp(emp_treat, emp_control)

# ECDF
def ecdf(data):
    x = np.sort(data)
    y = np.arange(1, len(x)+1) / len(x)
    return x, y

x1, y1 = ecdf(emp_control)
x2, y2 = ecdf(emp_treat)

#
min_len = min(len(y1), len(y2))
d_index = np.argmax(np.abs(y1[:min_len] - y2[:min_len]))
d_value = np.abs(y1[d_index] - y2[d_index])
d_x = x1[d_index]

plt.style.use('default')
plt.figure(figsize=(14, 7))

# 1. histogram
plt.subplot(2, 2, 1)
plt.hist(emp_control, bins=30, alpha=0.7, label="Control (did=0)",␣
↪color="skyblue")
plt.hist(emp_treat, bins=30, alpha=0.7, label="Treatment (did=1)",␣
↪color="orange")
plt.title("Histogram of EMP by DID")
plt.xlabel("EMP")
plt.ylabel("Frequency")
plt.legend()

# 2.Find the point of maximum distance between the two ECDFs
plt.subplot(2, 2, 2)
plt.plot(x1, y1, label="ECDF Control", color="skyblue")
plt.plot(x2, y2, label="ECDF Treatment", color="orange")
```

```
plt.title("Empirical CDF (ECDF)")
plt.xlabel("EMP")
plt.ylabel("ECDF")
plt.legend()

# 3. K-S plot
plt.subplot(2, 1, 2)
plt.plot(x1, y1, label="ECDF Control", color="skyblue")
plt.plot(x2, y2, label="ECDF Treatment", color="orange")
plt.vlines(d_x, y1[d_index], y2[d_index], color="red", linestyle="--", lw=2,
           label=f"Max Distance (D={ks_stat:.3f})")
plt.scatter([d_x], [y1[d_index]], color="skyblue")
plt.scatter([d_x], [y2[d_index]], color="orange")
plt.title("K-S Test Statistic Visualization")
plt.xlabel("EMP")
plt.ylabel("ECDF")
plt.legend()

plt.tight_layout()
plt.suptitle(f"K-S Test Results: Statistic={ks_stat:.3f}, P-value={p_value:.
 ↪3e}", fontsize=14, y=1.02)
plt.show()
```



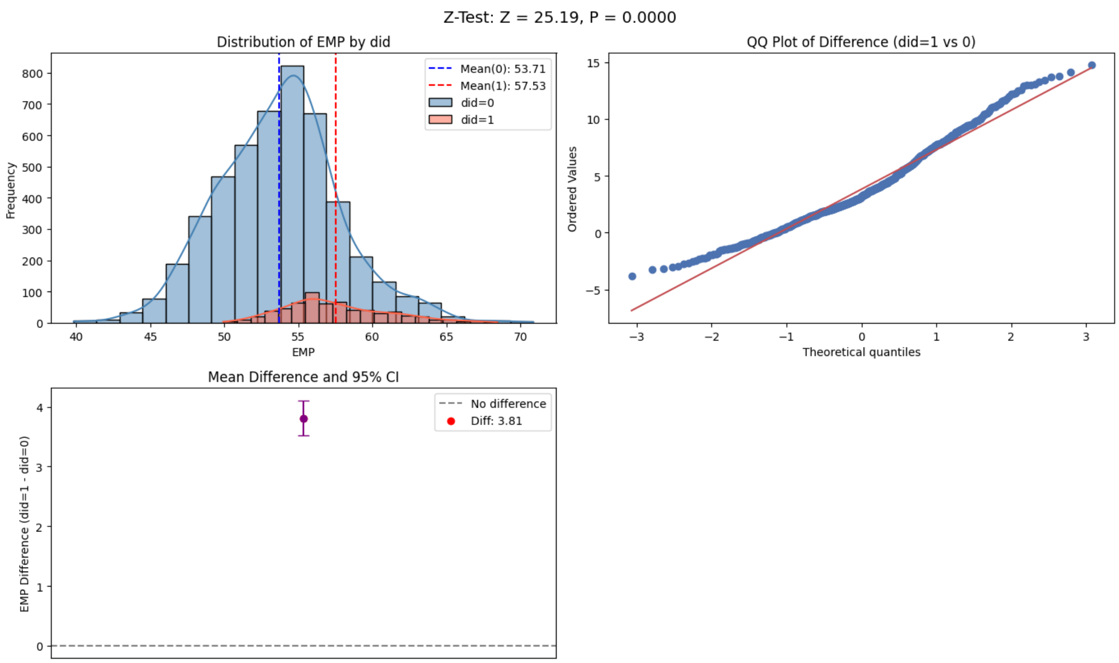
1.6 1.5 Difference in Means: Z-Test

To examine the difference in high-quality employment (EMP) between the treatment and control groups, we conduct a Z-test for means. The histogram and density plots reveal a rightward shift in the EMP distribution for the treatment group. The Z statistic is 25.19 with a p-value < 0.0001, indicating a statistically significant difference in group means. The bottom panel shows the mean difference (3.81) along with its 95% confidence interval, which does not cross zero.

[223]:
```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_stata("data_701.dta")

df = df[["EMP", "did"]].dropna()
df["EMP"] = pd.to_numeric(df["EMP"], errors="coerce")
df["did"] = pd.to_numeric(df["did"], errors="coerce")
df = df.dropna()

group0 = df[df["did"] == 0]["EMP"].values
group1 = df[df["did"] == 1]["EMP"].values

mean0, mean1 = np.mean(group0), np.mean(group1)
std0, std1 = np.std(group0, ddof=1), np.std(group1, ddof=1)
n0, n1 = len(group0), len(group1)

# calculate Z-score
pooled_std = np.sqrt((std0**2 / n0) + (std1**2 / n1))
z_statistic = (mean1 - mean0) / pooled_std
p_value = 2 * (1 - stats.norm.cdf(abs(z_statistic)))

print(f"Z-stat.: {z_statistic:.2f}")
print(f"p-value: {p_value:.4f}")

# CI 95%
z_critical = stats.norm.ppf(0.975)
margin_of_error = z_critical * pooled_std
ci_low = (mean1 - mean0) - margin_of_error
ci_high = (mean1 - mean0) + margin_of_error

plt.figure(figsize=(14, 8))

# Fig. 1: hist + KDE
plt.subplot(2, 2, 1)
sns.histplot(group0, kde=True, label="did=0", color="steelblue", bins=20)
sns.histplot(group1, kde=True, label="did=1", color="tomato", bins=20)
plt.axvline(mean0, color="blue", linestyle="--", label=f"Mean(0): {mean0:.2f}")
plt.axvline(mean1, color="red", linestyle="--", label=f"Mean(1): {mean1:.2f}")
plt.title("Distribution of EMP by did")
plt.xlabel("EMP")
```

```python
plt.ylabel("Frequency")
plt.legend()

# Fig 2: QQ plot
plt.subplot(2, 2, 2)
stats.probplot(group1 - group0.mean(), dist="norm", plot=plt)
plt.title("QQ Plot of Difference (did=1 vs 0)")

# Fig 3: mean diff + CI
plt.subplot(2, 2, 3)
plt.errorbar(1, mean1 - mean0, yerr=margin_of_error, fmt='o', color='purple',
    capsize=5)
plt.axhline(0, color="gray", linestyle="--", label="No difference")
plt.scatter(1, mean1 - mean0, color="red", label=f"Diff: {mean1 - mean0:.2f}")
plt.title("Mean Difference and 95% CI")
plt.ylabel("EMP Difference (did=1 - did=0)")
plt.xticks([])
plt.legend()

plt.tight_layout()
plt.suptitle(f"Z-Test: Z = {z_statistic:.2f}, P = {p_value:.4f}", fontsize=14,
    y=1.02)
plt.show()
```

```
Z-stat.: 25.19
p-value: 0.0000
```



2 2.Principal Component Analysis (PCA)

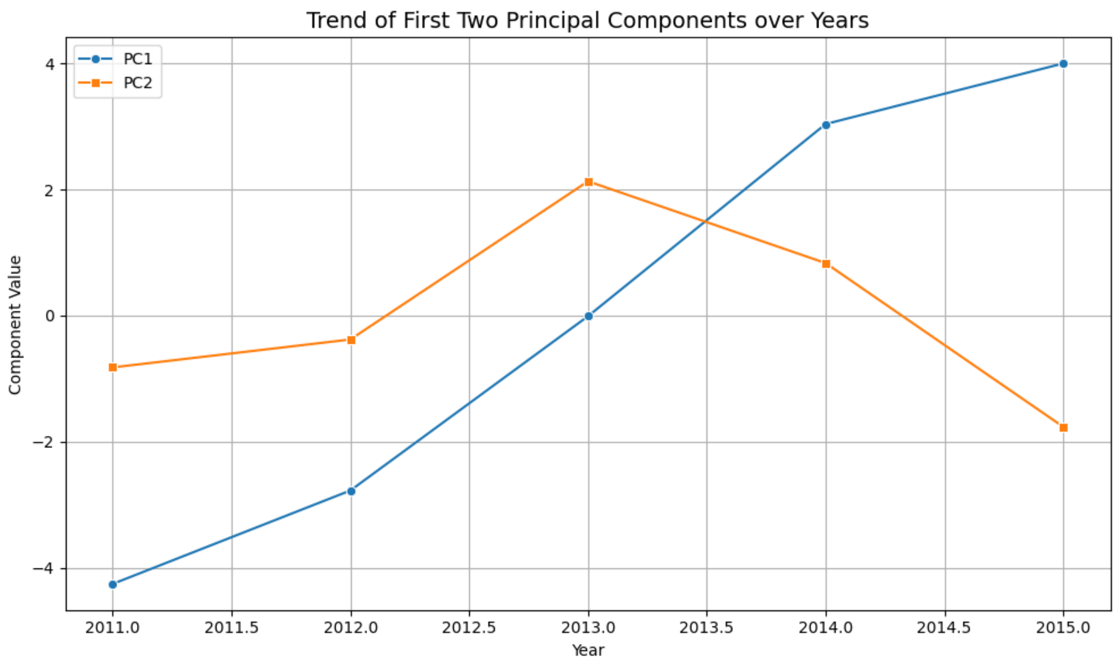
2.0.1 2.1.Principal Component Analysis (PCA): Variance Explained and Trend Over Time

The PCA results show that the first two principal components explain the majority of the variance in the employment quality indicators, with **PC1 accounting for 72.92%** and **PC2 for 13.10%**, respectively. The line plot below illustrates the temporal trends of PC1 and PC2 from 2011 to 2015, highlighting dynamic changes in the overall structure of employment quality across years.

[241]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

df = pd.read_stata("/Users/libingchen/Desktop/urbanization/Y.dta")

pca_vars = ['gdp_norm', 'employ_norm', 'thrid_norm', 'transport_norm',␣
 ↪'wage_norm',
            'insurance1_norm', 'insurance2_norm', 'insurance3_norm',
            'edu_norm', 'college_norm', 'hospital_norm', 'library_norm',
            'social_norm', 'aveedu_norm']

df_mean = df.groupby("year")[pca_vars].mean().reset_index()
df_mean_clean = df_mean.dropna()

#standard
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_mean_clean[pca_vars])

# PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
df_pca = pd.DataFrame(X_pca, columns=[f"PC{i + 1}" for i in range(X_pca.
 ↪shape[1])])
df_pca["year"] = df_mean_clean["year"].values

# Explained Variance Ratio by Principal Components
print("Explained Variance Ratio by Principal Components ")
for i, ratio in enumerate(pca.explained_variance_ratio_):
    print(f"PC{i+1}: {ratio:.4f}")

# Temporal Trends of the First Two Principal Components (PC1 and PC2)
plt.figure(figsize=(10, 6))
sns.lineplot(data=df_pca, x="year", y="PC1", marker='o', label="PC1")
sns.lineplot(data=df_pca, x="year", y="PC2", marker='s', label="PC2")
```

```python
plt.title("Trend of First Two Principal Components over Years", fontsize=14)
plt.xlabel("Year")
plt.ylabel("Component Value")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
Explained Variance Ratio by Principal Components
PC1: 0.7292
PC2: 0.1310
PC3: 0.0883
PC4: 0.0515
PC5: 0.0000
```



2.0.2 2.2.Principal Component Loadings Analysis

To explore which normalized indicators contribute the most to the principal components, we examined the component loadings. The bar charts below illustrate the loadings of each variable on PC1 and PC2.

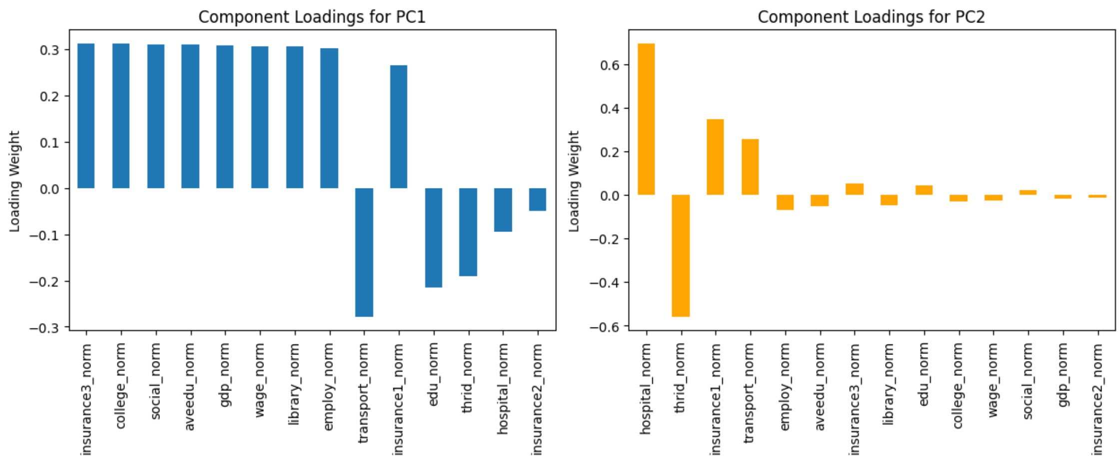
- For **PC1**, the most influential variables (with high absolute loadings) are `insurance3_norm`, `college_norm`, `social_norm`, and `aveedu_norm`, indicating these indicators play a dominant role in explaining the first principal component.
- For **PC2**, `hospital_norm` and `third_norm` have the largest contributions, suggesting that this component captures variations primarily driven by healthcare and tertiary industry employment.

The loading directions (positive or negative) indicate whether the variable increases or decreases the corresponding component value.

[243]:
```
actual_vars = df_mean_clean[pca_vars].columns.tolist()

loadings = pd.DataFrame(
    pca.components_.T,
    columns=[f"PC{i + 1}" for i in range(X_pca.shape[1])],
    index=actual_vars
)

# Create a summary DataFrame of absolute loadings for PC1 and PC2, sorted by PC1
loadings_summary = pd.DataFrame({
    "PC1 (abs)": loadings["PC1"].abs(),
    "PC2 (abs)": loadings["PC2"].abs()
}).sort_values(by="PC1 (abs)", ascending=False)

# Display top contributing variables
print("Top Contributing Variables to PC1 and PC2 (by Absolute Loadings):")
print(loadings_summary)

# Visualize the component loadings for PC1 and PC2
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
loadings["PC1"].sort_values(key=lambda x: abs(x), ascending=False).
 ↪plot(kind="bar")
plt.title("Component Loadings for PC1")
plt.ylabel("Loading Weight")

plt.subplot(1, 2, 2)
loadings["PC2"].sort_values(key=lambda x: abs(x), ascending=False).
 ↪plot(kind="bar", color='orange')
plt.title("Component Loadings for PC2")
plt.ylabel("Loading Weight")

plt.tight_layout()
plt.show()
```

```
Top Contributing Variables to PC1 and PC2 (by Absolute Loadings):
                 PC1 (abs)  PC2 (abs)
insurance3_norm   0.312012   0.049578
college_norm      0.310744   0.029223
social_norm       0.309962   0.022859
aveedu_norm       0.309274   0.051987
gdp_norm          0.307106   0.017603
wage_norm         0.306299   0.028122
library_norm      0.305138   0.048466
```

```
employ_norm        0.302007   0.071798
transport_norm     0.278453   0.254221
insurance1_norm    0.263963   0.347016
edu_norm           0.216141   0.045014
thrid_norm         0.191044   0.560577
hospital_norm      0.094647   0.695220
insurance2_norm    0.050244   0.014358
```

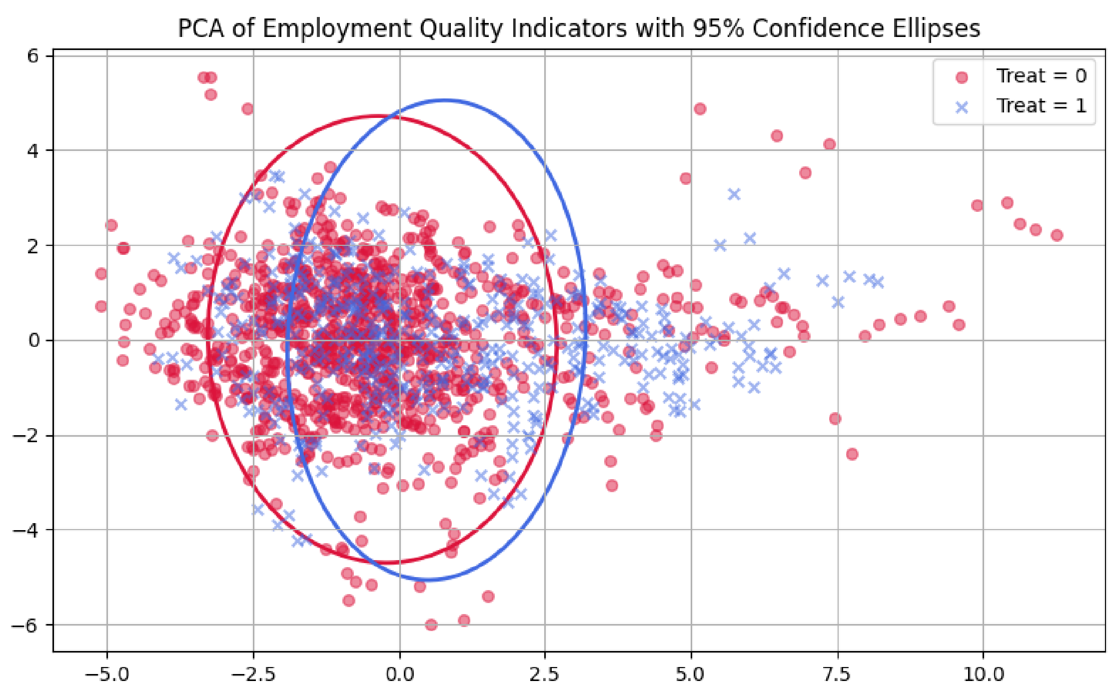


2.0.3 2.3. PCA Scatter Plot of Employment Quality Indicators

This PCA plot shows the distribution of treatment and control groups along the first two principal components. The 95% confidence ellipses suggest that the two groups have broadly overlapping distributions, indicating limited structural differences in employment quality indicators.

[230]:
```python
import numpy as np
from matplotlib.patches import Ellipse

def draw_confidence_ellipse(data, ax, n_std=2.0, edgecolor='black'):
    if len(data) < 2:
        return
    cov = np.cov(data.T)
    mean = np.mean(data, axis=0)
    vals, vecs = np.linalg.eigh(cov)
    order = vals.argsort()[::-1]
    theta = np.degrees(np.arctan2(*vecs[:, 0][::-1]))
    width, height = 2 * n_std * np.sqrt(vals[order])
    ellipse = Ellipse(xy=mean, width=width, height=height, angle=theta,
                      edgecolor=edgecolor, facecolor='none', lw=2)
    ax.add_patch(ellipse)

import matplotlib.pyplot as plt
```

```
df_clean = df.dropna(subset=pca_vars + ['treat'])
X_scaled = StandardScaler().fit_transform(df_clean[pca_vars])
X_pca = PCA(n_components=2).fit_transform(X_scaled)
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df_pca['treat'] = df_clean['treat'].values

colors = {0.0: 'crimson', 1.0: 'royalblue'}
markers = {0.0: 'o', 1.0: 'x'}
fig, ax = plt.subplots(figsize=(8, 5))

for t in df_pca['treat'].unique():
    group = df_pca[df_pca['treat'] == t]
    ax.scatter(group['PC1'], group['PC2'], label=f'Treat = {int(t)}',
               alpha=0.5, s=30, color=colors[t], marker=markers[t])
    draw_confidence_ellipse(group[['PC1', 'PC2']].values, ax,␣
 ↪edgecolor=colors[t])

ax.set_title("PCA of Employment Quality Indicators with 95% Confidence␣
 ↪Ellipses")
ax.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



2.0.4 2.4.Annual PCA of Employment Quality Indicators (2007–2021)

This panel displays year-by-year PCA results with 95% confidence ellipses for treatment and control groups. The consistent overlap between the two ellipses across years indicates that group-level differences in employment quality indicators remain relatively small and stable over time.

[244]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from matplotlib.patches import Ellipse


df = pd.read_stata("/Users/libingchen/Desktop/urbanization/Y.dta")

all_pca_vars = ['gdp_norm', 'employ_norm', 'thrid_norm', 'transport_norm',␣
↪'wage_norm',
                'insurance1_norm', 'insurance2_norm', 'insurance3_norm',
                'edu_norm', 'college_norm', 'hospital_norm', 'library_norm',
                'social_norm', 'aveedu_norm']

# confidence ellipse
def draw_confidence_ellipse(data, ax, n_std=2.0, edgecolor='black', **kwargs):
    if len(data) < 2:
        return
    cov = np.cov(data.T)
    mean = np.mean(data, axis=0)
    vals, vecs = np.linalg.eigh(cov)
    order = vals.argsort()[::-1]
    vals = vals[order]
    vecs = vecs[:, order]
    theta = np.degrees(np.arctan2(*vecs[:, 0][::-1]))
    width, height = 2 * n_std * np.sqrt(vals)
    ellipse = Ellipse(xy=mean, width=width, height=height, angle=theta,
                      edgecolor=edgecolor, facecolor='none', lw=2, **kwargs)
    ax.add_patch(ellipse)

# Plot layout parameters
years = list(range(2007, 2022))
ncols = 5
nrows = int(np.ceil(len(years) / ncols))
fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(4.2 * ncols, 4 *␣
↪nrows))
axes = axes.flatten()

# Main loop: plot PCA results year by year
for i, year in enumerate(years):
```

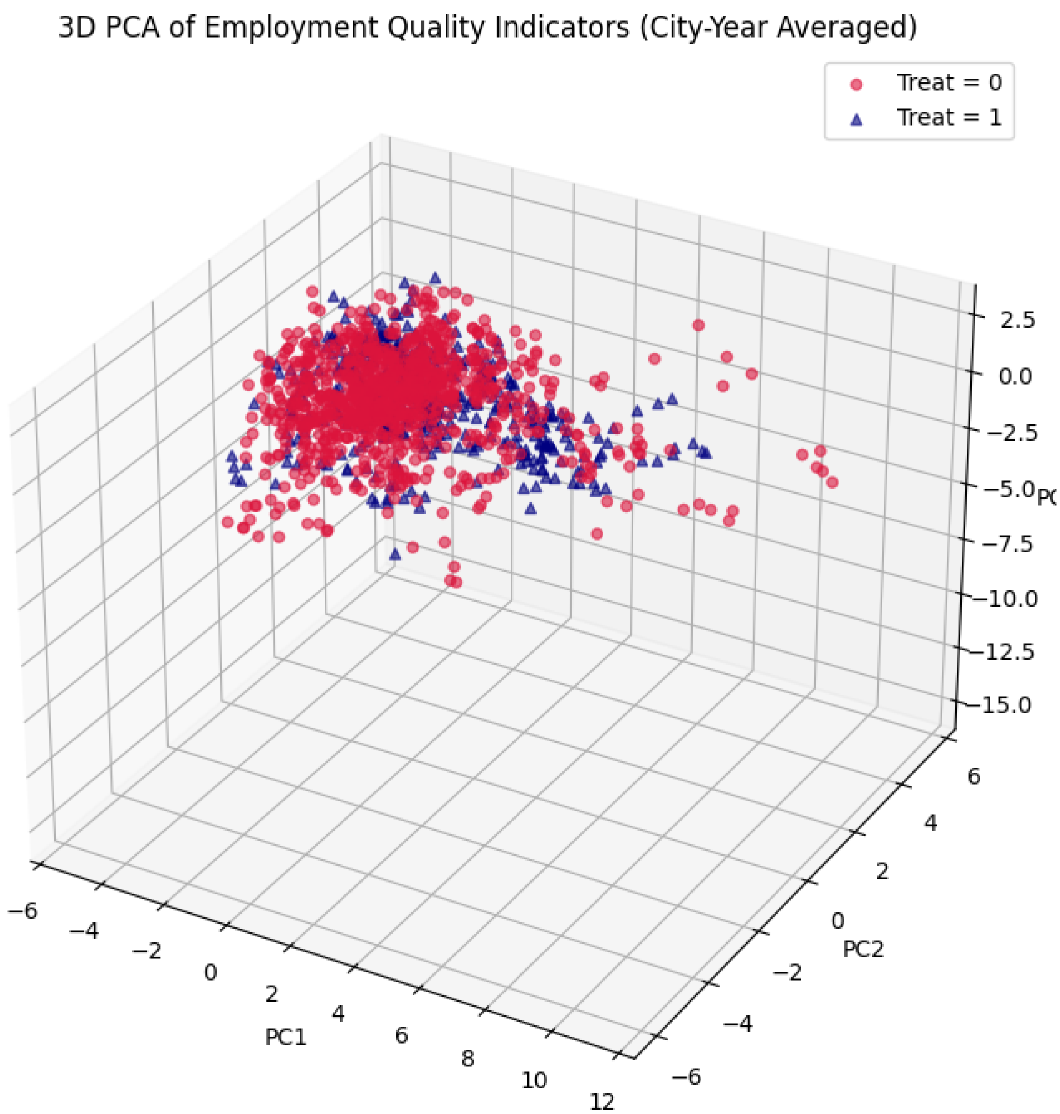
```python
    ax = axes[i]
    df_y = df[df['year'] == year].copy()

    # Check missing ratios for each variable
    missing_ratio = df_y[all_pca_vars].isnull().mean()
    valid_vars = missing_ratio[missing_ratio < 0.5].index.tolist()

    if len(valid_vars) < 2:
        ax.set_title(f"{year}: No Data")
        ax.axis('off')
        continue

    df_y = df_y.dropna(subset=valid_vars + ['treat'])

    if df_y.empty:
        ax.set_title(f"{year}: No Data")
        ax.axis('off')
        continue

    # Standardize
    X_scaled = StandardScaler().fit_transform(df_y[valid_vars])

    # PCA
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_scaled)
    df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
    df_pca['treat'] = df_y['treat'].values

    # Plotting
    colors = {0.0: 'crimson', 1.0: 'royalblue'}
    markers = {0.0: 'o', 1.0: 'x'}

    for t in df_pca['treat'].unique():
        group = df_pca[df_pca['treat'] == t]
        ax.scatter(group['PC1'], group['PC2'], alpha=0.5, s=25,
                   label=f'Treat={int(t)}', color=colors[t], marker=markers[t])
        draw_confidence_ellipse(group[['PC1', 'PC2']].values, ax,␣
↪edgecolor=colors[t])

    ax.set_title(f"{year}", fontsize=11)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.grid(True)

for j in range(i + 1, len(axes)):
    axes[j].axis('off')
```

```
handles, labels = axes[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper center', ncol=2, fontsize=12)
plt.suptitle("PCA by Year with 95% Confidence Ellipses (2007-2021)", fontsize=15)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

2.0.5 3.5. 3D PCA of EMP

[232]:
```
from mpl_toolkits.mplot3d import Axes3D

df_avg = df.dropna(subset=pca_vars + ['treat']).groupby(['city', 'year'], as_index=False).mean()
X_scaled = StandardScaler().fit_transform(df_avg[pca_vars])
X_pca = PCA(n_components=3).fit_transform(X_scaled)
df_avg['PC1'], df_avg['PC2'], df_avg['PC3'] = X_pca[:,0], X_pca[:,1], X_pca[:,2]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
colors = {0.0: 'crimson', 1.0: 'navy'}
markers = {0.0: 'o', 1.0: '^'}

for t in df_avg['treat'].unique():
    d = df_avg[df_avg['treat'] == t]
    ax.scatter(d['PC1'], d['PC2'], d['PC3'], label=f'Treat = {int(t)}',
               color=colors[t], marker=markers[t], alpha=0.6)
```

```
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
ax.set_title("3D PCA of Employment Quality Indicators (City-Year Averaged)")
ax.legend()
plt.tight_layout()
plt.show()
```



2.0.6 3.6. Annual 3D PCA of EMP

[245]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D

df = pd.read_stata("/Users/libingchen/Desktop/urbanization/Y.dta")

all_pca_vars = ['gdp_norm', 'employ_norm', 'thrid_norm', 'transport_norm', 'wage_norm',
                'insurance1_norm', 'insurance2_norm', 'insurance3_norm',
                'edu_norm', 'college_norm', 'hospital_norm', 'library_norm',
                'social_norm', 'aveedu_norm']

years = list(range(2007, 2022))
ncols = 5
nrows = int(np.ceil(len(years) / ncols))
fig = plt.figure(figsize=(5.2 * ncols, 4.5 * nrows))

for i, year in enumerate(years):
    ax = fig.add_subplot(nrows, ncols, i + 1, projection='3d')
    df_y = df[df['year'] == year].copy()

    #     50%
    missing_ratio = df_y[all_pca_vars].isnull().mean()
    valid_vars = missing_ratio[missing_ratio < 0.5].index.tolist()

    if len(valid_vars) < 3:
        ax.set_title(f"{year}: No Data")
        ax.axis('off')
        continue

    df_y = df_y.dropna(subset=valid_vars + ['treat'])
    if df_y.empty:
        ax.set_title(f"{year}: No Data")
        ax.axis('off')
        continue

    X_scaled = StandardScaler().fit_transform(df_y[valid_vars])
    pca = PCA(n_components=3)
    X_pca = pca.fit_transform(X_scaled)

    df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2', 'PC3'])
    df_pca['treat'] = df_y['treat'].values
```

```
    colors = {0.0: 'crimson', 1.0: 'royalblue'}
    markers = {0.0: 'o', 1.0: '^'}
    for t in df_pca['treat'].unique():
        group = df_pca[df_pca['treat'] == t]
        ax.scatter(group['PC1'], group['PC2'], group['PC3'],
                   alpha=0.6, s=25, c=colors[t], marker=markers[t], label=f'Treat={int(t)}')

    ax.set_title(f"{year}", fontsize=10)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_zticks([])
    ax.grid(False)

fig.suptitle("3D PCA by Year (2007-2021)", fontsize=16)
fig.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```