Joining Data

The pandas package is a powerful tool for manipulating and transforming data in Python. However, when working on an analysis, the data needed could be in multiple tables. This worksheet will focus on the vital skill of merging tables together.

As an example, we are considering wards data of Chicago city.

The city of Chicago is divided into fifty local neighborhoods called wards. We have a table with data about the local government offices in each ward. In this example, we want to merge the local government data with census data about the population of each ward.

The ward data

```
wards = pd.read_csv('Ward_Offices.csv')
print(wards.head())
print(wards.shape)
```

```
  ward  alderman          address
0 1     Proco "Joe" ...   2058 NORTH W...
1 2     Brian Hopkins     1400 NORTH
```

Census data

```
census = pd.read_csv('Ward_Census.csv')
print(census.head())
print(census.shape)
```

```
  ward  pop_2000  pop_2010  change  addre
0 1     52951     56149     6%      2765
1 2     54361     55805     3%      WM WA
```

Inner join

```
wards_census = wards.merge(census, on='ward')
print(wards_census.head(4))
```

```
  ward  alderman         address_x        zip_
0 1     Proco "Joe" ...  2058 NORTH W...  60647
1 2     Brian Hopkins    1400 NORTH       6062
```

Suffixes

```
wards_census = wards.merge(census, on='ward', suffixes=('_ward','_cen'))
print(wards_census.head())
print(wards_census.shape)
```

```
  ward  alderman         address_ward     zip_ward  pop_2000  pop_2010  ch
```

```
#Ex 1

taxi_owners = pd.read_pickle("taxi_owners.p")
taxi_vehicles = pd.read_pickle("taxi_vehicles.p")
taxi_own_veh = taxi_owners.merge(taxi_vehicles, on = "vid")
print(taxi_own_veh["fuel_type"].value_counts())
print()

print("The highest is : ", taxi_own_veh["fuel_type"].max())
```

```
# Ex 2

wards_altered = pd.read_csv("Wards_Offices_Altered.csv")
wards_census_altered = wards_altered.merge(census, on = "ward")
print(wards_census_altered)

print("there are 46 rows")
```

```
#Ex 3
licenses = pd.read_pickle("licenses.p")
biz_owners = pd.read_pickle("business_owners.p")


licenses_owners = licenses.merge(biz_owners, on = "account")
print(licenses_owners.head())
print("-----------------------------------------------")

counted_df = licenses_owners.groupby("title").agg({"account" : "count"})
print(counted_df.head())
print("-----------------------------------------------")

sorted_df = counted_df.sort_values(["account"], ascending = False)
print(sorted_df.head())
```

```
grants_license = grants.merge(licenses, on = 'zip')
print(grants_licenses.loc[grants_licenses["business"]== "REGGIE'S BAR & GRILL",
                          ["grant", "company", "account", "ward", "business"]])
```

```
grants_license_ward = grants.merge(license, on = ["address", "zip"]) \
.merge(wards, on = "ward", suffixes = ("bus", "_wards"))
```

```
cal = pd.read_pickle("cta_calendar.p")
ridership = pd.read_pickle("cta_ridership.p")
stations = pd.read_pickle("stations.p")


ridership_cal_station = ridership.merge
(cal, on = [ "year", "month", "day"]).merge(stations, on =
                                             ["station_id"])

fil = ((ridership_cal_station [ "month" ] == 7)
      & (ridership_cal_station [ 'day_type'] == 'Weekday')
      & (ridership_cal_station [ "station_name"] == "Wilson"))

print(ridership_cal_station.loc[fil, 'rides'].sum())
```

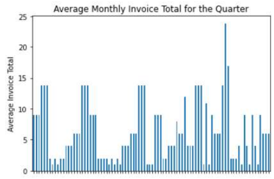
```
140005
```

```
toy_story = pd.read_csv("toy_story.csv")
taglines = pd.read_pickle("taglines.p")

toy_story_tag = toy_story.merge(taglines, on = "id", how="left")

#toy_story_tag = toy_story.merge(taglines, on = "id")

toy_story_tag
```

```
#Ex 8
import pandas as pd

# Load data
movies = pd.read_pickle('movies.p')

# Subset scifi_movies and action_movies
scifi_movies = movie_to_genres[movie_to_genres['genre'] == 'Science Fiction']
action_movies = movie_to_genres[movie_to_genres['genre'] == 'Action']

# Merge action_movies and scifi_movies with a right join and add suffixes
action_scifi = pd.merge(action_movies, scifi_movies, on='movie_id', how='right', suffixes=('_act', '_sci'))

# Subset rows where genre_act column is null (only science fiction)
scifi_only = action_scifi[action_scifi['genre_act'].isnull()]

# Merge movies and scifi_only with an inner join
result = pd.merge(movies, scifi_only, left_on='id', right_on='movie_id', how='inner')

# Display the result
print(result[['id', 'title']])
```

```
#Ex 9

import pandas as pd
import matplotlib.pyplot as plt

# Load data from CSV files
pop_movies = pd.read_csv('pop_movies.csv')
movie_to_genres = pd.read_csv('tdmb_movie_to_genres.csv')

# Merge using right join
genres_movies = pd.merge(movie_to_genres, pop_movies, left_on='movie_id', right_on='id', how='right')

# Group by genre and count the number of movies
genre_counts = genres_movies['genre'].value_counts().head(10)

# Plot the bar chart
plt.figure(figsize=(10, 6))
genre_counts.plot(kind='bar')
plt.xlabel('Genre')
plt.ylabel('Number of Movies')

plt.show()
```

```
#Ex 12

import pandas as pd
import matplotlib.pyplot as plt

inv_jul = pd.read_csv("inv_jul.csv")
inv_aug = pd.read_csv("inv_aug.csv")
inv_sep = pd.read_csv("inv_sep.csv")

avg_inv_by_month = pd.concat([inv_jul, inv_aug, inv_sep], keys=['7Jul', '8Aug', '9Sep'])

average_totals = avg_inv_by_month.groupby('invoice_date')['total'].agg('mean')

average_totals.plot(kind='bar')
plt.xlabel('Month')
plt.ylabel('Average Invoice Total')
plt.title('Average Monthly Invoice Total for the Quarter')
plt.xticks(rotation=0)
plt.show()
```

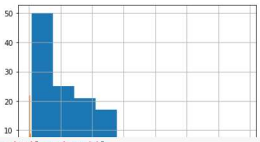



```
#Ex 14

import pandas as pd
import matplotlib.pyplot as plt

unemployment = pd.read_csv("unemployment.csv")
inflation = pd.read_csv("inflation.csv")

inflation_unemploy = pd.merge_ordered(inflation, unemployment, on='date')
print(inflation_unemploy)

plt.scatter(inflation_unemploy['unemployment_rate'], inflation_unemploy['cpi'])
plt.xlabel('Unemployment Rate')
plt.ylabel('CPI (Inflation)')
plt.title('Scatter Plot of Unemployment Rate vs. CPI (Inflation)')
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

rice_consumption = food_consumption(food_consumption["food_category"] == "rice")

rice_consumption("co2_emission")
```

```
print(food.groupby("food_category") ['co2_emission'].agg([np.var, np.std]))

import matplotlib.pyplot as plt
print(food[food["food_category"] == 'beef'] ['co2_emission'].hist())
print(food[food["food_category"] == "eggs"] ['co2_emission'].hist())
```

```
                     var         std
food_category
beef        88748.408132  297.906710
dairy       17671.891985  132.935669
eggs           21.371819    4.622966
fish          921.637349   30.358481
lamb_goat   16475.518363  128.356996
nuts           35.639652    5.969895
pork         3094.963537   55.632396
poultry       245.026801   15.653332
rice         2281.376243   47.763754
soybeans        0.879882    0.938020
wheat          71.023937    8.427570
AxesSubplot(0.125,0.125;0.775x0.755)
AxesSubplot(0.125,0.125;0.775x0.755)
```



```
perch = fish[fish['species'] == 'Perch']
print(perch.head())
sns.regplot(x='length_cm', y = 'mass_g', data = perch, ci=None)
plt.show()
perch['length_cm_cubed'] = perch['length_cm']**3
sns.regplot(x='length_cm_cubed', y = 'mass_g', data = perch, ci=None)
plt.show()
mdl_perch = ols('mass_g ~ length_cm_cubed', data=perch).fit()
print(mdl_perch.params)
explanatory_data = pd.DataFrame({'length_cm_cubed': np.arange(10,41,5)**3,
'length_cm': np.arange(10,41,5)})
prediction_data = explanatory_data.assign(mass_g=mdl_perch.predict(explanatory_data))
print(prediction_data)
fig = plt.figure()
sns.regplot(x='length_cm_cubed', y = 'mass_g', data = perch, ci=None)
sns.scatterplot(data=prediction_data, x = 'length_cm_cubed',
                y = "mass_g", color='red', marker='s')

plt.show()
```

```
   species  mass_g  length_cm
55   Perch     5.9        7.5
56   Perch    32.0       12.5
57   Perch    40.0       13.8
58   Perch    51.5       15.0
59   Perch    70.0       15.7
```

```
# Calculate total co2_emission per country: emissions_by_country
emissions_by_country = food.groupby('country')['co2_emission'].sum()
# Compute the first and third quantiles and IQR of emissions_by_country
q1 = np.quantile(emissions_by_country, 0.25)
q3 = np.quantile(emissions_by_country, 0.75)
iqr = q3 - q1
# Calculate the lower and upper cutoffs for outliers
lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr
# Subset emissions_by_country to find outliers
outliers = emissions_by_country[(emissions_by_country < lower) | (emissions_by_country > upper)]
print(outliers)
```

```
country
Argentina    2172.4
Name: co2_emission, dtype: float64
```

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Load the DataFrame from the CSV file
world_happiness = pd.read_csv('world_happiness.csv')

# Scatterplot without a trendline
sns.scatterplot(data=world_happiness, x='life_exp', y='happiness_score')
plt.title("Scatterplot of happiness_score vs. life_exp (No Trendline)")
plt.show()

# Scatterplot with a linear trendline and ci set to None
sns.regplot(data=world_happiness, x='life_exp', y='happiness_score', ci=None)
plt.title("Scatterplot of happiness_score vs. life_exp (Linear Trendline)")
plt.show()

# Calculate the correlation between life_exp and happiness_score
cor = world_happiness['life_exp'].corr(world_happiness['happiness_score'])

print(f"Correlation between life_exp and happiness_score: {cor:.2f}")
```


```
summary_stats = fish.groupby("species") ['mass_g'].mean()
print(summary_stats)
```

```
species
Bream    617.828571
Perch    382.239286
Pike     718.705882
Roach    152.050000
Name: mass_g, dtype: float64
```

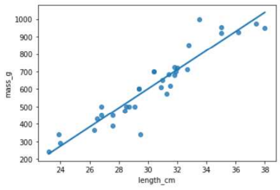
```
from statsmodels.formula.api import ols
mdl_mass_vs_species = ols('mass_g ~ species', data = fish).fit()

print(mdl_mass_vs_species.params)
```

```
Intercept           617.828571
species[T.Perch]   -235.589286
species[T.Pike]     100.877331
species[T.Roach]   -465.778571
dtype: float64
```

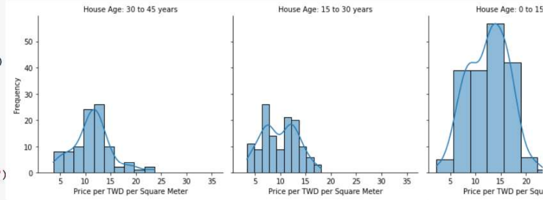
```
mdl_mass_vs_species = ols(
"mass_g ~ species", data=fish).fit()

print(mdl_mass_vs_species.params)
```

```
Intercept           617.828571
species[T.Perch]   -235.589286
species[T.Pike]     100.877331
species[T.Roach]   -465.778571
dtype: float64
```

```
sns.regplot(x="length_cm",
            y ="mass_g",
            data=bream,
            ci=None)
plt.show()
```

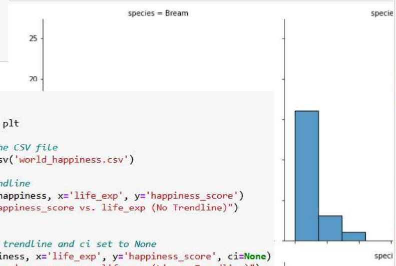



```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have loaded the taiwan_real_estate DataFrame

# Create a histogram with 3 panels (one for each house_age_years group)
g = sns.FacetGrid(taiwan_real_estate, col='house_age_years', col_wrap=3, height=4)
g.map(sns.histplot, 'price_twd_msq', bins=10, kde=True)

# Set labels and titles
g.set_axis_labels("Price per TWD per Square Meter", "Frequency")
g.set_titles("House Age: {col_name} years")

# Show the plot
plt.show()
```

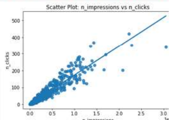


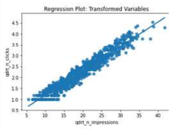

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Load the DataFrame from the CSV file
world_happiness = pd.read_csv('world_happiness.csv')

# Scatterplot of happiness_score versus gdp_per_cap
sns.scatterplot(data=world_happiness, x='gdp_per_cap', y='happiness_score')
plt.title("Scatterplot of happiness_score vs. gdp_per_cap")
plt.show()

# Calculate the correlation between gdp_per_cap and happiness_score
cor_gdp = world_happiness['gdp_per_cap'].corr(world_happiness['happiness_score'])
print(f"Correlation between gdp_per_cap and happiness_score: {cor_gdp:.2f}")

# Add a new column 'log_gdp_per_cap' to world_happiness with the log transformation
world_happiness['log_gdp_per_cap'] = np.log(world_happiness['gdp_per_cap'])

# Seaborn scatterplot of happiness_score versus log_gdp_per_cap
sns.scatterplot(data=world_happiness, x='log_gdp_per_cap', y='happiness_score')
plt.title("Scatterplot of happiness_score vs. log_gdp_per_cap")
plt.show()

# Calculate the correlation between log_gdp_per_cap and happiness_score
cor_log_gdp = world_happiness['log_gdp_per_cap'].corr(world_happiness['happiness_score'])
print(f"Correlation between log_gdp_per_cap and happiness_score: {cor_log_gdp:.2f}")
```


```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

fish = pd.read_csv("fish.csv")

sns.displot(data=fish, x='mass_g', col="species", col_wrap=2, bins=9)

plt.show()
```

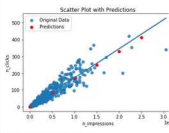



```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.formula.api import ols

ad_conversion = pd.read_csv("ad_conversion.csv")

sns.regplot(x="n_impressions", y="n_clicks", data=ad_conversion, ci=None)
plt.title("Scatter Plot: n_impressions vs n_clicks")
plt.xlabel("n_impressions")
plt.ylabel("n_clicks")
plt.show()

ad_conversion["qdrt_n_impressions"] = ad_conversion["n_impressions"] ** 0.25
ad_conversion["qdrt_n_clicks"] = ad_conversion["n_clicks"] ** 0.25
sns.regplot(x="qdrt_n_impressions", y="qdrt_n_clicks", data=ad_conversion, ci=None)
plt.title("Regression Plot: Transformed Variables")
plt.xlabel("qdrt_n_impressions")
plt.ylabel("qdrt_n_clicks")
plt.show()

# Do the points track the line more closely? Yes.

mdl_click_vs_impression = ols("qdrt_n_clicks ~ qdrt_n_impressions", data=ad_conversion).
print("Model Parameters:")
print(mdl_click_vs_impression.params)

explanatory_data = pd.DataFrame({
    "qdrt_n_impressions": np.arange(0, 3000000, 500000) ** 0.25,
    "n_impressions": np.arange(0, 3000000, 500000)
})
prediction_data = explanatory_data.assign(
    qdrt_n_clicks=mdl_click_vs_impression.predict(explanatory_data)
)

print("Prediction Data:")
print(prediction_data)

# Back transformation:
prediction_data["n_clicks"] = prediction_data["qdrt_n_clicks"] ** 4
sns.regplot(x="n_impressions", y="n_clicks", data=ad_conversion, ci=None, label="Origina
plt.scatter(
    x=prediction_data["n_impressions"],
    y=prediction_data["n_clicks"],
    color="red",
    label="Predictions",
)
plt.title("Scatter Plot with Predictions")
plt.xlabel("n_impressions")
plt.ylabel("n_clicks")
plt.legend()
plt.show()


r_squared = mdl_click_vs_impression.rsquared
print("R-squared:", r_squared)
residuals = ad_conversion["qdrt_n_clicks"] - mdl_click_vs_impression.fittedvalues
rse = np.sqrt(np.sum(residuals ** 2) / (len(ad_conversion) - 2))
print("RSE:", rse)

# Interpret the result:
# - A higher R-squared value indicates a better fit, with 1.0 being a perfect fit.
# - RSE measures the standard deviation of the residuals, indicating how well the model
# - A lower RSE suggests a better fit, as it means the model's predictions are closer to
```






```
Model Parameters:
Intercept             0.071748
qdrt_n_impressions    0.111533
dtype: float64
Prediction Data:
   qdrt_n_impressions  n_impressions  qdrt_n_clicks
0            0.000000              0       0.071748
1           26.591479         500000       3.037576
2           31.622777        1000000       3.598732
3           34.996355        1500000       3.974998
4           37.606031        2000000       4.266063
5           39.763536        2500000       4.506696
```




```
R-squared: 0.9445272817143905
RSE: 0.19690640896875727
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.formula.api import ols

# Assuming you have already loaded your dataset as "taiwan_real_estate"

# Create sqrt_dist_to_mrt_m
taiwan_real_estate["sqrt_dist_to_mrt_m"] = np.sqrt(taiwan_real_estate["dist_to_mrt_m"])

# Create a linear regression model
mdl_price_vs_dist = ols("price_twd_msq ~ sqrt_dist_to_mrt_m", data=taiwan_real_estate).fit()

# Create explanatory_data
explanatory_data = pd.DataFrame({
    "sqrt_dist_to_mrt_m": np.sqrt(np.arange(0, 81, 10) ** 2),
    "dist_to_mrt_m": np.arange(0, 81, 10) ** 2
})

# Create prediction_data by adding a column of predictions to explanatory_data
prediction_data = explanatory_data.assign(
    price_twd_msq=mdl_price_vs_dist.predict(explanatory_data)
)

# Create the initial regplot with square root values
sns.regplot(x="sqrt_dist_to_mrt_m", y="price_twd_msq", data=taiwan_real_estate, ci=None)

# Add a scatter plot for prediction_data with square root values (colored red)
plt.scatter(
    x=prediction_data["sqrt_dist_to_mrt_m"],
    y=prediction_data["price_twd_msq"],
    color="red",
    label="Predictions (sqrt)",
)

# Create a regplot with the original values
sns.regplot(x="dist_to_mrt_m", y="price_twd_msq", data=taiwan_real_estate, ci=None)

# Add a scatter plot for prediction_data with original values (colored red)
plt.scatter(
    x=prediction_data["dist_to_mrt_m"],
    y=prediction_data["price_twd_msq"],
    color="red",
    label="Predictions (original)",
)

# Show the updated plot
plt.legend()
plt.show()
```