Note: Mid Term Exam

Type: Lecture Reviewed: No

Data Science for workflow

1. Data Collection and Storage -> Collect, Storage.
2. Data Preparation -> Cleaning, Organized, Reformat.
3. Exploration & Visualization -> Image, Graph.
4. Experimentation & Prediction -> Estimate, Forecast.

What do we need for machine learning?

1. A well-defined question.
2. A set of example data.
3. A new set of data to use our algorithm on.

Traditional machine learning -> Prediction, Cluster.

IOT -> Physical device.

Deep learning -> Image recognition, Language.

Data Engineer -> **Data Collection and Storage**.

- Build data flow, pipeline, storage system.
- SQL, Python, Java.

Data Analyst -> **Data Preparation and Exploration & Visualization**.

- SQL, Excel.

Data Scientist -> **Experimentation & Prediction and Data Preparation**.

- Statistical, traditional machine learning.
- Python, R.

Machine Learning Scientist -> **Prediction and Data Preparation**.

- Deep Learning, Prediction, Classification.
- Python, R.

Open data -> API, Government data, Public.

- Free data.

Company data -> Survey, customer data, logistics data, web events.

- data-driven decisions, not open data.

Quantitative data -> counted, measured, and numbers.

Qualitative data -> observed but not measured.

Unstructured –> text, video and audio files that are stored in database.

Structured –> Relational database such as MySQL.

Data Type	Query Language
Document Database	NoSQL
Relational Database	SQL

Data pipelines -> **Transform & Load**.

- How de we keep it organized and easy to use?
  - Joining data sources into one data set.
  - Converting data structures to fit database schemas.
  - Removing irrelevant data.

Why prepare data?

- Real data is messy(Tidiness), Missing data, and Remove duplicates

Exploratory Data Analysis

- formulating hypotheses and assessing its main characteristics, with a strong emphasis on visualization.

What are experiments in data science?

- Experiments help drive decisions and draw conclusions.

1. Form a question
2. Form a hypothesis
3. Collect data
4. Test the hypothesis with statistical test
5. Interpret results

What is A/B Testing?

- Testing A case and B case and see which one produce better result.

Time series data

- Stock, gas price | Unemployment, heart, inflation rate | temperature | Height.

Forecasting time series will tell us about

- How much rainfall will we get next month?, Will traffic ease up in the next half hour?
- How will the stock market move in the next six hours?, What will be earth's population in 20 years?

How do we know the model is good?

- Data has features and labels.

What is supervised machine learning?

- Predictions from data with labels and features.
- Recommendation systems.
- Recognizing hand-written digits

Unsupervised machine learning -> **Clustering**

- Clustering is a set of machine learning algorithms that divide data into categories, called clusters.
- Clustering can help us see patterns in messy datasets.
- Machine Learning Scientists use clustering to divide customers into segments, images into categories, or behaviors into typical and anomalous.

Histogram plot -> distribution Scatter plot -> see two correlation between 2 subject Line plot -> see the trend of 2 subject

```
plt.hist(life_exp, bins=5) # for histogram plot with bins
plt.plot(x=, y=) # for line plot

# .loc[Start row: Stop row, Start column: Stop column]
brics.loc[‘BR’:'CH','country':'area']
# .iloc use index instead of string to specify row and column
brics.iloc[0:4,0:2]

plt.xscale("log")
plt.xlabel("")
plt.ylabel("")
plt.title("")

xtick_val = [1000, 10000, 100000] # for actual value
xtick_lab = ["1k", "10k", "100k"] # for text
plt.xticks(xtick_val, xtick_lab)
plt.yticks()

more_than_200 = brics['population']>=200 # output set of booleans
brics[more_than_200][['country', 'population']] # more than 2000 and showing only country a

brics[(brics['population']>1000) | (brics['area'] < 8)][['capital']] # doing comparison and
```

	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False
IN	18	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	45	Egypt	True

```
for lab,row in cars.iterrows():
    print(lab + ": " + str(row['cars_per_cap']))
    # lab -> index
```

Output:

```
US: 809
AUS: 731
JAP: 588
IN: 18
RU: 200
MOR: 70
EG: 45
```

```
# Adding new row to data frame
for lab,row in cars.iterrows():
    cars.loc[lab, "COUNTRY"] = row['country'].upper()
    # "COUNTRY" -> name setting for row
```

OR

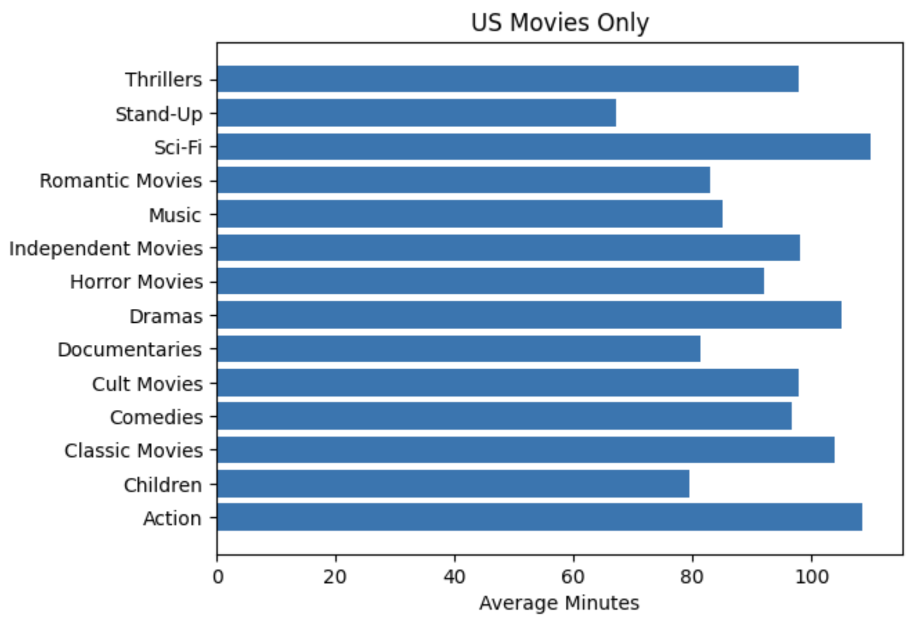
```
cars['COUNTRY'] = cars['country'].apply(str.upper)
```

```
cars['name_length'] = cars['country'].apply(len)
```

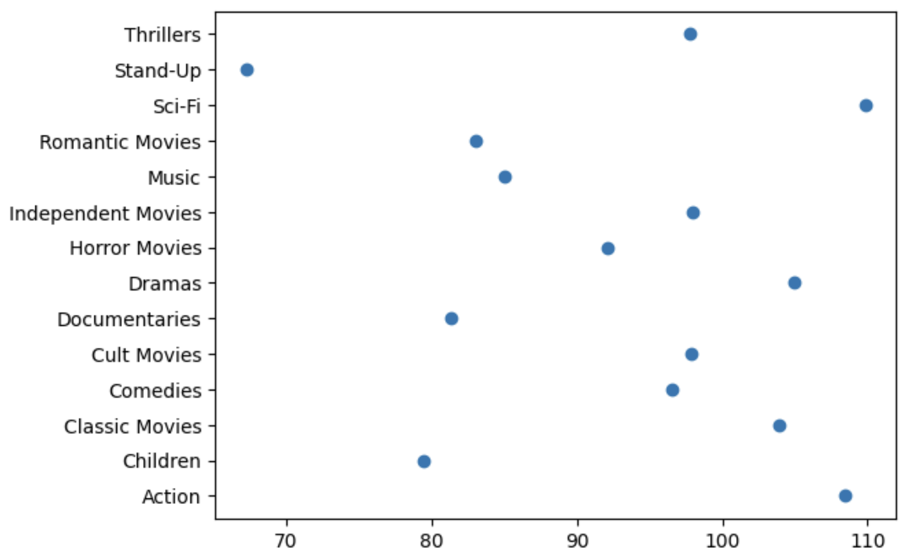
```
netflix_df.query('type == "Movie"')
netflix_df_movies_only[(netflix_df_movies_only["country"] == "United States")]

long_genre = netflix_us_only.groupby("genre")[["release_year", "duration"]].mean() # mean o
```

genre	release_year	duration
Action	2008.922449	108.428571
Children	2011.917241	79.441379
Classic Movies	1968.404762	103.880952
Comedies	2012.445122	96.576220
Cult Movies	1990.111111	97.888889
Documentaries	2016.128463	81.372796
Dramas	2012.984085	104.965517
Horror Movies	2014.414414	92.117117
Independent Movies	2016.000000	98.000000
Music	2016.600000	85.000000
Romantic Movies	2017.500000	83.000000
Sci-Fi	2011.833333	109.833333
Stand-Up	2014.449275	67.256039
Thrillers	2013.300000	97.775000

```
x = long_genre.index # need index for plotting
plt.barh(x, long_genre.duration) # horizontal bar
```




```
plt.scatter(long_genre.duration, x) # index on y-axis on scatter
```




```
release_year.groupby('country').count()
```

```
# ascending = True -> low to high -> ascending order
# ascending = False -> high to low -> descending order
```

```
release_year.groupby('country').count().sort_values(by=['title'], ascending=False).head(10)
```

```python
# must be number for each variable
india = count.iloc[1][0]
uk = count.iloc[2][0]
canada = count.iloc[3][0]

y = np.array([india, canada, uk])
labels = ["India", "Canada", "United Kingdom"]

plt.pie(y, labels=labels)

a = []
b = []
for lab, row in count.iterrows():
    a.append(row['title']) # number
    b.append(lab) # index

plt.figure(figsize=(15, 15))

def func(pct, allvals):
    absolute = int(np.round(pct / 100.0 * np.sum(allvals)))
    return f"{pct:.1f}%\n({absolute:d})"

# Pie Chart
plt.pie(a, labels=b, autopct=lambda x: func(x, a), pctdistance=0.85, startangle=90)

# draw circle
centre_circle = plt.Circle((0, 0), 0.50, fc='white')
fig = plt.gcf()

# Adding Circle in Pie chart
fig.gca().add_artist(centre_circle)

plt.title('Number of titles released by top 10 countries')

# sorting many column
homelessness.sort_values(by=["region", "family_members"], ascending=[True, False])

mojave_state = ['Arizona', 'California', 'Nevada', "Utah"]
mojave_homelessness = homelessness[homelessness['state'].isin(mojave_state)]

homelessness['individuals'] + homelessness['family_members']

# Dropping values
store_types = sales.drop_duplicates(subset=["store", "type"])

sales[sales['is_holiday'] == True]
holiday_dates.drop_duplicates(subset=["date"])
```

```
store_types["type"].value_counts()
store_types["type"].value_counts(normalize=True) # show in percentage of data in the data f

store_depts["department"].value_counts(sort=True) # sort=True make into descending

temperature.set_index(["country", "city"])
temperature.reset_index() # index 0, 1, 2, n

# index two value while "country" and "city" are index
row_to_keep = [("Brazil", "Rio De Janeiro"), ("Pakistan", "Lahore")]
temperature_ind.loc[row_to_keep]

sns.histplot(data=unemployment, x="2011", bins=20)

unemployment['2012'].min(), unemployment['2012'].max()
sns.boxplot(data=unemployment, x=unemployment['2012'], y=unemployment['continent'])

unemployment[["2010", "2011", "2012", "2013", "2014", "2015", "2016", "2017", \
"2018", "2019", "2020", "2021"] ].agg(["mean", "std"])

unemployment.groupby('continent')[["2010", "2011", "2012", "2013", "2014", \
"2015", "2016", "2017", "2018", "2019", "2020", "2021", ]].agg(["mean", "std"])
```

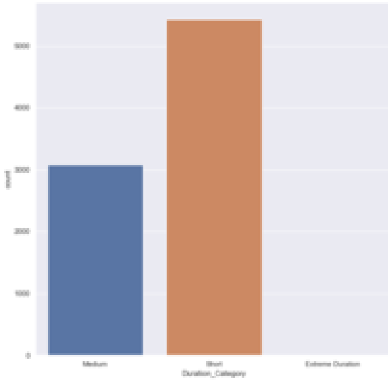
	2010		2011		2012		2013		2014		...	2017		2018	
	mean	std	mean	std	mean	std	mean	std	mean	std	...	mean	std	mean	std
continent															
Africa	9.343585	7.411259	9.369245	7.401556	9.240755	7.264542	9.132453	7.309285	9.121321	7.291359	...	9.284528	7.407620	9.237925	7.358425
Asia	6.240638	5.146175	5.942128	4.779575	5.835319	4.756904	5.852128	4.668405	5.853191	4.681301	...	6.171277	5.277201	6.090213	5.409128
Europe	11.008205	6.392063	10.947949	6.539538	11.325641	7.003527	11.466667	6.969209	10.971282	6.759765	...	8.359744	5.177845	7.427436	4.738206
North America	8.663333	5.115805	8.563333	5.377041	8.448889	5.495819	8.840556	6.081829	8.512222	5.801927	...	7.391111	5.326446	7.281111	5.253180
Oceania	3.622500	2.054721	3.647500	2.008466	4.103750	2.723118	3.980000	2.640119	3.976250	2.659205	...	3.872500	2.492834	3.851250	2.455893
South America	6.870833	2.807058	6.518333	2.801577	6.410833	2.936508	6.335000	2.808780	6.347500	2.834332	...	7.281667	3.398994	7.496667	3.408856

```
unemployment.groupby("continent").agg(
    mean_rate_2021 = ("2021", "mean"),
    std_rate_2021= ("2021", "std")
)
```

continent	mean_rate_2021	std_rate_2021
Africa	10.473585	8.131636
Asia	6.906170	5.414745
Europe	7.414872	3.947825
North America	9.155000	5.076482
Oceania	4.280000	2.671522
South America	9.924167	3.611624

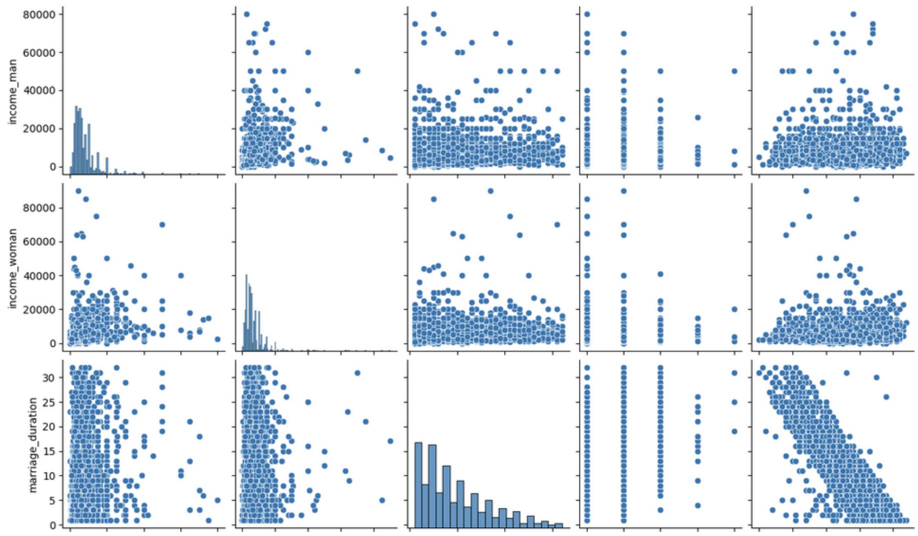
```
sns.barplot(data=unemployment, x="continent", y="2021")

airline.isna().sum() # print number of missing values.

threshold = len(airline) * 0.05
col_to_drop = airline.columns[airline.isna().sum() <= threshold]
airline.dropna(subset=col_to_drop, inplace=True)

airline.groupby("Airline")["Price"].median().to_dict() # output: {'Air Asia': 5192.0, 'Air
airline["Price"] = airline["Price"].fillna(airline["Airline"].map(a)) # fillna on price by

# Filter the DataFrame for object columns
non_numeric = airline.select_dtypes("object")
# Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route', 'Dep_Time',
# 'Arrival_Time', 'Duration', 'Total_Stops', 'Additional_Info'], dtype='object')

for col in non_numeric.columns:
    print(f"Number of unique values in {col} column: ", non_numeric[col].nunique())

duration_category = ["Short", "Medium", "Long"] # For label in the data frame
short_flights = "0h|1h|2h|3h|4h"
medium_flights = "5h|6h|7h|8h|9h"
long_flights = "10h|11h|12h|13h|14h|16h"
conditions = [
    (airline["Duration"].str.contains(short_flights)), # Short
    (airline["Duration"].str.contains(medium_flights)), # Medium
    (airline["Duration"].str.contains(long_flights)), # Long
```

```python
]
# Duration_Category output: Medium 5000, Short 1000
airline["Duration_Category"] = np.select(
    conditions, duration_category, default="Extreme Duration"
)
# y-axis -> will be the count of occurrence; by just mentioning x-axis
sns.countplot(data=airline, x=airline["Duration_Category"])
```

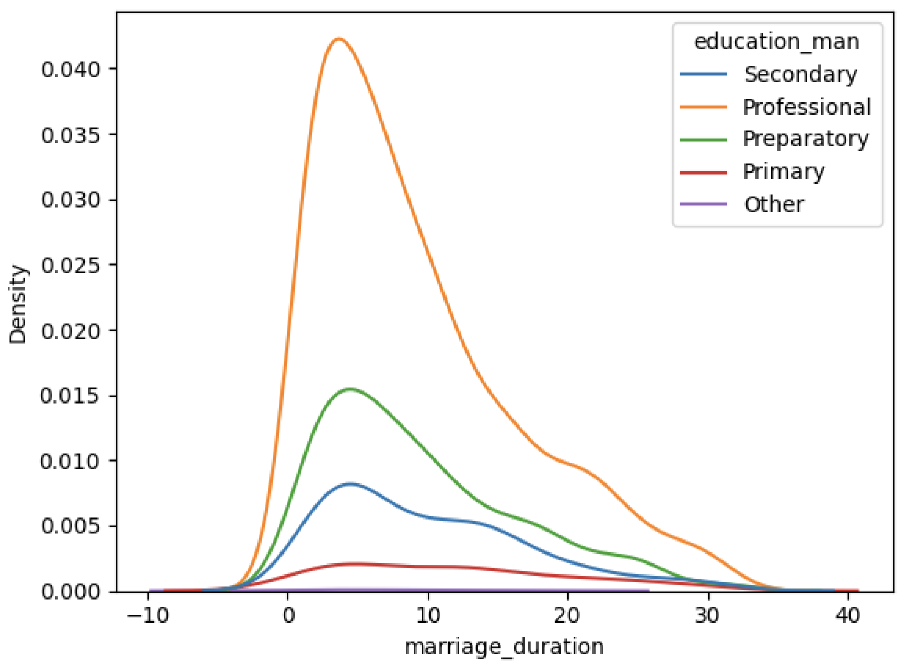


```python
airline["Duration"] = airline["Duration"].str.replace("h", ".")
airline["Duration"] = airline["Duration"].astype(float) # change type

# x.mean(), x.median()
airline.groupby("Airline")["Price"].transform(lambda x: x.std())

price_seventy_fifth = airline["Price"].quantile(0.75)
price_twenty_fifth = airline["Price"].quantile(0.25)
price_iqr = price_seventy_fifth - price_twenty_fifth

upper = price_seventy_fifth + (1.5 * price_iqr)
lower = price_twenty_fifth - (1.5 * price_iqr)

airline[(airline["Price"] < lower) | (airline["Price"] > upper)] # outlier

no_outlier = airline[(airline["Price"] > lower) & (airline["Price"] < upper)]
no_outlier["Price"].describe()
```

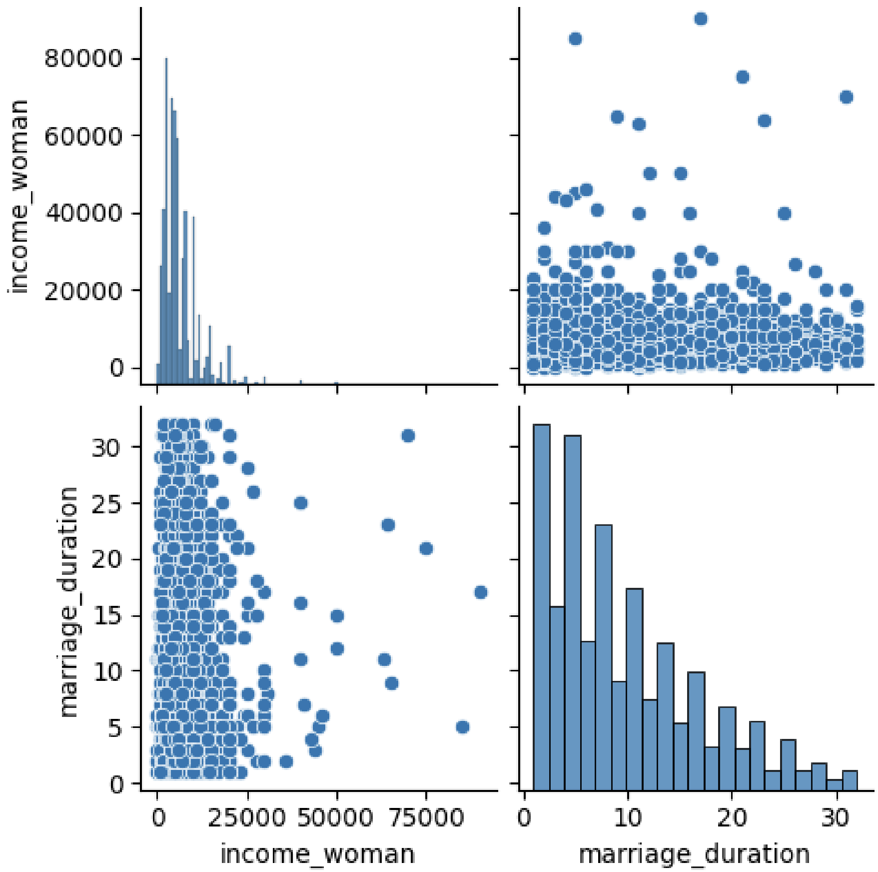
```python
divorce.dtypes # check types
pd.read_csv("data/h.csv", parse_dates=["date_of_response"],)
divorce["marriage_date"] = pd.to_datetime(divorce["marriage_date"]) # convert into date type
divorce["marriage_date"].dt.month # get month; dt.weekday, dt.year
sns.lineplot(
    data=divorce, x=divorce["marriage_month"], y=divorce["marriage_duration"]
)
sns.pairplot(data=divorce) # many plot
```




```
# hue="education_woman" for coloring in sns
sns.kdeplot(data=divorce, x="marriage_duration", hue="education_man") # cut=0, cumulative=T
```




```
sns.pairplot(data=divorce, vars=["income_woman", "marriage_duration"])
```




```
salary_rupee_usd["Job_Category"].value_counts()
```

```
Job_Category
Data Science         113
Data Engineering     111
Data Analytics        92
Machine Learning      49
Other                 28
Managerial            14
Name: count, dtype: int64
```

```
# check correlation between
pd.crosstab(salary_rupee_usd["Job_Category"], salary_rupee_usd["Company_Size"])
```

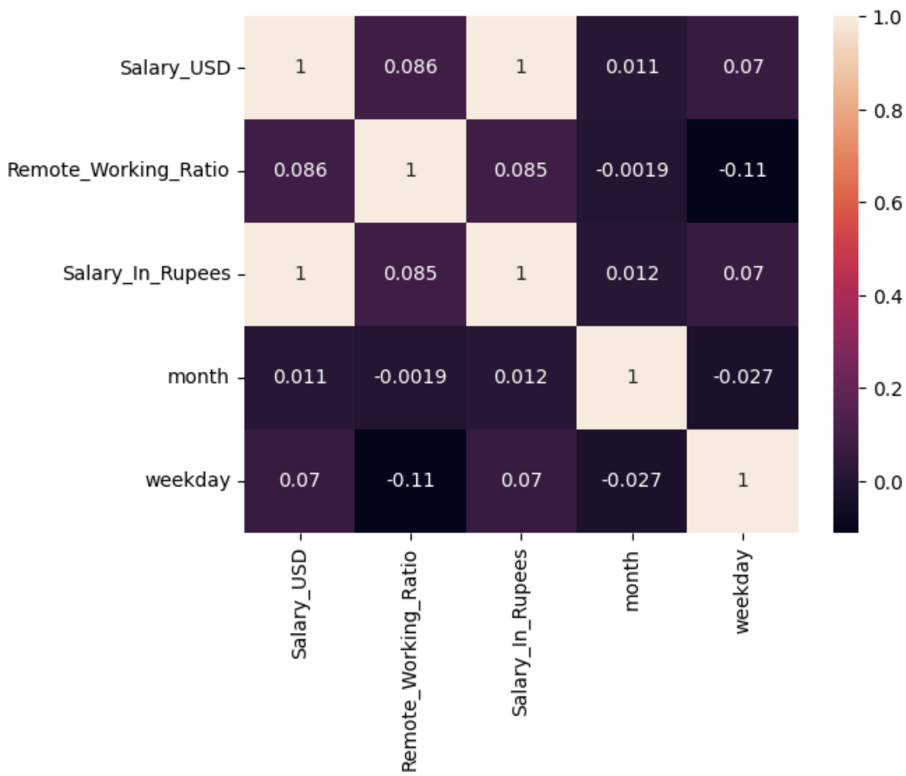
Company_Size	L	M	S
Job_Category			
Data Analytics	23	61	8
Data Engineering	28	72	11
Data Science	38	59	16
Machine Learning	17	19	13
Managerial	5	8	1
Other	13	9	6

```
pd.crosstab(salary_rupee_usd["Job_Category"], salary_rupee_usd["Company_Size"],
values=salary_rupee_usd["Salary_USD"], aggfunc="mean") # check by salary(mean)
```

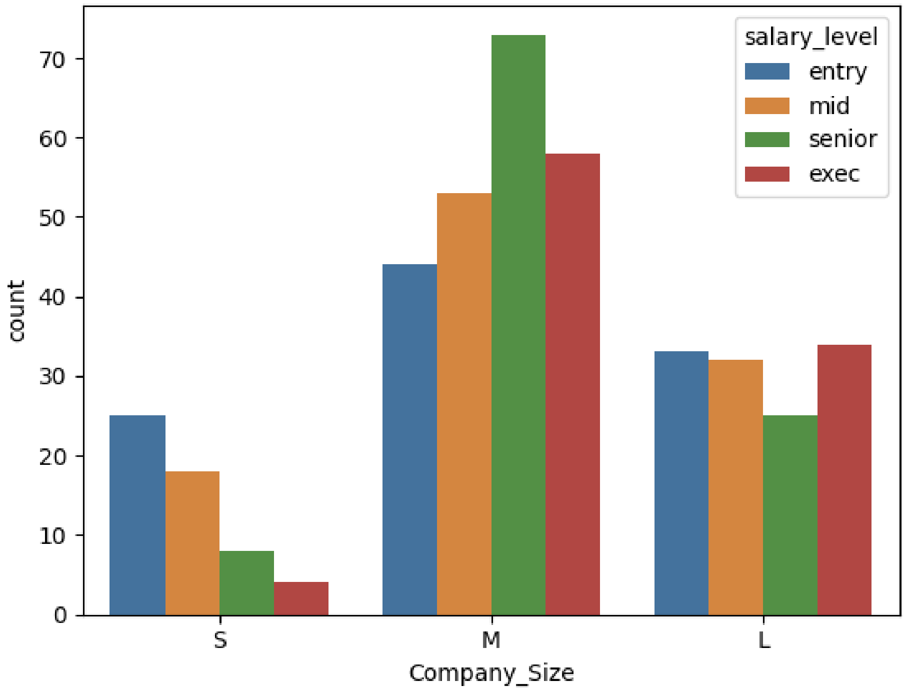
Company_Size	L	M	S
Job_Category			
Data Analytics	112851.749217	95912.685246	53741.877000
Data Engineering	118939.035000	121287.060500	86927.136000
Data Science	96489.520105	116044.455864	62241.749250
Machine Learning	140779.491529	100794.236842	78812.586462
Managerial	190551.448800	150713.628000	31484.700000
Other	92873.911385	89750.578667	69871.248000

```
pd.to_datetime(salaries["date_of_response"], format="%d/%m/%Y") # Change format
```

```
sns.heatmap(salaries[["Salary_USD", "Remote_Working_Ratio", "Salary_In_Rupees",\
                      "month", "weekday"]].corr(), annot=True)
```



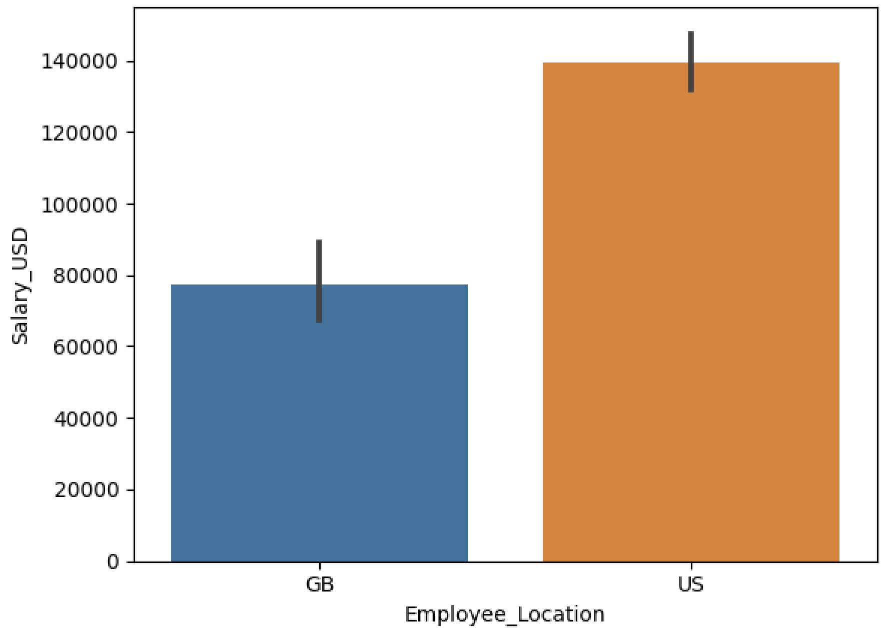

```
twenty_fifth = salaries["Salary_USD"].quantile(0.25)
salaries_median = salaries["Salary_USD"].median()
seventy_fifth = salaries["Salary_USD"].quantile(0.75)
largest = salaries["Salary_USD"].max()

salary_labels = ["entry", "mid", "senior", "exec"]
salary_ranges = [0, twenty_fifth, salaries_median, seventy_fifth, largest]
# bins -> find the range and labels -> labels it with salary_labels
salaries["salary_level"] = pd.cut(
    salaries["Salary_USD"], bins=salary_ranges, labels=salary_labels
)
sns.countplot(data=salaries, x="Company_Size", hue="salary_level")
```



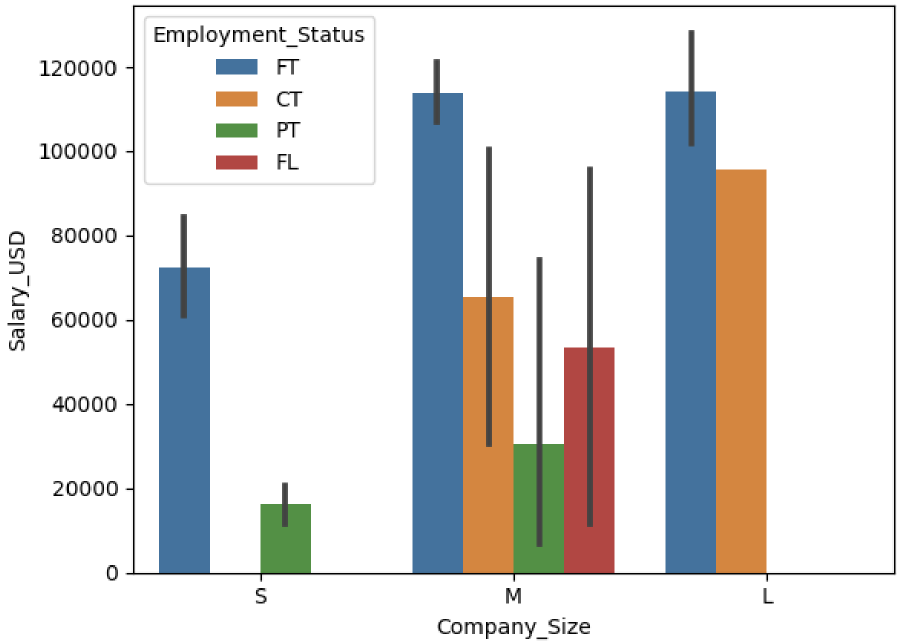

```
usa_and_gb = salaries[salaries["Employee_Location"].isin(["US", "GB"])]
```

```
sns.barplot(data=usa_and_gb, x="Employee_Location", y="Salary_USD")
```

```
data = salaries[salaries["Employee_Location"].isin(["US", "GB"])]
sns.barplot(data=data, x="Employee_Location", y="Salary_USD")
```




```
usa_and_gb = salaries["Employee_Location"].isin(["US", "GB"])
sns.barplot(
    data=salaries, x="Company_Size", y="Salary_USD", hue="Employment_Status"
)
```




```
np.logical_and(brics['area']>8, brics['area']<10)
np.logical_or(brics['population']>1000, brics['area']<3)

# find highest average temperature
temperature[temperature["avg_temp_c"] == temperature["avg_temp_c"].max()]

during_year_thailand = thailand[
    (thailand["date"] >= "2005-01-01") & (thailand["date"] <= "2010-01-01")
]
print(
    f"The avg. temp of Thailand during 2005-2010 is {round(during_year_thailand['avg_temp_c'
)
```