Hypothesis Testing Case Study | Yulu

Context: Yulu is India's leading micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India, Yulu provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting.

Yulu zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient!

Objective: Yulu has recently suffered considerable dips in its revenues. They have contracted a consulting company to understand the factors on which the demand for these shared electric cycles depends. Specifically, they want to understand the factors affecting the demand for these shared electric cycles in the Indian market.

Column Description:

- datetime: datetime
- season: season (1: spring, 2: summer, 3: fall, 4: winter)
- holiday: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- workingday: if day is neither weekend nor holiday is 1, otherwise is 0.
- weather:
    1. Clear, Few clouds, partly cloudy, partly cloudy
    2. Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    3. Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    4. Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp: temperature in Celsius
- atemp: feeling temperature in Celsius
- humidity: humidity
- windspeed: wind speed
- casual: count of casual users
- registered: count of registered users
- count: count of total rental bikes including both casual and registered

Importing Libraries

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

from scipy import stats

plt.rcParams['figure.figsize'] = (8, 6)
plt.rcParams['figure.dpi'] = 150

cmap_seagreen = sns.light_palette("seagreen", as_cmap=True)
sns.set_theme(palette="pastel")
```

Loading CSV File

In [ ]:
```
df = pd.read_csv("/content/drive/MyDrive/Scaler/Yulu Case Study/yulu_bike_sharing.csv")
print(df.shape, df.isna().sum().sum(), df.duplicated().sum(), df.nunique().sort_values(), sep="\n\n")
df.head()
```

```
(10886, 12)

0

0

holiday           2
workingday        2
season            4
weather           4
windspeed        28
temp             49
atemp            60
humidity         89
casual          309
registered      731
count           822
datetime      10886
dtype: int64
```

Out[ ]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

In [ ]:
```
#print(df.info(), "\n\n")
df.describe()
```

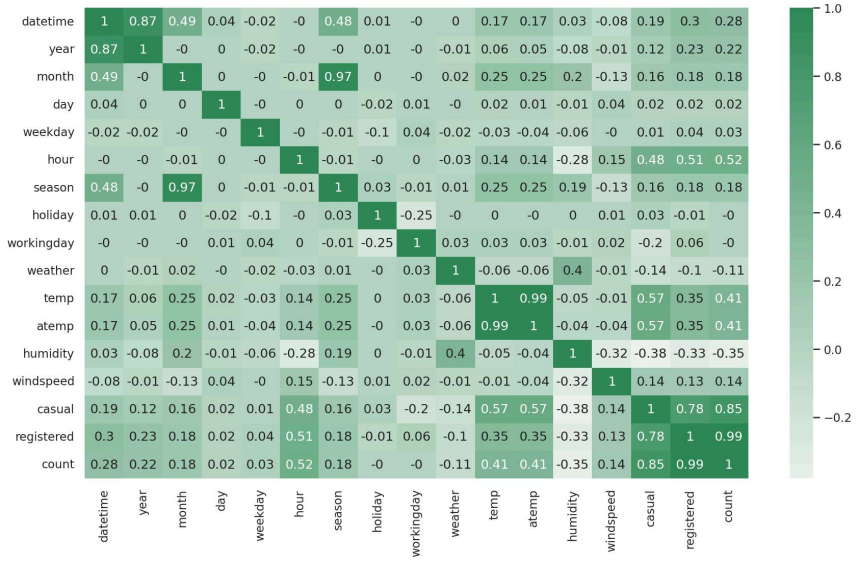
Out[ ]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

Data Cleaning and Feature Extraction

In [ ]:
```
# Updating data type before checking corelation
df["datetime"] = pd.to_datetime(df["datetime"])

# insert() was used to place the columns in the beginning
df.insert(1, 'year', df['datetime'].dt.year)
df.insert(2, 'month', pd.to_numeric(df['datetime'].dt.strftime('%m')))
df.insert(3, 'day', df['datetime'].dt.day)
df.insert(4, 'weekday', pd.to_numeric(df['datetime'].dt.strftime('%w')))
df.insert(5, 'hour', df['datetime'].dt.hour)


df["workingday"] = df["workingday"] == 1
df["holiday"] = df["holiday"] == 1
```

In [ ]:
```
# Checking corelation before modifing numerical columns

data = round(df.corr(method='spearman'), 2)

plt.figure(figsize=(14, 8))
sns.heatmap(data, annot=True, cmap=cmap_seagreen)
plt.show()
```



In [ ]:
```
categorical_columns = ["holiday", "workingday", "season", "weather"]
for col in categorical_columns:
    print(df[col].value_counts().reset_index(), end="\n\n")
```

```
   holiday  count
0    False  10575
1     True    311

   workingday  count
0        True   7412
1       False   3474

   season  count
0       4   2734
1       2   2733
2       3   2733
3       1   2686

   weather  count
0        1   7192
1        2   2834
2        3    859
3        4      1
```

In [ ]:
```
# season mapping
season_mapping = {1: "Spring", 2: "Summer", 3: "Fall", 4: "Winter"}
df["season"] = df["season"].map(season_mapping)

# weather mapping
# The weather with only one instance can be grouped with other similar weather condition
weather_mapping = {
    1: "Clear/Partly cloudy", #Clear, Few clouds, partly cloudy
    2: "Mist/Cloudy", #Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    3: "Light Snow/Rain + Thunder", # Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
```

```
    # Merged because there is only one instance of this weather:
    4: "Light Snow/Rain + Thunder" # Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
}
df["weather"] = df["weather"].map(weather_mapping)

# holiday mapping
def holiday_workingday(row):
    if (row["workingday"] == True) & (row["holiday"] == True):
        return "NA"
    elif (row["workingday"] == False) & (row["holiday"] == True):
        return "Holiday"
    elif (row["workingday"] == True) & (row["holiday"] == False):
        return "Work Day"
    elif (row["workingday"] == False) & (row["holiday"] == False):
        return "Weekend"
df["day_status"] = df.apply(holiday_workingday, axis=1)

# hour mapping
bins = [0, 3, 6, 9, 12, 15, 18, 21, 24]
labels = [
    "Midnight Hours", "Dawn", "Early Morning",
    "Morning", "Afternoon", "Late Afternoon",
    "Evening", "Night"
]
df["time_of_day"] = pd.cut(df["hour"], bins=bins, labels=labels, right=False)
```

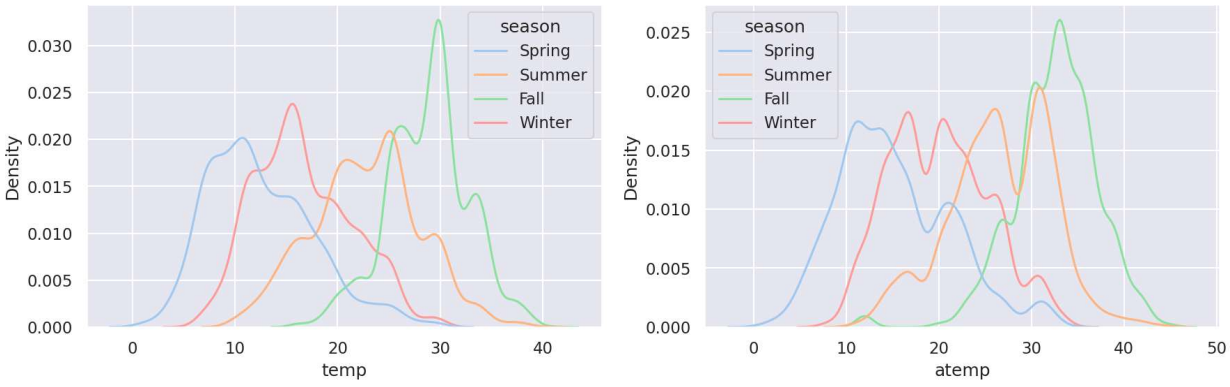
In [ ]:
```
df.head()
```

Out[ ]:

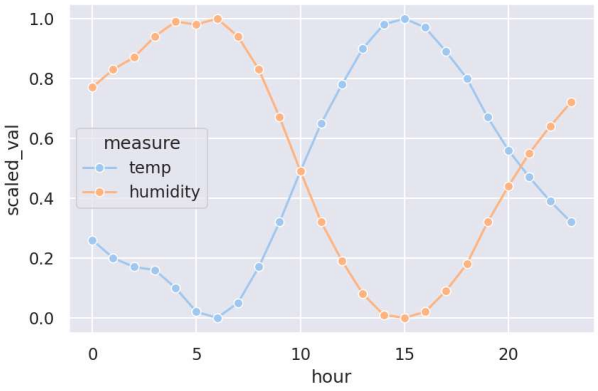
	datetime	year	month	day	weekday	hour	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	day_status	time_of_day
0	2011-01-01 00:00:00	2011	1	1	6	0	Spring	False	False	Clear/Partly cloudy	9.84	14.395	81	0.0	3	13	16	Weekend	Midnight Hours
1	2011-01-01 01:00:00	2011	1	1	6	1	Spring	False	False	Clear/Partly cloudy	9.02	13.635	80	0.0	8	32	40	Weekend	Midnight Hours
2	2011-01-01 02:00:00	2011	1	1	6	2	Spring	False	False	Clear/Partly cloudy	9.02	13.635	80	0.0	5	27	32	Weekend	Midnight Hours
3	2011-01-01 03:00:00	2011	1	1	6	3	Spring	False	False	Clear/Partly cloudy	9.84	14.395	75	0.0	3	10	13	Weekend	Dawn
4	2011-01-01 04:00:00	2011	1	1	6	4	Spring	False	False	Clear/Partly cloudy	9.84	14.395	75	0.0	0	1	1	Weekend	Dawn

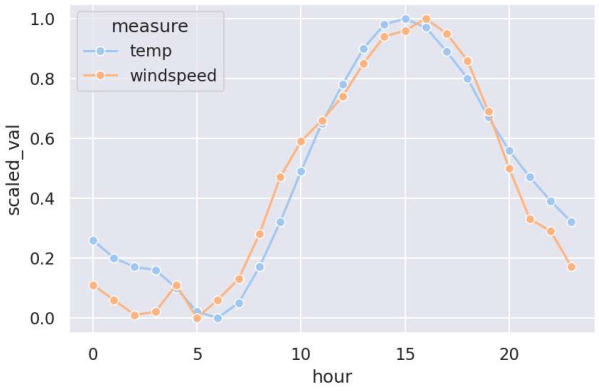
Exploratory Data Analysis

In [ ]:
```
fig = plt.figure(figsize=(14,4))

ax1 = fig.add_subplot(1,2,1)
sns.kdeplot(data=df, x=round(df["temp"]), hue="season", ax=ax1)

ax2 = fig.add_subplot(1,2,2)
sns.kdeplot(data=df, x=round(df["atemp"]), hue="season", ax=ax2)

plt.show()
```

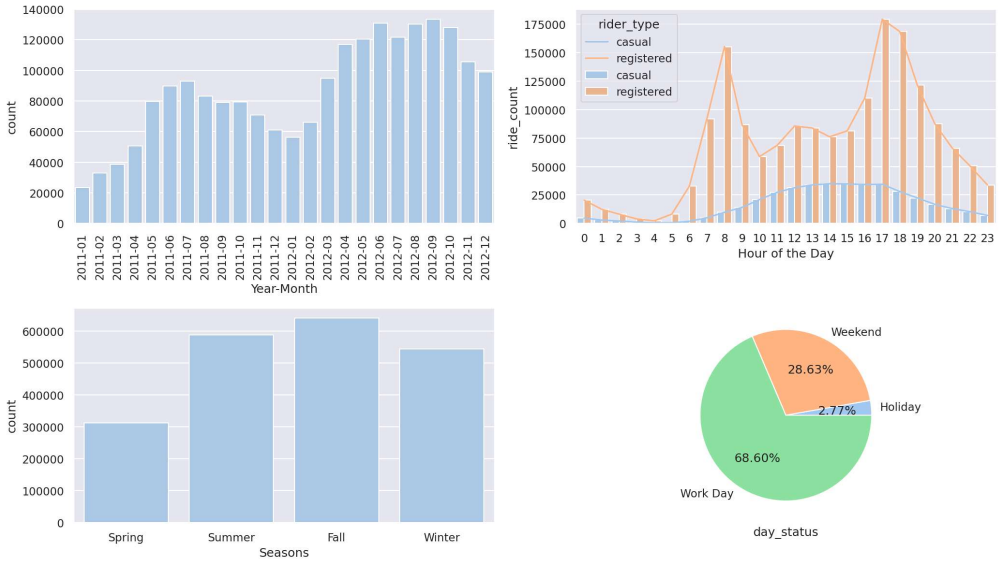


In [ ]:
```
def minMaxScaler(series, x=2):
    return round((series - series.min()) / (series.max() - series.min()), x)

fig = plt.figure(figsize=(14,4))

s1 = minMaxScaler(df.groupby("hour")["temp"].mean())
s2 = minMaxScaler(df.groupby("hour")["humidity"].mean())
data = pd.concat([s1,s2], axis=1).reset_index()
data = pd.melt(data, id_vars="hour", var_name="measure", value_name="scaled_val")

ax1 = fig.add_subplot(1,2,1)
sns.lineplot(data, x="hour", y="scaled_val", hue="measure", marker="o", ax=ax1)

s3 = minMaxScaler(df.groupby("hour")["windspeed"].mean())
data = pd.concat([s1,s3], axis=1).reset_index()
data = pd.melt(data, id_vars="hour", var_name="measure", value_name="scaled_val")

ax2 = fig.add_subplot(1,2,2)
sns.lineplot(data, x="hour", y="scaled_val", hue="measure", marker="o", ax=ax2)

plt.show()
```

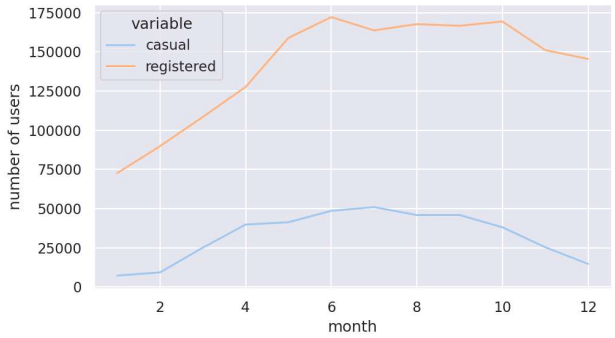


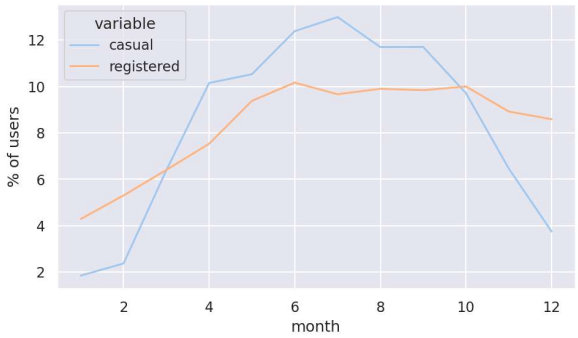


In [ ]:
```
fig = plt.figure(figsize = (14, 8))


month_year = df['datetime'].dt.strftime('%Y-%m')
# Remember when passing series to sns plots, it will treat the index as the x-axis and the values as the y-axis
# If I reset_index() after sum(), then i will have to explicetly specify both the axes
ax1 = fig.add_subplot(2,2,1)
sns.barplot(df.groupby(month_year)["count"].sum(), ax=ax1)
ax1.tick_params(axis='x', rotation=90)
ax1.set_xlabel("Year-Month")

data = df.melt(id_vars="hour", value_vars=["casual", "registered"], var_name="rider_type", value_name="ride_count")
data = data.groupby(["hour", "rider_type"])["ride_count"].sum().reset_index()
ax2 = fig.add_subplot(2,2,2)
sns.lineplot(data=data, x="hour", y="ride_count", ax=ax2, hue="rider_type")
sns.barplot(data=data, x="hour", y="ride_count", ax=ax2, hue="rider_type")
ax2.set_xlabel("Hour of the Day")

ax3 = fig.add_subplot(2,2,3)
sns.barplot(df.groupby("season")["count"].sum(), ax=ax3, order=["Spring", "Summer", "Fall", "Winter"])
ax3.set_xlabel("Seasons")

ax4 = fig.add_subplot(2,2,4)
day_status_data = df.groupby("day_status")["count"].sum()
ax4.pie(day_status_data, labels=day_status_data.index, autopct='%1.2f%%')
ax4.set_xlabel("day_status")

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

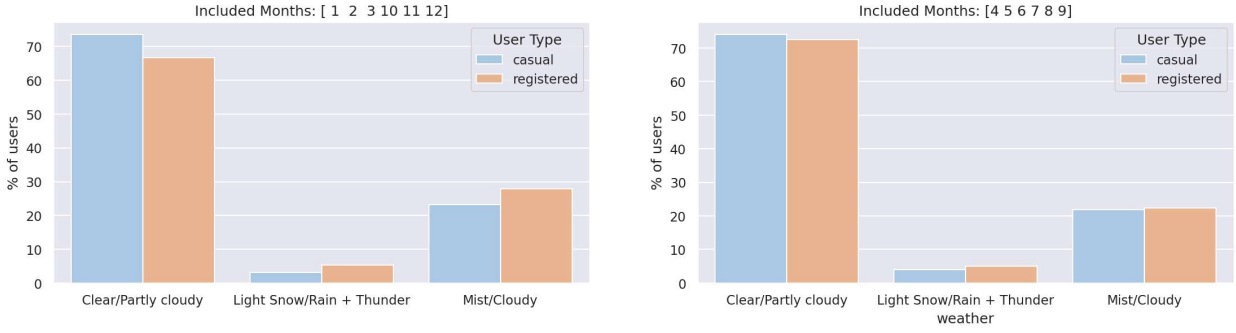


In [ ]:
```
data = df.groupby(["month"])[["casual", "registered"]].sum().reset_index()
data = data.melt(id_vars=["month"])
fig = plt.figure(figsize=(16,4))

ax1 = fig.add_subplot(1,2,1)
sns.lineplot(data, x="month", y="value", hue="variable", ax=ax1)
ax1.set_ylabel("number of users")

data = df.groupby(["month"])[["casual", "registered"]].sum().reset_index()
data["casual"] = data["casual"] / data["casual"].sum() * 100
data["registered"] = data["registered"] / data["registered"].sum() * 100
data = data.melt(id_vars=["month"])

ax2 = fig.add_subplot(1,2,2)
sns.lineplot(data, x="month", y="value", hue="variable", ax=ax2)
ax2.set_ylabel("% of users")

plt.show()
```

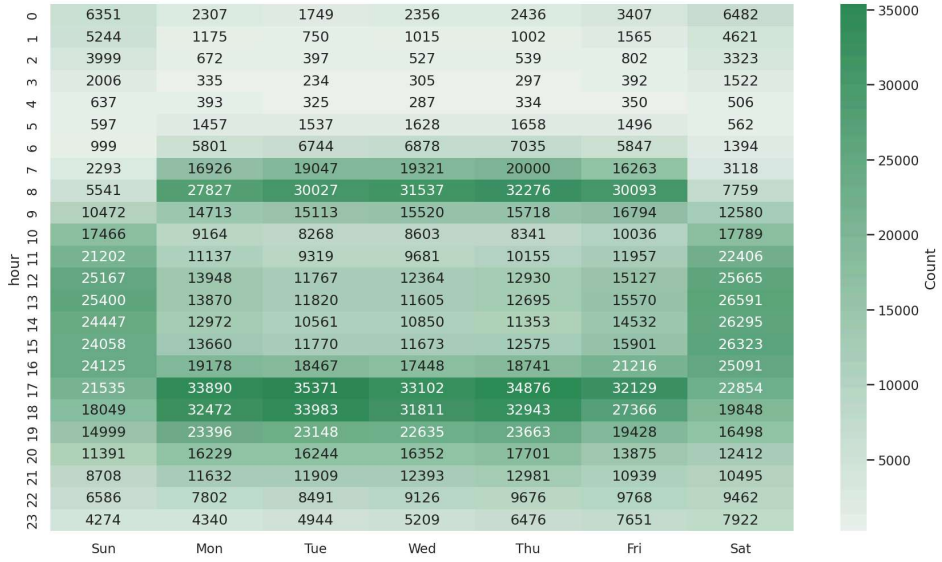




In [ ]:
```
data = df.loc[~df["month"].between(4,9)]
months = data["month"].unique()
data = data.groupby(["weather"])[["casual", "registered"]].sum().reset_index()
data["casual"] = data["casual"] / data["casual"].sum() * 100
data["registered"] = data["registered"] / data["registered"].sum() * 100
data = data.melt(id_vars=["weather"])
data.columns = ["weather", "User Type", "% of users"]

fig = plt.figure(figsize=(18, 4))

ax1 = fig.add_subplot(1,2,1)
sns.barplot(data, x="weather", y="% of users", hue="User Type", ax=ax1)
ax1.set_title(f"Included Months: {months}")
ax1.set_xlabel("")

data = df.loc[df["month"].between(4,9)]
months = data["month"].unique()
data = data.groupby(["weather"])[["casual", "registered"]].sum().reset_index()
data["casual"] = data["casual"] / data["casual"].sum() * 100
data["registered"] = data["registered"] / data["registered"].sum() * 100
data = data.melt(id_vars=["weather"])
data.columns = ["weather", "User Type", "% of users"]

ax2 = fig.add_subplot(1,2,2)
sns.barplot(data, x="weather", y="% of users", hue="User Type", ax=ax2)
ax2.set_title(f"Included Months: {months}")
ax1.set_xlabel("")

plt.show()
```



In [ ]:
```
#data = df.loc[df["day_status"] == "Holiday on Weekend"]
data = df.groupby(['weekday', 'hour'])['count'].sum().reset_index()
data = data.pivot(index='hour', columns='weekday', values='count')
data.columns = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']

plt.figure(figsize=(14,8))
sns.heatmap(data, cmap=cmap_seagreen, annot=True, fmt='g', cbar_kws={'label': 'Count'})
plt.show()
```



hour	Sun	Mon	Tue	Wed	Thu	Fri	Sat
0	6351	2307	1749	2356	2436	3407	6482
1	5244	1175	750	1015	1002	1565	4621
2	3999	672	397	527	539	802	3323
3	2006	335	234	305	297	392	1522
4	637	393	325	287	334	350	506
5	597	1457	1537	1628	1658	1496	562
6	999	5801	6744	6878	7035	5847	1394
7	2293	16926	19047	19321	20000	16263	3118
8	5541	27827	30027	31537	32276	30093	7759
9	10472	14713	15113	15520	15718	16794	12580
10	17466	9164	8268	8603	8341	10036	17789
11	21202	11137	9319	9681	10155	11957	22406
12	25167	13948	11767	12364	12930	15127	25665
13	25400	13870	11820	11605	12695	15570	26591
14	24447	12972	10561	10850	11353	14532	26295
15	24058	13660	11770	11673	12575	15901	26323
16	24125	19178	18467	17448	18741	21216	25091
17	21535	33890	35371	33102	34876	32129	22854
18	18049	32472	33983	31811	32943	27366	19848
19	14999	23396	23148	22635	23663	19428	16498
20	11391	16229	16244	16352	17701	13875	12412
21	8708	11632	11909	12393	12981	10939	10495
22	6586	7802	8491	9126	9676	9768	9462
23	4274	4340	4944	5209	6476	7651	7922

In [ ]:
```
cols = ["casual", "registered"]
fig = plt.figure(figsize=(8,8))
```

```
for i in range(len(cols)):
    col = cols[i]
    data = df#.loc[df["day_status"] == "Holiday"]
    data = data.groupby(['weekday', 'time_of_day'], observed=False)[col].sum().reset_index()
    data = data.pivot(index='weekday', columns='time_of_day', values=col)
    data.columns.name = None
    data = data[[ "Midnight Hours", "Dawn", "Early Morning", "Morning", "Afternoon", "Late Afternoon", "Evening", "Night"]]
    data.index = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']

    fig.add_subplot(2,1,i+1)
    sns.heatmap(data.T, cmap=cmap_seagreen, annot=True, fmt='g', cbar_kws={'label': col})

plt.show()
```

Hypothesis Testing

Test 1

- Null Hypothesis (H_o): The average number of rides on working days is >= number of rides on non working days.
- Alternative Hypothesis (H_a): The average number of rides on working days is < number of rides on non working days.
- Significance Level α = 0.05

In [ ]:
```
notworkday = df[df['day_status'] != "Work Day"]['count']
# Unequal sample sizes can lead to unequal variances
# Because notworkday are less than workday, only notworkday.count() samples are being used
workday = df[df['day_status'] == "Work Day"]['count'].sample(notworkday.count())

test_stat, pvalue = stats.levene(workday, notworkday)
print(f"pvalue is {round(pvalue,2)}, implying","similar variances" if pvalue > 0.05 else "non-similar variances")
```

```
pvalue is 0.79, implying similar variances
```

In [ ]:
```
test_stat, pvalue = stats.ttest_ind(workday, notworkday, alternative="less")
print(
    f"pvalue is {round(pvalue,2)}, implying we",
    "failed to reject null hypothesis and \nthe number of rides on working days is >= number of rides on non working days"
    if pvalue > 0.05 else
    "reject null hypothesis and \nthe number of rides on working days is < number of rides on non working days"
)
```

```
pvalue is 0.88, implying we failed to reject null hypothesis and
the number of rides on working days is >= number of rides on non working days
```

Test 2

- Null Hypothesis (H_o): The average number of rides is equal across all seasons. ($\mu1 = \mu2 = \mu3 = \mu4$, where $\mu1$, $\mu2$, $\mu3$, $\mu4$ are the average ride count for four seasons.)
- Alternative Hypothesis (H_a): The average number of rides in at least one season is different. (At least one mean is not equal to the others.)
- Significance Level α = 0.05

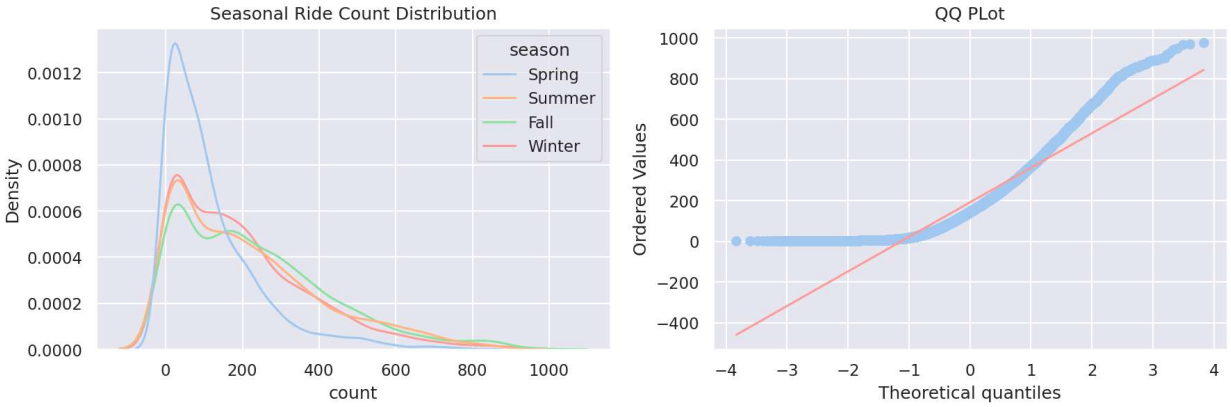
In [ ]:
```
# For equal sample count
n = df["season"].value_counts().min()
s1 = df.loc[df["season"] == "Spring", "count"].sample(n)
s2 = df.loc[df["season"] == "Summer", "count"].sample(n)
s3 = df.loc[df["season"] == "Fall", "count"].sample(n)
s4 = df.loc[df["season"] == "Winter", "count"].sample(n)
```

In [ ]:
```
# Test for Normality
# Null Hypothesis ($H_o$): The data comes from a normal distribution.
# Alternative Hypothesis ($H_a$): The data does not come from a normal distribution.

print("s1 p-value", stats.shapiro(s1)[1])
print("s2 p-value", stats.shapiro(s2)[1])
print("s3 p-value", stats.shapiro(s3)[1])
print("s4 p-value", stats.shapiro(s4)[1])
```

```
s1 p-value 8.749584618867163e-49
s2 p-value 1.2834643863634394e-38
s3 p-value 2.1538609300321335e-36
s4 p-value 2.272417114534660 8e-39
```

All p-values are < 0.05, therefore no season is normally distributed

In [ ]:
```
# Visual analysis

fig = plt.figure(figsize=(14,4))

ax1 = fig.add_subplot(1,2,1)
sns.kdeplot(df, x="count", hue="season", ax=ax1)
ax1.set_title("Seasonal Ride Count Distribution")

ax2 = fig.add_subplot(1,2,2)
stats.probplot(df["count"], dist='norm', plot=ax2)
ax2.set_title("QQ PLot")

plt.show()

print("Skewness : ", df['count'].skew())
print("Kurtosis : ", df['count'].kurt())
```




```
Skewness :  1.2420662117180776
Kurtosis :  1.3000929518398334
```

In [ ]:
```
# Test for equal variance
# Null Hypothesis ($H_o$): The variances of the groups are equal.
# Alternative Hypothesis ($H_a$): At least one group has a variance that is significantly different from the others.

print("p-value:", stats.levene(s1, s2, s3, s4)[1])
```

```
p-value: 1.0335905377801708e-118
```

The p-value is < 0.05 therefore we reject the null hypothesis; At least one season has different variance

The ANOVA asumptions are not met, therefore ideal test will be Kruskal-Wallis test

In [ ]:
```
stats.f_oneway(s1, s2, s3, s4)
```

Out[ ]:
```
F_onewayResult(statistic=237.97591286119527, pvalue=1.6714120471516456e-149)
```

Observation: As p-value is < 0.05 we reject the null hypothesis; The average number of rides in at least one season is different.

Test 3

- Null Hypothesis (H_o): The average number of rides is equal across all weathers. ($\mu1 = \mu2 = \mu3 = \mu4$, where $\mu1, \mu2, \mu3, \mu4$ are the average ride count for four weathers.)
- Alternative Hypothesis (H_a): The average number of rides in at least one weather is different. (At least one mean is not equal to the others.)
- Significance Level $\alpha = 0.05$

In [ ]:
```
# For equal sample count
n = df["weather"].value_counts().min()

w1, w2, w3 = df["weather"].value_counts().index

w1 = df.loc[df["weather"] == w1, "count"].sample(n)
w2 = df.loc[df["weather"] == w2, "count"].sample(n)
w3 = df.loc[df["weather"] == w3, "count"].sample(n)
```

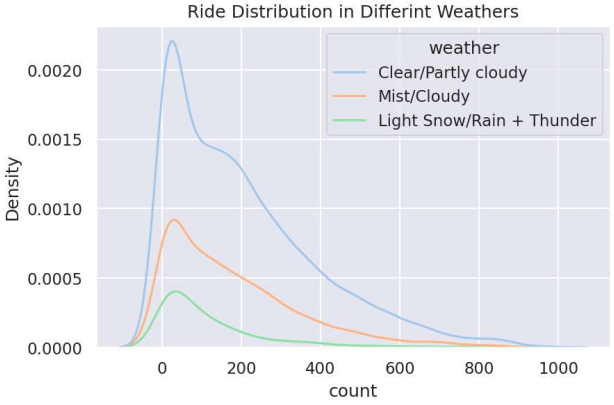
In [ ]:
```
# Test for Normality
# Null Hypothesis ($H_o$): The data comes from a normal distribution.
# Alternative Hypothesis ($H_a$): The data does not come from a normal distribution.

print("w1 p-value", stats.shapiro(s1)[1])
print("w2 p-value", stats.shapiro(s2)[1])
print("w3 p-value", stats.shapiro(s3)[1])
```

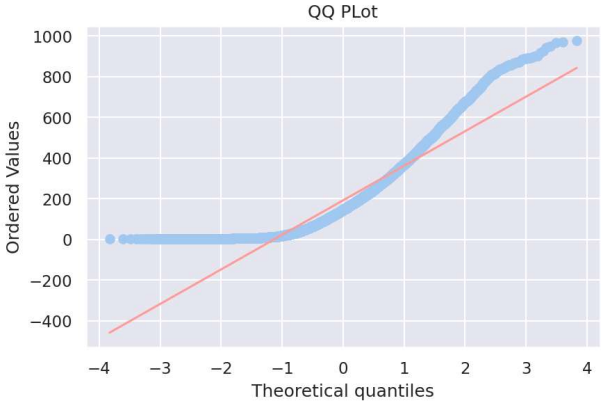
```
w1 p-value 8.749584618867163e-49
w2 p-value 1.2834643863634394e-38
w3 p-value 2.1538609300321335e-36
```

All p-values are < 0.05, therefore no season is normally distributed

In [ ]:
```
# Visual analysis

fig = plt.figure(figsize=(14,4))

ax1 = fig.add_subplot(1,2,1)
sns.kdeplot(df, x="count", hue="weather", ax=ax1)
ax1.set_title("Ride Distribution in Differint Weathers")

ax2 = fig.add_subplot(1,2,2)
stats.probplot(df["count"], dist='norm', plot=ax2)
ax2.set_title("QQ PLot")
```

```
plt.show()

print("Skewness : ", df['count'].skew())
print("Kurtosis : ", df['count'].kurt())
```






```
Skewness :  1.2420662117180776
Kurtosis :  1.3000929518398334
```

In [ ]:
```
# Test for equal variance
# Null Hypothesis ($H_o$): The variances of the groups are equal.
# Alternative Hypothesis ($H_a$): At least one group has a variance that is significantly different from the others.

print("p-value:", stats.levene(w1, w2, w3)[1])
```

```
p-value: 1.9888027196987587e-20
```

The p-value is < 0.05 therefore we reject the null hypothesis; At least one season has different variance

The ANOVA asumptions are not met, therefore ideal test will be Kruskal-Wallis test

In [ ]:
```
stats.f_oneway(w1, w2, w3)
```

Out[ ]:
```
F_onewayResult(statistic=64.46181844395787, pvalue=4.8123780735145934e-28)
```

Observation: As p-value is < 0.05 we reject the null hypothesis; The average number of rides in at least one weather is different.

Test 4

- Null Hypothesis (H_o): The weather and seasons are independent (no association).
- Alternative Hypothesis (H_a): The weather and seasons are not independent (there is an association).
- Significance Level α = 0.05

In [ ]:
```
data = pd.crosstab(df['season'], df['weather'], values=df['count'], aggfunc='sum')
data
```

Out[ ]:

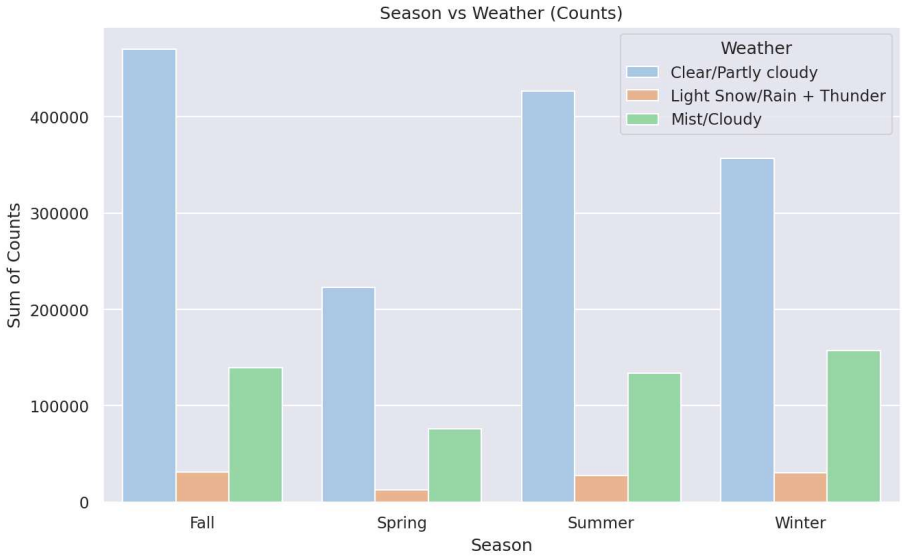
weather	Clear/Partly cloudy	Light Snow/Rain + Thunder	Mist/Cloudy
season			
Fall	470116	31160	139386
Spring	223009	13083	76406
Summer	426350	27755	134177
Winter	356588	30255	157191

In [ ]:
```
stats.chi2_contingency(data)
```

Out[ ]:
```
Chi2ContingencyResult(statistic=10788.223633468619, pvalue=0.0, dof=6, expected_freq=array([[453449.22392106,  31412.3065842 , 155800.46949473],
       [221180.55320416,  15322.09337053,  75995.35342531],
       [416375.58704392,  28844.06214505, 143062.35081104],
       [385057.63583086,  26674.53790022, 132301.82626892]]))
```

Observation: Since the p-value is less than the 0.05 significance level, we reject the null hypothesis. Hence, weather conditions are dependent on the ongoing season.

In [ ]:
```
crosstab_melt = data.reset_index().melt(id_vars='season', var_name='weather', value_name='count')

plt.figure(figsize=(10, 6))
sns.barplot(data=crosstab_melt, x='season', y='count', hue='weather')
plt.title("Season vs Weather (Counts)")
plt.xlabel("Season")
plt.ylabel("Sum of Counts")
plt.legend(title="Weather")
plt.show()
```



Insights

- There was an inverse correlation between temperature and humidity.
- Around 70% of the rides are booked on working days.
- The number of registered riders is significantly higher than casual riders.
- The number of rides booked by registered users peaks around 8 am and 5 pm.
- The number of rides booked is highest in fall and summer. Additionally, casual riders highly prefer these months to ride.
- Most casual riders ride on Saturdays and Sundays; whereas most registered riders ride from Monday to Friday.

Hypothesis Tests

Test 1 (T-test independence)

- Null Hypothesis (Ho): The average number of rides on working days is >= number of rides on non-working days.
- Alternative Hypothesis (Ha): The average number of rides on working days is < the number of rides on non-working days.
- **Observation:** p-value is 0.88, implying we failed to reject the null hypothesis and the number of rides on working days is >= number of rides on non-working days

Test 2 (ANOVA)

- Null Hypothesis (Ho): The average number of rides is equal across all seasons. ($\mu1 = \mu2 = \mu3 = \mu4$, where $\mu1$, $\mu2$, $\mu3$, $\mu4$ are the average ride count for four seasons.)
- Alternative Hypothesis (Ha): The average number of rides in at least one season is different. (At least one mean is not equal to the others.) -**Observation:** As the p-value is < 0.05 we reject the null hypothesis; The average number of rides in at least one season is different.

Test 3 (ANOVA)

- Null Hypothesis (Ho): The average number of rides is equal across all weathers. ($\mu1 = \mu2 = \mu3 = \mu4$, where $\mu1$, $\mu2$, $\mu3$, $\mu4$ are the average ride count for four weathers.)
- Alternative Hypothesis (Ha): The average number of rides in at least one weather is different. (At least one mean is not equal to the others.) -**Observation:** As p-value is < 0.05 we reject the null hypothesis; The average number of rides in at least one weather is different.

Test 4 (Chi-Square Test of Independence)

- Null Hypothesis (Ho): The weather and seasons are independent (no association).
- Alternative Hypothesis (Ha): The weather and seasons are not independent (there is an association). -**Observation:** We reject the null hypothesis since the p-value is less than the 0.05 significance level. Hence, weather conditions are dependent on the ongoing season.

Code for Saving Analysis File

In [ ]:
```
import subprocess
import os
from bs4 import BeautifulSoup

drive_source_file = "Scaler/Yulu Case Study/Yulu Case Study.ipynb"
style = """
    img{
        max-height: 30rem !important;
        max-width: 80vw !important;
        object-fit: contain;
    }
"""

def save_html_for_print(drive_source_file, style):
  command = f"jupyter nbconvert --to html '/content/drive/MyDrive/{drive_source_file}' --output '/content/temp.html'"
  result = subprocess.run(command, shell=True, text=True, capture_output=True)
  #print("STDOUT:", result.stdout, "\nSTDERR:", result.stderr, "\nReturn Code:", result.returncode)
  with open("/content/temp.html", "r") as file:
      html_content = file.read()
  soup = BeautifulSoup(html_content, "html.parser")
  style_tag = soup.new_tag("style")
  style_tag.string = style
  soup.head.append(style_tag)
  os.remove("/content/temp.html")
  new_file_path = f"/content/{drive_source_file.split('/')[-1].split('.')[0]}.html"
  with open(new_file_path, "w") as new_file:
      new_file.write(str(soup))

# self reminder: increase the DPI of sns plots before saving
save_html_for_print(drive_source_file, style)
```