


```
import pandas as pd
uber = pd.read_csv("uber-raw-data-janjune-15.csv")



# Assuming 'Pickup_date' column contains datetime values in the 'uber' Dat
uber['Pickup_date'] = pd.to_datetime(uber['Pickup_date'])

uber['Pickup_date'] = uber['Pickup_date'].apply(lambda x: x.replace(year=2

uber.head()
```





	Dispatching_base_num	Pickup_date	Affiliated_base_num	locationID
0	B02617	2022-05-17 09:47:00	B02617	141.0
1	B02617	2022-05-17 09:47:00	B02617	65.0
2	B02617	2022-05-17 09:47:00	B02617	100.0
3	B02617	2022-05-17 09:47:00	B02774	80.0
4	B02617	2022-05-17 09:47:00	B02617	90.0



```
uber.tail()
```

	Dispatching_base_num	Pickup_date	Affiliated_base_num	locationID
5641564	B02617	2022-02-27 14:30:27	B02617	11.0
5641565	B02617	2022-02-27 22:52:43	B02617	205.0
5641566	B02617	2022-02-27 09:07:39	B02617	107.0
5641567	B02617	2022-02-27 17:31:21	B02617	186.0
5641568	B02617	2022-02-27 06:40:09	NaN	NaN



```
uber = pd.read_csv('uber-raw-data-janjune-15.csv')
```

```
# Get the number of rows and columns
num_rows, num_columns = uber.shape
```

```
# Print the results
print(f'Number of Rows: {num_rows}')
print(f'Number of Columns: {num_columns}')
```

```
Number of Rows: 5641569
Number of Columns: 4
```

```
if uber.duplicated().any():
    print(f"There are as many as {uber.duplicated().sum()} duplicate data.")
else:
    print("There are no duplicate data.")
```

```
There are as many as 393939 duplicate data.
```

```
uber.drop_duplicates(inplace=True)
```

```
uber.head()
```

	Dispatching_base_num	Pickup_date	Affiliated_base_num	locationID	month	weekda
0	B02617	2022-05-17 09:47:00	B02617	141.0	May	Sunda
1	B02617	2022-05-17 09:47:00	B02617	65.0	May	Sunda
2	B02617	2022-05-17 09:47:00	B02617	100.0	May	Sunda
3	B02617	2022-05-17 09:47:00	B02774	80.0	May	Sunda
4	B02617	2022-05-17 09:47:00	B02617	90.0	May	Sunda

```
# Get the number of rows and columns
num_rows, num_columns = uber.shape
```

```
# Print the results
print(f'Number of Rows: {num_rows}')
print(f'Number of Columns: {num_columns}')
```

```
Number of Rows: 5247630
Number of Columns: 10
```

```
import pandas as pd
# Assuming 'Pickup_date' column in 'uber' DataFrame contains date strings
uber['Pickup_date'] = pd.to_datetime(uber['Pickup_date'], errors='coerce')
```

```
# Check for rows where conversion failed
invalid_dates = uber[uber['Pickup_date'].isnull()]
print(invalid_dates)
```

```
Empty DataFrame
Columns: [Dispatching_base_num, Pickup_date, Affiliated_base_num, locationID]
Index: []
```

```
uber["month"] = uber["Pickup_date"].dt.month_name()
a=uber["month"].value_counts()
print(a)
```

```
June      1129920
February   896499
April      881927
May        808493
January    770535
March      760256
Name: month, dtype: int64
```

```
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import matplotlib.gridspec as gridspec
```

```
# Sample data for months (replace this with your data)
months = ['January', 'February', 'March', 'April', 'May', 'June']
data = [23, 13.89, 18.25, 16.58, 20.89, 7.4]
```

```
# Calculate percentages
total = sum(data)
data_per = [d / total * 100 for d in data]
```

```
# Define colors for each month
cmap = plt.cm.get_cmap('viridis', len(months)) # Using 'viridis' colormap for this ex
```

```

norm = mcolors.Normalize(vmin=0, vmax=len(months) - 1)
colors = [cmap(norm(i)) for i in range(len(months))]

# Create figure and gridspec layout
fig = plt.figure(figsize=(10, 6))
gs = gridspec.GridSpec(1, 2, width_ratios=[4, 1]) # Divide the figure into two parts

# Create the pie chart
ax1 = plt.subplot(gs[0])
ax1.pie(data_per, labels=[f'{m}: {d:.2f}%' for m, d in zip(months, data_per)], colors=
circle = plt.Circle((0, 0), 0.7, color='white')
ax1.add_artist(circle)
ax1.set_aspect('equal') # Ensure pie chart is circular

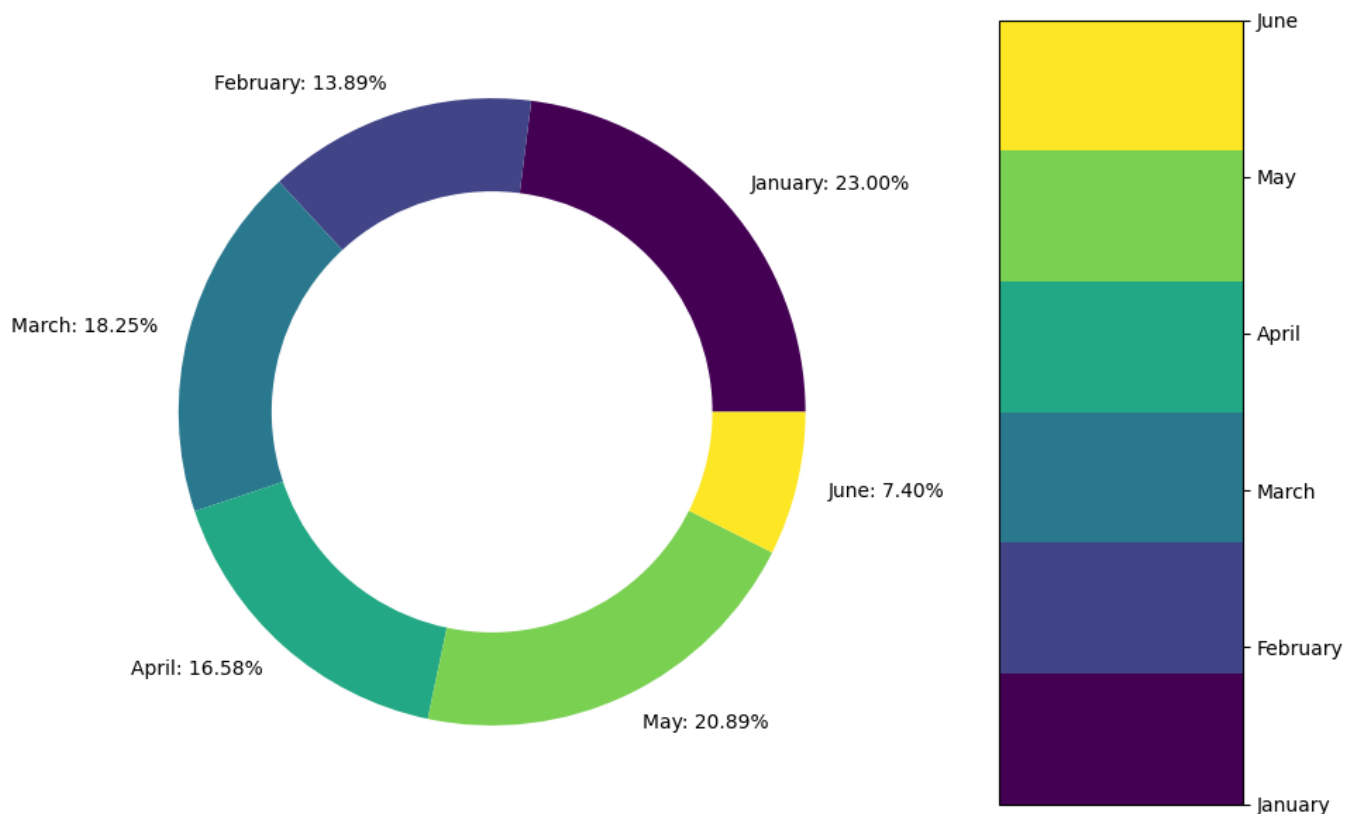
# Create a color bar with month labels
ax2 = plt.subplot(gs[1])
sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
cbar = plt.colorbar(sm, cax=ax2, ticks=range(len(months)))
cbar.set_ticklabels(months)

plt.tight_layout()
plt.show()

```




<ipython-input-118-72f8c5a7be70>:14: MatplotlibDeprecationWarning:

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two min



```
june = pd.read_csv("uber-raw-data-jun14.csv", skiprows=[590016])
```

```
uber_rush = june.groupby(['Lat', 'Lon'], as_index=False).size()  
uber_rush
```

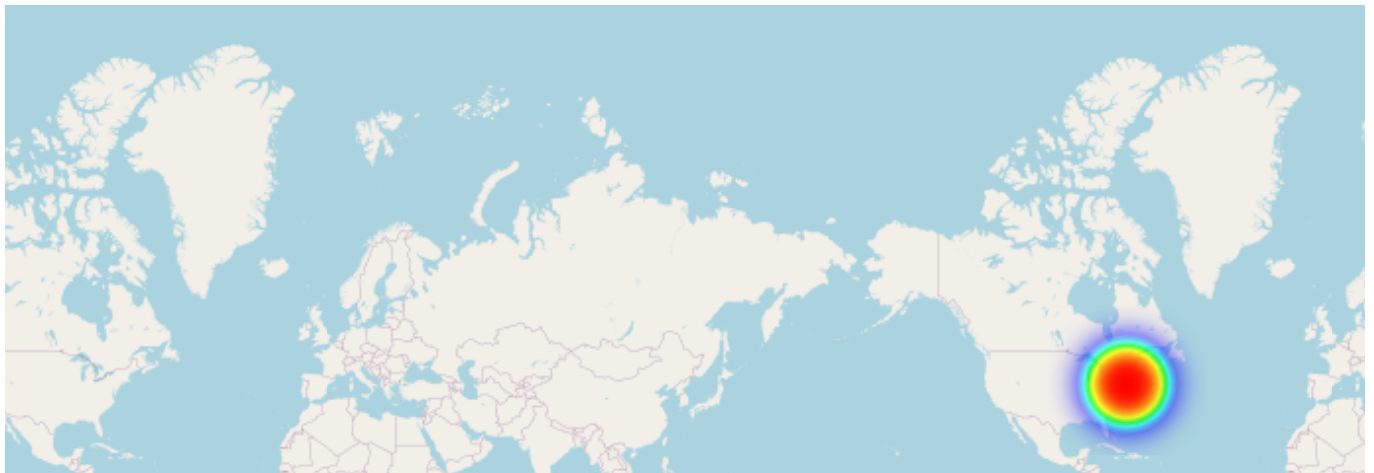
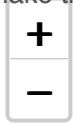
	Lat	Lon	size	
0	39.9558	-74.2063	1	
1	40.0682	-74.0533	1	
2	40.0830	-74.0708	1	
3	40.0937	-74.0374	1	
4	40.0937	-74.0373	1	
...	
212808	41.2242	-73.6490	1	
212809	41.2259	-73.6760	1	
212810	41.3074	-73.6682	1	
212811	41.3195	-73.6903	1	
212812	41.3199	-73.6902	1	

212813 rows × 3 columns

```
import folium
from folium.plugins import HeatMap

basemap = folium.Map()
HeatMap(uber_rush).add_to(basemap)
basemap
```

Make this Notebook Trusted to load map: File -> Trust Notebook



```
# Assuming you have the "weekday" column in the uber DataFrame
uber["weekday"] = uber["Pickup_date"].dt.day_name()

# Create a pivot table
pivot = pd.crosstab(index=uber["month"], columns=uber["weekday"])

# Define the desired order for months
month_order = ['January', 'February', 'March', 'April', 'May', 'June']

# Re-order the rows (months) based on the desired order
pivot = pivot.reindex(month_order)

# Re-order the columns
weekday_order = ['Friday', 'Monday', 'Saturday', 'Sunday', 'Thursday', 'Tuesday', 'Wednesday']
pivot = pivot[weekday_order]

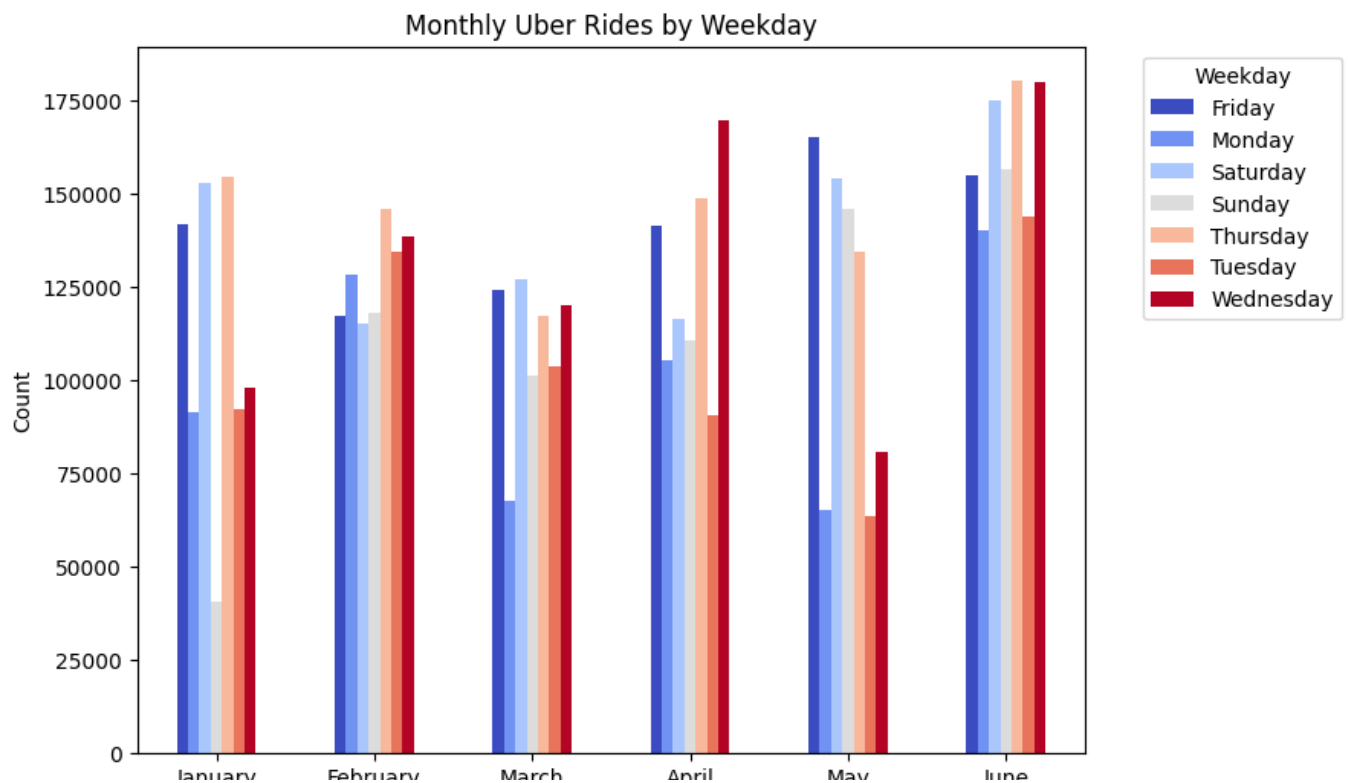
# Display the re-arranged DataFrame
print(pivot)
```

weekday month	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
January	141857	91070	152895	40432	154347	91984	97950
February	117128	128093	115225	117737	145784	134300	138232
March	124000	67516	126934	100927	117226	103626	120027
April	141076	105268	116132	110561	148701	90645	169544
May	165171	65002	154139	145892	134150	63412	80727
June	154604	140224	174703	156505	180291	143740	179853

```
plt.figure(figsize=(8, 6)) # Set the figure size
pivot.plot(kind="bar", figsize=(8, 6), cmap='coolwarm') # Use a coolwarm color palett
plt.xlabel("Month") # X-axis label
plt.ylabel("Count") # Y-axis label
plt.title("Monthly Uber Rides by Weekday") # Plot title
plt.xticks(rotation=0) # Set X-axis label rotation




# Show the plot
plt.legend(title="Weekday", bbox_to_anchor=(1.05, 1), loc='upper left') # Add legend
plt.show()
```

<Figure size 800x600 with 0 Axes>



#january has the highest rides


```
uber['hour'] = uber['Pickup_date'].dt.hour
summary = uber.groupby(['weekday', 'hour'], as_index=False).size()
summary
```

	weekday	hour	size	
0	Friday	0	36256	
1	Friday	1	19922	
2	Friday	2	12196	
3	Friday	3	8610	
4	Friday	4	10340	
...	
163	Wednesday	19	59472	
164	Wednesday	20	56996	
165	Wednesday	21	54515	
166	Wednesday	22	51141	
167	Wednesday	23	40552	

168 rows × 3 columns

```
import seaborn as sns

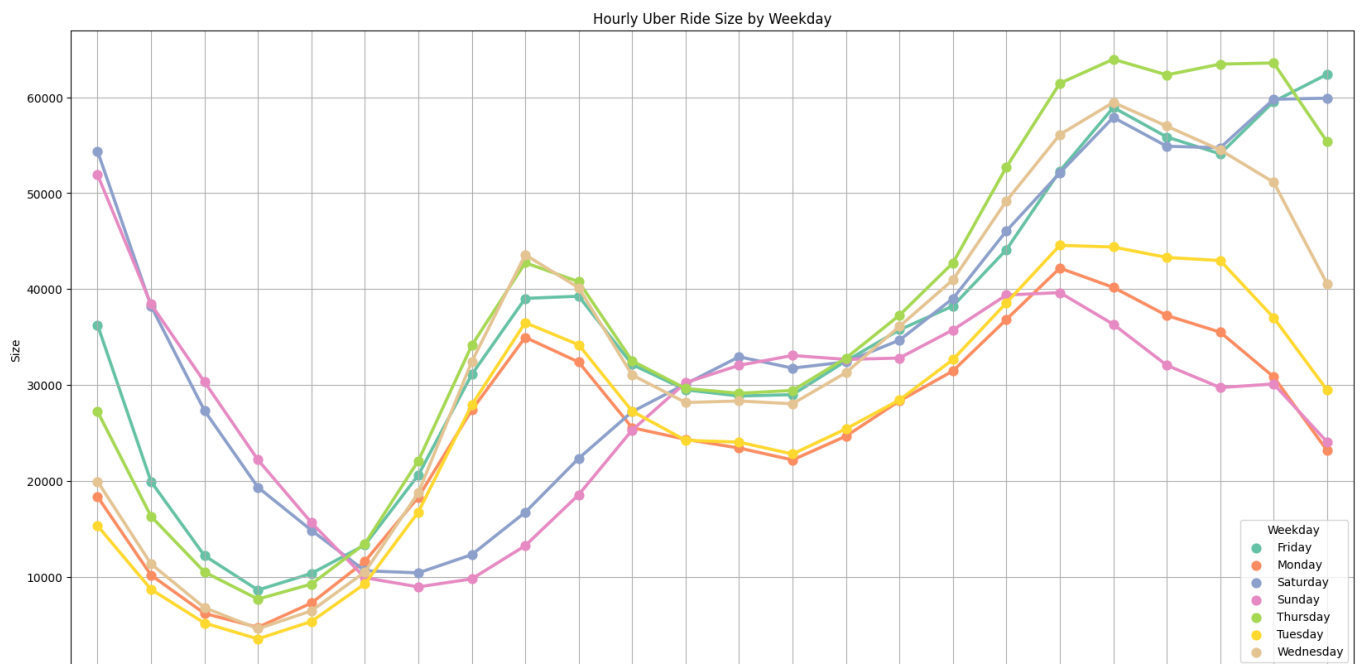
# Set the figure size
plt.figure(figsize=(20,10))

# Create the point plot
sns.pointplot(x="hour", y="size", hue="weekday", data=summary, palette="Set2")

# Add labels and title
plt.xlabel("Hour of the Day")
plt.ylabel("Size")
plt.title("Hourly Uber Ride Size by Weekday")

# Customize the legend
plt.legend(title="Weekday")

# Show the plot
plt.grid(True) # Add grid lines
plt.show()
```



#6 to 10 am on wednesday has highest rides of the day

```
uber_foils = pd.read_csv("Uber-Jan-Feb-FOIL.csv")
```

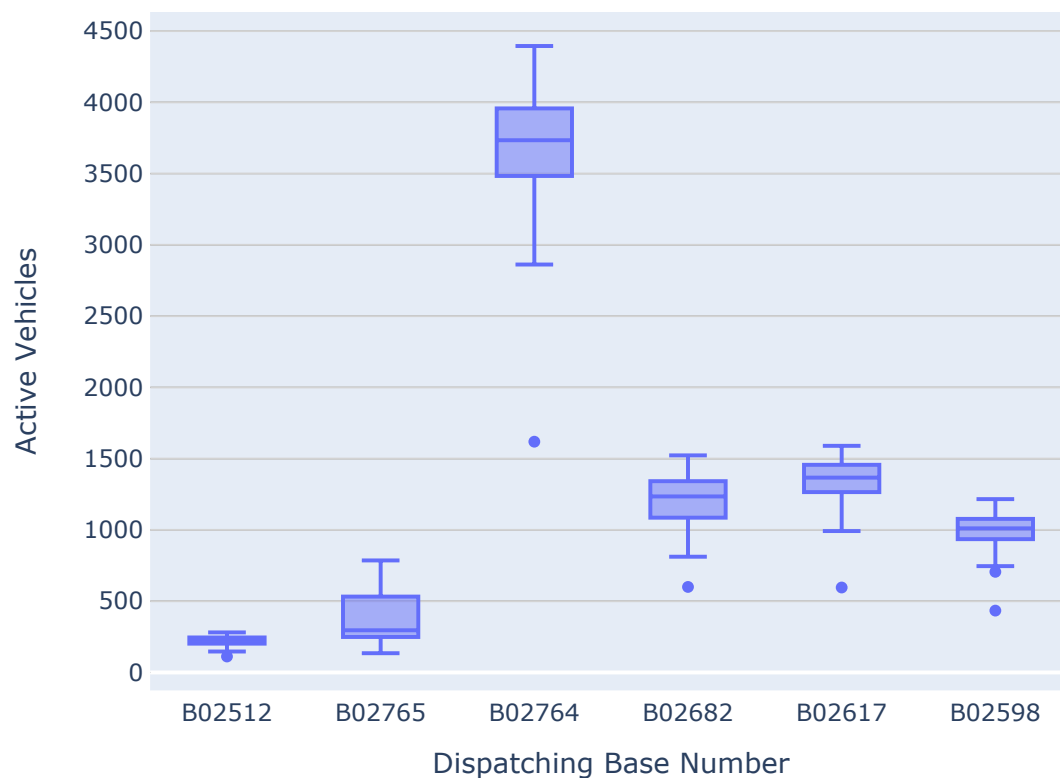
```
from plotly.offline import init_notebook_mode
import plotly.express as px

# Assuming 'dispatching_base_number' and 'active_vehicles' are columns in uber_foil Da
fig = px.box(
    x='dispatching_base_number',
    y='active_vehicles',
    data_frame=uber_foil,
    title="Box Plot of Active Vehicles by Dispatching Base Number",
    labels={'dispatching_base_number': 'Dispatching Base Number', 'active_vehicles': 'Active Vehicles'
})

# Customize the layout
fig.update_layout(
    xaxis=dict(showline=True, showgrid=False),
    yaxis=dict(showline=True, showgrid=True, gridcolor='lightgray'),
)

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

Box Plot of Active Vehicles by Dispatching Base Number



```

first = pd.read_csv("/content/other-Firstclass_B01536.csv", encoding='latin1')
# Try specifying a different delimiter (e.g., '\t' for tab-separated files)
# Read the file without specifying a delimiter to let Pandas infer the delimiter
# Read the file by specifying a different delimiter or handling errors
second = pd.read_csv("/content/other-Highclass_B01717.csv", encoding='latin1')


import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'first' and 'second' DataFrames have 'TIME' columns

# Function to parse datetime and handle exceptions
def parse_datetime(value):
    try:
        return pd.to_datetime(value)
    except (ValueError, TypeError, pd.errors.ParserError):
        return pd.NaT # Return NaT for values that cannot be parsed

# Convert 'TIME' column to datetime format in the 'first' DataFrame
first['TIME'] = first['TIME'].apply(parse_datetime)

# Convert 'TIME' column to datetime format in the 'second' DataFrame
second['TIME'] = second['TIME'].apply(parse_datetime)

# Drop rows with NaT values (rows with values that couldn't be parsed)
first = first.dropna(subset=['TIME'])
second = second.dropna(subset=['TIME'])

# Extract hour from the 'TIME' column to calculate counts per hour
first['Hour'] = first['TIME'].dt.hour
second['Hour'] = second['TIME'].dt.hour

# Calculate hourly counts
hourly_counts = first['Hour'].value_counts().sort_index()
hourly_counts_second = second['Hour'].value_counts().sort_index()

# Plotting code remains the same
plt.figure(figsize=(12, 6)) # Adjust figure size

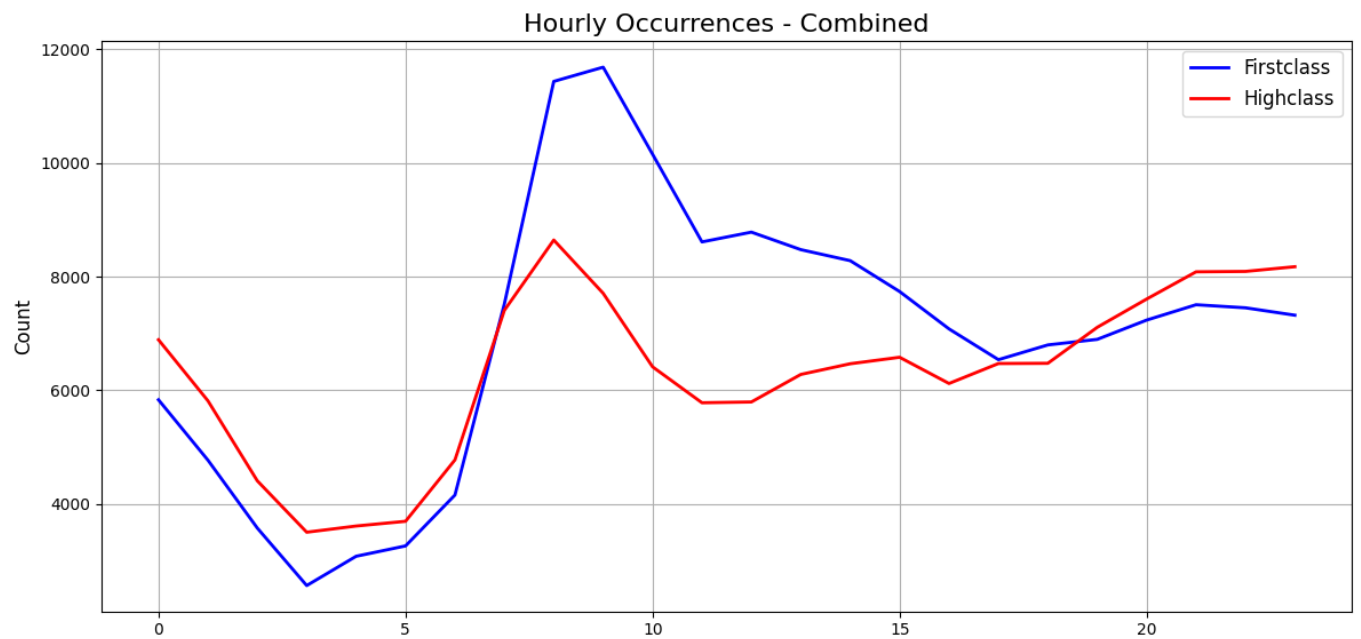
# Plotting 'first' DataFrame
hourly_counts.plot(label='Firstclass', linewidth=2, linestyle='-', color='blue')

# Plotting 'second' DataFrame
hourly_counts_second.plot(label='Highclass', linewidth=2, linestyle='-', color='red')

# Additional beautification and customization
plt.title('Hourly Occurrences - Combined', fontsize=16) # Title
plt.xlabel('Hour of the Day', fontsize=12) # X-axis label
plt.ylabel('Count', fontsize=12) # Y-axis label

```

```
plt.legend(fontsize=12) # Legend with font size
plt.grid(True) # Show grid lines
plt.tight_layout() # Adjust layout
plt.show()
```



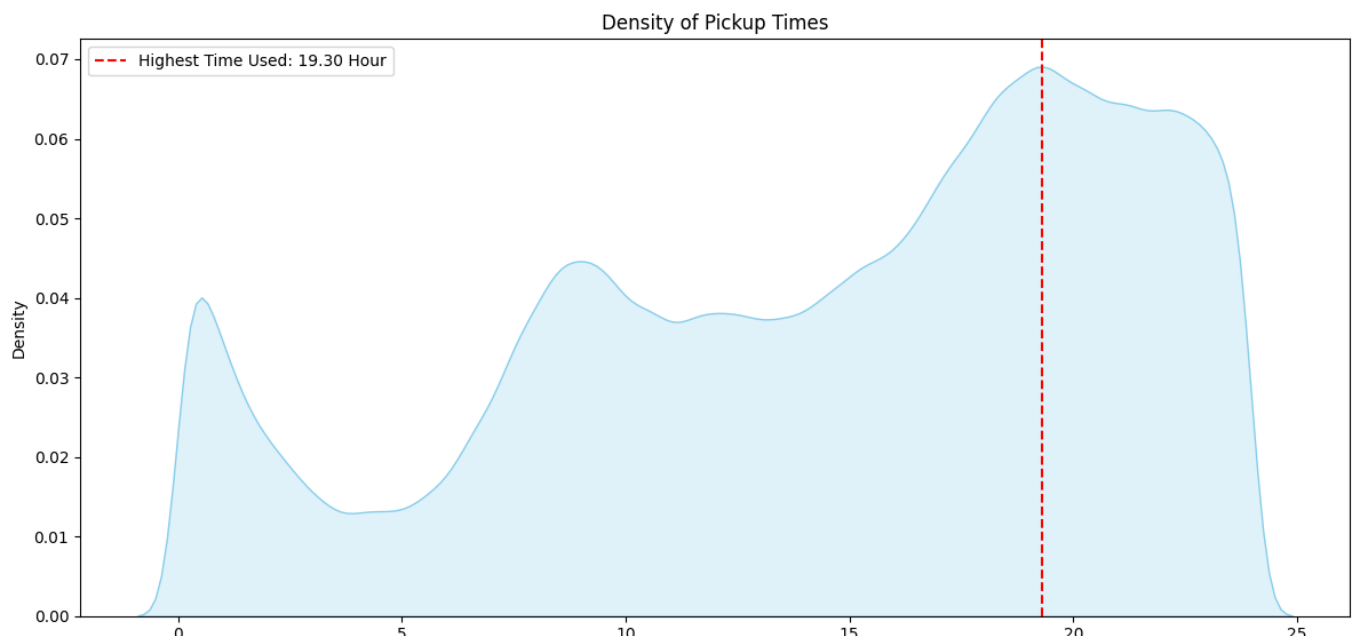
```
import seaborn as sns

# Convert pickup times to hours and minutes
uber['Pickup_time_hours'] = uber['Pickup_date'].dt.hour + uber['Pickup_date'].dt.minute

# Plot a KDE plot to visualize the density of pickup times in hours and minutes
plt.figure(figsize=(12, 6))
sns.kdeplot(data=uber['Pickup_time_hours'], fill=True, color='skyblue')
plt.xlabel('Pickup Time (Hour)')
plt.ylabel('Density')
plt.title('Density of Pickup Times')

# Find the hour with the highest occurrence
highest_time = uber['Pickup_time_hours'].mode().values[0]

# Annotate the highest time used
plt.axvline(x=highest_time, color='red', linestyle='--', label=f'Highest Time Used: {h}')
plt.legend()
plt.tight_layout()
plt.show()
```



```
# Convert 'Pickup_date' column to datetime format
uber['Pickup_date'] = pd.to_datetime(uber['Pickup_date'])

# Extract hour from the 'Pickup_date' column
uber['Hour'] = uber['Pickup_date'].dt.hour

# Find the hour with the highest occurrence
hour_with_highest_count = uber['Hour'].mode().values[0]

# Filter data for the hour with the highest occurrence
data_for_highest_hour = uber[uber['Hour'] == hour_with_highest_count]

# Find the location ID with the highest occurrence in this hour
location_with_highest_count = data_for_highest_hour['locationID'].mode().values[0]

# Print the hour with the highest occurrence and the location ID with the highest count
print(f"The hour with the highest occurrence: {hour_with_highest_count}")
print(f"The location ID with the highest occurrence at this hour: {location_with_highest_count}")
```

```
The hour with the highest occurrence: 19
The location ID with the highest occurrence at this hour: 161.0
```

```
#click this like for tableau
https://prod-uk-a.online.tableau.com/t/ieeeproject/views/ProductUtilization/ProductUtilization
```

```
#testing starts
```

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt

# Assuming 'Dispatching_base_num', 'locationID', and other relevant columns as feature
features = uber[['Dispatching_base_num', 'locationID', 'Pickup_date']]

# Convert 'Dispatching_base_num' to a categorical variable
features['Dispatching_base_num'] = features['Dispatching_base_num'].astype('category')

# Convert categorical variables to dummy/indicator variables
features = pd.get_dummies(features, drop_first=True)

# Update the target variable based on the actual column name in your dataset
target_column_name = 'Pickup_date'
target = uber[target_column_name]

# Split the data into training and testing sets (80% training, 20% testing)
```

```
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, r

# Identify datetime columns and drop them before imputation
datetime_columns = X_train.select_dtypes(include=['datetime64']).columns
X_train = X_train.drop(columns=datetime_columns)
X_test = X_test.drop(columns=datetime_columns)

# Impute missing values with the mean (you can change this strategy)
imputer = SimpleImputer(strategy='mean')
X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
X_test_imputed = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)

# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train_imputed, y_train)

# Make predictions on the test set
predictions = model.predict(X_test_imputed)

# Evaluate the model
# Convert 'y_true' to float64
# Convert 'y_true' to numeric format
y_true_numeric = y_test.astype(int) / 10**9 # Convert nanoseconds to seconds

# Evaluate the model
mse = mean_squared_error(y_true_numeric, predictions)
r2 = r2_score(y_true_numeric, predictions)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Visualize actual vs predicted values
# Reduce the number of data points to plot
num_points = min(len(y_test), 100)
plt.plot(y_test.index[:num_points], y_true_numeric[:num_points], label='Actual (Blue)')
plt.plot(y_test.index[:num_points], predictions[:num_points], label='Predicted (Red)',
plt.xlabel('Index')
plt.ylabel('Target Values')
plt.title('Actual vs Predicted Target Values')
plt.legend()
plt.show()
```



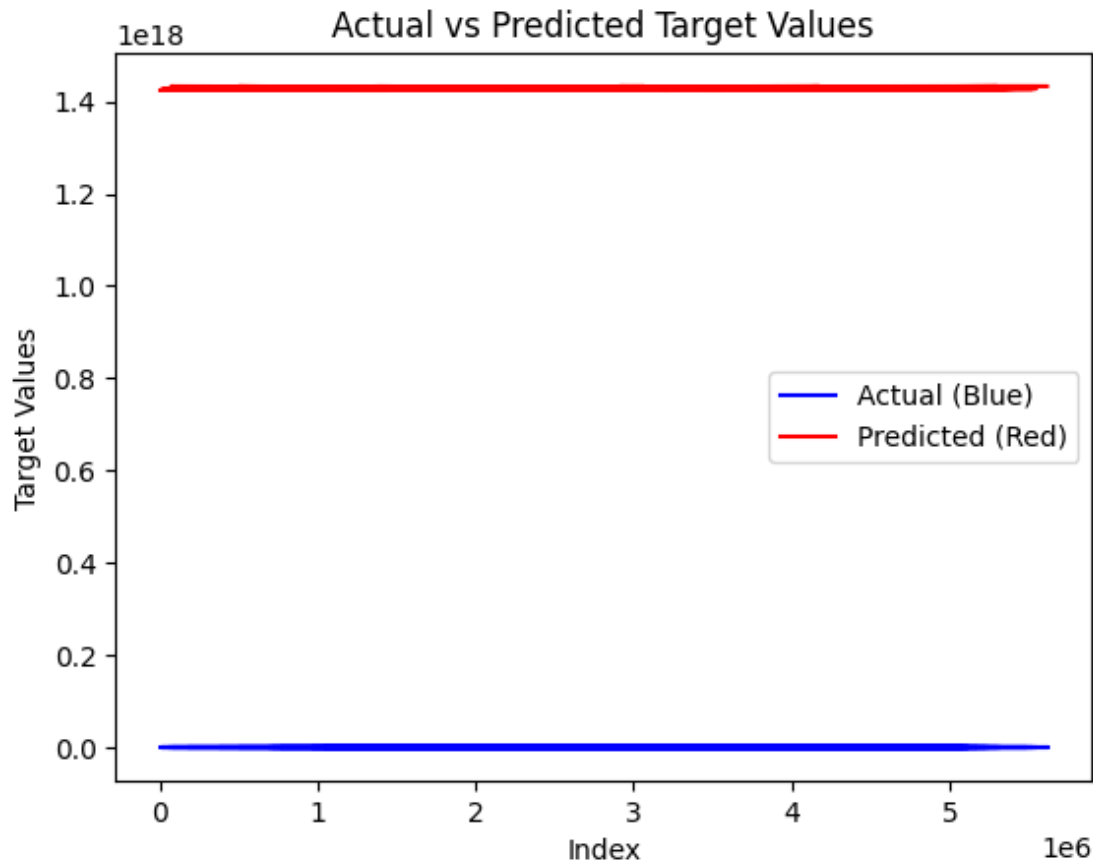
```
<ipython-input-138-e13079f59bb2>:13: SettingWithCopyWarning:
```

A value is trying to be set on a copy of a slice from a DataFrame.
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable>

Mean Squared Error: 2.04043727635832e+36

R-squared: -9.32210067459537e+22



```
from matplotlib import pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Assuming 'Pickup_date' column in 'uber' DataFrame contains date strings
# Replace 'your_dataset.csv' with the actual file path or dataset name

# Assuming 'Pickup_date' column contains datetime values in the 'uber' DataFrame
uber['Pickup_date'] = pd.to_datetime(uber['Pickup_date'])

# Change the year from 2015 to 2022
uber['Pickup_date'] = uber['Pickup_date'].apply(lambda x: x.replace(year=2022))

uber['Month'] = uber['Pickup_date'].dt.to_period('M').astype('str') # Ensure 'Month'
```

```
# Extracting additional information for each highest date
uber_data = uber.groupby('Month').agg({'Pickup_date': 'max', 'locationID': 'count'}).r
uber_data = uber_data.rename(columns={'locationID': 'Num_Ubers'})

# Convert 'Month' to numeric for model training
uber_data['Month_Num'] = pd.to_datetime(uber_data['Month']).dt.month

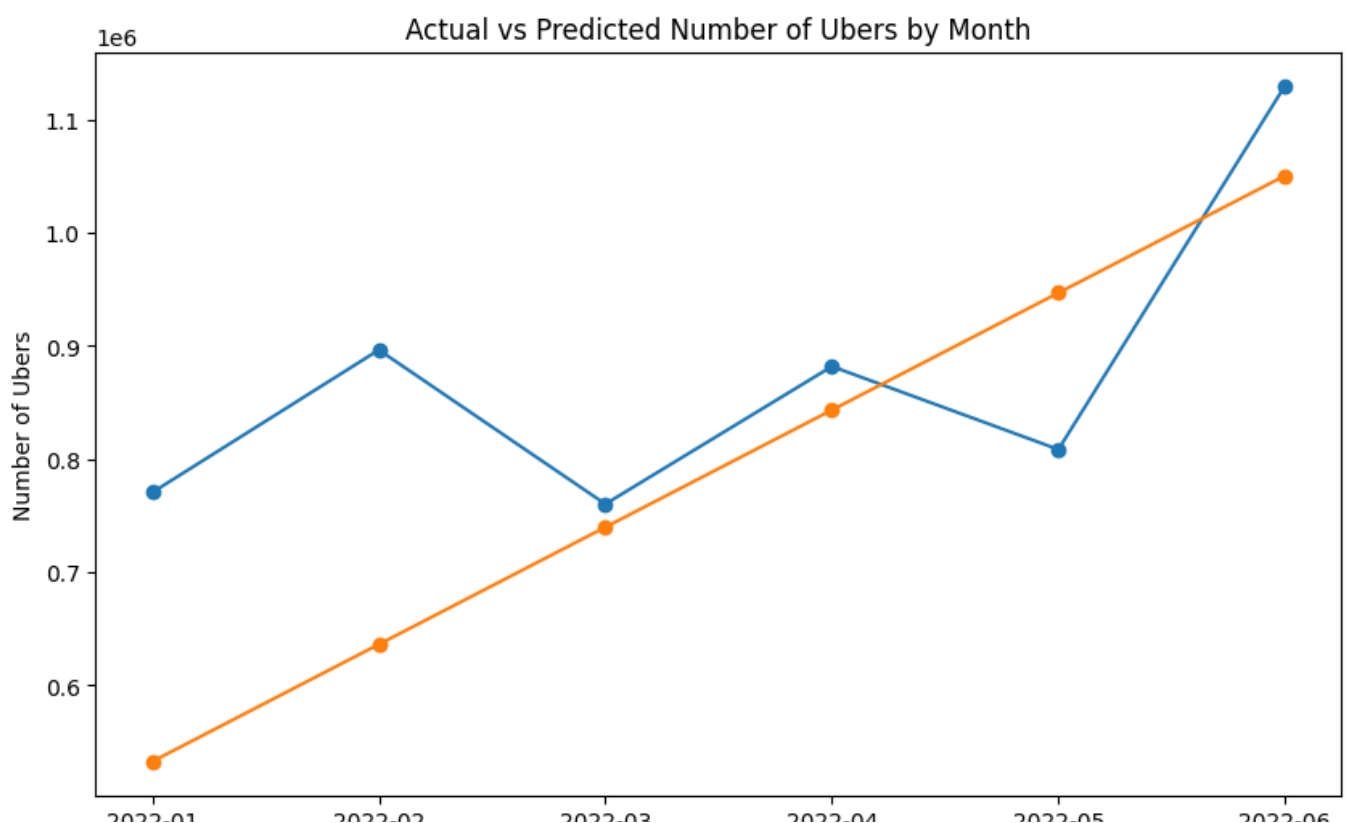
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(uber_data['Month_Num'].values.reshape

# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions for the entire dataset
uber_data['Predicted_Num_Ubers'] = model.predict(uber_data['Month_Num'].values.reshape

# Visualize the actual and predicted number of Ubers using a line plot
plt.figure(figsize=(10, 6))
plt.plot(uber_data['Month'], uber_data['Num_Ubers'], label='Actual', marker='o')
plt.plot(uber_data['Month'], uber_data['Predicted_Num_Ubers'], label='Predicted', mark
plt.xlabel("Month")
plt.ylabel("Number of Ubers")
plt.title("Actual vs Predicted Number of Ubers by Month")
plt.xticks
```

```
<function matplotlib.pyplot.xticks(ticks=None, labels=None, *, minor=False,
**kwargs)>
```



```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
import numpy as np

# Initialize the linear regression model
model = LinearRegression()

# Initialize the Lasso model
lasso_model = Lasso(alpha=0.1)

# Fit the linear regression model to the data
model.fit(X_train, y_train)

# Fit the Lasso model to the data
lasso_model.fit(X_train, y_train)

# Make predictions using the linear regression model
y_pred = model.predict(X_test)

# Make predictions using the Lasso model
lasso_pred = lasso_model.predict(X_test)

# Evaluate the performance of the linear regression model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE of the linear regression model:", rmse)

# Evaluate the performance of the Lasso model
rmse = np.sqrt(mean_squared_error(y_test, lasso_pred))
print("RMSE of the Lasso model:", rmse)

RMSE of the linear regression model: 249286.78679579054
RMSE of the Lasso model: 249286.5488360036
```

```
from matplotlib import pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso

# Assuming 'Pickup_date' column in 'uber' DataFrame contains date strings
# Replace 'your_dataset.csv' with the actual file path or dataset name
uber['Pickup_date'] = pd.to_datetime(uber['Pickup_date'])

uber['Pickup_date'] = uber['Pickup_date'].apply(lambda x: x.replace(year=2022))
uber['Month'] = uber['Pickup_date'].dt.to_period('M').astype('str') # Ensure 'Month'

# Extracting additional information for each highest date
uber_data = uber.groupby('Month').agg({'Pickup_date': 'max', 'locationID': 'count'}).r
uber_data = uber_data.rename(columns={'locationID': 'Num_Ubers'})

# Convert 'Month' to numeric for model training
uber_data['Month_Num'] = pd.to_datetime(uber_data['Month']).dt.month

# Drop rows with NaN values in the 'Month_Num' column
uber_data = uber_data.dropna(subset=['Month_Num'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(uber_data['Month_Num'].values.reshape(
# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Initialize the Lasso model
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)

# Make predictions for the entire dataset
uber_data['Predicted_Num_Ubers'] = model.predict(uber_data['Month_Num'].values.reshape
uber_data['Lasso_Predicted_Num_Ubers'] = lasso_model.predict(uber_data['Month_Num'].va
```

```
plt.figure(figsize=(10, 6))
plt.plot(uber_data['Month'], uber_data['Num_Ubers'], label='Actual')
plt.plot(uber_data['Month'], uber_data['Predicted_Num_Ubers'], label='Predicted')
plt.xlabel('Month')
plt.ylabel('Number of Ubers')
plt.legend()
plt.show()
```

