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

hour_df = pd.read_csv("hour.csv")
day_df = pd.read_csv("day.csv")

hour_df.head()
```


```
{"summary":"{\n  \"name\": \"hour_df\",\n  \"rows\": 17379,\n
\"fields\": [\n    {\n      \"column\": \"instant\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5017,\n        \"min\": 1,\n        \"max\": 17379,\n
\"num_unique_values\": 17379,\n        \"samples\": [\n
12831,\n          8689,\n          7092\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"dteday\",\n      \"properties\":
{\n        \"dtype\": \"object\",\n        \"num_unique_values\":
731,\n        \"samples\": [\n          \"2012-12-04\",\n
\"2011-02-03\",\n          \"2011-10-28\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"season\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 1,\n
\"min\": 1,\n        \"max\": 4,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          2,\n          4,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"yr\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"mnth\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3,\n        \"min\": 1,\n
\"max\": 12,\n        \"num_unique_values\": 12,\n        \"samples\":
[\n          11,\n          10\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"hr\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6,\n        \"min\": 0,\n
\"max\": 23,\n        \"num_unique_values\": 24,\n        \"samples\":
[\n          8,\n          16\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"holiday\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"weekday\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 0,\n
\"max\": 6,\n        \"num_unique_values\": 7,\n        \"samples\":
```

```
[\n          6,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"workingday\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"weathersit\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 4,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          2,\n          4\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"temp\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.19255612124972407,\n        \"min\":
0.02,\n        \"max\": 1.0,\n        \"num_unique_values\": 50,\n
\"samples\": [\n          0.16,\n          0.82\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"atemp\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\":
0.17185021563536587,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 65,\n        \"samples\": [\n          0.7879,\
n          0.9242\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hum\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.1929298340629125,\n        \"min\": 0.0,\n        \"max\":
1.0,\n        \"num_unique_values\": 89,\n        \"samples\": [\n
0.29,\n          0.61\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"windspeed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.12234022857279413,\n        \"min\":
0.0,\n        \"max\": 0.8507,\n        \"num_unique_values\": 30,\n
\"samples\": [\n          0.8507,\n          0.4925\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"casual\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 49,\n
\"min\": 0,\n        \"max\": 367,\n        \"num_unique_values\":
322,\n        \"samples\": [\n          201,\n          171\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"registered\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 151,\n        \"min\": 0,\n
\"max\": 886,\n        \"num_unique_values\": 776,\n
\"samples\": [\n          342,\n          744\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cnt\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 181,\n        \"min\": 1,\n
\"max\": 977,\n        \"num_unique_values\": 869,\n
\"samples\": [\n          594,\n          46\n        ],\n
```

\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"hour_df"}

```
day_df.head()
```

{"summary":"{\n \"name\": \"day_df\",\n \"rows\": 731,\n
\"fields\": [\n   {\n   \"column\": \"instant\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
211,\n   \"min\": 1,\n   \"max\": 731,\n
\"num_unique_values\": 731,\n   \"samples\": [\n   704,\n
34,\n   301\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"dteday\",\n   \"properties\": {\n   \"dtype\": \"object\",\n
\"num_unique_values\": 731,\n   \"samples\": [\n   \"2012-
12-04\",\n   \"2011-02-03\",\n   \"2011-10-28\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"season\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 1,\n
\"min\": 1,\n   \"max\": 4,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   2,\n   4,\n   1\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"yr\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"mnth\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 3,\n   \"min\": 1,\n
\"max\": 12,\n   \"num_unique_values\": 12,\n   \"samples\":
[\n   11,\n   10\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"holiday\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"weekday\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2,\n   \"min\": 0,\n
\"max\": 6,\n   \"num_unique_values\": 7,\n   \"samples\":
[\n   6,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"workingday\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"weathersit\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 1,\n
\"max\": 3,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   2,\n   1\n   ],\n   \"semantic_type\":

```
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"temp\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.18305099611148867,\n        \"min\":
0.0591304,\n        \"max\": 0.861667,\n        \"num_unique_values\":
499,\n        \"samples\": [\n          0.544167,\n          0.430435\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"atemp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.16296117838631127,\n        \"min\": 0.0790696,\n
\"max\": 0.840896,\n        \"num_unique_values\": 690,\n
\"samples\": [\n          0.463375,\n          0.599754\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hum\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.14242909513835394,\n
\"min\": 0.0,\n        \"max\": 0.9725,\n
\"num_unique_values\": 595,\n        \"samples\": [\n
0.707083,\n          0.718333\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"windspeed\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.07749787068166943,\n
\"min\": 0.0223917,\n        \"max\": 0.507463,\n
\"num_unique_values\": 650,\n        \"samples\": [\n
0.100742,\n          0.139308\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"casual\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 686,\n        \"min\": 2,\n
\"max\": 3410,\n        \"num_unique_values\": 606,\n
\"samples\": [\n          709,\n          449\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"registered\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1560,\n        \"min\": 20,\n        \"max\": 6946,\n
\"num_unique_values\": 679,\n        \"samples\": [\n          4531,\n
2553\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cnt\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1937,\n        \"min\": 22,\n        \"max\": 8714,\n
\"num_unique_values\": 696,\n        \"samples\": [\n          5170,\n
1607\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"day_df"}
```


```
hour_df.dtypes
```

```
instant         int64
dteday         object
season          int64
yr              int64
mnth            int64
hr              int64
```

```
holiday          int64
weekday          int64
workingday       int64
weathersit       int64
temp           float64
atemp          float64
hum            float64
windspeed      float64
casual           int64
registered       int64
cnt              int64
dtype: object
```

```
day_df.dtypes
```

```
instant          int64
dteday          object
season           int64
yr               int64
mnth             int64
holiday          int64
weekday          int64
workingday       int64
weathersit       int64
temp           float64
atemp          float64
hum            float64
windspeed      float64
casual           int64
registered       int64
cnt              int64
dtype: object
```

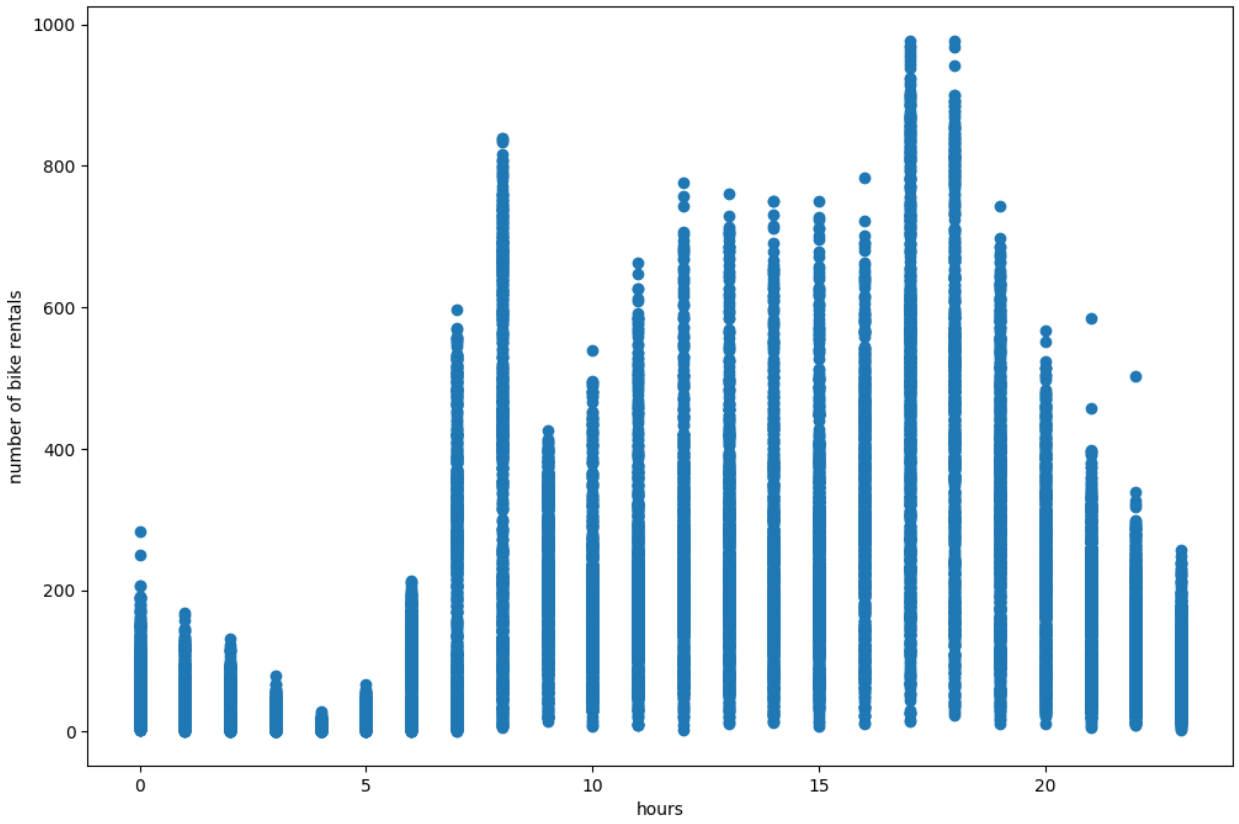
```
hour_df["dteday"] = pd.to_datetime(hour_df["dteday"])
day_df["dteday"] = pd.to_datetime(day_df["dteday"])
```

```
hour_df.isnull().sum()
day_df.isnull().sum()
```

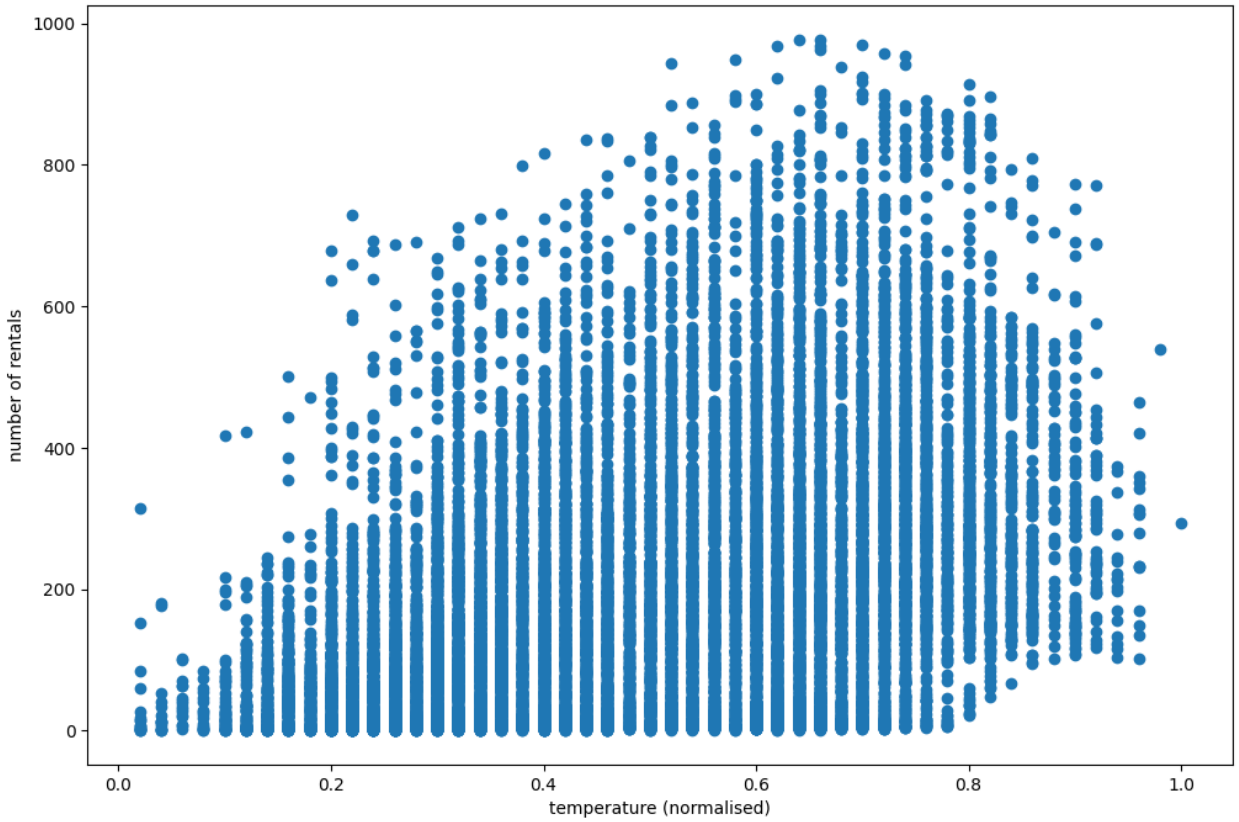
```
instant       0
dteday        0
season        0
yr            0
mnth          0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
```

```
hum           0
windspeed     0
casual        0
registered    0
cnt           0
dtype: int64
```

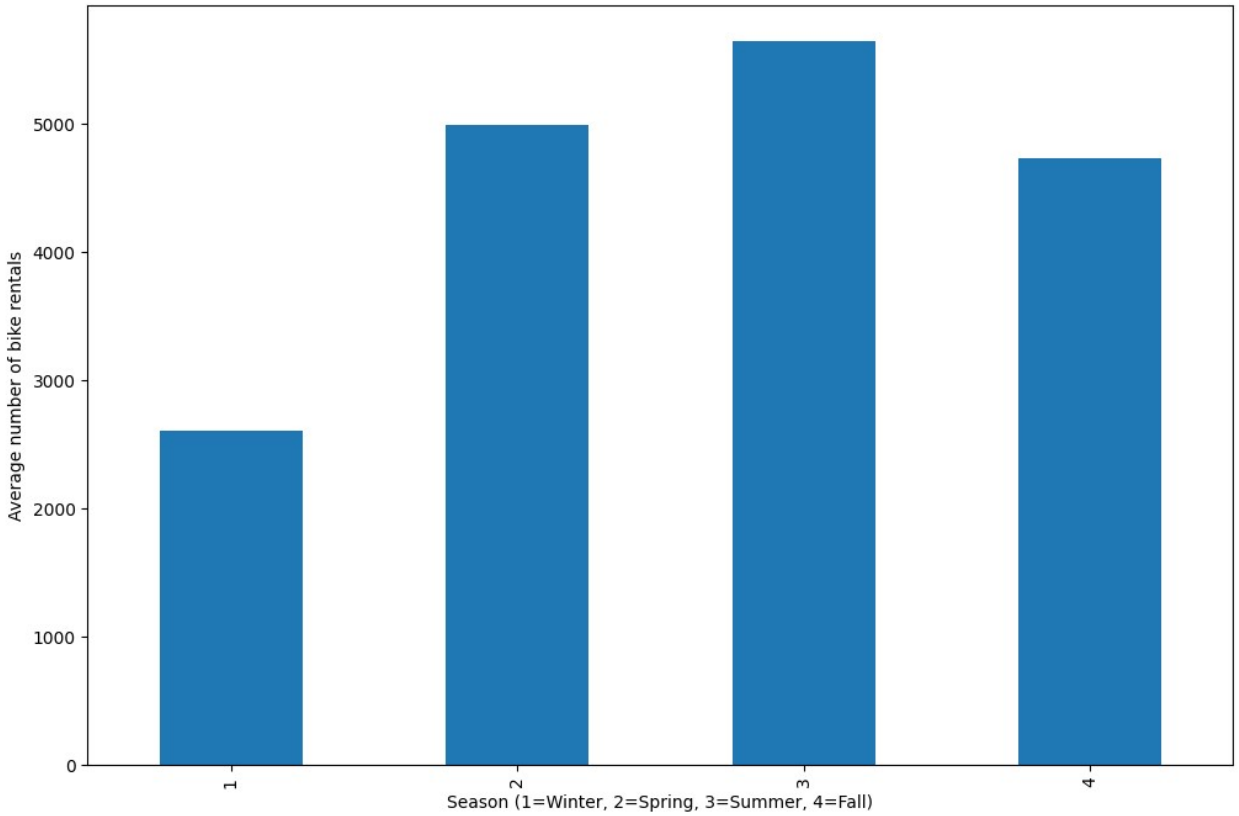
```
plt.figure(figsize = (12,8))
plt.plot(hour_df["hr"], hour_df["cnt"], "o")
plt.xlabel("hours")
plt.ylabel("number of bike rentals")
plt.show()
```




```
plt.figure(figsize = (12,8))
plt.scatter(hour_df["temp"], hour_df["cnt"])
plt.xlabel("temperature (normalised)")
plt.ylabel("number of rentals")
plt.show()
```

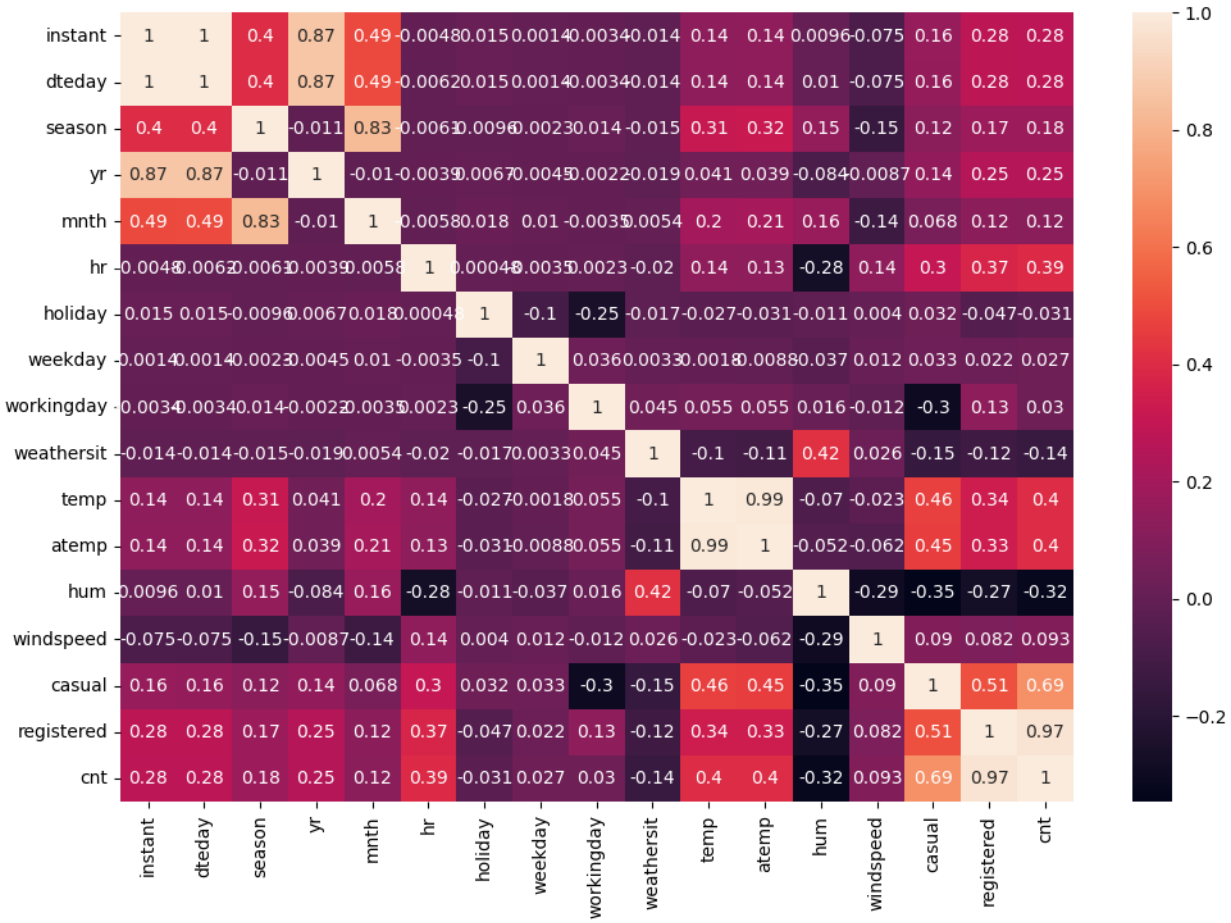



```
plt.figure(figsize=(12, 8))
season_avg = day_df.groupby("season")["cnt"].mean()
season_avg.plot(kind = "bar")
plt.xlabel("Season (1=Winter, 2=Spring, 3=Summer, 4=Fall)")
plt.ylabel("Average number of bike rentals")
plt.show()
```




```
corelation = hour_df.corr()
plt.figure(figsize = (12,8))
sns.heatmap(corelation, annot = True)
print(corelation["cnt"].sort_values(ascending = False))
```

```
cnt           1.000000
registered    0.972151
casual        0.694564
temp          0.404772
atemp         0.400929
hr            0.394071
instant       0.278379
dteday        0.277753
yr            0.250495
season        0.178056
mnth          0.120638
windspeed     0.093234
workingday    0.030284
weekday       0.026900
holiday      -0.030927
weathersit   -0.142426
hum          -0.322911
Name: cnt, dtype: float64
```

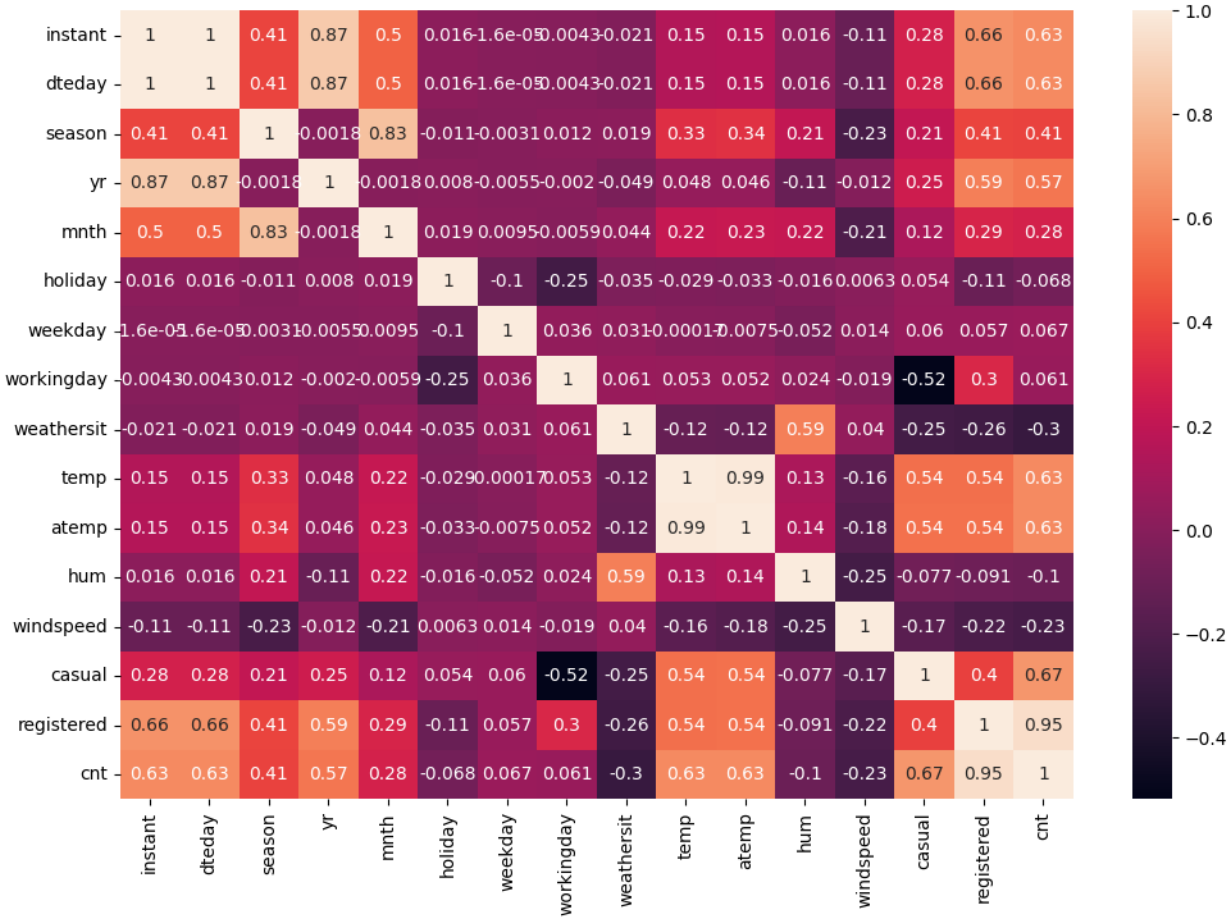



```
hour_df.drop(columns = ["registered", "casual", "atemp","yr",
"season", "mnth", "windspeed", "workingday",
"weekday","holiday","weathersit"], inplace=True)

corelation = day_df.corr()
plt.figure(figsize = (12,8))
sns.heatmap(corelation, annot = True)
print(corelation["cnt"].sort_values(ascending = False))
```

```
cnt           1.000000
registered    0.945517
casual        0.672804
atemp         0.631066
instant       0.628830
dteday        0.628830
temp          0.627494
yr            0.566710
season        0.406100
mnth          0.279977
weekday       0.067443
workingday    0.061156
holiday      -0.068348
```

```
hum           -0.100659
windspeed     -0.234545
weathersit    -0.297391
Name: cnt, dtype: float64
```




```
day_df.drop(columns = ["registered", "casual","atemp","holiday",
"workingday", "weekday", "hum"], inplace=True)

def entropy(y):
    class_counts = np.bincount(y)
    probabilities = class_counts / len(y)
    entropy = -np.sum([p * np.log2(p) for p in probabilities if p >
0])
    return entropy

def information_gain(X, y, feature_idx, threshold):
    left_idx = X[:, feature_idx] <= threshold
    right_idx = X[:, feature_idx] > threshold
    if len(left_idx) == 0 or len(right_idx) == 0:
        return 0
    n = len(y)
```

```
    n_left, n_right = len(y[left_idx]), len(y[right_idx])
    entropy_left = entropy(y[left_idx])
    entropy_right = entropy(y[right_idx])
    weighted_entropy = (n_left / n) * entropy_left + (n_right / n) *
entropy_right
    return entropy(y) - weighted_entropy

class DecisionTreeClassifierScratch:
    def __init__(self, max_depth=5, min_samples_split=2):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.tree = None

    def find_best_split(self, X, y):
        best_gain = 0
        best_split = None
        for feature_idx in range(X.shape[1]):
            thresholds = np.unique(X[:, feature_idx])
            for threshold in thresholds:
                gain = information_gain(X, y, feature_idx, threshold)
                if gain > best_gain:
                    best_gain = gain
                    best_split = {
                        'feature_idx': feature_idx,
                        'threshold': threshold,
                        'gain': gain
                    }
        return best_split

    def build_tree(self, X, y, depth=0):
        # Stopping criteria
        if depth == self.max_depth or len(y) < self.min_samples_split
or entropy(y) == 0:
            return np.bincount(y).argmax()

        # Find the best split
        best_split = self.find_best_split(X, y)
        if not best_split:
            return np.bincount(y).argmax()

        # Split the data
        left_idx = X[:, best_split["feature_idx"]] <=
best_split["threshold"]
        right_idx = X[:, best_split["feature_idx"]] >
best_split["threshold"]

        # Recursively build the tree
        left_branch = self.build_tree(X[left_idx], y[left_idx], depth
+ 1)
        right_branch = self.build_tree(X[right_idx], y[right_idx],
```

```
depth + 1)

        return {
            'feature_idx': best_split['feature_idx'],
            'threshold': best_split['threshold'],
            'left_branch': left_branch,
            'right_branch': right_branch
        }

    def fit(self, X, y):
        self.tree = self.build_tree(X, y)

    def predict_one(self, x, tree):
        if isinstance(tree, dict):
            if x[tree['feature_idx']] <= tree['threshold']:
                return self.predict_one(x, tree['left_branch'])
            else:
                return self.predict_one(x, tree['right_branch'])
        return tree

    def predict(self, X):
        return np.array([self.predict_one(x, self.tree) for x in X])

# Add demand_category column based on the mean of 'cnt'
hour_df["demand_category"] = np.where(hour_df["cnt"] >
hour_df["cnt"].mean(), 1, 0)

X = hour_df.drop(columns=["cnt", "demand_category"])
y = hour_df["demand_category"]

# Split data into training and test sets
train_ratio = 0.8
train_size = int(train_ratio * len(X))
indices = np.random.permutation(len(X))
train_idx, test_idx = indices[:train_size], indices[train_size:]
X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]


X_train_np = X_train.values
y_train_np = y_train.values
X_test_np = X_test.values
y_test_np = y_test.values

model = DecisionTreeClassifierScratch(max_depth=5)
model.fit(X_train_np, y_train_np)


y_pred = model.predict(X_test_np)
accuracy = np.sum(y_pred == y_test_np) / len(y_test_np)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.8311277330264673
```

```
day_df["demand_category"] = np.where(day_df["cnt"] >
day_df["cnt"].mean(), 1, 0)

X = day_df.drop(columns=["cnt", "demand_category"])
y = day_df["demand_category"]

# Split data into training and test sets
train_ratio = 0.8
train_size = int(train_ratio * len(X))
indices = np.random.permutation(len(X))
train_idx, test_idx = indices[:train_size], indices[train_size:]
X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

# Convert pandas DataFrames to numpy arrays
X_train_np = X_train.values
y_train_np = y_train.values
X_test_np = X_test.values
y_test_np = y_test.values

# Train the model
model = DecisionTreeClassifierScratch(max_depth=5)
model.fit(X_train_np, y_train_np)

# Make predictions
y_pred = model.predict(X_test_np)

# Calculate accuracy
accuracy = np.sum(y_pred == y_test_np) / len(y_test_np)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9591836734693877
```

some extra analysis on data I did (took help of pre built kmeans and libraries)

```
from sklearn.preprocessing import StandardScaler
hour_df = pd.read_csv("hour.csv")
day_df = pd.read_csv("day.csv")
# Select features for clustering
features = ['hr', 'temp', 'hum']
X = hour_df[features]

X = pd.get_dummies(X, drop_first=True)
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

from sklearn.cluster import KMeans
k = 5
kmeans = KMeans(n_clusters = 5, random_state = 42)
clusters = kmeans.fit_predict(X_scaled)

hour_df["demand_cluster"] = clusters

#average for each cluster
cluster_analysis = hour_df.groupby("demand_cluster")
["cnt"].mean().reset_index()
print(cluster_analysis)
```

```
   demand_cluster         cnt
0               0  193.462424
1               1   82.635835
2               2  201.021407
3               3  120.743869
4               4  356.380163
```

```
low = 120
high = 250
def label_demand_type(avg_count):
    if avg_count < low:
        return "Low demand"
    elif avg_count <= high:
        return "Medium demand"
    else:
        return "High demand"

# Apply the function to categorize demand
cluster_analysis['demand_type'] =
cluster_analysis['cnt'].apply(label_demand_type)
print(cluster_analysis)
```

```
   demand_cluster         cnt    demand_type
0               0  193.462424  Medium demand
1               1   82.635835     Low demand
2               2  201.021407  Medium demand
3               3  120.743869  Medium demand
4               4  356.380163    High demand
```

```
demand_type_mapping = cluster_analysis.set_index('demand_cluster')
['demand_type'].to_dict()
color_map = {
    'Low demand': 'red',
    'Medium demand': 'orange',
```

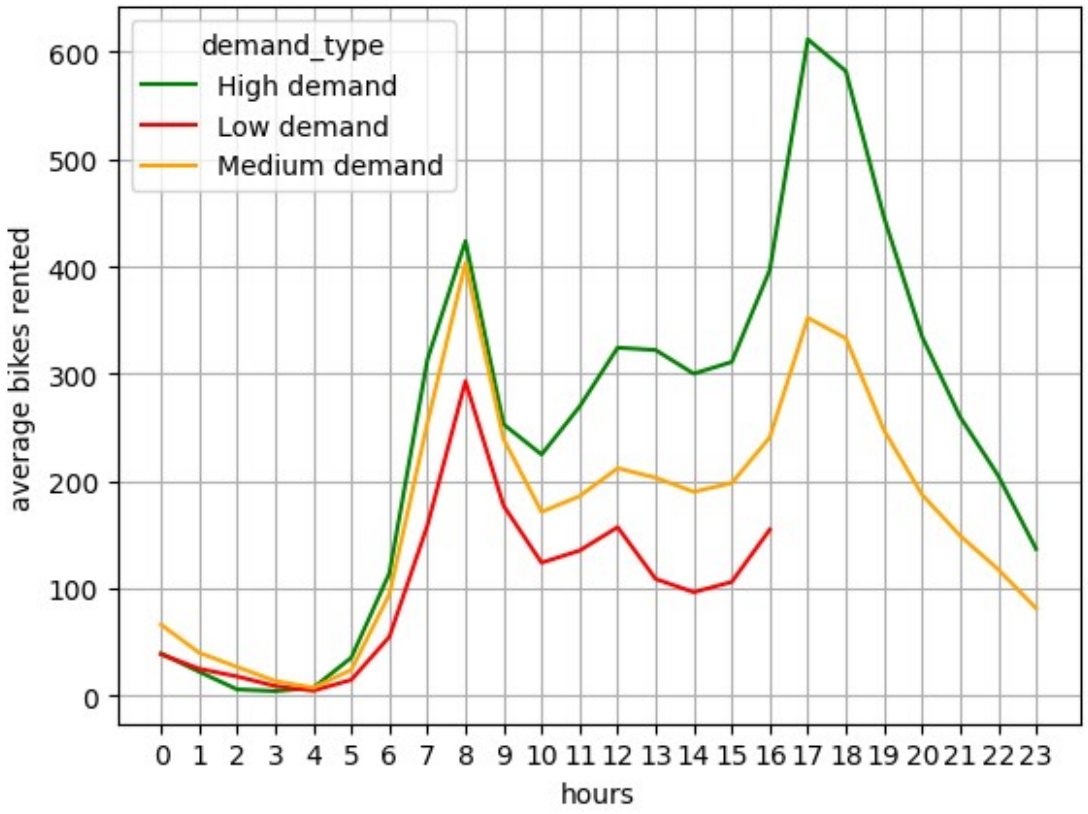
```
    'High demand': 'green'
}
hour_df['demand_type'] =
hour_df['demand_cluster'].map(demand_type_mapping)
```

hour of days vs demand type graph

```
hourly_avg = hour_df.groupby(['hr', 'demand_type'])
['cnt'].mean().unstack()

plt.figure(figsize=(12, 6))
hourly_avg.plot(kind="line", color=[color_map[d] for d in
hourly_avg.columns])
plt.xlabel("hours")
plt.ylabel("average bikes rented")
plt.xticks(range(0, 24))
plt.grid()
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



different humidity and temperatures vs rental demands

```
plt.figure(figsize=(12, 8))
sns.scatterplot(data=hour_df, x = "temp", y = "hum", hue =
"demand_type", palette=color_map, alpha=0.7)
plt.xlabel("temp")
plt.ylabel("hum")
plt.show()
```