In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import statsmodels.api as sm
from scipy.stats import chi2_contingency
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, classification_report, confusion_matrix
```

In [2]:
```
# read in dataset
data = pd.read_csv('Occupancy.csv')
```

In [3]:
```
# create datetime variable and drop date object variable
data['Datetime'] = pd.to_datetime(data['date'])
```

In [4]:
```
# check for missing values
data.isna().sum()
```

Out[4]:
```
date            0
Temperature     0
Humidity        0
Light           0
CO2             0
HumidityRatio   0
Occupancy       0
Datetime        0
dtype: int64
```

In [5]:
```
# reduce Light variable to binary
data['Light_on_off'] = np.where(data['Light'] > 0, 1, 0)
# get hour number from datetime column
data['Hour'] = data['Datetime'].dt.hour
# getting the weekday from the 'date' column
data['DOTW'] = pd.to_datetime(data['date']).dt.day_name()
# making the column 'DOTW' to be catergorical and will now be changed to numerical values representing the days of the week
day_mapping = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
    'Sunday': 7
}
# now mapping the new values to the 'DOTW' column
data['DOTW_encoded'] = data['DOTW'].map(day_mapping)
# create a binary variable for weekday or not
data['Weekday'] = np.where(data['DOTW_encoded'] < 6, 1, 0)
# create a binary variable for working hours or not
data['Working_Hours'] = np.where(np.logical_and(data['Hour'] >= 6, data['Hour'] <= 18), 1, 0)
# create a binary variable for working hours on a weekday
data['Workdayhrs'] = np.where(np.logical_and(data['Weekday'] == 1, data['Working_Hours'] == 1), 1, 0)
```

In [6]:
```
# function to split train/test two different ways, then run regression

def traintest1(x_train, x_test, y_train, y_test, var):
    # Create a logistic regression model
    model1 = LogisticRegression(max_iter=500)

    # Fit the model to the training data
    model1.fit(x_train, y_train)

    # Make predictions on the test data
    y_pred = model1.predict(x_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)

    # create a list of accuracy score to return
    acc_list = [var,round(accuracy,8)]

    return acc_list
```

In [7]:
```
# function to run all the predictor variables individually in regression and capture the statistics
def runreg(var, input_y, num):

    # x is the independent variable that influences y
    x1 = pd.DataFrame(data[var])

    # define the target
    # y is the dependent variable we are trying to predict
    #y1 = pd.DataFrame(input_y)
    y1 = input_y

    # Split the data into training and testing sets
    # a: random split
    x_atrain, x_atest, y_atrain, y_atest = train_test_split(x1, y1, test_size=0.2, random_state=42)
    # b: chronological/sequential split
    x_btrain, x_btest, y_btrain, y_btest = train_test_split(x1, y1, test_size=0.2, shuffle=False)

    # run regression and create a list of accuracy score to return
    acc_list_random = traintest1(x_atrain, x_atest, y_atrain, y_atest, var)
    acc_list_seq = traintest1(x_btrain, x_btest, y_btrain, y_btest, var)

    return acc_list_random,acc_list_seq
```

In [8]:
```
#LOGISTIC REGRESSION: DEPENDENT VARIABLES ARE BINARY

# create function to repeat the regression
def repeat_reg1(x_indep,y_dep,factor_list):

    # Split the data into training and testing sets
    x_indepa_train, x_indepa_test, y_depa_train, y_depa_test = train_test_split(x_indep, y_dep, test_size=0.2, random_state=42)
    x_indepb_train, x_indepb_test, y_depb_train, y_depb_test = train_test_split(x_indep, y_dep, test_size=0.2, shuffle=False)

    # Create a logistic regression model
    model_reg1 = LogisticRegression(max_iter=500)
    model_reg2 = LogisticRegression(max_iter=500)
```

```
    # Fit the model to the training data
    model_reg1.fit(x_indepa_train, y_depa_train)
    model_reg2.fit(x_indepb_train, y_depb_train)

    # Make predictions on the test data
    y_depa_pred = model_reg1.predict(x_indepa_test)
    y_depb_pred = model_reg2.predict(x_indepb_test)

    # Evaluate the model
    accuracy_reg1 = accuracy_score(y_depa_test, y_depa_pred)
    accuracy_reg2 = accuracy_score(y_depb_test, y_depb_pred)

    # save accuracy to list
    acc_list1 = ["Grouped Predictors",round(accuracy_reg1,8)]
    acc_list2 = ["Grouped Predictors",round(accuracy_reg2,8)]

    # create list for returning all accuracy scores
    acc_list_r = list()
    acc_list_s = list()

    # append grouped predictors
    acc_list_r.append(acc_list1)
    acc_list_s.append(acc_list2)

    # RUN REGRESSION FOR EACH VARIABLE SEPARATELY
    # iterate through factor_list
    for idx, x in enumerate(factor_list):
        acc_list_random,acc_list_seq = runreg(x, y_dep, idx+1)
        acc_list_r.append(acc_list_random)
        acc_list_s.append(acc_list_seq)

    return acc_list_r,acc_list_s
```

In [9]:
```
# dependent variable (Y) is Occupancy (binary 0/1)
# independent variables (X):
#     Float: Temperature, Humidity, Light, CO2
#     Integer: Hour, DOTW_encoded
#     Binary: Light_on_off, Weekday, Working_Hours, Workdayhrs

# Split the dataset into independent variables (X) and the dependent variable (y)
y_dep1 = data["Occupancy"]
x_indep1 = data[["Temperature","Humidity","Light","CO2","DOTW_encoded","Hour"]]
factor_list1 = ["Temperature","Humidity","Light","CO2","DOTW_encoded","Hour","Light_on_off","Weekday","Working_Hours","Workdayhrs"]

# run regressions
acc_list_occ_r,acc_list_occ_s = repeat_reg1(x_indep1,y_dep1,factor_list1)
```

In [10]:
```
# dependent variable (Y) is Weekday (binary 0/1)
# independent variables (X):
#     Float: Temperature, Humidity, Light, CO2
#     Integer: Hour
#     Binary: Light_on_off, Occupancy, Working_Hours
```

```
# Split the dataset into independent variables (X) and the dependent variable (y)
y_dep2 = data["Weekday"]
x_indep2 = data[["Temperature","Humidity","Light","CO2","Occupancy"]]
factor_list2 = ["Temperature","Humidity","Light","CO2","Light_on_off","Occupancy"]

# run regressions
acc_list_wkdy_r,acc_list_wkdy_s = repeat_reg1(x_indep2,y_dep2,factor_list2)
```

In [11]:
```
# dependent variable (Y) is Working_Hours (binary 0/1)
# independent variables (X):
#     Float: Temperature, Humidity, Light, CO2
#     Integer: DOTW_encoded
#     Binary: Light_on_off, Occupancy, Weekday

# Split the dataset into independent variables (X) and the dependent variable (y)
y_dep3 = data["Working_Hours"]
x_indep3 = data[["Temperature","Humidity","Light","CO2","Occupancy"]]
factor_list3 = ["Temperature","Humidity","Light","CO2","Light_on_off","Occupancy"]

# run regressions
acc_list_hrs_r,acc_list_hrs_s = repeat_reg1(x_indep3,y_dep3,factor_list3)
```

In [12]:
```
# dependent variable (Y) is Workdayhrs (binary 0/1)
# independent variables (X):
#     Float: Temperature, Humidity, Light, CO2
#     Integer: DOTW_encoded, Hour
#     Binary: Light_on_off, Occupancy

# Split the dataset into independent variables (X) and the dependent variable (y)
y_dep4 = data["Workdayhrs"]
x_indep4 = data[["Temperature","Humidity","Light","CO2","Occupancy"]]
factor_list4 = ["Temperature","Humidity","Light","CO2","Light_on_off","Occupancy"]

# run regressions
acc_list_wkdyhrs_r,acc_list_wkdyhrs_s = repeat_reg1(x_indep4,y_dep4,factor_list4)
```

In [13]:
```
# dependent variable (Y) is Light_on_off (binary 0/1)
# independent variables (X):
#     Float: Temperature, Humidity, CO2
#     Integer: DOTW_encoded, Hour, Working_Hours
#     Binary: Light_on_off, Occupancy, Weekday, Workdayhrs

# Split the dataset into independent variables (X) and the dependent variable (y)
y_dep5 = data["Light_on_off"]
x_indep5 = data[["Temperature","Humidity","CO2","Occupancy","DOTW_encoded","Hour"]]
factor_list5 = ["Temperature","Humidity","CO2","Occupancy","DOTW_encoded","Hour","Weekday","Working_Hours","Workdayhrs"]

# run regressions
acc_list_onoff_r,acc_list_onoff_s = repeat_reg1(x_indep5,y_dep5,factor_list5)
```

In [14]:
```
# create table from accuracy list
df1 = pd.DataFrame(acc_list_occ_r, columns = ['Random Split','Accuracy Score for Occupancy'])
df1.set_index('Random Split', inplace=True)
test1 = df1.transpose()
# create table from accuracy list
```

```
df2 = pd.DataFrame(acc_list_wkdy_r, columns = ['Random Split','Accuracy Score for Weekday'])
df2.set_index('Random Split', inplace=True)
test2 = df2.transpose()
# create table from accuracy list
df3 = pd.DataFrame(acc_list_hrs_r, columns = ['Random Split','Accuracy Score for Working_Hours'])
df3.set_index('Random Split', inplace=True)
test3 = df3.transpose()
# create table from accuracy list
df4 = pd.DataFrame(acc_list_wkdyhrs_r, columns = ['Random Split','Accuracy Score for Workdayhrs'])
df4.set_index('Random Split', inplace=True)
test4 = df4.transpose()
# create table from accuracy list
df5 = pd.DataFrame(acc_list_onoff_r, columns = ['Random Split','Accuracy Score for Light_on_off'])
df5.set_index('Random Split', inplace=True)
test5 = df5.transpose()
```

In [15]:
```
# create table from accuracy list
df_concat = pd.concat([test1,test2,test3,test4,test5], join='outer')
df_concat
```

Out[15]:

Random Split	Grouped Predictors	Temperature	Humidity	Light	CO2	DOTW_encoded	Hour	Light_on_off	Weekday	Working_Hours	Workdayhrs	Occupancy
Accuracy Score for Occupancy	0.992218	0.812014	0.776265	0.989786	0.790370	0.776265	0.776265	0.845331	0.776265	0.776265	0.864786	NaN
Accuracy Score for Weekday	0.830982	0.812743	0.707442	0.705253	0.760944	NaN	NaN	0.705253	NaN	NaN	NaN	0.705253
Accuracy Score for Working_Hours	0.854086	0.672909	0.521887	0.854086	0.630593	NaN	NaN	0.854086	NaN	NaN	NaN	0.699416
Accuracy Score for Workdayhrs	0.877432	0.809582	0.641051	0.851167	0.766294	NaN	NaN	0.778697	NaN	NaN	NaN	0.864786
Accuracy Score for Light_on_off	0.905399	0.822471	0.621595	NaN	0.698200	0.621595	0.621595	NaN	0.621595	0.854086	0.778697	0.845331

In [16]:
```
# create table from accuracy list
df1 = pd.DataFrame(acc_list_occ_s, columns = ['Chronological Split','Accuracy Score for Occupancy'])
df1.set_index('Chronological Split', inplace=True)
test1 = df1.transpose()
# create table from accuracy list
df2 = pd.DataFrame(acc_list_wkdy_s, columns = ['Chronological Split','Accuracy Score for Weekday'])
df2.set_index('Chronological Split', inplace=True)
test2 = df2.transpose()
# create table from accuracy list
df3 = pd.DataFrame(acc_list_hrs_s, columns = ['Chronological Split','Accuracy Score for Working_Hours'])
df3.set_index('Chronological Split', inplace=True)
test3 = df3.transpose()
# create table from accuracy list
df4 = pd.DataFrame(acc_list_wkdyhrs_s, columns = ['Chronological Split','Accuracy Score for Workdayhrs'])
df4.set_index('Chronological Split', inplace=True)
test4 = df4.transpose()
# create table from accuracy list
df5 = pd.DataFrame(acc_list_onoff_s, columns = ['Chronological Split','Accuracy Score for Light_on_off'])
```

```
df5.set_index('Chronological Split', inplace=True)
test5 = df5.transpose()
```

In [17]:
```
# create table from accuracy list
df_concat = pd.concat([test1,test2,test3,test4,test5], join='outer')
df_concat
```

Out[17]:

Chronological Split	Grouped Predictors	Temperature	Humidity	Light	CO2	DOTW_encoded	Hour	Light_on_off	Weekday	Working_Hours	Workdayhrs	Occupancy
Accuracy Score for Occupancy	0.995379	0.747325	0.733220	0.995379	0.664640	0.733220	0.733220	0.896158	0.733220	0.73322	0.838765	NaN
Accuracy Score for Weekday	0.834144	0.803988	0.836576	0.836576	0.836576	NaN	NaN	0.836576	NaN	NaN	NaN	0.836576
Accuracy Score for Working_Hours	0.736868	0.723979	0.518482	0.852140	0.518482	NaN	NaN	0.852140	NaN	NaN	NaN	0.748298
Accuracy Score for Workdayhrs	0.753891	0.751459	0.571984	0.841926	0.435311	NaN	NaN	0.794747	NaN	NaN	NaN	0.838765
Accuracy Score for Light_on_off	0.880593	0.877675	0.629377	NaN	0.339251	0.629377	0.629377	NaN	0.629377	0.85214	0.794747	0.896158