NAGARJUNA COLLEGE OF ENGINERING AND TECHNOLOGY
(An Autonomous Institute under VTU)
Mudugurki, Venkatgirikote, Devanahalli, Karnataka, Bengaluru-562164.

Department of CSE (Data Science)

Laboratory Manual

VI Semester – 2024

Statistical Inference for Data Science

(21CDI62)

Prepared by

Prof. Subhakar M. & Prof. Pooja Ahuja. S.

Assistant Professor
Department of CSE (Data Science)

PRACTICAL COMPONENTS

Sl. No	Experiments
1	Rainfall prediction data set – draw correlation between the features
2	Find the outliers in the Housing Price dataset
3	For a given dataset, display a chosen feature using different mean values
4	Display the confidence interval of a chosen feature based on a sample
5	Perform t-test on a feature in a dataset
6	Create Boxplots for different groups of a feature
7	Create a Linear Regression model for a dataset and display the error measures
8	Chose a dataset with categorical data and apply linear regression model
9	Apply Naive Bayes algorithm on a dataset and estimate the accuracy
10	Apply Logistic Regression algorithm on a dataset and estimate the accuracy

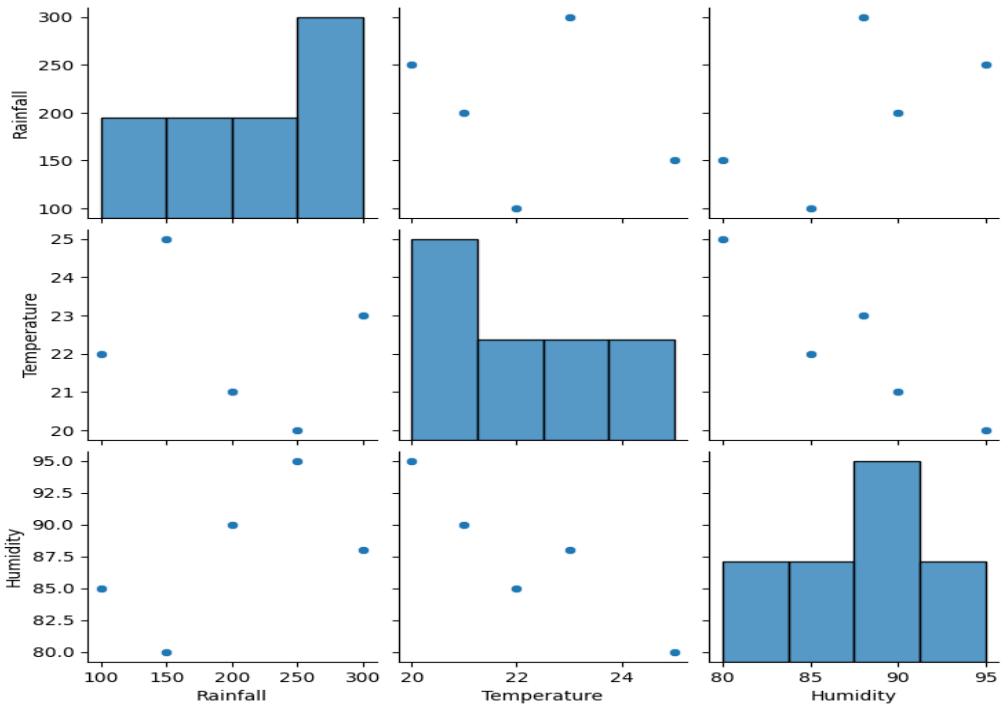
Lab Manual

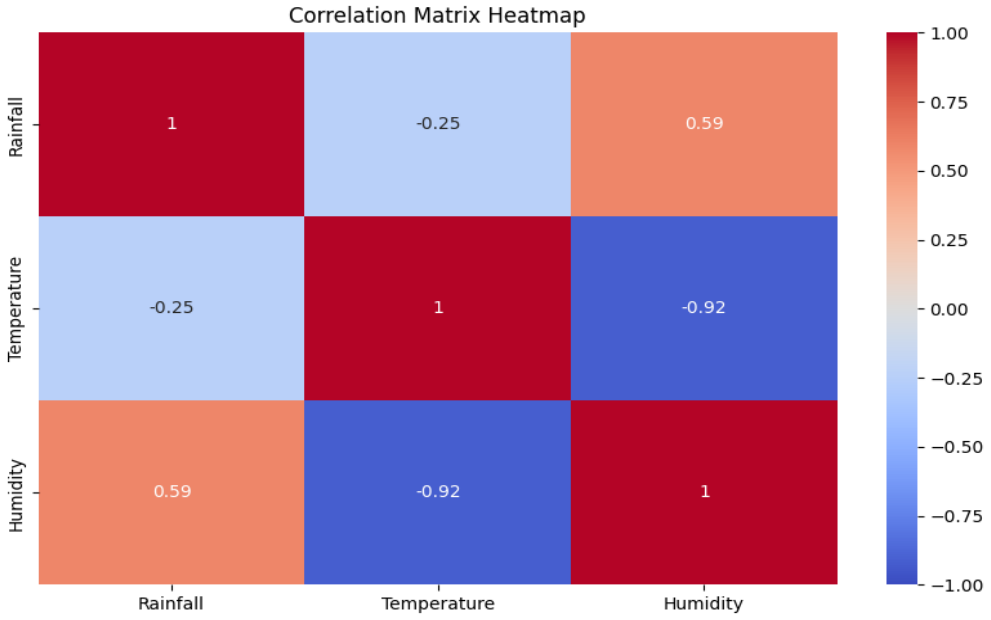
Statistical Inference for Data Science (21CDI62)

Week 1: Rainfall prediction data set – draw correlation between the features

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
data = {
   'Rainfall': [100, 150, 200, 250, 300],
   'Temperature': [22, 25, 21, 20, 23],
   'Humidity': [85, 80, 90, 95, 88]
}
df = pd.DataFrame(data)
correlation_matrix = df.corr()
print(correlation_matrix)
sns.pairplot(df)
plt.show()
```

```
                Rainfall      Temperature     Humidity
Rainfall        1.000000     -0.246598        0.593495
Temperature    -0.246598      1.000000       -0.919946
Humidity        0.593495     -0.919946        1.000000
```




Correlation Matrix Heatmap
Rainfall
Temperature
Humidity
1
-0.25
0.59
-0.25
1
-0.92
0.59
-0.92
1
Rainfall
Temperature
Humidity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

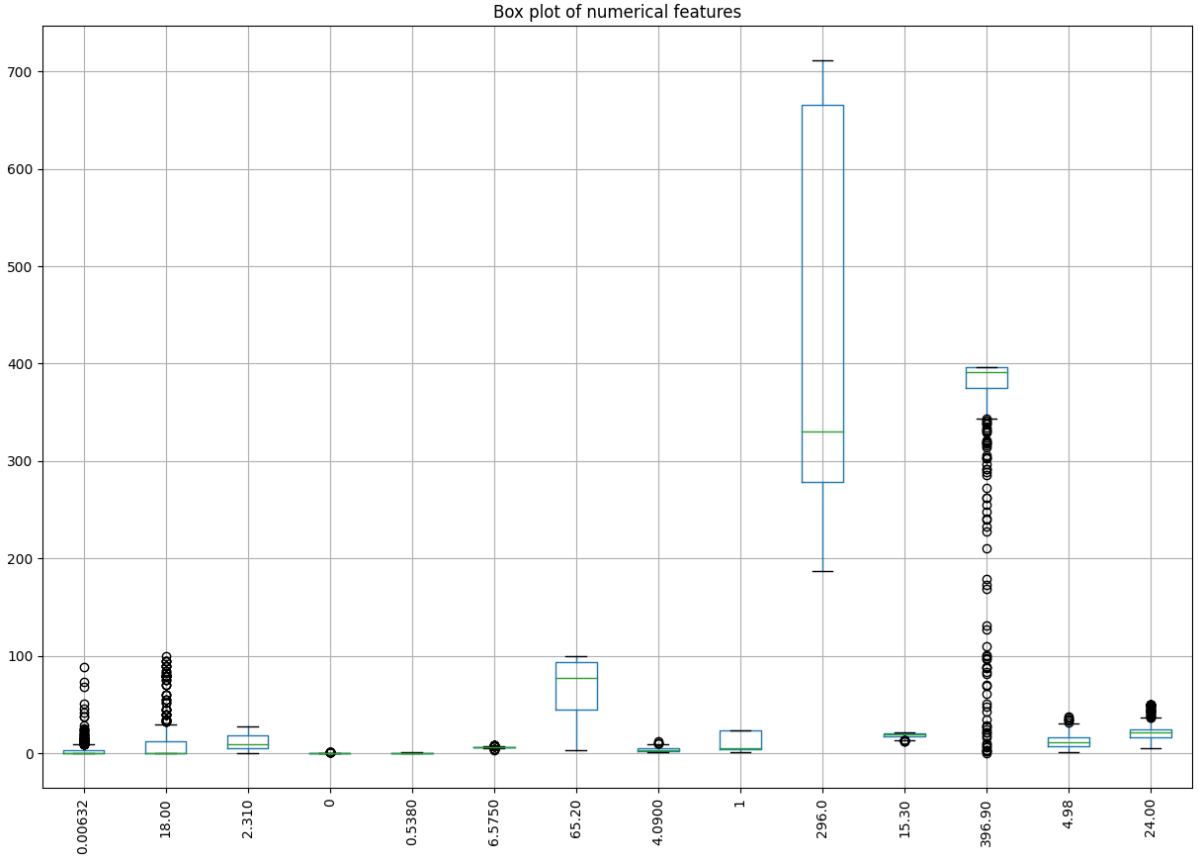
Week 2: Find the outliers in the Housing Price dataset

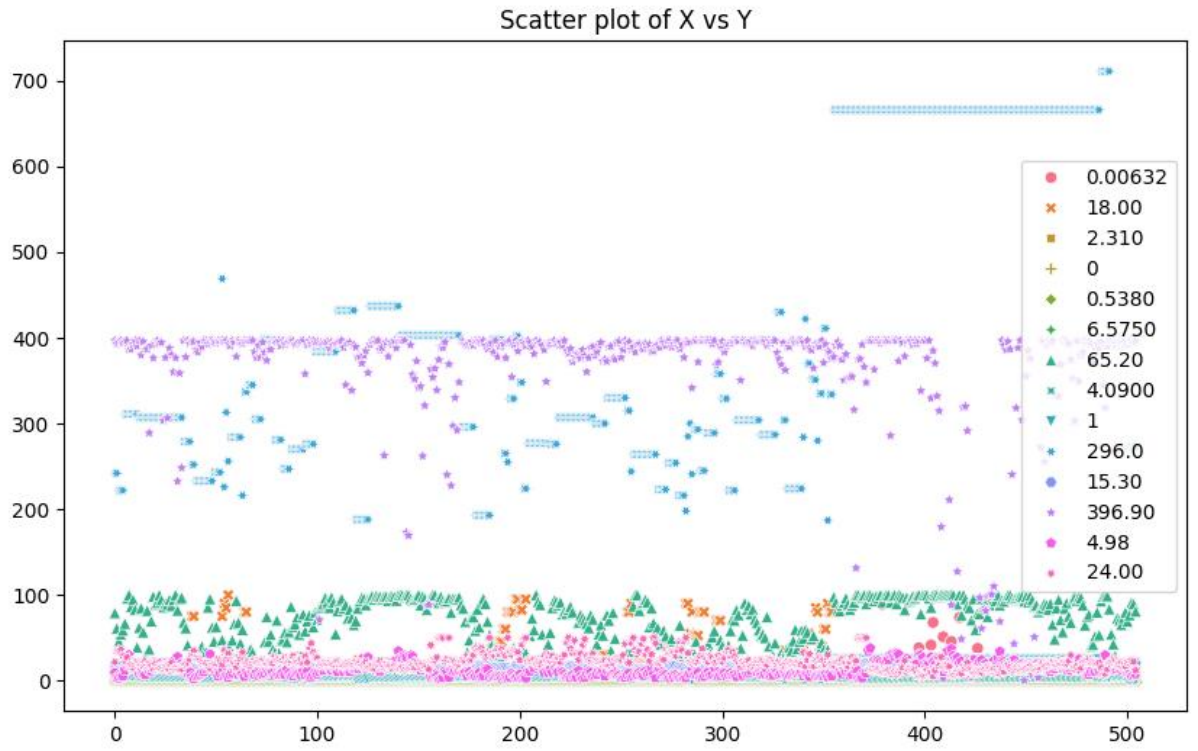
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

df = pd.read_csv('housing_data.csv')
print(df.head())
print(df.describe())
z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
outliers_z = np.where(z_scores > 3, True, False)
print("Outliers detected by Z-score method:\n", df[outliers_z.any(axis=1)])
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers_iqr = (df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))
print("Outliers detected by IQR method:\n", df[outliers_iqr.any(axis=1)])
numerical_columns = df.select_dtypes(include=[np.number]).columns
plt.figure(figsize=(15, 10))
df[numerical_columns].boxplot()
plt.xticks(rotation=90)
plt.title("Box plot of numerical features")
plt.show()
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Area', y='Price')
plt.title('Scatter plot of Price vs Area')
plt.show()
```

```
0.00632  18.00  2.310  0  0.5380  6.5750  65.20  4.0900  1  296.0
15.30  \
0  0.02731    0.0   7.07  0   0.469   6.421   78.9  4.9671  2  242.0
17.8
1  0.02729    0.0   7.07  0   0.469   7.185   61.1  4.9671  2  242.0
17.8
2  0.03237    0.0   2.18  0   0.458   6.998   45.8  6.0622  3  222.0
18.7
3  0.06905    0.0   2.18  0   0.458   7.147   54.2  6.0622  3  222.0
18.7
4  0.02985    0.0   2.18  0   0.458   6.430   58.7  6.0622  3  222.0
18.7

   396.90  4.98  24.00
0  396.90  9.14   21.6
1  392.83  4.03   34.7
```

```
2  394.63  2.94   33.4
3  396.90  5.33   36.2
4  394.12  5.21   28.7
          0.00632       18.00       2.310           0      0.5380
6.5750  \
count  505.000000  505.000000  505.000000  505.000000  505.000000
505.000000
mean     3.620667   11.350495   11.154257    0.069307    0.554728
6.284059
std      8.608572   23.343704    6.855868    0.254227    0.115990
0.703195
min      0.009060    0.000000    0.460000    0.000000    0.385000
3.561000
25%      0.082210    0.000000    5.190000    0.000000    0.449000
5.885000
50%      0.259150    0.000000    9.690000    0.000000    0.538000
6.208000
75%      3.678220   12.500000   18.100000    0.000000    0.624000
6.625000
max     88.976200  100.000000   27.740000    1.000000    0.871000
8.780000

            65.20      4.0900           1       296.0       15.30
396.90  \
count  505.000000  505.000000  505.000000  505.000000  505.000000
505.000000
mean    68.581584    3.794459    9.566337  408.459406   18.461782
356.594376
std     28.176371    2.107757    8.707553  168.629992    2.162520
91.367787
min      2.900000    1.129600    1.000000  187.000000   12.600000
0.320000
25%     45.000000    2.100000    4.000000  279.000000   17.400000
375.330000
50%     77.700000    3.199200    5.000000  330.000000   19.100000
391.430000
75%     94.100000    5.211900   24.000000  666.000000   20.200000
396.210000
max    100.000000   12.126500   24.000000  711.000000   22.000000
396.900000

             4.98       24.00
count  505.000000  505.000000
mean    12.668257   22.529901
std      7.139950    9.205991
min      1.730000    5.000000
25%      7.010000   17.000000
50%     11.380000   21.200000
75%     16.960000   25.000000
max     37.970000   50.000000
Outliers detected by Z-score method:
      0.00632  18.00  2.310  0  0.5380  6.5750  65.20  4.0900   1
296.0  \
54   0.01311   90.0   1.22  0   0.403   7.249   21.9  8.6966   5  226.0
55   0.02055   85.0   0.74  0   0.410   6.383   35.7  9.1876   2  313.0
56   0.01432  100.0   1.32  0   0.411   6.816   40.5  8.3248   5  256.0
101  0.22876    0.0   8.56  0   0.520   6.405   85.4  2.7147   5  384.0
140  1.62864    0.0  21.89  0   0.624   5.019  100.0  1.4394   4  437.0
..       ...    ...    ... ..     ...     ...    ...     ...  ..    ...
453  9.51363    0.0  18.10  0   0.713   6.728   94.1  2.4961  24  666.0
454  4.75237    0.0  18.10  0   0.713   6.525   86.5  2.4358  24  666.0
```

```
455  4.66883    0.0  18.10  0   0.713   5.976   87.9  2.5806  24  666.0
456  8.20058    0.0  18.10  0   0.713   5.936   80.3  2.7792  24  666.0
465  3.77498    0.0  18.10  0   0.655   5.952   84.7  2.8715  24  666.0

     15.30  396.90   4.98  24.00
54    17.9  395.93   4.81   35.4
55    17.3  396.90   5.77   24.7
56    15.1  392.90   3.95   31.6
101   20.9   70.80  10.63   18.6
140   21.2  396.90  34.41   14.4
..     ...     ...    ...    ...
453   20.2    6.68  18.71   14.9
454   20.2   50.92  18.13   14.1
455   20.2   10.48  19.01   12.7
456   20.2    3.50  16.94   13.5
465   20.2   22.01  17.15   19.0

[91 rows x 14 columns]
Outliers detected by IQR method:
       0.00632  18.00  2.310  0  0.5380  6.5750  65.20  4.0900   1
296.0  \
17    0.80271    0.0   8.14  0   0.538   5.456   36.6  3.7965   4
307.0
24    0.84054    0.0   8.14  0   0.538   5.599   85.7  4.4546   4
307.0
26    0.95577    0.0   8.14  0   0.538   6.047   88.8  4.4534   4
307.0
31    1.38799    0.0   8.14  0   0.538   5.950   82.0  3.9900   4
307.0
33    1.61282    0.0   8.14  0   0.538   6.096   96.9  3.7598   4
307.0
..        ...    ...    ... ..     ...     ...    ...     ...  ..
...
474   6.39312    0.0  18.10  0   0.584   6.162   97.4  2.2060  24
666.0
476  15.02340    0.0  18.10  0   0.614   5.304   97.3  2.1007  24
666.0
477  10.23300    0.0  18.10  0   0.614   6.185   96.7  2.1705  24
666.0
478  14.33370    0.0  18.10  0   0.614   6.229   88.0  1.9512  24
666.0
489   0.20746    0.0  27.74  0   0.609   5.093   98.0  1.8226   4
711.0

     15.30  396.90   4.98  24.00
17    21.0  288.99  11.69   20.2
24    21.0  303.42  16.51   13.9
26    21.0  306.38  17.28   14.8
31    21.0  232.60  27.71   13.2
33    21.0  248.31  20.34   13.5
..     ...     ...    ...    ...
474   20.2  302.76  24.10   13.3
476   20.2  349.48  24.91   12.0
477   20.2  379.70  18.03   14.6
478   20.2  383.32  13.11   21.4
489   20.1  318.43  29.68    8.1

[237 rows x 14 columns]
```

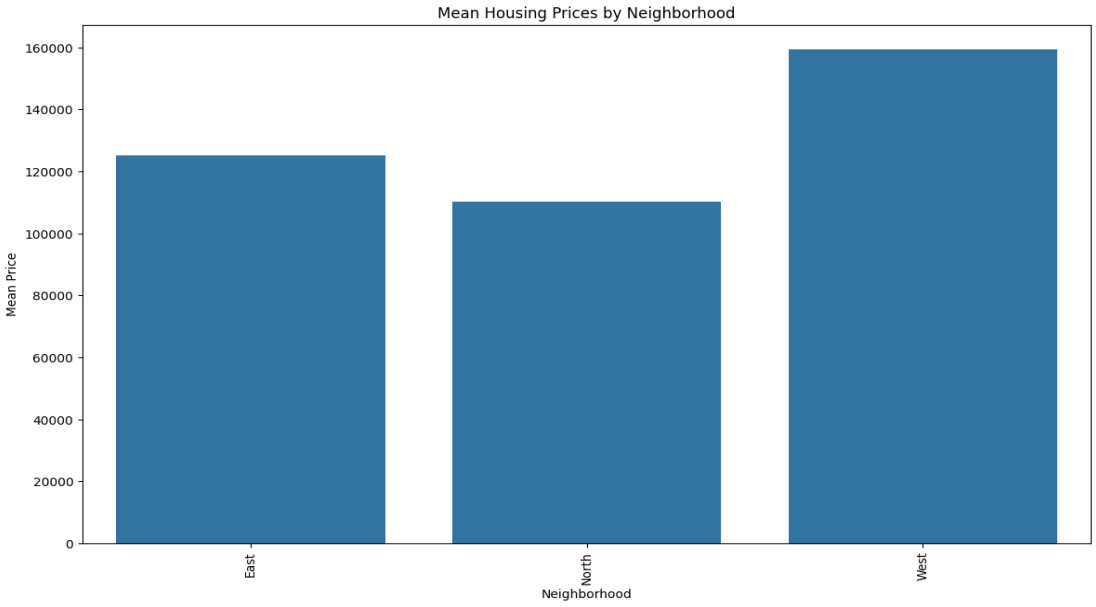

Box plot of numerical features
700
600
500
400
300
200
100
0
0.00632
18.00
2.310
0
0.5380
6.5750
65.20
4.0900
1
296.0
15.30
396.90
4.98
24.00


Scatter plot of X vs Y
700
600
500
400
300
200
100
0
0
100
200
300
400
500
0.00632
18.00
2.310
0
0.5380
6.5750
65.20
4.0900
1
296.0
15.30
396.90
4.98
24.00

Week 3: For a given dataset, display a chosen feature using different mean values

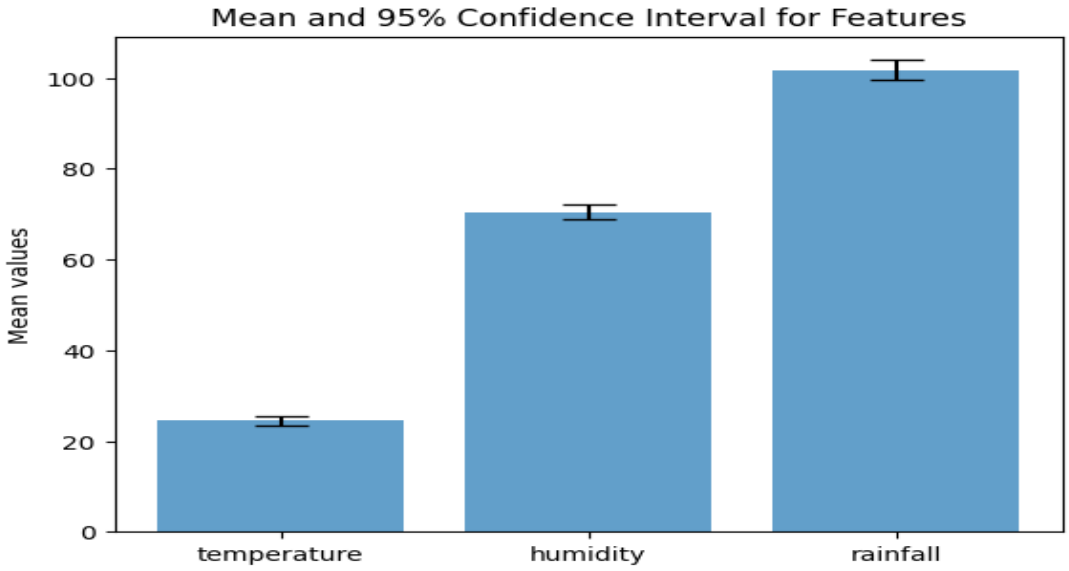
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('housing_data.csv')
print(df.head())
mean_prices_by_neighborhood = df.groupby('Neighborhood')['Price'].mean().reset_index()
print(mean_prices_by_neighborhood)
plt.figure(figsize=(14, 8))
sns.barplot(x='Neighborhood', y='Price', data=mean_prices_by_neighborhood)
plt.xticks(rotation=90)
plt.title('Mean Housing Prices by Neighborhood')
plt.xlabel('Neighborhood')
plt.ylabel('Mean Price')
plt.show()
```

```
Home   Price  SqFt  Bedrooms  Bathrooms  Offers Brick Neighborhood
0     1  114300  1790         2          2       2    No         East
1     2  114200  2030         4          2       3    No         East
2     3  114800  1740         3          2       1    No         East
3     4   94700  1980         3          2       3    No         East
4     5  119800  2130         3          3       3    No         East
  Neighborhood          Price
0         East  125231.111111
1        North  110154.545455
2         West  159294.871795
```



Week 4: Display the confidence interval of a chosen feature based on a sample

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt

data = {
   'temperature': [23, 25, 22, 24, 23, 24, 25, 26, 24, 23, 22, 24, 25, 26, 27, 28],
   'humidity': [65, 70, 68, 72, 69, 71, 70, 73, 67, 66, 68, 72, 71, 74, 75, 76],
   'rainfall': [100, 110, 95, 105, 98, 102, 99, 101, 104, 106, 97, 103, 108, 100, 99, 101]
}
df = pd.DataFrame(data)

def calculate_confidence_interval(feature, confidence_level=0.95):
   mean = np.mean(feature)
   sem = stats.sem(feature)
   ci = stats.t.interval(confidence_level, len(feature)-1, loc=mean, scale=sem)
   return mean, ci

features = ['temperature', 'humidity', 'rainfall']
confidence_level = 0.95

means = []
conf_intervals = []

for feature_name in features:
   feature = df[feature_name]
   mean, ci = calculate_confidence_interval(feature, confidence_level)
   means.append(mean)
   conf_intervals.append(ci)

fig, ax = plt.subplots()
x_pos = np.arange(len(features))
means = np.array(means)
conf_intervals = np.array(conf_intervals)

error = np.array([(mean - ci[0], ci[1] - mean) for mean, ci in zip(means, conf_intervals)]).T
ax.bar(x_pos, means, yerr=error, align='center', alpha=0.7, capsize=10)
ax.set_ylabel('Mean values')
ax.set_xticks(x_pos)
ax.set_xticklabels(features)
ax.set_title('Mean and 95% Confidence Interval for Features')
plt.show()
```


Mean and 95% Confidence Interval for Features
Mean values
0
20
40
60
80
100
temperature
humidity
rainfall

Week 5: Perform t-test on a feature in a dataset

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets
iris = datasets.load_iris()
sep_length = iris.data[:,0]
a_1, a_2 = train_test_split(sep_length, test_size=0.4, random_state=0)
b_1, b_2 = train_test_split(sep_length, test_size=0.4, random_state=1)
mu1 = np.mean(a_1)
mu2 = np.mean(b_1)
np.std(a_1)
np.std(b_1)
stats.ttest_ind(a_1, b_1, equal_var = False)
```

Output:

```
TtestResult(statistic=0.830066093774641, pvalue=0.4076270841218669,
df=175.8171155714046)
```

Week 6: Create Boxplots for different groups of a feature

```
import pandas as pd
import numpy as np
import scipy.stats as stats

# Load the house-prices dataset
# Assuming you have the dataset in a CSV file named 'house_prices.csv'
df = pd.read_csv('/content/house-prices.csv')

# Choose the feature for which you want to calculate the confidence interval
# For example, let's choose 'Price'
data = df['Price']

# Calculate the mean and standard error of the sample
mean = np.mean(data)
std_err = stats.sem(data)

# Define the confidence level
confidence = 0.95

# Calculate the margin of error
margin_of_error = std_err * stats.t.ppf((1 + confidence) / 2., len(data) - 1)

# Calculate the confidence interval
confidence_interval = (mean - margin_of_error, mean + margin_of_error)

print(f"Mean: {mean}")
print(f"Standard Error: {std_err}")
print(f"Confidence Interval: {confidence_interval}")
```

Output:

```
Mean: 130427.34375
Standard Error: 2374.8862164112793
Confidence Interval: (125727.87251232185, 135126.81498767814)
```

Week 7: Create a Linear Regression model for a dataset and display the error measures

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the dataset
url = "https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv"
df = pd.read_csv(url)
#df = pd.read_csv('/content/BostonHousing.csv')
# Display the first few rows of the dataset
print(df.head())

# Define the target variable and features
target = 'medv'
features = df.drop(columns=[target])

# Handle categorical variables using one-hot encoding
features = pd.get_dummies(features)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, df[target], test_size=0.2,
random_state=42)

# Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate error measures
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Display error measures
print(f"Mean Absolute Error (MAE): {mae}")
```

```
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

Output:

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax
ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
18.7

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2

Mean Absolute Error (MAE): 3.189091965887837
Mean Squared Error (MSE): 24.291119474973478
Root Mean Squared Error (RMSE): 4.928602182665332
R-squared (R2): 0.6687594935356326
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load the house-prices dataset
df = pd.read_csv('/content/house-prices.csv')

# Select features and target variable
# Assuming 'SalePrice' is the target variable and we use all other numerical features
X = df.select_dtypes(include=[np.number]).drop(columns=['Price'])
y = df['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Linear Regression model
```

```
model = LinearRegression()
model.fit(X_train, y_train)

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

# Calculate error measures
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

Output:

```
Mean Absolute Error (MAE): 13253.699656424242
Mean Squared Error (MSE): 254738265.975021
Root Mean Squared Error (RMSE): 15960.522108471921
```

Week 8: Chose a dataset with categorical data and apply linear regression model

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import OneHotEncoder

# Load the Titanic dataset
df =
pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv')

# Select features and target variable
# For simplicity, let's predict 'Fare' using other features including some categorical ones
# Drop columns that are not useful or have too many missing values
df = df.drop(columns=['Name', 'Ticket', 'Cabin', 'PassengerId'])

# Handle missing values
df = df.dropna()

# Separate features and target variable
X = df.drop(columns=['Fare'])
y = df['Fare']

# One-hot encode categorical variables
X = pd.get_dummies(X, drop_first=True)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

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

# Calculate error measures
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

Output:

```
Mean Absolute Error (MAE): 24.51059762079081
Mean Squared Error (MSE): 3621.956389227748
Root Mean Squared Error (RMSE): 60.182691774527235
```

Week 9: Apply Naive Bayes algorithm on a dataset and estimate the accuracy

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv('/content/house-prices.csv')
features = ['Home', 'Price', 'SqFt', 'Bedrooms', 'Bathrooms', 'Offers', 'Brick', 'Neighborhood']
X = df[features]
y = df['Price']
numerical_features = X.select_dtypes(include=[np.number]).columns
categorical_features = X.select_dtypes(include=[np.number, 'category']).columns
X[numerical_features] = X[numerical_features].fillna(X[numerical_features].median())
X[categorical_features] =
X[categorical_features].fillna(X[categorical_features].mode().iloc[0])
# Encode categorical variables
X = pd.get_dummies(X, drop_first=True)
# Convert the target variable into categorical classes
# For example, categorize into low, medium, and high price classes
y_class = pd.qcut(y, q=3, labels=['low', 'medium', 'high'])
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_class, test_size=0.2, random_state=42)
# Create and train the Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Output:

```
Accuracy: 0.8846153846153846
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
```

Output:

```
Accuracy: 1.0
```

Week 10: Apply Logistic Regression algorithm on a dataset and estimate the accuracy

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the house-prices dataset
df = pd.read_csv('/content/house-prices.csv')

# Select features and target variable
# For simplicity, let's use a few features and the target variable 'SalePrice'
features = ['Home', 'Price',  'SqFt',  'Bedrooms',  'Bathrooms', 'Offers'
]
X = df[features]
y = df['Price']

# Convert the target variable into categorical classes
# For example, categorize into low, medium, and high price classes
y_class = pd.qcut(y, q=3, labels=['low', 'medium', 'high'])

# Handle missing values using .loc to avoid the SettingWithCopyWarning
X.loc[:, 'Home'] = X['Home'].fillna(X['Home'].median())
X.loc[:, 'Price'] = X['Price'].fillna(X['Price'].median())
X.loc[:, 'SqFt'] = X['SqFt'].fillna(X['SqFt'].median())
X.loc[:, 'Bedrooms'] = X['Bedrooms'].fillna(X['Bedrooms'].median())
X.loc[:, 'Bathrooms'] = X['Bathrooms'].fillna(X['Bathrooms'].median())
```

```
X.loc[:, 'Offers'] = X['Offers'].fillna(X['Offers'].median())


# Encode categorical variables if there are any (not in this case)
# X = pd.get_dummies(X, drop_first=True)


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_class, test_size=0.2, random_state=42)
# Create and train the Logistic Regression model
model = LogisticRegression(max_iter=400)
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Output:

Accuracy: 0.5

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the Titanic dataset
df =
pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv')

# Select features and target variable
# For simplicity, let's predict 'Survived' using a few features
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X = df[features]
y = df['Survived']

# Handle missing values
X['Age'] = X['Age'].fillna(X['Age'].median())
X['Embarked'] = X['Embarked'].fillna(X['Embarked'].mode()[0])

# Encode categorical variables
X = pd.get_dummies(X, drop_first=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Logistic Regression model
model = LogisticRegression(max_iter=400)
model.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
```

Output:

Accuracy: 0.8100558659217877