experiment1

```
# import necessary libraries
import pandas as pd                    # pandas: for creating and manipulating the data
import seaborn as sns                  # seaborn: for making the plot
import matplotlib.pyplot as plt   # matplotlib: for additional plot customization
import numpy as np                        # numpy: for numerical operations (e.g., random
number generation)

# create a dataframe with the initial data
data = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie'],           # List of student names
'Math': [85, 90, 95],                          # Math scores for each student
'Science': [88, 92, 98]                        # Science scores for each student
})

# Add a new subject, "History", with randomly generated scores
data['History'] = np.random.randint(80, 100, len(data))                    # Generates random
history scores between 80 and 100

# Prepare the data for plotting: we need a tidy format (long format)
combined = pd.DataFrame({
'Name': list(data['Name']) *
3,                                                          # Duplicate the names for
the 3 subjects
'Subject': ['Math', 'Science', 'History'] *
len(data),                                          # List of subjects repeated for each
individual
'Score': list(data['Math']) + list(data['Science']) + list(data['History'])                         #
Combine the scores into one column
})

# Plot the data using seaborn's barplot function
sns.barplot(data=combined, x='Name', y='Score', hue='Subject')                              #
Create a bar plot, hue differentiates the subjects
plt.title('All
Scores')                                                                                  # Add
a title to the plot
plt.tight_layout()
        # Adjust layout to make sure everything fits properly
```

```
plt.show()
    # Display the plot
```

in r

```
# Load the necessary libraries
library(dplyr)                          # dplyr: for data manipulation (filtering, reshaping,
etc.)
library(ggplot2)                          # ggplot2: for creating plots

# Create a data frame with the initial data
data <- tibble(
Name = c("Alice", "Bob", "Charlie"),                    # List of student names
Math = c(85, 90, 95),                                   # Math scores for each student
Science = c(88, 92, 98)                                    # Science scores for each student
)

# Add a new subject, "History", with randomly generated scores
set.seed(0)                                                                    # Set a
random seed for reproducibility
data$History <- sample(80:100, nrow(data), replace = TRUE)          # Generate random
history scores between 80 and 100

# Combine the data into a long format (tidy format) for plotting
combined <- tibble(
Name = rep(data$Name, 3),                                                        #
Duplicate names for each subject (Math, Science, History)
Subject = rep(c("Math", "Science", "History"), each = nrow(data)),            # Repeat the
subject names for each individual
Score = c(data$Math, data$Science, data$History)                          # Combine the
scores for each subject into one column
)

# Plot the data using ggplot2
ggplot(combined, aes(x = Name, y = Score, fill = Subject)) +                               #
Create a bar plot with Name on x-axis, Score on y-axis, and bars filled by Subject
geom_col(position = "dodge")
+                                                        # Create grouped bars (dodge
makes them side by side)
ggtitle("All Scores") # Add a title to the plot
```

experiment 2

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt

# Load the CSV data (ensure 'data.csv' is in your working directory)
data = pd.read_csv('data.csv')

# Define the independent (Experience) and dependent (Salary) variables
X = data[['Experience']] # Independent variable (2D array)
y = data['Salary'] # Dependent variable

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)

# Plot actual vs predicted data
plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, y_pred, color='red', label='Predicted Line')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()

# Calculate and print R² and RMSE
r2 = r2_score(y, y_pred)
rmse = mean_squared_error(y, y_pred, squared=False)
print("R²:", r2)
```

```
print("RMSE:", rmse)

# Calculate residuals (actual - predicted)
residuals = y - y_pred

# Plot the residuals
plt.scatter(X, residuals, color='purple')
plt.axhline(y=0, color='black', linestyle='--')
plt.title('Residual Plot')
plt.xlabel('Experience')
plt.ylabel('Residuals')
plt.show()
```

in r

```
# Install required package (if not already installed)
# install.packages("ggplot2")

# Load necessary library for plotting
library(ggplot2)

# Load the CSV data (ensure 'data.csv' is in your working directory)
data <- read.csv("data.csv")

# Define the independent variable (X) and dependent variable (y)
X <- data$Experience # Independent variable
y <- data$Salary # Dependent variable

# Fit the linear regression model
model <- lm(Salary ~ Experience, data = data)

# Display the model summary (coefficients, R-squared, etc.)
summary(model)

# Make predictions using the model
y_pred <- predict(model, newdata = data)

# Plot actual vs predicted values
ggplot(data, aes(x = Experience, y = Salary)) +
```

```
geom_point(color = "blue") + # Actual data points (scatter plot)
geom_smooth(method = "lm", color = "red") + # Predicted line (regression line)
labs(title = "Simple Linear Regression", # Add title
x = "Experience", y = "Salary") # Label axes

# Calculate R-squared and RMSE
r2 <- summary(model)$r.squared
rmse <- sqrt(mean((y - y_pred)^2)) # Calculate Root Mean Squared Error
cat("R²:", r2, "\n")
cat("RMSE:", rmse, "\n")

# Calculate residuals (actual - predicted)
residuals <- y - y_pred

# Plot the residuals
ggplot(data, aes(x = Experience, y = residuals)) +
geom_point(color = "purple") +
geom_hline(yintercept = 0, color = "black", linetype = "dashed") +
labs(title = "Residual Plot", x = "Experience", y = "Residuals")
```

experiment 3 multiple regression

```
import pandas as pd # Import pandas for data manipulation
import statsmodels.api as sm # Import statsmodels for regression analysis
import seaborn as sns # Import seaborn for data visualization
import matplotlib.pyplot as plt # Import matplotlib for plotting

# Load data from CSV
data = pd.read_csv('employee_data.csv') # Read the employee data from
'employee_data.csv' file

# Convert categorical Education column to dummy variables
data = pd.get_dummies(data, columns=['Education'], drop_first=True)
# Create dummy variables for the 'Education' column (e.g., converts 'Education' to
'Education_Master' and 'Education_PhD')

# Features and target
X = data.drop(columns='Salary') # Define features (independent variables) by dropping the
target column 'Salary'
```

```
y = data['Salary'] # Define the target variable (dependent variable)

# Add constant for intercept
X_const = sm.add_constant(X) # Add a constant column to the feature matrix (intercept
term)

# Fit regression model
model = sm.OLS(y, X_const).fit() # Fit an OLS (Ordinary Least Squares) regression model
using statsmodels

print(model.summary()) # Print the summary of the regression model, including coefficients,
R-squared, etc.

# Predict for new employee: 6 years experience, PhD
new_data = pd.DataFrame({
'Experience': [6], # New employee has 6 years of experience
'Education_Master': [0], # The employee doesn't have a Master's degree (i.e., it's a PhD)
'Education_PhD': [1] # The employee has a PhD (this is the reference category after dummy
encoding)
})

new_data = sm.add_constant(new_data) # Add constant for the new data (intercept term)
prediction = model.predict(new_data) # Predict the salary for the new employee based on
the fitted model
print("Predicted Salary:", prediction.iloc[0]) # Print the predicted salary for the new
employee

# Plot actual vs predicted
sns.scatterplot(x=model.fittedvalues, y=y) # Create a scatter plot of actual vs predicted salary
values
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--') # Plot a red dashed line to represent
perfect predictions
plt.xlabel("Predicted Salary") # Label for the x-axis
plt.ylabel("Actual Salary") # Label for the y-axis
plt.title("Predicted vs Actual Salary") # Title of the plot
plt.show() # Show the plot
```

in r

```r
# Load necessary libraries
library(readr) # For reading CSV files
library(dplyr) # For data manipulation (if needed)
library(stats) # For fitting linear models (lm function)
library(ggplot2) # For plotting

# Load the data from CSV
data <- read_csv("employee_data.csv") # Assuming the CSV is in the current working
directory

# Convert categorical Education column to factors (automatically handled by lm function)
data$Education <- factor(data$Education)

# Create the multiple regression model
# Education is converted to dummy variables automatically by the 'lm' function
model <- lm(Salary ~ Experience + Education, data = data)

# Print summary of the regression model
summary(model)

# Predict for a new employee: 6 years experience, PhD
new_data <- data.frame(
Experience = 6,
Education = factor("PhD", levels = levels(data$Education)) # Assuming 'PhD' is one of the
Education categories
)
predicted_salary <- predict(model, new_data) # Predict using the fitted model
print(paste("Predicted Salary for new employee:", predicted_salary))

# Plot actual vs predicted salary values
ggplot(data, aes(x = fitted(model), y = Salary)) +
geom_point() + # Actual vs predicted values
geom_abline(slope = 1, intercept = 0, color = "red") + # Red line for perfect predictions
labs(x = "Predicted Salary", y = "Actual Salary", title = "Predicted vs Actual Salary") +
theme_minimal()
```

experiment 4 time series

```
# Load necessary libraries
library(lubridate) # For date manipulation
library(forecast) # For time series forecasting and analysis
library(dplyr) # For data manipulation (e.g., drop_na, arrange)

# Load the data
data <- read.csv("your_file.csv") # Replace "your_file.csv" with your actual file path

# Convert the 'Date' column to Date type
data$Date <- as.Date(data$Date, format="%Y-%m-%d")

# Remove rows with missing values in the 'close' column
data <- data %>% drop_na(close)

# Sort data by Date
data <- data %>% arrange(Date)

# Create a time series object
ts_data <- ts(data$close, start=c(year(min(data$Date)), month(min(data$Date))),
frequency=252)

# Plot the time series
plot(ts_data, main="TCS Stock Closing Price Time Series", ylab="Close Price", xlab="Time")
```

experiment 5 ARIMA MODEL

```
# load required libraries
library(forecast)
library(ggplot2)

# read the csv data (adjust path as needed)
solar_prod_input <- read.csv("c:/users/lab204/downloads/solar_prod.csv")

# convert to time series object (assuming monthly data)
solar_prod <- ts(solar_prod_input[, 2], start = c(1), frequency = 12) # adjust 'start' if needed

# plot original time series
plot(solar_prod, xlab = "time (months)", ylab = "solar production (kwh)", main = "solar
production time series")
```

```
# fit arima model (auto.arima can also be used)
arima_model <- arima(solar_prod, order = c(0, 1, 0), seasonal = list(order = c(1, 0, 0), period
= 12))

# print model summary
summary(arima_model)

# forecast next 12 periods
arima_forecast <- forecast(arima_model, h = 12)

# plot forecast with confidence intervals
plot(arima_forecast, main = "forecast of solar production")

# optionally extract forecast values and confidence intervals
pred_matrix <- cbind(
LB = arima_forecast$lower[, 2],
Pred = arima_forecast$mean,
UB = arima_forecast$upper[, 2]
)

# show prediction matrix
print(round(pred_matrix, 2))
```

experiment6 -email spam classifier code

```
import pandas as pd
import string
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Download stopwords
```

```
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Load dataset (replace with your actual dataset URL or file path)
url = 'https://raw.githubusercontent.com/justmarkham/pycon-2016-
tutorial/master/data/sms.tsv'
df = pd.read_csv(url, sep='\t', header=None, names=['label', 'message'])

# Preprocess text
def clean_text(msg):
msg = msg.lower()
msg = ''.join([char for char in msg if char not in string.punctuation])
msg = ' '.join([word for word in msg.split() if word not in stop_words])
return msg

df['cleaned'] = df['message'].apply(clean_text)

# Vectorization
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['cleaned'])

# Label encoding
y = df['label'].map({'ham': 0, 'spam': 1})

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = MultinomialNB()
model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Function to predict if a message is spam or not
def predict_spam(message):
# Clean the input message
cleaned_message = clean_text(message)
```

```
# Convert the message into a feature vector
message_vector = vectorizer.transform([cleaned_message])

# Predict the category (0 = not spam, 1 = spam)
prediction = model.predict(message_vector)

# Return the result
if prediction == 1:
return "This message is SPAM"
else:
return "This message is NOT SPAM"

# Test the function with an example
test_message = input("Enter the message to classify (spam or not): ")
result = predict_spam(test_message)
print(result)
```

SENTIMENT ANALYSIS

```
import nltk
from nltk.corpus import movie_reviews
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy

# Download necessary datasets
nltk.download('movie_reviews')
nltk.download('punkt')

# Function to extract features from a document (review)
def document_features(doc):
words = set(doc)
features = {}
for word in all_words:
features[word] = (word in words)
return features

# Load the movie reviews dataset
positive_reviews = movie_reviews.categories('pos')
negative_reviews = movie_reviews.categories('neg')
```

```
# Create a list of (document, category) pairs for training data
documents = []
for category in positive_reviews:
for fileid in movie_reviews.fileids(category):
documents.append((movie_reviews.words(fileid), 'pos'))
for category in negative_reviews:
for fileid in movie_reviews.fileids(category):
documents.append((movie_reviews.words(fileid), 'neg'))

# Shufle the documents to randomize the dataset
import random
random.shufle(documents)

# Get all words in the corpus for feature extraction
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())

# Select the most common 2000 words as features
word_features = list(all_words.keys())[:2000]

# Extract features for each document
featuresets = [(document_features(doc), category) for (doc, category) in documents]

# Split the data into training and testing sets (80% training, 20% testing)
train_set, test_set = featuresets[:int(len(featuresets) * 0.8)], featuresets[int(len(featuresets)
* 0.8):]

# Train a Naive Bayes classifier
classifier = NaiveBayesClassifier.train(train_set)

# Evaluate the classifier on the test set
accuracy_result = accuracy(classifier, test_set)
print(f"Accuracy: {accuracy_result * 100:.2f}%")

# Print the most informative features
classifier.show_most_informative_features(10)

# Function to classify a new review
def classify_review(review):
words_in_review = nltk.word_tokenize(review)
features = document_features(words_in_review)
```

```
return classifier.classify(features)

# Test the sentiment analysis with a custom review
test_review = input("Enter a movie review for sentiment analysis: ")
result = classify_review(test_review)
print(f"The sentiment of the review is: {result}")
```

EXPERIMENT 7

DIFFRENT VISUALTION IN R

```
# Install ggplot2 if not already installed
# install.packages("ggplot2")
library(ggplot2)

# 1. Line Plot
x <- seq(0, 10, by=0.1)
y <- sin(x)
plot(x, y, type="l", col="blue", main="Line Plot of Sine Wave", xlab="X-axis", ylab="Y-axis")
grid()

# 2. Bar Chart
categories <- c('A', 'B', 'C', 'D')
values <- c(5, 7, 3, 9)
barplot(values, names.arg=categories, col='green', main="Bar Chart Example",
xlab="Categories", ylab="Values")

# 3. Histogram
data <- rnorm(1000)
hist(data, breaks=30, col='purple', border='black', main="Histogram Example", xlab="Data
Values", ylab="Frequency")

# 4. Scatter Plot
x_scatter <- runif(50)
y_scatter <- runif(50)
plot(x_scatter, y_scatter, col="red", main="Scatter Plot Example", xlab="X-axis", ylab="Y-
```

```
axis")

# 5. Pie Chart
labels <- c('Python', 'C++', 'Java', 'JavaScript')
sizes <- c(40, 30, 20, 10)
pie(sizes, labels=labels, col=c('blue', 'green', 'orange', 'red'), main="Pie Chart Example")
```

EXPERIMENT 8 DIFFRENT VISUALIZATION IN PYTHON

```
import matplotlib.pyplot as plt
import numpy as np

# 1. Line Plot
x = np.arange(0, 10, 0.1)
y = np.sin(x)
plt.plot(x, y, label='Sine Wave', color='blue')
plt.title('Line Plot of Sine Wave')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)
plt.show()

# 2. Bar Chart
categories = ['A', 'B', 'C', 'D']
values = [5, 7, 3, 9]
plt.bar(categories, values, color='green')
plt.title('Bar Chart Example')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()

# 3. Histogram
data = np.random.randn(1000)
plt.hist(data, bins=30, color='purple', edgecolor='black')
plt.title('Histogram Example')
plt.xlabel('Data Values')
```

```
plt.ylabel('Frequency')
plt.show()

# 4. Scatter Plot
x_scatter = np.random.rand(50)
y_scatter = np.random.rand(50)
plt.scatter(x_scatter, y_scatter, color='red')
plt.title('Scatter Plot Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

# 5. Pie Chart
labels = ['Python', 'C++', 'Java', 'JavaScript']
sizes = [40, 30, 20, 10]
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=['blue', 'green',
'orange', 'red'])
plt.title('Pie Chart Example')
plt.show()

### EXP 8
import matplotlib.pyplot as plt

x=[10,20,30,40]
y=[20,25,35,55]
plt.plot(x,y)
plt.title("Line Chart")
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
plt.show()

import matplotlib.pyplot as plt
import pandas as pd

data =
pd.read_csv('/content/tips.csv')
x=data['day']
y=data['total_bill']
plt.bar(x,y)
plt.title("Tips Dataset")
plt.ylabel('Total Bill')
plt.xlabel('Day')
plt.show()
```

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

data =
pd.read_csv('/content/tips.csv')
x=data['day']
y=data['total_bill']
plt.scatter(x,y)
plt.title("Tips Dataset")
plt.ylabel('Total Bill')
plt.xlabel('Day')
plt.show()

import matplotlib.pyplot as plt
import pandas as pd

data =
pd.read_csv('/content/tips.csv')
cars=['AUDI','BMW','FORD','TESLA'
,'JAGUAR']
data = [23,10,35,15,12]
plt.pie(data,labels=cars)
plt.title("Cars Dataset")
plt.show()

import matplotlib.pyplot as plt
import pandas as pd

np.random.seed(10)
data
=[np.random.normal(0,std,100)
for std in range(1,4)]
plt.boxplot(data,vert=True,patch_
artist=True,

boxprops=dict(facecolor='blue'),

medianprops=dict(color='red'))
plt.xlabel('Data Set')
plt.ylabel('Value')
plt.title('Box Plot')
plt.show()

# 3. Histogram
```

```
data = np.random.randn(1000)
plt.hist(data, bins=30, color='purple', edgecolor='black')
plt.title('Histogram Example')
plt.xlabel('Data Values'
plt.ylabel('Frequency')
plt.show()
```