DAV Practical Codes

Experiment no: 2 - Simple Linear Regression

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

X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
Y = np.array([2, 4, 5, 4, 5, 7, 8, 8, 9, 10])
n = len(X)

sum_x = np.sum(X)
sum_y = np.sum(Y)
sum_xx = np.sum(X**2)
sum_xy = np.sum(X*Y)

m = (n*sum_xy - sum_x*sum_y) / (n*sum_xx - sum_x**2)
b = (sum_y - m*sum_x) / n

print(f"Slope(m): {m}")
print(f"Intercept(b): {b}")

Y_pred = m*X + b

plt.scatter(X, Y, color='blue', label='Data points')
plt.plot(X, Y_pred, color='red', label='Fitted line')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Simple Linear Regression')
plt.show()
```

Experiment no: 3 - Multiple Linear Regression

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

# Sample data
X1 = np.array([1, 2, 4, 3, 5])  # Feature 1: Hours studied
X2 = np.array([1, 3, 3, 2, 5])  # Feature 2: Practice tests taken
Y  = np.array([2, 4, 8, 6, 10]) # Target: Final exam score

# Add intercept column to X
X = np.column_stack((np.ones(len(X1)), X1, X2))

# Normal Equation: B = (X^T X)^(-1) X^T Y
B = np.linalg.inv(X.T @ X) @ X.T @ Y
b0, b1, b2 = B

print(f"Intercept (b0): {b0}")
print(f"Coefficient for Hours Studied (b1): {b1}")
print(f"Coefficient for Practice Tests (b2): {b2}")

# Predicted Y values
Y_pred = X @ B

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Scatter actual data points
ax.scatter(X1, X2, Y, color='blue', label='Actual data')

# Create grid to plot regression plane
x1_grid, x2_grid = np.meshgrid(
    np.linspace(min(X1), max(X1), 10),
    np.linspace(min(X2), max(X2), 10)
)
y_grid = b0 + b1 * x1_grid + b2 * x2_grid
```

```
# Plot regression plane
ax.plot_surface(x1_grid, x2_grid, y_grid, color='red', alpha=0.5)

# Labels and title
ax.set_xlabel('Hours Studied')
ax.set_ylabel('Practice Tests Taken')
ax.set_zlabel('Final Exam Score')
ax.set_title('Multiple Linear Regression')

plt.show()
```

Experiment no: 4 - Time series Analysis

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Step 1: Create a sample time series (monthly data)
dates = pd.date_range(start='2020-01-01', periods=36, freq='M')
np.random.seed(0)
data = 10 + 0.5 * np.arange(36) + 3 * np.sin(2 * np.pi * dates.month / 12) + np.ra
ts = pd.Series(data, index=dates)

# Step 2: Plot the time series
plt.figure(figsize=(10, 4))
plt.plot(ts, label='Original Time Series')
plt.title('Time Series Data')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```

```
# Step 3: Decompose the series
decomposition = seasonal_decompose(ts, model='additive', period=12)

# Plot components: Trend, Seasonal, Residual
decomposition.plot()
plt.suptitle("Decomposition of Time Series", fontsize=14)
plt.tight_layout()
plt.show()

# Step 4: Forecast using Simple Moving Average
rolling_avg = ts.rolling(window=3).mean()

# Plot original and forecast
plt.figure(figsize=(10, 4))
plt.plot(ts, label='Original')
plt.plot(rolling_avg, label='3-Month Moving Average', color='red')
plt.title('Simple Forecasting using Moving Average')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```

Experiment no: 5 - ARIMA model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA



np.random.seed(1)
dates = pd.date_range(start='2015-01-01', periods=100, freq='M')
data = pd.Series(50 + 0.8*np.arange(100) + np.random.normal(scale=5, size=10
```

```
plt.figure(figsize=(10, 4))
plt.plot(data, label='Original Time Series')
plt.title('Monthly Time Series')
plt.xlabel('Date')
plt.ylabel('Value')
plt.grid(True)
plt.legend()
plt.show()


model = ARIMA(data, order=(1, 1, 1))
model_fit = model.fit()

# Print summary of the model
print(model_fit.summary())

# Step 4: Forecast the next 12 steps
forecast = model_fit.forecast(steps=12)

# Step 5: Plot original + forecast
plt.figure(figsize=(10, 4))
plt.plot(data, label='Historical Data')
plt.plot(forecast.index, forecast, label='Forecast', color='red')
plt.title('ARIMA Forecast')
plt.xlabel('Date')
plt.ylabel('Value')
plt.grid(True)
plt.legend()
plt.show()
```

Experiment no: 6 - Sentiment analysis

```
import pandas as pd
import nltk
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download('all')

df = pd.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/dat

def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    filtered_tokens = [token for token in tokens if token not in stopwords.words('er
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens
    processed_text = ' '.join(lemmatized_tokens)
    return processed_text

df['reviewText'] = df['reviewText'].apply(preprocess_text)

analyzer = SentimentIntensityAnalyzer()

def get_sentiment(text):
    scores = analyzer.polarity_scores(text)
    sentiment = 1 if scores['pos'] > 0 else 0
    return sentiment

df['sentiment'] = df['reviewText'].apply(get_sentiment)

from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(df['Positive'], df['sentiment']))
print(classification_report(df['Positive'], df['sentiment']))
```

Experiment no: 7 -

library(ggcorrplot)
library(ggplot2)

```
library(dplyr)
library(tidyr)
library(summarytools)

setwd("/home/a/aman/")
data ← read.csv("iris.csv")

head(data)
sum(
is.na(data))
data ← na.omit(data)
data$sepal.length[
is.na(data$sepal.length)] ← mean(data$sepal.length, na.rm=TRUE)

summary(data)

cor_matrix ← cor(data[, 1:4])
ggcorrplot(cor_matrix, lab=TRUE, title= "Correlation Matrix")

ggplot(data, aes(x = sepal.length)) +
geom_histogram(binwidth = 0.1, fill = "skyblue", color = "black") +
labs(title = "Histogram of Sepal Length", x = "Sepal Length", y = "Frequency")
```

Experiment no: 8

```
library(ggplot2)
library(corrplot)
library(GGally)

data(iris)

ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
geom_point() +
labs(title = "Scatter Plot of Sepal Length vs Petal Length") +
theme_minimal()

ggpairs(iris, aes(color = Species))

cor_matrix ← cor(iris[, 1:4])
print(cor_matrix)

corrplot(cor_matrix, method = "circle", type = "upper", tl.cex = 0.8, tl.col =
"black")
```

```
pearson_corr <- cor(iris$Sepal.Length, iris$Petal.Length)
print(paste("Pearson Correlation between Sepal.Length and Petal.Length:",
pearson_corr))
```

Experiment no: 9

```python
import pandas as pd
import numpy as np

# Sample dataset
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'Salary': [50000, 60000, 70000, 80000]
}

# Create DataFrame
df = pd.DataFrame(data)

# Manipulate data
df['Bonus'] = df['Salary'] * 0.1
df['Salary'] += 5000  # Increment salary

# Summary
print("Data Preview:\n", df.head())
print("\nDescriptive Stats:\n", df.describe())

# NumPy operations
print("\nMean Age:", np.mean(df['Age']))
print("Std Salary:", np.std(df['Salary']))
print("Median Bonus:", np.median(df['Bonus']))
```

Experiment no: 10

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

```
data = {
'Name': ['Alice', 'Bob', 'Charlie', 'David'],
'Age': [25, 30, 35, 40],
'Salary': [55000, 65000, 75000, 85000],
'Department': ['HR', 'IT', 'IT', 'HR']
}
df = pd.DataFrame(data)
```

Histogram of Ages

```
df['Age'].plot(kind='hist', title='Age Distribution')
plt.xlabel('Age')
plt.show()
```

Bar chart of Salaries

```
df.plot(x='Name', y='Salary', kind='bar', title='Salary by Employee')
plt.ylabel('Salary')
plt.show()
```

Pie chart of Department counts

```
df['Department'].value_counts().plot(kind='pie', autopct='%1.1f%%',
title='Department Distribution')
plt.ylabel('')
plt.show()
```

Box plot of Salaries

```
df['Salary'].plot(kind='box', title='Salary Spread')
plt.ylabel('Salary')
plt.show()
```