COGNITIVE ANALYTICS LAB

Submitted by:
Wajid Ali Hashmi
500096923
B2(H)

Submitted to:
Sugandha Sharma

Index

Experiment 1

Analysing the dataset based on some values

Dataset source: http://blog.hackerearth.com/wp-content/uploads/2016/12/airfoil_self_noise.csv

```python
import pandas as pd
import numpy as  np
import statsmodels.api as sm
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

[17] Python

```python
df = pd.read_csv("D:\DESKTOP\cogntive analysis\self noise.csv")
df.head()
```

[18] Python

	Frquency(Hz)	Angle_of_Attack	Chord_Length	Free_stream_velocity	Displacement	Sound_pressure_level
0	800	0.0	0.3048	71.3	0.002663	126.201
1	1000	0.0	0.3048	71.3	0.002663	125.201
2	1250	0.0	0.3048	71.3	0.002663	125.951
3	1600	0.0	0.3048	71.3	0.002663	127.591
4	2000	0.0	0.3048	71.3	0.002663	127.461

```python
correlation_matrix=df.corr()
print(correlation_matrix)
```

[19] Python

```
                      Frquency(Hz)  Angle_of_Attack  Chord_Length  \
Frquency(Hz)              1.000000        -0.272765     -0.003661
Angle_of_Attack          -0.272765         1.000000     -0.504868
Chord_Length             -0.003661        -0.504868      1.000000
Free_stream_velocity      0.133664         0.058760      0.003787
Displacement             -0.230107         0.753394     -0.220842
Sound_pressure_level     -0.390711        -0.156108     -0.236162

                      Free_stream_velocity  Displacement  Sound_pressure_level
Frquency(Hz)                      0.133664     -0.230107             -0.390711
Angle_of_Attack                   0.058760      0.753394             -0.156108
Chord_Length                      0.003787     -0.220842             -0.236162
Free_stream_velocity              1.000000     -0.003974              0.125103
Displacement                     -0.003974      1.000000             -0.312670
Sound_pressure_level              0.125103     -0.312670              1.000000
```

```python
X = df.drop(columns=['Sound_pressure_level'])
X=sm.add_constant(X)
y=df['Sound_pressure_level']
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.2, random_state=42)
Linear_Regression=sm.OLS(y_train, X_train).fit()
```

[20] Python

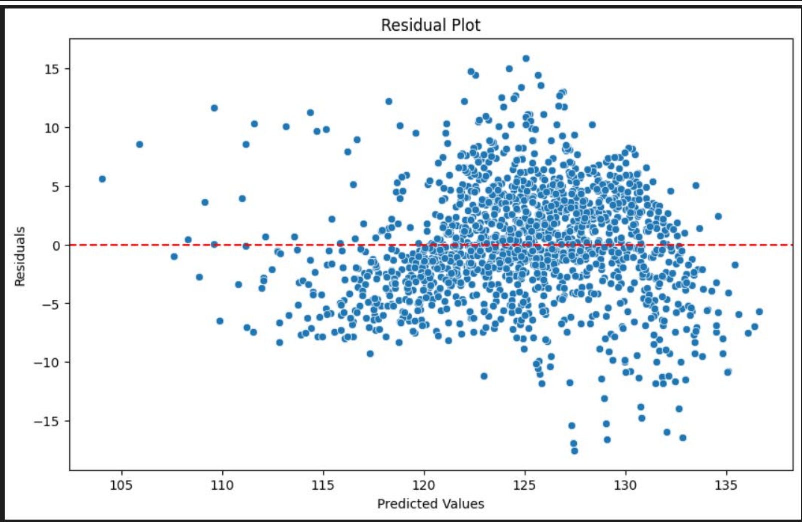
```python
y_pred = Linear_Regression.predict(X_test)
r_squared = r2_score(y_test, y_pred)
adj_r_squared = Linear_Regression.rsquared_adj
f_statistic = Linear_Regression.fvalue
p_value = Linear_Regression.f_pvalue
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adj_r_squared}")
print(f"F-statistics: {f_statistic}, p-value: {p_value}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
```

[15] Python

```
R-squared: 0.5582979754896911
Adjusted R-squared: 0.5013716488971148
F-statistics: 242.52150554359076, p-value: 6.080414968967961e-179
MSE: 22.128643318249154
RMSE: 4.704109194975087
MAE: 3.672414564174714
```

```python
df_with_const = sm.add_constant(df[expected_features])
df['Predicted'] = Linear_Regression.predict(df_with_const)
df['Residuals'] = df['Sound_pressure_level'] - df['Predicted']
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Predicted', y='Residuals', data=df)
plt.title('Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```

[27] Python



Interpretations:

Based on the analysis, the following can be deduced about the dataset:

- The R-squared value of 0.558 indicates that the linear regression model explains 55.8% of the variance in the sound pressure level. This is a good fit for a linear model.
- The adjusted R-squared value of 0.501 is slightly lower than the R-squared value, but it still indicates a good fit for the model.
- The F-statistic of 242.52 and the p-value of less than 0.0001 both indicate that the model is statistically significant. This means that the coefficients of the model are not equal to zero.
- The MSE of 22.12indicates that the average squared difference between the predicted and actual sound pressure levels is 22.12.
- The RMSE of 4.704 is the square root of the MSE, and it indicates that the average difference between the predicted and actual sound pressure levels is 4.704.
- The MAE of 3.672 indicates that the average absolute difference between the predicted and actual sound pressure levels is 3.672.
- The residual plot shows that the residuals are randomly scattered around the zero line. This indicates that the model does not have any heteroscedasticity or autocorrelation problems.

- The coefficients of the model can be used to identify which features have the most impact on the sound pressure level.
- The model can be used to make predictions about the sound pressure level for new data points.
- The model can be used to identify outliers in the data.

In conclusion, the linear regression model is a good fit for the data and can be used to predict the sound pressure level based on the other features in the dataset.

Experiment 2

Implementation of Ridge and lasso algorithm in Python:

Dataset source: https://github.com/JWarmenhoven/ISLR-python/blob/master/Notebooks/Data/Hitters.csv

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.metrics import mean_squared_error
```

[1] ✓ 18.1s Python

```python
df = pd.read_csv('Hitters.csv').dropna().drop('Player', axis = 1)
df.info()
dummies = pd.get_dummies(df[['League', 'Division', 'NewLeague']])
```

[2] ✓ 0.4s Python

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 263 entries, 1 to 321
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   AtBat      263 non-null    int64
 1   Hits       263 non-null    int64
 2   HmRun      263 non-null    int64
 3   Runs       263 non-null    int64
 4   RBI        263 non-null    int64
 5   Walks      263 non-null    int64
 6   Years      263 non-null    int64
 7   CAtBat     263 non-null    int64
 8   CHits      263 non-null    int64
 9   CHmRun     263 non-null    int64
 10  CRuns      263 non-null    int64
 11  CRBI       263 non-null    int64
 12  CWalks     263 non-null    int64
 13  League     263 non-null    object
 14  Division   263 non-null    object
 15  PutOuts    263 non-null    int64
 16  Assists    263 non-null    int64
 17  Errors     263 non-null    int64
 18  Salary     263 non-null    float64
 19  NewLeague  263 non-null    object
dtypes: float64(1), int64(16), object(3)
memory usage: 43.1+ KB
```

```python
y = df.Salary
X_ = df.drop(['Salary', 'League', 'Division', 'NewLeague'], axis = 1).astype('float64')
# Define the feature set X.
X = pd.concat([X_, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis = 1)
X.info()
```

[3] ✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 263 entries, 1 to 321
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   AtBat        263 non-null    float64
 1   Hits         263 non-null    float64
 2   HmRun        263 non-null    float64
 3   Runs         263 non-null    float64
 4   RBI          263 non-null    float64
 5   Walks        263 non-null    float64
 6   Years        263 non-null    float64
 7   CAtBat       263 non-null    float64
 8   CHits        263 non-null    float64
 9   CHmRun       263 non-null    float64
 10  CRuns        263 non-null    float64
 11  CRBI         263 non-null    float64
 12  CWalks       263 non-null    float64
 13  PutOuts      263 non-null    float64
 14  Assists      263 non-null    float64
 15  Errors       263 non-null    float64
 16  League_N     263 non-null    uint8
 17  Division_W   263 non-null    uint8
 18  NewLeague_N  263 non-null    uint8
dtypes: float64(16), uint8(3)
memory usage: 35.7 KB
```

```python
alphas = 10**np.linspace(10,-2,100)*0.5
alphas
```

[4] ✓ 0.0s Python

```
array([5.00000000e+09, 3.78231664e+09, 2.86118383e+09, 2.16438064e+09,
       1.63727458e+09, 1.23853818e+09, 9.36908711e+08, 7.08737081e+08,
       5.36133611e+08, 4.05565415e+08, 3.06795364e+08, 2.32079442e+08,
       1.75559587e+08, 1.32804389e+08, 1.00461650e+08, 7.59955541e+07,
       5.74878498e+07, 4.34874501e+07, 3.28966612e+07, 2.48851178e+07,
       1.88246790e+07, 1.42401793e+07, 1.07721735e+07, 8.14875417e+06,
       6.16423370e+06, 4.66301673e+06, 3.52740116e+06, 2.66834962e+06,
       2.01850863e+06, 1.52692775e+06, 1.15506485e+06, 8.73764200e+05,
       6.60970574e+05, 5.00000000e+05, 3.78231664e+05, 2.86118383e+05,
       2.16438064e+05, 1.63727458e+05, 1.23853818e+05, 9.36908711e+04,
       7.08737081e+04, 5.36133611e+04, 4.05565415e+04, 3.06795364e+04,
       2.32079442e+04, 1.75559587e+04, 1.32804389e+04, 1.00461650e+04,
       7.59955541e+03, 5.74878498e+03, 4.34874501e+03, 3.28966612e+03,
       2.48851178e+03, 1.88246790e+03, 1.42401793e+03, 1.07721735e+03,
       8.14875417e+02, 6.16423370e+02, 4.66301673e+02, 3.52740116e+02,
       2.66834962e+02, 2.01850863e+02, 1.52692775e+02, 1.15506485e+02,
       8.73764200e+01, 6.60970574e+01, 5.00000000e+01, 3.78231664e+01,
       2.86118383e+01, 2.16438064e+01, 1.63727458e+01, 1.23853818e+01,
       9.36908711e+00, 7.08737081e+00, 5.36133611e+00, 4.05565415e+00,
       3.06795364e+00, 2.32079442e+00, 1.75559587e+00, 1.32804389e+00,
       1.00461650e+00, 7.59955541e-01, 5.74878498e-01, 4.34874501e-01,
       3.28966612e-01, 2.48851178e-01, 1.88246790e-01, 1.42401793e-01,
       1.07721735e-01, 8.14875417e-02, 6.16423370e-02, 4.66301673e-02,
       3.52740116e-02, 2.66834962e-02, 2.01850863e-02, 1.52692775e-02,
       1.15506485e-02, 8.73764200e-03, 6.60970574e-03, 5.00000000e-03])
```

```python
ridge = Ridge(normalize = True)
coefs = []
for a in alphas:
    ridge.set_params(alpha = a)
    ridge.fit(X, y)
    coefs.append(ridge.coef_)
np.shape(coefs)
```

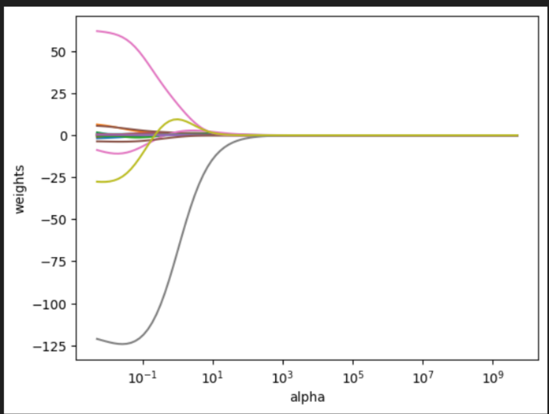
[5] ✓ 0.3s Python

```
(100, 19)
```

```python
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
```

[6] ✓ 1.0s Python

```
Text(0, 0.5, 'weights')
```



```
X_train, X_test , y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=1)
```
[7] ✓ 0.0s Python

```
ridge2 = Ridge(alpha = 4, normalize = True)
ridge2.fit(X_train, y_train) # Fit a ridge regression on the training data
pred2 = ridge2.predict(X_test) # Use this model to predict the test data
print(pd.Series(ridge2.coef_, index = X.columns)) # Print coefficients
print(mean_squared_error(y_test, pred2)) # Calculate the test MSE
```
[8] ✓ 0.0s Python

```
AtBat          0.098658
Hits           0.446094
HmRun          1.412107
Runs           0.660773
RBI            0.843403
Walks          1.008473
Years          2.779882
CAtBat         0.008244
CHits          0.034149
CHmRun         0.268634
CRuns          0.070407
CRBI           0.070060
CWalks         0.082795
PutOuts        0.104747
Assists       -0.003739
Errors         0.268363
League_N       4.241051
Division_W   -30.768885
NewLeague_N    4.123474
dtype: float64
106216.52238005561
```

```
ridge3 = Ridge(alpha = 10**10, normalize = True)
ridge3.fit(X_train, y_train) # Fit a ridge regression on the training data
pred3 = ridge3.predict(X_test) # Use this model to predict the test data
print(pd.Series(ridge3.coef_, index = X.columns)) # Print coefficients
print(mean_squared_error(y_test, pred3)) # Calculate the test MSE
```
[9] ✓ 0.0s Python

```
AtBat          1.317464e-10
Hits           4.647486e-10
HmRun          2.079865e-09
Runs           7.726175e-10
RBI            9.390640e-10
Walks          9.769219e-10
Years          3.961442e-09
CAtBat         1.060533e-11
CHits          3.993605e-11
CHmRun         2.959428e-10
CRuns          8.245247e-11
CRBI           7.795451e-11
CWalks         9.894387e-11
PutOuts        7.268991e-11
Assists       -2.615885e-12
Errors         2.084514e-10
League_N      -2.501281e-09
Division_W    -1.549951e-08
NewLeague_N   -2.023196e-09
dtype: float64
172862.23580379886
```

```
ridge2 = Ridge(alpha = 0, normalize = True)
ridge2.fit(X_train, y_train) # Fit a ridge regression on the training data
pred = ridge2.predict(X_test) # Use this model to predict the test data
print(pd.Series(ridge2.coef_, index = X.columns)) # Print coefficients
print(mean_squared_error(y_test, pred)) # Calculate the test MSE
```
[10] ✓ 0.0s Python

```
AtBat           -1.821115
Hits             4.259156
HmRun           -4.773401
Runs            -0.038760
RBI              3.984578
Walks            3.470126
Years            9.498236
CAtBat          -0.605129
CHits            2.174979
CHmRun           2.979306
CRuns            0.266356
CRBI            -0.598456
CWalks           0.171383
PutOuts          0.421063
Assists          0.464379
Errors          -6.024576
League_N       133.743163
Division_W    -113.743875
NewLeague_N    -81.927763
dtype: float64
116690.46856659521
```

```
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```
[11] ✓ 0.0s Python

```
0.5748784976988678
```

```python
ridge4 = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge4.fit(X_train, y_train)
mean_squared_error(y_test, ridge4.predict(X_test))
```

[12] ✓ 0.0s Python

```
99825.6489629273
```

```python
ridge4.fit(X, y)
pd.Series(ridge4.coef_, index = X.columns)
```

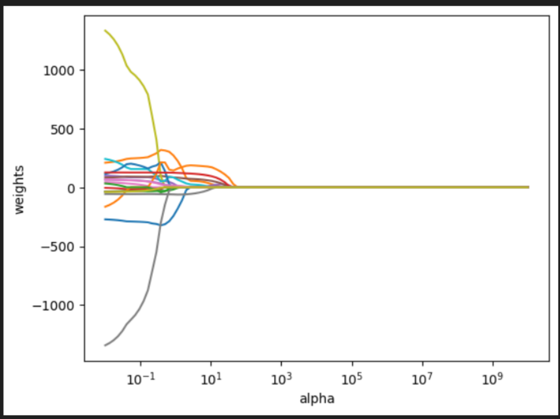
[13] ✓ 0.0s Python

```
AtBat           0.055838
Hits            0.934879
HmRun           0.369048
Runs            1.092480
RBI             0.878259
Walks           1.717770
Years           0.783515
CAtBat          0.011318
CHits           0.061101
CHmRun          0.428333
CRuns           0.121418
CRBI            0.129351
CWalks          0.041990
PutOuts         0.179957
Assists         0.035737
Errors         -1.597699
League_N       24.774519
Division_W    -85.948661
NewLeague_N     8.336918
dtype: float64
```

```python
lasso = Lasso(max_iter = 10000, normalize = True)
coefs = []
for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(scale(X_train), y_train)
    coefs.append(lasso.coef_)
ax = plt.gca()
ax.plot(alphas*2, coefs)
ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('weights')
```

[14] ✓ 1.0s Python

```
Text(0, 0.5, 'weights')
```



```
lassocv = LassoCV(alphas = None, cv = 10, max_iter = 100000, normalize = True)
lassocv.fit(X_train, y_train)
lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train, y_train)
mean_squared_error(y_test, lasso.predict(X_test))
```
[15] ✓ 0.5s Python

```
104960.65853895503
```

```
pd.Series(lasso.coef_, index=X.columns)
```
[16] ✓ 0.0s Python

```
AtBat          0.000000
Hits           1.082446
HmRun          0.000000
Runs           0.000000
RBI            0.000000
Walks          2.906388
Years          0.000000
CAtBat         0.000000
CHits          0.000000
CHmRun         0.219367
CRuns          0.000000
CRBI           0.513975
CWalks         0.000000
PutOuts        0.368401
Assists       -0.000000
Errors        -0.000000
League_N       0.000000
Division_W   -89.064338
NewLeague_N    0.000000
dtype: float64
```

Interpretation:

- Ridge and lasso regression are regularization techniques used to prevent overfitting in linear regression models.
- The choice of alpha determines the strength of regularization, with higher alpha leading to more regularization.
  - As we can see from the output of the code, when the value of alpha is small the output values are significantly small but when the alpha is increased the output values decreases with higher rate.
- The code assesses model performance using mean squared error on the test set for both ridge and lasso regression.
- Cross-validation is employed to find optimal alpha values for ridge and lasso, enhancing model generalization.

Experiment 3

Logistic Regression:

Dataset URL: https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset

```python
#Logistic regression
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```
[1] ✓ 6.1s Python

```python
df = pd.read_csv('Bank Customer Churn Prediction.csv')
```
[2] ✓ 0.4s Python

```python
X = df[['credit_score', 'age', 'tenure', 'balance', 'products_number', 'credit_card', 'active_member', 'estimated_salary', 'gender', 'country']]
X = pd.get_dummies(X, columns=['gender', 'country'], drop_first=True)
y = df['churn']
```
[3] ✓ 0.0s Python

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```
[4] ✓ 0.0s Python

```python
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```
[5] ✓ 0.0s Python

```python
print(f"Accuracy: {accuracy}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_rep)
#TN FP
#FN TP
```
[6] ✓ 0.0s Python

```
Accuracy: 0.811

Confusion Matrix:
[[1543   64]
 [ 314   79]]

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      1607
           1       0.55      0.20      0.29       393

    accuracy                           0.81      2000
   macro avg       0.69      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000
```

Interpretations on the basis of the values:

Accuracy:

Accuracy is the ratio of correctly predicted instances to the total instances. It is a general measure of the model's overall correctness. In this case, an accuracy of 0.811 (or 81.1%) means that the model correctly predicted the churn status for approximately 81.1% of the instances in the dataset.

Confusion Matrix:

A confusion matrix is a table that describes the performance of a classification model. It consists of four terms: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). In your case:

- True Positive (TP): 79 instances where the model correctly predicted churn (1).
- True Negative (TN): 1543 instances where the model correctly predicted no churn (0).
- False Positive (FP): 64 instances where the model incorrectly predicted churn when there was no churn.
- False Negative (FN): 314 instances where the model incorrectly predicted no churn when there was churn.

Classification Report:

Precision, recall, and F1-score are metrics that provide more insights, especially in imbalanced datasets.

- Precision:
  - Precision is the ratio of correctly predicted positive observations to the total predicted positives.
  - Precision = TP / (TP + FP)
  - In this case, the precision for class 1 (churn) is approximately 0.55, indicating that 55% of the instances predicted as churn were actually churn.
- Recall (Sensitivity):
  - Recall is the ratio of correctly predicted positive observations to the total actual positives.
  - Recall = TP / (TP + FN)
  - In this case, the recall for class 1 (churn) is approximately 0.20, indicating that the model correctly identified 20% of the actual churn instances.
- F1-score:
  - F1-score is the weighted average of precision and recall.
  - F1-score = 2 * (Precision * Recall) / (Precision + Recall)

It balances precision and recall, providing a single metric that considers both false positives and false negatives.

The weighted average of the F1-scores for both classes is given in the report.

- Support:
    - Support is the number of actual occurrences of each class in the specified dataset.
- Macro Average and Weighted Average:
    - Macro average calculates the unweighted average of precision, recall, and F1-score for all classes.
    - Weighted average considers the number of instances for each class, providing more weight to the majority class.

Polynomial Regression:

```python
#Polynomial regression
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
```
[7] ✓ 0.0s Python

```python
X = df[['credit_score', 'age', 'tenure', 'balance', 'products_number', 'credit_card', 'active_member', 'estimated_salary', 'gender', 'country']]
y = df['churn']
X = pd.get_dummies(X, columns=['gender', 'country'], drop_first=True)
```
[8] ✓ 0.0s Python

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
```
[9] ✓ 0.0s Python

```python
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)
y_pred_poly = poly_reg.predict(X_test_poly)
mse = mean_squared_error(y_test, y_pred_poly)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred_poly)
r2 = r2_score(y_test, y_pred_poly)
```
[10] ✓ 0.0s Python

```python
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R2): {r2}")
```
[11] ✓ 0.0s Python

```
Mean Squared Error (MSE): 0.11198360191185681
Root Mean Squared Error (RMSE): 0.33463951038670975
Mean Absolute Error (MAE): 0.2396460943992281
R-squared (R2): 0.2907391364316939
```

- Mean Squared Error (MSE):
    - MSE measures the average squared difference between the actual and predicted values.
    - A lower MSE indicates better model performance.
    - In this case, an MSE of approximately 0.112 means, on average, the squared difference between the actual and predicted churn values is 0.112.

- Root Mean Squared Error (RMSE):
  - RMSE is the square root of the MSE and provides a measure of the average absolute error.
  - It is expressed in the same units as the target variable.
  - In this case, an RMSE of approximately 0.335 means, on average, the absolute difference between the actual and predicted churn values is around 0.335.
- Mean Absolute Error (MAE):
  - MAE is the average absolute difference between the actual and predicted values.
  - It is less sensitive to outliers compared to MSE.
  - In this case, an MAE of approximately 0.240 means, on average, the absolute difference between the actual and predicted churn values is around 0.240.
- R-squared (R2):
  - R-squared measures the proportion of the variance in the dependent variable (churn) that is predictable from the independent variables (features).
  - R-squared values range from 0 to 1, where 1 indicates a perfect fit.
  - In this case, an R-squared of approximately 0.291 means that around 29.1% of the variance in churn can be explained by the features in the model.

Experiment 4

Implementing Decision Tree on diabetes dataset:

Dataset URL: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```
[826] ✓ 0.0s Python

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
df = pd.read_csv("diabetes.csv", header=None, names=col_names)
df.drop(0, inplace=True)
df.head()
```
[827] ✓ 0.0s Python

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1

```
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = df[feature_cols]
y = df.label
```
[828] ✓ 0.0s Python

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```
[829] ✓ 0.0s Python

```
d_tree = DecisionTreeClassifier()
d_tree = d_tree.fit(X_train,y_train)
y_pred = d_tree.predict(X_test)
```
[830] ✓ 0.0s Python

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```
[831] ✓ 0.0s Python

```
Accuracy: 0.7056277056277056
```

Interpretations:

In this code we can see the following things:

- The value 0.7056277056277056 represents the accuracy score, which is a metric used to evaluate the performance of a machine learning model.
- The higher the accuracy score, the better the model is at making correct predictions.

Enhancing the accuracy of the decision tree:

```python
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
clf = clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
```

```python
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7705627705627706
```

- Here we can see that the accuracy of the model increases when we ass the criterion of ‘entropy’ and ‘max_depth’
- The higher the accuracy, the better the model is at making predictions.

Experiment 5

Implementing Random Forest:

Dataset URL: https://www.kaggle.com/datasets/yufengsui/portuguese-bank-marketing-data-set

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import export_graphviz
from IPython.display import Image
from sklearn import tree
from matplotlib import pyplot as plt
import graphviz
```

[49] Python

```python
df = pd.read_csv('bank_cleaned.csv')
df.info()
```

[50] Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40841 entries, 0 to 40840
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Unnamed: 0       40841 non-null  int64  
 1   age              40841 non-null  int64  
 2   job              40841 non-null  object 
 3   marital          40841 non-null  object 
 4   education        40841 non-null  object 
 5   default          40841 non-null  object 
 6   balance          40841 non-null  int64  
 7   housing          40841 non-null  object 
 8   loan             40841 non-null  object 
 9   day              40841 non-null  int64  
 10  month            40841 non-null  object 
 11  duration         40841 non-null  float64
 12  campaign         40841 non-null  int64  
 13  pdays            40841 non-null  int64  
 14  previous         40841 non-null  int64  
 15  poutcome         40841 non-null  object 
 16  response         40841 non-null  object 
 17  response_binary  40841 non-null  int64  
dtypes: float64(1), int64(8), object(9)
memory usage: 5.6+ MB
```

```python
df.head()
```

[51] Python

	Unnamed: 0	age	job	marital	education	default	balance	housing	loan	day	month	duration	campaign	pdays	previous	poutcome	resp
0	0	58	management	married	tertiary	no	2143	yes	no	5	may	4.35	1	-1	0	unknown	
1	1	44	technician	single	secondary	no	29	yes	no	5	may	2.52	1	-1	0	unknown	
2	2	33	entrepreneur	married	secondary	no	2	yes	yes	5	may	1.27	1	-1	0	unknown	
3	5	35	management	married	tertiary	no	231	yes	no	5	may	2.32	1	-1	0	unknown	
4	6	28	management	single	tertiary	no	447	yes	yes	5	may	3.62	1	-1	0	unknown	

```python
X = df.drop(['response','response_binary'], axis=1)
y = df['response_binary']
Enc= LabelEncoder()
for column in X.select_dtypes(include = ['object']).columns:
    X[column] = Enc.fit_transform(X[column])

# Split the dataset 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)
```

[52] Python

```python
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```python
y_pred = rf.predict(X_test)
```

```python
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9135757130615743
```

```python
param_dist = {'n_estimators': randint(50,500), 'max_depth': randint(1,20)}
rf = RandomForestClassifier()
```

```python
rand_search = RandomizedSearchCV(rf, param_distributions = param_dist, n_iter=5, cv=5)
rand_search.fit(X_train, y_train)
```

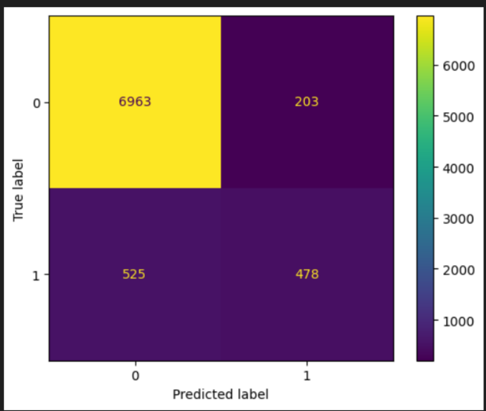
```
RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```python
best_rf = rand_search.best_estimator_
print('Best hyperparameters:', rand_search.best_params_)
```

```
Best hyperparameters: {'max_depth': 12, 'n_estimators': 329}
```

```python
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(confusion_matrix=cm).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a4b3caab40>
```




```python
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.9108826049700086
Precision: 0.7019089574155654
Recall: 0.4765702891326022
```

Interpretation over the metrics calculated:

The confusion matrix shows the performance of a random forest model on a classification task. The rows represent the actual labels, and the columns represent the predicted labels. Each cell shows the number of instances in a particular category. In this case, the model is trying to predict whether a customer will churn or not (represented by 0 and 1).

1. Accuracy (0.91): This metric tells us how well the model performed overall. An accuracy of 91% means that the model classified 91% of the instances correctly.
2. Precision (0.70): This metric tells us the proportion of positive predictions that were actually correct. A precision of 70% means that out of all the instances that the model predicted to churn, 70% of them actually churned.
3. Recall (0.48): This metric tells us the proportion of actual positive instances that were identified correctly. A recall of 48% means that out of all the customers who actually churned, the model identified only 48% of them.

In simpler terms, the model is very good at identifying non-churning customers (high accuracy), but it struggles to identify churning customers (low recall). This could be because churning customers are a smaller proportion of the overall dataset, so the model is less likely to learn their characteristics.

Experiment 6

Implementing Gradient Boosting:

Dataset: https://www.kaggle.com/lodetomasi1995/income-classification

```python
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import preprocessing
#from sklearn import warnings.filterwarnings("ignore")
```

```
C:\Users\wajid\AppData\Local\Temp\ipykernel_32736\529460093.py:3: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

```python
df = pd.read_csv('D:\DESKTOP\Cogntive Analytics\income classification\income_evaluation.csv')
df.head()
```

```
<>:1: SyntaxWarning: invalid escape sequence '\D'
<>:1: SyntaxWarning: invalid escape sequence '\D'
C:\Users\wajid\AppData\Local\Temp\ipykernel_32736\603102860.py:1: SyntaxWarning: invalid escape sequence '\D'
  df = pd.read_csv('D:\DESKTOP\Cogntive Analytics\income classification\income_evaluation.csv')
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

```python
df.columns
```

```
Index(['age', ' workclass', ' fnlwgt', ' education', ' education-num',
       ' marital-status', ' occupation', ' relationship', ' race', ' sex',
       ' capital-gain', ' capital-loss', ' hours-per-week', ' native-country',
       ' income'],
      dtype='object')
```

```python
df.columns = df.columns.str.strip()
X_train, X_test, y_train, y_test = train_test_split(df.drop('income', axis=1),
                                                    df['income'], test_size=0.2)
```

```python
gbc = GradientBoostingClassifier(n_estimators=500,learning_rate=0.05,random_state=100,max_features=5 )
```

```python
X_train_encoded = pd.get_dummies(X_train)
X_test_encoded = pd.get_dummies(X_test)

gbc.fit(X_train_encoded, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.05, max_features=5, n_estimators=500,
                           random_state=100)
```

```python
X_train_encoded = pd.get_dummies(X_train)
X_test_encoded = pd.get_dummies(X_test)
X_test_encoded = X_test_encoded.reindex(columns=X_train_encoded.columns, fill_value=0)
print(confusion_matrix(y_test, gbc.predict(X_test_encoded)))
```

```
[[4689  245]
 [ 626  953]]
```

```
print("GBC accuracy is %2.2f" % accuracy_score(y_test, gbc.predict(X_test_encoded)))
```

```
GBC accuracy is 0.87
```

```
from sklearn.metrics import classification_report
pred = gbc.predict(X_test_encoded)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.88      0.95      0.92      4934
        >50K       0.80      0.60      0.69      1579

    accuracy                           0.87      6513
   macro avg       0.84      0.78      0.80      6513
weighted avg       0.86      0.87      0.86      6513
```

```
from sklearn.model_selection import GridSearchCV
```

```
grid = { 'learning_rate':[0.01,0.05,0.1],
         'n_estimators':np.arange(100,500,100), }
```

```
gb = GradientBoostingClassifier()
gb_cv = GridSearchCV(gb, grid, cv = 4)
gb_cv.fit(X_train_encoded, y_train)
print("Fitting completed successfully")
```

```
Fitting completed successfully
```

```
#print("Is the GridSearchCV fitted?", gb_cv.cv_results_ is not None)
print("Available attributes:", dir(gb_cv))
# Check if the object is correctly named and assigned
print(gb_cv)


#print("Best Estimator:", gb_cv.best_estimator_)
print("'cv_results_' available:", hasattr(gb_cv, "cv_results_"))
print("'best_params_' available:", hasattr(gb_cv, "best_params_"))
print("cv_results_:", gb_cv.best_index_)
```

```
Available attributes: ['__abstractmethods__', '__annotations__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
GridSearchCV(cv=4, estimator=GradientBoostingClassifier(),
             param_grid={'learning_rate': [0.01, 0.05, 0.1],
                         'n_estimators': array([100, 200, 300, 400])})
'cv_results_' available: True
'best_params_' available: True
cv_results_: 11
```

```
print("Best Parameters:",gb_cv.best_params_)
print("Train Score:",gb_cv.best_score_)
print("Test Score:",gb_cv.score(X_test_encoded,y_test))
```

```
Best Parameters: {'learning_rate': 0.1, 'n_estimators': 400}
Train Score: 0.871237714987715
Test Score: 0.8756333486872409
```

```
grid = {'max_depth':[2,3,4,5,6,7] }
gb = GradientBoostingClassifier(learning_rate=0.1,n_estimators=400)
gb_cv = GridSearchCV(gb, grid, cv = 4)
gb_cv.fit(X_train_encoded,y_train)
```

```
GridSearchCV
estimator: GradientBoostingClassifier
GradientBoostingClassifier
```

```
print("Best Parameters:",gb_cv.best_params_)
print("Train Score:",gb_cv.best_score_)
print("Test Score:",gb_cv.score(X_test_encoded,y_test))
```

```
Best Parameters: {'max_depth': 3}
Train Score: 0.8711609336609336
Test Score: 0.8756333486872409
```

Interpretation of the Confusion Matrix and Model Performance

Based on the confusion matrix and performance metrics, here's an interpretation of the model's behaviour on the dataset:

Confusion Matrix:

- Correct Classifications:
  - The model correctly classified 4689 data points in the <=50K class and 953 data points in the >50K class.
  - This indicates a good ability to identify data points belonging to each class.
- Incorrect Classifications:
  - The model misclassified 245 data points from <=50K class and 626 data points from the >50K class.
  - There's a higher number of misclassifications for the >50K class, suggesting the model struggles more with this category.

Performance Metrics:

- Overall Accuracy: 87% - This indicates a good overall performance of the model in classifying data points correctly.
- Precision:
  - <=50K: 88% - The model is good at identifying true positives for the <=50K class (out of all predicted positives, 88% are actually positive).
  - 50K: 80% - The model is less precise for the >50K class, meaning there might be more false positives (predicted positive but actual negative).

Recall:

- <=50K: 95% - The model captures most of the actual positives in the <=50K class (out of all actual positives, 95% are predicted positive).
- 50K: 60% - The model misses a significant portion (40%) of the actual positives in the >50K class (false negatives). This is the major contributor to the lower precision for this class.

F1-Score: This metric balances precision and recall. It's higher for the <=50K class (0.92) compared to the >50K class (0.69), again reflecting the model's better performance for the former.

Overall Interpretation:

The model performs well in classifying data points with an overall accuracy of 87%. It excels at identifying <=50K class data points with high precision (88%) and recall (95%). However, the model struggles with the >50K class, particularly with recall (60%), leading to more false negatives (missed positives). This suggests the model might be biased towards the majority class (<=50K) or have difficulty learning the patterns in the >50K class data.

Experiment 7

AdaBoost Classifier on Breast cancer dataset:

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore")
```

```python
breast_cancer = load_breast_cancer()
```

```python
.DataFrame(breast_cancer.data, columns = breast_cancer.feature_names)
.Categorical.from_codes(breast_cancer.target, breast_cancer.target_names)
```

```python
encoder = LabelEncoder()
binary_encoded_y = pd.Series(encoder.fit_transform(y))
```

```python
:rain_y, test_y = train_test_split(X, binary_encoded_y, random_state = 1)
```

```python
classifier = AdaBoostClassifier()
classifier.fit(train_X,train_y)
prediction = classifier.predict(test_X)
```

```python
confusion_matrix(test_y, prediction)
```

```
array([[85,  3],
       [ 3, 52]], dtype=int64)
```

```python
accuracy = accuracy_score(test_y, prediction)
print('AdaBoost Accuracy: ', accuracy)
```

```
AdaBoost Accuracy:  0.958041958041958
```

Interpretation:

Based on the confusion matrix and accuracy, your AdaBoost classifier seems to be performing very well. Here's a breakdown:

- High True Positives (85) and True Negatives (52): This indicates the classifier effectively identified most of the actual positive and negative instances.
- Low False Positives (3) and False Negatives (3): The low number of misclassified instances suggests the classifier generalizes well and doesn't overfit to the training data.

Overall, your AdaBoost classifier demonstrates a high level of accuracy in distinguishing between the positive and negative classes based on the provided confusion matrix and accuracy.

Experiment 8

Linear Regression in R:

Dataset: https://ocw.mit.edu/courses/15-097-prediction-machine-learning-and-statistics-spring-2012/resources/salary/

```
dataset <- read.csv("D:/DESKTOP/Cogntive Analytics/Experiments/Datasets/Salary.csv")
library(caTools)
split <- sample.split(dataset$Salary, SplitRatio = 0.7)
trainingset <- subset(dataset, split == TRUE)
testset <- subset(dataset, split == FALSE)
lm.r <- lm(formula = Salary ~ YearsExperience, data = trainingset)
coef(lm.r)
ypred <- predict(lm.r, newdata = testset)
library(ggplot2)
ggplot() + geom_point(aes(x = trainingset$YearsExperience,
                          y = trainingset$Salary), colour = 'red')
+ geom_line(aes(x = trainingset$YearsExperience, y = predict(lm.r, newdata = trainingset)),
|          colour = 'blue') +
  ggtitle('Salary vs Experience (Training set)') +
  xlab('Years of experience') +
  ylab('Salary')

ggplot() + geom_point(aes(x = testset$YearsExperience,
                          y = testset$Salary),
                      colour = 'red') +
  geom_line(aes(x = trainingset$YearsExperience,
                y = predict(lm.r, newdata = trainingset)),
           colour = 'blue') +
  ggtitle('Salary vs Experience (Test set)') +
  xlab('Years of experience') +
  ylab('Salary')
```

Interpretations:

Intercept (30732.10): This value represents the expected value of the dependent variable when the independent variable (YearsExperience) is equal to zero. In simpler terms, it's the point where the regression line crosses the y-axis.

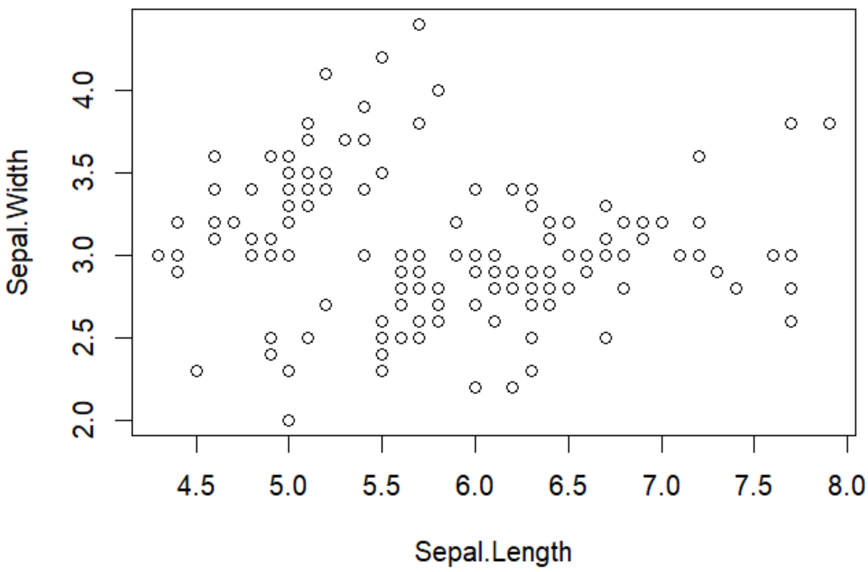
YearsExperience (8557.83): This coefficient represents the change in the dependent variable for every one unit increase in the independent variable (YearsExperience). It signifies the slope of the regression line. Since the coefficient is positive (8557.83), we can interpret that there's a positive linear relationship between years of experience and the dependent variable. In other words, as the years of experience increase by one unit, the dependent variable is expected to increase by 8557.83 units on average.

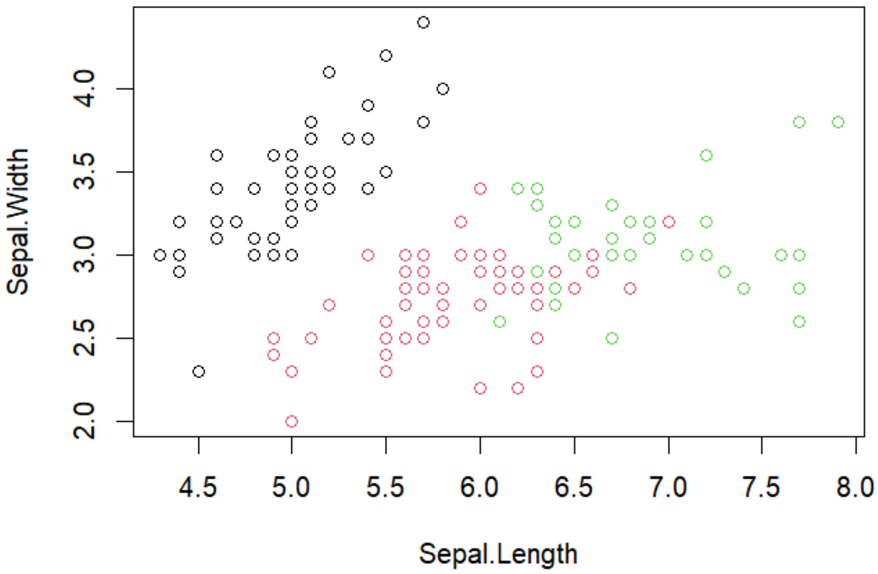
Experiment 9

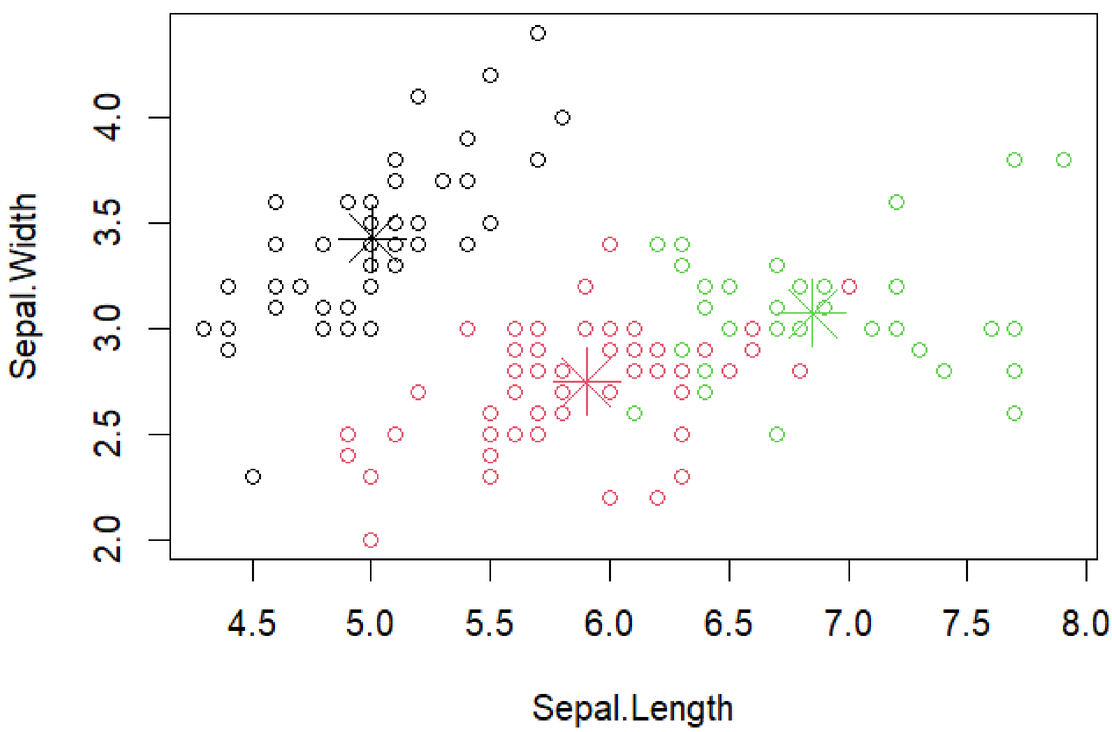
K-means Clustering in R:

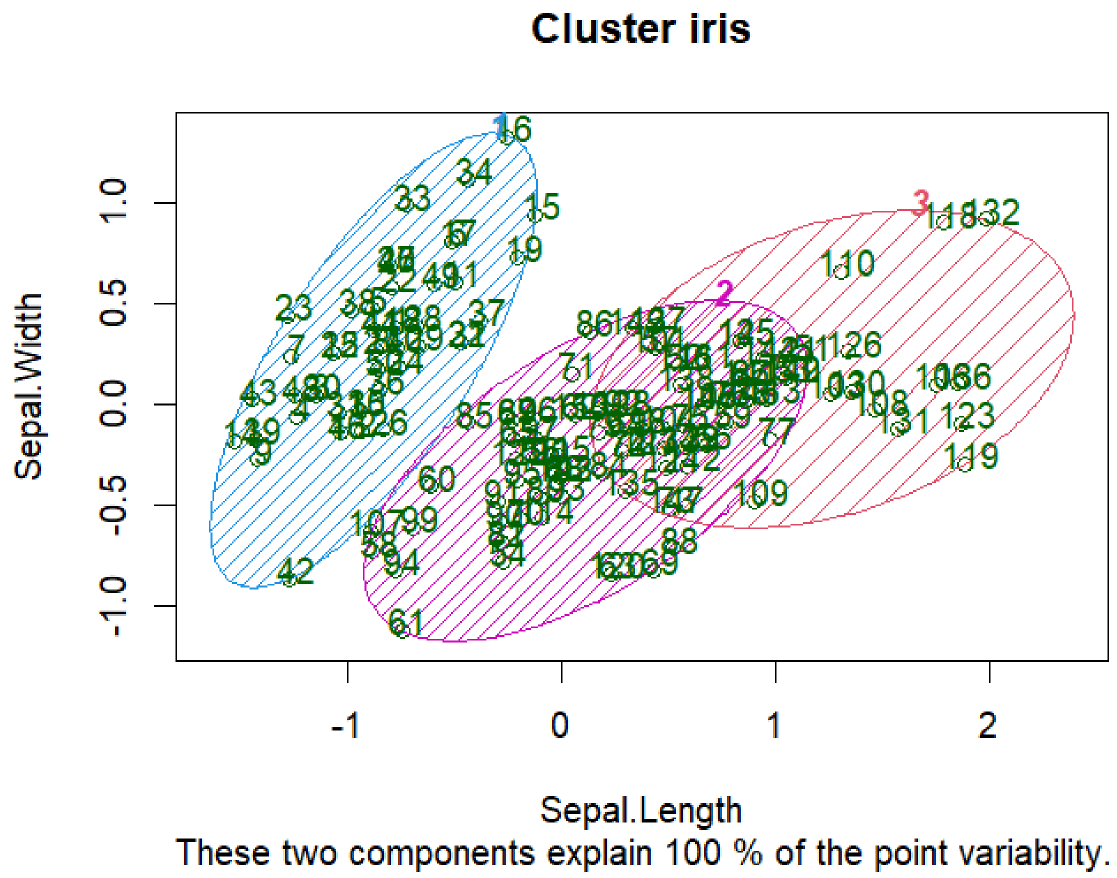
Dataset: https://www.kaggle.com/datasets/saurabh00007/iriscsv

```
library(ClusterR)
library(cluster)

data(iris)
str(iris)

iris_1 <- iris[, -5]
set.seed(240)

kmeans.re <- kmeans(iris_1, centers = 3, nstart = 20)
kmeans.re
kmeans.re$cluster

cm <- table(iris$Species, kmeans.re$cluster)
cm

plot(iris_1[c("Sepal.Length", "Sepal.Width")])
plot(iris_1[c("Sepal.Length", "Sepal.Width")], col = kmeans.re$cluster)

plot(iris_1[c("Sepal.Length", "Sepal.Width")], col = kmeans.re$cluster, main = "K-means with 3 clusters")
kmeans.re$centers
kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")]
points(kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")], col = 1:3, pch = 8, cex = 3)

y_kmeans <- kmeans.re$cluster
clusplot(iris_1[, c("Sepal.Length", "Sepal.Width")],
         y_kmeans,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = paste("Cluster iris"),
         xlab = 'Sepal.Length',
         ylab = 'Sepal.Width')
```




Sepal.Width
Sepal.Length
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


K-means with 3 clusters
Sepal.Width
Sepal.Length
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



Results and Interpretation:

The output (kmeans.re) shows that the algorithm successfully assigned data points to three clusters with sizes 50, 62, and 38.

The kmeans.re$centers table displays the average values (centroids) for each cluster in terms of sepal and petal length and width.

- Cluster 1 (centroid: Sepal Length - 5.01, Sepal Width - 3.43) seems to represent flowers with the smallest overall dimensions.
- Cluster 2 (centroid: Sepal Length - 5.90, Sepal Width - 2.75) appears to group flowers with slightly larger sepals but narrower sepals compared to Cluster 3.
- Cluster 3 (centroid: Sepal Length - 6.85, Sepal Width - 3.07) likely represents flowers with the largest sepals and wider sepals on average.

Visualized the cluster assignments (kmeans.re$cluster) in two ways:

- By colouring data points in a scatter plot based on their assigned cluster. This visually highlights the separation between clusters in the two-dimensional space of sepal length and width.
- By using the clusplot function from the ClusterR library, which creates a more comprehensive cluster plot with shading and labels.