Appendix_Codes document

1. import pandas as pd

```
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trail4.csv'

# Read the CSV file
data = pd.read_csv(file_path)

# Print descriptive statistics
print(data.describe())
```

2.

```
import pandas as pd
import numpy as np

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Find indices where value is 99999
indices = np.where(data == 99999)

# Extract row and column indices
row_indices = indices[0]
col_indices = indices[1]

# Print the indices
for row, col in zip(row_indices, col_indices):
    print(f"Row: {row}, Column: {col}")
```

3.

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

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
```

```
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Calculate the total number of missing values in each column
missing_values = data.isnull().sum()

# Print the total number of missing values in each column
print("Missing values in each column:")
print(missing_values)

# Data Visualization
plt.figure(figsize=(10, 6))
sns.barplot(x=missing_values.index, y=missing_values.values)
plt.xticks(rotation=90)
plt.title('Number of Missing Values in Each Column')
plt.ylabel('Number of Missing Values')
plt.xlabel('Columns')
plt.show()
```

4.

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

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Function to find outliers using IQR
def find_outliers(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()
    return outliers

# Calculate Q1, Q3, and IQR for the original data
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

# Find outliers in each column
outliers_before = find_outliers(data)
```

```
# Remove rows containing outliers
data_cleaned = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)]
outliers_after = find_outliers(data_cleaned)

# Print the total number of outliers in each column before and after removal
print("Outliers in each column before removal:")
print(outliers_before)
print("\nOutliers in each column after removal:")
print(outliers_after)

# Data Visualization
plt.figure(figsize=(15, 6))

# Plot before outlier removal
plt.subplot(1, 2, 1)
sns.barplot(x=outliers_before.index, y=outliers_before.values)
plt.xticks(rotation=90)
plt.title('Number of Outliers in Each Column (Before Removal)')
plt.ylabel('Number of Outliers')
plt.xlabel('Columns')

# Plot after outlier removal
plt.subplot(1, 2, 2)
sns.barplot(x=outliers_after.index, y=outliers_after.values)
plt.xticks(rotation=90)
plt.title('Number of Outliers in Each Column (After Removal)')
plt.ylabel('Number of Outliers')
plt.xlabel('Columns')

plt.tight_layout()
plt.show()
```

5.

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

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Calculate the correlation matrix
corr_matrix = data.corr()
```

```
# Plot the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix of All Variables')
plt.show()



correlation_with_target = corr_matrix['Risk'].sort_values(ascending=False)

# Print the ranking of features based on correlation with target
print("Ranking of feature columns based on correlation with target variable 'Risk':")
print(correlation_with_target)
```

6.

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

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Calculate the correlation matrix
corr_matrix = data.corr()

# Select the top 8 features based on absolute correlation with the target variable 'Risk'

top_features = corr_matrix['Risk'].abs().sort_values(ascending=False).head(9).index.tolist()

# Remove the target variable from the list to have only 8 attributes
top_features.remove('Risk')

# Plot the scatter matrix/pair plot of the top 8 features
sns.pairplot(data[top_features])
plt.suptitle('Scatter Matrix of Top 8 Features', size=16)
plt.show()
```

7.

Cleaned Audit_Dataset (removing rows). This is Initial accuracy

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score




file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Handle missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)

# Standardize the data
scaler = StandardScaler()
X_normalized = scaler.fit_transform(data_imputed.drop('Risk', axis=1))

# PCA transformation
pca = PCA(n_components=4)
X_pca = pca.fit_transform(X_normalized)

# Prepare the target variable
y = data_imputed['Risk']

# SVM Model
svm_classifier = SVC()

# 10-fold Cross-Validation
accuracy = cross_val_score(svm_classifier, X_pca, y, cv=10)

# Calculate Mean Accuracy and Standard Deviation
mean_accuracy = accuracy.mean()
std_deviation = accuracy.std()
```

```
    print(f"Mean Accuracy: {mean_accuracy}")
    print(f"Standard Deviation: {std_deviation}")
```

Cleaned Audit_Dataset (replacing missing values with means)

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

# Load dataset
file_path = 'C:/Users/tebib/OneDrive/Desktop/J2/Topics in intelligent
systems/Midterm/Audit_Trial4.csv'
data = pd.read_csv(file_path)


data = data.replace('0', pd.NA)


imputer = SimpleImputer(missing_values=pd.NA, strategy='mean')
data_imputed = imputer.fit_transform(data.drop('Risk', axis=1))
data_imputed = pd.DataFrame(data_imputed, columns=data.drop('Risk', axis=1).columns)

# Add the target variable 'Risk' back into the DataFrame
data_imputed['Risk'] = data['Risk'].values


svm_classifier = SVC()

# Separate features and target
X = data_imputed.drop('Risk', axis=1)
y = data_imputed['Risk']

# Perform 10-fold cross-validation
accuracy = cross_val_score(svm_classifier, X, y, cv=10)

# Calculate mean accuracy and standard deviation
mean_accuracy = accuracy.mean()
std_deviation = accuracy.std()
```

```
# Print the results
print("Mean Accuracy: ", mean_accuracy)
print("Standard Deviation: ", std_deviation)
```

Normalized Audit_Dataset

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Separate features and target
X = data.drop('Risk', axis=1)  # Replace 'Risk' with the actual name of your target column
y = data['Risk']  # Replace 'Risk' with the actual name of your target column

# Impute missing values with the mean
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
X_imputed = imputer.fit_transform(X)

# Normalize features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X_imputed)

# Initialize SVM classifier
svm_classifier = SVC()

# Perform 10-fold cross-validation
accuracies = cross_val_score(svm_classifier, X_normalized, y, cv=10)

# Calculate mean accuracy and standard deviation
mean_accuracy = np.mean(accuracies)
std_deviation = np.std(accuracies)

# Print the results
print("Normalized Audit_Dataset - SVM Accuracies:")
print("10-fold cross-validation mean average:", mean_accuracy)
print("Standard deviation of accuracies:", std_deviation)
```

Reduced feature Dataset to only half i.e 8 attributes, pick your best to show highest accuracy.

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Define the top 8 features
top_features = ['Score', 'PARA_A', 'TOTAL', 'SCORE_A', 'SCORE_B', 'District', 'PARA_B',
'MONEY_Marks']

# Select top features and target variable
X = data[top_features]
y = data['Risk']  # Replace 'Risk' with the actual name of your target column

# Handle missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
X_imputed = imputer.fit_transform(X)

# Normalize features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X_imputed)

# Initialize SVM classifier
svm_classifier = SVC()

# Perform 10-fold cross-validation
accuracies = cross_val_score(svm_classifier, X_normalized, y, cv=10)

# Calculate mean accuracy and standard deviation
mean_accuracy = accuracies.mean()
std_deviation = accuracies.std()
```

```
# Print the results
print("Reduced feature dataset (8 features) - SVM Accuracies:")
print("10-fold cross-validation mean average:", mean_accuracy)
print("Standard deviation of accuracies:", std_deviation)
```

Apply PCA on the Dataset and use only 4 features

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)

# Separate features and target
X = data.drop('Risk', axis=1)  # Replace 'Risk' with the actual name of your target column
y = data['Risk']  # Replace 'Risk' with the actual name of your target column

# Impute missing values and normalize features
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
X_imputed = imputer.fit_transform(X)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Apply PCA
pca = PCA(n_components=4)
X_pca = pca.fit_transform(X_scaled)

# Initialize SVM classifier
svm_classifier = SVC()

# Perform 10-fold cross-validation
accuracies = cross_val_score(svm_classifier, X_pca, y, cv=10)

# Calculate mean accuracy and standard deviation
mean_accuracy = np.mean(accuracies)
std_deviation = np.std(accuracies)
```

```
# Print the results
print("PCA with 4 features - SVM Accuracies:")
print("10-fold cross-validation mean average:", mean_accuracy)
print("Standard deviation of accuracies:", std_deviation)
```

8.

8.

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)


imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
data_imputed = imputer.fit_transform(data.drop('Risk', axis=1))  # Replace 'Risk' with the actual
target column name, if different

# Standardizing the features
scaler = StandardScaler()
data_normalized = scaler.fit_transform(data_imputed)

# Applying PCA
pca = PCA()
pca.fit(data_normalized)

# Print the explained variance ratio for each principal component
print("PCA Explained Variance Ratio:", pca.explained_variance_ratio_)
```

9.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler

# Load dataset
file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)


selected_features = ['Score', 'PARA_A', 'TOTAL', 'SCORE_A', 'SCORE_B', 'District',
'PARA_B', 'MONEY_Marks']


X = data[selected_features]
y = data['Risk']  # Replace 'Risk' with the actual name of your target column


X.fillna(X.mean(), inplace=True)


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Initialize the classifier
classifier = RandomForestClassifier()


classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)


cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

10.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, confusion_matrix,
cohen_kappa_score, roc_auc_score


file_path = r'C:\Users\tebib\OneDrive\Desktop\J2\Topics in intelligent
systems\Midterm\Audit_Trial4.csv'
data = pd.read_csv(file_path)


selected_features = ['Score', 'PARA_A', 'TOTAL', 'SCORE_A', 'SCORE_B', 'District',
'PARA_B', 'MONEY_Marks']


X = data[selected_features]
y = data['Risk']  # Replace 'Risk' with your target column name


X.fillna(X.mean(), inplace=True)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


classifier = RandomForestClassifier()


classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)
y_proba = classifier.predict_proba(X_test)[:, 1]  # Probabilities for AUC score


precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
false_alarm_rate = fp / (fp + tn)
kappa_accuracy = cohen_kappa_score(y_test, y_pred)
specificity = tn / (tn + fp)
auc_score = roc_auc_score(y_test, y_proba)


print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"False Alarm Rate: {false_alarm_rate}")
print(f"Kappa-Accuracy: {kappa_accuracy}")
print(f"Specificity: {specificity}")
print(f"AUC score: {auc_score}")
```