1)question:

```
import pandas as pd

# Load sample dataset
df = pd.read_csv('sample_dataset.csv')

# a) Identify potential customer segments
# Example: Analyze demographics, purchase history, and behavior to identify underserved segments

# b) Segment customers based on their likelihood to churn
# Example: Calculate churn likelihood using RFM (Recency, Frequency, Monetary) analysis

# c) Design a clustering algorithm
# Example: K-means clustering based on purchase history and demographics
from sklearn.cluster import KMeans

# Select features for clustering
features_for_clustering = ['PurchaseFrequency', 'PurchaseRecency', 'Age', 'Gender']

# Normalize features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[features_for_clustering])

# Choose number of clusters
k = 5
kmeans = KMeans(n_clusters=k, random_state=42)
df['Cluster'] = kmeans.fit_predict(df_scaled)

# d) Develop predictive models
# Example: Random Forest for predicting churn

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score


# Select features and target variable

X = df.drop(['CustomerID', 'Churn'], axis=1)

y = df['Churn']


# Split data into train and test sets

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


# Train Random Forest Classifier

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

rf_classifier.fit(X_train, y_train)


# Predict churn

y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)


print("Random Forest Accuracy:", accuracy)


# e) Utilize segments to improve marketing strategies and personalize user experiences

# Example: Send personalized recommendations, targeted promotions based on segments

# Example: Optimize website layout and content based on user segments
```

2nd question:

2. Design and develop a classification model to detect fraudulent transactions in a credit card dataset.

a) Discuss the importance of selecting relevant features and pre-processing techniques to improve the performance of fraud detection models.

b) Describe the evaluation metrics used to assess the performance of credit card fraud detection models. Explain the difference between precision, recall, and F1-score, and discuss their significance in the context of fraud detection.

c) Discuss the ethical considerations associated with credit card fraud detection. Explore the potential biases in fraud detection algorithms and the implications of false positives and false negatives on customers and businesses.

d) Visualize the performance of the model in various plots include Confusion matrix, RoC curve, Calibration and Residual Plots.

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, precision_recall_curve, average_precision_score

from sklearn.calibration import calibration_curve

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.pipeline import make_pipeline


# Load the dataset

data = pd.read_csv('credit_card_fraud_dataset.csv')


# Exploratory Data Analysis

print(data.head())

print(data.info())

print(data.describe())


# Preprocessing

# Feature selection

# Assuming all columns except 'Class' are features

X = data.drop('Class', axis=1)

y = data['Class']


# Splitting the dataset into train and test sets
```

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

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Dimensionality reduction using PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Model training
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train_pca, y_train)

# Model evaluation
y_pred = clf.predict(X_test_pca)
y_pred_proba = clf.predict_proba(X_test_pca)[:, 1]

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
plt.plot(fpr, tpr, color='blue', lw=2)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()

# AUC Score
auc_score = roc_auc_score(y_test, y_pred_proba)
print(f"AUC Score: {auc_score}")

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
average_precision = average_precision_score(y_test, y_pred_proba)
plt.plot(recall, precision, color='blue', lw=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()

# Average Precision Score
print(f"Average Precision Score: {average_precision}")

# Calibration Plot
prob_true, prob_pred = calibration_curve(y_test, y_pred_proba, n_bins=10)
plt.plot(prob_pred, prob_true, marker='o', color='blue', label='Calibration Plot')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.title('Calibration Plot')
plt.legend()
plt.show()
```

3rd question

3. Suppose you are working with a healthcare provider.

a) Perform necessary pre-processing task to fill-in the missing values and transform the required attributes.

b) Compute relevance among the attributes using Attribute Relevance Analysis – Perform Attribute Removal and Attribute Generalization

c) Build a classification model using Support Vector Machine (SVM) to understand whether patient has a disease or not.

d) Design and develop an ensemble classification model to predict whether a patient has a particular disease based on their symptoms and medical history.

e) How would you interpret the model's predictions and communicate them effectively to healthcare professionals?

Code:

```
import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report


# Load the dataset

data = pd.read_csv('healthcare_data.csv')


# Pre-processing tasks

# Handling missing values

imputer = SimpleImputer(strategy='mean')  # Impute missing values with mean for numerical attributes

numerical_cols = ['numerical_attribute1', 'numerical_attribute2']  # Update with numerical column names

data[numerical_cols] = imputer.fit_transform(data[numerical_cols])


imputer = SimpleImputer(strategy='most_frequent')  # Impute missing values with mode for categorical attributes

categorical_cols = ['categorical_attribute1', 'categorical_attribute2']  # Update with categorical column names

data[categorical_cols] = imputer.fit_transform(data[categorical_cols])
```

```
# Attribute transformation - One-hot encoding for categorical attributes

data = pd.get_dummies(data, columns=categorical_cols)


# Attribute Relevance Analysis - Attribute removal and generalization

# Remove irrelevant or redundant attributes

data.drop(['irrelevant_attribute1', 'irrelevant_attribute2'], axis=1, inplace=True)


# Splitting the dataset into features and target variable

X = data.drop('target_variable', axis=1)

y = data['target_variable']


# Splitting 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)


# Build a classification model using Support Vector Machine (SVM)

svm_classifier = SVC(kernel='linear')

svm_classifier.fit(X_train, y_train)

y_pred_svm = svm_classifier.predict(X_test)


# Model evaluation - SVM

accuracy_svm = accuracy_score(y_test, y_pred_svm)

print("Support Vector Machine (SVM) Classifier Accuracy:", accuracy_svm)

print("Support Vector Machine (SVM) Classifier Classification Report:")

print(classification_report(y_test, y_pred_svm))


# Build an ensemble classification model using Random Forest

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

rf_classifier.fit(X_train, y_train)

y_pred_rf = rf_classifier.predict(X_test)


# Model evaluation - Random Forest

accuracy_rf = accuracy_score(y_test, y_pred_rf)
```

```
print("\nRandom Forest Classifier Accuracy:", accuracy_rf)

print("Random Forest Classifier Classification Report:")

print(classification_report(y_test, y_pred_rf))
```

4th question

4. Create a classification system to analyze customer reviews for a product or service.

a) Clean the raw data by handling missing values, removing duplicates, and correcting

inconsistencies.

b) How would you handle challenges such as sarcasm, irony, and varying sentence

structures in natural language text?

c) Evaluate the performance of the recommendation model using appropriate evaluation

metrics (e.g., precision, recall, mean average precision, area under the curve).

d) Use techniques such as cross-validation or holdout evaluation to assess the

generalization performance of the model.

e) Incorporate user feedback into the recommendation model to improve its accuracy and

relevance over time.

code:

```
import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, average_precision_score, roc_auc_score

from nltk.tokenize import word_tokenize


# Load the raw data

raw_data = pd.read_csv('customer_reviews.csv')


# Pre-processing tasks

# Handling missing values

imputer = SimpleImputer(strategy='most_frequent')  # Impute missing values with mode for categorical attributes

raw_data['review_text'] = imputer.fit_transform(raw_data['review_text'].values.reshape(-1, 1))
```

```
# Removing duplicates
raw_data.drop_duplicates(inplace=True)

# Correcting inconsistencies
raw_data['review_text'] = raw_data['review_text'].str.lower()  # Convert text to lowercase
raw_data['review_text'] = raw_data['review_text'].str.replace('[^\w\s]', '')  # Remove special characters

# Handling challenges in natural language text
# Tokenization
raw_data['tokenized_review'] = raw_data['review_text'].apply(word_tokenize)

# Splitting the dataset into features and target variable
X = raw_data.drop('target_variable', axis=1)
y = raw_data['target_variable']

# Splitting 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)

# Build a classification model using Support Vector Machine (SVM)
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)
y_pred_svm = svm_classifier.predict(X_test)

# Model evaluation - SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
map_score_svm = average_precision_score(y_test, y_pred_svm)
auc_score_svm = roc_auc_score(y_test, y_pred_svm)

print("Support Vector Machine (SVM) Classifier Performance:")
print("Accuracy:", accuracy_svm)

print("Precision:", precision_svm)

print("Recall:", recall_svm)

print("F1-score:", f1_svm)

print("Mean Average Precision (MAP):", map_score_svm)

print("Area Under the Curve (AUC):", auc_score_svm)


# Build an ensemble classification model using Random Forest

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

rf_classifier.fit(X_train, y_train)

y_pred_rf = rf_classifier.predict(X_test)


# Model evaluation - Random Forest

accuracy_rf = accuracy_score(y_test, y_pred_rf)

precision_rf = precision_score(y_test, y_pred_rf)

recall_rf = recall_score(y_test, y_pred_rf)

f1_rf = f1_score(y_test, y_pred_rf)

map_score_rf = average_precision_score(y_test, y_pred_rf)

auc_score_rf = roc_auc_score(y_test, y_pred_rf)


print("\nRandom Forest Classifier Performance:")

print("Accuracy:", accuracy_rf)

print("Precision:", precision_rf)

print("Recall:", recall_rf)

print("F1-score:", f1_rf)

print("Mean Average Precision (MAP):", map_score_rf)

print("Area Under the Curve (AUC):", auc_score_rf)


# Cross-validation to assess generalization performance

cv_scores_svm = cross_val_score(svm_classifier, X, y, cv=5)

cv_scores_rf = cross_val_score(rf_classifier, X, y, cv=5)


print("\nCross-validation scores (SVM):", cv_scores_svm)
print("Mean cross-validation score (SVM):", cv_scores_svm.mean())
print("\nCross-validation scores (Random Forest):", cv_scores_rf)
print("Mean cross-validation score (Random Forest):", cv_scores_rf.mean())
```

5th question:

5. Design a classification system to classify emails as spam or non-spam.

a) Clean and pre-process the email data to remove noise, such as HTML tags, special characters, and irrelevant information.

b) Perform the normalization on the text by converting it to lowercase, removing stop words, and stemming or lemmatizing the words.

c) Compute the features include word frequencies, n-grams, TF-IDF scores, and metadata attributes.

d) Use techniques such as cross-validation or holdout evaluation to assess the generalization performance of the model.

e) Evaluate the performance of the classification model using appropriate evaluation metrics.

f) How would you handle the issue of imbalanced datasets, where the number of spam emails is significantly lower than non-spam emails?

Code:

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from imblearn.over_sampling import SMOTE

# Load the raw email data
raw_emails = pd.read_csv('email_data.csv')
```

```
# Clean and pre-process the email data
def clean_text(text):
    text = BeautifulSoup(text, "html.parser").get_text()  # Remove HTML tags
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters and numbers
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra whitespaces
    return text

def normalize_text(text):
    text = text.lower()  # Convert text to lowercase
    text = ' '.join(word for word in text.split() if word not in stop_words)  # Remove stop words
    text = ' '.join(stemmer.stem(word) for word in text.split())  # Apply stemming
    return text

cleaned_emails = raw_emails['email_text'].apply(clean_text)
normalized_emails = cleaned_emails.apply(normalize_text)

# Compute features
count_vectorizer = CountVectorizer()
word_freq_matrix = count_vectorizer.fit_transform(normalized_emails)

tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(normalized_emails)

# Split the dataset into features and target variable
X = tfidf_matrix
y = raw_emails['spam_label']

# Handle imbalanced datasets
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

```
# Build and evaluate the classification model
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
y_pred = nb_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("Classification Model Performance:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC-AUC score:", roc_auc)

# Evaluate generalization performance using cross-validation
cv_scores = cross_val_score(nb_classifier, X_resampled, y_resampled, cv=5)
print("\nCross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
```

6th question:

Build a classification model to predict whether the price of a stock will increase, decrease, or remain stable in the next trading period based on historical stock data and relevant market indicators.

a) Perform dimensionality reduction techniques if necessary to reduce the number of features and improve model efficiency.

b) Apply various statistical models for stock price prediction, such as linear regression, support vector machines, decision trees, random forests, or deep learning models.

c) Evaluate the performance of the trained models using appropriate evaluation metrics such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), or directional accuracy.

d) Continuously update and improve the prediction models based on feedback and changing market conditions.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, accuracy_score

# Load historical stock data and relevant market indicators
stock_data = pd.read_csv('stock_data.csv')

# Perform dimensionality reduction if necessary
# Example: PCA
X = stock_data.drop('target_variable', axis=1)  # Features
y = stock_data['target_variable']  # Target variable

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

pca = PCA(n_components=5)  # Choose number of components
X_reduced = pca.fit_transform(X_scaled)

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

# Apply various statistical models
# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_pred = lr_model.predict(X_test)

# Support Vector Machines (SVM)
svm_model = SVR(kernel='linear')
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)

# Decision Trees
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)

# Random Forests
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# Evaluate the performance of the trained models
# Example evaluation metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE)
mse_lr = mean_squared_error(y_test, lr_pred)
rmse_lr = mean_squared_error(y_test, lr_pred, squared=False)
mae_lr = mean_absolute_error(y_test, lr_pred)

mse_svm = mean_squared_error(y_test, svm_pred)
rmse_svm = mean_squared_error(y_test, svm_pred, squared=False)
mae_svm = mean_absolute_error(y_test, svm_pred)

mse_dt = mean_squared_error(y_test, dt_pred)
rmse_dt = mean_squared_error(y_test, dt_pred, squared=False)
mae_dt = mean_absolute_error(y_test, dt_pred)
```

```
mse_rf = mean_squared_error(y_test, rf_pred)

rmse_rf = mean_squared_error(y_test, rf_pred, squared=False)

mae_rf = mean_absolute_error(y_test, rf_pred)


print("Linear Regression:")

print("MSE:", mse_lr)

print("RMSE:", rmse_lr)

print("MAE:", mae_lr)


print("\nSupport Vector Machines (SVM):")

print("MSE:", mse_svm)

print("RMSE:", rmse_svm)

print("MAE:", mae_svm)


print("\nDecision Trees:")

print("MSE:", mse_dt)

print("RMSE:", rmse_dt)

print("MAE:", mae_dt)


print("\nRandom Forests:")

print("MSE:", mse_rf)

print("RMSE:", rmse_rf)

print("MAE:", mae_rf)


# Continuously update and improve the prediction models based on feedback and changing market conditions

# This could involve retraining the models with new data and fine-tuning hyperparameters
```

7th question:

7. Gather movie-related data from various sources, including movie databases (e.g., IMDb, TMDb), user ratings and reviews (e.g., from platforms like Netflix, Amazon Prime), and movie metadata (e.g., genre, release year, director).

a) Clean the raw data by handling missing values, removing duplicates, and correcting inconsistencies.

b) Transform the data into a suitable format for modelling, such as user-item interaction matrices or user-item feature matrices.

c) Create user and item embeddings using techniques like matrix factorization or deep learning embeddings.

d) Extract relevant features from the raw data that may influence movie recommendations, such as movie genres, cast and crew information, user demographics, and historical interactions.

e) Compute additional features like user similarity scores or movie similarity scores based on collaborative filtering or content-based methods.

f) Evaluate the performance of the trained models using appropriate evaluation metrics such as precision, recall, mean average precision, and ranking metrics (e.g., Normalized Discounted Cumulative Gain, Mean Reciprocal Rank).

g) Develop a classification/ clustering algorithm to group users with similar preferences for movies or TV shows in an online streaming platform.

h) How would you ensure scalability and real-time performance of the recommendation system as the user base grows?

Code:

```
import pandas as pd
from surprise import SVD
from surprise import Dataset, Reader
from surprise.model_selection import cross_validate
from surprise.accuracy import precision_at_k, recall_at_k
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Load Netflix ratings data
netflix_ratings = pd.read_csv('netflix_ratings.csv')

# Clean the raw data
netflix_ratings.dropna(inplace=True)
netflix_ratings.drop_duplicates(inplace=True)

# Correct inconsistencies if any


# Transform data into a suitable format for modeling

interaction_matrix = netflix_ratings.pivot(index='userID', columns='movieID', values='rating').fillna(0)


# Create user and item embeddings

reader = Reader(rating_scale=(1, 5))

data = Dataset.load_from_df(netflix_ratings[['userID', 'movieID', 'rating']], reader)

trainset = data.build_full_trainset()

model = SVD()

model.fit(trainset)


# Extract relevant features

# Extract movie genres, cast and crew information from IMDb and TMDb data (not included in this code)

# Extract user demographics from user ratings data (e.g., age, gender)

# Use historical interactions from user ratings data (already included in the interaction matrix)


# Compute additional features

user_similarity_matrix = cosine_similarity(interaction_matrix)

# Compute movie similarity scores based on movie features like genres, cast, and crew information (not included in this code)

# Use techniques like Jaccard similarity or cosine similarity


# Develop classification/clustering algorithm

kmeans = KMeans(n_clusters=5)

user_clusters = kmeans.fit_predict(interaction_matrix)




# Evaluate model performance

# Evaluate model using cross-validation

results = cross_validate(model, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)


# Calculate precision and recall at top k recommendations
```

```
k = 10  # Number of top recommendations
predictions = model.test(trainset.build_testset())
precision = precision_at_k(predictions, k=k, threshold=3.5)
recall = recall_at_k(predictions, k=k, threshold=3.5)

# Print evaluation results
print("Cross-validation results:")
for key, value in results.items():
    print(f"{key}: {value.mean()}")

print(f"Precision at {k}: {precision}")
print(f"Recall at {k}: {recall}")

# Visualization example
plt.hist(user_clusters, bins=5)
plt.title('User Clusters Distribution')
plt.xlabel('Cluster')
plt.ylabel('Number of Users')
plt.show()
```

8th questions:

. Create a classification model to identify faulty products in a manufacturing process based

on sensor data and quality control parameters.

a) Gather historical data related to the manufacturing process and product quality.

b) Collect data from various sources such as sensors, quality control reports,

maintenance records, and customer feedback.

c) Handle missing values through techniques such as imputation or deletion.

d) Normalize or scale the data to ensure consistent ranges and improve model

convergence.

e) Create derived features such as statistical aggregates, time-series features, or domain specific metrics.

f) Choose appropriate machine learning or statistical models for faulty product prediction,

such as classification algorithms (e.g., logistic regression, random forests, support

vector machines) or anomaly detection algorithms.

g) Evaluate the performance of the trained models using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, or area under the receiver operating characteristic curve (ROC-AUC).

h) How would you deploy and maintain the model in a production environment to minimize downtime and maximize productivity?

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# a) Gather historical data related to the manufacturing process and product quality.
manufacturing_data = pd.read_csv('manufacturing_data.csv')

# b) Collect data from various sources such as sensors, quality control reports, maintenance records, and customer feedback.

# c) Handle missing values through techniques such as imputation or deletion.
imputer = SimpleImputer(strategy='mean')
manufacturing_data_imputed = imputer.fit_transform(manufacturing_data)

# d) Normalize or scale the data to ensure consistent ranges and improve model convergence.
scaler = StandardScaler()
manufacturing_data_scaled = scaler.fit_transform(manufacturing_data_imputed)

# e) Create derived features such as statistical aggregates, time-series features, or domain-specific metrics.

# f) Choose appropriate machine learning or statistical models for faulty product prediction.
X = manufacturing_data_scaled[:, :-1]  # Features
y = manufacturing_data_scaled[:, -1]   # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train RandomForestClassifier
```

```
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)

# g) Evaluate the performance of the trained models using appropriate evaluation metrics.
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("Model Performance Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC-AUC score:", roc_auc)
```

9th question:

9. Suppose you are working for a social media platform. Design a clustering algorithm to segment users based on their behaviour, interests, and social interactions.

a) Perform the required pre-processing methods by selecting relevant features and scale them

b) Apply various Hierarchical and Density based Clustering methods

c) Visualize the clusters in 2D Plots using relevant feature set.

d) Compute evaluation metrics such as Cophenetic Correlation Coefficient, Silhouette Score, Hierarchical Clustering Consistency and Gap Statistics etc.

e) In addition to that, identify possible outliers among the data.

f) How would you leverage these user segments to enhance user engagement and targeted advertising

code:

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

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import AgglomerativeClustering, DBSCAN

from sklearn.metrics import silhouette_score

from scipy.cluster.hierarchy import dendrogram, linkage, cophenet

from sklearn.neighbors import LocalOutlierFactor


# Load user data

user_data = pd.read_csv('user_data.csv')


# a) Perform pre-processing: Select relevant features and scale them

relevant_features = user_data[['behavior_feature_1', 'behavior_feature_2', 'interests_feature_1', 'social_interaction_feature_1']]

scaler = StandardScaler()

scaled_features = scaler.fit_transform(relevant_features)


# b) Apply clustering methods

# Hierarchical clustering

hierarchical_clustering = AgglomerativeClustering(n_clusters=3)

hierarchical_labels = hierarchical_clustering.fit_predict(scaled_features)


# Density-based clustering

dbscan_clustering = DBSCAN(eps=0.5, min_samples=5)

dbscan_labels = dbscan_clustering.fit_predict(scaled_features)


# c) Visualize clusters in 2D plots

plt.figure(figsize=(12, 6))


plt.subplot(1, 2, 1)

plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=hierarchical_labels, cmap='viridis')

plt.title('Hierarchical Clustering')

plt.xlabel('Behavior Feature 1')

plt.ylabel('Behavior Feature 2')
plt.subplot(1, 2, 2)
plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=dbscan_labels, cmap='viridis')
plt.title('DBSCAN Clustering')
plt.xlabel('Behavior Feature 1')
plt.ylabel('Behavior Feature 2')

plt.show()

# d) Compute evaluation metrics
# Cophenetic Correlation Coefficient
Z = linkage(scaled_features, method='ward')
c, coph_dists = cophenet(Z, pdist(scaled_features))
print("Cophenetic Correlation Coefficient:", c)

# Silhouette Score
silhouette_hierarchical = silhouette_score(scaled_features, hierarchical_labels)
silhouette_dbscan = silhouette_score(scaled_features, dbscan_labels)
print("Silhouette Score (Hierarchical Clustering):", silhouette_hierarchical)
print("Silhouette Score (DBSCAN Clustering):", silhouette_dbscan)

# e) Identify outliers
lof = LocalOutlierFactor()
outlier_scores = lof.fit_predict(scaled_features)

outliers = scaled_features[outlier_scores == -1]
```

10th queston:

Design and develop a Heart disease detection system using following scenarios:

a) Perform the pre-processing techniques include handling missing values, encoding categorical variables, and scaling numerical features if necessary.

b) Build the model using multiple base classifiers that are diverse and complement each other's strengths and weaknesses.

c) Perform required feature selection methods to extract more discriminating feature set.

d) Combine the predictions of base classifiers using the chosen ensemble method (voting, bagging, boosting, or stacking).

 For voting ensembles, choose appropriate voting scheme (e.g., majority voting for classification).

 For bagging ensembles, aggregate predictions by averaging (for regression) or majority voting (for classification).

 For boosting ensembles, combine predictions using weighted averaging.

e) Evaluate the ensemble classification model's performance on the validation set using appropriate evaluation metrics, such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC).

f) Fine-tune hyperparameters of the ensemble model, such as the number of base classifiers, learning rates (for boosting), or the number of trees (for random forests).

g) Visualize the ensemble model's decision boundaries and feature importance to gain insights into the prediction process

code:

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
import numpy as np

# Load the dataset
heart_data = pd.read_csv('heart_disease_data.csv')

# a) Pre-processing
# Handling missing values
```

```
imputer = SimpleImputer(strategy='mean')

heart_data_imputed = imputer.fit_transform(heart_data)


# Encoding categorical variables

heart_data_encoded = pd.get_dummies(heart_data_imputed, columns=['categorical_column'])


# Scaling numerical features

scaler = StandardScaler()

heart_data_scaled = scaler.fit_transform(heart_data_encoded.iloc[:, :-1])


# Splitting data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(heart_data_scaled, heart_data_encoded['target'], test_size=0.2, random_state=42)


# b) Build the model using multiple base classifiers

base_classifiers = [

    ('rf', RandomForestClassifier()),

    ('gb', GradientBoostingClassifier()),

    ('dt', DecisionTreeClassifier()),

    ('lr', LogisticRegression())

]


# c) Feature selection

# No feature selection applied in this example


# d) Combine predictions of base classifiers using VotingClassifier

ensemble_model = VotingClassifier(estimators=base_classifiers, voting='soft')


# e) Evaluate ensemble classification model's performance

ensemble_model.fit(X_train, y_train)

y_pred = ensemble_model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)
```

```
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("Performance Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC-AUC score:", roc_auc)

# f) Fine-tune hyperparameters of the ensemble model
# No hyperparameter tuning applied in this example

# g) Visualize decision boundaries and feature importance
def plot_decision_boundary(X, y, model):
    feature_names = heart_data_encoded.columns[:-1]
    X_subset = X[:, :2]  # Considering only the first two features for visualization
    model.fit(X_subset, y)
    plot_decision_regions(X_subset, y, clf=model, legend=2)
    plt.xlabel(feature_names[0])
    plt.ylabel(feature_names[1])
    plt.title('Decision Boundary')
    plt.show()

plot_decision_boundary(X_train, y_train, ensemble_model)
feature_importance = ensemble_model.named_estimators_['rf'].feature_importances_
plt.bar(range(len(feature_importance)), feature_importance)
plt.xticks(range(len(feature_importance)), heart_data_encoded.columns[:-1], rotation=90)
plt.title('Feature Importance')
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.show()
```