import pandas as pd

path = 'customer\_segmentation\_data.csv'

df = pd.read\_csv(path)

df.head(20)

**stexcum enq papka vori mej qcum enq ev csv file-y ev bacum enq jupyteri file nuyn papki mej et depqum aranc path grelu read.csv-ov kardum a, df.read(20) arajin 20 toghn a tpum**

import requests

url = "https://realpython.github.io/fake-jobs/"

data = requests.get(url)

data

**kam ete url tvac lini senc enq import anum datan**

df.tail(20)

**verjin 20 toxn enq tpum**

df.info()

**informaciaya berum mer columneri u meji null arjeqneri mek el datatyperi masin**

df1 = df.dropna()

**ete null arjeqner enq unenum nor dataframe-i veragrum enq null-ery drop arac dataframe-y**

df['Occupation'].value\_counts()

**mer dataframe-I occupation columni mej exac amen tesaki info hashvuma, orinak 5 hat bjishk ka, 18 iravaban…**

df[['Occupation', 'Income Level', 'Customer ID']]

**tpuma menak es nshac columnery**

df.sample(20)

**random 20 toxa cuyc talis mer dataframe-ic**

df.iloc[0:15, [0, 1, 6, 10]]

**cuyca talis 0, 1, 6, 10 columneri 0-15 toxery**

df.query(‘Age>30’) or df.query(‘`Income Level`>100000’)

**cuyca talis 30ic mecerin menak kam nranc um ashxatavardzy 100000-ic shata**

df1.loc[:, 'Seat Comfort'] = df1['Seat Comfort'].astype(int)

**trvac column-i bolor toxery sarqum enq integer**

import matplotlib.pyplot as plt

plt.hist(df['Age'])

**histograma sarqum Age columni hamar**

df.describe()

**beruma tvayin columneri mijinnery, standard deviation, ev ayln**

df.hist(figsize = (10, 8))

plt.tight\_layout()

plt.show()

**beruma bolor tvayin columneri hamar histogramner**

import seaborn as sns

import matplotlib.pyplot as plt

sns.countplot(x='Occupation', data=df)

plt.xticks(rotation=90)

plt.show()

**Occupationi hamar u amen masnagitutyan qanaki histograma sarqum, x uxxi vra el gruma masnagitutyunnery 90 astichan teqac vor texavori**

sns.pairplot(df[['Age', 'Income Level', 'Coverage Amount']])

plt.show()

**3 columneri mijev bolor havanakan combinationnerob scatterplot u histograma sarqum**

sns.boxplot(data=df[['Occupation', 'Income Level']])

plt.show()

**boxplota sarqum grac erku columneri hamar**

numerical\_columns = df.select\_dtypes(include=['float64', 'int64']).columns

correlation\_matrix = df[numerical\_columns].corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.show()

**vercnuma menak float64 u int64 datatype-ery u drancov heatmap a sarqum**

plt.hist(df['Age'], bins=20)

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.title('Distribution of Age')

plt.show()

**eli histograma Age-I hamar vor y arancqy frequency-a isk x arancqy tariqna, u 20 hat bashnyaya ogtagorcum**

sns.scatterplot(x='Income Level', y='Coverage Amount', hue='Segmentation Group', data=df)

plt.xlabel('Income Level')

plt.ylabel('Coverage Amount')

plt.title('Income Level vs Coverage Amount')

plt.show()

**scatterplota sarqum vortex ogtagorcuma ham x u y arancqnery vorpes erku tarber columneri kaxvacutyuny iraric, bayc ham el gunayin tarberutyuna dnum keteri mijev yst 3rd columni**

df\_encoded = pd.get\_dummies(df, columns=['Gender', 'Marital Status', 'Education Level', 'Preferred Language'])

df\_encoded

**qani vor mer dataframe-um kan voch tvayin arjeqner iranc encode enq anum u nor dataframe enq sarqum, aysinqn liqy nor columner en avelanum voronq true en kam false**

df['Income x Coverage'] = df['Income Level'] \* df['Coverage Amount']

df['Income x Coverage']

**stex nor column enq stexcum vory urish erku columneri artadryalna, nuyny kara lini true u false unecox arjecneri hamar uxxaki columnery anpayman Boolean en linelu u bazmapatkman nshani texy & enq dnum**

bins = [0, 30, 60, float('inf')]

labels = ['Young', 'Adult', 'Elderly']

df['Age Group'] = pd.cut(df['Age'], bins=bins, labels=labels)

df['Age Group']

**nor column enq stexcum himnvelov Age-I vra, bajanum enq 3 masi 0-29, 30-59 u 60-ic anverj u dranc hamapatasxan anunner enq dnum**

df['Is\_Male'] = (df['Gender'] == 'Male').astype(int)

df['Is\_Male']

**nor column enq stexcum vortex genderum male-in 1 a talis isk female-in 0 (sexism☹)**

df['State'] = df['Geographic Information'].str.split(',').str[-1]

df['State']

**stex mardakanc aprelu vayry yst storaketi bajanuma u vercnuma storaketic heton et [-1] et hatvacna, nuyn dzev inch vor tarov karanq bajanenq**

df['Simplified\_Occupation'] = df['Occupation'].where(df['Occupation'].isin(['Doctor', 'Engineer', 'Teacher']), 'Other')

df['Simplified\_Occupation']

**nor column a stexcvum vortex qani vor occupationnery shat en, vercrel enq nranq voronq mez petq en dranq toxum enq, mnacacin talis enq other arjeqy**

df['Has\_Purchased\_Insurance'] = (df['Purchase History'] != '').astype(int)

df['Has\_Purchased\_Insurance']

**nor columna stexcvum vortex en mardik ovqer purchase history chunen 0 en stanum, isk ovqer unen 1**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

for col in df.select\_dtypes(include=['object']).columns:

df[col] = df[col].fillna('missing')

label\_encoder = LabelEncoder()

df\_encoded = df.apply(label\_encoder.fit\_transform)

X = df\_encoded.drop(columns=['Behavioral Data']) # Assuming 'Customer ID' is not a feature

y = df\_encoded['Behavioral Data']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Lracnuma datark vandaknery dataframe-i meji texy gruma missing, label encodera ogtagorcum vor voch tvayin tvyalnery karoxana ogtagorci u bajanuma datasety training u testing hatvacneri**

# Initialize classifiers

decision\_tree = DecisionTreeClassifier()

knn = KNeighborsClassifier()

svm = SVC()

naive\_bayes = GaussianNB()

gbm = GradientBoostingClassifier()

neural\_network = MLPClassifier()

logistic\_reg = LogisticRegression(max\_iter=6000, solver='liblinear', C=0.1, penalty='l2')

**Classification anelu tarber metodnery import aneluc heto variable-ner enq dnum vor dranc hamapatsxanen**

# Train classifiers

decision\_tree.fit(X\_train, y\_train)

knn.fit(X\_train, y\_train)

svm.fit(X\_train, y\_train)

naive\_bayes.fit(X\_train, y\_train)

gbm.fit(X\_train, y\_train)

neural\_network.fit(X\_train, y\_train)

logistic\_reg.fit(X\_train, y\_train)

**tarber metodnerov train enq anum datasety**

# Predictions

decision\_tree\_preds = decision\_tree.predict(X\_test)

knn\_preds = knn.predict(X\_test)

svm\_preds = svm.predict(X\_test)

naive\_bayes\_preds = naive\_bayes.predict(X\_test)

gbm\_preds = gbm.predict(X\_test)

neural\_network\_preds = neural\_network.predict(X\_test)

logistic\_reg\_preds = logistic\_reg.predict(X\_test)

**predict enq anum tarber metodnerov**

# Evaluate accuracy

decision\_tree\_accuracy = accuracy\_score(y\_test, decision\_tree\_preds)

knn\_accuracy = accuracy\_score(y\_test, knn\_preds)

svm\_accuracy = accuracy\_score(y\_test, svm\_preds)

naive\_bayes\_accuracy = accuracy\_score(y\_test, naive\_bayes\_preds)

gbm\_accuracy = accuracy\_score(y\_test, gbm\_preds)

neural\_network\_accuracy = accuracy\_score(y\_test, neural\_network\_preds)

logistic\_reg\_accuracy = accuracy\_score(y\_test, logistic\_reg\_preds)

**accuracy enq hashvum vor tesnenq vor metodna amena moty 1in**

print("Decision Tree Accuracy: ", decision\_tree\_accuracy)

print("KNN Accuracy: ", knn\_accuracy)

print("SVM Accuracy: ", svm\_accuracy)

print("Naive Bayes Accuracy: ", naive\_bayes\_accuracy)

print("Gradient Boosting Accuracy: ", gbm\_accuracy)

print("Neural Network Accuracy: ", neural\_network\_accuracy)

print("Logistic Regression Accuracy", logistic\_reg\_accuracy)

**tpum enq accuracy-nery**

import warnings

from sklearn.metrics import recall\_score, precision\_score, f1\_score, roc\_auc\_score

from sklearn.exceptions import UndefinedMetricWarning

# Suppress warnings

warnings.filterwarnings("ignore", category=UndefinedMetricWarning)

decision\_tree\_recall = recall\_score(y\_test, decision\_tree\_preds, average='weighted')

decision\_tree\_precision = precision\_score(y\_test, decision\_tree\_preds, average='weighted')

decision\_tree\_f1 = f1\_score(y\_test, decision\_tree\_preds, average='weighted')

decision\_tree\_roc\_auc = roc\_auc\_score(y\_test, decision\_tree.predict\_proba(X\_test), multi\_class='ovr')

print("Decision Tree Recall:", decision\_tree\_recall)

print("Decision Tree Precision:", decision\_tree\_precision)

print("Decision Tree F1 Score:", decision\_tree\_f1)

print("Decision Tree ROC AUC:", decision\_tree\_roc\_auc)

**decision tree-ov hashvum enq scorer**

knn\_recall = recall\_score(y\_test, knn\_preds, average='weighted')

knn\_precision = precision\_score(y\_test, knn\_preds, average='weighted')

knn\_f1 = f1\_score(y\_test, knn\_preds, average='weighted')

knn\_roc\_auc = roc\_auc\_score(y\_test, knn.predict\_proba(X\_test), multi\_class='ovr')

print("KNN Recall:", knn\_recall)

print("KNN Precision:", knn\_precision)

print("KNN F1 Score:", knn\_f1)

print("KNN ROC AUC:", knn\_roc\_auc)

**knn-i hamar hashvuma nuyn scorery**

svm\_recall = recall\_score(y\_test, svm\_preds, average='weighted')

svm\_precision = precision\_score(y\_test, svm\_preds, average='weighted')

svm\_f1 = f1\_score(y\_test, svm\_preds, average='weighted')

#svm\_roc\_auc = roc\_auc\_score(y\_test, svm.decision\_function(X\_test), average='weighted', multi\_class='ovr')

print("SVM Recall:", svm\_recall)

print("SVM Precision:", svm\_precision)

print("SVM F1 Score:", svm\_f1)

#print("SVM ROC AUC: is missing here")

**Svm-i hamar hashvuma nuyn scorery, stex roc-auc inch vor xndir er tvel u konkret es depqi hamar cher linum hashvel et**

naive\_bayes\_recall = recall\_score(y\_test, naive\_bayes\_preds, average='weighted')

naive\_bayes\_precision = precision\_score(y\_test, naive\_bayes\_preds, average='weighted')

naive\_bayes\_f1 = f1\_score(y\_test, naive\_bayes\_preds, average='weighted')

naive\_bayes\_roc\_auc = roc\_auc\_score(y\_test, naive\_bayes.predict\_proba(X\_test), multi\_class='ovr')

print("Naive Bayes Recall:", naive\_bayes\_recall)

print("Naive Bayes Precision:", naive\_bayes\_precision)

print("Naive Bayes F1 Score:", naive\_bayes\_f1)

print("Naive Bayes ROC AUC:", naive\_bayes\_roc\_auc)

**nuyny naïve bayes-i hamar**

gbm\_recall = recall\_score(y\_test, gbm\_preds, average='weighted')

gbm\_precision = precision\_score(y\_test, gbm\_preds, average='weighted')

gbm\_f1 = f1\_score(y\_test, gbm\_preds, average='weighted')

gbm\_roc\_auc = roc\_auc\_score(y\_test, gbm.predict\_proba(X\_test), multi\_class='ovr')

print("GBM Recall:", gbm\_recall)

print("GBM Precision:", gbm\_precision)

print("GBM F1 Score:", gbm\_f1)

print("GBM ROC AUC:", gbm\_roc\_auc)

**nuyny gbm-i hamar**

neural\_network\_recall = recall\_score(y\_test, neural\_network\_preds, average='weighted')

neural\_network\_precision = precision\_score(y\_test, neural\_network\_preds, average='weighted')

neural\_network\_f1 = f1\_score(y\_test, neural\_network\_preds, average='weighted')

neural\_network\_roc\_auc = roc\_auc\_score(y\_test, neural\_network.predict\_proba(X\_test), multi\_class='ovr')

print("Neural Network Recall:", neural\_network\_recall)

print("Neural Network Precision:", neural\_network\_precision)

print("Neural Network F1 Score:", neural\_network\_f1)

print("Neural Network ROC AUC:", neural\_network\_roc\_auc)

**nuyn neural network-i hamar**

logistic\_reg\_recall = recall\_score(y\_test, logistic\_reg\_preds, average='weighted')

logistic\_reg\_precision = precision\_score(y\_test, logistic\_reg\_preds, average='weighted')

logistic\_reg\_f1 = f1\_score(y\_test, logistic\_reg\_preds, average='weighted')

logistic\_reg\_roc\_auc = roc\_auc\_score(y\_test, logistic\_reg.predict\_proba(X\_test), multi\_class='ovr')

print("Logistic Regression Recall:", logistic\_reg\_recall)

print("Logistic Regression Precision:", logistic\_reg\_precision)

print("Logistic Regression F1 Score:", logistic\_reg\_f1)

print("Logistic Regression ROC AUC:", logistic\_reg\_roc\_auc)

**nuyny logistic regression hamar**

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

# Initialize regression models

linear\_reg = LinearRegression()

decision\_tree\_reg = DecisionTreeRegressor()

random\_forest\_reg = RandomForestRegressor()

# Train regression models

linear\_reg.fit(X\_train, y\_train)

decision\_tree\_reg.fit(X\_train, y\_train)

random\_forest\_reg.fit(X\_train, y\_train)

# Predictions

linear\_reg\_preds = linear\_reg.predict(X\_test)

decision\_tree\_reg\_preds = decision\_tree\_reg.predict(X\_test)

random\_forest\_reg\_preds = random\_forest\_reg.predict(X\_test)

# Evaluate performance using mean squared error

linear\_reg\_mse = mean\_squared\_error(y\_test, linear\_reg\_preds)

decision\_tree\_reg\_mse = mean\_squared\_error(y\_test, decision\_tree\_reg\_preds)

random\_forest\_reg\_mse = mean\_squared\_error(y\_test, random\_forest\_reg\_preds)

**nuyny araca regression-eri hamar, import enq anum, initialize enq anum, train heto test u mse enq hashvum**

# Print mean squared error for each model

print("Linear Regression MSE:", linear\_reg\_mse)

print("Decision Tree Regression MSE:", decision\_tree\_reg\_mse)

print("Random Forest Regression MSE:", random\_forest\_reg\_mse)

**tpum enq mse scorery regression-i tarber tesakneri hamar**

from sklearn.metrics import r2\_score

# Calculate additional evaluation metrics

linear\_reg\_r2 = r2\_score(y\_test, linear\_reg\_preds)

decision\_tree\_reg\_r2 = r2\_score(y\_test, decision\_tree\_reg\_preds)

random\_forest\_reg\_r2 = r2\_score(y\_test, random\_forest\_reg\_preds)

# Print additional evaluation metrics

print("Linear Regression R^2 Score:", linear\_reg\_r2)

print("Decision Tree Regression R^2 Score:", decision\_tree\_reg\_r2)

print("Random Forest Regression R^2 Score:", random\_forest\_reg\_r2)

**et nuyn regression-i tesakneri hamar R^2 score enq hashvum**

from sklearn.model\_selection import GridSearchCV

# Parameter grid for Linear Regression

linear\_param\_grid = {} # Linear regression does not have hyperparameters to tune

# GridSearchCV for Linear Regression

linear\_grid\_search = GridSearchCV(estimator=linear\_reg, param\_grid=linear\_param\_grid, cv=10, scoring='neg\_mean\_squared\_error')

linear\_grid\_search.fit(X\_train, y\_train)

print("Linear Regression - Best Parameters:", linear\_grid\_search.best\_params\_)

print("Linear Regression - Best Score:", -linear\_grid\_search.best\_score\_)

**grid search enq anum linear regression hamar vor best model gtnenq**

# Parameter grid for Decision Tree Regression

decision\_tree\_param\_grid = {'max\_depth': [None, 10, 20, 30]}

decision\_tree\_grid\_search = GridSearchCV(estimator=decision\_tree\_reg, param\_grid=decision\_tree\_param\_grid, cv=10, scoring='neg\_mean\_squared\_error')

decision\_tree\_grid\_search.fit(X\_train, y\_train)

print("Decision Tree Regression - Best Parameters:", decision\_tree\_grid\_search.best\_params\_)

print("Decision Tree Regression - Best Score:", -decision\_tree\_grid\_search.best\_score\_)

**grid search enq anum decision tree-i hamar regression-um**

# Parameter grid for Random Forest Regression

random\_forest\_param\_grid = {'n\_estimators': [50, 100, 200], 'max\_depth': [None, 10, 20, 30]}

random\_forest\_grid\_search = GridSearchCV(estimator=random\_forest\_reg, param\_grid=random\_forest\_param\_grid, cv=10, scoring='neg\_mean\_squared\_error')

random\_forest\_grid\_search.fit(X\_train, y\_train)

print("Random Forest Regression - Best Parameters:", random\_forest\_grid\_search.best\_params\_)

print("Random Forest Regression - Best Score:", -random\_forest\_grid\_search.best\_score\_)

**random forest-i hamar enq grid search anum**

# Define parameter grid for Decision Tree

decision\_tree\_param\_grid = {'max\_depth': [None, 10, 20, 30]}

decision\_tree\_grid\_search = GridSearchCV(estimator=decision\_tree, param\_grid=decision\_tree\_param\_grid, cv=10, scoring='accuracy')

decision\_tree\_grid\_search.fit(X\_train, y\_train)

print("Decision Tree - Best Parameters:", decision\_tree\_grid\_search.best\_params\_)

print("Decision Tree - Best Score:", decision\_tree\_grid\_search.best\_score\_)

**grid search decision tree-i hamar classificationum**

# Define parameter grid for KNN

knn\_param\_grid = {'n\_neighbors': [3, 5, 7]}

knn\_grid\_search = GridSearchCV(estimator=knn, param\_grid=knn\_param\_grid, cv=10, scoring='accuracy')

knn\_grid\_search.fit(X\_train, y\_train)

print("KNN - Best Parameters:", knn\_grid\_search.best\_params\_)

print("KNN - Best Score:", knn\_grid\_search.best\_score\_)

grid search knn-i hamar

# Define parameter grid for SVM

svm\_param\_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}

svm\_grid\_search = GridSearchCV(estimator=svm, param\_grid=svm\_param\_grid, cv=10, scoring='accuracy')

svm\_grid\_search.fit(X\_train, y\_train)

print("SVM - Best Parameters:", svm\_grid\_search.best\_params\_)

print("SVM - Best Score:", svm\_grid\_search.best\_score\_)

**grid search svm-i hamar**

# Define parameter grid for Naive Bayes

naive\_bayes\_param\_grid = {}

naive\_bayes\_grid\_search = GridSearchCV(estimator=naive\_bayes, param\_grid=naive\_bayes\_param\_grid, cv=10, scoring='accuracy')

naive\_bayes\_grid\_search.fit(X\_train, y\_train)

print("Naive Bayes - Best Parameters:", naive\_bayes\_grid\_search.best\_params\_)

print("Naive Bayes - Best Score:", naive\_bayes\_grid\_search.best\_score\_)

**grid search naïve bayes-i hamar**

# Define parameter grid for Gradient Boosting

gbm\_param\_grid = {'n\_estimators': [50, 100, 200], 'learning\_rate': [0.01, 0.1, 0.5]}

gbm\_grid\_search = GridSearchCV(estimator=gbm, param\_grid=gbm\_param\_grid, cv=10, scoring='accuracy')

gbm\_grid\_search.fit(X\_train, y\_train)

print("GBM - Best Parameters:", gbm\_grid\_search.best\_params\_)

print("GBM - Best Score:", gbm\_grid\_search.best\_score\_)

**grid search gbm-i hamar**

# Define parameter grid for Neural Network

neural\_network\_param\_grid = {'hidden\_layer\_sizes': [(50,), (100,), (50, 50)], 'alpha': [0.0001, 0.001, 0.01]}

neural\_network\_grid\_search = GridSearchCV(estimator=neural\_network, param\_grid=neural\_network\_param\_grid, cv=10, scoring='accuracy')

neural\_network\_grid\_search.fit(X\_train, y\_train)

print("Neural Network - Best Parameters:", neural\_network\_grid\_search.best\_params\_)

print("Neural Network - Best Score:", neural\_network\_grid\_search.best\_score\_)

**grid search neural network-i hamar**

# Define parameter grid for Logistic Regression

logistic\_reg\_param\_grid = {'C': [0.011, 0.1, 0.01, 3, 5, 25], 'penalty': ['l1', 'l2'], 'solver': ['liblinear']}

logistic\_reg\_grid\_search = GridSearchCV(estimator=logistic\_reg, param\_grid=logistic\_reg\_param\_grid, cv=10, scoring='accuracy')

logistic\_reg\_grid\_search.fit(X\_train, y\_train)

print("Logistic Regression - Best Parameters:", logistic\_reg\_grid\_search.best\_params\_)

print("Logistic Regression - Best Score:", logistic\_reg\_grid\_search.best\_score\_)

**grid search logistic regression-i hamar**

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import numpy as np

# Drop non-numeric columns

df\_numeric = df.select\_dtypes(include=['int', 'float'])

# Encode categorical variables if any

for col in df.select\_dtypes(include=['object']).columns:

label\_encoder = LabelEncoder()

df\_numeric[col] = label\_encoder.fit\_transform(df[col])

# Standardize the data

scaler = StandardScaler()

df\_scaled = scaler.fit\_transform(df\_numeric)

# Apply KMeans clustering

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(df\_scaled)

# Visualize the clusters or perform further analysis

print(df['Cluster'].value\_counts())

# Set the number of samples you want to visualize (e.g., 1000)

num\_samples = 100

# Randomly select num\_samples data points

sample\_indices = np.random.choice(len(df\_scaled), num\_samples, replace=False)

sampled\_data = df\_scaled[sample\_indices]

sampled\_labels = df['Cluster'].iloc[sample\_indices]

# Scatter plot of the sampled data with cluster color coding

plt.scatter(sampled\_data[:, 0], sampled\_data[:, 1], c=sampled\_labels, cmap='viridis')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('KMeans Clustering (Sampled)')

plt.colorbar(label='Cluster')

plt.show()

**bajanum enq dataset-i 100 sample 3 cluster-neri u scatterplot-ov cuyc enq talis k-means enq ogtagorcum**

from sklearn.datasets import make\_blobs

# Generate synthetic data using make\_blobs

X\_synthetic, \_ = make\_blobs(n\_samples=100, centers=3, n\_features=2, random\_state=42)

# Apply KMeans clustering

kmeans\_synthetic = KMeans(n\_clusters=3, random\_state=42)

y\_synthetic = kmeans\_synthetic.fit\_predict(X\_synthetic)

# Visualize the clusters

plt.scatter(X\_synthetic[:, 0], X\_synthetic[:, 1], c=y\_synthetic, cmap='viridis')

plt.title('Clustering with make\_blobs')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

**clusterneri enq bajanum voronq xmbavorum enq kopit asac 3 clusterov u scatterplotov cuyc enq talis, ogtagorcum enq make\_blobs**