1) sentiment analysis

```
from textblob import TextBlob

tex = """This world is cruel
And yet, I still wuv you
Even if I sacrifice everything
I'll still protect you
Even if I'm wrong
I don't doubt myself
Because what's right
Is to firmly believe in yourself"""

text = TextBlob(tex)

sent=text.sentiment.polarity

if sent > 0:
    print(f"Positive Sentence with a polarity of {sent}")
elif sent < 0:
    print(f"Negative Sentence with a polarity of {sent}")
else:
    print(f"Neutral Statement with a polarity of {sent}")
```

```
35]  ✓ 0.0s

Negative Sentence with a polarity of -0.3535714285714286
```

2) Develop a data filter solution to extract relevant information from the data set

This is a simple Data Science code to get insights from dataset similar to data visualization

Use any dataset you like to work with. for example, if you have dataset about Anime ratings then you can get insights such as Most Watched Show, Favourite Protoganist, Best Openings, Most Watched Show outside japan

modules you'll be needing pandas,numpy,matplotlib

use pandas to read and clean the dataset

use numpy to get mean median std

Example program :

CSV file named 'data.csv' should contain a column named 'text' that represents the data to be filtered and a column named 'label' that indicates the relevance of the data.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC

# Load data
data = pd.read_csv('data.csv')  # Assuming your data is stored in a CSV file

# Separate features and labels
X = data['text']  # Features
y = data['label']  # Labels

# Extract features using TF-IDF vectorization
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(X)
# Train the classifier
classifier = LinearSVC()
classifier.fit(X, y)
# Define a function to filter relevant data
def filter_data(text):
    features = vectorizer.transform([text])
    prediction = classifier.predict(features)
    return prediction[0]
# Test the filter function
example_text = "This is an example text. It contains relevant information."
filtered_label = filter_data(example_text)
print("Filtered Label:", filtered_label)
```

3) encryption

```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier

# Function to encrypt a message using Random Forest encryption
def encrypt_message(message, n_estimators=10):
    # Convert the message into numerical values
    numerical_message = [ord(char) for char in message]

    # Create the feature matrix and target vector
    X = np.array(numerical_message).reshape(-1, 1)
    y = np.random.randint(256, size=len(numerical_message))

    # Train a Random Forest classifier to learn the encryption
    rf = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
    rf.fit(X, y)

    # Encrypt the message by predicting the target values
    encrypted_message = rf.predict(X)

    # Convert the encrypted numerical values back to characters
    encrypted_message = ''.join([chr(num) for num in encrypted_message])

    return encrypted_message

# Example usage

message = input("enter the text :")
encrypted_message = encrypt_message(message)
print("Encrypted message:", encrypted_message)
```

The encrypt_message function takes a message as input and converts it into numerical values using the 'ord' function. It then creates a feature matrix " X " and a target vector " y ". The Random Forest classifier is trained on the feature matrix "X"and target vector "y "using the 'fit' method

Or refer

https://github.com/snowflowerinstitute/AD8612-SRP-Lab/tree/master/Encryption

4) fraud dection

```
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder
data = pd.read_csv("creditCard.csv")
#data = data.dropna()
data['Class'].isna().any()
total_transactions = len(data)
normal = len(data[data.Class == 0])
fraudulent = len(data[data.Class == 1])
fraud_percentage = round(fraudulent / normal * 100, 2)
print(f'Total number of Transactions are {total_transactions}')
print(f'Number of Normal Transactions are {normal}')
print(f'Number of fraudulent Transactions are {fraudulent}')
print(f'Percentage of fraud Transactions is {fraud_percentage * 100}%')
X = data.drop('Class', axis=1).values
y = data["Class"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=1)
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)

print(f'Accuracy score of the XGBoost model is {(accuracy_score(y_test,
xgb_yhat) * 100):.2f} %')
print(f'F1 score of the XGBoost model is {(f1_score(y_test, xgb_yhat) *
100):.2f} %')
```

5) Data segmentation

```
import random
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Generate random data
num_samples = 1000
data = []
for _ in range(num_samples):
    length = random.randint(5, 20)
    string = ''.join(random.choices('abcdefghijklmnopqrstuvwxyz', k=length))
    data.append(string)

# Extract features using TF-IDF vectorization
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(data)

# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
reduced_features = pca.fit_transform(features.toarray())

# Apply K-means clustering
k = 3  # Number of clusters
kmeans = KMeans(n_clusters=k)
kmeans.fit(reduced_features)

# Visualize clusters
colors = ['r', 'g', 'b']  # Define colors for different clusters
plt.figure(figsize=(8, 6))
for i, point in enumerate(reduced_features):
    cluster_label = kmeans.labels_[i]
    plt.scatter(point[0], point[1], c=colors[cluster_label])

plt.title('Data Segmentation')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

6) Data Enrichment

```
[ ]  !pip install ydata-profiling

[5]  from ydata_profiling import ProfileReport
     import pandas as pd
     import warnings
     warnings.filterwarnings("ignore")

[8]  df = pd.read_csv("animes.csv")
     df = df.dropna()
     df['title'].head()

...  0              Haikyuu!! Second Season
     1              Shigatsu wa Kimi no Uso
     2                        Made in Abyss
     3    Fullmetal Alchemist: Brotherhood
     4    Kizumonogatari III: Reiketsu-hen
     Name: title, dtype: object

[7]  report = ProfileReport(df, title="Anime Enrichment Report")
     report.to_notebook_iframe()

...  Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

</>  Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

</>  Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

it is used to enhance the existing dataset to show more deeper insights about the we have `

Once You run this program a HTML will render in your Jupyter prompt in that you can see all the detailed insights of the dataset we have

7) **Python code that creates a model and makes predictions using multiple machine learning algorithms using the scikit-learn library**

In this code, we first import the necessary modules from scikit-learn. We define our sample data, X representing the input features and y representing the target variable. We then split the data into training and testing sets using train_test_split.

Next, we create a list of models, including Linear Regression, Decision Tree Regressor, Random Forest Regressor, and Support Vector Regressor. We iterate over each model, train it on the training data, and make predictions on the testing data. We calculate the mean squared error (MSE) as an evaluation metric for each model.

Finally, we select the best model (in this case, Random Forest Regressor) and use it to make predictions on new data new_data. The predictions are stored in the predictions variable.

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
# Sample data
X = [[1], [2], [3], [4], [5]]  # Input features
y = [2, 4, 6, 8, 10]           # Target variable
# Split the data 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)
# Create a list of models
models = [
    LinearRegression(),
    DecisionTreeRegressor(),
    RandomForestRegressor(),
```

```
    SVR()
]
# Train and evaluate each model
for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print("Model:", type(model).__name__)
    print("Mean Squared Error:", mse)
    print()
# Make predictions using the best model
best_model = RandomForestRegressor()
best_model.fit(X, y)
new_data = [[6], [7], [8]]  # New data to predict
predictions = best_model.predict(new_data)
print("Predictions:", predictions)
```

8) Python code that uses multiple machine learning algorithms for data visualization using the matplotlib library:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Generate synthetic classification data
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2,
n_redundant=0,
                           n_clusters_per_class=1, random_state=42)

# Create a list of classifiers
classifiers = [
    LogisticRegression(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    SVC()
]
```

```
# Plot the decision boundary for each classifier
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

for clf, ax in zip(classifiers, axes.flatten()):
    clf.fit(X, y)
    h = 0.02  # step size in the mesh

    # Create a grid of points to evaluate the classifier
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,
h))

    # Predict the class for each point in the grid
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot the decision boundary
    ax.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu, alpha=0.8)

    # Plot the training points
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu, edgecolors='k')
    ax.set_title(type(clf).__name__)

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

In this code, we first import the necessary modules, including numpy, matplotlib, and various classifiers from scikit-learn. We generate synthetic classification data using the make_classification function.

Next, we create a list of classifiers, including Logistic Regression, Decision Tree Classifier, Random Forest Classifier, and Support Vector Classifier. We iterate over each classifier and fit it to the data. We then create a grid of points to evaluate the classifier's decision boundary and predict the class for each point in the grid.

Finally, we plot the decision boundary and training points for each classifier using the contourf and scatter functions from matplotlib. The resulting plots are arranged in a 2x2 grid using subplots.