Who Is The Killer?

Piraeus Vice Pattern Recognition Project

Student 1: Βασίλαινα Μαρία (ID: Π23015)
Student 2: Γρηγοράσκου Τερέζα (ID: Π22037)
Student 3: Λιβέρης Φώτης (ID: Π23104)

Department of Informatics
University of Piraeus

February 20, 2026

Abstract

In this project, we addressed the killer identification problem as a multiclass classification task. We first estimated class-specific means and covariances using Maximum Likelihood Estimation (MLE), and used them to build a Gaussian Bayes classifier. We then implemented a linear classifier, a non-linear SVM (RBF kernel), a two-hidden-layer MLP neural network, and an unsupervised PCA + k-means approach with majority-vote label mapping. All preprocessing (standardization and one-hot encoding) was fitted on the TRAIN split and applied to VAL and TEST to avoid data leakage. PCA was used both for visualization and as a preprocessing step for k-means. On the validation set, non-linear models (SVM and MLP) achieved the best performance, outperforming the Gaussian Bayes and linear classifiers. The PCA + k-means approach achieved lower accuracy, indicating that natural clusters in feature space do not perfectly align with killer identities. Overall, the features that proved to be of the most importance were those related with the victim and the crime scene. Last but not least, modeling nonlinear decision boundaries significantly improved classification performance.

Collectively, in order to classify data and reach our final prediction we followed the following steps.

- MLE (Maximum Likelihood Estimation)
- Gaussian Bayes classifier
- Linear classifier
- SVM (Sypport Vector Machine)
- MLP (Multi-Layer Preceptron)
- PCA (Principal Component Analysis)
- k-means

Contents

1 Introduction

In the current project we were asked to identify the most likely killer for each crime incident in the "Piraeus Vice" dataset. This is formulated as a multiclass classification problem, where each of the incidents must be related to one of the 8 possible killers based on the available features. So, the task is to build models that can accurately predict the killer ID.

Each row in the dataset corresponds to a single crime incident and contains the following features:

- The hour that the crime was commited
- The latitude and the lotitude, together forming the coordinates of the crime scene
- The victim's age
- The prevailing temperature at the time the crime was commited
- The prevailing humidity at the time the crime took place
- The distance between the crime scene and the nearest police precinct
- The population density of the area
- The weapon type (knife, handgun 9mm, revolver .38, shotgun, blunt object, unknown)
- The scene type (street, residence, business, other)
- The prevailing weather conditions (clear, rain, snow, fog, unknown)
- The victim gender (male, female)
- The incident ID
- The partition type (TRAIN, VAL, TEST)
- The killer ID (1 ... 8)

Comment: Further analysis will follow in section 2.

The main steps of analysis we are instructed to use are:

- Q1: Exploratory Data Analysis
- Q2: Parameter Estimation (MLE)
- Q3: Gaussian Bayes Classifier
- Q4: Linear Classifier
- Q5: Support Vector MAchine (SVM)
- Q6: Multilayer Perceptron (MLP)
- Q7: Principal Component Analysis (PCA)
- Q8: PCA + k-means

The following picture shows a high-level overview of the killer identification process.

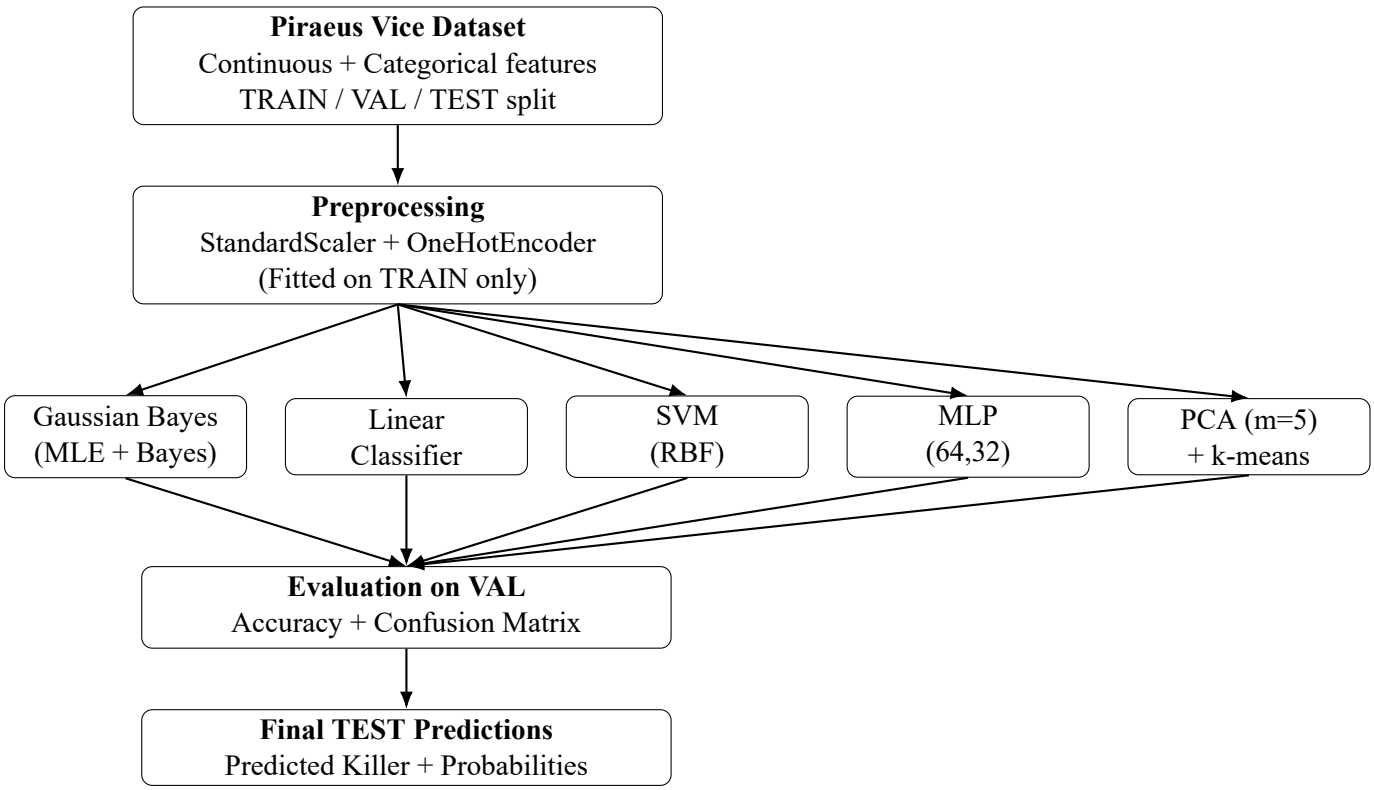


Figure 1: High-level overview of the killer identification pipeline.

2 Data description

The csv file we were given consists of a total of 4.800 rows, each one representing one crime incident. As we have already mentioned, it consists of 15 columns, one for each feature. An overview of these features and their corresponding types follows in section 2.1. The dataset contains a predefined split column that is either TRAIN, VAL or TEST. This means that the data is already partitioned for modeling and evaluation. As far as we can tell the dataset is not standardised or normalised.

2.1 Feature overview

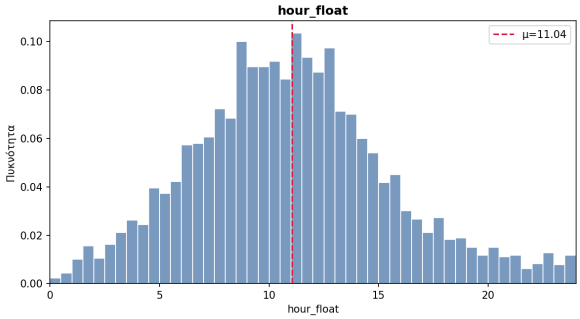
Table 1: Feature summary.

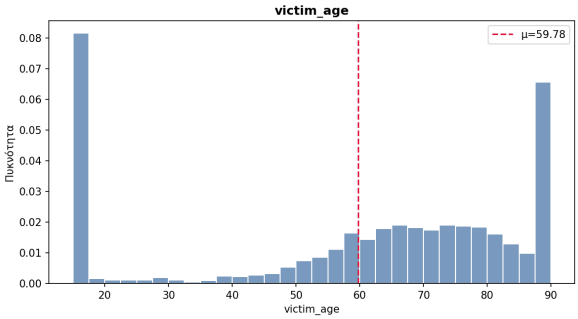
Feature name	Type
incident_id	identifier
hour_float	continuous
latitude	continuous
longitude	continuous
victim_age	continuous
temp_c	continuous
humidity	continuous
dist_precinct_km	continuous
pop_density	continuous
weapon_code	categorical
scene_type	categorical
weather	categorical
vic_gender	categorical (binary)
split	categorical
killer_id	identifier

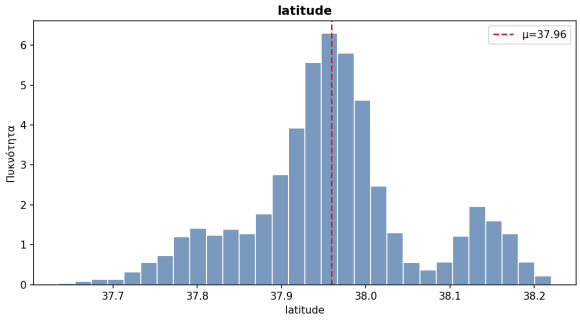
3 Results

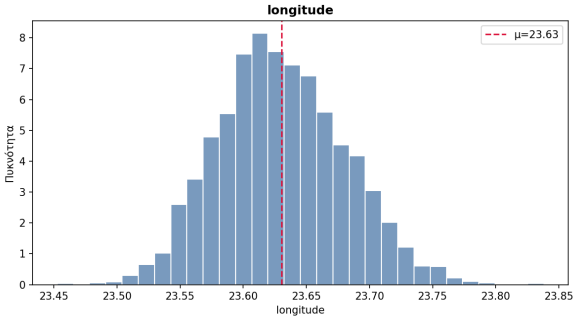
Q1: Exploratory analysis

The images that follow are the histograms for the key continuous variables (hour, age, latitude, longitude) that we produced.

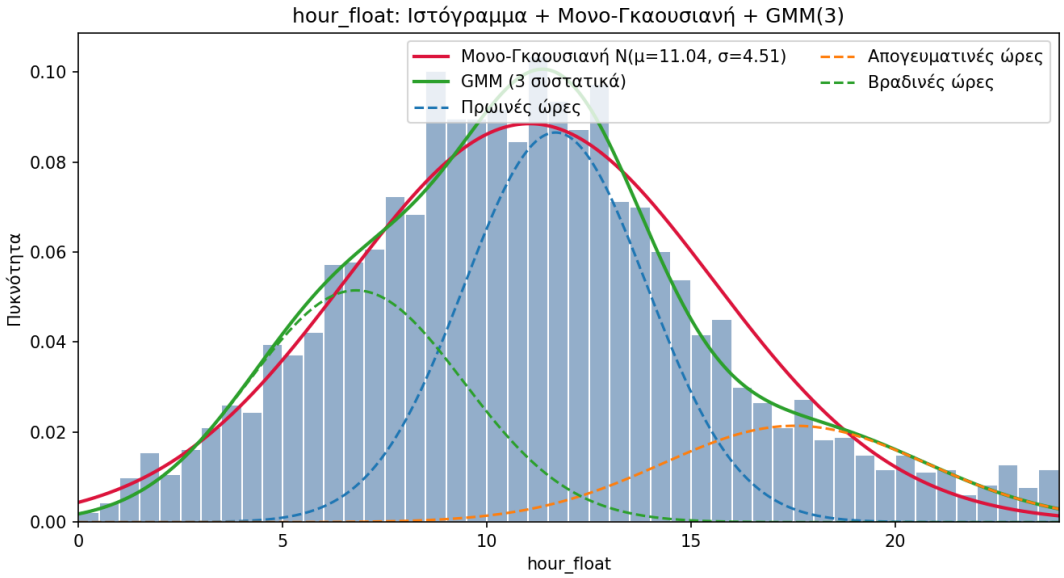




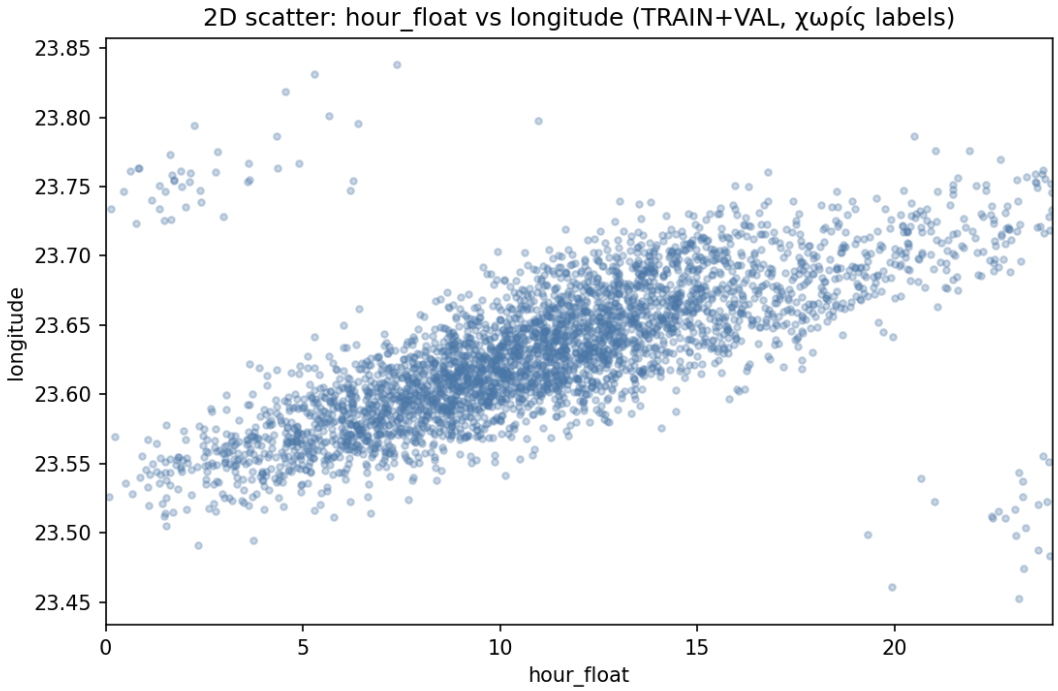




Below is the comparison of the single Gaussian fit and a mixture of Gaussians for `hour_float`.



Also, we provide you with the additional two-dimensional plot involving `hour_float` and `longtitude`. using TRAIN+VAL and without using any labels, as requested.



Our observations around the 4 key continuous variables are the following:

- Firstly, about the variable `hour_float` the incidents are at their peak during daytime, which is somewhat unexpected.
- As far as the variable `victim_age` is concerned it can easily be observed that the selected victims were either of a very young age or of a very old. With that said, we can safely come to the conclusion that they were targeted because they belonged to the most vulnerable age groups.
- The histogram of the variable `latitude` presents some fluctuations that are at their core closer to its average value.
- Lastly, concerning the variable `longtitude`, by observing both the corresponding histogram as well as the 2D scatter, it is clear that the incident density is also thicker closer to its average value.

Regarding the single Gaussian, we believe that it is inadequate. We can easily see that it is not symmetric. But it is not solely that. It also ignores some of the peaks. As a result it fails to represent correctly the given data. By splitting it to the 3-component Gaussian Mixture we achieve better interpretation of the times of the day as well as better representation of the data as it focuses on distinct times of the day.

Explanation

We have decided to use the library GaussianMixture that proved to be very usefull. Also, we created the following function in order to be able to create the histograms. As we can see it requires 5 parameters, 3 of which have default values in case they don't receive any. The parameters are:

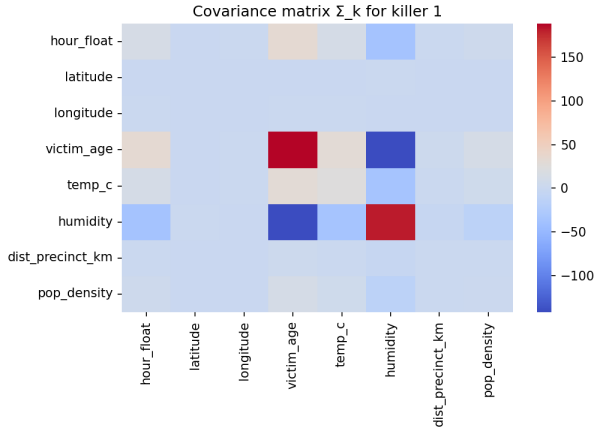
- series: a pandas series containing numerical data,
- title: it describes the column we are focusing on,
- bins: the number or range of lines that the histogram will consist of,
- kde: it controls whether the result will be a histogram or a smooth curve that represents the distribution of the data,
- xlim: it limits the range of the x-axis.

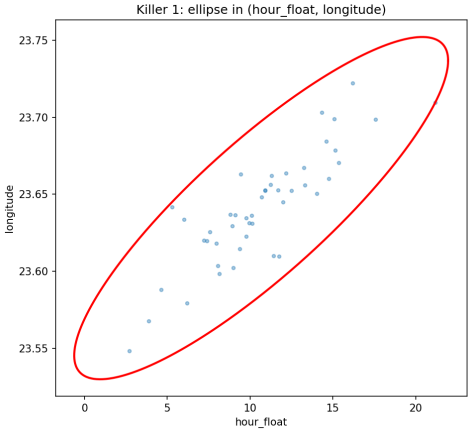
```python
def plot_hist_with_stats(series, title, bins=30, kde=False, xlim=None):
    s = series.dropna().values
    fig, ax = plt.subplots(figsize=(8, 4.5))
    sns.histplot(s, bins=bins, stat="density", kde=kde, edgecolor="white",
    ↪  color="#4C78A8", ax=ax)
    if xlim:
        ax.set_xlim(xlim)
    ax.set_title(title, fontweight="bold")
    ax.set_xlabel(title)
    ax.set_ylabel("Πυκνότητα")
    mu, sigma = np.mean(s), np.std(s, ddof=1)
    ax.axvline(mu, color="crimson", linestyle="--", label=f"μ={mu:.2f}")
    ax.legend()
    plt.tight_layout()
    return fig, ax, mu, sigma
```

Q2: Gaussian MLE per killer

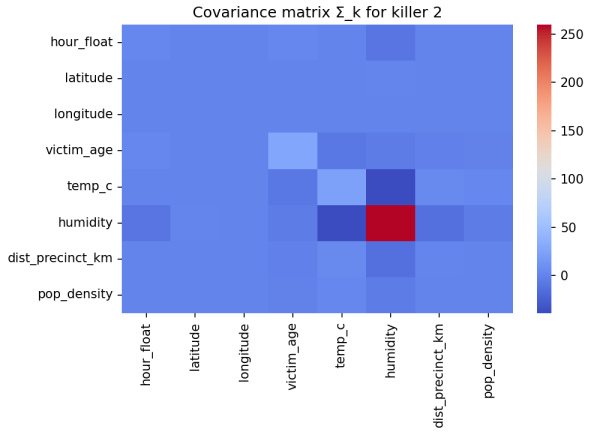
Below we present you one covariance heatmap and one 2D projection with ellipses per killer, as requested.

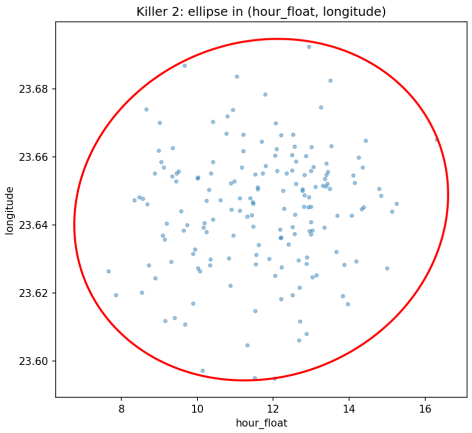
- Killer 1:



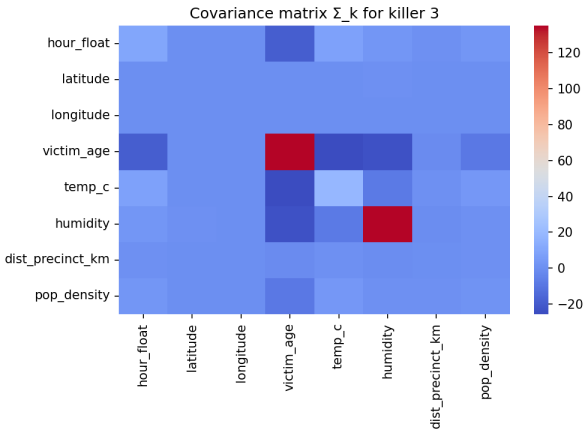


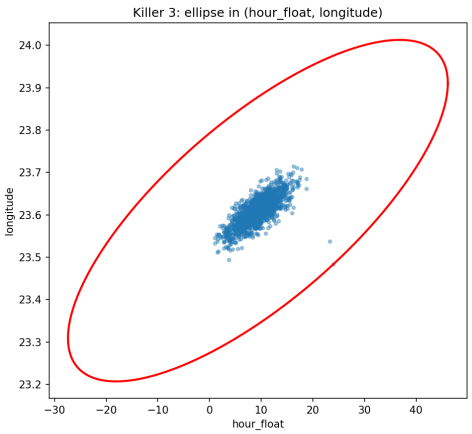
- Killer 2:



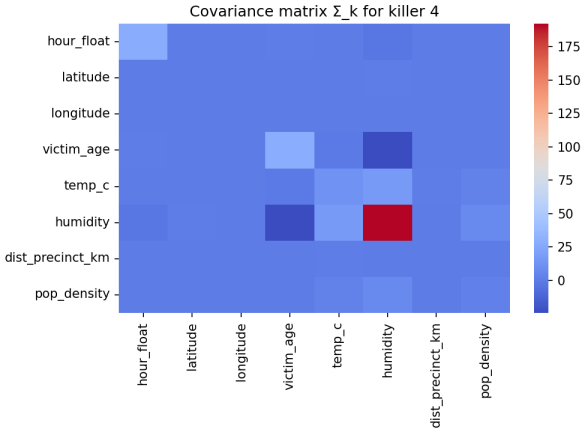


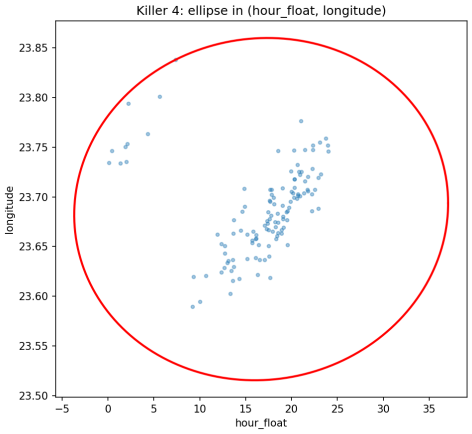
- Killer 3:



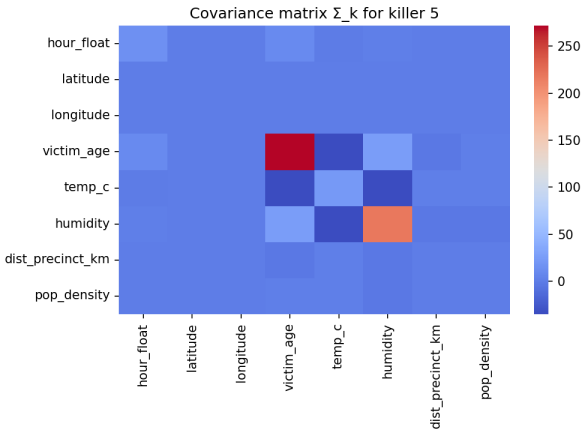


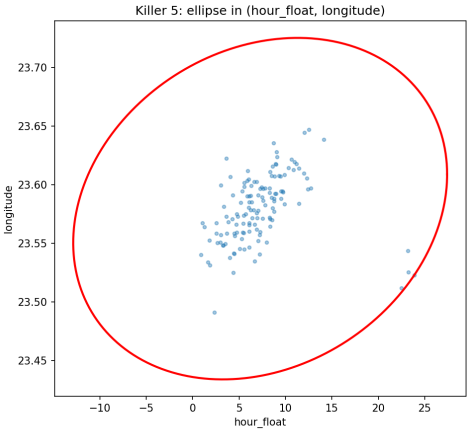
- Killer 4:



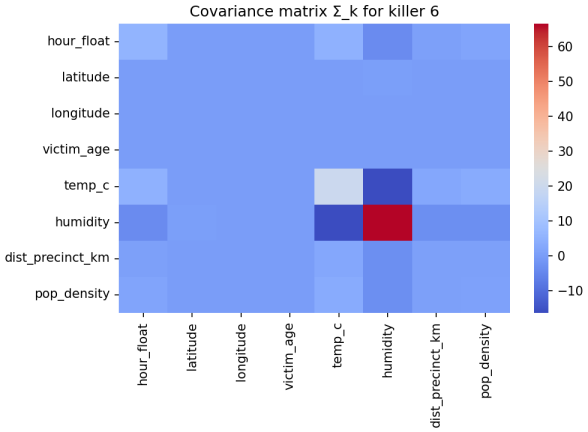


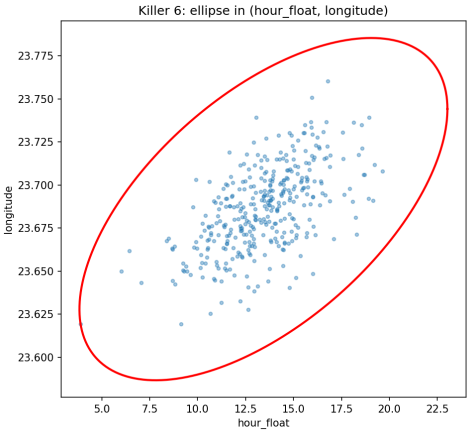
- Killer 5:



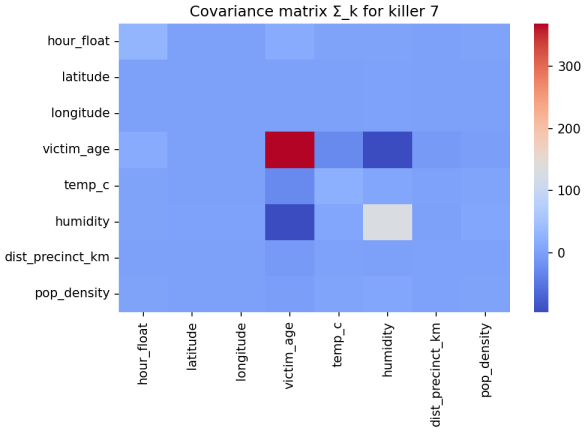


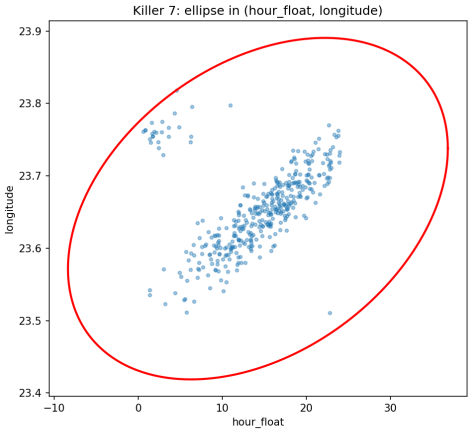
- Killer 6:



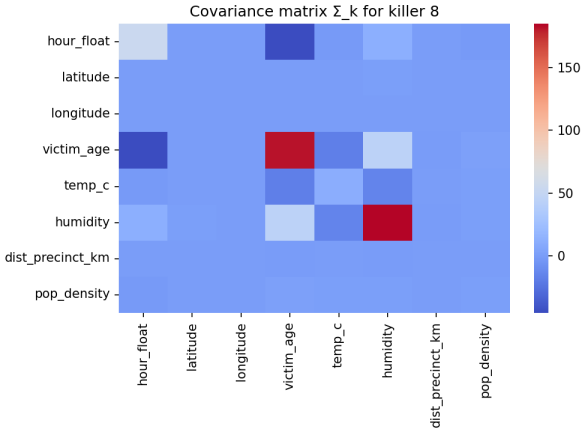


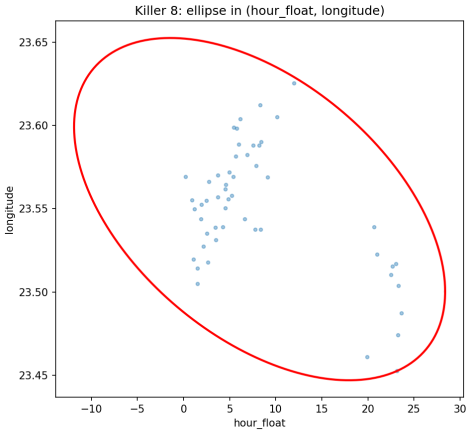
- Killer 7:





- Killer 8:





Explanation

To estimate the mean and covariance of the continuous features for each killer on TRAIN we assumed that, for each killer k, the continuous feature vector follows a multivariate Gaussian distribution. To estimate the parameters we used Maximum Likelihood Estimation (MLE), restricting ourselves only to the TRAIN split, where the killer labels are known. For each killer k, we collected all TRAIN incidents commited by that killer. This way, each killer's profile starts to assemble. Lastly, we visualise our results for better interpretation.

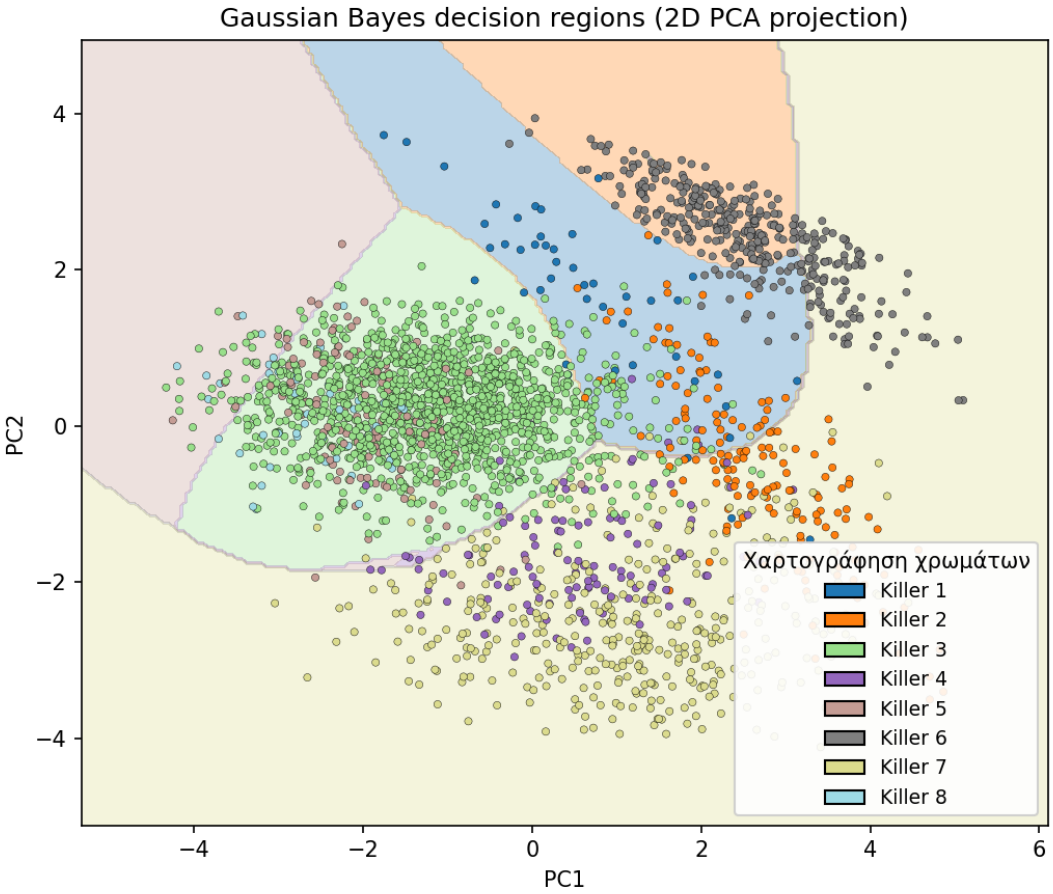
Q3: Multiclass Gaussian Bayes classifier

We evaluated the TRAIN accuracy and the VAL accuracy and concluded that they are equal to 89% and 90% respectively. Here is the requested confusion matrix on VAL.

Table 2: Confusion matrix for the Gaussian Bayes classifier on VAL.

True / Pred	1	2	3	4	5	6	7	8
1	13	1	2	0	0	1	0	0
2	1	59	1	0	0	1	0	0
3	3	0	474	0	5	0	7	2
4	0	0	2	38	1	0	4	0
5	0	0	32	1	11	0	1	3
6	0	0	0	0	0	133	0	0
7	1	2	0	3	0	0	137	0
8	0	0	15	0	2	0	0	2

We can see that the matrix produced is mostly diagonal and the killers with the highest presence are the killers 3, 6 and 7, which will be explained due to the accuracy of Q8 in the end of the assignment. Below follows the two-dimensional projection, visualising the decision regions induced by the Bayes classifier, with TRAIN points coloured by true killer. Each curved boundary seperates the feature regions of each killer.



Explanation

We implemented a multiclass Gaussian Bayes classifier using the parameters estimated in Q2. For each killer k, we modelled the continuous feature vector as a multivariate Gaussian distribution. For each incident x we calculated the long-posterior probability. Lastly, we evaluated the classifier on the VAL split by calculating the overall accuracy and of course prosuced the corresponding confusion matrix, as shown in thee code below.

```
def predict_proba(x):
    # returns vector of posterior probabilities for one x
    log_scores = []
    for k in killers:
        lg = np.log(pi[k]) + log_gaussian(
            x,
            mu_dict[k],
            invcov[k],
            logdet[k]
        )
        log_scores.append(lg)
    log_scores = np.array(log_scores)
    # softmax
    exp_scores = np.exp(log_scores - np.max(log_scores))
    return exp_scores / exp_scores.sum()

def predict_class(x):
    pp = predict_proba(x)
    return killers[np.argmax(pp)]

# Predict on TRAIN
ytrain_pred = np.array([predict_class(x) for x in Xtrain])
train_acc = accuracy_score(ytrain, ytrain_pred)

# Predict on VAL
yval_pred = np.array([predict_class(x) for x in Xval])
val_acc = accuracy_score(yval, yval_pred)

print("TRAIN accuracy:", train_acc)
print("VAL accuracy:", val_acc)
print("VAL confusion matrix:")
print(confusion_matrix(yval, yval_pred))
```

Q4: Linear classifier

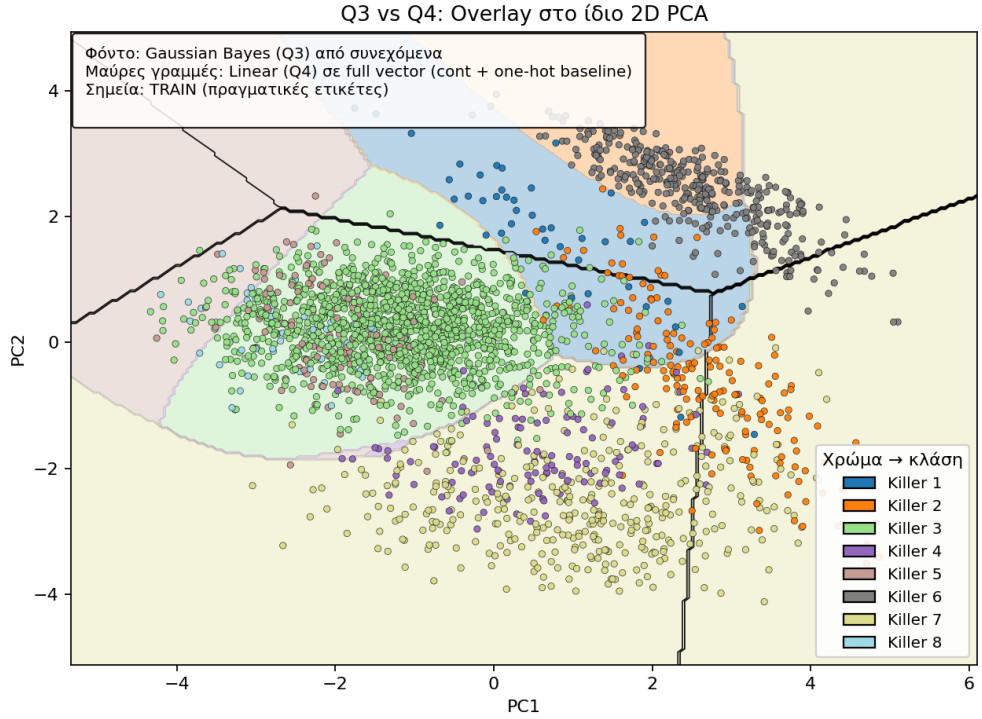
This time, the TRAIN accuracy and the VAL accuracy are equal to 77% and 78% respectively. We need to highlight that these numbers are lower than Q3's. That could mean that the Gaussian assumption is more accurate. Here is the confusion matrix on VAL.

Table 3: Confusion matrix for the Gaussian Bayes classifier on VAL.

True / Pred	1	2	3	4	5	6	7	8
1	0	1	9	0	2	4	1	0
2	1	38	17	0	0	3	3	0
3	0	4	473	0	4	1	9	0
4	1	0	4	0	0	0	40	0
5	0	1	28	1	14	0	4	0
6	0	3	12	0	0	117	1	0
7	3	4	26	0	1	1	108	0
8	0	0	12	0	4	0	3	0

Comparing it to Q3's matrix it obviously does not resemble a diagonal matrix which may mean that we have strayed from our end goal. While some numbers have slightly changed(e.g. 474 -> 473), most of them have completely changed. Also, the last column consists entirely of zeros. However, the killers with the highest presence remain the same.

Also, in the 2D PCA projection used in Q3, we overlayed the approximate linear decision boundaries and the result is the following.



When compared to Q3's RCA projection one can easily see that the regions are seperated by lines and clearly determine which one corresponds to each killer. Some may correspond to more than one killers. This happens because the linear model is not able to fully adapt to the shape of the data the way that the Gaussian Bayes does.

Explanation

We treated the killer identification as a discriminative multiclass classification problem using the full vector consisting of the 8 continuous features (hour_float, latitude, longitude, victim_age, temp_c, humidity, dist_precinct_km and pop_density) and 4 more cateforical features (weapon_code, scene_type, weather, vic_gender). We implemented the linear classifier given. We trained the model on the TRAIN split using MSE (Mean Squared Error loss). Lastly, we valuate the accuracy and the confusion matrix and produce the 2D PCA overlaying the one from Q3.

```
Xtr_t = torch.tensor(Xfull_train, dtype=torch.float32).to(device)  # (N, d_full)
Xva_t = torch.tensor(Xfull_val,   dtype=torch.float32).to(device)
ytr_1h = torch.eye(S, dtype=torch.float32)[torch.tensor(ytrain_idx)].to(device)

d_full = Xfull_train.shape[1]
linear = nn.Linear(d_full, S).to(device)
criterion = nn.MSELoss()
opt = optim.Adam(linear.parameters(), lr=1e-3)

linear.train()
for epoch in range(1500):
    opt.zero_grad()
    out = linear(Xtr_t)              # logits (S)
    loss = criterion(out, ytr_1h)    # SSE με one-hot
    loss.backward()
    opt.step()
    if epoch % 300 == 0:
        print(f"[Q4] epoch {epoch}, loss={loss.item():.4f}")

linear.eval()
with torch.no_grad():
    tr_logits = linear(Xtr_t).cpu().numpy()
    va_logits = linear(Xva_t).cpu().numpy()
tr_pred_idx = tr_logits.argmax(1)
va_pred_idx = va_logits.argmax(1)
print("Q4 TRAIN acc:", accuracy_score(ytrain_idx, tr_pred_idx))
print("Q4 VAL acc:",   accuracy_score(yval_idx,   va_pred_idx))
print("Q4 VAL confusion:\n", confusion_matrix(yval_idx, va_pred_idx))
```

Note to teacher: As you may have noticed if you have run our code, on the terminal also appear the Q3's results. Everything on terminal is exclusively for our convenience in order to make sure all things run smoothly and are executed correctly. It should not consern you as we have provided you with Q4's results in the previous section.

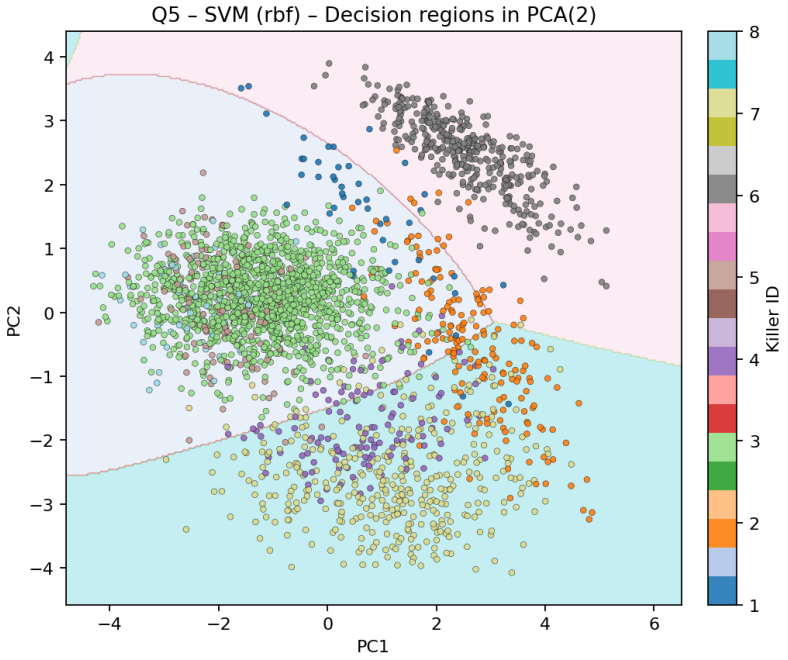
Q5: Support Vector Machine

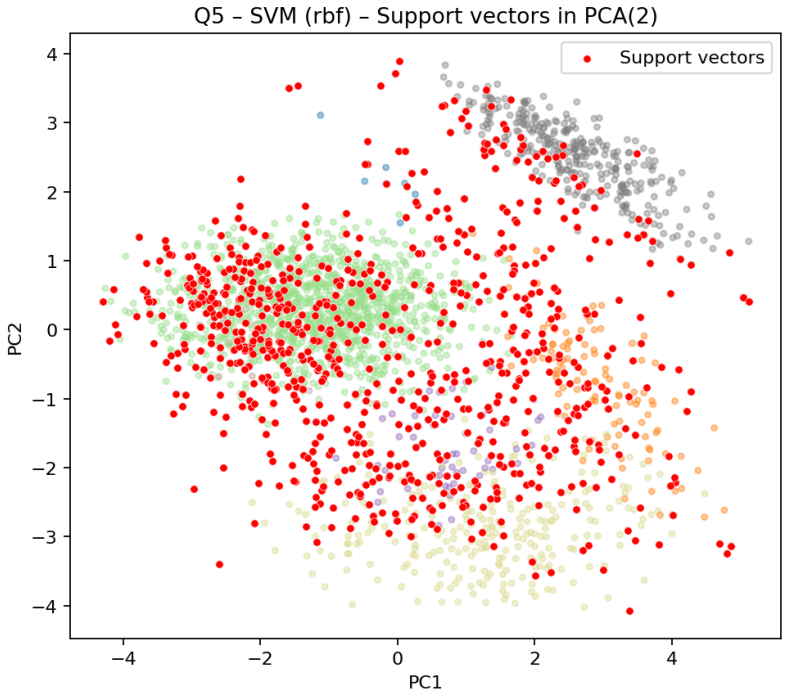
The VAL accuracy for the RBF method is 94%. The confusion matrix is the following. Once again it slightly resembles a diagonal matrix which means we are getting closer.

Table 4: Confusion matrix for the Gaussian Bayes classifier on VAL.

True / Pred	1	2	3	4	5	6	7	8
1	14	0	3	0	0	0	0	0
2	0	62	0	0	0	0	0	0
3	0	0	484	0	1	0	2	4
4	0	0	1	35	1	0	8	0
5	0	0	14	0	30	0	1	3
6	0	0	0	0	0	133	0	0
7	0	0	0	1	1	0	141	0
8	0	0	6	0	3	0	0	10

In the images that follow we can see the decision regions as well as the support vectors.





Highlighted in red are distributed the incidents that determine the characteristics of the killers' profiles.

Explanation

Our SVM setup is formed by choosing either RBF or Polynomial kernel. We prefered RBF as it allows non-linear decision boundaries. The code is modified for either one.

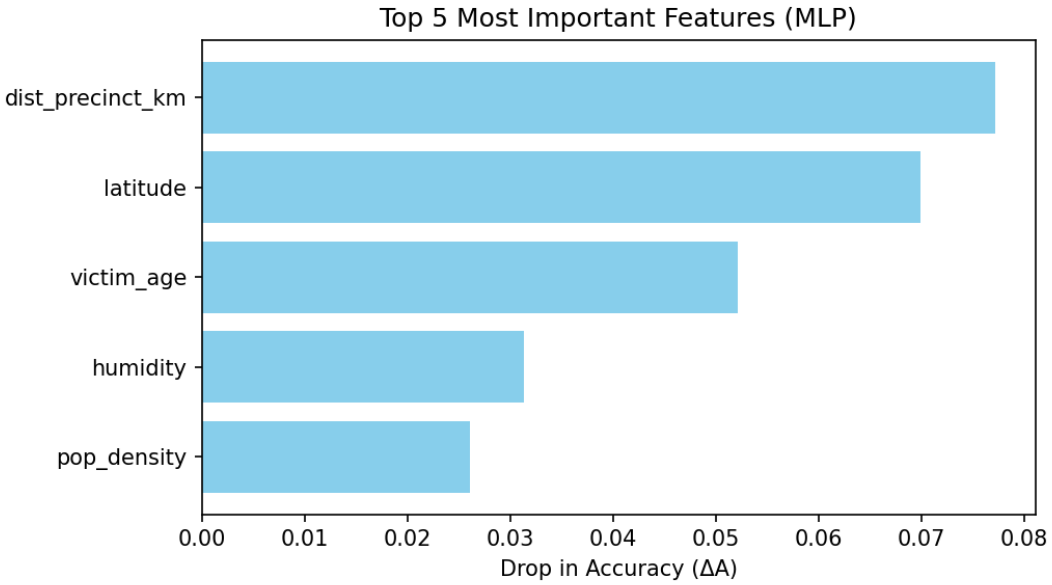
```python
def small_search_space(kernel: str):
    if kernel == "rbf":
        Cs = [0.3, 1, 3]
        gammas = ["scale", 0.1, 0.03]
        return [(c, g, None, None) for c in Cs for g in gammas]
    else:
        Cs = [0.3, 1, 3]
        gammas = ["scale", 0.1]
        degrees = [2]
        coef0s = [0, 1]
        return [(c, g, d, c0) for c in Cs for g in gammas for d in degrees
        ↪  for c0 in coef0s]
```

We continued with tuning the hyperparameters using a small manual grin search on VAL. Each candidate model was trained on TRAIN and evaluated on VAL. We chose the one that seemed more suitable to us based on the highest validation accuracy. Latsly, since the SVM is a binary classifier due to inheritance, we used One vs Rest strategy. This way, there is one binary SVN trained per killer, which means we have 8 classifiers, each one able to seperate class k from the rest and as far as prediction is concerned, it is the class with the highest score that is selected. All preprocessing was applied inside a Pipeline thus ensuring there will be no data leakage.

Q6: Multi-Layer Perceptron

The VAL accuracy is equal to 94% (same as the SVM's).
The features of most importance seem to be(in ascending order):

1. dist_previnct_km $\Delta A = 0.07$
2. latitude $\Delta A = 0.06$
3. victim_age $\Delta A = 0.05$
4. humidity $\Delta A = 0.03$
5. pop_density $\Delta A = 0.02$



Explanation

We implemented a Multilayer Perceptron (MLP) as classifier using the following code:

```
mlp = MLPClassifier(
    hidden_layer_sizes=(64, 32),    # 2 hidden layers
    activation="relu",
    solver="adam",
    max_iter=200,
    random_state=0,
    early_stopping=True
)
```

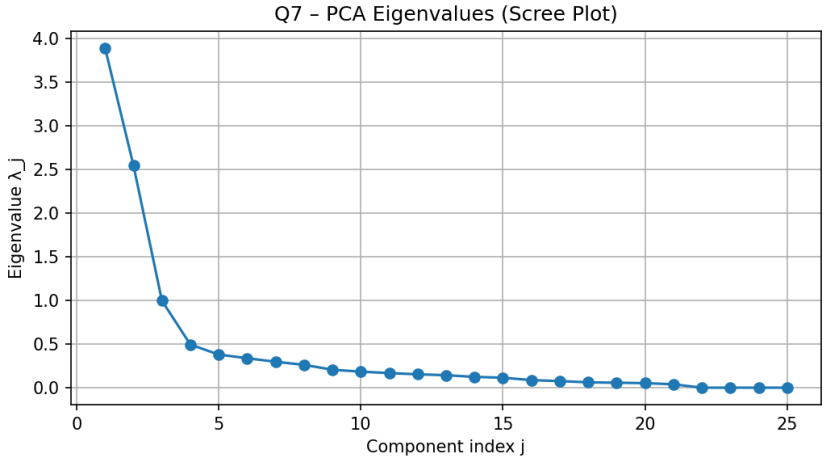
We continued by once again training the model on TRAIN, calculated VAL accuracy etc. Lastly, feature importance was estimated using permutation importance.

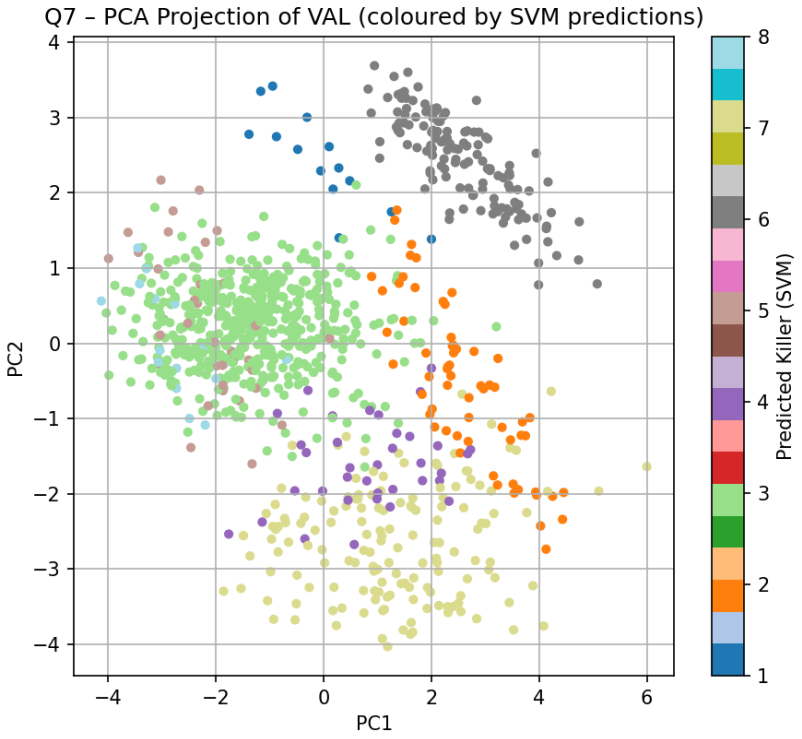
```
Z_val = model.named_steps["pre"].transform(X_val)

importances = []
rng = np.random.default_rng(0)

for j in range(Z_val.shape[1]):
    Zp = Z_val.copy()
    Zp[:, j] = rng.permutation(Zp[:, j])  # permute one feature
    pred = model.named_steps["mlp"].predict(Zp)
    Aj = accuracy_score(y_val, pred)
    importances.append((all_feat[j], A_base - Aj))

# Sort by importance
importances.sort(key=lambda x: x[1], reverse=True)
top5 = importances[:5]

print("\nTop 5 Most Important Features:")
for name, imp in top5:
    print(f"{name:35s} ΔA = {imp}")
```

The code presented computes the permutation feature importance by measuring how much the model's accuracy decreases when each feature is randomly permuted. First, the validation data are transformed using the preprocessing step of the pipeline so that the MLP receives inputs in the same format as during training. Then, for each feature, a copy of the transformed validation set is created and the values of that feature are randomly shuffled, breaking its relationship with the target variable. The modified dataset is passed to the trained MLP to obtain new predictions, and the resulting accuracy is compared with the original validation accuracy. The drop in accuracy quantifies the importance of that feature: a larger decrease indicates that the feature is more critical for the model's performance. Finally, all features are sorted by their impact, and the five most important ones are printed.

Q7: Principal Component Analysis

Below we provide you with a plot of eigenvalues versus component index and a PC1–PC2 scatter plot coloured by predicted killer labels, as requested.





Explanation

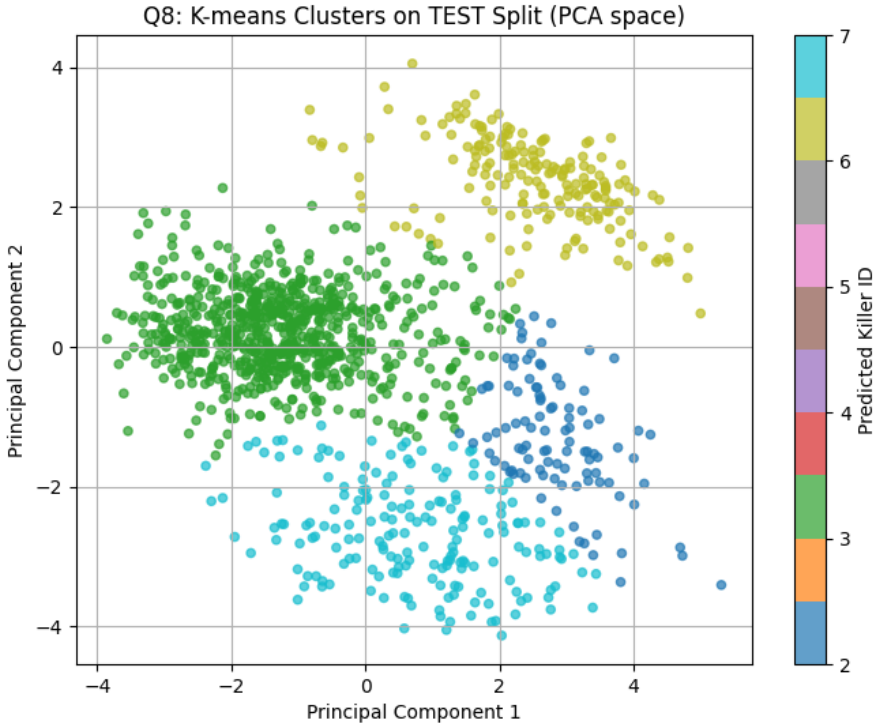
Before applying PCA, we started off by preprocessing the features this way once again ensuring no data leakage.

```
pre = ColumnTransformer([
    ("scaler", StandardScaler(), CONT),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse_output=False),
    ↪  CAT)
])
```

Subsequently, PCA was fitted on TRAIN only and then applied to both TRAIN and VAL. Firstly, we let PCA run without any restrictions on the number of components. However, for visualization purposes only we chose to set the parameter n_components to 2 to achieve projection in R^2.

Q8: k-means in PCA space

Below, we provide you with the plot with our final predicted killer labels Ci.



As you can see we came to the conclusion that there are 4 predicted killers in k-means clustering. These are killers 2, 3, 6 and 7. We would like to remind you that in Q3 we had predicted killers 3, 6 and 7. This means that even though the VAL accuracy of k-means is equal to 81%, it is still very close to the Gaussian Bayes and SVM models which have better accuracies.

Explanation

To start with, the data was once again preprocessed, fitted on TRAIN and applied on VAL and TEST. The PCA was applied on TRAIN only. The number of components was set to 2 (based on Q7's results). So, we ran k-means with k equal to S, where equals to the nnumber of killers(in our case 8). K-means is unsupervised, so the clusters do not correspond directly to the killer IDs. In order to convert clusters into class labels, we used majority voting on the TRAIN set. For each cluster, we identified all TRAIN samples assigned to that cluster, examined their true killer labels, and assigned each cluster the label of the most frequent killer in that cluster. For evaluation on VAL, samples were assigned to clusters using the fitted k-means model, mapped to killer IDs using the majority-vote mapping, and accuracy was of course calculated. Finally, for the TEST set, we produced a probability distribution over the clusters by calculating the Euclidean distance from each sample to all centroids. Distances were converted to scores using score = −distance, and a softmax approach was applied to obtain normalized probabilities.

```
test_clusters = kmeans.predict(Z_test_m)
test_pred = np.array([cluster_to_killer[c] for c in test_clusters])

distances = kmeans.transform(Z_test_m)

# 2. Μετατροπή αποστάσεων σε πιθανότητες (Softmax approach)
# Χρησιμοποιούμε exp(-dist) ώστε οι μικρές αποστάσεις να δίνουν μεγάλες πιθανότητες
inv_distances = np.exp(-distances)
# Κανονικοποίηση ώστε το άθροισμα κάθε γραμμής να είναι 1 [cite: 91]
probs_clusters = inv_distances / np.sum(inv_distances, axis=1, keepdims=True)

# 3. Δημιουργία DataFrame
submission = pd.DataFrame({"incident_id": test["incident_id"], "predicted_killer": test_pred})

# 4. Υπολογισμός πιθανοτήτων για κάθε killer_id (1 έως S)
for k in range(1, S + 1):
    # Βρίσκουμε ποια clusters (q) αντιστοιχούν στον δολοφόνο k [cite: 218]
    matching_clusters = [q for q, killer in cluster_to_killer.items() if killer == k]

    if matching_clusters:
        # Αθροίζουμε τις πιθανότητες των clusters που ανήκουν στον δολοφόνο k
        submission[f"p_killer_{k}"] = probs_clusters[:, matching_clusters].sum(axis=1)
    else:
        # Αν ο δολοφόνος k δεν κέρδισε κανένα cluster, του δίνουμε μια πολύ μικρή
        # βασική πιθανότητα (π.χ. από το πλησιέστερο cluster) για να μην είναι 0
        submission[f"p_killer_{k}"] = 1e-5

    # Επανεξισορρόπηση (Normalization) για να αθροίζουν ακριβώς στο 1 [cite: 91]
prob_cols = [f"p_killer_{k}" for k in range(1, S + 1)]
submission[prob_cols] = submission[prob_cols].div(submission[prob_cols].sum(axis=1), axis=0)
```

This code for the TEST set predicts clusters and labels similarly, then derives class probabilities by first obtaining distances of each example to all cluster centroids (kmeans.transform). Distances are converted to soft probabilities with a softmax over negative distances (exp(-distance)), so nearer centroids receive higher probability mass. These cluster-level probabilities are aggregated to class-level probabilities by summing over all clusters mapped to the same killer ID. The classes with no winning cluster get a tiny fallback probability to avoid exact zeros(1e-5). Finally, the per-row probabilities are renormalized to sum to 1, and a submission.csv is created containing the incident_id, the predicted label, and the calibrated probabilities p_killer_1 … p_killer_S.

4 Discussion and conclusions

Let's summarise the VAL accuracies of these methods up until now:

Table 5: Validation accuracy for different models.

Model	VAL accuracy
Gaussian Bayes	0.905
Linear classifier	0.782
SVM (RBF kernel)	0.948
MLP	0.944
PCA + k-means	0.813

So, in ascending order of performance we have:

1. SVM (Q5)
2. MLP (Q6)
3. Gaussian Bayes (Q3)
4. Linear classifier (Q4)
5. PCA + k-means (Q8)

We have concluded that the more helpful features in order to model behavior were geographic features (latitude and longtitude), victim-related information (their age and gender), information about the place where the crime was commited and the prevailing weather conditions (weather, temperature and scene type) and of course the weapon type (not necessarily in that order). With the use of all presented diagrams, confusion matrixes and the final file submision.csv, we highlighted the 4 killers (2, 3, 6 and 7) as the killers who are most likely to have commited the most crimes, especially in TEST. Potentially, more complex generative models could be included and explored. Additionally, there are other factors to could be analyzed and taken into consideration, such as crime frequency patterns, killer's motive or mental health disorders. Overall, the current study highlights the importance of combining proper preprocessing, nonlinear modeling and feature analysis when trying to solve real-world pattern recognition problems.

Code organisation

We implemented the code in a modular way, creating one script per question. Each script is responsible for producing the results and diagrams of its corresponding question. The diagrams from each question are stored in a separate folder. The dataset is also included in the main folder. Lastly, the prediction file `submission.csv` is also included in the main folder.

```
pattern_recognition
├── Q1_Diagrams
├── Q2_Diagrams
├── Q3_Diagrams
├── Q4_Diagrams
├── Q5_Diagrams
├── Q6_Diagrams
├── Q7_Diagrams
├── Q8_Diagrams
├── crimes.csv
├── q1.py
├── q2.py
├── q3.py
├── q4.py
├── q5.py
├── q6.py
├── q7.py
├── q8.py
└── submission.csv
```

Libraries and data load

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from sklearn.mixture import GaussianMixture

# Φόρτωση & φιλτράρισμα
csv_path = "crimes.csv"
df = pd.read_csv(csv_path)

# Κρατάμε μόνο TRAIN+VAL
df = df[df["split"].isin(["TRAIN", "VAL"])].copy()

# Τυπική επιβεβαίωση ότι έχουμε όλες τις απαιτούμενες στήλες
required_cols = ["hour_float", "victim_age", "latitude",
 ↪  "longitude"]
missing = [c for c in required_cols if c not in df.columns]
assert not missing, f"Λείπουν στήλες: {missing}"
```

In this specific script we load the required libraries and the data from the `crimes.csv` file. We split the data into TRAIN and VAL as it is required for Q1, a similar method is used for all the other questions. Lastly, we confirm that all the required columns are present in the dataset. This is a common practice to avoid errors later on.

Briefly describe your Python code structure (scripts, notebooks, helper modules) and how to reproduce:

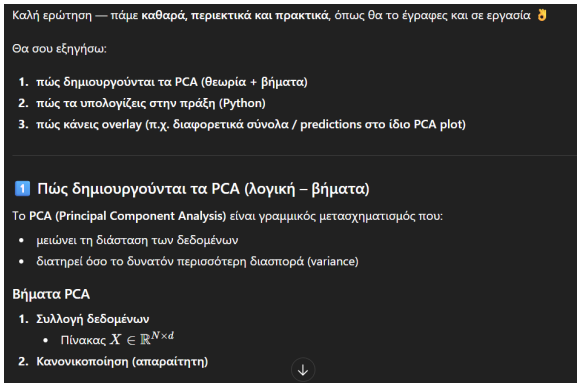
- the main results and figures in this report;
- the prediction file `submission.csv`.

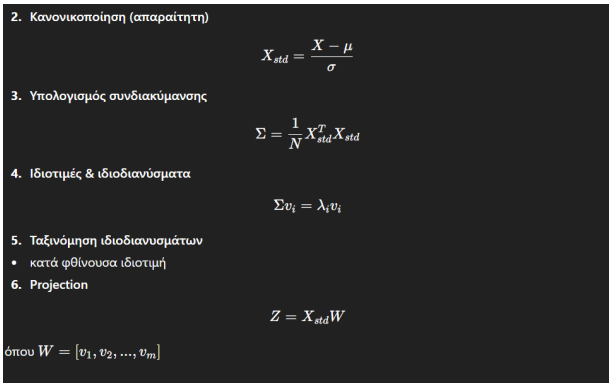
5 LLM prompts and responses

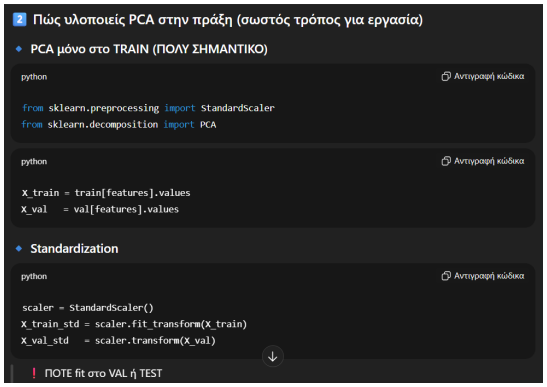
The contribution of an AI, specifically Copilot, in this project was purely auxiliary. In general, we consulted Copilot in order to provide us with formulas, types, libraries and assistance in diagrams' creation. In particular, we list all prompts and the corresponding responses we used during the preparation of this project.

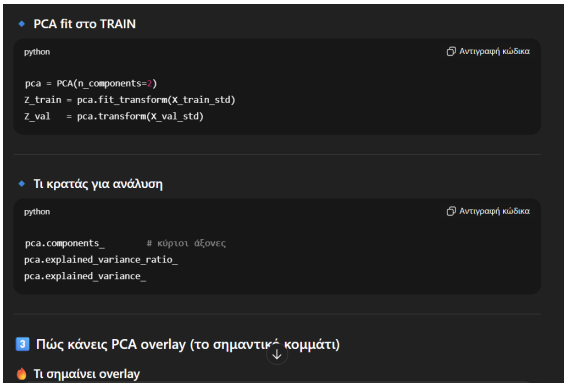
Comment: As far as Q3 is concerned, we asked Copilot how we can create a 2D PCA projection, as we were facing some problems. Also, we asked how we can include a legend box that would contain the killers and their corresponding colors. Unfortunately, we are unable to find the corresponding prompt and response.
In Q4 we asked Copilot how we can make a 2D PCA projection that overlays the one created in Q3.

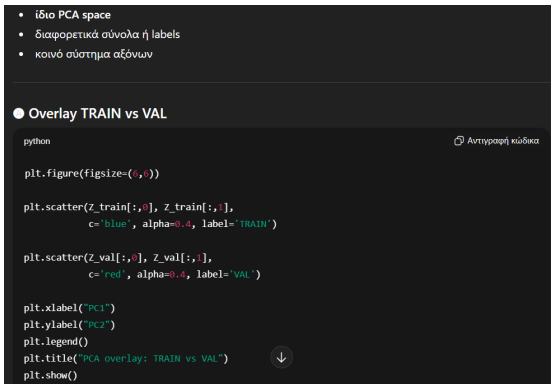
Prompt: Πως μπορούν να δημιουργηθούν PCA και πως μπορώ να τα κάνω overlay?
Response:











```
plt.title("PCA (colored by predicted killer)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

👉 Αυτό είναι **overlay σύγκρισης** (ίδιο PCA space)

🔵 Overlay πολλών μοντέλων

```python
plt.figure(figsize=(6,6))

plt.scatter(Z_val[:,0], Z_val[:,1],
            c=y_pred_svm, marker='o', label='SVM', alpha=0.5)

plt.scatter(Z_val[:,0], Z_val[:,1],
            c=y_pred_bayes, marker='x', label='Bayes', alpha=0.5)

plt.legend()
plt.title("Overlay predictions on PCA")
plt.show()
```

🧠 Πώς το περιγράφεις σε εργασία

> *«Το PCA υπολογίστηκε αποκλειστικά στο σύνολο εκπαίδευσης μετά από κανονικοποίηση. Το ίδιο PCA transformation εφαρμόστηκε στα σύνολα validation/test, επιτρέποντας άμεσο οπτικό συγκριτικό έλεγχο (overlay) των δεδομένων στο ίδιο υποχώρο.»*

⚠️ Κλασικά λάθη (να τα αποφύγεις)

- ❌ PCA fit σε όλο το dataset
- ❌ PCA χωρίς standardization
- ❌ διαφορετικό PCA για κάθε split
- ❌ σύγκριση διαφορετικών PCA axes

🧩 Σύνδεση με τα Q της εργασίας

- **Q3–Q6**: PCA overlay για decision regions
- **Q7**: Scree plot + επιλογή m
- **Q8**: PCA → k-means → overlay clusters

Prompt: Τί είναι το rbf kernel και πως μπορώ να το χρησιμοποιήσω σε python?
Response:

Καλή ερώτηση — το **RBF kernel** είναι από τα πιο βασικά και εξεταστικά θέματα σε SVM. Πάμε **ξεκάθαρα, δομημένα και με παράδειγμα Python** 👌

1️⃣ Τι είναι το RBF kernel (θεωρία)

RBF (Radial Basis Function) kernel = **Gaussian kernel**

Χρησιμοποιείται στα **Support Vector Machines** για να:

- χειριστούν **μη γραμμικά** διαχωρίσιμα δεδομένα
- μεταφέρουν τα δεδομένα σε **υψηλότερη διάσταση** όπου ο διαχωρισμός γίνεται γραμμικός

🔹 Μαθηματικός ορισμός

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$$

όπου:

- x, x': δύο σημεία δεδομένων
- $\|\cdot\|$: Ευκλείδεια απόσταση
- $\gamma > 0$: παράμετρος πλάτους Gaussian

🔹 Διαισθητική ερμηνεία

- Κοντινά σημεία → kernel ≈ 1
- Μακρινά σημεία → kernel ≈ 0
- Δημιουργεί **τοπικές περιοχές επιρροής**
- Επιτρέπει **καμπύλες (μη γραμμικές) αποφάσεις**

2️⃣ Πότε χρησιμοποιούμε RBF kernel

✔ Όταν τα δεδομένα **δεν διαχωρίζονται γραμμικά**
✔ Όταν **δεν γνωρίζουμε** τη σωστή μη γραμμικότητα
✔ Όταν τα features είναι **συνεχή**
✔ Default επιλογή σε SVM

3️⃣ Παράμετροι RBF SVM (πολύ σημαντικό)

🔸 C – Regularization

- Μεγάλο C → αυστηρός διαχωρισμός (overfitting)
- Μικρό C → πιο «μαλακό» όριο (underfitting)

🔸 γ (gamma)

- Μεγάλο γ → πολύ στενές Gaussian → overfitting
- Μικρό γ → πολύ φαρδιές Gaussian → underfitting

4️⃣ Πώς χρησιμοποιώ RBF kernel σε Python (scikit-learn)

🔹 Βασική χρήση

```python
from sklearn.svm import SVC

svm_rbf = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_rbf.fit(X_train, y_train)

y_pred = svm_rbf.predict(X_val)
```

🔹 Τι σημαίνει `gamma='scale'`

$$\gamma = \frac{1}{d \cdot \mathrm{Var}(X)}$$

✔ ασφαλής default επιλογή

5️⃣ Σωστή ροή για εργασία / εξετάσεις

🔹 Standardization (ΑΠΑΡΑΙΤΗΤΟ)

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_val_std   = scaler.transform(X_val)
```

Εκπαίδευση SVM

python — Αντιγραφή κώδικα

```
svm = SVC(kernel='rbf', C=10, gamma=0.1)
svm.fit(X_train_std, y_train)
```

Αξιολόγηση

python — Αντιγραφή κώδικα

```
from sklearn.metrics import accuracy_score, confusion_matrix

acc = accuracy_score(y_val, y_pred)
cm  = confusion_matrix(y_val, y_pred)
```

6 Grid Search για C και γ (όπως ζητάει εργασία)

python — Αντιγραφή κώδικα

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.01, 0.1, 1]
}

svm = SVC(kernel='rbf')

grid = GridSearchCV(svm, param_grid, cv=5)
grid.fit(X_train_std, y_train)

best_model = grid.best_estimator_
```

7 Οπτικοποίηση RBF decision regions σε PCA (overlay)

python — Αντιγραφή κώδικα

```
Z_train = pca.fit_transform(X_train_std)

svm.fit(Z_train, y_train)

# μετά grid για decision boundary
```

> Το RBF δημιουργεί **καμπύλα decision boundaries** στο PCA χώρο

Prompt: Στο q6, τι είναι το baseline accuracy και πως κάνουμε shuffle?
Response:

Πολύ καλή ερώτηση — στο **Q6** αυτό είναι *κλασικό σημείο που μπερδεύει*. Πάμε **ξεκάθαρα, εξεταστικά και πρακτικά**, ακριβώς όπως ζητάει η εκφώνηση 👌

1 Τι είναι το baseline accuracy (Q6)

👉 Ορισμός

Baseline accuracy =
η αρχική απόδοση του εκπαιδευμένου μοντέλου, χωρίς καμία τροποποίηση στα δεδομένα.

Στο Q6:

- είναι η **accuracy του MLP στο validation set**
- με **όλα τα features όπως είναι**

🔹 Γιατί τη χρειαζόμαστε

Χρησιμοποιείται ως:

- **σημείο αναφοράς**
- για να μετρήσουμε **πόσο σημαντικό είναι κάθε feature**

> Αν χαλάσω ένα feature και η accuracy πέσει πολύ → το feature είναι σημαντικό

🔹 Παράδειγμα

text — Αντιγραφή κώδικα

```
Baseline accuracy = 0.78
```

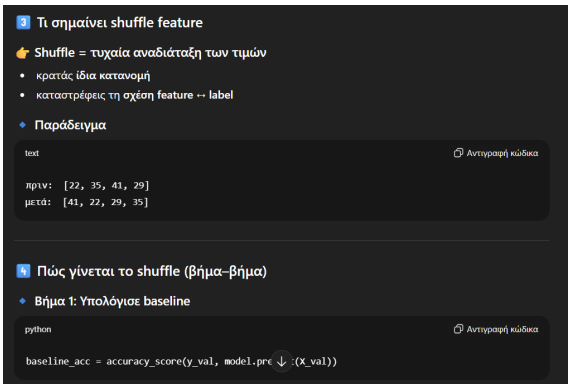
2 Τι είναι το Permutation Feature Importance

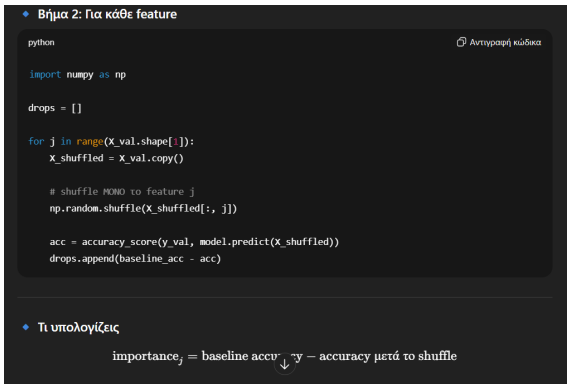
Στο Q6 **δεν** ζητά:

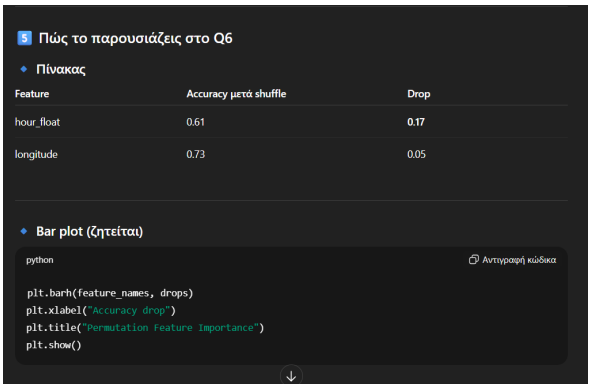
- weights
- gradients
- SHAP

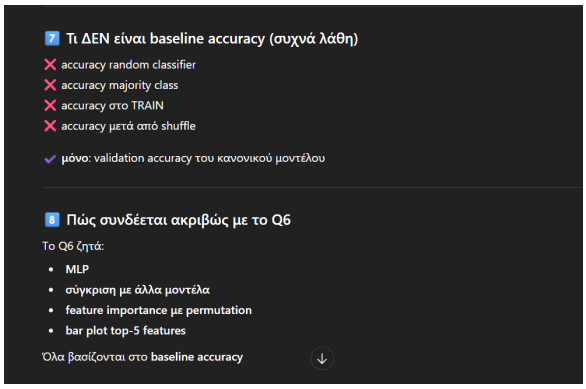
Ζητά **Permutation Importance**, δηλαδή:

> *«Χαλάμε» ένα feature ανακατεύοντας τις τιμές τ… και βλέπουμε πόσο πέφτει η accuracy.*







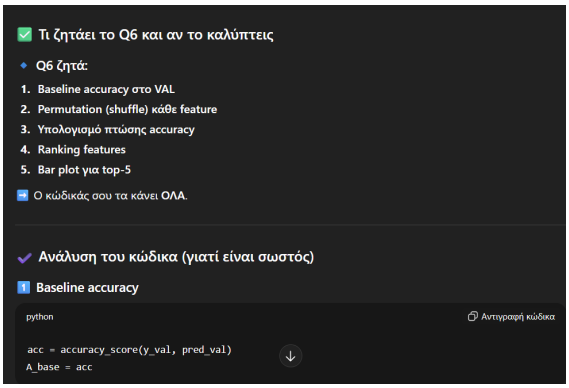


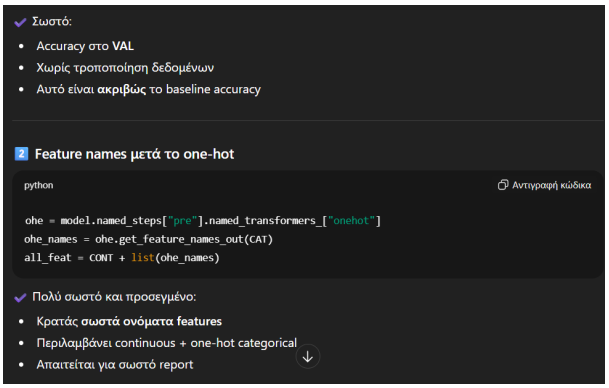
Prompt: Αυτός ο κώδικας απαντά στο ερώτημα?

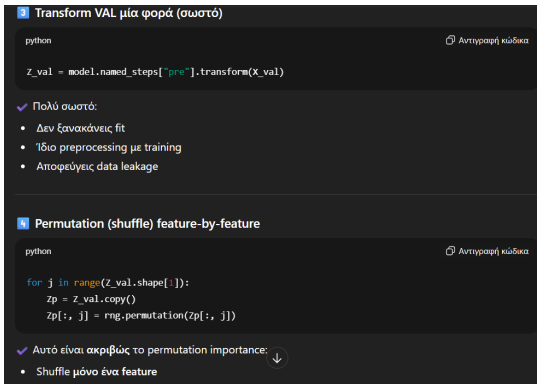
```python
acc = accuracy_score(y_val, pred_val)
print(f"VAL Accuracy: {acc}")

# ========================
# Permutation Feature Importance
# ========================

# Baseline
accuracy A_base = acc

# Get feature names after one-hot encoding
ohe = model.named_steps["pre"].named_transformers_["onehot"]
ohe_names = ohe.get_feature_names_out(CAT)
all_feat = CONT + list(ohe_names)

# Transform VAL once
Z_val = model.named_steps["pre"].transform(X_val)

importances = [] rng = np.random.default_rng(0)

for j in range(Z_val.shape[1]):
  Zp = Z_val.copy()
  Zp[:, j] = rng.permutation(Zp[:, j]) # permute one feature
  pred = model.named_steps["mlp"].predict(Zp)
  Aj = accuracy_score(y_val, pred)
  importances.append((all_feat[j], A_base - Aj))

# Sort by importance
importances.sort(key=lambda x: x[1], reverse=True)
```

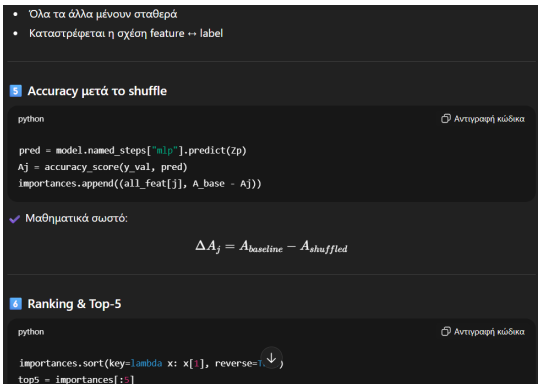
```python
top5 = importances[:5]

print("\nTop 5 Most Important Features:")
for name, imp in top5:
  print(f"{name:35s} ΔA = {imp}")

# ========================
# Plot top 5
# ========================

names = [x[0] for x in top5]
values = [x[1] for x in top5]

plt.figure(figsize=(7,4))
plt.barh(names, values, color="skyblue")
plt.gca().invert_yaxis()
plt.title("Top 5 Most Important Features (MLP)")
plt.xlabel("Drop in Accuracy (ΔA)")
plt.tight_layout()
plt.savefig("q6_top5_features.png", dpi=150)
```

Response:









7 Bar plot (όπως ζητείται)

python — Αντιγραφή κώδικα

```
plt.barh(names, values)
plt.title("Top 5 Most Important Features (MLP)")
```

✔ Καλύπτει πλήρως το ζητούμενο visualization

🟢 Τελική ετυμηγορία

- ✅ **ΝΑΙ, ο κώδικάς σου απαντά πλήρως και σωστά στο Q6**
- ✔ Ορθός ορισμός baseline accuracy
- ✔ Σωστό permutation feature importance
- ✔ Σωστό ranking & plot
- ✔ Report-ready

🔍 Μικρές παρατηρήσεις (προαιρετικές, όχι υποχρεωτικές)

🔹 (1) Reproducibility (το έχεις ήδη 👏)

python — Αντιγραφή κώδικα

```
rng = np.random.default_rng(0)
```

✔ Πολύ καλό — μπορείς να το αναφέρεις στο report

🔹 (2) Θεωρητική διευκρίνιση (μόνο στο κείμενο)

Μπορείς να γράψεις:

> *«Για τα one-hot encoded χαρακτηριστικά, το permutation importance αναφέρεται σε μεμονωμένες δυαδικές κατηγορίες.»*

(δεν χρειάζεται αλλαγή κώδικα)