École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

Rapport de stage de première année

Analyse descriptive et inférentielle des données du réseau ferroviaire

Étudiant : Soni Diedhiou

Établissement : ENSIIE

Encadrant académique : Abass Sagna

Organisme d'accueil :
Conservatoire National des Arts et Métiers (CNAM)
Département Mathématiques Statistiques (EPN06)
2, rue Conté, 75003 Paris

Tuteur de stage : Dariush Ghorbanzadeh

Dates du stage : 02/06/25 - 01/08/25

Table des matières

Acknowledgment

Je tiens à exprimer ma profonde gratitude à l'ensemble des personnes qui ont contribué au bon déroulement de mon stage au sein du Conservatoire National des Arts et Métiers (CNAM).

Je remercie tout particulièrement Monsieur Dariush Ghorbanzadeh, maître de conférences au CNAM, pour son encadrement, ses conseils avisés et sa disponibilité tout au long de ce travail. Ses orientations m'ont permis d'approfondir mes connaissances et de mener à bien les différentes étapes de ce projet.

Je souhaite aussi remercier l'École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise (ENSIIE) pour la qualité de son enseignement et pour m'avoir offert l'opportunité d'effectuer ce stage dans un environnement de recherche stimulant.

Abstract

This report presents the work carried out during my first-year internship at the Conservatoire National des Arts et Métiers (CNAM) in Paris, within the Department of Mathematics and Statistics (EPN06). The objective of the internship was to design and deploy a web application for the descriptive and inferential analysis of SNCF public railway data. The application allows users to explore TER punctuality indicators, to compute descriptive statistics, and to adjust probability laws (Normal, Gamma, Beta) using maximum likelihood estimation.

This work combined data analysis, statistical modeling, and web development with Flask, Pandas, Matplotlib, and SciPy. The application was deployed online on PythonAnywhere, making the tool accessible to both an academic audience and users interested in railway performance.

Glossary

SNCF Société Nationale des Chemins de fer Français.

TER Trains Express Régionaux.

CNAM Conservatoire National des Arts et Métiers.

ENSIIE École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise.

MLE Maximum Likelihood Estimation.

HTML HyperText Markup Language.

CSS Cascading Style Sheets.

API Application Programming Interface.

Table des figures

Chapitre 1

Introduction : Work environment and Internship

1.1 Enterprise : CNAM

Le Conservatoire National des Arts et Métiers (CNAM) est un établissement public d'enseignement supérieur et de recherche, situé à Paris. Créé en 1794, il a pour mission de former des ingénieurs et cadres dans de nombreux domaines scientifiques et techniques, et de contribuer à la recherche appliquée en partenariat avec le monde industriel. Au sein du CNAM, le département **Mathématiques et Statistiques (EPN06)** mène des activités d'enseignement et de recherche dans les domaines des probabilités, des statistiques, de l'analyse des données et de leurs applications à divers secteurs tels que les transports, l'industrie et les services.

1.2 Work department

Mon stage a été effectué dans le département **Mathématiques et Statistiques (EPN06)**, dirigé par Monsieur Dariush Ghorbanzadeh, maître de conférences. L'équipe pédagogique Mathématiques et Statistiques (EPN06) se focalise notamment sur : La modélisation mathématique, les outils mathématiques et les statistiques appliquées, avec des applications dans l'industrie, le commerce, la finance, l'assurance ou encore la santé Maths Cnam . Elle se compose d'une vingtaine d'enseignants-chercheurs répartis en deux pôles pédagogiques : Modélisation et Outils Mathématiques Statistiques Ces pôles soutiennent des formations autour des probabilités, des statistiques, de la fouille de données (data), des méthodes de prévision, ainsi que des techniques avancées comme l'optimisation, l'analyse des données massives.

1.3 Context/Problem

La ponctualité des trains constitue un enjeu majeur pour la qualité du service ferroviaire. En France, la SNCF met à disposition en libre accès des données sur la régularité mensuelle des trains TER, disponibles depuis janvier 2013 sur le portail `data.sncf.com`. Ces données comprennent notamment le nombre de trains programmés, circulés, annulés, ainsi que le taux de régularité mesuré à l'arrivée au terminus.

L'exploitation de ces données pose plusieurs défis :

— leur **prétraitement** et leur organisation afin de permettre une consultation claire par région, année et mois ;

— la **visualisation** et l'interprétation statistique de séries temporelles et d'indicateurs de performance ferroviaire ;

— la **modélisation probabiliste** des taux de régularité, qui nécessitent l'ajustement de lois de probabilité adaptées.

1.4 Internship objectives

L'objectif principal du stage a été de concevoir et de déployer une application web interactive permettant :

— d'explorer et de visualiser les données publiques de la SNCF relatives aux trains TER ;

— de réaliser une analyse descriptive des indicateurs de régularité (programmés, annulés, retardés, régularité) ;

— d'effectuer une analyse inférentielle en ajustant plusieurs lois de probabilité (normale, gamma, bêta) aux données observées, en utilisant notamment la méthode du maximum de vraisemblance ;

— de mettre en ligne l'outil développé, via le framework Flask et la plateforme `PythonAnywhere`.

1.5 Main contributions

Au terme de ce stage, les contributions principales sont :

— la réalisation d'une **application web** ergonomique et pédagogique (`http://sondieh.pythonanywhere.com`), intégrant à la fois des fonctionnalités d'exploration des données et des outils d'analyse statistique ;

— la mise en œuvre de **méthodes statistiques avancées**, incluant l'estimation par maximum de vraisemblance et la comparaison de lois de probabilité adaptées au problème étudié ;

— la **valorisation des données SNCF** par une présentation claire et interactive, accessible aussi bien à un public académique qu'à un public intéressé par la per-

formance ferroviaire.

Chapitre 2

Background and Preliminaries

2.1 Internship specifications

Le cahier des charges du stage consistait à concevoir une application web interactive permettant d'analyser les données ouvertes de la SNCF sur la régularité des trains TER. Les fonctionnalités attendues étaient les suivantes :

- charger et filtrer les données par région, année et mois ;
- afficher des tableaux récapitulatifs et des statistiques descriptives (moyenne, variance, skewness, kurtosis, etc.) ;
- générer des visualisations graphiques telles que des histogrammes ;
- proposer une étude statistique plus avancée, fondée sur l'ajustement de lois de probabilité (normale, gamma, bêta) ;
- comparer les lois candidates et retenir la plus adaptée ;
- mettre en ligne l'application, accessible via un navigateur internet.

2.2 Methods and tools used in the enterprise

Pour répondre à ces exigences, plusieurs outils et technologies ont été employés :

- **Langage Python** : langage principal utilisé pour le développement du projet.
- **Flask** : micro-framework Python permettant de créer et gérer l'application web, les routes, et le lien entre code et interface.
- **Pandas** : bibliothèque pour la manipulation et l'analyse de données tabulaires (chargement, filtrage, agrégation).
- **Matplotlib et Numpy** : bibliothèques utilisées pour le calcul numérique et la génération des graphiques (histogrammes, densités de probabilité).
- **SciPy** : utilisée pour l'optimisation et l'ajustement de lois statistiques (fonction de vraisemblance, estimateurs par maximum de vraisemblance).
- **HTML, Bootstrap et Jinja2** : pour la conception des interfaces utilisateurs (pages web dynamiques, navigation, mise en forme).

— **MathJax** : intégration de formules mathématiques dans les pages web.
— **PythonAnywhere** : plateforme de déploiement permettant de rendre l'application accessible en ligne.

2.3 Software architecture to be extended

L'architecture logicielle mise en place repose sur une séparation claire entre :
— `app.py` : cœur de l'application Flask, contenant les routes, la gestion des requêtes utilisateurs, le rendu des templates HTML et l'intégration des graphiques encodés en base64 ;
— `utils.py` : module utilitaire rassemblant les fonctions statistiques (calculs de moments, estimation des paramètres, ajustements de lois, génération de graphiques) ;
— le dossier `templates/` : contenant les pages HTML dynamiques (accueil, données, statistiques, histogrammes, estimation, étude théorique, ajustements par loi) ;
— le dossier `static/` : rassemblant les ressources statiques telles que les images (logos, illustrations).

Cette architecture assure une bonne modularité :
— la logique statistique est isolée dans un module dédié, réutilisable indépendamment de l'interface ;
— les templates HTML séparent la présentation (front-end) de la logique Python (back-end) ;
— le framework Flask assure la communication entre les deux, en transmettant les résultats des calculs aux pages web.

Chapitre 3

Implemented Solution and its Contributions

3.1 Study of potential solutions

Afin de modéliser les taux de régularité des trains TER, plusieurs lois de probabilité ont été envisagées. Le choix d'une loi dépend de la nature de la variable étudiée (bornée, positive, symétrique ou non) et de la forme empirique observée sur les histogrammes.

Loi Normale

La loi normale $\mathcal{N}(\mu, \sigma^2)$ est définie sur $\mathbb{R}$ et caractérisée par deux paramètres : la moyenne $\mu \in \mathbb{R}$ et l'écart-type $\sigma > 0$. Sa densité de probabilité est donnée par :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad x \in \mathbb{R}.$$

Elle présente une forme en cloche, symétrique autour de μ, et constitue un modèle de référence en statistique.

- **Avantages :** simplicité d'utilisation, interprétation intuitive des paramètres, adaptée à des données symétriques.
- **Inconvénients :** support non borné ; nécessite une troncature et une transformation affine pour s'adapter à des données limitées à l'intervalle $[0, 100]$.

Loi Gamma

La loi Gamma, définie par deux paramètres (α, β) avec $\alpha > 0$ (paramètre de forme) et $\beta > 0$ (paramètre de taux), est supportée sur $\mathbb{R}^+$. Sa densité de probabilité est donnée par :

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}, \quad x > 0,$$

où $\Gamma(\alpha)$ désigne la fonction Gamma d'Euler. Elle est souvent utilisée pour modéliser des durées ou des phénomènes asymétriques.

- **Avantages :** permet de représenter des distributions asymétriques, adaptée aux données strictement positives.
- **Inconvénients :** nécessite une troncature ou une transformation pour s'adapter à l'intervalle $[0, 100]$.

Loi Bêta

La loi Bêta est définie sur l'intervalle $[0, 1]$ avec deux paramètres (a, b), où $a > 0$ et $b > 0$. Sa densité de probabilité est donnée par :

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\,\Gamma(b)}\, x^{a-1}(1-x)^{b-1}, \quad x \in [0, 1].$$

Par un simple changement d'échelle, elle peut être appliquée aux taux de régularité compris entre 0 et 100.

- **Avantages :** support borné, grande flexibilité (formes symétriques ou fortement asymétriques), directement adaptée aux proportions et aux taux.
- **Inconvénients :** paramétrisation parfois moins intuitive que pour la loi Normale (les paramètres (a, b) n'ont pas d'interprétation directe en termes de moyenne et variance).

Synthèse

Ces trois lois présentent des caractéristiques complémentaires :

- la loi normale est un modèle standard pour des phénomènes centrés et symétriques ;
- la loi gamma est pertinente pour des distributions asymétriques et étalées ;
- la loi bêta est particulièrement adaptée pour les données bornées comme les taux de régularité.

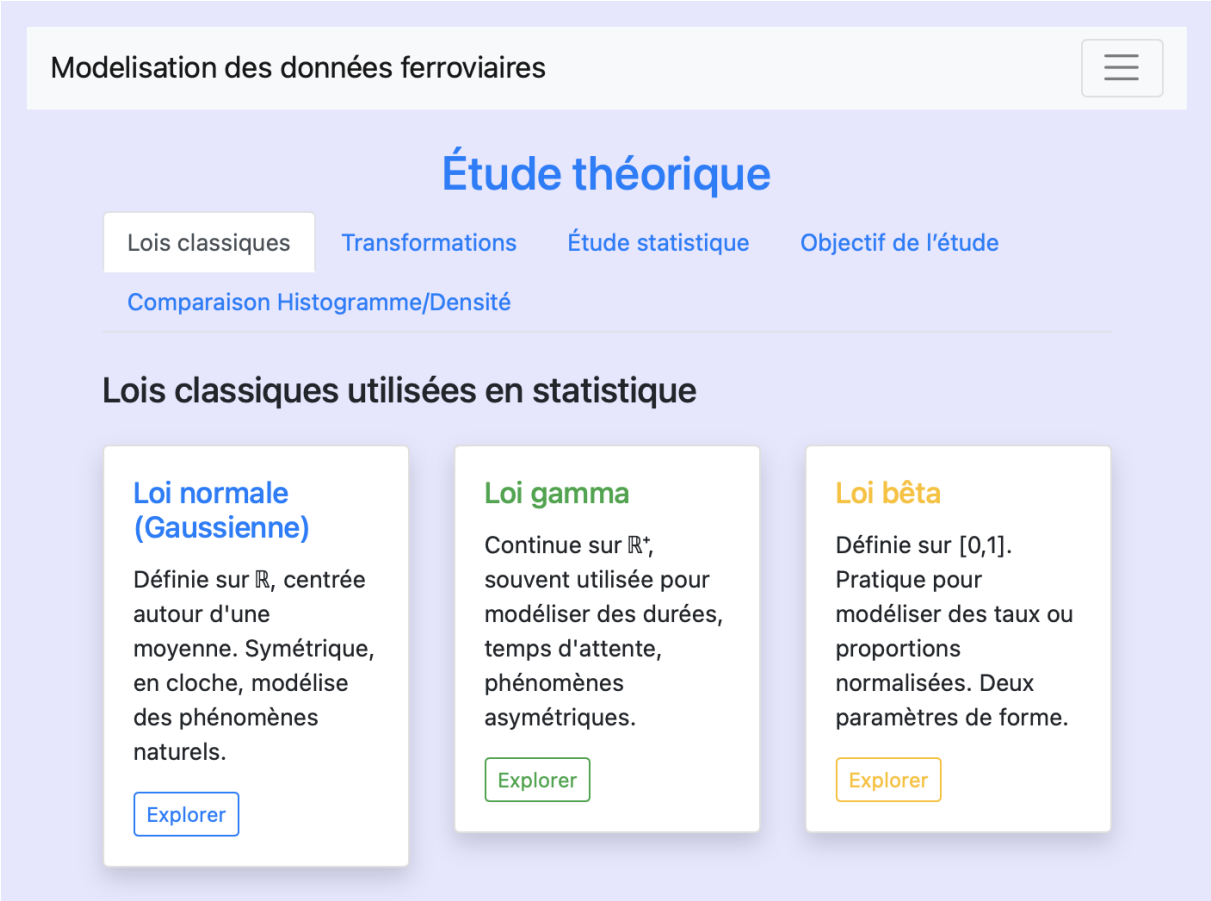


FIGURE 3.1 – Page *Étude théorique* de l'application web, présentant les lois candidates (Normale, Gamma, Bêta) utilisées pour modéliser les taux de régularité.

Pages pédagogiques interactives L'application intègre également des pages à visée pédagogique permettant de visualiser l'influence des paramètres des lois de probabilité. Pour chaque loi (Normale, Gamma, Bêta), l'utilisateur peut modifier les paramètres à l'aide de formulaires et observer immédiatement la modification de la courbe de densité.

Ces modules interactifs constituent un support précieux à l'apprentissage, car ils offrent une représentation visuelle de notions parfois abstraites comme l'effet de la variance sur la loi Normale, ou l'impact des paramètres de forme sur les lois Gamma et Bêta. Ils renforcent ainsi la dimension didactique du projet.

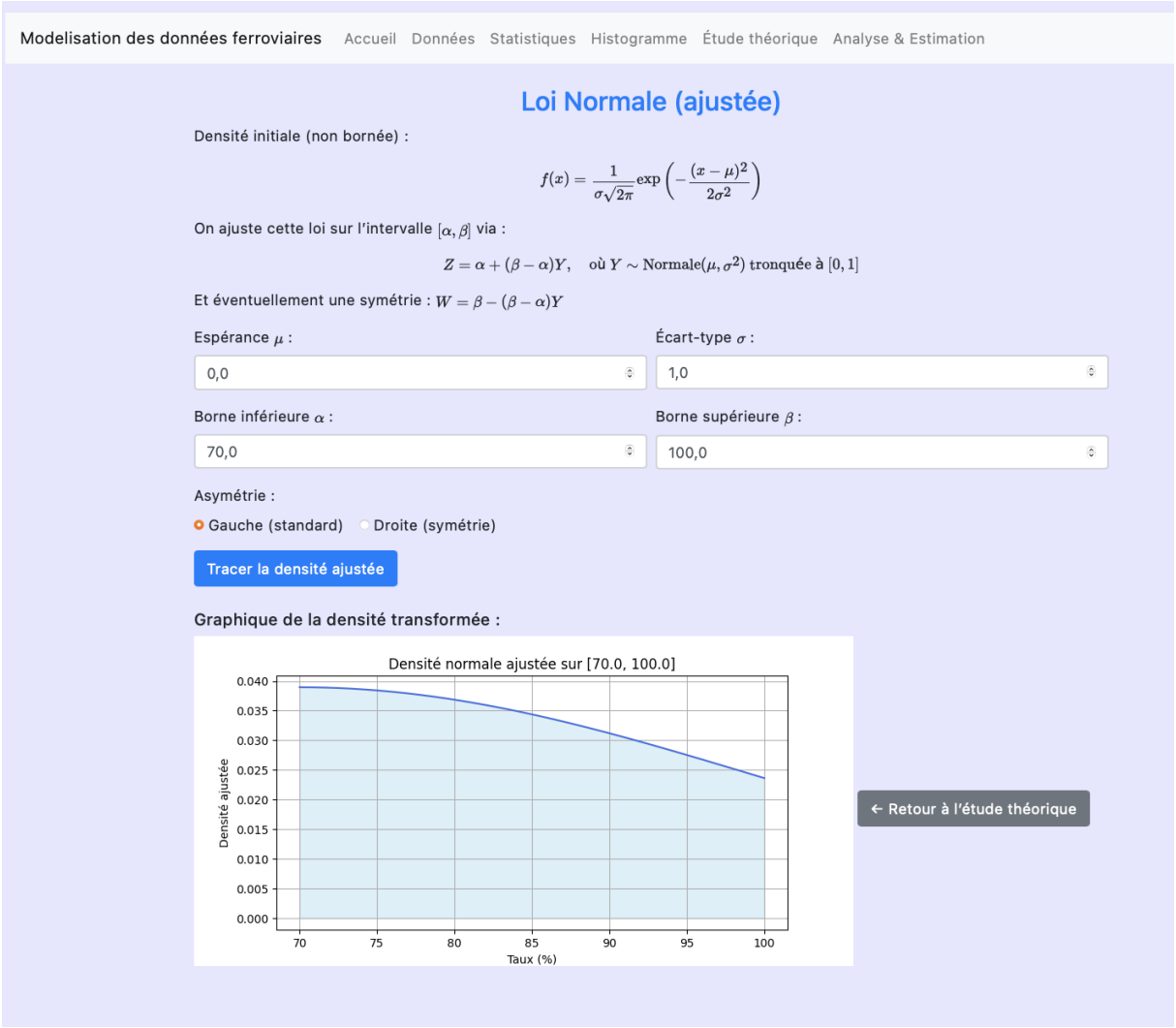


FIGURE 3.2 – Page pédagogique de l'application permettant de tracer la loi Normale et d'en observer les variations en fonction de μ et σ.

3.2 Implemented solution

3.2.1 Technical details

L'application a été développée en utilisant le framework **Flask**, avec une architecture basée sur deux modules principaux : `app.py` pour la gestion du serveur et des routes, et `utils.py` pour les fonctions statistiques et graphiques.

Chargement des données SNCF Les données ouvertes de la SNCF sont directement importées depuis le portail `https://ressources.data.sncf.com` à l'aide de la bibliothèque `pandas`.

Onglet Données L'onglet *Données* constitue le point d'entrée de l'application. Il permet à l'utilisateur de sélectionner une région, une année et un mois, puis de lancer le calcul des statistiques ou de l'estimation. Ces choix sont mémorisés au moyen d'une session Flask, afin d'être réutilisés dans les autres onglets (Statistiques, Histogramme, Estimation).

FIGURE 3.3 – Onglet *Données* de l'application web : sélection des filtres (région, année, mois).

Calcul de statistiques descriptives Le module `utils.py` contient des fonctions pour calculer les moments et dérivés statistiques (moyenne, variance, skewness, kurtosis). Par exemple, le calcul d'un moment d'ordre r est défini ainsi :

```
def Moment_r(data, r):
    data = [x for x in data if x is not None]
    fonc_r = lambda x: x**r
    S = functools.reduce(lambda x, y: x + y, map(fonc_r, data))
    return S / len(data)
```

À partir de ces fonctions, les moments centrés, l'asymétrie (skewness) et l'aplatissement (kurtosis) sont déduits, permettant d'alimenter la page `statistiques.html`.

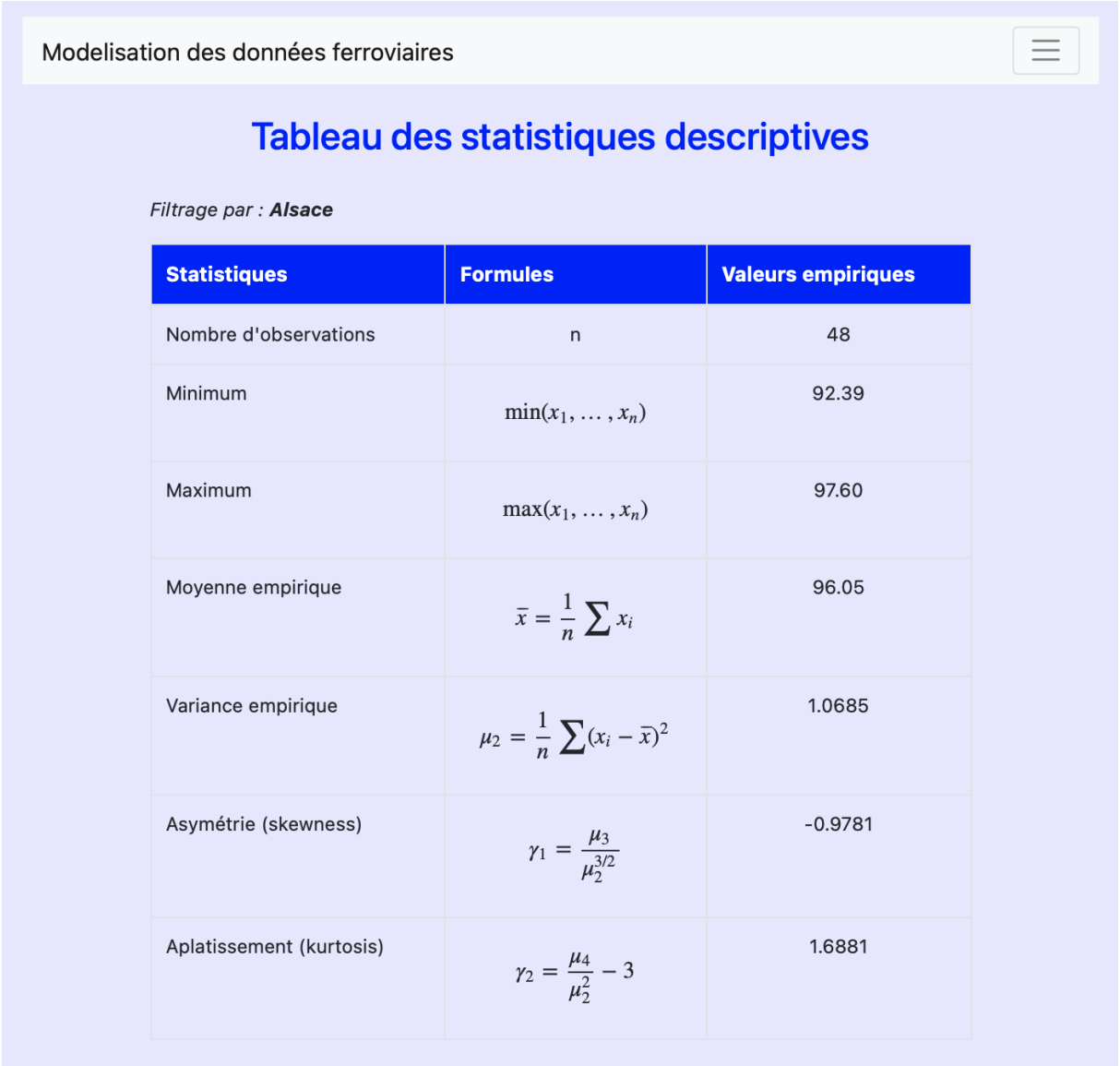


FIGURE 3.4 – Page *Statistiques descriptives* affichant les moments empiriques calculés à partir des données SNCF (moyenne, variance, asymétrie, aplatissement) pour la région Alsace.

Génération des histogrammes Les histogrammes sont générés avec `matplotlib`, puis encodés en base64 afin d'être affichés directement dans les pages HTML :

```
def Histo_Continue64(data, k=7):
    import numpy as np
    import matplotlib.pyplot as plt
    from base64 import b64encode
    import os

    plt.switch_backend('agg')
    plt.rcParams['hatch.color'] = [0.9, 0.9, 0.9]
```

```
data = np.array([x for x in data])
min_val = np.min(data)
max_val = 100  # borne sup rieure fix e

# Borne des classes entre min(data) et 100
Ext = [min_val + (max_val - min_val) * i / k for i in range(k +
    1)]
C = [0.5 * (Ext[i] + Ext[i + 1]) for i in range(k)]

# Effectifs des classes
NN = [((Ext[i] <= data) & (data <= Ext[i + 1])).sum() for i in
   range(k)]
indice_max = [i for i, n in enumerate(NN) if n == max(NN)]

TT = [f"{t:.4f}" for t in Ext]  #  tiquettes  des bornes

fig = plt.figure(figsize=(10, 7))
ax1 = fig.add_subplot(111)

ax1.spines['right'].set_visible(False)
ax1.spines['top'].set_visible(False)
ax1.spines['left'].set_visible(False)
ax1.xaxis.set_ticks_position('bottom')
ax1.set_yticks([])
largeur = Ext[1] - Ext[0]

for i in range(k):
    if i in indice_max:
        ax1.bar(C[i], NN[i], largeur, color=[0.15, 0.15, 0.80],
            edgecolor="white",
                hatch="/", lw=1., zorder=0, alpha=0.9)
    else:
        ax1.bar(C[i], NN[i], largeur, align='center', color=np.
           random.rand(3), edgecolor="white")

    ax1.text(C[i], NN[i], f"{NN[i]}", fontsize=9, style='italic
       ',
             ha='center', va='bottom')

ax1.axvline(x=np.mean(data), color='red', lw=4, label='valeur␣
   moyenne')
```

```
    ax1.set_xticks(Ext)
    ax1.set_xticklabels(TT, fontsize=9, rotation=45)
    ax1.set_xlim(min_val - 0.75 * largeur, max_val + 0.75 * largeur
        )
    ax1.set_ylim(0.0, np.max(NN) + 3.0)
    ax1.set_xlabel("Valeurs", fontsize=13, labelpad=0)
    ax1.set_ylabel("Effectifs", fontsize=14)

    plt.legend(loc='upper left')

    # Sauvegarde en base64
    plt.savefig('histo64.png')
    plt.close()
    with open('histo64.png', 'rb') as plot_file:
        base64_string = b64encode(plot_file.read()).decode()
    os.remove("histo64.png")

    return base64_string
```

Le résultat est intégré dans la page `histogramme.html` au moyen de la syntaxe Jinja2 :

```
<img src="{{image_file}}"\>
```

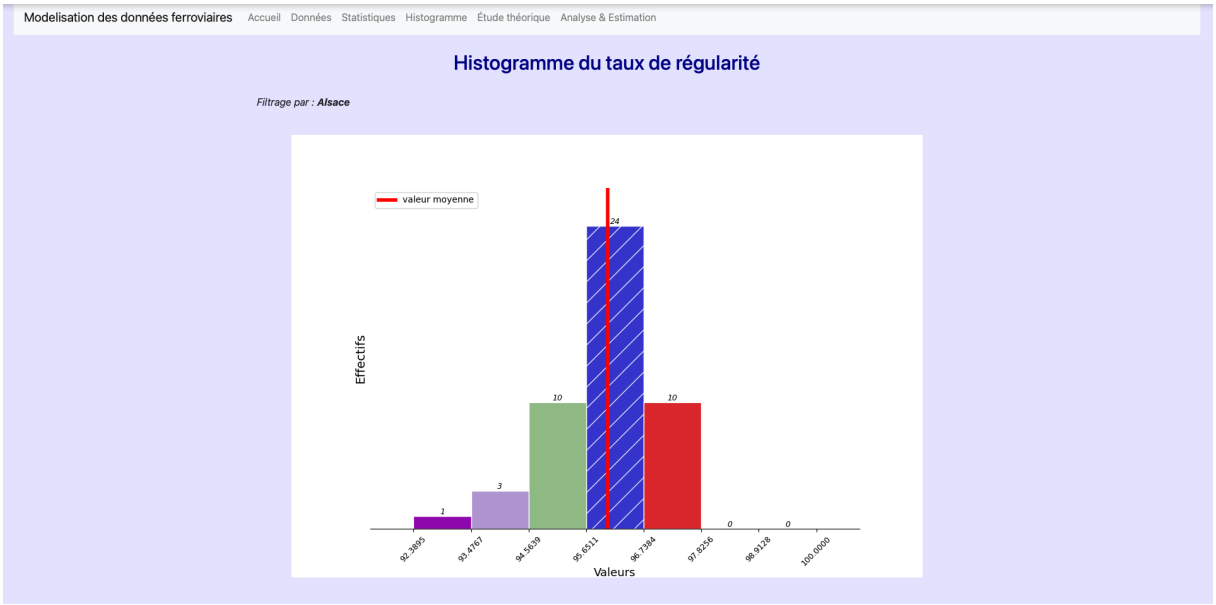


FIGURE 3.5 – Exemple d'histogramme du taux de régularité pour la région Alsace généré par l'application web, permettant de visualiser la distribution empirique des données.

Encodage base64 Les figures produites par l'application (histogrammes, densités estimées, comparaisons de lois) sont automatiquement encodées en base64 et intégrées direc-

tement dans les pages HTML. Cela permet d'obtenir un affichage immédiat et d'éviter la gestion de fichiers d'images intermédiaires.

Estimation par maximum de vraisemblance L'ajustement des lois (tronquée si nécessaire) de probabilité (normale, gamma, bêta) repose sur la maximisation de la log-vraisemblance. La bibliothèque `scipy.optimize` est utilisée pour optimiser les paramètres.

Nous regroupons l'ensemble du flux d'ajustement et de visualisation dans une fonction dédiée : nettoyage des données, normalisation sur $[0, 1]$, estimation par maximum de vraisemblance des paramètres des lois candidates (Normale tronquée, Bêta, Gamma tronquée), calcul des log-vraisemblances comparables sur l'échelle originale $[\min(X), 100]$, construction d'un histogramme normalisé en densité et sur-tracé des densités ajustées.

```
def ajustements_et_traces(data, k=7):
    data = np.array(data)
    data = data[~np.isnan(data)]

    if len(data) < 10:
        return None, None, None

    alpha = np.min(data)
    beta_lim = 100

    data_scaled = (data - alpha) / (beta_lim - alpha)
    data_scaled = data_scaled[(data_scaled > 1e-8) & (data_scaled <
        1 - 1e-8)]

    if len(data_scaled) < 10:
        return None, None, None

    # Estimations EMV
    mu, sigma = emv_normale(data_scaled)
    a_b, b_b = emv_beta(data_scaled)
    a_g, b_g = emv_gamma(data_scaled)

    # Densit s
    d_norm = np.maximum(densite_Z_Normale(data, mu, sigma**2, alpha
       , beta_lim), 1e-300)
    d_beta = np.maximum(densite_Z_Beta(data, a_b, b_b, alpha,
       beta_lim), 1e-300)
    d_gamma = np.maximum(densite_Z_gamma(data, a_g, b_g, alpha,
       beta_lim), 1e-300)
```

```
    # Log-vraisemblances
    logvrais_norm = np.sum(np.log(d_norm))
    logvrais_beta = np.sum(np.log(d_beta))
    logvrais_gamma = np.sum(np.log(d_gamma))

    # Histogramme en densit
    Ext = [alpha + (beta_lim - alpha) * i / k for i in range(k + 1)
       ]
    C = [(Ext[i] + Ext[i + 1]) / 2 for i in range(k)]
    largeur = Ext[1] - Ext[0]
    NN = [((Ext[i] <= data) & (data <= Ext[i + 1])).sum() for i in
       range(k)]
    indice_max = [i for i, n in enumerate(NN) if n == max(NN)]
    TT = ["{:.4f}".format(t) for t in Ext]

    fig, ax1 = plt.subplots(figsize=(10, 7))
    ax1.yaxis.set_ticks_position('left')
    ax1.xaxis.set_ticks_position('bottom')

    for i in range(k):
        height = NN[i] / (len(data) * largeur)
        color = [0.15, 0.15, 0.80] if i in indice_max else np.
           random.rand(3)
        hatch = "/" if i in indice_max else None
        ax1.bar(C[i], height, largeur, color=color, edgecolor="
           white", hatch=hatch, alpha=0.9 if i in indice_max else
           1)

    x_vals = np.linspace(alpha, beta_lim, 500)
    ax1.plot(x_vals, densite_Z_Normale(x_vals, mu, sigma**2, alpha,
        beta_lim), label="Normale", color="blue")
    ax1.plot(x_vals, densite_Z_Beta(x_vals, a_b, b_b, alpha,
       beta_lim), label="Beta", color="green")
    ax1.plot(x_vals, densite_Z_gamma(x_vals, a_g, b_g, alpha,
       beta_lim), label="Gamma", color="orange")

    ax1.axvline(np.mean(data), color='red', lw=2, label='Moyenne')
    ax1.set_xticks(Ext)
    ax1.set_xticklabels(TT, rotation=45, fontsize=9)
    ax1.set_xlim(alpha - 0.75 * largeur, beta_lim + 0.75 * largeur)
    ax1.set_xlabel("Valeurs", fontsize=13)
    ax1.set_ylabel("Densit ", fontsize=14)
    ax1.legend()
```

```
    # Export base64
    plt.tight_layout()
    plt.savefig("estimation_plot.png")
    plt.close()
    with open("estimation_plot.png", "rb") as f:
        plot_encoded = b64encode(f.read()).decode()
    os.remove("estimation_plot.png")

    resultats = {
        "normale": {"logvrais": logvrais_norm, "mu": mu, "sigma":
            sigma},
        "beta": {"logvrais": logvrais_beta, "alpha": a_b, "beta":
            b_b},
        "gamma": {"logvrais": logvrais_gamma, "alpha": a_g, "lambda
            ": b_g}
    }

    return resultats, len(data), plot_encoded
```

Ces estimations alimentent la page `estimation.html`, où un tableau compare les valeurs de log-vraisemblance pour les trois lois candidates.

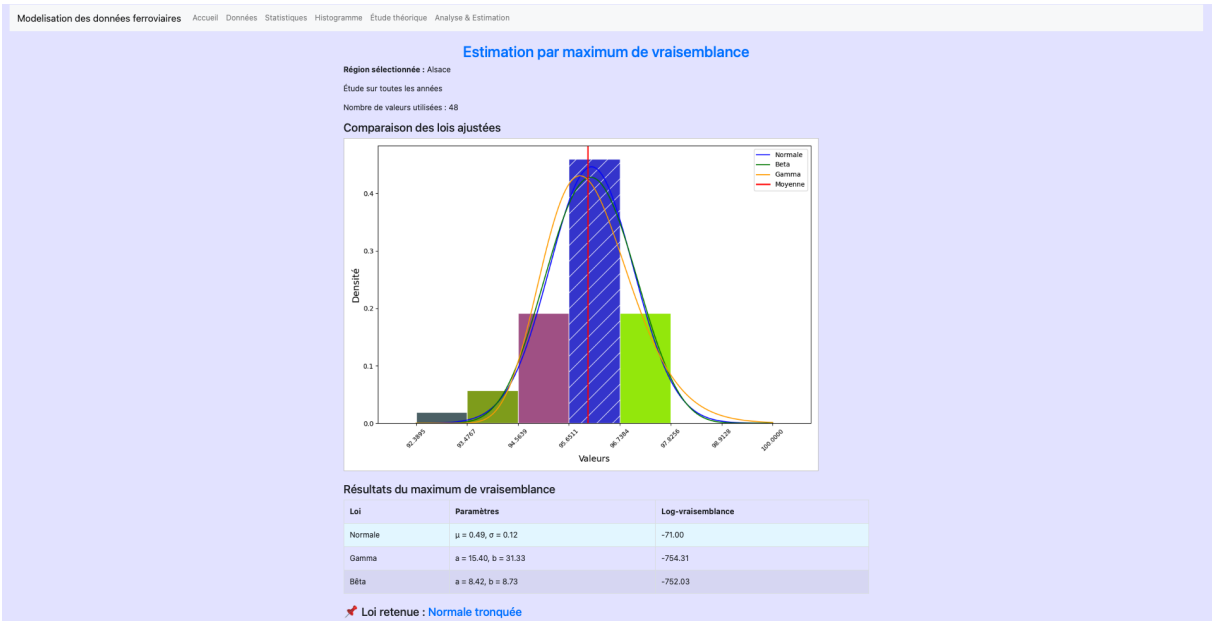


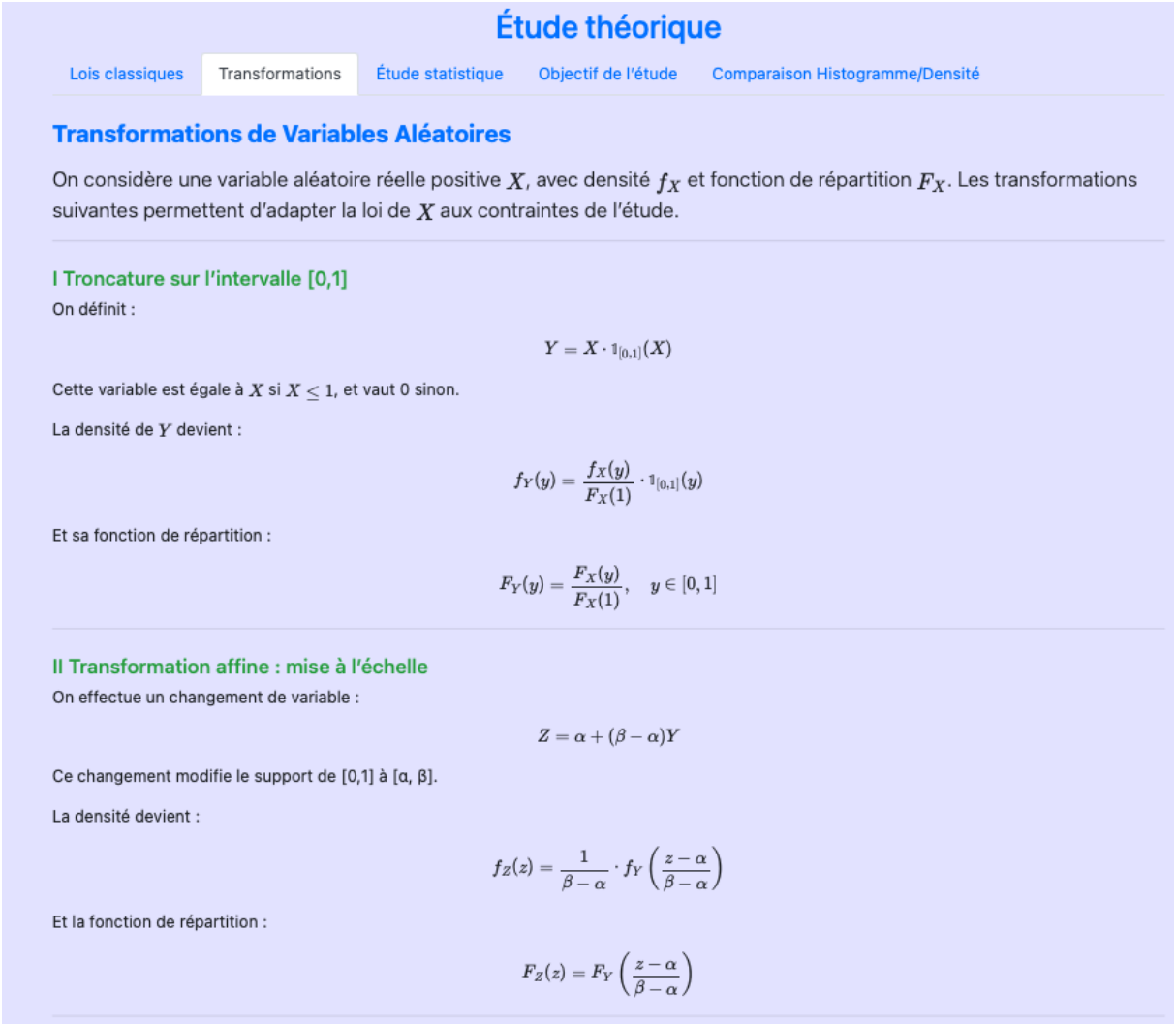
FIGURE 3.6 – Page *Analyse et estimation* comparant l'ajustement des lois Normale, Gamma et Bêta par la méthode du maximum de vraisemblance (MLE).

Choix du modèle par log-vraisemblance

Pour comparer les lois candidates (Normale, Gamma, Bêta), nous avons adopté le critère de la **log-vraisemblance maximale** : la loi retenue est celle qui présente la plus grande log-vraisemblance calculée à partir des données observées. Ce critère est particulièrement pertinent car la log-vraisemblance est plus simple à manipuler et plus stable numériquement que la vraisemblance elle-même. Ainsi, le modèle choisi correspond à la loi qui reproduit le mieux la distribution empirique des taux de régularité.

Transformation de la variable aléatoire

Une étape importante du travail a consisté à appliquer une transformation de variable pour rendre les données compatibles avec les lois théoriques. nous avons normalisé les observations en appliquant la transformation :



Étude théorique

Lois classiques | Transformations | Étude statistique | Objectif de l'étude | Comparaison Histogramme/Densité

Transformations de Variables Aléatoires

On considère une variable aléatoire réelle positive X, avec densité f_X et fonction de répartition F_X. Les transformations suivantes permettent d'adapter la loi de X aux contraintes de l'étude.

I Troncature sur l'intervalle [0,1]

On définit :

$$Y = X \cdot \mathbb{1}_{[0,1]}(X)$$

Cette variable est égale à X si $X \leq 1$, et vaut 0 sinon.

La densité de Y devient :

$$f_Y(y) = \frac{f_X(y)}{F_X(1)} \cdot \mathbb{1}_{[0,1]}(y)$$

Et sa fonction de répartition :

$$F_Y(y) = \frac{F_X(y)}{F_X(1)}, \quad y \in [0,1]$$

II Transformation affine : mise à l'échelle

On effectue un changement de variable :

$$Z = \alpha + (\beta - \alpha)Y$$

Ce changement modifie le support de [0,1] à [α, β].

La densité devient :

$$f_Z(z) = \frac{1}{\beta - \alpha} \cdot f_Y\left(\frac{z - \alpha}{\beta - \alpha}\right)$$

Et la fonction de répartition :

$$F_Z(z) = F_Y\left(\frac{z - \alpha}{\beta - \alpha}\right)$$

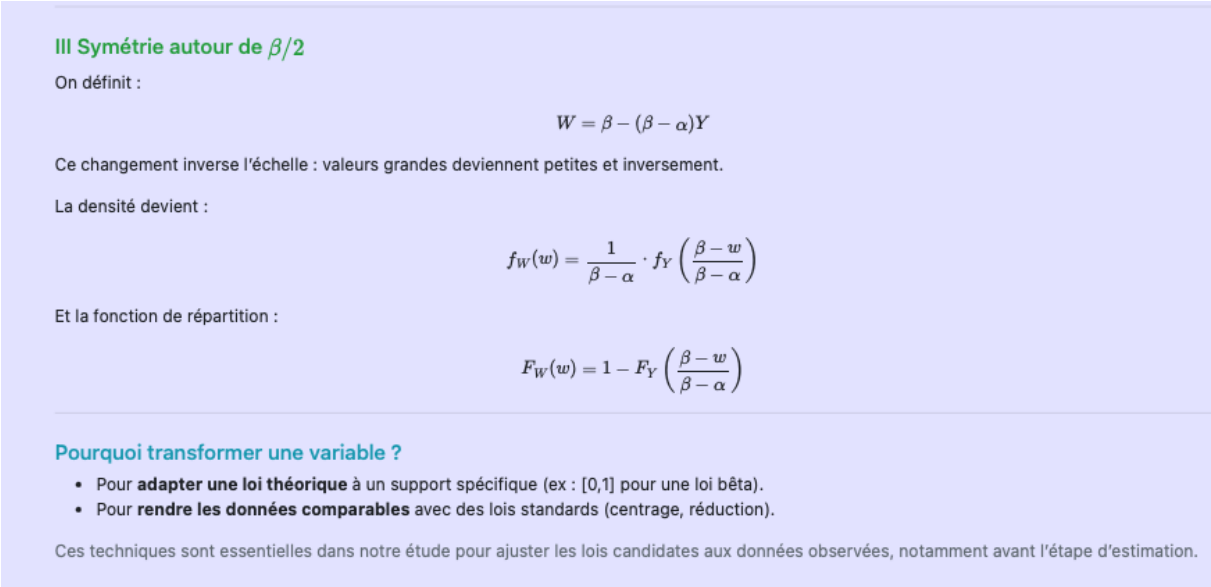


FIGURE 3.7 – Page de l'application web illustrant la transformation de la variable aléatoire afin d'adapter les données (bornées entre min(X) et 100%) aux lois de probabilité étudiées.

Intervalle des histogrammes

Dans la représentation graphique, nous avons choisi de tracer les histogrammes entre la plus petite valeur observée et 100%. Ce choix est cohérent avec la nature des données (taux de régularité exprimés en pourcentage), et permet de mieux comparer visuellement l'adéquation des lois ajustées aux observations.

Remarque méthodologique

Il convient de noter que le choix du modèle n'est pas limité au seul critère de la log-vraisemblance. Dans d'autres contextes, on pourrait prendre en compte des critères supplémentaires comme l'Akaike Information Criterion (AIC), le Bayesian Information Criterion (BIC), ou encore la qualité de l'ajustement visuel sur les histogrammes. Dans le cadre de ce stage, nous avons retenu la log-vraisemblance comme critère principal pour sa simplicité et sa pertinence sur les données étudiées.

3.2.2 Results of experiments or technical tests

L'application a été testée sur les données ouvertes de la SNCF, couvrant la régularité des trains TER depuis janvier 2013. Plusieurs expériences ont été menées afin de valider les fonctionnalités et d'illustrer les résultats obtenus.

Statistiques descriptives Pour un filtre donné (par exemple : région Île-de-France, année 2019, mois de janvier), l'application calcule automatiquement le nombre d'observa-

tions, la moyenne, la variance, l'asymétrie et l'aplatissement. Ces résultats sont affichés dans un tableau clair (voir Figure 3.4).

Histogrammes La visualisation graphique via histogrammes permet de mettre en évidence la forme des distributions. Les utilisateurs peuvent comparer ces histogrammes à des densités théoriques ajustées, après normalisation (voir Figure 3.5).

Estimation des lois L'outil permet d'ajuster automatiquement les lois Normale, Gamma et Bêta aux données filtrées, via la méthode du maximum de vraisemblance (MLE). Un tableau de synthèse compare les paramètres estimés et les valeurs de log-vraisemblance, afin de retenir la loi la plus adaptée (voir Figure 3.6).

3.2.3 Discussion of contributions

Le développement de cette application a permis de répondre à plusieurs objectifs, au-delà de la simple analyse statistique des données.

Ergonomie et accessibilité L'outil propose une interface web claire et intuitive, accessible via un simple navigateur. Grâce à la séparation entre le code Python (back-end) et les templates HTML (front-end), les utilisateurs peuvent interagir avec les données sans connaissances techniques particulières.

Dimension pédagogique Une attention particulière a été portée à l'aspect pédagogique. Les formules mathématiques sont affichées en LaTeX grâce à `MathJax`, et chaque méthode statistique est accompagnée d'une explication textuelle. Ainsi, l'application ne se limite pas à fournir des résultats, mais constitue aussi un support d'apprentissage sur les concepts de moments, de vraisemblance et d'ajustement de lois de probabilité.

Modularité logicielle La séparation claire entre `app.py` (gestion des routes Flask), `utils.py` (fonctions statistiques et graphiques) et le dossier `templates/` assure une grande modularité. Cette organisation facilite l'extension future du projet, par exemple en ajoutant d'autres lois de probabilité ou de nouveaux indicateurs statistiques.

Mise en ligne et valorisation des données L'application a été déployée sur la plateforme `PythonAnywhere`, rendant l'outil disponible en ligne (`http://sondieh.pythonanywhere.com`). Ce déploiement constitue une véritable valorisation des données publiques de la SNCF : il rend leur exploration et leur analyse accessibles à un public plus large, allant des chercheurs aux étudiants, voire aux usagers curieux.

Apports personnels Enfin, ce stage a été l'occasion de consolider plusieurs compétences :

- en **programmation web**, via l'utilisation du framework Flask et du langage HTML/CSS ;
- en **statistiques appliquées**, par la mise en œuvre de méthodes d'estimation et de modélisation ;
- en **gestion de projet**, grâce à la planification et à la structuration d'une application complète, depuis la collecte des données jusqu'à la mise en production.

Chapitre 4

Conclusion and Next Steps

4.1 Conclusion

Ce stage au sein du département de Mathématiques et Statistiques du CNAM a permis de développer une application web interactive dédiée à l'analyse descriptive et inférentielle des données de régularité des trains TER fournies par la SNCF.

L'objectif fixé — concevoir un outil capable d'explorer les données, de calculer des indicateurs statistiques, et d'ajuster des lois de probabilité — a été atteint. Le projet a combiné des compétences en programmation Python, en statistiques appliquées et en développement web, tout en intégrant une dimension pédagogique par la présentation des concepts théoriques et des formules mathématiques.

4.2 Lessons learned

Ce stage m'a apporté plusieurs enseignements majeurs :

- **Compétences techniques** : maîtrise renforcée du langage Python, de la bibliothèque `pandas` pour l'analyse de données, de `matplotlib` pour la visualisation, ainsi que du framework `Flask` pour le déploiement web.
- **Compétences statistiques** : application concrète de méthodes d'inférence (estimation par maximum de vraisemblance), comparaison de lois de probabilité et interprétation de mesures comme la skewness et le kurtosis.
- **Compétences transversales** : gestion d'un projet complet, organisation du code de manière modulaire, documentation des résultats et valorisation par la mise en ligne.
- **Gestion des sessions web** : une difficulté particulière a été rencontrée concernant la conservation des filtres sélectionnés (par exemple la région) lors de la navigation entre les pages. Ce problème a été résolu en utilisant le mécanisme de `session` de Flask, avec une clé secrète définie dans `app.py`. Cette solution m'a permis de mieux comprendre les concepts liés à la persistance des données dans les applications web.

4.3 Next steps

Le projet pourrait être enrichi et prolongé selon plusieurs axes :

— **Extension des modèles** : intégrer d'autres lois de probabilité (log-normale, Weibull, etc.) afin de comparer un spectre plus large de modèles.

— **Analyse prédictive** : utiliser des méthodes de prévision (séries temporelles, modèles ARIMA ou machine learning) pour anticiper la régularité future des trains.

— **Amélioration de l'interface** : offrir de nouvelles fonctionnalités interactives (choix dynamique des intervalles, visualisation multi-régions).

— **Mise en production avancée** : déployer l'application sur un serveur plus robuste ou dans un environnement cloud, avec une base de données actualisée automatiquement.

— **Application à d'autres jeux de données SNCF** : la méthodologie développée (analyse descriptive et ajustement de lois) pourrait être appliquée à d'autres données publiques disponibles sur le portail SNCF, telles que le nombre d'accidents, les retards voyageurs, ou la fréquentation des gares. Cela permettrait d'étendre l'outil à des problématiques de sécurité et de qualité de service.

Ainsi, ce travail constitue une première étape vers un outil complet d'analyse statistique appliqué au domaine des transports ferroviaires, conciliant rigueur scientifique et accessibilité.

Annexe A

DD&RS - Sustainable development and Social Responsibility

Le projet s'inscrit dans une logique de valorisation des données publiques ouvertes (open data), ce qui favorise la transparence et l'accessibilité de l'information pour les citoyens. En outre, l'analyse de la régularité ferroviaire participe indirectement à la réflexion sur la qualité et la durabilité des transports en commun, en encourageant leur usage comme alternative à la voiture individuelle. Le déploiement sur une plateforme en ligne (PythonAnywhere) réduit également la consommation de ressources matérielles, en mutualisant l'infrastructure serveur.

Annexe B

Technical details

Cette annexe regroupe les éléments techniques complémentaires :
— extraits de code Python (`app.py`, `utils.py`) ;

Flask app skeleton & session handling

Listing B.1 – Application Flask : structure de base et gestion de session

```
app = Flask(__name__)
app.secret_key = "12345"  # Cl  pour session

ssl._create_default_https_context = ssl._create_unverified_context

# Chargement des donn es
adresse_TER = "https://ressources.data.sncf.com/explore/dataset/
   regularite-mensuelle-ter/download/?format=csv&timezone=Europe/
   Berlin&lang=fr&use_labels_for_header=true&csv_separator=%3B"
df_TER = pd.read_csv(adresse_TER, sep=';')
df_TER['Date'] = pd.to_datetime(df_TER['Date'], errors='coerce')
df_TER['Ann e'] = df_TER['Date'].dt.year
df_TER['Mois'] = df_TER['Date'].dt.to_period('M').astype(str)
df_TER = df_TER.sort_values(['R gion', 'Date'])

@app.route('/reset')
def reset():
    session.pop('region', None)
    session.pop('annee', None)
    session.pop('mois', None)
    return redirect('/donnees')
```

Accueil (/)

Listing B.2 – Route d'accueil : présentation du stage et lien vers les données

```
@app.route("/")
def accueil():
    return render_template("accueil.html", title="Accueil")
```

Données (/donnees)

Listing B.3 – Route Données : filtres région/année/mois et tableau filtré

```
@app.route('/donnees', methods=['GET', 'POST'])
def donnees():
    regions = sorted(df_TER['R gion'].dropna().unique())
    annees = sorted(df_TER['Ann e'].dropna().unique())
    mois_list = sorted(df_TER['Mois'].dropna().unique())

    if request.method == 'POST':
        selected_region = request.form.get('region')
        selected_annee = request.form.get('annee')
        selected_mois = request.form.get('mois')
        action = request.form.get('action')

        if action == 'calculer_stats':
            session['region'] = selected_region
            session['annee'] = selected_annee
            session['mois'] = selected_mois
    else:
        selected_region = session.get('region')
        selected_annee = session.get('annee')
        selected_mois = session.get('mois')

    filtered = df_TER.copy()

    if selected_region:
        filtered = filtered[filtered['R gion'] == selected_region]
    if selected_annee:
        filtered = filtered[filtered['Ann e'] == int(
           selected_annee)]
    if selected_mois:
        filtered = filtered[filtered['Mois'] == selected_mois]
```

```
    if filtered.empty:
        filtered = None

    nb_lignes = len(filtered) if filtered is not None else 0

    return render_template("donnees.html",
                           title="Donn es",
                           regions=regions,
                           annees=annees,
                           mois_list=mois_list,
                           selected_region=selected_region,
                           selected_annee=selected_annee,
                           selected_mois=selected_mois,
                           filtered=filtered,
                           nb_lignes=nb_lignes)
```

Statistiques (/statistiques)

Listing B.4 – Route Statistiques : moments empiriques (n, min, max, mean, var, skew, kurt)

```
@app.route('/statistiques')
def statistiques():
    from utils import Moment_cr, Moment_r  # Import de tes
       fonctions perso

    selected_region = session.get('region')
    selected_annee = session.get('annee')
    selected_mois = session.get('mois')

    # On filtre dynamiquement
    filtered = df_TER.copy()
    if selected_region:
        filtered = filtered[filtered['R gion'] == selected_region]
    if selected_annee:
        filtered = filtered[filtered['Ann e'] == int(
           selected_annee)]
    if selected_mois:
        filtered = filtered[filtered['Mois'] == selected_mois]

    data_col = filtered['Taux␣de␣r gularit '].dropna().tolist()
```

```
    n = len(data_col)

    if n > 0:
        minimum = min(data_col)
        maximum = max(data_col)
        moyenne = Moment_r(data_col, 1)
        variance = Moment_cr(data_col, 2)
        skewness = Moment_cr(data_col, 3) / (variance ** 1.5) if
           variance else None
        kurtosis = Moment_cr(data_col, 4) / (variance ** 2) - 3 if
           variance else None
    else:
        minimum = maximum = moyenne = variance = skewness =
           kurtosis = None

    return render_template("statistiques.html",
                           title="Statistiques",
                           selected_region=selected_region,
                           selected_annee=selected_annee,
                           selected_mois=selected_mois,
                           n=n,
                           minimum=minimum,
                           maximum=maximum,
                           moyenne=moyenne,
                           variance=variance,
                           skewness=skewness,
                           kurtosis=kurtosis)
```

Histogramme (/histogramme)

Listing B.5 – Route Histogramme : génération PNG encodé base64 pour affichage

```
@app.route('/histogramme')
def histogramme():


    selected_region = session.get('region')
    selected_annee = session.get('annee')
    selected_mois = session.get('mois')


    filtered = df_TER.copy()
```

```
    if selected_region:
        filtered = filtered[filtered['R gion'] == selected_region]
    if selected_annee and selected_annee.isdigit():
        filtered = filtered[filtered['Ann e'] == int(
           selected_annee)]
    if selected_mois:
        filtered = filtered[filtered['Mois'] == selected_mois]

    data = filtered['Taux de r gularit '].dropna().tolist()
    if not data:
        return render_template("histogramme.html",
                               title="Histogramme",
                               image_file=None,
                               selected=True,
                               selected_region=selected_region,
                               selected_annee=selected_annee,
                               selected_mois=selected_mois)



    plot_html=Histo_Continue64(data,7)

    return render_template("histogramme.html",
                           title="Histogramme",
                           image_file=plot_html,
                           selected=True,
                           selected_region=selected_region,
                           selected_annee=selected_annee,
                           selected_mois=selected_mois)
```

Étude théorique (/etude et sous-pages)

Listing B.6 – Routes Étude théorique : hub + pages Normale, Gamma, Bêta avec tracés

```
@app.route("/etude")

@app.route('/etude', methods=['GET', 'POST'])
def etude():
    #par d faut
    mu = 0
    sigma = 1
    image_base64 = None
```

```
    if request.method == 'POST':
        try:
            mu = float(request.form.get('mu', 0))
            sigma = float(request.form.get('sigma', 1))
            #G n ration du graphe
            x = np.linspace(mu - 4*sigma, mu + 4*sigma, 1000)
            y = norm.pdf(x, mu, sigma)

            fig, ax = plt.subplots(figsize=(8, 4))
            ax.plot(x, y, color='blue')
            ax.fill_between(x, y, alpha=0.3)
            ax.axvline(mu, color='red', linestyle='--', label='  ')
            ax.set_title(f'Densit  normale pour    =  {mu},     =  {
               sigma}')
            ax.legend()

            buf = BytesIO()
            plt.savefig(buf, format="png")
            plt.close()
            buf.seek(0)
            image_base64 = base64.b64encode(buf.read()).decode('utf
               -8')

        except ValueError:
            mu, sigma = 0, 1  #en cas   d erreur

    return render_template("etude.html", title=" tude  Th orique",
        mu=mu, sigma=sigma, image_base64=image_base64)



@app.route('/etude/normale', methods=['GET', 'POST'])
def etude_normale():
    mu = 0
    sigma = 1
    alpha = 70
    beta_borne = 100
    asymetrie = "gauche"
    image_base64 = None

    if request.method == 'POST':
        try:
```

```
            mu = float(request.form.get('mu', 0))
            sigma = float(request.form.get('sigma', 1))
            alpha = float(request.form.get('alpha', 70))
            beta_borne = float(request.form.get('beta', 100))
            asymetrie = request.form.get('asymetrie', 'gauche')

            x = np.linspace(alpha, beta_borne, 300)
            if asymetrie == "droite":
                y = [densite_W_Normale(w, mu, sigma**2, alpha,
                    beta_borne) for w in x]
            else:
                y = [densite_Z_Normale(z, mu, sigma**2, alpha,
                    beta_borne) for z in x]

            fig, ax = plt.subplots(figsize=(8, 4))
            ax.plot(x, y, color='royalblue')
            ax.fill_between(x, y, alpha=0.3, color='skyblue')
            ax.set_title(f'Densit␣normale␣ajust e␣sur␣[{alpha},␣
                {beta_borne}]')
            ax.set_xlabel("Taux␣(%)")
            ax.set_ylabel("Densit␣ajust e")
            ax.grid()

            buf = io.BytesIO()
            plt.savefig(buf, format="png")
            plt.close()
            buf.seek(0)
            image_base64 = base64.b64encode(buf.read()).decode('utf
                -8')

        except Exception as e:
            print("Erreur␣:", e)

    return render_template("etude_normale.html",
                           mu=mu,
                           sigma=sigma,
                           alpha=alpha,
                           beta=beta_borne,
                           asymetrie=asymetrie,
                           image_base64=image_base64)


from scipy.stats import gamma
```

```
@app.route('/etude/gamma', methods=['GET', 'POST'])
def etude_gamma():
    a = 2
    b = 1
    alpha = 70
    beta_borne = 100
    asymetrie = "gauche"
    image_base64 = None

    if request.method == 'POST':
        try:
            a = float(request.form.get('a', 2))
            b = float(request.form.get('b', 1))
            alpha = float(request.form.get('alpha', 70))
            beta_borne = float(request.form.get('beta', 100))
            asymetrie = request.form.get('asymetrie', 'gauche')

            x = np.linspace(alpha, beta_borne, 300)
            if asymetrie == "droite":
                y = [densite_W_gamma(w, a, b, alpha, beta_borne)
                    for w in x]
            else:
                y = [densite_Z_gamma(z, a, b, alpha, beta_borne)
                    for z in x]

            fig, ax = plt.subplots(figsize=(8, 4))
            ax.plot(x, y, color='darkgreen')
            ax.fill_between(x, y, alpha=0.3, color='lightgreen')
            ax.set_title(f'Densit␣Gamma␣ajust e␣sur␣[{alpha},␣{
                beta_borne}]')
            ax.set_xlabel("Taux␣(%)")
            ax.set_ylabel("Densit␣ajust e")
            ax.grid()

            buf = io.BytesIO()
            plt.savefig(buf, format="png")
            plt.close()
            buf.seek(0)
            image_base64 = base64.b64encode(buf.read()).decode('utf
                -8')

        except Exception as e:
            print("Erreur␣:", e)
```

```
    return render_template("etude_gamma.html",
                           a=a,
                           b=b,
                           alpha=alpha,
                           beta=beta_borne,
                           asymetrie=asymetrie,
                           image_base64=image_base64)


from scipy.stats import beta
@app.route('/etude/beta', methods=['GET', 'POST'])
def etude_beta():
    a = 2.0
    b_param = 5.0
    alpha = 70
    beta_lim = 100
    asymetrie = "gauche"
    image_base64 = None

    if request.method == 'POST':
        try:
            a = float(request.form.get('a', 2.0))
            b_param = float(request.form.get('b', 5.0))
            alpha = float(request.form.get('alpha', 70))
            beta_lim = float(request.form.get('beta', 100))
            asymetrie = request.form.get('asymetrie', 'gauche')

            x = np.linspace(alpha, beta_lim, 300)
            if asymetrie == "droite":
                y = [densite_W_Beta(w, a, b_param, alpha, beta_lim)
                    for w in x]
            else:
                y = [densite_Z_Beta(z, a, b_param, alpha, beta_lim)
                    for z in x]

            fig, ax = plt.subplots(figsize=(8, 4))
            ax.plot(x, y, color='darkorange')
            ax.fill_between(x, y, alpha=0.3, color='orange')
            ax.set_title(f'Densit␣B ta␣ajust e␣sur␣[{alpha},␣{
               beta_lim}]')
            ax.set_xlabel("Taux␣(%)")
            ax.set_ylabel("Densit␣ajust e")
```

```
            ax.grid()

            buf = io.BytesIO()
            plt.savefig(buf, format="png")
            plt.close()
            buf.seek(0)
            image_base64 = base64.b64encode(buf.read()).decode('utf
                -8')

        except Exception as e:
            print("Erreur :", e)

    return render_template("etude_beta.html",
                           a=a,
                           b=b_param,
                           alpha=alpha,
                           beta=beta_lim,
                           asymetrie=asymetrie,
                           image_base64=image_base64)
```

Analyse & Estimation (/estimation)

Listing B.7 – Route Estimation : MLE et comparaison des lois (log-vraisemblances)

```
@app.route('/estimation')
def estimation():
    from utils import ajustements_et_traces

    selected_region = session.get('region')
    if not selected_region:
        return render_template("estimation.html", title="Estimation
            ", erreur="Veuillez s lectionner une r gion.")

    filtered = df_TER[df_TER['R gion'] == selected_region]
    data = filtered['Taux de r gularit '].dropna().tolist()


    resultats, n, image = ajustements_et_traces(data)


    if resultats is None:
```

```
        return render_template("estimation.html", title="Estimation
            ",
                                erreur="Pas assez de donn es (au
                                    moins 10 requises).",
                                selected_region=selected_region,
                                n=n)

    #chercher la meilleure loi
    meilleure = max(resultats.items(), key=lambda x: x[1]['logvrais
        '])
    meilleure_loi = {
        "nom": meilleure[0].capitalize(),
        "logvrais": meilleure[1]['logvrais'],
        **meilleure[1]
    }

    return render_template("estimation.html", title="Estimation",
                           selected_region=selected_region,
                           resultats=resultats,
                           image=image,
                           n=n,
                           meilleure_loi=meilleure_loi)
```

Bibliographie

[1] D. Ghorbanzadeh, *Exercices de Mathématiques du Signal Aléatoire*, Polycopié interne , Département de Mathématiques, CNAM, Paris, 2017/2018.

[2] SNCF Open Data, *Données de régularité mensuelle des trains TER*, Disponible sur : `https://ressources.data.sncf.com`

[3] Flask Documentation, *Flask Web Development*, Disponible sur : `https://flask.palletsprojects.com`

[4] Wes McKinney, *Python for Data Analysis*, O'Reilly Media, 2ème édition, 2017.

[5] Virtanen, P. et al., *SciPy 1.0 : Fundamental Algorithms for Scientific Computing in Python*, Nature Methods, 17, 261–272, 2020.

[6] Casella, G. & Berger, R. L., *Statistical Inference*, Duxbury, 2ème édition, 2002.

[7] PythonAnywhere Documentation, Disponible sur : `https://help.pythonanywhere.com`