CHAPITRE I. MANIPULATION ET PRÉTRAITEMENT DES DONNÉES

Chargement des modules et fonctions

```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from scipy import stats
import matplotlib.ticker as ticker
from scipy.stats import kendalltau, spearmanr, chi2_contingency, ttest_ind, bartlett, kruskal, mannwhitneyu,f_oneway
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import plotly.express as px
import plotly.graph_objects as go
import warnings
warnings.filterwarnings('ignore')
```

Importation de bases de données

```python
# Base (db_freq)
base_db_freq = pd.read_excel("db_freq.xlsx")
```

```python
# Base(db_prem)
base_db_prem = pd.read_csv("db_prem.csv", sep=";", decimal=",")
```

```python
# Base db_sev
db_sev = pd.read_csv('db_sev.txt', delimiter='\t')
```

TRAITEMENT ET ANALYSE DE LA BASE: "BASE_DB_PREM"

1- Analyse structurelle & Extractions

a) Analyse de format

```python
base1 = base_db_prem.copy()
```

```python
base1.head()
```

```python
base1.shape
```

```python
base1.info()
```

2- Qualité des données : Détection et traitement des anomalies

2.1 Gestion des doublons

```python
print("Le nombre des doublons présent dans la base est de: ", len(base1["IDpol"]) - base1["IDpol"].nunique())
```

```python
print("Le nombre des doublons par 'IDpol' et 'Year'présent dans la base est: ")
len(base1[["IDpol","Year"]]) - base1[["IDpol","Year"]].nunique()
```

```python
print("La somme des doublons purs sans Year est de: ", base1.duplicated(subset=base1.columns.difference(['Year'])).sum())
```

```python
print("Somme des doublons purs est :", sum(base1.duplicated()))
```

```python
print("Somme des doublons par clé primaire est de: ",sum(base1.duplicated(subset = "IDpol")))
```

```python
# Tous les doublons, triés par "IDpol"
base1[base1.duplicated(subset = "IDpol", keep = False)].sort_values(by = "IDpol")
```

```python
print("Suppression des doublons purs en utilisant toutes les colonnes sauf  'Year' et en gardant la première occurrence")
base1.drop_duplicates(subset=[col for col in base1.columns if col != 'Year'], keep='first', inplace=True)
```

```python
base1.shape
```

```python
print("Le nombre des doublons présent dans la base est: ",
      base1.duplicated(subset=base1.columns.difference(['Year'])).sum())
```

2.2 Gestion des données manquantes (DM)

```python
base1.count()
```

```python
print("Le nombre de valeurs manquantes pour chaque variable est affiché ci-dessous de manière décroissante: ")
print()
print(base1.isna().sum().sort_values(ascending=False))
```

```python
print("Le nombre de d'oservation pour chaque variable est affiché ci-dessous de manière décroissante: ")
print()
print(base1.notna().sum().sort_values(ascending=False))
```

```python
print("Le type de données pour chaque variable est affiché ci-dessous: ")
print()
print(base1.dtypes)
```

```python
print("Suppresion des variables 'MaritalStatus','JobCode','Channel', car elles possèdent beaucoup de données manquantes: ")
base1 = base1.drop(["MaritalStatus","JobCode","Channel"], axis=1)
```

3- Imputation univariée

3-1 Imputation univariée des variables quantitatives

```python
print("Imputation des NA par le médian de la variable 'BonusMalus' ")
base1.BonusMalus = base1.BonusMalus.fillna(base1.BonusMalus.median())
```

```python
print("Imputation des NA par la moyenne de la variable 'VehAge'")
base1.VehAge =base1.VehAge.fillna(base1.VehAge.mean())
```

3-2 Imputation univariée des variables qualitatives

```python
print("Le tableau de fréquence des modalités de la variable 'VehGas' est: ")
base1.VehGas.value_counts(dropna=False)
```

```python
print("Imputation par la modalité 'Regular' des NA de la variable 'VehGas'")
base1.VehGas = base1.VehGas.fillna(base1.VehGas.mode()[0])
```

```python
print("Le tableau de fréquence des modalités de la variable 'VehUsage'")
base1.VehUsage.value_counts(dropna=False)
```

```python
print("Imputation par la modalité 'Private+trip to office' ")
base1.loc[:, "VehUsage"] = base1["VehUsage"].fillna(base1["VehUsage"].mode()[0])
```

```python
print("Tableau de fréquence des modalités de la variable 'VehPower'")
base1.VehPower.value_counts(dropna=False)
```

```python
print("Suppression des nan de la colonne 'VehPower', car on ne sait pas par quelle modalité le remplacer (pas modalité dominante)")
base1 = base1.dropna(subset=['VehPower'])
```

```python
print('Verification finale des NaN dans chaque colonne de la base de données:')
base1.isna().sum()
```

4- Gestion des données incohérantes

4-1 Quantitative

```python
print("Les statistiques descriptives des variables numériques:")
base1.DrivAge.describe()
```

```python
print("Le nombre d'occurrences de 'DrivAge'")
base1.DrivAge.value_counts().sort_index()
```

```python
print("Remplacer les valeurs d'âge inférieures à 18 ans ou supérieures à 100 ans par NaN")
base1.loc[(base1['DrivAge'] < 18) | (base1['DrivAge'] > 100), 'DrivAge'] = np.nan
```

```python
print("Imputation des NA par la médiane")
base1.DrivAge= base1.DrivAge.fillna(base1.DrivAge.median())
```

```python
print("Le nombre d'occurrences de 'DrivAg'")
base1.DrivAge.value_counts().sort_index()
```

4-2: Qualtitative

```python
print("Tableau de fréquence des modalités ")
base1.DrivGender.value_counts(dropna = False)
```

```python
print("Imputation de modalités de la variable 'DrivGender'")
base1.DrivGender.replace({'Male': "M", "H" : "M", "Female":"F"},inplace=True)
```

```python
base1.DrivGender.value_counts(dropna = False) # Imputation de modalités ok!
```

TRAITEMENT ET ANALYSE DE LA BASE: "DB_FREQ"

```python
base2 = base_db_freq.copy()
base2.shape
```

```python
base2.head(10)
```

```python
base2.info(1)
```

1- Qualité des données : détection et traitement des anomalies¶

1.1 Doublons

```python
len(base2["IDpol"]) - base2["IDpol"].nunique()
```

```python
base2.duplicated(subset=base2.columns.difference(['Year'])).sum()
```

```python
sum(base2.duplicated())
```

```python
print("Les doublons triés")
base2[base2.duplicated(subset = "IDpol", keep = False)].sort_values(by = "IDpol")
```

```python
# Grouper les données par IDpol et calculer la somme des autres colonnes
base2 = base2.groupby(['IDpol']).sum()

# Ajouter la colonne Year aux données groupées
base2.reset_index(inplace=True)
```

```python
base2.duplicated(subset=base2.columns.difference(['Year'])).sum() # somme les doublons purs sans "Year", Doublons ok!
```

2- Gestion des données manquantes (DM)

```python
print("Le nombre d'observation par variable")
base2.count()
```

```python
print("Le nombre de NaN par variable est de:")
base2.isna().sum()
```

```python
print("pas de données incohérentes")
```

TRAITEMENT ET ANALYSE DE LA BASE: "DB_SEV"

```python
base3 = db_sev.copy()
```

```python
base3.shape
```

```python
base3.head(2)
```

1- Gestion des doublons

```python
print("Le nombre des doublons dans la base est de :",len(base3["IDpol"]) - base3["IDpol"].nunique())
```

```python
print("La somme les doublons purs est de: ")
sum(base3.duplicated())
```

```python
print("Les doublons triés")
base3[base3.duplicated(subset = "IDpol", keep = False)].sort_values(by = "IDpol")
```

```python
base3.dtypes
```

```python
print("Conversion Payment en format numérique et remplacer les annomalies par des NaN")
base3['Payment'] = pd.to_numeric(base3['Payment'], errors='coerce'); base3.dtypes
```

```python
print("Agréger base3 par la clé primaire 'IDpol' pour eliminer les doublons")
base3 = base3.groupby('IDpol').agg({'OccurDate': 'sum', 'Payment': 'sum', 'IDclaim': 'sum', 'Guarantee': 'first'}).reset_index()
```

```python
base3.head()
```

```python
len(base3["IDpol"]) - base3["IDpol"].nunique()
```

2- Données manquantes

```python
base3.isna().sum()
```

3- Données incohérentes

```python
 print("Pas de données incohérentes dans la base 3")
```

4- Les jointures

4-1 Jointure entre base1 et base2 que l'on note par : base_t1

```python
base_t1 =  pd.merge(base1, base2, on = ['IDpol'], how = 'left'); base_t1.shape
```

```python
base_t1.head(3)
```

```python
print("Prémière verification des valeurs manquantes dans la base_t1 de jointure")
base_t1.isna().sum()
```

```python
base_t1.fillna(0, inplace=True)
```

```python
print("Deuxième verification des valeurs manquantes dans la base_t1 de jointure")
base_t1.isna().sum()
```

4-2 Jointure entre base_t1 et base3 que l'on note par : base_t2

```python
print("Prémière verification des valeurs manquantes dans la base_t2 de jointure")
base_t2 =  pd.merge(base_t1, base3, on = ['IDpol'],how = 'left'); base_t2.shape
```

```python
base_t2.isna().sum()
```

```python
base_t2.fillna(0, inplace=True)
```

```python
print("Deuxième verification des valeurs manquantes dans la base_t2 de jointure")
base_t2.isna().sum()
```

```python
print("La base finale qui est le résultat de la fusion entre base1, base2 et base3")
base_t2 .head(5)
```

```python
base_t2.info()
```

CHAPITRE II. ANALYSE ÉCONOMÉTRIQUE

1- Analyse des données (univariée):

```python
base_t2.head(2)
```

```python
print(base_t2.dtypes)
```

```python
#Vérification de si il existe toujours des valeurs manquantes
print(base_t2.isnull().sum())
```

```python
for i in range(len(base_t2.columns)):
    if base_t2.iloc[:,i].dtypes != 'float64':
        print(pd.value_counts(base_t2.iloc[:,i].astype(str)).sort_index())
```

```python
print("Les Statisques descriptives")
print(base_t2.describe())
```

```python
#Vérification de la distribution si normal(histogramme)
variables_num = base_t2.select_dtypes(include=['float64', 'int64'])
non_empty_variables = variables_num.columns[variables_num.count() > 0]
n_variables = len(non_empty_variables)

n_rows = int(n_variables ** 0.5) + 1
fig, axes = plt.subplots(nrows=n_rows, ncols=n_rows, figsize=(15, 10))
fig.subplots_adjust(hspace=0.3)

for i, column in enumerate(non_empty_variables):
    ax = axes[i // n_rows, i % n_rows]
    ax.hist(variables_num[column].dropna())
    ax.set_title(column)
if n_variables < n_rows ** 2:
    for i in range(n_variables, n_rows ** 2):
        fig.delaxes(axes[i // n_rows, i % n_rows])

fig.tight_layout()
```

```python
  #Ici génération d'echantillon aléatoire de 100 valeurs
np.random.seed(42)
sample = np.random.normal(loc=0, scale=1, size=100)

statistic, p_value = stats.shapiro(sample)

print("Statistique de test de normalité :", statistic)
print("Valeur p :", p_value)
```

2- Analyse des données (multivariée)

```python
print("Tableau croisé de variable entre une variable catégorielle et numérique")
print()
cross_tab = pd.crosstab(base_t2['DrivGender'], base_t2['BonusMalus'])
print(cross_tab)
```

```python
count_sexe=base_t2['DrivGender'].value_counts()
print(count_sexe)
```

```python
Bonus_Malus_filtre = base_t2[base_t2['BonusMalus'] < 100]
count_by_sexe = Bonus_Malus_filtre['DrivGender'].value_counts()
print(count_by_sexe)
```

```python
Bonus_Malus_filtre = base_t2[base_t2['BonusMalus'] > 100]
count_by_sexe_2 = Bonus_Malus_filtre['DrivGender'].value_counts()
print(count_by_sexe_2)
```

```python
fig, axs = plt.subplots(1, 2, figsize=(12, 7))

# Ajout d'un jeu de couleurs pour améliorer l'apparence
colors = ["#19c213", "#ff7f0e"]

axs[0].pie(count_by_sexe, labels=count_by_sexe.index, autopct='%1.1f%%', colors=colors, textprops={'fontsize': 12})
axs[0].set_title('Répartition de personne avec un Bonus (< 100) par sexe', fontsize=12)

axs[1].pie(count_by_sexe_2, labels=count_by_sexe_2.index, autopct='%1.1f%%', colors=colors, textprops={'fontsize': 12})
axs[1].set_title('Répartition de personne avec un Malus (> 100) par sexe', fontsize=12)

plt.savefig("Répartition de personne avec un Bonus inf à 100 par sexe.png")
plt.savefig("CCC.png")
plt.tight_layout()
plt.show()
```

```python
Bonus_Malus_filtre = base_t2[base_t2['BonusMalus'] < 100]
count_by_region = Bonus_Malus_filtre['Region'].value_counts()
print(count_by_region)

Bonus_Malus_filtre = base_t2[base_t2['BonusMalus'] > 100]
count_by_region_2 = Bonus_Malus_filtre['Region'].value_counts()
print(count_by_region_2)
```

```python
fig, axs = plt.subplots(1, 2, figsize=(18, 7))

# Utilisation de barplot au lieu de pie
bars1 = axs[0].barh(count_by_region.index, count_by_region, color="#1f77b4")
axs[0].set_title('Répartition de personne avec un Bonus (< 100) par région', fontsize=12)
axs[0].invert_yaxis()  # Inverser l'axe y pour que les barres soient affichées du haut vers le bas

# Ajouter des pourcentages sur les barres
for bar in bars1:
    width = bar.get_width()
    axs[0].text(width,
                bar.get_y() + bar.get_height()/2,
                f'{width/len(base1)*100:.1f}%',
                ha='left',
                va='center')

bars2 = axs[1].barh(count_by_region_2.index, count_by_region_2, color="#ff7f0e")
axs[1].set_title('Répartition de personne avec un Malus (> 100) par région', fontsize=12)
axs[1].invert_yaxis()

# Ajouter des pourcentages sur les barres
for bar in bars2:
    width = bar.get_width()
    axs[1].text(width,
                bar.get_y() + bar.get_height()/2,
                f'{width/len(base1)*100:.1f}%',
                ha='left',
                va='center')
plt.savefig("Repartition_de_personne_avec_Malus_plus_de_100_par_region.png")
plt.tight_layout()
plt.show()
```

Analyse de la distribution

```python
measurements = np.random.normal(loc=20, scale=5, size=100)
fig, ax = plt.subplots(figsize=(10, 7))
stats.probplot(measurements, dist="norm", plot=ax)
ax.set_title("Graphique quantile-quantile (Q-Q plot)")
plt.show()
fig.savefig("qq_plot.png")
```

Analyse de corrélation multiple (vérification de la relation linéaire entre plusieurs vars), la base est estimé paramétrique

```python
for col in base_t2.select_dtypes(include=['object']).columns:
    print(f'\n{col}:')
    print(base_t2[col].value_counts())

df = pd.get_dummies(base_t2, drop_first=True)
```

```python
baseTransf = pd.get_dummies(base_t2, drop_first=True)
subset = baseTransf.iloc[:, :10]  # change to a number of columns that fits in your memory
correlations = subset.corr()
```

```python
correlations
```

```python
plt.figure(figsize=(10,6))
sns.heatmap(correlations)
plt.savefig('Correlation.png')
```

```python
#Décocmposition de la matrice de corrélation (calcul des valeurs propres)
eigenvalues, eigenvectors = np.linalg.eig(corr_matrix)

for i in range(len(eigenvalues)):
    print(f"Valeur propre {i+1}: {eigenvalues[i]}")
    print(f"Vecteur propre {i+1}: {eigenvectors[:, i]}\n")
```

```python
n_components = len(eigenvalues)
component_indices = range(1, n_components + 1)
plt.figure(figsize=(10, 6))  # Agrandissement de la figure
plt.plot(component_indices, eigenvalues, marker='o', linestyle='-')
plt.xlabel('Composantes principales')
plt.ylabel('Valeurs propres')
plt.title('Graphique des valeurs propres')
plt.xticks(component_indices)
plt.grid(True)
plt.savefig("Graphique des valeurs propres.png")
plt.show()
```

```python
#Calcul de la variance total suie aux calcul des valeurs propres ;
total_variance = np.sum(eigenvalues)
#Calcul variance expliqué et var expliqué cumulée
explained_variance = eigenvalues / total_variance
cumulative_variance = np.cumsum(explained_variance)

print("Variance totale :", total_variance)
print("Variance expliquée cumulée :", cumulative_variance)
```

```python
plt.figure(figsize=(10, 6))  # Agrandissement de la figure
plt.plot(range(1, len(eigenvalues) + 1), cumulative_variance, marker='o', linestyle='-')
plt.xlabel('Nombre de composantes principales')
plt.ylabel('Variance expliquée cumulée (%)')
plt.title('Graphique de variance expliquée cumulée')
plt.grid(True)
plt.savefig('variance_cumulee.png')  # Enregistrement de l'image
plt.show()
```

```python
selected_components_index = np.where(cumulative_variance >= 0.8)[0][0]
selected_components = range(selected_components_index + 1)
print(selected_components)
```

```python
projection_matrix = np.dot(corr_matrix, np.eye(len(corr_matrix)))[:, selected_components]
print(projection_matrix)
```

CHAPITRE III. LE MACHINE LEARNING

```python
base = base_t2.copy()
```

```python
# Supprimer les colonnes non pertinentes pour la régression linéaire
X_reg = base.drop(['Payment', 'IDpol', 'Year_x', 'LicenceNb', 'Marketing', 'Year_y', 'IDclaim'], axis=1)
```

1. Régression : Prédire le montant du paiement

```python
# Régression linéaire
X_reg = X_reg.select_dtypes(include=[np.number])
y_reg = base['Payment']
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_reg, y_reg, test_size=0.3, random_state=42)
reg = LinearRegression().fit(X_train_reg, y_train_reg)
y_pred_reg = reg.predict(X_test_reg)
rmse = mean_squared_error(y_test_reg, y_pred_reg, squared=False)
print("L'Erreur Quadratique Moyenne du modèle est de :", rmse)
```

2 Modèle de régression de forêt aléatoire

```python
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
# Ajuster le modèle de forêt aléatoire aux données d'entraînement
rf_reg.fit(X_train_reg, y_train_reg)
# Utilisation le modèle ajusté pour prédire les paiements sur l'ensemble de test
y_pred_rf_reg = rf_reg.predict(X_test_reg)
# Calculer le RMSE pour les prédictions de la forêt aléatoire
rmse_rf = mean_squared_error(y_test_reg, y_pred_rf_reg, squared=False)
print("L'Erreur Quadratique Moyenne (Forêt Aléatoire) du modèle est :", rmse_rf)
```

```python
# Les modèles
models = ['Linear Regression', 'Random Forest']
# Les valeurs de RMSE
rmse_values = [2010.2807283220316, 2378.852240326708]
# Création du graphique à barres
plt.figure(figsize=(8,6))
plt.bar(models, rmse_values, color=['blue', 'green'])

# Ajout de titres et d'étiquettes
plt.title('Comparaison des erreurs RMSE pour différents modèles de régression')
plt.xlabel('Modèles')
plt.ylabel('RMSE')
plt.savefig('Comparaison des erreurs RMSE pour différents modèles de régression.png')
plt.show()
```

```python
# Création des ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_reg , y_reg, test_size=0.2, random_state=42)

# Formation du modèle
grid_search.fit(X_train, y_train)

# Obtenir le meilleur modèle
best_model = grid_search.best_estimator_

# Prédire sur l'ensemble de test
y_pred = best_model.predict(X_test)

# Calculer les métriques
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# les métriques
print("l'erreur quadratique moyenne du modèle MSE est de :", mse)
print("L'erreur absolue moyenne du modèle MAE est de:", mae)
print("Le coefficient de détermination R^2 est de: ", r2)
```

```python
from sklearn.model_selection import cross_val_score
ridge = Ridge(alpha=grid_search.best_params_['alpha'])
# Effectuer une validation croisée k-fold
scores = cross_val_score(ridge, X_reg, base['Payment'], cv=5)
print('Le Score de validation croisée est de', scores)
print()
print('Le Score moyen de validation croisée est de :', scores.mean())
```

ANALYSE EXPLORATOIRE APPROFONDIE:

Voyons une analyse exploratoire des données pour de découvrir des tendances intéressantes, comme les régions où les dommages sont les plus importantspar exemple

```python
# Agrégation les données par région et calcul de la somme des dommages pour chaque région
damage_by_region = base.groupby('Region')['Damage'].sum()

# Trie de régions par dommages totaux en ordre décroissant
damage_by_region = damage_by_region.sort_values(ascending=False)

print(damage_by_region)
```

```python
plt.figure(figsize=(15,8))
bars = plt.barh(damage_by_region.index, damage_by_region.values, color='skyblue', edgecolor='black')

plt.xlabel('Total des dommages', fontsize=14)
plt.ylabel('Région', fontsize=14)
plt.title('Total des dommages par région', fontsize=16)

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Calculer le total pour obtenir les pourcentages
total = damage_by_region.values.sum()

# Ajouter les pourcentages sur les barres
for bar in bars:
    width = bar.get_width()
    plt.text(width, bar.get_y() + bar.get_height()/2,
             f' {width/total:.1%}',  # format as percentage with one decimal place
             va='center', ha='left', fontsize=12)

plt.tight_layout()
plt.savefig('Total des dommages par région.png')
plt.show()
```

Calculer la somme des dommages par région et le nombre de polices d'assurance par région :

```python
# Somme des dommages par région
damage_by_region = base.groupby('Region')['Damage'].sum()

# Nombre de polices par région
policies_by_region = base['Region'].value_counts()
print(policies_by_region)
```

On peut normaliser les dommages par le nombre de polices pour obtenir les dommages moyens par police dans chaque région :

```python
# Dommages moyens par police par région
average_damage_by_region = damage_by_region / policies_by_region
average_damage_by_region = average_damage_by_region.sort_values(ascending=False)
print(average_damage_by_region)
```

```python
# Définir les dimensions de la figure
fig, ax = plt.subplots(figsize=(12, 8))

# Créer un graphique à barres avec les régions sur l'axe des x et la moyenne des dommages sur l'axe des y
bars = ax.bar(average_damage_by_region.index, average_damage_by_region.values, color='indigo', edgecolor='black')

# Définir le titre et les étiquettes des axes
ax.set_title('Moyenne des dommages par police par région', fontsize=16)
ax.set_xlabel('Région', fontsize=14)
ax.set_ylabel('Moyenne des dommages par police', fontsize=14)

# Rotation des étiquettes de l'axe des x pour une meilleure lisibilité
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)

# Ajouter des grilles pour une meilleure lisibilité
ax.grid(True)

# Enregistrer la figure
plt.tight_layout()
plt.savefig('Moyenne des dommages par police par région.png')

# Afficher le graphique
plt.show()
```

Voyons en comment les dommages varient en fonction du type de véhicule. Pour ce faire, on peut regrouper les données par VehClass et calculer la somme des dommages pour chaque type de véhicule :

```python
# Somme des dommages par type de véhicule
damage_by_vehclass = base.groupby('VehClass')['Damage'].sum()
print(damage_by_vehclass)
```

```python
# Définir les dimensions de la figure
fig, ax = plt.subplots(figsize=(12, 8))

# Créer un graphique à barres avec les régions sur l'axe des x et la moyenne des dommages sur l'axe des y
bars = ax.bar(damage_by_vehclass.index, damage_by_vehclass.values, color='dodgerblue', edgecolor='black')

# Définissez le titre et les étiquettes des axes
ax.set_title('Somme des dommages par type de véhicule', fontsize=16)
ax.set_xlabel('Type de véhicule', fontsize=16)
ax.set_ylabel('Somme des dommages',fontsize=14)

# Rotation des étiquettes de l'axe des x pour une meilleure lisibilité
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)

# Ajouter des grilles pour une meilleure lisibilité
ax.grid(True)

# Enregistrer la figure
plt.tight_layout()
plt.savefig('Somme des dommages par type de véhicule.png')

# Afficher le graphique
plt.show()
```

Et on peut faire de même pour le nombre de polices par type de véhicule, puis calculer les dommages moyens par police pour chaque type de véhicule :

```python
# Nombre de polices par type de véhicule
policies_by_vehclass = base['VehClass'].value_counts()

# Dommages moyens par police par type de véhicule
average_damage_by_vehclass = damage_by_vehclass / policies_by_vehclass
average_damage_by_vehclass = average_damage_by_vehclass.sort_values(ascending=False)
print(average_damage_by_vehclass)
```

```python
# Définir les dimensions de la figure
fig, ax = plt.subplots(figsize=(12, 8))

# Créer un graphique à barres avec les régions sur l'axe des x et la moyenne des dommages sur l'axe des y
bars = ax.bar(average_damage_by_vehclass.index, average_damage_by_vehclass.values, color='purple', edgecolor='black')

# Définir le titre et les étiquettes des axes
ax.set_title('Dommages moyens par police par type de véhicule', fontsize=16)
ax.set_xlabel('Type de véhicule', fontsize=16)
ax.set_ylabel('Dommages moyens par police', fontsize=14)

# Rotation des étiquettes de l'axe des x pour une meilleure lisibilité
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)

# Ajouter des grilles pour une meilleure lisibilité
ax.grid(True)
# Enregistrer la figure
plt.tight_layout()
plt.savefig('Dommages moyens par police par type de véhicule.png')
# Afficher le graphique
plt.show()
```

Enfin, analysons l'impact de l'âge et du sexe du conducteur sur le total des dommages. Pour cela, nous pouvons calculer la somme des dommages par âge et par sexe du conducteur.

```python
# Agréger les données par âge du conducteur et calculez la somme des dommages pour chaque âge
damage_by_age = base.groupby('DrivAge')['Damage'].sum()

# Trier les âges par dommages totaux en ordre décroissant
damage_by_age = damage_by_age.sort_values(ascending=False)

print(damage_by_age)

# Agréger les données par sexe du conducteur et calculez la somme des dommages pour chaque sexe
damage_by_gender = base.groupby('DrivGender')['Damage'].sum()

# Trier les sexes par dommages totaux en ordre décroissant
damage_by_gender = damage_by_gender.sort_values(ascending=False)

print(damage_by_gender)
```

```python
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 14))
# graphique linéaire avec l'âge du conducteur sur l'axe des x et la somme des dommages sur l'axe des y
ax1.plot(damage_by_age.index, damage_by_age.values, color='blue', marker='o')
ax1.set_title('Somme des dommages en fonction de l\'âge du conducteur', fontsize=13)
ax1.set_xlabel('Âge du conducteur', fontsize=14)
ax1.set_ylabel('Somme des dommages', fontsize=14)
ax1.grid(True)

# Annoter la valeur maximale et minimale sur le graphique linéaire
min_val = damage_by_age.values.min()
max_val = damage_by_age.values.max()
min_idx = damage_by_age.index[damage_by_age.values.argmin()]
max_idx = damage_by_age.index[damage_by_age.values.argmax()]
ax1.annotate(f'Min: {min_val}', xy=(min_idx, min_val), xytext=(min_idx, min_val-5), arrowprops=dict(facecolor='red', shrink=0.05))
ax1.annotate(f'Max: {max_val}', xy=(max_idx, max_val), xytext=(max_idx, max_val+5), arrowprops=dict(facecolor='green', shrink=0.05))

# graphique à barres avec le sexe du conducteur sur l'axe des x et la somme des dommages sur l'axe des y
bars = ax2.bar(damage_by_gender.index, damage_by_gender.values, color=['orchid', 'skyblue'])
ax2.set_title('Somme des dommages en fonction du sexe du conducteur', fontsize=13)
ax2.set_xlabel('Sexe du conducteur', fontsize=14)
ax2.set_ylabel('Somme des dommages', fontsize=14)
ax2.grid(axis='y')

# Ajouter des annotations pour chaque barre dans le graphique à barres
for bar in bars:
    yval = bar.get_height()
    ax2.text(bar.get_x() + bar.get_width()/2, yval, int(yval), ha='center', va='bottom', fontsize=12)

# Réduire le nombre de graduations sur l'axe des x pour le graphique de l'âge pour une meilleure lisibilité
ax1.xaxis.set_major_locator(ticker.MultipleLocator(5))

# Enregistrer et affichez les graphiques
plt.tight_layout()
plt.savefig("Somme des dommages en fonction de l'âge et du sexe du conducteur.png")
plt.show()
```