Classification de crimes à San Francisco

June 7, 2021

1 Classification de crimes à San Francisco - Ambroise Odonnat, Waël Doulazmi, Paul-Eloi Mangion, Alexis Gadonneix et Rémi Simon

Notre sujet est un problème de classification à partir d'un très grand dataset de train constitué à la fois de données numériques et de données catégorielles (et un peu de données textuelles que nous laisseront de côté). Les enjeux sont de sélectionner et de traiter les données pertinentes puis d'essayer un certain nombre d'algorithmes d'apprentissage pour trouver le plus adapté au problème.

[ ]:
```
import sys
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

sns.set_theme()
plt.figure(figsize=(15,10))
```

[ ]:
```
<Figure size 1080x720 with 0 Axes>
```

```
<Figure size 1080x720 with 0 Axes>
```

[ ]:
```
#Read training data

train_data0 = pd.read_csv('data/train.csv')

train_data=train_data0.copy()
print(train_data.shape)
print('data loaded')

data_sample=train_data.sample(n=50000)

train_data.head()
```

```
(878049, 9)
data loaded
```

[ ]:
```
                  Dates        Category                     Descript  \
0  2015-05-13 23:53:00        WARRANTS                WARRANT ARREST
1  2015-05-13 23:53:00  OTHER OFFENSES      TRAFFIC VIOLATION ARREST
2  2015-05-13 23:33:00  OTHER OFFENSES      TRAFFIC VIOLATION ARREST
3  2015-05-13 23:30:00   LARCENY/THEFT  GRAND THEFT FROM LOCKED AUTO
4  2015-05-13 23:30:00   LARCENY/THEFT  GRAND THEFT FROM LOCKED AUTO

   DayOfWeek PdDistrict      Resolution                    Address  \
0  Wednesday   NORTHERN  ARREST, BOOKED         OAK ST / LAGUNA ST
1  Wednesday   NORTHERN  ARREST, BOOKED         OAK ST / LAGUNA ST
2  Wednesday   NORTHERN  ARREST, BOOKED  VANNESS AV / GREENWICH ST
3  Wednesday   NORTHERN            NONE   1500 Block of LOMBARD ST
4  Wednesday       PARK            NONE  100 Block of BRODERICK ST

            X          Y
0 -122.425892  37.774599
1 -122.425892  37.774599
2 -122.424363  37.800414
3 -122.426995  37.800873
4 -122.438738  37.771541
```

1.1 Visualisation et pré-traitement des données

Dans un premier temps, nous allons tenter de sélectionner les données les plus pertinentes et une bonne méthode pour les modifier afin qu'elles soient le plus exploitable possible par nos modèles.

La sélection des données pertinentes va passer en partie par la visualisation de ces données. On ne visualisera que sur un échantillon du dataset pour que ce soit plus clair.

On commence par quelques modifications naturelles : on récupère l'ensemble des catégories et on transforme les jours de la semaine en valeurs numériques. On omet les données textuelles qui ne sont présentes que dans le set de train (Descript et Resolution) ainsi que la chaîne de caractère 'Adress' qui nous semblait difficilement exploitable et un peu redondante avec les coordonnées géographiques.

[ ]:
```
#Get categories (output to find)

cat = train_data.Category.unique()
print(cat)
list_cat=np.copy(cat)
np.save('categories.npy', list_cat)
train_data.drop(['Descript','Resolution','Address'],inplace =True, axis=1) ␣
↪#data useless, unsignificant, estimated useless
train_data['DayOfWeek'].
↪replace(to_replace=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
↪for i in range(0,7)],inplace=True)
```

```
train_data.head()
```

```
['WARRANTS' 'OTHER OFFENSES' 'LARCENY/THEFT' 'VEHICLE THEFT' 'VANDALISM'
 'NON-CRIMINAL' 'ROBBERY' 'ASSAULT' 'WEAPON LAWS' 'BURGLARY'
 'SUSPICIOUS OCC' 'DRUNKENNESS' 'FORGERY/COUNTERFEITING' 'DRUG/NARCOTIC'
 'STOLEN PROPERTY' 'SECONDARY CODES' 'TRESPASS' 'MISSING PERSON' 'FRAUD'
 'KIDNAPPING' 'RUNAWAY' 'DRIVING UNDER THE INFLUENCE'
 'SEX OFFENSES FORCIBLE' 'PROSTITUTION' 'DISORDERLY CONDUCT' 'ARSON'
 'FAMILY OFFENSES' 'LIQUOR LAWS' 'BRIBERY' 'EMBEZZLEMENT' 'SUICIDE'
 'LOITERING' 'SEX OFFENSES NON FORCIBLE' 'EXTORTION' 'GAMBLING'
 'BAD CHECKS' 'TREA' 'RECOVERED VEHICLE' 'PORNOGRAPHY/OBSCENE MAT']
```

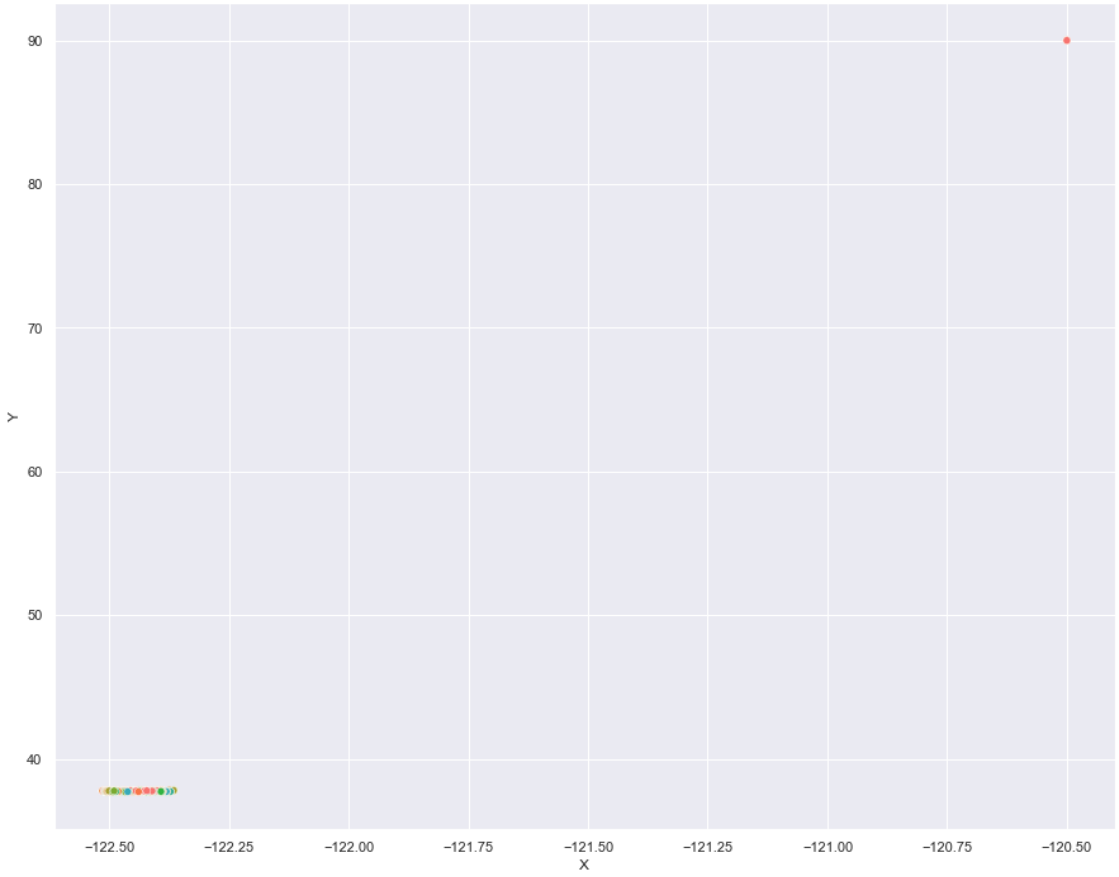
[ ]:

	Dates	Category	DayOfWeek	PdDistrict	X	Y
0	2015-05-13 23:53:00	WARRANTS	2	NORTHERN	-122.425892	37.774599
1	2015-05-13 23:53:00	OTHER OFFENSES	2	NORTHERN	-122.425892	37.774599
2	2015-05-13 23:33:00	OTHER OFFENSES	2	NORTHERN	-122.424363	37.800414
3	2015-05-13 23:30:00	LARCENY/THEFT	2	NORTHERN	-122.426995	37.800873
4	2015-05-13 23:30:00	LARCENY/THEFT	2	PARK	-122.438738	37.771541

On va ensuite tenter de visualiser les données géographiques.

[ ]:
```
plt.figure(figsize=(15,12))
sns.scatterplot(
    data=data_sample,
    x="X", y="Y",
    hue="Category",
    legend=False)
```

[ ]:
```
<AxesSubplot:xlabel='X', ylabel='Y'>
```

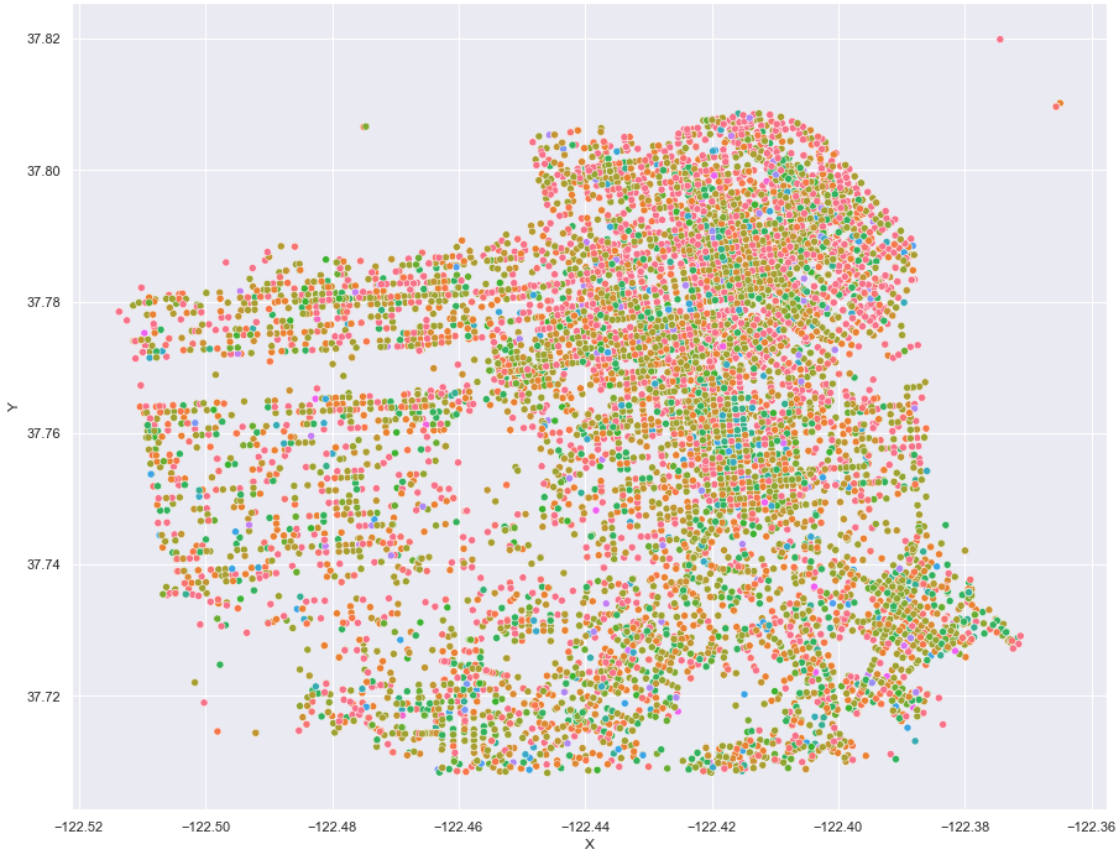


Ce premier graphe nous indique clairement la présence d'outliers géographiques situés très loin (plusieurs milliers de km !) de San Francisco. Nous avons donc décider de supprimer ces outliers (qui représentent moins de 0,1% du dataset). On retente ensuite de visualiser.

[ ]:
```
train_data.drop(train_data[(train_data.Y>60)|(train_data.X>-122)].index,␣
↪inplace=True)  #outliers
data_sample=train_data.sample(n=20000)
```

[ ]:
```
plt.figure(figsize=(15,12))
sns.scatterplot(
    data=data_sample,
    x="X", y="Y",
    hue="Category",
    legend=False)
```

[ ]:
```
<AxesSubplot:xlabel='X', ylabel='Y'>
```



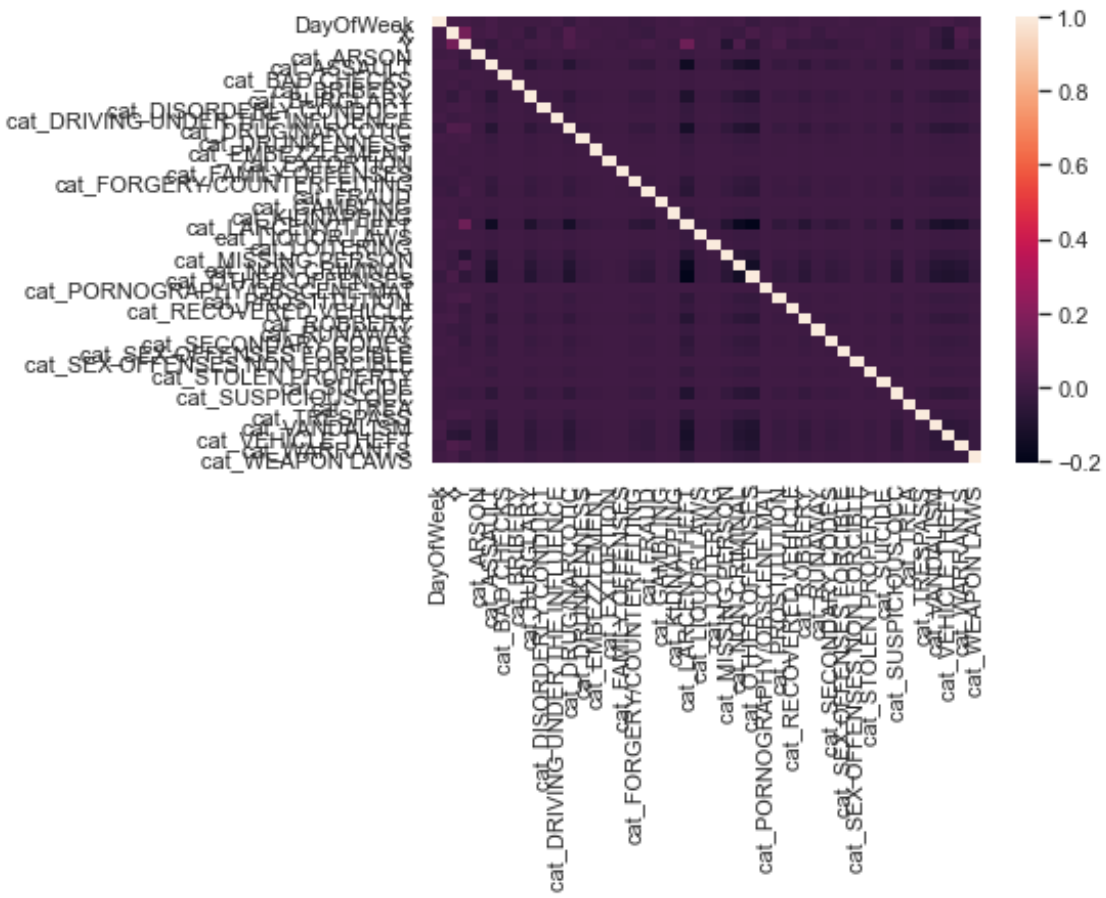
C'est beaucoup plus intéressant ! On ne peut pas dire à l'oeil nu si le type de crime (couleurs des points) dépend du lieu mais en tout cas on peut voir que la densité de crime varie beaucoup dans la ville. On peut reconnaître certaines grandes avenues très fréquentées, ainsi que le parc (rectangle sans crime en haut à gauche).

On peut essayer de visualiser la matrice de covariance sur le set de données pour regarder s' il y a des liens forts entre certaines données ou si certaines colonnes sont très pertinentes pour la classification. Cependant, les résultats attendus sont catégoriels donc on doit passer par un encodage one hot (seulement pour dessiner la matrice).

[ ]:
```
df=train_data.sample(n=100000)

onehot = pd.concat([df,pd.get_dummies(df['Category'], prefix='cat')],axis=1).
 ↪drop(['Category'],axis=1)
```

[ ]:
```
corr = onehot.corr()

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
```

[ ]: <AxesSubplot:>



On y voit peu de choses malheureusement à cause du très grand nombre de colonnes ajoutées par le one hot. Il n'est donc pas évident de décider quelles données sont les plus pertinentes. On essaiera néanmoins par la suite d'étudier d'autres corrélations.

Pour le moment, continuons de formatter nos données. Le temps est pour l'instant sous forme d'une chaîne de caractères. On extrait un maximum d'informations de cette chaine (on connait deja le jour de la semaine) : Temps total de la journée en secondes, année, mois, jour du mois et heure de la journée. On s'affranchit de l'extraction des minutes et des secondes qui ne semblent pas très determinantes.

On remplace également les catégories par des données chiffrées (cela induit un ordre non désiré entre les catégories mais le nombre élevé de catégories rend les onehot inexploitables pour nos algorithmes). On peut donc retenter l'expérience des corrélations! (en retirant les données que l'on sait fortement corrélées (Time et Hour ou X et Y).

[ ]:
```
#Format time data

train_data.Dates = pd.to_datetime(train_data.Dates)
```

```
train_data['Time'] = train_data.Dates.dt.hour*3600 + train_data.Dates.dt.
 ↪minute*60 + train_data.Dates.dt.second
train_data['Hour'] = train_data.Dates.dt.hour
train_data['Day'] = train_data.Dates.dt.day
train_data['Month']=train_data.Dates.dt.month
train_data['Year']=train_data.Dates.dt.year
for key in ['Time','Hour','Day','Month','Year']:
    train_data[key]=pd.to_numeric(train_data[key])


train_data['Category'].replace(to_replace=cat,value=[i for i in␣
 ↪range(len(cat))],inplace=True)
train_data.head()
```

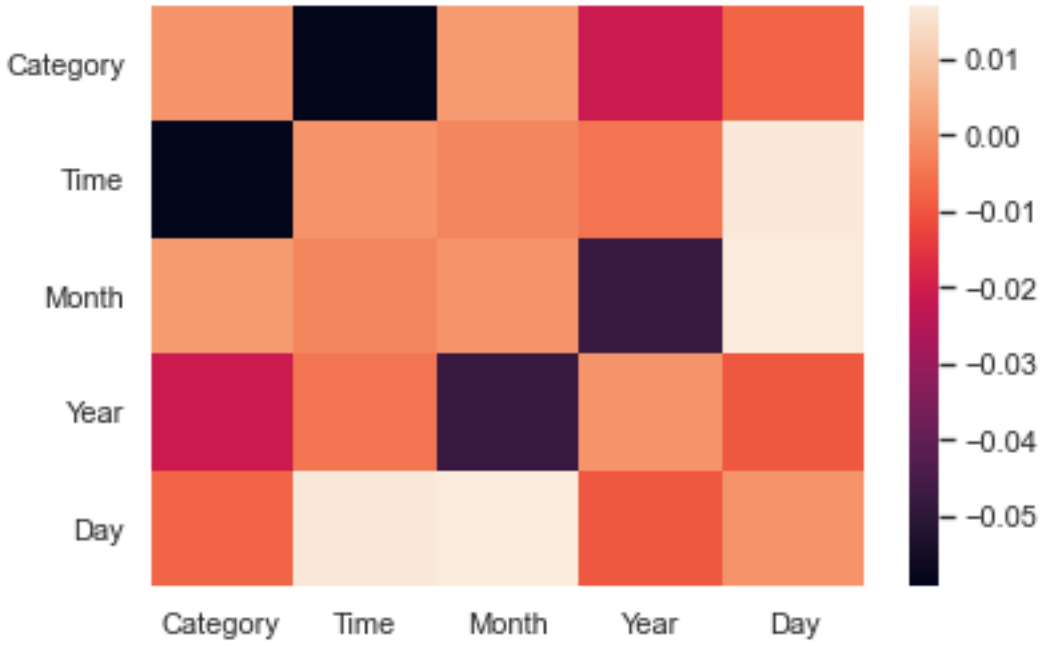
[ ]:

	Dates	Category	DayOfWeek	PdDistrict	X	Y	\
0	2015-05-13 23:53:00	0	2	NORTHERN	-122.425892	37.774599	
1	2015-05-13 23:53:00	1	2	NORTHERN	-122.425892	37.774599	
2	2015-05-13 23:33:00	1	2	NORTHERN	-122.424363	37.800414	
3	2015-05-13 23:30:00	2	2	NORTHERN	-122.426995	37.800873	
4	2015-05-13 23:30:00	2	2	PARK	-122.438738	37.771541	

	Time	Hour	Day	Month	Year
0	85980	23	13	5	2015
1	85980	23	13	5	2015
2	84780	23	13	5	2015
3	84600	23	13	5	2015
4	84600	23	13	5	2015

[ ]:
```
corr = train_data[['Category','Time','Month','Year','Day']].corr()
np.fill_diagonal(corr.values, 0)

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
```

[ ]: <AxesSubplot:>



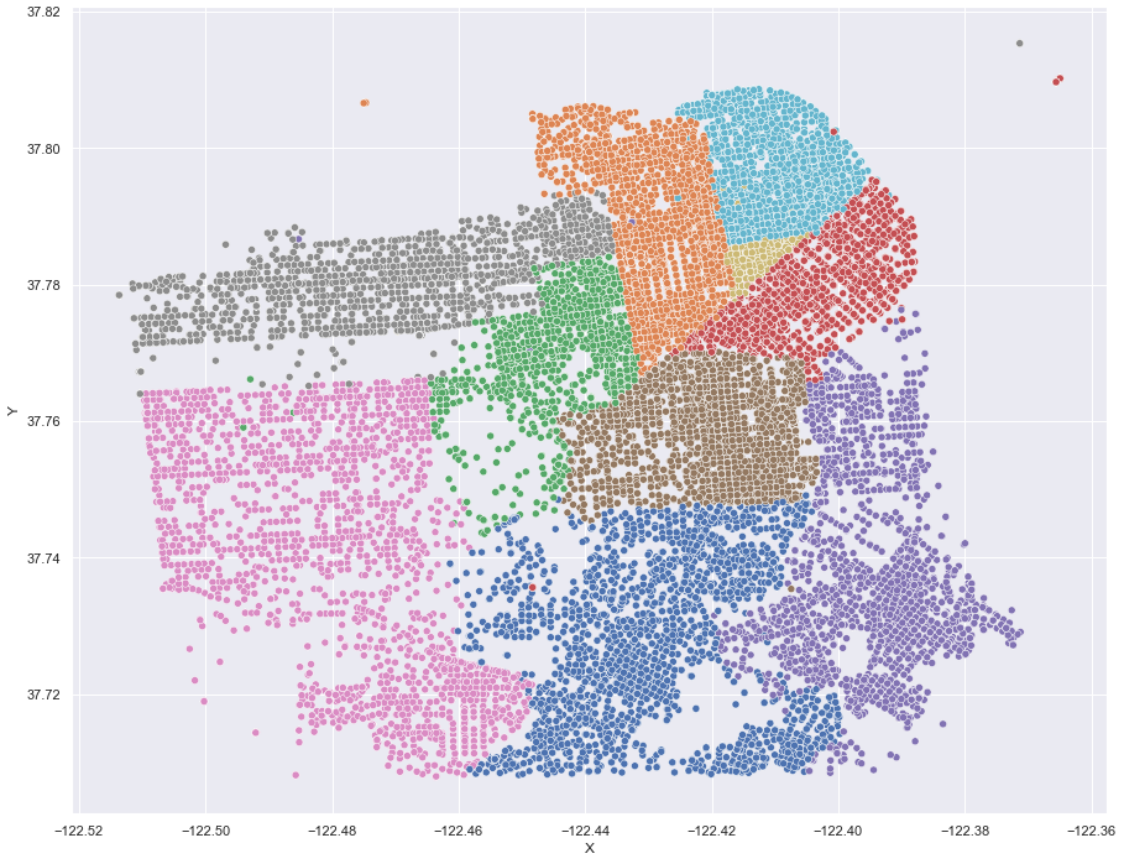
(On a mis les auto corrélations à 0 pour qu'elles n'écrasent pas les autres)

Le orange/beige correspond à 0, Blanc et noir correspondent aux corrélations (positives ou négatives). On ne pousse pas l'interprétation des relations de covariance entre ces données (on rappelle que les données catégorielles sont devenues numériques) mais on peut voir que les corrélations varient et que la plupart des paramètres semblent influer sur la catégorie de crime.

On a pour le moment laissé une catégorie de côté : le district. C'est un critère à priori redondant avec les coordonnées géographiques mais voyons ce qu'il en est sur notre "carte".

[ ]:
```
plt.figure(figsize=(15,12))

sample=train_data.sample(n=40000)
sns.scatterplot(
    data=sample,
    x="X", y="Y",
    hue="PdDistrict",
    legend=False)
```

[ ]:
```
<AxesSubplot:xlabel='X', ylabel='Y'>
```



Effectivement, les districts semblent parfaitement définis géographiquement, chacun a ses rues et ne s'aventure que très rarement dans un autre district (pas de spécialisation à priori). On conserve cette donnée qui encode un autre niveau de précision de la localisation, des zones géographiques. De plus, ces données catégorielles induisent logiquement une topologie (les zone bleu clair est plus proche de la zone orange que de la zone rose). On va donc remplacer la colonne PdDistrict par deux colonnes qui contiennent les coordonnées géographiques du barycentre de chaque district.

[ ]:
```
def District_barycenters(data):
    Districts = data['PdDistrict'].unique()
    Bary_coord = {}
    for i in range(len(Districts)):
        test = data[['PdDistrict','X','Y']]
        district = test.loc[test['PdDistrict']==Districts[i],:]
        abs = district['X']
        ord = district['Y']
        mean_abs = abs.mean()
        mean_ord = ord.mean()
        Bary_coord[Districts[i]] = (mean_abs,mean_ord)
    data['X_Bar_District'] = data['PdDistrict']
    data['Y_Bar_District'] = data['PdDistrict']
```

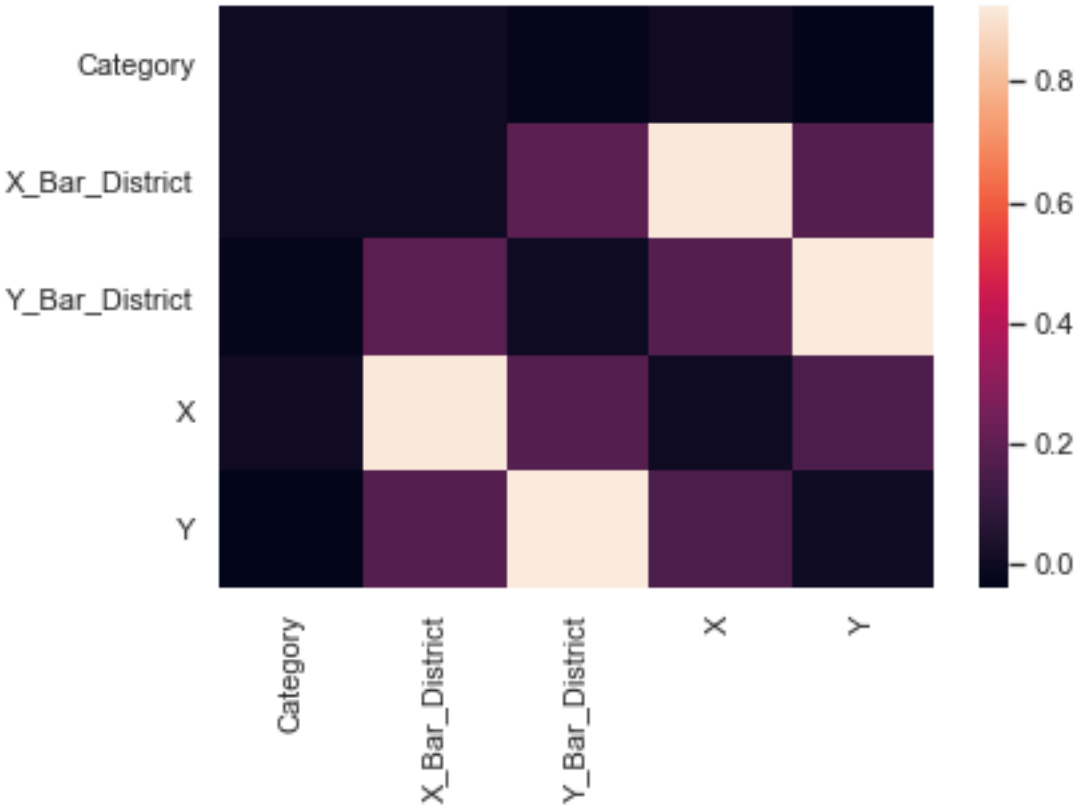
```
    data.drop('PdDistrict', inplace = True, axis=1)
    data['X_Bar_District'].
 ↪replace(to_replace=Districts,value=[Bary_coord[Districts[i]][0] for i in␣
 ↪range(len(Districts))],                    inplace=True)
    data['Y_Bar_District'].
 ↪replace(to_replace=Districts,value=[Bary_coord[Districts[i]][1] for i in␣
 ↪range(len(Districts))],                    inplace=True)
District_barycenters(train_data)
train_data.head()
```

[ ]:

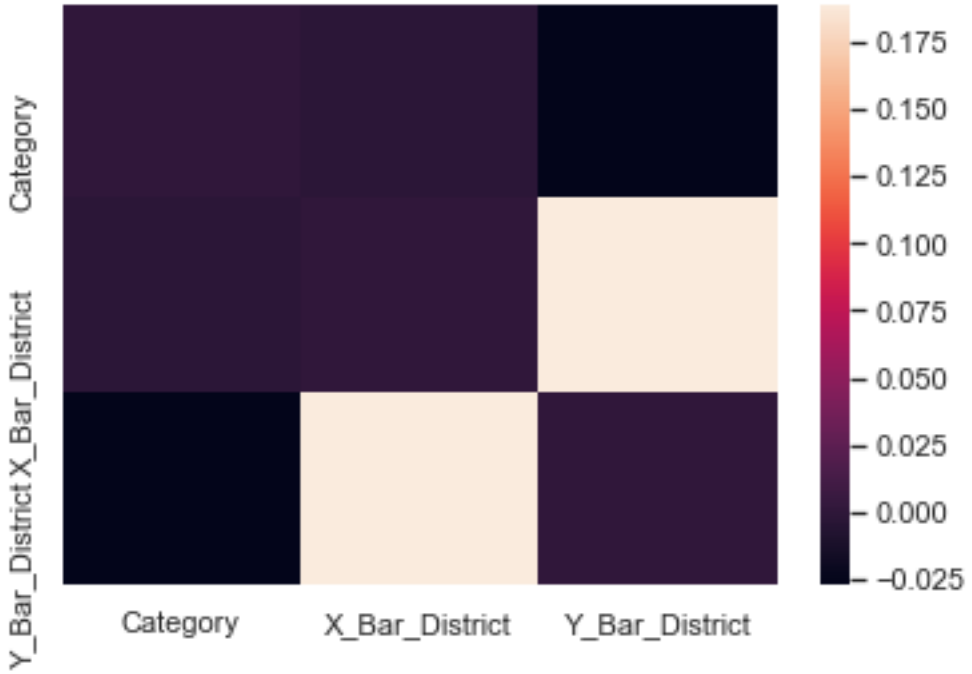
	Dates	Category	DayOfWeek	X	Y	Time	\
0	2015-05-13 23:53:00	0	2	-122.425892	37.774599	85980	
1	2015-05-13 23:53:00	1	2	-122.425892	37.774599	85980	
2	2015-05-13 23:33:00	1	2	-122.424363	37.800414	84780	
3	2015-05-13 23:30:00	2	2	-122.426995	37.800873	84600	
4	2015-05-13 23:30:00	2	2	-122.438738	37.771541	84600	

	Hour	Day	Month	Year	X_Bar_District	Y_Bar_District
0	23	13	5	2015	-122.426647	37.786379
1	23	13	5	2015	-122.426647	37.786379
2	23	13	5	2015	-122.426647	37.786379
3	23	13	5	2015	-122.426647	37.786379
4	23	13	5	2015	-122.445448	37.770299

[ ]:
```
corr = train_data[['Category','X_Bar_District','Y_Bar_District','X','Y']].corr()
np.fill_diagonal(corr.values, 0)

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)

plt.show()

corr = train_data[['Category','X_Bar_District','Y_Bar_District']].corr()
np.fill_diagonal(corr.values, 0)

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
```



[ ]: <AxesSubplot:>

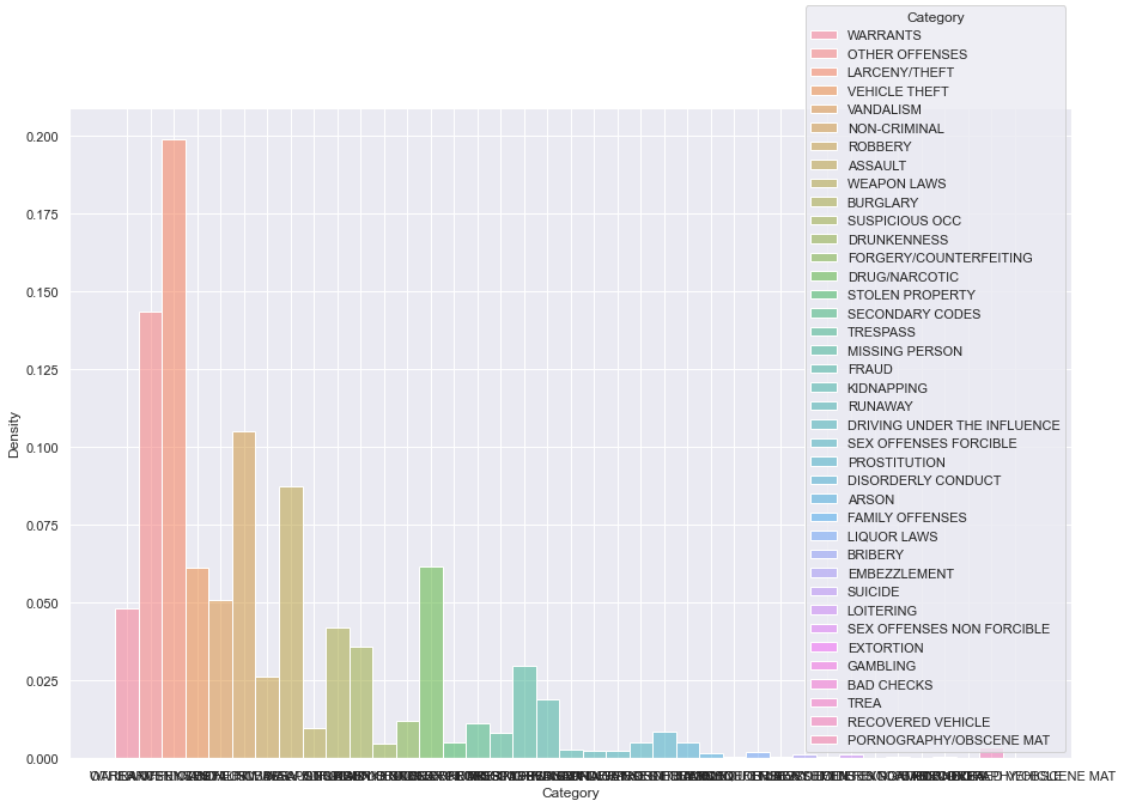


On voit que ces nouvelles données sont naturellemeny très corrélées avec les données géographiques mais elles ne sont pas complétement indépendantes de la catégorie et peuvent apporter un autre niveau d'information utile pour certains algorithmes.

Comme on peut le voir sur le graphe suivant, une des difficultés de ce problème de classification est le très grand déséquilibre dans les données. En effet, moins de 10 des 40 catégories représentent près de 90% du dataset tandis que certaines sont extrèmement peu représentées. C'est d'ailleurs pour cela que la comparaison des résultats se fait à l'aide de la log loss et non de la 0-1 loss (l'argmax correspond presque toujours à une des grosses catégories).

[ ]:
```
plt.figure(figsize=(15,10))
sns.histplot(
    data=train_data0,
    x="Category",
    stat='density',
    discrete=True,
    hue="Category",
)
```

[ ]: <AxesSubplot:xlabel='Category', ylabel='Density'>



L'entraînement est souvent fait sur une partie seulement du dataset. Le problème avec des données aussi déséquilibrées est qu'un tirage iid sur un sample de petite taille (devant celle du dataset entier) a peu de chance de conserver les proportions des crimes du dataset entier (à cause des ordres de grandeur qui séparent les pourcentages). Les fonctions suivantes visent à ajuster les samples iid pour forcer le respect de ces proportions (ainsi que la présence d'au moins un représentant de toutes les catégories).

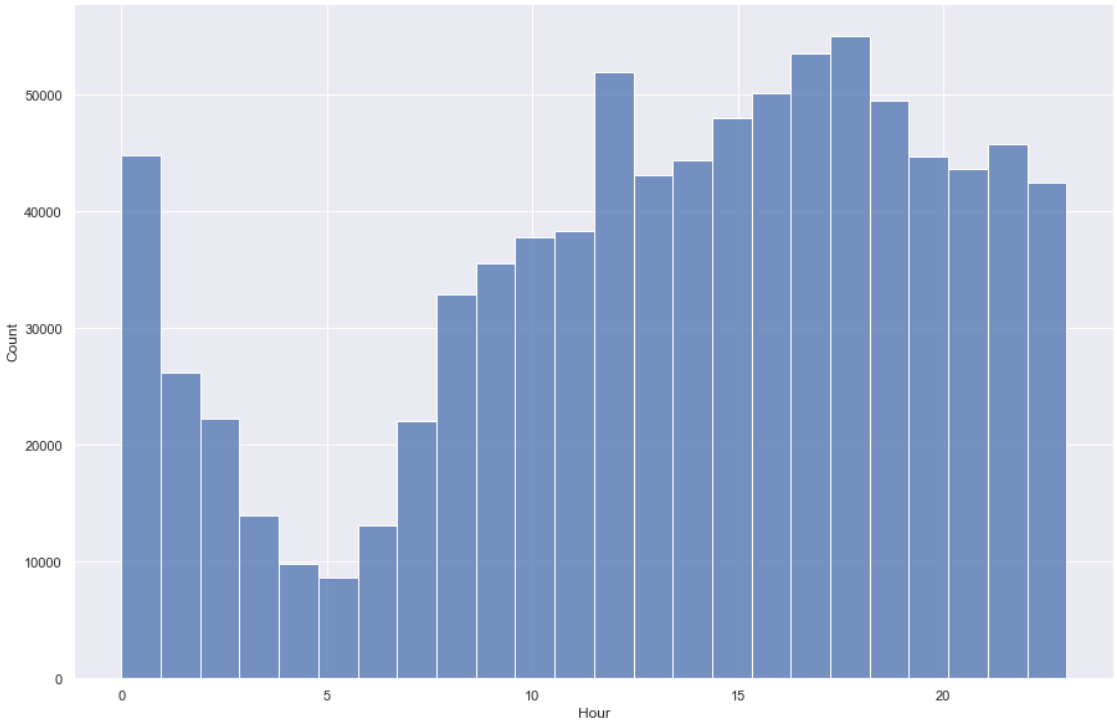
[ ]:
```
def percentage_per_category(data):
    percentage_per_category = np.zeros(39)
    for c in range(39):
        #compute the percentage of category c in data
        percentage_per_category[c] = data.loc[data['Category']==c,:].count().
 ↪unique()/len(data)
    return percentage_per_category
```

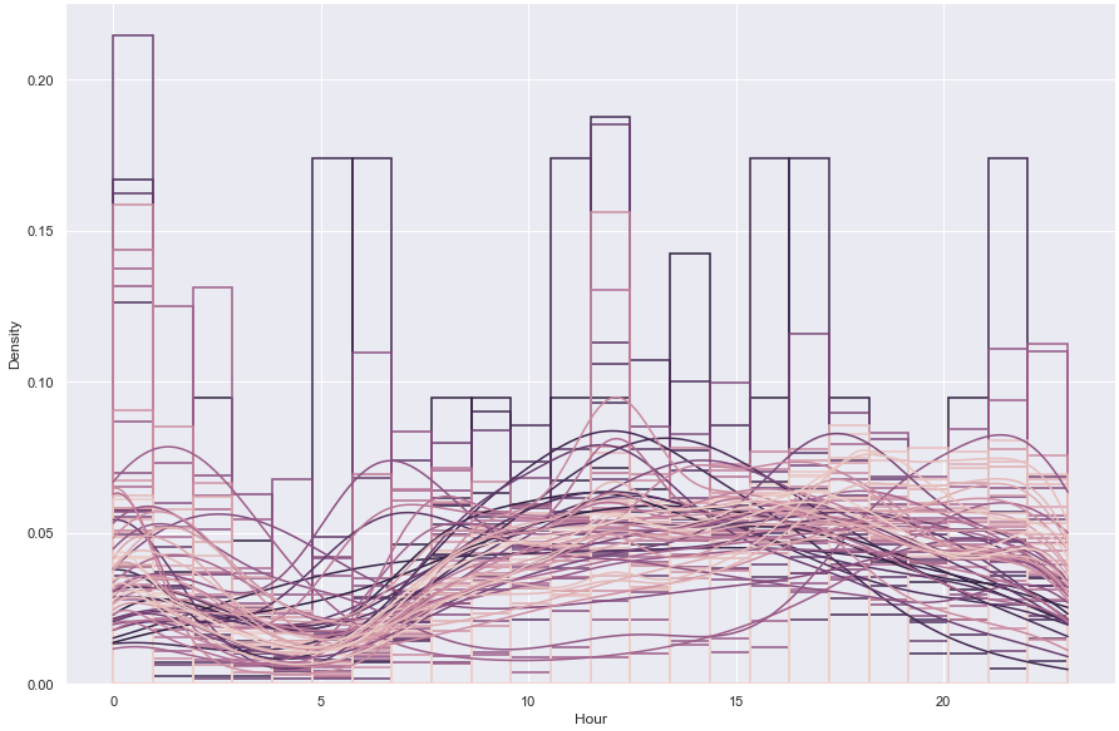
[ ]:
```
# this function enables to add data in the sample in order to minimize (less␣
 ↪than epsilon = 1e-3 here) the gap between the proportion of
# each category in the real data (df) and in the sample (dfsample)
def rebuild_data(data,sample, epsilon = 1e-3):
    real_percentage = percentage_per_category(data)
    proportion = percentage_per_category(sample)
    category = np.sort(data['Category'].unique())
    N = len(category)
    for i in range(N):
        m = category[i]
        incomplete = True
        error = np.abs(real_percentage[m] - proportion[m])
        while incomplete:
            data_cat = data.loc[data['Category'] == m,:]
            n = len(data_cat)
            index = np.random.randint(1,n)
            new_line = data_cat[index:index+1]
            sample = sample.append(new_line,ignore_index = True)
            proportion = percentage_per_category(sample)
            incomplete = real_percentage[m] - proportion[m] > epsilon
            error = np.abs(real_percentage[m] - proportion[m])
    return sample
```

Les données temporelles sont sans doute celles qui apportent le plus d'information sur la catégorie de crime. Comme on peut le remarquer sur les graphes ci-dessous, ces données, mis à part les années, sont cycliques. Il semble de plus que la répartition soit légérement différente selon les catégories de crimes (avec la même allure général, c'est à dire que la plupart des crimes sont commis la journée). C'est ce qu'on peut voir sur le second graphe en regardant en particulier les kernel density estimations (les courbes lisses).

[ ]:
```
plt.figure(figsize=(15,10))
sns.histplot(
    data = train_data,
```

```
    x = 'Hour',
    bins=24
)

plt.show()
plt.figure(figsize=(15,10))
sns.histplot(
    data = train_data,
    x = 'Hour',
    hue='Category',
    bins=24,
    legend=False,
    stat='density',
    common_norm=False,
    kde=True,
    fill=False
)

plt.show()
```





Pour conserver le caractère cyclique de ces données, chacune va être remplacée par deux valeurs : son sinus et son cosinus. Cela permet de placer les valeurs sur un cercle (assez logique pour lire l'heure!). On applique également cela à Year avec une période très grande pour des raisons d'homogénéité. On peut également choisir de simplement la normaliser pour ramener la donnée entre -1 et 1.

[ ]:
```
#Cell to run if you want to have cyclic year

train_data['Sin_Year'] = np.sin(2 * np.pi * train_data['Year'] / (10000))  #␣
↪cycle rpz to explore
train_data['Cos_Year'] = np.cos(2 * np.pi * train_data['Year'] / (10000))


train_data.drop(['Year'],inplace =True, axis=1)
```

[ ]:
```
train_data['Sin_Time'] = np.sin(2 * np.pi * train_data['Time'] / (10000))  #␣
↪cycle rpz to explore
train_data['Cos_Time'] = np.cos(2 * np.pi * train_data['Time'] / (10000))

train_data['Sin_Hour'] = np.sin(2 * np.pi * train_data['Hour'] / 24)  # cycle␣
↪rpz to explore
train_data['Cos_Hour'] = np.cos(2 * np.pi * train_data['Hour'] / 24)
```

```
train_data['Sin_Day_m'] = np.sin(2 * np.pi * train_data['Day'] / 31)  # cycle␣
↪rpz to explore
train_data['Cos_Day_m'] = np.cos(2 * np.pi * train_data['Day'] / 31)

train_data['Sin_Month'] = np.sin(2 * np.pi * train_data['Month'] / 12)  # cycle␣
↪rpz to explore
train_data['Cos_Month'] = np.cos(2 * np.pi * train_data['Month'] / 12)

train_data['Sin_Day_w'] = np.sin(2 * np.pi * train_data['DayOfWeek'] / 7)  #␣
↪cycle rpz to explore
train_data['Cos_Day_w'] = np.cos(2 * np.pi * train_data['DayOfWeek'] / 7)

train_data.drop(['Dates','Time','Hour','Day','Month','DayOfWeek'],inplace␣
↪=True, axis=1)

train_data.head()
```

[ ]:

	Category	X	Y	X_Bar_District	Y_Bar_District	Sin_Year	\
0	0	-122.425892	37.774599	-122.426647	37.786379	0.953927	
1	1	-122.425892	37.774599	-122.426647	37.786379	0.953927	
2	1	-122.424363	37.800414	-122.426647	37.786379	0.953927	
3	2	-122.426995	37.800873	-122.426647	37.786379	0.953927	
4	2	-122.438738	37.771541	-122.445448	37.770299	0.953927	

	Cos_Year	Sin_Time	Cos_Time	Sin_Hour	Cos_Hour	Sin_Day_m	Cos_Day_m	\
0	0.30004	-0.577573	-0.816339	-0.258819	0.965926	0.485302	-0.874347	
1	0.30004	-0.577573	-0.816339	-0.258819	0.965926	0.485302	-0.874347	
2	0.30004	0.137790	-0.990461	-0.258819	0.965926	0.485302	-0.874347	
3	0.30004	0.248690	-0.968583	-0.258819	0.965926	0.485302	-0.874347	
4	0.30004	0.248690	-0.968583	-0.258819	0.965926	0.485302	-0.874347	

	Sin_Month	Cos_Month	Sin_Day_w	Cos_Day_w
0	0.5	-0.866025	0.974928	-0.222521
1	0.5	-0.866025	0.974928	-0.222521
2	0.5	-0.866025	0.974928	-0.222521
3	0.5	-0.866025	0.974928	-0.222521
4	0.5	-0.866025	0.974928	-0.222521

[ ]:
```
#Cell to run if you want to have year normalized between -1 and 1
def normalize_year(data):
    years = data.Dates.dt.year.unique()
    years = sorted(years)
    a = 2/(years[-1] - years[0])
    b = -a*(years[-1]+years[0])/2
    data['Year'].replace(to_replace=years,value=[a*year +b for year in␣
↪years],inplace=True)
```

```
normalize_year(train_data)
train_data.head()
```

On applique maintenant les mêmes modifications au set de test et on va pouvoir commencer à tester des algorithmes.

[ ]:
```
test_data = pd.read_csv('data/test.csv')
print('test data loaded')
test_data.drop(['Address'],inplace =True, axis=1)  #data useless, unsignificant
test_data['DayOfWeek'].
 ↪replace(to_replace=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
 ↪for i in range(0,7)],inplace=True)

test_data.Dates = pd.to_datetime(test_data.Dates)
test_data['Time'] = test_data.Dates.dt.hour*3600 + test_data.Dates.dt.minute*60␣
 ↪+ test_data.Dates.dt.second
test_data['Hour'] = test_data.Dates.dt.hour
test_data['Day'] = test_data.Dates.dt.day
test_data['Month']=test_data.Dates.dt.month
test_data['Year']=test_data.Dates.dt.year
for key in ['Time','Hour','Day','Month','Year']:
    test_data[key]=pd.to_numeric(test_data[key])
District_barycenters(test_data)

test_data['Sin_Time'] = np.sin(2 * np.pi * test_data['Time'] / (24*60*60))  #␣
 ↪cycle rpz to explore
test_data['Cos_Time'] = np.cos(2 * np.pi * test_data['Time'] / (24*60*60))

test_data['Sin_Hour'] = np.sin(2 * np.pi * test_data['Hour'] / 24)  # cycle rpz␣
 ↪to explore
test_data['Cos_Hour'] = np.cos(2 * np.pi * test_data['Hour'] / 24)

test_data['Sin_Day_m'] = np.sin(2 * np.pi * test_data['Day'] / 31)  # cycle rpz␣
 ↪to explore
test_data['Cos_Day_m'] = np.cos(2 * np.pi * test_data['Day'] / 31)

test_data['Sin_Month'] = np.sin(2 * np.pi * test_data['Month'] / 12)  # cycle␣
 ↪rpz to explore
test_data['Cos_Month'] = np.cos(2 * np.pi * test_data['Month'] / 12)

test_data['Sin_Day_w'] = np.sin(2 * np.pi * test_data['DayOfWeek'] / 7)  #␣
 ↪cycle rpz to explore
test_data['Cos_Day_w'] = np.cos(2 * np.pi * test_data['DayOfWeek'] / 7)

test_data.drop(['Id','Time','Hour','Day','Month','DayOfWeek','Dates'],inplace␣
 ↪=True, axis=1)
```

```
test_data.head()
```

```
test data loaded
```

[ ]:
```
             X          Y  Year  X_Bar_District  Y_Bar_District  Sin_Time  \
0 -122.399588  37.735051   1.0     -122.393457       37.740094 -0.004363
1 -122.391523  37.732432   1.0     -122.393457       37.740094 -0.039260
2 -122.426002  37.792212   1.0     -122.426336       37.795198 -0.043619
3 -122.437394  37.721412   1.0     -122.428722       37.728411 -0.065403
4 -122.437394  37.721412   1.0     -122.428722       37.728411 -0.065403

   Cos_Time  Sin_Hour  Cos_Hour  Sin_Day_m  Cos_Day_m  Sin_Month  Cos_Month  \
0  0.999990 -0.258819  0.965926   0.897805  -0.440394        0.5  -0.866025
1  0.999229 -0.258819  0.965926   0.897805  -0.440394        0.5  -0.866025
2  0.999048 -0.258819  0.965926   0.897805  -0.440394        0.5  -0.866025
3  0.997859 -0.258819  0.965926   0.897805  -0.440394        0.5  -0.866025
4  0.997859 -0.258819  0.965926   0.897805  -0.440394        0.5  -0.866025

   Sin_Day_w  Cos_Day_w
0  -0.781831    0.62349
1  -0.781831    0.62349
2  -0.781831    0.62349
3  -0.781831    0.62349
4  -0.781831    0.62349
```

[ ]:
```
#Cell to run if you want to have cyclic year

test_data['Sin_Year'] = np.sin(2 * np.pi * test_data['Year'] / (10000))  #␣
↪cycle rpz to explore
test_data['Cos_Year'] = np.cos(2 * np.pi * test_data['Year'] / (10000))


train_data.drop(['Year'],inplace =True, axis=1)
```

[ ]:
```
#Run to normalize year

normalize_year(test_data)
```

[ ]:
```
train_data.to_csv("pre_processing_train_data.csv")
test_data.to_csv("pre_processing_test_data.csv")
```