Data Analysis with Python A0131637 9

Module 02

## Data - Preprocessing (Data Cleaning, Data Wrangling)

Raw Data ⟹ Convert Transform ⟹ Another forma ⟹ Prepare to Analysis

### Learning Objectives

- Identify and handle missing values → Cuando los datos están vacíos
- Data formatting → Datos de varias fuentes, formatos, unidades, convenciones
- Data normalization (centering/scaling) → Diferentes rangos
- Data Binning → Cria categorías de un conjunto de datos numéricos
- Turning Categorical Values to Numeric Values

### Simple Dataframe Operations

Python realiza operaciones sobre las columnas

Column (Samples) →
→ `df["symboling"]`
→ `df["body-style"]`
} Acceso a estas columnas

→
→ `df["symboling"] = df["symboling"] + 1` ⟹ Suma en toda la columna "symboling"

# Dealing with Missing Values in Python

- Missing value occur when no data value is store for a variable (feature) in an observation
- Could be represented as "?", "N/A", 0 or just a blank cell

## How to deal with missing data?

→ Check with the data collection source

Drop the missing values
- drop the variable
- drop the data entry

Replace the missing values
- replace it with an average
- replace it by frequency
- replace it based on other functions

Leave it as missing data

Como descartar valores faltantes en Python??

* Use dataframes.dropna():

| highway-mpg | price | highway-mpg | price |
|---|---|---|---|
| 20 | 23875 | 26 | 23875 |
| 22 | NaN | 29 | 16430 |
| 29 | 16430 | ... | ... |
| ... | ... | | |

axis = 0 drop the entire row
axis = 1 drop the entire column

df.dropna(subset = ["price"], axis = 0, inplace = True)

df = df.dropna(subset = ['price'], axis = 0)

df.dropna(subset = ["price"], axis = 0, inplace = True)

How to replace missing values in Python

dataframe.replace(missing_value, new_value):

```
mean = df["normalized-losses"].mean()
df["normalized-losses"].replace(np.nan, mean)
```

## Data Formating in Python

- Data are usually collected from different places and stored in different formats
- Bringing data into a common standard of expressions allow users to make meaningful comparison

Non-formatted
- confusing
- hard to aggregate
- hard to compare

| City |
| --- |
| NY |
| New York |
| N.Y. |
| N.Y. |

| City |
| --- |
| New York |
| New York |
| New York |
| New York |

formatted
- more clean
- easy to aggregate
- easy to compare

## Applying calculations to an entire column

- Convert "mpg" to "L/100Km" in Car dataset

| city mpg | city-L/100km |
|---|---|
| 21 | 11.2 |
| 21 | 11.2 |
| 19 | 12.4 |

```
df["city-mpg"] = 235 / df["city-mpg"]
df.rename(columns = {"city-mpg": "city-L/100km"}, inplace = True)
```

## Incorrect data types

- Sometimes the wrong data type is assigned to a feature

```
df["price"].tail(5)
```

| | |
|---|---|
| 200 | 16845 |
| 201 | 19045 |
| 202 | 21485 |
| 203 | 22470 |
| 204 | 22625 |

Name: price, dtype: object

## Data Types in Python and Pandas

- There are many data types in pandas
- Objects: "A", "Hello"...
- Int64: 1, 3, 5
- Float64: 2.123, 632.31.0.12

## Converting data types

To identify data types:

- Use dataframe.dtypes() to identify data type
- Use dataframe.astype() to convert data type

Example: convert data type to integer in column "price"

```
df["price"] = df["price"].astype("int")
```

# Data Normalization in Python

- Uniform the features value with different range.

| Length | Width | Height |
|---|---|---|
| 168.8 | 64.1 | 48.8 |
| 168.8 | 64.1 | 48.8 |
| 171.2 | 65.5 | 52.4 |
| 176.6 | 66.2 | 54.3 |

| | | | |
|---|---|---|---|
| scale | [150, 250] | [50, 100] | [50, 100] |
| impact | large | small | small |

| Age | Income |
|---|---|
| 20 | 100000 |
| 30 | 20000 |
| 40 | 500000 |

Not-normalized

- "Age" and "income" are in different range
- Hard to compare.
- "Income" will influence the result more

| Age | Income |
|---|---|
| 0.2 | 0.2 |
| 0.3 | 0.04 |
| 0.4 | 1 |

Normalized

- similar value range
- similar intansic influence on analytical model.

# Binning

- Binning: Grouping of values into "bins"
- Converts numeric into categorical variables
- Group a set of numerical values into a set of "bins"
- "price" is a feature range from 5,000 to 45,500 (in order to have a better representation of price)

price: 5000, 10000, 12000, 12000, 30000, 31000, 39000, ~~40000~~, ~~45~~500

bins: low, mid, high

## Binning in Python Pandas

```
bins = np.linspace(min(df["price"]), max(df["price"]), 4)

group_names = ["Low", "Medium", "High"]

df["price-binned"] = pd.cut(df["price"], bins, labels = group_names,
                            include_lowest = True)
```

## Visualizing Binned data

### E.g. Histograms

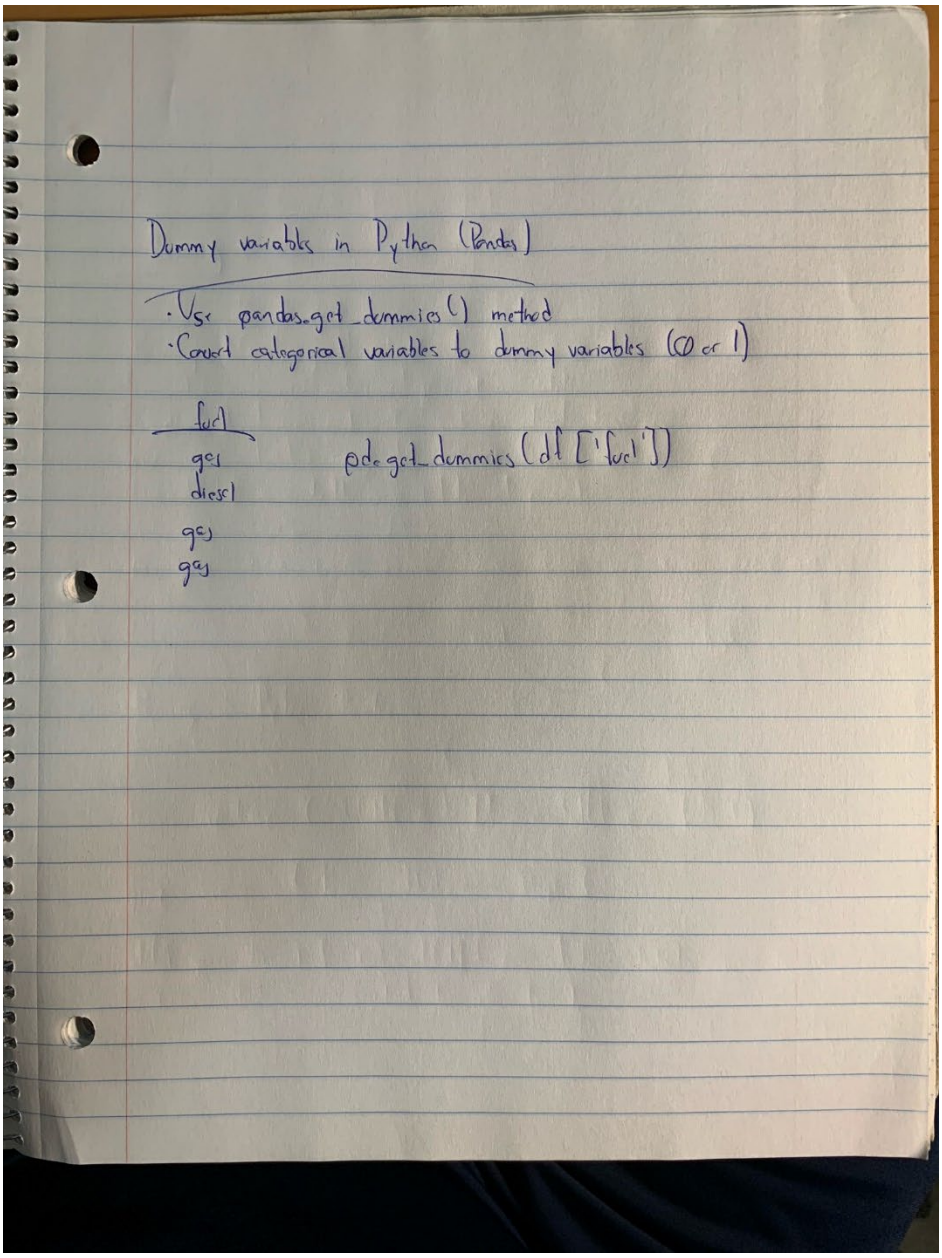
Turning
Categorical → Numeric

Problem:
- Most statistical models cannot take in the objects / strings as input

Solution:
- Add dummy variables for each unique category
- Assign 0 or 1 in each category

"One-hot encoding"

| Car | fuel | ~ | gas | diesel |
|---|---|---|---|---|
| A | gas | ~ | 1 | 0 |
| B | diesel | ~ | 0 | 1 |
| C | gas | ~ | 1 | 0 |
| D | gas | ~ | 1 | 0 |



Dummy variables in Python (Pandas)

- Use pandas.get_dummies() method
- Convert categorical variables to dummy variables (0 or 1)

| fuel |
| --- |
| gas |
| diesel |
| gas |
| gas |

pd.get_dummies(df['fuel'])

# Laboratorio #02

## Data Wrangling

- Manejar valores perdidos.
- Corregir formato de los datos.
- Estandarizar y normalizar datos.

```
#Importar las librerias de Pandas
import pandas as pd
import matplotlib.pylab as plt
```

```
#Colocamos la ruta del archivo, y los encabezados de las diferentes columnas
filename = "/content/sample_data/imports-85.data"

headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style"
         "drive-wheels","engine-location","wheel-base", "length","width","height","curb-weight","engine
         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio","horsepowe
         "peak-rpm","city-mpg","highway-mpg","price"]
```

```
#Lectura del conjunto de datos
df = pd.read_csv(filename, names = headers)
```

```
#Observamos los primeros 5 renglones del conjunto de datos
df.head()
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 |
| **1** | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 |
| **2** | 1 | ? | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 |
| **3** | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 |
| **4** | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 |

5 rows × 26 columns

# Identificando y manejando valores perdidos

```
import numpy as np

#Reemplazamos "?" con NaN
df.replace("?", np.nan, inplace = True)
df.head(5)
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | dr wh |
|---|---|---|---|---|---|---|---|---|
| **0** | 3 | NaN | alfa-romero | gas | std | two | convertible | |
| **1** | 3 | NaN | alfa-romero | gas | std | two | convertible | |
| **2** | 1 | NaN | alfa-romero | gas | std | two | hatchback | |
| **3** | 2 | 164 | audi | gas | std | four | sedan | |
| **4** | 2 | 164 | audi | gas | std | four | sedan | |

5 rows × 26 columns

```
#Evaluación de los datos perdidos
#Metodos para detectar datos perdidos
missing_data = df.isnull()
missing_data.head(5)
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels |
|---|---|---|---|---|---|---|---|---|
| **0** | False | True | False | False | False | False | False | False |
| **1** | False | True | False | False | False | False | False | False |
| **2** | False | True | False | False | False | False | False | False |
| **3** | False | False | False | False | False | False | False | False |
| **4** | False | False | False | False | False | False | False | False |

5 rows × 26 columns

```
#Conteo de valores perdidos en cada columna
for column in missing_data.columns.values.tolist():
  print(column)
  print(missing_data[column].value_counts())
  print("")
```

```
    Name: width, dtype: int64

    height
    False    205
    Name: height, dtype: int64


    curb-weight
    False    205
    Name: curb-weight, dtype: int64

    engine-type
    False    205
    Name: engine-type, dtype: int64

    num-of-cylinders
    False    205
    Name: num-of-cylinders, dtype: int64

    engine-size
    False    205
    Name: engine-size, dtype: int64

    fuel-system
    False    205
    Name: fuel-system, dtype: int64

    bore
    False    201
    True       4
    Name: bore, dtype: int64

    stroke
    False    201
    True       4
    Name: stroke, dtype: int64

    compression-ratio
    False    205
    Name: compression-ratio, dtype: int64

    horsepower
    False    203
    True       2
    Name: horsepower, dtype: int64

    peak-rpm
    False    203
    True       2
    Name: peak-rpm, dtype: int64

    city-mpg
    False    205
    Name: city-mpg, dtype: int64
```

```
highway-mpg
False    205
Name: highway-mpg, dtype: int64
```

```
#Lidiar con datos perdidos o faltantes
  #Eliminar toda la columna
  #Eliminar todo el renglon
#Reemplazar los datos
  #Reemplazar por la media.
  #Reemplazar por la frecuencia
  #Reemplazar basado en otras funciones

#Calcula el valor de la media de la columna "normalized-losses"
avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0)
print("Average of normalized-loses:", avg_norm_loss)
```

```
Average of normalized-loses: 122.0
```

```
#Reemplaza "NaN" con el valor medio dentro de la columna "normalized-losses"
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)


#Calcula el valor de la media de la columna "bore"
avg_bore = df["bore"].astype("float").mean(axis=0)
print("Average of bore:", avg_bore)
```

```
Average of bore: 3.3297512437810943
```

```
#Reemplaza "NaN" con el valor medio dentro de la columna "bore"
df["bore"].replace(np.nan, avg_bore, inplace=True)


#Reemplaza "NaN" con el valor medio dentro de la columna "stroke"
avg_stroke = df["stroke"].astype("float").mean(axis=0)
print("Average of stroke:", avg_stroke)
df["stroke"].replace(np.nan, avg_stroke, inplace=True)
```

```
Average of stroke: 3.255422885572139
```

```
#Reemplaza "NaN" con el valor medio dentro de la columna "horsepower"
avg_horsepower = df["horsepower"].astype("float").mean(axis=0)
print("Average of horsepower:", avg_horsepower)
df["horsepower"].replace(np.nan, avg_horsepower, inplace=True)
```

```
Average of horsepower: 104.25615763546797
```

```
#Reemplaza "NaN" con el valor medio dentro de la columna "peak-rpm"
avg_peakrpm = df["peak-rpm"].astype("float").mean(axis=0)
print("Average of peak rpm:", avg_peakrpm)
df["peak-rpm"].replace(np.nan, avg_peakrpm, inplace=True)
```

```
Average of peak rpm: 5125.369458128079
```

```
#Observamos cuales son los valores presentes en la columna
df["num-of-doors"].value_counts()
```

```
four    114
two      89
Name: num-of-doors, dtype: int64
```

```
#Observamos cual es el valor más frecuente
df["num-of-doors"].value_counts().idxmax()
```

```
'four'
```

```
#Reemplazar "NaN" con el valor más frecuente
df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

```
#Eliminamos todos los renglones que no tengan datos en la columna de precio
df.dropna(subset=["price"], axis=0, inplace=True)
df.reset_index(drop=True, inplace=True)
df.head()

#EL CONJUNTO DE DATOS NO TIENE VALORES FALTANTES
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | dr wh |
|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122.0 | alfa-romero | gas | std | two | convertible | |
| **1** | 3 | 122.0 | alfa-romero | gas | std | two | convertible | |
| **2** | 1 | 122.0 | alfa-romero | gas | std | two | hatchback | |
| **3** | 2 | 164 | audi | gas | std | four | sedan | |
| **4** | 2 | 164 | audi | gas | std | four | sedan | |

5 rows × 26 columns

```
#Corregimos el formato de los datos

df.dtypes
```

```
symboling             int64
normalized-losses    object
make                 object
fuel-type            object
aspiration           object
num-of-doors         object
body-style           object
```

```
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

```
#Convertimos el tipo de datos al formato apropiado
df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
df[["price"]] = df[["price"]].astype("float")
df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
```

```
df.dtypes
```

```
symboling              int64
normalized-losses      int64
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower            object
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
dtype: object
```

# Estandarizacion de los datos

```
df.head()
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | dr wh |
|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122 | alfa-romero | gas | std | two | convertible | |
| **1** | 3 | 122 | alfa-romero | gas | std | two | convertible | |
| **2** | 1 | 122 | alfa-romero | gas | std | two | hatchback | |
| **3** | 2 | 164 | audi | gas | std | four | sedan | |
| **4** | 2 | 164 | audi | gas | std | four | sedan | |

5 rows × 26 columns

```
#Convierte mpg a L/100 km por la operación matematica
df["city-L/100km"] = 235/df["city-mpg"]
df.head()
```

| uel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | fuel-system | b |
|---|---|---|---|---|---|---|---|---|---|
| gas | std | two | convertible | rwd | front | 88.6 | ... | mpfi | |
| gas | std | two | convertible | rwd | front | 88.6 | ... | mpfi | |
| gas | std | two | hatchback | rwd | front | 94.5 | ... | mpfi | |
| gas | std | four | sedan | fwd | front | 99.8 | ... | mpfi | |
| gas | std | four | sedan | 4wd | front | 99.4 | ... | mpfi | |

```
#Transformamos la columna "highway-mpg" to "highway-L/100km"
df["highway-mpg"] = 235/df["highway-mpg"]
#Renombra la columna
```

```
df.rename(columns={"highway-mpg":"highway-L/100km"}, inplace=True)
df.head()
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | dr wh |
|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122 | alfa-romero | gas | std | two | convertible | |
| **1** | 3 | 122 | alfa-romero | gas | std | two | convertible | |
| **2** | 1 | 122 | alfa-romero | gas | std | two | hatchback | |
| **3** | 2 | 164 | audi | gas | std | four | sedan | |
| **4** | 2 | 164 | audi | gas | std | four | sedan | |

5 rows × 27 columns

# Normalizacion de Datos

```
#Normalizamos el valor en el rango de 0 y 1
#Reemplazamos el valor original por el (valor original)/(valor maximo)
df['length'] = df['length']/df['length'].max()
df['width'] = df['width']/df['width'].max()
df['height'] = df['height']/df['height'].max()
df[["length","width","height"]].head()
```

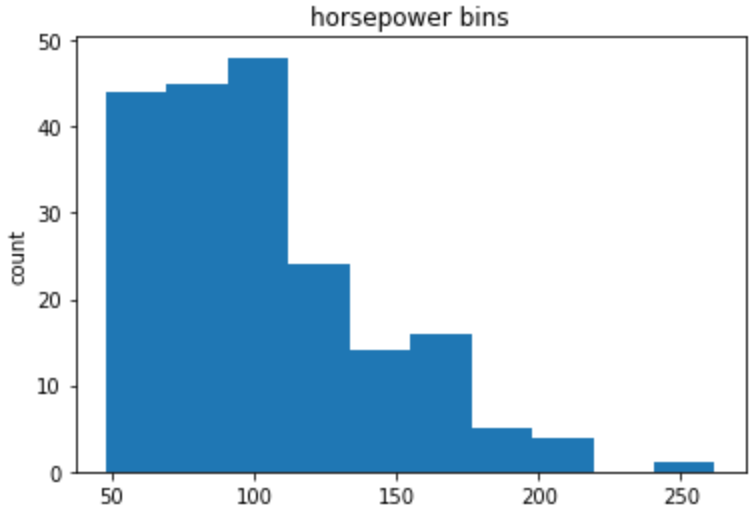
| | length | width | height |
|---|---|---|---|
| **0** | 0.811148 | 0.890278 | 0.816054 |
| **1** | 0.811148 | 0.890278 | 0.816054 |
| **2** | 0.822681 | 0.909722 | 0.876254 |
| **3** | 0.848630 | 0.919444 | 0.908027 |
| **4** | 0.848630 | 0.922222 | 0.908027 |

```
#Binning, proceso de transformar valores numericos en variables categoricas
df["horsepower"]=df["horsepower"].astype(int, copy=True)
```

```
#Realizamos un histograma de los datos
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["horsepower"])
```

```
# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```




```
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
bins
```

```
array([ 48.        , 119.33333333, 190.66666667, 262.        ])
```

```
group_names = ['Low', 'Medium', 'High']
```

```
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names, include_lowest=True )
df[['horsepower','horsepower-binned']].head(20)
```

| | horsepower | horsepower-binned |
|---|---|---|
| **0** | 111 | Low |
| **1** | 111 | Low |
| **2** | 154 | Medium |
| **3** | 102 | Low |
| **4** | 115 | Low |
| **5** | 110 | Low |
| **6** | 110 | Low |
| **7** | 110 | Low |

```
df["horsepower-binned"].value_counts()
```

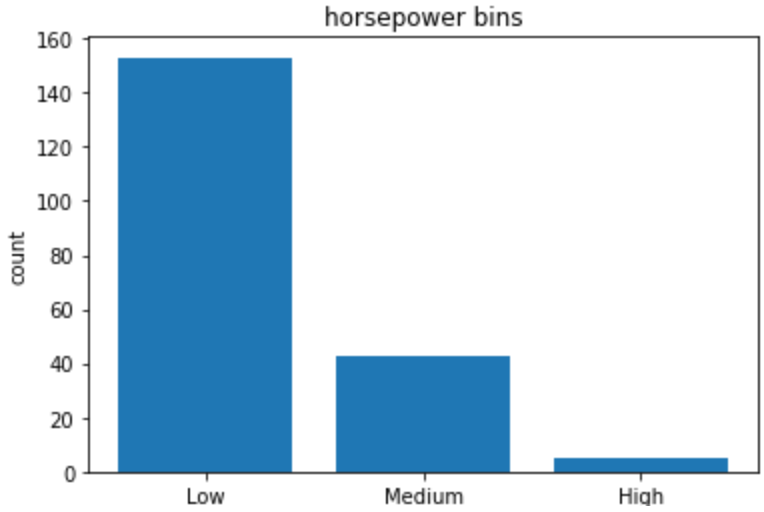
```
Low       153
Medium     43
High        5
Name: horsepower-binned, dtype: int64
```

| | | |
|---|---|---|
| **13** | 121 | Medium |

```
import matplotlib as plt
from matplotlib import pyplot
pyplot.bar(group_names, df["horsepower-binned"].value_counts())

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```

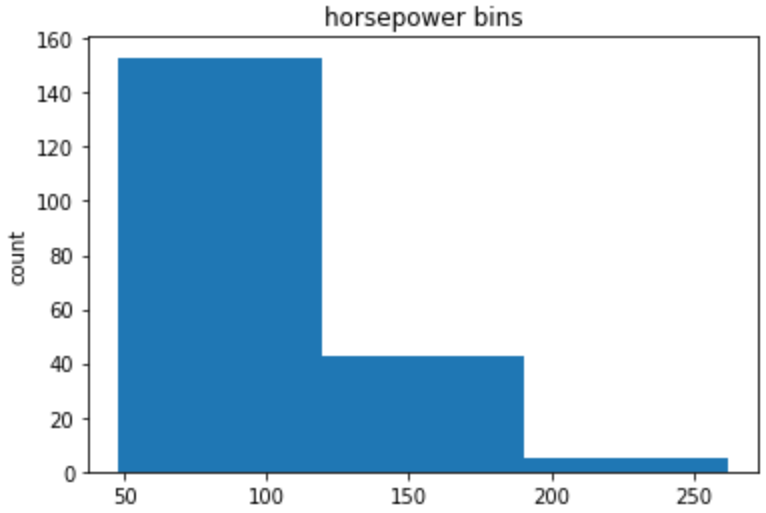



```
import matplotlib as plt
from matplotlib import pyplot


# draw historgram of attribute "horsepower" with bins = 3
plt.pyplot.hist(df["horsepower"], bins = 3)
```

```
# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```




# Variable Indicador (Variable Dummy)

```
#Nombre de las columnas
df.columns
```

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-L/100km', 'price', 'city-L/100km', 'horsepower-binned'],
      dtype='object')
```

```
#Variable dummy
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

| | diesel | gas |
|---|---|---|
| **0** | 0 | 1 |
| **1** | 0 | 1 |
| **2** | 0 | 1 |
| **3** | 0 | 1 |
| **4** | 0 | 1 |

```
#Cambia el nombre de la columna
```

```
dummy_variable_1.rename(columns={'gas':'fuel-type-gas', 'diesel':'fuel-type-diesel'}, inplace=True)
dummy_variable_1.head()
```

| | fuel-type-diesel | fuel-type-gas |
|---|---|---|
| **0** | 0 | 1 |
| **1** | 0 | 1 |
| **2** | 0 | 1 |
| **3** | 0 | 1 |
| **4** | 0 | 1 |

```
df = pd.concat([df, dummy_variable_1], axis=1)
df.drop("fuel-type", axis = 1, inplace=True)
df.head()
```

| num-of-doors | body-style | drive-wheels | engine-location | wheel-base | length | ... | compression-ratio | hc |
|---|---|---|---|---|---|---|---|---|
| two | convertible | rwd | front | 88.6 | 0.811148 | ... | 9.0 | |
| two | convertible | rwd | front | 88.6 | 0.811148 | ... | 9.0 | |
| two | hatchback | rwd | front | 94.5 | 0.822681 | ... | 9.0 | |
| four | sedan | fwd | front | 99.8 | 0.848630 | ... | 10.0 | |
| four | sedan | 4wd | front | 99.4 | 0.848630 | ... | 8.0 | |

```
#Crea una variable dummy
dummy_variable_2 = pd.get_dummies(df['aspiration'])
dummy_variable_2.rename(columns={'std':'aspiration-std', 'turbo': 'aspiration-turbo'}, inplace=True)
dummy_variable_2.head()
```

| | aspiration-std | aspiration-turbo |
|---|---|---|
| **0** | 1 | 0 |
| **1** | 1 | 0 |
| **2** | 1 | 0 |
| **3** | 1 | 0 |
| **4** | 1 | 0 |

```
#Une el nuevo dataframe al dataframe original
df = pd.concat([df, dummy_variable_2], axis=1)
#Quitamos la columna original "aspiration" del "dataframe"
df.drop('aspiration', axis = 1, inplace=True)


#Salvamos el conjunto de datos
df.to_csv('clean_df.csv')
```

Course Progress for 'Francisco_Arias' (A01316379@tec.mx)