In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Load the Auto dataset in a DataFrame using Pandas

In [ ]:

```
data = pd.read_csv('data/auto-dataset.csv')
print(data)
```

```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0         130    3504          12.0    70
1    15.0          8         350.0         165    3693          11.5    70
2    18.0          8         318.0         150    3436          11.0    70
3    16.0          8         304.0         150    3433          12.0    70
4    17.0          8         302.0         140    3449          10.5    70
..    ...        ...           ...         ...     ...           ...   ...
387  27.0          4         140.0          86    2790          15.6    82
388  44.0          4          97.0          52    2130          24.6    82
389  32.0          4         135.0          84    2295          11.6    82
390  28.0          4         120.0          79    2625          18.6    82
391  31.0          4         119.0          82    2720          19.4    82

     origin                       name
0         1  chevrolet chevelle malibu
1         1          buick skylark 320
2         1         plymouth satellite
3         1              amc rebel sst
4         1                ford torino
..      ...                        ...
387       1            ford mustang gl
388       2                  vw pickup
389       1              dodge rampage
390       1                ford ranger
391       1                 chevy s-10

[392 rows x 9 columns]
```

1. Scatterplots between features

Using the plot() function in matplotlib and create scatterplots between all the variables.

Is the relationship between those variables linear? Describe exactly four different connections between the variables.

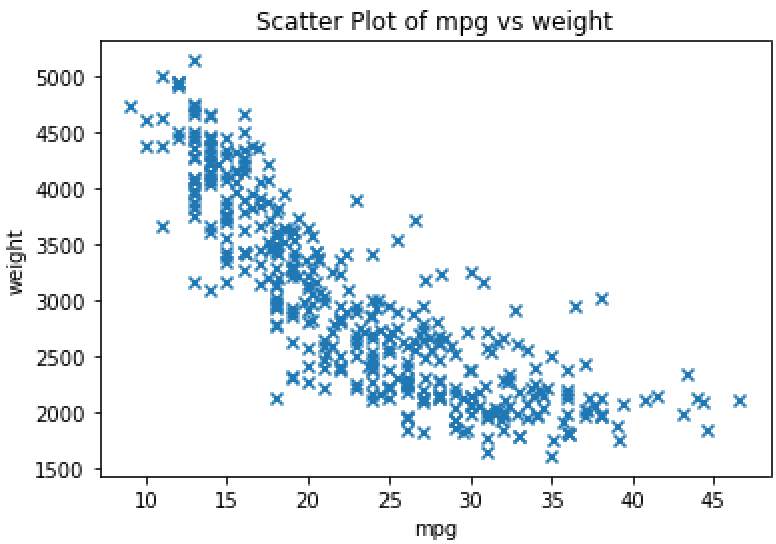
(Exclude the name variable, which is qualitative.)

In [ ]:

```
feat1 = data['mpg']
feat2 = data['weight']
plt.figure()
plt.scatter(feat1, feat2, marker='x')
plt.xlabel('mpg')
plt.ylabel('weight')
plt.title('Scatter Plot of mpg vs weight')
#Similarly add scatter plots for every pair of features
```

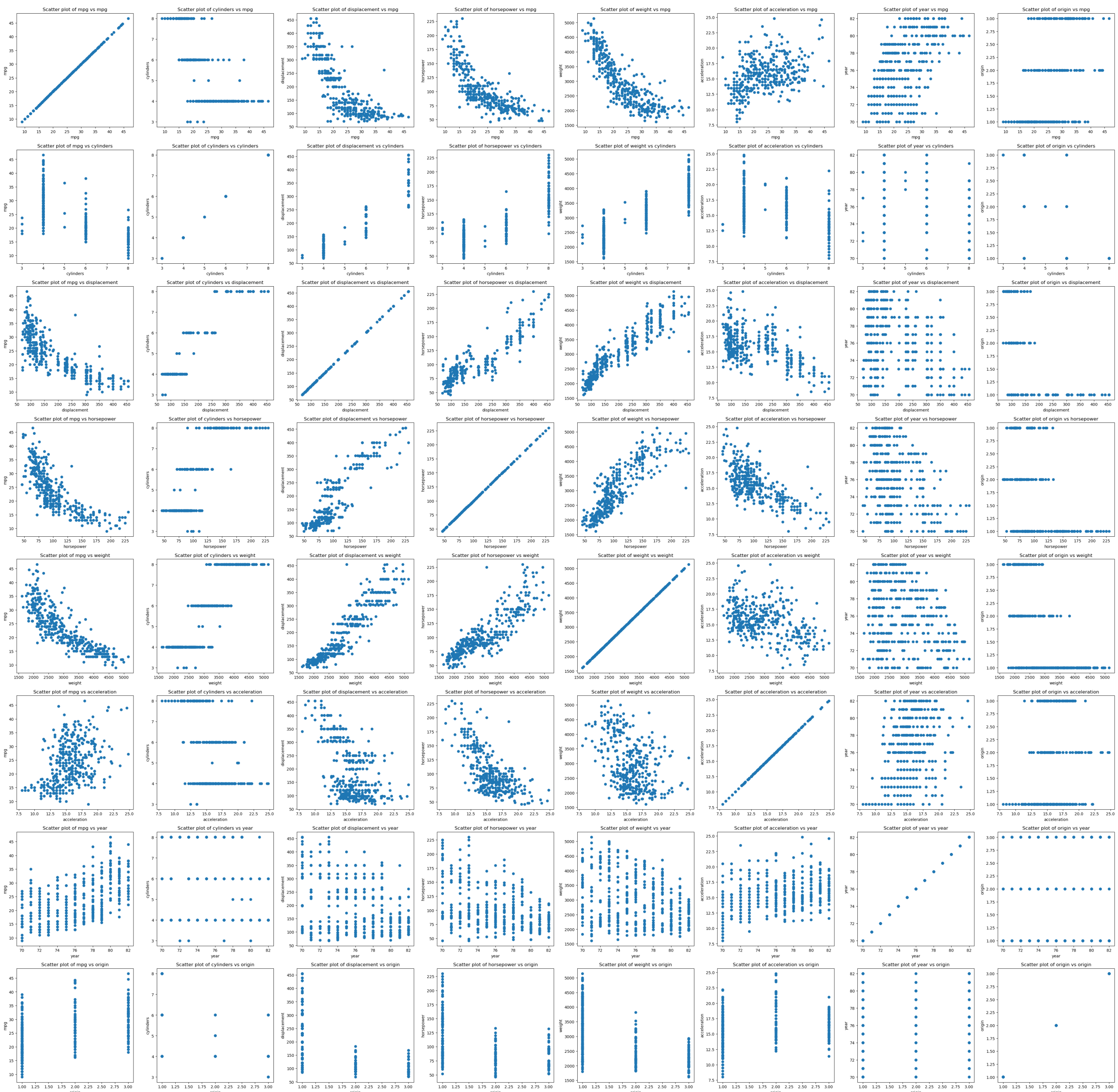
Out[ ]:

```
Text(0.5, 1.0, 'Scatter Plot of mpg vs weight')
```



In [ ]:

```
# Plot the feature plots to observe the relationships between all the predictors
feats = ["mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration", "y
figure, axis = plt.subplots(8, 8 , figsize=(50, 50), dpi=100)
for i in range(8):
    for j in range(8):
        index = (i ,j)
        plt.subplot2grid((8,8), index, rowspan = 1, colspan = 1)
        plt.scatter(data[feats[i]], data[feats[j]], label = '.')
        plt.title(f"Scatter plot of {feats[j]} vs {feats[i]}")
        plt.xlabel(feats[i])
        plt.ylabel(feats[j])
plt.show()
```



Describe exactly four different connections between the variables:

2/2

```
1) horsepower vs displacement appears to have a positive, roughly linear
relationship.
2) horsepower vs acceleratioj appears to have a negative, roughly linear
relationship (though starting to look like a banana).
3) mpg vs displacement, mpg vs horsepower, and mpg vs weight all show a
negative, banana-shaped (quadratic, perhaps?) relationship.
4) year vs origin doesn't appear to have any sort of relationship.
```

2. Correlation

Detect the two variable pairs in the scatterplots that appear to be the most highly correlated and anti-correlated, respectively.

Justify your choice using the np.corrcoef() function.

Do the results from np.corrcoef() differ from what you see in the plot?

In [ ]:

```
# Use np corrcoef to observe the correlation between  predictors
corr_disp_wt = np.corrcoef(data["displacement"], data["weight"])
print(f"displacement vs weight: {corr_disp_wt[0][1]}")
# displacement vs weight looks higly correlated, and indeed has a correlation coefficie
corr_disp_hp = np.corrcoef(data["displacement"], data["horsepower"])
print(f"diplacement vs horsepwer: {corr_disp_hp[0][1]}")
corr_accel_year = np.corrcoef(data["acceleration"], data["year"])
print(f"acceleration vs year: {corr_accel_year[0][1]}")
corr_orig_year = np.corrcoef(data["origin"], data["year"])
print(f"origin vs year: {corr_orig_year[0][1]}")
# origin vs year looks pretty uncorrelated, and has a correlation of only 0.18

# But let's test it, just to be sure.
feats = ["mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration", "y
min = 1
minners = ""
max = 0
maxers = ""
for i in range(8):
    for j in range(8):
        if i == j:
            continue
        score = abs(np.corrcoef(data[feats[i]], data[feats[j]]))
        if score[0][1] < min:
            min = score[0][1]
            minners = f"{feats[i]} vs {feats[j]}"
        if score[0][1] > max:
            max = score[0][1]
            maxers = f"{feats[i]} vs {feats[j]}"
print(f"Overall least correlated: {minners} @ {round(min,2)},\n Overall most correlated
```

```
displacement vs weight: 0.9329944040890104
diplacement vs horsepwer: 0.8972570018434686
acceleration vs year: 0.29031611333652
origin vs year: 0.18152771836633097
Overall least correlated: origin vs year @ 0.18,
 Overall most correlated: cylinders vs displacement @ 0.95
```

Results:

displacement vs weight: 0.9329944040890104

what about anti-cor?
anti-correlation means negative correlation.

1/2 diplacement vs horsepwer: 0.8972570018434686

acceleration vs year: 0.29031611333652

origin vs year: 0.18152771836633097

Overall least correlated: origin vs year @ 0.18,

Overall most correlated: cylinders vs displacement @ 0.95

3. Linear Regression

Perform simple linear regression with mpg as the response using the variables:

```
cylinders, displacement, horsepower, year
```

as features using the LinearRegression() function provided in the sklearn package.

Which predictors appear to have a statistically significant relationship to the outcome (use an appropriate measurement)?

How good are the resulting models (provide all four R2 values)?

In [ ]:

```python
from sklearn.linear_model import LinearRegression

linear_model = LinearRegression()

y = np.array(data['mpg'])
x = np.array(data['cylinders'])
print(x.shape, y.shape)
# Use the fit function and the score function in linear regression module of sklearn to
linear_model.fit(x.reshape(-1, 1), y)
linear_model.score(x.reshape(-1, 1), y)
```

```
(392,) (392,)
```

Out[ ]:

```
0.6046889889441246
```

In [ ]:

```python
# Fit the linear regression on mpg for the other features: cylinder, horsepower, year a
feats = ["cylinders", "displacement", "horsepower", "year"]
y = np.array(data['mpg'])
for feat in feats:
    linear_model = LinearRegression()
    x = np.array(data[feat])
    linear_model.fit(x.reshape(-1, 1), y)
    print(f"Score for mpg as a function of {feat}:", linear_model.score(x.reshape(-1, 1
```

```
Score for mpg as a function of cylinders: 0.6046889889441246
Score for mpg as a function of displacement: 0.6482294003193044
Score for mpg as a function of horsepower: 0.6059482578894348
Score for mpg as a function of year: 0.33702781330962295
```

Results:


```
Score for mpg as a function of cylinders: 0.6046889889441246
Score for mpg as a function of displacement: 0.6482294003193044
Score for mpg as a function of horsepower: 0.6059482578894348
Score for mpg as a function of year: 0.33702781330962295
```


4/4

mpg as a function of year seems to be the least correlated, with an R^2 of only 0.34

R^2 shows the goodness of fit

Out of a best possible R^2 value of 1.0, the other variables being between 0.60 and 0.65 does not seem to be all that high of an impact.

4. Multiple Linear Regression

Use the LinearRegression() function to perform one multiple linear regression with mpg as the response and all other variables except name as the predictors.

2/4

Use the score() and get params() functions to print the results.

Compare the full model to those generated in 5.3:

Comment on the goodness of fit?

```
(a) How is the model fit (using R^2)?
```

Comment on the sign of coeffs?

```
(b) What can you observe in the different models concerning the
significance of the relationship between response and individual
predictors?
    What does the sign of the coefficient (i.e. of the estimate) tell you
about the relationship between the predictor and the response?
    Provide an example from your fitted model.
```

In [ ]:

```
#Multiple Linear Regression
#Create a 2D matrix with each column representing all features
#Look up np.concatenate or np.stack
feats = ["mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration", "y
mpg_array = np.array(data["mpg"])
cylinders_array = np.array(data["cylinders"])
displacement_array = np.array(data["displacement"])
horsepower_array = np.array(data["horsepower"])
weight_array = np.array(data["weight"])
acceleration_array = np.array(data["acceleration"])
year_array = np.array(data["year"])
origin_array = np.array(data["origin"])

combined_arrays = np.stack((cylinders_array, displacement_array, horsepower_array, weig

#Print the predictions of the your fitted model and the weights of Linear Regression (L
multiple_linear_model = LinearRegression()
multiple_linear_model.fit(combined_arrays, mpg_array)
mlm_score = multiple_linear_model.score(combined_arrays, mpg_array)
print(mlm_score)
#params = multiple_linear_model.get_params()
#print(params)
coefs = multiple_linear_model.coef_
print(coefs)
predictions = multiple_linear_model.predict(combined_arrays)
print(predictions)
```

```
0.8214780764810599
[-0.49337632  0.01989564 -0.01695114 -0.00647404  0.08057584  0.75077268
  1.4261405 ]
[15.00095865 13.99929917 15.24044696 15.06190592 14.96717762 10.69562338
 10.6553509  10.69318951 10.21140958 13.11319398 15.29186157 14.14690266
 14.64696189 18.88015369 24.13062472 19.03847473 19.40643664 20.88068005
 25.45738302 27.13648841 21.03963303 22.25408365 22.7074409  23.25050717
 20.3005362   7.58306997  8.41074023  8.30037285  6.44696729 26.20815569
 23.39053912 25.73308381 21.46783861 16.23364804 17.52793963 17.90615253
 17.43525382 11.44975337 10.58385513 12.14999889 11.87950268  6.98211816
  8.88321853  6.24768971 19.73241559 23.0454129  17.79193175 18.88084577
 23.26421626 25.13116053 25.49443983 25.22718096 28.75457004 29.66193221
 27.53555535 25.15803406 26.28101808 24.47954013 26.03152794 23.5313529
 24.17758427 11.77971323 11.88636515 12.41807558 13.02233473 14.97586698
 10.21781246 10.59400653 10.80705681 11.4328463  25.38717749 13.63215327
```

```
12.81807391 11.44132342 12.83385802 20.40479902 24.02910196 20.91375517
25.73501085 23.04318302 26.06966453 24.82972756 24.08232262 27.31430135
13.56803385 15.72663966 14.80166104 13.75354471 15.40597996  9.11316466
12.54946701 12.17342553 12.58907167 10.40842679  9.13895663 15.41209672
19.87451504 19.5611008  21.19768398 21.30747992 20.8714236  28.68457937
 9.25995356  9.44005829 10.26512597 10.8598469  22.12705892 27.10765986
24.63256448 26.37596114 27.59026112 24.56264013 22.44813226 25.47613389
14.19310096 12.19610795 28.47323412 26.78124528 23.60437753 21.65696035
17.7323046  22.91659991 22.85694927 16.14480938 20.3806236  22.23331457
19.85580312 29.98624525 24.32417673 30.76017502 24.21820401 16.97485381
18.00971582 17.44005836 13.77224574 10.9854969  11.83497896 10.71579784
13.01240031 26.72394641 28.19392676 26.1032573  31.92807295 29.82441039
25.72165832 27.33843339 26.84143838 26.48161957 26.9854206  27.99467507
20.57949571 19.6449369  20.78200302 22.43502716 12.35099671 13.45751462
12.4008917  11.98140149 16.6846717  17.01074549 18.20660808 17.44824662
21.82068634 20.151559   20.84109951 29.28703926 24.20538532 22.89992468
24.71528484 26.02804741 27.4930715  27.01110186 21.16309705 29.16015946
21.22722108 24.41349693 23.09011142 22.81674455 24.22188715 32.09569464
26.68698266 28.64327193 25.06814935 26.73078129 28.29854643 14.77382297
15.02480599 16.80340145 15.46949369 21.39786251 20.96109442 22.83057359
22.84206873 28.71669964 27.99652467 29.9270473  32.83840974 18.8770368
20.51175467 19.12289695 22.59151822 30.61330685 31.13619763 30.17362697
25.34774195 22.44828555 16.659432   22.20933012 24.71142766 17.65541354
13.85034576 16.53964233 17.27834413 17.9215907  31.94430733 28.1369806
31.74200818 27.20228117 32.16218709 17.56835655 16.62475777 16.24043964
15.26959078 20.69904022 21.06110075 19.76376439 21.07958092 16.55117173
16.10614612 15.79051915 15.63121352 30.68623095 25.10657282 30.35729138
24.75754377 29.01707404 28.4340407  32.16059618 29.03548977 26.11364564
26.21144222 26.59475552 32.07896737 31.132481   33.0293463  32.24874085
34.10835103 21.59297517 19.6576677  20.39227537 21.28505309 23.21736023
24.4235579  25.71962878 21.76905783 23.50212832 22.01457196 23.8163016
20.4903419  22.04686279 21.32122206 20.38330848 22.62637987 17.46441064
28.96950257 29.27303384 30.66423609 28.34340913 29.61271129 25.69268713
25.29260528 30.0005767  25.54720814 23.12528937 25.84869065 21.41895534
31.10502466 31.95932201 23.43759872 25.20586151 25.52290817 23.97740445
22.7551566  19.81680074 20.35742826 19.67460936 20.03931365 16.88153561
19.05450791 20.63469186 19.73125191 32.2040919  33.44524117 30.90142267
26.51946521 23.57987219 20.5695695  25.91921953 22.86655646 29.26841024
29.72909134 33.45631692 31.00700967 27.02739892 26.16141866 25.61056544
27.53982752 31.68775723 34.67584782 30.58316132 34.00784694 27.76666717
26.46762239 25.95267609 23.87820651 31.43773481 29.98650295 31.21749611
31.40718811 32.58673398 33.53957275 26.65283567 33.6244257  32.94922355
31.49186438 27.0054564  26.20710489 34.95803903 33.50598861 33.73009694
27.34555857 30.67926574 29.68930984 32.60087719 29.4534759  28.889473
28.78930104 27.01084268 29.90763811 36.4874435  32.85555202 36.39650161
34.75856959 35.2653094  34.74098289 34.99513155 30.92848188 31.96064606
30.15443279 32.27360779 33.60922273 32.91613431 30.83443461 31.44759457
26.65470916 26.20406605 28.52898099 27.92594196 23.95554276 23.67201668
26.11473697 24.000706   29.24857498 28.94140762 30.47920275 29.21969186
30.00182981 29.02005581 27.79675446 34.472563   35.66420933 35.97193622
32.2458033  32.19737296 34.73003366 34.40042321 34.43949485 35.8117604
35.86816348 35.7142301  26.80267915 28.43316736 29.77472282 28.35862054
31.75418253 30.76357903 27.5717334  28.31955391 34.46457181 31.13632611
29.35024372 28.72892119]
```

5. Residuals

Use the plot() function to produce the residual versus fitted plot of the multiple linear regression fit.

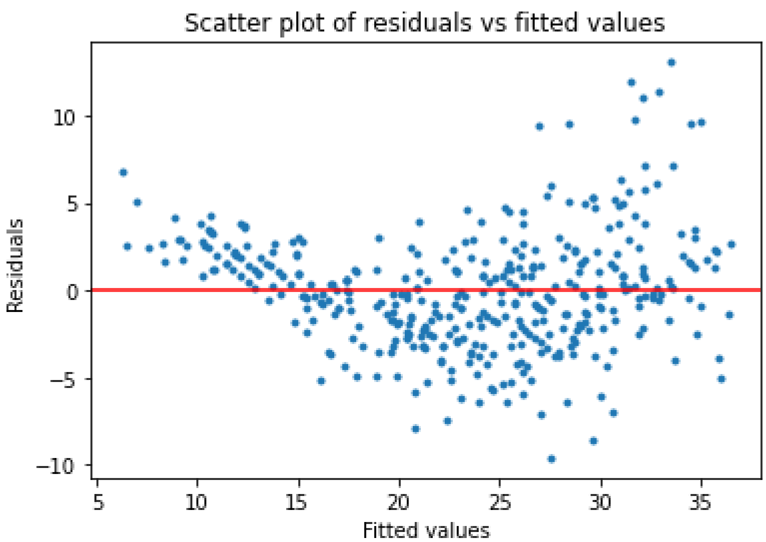
Identify the residual plot.

Does the residual plot suggest any non-linearity in the data (provide an explanation)?

In [ ]:

```
#Make a scatter plot of the residual vs the predictions of your linear regression model
#predictions = multiple_linear_model.predict(combined_arrays)
residuals = (data['mpg'] - predictions)
plt.figure()
plt.scatter(predictions, residuals, marker='.')
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title("Scatter plot of residuals vs fitted values")
plt.axhline(y = np.nanmean(residuals), color = "red")
```

Out[ ]: <matplotlib.lines.Line2D at 0x19b8639d7f0>



2/2

The residual plot shows non-linearity (banana in space), indicating a potential quadratic term in the model.

The residuals also display evidence of heteroscedacticity, increasing variance as the fitted value increased.