Homework 1: Applied Machine Learning

This assignment covers contents of the first three lectures.

The emphasis for this assignment would be on the following:

1. Data Visualization and Analysis
2. Linear Models for Regression and Classification
3. Support Vector Machines

In [1]:
```
import warnings

def fxn():
    warnings.warn("deprecated", DeprecationWarning)

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    fxn()
```

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from numpy.linalg import inv
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder
from sklearn.metrics import r2_score
from sklearn.svm import LinearSVC, SVC
```

Part 1: Data Visualization and Analysis

> "Visualization gives you answers to questions you didn't know you had." ~ Ben Schneiderman

Data visualization comes in handy when we want to understand data characteristics and read patterns in datasets with thousands of samples and features.

Note: Remember to label plot axes while plotting.

The dataset to be used for this section is car_price.csv.

In [3]:
```
# Load the dataset
car_price_df = pd.read_csv('car_price.csv')
```

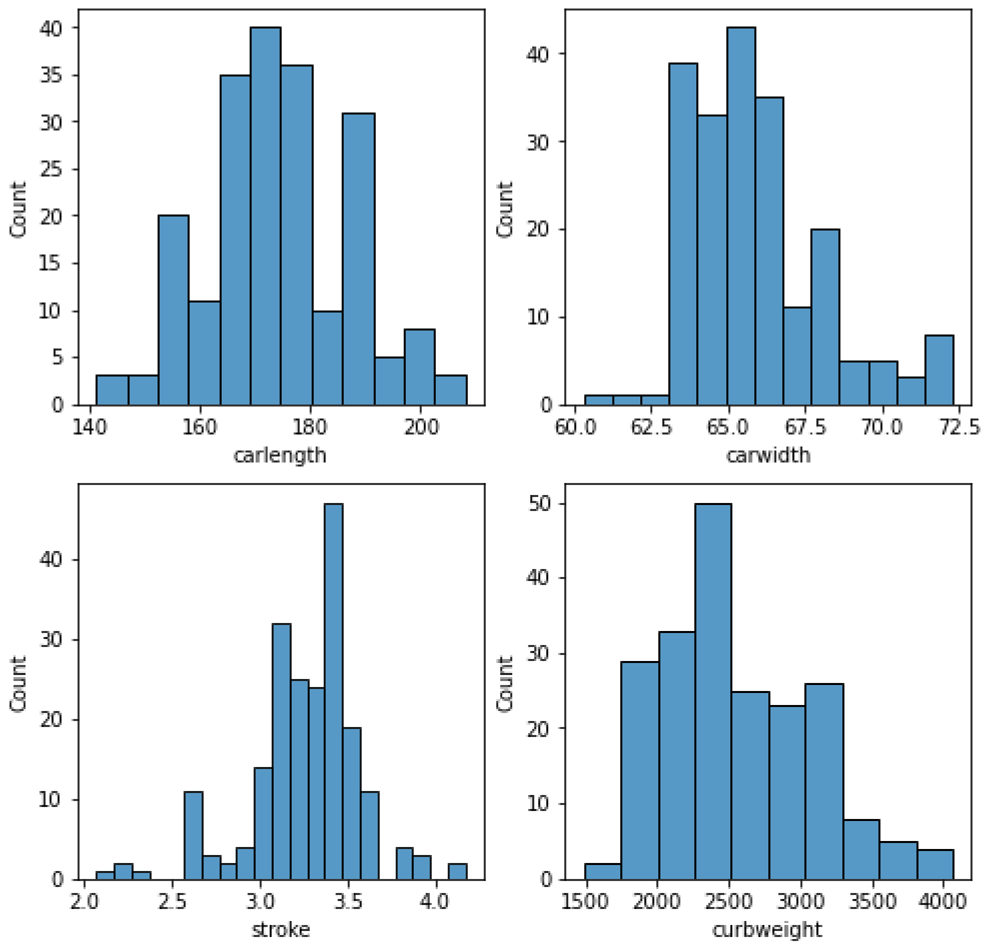
1.1 Plot the distribution of the following features as a small multiple of histograms.

1. carlength

2. carwidth
3. stroke
4. curbweight

In [4]:
```
### Code here
fig, ax=plt.subplots(2,2,figsize=(8,8),sharex=False,sharey=False)
axs=ax.flatten()
sns.histplot(x='carlength',data=car_price_df,ax=axs[0])
sns.histplot(x='carwidth',data=car_price_df,ax=axs[1])
sns.histplot(x='stroke',data=car_price_df,ax=axs[2])
sns.histplot(x='curbweight',data=car_price_df,ax=axs[3])
```

Out[4]:
```
<AxesSubplot:xlabel='curbweight', ylabel='Count'>
```



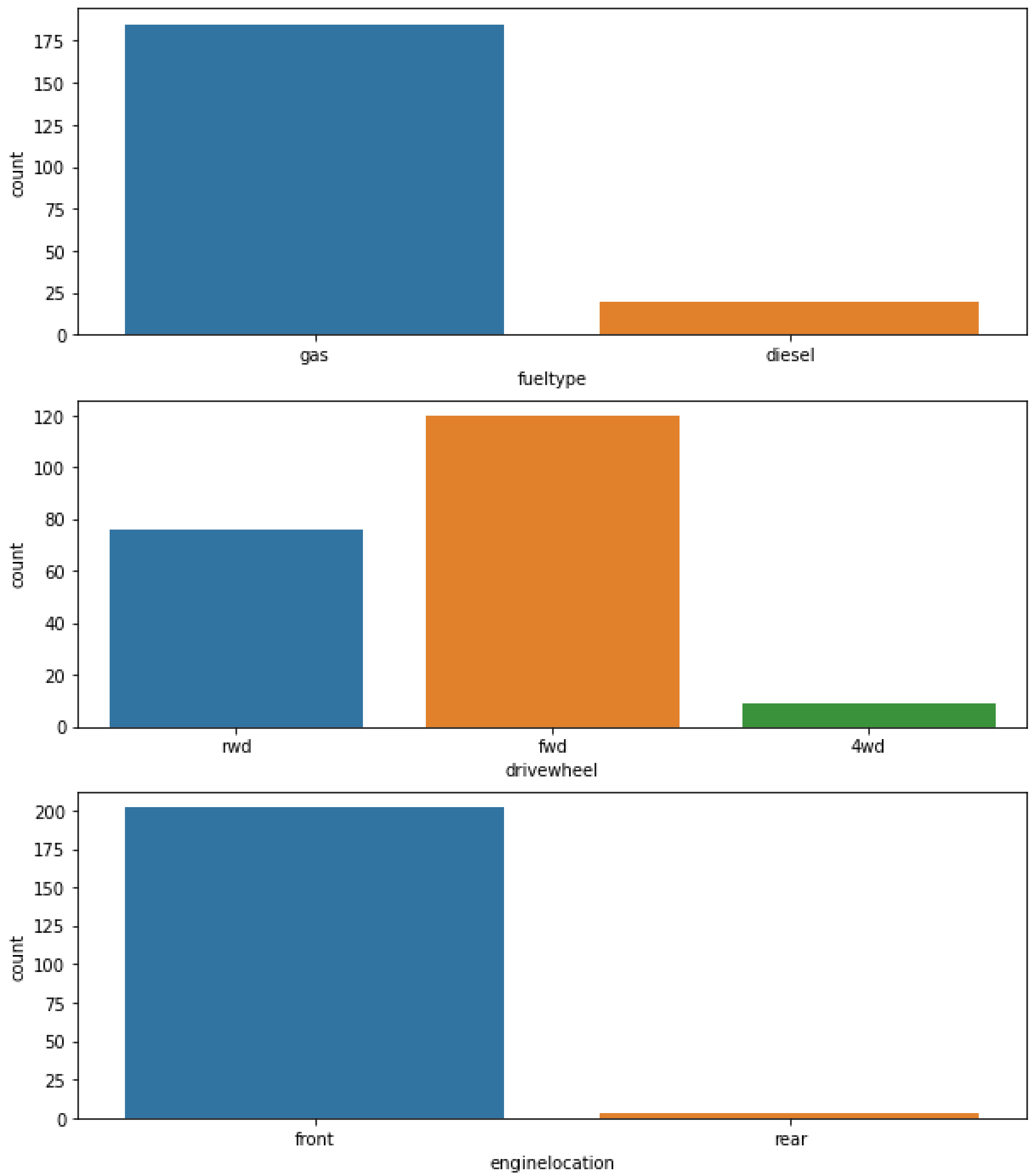
1.2 Plot a small multiple of bar charts to understand data distribution of the following categorical variables

1. fueltype
2. drivewheel
3. enginelocation

In [5]:
```
### Code here
#fig, ax=plt.subplots(3,1,figsize=(10,12),sharex=False,sharey=False)
#axs=ax.flatten()
#sns.barplot(x='fueltype',y="price",data=car_price_df,ax=axs[0])
#sns.barplot(x='drivewheel',y="price",data=car_price_df,ax=axs[1])
```

```
#sns.barplot(x='enginelocation',y="price",data=car_price_df,ax=axs[2])
#sns.barplot(x='curbweight',data=car_price_df,ax=axs[3])
fig, ax=plt.subplots(3,1,figsize=(10,12),sharex=False,sharey=False)
axs=ax.flatten()
sns.countplot(x='fueltype',data=car_price_df,ax=axs[0])
sns.countplot(x='drivewheel',data=car_price_df,ax=axs[1])
sns.countplot(x='enginelocation',data=car_price_df,ax=axs[2])
```

Out[5]: <AxesSubplot:xlabel='enginelocation', ylabel='count'>



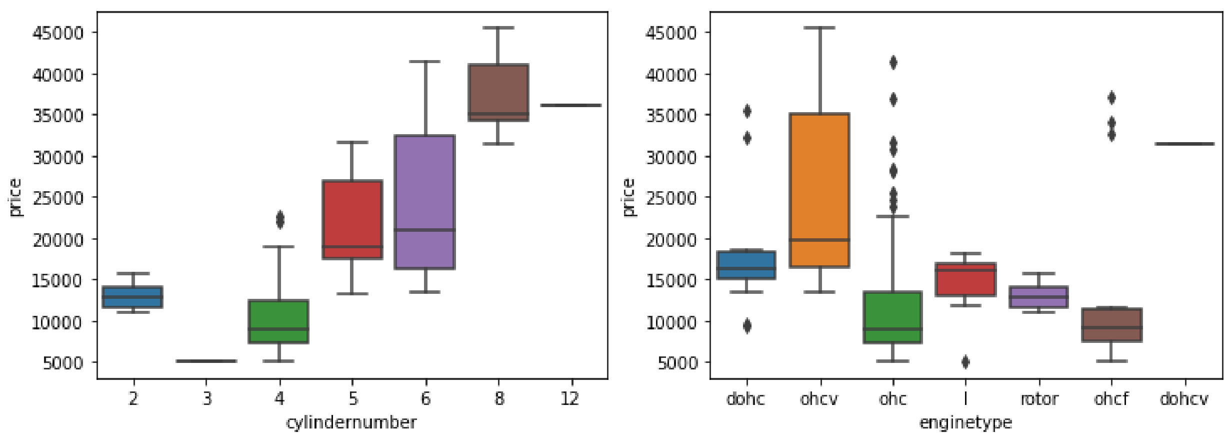
1.3 Plot relationships between the following features and the target variable *price* as a small multiple of boxplots.

1. cylindernumber
2. enginetype

Note: Make sure to order the x-axis labels in increasing order for cylindernumber.

In [6]:
```
### Code here
from sklearn.preprocessing import OrdinalEncoder
#enc=OrdinalEncoder(categories=[["four","six","five","three","twelve","two","eight"]])
#car_price_df['cylindernumber_ord']=enc.fit_transform(car_price_df['cylindernumber'].t
car_price_df=car_price_df.replace("four",4)
car_price_df=car_price_df.replace("six",6)
car_price_df=car_price_df.replace("five",5)
car_price_df=car_price_df.replace("three",3)
car_price_df=car_price_df.replace("twelve",12)
car_price_df=car_price_df.replace("two",2)
car_price_df=car_price_df.replace("eight",8)
fig, ax=plt.subplots(1,2,figsize=(12,4),sharex=False,sharey=False)
axs=ax.flatten()
sns.boxplot(x='cylindernumber',y="price",data=car_price_df,ax=axs[0])
sns.boxplot(x='enginetype',y="price",data=car_price_df,ax=axs[1])
```

Out[6]:
```
<AxesSubplot:xlabel='enginetype', ylabel='price'>
```



1.4 What do you infer from the visualization above. Comment on the skewness of the distributions (histograms), class imbalance (bar charts), and relationship between categories and price of the car (boxplots).

In [7]:
```
#### Comment here
#We can see that the cars with cylindernumber=4,5,6 or 8 has a rightly skewed price. W
#cylindernumber=2 seems to be symmetric.Cars with cylindernumber=12 has a small range
#We can also see that cars with enginetype=dohc,ohcv or ohc seem to be rightly skewed
#Similarly, cars with enginetype=l seems to be skewed to the left in terms of price. A
#or ohcf seems to be symmetric in terms of price.Cars with enginetype=dohcv has a smal
```

Part 2: Linear Models for Regression and Classification

In this section, we will be implementing three linear models **linear regression, logistic regression, and SVM**. We will see that despite some of their differences at the surface, these linear models (and many machine learning models in general) are fundamentally doing the same thing - that is, optimizing model parameters to minimize a loss function on data.

2.1 Linear Regression

In part 1, we will use two datasets - synthetic and Car Price to train and evaluate our linear regression model.

Synthetic Data

2.1.1 Generate 100 samples of synthetic data using the following equations.

$\epsilon \sim \mathcal{N}(0, 4)$

$y = 7x - 8 + \epsilon$

You may use np.random.normal() for generating ϵ.

In [8]:
```
np.random.seed(0)
X = np.linspace(0, 15, 100)
epsilon = np.random.normal(0,4)   ### Code here
y = 7*X - 8 + epsilon ### Code here
```

To apply linear regression, we need to first check if the assumptions of linear regression are not violated.

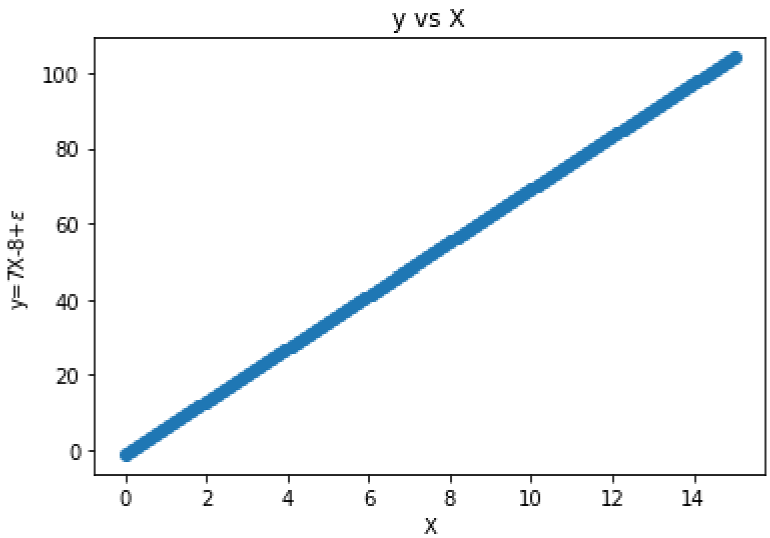
Assumptions of Linear Regression:

- Linearity: is a linear (technically affine) function of x.
- Independence: the x's are independently drawn, and not dependent on each other.
- Homoscedasticity: the ϵ's, and thus the y's, have constant variance.
- Normality: the ϵ's are drawn from a Normal distribution (i.e. Normally-distributed errors)

These properties, as well as the simplicity of this dataset, will make it a good test case to check if our linear regression model is working properly.

2.1.2 Plot y vs X in the synthetic dataset as a scatter plot. Label your axes and make sure your y-axis starts from 0. Do the features have linear relationship?

In [9]:
```
### Code here
plt.scatter(X,y)
plt.xlabel("X")
plt.ylabel("y=7X-8+$\epsilon$")
plt.title("y vs X")
plt.show()
```



In [10]:
```
#### Comment here
#Yes, it does have a linear relationship, specifically it has a strong positive one.
```

Car Price Prediction Dataset

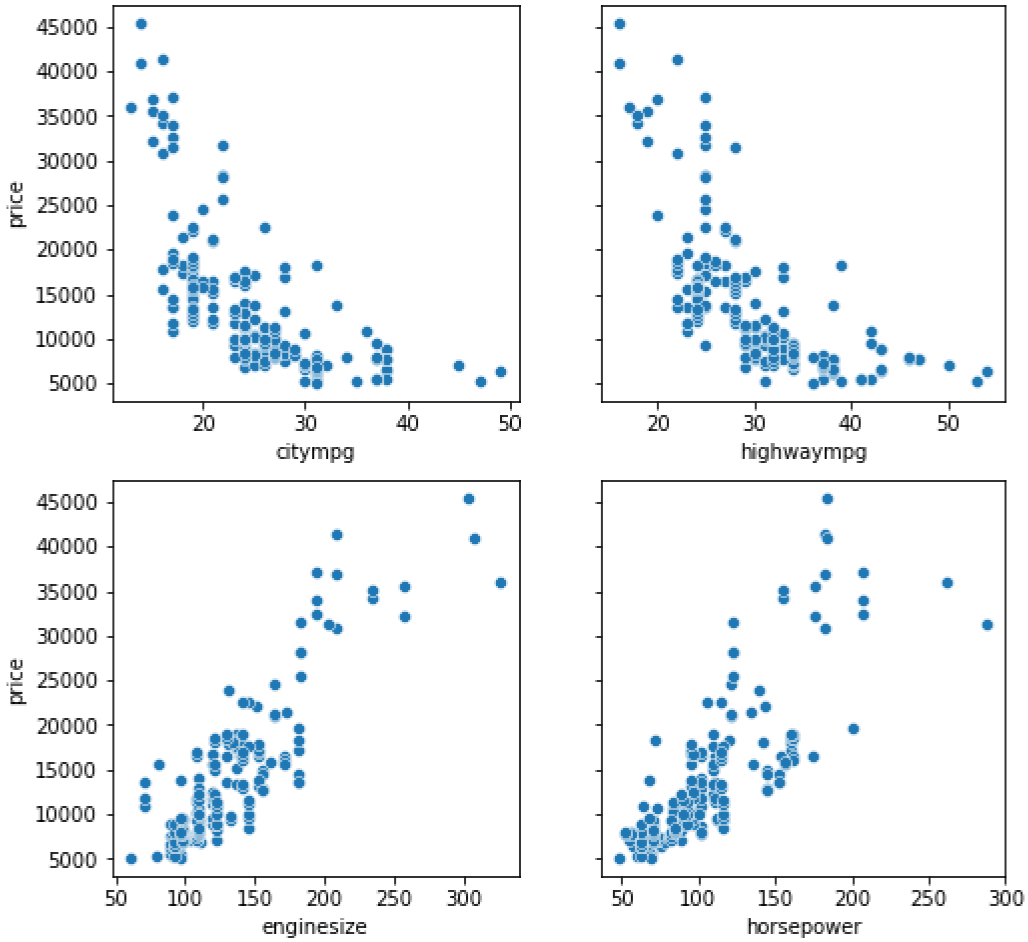
The objective of this dataset is to predict the price of a car based on its characterisitics. We will use linear regression to predict the price using its features.

In [11]:
```
# split data into features and labels
car_price_X = car_price_df.drop(columns=['price'])
car_price_y = car_price_df['price']
```

2.1.3 Plot the relationships between the label (price) and the continuous features (citympg, highwaympg, enginesize, horsepower) using a small multiple of scatter plots. Make sure to label the axes.

In [12]:
```
### Code here
fig, ax=plt.subplots(2,2,figsize=(8,8),sharex=False,sharey=True)
axs=ax.flatten()
sns.scatterplot(x=car_price_X['citympg'],y=car_price_y,ax=axs[0])
sns.scatterplot(x=car_price_X['highwaympg'],y=car_price_y,ax=axs[1])
sns.scatterplot(x=car_price_X['enginesize'],y=car_price_y,ax=axs[2])
sns.scatterplot(x=car_price_X['horsepower'],y=car_price_y,ax=axs[3])
```

Out[12]:
```
<AxesSubplot:xlabel='horsepower', ylabel='price'>
```



2.1.4 From the visualizations above, do you think linear regression is a good model for this problem? Why and/or why not? Please explain.

In [13]:
```
#### Comment here
#From the data, I do think that linear regression is a good model for the problem. Thi
#to have a linear relationship with price as we can see in the scatterplots above. The
#a negative linear relationship with price while the variables enginesize and horsepow
#linear relationship with price.
```

Data Preprocessing

Before we can fit a linear regression model, there are several pre-processing steps we should apply to the datasets:

1. Encode categorial features appropriately.
2. Remove highly collinear features by reading the correlation plot.
3. Split the dataset into training (60%), validation (20%), and test (20%) sets.
4. Standardize the columns in the feature matrices X_train, X_val, and X_test to have zero mean and unit variance. To avoid information leakage, learn the standardization parameters (mean, variance) from X_train, and apply it to X_train, X_val, and X_test.
5. Add a column of ones to the feature matrices X_train, X_val, and X_test. This is a common trick so that we can learn a coefficient for the bias term of a linear model.

The processing steps on the synthetic dataset have been provided for you below as a reference:

Note: Generate the synthetic data before running the next cell to avoid errors.

In [14]:
```
X = X.reshape((100, 1))   # Turn the X vector into a feature matrix X

# 1. No categorical features in the synthetic dataset (skip this step)

# 2. Only one feature vector

# 3. Split the dataset into training (60%), validation (20%), and test (20%) sets
X_dev, X_test, y_dev, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
X_train, X_val, y_train, y_val = train_test_split(X_dev, y_dev, test_size=0.25, randon

# 4. Standardize the columns in the feature matrices
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)   # Fit and transform scalar on X_train
X_val = scaler.transform(X_val)           # Transform X_val
X_test = scaler.transform(X_test)         # Transform X_test

# 5. Add a column of ones to the feature matrices
X_train = np.hstack([np.ones((X_train.shape[0], 1)), X_train])
X_val = np.hstack([np.ones((X_val.shape[0], 1)), X_val])
X_test = np.hstack([np.ones((X_test.shape[0], 1)), X_test])

print(X_train[:5], '\n\n', y_train[:5])
```

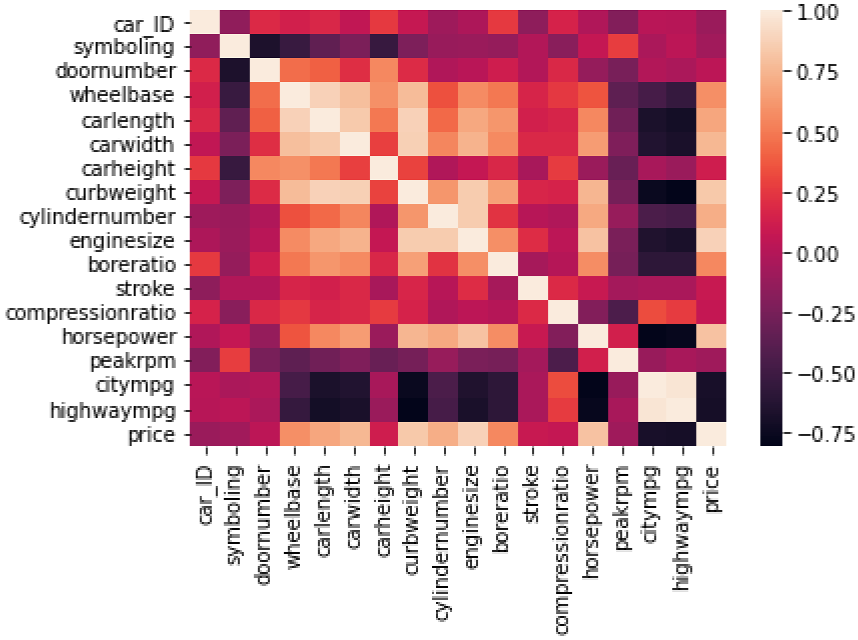
```
[[ 1.          0.53651502]
 [ 1.         -1.00836082]
 [ 1.         -0.72094206]
 [ 1.         -0.25388657]
 [ 1.          0.64429705]]

 [69.05620938 23.45014878 31.93499726 45.72287605 72.23802757]
```

2.1.5 Encode the categorical variables of the CarPrice dataset.

In [16]:
```
### Code here
ohe=OneHotEncoder()
car_price_df_trans=ohe.fit_transform(car_price_df)
#car_price_df=ohe.fit_transform(car_price_df)
```

2.1.6 Plot the correlation matrix, and check if there is high correlation between the given numerical features (Threshold >=0.9). If yes, drop one from each pair of highly correlated features from the dataframe. Why is necessary to drop those columns before proceeding further?

In [17]:
```
### Code here
corr_matrix=car_price_df.corr()
sns.heatmap(corr_matrix)
plt.show()
print(corr_matrix)
car_price_df=car_price_df.drop(['highwaympg'],axis=1)
```


car_ID
symboling
doornumber
wheelbase
carlength
carwidth
carheight
curbweight
cylindernumber
enginesize
boreratio
stroke
compressionratio
horsepower
peakrpm
citympg
highwaympg
price
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

```
                  car_ID  symboling  doornumber  wheelbase  carlength  \
car_ID          1.000000  -0.151621    0.190352   0.129729   0.170636   
symboling      -0.151621   1.000000   -0.664073  -0.531954  -0.357612   
doornumber      0.190352  -0.664073    1.000000   0.447357   0.398568   
wheelbase       0.129729  -0.531954    0.447357   1.000000   0.874587   
carlength       0.170636  -0.357612    0.398568   0.874587   1.000000   
carwidth        0.052387  -0.232919    0.207168   0.795144   0.841118   
carheight       0.255960  -0.541038    0.552208   0.589435   0.491029   
curbweight      0.071962  -0.227691    0.197379   0.776386   0.877728   
cylindernumber -0.094493  -0.113129   -0.016009   0.339507   0.430672   
enginesize     -0.033930  -0.105790    0.020742   0.569329   0.683360   
boreratio       0.260064  -0.130051    0.119258   0.488750   0.606454   
stroke         -0.160824  -0.008735   -0.011082   0.160959   0.129533   
compressionratio 0.150276 -0.178515    0.177888   0.249786   0.158414   
horsepower     -0.015006   0.070873   -0.126947   0.353294   0.552623   
peakrpm        -0.203789   0.273606   -0.247668  -0.360469  -0.287242   
citympg         0.015940  -0.035823   -0.012417  -0.470414  -0.670909   
highwaympg      0.011255   0.034606   -0.036330  -0.544082  -0.704662   
price          -0.109093  -0.079978    0.031835   0.577816   0.682920   

                carwidth  carheight  curbweight  cylindernumber  enginesize  \
car_ID          0.052387   0.255960    0.071962       -0.094493   -0.033930   
symboling      -0.232919  -0.541038   -0.227691       -0.113129   -0.105790   
doornumber      0.207168   0.552208    0.197379       -0.016009    0.020742   
wheelbase       0.795144   0.589435    0.776386        0.339507    0.569329   
carlength       0.841118   0.491029    0.877728        0.430672    0.683360   
carwidth        1.000000   0.279210    0.867032        0.545007    0.735433   
carheight       0.279210   1.000000    0.295572       -0.013995    0.067149   
curbweight      0.867032   0.295572    1.000000        0.609727    0.850594   
cylindernumber  0.545007  -0.013995    0.609727        1.000000    0.846031   
enginesize      0.735433   0.067149    0.850594        0.846031    1.000000   
boreratio       0.559150   0.171071    0.648480        0.231399    0.583774   
stroke          0.182942  -0.055307    0.168790        0.008210    0.203129   
compressionratio 0.181129  0.261214    0.151362       -0.020002    0.028971   
horsepower      0.640732  -0.108802    0.750739        0.692016    0.809769   
peakrpm        -0.220012  -0.320411   -0.266243       -0.124172   -0.244660   
citympg        -0.642704  -0.048640   -0.757414       -0.445837   -0.653658   
highwaympg     -0.677218  -0.107358   -0.797465       -0.466666   -0.677470   
price           0.759325   0.119336    0.835305        0.718305    0.874145   

                boreratio    stroke  compressionratio  horsepower   peakrpm  \
car_ID           0.260064 -0.160824          0.150276   -0.015006 -0.203789   
symboling       -0.130051 -0.008735         -0.178515    0.070873  0.273606   
doornumber       0.119258 -0.011082          0.177888   -0.126947 -0.247668   
wheelbase        0.488750  0.160959          0.249786    0.353294 -0.360469   
carlength        0.606454  0.129533          0.158414    0.552623 -0.287242   
carwidth         0.559150  0.182942          0.181129    0.640732 -0.220012   
carheight        0.171071 -0.055307          0.261214   -0.108802 -0.320411   
curbweight       0.648480  0.168790          0.151362    0.750739 -0.266243   
cylindernumber   0.231399  0.008210         -0.020002    0.692016 -0.124172   
enginesize       0.583774  0.203129          0.028971    0.809769 -0.244660   
boreratio        1.000000 -0.055909          0.005197    0.573677 -0.254976   
stroke          -0.055909  1.000000          0.186110    0.080940 -0.067964   
compressionratio 0.005197  0.186110          1.000000   -0.204326 -0.435741   
horsepower       0.573677  0.080940         -0.204326    1.000000  0.131073   
peakrpm         -0.254976 -0.067964         -0.435741    0.131073  1.000000   
citympg         -0.584532 -0.042145          0.324701   -0.801456 -0.113544   
highwaympg      -0.587012 -0.043931          0.265201   -0.770544 -0.054275   
price            0.553173  0.079443          0.067984    0.808139 -0.085267   
```

```
                  citympg  highwaympg     price
car_ID           0.015940    0.011255 -0.109093
symboling       -0.035823    0.034606 -0.079978
doornumber      -0.012417   -0.036330  0.031835
wheelbase       -0.470414   -0.544082  0.577816
carlength       -0.670909   -0.704662  0.682920
carwidth        -0.642704   -0.677218  0.759325
carheight       -0.048640   -0.107358  0.119336
curbweight      -0.757414   -0.797465  0.835305
cylindernumber  -0.445837   -0.466666  0.718305
enginesize      -0.653658   -0.677470  0.874145
boreratio       -0.584532   -0.587012  0.553173
stroke          -0.042145   -0.043931  0.079443
compressionratio 0.324701    0.265201  0.067984
horsepower      -0.801456   -0.770544  0.808139
peakrpm         -0.113544   -0.054275 -0.085267
citympg          1.000000    0.971337 -0.685751
highwaympg       0.971337    1.000000 -0.697599
price           -0.685751   -0.697599  1.000000
```

In [18]:

```
#### Comment here
#We drop the highly correlated features beacuase it is unlikely that they will provide
#Also, with these highly correlated features, you will have a more complex model which
```

2.1.7 Split the dataset into training (60%), validation (20%), and test (20%) sets. Use random_state = 0.

In [26]:

```
### Code here
car_price_X=ohe.fit_transform(car_price_X)
car_price_X_dev,car_price_X_test,car_price_y_dev,car_price_y_test= train_test_split(ca
car_price_X_train, car_price_X_val, car_price_y_train, car_price_y_val=train_test_spli
```

2.1.8 Standardize the columns in the feature matrices.

In [28]:

```
### Code here
scaler=StandardScaler(with_mean=False)
car_price_X_train=scaler.fit_transform(car_price_X_train)
car_price_X_val=scaler.transform(car_price_X_val)
car_price_X_test=scaler.transform(car_price_X_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [28], in <cell line: 4>()
      2 scaler=StandardScaler(with_mean=False)
      3 car_price_X_train=scaler.fit_transform(car_price_X_train)
----> 4 car_price_X_val=scaler.transform(car_price_X_val)
      5 car_price_X_test=scaler.transform(car_price_X_test)

File ~\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:973, in StandardSca
ler.transform(self, X, copy)
    970 check_is_fitted(self)
    972 copy = copy if copy is not None else self.copy
--> 973 X = self._validate_data(
    974     X,
    975     reset=False,
    976     accept_sparse="csr",
    977     copy=copy,
    978     estimator=self,
    979     dtype=FLOAT_DTYPES,
    980     force_all_finite="allow-nan",
    981 )
    983 if sparse.issparse(X):
    984     if self.with_mean:

File ~\anaconda3\lib\site-packages\sklearn\base.py:585, in BaseEstimator._validate_da
ta(self, X, y, reset, validate_separately, **check_params)
    582     out = X, y
    584 if not no_val_X and check_params.get("ensure_2d", True):
--> 585     self._check_n_features(X, reset=reset)
    587 return out

File ~\anaconda3\lib\site-packages\sklearn\base.py:400, in BaseEstimator._check_n_fea
tures(self, X, reset)
    397     return
    399 if n_features != self.n_features_in_:
--> 400     raise ValueError(
    401         f"X has {n_features} features, but {self.__class__.__name__} "
    402         f"is expecting {self.n_features_in_} features as input."
    403     )

ValueError: X has 1080 features, but StandardScaler is expecting 807 features as inpu
t.
```

2.1.9 Add a column of ones to the feature matrices for the bias term.

In [29]:

```
### Code here
print(np.ones((car_price_X_train.shape[0],1)).shape)
print(car_price_X_train)
car_price_X_train=np.hstack([np.ones((car_price_X_train.shape[0],1)),car_price_X_trair
print(car_price_X_train)
car_price_X_val=np.hstack([np.ones((car_price_X_val.shape[0],1)),car_price_X_val])
car_price_X_test=np.hstack([np.ones((car_price_X_test.shape[0],1)),car_price_X_test])
```

```
(123, 1)
  (0, 27)	11.13589676322978
  (0, 126)	2.2359940811043395
  (0, 155)	11.13589676322978
  (0, 228)	3.055959569249713
  (0, 229)	2.523375565076321
  (0, 231)	2.0500000000000003
  (0, 236)	2.0032467016022037
  (0, 239)	2.035910485961825
  (0, 241)	7.906739462358669
  (0, 254)	4.316453799013118
  (0, 306)	11.13589676322978
  (0, 362)	11.13589676322978
  (0, 390)	11.13589676322978
  (0, 462)	11.13589676322978
  (0, 540)	2.2793416487328857
  (0, 545)	2.355820440662382
  (0, 561)	4.05518991138409
  (0, 586)	2.1213203435596424
  (0, 602)	3.5042877261338674
  (0, 655)	5.063829862994603
  (0, 672)	11.13589676322978
  (0, 708)	11.13589676322978
  (0, 750)	2.710013792821275
  (0, 767)	4.642335839992843
  (0, 794)	5.063829862994603
  :	:
  (122, 120)	11.13589676322978
  (122, 124)	3.2526481948898622
  (122, 222)	11.13589676322978
  (122, 228)	3.055959569249713
  (122, 229)	2.523375565076321
  (122, 232)	2.05
  (122, 236)	2.0032467016022037
  (122, 240)	2.0971325938717054
  (122, 241)	7.906739462358669
  (122, 280)	6.482669203345178
  (122, 336)	4.642335839992843
  (122, 374)	6.482669203345178
  (122, 411)	4.316453799013118
  (122, 508)	11.13589676322978
  (122, 540)	2.2793416487328857
  (122, 545)	2.355820440662382
  (122, 570)	5.6376957567235
  (122, 589)	2.0242971574306474
  (122, 624)	5.063829862994603
  (122, 638)	3.8400018274849455
  (122, 677)	4.316453799013118
  (122, 713)	5.6376957567235
  (122, 749)	4.316453799013118
  (122, 765)	5.063829862994603
  (122, 791)	3.8400018274849455
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [29], in <cell line: 4>()
      2 print(np.ones((car_price_X_train.shape[0],1)).shape)
      3 print(car_price_X_train)
----> 4 car_price_X_train=np.hstack([np.ones((car_price_X_train.shape[0],1)),car_pric
e_X_train])
      5 print(car_price_X_train)
      6 car_price_X_val=np.hstack([np.ones((car_price_X_val.shape[0],1)),car_price_X_
val])

File <__array_function__ internals>:5, in hstack(*args, **kwargs)

File ~\anaconda3\lib\site-packages\numpy\core\shape_base.py:345, in hstack(tup)
    343     return _nx.concatenate(arrs, 0)
    344 else:
--> 345     return _nx.concatenate(arrs, 1)

File <__array_function__ internals>:5, in concatenate(*args, **kwargs)

ValueError: all the input arrays must have same number of dimensions, but the array a
t index 0 has 2 dimension(s) and the array at index 1 has 1 dimension(s)
```

At the end of this pre-processing, you should have the following vectors and matrices:

- Syntheic dataset: X_train, X_val, X_test, y_train, y_val, y_test
- Car Price Prediction dataset: car_price_X_train, car_price_X_val, car_price_X_test, car_price_y_train, car_price_y_val, car_price_y_test

Implement Linear Regression

Now, we can implement our linear regression model! Specifically, we will be implementing ridge regression, which is linear regression with L2 regularization. Given an (m x n) feature matrix X, an (m x 1) label vector y, and an (n x 1) weight vector w, the hypothesis function for linear regression is:

$$y = Xw$$

Note that we can omit the bias term here because we have included a column of ones in our X matrix, so the bias term is learned implicitly as a part of w. This will make our implementation easier.

Our objective in linear regression is to learn the weights w which best fit the data. This notion can be formalized as finding the optimal w which minimizes the following loss function:

$$\min_{w} \|Xw - y\|_2^2 + \alpha \|w\|_2^2$$

This is the ridge regression loss function. The $\|Xw - y\|_2^2$ term penalizes predictions Xw which are not close to the label y. And the $\alpha\|w\|_2^2$ penalizes large weight values, to favor a simpler, more generalizable model. The α hyperparameter, known as the regularization parameter, is used to tune the complexity of the model - a higher α results in smaller weights and lower complexity, and vice versa. Setting $\alpha = 0$ gives us vanilla linear regression.

Conveniently, ridge regression has a closed-form solution which gives us the optimal w without having to do iterative methods such as gradient descent. The closed-form solution, known as the Normal Equations, is given by:

$$w = (X^T X + \alpha I)^{-1} X^T y$$

2.1.10 Implement a `LinearRegression` class with two methods: `train` and `predict`.

Note: You may NOT use sklearn for this implementation. You may, however, use `np.linalg.solve` to find the closed-form solution. It is highly recommended that you vectorize your code.

In [30]:
```
class LinearRegression():
    '''
    Linear regression model with L2-regularization (i.e. ridge regression).

    Attributes
    ----------
    alpha: regularization parameter
    w: (n x 1) weight vector
    '''

    def __init__(self, alpha=0):
        self.alpha = alpha
        self.w = None

    def train(self, X, y):
        '''Trains model using ridge regression closed-form solution
        (sets w to its optimal value).

        Parameters
        ----------
        X : (m x n) feature matrix
        y: (m x 1) label vector

        Returns
        -------
        None
        '''
        ### Your code here
       # w=np.dot(np.linalg.inv(np.dot(np.transpose(X),X) + self.alpha*np.identity(np.
        inner=np.transpose(X) @ X
        inner=inner + self.alpha*np.identity(X.shape[1])
        inner=np.linalg.pinv(inner)
        inner=inner @ np.transpose(X) @ y
        self.w=inner

        #print(self.w)
        pass

    def predict(self, X):
        '''Predicts on X using trained model.
```

```
        Parameters
        ----------
        X : (m x n) feature matrix

        Returns
        -------
        y_pred: (m x 1) prediction vector
        '''
        ### Your code here
       # X=np.append(X,np.ones((X.shape[0],1)),axis=1)
        y_pred=X @ self.w
       # y_pred=np.linalg.solve(np.dot(np.linalg.inv(np.dot(np.transpose(X),X) + self.
        #y_pred=np.dot(X,self.w)
        return y_pred
        pass
```

Train, Evaluate, and Interpret LR Model

2.1.11 Using your `LinearRegression` implementation above, train a vanilla linear regression model ($\alpha = 0$) on (X_train, y_train) from the synthetic dataset. Use this trained model to predict on X_test. Report the first 3 and last 3 predictions on X_test, along with the actual labels in y_test.

In [31]:

```
### Code here
model=LinearRegression(alpha=0)
model.train(X_train,y_train)
y_pred=model.predict(X_test)
print("First 3 Predictions: {}".format(y_pred[:3]))
print("First 3 Actual Values: {}".format(y_test[:3]))
print("Last 3 Predictions: {}".format(y_pred[-3:]))
print("First 3 Actual Values: {}".format(y_test[-3:]))
```

```
First 3 Predictions: [26.63196696 90.2683306   1.17742151]
First 3 Actual Values: [26.63196696 90.2683306   1.17742151]
Last 3 Predictions: [24.51075484 34.05620938  7.54105787]
First 3 Actual Values: [24.51075484 34.05620938  7.54105787]
```

2.1.12 Plot a scatter plot of y_test vs. X_test (just the non-ones column). Then, using the weights from the trained model above, plot the best-fit line for this data on the same figure.

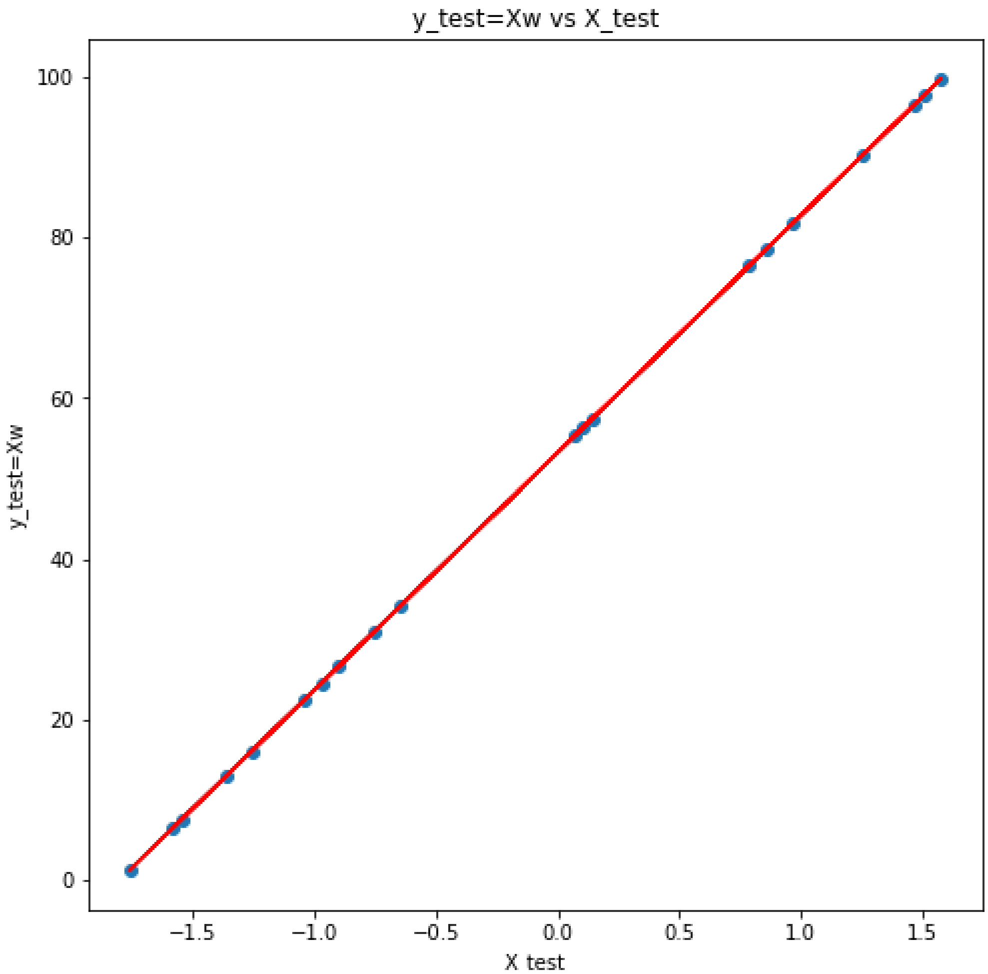
> If your line goes through the data points, you have likely implemented the linear regression correctly!

In [32]:

```
### Code here
fig, ax=plt.subplots(1,1,figsize=(8,8))
#axs=ax.flatten()
#print(X_test)
#sns.scatterplot(x=car_price_X_test,y=car_price_y_test,ax=ax)
plt.scatter(X_test[:,[1]],y_test)
plt.plot(X_test[:,[1]],X_test @ model.w,color='red')
plt.xlabel('X_test')
plt.ylabel('y_test=Xw')
```

```
plt.title('y_test=Xw vs X_test')
#sns.regplot(X_test,y_test)
```

Out[32]:
```
Text(0.5, 1.0, 'y_test=Xw vs X_test')
```



2.1.13 Train a linear regression model ($\alpha = 0$) on the car price training data. Make predictions and report the R^2 score on the training, validation, and test sets. Report the first 3 and last 3 predictions on the test set, along with the actual labels.

In [33]:
```
### Code here
model=LinearRegression(alpha=0)
model.train(car_price_X_train,car_price_y_train)
car_pred_y_test=model.predict(car_price_X_test)
car_pred_y_val=model.predict(car_price_X_val)
car_pred_y_train=model.predict(car_price_X_train)
R_sq_train = r2_score(car_price_y_train, car_pred_y_train)
R_sq_val = r2_score(car_price_y_val, car_pred_y_val)
R_sq_test = r2_score(car_price_y_test, car_pred_y_test)
print("R-squared for train set: {}".format(R_sq_train))
print("R-squared for validation set: {}".format(R_sq_val))
print("R-squared for test set: {}".format(R_sq_test))
print("Testing First 3 Predictions: {}".format(y_test[:3]))
print("Testing First 3 Actual Values: \n{}".format(car_price_y_test[:3]))
print("Testing Last 3 Predictions: {}".format(y_test[-3:]))
print("Testing Last 3 Actual Values: \n{}".format(car_price_y_test[-3:]))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [33], in <cell line: 4>()
      2 model=LinearRegression(alpha=0)
      3 model.train(car_price_X_train,car_price_y_train)
----> 4 car_pred_y_test=model.predict(car_price_X_test)
      5 car_pred_y_val=model.predict(car_price_X_val)
      6 car_pred_y_train=model.predict(car_price_X_train)

Input In [30], in LinearRegression.predict(self, X)
     40 '''Predicts on X using trained model.
     41 
     42 Parameters
   (...)
     48 y_pred: (m x 1) prediction vector
     49 '''
     50 ### Your code here
     51 # X=np.append(X,np.ones((X.shape[0],1)),axis=1)
---> 52 y_pred=X @ self.w
     53 # y_pred=np.linalg.solve(np.dot(np.linalg.inv(np.dot(np.transpose(X),X) + sel
f.alpha*np.identity(np.transpose(X).shape[0])),np.dot(np.transpose(X),y)),w)
     54 #y_pred=np.dot(X,self.w)
     55 return y_pred

File ~\anaconda3\lib\site-packages\scipy\sparse\base.py:560, in spmatrix.__matmul__(s
elf, other)
    557 if isscalarlike(other):
    558     raise ValueError("Scalar operands are not allowed, "
    559                      "use '*' instead")
--> 560 return self.__mul__(other)

File ~\anaconda3\lib\site-packages\scipy\sparse\base.py:498, in spmatrix.__mul__(sel
f, other)
    495 if other.ndim == 1 or other.ndim == 2 and other.shape[1] == 1:
    496     # dense row or column vector
    497     if other.shape != (N,) and other.shape != (N, 1):
--> 498         raise ValueError('dimension mismatch')
    500     result = self._mul_vector(np.ravel(other))
    502     if isinstance(other, np.matrix):

ValueError: dimension mismatch
```

2.1.14 As a baseline model, use the mean of the training labels (car_price_y_train) as the prediction for all instances. Report the R^2 on the training, validation, and test sets using this baseline.

> This is a common baseline used in regression problems and tells you if your model is any good. Your linear regression R^2 should be much higher than these baseline R^2.

In [34]:
```
### Code here
#corr_matrix_mu_tr = np.corrcoef(car_price_y_train, np.full(car_price_y_train.shape,np
#corr_mu_tr = corr_matrix_mu_tr[0,1]
R_sq_mu_tr = r2_score(np.full(car_pred_y_train.shape,np.mean(car_price_y_train)), car_
print("R-squared for training set: {}".format(R_sq_mu_tr))
R_sq_mu_val =r2_score(np.full(car_pred_y_val.shape,np.mean(car_price_y_train)), car_pr
print("R-squared for validation set: {}".format(R_sq_mu_val))
```

```
R_sq_mu_test=r2_score(np.full(car_pred_y_test.shape,np.mean(car_price_y_train)), car_p
print("R-squared for testing set: {}".format(R_sq_mu_test))
```

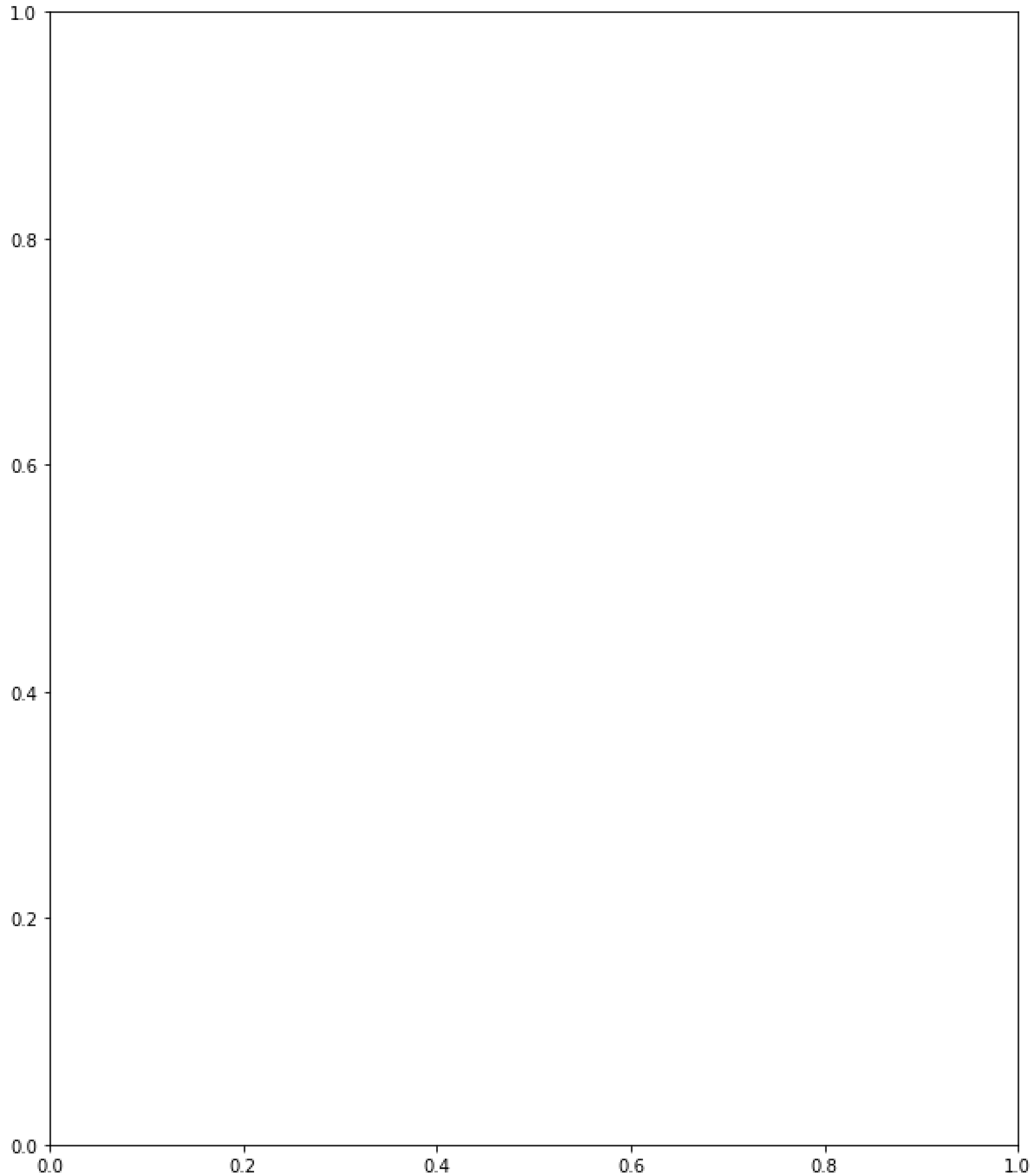
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [34], in <cell line: 4>()
      1 ### Code here
      2 #corr_matrix_mu_tr = np.corrcoef(car_price_y_train, np.full(car_price_y_trai
n.shape,np.mean(car_price_y_train)))
      3 #corr_mu_tr = corr_matrix_mu_tr[0,1]
----> 4 R_sq_mu_tr = r2_score(np.full(car_pred_y_train.shape,np.mean(car_price_y_trai
n)), car_pred_y_train)
      5 print("R-squared for training set: {}".format(R_sq_mu_tr))
      6 R_sq_mu_val =r2_score(np.full(car_pred_y_val.shape,np.mean(car_price_y_trai
n)), car_pred_y_val)

NameError: name 'car_pred_y_train' is not defined
```

2.1.15 Interpret your model trained on the car price dataset using a bar chart of the model weights. Make sure to label the bars (x-axis) and don't forget the bias term!

In [35]:

```
### Code here
fig, ax=plt.subplots(1,1,figsize=(10,12),sharex=False,sharey=False)
plt.plot(car_price_X_train[:,[1]],car_pred_y_train)
plt.xlabel('Car Price X Train')
plt.ylabel('Predicted Y train Values')
feat = ['car_ID',  'symboling', 'CarName', 'fueltype', 'aspiration','doornumber', 'car
        'enginelocation',  'wheelbase','cylindernumber',  'enginesize',  'fuelsystem',
        'compressionratio',  'horsepower', 'peakrpm',  'citympg',  'highwaympg' ,'bias
#feat=feat.reshape(-1,1)
modelcar=LinearRegression(alpha=0)
modelcar.train(car_price_X_train,car_price_y_train)
print(car_price_X_train)
print(car_price_X)
print(car_price_X_train)
print(modelcar.w)
#plt.plot(,modelcar.w)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [35], in <cell line: 3>()
      1 ### Code here
      2 fig, ax=plt.subplots(1,1,figsize=(10,12),sharex=False,sharey=False)
----> 3 plt.plot(car_price_X_train[:,[1]],car_pred_y_train)
      4 plt.xlabel('Car Price X Train')
      5 plt.ylabel('Predicted Y train Values')

NameError: name 'car_pred_y_train' is not defined
```



2.1.16 According to your model, which features are the greatest contributors to the car price?

In [ ]:
```
#### Comment here
```

Hyperparameter Tuning (α)

Now, let's do ridge regression and tune the α regularization parameter on the car price dataset.

2.1.17 Sweep out values for α using alphas = np.logspace(-5, 1, 20). Perform a grid search over these α values, recording the training and validation R^2 for each α. A simple grid search is fine, no need for k-fold cross validation. Plot the training and validation R^2 as a

function of α on a single figure. Make sure to label the axes and the training and validation R^2 curves. Use a log scale for the x-axis.

In [36]:

```
### Code here
alphas=np.logspace(-5,1,20)
model=LinearRegression().train(car_price_X_train,car_price_y_train)
rvals=[]
rtrs=[]
for a in alphas:
    rr=LinearRegression(a)
    rr.train(car_price_X_train,car_price_y_train)
    rr_train=rr.predict(car_price_X_train)
    R_sq_train=r2_score(car_price_X_train[:,[1]],rr_train)
    rtrs.append(R_sq_train)
    rr_vals=rr.predict(car_price_X_val)
    R_sq_val=r2_score(car_price_X_val[:,[1]],rr_vals)
    rvals.append(R_sq_val)
plt.plot(alphas,rtrs,label="training")
plt.plot(alphas,rvals,label="validation")
plt.xlabel(r'$ \alpha $')
plt.ylabel("R-squared")
plt.legend()
plt.show()
#sns.regplot
#model=Ridge()
#archCV(estimator=model,param_grid=dict(alpha=alphas))
#grid.fit(car_price_X_train,car_price_y_train)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [36], in <cell line: 6>()
      8 rr.train(car_price_X_train,car_price_y_train)
      9 rr_train=rr.predict(car_price_X_train)
---> 10 R_sq_train=r2_score(car_price_X_train[:,[1]],rr_train)
     11 rtrs.append(R_sq_train)
     12 rr_vals=rr.predict(car_price_X_val)

File ~\anaconda3\lib\site-packages\sklearn\metrics\_regression.py:789, in r2_score(y_
true, y_pred, sample_weight, multioutput)
    702 def r2_score(y_true, y_pred, *, sample_weight=None, multioutput="uniform_aver
age"):
    703     """:math:`R^2` (coefficient of determination) regression score function.
    704 
    705     Best possible score is 1.0 and it can be negative (because the
   (...)
    787     -3.0
    788     """
--> 789     y_type, y_true, y_pred, multioutput = _check_reg_targets(
    790         y_true, y_pred, multioutput
    791     )
    792     check_consistent_length(y_true, y_pred, sample_weight)
    794     if _num_samples(y_pred) < 2:

File ~\anaconda3\lib\site-packages\sklearn\metrics\_regression.py:95, in _check_reg_t
argets(y_true, y_pred, multioutput, dtype)
     61 """Check that y_true and y_pred belong to the same regression task.
     62 
     63 Parameters
   (...)
     92     the dtype argument passed to check_array.
     93 """
     94 check_consistent_length(y_true, y_pred)
---> 95 y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
     96 y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
     98 if y_true.ndim == 1:

File ~\anaconda3\lib\site-packages\sklearn\utils\validation.py:720, in check_array(ar
ray, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure
_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    718 if sp.issparse(array):
    719     _ensure_no_complex_data(array)
--> 720     array = _ensure_sparse_format(
    721         array,
    722         accept_sparse=accept_sparse,
    723         dtype=dtype,
    724         copy=copy,
    725         force_all_finite=force_all_finite,
    726         accept_large_sparse=accept_large_sparse,
    727     )
    728 else:
    729     # If np.array(..) gives ComplexWarning, then we convert the warning
    730     # to an error. This is needed because specifying a non complex
    731     # dtype to the function converts complex to real dtype,
    732     # thereby passing the test made in the lines following the scope
    733     # of warnings context manager.
    734     with warnings.catch_warnings():

File ~\anaconda3\lib\site-packages\sklearn\utils\validation.py:440, in _ensure_sparse
```

```
_format(spmatrix, accept_sparse, dtype, copy, force_all_finite, accept_large_sparse)
    437 _check_large_sparse(spmatrix, accept_large_sparse)
    439 if accept_sparse is False:
--> 440     raise TypeError(
    441         "A sparse matrix was passed, but dense "
    442         "data is required. Use X.toarray() to "
    443         "convert to a dense numpy array."
    444     )
    445 elif isinstance(accept_sparse, (list, tuple)):
    446     if len(accept_sparse) == 0:

TypeError: A sparse matrix was passed, but dense data is required. Use X.toarray() to
convert to a dense numpy array.
```

2.1.18 Explain your plot above. How do training and validation R^2 behave with decreasing model complexity (increasing α)?

In [37]:
```
#### Comment here
#As model complexity decreases and alpha increases, the training and validation R-squa
```

2.2 Logistic Regression

In this part, we will be using a heart disease dataset for classification.

The classification goal is to predict whether the patient has 10-year risk of future coronary heart disease (CHD).The dataset provides information about patients, over 4,000 records and 15 attributes.

Variables:

Each attribute is a potential risk factor. There are both demographic, behavioral and medical risk factors.

Demographic:

- Sex: male or female(Nominal)
- Age: Age of the patient;(Continuous - Although the recorded ages have been truncated to whole numbers, the concept of age is continuous)

Behavioral:

- Current Smoker: whether or not the patient is a current smoker (Nominal)
- Cigs Per Day: the number of cigarettes that the person smoked on average in one day.(can be considered continuous as one can have any number of cigarettes, even half a cigarette.)

Medical(history):

- BP Meds: whether or not the patient was on blood pressure medication (Nominal)
- Prevalent Stroke: whether or not the patient had previously had a stroke (Nominal)
- Prevalent Hyp: whether or not the patient was hypertensive (Nominal)
- Diabetes: whether or not the patient had diabetes (Nominal)

Medical(current):

- Tot Chol: total cholesterol level (Continuous)
- Sys BP: systolic blood pressure (Continuous)
- Dia BP: diastolic blood pressure (Continuous)
- BMI: Body Mass Index (Continuous)
- Heart Rate: heart rate (Continuous - In medical research, variables such as heart rate though in fact discrete, yet are considered continuous because of large number of possible values.)
- Glucose: glucose level (Continuous)

Predict variable (desired target):

- 10 year risk of coronary heart disease CHD (binary: “1”, means “Yes”, “0” means “No”)

In [38]:

```
heart_disease_df = pd.read_csv('heart_disease.csv')
heart_disease_df.head()
```

Out[38]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabet
0	1	39	4.0	0	0.0	0.0	0	0	
1	0	46	2.0	0	0.0	0.0	0	0	
2	1	48	1.0	1	20.0	0.0	0	0	
3	0	61	3.0	1	30.0	0.0	0	1	
4	0	46	3.0	1	23.0	0.0	0	0	

Missing Value Analysis

2.2.1 Are there any missing values in the dataset? If so, what can be done about it? (Think if removing is an option?)

In [39]:

```
### Code here
heart_disease_df.isna().sum()
```

Out[39]:
```
male                0
age                 0
education         105
currentSmoker       0
cigsPerDay         29
BPMeds             53
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            50
sysBP               0
diaBP               0
BMI                19
heartRate           1
glucose           388
TenYearCHD          0
dtype: int64
```

In [40]:
```
heart_disease_df=heart_disease_df.dropna()
#### Comment here
#You can either drop the rows with missing values (if the dataset is large) or impute
#its column mean (if the dataset is small).
```

2.2.2 Do you think that the distribution of labels is balanced? Why/why not? Hint: Find the probability of the different categories.

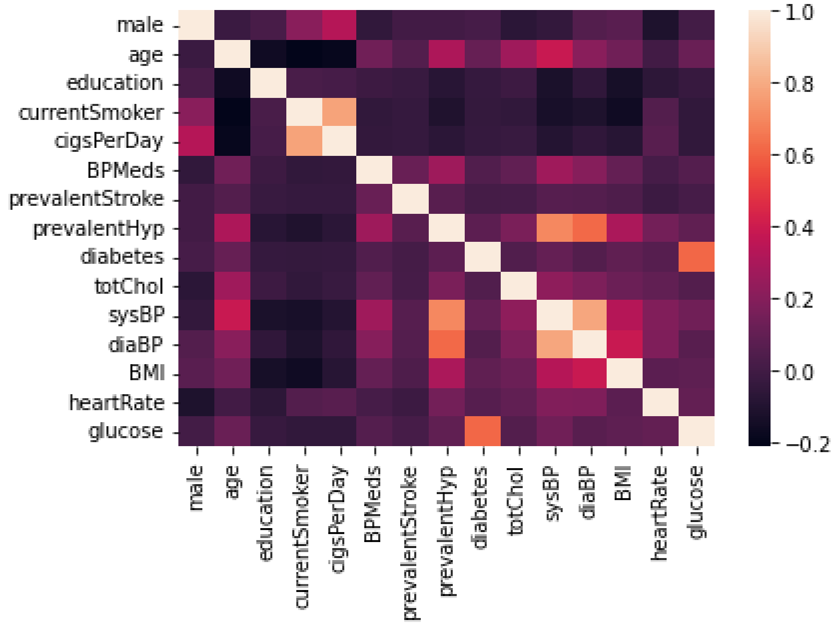
In [41]:
```
### Code here
heart_disease_X=heart_disease_df.drop(columns=['TenYearCHD'])
heart_disease_y=heart_disease_df['TenYearCHD']
print(heart_disease_y.value_counts(normalize=True))
```

```
0    0.847648
1    0.152352
Name: TenYearCHD, dtype: float64
```

In [42]:
```
#### Comment here
#No, it is not because of the probabilities of the labels are not .5
```

2.2.3 Plot the correlation matrix (first separate features and Y variable), and check if there is high correlation between the given numerical features (Threshold >=0.9). If yes, drop those highly correlated features from the dataframe.

In [43]:
```
### Code here
heart_disease_X=heart_disease_df.drop(columns=['TenYearCHD'])
heart_disease_y=heart_disease_df['TenYearCHD']
corr_matrix=heart_disease_X.corr()
sns.heatmap(corr_matrix)
plt.show()
print(corr_matrix)
#car_price_df=car_price_df.drop(['highwaympg'],axis=1)
```


male
age
education
currentSmoker
cigsPerDay
BPMeds
prevalentStroke
prevalentHyp
diabetes
totChol
sysBP
diaBP
BMI
heartRate
glucose
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

```
                    male       age  education  currentSmoker  cigsPerDay  \
male            1.000000 -0.024387   0.017677       0.206778    0.331243   
age            -0.024387  1.000000  -0.158961      -0.210862   -0.189099   
education       0.017677 -0.158961   1.000000       0.025253    0.013527   
currentSmoker   0.206778 -0.210862   0.025253       1.000000    0.773819   
cigsPerDay      0.331243 -0.189099   0.013527       0.773819    1.000000   
BPMeds         -0.052128  0.134670  -0.013647      -0.051936   -0.046479   
prevalentStroke -0.002308  0.050864  -0.030353      -0.038159   -0.036283   
prevalentHyp    0.000806  0.306693  -0.079100      -0.107561   -0.069890   
diabetes        0.013833  0.109027  -0.039547      -0.041859   -0.036934   
totChol        -0.070229  0.267764  -0.012956      -0.051119   -0.030222   
sysBP          -0.045484  0.388551  -0.124511      -0.134371   -0.094764   
diaBP           0.051575  0.208880  -0.058502      -0.115748   -0.056650   
BMI             0.072867  0.137172  -0.137280      -0.159574   -0.086888   
heartRate      -0.114923 -0.002685  -0.064254       0.050452    0.063549   
glucose         0.003048  0.118245  -0.031874      -0.053346   -0.053803   

                  BPMeds  prevalentStroke  prevalentHyp  diabetes   totChol  \
male           -0.052128        -0.002308      0.000806  0.013833 -0.070229   
age             0.134670         0.050864      0.306693  0.109027  0.267764   
education      -0.013647        -0.030353     -0.079100 -0.039547 -0.012956   
currentSmoker  -0.051936        -0.038159     -0.107561 -0.041859 -0.051119   
cigsPerDay     -0.046479        -0.036283     -0.069890 -0.036934 -0.030222   
BPMeds          1.000000         0.113119      0.263047  0.049051  0.094011   
prevalentStroke  0.113119         1.000000      0.066098  0.009619  0.012697   
prevalentHyp    0.263047         0.066098      1.000000  0.080623  0.167074   
diabetes        0.049051         0.009619      0.080623  1.000000  0.048371   
totChol         0.094011         0.012697      0.167074  0.048371  1.000000   
sysBP           0.271291         0.061080      0.697790  0.102574  0.220130   
diaBP           0.199750         0.055878      0.617634  0.050767  0.174986   
BMI             0.105603         0.036478      0.302917  0.088970  0.120799   
heartRate       0.012894        -0.017020      0.147333  0.060996  0.093057   
glucose         0.054210         0.016051      0.087129  0.614817  0.049749   

                   sysBP     diaBP       BMI  heartRate   glucose  
male           -0.045484  0.051575  0.072867  -0.114923  0.003048  
age             0.388551  0.208880  0.137172  -0.002685  0.118245  
education      -0.124511 -0.058502 -0.137280  -0.064254 -0.031874  
currentSmoker  -0.134371 -0.115748 -0.159574   0.050452 -0.053346  
cigsPerDay     -0.094764 -0.056650 -0.086888   0.063549 -0.053803  
BPMeds          0.271291  0.199750  0.105603   0.012894  0.054210  
prevalentStroke  0.061080  0.055878  0.036478  -0.017020  0.016051  
prevalentHyp    0.697790  0.617634  0.302917   0.147333  0.087129  
diabetes        0.102574  0.050767  0.088970   0.060996  0.614817  
totChol         0.220130  0.174986  0.120799   0.093057  0.049749  
sysBP           1.000000  0.786727  0.331004   0.184901  0.134702  
diaBP           0.786727  1.000000  0.385611   0.179008  0.063704  
BMI             0.331004  0.385611  1.000000   0.074401  0.083671  
heartRate       0.184901  0.179008  0.074401   1.000000  0.097026  
glucose         0.134702  0.063704  0.083671   0.097026  1.000000  
```

In [44]:
```
#### Comment here
#No, highly correlated numerical features
```

2.2.4 Apply the following pre-processing steps:

1. Convert the label from a Pandas series to a Numpy (m x 1) vector. If you don't do this, it may cause problems when implementing the logistic regression model.

2. Split the dataset into training (60%), validation (20%), and test (20%) sets.
3. Standardize the columns in the feature matrices. To avoid information leakage, learn the standardization parameters from training, and then apply training, validation and test dataset.
4. Add a column of ones to the feature matrices of train, validation and test dataset. This is a common trick so that we can learn a coefficient for the bias term of a linear model.

In [45]:
```
### Code here
#1 Already  numpy vector
print(type(heart_disease_y))
#2
heart_disease_X_dev, heart_disease_X_test, heart_disease_y_dev, heart_disease_y_test =
heart_disease_X_train, heart_disease_X_val, heart_disease_y_train, heart_disease_y_val

#3
scaler = StandardScaler()
heart_disease_X_train = scaler.fit_transform(heart_disease_X_train)   # Fit and trans
heart_disease_X_test = scaler.transform(heart_disease_X_test)
heart_disease_X_val = scaler.transform(heart_disease_X_val)
# Transform X_val
#heart_disease_X_test = scaler.transform(X_test)         # Transform X_test

# 5. Add a column of ones to the feature matrices
heart_disease_X_train = np.hstack([np.ones((heart_disease_X_train.shape[0], 1)), heart
heart_disease_X_val = np.hstack([np.ones((heart_disease_X_val.shape[0], 1)), heart_dis
heart_disease_X_test = np.hstack([np.ones((heart_disease_X_test.shape[0], 1)), heart_d

print(heart_disease_X_train[:5], '\n\n', heart_disease_y_train[:5])
```

```
<class 'pandas.core.series.Series'>
[[ 1.          1.10947093  1.4718344  -0.96232538 -0.95845457 -0.74500255
  -0.17752347 -0.07722242 -0.66825887 -0.16627571  0.60079213  0.12520652
  -0.24664238  0.51694627 -2.15100438 -0.07698024]
 [ 1.         -0.90133051 -1.12062745  0.00308155  1.04334626  0.10455836
  -0.17752347 -0.07722242 -0.66825887 -0.16627571 -0.49865133 -0.89891325
  -0.54085384  0.33957544 -0.07171784 -0.20135476]
 [ 1.         -0.90133051 -0.76710993  0.00308155 -0.95845457 -0.74500255
  -0.17752347 -0.07722242 -0.66825887 -0.16627571  0.62322976 -0.05685922
  -0.16258197 -0.44627585  2.00756869  0.50343415]
 [ 1.          1.10947093 -0.64927075  0.96848849  1.04334626  1.20898754
  -0.17752347 -0.07722242 -0.66825887 -0.16627571  0.75785549 -0.78512217
  -0.58288404 -0.41425056  0.34413946 -0.20135476]
 [ 1.         -0.90133051 -1.59198415  0.00308155 -0.95845457 -0.74500255
  -0.17752347 -0.07722242 -0.66825887 -0.16627571 -0.65571468 -1.14925364
  -0.79303508 -1.04736587 -0.07171784 -0.36718744]]

 3245    0
1087    0
136     0
1396    0
3173    0
Name: TenYearCHD, dtype: int64
```

Implement Logistic Regression

We will now implement logistic regression with L2 regularization. Given an (m x n) feature matrix X, an (m x 1) label vector y, and an (n x 1) weight vector w, the hypothesis function for logistic regression is:

$$y = \sigma(Xw)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$, i.e. the sigmoid function. This function scales the prediction to be a probability between 0 and 1, and can then be thresholded to get a discrete class prediction.

Just as with linear regression, our objective in logistic regression is to learn the weights w which best fit the data. For L2-regularized logistic regression, we find an optimal w to minimize the following loss function:

$$\min_w \ -y^T \log(\sigma(Xw)) \ - \ (\mathbf{1} - y)^T \log(\mathbf{1} - \sigma(Xw)) \ + \ \alpha\|w\|_2^2$$

Unlike linear regression, however, logistic regression has no closed-form solution for the optimal w. So, we will use gradient descent to find the optimal w. The (n x 1) gradient vector g for the loss function above is:

$$g = X^T\Big(\sigma(Xw) - y\Big) + 2\alpha w$$

Below is pseudocode for gradient descent to find the optimal w. You should first initialize w (e.g. to a (n x 1) zero vector). Then, for some number of epochs t, you should update w with $w - \eta g$, where η is the learning rate and g is the gradient. You can learn more about gradient descent here.

> $w = \mathbf{0}$
>
> $\text{for } i = 1, 2, \ldots, t$
>
> $\quad w = w - \eta g$

A LogisticRegression class with five methods: train, predict, calculate_loss, calculate_gradient, and calculate_sigmoid has been implemented for you below.

In [46]:
```
class LogisticRegression():
    '''
    Logistic regression model with L2 regularization.

    Attributes
    ----------
    alpha: regularization parameter
    t: number of epochs to run gradient descent
    eta: learning rate for gradient descent
    w: (n x 1) weight vector
    '''

    def __init__(self, alpha=0, t=100, eta=1e-3):
        self.alpha = alpha
        self.t = t
```

```
        self.eta = eta
        self.w = None

    def train(self, X, y):
        '''Trains logistic regression model using gradient descent
        (sets w to its optimal value).

        Parameters
        ----------
        X : (m x n) feature matrix
        y: (m x 1) label vector

        Returns
        -------
        losses: (t x 1) vector of losses at each epoch of gradient descent
        '''

        loss = list()
        self.w = np.zeros((X.shape[1],1))
        for i in range(self.t):
            self.w = self.w - (self.eta * self.calculate_gradient(X, y))
            loss.append(self.calculate_loss(X, y))
        return loss

    def predict(self, X):
        '''Predicts on X using trained model. Make sure to threshold
        the predicted probability to return a 0 or 1 prediction.

        Parameters
        ----------
        X : (m x n) feature matrix

        Returns
        -------
        y_pred: (m x 1) 0/1 prediction vector
        '''
        y_pred = self.calculate_sigmoid(X.dot(self.w))
        y_pred[y_pred >= 0.5] = 1
        y_pred[y_pred < 0.5] = 0
        return y_pred

    def calculate_loss(self, X, y):
        '''Calculates the logistic regression loss using X, y, w,
        and alpha. Useful as a helper function for train().

        Parameters
        ----------
        X : (m x n) feature matrix
        y: (m x 1) label vector

        Returns
        -------
        loss: (scalar) logistic regression loss
        '''
        return -y.T.dot(np.log(self.calculate_sigmoid(X.dot(self.w)))) - (1-y).T.dot(

    def calculate_gradient(self, X, y):
        '''Calculates the gradient of the logistic regression loss
        using X, y, w, and alpha. Useful as a helper function
        for train().
```

```
        Parameters
        ----------
        X : (m x n) feature matrix
        y: (m x 1) label vector

        Returns
        -------
        gradient: (n x 1) gradient vector for logistic regression loss
        '''
        return X.T.dot(self.calculate_sigmoid( X.dot(self.w)) - y) + 2*self.alpha*self


    def calculate_sigmoid(self, x):
        '''Calculates the sigmoid function on each element in vector x.
        Useful as a helper function for predict(), calculate_loss(),
        and calculate_gradient().

        Parameters
        ----------
        x: (m x 1) vector

        Returns
        -------
        sigmoid_x: (m x 1) vector of sigmoid on each element in x
        '''
        return (1)/(1 + np.exp(-x.astype('float')))
```

2.2.5 Plot Loss over Epoch and Search the space randomly to find best hyperparameters.

A: Using your implementation above, train a logistic regression model **(alpha=0, t=100, eta=1e-3)** on the voice recognition training data. Plot the training loss over epochs. Make sure to label your axes. You should see the loss decreasing and start to converge.

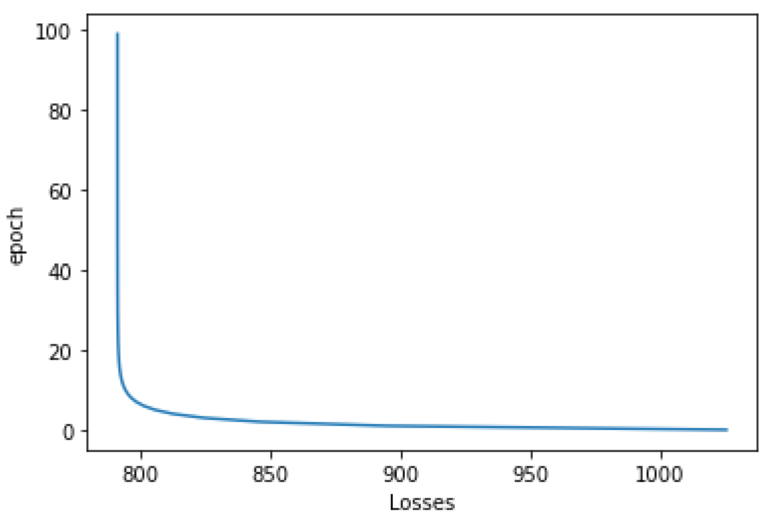
B: Using **alpha between (0,1), eta between(0, 0.001) and t between (0, 100)**, find the best hyperparameters for LogisticRegression. You can randomly search the space 20 times to find the best hyperparameters.

C. Compare accuracy on the test dataset for both the scenarios.

In [47]:
```
### Code here
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score
import random
#A
heart_disease_y_train=heart_disease_y_train.values.reshape(-1,1)
#print(heart_disease_X.shape)
#model=logreg
model=LogisticRegression(alpha=0, t=100, eta=1e-3)
losses=model.train(heart_disease_X_train,heart_disease_y_train)
losses=np.asarray(losses).flatten()
losses=losses.reshape(-1,1)
#print(np.shape(losses))
epochs=list(range(100))
epochs=np.asarray(epochs).reshape(-1,1)
plt.plot(losses,epochs)
plt.xlabel('Losses')
```

```
plt.ylabel('epoch')
#B for loop
val_scores=[]
avals=[]
tvals=[]
eta_vals=[]
alphas=list(np.arange(0, 1.01, 0.001))
etas=list(np.arange(0, 0.001, 0.000001))
ts=list(range(0,101))
for i in range(0,20):
    a=random.choice(alphas)
    t1=random.choice(ts)
    eta1=random.choice(etas)
    modelB=LogisticRegression(alpha=a,t=t1,eta=eta1)
    modelB.train(heart_disease_X_train,heart_disease_y_train)
    modelB_pred=modelB.predict(heart_disease_X_val)
    val_scores.append(accuracy_score(modelB_pred,heart_disease_y_val))
    avals.append(a)
    tvals.append(t1)
    eta_vals.append(eta1)
print(f"Best Accuracy Score: {np.max(val_scores): .3f}")
idx=val_scores.index(max(val_scores))
print(f"Best Hyperparameter alpha: {alphas[idx]: .3f}")
print(f"Best Hyperparameter t: {ts[idx]: .3f}")
print(f"Best Hyperparameter eta: {etas[idx]: .3f}")
#C
model=LogisticRegression(alpha=0, t=100, eta=1e-3)
model.train(heart_disease_X_train,heart_disease_y_train)
a_pred=model.predict(heart_disease_X_test)
print(f"Accuracy Score for Part A: {accuracy_score(a_pred,heart_disease_y_test): .3f}'
modelB=LogisticRegression(alpha=alphas[idx], t=ts[idx], eta=etas[idx])
modelB.train(heart_disease_X_train,heart_disease_y_train)
b_pred=model.predict(heart_disease_X_test)
print(f"Accuracy Score for Part B: {accuracy_score(b_pred,heart_disease_y_test): .3f}'
```

```
Best Accuracy Score:  0.821
Best Hyperparameter alpha:  0.002
Best Hyperparameter t:  2.000
Best Hyperparameter eta:  0.000
Accuracy Score for Part A:  0.854
Accuracy Score for Part B:  0.854
```



2.2.6 Do you think the model is performing well keeping the class distribution in mind?

In [48]:
```
#### Comment here
#This model is actually doing pretty well given the class imbalance. This is because w
#likely the model will predict the datapoint to be that of the majority class. This wi
#in this situation the accuracy is actually pretty high. So, the model is performing w
```

> We will look into different evaluation metrics in Lecture 5 that will help us with such imbalanced datasets.

Feature Importance

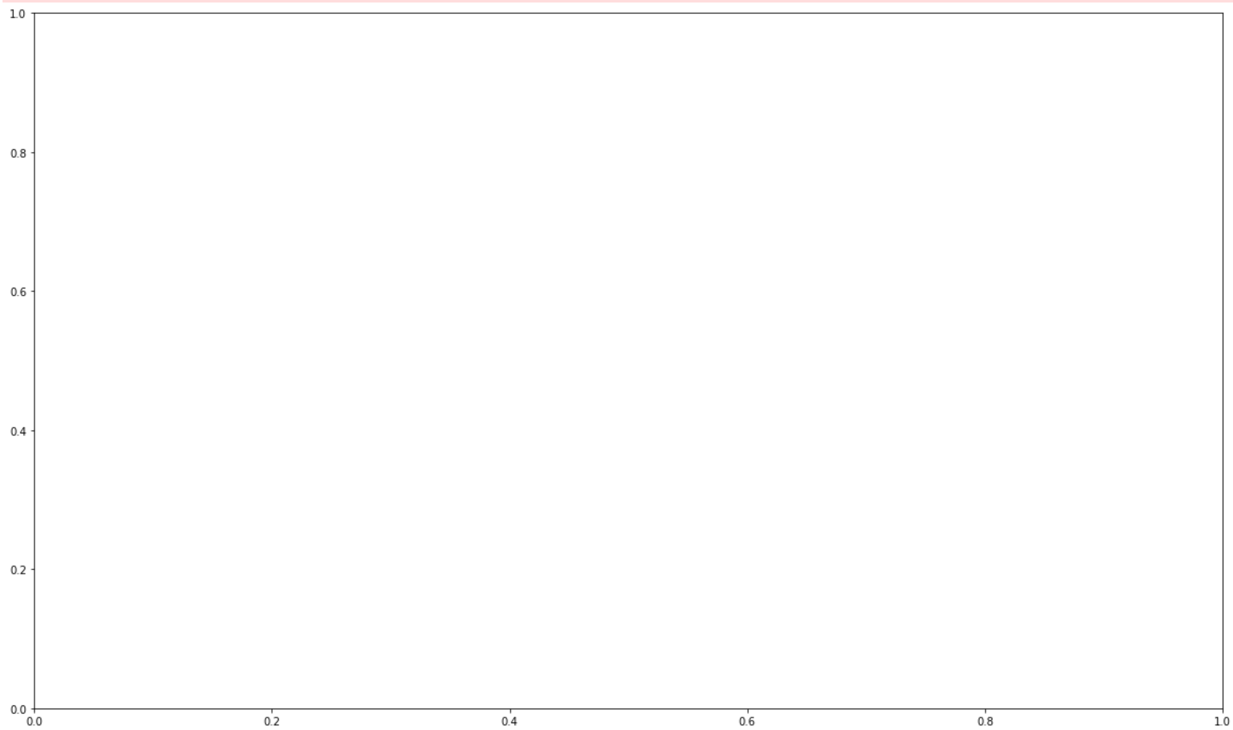
2.2.7 Interpret your trained model using a bar chart of the model weights. Make sure to label the bars (x-axis) and don't forget the bias term!

In [49]:
```
### Code here
#model=LogisticRegression()
#heart_disease_X_train=heart_disease_X_train.reshape(heart_disease_X_train.shape[0],2)
#model.train(heart_disease_X_train[:,[1]],heart_disease_y_train)
fig = plt.figure(figsize = (20,12))
#xval = np.zeros((31))
yw = model.w
#heart_disease_X_train = np.hstack([np.ones((heart_disease_X_train.shape[0], 1)), hear
#bias=[np.ones((heart_disease_X.shape[0], 1)), heart_disease_X]
#print(bias)
hd=heart_disease_X
hd['bias']=1
#hd.insert(len(hd.columns),'bias',)
#print(hd)
#hd['bias']=bias
cols=hd.columns.values.reshape(-1,1)
#print(cols.shape)
#print(heart_disease_X)
yw=yw.reshape(-1,1)
print(type(yw))
#print(yw.shape)
#xval.shape
plt.bar(cols, yw)
#ax.tick_params(axis='x', rotation=90)
ax.set_ylabel('feature importance (coefficient)')
ax.set_title('feature importance across features')
plt.show()
```

```
<class 'numpy.ndarray'>
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [49], in <cell line: 23>()
     20 print(type(yw))
     21 #print(yw.shape)
     22 #xval.shape
---> 23 plt.bar(cols, yw)
     24 #ax.tick_params(axis='x', rotation=90)
     25 ax.set_ylabel('feature importance (coefficient)')

File ~\anaconda3\lib\site-packages\matplotlib\pyplot.py:2387, in bar(x, height, widt
h, bottom, align, data, **kwargs)
   2383 @_copy_docstring_and_deprecators(Axes.bar)
   2384 def bar(
   2385         x, height, width=0.8, bottom=None, *, align='center',
   2386         data=None, **kwargs):
-> 2387     return gca().bar(
   2388         x, height, width=width, bottom=bottom, align=align,
   2389         **({"data": data} if data is not None else {}), **kwargs)

File ~\anaconda3\lib\site-packages\matplotlib\__init__.py:1412, in _preprocess_data.<
locals>.inner(ax, data, *args, **kwargs)
   1409 @functools.wraps(func)
   1410 def inner(ax, *args, data=None, **kwargs):
   1411     if data is None:
-> 1412         return func(ax, *map(sanitize_sequence, args), **kwargs)
   1414     bound = new_sig.bind(ax, *args, **kwargs)
   1415     auto_label = (bound.arguments.get(label_namer)
   1416                   or bound.kwargs.get(label_namer))

File ~\anaconda3\lib\site-packages\matplotlib\axes\_axes.py:2317, in Axes.bar(self,
x, height, width, bottom, align, **kwargs)
   2314         x = 0
   2316 if orientation == 'vertical':
-> 2317     self._process_unit_info(
   2318         [("x", x), ("y", height)], kwargs, convert=False)
   2319     if log:
   2320         self.set_yscale('log', nonpositive='clip')

File ~\anaconda3\lib\site-packages\matplotlib\axes\_base.py:2521, in _AxesBase._proce
ss_unit_info(self, datasets, kwargs, convert)
   2519     # Update from data if axis is already set but no unit is set yet.
   2520     if axis is not None and data is not None and not axis.have_units():
-> 2521         axis.update_units(data)
   2522 for axis_name, axis in axis_map.items():
   2523     # Return if no axis is set.
   2524     if axis is None:

File ~\anaconda3\lib\site-packages\matplotlib\axis.py:1449, in Axis.update_units(sel
f, data)
   1447 neednew = self.converter != converter
   1448 self.converter = converter
-> 1449 default = self.converter.default_units(data, self)
   1450 if default is not None and self.units is None:
   1451     self.set_units(default)

File ~\anaconda3\lib\site-packages\matplotlib\category.py:116, in StrCategoryConverte
r.default_units(data, axis)
    114 # the conversion call stack is default_units -> axis_info -> convert
    115 if axis.units is None:
```

```
--> 116     axis.set_units(UnitData(data))
    117 else:
    118     axis.units.update(data)

File ~\anaconda3\lib\site-packages\matplotlib\category.py:192, in UnitData.__init__(s
elf, data)
    190 self._counter = itertools.count()
    191 if data is not None:
--> 192     self.update(data)

File ~\anaconda3\lib\site-packages\matplotlib\category.py:225, in UnitData.update(sel
f, data)
    223 # check if convertible to number:
    224 convertible = True
--> 225 for val in OrderedDict.fromkeys(data):
    226     # OrderedDict just iterates over unique values in data.
    227     _api.check_isinstance((str, bytes), value=val)
    228     if convertible:
    229         # this will only be called so long as convertible is True.

TypeError: unhashable type: 'numpy.ndarray'
```



In [ ]:
```
#### Comment here
```

Part 3: Support Vector Machines

In this part, we will be using support vector machines for classification on the heart disease dataset.

Train Primal SVM

3.1 Train a primal SVM (with default parameters) on the heart disease dataset. Make predictions and report the accuracy on the training, validation, and test sets.

In [50]:
```
### Code here
SVMPrimal = LinearSVC(dual=False)
SVMPrimal.fit(heart_disease_X_train,heart_disease_y_train)
prim_train=SVMPrimal.predict(heart_disease_X_train)
prim_val=SVMPrimal.predict(heart_disease_X_val)
prim_test=SVMPrimal.predict(heart_disease_X_test)
print(f"Accuracy Score for Training: {accuracy_score(prim_train,heart_disease_y_train)
print(f"Accuracy Score for Validation: {accuracy_score(prim_val,heart_disease_y_val):
print(f"Accuracy Score for Testing: {accuracy_score(prim_test,heart_disease_y_test): .
```

```
Accuracy Score for Training:  0.861
Accuracy Score for Validation:  0.821
Accuracy Score for Testing:  0.852
```

```
C:\Users\cherr\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected. Please chang
e the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Train Dual SVM

3.2 Train a dual SVM (with default parameters) on the heart disease dataset. Make predictions and report the accuracy on the training, validation, and test sets.

In [51]:
```
### Code here
SVMDual = LinearSVC(dual=False)
SVMDual.fit(heart_disease_X_train,heart_disease_y_train)
dual_train=SVMDual.predict(heart_disease_X_train)
dual_val=SVMDual.predict(heart_disease_X_val)
dual_test=SVMDual.predict(heart_disease_X_test)
print(f"Accuracy Score for Training: {accuracy_score(dual_train,heart_disease_y_train)
print(f"Accuracy Score for Validation: {accuracy_score(dual_val,heart_disease_y_val):
print(f"Accuracy Score for Testing: {accuracy_score(dual_test,heart_disease_y_test): .
```

```
Accuracy Score for Training:  0.861
Accuracy Score for Validation:  0.821
Accuracy Score for Testing:  0.852
```

```
C:\Users\cherr\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected. Please chang
e the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```