Introduction:

In this project, I will analyze used car prices and finally build a machine learning algorithm that can predict the price that you should list your used car for.

The data I obtained for this came from kaggle. It's a data set that was created by being scraped off of craigslist. The original dataset has over five-hundred thousand data entries but I will just be using a small portion of it.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import seaborn as sns
from math import ceil, floor
import numpy as np
import re
```

Cleaning the Data:

The original data set has many different variables but a lot of them are not needed for this. I took the original data set and took it down to five different variables: Price, Year, Manufacturer, Cyliners and Odometer.

A lot of the data has missing values so the way I will deal with this is to just drop all rows that contain an N/A value. There were also many value with empty strings. These were also all dropped.

There was also many values in price and odometer that appear to be outliers. For the Odometer variable, I filtered them out by dropping the bottom 2.5% and the top 97.5%. For the Price variable, I dropped any value under 100 dollars and dropped any value over 60,000$.

For cylinders, I dropped the cylinders and just left the number instead.

In [2]:
```
cars = pd.read_csv('vehicles.csv', nrows=10000, keep_default_na=False)
cars_columns = ['price', 'year', 'manufacturer', 'cylinders', 'odometer']
cars = cars[cars_columns]
cars.iloc[1500:1503]
```

Out[2]:

	price	year	manufacturer	cylinders	odometer
1500	6980	2006	infiniti	6 cylinders	144632
1501	4450	2004	cadillac	6 cylinders	144850
1502	7980	2009	bmw	6 cylinders	123000

In [3]:
```
# Cleaning cars.odometer
cars.odometer = pd.to_numeric(cars.odometer)
cars = cars.dropna(subset=(['odometer']))
cars = cars.where((cars.odometer < floor(np.percentile(cars.odometer, 97.5)))
                & (cars.odometer > floor(np.percentile(cars.odometer, 2.5))))
cars = cars.dropna(subset=(['odometer']))
cars.odometer = cars.odometer.astype('int32')
```

In [4]:
```
# Cleaning cars.cylinders
# cars.cylinders = cars.cylinders[cars.cylinders == '8 cylinders']
cars.cylinders = cars.cylinders[cars.cylinders.isin(['8 cylinders', '6 cylinders', '4 cylinders'])]
cars = cars.dropna(subset=['cylinders'])
cars.cylinders = cars.cylinders.str[0]
```

In [5]:
```
# Cleaning cars.price
cars = cars.where((cars.price >= 100) & (cars.price <= 60000))
cars = cars.dropna(subset=['price'])
cars.price = cars.price.astype('int32')
# Cleaning cars.year
cars.year = cars.year.replace('', np.nan)
cars = cars.dropna(subset=['year'])
cars.odometer = cars.odometer.astype('int32')
# Cleaning cars.manufacturer
cars.manufacturer = cars.manufacturer.replace('', np.nan)
cars = cars.dropna(subset=['manufacturer'])
cars.reset_index(drop=True, inplace=True)
```

In [6]:
```
print('Length of the cleansed data set that we will be using: \t',len(cars))
```

```
Length of the cleansed data set that we will be using:   5765
```

Feature Engineering:

In addition to the variables included in the dataset, I will also be adding two new ones.

The first one will be the yearly milage. The second one will be the age of the car.

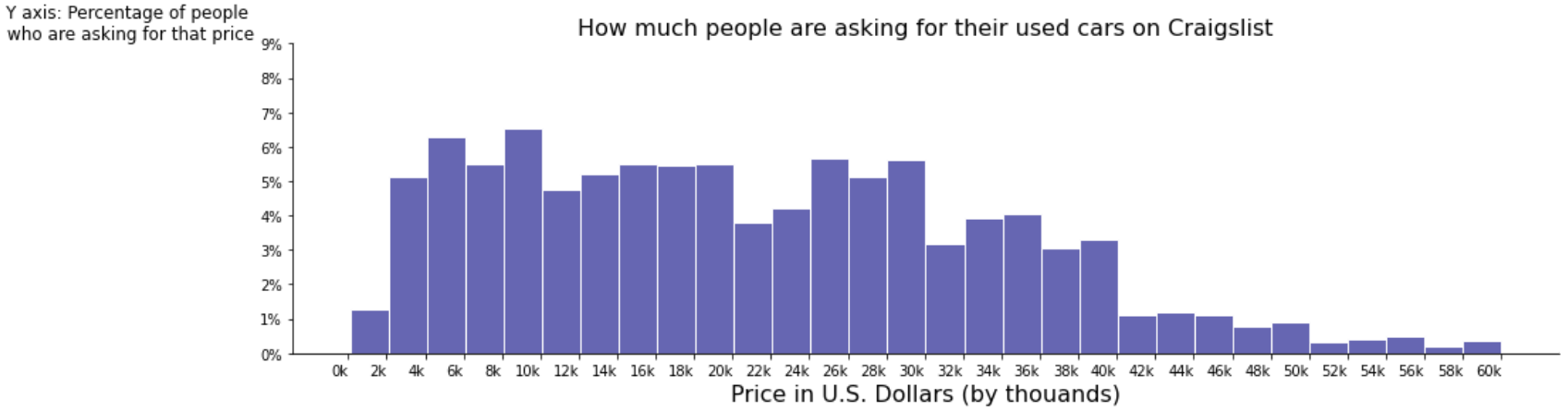
In [7]:
```
# Creating a new variable - Yearly milage
from datetime import datetime

cars['yearly_milage'] = cars.apply(
    lambda row: ceil((row.odometer) / (datetime.now().year-int(row.year))) if int(row.year) != datetime.now().ye
    else ceil((row.odometer) / ((datetime.now().year + 1) - int(row.year))), axis=1)
```

In [8]:
```
# Creating a new variable - Car age
def car_age(year):
    age = datetime.now().year - int(year)
    return age
cars['age'] = cars.year.apply(car_age)
cars.head()
```

Out[8]:

	price	year	manufacturer	cylinders	odometer	yearly_milage	age
0	33590	2014	gmc	8	57923	8275	7
1	22590	2010	chevrolet	8	71229	6476	11
2	39590	2020	chevrolet	8	19160	19160	1
3	30990	2017	toyota	8	41124	10281	4
4	15000	2013	ford	6	128000	16000	8

In [9]:
```
fig, ax = plt.subplots(figsize=(16,4))
sns.set_style('whitegrid')
sns.histplot(cars.price, binrange=(100, 60000), bins=30, stat='probability', alpha=0.6, color='navy')
plt.title('How much people are asking for their used cars on Craigslist', fontsize=16)
plt.xlabel('Price in U.S. Dollars (by thouands)', fontsize=16)
plt.ylabel('Y axis: Percentage of people \n who are asking for that price', rotation=360, fontsize=12, loc='top'
ax.set_xticks([x*2000 for x in range(0,31)])
ax.set_xticklabels([str(x)+'k' for x in range(0,62,2)], fontdict={'horizontalalignment': 'right'})
ax.set_yticks([x/100 for x in range(0,10)])
ax.set_yticklabels([f'{x}%' for x in range(0,10)])
sns.despine()
plt.show()
plt.clf()
```

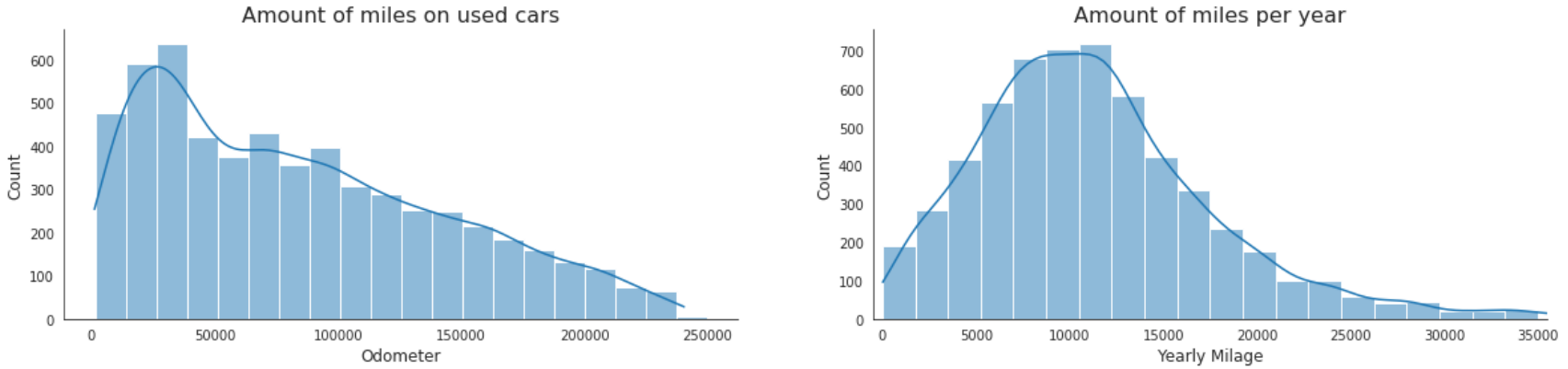



```
<Figure size 432x288 with 0 Axes>
```

In [10]:
```
from scipy.stats import skew, iqr, mode
print(cars.price.describe())
print('skew: \t', skew(cars.price))
print('iqr: \t', iqr(cars.price))
print('mode: \t', int(mode(cars.price)[0]))
print('mode freq.:', int(mode(cars.price)[1]))
print('range: \t', np.ptp(cars.price))
```

```
count     5765.000000
mean     21375.265048
std      12730.511021
min        100.000000
25%      10495.000000
50%      19995.000000
75%      29997.000000
max      59999.000000
Name: price, dtype: float64
skew:    0.4594560596332271
iqr:     19502.0
mode:    5500
mode freq.: 44
range:   59899
```

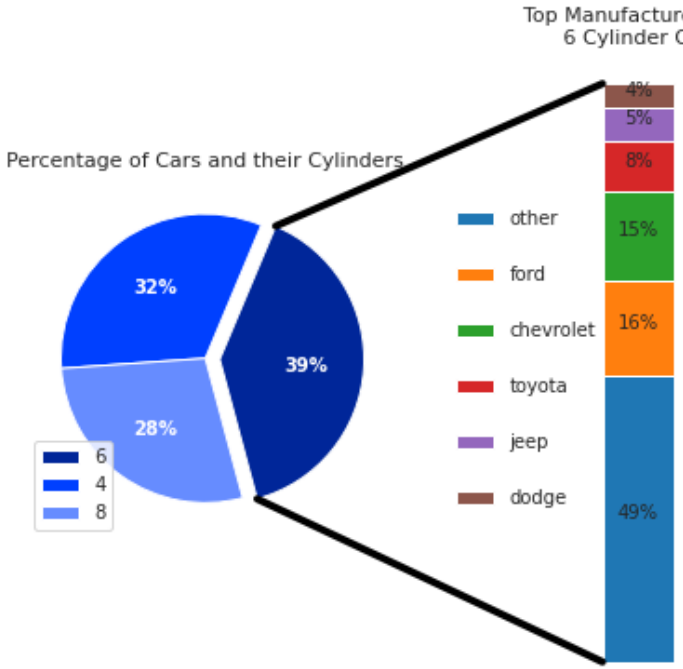
In [11]:
```
fig, ax = plt.subplots(figsize=(20,4))
sns.set_style('white')
plt.subplot(121)
sns.histplot(cars.odometer, binrange=(1500, 250000), bins=20, kde=True)
plt.title('Amount of miles on used cars', fontsize=16)
plt.xlabel('Odometer', fontsize=12)
plt.ylabel('Count', fontsize=12)

plt.subplot(122)
sns.histplot(cars.yearly_milage, binrange=(0, 35000), bins=20, kde=True)
plt.xlim(-500, 35500)
ax.set_xticks([range(0,35001,20)])
plt.xlabel('Yearly Milage', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Amount of miles per year', fontsize=16)
sns.despine()
plt.show()
```

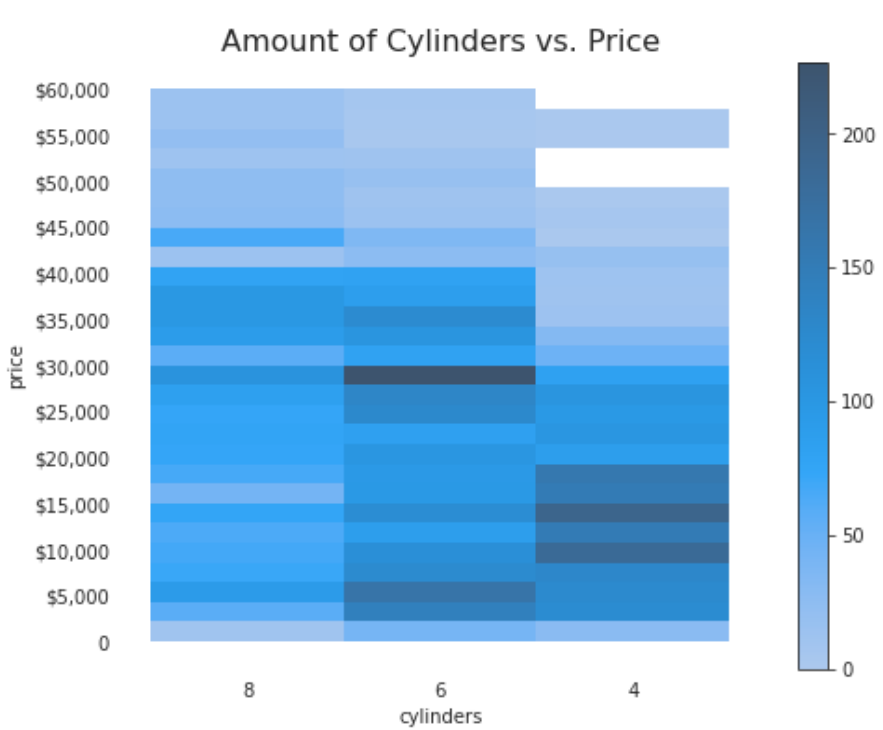


In [12]:
```
cars_mask = cars.copy()
six = cars_mask.manufacturer[cars.cylinders == '6'].value_counts().index[:5].to_list()
cars_mask['mask'] = cars_mask.manufacturer.apply(lambda x: 'other' if x not in six else x)
cars_mask.head()
six = cars_mask.iloc[:, -1]
six_l = six.value_counts(normalize=True).index
six = six.value_counts(normalize=True).values
```

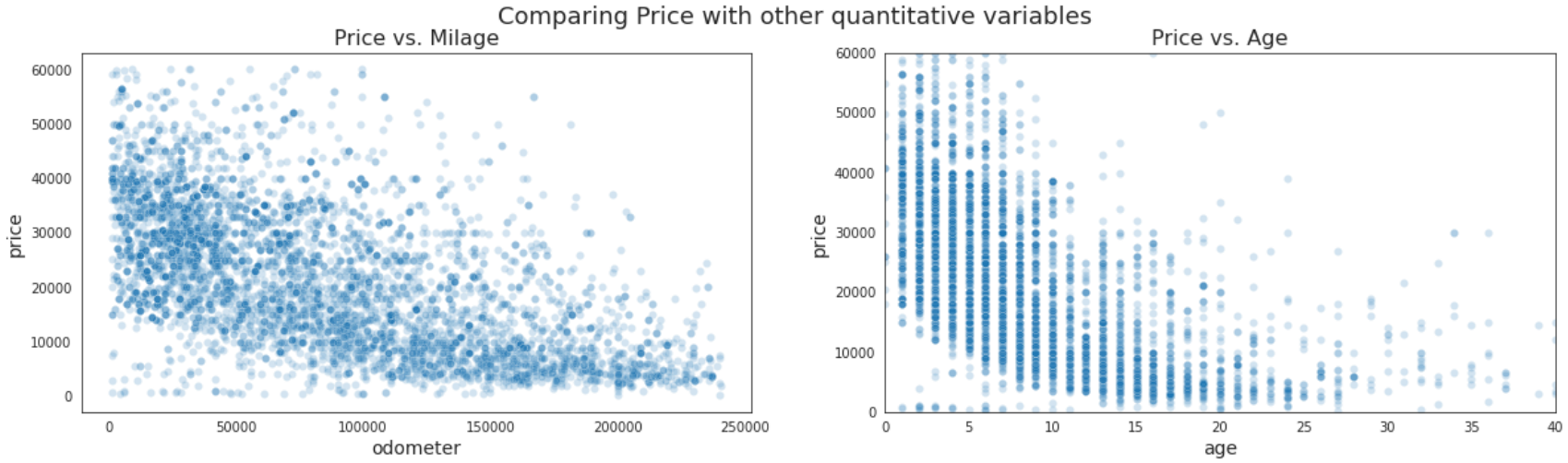
In [13]:
```
# Create bar of pie chart:
from matplotlib.patches import ConnectionPatch
fig = plt.figure(figsize=(16,6))

gs = fig.add_gridspec(1,4)
ax1 = fig.add_subplot(gs[0,0])
ax1.pie(cars.cylinders.value_counts(), labels=cars.cylinders.value_counts().index, autopct='%d%%',
        explode=[0.1, 0, 0], startangle=285, colors=['#002699', '#0040ff', '#668cff'],
        textprops={'color':"w", 'fontsize': 10, 'weight': 'bold'})

ax1.set_title('Percentage of Cars and their Cylinders', fontsize=11)
plt.legend(loc='lower left')
# bar chart parameters:
xpos = 0
bottom = 0
ratios = six
width = .2

ax2 = fig.add_subplot(gs[0,1])

for j in range(len(ratios)):
    height = ratios[j]
    ax2.bar(xpos, height, width, bottom=bottom, label=six_l[j])
    ypos = bottom + ax2.patches[j].get_height() / 2
    bottom += height
    ax2.text(xpos, ypos, "%d%%" % (ax2.patches[j].get_height() * 100), ha='center')


ax2.set_title('Top Manufacturers for \n 6 Cylinder Cars', fontsize=11)
ax2.legend(six_l, loc='center left', bbox_to_anchor=[-0.05, 0.5], frameon=False,
           fontsize=10, labelspacing=2)

ax2.axis('off')
ax2.set_xlim(- 2.5 * width, 2.5 * width)

# use ConnectionPatch to draw lines between the two plots
# get the wedge data
theta1, theta2 = ax1.patches[0].theta1, ax1.patches[0].theta2
center, r = ax1.patches[0].center, ax1.patches[0].r
bar_height = sum([item.get_height() for item in ax2.patches])

# draw top connecting line
```

```
x = r * np.cos(np.pi / 180 * theta2) + center[0]
y = r * np.sin(np.pi / 180 * theta2) + center[1]
con = ConnectionPatch(xyA=(-width / 2, bar_height), coordsA=ax2.transData,
                      xyB=(x, y), coordsB=ax1.transData)
con.set_color([0, 0, 0])
con.set_linewidth(4)
ax2.add_artist(con)

# draw bottom connecting line
x = r * np.cos(np.pi / 180 * theta1) + center[0]
y = r * np.sin(np.pi / 180 * theta1) + center[1]
con = ConnectionPatch(xyA=(-width / 2, 0), coordsA=ax2.transData,
                      xyB=(x, y), coordsB=ax1.transData)
con.set_color([0, 0, 0])
ax2.add_artist(con)
con.set_linewidth(4)

# Third plot
ax3 = fig.add_subplot(gs[0,2:])
sns.set_style('white')
sns.despine(bottom=True, left=True)
plt.title('Amount of Cylinders vs. Price', fontsize=16)
sns.histplot(data=cars, x='cylinders', y='price', cbar=True)
ax3.set_yticks(range(0,60001,5000))
ax3.set_yticklabels(f'${x},000' if x != 0 else x for x in range(0,61,5))

plt.show()
```

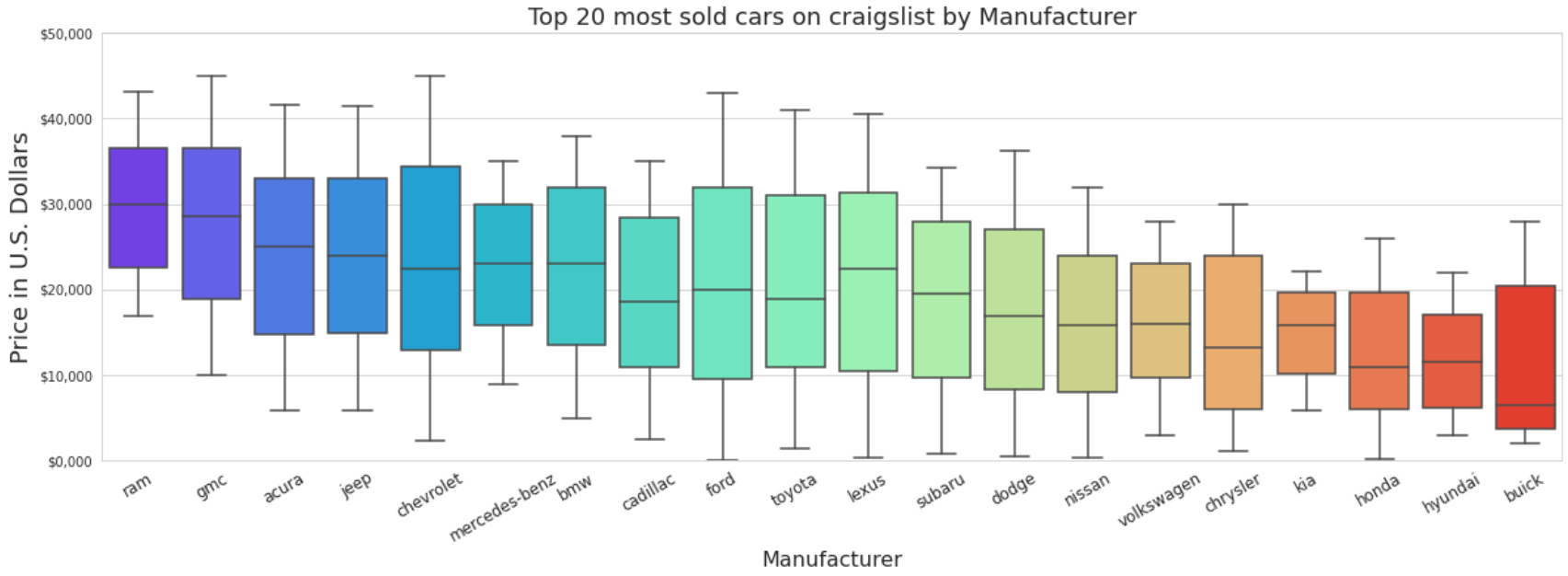

Top Manufacturers for
6 Cylinder Cars
Percentage of Cars and their Cylinders
32%
39%
28%
6
4
8
other
ford
chevrolet
toyota
jeep
dodge
4%
5%
8%
15%
16%
49%


Amount of Cylinders vs. Price
price
$60,000
$55,000
$50,000
$45,000
$40,000
$35,000
$30,000
$25,000
$20,000
$15,000
$10,000
$5,000
0
8
6
4
cylinders
200
150
100
50
0

In [14]:
```
fig, ax= plt.subplots(figsize=(20,5))
plt.suptitle('Comparing Price with other quantitative variables', fontsize=18)
plt.subplot(121)
sns.scatterplot(data=cars, x='odometer', y='price', alpha=0.2)
plt.xlabel('odometer', fontsize=14)
plt.ylabel('price', fontsize=14)
plt.title('Price vs. Milage', fontsize=16)
plt.subplot(122)
sns.scatterplot(data=cars, x='age', y='price', alpha=0.2)
plt.xlabel('age', fontsize=14)
plt.ylabel('price', fontsize=14)
plt.xlim(0,40)
plt.ylim(0, 60000)
plt.title('Price vs. Age', fontsize=16)

plt.show()
```



In [15]:
```
popular_makers = cars.manufacturer.value_counts()[:20].index.to_list()
popular_makers = cars[cars.manufacturer.isin(popular_makers)]
popular_order = popular_makers.groupby('manufacturer').price.mean().sort_values(ascending=False).index.to_list()
sns.set_style('whitegrid')
fig, ax = plt.subplots(figsize=(20,6))
sns.boxplot(data=popular_makers, x='manufacturer', y='price', order=popular_order,
                                          fliersize=False, whis=0.5, palette='rainbow')
plt.title('Top 20 most sold cars on craigslist by Manufacturer', fontsize=18)
ax.set_xticks(range(20))
ax.set_xticklabels(popular_order, rotation=30, fontsize=12)
plt.xlabel('Manufacturer', fontsize=16)
plt.ylabel('Price in U.S. Dollars', fontsize=18)
plt.ylim(0, 50000)
ax.set_yticks(range(0,50001,10000))
ax.set_yticklabels([f'${x},000' for x in range(0,51,10)])
plt.show()
```



In [16]:
```
cars.year = cars.year.astype('int32')
cars.odometer = cars.odometer.astype('int32')
numeric_variable_labels = ['price', 'year','odometer', 'yearly_milage', 'age']
numeric_variables = cars.loc[:, numeric_variable_labels].values
```

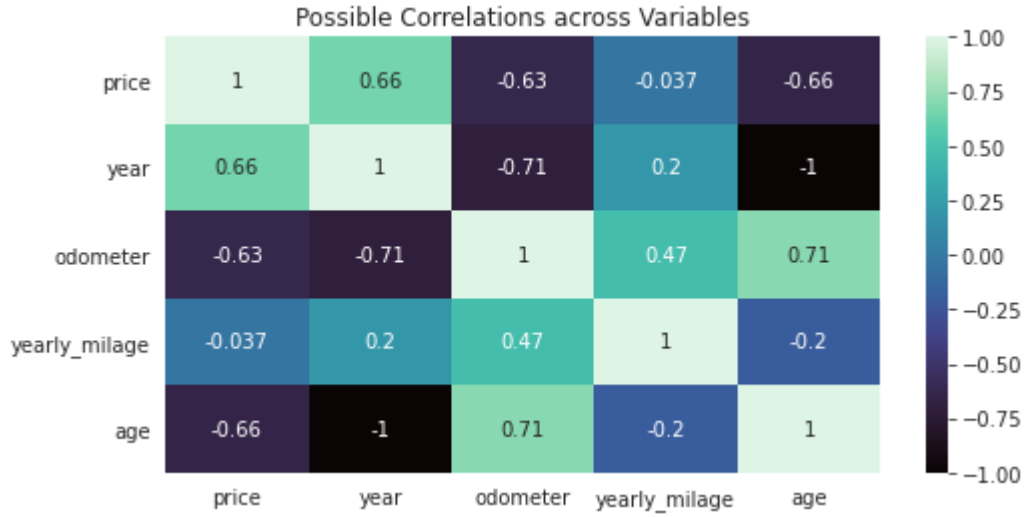
Preparing the Data for Machine Learning:

Before building the machine learning models I will standardize the data and then log transform it.

In [17]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PowerTransformer
num_pipeline = Pipeline([('std_scaler', StandardScaler()),
                         ('log_trans', PowerTransformer())])
numeric_variables_scale = num_pipeline.fit_transform(numeric_variables)
```

In [18]:
```
cars_scaled = pd.DataFrame(numeric_variables_scale, columns=numeric_variable_labels)
```

In [19]:
```
plt.figure(figsize=(8,4))
sns.heatmap(cars_scaled.corr(), annot=True, cmap='mako')
plt.title('Possible Correlations across Variables')
plt.show()
```



Transforming the categorical data to better fit the machine learning algorithm:

This will be done through One Hot Encoding for all the categorical variables. This includes cylinder count and manufacturer.

In [20]:
```
# Getting nominal variables ready for the machine learning model:
cylinders_ohe = pd.get_dummies(cars.cylinders)
maker_ohe = pd.get_dummies(cars.manufacturer)
cars_scaled = cars_scaled[['year', 'odometer', 'yearly_milage', 'age']]
X = pd.concat([cars_scaled, cylinders_ohe, maker_ohe], axis=1)
y = cars.price.to_numpy()
X.head(3)
```

Out[20]:

	year	odometer	yearly_milage	age	4	6	8	acura	alfa-romeo	audi	...	nissan	pontiac	porsche	ram	rover	saturn	sul
0	-0.081264	-0.321705	-0.386874	0.081264	0	0	1	0	0	0	...	0	0	0	0	0	0	
1	-0.742365	-0.052872	-0.769001	0.742365	0	0	1	0	0	0	...	0	0	0	0	0	0	
2	1.729436	-1.227511	1.168414	-1.729436	0	0	1	0	0	0	...	0	0	0	0	0	0	

3 rows × 43 columns

In [21]:
```
numeric_variable_labels = ['price', 'year','odometer', 'yearly_milage', 'age']
numeric_variables = cars.loc[:, numeric_variable_labels].values
```

In [22]:
```
# Splitting the model into a train and test set:
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, labels = train_test_split(
        X.values, y.reshape(-1,1), train_size=0.8, test_size=0.2, random_state=22)
from sklearn.metrics import r2_score
```

Creating the model using three different algorithms:

1) Decision Trees

2) Random Forests

3) K Nearest Neighbors

I will also implement a grid search into all of them to see which hyper parameters are best.

In [23]:
```
# Decision Tree Model:
from sklearn.tree import DecisionTreeRegressor
d_tree= DecisionTreeRegressor(max_depth=13, random_state=22)
d_tree.fit(x_train, y_train)
dt_predicted = d_tree.predict(x_test)
print('Decision Tree \t r2 score:', round(r2_score(labels, dt_predicted), 3))

# Fine tuning the model:
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(d_tree, {'max_depth': [x for x in range(1,25)]},
                           scoring='r2', return_train_score=True)
grid_search.fit(x_train, y_train)
print('Best Parameter:',grid_search.best_params_)
```

```
Decision Tree    r2 score: 0.734
Best Parameter: {'max_depth': 13}
```

In [24]:
```
# Random Forest Model
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=161, max_features=6, random_state=22)
rf_model.fit(x_train, y_train.reshape(-1))
rf_predicted = rf_model.predict(x_test)
print('Random Forest \t r2 score:', round(r2_score(labels, rf_predicted), 3))

# Fine tuning the model:
grid_search = GridSearchCV(rf_model, {'n_estimators': [x for x in range(1,201,20)]},
                                        scoring='r2', return_train_score=True)
grid_search.fit(x_train, y_train.reshape(-1))
print('Best Parameter:',grid_search.best_params_)
```

```
Random Forest    r2 score: 0.841
Best Parameter: {'n_estimators': 161}
```

In [25]:
```
# K Nearest Neighbors Model:
from sklearn.neighbors import KNeighborsRegressor
k_model = KNeighborsRegressor(n_neighbors=4)
k_model.fit(x_train, y_train)
k_predicted = k_model.predict(x_test)
print('K Nearest Neighbors \t r2 score:', round(r2_score(labels, k_predicted), 3))

# Fine tuning the model:
grid_search = GridSearchCV(k_model, {'n_neighbors': [x for x in range(1,11)]},
                                    scoring='r2', return_train_score=True)
grid_search.fit(x_train, y_train.reshape(-1))
print('Best Parameter:',grid_search.best_params_)
```

```
K Nearest Neighbors      r2 score: 0.797
Best Parameter: {'n_neighbors': 4}
```

Creating a funtion for reusability:

The function will allow you to see how much you should list your used car for on craigslist. The parameters to input are 1) year, 2) maker, 3) cylinders, 4) miles, 5) test type

In [26]:

```
def used_car_price_finder(year, manufacturer, cylinders, odometer, test='RF'):
    age = car_age(2017)
    manufacturer = manufacturer.lower()
    yearly_milage = None
    if age != 0 :
        yearly_milage = ceil(odometer/age)
    elif age == 0:
        yearly_milage = ceil (odometer/1)
    quantitative_var = ['year', 'odometer', 'yearly_milage', 'age']
    cars_data = cars.loc[:, quantitative_var]
    cars_data.iloc[[-1], :] = [str(year), odometer, yearly_milage, age]
    num_pipeline = Pipeline([('std_scaler', StandardScaler()),
                             ('log_trans', PowerTransformer())])
    cars_data = num_pipeline.fit_transform(cars_data)
    my_car = cars_data[-1]
    cylinders = X.columns.to_list()[4:7]
    makers = X.columns.to_list()[7:]
    car_maker = []
    car_cylinder = []
    for maker in makers:
        if maker == manufacturer:
            car_maker.append(1)
        else:
            car_maker.append(0)
    for cylinder in cylinders:
        if cylinder == str(4):
            car_cylinder.append(1)
        else:
            car_cylinder.append(0)
    my_car = list(my_car) + car_cylinder + car_maker
    if test == 'RF':
        predicted = rf_model.predict(np.array([my_car]))
    elif test == 'DT':
        predicted = d_tree.predict(np.array([my_car]))
    elif test == 'KNN':
        predicted = k_model.predict(np.array([my_car]))
    elif test == 'LR':
        predicted = multi_linear.predict(np.array([my_car]))
    predicted = floor(predicted)
    return f'According to craigslist, you should sell your car for ${predicted} dollars.'
```

Testing the model:

How much should I sell my 2017 Hyundai with 50,000 miles for?

In [27]:
```
print('Random Forest Model:\t', used_car_price_finder(2017, 'hyundai', 4, 50000, test='RF'))
print('Decision Tree Model:\t', used_car_price_finder(2017, 'hyundai', 4, 50000, test='DT'))
print('KNN Model:\t\t', used_car_price_finder(2017, 'hyundai', 4, 50000, test='KNN'))
```

```
Random Forest Model:	According to craigslist, you should sell your car for $18046 dollars.
Decision Tree Model:	According to craigslist, you should sell your car for $18470 dollars.
KNN Model:		According to craigslist, you should sell your car for $16499 dollars.
```