In [1]:
```
import pandas as pd
```

In [2]:
```
#Importing dataset
df = pd.read_csv("vehicles.csv")
df
```

Out[2]:

	id	url	region	region_url	price	year	manufacturer	model	condition	cylin
0	7222695916	https://prescott.craigslist.org/cto/d/prescott...	prescott	https://prescott.craigslist.org	6000	NaN	NaN	NaN	NaN	
1	7218891961	https://fayar.craigslist.org/ctd/d/bentonville...	fayetteville	https://fayar.craigslist.org	11900	NaN	NaN	NaN	NaN	
2	7221797935	https://keys.craigslist.org/cto/d/summerland-k...	florida keys	https://keys.craigslist.org	21000	NaN	NaN	NaN	NaN	
3	7222270760	https://worcester.craigslist.org/cto/d/west-br...	worcester / central MA	https://worcester.craigslist.org	1500	NaN	NaN	NaN	NaN	
4	7210384030	https://greensboro.craigslist.org/cto/d/trinit...	greensboro	https://greensboro.craigslist.org	4900	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...
426875	7301591192	https://wyoming.craigslist.org/ctd/d/atlanta-2...	wyoming	https://wyoming.craigslist.org	23590	2019.0	nissan	maxima s sedan 4d	good	cylin
426876	7301591187	https://wyoming.craigslist.org/ctd/d/atlanta-2...	wyoming	https://wyoming.craigslist.org	30590	2020.0	volvo	s60 t5 momentum sedan 4d	good	
426877	7301591147	https://wyoming.craigslist.org/ctd/d/atlanta-2...	wyoming	https://wyoming.craigslist.org	34990	2020.0	cadillac	xt4 sport suv 4d	good	
426878	7301591140	https://wyoming.craigslist.org/ctd/d/atlanta-2...	wyoming	https://wyoming.craigslist.org	28990	2018.0	lexus	es 350 sedan 4d	good	cylin
426879	7301591129	https://wyoming.craigslist.org/ctd/d/atlanta-2...	wyoming	https://wyoming.craigslist.org	30590	2019.0	bmw	4 series 430i gran coupe	good	

426880 rows × 26 columns

Preliminary Data Exploration

In [3]:
```
#checking the shape of our dataset
df.shape
```

Out[3]:
```
(426880, 26)
```

In [4]:
```
#getting all columns in our dataset
df.columns
```

Out[4]:
```
Index(['id', 'url', 'region', 'region_url', 'price', 'year', 'manufacturer',
       'model', 'condition', 'cylinders', 'fuel', 'odometer', 'title_status',
       'transmission', 'VIN', 'drive', 'size', 'type', 'paint_color',
       'image_url', 'description', 'county', 'state', 'lat', 'long',
       'posting_date'],
      dtype='object')
```

In [5]:
```
#Checking info like number of columns, datatype of columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426880 entries, 0 to 426879
Data columns (total 26 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            426880 non-null  int64  
 1   url           426880 non-null  object 
 2   region        426880 non-null  object 
 3   region_url    426880 non-null  object 
 4   price         426880 non-null  int64  
 5   year          425675 non-null  float64
 6   manufacturer  409234 non-null  object 
 7   model         421603 non-null  object 
 8   condition     252776 non-null  object 
 9   cylinders     249202 non-null  object 
 10  fuel          423867 non-null  object 
 11  odometer      422480 non-null  float64
 12  title_status  418638 non-null  object 
 13  transmission  424324 non-null  object 
 14  VIN           265838 non-null  object 
 15  drive         296313 non-null  object 
 16  size          120519 non-null  object 
 17  type          334022 non-null  object 
 18  paint_color   296677 non-null  object 
 19  image_url     426812 non-null  object 
 20  description   426810 non-null  object 
 21  county        0 non-null       float64
 22  state         426880 non-null  object 
 23  lat           420331 non-null  float64
 24  long          420331 non-null  float64
 25  posting_date  426812 non-null  object 
dtypes: float64(5), int64(2), object(19)
memory usage: 84.7+ MB
```

In [6]:
```
#Check for missing values
df.isnull().sum()
```

Out[6]:
```
id                  0
url                 0
region              0
region_url          0
price               0
year             1205
manufacturer    17646
model            5277
condition      174104
cylinders      177678
fuel             3013
odometer         4400
title_status     8242
transmission     2556
VIN            161042
drive          130567
size           306361
type            92858
paint_color    130203
image_url          68
description        70
county         426880
state               0
lat              6549
long             6549
posting_date       68
dtype: int64
```

In [7]:
```
#understanding more about our features
list1 = ['manufacturer','condition','cylinders', 'fuel','title_status',
       'transmission','drive', 'size', 'type']
```

In [8]:
```
for i in list1:
    print(i, df[i].unique())
```

```
manufacturer [nan 'gmc' 'chevrolet' 'toyota' 'ford' 'jeep' 'nissan' 'ram' 'mazda'
 'cadillac' 'honda' 'dodge' 'lexus' 'jaguar' 'buick' 'chrysler' 'volvo'
 'audi' 'infiniti' 'lincoln' 'alfa-romeo' 'subaru' 'acura' 'hyundai'
 'mercedes-benz' 'bmw' 'mitsubishi' 'volkswagen' 'porsche' 'kia' 'rover'
 'ferrari' 'mini' 'pontiac' 'fiat' 'tesla' 'saturn' 'mercury'
 'harley-davidson' 'datsun' 'aston-martin' 'land rover' 'morgan']
condition [nan 'good' 'excellent' 'fair' 'like new' 'new' 'salvage']
cylinders [nan '8 cylinders' '6 cylinders' '4 cylinders' '5 cylinders' 'other'
 '3 cylinders' '10 cylinders' '12 cylinders']
fuel [nan 'gas' 'other' 'diesel' 'hybrid' 'electric']
title_status [nan 'clean' 'rebuilt' 'lien' 'salvage' 'missing' 'parts only']
transmission [nan 'other' 'automatic' 'manual']
drive [nan 'rwd' '4wd' 'fwd']
size [nan 'full-size' 'mid-size' 'compact' 'sub-compact']
type [nan 'pickup' 'truck' 'other' 'coupe' 'SUV' 'hatchback' 'mini-van' 'sedan'
 'offroad' 'bus' 'van' 'convertible' 'wagon']
```

In [9]:
```
for i in list1:
    print(i, df[i].nunique())
```

```
manufacturer 42
condition 6
cylinders 8
fuel 5
title_status 6
transmission 3
drive 3
size 4
type 13
```

In [10]:
```
df["year"].nunique()
```

Out[10]:
```
114
```

In [11]:
```
df["year"].unique()
```

Out[11]:
```
array([  nan, 2014., 2010., 2020., 2017., 2013., 2012., 2016., 2019.,
       2011., 1992., 2018., 2004., 2015., 2001., 2006., 1968., 2003.,
       2008., 2007., 2005., 1966., 2009., 1998., 2002., 1999., 2021.,
       1997., 1976., 1969., 1995., 1978., 1954., 1979., 1970., 1974.,
       1996., 1987., 2000., 1955., 1960., 1991., 1972., 1988., 1994.,
       1929., 1984., 1986., 1989., 1973., 1946., 1933., 1958., 1937.,
       1985., 1957., 1953., 1942., 1963., 1977., 1993., 1903., 1990.,
       1965., 1982., 1948., 1983., 1936., 1932., 1951., 1931., 1980.,
       1967., 1971., 1947., 1981., 1926., 1962., 1975., 1964., 1934.,
       1952., 1940., 1959., 1950., 1930., 1956., 1922., 1928., 2022.,
       1901., 1941., 1924., 1927., 1939., 1923., 1949., 1961., 1935.,
       1918., 1900., 1938., 1913., 1916., 1943., 1925., 1921., 1915.,
       1945., 1902., 1905., 1920., 1944., 1910., 1909.])
```

Data Cleaning

In [12]:
```
# Dropping unnecessary columns
df.drop(['id', 'url', 'region_url', 'VIN', 'image_url',
         'description', 'county', 'lat', 'long',
         'posting_date', 'size', 'state'], axis=1, inplace=True)
```

In [13]:
```
#Shape of our dataset has been reduced from 26 columns to 14 columns as we have removed unnecessary columns
df.shape
```

Out[13]:
```
(426880, 14)
```

In [14]:
```
#checking null values
df.isnull().sum()
```

Out[14]:
```
region              0
price               0
year             1205
manufacturer    17646
model            5277
condition      174104
cylinders      177678
fuel             3013
odometer         4400
title_status     8242
transmission     2556
drive          130567
type            92858
paint_color    130203
dtype: int64
```

Handling missing values

In [15]:
```
df.isna().sum()/df.shape[0]*100
```

Out[15]:
```
region          0.000000
price           0.000000
year            0.282281
manufacturer    4.133714
model           1.236179
condition      40.785232
cylinders      41.622470
fuel            0.705819
odometer        1.030735
title_status    1.930753
transmission    0.598763
drive          30.586347
type           21.752717
paint_color    30.501078
dtype: float64
```

In [16]:
```
# dropping the rows with null values less than five percent
df = df.dropna(subset=['year', 'odometer', 'manufacturer', 'model','fuel','title_status','transmission'])
```

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
region              0
price               0
year                0
manufacturer        0
model               0
condition      157282
cylinders      161353
fuel                0
odometer            0
title_status        0
transmission        0
drive          115076
type            82628
paint_color    112494
dtype: int64
```

In [18]:
```
df.fillna('unknown', inplace=True)
```

```
C:\Users\Vikas Reddy\AppData\Local\Temp\ipykernel_4408\3438581698.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.fillna('unknown', inplace=True)
```

In [19]:
```
df.duplicated().sum()
```

Out[19]:
```
51015
```

In [20]:
```
# dropping duplicates
df = df.drop_duplicates()
```

In [21]:
```
# final rows and columns
df.shape
```

Out[21]:
```
(338589, 14)
```

In [22]:
```
#changing type of year
df['year'].dtype
```

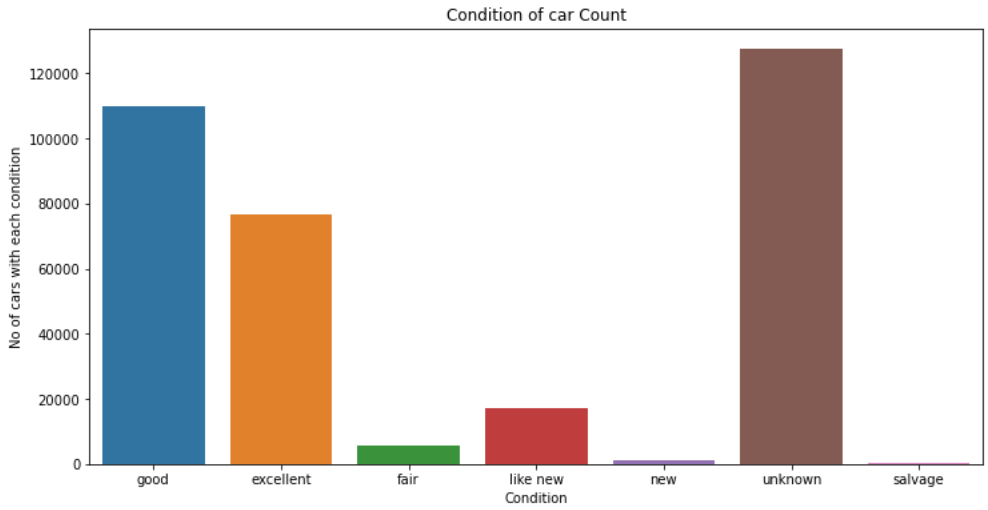
Out[22]:
```
dtype('float64')
```

In [23]:
```
df['year'] = df['year'].astype(int)
df['year'].dtype
```

Out[23]:
```
dtype('int32')
```

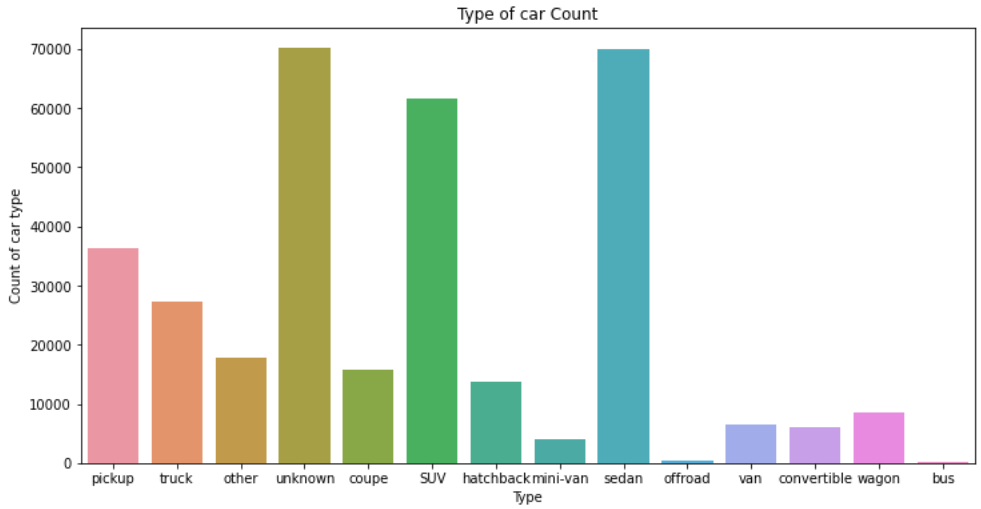
In [24]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [25]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='condition')

plt.title('Condition of car Count')
plt.xlabel('Condition')
plt.ylabel('No of cars with each condition')
plt.show()
```



The figure above visually represents the quantity of cars categorized by their respective conditions. All the cars are mostly good and excellent. We have very few number of fair and new cars.

In [26]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='type')

plt.title('Type of car Count')
plt.xlabel('Type')
plt.ylabel('Count of car type')
plt.show()
```



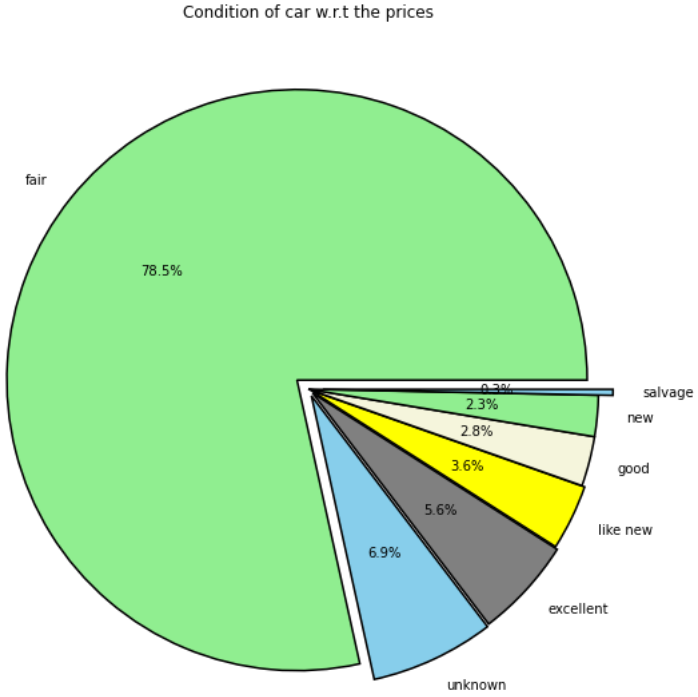
the above figure displays the number of cars in each type. We have most sedans and next comes SUV excluding unknowns and the least are bus and offroad.

Exploratory Data Analysis

Distribution of Car Prices by Condition

In [27]:
```
explode = (0.05, 0.025, 0.02, 0.01, 0.0,0.0,0.05)
colors = ( "lightgreen", "skyblue","grey", "yellow", "beige")
wedgeprops={"edgecolor":"black",'linewidth': 1.5,'antialiased': True}

# Group the data by condition and calculate the mean price for each condition
df.groupby('condition')['price'].mean().sort_values(ascending=False).plot(ylabel="",
                                                                          colors= colors,
                                                                          explode= explode,
                                                                          autopct='%1.1f%%',
                                                                          title = 'Condition of car w.r.t the prices',
                                                                          figsize=(10,10),
                                                                          kind='pie',
                                                                          wedgeprops = wedgeprops,
                                                                          stacked=True)
```

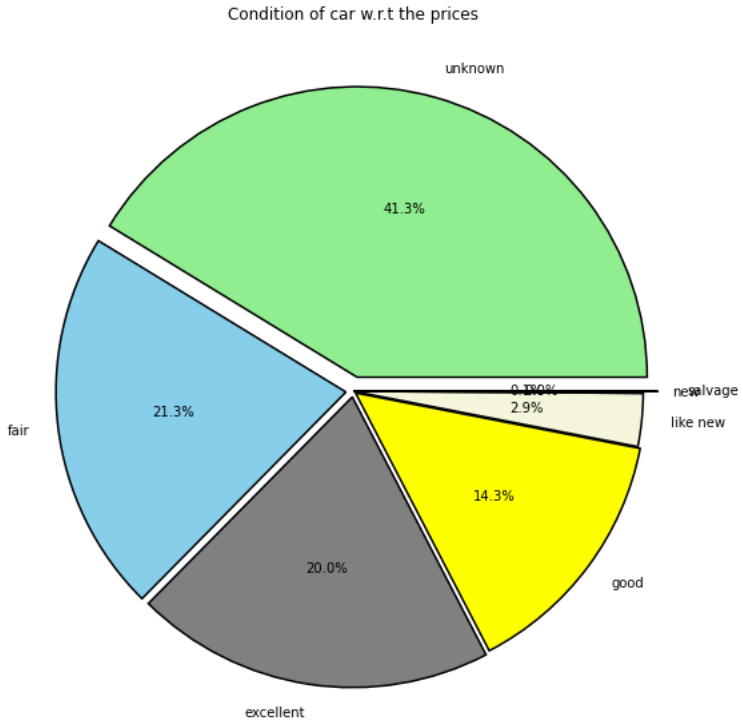
Out[27]:
```
<AxesSubplot:title={'center':'Condition of car w.r.t the prices'}>
```



The above pie chart visualizes the average prices of used cars grouped by their condition. The result shows us that fair condition cars account for about 79% of the total average prices. This suggests fair condition makes up the largest share of used cars in terms of average price. The other conditions - good, excellent, like new - make up smaller percentages. From the above chart we can conclude that the fair condition cars dominate the used car market in terms of price.

In [28]:
```
explode = (0.05, 0.025, 0.02, 0.01, 0.0,0.0,0.05)
colors = ( "lightgreen", "skyblue","grey", "yellow", "beige")
wedgeprops={"edgecolor":"black",'linewidth': 1.5,'antialiased': True}

# Group the data by condition and calculate the total price for each condition
df.groupby('condition')['price'].sum().sort_values(ascending=False).plot(ylabel="",
                                                                         colors= colors,
                                                                         explode= explode,
                                                                         autopct='%1.1f%%',
                                                                         title = 'Condition of car w.r.t the prices',
                                                                         figsize=(10,10),
                                                                         kind='pie',
                                                                         wedgeprops = wedgeprops,
                                                                         stacked=True)
```

Out[28]:
```
<AxesSubplot:title={'center':'Condition of car w.r.t the prices'}>
```

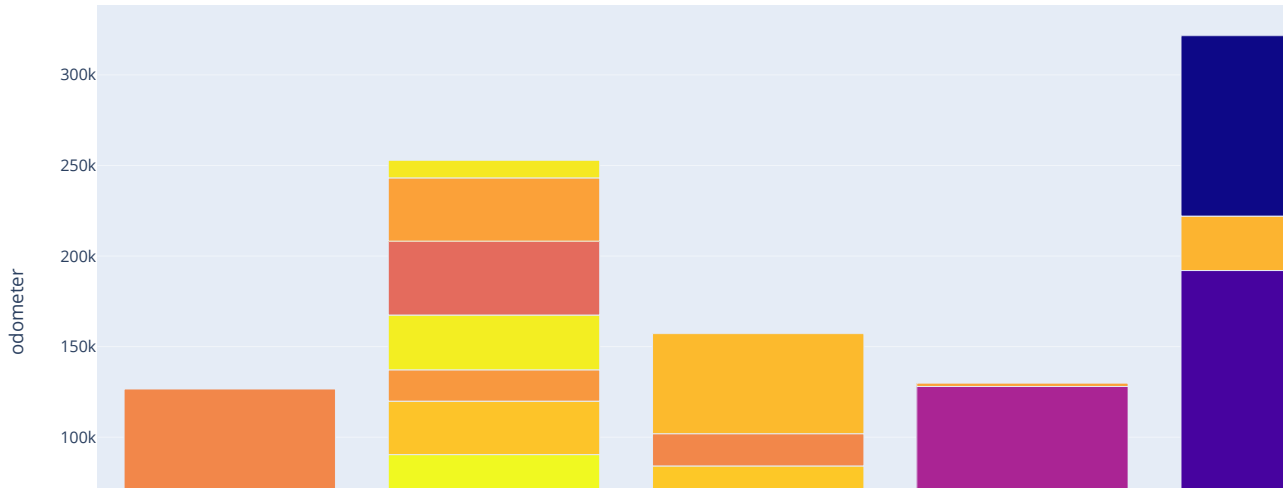


The above pie chart provides an information of the distribution of used car prices across different condition categories. Insights from the chart are that fair and good condition cars command the largest shares of the market at 21% and 20% respectively. This suggests us that huge amount of used car stock falls into the mid-tier condition categories, while excellent condition is less.

In [29]:
```
import plotly.express as px
```

Price Variation Based on Odometer Readings

In [30]:
```
fig = px.bar(df.head(20),
             x='manufacturer',
             y='odometer',
             color='price',
             hover_name="manufacturer",
             hover_data=["region", "model","fuel"],
             height=600)
fig.show()
```



From the above interactive plot jeep has the lowest price as it has highest selling odometers and chevrolet has the highest prices as it has less odometer range. Thus, we can conclude that the higher the odometer range the lower the price and vice versa.

In [31]:
```
# fig = px.bar(df.head(20),
#              x='manufacturer',
#              y='price',
#              color='odometer',
#              hover_name="manufacturer",
#              hover_data=["region", "model","fuel"],
#              height=600)
# fig.show()
```

In [32]:
```
# fig = px.bar(df.head(20),
#              x='odometer',  # X-axis: Odometer
#              y='price',     # Y-axis: Price
#              color='manufacturer',  # Color by manufacturer
#              hover_name="manufacturer",
#              hover_data=["region", "model", "fuel"],
#              height=600)
# fig.show()
```

In [33]:
```
# fig = px.bar(df.head(20),
#              x='manufacturer',
#              y='price',
#              color='fuel',  # Group bars by fuel type
#              hover_name="manufacturer",
#              hover_data=["region", "model", "fuel"],
#              height=600)
# fig.show()
```

In [34]:
```
# fig = px.bar(df.head(20),
#              x='condition',
#              y='odometer',
#              color='manufacturer',
#              hover_name="manufacturer",
#              hover_data=["region", "model","fuel","price"],
#              height=600)
# fig.show()
```

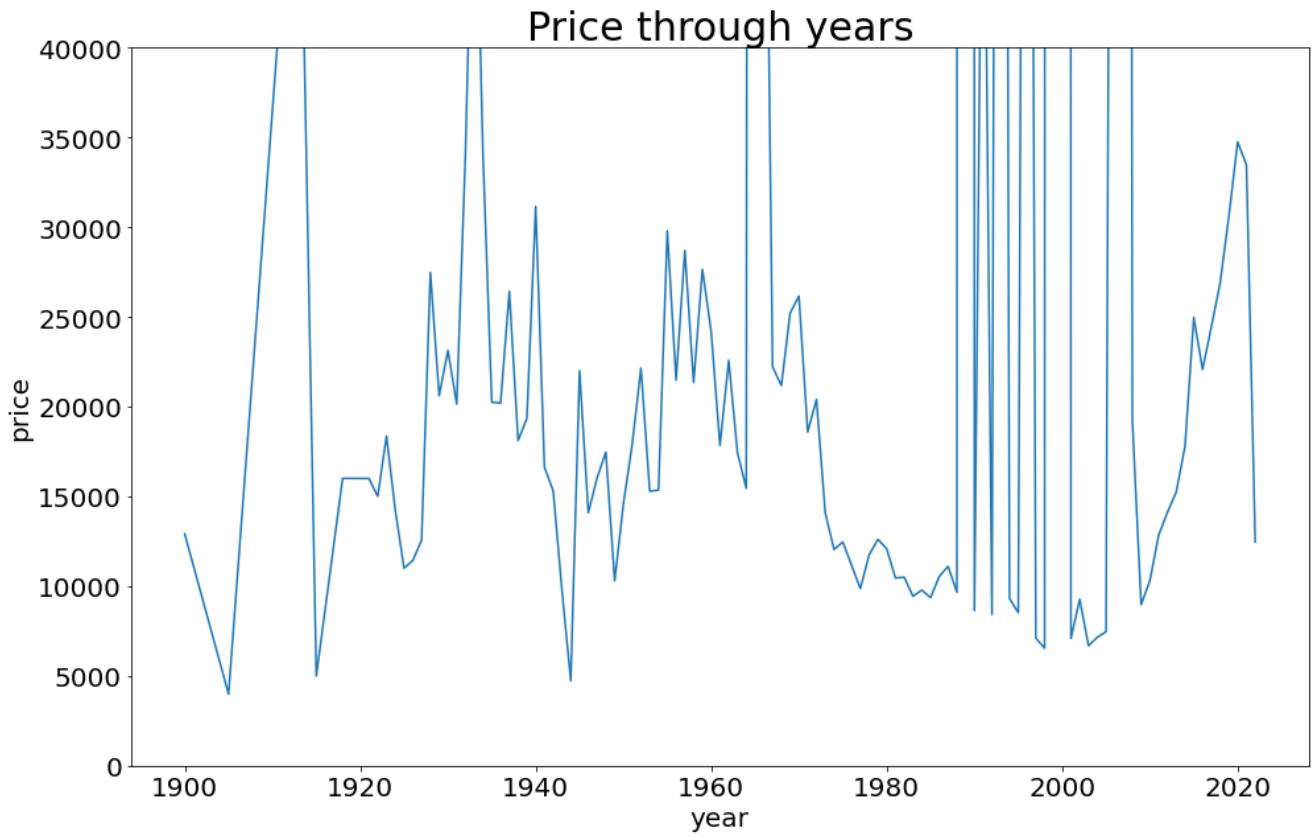
Price Trends of Cars by Year

In [35]:
```
#checking the percentage of car prices above 40k
total_cars = len(df)
count_of_expensive_cars = len(df[df["price"] > 40000])
percentage_expensive_cars = (count_of_expensive_cars / total_cars) * 100
percentage_expensive_cars
```

Out[35]:
```
6.593835003499818
```

In [36]:
```
fig = plt.figure(figsize=(16,10))
axes = fig.add_subplot(111)
# Calculating the mean prices of cars grouped by year
df1 = df.groupby('year').mean()['price']
# Creating a line plot for the prices over the years
df1.plot(kind='line', title='Price through years', fontsize=20)
plt.ylabel('price')
axes.title.set_fontsize(30)
axes.xaxis.label.set_fontsize(20)
axes.yaxis.label.set_fontsize(20)

#setting the Y limit max to 40k as above them are very less for better understanfing
axes.set_ylim(0, 40000)
```

Out[36]:
```
(0.0, 40000.0)
```

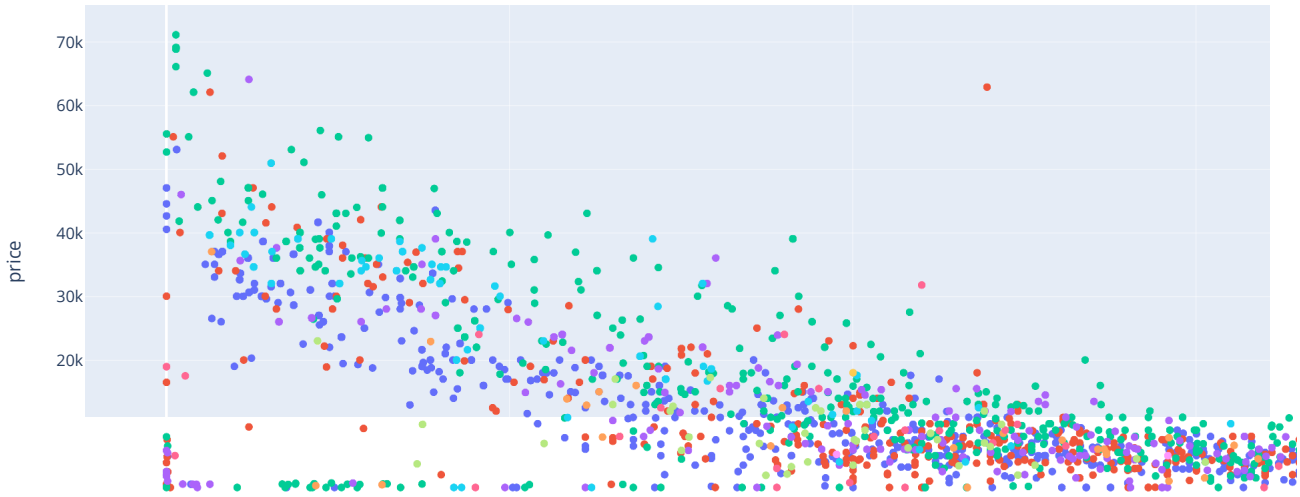


The line chart shows the average price of used cars grouped by the year of manufacture. From the above plot we can say that from the 1990s to the early 2010s, people were purchasing used cars at relatively high prices. This suggests that during this period, older used cars were valued quite highly, possibly due to factors such as limited availability, demand for specific models, or collector's value.

After the early 2010s, there is a noticeable decrease in the average price of used cars and then there is a rise in the cars price at covid time.

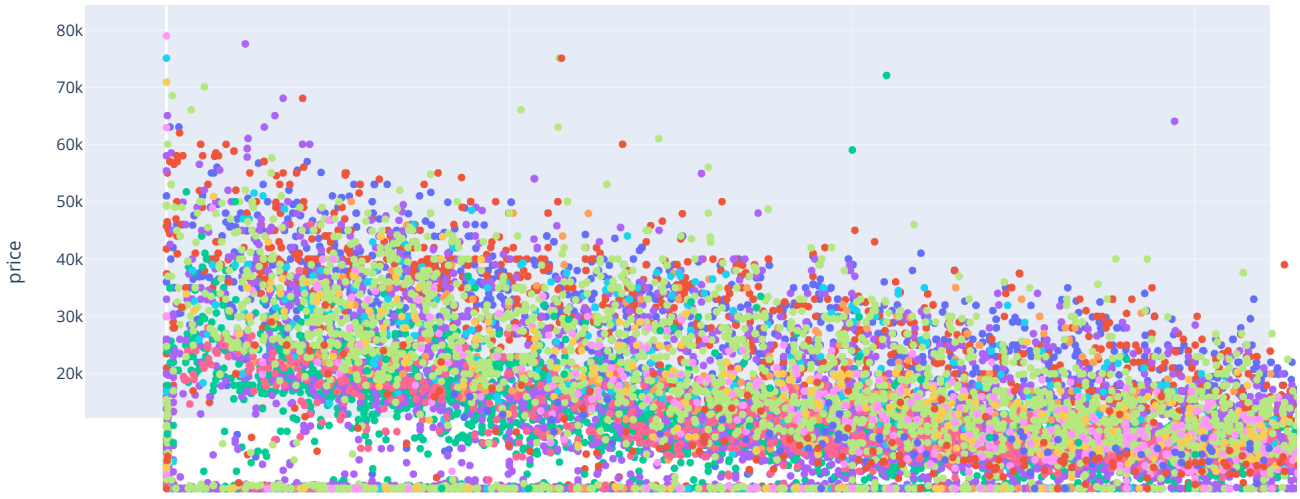
Cars Price vs. Odometer Reading by Car Type

In [37]:
```
tempdf = df[df['manufacturer'] == 'volvo']
tempdf = tempdf[tempdf['price'] < 80000]
tempdf = tempdf[tempdf['odometer'] < 200000]
px.scatter(tempdf, x='odometer', y='price', color='type', hover_name = 'model')
```



From the above interactive scatter plot we can conclude that most people who are buying Volvo prefer SUV and buy them for higher prices and next comes the sedan.

In [38]:
```
tempdf = df[df['manufacturer'] == 'toyota']
tempdf = tempdf[tempdf['price'] < 80000]
tempdf = tempdf[tempdf['odometer'] < 200000]
px.scatter(tempdf, x='odometer', y='price', color='type', hover_name = 'model')
```



Here for Toyota people tend to buy with more price for truck but the higher number of people are preferring SUV over others.

We have done analysis of our target variable Price based on our features(condition, odometer, type, year, fuel)

In [ ]: