In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import PolynomialFeatures
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import matplotlib
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

from scipy import stats

# Core scikit learn cross validation tools also
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.model_selection import ShuffleSplit
import matplotlib.cm as cm

from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split, GridSearchCV, cr
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score

import warnings
warnings.filterwarnings('ignore')
```

Problem Statement

We are analyzing data of the entire second-hand car market across different states in the United States.We analyze the situation of used car sales on cars.com across different states in the United States and prices based on different features of used cars. We used different machine learning models to figure out the best model for predicting price.

Hypotheses

1. The sales and prices of used cars across states in the US may be influenced by multiple factors, including year, mileage and make as main factors.
2. More developed states, such as California and New York, may have higher sales and prices for used cars, while states with smaller populations or weaker economies, such as Wyoming and South Dakota, may have lower sales and prices for used cars.
3. There may be a correlation between the sales and prices of used cars, i.e., make with higher sales may have lower average prices, while make with lower sales may have higher average prices.

About Data

This data was scraped from postings on a popular online auto marketplace. The dataset contains information from each posting, including features such as the make, model, year, exterior color, drivetrain, etc.

Column Info:

- Year: Model year
- Make: Brand of the car
- Model: Car model
- Used/New: Whether the car being sold is used, new, or certified.
- Price: Listing price
- Consumer Rating: Average consumer rating based on submitted consumer reviews
- Consumer Reviews: The number of reviews submitted for the car in the listing
- SellerType: Whether the seller is a dealer or private
- SellerName: Name of the seller
- StreetName: Name of the street where the seller is located
- State: State of the seller's location
- Zipcode: Zipcode of the seller's location
- DealType: How good is the deal based on the average market price for the car in the listing? (Great, Good, Fair, NA)
- ComfortRating: How consumers rated the car's comfort
- InteriorDesignRating: How consumers rated the car's interior design
- PerformanceRating: How consumers rated the car's performance
- ValueForMoneyRating: How consumers rated the car's value for the price
- ExteriorStylingRating: How consumers rated the car's exterior styling
- ReliabilityRating: How consumers rated the car's reliability
- ExteriorColor: The car's exterior color
- InteriorColor: The car's interior color
- Drivetrain: The drivetrain type of the car
- MinMPG: Bottom end miles per gallon
- MaxMPG: Top end miles per gallon
- FuelType: Type of fuel that the car uses. (Gas, Electric, Hybrid, etc.)
- Transmission: Type of transmission
- Engine: Name of the engine
- VIN: VIN Number
- Stock# : The listing's stock number
- Mileage: Number of miles on the car

Data Preparing

In [24]:
```
usedcar = pd.read_csv('cars_raw.csv')
usedcar.head()
```

Out[24]:

	Year	Make	Model	Used/New	Price	ConsumerRating	ConsumerReviews	SellerType
0	2019	Toyota	Sienna SE	Used	$39,998	4.6	45	Dealer
1	2018	Ford	F-150 Lariat	Used	$49,985	4.8	817	Dealer
2	2017	RAM	1500 Laramie	Used	$41,860	4.7	495	Dealer
3	2021	Honda	Accord Sport SE	Used	$28,500	5.0	36	Dealer
4	2020	Lexus	RX 350	Used	$49,000	4.8	76	Dealer

5 rows × 32 columns

In [25]:
```
usedcar.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9379 entries, 0 to 9378
Data columns (total 32 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Year                   9379 non-null   int64  
 1   Make                   9379 non-null   object 
 2   Model                  9379 non-null   object 
 3   Used/New               9379 non-null   object 
 4   Price                  9379 non-null   object 
 5   ConsumerRating         9379 non-null   float64
 6   ConsumerReviews        9379 non-null   int64  
 7   SellerType             9379 non-null   object 
 8   SellerName             9379 non-null   object 
 9   SellerRating           9379 non-null   float64
 10  SellerReviews          9379 non-null   int64  
 11  StreetName             9379 non-null   object 
 12  State                  9379 non-null   object 
 13  Zipcode                9379 non-null   object 
 14  DealType               9157 non-null   object 
 15  ComfortRating          9379 non-null   float64
 16  InteriorDesignRating   9379 non-null   float64
 17  PerformanceRating      9379 non-null   float64
 18  ValueForMoneyRating    9379 non-null   float64
 19  ExteriorStylingRating  9379 non-null   float64
 20  ReliabilityRating      9379 non-null   float64
 21  ExteriorColor          9379 non-null   object 
 22  InteriorColor          9379 non-null   object 
 23  Drivetrain             9379 non-null   object 
 24  MinMPG                 9379 non-null   int64  
 25  MaxMPG                 9379 non-null   int64  
 26  FuelType               9379 non-null   object 
 27  Transmission           9379 non-null   object 
 28  Engine                 9379 non-null   object 
 29  VIN                    9379 non-null   object 
 30  Stock#                 9379 non-null   object 
 31  Mileage                9379 non-null   int64  
dtypes: float64(8), int64(6), object(18)
memory usage: 2.3+ MB
```

In [26]:
```
usedcar.describe()
```

Out[26]:

	Year	ConsumerRating	ConsumerReviews	SellerRating	SellerReviews	ComfortF
count	9379.000000	9379.000000	9379.000000	9379.000000	9379.000000	9379.0
mean	2018.721719	4.702825	133.187014	4.412571	984.089988	4.7
std	2.221708	0.240795	154.985640	0.626258	1609.039864	0.2
min	2001.000000	2.500000	1.000000	1.000000	1.000000	3.0
25%	2018.000000	4.700000	30.000000	4.300000	112.000000	4.7
50%	2019.000000	4.800000	75.000000	4.600000	542.000000	4.8
75%	2020.000000	4.800000	182.000000	4.800000	1272.000000	4.9
max	2022.000000	5.000000	817.000000	5.000000	27824.000000	5.0

In [27]:
```
usedcar.duplicated().sum()
```

Out[27]: 872

In [28]:
```
# Drop duplicates
usedcar.drop_duplicates(inplace=True)
```

In [29]:
```
usedcar.shape
```

Out[29]: (8507, 32)

In [30]:
```
usedcar.isnull().sum()
```

Out[30]:
```
Year                      0
Make                      0
Model                     0
Used/New                  0
Price                     0
ConsumerRating            0
ConsumerReviews           0
SellerType                0
SellerName                0
SellerRating              0
SellerReviews             0
StreetName                0
State                     0
Zipcode                   0
DealType                206
ComfortRating             0
InteriorDesignRating      0
PerformanceRating         0
ValueForMoneyRating       0
ExteriorStylingRating     0
ReliabilityRating         0
ExteriorColor             0
InteriorColor             0
Drivetrain                0
MinMPG                    0
MaxMPG                    0
FuelType                  0
Transmission              0
Engine                    0
VIN                       0
Stock#                    0
Mileage                   0
dtype: int64
```

Since only about 2.6% of data is missing, so we can drop them.

In [31]:
```
# Drop mull value
usedcar.dropna(inplace=True)
usedcar.isna().sum()
```

Out[31]:
```
Year                     0
Make                     0
Model                    0
Used/New                 0
Price                    0
ConsumerRating           0
ConsumerReviews          0
SellerType               0
SellerName               0
SellerRating             0
SellerReviews            0
StreetName               0
State                    0
Zipcode                  0
DealType                 0
ComfortRating            0
InteriorDesignRating     0
PerformanceRating        0
ValueForMoneyRating      0
ExteriorStylingRating    0
ReliabilityRating        0
ExteriorColor            0
InteriorColor            0
Drivetrain               0
MinMPG                   0
MaxMPG                   0
FuelType                 0
Transmission             0
Engine                   0
VIN                      0
Stock#                   0
Mileage                  0
dtype: int64
```

In [32]:
```
# Drop $ of Price and change to float
usedcar.drop(usedcar[usedcar["Price"]=="Not Priced"].index, inplace=Tr
usedcar["Price"]= usedcar["Price"].str.replace(',','.')
usedcar["Price"] = usedcar["Price"].str[1:]
usedcar["Price"] = usedcar["Price"].astype(float)
usedcar.head()
```

Out[32]:

	Year	Make	Model	Used/New	Price	ConsumerRating	ConsumerReviews	SellerType
0	2019	Toyota	Sienna SE	Used	39.998	4.6	45	Dealer
1	2018	Ford	F-150 Lariat	Used	49.985	4.8	817	Dealer
2	2017	RAM	1500 Laramie	Used	41.860	4.7	495	Dealer
4	2020	Lexus	RX 350	Used	49.000	4.8	76	Dealer
5	2012	Toyota	4Runner SR5	Used	23.541	4.7	34	Dealer

5 rows × 32 columns

In [33]:
```
# check unique value of Used/New
usedcar['Used/New'].unique()
```

Out[33]:
```
array(['Used', 'Dodge Certified', 'Acura Certified', 'Honda Certified
',
       'Mercedes-Benz Certified', 'Ford Certified', 'Toyota Certified
',
       'BMW Certified', 'Porsche Certified', 'Cadillac Certified',
       'Volvo Certified', 'Nissan Certified', 'Subaru Certified',
       'Volkswagen Certified', 'INFINITI Certified',
       'Chevrolet Certified', 'Kia Certified', 'RAM Certified',
       'Jeep Certified', 'GMC Certified', 'Buick Certified',
       'Alfa Romeo Certified', 'MINI Certified', 'Maserati Certified'
],
      dtype=object)
```

In [34]:
```
# Standardize the categories of 'used' and 'certified'
replace_list = usedcar["Used/New"].unique()
replace_list = replace_list[replace_list != 'Used']
for i in replace_list:
    usedcar["Used/New"].replace({i : 'Certified'}, inplace=True)
usedcar["Used/New"].unique()
```

Out[34]:
```
array(['Used', 'Certified'], dtype=object)
```

In [35]:
```
# Check unique value of Drivetrain
usedcar["Drivetrain"].unique()
```

Out[35]:
```
array(['Front-wheel Drive', 'Four-wheel Drive', 'Rear-wheel Drive',
       'All-wheel Drive', '4WD', 'AWD', 'RWD', 'FWD', 'Front Wheel Dr
ive',
       '–'], dtype=object)
```

In [36]:
```
# Change Drivetrain categories
usedcar["Drivetrain"].replace({'Front-wheel Drive' : 'FWD'}, inplace=T
usedcar["Drivetrain"].replace({'Four-wheel Drive' : '4WD'}, inplace=Tr
usedcar["Drivetrain"].replace({'Rear-wheel Drive' : 'RWD'}, inplace=Tr
usedcar["Drivetrain"].replace({'All-wheel Drive' : 'AWD'}, inplace=Tru
usedcar["Drivetrain"].replace({'Front Wheel Drive' : 'FWD'}, inplace=T
usedcar["Drivetrain"].unique()
```

Out[36]:
```
array(['FWD', '4WD', 'RWD', 'AWD', '–'], dtype=object)
```

In [37]:
```
# Check number of rows of Drivetrain with '–'
len(usedcar[usedcar['Drivetrain']=='–'])
```

Out[37]:
```
6
```

In [38]:
```
# Only 6 rows are "–" so we can drop them
usedcar.drop(usedcar[usedcar['Drivetrain'] == '–'].index, inplace=True
usedcar["Drivetrain"].unique()
```

Out[38]:
```
array(['FWD', '4WD', 'RWD', 'AWD'], dtype=object)
```

In [39]:
```
# Check unique value of State
usedcar['State'].unique()
```

Out[39]:
```
array(['CA', 'NV', 'AZ', 'UT', 'WA', 'ID', 'TX', 'NE', 'KS', 'MN', 'W
I',
       'MO', 'LA', 'IL', 'TN', 'IN', 'GA', 'OH', 'SC', 'FL', 'VA', 'P
A',
       'NJ', 'NY', 'MA', 'OR', 'CO', 'OK', 'AR', 'MI', 'NC', 'MD', 'D
E',
       'NH', 'SD', 'AL', 'KY', 'VT', 'IA', 'CT', 'MS', 'RI', 'HI', 'R
T',
       'ND', 'Michigan', 'WV', 'Bldg', 'NM', 'ME', 'AZ–101', 'US–12',
       'WY', 'MT', 'Glens', 'Suite', 'SE', 'AK', 'US–169'], dtype=obj
ect)
```

Among all unique data in State, 'RT', 'Michigan', 'Bldg', 'AZ-101', 'US-12', 'Glens', 'Suite', 'SE', and 'US-169' are not name of a state.

In [40]:
```
# Change name of state
usedcar["State"].replace({'Michigan' : 'MI'}, inplace=True)
usedcar["State"].replace({'AZ-101' : 'AZ'}, inplace=True)

# Check zipcode of the wrong state
St_list=['RT', 'Bldg', 'US-12', 'Glens', 'Suite', 'SE', 'US-169' ]
for st in St_list:
    print(usedcar.loc[usedcar['State'] == st, ['Zipcode', 'State']])
```

```
     Zipcode State
641        1    RT
733        1    RT
1976       1    RT
2816       1    RT
3479       1    RT
4245       1    RT
4523       1    RT
6372       1    RT
8169       1    RT
8170       1    RT
     Zipcode State
1424       B  Bldg
3034       B  Bldg
     Zipcode  State
5126     Fox  US-12
     Zipcode  State
6269   Falls  Glens
     Zipcode  State
6579     102  Suite
       Zipcode State
7003  Leesburg    SE
         Zipcode   State
8110  Smithville  US-169
```

Since we don't have effect zipcode for reference and there are not many rows, we can drop them directly.

In [41]:
```
usedcar = usedcar[~usedcar['State'].isin(St_list)]
usedcar['State'].unique()
```

Out[41]:
```
array(['CA', 'NV', 'AZ', 'UT', 'WA', 'ID', 'TX', 'NE', 'KS', 'MN', 'W
I',
       'MO', 'LA', 'IL', 'TN', 'IN', 'GA', 'OH', 'SC', 'FL', 'VA', 'P
A',
       'NJ', 'NY', 'MA', 'OR', 'CO', 'OK', 'AR', 'MI', 'NC', 'MD', 'D
E',
       'NH', 'SD', 'AL', 'KY', 'VT', 'IA', 'CT', 'MS', 'RI', 'HI', 'N
D',
       'WV', 'NM', 'ME', 'WY', 'MT', 'AK'], dtype=object)
```

We will drop zipcode and street name as there are too many unique values and we can use state to do geographic analysis.

In [42]:
```
usedcar = usedcar.drop('Zipcode', axis=1)
usedcar = usedcar.drop('StreetName', axis=1)
```

In [43]:
```
# drop SellerName and Stock# as they don't have strong relationship wi
usedcar = usedcar.drop('SellerName', axis=1)
usedcar = usedcar.drop('Stock#', axis=1)
```

In [44]:
```
# Check value counts of unique seller type
usedcar['SellerType'].value_counts()
```

Out[44]:
```
Dealer     8244
Private      30
Name: SellerType, dtype: int64
```

The majority of the seller types are dealers, making it difficult to analyze the influence of dealers and private sellers on the price. Therefore, we will drop the 'SellerType' column and assume that both dealer and private seller types contribute equally to the price.

In [45]:
```
# Drop SellerType column
usedcar = usedcar.drop('SellerType', axis=1)
```

In [46]:
```
#Check unique values of transmission
usedcar['Transmission'].nunique()
```

Out[46]:
```
86
```

In [47]:
```
#Check unique values of Engine
usedcar['Engine'].nunique()
```

Out[47]: 302

In [48]:
```
# Drop Engine
usedcar = usedcar.drop('Engine', axis=1)
# Drop Vin since it's only the Identification Number and not related t
usedcar = usedcar.drop('VIN', axis=1)
```

In [49]:
```
usedcar['FuelType'].unique()
```

Out[49]:
```
array(['Gasoline', 'Gasoline Fuel', 'Electric Fuel System',
       'E85 Flex Fuel', 'Electric', 'Hybrid', '–', 'Flex Fuel Capabil
ity',
       'Diesel', 'Gasoline/Mild Electric Hybrid', 'Flexible Fuel'],
      dtype=object)
```

In [50]:
```
usedcar["FuelType"].replace({'Gasoline Fuel' : 'Gasoline'}, inplace=Tr
usedcar["FuelType"].replace({'Electric Fuel System' : 'Electric'}, inp
usedcar["FuelType"].replace({'E85 Flex Fuel' : 'Flexible'}, inplace=Tr
usedcar["FuelType"].replace({'Flex Fuel Capability' : 'Flexible'}, inp
usedcar["FuelType"].replace({'Flexible Fuel' : 'Flexible'}, inplace=Tr
usedcar["FuelType"].replace({'Gasoline/Mild Electric Hybrid' : 'Hybric
```

In [51]:
```
usedcar["FuelType"].value_counts()['–']
```

Out[51]: 29

In [52]:
```
usedcar.drop(usedcar[usedcar['FuelType'] == '–'].index, inplace=True)
usedcar['FuelType'].unique()
```

Out[52]:
```
array(['Gasoline', 'Electric', 'Flexible', 'Hybrid', 'Diesel'],
      dtype=object)
```

In [53]:
```
usedcar["ExteriorColor"].nunique()
```

Out[53]: 924

In [54]:
```
usedcar["InteriorColor"].nunique()
```

Out[54]: 357

There are too many types of color so it's pretty hard analyze the relationship between price and color.

In [55]:
```
# Drop color
usedcar = usedcar.drop('InteriorColor', axis=1)
usedcar = usedcar.drop('ExteriorColor', axis=1)
```

In [56]:
```
#Check unique value of DealType
usedcar["DealType"].value_counts()
```

Out[56]:
```
Good     4988
Great    2158
Fair     1099
Name: DealType, dtype: int64
```

In [57]:
```
usedcar.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8245 entries, 0 to 9378
Data columns (total 23 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Year                   8245 non-null   int64  
 1   Make                   8245 non-null   object 
 2   Model                  8245 non-null   object 
 3   Used/New               8245 non-null   object 
 4   Price                  8245 non-null   float64
 5   ConsumerRating         8245 non-null   float64
 6   ConsumerReviews        8245 non-null   int64  
 7   SellerRating           8245 non-null   float64
 8   SellerReviews          8245 non-null   int64  
 9   State                  8245 non-null   object 
 10  DealType               8245 non-null   object 
 11  ComfortRating          8245 non-null   float64
 12  InteriorDesignRating   8245 non-null   float64
 13  PerformanceRating      8245 non-null   float64
 14  ValueForMoneyRating    8245 non-null   float64
 15  ExteriorStylingRating  8245 non-null   float64
 16  ReliabilityRating      8245 non-null   float64
 17  Drivetrain             8245 non-null   object 
 18  MinMPG                 8245 non-null   int64  
 19  MaxMPG                 8245 non-null   int64  
 20  FuelType               8245 non-null   object 
 21  Transmission           8245 non-null   object 
 22  Mileage                8245 non-null   int64  
dtypes: float64(9), int64(6), object(8)
memory usage: 1.5+ MB
```

In [58]:
```
#Remove Outlier

numeric_cols = usedcar.select_dtypes(include=['number']).columns.tolis

# Use IQR remove outlier
for col in numeric_cols:
    q1 = usedcar[col].quantile(0.25)
    q3 = usedcar[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    usedcar = usedcar.loc[(usedcar[col] >= lower_bound) & (usedcar[col
```

Data Visualization

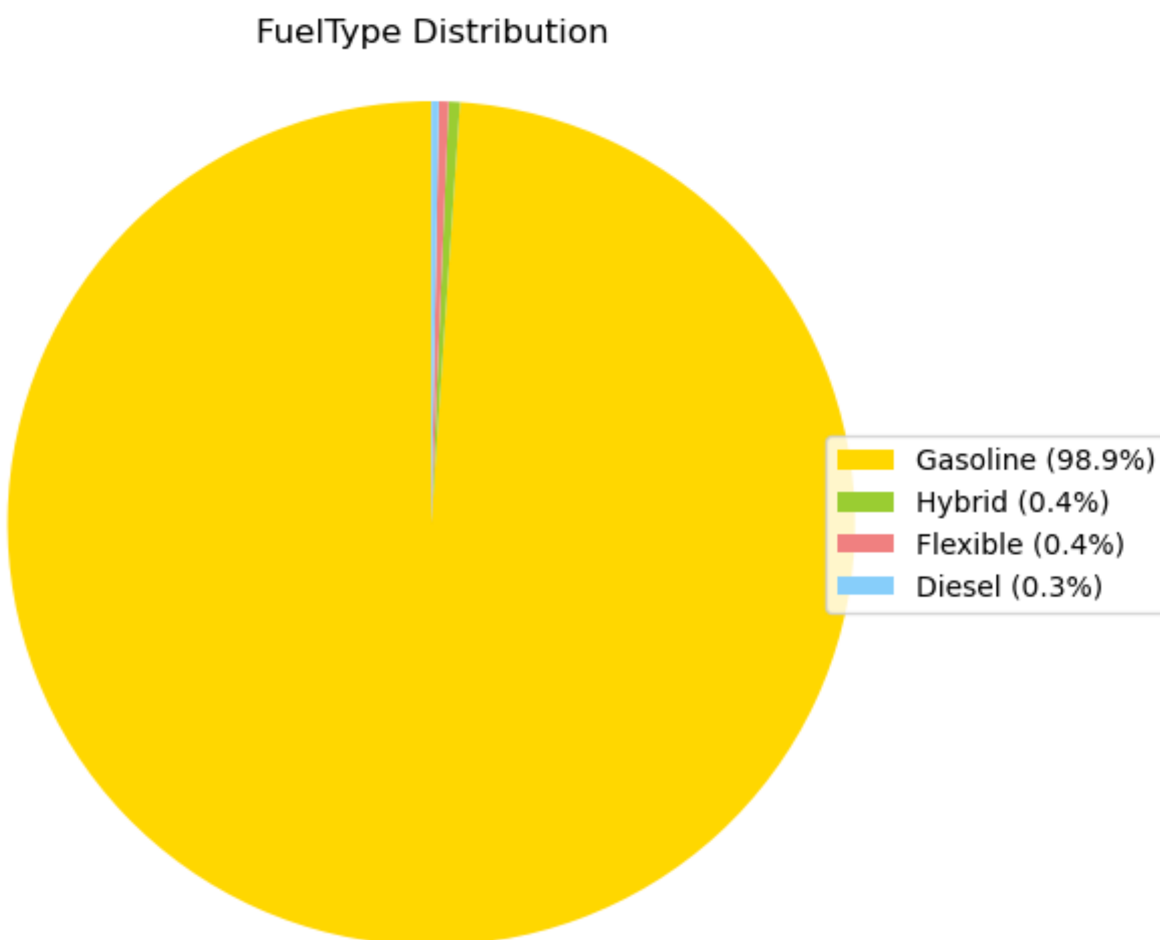
Sales Distribution

We believe that the sales of used cars may be related to geographical factors and that consumers have some preferences for car makes, fuel type, drivetrain, etc. In this part, we want to map the car sale amount vs geograhical location

In [95]:
```
import plotly.express as px

# Aggregate car sales by state
car_sales_by_state = usedcar.groupby('State').size().reset_index(name=

# Create the choropleth map using Plotly
fig = px.choropleth(car_sales_by_state,
                    locations='State',  # Column containing the state a
                    locationmode='USA-states',  # Specify the location
                    color='CarSales',  # Column containing the car sale
                    color_continuous_scale='pinkyl',
                    scope='usa',  # Limit the map scope to the USA
                    labels={'CarSales':'Car Sales'},
                    title='Car Sales by State')

fig.show()
```

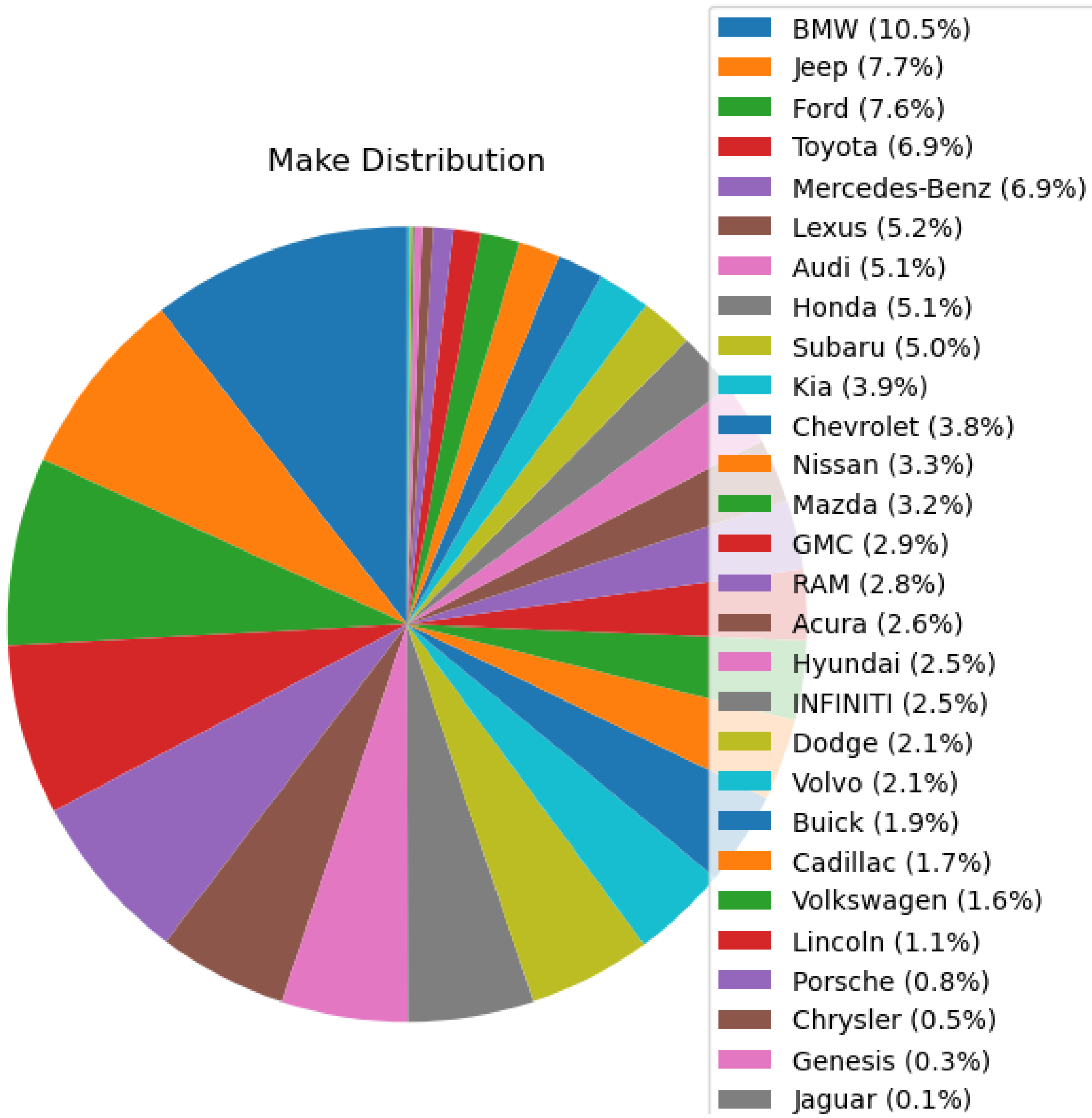
It shows the car sales figures by each U.S state in different color scales, the deeper the color, the higher the sales figure. From the graph, we can see that Texas, California and Florida are the states with the highest sales figures.

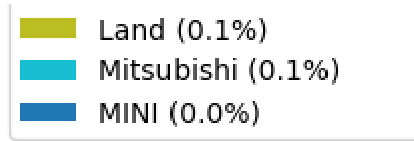
In [60]:
```
fueltype_counts = usedcar['FuelType'].value_counts()
labels = usedcar['FuelType'].value_counts().index.tolist()
sizes = fueltype_counts.values.tolist()
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'orange

fig, ax = plt.subplots(figsize=(6, 6))
ax.pie(sizes, colors=colors, startangle=90, pctdistance=0.8)
legend_labels = [f'{label} ({size/sum(sizes)*100:.1f}%)' for label, si
plt.legend(legend_labels, loc='right', bbox_to_anchor=(1.3, 0.5))
ax.axis('equal')
ax.set_title('FuelType Distribution')

plt.show()
```



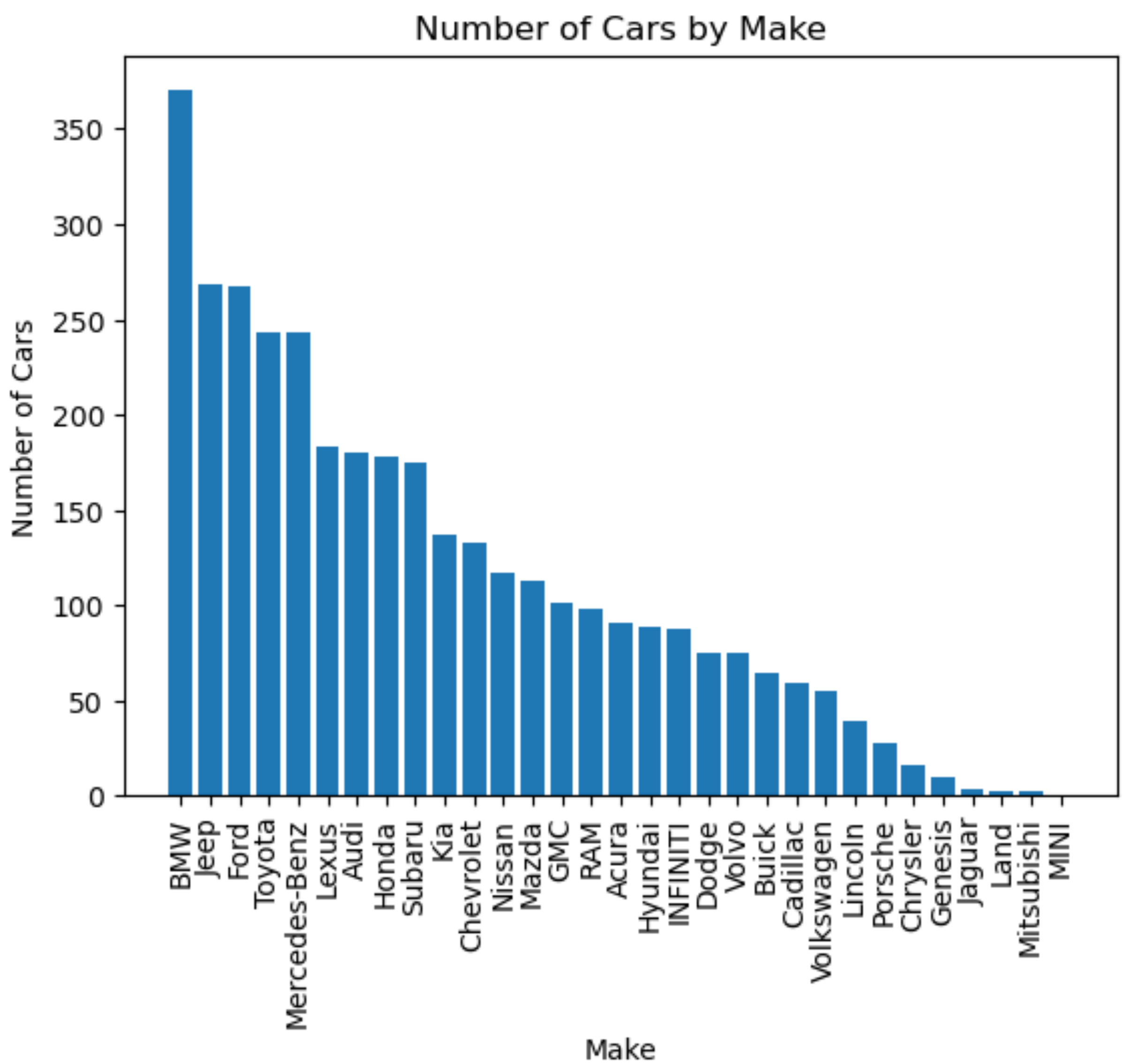
We could see that the majority of cars are gasoline fuel in US usedcar market. This may because gasoline is affordable and well-established.

In [61]:
```
Make_counts = usedcar['Make'].value_counts()
labels = usedcar['Make'].value_counts().index.tolist()
sizes = Make_counts.values.tolist()

fig, ax = plt.subplots(figsize=(6, 6))
ax.pie(sizes, startangle=90, pctdistance=0.8)
legend_labels = [f'{label} ({size/sum(sizes)*100:.1f}%)' for label, si
plt.legend(legend_labels, loc='right', bbox_to_anchor=(1.3, 0.5))
ax.axis('equal')
ax.set_title('Make Distribution')

plt.show()
```



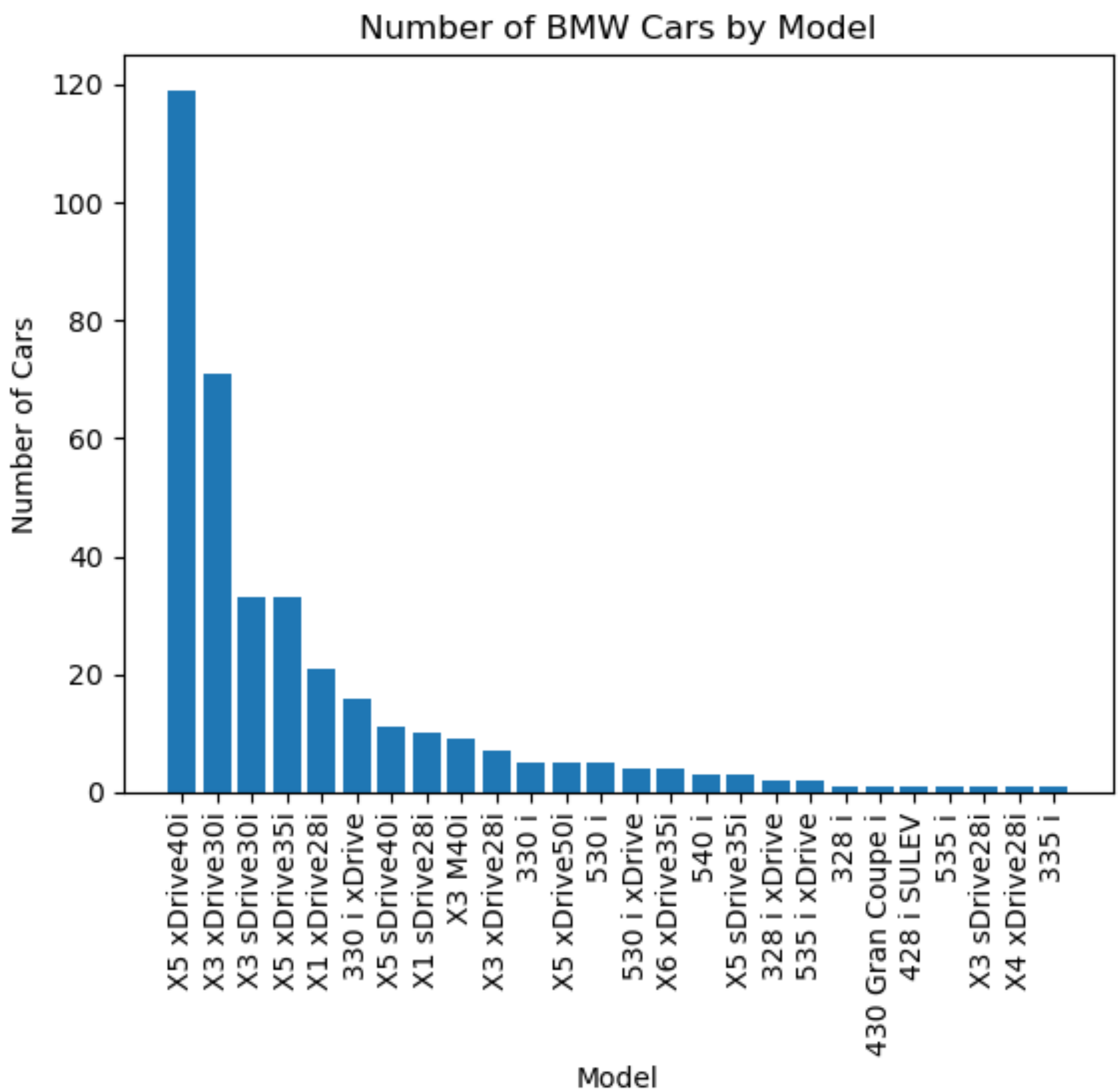


In [62]:
```
counts = usedcar['Make'].value_counts()
counts = counts.sort_values(ascending=False)

# Plot a bar chart of the counts
plt.bar(counts.index, counts.values)
plt.title('Number of Cars by Make')
plt.xlabel('Make')
plt.ylabel('Number of Cars')
plt.xticks(rotation=90)
plt.show()
```



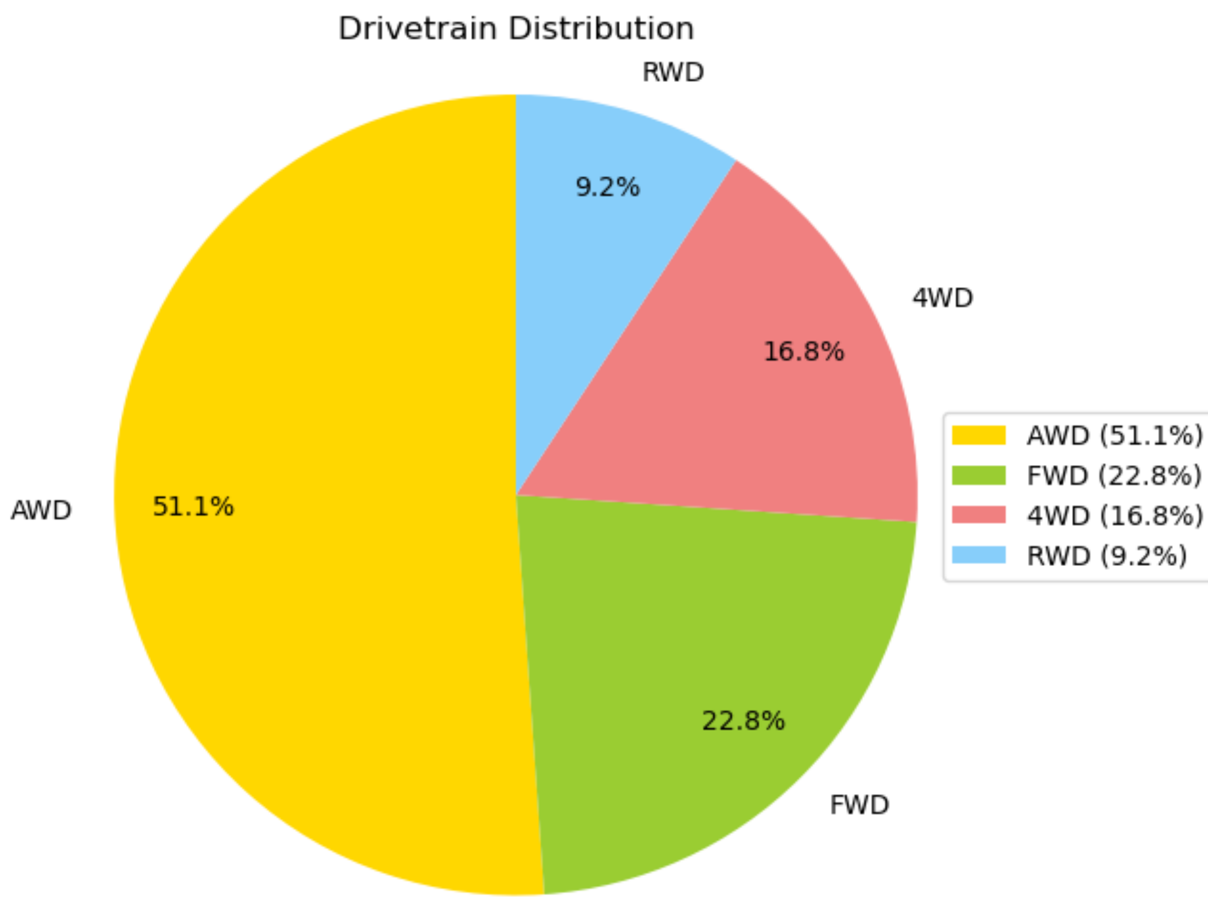
We can see that German cars, American cars and Japanese cars in the United States used car market sales of the best, where BMW sales are the highest, with a 10% market share.

In [63]:
```
bmw_cars = usedcar[usedcar['Make'] == 'BMW']
counts = bmw_cars['Model'].value_counts()
counts = counts.sort_values(ascending=False)

# Plot a bar chart of the counts
plt.bar(counts.index, counts.values)
plt.title('Number of BMW Cars by Model')
plt.xlabel('Model')
plt.ylabel('Number of Cars')
plt.xticks(rotation=90)
plt.show()
```



X5 is the most popular model of BMW in US usedcar market.

In [64]:
```
Drivetrain_counts = usedcar['Drivetrain'].value_counts()
labels = usedcar['Drivetrain'].value_counts().index.tolist()
sizes = Drivetrain_counts.values.tolist()
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'orange

fig, ax = plt.subplots(figsize=(6, 6))
ax.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',startang
legend_labels = [f'{label} ({size/sum(sizes)*100:.1f}%)' for label, si
plt.legend(legend_labels, loc='right', bbox_to_anchor=(1.3, 0.5))
ax.axis('equal')
ax.set_title('Drivetrain Distribution')

plt.show()
```



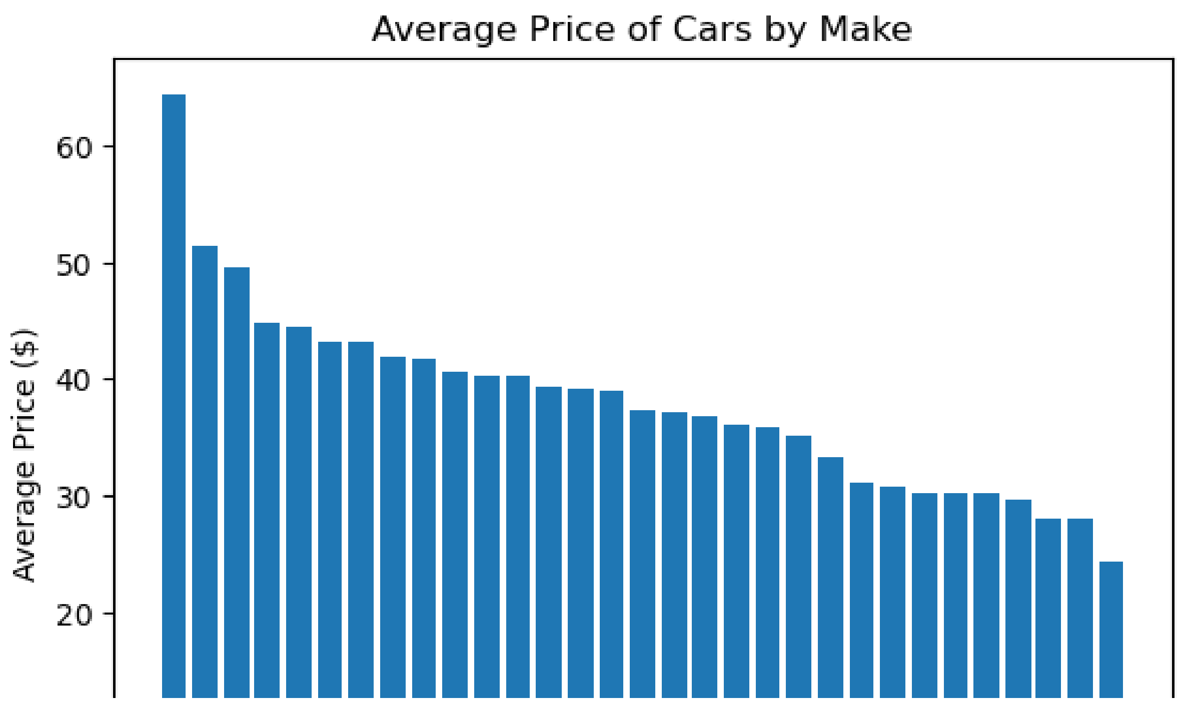
It would appear that all-wheel drive and front-wheel drive composite about 70% of the used car market. All-wheel drive and front-wheel drive are more popular in the US used car market for several reasons:

1. Front-wheel drive and all-wheel drive vehicles generally get better gas mileage than rear-wheel drive and 4-wheel drive vehicles. This is because they have fewer moving parts and less weight in the drivetrain, which can reduce the amount of power needed to move the car.
2. All-wheel drive and front-wheel drive vehicles are typically less expensive to produce than rear-wheel drive and 4-wheel drive vehicles. As a result, they are often more affordable in the used car market.
3. In areas with inclement weather, all-wheel drive and front-wheel drive vehicles offer better traction and stability, while rear-wheel drive and 4-wheel drive vehicles can be more challenging to handle in snowy or wet conditions.
4. Front-wheel drive vehicles are easier to handle in urban driving conditions because they have a smaller turning radius and are generally more maneuverable. All-wheel drive vehicles offer better handling on winding roads and at high speeds.

Price Analysis

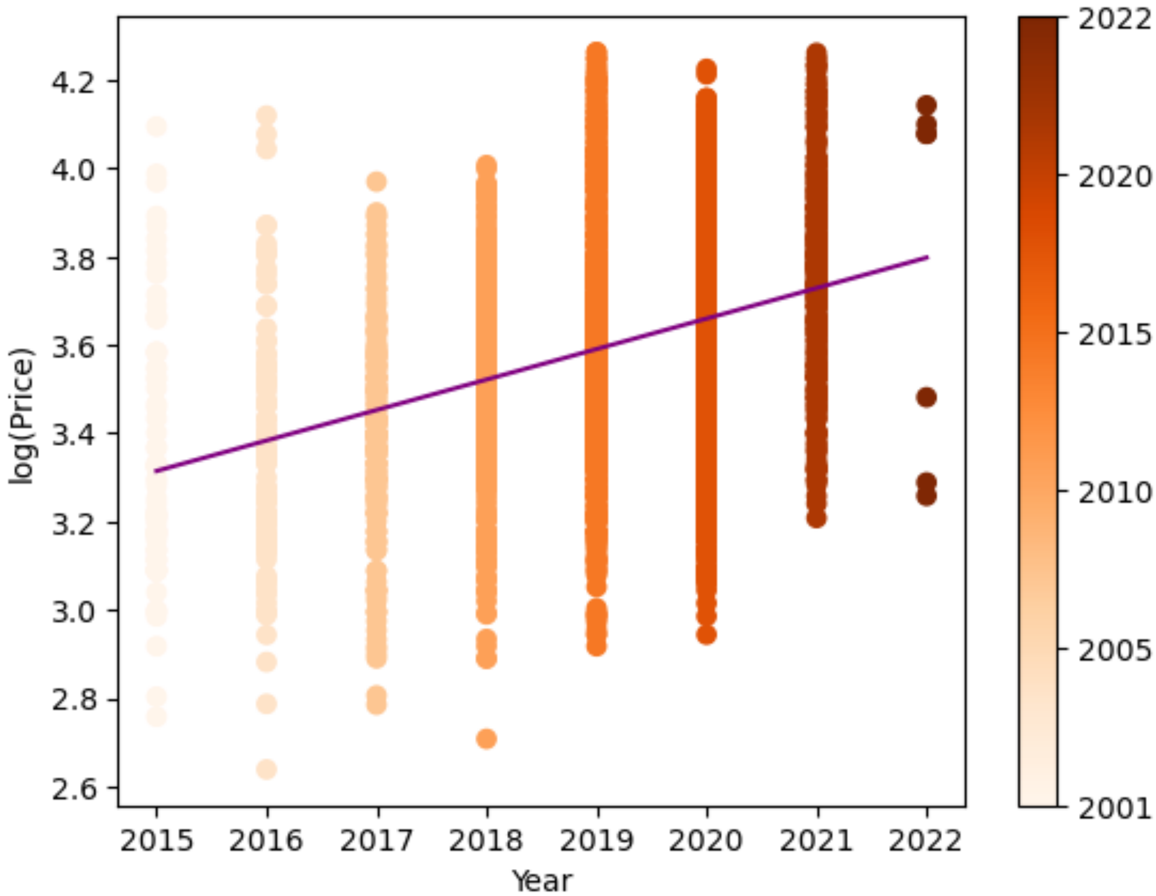
We assume the main factors that influence price are make, year and mileage. First we want to plot the data to visualize insights of this.

In [65]:
```
avg_price_by_make = usedcar.groupby('Make')['Price'].mean()
avg_price_by_make = avg_price_by_make.sort_values(ascending=False)

# Plot a bar chart of the averages
plt.bar(avg_price_by_make.index, avg_price_by_make.values)
plt.title('Average Price of Cars by Make')
plt.xlabel('Make')
plt.ylabel('Average Price ($)')
plt.xticks(rotation=90)
plt.show()
```



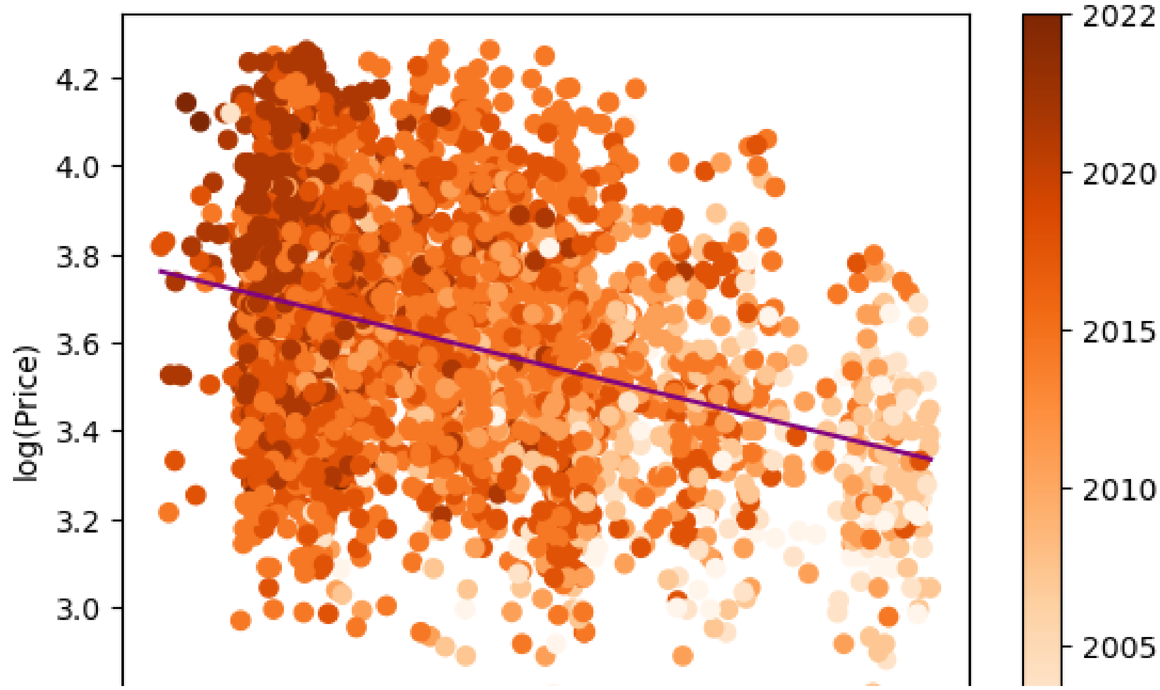
- When we delve into the chart, it becomes apparent that luxury car brands such as Bentley, Ferrari, and Lamborghini have the highest average prices. These brands are known for their high-performance and prestigious vehicles, which naturally come with a heftier price tag. On the other hand, more affordable options like Scion and Mercury fall on the lower end of the price range, making them attractive choices for budget-conscious buyers.
- In between these extremes, we see a wide range of car makes with varying average prices. This information is incredibly useful for us as it gives us insights into the relative affordability or premium associated with different car brands. Whether we're in the market for a luxurious ride or a more budget-friendly option, this chart serves as a helpful reference point for understanding the average price range of each make.

In [66]:
```
cmap = cm.Oranges
colors = usedcar['Year'].map(lambda x: (x - usedcar['Year'].min()) / (
plt.scatter(usedcar['Year'], np.log(usedcar['Price']), c=colors, cmap=

p = np.polyfit(usedcar['Year'], np.log(usedcar['Price']), 1)
x_new = np.linspace(usedcar['Year'].min(), usedcar['Year'].max(), 50)
y_new = p[0] * x_new + p[1]
plt.plot(x_new, y_new, color='Purple')

cbar = plt.colorbar()
cbar.ax.set_yticklabels([2001, 2005, 2010, 2015, 2020, 2022])


cbar.set_label('Year')
plt.xlabel('Year')
plt.ylabel('log(Price)')
plt.show()
```



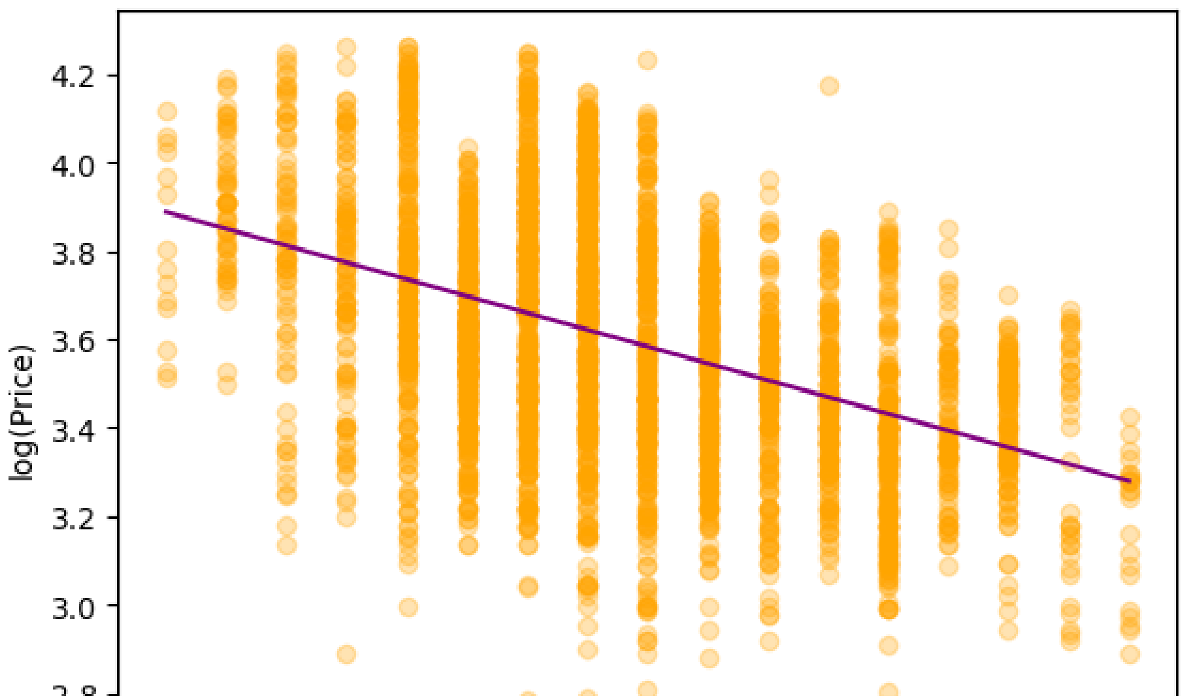
The above graph shows a positive correlation between the logarithm of price and year. This indicates that newer cars tend to be more expensive than older cars. The scatterplot shows a wide range of prices for each year, but the general trend is upward.

In [67]:
```
cmap = cm.Oranges
colors = usedcar['Year'].map(lambda x: (x - usedcar['Year'].min()) / (
plt.scatter(usedcar['Mileage'], np.log(usedcar['Price']), c=colors, cm

p = np.polyfit(usedcar['Mileage'], np.log(usedcar['Price']), 1)
x_new = np.linspace(usedcar['Mileage'].min(), usedcar['Mileage'].max()
y_new = p[0] * x_new + p[1]
plt.plot(x_new, y_new, color='Purple')

cbar = plt.colorbar()
cbar.ax.set_yticklabels([2001, 2005, 2010, 2015, 2020, 2022])


cbar.set_label('Year')
plt.xlabel('Mileage')
plt.ylabel('log(Price)')
plt.show()
```



- The scatter plot shows a negative correlation between the log price and mileage. This suggests that as the mileage of a used car increases, its price tends to decrease. The darker colors in the graph indicate newer model years, and we can see that newer cars generally have lower mileage and higher prices.
- While our analysis showed a negative correlation between mileage and price overall, year also plays an important role here. There were some cases where new cars with high mileage tended to have higher prices. Similarly, some older cars with very low mileage will have very premium prices. This could be due to factors such as the car's model, features, or rarity. In some special cases, certain vehicles with high mileage and old age can still fetch high prices due to their collectible value.

In [68]:
```
plt.scatter(usedcar['MinMPG'], np.log(usedcar['Price']),color='Orange'
p = np.polyfit(usedcar['MinMPG'], np.log(usedcar['Price']), 1)
x_new = np.linspace(usedcar['MinMPG'].min(), usedcar['MinMPG'].max(),
y_new = p[0] * x_new + p[1]
plt.plot(x_new, y_new, color='Purple')
plt.xlabel('MinMPG')
plt.ylabel('log(Price)')
plt.show()
```

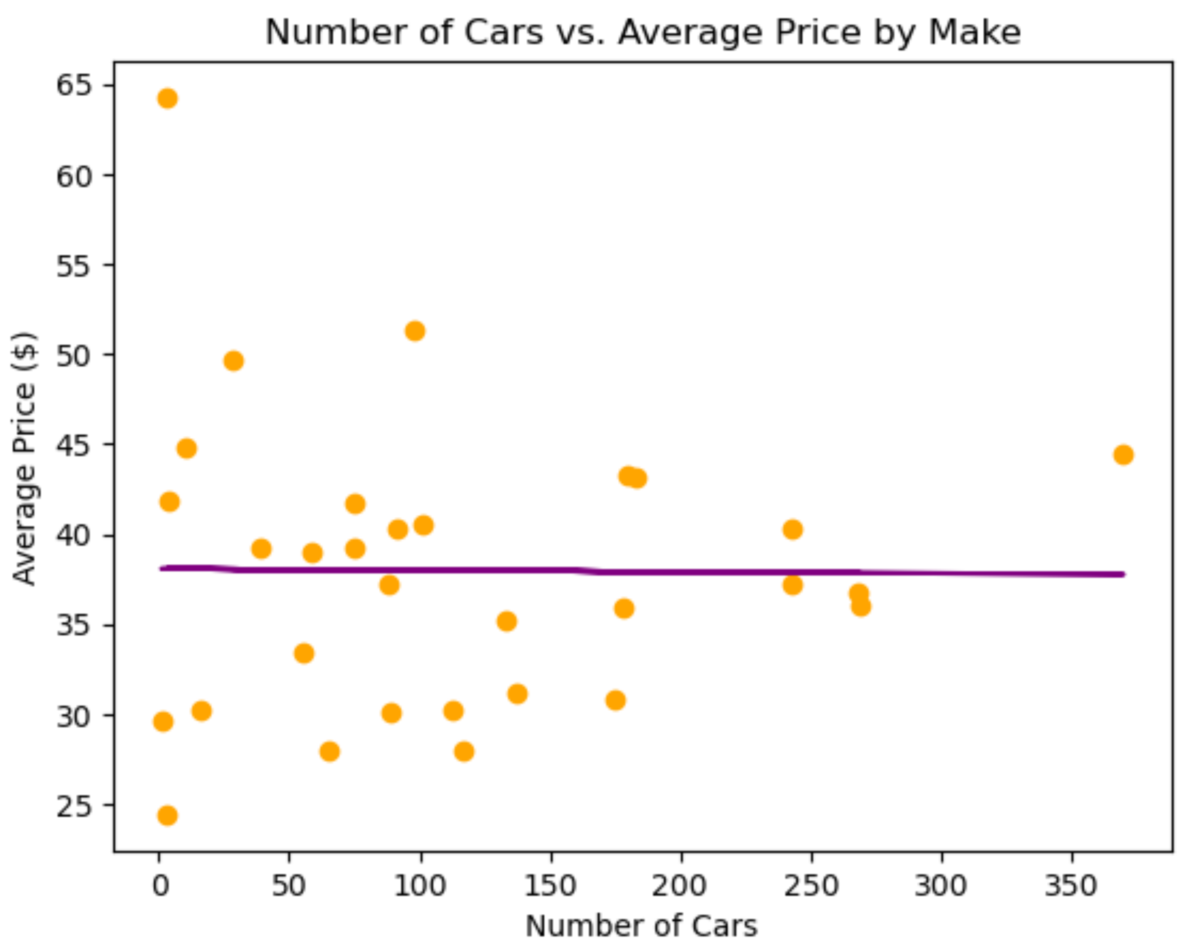


The graph suggests that cars with higher minimum MPG tend to be less expensive than cars with lower minimum MPG. This may because fuel-efficient cars could be those economy vehicles that are more affordable due to their lower operating costs.

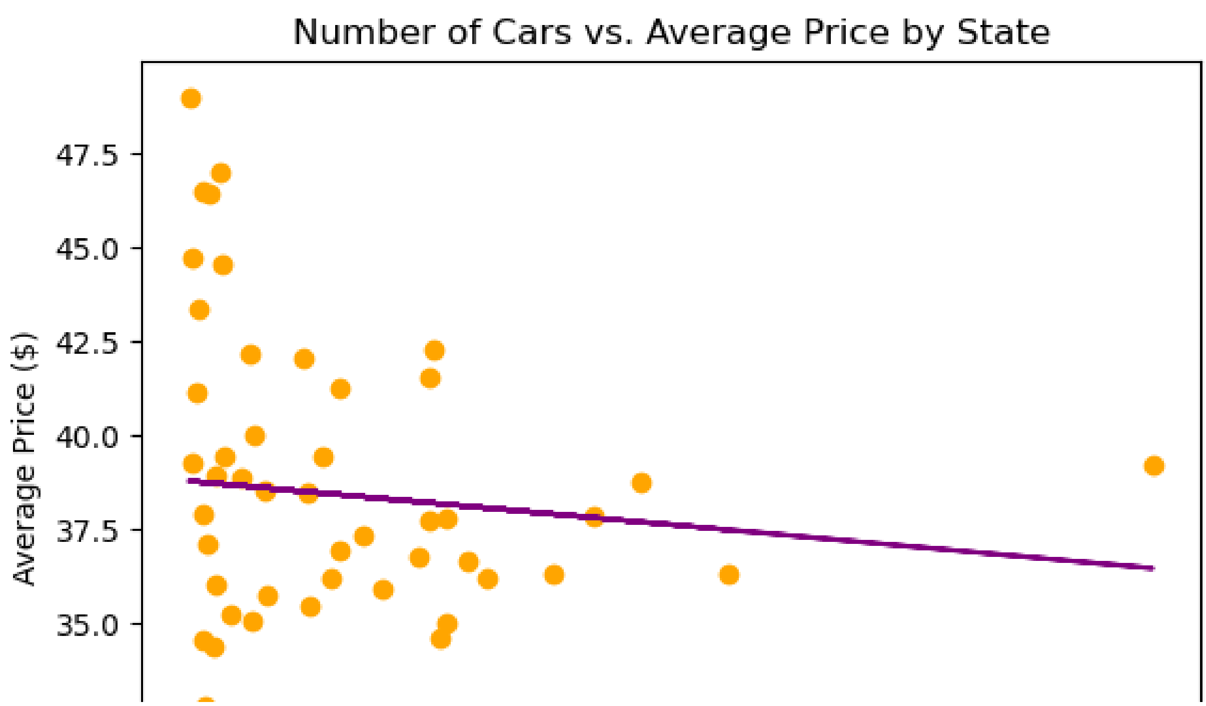
In [69]:
```
# Aggregate car prices by state
avg_price_by_state = usedcar.groupby('State')['Price'].mean().reset_ir

# Create the choropleth map using Plotly
fig = px.choropleth(avg_price_by_state,
                    locations='State',  # Column containing the state
                    locationmode='USA-states',  # Specify the location
                    color='Price',  # Column containing the average pr
                    color_continuous_scale='pinkyl',
                    scope='usa',  # Limit the map scope to the USA
                    labels={'Price': 'Average Price ($)'},
                    title='Average Price of Used Cars by State')

fig.show()
```

Obviously, the average price of used cars varies from state to state, which may be related to the economic development of the state, the price level, the purchasing power of the residents, etc. Contrary to our assumptions, Wyoming, Iowa, etc. have the highest prices, while developed states such as California and New York are not very expensive. This could be due to a lower supply and higher demand in these areas, resulting in higher used car prices, or it could be due to the fact that states with high levels of economic development, such as New York and California, may have more cars for sale, resulting in a high supply and thus depressing the prices of used cars

In [70]:
```
num_cars_by_make = usedcar.groupby('Make')['Price'].count()

# Calculate the average price by make
avg_price_by_make = usedcar.groupby('Make')['Price'].mean()

# Create a scatter plot of number of cars vs. average price by make
plt.scatter(num_cars_by_make, avg_price_by_make,color='orange')
plt.title('Number of Cars vs. Average Price by Make')
plt.xlabel('Number of Cars')
plt.ylabel('Average Price ($)')

X = num_cars_by_make.values.reshape(-1, 1)
y = avg_price_by_make.values.reshape(-1, 1)
model = LinearRegression()
model.fit(X, y)

# Add the regression line to the scatter plot
plt.plot(X, model.predict(X), color='purple')

plt.show()
```



In [71]:
```
num_cars_by_state = usedcar.groupby('State')['Price'].count()

# Calculate the average price by make
avg_price_by_state = usedcar.groupby('State')['Price'].mean()

# Create a scatter plot of number of cars vs. average price by make
plt.scatter(num_cars_by_state, avg_price_by_state,color='orange')
plt.title('Number of Cars vs. Average Price by State')
plt.xlabel('Number of Cars')
plt.ylabel('Average Price ($)')

X = num_cars_by_state.values.reshape(-1, 1)
y = avg_price_by_state.values.reshape(-1, 1)
model = LinearRegression()
model.fit(X, y)

# Add the regression line to the scatter plot
plt.plot(X, model.predict(X), color='purple')

plt.show()
```



When we looked at the relationship between sales and prices, we found that there was no clear correlation for different makes, possibly because cars of different price ranges have their own audience and stable sales. On the other hand, we found that the states with high sales have lower prices than those with low sales, which confirms our previous speculation that certain developed states have high demand and supply, resulting in fierce competition and lower prices.

Rating Analysis

In this portion, we want to dig deeper and try to make a connection between seller review, rating, model with its price.

In [84]:
```
import matplotlib.pyplot as plt

# Group the data by "SellerRating" and calculate the average price
avg_price_by_rating = usedcar.groupby('SellerRating')['Price'].mean()
avg_price_by_rating = avg_price_by_rating.sort_values(ascending=False)

# Plot a bar chart of the averages with reduced bar width
plt.bar(avg_price_by_rating.index, avg_price_by_rating.values, width=0
plt.title('Average Price by Seller Rating')
plt.xlabel('Seller Rating')
plt.ylabel('Average Price')
plt.xticks(rotation=90)
plt.show()
```

Although after rate of 3.75 is pretty much same for the price prediction. it shows that rating from 3.5 to 3.75 is less average price

Model

Preprocessing

In [86]:

```
# Choose features that have correlation coefficients >0.05
correlations = usedcar.corr()['Price'].abs()
threshold = 0.05
numerical_features = [feature for feature in correlations.index if cor
categorical_features = ['Make', 'Model', 'Used/New', 'State', 'Drivetr

# Select X and y
X = usedcar.drop('Price', axis=1)
y = usedcar['Price']

#Scale
scaler = StandardScaler()
X_numerical = scaler.fit_transform(X[numerical_features])

# Converting categorical features to numerical features
encoder = OneHotEncoder(sparse=False)
X_categorical = encoder.fit_transform(X[categorical_features])

# Combine numeric and categorical features
X_preprocessed = np.concatenate((X_numerical, X_categorical), axis=1)

# Split train and test data
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y,

# PCA to reduce features
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

Linear

In [87]:
```
lr = LinearRegression()
lr.fit(X_train_pca, y_train)
y_pred_linear = lr.predict(X_test_pca)
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)
print(f"Linear Regression mean squared error: {mse_linear}")
print(f"Linear Regression R2 score: {r2_linear}\n")
```

```
Linear Regression mean squared error: 24.926164351151535
Linear Regression R2 score: 0.7776597302486512
```

Ridge

In [88]:
```
ridge = Ridge(alpha=1)
ridge.fit(X_train_pca, y_train)
y_pred_ridge = ridge.predict(X_test_pca)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)
print(f"Ridge Regression mean squared error: {mse_ridge}")
print(f"Ridge Regression R2 score: {r2_ridge}\n")
```

```
Ridge Regression mean squared error: 24.90261997271936
Ridge Regression R2 score: 0.7778697450498855
```

KNN Regression

In [89]:
```
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(X_train_pca, y_train)
y_pred_knn = knn.predict(X_test_pca)
mse_knn = mean_squared_error(y_test, y_pred_knn)
r2_knn = r2_score(y_test, y_pred_knn)
print(f"KNN Regression mean squared error: {mse_knn}")
print(f"KNN Regression R2 score: {r2_knn}\n")
```

```
KNN Regression mean squared error: 19.80248053340931
KNN Regression R2 score: 0.8233627604505201
```

SVR

In [90]:
```
svr = SVR(kernel='linear', C=1)
svr.fit(X_train_pca, y_train)
y_pred_svr = svr.predict(X_test_pca)
mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)
print(f"SVR mean squared error: {mse_svr}")
print(f"SVR R2 score: {r2_svr}\n")
```

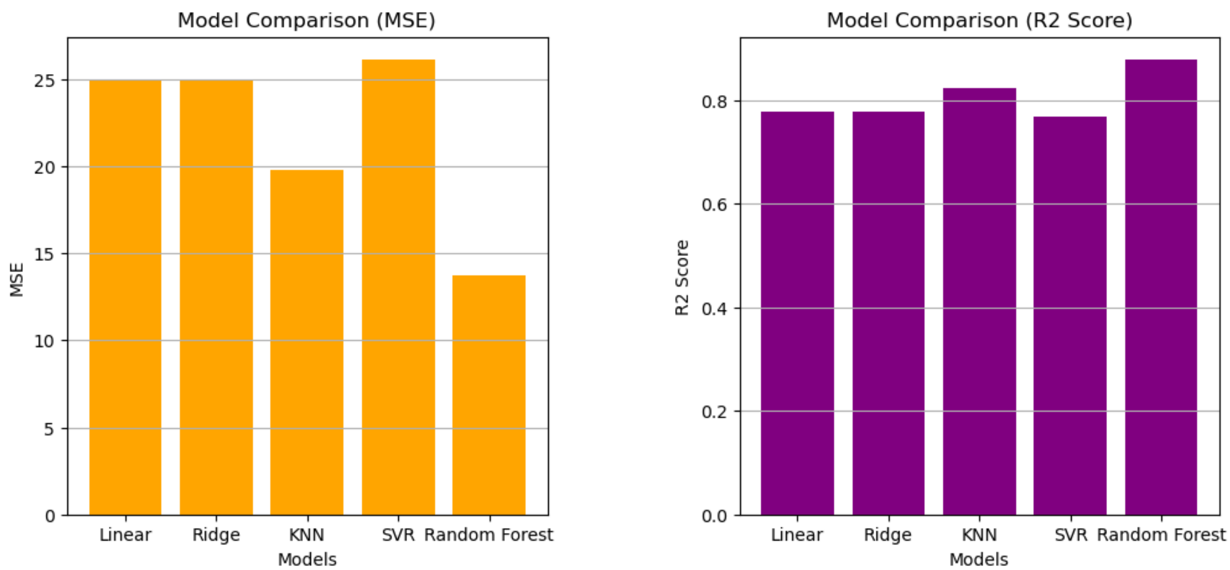
```
SVR mean squared error: 26.110828228631387
SVR R2 score: 0.7670925815139772
```

Random Forest

In [91]:
```
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print(f"Random Forest Regression mean squared error: {mse_rf}")
print(f"Random Forest Regression R2 score: {r2_rf}\n")
```

```
Random Forest Regression mean squared error: 13.6978067746439
Random Forest Regression R2 score: 0.8778161769949383
```

In [92]:
```
model_names = ['Linear', 'Ridge', 'KNN', 'SVR', 'Random Forest']

mse_sorted = [mse_linear,mse_ridge, mse_knn,mse_svr,mse_rf]
r2_sorted = [r2_linear,r2_ridge,r2_knn,r2_svr,r2_rf]


fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

#  MSE bar chart
ax1.bar(model_names, mse_sorted, label='MSE', color='Orange')
ax1.set_xlabel('Models')
ax1.set_ylabel('MSE')
ax1.set_title('Model Comparison (MSE)')
ax1.grid(axis='y')

#  Score bar chart
ax2.bar(model_names, r2_sorted, label='R2 Score', color='Purple')
ax2.set_xlabel('Models')
ax2.set_ylabel('R2 Score')
ax2.set_title('Model Comparison (R2 Score)')
ax2.grid(axis='y')

plt.subplots_adjust(wspace=0.4)
plt.show()
```



According to the model results it can be seen that random forest is the best model with MSE of 14 and Score of 0.88.

the reason that random forest outperform the rest is 1: Random Forest is a robust algorithm that can handle noisy or irrelevant features in the dataset especially when dataset is big. 2. It respond very well towards high dimesional data: In this dataset, we have a lot of features that can be included in the analysis making random forest one of the best option because it focus on the most informative features than being distracted from noises like other

Model Improvement

Random Forest

In [93]:
```
# Choose best parameters for random forest
rf_params = {'n_estimators': [10, 50, 100,200],
             'max_depth': [None, 10, 20, 30],
             'min_samples_split': [2, 5, 10]}
shuffle = ShuffleSplit(n_splits=50, test_size=0.25)
grid_rf = GridSearchCV(rf, param_grid=rf_params, cv=shuffle,return_tra
grid_rf.fit(X_train,y_train)
print("best param:",grid_rf.best_params_)
print("best train score:", grid_rf.cv_results_['mean_train_score'][gri
print("best test score:",grid_rf.best_score_)
```

```
best param: {'max_depth': 30, 'min_samples_split': 2, 'n_estimators':
200}
best train score: 0.9794306019704928
best test score: 0.8537318713550954
```

In [94]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import ShuffleSplit, GridSearchCV

# Define the RandomForestRegressor model
rf = RandomForestRegressor()

# Define the parameter grid for GridSearchCV
rf_params = {'n_estimators': [100, 200, 300],
             'max_depth': [None, 5, 10],
             'min_samples_split': [2, 5, 10]}

# Define the shuffle split for cross-validation
shuffle = ShuffleSplit(n_splits=50, test_size=0.25)

# Create the GridSearchCV object
grid_rf = GridSearchCV(rf, param_grid=rf_params, cv=shuffle, return_tr

# Fit the GridSearchCV on the training data
grid_rf.fit(X_train, y_train)

# Print the best parameters found
print("Best parameters:", grid_rf.best_params_)
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estim
ators': 200}
```

Type *Markdown* and LaTeX: α^2

Summary

1. The sales and prices of used cars across states in the US may be influenced by multiple factors, including year, mileage and make as main factors.

In the previous plot, the mileage and year has very noticable trend with the price which is more identiable as a linear correlation, make shows more interesting insights but it is still a very important main factor

2. More developed states, such as California and New York, may have higher sales and prices for used cars, while states with smaller populations or weaker economies, such as Wyoming and South Dakota, may have lower sales and prices for used cars.

According to our visualization. our assumption is a bit off as the developed states such as Cali and New York, used car prices are not as high, the highest price is among the mid-west. Our assumption is that big cities like Cali and New York, especially in NY, driving demand is lower because of congestion therefore decentize driving wants. Maybe other factors are brand new car markets are more appealing than used car market etc.

3. There may be a correlation between the sales and prices of used cars, i.e., make with higher sales may have lower average prices, while make with lower sales may have higher average prices.

With this assumption, our visualization demonstrated a more sophsicated answer: for luxuary used car market, average price can have high variance based on its model, it performance and functionality but on the average still higher price despite used. On more low end of cars, they have less price variance as compared to high end, therefore less extremes as well