Dataset:

Car Details from CarDekho (Used Car Dataset)

Goal:

Building predictive and analytical models for used car pricing and market trends

Research Questions:

1.Can we predict the selling price of a used car based on its features (year, km driven, fuel type, transmission, etc.)? Use: Helps car dealers, buyers, and sellers estimate fair market prices and make informed decisions while buying or selling used cars.

2.Can we classify whether a car’s resale value is high, medium, or low using its attributes (brand, age, mileage, etc.)? Use: Supports customers in understanding car depreciation rates and assists businesses in segmenting cars for pricing strategy and loan evaluation.

3.What are the key factors influencing car prices across different fuel types and transmission modes (manual vs automatic)? Use: Provides insights for manufacturers, dealers, and policymakers into consumer preferences and market demand trends

```
# Mount the data
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
#importing Numpy and pandas
import numpy  as np
import pandas as pd

#Reading csv file from drive
# read the data
df=pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/CAR DETAILS FROM CAR DEKHO.csv')
#Shape of the data
print("Shape of data :")
df.shape
```

```
Shape of data :
(4340, 8)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
```

```
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

```
df.describe()
```

	year	selling_price	km_driven
count	4340.000000	4.340000e+03	4340.000000
mean	2013.090783	5.041273e+05	66215.777419
std	4.215344	5.785487e+05	46644.102194
min	1992.000000	2.000000e+04	1.000000
25%	2011.000000	2.087498e+05	35000.000000
50%	2014.000000	3.500000e+05	60000.000000
75%	2016.000000	6.000000e+05	90000.000000
max	2020.000000	8.900000e+06	806599.000000

```
# dropping the rows where 'high ' is NaN
#df = df.dropna(subset='high')
df = df.dropna()
```

```
df.isnull().sum()
df.shape
```

```
(4340, 8)
```

```
df.isnull().sum()
```

	0
name	0
year	0
selling_price	0
km_driven	0
fuel	0
seller_type	0
transmission	0
owner	0

dtype: int64

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   name           4340 non-null   object
 1   year           4340 non-null   int64 
 2   selling_price  4340 non-null   int64 
 3   km_driven      4340 non-null   int64 
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

```
df.isnull().values.any()
```

```
np.False_
```

```
# Handling Outliers
numeric_cols = ['selling_price', 'km_driven']

for col in numeric_cols:
```

```
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    print(f"\nColumn: {col}")
    print("Q1:", Q1)
    print("Q3:", Q3)
    print("Lower Bound:", lower_bound)
    print("Upper Bound:", upper_bound)
```

```
Column: selling_price
Q1: 208749.75
Q3: 600000.0
Lower Bound: -378125.625
Upper Bound: 1186875.375

Column: km_driven
Q1: 35000.0
Q3: 90000.0
Lower Bound: -47500.0
Upper Bound: 172500.0
```

```
# Detect outliers
outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
print("Number of outliers :",len(outliers))
```

```
Number of outliers : 110
```

```
numeric_cols = ['selling_price', 'km_driven']

for col in numeric_cols:
    # Calculate Q1 and Q3
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)

    # IQR
    IQR = Q3 - Q1

    # Bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
```

```
    # Remove outliers
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

# Final dataset size after removing outliers
df.shape
```

```
(3962, 8)
```

```
# Detect outliers
outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
print("Number of outliers :",len(outliers))
```

```
Number of outliers : 0
```

```
# 2. Normalization
# Importing Min-Max Scaler
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Numeric columns to normalize
num_cols = ['selling_price', 'km_driven']

# --- Min-Max Scaling ---
minmax = MinMaxScaler()
scaled = minmax.fit_transform(df[num_cols])

# Convert to DataFrame with 4 decimal places
df_normalized = pd.DataFrame(scaled.round(4), columns=num_cols, index=df.index)

# Replace original columns with normalized ones
df_updated = df.copy()
df_updated[num_cols] = df_normalized


# ------------------------
# Binning using mean ± std
# ------------------------

# 1. Bins for selling_price
price_bins = [
    df['selling_price'].min(),
    df['selling_price'].mean() - df['selling_price'].std(),
    df['selling_price'].mean() + df['selling_price'].std(),
    df['selling_price'].max()
```

```
]

# Ensure sorted and unique bins
price_bins = sorted(set(price_bins))

price_labels = ['Low', 'Moderate', 'High'][:len(price_bins)-1]

df_updated['price_bins'] = pd.cut(df['selling_price'], bins=price_bins, labels=price_labels, include_lowest=True)


# 2. Bins for km_driven
km_bins = [
    df['km_driven'].min(),
    df['km_driven'].mean() - df['km_driven'].std(),
    df['km_driven'].mean() + df['km_driven'].std(),
    df['km_driven'].max()
]

# Sorted/unique bins
km_bins = sorted(set(km_bins))

km_labels = ['Low', 'Moderate', 'High'][:len(km_bins)-1]

df_updated['km_bins'] = pd.cut(df['km_driven'], bins=km_bins, labels=km_labels, include_lowest=True)


# Save cleaned dataset
df_updated.to_csv("/content/gdrive/My Drive/Colab Notebooks/car_dekho_cleaned.csv", index=False)
```

```
df.head()
```

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner
0	Maruti 800 AC	2007	60000	70000	Petrol	Individual	Manual	First Owner
1	Maruti Wagon R LXI Minor	2007	135000	50000	Petrol	Individual	Manual	First Owner
2	Hyundai Verna 1.6 SX	2012	600000	100000	Diesel	Individual	Manual	First Owner
3	Datsun RediGO T Option	2017	250000	46000	Petrol	Individual	Manual	First Owner
4	Honda Amaze VX i-DTEC	2014	450000	141000	Diesel	Individual	Manual	Second Owner

Next steps: Generate code with df New interactive sheet

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

# Make a copy of original dataframe
df_updated = df.copy()

# Numeric columns in Car dataset
num_cols = ['year', 'selling_price', 'km_driven']

# ------------------------------------------------
# STEP 1: MANUAL BINNING (Equal-Width Bins)
# ------------------------------------------------

num_manual_bins = 4
labels_manual = ['Low', 'Moderate', 'High', 'Very High']

for col in num_cols:
    min_val = df[col].min()
    max_val = df[col].max()

    # Create equal-width bins using linspace
    bins_manual = np.linspace(min_val, max_val, num_manual_bins + 1)

    df_updated[col + '_manual_bins'] = pd.cut(
        df[col],
        bins=bins_manual,
        labels=labels_manual,
        include_lowest=True
    )

# ------------------------------------------------
# STEP 2: DYNAMIC BINNING (Mean ± Std Bins)
# ------------------------------------------------

for col in num_cols:
    mean = df[col].mean()
    std = df[col].std()

    bin1 = df[col].min()
    bin2 = mean - std
    bin3 = mean + std
    bin4 = df[col].max()

    # Avoid overlapping bins
    if bin2 <= bin1: bin2 = bin1 + 1e-5
```

```
        if bin3 <= bin2: bin3 = bin2 + 1e-5
        if bin4 <= bin3: bin4 = bin3 + 1e-5

        bins_dynamic = [bin1, bin2, bin3, bin4]
        labels_dynamic = ['Low', 'Moderate', 'High']

        df_updated[col + '_dynamic_bins'] = pd.cut(
            df[col],
            bins=bins_dynamic,
            labels=labels_dynamic,
            include_lowest=True
        )

    # ---------------------------------------------
    # STEP 3: Plot Distributions
    # ---------------------------------------------

    for col in num_cols:
        plt.figure(figsize=(12,5))

        # Original distribution
        plt.subplot(1,3,1)
        df[col].hist(bins=30, edgecolor='black')
        plt.title(f"Original Distribution - {col}")
        plt.xlabel(col)
        plt.ylabel("Frequency")

        # Manual Bins Distribution
        plt.subplot(1,3,2)
        df_updated[col + '_manual_bins'].value_counts().sort_index().plot(
            kind='bar', edgecolor='black'
        )
        plt.title(f"Manual Bins - {col}")
        plt.xlabel("Category")
        plt.ylabel("Count")
        plt.xticks(rotation=45)

        # Dynamic Bins Distribution
        plt.subplot(1,3,3)
        df_updated[col + '_dynamic_bins'].value_counts().sort_index().plot(
            kind='bar', edgecolor='black'
        )
        plt.title(f"Dynamic Bins - {col}")
        plt.xlabel("Category")
        plt.ylabel("Count")
        plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()
```