

**Minor Project Report**

**on**

**[**Expense Tracker System**]**

Submitted to

**LOVELY PROFESSIONAL UNIVERSITY**

in partial fulfilment of the requirements for the award of degree of

**Master of Computer Applications**

**LOVELY FACULTY OF TECHNOLOGY & SCIENCES**

**LOVELY PROFESSIONAL UNIVERSITY**

**PUNJAB**

**[2023-2024]**

**Submitted By Supervised By**

Md Aiyaz  **Dr. Tarandeep Singh Walia**

12215953 ( Dept. of Computer Science)

|  |  |  |
| --- | --- | --- |
| **Table of Content** | | |
| ***Sr. No.*** | ***Content*** | ***Page Index*** |
| 1 | Introduction about project | 3-4 |
| 2 | Project Modules and its description | 4-5 |
| 3 | Coding | 6-23 |
| 4 | Screenshot output | 24-38 |

1. **Introduction about project**

**Introduction to Project: Expense Tracker System**

This project aims to develop an expense tracker system for cars. The system will allow users to:

* Import a dataset of car expenses (car.csv)
* Clean and prepare the dataset (car1.csv)
* Add, delete, and update car expenses.
* View expenses by car make.
* Track fuel costs
* Track maintenance costs
* Track insurance costs
* Track depreciation
* Choose a car to view detailed expense information.
* Use machine learning to predict the price and total expense of a car.
* Collect insurance expense data and visualize it.

The system will be developed in a modular fashion, with each module responsible for a specific task. This will make the system easier to develop, maintain, and extend.

The system will be implemented using a variety of technologies, including Python, and machine learning libraries.

**Benefits of the System**

The system will provide several benefits to users, including:

* The ability to track and manage car expenses in a more efficient and organized way.
* The ability to identify areas where car expenses can be reduced.
* The ability to make more informed decisions about car ownership and maintenance.
* The ability to use machine learning to predict the future cost of car ownership.

**Target Audience**

The target audience for the system is anyone who owns or operates a car. This includes individuals, families, and businesses.

**Conclusion**

The expense tracker system will provide a valuable tool for car owners to track and manage their expenses. The system will be easy to use and provide users with valuable insights into their car expenses.

1. **Project Modules and its description**

**T**he project modules for the expense tracker system and their descriptions:

**Module 1: Data Import and Cleaning**

This module is responsible for importing the car expense dataset (car.csv) and cleaning it to prepare it for use in the other modules. This may involve removing duplicate data, correcting errors, and converting data to a consistent format.

**Module 2: Data Management**

This module provides users with the ability to add, delete, and update car expenses. It also provides users with the ability to view expenses by car make.

**Module 3: Fuel Cost Tracking**

This module allows users to track their fuel costs for each car. This includes tracking the date, mileage, and amount spent on fuel.

**Module 4: Maintenance Cost Tracking**

This module allows users to track their maintenance costs for each car. This includes tracking the date, description, and amount spent on maintenance.

**Module 5: Insurance Cost Tracking**

This module allows users to track their insurance costs for each car. This includes tracking the policy number, premium, and expiration date.

**Module 6: Depreciation Tracking**

This module allows users to track the depreciation of each car. This includes tracking the purchase price, depreciation rate, and current value of the car.

**Module 7: Car Expense Visualization**

This module provides users with a variety of visualizations to help them understand their car expenses. This may include charts and graphs showing fuel costs, maintenance costs, insurance costs, and depreciation over time.

**Module 8: Machine Learning Prediction**

This module uses machine learning to predict the price and total expense of a car. This information can be used to help users make more informed decisions about car ownership and maintenance.

**Module 9: Insurance Expense Collection and Visualization**

This module collects insurance expense data from users and visualizes it to help users understand their insurance costs. This may include charts and graphs showing insurance costs by car make, model, and year.

|  |  |
| --- | --- |
| **Module** | **Description** |
| Data Import and Cleaning | Imports and cleans the car expense dataset |
| Data Management | Allows users to add, delete, and update car expenses, and view expenses by car make |
| Fuel Cost Tracking | Allows users to track their fuel costs for each car |
| Maintenance Cost Tracking | Allows users to track their maintenance costs for each car |
| Insurance Cost Tracking | Allows users to track their insurance costs for each car |
| Depreciation Tracking | Allows users to track the depreciation of each car |
| Car Expense Visualization | Provides users with visualizations to help them understand their car expenses |
| Machine Learning Prediction | Uses machine learning to predict the price and total expense of a car |
| Insurance Expense Collection and Visualization | Collects insurance expense data from users and visualizes it to help users understand their insurance costs |

1. **Coding**

\*Packages to load in \*

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")

**Data Loading**

df\_automobile = pd.read\_csv("CAR.csv")

df\_automobile

**Admin Access**

1.Adding 2. updating 3.deleteing

import pandas as pd

df\_automobile = pd.read\_csv("CAR.csv")

# Function to add data to the DataFrame

def add\_data(dataframe):

    new\_data = {}

    for col in dataframe.columns:

        new\_data[col] = input(f"Enter {col}: ")

    dataframe = dataframe.append(new\_data, ignore\_index=True)

    return dataframe

# Function to update data in the DataFrame

def update\_data(dataframe):

    index\_to\_update = int(input("Enter the index of the entry to update: "))

    column\_to\_update = input("Enter the column to update: ")

    new\_value = input("Enter the new value: ")

    dataframe.at[index\_to\_update, column\_to\_update] = new\_value

    return dataframe

# Function to delete data from the DataFrame

def delete\_data(dataframe):

    index\_to\_delete = int(input("Enter the index of the entry to delete: "))

    dataframe = dataframe.drop(index\_to\_delete)

    dataframe = dataframe.reset\_index(drop=True)

    return dataframe

while True:

    choice = input("Enter 'A' to add data, 'U' to update data, 'D' to delete data, or 'Q' to quit: ").upper()

    if choice == 'A':

        df\_automobile = add\_data(df\_automobile)

    elif choice == 'U':

        df\_automobile = update\_data(df\_automobile)

    elif choice == 'D':

        df\_automobile = delete\_data(df\_automobile)

    elif choice == 'Q':

        break

    else:

        print("Invalid choice. Please try again.")

print(df\_automobile)

**Data Cleaning**

\*Data contains "?" replace it with NAN \*

df\_data = df\_automobile.replace('?',np.NAN)

df\_data.isnull().sum()

**Missing Data**

fill missing data of normalised-losses, Price, and many with the respective column mean Fill missing data category Number of doors with the mode of the column i.e. Four

# Handling NaN values in 'Normalized-Losses'

df\_temp = df\_automobile[df\_automobile['Normalized-Losses']!='?']

normalised\_mean = df\_temp['Normalized-Losses'].astype(float).mean()

df\_automobile['Normalized-Losses'] = df\_automobile['Normalized-Losses'].replace('?',normalised\_mean).astype(float)

# Handling NaN values in 'Price'

df\_temp = df\_automobile[df\_automobile['Price']!='?']

normalised\_mean = df\_temp['Price'].astype(float).mean()

df\_automobile['Price'] = df\_automobile['Price'].replace('?',normalised\_mean).astype(float)

df\_temp = df\_automobile[df\_automobile['horsepower']!='?']

normalised\_mean = df\_temp['horsepower'].astype(int).mean()

df\_automobile['horsepower'] = df\_automobile['horsepower'].replace('?',normalised\_mean).astype(int)

df\_temp = df\_automobile[df\_automobile['peak-rpm']!='?']

normalised\_mean = df\_temp['peak-rpm'].astype(int).mean()

df\_automobile['peak-rpm'] = df\_automobile['peak-rpm'].replace('?',normalised\_mean).astype(int)

df\_temp = df\_automobile[df\_automobile['bore']!='?']

normalised\_mean = df\_temp['bore'].astype(float).mean()

df\_automobile['bore'] = df\_automobile['bore'].replace('?',normalised\_mean).astype(float)

df\_temp = df\_automobile[df\_automobile['stroke']!='?']

normalised\_mean = df\_temp['stroke'].astype(float).mean()

df\_automobile['stroke'] = df\_automobile['stroke'].replace('?',normalised\_mean).astype(float)

# Handling NaN values in 'Num-Of-Doors'

df\_automobile['Num-Of-Doors'] = df\_automobile['Num-Of-Doors'].replace('?','four')

# Other columns can be handled similarly

# Example for 'Mileage'

df\_temp = df\_automobile[df\_automobile['Mileage']!='?']

normalised\_mean = df\_temp['Mileage'].astype(float).mean()

df\_automobile['Mileage'] = df\_automobile['Mileage'].replace('?',normalised\_mean).astype(float)

# Example for 'Year'

df\_temp = df\_automobile[df\_automobile['Year']!='?']

normalised\_mean = df\_temp['Year'].astype(float).mean()

df\_automobile['Year'] = df\_automobile['Year'].replace('?',normalised\_mean).astype(float)

# Example for 'Fuel-Type'

df\_automobile['Fuel-Type'] = df\_automobile['Fuel-Type'].fillna('Unknow')

# Example for 'Model'

df\_automobile['Model'] = df\_automobile['Model'].fillna('Unknown')

# Example for 'Make'

df\_automobile['Make'] = df\_automobile['Make'].fillna('Unknown')

# Example for 'Body-Style'

df\_automobile['Body-Style'] = df\_automobile['Body-Style'].fillna('Unknown')

# Example for 'Drive-Wheels'

df\_automobile['Drive-Wheels'] = df\_automobile['Drive-Wheels'].fillna('Unknown')

# Example for 'Engine-Location'

df\_automobile['Engine-Location'] = df\_automobile['Engine-Location'].fillna('Unknown')

# Example for 'Condition'

df\_automobile['Condition'] = df\_automobile['Condition'].fillna('Unknown')

# Example for 'Payment method '

df\_automobile['Payment method'] = df\_automobile['Payment method'].fillna('Unknown')

# Example for 'Payment method '

df\_automobile['Payment method'] = df\_automobile['Payment method'].fillna('Unknown')

# Example for 'Maintenance-Costs '

df\_automobile['Maintenance-Costs'] = df\_automobile['Maintenance-Costs'].fillna('Unknown')

# Example for 'Date '

df\_automobile['Date'] = df\_automobile['Date'].fillna('Unknown')

# You can adapt this pattern for the other columns as well .

# Save the updated DataFrame to a new CSV file

df\_automobile.to\_csv('car1.csv', index=False)

df\_automobile.head()

**Summary statistics of variable**

df\_automobile.describe()

**Transpose**

df\_automobile.describe().T

**User Module**

1. View Expenses of Cars by Make
2. Track Fuel Costs for a Car
3. Track Maintenance Costs for a Car
4. Track Insurance Costs for a Car
5. Track Depreciation for a Car
6. Choose a Car to View Details

df\_automobile = pd.read\_csv("car1.csv")

df\_automobile['Date'] = pd.to\_datetime(df\_automobile['Date'])

# Function to add data to the DataFrame

def add\_data(dataframe):

    new\_data = {}

    for col in dataframe.columns:

        new\_data[col] = input(f"Enter {col}: ")

    dataframe = dataframe.append(new\_data, ignore\_index=True)

    return dataframe

# Function to update data in the DataFrame

def update\_data(dataframe):

    index\_to\_update = int(input("Enter the index of the entry to update: "))

    if 0 <= index\_to\_update < len(dataframe):

        column\_to\_update = input("Enter the column to update: ")

        if column\_to\_update in dataframe.columns:

            new\_value = input("Enter the new value: ")

            dataframe.at[index\_to\_update, column\_to\_update] = new\_value

        else:

            print("Invalid column name.")

    else:

        print("Invalid index.")

    return dataframe

# Function to delete data from the DataFrame

def delete\_data(dataframe):

    index\_to\_delete = int(input("Enter the index of the entry to delete: "))

    if 0 <= index\_to\_delete < len(dataframe):

        dataframe = dataframe.drop(index\_to\_delete)

        dataframe = dataframe.reset\_index(drop=True)

    else:

        print("Invalid index.")

    return dataframe

# Function to view expenses of cars

def view\_car\_expenses(dataframe, choice):

    if choice == '1':

        make = input("Enter the make of the car: ")

        if make in dataframe['Make'].unique():

            cars\_by\_make = dataframe[dataframe['Make'] == make]

            cars\_by\_make['Price'] = pd.to\_numeric(cars\_by\_make['Price'], errors='coerce')

            cars\_by\_make = cars\_by\_make.dropna(subset=['Price'])

            total\_amount = sum(cars\_by\_make['Price'])

            print(cars\_by\_make)

            print(f"Total amount for {make}: {total\_amount}")

            years = cars\_by\_make['Year']

            plt.pie(cars\_by\_make['Price'], labels=years, autopct='%1.1f%%')

            plt.title(f"Expenses for {make} by Year")

            plt.show()

        else:

            print("No cars found with the specified make.")

# Function to track fuel costs

def track\_fuel\_costs(dataframe, car\_index):

    car\_data = dataframe.iloc[car\_index]

    car\_mileage = pd.to\_numeric(car\_data["Mileage"], errors='coerce')

    car\_fuel\_type = pd.to\_numeric(car\_data["Fuel-Type"], errors='coerce')

    # Calculate fuel costs based on mileage and fuel type

    fuel\_costs = car\_mileage \* car\_fuel\_type

    # Visualize fuel costs

    plt.bar(['Mileage', 'Fuel Type'], [car\_mileage, car\_fuel\_type])

    plt.title(f'Fuel Costs for {car\_data["Make"]} {car\_data["Model"]}')

    plt.xlabel('Category')

    plt.ylabel('Value')

    plt.show()

# Function to track maintenance costs

def track\_maintenance\_costs(dataframe, car\_index):

    car\_data = dataframe.iloc[car\_index]

    maintenance\_types = ['Excellent', 'Good', 'Fair', 'Poor']

    maintenance\_costs = [0, 0, 0, 0]

    for index, row in dataframe.iterrows():

        if row['Condition'] == 'Excellent':

            maintenance\_costs[0] += row['Maintenance-Costs']

        elif row['Condition'] == 'Good':

            maintenance\_costs[1] += row['Maintenance-Costs']

        elif row['Condition'] == 'Fair':

            maintenance\_costs[2] += row['Maintenance-Costs']

        elif row['Condition'] == 'Poor':

            maintenance\_costs[3] += row['Maintenance-Costs']

    # Visualize maintenance costs

    plt.bar(maintenance\_types, maintenance\_costs)

    plt.title(f'Maintenance Costs for {car\_data["Make"]} {car\_data["Model"]}')

    plt.xlabel('Condition')

    plt.ylabel('Costs')

    plt.show()

# Function to track insurance costs

def track\_insurance\_costs(dataframe, car\_index):

    car\_data = dataframe.iloc[car\_index]

    # Logic to calculate insurance costs

    insurance\_costs = 500 + (2023 - car\_data['Year']) \* 100  #Example formula for insurance costs

    # Visualize insurance costs

    factors = ['Basic Coverage', 'Collision Coverage', 'Comprehensive Coverage']

    costs = [insurance\_costs \* 0.6, insurance\_costs \* 0.3, insurance\_costs \* 0.1]  # Example distribution of insurance costs

    plt.bar(factors, costs)

    plt.title(f'Insurance Costs for {car\_data["Make"]} {car\_data["Model"]}')

    plt.xlabel('Insurance Types')

    plt.ylabel('Costs')

    plt.show()

# Function to track depreciation

def track\_depreciation(dataframe, car\_index):

    car\_data = dataframe.iloc[car\_index]

    # Logic to calculate depreciation

    start\_year = int(car\_data['Year'])  # Convert to integer type

    years = [year for year in range(start\_year, 2024)]  # Example list of years for the calculation

    depreciation = [5000 \* (1 - (year - start\_year) \* 0.1) for year in years]  # Example formula for depreciation

    # Visualize depreciation

    plt.plot(years, depreciation, marker='o')

    plt.title(f'Depreciation for {car\_data["Make"]} {car\_data["Model"]}')

    plt.xlabel('Years')

    plt.ylabel('Value ($)')

    plt.show()

# Example usage

while True:

    print("1. View Expenses of Cars by Make")

    print("2. Track Fuel Costs for a Car")

    print("3. Track Maintenance Costs for a Car")

    print("4. Track Insurance Costs for a Car")

    print("5. Track Depreciation for a Car")

    print("6. Choose a Car to View Details")

    print("7. Quit")

    choice = input("Enter your choice: ")

    if choice == '1':

        view\_car\_expenses(df\_automobile, choice)

    elif choice == '2':

        car\_index = int(input("Enter the index of the car: "))

        track\_fuel\_costs(df\_automobile, car\_index)

    elif choice == '3':

        car\_index = int(input("Enter the index of the car: "))

        if 0 <= car\_index < len(df\_automobile):

            track\_maintenance\_costs(df\_automobile, car\_index)

        else:

            print("Invalid index.")

    elif choice == '4':

        car\_index = int(input("Enter the index of the car: "))

        track\_insurance\_costs(df\_automobile, car\_index)

    elif choice == '5':

        car\_index = int(input("Enter the index of the car: "))

        track\_depreciation(df\_automobile, car\_index)

    elif choice == '6':

        index = int(input("Enter the index of the car: "))

        if 0 <= index < len(df\_automobile):

            print(df\_automobile.iloc[index])

        else:

            print("Invalid index.")

    elif choice == '7':

        break

    else:

        print("Invalid choice. Please try again.")

**Total Expenses by Payment Method**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load your dataset

df = pd.read\_csv('car1.csv')

# Handling missing or invalid values

df = df.replace('?', pd.NA)  # Replace '?' with NaN for further processing

df = df.apply(pd.to\_numeric, errors='ignore')  # Convert columns to numeric, ignore errors

# Convert the 'Date' column to a datetime type

df['Date'] = pd.to\_datetime(df['Date'])

# Calculate expenses based on Price and Maintenance-Costs

df['Expense'] = df['Price'] + df['Maintenance-Costs']  # Adjust the calculation based on your specific expense calculation

# Convert the 'Expense' column to a numeric data type

df['Expense'] = pd.to\_numeric(df['Expense'])

# Ensure that 'Payment method' is a categorical variable

df['Payment method'] = df['Payment method'].astype('category')

# Create a dashboard to show the expenses of different payment methods

# First, visualize the expenses based on payment method

plt.figure(figsize=(12, 6))

sns.barplot(x='Payment method', y='Expense', data=df, estimator=sum, ci=None)

plt.title('Total Expenses by Payment Method')

plt.xlabel('Payment Method')

plt.ylabel('Total Expenses')

plt.show()

# Next, create a time-based analysis of expenses

# For example, expenses by year

plt.figure(figsize=(12, 6))

sns.lineplot(x='Date', y='Expense', data=df, estimator=sum, ci=None)

plt.title('Expenses by Year')

plt.xlabel('Year')

plt.ylabel('Total Expenses')

plt.show()

# You can further customize and create more visualizations based on your requirements and dataset.

# Finally, calculate the total expenses by payment method

total\_expenses = df.groupby('Payment method')['Expense'].sum()

print("Total Expenses by Payment Method:")

print(total\_expenses)

**Machine learning & Visualization**

df\_automobile = pd.read\_csv("car1.csv")

df\_automobile

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

import numpy as np

# Define features and target variable

features = ['engine-size', 'horsepower', 'curb-weight', 'city-mpg', 'engine-type']

target = 'Price'

# Split the data

X = df\_automobile[features]

y = df\_automobile[target]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define a preprocessor for one-hot encoding

categorical\_features = ['engine-type']

categorical\_transformer = Pipeline(steps=[

    ('onehot', OneHotEncoder(sparse=False, handle\_unknown='ignore'))

])

preprocessor = ColumnTransformer(

    transformers=[

        ('cat', categorical\_transformer, categorical\_features)

    ])

# Combine the preprocessor with the model

model = Pipeline(steps=[('preprocessor', preprocessor),

                       ('regressor', LinearRegression())])

# Train the model

model.fit(X\_train, y\_train)

# Make predictions and evaluate the model

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Absolute Error: {mae}")

print(f"Mean Squared Error: {mse}")

print(f"Root Mean Squared Error: {rmse}")

print(f"R-squared (R2): {r2}")

**prediction Price by car-wise**

# Assuming the previous code has been implemented

# Use the trained model to make predictions on the test data

y\_pred = model.predict(X\_test)

# Visualize the predictions

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred, c='blue', label='Scatter Plot')

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], 'k--', lw=2, c='red', label='Regression Line')

plt.xlabel('Actual Price')

plt.ylabel('Predicted Price')

plt.title('Actual Price vs. Predicted Price')

plt.legend()

plt.show()

# Evaluation Metrics

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Absolute Error: {mae}")

print(f"Mean Squared Error: {mse}")

print(f"Root Mean Squared Error: {rmse}")

print(f"R-squared (R2): {r2}")

**prediction Price by car-wise**

**prediction Price by car-wise**

# Assuming the previous code has been implemented

# Make predictions on the entire dataset

all\_predictions = model.predict(X)

# Add the predicted prices as a new column in the DataFrame

df\_automobile['predicted\_price'] = all\_predictions

# Display the car model-wise actual and predicted prices

car\_model\_prices = df\_automobile[['Make', 'Price', 'predicted\_price']].groupby('Make').mean()

car\_model\_prices.reset\_index(inplace=True)

# Plot the car model-wise actual and predicted prices

plt.figure(figsize=(12, 6))

sns.barplot(x='Make', y='Price', data=car\_model\_prices, color='blue', label='Actual Price')

sns.barplot(x='Make', y='predicted\_price', data=car\_model\_prices, color='red', label='Predicted Price')

plt.xticks(rotation=90)

plt.xlabel('Car Model')

plt.ylabel('Price')

plt.title('Car Model-wise Actual and Predicted Prices')

plt.legend()

plt.show()

**The expenses of the old price and the new predicted price**

import plotly.express as px

# Assuming df\_automobile is the DataFrame containing the data

# Group data by 'Make' (car model) and calculate the mean of 'Price' and 'predicted\_price'

car\_prices = df\_automobile.groupby('Make').agg({'Price': 'mean', 'predicted\_price': 'mean'}).reset\_index()

# Create an interactive bar plot using Plotly Express

fig = px.bar(car\_prices, x='Make', y=['Price', 'predicted\_price'], barmode='group',

             title='Expense Comparison of Old Price and New Predicted Price by Car Make')

# Add labels

fig.update\_layout(xaxis\_title='Car Make', yaxis\_title='Price', legend\_title='Price Type')

fig.show()

**Shows the total expenses of the old price of cars and the total of the new predicted price**

import matplotlib.pyplot as plt

# Assuming df\_automobile is the DataFrame containing the data

# Calculate the total expenses of the old price of cars

total\_old\_price = df\_automobile['Price'].sum()

# Calculate the total of the new predicted price

total\_new\_price = df\_automobile['predicted\_price'].sum()

# Data for the pie chart

labels = ['Total Old Price', 'Total New Predicted Price']

sizes = [total\_old\_price, total\_new\_price]

colors = ['#ff9999', '#66b3ff']

# Plotting the pie chart

plt.figure(figsize=(8, 6))

plt.pie(sizes, labels=labels, colors=colors, autopct=lambda p: '${:.0f}'.format(p \* sum(sizes) / 100), startangle=90)

plt.title('Total Expenses of Old Price vs. Total of New Predicted Price')

plt.show()

import matplotlib.pyplot as plt

# Assuming df\_automobile is the DataFrame containing the data

# Group the data by year and calculate the total expenses for each year

yearly\_expenses = df\_automobile.groupby('Year')['Price'].sum().reset\_index()

# Assuming the current year is 2023, you can change it according to your requirement

current\_year = 2023

# Filter the data for the current and future years

future\_years = range(current\_year + 1, current\_year + 6)  # Predict for the next 5 years

future\_expenses = []

for year in future\_years:

    future\_expense = df\_automobile[df\_automobile['Year'] == year]['predicted\_price'].sum()

    future\_expenses.append(future\_expense)

# Visualize the yearly expenses and future spend prediction

plt.figure(figsize=(10, 6))

plt.plot(yearly\_expenses['Year'], yearly\_expenses['Price'], marker='o', linestyle='-', color='b', label='Yearly Expenses')

plt.plot(future\_years, future\_expenses, marker='o', linestyle='--', color='r', label='Future Spend Prediction')

plt.xlabel('Year')

plt.ylabel('Total Expenses')

plt.title('Yearly Expenses and Future Spend Prediction')

plt.legend()

plt.show()

# Calculating the amount needed in the account for future orders

future\_order\_amount = sum(future\_expenses) - sum(yearly\_expenses[yearly\_expenses['Year'] == current\_year]['Price'])

print(f"Amount needed in the account for future orders: {future\_order\_amount}")

import pandas as pd

import plotly.express as px

# Assuming df\_automobile is the DataFrame containing the data

# Calculate expenses as 3/4 of the price

df\_automobile['expenses'] = df\_automobile['Price'] \* 3/4  # Adjust the factor according to your expense calculation

# Calculate insurance as 1/20 of the price

df\_automobile['insurance'] = df\_automobile['Price'] \* 1/20  # Adjust the factor according to your insurance calculation

# Group data by 'Make' (car model), 'Year', and 'Condition' columns and calculate the mean price, expenses, and insurance

expensive\_data = df\_automobile.groupby(['Make', 'Year', 'Condition'])[['Price', 'expenses', 'insurance']].mean().reset\_index()

# Create an interactive line plot using Plotly Express

fig = px.line(expensive\_data, x='Year', y='insurance', color='Make', title="Average Insurance Trend by Car Model and Year")

# Add a dropdown menu for car model selection

fig.update\_layout(updatemenus=[dict(type="dropdown", showactive=True, buttons=[

    dict(label=car\_model, method="update", args=[{"y": [expensive\_data[expensive\_data['Make'] == car\_model]['insurance']]}])

    for car\_model in expensive\_data['Make'].unique()

])])

# Show the interactive line plot

fig.show()

**select the car and view the Expense of that car**

import pandas as pd

import ipywidgets as widgets

from IPython.display import display

# Assuming df\_automobile is the DataFrame containing the data

# Calculate expenses as 3/4 of the price

df\_automobile['expenses'] = df\_automobile['Price'] \* 3 / 4  # Adjust the factor according to your expense calculation

# Group data by 'Make' (car model), 'Year', 'Condition', and 'insurance' columns and calculate the mean price and expenses

expensive\_data = df\_automobile.groupby(['Make', 'Year', 'Condition', 'insurance'])[['Price', 'expenses']].mean().reset\_index()

# Create a function to generate a summary table for the selected car

def create\_summary\_table(car\_model, data):

    selected\_data = data[data['Make'] == car\_model]

    total\_price\_sum = selected\_data['Price'].sum()

    total\_expenses\_sum = selected\_data['expenses'].sum()

    insurance = selected\_data['insurance'].values[0]

    summary\_df = pd.DataFrame({'Make': [car\_model], 'Total Price Sum': [total\_price\_sum], 'Total Expenses Sum': [total\_expenses\_sum], 'Insurance': [insurance]})

    return summary\_df

# Get unique car models for the dropdown list

car\_models = expensive\_data['Make'].unique()

dropdown = widgets.Dropdown(options=car\_models, description='Car Model:')

# Display the dropdown

display(dropdown)

# Define a function to update the summary table based on the dropdown value

def on\_change(change):

    if change['type'] == 'change' and change['name'] == 'value':

        selected\_car = change['new']

        summary\_table = create\_summary\_table(selected\_car, expensive\_data)

        print(summary\_table)

# Call the on\_change function when the dropdown value changes

dropdown.observe(on\_change)

# Example usage for the initial value in the dropdown list

initial\_car = dropdown.value

summary\_table = create\_summary\_table(initial\_car, expensive\_data)

print(summary\_table)

import plotly.express as px

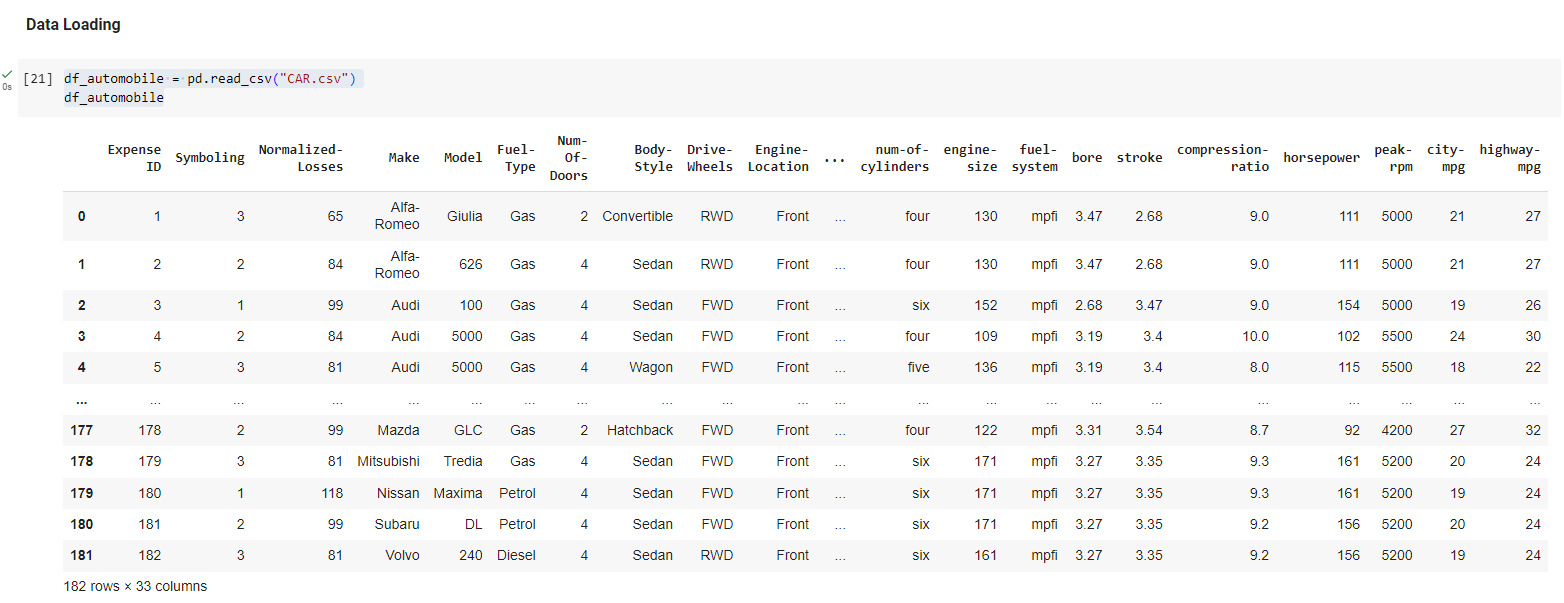
# Assuming df\_automobile is your DataFrame

fig = px.histogram(df\_automobile, x='Price', nbins=30, title='Price Distribution')

fig.show()

1. **Screenshot output**

**Data Loading**

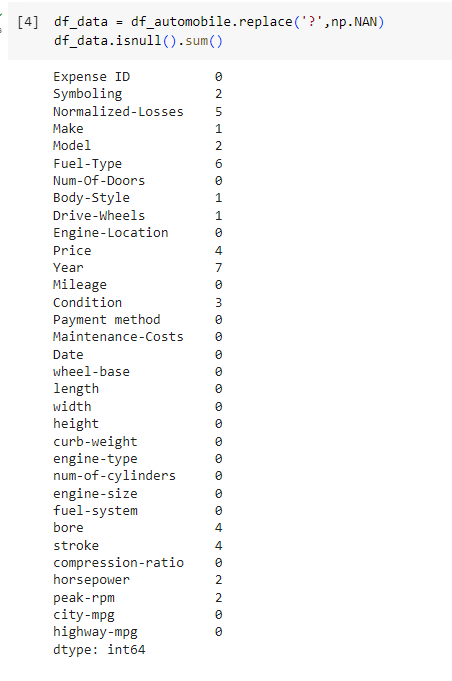


**Admin access**

****

**A screenshot of a computer

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated** **A screenshot of a computer

Description automatically generated** A screenshot of a computer

Description automatically generated A screenshot of a graph

Description automatically generated A blue square with white text

Description automatically generated A screen shot of a graph

Description automatically generated A graph of blue squares

Description automatically generated with medium confidence A screen shot of a graph

Description automatically generated A screenshot of a computer

Description automatically generated A graph of a graph showing a number of expenses

Description automatically generated with medium confidence A graph with blue lines

Description automatically generated A computer error message

Description automatically generated A graph with a red line and blue dots

Description automatically generated A graph of a car model

Description automatically generated A blue and red circle with black numbers

Description automatically generated A graph with blue dots and red dots

Description automatically generated A graph showing different colored lines

Description automatically generated A graph with numbers and a bar

Description automatically generated A screenshot of a computer

Description automatically generated