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

## Budget

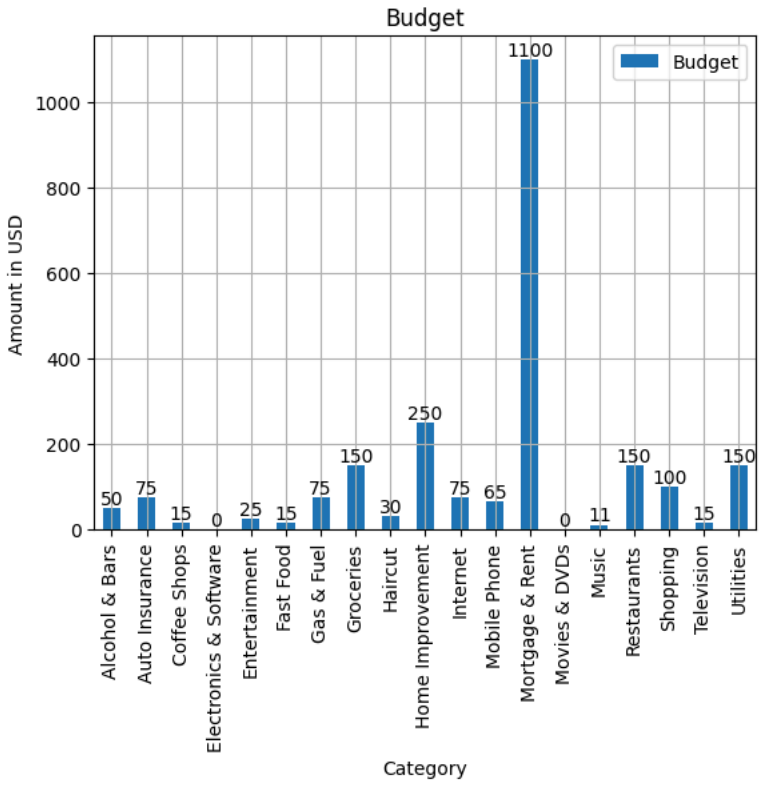
```
df = pd.read_csv('Budget.csv')
df.shape
```

```
(19, 2)
```

```
df['Category'].unique()
```

```
array(['Alcohol & Bars', 'Auto Insurance', 'Coffee Shops',
       'Electronics & Software', 'Entertainment', 'Fast Food',
       'Gas & Fuel', 'Groceries', 'Haircut', 'Home Improvement',
       'Internet', 'Mobile Phone', 'Mortgage & Rent', 'Movies & DVDs',
       'Music', 'Restaurants', 'Shopping', 'Television', 'Utilities'],
      dtype=object)
```

```
ax = df.plot(kind='bar', x='Category', title="Budget", ylabel='Amount in USD', grid=True)
ax.bar_label(ax.containers[0])
plt.show()
```




## Personal Transactions

```
df_fin = pd.read_csv('personal_transactions.csv', parse_dates=['Date'])
df_fin.shape
```

```
(806, 6)
```

```
df_fin.columns
```

```
Index(['Date', 'Description', 'Amount', 'Transaction Type', 'Category',
       'Account Name'],
      dtype='object')
```

```
df_fin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 806 entries, 0 to 805
Data columns (total 6 columns):
```

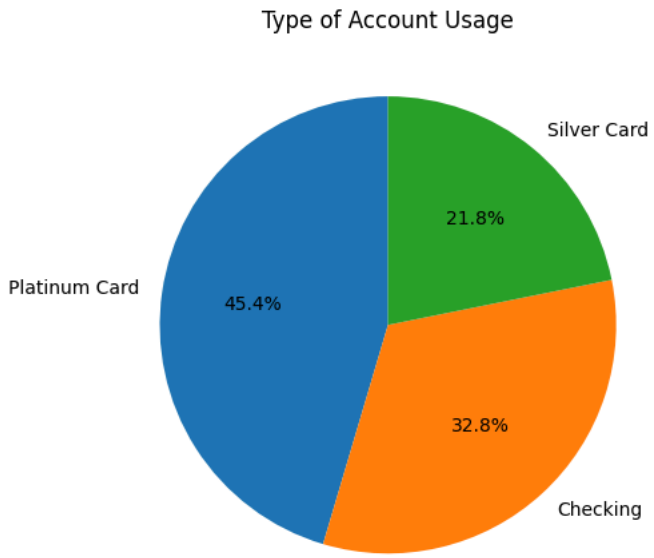
```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              806 non-null    datetime64[ns]
 1   Description       806 non-null    object
 2   Amount            806 non-null    float64
 3   Transaction Type  806 non-null    object
 4   Category          806 non-null    object
 5   Account Name      806 non-null    object
dtypes: datetime64[ns](1), float64(1), object(4)
memory usage: 37.9+ KB
```

```
df_fin['Account Name'].unique()
```

```
array(['Platinum Card', 'Checking', 'Silver Card'], dtype=object)
```

```
account_count = df_fin['Account Name'].value_counts()
```

```
plt.pie(account_count, labels=account_count.index, autopct='%1.1f%%', startangle=90)
plt.title("Type of Account Usage")
plt.tight_layout()
plt.show()
```




```
df_fin.Category.unique()
```

```
array(['Shopping', 'Mortgage & Rent', 'Restaurants',
       'Credit Card Payment', 'Movies & DVDs', 'Home Improvement',
       'Utilities', 'Music', 'Mobile Phone', 'Gas & Fuel', 'Groceries',
       'Paycheck', 'Fast Food', 'Coffee Shops', 'Internet', 'Haircut',
       'Alcohol & Bars', 'Auto Insurance', 'Entertainment',
       'Food & Dining', 'Television', 'Electronics & Software'],
      dtype=object)
```

```
df_fin[df_fin.duplicated]
```

| Date | Description | Amount | Transaction Type | Category | Account Name |
|---|---|---|---|---|---|

```
df_fin['Category'] = df_fin['Category'].replace('Food & Dining', 'Fast Food')
```

```
df_fin['Date'].min(), df_fin['Date'].max()
```

```
(Timestamp('2018-01-01 00:00:00'), Timestamp('2019-09-30 00:00:00'))
```

```
df_fin['month'] = df_fin['Date'].dt.month
```

```
df_fin['year'] = df_fin['Date'].dt.year
```

```
df_fin[df_fin['Amount'] <= 0]
```

| Date | Description | Amount | Transaction Type | Category | Account Name | month | year |
|---|---|---|---|---|---|---|---|

```
df_fin['Transaction Type'].unique()
```

```
array(['debit', 'credit'], dtype=object)
```

```
df_fin[df_fin['Transaction Type'] == 'credit']['Category'].unique()
```

```
array(['Credit Card Payment', 'Paycheck'], dtype=object)
```

```
df_fin['Transaction Type'] = np.where(
    (df_fin['Category'] == 'Credit Card Payment') & (df_fin['Transaction Type'] == 'credit'),
    'debit',
    df_fin['Transaction Type']
)
```

```
df_fin[df_fin['Transaction Type'] == 'credit']['Category'].unique()
```

```
array(['Paycheck'], dtype=object)
```

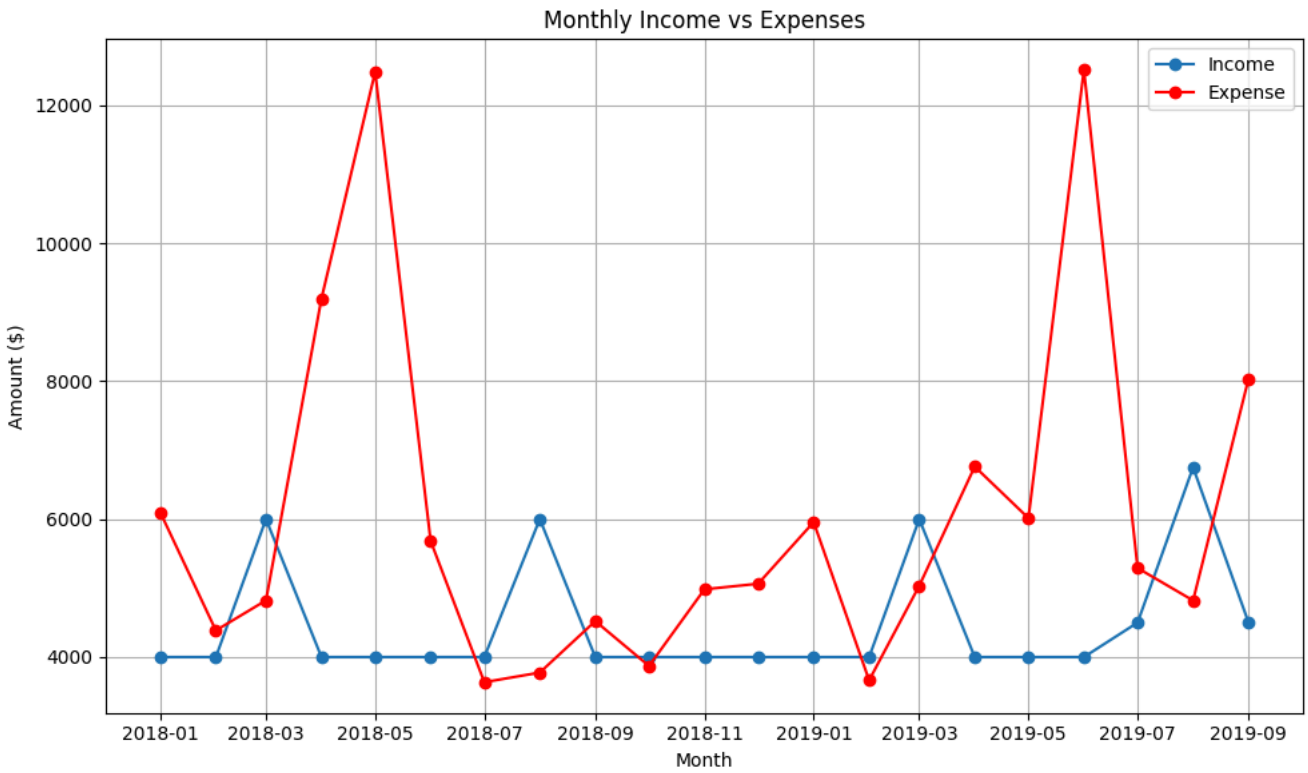
```
#TODO check some months have higher paycheck
```

```
credit_data = df_fin[df_fin['Transaction Type'] == 'credit'].groupby(['year','month']).sum('Amount')
debit_data = df_fin[df_fin['Transaction Type'] == 'debit'].groupby(['year','month']).sum('Amount')
```

```
credit_data.reset_index(inplace=True)
debit_data.reset_index(inplace=True)

credit_data['date'] = pd.to_datetime(credit_data[['year', 'month']].assign(day=1))
debit_data['date'] = pd.to_datetime(debit_data[['year', 'month']].assign(day=1))
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.plot(credit_data['date'], credit_data['Amount'], label = 'Income', marker='o')
plt.plot(debit_data['date'], debit_data['Amount'], label = 'Expense', marker='o', color='r')
plt.title('Monthly Income vs Expenses')
plt.xlabel('Month')
plt.ylabel('Amount ($)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
df_fin[(df_fin['Transaction Type'] == 'debit') & (df_fin['month'] == 10) & (df_fin['year'] == 2018)]
```

| | Date | Description | Amount | Transaction Type | Category | Account Name | month | year |
|---|---|---|---|---|---|---|---|---|
| **350** | 2018-10-01 | Starbucks | 3.00 | debit | Coffee Shops | Platinum Card | 10 | 2018 |
| **351** | 2018-10-01 | Credit Card Payment | 128.12 | debit | Credit Card Payment | Checking | 10 | 2018 |
| **352** | 2018-10-01 | Grocery Store | 15.66 | debit | Groceries | Platinum Card | 10 | 2018 |
| **353** | 2018-10-01 | Amazon | 13.13 | debit | Shopping | Platinum Card | 10 | 2018 |
| **354** | 2018-10-02 | Credit Card Payment | 124.03 | debit | Credit Card Payment | Platinum Card | 10 | 2018 |
| **355** | 2018-10-02 | Credit Card Payment | 124.03 | debit | Credit Card Payment | Checking | 10 | 2018 |
| **356** | 2018-10-02 | Mortgage Payment | 1209.18 | debit | Mortgage & Rent | Checking | 10 | 2018 |
| **357** | 2018-10-04 | Netflix | 11.76 | debit | Movies & DVDs | Platinum Card | 10 | 2018 |
| **358** | 2018-10-06 | American Tavern | 27.00 | debit | Restaurants | Silver Card | 10 | 2018 |
| **359** | 2018-10-08 | Shell | 38.06 | debit | Gas & Fuel | Silver Card | 10 | 2018 |
| **360** | 2018-10-08 | Hardware Store | 80.65 | debit | Home Improvement | Silver Card | 10 | 2018 |
| **361** | 2018-10-08 | Hardware Store | 31.20 | debit | Home Improvement | Silver Card | 10 | 2018 |
| **362** | 2018-10-09 | Spotify | 10.69 | debit | Music | Platinum Card | 10 | 2018 |
| **363** | 2018-10-09 | Amazon | 19.98 | debit | Shopping | Platinum Card | 10 | 2018 |
| **364** | 2018-10-09 | Gas Company | 30.00 | debit | Utilities | Checking | 10 | 2018 |
| **365** | 2018-10-10 | Grocery Store | 53.68 | debit | Groceries | Silver Card | 10 | 2018 |
| **366** | 2018-10-11 | Phone Company | 89.40 | debit | Mobile Phone | Checking | 10 | 2018 |
| **368** | 2018-10-16 | Power Company | 60.00 | debit | Utilities | Checking | 10 | 2018 |
| **369** | 2018-10-17 | City Water Charges | 35.00 | debit | Utilities | Checking | 10 | 2018 |
| **370** | 2018-10-18 | State Farm | 75.00 | debit | Auto Insurance | Checking | 10 | 2018 |
| **371** | 2018-10-18 | Grocery Store | 33.55 | debit | Groceries | Platinum Card | 10 | 2018 |
| **372** | 2018-10-18 | Hardware Store | 45.24 | debit | Home Improvement | Platinum Card | 10 | 2018 |
| **373** | 2018-10-18 | Brunch Restaurant | 8.00 | debit | Restaurants | Platinum Card | 10 | 2018 |
| **374** | 2018-10-21 | Credit Card Payment | 544.37 | debit | Credit Card Payment | Platinum Card | 10 | 2018 |
| **375** | 2018-10-22 | Credit Card Payment | 353.83 | debit | Credit Card Payment | Silver Card | 10 | 2018 |
| **376** | 2018-10-22 | Credit Card Payment | 353.83 | debit | Credit Card Payment | Checking | 10 | 2018 |
| **377** | 2018-10-22 | BP | 34.66 | debit | Gas & Fuel | Platinum Card | 10 | 2018 |
| **378** | 2018-10-23 | Grocery Store | 7.57 | debit | Groceries | Platinum Card | 10 | 2018 |
| **379** | 2018-10-25 | Internet Service Provider | 74.99 | debit | Internet | Checking | 10 | 2018 |
| **380** | 2018-10-25 | Amazon | 29.98 | debit | Shopping | Platinum Card | 10 | 2018 |
| **382** | 2018-10-27 | American Tavern | 25.40 | debit | Restaurants | Silver Card | 10 | 2018 |
| **383** | 2018-10-28 | Brewing Company | 12.71 | debit | Alcohol & Bars | Platinum Card | 10 | 2018 |
| **384** | 2018-10-28 | Seafood Restaurant | 14.75 | debit | Fast Food | Platinum Card | 10 | 2018 |
| **385** | 2018-10-28 | Grocery Store | 92.49 | debit | Groceries | Platinum Card | 10 | 2018 |
| **386** | 2018-10-28 | Italian Restaurant | 54.00 | debit | Restaurants | Platinum Card | 10 | 2018 |
| **387** | 2018-10-31 | Grocery Store | 5.64 | debit | Groceries | Platinum Card | 10 | 2018 |

## Alpha Vantage

```
import requests
```

```
url = 'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=IBM&apikey=1Y92HFSN1KP5GN0Q&outputsize=full'
r = requests.get(url)
data = r.json()

print(data)
```

```
{'Meta Data': {'1. Information': 'Daily Prices (open, high, low, close) and Volumes', '2. Symbol': 'IBM', '3. Last Refreshed': '2025
```

```
# Your data is already a dictionary with nested structure
# Extract the time series data
time_series = data['Time Series (Daily)']

```

```
# Convert to DataFrame
stock_data = pd.DataFrame.from_dict(time_series, orient='index')

# Convert string columns to numeric
for col in stock_data.columns:
    stock_data[col] = pd.to_numeric(stock_data[col])

# Rename columns for easier access
stock_data.columns = ['open', 'high', 'low', 'close', 'volume']

# Convert index to datetime
stock_data.index = pd.to_datetime(stock_data.index)

# Sort by date (oldest to newest)
stock_data = stock_data.sort_index()

print(stock_data.shape)
print(stock_data.head())
```

```
(6482, 5)
                open   high    low  close    volume
1999-11-01  98.50  98.81  96.37  96.75   9551800
1999-11-02  96.75  96.81  93.69  94.81  11105400
1999-11-03  95.87  95.94  93.50  94.37  10369100
1999-11-04  94.44  94.44  90.00  91.56  16697600
1999-11-05  92.75  92.94  90.19  90.25  13737600
```

```
stock_data.shape
```

```
(6482, 5)
```

```
start_date = '2022-01-01'
stock_data_3y = stock_data[stock_data.index >= start_date]
```

```
stock_data_3y.shape
```

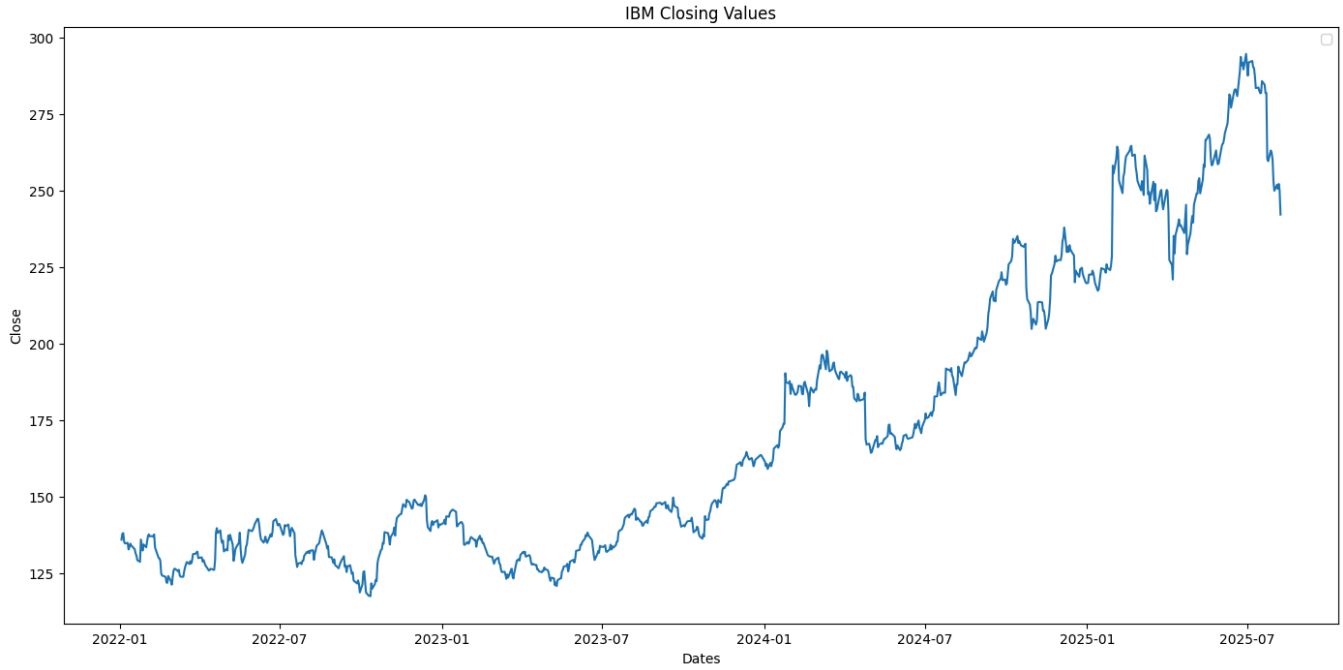
```
(903, 5)
```

```
df= stock_data_3y.copy()
```

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import lag_plot
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
```

```
# Plot the closing values for Microsoft
plt.figure(figsize=(17,8))
plt.plot(df['close'])
plt.title('IBM Closing Values')
plt.xlabel('Dates')
plt.ylabel('Close')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b630820af50>
```




```
df.tail(10)
```

| | open | high | low | close | volume |
|---|---|---|---|---|---|
| **2025-07-28** | 260.300 | 264.0000 | 259.610 | 263.21 | 5192516 |
| **2025-07-29** | 264.300 | 265.7999 | 261.020 | 262.41 | 4627265 |
| **2025-07-30** | 261.600 | 262.0000 | 258.900 | 260.26 | 3718290 |
| **2025-07-31** | 259.570 | 259.9900 | 252.220 | 253.15 | 6739092 |
| **2025-08-01** | 251.405 | 251.4791 | 245.610 | 250.05 | 9683404 |
| **2025-08-04** | 251.050 | 252.0800 | 248.110 | 251.98 | 5280588 |
| **2025-08-05** | 252.000 | 252.8000 | 248.995 | 250.67 | 5823016 |
| **2025-08-06** | 251.530 | 254.3200 | 249.280 | 252.28 | 3692105 |
| **2025-08-07** | 252.810 | 255.0000 | 248.875 | 250.16 | 6251285 |
| **2025-08-08** | 248.880 | 249.4800 | 241.650 | 242.27 | 6828390 |

```
df.index = pd.to_datetime(df.index)
full_range = pd.date_range(start=df.index.min(), end=df.index.max(), freq='D')
df_full = df.reindex(full_range)

df_full[['open','high','low','close']] = df_full[['open','high','low','close']].ffill()  # forward fill for prices
df_full['volume'] = df_full['volume'].fillna(0)  # no trades on holidays


df_full.index.name = 'date'
print(df_full.tail(10))
```

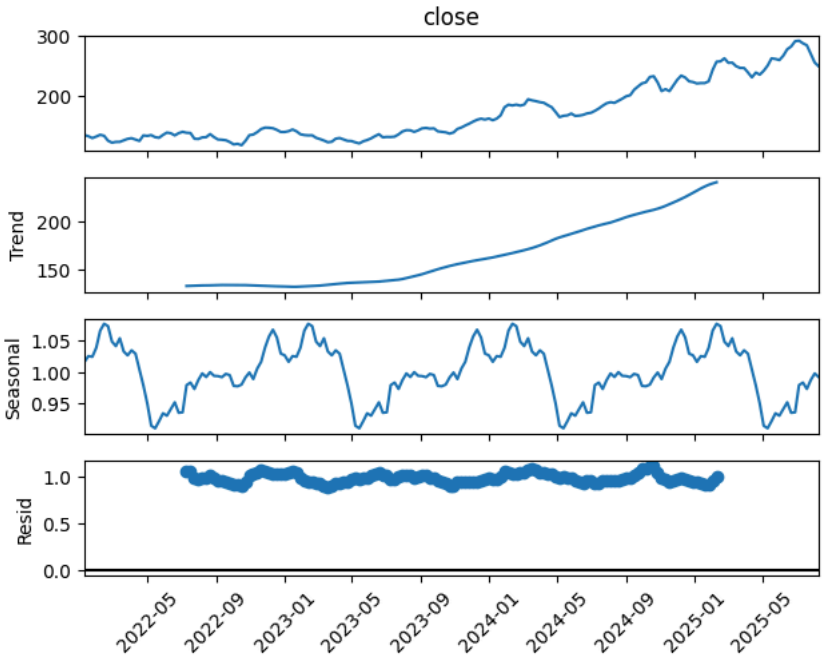
```
                open      high      low   close     volume
date
2025-07-30  261.600  262.0000  258.900  260.26  3718290.0
2025-07-31  259.570  259.9900  252.220  253.15  6739092.0
2025-08-01  251.405  251.4791  245.610  250.05  9683404.0
2025-08-02  251.405  251.4791  245.610  250.05        0.0
2025-08-03  251.405  251.4791  245.610  250.05        0.0
2025-08-04  251.050  252.0800  248.110  251.98  5280588.0
2025-08-05  252.000  252.8000  248.995  250.67  5823016.0
2025-08-06  251.530  254.3200  249.280  252.28  3692105.0
2025-08-07  252.810  255.0000  248.875  250.16  6251285.0
```

```
2025-08-08  248.880  249.4800  241.650  242.27  6828390.0
```

```
df_weekly = df_full.resample('W').mean()
```

```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df_weekly['close'], model='multiplicative')
```

```
result.plot()
plt.tight_layout()
plt.xticks(rotation=45)
plt.show()
```




## Weekly

```
url = 'https://www.alphavantage.co/query?function=TIME_SERIES_WEEKLY_ADJUSTED&symbol=IBM&apikey=1Y92HFSN1KP5GN0Q&outputsize=full'
r = requests.get(url)
data = r.json()

print(data)
```

```
{'Meta Data': {'1. Information': 'Weekly Adjusted Prices and Volumes', '2. Symbol': 'IBM', '3. Last Refreshed': '2025-08-08', '4. Ti
```

```
time_series = data['Weekly Adjusted Time Series']
```

```
stock_data = pd.DataFrame.from_dict(time_series, orient='index')
```

```
for col in stock_data.columns:
    stock_data[col] = pd.to_numeric(stock_data[col])
```

```
stock_data.columns = ['open', 'high', 'low', 'close', 'adjusted_close', 'volume', 'dividend_amount']
```

```
stock_data.index = pd.to_datetime(stock_data.index)
```

```
stock_data = stock_data.sort_index()
```

```
# Calculate daily returns
stock_data['returns'] = stock_data['adjusted_close'].pct_change()

# Annualized return (mean * trading days)
annual_return = stock_data['returns'].mean() * 252

# Annualized volatility (std * sqrt(trading days))
annual_volatility = stock_data['returns'].std() * (252 ** 0.5)

print(f"Annual Return: {annual_return:.2%}")
```

```
print(f"Annual Volatility: {annual_volatility:.2%}")

```

```
Annual Return: 46.40%
Annual Volatility: 56.34%
```

```
def classify_stock(ret, vol):
    if ret >= 0.15 and vol <= 0.20:
        return "High Return / Low Risk"
    elif ret >= 0.15 and vol > 0.20:
        return "High Return / High Risk"
    elif ret < 0.15 and vol <= 0.20:
        return "Low Return / Low Risk"
    else:
        return "Low Return / High Risk"

category = classify_stock(annual_return, annual_volatility)
print("Category:", category)

```

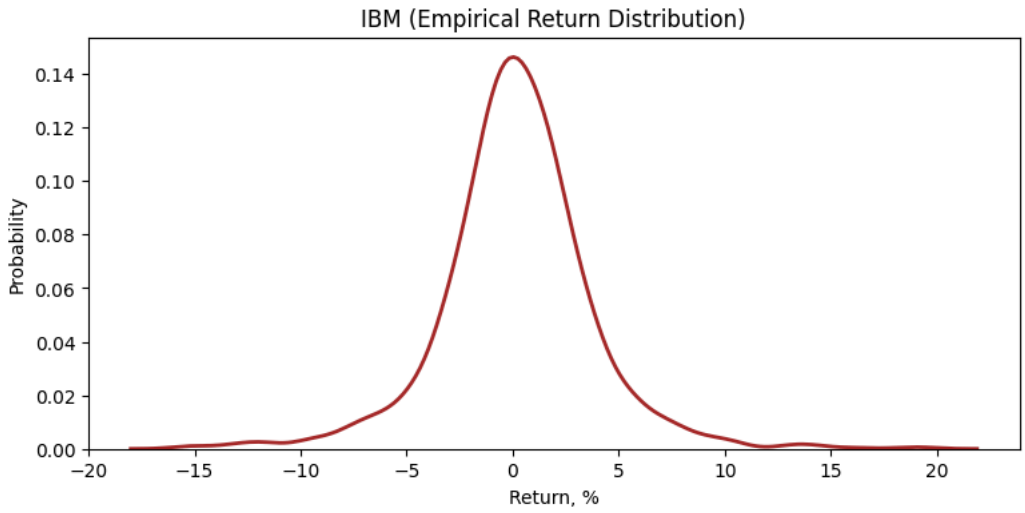
```
Category: High Return / High Risk
```

```
returns = stock_data['returns'].dropna()
```

```
plt.figure(figsize=(9,4))
sns.kdeplot(returns * 100, color='brown', linewidth=2)
plt.xlabel("Return, %")
plt.ylabel("Probability")
plt.title("IBM (Empirical Return Distribution)")


```

```
Text(0.5, 1.0, 'IBM (Empirical Return Distribution)')
```




```
Start coding or generate with AI.
```

## Insurance Dataset

```
ins_data = pd.read_csv('insurance_dataset.csv')
ins_data.shape
```

```
(1000000, 12)
```

```
ins_data.head()
```

| | age | gender | bmi | children | smoker | region | medical_history | family_medical_history | exercise_frequency | occupation | coverage_ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 46 | male | 21.45 | 5 | yes | southeast | Diabetes | NaN | Never | Blue collar | Pr |
| **1** | 25 | female | 25.38 | 2 | yes | northwest | Diabetes | High blood pressure | Occasionally | White collar | Pr |
| **2** | 38 | male | 44.88 | 2 | yes | southwest | NaN | High blood pressure | Occasionally | Blue collar | Pr |
| **3** | 25 | male | 19.89 | 0 | no | northwest | NaN | Diabetes | Rarely | White collar | St |
| **4** | 49 | male | 38.21 | 3 | yes | northwest | Diabetes | High blood pressure | Rarely | White collar | St |

```
ins_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 12 columns):
 #   Column                  Non-Null Count    Dtype
---  ------                  --------------    -----
 0   age                     1000000 non-null  int64
 1   gender                  1000000 non-null  object
 2   bmi                     1000000 non-null  float64
 3   children                1000000 non-null  int64
 4   smoker                  1000000 non-null  object
 5   region                  1000000 non-null  object
 6   medical_history         749238 non-null   object
 7   family_medical_history  749596 non-null   object
 8   exercise_frequency      1000000 non-null  object
 9   occupation              1000000 non-null  object
 10  coverage_level          1000000 non-null  object
 11  charges                 1000000 non-null  float64
dtypes: float64(2), int64(2), object(8)
memory usage: 91.6+ MB
```

```
ins_data.medical_history.unique()
```

```
array(['Diabetes', nan, 'High blood pressure', 'Heart disease'],
      dtype=object)
```

```
ins_data.family_medical_history.unique()
```

```
array([nan, 'High blood pressure', 'Diabetes', 'Heart disease'],
      dtype=object)
```

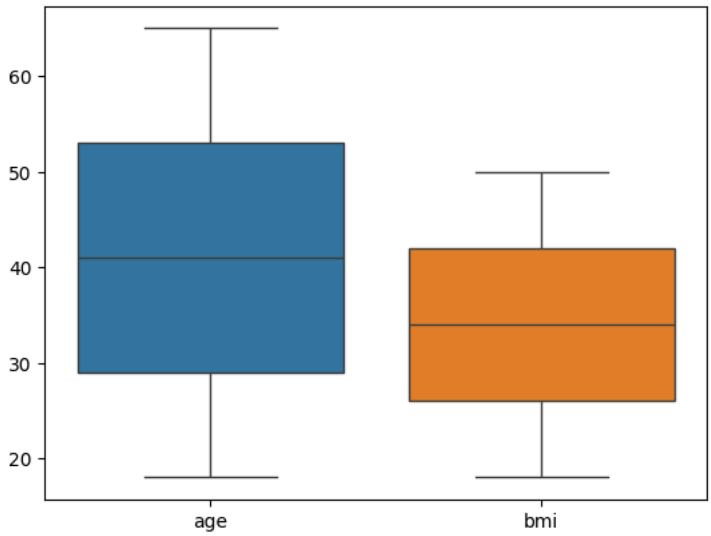
```
ins_data['medical_history'] = ins_data['medical_history'].fillna('No Record')
ins_data['family_medical_history'] = ins_data['family_medical_history'].fillna('No Record')
```

```
ins_data.describe()
```

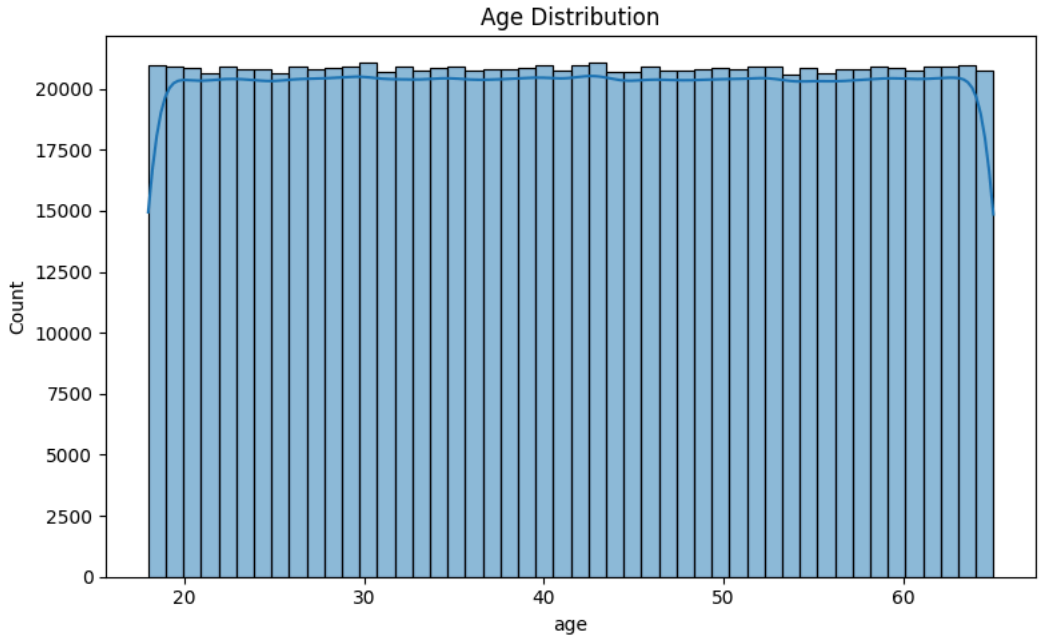
| | age | bmi | children | charges |
|---|---|---|---|---|
| **count** | 1000000.000000 | 1000000.000000 | 1000000.000000 | 1000000.000000 |
| **mean** | 41.495282 | 34.001839 | 2.499886 | 16735.117481 |
| **std** | 13.855189 | 9.231680 | 1.707679 | 4415.808211 |
| **min** | 18.000000 | 18.000000 | 0.000000 | 3445.011643 |
| **25%** | 29.000000 | 26.020000 | 1.000000 | 13600.372379 |
| **50%** | 41.000000 | 34.000000 | 2.000000 | 16622.127973 |
| **75%** | 53.000000 | 41.990000 | 4.000000 | 19781.465410 |
| **max** | 65.000000 | 50.000000 | 5.000000 | 32561.560374 |

```
import seaborn as sns
sns.boxplot(ins_data[['age','bmi']])
```

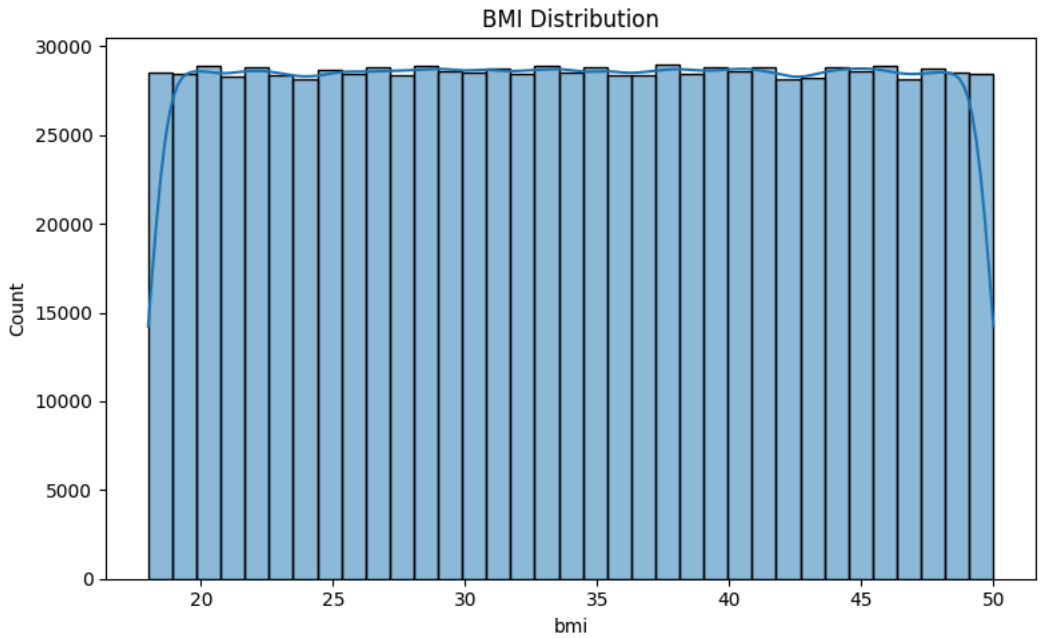
```
<Axes: >
```



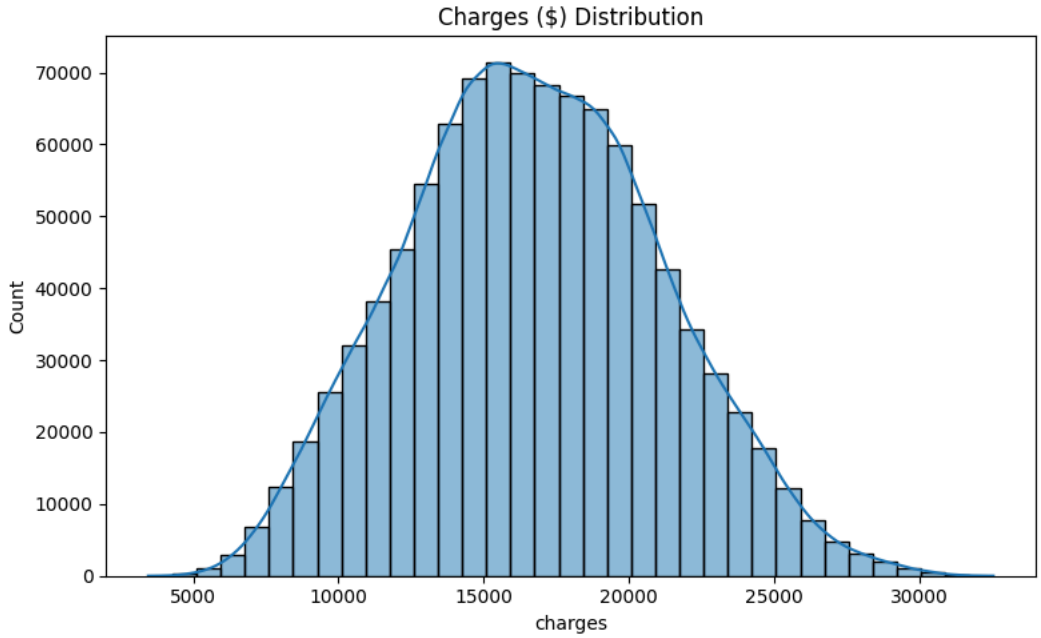
```
plt.figure(figsize=(8,5))
sns.histplot(data=ins_data, x="age", kde=True, bins=48)
plt.title("Age Distribution")
plt.tight_layout()
plt.show()
```



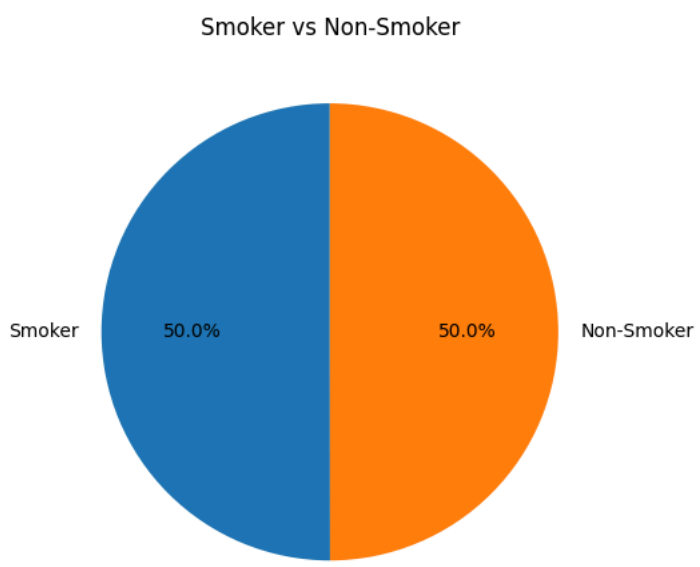

```
plt.figure(figsize=(8,5))
sns.histplot(data=ins_data, x="bmi", kde=True, bins=35)
plt.title("BMI Distribution")
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(8,5))
sns.histplot(data=ins_data, x="charges", kde=True, bins=35)
plt.title("Charges ($) Distribution")
plt.tight_layout()
plt.show()
```




```
smoker_labels = ins_data['smoker'].map(lambda x: 'Smoker' if x == 'yes' else 'Non-Smoker')
smoker_counts = smoker_labels.value_counts()
plt.pie(smoker_counts, labels=smoker_counts.index, autopct='%1.1f%%', startangle=90)
plt.title("Smoker vs Non-Smoker")
plt.tight_layout()
plt.show()
```



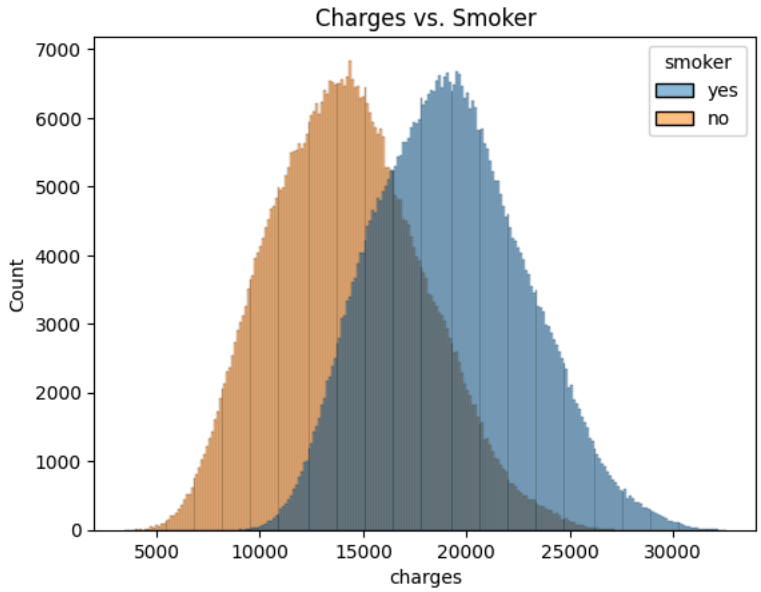

```
smoker_counts
```

| smoker | count |
|---|---|
| **Smoker** | 500129 |
| **Non-Smoker** | 499871 |

**dtype:** int64

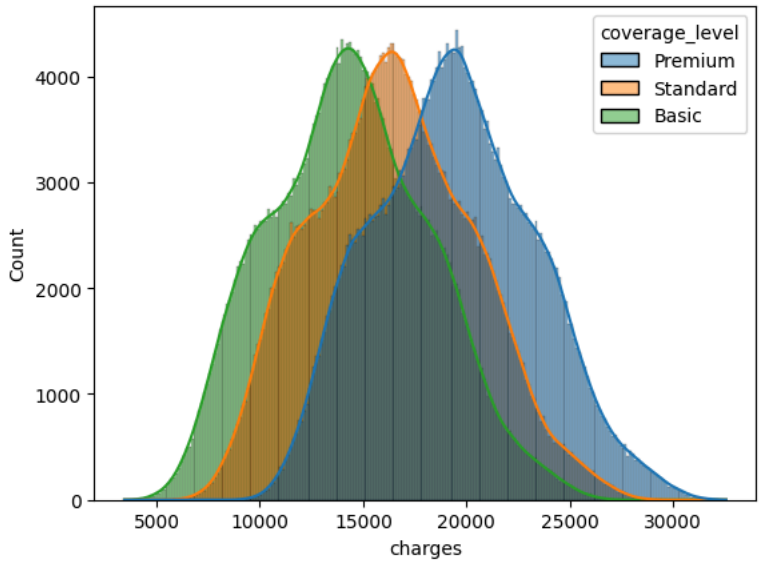
```
fig = sns.histplot(ins_data,
                   x='charges',
                   hue='smoker'
)
plt.title('Charges vs. Smoker')
```

```
Text(0.5, 1.0, 'Charges vs. Smoker')
```



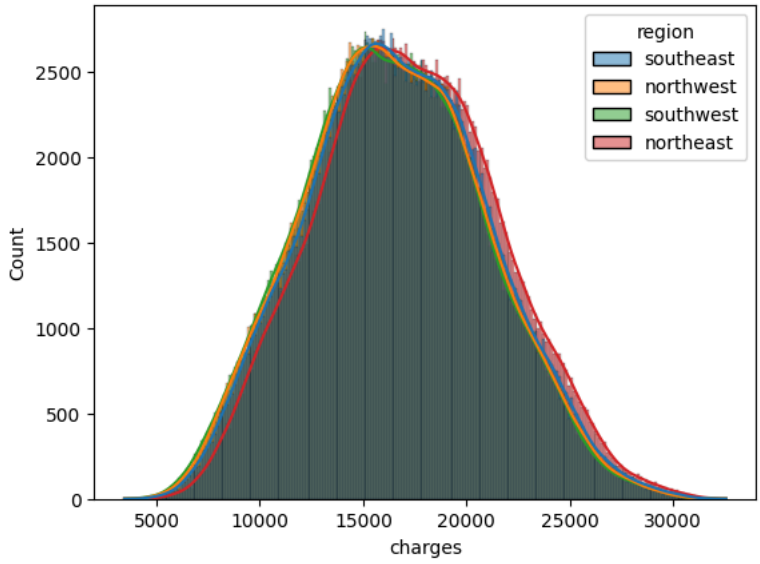

```
sns.histplot(data=ins_data,x='charges',hue='coverage_level', kde=True)
```

```
<Axes: xlabel='charges', ylabel='Count'>
```



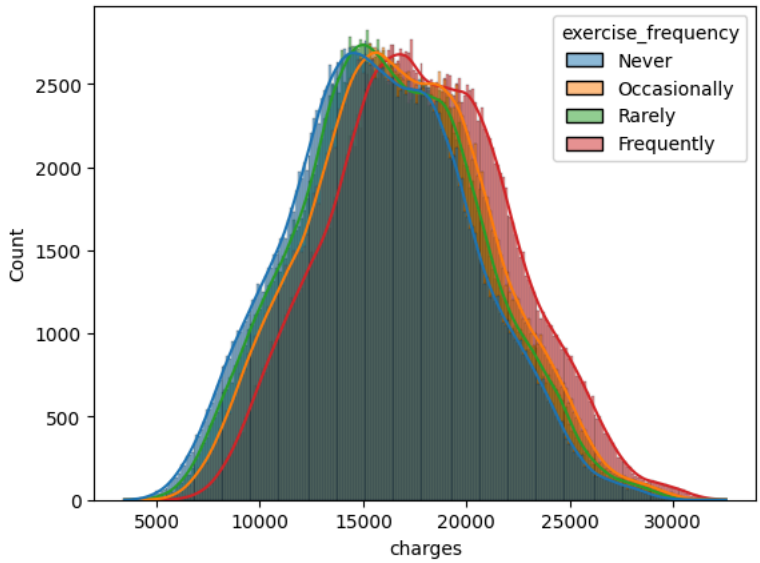

```
sns.histplot(data=ins_data,x='charges',hue='region', kde=True)
```

```
<Axes: xlabel='charges', ylabel='Count'>
```



```
sns.histplot(data=ins_data,x='charges',hue='exercise_frequency', kde=True)
```

```
<Axes: xlabel='charges', ylabel='Count'>
```




```
ins_data[ins_data.duplicated]
```

| | age | gender | bmi | children | smoker | region | medical_history | family_medical_history | exercise_frequency | occupation | coverage_level |
|---|---|---|---|---|---|---|---|---|---|---|---|

```
import pandas as pd

df_encoded = pd.get_dummies(ins_data, columns=['smoker', 'region', 'gender', 'medical_history','family_medical_history','exercise_fr
```

```
cols = [c for c in df_encoded.columns if c != 'charges'] + ['charges']
df_encoded = df_encoded[cols]
```

```
plt.figure(figsize=(30,25))
sns.heatmap(df_encoded.corr(), cmap='coolwarm', annot=True, fmt='0.2f' )
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.show()
```