```python
import pandas as pd
import numpy as np

df=pd.read_excel(r"D:\download\data (1).xlsx")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
Cell In[1], line 1
----> 1 df=pd.read_excel(r"D:\download\data (1).xlsx")

NameError: name 'pd' is not defined
```

```python
df.head(5)
```

```
          Income  Age  Dependents     Occupation City_Tier
expenses  \
0   44637.249636   49           0  Self_Employed    Tier_1
33371.621929
1   26858.596592   34           2        Retired    Tier_2
17181.777859
2   50367.605084   35           1        Student    Tier_3
36476.154459
3  101455.600247   21           0  Self_Employed    Tier_3
69837.646632
4   24875.283548   52           4   Professional    Tier_2
18609.583016

   Loan_Repayment    Insurance     Groceries    Transport  ...  \
0        0.000000  2206.490129   6658.768341  2636.970696  ...
1        0.000000   869.522617   2818.444460  1543.018778  ...
2     4612.103386  2201.800050   6313.222081  3221.396403  ...
3     6809.441427  4889.418087  14690.149363  7106.130005  ...
4     3112.609398   635.907170   3034.329665  1276.155163  ...

   Desired_Savings  Disposable_Income  Potential_Savings_Groceries  \
0      6200.537192       11265.627707                  1685.696222
1      1923.176434        9676.818733                   540.306561
2      7050.360422       13891.450624                  1466.073984
3     16694.965136       31617.953615                  1875.932770
4      1874.099434        6265.700532                   788.953124

   Potential_Savings_Transport  Potential_Savings_Eating_Out  \
0                   328.895281                    465.769172
1                   119.347139                    141.866089
2                   473.549752                    410.857129
3                   762.020789                   1241.017448
4                    68.160766                     61.712505

   Potential_Savings_Entertainment  Potential_Savings_Utilities  \
0                        195.151320                             678.292859
1                        234.131168                             286.668408
2                        459.965256                             488.383423
3                        320.190594                            1389.815033
4                        187.173750                             194.117130

   Potential_Savings_Healthcare  Potential_Savings_Education  \
0                     67.682471                     0.000000
1                      6.603212                    56.306874
2                      7.290892                   106.653597
3                    193.502754                     0.000000
4                     47.294591                    67.388120

   Potential_Savings_Miscellaneous
0                        85.735517
1                        97.388606
2                       138.542422
3                       296.041183
4                        96.557076

[5 rows x 28 columns]
```

```
df.shape
```

```
(20000, 28)
```

```
df.columns
```

```
Index(['Income', 'Age', 'Dependents', 'Occupation', 'City_Tier',
'expenses',
       'Loan_Repayment', 'Insurance', 'Groceries', 'Transport',
'Eating_Out',
       'Entertainment', 'Utilities', 'Healthcare', 'Education',
       'Miscellaneous', 'Unnamed: 16', 'Desired_Savings_Percentage',
       'Desired_Savings', 'Disposable_Income',
'Potential_Savings_Groceries',
       'Potential_Savings_Transport', 'Potential_Savings_Eating_Out',
       'Potential_Savings_Entertainment',
'Potential_Savings_Utilities',
       'Potential_Savings_Healthcare', 'Potential_Savings_Education',
       'Potential_Savings_Miscellaneous'],
      dtype='object')
```

```
x=df.iloc[:,0:6].values
```

```
y=df.iloc[:,18].values
```

```
x
```

```
array([[44637.2496356859, 49, 0, 'Self_Employed', 'Tier_1',
        33371.621928834255],
       [26858.5965917295, 34, 2, 'Retired', 'Tier_2',
17181.777859045094],
       [50367.6050835768, 35, 1, 'Student', 'Tier_3',
36476.15445928645],
       ...,
       [40604.5673726763, 30, 1, 'Professional', 'Tier_2',
        38336.66223874504],
       [118157.817239995, 27, 2, 'Professional', 'Tier_1',
        107554.13242645186],
       [8209.24976874268, 62, 3, 'Professional', 'Tier_1',
        7348.899209965442]], dtype=object)
```

```
y
```

```
array([ 6200.53719244,  1923.1764339 ,  7050.36042169, ...,
        2267.90513393, 10603.68481354,   531.04400553])
```

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

```
Income                                0
Age                                   0
Dependents                            0
Occupation                            0
City_Tier                             0
expenses                              0
Loan_Repayment                        0
Insurance                             0
Groceries                             0
Transport                             0
Eating_Out                            0
Entertainment                         0
Utilities                             0
Healthcare                            0
Education                             0
Miscellaneous                         0
Unnamed: 16                       20000
Desired_Savings_Percentage            0
Desired_Savings                       0
Disposable_Income                     0
Potential_Savings_Groceries           0
Potential_Savings_Transport           0
Potential_Savings_Eating_Out          0
Potential_Savings_Entertainment       0
Potential_Savings_Utilities           0
Potential_Savings_Healthcare          0
Potential_Savings_Education           0
Potential_Savings_Miscellaneous       0
dtype: int64
```

```
from sklearn.preprocessing import OrdinalEncoder
```

```python
from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
m_state=0)

transformer = ColumnTransformer(transformers=[
    ('encoder1', OrdinalEncoder(categories=[['Tier_3', 'Tier_2',
'Tier_1']]), [4]),
    ('encoder2', OneHotEncoder(drop="first"), [3])
], remainder='passthrough')

# Transform training and test data
X_train_transformed = transformer.fit_transform(x_train)
X_test_transformed = transformer.transform(x_test)

X_train_transformed
```

```
array([[2.0, 0.0, 1.0, ..., 36, 2, 34140.31020457392],
       [2.0, 0.0, 0.0, ..., 47, 4, 10244.395427778194],
       [1.0, 0.0, 0.0, ..., 35, 3, 98551.5712948812],
       ...,
       [0.0, 0.0, 0.0, ..., 31, 2, 15960.713927009849],
       [1.0, 0.0, 1.0, ..., 26, 4, 61929.80381889947],
       [2.0, 0.0, 0.0, ..., 43, 0, 37620.856361997394]], dtype=object)
```

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_transformed)
X_test_scaled = scaler.transform(X_test_transformed)

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

```
LinearRegression()
```

```python
y_pred = model.predict(X_test_scaled)

y_pred
```

```
array([ -241.64897336,  5807.56395151,  -138.19172317, ...,
        5907.43522445,  6493.56218864, 10084.07977449])
```

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("MSE:", mse)
print("MAE:", mae)
print("R² Score:", r2)
```

```
MSE: 4648174.7459285725
MAE: 1407.369295311036
R² Score: 0.9142419187040344
```

```
from sklearn.linear_model import RidgeCV, LassoCV
lasso = LassoCV(alphas=[0.01, 0.1, 1.0, 10.0], cv=5)
lasso.fit(X_train_scaled, y_train)
lasso_pred = lasso.predict(X_test_scaled)

print("\n Lasso Regression Results:")
print("Best alpha:", lasso.alpha_)
print("R2 Score:", r2_score(y_test, lasso_pred))
```

```
 Lasso Regression Results:
Best alpha: 0.01
R2 Score: 0.9142416336019124
```

```
import xgboost as xgb

model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1,
max_depth=3, reg_alpha=0, reg_lambda=1)
model.fit(X_train_scaled, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None,
early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None,
feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.1, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=3,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None,
n_estimators=100,
             n_jobs=None, num_parallel_tree=None, ...)
```

```
y_pred = model.predict(X_test_scaled)
r2 = r2_score(y_test, y_pred)

y_pred
r2
```

```
0.9148043147146206
```

```
y_pred
```

```
array([1069.4144, 5950.5566,  907.5372, ..., 5377.032 , 5762.297 ,
       7727.817 ], dtype=float32)
```

```
r2
```

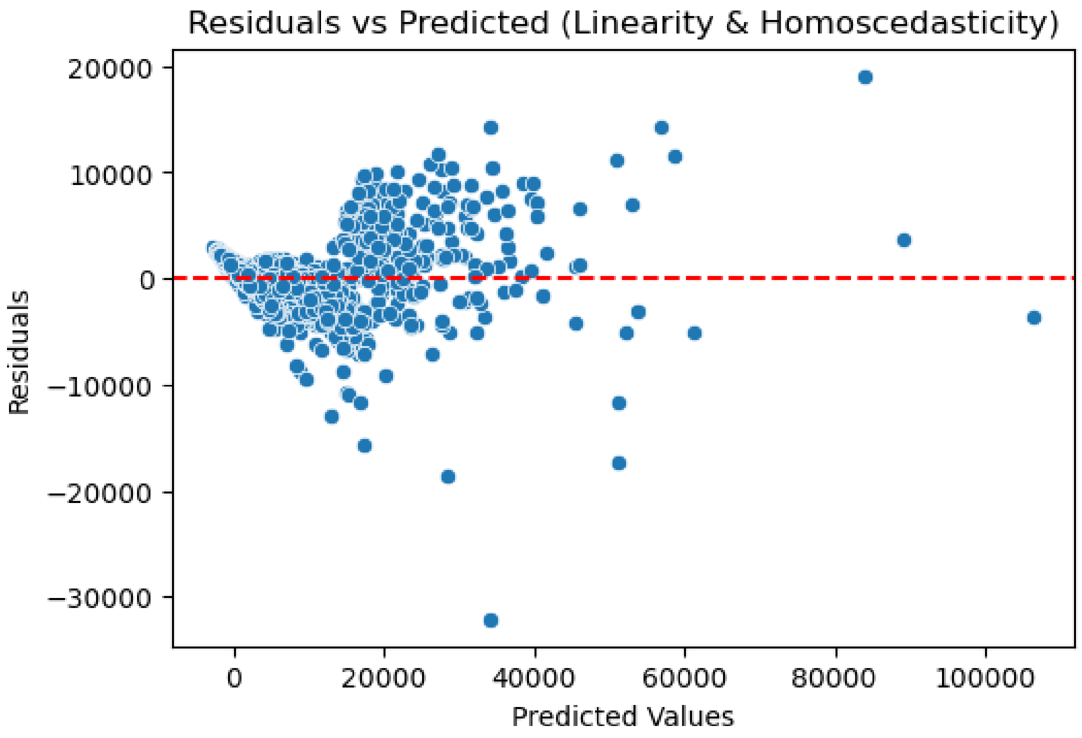
```
0.9148043147146206
```

```python
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Define parameter grid
param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}

# ----- Ridge Regression -----
ridge = Ridge()
ridge_cv = GridSearchCV(ridge, param_grid, cv=5, scoring='r2')
ridge_cv.fit(X_train_scaled, y_train)

# Predict and evaluate Ridge
ridge_pred = ridge_cv.predict(X_test_scaled)
ridge_r2 = r2_score(y_test, ridge_pred)
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_pred))

# ----- Lasso Regression -----
lasso = Lasso(max_iter=10000)
lasso_cv = GridSearchCV(lasso, param_grid, cv=5, scoring='r2')
lasso_cv.fit(X_train_scaled, y_train)

# Predict and evaluate Lasso
lasso_pred = lasso_cv.predict(X_test_scaled)
lasso_r2 = r2_score(y_test, lasso_pred)
lasso_rmse = np.sqrt(mean_squared_error(y_test, lasso_pred))

# ----- Print Results -----
print("□ Ridge Regression:")
print("Best alpha:", ridge_cv.best_params_['alpha'])
print("R² Score:", ridge_r2)
print("RMSE:", ridge_rmse)

print("\n□ Lasso Regression:")
print("Best alpha:", lasso_cv.best_params_['alpha'])
print("R² Score:", lasso_r2)
print("RMSE:", lasso_rmse)
```

```
□ Ridge Regression:
Best alpha: 1
R² Score: 0.9142220221090633
RMSE: 2156.212689300464
```

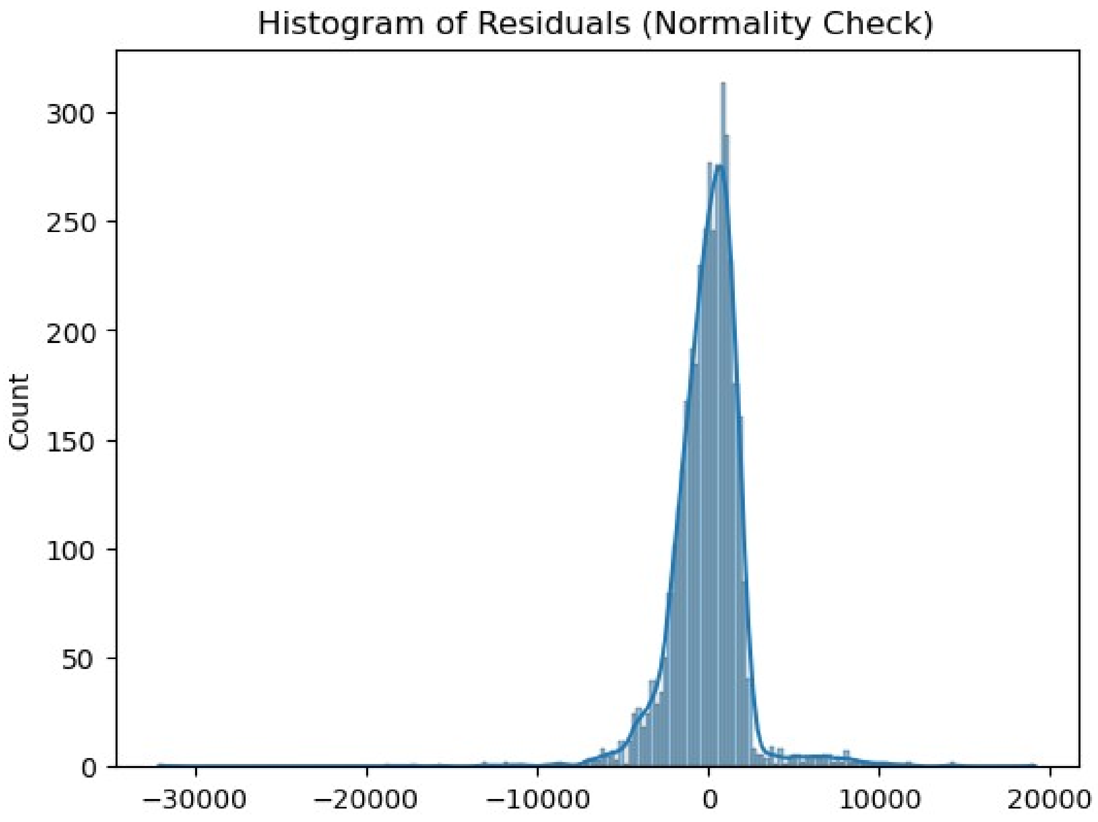
```
 Lasso Regression:
Best alpha: 0.1
R² Score: 0.9142390374803955
RMSE: 2155.998819842906

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

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.stats.outliers_influence import
variance_inflation_factor
from statsmodels.stats.stattools import durbin_watson
from statsmodels.stats.diagnostic import het_breuschpagan
from scipy.stats import shapiro
import statsmodels.api as sm

# Train model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Predict
y_pred = model.predict(X_test_scaled)
residuals = y_test - y_pred

# 1. Linearity & Homoscedasticity (Residuals vs Predicted)
plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted (Linearity & Homoscedasticity)")
plt.show()

# 2. Independence (Durbin-Watson)
dw = durbin_watson(residuals)
print(f" Durbin-Watson Statistic: {dw:.3f} (≈2 is ideal)")

# 3. Homoscedasticity (Breusch-Pagan Test)
X_test_const = sm.add_constant(X_test_scaled)
bp_test = het_breuschpagan(residuals, X_test_const)
labels = ['LM stat', 'LM p-value', 'F stat', 'F p-value']
print("\n Breusch-Pagan Test Results:")
print(dict(zip(labels, bp_test)))

# 4. Normality of Residuals
# Histogram
sns.histplot(residuals, kde=True)
```

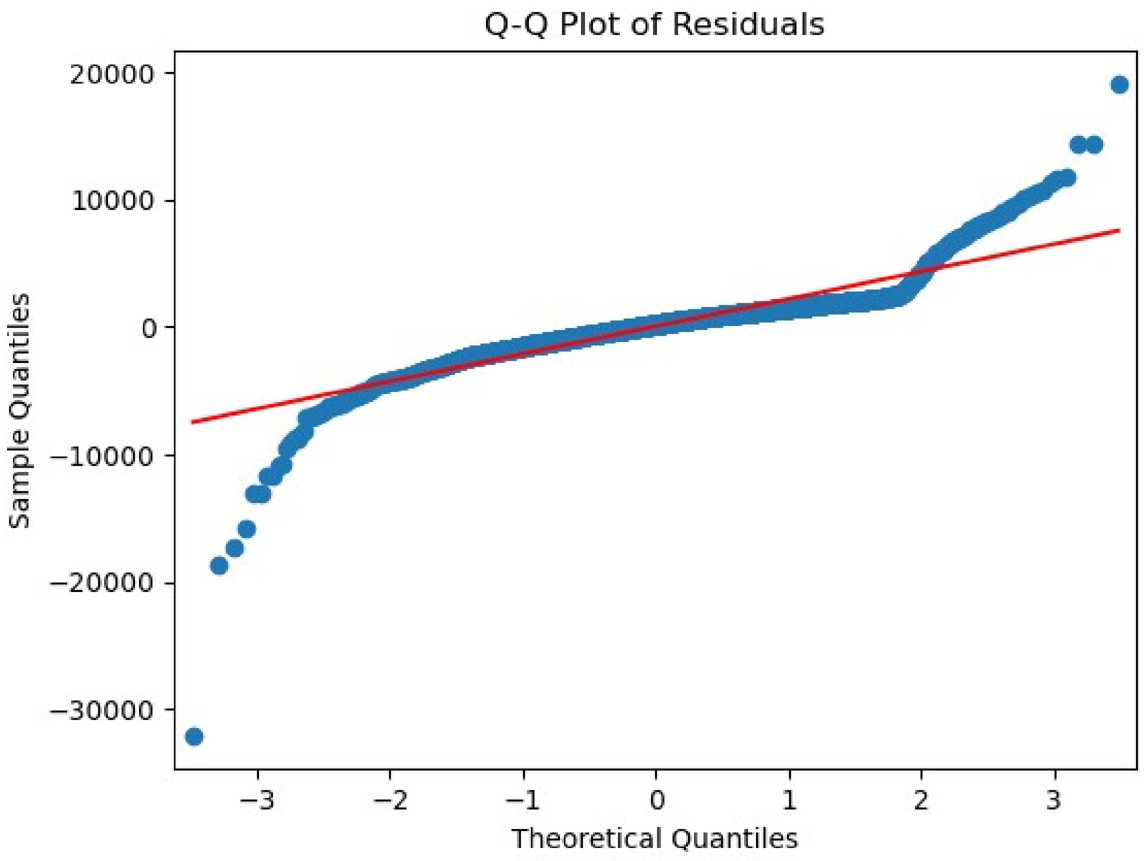
```
plt.title("Histogram of Residuals (Normality Check)")
plt.show()

# Q-Q Plot
sm.qqplot(residuals, line='s')
plt.title("Q-Q Plot of Residuals")
plt.show()

# Shapiro-Wilk Test
shapiro_stat, shapiro_p = shapiro(residuals)
print(f"\n Shapiro-Wilk Test p-value: {shapiro_p:.4f} (p > 0.05 ⇒ 
Normal)")

# 5. Multicollinearity (VIF)
vif_data = pd.DataFrame()
vif_data["Feature"] = [f"X{i+1}" for i in 
range(X_train_scaled.shape[1])]
vif_data["VIF"] = [variance_inflation_factor(X_train_scaled, i) for i 
in range(X_train_scaled.shape[1])]
print("\n Variance Inflation Factor (VIF):")
print(vif_data)
```



```
 Durbin-Watson Statistic: 2.003 (≈2 is ideal)

 Breusch-Pagan Test Results:
```

```
{'LM stat': 1041.7415252144006, 'LM p-value': 1.4562785548263177e-219,
'F stat': 175.67728033940617, 'F p-value': 8.205837421306955e-255}
```





```
 Shapiro-Wilk Test p-value: 0.0000 (p > 0.05 ⇒ Normal)

 Variance Inflation Factor (VIF):
  Feature        VIF
0      X1   1.208422
1      X2   1.498286
2      X3   1.496220
3      X4   1.496323
4      X5  36.984607
5      X6   1.000397
6      X7   1.031433
7      X8  37.198792

import pandas as pd
from statsmodels.stats.outliers_influence import
variance_inflation_factor

# Select only the first 6 features
X_train_top6 = X_train_scaled[:, 0:6]

# VIF calculation
vif_data = pd.DataFrame()
```

```
vif_data["Feature"] = [f"X{i+1}" for i in range(6)]
vif_data["VIF"] = [variance_inflation_factor(X_train_top6, i) for i in
range(6)]

print("□ VIF for First 6 Variables:")
print(vif_data)
```

```
□ VIF for First 6 Variables:
  Feature       VIF
0      X1  1.000197
1      X2  1.498282
2      X3  1.496084
3      X4  1.496309
4      X5  1.000348
5      X6  1.000313
```