bivariate_tests.py

```
import os
import pandas as pd
import numpy as np
from scipy.stats import shapiro, spearmanr, kruskal
import warnings
from setup import load_and_clean_games

# set working directory
os.chdir(os.path.dirname(os.path.abspath(__file__)))

def print_bivariate_tests(cleaned_games):
    warnings.filterwarnings("ignore")  # ignore warning for corr tests

    # quantitative variables
    numeric_cols = cleaned_games[[
    "Average playtime forever", "Peak CCU", "Price", "Recommendations",
    "Required age", "Positive", "Negative", "total_reviews", "positive_ratio"
    ]].copy()

    # log10(x+1) to avoid infinite values
    numeric_cols_log = np.log10(numeric_cols + 1)

    # normality test with lilie
    lillie_table = pd.DataFrame({
    "p-value lillie": [
    shapiro(numeric_cols_log[col].dropna())[1]
    for col in numeric_cols_log.columns
    ]
    }, index=numeric_cols_log.columns)
```

```
print("lillie test (log10 transformed variables):")
print(lillie_table)

# spearman correlation with average.playtime.forever
spearman_results = {}
rho_values = {}

for col in numeric_cols.columns:
if col != "Average playtime forever":
rho, pval = spearmanr(numeric_cols["Average playtime forever"], numeric_cols[col], nan_policy="omit")
spearman_results[col] = pval
rho_values[col] = rho

spearman_table = pd.DataFrame({
"p-value spearman": pd.Series(spearman_results),
"rho (Spearman)": pd.Series(rho_values)
})
print("\nSpearman Correlations with Average playtime forever:")
print(spearman_table)

# kruskal-wallis tests for qualitative variables
quali_cols = ["Estimated owners", "rating"]
kruskal_results = []

for col in quali_cols:
if col in cleaned_games.columns:
groups = [group["Average playtime forever"].dropna() for _, group in cleaned_games.groupby(col)]
if len(groups) > 1:
stat, pval = kruskal(*groups)
kruskal_results.append({
```

```
                "Test": col,
                "H_statistic": stat,
                "p_value": pval
            })

        kruskal_table = pd.DataFrame(kruskal_results)
        print("\nKruskal-Wallis Test Results:")
        print(kruskal_table)

        return lillie_table, spearman_table, kruskal_table

if __name__ == "__main__":
    filepath = "../steam_data/games.csv"
    games = load_and_clean_games(filepath)
    # print(games.head())
    cleaned_games = games[[
    "Average playtime forever", "Estimated owners",
    "Peak CCU", "Price", "Recommendations", "Required age",
    "Positive", "Negative", "total_reviews", "positive_ratio"
    ]]
    print_bivariate_tests(cleaned_games)
```

classification.py

```
import os
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import LabelEncoder, StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import norm
```

```
from setup import load_and_clean_games, clean_column_names

# load and clean the dataset, select variables, encode target variable
def preprocess_data(filepath):
	games = load_and_clean_games(filepath)
	gamesc = games[[
	"Average playtime forever", "Estimated owners",
	"Peak CCU", "Price", "Recommendations", "Required age",
	"Positive", "Negative"
	]]
	gamesc = clean_column_names(gamesc).dropna()

	# rename estimated owners to more readable labels
	categories = {
	"0 - 20000": "0-20k", "20000 - 50000": "20k-50k", "50000 - 100000": "50k-100k",
	"100000 - 200000": "100k-200k", "200000 - 500000": "200k-500k", "500000 - 
1000000": "500k-1M",
	"1000000 - 2000000": "1M-2M", "2000000 - 5000000": "2M-5M", "5000000 - 
10000000": "5M-10M",
	"10000000 - 20000000": "10M-20M", "20000000 - 50000000": "20M-50M",
	"50000000 - 100000000": "50M-100M", "100000000 - 200000000": "100M-200M"
	}
	gamesc['estimated_owners'] = gamesc['estimated_owners'].map(categories)
	le = LabelEncoder()
	gamesc['estimated_owners'] = le.fit_transform(gamesc['estimated_owners'])

	return gamesc



# transform and scale the selected features, return both logged and standardized versions
def prepare_features(gamesc, X):
	X_data = gamesc[X]
```

```
    X_data_log = np.log1p(X_data).clip(lower=0, upper=100)
    X_data_log = sm.add_constant(X_data_log)

    # standardize
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_data_log.drop(columns='const'))
    X_data_scaled = np.column_stack([np.ones(X_scaled.shape[0]), X_scaled])
    return X_data_log, X_data_scaled


def fit_multinomial_logit(Y, X_scaled):
    model = sm.MNLogit(Y, X_scaled)
    return model.fit()

# display model coefficients, p-values, AIC, and pseudo R2
def print_model_summary(model_fit):
    print(model_fit.summary())

    # p-values
    z = model_fit.params / model_fit.bse
    p_values = 2 * (1 - norm.cdf(np.abs(z)))
    print("\n--- P-values of coefficients ---")
    print(np.round(p_values, 4))

    # AIC and pseudo R2
    print("\n--- AIC ---")
    print(model_fit.aic)
    print("\n--- Pseudo R² ---")
    print(model_fit.prsquared)
```

```
def compute_vif(X_data_log, X_vars):
    X_no_const = X_data_log.drop(columns='const')
    vif_data = pd.DataFrame()
    vif_data["Variable"] = X_vars
    vif_data["VIF"] = [variance_inflation_factor(X_no_const.values, i)
        for i in range(X_no_const.shape[1])]
    print("\n--- VIF (multicollinearity) ---")
    print(vif_data)

# confusion matrix and accuracy
def evaluate_model(model_fit, X_scaled, Y_true):
    pred = model_fit.predict(X_scaled)
    pred_class = np.argmax(pred, axis=1)
    conf_matrix = pd.crosstab(pred_class, Y_true)
    print("\n--- Confusion Matrix ---")
    print(conf_matrix)

    accuracy = np.mean(pred_class == Y_true)
    print("\n--- Accuracy ---")
    print(np.round(accuracy, 4))


def run_classification(filepath, X_vars):
    gamesc = preprocess_data(filepath)
    X_data_log, X_scaled = prepare_features(gamesc, X_vars)
    model_fit = fit_multinomial_logit(gamesc['estimated_owners'], X_scaled)
    print_model_summary(model_fit)
    compute_vif(X_data_log, X_vars)
    evaluate_model(model_fit, X_scaled, gamesc['estimated_owners'])
```

```
if __name__ == "__main__":
    os.chdir(os.path.dirname(os.path.abspath(__file__)))
    filepath = "../steam_data/games.csv"
    variables = ['positive', 'peak_ccu']
    run_classification(filepath, variables)
```

correlation_matrix.py

```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from setup import load_and_clean_games

# set working directory
os.chdir(os.path.dirname(os.path.abspath(__file__)))

def print_correlation_matrix(cleaned_games):
	# keep numeric variables
	numeric_vars = cleaned_games.select_dtypes(include=[np.number])

	# spearman correlation (because no linearity)
	cor_matrix = numeric_vars.corr(method='spearman')
	print("Spearman Correlation Matrix:")
	print(cor_matrix)

	# Heatmap type corrplot
	plt.figure(figsize=(10, 8))
	mask = np.triu(np.ones_like(cor_matrix, dtype=bool))  # hide lower triangle
	cmap = sns.diverging_palette(240, 10, as_cmap=True)
	sns.heatmap(cor_matrix, mask=mask, cmap=cmap, center=0, annot=True,
	fmt=".2f", square=True, linewidths=.5, cbar_kws={"shrink": .8})
	plt.title("Spearman Correlation Matrix (upper triangle)")
	plt.show()

	# other heatmap style
	plt.figure(figsize=(10, 8))
```

```
    sns.heatmap(cor_matrix, cmap="coolwarm", annot=True, fmt=".2f",
    square=True, cbar=True, linewidths=.5)
    plt.title("Spearman Correlation Heatmap (full)")
    plt.show()

if __name__ == "__main__":
    filepath = "../steam_data/games.csv"
    games = load_and_clean_games(filepath)
    cleaned_games = games[[
    "Average playtime forever", "Estimated owners",
    "Peak CCU", "Price", "Recommendations", "Required age",
    "Positive", "Negative"
    ]]
    print_correlation_matrix(cleaned_games)
```

lm_and_hypotheses.py

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import zscore
import statsmodels.formula.api as smf
from statsmodels.stats.stattools import durbin_watson
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm

from setup import load_and_clean_games, clean_column_names

# set working directory
os.chdir(os.path.dirname(os.path.abspath(__file__)))

# create linear model with given dataset, Y, quantitative variables X and qualitative variables
def create_lm(dataset, Y, X, categories):
    if len(categories) == 0:
    formula = f"{Y} ~ {' + '.join(X)}"
    else:
    formula = f"{Y} ~ {' + '.join(X + categories)}"

    model = smf.ols(formula=formula, data=dataset).fit()
    return model

# check essential hypotheses for given linear model and data
def check_lm_hypotheses(model, data):
```

```
print("Vérification des hypothèses pour le modèle :")
print(model.model.formula)
print("\n")

# 1. residuals vs fitted values for linearity check
fitted_vals = model.fittedvalues
residuals = model.resid
plt.figure(figsize=(6, 4))
sns.residplot(x=fitted_vals, y=residuals, lowess=True, line_kws={'color': 'red'})
plt.xlabel("Valeurs ajustées")
plt.ylabel("Résidus")
plt.title("1. Résidus vs valeurs ajustées")
plt.axhline(0, color='gray', linestyle='--')
plt.show()

# 2. residuals normality
sm.qqplot(residuals, line='45', fit=True)
plt.title("2. residuals QQ-plot")
plt.show()

# 3. residuals independance
standardized_residuals = residuals / np.std(residuals)
sqrt_std_resid = np.sqrt(np.abs(standardized_residuals))
plt.figure(figsize=(6, 4))
sns.scatterplot(x=fitted_vals, y=sqrt_std_resid, alpha=0.4)
sns.regplot(x=fitted_vals, y=sqrt_std_resid, scatter=False, lowess=True,
line_kws={'color': 'red'})
plt.title("3. Écarts à l'effet de levier")
plt.xlabel("Valeurs ajustées")
plt.ylabel("√|Résidus standardisés|")
plt.show()
```

```
# 4. homoscedasticity
abs_resid = np.abs(residuals)
plt.figure(figsize=(6, 4))
sns.scatterplot(x=fitted_vals, y=abs_resid, alpha=0.4)
sns.regplot(x=fitted_vals, y=abs_resid, scatter=False, lowess=True, line_kws={'color':
'red'})
plt.title("4. Hétéroscédasticité : résidus absolus vs ajustés")
plt.xlabel("Valeurs ajustées")
plt.ylabel("Résidus absolus")
plt.show()

# 5. autocorrelation with durbin-watson and acf
dw_stat = durbin_watson(residuals)
print(f"\n5. Durbin-Watson : {dw_stat:.3f} (attendu ≈ 2)\n")

# ACF
sm.graphics.tsa.plot_acf(residuals, lags=40)
plt.title("5. ACF des résidus")
plt.show()

# 6. histogram and boxplot of residuals
plt.figure(figsize=(4, 4))
sns.boxplot(y=residuals)
plt.title("Boxplot des résidus")
plt.show()

plt.figure(figsize=(6, 4))
sns.histplot(residuals[residuals < np.quantile(residuals, 1)], bins=50, kde=True,
color="blue")
plt.title("Histogramme des résidus")
plt.xlabel("Résidus")
```

```
plt.show()

plt.figure(figsize=(6, 4))
sns.histplot(residuals[residuals < np.quantile(residuals, 0.99)], bins=50, kde=True, color="blue")
plt.title("Histogramme des résidus (sans top 1%)")
plt.xlabel("Résidus")
plt.show()

# 7. multicolinearity with VIF
print("7. VIF (Variance Inflation Factor) :")
X = model.model.exog
vif_data = pd.DataFrame()
vif_data["Variable"] = model.model.exog_names
vif_data["VIF"] = [variance_inflation_factor(X, i) for i in range(X.shape[1])]
print(vif_data)
print("\nVariables avec VIF > 5 :")
print(vif_data[vif_data["VIF"] > 5]["Variable"].tolist())

# 8. Cook's distance
cooks_d = model.get_influence().cooks_distance[0]
seuil = 4 / len(data)
plt.figure(figsize=(8, 4))
plt.stem(np.arange(len(cooks_d)), cooks_d, markerfmt=",")
plt.axhline(y=seuil, color="red", linestyle="--", linewidth=2)
plt.title("8. Distance de Cook avec seuil 4/n")
plt.xlabel("Index de l'observation")
plt.ylabel("Distance de Cook")
plt.legend([f"Seuil = 4/n ≈ {seuil:.5f}"], loc="upper right")
plt.tight_layout()
plt.show()
```

```
# apply transformation according to the specified method
def apply_transformations(data, variables, method="log"):
    data = data.copy()
    for var in variables:
    if method == "log":
    data[var] = np.log10(data[var] + 1)
    elif method == "sqrt":
    data[var] = np.sqrt(data[var])
    elif method == "standardize":
    data[var] = (data[var] - data[var].mean()) / data[var].std()
    elif method == "normalize":
    data[var] = (data[var] - data[var].min()) / (data[var].max() - data[var].min())
    return data

# detects high influence point according to Cook's distance
def detect_cook(model, threshold=None):
    influence = model.get_influence()
    cooks_d = influence.cooks_distance[0]
    if threshold is None:
    threshold = 4 / model.nobs
    return np.where(cooks_d > threshold)[0]

# detects large residuals
def detect_large_residuals(model, threshold=3):
    influence = model.get_influence()
    student_resid = influence.resid_studentized_external
    return np.where(np.abs(student_resid) > threshold)[0]

# detects outliers according to zscore=3
def detect_outliers_data(dataset, threshold=3):
```

```
    numeric_data = dataset.select_dtypes(include=[np.number])
    z_scores = zscore(numeric_data, nan_policy='omit')
    outlier_rows = np.where(np.abs(z_scores) > threshold)
    return np.unique(outlier_rows[0])

# removes large residuals, outliers and high influence point according to Cook's distance
def clean_model(model, dataset):
    idx_cook = detect_cook(model)
    idx_resid = detect_large_residuals(model)
    idx_outliers = detect_outliers_data(dataset)
    idx_to_remove = np.unique(np.concatenate([idx_cook, idx_resid, idx_outliers]))
    cleaned_data = dataset.drop(index=idx_to_remove)
    return {
    'data': cleaned_data,
    'removed': idx_to_remove
    }

if __name__ == "__main__":
    filepath = "../steam_data/games.csv"
    games = load_and_clean_games(filepath)
    gamesc = games[[
    "Average playtime forever", "Estimated owners",
    "Peak CCU", "Price", "Recommendations", "Required age",
    "Positive", "Negative"
    ]]
    gamesc = clean_column_names(gamesc)
    Y = "average_playtime_forever"
    X = ["peak_ccu", "positive", "negative", "recommendations", "price", "required_age"]
    categories = ["estimated_owners"]

    model = create_lm(gamesc, Y, X, categories)
```

```
print(model.summary())
check_lm_hypotheses(model, gamesc)

variables_to_transform = ["peak_ccu", "positive", "negative", "recommendations", "price"]
gamesc_log = apply_transformations(gamesc, variables_to_transform, "log")
model_log = create_lm(gamesc_log, Y, X, categories)
print(model_log.summary())
check_lm_hypotheses(model_log, gamesc_log)

cleaned_result = clean_model(model, gamesc_log)
gamesc_log_clean = cleaned_result['data']
model_log_clean = create_lm(gamesc_log_clean, Y, X, categories)
print(model_log_clean.summary())
check_lm_hypotheses(model_log_clean, gamesc_log_clean)
```

lm_selection.py

```
import os
import pandas as pd
import numpy as np
import statsmodels.api as sm
from itertools import combinations
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

from setup import load_and_clean_games, clean_column_names

# set working directory to script location
os.chdir(os.path.dirname(os.path.abspath(__file__)))

# performs exhaustive model selection based on aic or bic, testing all combinations up to a given level
def select_model_glmulti(data, target_col='average_playtime_forever', criterion='aic', level=1):
    X = data.drop(columns=[target_col])
    y = data[target_col]

    best_score = np.inf
    best_model = None
    best_formula = None

    for i in range(1, level + 1):
    for combo in combinations(X.columns, i):
    X_combo = sm.add_constant(X[list(combo)])
    model = sm.OLS(y, X_combo).fit()
    score = model.aic if criterion == 'aic' else model.bic
```

```
        if score < best_score:
        best_score = score
        best_model = model
        best_formula = combo

        print(f"best formula: {best_formula}, score ({criterion}): {best_score:.2f}")
        return best_model

# performs stepwise selection (forward, backward, or both) based on aic or bic
def run_stepwise(data, target_col='average_playtime_forever', direction='forward',
criterion='aic'):
        def compute_score(model):
        return model.aic if criterion == 'aic' else model.bic

        X = data.drop(columns=[target_col])
        y = data[target_col]
        included = []
        changed = True
        best_score = None

        while changed:
        changed = False
        excluded = list(set(X.columns) - set(included))
        candidates = []

        if direction in ['forward', 'both']:
        for new_col in excluded:
        model = sm.OLS(y, sm.add_constant(data[included + [new_col]])).fit()
        score = compute_score(model)
        candidates.append((score, new_col))
```

```
	if direction in ['backward', 'both'] and included:
	for col in included:
	temp_included = included.copy()
	temp_included.remove(col)
	if temp_included:
		model = sm.OLS(y, sm.add_constant(data[temp_included])).fit()
		score = compute_score(model)
		candidates.append((score, f"-{col}"))

	if not candidates:
	break

	candidates.sort()
	best_candidate = candidates[0]
	if best_score is None or best_candidate[0] < best_score:
	best_score = best_candidate[0]
	col = best_candidate[1]
	if col.startswith('-'):
	included.remove(col[1:])
	else:
	included.append(col)
	changed = True

	final_model = sm.OLS(y, sm.add_constant(data[included])).fit()
	print(f"final variables: {included}")
	return final_model

# transforms a dataset with categorical variables using one-hot encoding
def turn_data_to_num(data, target_col='average_playtime_forever'):
	# separate numerical and categorical variables
	X = data.drop(columns=[target_col])
```

```
	y = data[target_col]

	categorical_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()

	# pipeline for one-hot encoding
	transformer = ColumnTransformer(transformers=[
	('cat', OneHotEncoder(drop='first', sparse_output=False), categorical_cols)
	], remainder='passthrough')

	X_transformed = transformer.fit_transform(X)
	feature_names = transformer.get_feature_names_out()

	df_transformed = pd.DataFrame(X_transformed, columns=feature_names)
	df_transformed[target_col] = y.reset_index(drop=True)
	return df_transformed

if __name__ == "__main__":
	filepath = "../steam_data/games.csv"
	games = load_and_clean_games(filepath)
	gamesc = games[[
	"Average playtime forever", "Estimated owners",
	"Peak CCU", "Price", "Recommendations", "Required age",
	"Positive", "Negative"
	]]
	gamesc = clean_column_names(gamesc)

	# turn categorical variables to numerical variables with indicator function
	gamesc_num = turn_data_to_num(gamesc, target_col='average_playtime_forever')

	print("\n--- selection via glmulti (level 1) ---")
	glmulti_model = select_model_glmulti(gamesc_num, criterion='aic', level=1)
```

```
print(glmulti_model.summary())

print("\n--- stepwise selection (forward + aic) ---")
stepwise_model = run_stepwise(gamesc_num, direction='forward', criterion='aic')
print(stepwise_model.summary())
```

setup.py

```
import pandas as pd
import os
import re

os.chdir(os.path.dirname(os.path.abspath(__file__)))

def create_total_reviews(positive, negative):
    return positive + negative

def create_positive_ratio(positive, negative):
    total = positive + negative
    if total == 0:
    return 0
    return (positive / total) * 100

def create_rating(positive, negative):
    total = positive + negative
    if total == 0:
    return "Not enough reviews"

    ratio = (positive / total) * 100

    if total >= 500:
    if 95 <= ratio <= 100:
    return "Overwhelmingly Positive"
    elif 80 <= ratio < 95:
    return "Very Positive"
    elif 70 <= ratio < 80:
    return "Mostly Positive"
```

```
elif 40 <= ratio < 70:
return "Mixed"
elif 20 <= ratio < 40:
return "Mostly Negative"
else:
return "Overwhelmingly Negative"
elif 50 <= total <= 499:
if 80 <= ratio <= 100:
return "Very Positive"
elif 70 <= ratio < 80:
return "Mostly Positive"
elif 40 <= ratio < 70:
return "Mixed"
elif 20 <= ratio < 40:
return "Mostly Negative"
else:
return "Very Negative"
elif 10 <= total <= 49:
if 80 <= ratio <= 100:
return "Positive"
elif 70 <= ratio < 80:
return "Mostly Positive"
elif 40 <= ratio < 70:
return "Mixed"
elif 20 <= ratio < 40:
return "Mostly Negative"
else:
return "Negative"
else:
return "Not enough reviews"
```

```
def load_and_clean_games(filepath):
	df = pd.read_csv(filepath)

	# flter games with playtime > 0 and Peak.CCU > 0
	df = df[(df["Average playtime forever"] > 0) & (df["Peak CCU"] > 0)]

	# fill NA with 0
	df.fillna(0, inplace=True)

	# create new variables
	df["total_reviews"] = df.apply(lambda row: create_total_reviews(row["Positive"],
row["Negative"]), axis=1)
	df["positive_ratio"] = df.apply(lambda row: create_positive_ratio(row["Positive"],
row["Negative"]), axis=1)
	df["rating"] = df.apply(lambda row: create_rating(row["Positive"], row["Negative"]),
axis=1)

	# set order for rating
	rating_levels = [
	"Overwhelmingly Positive", "Very Positive", "Positive",
	"Mostly Positive", "Mixed", "Mostly Negative", "Negative",
	"Very Negative", "Overwhelmingly Negative", "Not enough reviews"
	]
	df["rating"] = pd.Categorical(df["rating"], categories=rating_levels, ordered=True)

	# select variables
	df = df[[
	"Name", "Publishers", "Average playtime forever", "Estimated owners",
	"Peak CCU", "rating", "Price", "Recommendations", "Required age",
	"Positive", "Negative", "total_reviews", "positive_ratio"
	]]
	df = df.reset_index(drop=True)
```

```
    return df

def clean_column_names(df):
    """ cleans column names
    """
    new_columns = {
    col: re.sub(r'\W+', '_', col.strip().lower())
    for col in df.columns
    }
    df = df.rename(columns=new_columns)
    return df

if __name__ == "__main__":
    filepath = "../steam_data/games.csv"
    gamesc = load_and_clean_games(filepath)
    print(gamesc.head())
```

univ_rating.py

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from setup import load_and_clean_games

# set working directory
os.chdir(os.path.dirname(os.path.abspath(__file__)))

def plot_rating_distribution(gamesc):
    # frequency
    rating_counts = gamesc["rating"].value_counts()
    print("Fréquences absolues :")
    print(rating_counts)

    # precentages
    rating_percent = rating_counts / rating_counts.sum() * 100
    print("\nPourcentages (%):")
    print(rating_percent.round(2))

    # df for plot
    rating_df = pd.DataFrame({
    "rating": rating_counts.index,
    "percentage": rating_percent.values
    })

    # horizontal graph
    plt.figure(figsize=(8, 5))
```

```
    sns.barplot(data=rating_df, y="rating", x="percentage", color="steelblue",
edgecolor="black")
    plt.title("Proportion of Game Ratings", fontsize=14, weight='bold')
    plt.xlabel("Proportion (%)")
    plt.ylabel("Rating")
    plt.xlim(0, rating_df["percentage"].max() * 1.1)
    plt.grid(axis="x", linestyle="--", alpha=0.5)
    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    filepath = "../steam_data/games.csv"
    games = load_and_clean_games(filepath)
    cleaned_games = games[[
    "Average playtime forever", "Estimated owners",
    "Peak CCU", "Price", "Recommendations", "Required age",
    "Positive", "Negative", "rating"
    ]]
    plot_rating_distribution(cleaned_games)
```

univariate_tests.py

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from setup import load_and_clean_games

# set working directory
os.chdir(os.path.dirname(os.path.abspath(__file__)))

def print_boxplots(cleaned_games: pd.DataFrame) -> None:

    plt.boxplot(cleaned_games['Average playtime forever'].dropna())
    plt.title("Distribution du temps de jeu moyen")
    plt.ylabel("Temps de jeu moyen (minutes)")
    plt.show()

    plt.boxplot(cleaned_games['Recommendations'].dropna())
    plt.title("Distribution du nombre de recommandations")
    plt.ylabel("Nombre de recommandations")
    plt.show()

    plt.boxplot(cleaned_games['Negative'].dropna())
    plt.title("Distribution du nombre d'avis négatifs")
    plt.ylabel("Nombre d'avis négatifs")
    plt.show()

    plt.boxplot(cleaned_games['Positive'].dropna())
    plt.title("Distribution du nombre d'avis positifs")
```

```
    plt.ylabel("Nombre d'avis positifs")
    plt.show()

    plt.boxplot(cleaned_games['Price'].dropna())
    plt.title("Distribution du prix des jeux")
    plt.ylabel("Prix en dollars")
    plt.show()

    plt.boxplot(cleaned_games['Peak CCU'].dropna())
    plt.title("Distribution des maximums de joueurs simultanés atteints")
    plt.ylabel("Nombre de joueurs simultanés")
    plt.show()

def categorize_age(age: int) -> str:
    if pd.isna(age) or age == "" or age < 12:
    return "Tout public"
    elif 12 <= age < 16:
    return "+12"
    elif 16 <= age < 18:
    return "+16"
    else:
    return "+18"

def print_required_age(cleaned_games: pd.DataFrame) -> None:
    cleaned_games['age_Category'] = cleaned_games['Required
age'].apply(categorize_age)
    sns.countplot(data=cleaned_games, x='age_Category', order=["Tout public", "+12",
"+16", "+18"], color="steelblue")
    plt.title("Fréquence des catégories d'âge")
    plt.xlabel("Âge requis")
    plt.ylabel("Nombre de jeux")
    plt.tight_layout()
```

```
    plt.show()

def print_owners(cleaned_games):
    # rename categories to more readable labels
    new_labels = {
    "0 - 20000": "0-20k",
    "20000 - 50000": "20k-50k",
    "50000 - 100000": "50k-100k",
    "100000 - 200000": "100k-200k",
    "200000 - 500000": "200k-500k",
    "500000 - 1000000": "500k-1M",
    "1000000 - 2000000": "1M-2M",
    "2000000 - 5000000": "2M-5M",
    "5000000 - 10000000": "5M-10M",
    "10000000 - 20000000": "10M-20M",
    "20000000 - 50000000": "20M-50M",
    "50000000 - 100000000": "50M-100M",
    "100000000 - 200000000": "100M-200M"
    }

    cleaned_games["Estimated owners2"] = cleaned_games["Estimated
owners"].map(new_labels)

    # set categories order
    ordered_categories = list(new_labels.values())
    cleaned_games["Estimated owners2"] = pd.Categorical(
    cleaned_games["Estimated owners2"],
    categories=ordered_categories,
    ordered=True
    )
```

```
        print("Catégories ordonnées :", cleaned_games["Estimated owners2"].cat.categories)

        plt.figure(figsize=(10, 6))
        sns.countplot(data=cleaned_games, x="Estimated owners2", color="steelblue")
        plt.title("Distribution of Estimated Owners")
        plt.xlabel("Estimated Owners")
        plt.ylabel("Number of Games")
        plt.xticks(rotation=45, ha="right")
        plt.tight_layout()
        plt.show()

if __name__ == "__main__":
        filepath = "../steam_data/games.csv"
        games = load_and_clean_games(filepath)
        cleaned_games = games[[
        "Average playtime forever", "Estimated owners",
        "Peak CCU", "Price", "Recommendations", "Required age",
        "Positive", "Negative"
        ]]
        print_boxplots(cleaned_games)
        print_required_age(cleaned_games)
        print_owners(cleaned_games)
```

univariate_tests_log.py

```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from setup import load_and_clean_games

os.chdir(os.path.dirname(os.path.abspath(__file__)))

def plot_density(data, column, title, log=False):
    plt.figure(figsize=(8, 5))
    if log:
    # avoid log(0)
    data = data[data[column] > 0]
    x = np.log10(data[column])
    xlabel = f"log10({column})"
    else:
    x = data[column]
    xlabel = column
    sns.kdeplot(x, fill=True, color="blue", alpha=0.3)
    plt.title(title, fontsize=14, fontweight='bold')
    plt.xlabel(xlabel)
    plt.ylabel("Density")
    plt.grid(True, linestyle="--", alpha=0.5)
    plt.tight_layout()
    plt.show()

def print_log_tests(cleaned_games):
    plot_density(cleaned_games, "Peak CCU", "Density Plot of Peak CCU (log10 scale)",
log=True)
```

```
    plot_density(cleaned_games, "Price", "Density Plot of Price", log=False)
    plot_density(cleaned_games, "Positive", "Density Plot of number of positive reviews (log10 scale)", log=True)
    plot_density(cleaned_games, "Negative", "Density Plot of number of negative reviews (log10 scale)", log=True)
    plot_density(cleaned_games, "Recommendations", "Density Plot of number of recommendations (log10 scale)", log=True)
    plot_density(cleaned_games, "Average playtime forever", "Density Plot of Average playtime forever (log10 scale)", log=True)


if __name__ == "__main__":
    filepath = "../steam_data/games.csv"
    games = load_and_clean_games(filepath)
    cleaned_games = games[[
    "Average playtime forever", "Estimated owners",
    "Peak CCU", "Price", "Recommendations", "Required age",
    "Positive", "Negative"
    ]]
    print_log_tests(cleaned_games)
```