1. Probability a. Calculating the simple probabilities b. Applications of Probability distributions to real life problems python program

```
import math
def calculate_probability(favorable_outcomes, total_outcomes):
    probability = favorable_outcomes / total_outcomes
    return probability
def calculate_binomial_probability(n, p, k):
    q = 1 - p  # probability of failure
    binomial_coefficient = math.comb(n, k)  # binomial coefficient: n choose k
    probability = binomial_coefficient * (p ** k) * (q ** (n - k))
    return probability
favorable_outcomes_dice = 1
total_outcomes_dice = 6
num_trials_coin = 5
probability_of_heads_coin = 0.5  # Fair coin, so the probability of heads is 0.5
probability_dice = calculate_probability(favorable_outcomes_dice, total_outcomes_dice)
result_probability_coin = calculate_binomial_probability(num_trials_coin, probability_of_heads_coin, 3)
print(f"The probability of rolling a 4 on a fair six-sided die is: {probability_dice:.2f}")
print(f"The probability of getting exactly 3 heads in 5 coin flips is: {result_probability_coin:.4f}")
```

2. Test of significance a. T-Test: one sample, two independent samples and paired

b. ANOVA & Chi-Square Test

```
import numpy as np
def calculate_t_statistic(sample, population_mean):
    sample_mean = np.mean(sample)
    sample_std = np.std(sample, ddof=1)
    n = len(sample)
    t_stat = (sample_mean - population_mean) / (sample_std / np.sqrt(n))
    return t_stat
```

```
def calculate_p_value(t_stat, degrees_of_freedom):
    p_value = 2 * (1 - abs(t_stat))
    return p_value
def one_sample_t_test(sample, population_mean):
    t_stat = calculate_t_statistic(sample, population_mean)
    df = len(sample) - 1
    p_value = calculate_p_value(t_stat, df)
    return t_stat, p_value

def two_independent_samples_t_test(sample1, sample2):
    mean_diff = np.mean(sample1) - np.mean(sample2)
    n1, n2 = len(sample1), len(sample2)
    s1, s2 = np.var(sample1, ddof=1), np.var(sample2, ddof=1)
    pooled_var = ((n1 - 1) * s1 + (n2 - 1) * s2) / (n1 + n2 - 2)
    standard_error = np.sqrt(pooled_var * (1/n1 + 1/n2))
    t_stat = mean_diff / standard_error
    df = n1 + n2 - 2
    p_value = calculate_p_value(t_stat, df)
    return t_stat, p_value
def paired_t_test(before, after):
    differences = after - before
    mean_diff = np.mean(differences)
    std_diff = np.std(differences, ddof=1)
    n = len(differences)
    t_stat = mean_diff / (std_diff / np.sqrt(n))
    df = n - 1
    p_value = calculate_p_value(t_stat, df)
    return t_stat, p_value
def anova_test(*groups):
    all_data = np.concatenate(groups)
    overall_mean = np.mean(all_data)
```

```
    ss_total = np.sum((all_data - overall_mean) ** 2)
    ss_between = 0
    for group in groups:
        group_mean = np.mean(group)
        ss_between += len(group) * (group_mean - overall_mean) ** 2
    ss_within = ss_total - ss_between
    df_between = len(groups) - 1
    df_within = len(all_data) - len(groups)
    ms_between = ss_between / df_between
    ms_within = ss_within / df_within
    f_stat = ms_between / ms_within
    p_value = calculate_p_value(f_stat, df_between)
    return f_stat, p_value
def chi_square_test(observed):
    row_totals = np.sum(observed, axis=1)
    col_totals = np.sum(observed, axis=0)
    total = np.sum(observed)
    expected = np.outer(row_totals, col_totals) / total
    chi2_stat = np.sum((observed - expected) ** 2 / expected)
    df = (len(row_totals) - 1) * (len(col_totals) - 1)
    p_value = calculate_p_value(chi2_stat, df)
    return chi2_stat, p_value, df
data_one_sample = np.array
```

3.Correlation and Regression analysis a. Scattered diagram, calculating of correlation coefficient b. Linear regression: fitting, testing model adequacy and prediction (simple and multiple) c. Fitting of logistic regression

```
import numpy as np
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
import matplotlib.pyplot as plt
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array([2, 4, 5, 4, 5, 7, 8, 9, 10, 12])
plt.scatter(x, y)
plt.title("Scatter Diagram")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
correlation_coefficient = np.corrcoef(x, y)[0, 1]
print(f"Correlation Coefficient: {correlation_coefficient:.4f}")
x_simple = x.reshape(-1, 1)
model_simple = LinearRegression().fit(x_simple, y)
y_pred_simple = model_simple.predict(x_simple)
plt.scatter(x, y)
plt.plot(x, y_pred_simple, color='red')
plt.title("Simple Linear Regression")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
mse_simple = mean_squared_error(y, y_pred_simple)
r2_simple = r2_score(y, y_pred_simple)
print(f"Mean Squared Error (MSE) for Simple Linear Regression: {mse_simple:.4f}")
print(f"R-squared (R2) for Simple Linear Regression: {r2_simple:.4f}")
x_multi = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [8, 9], [9, 10], [10, 11]])
model_multi = LinearRegression().fit(x_multi, y)
y_pred_multi = model_multi.predict(x_multi)
mse_multi = mean_squared_error(y, y_pred_multi)
r2_multi = r2_score(y, y_pred_multi)
print(f"Mean Squared Error (MSE) for Multiple Linear Regression: {mse_multi:.4f}")
print(f"R-squared (R2) for Multiple Linear Regression: {r2_multi:.4f}")
```

```
x_logistic = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]])
y_logistic = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
x_train, x_test, y_train, y_test = train_test_split(x_logistic, y_logistic, test_size=0.2, random_state=42)
model_logistic = LogisticRegression().fit(x_train, y_train)
y_pred_logistic = model_logistic.predict(x_test)
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
print(f"Accuracy for Logistic Regression: {accuracy_logistic:.4f}")
```