Task 31: Percentage of Items in a Normal Distribution

Given:

- Mean (μ\muμ) = 60
- Standard deviation (σ\sigmaσ) = 10

We will use the standard normal distribution (Z-score) to find the percentages.

(i) Between 60 and 72:

Z=X−μσZ = \frac{X - \mu}{\sigma}Z=σX−μ Z60=60−6010=0Z_{60} = \frac{60 - 60}{10} = 0Z60=1060−60=0 Z72=72−6010=1.2Z_{72} = \frac{72 - 60}{10} = 1.2Z72=1072−60=1.2

Using Z-tables or a standard normal distribution calculator: P(0<Z<1.2)≈0.3849P(0 < Z < 1.2) \approx 0.3849P(0<Z<1.2)≈0.3849 So, approximately 38.49% of items are between 60 and 72.

(ii) Between 50 and 60:

Z50=50−6010=−1Z_{50} = \frac{50 - 60}{10} = -1Z50=1050−60=−1 Z60=0Z_{60} = 0Z60=0

Using Z-tables: P(−1<Z<0)≈0.3413P(-1 < Z < 0) \approx 0.3413P(−1<Z<0)≈0.3413 So, approximately 34.13% of items are between 50 and 60.

(iii) Beyond 72:

P(Z>1.2)=1−P(Z<1.2)P(Z > 1.2) = 1 - P(Z < 1.2)P(Z>1.2)=1−P(Z<1.2) P(Z<1.2)≈0.8849P(Z < 1.2) \approx 0.8849P(Z<1.2)≈0.8849 P(Z>1.2)≈1−0.8849=0.1151P(Z > 1.2) \approx 1 - 0.8849 = 0.1151P(Z>1.2)≈1−0.8849=0.1151 So, approximately 11.51% of items are beyond 72.

(iv) Between 70 and 80:

Z70=70−6010=1Z_{70} = \frac{70 - 60}{10} = 1Z70=1070−60=1 Z80=80−6010=2Z_{80} = \frac{80 - 60}{10} = 2Z80=1080−60=2

Using Z-tables: P(1<Z<2)≈0.1359P(1 < Z < 2) \approx 0.1359P(1<Z<2)≈0.1359 So, approximately 13.59% of items are between 70 and 80.

Task 32: Proportion of Students Scoring More than a Certain Mark

Given:

- Mean (μ\muμ) = 49
- Standard deviation (σ\sigmaσ) = 6
- Total students = 15000

(a) More than 55 marks:

Z55=55−496=1Z_{55} = \frac{55 - 49}{6} = 1Z55=655−49=1

Using Z-tables: P(Z>1)=1−P(Z<1)≈1−0.8413=0.1587P(Z > 1) = 1 - P(Z < 1) \approx 1 - 0.8413 = 0.1587P(Z>1)=1−P(Z<1)≈1−0.8413=0.1587

Proportion of students = 0.1587×15000≈23800.1587 \times 15000 \approx 23800.1587×15000≈2380

(b) More than 70 marks:

Z70=70−496≈3.5Z_{70} = \frac{70 - 49}{6} \approx 3.5Z70=670−49≈3.5

Using Z-tables: P(Z>3.5)≈1−0.99995=0.00005P(Z > 3.5) \approx 1 - 0.99995 = 0.00005P(Z>3.5)≈1−0.99995=0.00005

Proportion of students = 0.00005×15000≈0.750.00005 \times 15000 \approx 0.750.00005×15000≈0.75

So, very few students (less than 1) are expected to score more than 70 marks.

Task 33: Number of Students with Heights

Given:

- Mean (μ\muμ) = 65 inches
- Standard deviation (σ\sigmaσ) = 5 inches
- Total students = 500

(a) Greater than 70 inches:

Z70=70−655=1Z_{70} = \frac{70 - 65}{5} = 1Z70=570−65=1

Using Z-tables: P(Z>1)=1−P(Z<1)≈1−0.8413=0.1587P(Z > 1) = 1 - P(Z < 1) \approx 1 - 0.8413 = 0.1587P(Z>1)=1−P(Z<1)≈1−0.8413=0.1587

Number of students = 0.1587×500≈79.350.1587 \times 500 \approx 79.350.1587×500≈79.35
Approximately 79 students have heights greater than 70 inches.

(b) Between 60 and 70 inches:

Z60=60−655=−1Z_{60} = \frac{60 - 65}{5} = -1Z60=560−65=−1 Z70=1Z_{70} = 1Z70=1

Using Z-tables: P(−1<Z<1)≈0.6826P(-1 < Z < 1) \approx 0.6826P(−1<Z<1)≈0.6826

Number of students = 0.6826×500≈341.30.6826 \times 500 \approx 341.30.6826×500≈341.3
Approximately 341 students have heights between 60 and 70 inches.

Task 34: Statistical Hypothesis and Errors in Hypothesis Testing

Statistical Hypothesis: A statistical hypothesis is an assumption or claim about the population parameter. It can be tested using sample data. Hypothesis testing is a method of making decisions or inferences about population parameters based on sample statistics.

Errors in Hypothesis Testing:

1. **Type I Error (α)**:
   - Occurs when the null hypothesis is true, but we reject it.
   - The probability of committing a Type I error is denoted by α (significance level).
2. **Type II Error (β)**:
   - Occurs when the null hypothesis is false, but we fail to reject it.
   - The probability of committing a Type II error is denoted by β.

Sample: A sample is a subset of a population used to make inferences about the population.

- **Large Samples**:
  - Typically, a sample size greater than 30 is considered large.
  - The Central Limit Theorem applies, making the sample mean approximately normally distributed.
- **Small Samples**:
  - Sample size less than 30.
  - The t-distribution is often used for hypothesis testing.

Task 35: Hypothesis Test for Population Standard Deviation

Given:

- Sample size (n) = 25
- Sample standard deviation (s) = 9.0
- Population standard deviation (σ\sigmaσ) = 10.5

Hypothesis:

- Null hypothesis (H0H_0H0): σ=10.5\sigma = 10.5σ=10.5
- Alternative hypothesis (H1H_1H1): σ≠10.5\sigma \neq 10.5σ=10.5

Test Statistic: Using the chi-square distribution: χ2=(n−1)s2σ2\chi^2 = \frac{(n-1)s^2}{\sigma^2}χ2=σ2(n−1)s2

χ2=(25−1)×9210.52\chi^2 = \frac{(25-1) \times 9^2}{10.5^2}χ2=10.52(25−1)×92 χ2=24×81110.25\chi^2 = \frac{24 \times 81}{110.25}χ2=110.2524×81 χ2=1944110.25\chi^2 = \frac{1944}{110.25}χ2=110.251944 χ2≈17.64\chi^2 \approx 17.64χ2≈17.64

Degrees of freedom (df) = n−1=24n-1 = 24n−1=24

Using the chi-square table or a calculator, we can determine the critical values for a given significance level (typically 0.05). If the calculated chi-square value falls outside the critical region, we reject the null hypothesis.

Let's determine this with Python:

```
from scipy.stats import chi2

# Given values
n = 25
s = 9.0
sigma = 10.5

# Calculate the chi-square value
chi_square = (n - 1) * (s**2) / (sigma**2)
chi_square

# Degrees of freedom
df = n - 1

# Critical value at 0.05 significance level (two-tailed test)
alpha = 0.05
critical_value_low = chi2.ppf(alpha / 2, df)
critical_value_high = chi2.ppf(1 - alpha / 2, df)

chi_square, critical_value_low, critical_value_high
```

The calculated chi-square value is approximately 17.63. The critical values for a 0.05 significance level (two-tailed test) are:

- Lower critical value: 12.40
- Upper critical value: 39.36

Task 37: Chi-Square Test for Uniform Distribution of Grades

Given:

- Grades: [A, B, C, D, E]
- Frequencies: [15, 17, 30, 22, 16]

Total students = 100

Hypothesis:

- Null hypothesis (H0H_0H0): The grades are uniformly distributed.
- Alternative hypothesis (H1H_1H1): The grades are not uniformly distributed.

If the distribution is uniform, each grade would be expected to occur equally often.

Expected frequency for each grade: Expected frequency=Total frequencyNumber of categories=1005=20\text{Expected frequency} = \frac{\text{Total frequency}}{\text{Number of categories}} = \frac{100}{5} = 20Expected frequency=Number of categoriesTotal frequency=5100=20

Chi-Square Test Statistic: χ2=∑(Oi−Ei)2Ei\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}χ2=∑Ei(Oi−Ei)2 Where OiO_iOi is the observed frequency and EiE_iEi is the expected frequency.

Let's calculate this:

Grade	Observed (O)	Expected (E)	(O−E)2/E(O - E)^2 / E(O−E)2/E
A	15	20	(15−20)220\frac{(15 - 20)^2}{20}20(15−20) 2
B	17	20	(17−20)220\frac{(17 - 20)^2}{20}20(17−20) 2
C	30	20	(30−20)220\frac{(30 - 20)^2}{20}20(30−20) 2
D	22	20	(22−20)220\frac{(22 - 20)^2}{20}20(22−20) 2
E	16	20	(16−20)220\frac{(16 - 20)^2}{20}20(16−20) 2

Let's calculate these values:

The calculated chi-square statistic is approximately 7.7. With 4 degrees of freedom, the p-value is approximately 0.103.

Interpretation:

- At a significance level of 0.05, the p-value (0.103) is greater than 0.05.
- Therefore, we fail to reject the null hypothesis.

Conclusion: There is not enough evidence to suggest that the distribution of grades is not uniform.

Task 39: Basic Flask Route

To create a basic Flask route that displays "Hello, World!" on the homepage:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello, World!"

if __name__ == '__main__':
    app.run(debug=True)
```

Task 40: Handling Form Submissions in Flask Using POST Requests

To set up a Flask application to handle form submissions using POST requests:

Install Flask:

```
pip install Flask
```

1.

Create a Flask app with a form:

```
from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        name = request.form['name']
        return f"Hello, {name}!"
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

2.

Create a `templates/index.html` file with a form:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Form</title>
```

```
</head>
<body>
    <form method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name">
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

3.

Task 41: Flask Route with URL Parameter

To create a Flask route that accepts a parameter in the URL and displays it on the page:

```
from flask import Flask

app = Flask(__name__)

@app.route('/hello/<name>')
def hello_name(name):
    return f"Hello, {name}!"

if __name__ == '__main__':
    app.run(debug=True)
```

Task 42: Implementing User Authentication in Flask

User authentication can be implemented using Flask-Login. Here are the steps:

Install Flask-Login:

```
pip install Flask-Login
```

1.

Set up user authentication:

```
from flask import Flask, render_template, redirect, url_for, request
from flask_login import LoginManager, UserMixin, login_user,
login_required, logout_user, current_user

app = Flask(__name__)
app.secret_key = 'supersecretkey'
login_manager = LoginManager()
```

```
login_manager.init_app(app)

class User(UserMixin):
    def __init__(self, id, username, password):
        self.id = id
        self.username = username
        self.password = password

users = [User(id=1, username='user', password='password')]

@login_manager.user_loader
def load_user(user_id):
    return next((user for user in users if user.id == int(user_id)),
None)

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = next((user for user in users if user.username ==
username and user.password == password), None)
        if user:
            login_user(user)
            return redirect(url_for('protected'))
    return render_template('login.html')

@app.route('/protected')
@login_required
def protected():
    return f"Hello, {current_user.username}!"

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)
```

2.

Create a `templates/login.html` file:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
<body>
    <form method="POST">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
        <label for="password">Password:</label>
        <input type="password" id="password" name="password">
        <input type="submit" value="Login">
    </form>
</body>
</html>
```

3.

Task 43: Connecting Flask to SQLite Using SQLAlchemy

To connect a Flask app to a SQLite database using SQLAlchemy:

Install Flask-SQLAlchemy:

```
pip install Flask-SQLAlchemy
```

1.

Set up the Flask app with SQLAlchemy:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)

    def __repr__(self):
        return f'<User {self.username}>'
```

```
@app.route('/')
def index():
    db.create_all()
    return "Database Connected!"

if __name__ == '__main__':
    app.run(debug=True)
```

2.

Task 44: Creating a RESTful API Endpoint Returning JSON Data

To create a RESTful API endpoint in Flask that returns JSON data:

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/api/data', methods=['GET'])
def get_data():
    data = {
        'name': 'John Doe',
        'age': 30,
        'location': 'New York'
    }
    return jsonify(data)

if __name__ == '__main__':
    app.run(debug=True)
```

Task 45: Using Flask-WTF to Create and Validate Forms

To use Flask-WTF for creating and validating forms:

Install Flask-WTF:

```
pip install Flask-WTF
```

1.

Create a Flask app with a form:

```
from flask import Flask, render_template, request
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitButton
from wtforms.validators import DataRequired
```

```
app = Flask(__name__)
app.secret_key = 'supersecretkey'

class MyForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired()])
    submit = SubmitButton('Submit')

@app.route('/', methods=['GET', 'POST'])
def index():
    form = MyForm()
    if form.validate_on_submit():
        name = form.name.data
        return f"Hello, {name}!"
    return render_template('index.html', form=form)

if __name__ == '__main__':
    app.run(debug=True)
```

2.

Create a `templates/index.html` file:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Form</title>
</head>
<body>
    <form method="POST">
        {{ form.hidden_tag() }}
        {{ form.name.label }} {{ form.name }}
        {{ form.submit }}
    </form>
</body>
</html>
```

3.

Task 46: Implementing File Uploads in Flask

To implement file uploads in Flask:

```
pip install Flask
```

1.

Create a Flask app with file upload functionality:

```
from flask import Flask, request, render_template

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = '/path/to/upload/folder'

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file:
            file.save(os.path.join(app.config['UPLOAD_FOLDER'],
file.filename))
            return 'File uploaded successfully!'
    return render_template('upload.html')

if __name__ == '__main__':
    app.run(debug=True)
```

2.

Create a `templates/upload.html` file:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Upload File</title>
</head>
<body>
    <form method="POST" enctype="multipart/form-data">
        <input type="file" name="file">
        <input type="submit" value="Upload">
    </form>
</body>
</html>
```

3.

Task 47: Creating a Flask Blueprint

Flask Blueprints allow you to organize your application into modules:

Create a blueprint in a separate file (e.g., `main.py`):

```
from flask import Blueprint

main = Blueprint('main', __name__)

@main.route('/')
def index():
    return "Hello from Blueprint!"
```

1.

Register the blueprint in your main app file (e.g., `app.py`):

```
from flask import Flask
```

2. `&#8203;:citation[oaicite:0]{index=0}&#8203;`