Statistical Methods in Python for Data Analysts

Comprehensive Course Textbook

Prepared for data analysts and learners seeking in-depth understanding of statistical methods using Python.

# Preface

This textbook is designed to support the learning journey of aspiring data analysts, researchers, and professionals who wish to master statistical methods using Python. Each chapter corresponds to a core topic covered in the course, blending theoretical insights with practical examples and Python implementations.

# Table of Contents

1. Module 1: Foundations of Statistical Thinking
2. Module 2: Hypothesis Testing
3. Module 3: Correlation and Simple Regression
4. Module 4: Multiple Linear Regression
5. Module 5: Advanced Regression Techniques
6. Module 6: Ensemble Learning and Decision Trees
7. Module 7: ANOVA and Beyond
8. Module 8: Resampling and Simulation
9. Module 9: Capstone Projects
10. Module 10: Final Assessment & Certification
11. Appendix: Python Setup and Resources

# Module 1: Foundations of Statistical Thinking

## 1. Introduction to Statistics

Statistics is the science of collecting, analyzing, interpreting, and presenting data. It helps analysts identify patterns, make decisions based on data, and draw conclusions from sample datasets about broader populations.

## 2. Descriptive vs Inferential Statistics

Descriptive statistics summarize and organize characteristics of a dataset (e.g., average income). Inferential statistics, on the other hand, use a random sample of data taken from a population to describe and make inferences about the population.

## 3. Types of Data and Measurement Scales

Data comes in four basic types:  
- \*\*Nominal\*\*: Categories without a natural order (e.g., gender, race).  
- \*\*Ordinal\*\*: Categories with a meaningful order (e.g., satisfaction rating).  
- \*\*Interval\*\*: Numeric values without a true zero (e.g., temperature in Celsius).  
- \*\*Ratio\*\*: Numeric values with a true zero (e.g., height, weight, income).

## 4. Measures of Central Tendency and Dispersion

Central tendency:  
- \*\*Mean\*\*: Average of data points.  
- \*\*Median\*\*: Middle value in sorted data.  
- \*\*Mode\*\*: Most frequent value.

Dispersion:  
- \*\*Range\*\*: Difference between max and min.  
- \*\*Variance\*\*: Average squared deviation from the mean.  
- \*\*Standard Deviation\*\*: Square root of variance.

## 5. Probability and Common Distributions

Probability is the likelihood of an event occurring. Distributions help us understand how data behaves:  
- \*\*Normal Distribution\*\*: Bell-shaped curve. Common in natural phenomena.  
- \*\*Binomial Distribution\*\*: Discrete outcomes (e.g., coin flips).  
- \*\*Poisson Distribution\*\*: Rare events over time (e.g., system failures).

## 6. Visualizing Data in Python

Histograms, box plots, and scatterplots are useful tools for visualizing data distributions and relationships.

## 7. Python Implementation Example

```python  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt

# Generate synthetic data  
data = np.random.normal(100, 15, 1000)  
df = pd.DataFrame(data, columns=['Scores'])

# Summary statistics  
print(df.describe())

# Visualization  
df.plot(kind='hist', bins=20, title='Score Distribution')  
plt.xlabel('Scores')  
plt.show()  
```

## 8. Exercises and Reflections

- Generate a dataset of 500 values using `np.random.normal()` with a mean of 75 and standard deviation of 10.  
- Create a histogram and compute summary statistics.  
- Reflect: What does the shape of the histogram tell you?  
- Identify if the data appears normally distributed. Why or why not?

# Module 2: Hypothesis Testing

## 1. Introduction to Hypothesis Testing

Hypothesis testing involves making an assumption (hypothesis) about a population parameter and using sample data to evaluate its validity.

- \*\*Null Hypothesis (H0)\*\*: The default assumption (e.g., no difference, no effect).  
- \*\*Alternative Hypothesis (H1)\*\*: What we aim to support (e.g., a difference exists).

## 2. Steps in Hypothesis Testing

1. State H0 and H1  
2. Choose a significance level (α, typically 0.05)  
3. Select an appropriate test  
4. Calculate test statistic and p-value  
5. Compare p-value with α to accept or reject H0

## 3. Understanding p-values and Significance

A p-value is the probability of obtaining test results at least as extreme as the observed result under H0. If p-value < α, we reject H0.

## 4. Types of Errors

- \*\*Type I Error\*\*: Incorrectly rejecting a true null hypothesis (false positive)  
- \*\*Type II Error\*\*: Failing to reject a false null hypothesis (false negative)

## 5. Confidence Intervals

A confidence interval gives a range of values within which the true population parameter likely falls. A 95% CI means we expect the true parameter to fall in the interval in 95 out of 100 repeated samples.

## 6. T-Tests (One-Sample and Two-Sample)

- \*\*One-Sample T-Test\*\*: Compares sample mean to known value  
- \*\*Two-Sample T-Test\*\*: Compares means from two independent groups

## 7. Chi-Squared Test for Independence

Used to test whether two categorical variables are independent.

Example: Is gender independent of product preference?

## 8. Python Example: Two-Sample T-Test

```python  
import numpy as np  
from scipy.stats import ttest\_ind

group1 = np.random.normal(60, 10, 50)  
group2 = np.random.normal(65, 12, 50)  
stat, p = ttest\_ind(group1, group2)  
print(f'T-statistic: {stat:.3f}, P-value: {p:.3f}')  
```

## 9. Python Example: Chi-Squared Test

```python  
import pandas as pd  
from scipy.stats import chi2\_contingency

data = pd.DataFrame({'Gender': ['M', 'F', 'M', 'F', 'M', 'F'],  
 'Preference': ['A', 'B', 'B', 'A', 'A', 'B']})  
contingency = pd.crosstab(data['Gender'], data['Preference'])  
chi2, p, dof, expected = chi2\_contingency(contingency)  
print(f'Chi-squared: {chi2:.2f}, P-value: {p:.3f}')  
```

## 10. Exercises and Reflections

- Conduct a one-sample t-test comparing a sample’s average to a known population mean.  
- Perform a chi-squared test using a 2x2 contingency table.  
- Reflect: What does rejecting the null hypothesis imply in your test?  
- Think of a real-world scenario where each test (t-test and chi-squared) could be applied.

# Module 3: Correlation and Simple Regression

## 1. Introduction to Correlation

Correlation measures the strength and direction of a linear relationship between two quantitative variables.

- \*\*Positive correlation\*\*: Both variables increase together  
- \*\*Negative correlation\*\*: One increases while the other decreases  
- \*\*No correlation\*\*: No linear relationship

## 2. Pearson vs Spearman Correlation

- \*\*Pearson Correlation\*\*: Measures linear relationships; sensitive to outliers  
- \*\*Spearman Correlation\*\*: Rank-based; captures monotonic relationships

## 3. Interpreting Correlation Coefficients

The Pearson correlation coefficient `r` ranges from -1 to 1:  
- `r = 1`: Perfect positive linear correlation  
- `r = -1`: Perfect negative linear correlation  
- `r = 0`: No linear correlation  
Values close to -1 or 1 indicate stronger linear relationships.

## 4. Introduction to Simple Linear Regression

Simple linear regression models the relationship between a dependent variable `Y` and one independent variable `X`.

Model: `Y = β₀ + β₁X + ε`  
- `β₀`: Intercept  
- `β₁`: Slope  
- `ε`: Error term

## 5. Assumptions of Linear Regression

- Linearity: Relationship between X and Y is linear  
- Independence: Observations are independent  
- Homoscedasticity: Constant variance of residuals  
- Normality: Residuals are normally distributed

## 6. Fitting and Interpreting a Linear Model in Python

```python  
import numpy as np  
import matplotlib.pyplot as plt  
from scipy.stats import linregress

# Simulate data  
x = np.arange(50)  
y = 2\*x + np.random.normal(0, 10, 50)

slope, intercept, r\_value, \_, \_ = linregress(x, y)  
print(f"Slope: {slope:.2f}, Intercept: {intercept:.2f}, R-squared: {r\_value\*\*2:.2f}")

# Plot  
plt.scatter(x, y)  
plt.plot(x, slope\*x + intercept, color='red')  
plt.title('Simple Linear Regression')  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.show()  
```

## 7. Visualizing Correlations

Heatmaps and pairplots are useful tools for exploring correlations among multiple variables.

```python  
import seaborn as sns  
import pandas as pd

# Sample dataframe  
df = pd.DataFrame({'x': x, 'y': y})  
sns.heatmap(df.corr(), annot=True)  
plt.title('Correlation Matrix')  
plt.show()  
```

## 8. Exercises and Reflections

- Generate two numerical variables with strong positive correlation and compute Pearson’s r.  
- Build a simple regression model to predict sales from advertising spend.  
- Reflect: What does the slope represent in the model?  
- Try introducing outliers. How does this affect your model and r-value?

# Module 4: Multiple Linear Regression

## 1. Introduction

Multiple Linear Regression (MLR) extends simple regression to include multiple predictors.

## 2. Model Assumptions

- Linearity, multicollinearity, homoscedasticity, and normality.

## 3. Building MLR in Python

```python  
import statsmodels.api as sm  
X = df[['x1', 'x2']]  
X = sm.add\_constant(X)  
model = sm.OLS(df['y'], X).fit()  
print(model.summary())  
```

## 4. Interpreting Coefficients

Understand partial effects of predictors on the outcome.

## 5. Multicollinearity

Detected using VIF (Variance Inflation Factor).

## 6. Residual Analysis

Plot residuals to check assumptions.

## 7. Exercises

- Build an MLR model and interpret coefficients.  
- Check VIF and residual plots.

# Module 5: Advanced Regression Techniques

## 1. Introduction

This module introduces regularization and logistic regression.

## 2. Ridge and Lasso Regression

- Used to handle overfitting and multicollinearity.

## 3. Logistic Regression

Used for binary classification tasks.

## 4. Model Evaluation

ROC curves, confusion matrix, accuracy.

## 5. Python Example

```python  
from sklearn.linear\_model import Ridge, LogisticRegression  
model = Ridge(alpha=1.0).fit(X, y)  
```

## 6. Exercises

- Apply Ridge and Logistic Regression to a dataset.

# Module 6: Ensemble Learning and Decision Trees

## 1. Introduction

Tree-based models provide intuitive classification/regression tools.

## 2. Decision Trees

Basics and overfitting issues.

## 3. Random Forests

Ensemble method that builds multiple trees.

## 4. Feature Importance

Identify top predictors.

## 5. Cross-validation

Improve reliability.

## 6. Python Example

```python  
from sklearn.ensemble import RandomForestClassifier  
model = RandomForestClassifier().fit(X, y)  
```

## 7. Exercises

- Train and evaluate a Random Forest model.

# Module 7: ANOVA and Beyond

## 1. Introduction

ANOVA tests whether group means differ.

## 2. One-Way ANOVA

Used for one factor with multiple levels.

## 3. Two-Way ANOVA

Two categorical variables.

## 4. Assumptions

Normality, equal variances.

## 5. Python Example

```python  
import statsmodels.api as sm  
from statsmodels.formula.api import ols  
model = ols('score ~ C(group)', data=df).fit()  
sm.stats.anova\_lm(model, typ=2)  
```

## 6. Exercises

- Run a one-way ANOVA on grouped data.

# Module 8: Resampling and Simulation

## 1. Introduction

Bootstrap and permutation methods help estimate uncertainty.

## 2. Bootstrap

Repeatedly sample with replacement.

## 3. Permutation Testing

Shuffle labels to test null hypotheses.

## 4. Monte Carlo Simulations

Use randomness to simulate outcomes.

## 5. Python Example

```python  
means = [np.mean(np.random.choice(data, size=len(data), replace=True)) for \_ in range(1000)]  
```

## 6. Exercises

- Use bootstrap to estimate a 95% confidence interval.

# Module 9: Capstone Projects

## 1. Project 1

Predict housing prices using MLR and regularization.

## 2. Project 2

Classify medical outcomes using logistic regression and random forests.

## 3. Project 3

Customer churn analysis using ensemble methods.

## 4. Report Template

- Describe methodology, EDA, model, evaluation, and reflection.

# Module 10: Final Assessment & Certification

## 1. Instructions

Complete a final open-ended analysis using all tools learned.

## 2. Checklist

- EDA  
- Modeling  
- Evaluation  
- Documentation

## 3. Submission

Submit a polished notebook and a short report.

## 4. Peer Review

Evaluate and learn from peers.

# Appendix: Python Setup and Resources

- Install Python from https://python.org  
- Recommended tools: JupyterLab, VS Code  
- Libraries: numpy, pandas, matplotlib, seaborn, scipy, statsmodels, scikit-learn  
- Dataset Sources: Kaggle, UCI ML Repository, open government data portals  
- GitHub repository: [To be linked with notebooks]

# Module 4: Multiple Linear Regression

## 1. Introduction

Multiple Linear Regression (MLR) is an extension of simple linear regression used to model the relationship between one continuous dependent variable and two or more independent variables. It allows analysts to control for multiple factors and uncover how each predictor contributes to the outcome.

## 2. When to Use MLR

MLR is appropriate when:  
- You want to model a continuous outcome based on multiple inputs.  
- Your predictors are either continuous or categorical (with encoding).  
- The relationship between predictors and the response is approximately linear.

## 3. The MLR Model

The general form:

`Y = β₀ + β₁X₁ + β₂X₂ + ... + βₖXₖ + ε`

- `Y`: Dependent variable  
- `X₁...Xₖ`: Independent variables  
- `β₀`: Intercept  
- `β₁...βₖ`: Regression coefficients  
- `ε`: Error term

Each coefficient represents the expected change in `Y` for a one-unit increase in the corresponding `X`, holding all other variables constant.

## 4. Assumptions of MLR

- \*\*Linearity\*\*: The relationship between X and Y is linear.  
- \*\*Independence\*\*: Observations are independent of each other.  
- \*\*Homoscedasticity\*\*: Constant variance of residuals.  
- \*\*Normality\*\*: Residuals should be approximately normally distributed.  
- \*\*No multicollinearity\*\*: Predictors should not be highly correlated.

## 5. Building an MLR Model in Python

```python  
import pandas as pd  
import statsmodels.api as sm

# Example dataset  
df = pd.DataFrame({  
 'hours\_studied': [2, 3, 5, 7, 9, 10],  
 'attendance': [80, 85, 90, 95, 98, 99],  
 'score': [50, 55, 65, 70, 78, 85]  
})

# Define predictors and response  
X = df[['hours\_studied', 'attendance']]  
X = sm.add\_constant(X) # Adds intercept  
y = df['score']

# Fit model  
model = sm.OLS(y, X).fit()  
print(model.summary())  
```

## 6. Interpreting Model Output

Focus on:  
- \*\*Coefficients\*\*: How each predictor affects the outcome  
- \*\*R-squared\*\*: Percentage of variance explained  
- \*\*p-values\*\*: Indicate statistical significance  
- \*\*Confidence intervals\*\*: Precision of estimates

## 7. Checking Multicollinearity

Multicollinearity can inflate standard errors and make coefficient estimates unreliable. Use Variance Inflation Factor (VIF):  
```python  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
vif = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]  
print(vif)  
```

## 8. Residual Analysis

Residual plots help check assumptions. Plot residuals vs fitted values:  
```python  
import matplotlib.pyplot as plt  
residuals = model.resid  
fitted = model.fittedvalues  
plt.scatter(fitted, residuals)  
plt.axhline(y=0, color='r', linestyle='--')  
plt.xlabel('Fitted Values')  
plt.ylabel('Residuals')  
plt.title('Residual Plot')  
plt.show()  
```

## 9. Exercises and Reflections

- Use a dataset to model house prices based on multiple features (e.g., square footage, number of bedrooms).  
- Check for multicollinearity using VIF.  
- Interpret the coefficients of your model.  
- Create a residual plot. What does it tell you about your model assumptions?

# Module 4: Multiple Linear Regression

## 1. Introduction

Multiple Linear Regression (MLR) is a foundational technique in predictive analytics that extends simple linear regression to account for more than one predictor variable. It allows us to evaluate the combined and individual effects of multiple independent variables on a single continuous dependent variable.

MLR is widely used in business, economics, health, social sciences, and many fields where understanding relationships between variables is essential. For example, a data analyst may want to predict a student’s exam score based on hours studied, attendance rate, and number of assignments completed.

## 2. The MLR Equation and Interpretation

The general MLR equation is:

`Y = β₀ + β₁X₁ + β₂X₂ + ... + βₖXₖ + ε`

Where:  
- `Y` is the dependent (response) variable  
- `X₁...Xₖ` are the independent (predictor) variables  
- `β₀` is the intercept  
- `β₁...βₖ` are the coefficients for each predictor  
- `ε` is the error term

\*\*Interpretation:\*\* A coefficient β represents the expected change in `Y` when the corresponding `X` increases by one unit, holding all other predictors constant.

## 3. Example Scenario: Predicting Student Scores

Let’s say we have data from six students on the number of hours they studied, their attendance percentage, and the exam score they received.

We want to predict a student's score based on study hours and attendance.

Here is how we can build this model in Python:

## 4. Fitting an MLR Model in Python

```python  
import pandas as pd  
import statsmodels.api as sm

# Simulated dataset  
df = pd.DataFrame({  
 'hours\_studied': [2, 3, 5, 7, 9, 10],  
 'attendance': [80, 85, 90, 95, 98, 99],  
 'score': [50, 55, 65, 70, 78, 85]  
})

X = df[['hours\_studied', 'attendance']]  
X = sm.add\_constant(X) # Add intercept  
y = df['score']

model = sm.OLS(y, X).fit()  
print(model.summary())  
```

## 5. Interpreting the Output

The `summary()` function gives us:  
- \*\*Coefficients\*\*: These show how much the dependent variable changes with a unit change in each predictor.  
- \*\*R-squared\*\*: The proportion of variance in `Y` explained by the predictors.  
- \*\*p-values\*\*: Tell us if a coefficient is statistically significant (typically < 0.05).

In this example, we can analyze whether 'hours\_studied' or 'attendance' has more influence on score outcomes.

## 6. Assumptions of MLR

To ensure valid results, MLR relies on the following assumptions:  
- \*\*Linearity\*\*: The relationship between each predictor and the outcome is linear.  
- \*\*Independence\*\*: Observations are independent.  
- \*\*Homoscedasticity\*\*: Constant variance of residuals across predicted values.  
- \*\*Normality of Residuals\*\*: Residuals should be approximately normally distributed.  
- \*\*No multicollinearity\*\*: Predictors should not be highly correlated with each other.

## 7. Multicollinearity Check with VIF

High multicollinearity can make it hard to interpret the model. Use Variance Inflation Factor (VIF) to detect it:  
```python  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
import numpy as np

vif\_data = pd.DataFrame()  
vif\_data['feature'] = X.columns  
vif\_data['VIF'] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]  
print(vif\_data)  
```

## 8. Visual Diagnostic: Residual Plot

Check for homoscedasticity and linearity using a residual plot:  
```python  
import matplotlib.pyplot as plt  
residuals = model.resid  
fitted = model.fittedvalues

plt.scatter(fitted, residuals)  
plt.axhline(y=0, color='red', linestyle='--')  
plt.xlabel('Fitted Values')  
plt.ylabel('Residuals')  
plt.title('Residuals vs Fitted Values')  
plt.show()  
```

## 9. Best Practices and Pitfalls

- Always check assumptions before trusting your model.  
- Beware of using too many predictors without enough data.  
- Watch out for outliers that may overly influence your coefficients.  
- Use domain knowledge to select meaningful predictors.

## 10. Exercises

- Using any dataset, build a multiple linear regression model with 2–3 predictors.  
- Check multicollinearity using VIF.  
- Plot residuals and verify assumptions.  
- Interpret your coefficients in the context of the dataset.  
- Reflect: Did adding more predictors improve your model? Why or why not?