DATASCI 347: COVID-19 Case Study

Midterm Exam Study Guide

Contents

1 Essential Python Libraries **3**
- 1.1 Required Imports . . . 3
- 1.2 Library Functions . . . 3

2 Data Preparation **3**
- 2.1 Reading and Subsetting Data . . . 3
- 2.2 Understanding Your Data . . . 4

3 Exploratory Data Analysis (EDA) **4**
- 3.1 Aggregation and Grouping . . . 4
- 3.2 Histograms . . . 4
- 3.3 Boxplots . . . 5
- 3.4 Finding Maximum/Minimum Values . . . 6

4 Linear Regression Analysis **6**
- 4.1 Simple Linear Regression . . . 6
- 4.2 Extracting Model Information . . . 7
- 4.3 Multiple Linear Regression . . . 7
- 4.4 Interpreting Coefficients . . . 8
- 4.5 Testing Categorical Variables . . . 8
- 4.6 Making Predictions . . . 8
- 4.7 Model Diagnostics . . . 9

5 LASSO Regression **10**
- 5.1 Understanding LASSO . . . 10
- 5.2 Data Preparation for LASSO . . . 10
- 5.3 Forcing Variables into LASSO . . . 11
- 5.4 Running LASSO with Cross-Validation . . . 11
- 5.5 Extracting Selected Variables . . . 12
- 5.6 Final Model After LASSO . . . 13

6 Workflow **13**
- 6.1 Complete Analysis Checklist . . . 13

7 Common Python Patterns for This Exam **15**
7.1 Pattern 1: Group Statistics and Visualization . . . 15
7.2 Pattern 2: Simple Regression and Significance Testing . . . 15
7.3 Pattern 3: Multiple Regression with Categorical Variable . . . 15
7.4 Pattern 4: Model Diagnostics . . . 16
7.5 Pattern 5: LASSO and Final Model . . . 16

8 Key Concepts to Master **17**
8.1 Statistical Concepts . . . 17
8.2 Python Skills . . . 17

9 Practice Problems **18**
9.1 Problem Set 1: Data Preparation and EDA . . . 18
9.2 Problem Set 2: Simple Regression . . . 18
9.3 Problem Set 3: Multiple Regression . . . 18
9.4 Problem Set 4: LASSO and Final Model . . . 18

10 Exam Tips **19**
10.1 Time Management . . . 19
10.2 Common Mistakes to Avoid . . . 19
10.3 What to Show . . . 19
10.4 Getting Unstuck . . . 20

11 Final Checklist Before Exam **20**

1 Essential Python Libraries

1.1 Required Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.stats.anova import anova_lm
from sklearn.linear_model import Lasso, LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
```

1.2 Library Functions

- `pandas`: DataFrame operations, groupby, aggregation, column selection
- `numpy`: Mathematical operations, log transformations
- `matplotlib/seaborn`: Histograms, boxplots, scatter plots, residual plots
- `statsmodels`: Linear regression, ANOVA F-tests, coefficient extraction
- `sklearn`: LASSO regularization, cross-validation, standardization

2 Data Preparation

2.1 Reading and Subsetting Data

Reading CSV files:

```
# Basic reading
df = pd.read_csv('filename.csv')

# Common parameters
df = pd.read_csv('filename.csv', index_col=0)  # Set first column as index
```

Selecting specific columns:

```
# Method 1: List of columns
columns_to_keep = ['col1', 'col2', 'col3']
df_subset = df[columns_to_keep]

# Method 2: Direct selection
df_subset = df[['col1', 'col2', 'col3']]
```

Practice Task: Given a dataset with 50 columns, select only 10 specific columns including the response variable and create a new DataFrame.

2.2 Understanding Your Data

```
# View first few rows
df.head()

# Check dimensions
df.shape

# Column names
df.columns

# Data types
df.dtypes

# Summary statistics
df.describe()

# Check for missing values
df.isnull().sum()
```

Practice Task: Load a dataset and report: (1) number of rows, (2) number of columns, (3) column names, (4) data types of each column.

3 Exploratory Data Analysis (EDA)

3.1 Aggregation and Grouping

Computing group averages:

```
# Calculate mean by group
group_means = df.groupby('categorical_column')['numeric_column'].mean()

# Multiple aggregations
group_stats = df.groupby('categorical_column').agg({
    'numeric_column': ['mean', 'std', 'count']
})

# Reset index for easier use
group_means_df = df.groupby('categorical_column')['numeric_column'].mean()
    .reset_index()
group_means_df.columns = ['categorical_column', 'mean_value']
```

Practice Task: Given a dataset with states and death rates, compute the average death rate for each state. Store results in a new DataFrame with two columns: State and average death rate.

3.2 Histograms

Creating histograms:

```
# Basic histogram
plt.figure(figsize=(10, 6))
plt.hist(data, bins=20, edgecolor='black')
plt.xlabel('Variable Name')
plt.ylabel('Frequency')
```

```python
plt.title('Histogram of Variable')
plt.show()

# Using seaborn
plt.figure(figsize=(10, 6))
sns.histplot(data, bins=20, kde=True)
plt.xlabel('Variable Name')
plt.ylabel('Count')
plt.title('Histogram of Variable')
plt.show()

# Histogram from grouped data
plt.figure(figsize=(10, 6))
plt.hist(group_means, bins=15, edgecolor='black', alpha=0.7)
plt.xlabel('Average Log Death Rate by State')
plt.ylabel('Number of States')
plt.title('Distribution of State-Level Average Log Death Rates')
plt.show()
```

Practice Task: Create a histogram showing the distribution of average values across groups. Choose appropriate bin size and add labels.

Interpreting histograms:

- Shape: Is it symmetric, skewed left, or skewed right?
- Center: Where is the middle of the distribution?
- Spread: How variable are the values? Wide or narrow range?
- Outliers: Are there unusual values far from the rest?

3.3 Boxplots

Creating boxplots by group:

```python
# Using matplotlib
plt.figure(figsize=(16, 8))
df.boxplot(column='numeric_variable', by='categorical_variable')
plt.xlabel('Categorical Variable')
plt.ylabel('Numeric Variable')
plt.title('Boxplot of Numeric Variable by Categorical Variable')
plt.suptitle('')  # Remove default title
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Using seaborn (recommended for categorical data)
plt.figure(figsize=(16, 8))
sns.boxplot(data=df, x='categorical_variable', y='numeric_variable')
plt.xlabel('Categorical Variable')
plt.ylabel('Numeric Variable')
plt.title('Boxplot of Numeric Variable by Categorical Variable')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Practice Task: Create boxplots of a continuous variable grouped by a categorical variable with 50 categories. Rotate x-axis labels for readability.

Interpreting boxplots:

- Within-group variability: How wide is each box? How long are the whiskers?
- Between-group differences: Do the medians differ across groups?
- Outliers: Are there points beyond the whiskers?
- Overlap: Do the boxes overlap or are they clearly separated?

3.4 Finding Maximum/Minimum Values

Identifying extremes:

```python
# Find maximum value
max_value = df['column_name'].max()

# Find row with maximum value
max_row = df[df['column_name'] == df['column_name'].max()]

# Alternative: using idxmax
max_idx = df['column_name'].idxmax()
max_row = df.loc[max_idx]

# For grouped data
group_means = df.groupby('group_column')['value_column'].mean()
max_group = group_means.idxmax()  # Group with highest mean
max_value = group_means.max()     # Highest mean value

print(f"Group with highest average: {max_group}")
print(f"Value: {max_value}")
```

Practice Task: Given state-level averages, identify which state has the highest average and report both the state name and the value.

4 Linear Regression Analysis

4.1 Simple Linear Regression

Fitting a simple linear model:

```python
# Method 1: Using formula API (similar to R)
import statsmodels.formula.api as smf

model = smf.ols('response ~ predictor', data=df).fit()
print(model.summary())

# Method 2: Using arrays
import statsmodels.api as sm

X = df[['predictor']]
X = sm.add_constant(X)  # Add intercept
```

```python
y = df['response']

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

Practice Task: Fit a linear regression model with one continuous predictor. Extract and report the coefficient, standard error, t-statistic, and p-value for the predictor.

4.2 Extracting Model Information

Getting coefficients and statistics:

```python
# Summary table
print(model.summary())

# Coefficients
coefficients = model.params
print(coefficients)

# P-values
p_values = model.pvalues
print(p_values)

# Confidence intervals
conf_int = model.conf_int(alpha=0.01)  # 99% CI
print(conf_int)

# Specific coefficient
beta_age = model.params['Age65AndOlderPct2010']
p_value_age = model.pvalues['Age65AndOlderPct2010']

# Check significance at alpha = 0.01
is_significant = p_value_age < 0.01
print(f"Significant at 0.01 level: {is_significant}")
```

Practice Task: Fit a model and determine if a specific predictor is significant at the 0.01 level. Report the p-value.

4.3 Multiple Linear Regression

Model with multiple predictors:

```python
# Including both categorical and continuous variables
model = smf.ols('response ~ C(State) + Age65AndOlderPct2010', data=df).fit()
print(model.summary())

# Note: C(State) treats State as categorical
# Without C(), it would treat it as numeric (wrong!)
```

Practice Task: Fit a multiple regression model with one categorical variable (50 levels) and one continuous variable. Verify that the categorical variable is properly encoded.

4.4 Interpreting Coefficients

Continuous predictor coefficient:

- Interpretation: For a one-unit increase in X, the response Y changes by β units, holding all other variables constant.
- Example: If $\beta_{Age} = 0.05$, then for each 1% increase in Age65AndOlderPct2010, log death rate increases by 0.05, holding State constant.

Categorical variable coefficients:

- Python (like R) uses dummy variable encoding with a reference level
- Each coefficient represents the difference from the reference category
- If AL is the reference and $\beta_{NJ} = 0.3$, then NJ has a log death rate 0.3 higher than AL, holding other variables constant

Practice Task: Write out the interpretation of two coefficients from your model: one continuous and one categorical.

4.5 Testing Categorical Variables

ANOVA F-test for categorical variables:

```
# Fit full model (with State)
model_full = smf.ols('response ~ C(State) + Age65AndOlderPct2010', data=df
    ).fit()

# Fit reduced model (without State)
model_reduced = smf.ols('response ~ Age65AndOlderPct2010', data=df).fit()

# ANOVA comparison
anova_results = anova_lm(model_reduced, model_full)
print(anova_results)

# Extract F-statistic and p-value
f_statistic = anova_results['F'][1]
p_value = anova_results['Pr(>F)'][1]

print(f"F-statistic: {f_statistic}")
print(f"P-value: {p_value}")

# Test significance at 0.01 level
is_significant = p_value < 0.01
print(f"State is significant at 0.01 level: {is_significant}")
```

Practice Task: Test whether a categorical variable with many levels is significant in a regression model at the 0.01 level. Report the F-statistic and p-value.

4.6 Making Predictions

Manual prediction calculation:

```python
# Extract coefficients
intercept = model.params['Intercept']
beta_age = model.params['Age65AndOlderPct2010']
beta_nj = model.params['C(State)[T.NJ]']  # If NJ is in model
# Note: For reference level (e.g., AL), coefficient is 0

# Prediction for NJ with Age = 20
pred_nj = intercept + beta_nj + beta_age * 20
print(f"Predicted log death rate for NJ: {pred_nj}")

# Prediction for AL (reference level) with Age = 20
pred_al = intercept + 0 + beta_age * 20  # beta_AL = 0 (reference)
print(f"Predicted log death rate for AL: {pred_al}")
```

Understanding reference levels:

```python
# Check which level is reference
# In Python, alphabetically first level is usually reference
# Can also check from model.params keys
print(model.params.index)
# Reference level won't appear (e.g., if you see C(State)[T.AK],
# C(State)[T.AR], etc. but not C(State)[T.AL], then AL is reference)
```

Practice Task: Given a fitted model with State and Age, manually calculate predicted values for two different states with the same Age value. Show your formula and calculations.

4.7 Model Diagnostics

Residual plots:

```python
# Get fitted values and residuals
fitted_values = model.fittedvalues
residuals = model.resid

# Residuals vs Fitted plot
plt.figure(figsize=(10, 6))
plt.scatter(fitted_values, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted Values')
plt.show()

# Check for patterns:
# - Should see random scatter around 0
# - No funnel shape (indicates constant variance)
# - No curved pattern (indicates linearity)
```

Normal Q-Q plot:

```python
# Q-Q plot for normality
plt.figure(figsize=(10, 6))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title('Normal Q-Q Plot')
plt.show()
```

```

# Alternative using statsmodels
import statsmodels.api as sm
fig = sm.qqplot(residuals, line='45', fit=True)
plt.title('Normal Q-Q Plot')
plt.show()

# Check: Points should fall close to diagonal line
```

Scale-Location plot:

```
# Scale-Location plot (check homoscedasticity)
plt.figure(figsize=(10, 6))
plt.scatter(fitted_values, np.sqrt(np.abs(residuals)), alpha=0.5)
plt.xlabel('Fitted Values')
plt.ylabel('Square Root of |Residuals|')
plt.title('Scale-Location Plot')
plt.show()

# Check: Should see roughly horizontal band
```

Practice Task: Create residual and Q-Q plots for your model. Assess: (1) linearity, (2) constant variance, (3) normality of residuals. Write 2-3 sentences summarizing whether assumptions are reasonably met.

What to look for:

- **Linearity:** Residuals vs Fitted should show random scatter with no pattern
- **Constant Variance:** Spread of residuals should be constant across fitted values
- **Normality:** Q-Q plot points should follow the diagonal line closely
- **Independence:** (Usually assumed; no obvious time/spatial patterns)

5 LASSO Regression

5.1 Understanding LASSO

LASSO (Least Absolute Shrinkage and Selection Operator) is a regularization method that:

- Performs variable selection by shrinking some coefficients to exactly zero
- Helps prevent overfitting when you have many predictors
- Uses cross-validation to choose the optimal penalty parameter (λ)

5.2 Data Preparation for LASSO

Creating design matrix:

```
# Separate response and predictors
y = df['log_death_rate']

# For categorical variables, create dummy variables
X = pd.get_dummies(df[predictor_columns], drop_first=False)
```

```

# Important: We'll handle State separately to force it in
```

5.3 Forcing Variables into LASSO

Creating penalty factor to force State variables:

```
# Create dummy variables for all predictors
X_all = pd.get_dummies(df.drop('log_death_rate', axis=1),
                       columns=['State'],
                       drop_first=True)

# Identify which columns are State dummies
state_columns = [col for col in X_all.columns if col.startswith('State_')]

# To force State in: manually remove State dummies from penalization
# We'll do this by separating State and other predictors
X_state = X_all[state_columns]
X_other = X_all.drop(columns=state_columns)

# Standardize only the non-State predictors
scaler = StandardScaler()
X_other_scaled = scaler.fit_transform(X_other)
X_other_scaled = pd.DataFrame(X_other_scaled,
                              columns=X_other.columns,
                              index=X_other.index)

# Combine: State (unpenalized in implementation) + other (penalized)
# Note: This requires custom implementation or offsetting
```

Alternative approach - simpler for exam:

```
# Run LASSO on non-State variables only
# Then add State back to final model
predictors_for_lasso = ['PovertyAllAgesPct', 'UnempRate2019',
                        'PctEmpFIRE', 'PopDensity2010',
                        'Age65AndOlderPct2010', 'Ed5CollegePlusPct',
                        'WhiteNonHispanicPct2010', 'HiCreativeClass2000']
                        # etc. - all continuous predictors

X = df[predictors_for_lasso]
y = df['log_death_rate']
```

5.4 Running LASSO with Cross-Validation

Using LassoCV:

```
from sklearn.linear_model import LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```

# Set up cross-validation
np.random.seed(1)  # For reproducibility
cv = KFold(n_splits=10, shuffle=True, random_state=1)

# Fit LASSO with cross-validation
lasso_cv = LassoCV(cv=cv, random_state=1, max_iter=10000)
lasso_cv.fit(X_scaled, y)

# Best lambda
best_lambda = lasso_cv.alpha_
print(f"Best lambda: {best_lambda}")
```

Using lambda.1se criterion:

```
# For lambda.1se, we need to use LassoCV differently
alphas = np.logspace(-4, 1, 100)

lasso_cv = LassoCV(cv=cv, alphas=alphas, random_state=1, max_iter=10000)
lasso_cv.fit(X_scaled, y)

# Get cross-validation results
cv_results = []
for alpha in alphas:
    lasso = Lasso(alpha=alpha, max_iter=10000)
    scores = []
    for train_idx, test_idx in cv.split(X_scaled):
        X_train, X_test = X_scaled[train_idx], X_scaled[test_idx]
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
        lasso.fit(X_train, y_train)
        score = lasso.score(X_test, y_test)
        scores.append(score)
    cv_results.append((alpha, np.mean(scores), np.std(scores)))

# Find lambda.1se (largest lambda within 1 SE of best)
cv_df = pd.DataFrame(cv_results, columns=['alpha', 'mean_score', '
    std_score'])
best_score = cv_df['mean_score'].max()
best_std = cv_df[cv_df['mean_score'] == best_score]['std_score'].values[0]
threshold = best_score - best_std

lambda_1se = cv_df[cv_df['mean_score'] >= threshold]['alpha'].max()
print(f"Lambda 1se: {lambda_1se}")
```

5.5 Extracting Selected Variables

Identifying non-zero coefficients:

```
# Fit LASSO with chosen lambda
lasso_final = Lasso(alpha=lambda_1se, max_iter=10000)
lasso_final.fit(X_scaled, y)

# Get coefficients
coefficients = pd.DataFrame({
```

```
    'variable': X.columns,
    'coefficient': lasso_final.coef_
})

# Select variables with non-zero coefficients
selected_vars = coefficients[coefficients['coefficient'] != 0]['variable'
    ].tolist()
print("Variables selected by LASSO:")
print(selected_vars)
```

Practice Task: Run LASSO with 10-fold cross-validation, lambda.1se criterion, and seed=1. List all variables with non-zero coefficients.

5.6 Final Model After LASSO

Building the final linear model:

```
# Combine State with LASSO-selected variables
# Always include Age65AndOlderPct2010 as specified
final_predictors = ['State'] + selected_vars
if 'Age65AndOlderPct2010' not in final_predictors:
    final_predictors.append('Age65AndOlderPct2010')

# Create formula for regression
formula = 'log_death_rate ~ C(State) + ' + ' + '.join(
    [var for var in final_predictors if var != 'State']
)

# Fit final model
fit_final = smf.ols(formula, data=df).fit()
print(fit_final.summary())
```

Practice Task: After running LASSO, create a final linear model that includes: (1) State as categorical, (2) all LASSO-selected variables, (3) Age65AndOlderPct2010 regardless of LASSO selection.

6 Workflow

6.1 Complete Analysis Checklist

Step 1: Data Preparation

1. Load data using `pd.read_csv()`
2. Select relevant columns
3. Check for missing values
4. Verify data types

Step 2: EDA

1. Compute summary statistics by groups
2. Create histograms of aggregated data

3. Create boxplots by categorical variable
4. Identify maximum/minimum values
5. Write brief interpretations (2-3 sentences)

Step 3: Simple Regression

1. Fit simple linear model
2. Extract p-value for predictor
3. Test significance at specified level (e.g., 0.01)
4. Report result

Step 4: Multiple Regression

1. Fit model with categorical and continuous predictors
2. Test significance of continuous predictor
3. Interpret coefficient of continuous predictor
4. Compare with simple regression results
5. Test significance of categorical variable using F-test
6. Make manual predictions for specific cases
7. Create diagnostic plots
8. Assess model assumptions (2-3 sentences)

Step 5: Variable Selection

1. Prepare data for LASSO
2. Run LASSO with cross-validation
3. Extract selected variables
4. Fit final linear model with selected variables
5. Test significance of key variables
6. Identify extreme cases (max/min predictions)
7. Write summary of findings (3-4 sentences)

7 Common Python Patterns for This Exam

7.1 Pattern 1: Group Statistics and Visualization

```
# Calculate group means
group_means = df.groupby('group_var')['value_var'].mean()

# Create histogram
plt.figure(figsize=(10, 6))
plt.hist(group_means, bins=20, edgecolor='black')
plt.xlabel('Average Value')
plt.ylabel('Frequency')
plt.title('Distribution of Group Averages')
plt.show()

# Create boxplot
plt.figure(figsize=(16, 8))
sns.boxplot(data=df, x='group_var', y='value_var')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Find maximum
max_group = group_means.idxmax()
max_value = group_means.max()
```

7.2 Pattern 2: Simple Regression and Significance Testing

```
# Fit model
model = smf.ols('response ~ predictor', data=df).fit()

# Extract p-value
p_value = model.pvalues['predictor']

# Test significance
alpha = 0.01
is_significant = p_value < alpha
print(f"P-value: {p_value:.6f}")
print(f"Significant at {alpha} level: {is_significant}")
```

7.3 Pattern 3: Multiple Regression with Categorical Variable

```
# Fit model
model = smf.ols('response ~ C(category) + continuous', data=df).fit()

# Test continuous predictor
p_value_cont = model.pvalues['continuous']

# Test categorical variable
model_reduced = smf.ols('response ~ continuous', data=df).fit()
anova_result = anova_lm(model_reduced, model)
```

```
p_value_cat = anova_result['Pr(>F)'][1]

# Extract coefficient
beta_cont = model.params['continuous']

# Manual prediction
intercept = model.params['Intercept']
beta_category = model.params['C(category)[T.level]']  # If not reference
pred = intercept + beta_category + beta_cont * value
```

7.4 Pattern 4: Model Diagnostics

```
# Get residuals and fitted values
fitted = model.fittedvalues
resid = model.resid

# Residual plot
plt.figure(figsize=(10, 6))
plt.scatter(fitted, resid, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted')
plt.show()

# Q-Q plot
plt.figure(figsize=(10, 6))
stats.probplot(resid, dist="norm", plot=plt)
plt.title('Normal Q-Q Plot')
plt.show()
```

7.5 Pattern 5: LASSO and Final Model

```
# Prepare data
X = df[predictor_list]
y = df['response']

# Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# LASSO with CV
np.random.seed(1)
cv = KFold(n_splits=10, shuffle=True, random_state=1)
lasso = LassoCV(cv=cv, random_state=1, max_iter=10000)
lasso.fit(X_scaled, y)

# Extract selected variables
selected = [predictor_list[i] for i in range(len(predictor_list))
            if lasso.coef_[i] != 0]

```

```
# Build final model
formula = 'response ~ C(State) + ' + ' + '.join(selected)
fit_final = smf.ols(formula, data=df).fit()
```

8 Key Concepts to Master

8.1 Statistical Concepts

1. **P-value interpretation:** Probability of observing data as extreme as ours if null hypothesis is true. Small p-value (¡ α) means reject null.
2. **Significance level:** $\alpha = 0.01$ means 1% chance of Type I error. Reject null if p-value ¡ 0.01.
3. **Coefficient interpretation:** In linear regression, β represents change in response for one-unit change in predictor, holding others constant.
4. **R-squared:** Proportion of variance in response explained by model. Higher is better, but watch for overfitting.
5. **F-test:** Tests whether a group of coefficients (e.g., all State dummies) are jointly significant.
6. **Model assumptions:** Linearity, independence, constant variance, normality of residuals.
7. **Dummy variables:** For categorical variable with k levels, create k-1 dummies. One level is reference (coefficient = 0).
8. **LASSO:** Shrinks coefficients, some to zero. Performs variable selection. Lambda controls penalty strength.
9. **Cross-validation:** Assesses model performance on held-out data. Helps choose lambda.
10. **Lambda.1se:** Largest lambda whose cross-validation error is within 1 standard error of minimum. Prefers simpler models.

8.2 Python Skills

1. Reading CSV files into DataFrames
2. Selecting and subsetting columns
3. Grouping and aggregating data
4. Creating histograms and boxplots with proper labels
5. Fitting linear models with continuous and categorical predictors
6. Extracting coefficients, p-values, and confidence intervals
7. Performing F-tests for categorical variables
8. Making manual predictions from regression equations
9. Creating residual and Q-Q plots

10. Running LASSO with cross-validation
11. Standardizing features
12. Extracting non-zero coefficients from LASSO
13. Building final models with selected variables

9 Practice Problems

9.1 Problem Set 1: Data Preparation and EDA

Problem 1.1: Load a dataset with 3000 rows and 40 columns. Select 15 specific columns. Report dimensions of new dataset.

Problem 1.2: Compute average of a continuous variable grouped by a categorical variable with 50 levels. Create a histogram of these 50 averages.

Problem 1.3: Create boxplots of a continuous variable by the same categorical variable. Rotate x-axis labels 90 degrees.

Problem 1.4: Identify which group has the highest average. Report group name and value.

Problem 1.5: Write 2-3 sentences describing variability across groups based on histogram and boxplots.

9.2 Problem Set 2: Simple Regression

Problem 2.1: Fit simple linear regression: $y \sim x$. Extract p-value for x.

Problem 2.2: Test if x is significant at $\alpha = 0.01$ level. State conclusion.

Problem 2.3: Extract coefficient of x and interpret it in context.

Problem 2.4: Compute R^2 and interpret it.

9.3 Problem Set 3: Multiple Regression

Problem 3.1: Fit model: $y \sim$ categorical (50 levels) + continuous. Ensure categorical is properly encoded.

Problem 3.2: Test significance of continuous predictor at $\alpha = 0.01$. Compare to simple regression.

Problem 3.3: Write interpretation of continuous predictor coefficient. Explain why it differs from simple regression.

Problem 3.4: Perform F-test to determine if categorical variable is significant at $\alpha = 0.01$.

Problem 3.5: Manually calculate predicted y for two specific levels of categorical variable (including reference level) with same continuous predictor value. Show formulas.

Problem 3.6: Create residual vs fitted plot and Q-Q plot. Write 2-3 sentences assessing model assumptions.

9.4 Problem Set 4: LASSO and Final Model

Problem 4.1: Run LASSO on 20 continuous predictors using 10-fold CV, seed=1, lambda.1se. List selected variables.

Problem 4.2: Fit final model including: categorical variable (50 levels), LASSO-selected variables, and one specific variable regardless of LASSO selection.

Problem 4.3: Test if categorical variable is significant at $\alpha = 0.01$ in final model.

Problem 4.4: Test if specific continuous variable is significant at $\alpha = 0.01$ in final model.

Problem 4.5: Among all categorical levels, which has highest predicted value controlling for other variables?

Problem 4.6: Write 3-4 sentence summary of findings from final model.

10 Exam Tips

10.1 Time Management

- Read all questions first (5 minutes)
- EDA questions: 30-40 minutes
- Simple regression: 15-20 minutes
- Multiple regression: 40-50 minutes
- LASSO and final model: 50-60 minutes
- Review and check: 15 minutes

10.2 Common Mistakes to Avoid

- Forgetting to use `C()` for categorical variables in formulas
- Not rotating x-axis labels on boxplots with many categories
- Comparing p-values to wrong significance level
- Forgetting that reference level has coefficient = 0
- Not standardizing data before LASSO
- Forgetting to set random seed for reproducibility
- Not including all required variables in final model
- Writing interpretations without specifying "holding other variables constant"

10.3 What to Show

- All code used to produce answers
- All plots with proper labels and titles
- All numerical answers clearly stated
- Brief written interpretations where requested
- Formulas for manual calculations

10.4 Getting Unstuck

- If LASSO doesn't work: use provided backup variable list
- If plot formatting is messy: move on, focus on content
- If coefficient extraction confusing: print entire `model.params`
- If prediction calculation stuck: show formula even if can't compute
- Use `.head()`, `.shape`, `.columns` to debug data issues

11 Final Checklist Before Exam

☐ Can load CSV and select columns

☐ Can compute group means

☐ Can create histogram with labels

☐ Can create boxplot with rotated labels

☐ Can find maximum value and corresponding group

☐ Can fit simple linear regression

☐ Can extract and interpret p-values

☐ Can test significance at specified level

☐ Can fit multiple regression with categorical and continuous predictors

☐ Can interpret coefficients

☐ Can perform F-test for categorical variable

☐ Can manually calculate predictions

☐ Can create residual and Q-Q plots

☐ Can assess model assumptions

☐ Can run LASSO with cross-validation

☐ Can extract selected variables from LASSO

☐ Can build final model with multiple variable sources

☐ Can write concise interpretations

You are ready for this exam. Work through these practice problems systematically, and you will perform well.