Exercise 19

```
import os
import numpy as np
import polars as pl
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, StratifiedKFold,
GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.utils import resample

# Import data and clean
leads = (pl.read_parquet(os.path.join('data', 'leads.parquet'))
    # Make the outcome binary where "Qualified" = 1
    .with_columns(
        pl.when(pl.col('Stage') == 'Qualified').then(1)
        .when(pl.col('Stage') == 'Disqualified').then(0)
        .alias('qualified')
    )
    # Dummy code Industry, Employees, TimeZone, LeadSource, and EmployeeId
    .to_dummies(
        columns = ['Industry', 'Employees', 'TimeZone', 'LeadSource',
'EmployeeId'],
        drop_first = False
    )
    .select(pl.exclude('Stage', 'Amount', 'Industry_Business',
'Employees_Small', 'TimeZone_EST', 'LeadSource_Purchased List',
'EmployeeId_1'))
    # Log-transform all activity counts
    .with_columns((pl.col('ActivityTypeEmail') +
1).log().alias('log_ActivityTypeEmail'))
    .with_columns((pl.col('ActivityTypePhone Call') +
1).log().alias('log_ActivityTypePhoneCall'))
    .with_columns((pl.col('ActivityTypeEmail Response') +
1).log().alias('log_ActivityTypeEmailResponse'))
    .with_columns((pl.col('ActivityTypeMeeting') +
1).log().alias('log_ActivityTypeMeeting'))
    .with_columns((pl.col('ActivityTypeLead Handraise') +
1).log().alias('log_ActivityTypeLeadHandraise'))
    .with_columns((pl.col('ActivityTypeWeb Schedule') +
1).log().alias('log_ActivityTypeWebSchedule'))
    # Rename columns to remove spaces
```

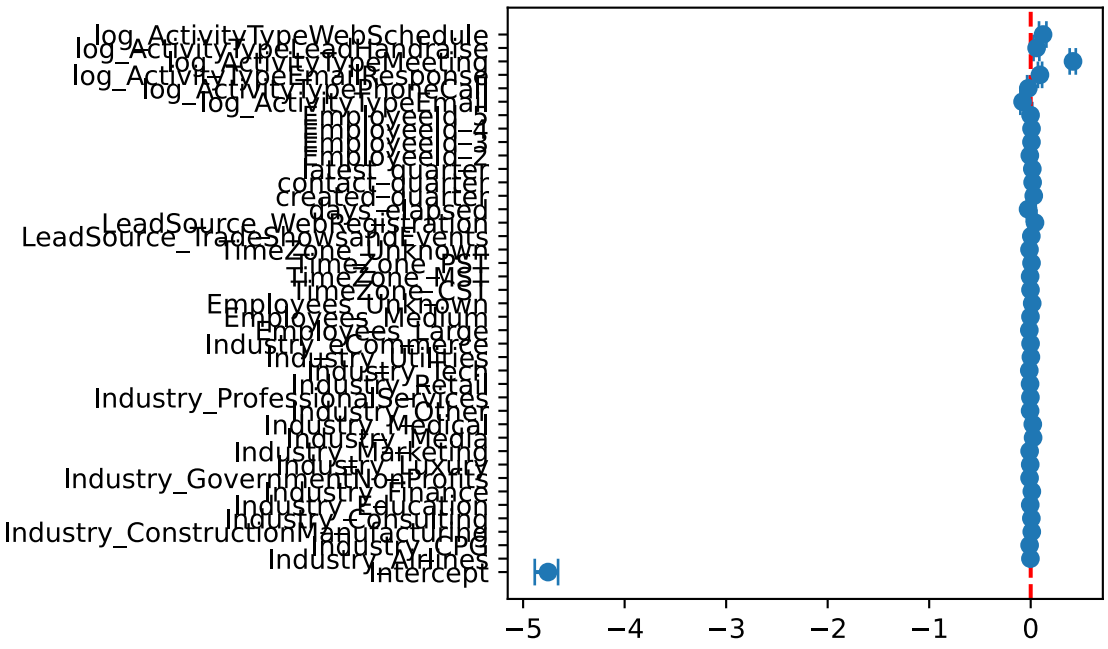
```
    .rename({
        'Industry_Construction & Manufacturing':
'Industry_ConstructionManufacturing',
        'Industry_Government & Non-Profits': 'Industry_GovernmentNonProfits',
        'Industry_Professional Services': 'Industry_ProfessionalServices',
        'LeadSource_Trade Shows and Events': 'LeadSource_TradeShowsandEvents',
        'LeadSource_Web Registration': 'LeadSource_WebRegistration'
    })
    # Remove the original activity count columns
    .select(
        pl.exclude(
            'ActivityTypeEmail', 'ActivityTypePhone Call', 'ActivityTypeEmail
Response',
            'ActivityTypeMeeting', 'ActivityTypeLead Handraise',
'ActivityTypeWeb Schedule'
        )
    )
)

# Specify the design matrix and outcome
X = leads.select(pl.exclude('qualified'))
y = leads.select('qualified')

# Specify predictors
predictors = [
    'Industry_Airlines', 'Industry_CPG',
    'Industry_ConstructionManufacturing', 'Industry_Consulting',
    'Industry_Education', 'Industry_Finance', 'Industry_GovernmentNonProfits',
    'Industry_Luxury', 'Industry_Marketing', 'Industry_Media',
'Industry_Medical',
    'Industry_Other', 'Industry_ProfessionalServices', 'Industry_Retail',
    'Industry_Tech', 'Industry_Utilities', 'Industry_eCommerce',
'Employees_Large',
    'Employees_Medium', 'Employees_Unknown', 'TimeZone_CST', 'TimeZone_MST',
    'TimeZone_PST', 'TimeZone_Unknown', 'LeadSource_TradeShowsandEvents',
    'LeadSource_WebRegistration', 'days_elapsed', 'created_quarter',
    'contact_quarter', 'latest_quarter', 'EmployeeId_2', 'EmployeeId_3',
    'EmployeeId_4', 'EmployeeId_5', 'log_ActivityTypeEmail',
    'log_ActivityTypePhoneCall', 'log_ActivityTypeEmailResponse',
    'log_ActivityTypeMeeting', 'log_ActivityTypeLeadHandraise',
    'log_ActivityTypeWebSchedule'
]
# Split data into training (temp) and testing data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.2, random_state = 42, stratify = y.to_numpy()
)
y_train = np.ravel(y_train.to_numpy())
y_test = np.ravel(y_test.to_numpy())
```

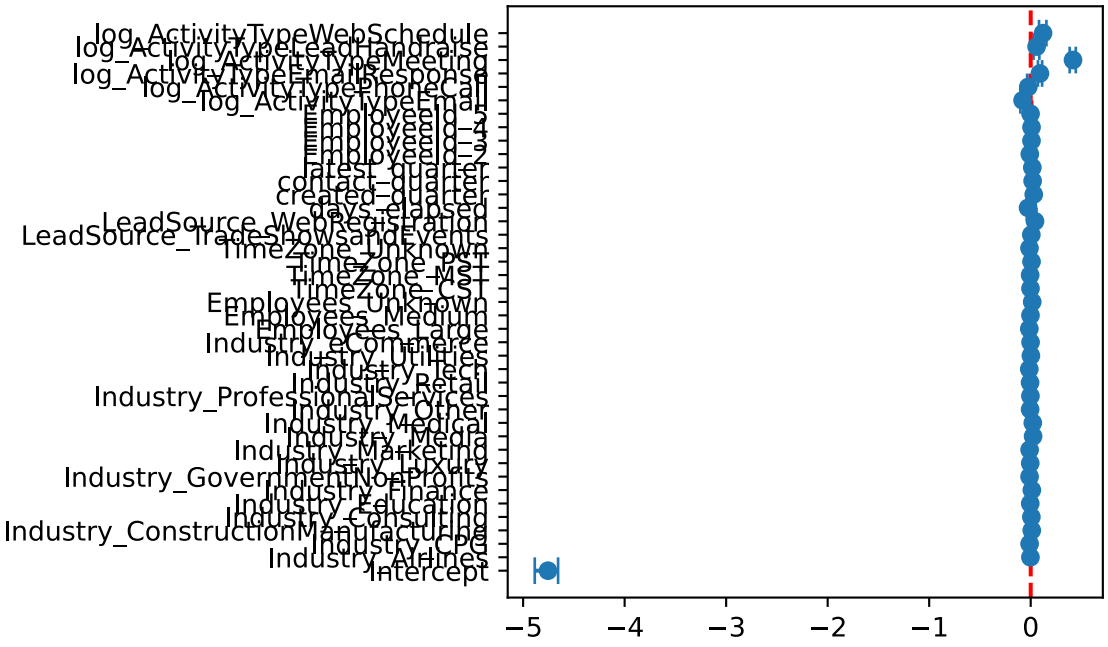
```
# Create a pipeline
ridge_pipe = Pipeline([
  ('feature_engineering', StandardScaler()),
  ('classification', LogisticRegression(penalty = 'l2'))
])

# Create a grid using a log scale (inverse of regularization strength)
hyper_grid = {'classification__C': np.logspace(-10, 10, 30)}

# Use the grid to tune hyperparameters via cross-validation
kfold_cv = StratifiedKFold(n_splits = 5)
tune = GridSearchCV(
  ridge_pipe, hyper_grid, scoring = 'accuracy',
  cv = kfold_cv, n_jobs = 1, refit = True
)
tune.fit(X_train, y_train)

# Extract the best hyperparameter and CV accuracy
best_C = tune.best_params_['classification__C']
best_cv_score = tune.best_score_

print(
  f'Best C: {best_C}',
  f'Best CV Accuracy: {best_cv_score:.4f}',
  sep = '\n'
)

# Extract point estimates from refit best_estimator
best_estimator = tune.best_estimator_
ridge_final = best_estimator.named_steps['classification']
intercept = ridge_final.intercept_.ravel()
slopes = ridge_final.coef_.ravel()
point_est = np.concatenate([intercept, slopes])

point_est

# Update the pipeline for bootstrapping
ridge_pipe = Pipeline([
  ('feature_engineering', StandardScaler()),
  ('classification', LogisticRegression(penalty = 'l2', C = best_C))
])

# Bootstrap confidence intervals
n_samples = 100
boot_est = np.empty((n_samples, len(point_est)))
for b in range(n_samples):
  # Resample data with replacement
```

```
  X_b, y_b = resample(X_train, y_train, replace = True, random_state = 42 + b)

  # Fit logistic regression on resampled data
  ridge_pipe.fit(X_b, y_b)
  ridge_b = ridge_pipe.named_steps['classification']

  # Extract point estimates using resampled data
  intercept_b = ridge_b.intercept_.ravel()
  slopes_b = ridge_b.coef_.ravel()
  point_est_b = np.concatenate([intercept_b, slopes_b])

  # Save point estimates using resampled data
  boot_est[b, :] = point_est_b

ci_lower = np.percentile(boot_est, 2.5, axis=0)
ci_upper = np.percentile(boot_est, 97.5, axis=0)

# Output of point and interval estimates
int_est = pl.DataFrame({
  'predictors': ['Intercept'] + predictors,
  'point_est': point_est,
  'ci_lower': ci_lower,
  'ci_upper': ci_upper
})

# Plot the confidence intervals
plt.figure(figsize=(4, 4))
plt.errorbar(
  int_est['point_est'],
  int_est['predictors'],
  xerr=[
    int_est['point_est'] - int_est['ci_lower'],
    int_est['ci_upper'] - int_est['point_est']
  ],
  fmt='o',
  capsize=5,
  label='Estimates')
plt.axvline(0, color='red', linestyle='--', label='y=0')

# Output of point and interval estimates
int_est = pl.DataFrame({
  'predictors': ['Intercept'] + predictors,
  'point_est': point_est,
  'ci_lower': ci_lower,
  'ci_upper': ci_upper
})

# Plot the confidence intervals
```

```
plt.figure(figsize=(4, 4))
plt.errorbar(
  int_est['point_est'],
  int_est['predictors'],
  xerr=[
    int_est['point_est'] - int_est['ci_lower'],
    int_est['ci_upper'] - int_est['point_est']
  ],
  fmt='o',
  capsize=5,
  label='Estimates')
plt.axvline(0, color='red', linestyle='--', label='y=0')
```

```
Best C: 0.0007880462815669921
Best CV Accuracy: 0.9871
```



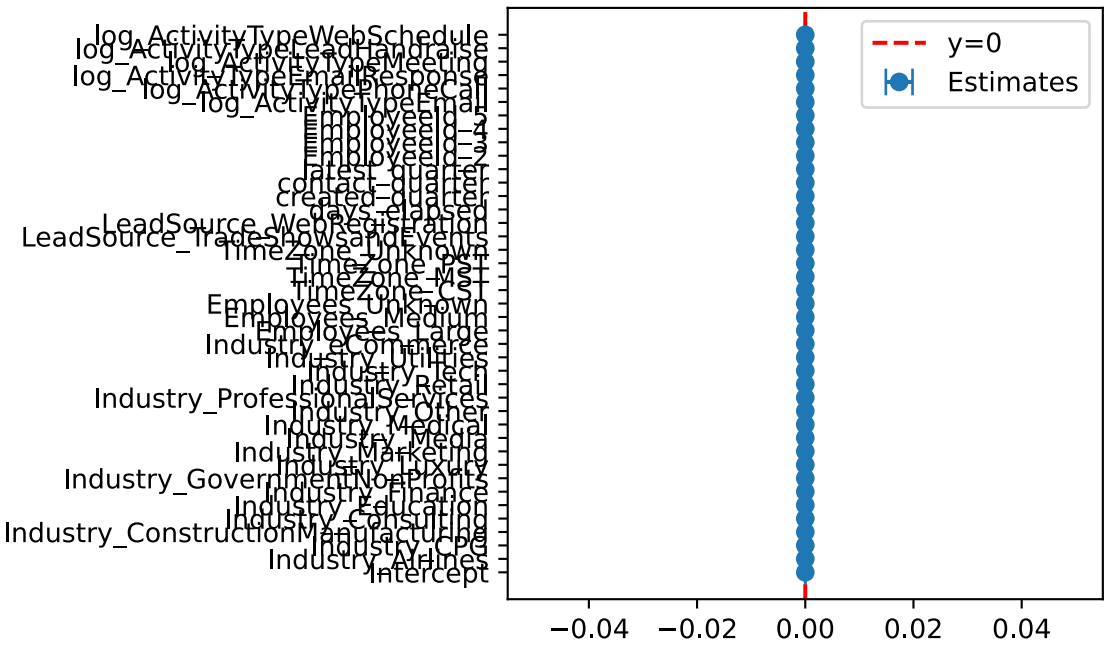



```
# Create a pipeline for lasso regression
lasso_pipe = Pipeline([
  ('feature_engineering', StandardScaler()),
  ('classification', LogisticRegression(penalty='l1', solver='liblinear'))  #
L1 requires liblinear or saga
])

# Create a grid using a range of log spaces (inverse of regularization
strength)
hyper_grid = {'classification__C': np.logspace(-10, 10, 30)}

# Use the grid to tune hyperparameters via cross-validation
kfold_cv = StratifiedKFold(n_splits=5)
tune = GridSearchCV(
  lasso_pipe, hyper_grid, scoring='accuracy',
  cv=kfold_cv, n_jobs=1, refit=True
)
tune.fit(X_train, y_train.ravel())

# Extract the best hyperparameter and CV accuracy
best_C = tune.best_params_['classification__C']
best_cv_score = tune.best_score_

print(
  f'Best C: {best_C}',
  f'Best CV Accuracy: {best_cv_score:.4f}',
  sep='\n'
```

```
)

# Extract point estimates from refit best_estimator
best_estimator = tune.best_estimator_
lasso_final = best_estimator.named_steps['classification']
intercept = lasso_final.intercept_.ravel()
slopes = lasso_final.coef_.ravel()
point_est = np.concatenate([intercept, slopes])

point_est

# Update the pipeline for bootstrapping
lasso_pipe = Pipeline([
  ('feature_engineering', StandardScaler()),
  ('classification', LogisticRegression(penalty='l1', solver='liblinear',
C=best_C))
])

# Bootstrap confidence intervals
n_samples = 100
boot_est = np.empty((n_samples, len(point_est)))
for b in range(n_samples):
  # Resample data with replacement
  X_b, y_b = resample(X_train, y_train, replace=True, random_state=42 + b)

  # Fit logistic regression on resampled data
  lasso_pipe.fit(X_b, y_b)
  lasso_b = lasso_pipe.named_steps['classification']

  # Extract point estimates using resampled data
  intercept_b = lasso_b.intercept_.ravel()
  slopes_b = lasso_b.coef_.ravel()
  point_est_b = np.concatenate([intercept_b, slopes_b])

  # Save point estimates using resampled data
  boot_est[b, :] = point_est_b

ci_lower = np.percentile(boot_est, 2.5, axis=0)
ci_upper = np.percentile(boot_est, 97.5, axis=0)

# Output of point and interval estimates
int_est = pl.DataFrame({
  'predictors': ['Intercept'] + predictors,
  'point_est': point_est,
  'ci_lower': ci_lower,
  'ci_upper': ci_upper
})
```

```
# Plot the confidence intervals
plt.figure(figsize=(4, 4))
plt.errorbar(
  int_est['point_est'],
  int_est['predictors'],
  xerr=[
    int_est['point_est'] - int_est['ci_lower'],
    int_est['ci_upper'] - int_est['point_est']
  ],
  fmt='o',
  capsize=5,
  label='Estimates')
plt.axvline(0, color='red', linestyle='--', label='y=0')
plt.legend()
plt.show()
```

```
Best C: 1e-10
Best CV Accuracy: 0.9870
```




```
# Create a pipeline for elastic net regression
elastic_net_pipe = Pipeline([
   ('feature_engineering', StandardScaler()),
   ('classification', LogisticRegression(
    penalty = 'elasticnet',
    solver = 'saga', max_iter=5000, random_state=42
```

```
  ))
])

# Create a grid using a log scale (inverse of regularization strength) and a
range of l1 ratios
hyper_grid = {
  'classification__C': np.logspace(-10, 10, 30),
  'classification__l1_ratio': [0.1, 0.25, 0.5, 0.75, 0.9]
}

# Use the grid to tune hyperparameters via cross-validation
kfold_cv = StratifiedKFold(n_splits=5)
tune = GridSearchCV(
  elastic_net_pipe, hyper_grid, scoring='accuracy',
  cv=kfold_cv, n_jobs=1, refit=True
)
tune.fit(X_train, y_train.ravel())

# Extract the best hyperparameters and CV accuracy
best_C = tune.best_params_['classification__C']
best_l1 = tune.best_params_['classification__l1_ratio']
best_cv_score = tune.best_score_

print(
  f'Best C: {best_C}',
  f'Best l1_ratio: {best_l1}',
  f'Best CV Accuracy: {best_cv_score:.4f}',
  sep = '\n'
)

# Extract point estimates from refit best_estimator
best_estimator = tune.best_estimator_
elastic_net_final = best_estimator.named_steps['classification']
intercept = elastic_net_final.intercept_.ravel()
slopes = elastic_net_final.coef_.ravel()
point_est = np.concatenate([intercept, slopes])

point_est
'''
# Update the pipeline for bootstrapping
elastic_net_pipe = Pipeline([
   ('feature_engineering', StandardScaler()),
   ('classification', LogisticRegression(
    penalty = 'elasticnet',
    solver = 'saga', max_iter=5000, random_state=42, C=best_C,
l1_ratio=best_l1
  ))
])
```

```
# Bootstrap confidence intervals
n_samples = 100
boot_est = np.empty((n_samples, len(point_est)))
for b in range(n_samples):
  # Resample data with replacement
  X_b, y_b = resample(X_train, y_train, replace=True, random_state=42 + b)

  # Fit logistic regression on resampled data
  elastic_net_pipe.fit(X_b, y_b)
  elastic_net_b = elastic_net_pipe.named_steps['classification']

  # Extract point estimates using resampled data
  intercept_b = elastic_net_b.intercept_.ravel()
  slopes_b = elastic_net_b.coef_.ravel()
  point_est_b = np.concatenate([intercept_b, slopes_b])

  # Save point estimates using resampled data
  boot_est[b, :] = point_est_b

ci_lower = np.percentile(boot_est, 2.5, axis=0)
ci_upper = np.percentile(boot_est, 97.5, axis=0)

# Output of point and interval estimates
int_est = pl.DataFrame({
  'predictors': ['Intercept'] + predictors,
  'point_est': point_est,
  'ci_lower': ci_lower,
  'ci_upper': ci_upper
})

# Plot the confidence intervals
plt.figure(figsize=(4, 4))
plt.errorbar(
  int_est['point_est'],
  int_est['predictors'],
  xerr=[
    int_est['point_est'] - int_est['ci_lower'],
    int_est['ci_upper'] - int_est['point_est']
  ],
  fmt='o',
  capsize=5,
  label='Estimates')
plt.axvline(0, color='red', linestyle='--', label='y=0')
plt.legend()
plt.show()'''
```

```
Best C: 0.45203536563602403
Best l1_ratio: 0.1
Best CV Accuracy: 0.9871
```

```
"\n# Update the pipeline for bootstrapping\nelastic_net_pipe = Pipeline([\n
('feature_engineering', StandardScaler()),\n   ('classification',
LogisticRegression(\n    penalty = 'elasticnet', \n    solver = 'saga',
max_iter=5000, random_state=42, C=best_C, l1_ratio=best_l1\n  ))\n])\n\n#
Bootstrap confidence intervals\nn_samples = 100\nboot_est =
np.empty((n_samples, len(point_est)))\nfor b in range(n_samples):\n  #
Resample data with replacement\n  X_b, y_b = resample(X_train, y_train,
replace=True, random_state=42 + b)\n\n  # Fit logistic regression on resampled
data\n  elastic_net_pipe.fit(X_b, y_b)\n  elastic_net_b =
elastic_net_pipe.named_steps['classification']\n\n  # Extract point estimates
using resampled data\n  intercept_b = elastic_net_b.intercept_.ravel()\n
slopes_b = elastic_net_b.coef_.ravel()\n  point_est_b =
np.concatenate([intercept_b, slopes_b])\n\n  # Save point estimates using
resampled data\n  boot_est[b, :] = point_est_b\n\nci_lower =
np.percentile(boot_est, 2.5, axis=0)\nci_upper = np.percentile(boot_est, 97.5,
axis=0)\n\n# Output of point and interval estimates\nint_est =
pl.DataFrame({\n  'predictors': ['Intercept'] + predictors,\n  'point_est':
point_est,\n  'ci_lower': ci_lower,\n  'ci_upper': ci_upper\n})\n\n# Plot the
confidence intervals\nplt.figure(figsize=(4, 4))\nplt.errorbar(\n
int_est['point_est'], \n  int_est['predictors'],\n  xerr=[\n
int_est['point_est'] - int_est['ci_lower'], \n    int_est['ci_upper'] -
int_est['point_est']\n  ], \n  fmt='o', \n  capsize=5, \n
label='Estimates')\nplt.axvline(0, color='red', linestyle='--',
label='y=0')\nplt.legend()\nplt.show()"
```

For this analysis, we ran three types of penalized regression: ridge, lasso, and elastic net. From our accuracy scores, ridge and elastic net both had the highest accuracy at 0.9871, while lasso regression had an accuracy of 0.9870, which was the same accuracy as the non-penalized logit model. Because of computational complexity, we did not create a confidence interval plot for the elastic net regression model. For the ridge regression confidence interval plot, we see thatmost of our predictors are not statistically significant, including the dummy variables for every industry and quarters for creation/reponse. Having a greater level of activity for things such as meetings, email responses, lead handraises, and web schedules are associated with a higher log-likelihood of being a qualified lead. Having a higher number of calls and emails is associated with a lower log-likelihood of being a qualified lead.