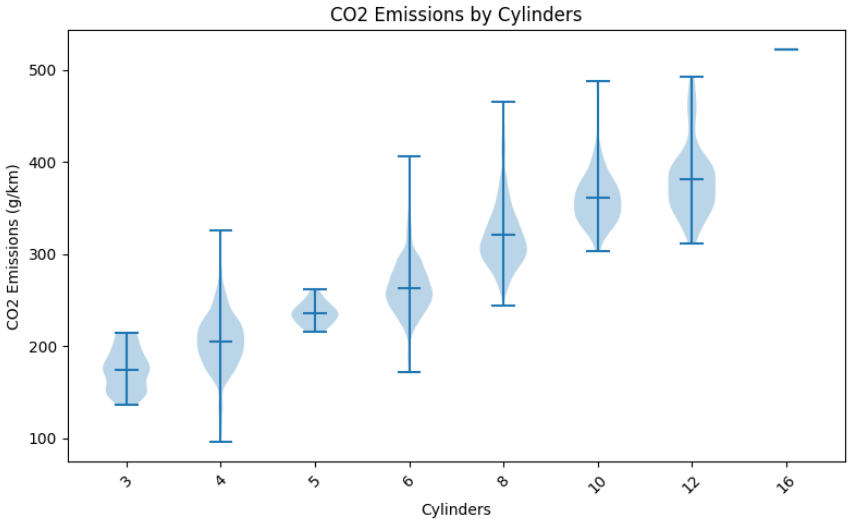
Term paper - EBA35303 Machine Learning and Forecasting

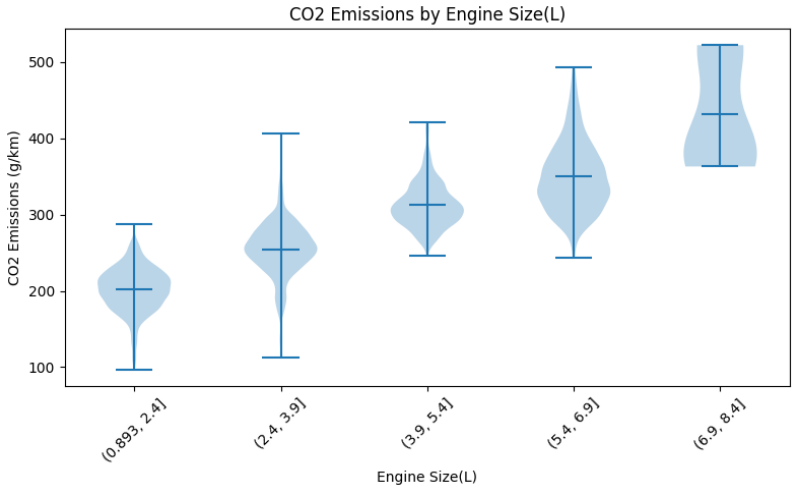
1 Linear Regression

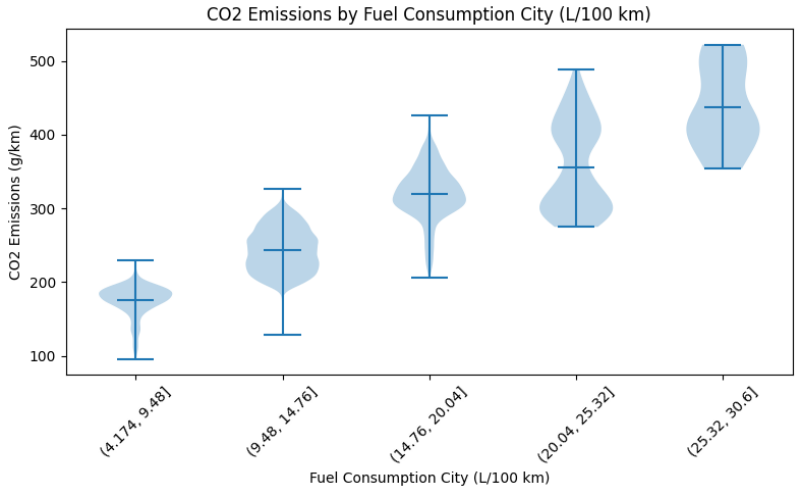
1.1Violin plots where CO2 emissions are on the y-axis.

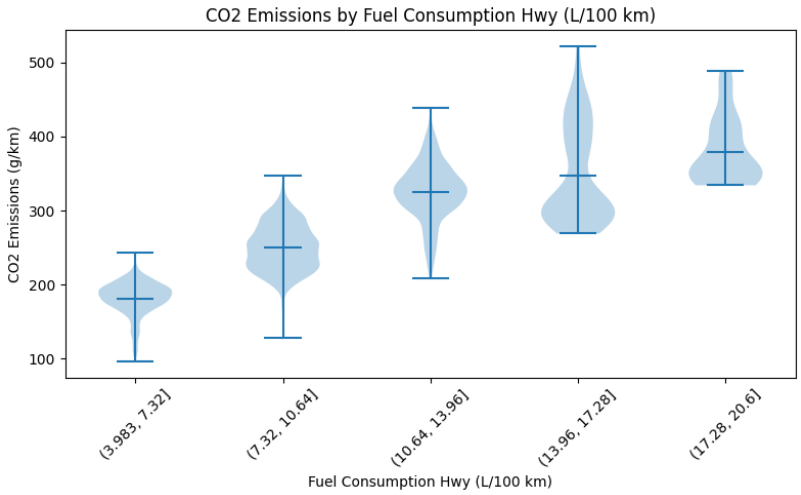


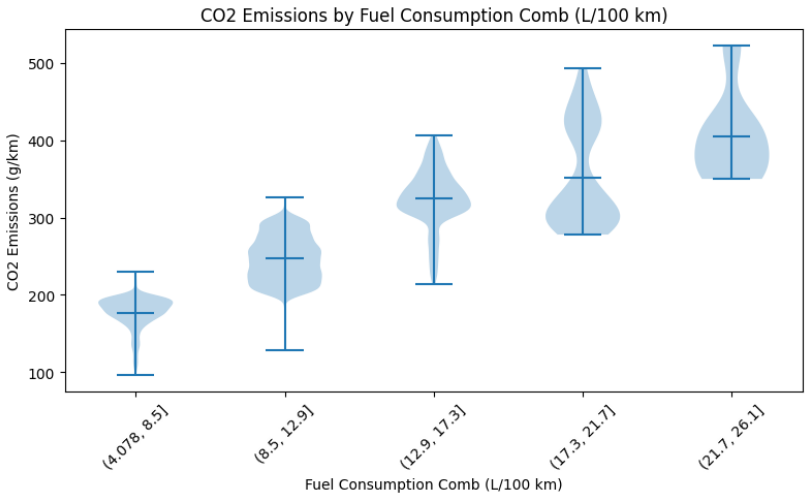
The plot shows CO2 emissions by number of cylinders, with a general upward trend as cylinder count increases. The smaller spread for 3 and 5 cylinder engines may suggest these are less common in the dataset.

The other violin plots use continuous variables, so the x-axis doesn't increment by fixed steps. To visualize the data meaningfully, the values is grouped into 5 bins. This makes the plot easier to interpret despite the floating points.

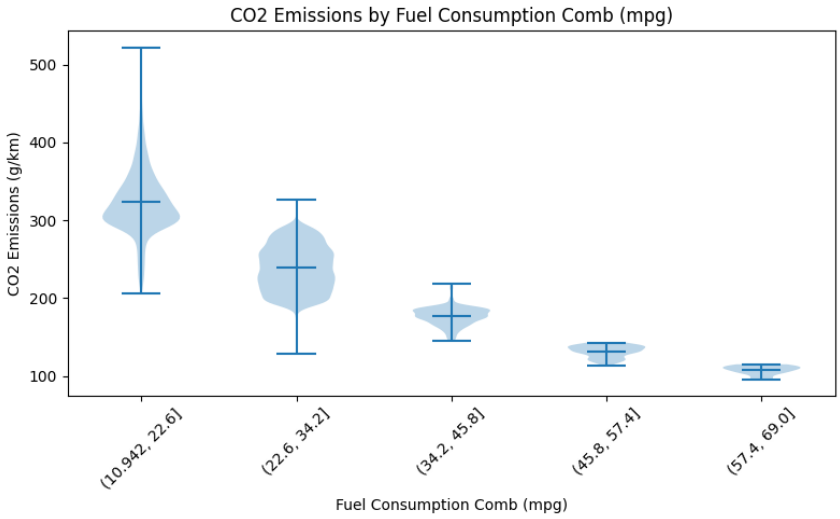






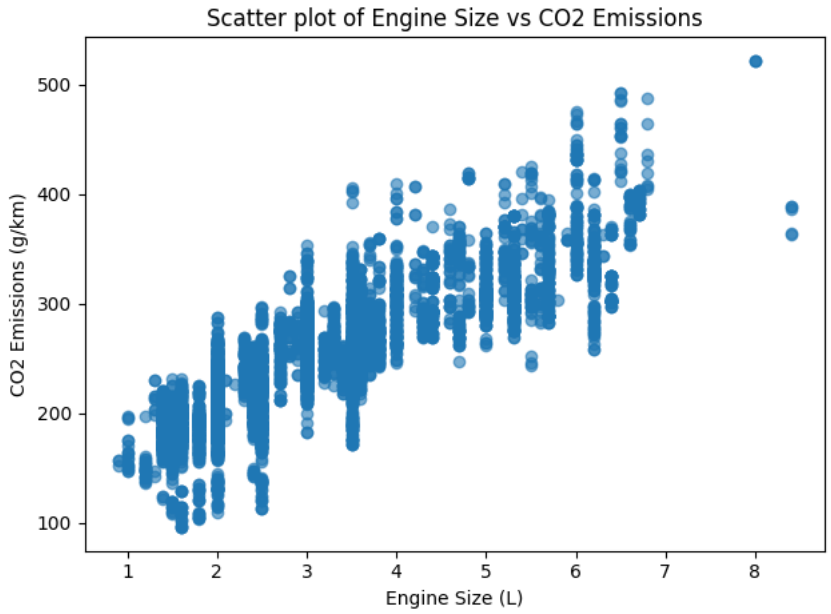


The plots of engine size, city consumption, highway consumption, and combined consumption all show a clear trend, indicating a consistent relationship with CO2 emissions.

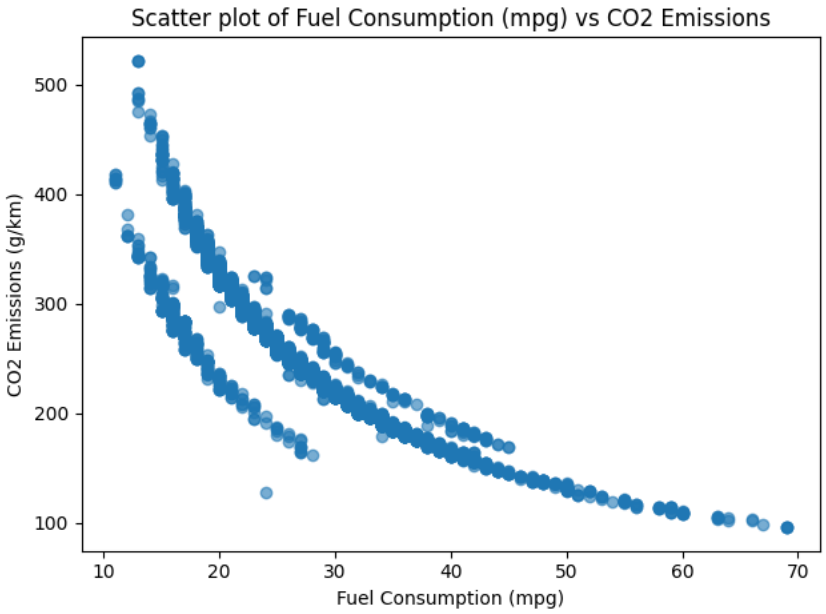


Fuel consumption (in miles per gallon) shows a clear downward trend in relation to CO2 emissions. This means that as a car becomes more fuel-efficient (higher mpg), it emits less CO2 per km. Driven.

1.2 Scatter plot

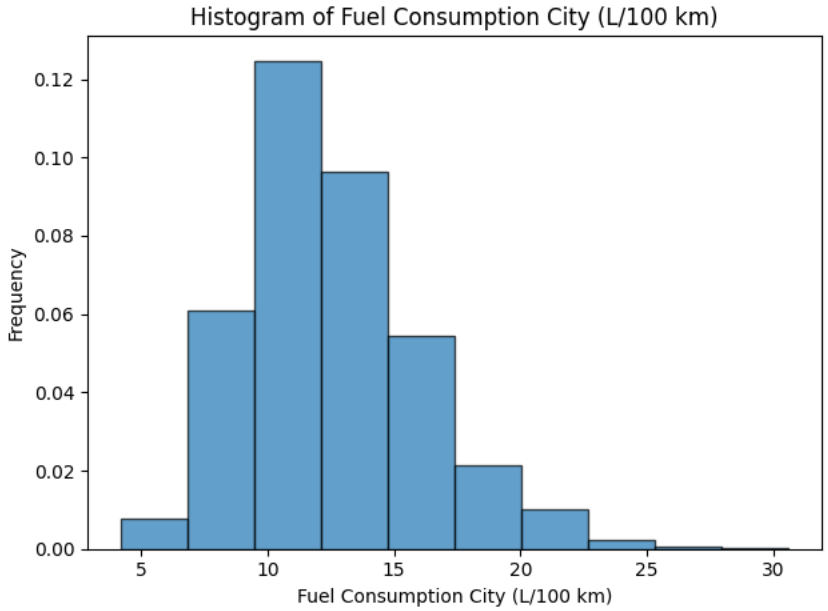


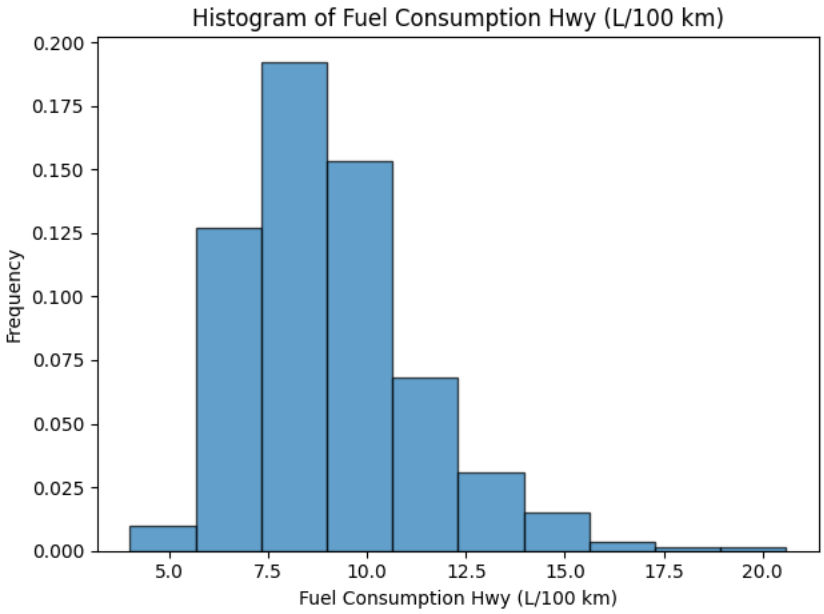
The scatter plot illustrates a positive correlation between engine size and CO2 emissions. However, there is considerable variability in emissions for any given engine size.



The scatter plot of CO2 vs fuel consumption shows a downward trend, but there appear to be three separate patterns. Applying a third variable for colouring could reveal the underlying group separation more clearly.

1.3 Histogram



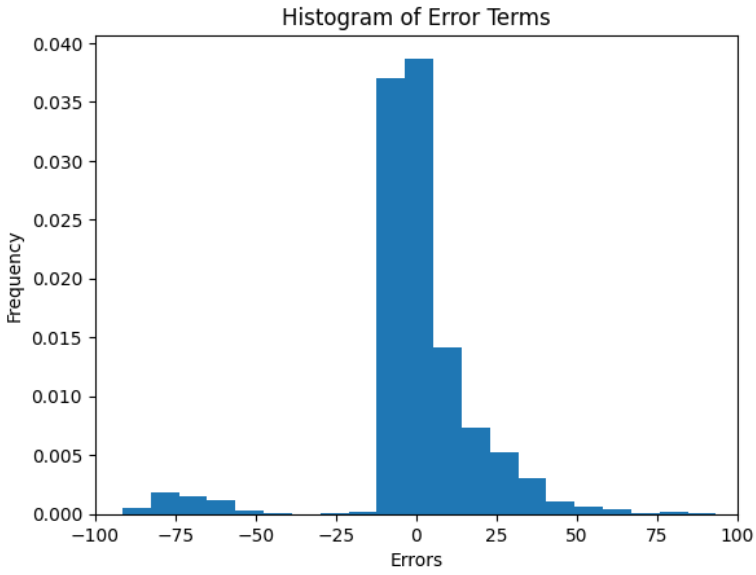


Both histograms show distributions that appear skewed to the right. Generally, we see higher fuel consumption per 100 km in the city compared to highway driving, which logically makes sense. In cities, cars typically move at lower speeds and encounter more stops and idling. On highways, vehicles maintain higher speeds and have fewer instances of traffic congestion. This explains the lower fuel consumption per 100 km on highways compared to city driving.

1.4 Estimated beta values are $\beta_0 = 945.8420$ and $\beta_1 = -211.9467$

1.5 The p-value for the intercept (const) is 0.000, and p-value for the coefficient on "Fuel Consumption Comb (mpg)" is also 0.000. Since both p-values are less than 0.10, we conclude both are statistically significant at 10% significant level.

1.6 The coefficient for Fuel Consumption Comb (mpg) is –211.95, which means that for every one-unit increase in fuel consumption (measured in mpg), the prestige score is expected to decrease by approximately 211.95 units, holding all else constant.



1.7 The error terms do not clearly follow a normal distribution. While most values are centred around zero, the histogram is skewed to the right and shows heavy tails, with several extreme values on both sides. This suggests a deviation from the OLS assumption of normally distributed errors.

2 Logistic Regression

2.1 The threshold became t = 246.0, which is the median of "CO2 Emissions (g/km)". This threshold gives a nearly even split, where:

- y = 1 (CO2 < 246.0): 3675 observations
- y = 0 (CO2 ≥ 246.0): 3710 observations

2.2 Model 1 done, look appendix.

2.3 Logistic regression of model 1:

$$\log\left(\frac{\Pr(y=1|x)}{1-\Pr(y=1|x)}\right) = \beta_0 + \beta_1 Engine\ Size + \beta_2 \log\left(Fuel\ C.Comb(mpg)\right)$$

Coefficient (for model 1)	Estimate	p-value	Significant (10%)
$\beta_0(intercept)$	-70.0645	0.000	Yes
$\beta_1(Engine\ Size)$	-1.0322	0.000	Yes
$\beta_2 \log\left(Fuel\ C.Comb(mpg)\right)$	22.2256	0.000	Yes

All coefficient estimates in model 1 are statistically significant at the 10% significance level. Since the p-values are rounded to four decimal places, each

appears as 0.000. This suggests that, given this model and dataset, Engine Size and log(Fuel Consumption Comb (mpg)) are meaningful predictors of the dependent variable.

2.4

(a) Logistic regression of model 2:

$$\log\left(\frac{\Pr(y=1|x)}{1-\Pr(y=1|x)}\right) = \beta_0 + \beta_1 Engine\ Size$$

Coefficient (for model 2)	Estimate	p-value	Significant (10%)
$\beta_0(intercept)$	8.0968	0.000	Yes
$\beta_1(Engine\ Size)$	-2.7655	0.000	Yes

(b) Logistic regression of model 3:

$$\log\left(\frac{\Pr(y=1|x)}{1-\Pr(y=1|x)}\right) = \beta_0 + \beta_1 \log\left(Fuel\ C.Comb(mpg)\right)$$

Coefficient (for model 3)	Estimate	p-value	Significant (10%)
$\beta_0(intercept)$	-86.8736	0.000	Yes
$\beta_1 \log\left(Fuel\ C.Comb(mpg)\right)$	26.4246	0.000	Yes

For model 2, the estimate $\beta_1(Engine\ Size)$ is -2.7655, and for model 3, the estimate $\beta_1 \log\left(Fuel\ C.Comb(mpg)\right)$ is 26.4246. Both coefficients from each model have a p-value of 0.000, indicating that they are statistically significant at the 10% significance level.

2.5 From a statistical point of view, Model 1 is the most robust, as it combines both Engine Size and log(Fuel Consumption Comb (mpg)) and also achieves the lowest loss (Current function value = 0.17688). The model captures more information through multiple predictors, so if the goal is highest accuracy, Model 1 is the best choice.

However, Model 3 also performs well, with only a slightly higher loss (Current function value = 0.18673), and it has the highest estimated β_1 (26.4246) among all models. This indicates a strong influence on the dependent variable. If the

goal is to predict the outcome using a single, strong predictor like log(Fuel Consumption Comb (mpg)), Model 3 is a simple and effective candidate.

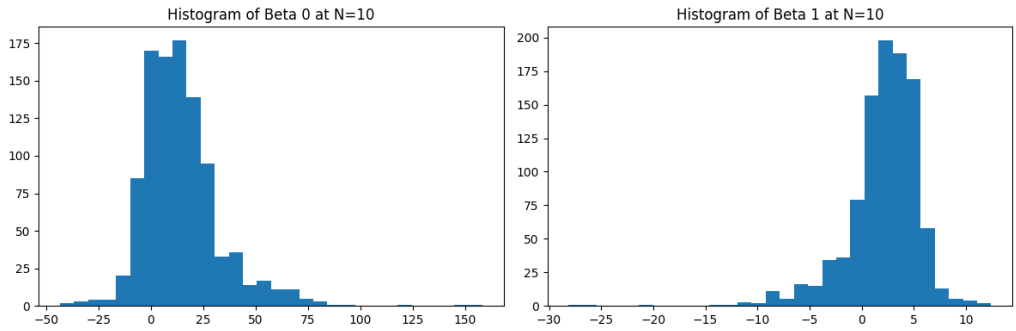
Model 2 performs the worst. Although its β_1 is statistically significant, it has the highest loss (Current function value = 0.30492) and the lowest effect size ($\beta_1 = -2.7655$), making it the least suitable among the three.

Model 1 offers the best accuracy, model 3 is simple and efficient, while model 2 is the least effective. Choosing between Model 1 or Model 3 depends on whether the priority is maximum accuracy or simplicity.

3 Uncertainty in linear models

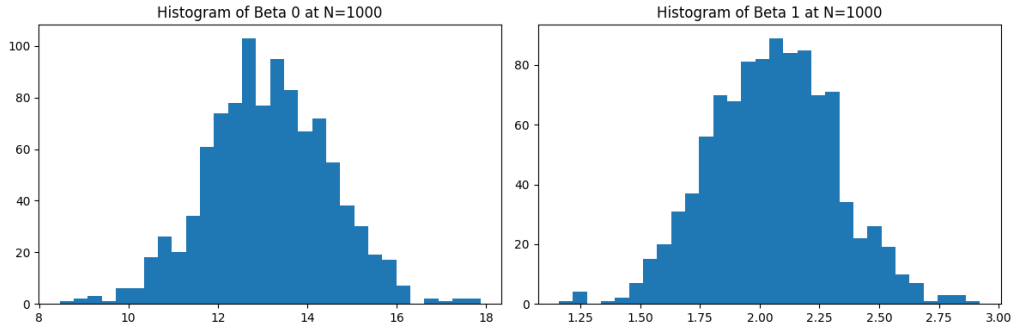
3.1, 3.2, 3.3 and 3.4 done. Look at appendix.

3.5:



Each histogram showing beta 0 and 1 at N=10, shows a distribution curve.

3.6 and 3.7:



Both sets of histograms show clearly a normal distribution curve. When N=10 the curve is more distributed around the mean, while the histogram with N=1000 have a flatter distribution. This is because generating more number results in more standard deviation from the mean.

3.8 All coefficients are positive, and the model has a high R-squared of 0.991, meaning it explains nearly all the variation in the dependent variable. All coefficients are statistically significant at the 10% level ($p < 0.1$).

3.9 The problem with the last model is multicollinearity. In the model, x_3 is created as a linear combination: $x_3 = 3x_2 + 2x_3$. This means x_1, x_2 and x_3 are no longer independent, as they rely on each other. As a result, the model cannot accurately estimate the individual effect of each variable. This is especially evident in the large standard errors for x_1 and x_2.

To fix the issue, you can either drop x_3 and keep x_1 and x_2, or use only x_3 . This will remove the multicollinearity and make the model more stable and interpretable.

4 K-fold Cross-Validation and PCA

4.1 Classification Accuracy: **0.9143**

	Predicted false	Predicted true
Actual false	53215	744
Actual true	4399	1642

4.2

Average accuracy: **0.9118**

Standard deviation: **0.0022**

F.	1	2	3	5	6	7	8	9	10
A.	0.9136	0.9111	0.9120	0.9116	0.9110	0.9105	0.9080	0.9098	0.9162

4.3

	80 Components	140 Components	190 Components
Avg. Accuracy	0.9016	0.9081	0.9128
Std. Accuracy	0.0021	0.0022	0.0017

4.4

The full feature model has the highest classification accuracy of 0.9143 using a simple 70/30 train-test split. However, since no cross-validation (CV) was performed, the result may be overfitting and not a reliable estimate of the model's generalization performance. Further testing using cross-validation is recommended.

The 10-fold CV model achieved an average accuracy of 0.9118 with a standard deviation of 0.0022, which is a more robust and reliable measure of performance, as it averages results across multiple train/test splits.

The PCA-based models were evaluated using 10-fold CV with 80, 140, and 190 components. The best result was with 190 components, achieving an accuracy of 0.9128 and a standard deviation of 0.0017. This approach balances high accuracy with improved generalization and reduced dimensionality, which lowers the risk of overfitting and may also mitigate issues such as multicollinearity. However, dimensionality reduction may cause some information loss.

The retail company should use the PCA model with 190 components and 10-fold cross-validation. It provides nearly the same accuracy as the full feature model, while also being more efficient and statistically reliable.

4.5

With a threshold of 0.1 instead of default 0.5, we get the following table:

	80 Components	140 Components	190 Components
Avg. Accuracy	0.6907	0.7437	0.7737
Std. Accuracy	0.0031	0.0028	0.0019

Lowering the threshold to 0.1, it increases false positives at the expense of true negatives, which causes the overall accuracy to collapse.

Appendix

May 29, 2025

[ ]:
```
#package import
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

#load data
emission_data = pd.read_csv("co2.csv")
```

1 Linear Regression

[ ]:
```
#1.1

emission_violin = ["Engine Size(L)", "Cylinders", "Fuel Consumption City (L/100␣
 ↪km)", "Fuel Consumption Hwy (L/100 km)", "Fuel Consumption Comb (L/100 km)",␣
 ↪"Fuel Consumption Comb (mpg)"]

categories = emission_data.groupby("Vehicle Class")[emission_violin + ["CO2␣
 ↪Emissions(g/km)"]]

def violin(emission_violin, data):
    for col in emission_violin:
        if data[col].nunique() > 10:
            data["group"] = pd.cut(data[col], bins=5)
        else:
            data["group"] = data[col]

        categories = data.groupby("group", observed=False)[["CO2 Emissions(g/
 ↪km)"]]

        violin_data = []
        for group in categories.groups:
            violin_data.append(categories.get_group(group)["CO2 Emissions(g/
 ↪km)"])

        if len(violin_data) < 2:
```

```
            continue

        plt.figure(figsize=(8, 5))
        plt.violinplot(violin_data, showmeans=True)
        plt.xticks(ticks=range(1, len(categories.groups)+1), labels=[str(g) for␣
 ↪g in categories.groups], rotation=45)
        plt.xlabel(col)
        plt.ylabel("CO2 Emissions (g/km)")
        plt.title(f"CO2 Emissions by {col}")
        plt.tight_layout()
        plt.show()

violin(emission_violin, emission_data)
```

[ ]:
```
#1.2

def scatter(x, y, x_label, y_label, title):
    plt.scatter(x, y, alpha=0.6)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title)
    plt.tight_layout()
    plt.show()

#plot 1
scatter(
    emission_data["Engine Size(L)"],
    emission_data["CO2 Emissions(g/km)"],
    "Engine Size (L)",
    "CO2 Emissions (g/km)",
    "Scatter plot of Engine Size vs CO2 Emissions"
)

#plot 2
scatter(
    emission_data["Fuel Consumption Comb (mpg)"],
    emission_data["CO2 Emissions(g/km)"],
    "Fuel Consumption (mpg)",
    "CO2 Emissions (g/km)",
    "Scatter plot of Fuel Consumption (mpg) vs CO2 Emissions"
)
```

[ ]:
```
#1.3

fuel_hist = ["Fuel Consumption City (L/100 km)", "Fuel Consumption Hwy (L/100␣
 ↪km)"]
```

```
def hist(fuel_hist, data):
    for i in fuel_hist:
        plt.hist(data[i], density=True, edgecolor="black", alpha=0.7)
        plt.xlabel(i)
        plt.ylabel("Frequency")
        plt.title("Histogram of " + i)
        plt.tight_layout()
        plt.show()

hist(fuel_hist, emission_data)
```

[ ]:
```
#1.4, 1.5, 1.6

X = np.log(emission_data[["Fuel Consumption Comb (mpg)"]])
Y = emission_data["CO2 Emissions(g/km)"]

X = sm.add_constant(X)

model = sm.OLS(Y, X).fit()

beta_0 = model.params.iloc[0]
beta_1 = model.params.iloc[1]
pval_0 = model.pvalues.iloc[0]
pval_1 = model.pvalues.iloc[1]

#print("beta_0 (intercept) = " + str(beta_0))
#print("beta_1 (log mpg) = " + str(beta_1))
#print("p-value for beta_0 =", pval_0)
#print("p-value for beta_1 =", pval_1)
```

[ ]:
```
#1.7

errors = Y - model.fittedvalues

plt.hist(errors, density=True)
plt.xlabel("Errors")
plt.ylabel("Frequency")
plt.title("Histogram of Error Terms")
plt.show()
```

2 Logistic Regression

[ ]:
```
emission_data_log = emission_data.copy()
#emission_data_log.head()
```

[ ]:
```
#2.1

t = emission_data_log["CO2 Emissions(g/km)"].median()
emission_data_log["y"] = (emission_data_log["CO2 Emissions(g/km)"] < t).
 ↪astype(int)
counts = emission_data_log["y"].value_counts()
#print("Threshold t selected:", t)
#print("Count of y = 1 (CO2 < t):", counts[1])
#print("Count of y = 0 (CO2 >= t):", counts[0])
```

[ ]:
```
#2.2 and 2.3, model 1
X = emission_data_log[["Engine Size(L)"]].copy()
X["log_mpg"] = np.log(emission_data_log["Fuel Consumption Comb (mpg)"])

X = sm.add_constant(X)

y = emission_data_log["y"]

logit_model = sm.Logit(y, X)
result = logit_model.fit()

beta_0_log = result.params.iloc[0]
beta_1_log = result.params.iloc[1]
beta_2_log = result.params.iloc[2]

pval_0_log = result.pvalues.iloc[0]
pval_1_log = result.pvalues.iloc[1]
pval_2_log = result.pvalues.iloc[2]

#print("beta_0 (intercept) =", beta_0_log)
#print("beta_2 (Engine Size) =", beta_1_log)
#print("beta_3 (Fuel Consumption Comb) =", beta_2_log)

#print("p-value for beta_0 =", pval_0_log)
#print("p-value for beta_1 =", pval_1_log)
#print("p-value for beta_2 =", pval_2_log)
```

[ ]:
```
#2.4a model 2
X_a = sm.add_constant(emission_data_log[["Engine Size(L)"]])
y = emission_data_log["y"]

logit_a = sm.Logit(y, X_a)
result_a = logit_a.fit()

beta_0_a = result_a.params.iloc[0]
beta_1_a = result_a.params.iloc[1]
```

```
pval_0_a = result_a.pvalues.iloc[0]
pval_1_a = result_a.pvalues.iloc[1]

#print("beta_0 (intercept) =", beta_0_a)
#print("beta_2 (Engine Size) =", beta_1_a)

#print("p-value for beta_0 =", pval_0_a)
#print("p-value for beta_1 =", pval_1_a)
```

[ ]:
```
#2.4b model 3
X_b = np.log(emission_data_log[["Fuel Consumption Comb (mpg)"]])
X_b = sm.add_constant(X_b)

logit_b = sm.Logit(y, X_b)
result_b = logit_b.fit()


beta_0_b = result_b.params.iloc[0]
beta_1_b = result_b.params.iloc[1]

pval_0_b = result_b.pvalues.iloc[0]
pval_1_b = result_b.pvalues.iloc[1]

#print("beta_0 (intercept) =", beta_0_b)
#print("beta_2 (Engine Size) =", beta_1_b)

#print("p-value for beta_0 =", pval_0_b)
#print("p-value for beta_1 =", pval_1_b)
```

3 Uncertainty in linear models

[ ]:
```
#3.1
np.random.seed(102030)

cov = np.array([[1, 0.4, 0],
                [0.4, 1.5, 0],
                [0, 0, 2]])

mu = np.array([5, 2, 1])

N = 10


x1, x2, x3 = np.random.multivariate_normal(mu, cov, N).T

#3.2
```

```
y = 7 + x1 + 3*x2 + 5*x3 + np.random.normal(loc=0, scale=1, size=N)

#3.3
B = 1000
b_data = []

for i in range(B):
    indices = np.random.choice(N, size=N, replace=True)
    x_b = x1[indices]
    y_b = y[indices]
    b_data.append((x_b, y_b))

#3.4
beta_b_data = []
for x_b, y_b in b_data:
    X_b = np.vstack((np.ones(N), x_b)).T
    beta_b = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y_b
    beta_b_data.append(beta_b)

beta_bootstraps = np.array(beta_b_data)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))

for i in range(2):
    ax[i].hist(beta_bootstraps[:, i], bins=30)
    ax[i].set_title(f"Histogram of Beta {i} at N={N}")

plt.tight_layout()
plt.show()
```

[ ]:
```
#3.6
np.random.seed(102030)

cov = np.array([[1, 0.4, 0],
                [0.4, 1.5, 0],
                [0, 0, 2]])

mu = np.array([5, 2, 1])

N = 1000


x1, x2, x3 = np.random.multivariate_normal(mu, cov, N).T
y = 7 + x1 + 3*x2 + 5*x3 + np.random.normal(loc=0, scale=1, size=N)

#3.7
B = 1000
```

```
b_data = []

for i in range(B):
    indices = np.random.choice(N, size=N, replace=True)
    x_b = x1[indices]
    y_b = y[indices]
    b_data.append((x_b, y_b))

beta_b_data = []
for x_b, y_b in b_data:
    X_b = np.vstack((np.ones(N), x_b)).T
    beta_b = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y_b
    beta_b_data.append(beta_b)

beta_bootstraps = np.array(beta_b_data)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 4))

for i in range(2):
    ax[i].hist(beta_bootstraps[:, i], bins=30)
    ax[i].set_title(f"Histogram of Beta {i} at N={N}")

plt.tight_layout()
plt.show()
```

[ ]:
```
#3.8
np.random.seed(102030)

def true_dgp(x1, x2, x3):
    return 7 + x1 + 3*x2 + 5*x3

mu = np.array([5, 2, 1])
cov = np.array([[1, 0.4, 0],
                [0.4, 1.5, 0],
                [0, 0, 2]])
N = 10
x1, x2, x3 = np.random.multivariate_normal(mu, cov, N).T

y = true_dgp(x1, x2, x3) + np.random.normal(loc=0, scale=1, size=N)

X = np.column_stack((x1, x2, x3))
X = sm.add_constant(X)

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

4 K-fold Cross-Validation and PCA

[ ]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, KFold, cross_val_score,␣
 ↪cross_val_predict
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
```

[ ]:
```
data_logistic = pd.read_csv("data_logistic.csv")
#data_logistic.head()
```

[ ]:
```
#4.1
X = data_logistic.drop(["target", "ID_code"], axis=1)
y = data_logistic["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=102030)

X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)


logit_model = sm.Logit(y_train, X_train)
result = logit_model.fit()

y_pred_prob = result.predict(X_test)
y_pred = (y_pred_prob >= 0.5).astype(int)

cm = confusion_matrix(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)

#print(cm)
#print(acc)
```

[ ]:
```
#4.2
model = LogisticRegression(max_iter=100)

kf = KFold(n_splits=10, shuffle=True, random_state=102030)

scores = cross_val_score(model, X, y, cv=kf, scoring="accuracy")

mean_as = np.mean(scores)
std_as = np.std(scores)
```

[ ]:
```
#print(mean_as)
#print(std_as)
#print(scores)
```

[ ]:
```
results_df = pd.DataFrame({
    "Fold": range(1, 11),
    "Accuracy": scores.round(4)
})

mean_std_row = pd.DataFrame({
    "Fold": ["Average", "Standard Deviation"],
    "Accuracy": [mean_as.round(4), std_as.round(4)]
})

results_df = pd.concat([results_df, mean_std_row], ignore_index=True)
results_df
```

[ ]:
```
kf10 = KFold(n_splits=10, shuffle=True, random_state=102030)
n_components_list = [80, 140, 190]
```

[ ]:
```
#4.3

results = []
for n in n_components_list:
    pipeline =␣
 ↪make_pipeline(PCA(n_components=n),LogisticRegression(max_iter=100))
    scores = cross_val_score(pipeline, X, y, cv=kf10, scoring="accuracy")

    results.append({
    "Components": n,
        "Avg. Accuracy": np.mean(scores),
        "Std. Accuracy": np.std(scores)
    })

df_pca = pd.DataFrame(results)
```

[ ]:
```
#print(df_pca.round(4))
```

[ ]:
```
#4.5

threshold = 0.10

results = []
for n in n_components_list:
    pipeline =␣
 ↪make_pipeline(PCA(n_components=n),LogisticRegression(max_iter=100))
    probas = cross_val_predict(pipeline, X, y, cv=kf10, method="predict_proba")
```

```
    y_pred = (probas[:, 1] >= threshold).astype(int)
    y_true = y.values

    accs = []
    for train_idx, test_idx in kf10.split(X):
        accs.append(
            accuracy_score(
                y_true[test_idx],
                y_pred[test_idx]
            )
        )

    results.append({
        "Components": n,
        "Avg. Accuracy": np.mean(accs),
        "Std. Accuracy": np.std(accs)
    })

df_pca_threshold = pd.DataFrame(results)
```

[ ]:
```
#print(df_pca_threshold.round(4))
```