In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
```

Question 1

In [2]:
```
weather = pd.read_csv('weatherAUS.csv')
weather.head()
```

Out[2]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pre
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	22.0	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	25.0	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	30.0	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	16.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	33.0	

5 rows × 24 columns

In [3]:
```
weather.describe()
```

Out[3]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity
count	141556.000000	141871.000000	140787.000000	81350.000000	74377.000000	132923.000000	140845.000000	139563.000000	140419.00
mean	12.186400	23.226784	2.349974	5.469824	7.624853	39.984292	14.001988	18.637576	68.84
std	6.403283	7.117618	8.465173	4.188537	3.781525	13.588801	8.893337	8.803345	19.05
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.00
25%	7.600000	17.900000	0.000000	2.600000	4.900000	31.000000	7.000000	13.000000	57.00
50%	12.000000	22.600000	0.000000	4.800000	8.500000	39.000000	13.000000	19.000000	70.00
75%	16.800000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	83.00
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	100.00

In [4]:

```
weather = weather[['MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
                   'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Temp9am', 'Temp3pm', 'RainToday', 'RainTomorrow']]
weather = weather.dropna()
```

In [5]:

```
weather = weather.replace(to_replace= ['Yes','No'], value = [1,0])
weather.head()
```

Out[5]:

	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm
0	13.4	22.9	0.6	44.0	20.0	24.0	71.0	22.0	1007.7	1007.1
1	7.4	25.1	0.0	44.0	4.0	22.0	44.0	25.0	1010.6	1007.8
2	12.9	25.7	0.0	46.0	19.0	26.0	38.0	30.0	1007.6	1008.7
3	9.2	28.0	0.0	24.0	11.0	9.0	45.0	16.0	1017.6	1012.8
4	17.5	32.3	1.0	41.0	7.0	20.0	82.0	33.0	1010.8	1006.0

Question 2

In [6]:

```
X_train, X_test, y_train, y_test = train_test_split(weather.drop('RainTomorrow', axis = 1),
    weather['RainTomorrow'], test_size=0.2, random_state=100)
```

```
model = sm.Logit(y_train,sm.add_constant(X_train)).fit()
model.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.352093
         Iterations 7
```

Out[6]:

Logit Regression Results

Dep. Variable:	RainTomorrow	**No. Observations:**	95672
Model:	Logit	**Df Residuals:**	95658
Method:	MLE	**Df Model:**	13
Date:	Mon, 22 Apr 2024	**Pseudo R-squ.:**	0.3323
Time:	20:16:32	**Log-Likelihood:**	-33685.
converged:	True	**LL-Null:**	-50447.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	59.1006	1.744	33.896	0.000	55.683	62.518
MinTemp	0.0217	0.005	4.225	0.000	0.012	0.032
MaxTemp	0.0018	0.009	0.213	0.831	-0.015	0.019
Rainfall	0.0080	0.001	6.251	0.000	0.005	0.011
WindGustSpeed	0.0638	0.001	54.936	0.000	0.062	0.066
WindSpeed9am	-0.0093	0.002	-6.128	0.000	-0.012	-0.006
WindSpeed3pm	-0.0422	0.002	-26.869	0.000	-0.045	-0.039
Humidity9am	0.0053	0.001	4.715	0.000	0.003	0.007
Humidity3pm	0.0638	0.001	54.115	0.000	0.062	0.066
Pressure9am	0.1667	0.006	28.852	0.000	0.155	0.178
Pressure3pm	-0.2318	0.006	-39.770	0.000	-0.243	-0.220
Temp9am	0.0050	0.008	0.635	0.525	-0.010	0.021
Temp3pm	-0.0473	0.010	-4.895	0.000	-0.066	-0.028
RainToday	0.5235	0.026	19.954	0.000	0.472	0.575

In [7]:

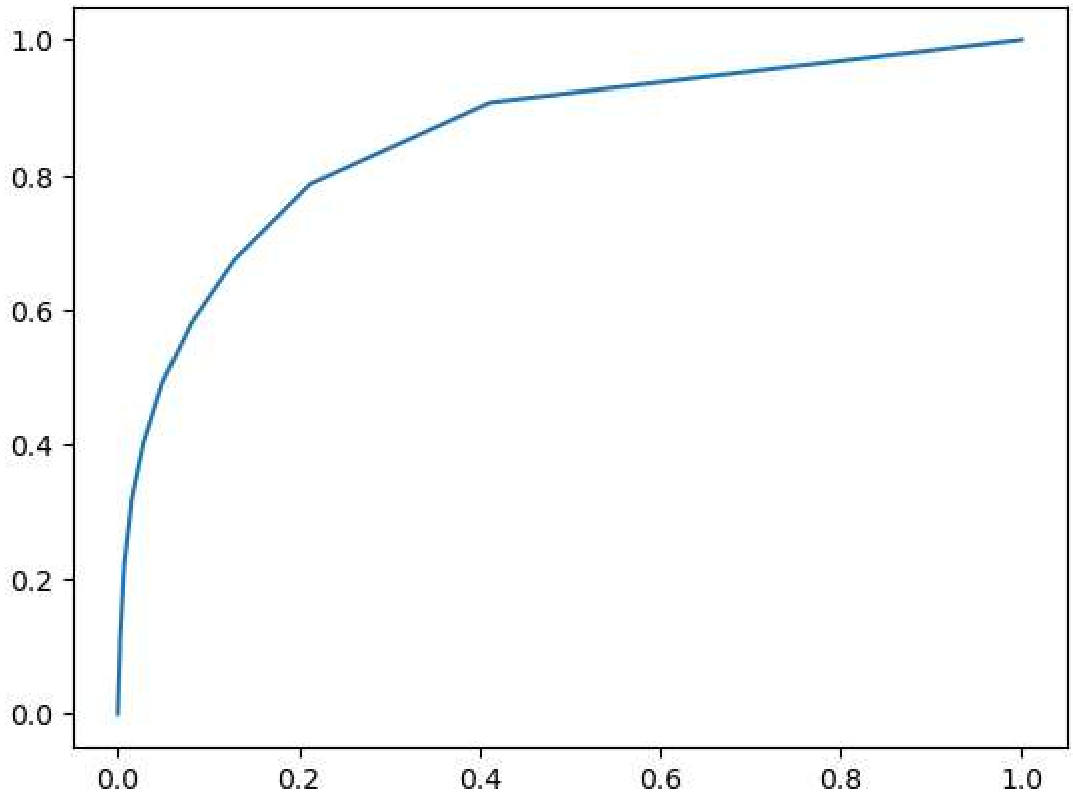
```
prob = model.predict(sm.add_constant(X_test))
predictions = (prob >= 0.5).astype(int)
```

In [8]:
```
tp = np.sum((predictions == 1) & (y_test == 1))
tn = np.sum((predictions == 0) & (y_test == 0))
fp = np.sum((predictions == 1) & (y_test == 0))
fn = np.sum((predictions == 0) & (y_test == 1))

print("True positive: " + str(tp) + ", True negative: " + str(tn) + ", False positive: " + str(fp) + ", False negative:
```

```
True positive: 2513, True negative: 17882, False positive: 926, False negative: 2597
```

In [9]:
```
tpr = []
fpr = []
vals = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

for i in vals:
    predictions_temp = (prob >= i).astype(int)
    tp_temp = np.sum((predictions_temp == 1) & (y_test == 1))
    tn_temp = np.sum((predictions_temp == 0) & (y_test == 0))
    fp_temp = np.sum((predictions_temp == 1) & (y_test == 0))
    fn_temp = np.sum((predictions_temp == 0) & (y_test == 1))
    fpr.append((fp_temp / (fp_temp + tn_temp)))
    tpr.append((tp_temp / (tp_temp + fn_temp)))
plt.plot(fpr,tpr)
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x1e732f058d0>]
```



A perfect classification model would have a 90 degree angle, going straight up from 0,0 and then straight right to 1,0. For a completely random model, it would likely look like a straight line from 0,0 to 1,0.

Question 3

In [10]:
```
n = 1000 # number of datapoints
p = 100 # number of predictors
# x is a random nxp matrix of 0s and 1s
# Each entry is 1 with probability 0.1
x = 1.0*(np.random.rand(n,p) <= 0.1)
y = 15 + np.random.randn(n)
```

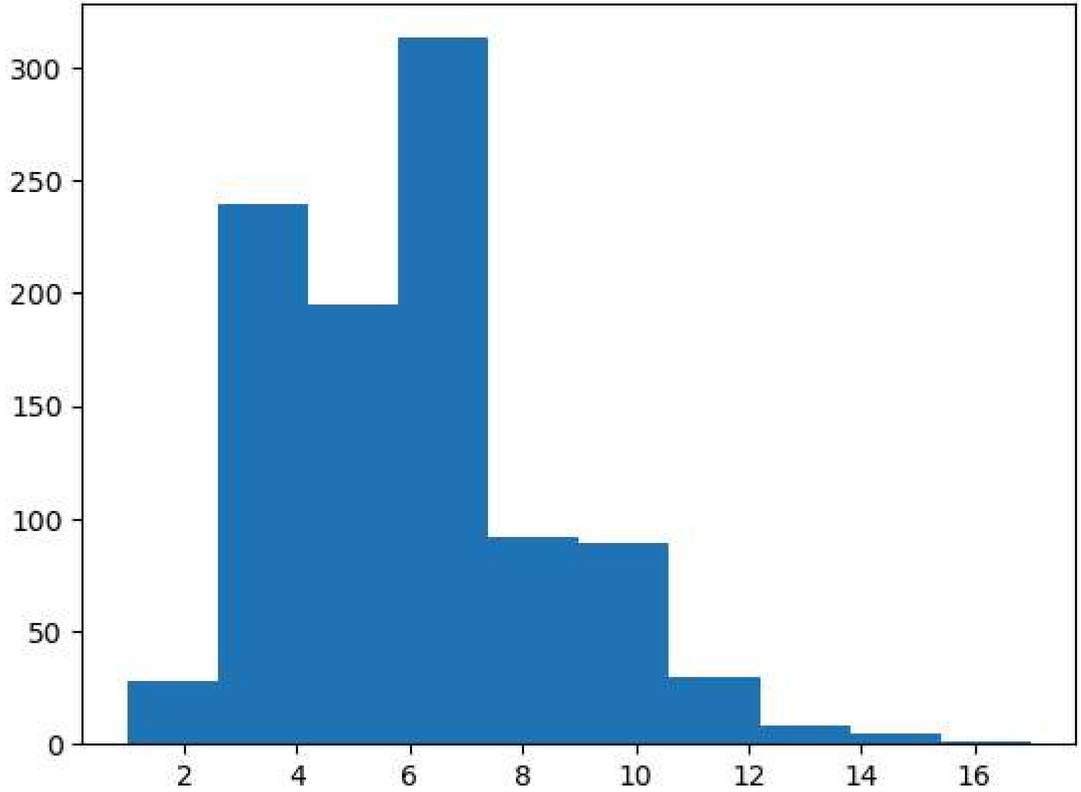
Effectively 0 of the B_j have a non-zero value.

In [11]:
```
b1_count = 0
p_val_count = []
for i in range(1000):
    n = 1000
    p = 100
    x = 1.0*(np.random.rand(n,p) <= 0.1)
    y = 15 + np.random.randn(n)
    mod = sm.OLS(y, sm.add_constant(x)).fit()
    if mod.pvalues[1] < 0.05:
        b1_count += 1
    p_val_count.append(sum(p < 0.05 for p in mod.pvalues))
```

In [12]:
```
print("Fraction of b1s less than 0.05: " + str(b1_count/1000))
plt.hist(p_val_count)
```

```
Fraction of b1s less than 0.05: 0.052
```

Out[12]:
```
(array([ 28., 239., 195., 313.,  92.,  89.,  30.,   8.,   5.,   1.]),
 array([ 1. ,  2.6,  4.2,  5.8,  7.4,  9. , 10.6, 12.2, 13.8, 15.4, 17. ]),
 <BarContainer object of 10 artists>)
```



Question 4

In [13]:
```
def logistic(x):
    return 1./(1+np.exp(-x))

n = 10000
df = pd.DataFrame()

df['age'] = np.random.normal(46.1, 16.6, n)
df['bfp'] = np.random.normal(23.3,7.0,n) # body fat percentage
blood_pressure_before_medicine = 112.15 + 0.89*df['bfp']-0.003*df['age']

prob_seek_treatment = logistic(.5*(blood_pressure_before_medicine-130))
df['is_treated'] = (np.random.rand(n)<prob_seek_treatment).astype(int)
df['blood_pressure'] = blood_pressure_before_medicine-df['is_treated']*10
```

```
df.head()
```

Out[13]:

	age	bfp	is_treated	blood_pressure
0	52.744070	25.944143	1	125.082055
1	37.835560	28.911093	1	127.767366
2	26.934715	20.092895	0	129.951872
3	28.756625	19.463991	1	119.386682
4	57.823807	27.040942	0	136.042967

In [14]:

```
model = sm.OLS(df['blood_pressure'], sm.add_constant(df['is_treated'])).fit()
model.summary()
```

Out[14]:

OLS Regression Results

Dep. Variable:	blood_pressure	**R-squared:**	0.013
Model:	OLS	**Adj. R-squared:**	0.013
Method:	Least Squares	**F-statistic:**	135.4
Date:	Mon, 22 Apr 2024	**Prob (F-statistic):**	4.27e-31
Time:	20:16:50	**Log-Likelihood:**	-29470.
No. Observations:	10000	**AIC:**	5.894e+04
Df Residuals:	9998	**BIC:**	5.896e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	127.0524	0.077	1647.005	0.000	126.901	127.204
is_treated	-1.1195	0.096	-11.636	0.000	-1.308	-0.931

Omnibus:	57.807	**Durbin-Watson:**	1.987
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	59.174
Skew:	0.177	**Prob(JB):**	1.41e-13
Kurtosis:	3.132	**Cond. No.**	3.11

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The estimated effect of being on treatment is a decrease in blood pressure of 1.217.

In [15]:

```
treated_avg = np.mean(df[df['is_treated'] ==1]['blood_pressure'])
untreated_avg = np.mean(df[df['is_treated'] ==0]['blood_pressure'])
treated_avg - untreated_avg
```

Out[15]:
```
-1.1195187112161165
```

In [16]:
```
df['blood_pressure_before_medicine'] = 112.15 + 0.89*df['bfp']-0.003*df['age']
treated_avg_b4 = np.mean(df[df['is_treated'] ==1]['blood_pressure_before_medicine'])
untreated_avg_b4 = np.mean(df[df['is_treated'] ==0]['blood_pressure_before_medicine'])

print("Avg blood pressure before medication for untreated group: " + str(untreated_avg_b4))
print("Avg blood pressure before medication for treated group: " + str(treated_avg_b4))
print("Difference between untreated and treated: " + str(untreated_avg_b4-treated_avg_b4))
print("The blood pressure before medicine was on average higher for the group that got treatment.")
```

```
Avg blood pressure before medication for untreated group: 127.05240623262512
Avg blood pressure before medication for treated group: 135.93288752140901
Difference between untreated and treated: -8.880481288783898
The blood pressure before medicine was on average higher for the group that got treatment.
```

Based on the code generating the data, there is a decrease of 10 in blood pressure for the group that received treatment.

Question 5

a) estimated treatment effect (-1.217) = avg_blood_pressure_before_medication_treatment(135.859) - avg_blood_pressure_before_medication_no_treatment(127.076) + causal_effect_of_treatment(-10)

b) The causal effect is different because the probability of a person seeking treatment is dependent on their blood pressure before treatment. Those who have higher blood pressure are more likely to seek treatment, so the estimated treatment effect from the regression is lower compared to looking at how the data was formed.

In [17]:
```
model = sm.OLS(df['blood_pressure'], sm.add_constant(df[['is_treated','blood_pressure_before_medicine']])).fit()
model.summary()
```

Out[17]:

OLS Regression Results

Dep. Variable:	blood_pressure	**R-squared:**	1.000
Model:	OLS	**Adj. R-squared:**	1.000
Method:	Least Squares	**F-statistic:**	3.866e+30
Date:	Mon, 22 Apr 2024	**Prob (F-statistic):**	0.00
Time:	20:16:50	**Log-Likelihood:**	2.8003e+05
No. Observations:	10000	**AIC:**	-5.600e+05
Df Residuals:	9997	**BIC:**	-5.600e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.14e-13	4.61e-14	11.154	0.000	4.24e-13	6.04e-13
is_treated	-10.0000	4.74e-15	-2.11e+15	0.000	-10.000	-10.000
blood_pressure_before_medicine	1.0000	3.62e-16	2.76e+15	0.000	1.000	1.000

Omnibus:	48234.906	**Durbin-Watson:**	1.786
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1695.307
Skew:	-0.597	**Prob(JB):**	0.00
Kurtosis:	1.374	**Cond. No.**	3.68e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.68e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Question 6

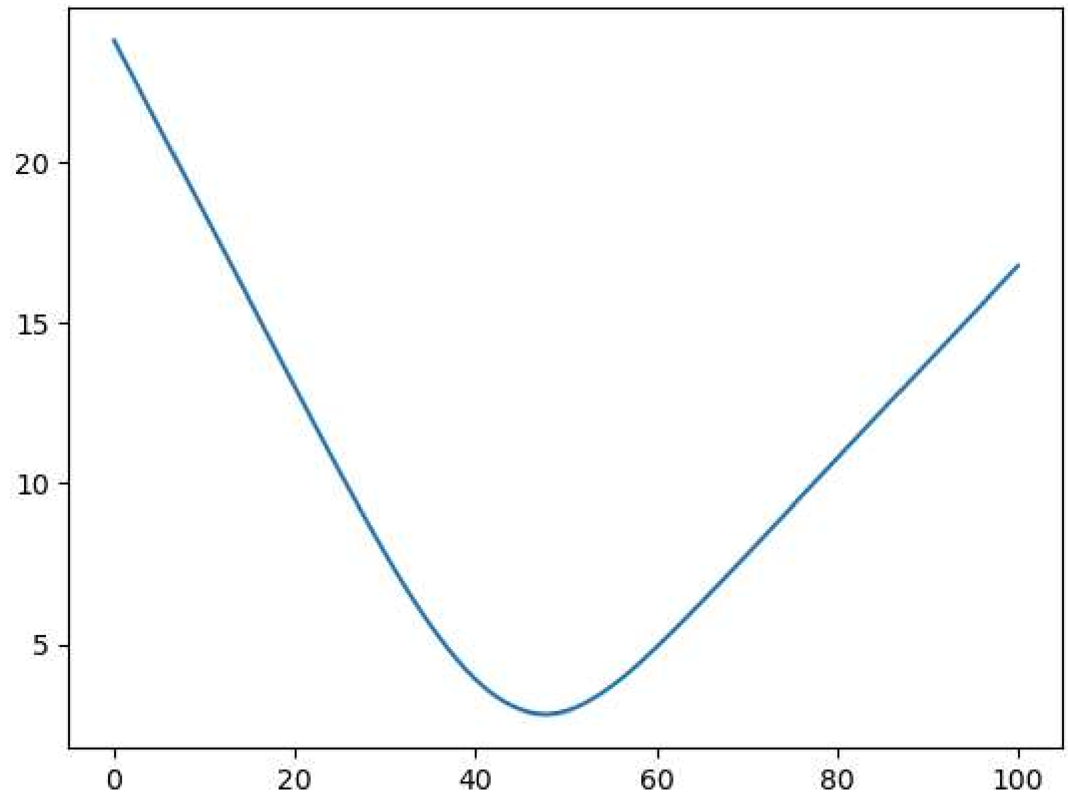
In [18]:
```
bagels = pd.read_csv("bagels.csv")
bagels.head()
```

Out[18]:

	day	bagel_demand
0	0	44.0
1	1	34.0
2	2	44.0
3	3	50.0
4	4	52.0

In [19]:
```
Q_vals = range(101)
demand = bagels['bagel_demand']
under = 1.08 - .54
over = .54-.24

def avg_cost(q):
    cost_o = over * (q-demand).clip(lower=0)
    cost_u = under * (demand - q).clip(lower=0)
    return np.mean(cost_o +cost_u)

cost_vals = [avg_cost(q) for q in Q_vals]
plt.plot(Q_vals, cost_vals)
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x1e738c9da10>]
```



In [20]:
```
min_cost = float(10000000)
optimal_Q = None

for q in range(100):
    cost = avg_cost(q)
    if cost < min_cost:
        min_cost = cost
        optimal_Q = q

print("Optimal number of bagels to order:", optimal_Q)
```

```
Optimal number of bagels to order: 48
```