# MFE 431 - Data Analytics and Machine Learning - PS3


- Simon Geller
- Aliaksei Kanstantsinau
- Gabriel de La Noue
- Adrien Flambard
- Marc Khamis


In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

Use the LendingClub_LoanStats3a_v12.csv dataset available at BruinLearn (Week 3) for this exercise. The data is downloaded from Lending Club's website and there is additional information about the data there although they now require that you sign up for a user profile.

a. We will use the column "loan_status" as the indicator for whether the loan was paid or there was a default.
(i) Drop all rows where "loan_status" is not equal to either "Fully Paid" or "Charged Off." Define the new variable Default as 1 (or TRUE) if "loan_status" is equal to "Charged Off", and 0 (or FALSE) otherwise.

In [2]:
```
lcData = pd.read_csv('../Downloads/LendingClub_LoanStats3a_v12.csv')
statuses = ['Fully Paid', 'Charged Off']
status_map = {'Fully Paid': 0, 'Charged Off': 1}
grade_map = {'A': 7, 'B':6, 'C':5, 'D':4, 'E':3, 'F':2, 'G':1}
lcData = lcData[lcData['loan_status'].isin(statuses)]
lcData['loan_status'] = lcData['loan_status'].map(status_map)
lcData['grade'] = lcData['grade'].map(grade_map)

terms = lcData['term'].unique()
lcData['is_36'] = (lcData['term']==terms[0]).astype(int)
lcData['is_60'] = (lcData['term']==terms[1]).astype(int)
print(lcData[['term', 'is_36', 'is_60']])
```

```
              term  is_36  is_60
0        36 months      1      0
1        60 months      0      1
2        36 months      1      0
3        36 months      1      0
5        36 months      1      0
...            ...    ...    ...
39781    36 months      1      0
39782    36 months      1      0
39783    36 months      1      0
39784    36 months      1      0
39785    36 months      1      0

[39412 rows x 3 columns]
```

```
/var/folders/vv/3nnd1g4506z6vdqnf44fkr2c0000gn/T/ipykernel_27433/2129906761.
py:1: DtypeWarning: Columns (21,24,29,31) have mixed types. Specify dtype op
tion on import or set low_memory=False.
  lcData = pd.read_csv('../Downloads/LendingClub_LoanStats3a_v12.csv')
```

(ii) Report the average default rate in the sample (number of defaults divided by total number of loans)

In [3]:
```python
avg_rate = lcData['loan_status'].mean()
print(f"Average Default Rate is {avg_rate}")
```

```
Average Default Rate is 0.14353496397036436
```

b. LendingClub gives a "grade" to each borrower, designed as a score of each borrowers creditworthiness. The best grade is "A", the worst grade is "G".

(i) Run a logistic regression of the Default variable on the grade. Report and explain the regression output. I.e., what is the interpretation of the coefficients? Do the numbers 'make sense'.

In [4]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import statsmodels.api as sm

X = lcData[['grade']]
X1 = sm.add_constant(X)
y = lcData[['loan_status']]
model = LogisticRegression()
model.fit(X, y)

y_pred = model.predict(X)
accuracy = accuracy_score(y, y_pred)

accuracy = accuracy_score(y, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y, y_pred))
```

```
OLS_model = sm.OLS(y, X1).fit()
print(OLS_model.summary())
```

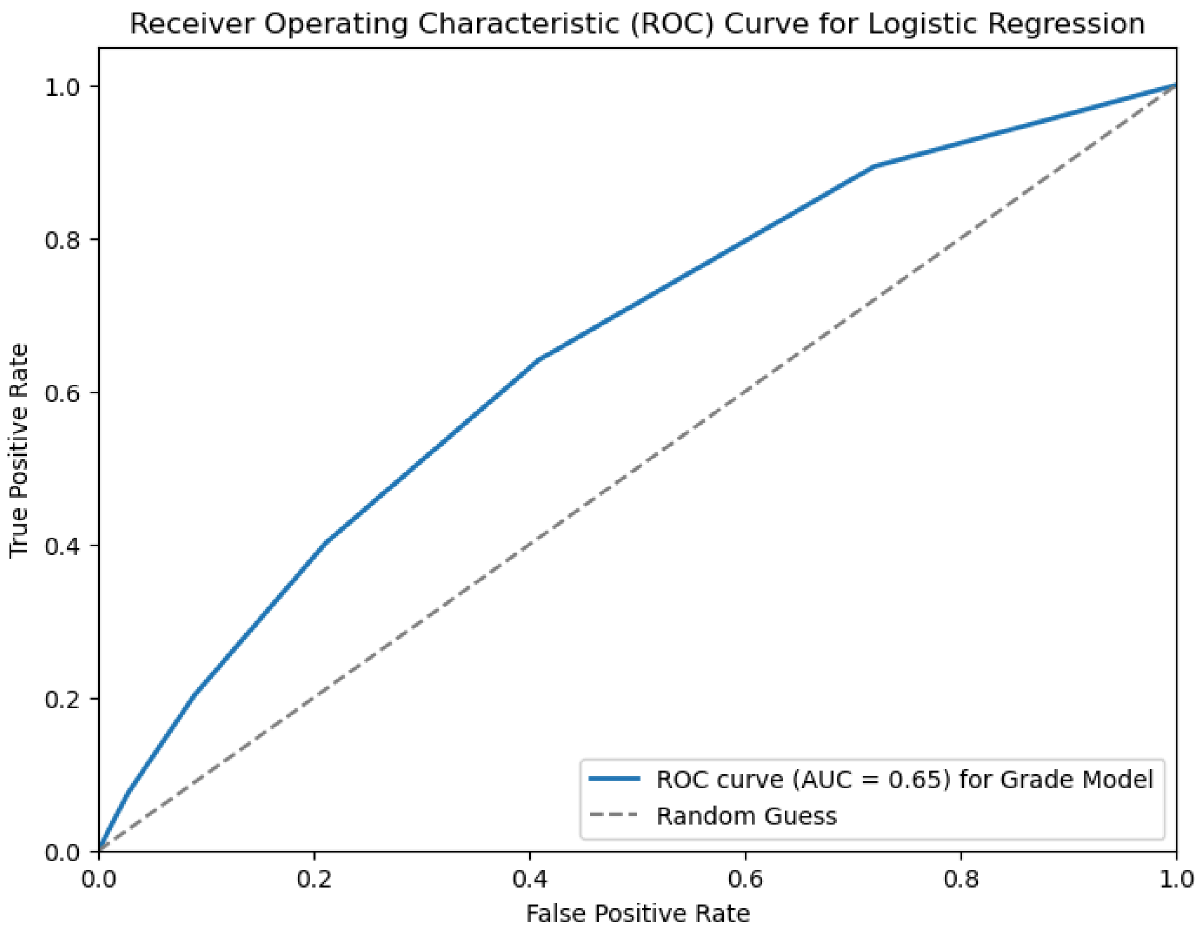
```
Accuracy: 0.8564650360296356
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92     33755
           1       0.00      0.00      0.00      5657

    accuracy                           0.86     39412
   macro avg       0.43      0.50      0.46     39412
weighted avg       0.73      0.86      0.79     39412

                            OLS Regression Results
============================================================================
==
Dep. Variable:            loan_status   R-squared:                       0.0
38
Model:                            OLS   Adj. R-squared:                  0.0
38
Method:                 Least Squares   F-statistic:                     155
7.
Date:                Mon, 22 Apr 2024   Prob (F-statistic):               0.
00
Time:                        08:02:22   Log-Likelihood:                -1385
3.
No. Observations:               39412   AIC:                         2.771e+
04
Df Residuals:                   39410   BIC:                         2.773e+
04
Df Model:                           1
Covariance Type:            nonrobust
============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
----------------------------------------------------------------------------
--
const          0.4114      0.007     58.723      0.000       0.398       0.4
25
grade         -0.0493      0.001    -39.460      0.000      -0.052      -0.0
47
============================================================================
==
Omnibus:                    12915.795   Durbin-Watson:                   1.9
90
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            30824.1
87
Skew:                           1.920   Prob(JB):                         0.
00
Kurtosis:                       5.007   Cond. No.                          2
3.3
============================================================================
==

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

```
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/utils/
validation.py:1143: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

(ii) Construct and report a test of whether the model performs better than the null model where only "beta0", and no conditioning information, is present in the logistic model.

(iii) Construct the lift table and the ROC curve for this model. [When constructing these, think a little about the nature of your model. For instance, does it make sense to use deciles in the lift table or would a different type of cutoff be more sensible?] Explain the interpretation of the numbers in the lift table and the lines and axis in the ROC curve. Does the model perform better than a random guess?

In [5]:
```
def plot_roc(deps, plt_name='Logistic Regression', labels = ['Model']):
    plt.figure(figsize=(8, 6))
    for i in range(len(deps)):
        model, X = deps[i]
        y_probs = model.predict_proba(X)[:, 1]
        fpr, tpr, thresholds = roc_curve(y, y_probs)
        roc_auc = roc_auc_score(y, y_probs)
        thresholds = thresholds[~np.isinf(thresholds)]
        plt.plot(fpr, tpr, lw=2, label=f'ROC curve (AUC = %0.2f) for {labels
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--', label='Random Gue
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f"Receiver Operating Characteristic (ROC) Curve for {plt_name}
    plt.legend(loc="lower right")
    plt.show()

plot_roc([[model, X]], labels = ['Grade Model'])
```




(iv) Assume that each loan is for $100, and that you make a $1 profit if there is no default, but lose $10 if there is a default (both given in present value terms to keep things easy). Using data from the ROC curve (True Positive Rate and False Positive Rate) along with the average rate of default (total number of defaults divided by total number of loans), what is the cutoff default probability you should use as your decision criterion to maximize profits? Plot the corresponding point on the ROC curve.

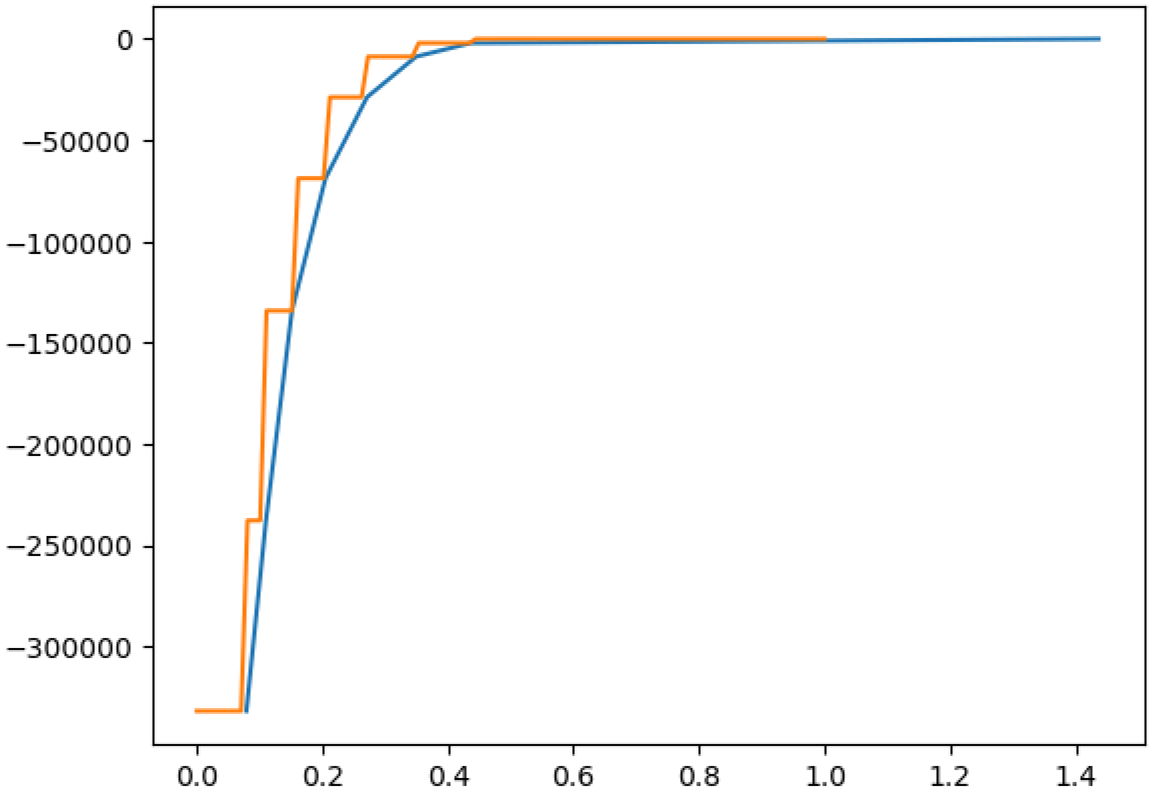
In [6]:
```
profit_non_default = 1
loss_default = 10

y_probs = model.predict_proba(X)[:, 1]
fpr, tpr, thresholds = roc_curve(y, y_probs)
roc_auc = roc_auc_score(y, y_probs)
thresholds = thresholds[~np.isinf(thresholds)]

# Calculate the average rate of default
default_rate = sum(y['loan_status']) / len(y)

thresholds2 = np.linspace(0, 1, 100)
expected_profits = []
expected_profits2 = []
for threshold in thresholds:
    # Predict default status based on the cutoff probability
    y_pred_threshold = y_probs >= threshold
```

```
    # Calculate True Positives (TP) and False Positives (FP)
    TP = sum((y['loan_status'] == 1) & (y_pred_threshold == 1))
    FP = sum((y['loan_status'] == 0) & (y_pred_threshold == 1))

    # Calculate expected profit
    expected_profit = TP * profit_non_default - FP * loss_default
    expected_profits.append(expected_profit)

for threshold in thresholds2:
    # Predict default status based on the cutoff probability
    y_pred_threshold = y_probs >= threshold

    # Calculate True Positives (TP) and False Positives (FP)
    TP = sum((y['loan_status'] == 1) & (y_pred_threshold == 1))
    FP = sum((y['loan_status'] == 0) & (y_pred_threshold == 1))

    # Calculate expected profit
    expected_profit = TP * profit_non_default - FP * loss_default
    expected_profits2.append(expected_profit)

# Find the index of the optimal cutoff based on maximum expected profit
optimal_cutoff_index = np.argmax(expected_profits)
optimal_cutoff_threshold = thresholds[optimal_cutoff_index]

optimal_cutoff_index2 = np.argmax(expected_profits2)
optimal_cutoff_threshold = thresholds2[optimal_cutoff_index2]

print("Optimal Cutoff Default Probability:", optimal_cutoff_threshold)

#The model effectively maps 8 degrees of freedom to [0,1], has its last step
plt.plot(thresholds, expected_profits)
plt.plot(thresholds2, expected_profits2)
```

```
Optimal Cutoff Default Probability: 0.4444444444444445
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x303ad0d10>]
```




c. Next, we will see if it is possible to do better than the internal "grade"-variable, using other information about the borrower and the loan as provided by LendingClub.
(i) First, consider a logistic regression model that uses only loan amount (loan_amnt) and annual income (annual_inc) as explanatory variables. Report the regression results. Show the lift table, comparing to the 'grade'-model from a. Plot the ROC curves of both the 'grade'-model and the alternative model. Which model performs better?

In [7]:
```
X2 = lcData[['loan_amnt', 'annual_inc']]
model2 = LogisticRegression()
model2.fit(X2, y)

y_pred2 = model2.predict(X2)
accuracy = accuracy_score(y, y_pred2)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y, y_pred2))
```

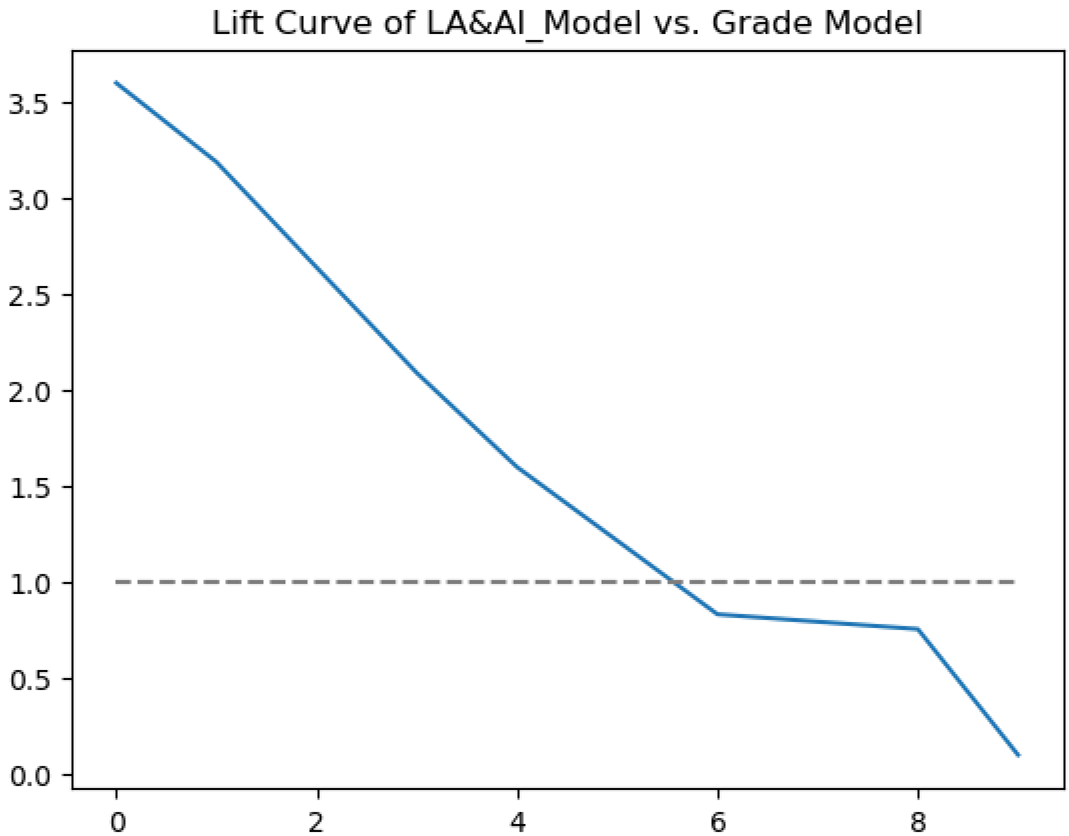
```
Accuracy: 0.8564650360296356
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92     33755
           1       0.00      0.00      0.00      5657

    accuracy                           0.86     39412
   macro avg       0.43      0.50      0.46     39412
weighted avg       0.73      0.86      0.79     39412
```

```
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/utils/
validation.py:1143: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

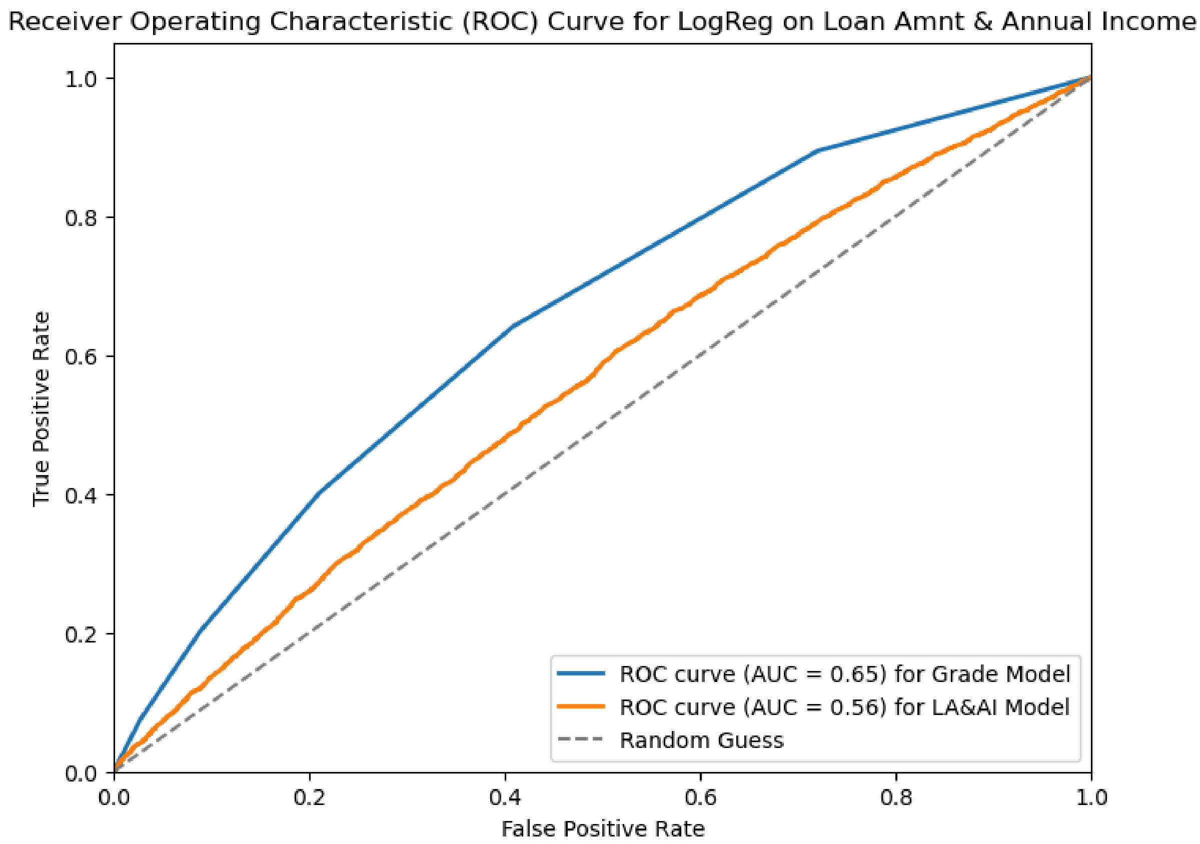
In [8]:

```python
def lift_table_vs_model1(model_new, X_new, model_name='Model_new'):
    # Predict probabilities for both models
    y_probs1 = model.predict_proba(X)[:, 1]
    y_probs2 = model_new.predict_proba(X_new)[:, 1]

    # Create DataFrame to hold predicted probabilities and actual response
    lift_table = pd.DataFrame({'Predicted_Prob_Model1': y_probs1,
                               'Predicted_Prob_Model2': y_probs2})

    # Sort by predicted probability from model 1 in descending order
    lift_table = lift_table.drop_duplicates(subset=['Predicted_Prob_Model1']
    lift_table = lift_table.sort_values(by='Predicted_Prob_Model1', ascendir

    # Calculate deciles based on predicted probabilities from model 1
    lift_table['Decile'] = pd.qcut(lift_table['Predicted_Prob_Model1'], 10,

    # Group by decile and calculate average predicted probability for each m
    lift_summary = lift_table.groupby('Decile').agg({'Predicted_Prob_Model1'
                                                     'Predicted_Prob_Model2'

    # Calculate lift for each decile
    lift_summary[f'{model_name}_lift'] = lift_summary['Predicted_Prob_Model2
    return lift_summary

def plot_lift_curve(model_new, X_new, model_name = 'Model_new'):
    lift_table = lift_table_vs_model1(model_new, X_new, model_name)
    plt.plot(lift_table.index, lift_table[f'{model_name}_lift'], label = moc
    plt.plot(lift_table.index, [1]*len(lift_table.index), linestyle='--', cc
    plt.title(f'Lift Curve of {model_name} vs. Grade Model')
    plt.show()
lift_summary2 = lift_table_vs_model1(model2, X2)

print(lift_summary2)
plot_lift_curve(model2, X2, 'LA&AI_Model')
```

| Decile | Predicted_Prob_Model1 | Predicted_Prob_Model2 | Model_new_lift |
|---|---|---|---|
| 0 | 0.079067 | 0.284199 | 3.594402 |
| 1 | 0.110215 | 0.350886 | 3.183636 |
| 3 | 0.151614 | 0.316383 | 2.086761 |
| 4 | 0.204981 | 0.327477 | 1.597595 |
| 6 | 0.271130 | 0.225637 | 0.832211 |
| 8 | 0.349247 | 0.263793 | 0.755319 |
| 9 | 0.436396 | 0.044106 | 0.101070 |




In [9]:
```
models = [[model, X], [model2, X2]]
labels2 = ['Grade Model', 'LA&AI Model']
plot_roc(models, plt_name='LogReg on Loan Amnt & Annual Income', labels = la
```




In [10]:

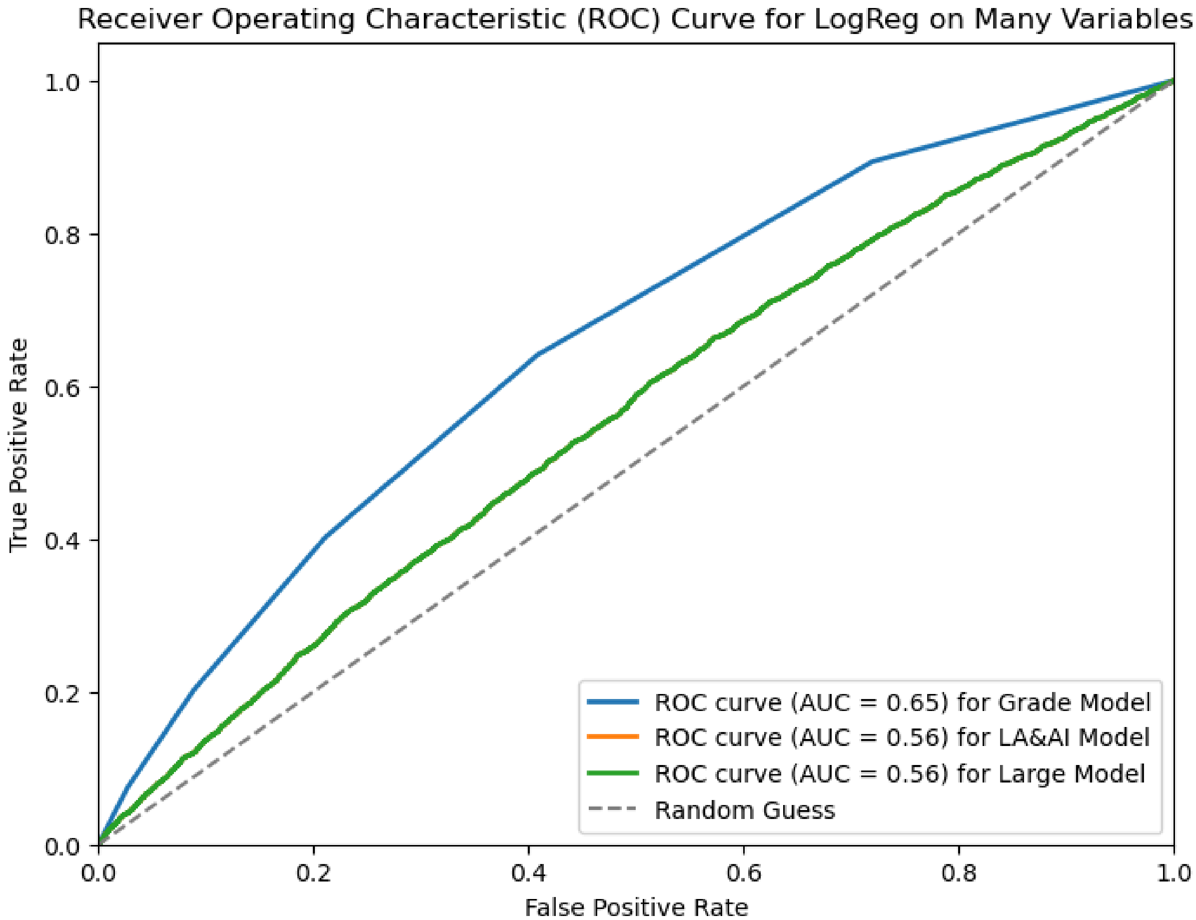
```
#Sklearn obviously doesn't like strings so we map strs to bool is_36 and is_
X3 = lcData[['loan_amnt', 'annual_inc', 'is_36', 'is_60', 'int_rate']]
model3 = LogisticRegression()
model3.fit(X3, y)

y_pred3 = model3.predict(X3)
accuracy = accuracy_score(y, y_pred3)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y, y_pred3))
```

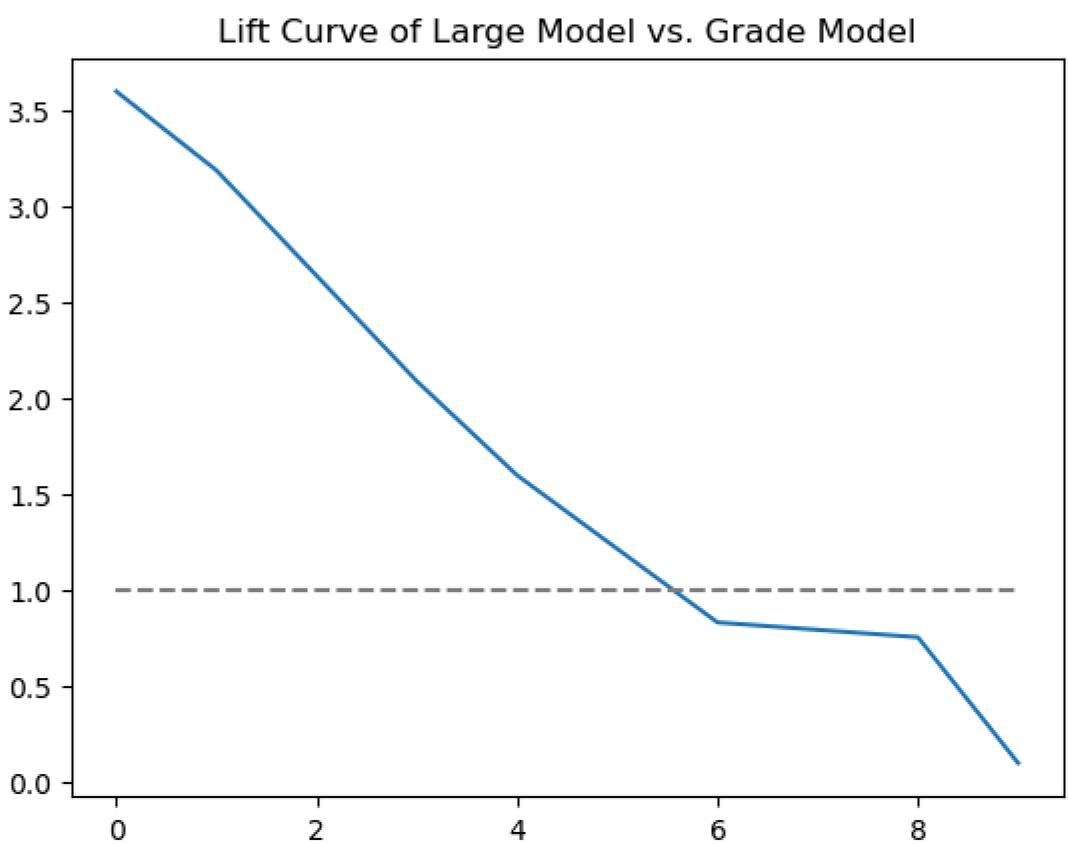
```
Accuracy: 0.8564650360296356
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92     33755
           1       0.00      0.00      0.00      5657

    accuracy                           0.86     39412
   macro avg       0.43      0.50      0.46     39412
weighted avg       0.73      0.86      0.79     39412
```

```
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/utils/
validation.py:1143: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

(ii) Now, include also information from the loan itself. In particular, include the maturity of the loan (term) and the interest rate (int_rate) in the logistic regression. Report the output. How does Python handle the term variable? In particular, what is the interpretation of the regression coefficient? Again show the lift table and ROC curve relative to the original 'grade' model. Now, which model is better? What is the likely explanation for why this new model performs better/worse?

In [11]:
```
models = [[model, X], [model2, X2], [model3, X3]]
labels3 = ['Grade Model', 'LA&AI Model', 'Large Model']
plot_roc(models, plt_name='LogReg on Many Variables', labels = labels3)
plot_lift_curve(model3, X3, labels3[2])
```


Receiver Operating Characteristic (ROC) Curve for LogReg on Many Variables
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.65) for Grade Model
ROC curve (AUC = 0.56) for LA&AI Model
ROC curve (AUC = 0.56) for Large Model
Random Guess


Lift Curve of Large Model vs. Grade Model
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8

In [12]:
```
lift_table3 = lift_table_vs_model1(model3, X3)
lift_table3
# decile = np.linspace(0, 9, 10)
# print(lift_table3['Predicted_Prob_Model1'])
# plt.plot(decile, lift_table3['Predicted_Prob_Model1'])
# plt.plot(decile, lift_table3['Predicted_Prob_Model2'])
```

Out[12]:

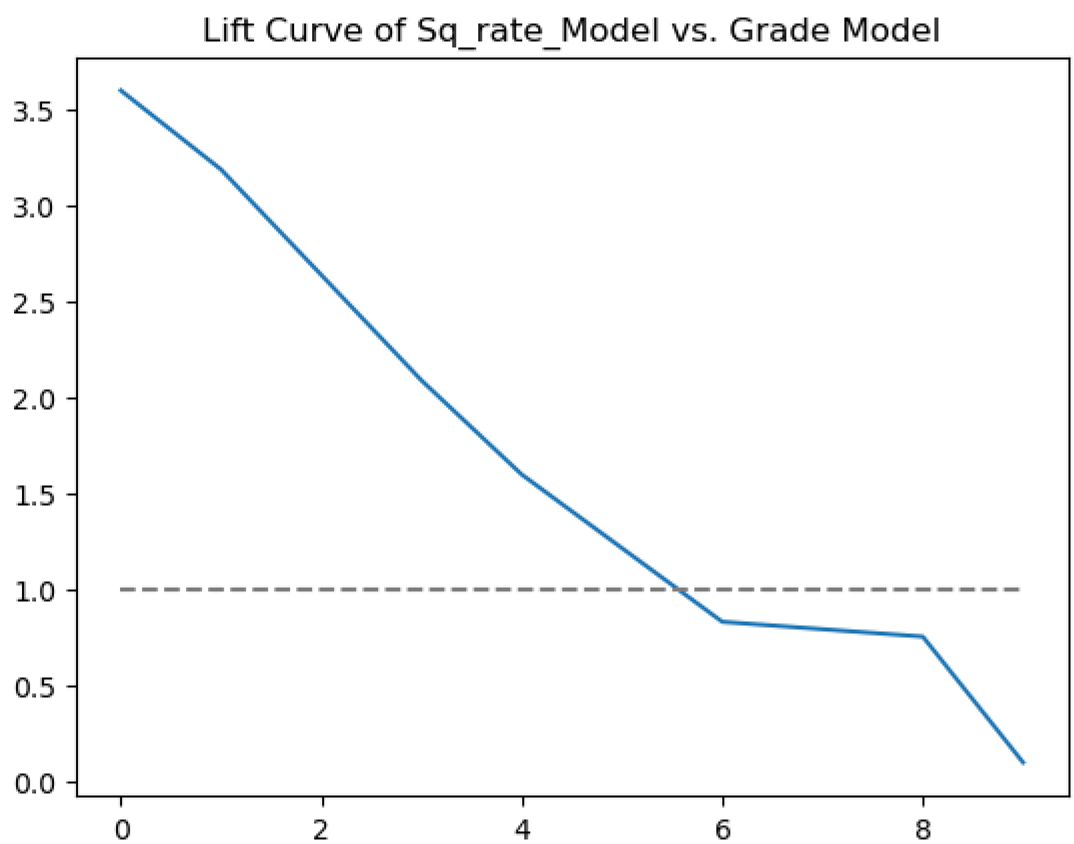
| Decile | Predicted_Prob_Model1 | Predicted_Prob_Model2 | Model_new_lift |
|---|---|---|---|
| **0** | 0.079067 | 0.284199 | 3.594401 |
| **1** | 0.110215 | 0.350886 | 3.183636 |
| **3** | 0.151614 | 0.316383 | 2.086760 |
| **4** | 0.204981 | 0.327477 | 1.597595 |
| **6** | 0.271130 | 0.225637 | 0.832210 |
| **8** | 0.349247 | 0.263793 | 0.755319 |
| **9** | 0.436396 | 0.044106 | 0.101070 |

In [13]:
```
lcData['sq_rate'] = lcData['int_rate']**2
X4 = X3.assign(sq_rate=lcData['sq_rate'])
model4 = LogisticRegression()
model4.fit(X4, y)

y_pred4 = model4.predict(X4)
accuracy = accuracy_score(y, y_pred4)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y, y_pred4))
```

```
Accuracy: 0.8564650360296356
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92     33755
           1       0.00      0.00      0.00      5657

    accuracy                           0.86     39412
   macro avg       0.43      0.50      0.46     39412
weighted avg       0.73      0.86      0.79     39412
```

```
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/utils/
validation.py:1143: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/a.kanstantsinau/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
ill-defined and being set to 0.0 in labels with no predicted samples. Use `z
ero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [14]:
```
plot_lift_curve(model4, X4, 'Sq_rate_Model')
lift_table_vs_model1(model4, X4, 'Sq_rate_Model')
```



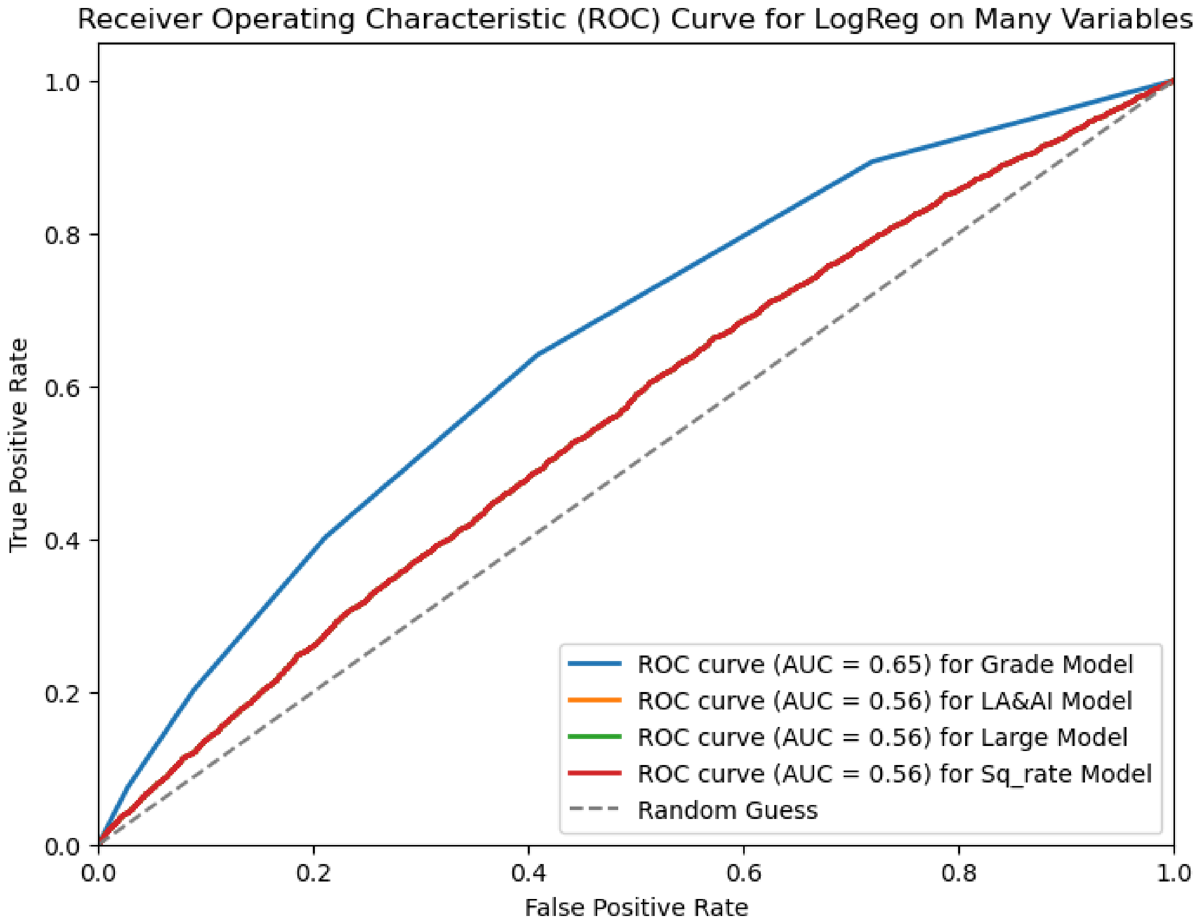
Out[14]:

| Decile | Predicted_Prob_Model1 | Predicted_Prob_Model2 | Sq_rate_Model_lift |
|---|---|---|---|
| 0 | 0.079067 | 0.284199 | 3.594401 |
| 1 | 0.110215 | 0.350886 | 3.183636 |
| 3 | 0.151614 | 0.316383 | 2.086760 |
| 4 | 0.204981 | 0.327477 | 1.597595 |
| 6 | 0.271130 | 0.225637 | 0.832210 |
| 8 | 0.349247 | 0.263793 | 0.755319 |
| 9 | 0.436396 | 0.044106 | 0.101070 |

(iii) Create the squared of the interest rate and add this variable to the last model. Is the coefficient on this variable significant? Please give an intuition for what the coefficients on both int_rate and its squared value imply for the relationship between defaults and the interest rate.

In [15]:

```
models = [[model, X], [model2, X2], [model3, X3], [model4, X4]]
labels4 = ['Grade Model', 'LA&AI Model', 'Large Model', 'Sq_rate Model']
plot_roc(models, plt_name='LogReg on Many Variables', labels = labels4)
print(model3.coef_[0][4])
print(model4.coef_[0][4:])
```




```
-6.697269828267191e-10
[-6.69726983e-10  2.73148297e-11]
```

The model seems to perform worse when adding a term dependent on the square of the interest rate. This likely suggests that the dependence of the default rate to the interest rate is likely closer to $O(r)$ rather than it is to $O(r^2)$ hence why adding a term proportional to the square of the interest rate degrades model performance. As we see from the coefficients, the model including the squared interest rate fits default rate onto $1.90198827*r + 0.49913123*r^2$ whereas model3 fits it onto $11.17886728949225*r$, which seems to perform better.