VXX OPTIONS PRICING

Black-Scholes vs Deep Learning

INTRODUCTION

- Options are financial derivative contracts that give buyers the right to buy or sell a security at a predetermined price on or before a specified date
- Options are used by traders to hedge against and speculate on future price changes of an underlying security
- The accurate valuation of options is crucial in managing financial risk when hedging or speculating thus there is a substantial amount of research on the subject
- This project explores options pricing via…

OPTION VALUATION MODELS

- Early valuation models suffer from various assumptions about financial markets and individual securities that limit their practical efficacy
- Enabled by improvements in data availability and computing power, machine learning models that avoid many of these assumptions have been shown to improve performance in practice
- Artificial neural networks (ANNs) are one type of machine (deep) learning model that do not require assumptions and are considered universal function approximators given their ability to represent a wide variety of arbitrary functions
- These features allow ANNs to generalize well and make them a good choice for pricing options

PROJECT

- **Goal**: Compare the performance of the seminal Black-Scholes (BS) model to the performance of a multi-layer perceptron (MLP) deep learning model in pricing VXX options. Replicating the results of [1]Option Pricing with Deep Learning in accordance with [2]Pricing options and computing implied volatilities using neural networks
- **Data:**
  - VXX Options - 3,147,375 observations from 2010 to 2019 - STRIKE PRICE & TIME TO MATURITY
  - VXX Securities - from 2010 to 2019 corresponding with the options - UNDERLYING SECURITY PRICE & IMPLIED VOLATILITY
  - [3]Treasury Yields Data – from 2010 to 2019 corresponding with the options – RISK FREE RATE

  * The iPath S&P 500 VIX Short-Term Futures ETN (VXX) was an ETN and thus traded like an ETF until its ultimate expiration on Jan. 31, 2019. Over its life the VXX was primarily used to speculate on and hedge against market volatility as tracked by the VIX
- **Method:**
  - BS - simple to implement and measure performance with the given formula
  - MLP - multi-layer perceptron model with inputs identical to the BS model so that performance can be directly compared. Hyperparameters will initially be set in accordance with the two model papers where appropriate.

BLACK-SCHOLES FORMULA

- Notable assumptions:
  - Option can only be exercised at expiration (European)
  - No dividends are paid out over the life of the option
  - Efficient markets (no arbitrage)
  - Known & constant risk-free rate and volatility
  - Returns of underlying asset are log-normally distributed (prices follow a random walk with constant drift and volatility)
- Model Inputs:
  - S = Underlying security price
  - σ = Underlying security implied volatility
  - X = Option strike price
  - r = Risk-free rate
  - t = Time to expiration (years)

- Call Option Price
  $C = S \cdot \Phi(d_1) - Xe^{-rt} \cdot \Phi(d_2)$
- Put Option Price
  $P = Xe^{-r\,t}[1 - \Phi(d2)] - S\,[1 - \Phi(d_1)]$
- Note:
  $d_1 = [\ln(S/X) + (r + \sigma^2/2)t] \,/\, \sigma\sqrt{t}$
  $d_2 = d_1 - \sigma\sqrt{t}$
  Φ = cumulative density function of the normal distribution

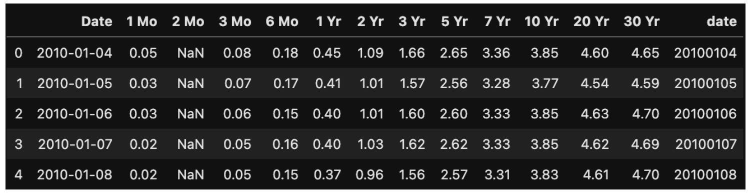
MULTI-LAYER PERCEPTRON MODEL

- Notable Assumptions:
  - Underlying security implied volatility
- Model Inputs
  - Underlying security price
  - Underlying security implied volatility
  - Option strike price
  - Risk-free rate
  - Time to expiration (years)
- Model Outputs
  - Bid Price
  - Ask Price

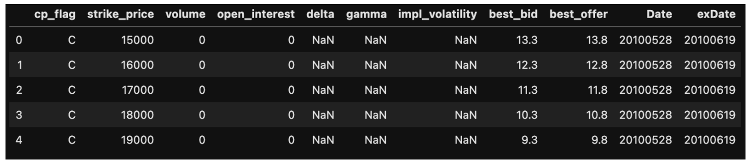
- Hyperparameters:
  - 5 input nodes for each model input
  - 3 hidden layers with 400 nodes each
    - Batch Normalization
    - Leaky ReLu
  - Two output nodes for bid and ask price
    - ReLu
  - Batch Size: 1024
  - Epochs: 250
  - Optimizer: Adam

PROCESS

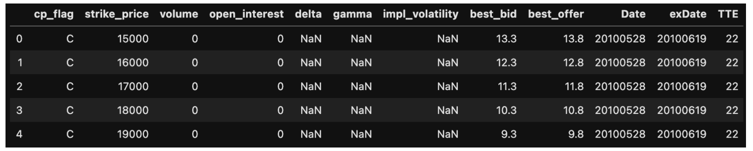
1. **Treasury Pre-Processing**
2. **Options Pre-Processing**
3. **Options Feature Engineering**
4. **Securities Feature Engineering**
5. **Data Integration and Reduction**
6. **Model Building**



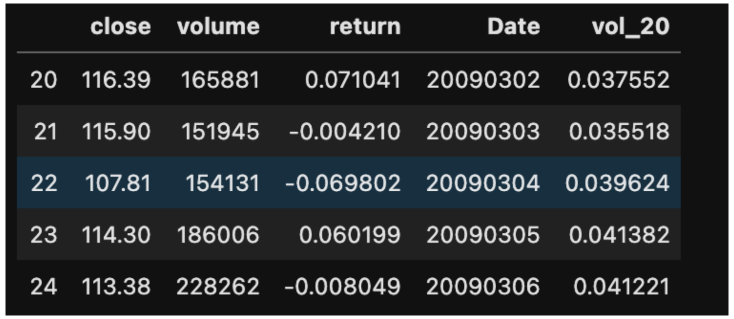
	Date	1 Mo	2 Mo	3 Mo	6 Mo	1 Yr	2 Yr	3 Yr	5 Yr	7 Yr	10 Yr	20 Yr	30 Yr	date
0	2010-01-04	0.05	NaN	0.08	0.18	0.45	1.09	1.66	2.65	3.36	3.85	4.60	4.65	20100104
1	2010-01-05	0.03	NaN	0.07	0.17	0.41	1.01	1.57	2.56	3.28	3.77	4.54	4.59	20100105
2	2010-01-06	0.03	NaN	0.06	0.15	0.40	1.01	1.60	2.60	3.33	3.85	4.63	4.70	20100106
3	2010-01-07	0.02	NaN	0.05	0.16	0.40	1.03	1.62	2.62	3.33	3.85	4.62	4.69	20100107
4	2010-01-08	0.02	NaN	0.05	0.15	0.37	0.96	1.56	2.57	3.31	3.83	4.61	4.70	20100108



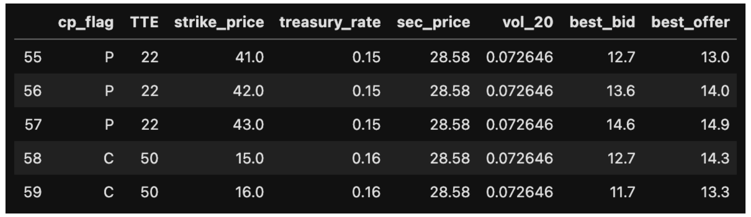
	cp_flag	strike_price	volume	open_interest	delta	gamma	impl_volatility	best_bid	best_offer	Date	exDate
0	C	15000	0	0	NaN	NaN	NaN	13.3	13.8	20100528	20100619
1	C	16000	0	0	NaN	NaN	NaN	12.3	12.8	20100528	20100619
2	C	17000	0	0	NaN	NaN	NaN	11.3	11.8	20100528	20100619
3	C	18000	0	0	NaN	NaN	NaN	10.3	10.8	20100528	20100619
4	C	19000	0	0	NaN	NaN	NaN	9.3	9.8	20100528	20100619



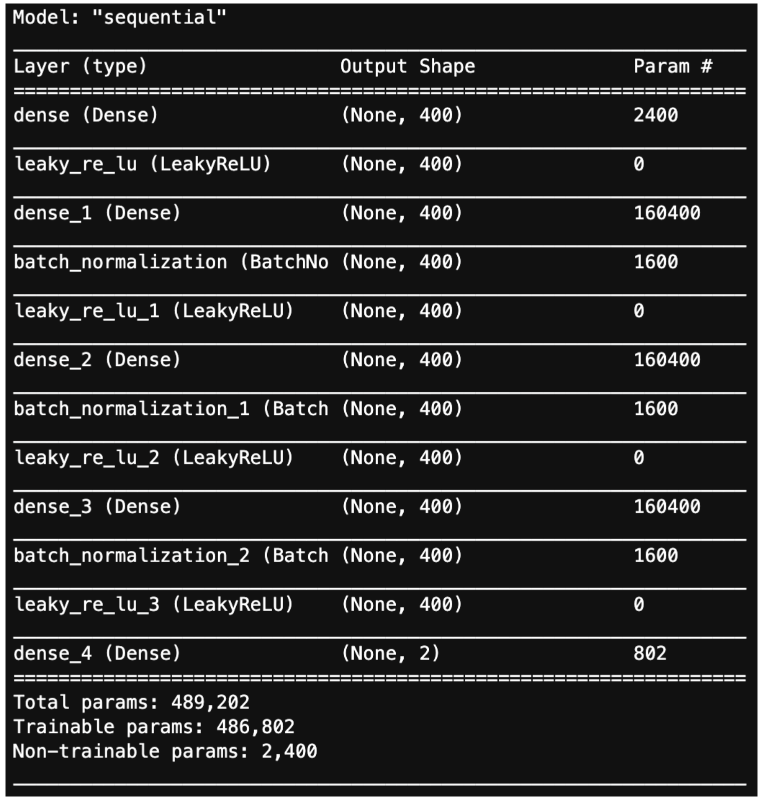
	cp_flag	strike_price	volume	open_interest	delta	gamma	impl_volatility	best_bid	best_offer	Date	exDate	TTE
0	C	15000	0	0	NaN	NaN	NaN	13.3	13.8	20100528	20100619	22
1	C	16000	0	0	NaN	NaN	NaN	12.3	12.8	20100528	20100619	22
2	C	17000	0	0	NaN	NaN	NaN	11.3	11.8	20100528	20100619	22
3	C	18000	0	0	NaN	NaN	NaN	10.3	10.8	20100528	20100619	22
4	C	19000	0	0	NaN	NaN	NaN	9.3	9.8	20100528	20100619	22



	close	volume	return	Date	vol_20
20	116.39	165881	0.071041	20090302	0.037552
21	115.90	151945	-0.004210	20090303	0.035518
22	107.81	154131	-0.069802	20090304	0.039624
23	114.30	186006	0.060199	20090305	0.041382
24	113.38	228262	-0.008049	20090306	0.041221



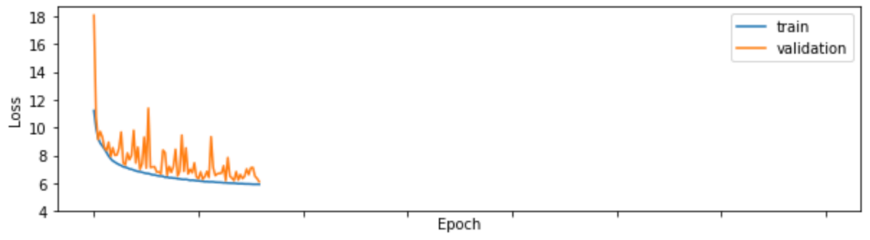
	cp_flag	TTE	strike_price	treasury_rate	sec_price	vol_20	best_bid	best_offer
55	P	22	41.0	0.15	28.58	0.072646	12.7	13.0
56	P	22	42.0	0.15	28.58	0.072646	13.6	14.0
57	P	22	43.0	0.15	28.58	0.072646	14.6	14.9
58	C	50	15.0	0.16	28.58	0.072646	12.7	14.3
59	C	50	16.0	0.16	28.58	0.072646	11.7	13.3



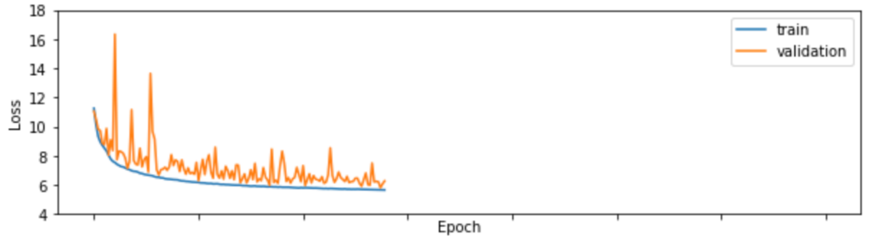
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 400)               2400
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 400)               160400
_________________________________________________________________
batch_normalization (BatchNo (None, 400)               1600
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 400)               0
_________________________________________________________________
dense_2 (Dense)              (None, 400)               160400
_________________________________________________________________
batch_normalization_1 (Batch (None, 400)               1600
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 400)               160400
_________________________________________________________________
batch_normalization_2 (Batch (None, 400)               1600
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 400)               0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 802
=================================================================
Total params: 489,202
Trainable params: 486,802
Non-trainable params: 2,400
_________________________________________________________________
```

RESULTS & ANALYSIS - TRAIN

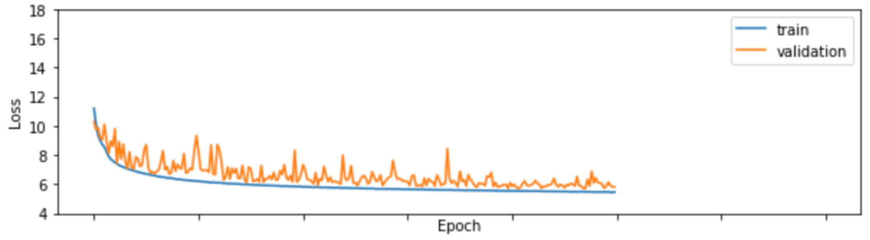
1



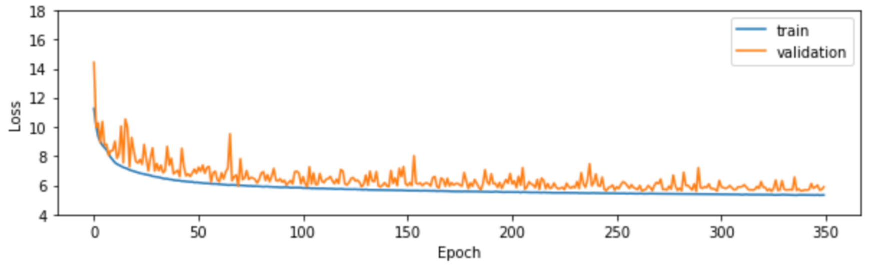
2



3



4



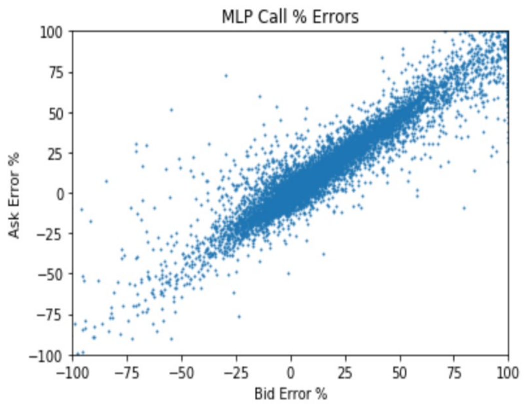
Loss	80	140	250	350
1	5.93			
2	5.93	5.68		
3	5.95	5.68	5.46	
4	5.94	5.67	5.45	5.33

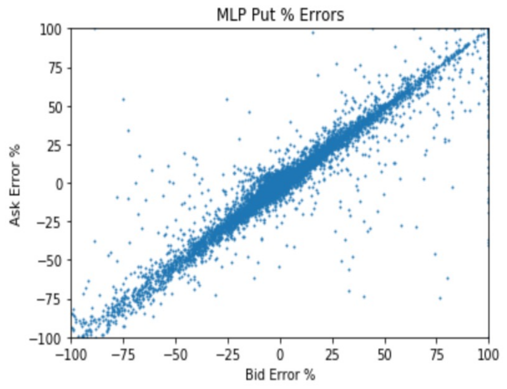
Validation Loss	80	140	250	350
1	6.64			
2	6.71	6.13		
3	6.40	6.23	5.90	
4	6.40	6.19	5.80	5.83

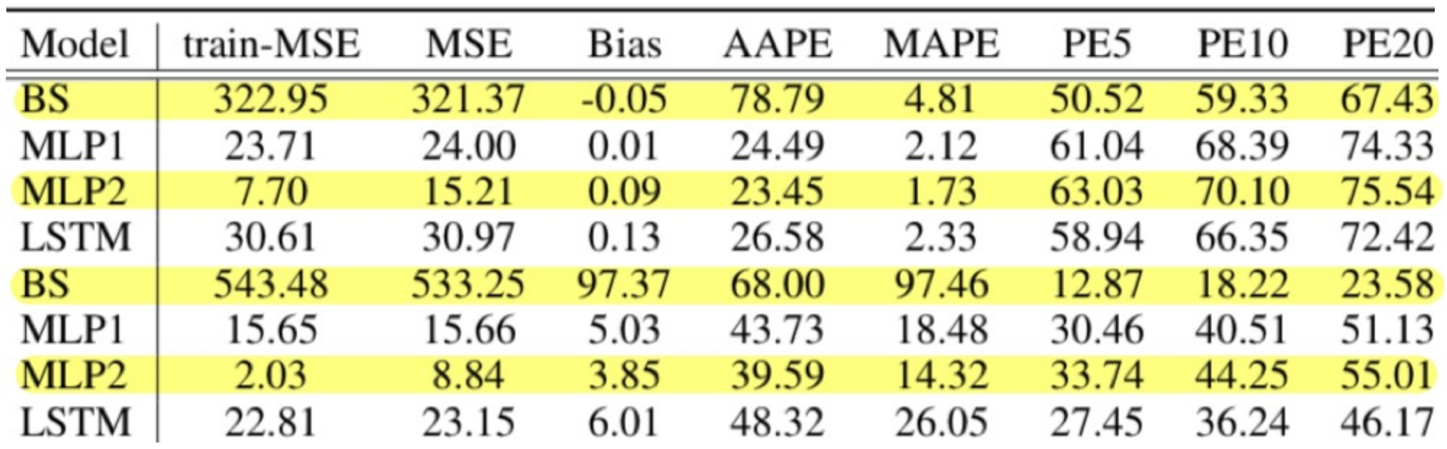
*plots created using [3]Matplotlib

RESULTS & ANALYSIS - TEST

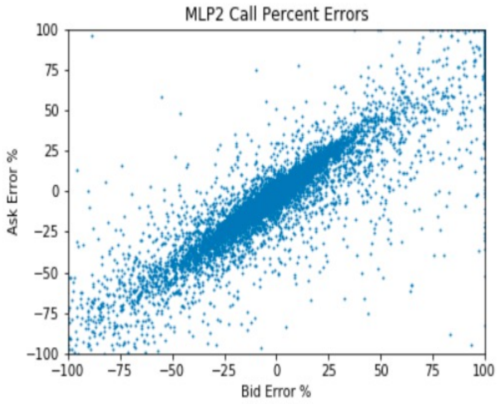
		MSE	BIAS	AAPE	MAPE	PE5	PE10	PE20
Call	MLP	5.14	16.98	863.24	29.58	16.62	28.60	42.37
	BS	36.65	100	2517.93	100	16.19	18.85	21.81
Put	MLP	10.77	0.69	1055.32	8.61	42.12	51.84	60.72
	BS	384.31	193.42	29202.97	427.18	3.61	4.82	7.23

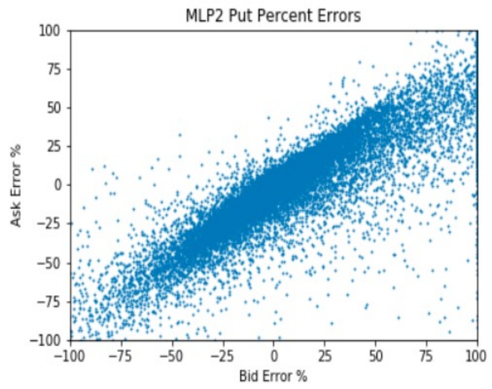






	Model	train-MSE	MSE	Bias	AAPE	MAPE	PE5	PE10	PE20
Call	BS	322.95	321.37	-0.05	78.79	4.81	50.52	59.33	67.43
	MLP1	23.71	24.00	0.01	24.49	2.12	61.04	68.39	74.33
	MLP2	7.70	15.21	0.09	23.45	1.73	63.03	70.10	75.54
	LSTM	30.61	30.97	0.13	26.58	2.33	58.94	66.35	72.42
Put	BS	543.48	533.25	97.37	68.00	97.46	12.87	18.22	23.58
	MLP1	15.65	15.66	5.03	43.73	18.48	30.46	40.51	51.13
	MLP2	2.03	8.84	3.85	39.59	14.32	33.74	44.25	55.01
	LSTM	22.81	23.15	6.01	48.32	26.05	27.45	36.24	46.17

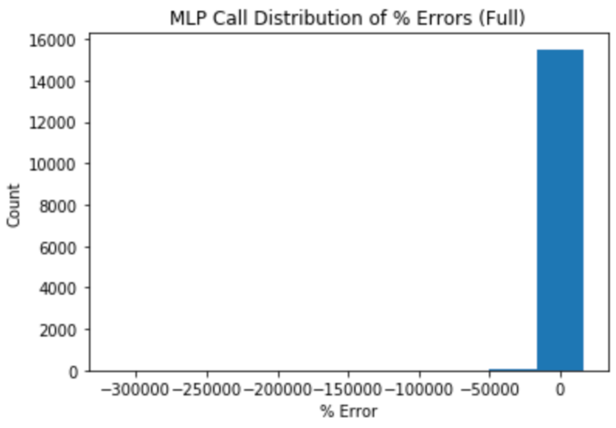


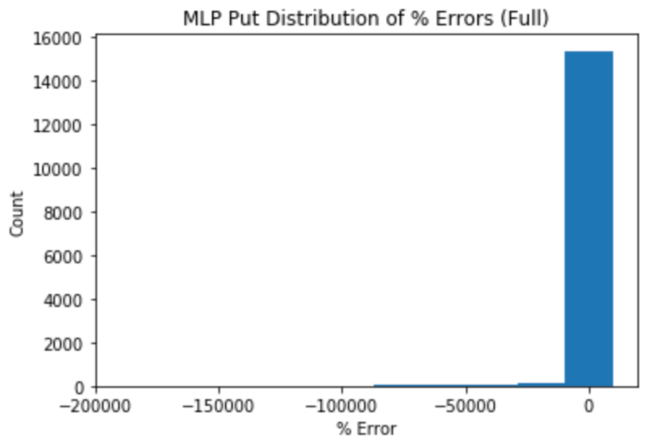


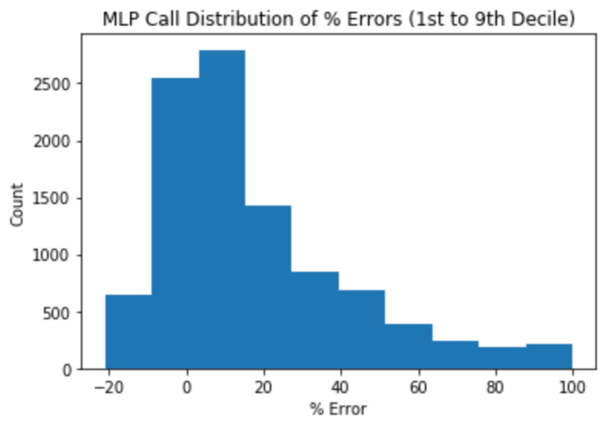
* All metrics in percentages except for MSE

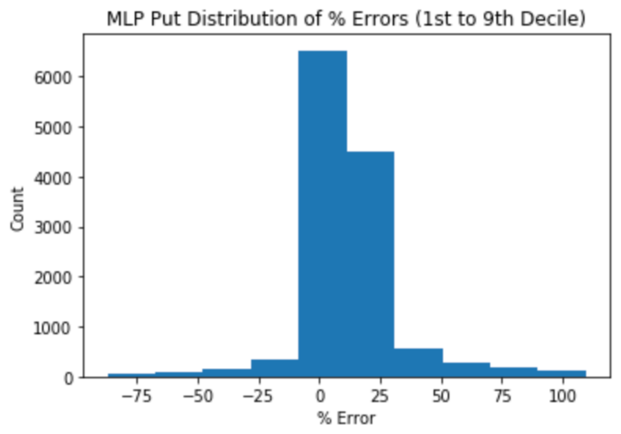
RESULTS & ANALYSIS - TEST

		Obs.	Min	Max	AVG	BIAS	AAPE	MAPE	PE5	PE10	PE20
Full	Call	15,604	-333,557	100	-786.88	16.98	863.24	29.58	16.62	28.60	42.37
	Put	15,604	-193,356	100	-1020.92	0.69	1055.32	8.61	42.12	51.84	60.72
1st to 9th	Call	9,960	-21.04	99.96	17.31	9.83	20.31	11.77	26.03	44.81	66.39
	Put	12,251	-97.07	99.74	0.84	0.59	13.72	4.22	53.65	66.03	77.33









* All metrics in percentages except for MSE

CONCLUSION

- The MLP neural network models achieved performance significantly superior to the BS model when pricing VXX options given identical model inputs
- The MLP neural network models tend to overestimate option prices (especially call options)
- Trimming the test percent errors to exclude unusually large errors that would be dismissed in practice results in improved model performance
- Model results were comparable to [1]Option Pricing with Deep Learning indicating some consistency in using MLP models to price options across different market conditions
- Future work could include additional model inputs, additional hyperparameter tuning and the testing of different network architectures

CITATIONS

[1] Ke, Alexander, and Andrew Yang. 2019, *Option Pricing with Deep Learning*, cs230.stanford.edu/projects_fall_2019/reports/26260984.pdf.

[2] S.Liu, C.Oosterlee, and S.Bohte. Pricing options and computing implied volatilities using neural networks. *Risks*, 7:16, 02 2019.

[3] U.S. Department of the Treasury. Daily treasury yield curve rates. https://www.treasury.gov/ resource-center/data-chart-center/interest-rates/pages/textview.aspx?data=yield, 2019.

[4] J.D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vo. 9, no. 3, pp. 90-95, 2007.

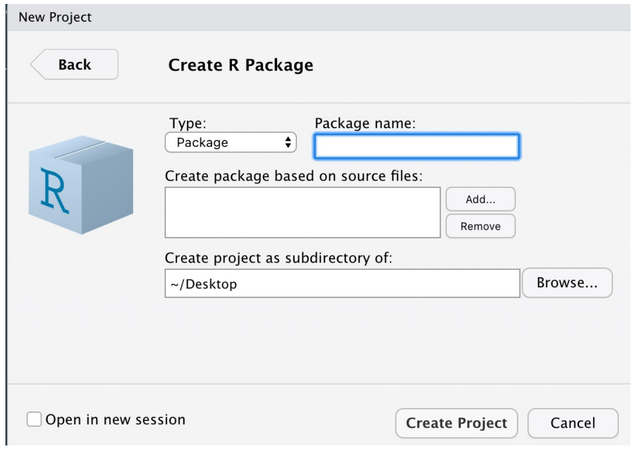
[5] Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras

CREATING AN R PACKAGE

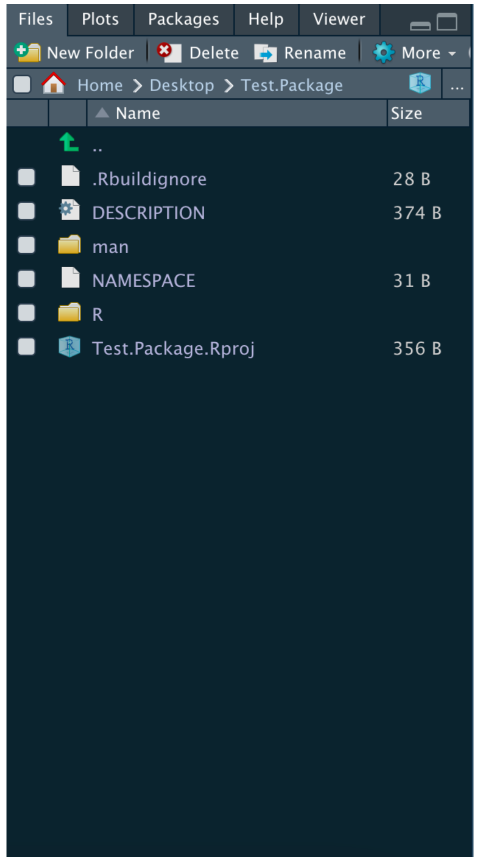
METHOD 1: R-STUDIO

1. Write the package code in a single or multiple .R file(s) and save to a known directory
2. Install 'devtools' package using the following code:
   - install.packages('devtools')
3. Create a new R Package by clicking:
   - File → New Project → New Directory → R Package
4. Type in the package name, click 'Add…' and select the .R file(s) with the package code, then click 'Create Project'
5. Observe the file directory on the bottom right hand side
   - 'R' folder: contains .R file(s) with package code
   - 'man' folder: contains .Rd help files for each function. If this file is empty see step 6

4



5



Name	Size
..	
.Rbuildignore	28 B
DESCRIPTION	374 B
man	
NAMESPACE	31 B
R	
Test.Package.Rproj	356 B

METHOD 1: R-STUDIO

6. Create new documentation files by clicking:

- File → New File → R Documentation

Enter the title of the function, select function from the Rd template menu and select OK

7. Add title text in the title { } field then save the .Rd file in the 'man' file directory from step 5

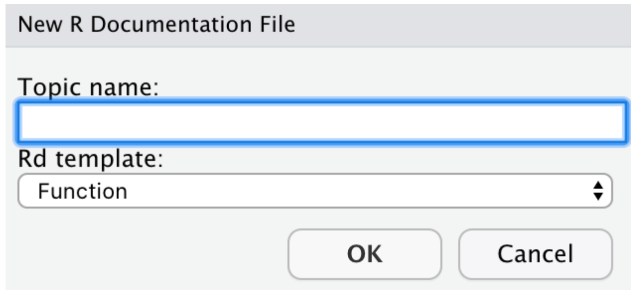
8. Compile the package by clicking:

- Build → Clean and Rebuild
- OR use the command 'CTRL + SHIFT + B'

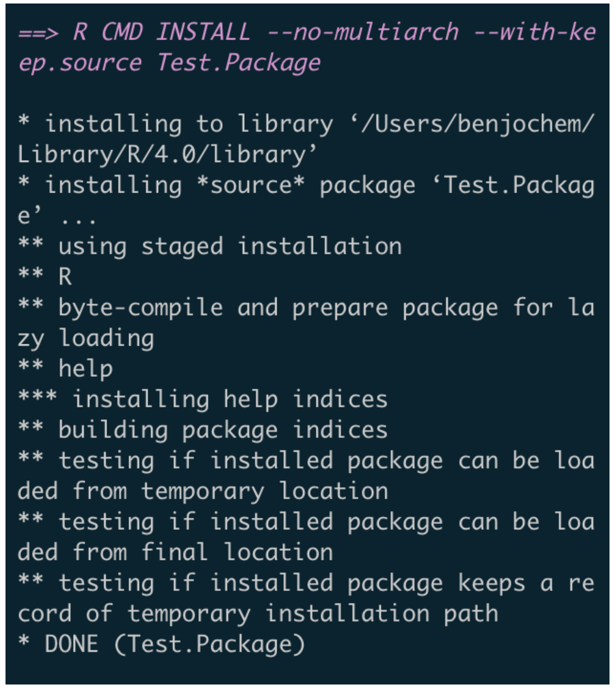
9. If correctly executed, the package will automatically load and you will see 'library(package)'

- Test the functions to make sure they work and make any needed edits to .R or .Rd files
- NOTE: if edits are made step 8 must be repeated as the package needs to be compiled again

6



9



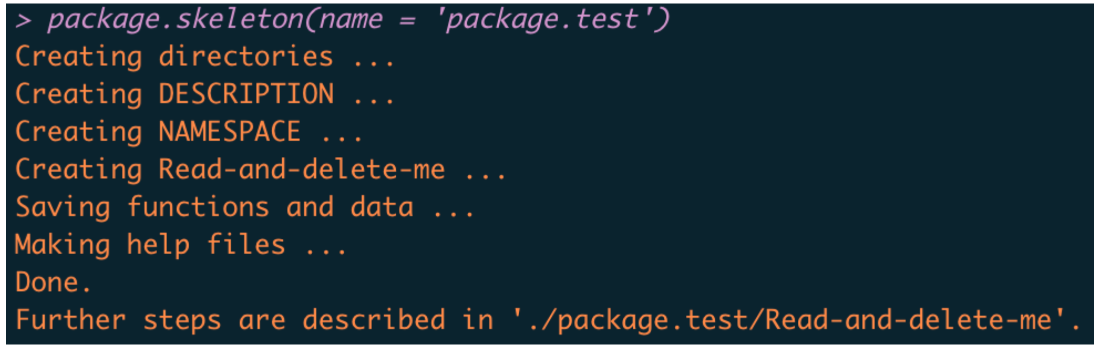
```
==> R CMD INSTALL --no-multiarch --with-ke
ep.source Test.Package

* installing to library '/Users/benjochem/
Library/R/4.0/library'
* installing *source* package 'Test.Packag
e' ...
** using staged installation
** R
** byte-compile and prepare package for la
zy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loa
ded from temporary location
** testing if installed package can be loa
ded from final location
** testing if installed package keeps a re
cord of temporary installation path
* DONE (Test.Package)
```

METHOD 2: COMMAND LINE

1. Open R or RStudio and note your current directory
2. Install / load the 'devtools' and 'roxygen2' libraries
3. Write the package code in a .R file and run the following command in the same file:
   - package.skeleton(name = 'package_name')

   This will create a new folder in the directory noted above with the provided package name
4. Click on the newly created folder with your package name and navigate to the 'man' folder
   - Here you should see .Rd files for your package and all of the functions in the package
   - Use RStudio or a text editor to click on these files and add a title and any additional information

2 – Output of 'package.skeleton(name = 'package_name')'



```
> package.skeleton(name = 'package.test')
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './package.test/Read-and-delete-me'.
```

METHOD 2: COMMAND LINE (MAC)

5. Click on the Terminal application and go to the directory that your package is located in by using the command:

   - Cd '/ directory goes here /'

6. Create the R package by using the following commands in the following order hitting return after each:

   - R CMD build package_name
   - R CMD INSTALL package_name_0.1.tar.gz

7. Attempt to load your package into R or RStudio. Test your functions

   - If any changes are made to the package, package functions or help files then step 6 must be repeated

5 – Terminal Icon