Question 1: House Price Prediction

Team Members:

Marcel Santos de Carvalho, id 79083

Loris Baudry, id 79794

Alex Palacios, id 73713

Responsible for this notebook: Loris Baudry

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

1. Loading and Preview of the Data Frame

1.1. Loading

In [5]:
```
# We read the file into the data frame
df = pd.read_csv('housing/train.csv')
```

1.2. Preview of the Data Frame

In [6]:
```
# Have a preview of the first 10 rows
df.head(5)
```

Out[6]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeatu
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	Na

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeatu
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	Na
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Na
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Na
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Na

5 rows × 81 columns

In [7]:
```
# Find out how many data rows and columns we have
df.shape
```

Out[7]:
```
(1460, 81)
```

In [9]:
```
a=open('housing/data_description.txt')
descr=a.read()
a.close()
print(descr)
```

```
MSSubClass: Identifies the type of dwelling involved in the sale.

        20      1-STORY 1946 & NEWER ALL STYLES
        30      1-STORY 1945 & OLDER
        40      1-STORY W/FINISHED ATTIC ALL AGES
        45      1-1/2 STORY - UNFINISHED ALL AGES
        50      1-1/2 STORY FINISHED ALL AGES
        60      2-STORY 1946 & NEWER
        70      2-STORY 1945 & OLDER
        75      2-1/2 STORY ALL AGES
        80      SPLIT OR MULTI-LEVEL
        85      SPLIT FOYER
        90      DUPLEX - ALL STYLES AND AGES
       120      1-STORY PUD (Planned Unit Development) - 1946 & NEWER
       150      1-1/2 STORY PUD - ALL AGES
       160      2-STORY PUD - 1946 & NEWER
       180      PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
       190      2 FAMILY CONVERSION - ALL STYLES AND AGES
```

MSZoning: Identifies the general zoning classification of the sale.

A	Agriculture
C	Commercial
FV	Floating Village Residential
I	Industrial
RH	Residential High Density
RL	Residential Low Density
RP	Residential Low Density Park
RM	Residential Medium Density

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

Grvl	Gravel
Pave	Paved

Alley: Type of alley access to property

Grvl	Gravel
Pave	Paved
NA	No alley access

LotShape: General shape of property

Reg	Regular
IR1	Slightly irregular
IR2	Moderately Irregular
IR3	Irregular

LandContour: Flatness of the property

Lvl	Near Flat/Level
Bnk	Banked - Quick and significant rise from street grade to building
HLS	Hillside - Significant slope from side to side
Low	Depression

Utilities: Type of utilities available

AllPub	All public Utilities (E,G,W,& S)
NoSewr	Electricity, Gas, and Water (Septic Tank)
NoSeWa	Electricity and Gas Only

```
       ELO      Electricity only

LotConfig: Lot configuration

       Inside   Inside lot
       Corner   Corner lot
       CulDSac  Cul-de-sac
       FR2      Frontage on 2 sides of property
       FR3      Frontage on 3 sides of property

LandSlope: Slope of property

       Gtl      Gentle slope
       Mod      Moderate Slope
       Sev      Severe Slope

Neighborhood: Physical locations within Ames city limits

       Blmngtn  Bloomington Heights
       Blueste  Bluestem
       BrDale   Briardale
       BrkSide  Brookside
       ClearCr  Clear Creek
       CollgCr  College Creek
       Crawfor  Crawford
       Edwards  Edwards
       Gilbert  Gilbert
       IDOTRR   Iowa DOT and Rail Road
       MeadowV  Meadow Village
       Mitchel  Mitchell
       Names    North Ames
       NoRidge  Northridge
       NPkVill  Northpark Villa
       NridgHt  Northridge Heights
       NWAmes   Northwest Ames
       OldTown  Old Town
       SWISU    South & West of Iowa State University
       Sawyer   Sawyer
       SawyerW  Sawyer West
       Somerst  Somerset
       StoneBr  Stone Brook
       Timber   Timberland
       Veenker  Veenker

Condition1: Proximity to various conditions
```

```
       Artery   Adjacent to arterial street
       Feedr    Adjacent to feeder street
       Norm     Normal
       RRNn     Within 200' of North-South Railroad
       RRAn     Adjacent to North-South Railroad
       PosN     Near positive off-site feature--park, greenbelt, etc.
       PosA     Adjacent to postive off-site feature
       RRNe     Within 200' of East-West Railroad
       RRAe     Adjacent to East-West Railroad

Condition2: Proximity to various conditions (if more than one is present)

       Artery   Adjacent to arterial street
       Feedr    Adjacent to feeder street
       Norm     Normal
       RRNn     Within 200' of North-South Railroad
       RRAn     Adjacent to North-South Railroad
       PosN     Near positive off-site feature--park, greenbelt, etc.
       PosA     Adjacent to postive off-site feature
       RRNe     Within 200' of East-West Railroad
       RRAe     Adjacent to East-West Railroad

BldgType: Type of dwelling

       1Fam     Single-family Detached
       2FmCon   Two-family Conversion; originally built as one-family dwelling
       Duplx    Duplex
       TwnhsE   Townhouse End Unit
       TwnhsI   Townhouse Inside Unit

HouseStyle: Style of dwelling

       1Story   One story
       1.5Fin   One and one-half story: 2nd level finished
       1.5Unf   One and one-half story: 2nd level unfinished
       2Story   Two story
       2.5Fin   Two and one-half story: 2nd level finished
       2.5Unf   Two and one-half story: 2nd level unfinished
       SFoyer   Split Foyer
       SLvl     Split Level

OverallQual: Rates the overall material and finish of the house

       10       Very Excellent
```

```
       9        Excellent
       8        Very Good
       7        Good
       6        Above Average
       5        Average
       4        Below Average
       3        Fair
       2        Poor
       1        Very Poor

OverallCond: Rates the overall condition of the house

       10       Very Excellent
       9        Excellent
       8        Very Good
       7        Good
       6        Above Average
       5        Average
       4        Below Average
       3        Fair
       2        Poor
       1        Very Poor

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

       Flat     Flat
       Gable    Gable
       Gambrel  Gabrel (Barn)
       Hip      Hip
       Mansard  Mansard
       Shed     Shed

RoofMatl: Roof material

       ClyTile  Clay or Tile
       CompShg  Standard (Composite) Shingle
       Membran  Membrane
       Metal    Metal
       Roll     Roll
       Tar&Grv  Gravel & Tar
       WdShake  Wood Shakes
```

```
       WdShngl  Wood Shingles

Exterior1st: Exterior covering on house

       AsbShng  Asbestos Shingles
       AsphShn  Asphalt Shingles
       BrkComm  Brick Common
       BrkFace  Brick Face
       CBlock   Cinder Block
       CemntBd  Cement Board
       HdBoard  Hard Board
       ImStucc  Imitation Stucco
       MetalSd  Metal Siding
       Other    Other
       Plywood  Plywood
       PreCast  PreCast
       Stone    Stone
       Stucco   Stucco
       VinylSd  Vinyl Siding
       Wd Sdng  Wood Siding
       WdShing  Wood Shingles

Exterior2nd: Exterior covering on house (if more than one material)

       AsbShng  Asbestos Shingles
       AsphShn  Asphalt Shingles
       BrkComm  Brick Common
       BrkFace  Brick Face
       CBlock   Cinder Block
       CemntBd  Cement Board
       HdBoard  Hard Board
       ImStucc  Imitation Stucco
       MetalSd  Metal Siding
       Other    Other
       Plywood  Plywood
       PreCast  PreCast
       Stone    Stone
       Stucco   Stucco
       VinylSd  Vinyl Siding
       Wd Sdng  Wood Siding
       WdShing  Wood Shingles

MasVnrType: Masonry veneer type

       BrkCmn   Brick Common
```

```
       BrkFace  Brick Face
       CBlock   Cinder Block
       None     None
       Stone    Stone

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

       Ex       Excellent
       Gd       Good
       TA       Average/Typical
       Fa       Fair
       Po       Poor

ExterCond: Evaluates the present condition of the material on the exterior

       Ex       Excellent
       Gd       Good
       TA       Average/Typical
       Fa       Fair
       Po       Poor

Foundation: Type of foundation

       BrkTil   Brick & Tile
       CBlock   Cinder Block
       PConc    Poured Contrete
       Slab     Slab
       Stone    Stone
       Wood     Wood

BsmtQual: Evaluates the height of the basement

       Ex       Excellent (100+ inches)
       Gd       Good (90-99 inches)
       TA       Typical (80-89 inches)
       Fa       Fair (70-79 inches)
       Po       Poor (<70 inches
       NA       No Basement

BsmtCond: Evaluates the general condition of the basement

       Ex       Excellent
       Gd       Good
```

```
       TA       Typical - slight dampness allowed
       Fa       Fair - dampness or some cracking or settling
       Po       Poor - Severe cracking, settling, or wetness
       NA       No Basement

BsmtExposure: Refers to walkout or garden level walls

       Gd       Good Exposure
       Av       Average Exposure (split levels or foyers typically score average or above)
       Mn       Mimimum Exposure
       No       No Exposure
       NA       No Basement

BsmtFinType1: Rating of basement finished area

       GLQ      Good Living Quarters
       ALQ      Average Living Quarters
       BLQ      Below Average Living Quarters
       Rec      Average Rec Room
       LwQ      Low Quality
       Unf      Unfinshed
       NA       No Basement

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

       GLQ      Good Living Quarters
       ALQ      Average Living Quarters
       BLQ      Below Average Living Quarters
       Rec      Average Rec Room
       LwQ      Low Quality
       Unf      Unfinshed
       NA       No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

       Floor    Floor Furnace
       GasA     Gas forced warm air furnace
```

```
       GasW     Gas hot water or steam heat
       Grav     Gravity furnace
       OthW     Hot water or steam heat other than gas
       Wall     Wall furnace

HeatingQC: Heating quality and condition

       Ex       Excellent
       Gd       Good
       TA       Average/Typical
       Fa       Fair
       Po       Poor

CentralAir: Central air conditioning

       N        No
       Y        Yes

Electrical: Electrical system

       SBrkr    Standard Circuit Breakers & Romex
       FuseA    Fuse Box over 60 AMP and all Romex wiring (Average)
       FuseF    60 AMP Fuse Box and mostly Romex wiring (Fair)
       FuseP    60 AMP Fuse Box and mostly knob & tube wiring (poor)
       Mix      Mixed

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade
```

KitchenQual: Kitchen quality

Ex	Excellent
Gd	Good
TA	Typical/Average
Fa	Fair
Po	Poor

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

Typ	Typical Functionality
Min1	Minor Deductions 1
Min2	Minor Deductions 2
Mod	Moderate Deductions
Maj1	Major Deductions 1
Maj2	Major Deductions 2
Sev	Severely Damaged
Sal	Salvage only

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

Ex	Excellent - Exceptional Masonry Fireplace
Gd	Good - Masonry Fireplace in main level
TA	Average - Prefabricated Fireplace in main living area or Masonry Fireplace in basement
Fa	Fair - Prefabricated Fireplace in basement
Po	Poor - Ben Franklin Stove
NA	No Fireplace

GarageType: Garage location

2Types	More than one type of garage
Attchd	Attached to home
Basment	Basement Garage
BuiltIn	Built-In (Garage part of house - typically has room above garage)
CarPort	Car Port
Detchd	Detached from home
NA	No Garage

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

Fin	Finished
RFn	Rough Finished
Unf	Unfinished
NA	No Garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

Ex	Excellent
Gd	Good
TA	Typical/Average
Fa	Fair
Po	Poor
NA	No Garage

GarageCond: Garage condition

Ex	Excellent
Gd	Good
TA	Typical/Average
Fa	Fair
Po	Poor
NA	No Garage

PavedDrive: Paved driveway

Y	Paved
P	Partial Pavement
N	Dirt/Gravel

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
NA	No Pool

Fence: Fence quality

GdPrv	Good Privacy
MnPrv	Minimum Privacy
GdWo	Good Wood
MnWw	Minimum Wood/Wire
NA	No Fence

MiscFeature: Miscellaneous feature not covered in other categories

Elev	Elevator
Gar2	2nd Garage (if not described in garage section)
Othr	Other
Shed	Shed (over 100 SF)
TenC	Tennis Court
NA	None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

WD	Warranty Deed - Conventional
CWD	Warranty Deed - Cash
VWD	Warranty Deed - VA Loan
New	Home just constructed and sold
COD	Court Officer Deed/Estate
Con	Contract 15% Down payment regular terms
ConLw	Contract Low Down payment and low interest
ConLI	Contract Low Interest
ConLD	Contract Low Down
Oth	Other

```
SaleCondition: Condition of sale

       Normal   Normal Sale
       Abnorml  Abnormal Sale -  trade, foreclosure, short sale
       AdjLand  Adjoining Land Purchase
       Alloca   Allocation - two linked properties with separate deeds, typically condo with a garage unit
       Family   Sale between family members
       Partial  Home was not completed when last assessed (associated with New Homes)
```

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
```

```
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
59  GarageYrBlt    1379 non-null   float64
60  GarageFinish   1379 non-null   object
61  GarageCars     1460 non-null   int64
62  GarageArea     1460 non-null   int64
63  GarageQual     1379 non-null   object
64  GarageCond     1379 non-null   object
65  PavedDrive     1460 non-null   object
66  WoodDeckSF     1460 non-null   int64
67  OpenPorchSF    1460 non-null   int64
68  EnclosedPorch  1460 non-null   int64
69  3SsnPorch      1460 non-null   int64
70  ScreenPorch    1460 non-null   int64
```

```
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

2. Clean and Prepare data for model

2.1. Drop Columns with lots of nulls

In [11]:
```
# Drop ID column which is not a feature
df.drop(['Id'],axis=1,inplace=True)
```

In [12]:
```
# Drop of columns with lots of nulls
df.drop(['PoolQC','Fence','MiscFeature', 'Alley'],axis=1,inplace=True)
```

2.2. Filling Missing Values

In [13]:
```
# filling missing categorical columns with mode
## We have chosen to fill these features with mode instead of deleting all the missing values because doing that reduces
df['BsmtCond']=df['BsmtCond'].fillna(df['BsmtCond'].mode()[0])
df['BsmtQual']=df['BsmtQual'].fillna(df['BsmtQual'].mode()[0])
df['FireplaceQu']=df['FireplaceQu'].fillna(df['FireplaceQu'].mode()[0])
df['GarageType']=df['GarageType'].fillna(df['GarageType'].mode()[0])
df['GarageFinish']=df['GarageFinish'].fillna(df['GarageFinish'].mode()[0])
df['GarageQual']=df['GarageQual'].fillna(df['GarageQual'].mode()[0])
df['GarageCond']=df['GarageCond'].fillna(df['GarageCond'].mode()[0])
df['GarageYrBlt']=df['GarageYrBlt'].fillna(df['GarageYrBlt'].mode()[0])
df['MasVnrType']=df['MasVnrType'].fillna(df['MasVnrType'].mode()[0])
df['MasVnrArea']=df['MasVnrArea'].fillna(df['MasVnrArea'].mode()[0])
```

```
df['BsmtExposure']=df['BsmtExposure'].fillna(df['BsmtExposure'].mode()[0])
df['BsmtFinType2']=df['BsmtFinType2'].fillna(df['BsmtFinType2'].mode()[0])
```

In [14]:
```
# filling missing continuous columns with mean
df['LotFrontage']=df['LotFrontage'].fillna(df['LotFrontage'].mean())
df.dropna()
```

Out[14]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	EnclosedPorch	3SsnP
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	...	0	
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	...	0	
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	...	272	
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1456	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1457	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1458	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	112	
1459	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	

1422 rows × 76 columns

2.3. Remove Remaining Nulls

In [15]:
```
df.dropna(inplace=True)
```

In [16]:
```
df.shape
```

Out[16]:
```
(1422, 76)
```

2.4. Handle Categorical Data

In [17]:
```
# We need to adress categorical data by transform each feature into dummies variables. We used hotencoding to do this
df_text = df.select_dtypes(include=object)
one_hot = pd.get_dummies(df_text)
df_join = df.join(one_hot)
df_join.head()
```

Out[17]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	SaleType_ConLw	SaleTyp
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	...	0	
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	...	0	
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	...	0	
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	...	0	

5 rows × 311 columns

In [18]:
```
df = df_join.select_dtypes(exclude=['object'])
df.head()
```

Out[18]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	SaleType
0	60	65.0	8450	7	5	2003	2003	196.0	706	0	...	
1	20	80.0	9600	6	8	1976	1976	0.0	978	0	...	
2	60	68.0	11250	7	5	2001	2002	162.0	486	0	...	
3	70	60.0	9550	7	5	1915	1970	0.0	216	0	...	
4	60	84.0	14260	8	5	2000	2000	350.0	655	0	...	

5 rows × 272 columns

In [19]:
```
df.columns
```

Out[19]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       ...
       'SaleType_ConLw', 'SaleType_New', 'SaleType_Oth', 'SaleType_WD',
       'SaleCondition_Abnorml', 'SaleCondition_AdjLand',
       'SaleCondition_Alloca', 'SaleCondition_Family', 'SaleCondition_Normal',
       'SaleCondition_Partial'],
      dtype='object', length=272)
```

In [20]:
```
df.shape
```

Out[20]:
```
(1422, 272)
```

3. Select Relevant Features to build the model

3.1. Analysis of Features correlation with SalePrice

In [21]:
```
# We delete features having a correlation > 80% with another feature

corr = df.corr()
columns = np.full((corr.shape[0],), True, dtype=bool)
for i in range(corr.shape[0]):
    for j in range(i+1, corr.shape[0]):
        if corr.iloc[i,j] >= 0.80:
            if columns[j]:
                columns[j] = False
selected_columns = df.columns[columns]
df = df[selected_columns]
```

In [22]:
```
df.shape
```

Out[22]:
```
(1422, 257)
```

In [23]:
```
# Analysis of features correlations with SalePrice and selection of features with a correlation > 0.40
corr_matrix = df.corr()
corr_analysis = corr_matrix["SalePrice"].sort_values(ascending=False)
```

```
relevant_features = corr_analysis[corr_analysis>0.40]
relevant_features
```

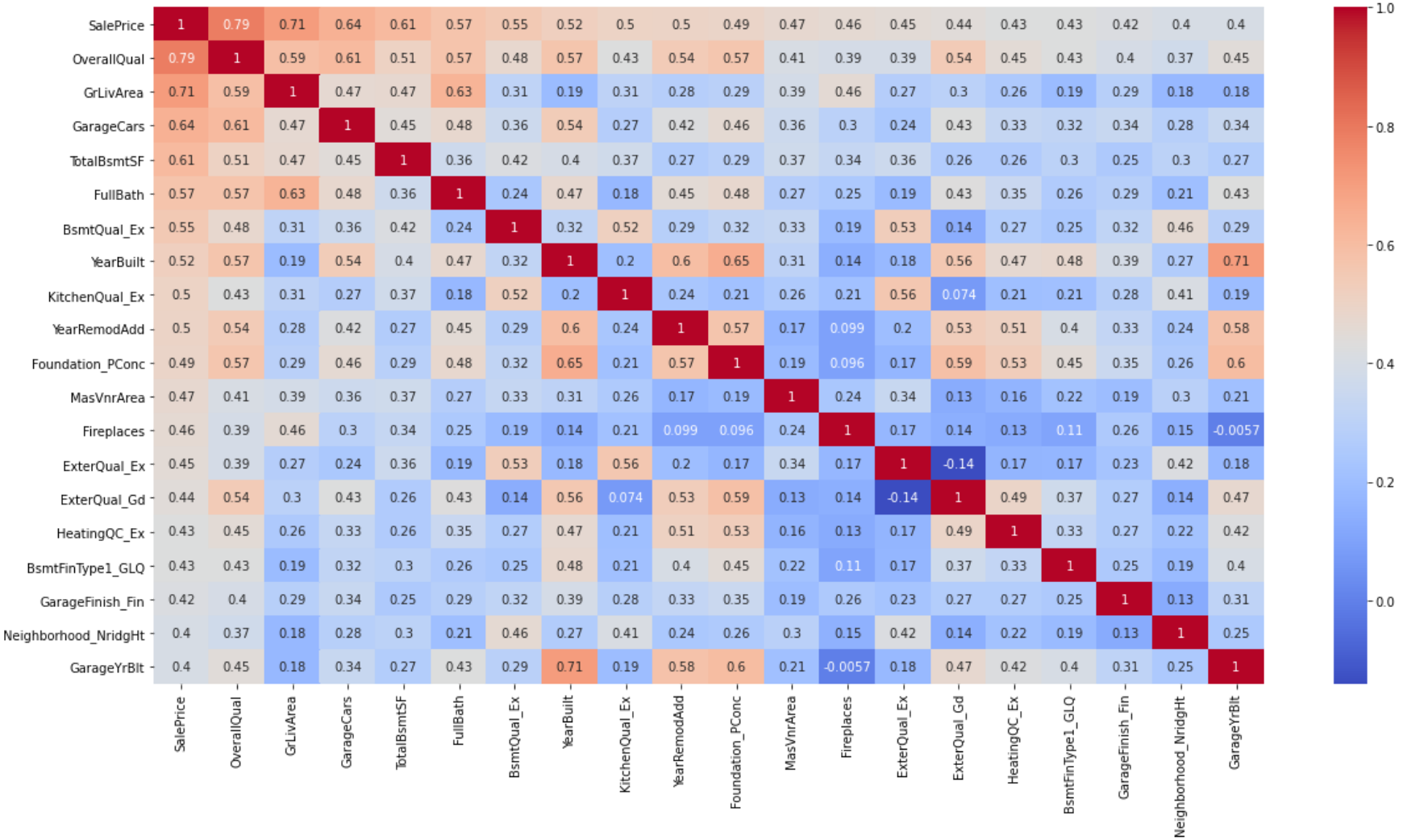
Out[23]:
```
SalePrice               1.000000
OverallQual             0.787985
GrLivArea               0.709303
GarageCars              0.643039
TotalBsmtSF             0.610085
FullBath                0.573755
BsmtQual_Ex             0.553768
YearBuilt               0.519014
KitchenQual_Ex          0.504655
YearRemodAdd            0.500512
Foundation_PConc        0.492300
MasVnrArea              0.470117
Fireplaces              0.461108
ExterQual_Ex            0.452715
ExterQual_Gd            0.443755
HeatingQC_Ex            0.428411
BsmtFinType1_GLQ        0.426911
GarageFinish_Fin        0.417049
Neighborhood_NridgHt    0.402007
GarageYrBlt             0.400598
Name: SalePrice, dtype: float64
```

In [24]:
```
relevant_features.index
```

Out[24]:
```
Index(['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF',
       'FullBath', 'BsmtQual_Ex', 'YearBuilt', 'KitchenQual_Ex',
       'YearRemodAdd', 'Foundation_PConc', 'MasVnrArea', 'Fireplaces',
       'ExterQual_Ex', 'ExterQual_Gd', 'HeatingQC_Ex', 'BsmtFinType1_GLQ',
       'GarageFinish_Fin', 'Neighborhood_NridgHt', 'GarageYrBlt'],
      dtype='object')
```

In [25]:
```
# How does the correlation matrix looks like
corr = df[relevant_features.index].corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[25]:
```
<AxesSubplot:>
```



In [26]:

```
X_corr = df[relevant_features.index]
X_corr=X_corr.drop('SalePrice',axis=1)
```

3.2. Analysis of Features P-values

In [27]:

```
#We will remove thos features having P-values higher that 5% as this means they are not statistically significant
import pandas as pd
import numpy as np
from sklearn import datasets, linear_model
```

```python
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from scipy import stats

X = X_corr
y = df['SalePrice']

X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.814
Model:                            OLS   Adj. R-squared:                  0.811
Method:                 Least Squares   F-statistic:                     322.1
Date:                Mon, 18 Oct 2021   Prob (F-statistic):               0.00
Time:                        11:12:06   Log-Likelihood:                -16866.
No. Observations:                1422   AIC:                         3.377e+04
Df Residuals:                    1402   BIC:                         3.388e+04
Df Model:                          19
Covariance Type:            nonrobust
==============================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                -7.592e+05    1.5e+05     -5.049      0.000   -1.05e+06   -4.64e+05
OverallQual           1.156e+04   1241.448      9.308      0.000    9119.749     1.4e+04
GrLivArea               46.5397      2.933     15.865      0.000      40.785      52.294
GarageCars            1.217e+04   1716.164      7.093      0.000    8806.840    1.55e+04
TotalBsmtSF             17.4625      2.931      5.959      0.000      11.714      23.211
FullBath             -1255.7330   2530.860     -0.496      0.620   -6220.413    3708.947
BsmtQual_Ex           3.033e+04   4477.037      6.774      0.000    2.15e+04    3.91e+04
YearBuilt              178.1074     56.734      3.139      0.002      66.815     289.400
KitchenQual_Ex        2.669e+04   4770.875      5.593      0.000    1.73e+04     3.6e+04
YearRemodAdd           240.4933     64.858      3.708      0.000     113.264     367.722
Foundation_PConc     -2862.9831   2881.627     -0.994      0.321   -8515.749    2789.782
MasVnrArea              17.2780      6.029      2.866      0.004       5.452      29.104
Fireplaces            1.029e+04   1700.262      6.054      0.000    6958.345    1.36e+04
ExterQual_Ex          2.207e+04   7174.533      3.075      0.002    7991.129    3.61e+04
ExterQual_Gd          7167.6610   3052.458      2.348      0.019    1179.785    1.32e+04
HeatingQC_Ex          3885.0799   2340.554      1.660      0.097    -706.285    8476.444
BsmtFinType1_GLQ      1.302e+04   2402.548      5.420      0.000    8309.464    1.77e+04
GarageFinish_Fin      2749.5435   2485.241      1.106      0.269   -2125.647    7624.735
Neighborhood_NridgHt  1.131e+04   4907.480      2.305      0.021    1686.731    2.09e+04
```

```
GarageYrBlt            -44.8198     57.468     -0.780      0.436    -157.553      67.914
==============================================================================
Omnibus:                      731.858   Durbin-Watson:                   1.979
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            97116.550
Skew:                          -1.368   Prob(JB):                         0.00
Kurtosis:                      43.393   Cond. No.                     6.43e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.43e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

3.3. Result - Selected Features

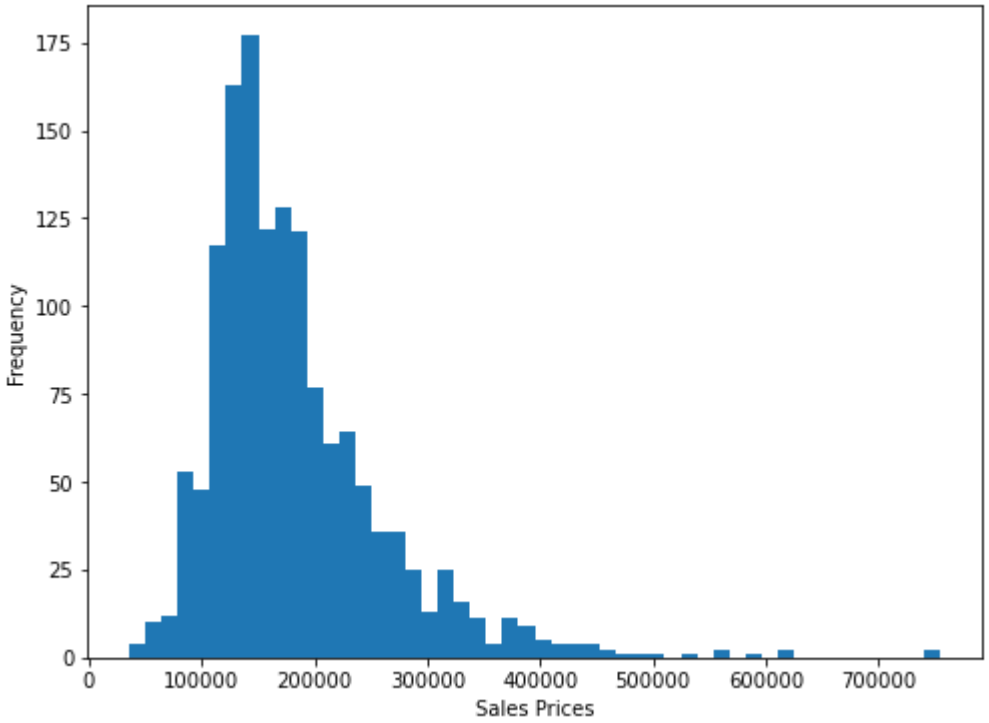
In [28]:
```
selected_features=X.columns[(est2.pvalues<0.05).drop('const')]
selected_features
```

Out[28]:
```
Index(['OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'BsmtQual_Ex',
       'YearBuilt', 'KitchenQual_Ex', 'YearRemodAdd', 'MasVnrArea',
       'Fireplaces', 'ExterQual_Ex', 'ExterQual_Gd', 'BsmtFinType1_GLQ',
       'Neighborhood_NridgHt'],
      dtype='object')
```

4. Visualise the data and look for outliers to remove

4.1. Label

In [29]:
```
# Study of the distribution of the column we want to predict: SalePrice
plt.figure(figsize=(8, 6))
plt.hist(df.SalePrice,bins=50);
plt.xlabel('Sales Prices')
plt.ylabel('Frequency');
```



In [30]:
```
# we drop outliers
df = df[df.SalePrice<600000]
```

In [31]:
```
df['SalePrice'].describe()
```

Out[31]:
```
count      1418.000000
mean     181475.057828
std       74838.466681
min       34900.000000
25%      131500.000000
50%      164995.000000
75%      215000.000000
max      582933.000000
Name: SalePrice, dtype: float64
```

4.2. Features

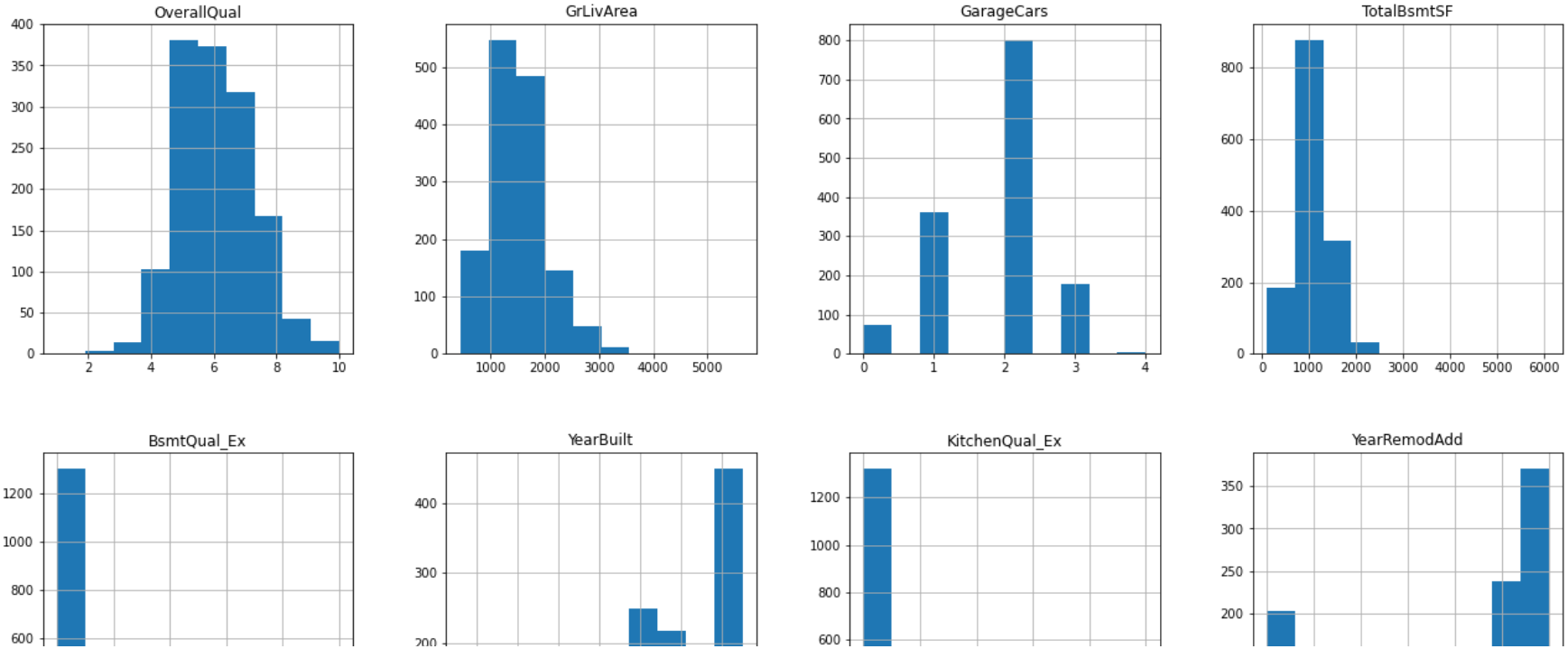
In [32]:
```
df[selected_features].describe()
```

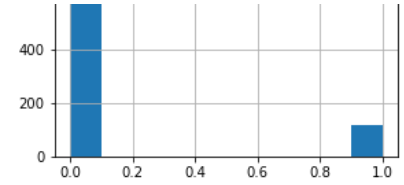
Out[32]:

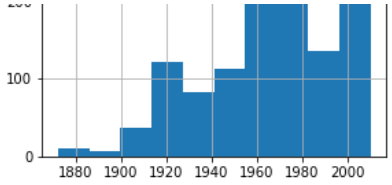
	OverallQual	GrLivArea	GarageCars	TotalBsmtSF	BsmtQual_Ex	YearBuilt	KitchenQual_Ex	YearRemodAdd	MasVnrArea	Fireplac
count	1418.000000	1418.000000	1418.000000	1418.000000	1418.000000	1418.000000	1418.000000	1418.000000	1418.000000	1418.0000
mean	6.136812	1515.299718	1.772920	1082.061354	0.082511	1971.550071	0.068406	1985.267278	102.898449	0.6191
std	1.347639	512.338546	0.742348	404.649896	0.275238	30.386721	0.252531	20.463536	175.890490	0.6432
min	1.000000	438.000000	0.000000	105.000000	0.000000	1872.000000	0.000000	1950.000000	0.000000	0.0000
25%	5.000000	1137.250000	1.000000	810.250000	0.000000	1954.000000	0.000000	1968.000000	0.000000	0.0000
50%	6.000000	1467.000000	2.000000	1004.000000	0.000000	1973.000000	0.000000	1994.000000	0.000000	1.0000
75%	7.000000	1779.000000	2.000000	1305.500000	0.000000	2001.000000	0.000000	2004.000000	166.750000	1.0000
max	10.000000	5642.000000	4.000000	6110.000000	1.000000	2010.000000	1.000000	2010.000000	1600.000000	3.0000

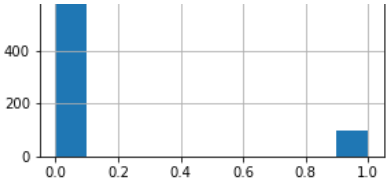
In [33]:

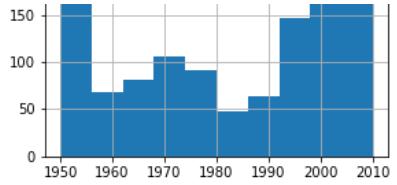
```
# Histograms of Selected Features
df[selected_features].hist(figsize=(22, 24));
```

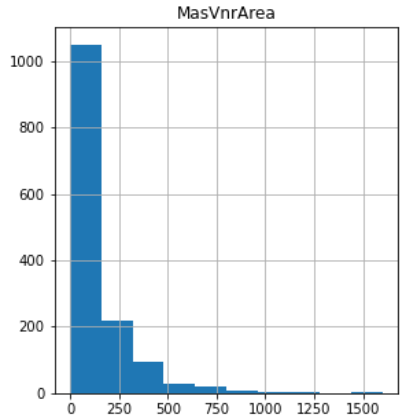


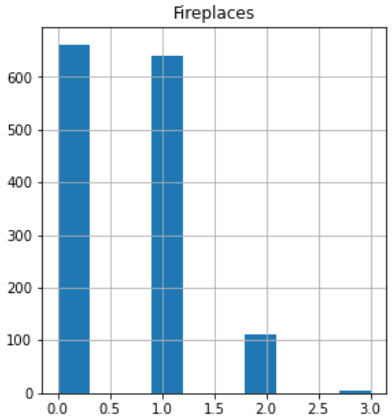


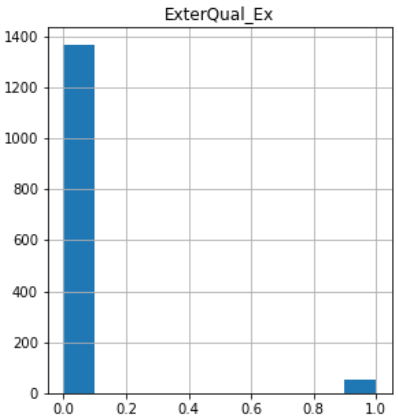


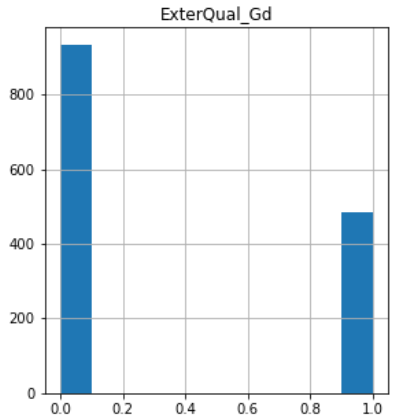


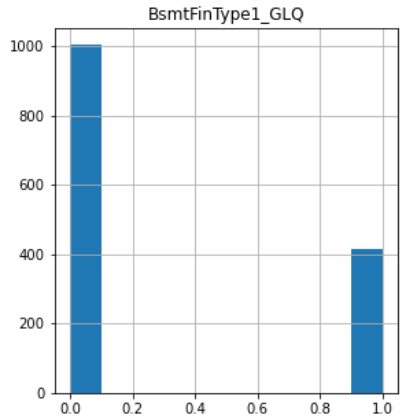


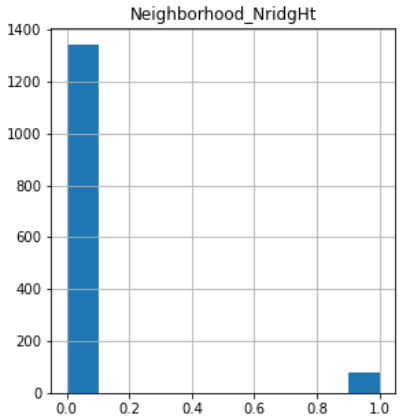










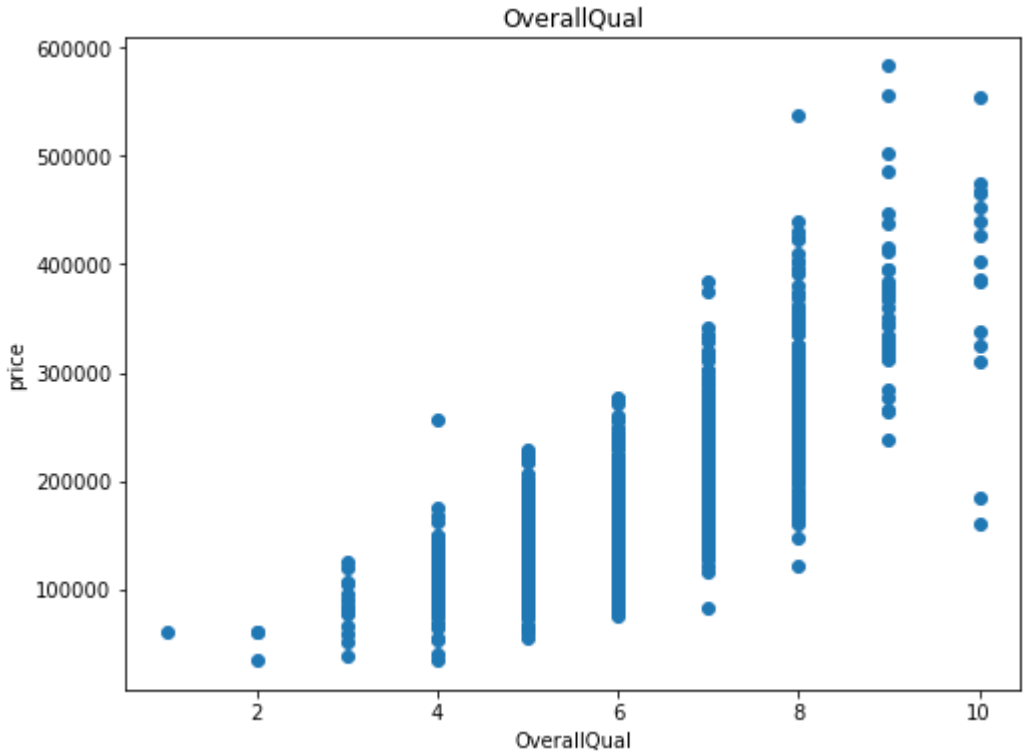


In [34]:

```
def plotFeatureName(featureName):
    plt.figure(figsize=(8,6))
    plt.scatter(df[featureName], df['SalePrice'])
    plt.title(featureName)
    plt.xlabel(featureName)
    plt.ylabel('price')
```

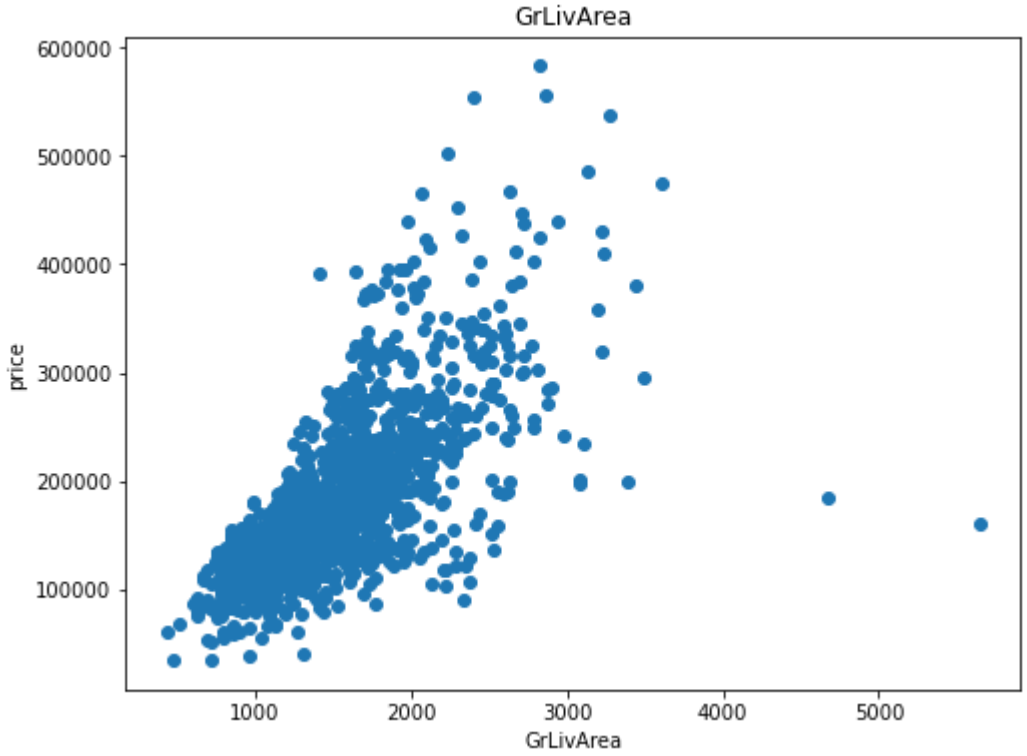
In [35]:

```
plotFeatureName('OverallQual')
```



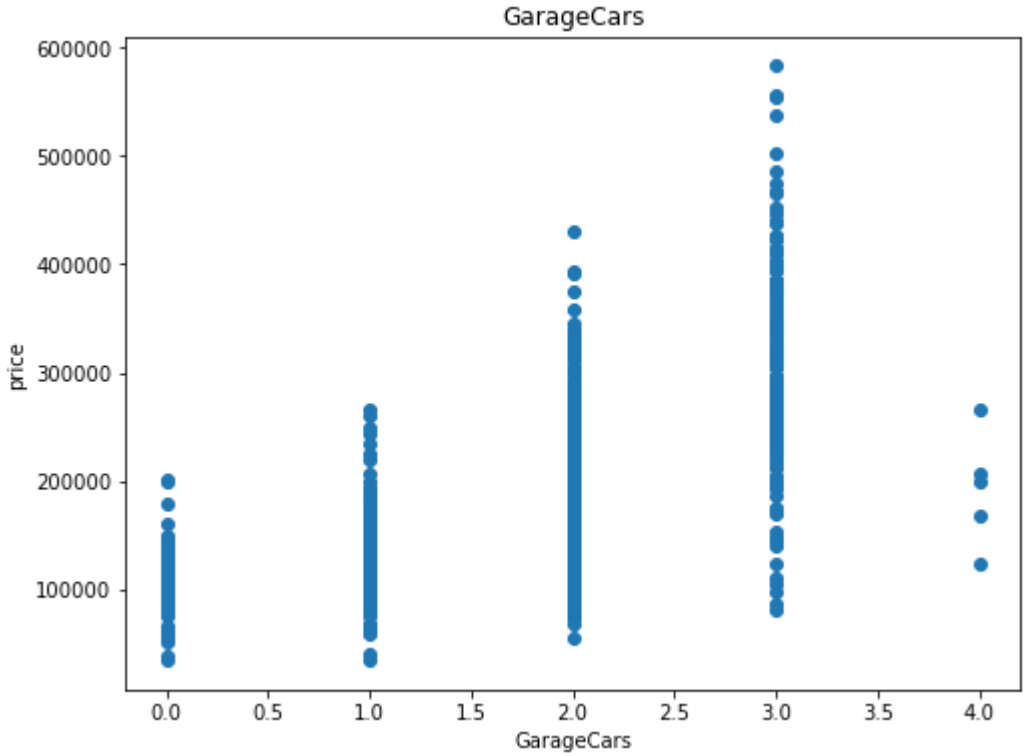
In [36]:

```
plotFeatureName('GrLivArea')
```

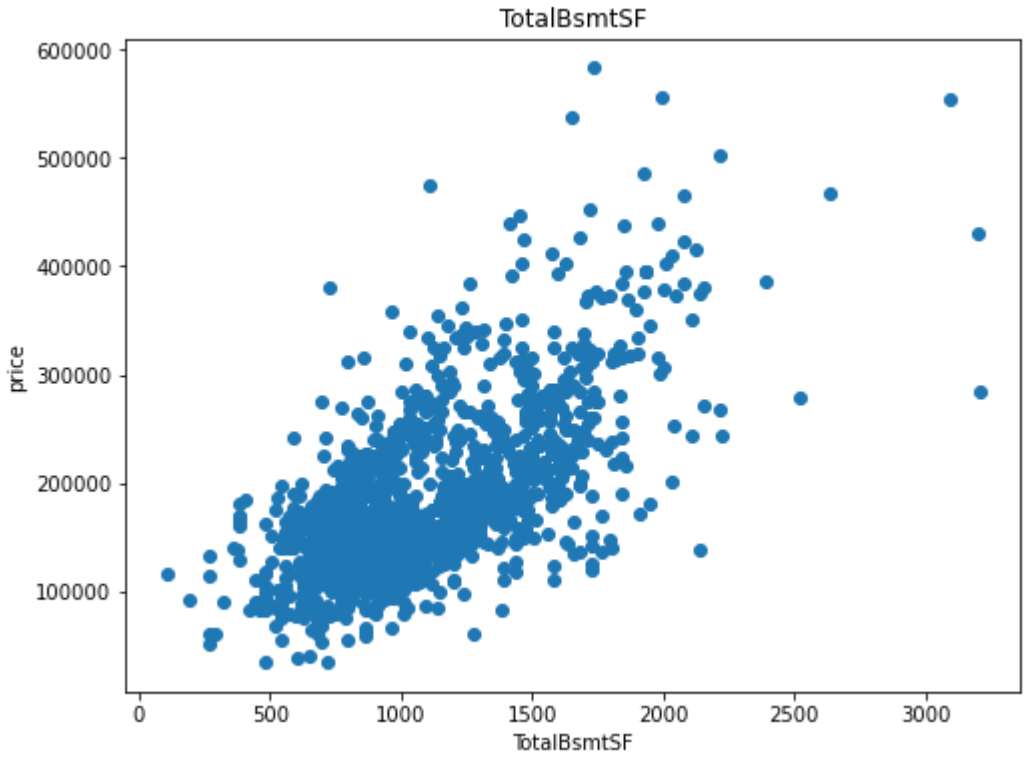


In [37]:
```
#Remove Outliers
df = df[df['GrLivArea']<4000]
```

In [38]:
```
plotFeatureName('GarageCars')
```

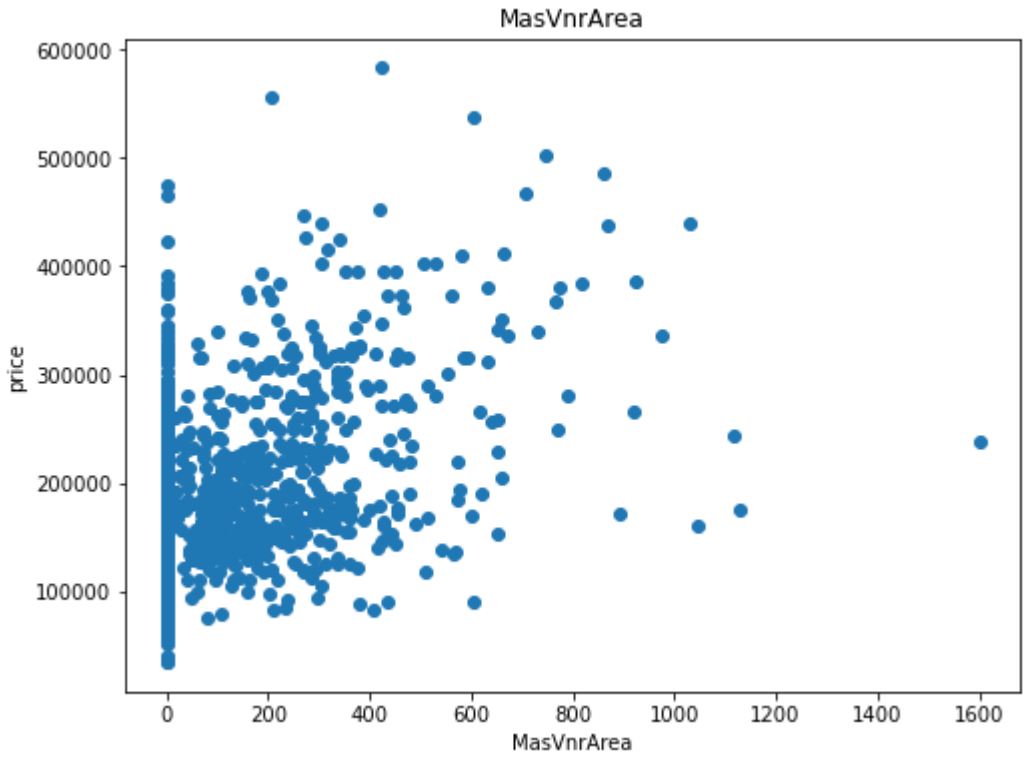


In [39]:
```
plotFeatureName('TotalBsmtSF')
```



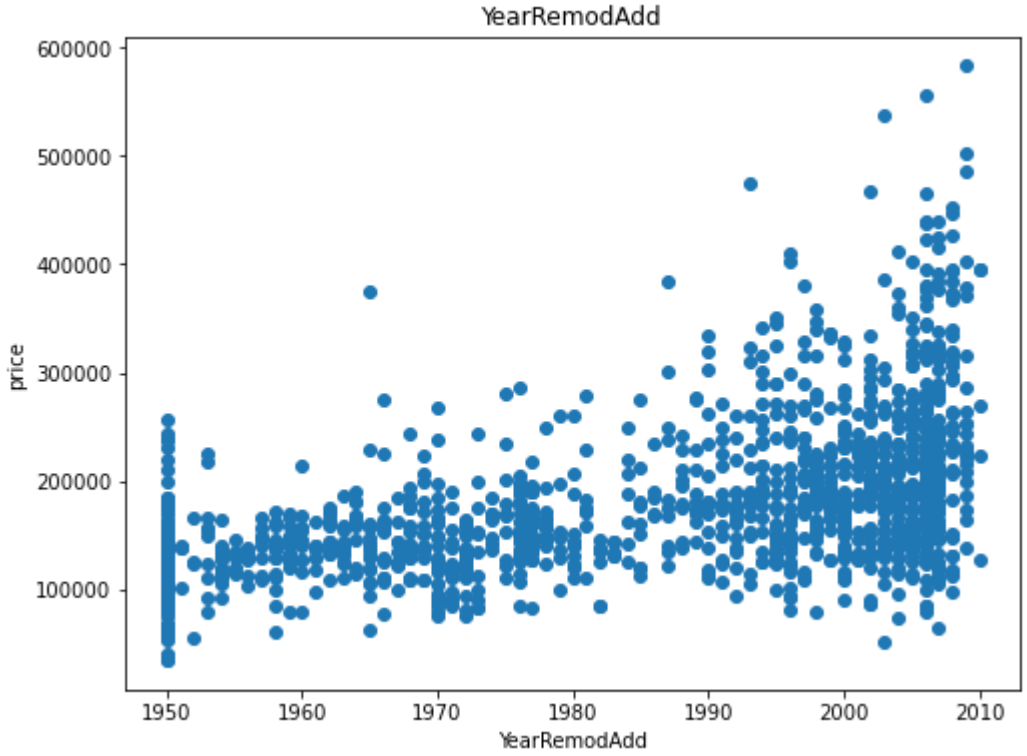
In [40]:
```
#Remove Outliers
df = df[df['TotalBsmtSF']<3000]
```

In [41]:
```
plotFeatureName('MasVnrArea')
```



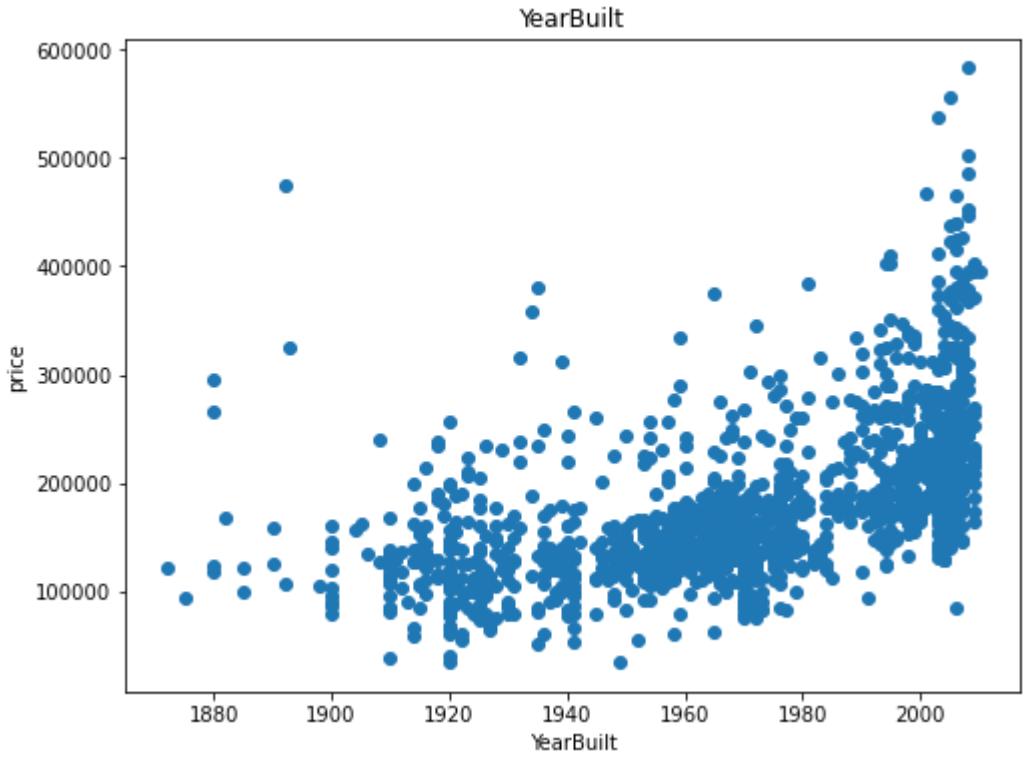
In [42]:
```
#Remove Outliers
df = df[df['MasVnrArea']<1400]
```

In [43]:
```
plotFeatureName('YearRemodAdd')
```



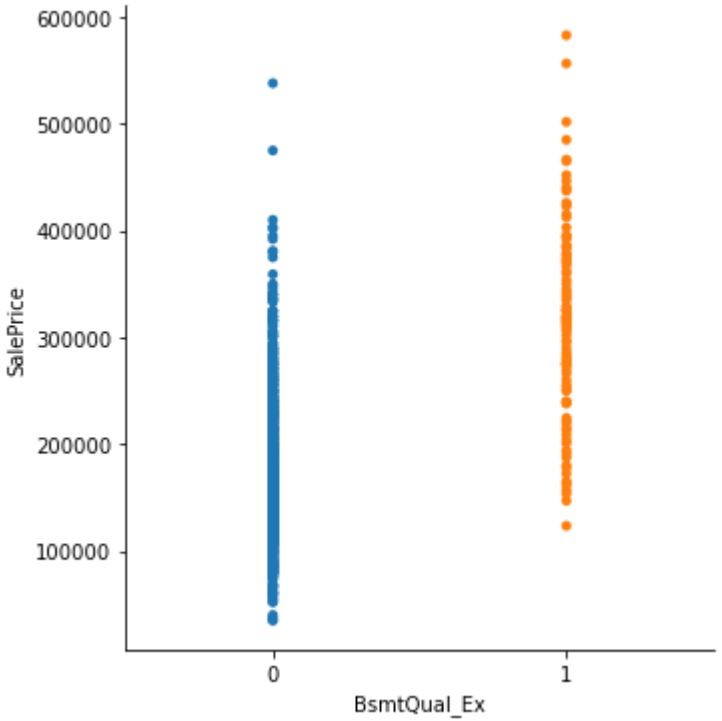
In [44]:

```
plotFeatureName('YearBuilt')
```



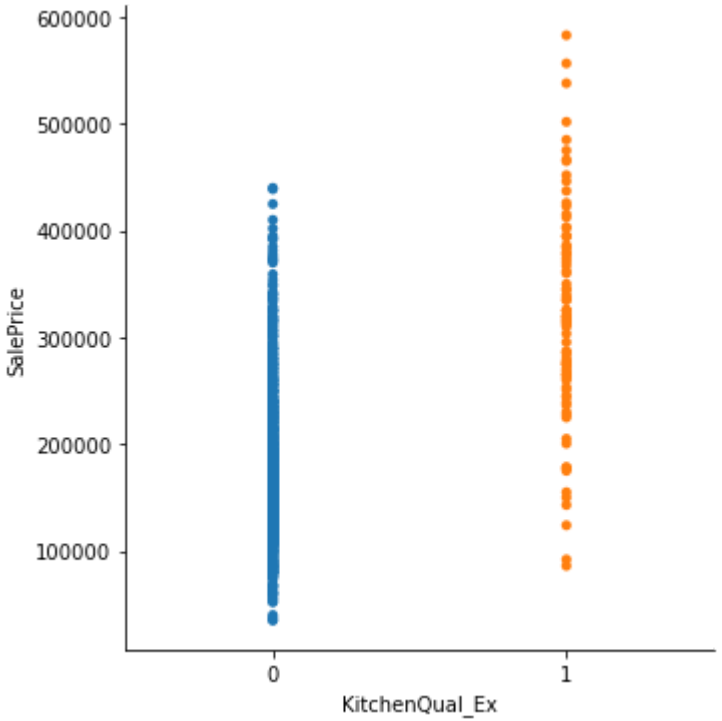
In [45]:
```
sns.catplot(x="BsmtQual_Ex", y="SalePrice", jitter=False, data=df)
```

Out[45]:
```
<seaborn.axisgrid.FacetGrid at 0x23bc39a94c0>
```



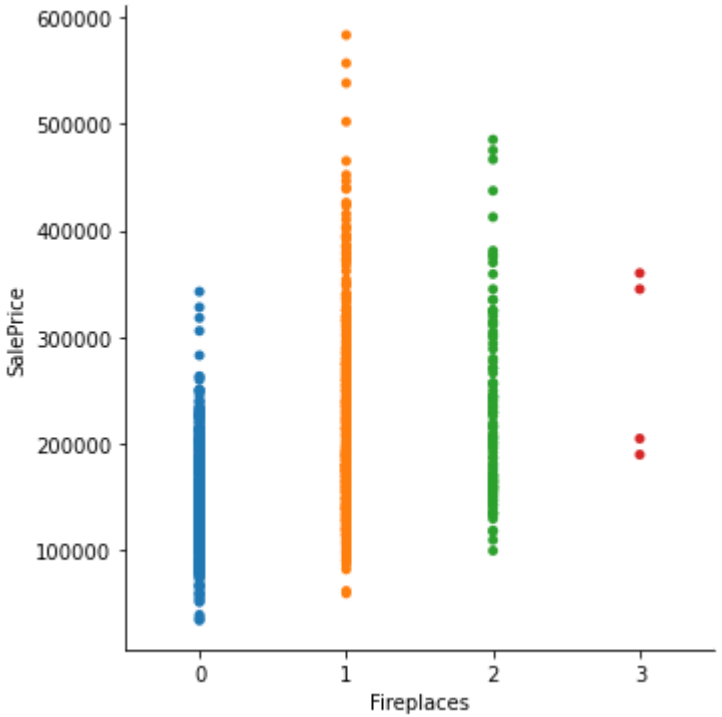
In [46]:
```
sns.catplot(x="KitchenQual_Ex", y="SalePrice", jitter=False, data=df)
```

Out[46]:
```
<seaborn.axisgrid.FacetGrid at 0x23bc3a26340>
```



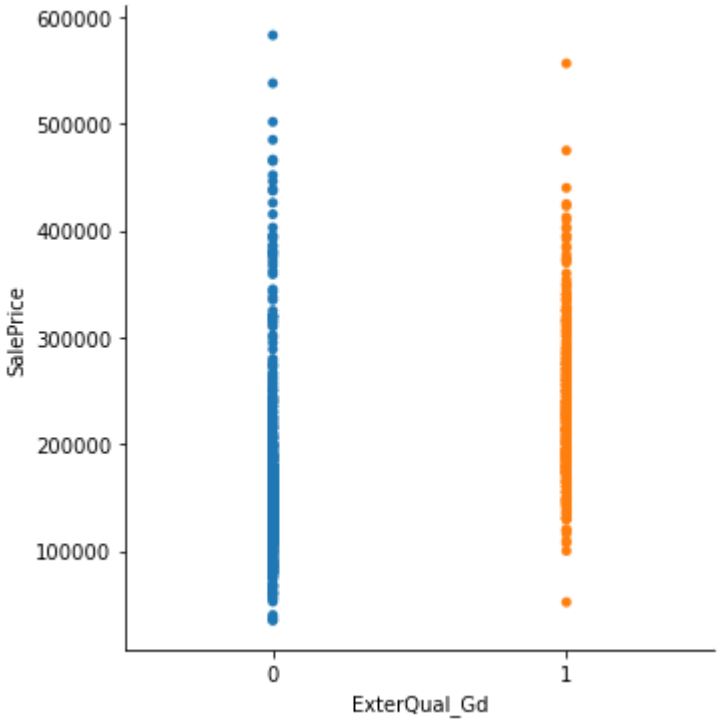
In [47]:
```
sns.catplot(x="Fireplaces", y="SalePrice", jitter=False, data=df)
```

Out[47]:
```
<seaborn.axisgrid.FacetGrid at 0x23bc394bac0>
```



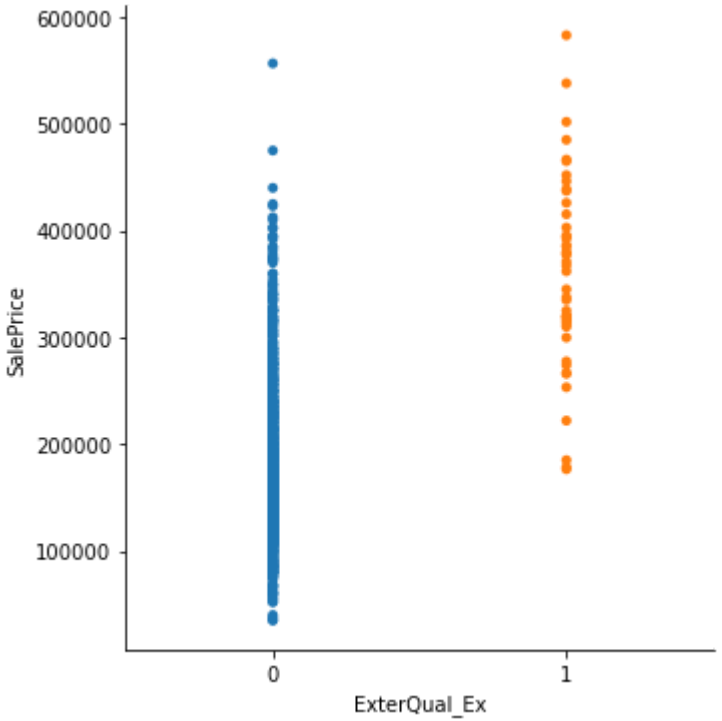
In [48]:
```
sns.catplot(x="ExterQual_Gd", y="SalePrice", jitter=False, data=df)
```

Out[48]:
```
<seaborn.axisgrid.FacetGrid at 0x23bc3a5af10>
```



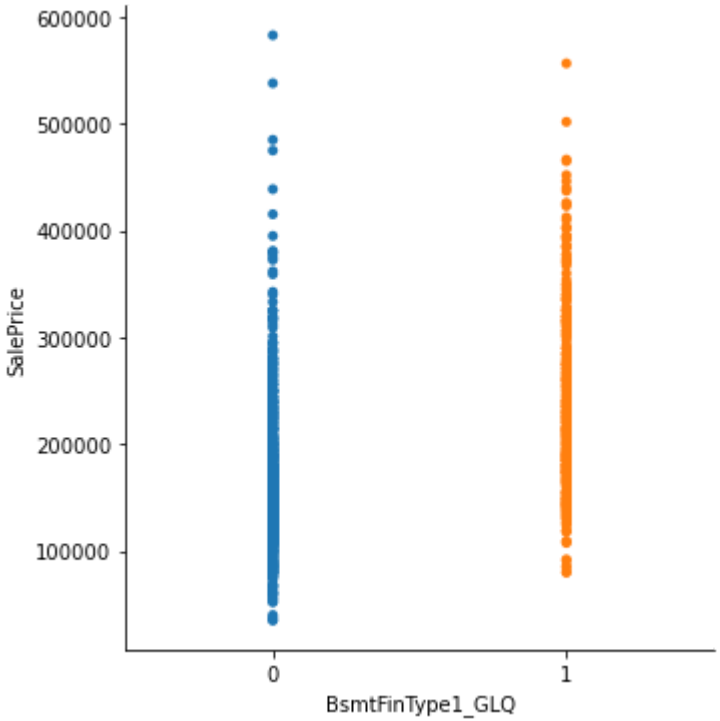
In [49]:
```
sns.catplot(x="ExterQual_Ex", y="SalePrice", jitter=False, data=df)
```

Out[49]:
```
<seaborn.axisgrid.FacetGrid at 0x23bc3a6c7c0>
```



In [50]:
```
sns.catplot(x="BsmtFinType1_GLQ", y="SalePrice", jitter=False, data=df)
```

Out[50]:
```
<seaborn.axisgrid.FacetGrid at 0x23bc323fdf0>
```



5. Linear Regression Model

In [51]:
```
X = df[selected_features]
```

In [52]:
```
y = df['SalePrice']
```

In [53]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.33, random_state = 999)

print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)
```

```
(946, 14)
```

```
(466, 14)
(946,)
(466,)
```

In [54]:
```
# Load linear regression model
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
```

In [55]:
```
# Fit to the training data
lm.fit(X_train, y_train)
```

Out[55]:
```
LinearRegression()
```

In [56]:
```
print(pd.Series(lm.coef_, index = X.columns))
```

```
OverallQual              9512.960943
GrLivArea                  50.366693
GarageCars               7940.565914
TotalBsmtSF                31.666561
BsmtQual_Ex              20423.451899
YearBuilt                  188.174504
KitchenQual_Ex           27576.205858
YearRemodAdd               251.907573
MasVnrArea                  11.327168
Fireplaces                9899.751313
ExterQual_Ex             45082.824115
ExterQual_Gd              8227.222280
BsmtFinType1_GLQ         13561.968154
Neighborhood_NridgHt      2167.276279
dtype: float64
```

In [57]:
```
lm.intercept_
```

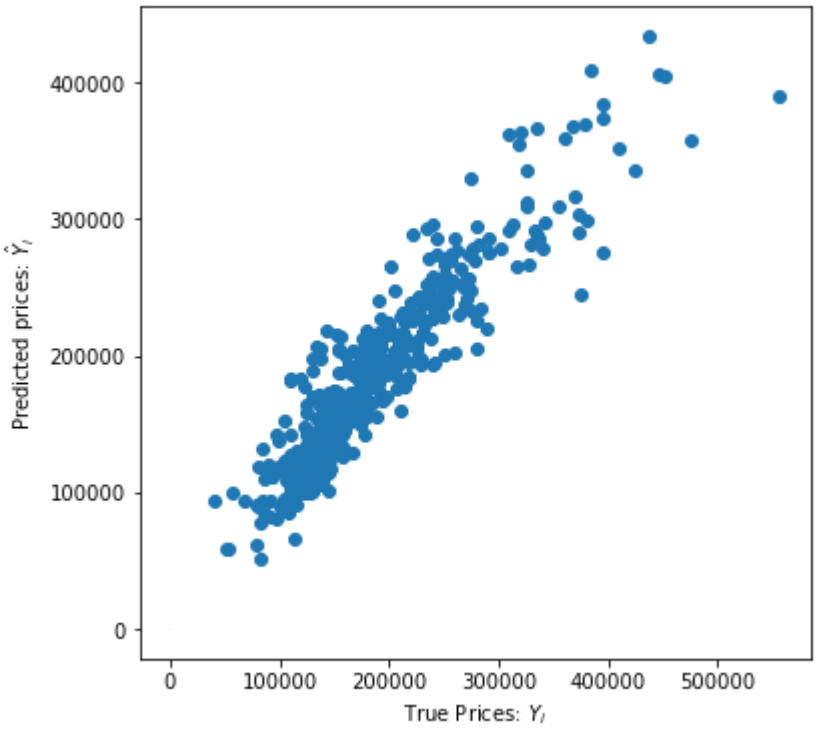
Out[57]:
```
-892008.158979196
```

In [58]:
```
y_pred = lm.predict(X_test)
```

In [59]:
```
plt.figure(figsize=(6,6))
```

```
plt.scatter(y_test, y_pred)
plt.xlabel("True Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.plot([0, 50], [0, 50], '--k')
```

Out[59]:
```
[<matplotlib.lines.Line2D at 0x23bc321d9a0>]
```



In [60]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [61]:
```
# How well did it do on the training set
y_pred = lm.predict(X_train)

print("Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_train, y_pred)))
print('R squared: %.2f' % r2_score(y_train, y_pred))
```

```
Mean squared error: 26986.12
R squared: 0.87
```

In [62]:
```
# See how well it does on the test set
y_pred = lm.predict(X_test)

print("Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, y_pred)))
print('R squared: %.2f' % r2_score(y_test, y_pred))
```

```
Mean squared error: 28887.91
R squared: 0.85
```

6. Ridge & Lasso Regression

6.1. Ridge Regression

In [63]:
```
from sklearn.linear_model import Ridge
```

In [64]:
```
#Ridge regression coefficients
ridge_reg = Ridge(alpha=0.5)
ridge_reg.fit(X_train, y_train)
pred_train_ridge_reg= ridge_reg.predict(X_train)
print(pd.Series(ridge_reg.fit(X_train, y_train).coef_, index = X.columns))
```

```
OverallQual              9576.103043
GrLivArea                  50.404072
GarageCars               7928.191742
TotalBsmtSF                31.768249
BsmtQual_Ex              20443.909392
YearBuilt                  188.831826
KitchenQual_Ex           27517.139003
YearRemodAdd               253.572611
MasVnrArea                  11.491147
Fireplaces                9870.448268
ExterQual_Ex             43852.532750
ExterQual_Gd              7954.015800
BsmtFinType1_GLQ         13532.933062
Neighborhood_NridgHt      2362.695136
dtype: float64
```

In [65]:
```
# How well did it do on the training set
print("Mean squared error - Training set: %.2f" % np.sqrt(mean_squared_error(y_train, pred_train_ridge_reg)))
print('R squared - Training set: %.2f' % r2_score(y_train, pred_train_ridge_reg))
```

```
Mean squared error - Training set: 26986.68
R squared - Training set: 0.87
```

In [66]:
```
# How well it does on the test set
pred_test_ridge_reg= ridge_reg.predict(X_test)
print("Mean squared error - Test set:: %.2f" % np.sqrt(mean_squared_error(y_test, pred_test_ridge_reg)))
print('R squared - Test set: %.2f' % r2_score(y_test, pred_test_ridge_reg))
```

```
Mean squared error - Test set:: 28859.79
R squared - Test set: 0.85
```

6.2. Lasso Regression

In [67]:
```
from sklearn.linear_model import Lasso
```

In [68]:
```
#Lasso regression coefficients
lasso_reg = Lasso(alpha=0.01)
lasso_reg.fit(X_train, y_train)
pred_train_lasso= lasso_reg.predict(X_train)
print(pd.Series(lasso_reg.fit(X_train, y_train).coef_, index = X.columns))
```

```
OverallQual              9513.001985
GrLivArea                  50.366729
GarageCars               7940.539976
TotalBsmtSF                31.666653
BsmtQual_Ex              20423.404199
YearBuilt                  188.175523
KitchenQual_Ex           27576.146253
YearRemodAdd               251.908946
MasVnrArea                  11.327292
Fireplaces                9899.715270
ExterQual_Ex             45082.361052
ExterQual_Gd              8227.045692
BsmtFinType1_GLQ         13561.912694
Neighborhood_NridgHt      2167.135317
dtype: float64
```

In [69]:
```
# How well did it do on the training set
print("Mean squared error - Training set: %.2f" % np.sqrt(mean_squared_error(y_train, pred_train_lasso)))
print('R squared - Training set: %.2f' % r2_score(y_train, pred_train_lasso))
```

```
Mean squared error - Training set: 26986.12
R squared - Training set: 0.87
```

In [70]:
```
# How well it does on the test set
pred_test_lasso= lasso_reg.predict(X_test)
print("Mean squared error - Test set:: %.2f" % np.sqrt(mean_squared_error(y_test, pred_test_lasso)))
print('R squared - Test set: %.2f' % r2_score(y_test, pred_test_lasso))
```

```
Mean squared error - Test set:: 28887.90
R squared - Test set: 0.85
```

In [ ]:
```
#In general we the three models performed similarly. We ran the test under different scenarios, like dropping al NAs valu
#or without removing the high P-values features and in all cases we got an R-squared for the test sample above 0.80. We b
#for this data sample, given the features provided.
```

In [607…
```
################################################################################
```

Question 2: Titanic Classification

Team Members:

Marcel Santos de Carvalho, id 79083

Loris Baudry, id 79794

Alex Palacios, id 73713

Responsible for this notebook: Marcel Santos de Carvalho

In [608…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import matplotlib as mpl
%matplotlib inline
warnings.filterwarnings('ignore')
```

1. Load the data

In [613…
```
df = pd.read_csv('IntroToML_Coursework_one/titanic/train.csv')
```

2. Visualise and analyse the data

In [614…
```
type(df)
```

Out[614…
```
pandas.core.frame.DataFrame
```

In [615…
```
df.shape
```

Out[615…
```
(891, 12)
```

In [616…
```
df.columns
```

Out[616…
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [617…
```
df.head()
```

Out[617…

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [618…

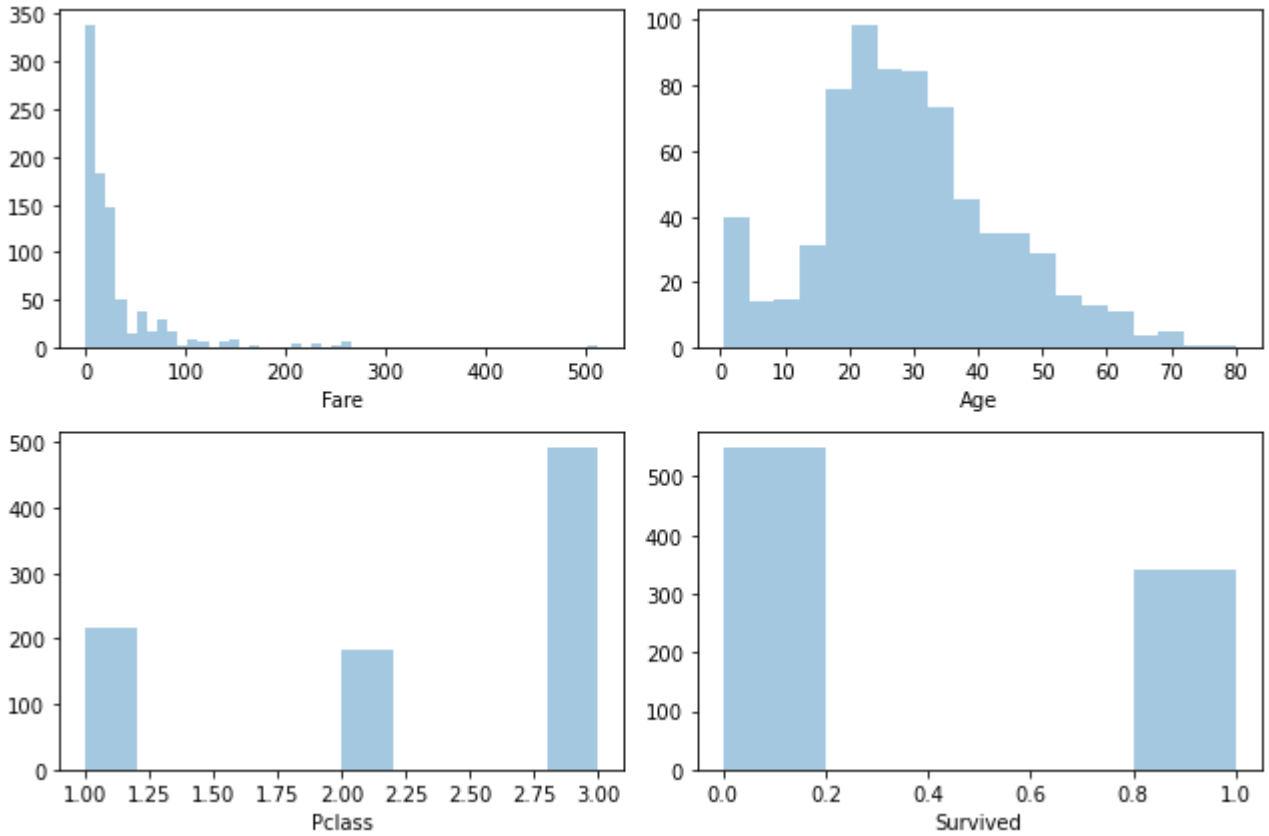
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

2.1 Histograms

In [619…

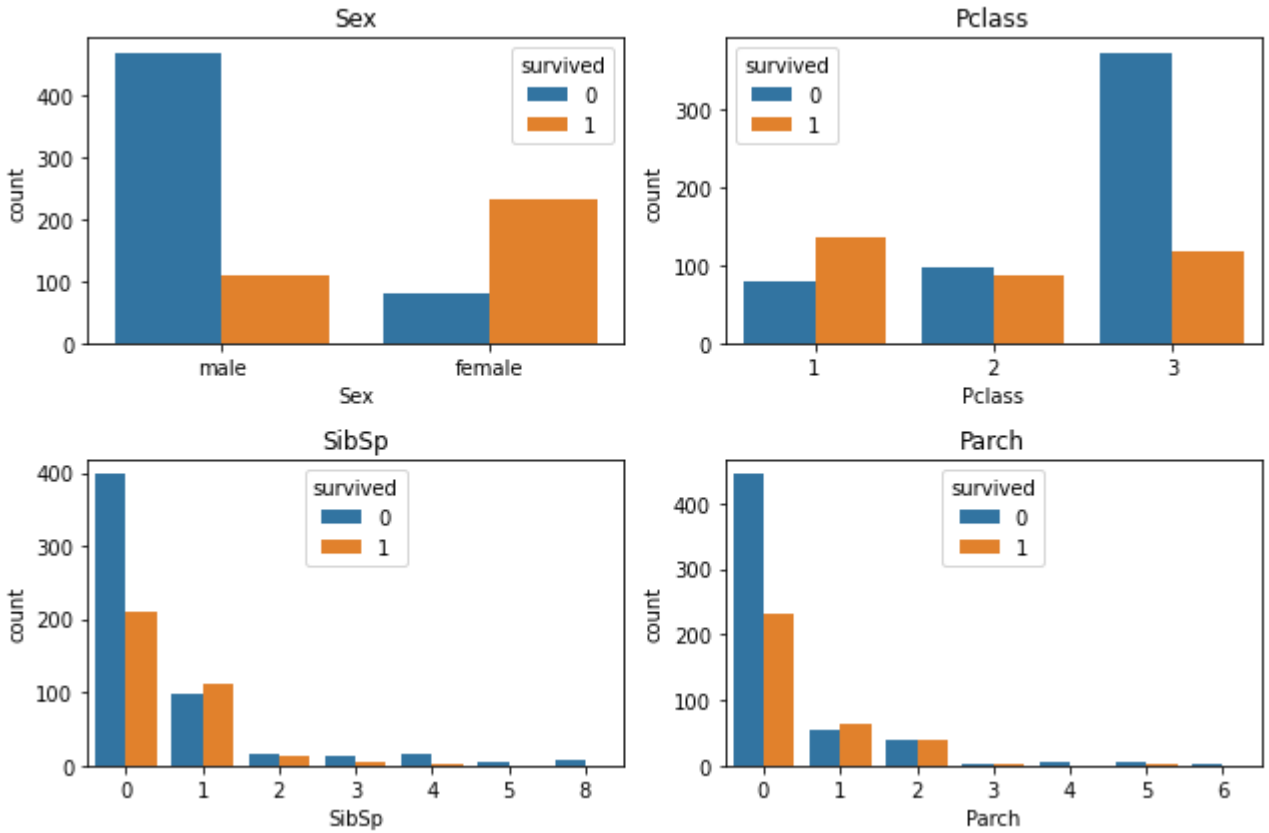
```
df_hist = df[['Fare', 'Age', 'Pclass', 'Survived']]

fig, ax = plt.subplots(ncols=2, nrows=2, figsize=(9, 6))
index=0
ax = ax.flatten()

for col, values in df_hist.items():
    sns.distplot(values, ax=ax[index], kde=False)
    index += 1
plt.tight_layout()
```



2.2 Survivors by feature

In [620…

```
cols = ['Sex', 'Pclass', 'SibSp', 'Parch']
nrows = 2
ncols = 2

fig, axs = plt.subplots(nrows, ncols, figsize = (9, 6))

for r in range(0, nrows):
    for c in range(0, ncols):
        i = r*ncols + c # i goes through the columns
        ax = axs[r][c] # positioning in subplots
        sns.countplot(df[cols[i]], hue=df['Survived'], ax=ax)
        ax.set_title(cols[i])
        ax.legend(title='survived')
plt.tight_layout()
```



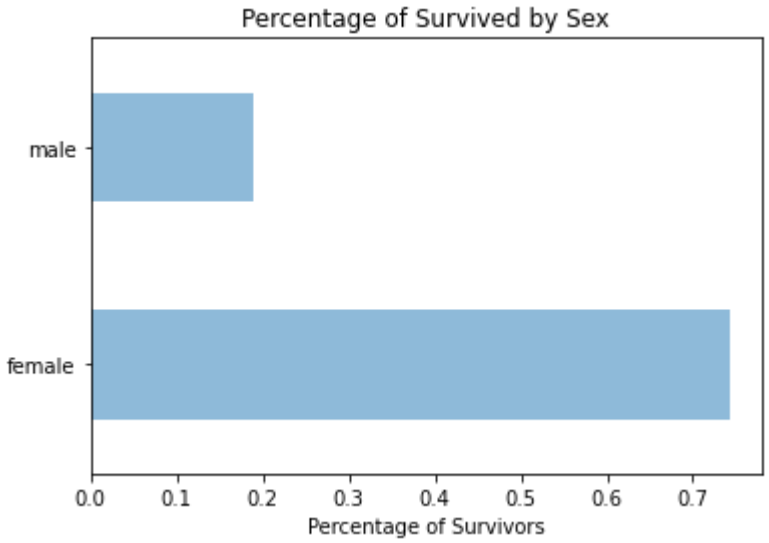
2.3 Percentage survived by sex

In [621…

```
# Since we belive that 'Sex' is an important feature to analyse the label, we will look at percentage of survived by sex
df_survived = df[df.Survived == 1]

plt.title("Percentage of Survived by Sex")
numer = df_survived.Sex.value_counts().sort_index()
denom = df.Sex.value_counts().sort_index()
(numer/denom).plot(kind='barh', alpha=0.5)
plt.xlabel("Percentage of Survivors")
```

Out[621…

```
Text(0.5, 0, 'Percentage of Survivors')
```



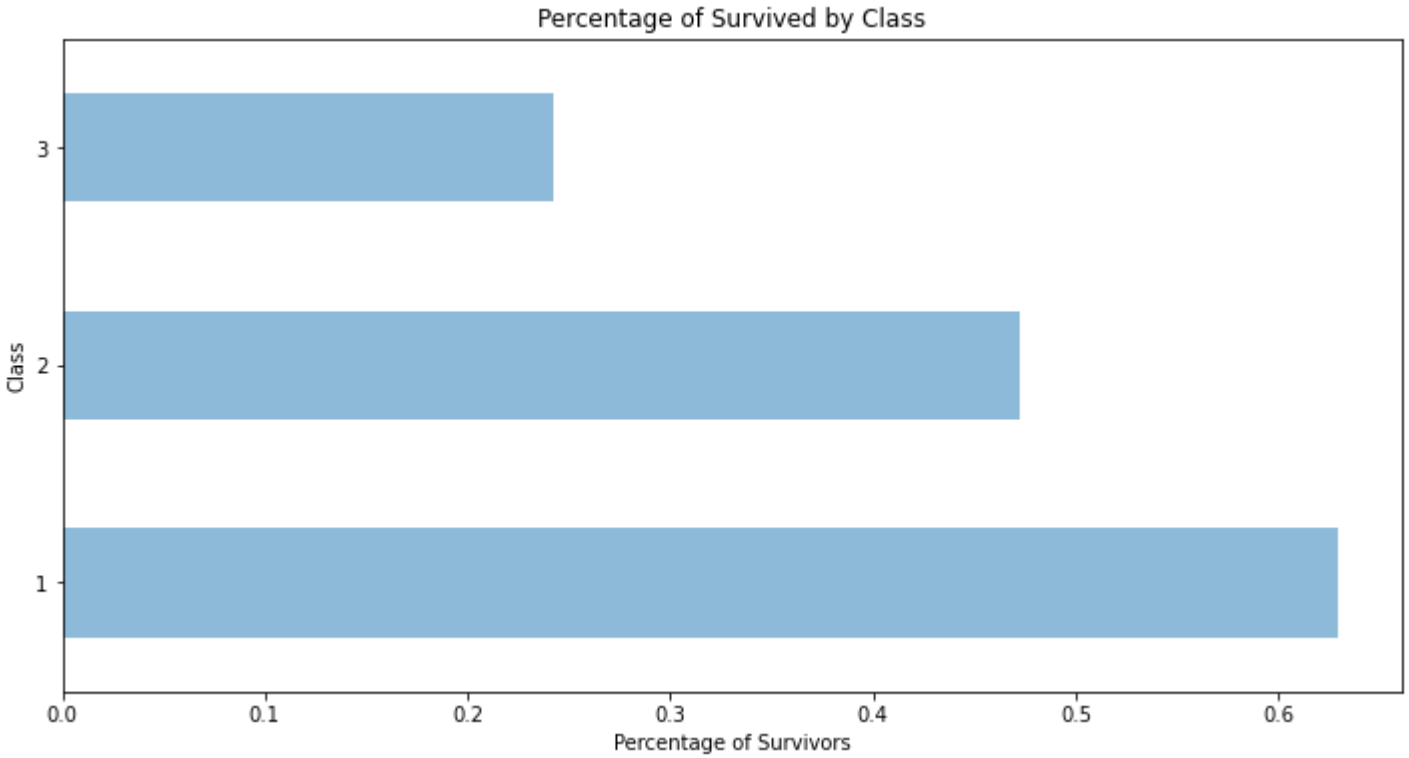
2.4 Percentage survived by Class

In [622…

```
# Since we belive that 'Pclass' is an important feature to analyse the label, we will look at percentage of survived by c
plt.figure(figsize=(12, 6))
plt.title("Percentage of Survived by Class")
numer2 = df_survived.Pclass.value_counts().sort_index()
denom2 = df.Pclass.value_counts().sort_index()
(numer2/denom2).plot(kind='barh', alpha=0.5)
plt.ylabel("Class")
plt.xlabel("Percentage of Survivors")
```

Out[622…

```
Text(0.5, 0, 'Percentage of Survivors')
```



In [623…

```
# Conclusions:

# We have a data frame with 891 samples, 11 features and 1 label
# We can already spot problems with features 'Age', 'Cabin', 'Embarked'

# Proportionally, women survived more than man
# Porportionally, higher classes (1 being highest) survived more than lower classes (3 being lowest)
```

3. Objective and methodology

In [624…

```
# Our aim is to predict the 'survived' column

# We will approach this by:
## (i) Handling categorical and other data
## (ii) Looking for missing data and other issues
## (iii) Visualising and looking for outliers in the data
## (iv) Splitting the data into a 33%/67% train/split
```

```
## (v) Using a logistic regression, KNN, decision tree and SVM model to see which performs best
## (vi) Removing one feature at a time and seeing its effect
```

4. Handle categorical and other data

4.1 'Embarked' feature

In [625…
```
df['Embarked'].unique()
```

Out[625…
```
array(['S', 'C', 'Q', nan], dtype=object)
```

In [626…
```
df['Embarked'].value_counts()
```

Out[626…
```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

In [627…
```
emb_mapping = {'S':1, 'C':2, 'Q':3}
df['EmbMap'] = df.Embarked.map(emb_mapping)
df.drop('Embarked', axis=1, inplace=True)

df.head()
```

Out[627…

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	EmbMap
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	1.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	2.0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	1.0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	1.0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	1.0

4.2 'Sex' Feature

In [628…

```
sex_mapping = {'male':0, 'female':1}
df['SexMap'] = df.Sex.map(sex_mapping)
df.drop('Sex', axis=1, inplace=True)

df.head()
```

Out[628…

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	EmbMap	SexMap
0	1	0	3	Braund, Mr. Owen Harris	22.0	1	0	A/5 21171	7.2500	NaN	1.0	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0	PC 17599	71.2833	C85	2.0	1
2	3	1	3	Heikkinen, Miss. Laina	26.0	0	0	STON/O2. 3101282	7.9250	NaN	1.0	1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0	113803	53.1000	C123	1.0	1
4	5	0	3	Allen, Mr. William Henry	35.0	0	0	373450	8.0500	NaN	1.0	0

5. Look for missing data and other issues

5.1 Missing data in 'Cabin'

In [629…

```
# The feature cabin seems to replicate the info in Pclass, given that only higher class passengers have cabins
# Therefore, we are dropping this feature

df.drop('Cabin', axis=1, inplace=True)

df.head()
```

Out[629…

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	EmbMap	SexMap
0	1	0	3	Braund, Mr. Owen Harris	22.0	1	0	A/5 21171	7.2500	1.0	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0	PC 17599	71.2833	2.0	1

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	EmbMap	SexMap
2	3	1	3	Heikkinen, Miss. Laina	26.0	0	0	STON/O2. 3101282	7.9250	1.0	1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0	113803	53.1000	1.0	1
4	5	0	3	Allen, Mr. William Henry	35.0	0	0	373450	8.0500	1.0	0

5.2 Missing data in 'Age'

In [630…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Ticket       891 non-null    object
 8   Fare         891 non-null    float64
 9   EmbMap       889 non-null    float64
 10  SexMap       891 non-null    int64
dtypes: float64(3), int64(6), object(2)
memory usage: 76.7+ KB
```

In [631…

```
# We will fill the na data in Age with the average age

averageAge = df['Age'].mean()
averageAge
```

Out[631…

```
29.69911764705882
```

In [632…

```
# Checking the average age of survivors to see if there is any significant difference
```

```
df_survived['Age'].mean()
```

Out[632…
```
28.343689655172415
```

In [633…
```
# There is no significant difference, so we assign the mean to the missing data using .fillna

df['Age'] = df['Age'].fillna(averageAge)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Age          891 non-null    float64
 5   SibSp        891 non-null    int64  
 6   Parch        891 non-null    int64  
 7   Ticket       891 non-null    object 
 8   Fare         891 non-null    float64
 9   EmbMap       889 non-null    float64
 10  SexMap       891 non-null    int64  
dtypes: float64(3), int64(6), object(2)
memory usage: 76.7+ KB
```

5.3 Missing data in EmbMap

In [634…
```
# We have 2 missing data in EmbMap.

## Since it is only 2 samples, we can simply drop these rows from df

df = df.dropna()

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
```

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  889 non-null    int64
 1   Survived     889 non-null    int64
 2   Pclass       889 non-null    int64
 3   Name         889 non-null    object
 4   Age          889 non-null    float64
 5   SibSp        889 non-null    int64
 6   Parch        889 non-null    int64
 7   Ticket       889 non-null    object
 8   Fare         889 non-null    float64
 9   EmbMap       889 non-null    float64
 10  SexMap       889 non-null    int64
dtypes: float64(3), int64(6), object(2)
memory usage: 83.3+ KB
```

5.4 Standarizing the 'Age' and 'Fare' feature

In [635…

```
type(df.Age.values)
```

Out[635…

```
numpy.ndarray
```

In [636…

```
df.Age.values.shape
```

Out[636…

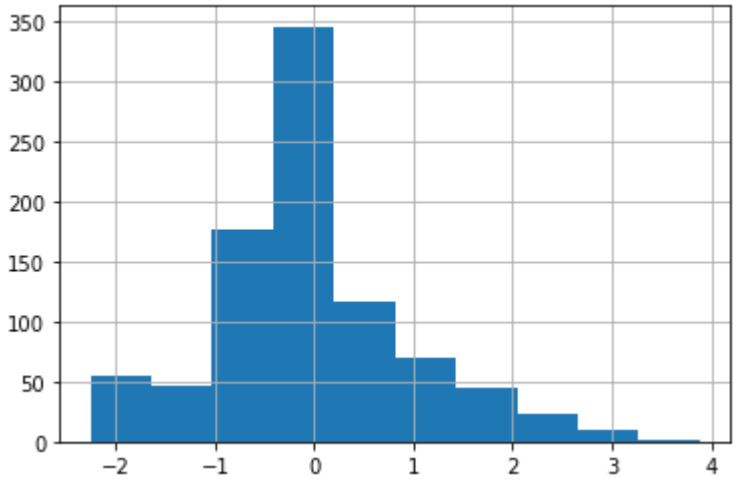
```
(889,)
```

In [637…

```
from sklearn.preprocessing import StandardScaler
df['AgeSD'] = StandardScaler().fit_transform(df.Age.values.reshape(-1,1))
df.drop('Age', axis=1, inplace=True)

df.AgeSD.hist()
```

Out[637…

```
<AxesSubplot:>
```



In [638…

```
type(df.Fare.values)
```

Out[638…

```
numpy.ndarray
```

In [639…

```
df.Fare.values.shape
```

Out[639…

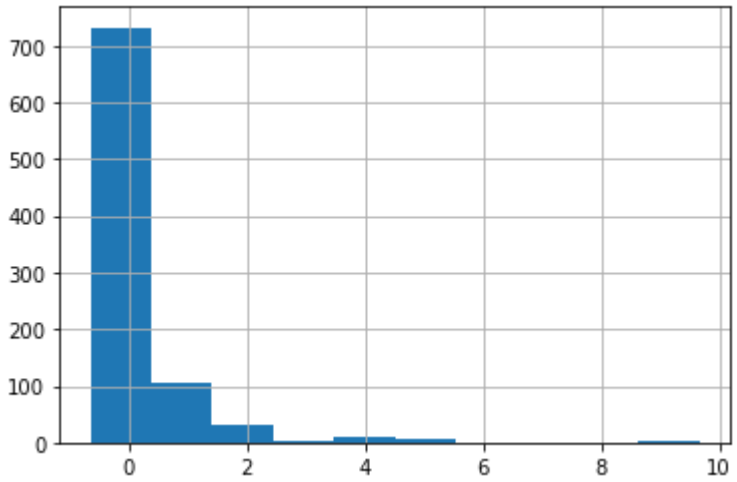
```
(889,)
```

In [640…

```
from sklearn.preprocessing import StandardScaler
df['FareSD'] = StandardScaler().fit_transform(df.Fare.values.reshape(-1,1))
df.drop('Fare', axis=1, inplace=True)

df.FareSD.hist()
```

Out[640…

```
<AxesSubplot:>
```



5.5 Removing other redundant features

In [641…

```
# We will drop the 'Name', 'PassengerId' and 'Ticket' for they will not help in our analysis
```

In [642…

```
df.drop(['Name'], axis=1, inplace=True)
df.drop(['PassengerId'], axis=1, inplace=True)
df.drop(['Ticket'], axis=1, inplace=True)
```

6. Visualise the data and look for outliers to remove

In [643…

```
df
```

Out[643…

	Survived	Pclass	SibSp	Parch	EmbMap	SexMap	AgeSD	FareSD
0	0	3	1	0	1.0	0	-0.590495	-0.500240
1	1	1	1	0	2.0	1	0.643971	0.788947
2	1	3	0	0	1.0	1	-0.281878	-0.486650
3	1	1	1	0	1.0	1	0.412509	0.422861
4	0	3	0	0	1.0	0	0.412509	-0.484133

	Survived	Pclass	SibSp	Parch	EmbMap	SexMap	AgeSD	FareSD
...	...	...	...	...	...	...	...	...
886	0	2	0	0	1.0	0	-0.204724	-0.384475
887	1	1	0	0	1.0	1	-0.821957	-0.042213
888	0	3	1	2	1.0	1	0.003524	-0.174084
889	1	1	0	0	2.0	0	-0.281878	-0.042213
890	0	3	0	0	3.0	0	0.181046	-0.490173

889 rows × 8 columns

In [644…

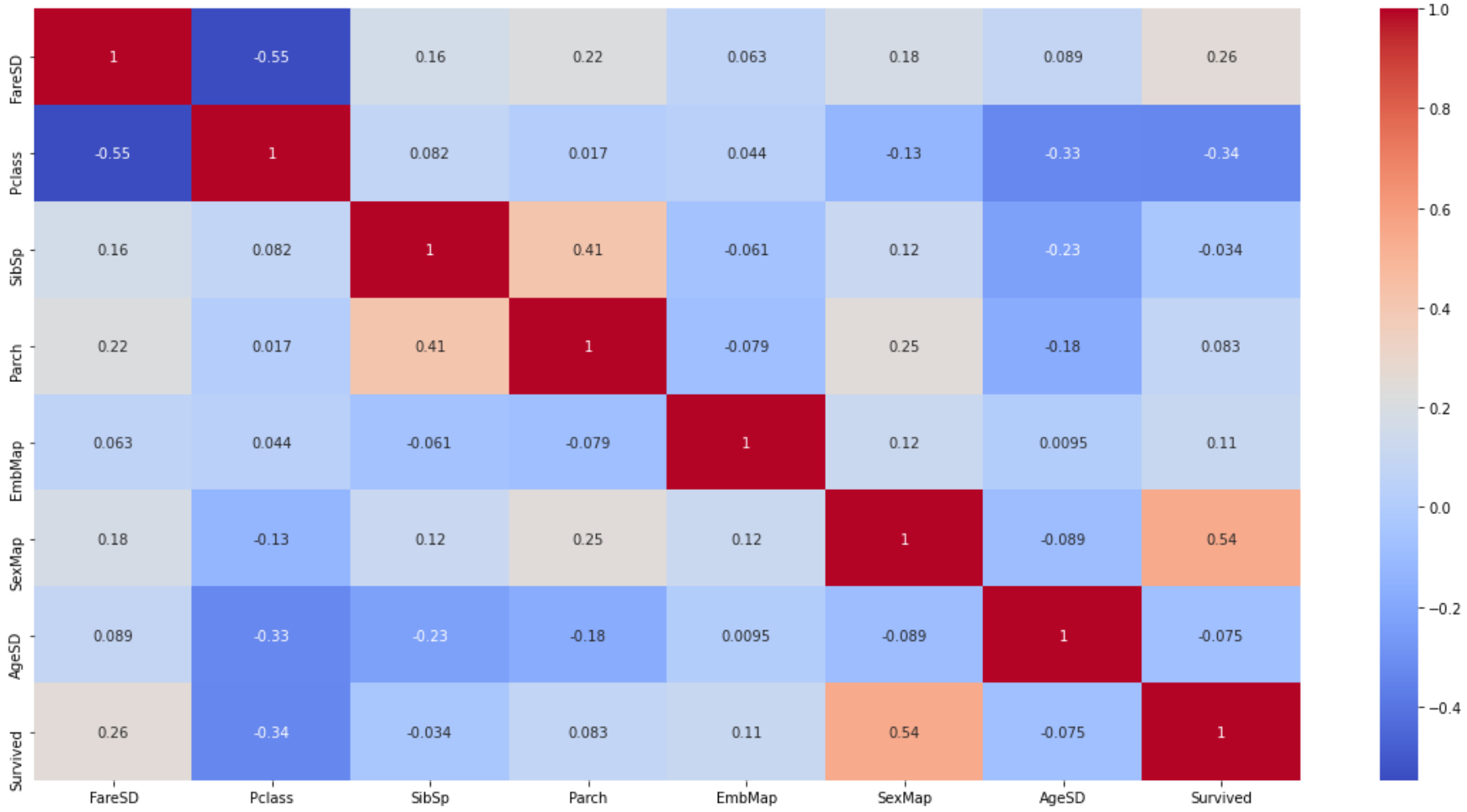
```
# Reordering columns for organization

df = df[['FareSD', 'Pclass', 'SibSp', 'Parch', 'EmbMap', 'SexMap', 'AgeSD', 'Survived']]
```

In [645…

```
## Plot a correlation matrix to see the correlations between (i) features x label and (ii) features x features

corr = df.corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[645…

```
<AxesSubplot:>
```



In [646…

```
# Brief analysis of correlation matrix:

## As we can observe, 'SexMap' and 'Pclass' are the most relevant features to determine who survived
## Note that 'FareSD' also has a relatively high correlation with the label, but it also has a high correlation with 'Pcl
### which indicates that part of the information provived by these features are duplicated
## We will maintain both for now and see how things go in step 9
```

7. Regression model and train/test split

7.1 Creating matrix X and vector y

In [647…
```
X = df[['FareSD', 'Pclass', 'SibSp', 'Parch', 'EmbMap', 'SexMap', 'AgeSD']]
y = np.array(df['Survived'])

X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   FareSD  889 non-null    float64
 1   Pclass  889 non-null    int64  
 2   SibSp   889 non-null    int64  
 3   Parch   889 non-null    int64  
 4   EmbMap  889 non-null    float64
 5   SexMap  889 non-null    int64  
 6   AgeSD   889 non-null    float64
dtypes: float64(3), int64(4)
memory usage: 55.6 KB
```

In [648…
```
y.shape
```

Out[648…
```
(889,)
```

7.2 Usefull functions from HelpFunctions

In [649…
```
# We will use this function from 'HelpFunctions' to plot the confusion matrices
from sklearn.metrics import plot_confusion_matrix

def plot_cm(clf, X, y, labs):

    mpl.rcParams.update({'font.size': 16})
    cm = plot_confusion_matrix(clf, X, y, display_labels=labs,cmap=mpl.cm.Blues);
```

7.3 Defining the 'train' function

In [650…
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

```
def train(model, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=999)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print('Model Report')
    print('Accuracy: ', np.around(accuracy_score(y_test, y_pred),4))
    print('Precision: ', np.around(precision_score(y_test, y_pred),4))
    print('Recall: ', np.around(recall_score(y_test, y_pred),4))
    print('f1: ', np.around(f1_score(y_test, y_pred),4))
    plot_cm(model, X, y, labs=('Not Survived', 'Survived'))
```

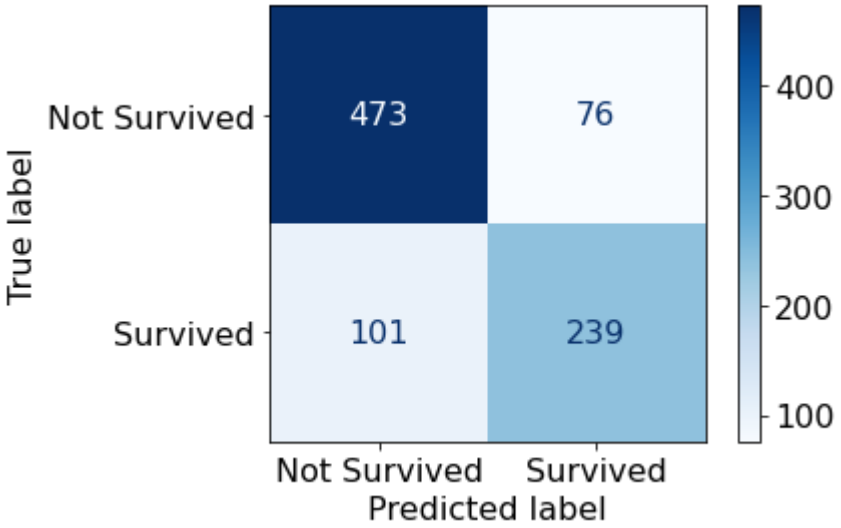
8. Choosing the best classification model

8.1 Logistic Regression

In [651…

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='liblinear')
train(model, X, y)
```

```
Model Report
Accuracy:  0.7925
Precision:  0.699
Recall:  0.7059
f1:  0.7024
```

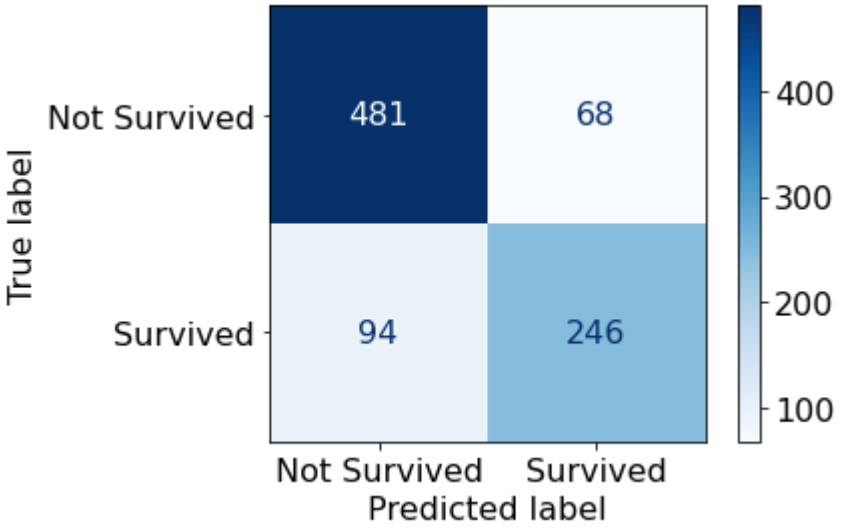


8.2 KNN

In [652…

```
from sklearn.neighbors import KNeighborsClassifier

k=9  # We tested odd k's from 3 to 31, 9 resulted in the best accuracy and precision metrics
model = KNeighborsClassifier(k)
train(model, X, y)
```

```
Model Report
Accuracy:  0.8027
Precision:  0.7292
Recall:  0.6863
f1:  0.7071
```

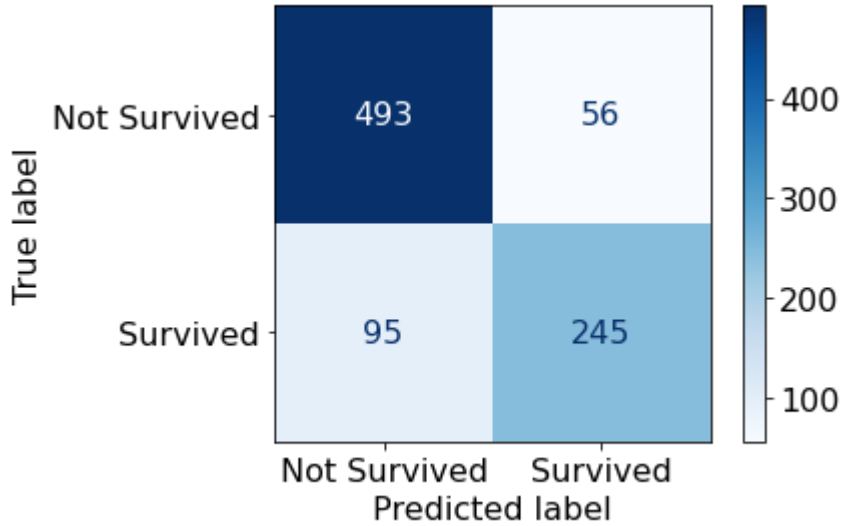


8.3 Decision Tree

In [653…

```
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
from os import system
from graphviz import Source
from IPython.display import SVG

model = DecisionTreeClassifier(random_state=1, max_depth=3)
train(model, X, y)
```

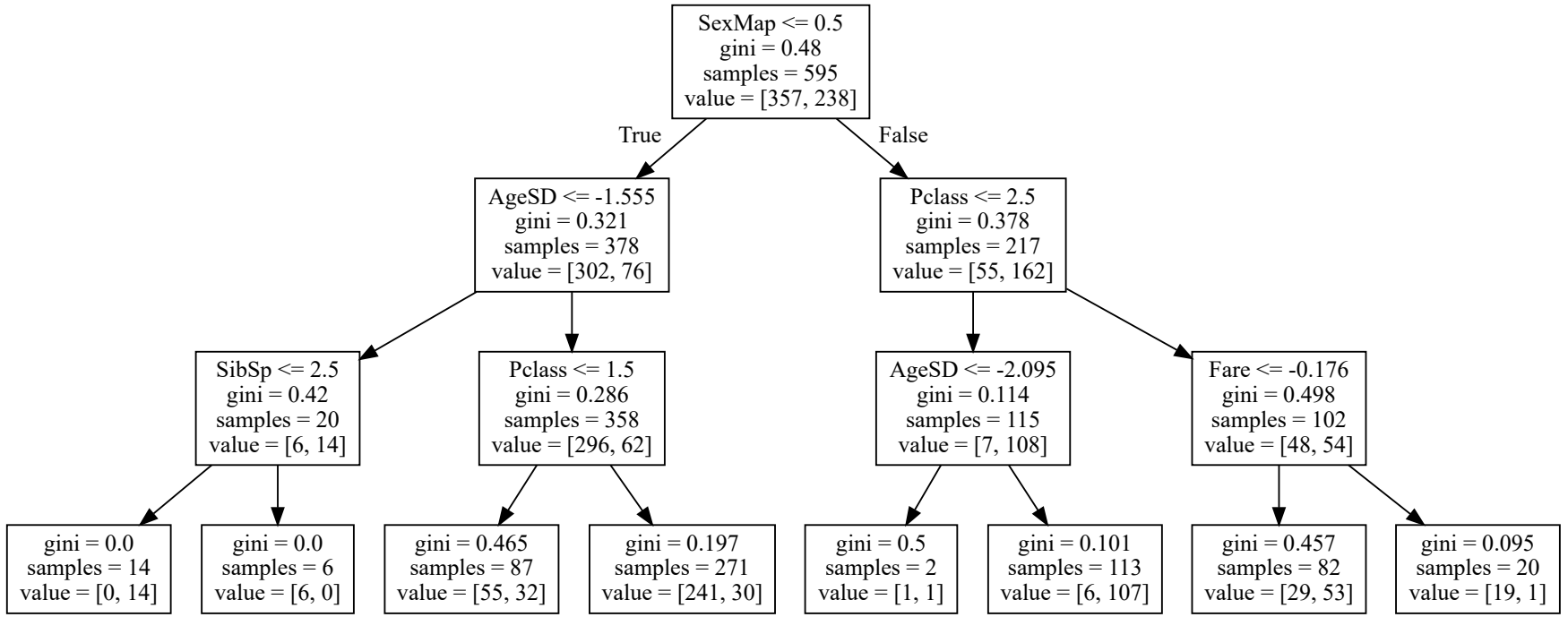
```
Model Report
Accuracy:  0.8231
Precision:  0.7717
Recall:  0.6961
f1:  0.732
```



In [654…

```
graph = Source(export_graphviz(model, out_file=None, feature_names=['Fare', 'Pclass', 'SibSp', 'Parch', 'EmbMap', 'SexMap
SVG(graph.pipe(format='svg'))
```

Out[654…

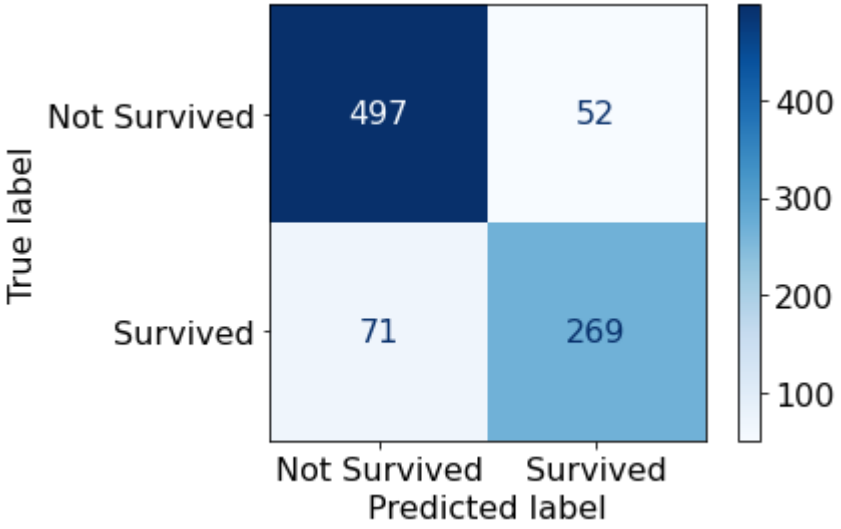


8.4 SVM

In [655…

```
from sklearn.svm import SVC

model = SVC(kernel='poly', coef0 = 1, C=10000)
train(model, X, y)
```

```
Model Report
Accuracy:  0.7959
Precision:  0.7283
Recall:  0.6569
f1:  0.6907
```



8.5 Comparing and choosing final model

In [656…

```
L = {'Logistic Regression':[0.7925,0.699, 0.7059, 0.7024], 'KNN':[0.8027,0.7292, 0.6863, 0.7071], 'Decision Tree':[0.8231

df_results = pd.DataFrame(L, index=['Accuracy', 'Precision', 'Recall', 'F1'])

df_results
```

Out[656…

	Logistic Regression	KNN	Decision Tree	SVM
Accuracy	0.7925	0.8027	0.8231	0.7959
Precision	0.6990	0.7292	0.7717	0.7283

	Logistic Regression	KNN	Decision Tree	SVM
Recall	0.7059	0.6863	0.6961	0.6569
F1	0.7024	0.7071	0.7320	0.6907

In [657…

```
# Conclusion:

## As can be observed in the DataFrame above, the Decision Tree model outperformed the three other models in all 4 metric
## Therefore, we will continue our analysis using the Decision Tree
```

9. Removing features and re-training model

In [658…

```
## We will remove one feature at a time and re-training the Decision Tree
```

In [659…

```
# We will name the new feature matrices by highlighting the removed feature
## Example: 'X_FareSD' is matrice X without the feature 'FareSD'

X_FareSD = X.drop(['FareSD'], axis=1)
X_Pclass = X.drop(['Pclass'], axis=1)
X_SibSp = X.drop(['SibSp'], axis=1)
X_Parch = X.drop(['Parch'], axis=1)
X_EmbMap = X.drop(['EmbMap'], axis=1)
X_SexMap = X.drop(['SexMap'], axis=1)
X_AgeSD = X.drop(['AgeSD'], axis=1)

L_X = [X, X_FareSD, X_Pclass, X_SibSp, X_Parch, X_EmbMap, X_SexMap, X_AgeSD]
L_f = ['All', 'FareSD', 'Pclass', 'SibSp', 'Parch', 'EmbMap', 'SexMap', 'AgeSD']
```

In [660…

```
results = pd.DataFrame(index=['Accuracy', 'Precision', 'Recall', 'F1'])
```

In [661…

```
def train_2(model, X, y, name='Feature'):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=999)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    stats = [np.around(accuracy_score(y_test, y_pred),4),np.around(precision_score(y_test, y_pred),4),
```

```
        np.around(recall_score(y_test, y_pred),4), np.around(f1_score(y_test, y_pred),4)]
    results[name] = stats
```

In [662…

```
model = DecisionTreeClassifier(random_state=1, max_depth=3)
```

In [663…

```
i = 0
for _ in range(0, len(L_X)):
    train_2(model, L_X[_], y, name = L_f[i])
    i = i + 1

results
```

Out[663…

	All	FareSD	Pclass	SibSp	Parch	EmbMap	SexMap	AgeSD
Accuracy	0.8231	0.8435	0.8197	0.8129	0.8231	0.8231	0.7041	0.8095
Precision	0.7717	0.8889	0.7579	0.7640	0.7717	0.7717	0.7273	0.7614
Recall	0.6961	0.6275	0.7059	0.6667	0.6961	0.6961	0.2353	0.6569
F1	0.7320	0.7356	0.7310	0.7120	0.7320	0.7320	0.3556	0.7053

In [664…

```
# Conclusion:

# The results shown in the DataFrame above are very interesting.
## We can interpret that the lower the metrics when the feature is removed, the more important the feature is.
## That is because without that feature our model works more poorly.

# 'Fare', 'Parch' and 'EmbMap'
## We notice that when we took out 'EmbMap' and 'Parch' none of the metrcis changed
## This shows that, independently, those features don't help to predict the label
## When we removed 'Fare', the metrics actually improved
## This shows that that our prediction is improved disconsidering that feature, so we are going to drop it and re-train t
```

In [665…

```
results = pd.DataFrame(index=['Accuracy', 'Precision', 'Recall', 'F1'])

X.drop(['FareSD'], axis=1, inplace=True)

X_Pclass = X.drop(['Pclass'], axis=1)
X_SibSp = X.drop(['SibSp'], axis=1)
```

```
X_Parch = X.drop(['Parch'], axis=1)
X_EmbMap = X.drop(['EmbMap'], axis=1)
X_SexMap = X.drop(['SexMap'], axis=1)
X_AgeSD = X.drop(['AgeSD'], axis=1)

L_X = [X, X_Pclass, X_SibSp, X_Parch, X_EmbMap, X_SexMap, X_AgeSD]
L_f = ['All', 'Pclass', 'SibSp', 'Parch', 'EmbMap', 'SexMap', 'AgeSD']
```

In [666…

```
i = 0
for _ in range(0, len(L_X)):
    train_2(model, L_X[_], y, name = L_f[i])
    i = i + 1

results
```

Out[666…

	All	Pclass	SibSp	Parch	EmbMap	SexMap	AgeSD
Accuracy	0.8435	0.8197	0.8333	0.8435	0.8197	0.7279	0.8265
Precision	0.8889	0.7579	0.8841	0.8889	0.7816	0.6618	0.8493
Recall	0.6275	0.7059	0.5980	0.6275	0.6667	0.4412	0.6078
F1	0.7356	0.7310	0.7135	0.7356	0.7196	0.5294	0.7086

In [667…

```
# Now we see that when we remove Parch the results don't change, so we drop it and retrain the model
```

In [668…

```
results = pd.DataFrame(index=['Accuracy', 'Precision', 'Recall', 'F1'])

X.drop(['Parch'], axis=1, inplace=True)

X_Pclass = X.drop(['Pclass'], axis=1)
X_SibSp = X.drop(['SibSp'], axis=1)
X_EmbMap = X.drop(['EmbMap'], axis=1)
X_SexMap = X.drop(['SexMap'], axis=1)
X_AgeSD = X.drop(['AgeSD'], axis=1)

L_X = [X, X_Pclass, X_SibSp, X_EmbMap, X_SexMap, X_AgeSD]
L_f = ['All', 'Pclass', 'SibSp', 'EmbMap', 'SexMap', 'AgeSD']

i = 0
```

```
for _ in range(0, len(L_X)):
    train_2(model, L_X[_], y, name = L_f[i])
    i = i + 1

results
```

Out[668…

	All	Pclass	SibSp	EmbMap	SexMap	AgeSD
Accuracy	0.8435	0.8231	0.8333	0.8231	0.7279	0.8299
Precision	0.8889	0.7604	0.8841	0.8125	0.6618	0.8824
Recall	0.6275	0.7157	0.5980	0.6373	0.4412	0.5882
F1	0.7356	0.7374	0.7135	0.7143	0.5294	0.7059

In [669…

```
## Since the weight of importance between false positives and false negatives is equal in this analysis we wil classify i

# Importance Hierarchy of features:
## SexMap
## AgeSD
## Pclass
## EmbMap
## SibSp
```

In [670…

```
##########################################################################
```

Question 3: Wisconsin Cancer

Team Members:

Marcel Santos de Carvalho, id 79083

Loris Baudry, id 79794

Alex Palacios, id 73713

Responsible for this notebook: Loris Baudry

In [671…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('seaborn')

%matplotlib inline
%pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\users\apala\anaconda3\lib\site-packages (0.17)
Note: you may need to restart the kernel to use updated packages.
```

1.- Data import

In [672…

```
from sklearn.datasets import load_breast_cancer
data=load_breast_cancer()
```

2.- Data Visualization and Analysis

2.1 Data Description

The Breast Cancer Dataset is comprised by 569 observations of 30 features and 2 labels. The classes are Malignant (0) and Bening (1). From the 30 features, all of them are numerical and there is no missing date in any of the features observations. Nonetheless, the scale of some features largly differs so we will standardize the features magnitude with the MinMax Scaling Method.

In [673…

```
print(data.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class
```

:Attribute Information:
    - radius (mean of distances from center to points on the perimeter)
    - texture (standard deviation of gray-scale values)
    - perimeter
    - area
    - smoothness (local variation in radius lengths)
    - compactness (perimeter^2 / area - 1.0)
    - concavity (severity of concave portions of the contour)
    - concave points (number of concave portions of the contour)
    - symmetry
    - fractal dimension ("coastline approximation" - 1)

    The mean, standard error, and "worst" or largest (mean of the three
    worst/largest values) of these features were computed for each image,
    resulting in 30 features. For instance, field 0 is Mean Radius, field
    10 is Radius SE, field 20 is Worst Radius.

    - class:
            - WDBC-Malignant
            - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079

```
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].
```

```
This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

In [674…

```
x=pd.DataFrame(data.data)
```

In [675…

```
x.shape
```

Out[675…

```
(569, 30)
```

In [676…

```
x.head()
```

Out[676…

	0	1	2	3	4	5	6	7	8	9	…	20	21	22	23	24	25	26	2
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	…	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.265
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	…	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.186
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	…	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.243
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	…	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.257
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	…	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.162

5 rows × 30 columns

In [677…
```
data.feature_names
```

Out[677…
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [678…
```
x.columns=data.feature_names
```

In [679…
```
x.head()
```

Out[679…

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	wo a
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	201
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	195
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	170
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	56
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	157

5 rows × 30 columns

In [680…
```
y=data.target
```

In [681…
```
print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0]
```

```
1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1
1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

In [682…]
```
print(data.target_names)
```

```
['malignant' 'benign']
```

In [683…]
```
types = []
for i in y:
    name = data['target_names'][i]
    types.append(name)
```

In [684…]
```
x['types'] = types
```

In [685…]
```
x.head()
```

Out[685…]

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	sm
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	

5 rows × 31 columns

In [686…

```
x.tail()
```

Out[686…

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6

5 rows × 31 columns

In [687…

```
x.isna().count()
```

Out[687…

```
mean radius                569
mean texture               569
mean perimeter             569
mean area                  569
mean smoothness            569
mean compactness           569
mean concavity             569
mean concave points        569
mean symmetry              569
mean fractal dimension     569
radius error               569
texture error              569
perimeter error            569
area error                 569
smoothness error           569
compactness error          569
concavity error            569
concave points error       569
```

```
symmetry error             569
fractal dimension error    569
worst radius               569
worst texture              569
worst perimeter            569
worst area                 569
worst smoothness           569
worst compactness          569
worst concavity            569
worst concave points       569
worst symmetry             569
worst fractal dimension    569
types                      569
dtype: int64
```

In [688…

```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
```

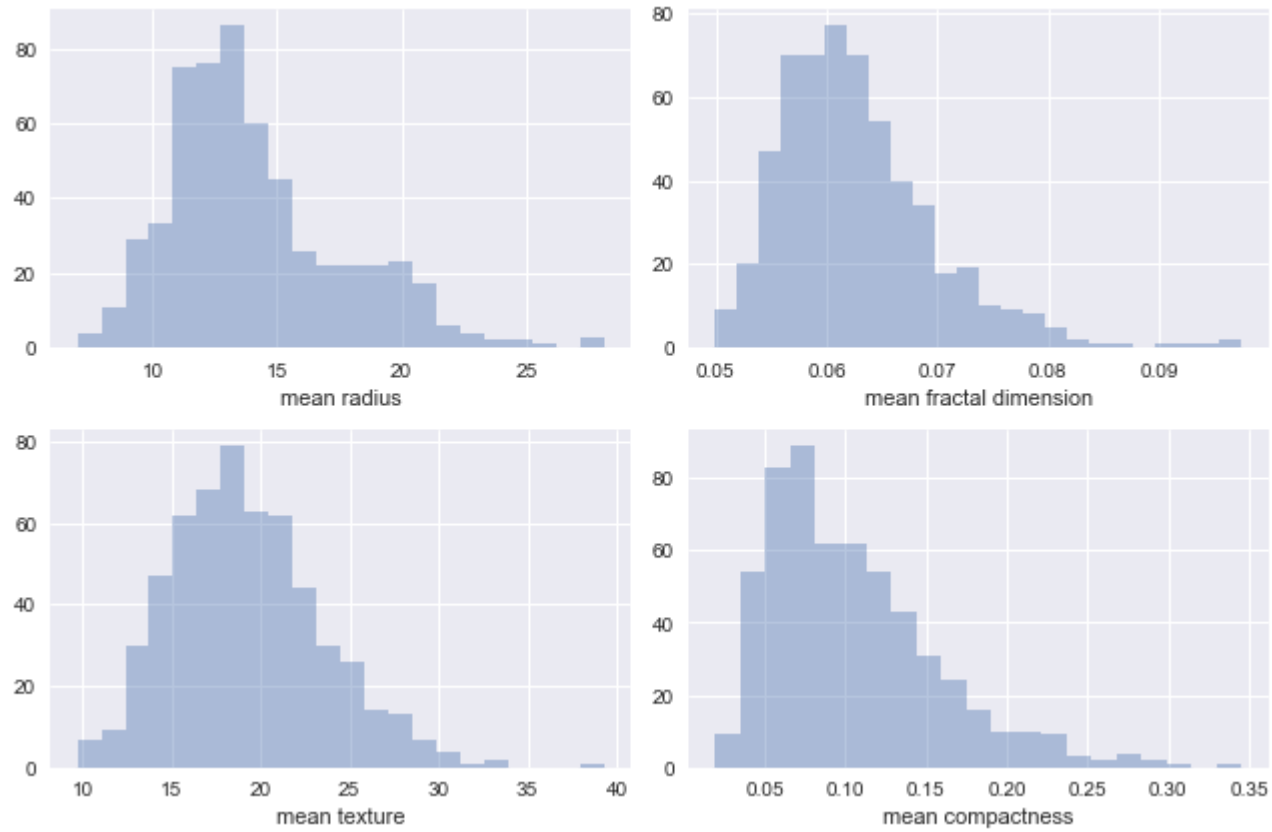
```
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  types                    569 non-null    object
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

2.1 Feature Standardadization

We chose 4 randome feautres to show the difference in the scale of the features and how the MinMa Scaling method helped us to fix that problem by replotting the rescaled features

In [689…

```
import seaborn as sns
x_hist = x[['mean radius', 'mean fractal dimension', 'mean texture', 'mean compactness']]

fig, ax = plt.subplots(ncols=2, nrows=2, figsize=(9, 6))
index=0
ax = ax.flatten()

for col, values in x_hist.items():
    sns.distplot(values, ax=ax[index], kde=False)
    index += 1
plt.tight_layout()
```



In [690…

```
from sklearn.preprocessing import MinMaxScaler
x.iloc[:,0:-1] = MinMaxScaler().fit_transform(x.iloc[:,0:-1])
```

In [691…

```
x.head()
```

Out[691…

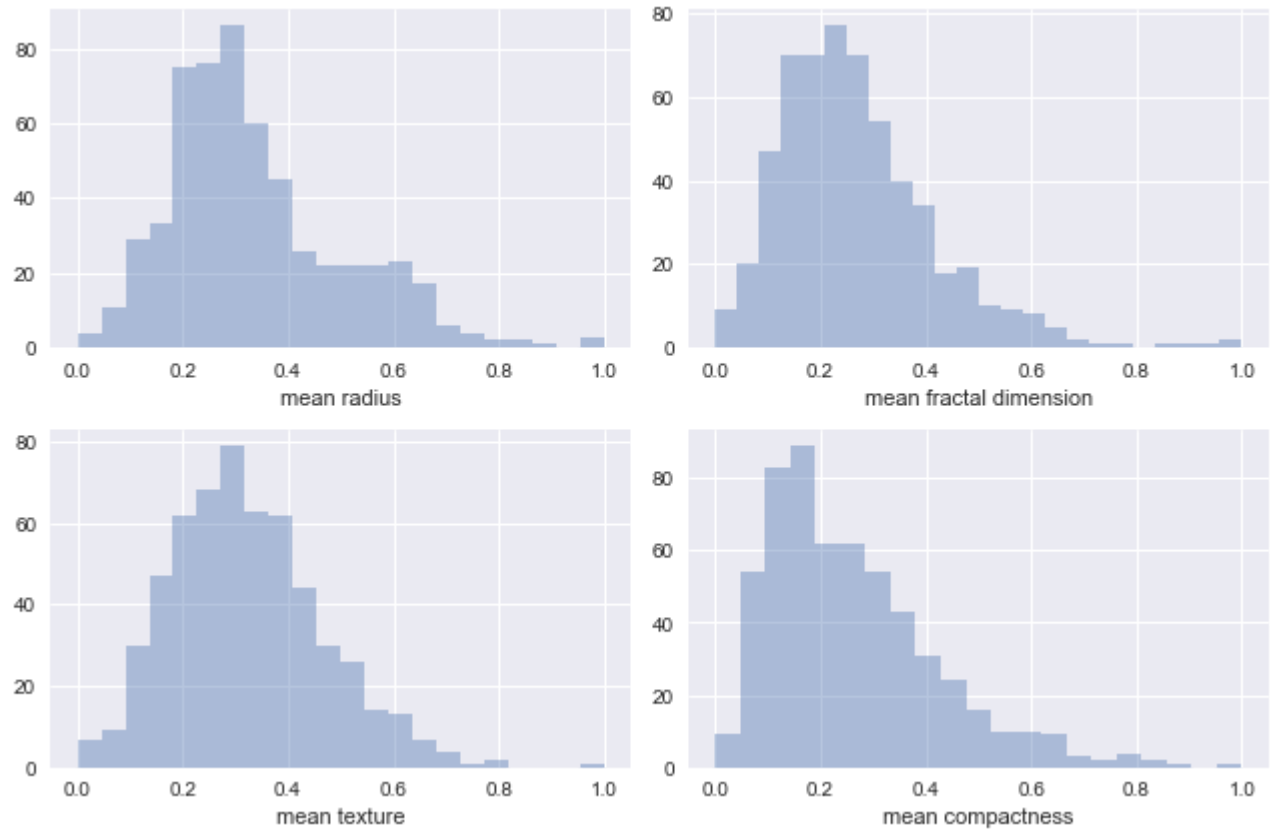
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	w
0	0.521037	0.022658	0.545989	0.363733	0.593753	0.792037	0.703140	0.731113	0.686364	0.605518	...	0.141525	0.668310	0.45
1	0.643144	0.272574	0.615783	0.501591	0.289880	0.181768	0.203608	0.348757	0.379798	0.141323	...	0.303571	0.539818	0.43
2	0.601496	0.390260	0.595743	0.449417	0.514309	0.431017	0.462512	0.635686	0.509596	0.211247	...	0.360075	0.508442	0.37
3	0.210090	0.360839	0.233501	0.102906	0.811321	0.811361	0.565604	0.522863	0.776263	1.000000	...	0.385928	0.241347	0.09

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	w
4	0.629893	0.156578	0.630986	0.489290	0.430351	0.347893	0.463918	0.518390	0.378283	0.186816	...	0.123934	0.506948	0.34

5 rows × 31 columns

In [692…

```
x_hist = x[['mean radius', 'mean fractal dimension', 'mean texture', 'mean compactness']]

fig, ax = plt.subplots(ncols=2, nrows=2, figsize=(9, 6))
index=0
ax = ax.flatten()

for col, values in x_hist.items():
    sns.distplot(values, ax=ax[index], kde=False)
    index += 1
plt.tight_layout()
```



In [693…

```
#from sklearn.preprocessing import StandardScaler
#x.iloc[:,0:-1] = StandardScaler().fit_transform(x.iloc[:,0:-1])
#x["mean radius"].hist()
```

In [694…

```
#x.head()
```

3.- Explanatory Power Analysis

3.1 Correlations Analysis and P-values

As part of the data analysis, we will look for hihgly correlated features as having multicollinearity implies that there might be some regressors not adding much information to the model. We will try to idenify those variables that seem not to have high explanatory power

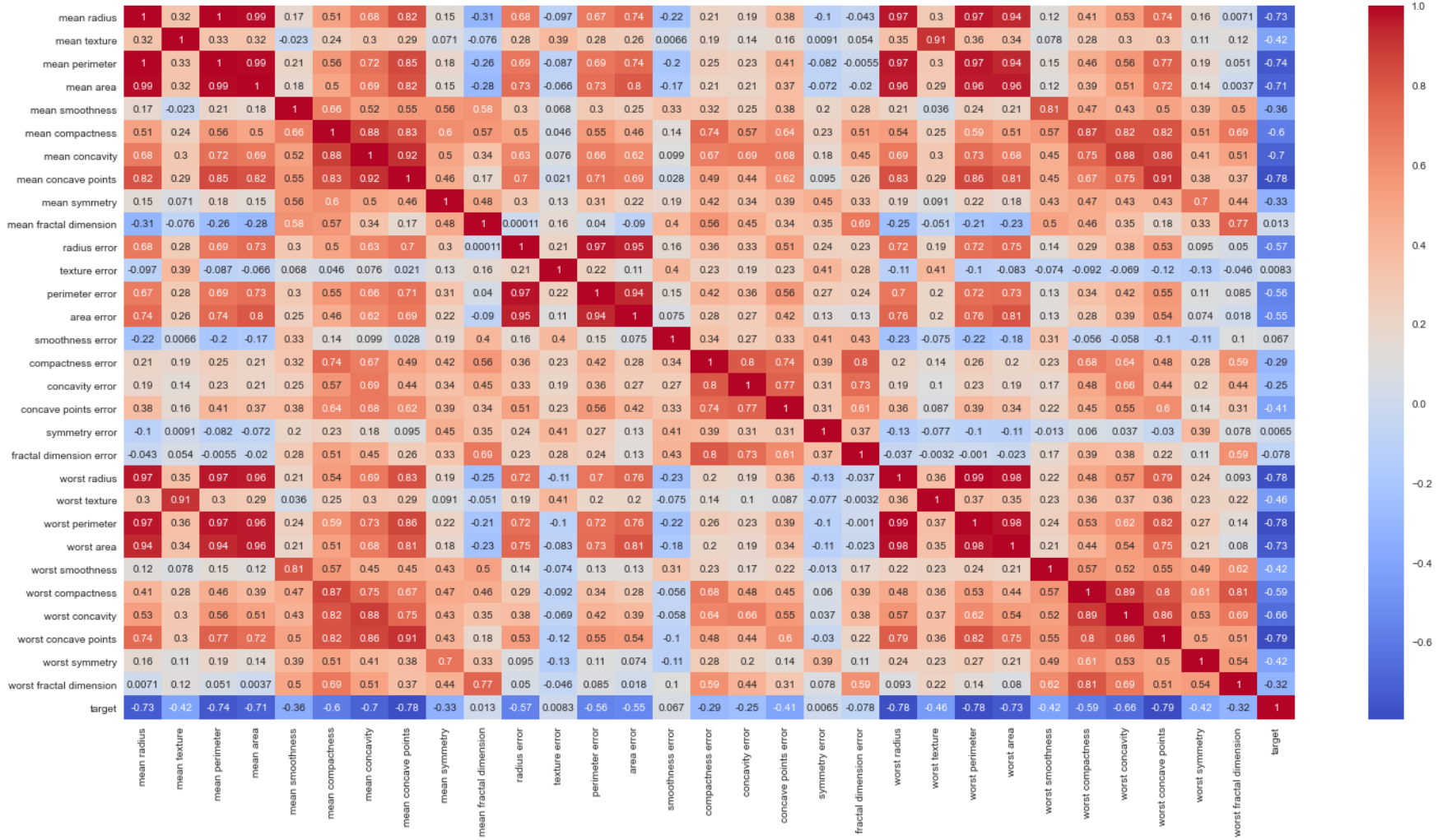
and to create a second sample set without them. For the remaining of the work we will do each analysis with two samples to compare the results

In [695…

```
## Plot a correlation matrix to see the correlations between (i) features x label and (ii) features x features
x['target']=y
import seaborn as sns
corr = x.corr()
plt.figure(figsize=(25,12.5))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[695…

```
<AxesSubplot:>
```



From the correlation matrix is easy to see that some variables are hihgly correlated and others have veyr low correlation regarding the targe variable:

```
1.- Mean Perimeter, mean area, worst radius, worst perimeter and worst area are highly correlated with
mean radius
2.- Worst texture is highly correlated with mean texture
3.- Mean concavity is highly correlated with mean concave points
4.- Worst concave points is highly correlated with mean concave points
5.- Perimeter error and area error are highly correlated with radius error
6.- Perimeter error is highly correlate with area error
7.- Worst perimeter and worst area are highly correlated with worst radius
8.- Worst area is highly correlated with worst perimeter
9.- Mean fractal dimension, texture error, smoothness error, symmetry error, fractal dimension error,
have very low correlation with target.
```

Let us see how the correlation matrix shows after removing the following features:

```
 - Mean concave points, mean texture, perimeter error, area error, worst perimeter, worst area, mean
perimeter, mean area, worst radius, worst perimeter and worst area, mean fractal dimension, texture
error, smoothness error, symmetry error, fractal dimension error
```

In [696…

```
x1=x.drop(["mean concave points", "mean texture", "perimeter error", "area error", "worst perimeter", "worst area", "mear
x1.shape
```

Out[696…

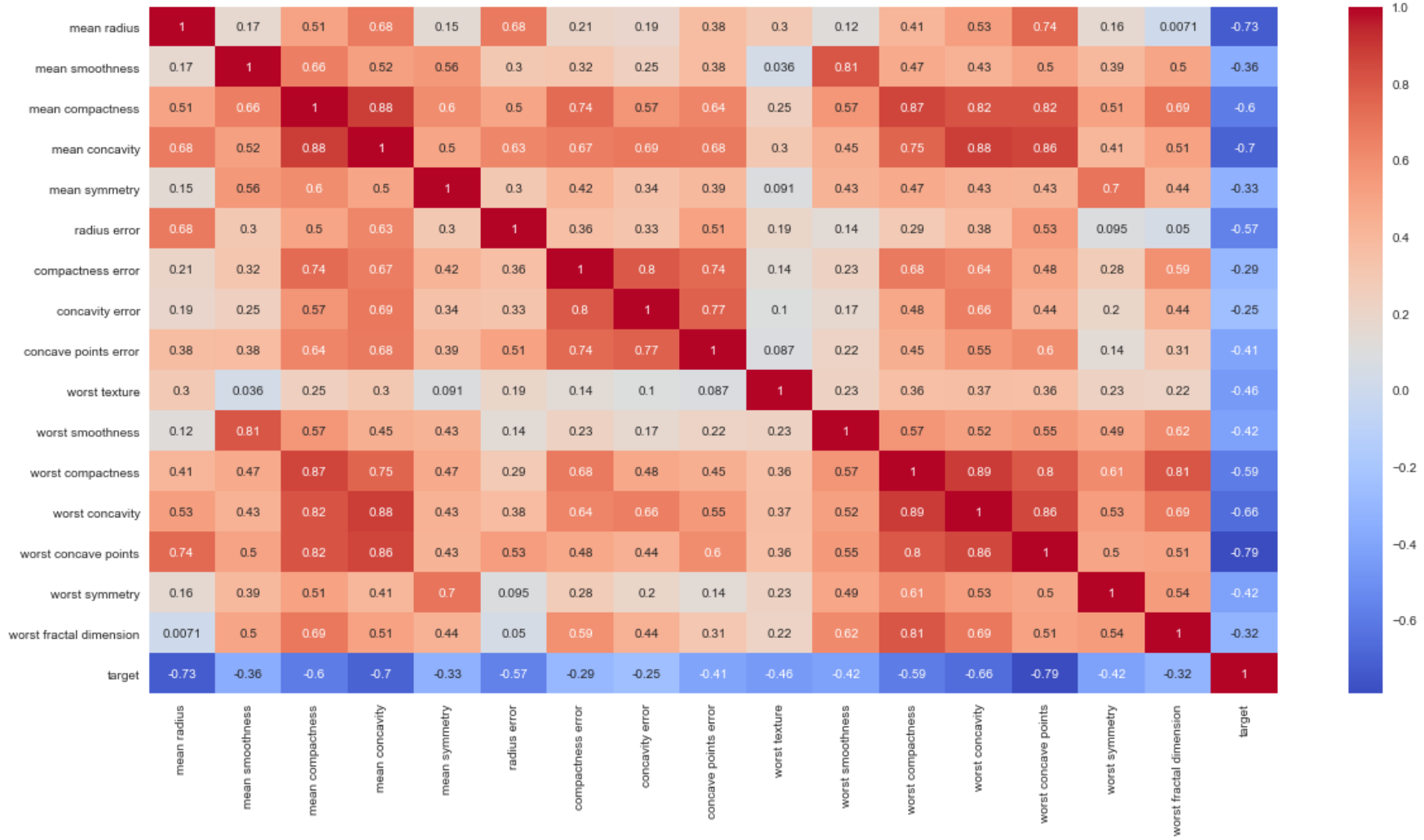
```
(569, 18)
```

In [697…

```
corr = x1.corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[697…

```
<AxesSubplot:>
```



We also compute the p-values of the regressors. Noramlly, a p-value higher than 5% (the common significance level) means that the regressor coefficient is not statistically different from zero. In other words, you could erase that regressor. From the analysis below, it seems that "mean compactness", "concavity error", "worst area" and "worst radius" are the variables with the highest explanatory power.Nevertheless, we won't be erasing the features with hihg p-values as would have very little features to work on.

In [699…

```
import statsmodels.api as sm
xt=x.drop(["types","target"],axis=1)
xt2= sm.add_constant(xt)
est = sm.OLS(y, xt2,hasconstant=True)
```

```
est2 = est.fit()
print(est2.summary(alpha=0.05))
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.774
Model:                            OLS   Adj. R-squared:                  0.762
Method:                 Least Squares   F-statistic:                     61.53
Date:                Mon, 18 Oct 2021   Prob (F-statistic):          6.05e-153
Time:                        10:28:47   Log-Likelihood:                 29.650
No. Observations:                 569   AIC:                             2.699
Df Residuals:                     538   BIC:                             137.4
Df Model:                          30
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                            1.7323      0.104     16.580      0.000       1.527       1.937
mean radius                      4.6013      3.666      1.255      0.210      -2.600      11.803
mean texture                    -0.1344      0.235     -0.572      0.567      -0.596       0.327
mean perimeter                  -3.4354      3.632     -0.946      0.345     -10.570       3.699
mean area                       -0.7493      1.238     -0.605      0.545      -3.182       1.683
mean smoothness                 -0.0094      0.223     -0.042      0.967      -0.448       0.430
mean compactness                 1.3765      0.435      3.166      0.002       0.522       2.231
mean concavity                  -0.5967      0.446     -1.337      0.182      -1.474       0.280
mean concave points             -0.4309      0.398     -1.082      0.280      -1.213       0.351
mean symmetry                   -0.0203      0.147     -0.138      0.890      -0.309       0.269
mean fractal dimension          -0.0016      0.265     -0.006      0.995      -0.521       0.518
radius error                    -1.2011      0.857     -1.401      0.162      -2.885       0.483
texture error                    0.0306      0.167      0.183      0.855      -0.297       0.358
perimeter error                  0.4779      0.873      0.548      0.584      -1.236       2.192
area error                       0.4943      0.748      0.660      0.509      -0.976       1.964
smoothness error                -0.4664      0.195     -2.393      0.017      -0.849      -0.084
compactness error               -0.0086      0.289     -0.030      0.976      -0.576       0.559
concavity error                  1.4119      0.515      2.741      0.006       0.400       2.424
concave points error            -0.5579      0.288     -1.938      0.053      -1.123       0.007
symmetry error                  -0.1206      0.194     -0.622      0.534      -0.501       0.260
fractal dimension error          0.2069      0.338      0.612      0.541      -0.457       0.871
worst radius                    -5.4866      1.629     -3.367      0.001      -8.688      -2.286
worst texture                   -0.2686      0.261     -1.030      0.303      -0.781       0.244
worst perimeter                  0.4889      1.192      0.410      0.682      -1.852       2.830
worst area                       4.1145      1.301      3.163      0.002       1.559       6.669
worst smoothness                -0.0822      0.217     -0.378      0.705      -0.509       0.345
worst compactness               -0.0692      0.395     -0.175      0.861      -0.845       0.706
worst concavity                 -0.4773      0.336     -1.419      0.156      -1.138       0.183
```

```
worst concave points       -0.1351      0.266     -0.508      0.612      -0.658       0.387
worst symmetry             -0.2825      0.251     -1.126      0.260      -0.775       0.210
worst fractal dimension    -0.6561      0.363     -1.806      0.072      -1.370       0.058
==============================================================================
Omnibus:                       32.654   Durbin-Watson:                   1.794
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               36.690
Skew:                          -0.603   Prob(JB):                     1.08e-08
Kurtosis:                       3.302   Cond. No.                         923.
==============================================================================

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

3.2 Features distribution by class

We will create a distribution plot to see how differentiated are each feature in term of the labels, this will also give us an idea of which features my have higher explanatory power such as mean perimeter. Even though, in the middle of the plot there is no celar difference between classes, this feature is clearly differentiated at the limits of the x-axis

In [700…

```python
plt_idx = 1
plt.figure(figsize=(20, 15))

for index1 in range(0,30):
        f1 = x.columns[index1]
        xp = x[f1]
        plt.subplot(6,5,plt_idx,label=f1)
        plt.scatter(xp,y,c = 'red')
        plt.xlabel(f1,labelpad=-75)
        plt_idx = plt_idx+1
```

4.- Metrics for model evaluation

For the model evaluation we will be using the next metrics:

$$Accuracy=\frac{TP+TN}{FP+FN+TP+TN}$$

$$Precision=\frac{TP}{TP+FP}$$

$$Recall=\frac{TP}{TP+FN}$$

$$F1=2\times\frac{Precision\times Recall}{Precision+Recall}$$

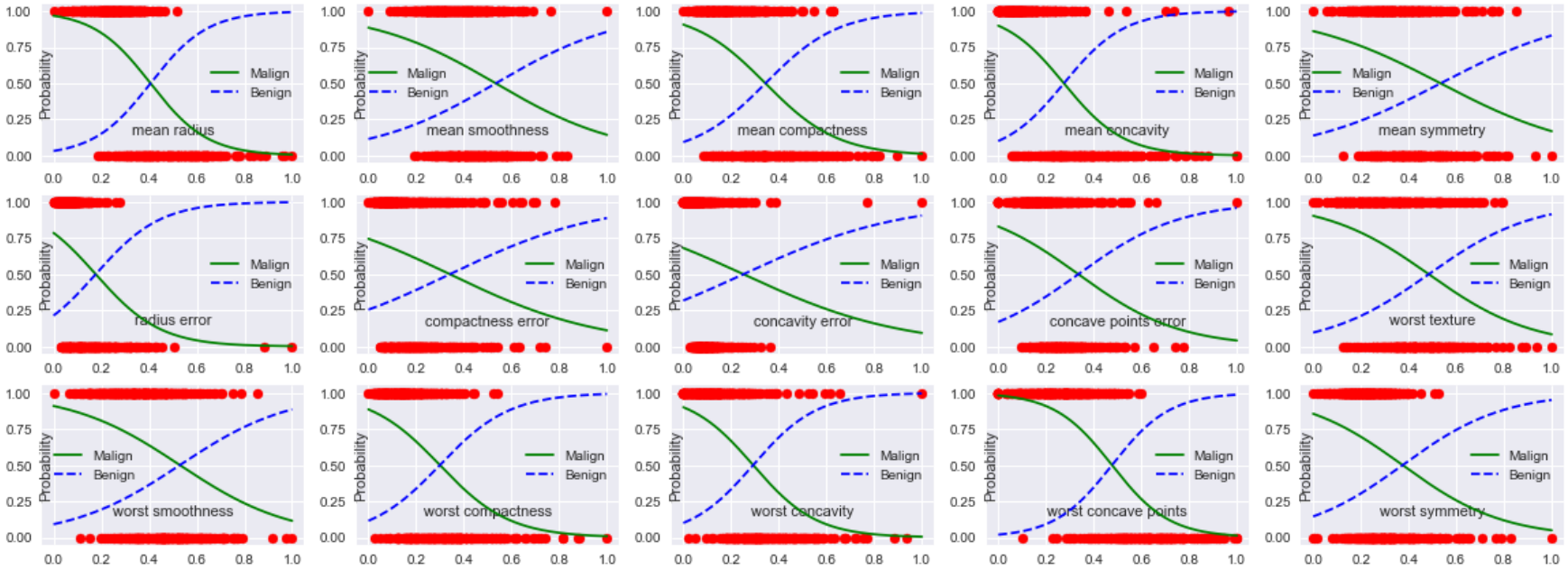
5.- Model Fitting

We will now fit different models to our data. We will apply each model twice. One for the original data and one for the second set in which we remove some variables. We will fit the model to a train set (70% of total data randomly selected) and then predict values for the rest 30%. The models we will aplly are Logistic Regression, K-Nearest Neighbor, Decision Tree and Supported Vector Machine

5.1 Logistic Regression

In [701…

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver='liblinear')
plt_idx = 1
plt.figure(figsize=(20, 15))

for index1 in range(0,15):
        f1 = x1.columns[index1]
        xp = x1[f1].values.reshape(569,1)
        log_reg.fit(xp,y)
        X_new = np.linspace(xp.min(),xp.max(),1000).reshape(-1,1)
        y_proba = log_reg.predict_proba(X_new)
        plt.subplot(6,5,plt_idx)
        plt.scatter(xp,y,c = 'red')
        plt.plot(X_new,y_proba[:,1],"g-",label="Malign",)
        plt.plot(X_new,y_proba[:,0],"b--",label="Benign")
        plt.xlabel(f1,labelpad=-45,loc='center')
        plt.ylabel('Probability',loc='center',labelpad=-35)
        plt.legend()
        plt_idx = plt_idx+1
```



After cleaning for high correlated variables, its easy to observe that some features do not have high explanatory power when isolated, as the classes ar not clearly differentiated. We will fit the model now with all the features at the same time.

We will define a function generation a confusion matrix, which has on the diagonal the correct prections to show graphically how well the model fit the data. Afterwards, there is a another function that we create so that the data are split between train and test set, the model is fit on the train set and the the metrics described before are applied to evaluate the model

In [702…

```
from sklearn.metrics import plot_confusion_matrix
import matplotlib as mpl

def plot_cm(clf, X, y, labs):

    mpl.rcParams.update({'font.size': 16})
    cm = plot_confusion_matrix(clf, X, y, display_labels=labs,cmap=mpl.cm.Blues);
```

In [703…

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

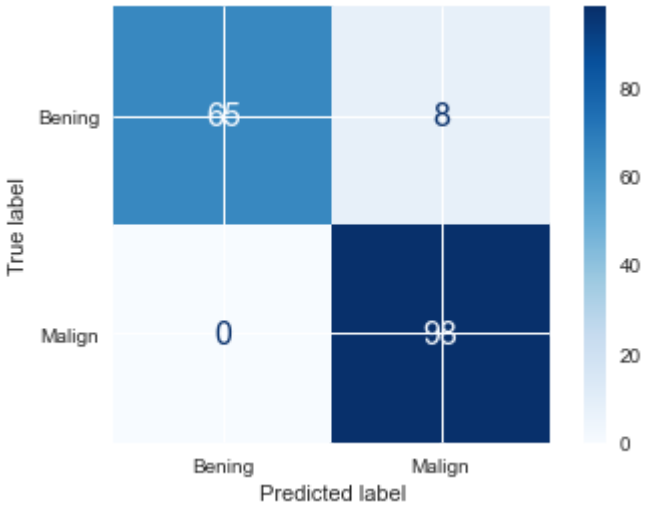
```
def train(model, z, w):
    x_train, x_test, y_train, y_test = train_test_split(z, w, test_size=.30, random_state=999)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)

    print('Model Report')
    print('Accuracy: ', np.around(accuracy_score(y_test, y_pred,normalize=True),4))
    print('Precision: ', np.around(precision_score(y_test, y_pred),4))
    print('Recall: ', np.around(recall_score(y_test, y_pred),4))
    print('f1: ', np.around(f1_score(y_test, y_pred),4))
    plot_cm(model, x_test, y_test, labs=('Bening', 'Malign'))
```

5.1.1 Logistic regression full sample

In [704…

```
xt=x.drop(["types","target"],axis=1)
model = LogisticRegression(solver='liblinear')
train(model, xt, y)
```

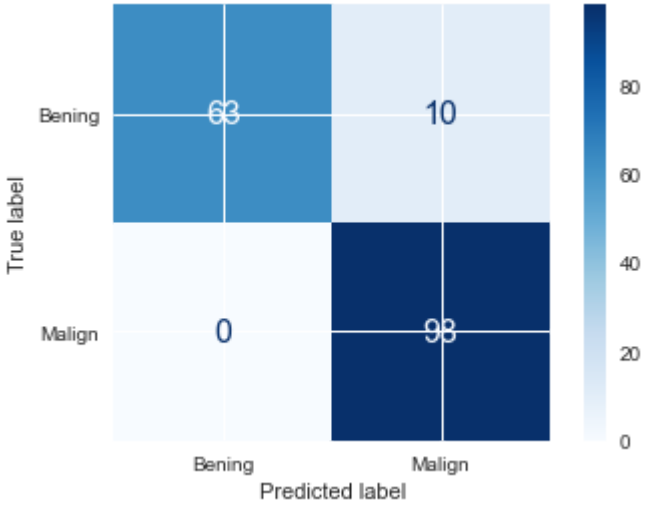
```
Model Report
Accuracy:  0.9532
Precision:  0.9245
Recall:  1.0
f1:  0.9608
```



5.1.2 Logistic regression subsample

In [705…
```
xt=x1.drop(["types","target"],axis=1)
model = LogisticRegression(solver='liblinear')
train(model, xt, y)
```

```
Model Report
Accuracy:  0.9415
Precision:  0.9074
Recall:  1.0
f1:  0.9515
```



The results actually worsen after removing the selected variables, although not by much.

5.2 K-Nearest Neighbor

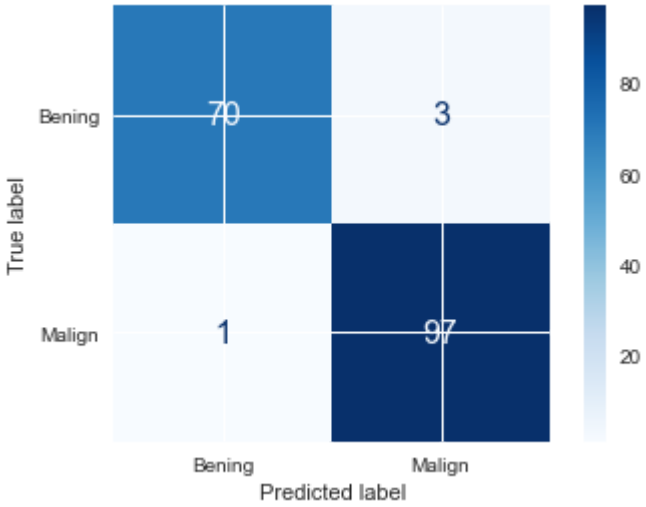
5.2.1 K-Nearest Neighbor full sample

In [706…
```
from sklearn.neighbors import KNeighborsClassifier
```

In [707…
```
K=3
model = KNeighborsClassifier(K,p=2)
xt=x.drop(["types","target"],axis=1)
train(model, xt, y)
```

```
Model Report
Accuracy:  0.9766
```

```
Precision:  0.97
Recall:  0.9898
f1:  0.9798
```

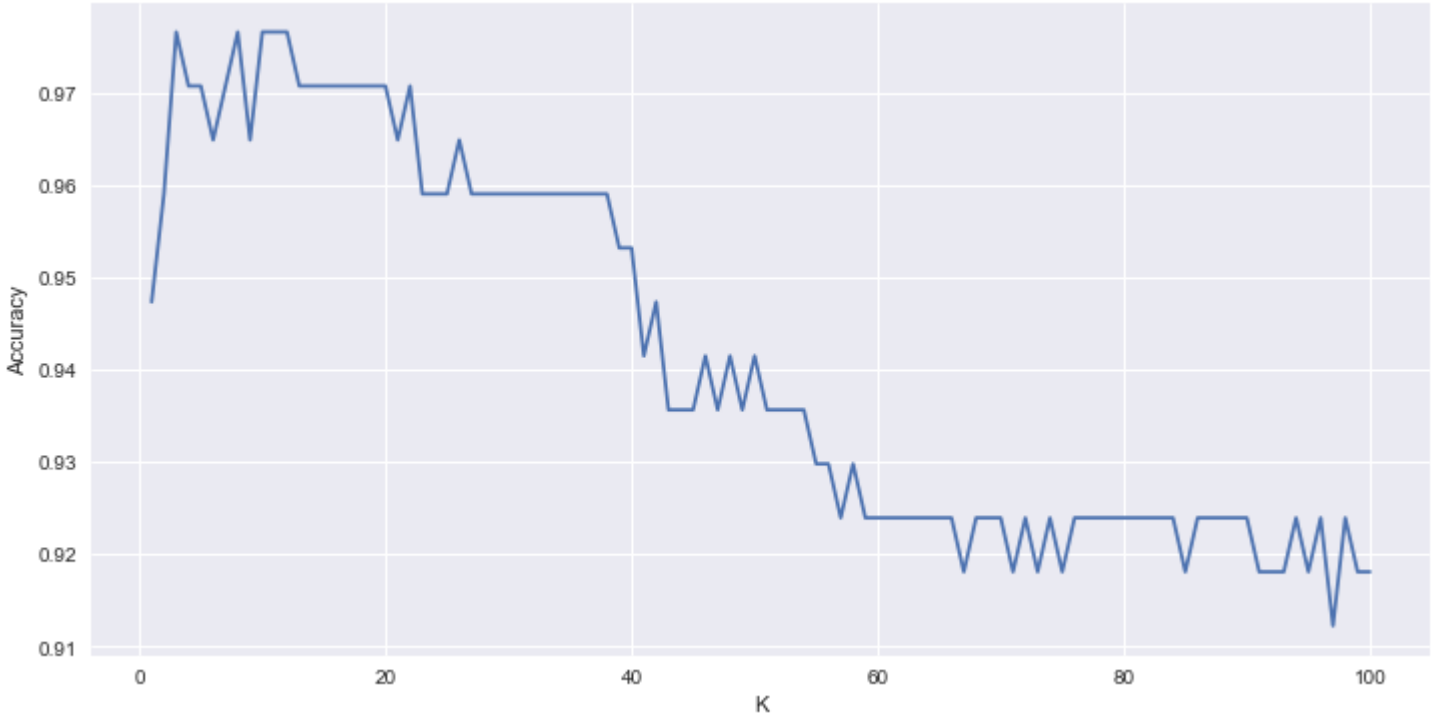


5.2.1.2 K variation analysis

We will see how the accuracy of our prediction changes as we vary the number of neighbors taken to fit the model

In [708…

```
from sklearn.metrics import confusion_matrix
accs = []
ks = []
xt=x.drop(["types","target"],axis=1)
for k in range(1,101,1):
    x_train, x_test, y_train, y_test = train_test_split(xt, y, test_size=0.30, random_state=999)
    knn = KNeighborsClassifier(n_neighbors = k, p=2)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    acc = (cm[0][0]+cm[1][1])/len(y_test)
    ks.append(k)
    accs.append(acc)

plt.figure(figsize=(12, 6))
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.plot(ks,accs);
```



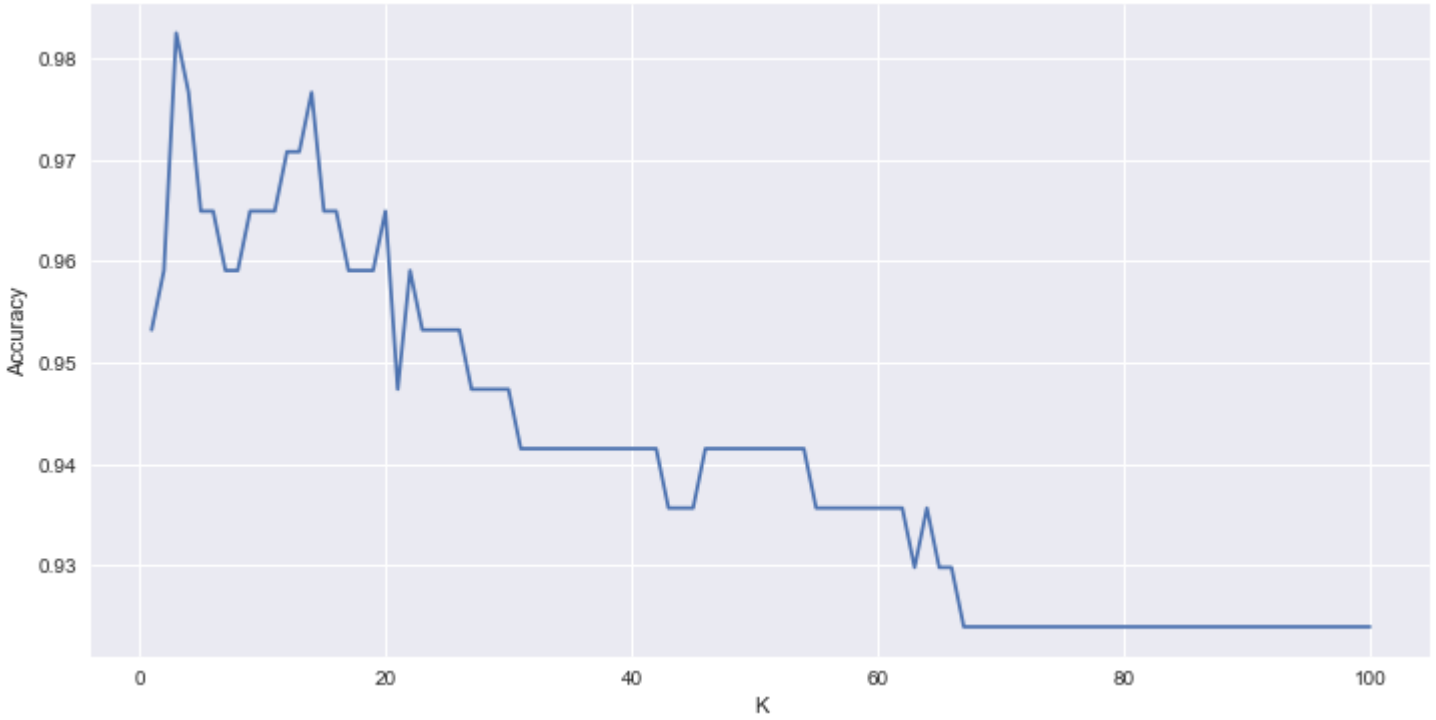
5.2.1.2 Check Analysis

Is easy to see that when K=3 we reach the highest accuracy. We will now see how the graph changes when wee change the method to measure the distance. The first method used above was the Euclidean method (p=2). Now we will use the Manhattan method (p=1). Finally, we decided to use the Manhattan method as it reaches the highest accuracy. Now, we will corroborate that accuracy decays as k increases by fittign the model for different ks

In [709…

```
from sklearn.metrics import confusion_matrix
accs = []
ks = []
xt=x.drop(["types","target"],axis=1)
for k in range(1,101,1):
    x_train, x_test, y_train, y_test = train_test_split(xt, y, test_size=0.30, random_state=999)
    knn = KNeighborsClassifier(n_neighbors = k, p=1)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    acc = (cm[0][0]+cm[1][1])/len(y_test)
    ks.append(k)
```

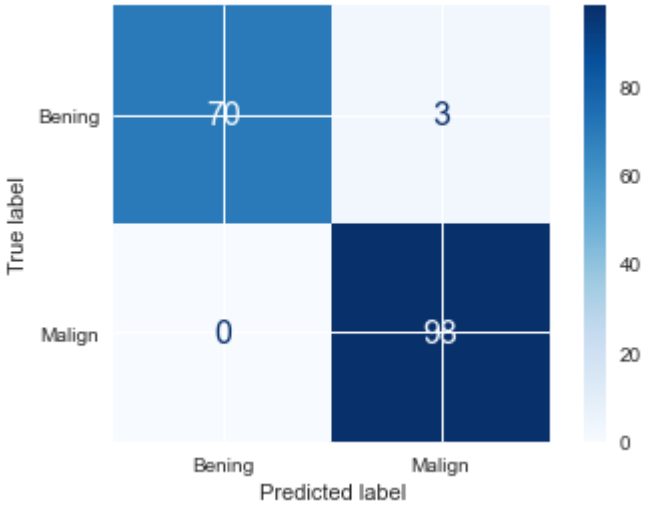
```
    accs.append(acc)

plt.figure(figsize=(12, 6))
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.plot(ks,accs);
```



In [710…

```
K=3
model = KNeighborsClassifier(K,p=1)
xt=x.drop(["types","target"],axis=1)
train(model, xt, y)
```

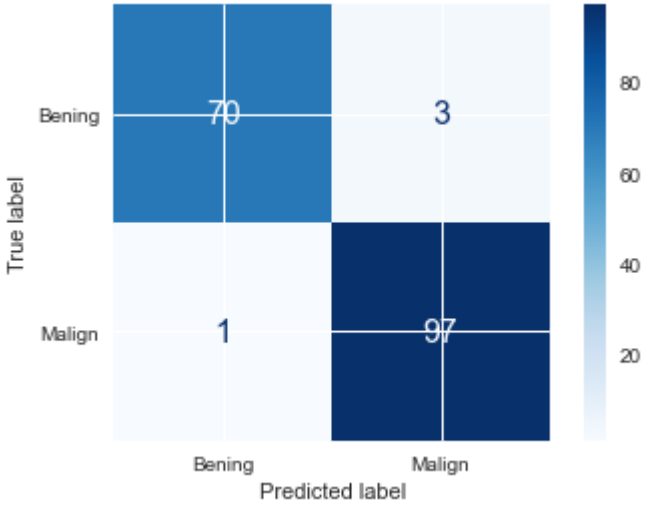
```
Model Report
Accuracy:  0.9825
Precision:  0.9703
Recall:  1.0
f1:  0.9849
```



In [711…

```
K=4
model = KNeighborsClassifier(K,p=1)
xt=x.drop(["types","target"],axis=1)
train(model, xt, y)
```

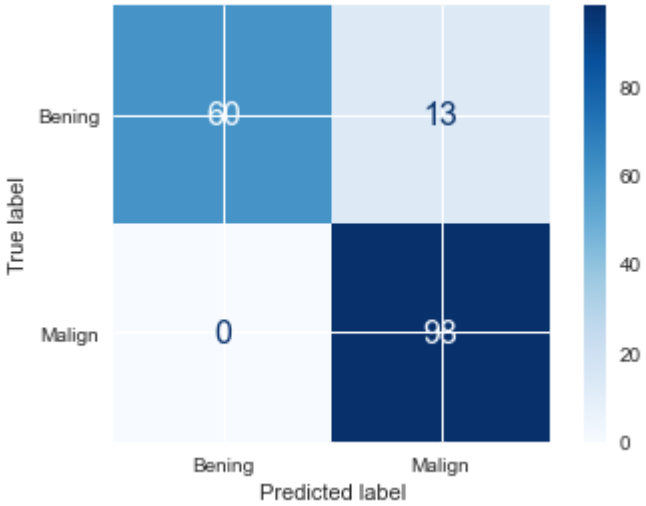
```
Model Report
Accuracy:  0.9766
Precision:  0.97
Recall:  0.9898
f1:  0.9798
```



In [712…

```
K=70
model = KNeighborsClassifier(K,p=1)
xt=x.drop(["types","target"],axis=1)
train(model, xt, y)
```

```
Model Report
Accuracy:  0.924
Precision:  0.8829
Recall:  1.0
f1:  0.9378
```

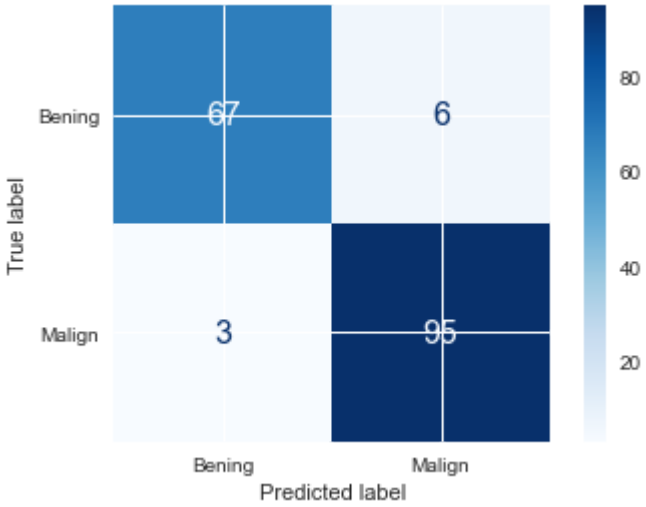


5.2.2 K-Nearest Neighbor subsample

In [713…

```
K=3
model = KNeighborsClassifier(K,p=2)
xt=x1.drop(["types","target"],axis=1)
train(model, xt, y)
```

```
Model Report
Accuracy:  0.9474
Precision:  0.9406
Recall:  0.9694
f1:  0.9548
```

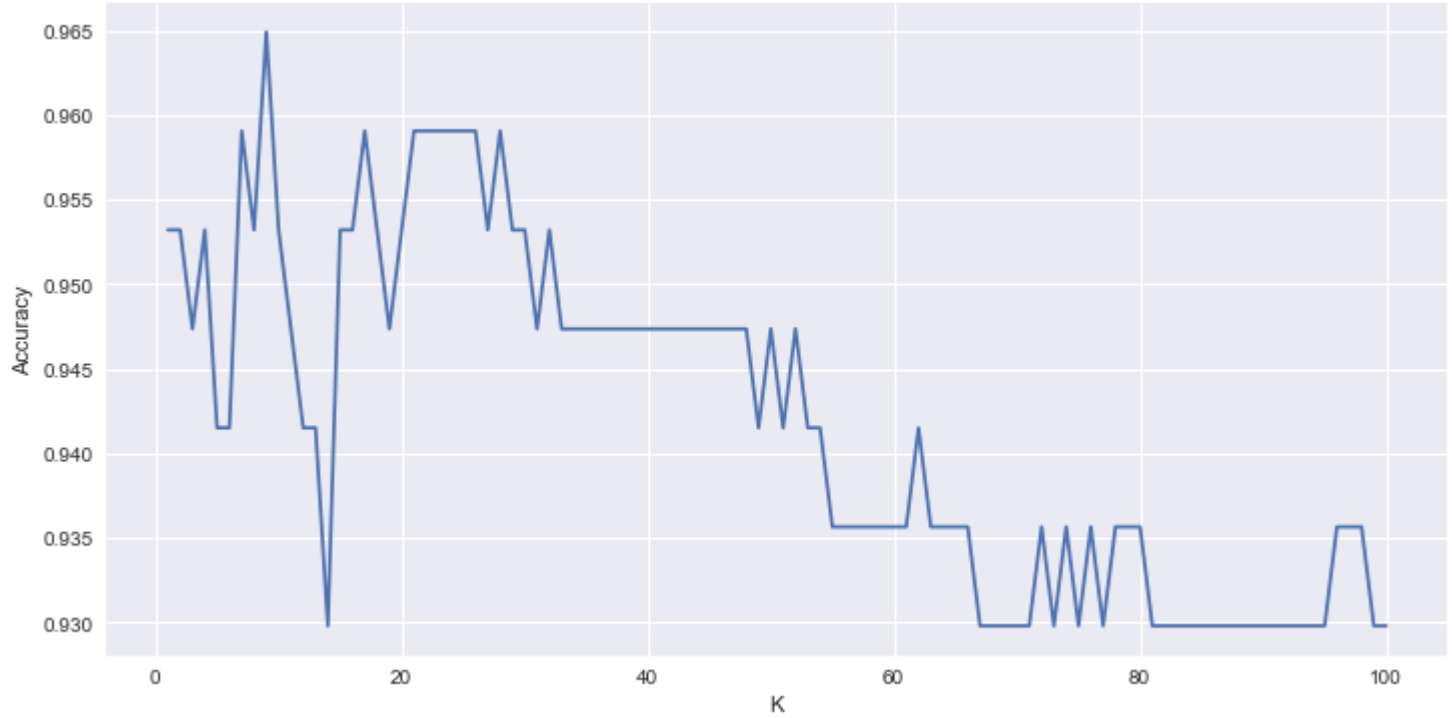


5.2.2.2 K variation analysis

We will see again how the accuracy of our prediction changes as we vary the number of neighbors taken to fit the model

In [714…

```
from sklearn.metrics import confusion_matrix
accs = []
ks = []
xt=x1.drop(["types","target"],axis=1)
for k in range(1,101,1):
    x_train, x_test, y_train, y_test = train_test_split(xt, y, test_size=0.30, random_state=999)
    knn = KNeighborsClassifier(n_neighbors = k, p=2)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    acc = (cm[0][0]+cm[1][1])/len(y_test)
    ks.append(k)
    accs.append(acc)

plt.figure(figsize=(12, 6))
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.plot(ks,accs);
```



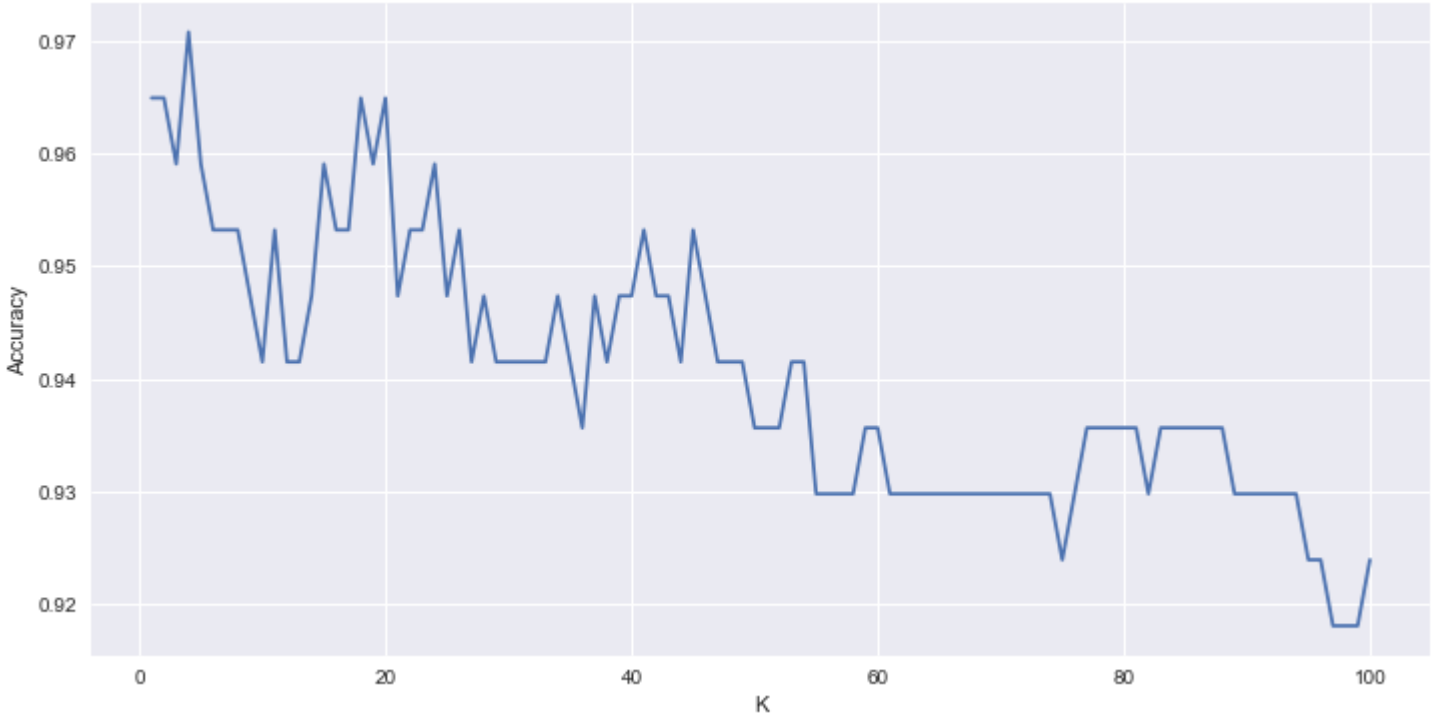
5.2.1.2 Check Analysis

In this case, when K=3 we reach again the highest accuracy. We will repeat the same step before to seehow the graph changes when we change the method to measure the distance. In this case, we decided again to use the Manhattan method as it reaches the highest accuracy. Now, we will corroborate that accuracy decays as k increases by fittign the model for different ks

In [715…

```
from sklearn.metrics import confusion_matrix
accs = []
ks = []
xt=x1.drop(["types","target"],axis=1)
for k in range(1,101,1):
    x_train, x_test, y_train, y_test = train_test_split(xt, y, test_size=0.30, random_state=999)
    knn = KNeighborsClassifier(n_neighbors = k, p=1)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    acc = (cm[0][0]+cm[1][1])/len(y_test)
    ks.append(k)
    accs.append(acc)
```

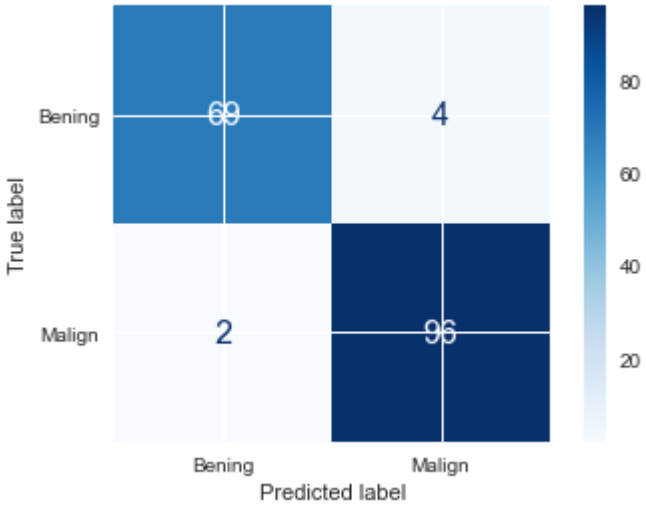
```
plt.figure(figsize=(12, 6))
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.plot(ks,accs);
```



In [716…

```
K=1
model = KNeighborsClassifier(K,p=1)
xt=x1.drop(["types","target"],axis=1)
train(model, xt, y)
```

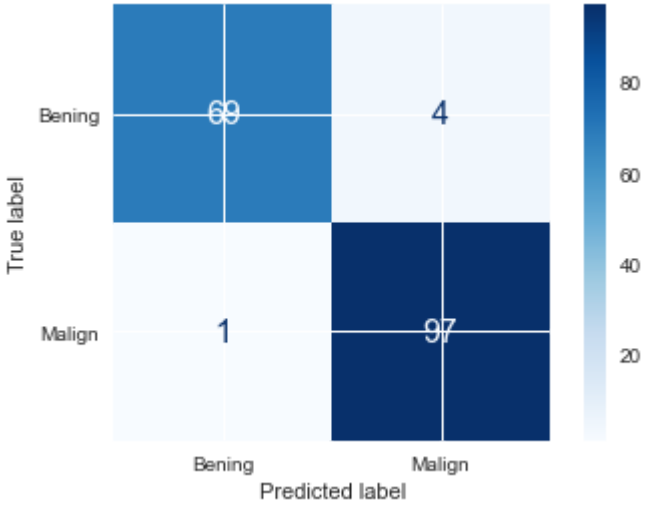
```
Model Report
Accuracy:  0.9649
Precision:  0.96
Recall:  0.9796
f1:  0.9697
```



In [717…

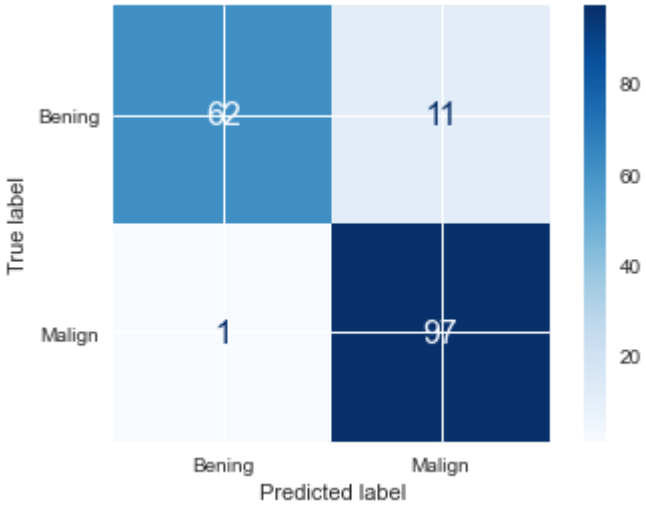
```
K=4
model = KNeighborsClassifier(K,p=1)
xt=x1.drop(["types","target"],axis=1)
train(model, xt, y)
```

```
Model Report
Accuracy:  0.9708
Precision:  0.9604
Recall:  0.9898
f1:  0.9749
```



In [718…
```
K=70
model = KNeighborsClassifier(K,p=1)
xt=x1.drop(["types","target"],axis=1)
train(model, xt, y)
```

```
Model Report
Accuracy:  0.9298
Precision:  0.8981
Recall:  0.9898
f1:  0.9417
```



Similarly as the logistic regression case, the metrics worsen after removing the selected variables but not in a significant way

5.3 Decision Tree

We will now fit the Decision Tree model, setting a maximum depth of ramifications.

In [719…
```
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
```

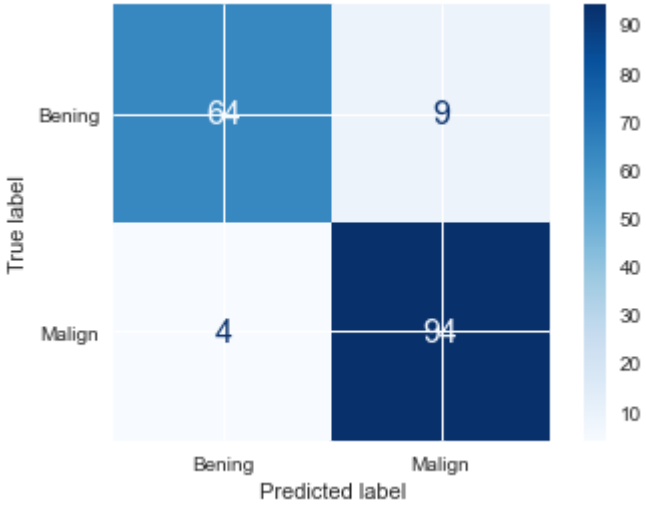
5.3.1 Decision Tree full sample

As we can see, all the final leafs have gini meausres equal to 0 meaning that the model has exactly fitted the data

In [720…

```
model = DecisionTreeClassifier(random_state=235, max_depth=10)
xt=x.drop(["types","target"],axis=1)
train(model, xt, y)
```

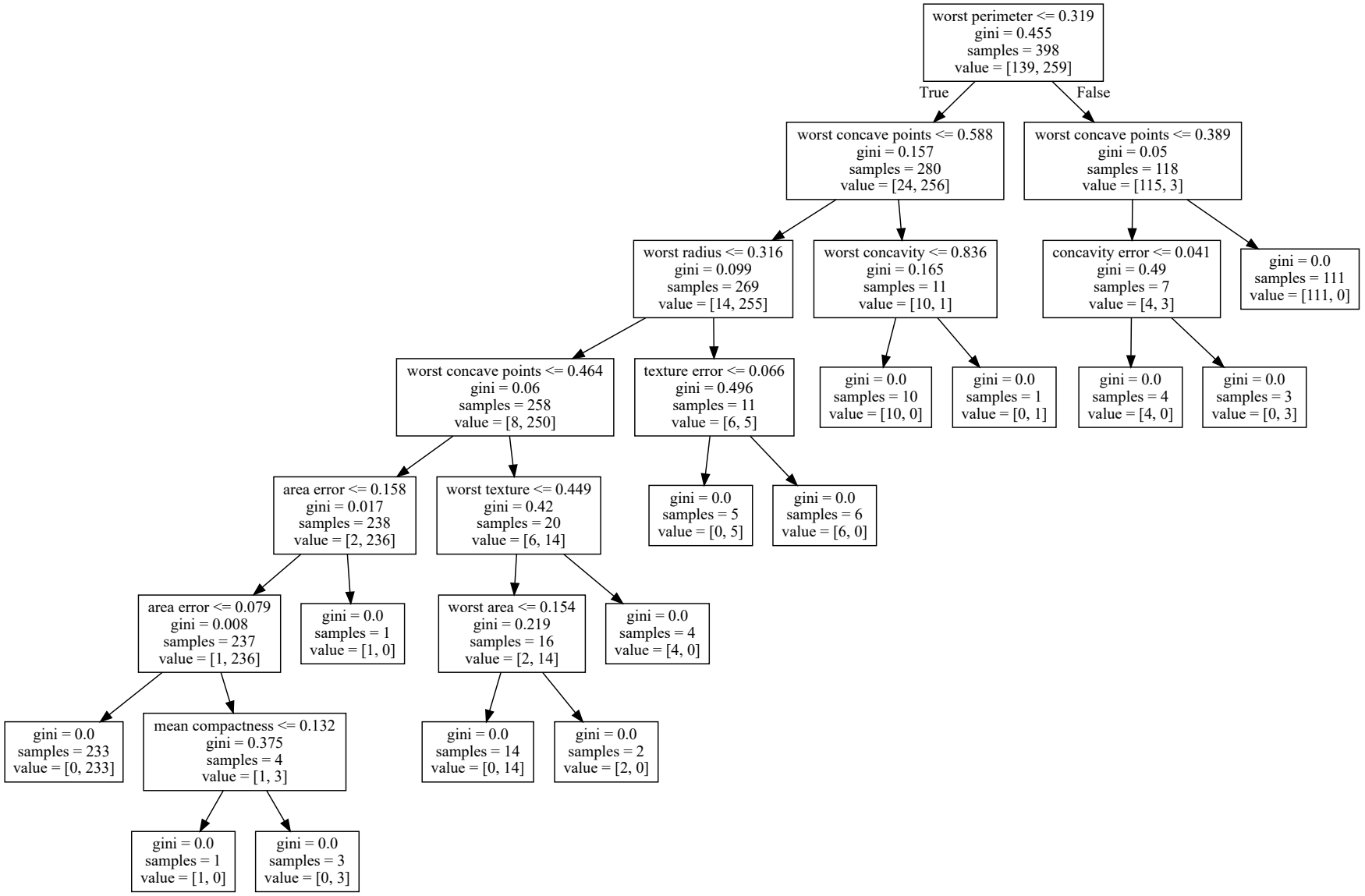
```
Model Report
Accuracy:  0.924
Precision:  0.9126
Recall:  0.9592
f1:  0.9353
```



In [721…

```
from sklearn.tree import export_graphviz
from os import system
from graphviz import Source
from IPython.display import SVG
graph = Source(export_graphviz(model, out_file=None, feature_names = xt.columns))
SVG(graph.pipe(format='svg'))
```

Out[721…



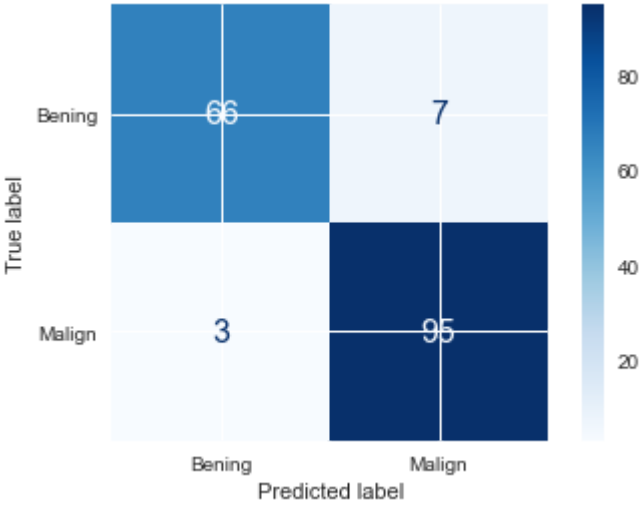
5.3.2 Decision Tree subsample

In this case, the gini metrics are all equal to zero in the final leafs, so the model fitted the data exacly again. In this case, the subsample actually allowed for an improvement in the perfomance measures, although they are still lower that the ones obtained in the previos two methods

In [722…

```
model = DecisionTreeClassifier(random_state=235, max_depth=10)
xt=x1.drop(["types","target"],axis=1)
train(model, xt, y)
```

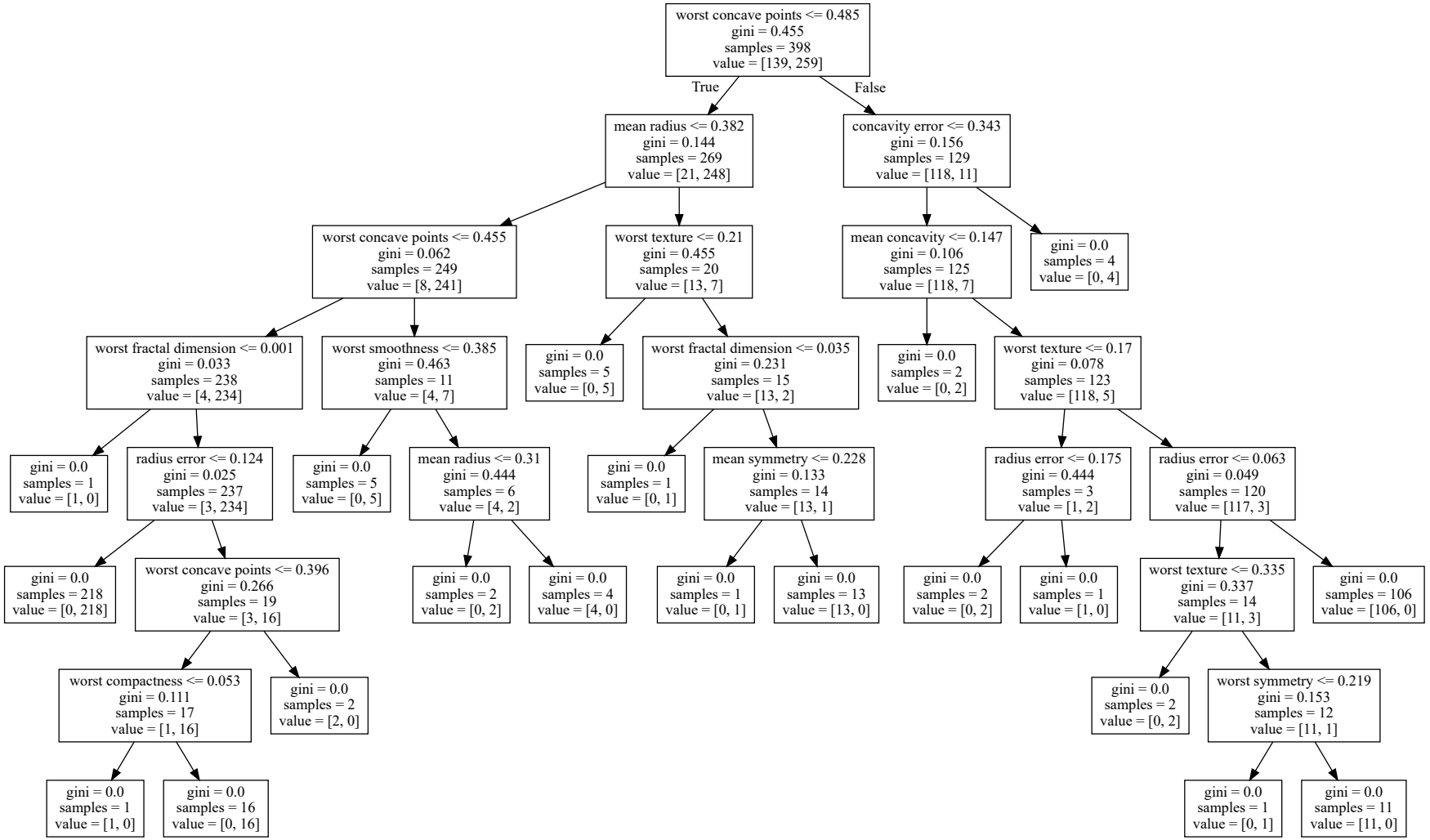
```
Model Report
Accuracy:  0.9415
Precision:  0.9314
Recall:  0.9694
f1:  0.95
```



In [723…

```
from sklearn.tree import export_graphviz
from os import system
from graphviz import Source
from IPython.display import SVG
graph = Source(export_graphviz(model, out_file=None, feature_names = xt.columns))
SVG(graph.pipe(format='svg'))
```

Out[723…



5.4 Support Vector Machines

In [724…

```
from sklearn.svm import SVC
```

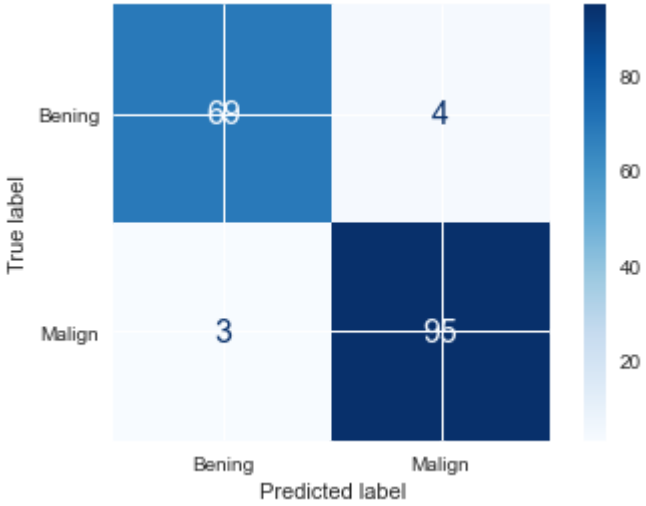
5.4.1 Support Vector Machines full sample

We will fit the model using a linear, radial basis and polynomial kernels (mapping methods)

In [725…

```
model = SVC(kernel = 'linear', C=1000)
xt=x.drop(["types","target"],axis=1)
train(model, xt, y)
```

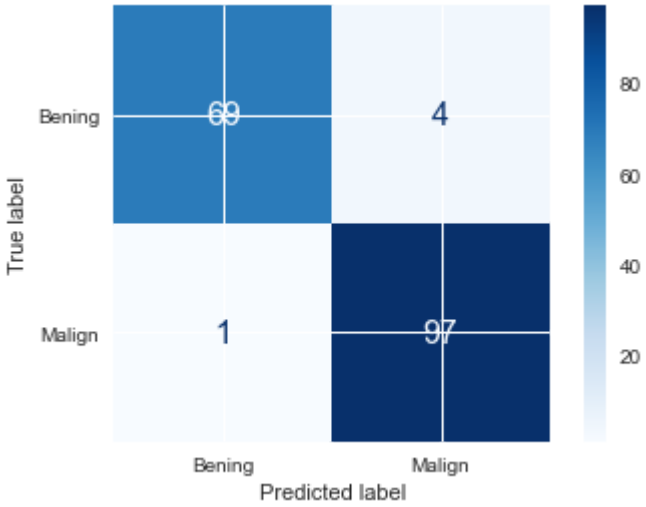
```
Model Report
Accuracy:  0.9591
Precision:  0.9596
Recall:  0.9694
f1:  0.9645
```



In [726…

```
model = SVC(kernel = 'rbf', C=1000)
xt=x.drop(["types","target"],axis=1)
train(model, xt, y)
```

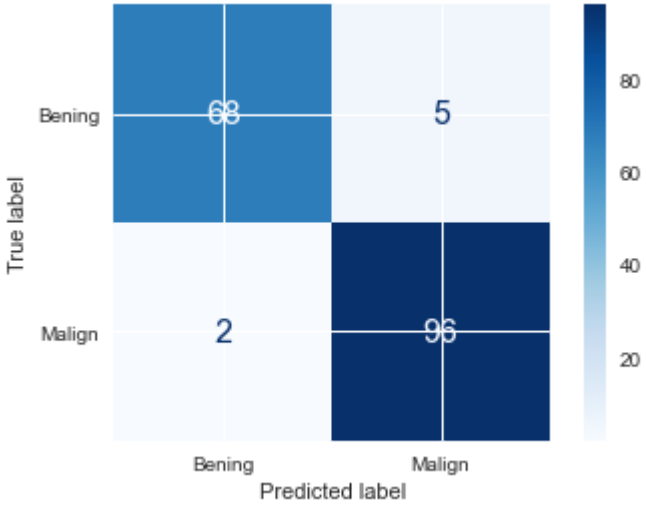
```
Model Report
Accuracy:  0.9708
Precision:  0.9604
Recall:  0.9898
f1:  0.9749
```



In [727…

```
model = SVC(kernel = 'poly', C=1000)
xt=x.drop(["types","target"],axis=1)
train(model, xt, y)
```

```
Model Report
Accuracy:  0.9591
Precision:  0.9505
Recall:  0.9796
f1:  0.9648
```



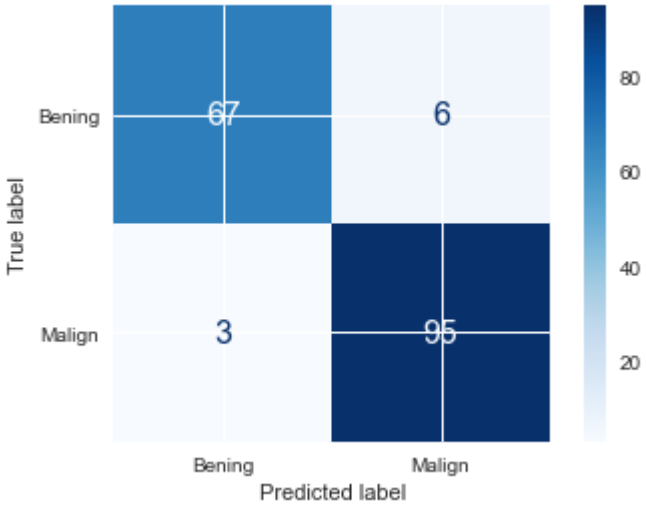
The radial basis kernel yields the best metrics. Let us explore what happens when we dropped the selected variables

5.4.2 Support Vector Machines subsample

In [728…

```
model = SVC(kernel = 'linear', C=1000)
xt=x1.drop(["types","target"],axis=1)
train(model, xt, y)
```

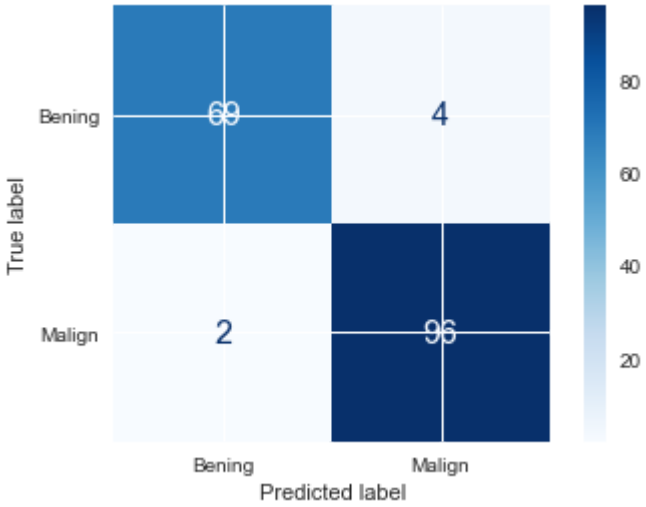
```
Model Report
Accuracy:  0.9474
Precision:  0.9406
Recall:  0.9694
f1:  0.9548
```



In [729…

```
model = SVC(kernel = 'rbf', C=1000)
xt=x1.drop(["types","target"],axis=1)
train(model, xt, y)
```

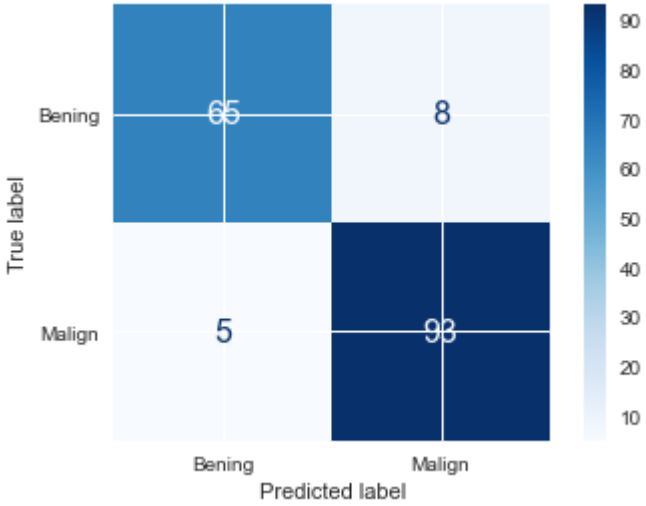
```
Model Report
Accuracy:  0.9649
Precision:  0.96
Recall:  0.9796
f1:  0.9697
```



In [730…

```
model = SVC(kernel = 'poly', C=1000)
xt=x1.drop(["types","target"],axis=1)
train(model, xt, y)
```

```
Model Report
Accuracy:  0.924
Precision:  0.9208
Recall:  0.949
f1:  0.9347
```



We obtained the same results, the radial basis kernel yields the best result. Also, the analysis is consistent with the logistic and the K-neares

neighvor methods, as dropping the selected variables actually worsen the performance

6.- Identify the best method

For selecting the best method, we compily in one talbe the performance metrics for the best case in each method. The best method was the K-Nearest Neighbor with the Manhattan method and K=3 (full sample). The next better ones were the Logistic Regression and the Support Vector Machines with very little difference between them (full sample). Is interesting to see that the worst method was the one that actually performed better with the subsample.

In [731…

```
L = {'Logistic Regression':[0.9532,0.9245, 1.000, 0.9608], 'KNN':[0.9825,0.9703, 1.0000, 0.9849], 'Decision Tree':[0.9240

df_results = pd.DataFrame(L, index=['Accuracy', 'Precision', 'Recall', 'F1'])

df_results
```

Out[731…

	Logistic Regression	KNN	Decision Tree	SVM
Accuracy	0.9532	0.9825	0.9240	0.9708
Precision	0.9245	0.9703	0.9126	0.9604
Recall	1.0000	1.0000	0.9592	0.9898
F1	0.9608	0.9849	0.9353	0.9749

7.- Variable significance analysis

We will perform the next analysis with the best model from the table above. We will remove one variable at a time and see how the performance measure of the model changes

In [732…

```
results = pd.DataFrame(index=['Accuracy', 'Precision', 'Recall', 'F1'])
def train_2(model, x, y, name='Feature'):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.30, random_state=999)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)

    stats = [np.around(accuracy_score(y_test, y_pred),4),np.around(precision_score(y_test, y_pred),4),
```

```
        np.around(recall_score(y_test, y_pred),4), np.around(f1_score(y_test, y_pred),4)]
    results[name] = stats
```

In [733…

```
K=3
model = KNeighborsClassifier(K,p=1)
xt=x.drop(["types","target"],axis=1)
i = 0
for i in range(0, len(data.feature_names)):
    train_2(model, xt.drop(data.feature_names[i],axis=1), y, name = data.feature_names[i])
    i = i + 1

results.sort_values(by="Accuracy",axis=1)
```

Out[733…

	mean texture	worst texture	concave points error	mean radius	worst concavity	worst compactness	worst perimeter	worst radius	concavity error	compactness error	...	fractal dimension error	mean concave points	symi
Accuracy	0.9708	0.9708	0.9708	0.9766	0.9766	0.9766	0.9766	0.9766	0.9766	0.9766	...	0.9825	0.9825	(
Precision	0.9604	0.9697	0.9604	0.9700	0.9608	0.9608	0.9700	0.9700	0.9608	0.9700	...	0.9703	0.9703	(
Recall	0.9898	0.9796	0.9898	0.9898	1.0000	1.0000	0.9898	0.9898	1.0000	0.9898	...	1.0000	1.0000	1
F1	0.9749	0.9746	0.9749	0.9798	0.9800	0.9800	0.9798	0.9798	0.9800	0.9798	...	0.9849	0.9849	(

4 rows × 30 columns

The less significant measures are "mean symmetry", "worst area", "mean concave points", "fractal dimension error", "mean compactness", "mean smoothness", "symmetry error", "worst concave points", "mean fractal dimension" and "worst smoothness" as the performance measures are the highest when removing them. The name of the column in the table above indictes which feature has been removed. We will fit the model again now removing those variables

In [734…

```
results=[]
results = pd.DataFrame(index=['Accuracy', 'Precision', 'Recall', 'F1'])
K=4
model = KNeighborsClassifier(K,p=1)
xt=x.drop(["types","target","mean symmetry", "worst area", "mean concave points", "fractal dimension error","mean compact
names=list(xt.columns)
i = 0
for i in range(0, len(names)):
```

```
    train_2(model, xt.drop(names[i],axis=1), y, name = names[i])
    i = i + 1

results.sort_values(by="Accuracy",axis=1)
```

Out[734...

	worst texture	worst symmetry	worst fractal dimension	mean texture	mean concavity	radius error	perimeter error	concave points error	compactness error	worst concavity	worst compactness	smoothness error	c
Accuracy	0.9591	0.9649	0.9708	0.9708	0.9708	0.9708	0.9708	0.9708	0.9708	0.9766	0.9766	0.9766	
Precision	0.9596	0.9792	0.9794	0.9794	0.9697	0.9697	0.9697	0.9794	0.9794	0.9796	0.9796	0.9796	
Recall	0.9694	0.9592	0.9694	0.9694	0.9796	0.9796	0.9796	0.9694	0.9694	0.9796	0.9796	0.9796	
F1	0.9645	0.9691	0.9744	0.9744	0.9746	0.9746	0.9746	0.9744	0.9744	0.9796	0.9796	0.9796	

Is interesting how if we remove "Worst Perimeter", we get a better performance measures that without removing any variable

In [735...

```
results=[]
results = pd.DataFrame(index=['Accuracy', 'Precision', 'Recall', 'F1'])
K=4
model = KNeighborsClassifier(K,p=1)
xt=x.drop(["types","target","mean symmetry", "worst area", "mean concave points", "fractal dimension error","mean compact
names=list(xt.columns)
i = 0
for i in range(0, len(names)):
    train_2(model, xt.drop(names[i],axis=1), y, name = names[i])
    i = i + 1

results.sort_values(by="Accuracy",axis=1)
```

Out[735...

	worst texture	worst symmetry	mean texture	worst fractal dimension	worst compactness	worst concavity	mean concavity	radius error	perimeter error	area error	compactness error	concavity error	wors radiu
Accuracy	0.9649	0.9649	0.9708	0.9766	0.9766	0.9825	0.9825	0.9825	0.9825	0.9825	0.9825	0.9825	0.988
Precision	0.9600	0.9694	0.9794	0.9796	0.9796	0.9703	0.9798	0.9703	0.9703	0.9703	0.9798	0.9798	0.980
Recall	0.9796	0.9694	0.9694	0.9796	0.9796	1.0000	0.9898	1.0000	1.0000	1.0000	0.9898	0.9898	1.000
F1	0.9697	0.9694	0.9744	0.9796	0.9796	0.9849	0.9848	0.9849	0.9849	0.9849	0.9848	0.9848	0.989

Finally, we will repeat the analysis by removing the variables we selected from the correlation analysis

In [736…

```
results=[]
results = pd.DataFrame(index=['Accuracy', 'Precision', 'Recall', 'F1'])
K=3
model = KNeighborsClassifier(K,p=1)
xt=x1.drop(["types","target"],axis=1)
names=list(xt.columns)
i = 0
for i in range(0, len(names)):
    train_2(model, xt.drop(names[i],axis=1), y, name = names[i])
    i = i + 1

results.sort_values(by="Accuracy",axis=1)
```

Out[736…

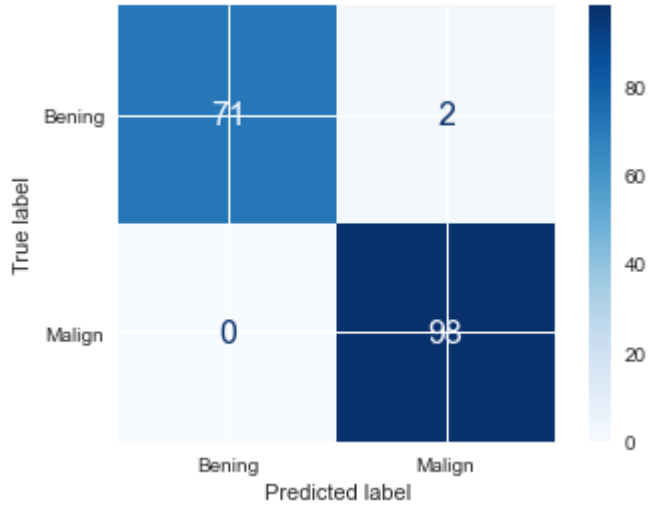
	mean radius	worst smoothness	mean smoothness	mean symmetry	worst texture	worst concavity	worst concave points	mean compactness	mean concavity	compactness error	worst symmetry	wor fract dimensi
Accuracy	0.9474	0.9474	0.9532	0.9532	0.9532	0.9532	0.9532	0.9591	0.9591	0.9591	0.9591	0.95
Precision	0.9320	0.9238	0.9327	0.9327	0.9412	0.9327	0.9327	0.9417	0.9417	0.9417	0.9417	0.94
Recall	0.9796	0.9898	0.9898	0.9898	0.9796	0.9898	0.9898	0.9898	0.9898	0.9898	0.9898	0.98
F1	0.9552	0.9557	0.9604	0.9604	0.9600	0.9604	0.9604	0.9652	0.9652	0.9652	0.9652	0.96

The results are better when we remove the variables selected in the last feature selection analysis

In [737…

```
K=4
model = KNeighborsClassifier(K,p=1)
xt=x.drop(["types","target","mean symmetry", "worst area", "mean concave points", "fractal dimension error","mean compact
train(model, xt, y)
```

```
Model Report
Accuracy:  0.9883
Precision:  0.98
Recall:  1.0
f1:  0.9899
```



8.- Conclusion

Even though, all classifiers methods actually performed very well, the K-Nearest Neighbor seems to be the most effective. Nevertheless, given thi high number of variables, the model may have been overfitted. We discard this possibility as we removed several features at a time and the explanatory power remained quite high as reflected by the performance measures yielded when fitting the model on the subsample data. We can conclude that some features, when highly correlated to other variables, do not add much to our classifiers methods, so the most efficient choice is to delete them. Finally, given the performance measures obtained we could rely without doubt into our classifiers to do predicitons regarding the Breast Cancer. A larger sampel would be helpful to have more solid basis.

In [ ]: