P1:

A R package named "impute" is used for imputation of missing data point. Using the summary function in R, it is clear that the column with missing information are work class, occupation and native country.

The function in R is given as follow:

```
knn.impute(data, k = N, cat.var = 1:ncol(data), to.impute = 1:nrow(data),
using = 1:nrow(data))
```

whose argument is listed as follow:

data
a data frame.
k
number of neighbors to be used; for categorical variables the mode of the neighbors is used, for
continuous variables the median value is used instead.
cat.var
vector containing the indices of the variables to be considered as categorical. Default: all variables.
to.impute
vector indicating which rows of the dataset are to be imputed. Default: impute all rows.
using
vector indicating which rows of the dataset are to be used to search for neighbours. Default: use all rows.

.

The given data set is imported as data frame in R. Variables are split into 3 categories – categorical, ranked
categorical, continuous for preprocessing:

Category 1: work class, education, marital status, occupation, relationship, race, native country.
Category 2: education-num
Category3: age, fnlwgt, capital-gain, capital-loss, hours per week.

Some preprocessing to the data is made using R, which are designed to achieve following result:

For variables in category 1 are treated as purely categorical, strings are converted to numerical values for convenience, but the distance is computed as 0(if same) and1(if different), which is done by setting columns as categorical in R using "cat.var" argument in the function above.

For variables in category 2 and 3, values are treated as continuous and are normalized to (0,1) for the to make sure each variable weighs the same when computing KNN. Then Euclidean Distance is used for computation (which is the default of the impute package in R for continuous variable, so no specification has to be made).

Then, setting k=5, specify the variables in category 1 as categorical using argument "cat.var" to impute both training and test data. The missing value is imputed using the mode of its 5 closet neighbors, only rows with no missing value is used in imputation.

Note: The default of the given R function will impute missing result of continuous variable as mean of its n-nearest neighbors. Given in the data set all missing entries are categorical variables, it should not be a concern here.

P2:

The cleaned data in is loaded and processed in R.

First, try to train the model with ID3 with no limitation on maximum depth and no pruning by simple recursion, R package named "data.tree" is used for implementation. Since ID3 cannot handle continuous variables in nature, all continuous variables are manually split into 4 categories by using mean, upper & lower quartiles and treated as categorical variables.

The pseudo code of ID3 algorithm is given as follow:

> if
> the data-set is pure (e.g. all >50k) then
> construct a leaf having the name of the class (e.g. '')
>
> else
> - choose the feature with the highest information gain (e.g. 'race')
> - for each value of that feature (e.g. 'Asian', 'white', 'black'):
> 1. take the subset of the data-set having that feature value
> 2. construct a child node having the name of that feature value (e.g. 'black')
> 3. call the algorithm recursively on the child node and the subset

Using the implementation above, it is predictable that the model would result in serious overfitting as absolutely no pruning is performed. The simple algorithm gives 64.4% accuracy* (with 35.6% classification error) for the testing dataset. Given simple guess would give 50% accuracy, the result is not that satisfactory.

A feasible way to reduce overfitting is doing some pre-examination on the data using statistical tools. After examining the correlation between variables using scattered plot, we can find that "education-num" and "education" are highly correlated. Also, "relationship" and "marital status" are also highly correlated. The correlation is reasonable given the nature of those predictors. "education" and "marital status" are removed from the set of features that could be selected for simplification. Also, given that native country has more than 97% "United States", and the training data has one more category compare to test data, "native country" is removed from selectable features for the decision tree.

Then, a R package "tree" which implemented a CART decision tree algorithm, which is better considering we are treating continuous variables here, by default are used to build the decision tree. The description of the algorithm implemented in the packages are given as follow:

> A tree is grown by binary recursive partitioning using the response in the specified formula and choosing splits from the terms of the right-hand-side. Numeric variables are divided into X < a and X > a; the levels of an unordered factor are divided into two non-empty groups. The split which maximizes the reduction in impurity is chosen, the data set split and the process repeated. Splitting continues until the terminal nodes are too small or too few to be split.

The command using to fit the tree are as follow:

```
census_tree<-rpart(over50k ~ ., data=training, method="class")
 parms = list(split = "information"))
```

The code above specify that the training data are used to build a decision tree for the purpose of classification using the algorithm stated above, the impurity measure chosen is information gain.

In order to reduce overfitting, both pre-pruning and post pruning are adopted. The maximum depth is limited to 9 for attempt to pre-pruning the tree. The result model accuracy is 84.4%.

Post pruning are performed using the prune functions provided in the package:

```
prune(fit, cp= )
```

Where cp is the complexity parameter. The Tree is pruned using the complexity parameter which minimize the cross-validation error within the training set(the cross-validation by default choosing 10% data as testing each time), which can be checked by printcp() function provided in the package. After testing, the optimized pruned tree has a predicting accuracy of 85.7% on the testing dataset.

** Note: All accuracy mentioned above refers to 1 - false classification rate .*

P3:

The cleaned data in is loaded and processed in R, the library "randomforest" is used to train the model.

The pseudo code for the randomforest package and the model is given as follows:

> Learn a decision-tree using a modified decision-tree learning algorithm, which is given as follow: at each node of the tree, instead of examining all possible feature-splits, we randomly select some subset of the features $f \subseteq F$, where F is the set of features. The node then splits on the best feature in f rather than F.
>
> For each tree in the forest, instead of using whole data set, we select a bootstrap sample from S where S (i) denotes the ith bootstrap.

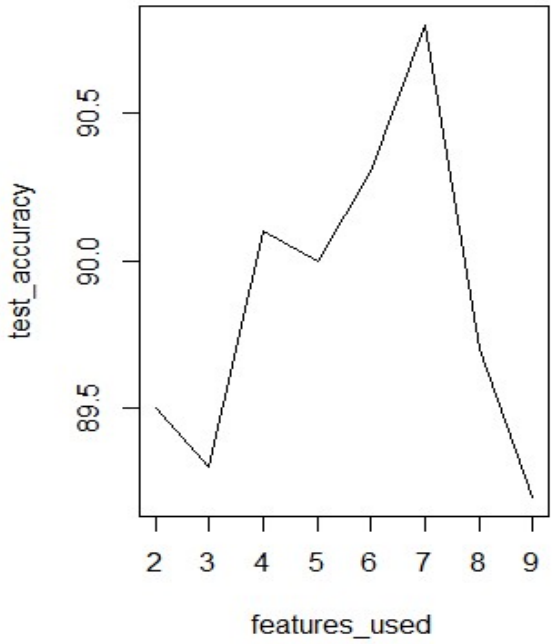
There are two boostrapping in the algorithm, selecting random features from all features at each split and select random boostrap sample from all samples for each tree. For the random forest function, the major parameter to be tuned including: the number of features selected each time split the tree (f), the number of instance to be selected for each tree (S(i)), and the number of trees to grow. While the last one is not actually a boostrapping factor, for each random forest the value optimizing the test accuracy is used.

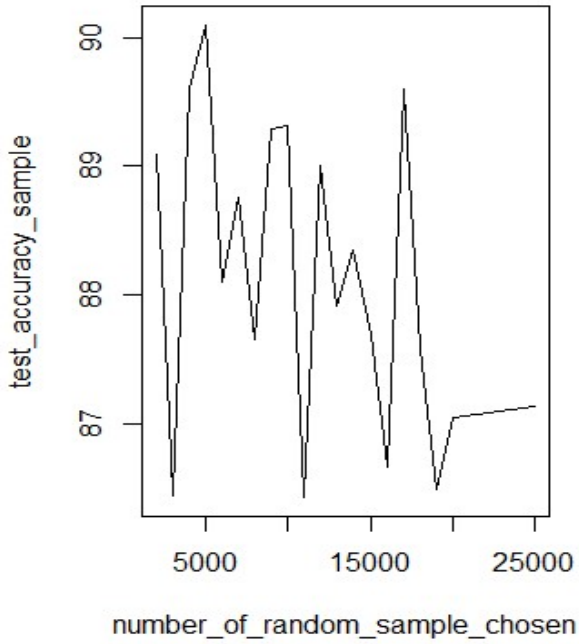
First, keeping the number of instances selected constant (sample size $= 5000$ for this purpose), consider the boostraping of features only. Try to fit the random forest model with different number of features selected randomly for each split. Note that the default parameter of number of feature selected each time is sqrt(p), where p is total number of features available, which is 11 in the given case (for consistency, 3 features removed for the decision tree models are also removed here). Using minimum 2 features, maximum 9 features gives test accuracy from 89.2 to 90.8, where the highest accuracy is resulted from using 7 features, and lowest from using 9 features. And the middle parameters seem to give better result. Possible theory behind this could be that while increasing features selected each time generally improves the performance of the model as at each node now we have a higher number of options to be considered, it decreases the diversity of individual tree and makes them correlated.

Then, keeping the number of features randomly selected each time when splitting the tree constant ($f = 7$ for this purpose). Try to fit a random forest model with different number of randomly selected sample. Different parameters give testing accuracy ranging from 86.4 to 90.1, yet there's no specific trend of choosing larger/smaller proportion of sample would give better result.

The plot for number of feature and instance selected against test accuracy are attached on next page.

Using the same decision tree algorithm, the optimized random forest gives a 90.5% testing accuracy while the decision tree gives a 85.7% accuracy. Although the random forest increases the accuracy, there's not a significant improvement as decision tree has already provide reasonable modeling. The advantage of random forest could possibly be more clearly demonstrated using dataset with more features but fewer instances.


test_accuracy
89.5
90.0
90.5
2
3
4
5
6
7
8
9
features_used


test_accuracy_sample
87
88
89
90
5000
15000
25000
number_of_random_sample_chosen

P4:

The cleaned data is imported and processed using R. A R package named "e1071" designed for SVM is used to train the model. All model is directly fitted using the svm()function, corresponding kernel type and degree for polynomial kernel is specified.

Using summary function n R, the number of supported vector is printed directly, the prediction accuracy is computed from confusion matrix generated from table() and predict() function.

The following table summarizes the fitted model with their # of supported vectors and prediction accuracy.

Kernel Type	RBF	Polynomial `(2)	Polynomial (3)	Polynomial (4)	Polynomial (5)	Linear
# of Support Vectors	10550	11605	13354	14207	14393	10600
Prediction Accuracy	84.7%	84.3%	82.1%	77.8%	77.3%	84.6%

*the higher degree polynomial kernel is not included here as the prediction accuracy drops further as degree increases.

The linear model, although not required in this question, provide good prediction accuracy. The RBF gives the best prediction accuracy here, and the prediction accuracy of polynomial kernel decreases with degree increases. An interesting trend to observe is that the number of supported vectors and prediction accuracy seems to be inversely related, yet more modeled shall be trained to support this assumption. Another trend shown in confusion matrix but not here is that model with more support vector tends to produce small portion of false positive (predict > 50k, actual <=50k), but model with less support vector tends to produce false positive and false negative more evenly.

The coefficient of RBF in descending order are plotted here.